ĐẠI HỌC QUỐC GIA – THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN THỰC HÀNH 2: IMAGE PROCESSING

MÔN HỌC: TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CÔNG NGHỆ THÔNG TIN - LỚP: 22CLC10

GIẢNG VIÊN LÝ THUYẾT:
Thầy Vũ Quốc Hoàng

GIẢNG VIÊN THỰC HÀNH:
Cô Phan Thị Phương Uyên
Thầy Nguyễn Ngọc Toàn
Thầy Nguyễn Văn Quang Huy

THÔNG TIN SINH VIÊN:
Họ và tên: Trần Ngọc Uyển Nhi
MSSV: 22127313

Thành phố Hồ Chí Minh, 2024

MỤC LỤC

I. Mô Tả Ý Tưởng Cài Đặt Các Phương Thức Xử Lý Ảnh

1. Điều chỉnh Độ sáng

Thông thường, một tấm ảnh mà chúng ta nhìn thấy được ở trên các thiết bị điện tử sẽ được hình thành từ sự kết hợp giữa 3 kênh màu RGB – Red, Green, Blue. Mỗi kênh màu sẽ được lưu bằng 8 bit nên chúng có giá trị từ 0 đến 255, với mỗi giá trị kết hợp từ 3 kênh màu đều sẽ cho ra những màu sắc khác nhau.

Figure 1: Ví dụ về sự kết hợp màu sắc giữa 3 kênh màu RGB

Có một điểm đáng lưu ý ở đây chính là độ sáng của một tấm ảnh bình thường sẽ có sự liên quan chặt chẽ tới giá trị của các kênh màu. Hãy nhìn vào ví dụ được thực hiện ở trang 1 trang web RGB Calculator [1] dưới đây, chúng ta sẽ rút ra được kết luận rằng nếu giá trị của các kênh màu càng lớn thì màu sắc được hiển thị sẽ càng "sáng", và ngược lại nếu giá trị càng nhỏ thì màu sắc sẽ càng "tối".

Figure 2: Mối quan hệ của độ sáng của màu với giá trị của các kênh màu

Từ kết luận được rút ra ở trên, em nhận thấy rằng để có thể được điều chỉnh độ sáng tối của ảnh thì về mặt lý thuyết chúng ta cần phải tăng thêm hoặc giảm đi giá trị của các kênh màu. Còn về mặt thực hành việc tăng độ sáng với 1 tấm ảnh gồm nhiều điểm màu, em đã tham khảo ở trang web của thư viện OpenCV [2] và có được nhận định như sau: Giả sử như giá trị của điểm ảnh tại vị trí dòng thứ i, cột thứ j được biểu diễn bằng biểu thức $f(i,j)$ (có giá trị trong đoạn $[0,255]$) thì giá trị sau khi xử lý điểm ảnh là $g(i,j)$, chúng ta sẽ có được biểu thức $g(i,j) = \alpha f(i,j) + \beta$. Trong đó α và β lần lượt là các tham số điều chỉnh độ tương phản và sáng tối của điểm màu đó. Áp dụng biểu thức này cho toàn bộ điểm màu trên ảnh chúng ta sẽ thay đổi được độ sáng tối của tấm ảnh dựa trên mong muốn.

Figure 3: Một ví dụ của thao tác cộng hoặc trừ 1 số nguyên cho toàn bộ giá trị RGB của ảnh để điều chỉnh độ sáng của ảnh

Để cài đặt chức năng này bằng thư viện python, chúng ta sẽ trước tiên chuyển bức ảnh từ người dùng thành 1 ma trận - ndarray. Vì ndarray hỗ trợ kỹ thuật broadcasting, chúng ta chỉ cần cộng ma trận ảnh với một giá trị β, và numpy sẽ tự động cộng giá trị đó cho toàn bộ các phần tử của ma trận. Sau khi thực hiện thao tác này, em sử dụng hàm numpy.clip() để giới hạn lại giá trị của các điểm ảnh, đảm bảo chúng nằm trong khoảng $[0,255]$. Ngoài ra nếu thầy cô đọc phần code của em có thể sẽ thắc mắc tại sao em lại chuyển kiểu dữ liệu của ma trận ảnh thành 'float' trước khi cộng, đó là vì em muốn cho phép người dùng có thể nhập vào giá trị β là số thực và ma trận ảnh cũng nên là số thực để có thể thực hiện thao tác cộng trừ giữa các giá trị có cùng kiểu dữ liệu với nhau (Trong quá trình kiểm tra lại em nhận ra python sẽ tự động chuyển ma trận ảnh từ kiểu dữ liệu 'uint8' thành kiểu 'float' nếu như em cộng trừ ma trận với 1 số 'float', nhưng để đảm bảo tính bao quát đúng đắn em vẫn sẽ chuyển ma trận thành 'float' trước rồi mới xử lý). Sau khi dùng hàm numpy.clip() em sẽ cast ma trận lại thành kiểu dữ liệu 'uint8' vì giá trị kênh màu là các số nguyên dương từ 0 đến 255, nếu đọc được số thực hay giá trị điểm màu ngoài khoảng này, chương trình sẽ báo lỗi.

Dưới đây là kết quả của em trong việc cài đặt chức năng này:

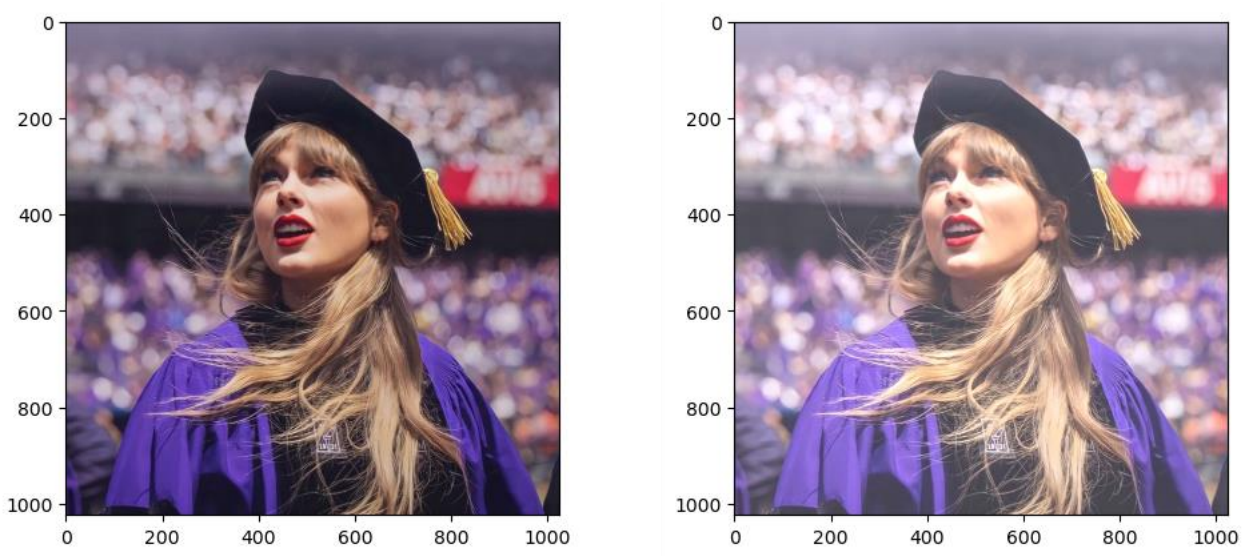


Figure 4: Ảnh gốc và Ảnh sau khi tăng độ sáng lên 50

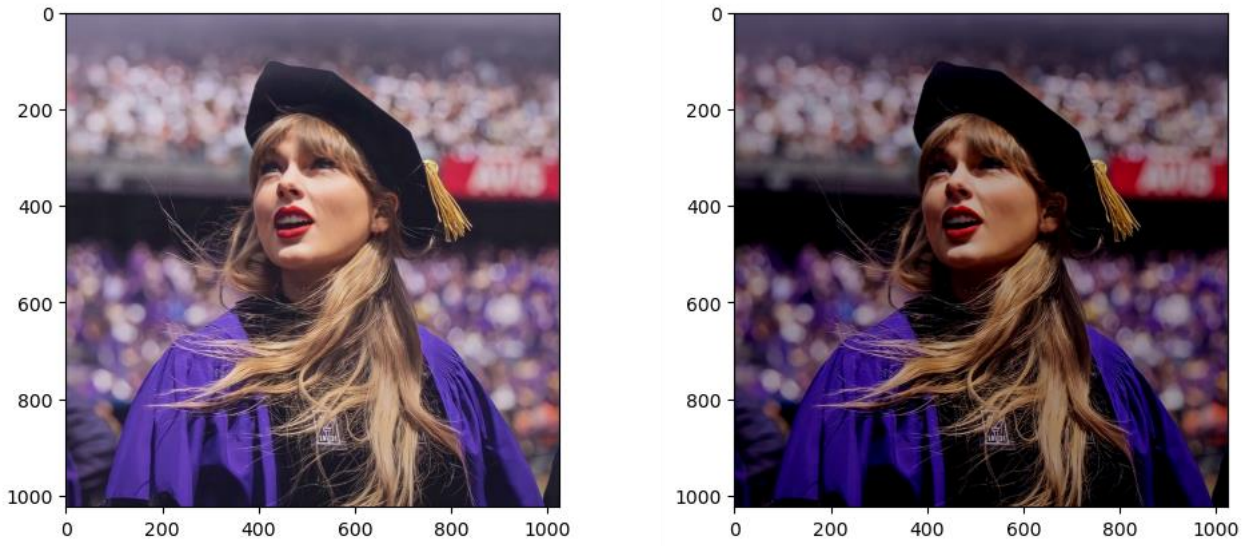


Figure 5: Ảnh gốc và Ảnh sau khi giảm độ sáng đi 50

2. Điều chỉnh Độ tương phản

Để thực hiện được chức năng này, trước tiên chúng ta cần hiểu Độ tương phản là gì? Độ tương phản nói đến khoảng cách giữa các điểm màu trong một ảnh, khi ta tăng độ tương phản tức là ta đang tăng khoảng cách giá trị màu giữa các điểm ảnh. Đôi khi trong quá trình xử lý ảnh, chúng ta sẽ cần tăng hoặc giảm độ tương phản của một tấm ảnh, và như đã nêu ở mục điều chỉnh độ sáng bên trên, để xử lý yêu cầu điều chỉnh độ tương phản, chúng ta chỉ việc nhân toàn bộ điểm ảnh với 1 con số để tăng hoặc giảm độ tương phản của chúng.

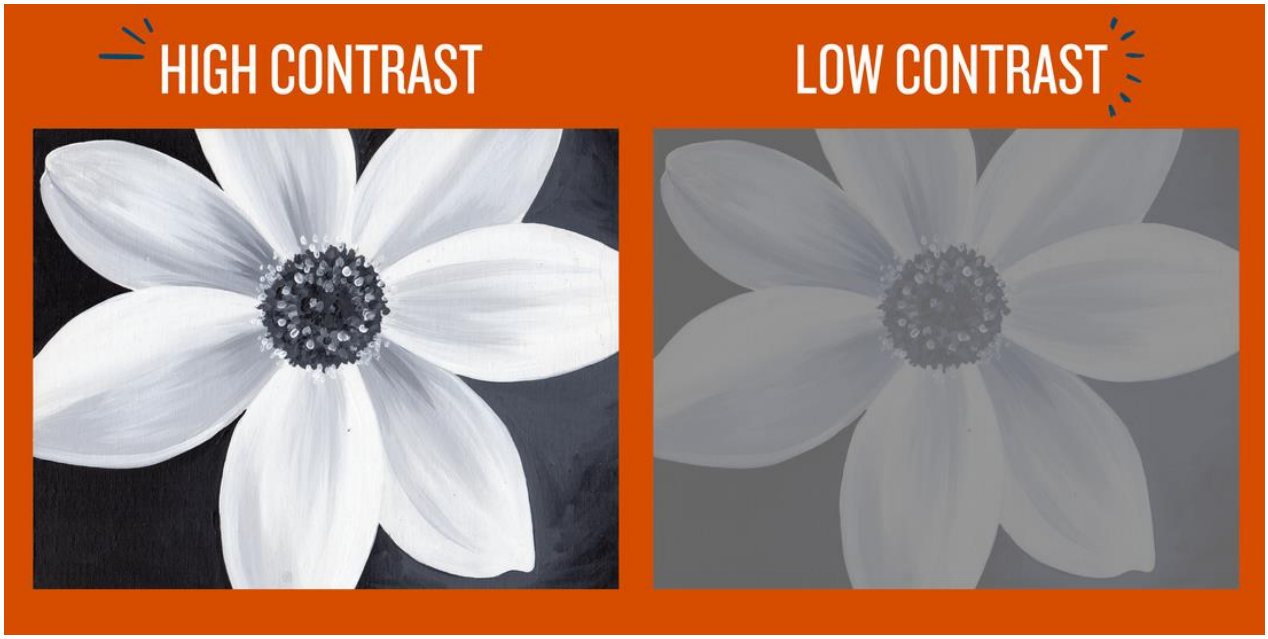


Figure 6: Ví dụ về ảnh có độ tương phản cao và thấp

Để giải thích rõ hơn về việc tăng giảm khoảng cách bằng cách nhân 1 số cho toàn bộ điểm ảnh, em sẽ lấy một ví dụ như sau: Giả sử ta có 2 giá trị điểm màu ban đầu 50 và 100, khoảng cách giữa chúng là abs(50 – 100) = 50.

- **Giảm độ tương phản ($\alpha = 0,5$):** Sau khi nhân với α, giá trị điểm màu mới lần lượt là 25 và 50 và khoảng cách sau khi giảm của chúng là abs(25 – 50) = 25.
- **Tăng độ tương phản ($\alpha = 2$):** Sau khi nhân với α, giá trị điểm màu mới lần lượt là 100 và 200 và khoảng cách sau khi giảm của chúng là abs(100 – 200) = 100.

Trong ví dụ này, nhân các giá trị màu với 0.5 đã làm giảm khoảng cách giữa chúng từ 50 xuống còn 25, do đó giảm độ tương phản của ảnh. Ngược lại, việc nhân các giá trị màu với 2 đã làm tăng khoảng cách giữa chúng từ 50 lên 100, do đó tăng độ tương phản của ảnh.

Để thực hiện được chức năng này, em sẽ thực hiện thao tác hoàn toàn giống với chức năng Điều chỉnh Độ sáng như bên trên, nhưng thay vì thực hiện phép cộng, em sẽ thực hiện phép nhân cho toàn bộ ma trận nhờ broadcasting của ndarray. Ở chức năng việc chuyển kiểu dữ liệu của ma trận thành ‘float’ sẽ giúp chương trình tính toán chính xác hơn giá trị mới của các điểm màu.

Dưới đây là kết quả của em trong việc cài đặt chức năng này:

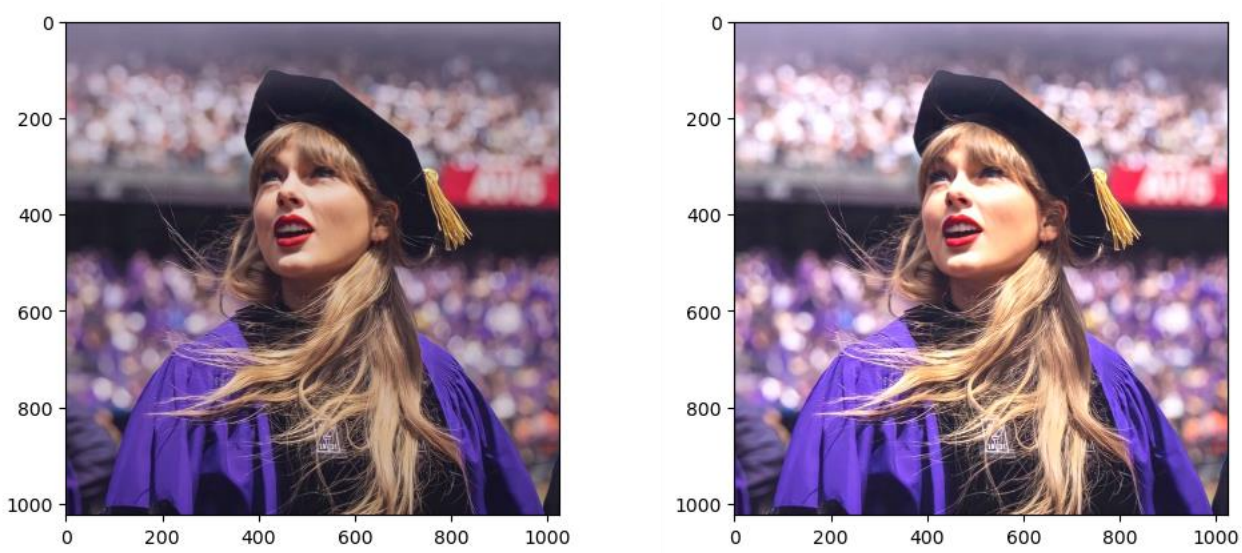


Figure 7: Ảnh gốc và Ảnh sau khi tăng độ tương phản (alpha = 1.2)

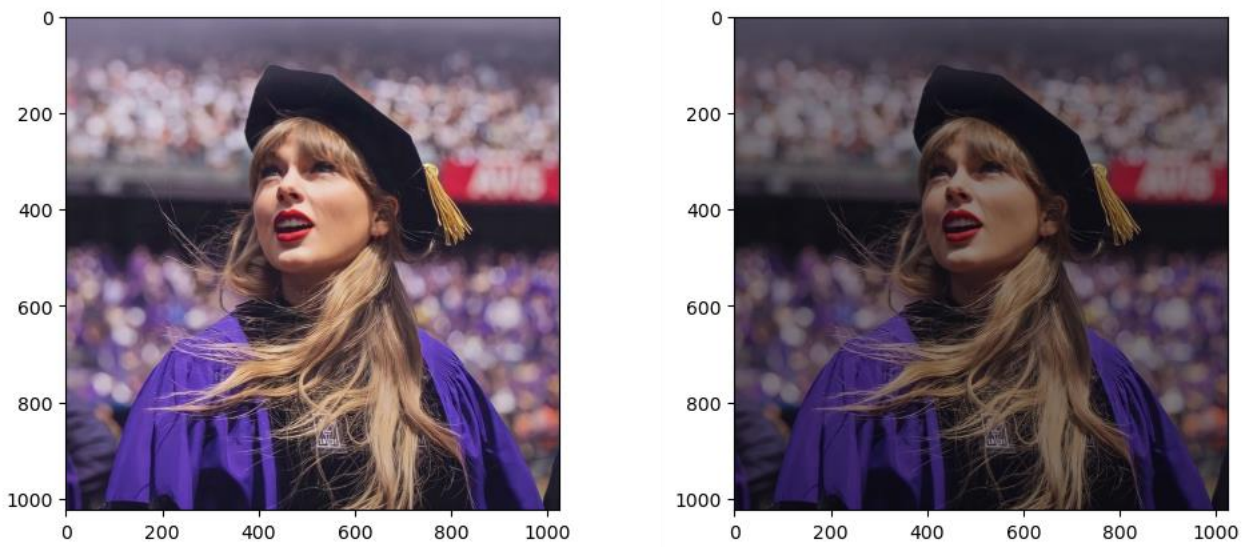


Figure 8: Ảnh gốc và Ảnh sau khi giảm độ tương phản (alpha = 0.6)

3. Lật Ảnh (Chiều ngang và Chiều dọc)

Việc lật ảnh ngang hay dọc có thể hiểu là chúng ta sẽ giữ nguyên các giá trị của các điểm ảnh nhưng đổi vị trí của các điểm ảnh này sang vị trí đối xứng thông qua trục nằm ngang hoặc nằm dọc của tấm ảnh. Thư viện numpy có 2 hàm hỗ trợ trong việc xử lý chức năng này, đó chính là hàm numpy.fliplr() và hàm numpy.flipud().

numpy.fliplr() [8] là viết tắt của flip left to right, tức là lật từ trái sang phải, giúp đảo ngược vị trí cột của các điểm ảnh, làm ảnh lật ảnh theo chiều ngang và hàm này tương đương với việc sử dụng slicing img_2d[:,::-1,:]. Do việc đảo vị trí sang chiều ngang chính là đảo vị trí các cột của ảnh (từ 0 đầu sang cột n – 1, cột 1 sang n – 2,…) nên chúng ta sẽ thêm lại các cột (chiều thứ 2 của ma trận theo thứ tự ngược) sang ma trận mới.

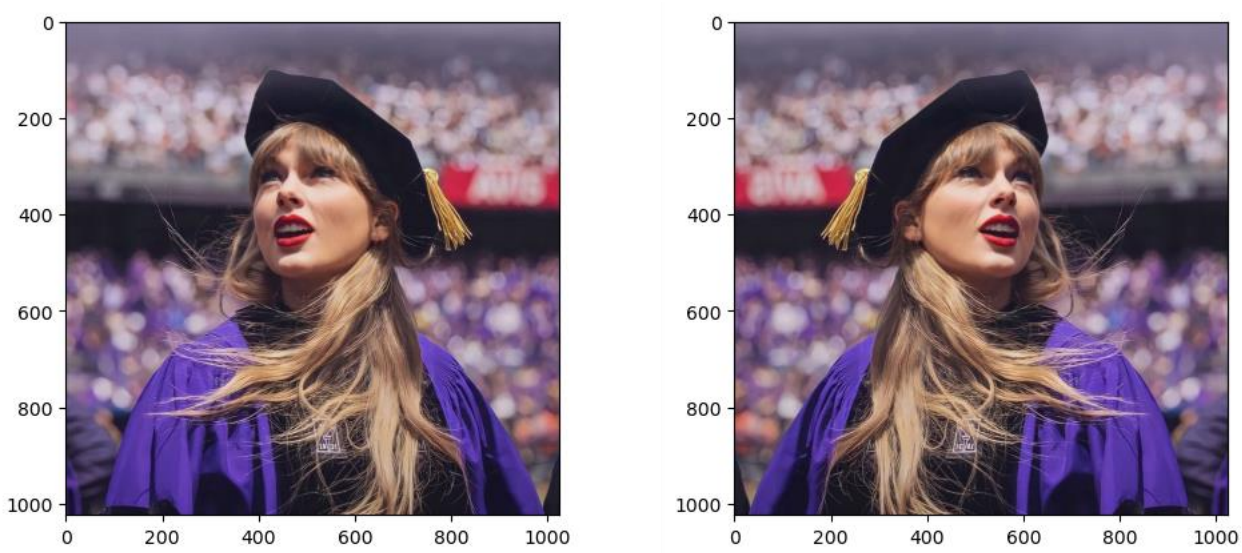


Figure 9: Ảnh gốc và Ảnh sau lật chiều ngang (horizontally)

Tương tự, numpy.flipud() [7] là viết tắt của flip up to down, tức là lật từ trên xuống dưới, giúp đảo ngược vị trí dòng của các điểm ảnh, làm ảnh lật ảnh theo chiều dọc và hàm này tương đương với việc sử dụng slicing img_2d[::-1,:,:] - đảo vị trí các dòng của ảnh (từ dòng 0 sang dòng n – 1, dòng 1 sang n – 2,…) nên chúng ta sẽ thêm lại các dòng (chiều thứ 1 của ma trận theo thứ tự ngược) sang ma trận mới.

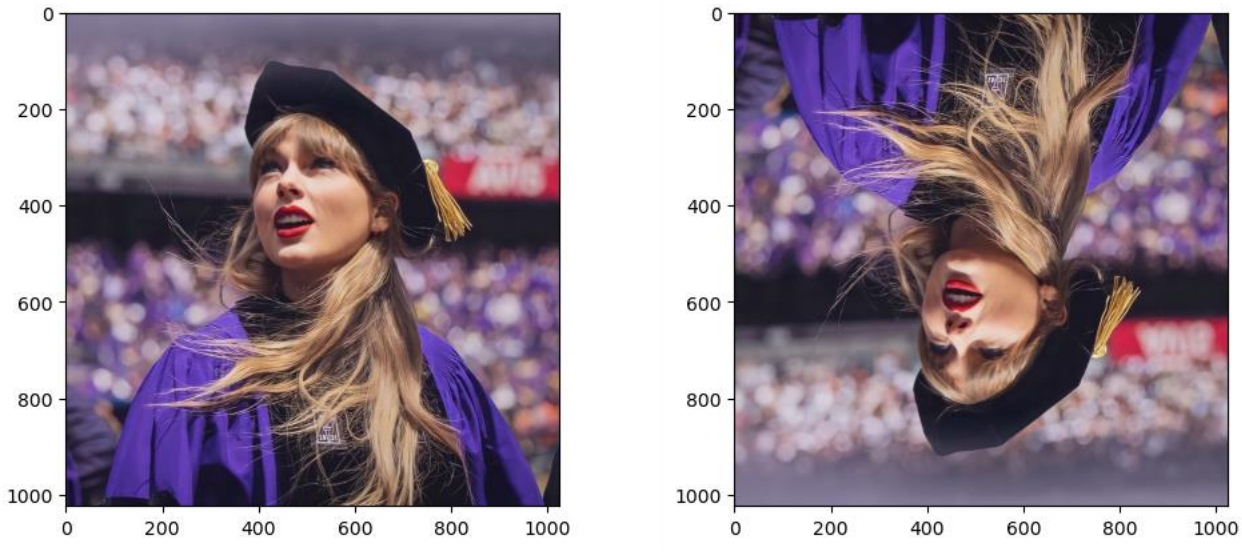


Figure 10: Ảnh gốc và Ảnh sau lật chiều dọc (vertically)

4. Chuyển đổi Ảnh gốc sang Ảnh Xám hoặc Ảnh Sepia

a. Ảnh xám

Để thực hiện chức năng chuyển từ ảnh RGB bình thường sang dạy Grayscale (Ảnh xám), em đã tham khảo [11] và tìm ra được một số phương pháp để xử lý việc này. Trước tiên, chúng ta cần hiểu cách mà một tấm ảnh xám được tạo ra. Từ nhận định về độ sáng và nhận định ở trang [11], em có thể hiểu đại khái được để tạo ra một tấm ảnh xám chúng ta sẽ "giảm" độ sáng của các màu lại, vì giá trị của các kênh màu càng lớn thì màu sắc của kênh màu đó cũng sẽ được thể hiện rõ ràng hơn đối với mắt người. Ví dụ như RGB(255,0,0) sẽ

thể hiển ra điểm màu màu đỏ chính xác nhất và "rõ" nhất; (128,0,0) sẽ cho ra một màu đỏ nhưng đã bớt rực rỡ đi, người ta còn gọi đây là màu đỏ nâu. Và khi sang đến màu (240,128,128) chúng ta sẽ thấy nó đã biến thành màu hồng nhẹ, hay còn gọi là màu san hô, điểm màu này vẫn giữ được màu hồng là vì giá trị kênh màu đỏ của nó là lớn nhất, thêm vào đó các giá trị màu Green và Blue đã làm cho sắc đỏ có thiên hướng được dịu đi. Chung quy lại là, màu sắc sẽ càng rõ ràng và rực rỡ nếu giá trị của chúng càng lớn, và cũng như vậy, ảnh xám chính là ảnh có giá trị các kênh màu nhỏ nhưng đủ sáng để ta có thể nhận biết được "nội dung" của tấm ảnh.

Color	Color Name	HEX Code	RGB Code
	Black	#000000	rgb(0, 0, 0)
	Maroon	#800000	rgb(128, 0, 0)
	DarkRed	#8b0000	rgb(139, 0, 0)
	Red	#ff0000	rgb(255, 0, 0)
	FireBrick	#b22222	rgb(178, 34, 34)
	Brown	#a52a2a	rgb(165, 42, 42)
	IndianRed	#cd5c5c	rgb(205, 92, 92)
	RosyBrown	#bc8f8f	rgb(188, 143, 143)
	LightCoral	#f08080	rgb(240, 128, 128)

Figure 11: Ví dụ về độ rõ của các kênh màu có quan hệ với giá trị kênh màu đó

Một trong những cách để tạo ra tấm ảnh xám [4] đó chính là sử dụng phương pháp Average method, phương pháp này nói rằng chúng ta hãy lấy giá trị trung bình của 3 kênh màu trong điểm ảnh ban đầu và gán nó lại cho từng kênh màu của điểm ảnh. Ví dụ như điểm ảnh ban đầu là [120, 240, 60], sau khi sử dụng Average method nó sẽ chuyển thành [140, 140, 140]. Đây là một phương pháp dễ hiểu, song lại không hiệu quả [12] vì dựa trên một số nghiên nghiên cứu về thị giác con người, chúng ta biết rằng mắt của chúng ta phản ứng với mỗi màu theo cách khác nhau. Cụ thể, mắt của chúng ta nhạy cảm hơn với màu xanh lá cây, sau đó là màu đỏ và cuối cùng là màu xanh dương. Do đó, các trọng số trong phương trình trên nên được thay đổi thay vì chúng ta chỉ xem trọng số của chúng là như nhau.

Một phương pháp phổ biến khác và cũng là phương pháp mà em áp dụng để thực hiện chức năng này đó chính là Luminosity method. Phương pháp này cũng sẽ lấy giá trị của 3 kênh màu trong điểm màu, tính toán ra 1 giá trị mới và gán nó lại cho cả ba kênh màu, điểm khác biệt của nó đó chính là trọng số của các kênh màu đã được thay đổi như sau:

$$0.2989 * R + 0.5870 * G + 0.1140 * B$$

Như nhận định bên trên về ảnh hưởng của các kênh màu tới mắt, ta có thể thấy trong biểu thức này, trọng số của kênh màu Green là lớn nhất, tới Red và cuối cùng là tới Blue.

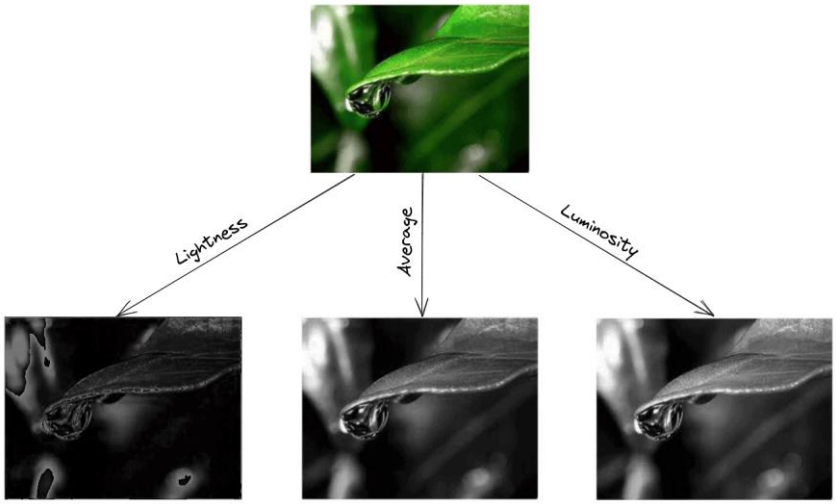


Figure 12: Ví dụ về việc chuyển đổi ảnh xám sử dụng các phương pháp khác nhau

Để cài đặt chức năng này, trước tiên em sẽ sử dụng slicing để lấy ra các mảng giá trị cho từng kênh màu, mảng mới này sẽ có kích thước là height x width x 1 vì hiện tại nó chỉ đang có 1 kênh màu duy nhất. Sau đó thực hiện tính toán ra 1 mảng height x width x 1 mới có giá trị mới là giá trị được tính toán từ 3 kênh màu ban đầu, nhân với trọng số tương ứng cho từng kênh màu giống biểu thức bên trên. Sau đó em sẽ sử dụng hàm numpy.stack() – một hàm thường được sử dụng để ghép các mảng lại theo chiều mong muốn để ghép mảng vừa được mới tạo lần lượt vô vị trí cho từng kênh màu, thay vì chúng ta phải gán lại giá trị cho từng kênh màu bên trên nhưng kết quả vẫn là giống nhau. Em truyền vào 3 ma trận giống nhau và truyền vào tham số axis = -1 để hàm stack() hiểu là em muốn xếp chúng theo chiều cuối cùng (chiều dành cho các kênh màu) mà ma trận mới có thể có từ 3 ma trận ban đầu truyền vào. Em cũng có thực hiện chuyển đổi kiểu dữ liệu ban đầu của ma trận sang ‘float’ vì chúng ta sẽ tính toán với số thực, sau khi tính toán em sẽ dùng hàm clip() để giới hạn lại giá trị của các kênh màu như yêu cầu trên (dù điều này có thể không cần thiết vì tổng của 3 trọng số là 0.2989 + 0.5870 + 0.1140 = 0.9999 < 1), và cuối cùng chuyển giá trị lại về kiểu ‘uint8’ và trả về ảnh mới.

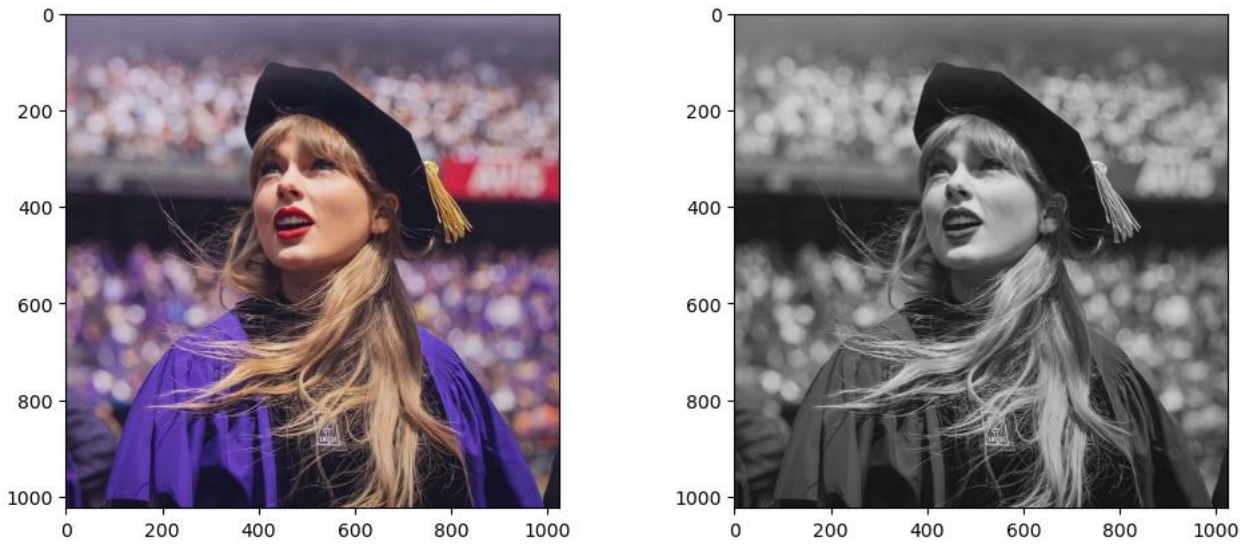


Figure 13: Ảnh gốc và Ảnh xám

b. Ảnh sepia

Tiếp tục thực hiện chức năng chuyển ảnh gốc sang ảnh sepia. Trước tiên ta cần hiểu filter sepia là gì, theo như em tìm hiểu, xử lý màu sepia tức là làm cho bức ảnh có một một tông màu nâu, giống như phong cách cổ điển. Tên gọi "sepia" xuất phát từ sắc tố nâu đen được chiết xuất từ túi mực của con mực nang, vốn được sử dụng phổ biến trong các quá trình nhiếp ảnh ban đầu. Khi nói về quá trình xử lý ảnh sepia ở dạng kỹ thuật số, filter sepia thường điều chỉnh cân bằng màu sắc để chuyển các tông màu sang các sắc thái ấm, nâu, thường với vẻ ngoài hơi mờ và độ tương phản thấp, giúp tạo ra một bức ảnh có phong cách hoài cổ, cũ kỹ và gợi nhớ về thời gian trước đây.

Và để thực hiện được điều này, những nhà nghiên cứu trước đó cũng tìm ra được công thức tính toán ra trọng số cho ra màu ảnh sepia [5] như sau:

```
newRed = 0.393*R + 0.769*G + 0.189*B
newGreen = 0.349*R + 0.686*G + 0.168*B
newBlue = 0.272*R + 0.534*G + 0.131*B
```

Khác một chú so với phương pháp luminosity bên trên, thay vì sử dụng cùng 1 giá trị mới cho tất cả kênh màu, để tạo được màu sepia chúng ta sẽ phải tính toán riêng giá trị mới cho mỗi kênh màu như trong biểu thức trên.

Vì vậy về phần cài đặt em sẽ thực hiện hoàn toàn tương tự phần chuyển sang ảnh xám bên trên, nhưng em sẽ tính riêng giá trị cho kênh màu mới và truyền vào 3 tham số là 3 mảng chứa giá trị màu mới vào hàm stack(). Nếu chúng ta lấy trọng số của các kênh màu cộng lại, ta sẽ thấy chúng đều lớn hơn 1, tức là nếu giá trị của 3 kênh màu ban đầu đều là 255, 255, 255 thì giá trị mới tính được sẽ vượt quá 255, khiến cho việc hiển thị ảnh báo lỗi nên sử dụng hàm numpy.clip() là cần thiết ở trường hợp này. Một lưu ý nhỏ khác là chúng ta nên giới hạn lại giá trị trước khi chuyển giá trị của mảng lại về kiểu 'uint8', nếu không ảnh sẽ hiển thị sai màu, do là vì nếu chúng ta lấy giá trị lớn hơn 255 được chuyển thành unit8, numpy sẽ lấy giá trị đó chia lấy dư cho 255, làm màu sắc bị hiển thị sai. Đó cũng là lý do em sẽ luôn dùng hàm clip() cho ma trận trước khi dùng astype() cho ma trận đó.

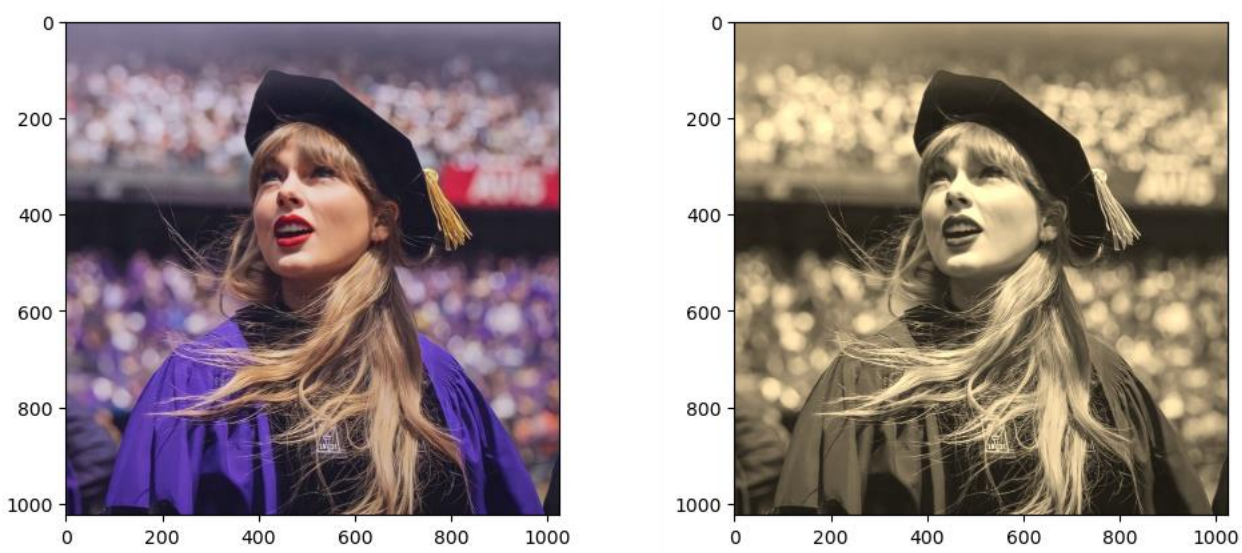


Figure 14: Ảnh gốc và Ảnh sepia

5. Làm mờ và Làm sắc nét ảnh

Để thực hiện được chức năng này, trước tiên ta cần hiểu về một thuật ngữ quan trọng trong việc xử lý ảnh được là kernel. Nói đơn giản Kernel một ma trận thường có kích thước 3x3 hoặc 5x5, được sử dụng để áp dụng các hiệu ứng lên ảnh. Kernel đóng vai trò như một bộ lọc, giúp thực hiện các thao tác như làm mờ, làm sắc nét, khắc nổi cho ảnh và tương ứng với một chức năng giá trị và kích thước của kernel sẽ khác nhau. Quá trình áp dụng một kernel mong muốn lên ảnh được gọi là phép tích chập (convolution). Lấy ví dụ, nếu ta có 1 kernel kích thước 3x3 thì nó sẽ đóng vai trò như 1 "cửa sổ" duyệt lần lượt qua 3x3 điểm ảnh mỗi lần lần lượt trên tấm ảnh, lấy giá trị tương ứng trên kernel nhân với giá trị của điểm ảnh và sau đó cộng lại và trả về giá trị thu được cho điểm ảnh chính giữa (hàng 2 cột 2). Vì nó luôn tính toán và trả về giá trị tính toán được cho điểm ở giữa kernel nên kernel thường bắt buộc phải có kích thước là số lẻ (9 ô, 25 ô,…).

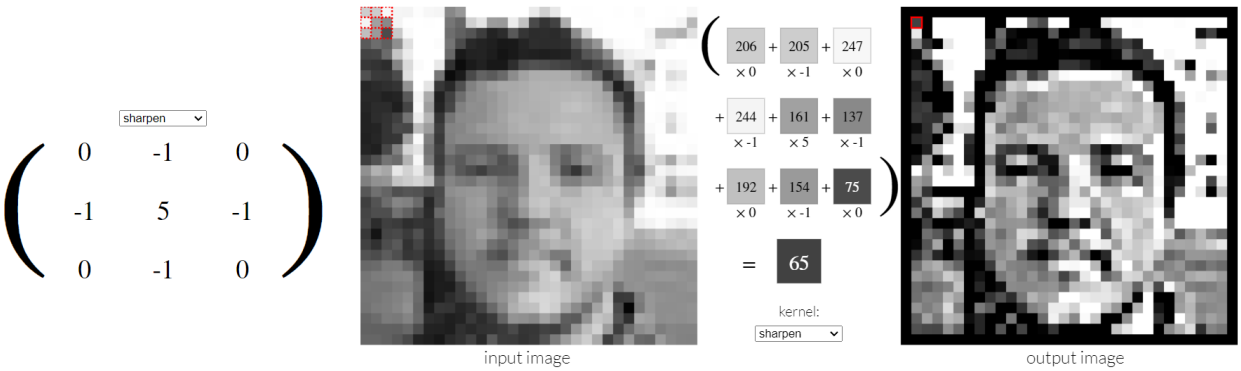


Figure 15: Minh họa cách áp dụng một kernel để làm sắc nét ảnh [13]

Và tương tự, khi áp dụng kernel cho ảnh có 3 kênh màu (thường là Red, Green, Blue - RGB), Kernel sẽ được áp dụng riêng biệt cho từng kênh màu, sử dụng cùng một kernel cho cả 3 kênh R, G, và B, với phép tích chập được thực hiện độc lập trên mỗi kênh. Trong quá trình xử lý, kernel trượt qua toàn bộ kênh màu, thực hiện phép nhân ma trận tại mỗi vị trí, và kết quả của mỗi phép nhân được cộng lại để tạo ra một giá trị pixel mới cho kênh màu

đó. Bằng cách áp dụng cùng một kernel cho từng kênh màu, ta đảm bảo rằng các mối quan hệ giữa các kênh màu được giữ nguyên, giúp bảo toàn thông tin màu sắc của ảnh gốc trong ảnh kết quả.

Để thực hiện được chức năng làm mờ và làm sắc nét, chúng ta cần chọn ra kernel được dùng cho chức năng tương ứng. Ví dụ như đối với chức năng làm mờ ta có thể chọn kernel = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]]) / 9 hoặc 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]]) / 256 và với chức năng làm sắc nét ta có thể chọn kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]]).

Sau khi tìm được kernel phù hợp chúng ta sẽ giải quyết vấn đề tiếp là tính toán ma trận mới dựa trên kernel được truyền vào. Ý tưởng ban đầu của em là lấy từng cụm điểm ảnh từ ma trận ảnh có kích thước bằng với kích thước kernel, nhân với ma trận kernel và lấy giá trị trả về gán cho điểm ảnh trung tâm như sau:

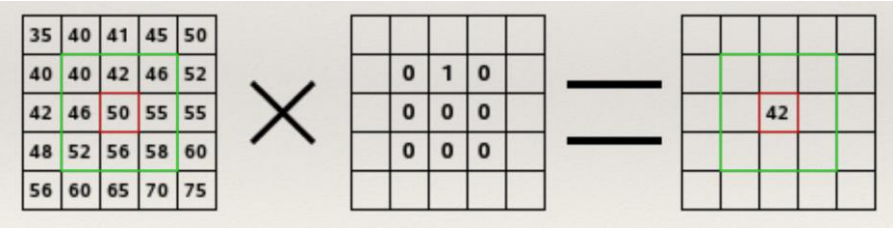


Figure 16: Minh họa cách áp dụng một kernel để làm sắc nét ảnh [14]

Nhưng để làm được điều đó chúng ta sẽ cần sử dụng 3 vòng for, 1 vòng for duyệt qua hàng, 1 vòng duyệt qua cột và 1 dòng duyệt qua kênh màu, điều này sẽ khiến việc xử lý trở nên mất nhiều thời gian hơn. Vậy nên em đã tìm hiểu và tìm ra một phương pháp xử lý nhanh cho chức năng này là sử dụng sliding windows [10] có cách thực hiện tương tự nhưng cải thiện được tốc độ so với việc duyệt qua vòng lặp for. Sliding windows là một cửa sổ trượt có thể được áp dụng trong việc duyệt kernel qua từng điểm ảnh. Như ví dụ visualizer ở trang web [13], ta có thể thấy kernel hoạt động như 1 filter hay 1 cửa sổ đi lần lượt qua từng cụm điểm ảnh để tính toán ra giá trị mới, vậy nên ý tưởng để thực hiện của em cũng là tạo một sliding windows trượt qua từng cụm điểm ảnh để xử lý.

Có 1 function trong numpy có thể tạo ra được cửa sổ này đó chính là numpy.lib.stride_tricks.as_strided(). Hàm này nhận vào các tham số là padded_image (Sử dụng as_strided để tạo một mảng mới có hình dạng và strides cụ thể từ một mảng đã có mà không sao chép dữ liệu), tham số shape để xác định hình dạng của mảng đầu ra và được tạo ra bằng câu lệnh shape = (image_height, image_width, kernel_height, kernel_width, channels), cuối cùng strides là một tuple giúp xác định cách di chuyển trong bộ nhớ để tạo ra các phần tử của mảng mới. Kết quả là một mảng chứa các cửa sổ trượt (sliding windows) của padded_image. Chúng ta sẽ đi tìm từng tham số truyền vào của mảng sau đây.

Trước khi duyệt qua tấm ảnh, để kernel có thể hoạt động với những điểm ảnh ở viền bức ảnh và hàm np.lib.stride_tricks.as_strided() có thể duyệt đúng phần tử trong bộ nhớ mà không báo lỗi, chúng ta cần phải thêm phần đệm cho tấm ảnh ban đầu. Phần đệm này sẽ

được tính bằng cách lấy phần nguyên của kernel size chia cho 2, ví dụ nếu kernel size là 3 thì ảnh mới sẽ có đệm thêm 1 kích cỡ. Thao tác này được mô tả như ảnh dưới đây:

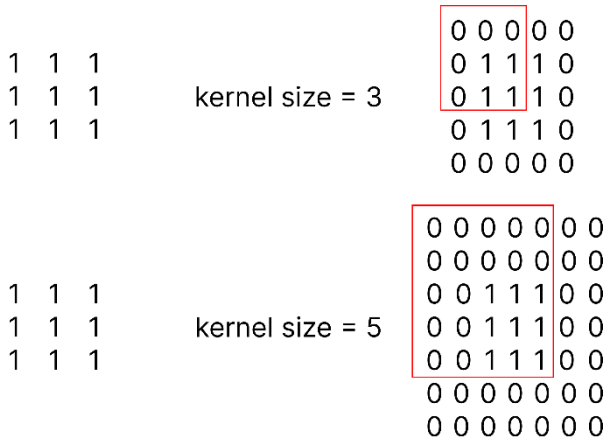


Figure 16: Minh họa việc thêm padding ảnh

Thao tác này sẽ được thực hiện bởi hàm numpy.pad() như sau: padded_image = np.pad(image, ((pad_height, pad_height), (pad_width, pad_width), (0, 0)), mode='constant'). Trong đó truyền vào (pad_height, pad_height) ở tham số đầu tức là thêm pad_height dòng đệm vào phía trên và phía dưới của ma trận image; (pad_width, pad_width) ở tham số thứ hai là thêm pad_width cột đệm vào bên trái và bên phải của ma trận image; (0, 0) ở tham số thứ ba nghĩa là không thêm phần tử đệm vào trục thứ ba của ma trận image (chiều dành cho các kênh màu) và cuối cùng mode='constant' có nghĩa là các phần tử đệm được thêm vào sẽ có giá trị không 0 như ảnh trên.

Tiếp theo ta cần xác định strides, ở đây em đã dùng lệnh strides = padded_image.strides[:2] + padded_image.strides[:2] + padded_image.strides[2:]. Giải thích kỹ câu lệnh này hơn thì nó sẽ giúp tạo một tuple mới chứa thông tin về cách di chuyển trong bộ nhớ để tạo ra các cửa sổ trượt padded_image.strides[:2] lấy bước nhảy cho chiều cao và chiều rộng, lặp lại hai lần để tạo bước nhảy cho kernel. Cuối cùng là thêm padded_image.strides[2:] để giữ nguyên bước nhảy cho kênh màu.

Câu lệnh này thực hiện phép tích tensordot (dot product) giữa sliding_windows và kernel dọc theo các trục xác định. Tham số axes=((2, 3), (0, 1)) chỉ định rằng trục thứ 2 và 3 của sliding_windows sẽ được nhân với trục thứ 0 và 1 của kernel. Kết quả là một mảng chứa kết quả của phép tích chập giữa padded_image và kernel.

Cuối cùng sau khi đã có sliding_windows và kernel, em sẽ dùng hàm tensordot để thực hiện phép tính chập giữa chúng bằng cách nhân và cộng các phần tử tương ứng. Hàm tensordot là một hàm thực hiện phép tích vô hướng (dot product) giữa các tensor (mảng đa chiều), cho phép chúng ta thực hiện các phép nhân ma trận hoặc phép nhân tensor trên các trục được chỉ định một cách nhanh chóng thay vì chúng ta phải duyệt qua 3 vòng for làm chương trình bị chậm. Em sẽ truyền vào sliding_windows và kernel kèm than số axes=((2, 3), (0, 1)) – tức là các trục thứ 2 và 3 của sliding_windows (tương ứng với kích thước kernel) sẽ nhân với các trục thứ 0 và 1 của kernel.

Cuối cùng sau khi đã chọn kernel cho chức năng làm mờ hoặc làm sắc nét tương ứng, em sẽ thực hiện nhân kernel với ma trận và cuối cùng là dùng hàm clip() và astype() cho ma trận thu được.

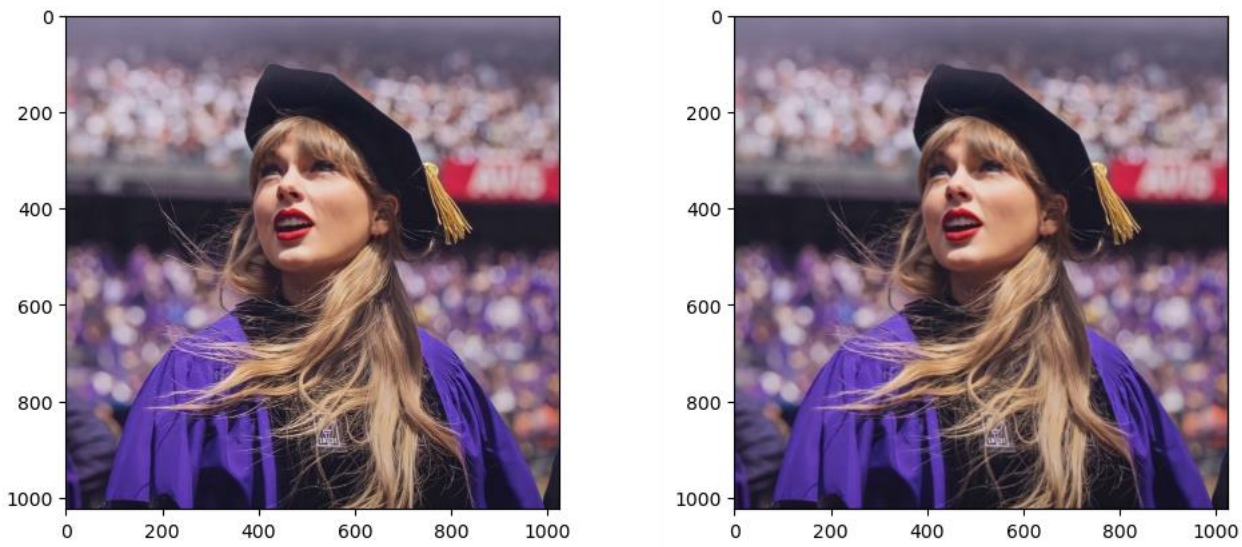


Figure 17: Ảnh gốc và Ảnh sau khi được làm mờ với kernel Box Blur [15]

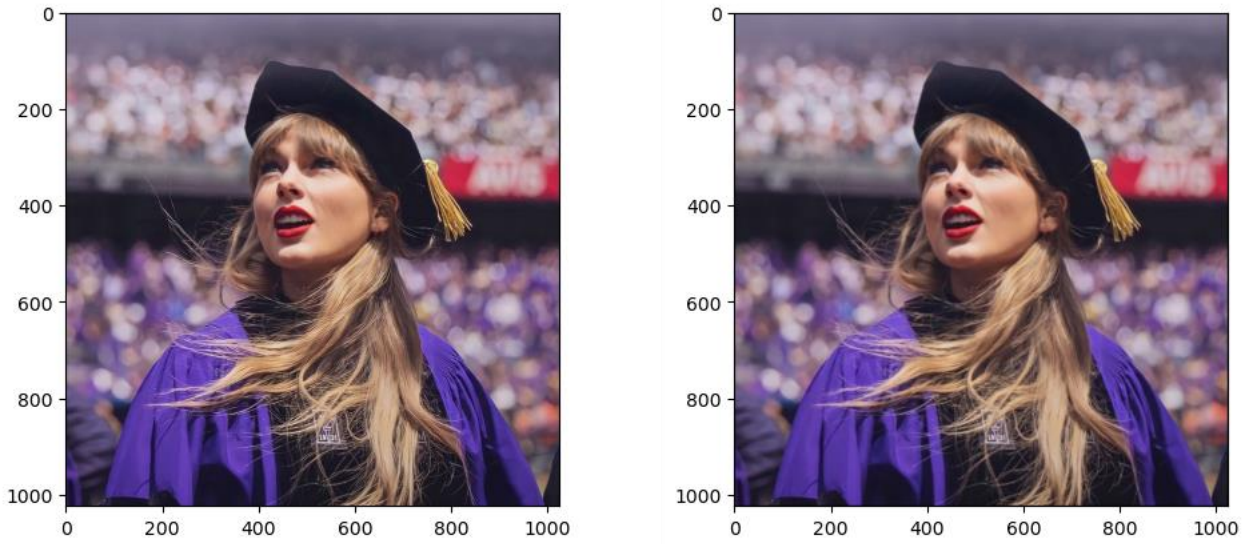


Figure 18: Ảnh gốc và Ảnh sau khi được làm mờ với kernel Gaussian blur 5 × 5 [15]

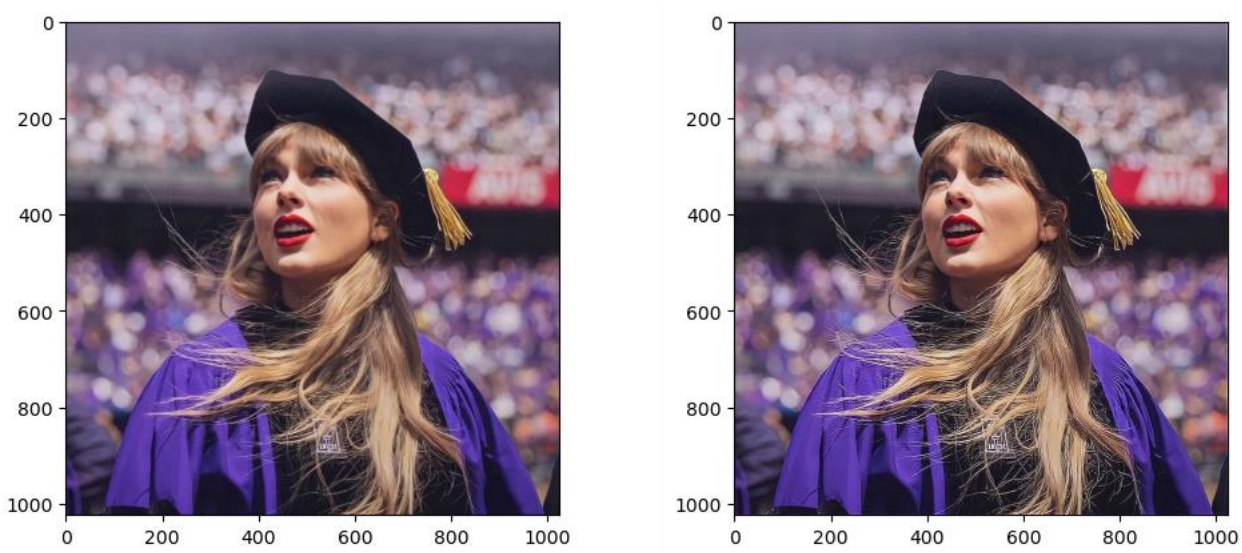


Figure 19: Ảnh gốc và Ảnh sau khi được làm sắc nét [15]

6. Cắt phần ảnh ở trung tâm theo kích thước

Vì chức năng yêu cầu sẽ cắt ảnh một ảnh theo kích thước từ phần trung tâm của ảnh, nên để cài đặt trước tiên chúng ta sẽ cho phép nhập vào kích thước chiều dài và chiều rộng của ảnh mong muốn, sau đó tính toán xem với kích thước như vậy chúng ta sẽ lấy ra các điểm ảnh từ dòng nào, kết thúc ở dòng nào và từ cột nào, kết thúc ở cột nào. Sau khi lấy kích thước ảnh ban đầu, em sẽ tính toán dòng bắt đầu lấy và dòng cuối cùng lấy của tấm ảnh bằng cách lấy chiều dài ảnh gốc chia đôi, trừ đi chiều dài ảnh mới chia đôi là ta sẽ có được dòng bắt đầu, nếu lấy chiều dài ảnh gốc chia đôi và cộng chiều dài ảnh mới chia đôi ta sẽ có được dòng cuối cùng cần lấy. Làm thao tác tương tự cho chiều rộng và sử dụng slicing trong numpy để lấy ra ảnh mong muốn. Với cách tiếp cận này chúng ta có thể crop 1 ảnh gốc là hình vuông hoặc hình chữ nhật tùy ý.

Vì ảnh lấy ra không thể có số dòng hoặc số cột là âm nên nếu nhập vào kích thước có chiều dài (hoặc chiều rộng) lớn hơn tấm chiều dài (hoặc chiều rộng) tấm ảnh gốc thì chương trình sẽ báo lỗi.

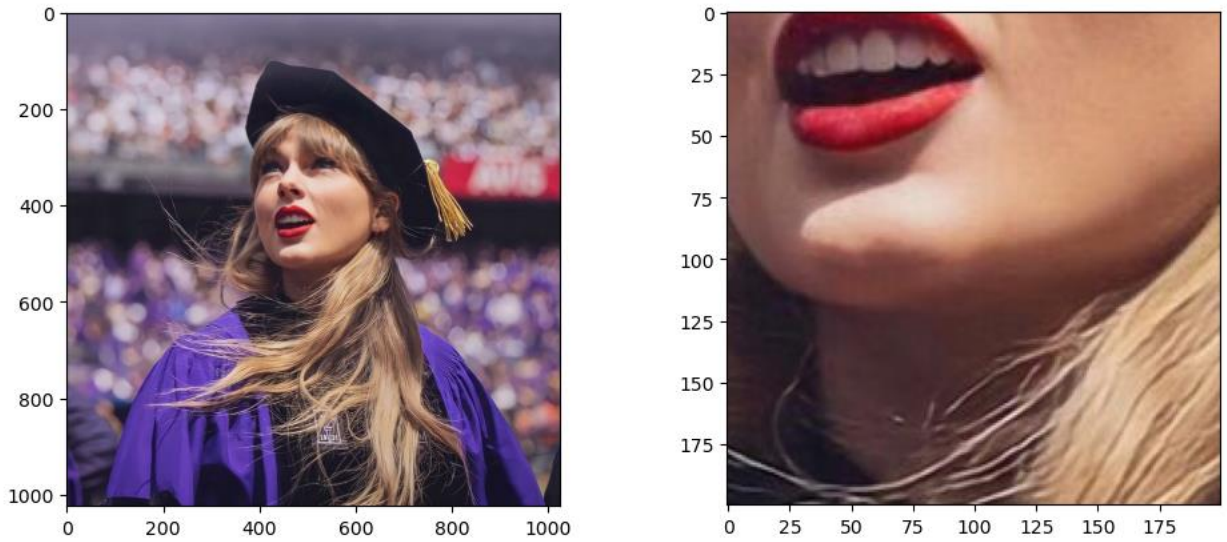


Figure 20: Ảnh gốc và Ảnh sau được cắt từ trung tâm Ảnh gốc với kích thước 200 x 200

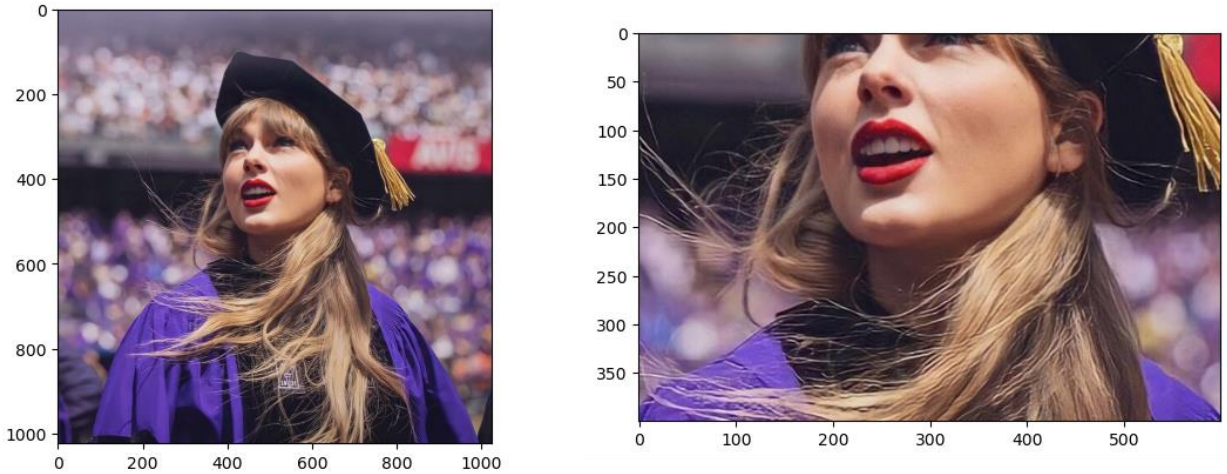


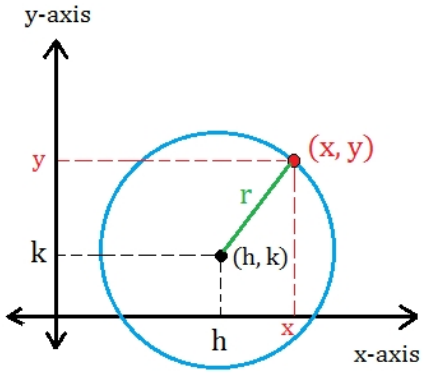
Figure 21: Ảnh gốc và Ảnh sau được cắt từ trung tâm Ảnh gốc với kích thước 400 x 600

7. Cắt ảnh theo khung (Hình tròn và Hình elip)

Để thực hiện chức năng apply một khung hình tròn vào tấm ảnh, với những giá trị điểm màu bên ngoài khung sẽ được chuyển thành màu đen rgb(0,0,0), ý tưởng ban đầu của em là sử dụng một mảng (gọi là mask) có kích thước số dòng và số cột tương tự với ảnh gốc nhưng mảng này sẽ chứa giá trị Boolean chỉ gồm True hoặc False. Một vị trí được đánh dấu là True nếu sau khi tính toán, ta biết được vị trí đó sẽ nằm ở bên trong của khung hình (được hiển thị) mà chúng ta muốn apply, và ngược lại nó sẽ là False nếu vị trí đó nằm bên ngoài khung (không được hiển thị). Và cuối cùng chúng ta sẽ sử dụng câu lệnh img[~mask] = 0 để chuyển những điểm màu từ ảnh gốc có vị trí được đánh dấu là True ở trong mask thành giá trị 0 (Thao tác này sẽ giúp gán 0 cho cả 3 kênh màu). Thao tác này có nghĩa là chúng ta sẽ sử dụng bitwise để đảo giá trị False thành True, True thành False, và nếu điểm ảnh nào trong img ban đầu được đánh dấu là False ở trong ~mask thì giá trị của các kênh màu nó sẽ chuyển về thành (0,0,0). Điều này sẽ áp dụng được cho tất cả các khung hình mà chúng ta muốn cắt ảnh theo đó, vậy khó khăn bây giờ chính là chúng ta cần tính toán được những điểm ảnh nào nằm bên trong khung và những điểm ảnh nào nằm bên ngoài khung.

a. Khung tròn

Ở kiến thức phổ thông, em đã biết được rằng trong hệ trục tọa độ x, y thì phương trình hình tròn sẽ có dạng như sau:



$$(x - h)^2 + (y - k)^2 = r^2$$

Figure 22: Ảnh minh họa phương trình đường tròn

Để kiểm tra một điểm có thuộc đường tròn có tâm (h, k), bán kính r hay không chúng ta sẽ thế tọa độ x, y của nó vào phương trình $(x - h)^2 + (y - h)^2$, nếu kết quả trả ra là lớn hơn r^2 thì điểm đó sẽ nằm ngoài hình tròn, và ngược lại nếu nó nhỏ hơn hoặc bằng r^2 thì nó sẽ nằm bên trong hình tròn.

Em sẽ áp dụng lý thuyết nêu trên vào phần cài đặt của mình, trước tiên em sẽ cần biết tâm và bán kính của đường tròn là bao nhiêu. Vì đây là hình tròn tiếp tuyến với 4 cạnh hình vuông nên chúng ta có thể dễ dàng chỉ ra được tâm của hình tròn cũng chính là tâm của hình vuông – lấy chiều dài và chiều rộng lần lượt chia 2 lấy thương (trường hợp chiều dài và chiều rộng là số lẻ thì ta sẽ lấy phần nguyên của kết quả) để tìm ra tâm. Bán kính của hình tròn cũng chính là bằng khoảng cách từ rìa hình vuông đến tâm, tức là bằng chiều dài chia đôi hoặc chiều rộng chia đôi.

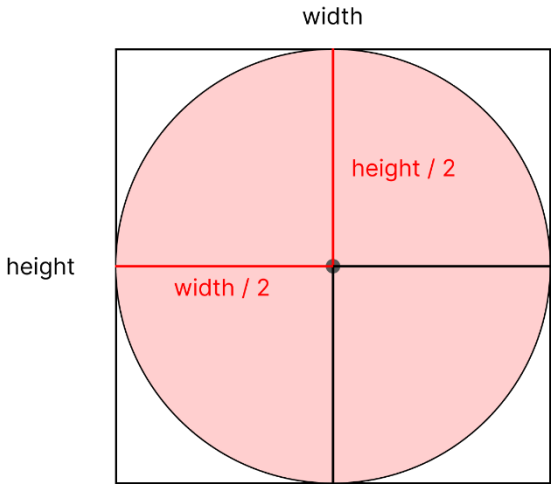


Figure 23: Ảnh minh họa về việc tính bán kính và tâm đường tròn

Sau đó em sẽ dùng hàm numpy.ogrid[] để lấy ra số index của các vị trí điểm ảnh trong ảnh, hay chúng ta có thể hiểu nó chính là tọa độ của điểm ảnh đó. Câu lệnh X, Y =

np.ogrid[:width, :height] sẽ lấy ra một mảng X chứa tọa độ trên trục hoành của điểm ảnh, và một mảng Y chứa tọa độ trên trục tung của điểm ảnh. Giải thích rõ hơn thì hàm ogrid[:width, :height] sẽ trả về hai mảng, một mảng gồm các số từ 0 tới width (hoành độ), và một mảng gồm các số từ 0 tới height (tung độ). Và sau khi đã có hoành độ và tung độ của các điểm ảnh, chúng ta sẽ chỉ việc thế vào phương trình trên để tạo ra một mảng chứa toàn khoảng cách từ điểm có hoành độ tung độ tương ứng tới tâm. So sánh khoảng cách này với bán kính của hình tròn, nếu khoảng cách của điểm ảnh đó nhỏ hơn hoặc bằng bán kính, vị trí của điểm ảnh sẽ đánh dấu True – tức là giữ lại màu sắc, và ngược lại chuyển điểm ảnh đó thành màu đen. Thao tác trên sẽ được thực hiện nhanh chóng qua câu lệnh mask = dist_from_center <= radius, tạo ra 1 mảng boolean mask với các giá trị true false tương ứng.

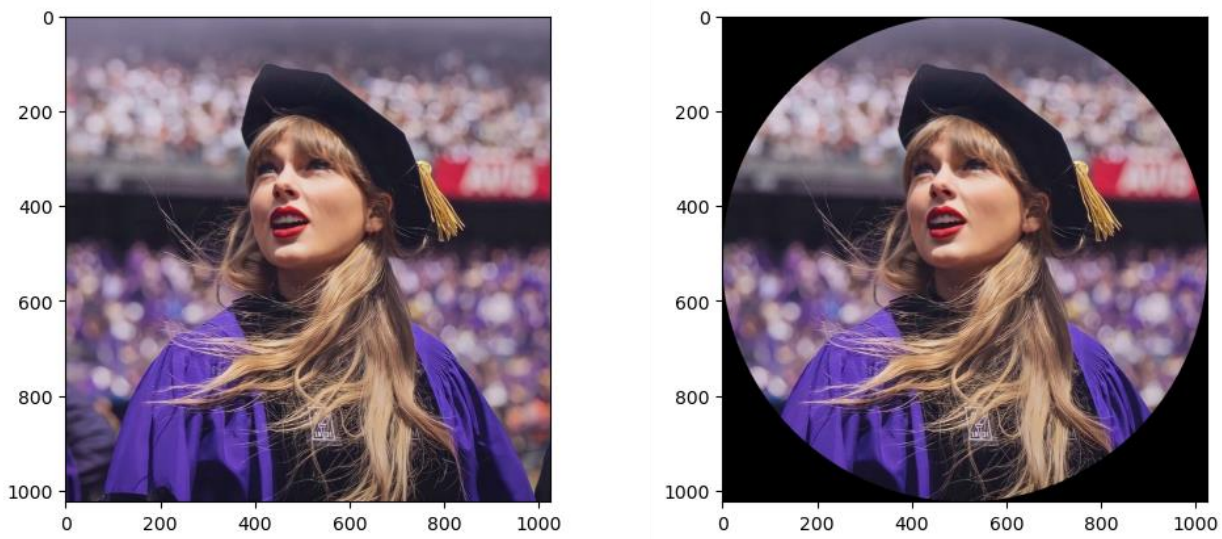


Figure 24: Ảnh gốc và Ảnh được cắt theo khung hình tròn

b. Khung elip

Để thực hiện được chức năng cắt ảnh theo khung elip, em cũng sẽ dùng ý tưởng trên để có thể apply khung hình mong muốn vô tấm ảnh. Vấn đề được đặt ra tiếp theo đó chính là chúng ta phải viết được phương trình của hình elip này. Quan sát hình trong yêu cầu chúng ta sẽ dễ nhận thấy hình khung ảnh này được tạo bởi 2 hình elip chéo nhau, mỗi hình đều quay phải hoặc quay trái 1 góc 45 độ và mỗi hình đều có tiếp tuyến với 4 cạnh của hình vuông. Quay lại kiến thức phổ thông, chúng ta đều biết là một hình elip sẽ gồm các yếu tố là tâm, độ dài trục lớn, độ dài trục nhỏ và 2 tiêu cự đối xứng nhau qua tâm và 3 điểm này nằm trên cùng 1 đường thẳng. Vậy để xây dựng được hình elip, trước tiên chúng ta cần biết tâm và độ dài của 2 trục lớn và nhỏ của hình.

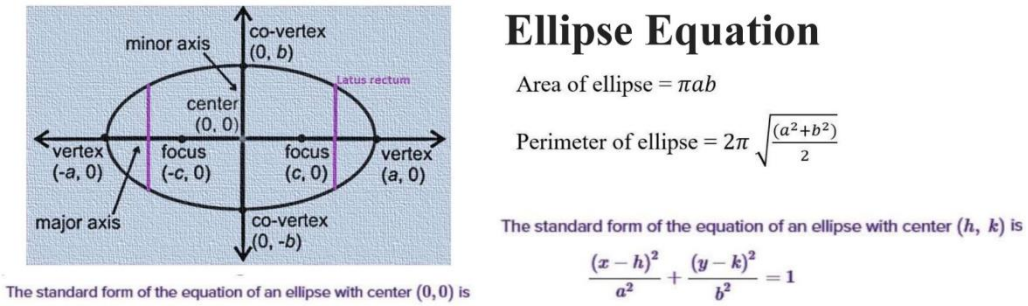


Figure 25: Ảnh minh họa phương trình elip

Để xác định được độ dài trục lớn và trục nhỏ cho hình elip có tính chất nếu trên, em đã thử tìm hiểu, tham khảo từ bạn Nguyễn Anh Kiệt (MSSV: 22127220) và biết được 1 thuật ngữ gọi là Director circle [9]. Director circle của một hình elip là một đường tròn chứa tất cả các điểm mà tại điểm đó, sẽ có hai tiếp tuyến của hình elip đồng tâm vuông góc với nhau. Một điều đặc biệt khác là nếu ta gọi độ dài trục lớn của elip là a, độ dài trục nhỏ là b thì bán kính của đường tròn trên cũng chính là $\sqrt{a^2 + b^2}$. Điều đó sẽ có áp dụng mạnh mẽ tới việc thực hiện yêu cầu này vì chúng ta cũng đang tìm kiếm 1 hình elip tiếp tuyến với 4 cạnh của hình vuông, cũng là hình có các cặp cạnh liền kề vuông góc với nhau.

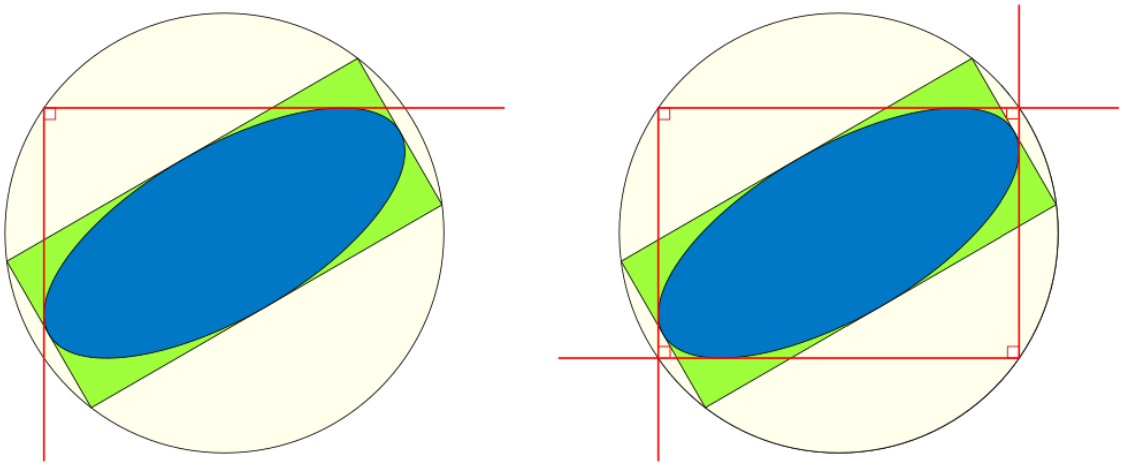


Figure 26: Ví dụ 1 về Director circle

Xem hình ảnh trong ví dụ trên, chúng ta sẽ dần hình dung ra được sự liên quan của Director circle đến đề bài, ngoài ra ta dễ dàng thay đổi vị trí của 4 điểm khác đi một chút sao cho nó tạo thành 1 hình vuông, chúng ta sẽ thu được một hình elip giống như đề.

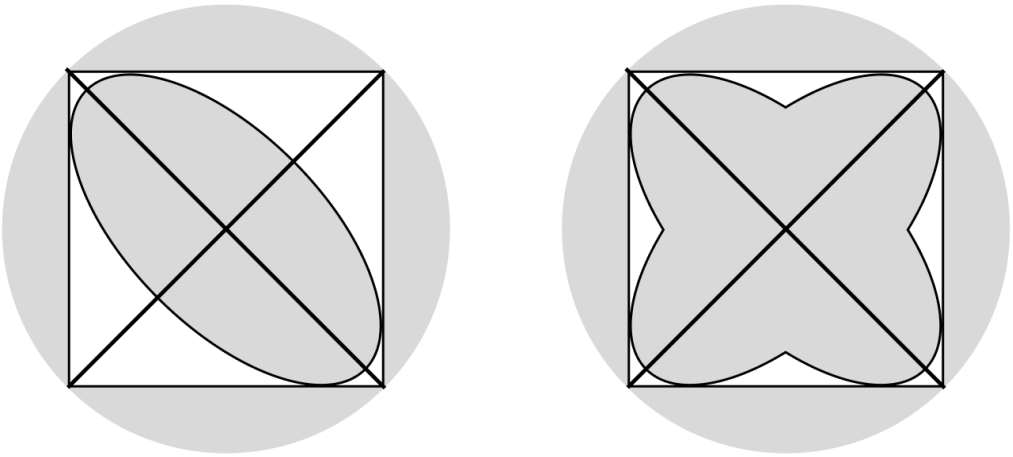


Figure 27: Ví dụ 2 về Director circle

Quan sát hình ảnh trên ta có được một mối liên hệ giữa chiều dài của ảnh (gọi tắt là h) và bán kính của hình tròn bên ngoài là $h = \sqrt{(a^2 + b^2) + (a^2 + b^2)} = \sqrt{2a^2 + 2b^2}$ (Và a, b cũng lần lượt là trục lớn và trục nhỏ của hình elip bên trong). Sẽ có vô số cặp a, b thỏa mãn được nhưng trong quá trình làm chức năng này em tìm được một cặp a, b cho ra hình ảnh thu được sát với yêu cầu đề bài là $a = 1{,}75b$.

Thế ngược lại vào biểu thức trên ta sẽ tính được chiều dài của ảnh $h = \sqrt{6.125b^2 + 2b^2} = \sqrt{8.125b^2} = \sqrt{8.125}b$. Suy ra độ dài trục nhỏ $b = \frac{h}{\sqrt{8.125}}$ và độ dài trục lớn $a = \frac{1.75h}{\sqrt{8.125}}$

Vậy là chúng ta đã tính được độ dài trục lớn, trục nhỏ của hình elip dựa trên chiều dài ảnh cho trước, tiếp theo chúng ta cần viết được phương trình elip. Sau khi tìm hiểu về cách quay một phương trình elip theo 1 góc cho trước ở [10], em tìm thấy được phương trình sau :

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

Trong đó x, y lần lượt là hoành độ và tung độ của điểm, (h, k) là tâm của elip và cũng là tâm cũng ảnh, a và b lần lượt là các độ dài trục lớn và trục nhỏ của elip và A là góc mà ta muốn phương trình elip tương ứng quay theo. Phương trình này là một phiên bản khác của phương trình elip và giống hầu hết với phương trình elip thông thường, và thật ra nếu ta thế góc quay A = 0, nó sẽ ra 1 phương trình elip chính tắc mà chúng ta thường hay gặp vì sin(0) = 0 và cos(0) = 1. Vậy để quay hình elip theo chiều kim đồng hồ hoặc ngược kim đồng hồ 1 góc 45 độ, ta có thể thế A = $\frac{45}{180}\pi = \frac{\pi}{4}$ và A = $\frac{-\pi}{4}$ lần lượt vào phương trình, ta sẽ thu được 2 phương trình hình elip mới có góc quay tương ứng :

$$\frac{((x-h)\cos\left(-\frac{\pi}{4}\right) + (y-k)\sin\left(-\frac{\pi}{4}\right))^2}{a^2} + \frac{((x-h)\sin\left(-\frac{\pi}{4}\right) + (y-k)\cos\left(-\frac{\pi}{4}\right))^2}{b^2} \leq 1$$

$$\frac{((x-h)\cos\left(\frac{\pi}{4}\right) + (y-k)\sin\left(\frac{\pi}{4}\right))^2}{a^2} + \frac{((x-h)\sin\left(\frac{\pi}{4}\right) + (y-k)\cos\left(\frac{\pi}{4}\right))^2}{b^2} \leq 1$$

Phần cài đặt của chức năng này sẽ được thực hiện tương tự như phần của chức năng cắt khung theo hình tròn bên trên. Trước tiên em sẽ tính toán độ dài trục lớn và trục nhỏ của hình elip dựa trên kích thước ảnh cho trước từ biểu thức được suy ra ở trên, sau đó lấy tung độ và hoành độ của điểm bằng numpy.ogrid[]. Tạo ra 2 ma trận Boolean mask cho phương trình elip viết được bên trên, ma trận mask này được tạo ra bằng cách thế lần lượt các tọa độ (x, y) của các điểm ảnh vào phương trình, nếu trả về kết quả ≤ 1 thì điểm ảnh đó thuộc hình elip, đánh dấu là True và ngược lại sẽ được đánh dấu là False. Ngoài ra vì ta cần chồng 2 elip này lên nhau nên ta sẽ dùng toán tử OR | , chỉ cần vị trí đó là True ở 1 trong 2 mask được tạo ra thì chúng ta sẽ giữ lại màu của điểm ảnh, nếu vị trí đó ở cả 2 mask đều là False ta sẽ chuyển nó thành màu đen (0,0,0)

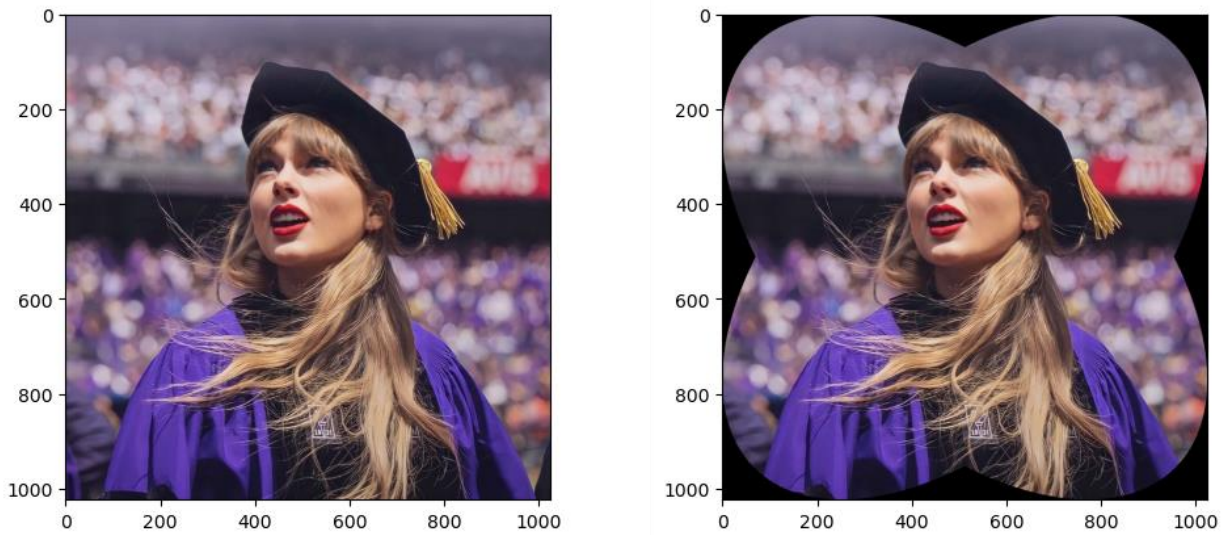


Figure 28: Ảnh gốc và Ảnh sau khi được cắt theo khung hai hình elip chéo nhau

8. Phóng to và Thu nhỏ ảnh

a. Phóng to ảnh 2x

Ý tưởng của việc thực hiện chức năng phóng to là chúng ta sẽ lặp lại các điểm ảnh từ ảnh ban đầu theo cả 2 chiều ngang và dọc, và lưu ý rằng các điểm ảnh mới được lặp lại phải nằm ngay cạnh bên, liền kề với điểm ảnh cũ để chúng ta có thể thu được tấm ảnh mới có kích thước lớn hơn những vẫn giữ đúng nội dụng của tấm ảnh ban đầu. Để thực hiện được chức năng Phóng to ảnh lên 2 lần, em sẽ sử dụng hàm repeat() của numpy. Hàm này sẽ cho phép người dùng lặp lại phần tử trong mảng theo số lần mong muốn, ngoài ra người dùng có thể chọn chiều muốn lặp lại bằng cách truyền vào tham số axis, nếu axis = 0 tức là lặp lại theo dòng, axis = 1 tức là lặp lại theo cột. Ví dụ nếu ta có một mảng ban đầu là [[2, 4, 6], [3, 5, 7]], nếu ta cho lặp lại 2 lần theo lần lượt 2 chiều dọc và ngang ta sẽ thu được ma trận mới là [[2, 4, 6], [2, 4, 6], [3, 5, 7], [3, 5, 7]] và [[2 2 4 4 6 6], [3 3 5 5 7 7]]. Có thể thấy rằng các phần tử mới được lặp lại và phần tử ban đầu đều nằm sát bên cạnh nhau, điều này khiến cho hàm numpy.repeat() có thể được sử dụng trong trường hợp này. Em sử dụng câu lệnh sau để thực hiện chức năng:

```
new_img_2d = np.repeat(np.repeat(img_2d, zoom, axis=0), zoom, axis=1)
```

Trong câu lệnh trên em đã cho tấm ảnh được lặp lại "zoom" lần theo cả 2 chiều ngang và dọc vì ta cần tăng kích thước của ảnh theo cả 2 chiều để giữ được nội dung ban đầu của tấm ảnh nhưng vẫn có được 1 tấm ảnh mới lớn hơn. Ngoài ra việc truyền vào hệ số zoom có thể cho phép người dùng tăng kích thước ảnh lên "zoom" lần tùy thích, nhưng em sẽ chỉ để mặc định zoom là 2 lần trong đồ án này.

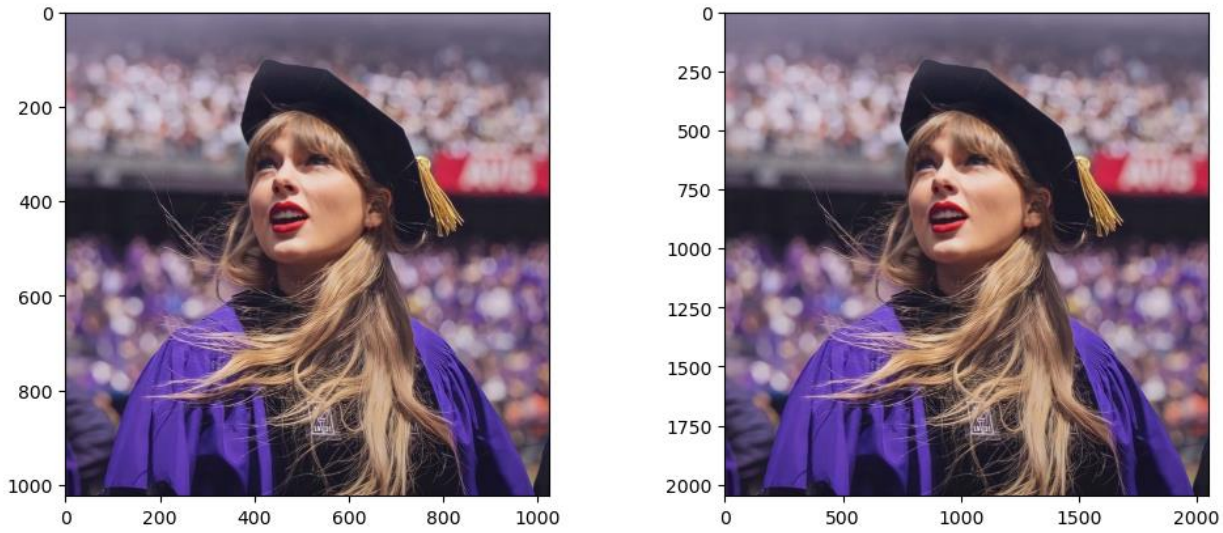


Figure 29: Ảnh gốc và Ảnh sau khi được phóng to 2 lần

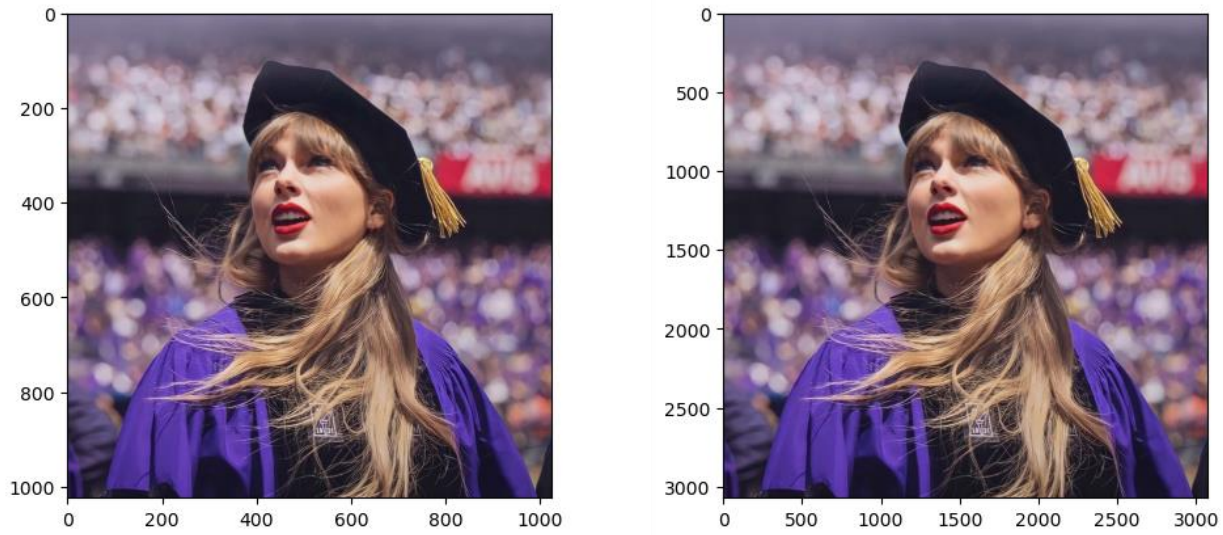


Figure 30: Ảnh gốc và Ảnh sau khi được phóng to 3 lần

b. Thu nhỏ ảnh 2x

Ý tưởng của việc thực hiện chức năng này là em sẽ gom nhóm các điểm ảnh lại với nhau, và tính giá trị của điểm ảnh mới bằng cách lấy giá trị trung bình của các điểm ảnh trong nhóm đó. Có 1 hàm trong thư viện numpy giúp chúng ta "gom nhóm" các điểm ảnh lại đó chính là hàm reshape.

Giả sử chúng ta có 1 ảnh với kích thước ban đầu là HeightxWidthx3 và chúng ta đang mong muốn ảnh đầu ra có kích thước là NewHeightxNewWidthx3, em sẽ tính toán xem số lượng điểm ảnh cần gom nhóm lại cho điểm ảnh theo từng dòng và cột là bao nhiêu bằng cách lấy

Height chia cho NewHeight ra số điểm ảnh cần gom nhóm theo dòng (gọi tạm là BlockH) và lấy Width chia cho NewWidth ra số điểm ảnh cần gom nhóm theo cột (BlockW) và sau đó dùng hàm reshape để chuyển nó thành ma trận mới có 5 chiều: img_2d.reshape((newheight, height // newheight, newwidth, width // newwidth, channels)), trong đó các chiều lần lượt là số dòng mới, số dòng sẽ được gom nhóm từ ảnh ban đầu để tạo thành ảnh mới, số cột mới, số cột sẽ được góm nhóm từ ảnh ban đầu để tạo thành ảnh mới và số lượng kênh màu của ảnh. Sau đó em sẽ dùng hàm mean để tính giá trị trung bình cho chiều thứ 4 của ma trận mới được reshape, tức là em đang tức giá trị trung bình của các cột điểm ảnh bị gom nhóm và sau đó gán nó lại cho các cột mới của tấm ảnh. Tương tự em tiếp tục dùng hàm mean để tính giá trị cho chiều thứ 2 của ma trận và trả về kết quả cuối cùng là ma trận mới có 3 chiều – cũng chính là tấm ảnh đã được thu nhỏ 2x.

Ở phần thu nhỏ em cũng sẽ chỉ cho truyền vào zoom_factor là 0.5 để tránh trường hợp người dùng truyền vào những zoom_factor mà sau khi tính toán lại chiều dài, chiều rộng mới thì hàm reshape() không thể chuyển từ ma trận ban đầu thành ma trận mới được. Ngoài ra sẽ có một số trường hợp ta truyền ảnh có chiều dài và và chiều rộng vào là 1 số lẻ, làm cho hàm reshape không thể hoạt động. Ví dụ ta truyền vào ảnh 901 x 900, thì nếu thực hiện thao tác reshape thành (900, 2, 900, 2, 3), chương trình sẽ báo lỗi vì 901 x 900 không bằng với 450 x 2 x 450 x 2. Vậy nên nếu gặp phải trường hợp trên em sẽ bỏ hẳn hàng hoặc cột bị lẻ đó bằng thao tác slicing img_2d = img_2d[:-1, :, :] hoặc img_2d = img_2d[:, :-1, :]. Do có tính giá trị trung bình, em cũng có sử dụng hàm clip và astype để ảnh được hiển thị đúng,

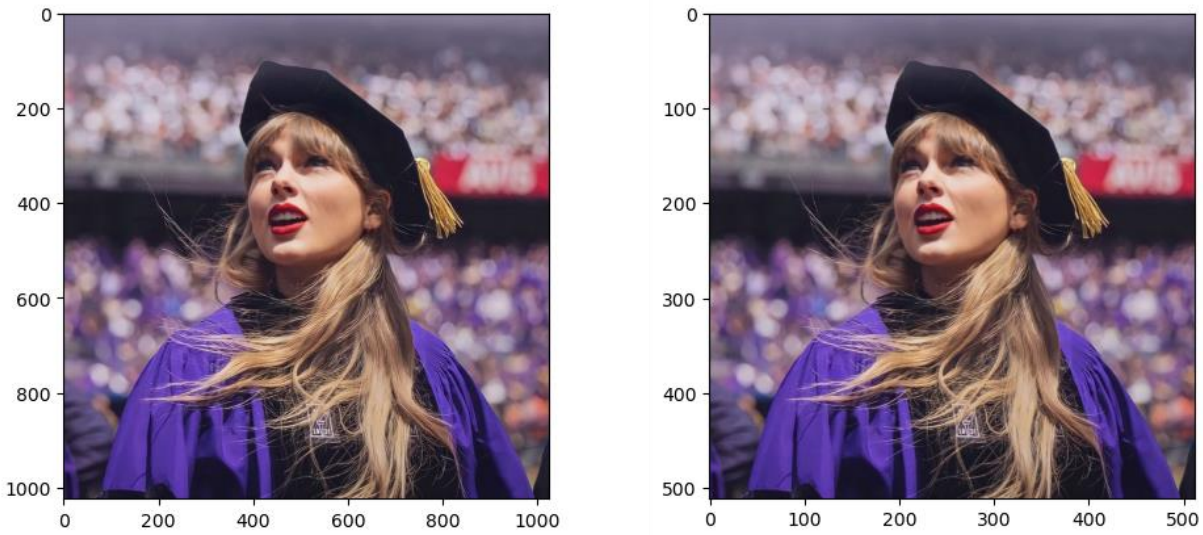


Figure 31: Ảnh gốc và Ảnh sau được thu nhỏ 2 lần

II. Mô Tả Các Hàm Được Cài Đặt

1. Các hàm đọc và lưu ảnh

`def read_img(img_path):` Hàm này sẽ đọc vào tên của ảnh "img_path" (Ví dụ: "taylor.jpg"), đọc ảnh từ đường dẫn và trả về ảnh dưới dạng 1 ndarray.

`def show_img(img):` Hàm này sẽ nhận vào một ndarray và hiển thị nó bằng lệnh plt.imshow(img)

`def save_img (img, img_file_name):` Hàm này sẽ nhận vào một ndarray và tên image muốn lưu cho tấm ảnh và dùng lệnh Image.fromarray(img) và cuối cùng là dùng hàm save() từ Image.
`def fix_image_path(original_name, auxiliary):` Hàm này sẽ nhận vào tên file ban đầu và thêm vào tên phụ vào tên gốc và trả về file_path hoàn chỉnh.

2. Các hàm xử lý ảnh

`def adjust_brightness(img_2d, beta):` Nhận vào 1 ndarray, 1 số float beta và điều chỉnh độ sáng của hình ảnh bằng cách thêm một giá trị cố định vào tất cả các pixel rồi trả về 1 mảng ndarray đã được tăng độ sáng.
`def adjust_contrast(img_2d, alpha):` Nhận vào 1 ndarray, 1 số float alpha và điều chỉnh độ tương phản của hình ảnh bằng cách nhân tất cả các pixel với một giá trị cố định rồi trả về 1 mảng ndarray đã được giảm độ tương phản.
`def flip(img_2d, direction):` Nhận vào 1 ndarray, 1 string direction và lật hình ảnh theo chiều ngang hoặc chiều dọc dựa trên direction rồi trả về 1 mảng ndarray đã được lật.
`def to_grayscale(img_2d):` Nhận vào 1 ndarray và chuyển đổi hình ảnh sang ảnh grayscale bằng cách sử dụng phương pháp luminosity tính toán lại màu của các kênh RGB rồi trả về 1 mảng ndarray đã được chỉnh sang màu xám.
`def to_sepia(img_2d):` Nhận vào 1 ndarray và chuyển đổi hình ảnh sang ảnh sepia bằng cách tính toán lại màu của các kênh RGB theo bộ lọc sepia rồi trả về 1 mảng ndarray đã được chỉnh sang màu sepia.
`def convolve2d(image, kernel):` Nhận vào 1 mảng ndarray của hình ảnh và 1 mảng ndarray của kernel và dụng tích chập 2D cho hình ảnh sử dụng một kernel cố định và trả về ndarray mới.
`def blur_img(img_2d):` Nhận vào 1 mảng ndarray img_2d, khai báo kernel tương ứng với chức năng làm mờ, gọi hàm convolve2d() và trả về 1 ndarray đã xử lý.
`def sharpen_img(img_2d):` Nhận vào 1 mảng ndarray img_2d, khai báo kernel tương ứng với chức năng làm sắc nét, gọi hàm convolve2d() và trả về 1 ndarray đã xử lý.
`def crop_img(img_2d, size):` Nhận vào 1 mảng ndarray ban đầu và 1 tuple chứa kích thước chiều dài và chiều rộng ảnh muốn cắt, cắt ảnh phần trung tâm theo kích thước trả về 1 ndaraay đã được xử lý.
`def circular_mask(img_2d):` Nhận vào 1 mảng ndarray ban đầu, cắt ảnh theo khung hình tròn và trả về 1 ndaraay đã được xử lý.
`def ellipse_mask(img_2d):` Nhận vào 1 mảng ndarray ban đầu, cắt ảnh theo khung hình elip và trả về 1 ndaraay đã được xử lý.
`def zoom_image(img_2d, zoom):` Nhận vào 1 mảng ndarray ban đầu, 1 số float zoom_factor, tăng hoặc giảm kích thước ảnh tùy vào zoom_factor và trả về 1 ndarray đã được xử lý.

3. Các hàm xử lý ảnh

`def main():` Hàm này khi khởi chạy sẽ cho phép người dùng nhập vào tên ảnh và miền mở rộng (Ví dụ: “Taylor.jpg”) và chọn những thao tác mong muốn để xử lý tấm ảnh gồm:

0. Execute all modes: Thực hiện toàn bộ các thao tác xử lý ảnh, show ảnh và save ảnh về máy, hàm sẽ hỏi thêm các thông số muốn truyền vào cho các thao tác Chỉnh độ sáng, Chỉnh độ tương phản và kích thước của ảnh muốn crop.
1. Adjust brightness (Điều chỉnh độ sáng): Truyền vào 1 số âm hoặc dương để giảm hoặc tăng độ sáng tương ứng.

2. Adjust contrast (Điều chỉnh độ tương phản): Truyền vào 1 số từ 0 tới 1 hoặc từ 1 trở lên để giảm hoặc tăng độ tương phản.
3. Flip image (vertical or horizontal) (Lật ảnh dọc hoặc ngang): Cho phép người dùng chọn chế độ lật ảnh, lật theo chiều dọc hoặc chiều ngang.
4. Convert image (grayscale or sepia): Chuyển ảnh từ màu bình thường sang ảnh xám hoặc ảnh sepia.
5. Kernel image-processing (blur or sharpen): Sử dụng kernel để xử lý ảnh, cho phép người dùng chọn giữa làm mờ hoặc làm sắc nét ảnh
6. Crop image (Cắt ảnh trung tâm): Cho phép người dùng nhập kích thước ảnh mới muốn cắt từ trung tâm của ảnh.
7. Mask image (circular or ellipse) (Cắt ảnh theo khung): Cho phép người dùng lựa chọn giữa việc cắt ảnh theo khung hình tròn hoặc hình elip.
8. Zoom image (zoom in or zoom out) (Thay đổi kích thước ảnh): Cho phép người dùng thay đổi kích thước ảnh thành 2x hoặc 0.5x.

III. Kết Quả Và Đánh Giá Mức Độ Hoàn Thành Công Việc

STT	Chức năng/Hàm	Mức độ hoàn thành	Ảnh kết quả
1	Thay đổi độ sáng	100%	Figure 4, Figure 5
2	Thay đổi độ tương phản	100%	Figure 7, Figure 8
3.1	Lật ảnh ngang	100%	Figure 9
3.2	Lật ảnh dọc	100%	Figure 10
4.1	RGB thành ảnh xám	100%	Figure 13
4.2	RGB thành ảnh sepia	100%	Figure 14
5.1	Làm mờ ảnh	100%	Figure 17, Figure 18
5.2	Làm sắc nét ảnh	100%	Figure 19
6	Cắt trung tâm ảnh theo kích thước	100%	Figure 20, Figure 21
7.1	Cắt ảnh theo khung tròn	100%	Figure 24
7.2	Cắt ảnh theo khung elip	100%	Figure 28
8	Hàm main	100%	
9	Phóng to/Thu nhỏ 2x	100%	Figure 29, Figure 30 Figure 31

TÀI LIỆU THAM KHẢO

[1] W3Schools, Colors RGB and RGBA, Ngày truy cập: 17/7/2024 https://www.w3schools.com/colors/colors_rgb.asp

[2] OpenCV, Basic Linear Transform, Ngày truy cập: 17/7/2024 https://docs.opencv.org/4.x/d3/dc1/tutorial_basic_linear_transform.html

[3] Code.org, Images, Pixels and RGB, Ngày truy cập: 17/7/2024 https://www.youtube.com/watch?v=15aqFQQVBWU

[4] Tutorialpoints, Grayscale to RGB Conversion, Ngày truy cập: 17/7/2024 https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

[5] Geeksforgeeks, Sepia Image, Image Processing in Java – Colored Image to Sepia Image Conversion, Ngày truy cập: 17/7/2024 https://www.geeksforgeeks.org/image-processing-in-java-colored-image-to-sepia-image-conversion/

[6] Stackoverflow, Sliding window of M-by-N shape numpy.ndarray, Ngày truy cập: 20/7/2024 https://stackoverflow.com/questions/15722324/sliding-window-of-m-by-n-shape-numpy-ndarray

[7] Numpy, flipud, Ngày truy cập 17/7/2024 https://numpy.org/doc/stable/reference/generated/numpy.flipud.html

[8] Numpy, flipud, Ngày truy cập 17/7/2024 https://numpy.org/doc/stable/reference/generated/numpy.fliplr.html

[9] Wikipedia, Director Circle, Ngày truy cập: 23/7/2024 https://en.wikipedia.org/wiki/Director_circle

[10] Geeksforgeeks, Sepia Image, Image Processing in Java – Colored Image to Sepia Image Conversion, Ngày truy cập: 17/7/2024 https://math.stackexchange.com/questions/426150/what-is-the-general-equation-of-the-ellipse-that-is-not-in-the-origin-and-rotate

[11] Mathematics, What is the general equation of the ellipse that is not in the origin and rotated by an angle?, Ngày truy cập: 23/7/2024 https://e2eml.school/convert_rgb_to_grayscale

[12] Baeldung, How to Convert an RGB Image to a Grayscale, Ngày truy cập: 17/7/2024 https://www.baeldung.com/cs/convert-rgb-to-grayscale

[13] Setosa, Image Kernels, Ngày truy cập: 20/7/2024 https://setosa.io/ev/image-kernels/

[14] Dr. Sarah Abraham University of Texas at Austin Computer Science Departmen, Image Manipulation: Filters and Convolutions, Ngày truy cập: 21/7/2024 https://www.cs.utexas.edu/~theshark/courses/cs324e/lectures/cs324e-6.pdf

[15] Wikipedia, Kernel (image processing), Ngày truy cập: 20/7/2024 https://en.wikipedia.org/wiki/Kernel_(image_processing)
[16] Figma to draw Illustration firgure 16, 23, 27