



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN



# **BÁO CÁO ĐỒ ÁN 2 TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CÔNG NGHỆ THÔNG TIN**

**TÊN ĐỒ ÁN:  
IMAGE PROCESSING**

**LỚP: 21CLC05**

**Thông tin cá nhân:**

21127099    Nguyễn Tấn Lộc

**Giảng viên hướng dẫn:**

Thầy Vũ Quốc Hoàng

Thầy Lê Thanh Tùng

Cô Phạm Thị Phương Uyên

Thầy Nguyễn Văn Quang Huy

## MỤC LỤC

<b>THÔNG TIN CÁ NHÂN.....</b>	<b>2</b>
<b>THÔNG TIN ĐỒ ÁN .....</b>	<b>2</b>
<b>KẾT QUẢ ĐỒ ÁN .....</b>	<b>3</b>
<b>1. Bảng đánh giá các chức năng hoàn thành.....</b>	<b>3</b>
<b>2. Liệt kê các chức năng.....</b>	<b>8</b>
2.1. Hàm thay đổi độ sáng .....	10
2.2. Hàm thay đổi độ tương phản.....	11
2.3. Hàm lật ảnh (ngang -dọc) .....	13
2.4. Chuyển đổi ảnh RGB sang ảnh xám .....	14
2.5. Chuyển đổi ảnh RGB sang ảnh sepia.....	15
2.6. Làm mờ ảnh .....	18
2.7. Làm sắc nét ảnh.....	19
2.8. Cắt ảnh theo kích thước (cắt ảnh ở trung tâm).....	21
2.9. Cắt ảnh theo khung tròn .....	22
2.10. Cắt ảnh theo khung là 2 elip chéo nhau.....	26
2.11. Hàm main.....	28
<b>3. Nhận xét tổng thể.....</b>	<b>28</b>
<b>4. Điểm yếu bản thân.....</b>	<b>28</b>
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>29</b>

## THÔNG TIN CÁ NHÂN

**Họ tên:** Nguyễn Tấn Lộc

**Mã số sinh viên:** 21127099

**Lớp:** 21CLC5

## THÔNG TIN ĐỒ ÁN

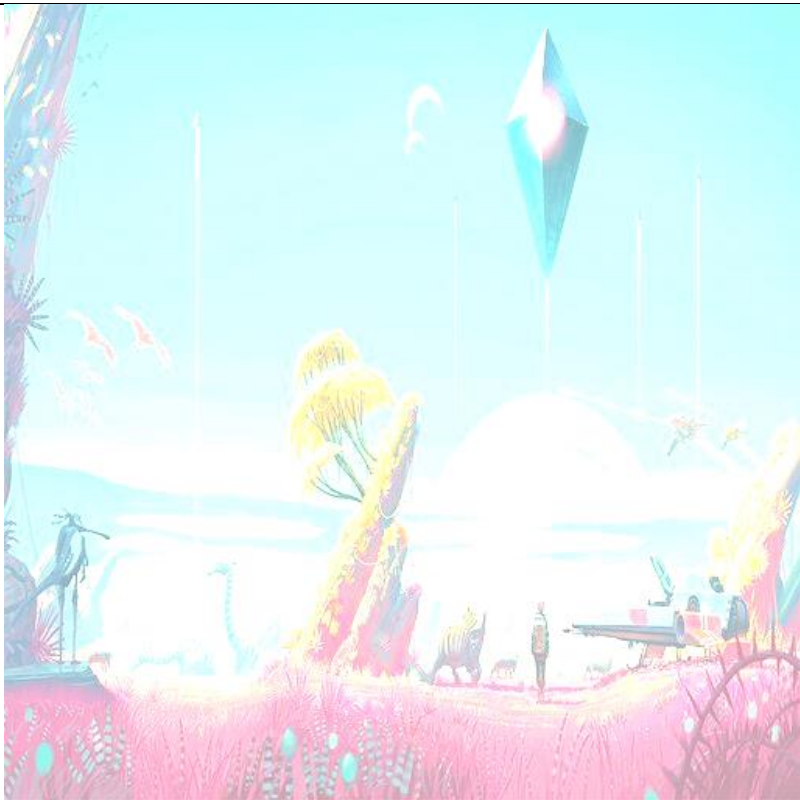
**Mã học phần:** MTH00057


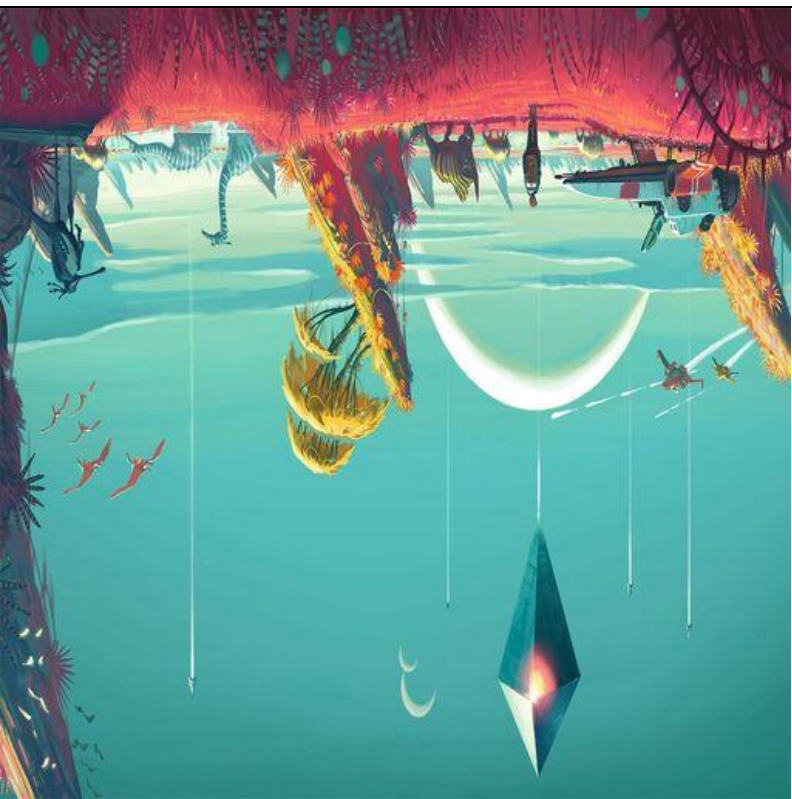
**Tên học phần:** Toán ứng dụng và thống kê cho công nghệ thông tin

**Tên đồ án:** Color Compression



# KẾT QUẢ ĐỒ ÁN



## 1. Bảng đánh giá các chức năng hoàn thành



STT	Chức năng	Hoàn thành	Ghi chú
1	Thay đổi độ sáng cho ảnh		

2	Thay đổi độ tương phản		
3	Lật ảnh (ngang – dọc)		




4	Chuyển đổi ảnh RGB sang ảnh xám		
5	Chuyển đổi ảnh RGB sang ảnh sephia		

6	Làm mờ ảnh		Sử dụng Gaussian Blur 3x3
7	Làm sắc nét ảnh		

8	Cắt ảnh theo kích thước (cắt ở trung tâm)		
9	Cắt ảnh theo khung tròn		2 cách
10	Viết hàm main xử lý	100%	



11	Cắt ảnh theo khung là 2 elip chéo nhau		
----	--	---	--

## 2. Liệt kê các chức năng

- Ở chức năng cắt ảnh theo khung tròn, em thực hiện 2 cách nên tổng cộng thì sẽ có 3 hàm: cách 1 (2 hàm), cách 2 (1 hàm).
- Em có tổng cộng 18 hàm, trong đó, có 12 hàm cho các chức năng chính, các hàm còn lại là hỗ trợ cho các chức năng chính:
  - Hàm `adjust_brightness` → Đây là hàm thay đổi độ sáng cho ảnh.
  - Hàm `change_contrast` → Đây là hàm thay đổi độ tương phản cho ảnh.
  - Hàm `flip_image` → Đây là hàm lật ảnh (ngang – dọc).
  - Hàm `convert_to_gray` → Đây là hàm chuyển đổi ảnh RGB sang ảnh xám.
  - Hàm `convert_to_sephia` → Đây là hàm chuyển đổi ảnh RGB sang ảnh sephia.
  - Hàm `blurred_image` → Đây là hàm làm mờ ảnh.
  - Hàm `sharpen_image` → Đây là hàm làm sắc nét ảnh.
  - (Hàm hỗ trợ) Hàm `convolve` → Đây là hàm hỗ trợ cho chức năng làm mờ và làm sắc nét ảnh.



- (Hàm hỗ trợ) Hàm gaussian\_blur → Đây là hàm hỗ trợ cho chức năng làm mờ ảnh bằng cách sử dụng Gaussian Blur.
- (Hàm hỗ trợ) Hàm sharpen → Đây là hàm hỗ trợ cho chức năng làm mờ ảnh bằng cách sử dụng Sharpen.
- Hàm crop\_image\_center → Đây là hàm cắt ảnh theo kích thước (cắt ở trung tâm).
- Hàm crop\_image\_with\_circle\_1 → Đây là hàm cắt ảnh theo khung tròn theo cách 1.
- Hàm crop\_image\_with\_circle\_2 → Đây là hàm cắt ảnh theo khung tròn theo cách 2.
- (Hàm hỗ trợ) Hàm rotate\_point\_circle → Đây là hàm hỗ trợ cho chức năng cắt ảnh theo khung tròn theo cách 1.
- Hàm crop\_image\_with\_ellipses → Đây là hàm cắt ảnh theo khung là 2 elip chéo nhau.
- (Hàm hỗ trợ) Hàm rotate\_point\_ellipses → Đây là hàm hỗ trợ cho chức năng cắt ảnh theo khung là 2 elip chéo nhau.
- Hàm main → Đây là hàm thực hiện tất cả các tác vụ input, output và lưu ảnh.

## ẢNH ĐƯỢC SỬ DỤNG ĐỂ THỰC HIỆN ĐỒ ÁN CÓ KÍCH THƯỚC 512X512



### 2.1. Hàm thay đổi độ sáng

#### 2.1.1. Ý tưởng:

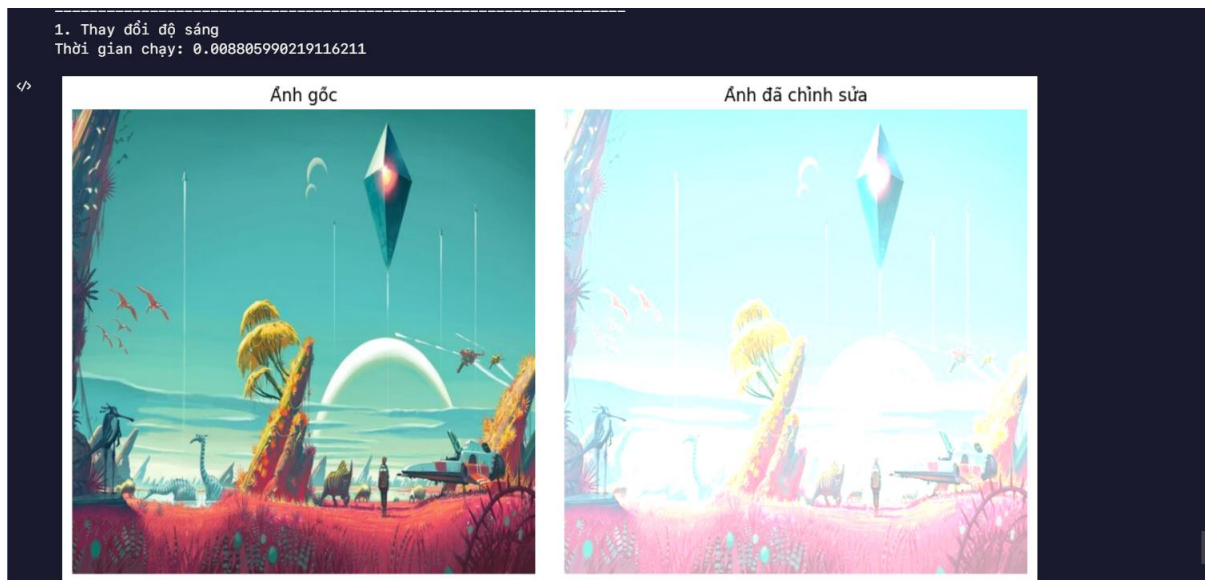
- Muốn điều chỉnh độ sáng của ảnh em sử dụng cách thêm một giá trị trung bình của pixel vào từng pixel của ảnh nhưng thành phần màu của ảnh không nằm ngoài  $[0,255]$ .

#### 2.1.2. Mô tả:

- *Input:*
  - o **image\_path:** Tên của tập tin ảnh.
- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Hầu hết các hàm chức năng của em sẽ phải mở ảnh bằng **Image.open** và chuyển ảnh về mảng Numpy bằng **np.array**.

- Điều chỉnh tăng độ sáng của ảnh bằng cách thêm giá trị trung bình của pixel vào từng pixel của ảnh. Điều này được em thực hiện bằng phép cộng **image\_array + np.mean** trong đó **np.mean** là tính giá trị trung bình của pixel trên toàn bộ ảnh. Nếu muốn giảm độ sáng thì thay vì cộng thêm giá trị trung bình thì mình sẽ trừ giá trị trung bình.
- Sử dụng **np.clip** để đảm bảo các giá trị pixel nằm trong khoảng [0,255]
- Cuối cùng, chuyển đổi tượng lại thành ảnh PIL bằng **Image.fromarray**. (Phải định dạng các pixel là kiểu nguyên 8-bit không dấu).

### 2.1.3. Kết quả:



### 2.1.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã sáng hơn).
- Một số khu vực sáng quá có thể làm bị thiếu thông tin.
- Vì ảnh này đã xuất hiện hiện tượng là ảnh quá sáng vì giá trị trung bình của các pixel quá cao.
- Thời gian chạy nhanh.

## 2.2. Hàm thay đổi độ tương phản

### 2.2.1. Ý tưởng:

- Đầu tiên phải có một hệ số tương phản, chỉ cần thực hiện phép toán nhân hệ số tương phản đó với mảng của ảnh nhưng thành phần màu của ảnh không nằm ngoài [0,255]. Độ tương phản tăng hay giảm dựa theo hệ số tương phản (lớn hơn 1 là tăng, ngược lại là giảm).

### 2.2.2. Mô tả:

#### - *Input:*

- **image\_path:** Tên của tập tin ảnh.
- **contrast\_factor:** Hệ số tương phản

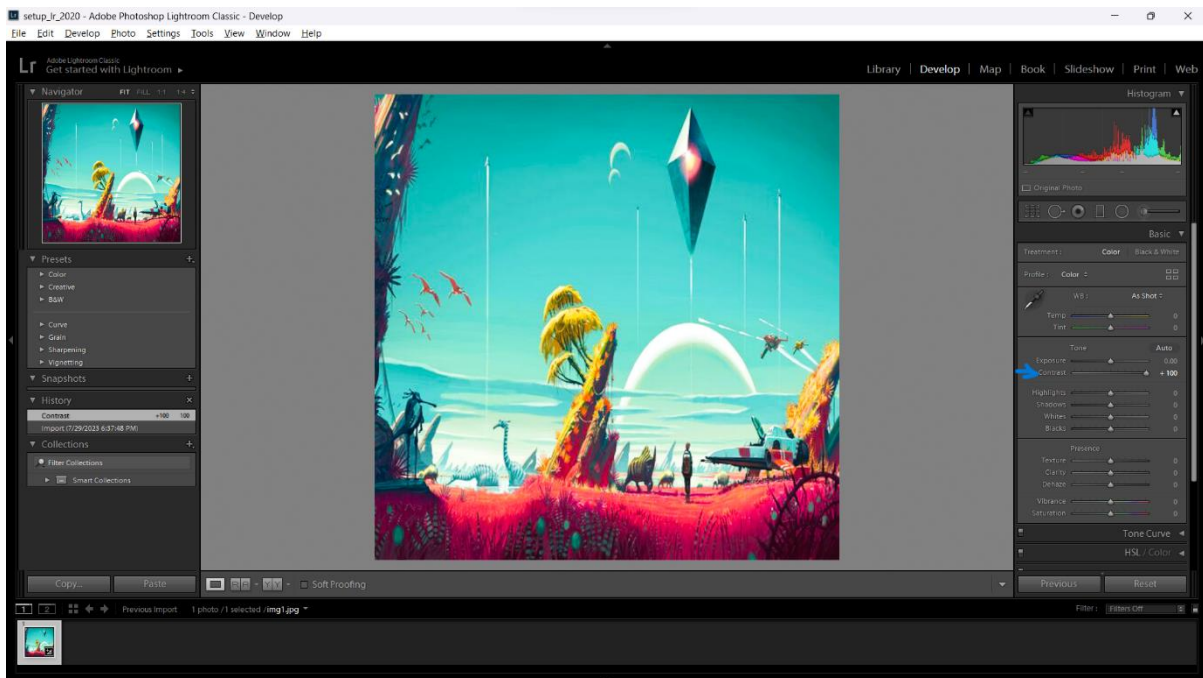
#### - *Output:*

- Ảnh đã được xử lý.

#### - *Các bước thực hiện:*

- Mở ảnh, chuyển sang mảng Numpy.
- Em có 3 cách:
  - **Cách 1:** Đơn giản là nhân mỗi giá trị pixel với hệ số tương phản.
  - **Cách 2:** Tương tự cách 1, nhưng trừ đi thêm giá trị trung bình của ảnh. Điều này giúp cho khi điều chỉnh sẽ không làm thay đổi mức xám trung bình của ảnh.
  - **Cách 3:** Tương tự với cách 2, trừ đi giá trị trung bình của ảnh, nhân với hệ số tương phản và sau đó cộng lại giá trị trung bình ban đầu.

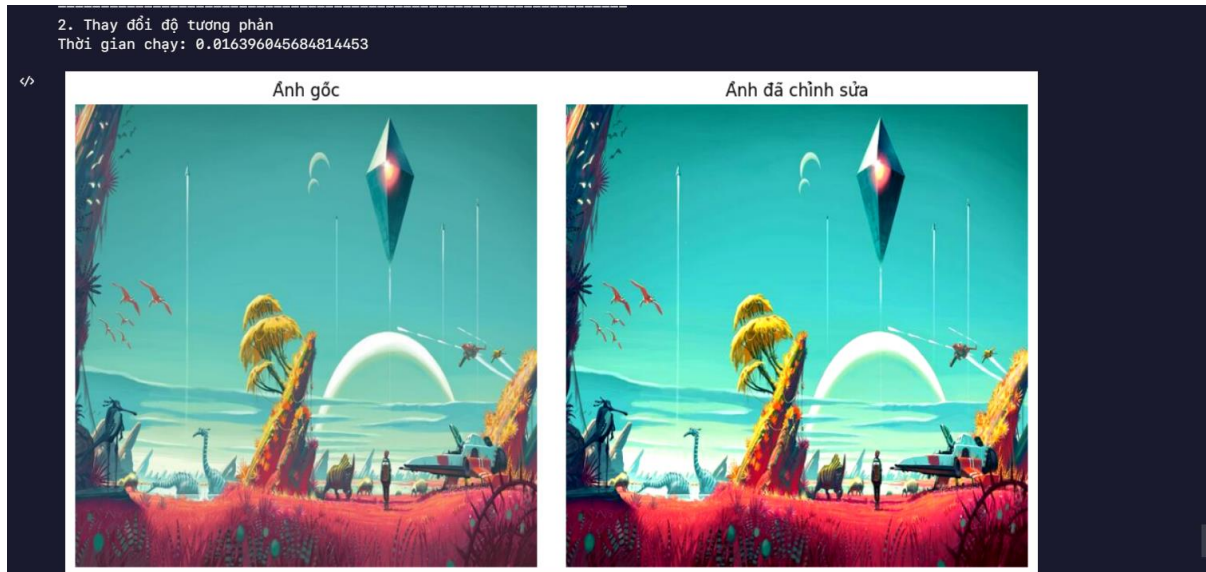
➔ Ở cách 1, thì em lại thấy kết quả xuất ra không giống với độ tương phản lắm. (Em thấy kết quả giống với thay đổi độ sáng). Vì để kiểm tra việc thay đổi độ tương phản. Em có sử dụng phần mềm Adobe Lightroom 2020 để kiểm chứng. Kết quả bên dưới cho thấy cách 1 chưa hợp lý nên em đã sử dụng cách 3 để thực hiện chức năng.





- Sử dụng **np.clip** để đảm bảo các giá trị pixel nằm trong khoảng  $[0, 255]$ .
- Cuối cùng, chuyển đổi tượng lại thành ảnh PIL bằng **Image.fromarray**.

### 2.2.3. Kết quả:



### 2.2.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã có độ tương phản cao hơn).
- Thời gian chạy nhanh.

## 2.3. **Hàm lật ảnh (ngang -dọc)**

### 2.3.1. Ý tưởng:

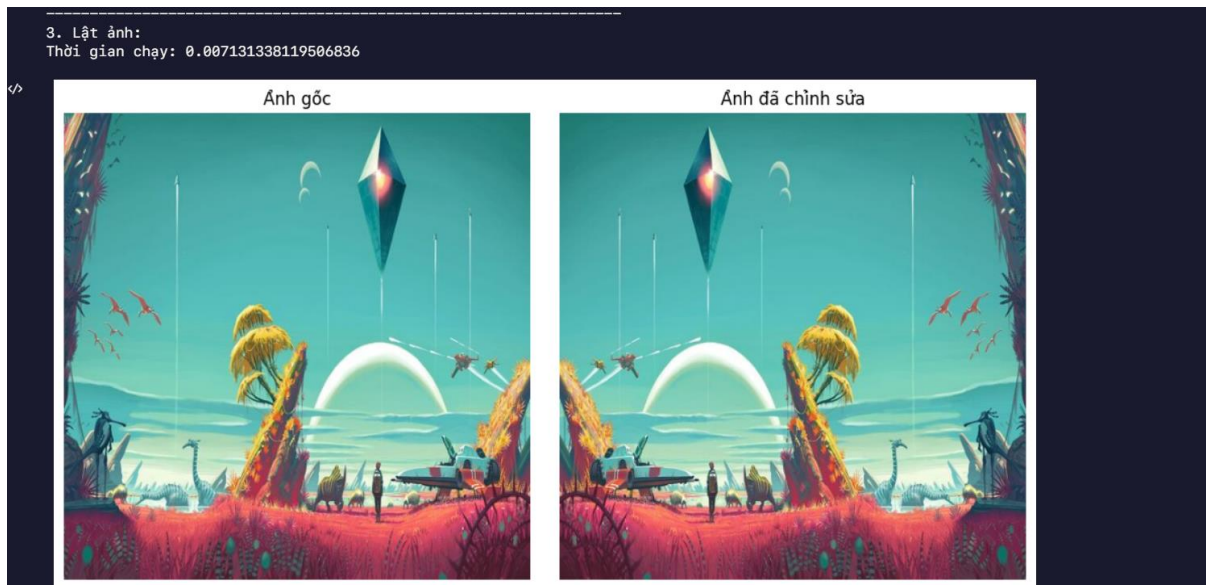
- Thực hiện thao tác lật ma trận của ảnh. Đối với lật ảnh ngang, ta sẽ lật ma trận của ảnh theo trục ngang (đảo các dòng), còn đối với lật ảnh dọc, ta sẽ lật ma trận của ảnh theo trục dọc (đảo các cột).

### 2.3.2. Mô tả:

- *Input:*
  - **image\_path:** Tên của tập tin ảnh.
  - **direction:** Hướng lật ảnh (h: lật ngang, v: lật dọc).
- *Output:*
  - Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - Mở ảnh, chuyển sang mảng Numpy.
  - Dựa vào **direction**, em sử dụng hàm **np.flip** để lật ma trận của ảnh theo trục ngang ( $axis=1$ ) hoặc trục dọc ( $axis=0$ ).

- Cuối cùng, chuyển đổi tượng lại thành ảnh PIL bằng **Image.fromarray**.

### 2.3.3. Kết quả:



### 2.3.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã có lật ngang).
- Các chi tiết của ảnh gốc vẫn giữ được, không bị thay đổi điểm màu nào.
- Thời gian chạy nhanh.

## 2.4. Chuyển đổi ảnh RGB sang ảnh xám

### 2.4.1. Ý tưởng:

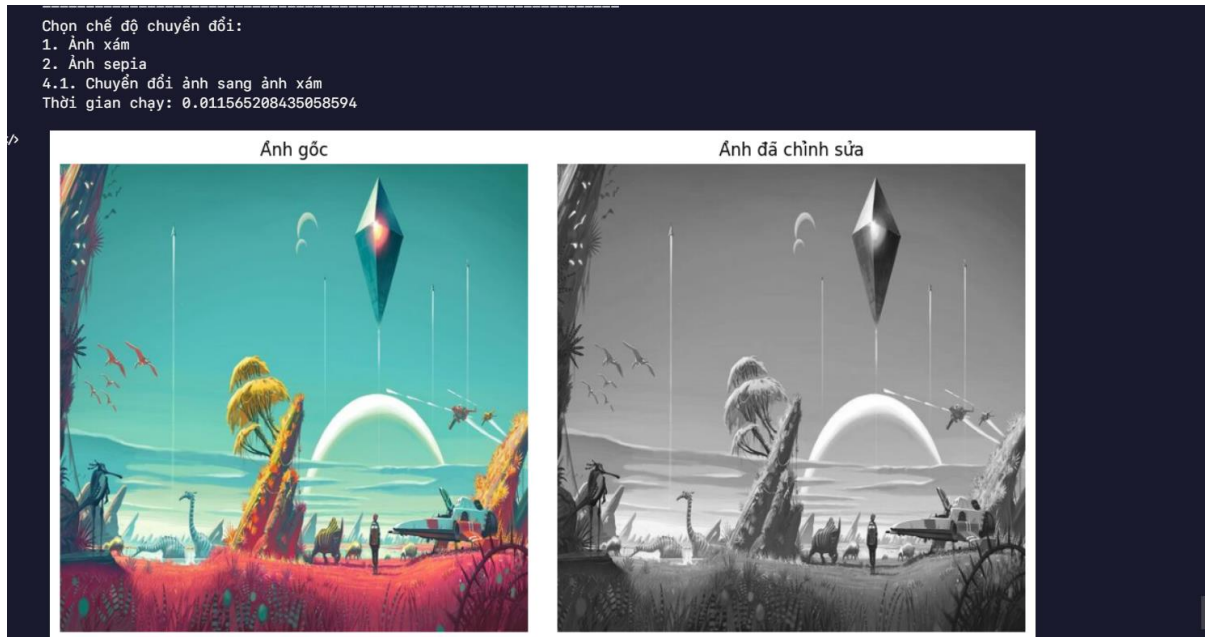
- Sử dụng phương pháp trọng số (weighted method) để tính toán giá trị mức xám của từng pixel trong ảnh. Dựa vào nguồn đã tham khảo <sup>[1]</sup>, Em chọn các trọng số như sau: **30% màu đỏ, 59% màu xanh lá, 11% màu xanh dương.**

### 2.4.2. Mô tả:

- *Input:*
  - **image\_path:** Tên của tập tin ảnh.
- *Output:*
  - Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - Mở ảnh, chuyển sang mảng Numpy.
  - Nhân ma trận giữa mảng ảnh và mảng trọng số ở trên [0.3, 0.59, 0.11] để tính giá trị mức xám của từng pixel.
  - Sử dụng **np.minimum** để đảm bảo các giá trị pixel không vượt quá 255 (giống **np.clip**).

- Cuối cùng, chuyển đổi tượng lại thành ảnh PIL bằng **Image.fromarray**.

### 2.4.3. Kết quả:



### 2.4.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã chuyển sang màu xám).
- Ảnh trở nên đơn sắc, không còn màu sắc nhiều như ảnh gốc.
- Mức xám của từng pixel phản ánh giá trị trung bình của màu trong kênh đỏ, xanh lá, và xanh dương của ảnh gốc.
- Thời gian chạy nhanh.

## 2.5. Chuyển đổi ảnh RGB sang ảnh sepia

### 2.5.1. Ý tưởng:

- Sử dụng ma trận biến đổi màu để tính toán giá trị màu mới cho từng pixel trong ảnh gốc, dựa trên công thức dựa trên nguồn đã tham khảo [2]. Dưới đây là ma trận A.

$$\text{sepia\_r} = 0.393 \cdot R + 0.769 \cdot G + 0.189 \cdot B$$

$$\text{sepia\_g} = 0.349 \cdot R + 0.686 \cdot G + 0.168 \cdot B$$

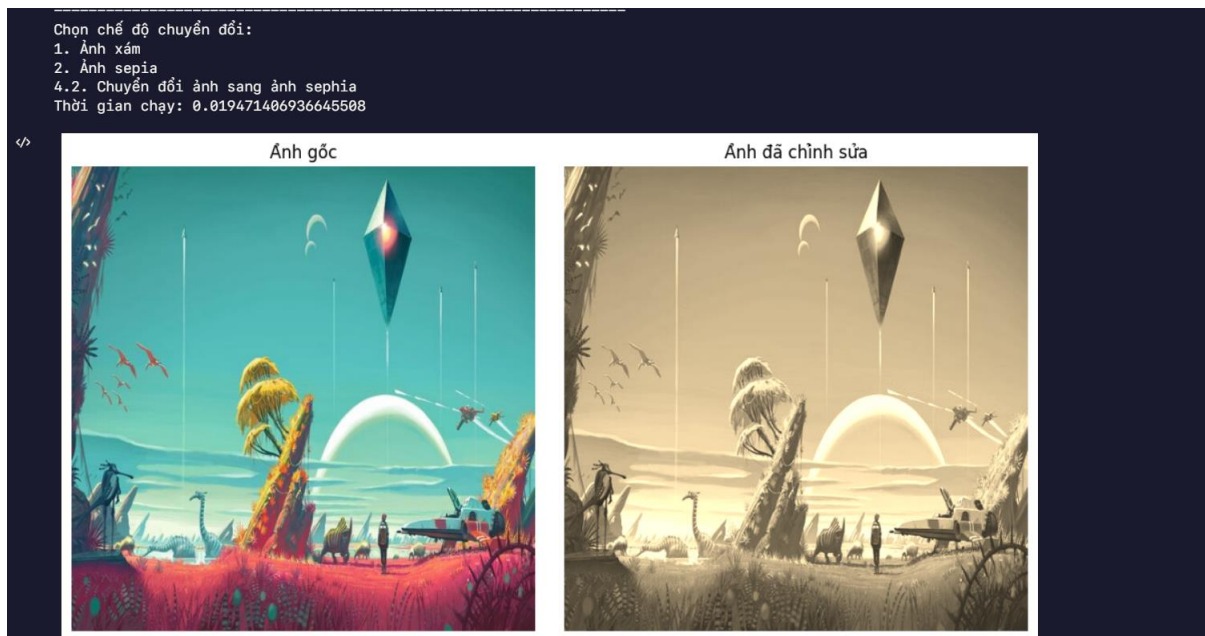
$$\text{sepia\_b} = 0.272 \cdot R + 0.534 \cdot G + 0.131 \cdot B$$

### 2.5.2. Mô tả:

- *Input:*

- **image\_path:** Tên của tập tin ảnh.
- **Output:**
  - Ảnh đã được xử lý.
- **Các bước thực hiện:**
  - Mở ảnh, chuyển sang mảng Numpy.
  - Nhân ma trận các pixel với ma trận A (đã chuyển vị) tức là A.T
  - Sử dụng **np.minimum** để đảm bảo các giá trị pixel không vượt quá 255 (giống **np.clip**).
  - Cuối cùng, chuyển đối tượng lại thành ảnh PIL bằng **Image.fromarray**.

### 2.5.3. Kết quả:



### 2.5.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã chuyển sang màu sepia).
- Ảnh có tông màu nâu cổ điển, hoài niệm.
- Tính tự nhiên của ảnh gốc bị giảm.
- Thời gian chạy nhanh.



VỚI 2 CHỨC NĂNG LÀM MỜ ẢNH VÀ LÀM SẮC NÉT ẢNH, EM SỬ DỤNG CHUNG 1 HÀM CONVOLVE LÀ HÀM CHẬP GIỮA 1 KÊNH VÀ KERNEL

```
1 def convolve(channel, kernel):
2     # Sử dụng np.pad để thêm phần đệm xung quanh kênh
3     padded_channel = np.pad(channel, 1, mode='edge')
4
5     # Áp dụng chập tại đây
6     result = np.zeros_like(channel, dtype=float)
7     for i in range(3):
8         for j in range(3):
9             result += padded_channel[i:i+channel.shape[0], j:j+channel.shape[1]] * kernel[i, j]
10
11     return result
```

- Giải thích hàm:

○ *Input:*

- **channel:** mảng Numpy 2D là kênh màu của ảnh (R, G, B) được truyền vào các hàm làm mờ và làm sắc nét để áp dụng phép tích chập lên nó.
- **kernel:** ma trận trọng số có kích thước 3x3 dùng để thực hiện phép tích chập. Kernel này được áp dụng lên kênh màu để làm mờ ảnh hoặc làm sắc nét ảnh.

○ *Output:*

- Trả về 1 mảng chứa kết quả sau khi áp dụng phép tích chập → chứa giá trị mới cho từng điểm ảnh trong kênh màu sau khi đã làm mờ.

○ *Các bước thực hiện:*

- Trước khi áp dụng phép tích chập, em phải thêm phần đệm xung quanh kênh màu để tránh mất thông tin bằng cách sử dụng np.pad.
- Sử dụng 2 vòng lặp để duyệt ma trận kernel và tích chập kernel với kênh màu. Cụ thể, với mỗi vị trí (i, j) trong kernel, em lấy một vùng con (sub-array) của kênh màu (có kích thước 3x3) bắt đầu từ (i, j) và nhân từng phần tử của kernel với phần tử tương ứng trong vùng con của kênh màu. Sau đó, ta cộng các kết quả nhân lại để tính tổng giá trị mới cho vị trí (i, j) trong kênh màu.

## 2.6. Làm mờ ảnh

### 2.6.1. Ý tưởng:

- Thực hiện trung bình cộng giá trị màu của các điểm ảnh trong vùng xung quanh mỗi điểm ảnh, sử dụng một ma trận trọng số gọi là **kernel** bên dưới được em tham khảo tại nguồn <sup>[3]</sup>.
- Kernel có kích thước 3x3 hoặc 5x5, và giá trị của kernel được tính toán sao cho tổng các trọng số trong kernel bằng 1 để đảm bảo tổng các giá trị màu của các điểm ảnh sau khi làm mờ vẫn giữ nguyên, không làm thay đổi độ sáng tổng thể của ảnh. Em sử dụng kernel 3x3.

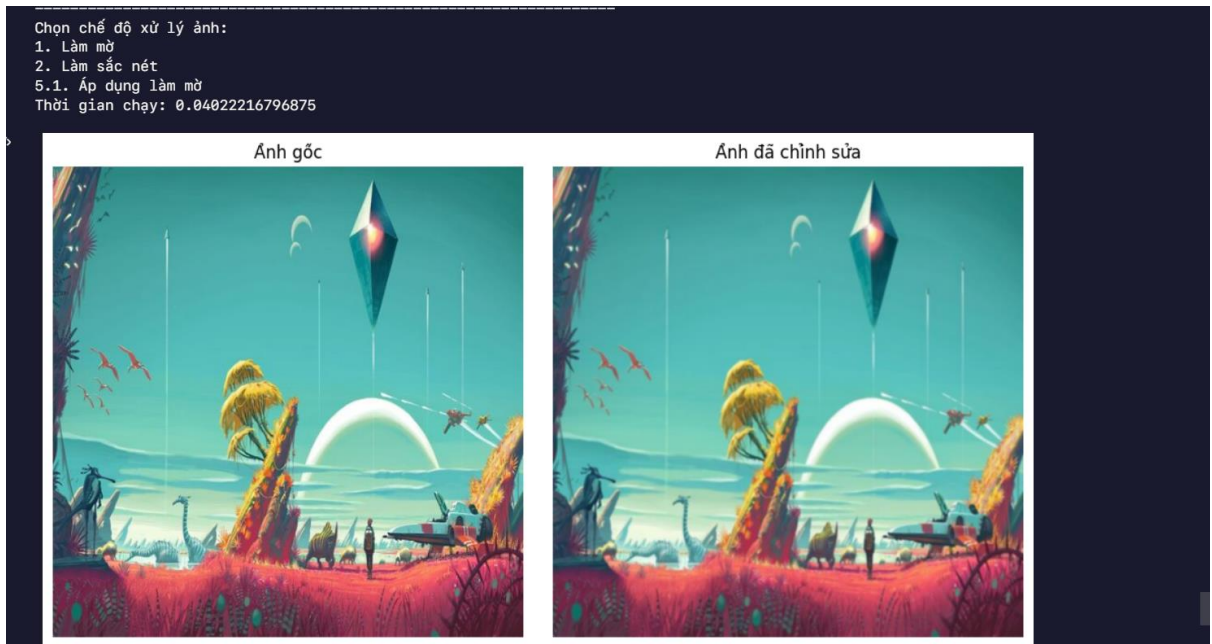
1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

### 2.6.2. Mô tả:

- Hàm **blurred\_image** sẽ là hàm chính thực hiện làm mờ ảnh.
- *Input:*
  - o **image\_path:** Tên của tập tin ảnh.
- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Mở ảnh, chuyển sang mảng Numpy.
  - o Áp dụng phép làm mờ Gaussian Blur bằng hàm **gaussian\_blur**.
  - o Sử dụng **np.clip** để đảm bảo các giá trị pixel nằm trong khoảng [0,255].
  - o Cuối cùng, chuyển đối tượng lại thành ảnh PIL bằng **Image.fromarray**.
- Hàm **gaussian\_blur** sẽ là hàm hỗ trợ thực hiện phép làm mờ Gaussian Blur.
- *Input:*
  - o **img:** Mảng chứa các giá trị màu của ảnh.
- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Tạo ra 1 kernel là ma trận trọng số 3x3.
  - o Khai báo 1 mảng các giá trị 0 với kích thước giống với ảnh gốc.

- Sau đó thực hiện vòng lặp duyệt qua từng kênh màu (channel) bằng cách chấp kernel và channel lại. Mỗi kênh màu được em xử lý độc lập.

### 2.6.3. Kết quả:



### 2.6.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã được làm mờ).
- Ảnh mờ tự nhiên, các chi tiết được làm mờ nhẹ nhàng, giúp làm giảm độ nhiễu.
- Thời gian chạy nhanh.

## 2.7. Làm sắc nét ảnh

### 2.7.1. Ý tưởng:

- Thực hiện tương tự như chức năng làm mờ ảnh là tính trung bình cộng giá trị màu của các điểm ảnh trong vùng xung quanh mỗi điểm ảnh, sử dụng một ma trận trọng số gọi là **kernel** bên dưới được em tham khảo tại nguồn [3].

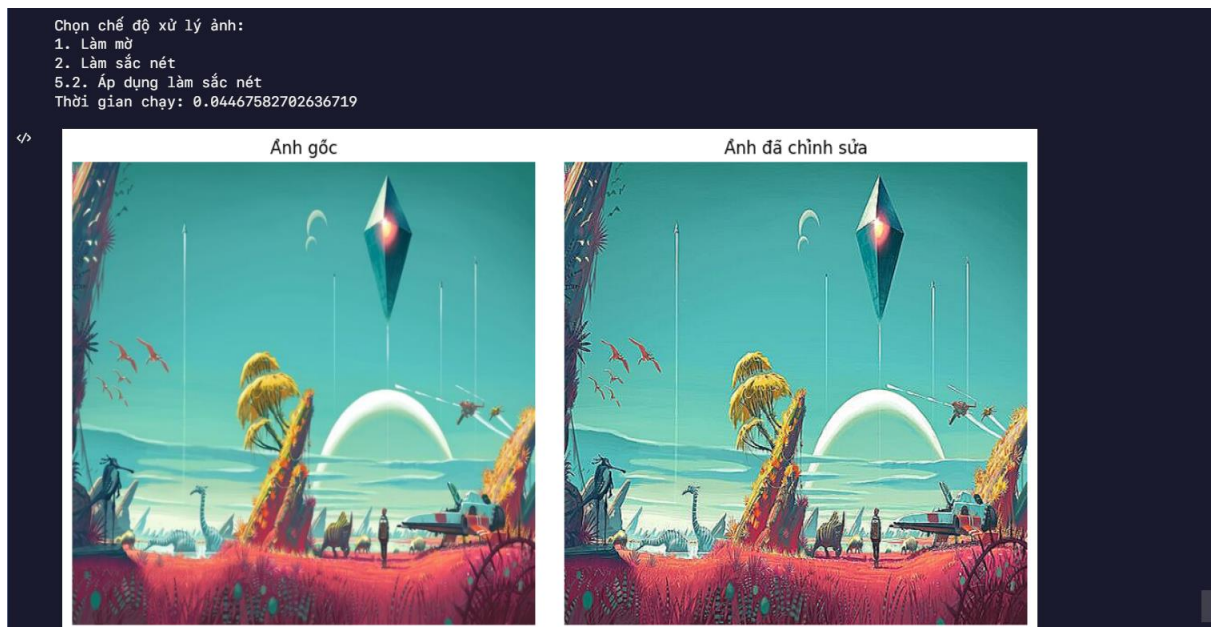
0	-1	0
-1	5	-1
0	-1	0

### 2.7.2. Mô tả:

- Hàm **sharpen\_image** sẽ là hàm chính thực hiện làm sắc nét ảnh.
- *Input:*
  - **image\_path:** Tên của tập tin ảnh.

- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Mở ảnh, chuyển sang mảng Numpy.
  - o Áp dụng phép làm sắc nét Sharpen bằng hàm **sharpen**.
  - o Sử dụng **np.clip** để đảm bảo các giá trị pixel nằm trong khoảng [0,255].
  - o Cuối cùng, chuyển đổi tượng lại thành ảnh PIL bằng **Image.fromarray**.
- Hàm **sharpen** sẽ là hàm hỗ trợ thực hiện phép làm sắc nét Sharpen.
- *Input:*
  - o **img**: Mảng chứa các giá trị màu của ảnh.
- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Tạo ra 1 kernel là ma trận trọng số 3x3.
  - o Khai báo 1 mảng các giá trị 0 với kích thước giống với ảnh gốc.
  - o Sau đó thực hiện vòng lặp duyệt qua từng kênh màu (channel) bằng cách chập kernel và channel lại. Mỗi kênh màu được em xử lý độc lập.

### 2.7.3. Kết quả:



### 2.7.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã được làm sắc nét).



- Ảnh sắc nét hơn, các chi tiết được làm nổi bật, các đường nét trong ảnh làm cho các đối tượng chân thực.
- Thời gian chạy nhanh.

## 2.8. Cắt ảnh theo kích thước (cắt ảnh ở trung tâm)

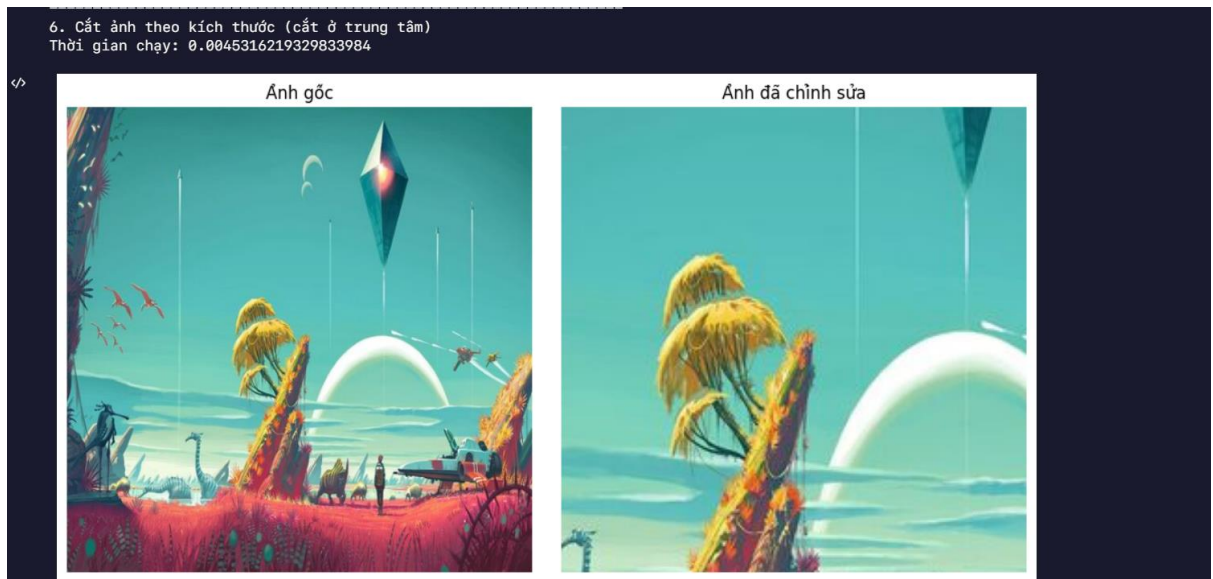
### 2.8.1. Ý tưởng:

- Tính toán điểm bắt đầu nằm ở trung tâm ảnh và điểm kết thúc để ảnh được cắt đúng theo kích thước mới.

### 2.8.2. Mô tả:

- *Input:*
  - o **image\_path:** Tên của tập tin ảnh.
  - o **new\_width:** Kích thước mới của ảnh sau khi cắt (chiều rộng).
  - o **new\_height:** Kích thước mới của ảnh sau khi cắt (chiều cao).
- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Mở ảnh, chuyển sang mảng Numpy.
  - o Lấy kích thước ảnh gốc bằng **shape**.
  - o Tính toán điểm bắt đầu (**start\_x** và **start\_y**) để cắt ảnh sao cho phần ảnh cắt được nằm ở trung tâm của ảnh gốc. Em thực hiện bằng cách tính trung bình của các kích thước và trừ đi kích thước mới (**new\_width** và **new\_height**) chia 2.
  - o Tính toán điểm kết thúc (**end\_x** và **end\_y**) để cắt ảnh sao cho phần ảnh cắt có kích thước mới (**new\_width** và **new\_height**).
  - o Truy xuất phần ảnh cần cắt từ mảng **image\_array** bằng cách sử dụng slicing<sup>[4]</sup> với các chỉ số **start\_x**, **start\_y**, **end\_x** và **end\_y**.
  - o Cuối cùng, chuyển đối tượng lại thành ảnh PIL bằng **Image.fromarray**.

### 2.8.3. Kết quả:



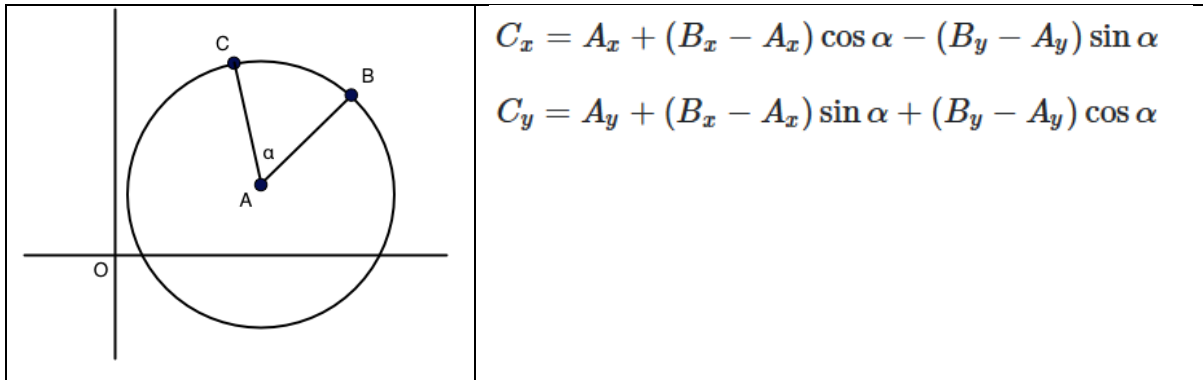
#### 2.8.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã cắt đúng kích thước và ở trung tâm).
- Ảnh được cắt đúng vị trí và kích thước, không làm giảm chất lượng hay gây mất mát thông tin quan trọng trong ảnh.
- Thời gian chạy nhanh.

## 2.9. Cắt ảnh theo khung tròn

### 2.9.1. Ý tưởng:

- Em có 2 cách:
  - o Cách 1:
    - Sử dụng một mask (mặt nạ) để xác định vùng ảnh cần cắt. Mask được tạo bằng cách duyệt qua từng điểm ảnh trong ảnh gốc, sau đó tính toán tọa độ xoay cho mỗi điểm. Nếu tọa độ điểm xoay thuộc vùng trong khung tròn (kiểm tra bằng cách so sánh khoảng cách từ điểm xoay đến tâm của khung tròn với bán kính), thì giá trị tương ứng trong mask sẽ được đánh dấu là 255 (trắng). Công thức tọa độ xoay như sau<sup>[5]</sup>.



○ Cách 2:

- Vì em thấy không cần thiết phải xoay các điểm mà có thể dùng cách khác đó là tính khoảng cách từ mỗi điểm trên lưới đến tâm của hình tròn (ở giữa ảnh) bằng cách dùng khoảng cách Euclidean từ mỗi điểm đến tâm của hình tròn và chỉ giữ lại những điểm có khoảng cách đến tâm nhỏ hơn hoặc bằng bán kính.

### 2.9.2. Mô tả:

- Cách 1:

- Hàm **crop\_image\_with\_circle\_1** sẽ là hàm chính thực hiện cắt ảnh theo khung tròn.

- *Input:*

- **image\_path:** Tên của tập tin ảnh.
- **x\_center:** Tọa độ x của tâm của khung tròn.
- **y\_center:** Tọa độ y của tâm của khung tròn.
- **radius:** Bán kính của khung tròn.
- **angle:** Góc xoay (đơn vị: độ) của ảnh quanh tâm của khung tròn.

- *Output:*

- Ảnh đã được xử lý.

- *Các bước thực hiện:*

- Mở ảnh, chuyển sang mảng Numpy.
- Tạo một mask có kích thước tương đương với ảnh gốc, ban đầu toàn bộ các giá trị đều là 0 (màu đen).
- Duyệt qua từng điểm ảnh trong ảnh gốc và tính toán tọa độ xoay của mỗi điểm dựa trên hàm **rotate\_point\_circle** với thông số tâm, bán kính và góc truyền vào.

- Nếu tọa độ điểm xoay nằm trong khung tròn (*kiểm tra bằng cách so sánh khoảng cách từ điểm xoay đến tâm với bán kính với công thức  $(x-x_0)^2 + (y-y_0)^2 \leq R^2$* ), thì giá trị tương ứng trong mask sẽ được đánh dấu là 255 (trắng).
- Sử dụng mask để cắt ảnh gốc theo vùng trong khung tròn và tạo thành mảng mới bằng phép đối chiếu (**np.where**).
- Cuối cùng, chuyển đối tượng lại thành ảnh PIL bằng **Image.fromarray**.
- Hàm **rotate\_point\_circle** sẽ là hàm hỗ trợ dùng để xoay các tọa độ.
- *Input:*
  - **x**: Tọa độ x của điểm cần xoay.
  - **y**: Tọa độ y của điểm cần xoay.
  - **angle\_deg**: Góc xoay (đơn vị: độ).
  - **x\_center**: Tọa độ x của tâm của khung tròn.
  - **y\_center**: Tọa độ y của tâm của khung tròn.
- *Output:*
  - Tọa độ của điểm sau khi đã xoay quanh tâm của khung tròn.
- *Các bước thực hiện:*
  - Chuyển đổi góc xoay từ đơn vị độ sang radian bằng hàm **np.radians**. Góc xoay em chọn là 45 độ.
  - Tính toán tọa độ của điểm sau khi xoay xung quanh tâm của khung tròn theo công thức:

$$x_{rot} = (x - x_{center}) * \cos(\text{angle\_rad}) - (y - y_{center}) * \sin(\text{angle\_rad}) + x_{center}$$

$$y_{rot} = (x - x_{center}) * \sin(\text{angle\_rad}) + (y - y_{center}) * \cos(\text{angle\_rad}) + y_{center}$$

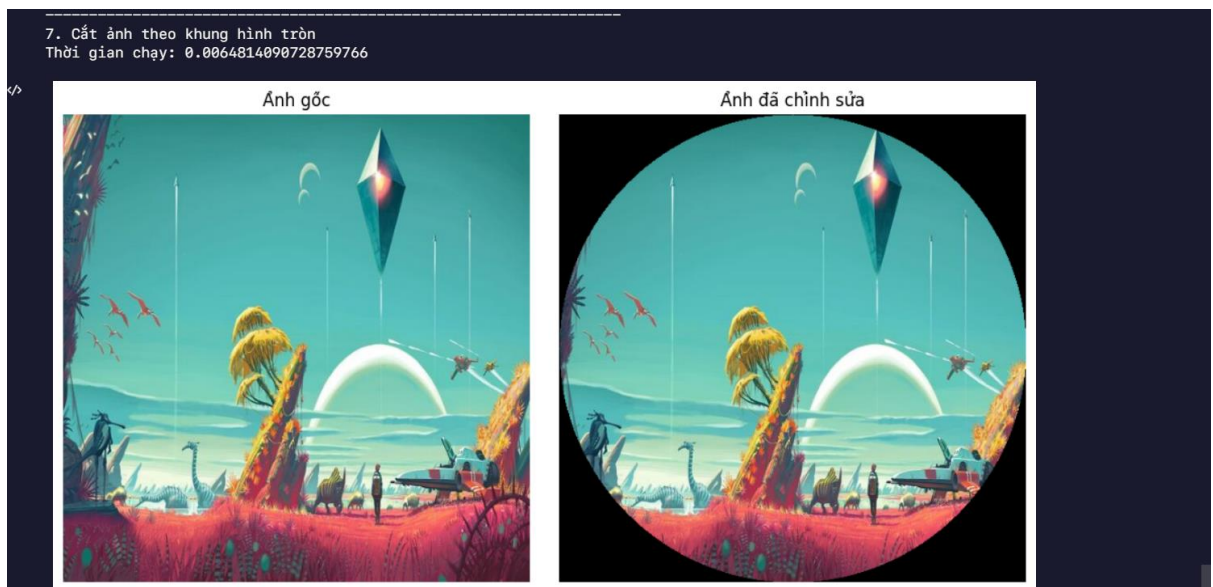
$$x_{center}, y_{center} = 256, 256$$

- Cách 2:
- Hàm **crop\_image\_with\_circle\_2** sẽ là hàm chính thực hiện cắt ảnh theo khung tròn.
- *Input:*
  - **image\_path**: Tên của tập tin ảnh.
  - **radius**: Bán kính khung tròn.
- *Output:*



- Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - Mở ảnh, chuyển sang mảng Numpy.
  - Tính toán chiều rộng và chiều cao của ảnh.
  - Tạo một lưới 2D với kích thước tương ứng với ảnh, để tính toán khoảng cách từ mỗi điểm đến tâm của hình tròn bằng cách dùng **np.ogrid**.
  - Tính khoảng cách từ mỗi điểm trên lưới đến tâm của hình tròn, dựa trên vị trí x và y của điểm.
  - Tạo một mask (mặt nạ) để chỉ giữ lại những điểm nằm trong khung tròn, bằng cách so sánh khoảng cách tính được ở bước trước với bán kính radius.
  - Cắt ảnh dựa trên mask đã tạo, bằng cách đặt màu đen cho các điểm không nằm trong khung tròn.
  - Cuối cùng, chuyển đối tượng lại thành ảnh PIL bằng **Image.fromarray**.

### 2.9.3. Kết quả:



### 2.9.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã cắt đúng thành khung hình tròn).
- Thời gian chạy nhanh.

## 2.10. Cắt ảnh theo khung là 2 elip chéo nhau

### 2.10.1. Ý tưởng:

- Sử dụng một mask (mặt nạ) để xác định vùng ảnh cần cắt. Mask được tạo bằng cách duyệt qua từng điểm ảnh trong ảnh gốc, sau đó tính toán tọa độ xoay cho mỗi điểm theo hai hình elip. Nếu một điểm nằm trong một trong hai hình elip (tức là nó thỏa mãn phương trình của elip của em), thì điểm tương ứng trong mask sẽ được đặt thành màu trắng, ngược lại, giữ nguyên giá trị màu đen. Công thức tọa độ xoay như sau<sup>[6]</sup>.

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

### 2.10.2. Mô tả:

- Hàm **crop\_image\_with\_ellipses** sẽ là hàm chính thực hiện cắt ảnh theo khung là hai hình elip chéo nhau.
- *Input:*
  - o **image\_path:** Tên của tập tin ảnh.
  - o **a1:** Bán trục lớn của hình elip thứ nhất.
  - o **b1:** Bán trục nhỏ của hình elip thứ nhất.
  - o **angle1:** Góc xoay (đơn vị là độ) của hình elip thứ nhất.
  - o **a2:** Bán trục lớn của hình elip thứ hai.
  - o **b2:** Bán trục nhỏ của hình elip thứ hai.
  - o **angle2:** Góc xoay (đơn vị là độ) của hình elip thứ hai.
- *Output:*
  - o Ảnh đã được xử lý.
- *Các bước thực hiện:*
  - o Mở ảnh, chuyển sang mảng Numpy.
  - o Tạo một mảng mask có cùng kích thước với ảnh và ban đầu có tất cả các giá trị là 0.
  - o Em duyệt qua từng điểm ảnh trong ảnh gốc. Tại mỗi điểm, tính toán tọa độ sau khi xoay theo hai hình elip sử dụng hàm **rotate\_point\_ellipses**.
  - o Kiểm tra xem mỗi điểm có nằm trong một trong hai hình elip bằng phương trình elip cơ bản ( $x^2/a^2 + y^2/b^2 \leq 1$ ). Nếu điểm nằm trong hình elip, thì ta gán giá trị 255 (màu trắng) cho điểm tương ứng trong mask.
  - o Sử dụng mask để cắt ảnh gốc theo vùng trong khung và tạo thành mảng mới bằng phép đối chiếu (**np.where**).

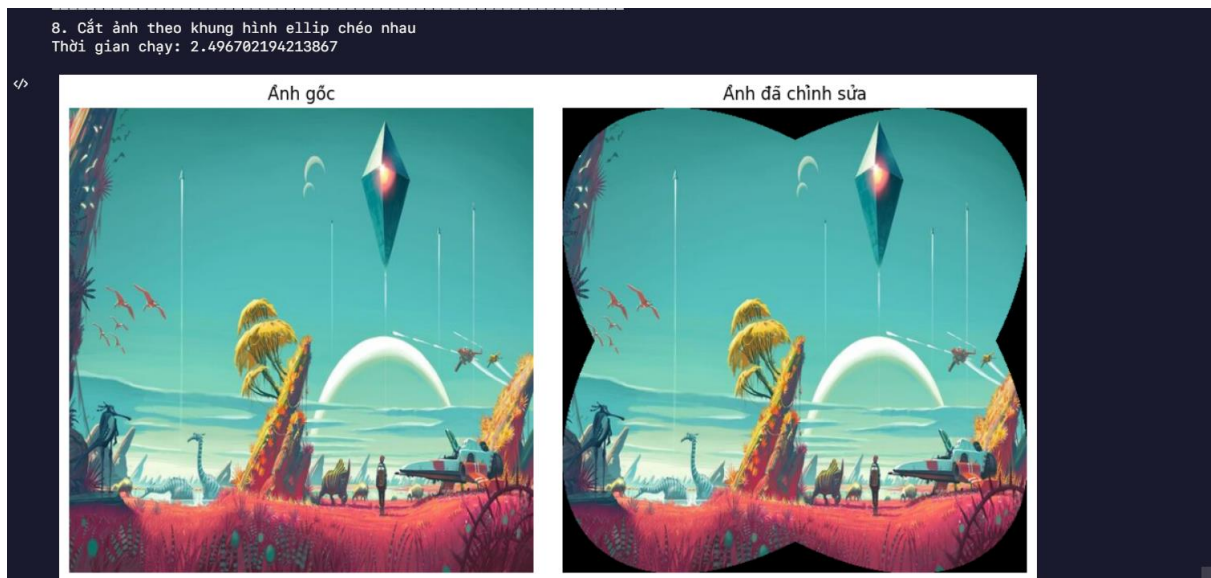
- Cuối cùng, chuyển đổi tượng lại thành ảnh PIL bằng **Image.fromarray**.
- Hàm **rotate\_point\_ellipses** sẽ là hàm hỗ trợ dùng để xoay các tọa độ.
- *Input:*
  - **x**: Tọa độ x của điểm cần xoay.
  - **y**: Tọa độ y của điểm cần xoay.
  - **angle\_deg**: Góc xoay (đơn vị: độ).
- *Output:*
  - Tọa độ của điểm sau khi đã xoay quanh tâm của khung tròn.
- *Các bước thực hiện:*
  - Chuyển đổi góc xoay từ đơn vị độ sang radian bằng hàm **np.radians**. Góc xoay của hình elip 1 em chọn là 45 độ. Góc xoay của hình elip 2 em chọn là 135 độ.
  - Tính toán tọa độ của điểm sau khi xoay xung quanh tâm của khung tròn theo công thức:

$$x_{rot} = x * \cos(\text{angle\_rad}) - y * \sin(\text{angle\_rad})$$

$$y_{rot} = x * \sin(\text{angle\_rad}) + y * \cos(\text{angle\_rad})$$

$$\mathbf{x, y = x - 256, y - 256}$$

### 2.10.3. Kết quả:



### 2.10.4. Nhận xét:

- Thực hiện đúng yêu cầu của bài toán. Ảnh đã thay đổi (ở đây ảnh đã cắt đúng thành khung 2 hình elip chéo nhau).
- Thời gian chạy khá tương đối.

## 2.11. Hàm main

- Các bước trong hàm main:
  - o Yêu cầu người dùng nhập tên tập tin ảnh để xử lý (image\_path) và chọn chức năng xử lý ảnh (function\_choice).
  - o Thực hiện xử lý ảnh dựa trên các chức năng lựa chọn bởi người dùng:

- a. Thay đổi độ sáng ảnh.
    - b. Thay đổi độ tương phản ảnh.
    - c. Lật ảnh (ngang hoặc dọc).
    - d. Chuyển đổi ảnh sang ảnh xám hoặc ảnh sepia.
    - e. Áp dụng làm mờ hoặc làm sắc nét ảnh.
    - f. Cắt ảnh theo kích thước ở trung tâm.
    - g. Cắt ảnh theo khung hình tròn.
    - h. Cắt ảnh theo khung hình elip chéo nhau.
  - o Trả về kết quả xử lý và lưu ảnh đã xử lý với tên mới tùy thuộc vào chức năng xử lý.

## 3. Nhận xét tổng thể

- Nhìn chung, các chức năng đã được thực hiện khá tốt.
- Về phần thời gian, các chức năng được xử lý tối ưu.
- Về phần chi tiết từng chức năng đã làm đúng theo yêu cầu đề bài.

## 4. Điểm yếu bản thân

- Sau khi thực hiện đồ án này, em nhận ra được một số vấn đề mình cần cải thiện:
  - o Về phần trình bày, em cần trình bày code gọn gàng hơn nữa.
  - o Về phần thuật toán, em cần tìm hiểu những cách tốt để tối ưu thời gian hơn.

→ Cuối cùng, sau khi hoàn thành đồ án này, em xin chân thành cảm ơn cô Phạm Thị Phương Uyên, thầy Vũ Quốc Hoàng, thầy Nguyễn Văn Quang Huy và thầy Lê Thanh Tùng đã dành thời gian và hỗ trợ em trong quá trình làm đồ án này. Nhờ có sự hướng dẫn từ các thầy cô, em đã được học hỏi thêm nhiều kinh nghiệm quý báu. Em rất biết ơn và trân trọng sự giúp đỡ từ phía các thầy cô. Về phần báo cáo, em đã chia từng mục nhỏ thành các Bookmark để thầy cô dễ dàng theo dõi và đọc. Mong rằng điều này đã giúp làm cho báo cáo trở nên rõ ràng và dễ hiểu hơn. Một lần nữa, em xin gửi lời cảm ơn và lời chúc tốt đẹp đến với thầy cô.



## TÀI LIỆU THAM KHẢO

- [0]: Tài liệu của giáo viên
- [1]: Grayscale to RGB Conversion from Tutorialspoint:  
[https://www.tutorialspoint.com/dip/grayscale\\_to\\_rgb\\_conversion.htm](https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm)
- [2]: Using numpy to apply a sepia effect to a 3D array from Stackoverflow:  
<https://stackoverflow.com/questions/23802725/using-numpy-to-apply-a-sepia-effect-to-a-3d-array>
- [3]: Kernel (image processing) from Wikipedia:  
[https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))
- [4]: Python List Slicing by riturajsaha from Geeksforgeeks:  
<https://www.geeksforgeeks.org/python-list-slicing/>
- [5]: Rotate a point on a circle with known radius and position from math.stackexchange:  
<https://math.stackexchange.com/questions/1384994/rotate-a-point-on-a-circle-with-known-radius-and-position>
- [6]: The Journal of Online Mathematics and Its Applications, Volume 8 (2008)  
- The Most Marvelous Theorem in Mathematics, Dan Kalman: **General Equation of an Ellipse**:  
[https://www.maa.org/external\\_archive/joma/Volume8/Kalman/General.html](https://www.maa.org/external_archive/joma/Volume8/Kalman/General.html)
- NumPy Documentation: <https://numpy.org/doc/>

---HẾT---