

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA TP.HCM**

KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN MÔN HỌC

Toán ứng dụng và thống kê cho công nghệ thông tin
Đồ án 2: IMAGE PROCESSING

Giảng viên hướng dẫn

Phan Thị Phương Uyên

Vũ Quốc Hoàng

Nguyễn Văn Quang Huy

Sinh viên thực hiện

Nguyễn Thị Minh Minh 21127528

Ngày 30 tháng 7 năm 2023

Mục lục

1 Đánh giá hoàn thành	4
2 Thư viện đã dùng	4
3 Thay đổi độ sáng của ảnh	4
3.1 Ý tưởng thực hiện	4
3.2 Hàm <code>adjust_brightness(image_arr, level_brightness)</code>	4
3.3 Hình ảnh kết quả	5
3.4 Nhận xét kết quả	5
4 Thay đổi độ tương phản của ảnh	6
4.1 Ý tưởng thực hiện	6
4.2 Hàm <code>adjust_contrast(image_arr, level_contrast)</code>	6
4.3 Hình ảnh kết quả	7
4.4 Nhận xét kết quả	7
5 Lật ảnh	7
5.1 Ý tưởng thực hiện	7
5.2 Hàm <code>flip_updown(image_arr)</code> và <code>flip_rightleft(image_arr)</code>	8
5.3 Hình ảnh kết quả	8
5.4 Nhận xét kết quả	9
6 Chuyển đổi ảnh RGB về grayscale/sephia	9
6.1 Ý tưởng thực hiện	9
6.2 Hàm <code>def convert_gray_sephia(image_arr, method = 'grayscale')</code>	10
6.3 Hình ảnh kết quả	10
6.4 Nhận xét kết quả	11
7 Làm mờ và làm nét ảnh	11
7.1 Ý tưởng thực hiện	11
7.2 Mô tả hàm	11
7.3 Hình ảnh kết quả	13
7.4 Nhận xét kết quả	15
8 Cắt ảnh theo kích thước (ở trung tâm)	15
8.1 Ý tưởng thực hiện	15
8.2 Hàm <code>crop_center(image_arr, height, width)</code>	15
8.3 Hình ảnh kết quả	16
8.4 Nhận xét kết quả	16
9 Cắt ảnh theo khung hình tròn	16
9.1 Ý tưởng thực hiện	16
9.2 Hàm <code>crop_circle(image_arr)</code>	16

9.3 Hình ảnh kết quả	17
9.4 Nhận xét kết quả	17
10 Cắt ảnh theo khung là hai hình elip chéo nhau. [ADVANCED]	17
10.1 Ý tưởng thực hiện	17
10.2 Hàm <code>crop_to_ellipse(image_arr)</code>	18
10.3 Hình ảnh kết quả	19
10.4 Nhận xét kết quả	19
11 Các hàm đọc ghi hình ảnh	19
11.1 Đọc và chuyển ảnh về ma trận điểm ảnh	19
11.2 Lưu ảnh đã được chuyển đổi	20
12 Hàm main	20
Tài liệu tham khảo	21

1 Đánh giá hoàn thành

Chức năng	Đánh giá mức độ hoàn thành
Thay đổi độ sáng	100%
Thay đổi độ tương phản	100%
Lật ảnh	100%
Chuyển đổi ảnh RGB về grayscale/ sepia	100%
Làm mờ và làm nét ảnh	100%
Cắt ảnh từ trung tâm	100%
Cắt ảnh theo khung hình tròn	100%
Cắt ảnh theo khung là hai hình ellipse chéo nhau	100%

2 Thư viện đã dùng

- PIL: Dùng để đọc, ghi ảnh.
- numpy: Dùng để thực hiện tính toán trên ma trận.
- matplotlib: Dùng để hiển thị ảnh.

3 Thay đổi độ sáng của ảnh

3.1 Ý tưởng thực hiện

Trong xử lý ảnh kỹ thuật số, một tấm ảnh được biểu diễn dưới dạng ma trận các pixel, trong đó mỗi pixel thể hiện bằng giá trị màu tại một vị trí cụ thể trên tấm ảnh. Ở dạng màu R, G, B, mỗi điểm ảnh có ba kênh màu: R: Đỏ, G: Xanh lá cây, B: Xanh da trời; và giá trị mỗi kênh nằm trong khoảng [0, 255]. Để thay đổi độ sáng của một tấm ảnh, chúng ta có thể sử dụng một số kỹ thuật đơn giản như điều chỉnh giá trị màu của mỗi pixel. Ví dụ, để làm cho ảnh trở nên sáng hơn, chúng ta có thể tăng giá trị của mỗi kênh màu (R, G, B) của mỗi điểm ảnh. Điều này cho phép ảnh hiển thị màu sáng hơn tại mỗi vị trí, tạo ra hiệu ứng độ sáng tổng thể của ảnh. Ngược lại, nếu muốn làm ảnh tối hơn, chúng ta có thể giảm giá trị của mỗi kênh màu. Cách thức này sẽ làm mất đi sự sáng bóng và tạo ra các màu tối hơn.

3.2 Hàm `adjust_brightness(image_arr, level_brightness)`

```
def adjust_brightness(image_arr, level_brightness):
    img_result = image_arr.astype(np.uint16) + level_brightness
    img_result = np.clip(img_result, 0, 255)
    return img_result.astype(np.uint8)
```

Ở thuật toán này, đơn giản chúng ta chỉ cần cộng thêm cho ma trận ảnh 1 lượng giá trị độ sáng gọi là **level_brightness**. Tuy nhiên, để ảnh giữ đúng tính chất, ta cần chuyển về dạng **uint16** trước khi cộng bởi vì giá trị điểm ảnh có kiểu dữ liệu mặc định là **uint8** có giá trị trong khoảng từ 0 đến 255, nên khi tăng độ sáng có thể vượt quá giá trị tối đa là 255. Chuyển sang **uint16** giúp tránh được hiện tượng tràn số. Sau cùng, dùng hàm **np.clip** để đảm bảo giá trị mỗi điểm ảnh đều nằm trong khoảng hợp lệ và trả về lại mang kết quả với kiểu dữ liệu **uint8** ban đầu.

3.3 Hình ảnh kết quả



Hình 1: Original picture.



(a) Bright picture



(b) Dark picture

3.4 Nhận xét kết quả

- Với độ sáng là $30 > 0$, thì hình ảnh đã được làm sáng lên đáng kể so với ảnh gốc ban đầu.
- Với độ sáng là $-30 < 0$, thì hình ảnh đã được làm tối đi so với ảnh gốc ban đầu.

4 Thay đổi độ tương phản của ảnh

4.1 Ý tưởng thực hiện

Tương tự với việc thay đổi độ sáng của ảnh, để thay đổi độ tương phản của ảnh, chương trình cũng sẽ thay đổi giá trị của từng pixel. Để tăng độ tương phản của ảnh, ta cần gia tăng khoảng cách giữa các pixel có giá trị cao và các pixel có giá trị thấp. Ngược lại để giảm độ tương phản, ta cần thu hẹp khoảng cách giữa chúng. Cụ thể ta có thể sử dụng công thức sau để thay đổi độ tương phản của ảnh:

$$g(i, j) = \alpha [f(i, j) - 128] + 128$$

Trong đó:

- $f(i, j)$ là giá trị pixel của ảnh gốc trước khi thay đổi độ tương phản.
- $g(i, i)$ là giá trị pixel của ảnh sau khi thay đổi độ tương phản.
- α là cường độ thay đổi độ tương phản. Giá trị $\alpha > 1$ sẽ tăng độ tương phản, $\alpha < 1$ sẽ giảm độ tương phản, và $\alpha = 1$ sẽ giữ nguyên độ tương phản ban đầu.

Giá trị 128 trong công thức là giá trị trung bình của khoảng giá trị màu từ 0 đến 255. Bằng cách trừ giá trị trung bình này từ giá trị pixel $f(i, j)$, ta điều chỉnh các pixel dương thành các giá trị âm và pixel âm thành các giá trị dương, từ đó làm tăng khoảng cách giữa chúng và tăng độ tương phản của ảnh.

4.2 Hàm `adjust_contrast(image_arr, level_contrast)`

```
def adjust_contrast(image_arr, level_contrast):
    img_result = level_contrast*(image_arr.astype(np.float32) - 128) + 128
    img_result = np.clip(img_result, 0, 255)
    return img_result.astype(np.uint8)
```

Với ý tưởng đã được triển khai, hàm `adjust_contrast` có nhiệm vụ chuyển đổi ma trận ảnh sang contrast với công thức đã cho. Điều quan trọng là cần phải chỉnh kiểu dữ liệu của ma trận để tránh việc tràn số.

4.3 Hình ảnh kết quả



Hình 2: Original picture.



(a) Increased contrast picture



(b) Decreased contrast picture

4.4 Nhận xét kết quả

- Với độ tương phản là $1.5 > 1$, thì ảnh sẽ được tăng độ tương phản. Lúc này, ảnh sẽ trở nên đậm hơn, các chi tiết trên ảnh sẽ trở nên nổi bật và tương phản với nhau hơn.
- Với độ tương phản là $0.5 < 1$, thì ảnh sẽ được giảm độ tương phản. Lúc này, ảnh sẽ trở nên mờ hơn và có thể mất đi một số chi tiết, ngoài ra còn có thể gây ra hiệu ứng màu âm bản như ảnh minh họa.

5 Lật ảnh

5.1 Ý tưởng thực hiện

Dối với việc lật ảnh thì lại tương đối đơn giản khi ta chỉ cần thay đổi vị trí của từng pixel đến vị trí tương ứng với yêu cầu. Ví dụ:

- Lật ảnh ngang: Với yêu cầu này, ta chỉ cần di chuyển các pixel nằm bên phải sang bên trái và tương tự với các pixel nằm bên trái, di chuyển chúng sang bên phải. Hay nói cách khác, ta lấy đối xứng các pixel qua trục dọc chính giữa của ảnh.
- Lật ảnh dọc: Tương tự như việc lật ảnh ngang, ta lấy đối xứng các pixel qua trục ngang chính giữa của ảnh.

5.2 Hàm flip_updown(image_arr) và flip_rightleft(image_arr)

```
def flip_updown(image_arr):
    return np.flipud(image_arr)
def flip_rightleft(image_arr):
    return np.fliplr(image_arr)
```

Ở hàm này, ta chỉ đơn giản sử dụng thư viện `numpy` gọi hàm `np.flipud` để di chuyển đối xứng các pixel qua trục ngang chính giữa của ảnh, `np.fliplr` để di chuyển đối xứng các pixel qua trục dọc chính giữa của ảnh.

5.3 Hình ảnh kết quả



Hình 3: Original picture.



(a) Updown picture



(b) Rightleft picture

5.4 Nhận xét kết quả

Ảnh đã được chuyển về đúng chiều được chọn.

6 Chuyển đổi ảnh RGB về grayscale/sepia

6.1 Ý tưởng thực hiện

Ý tưởng của hệ màu grayscale là biến đổi bức ảnh ban đầu thành ảnh xám, trong đó mỗi điểm ảnh chỉ có một kênh màu duy nhất thay vì ba kênh màu RGB như trước đây. Phương pháp này loại bỏ thông tin về màu sắc và giữ lại thông tin về độ sáng của ảnh. Với ảnh màu RGB, chúng ta có thể tính giá trị trung bình của ba kênh màu (R, G, B) tại mỗi điểm ảnh và sử dụng giá trị đó làm giá trị cho cả ba kênh màu mới. Kết quả là mỗi điểm ảnh sẽ có giá trị độ sáng duy nhất, tạo nên ảnh xám.

Để tạo hiệu ứng sepia cho ảnh, ta sẽ lần lượt thay đổi các pixel của ảnh bằng cách nhân ma trận ảnh ban đầu với ma trận các giá trị để tạo hiệu ứng. Cụ thể, ta sử dụng công thức:

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

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

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

Khi đó, ta nhân lần lượt các hệ màu với các vector tương ứng để tạo ra các pixel màu mới.

6.2 Hàm def convert_gray_sepia(image_arr, method = 'grayscale')

```
def convert_gray_sepia(image_arr, method = 'grayscale'):
    if method == 'grayscale':
        return np.mean(image_arr.astype(np.float64), axis = 2).astype(np.uint8)
    elif method == 'sepia':
        image_arr = image_arr.astype(np.float64)
        R_scalar = np.array([0.393, 0.769, 0.189])
        G_scalar = np.array([0.349, 0.686, 0.168])
        B_scalar = np.array([0.272, 0.534, 0.131])
        copy_image = image_arr.copy()
        image_arr[:, :, 0] = np.dot(copy_image, R_scalar)
        image_arr[:, :, 1] = np.dot(copy_image, G_scalar)
        image_arr[:, :, 2] = np.dot(copy_image, B_scalar)

    return np.clip(image_arr, 0, 255).astype(np.uint8)
```

- Ở mode grayscale, mảng kết quả được trả về là mảng gốc đã được xử lý bằng cách lấy trung bình cộng giá trị của 3 kênh màu R, G, B của mỗi pixel với hàm `np.mean` và lấy `mean` theo chiều thứ 3 của ma trận là `axis = 2`.
- Ở mode sepia, ta tiến hành nhân tích vô hướng của ma trận từng kênh màu với các chỉ số `R_scalar`, `G_scalar` và `B_scalar` đã cho với `np.dot` và cuối cùng trả lại mảng kết quả đã được xử lý từng kênh màu.

6.3 Hình ảnh kết quả



Hình 4: Original picture.



(a) Gray picture



(b) Sepia picture

6.4 Nhận xét kết quả

- Ảnh gốc đã được chuyển về với dạng ảnh xám với gam màu chỉ còn dao động từ đen đến trắng. Tuy số màu đã bị giảm đi nhưng vẫn giữ được nét gốc của ảnh.
- Ảnh sepia được chuyển về dạng ảnh với gam màu nâu và vàng. Số màu bị giảm nhưng vẫn giữ được nét gốc của ảnh, tạo cảm giác ấm áp, cổ điển cho ảnh.

7 Làm mờ và làm nét ảnh

7.1 Ý tưởng thực hiện

Làm mờ và làm nét ảnh là hai kỹ thuật thú vị trong xử lý ảnh, giúp thay đổi độ mịn và sắc nét của hình ảnh một cách hiệu quả. Để thực hiện hai kỹ thuật này, chúng ta sử dụng phép convolution - một phương pháp biến đổi trong xử lý ảnh.

Convolution (tích chập) là một kĩ thuật xây dựng ma trận mới bằng cách tính các pixel mới dựa trên các pixel lân cận của vị trí đó với kernel là một ma trận quyết định sự ảnh hưởng của các pixel lân cận lên giá trị mới. Thông qua việc áp dụng các kernel phù hợp, ta có thể tạo ra các kết quả như làm mờ hay làm nét ảnh.

Đối với làm mờ, ta có thể sử dụng Box Blur Kernel hoặc Gaussian Blur Kernel. Bên cạnh đó, ta sử dụng Unsharp Mask Kernel để làm nét hình ảnh.

7.2 Mô tả hàm

- Hàm `def blur_image(image_arr)`

```


def blur_image(image_arr):
    #save the boundary in the return image array
    new_image = np.ones_like(image_arr)
    gaussian_kernel = np.array([[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]])
    gaussian_kernel = gaussian_kernel/256
    if(image_arr.ndim == 3):
        for channel in range(3):
            new_image[:, :, channel] = convolution_blur(image_arr[:, :, channel], gaussian_kernel)

    else:
        new_image = convolution_blur(image_arr, gaussian_kernel)

    return np.clip(new_image,0,255)

def convolution_blur(channel_matrix, gausian_kernel):
    result_matrix = np.ones_like(channel_matrix)

    result_matrix[:1,:] = channel_matrix[:1,:]
    result_matrix[:-2,:] = channel_matrix[:-2,:]
    result_matrix[:,1] = channel_matrix[:,1]
    result_matrix[:,:-2] = channel_matrix[:, :-2]

    for i in range(2, channel_matrix.shape[0]-3):
        for j in range(2, channel_matrix.shape[1]-3):
            sub_arr = channel_matrix[i-2:i+3, j-2:j+3]
            result_matrix[i][j] = np.sum(sub_arr*gausian_kernel)

    return result_matrix

```

- Chúng ta có thêm 1 hàm phụ `convolution_blur` để tính lại ma trận của từng kênh màu, bằng cách cho chạy vòng for từ dòng 2 cột 2 của ma trận, mỗi lần chạy sẽ thực hiện nhân từng phần tử ma trận với từng phần tử của `gaussian_kernel` tại vị trí tương ứng, sau đó lấy tổng kết quả gán vào vị trí điểm ảnh với kênh màu hiện tại.
- Ở hàm chính `blur_image` sẽ tiến hành tích chập các kênh màu đã được xử lý qua hàm `convolution_blur` và trả về kết quả. Nếu như là ảnh grayscale thì chỉ cần tính trên 1 kênh màu và trả về kết quả.
- Hàm `def sharpen_image(image_arr)`

```

● ● ●

def sharpen_image(image_arr):
    #save the boundary in the return image array
    new_image = np.ones_like(image_arr)
    gaussian_kernel = np.array([[1,4,6,4,1],
                                [4,16,24,16,4],
                                [6,24,-476,24,6],
                                [4,16,24,16,4],
                                [1,4,6,4,1]])/-256
    if(image_arr.ndim == 3):
        for channel in range(3):
            new_image[:, :, channel] = convolution_sharpen(image_arr[:, :, channel], gaussian_kernel)

    else:
        new_image = convolution_sharpen(image_arr, gaussian_kernel)
    return np.clip(new_image,0,255)

def convolution_sharpen(channel_matrix, gausian_kernel):
    result_matrix = np.ones_like(channel_matrix)
    result_matrix[:1,:] = channel_matrix[:1,:]
    result_matrix[:-2,:] = channel_matrix[:-2,:]
    result_matrix[:, :1] = channel_matrix[:, :1]
    result_matrix[:, :-2] = channel_matrix[:, :-2]

    for i in range(2, channel_matrix.shape[0]-3):
        for j in range(2, channel_matrix.shape[1]-3):
            sub_arr = channel_matrix[i-2:i+3, j-2:j+3]
            result_matrix[i][j] = np.sum(sub_arr*gausian_kernel)

    return result_matrix

```

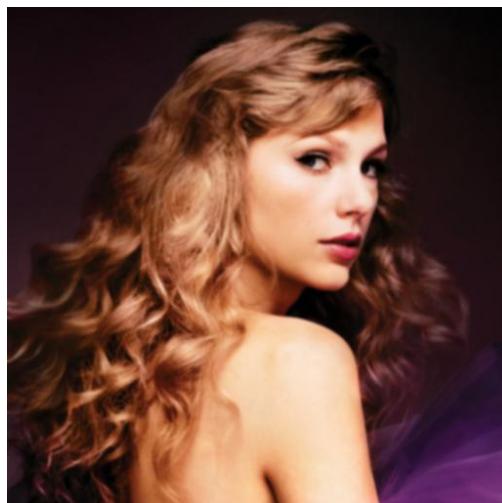
Ở hàm thực hiện sắc nét ảnh, ta cũng thực hiện tương tự như blur_image tuy nhiên kernel Gaussian sẽ khác để thực hiện làm sắc nét ảnh.

7.3 Hình ảnh kết quả

- Ảnh màu: 512 x 512



Hình 5: Original picture.

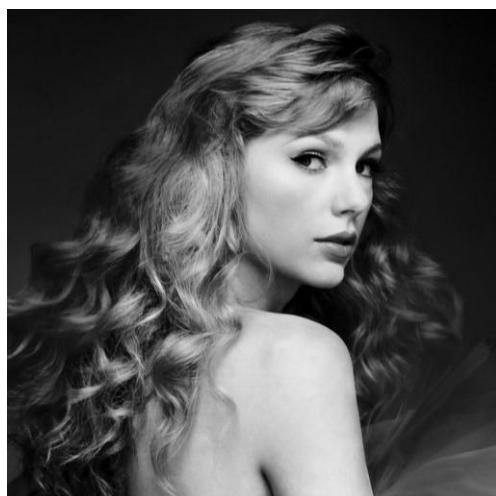


(a) Blurred picture

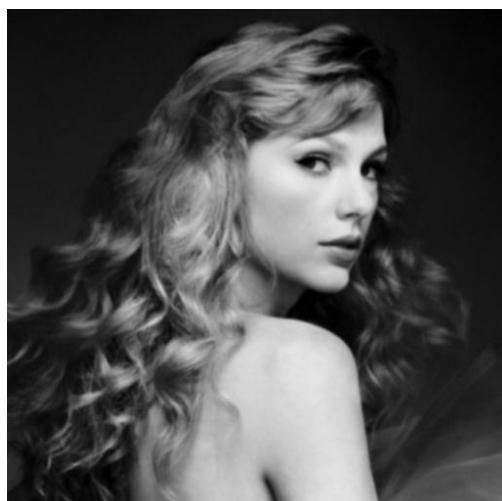


(b) Sharpened picture

- Ảnh trăng đen: 512 x 512



Hình 6: Original picture.



(a) Blurred picture



(b) Sharpened picture

7.4 Nhận xét kết quả

- Ở thuật toán đã cài đặt, sẽ có thể làm sắc nét và làm mờ cho cả ảnh màu và ảnh trắng đen.
- Ảnh blurred picture được làm mờ từ ảnh gốc và ảnh sharpened picture được làm sắc nét lại từ ảnh mờ.
- Nhìn chung, thì ảnh được làm mờ có kết quả khá mờ hơn so với ảnh gốc. Và ảnh sắc nét cũng được làm sắc nét hơn so với ảnh mờ.
- Với Gaussian kernel thì ma trận 5x5 sẽ làm ảnh mờ/sắc nét hơn so với ma trận 3x3.
- Với ảnh trắng đen thì thời gian làm mờ/sắc nét sẽ ngắn hơn nhiều so với ảnh màu bởi vì chỉ cần thực hiện convolution trên 1 kênh màu.
- Với ảnh càng lớn thì thời gian thực hiện thuật toán sẽ càng được kéo dài. Với ảnh màu thì thời gian làm mờ/sắc nét lần lượt là: 4.97s và 5.15s; đối với ảnh trắng đen thì thời gian làm mờ/sắc nét lần lượt là: 1.7s và 1.59s.

8 Cắt ảnh theo kích thước (ở trung tâm)

8.1 Ý tưởng thực hiện

Để cắt ảnh theo kích thước cho sẵn, ta cần xác định tọa độ của các góc ảnh dựa vào thông tin về chiều dài, chiều rộng của ảnh, và tọa độ tâm của ảnh. Sau đó, ta có thể cắt phần ảnh mong muốn từ tâm của ảnh gốc bằng cách sử dụng các tọa độ này.

8.2 Hàm crop_center(image_arr, height, width)



```

def crop_center(image_arr, height, width):
    width = min(width, image_arr.shape[1])
    height = min(height, image_arr.shape[0])
    center = [image_arr.shape[1]//2, image_arr.shape[0]//2]
    first_corner = [center[0] - width//2, center[1] - height//2]
    return image_arr[first_corner[1] : first_corner[1] + height, first_corner[0] : first_corner[0] + width,:]

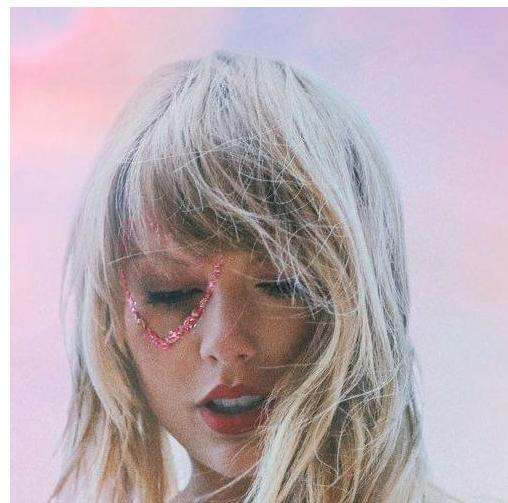
```

Dầu tiên, để kiểm tra dữ kiện đầu vào có hợp lệ hay không, ta sẽ lấy min của kích thước chiều dài/ rộng muốn cắt so với chiều dài/ rộng của ảnh và lấy kết quả min để cắt. Tiếp theo, tính tọa độ tâm của hình bằng cách lấy chiều dài/ rộng chia 2; từ tâm và giá trị cần cắt xác định tọa độ góc trái trên của ảnh. Từ đó xác định được các góc còn lại và trả về ma trận mới từ tọa độ các góc.

8.3 Hình ảnh kết quả



(c) Original picture



(d) Center crop picture

8.4 Nhận xét kết quả

- Ảnh đã được cắt đúng theo cầu, cắt ở trung tâm với kích thước dài rộng cho trước.
- Ở ảnh trên, ảnh gốc có kích thước là 1200x1200 và ảnh mới được cắt ra từ trung tâm với kích thước 500x500.

9 Cắt ảnh theo khung hình tròn

9.1 Ý tưởng thực hiện

Để thực hiện cắt ảnh theo khung tròn, ý tưởng chính là xác định được vùng hình tròn tại trung tâm ảnh bằng cách viết phương trình đường tròn. Sau đó, chúng ta sẽ giữ lại phần ảnh nằm trong vùng hình tròn và tô màu đen cho phần ảnh nằm ngoài khung tròn để tạo ra hiệu ứng cắt ảnh theo khung tròn.

9.2 Hàm crop_circle(image_arr)

```

● ● ●
def create_circle_mask(image_arr):
    center_y = image_arr.shape[0]//2
    center_x = image_arr.shape[1]//2
    y, x = np.ogrid[0:image_arr.shape[0], 0:image_arr.shape[1]]
    radius = min(center_x, center_y)
    circle_mask = (x-center_x)**2 + (y-center_y)**2
    return circle_mask <= radius **2

```

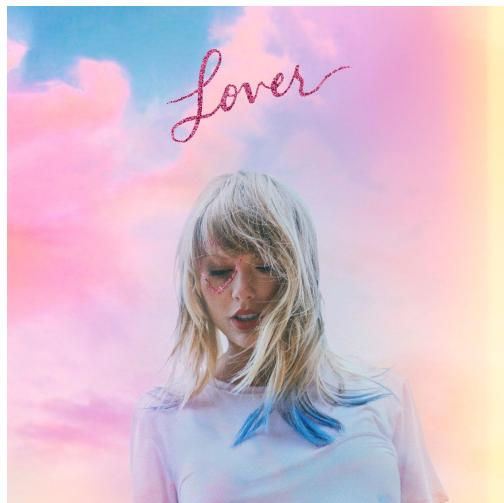
```

def crop_circle(image_arr):
    mask = create_circle_mask(image_arr)
    new_image = image_arr.copy()
    new_image[~mask] = 0
    return new_image

```

- Trước tiên để cắt ảnh ra hình tròn, ta sẽ tiến hành viết hàm phụ `create_circle_mask` xác định miền ảnh sẽ được giữ lại. Sử dụng công thức: $(x - x_{center})^2 + (y - y_{center})^2 \leq r^2$ với tọa độ tâm được xác định bằng cách lấy chiều thứ nhất và thứ hai của ma trận chia 2.
- Sau khi đã xác định được miền tròn, hàm `crop_circle` sẽ tiến hành tô đen phần pixel ngoài miền đã tạo: `new_image[~mask] = 0`, giúp có hiệu ứng cắt ảnh theo khung tròn.

9.3 Hình ảnh kết quả



(e) Original picture



(f) Circle crop picture

9.4 Nhận xét kết quả

Ảnh đã được cắt theo hình tròn đúng theo yêu cầu.

10 Cắt ảnh theo khung là hai hình elip chéo nhau. [ADVANCED]

10.1 Ý tưởng thực hiện

Ý tưởng thực hiện cắt ảnh theo hai khung ellipse chéo nhau khá giống với hình tròn. Chúng ta cũng sẽ xác định miền ellipse sẽ giữ ảnh lại bằng cách viết phương trình ellipse nghiêng 1 góc 45° và 1 ellipse khác nghiêng 1 góc 135° , lấy hợp của 2 ellipse được tạo với phép \cup . Sau đó tô đen phần ngoài vùng ellipse để tạo hiệu ứng cắt ảnh.

Hai công thức sẽ được áp dụng vào quá trình cắt ảnh:

- Đầu tiên, cắt ảnh về hình vuông ở trung tâm với cạnh là min của chiều dài, rộng hiện tại của ảnh.
- Xác định độ dài của trục lớn, trục bé của ellipse sao cho ellipse nằm xiên này sẽ chạm được 4 cạnh hình vuông qua công thức (a : 1 nửa độ dài trục lớn, b : 1 nửa độ dài trục bé): $a^2 + b^2 = \frac{x^2}{2}$
- Xác định miền của ellipse qua công thức:

$$\left(\frac{(x-x_{\text{center}}) \cos \alpha + (y-y_{\text{center}}) \sin \alpha}{a} \right)^2 + \left(\frac{(x-x_{\text{center}}) \sin \alpha - (y-y_{\text{center}}) \cos \alpha}{b} \right)^2 \leq 1$$

10.2 Hàm crop_to_ellipse(image_arr)

```

def crop_to_square(image_arr):
    side = min(image_arr.shape[0], image_arr.shape[1]) #the length side of square
    center = [image_arr.shape[1]//2, image_arr.shape[0]//2]
    first_corner = [center[0] - side//2, center[1] - side // 2]
    return image_arr[first_corner[1] : first_corner[1] + side, first_corner[0] : first_corner[0] + side, :]

def create_mask_ellipse(image_arr, angle):
    side = min(image_arr.shape[0], image_arr.shape[1])
    center = [image_arr.shape[1]//2, image_arr.shape[0]//2]

    #major axis and minor axis is in form of power of 2

    major_axis = 0.4225* side**2
    minor_axis = 0.0775* side**2
    y, x = np.ogrid[:image_arr.shape[0], :image_arr.shape[1]]

    x_val = x - center[0]
    y_val = y - center[1]
    radian_angle = np.radians(angle)
    cos_val = np.cos(radian_angle)
    sin_val = np.sin(radian_angle)

    ellipse_domain = ((x_val*cos_val + y_val*sin_val)**2/major_axis) + ((x_val*sin_val - y_val*cos_val)**2/minor_axis)
    return ellipse_domain <= 1

def crop_to_ellipse(image_arr):
    square_img = crop_to_square(image_arr)

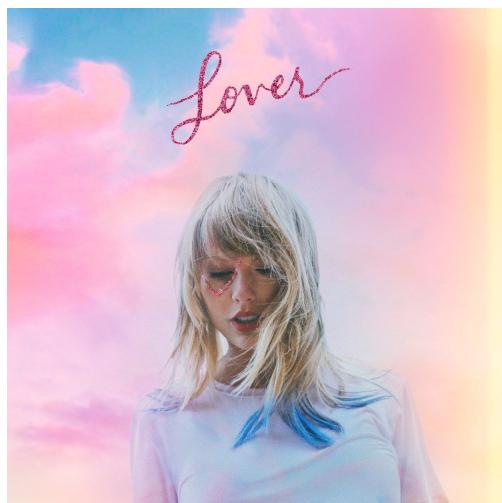
    ellipse1 = create_mask_ellipse(square_img, 45)
    ellipse2 = create_mask_ellipse(square_img, 135)
    ellipse_combine = ellipse1 | ellipse2

    new_image = square_img.copy()
    new_image[~ellipse_combine] = 0
    return new_image

```

- Hàm `crop_to_square` sẽ chuyển mọi ảnh về hình vuông với cạnh là min của chiều dài và rộng của ảnh gốc.
- Hàm `create_mask_ellipse` sẽ giúp chúng ta xác định miền ellipse cần giữ lại. Ở đây, trục lớn và trục bé được xác định giá trị ở dạng bình phương. Lấy một cách tùy ý giá trị của trục lớn, cho độ dài trục lớn là xấp xỉ $1.3 * \text{side}$ (side : độ dài hình vuông), suy ra ta có: $\text{major_axis} = 0.4225 * \text{side}^2$. Sử dụng công thức trên, tính được $\text{minor_axis} = 0.0775 * \text{side}^2$ sao cho ellipse có thể chạm được 4 cạnh hình vuông. Cuối cùng, sử dụng phương trình ellipse xiên nêu trên để trả về miền giá trị cần giữ.
- Hàm `crop_to_ellipse` tiến hành cắt ảnh thành hình vuông trước. Sau đó tạo 2 mask ellipses theo 2 đường chéo của hình vuông bằng cách cho `ellipse1` nghiêng 1 góc 45° , `ellipse2` nghiêng 1 góc 135° và lấy `ellipse1 | ellipse2`. Lúc này, ta có `ellipse_combine` là miền ảnh sẽ giữ lại, tiến hành tô đen phần ngoài để tạo hiệu ứng cắt ảnh.

10.3 Hình ảnh kết quả



(g) Original picture



(h) Skewed ellipses crop picture

10.4 Nhận xét kết quả

Hình ảnh đã được cắt theo hai 2 hình ellipse chéo nhau. Nếu muốn tăng kích thước của vùng ảnh được giữ, ta có thể tăng độ dài cho trục lớn và trục bé của ellipse.

11 Các hàm đọc ghi hình ảnh

11.1 Đọc và chuyển ảnh về ma trận điểm ảnh

```

● ● ●

def read_Image_array(img_path):
    try:
        if img_path.split(".")[-1] == 'png':
            convert_png_jpg(img_path, img_path.split(".")[0] + ".jpg")
            img_path = img_path.split(".")[0]+'.jpg'

        image = Image.open(img_path)
        if image.mode != 'RGB':
            image = image.convert('RGB')
        return np.array(image)
    except Exception as e:
        print(f"Error while reading the image: {e}")
    return None

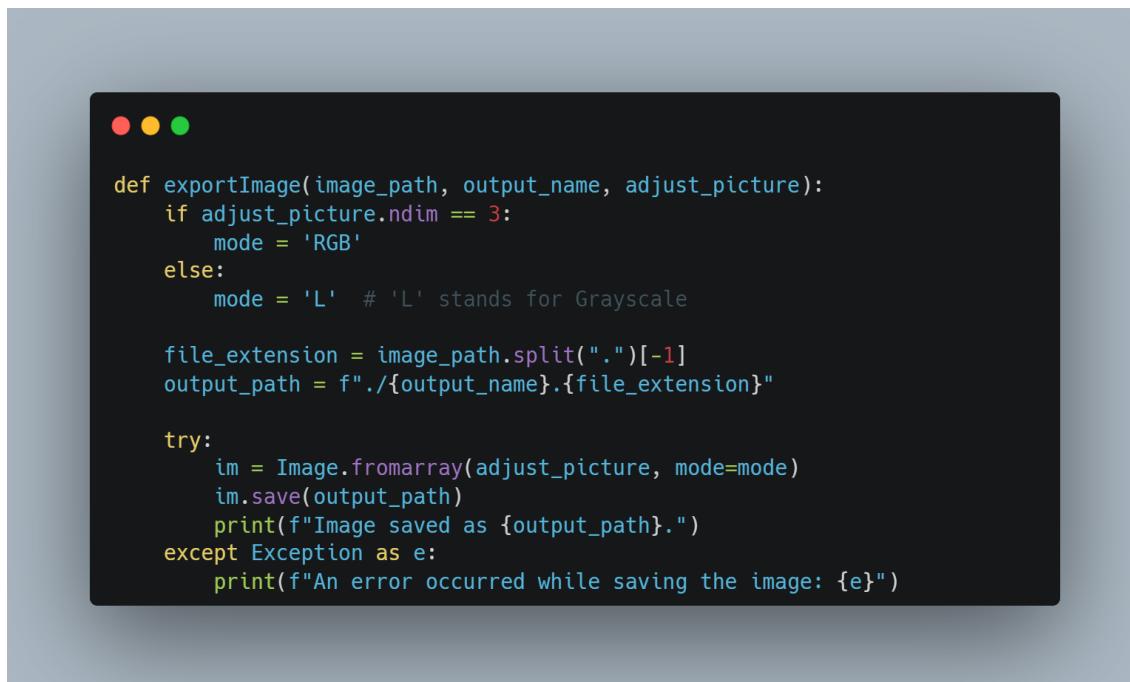
```

- Để đọc ảnh thì em sử dụng chủ yếu là các hàm của thư viện `Image`. Tuy nhiên, trước khi ảnh được chuyển đổi về dạng `numpy array` thì cần phải kiểm tra trước 1 số điều kiện. Các ảnh màu thường được lưu dưới dạng ma trận R, G, B với kiểu dữ liệu trong khoảng từ 0 đến 255. Nhưng trong 1 số trường hợp, ảnh sẽ có thêm kênh thứ 4 là kênh A (`alpha`), dùng hàm `image.convert` để chuyển về lại dạng

RGB. Ngoài ra, đối với ảnh có đuôi .png thì 1 số trường hợp có kiểu dữ liệu ảnh là dưới dạng float từ 0.0 đến 1.0; đối với các trường hợp này, thì em xử lý bằng cách chuyển ảnh về đuôi jpg và đem ảnh mới này lên lại để tiếp tục dùng các chức năng khác.

- Dùng np.array để chuyển ảnh về ma trận điểm ảnh.

11.2 Lưu ảnh đã được chuyển đổi



```
def exportImage(image_path, output_name, adjust_picture):
    if adjust_picture.ndim == 3:
        mode = 'RGB'
    else:
        mode = 'L' # 'L' stands for Grayscale

    file_extension = image_path.split(".")[-1]
    output_path = f"./{output_name}.{file_extension}"

    try:
        im = Image.fromarray(adjust_picture, mode=mode)
        im.save(output_path)
        print(f"Image saved as {output_path}.")
    except Exception as e:
        print(f"An error occurred while saving the image: {e}")
```

- Lưu ảnh đã được xử lý về với save của thư viện Image, tên ảnh là tên ảnh gốc _chúcnăng có đuôi file trùng với đuôi file gốc.

12 Hàm main

Cho xuất ra MENU chức năng, sau đó người dùng sẽ nhập tên ảnh (VD: test1.jpg), tiếp tục chọn chức năng tương ứng với MENU và nhận kết quả.

Tài liệu

[1] Lab 03 Project 2, document lab from TA.

[2] Ngoc Tien Github, Sep 20 2022.

Available at: <https://github.com/NgocTien0110/Applied-Mathematics-and-Statistics>

[3] Numpy Library.

Available at: <https://numpy.org/>

[4] Formula of sepia picture, May 2023.

Available at: <http://leware.net/photo/blogSepia>

[5] Image processing with Blur and Sharpen.

Available at: [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

[6] Formula for axis in ellipse inscribed a square.

Available at: <http://users.math.uoc.gr/~pamfilos>

[7] Formula for skewed ellipse.

Available at: https://www.maa.org/external_archive/joma/Volume8/Kalman/General

[8] Save file with different extension.

Available at: <https://stackoverflow.com/save-file-extension>