

**University of Science**  
**Viet Nam National University - Ho Chi Minh city**



**Báo cáo đồ án 2  
IMAGE PROCESSING**

**Giảng viên lý thuyết** Vũ Quốc Hoàng

**Giảng viên thực hành** Phan Thị Phương Uyên

Nguyễn Văn Quang Huy

**Sinh viên** Vũ Đình Chương      21127236

# Mục lục

<b>1 Mức độ hoàn thành</b>	<b>3</b>
<b>2 Tổng quan lý thuyết</b>	<b>3</b>
2.1 Vấn đề đặt ra . . . . .	3
2.2 Sơ lược về xử lý ảnh . . . . .	3
<b>3 Mô tả các chức năng</b>	<b>4</b>
3.1 Thay đổi độ sáng cho ảnh . . . . .	4
3.2 Thay đổi độ tương phản . . . . .	4
3.3 Lật ảnh (ngang - dọc) . . . . .	5
3.4 Chuyển đổi ảnh RGB thanh xám/sepia . . . . .	6
3.5 Làm mờ/ sắc nét ảnh . . . . .	7
3.6 Cắt ảnh theo kích thước (cắt ở trung tâm) . . . . .	12
3.7 Cắt ảnh theo khung tròn . . . . .	13
3.8 Cắt ảnh theo khung 2 ellipse chéo nhau . . . . .	14
<b>4 Các hàm bổ trợ khác</b>	<b>16</b>
4.1 importImage() . . . . .	16
4.2 show_img() . . . . .	16
4.3 export_img() . . . . .	16
4.4 int2float_img()/float2int_img() . . . . .	16
4.5 fftconvolution() . . . . .	16
4.6 Các hàm printImage cho từng chức năng . . . . .	17
4.7 choice() . . . . .	17
4.8 processing() . . . . .	17
4.9 main() . . . . .	17
<b>5 Kết quả</b>	<b>18</b>
5.1 Thay đổi độ sáng cho ảnh . . . . .	18
5.2 Thay đổi độ tương phản . . . . .	18
5.3 Lật ảnh (ngang - dọc) . . . . .	19
5.4 Chuyển đổi ảnh RGB thanh xám/sepia . . . . .	19
5.5 Làm mờ/ sắc nét ảnh . . . . .	20
5.6 Cắt ảnh theo kích thước (cắt ở trung tâm) . . . . .	21
5.7 Cắt ảnh theo khung tròn . . . . .	22
5.8 Cắt ảnh theo khung 2 ellipse chéo nhau . . . . .	22
5.9 Thời gian xử lý . . . . .	22

**6 Tài liệu tham khảo**

**23**

# 1 Mức độ hoàn thành

STT	Tên chức năng	Kết quả đầu ra	Phần trăm hoàn thành
1	Thay đổi độ sáng cho ảnh	Tăng sáng cho ảnh	100%
2	Thay đổi độ tương phản	Tăng tương phản của ảnh	100%
3	Lật ảnh ngang, dọc	Ảnh có thể lật ngang, dọc	100%
4	Chuyển đổi ảnh RGB thành xám, sepia	Ảnh thay đổi thành xám, sepia	100%
5	Làm mờ, sắc nét ảnh	Ảnh gốc đầu ra đã làm mờ, ảnh mờ đầu vào đã làm sắc nét	100%
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	Ảnh đầu ra đã được cắt theo kích thước vào trung tâm	100%
7	Cắt ảnh theo khung hình tròn	Ảnh đã được cắt hình tròn nội tiếp	100%
8	Cắt ảnh theo khung 2 ellipse chéo nhau	Ảnh đã được cắt ra 2 hình ellipse theo hai đường chéo	???

# 2 Tổng quan lý thuyết

## 2.1 Vấn đề đặt ra

Cài đặt chương trình xử lý ảnh với các chức năng sau:

1. Thay đổi độ sáng cho ảnh
2. Thay đổi độ tương phản
3. Lật ảnh (ngang-dọc)
4. Chuyển đổi ảnh RGB thành xám/sephia
5. Làm mờ/sắc nét ảnh
6. Cắt ảnh theo kích thước (cắt ở trung tâm)
7. Cắt ảnh theo khung hình tròn
8. Cắt ảnh theo khung 2 ellipse chéo nhau

## 2.2 Sơ lược về xử lý ảnh

Xử lý ảnh là quá trình chuyển đổi một hình ảnh sang dạng kỹ thuật số và thực hiện các thao tác nhất định để nhận được một số thông tin hữu ích từ hình ảnh đó. Hệ thống xử lý hình ảnh thường coi tất cả các hình ảnh là tín hiệu 2D khi áp dụng một số phương pháp xử lý tín hiệu đã xác định trước.

Các loại xử lý hình ảnh chính:

- Nhận diện – Phân biệt hoặc phát hiện các đối tượng trong hình ảnh
- Làm sắc nét và phục hồi – Tạo hình ảnh nâng cao từ hình ảnh gốc
- Nhận dạng mẫu – Đo các mẫu khác nhau xung quanh các đối tượng trong hình ảnh
- Truy xuất – Duyệt và tìm kiếm hình ảnh từ một cơ sở dữ liệu lớn gồm các hình ảnh kỹ thuật số tương tự như hình ảnh gốc

### 3 Mô tả các chức năng

#### 3.1 Thay đổi độ sáng cho ảnh

##### Ý tưởng cho chức năng

- Ý tưởng tăng độ sáng cho ảnh là cộng thêm một số a vào các pixel màu.
- Độ sáng tăng lên của các pixel phụ thuộc vào số a và điểm màu hiện tại.
- Nếu a là số âm thì ảnh sẽ trở nên tối đi và ngược lại sẽ sáng lên.
- Xử lý trường hợp vượt quá số màu thì sẽ giữ giá trị điểm màu trong khoảng 0-255 để tránh việc bị đảo giá trị 256 thành 0 hay -1 thành 255 dẫn đến việc bị cháy ảnh.

##### Thực hiện

- Sử dụng phép cộng ma trận cho một số a để tăng độ sáng cho ảnh.
- Trước khi sử dụng phép cộng sử dụng hàm int2float\_img() để chuyển khoảng màu của ảnh từ 0 - 1. Tuy nhiên khi chuyển như thế thì số a sẽ phải giảm đi số lần giống với ảnh.
- Sau khi cộng ma trận với a, để tránh việc giá trị màu không nằm trong khoảng từ 0-1. Sử dụng hàm numpy.clip() để giới hạn ma trận trong khoảng từ 0-1.

---

```
1 brighter\_img = np.clip(brighter\_img, 0, 1)
```

---

- Cuối cùng sẽ sử dụng hàm float2int\_img() để trả ảnh lại về giá trị màu từ 0-255.

#### 3.2 Thay đổi độ tương phản

##### Ý tưởng cho chức năng

- Ý tưởng tăng tương phản cho ảnh là nhân thêm một số a vào các pixel màu, vì khi nhân lên khoảng cách giữa các giá trị màu sẽ theo cấp số nhân a vì vậy sẽ gia tăng khoảng cách giữa các điểm màu cũng chính là tương phản.

- Độ tương phản tăng lên của các pixel phụ thuộc vào số a và điểm màu hiện tại.
- Trong trường hợp a là giá trị âm nếu giá trị a quá nhỏ thì sẽ dẫn đến trở thành ảnh màu đen, ngược lại khi số a lớn thì ảnh sẽ trở thành ảnh trắng. - Xử lý trường hợp vượt quá số màu thì sẽ giữ giá trị điểm màu trong khoảng 0-255 để tránh việc bị đảo giá trị 256 thành 0 hay -1 thành 255 dẫn đến việc bị cháy ảnh.

## Thực hiện

- Sử dụng phép nhân ma trận cho một số a để tăng độ sáng cho ảnh.
- Trước khi sử dụng phép cộng sử dụng hàm int2float\_img() để chuyển khoảng màu của ảnh từ 0 - 1. Không giống như phép cộng ma trận ở phép nhân a không cần phải thay đổi.
- Sau khi nhân ma trận với a, để tránh việc giá trị màu không nằm trong khoảng từ 0-1. Sử dụng hàm numpy.clip() để giới hạn ma trận trong khoảng từ 0-1.

---

```
1 contrast_img = np.clip(contrast_img, 0, 1)
```

---

- Cuối cùng sẽ sử dụng hàm float2int\_img() để trả ảnh lại về giá trị màu từ 0-255.

## 3.3 Lật ảnh (ngang - dọc)

### Ý tưởng cho chức năng

#### Lật ảnh ngang

- Ý tưởng để lật ảnh ngang chính là đọc ảnh theo từng cột chiều từ phải sang trái.

#### Lật ảnh dọc

- Ý tưởng để lật ảnh ngang chính là đọc ảnh theo từng dòng chiều từ dưới lên trên.

## Thực hiện

- Ở cả hai chức năng lật ảnh dọc hay lật ảnh ngang. Ta sẽ sử dụng kỹ thuật slicing trong python để đảo thứ tự đọc.
- Nguyên tắc của kỹ thuật slicing cho list:

Lst[ Initial : End : IndexJump ]

- Kỹ thuật Slicing sẽ duyệt từ phần tử Initial đến phần tử End và có bước nhảy IndexJump tương tự như vòng lặp for.

#### Lật ảnh ngang

- Nhưng khi giá trị như initial, End,... là dấu âm sẽ đảo ngược. Ví dụ với mảng có 5 phần tử khi initial là -2 thì giá trị initial sẽ đọc lần lượt là 2, 1, 0 .

- Đổi với chức năng lật ảnh ngang để đọc ảnh theo từng cột theo chiều từ trái sang phải sẽ slicing theo "img[:, ::-1]". Cần phải truy xuất theo chiều thứ 2 của ảnh là cột và với phần IndexJump có giá trị là -1 nên ảnh sẽ đọc theo chiều ngược lại là từng cột từ phải sang trái.

#### Lật ảnh đọc

- Tương tự như đổi với lật ảnh ngang, chức năng lật ảnh đọc cần slicing theo "img[::-1]" tức là đọc ảnh theo chiều ngược lại từ dưới lên trên.

- Sử dụng thêm các hàm int2float\_img() và float2int\_img() để giúp cho việc xử lý ảnh được nhanh hơn và tránh việc lệch màu ra khỏi giá trị int 8 bit và dùng numpy.clip() để đảm bảo ảnh luôn đúng trong khoảng 0-1 (float) hay 0-255 (int)

## 3.4 Chuyển đổi ảnh RGB thành xám/sepia

### Ý tưởng cho chức năng

#### Ảnh xám

- Ý tưởng để chuyển đổi từ ảnh RGB sang ảnh xám (gray scale) sử dụng phương pháp luminosity (luminosity method). Lấy giá trị từ tổng trọng số từ các kênh màu RGB. Có công thức là:

$$\text{Grayscale} = (0.3 * R) + (0.59 * G) + (0.11 * B)$$

- Có nhiều cách thức để chuyển ảnh từ RGB sang ảnh xám tuy nhiên với phương pháp luminosity sử dụng thông số từ các kênh màu sẽ giúp cho ảnh đầu ra sáng hơn, rõ nét và dịu hơn so với các phương pháp khác.

**Ảnh sepia** - Hiệu ứng Sepia là một loại hiệu ứng màu sẽ biến đổi hình ảnh ban đầu thành các gam màu nâu ấm áp, mang tính chất cổ điển. Hiệu ứng này thường được sử dụng để làm cho hình ảnh trở nên cổ điển và tạo ra cảm giác hoài niệm, như những bức ảnh từ những ngày xưa.

Ma trận cho công thức ảnh sepia:

[0.393, 0.769, 0.189]

[0.349, 0.686, 0.168]

[0.272, 0.534, 0.131]]

- Để cho ra được bức ảnh màu sepia ta sẽ nhân hai ma trận ảnh với ma trận chuyển vị của sepia. Đổi với mỗi dòng trong ma trận sepia tương ứng cho giá trị RGB trong ảnh. Ta tính toán như sau.

- Ví dụ đổi với R (Red) = 150, G (Green) = 100, B (Blue) = 50 trên ảnh và dòng đầu tiên của ma trận sepia [0.393, 0.769, 0.189]. Ta tính như sau:

+ Nhân từng giá trị R G B cho tương ứng từng giá trị trong sepia. Giá trị tương ứng cho dòng đầu là R (Red) nên pixel có giá trị R tương ứng mới là:

$$- R\text{-new} = R * \text{sepia}[0] + G * \text{sepia}[1] + B * \text{sepia}[2]$$

- $R\text{-new} = 150 * 0.393 + 100 * 0.769 + 50 * 0.189$
  - $R\text{-new} = 145.3$  và cũng tương tự cho G và B
- Tuy nhiên thay vì xử lý từng dòng như thế ta có thể dựa trên nguyên tắc nhân ma trận là  $IMG * SEPIA^T$  thì điểm  $IMG[1][1]$  sẽ bằng  $IMG[1] * SEPIA^T[1]$  cũng chính là ví dụ ta vừa làm ở trên.

## Thực hiện

### Ảnh xám

- Với việc sử dụng phương pháp luminosity ta sẽ sử dụng list gray\_filter để lưu giá trị luminosity

```
1 gray_filter = np.array([0.29, 0.59, 0.11])
```

- Sau đó nhân theo mảng ảnh sử dụng kỹ thuật slicing  $[:, :, :3]$  tương ứng với việc chỉ sử dụng ba giá trị màu RGB trong từng pixel nhân với gray\_filter.

+  $[:, :, :3]$  là kỹ thuật slicing để lấy 3 phần tử ở chiều thứ ba chính là RGB

```
1 gray_img = np.dot(img[:, :, :3], gray_filter)
```

### Ảnh sepia

- Sau khi có thể áp dụng nhân ma trận vào xử lý thì phần thực hiện trở nên đã dễ hơn. Sử dụng hàm nhân ma trận **numpy.dot()** để nhân ma trận ảnh với ma trận chuyển vị của ma trận màu sepia

```
1 sepia_img = np.dot(img, sepia_filter.T)
```

- Sử dụng thêm các hàm **int2float\_img()** và **float2int\_img()** để giúp cho việc xử lý ảnh được nhanh hơn và tránh việc lệch màu ra khỏi giá trị int 8 bit và dùng **numpy.clip()** để đảm bảo ảnh luôn đúng trong khoảng 0-1 (float) hay 0-255 (int).

## 3.5 Làm mờ/ sắc nét ảnh

### Ý tưởng cho chức năng

- Để thực hiện được hai chức năng này ta cần phải sử dụng tích chập ảnh (convolution) với kernel tương ứng với làm mờ và làm sắc nét ảnh để cho ra ảnh tương ứng.
- Tích chập (Convolution) thường được sử dụng để thực hiện các thao tác như lọc (filtering) hoặc phát hiện đặc trưng. Khi thực hiện phép tích chập, ma trận kernel được trượt qua các

vùng con của dữ liệu đầu vào (như một cửa sổ trượt) và thực hiện phép nhân từng phần tử của kernel với các giá trị tương ứng trong vùng con của dữ liệu. Sau đó, các kết quả phép nhân được cộng lại để tạo thành một giá trị mới cho điểm dữ liệu đang xét.

Một ví dụ cho việc sử dụng convolution ta sử có ví dụ sau:

Ta có ma trận A:

$$\begin{bmatrix} 1, 2, 2 \\ 2, 1, 2 \\ 2, 2, 1 \end{bmatrix}$$

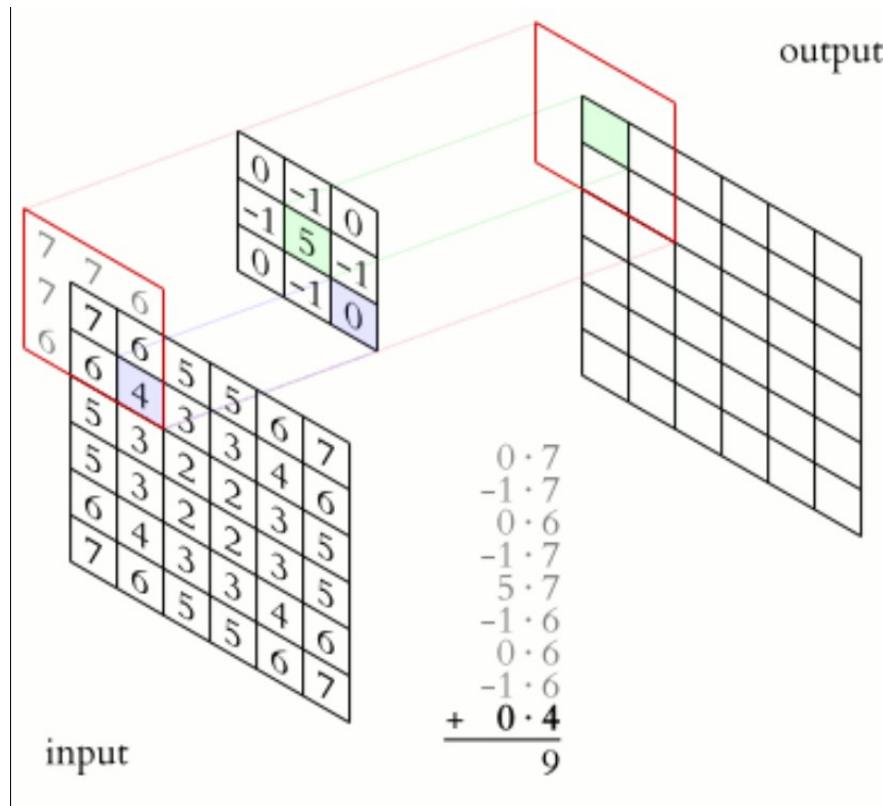
Ta có kernel K:

$$\begin{bmatrix} 1/9, 1/9, 1/9 \\ 1/9, 1/9, 1/9 \\ 1/9, 1/9, 1/9 \end{bmatrix}$$

Sau khi thêm padding:

$$\begin{bmatrix} 1, 1, 2 \\ 1, 1, 2 \\ 2, 2, 1 \end{bmatrix}$$

Khi sử dụng convolution ta đối với vị trí A[0][0] và Kernel K ta có kết quả như sau: A-new[0][0] = 2 \* (1/9) + 1 \* (1/9) + 2 \* (1/9) + 1 \* (1/9) + 1 \* (1/9) + 1 \* (1/9) + 2 \* (1/9) + 1 \* (1/9) + 2 \* (1/9) = 13/9 xấp xỉ 1,44

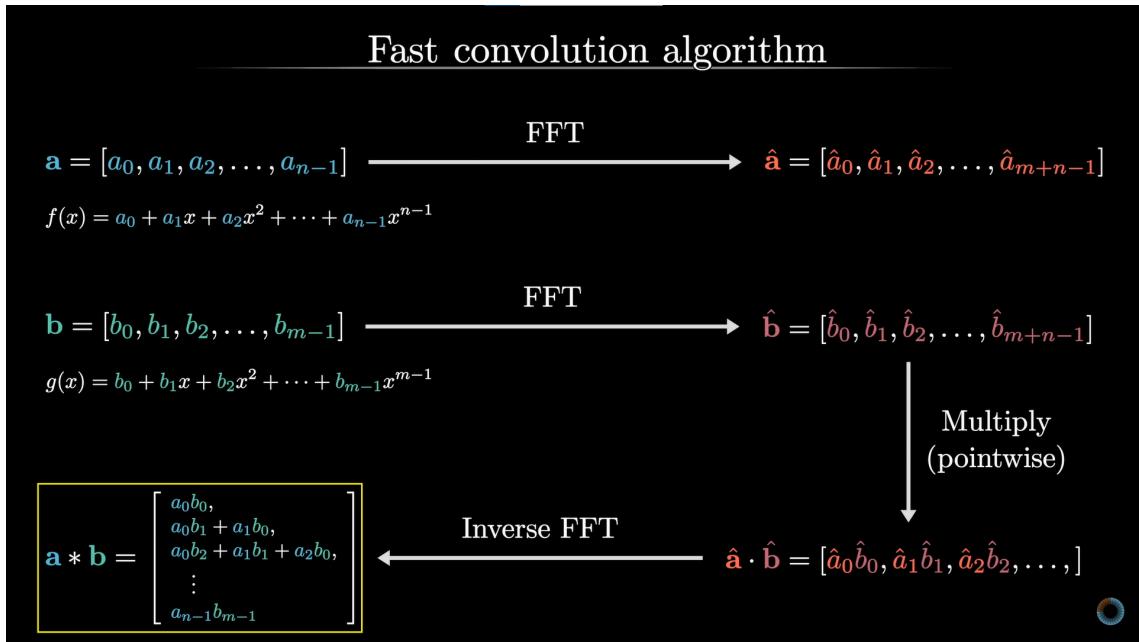


- Trước bước sử dụng convolution ta buộc phải xoay ma trận kernel  $180^\circ$ . Tuy nhiên kernel mà ta đang sử dụng là ma trận đối xứng nên việc xoay  $180^\circ$  không có ý nghĩa.

Phương pháp là thêm các padding ảo vào ma trận gốc để các phần tử ngoài lề được xem như là tâm của một ma trận, sau đó tiếp tục với convolution.

- Như vậy convolution trên có độ phức tạp là  $O(N^2)$ . Như vậy nếu đối với ảnh lớn hoặc có kernel lớn sẽ mất rất nhiều thời gian. Vì vậy sau khi tham khảo cách khác đó chính là sử dụng phương pháp khác cho việc convolution chính là FFT convolution.

Khác với convolution thông thường FFT convolution sử dụng phép biến đổi FFT (fast Fourier transform) cho ma trận ảnh và kernel sau đó thực hiện phép nhân element-wise giữa A và K sau khi biến đổi và cuối cùng là sử dụng phép biến đổi ngược IFFT (inverse fast Fourier transform) để đưa trở lại không gian ảnh.



- Kernel được sử dụng cho chức năng làm mờ ảnh và làm sắc nét ảnh lần lượt là:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

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

## Thực hiện

- Sử dụng FFT convolution được hỗ trợ bởi numpy.fft cụ thể là hàm numpy.fft.fft2() dùng để biến đổi ma trận hai chiều sang FFT và hàm numpy.fft.ifft2() là hàm với chức năng chuyển đổi ngược lại từ FFT về không gian ảnh.

- Ta sử lý riêng biệt cho từng kênh màu như sau:

```

1   for i in range(image.shape[2]):
2       new_img[:, :, i] = np.real(np.fft.ifft2(np.fft.fft2(image
3      [:, :, i]) * np.fft.fft2(kernel, s=image[:, :, i].shape)))

```

- Trong đó ma trận new\_img này sẽ lưu trữ kết quả của phép tích chập sau khi sử dụng FFT.

---

```

1 new_img[:, :, i] = np.real(np.fft.ifft2(np.fft.fft2(image[:, :, i]) * np.fft.fft2(kernel, s=image[:, :, i].shape)))

```

---

- Trong mỗi vòng lặp, hàm thực hiện các bước sau để tính toán phép tích chập.
  - + np.fft.fft2(image[:, :, i]): Thực hiện FFT cho kênh màu thứ i của hình ảnh. FFT được sử dụng để biến đổi hình ảnh từ không gian thời gian sang không gian FFT.
  - + np.fft.fft2(kernel, s=image[:, :, i].shape): Thực hiện FFT cho kernel với kích thước tương tự như kênh màu thứ i của hình ảnh. Điều này đảm bảo rằng kernel và hình ảnh có cùng kích thước để thực element-wise.
  - + np.fft.ifft2(...): Thực hiện FFT nghịch (Inverse FFT) để quay trở lại không gian thời gian ảnh sau khi thực hiện phép nhân.
  - + np.real(...): Lấy phần thực của kết quả sau khi thực hiện Inverse FFT. Vì numpy.fft sau khi biến đổi sẽ trở thành số phức nên sau khi thực hiện phép tích chập chỉ cần quan tâm đến phần thực của kết quả, sẽ bỏ qua phần ảo.
- Đây cũng chính là hàm fftconvolution(), phần xử lý chính cho chức năng làm mờ ảnh là làm sắc nét ảnh.
- Tuy nhiên với numpy.fft sẽ sử dụng padding 0 nên có thể sẽ dẫn đến việc các pixel cạnh trên và cạnh trái của ảnh bị lỗi màu so với ảnh gốc.

### Đối với chức năng làm mờ

---

```

1 blur_img[0:3, :] = img[0:3, :]

```

---

- Dùng để gán lại 3 dòng đầu của ảnh gốc đầu vào sau khi sử dụng fftconvolution().

### Đối với chức năng làm nét

---

```

1 sharpen_img[0:3, :] = img[0:3, :]
2 sharpen_img[:, 0:2] = img[:, 0:2]

```

---

- Dùng để gán lại 3 dòng và 2 cột đầu của ảnh gốc đầu vào sau khi sử dụng fftconvolution().
- Sử dụng thêm các hàm int2float\_img() và float2int\_img() để giúp cho việc xử lý ảnh được nhanh hơn và tránh việc lệch màu ra khỏi giá trị int 8 bit và dùng numpy.clip() để đảm bảo ảnh luôn đúng trong khoảng 0-1 (float) hay 0-255 (int).

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

#### Ý tưởng cho chức năng

- Ý tưởng cơ bản: Chỉ đọc các điểm màu bên trong khung cắt mong muốn.
- Sau đó, dựa vào chiều cao, độ rộng của ảnh có sẵn, và cạnh của hình thành phẩm, ta tính được tọa độ trái trên, trái dưới, phải trên, phải dưới của ảnh mới.
- Cuối cùng chép các điểm màu mới trong theo các tọa độ đã xác định ở trên.

#### Thực hiện

- Bước đầu tiên: lấy chiều cao và chiều rộng từ ảnh.
- Bước 2: Tính chiều dài rộng của ảnh cần cắt.

---

```
1 size = int(min(height, width) * scale)
```

---

- + Chọn cảnh nhỏ hơn trong hai cạnh của ảnh để nhân với scale làm kích thước cho ảnh sau khi cắt.
- + Bắt buộc phải ép kiểu int vì size cũng đồng thời là số điểm ảnh có trên ảnh mới, mà điểm ảnh thì không có float ví dụ như điểm ảnh thứ 200.5 là không tồn tại.
- + Scale được chỉnh mặc định là 0.6.
- Bước 3: Tính tọa độ trái trên, trái dưới, phải trên, phải dưới của ảnh mới.

---

```
1 x = (width - size) // 2
2 y = (height - size) // 2
```

---

- + toán tử // là toán tử phép chia nhưng sẽ trả về kết quả là số nguyên
- + x là tọa độ trái trên.
- + y là tọa độ trái dưới.
- + 2 tọa độ còn lại bên phải sẽ được tính trực tiếp bằng cách cộng với size.
- Bước 4: Trả về ảnh bằng sử dụng kỹ thuật slicing để trả về các điểm ảnh đã xác định.

---

```
1 img[y:y + size, x:x + size]
```

---

### 3.7 Cắt ảnh theo khung tròn

#### Ý tưởng cho chức năng

- Sử dụng phương trình đường tròn để tạo mask dùng để cắt ảnh:

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

- Với a, b là tọa độ tâm của đường tròn
- R là bán kính của đường tròn
- Những điểm không thuộc phương trình đường tròn sẽ nhận giá trị màu 0 (màu đen).

#### Thực hiện

- Bước đầu tiên: Lấy chiều dài chiều, chiều rộng của ảnh để tính tâm đường tròn.
- Bước 2: Tính tâm đường tròn bằng cách  $x\_center = width // 2$ ,  $y\_center = height // 2$ .
- Bước 3: Sử dụng hàm `numpy.ogrid()` để tạo ra một grid hai chiều theo kích thước của ảnh.

---

```
1 x , y = np.ogrid[:width, :height]
```

---

- Bước 3: Tính bán kính của đường tròn bằng cách lấy  $\min(x\_center, y\_center)$  để đảm bảo đường tròn nội tiếp bên trong ảnh.
- Bước 4: Tạo mask để lưu các điểm ảnh theo phương trình đường tròn đã ghi ở trên

---

```
1 mask = ((x - x_center) ** 2 + (y - y_center) ** 2) <=
radius ** 2
```

---

- "...<= radius \*\*2" được dùng để lấy các điểm bên trong phương trình đường tròn
- Bước 5: Tạo ra một mảng 0 numpy giống với ảnh gốc bằng hàm `numpy.zeros_like()` để lấy các điểm ảnh đã lưu trong mask trong phương trình đường tròn.
- Bước cuối: lưu các điểm ảnh vào mảng dưới trên các điểm màu lưu trong mask

---

```
1 circle_img[mask] = img[mask]
```

---

- Sử dụng thêm các hàm `int2float_img()` và `float2int_img()` để giúp cho việc xử lý ảnh được nhanh hơn và tránh việc lệch màu ra khỏi giá trị int 8 bit và dùng `numpy.clip()` để đảm bảo ảnh luôn đúng trong khoảng 0-1 (float) hay 0-255 (int).

### 3.8 Cắt ảnh theo khung 2 ellipse chéo nhau

#### Ý tưởng cho chức năng

- Sử dụng phương trình ellipse nghiêm để tạo mask dùng để cắt ảnh:

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

- Với a, b là giá trị hai đường chéo của ellipse
- h, k lần lượt là tâm (h, k) của ellipse

- Đây chỉ là phương trình một hình ellipse chéo theo đường chéo của ảnh, để thực hiện hai đường ellipse trên hai đường chéo thì ta tạo thêm một hình ellipse nữa đối xứng qua tâm ellipse tức là đảo lại giá trị a b của ellipse ban đầu là giá trị b a của ellipse sau.

- Tiếp theo tạo ra mask bằng cách giao các điểm ảnh thu được của ellipse.  
- Những điểm không thuộc mask sẽ nhận giá trị màu 0 (màu đen). - Từ phương trình ban đầu với các phép biến đổi suy ra được tỉ lệ của a b với điểm giao bằng 1/4 cạnh đối với ảnh 512x512 là:

$$\frac{389.4}{a} = 1 + \frac{187,66}{b}$$

Từ đó tính ra được a b gần đúng lần lượt là 450 và 244.2.

#### Thực hiện

Tương tự như cắt ảnh hình tròn ta có các bước:

- Sử dụng phương trình đường tròn để tạo mask dùng để cắt ảnh:

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

- Với a, b là toạ độ tâm của đường tròn
- R là bán kính của đường tròn
- Những điểm không thuộc phương trình đường tròn sẽ nhận giá trị màu 0 (màu đen).

## Thực hiện

- Bước đầu tiên: Lấy chiều dài chiều, chiều rộng của ảnh để tính tâm đường tròn.
- Bước 2: Tính tâm đường tròn bằng cách  $x\_center = width // 2$ ,  $y\_center = height // 2$ .
- Bước 3: Sử dụng hàm numpy.ogrid() để tạo ra một grid hai chiều theo kích thước của ảnh.

---

```
1   x , y = np.ogrid[:width, :height]
```

---

- Bước 3: Tính bán kính của đường tròn bằng cách lấy  $\min(x\_center, y\_center)$  để đảm bảo đường tròn nội tiếp bên trong ảnh.
- Bước 4: Tạo hai mask để lưu các điểm ảnh theo 2 phương trình đường ellipse đối xứng đã ghi ở trên

---

```
1   mask1 = ((x - x_center) + (y - y_center)) ** 2 / (450 ** 2) +
        ((x - x_center) - (y - y_center)) ** 2 / (244.2 ** 2) <= 1
2   mask2 = ((x - x_center) + (y - y_center)) ** 2 / (244.2 ** 2)
        + ((x - x_center) - (y - y_center)) ** 2 / (450 ** 2) <= 1
```

---

- Bước 5: Dùng toán tử "|" hay or để giao hai 2 mask trên

---

```
1   mask = mask1 | mask2
```

---

- Các bước còn lại như ở chức năng cắt ảnh khung hình tròn.

## 4 Các hàm bổ trợ khác

### 4.1 importImage()

- Input: đường dẫn đến vị trí của ảnh.
- Output: trả về thông tin ảnh.
- Hàm này được dùng để lấy thông tin của ảnh bằng PIL.Image.open() và trả về thông tin ảnh sau khi chuyển thành numpy.array().

### 4.2 show\_img()

- Input: Ảnh gốc, ảnh mới, chức năng ảnh đầu, ảnh khác, chức năng ảnh sau.
- Output: Show ra tất cả các ảnh đầu vào.
- Dùng để show ra ảnh gốc và ảnh chức năng để so sánh. Trường hợp sẽ in ra ba ảnh là trong trường hợp sử dụng chức năng làm nét ảnh vì ảnh đầu vào sẽ là ảnh đã được làm mờ.

### 4.3 export\_img()

- Input: Ảnh sau khi đã chỉnh sửa chức năng, tên chức năng.
- Output: Ảnh được xuất ra ngoài folder.
- Dùng để lưu ảnh ra ngoài folder.

### 4.4 int2float\_img()/float2int\_img()

**int2float\_img()** - Input: ảnh gốc

- Output: Trả về ảnh đã được chuyển thành 0-1, giá trị lớn nhất của ảnh.
- Dùng để chuyển giá trị ảnh về dạng 0-1 bằng cách chia cho giá trị lớn nhất của ảnh.

**float2int\_img()** - Input: ảnh sau khi chuyển đổi về 0-1, và giá trị lớn nhất của ảnh gốc.

- Output: Trả về ảnh đã được chuyển thành 0-255.
- Dùng để chuyển giá trị ảnh về dạng 0-255 bằng cách nhân cho giá trị lớn nhất của ảnh.

### 4.5 fftconvolution()

- Input: Ảnh, kernel.
- Output: Ảnh đã xử lý FFT convolution với kernel.

- Chi tiết xử lý đã được đề cập ở phần chức năng làm mờ, làm sắc nét

## 4.6 Các hàm printImage cho từng chức năng

- Input: Ảnh gốc.
- Output: Ảnh đã được xử lý, show ảnh, xuất ảnh.
- Hàm dùng để thực hiện xử lý ảnh, show ảnh và export ảnh.

## 4.7 choice()

- Input: Không có.
- Output: Xuất ra màn hình lựa chọn chức năng, trả về lựa chọn của người dùng.
- Dùng để lấy lựa chọn xử lý chức năng mà người dùng mong muốn. Thứ tự đánh số tương tự như mục 3 (Mô tả chức năng) nhưng có thêm lựa chọn số 0 là xử lý tất cả các chức năng ở mục 3.

## 4.8 processing()

- Input: Ảnh gốc, lựa chọn chức năng.
- Output: Ảnh đã xử lý dựa trên chức năng đã chọn.
- Dùng để gọi lại các hàm printImage để xử lý chức năng được lựa chọn. Nếu lựa chọn không có trong hàm choice() sẽ in ra chuỗi "Invalid option"

## 4.9 main()

- Input: Không có.
- Output: Thực hiện chức năng noise trên.
- Dùng để lấy đường dẫn đến ảnh cũng như lựa chọn của người dùng sau đó xử lý và trả về ảnh đã xử lý xong.

## 5 Kết quả

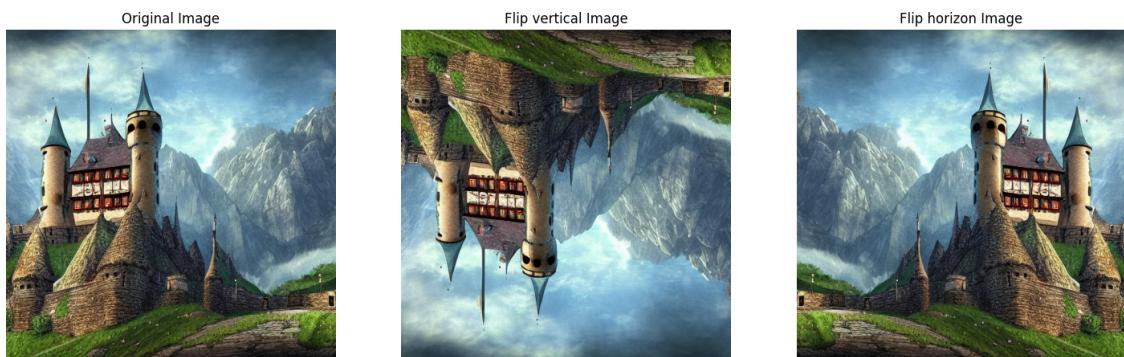
### 5.1 Thay đổi độ sáng cho ảnh



### 5.2 Thay đổi độ tương phản

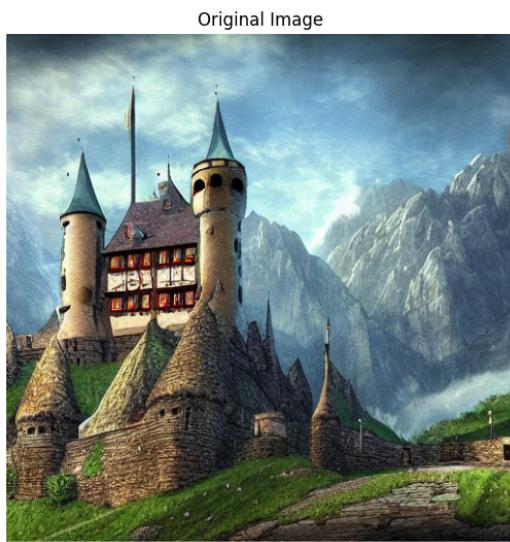


### 5.3 Lật ảnh (ngang - dọc)



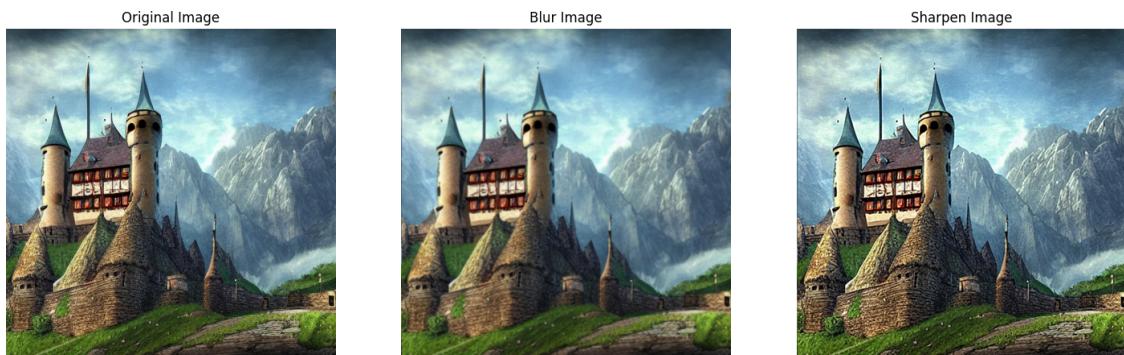
### 5.4 Chuyển đổi ảnh RGB thành xám/sepia





## 5.5 Làm mờ/ sắc nét ảnh

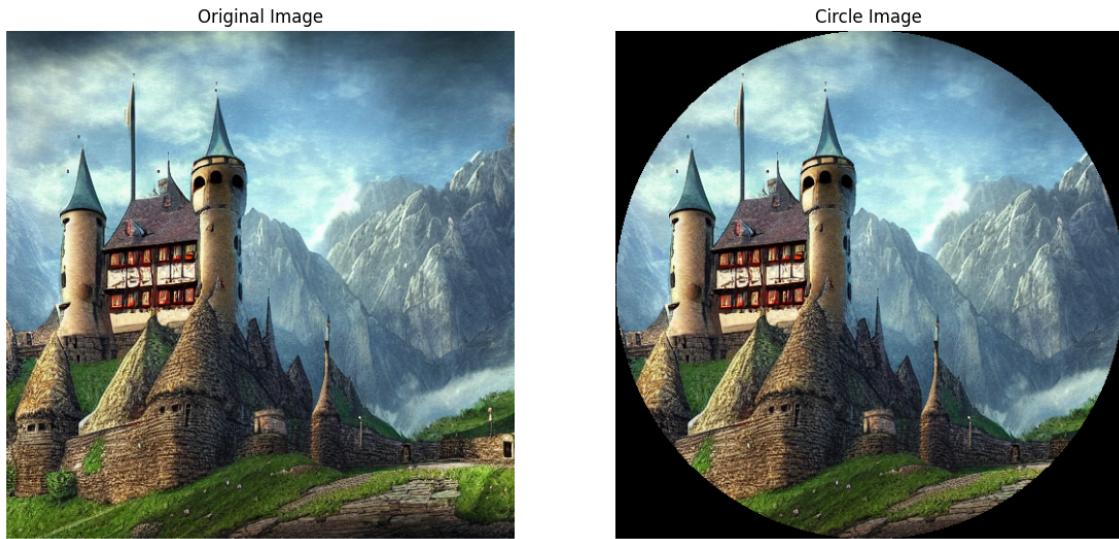




## 5.6 Cắt ảnh theo kích thước (cắt ở trung tâm)



## 5.7 Cắt ảnh theo khung tròn



## 5.8 Cắt ảnh theo khung 2 ellipse chéo nhau



## 5.9 Thời gian xử lý

- Bảng dưới đây là thời gian xử lý trung bình cho mỗi chức năng (một chức năng nhỏ cho chức năng có hai lựa chọn) dựa trên nhiều lần xử lý.

STT	Tên chức năng	Thời gian xử lý trung bình (s)
1	Thay đổi độ sáng cho ảnh	0.012
2	Thay đổi độ tương phản	0.013
3	Lật ảnh ngang, dọc	0.019
4	Chuyển đổi ảnh RGB thành xám, sepia	0.013 (gray) / 0.026 (sepia)
5	Làm mờ, sắc nét ảnh	0.125
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	0.005
7	Cắt ảnh theo khung hình tròn	0.025
8	Cắt ảnh theo khung 2 ellipse chéo nhau	0.035

## 6 Tài liệu tham khảo

### Tài liệu

- [1] Ý tưởng ảnh màu xám
- [2] Ý tưởng ảnh màu sepia
- [3] Ý tưởng ảnh màu sepia
- [4] Kỹ thuật Slicing
- [5] Kernel cho ảnh và lý thuyết convolution
- [6] Lý thuyết convolution và FFT convolution
- [7] Phương trình ellipse nghiên
- [8] Phương trình ellipse nghiên
- [9] Thư viện numpy