

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Báo cáo đồ án

Đề tài: Image Processing

Môn học: Toán ứng dụng và thống kê cho Công nghệ thông tin

Sinh viên thực hiện:

22127098 - Đinh Vũ Gia Hân

Giáo viên hướng dẫn:

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	1
2	Giới Thiệu	2
3	Thông tin sinh viên	3
4	Ý tưởng thực hiện và mô tả các hàm chức năng:	3
4.1	Giới thiệu chung về đồ án	3
4.2	Các thư viện cần dùng	3
4.3	Mô tả các hàm chức năng	4
4.3.1	def read_img(img_path)	4
4.3.2	def show_img(result_img, func)	4
4.3.3	def save_img(input_file, result_img, func)	4
4.3.4	def main()	5
4.3.5	def adjust_brightness(img, brightness)	5
4.3.6	def adjust_contrast(img, contrast)	6
4.3.7	def flip_horizontal(img)	7
4.3.8	def flip_vertical(img)	8
4.3.9	def RGB_to_gray(img)	9
4.3.10	def RGB_to_sepia(img)	10
4.3.11	def blur_image(img)	11
4.3.12	def sharpen_image(img)	14
4.3.13	def crop_image(img, crop_height, crop_width)	15
4.3.14	def circle_frame(img)	15
4.3.15	def ellipse_frame(img)	17
4.3.16	def zoom_in(img, factor)	21
4.3.17	def zoom_out(img, factor)	21
5	Đánh giá mức độ hoàn thành	22

Danh sách bảng

1	Bảng đánh giá mức độ hoàn thành và hình ảnh kết quả cho từng chức năng	22
2	Bảng đánh giá mức độ hoàn thành và hình ảnh kết quả cho từng chức năng	23

Danh sách hình vẽ

1	Ví dụ về phép quay trục	17
---	-----------------------------------	----

1 Lời cảm ơn

Để hoàn thành được đồ án này, em xin chân thành cảm ơn quý thầy, cô đã tạo cơ hội cho em được học tập, nghiên cứu và tích lũy kiến thức để thực hiện đồ án này. Trên hết, em xin được gửi lời cảm ơn chân thành nhất đến giảng viên Phan Thị Phương Uyên đã tận tình chỉ dẫn và đưa ra nhiều lời khuyên bổ ích giúp em hoàn thành đồ án này một cách tốt nhất.

Em cũng xin gửi lời tri ân tới những nhà phát triển của các thư viện Numpy, Pillow và Matplotlib. Nhờ có tài liệu hướng dẫn chi tiết trên trang web của họ, em đã dễ dàng làm quen và ứng dụng các hàm cần thiết cho đồ án này.

Đặc biệt, em xin gửi lời cảm ơn chân thành đến ChatGPT, mô hình ngôn ngữ lớn, vì đã hỗ trợ em trong việc thu thập và tóm tắt thông tin quan trọng liên quan đến việc thực hiện đồ án Image Processing.

Do kiến thức của bản thân còn nhiều hạn chế và thiếu kinh nghiệm thực tiễn nên nội dung báo cáo khó tránh khỏi những thiếu sót. Em rất mong nhận được những lời góp ý thêm từ quý thầy cô.

Trân trọng.

2 Giới Thiệu

Xử lý ảnh (Image Processing) là một lĩnh vực quan trọng trong công nghệ thông tin và khoa học máy tính, với ứng dụng rộng rãi trong nhiều lĩnh vực như y tế, an ninh, giao thông, và truyền thông. Đồ án này nhằm nghiên cứu và ứng dụng các kỹ thuật xử lý ảnh để giải quyết một số vấn đề cụ thể trong thực tế.

Trong đồ án này, em sẽ tập trung vào việc phân tích và xử lý các hình ảnh kỹ thuật số bằng cách sử dụng các công cụ và thư viện mạnh mẽ như Numpy, Pillow, và Matplotlib. Numpy sẽ hỗ trợ trong việc xử lý dữ liệu dưới dạng mảng đa chiều, Pillow sẽ cung cấp các phương thức để xử lý và thao tác trên các hình ảnh, trong khi đó Matplotlib sẽ giúp em trực quan hóa các kết quả một cách hiệu quả.

Đồ án này không chỉ giúp em củng cố kiến thức về lý thuyết xử lý ảnh mà còn trang bị cho em những kỹ năng thực hành cần thiết để giải quyết các bài toán thực tế trong lĩnh vực này. Qua đó, em hy vọng sẽ đóng góp một phần nhỏ vào sự phát triển của công nghệ xử lý ảnh và mở ra những hướng nghiên cứu mới trong tương lai.

Bố cục các phần còn lại được tổ chức như sau, Phần 3 là các thông tin của sinh viên thực hiện báo cáo. Phần 4 với các ý tưởng và mô tả về thuật toán và các hàm tiện ích. Cuối cùng phần 5 sẽ đưa ra đánh giá mức độ hoàn thành các công việc.

3 Thông tin sinh viên

- Họ tên: Đinh Vũ Gia Hân
- MSSV: 22127098
- Lớp: 22CLC08
- Email: dvghan22@clc.fitus.edu.vn

4 Ý tưởng thực hiện và mô tả các hàm chức năng:

4.1 Giới thiệu chung về đồ án

Trong đồ án trước, ta đã biết được rằng mỗi bức ảnh thực chất là một ma trận các pixel mà mỗi pixel biểu diễn một giá trị xám hoặc một điểm màu. Mỗi điểm màu được biểu diễn bằng bộ 3 số (r, g, b) . Để tiện cho việc xử lý ảnh thì sẽ tách ma trận pixel ra 3 kênh màu red, green, blue. Vì vậy trong đồ án này, ta sẽ vận dụng các phép tính trên ma trận đã học để xây dựng các thuật toán liên quan đến xử lý ảnh:

- Thay đổi độ sáng
- Thay đổi độ tương phản
- Lật ảnh ngang/dọc
- Chuyển ảnh màu (RGB) thành ảnh xám/sepia
- Làm mờ/sắc nét ảnh
- Cắt ảnh theo kích thước (cắt ở trung tâm)
- Cắt ảnh theo khung tròn/elip.

4.2 Các thư viện cần dùng

- Các thư viện bắt buộc sử dụng bao gồm:
 1. Numpy: dùng để tính toán ma trận
 2. Pillow: dùng để đọc ghi ảnh
 3. Matplotlib: dùng để hiển thị ảnh
- Các thư viện dùng để kiểm thử và viết báo cáo bao gồm:
 1. Time: dùng để tính toán thời gian thực thi

4.3 Mô tả các hàm chức năng

4.3.1 def read_img(img_path)

1. **Mô tả:** read_img được dùng để đọc hình ảnh từ đường dẫn cho trước.
2. **Tham số đầu vào:** Một biến string chứa đường dẫn đến file ảnh cần đọc.
3. **Các bước thực hiện:** Sử dụng hàm Image.open() trong thư viện Pillow để đọc ảnh từ đường dẫn cho trước. Nếu đường dẫn không tồn tại hoặc xảy ra bất kì lỗi gì khác thì in thông báo lỗi.
4. **Kết quả đầu ra:** Một đối tượng thuộc lớp PIL.Image.Image

4.3.2 def show_img(result_img, func)

1. **Mô tả:** show_img được dùng để hiển thị hình ảnh sau khi được xử lý.
2. **Tham số đầu vào:** Một biến thuộc lớp PIL.Image.Image chứa ảnh sau khi xử lý, và cuối cùng là biến string lưu tên hàm chức năng được dùng để xử lý ảnh.
3. **Các bước thực hiện:** Sử dụng hàm matplotlib.pyplot.imshow() để hiển thị hình ảnh. Nếu ảnh như hàm chức năng dùng để xử lý ảnh là hàm RGB_to_gray thì sẽ hiển thị hình ảnh với tham số cmap='gray'.
4. **Kết quả đầu ra:** Hình ảnh được hiển thị trên màn hình.

4.3.3 def save_img(input_file, result_img, func)

1. **Mô tả:** save_img là hàm dùng để lưu hình ảnh vào cùng thư mục với ảnh gốc.
2. **Tham số đầu vào:** Một biến string lưu tên ảnh được nhập, một biến thuộc lớp PIL.Image.Image chứa ảnh sau khi xử lý, và cuối cùng là biến string lưu tên hàm chức năng được dùng để xử lý ảnh.
3. **Các bước thực hiện:** Sử dụng hàm numpy.split() để lấy được tên file và đuôi file. Ảnh kết quả sẽ được lưu tên với cấu trúc 'tenfile_chucnang.duoiinfile' bằng hàm Image.save() trong thư viện Pillow.
4. **Kết quả đầu ra:** Ảnh đã qua xử lý được lưu vào cùng thư mục với ảnh gốc.

4.3.4 def main()

1. **Mô tả:** main được dùng để xử lý dữ liệu đầu vào và thực thi các chức năng của chương trình.
2. **Tham số đầu vào:** Không có tham số đầu vào.
3. **Các bước thực hiện:**
 - Cho phép người dùng nhập tên file ảnh và chọn chức năng muốn thực hiện.
 - Đọc file ảnh và đưa vào xử lý ảnh.
 - In ra ảnh kết quả và lưu kết quả vào cùng thư mục với ảnh gốc.
4. **Kết quả đầu ra:** In ra ảnh kết quả.

4.3.5 def adjust_brightness(img, brightness)

1. **Ý tưởng:** Điều chỉnh độ sáng của bức ảnh là thuật toán xử lý hình ảnh dễ nhất. Để thay đổi độ sáng của một bức ảnh, ta chỉ đơn giản thay đổi giá trị mỗi kênh màu của các điểm ảnh bằng cách thêm một giá trị như nhau thuộc đoạn $[-255; 255]$. Giá trị thêm càng lớn thì ảnh càng sáng. Ngược lại, giá trị thêm càng nhỏ thì ảnh càng tối [1].
2. **Mô tả:** adjust_brightness dùng để thay đổi độ sáng của ảnh với một giá trị cho trước.
3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc và một biến int lưu giá trị độ sáng muốn tăng hoặc giảm.
4. **Các bước thực hiện:**
 - Đầu tiên, ta dùng hàm numpy.clip() để giới hạn độ sáng muốn tăng hoặc giảm trong đoạn $[-255; 255]$ nghĩa là giá trị nhỏ hơn -255 sẽ được đặt thành -255 và giá trị lớn hơn 255 sẽ được đặt thành 255.
 - Sử dụng hàm numpy.array() để chuyển ảnh gốc truyền vào thành ma trận img_arr hai chiều chứa các điểm ảnh.
 - Cộng vào 3 kênh màu của tất cả các điểm ảnh một lượng bằng nhau để thay đổi độ sáng bằng toán tử (+) một số với mảng. Sử dụng hàm numpy.clip() để giới hạn giá trị các kênh màu trong đoạn $[0; 255]$. Lưu ma trận điểm ảnh với vào bright_img_arr.
 - Chuyển ma trận bright_img_arr thành ảnh kết quả nhờ hàm Image.fromarray().
5. **Kết quả đầu ra:** Ảnh sau khi được điều chỉnh độ sáng.

4.3.6 def adjust_contrast(img, contrast)

1. Ý tưởng:

- Độ tương phản (Contrast) là chỉ số chỉ sự khác biệt giữa hai màu đen và trắng trên màn hình. Khoảng cách giữa hai mức đen trắng gần nhau nhất được gọi là step, và trong khoảng từ mức sáng nhất (max level) đến tối nhất (min level) càng có nhiều step, thì màn hình càng có khả năng hiển thị sắc nét.
- Để điều chỉnh độ tương phản của một bức ảnh, ta sẽ sử dụng công thức được đưa ra bởi Francis G. Loch [2].

– Bước đầu tiên là tính hệ số hiệu chỉnh độ tương phản được tính theo công thức sau:

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

- Để thuật toán hoạt động chính xác, giá trị của hệ số hiệu chỉnh độ tương phản (F) cần được lưu trữ dưới dạng số thực. Giá trị C trong công thức trên biểu thị mức độ tương phản mong muốn.
- Bước tiếp theo là thực hiện việc điều chỉnh độ tương phản thực tế. Công thức sau đây cho thấy sự điều chỉnh độ tương phản được thực hiện đối với kênh màu red:

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

- Áp dụng tương tự cho các kênh màu còn lại thì ta sẽ được một bức ảnh với độ tương phản với. Giá trị C sẽ nằm trong đoạn $[-255; 255]$. C càng nhỏ thì độ tương phản càng giảm, C càng lớn thì độ tương phản càng tăng

2. **Mô tả:** adjust_contrast là hàm dùng để thay đổi độ tương phản của ảnh với một giá trị cho trước.

3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc và một biến int lưu giá trị độ sáng muốn tăng hoặc giảm.

4. Các bước thực hiện:

- Đầu tiên, ta dùng hàm numpy.clip() để giới hạn độ tương phản muốn tăng hoặc giảm trong đoạn $[-255; 255]$.
- Sử dụng công thức (1) để tính hệ số điều chỉnh factor.
- Sử dụng hàm numpy.array() để chuyển ảnh gốc truyền vào thành ma trận img_arr hai chiều chứa các điểm ảnh.

- Áp dụng công thức (2) để điều chỉnh độ tương phản của ma trận điểm ảnh bằng toán tử (+) và (*) một số với ma trận. Sử dụng hàm `numpy.clip()` để giới hạn giá trị các kênh màu trong đoạn `[0; 255]`. Lưu ma trận điểm ảnh với vào `contrast_img_arr`.
- Chuyển ma trận `contrast_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được điều chỉnh độ tương phản

4.3.7 def flip_horizontal(img)

1. Ý tưởng:

- Một bức ảnh thực chất là một ma trận hai chiều chứa các điểm ảnh. Vì vậy để lật ảnh ngang thì ta chỉ cần đảo các cột của ma trận điểm ảnh:

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ i_{n1} & i_{n2} & \cdots & i_{nn} \end{bmatrix} \longrightarrow \begin{bmatrix} i_{1n} & \cdots & i_{12} & i_{11} \\ \vdots & \ddots & \vdots & \vdots \\ i_{nn} & \cdots & i_{n2} & i_{n1} \end{bmatrix}$$

- Để đảo các cột của ma trận điểm ảnh thì ta sẽ sử dụng kỹ thuật Slicing trong Python Numpy. **Slicing** có nghĩa là lấy các phần tử trong một khoảng giá trị từ giá trị index tới giá trị index khác.
- Chúng ta cũng có thể sử dụng Slicing với ma trận hai chiều Numpy. Khi sử dụng Slicing trong ma trận hai chiều ta cần xác định được vị trí hàng bắt đầu cắt, vị trí hàng kết thúc cắt và vị trí cột bắt đầu cắt, vị trí cột kết thúc cắt với cú pháp như sau:

$$[row_start : row_end, col_start : col_end]$$

- Nếu như ta muốn lấy các phần tử cách nhau n bước nhảy, ta chỉ cần sử dụng Slicing `[::n]`.
- Lưu ý: ta cũng có thể sử dụng hàm `numpy.fliplr()` để đảo ngược thứ tự các phần tử dọc theo trục 1 (trái/phải). Đối với mảng hai chiều, thao tác này sẽ lật các mục trong mỗi hàng theo hướng trái/phải. Các cột được giữ nguyên nhưng xuất hiện theo thứ tự khác so với trước đây [3].

2. **Mô tả:** `flip_horizontal` dùng để lật ảnh theo chiều ngang.

3. **Tham số đầu vào:** Một biến `PIL.Image.Image` chứa ảnh gốc.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh.
- Sử dụng kỹ thuật Slicing để đảo các cột của ma trận điểm ảnh `img_arr[:, ::-1]`. Lưu ma trận điểm ảnh với vào `flipped_img_arr`.
 - `img_arr[:, ::-1]` sẽ cắt ra tất cả các hàng và tất cả các cột với bước nhảy -1. Với bước nhảy -1, Slicing sẽ lấy ra các cột từ cuối đến 0.
- Chuyển ma trận `flipped_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được lật ngang.

4.3.8 def flip_vertical(img)

1. Ý tưởng:

- Để lật ảnh dọc thì ta chỉ cần đảo các dòng của ma trận điểm ảnh:

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ i_{n1} & i_{n2} & \cdots & i_{nn} \end{bmatrix} \longrightarrow \begin{bmatrix} i_{n1} & i_{n2} & \cdots & i_{nn} \\ \vdots & \ddots & \vdots & \vdots \\ i_{11} & i_{12} & \cdots & i_{1n} \end{bmatrix}$$

- Tương tự như lật ảnh ngang, ta sẽ sử dụng kỹ thuật Slicing trong Python Numpy để đảo các dòng của ma trận điểm ảnh.
- Lưu ý: ta cũng có thể sử dụng hàm **`numpy.flipud()`** để đảo ngược thứ tự các phần tử dọc theo trục 0 (lên/xuống). Đối với mảng hai chiều, thao tác này sẽ lật các mục trong mỗi cột theo hướng lên/xuống. Các hàng được giữ nguyên nhưng xuất hiện theo thứ tự khác so với trước đây [4].

2. **Mô tả:** `flip_vertical` dùng để lật ảnh theo chiều dọc.

3. **Tham số đầu vào:** Một biến `PIL.Image.Image` chứa ảnh gốc.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh.
- Sử dụng kỹ thuật Slicing để đảo các dòng của ma trận điểm ảnh `img_arr[::-1, :]`. Lưu ma trận điểm ảnh với vào `flipped_img_arr`.
 - `img_arr[::-1, :]` sẽ cắt ra tất cả các hàng với bước nhảy -1 và tất cả các cột. Với bước nhảy -1, Slicing sẽ lấy ra các dòng từ cuối đến 0.

- Chuyển ma trận `flipped_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được lật dọc.

4.3.9 `def RGB_to_gray(img)`

1. Ý tưởng:

- Tương tự ảnh màu, ảnh xám cũng là một ma trận hai chiều điểm ảnh. Tuy nhiên mỗi pixel trong ảnh xám chỉ cần biểu diễn bằng một giá trị nguyên trong khoảng từ $[0, 255]$ thay vì (r, g, b) như trong ảnh màu.
- Để chuyển từ ảnh màu thành ảnh xám thì ta có hai cách: Average method và Weighted method (Luminosity method)
- **Average method** sẽ lấy trung bình ba màu đỏ, xanh lá cây, xanh lam và trả về một hình ảnh khá đen. Vấn đề này phát sinh do việc lấy trung bình của ba màu. Vì ba màu khác nhau có ba bước sóng khác nhau và có sự đóng góp riêng trong việc hình thành hình ảnh nên ta phải lấy trung bình theo sự đóng góp của chúng.
- **Weighted method** là một giả pháp cho vấn đề ở Average method. Vì màu đỏ có nhiều bước sóng hơn trong cả ba màu và màu xanh lá cây là màu không chỉ có bước sóng ít hơn màu đỏ mà còn màu xanh lá cây là màu mang lại hiệu ứng dịu mắt hơn. Điều đó có nghĩa là ta phải giảm sự đóng góp của màu đỏ và tăng sự đóng góp của màu xanh lá cây, đồng thời đặt sự đóng góp của màu xanh lam vào giữa hai màu này.
- Do đó ta sẽ sử dụng công thức trong Weighted method đã được đưa ra bởi trang tutorialspoint [5]:

$$New\ grayscale\ image = (0.3R + 0.59G + 0.11B)$$

2. **Mô tả:** `RGB_to_gray` dùng để chuyển ảnh màu thành ảnh xám.

3. **Tham số đầu vào:** Một biến `PIL.Image.Image` chứa ảnh gốc.

4. Các bước thực hiện:

- Sử dụng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh.
- Tạo một mảng `color_weight` để chứa trọng số màu của các kênh màu.
- Dùng hàm `numpy.matmul()` để thực tính tích vô hướng các điểm ảnh trong ma trận gốc với mảng trọng số màu nhờ toán tử $(*)$ mảng với mảng. Sử dụng hàm `numpy.clip()` để giới hạn giá trị các kênh màu trong đoạn $[0; 255]$. Lưu ma trận điểm ảnh với vào `gray_img_arr`.

- Chuyển ma trận `gray_img_arr` thành ảnh kết quả nhờ hàm `Image.formarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được chuyển thành ảnh xám.

4.3.10 def RGB_to_sepia(img)

1. Ý tưởng:

- Ảnh sepia là ảnh có chất màu nâu sẫm lấy từ túi mực của con mực làm cho bức ảnh trở nên cổ điển và ấm áp.
- Để chuyển từ ảnh RGB thành sepia, ta có thể thay đổi trọng số của ba kênh màu đỏ, xanh lá cây, xanh lam của mỗi điểm ảnh bằng các hệ số được xác định trước, dựa trên sự đóng góp của từng kênh vào hình ảnh sepia cuối cùng.
- Do đó ta sẽ sử dụng Transformation Factor đã được đưa ra bởi trang tutorialspoint [6]:

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

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

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

- Tương tự như ảnh xám, để thực hiện được phép tính này, ta thực hiện phép nhân ma trận điểm ảnh với ma trận trọng số màu.

2. **Mô tả:** `RGB_to_sepia` dùng để chuyển ảnh màu thành ảnh sepia.

3. **Tham số đầu vào:** Một biến `PIL.Image.Image` chứa ảnh gốc.

4. Các bước thực hiện:

- Sử dụng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh.
- Tạo một mảng `color_weight` để chứa trọng số màu của các kênh màu.
- Dùng hàm `numpy.matmul()` để thực tính tích vô hướng các điểm ảnh trong ma trận gốc với ma trận trọng số màu nhờ toán tử `(*)` mảng với mảng. Sử dụng hàm `numpy.clip()` để giới hạn giá trị các kênh màu trong đoạn `[0; 255]`. Lưu ma trận điểm ảnh với vào `sepia_img_arr`.
- Chuyển ma trận `sepia_img_arr` thành ảnh kết quả nhờ hàm `Image.formarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được chuyển thành ảnh sepia.

4.3.11 def blur_image(img)

1. Ý tưởng:

- Để làm mờ một bức ảnh thì ta cần tính toán lại giá trị màu của toàn bộ điểm ảnh trong ảnh gốc.
- Để phân bố lại màu sắc của bức ảnh, ta sẽ áp dụng một bộ lọc Gaussian Kernel lên ảnh.
- Trong kỹ thuật xử lý ảnh, kernel là một ma trận nhỏ dùng để làm mờ, làm sắc nét, làm nổi,... Điều này được thực hiện bằng cách thực hiện phép tích chập (convolution) giữa Gaussian Kernel và ma trận điểm ảnh gốc. Hay đơn giản hơn, khi mỗi pixel trong ảnh đầu ra là một giá trị trung bình được tính bởi các pixel lân cận (bao gồm cả chính nó) với trọng số lớn nhất tại điểm đó. Các điểm càng xa điểm đang xét, trọng số càng nhỏ.
- Tích chập (Convolution) là quá trình thêm từng phần tử của hình ảnh vào các phần tử lân cận cục bộ của nó, được tính theo trọng số của kernel. Với mỗi phần tử x_{ij} trong ma trận ảnh gốc X lấy ra một ma trận con bằng kích thước của kernel K sao cho x_{ij} là trung tâm, gọi ma trận con này là ma trận SubX. Thực hiện phép tích chập SubX với K ta sẽ thu được giá trị màu của điểm ảnh kết quả. Gán kết quả này vào y_{ij} của ma trận kết quả Y. Sau khi duyệt qua tất cả các điểm ảnh của ma trận X ta sẽ thu được ma trận Y đã được làm mờ.
- Tuy nhiên đối với các điểm ảnh x_{ij} nằm ở rìa sẽ không có đủ phần tử lân cận để tìm ma trận con SubX sao cho x_{ij} nằm ở trung tâm. Do đó ta cần thêm vào 4 cạnh của ma trận X các padding. Padding là các điểm ảnh 0 để đảm bảo chất lượng màu sắc không bị ảnh hưởng.
- Để tính các trọng số trong Gaussian Kernel ta sử dụng hàm Gauss [7]:

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

- Trong đó x, y là khoảng cách của vị trí đang xét trong ma trận với điểm trung tâm của ma trận, σ là độ lệch chuẩn trong phân phối Gauss.
- Có hai ma trận Gaussian Kernel cơ bản thường được dùng để làm mờ ảnh: Gaussian blur 3x3 và Gaussian blur 5x5. Tuy nhiên, với kích thước bộ lọc càng lớn thì hiệu quả

nó mang lại càng tốt, nên ta sẽ chọn Gaussian blur 5x5 để sử dụng trong đồ án này.

$$Gaussian_Kernel_5 = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- Để hiểu rõ hơn, sau đây là một ví dụ về việc làm mờ bức ảnh 2x2 với Gaussian Kernel 5x5:

– Bước 1: Thêm các padding vào ma trận gốc.

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

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

$$SubX \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{55}{64} \longrightarrow Y \begin{bmatrix} \frac{55}{64} & \dots \\ \dots & \dots \end{bmatrix}$$

- Bước 3: Lập lại bước 2 cho tất cả các phần tử của ma trận gốc.

$$SubX \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{15}{16} \longrightarrow Y \begin{bmatrix} \frac{55}{64} & \frac{15}{16} \\ \dots & \dots \end{bmatrix}$$

$$\begin{aligned}
 &SubX \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{65}{64} \longrightarrow Y \begin{bmatrix} \frac{55}{64} & \frac{15}{16} \\ \frac{65}{64} & \dots \end{bmatrix} \\
 &SubX \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times K \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{69}{64} \longrightarrow Y \begin{bmatrix} \frac{55}{64} & \frac{15}{16} \\ \frac{65}{64} & \frac{69}{64} \end{bmatrix}
 \end{aligned}$$

- Bước 4: Sau khi duyệt qua tất cả các phần tử, ta được ma trận kết quả Y đã được làm mờ.

2. **Mô tả:** blur_image dùng để làm mờ ảnh.

3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm `numpy.array()` để tạo một mảng để chứa Gaussian Kernel.
- Sử dụng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh dưới dạng số thực.
- Kết hợp `numpy.shape()` và Slicing để xác định kích thước của ảnh gốc cũng như kernel.
- Tính số padding cần thêm vào để đảm bảo các phần tử ngoài rìa đó đủ phần tử lân cận để tạo thành ma trận con bằng với kernel.
- Tạo ma trận 0 có cùng kích thước với ảnh gốc để lưu kết quả bằng hàm `numpy.zeros()`.
- Tạo một ma trận tmp với kích thước bằng kích thước của ảnh gốc cộng thêm padding.
- Sử dụng Slicing để gán kết quả của ma trận gốc vào trung tâm ma trận tmp vừa tạo.
- Sử dụng 2 vòng lặp for để duyệt qua tất cả các phần tử trong ma trận gốc. Nhờ vào kỹ thuật Slicing ta sẽ lấy ra được các ma trận con có cùng kích thước với kernel với điểm đang xét là trung tâm. Do ma trận điểm ảnh có thêm một chiều không gian thứ 3 để lưu các kênh màu nên ta dùng hàm `numpy.newaxis()` để thêm một chiều không gian cho kernel.
- Sử dụng toán tử (*) một mảng với một mảng kết hợp với hàm `numpy.sum()` để thực hiện phép tích chập. Sử dụng hàm `numpy.clip()` để giới hạn giá trị các kênh màu trong đoạn [0; 255]. Ma trận kết quả sẽ được lưu trong `blurred_img_arr`.

- Chuyển ma trận `blurred_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được làm mờ.

4.3.12 `def sharpen_image(img)`

1. **Ý tưởng:**

- Để làm sắc nét một bức ảnh thì ta sẽ lặp lại các bước tương tự như làm mờ ảnh.
- Tuy nhiên, ta sẽ áp dụng một bộ lọc khác tên là Sharpen Kernel lên ảnh.

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

2. **Mô tả:** `sharpen_image` dùng để làm sắc nét ảnh.

3. **Tham số đầu vào:** Một biến `PIL.Image.Image` chứa ảnh gốc.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm `numpy.array()` để tạo một mảng để chứa Sharpen Kernel.
- Sử dụng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh dưới dạng số thực.
- Kết hợp `numpy.shape()` và Slicing để xác định kích thước của ảnh gốc cũng như kernel.
- Tính số padding cần thêm vào để đảm bảo các phần tử ngoài rìa đó đủ phần tử lân cận để tạo thành ma trận con bằng với kernel.
- Tạo ma trận 0 có cùng kích thước với ảnh gốc để lưu kết quả bằng hàm `numpy.zeros()`.
- Tạo một ma trận `tmp` với kích thước bằng kích thước của ảnh gốc cộng thêm padding.
- Sử dụng Slicing để gán kết quả của ma trận gốc vào trung tâm ma trận `tmp` vừa tạo.
- Sử dụng 2 vòng lặp `for` để duyệt qua tất cả các phần tử trong ma trận gốc. Nhờ vào kỹ thuật Slicing ta sẽ lấy ra được các ma trận con có cùng kích thước với kernel với điểm đang xét là trung tâm. Do ma trận điểm ảnh có thêm một chiều không gian thứ 3 để lưu các kênh màu nên ta dùng hàm `numpy.newaxis()` để thêm một chiều không gian cho kernel.
- Sử dụng toán tử `(*)` một mảng với một mảng kết hợp với hàm `numpy.sum()` để thực hiện phép tích chập. Sử dụng hàm `numpy.clip()` để giới hạn giá trị các kênh màu trong đoạn `[0; 255]`. Ma trận kết quả sẽ được lưu trong `sharpen_img_arr`.

- Chuyển ma trận `sharpen_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được làm sắc nét.

4.3.13 `def crop_image(img, crop_height, crop_width)`

1. **Ý tưởng:**

- Mỗi bức ảnh thực chất là một ma trận hai chiều điểm ảnh. Vì vậy để cắt được một bức ảnh ở trung tâm thì ta chỉ đơn giản xóa đi các dòng và cột ở rìa ma trận.
- Để cắt một bức ảnh, ta sẽ xóa đi các dòng và cột bằng cách sao chép các phần được giữ lại và lưu vào một ma trận mới.

2. **Mô tả:** `crop_image` dùng để cắt ảnh từ vị trí trung tâm.

3. **Tham số đầu vào:** Một biến `PIL.Image.Image` chứa ảnh gốc và hai biến `int` chứa kích thước ảnh muốn cắt.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh.
- Kết hợp `numpy.shape()` và Slicing để xác định kích thước của ma trận điểm ảnh.
- Tính tọa độ điểm bắt đầu để cắt. Sử dụng hàm `numpy.clip()` để đảm bảo kích thước ảnh muốn cắt luôn nhỏ hơn hoặc bằng ảnh gốc.
- Sử dụng Slicing để cắt đi các dòng và cột ở phần rìa và lưu ma trận điểm ảnh với vào `cropped_img_arr`.
- Chuyển ma trận `cropped_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được cắt.

4.3.14 `def circle_frame(img)`

1. **Ý tưởng:**

- Như đã biết, mỗi bức ảnh là một ma trận hai chiều điểm ảnh với mỗi điểm ảnh có tọa độ (i, j) . Do đó để cắt một bức ảnh vuông theo khung tròn, ta sẽ sử dụng kiến thức về phương trình đường tròn. Các điểm ảnh nằm bên trong đường tròn sẽ được giữ nguyên màu sắc, các điểm ảnh nằm bên ngoài sẽ được đổi thành màu đen.

- Phương trình đường tròn tâm I có tọa độ (a, b) và bán kính R:

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

- Để một điểm ảnh có tọa độ (i, j) nằm trong đường tròn thì nó phải thỏa điều kiện sau:

$$(i - a)^2 + (j - b)^2 \leq R^2 (*)$$

- Tuy nhiên khung tròn được cắt phải tiếp xúc với bốn cạnh của bức ảnh, do đó tọa độ tâm của đường tròn trùng với tâm của bức ảnh và bán kính của đường tròn phải bằng một nửa kích thước cạnh của ảnh.

2. **Mô tả:** circle_frame dùng để cắt ảnh vuông theo khung tròn.

3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm numpy.array() để chuyển ảnh gốc truyền vào thành ma trận img_arr hai chiều chứa các điểm ảnh.
- Lấy kích thước một cạnh của ma trận điểm ảnh và tính tọa độ tâm và bán kính của đường tròn cần cắt.
- Tạo hai ma trận matrix_idx_i và matrix_idx_j để chứa tọa độ hàng và tọa độ cột. Sử dụng kết hợp hai hàm numpy.arange(size) và numpy.tile() để sao chép một ma trận một chiều chứa các giá trị từ 0 đến size - 1 và ghép nối với nhau theo tham số reps. Do reps = (size, 1) nên ma trận này sẽ được sao chép size lần theo hàng (chiều 0) và một lần theo cột (chiều 1). Lưu ma trận vừa tạo vào matrix_idx_j.
- Do ma trận matrix_idx_i là ma trận chuyển vị của matrix_idx_j nên ta dùng hàm numpy.transpose() để thực hiện phép chuyển vị matrix_idx_j.
- Dùng một mảng boolean có cùng kích thước với ma trận để đánh dấu các điểm ảnh nằm bên ngoài đường tròn (không thỏa điều kiện (*)).
- Do mask là mảng boolean nên Với các điểm ảnh nằm bên ngoài đường tròn, d img[mask] = (0, 0, 0) sẽ đặt chúng thành màu đen.
- Chuyển ma trận degreele_img_arr thành ảnh kết quả nhờ hàm Image.fromarray().

5. **Kết quả đầu ra:** Ảnh sau khi được cắt.

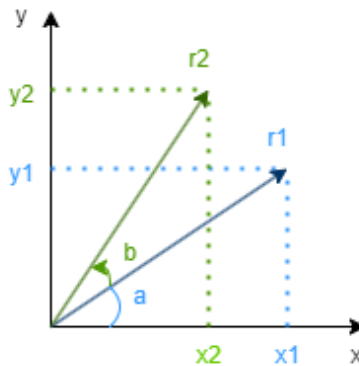
4.3.15 def ellipse_frame(img)

1. Ý tưởng:

- Để cắt ảnh theo khung 2 hình elip chéo nhau thì ta cũng thực hiện tương tự như cắt ảnh theo khung hình tròn. Tuy nhiên việc cắt theo khung elip phức tạp hơn do ta phải cắt 2 hình elip sao cho trục của chúng trùng với 2 đường chéo và tiếp xúc đầy đủ với 4 cạnh của ảnh vuông.
- Như đã được học ở lớp 10, một đường elip tâm (m, n) với trục lớn, trục nhỏ lần lượt là a, b có phương trình như sau:

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

- Để elip có 2 trục nằm trên 2 đường chéo của bức ảnh, ta cần thực hiện phép quay trục một góc 45° hoặc 135° . Như vậy ta sẽ có hai hình elip chéo nhau có trục trùng với đường chéo của ảnh.
- Ví dụ về phép quay vector:
 - Giả sử ta có một vector r_1 hai chiều (x_1, y_1) tạo với trục x một góc a . Quay r_1 một góc b ta được một vector r_2 hai chiều (x_2, y_2) .



Hình 1: Ví dụ về phép quay trục

- Ta có tọa độ của vector r_1 và r_2 được tính như sau:

$$x_1 = r_1 \cos(a)$$

$$y_1 = r_1 \sin(a)$$

$$x_2 = r_2 \cos(a + b)$$

$$y_2 = r_2 \sin(a + b)$$

- Để khai triển $\cos(a+b)$ và $\sin(a+b)$, ta Sử dụng công thức lượng giác:

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$

$$\sin(a+b) = \cos(a)\sin(b) + \sin(a)\cos(b)$$

- Ta thấy rằng khi quay r_1 thì độ dài của nó vẫn giữ nguyên và $x_2 = y_1$, $y_2 = x_1$. Do đó khai triển ra ta được:

$$x_2 = r_2 \cos(a) \cos(b) - r_2 \sin(a) \sin(b)$$

$$y_2 = r_2 \cos(a) \sin(b) + r_2 \sin(a) \cos(b)$$

$$\begin{aligned} \longrightarrow x_2 &= r_1 \cos(a) \cos(b) - r_1 \sin(a) \sin(b) \\ y_2 &= r_1 \cos(a) \sin(b) + r_1 \sin(a) \cos(b) \end{aligned}$$

$$\begin{aligned} \longrightarrow x_2 &= x_1 \cos(b) - y_1 \sin(b) \\ y_2 &= x_1 \sin(b) + y_1 \cos(b) \end{aligned} \quad (*)$$

- Tóm lại để xoay một vector một góc b , ta sẽ áp dụng công thức (*) được chứng minh ở trên.

- Áp dụng công thức (*) để xoay trục elip một góc α ta được:

$$\frac{(x\cos(\alpha) - y\sin(\alpha))^2}{a^2} + \frac{(x\sin(\alpha) + y\cos(\alpha))^2}{b^2} = 1$$

- Tiếp theo ta cần giải quyết sao cho elip phải tiếp xúc với cả 4 cạnh của bức ảnh. Để giải quyết vấn đề này, ta cần tìm ra được mối liên hệ giữa trục lớn, trục nhỏ của elip và kích thước của bức ảnh.
- Xét hình elip với trục xoay góc 45° so với trục của bức ảnh. Ta khai triển và tách x, y ra riêng biệt:

$$\frac{(x\cos(\frac{\pi}{4}) - y\sin(\frac{\pi}{4}))^2}{a^2} + \frac{(x\sin(\frac{\pi}{4}) + y\cos(\frac{\pi}{4}))^2}{b^2} = 1$$

$$\longleftrightarrow x^2(\frac{\cos^2(\frac{\pi}{4})}{a^2} + \frac{\sin^2(\frac{\pi}{4})}{b^2}) + y^2(\frac{\sin^2(\frac{\pi}{4})}{a^2} + \frac{\cos^2(\frac{\pi}{4})}{b^2}) + xy(\frac{\sin(\frac{\pi}{2})}{b^2} - \frac{\sin(\frac{\pi}{2})}{a^2}) = 1$$

$$\longleftrightarrow x^2(\frac{1}{2a^2} + \frac{1}{2b^2}) + y^2(\frac{1}{2a^2} + \frac{1}{2b^2}) + xy(\frac{1}{b^2} - \frac{1}{a^2}) - 1 = 0 \quad (**)$$

- Do hình elip đối xứng qua hai trục lớn và nhỏ nên ta chỉ cần tìm điều kiện để elip tiếp xúc với một cạnh thì cũng sẽ thỏa tiếp xúc 3 cạnh còn lại.

- Giả sử elip tiếp xúc với cạnh hình vuông tại điểm có tung độ $y = c$, thì ta cần tìm a, b sao cho chỉ có duy nhất một nghiệm x thỏa phương trình (**). Thay $y = c$ vào (**) ta được:

$$f(x) = x^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) + c\left(\frac{1}{b^2} - \frac{1}{a^2}\right)x + c^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) - 1$$

- Phương trình giờ đây đã trở thành hàm một biến và là một parabol. Do đó để phương trình có một nghiệm duy nhất thì đỉnh của parabol phải nằm trên trục Ox. Mà tọa độ đỉnh của parabol $y = ax^2 + bx + c$ là $(\frac{-b}{2a}, f(\frac{-b}{2a}))$. Suy ra $f(\frac{-b}{2a})$ phải bằng 0.

$$f\left(\frac{c(\frac{1}{a^2} - \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}\right) = 0$$

$$\longleftrightarrow \left(\frac{c(\frac{1}{a^2} - \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}\right)^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) + c\left(\frac{1}{b^2} - \frac{1}{a^2}\right)\left(\frac{c(\frac{1}{a^2} - \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}\right) + c^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) - 1 = 0$$

$$\longleftrightarrow \frac{(c(\frac{1}{a^2} - \frac{1}{b^2}))^2}{2(\frac{1}{a^2} + \frac{1}{b^2})} + \frac{c^2(\frac{1}{a^2} - \frac{1}{b^2})^2}{\frac{1}{a^2} + \frac{1}{b^2}} + c^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) - 1 = 0$$

$$\longleftrightarrow -\frac{(c(\frac{1}{a^2} - \frac{1}{b^2}))^2}{2(\frac{1}{a^2} + \frac{1}{b^2})} + c^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) - 1 = 0$$

$$\longleftrightarrow \frac{c^2}{2}\left(\frac{1}{a^2} + \frac{1}{b^2} - \frac{(\frac{1}{a^2} - \frac{1}{b^2})^2}{\frac{1}{a^2} + \frac{1}{b^2}}\right) = 1$$

...

$$\longleftrightarrow a^2 + b^2 = c^2$$

- Vì tâm của bức ảnh trùng với gốc tọa độ O nên c sẽ bằng một nửa kích thước của cạnh của bức ảnh. Vì vậy để elip tiếp xúc với bốn cạnh của bức ảnh và hai elip trùng với hai đường chéo thì góc quay phải là 45° và 135° , đồng thời độ dài hai trục phải thỏa: $a^2 + b^2 = \frac{canh^2}{4}$
- Ta có thể chọn a, b tùy ý mà vẫn đảm bảo a lớn hơn b . Do đó ở đây ta sẽ chọn:

$$a^2 = \frac{3}{8}canh^2$$

$$b^2 = \frac{1}{8}canh^2$$

- Do đó ta có phương trình đường elip tâm (m, n) với trục xoay một góc a như sau:

$$(E) : \frac{((x-m)\cos(a) - (y-n)\sin(a))^2}{\frac{3}{8}canh^2} + \frac{((x-m)\sin(a) - (y-n)\cos(a))^2}{\frac{1}{8}canh^2} = 1 (***)$$

- Vậy để biết một điểm ảnh có nằm trong đường elip hay không, ta chỉ cần thay tọa độ của nó vào (***) để biết.

2. **Mô tả:** ellipse_frame dùng để cắt ảnh theo khung 2 hình elip chéo nhau.

3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc.

4. **Các bước thực hiện:**

- Đầu tiên, ta dùng hàm `numpy.array()` để chuyển ảnh gốc truyền vào thành ma trận `img_arr` hai chiều chứa các điểm ảnh.
- Lấy kích thước một cạnh của ma trận điểm ảnh và tính tọa độ tâm của đường elip cần cắt.
- Đặt giá trị trục lớn và trục nhỏ sao cho thỏa điều kiện đã chứng minh ở trên.
- Tương tự như cắt ảnh theo khung tròn, ta cũng sẽ tạo hai ma trận `matrix_idx_i` và `matrix_idx_j` để lưu tọa độ dòng và cột.
- Hàm hỗ trợ `mask_degree()` với các tham số: trục lớn, trục nhỏ, góc quay, `matrix_idx_i`, `matrix_idx_j`. Hàm này dựa trên công thức mà ta đã lập ra ở phần trên để tính toán và trả về mảng boolean đánh dấu các điểm ảnh nằm bên ngoài hai đường elip.
- `mask1` dùng để đánh dấu các điểm ảnh nằm bên ngoài đường elip có trục quay một góc 45° so với trục của ảnh và `mask2` dùng để đánh dấu các điểm ảnh nằm bên ngoài đường elip có trục quay một góc 135° so với trục của ảnh.
- sử dụng hàm `numpy.logical_and()` để thực hiện toán tử AND giữa các phần tử của `mask1` và `mask2`. Sau khi thực hiện ta sẽ được ma trận `masks` đánh dấu những điểm ảnh nằm ở ngoài cả hai elip.
- Do `masks` là mảng boolean nên Với các điểm ảnh nằm bên ngoài đường tròn, `d img[masks] = (0, 0, 0)` sẽ đặt chúng thành màu đen.
- Chuyển ma trận `ellipse_img_arr` thành ảnh kết quả nhờ hàm `Image.fromarray()`.

5. **Kết quả đầu ra:** Ảnh sau khi được cắt.

4.3.16 def zoom_in(img, factor)

1. **Ý tưởng:** Để phóng to một bức ảnh, tức là ta sẽ thấy các chi tiết của chúng gần hơn, do đó ta sẽ phải thêm các điểm ảnh vào ma trận điểm ảnh để màu sắc được rõ hơn.
2. **Mô tả:** zoom_in dùng để phóng to một bức ảnh.
3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc và một biến float để lưu giá trị muốn phóng to.
4. **Các bước thực hiện:**
 - Đầu tiên, ta dùng hàm numpy.array() để chuyển ảnh gốc truyền vào thành ma trận img_arr hai chiều chứa các điểm ảnh.
 - Sử dụng numpy.repeat() để tạo ra một ma trận điểm ảnh mới với từng pixel được thêm vào factor lần. Lưu ma trận mới trong zoomed_in_arr
 - Chuyển ma trận zoomed_in_arr thành ảnh kết quả nhờ hàm Image.fromarray()
5. **Kết quả đầu ra:** Ảnh sau khi được phóng to.


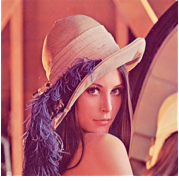

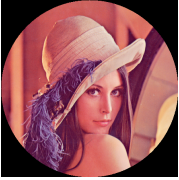



4.3.17 def zoom_out(img, factor)

1. **Ý tưởng:** Để thu nhỏ một bức ảnh, tức là ta sẽ thấy các chi tiết của chúng xa hơn, do đó ta sẽ phải chọn ra một số màu đại diện và bỏ bớt điểm ảnh đi.
2. **Mô tả:** zoom_out dùng để thu nhỏ một bức ảnh.
3. **Tham số đầu vào:** Một biến PIL.Image.Image chứa ảnh gốc và một biến float để lưu giá trị muốn thu nhỏ.
4. **Các bước thực hiện:**
 - Đầu tiên, ta dùng hàm numpy.array() để chuyển ảnh gốc truyền vào thành ma trận img_arr hai chiều chứa các điểm ảnh.
 - Sử dụng Slicing để tạo ra một ma trận mới. Ở đây ta sẽ lấy các điểm ảnh cách nhau factor bước nhảy. Lưu ma trận mới trong zoomed_out_arr
 - Chuyển ma trận zoomed_out_arr thành ảnh kết quả nhờ hàm Image.fromarray()
5. **Kết quả đầu ra:** Ảnh sau khi được thu nhỏ.

5 Đánh giá 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 = 5, sigma = 5)	100%	
5.2	Sắc nét ảnh (size & sigma = 5)	100%	
6	Cắt ảnh theo kích thước (0.5)	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] [Image Processing Algorithms Part 4: Brightness Adjustment](#), Accessed date: 17/07/2024.
- [2] [Image Processing Algorithms Part 5: Contrast Adjustment](#), Accessed date: 17/07/2024.
- [3] [NumPy: numpy.fliplr](#), Accessed date: 17/07/2024.
- [4] [NumPy: numpy.flipud](#), Accessed date: 17/07/2024.
- [5] [Grayscale to RGB Conversion](#), Accessed date: 18/07/2024.
- [6] [Mahotas: RGB to Sepia](#), Accessed date: 18/07/2024.
- [7] [Gaussian Blur](#), Accessed date: 20/07/2024.
- [8] [Kernel \(Image Processing\)](#), Accessed date: 20/07/2024.
- [9] [Rotation matrix](#), Accessed date: 20/07/2024.
- [10] [Zoom in and zoom out](#), Accessed date: 20/07/2024.