

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG – HCM
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN

PROJECT 2: IMAGE PROCESSING

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

Giảng viên:

Vũ Quốc Hoàng

Nguyễn Văn Quang Huy

Nguyễn Ngọc Toàn

Phan Thị Phương Uyên

Sinh viên thực hiện:

Dương Trung Nghĩa

MSSV:

22127293

Lớp:

22CLC05

Thành phố Hồ Chí Minh, năm 2024

MỤC LỤC

MỤC LỤC.....	1
NHẬN XÉT CỦA GIẢNG VIÊN	2
MÔ TẢ BÀI LÀM.....	3
1/ THAY ĐỔI ĐỘ SÁNG	3
2/ THAY ĐỔI ĐỘ TƯƠNG PHẢN	3
3/ LẬT ẢNH NGANG – DỌC.....	4
4.1/ RGB THÀNH ẢNH XÁM.....	5
4.2/ RGB THÀNH ẢNH SEPIA.....	6
6/ CẮT ẢNH THEO KÍCH THƯỚC	10
7.1/ CẮT ẢNH THEO KHUNG TRÒN	11
7.2/ CẮT ẢNH THEO KHUNG ELIP	12
8/ HÀM MAIN.....	14
9/ CÁC HÀM PHỤ TRỢ.....	15
BẢNG ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH	17
TÀI LIỆU THAM KHẢO	22

[illegible]

Giảng viên

MÔ TẢ BÀI LÀM

1/ THAY ĐỔI ĐỘ SÁNG

Ý tưởng thực hiện: [5]

- Bức ảnh sáng hơn khi có nhiều màu sáng hơn, để làm bức ảnh sáng hơn ta cộng một số α vào mỗi phần tử trên mảng màu, càng gần giá trị 255 thì bức ảnh càng sáng.
- Sự thay đổi độ sáng của các điểm ảnh trên ảnh phụ thuộc vào độ lớn của số α .
- Nếu số α âm, ảnh sẽ tối đi và ngược lại số α dương thì ảnh sẽ sáng hơn.

Mô tả hàm `brightness(img, bias)`:

- Chức năng: Điều chỉnh độ sáng của hình ảnh bằng cách thêm “bias” vào mỗi giá trị pixel.
- Đầu vào:
 - “img”: mảng numpy 3D (height, width, channels).
 - “bias”: int - Giá trị điều chỉnh độ sáng. Giá trị dương tăng độ sáng, giá trị âm giảm độ sáng.
- Đầu ra: “res” np.ndarray - Trả về mảng numpy 3D đã được điều chỉnh độ sáng.
- Mô tả: Chuyển đổi mảng ảnh sang kiểu “uint16” bằng hàm `astype()` để tránh tràn số khi cộng bias, sau đó cộng “broadcasting” bias vào mỗi giá trị pixel. Sử dụng hàm `np.clip()` từ “numpy” để đảm bảo các giá trị pixel nằm trong khoảng [0, 255], và cuối cùng chuyển đổi kết quả về kiểu uint8.

2/ THAY ĐỔI ĐỘ TƯƠNG PHẢN

Ý tưởng thực hiện: [5]

- Độ tương phản là sự chênh lệch (gap) giữa sáng và tối, để tăng độ tương phản thì phải tăng gap giữa các điểm ảnh bằng cách nhân vô hướng với một số α vào mỗi phần tử trên mảng màu.
- Độ tương phản phụ thuộc vào độ lớn của α .
- Nếu số $\alpha > 1$ thì độ tương phản tăng còn nếu $\alpha < 1$ thì độ tương phản giảm.

Mô tả hàm `contrast(img, bias)`:

- Chức năng: Điều chỉnh độ tương phản của hình ảnh bằng cách sử dụng “bias”.
- Đầu vào:
 - “img”: mảng numpy 3D (height, width, channels).
 - “bias”: int - Giá trị điều chỉnh độ tương phản. Giá trị dương tăng độ tương phản, giá trị âm giảm độ tương phản.
- Đầu ra: “res” np.ndarray - Trả về mảng numpy 3D đã được điều chỉnh độ tương phản.
- Mô tả: Chuyển đổi mảng ảnh sang kiểu “uint16” bằng hàm astype() để tránh tràn số khi nhân với bias, sau đó nhân “broadcasting” bias với mỗi giá trị pixel. Sử dụng hàm np.clip() từ “numpy” để đảm bảo các giá trị pixel nằm trong khoảng [0, 255], và cuối cùng chuyển đổi kết quả về kiểu “uint8” bằng hàm astype().

3/ LẬT ẢNH NGANG – DỌC

Ý tưởng thực hiện: [5]

- Một ảnh là một mảng có nhiều hàng, nhiều cột để lật ảnh dọc, ta cần đảo ngược vị trí các hàng trong mảng, trong khi để lật ảnh ngang, ta giữ nguyên các hàng chỉ đảo ngược vị trí các cột trong mảng. Ví dụ hình ảnh ở dạng 2D:

1	2	3
4	5	6
7	8	9

Ảnh gốc chưa lật



7	8	9
4	5	6
1	2	3

Ảnh đã lật dọc



3	2	1
6	5	4
9	8	7

Ảnh đã lật ngang

Mô tả hàm flip(img, mode):

- Chức năng: Lật hình ảnh theo chiều dọc hoặc chiều ngang.
- Đầu vào:
 - “img”: mảng numpy 3D (height, width, channels).
 - “mode”: string - Chế độ lật. Có thể là “vertical” cho lật theo chiều dọc hoặc “horizontal” cho lật theo chiều ngang.
- Đầu ra: “res” np.ndarray - Trả về mảng numpy 3D đã được lật.
- Mô tả:
 - Nếu mode là “vertical”, sử dụng slicing `img[::-1]` để lật hình ảnh theo chiều dọc, tất cả các hàng của hình ảnh sẽ được đảo ngược. `[::-1]` slicing với cú pháp “start:stop:step”, trong đó “start” và “stop” bị bỏ qua (tức là lấy toàn bộ mảng), còn step là -1 để đảo ngược thứ tự của các phần tử trong mảng.
 - Nếu mode là “horizontal”, sử dụng slicing `img[:, ::-1]` để lật hình ảnh theo chiều ngang, tất cả các cột của mỗi hàng sẽ được đảo ngược. `[:, ::-1]` slicing với cú pháp “start:stop:step” cho các cột của mỗi hàng, trong đó “start” và “stop” bị bỏ qua (tức là lấy toàn bộ các cột), còn step là -1 để đảo ngược thứ tự của các phần tử trong mỗi hàng.
 - Còn nếu mode không phải “vertical” hoặc “horizontal”, hình ảnh sẽ được lật theo chiều dọc mặc định bằng cách sử dụng slicing `img[::-1]` như trên.

4.1/ RGB THÀNH ẢNH XÁM

Ý tưởng thực hiện:

- Ảnh xám là từ ảnh có 3 kênh màu (R3) chúng ta biến thành 1 kênh màu (R1) tương đương cả ba kênh màu R, G, B có giá trị bằng nhau. Các phương pháp chuyển đổi ảnh RGB sang ảnh xám bao gồm:

1. Gán giá trị từ một kênh màu: [5]

- Gán giá trị $R = G = B$ từ một kênh màu bất kỳ.
- Kết quả có thể bị mất chi tiết do màu quá đậm hoặc quá nhạt

2. Phương pháp trung bình: [3]

- $\text{Grayscale} = (R + G + B) / 3$

- Cho kết quả cân bằng hơn giữa các màu, nhưng có thể tạo ra màu xám quá đậm.

3. Phương pháp trọng số (Luminosity): [3]

- Grayscale = (0.3 * R) + (0.59 * G) + (0.11 * B)
- Được coi là phương pháp tốt nhất, giảm đóng góp của màu đỏ, tăng màu xanh lá cây, và cân bằng màu xanh lam. Cho ra màu xám dịu, rõ nét, và giữ chi tiết tốt hơn.

⇒ Do đó, phương pháp trọng số là phương pháp được thầy/cô khuyến nghị cho hàm chức năng này.

$$\begin{bmatrix} R' & G' & B' \end{bmatrix} = \begin{bmatrix} R & G & B \end{bmatrix} * \begin{bmatrix} 0.3 \\ 0.59 \\ 0.11 \end{bmatrix}$$

Như vậy, để tối ưu cho hàm chức năng này, lấy mảng hình ảnh dot product (@) với mảng trọng số.

Mô tả hàm gray(img):

- Chức năng: Chuyển đổi một hình ảnh sang ảnh xám.
- Đầu vào: “img” - Mảng numpy 3D (height, width, channels).
- Đầu ra: “res” np.ndarray - Trả về mảng numpy 2D (height, width) đã được chuyển sang thang độ xám.
- Mô tả:
 - Tạo mảng trọng số là [0.3, 0.59, 0.11] tương ứng với các kênh Red, Green và Blue.
 - Áp dụng phép nhân 2 ma trận bằng hàm np.dot() giữa mảng ảnh (... , 3) và mảng trọng số (3,) để tính toán giá trị độ xám cho từng pixel.
 - Sử dụng hàm np.clip() từ “numpy” để đảm bảo các giá trị pixel nằm trong khoảng [0, 255], và cuối cùng chuyển đổi kết quả về kiểu “uint8” bằng hàm astype() .

4.2/ RGB THÀNH ẢNH SEPIA

Ý tưởng thực hiện:

- Để tạo ra màu sepia, ta có công thức cố định như sau: [4]

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

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

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

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

Mô tả hàm sepia(img):

- Chức năng: Chuyển đổi một hình ảnh sang màu sepia.
- Đầu vào: “img” - Mảng numpy 3D (height, width, channels).
- Đầu ra: “res” np.ndarray - Trả về mảng numpy 3D (height, width, channels) đã được chuyển sang màu sepia.
- Mô tả hàm sepia(img):

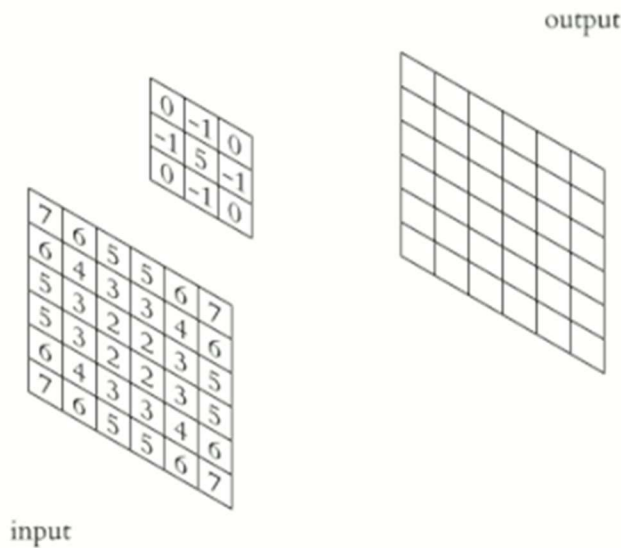
- Sử dụng một ma trận lọc sepia để chuyển đổi các kênh màu RGB của hình ảnh sang màu sepia:

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

- Áp dụng phép nhân ma trận giữa mảng ảnh (... , 3) và ma trận lọc sepia (3,3). Tuy nhiên, theo nguyên tắc nhân 2 ma trận là hàng ma trận này nhân cột ma trận kia, nên ở đây sử dụng “transport” ma trận lọc sepia để có kết quả đúng.
- Sử dụng hàm np.clip() từ numpy để đảm bảo các giá trị pixel nằm trong khoảng [0, 255], và cuối cùng chuyển đổi kết quả về kiểu “uint8” bằng hàm astype().

5/ LÀM MỜ / SẮC NÉT ẢNH

Ý tưởng thực hiện:



- Ý tưởng là duyệt từng điểm ảnh theo chiều cao và độ rộng, lấy mảng con từ vị trí được duyệt tới kích thước mảng hình ảnh – kích thước mảng kernel + 1, nhưng vấn đề là những phần tử ngoài lề không xử lý được. [2]
- Vì thế phải thêm các padding ảo có cùng giá trị với phần tử ngoài lề vào ma trận gốc để các phần tử ngoài lề được xem như là tâm của một ma trận.
- Thay vì sử dụng vòng lặp để duyệt từng điểm ảnh theo chiều cao và độ rộng, điều này mất nhiều thời gian, thì duyệt từng phần tử trong kernel, nhân với mảng đệm rồi cộng gộp vào sẽ nhanh hơn. Ví dụ mảng hình ảnh 2D gốc ta có:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} (\text{mảng gốc}) \rightarrow \begin{bmatrix} 1 & 1 & 2 & 3 & 3 \\ 1 & 1 & 2 & 3 & 3 \\ 4 & 4 & 5 & 6 & 6 \\ 7 & 7 & 8 & 9 & 9 \\ 7 & 7 & 8 & 9 & 9 \end{bmatrix} (\text{sau khi padding}),$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} (\text{mảng kernel})$$

Với phần tử đầu tiên trong kernel là 0, ta có:

$$\rightarrow \begin{bmatrix} 1 * 0 & 1 * 0 & 2 * 0 \\ 1 * 0 & 1 * 0 & 2 * 0 \\ 4 * 0 & 5 * 0 & 6 * 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} (\text{mảng mới})$$

Với phần tử tiếp theo trong kernel là -1, ta có:

$$\rightarrow \begin{bmatrix} 1 * -1 & 2 * -1 & 3 * -1 \\ 1 * -1 & 2 * -1 & 3 * -1 \\ 4 * -1 & 5 * -1 & 6 * -1 \end{bmatrix} \rightarrow \begin{bmatrix} -1 & -2 & -3 \\ -1 & -2 & -3 \\ -4 & -5 & -6 \end{bmatrix} (\text{sau khi cộng gộp})$$

Tương tự như trên cho tới phần tử cuối cùng trong kernel là 0, ta có:

$$\begin{bmatrix} -3 & -1 & 1 \\ 3 & 5 & 7 \\ 9 & 11 & 13 \end{bmatrix}$$

- Đã kiểm tra với cách duyệt từng điểm ảnh và có kết quả tương tự.

Mô tả hàm `kernel_initial(mode)` và `convolution(img,mode)`:

① Hàm `kernel_initial(mode)`:

- Chức năng: Khởi tạo một kernel tích chập để làm mờ hoặc làm nét hình ảnh.
- Đầu vào: “mode” (str) - Loại kernel, có thể là “blur” hoặc “sharpen”. Mặc định là “blur”.
- Đầu ra: “kernel” np.ndarray - Mảng numpy 2D đại diện cho kernel tích chập.
- Mô tả:
 - Nếu mode là “blur”, kernel làm mờ 3x3 bằng hàm `np.array()` với dtype = `np.float32` với trọng số là: [2]

Gaussian blur 3 × 3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$
---	--

- Nếu mode là “sharpen”, kernel làm nét 3x3 bằng hàm `np.array()` tương tự dtype = `np.float32` nhưng trọng số được sử dụng sẽ là: [2]

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

- Nếu mode không phải “blur” hoặc “sharpen”, mặc định sử dụng kernel làm mờ.

② Hàm `convolution(img,mode)`:

- Chức năng: Áp dụng bộ lọc làm mờ/làm nét cho một hình ảnh.
- Đầu vào:
 - “img” (mảng numpy) - Hình ảnh dưới dạng mảng numpy.
 - “mode” (str) - Chế độ tích chập, có thể là “blur” hoặc “sharpen”.

- Đầu ra: “res” np.ndarray - Mảng numpy 3D đã được áp dụng bộ lọc làm mờ/làm nét.
- Mô tả:
 - Sử dụng hàm “kernel_initial” ở trên để khởi tạo kernel dựa trên chế độ được chọn.
 - Lấy kích thước của ảnh bằng hàm shape() trong thư viện “numpy” để duyệt qua từng điểm ảnh. Đồng thời, tạo ra mảng giá trị 0 mới từ mảng “img” với dtype = np.float32 bằng hàm np.zeros_like().
 - Đệm ảnh sử dụng np.pad của thư viện “numpy” với tham số truyền vào là mảng cần đệm (img), một tuple chứa các thông số đệm cho mỗi trục lần lượt là (1,1) đệm 1 hàng trên và 1 hàng dưới (trục height), (1,1) đệm cột trái và cột phải (trục width) và (0,0) có nghĩa không đệm thêm cho trục channels cuối cùng là chế độ đệm “edge” sao chép các giá trị biên của ảnh gốc cho phần đệm, để xử lý các cạnh của ảnh.
 - Đầu tiên, vòng lặp ngoài cùng duyệt qua từng kênh màu của ảnh (channels). Sau đó, hai vòng lặp bên trong duyệt qua các phần tử của kernel. Với mỗi phần tử của kernel, sẽ lấy mảng con từ ảnh đã đệm, bắt đầu từ vị trí tương ứng của điểm ảnh hiện tại đến kích thước mảng “img” và nhân với giá trị của phần tử kernel tương ứng, rồi cộng gộp kết quả vào ma trận kết quả res thông qua các chỉ số thích hợp. Mảng con này được xác định bởi các chỉ số i và j của kernel, cũng như chiều cao và chiều rộng của ảnh.
 - Sử dụng np.clip() của thư viện “numpy” để đảm bảo giá trị pixel nằm trong khoảng [0, 255] và chuyển đổi kết quả sang kiểu “uint8” bằng hàm astype().

6/ CẮT ẢNH THEO KÍCH THƯỚC

Ý tưởng thực hiện: [5]

- Để cắt ảnh theo kích thước, ta cần xác định cạnh hình vuông cần cắt của ảnh sau khi cắt. Sau đó, từ chiều cao, độ rộng của ảnh và cạnh xác định trước đó, ta tính được tọa độ các góc trái trên, trái dưới, phải trên và phải dưới của ảnh mới
- Sử dụng kỹ thuật “slicing” để hàm chức năng chạy nhanh hơn.

Mô tả hàm center_crop(img, size):

- Chức năng: Cắt phần trung tâm của hình ảnh theo kích thước được chỉ định.
- Đầu vào:
 - “img” (mảng numpy) - Hình ảnh dưới dạng mảng numpy.
 - “size” (số nguyên, tùy chọn) - Kích thước của phần cắt (size x size). Mặc định là 256.
- Đầu ra: “res” np.ndarray - Mảng numpy 3D đại diện cho phần trung tâm của hình ảnh đã được cắt.
- Mô tả:
 - Lấy kích thước chiều cao (height) và chiều rộng (width) để xác định vị trí các góc ảnh trung tâm.
 - Tính toán vị trí bắt đầu và kết thúc của phần cắt dựa trên kích thước của ảnh và kích thước cần cắt như miêu tả trên phần ý tưởng.
 - Sử dụng slicing để lấy phần trung tâm của hình ảnh với cú pháp [top:bottom, left:right, :] với “top:bottom” xác định các hàng của mảng muốn cắt, “top” là chỉ số hàng bắt đầu và “bottom” là chỉ số hàng kết thúc cắt kết quả sẽ là hình ảnh được cắt với chiều cao khoảng từ “top” đến “bottom – 1”. Tương tự cho chiều rộng với “left:right” còn các kênh màu giữ nguyên.

7.1/ CẮT ẢNH THEO KHUNG TRÒN

Ý tưởng thực hiện: [5]

- Ta có công thức phương trình đường tròn là:

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

Với (a,b) là tâm đường tròn và R là bán kính đường tròn

- Để cắt ảnh theo khung tròn, thì đường tròn sẽ tiếp xúc với các cạnh của ảnh => bán kính (R) sẽ bé hơn hoặc bằng cạnh nửa cạnh nhỏ nhất của hình.
- Để xác định tọa độ (a,b) ta sẽ tính toán dựa trên trung điểm của chiều cao và rộng (height, width).
- Những điểm không thuộc và không nằm trong đường tròn thì đưa về giá trị 0

Mô tả hàm circle_crop(img):

- Chức năng: Cắt hình ảnh theo hình tròn.

- Đầu vào: “img” (mảng numpy) Hình ảnh dưới dạng mảng numpy 3D (chiều cao, chiều rộng, số kênh màu).
- Đầu ra: “res” (mảng numpy) Hình ảnh đã được cắt thành hình tròn dưới dạng mảng numpy 3D.
- Mô tả:
 - Lấy chiều cao, chiều rộng của hình ảnh bằng hàm `np.shape()`. Tạo tuple chứa tọa độ tâm từ công thức tính trung điểm chiều cao và chiều rộng (`height // 2, width // 2`).
 - Tiếp theo, tạo ra một lưới (grid) 2 chiều (x, y) với kích thước là chiều rộng và chiều cao của ảnh. Các giá trị x và y lần lượt tạo ra một ma trận chứa các giá trị từ 0 đến (`width - 1`) và từ 0 đến (`height - 1`) bằng hàm `np.ogrid()`.
 - Ta sẽ sử dụng `np.ogrid` để tạo ra các ma trận 1 chiều mà mỗi phần tử chứa giá trị của trục y và trục x tương ứng.
 - Phép gán này sẽ tạo ra hai ma trận có kích thước lần lượt là (`height, 1`) và (`1, width`). Và Python sẽ tự động broadcasting các ma trận này để tạo thành hai ma trận có kích thước (`height, width`). Kết quả là chúng ta sẽ có một lưới 2 chiều y và x có kích thước bằng kích thước của ảnh đầu vào (`height, width`).
 - Như vậy, sau khi thực hiện hai dòng code trên, y sẽ là ma trận 2 chiều chứa các giá trị từ 0 đến `height - 1`, và x sẽ là ma trận 2 chiều chứa các giá trị từ 0 đến `width - 1`. Điều này cho phép chúng ta thực hiện các phép toán trên các điểm trên lưới của ảnh.
 - “radius” bán kính của hình tròn, là giá trị nhỏ hơn giữa tọa độ x và y của tâm ảnh. Điều này đảm bảo hình tròn nằm gọn trong ảnh.
 - “mask” một mảng boolean cùng kích thước với ảnh, xác định các điểm nằm ngoài hình tròn bằng công thức trên ý tưởng. Các giá trị True đại diện cho các điểm nằm ngoài hình tròn.
 - Tạo một bản sao “res” từ mảng “img” ban đầu và gán các điểm màu nằm ngoài hình tròn với giá trị 0.

7.2/ CẮT ẢNH THEO KHUNG ELIP

Ý tưởng thực hiện:

- Phương trình elip chuẩn:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

- Để dịch chuyển elip từ gốc tọa độ (0,0) đến một tâm hình ảnh, phương trình trở thành:

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

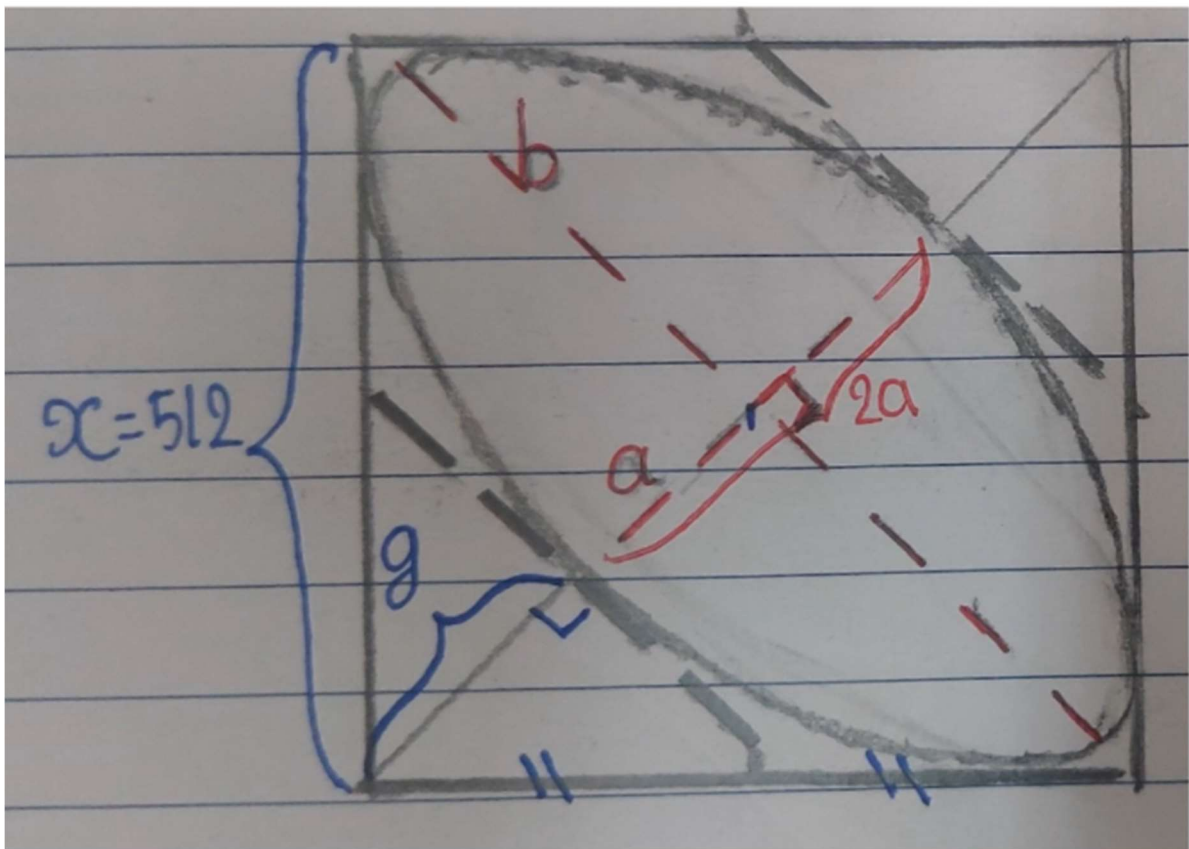
- Để xoay elip quanh tâm (h,k) một góc A, cần dùng phép biến đổi tọa độ [1]. Công thức tổng quát để xoay một điểm (x,y) quanh một điểm cố định (h,k) một góc A là:

$$\begin{aligned} x' &= (x-h) \cos(A) + (y-k) \sin(A) \\ y' &= -(x-h) \sin(A) + (y-k) \cos(A) \end{aligned}$$

- Thay vào phương trình đường ellipse ta được phương trình sau:

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

Với a,b là nửa trục lớn, nhỏ của elip; xác định a,b sao cho hình elip tiếp xúc với cạnh của hình



- Với hình trên, ta giả sử đường nối từ trung điểm cạnh dưới và cạnh trái hình vuông tiếp xúc với điểm xa nhất của ellipse. Từ đó, tính được đường nối đó bằng định lý pytago là $\sqrt{\left(\frac{1}{2}x\right)^2 + \left(\frac{1}{2}x\right)^2} = \frac{\sqrt{2}}{2}x$, sau đó ta tính được cạnh $g = \sqrt{\left(\frac{1}{2}x\right)^2 - \left(\frac{\sqrt{2}}{4}x\right)^2} = \frac{\sqrt{2}}{4}x$, cuối cùng tính được $a = \frac{\sqrt{2}x - \frac{\sqrt{2}}{2}x}{2} = \frac{\sqrt{2}}{4}x$
- Từ hình trên, ta tính cạnh b với công thức $b = a\sqrt{3}$
- Những điểm không thuộc và không nằm trong đường tròn thì đưa về giá trị 0
- Để có 2 hình elip chéo nhau (90 độ), thì elip 1 có góc A là 45 độ $\Leftrightarrow \frac{\pi}{4}$ còn elip 2 có góc A là 135 độ $\Leftrightarrow \frac{3\pi}{4}$

Mô tả hàm ellipse_crop(img):

- Chức năng: Cắt hình ảnh thành hình dạng được xác định bởi giao của hai elip chéo nhau.
- Đầu vào: “img” (mảng numpy) Hình ảnh dưới dạng mảng numpy 3D (chiều cao, chiều rộng, số kênh màu).
- Đầu ra: “res” (mảng numpy) Hình ảnh đã được cắt thành hình yêu cầu dưới dạng mảng numpy 3D.
- Mô tả:
 - Xác định tâm như cắt hình theo đường tròn, sau đó tính trục nhỏ, lớn (a,b) theo công thức trên phần ý tưởng.
 - Tương tự cắt theo đường tròn, tạo hai mảng “numpy” 2D x và y, biểu thị các tọa độ x và y của mỗi điểm ảnh trong hình ảnh qua hàm np.orgid().
 - Tính tọa độ dịch chuyển từ (0,0) đến tâm ảnh sau đó sử dụng công thức xoay trên phần ý tưởng để có 2 phương trình elip chéo nhau như trên phần ý tưởng.
 - Sử dụng chỉ số boolean để xác định các điểm nằm ngoài 2 elip và tạo bản sao “res” dựa trên mảng “img” đặt các điểm ngoài 2 elip là giá trị 0.

8/ HÀM MAIN

Ý tưởng thực hiện:

- Theo yêu cầu được đưa ra, hàm main() cho phép người dùng nhập vào tên tập tin ảnh mỗi khi hàm main được thực thi.

- Cho phép người dùng lựa chọn chức năng xử lý ảnh (từ 1 đến 7, đối với chức năng 4 cho phép lựa chọn giữa lật ngang hoặc lật dọc). Lựa chọn 0 cho phép thực hiện tất cả chức năng với tên file đầu ra tương ứng với từng chức năng.

Ví dụ:

- Đầu vào: cat.png
- Chức năng: Làm mờ
- Đầu ra: cat_blur.png

Mô tả hàm main():

- Chức năng: Gọi các hàm chức năng đã trình bày ở trên để thực hiện quá trình chỉnh sửa hình ảnh, hiển thị kết quả và lưu về đường dẫn trên máy.
- Các bước thực hiện:
 - B1: Nhập đường dẫn đến file ảnh trong máy, truyền vào hàm đọc ảnh (read_img).
 - B2: Nhập lựa chọn để thực hiện chức năng tương ứng. Nếu lựa chọn là “0” thì thực hiện tất cả các chức năng
 - B3: Nếu người dùng chọn 0, tất cả các chức năng xử lý hình ảnh được thực hiện và lưu vào danh sách images chứa tuple (mảng trả về, chuỗi tên chức năng). Nếu người dùng chọn từ 1 đến 10, hàm process_image sẽ xử lý hình ảnh tương ứng và trả về danh sách các hình ảnh đã xử lý cùng với tên tương ứng của chức năng.
 - B4: Xác định dấu chấm cuối cùng trong “img_path” để tách tên tệp và phần mở rộng, sau đó lấy tên tệp “filename” và phần mở rộng “extension”.
 - B5: Sử dụng vòng lặp qua từng giá trị trong “images”, lưu từng hình ảnh đã xử lý với “img_path” mới với format là “filename_tên chức năng.extension”

9/ CÁC HÀM PHỤ TRỢ

① Hàm read_img(img_path):

- Chức năng: Đọc hình ảnh từ đường dẫn file.
- Đầu vào: “img_path” (str) – đường dẫn của file ảnh
- Đầu ra: Trả về dưới dạng mảng 3D numpy (height, width, channels).
- Mô tả: Sử dụng module “Image” từ thư viện “PIL” gọi hàm open() truyền vào đường dẫn ảnh trả về đối tượng Image (một định dạng đặc biệt của “Pillow” cho phép thực hiện các thao tác khác nhau trên hình ảnh). Sau đó, gọi hàm

array() từ thư viện “Numpy” chuyển đổi đối tượng Image thành mảng 3D (height, width, channels).

② Hàm show_img(img):

- Chức năng: Dùng để hiển thị hình ảnh 3D sử dụng thư viện “matplotlib”. Hàm này giúp kiểm tra kết quả của các bước xử lý ảnh.
- Đầu vào: “img” (mảng numpy) 3D (height, width, channels) của hình ảnh.
- Đầu ra: Không trả về giá trị chỉ thực hiện chức năng hiển thị ảnh trên màn hình.
- Mô tả: Gọi các hàm hỗ trợ của thư viện “matplotlib” sau:
 - imshow(img): hiển thị mảng 3D numpy được truyền vào (img) dưới dạng hình ảnh trên không gian vẽ của “matplotlib”.
 - show(): hiển thị hình ảnh đã được vẽ lên không gian vẽ của “matplotlib” ra giao diện “jupyter notebook” nếu không các hình vẽ sẽ đè lên nhau.

③ Hàm save_img(img,img_path):

- Chức năng: Lưu ảnh dưới định dạng ảnh truyền vào.
- Đầu vào:
 - “img”: Mảng 3D của ảnh
 - “img_path”: đường dẫn lưu ảnh.
- Đầu ra: Không trả về giá trị chỉ thực hiện chức năng lưu ảnh vào file theo định dạng truyền vào ban đầu.
- Mô tả: Tạo bộ nhớ hình ảnh từ một đối tượng mảng với kiểu dữ liệu là “uint8” bằng hàm astype() sau đó dùng hàm save() truyền vào đường dẫn để lưu ảnh.

BẢNG ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH







Hình: Ảnh gốc

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%	
5.1	Làm mờ ảnh	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	Phóng to/ Thu nhỏ 2x	0%	

TÀI LIỆU THAM KHẢO

- [1] *Teaching of Multimedia Presentation Creation. (n.d.). ERIC - Education Resources Information Center. Retrieved July 30, 2024, from <https://files.eric.ed.gov/fulltext/EJ1238623.pdf>*
- [2] *Kernel (image processing). (n.d.). Wikipedia. Retrieved July 31, 2024, from [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))*
- [3] *Grayscale to RGB Conversion. (n.d.). Tutorials Point. Retrieved July 31, 2024, from https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm*
- [4] *Shakeel, Y. (n.d.). How to convert a color image into sepia image - Image Processing Project. dyclassroom. Retrieved July 31, 2024, from <https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image>*
- [5] *Tài liệu của cô Phạm Thị Phương Uyên (lab03_project02) và video hướng dẫn làm đồ án*