

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Báo cáo thực hành Image Processing

Môn học: Toán ứng dụng và thống kê cho CNTT

MTH00057-23CLC04

Sinh viên thực hiện
Nguyễn Trần Thiên An

Giảng viên hướng dẫn:
Nguyễn Đình Thúc
Trần Hà Sơn
Nguyễn Ngọc Toàn
Nguyễn Văn Quang Huy

Ngày 30 tháng 7 năm 2025

Mục lục

1 Thông tin cá nhân và đồ án	2
1.1 Thông tin cá nhân	2
1.2 Giới thiệu đồ án	2
2 Ý tưởng thực hiện	2
2.1 Input/Output	2
2.2 Mục tiêu chính	2
2.3 Ý tưởng giải quyết	3
3 Chi tiết thực hiện	3
3.1 Cấu trúc	3
3.2 Mô tả	4
3.2.1 Thay đổi độ sáng	4
3.2.2 Thay đổi độ tương phản	4
3.2.3 Lật ảnh	5
3.2.4 Tạo ảnh xám	5
3.2.5 Tạo ảnh sepia	6
3.2.6 Làm mờ ảnh	7
3.2.7 Làm nét ảnh	8
3.2.8 Cắt ảnh 1/4 tính từ trung tâm	9
3.2.9 Cắt ảnh theo khung hình tròn	10
3.2.10 Cắt ảnh theo khung 2 hình elip chéo nhau	10
3.2.11 Hàm thể hiện các chức năng	16
4 Kết quả thực nghiệm và kết luận	16
4.1 Cấu hình máy tính	16
4.2 Thực nghiệm	17
4.2.1 Thay đổi độ sáng	18
4.2.2 Thay đổi độ tương phản	18
4.2.3 Lật hình ảnh dọc/ngang	19
4.2.4 Chuyển đổi hình ảnh sang ảnh xám	20
4.2.5 Chuyển đổi hình ảnh sang ảnh sepia	20
4.2.6 Làm mờ ảnh	21
4.2.7 Làm sắc nét ảnh	21
4.2.8 Cắt ảnh từ trung tâm 1/4 và Cắt ảnh theo khung tròn	23
4.2.9 Cắt ảnh theo khung 2 hình ellipse chéo nhau	23
5 Tài liệu	25
6 Acknowledgement	26

1 Thông tin cá nhân và đồ án

1.1 Thông tin cá nhân

Lớp	MSSV	Tên
23CLC04	23127315	Nguyễn Trần Thiên An

Bảng 1: Thông tin của cá nhân

1.2 Giới thiệu đồ án

Đồ án này tập trung vào việc triển khai các chức năng xử lý ảnh cơ bản bằng thư viện NumPy, bao gồm tăng độ sáng, điều chỉnh độ tương phản, lật ảnh, chuyển sang thang độ xám, áp dụng màu sepia, làm mờ, làm nét, và cắt ảnh theo các hình dạng (vuông, tròn, và ellipse). Mục tiêu chính là hiểu rõ cách thức hoạt động của từng phép biến đổi, lý do sử dụng các công thức cụ thể, và so sánh với các phương pháp khác để làm nổi bật ưu điểm của cách tiếp cận này. Các hàm được viết để minh họa rõ ràng các thuật toán, sử dụng NumPy để thao tác trên mảng ảnh mà không phụ thuộc vào các thư viện xử lý ảnh nâng cao như OpenCV hay Pillow.

2 Ý tưởng thực hiện

2.1 Input/Output

Input: Ảnh đầu vào dưới dạng mảng NumPy 3 chiều (chiều cao, chiều rộng, 3 kênh màu RGB).

Output: Ảnh đã xử lý tương ứng với lựa chọn của người dùng (ví dụ: ảnh sáng hơn, ảnh được lật, ảnh màu sepia, v.v.).

2.2 Mục tiêu chính

- Thực hiện các chức năng xử lý ảnh cơ bản bằng NumPy.
- Hiểu rõ cách thức hoạt động của từng phép biến đổi và lý do sử dụng các công thức cụ thể.
- So sánh các phương pháp được sử dụng với các cách tiếp cận khác để làm nổi bật ưu điểm.

2.3 Ý tưởng giải quyết

Mục tiêu là sử dụng NumPy để thao tác trên mảng ảnh, triển khai từng chức năng theo các công thức hoặc phương pháp tiêu chuẩn trong xử lý ảnh:

- **Tăng độ sáng:** Cộng một hằng số (α) cho mỗi pixel.
- **Điều chỉnh độ tương phản:** Sử dụng công thức: $res = mean + \alpha \times (img - mean)$ để tăng/giảm độ tương phản mà không làm ảnh hưởng độ sáng.
- **Lật ảnh:** Đảo ngược thứ tự phần tử (`axis` cho trước) để lật ảnh theo chiều dọc hoặc ngang.
- **Thang độ xám:** Tính trung bình các kênh màu RGB.
- **Màu sepia:** Nhân mảng pixel với ma trận biến đổi sepia tiêu chuẩn.
- **Làm mờ:** Áp dụng bộ lọc hộp (box blur) với kernel $\frac{1}{9} \times np.ones((3, 3))$.
- **Làm nét:** Sử dụng kernel làm nét tiêu chuẩn.
- **Cắt ảnh:** Tạo mask hình vuông, tròn, hoặc ellipse để giữ lại phần ảnh mong muốn.

3 Chi tiết thực hiện

3.1 Cấu trúc

Mã nguồn bao gồm các hàm riêng biệt cho từng chức năng xử lý ảnh và một hàm chính `process_image` để gọi hàm tương ứng dựa trên lựa chọn của người dùng.

STT	Hàm	Chức năng
1	<code>read_img()</code>	Đọc ảnh và chuyển thành mảng NumPy dạng RGB.
2	<code>show_img()</code>	Hiển thị ảnh bằng thư viện matplotlib.
3	<code>save_img()</code>	Lưu ảnh dưới định dạng từ ảnh gốc.
4	<code>increase_brightness()</code>	Tinh chỉnh độ sáng của hình ảnh.
5	<code>contrast()</code>	Tinh chỉnh độ tương phản của hình ảnh.
6	<code>flip_img()</code>	Lật ảnh ngang/dọc.
7	<code>grayscale()</code>	Tạo hiệu ứng xám cho ảnh.
8	<code>sepia()</code>	Tạo hiệu ứng màu sepia cho ảnh.
9	<code>blur_img()</code>	Làm mờ ảnh bằng Convolution.
10	<code>sharpen_img()</code>	Làm sắc nét ảnh bằng Convolution.
11	<code>crop_img_a_quarter()</code>	Cắt ảnh 1/4 từ trung tâm.
12	<code>crop_img_circle()</code>	Tạo khung hình tròn bên ngoài màu đen.
13	<code>find_formula_ellipses()</code>	Tạo công thức 2 hình ellipse chéo nhau trong hình vuông hoặc chữ nhật.
14	<code>crop_img_ellipse()</code>	Tạo khung 2 hình ellipse chéo nhau cho ảnh.
15	<code>process_image()</code>	Hàm điều phối các chức năng.
16	<code>main()</code>	Gọi hàm <code>process_image()</code> và thực hiện.

Bảng 2: Cấu trúc sơ bộ các hàm

3.2 Mô tả

Dưới đây là chi tiết từng hàm, bao gồm công thức, giải thích công thức và ưu điểm của từng chức năng.

3.2.1 Thay đổi độ sáng

Hàm: `increase_brightness()`

1. **Công thức:** $res = img + \alpha$, sau đó clip giá trị về khoảng [0, 255].
2. **Giải thích công thức:** Bởi vì giá trị của mỗi màu trong pixel có giá trị từ 0 đến 255, biểu thị mức độ sáng – 0 là đen hoàn toàn, 255 là trắng hoàn toàn. Do đó, khi ta cộng thêm một số α vào tất cả các màu của pixel, tức là ta đẩy mọi giá trị màu pixel lên cao hơn (hướng đến 255 - màu trắng sáng), khiến toàn bộ ảnh trở nên sáng hơn. Phương pháp này đơn giản, trực quan, và tăng độ sáng đều trên toàn ảnh. Việc chuyển sang kiểu `np.int16` trước khi cộng đảm bảo không bị tràn số, và `np.clip` giữ giá trị pixel trong phạm vi hợp lệ (0 – 255).
3. **Ưu điểm:** Nhanh và dễ triển khai khi chỉ thực hiện 1 phép $+ \alpha$.

3.2.2 Thay đổi độ tương phản

Hàm: `contrast()`

1. **Công thức:** $res = mean + \alpha \times (img - mean)$.
 - $mean$: giá trị trung bình của ảnh theo từng kênh màu.
 - img : ảnh gốc chứa giá trị các điểm ảnh.
 - $img - mean$: độ lệch của các điểm ảnh so với mức trung bình.
 - α : Hệ số tăng cường độ tương phản, thường có giá trị ≥ 0 . Khi $\alpha > 1$ thì độ tương phản được tăng lên; khi $\alpha < 1$ thì độ tương phản giảm. Nếu $\alpha = 1$ thì ảnh giữ nguyên độ tương phản ban đầu.
2. **Giải thích công thức:**
 - Ta lấy giá trị điểm ảnh so sánh với giá trị trung bình ($mean$) để tìm độ lệch, tức là xác định điểm ảnh đó sáng hơn hay tối hơn mức trung bình bao nhiêu.
 - Sau đó, ta nhân độ lệch này với α để phóng đại ($\text{tăng } \alpha > 1$) hoặc thu nhỏ ($\alpha < 1$) sự khác biệt này, tức tăng hoặc giảm khoảng cách tới $mean$.
 - Cuối cùng, ta cộng kết quả vừa tính với $mean$ để thu được giá trị điểm ảnh mới (res), giúp các điểm ảnh phân bố xung quanh $mean$, từ đó đảm bảo tổng thể độ sáng trung bình của ảnh không thay đổi.
3. **Ưu điểm:** Công thức này giữ nguyên giá trị trung bình ($mean$) của ảnh, đảm bảo độ sáng tổng thể không thay đổi khi điều chỉnh độ tương phản. Nếu chỉ sử dụng công thức $res = \alpha \times img$ sẽ làm cho ảnh có thể bị tăng/giảm độ sáng không theo ý muốn, dẫn đến làm giảm hiệu năng tăng giảm độ tương phản. Ví dụ ở [4.2.2 \(QUAN TRỌNG\)](#) sẽ giúp thầy hiểu rõ sự khác biệt giữa 2 công thức.

3.2.3 Lật ảnh

Hàm: `flip_img()`

1. **Công thức:** Tận dụng hàm `np.flip()` có sẵn để hoán đổi các cột với nhau/các hàng với nhau.
2. **Ưu điểm:** Phương pháp này trực tiếp và dễ hiểu và nhanh khi hiểu các parameter của hàm `np.flip()` cần truyền. Điều này, giúp số lượng dòng code ít hơn so với việc chạy vòng `for` 2 lần để hoán đổi từng pixel. Tốc độ nhanh hơn nhiều so với việc hoán đổi từng pixel khi duyệt vòng lặp lồng.

3.2.4 Tạo ảnh xám

Hàm: `grayscale()`

1. **Công thức:**

- **Phương pháp 1:** Tính trung bình giá trị 3 màu R, G, B

$$\text{gray}[i, j] = \frac{\overbrace{\text{img}[i, j, 0]}^R + \overbrace{\text{img}[i, j, 1]}^G + \overbrace{\text{img}[i, j, 2]}^B}{3}.$$

- **Phương pháp 2:** Dùng công thức Luma: $\text{Gray} = 0.299\text{R} + 0.587\text{G} + 0.114\text{B}$

$$[\text{R}, \text{G}, \text{B}] \times \begin{bmatrix} 0.299 \\ 0.587 \\ 0.114 \end{bmatrix}$$

Cả 2 cách đều dùng `np.float32` để tính toán chính xác, sau đó trả về kiểu `np.uint8`.

2. **Giải thích công thức:**

- **Phương pháp 1:** Với việc tính trung bình của 3 màu, mỗi màu đều góp tỷ lệ $\frac{1}{3} \approx 0.333$ cho việc tính tổng. Đây là điểm khác biệt lớn so với phương pháp 2 (tỷ lệ là khác nhau). Tổng của tất cả, ta có màu xám đại diện cho mức độ sáng của pixel khi sáng nhất - màu trắng (= 255); tối nhất - màu đen (= 0).

- **Phương pháp 2:** $0.299\text{R} + 0.587\text{G} + 0.114\text{B}$

Mỗi giá trị màu được nhân với trọng số tỉ lệ nhạy sáng mắt người:

- Màu đỏ góp $\approx 29.9\%$
- Màu xanh lá góp $\approx 58.7\%$
- Màu xanh dương góp $\approx 11.4\%$

\Rightarrow Tổng tất cả, ta được 1 kênh màu duy nhất là 1 màu xám. Và **vì mắt người nhạy cảm nhất với màu xanh lá**, nên tỷ lệ cao giúp ảnh xám sẽ phản ánh độ sáng thật hơn, tự nhiên hơn, hợp với mắt người (nguồn: [The Human Eye's Response to Light](#)). Cũng như phương pháp 1, màu xám cũng đại diện cho mức độ sáng của pixel khi sáng nhất - màu trắng (= 255); tối nhất - màu đen (= 0). Đồng thời, về mặt lịch sử, công thức này đã là một phần của các tiêu chuẩn truyền hình từ giữa thế kỷ 20, bắt nguồn từ quá trình phát triển các hệ thống truyền hình màu (nguồn: [Wikipedia-Grayscale](#))

Tổng kết: Mặc dù trong quá trình chạy thử nghiệm, thời gian chạy hàm của cả 2 phương pháp đều **không chênh lệch quá lớn** nhưng với lý do phương pháp 2 được sử dụng rộng rãi trong grayscale và hợp mắt người hơn khi màu xanh lá chiếm tỷ lệ cao. Em xin phép được chọn sử dụng phương pháp 2 làm phương pháp chuyển đổi ảnh xám trong đồ án của em.

Lưu ý: Vì ảnh ban đầu có 3 kênh màu (R, G, B) nhưng sau khi chạy thuật toán kết quả chỉ còn lại 1 kênh màu (Gray) nên trong hàm `show_img()`, cần thêm tính năng kiểm tra số lượng kênh màu của tham số ảnh truyền vào. Từ đó, thêm '`cmap='gray'`' để hiển thị được màu xám theo mong muốn.

3. **Ưu điểm:** Sử dụng phương pháp nào cũng đơn giản và dễ triển khai, nhưng phương pháp 2 được ưa chuộng hơn vì độ sử dụng rộng rãi trong lịch sử và cũng vì ý nghĩa mặt sinh học khi mắt người nhạy cảm với màu có tính *xanh lá* nhiều hơn.

3.2.5 Tạo ảnh sepia

Hàm: `sepiascale()`

1. **Công thức:** Nhân từng pixel với ma trận sepia:

$$\begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

2. **Giải thích công thức:** Hiệu ứng màu sepia có một tông màu ám, ngả vàng nâu, gợi nhớ phong cách ảnh cổ điển.

- Tăng cường kênh đỏ (Red) với các trọng số cao nhất (so với các dòng khác của ma trận) ở đó (0.393, 0.769, 0.189) để tạo sắc đỏ vàng đặc trưng.
- Dựa vào phần cân bằng màu vàng nâu bằng cách phối trộn vừa phải kênh xanh lá (Green) với các hệ số như 0.349 và 0.686.
- Giảm bớt độ xanh dương (Blue) bằng các hệ số thấp hơn (0.272, 0.534, 0.131) làm ảnh trầm và ít xanh hơn.
- Ví dụ:

- Kênh màu ban đầu: $[R, G, B] = [67, 82, 87]$
- Sau khi nhân: $[R, G, B] = [105.832, 94.251, 73.409004]$

⇒ Mặc dù ban đầu về tỉ lệ giá trị thì màu đỏ nhỏ nhất ($R = 67$), sau đó là màu xanh lá ($G = 82$), lớn nhất là màu xanh dương ($B = 87$). Nhưng sau khi chuyển đổi sang màu sepia (sau khi nhân ma trận) tỷ lệ màu đã thay đổi với sự vượt trội giá trị của màu đỏ sau đó tới màu xanh lá và nhỏ nhất hiện tại là màu xanh dương ($R = 105.832, G = 94.251, B = 73.409004$).

3. **Ưu điểm:** Đảm bảo kết quả nhất quán và đúng với hiệu ứng sepia truyền thống.

3.2.6 Làm mờ ảnh

Hàm: `blur_img()`

- Công thức: Sử dụng kernel box blur:

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Và sử dụng công thức Matrix Convolution (nguồn: [Convolution-Vicmazet](#)):

$$f(x, y) = (g * h)(x, y) = \sum_m \sum_n g(x - m, y - n) h(m, n)$$

để nhân từng **vùng ảnh** từ ảnh gốc với kernel box blur.

- Giải thích công thức: Để hiểu rõ kỹ thuật làm mờ ảnh, cần giải thích rõ về việc công thức Convolution hoạt động (nguồn hình ảnh: [Convolution-Vicmazet](#)).

$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$
$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$	$f_{2,5}$
$f_{3,1}$	$f_{3,2}$	$f_{3,3}$	$f_{3,4}$	$f_{3,5}$
$f_{4,1}$	$f_{4,2}$	$f_{4,3}$	$f_{4,4}$	$f_{4,5}$
$f_{5,1}$	$f_{5,2}$	$f_{5,3}$	$f_{5,4}$	$f_{5,5}$

=

$g_{1,1}$	$g_{1,2}$	$g_{1,3}$	$g_{1,4}$	$g_{1,5}$
$g_{2,1}$	$g_{2,2}$	$g_{2,3}$	$g_{2,4}$	$g_{2,5}$
$g_{3,1}$	$g_{3,2}$	$g_{3,3}$	$g_{3,4}$	$g_{3,5}$
$g_{4,1}$	$g_{4,2}$	$g_{4,3}$	$g_{4,4}$	$g_{4,5}$
$g_{5,1}$	$g_{5,2}$	$g_{5,3}$	$g_{5,4}$	$g_{5,5}$

*

-1	0	$+1$
$h_{-, -}$	$h_{-, 0}$	$h_{-, +}$
$h_{0, -}$	$h_{0, 0}$	$h_{0, +}$
$h_{+, -}$	$h_{+, 0}$	$h_{+, +}$

$$f_{2,2} = g_{3,3}h_{-, -} + g_{3,2}h_{-, 0} + g_{3,1}h_{-, +} + g_{2,3}h_{0, -} + g_{2,2}h_{0, 0} + g_{2,1}h_{0, +} + g_{1,3}h_{+, -} + g_{1,2}h_{+, 0} + g_{1,1}h_{+, +}$$

Hình 1: Ví dụ tính toán pixel (2, 2) của f

Xác định vùng lân cận: Pixel (2, 2) trong f tương ứng với vùng 3x3 trong g từ hàng 1 đến 3 và cột 1 đến 3:

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \end{bmatrix}; \quad \text{kernel } h = \begin{bmatrix} h_{-, -} & h_{-, 0} & h_{-, +} \\ h_{0, -} & h_{0, 0} & h_{0, +} \\ h_{+, -} & h_{+, 0} & h_{+, +} \end{bmatrix}$$

Mỗi phần tử trong vùng lân cận của g được nhân với phần tử tương ứng trong h , sau đó tổng hợp lại theo công thức $(g * h)(x, y) = \sum_m \sum_n g(x - m, y - n) h(m, n)$.

Việc lấy các cặp phần tử từ g và h giống như việc lật (flip) ma trận h theo chiều dọc, sau đó lật (flip) theo chiều ngang rồi lấy cặp theo từng vị trí (i, j) . Sau đó tính tổng lại, ta được $f_{2,2}$.

Pixel (2, 2) của f đại diện cho kết quả tích chập tại vị trí trung tâm của vùng 3x3 trong ảnh gốc g , phản ánh ảnh hưởng của các pixel lân cận theo trọng số được định nghĩa bởi kernel h .

Kernel sử dụng: Việc sử dụng kernel box blur nêu trên:

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

cũng thể hiện rõ ý nghĩa thuật toán - tại mỗi pixel, gán giá trị mới là **trung bình giá trị của các pixel xung quanh** (tính **tổng** và sử dụng **phép chia** cho 9 - số lượng pixel trong **vùng ảnh** được xét).

Padding cho ảnh: Vì ở pixel thuộc các dòng đầu/cuối và cột đầu/cuối, chúng ta không có đủ pixel xung quanh để tạo thành **vùng ảnh** cho việc nhân ma trận kernel. Do đó, chúng ta cần padding thêm các cột/dòng cho ảnh. Ở việc làm mờ, tận dụng hàm `np.pad()` của numpy, em chỉnh `mode='constant'`. Trong trường hợp này, các pixel mới được thêm vào biên được đặt thành 0 (giá trị default của `mode='constant'`). Để lý giải cho việc này thì chúng ta cần xem xét:

- Tính chất làm mờ: Mục tiêu của việc làm mờ là giảm nhiễu và làm mịn ảnh bằng cách lấy trung bình hoặc trọng số của các pixel lân cận. Khi padding bằng 0, các pixel ở biên sẽ bị ảnh hưởng bởi giá trị 0 này, **không làm nổi bật lên tính chất của các cạnh padding**.
 - Tốc độ tính toán: Sử dụng giá trị 0 cũng giúp tính toán nhanh hơn.
3. **Ưu điểm:** Đơn giản và dễ hiểu với kernel default (chỉ lấy trung bình các cạnh xung quanh), tạo hiệu ứng làm mờ bằng cách lấy trung bình các pixel lân cận. Nhanh và phù hợp cho các ứng dụng cơ bản, nhưng hiệu ứng làm mờ vẫn chưa đủ mạnh.

3.2.7 Làm nét ảnh

Hàm: `sharpen_img()`

1. **Công thức:** Có 2 loại kernel làm nét em muốn đề cập đến. Sau khi giải thích, em sẽ lựa chọn 1 trong 2 làm kernel default cho hàm.

Kernel loại 1 (nguồn: [Wikipedia-Kernel \(image processing\)](#)):

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

Kernel loại 2 (nguồn: [Ludwig_ImageConvolution](#)):

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

2. **Giải thích công thức:** Về cách Convolution hoạt động cũng như việc **làm mờ ảnh** nên em sẽ không tiếp tục đề cập đến ở phần này.

Kernel sử dụng: Sau quá trình thực nghiệm và so sánh (ở 4.2.7). Kernel loại 2 cho hình ảnh sắc nét hơn (thường là nhu cầu chủ yếu để nhìn rõ hình dạng của các ảnh mờ). Nên em quyết định sử dụng kernel loại 2 để làm default kernel cho hàm `sharpen_img()`. Để giải thích cho việc kernel loại 2 hoạt động tốt hơn, em nghĩ có 2 yếu tố sau:

- Giá trị trung tâm của kernel được tăng lên ($5 < 9$). Khi này, cứ mỗi pixel được xét đến, màu của nó chiếm tỷ trọng cao trong phép tính TỔNG.
- Giá trị xung quanh có thêm giá trị -1 thay cho 0 . Điều này càng làm giảm độ ảnh hưởng của các giá trị điểm màu pixel xung quanh trong phép tính TỔNG.

Padding cho ảnh: Tương tự ở việc **làm mờ ảnh**, chúng ta cần phải padding cho ảnh. Ở việc làm sắc nét, tận dụng hàm `np.pad()` của numpy, em chỉnh `mode='edge'`. Trong trường hợp này, `mode='edge'` lấp đầy vùng padding bằng cách lặp lại giá trị của pixel biên nhất gần nhất. Để lý giải cho việc sử dụng này thì chúng ta cần xem xét:

- Bảo toàn thông tin biên: Mục tiêu của phép làm sắc nét là tăng cường các cạnh và chi tiết trong ảnh bằng cách tăng độ tương phản. Nếu sử dụng padding 'constant' với giá trị 0 , các pixel biên có **xu hướng kéo về 0** , có thể làm mất đi thông tin biên tự nhiên. Mode 'edge' giúp hình ảnh của em giữ nguyên giá trị biên, **đảm bảo rằng các cạnh thực sự trong ảnh được tăng cường một cách chính xác**.

3. **Ưu điểm:** Kernel này **tăng cường giá trị pixel trung tâm** (giá trị 9) và **giảm giá trị** các pixel **xung quanh** (các giá trị -1) tạo hiệu ứng làm nét. Phương pháp này đơn giản và hiệu quả cho các trường hợp cơ bản, hoạt động tốt cho việc làm sắc nét rõ vật thể. (xem tại 4.2.7)

3.2.8 Cắt ảnh 1/4 tính từ trung tâm

Hàm: `crop_img_a_quarter()`

1. **Công thức:** Cho ảnh đầu vào có chiều cao m và chiều rộng n . Kích thước vùng cắt được xác định bởi:

$$\text{input_size} = \left(\left\lfloor \frac{m}{2} \right\rfloor, \left\lfloor \frac{n}{2} \right\rfloor \right)$$

Tâm ảnh được tính:

$$\text{center} = \left(\frac{m - 1}{2}, \frac{n - 1}{2} \right)$$

Điểm bắt đầu cắt (tọa độ góc trên bên trái của vùng cắt) là:

$$\text{idx} = \left(\text{center}_x - \frac{\text{input_size}_0}{2}, \text{center}_y - \frac{\text{input_size}_1}{2} \right)$$

Kết quả là cropped - ma trận con của img, lấy từ dòng i_0 đến $i_0 + s_0$, và từ cột i_1 đến $i_1 + s_1$.

Trong đó: $i_0 = \text{idx}_0$, $s_0 = \text{input_size}_0$, $i_1 = \text{idx}_1$, $s_1 = \text{input_size}_1$

2. Giải thích công thức:

- Hàm lấy một vùng hình chữ nhật / hình vuông (tuỳ vào kích thước ảnh gốc) ở vị trí giữa ảnh gốc, với chiều cao và chiều rộng bằng một nửa chiều cao và chiều rộng ảnh gốc.
- Tâm ảnh được xác định để biết trung điểm ảnh sao cho vùng cắt được căn giữa chuẩn xác.
- Điểm bắt đầu cắt được tính bằng cách trừ đi nửa kích thước vùng cắt khỏi tâm.
- Cuối cùng, slicing (cắt mảng) được sử dụng để lấy vùng ảnh theo tọa độ tính được.

Note: Kích thước vùng cắt bằng một phần tư diện tích ảnh gốc (bởi vì $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$), giúp giảm kích thước ảnh nhưng vẫn giữ lại vùng trung tâm quan trọng.

3. **Ưu điểm:** Chỉ cần tìm điểm pixel góc trên trái của vùng crop (nhờ tâm ảnh) và tận dụng khả năng slicing của Python. Phương pháp này đơn giản, trực quan, dễ hiểu, và dễ dàng thực hiện bằng numpy slicing hiệu quả. Giữ được phần trung tâm và có thể dễ dàng mở rộng để thay đổi tỉ lệ crop bằng cách sử dụng tham số `alpha` (`default alpha=0.25`) để điều chỉnh kích thước vùng cắt.

3.2.9 Cắt ảnh theo khung hình tròn

Hàm: `crop_img_circle()`

1. **Công thức:** Giữ lại màu các điểm pixel (x, y) thoả:

$$f(x, y) := (x - \text{center}_x)^2 + (y - \text{center}_y)^2 \leq r^2 \quad (1)$$

Với tâm và bán kính được tính như sau:

$$\begin{cases} \text{center} = (\text{center}_x, \text{center}_y) = \left(\frac{\text{width}-1}{2}, \frac{\text{height}-1}{2}\right) \\ r = \frac{\min(\text{width}, \text{height})}{2} \end{cases} \quad (2)$$

Còn lại, gán giá trị 0 cho từng màu của các pixel ngoài vùng.

2. **Ưu điểm:** Dùng tính chất cơ bản của hình học khi biết tâm và bán kính để xác định hình tròn cần. Từ đó, xác định được vùng cần giữ lại màu là trọng tâm của khung ảnh.

3.2.10 Cắt ảnh theo khung 2 hình elip chéo nhau

Hàm: `crop_img_ellipse()`

1. **Công thức:** Cho trước θ , chỉ giữ lại các điểm ảnh có tọa độ (x, y) thoả:

$$\begin{cases} \sin^2 \theta \geq \frac{x^2}{A^2} + \frac{y^2}{B^2} + \frac{2xy}{AB} \cos \theta \\ \sin^2 \theta \geq \frac{x^2}{A^2} + \frac{y^2}{B^2} - \frac{2xy}{AB} \cos \theta \end{cases}, \quad \frac{\pi}{2} \leq \theta < \pi$$

Còn lại, gán giá trị 0 cho từng màu của các pixel ngoài vùng.

2. Giải thích công thức:

Công thức Lissajous Ellipse

Dựa trên nguồn [Wikipedia-Lissajous Curve](#) ta có công thức: $\begin{cases} x = A \sin(at) \\ y = B \sin(at + \theta) \end{cases}$

Dựa vào chú thích trong [Wikipedia-Lissajous Curve](#), để tạo hình ellipse, ta cần tần số $a = b$ nên đặt $a = b = 2\pi$. Cuối cùng ta được công thức Lissajous Ellipse tổng quát theo x, y :

$$\begin{cases} x = A \sin(2\pi t) \\ y = B \sin(2\pi t + \theta) \end{cases} \quad (3)$$

Biến đổi công thức Lissajous Ellipse thành phương trình biểu diễn theo θ, A, B

Ta có công thức lượng giác sau:

$$\sin X + \sin Y = 2 \sin\left(\frac{X+Y}{2}\right) \cos\left(\frac{X-Y}{2}\right)$$

Với $X = 2\pi t$ và $Y = 2\pi t + \theta$, thay vào ta được:

$$\begin{aligned} \frac{x}{A} + \frac{y}{B} &= 2 \sin\left(\frac{2\pi t + (2\pi t + \theta)}{2}\right) \cos\left(\frac{2\pi t - (2\pi t + \theta)}{2}\right) \\ &= 2 \sin\left(\frac{4\pi t + \theta}{2}\right) \cos\left(\frac{-\theta}{2}\right) \\ &= 2 \sin\left(2\pi t + \frac{\theta}{2}\right) \cos\left(\frac{\theta}{2}\right) \end{aligned}$$

Với $\theta \neq \pi$, suy ra:

$$\sin\left(2\pi t + \frac{\theta}{2}\right) = \frac{\frac{x}{A} + \frac{y}{B}}{2 \cos\left(\frac{\theta}{2}\right)} \quad (4)$$

Tương tự, ta có công thức:

$$\sin X - \sin Y = 2 \cos\left(\frac{X+Y}{2}\right) \sin\left(\frac{X-Y}{2}\right)$$

Với $X = 2\pi t$ và $Y = 2\pi t + \theta$, thay vào ta được:

$$\begin{aligned} \frac{x}{A} - \frac{y}{B} &= 2 \cos\left(\frac{2\pi t + (2\pi t + \theta)}{2}\right) \sin\left(\frac{2\pi t - (2\pi t + \theta)}{2}\right) \\ &= 2 \cos\left(2\pi t + \frac{\theta}{2}\right) \sin\left(-\frac{\theta}{2}\right) \\ &= -2 \cos\left(2\pi t + \frac{\theta}{2}\right) \sin\left(\frac{\theta}{2}\right) \end{aligned}$$

Với $\theta \neq 0$, suy ra:

$$\cos\left(2\pi t + \frac{\theta}{2}\right) = -\frac{\frac{x}{A} - \frac{y}{B}}{2 \sin\left(\frac{\theta}{2}\right)} \quad (5)$$

Từ (4) và (5), với $\theta \neq 0, \pi$ áp dụng đẳng thức lượng giác $\cos^2 \alpha + \sin^2 \alpha = 1$, ta có:

$$\begin{aligned} \cos^2\left(2\pi t + \frac{\theta}{2}\right) + \sin^2\left(2\pi t + \frac{\theta}{2}\right) &= \left(\frac{\frac{x}{A} - \frac{y}{B}}{-2 \sin\left(\frac{\theta}{2}\right)}\right)^2 + \left(\frac{\frac{x}{A} + \frac{y}{B}}{2 \cos\left(\frac{\theta}{2}\right)}\right)^2 \\ \Leftrightarrow 1 &= \frac{\left(\frac{x}{A} - \frac{y}{B}\right)^2}{4 \sin^2\left(\frac{\theta}{2}\right)} + \frac{\left(\frac{x}{A} + \frac{y}{B}\right)^2}{4 \cos^2\left(\frac{\theta}{2}\right)} \end{aligned}$$

Đặt chung mẫu số:

$$1 = \frac{\left(\frac{x}{A} - \frac{y}{B}\right)^2 \cos^2\left(\frac{\theta}{2}\right) + \left(\frac{x}{A} + \frac{y}{B}\right)^2 \sin^2\left(\frac{\theta}{2}\right)}{4 \cos^2\left(\frac{\theta}{2}\right) \sin^2\left(\frac{\theta}{2}\right)}$$

Nhân cả hai vế với $4 \cos^2\left(\frac{\theta}{2}\right) \sin^2\left(\frac{\theta}{2}\right)$:

$$4 \cos^2\left(\frac{\theta}{2}\right) \sin^2\left(\frac{\theta}{2}\right) = \left(\frac{x}{A} - \frac{y}{B}\right)^2 \cos^2\left(\frac{\theta}{2}\right) + \left(\frac{x}{A} + \frac{y}{B}\right)^2 \sin^2\left(\frac{\theta}{2}\right)$$

Vì $4 \cos^2\left(\frac{\theta}{2}\right) \sin^2\left(\frac{\theta}{2}\right) = \sin^2 \theta$, ta được:

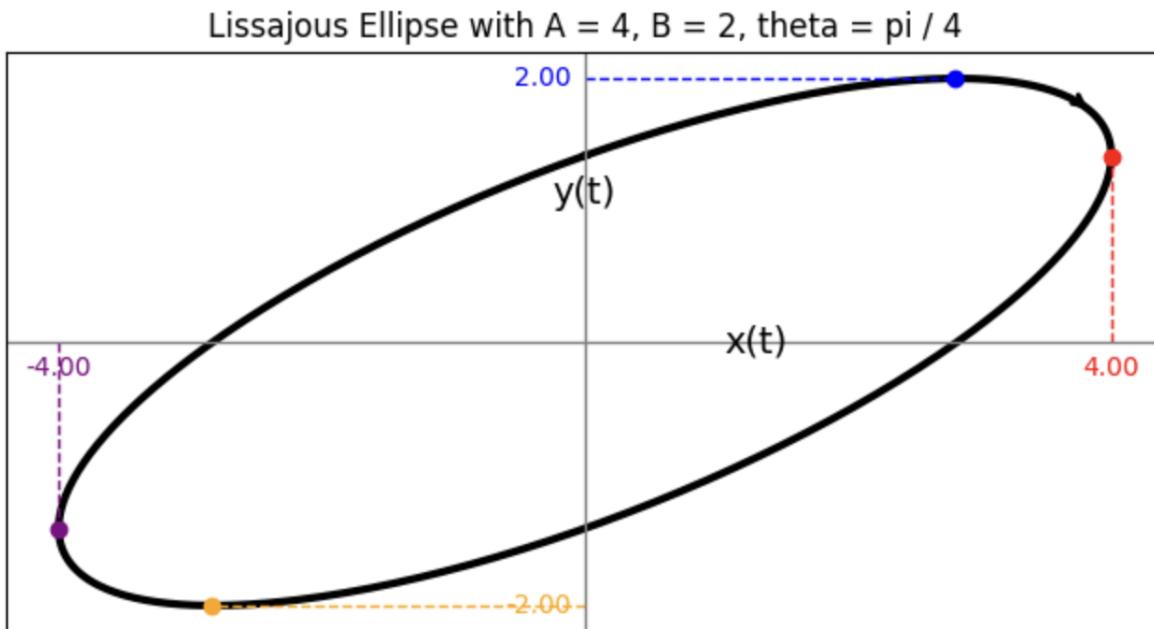
$$\begin{aligned} \sin^2 \theta &= \left(\left(\frac{x}{A}\right)^2 - \frac{2xy}{AB} + \left(\frac{y}{B}\right)^2\right) \cos^2\left(\frac{\theta}{2}\right) + \left(\left(\frac{x}{A}\right)^2 + \frac{2xy}{AB} + \left(\frac{y}{B}\right)^2\right) \sin^2\left(\frac{\theta}{2}\right) \\ &= \left[\cos^2\left(\frac{\theta}{2}\right) + \sin^2\left(\frac{\theta}{2}\right)\right] \left(\frac{x^2}{A^2} + \frac{y^2}{B^2}\right) - \frac{2xy}{AB} \left[\cos^2\left(\frac{\theta}{2}\right) - \sin^2\left(\frac{\theta}{2}\right)\right] \\ &= \frac{x^2}{A^2} + \frac{y^2}{B^2} - \frac{2xy}{AB} \cos \theta \end{aligned}$$

Vậy, phương trình Lissajous Ellipse biểu diễn theo θ, A, B :

$$\sin^2 \theta = \frac{x^2}{A^2} + \frac{y^2}{B^2} - \frac{2xy}{AB} \cos \theta \quad (6)$$

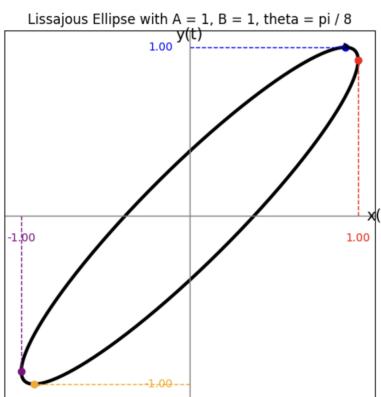
Minh họa hình ảnh đồ thị từ công thức Lissajous Ellipse theo θ, A, B

Ta sẽ thử vẽ 1 hình Lissajous Ellipse theo biểu thức (4) với $A = 4, B = 2, \theta = \frac{\pi}{4}$:

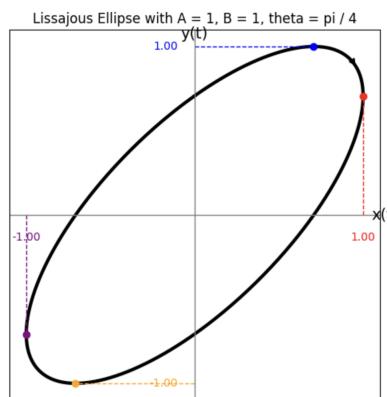


Hình 2: Lissajous Ellipse với $A = 4, B = 2, \theta = \frac{\pi}{4}$

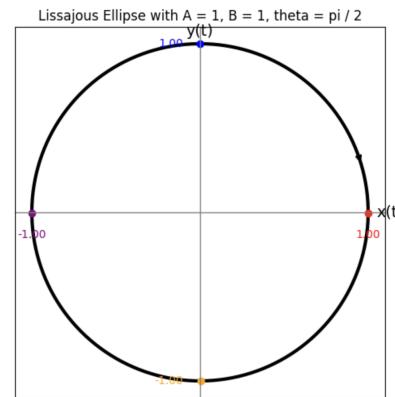
Khi này với $A = 4$, giá trị lớn nhất mà x có thể đạt được là $x = 4$. Tương tự, với $B = 2$, giá trị lớn nhất mà y có thể đạt được là $y = 2$. Do đó, vì giá trị của 2 hàm sin, cos đều thuộc $[-1, 1]$ nên với giá trị $A = 4, B = 2$ thì Lissajous Ellipse sẽ bị ảnh hưởng, tức hình Ellipse đang nội tiếp hình chữ nhật có chiều dài và rộng lần lượt là $a = 4$, $b = 2$. Vậy để Lissajous Ellipse nội tiếp hình vuông (cạnh đều bằng nhau) thì ta cần $A = B$, và với ảnh 512×512 thì cần $A = B = 512$. Nhưng để đơn giản hóa ta chuẩn hóa $A = B = 1$:



Hình 3: $A = 1, B = 1, \theta = \frac{\pi}{8}$



Hình 4: $A = 1, B = 1, \theta = \frac{\pi}{4}$



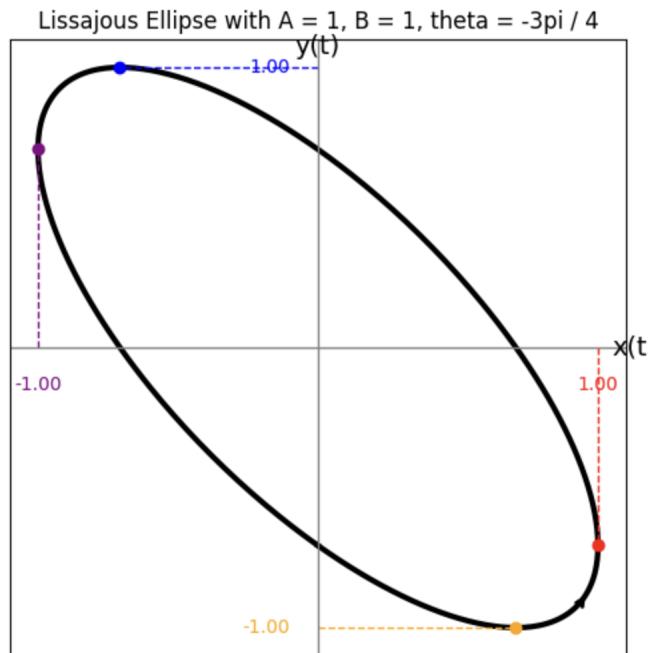
Hình 5: $A = 1, B = 1, \theta = \frac{\pi}{2}$

Hình 6: Lissajous ellipse nội tiếp hình vuông 2×2 với θ biến thiên

Ta có thể thấy hiện tại hình Ellipse đã nội tiếp hình vuông có cạnh $a = 1$. Nhưng khi này với θ thay đổi từ $\frac{\pi}{8} \rightarrow \frac{\pi}{4} \rightarrow \frac{\pi}{2}$. Hình dạng Lissajous Ellipse cũng đã thay đổi.

- Với $\frac{\pi}{8}$ thì Ellipse hẹp hơn tức chỉ số eccentricity $e = \sqrt{1 - \frac{b^2}{a^2}}$ của hình Ellipse còn nhỏ. (theo [Wiki-Ellipse](#)).
- Với $\frac{\pi}{4}$ thì Ellipse trở nên tròn hơn tức e cao hơn so với $\frac{\pi}{8}$.
- Với $\frac{\pi}{2}$ thì Ellipse trở thành hình tròn tức tiêu cự $c = 0$ dẫn tới $e = 0 \rightarrow b = a$.

Mặt khác, khi thay đổi $\theta = -\frac{3\pi}{4}$:



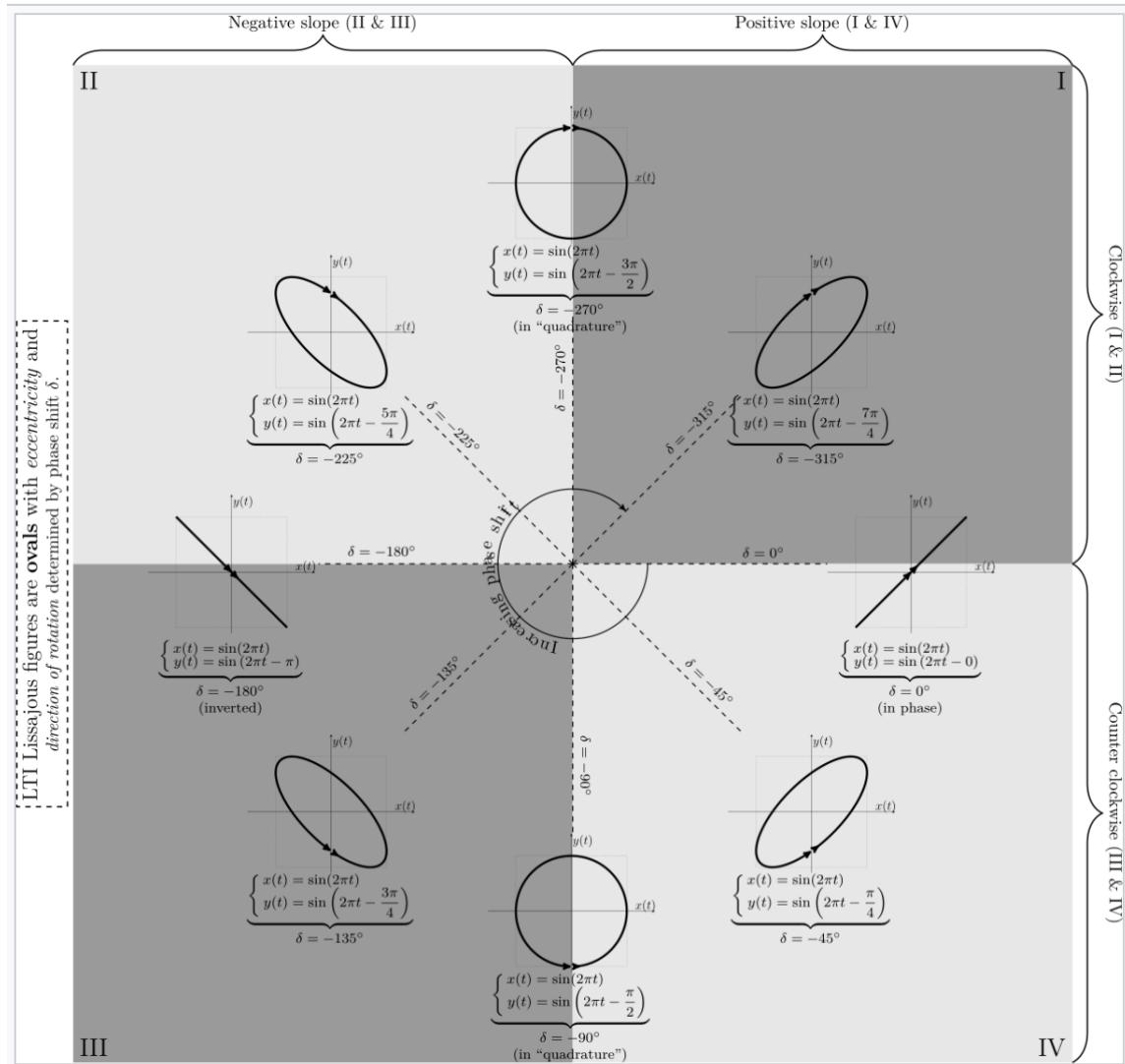
Hình 7: Lissajous Ellipse với $A = 1, B = 1, \theta = -\frac{3\pi}{4}$

Chiều đã thay đổi, như vậy chúng ta chỉ cần thay đổi giá trị θ trong 1 khoảng hợp lý (ánh xạ về miền giá trị từ $[0, 1]$) với 2 bất phương trình Lissajous Ellipse:

$$\begin{cases} \sin^2 \theta \geq \frac{x^2}{A^2} + \frac{y^2}{B^2} + \frac{2xy}{AB} \cos \theta \\ \sin^2 \theta \geq \frac{x^2}{A^2} + \frac{y^2}{B^2} - \frac{2xy}{AB} \cos \theta \end{cases}, \quad \frac{\pi}{2} \leq \theta < \pi \quad (7)$$

sẽ có thể lấy được khung các điểm ảnh chỉ nằm trong 2 hình Ellipse chéo nhau nội tiếp hình vuông cạnh $a = A = B$ thoả yêu cầu (các điểm ảnh nằm ngoài sẽ được gán giá trị 0 cho từng màu).

Chú thích: Vì giá trị θ thay đổi từ $[-2\pi, 2\pi]$ sẽ dẫn đến quay nhiều chiều Ellipse (như hình sau, nguồn: [Wiki-Lissajous curve](#)). Nên ta chỉ cần xét 1 miền xác định cho θ , ở đây là $[\frac{\pi}{2}, \pi]$.

Hình 8: $a = b = 2\pi$ và sự thay đổi θ ảnh hưởng hình dạng Lissajous Ellipse

Ánh xạ θ từ miền xác định $[0, 1]$ về $[\frac{\pi}{2}, \pi]$

Với $f : [0, 1] \rightarrow [\frac{\pi}{2}, \pi]$, sử dụng công thức:

$$\theta = f(\alpha) = \alpha \times \frac{\pi}{2} + \frac{\pi}{2}, \quad \forall \alpha \in [0, 1]$$

3. Ưu điểm: Cho phép vẽ hình Ellipse ảnh theo các hình dạng khác nhau, từ hẹp tối tròn chỉ bằng việc thay đổi θ khi đã biết A, B dựa trên độ dài hình ảnh (bất kể hình vuông hay hình chữ nhật).

3.2.11 Hàm thể hiện các chức năng

Hàm: process_image()

- 0: Thực hiện tất cả các xử lý: sáng, tương phản, lật, xám, sepia, mờ, nét, crop giữa, crop tròn/ellipse
- 1: Tăng/giảm độ sáng
- 2: Tăng/giảm độ tương phản
- 3: Lật ảnh dọc và ngang
- 4: Chuyển ảnh sang xám và sepia
- 5: Làm mờ và làm nét ảnh
- 6: Cắt vùng trung tâm của ảnh
- 7: Cắt ảnh theo hình tròn và hình ellipse

4 Kết quả thực nghiệm và kết luận

4.1 Cấu hình máy tính

Thông tin máy tính sử dụng cho thử.

- Loại: Laptop
- Tên: MacBook Pro (13-inch, M1, 2020)
- Chip: Apple M1 chip
 - 8-core CPU with 4 performance cores and 4 efficiency cores
 - 8-core GPU
 - 16-core Neural Engine
- RAM: 16GB

4.2 Thực nghiệm

Thời gian chạy của riêng thuật toán **K-Means**, hàm `kmeans()`:

STT	Loại xử lý ảnh	Thời gian ước tính
1	Ảnh được thay đổi độ sáng	$\approx 0.00318s.$
2	Ảnh được thay đổi độ tương phản	$\approx 0.0155s.$
3	Lật ảnh (dọc + ngang)	$\approx 0.000152s.$
4	Đổi thành ảnh màu xám (công thức Luma)	$\approx 0.00219s.$
5	Đổi thành ảnh màu sepia	$\approx 0.006397s.$
6	Ảnh được làm mờ	$\approx 3.593s.$
7	Ảnh được làm nét (kernel loại 2)	$\approx 4.3907s.$
8	Ảnh được cắt 1/4 từ trung tâm	$\approx 0.000165s.$
9	Ảnh được cắt khung hình tròn	$\approx 0.00369s.$
10	Ảnh được cắt theo khung 2 ellipse chéo nhau	$\approx 0.00431s.$

Bảng 3: Thời gian chạy các hàm xử lý ảnh, đo bằng thư viện `time` trong Python



Hình 9: Ảnh gốc 512×512

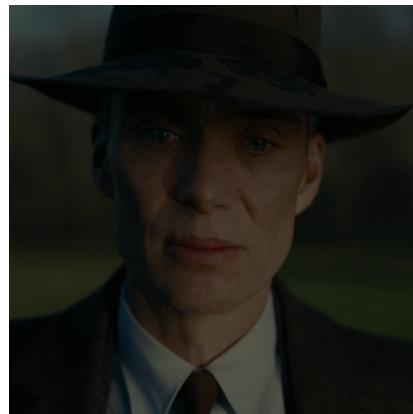
4.2.1 Thay đổi độ sáng

Hình 10: Ảnh giảm độ sáng ($\alpha = -50$)Hình 11: Ảnh tăng độ sáng ($\alpha = 50$)

Hiệu ứng trực quan:

- Giảm độ sáng: nhiều điểm có giá trị màu $\approx 0 \rightarrow$ màu đen. Khiến vài điểm ảnh không còn thấy rõ. Thể hiện công thức giảm độ sáng hoạt động hiệu quả.
- Tăng độ sáng: nhiều điểm có giá trị màu tăng lên. Làm rõ mặt người dưới nón. Do đó, công thức tăng độ sáng đã hoạt động rất tốt.

4.2.2 Thay đổi độ tương phản

Hình 12: Giảm contrast ($\alpha = 0.4$) (dùng mean)Hình 13: Giảm contrast ($\alpha = 0.4$) (chỉ $\times \alpha$)

Giảm độ tương phản: Phương pháp $res = img \times \alpha$ hoàn toàn **KHÔNG TỐT**, khi màu của ảnh bị giảm rất nhiều, làm cho nhiều chi tiết bị mất (nhiều vùng ≈ 0 dần về màu đen). Trong khi đó, phương pháp dùng `mean` cho ảnh đúng với chất lượng sau khi giảm (mờ nhạt, màu không bị đen). Vậy nên **công thức** $res = mean + \alpha \times (img - mean)$ **thể hiện rất tốt trong việc giảm độ tương phản**.

Hình 14: Tăng contrast ($\alpha = 1.8$) (dùng mean)Hình 15: Tăng contrast ($\alpha = 1.8$) (chỉ $\times \alpha$)

So sánh giữa 2 công thức: Với 2 ảnh thực hiện tăng độ tương phản với $\alpha = 1.8$ thì ảnh sử dụng công thức 2 $res = \alpha \times img$ mặc dù phần tạo được hiệu ứng tương phản nhưng vẫn bị **chói độ sáng** ở một vài vị trí → **điều không nên**. Nhưng với công thức 1: $res = mean + \alpha \times (img - mean)$ độ tương phản tăng rõ rệt khi một phần nét mặt ở vùng mắt bên phải được làm tối, giúp hình ảnh có độ tương phản đẹp hơn nhiều so với ảnh gốc. **Công thức 1 vẫn cho độ tương phản tốt hơn.**

4.2.3 Lật hình ảnh dọc/ngang



Hình 16: Ảnh lật dọc



Hình 17: Ảnh lật ngang

Hiệu ứng: Ảnh đều được lật ngang, dọc 1 cách đối xứng lần lượt qua trực ngang và trực dọc. Đồng thời, thuật toán chạy cả 2 hành động lật chỉ tốc $\approx 0.000152s$ là thời gian ngắn nhất chạy tất cả các hàm. Thể hiện tốc độ vượt trội, dễ hiểu và dễ được ứng dụng của thuật toán.

4.2.4 Chuyển đổi hình ảnh sang ảnh xám



Hình 18: Ảnh xám theo phương pháp 1 (mean) Hình 19: Ảnh xám theo phương pháp 2 (Luma)

Hiệu ứng: Bằng mắt thường, ta không thấy rõ sự khác biệt nên cả 2 phương pháp đều hoạt động rất tốt. Tuy nhiên, sau khi **nhìn kỹ**, ở một vài điểm trên nón thì **vết hình trên nón của phương pháp 2 đậm hơn so với phương pháp 1.** \Rightarrow Mặc dù dẫn chứng chưa rõ ràng nhưng có thể nhận xét là **độ tương phản mà phương pháp 2 mang lại là tốt hơn.**

Thời gian: Dùng phương pháp 1 (tính trung bình của 3 màu sử dụng `np.mean()`) tốn ≈ 0.00254 , trong khi dùng phương pháp 2 (công thức Luma) chỉ tốn $\approx 0.00219s$. Cả 2 phương pháp đều **không** chênh lệch nhau đáng kể.

4.2.5 Chuyển đổi hình ảnh sang ảnh sepia



Hình 20: Ảnh gốc 512×512

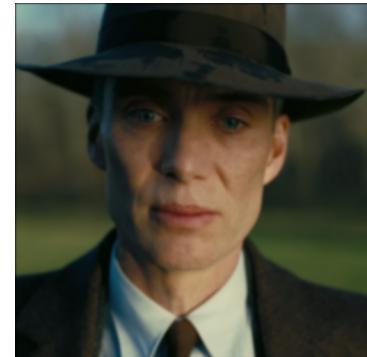
Hình 21: Ảnh chuyển đổi thành ảnh sepia

Màu sắc hình ảnh: Hình ảnh mang tông sepia ấm áp, cổ điển với tốc độ thấp ($\approx 0.006397s$). Thể hiện thuật toán hoạt động rất tốt để mang lại dạng ảnh sepia.

4.2.6 Làm mờ ảnh

Hình 22: Ảnh gốc 512×512 

Hình 23: Ảnh làm mờ

Hình 24: Ảnh làm mờ $\times 4$ lần

Hiệu ứng trực quan:

- Hình ảnh làm mờ chỉ 1 lần cho kết quả mờ **KHÔNG** quá tốt. Nhiều chi tiết vẫn còn nhận thấy rõ được.
- Hình ảnh làm mờ $\times 4$ cho kết quả mờ tốt hơn (chạy hàm làm mờ 4 lần liên tiếp). Nhưng có thể thấy dấu hiệu việc viền của hình ảnh đã trở nên đen hơn, cho thấy giá trị padding bằng 0 ảnh hưởng nhiều đến các điểm biên.

⇒ Việc làm mờ với mức độ nào tuỳ thuộc vào nhu cầu người sử dụng. Nếu muốn làm mờ hơn thì có thể xem xét việc chạy hàm nhiều lần, nhưng đánh đổi về mặt thời gian khi thời gian chạy nhân 4 lần (1 lần chạy: $\approx 3.593s$).

4.2.7 Làm sắc nét ảnh

Hình 25: Làm sắc nét ảnh với *kernel loại 1*Hình 26: Làm sắc nét ảnh với *kernel loại 2*

Mức độ sắc nét: Từ hình, ta thấy Kernel loại 2 (Hình 27) tạo ra **hiệu ứng sắc nét mạnh hơn** so với Kernel loại 1 (đường viền mũi, các điểm trên khuôn mặt và nếp áo sơ mi). Ngược lại, hiệu ứng hình 26 không quá mạnh.

Thời gian: Xem xét về tốc độ chạy của hàm với 2 loại kernel:

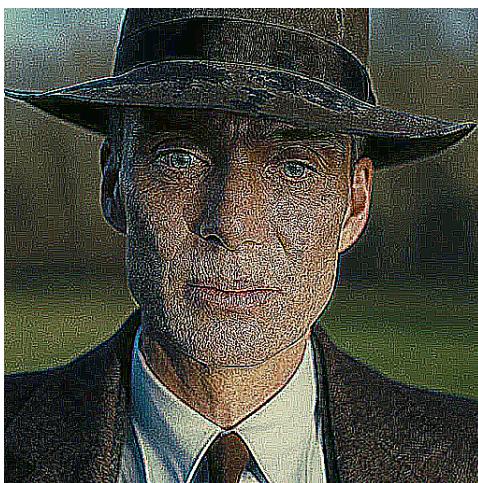
- Thời gian chạy kernel loại 1: $\approx 4.239s$
- Thời gian chạy kernel loại 2: $\approx 4.3907s$

⇒ Kernel loại 1 tỏ ra hiệu quả hơn về mặt tốc độ so với kernel loại 2, có thể do thiết kế đơn giản hơn với các giá trị 0. Tuy nhiên, sự khác biệt thời gian là **rất nhỏ** và có thể **không** phải là yếu tố quyết định khi chọn kernel.

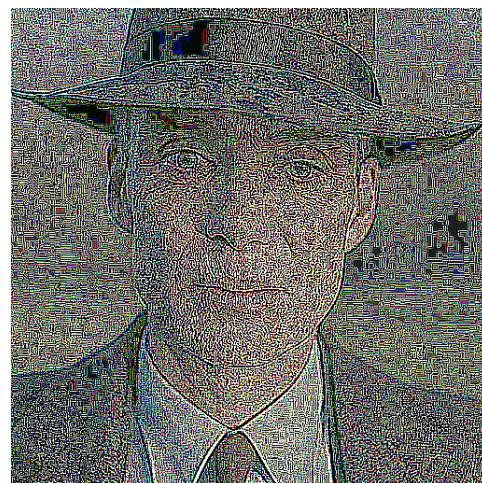
⇒ **Vậy nên việc chọn kernel nào là tùy thuộc vào độ sắc nét mong muốn.**

Chạy nhiều lần để tăng sắc nét: Chọn kernel loại 2 để chạy làm sắc nét trên ảnh đã được làm sắc nét. Trái ngược với *làm mờ ảnh*, khi muốn tăng hiệu ứng sắc nét bằng việc chạy nhiều lần, thì tới lần 2, xuất hiện nhiều đốm trắng ở vài điểm và tại lần 3 hình ảnh bị đốm trắng rất nhiều.

⇒ Thể hiện việc **tích chập nhiều lần** khiến một vài điểm bị **tràn số** rất nhiều, khi đó hình ảnh **không còn tốt** nữa. Vậy nên, theo em, ý tưởng **chạy hàm sắc nét nhiều lần để tăng tính sắc nét là không phù hợp, gây phản tác dụng.**

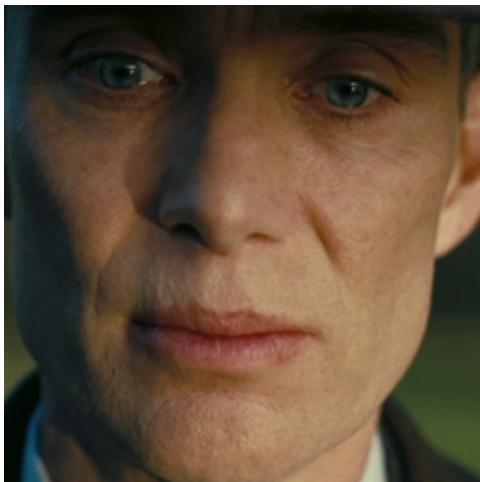


Hình 27: Ảnh được làm sắc nét $\times 2$



Hình 28: Ảnh được làm sắc nét $\times 3$

4.2.8 Cắt ảnh từ trung tâm 1/4 và Cắt ảnh theo khung tròn



Hình 29: Ảnh được cắt 1/4 từ trung tâm



Hình 30: Ảnh cắt theo khung hình tròn

Cắt ảnh theo trung tâm: Hình ảnh tập trung vào vùng trung tâm hơn. Tốc độ chạy cũng không chậm (chỉ $\approx 0.000165s$). Cho thấy công thức hoạt động tốt.

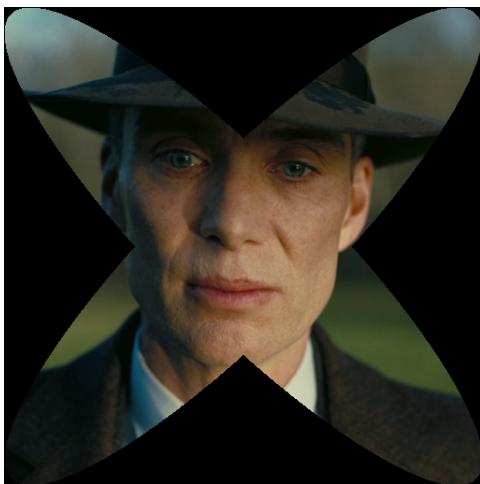
Cắt ảnh theo khung tròn: Vùng tròn chứa ánh sáng, cà vạt, một phần mũ, góc thành đèn, giống hiệu ứng nhìn vào camera. Dù là các điểm pixel thì có thể việc tạo khung tròn KHÔNG thật sự tròn như mong muốn. Nhưng dẫu vậy, công thức vẫn cho hiệu quả tốt (càng tốt khi kích thước ảnh càng lớn) với thời gian thấp ($\approx 0.00369s$) khi chỉ cần ứng dụng tính chất hình tròn.

4.2.9 Cắt ảnh theo khung 2 hình ellipse chéo nhau

Đã chứng minh ở Mục 2, ta có:

$$\theta = \alpha * \frac{\pi}{2} + \frac{\pi}{2}$$

Hình 31: $\alpha = 0.3 \rightarrow \theta = \frac{13\pi}{20}$ Hình 32: $\alpha = 0.5 \rightarrow \theta = \frac{3\pi}{4}$

Hình 33: $\alpha = 0.7 \rightarrow \theta = \frac{17\pi}{20}$ Hình 34: $\alpha = 0.85 \rightarrow \theta = \frac{37\pi}{40}$ Hình 35: Ảnh 1024×512 được cắt với $A = 2, B = 1$ và $\alpha = 0.5 \rightarrow \theta = \frac{3\pi}{4}$

Nhận xét: Chỉ với thời gian chạy rất ngắn ($\approx 0.00431s$ gần với thời gian chạy cắt khung hình tròn $\approx 0.00369s$), việc tạo khung trở nên dễ dàng chỉ bằng công thức toán học mạnh mẽ. Dù là ảnh truyền vào là **Hình Vuông** hay **Hình Chữ Nhật**, ta chỉ cần chỉnh $\alpha \in (0, 1)$, ta sẽ có được khung theo tỷ lệ ellipse nội tiếp theo mong muốn. Như hình:

- $\alpha = 0.3$ cho hình ellipse tròn bự hơn.
- $\alpha = 0.5$ cho hình độ dài tiêu cự trung bình.
- $\alpha = 0.7$ cho hình có tâm sai lớn.
- $\alpha = 0.85$ cho hình có tâm sai rất lớn. Càng về 1 thì hình càng giống đường thẳng.

5 Tài liệu

Thay đổi độ tương phản:

- [1] Pillow, Source code for PIL.ImageEnhance, Contrast, [17/07/2025]

Lật ảnh ngang/dọc:

- [1] Numpy Developers, *numpy.flip* — NumPy v2.3 Manual. [16/07/2025].

Tạo ảnh xám:

- [1] Iowa State University, The Human Eye's Response to Light. [29/07/2025].

- [2] Wikipedia, Grayscale. [20/07/2025].

Tạo ảnh sepia:

- [1] GeeksForGeeks, mage Processing in Java - Colored Image to Sepia Image Conversion, 2025. [21/07/2025].

Làm mờ và làm sắc nét ảnh:

- [1] 3Blue1Brown, But what is a convolution? YouTube, 2020. [22/07/2025].

- [2] 3Blue1Brown, Convolutions in Image Processing | Week 1, lecture 6 | MIT 18.S191 Fall 2020 YouTube, 2020. [22/07/2025].

- [3] Vicmazet, Convolutions. [22/07/2025].

- [4] Wikipedia, Kernel (image processing), [22/07/2025]

- [5] Just A Coder, The Science of Blur Image || Blur From Scratch with YouTube, 2025. [22/07/2025].

Cắt hình ảnh theo khung 2 hình ellipse chéo nhau:

- [1] Wikipedia, Ellipse, [23/07/2025]

- [2] Wikipedia, Lissajous curve, [23/07/2025]

- [3] Ng Chung Tak, Eliminating θ from $a \cos(\theta - \alpha) = x$ and $b \cos(\theta - \beta) = y$ Math Stack Exchange, 2018. [24/07/2025].

- [4] Ng Chung Tak, Mathematics Stack Exchange | Geometry - What is the largest ellipse of given eccentricity that can be inscribed into square? Math Stack Exchange, 2024. [24/07/2025].

- [5] Mr. P Solver, The Python Function You NEED For 2D Data - Numpy Meshgrid YouTube, 2022. [24/07/2025].

6 Acknowledgement

Trước hết, em xin chân thành cảm ơn thầy **Nguyễn Ngọc Toàn** đã tận tình giảng giải thuật toán trên lớp, giúp em hiểu rõ hơn về các chức năng và cách triển khai trong xử lý ảnh, cũng như hỗ trợ cả lớp trong việc lập trình Python tại các buổi Lab. Bên cạnh đó, em đã nhận được sự hỗ trợ từ các công cụ như **ChatGPT** và **Gemini** trong việc tìm kiếm tài liệu liên quan đến thuật toán *cắt khung theo 2 hình ellipse chéo nhau*. Các công cụ này cũng hỗ trợ em giải thích cú pháp Python (như `meshgrid`) thông qua các ví dụ cụ thể khi em gặp khó khăn trong quá trình tham khảo tài liệu, đồng thời giúp em chỉnh sửa câu văn và văn phong trong các phần **2.3** và **4.2** của báo cáo.