

ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA VẬT LÝ



NGUYỄN TIẾN ĐẠT

KHAI THÁC MỘT SỐ THƯ VIỆN PYTHON CHO BÀI  
TOÁN THỊ GIÁC MÁY TÍNH

Tiểu luận

Ngành Kỹ thuật điện tử tin học  
(Chương trình đào tạo chuẩn)

Hà Nội - 2024

ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
**KHOA VẬT LÝ**



NGUYỄN TIỀN ĐẠT

**KHAI THÁC MỘT SỐ THƯ VIỆN PYTHON CHO BÀI  
TOÁN THỊ GIÁC MÁY TÍNH**

Tiểu luận

Ngành Kỹ thuật điện tử tin học  
(Chương trình đào tạo chuẩn)

Giảng viên hướng dẫn: TS. Phạm Tiến Lâm

Hà Nội - 2024

*“Cái tôi và sự hiểu biết tỳ lệ nghịch với nhau. Hiểu biết càng nhiều cái tôi càng bé. Hiểu biết càng ít, cái tôi càng to. ”*

Albert Einstein

## ***Lời cảm ơn***

Để có thể hoàn thành tiểu luận này, tôi xin gửi lời cảm ơn chân thành tới thầy TS. Phạm Tiên Lâm. Trong suốt quá trình thực hiện đề tài này, thầy luôn là người đồng hành và hướng dẫn tôi nhiệt tình, với những lời góp ý nhận xét rất quý báu đã tạo điều kiện thuận lợi cho tôi để hoàn thành tốt tiểu luận.

Tôi xin gửi lời cảm ơn chân thành tới các anh chị, các bạn, đặc biệt là anh Đặng Văn Báu đã giúp đỡ tôi rất nhiều trong quá trình thực hiện đề tài này.

Nhân dịp này, tôi cũng xin gửi lời cảm ơn chân thành tới bố mẹ, người thân đã luôn bên cạnh cổ vũ, động viên tôi trong thời suốt thời gian học tập

# Mục lục

<b>Lời cảm ơn</b>	ii
<b>Danh sách hình vẽ</b>	vi
<b>Danh sách bảng</b>	viii
<b>MỞ ĐẦU</b>	1
<b>1 :Tổng quan về xử lý ảnh và các kỹ thuật trích dẫn đặc trưng</b>	<b>2</b>
1.1 Tổng quan về xử lý ảnh: . . . . .	2
1.1.1 Thể nào là ảnh số: . . . . .	2
1.1.2 Quá trình xử lý ảnh: . . . . .	2
1.1.3 Không gian màu: . . . . .	4
1.1.4 Độ phân giải ảnh: . . . . .	4
1.2 Các kỹ thuật trích xuất ảnh: . . . . .	6
1.2.1 Biến đổi cường độ và lọc không gian (Intensity Transformations và Spatial Filtering): . . . . .	6
Một vài vấn đề cơ bản: . . . . .	6
Một vài vấn đề cơ bản về hàm biến đổi cường độ: . . . . .	8
1.2.2 Đổi chiều histogram: . . . . .	11
1.2.3 Làm mờ ảnh, làm sắc nét: . . . . .	14
1.2.4 Màu sắc ảnh: . . . . .	16
Các mẫu màu: . . . . .	17
Xử lý giả màu: . . . . .	18
1.2.5 Phép biến đổi hình thái học: . . . . .	20
Co rút: . . . . .	20
Giãn nở: . . . . .	20
Mở (Opening) và Đóng (Closing): . . . . .	21
Lý thuyết trúng hoặc trượt: . . . . .	22
1.2.6 Phép biến đổi hình học: . . . . .	23
Phép dịch: . . . . .	23
Phép quay: . . . . .	24
Phép chuyển đổi tỷ lệ: . . . . .	25
Phép biến đổi phản xạ: . . . . .	27
1.2.7 Phân đoạn ảnh: . . . . .	27
Nhận diện điểm cô lập: . . . . .	28

Nhận diện đường: . . . . .	29
Nhận diện cạnh: . . . . .	30
Nhận diện cạnh Laplacian: . . . . .	33
<b>2 Các kỹ thuật xử lý ảnh:</b>	<b>34</b>
2.1 Các lệnh khai báo và đọc dữ liệu: . . . . .	34
2.1.1 Lệnh khai báo: . . . . .	34
2.1.2 Đọc dữ liệu: . . . . .	34
2.2 Các lệnh liên quan màu sắc, kích thước ảnh: . . . . .	36
2.2.1 Các lệnh liên quan màu sắc: . . . . .	36
Lệnh chuyển đổi giữa các không gian màu: . . . . .	36
Lệnh chuyển đổi từ ảnh màu sang ảnh xám: . . . . .	37
Lệnh lấy ra một kênh màu của ảnh đầu vào: . . . . .	38
Lệnh lấy giá trị từng màu trong không gian màu của 1 pixel: . . . . .	39
2.2.2 Lệnh liên quan đến kích thước ảnh: . . . . .	39
Lệnh lấy ra kích thước ảnh: . . . . .	39
Lệnh thay đổi kích thước ảnh: . . . . .	40
2.3 Vẽ với OpenCV: . . . . .	40
2.3.1 Các hình khối cơ bản: . . . . .	40
Hình chữ nhật: . . . . .	40
Hình tròn: . . . . .	41
Văn bản: . . . . .	43
Vẽ đường thẳng: . . . . .	43
Vẽ đa giác: . . . . .	44
2.3.2 Các tác vụ cơ bản với ảnh: . . . . .	46
Cộng ảnh: . . . . .	46
Dán ảnh: . . . . .	47
Cắt ảnh: . . . . .	48
Trộn ảnh: . . . . .	49
2.4 Biến đổi cường độ: . . . . .	49
2.4.1 Biến đổi Log: . . . . .	49
2.4.2 Phép biến đổi lũy thừa: . . . . .	50
2.4.3 Biến đổi tuyến tính: . . . . .	51
2.5 Làm mịn ảnh: . . . . .	53
2.5.1 Blur: . . . . .	53
2.5.2 Gaussian_Blur: . . . . .	55
2.5.3 Median_Blur: . . . . .	56
2.5.4 Bilateral_Blur: . . . . .	57
2.6 Phép biến đổi hình thái: . . . . .	58

2.6.1	Đọc, tạo bộ lọc:	58
2.6.2	Giãn nở hình ảnh:	59
2.6.3	Co rút hình ảnh:	60
2.6.4	Phép mở:	61
2.6.5	Phép đóng:	62
2.7	Đạo hàm ảnh:	64
2.7.1	Đạo hàm theo x và y:	64
2.7.2	Đạo hàm ảnh có trọng số:	65
2.7.3	Đạo hàm Laplacian:	66
2.8	Làm sắc nét ảnh:	68
2.9	Quay hình:	71
2.10	Lật ảnh:	72
2.11	Nhận diện các phần của vật thể:	73
2.11.1	Nhận diện góc:	73
2.11.2	Nhận diện cạnh:	74
2.11.3	Nhận diện đường bao:	75
<b>3</b>	<b>Diffusion Model</b>	<b>77</b>
3.1	Tổng quan về hoạt động của mô hình:	77
3.1.1	Forward Process:	77
3.1.2	Reverse Process:	78
3.2	Mục đích sử dụng:	79
3.2.1	Giảm nhiễu:	79
3.2.2	Tạo ảnh:	80
3.2.3	Tăng cường độ phân giải:	81
3.3	Dữ liệu cần:	83
3.3.1	Dữ liệu huấn luyện:	83
Giảm nhiễu:	83	
Tạo ảnh:	83	
Tăng chất lượng ảnh:	83	
3.3.2	Dữ liệu kiểm tra mô hình:	83
Giảm nhiễu:	83	
Tạo ảnh:	84	
Tăng chất lượng ảnh:	84	
3.4	Các ứng dụng chính của Diffusion Model:	84
<b>Kết Luận</b>	<b>86</b>	
<b>Tài liệu tham khảo</b>	<b>87</b>	

## Danh sách hình vẽ

1.1	Mẫu 4x4 và màu sắc.	3
1.2	Ảnh hưởng độ phân giải không gian	5
1.3	Ảnh hưởng độ phân giải cường độ	6
1.4	Lân cận của 1 điểm ảnh	7
1.5	Biểu diễn một số hàm thay đổi cường độ	8
1.6	Ảnh x-rays tuyến vú	9
1.7	Phổ Fourier	9
1.8	Biểu đồ biến đổi Gamma	10
1.9	Ví dụ biến đổi Gamma	11
1.10	Histogram của ảnh	13
1.11	Bộ lọc làm mờ theo không gian.	14
1.12	Hoạt động của bộ lọc tần số.	16
1.13	Mẫu màu RGB	18
1.14	Phương pháp mã màu	19
1.15	Phép mờ	21
1.16	Phép đóng	22
1.17	Phép dịch chuyển	23
1.18	Phép quay	24
1.19	Phép chuyển đổi tỷ lệ	26
1.20	Phép phản xạ	27
1.21	Điểm cô lập	29
1.22	Kết quả các phản hồi lớn nhất khác nhau.	30
1.23	Cạnh lý tưởng.	31
1.24	Lọc cạnh có bộ lọc mịn và không	32
2.1	Lệnh cv2.imread()	35
2.2	Chuyển đổi không gian màu	36
2.3	Chuyển đổi ảnh xám	37
2.4	Lấy một kênh màu	38
2.5	Cường độ màu sắc 1 pixel	39
2.6	Kích thước một ảnh	39
2.7	Vẽ hình vuông.	41
2.8	Vẽ hình tròn.	42
2.9	Vẽ bản	44
2.10	Vẽ đường thẳng.	45
2.11	Vẽ đa giác.	46

2.12	Vẽ cộng ảnh.	47
2.13	Dán ảnh.	48
2.14	Cắt ảnh.	48
2.15	Trộn ảnh.	49
2.16	Biến đổi Log.	50
2.17	Biến đổi lũy thừa.	51
2.18	Biến đổi tuyến tính.	52
2.19	Blur.	53
2.20	Gaussian_Blur.	55
2.21	Median_Blur.	56
2.22	Bilateral_Blur.	57
2.23	Dilation	60
2.24	Erosion.	61
2.25	Opening.	62
2.26	Closing.	63
2.27	Đạo hàm Sobel.	64
2.28	Đạo hàm có trọng số.	66
2.29	Đạo hàm Laplacian.	67
2.30	Sắc nét ảnh(1).	69
2.31	Sắc nét ảnh(2).	70
2.32	Sắc nét ảnh(3).	70
2.33	Sắc nét ảnh(4).	71
2.34	Xoay ảnh.	72
2.35	Lật ảnh	72
2.36	Coner_Harris	74
2.37	Canny	75
2.38	Contour(1)	76
2.39	Contour(2)	76
3.1	Forward Process	78
3.2	Reverse Process	79

## **Danh sách bảng**

# MỞ ĐẦU

Thị giác máy là một lĩnh vực đã và đang phát triển. Khái niệm xử lý hình ảnh và thị giác máy – Computer Vision có liên quan tới nhiều ngành học và hướng nghiên cứu khác nhau. Từ những năm 1970, khi mà năng lực tính toán của máy tính ngày càng mạnh mẽ hơn, các máy tính có thể xử lý được những tệp dữ liệu lớn như hình ảnh, các đoạn phim thì khái niệm và các kỹ thuật về thị giác máy ngày càng được nhắc đến và nghiên cứu nhiều hơn cho tới ngày nay.

Thị giác máy bao gồm lý thuyết và các kỹ thuật liên quan nhằm tạo ra một hệ thống nhân tạo có thể tiếp nhận các kiến thức từ hình ảnh hoặc các tập dữ liệu đa chiều. Việc kết hợp thị giác máy với các kỹ thuật khác cho chúng ta rất nhiều ứng dụng ngoài đời sống hằng ngày. Ngày nay, ứng dụng của thị giác máy đã trở nên rất rộng lớn và đa dạng, len lỏi vào mọi lĩnh vực.

Trong thời gian thực hiện tiểu luận này, em đã từng bước tiếp cận tới cơ sở và các phép xử lý ảnh số, đồng thời cũng nghiên cứu một số thư viện trong Python để thực hiện các bước xử lý hình ảnh số nói trên như: NumPy, Matplotlib, OpenCV...

Báo cáo tiểu luận bao gồm:

- **Chương 1:** Tổng quan về xử lý ảnh và các kỹ thuật trích xuất đặc trưng.
- **Chương 2:** Các kỹ thuật xử lý ảnh.
- **Chương 3:** Mô hình khuếch tán (Diffusion Model).
- **Chương 4:** Kết luận.

# **Chương 1 :Tổng quan về xử lý ảnh và các kỹ thuật trích dẫn đặc trưng**

## **1.1 Tổng quan về xử lý ảnh:**

### **1.1.1 Thể nào là ảnh số:**

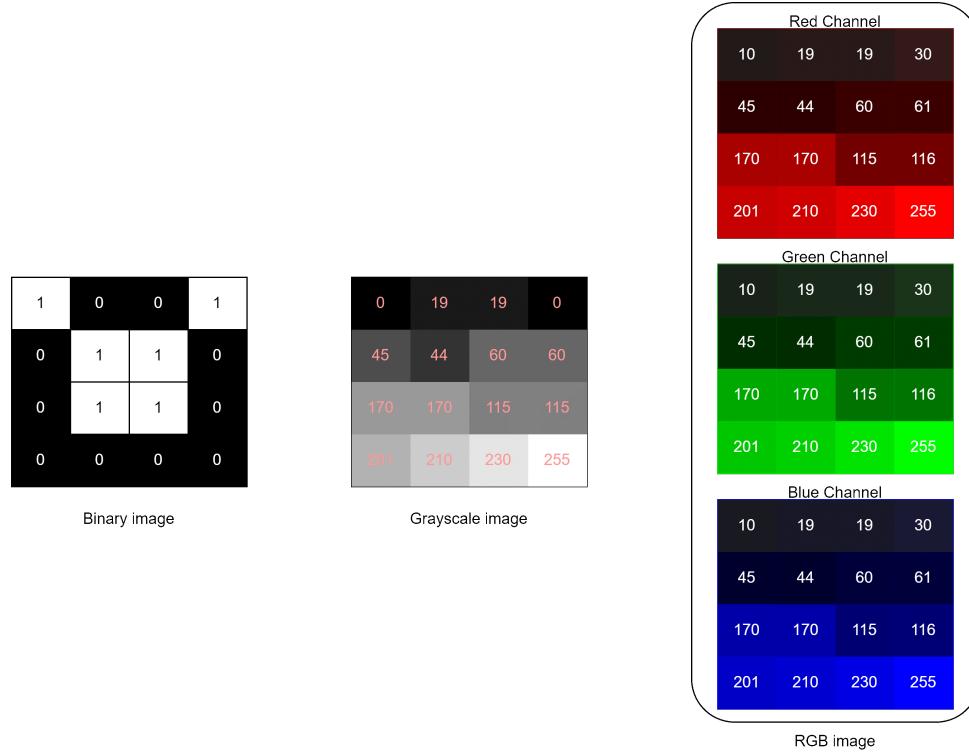
Ảnh số (digital image) là một thành phần biểu diễn hình ảnh trong hầu hết các thiết bị điện tử hiện nay như máy ảnh, điện thoại, máy tính, các công cụ hỗ trợ có sử dụng ảnh. Ảnh số là sự biểu diễn thông tin hình ảnh thông qua các con số, thường là số nhị phân. Có thể phân làm 2 loại ảnh số: ảnh raster và ảnh vector.

- *Ảnh Raster*: là tập hợp hữu hạn các giá trị số, gọi là điểm ảnh – pixel. Thông thường hình ảnh được chia thành các hàng, cột chứa điểm ảnh. Điểm ảnh là thành phần bé nhất biểu diễn ảnh, có các giá trị số biểu diễn màu sắc, độ sáng... của một thành phần trong bức ảnh. Ảnh raster thường thu được từ camera, các máy chiếu, chụp, quét... và chính là đối tượng chính trong xử lý ảnh và thị giác máy tính.
- *Ảnh vector*: là tập hợp các thành phần đơn giản của hình học như điểm, đường thẳng, hình khối... Thay vì được lưu lại thành các ma trận điểm ảnh như ảnh raster thì ảnh vector được biểu diễn dưới dạng tọa độ các thành phần trong ảnh. Chính vì điều này khiến ảnh vector có thể kéo giãn, phóng to thu nhỏ tùy ý mà không bị vỡ, không xuất hiện răng cưa như ảnh raster. Dữ liệu trong ảnh vector nhỏ, do vậy thường tiếp kiệm dung lượng lưu trữ hơn ảnh raster. Tuy thế màu sắc ảnh vector thường không thật, sắc độ ít tinh tế hơn ảnh raster. Loại ảnh này thường không xuất hiện hay được đề cập đến trong xử lý ảnh/ thị giác máy tính...

### **1.1.2 Quá trình xử lý ảnh:**

Để thu được hình ảnh số ta phải trải qua 2 quá trình cơ bản để xử lý ảnh gọi là: lấy mẫu (sampling) và lượng tử hóa (quantization). Một ảnh số  $f(x, y)$  bao gồm tọa độ  $x$  và tọa độ  $y$ . Ngoài ra, nó còn chứa biên độ của hàm. Lượng tử hóa là quá trình số hóa biên độ của hàm trong khi lấy mẫu để cập đến số hóa vị trí tọa độ. Hình ảnh số cơ bản có

3 loại: hình ảnh đơn sắc hay hình ảnh nhị phân, hình ảnh thang xám và hình ảnh màu. Hình ảnh đơn sắc là hình ảnh mà tại 1 điểm  $(x, y)$  xác định sẽ mang 1 giá trị 0 nếu là đen và là giá trị 1 nếu là trắng. Hình ảnh thang xám có giá trị cường độ trong khoảng từ 0 đến 255, trong đó 0 là màu đen, và mờ dần đến 255 là màu trắng. Trong khi đó, ảnh màu như ảnh RGB gồm 3 màu là đỏ, xanh lá cây và xanh dương. Với mỗi màu đều có cường độ từ 0 đến 255. Kết quả của quá trình lấy mẫu và lượng tử hóa là một ma trận hàng và cột chứa các số thực.



Hình 1.1: Hình ảnh mẫu  $4 \times 4$  và màu sắc của chúng ở các dạng nhị phân, hình ảnh xám, ảnh màu.

Ý tưởng chính cho việc số hóa là việc lưu trữ và truy xuất cho ảnh số nhanh và rẻ hơn nhiều so với ảnh analog. Hình ảnh số cho phép nén bằng nhiều phương pháp khác nhau để làm chúng dễ dàng hơn trong việc truyền nhận dữ liệu. Đồng thời, chúng ta có thể phân đoạn hình ảnh số bằng cách các gián đoạn như điểm, đường, cạnh. Điều này cho phép hình ảnh được sử dụng cho các ứng dụng chỉnh sửa ảnh. Hơn nữa, lĩnh vực thiết kế và tạo mẫu được thực hiện dễ dàng hơn so với hình ảnh kỹ thuật số. Do đó, nó làm giảm chi phí.

### **1.1.3 Không gian màu:**

Các không gian màu là một mô hình toán học dùng để mô tả các màu sắc trong thực tế được biểu diễn dưới dạng số học.

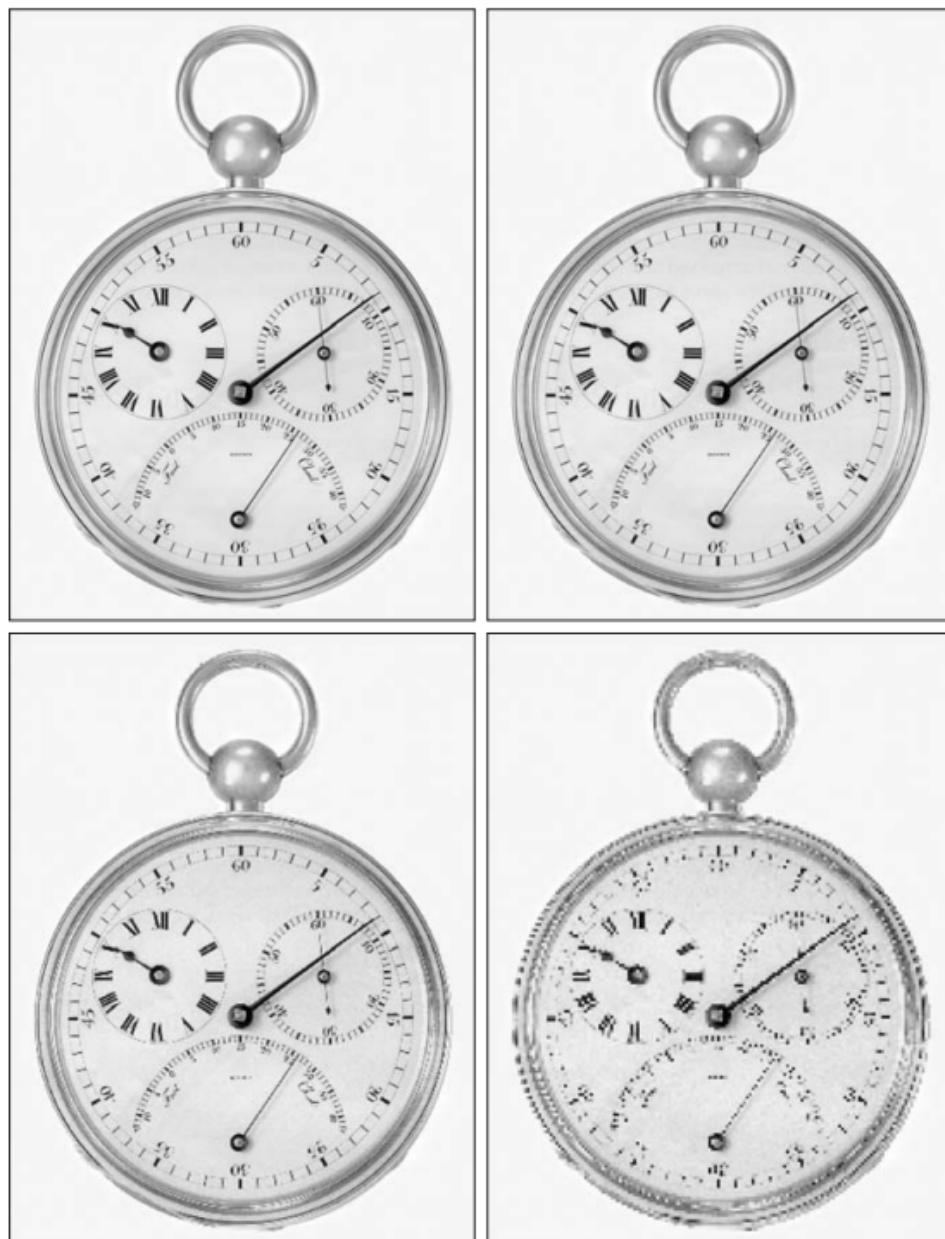
- *Không gian màu RGB*: Là không gian màu phổ biến nhất trong máy tính, máy ảnh, điện thoại và nhiều thiết bị kỹ thuật số khác. Không gian màu này khá gần với cách mắt người tổng hợp màu sắc. Nguyên lý cơ bản là sử dụng 3 màu sắc cơ bản R (đỏ), G (xanh lục) và B (xanh lam) để biểu diễn tất cả các màu sắc. Thông thường, trong mô hình 24 bit (không gian màu mặc định sử dụng bởi OpenCV - tuy nhiên OpenCV đảo 2 kênh R và B, trở thành BGR), mỗi kênh màu sẽ sử dụng 8 bit để biểu diễn, tức là giá trị R, G, B nằm trong khoảng 0 - 255. Bộ 3 số này biểu diễn cho từng điểm ảnh, mỗi số biểu diễn cho cường độ của một màu. Với mô hình biểu diễn 24 bit, số lượng màu tối đa sẽ là:  $255 \times 255 \times 255 = 16581375$ . [1]
- *Không gian màu CMYK*: chỉ mô hình màu loại trừ, thường sử dụng trong in ấn. Mô hình này dựa trên cơ sở trộn các chất màu của các màu: Cyan, Magenta, Yellow, và Key đại diện cho màu đen. Hỗn hợp của các màu CMY lý tưởng là loại trừ. Nguyên lý hoạt động của CMYK là dựa trên cơ sở hấp thụ ánh sáng. Màu mà người ta nhìn thấy là từ phần ánh sáng không bị hấp thụ. [1]
- *Không gian màu HSV*: hay còn gọi là HSB, là một cách tự nhiên hơn để mô tả màu sắc, dựa trên 3 số liệu: H (Hue) là vùng màu, S (Saturation) là độ bão hòa màu, V hay B (Brightness hay Value) là độ sáng. [1]

### **1.1.4 Độ phân giải ảnh:**

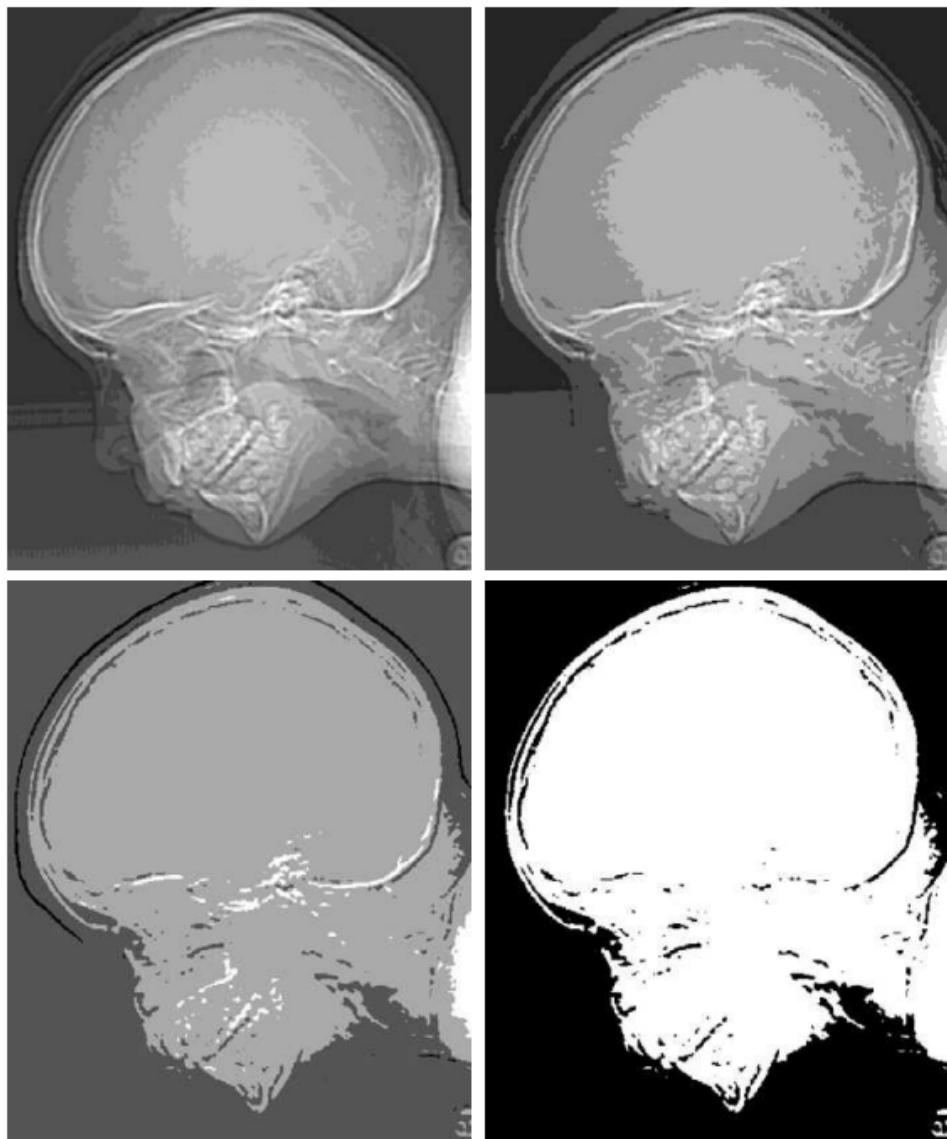
Một định nghĩa được biết đến rộng rãi của độ phân giải ảnh là mức độ chi tiết của ảnh. Nó thường được coi là tương đương với số pixel trong ảnh số. Có hai thành phần chính trong độ phân giải ảnh là: độ phân giải không gian và độ phân giải cường độ.

- *Độ phân giải không gian*: Là thước đo chi tiết nhỏ nhất có thể nhận ra trong hình ảnh. Nó thường đề cập đến số pixel được sử dụng để tạo ra một hình ảnh. Độ phân giải không gian càng lớn, chất lượng hình ảnh càng cao, số pixel sử dụng càng nhiều hơn. [1]

- *Độ phân giải cường độ*: Một cách đơn giản, độ phân giải cường độ được hiểu là sự thay đổi nhỏ nhất có thể nhận thấy ở mức cường độ. Càng sử dụng nhiều mức cường độ, càng thu được hình ảnh mịn hơn. [1]



Hình 1.2: Ảnh hưởng của độ phân giải không gian lên độ phân giải của ảnh.



Hình 1.3: Ảnh hưởng của độ phân giải cường độ lên độ phân giải ảnh

## 1.2 Các kỹ thuật trích xuất ảnh:

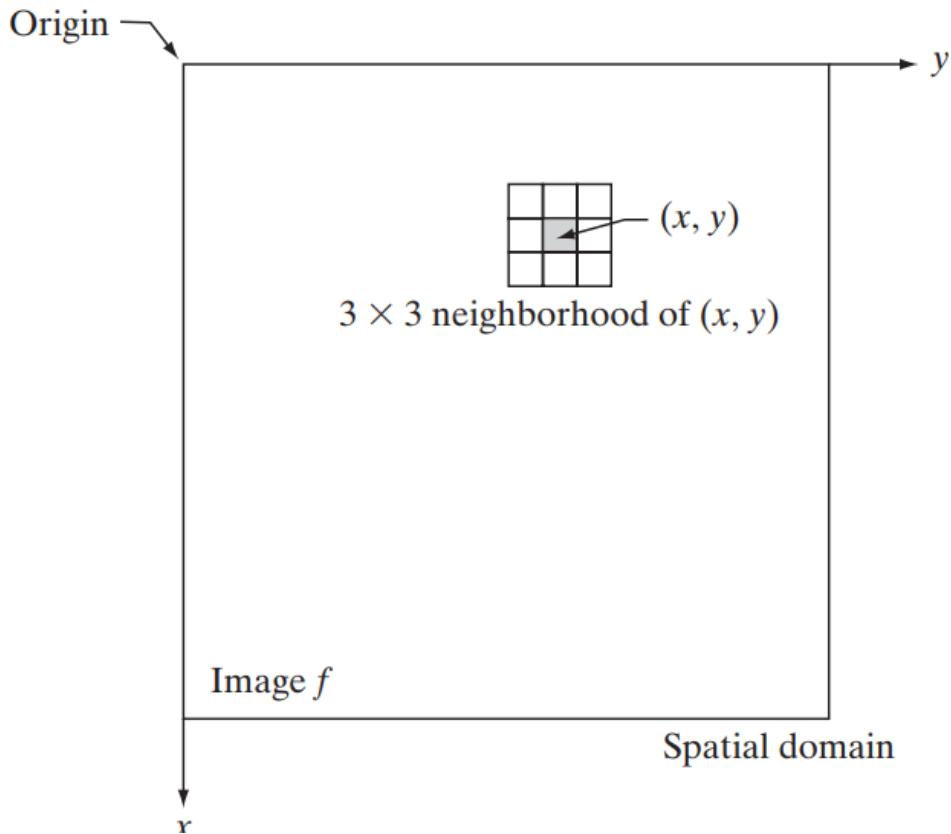
### 1.2.1 Biến đổi cường độ và lọc không gian (Intensity Transformations và Spatial Filtering):

**Một vài vấn đề cơ bản:**

Các quy trình trong miền không gian về kỹ thuật trong phần này được biểu diễn bằng biểu thức:

$$g(x,y) = T[f(x,y)] \quad (1.1)$$

Trong đó,  $f(x, y)$  là hình ảnh cần xử lý,  $g(x, y)$  là hình ảnh sau khi được xử lý,  $T$  là toán tử trên  $f(x, y)$  được xác định trên một vùng lân cận của điểm  $(x, y)$ . Toán tử này có thể tác động vào một ảnh hay một tập hợp các hình ảnh, ví dụ như là việc tổng hợp từng pixel của một chuỗi ảnh để giảm nhiễu.



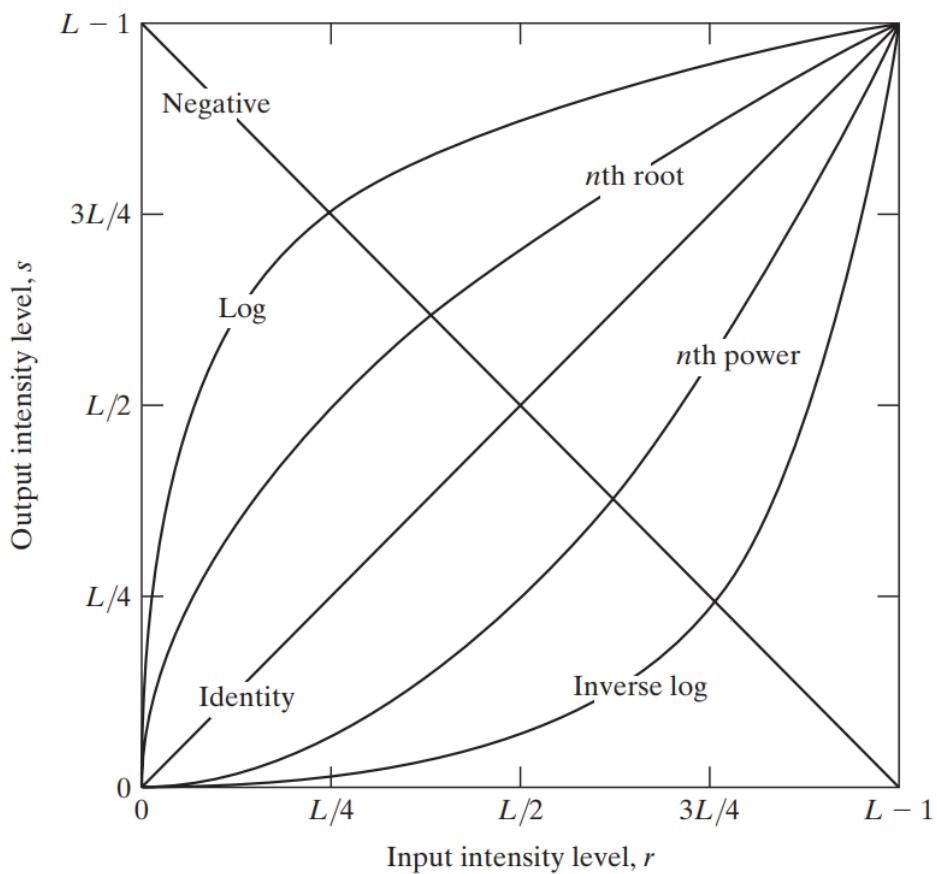
Hình 1.4: Một vùng lân cận  $3 \times 3$  của 1 điểm  $(x, y)$  trong một hình ảnh trong miền không gian..

Như hình vẽ trên, tại 1 điểm  $(x, y)$  bất kỳ của 1 hình ảnh trong miền không gian, các điểm xung quanh điểm  $(x, y)$  được gọi là lân cận của điểm  $(x, y)$  (ở đây là lân cận  $3 \times 3$ ). Lân cận thường được chọn là hình chữ nhật, có kích thước nhỏ hơn nhiều so với toàn thể ảnh. Kết quả đầu ra của điểm  $(x, y)$  là sự áp dụng toán tử  $T$  vào cho các điểm lân cận điểm  $(x, y)$  theo hàm  $f$ . Quá trình này sẽ tiếp tục lặp lại với sự thay đổi điểm  $(x, y)$ . Thông thường, ảnh bắt đầu được xử lý từ phía bên trái của ảnh, tiếp tục từng pixel một theo chiều ngang, từng dọc một. Với các điểm ở góc, biên các pixel sẽ được bỏ qua hoặc đặt là 0 hay một giá trị xác định nào đó, độ dày biên lót phụ thuộc vào kích thước vùng lân cận.

Quá trình mới được miêu tả ở trên được gọi là bộ lọc không gian, trong 1 khu vực hoạt động và toán tử được xác định từ trước. Ngoài ra người ta còn gọi quá trình này spatial mask, kernel, template, window. Lân cận nhỏ nhất có thể là 1x1. Trong trường hợp này, đầu ra hoàn toàn chỉ phụ thuộc vào đầu vào, T trở thành hàm biến đổi cường độ. [1]

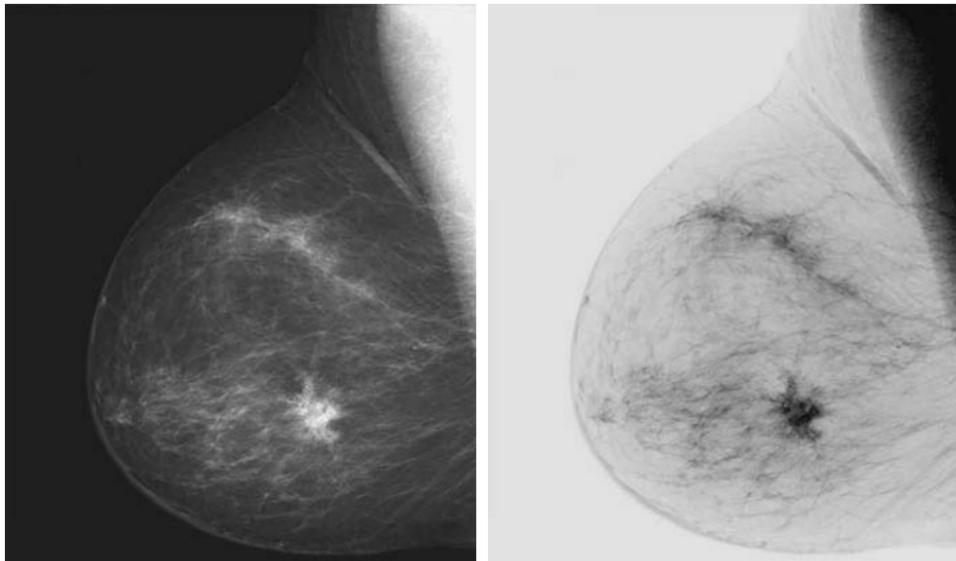
### Một vài vấn đề cơ bản về hàm biến đổi cường độ:

Biến đổi cường độ là một trong những kỹ thuật xử lý ảnh cơ bản nhất. Giá trị của các pixel trước và sau khi biến đổi được ký hiệu lần lượt là r và s. Hàm biến đổi có dạng:  $s = T(r)$ , trong đó T là phép biến đổi ánh xạ từ pixel giá trị r sang pixel giá trị s. Vì chúng ta đang giải quyết một ảnh được lượng tử và số hóa nên giá trị của hàm 1 biến thường được lưu trữ vào 1 mảng 1 chiều và được ánh xạ từ r sang s qua việc tra cứu bảng. Có 3 loại hàm cơ bản được sử dụng để cải thiện hình ảnh: tuyến tính (biến đổi âm và biến đổi đồng nhất), hàm logarithmic (biến đổi log và biến đổi nghịch đảo log) và hàm power - law (biến đổi lũy thừa bậc n và biến đổi căn bậc n).

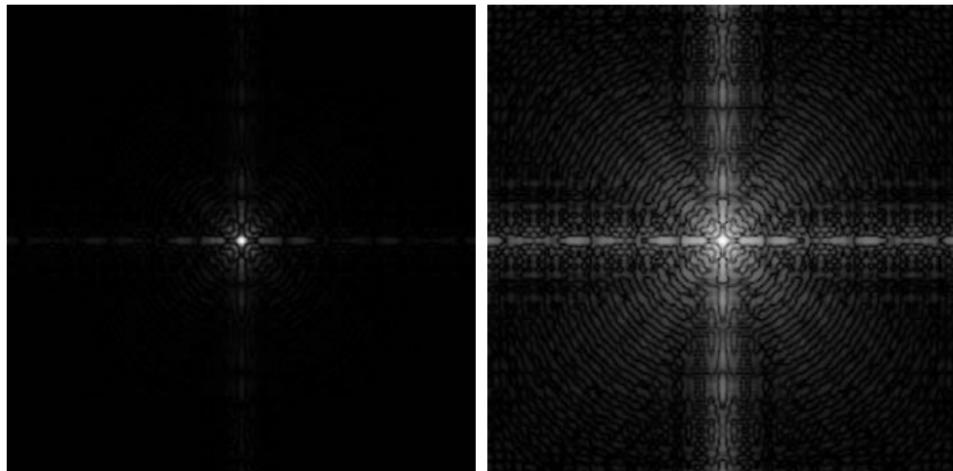


Hình 1.5: Đường biểu diễn của một số hàm thay đổi cường độ.

- *Biến đổi âm bản*: Một bản âm của hình ảnh có cường độ trong khoảng  $[0, L - 1]$  thu được bằng cách sử dụng biến đổi âm có biểu thức là: “ $s = L - 1 - r$ ”. Đảo ngược cường độ tạo ra một ảnh âm bản của ảnh được chụp. Kỹ thuật này thường được sử dụng để làm nổi bật vật thể xám hay trắng trong vùng tối của ảnh, đặc biệt là khi các vùng đen chiếm phần lớn về kích thước. [1]



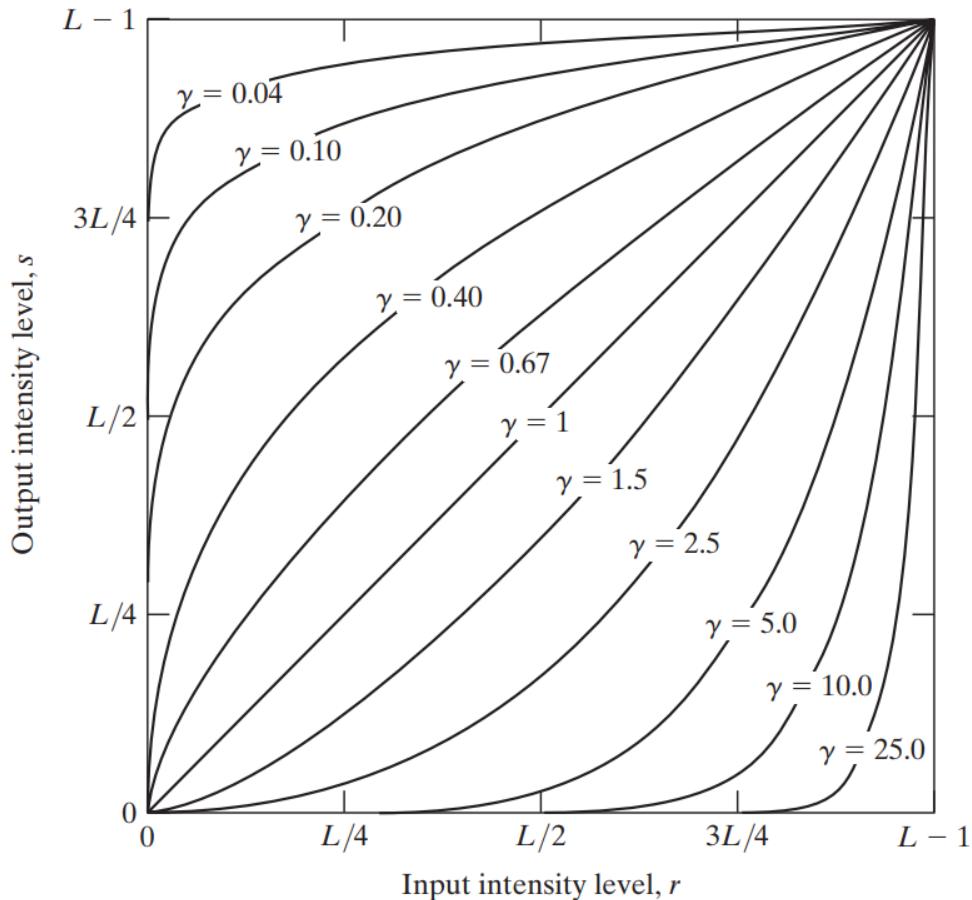
Hình 1.6: Ảnh X-rays tuyến vú và ảnh âm thu được sau khi được biến đổi.



Hình 1.7: Phổ Fourier và kết quả của việc áp dụng phép biến đổi logarithmic với  $c = 1$ .

- *Biến đổi logarithmic*: Biểu diễn chung của biến đổi logarithmic là  $s = c * \log(1 + r)$ , trong đó  $c$  là một hằng số và giả thuyết rằng  $r \geq 0$ . Hình dạng đường cong log

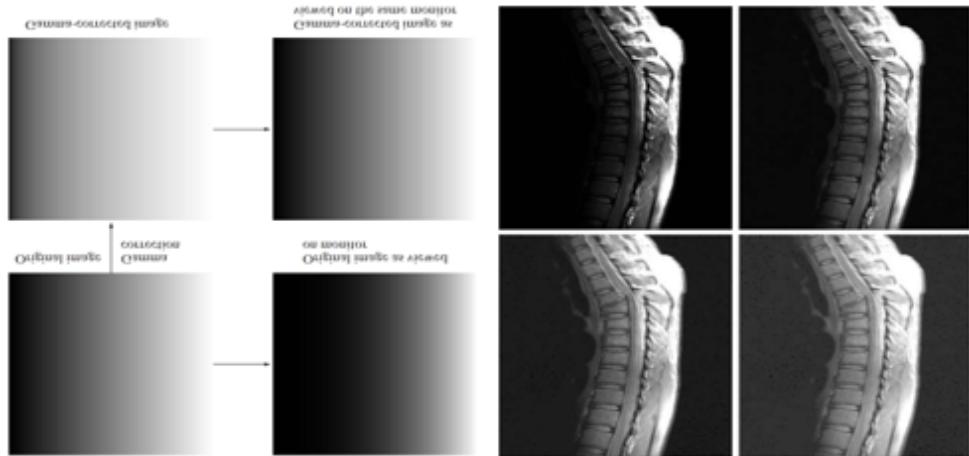
trong hình 5 bên trên cho thấy rằng biến đổi này ánh xạ một phạm vi hẹp của giá trị cường độ thấp đầu vào thành một phạm vi rộng hơn ở đầu ra, ngược lại với đầu vào có giá trị cường độ cao. Chúng ta sử dụng một biến đổi của loại này để mở rộng các giá trị của các pixel tối trong hình ảnh khi nén các giá trị cao hơn. Điều ngược lại là đúng với biến đổi nghịch đảo log. [1]



Hình 1.8: Biểu đồ của phương trình với các giá trị  $\gamma$  khác nhau.

- *Biến đổi power – law:* Biến đổi power – law có biểu diễn cơ bản dưới dạng:  $s = c * r^\gamma$  trong đó:  $c$  và  $\gamma$  là các hằng số dương. Trong một số trường hợp, biểu thức trên có thể được biểu diễn dưới dạng:  $s = c * (r + \varepsilon)^\gamma$  để tính đến một độ lệch (có nghĩa là một đầu ra có thể đo được khi đầu vào là không). Giống như trường hợp biến đổi log, các đường cong power – law với giá trị  $\gamma \leq 1$ , ánh xạ một phạm vi hẹp của giá trị đầu vào tối thành một phạm vi rộng hơn của các giá trị đầu ra, với đối lập là đúng cho các giá trị đầu vào cấp cao. Tuy nhiên, khác với hàm log, chúng ta nhận thấy ở đây một họ các đường cong biến đổi có thể thu được bằng cách thay

đổi  $\gamma$ . Quy trình được sử dụng để sửa chữa những hiện tượng phản ứng power – law này được gọi là hiệu chỉnh gamma. Hiệu chỉnh Gamma rất quan trọng nếu cần chú trọng vào việc hiển thị một cách chính xác trên các màn hình máy tính. Các biến đổi power – law cũng hữu ích cho việc điều chỉnh đối sáng mục đích chung. [1]



Hình 1.9: Một ví dụ về biến đổi power – law và hiệu chỉnh Gamma.

### 1.2.2 *Đổi chiều histogram:*

Histogram là cơ sở cho nhiều kỹ thuật xử lý trong miền không gian. Việc xử lý histogram có thể được sử dụng để cải thiện ảnh, nén ảnh hay phân đoạn ảnh. Histogram dễ tính toán trong phần mềm, thích hợp cho việc triển khai phần cứng một cách hiệu quả, làm cho chúng trở thành một công cụ phổ biến cho việc xử lý hình ảnh trong thời gian thực.

Histogram của một ảnh số có mức cường độ trong khoảng  $[0, L - 1]$  là một hàm rời rạc  $h(r_k) = n_k$  trong đó  $r_k$  là giá trị cường độ thứ k còn  $n_k$  là số các pixel có giá trị cường độ là  $r_k$ . Một cách thông dụng để chuẩn hóa histogram là chia từng phần của nó cho tổng số pixel trong hình ảnh, được biểu hiện bởi tích  $MN$ , trong đó,  $M$  và  $N$  lần lượt là chiều rộng và chiều cao của ảnh.

Do đó, histogram được chuẩn hóa bởi  $p(r_k) = r_k/MN$ , với  $k$  thay đổi từ 0 đến  $L - 1$ . Một cách không hoàn toàn chính xác, là một ước lượng về xác suất của cường độ một hình ảnh. Tổng các thành phần của 1 histogram đã được chuẩn hóa là 1.

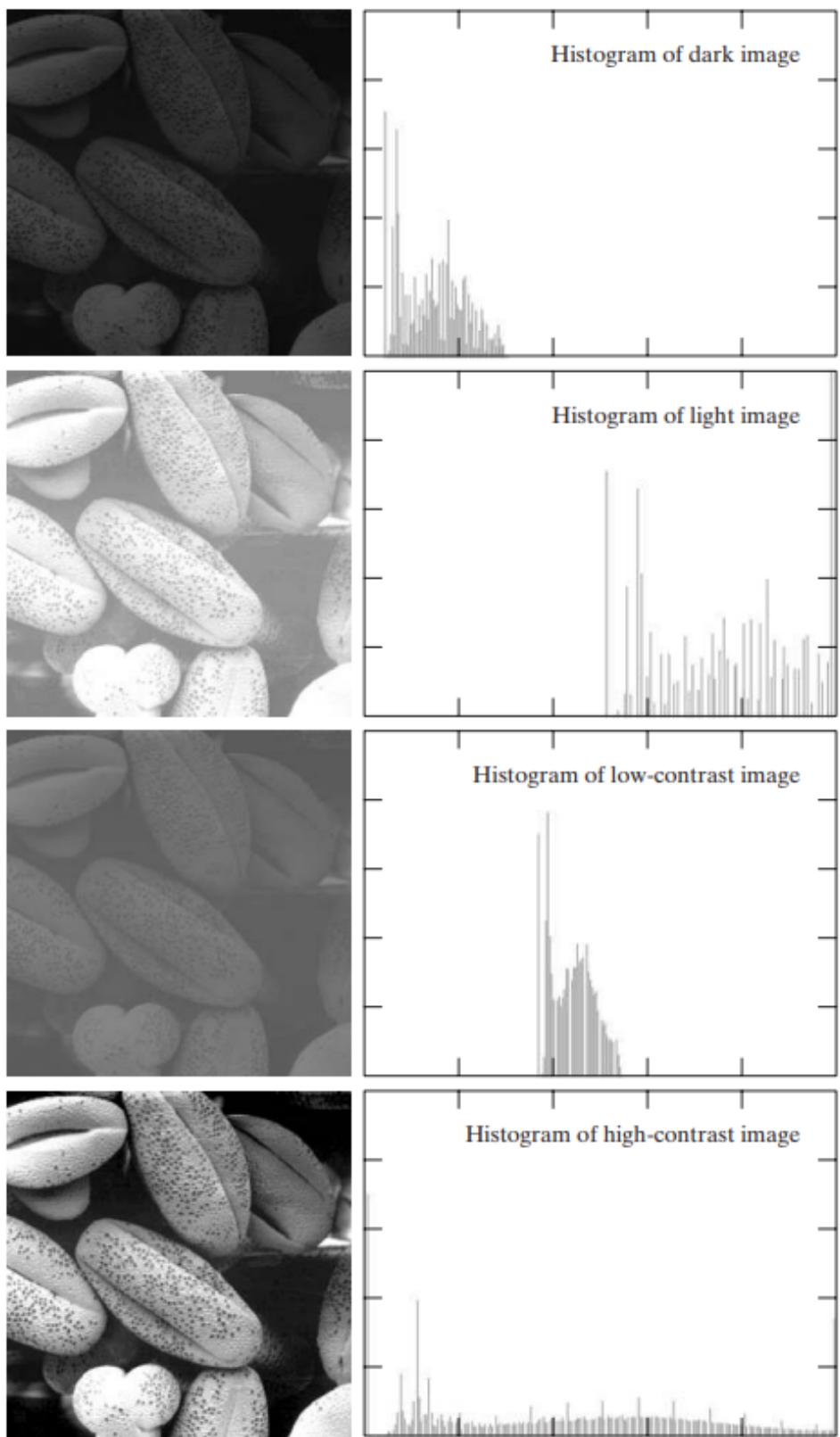
Với các hình ảnh tối, các thành phần của histogram được tập trung về phía trái của cường độ. Ngược lại với các hình ảnh sáng, các thành phần histogram tập trung về phía bên phải của cường độ.

Một hình ảnh có độ tương phản thấp thì histogram thường hẹp và nằm ở gần giữa thang đo độ sáng. Với một hình ảnh có độ tương phản cao, histogram bao gồm một vùng rộng lớn của thang đo độ sáng, phân phối các điểm ảnh không quá độc lập, thường có rất ít những đường dọc cao hơn so với mức trung bình.

Có thể thấy là một hình ảnh mà các điểm ảnh có xu hướng bao phủ rộng thang đo độ sáng và có xu hướng phân phối đều, sẽ có là hình ảnh có độ tương phản cao và sẽ thể hiện một loạt các tông màu xám. Kết quả chung sẽ là một hình ảnh cho thấy nhiều chi tiết về cấp độ xám và có phạm vi động lớn.

Trong xử lý hình ảnh, ta thường có nhu cầu điều chỉnh lại độ tương phản của ảnh. Trong các trường hợp như vậy, chúng ta sử dụng một kỹ thuật biến đổi cường độ được biết đến là cân bằng histogram. Cân bằng histogram là quá trình phân phối đồng đều histogram của ảnh trên toàn trực độ sáng bằng cách chọn một hàm biến đổi cường độ phù hợp.

Xét các cấp độ ánh sáng của một hình ảnh  $r$  là liên tục. Giá trị của  $r$  sẽ nằm trong khoảng  $[0, L - 1]$ . Xét một hàm biến đổi:  $s = T(r)$ , trong đó  $s$  là kết quả hình ảnh thu được,  $T(r)$  có một số ràng buộc.  $T(r)$  là một hàm tăng liên tục và  $0 \leq T(r) \leq L - 1$  làm cho hàm  $T(r)$  trở thành một hàm song ánh. Gọi hàm mật độ xác suất (pdf) của  $r$  là  $p_r(x)$  và hàm phân phối tích lũy (CDF) của  $r$  là  $F_r(x)$ . Lúc này CDF của  $s$  là:  $F_s(x) = P(s \leq x) = P(T(r) \leq x) = P(r \leq T^{-1}(x)) = F_r(T^{-1}(x))$ . Ta có pdf của  $s$  thu được bằng cách đạo hàm  $F_s(x)$  theo  $x$ :  $p_s(s) = p_r(r) \frac{dr}{ds}$ . Nếu định nghĩa hàm biến đổi:  $s = T(s) = (L - 1) \int_0^s p_r(x) dx$ . Sử dụng tích phân Leibnitz, sử dụng đạo hàm trên, ta có:  $p_r(r) \frac{dr}{ds} = p_r \frac{1}{(L-1)p_r(r)} = \frac{1}{L-1}$ . Ta thu được hàm xác suất của  $s$  là đồng đều. Với trường hợp, trường là rời rạc, ta thay dấu tích phân bằng dấu tổng, được biểu diễn  $s_k = T(r_k) = (L - 1) \sum_{j=0}^k p_r(r_j) = \frac{(L-1)}{N^2} \sum_{j=0}^k n_j$ . Do  $s$  là số nguyên, nếu thu được từ hàm một giá trị không nguyên đều phải làm tròn đến số nguyên gần nhất. [1]

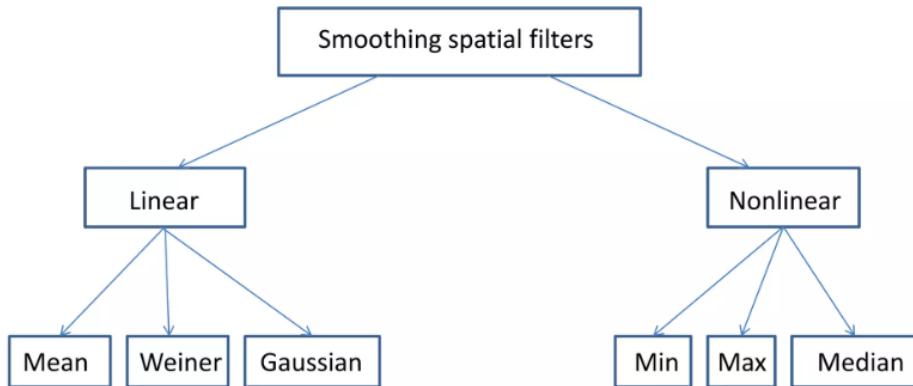


Hình 1.10: 4 kiểu ảnh cơ bản (tối, sáng, độ tương phản thấp, độ tương phản cao) và histogram

### 1.2.3 *Làm mờ ảnh, làm sắc nét:*

Làm mờ ảnh thường được sử dụng để làm giảm nhiễu trong ảnh. Đây là kỹ thuật quan trọng trong việc cải thiện hình ảnh với việc giảm nhiễu của ảnh. Do đó, nó là một hàm quan trọng trong quá trình xử lý ảnh của nhiều phần mềm xử lý ảnh. Làm mờ ảnh là một kỹ thuật cải thiện chất lượng ảnh. Làm mờ ảnh được biểu diễn bằng bộ lọc không gian và bộ lọc tần suất.

Bộ lọc không gian là một bộ lọc hoạt động trực tiếp trên từng pixel của ảnh. Chu trình đơn giản là chuyển bộ lọc từ điểm này tới điểm khác trong một hình ảnh. Bộ lọc làm mờ được dùng để giảm nhiễu và phép làm mờ. Thông qua việc xem xét các pixel xung quanh để đưa ra một phiên bản chính xác của các pixel này. Bằng việc xét các pixel lân cận, những pixel nhiễu đặc biệt nghiêm trọng có thể bị lọc ra. Tuy nhiên, pixel nhiễu có thể biểu diễn những chi tiết tinh tế ban đầu, những nó cũng sẽ bị mất trong quá trình làm mờ.



Hình 1.11: Bộ lọc làm mờ theo không gian.

Bộ lọc làm mờ tuyến tính là trung bình các pixel chứa ở khu vực lân cận của bộ lọc có thể là bộ lọc trung bình hoặc bộ lọc thông thấp: lọc trung tâm và lọc Gaussian. Lọc trung tâm đơn giản là thay thế giá trị của pixel bằng trung bình các giá trị của lân cận xung quanh bao gồm chính nó. Thông thường một bộ lọc ma trận vuông 3x3 được sử dụng, dù những bộ lọc lớn hơn có thể được sử dụng để làm mịn mạnh hơn. Tuy nhiên, việc làm mờ này sẽ làm mất các chi tiết ảnh tinh tế. Bộ lọc Gaussian làm mờ bằng cách tính trung bình có trọng số trọng bộ lọc. Nó được sử dụng làm mờ ảnh và loại bỏ các chi tiết và nhiễu. Bằng việc đặt trọng số lớn cho pixel trung tâm và giảm dần các trọng

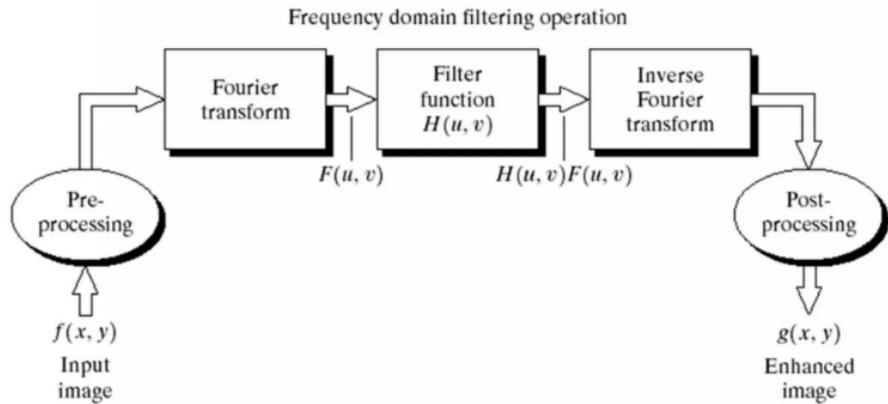
số cho các lân cận. Các xa pixel trung tâm thì trọng số càng nhỏ. Làm mờ Gaussian cho ra một hiệu quả làm mờ hoàn toàn mà không gây ra hiệu ứng phụ, biểu thức của bộ lọc Gaussian:  $\frac{1}{2\pi\sigma^2} \exp -\frac{x^2+y^2}{2\sigma^2}$

Với một bộ lọc không gian phi tuyến tính là các bộ lọc thống kê có thứ tự, phản ứng của chúng dựa trên việc sắp xếp các pixel trong khu vực hình ảnh được bao quanh bởi bộ lọc và sau đó thay thế giá trị của pixel trung tâm bằng giá trị được xác định thông qua kết quả sắp xếp. Có 3 loại bộ lọc phi tuyến tính là bộ lọc lớn nhất nhỏ nhất, bộ lọc trung tuyến, bộ lọc điểm chính giữa.

- *Bộ lọc lớn nhất nhỏ nhất:* Bộ lọc nhỏ nhất lựa chọn giá trị nhỏ nhất và bộ lọc lớn nhất lựa chọn giá trị lớn nhất. Bộ lọc lớn thường được dùng để tìm điểm sáng nhất. Bộ lọc nhỏ thường được dùng để tìm điểm tối nhất. Cả hai bộ lọc đều yêu cầu sắp xếp dữ liệu. [1]
- *Bộ lọc trung tuyến:* Nó làm mờ các pixel có giá trị khác biệt đáng kể với các pixel xung quanh mà không làm ảnh hưởng các pixel khác. Bộ lọc này rất phù hợp nhiều “salt and pepper” (là một loại nhiễu nơi mà một số pixel được thay đổi thành giá trị cực lớn (trắng) hoặc cực bé (tối) tạo nên các điểm sáng tối ngẫu nhiên. [1]
- *Bộ lọc điểm chính giữa:* Bộ lọc điểm chính giữa làm mờ ảnh bằng việc thay đổi pixel bằng trung bình của pixel giá trị cao nhất và pixel giá trị pixel nhỏ nhất, điểm chính giữa: ( $\text{sáng nhất} + \text{tối nhất}$ ) / 2. [1]

Bộ lọc tần số là mô hình cơ bản để lọc trong miền tần số trong đó  $F(u, v)$  là hàm biến đổi Fourier của hình ảnh cần được làm mờ,  $H(u, v)$  là hàm chuyển đổi của bộ lọc. Làm mờ cơ bản là một hoạt động chuyển động tần số thấp trong tần số. Nó sẽ tính toán nhanh hơn so với miền không gian.

Về cơ bản, bộ lọc được phân loại theo các tính chất trong miền tần số: bộ lọc thông thấp để làm mờ, bộ lọc thông cao để làm sắc nét, bộ lọc truyền qua và bộ lọc dừng. Có nhiều loại bộ lọc thông thấp: bộ lọc thông thấp lý tưởng, bộ lọc thông thấp Butterworth, bộ lọc thông thấp Gaussian.



Hình 1.12: Hoạt động của bộ lọc tần số.

- **Bộ lọc thông thấp lý tưởng:** Bộ lọc thông thấp đơn giản nhất là bộ lọc loại tất cả các thành phần tần số cao của biến đổi Fourier mà có khoảng cách  $D_0$  lớn hơn một giá trị xác định từ gốc của biến đổi. Hàm chuyển của một bộ lọc thông thấp lý tưởng:  $H(u, v) = 1$  nếu  $D(u, v) \leq D_0$  và  $= 0$  nếu  $D(u, v) \geq D_0$  trong đó  $D(u, v)$  là khoảng cách từ điểm  $(u, v)$  đến điểm giữa trung tâm của hình chữ nhật tần số  $D(u, v) = [(u - \frac{M}{2})^2 + (v - \frac{N}{2})^2]^{\frac{1}{2}}$  [1]
- **Bộ lọc thông thấp Butterworth:**  $H(u, v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$  [1]
- **Bộ lọc thông thấp Gaussian:**  $H(u, v) = e^{-\frac{[D(u,v)]^2}{2(D_0)^2}}$  [1]

Bộ làm sắc nét được sử dụng nhằm loại bỏ mờ và làm nổi bật các cạnh. Bộ lọc làm nét cũng có hai loại là bộ lọc làm nét không gian và bộ lọc làm nét theo tần số. Trong đó, bộ lọc làm nét theo không gian dựa trên đạo hàm cấp một và cấp hai. Còn bộ lọc làm nét theo tần số thì cũng có 3 loại tương tự bộ lọc làm mờ. Tuy nhiên, khác với bộ lọc làm mờ sử dụng 1 bộ lọc thông thấp để tăng hiệu quả làm mờ thì với bộ làm sắc nét ta sử dụng một bộ lọc thông cao nhằm giữ lại những nét tinh tế riêng biệt của hình ảnh.[1]

#### 1.2.4 Màu sắc ảnh:

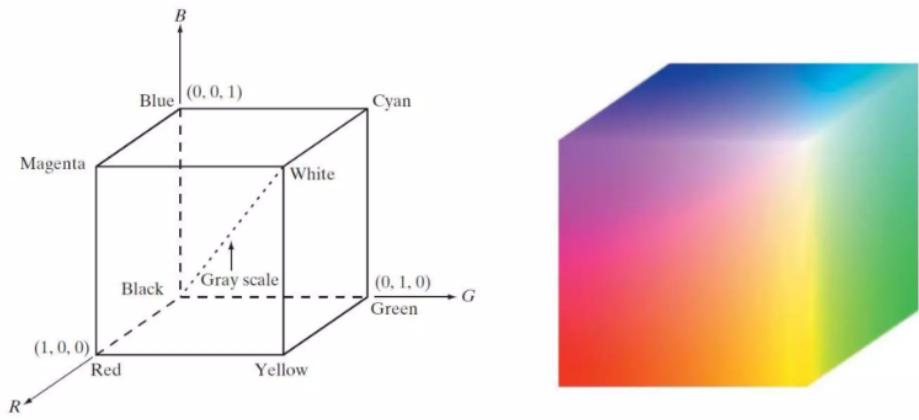
Màu sắc là một công cụ mô tả rất hữu dụng trong việc phân biệt đặc điểm của các vật thể và phân biệt chúng với nền. Quá trình xử lý ảnh gồm 2 hướng chính: tất cả màu và giả màu. Trong tất cả màu, hình ảnh được thu thập bằng một cảm biến đủ màu. Với giả

màu, vấn đề là gán một màu cho một độ sáng màu xám cụ thể hoặc một khoảng độ sáng cụ thể. Các kỹ thuật xử lý màu giờ được sử dụng trong rất nhiều lĩnh vực như là xuất bản, trực quan hóa và trên Internet. Nếu bạn mới tiếp cận với L<sup>A</sup>T<sub>E</sub>X thì bạn nên đọc hết phần thông tin còn lại trong tài liệu này.

## Các mẫu màu:

Mục đích của các mẫu màu (còn gọi là không gian màu hay hệ thống màu) là hỗ trợ mô tả chi tiết về màu sắc theo nhiều tiêu chuẩn khác nhau. Về cơ bản, một mô hình màu là một đặc tả của một hệ tọa độ và một không gian con trong hệ tọa độ đó, trong đó mỗi màu được biểu diễn bằng một điểm duy nhất. Hầu hết các mô hình màu được sử dụng hiện nay được định hình theo hai hướng chính: về phân cứng hoặc về ứng dụng nơi mà xử lý màu là mục tiêu. Đối với xử lý hình ảnh số, các mô hình phổ biến được sử dụng thực tế bao gồm mô hình RGB cho màn hình màu, mô hình CMY và CMYK cho máy in màu, cùng với mô hình HSI phản ánh chặt chẽ cách con người mô tả và hiểu màu sắc.

- *Mẫu RGB*: Trong mẫu RGB, mỗi màu xuất hiện trong các thành phần cơ bản của đỏ, xanh dương, xanh lục. Không gian màu của quan tâm là hình lập phương, trong đó 3 màu cơ bản là 3 trục, 3 màu phụ là xanh nhạt (cyan), đỏ tía (magenta) và vàng ở 3 góc còn lại và là sự tổng hợp màu sắc của 3 màu gốc trong đó: xanh nhạt (cyan) được tổng hợp từ xanh lục và xanh dương, đỏ tía (magenta) được tổng hợp từ đỏ và xanh dương và vàng được tổng hợp từ đỏ và xanh lục. Màu đen nằm ở gốc của 3 trục đỏ, xanh lục, xanh dương; còn màu trắng thì nằm ở của hình lập phương có khoảng cách xa nhất với điểm đem. Khoảng cách giữa điểm trắng và điểm đen chính là thang độ xám. Mỗi màu trong mẫu này là điểm trong trên hoặc trong hình lập phương này. Các giá trị đỏ, xanh lục, xanh dương đều được cho trong khoảng từ [0, 1]. Tạo hình ảnh RGB của một mặt phẳng màu chấn (cross-sectional color plane) thường liên quan đến việc kết hợp các kênh màu để tạo ra một hình ảnh màu tổng hợp. Với các kênh màu riêng lẻ, bạn có thể xếp chúng lại để tạo ra một hình ảnh màu RGB.
- *Mẫu CMY hay CMYK*: là các màu phụ của ánh sáng, phần lớn các thiết bị sử dụng các chất tạo màu lên giấy như máy in đều yêu cầu thông tin đầu vào dưới dạng



Hình 1.13: Sơ đồ của mẫu màu RGB dưới dạng một khối hộp.

CMY hoặc thực hiện 1 bước chuyển từ mẫu RGB sang CMY. Các giá trị của màu cũng được đặt trong khoảng  $[0, 1]$ . Các màu phụ của ánh sáng thuần thu được bằng cách lấy 1 trừ đi giá trị của từng màu chính, trong đó, màu xanh nhạt thuần thu được bằng việc loại bỏ màu đỏ, màu đỏ tía thì là loại bỏ màu xanh lục còn màu vàng là loại bỏ màu xanh dương.

- *Mẫu HSI*: các mẫu RGB hay CMY là những mẫu màu lý tưởng cho các phần cứng. Tuy nhiên, các mẫu này không phù hợp để miêu tả màu mà con người giải thích. Khi nhìn một vật thể màu, chúng ta mô tả nó bằng màu sắc, độ bão hòa, độ sáng tối. Việc nhận biết cường độ là điều quan trọng trong mô tả cảm quan màu sắc của con người. Mẫu HSI tập trung mô tả cường độ mà các màu sắc mang trong hình ảnh. Từ đó, mẫu HSI là công cụ lý tưởng trong phát triển các thuật toán xử lý ảnh dựa trên việc mô tả màu sắc.

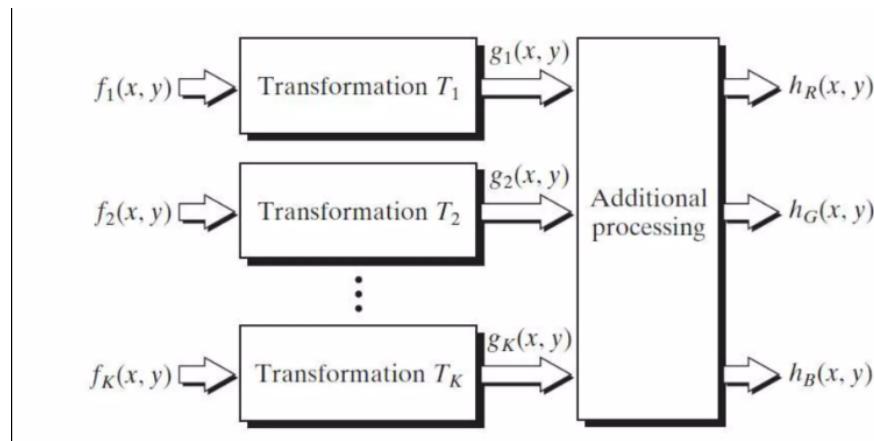
### Xử lý giả màu:

Xử lý màu giả là quá trình chuyển màu thành giá trị xám dựa trên các điểm nhấn đặc biệt. Quy tắc sử dụng của màu giả là khả năng hình ảnh hóa của con người và diễn giải thang đo xám trong hình ảnh.

- *Cắt cường độ*: kỹ thuật cắt cường độ là một trong những kỹ thuật đơn giản nhất trong xử lý màu giả. Nếu một hình ảnh được biểu diễn dưới dạng là 1 hàm 3D, công thức có thể xem như một cách để đặt các mặt phẳng song song với mặt phẳng

tọa độ của một hình ảnh, từ đó thu được hàm tại khu vực giao nhau. Nếu có nhiều màu khác nhau được đặt lên mỗi phần của mặt phẳng cắt thì các giá trị cường độ lớn hơn của mặt phẳng cắt đều được đặt về 1 màu, tương tự với các giá trị thấp hơn. Giá trị cường độ của mặt cắt được đặt là một trong hai màu đã xác định ở trên. Kết quả thu được là một ảnh gồm 2 màu có sự xuất hiện được kiểm soát bởi sự di chuyển lén xuống của mặt phẳng cắt trên trục cường độ. Cơ bản, kỹ thuật này được tóm tắt như sau: Giả sử thang đo độ xám có giá trị từ  $[0, L - 1]$ ;  $l_0$  biểu diễn mặt phẳng màu đen [ $l(x, y) = 0$ ] và  $l_{L-1}$  biểu diễn mặt phẳng trắng [ $l(x, y) = L - 1$ ]. Giả sử có một mặt phẳng P vuông góc với trục cường độ được định nghĩa tại các mức  $l_1, l_2, \dots, l_p$ . Sau đó, giả sử rằng  $0 < P < L - 1$ , mặt phẳng P chia bảng màu xám thành  $P + 1$  khoảng  $V_1, V_2, \dots, V_{p+1}$ . Cường độ màu được điền vào có mối quan hệ:  $f(x, y) = c_k$  nếu  $f(x, y) \in V_k$  khi  $c_k$  là màu sắc thuộc cường độ  $k^{th}V_k$  được định nghĩa bởi mặt phẳng chia giữa  $l = k - 1$  và  $l = k$ . [1]

- *Chuyển đổi từ cường độ thành màu sắc:* các phép biến đổi có khả năng đạt được một loạt rộng các kết quả tăng cường giả màu so với kỹ thuật chia đơn giản. Ý tưởng ẩn sau cách tiếp cận này là thực hiện độc lập 3 biến đổi về cường độ của bất kỳ pixel đầu vào nào. Các kết quả thu được chia về 3 kênh màu cơ bản. Phương pháp này tạo ra một hình ảnh tổng hợp mà nội dung màu sắc của nó được kiểm soát bởi tính chất của hàm biến đổi. [1]



Hình 1.14: Phương pháp mã màu giả khi có nhiều hình ảnh đơn sắc  
khả dụng.

### 1.2.5 Phép biến đổi hình thái học:

Phép biến đổi hình thái học là các biến đổi dựa trên hình dạng ảnh. Nó thường được biểu diễn trong ảnh nhị phân. Đầu vào sẽ bao gồm: ảnh gốc và kernel. Hai phép biến đổi hình thái học cơ bản là co rút (Erosion) và giãn nở (Dilation). Đồng thời cũng có nhiều dạng như dạng mở (Opening), đóng (Closing), đạo hàm (Gradient), ...

#### Co rút:

Với A và B là tập hợp trong  $Z^2$ , phép co rút của A bởi B được ký hiệu là  $A \ominus B$ , được định nghĩa là:  $A \ominus B = z|(B)_z \subseteq A$ . Phương trình  $A \ominus B$  biểu thị rằng sự co rút của tập hợp A bởi phần tử kết cấu B là tập hợp của tất cả các điểm nơi dịch chuyển của B nằm hoàn toàn trong A. Nói một cách đơn giản, đó là tập hợp các điểm sao cho khi mỗi điểm trong B được đặt lên trên A, tất cả các phần của B đều nằm trong A. Trong các cuộc thảo luận, B thường được gọi là phần tử kết cấu. Phương trình  $A \ominus B = z|(B)_z \cap A^c = \emptyset$  đại diện cho mô tả toán học của khái niệm này. Công thức này tương đương với tuyên bố rằng B không được chia sẻ bất kỳ phần tử chung nào với phần còn lại của A. Phép co rút làm thu gọn hay mảnh hơn các vật thể trong ảnh nhị phân. Trên thực tế, ta xem co rút là một phép lọc biến đổi hình thái làm các chi tiết ảnh mảnh hơn so với kernel lọc khỏi hình ảnh. [1]

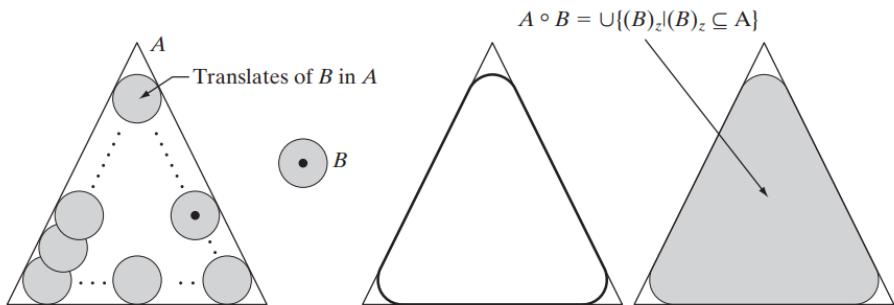
#### Giãn nở:

Với A và B là các tập hợp trong  $Z^2$ , phép giãn nở A bởi B được ký hiệu là  $A \oplus B$  và được định nghĩa là:  $A \oplus B = z|((B)_z \cap A \neq \emptyset$ . Phương trình này dựa trên việc phản ánh B về gốc tọa độ của nó và dịch chuyển phản xạ này theo z (xem Hình 9.1). Sự giãn nở của khía đó là tập hợp tất cả các chuyển vị, sao cho và chồng lên nhau bởi ít nhất một phần tử. Dựa trên cách giải thích này, phương trình (9.2-3) có thể viết tương đương là:  $A \oplus B = z|[(B)^z \cap A] \subseteq A$ . Ngược lại với co rút, giãn nở làm dày hơn hay to hơn các vật thể trong ảnh nhị phân. Phương thức và mức độ cụ thể về độ dày được kiểm soát bởi kernel lọc. [1]

## Mở (Opening) và Đóng (Closing):

Như đã tìm hiểu ở trên, giãn nở (dilation) làm phồng hay phóng to các thành phần của ảnh, trong khi đó co rút (erosion) làm mảnh chung. Tiếp theo chúng ta sẽ nói đến hai phép toán hình thái quan trọng khác là: mở (opening) và đóng (closing). Phép mở thường làm phẳng đường viền của vật thể, phá vỡ các eo hẹp và loại bỏ các phần nhô ra mỏng. Phép đóng cũng làm phẳng các phần của đường viền, nhưng, trái ngược với phép mở, nó thường hợp nhất các đoạn hẹp và các vịnh mỏng dài, loại bỏ các lỗ nhỏ và lấp đầy các khoảng trống trên đường viền.

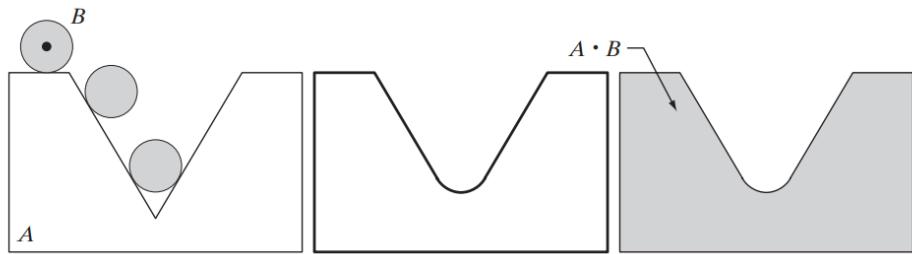
Phép mở của tập hợp A bởi kernel B, được kí hiệu là  $A \circ B$  và được định nghĩa là:  $A \circ B = (A \ominus B) \oplus B$ . Có thể thấy, phép mở A bởi kernel B về cơ bản là co rút A theo B, sau đó lại giãn nở kết quả thu được theo B. Phép mở có thể diễn giải hình học như hình 15. Ta coi kernel B như một “quả bóng lăn phẳng”. Ranh giới của A với B được tạo thành bởi các điểm B tiến xa nhất vào ranh giới A. Tính chất hình học này dẫn đến một công thức lý thuyết tập hợp, phát biểu rằng việc mở A theo B có được bằng cách lấy hợp tất cả B phù hợp với A. Ta thu được biểu thức:  $A \circ B = (B)_z | (B)_z \subseteq A$  trong đó U. biểu thị sự kết hợp tất cả các bộ bên trong dấu ngoặc nhọn. [1]



Hình 1.15: Hình ảnh mô tả phép mở.

Tương tự, ta thu được phép đóng tập hợp A bởi kernel B được kí hiệu là  $A \cdot B$  và được định nghĩa là:  $A \cdot B = (A \oplus B) \ominus B$ . Ta có thể hiểu biểu thức là phép đóng A bởi kernel B đơn giản là giãn nở A bởi kernel B rồi lấy kết quả thu được đem co rút lại bởi kernel B. Phép đóng cũng có các mô tả hình học tương tự, trừ việc ta sẽ lăn B bên ngoài đường bao. Như đã phân tích, ta thấy phép mở và phép đóng là đối ngẫu của nhau, nên việc ta

lăn B bên ngoài đường bao là được dự đoán trước. Một điểm w là phần tử của  $A \cdot B$  khi và chỉ khi  $(B)_z \cap A \neq \emptyset$  với bất kỳ  $(B)_z$  nào chứa w. [1]



Hình 1.16: Hình ảnh mô tả phép đóng.

Như co rút và giãn nở, phép mở và phép đóng là đối ngẫu của nhau về mặt bù và phản xạ và được biểu diễn là:  $(A \cdot B)^c = (A^c \circ B) \vee (A \circ B)^c = (A^c \cdot B)$ . Đồng thời phép mở và đóng cũng thỏa mãn một số tính chất sau:

Phép mở	Phép đóng
1 $A \circ B$ là tập con (hình ảnh phụ) của A	$A$ là tập con (hình ảnh phụ) của $A \cdot B$
2 Nếu C là tập con của D thì $C \circ B$ là tập con của $D \circ B$	Nếu C là tập con của D thì $C \cdot B$ là tập con của $D \cdot B$
3 $(A \circ B) \circ B = A \circ B$	$(A \cdot B) \cdot B = A \cdot B$

### Lý thuyết trúng hoặc trượt:

Phép biến đổi hình thái trúng hoặc trượt là một công cụ cơ bản để nhận diện vật thể. Cụ thể là nó tìm những pixel có vùng lân cận khớp với hình dạng của phần tử cấu trúc thứ nhất  $B_1$  trong khi không khớp với hình dạng của phần tử cấu trúc thứ hai  $B_2$ :  $A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2)$ . Vì vậy, phép biến đổi trúng hoặc trượt gồm 3 bước: Co rút hình ảnh A với kernel B1. Sau đó, co rút phần bù của hình ảnh A với kernel B2. Sau đó lấy giao của bước 1 và bước 2. Bất cứ bước nào trong 3 bước trên đều được gọi là phép biến đổi trúng hoặc trượt. Lý do sử dụng kernel với các đối tượng và phần tử liên kết với nền là dựa trên định nghĩa giả định rằng hai hoặc nhiều đối tượng chỉ khác biệt nếu chúng tạo thành các tập hợp rời rạc (ngắt kết nối). Điều này được đảm bảo bằng cách yêu cầu mỗi đối tượng có nền dày ít nhất một pixel xung quanh nó. Trong một số ứng dụng, chúng ta có thể quan tâm đến việc phát hiện các mảng (kết hợp) nhất định của các số 1 và 0 trong một tập hợp, trong trường hợp đó không cần có nền. Trong những trường hợp

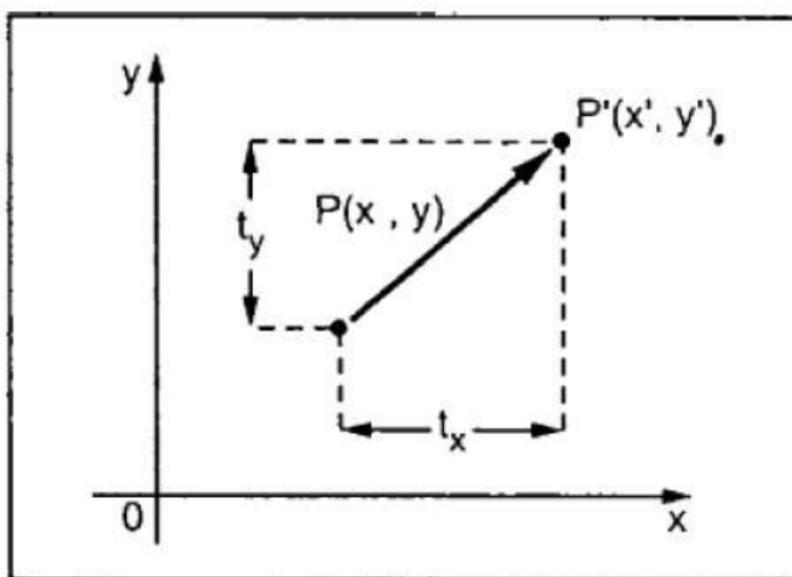
như vậy, phép biến đổi trúng hoặc trượt sẽ giảm xuống mức co rút đơn giản. Như đã chỉ ra trước đây, co rút vẫn là một tập hợp các kết quả trùng khớp, nhưng không có yêu cầu bổ sung về kết quả khớp nền để phát hiện các đối tượng riêng lẻ. Sơ đồ phát hiện mẫu đơn giản hóa này được sử dụng trong một số thuật toán được phát triển trong phần sau.

### 1.2.6 Phép biến đổi hình học:

Trong thực tế, để việc xử lý ảnh được dễ dàng hơn thì người ta có thể sử dụng các phép biến đổi hình học như lật lại hình, hay quay hình để có thể xử lý những vật thể, hình ảnh một cách chính xác hơn.

#### Phép dịch:

Một phép dịch là một phép biến đổi dịch chuyển vật thể từ vị trí này tới vị trí khác. Ta có thể chuyển một điểm trong 2D bằng cách thêm vào một phần tử dịch  $(t_x, t_y)$  vào các thành phần ban đầu X, Y tạo thành các thành phần mới X', Y'.

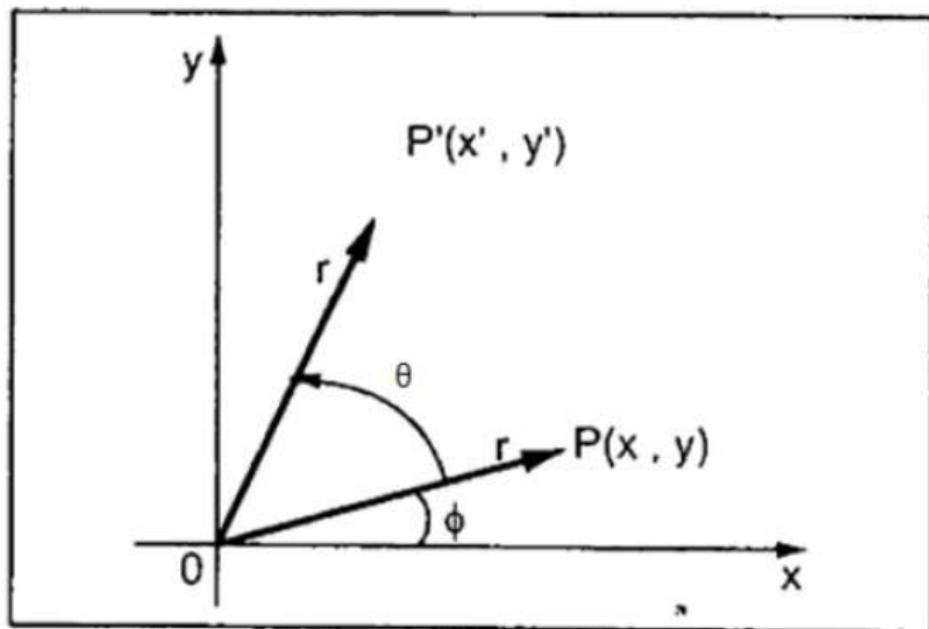


Hình 1.17: Hình ảnh mô tả phép dịch chuyển.

Từ hình trên có thể thấy,  $X' = X + t_x$ ;  $Y' = Y + t_y$ . Cặp  $(t_x, t_y)$  được gọi là vectơ dịch chuyển. Công thức trên có thể được viết lại dưới dạng vectơ cột:  $P = \begin{bmatrix} X \\ Y \end{bmatrix}$ ;  $p' = \begin{bmatrix} X' \\ Y' \end{bmatrix}$ ;  $T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$ . Ta cũng có thể viết dưới dạng:  $P' = P + T.[1]$

### Phép quay:

Với phép quay, chúng ta có thể quay một vật thể một góc  $\theta$  khỏi vị trí ban đầu của nó. Như hình 17, ta có thể thấy điểm  $(X, Y)$  được đặt ở góc  $\varphi$  so với trục X với khoảng cách  $r$  so với góc tọa độ. Giả sử nếu ta muốn quay điểm P một góc  $\theta$ . Sau khi quay điểm P tới một vị trí mới, ta thu được điểm  $(X', Y')$ .



Hình 1.18: Hình ảnh mô tả phép quay.

Sử dụng lượng giác tiêu chuẩn, tọa độ ban đầu của điểm P được biểu diễn là:

$$X = r \cos \varphi \quad \dots(1)$$

$$Y = r \sin \varphi \quad \dots(2)$$

Tương tự, ta có thể biểu diễn điểm P' kiểu:

$$x' = r \cos(\varphi + \theta) = r \cos \varphi \cos \theta - r \sin \varphi \sin \theta \quad \dots(3)$$

$$y' = r \sin(\varphi + \theta) = r \cos \varphi \sin \theta + r \sin \varphi \cos \theta \quad \dots(4)$$

Thay phương trình 1 và 2 tương ứng vào phương trình 3 và 4, ta sẽ thu được:

$$x' = x \cos \theta - y \sin \theta \quad (1.6)$$

$$y' = x \sin \theta + y \cos \theta \quad (1.7)$$

Nếu biểu diễn biểu thức trên dưới ma trận:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} P' = P.R \quad (1.8)$$

Trong khi đó, R là vecto quay:

$$R = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad (1.9)$$

Góc quay có thể dương hoặc âm. Với góc quay dương, chúng ta có thể dùng ma trận quay trên. Tuy nhiên với góc quay âm, ma trận được thay đổi như sau:

$$R = \begin{bmatrix} \cos(-\theta) & \sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (1.10)$$

[1]

### Phép chuyển đổi tỷ lệ:

Để thay đổi cỡ của một vật thể, phép chuyển đổi tỷ lệ được sử dụng. Trong quá trình chuyển đổi tỷ lệ, ta có thể mở rộng hoặc thu nhỏ kích thước vật thể. Để có thể thay đổi tỉ lệ, ta nhân các tham số ban đầu của vật thể với tham số tỷ lệ để thu được kết quả mong muốn.

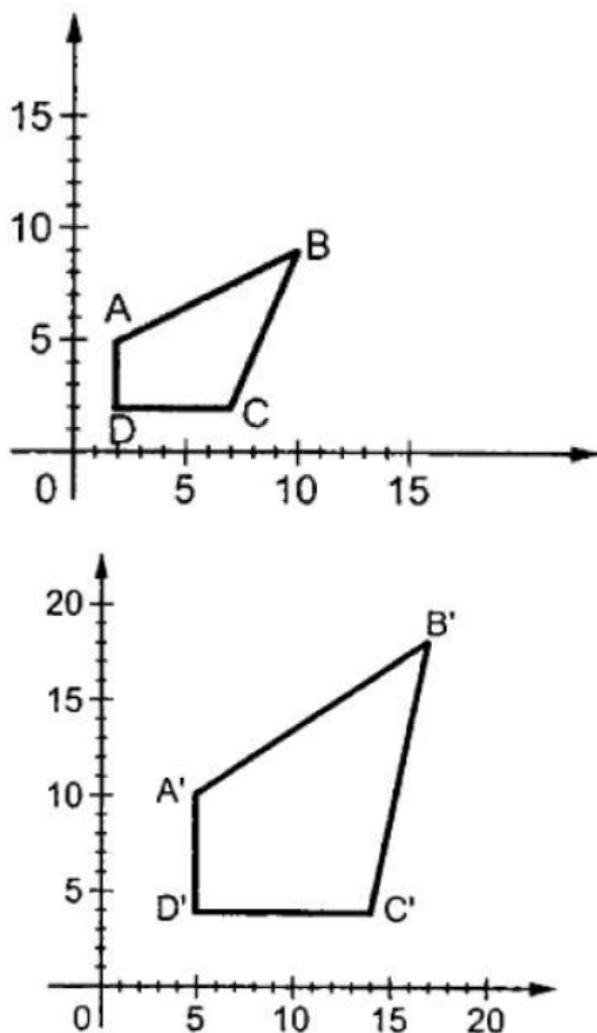
Giả sử với các giá trị ban đầu X, Y, tham số tỷ lệ là  $(S_x, S_y)$  và các giá trị sau khi thay đổi là  $X'$ ,  $Y'$ . Phép biến đổi này có thể được biểu diễn dưới dạng toán học như sau:  $X' = X.S_x$  và  $Y' = Y.S_y$ . Tham số tỷ lệ  $S_x, S_y$  thay đổi tỷ lệ vật thể theo hướng X, Y tương ứng. Hệ thức trên có thể được biểu diễn lại dưới dạng ma trận như sau:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix} \quad (1.11)$$

OR

$$P' = P.S \quad (1.12)$$

Trong đó, S là ma trận tỷ lệ. Phép tỷ lệ có thể được biểu diễn như hình 19.

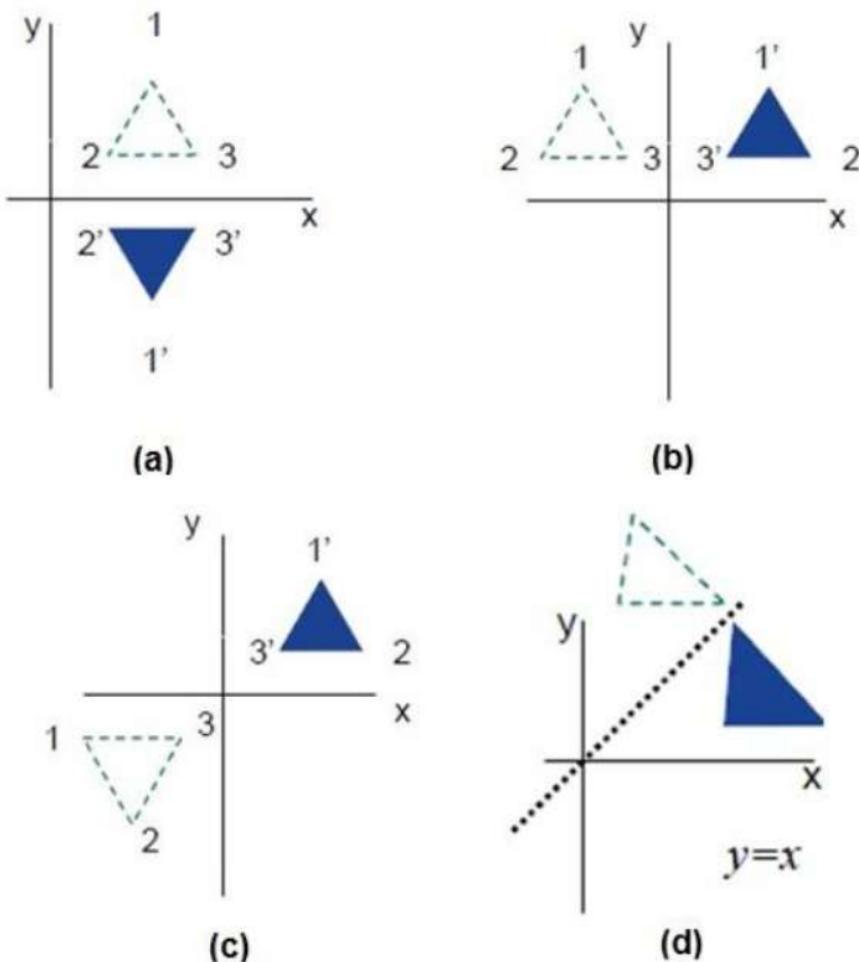


Hình 1.19: Hình ảnh mô tả phép chuyển đổi tỷ lệ.

Nếu chúng ta cung cấp một tham số tỷ lệ  $S$  nhỏ hơn 1, ta có thể làm giảm kích thước của vật thể. Nếu chúng ta cung cấp một tham số tỷ lệ  $S$  lớn hơn 1, ta có thể tăng kích thước hình ảnh. [1]

## Phép biến đổi phản xạ:

Phép phản xạ giúp ta thu được một hình ảnh gương của hình ảnh gốc. Hay một cách nói khác, là trường hợp quay  $180^\circ$ . Trong phép biến đổi phản xạ, kích thước của hình ảnh không thay đổi. Hình 20 miêu tả phép phản xạ với trục X, Y và về gốc tọa độ tương ứng. [1]



Hình 1.20: Hình ảnh mô tả phép phản xạ.

### 1.2.7 Phân đoạn ảnh:

Trọng tâm của phần này là các phương pháp phân đoạn dựa trên việc phát hiện những thay đổi về cục bộ, sắc nét và cường độ. Ba loại đặc điểm hình ảnh mà chúng ta quan tâm là các điểm, đường và cạnh.

Pixel biên là những pixel tại đó cường độ sáng của hình ảnh đột ngột và các cạnh là tập hợp các pixel cạnh được kết nối.

Bộ phát hiện cạnh là phương pháp xử lý hình ảnh cục bộ được thiết kế để phát hiện các pixel cạnh. Một đường có thể coi là một cạnh trong đó cường độ của nền hai bên cao hơn hoặc thấp hơn nhiều so với cường độ các pixel trên cạnh.

Như đã nói trước đó, việc tính trung bình các pixel xung quanh sẽ giúp làm mịn bước ảnh hơn. Cho rằng tính trung bình tương tự tích phân, tương tự như vậy, chúng ta sẽ sử dụng đạo hàm để phát hiện sự thay đổi cục bộ, đột ngột về cường độ. Đạo hàm của hàm số được xác định theo độ sai phân. Có nhiều cách để nhận biết sự khác biệt này nhưng như ta đã quy ước rằng bất cứ phép tính nào được sử dụng cho đạo hàm bậc nhất phải bằng 0 khi cường độ không thay đổi; phải khác 0 khi bắt đầu có sự thay đổi cường độ; và phải khác 0 tại các điểm dọc dốc cường độ. Tương tự với các đạo hàm bậc hai, phải bằng 0 tại điểm cường độ không đổi; khác 0 tại điểm bắt đầu có sự thay đổi cường độ và bằng 0 dọc các đường dốc cường độ. Vì các đại lượng đang xử lý có giá trị hữu hạn cho nên sự thay đổi cường độ tối đa có thể xảy ra cũng hữu hạn và khoảng cách ngắn nhất mà sự thay đổi có thể xảy ra là các pixel liền kề nhau.

Về cơ bản, đạo hàm bậc nhất tìm tạo ra các cạnh dày hơn trong ảnh. Đạo hàm bậc hai có phản ứng mạnh hơn với các chi tiết nhỏ như đường mảnh, điểm nhiễu cô lập,... Đạo hàm bậc hai tạo ra phản ứng hai cạnh khi chuyển đổi cường độ và bước chuyển tiếp. Dấu của đạo hàm bậc hai có thể được sử dụng để xác định một cạnh chuyển từ sáng sang tối hay ngược lại. [1]

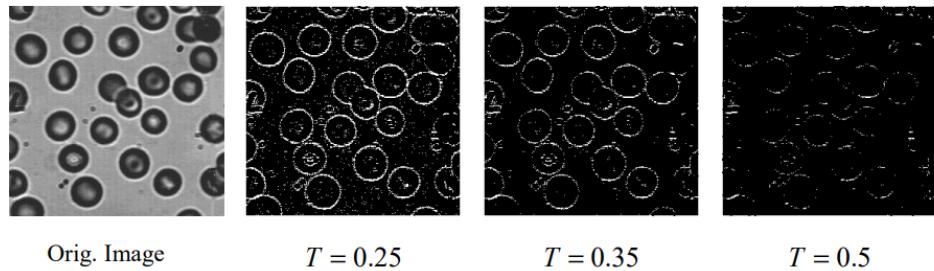
### Nhận diện điểm cô lập:

Cường độ sáng của các điểm cô lập thường rất khác biệt so với các điểm lân cận nó. Nó có thể được nhận biết bằng một kernel 3x3

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Đầu ra thường được đặt trước một ngưỡng  $T$  xác định. Chúng ta có thể xác định một điểm cô lập trong trường hợp ngưỡng  $T$  là một ngưỡng không âm xác định trước bằng hệ thức:

$$|8f_5 - (f_1 + f_2 + f_3 + f_4 + f_6 + f_7 + f_8 + f_9)| > T \quad (1.13)$$



Hình 1.21: Các điểm cô lập ở các mức ngưỡng khác nhau.

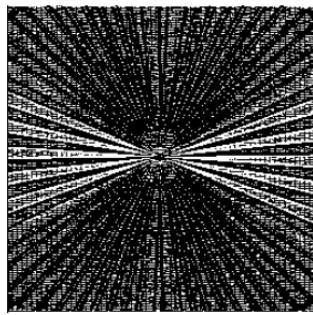
### Nhận diện đường:

Việc nhận diện cạnh được thực hiện thông qua bốn bộ lọc:

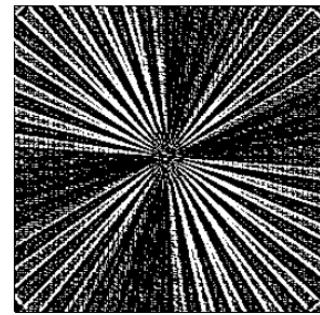
$$D_0 = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{Detects horizontal lines} \quad D_{90} = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \quad \text{Detects vertical}$$

$$D_{45} = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \quad \text{Detects } 45^\circ \text{ lines} \quad D_{135} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \quad \text{Detects } 135^\circ$$

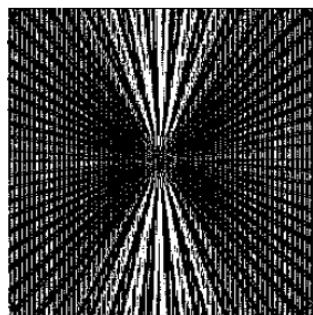
Lấy  $R_0, R_{45}, R_{90}, R_{135}$  là các phản hồi cho  $D_0, D_{45}, D_{90}, D_{135}$  tương ứng. Tại một pixel  $(m, n)$  cho trước, nếu  $R_{135}$  là lớn nhất trong bộ bốn số phản hồi thì ta có thể nói là dòng  $135^\circ$  là dòng có nhiều khả năng đi qua pixel đó nhất.[1]



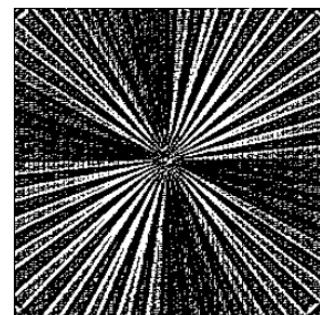
$$R_{0^\circ} = \max\{R_{0^\circ}, R_{45^\circ}, R_{90^\circ}, R_{135^\circ}\}$$



$$R_{45^\circ} = \max\{R_{0^\circ}, R_{45^\circ}, R_{90^\circ}, R_{135^\circ}\}$$



$$R_{90^\circ} = \max\{R_{0^\circ}, R_{45^\circ}, R_{90^\circ}, R_{135^\circ}\}$$



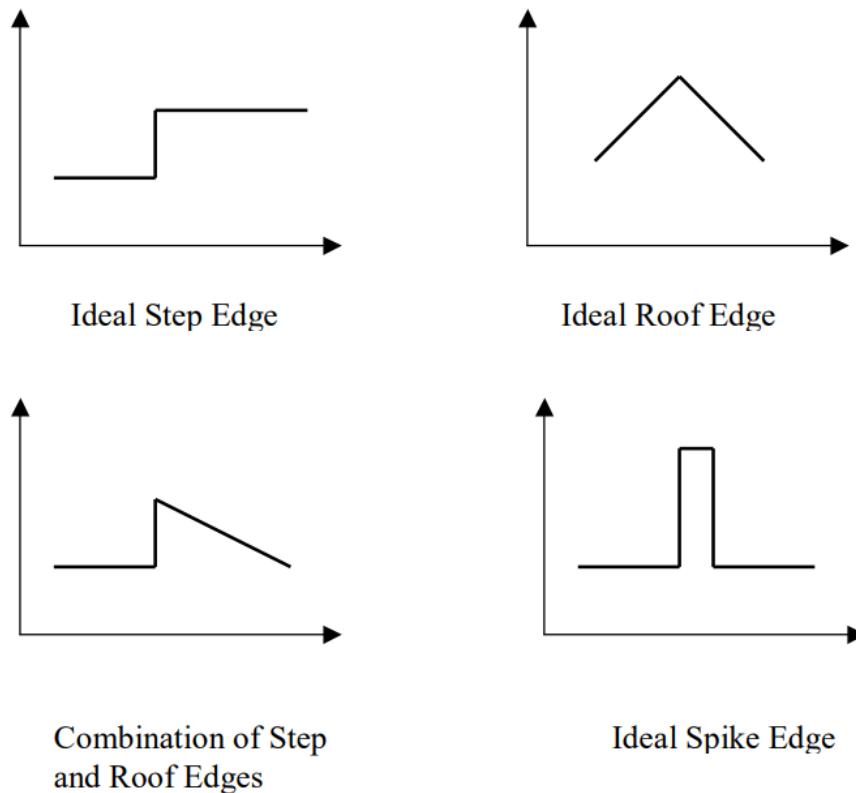
$$R_{135^\circ} = \max\{R_{0^\circ}, R_{45^\circ}, R_{90^\circ}, R_{135^\circ}\}$$

Hình 1.22: Kết quả thu được với các phản hồi lớn nhất khác nhau.

### Nhận diện cạnh:

Trên thực tế, việc nhận biết các điểm và đường thường không có nhiều ứng dụng trong thực tế. Người ta thường quan tâm nhiều hơn đến việc nhận diện ranh giới giữa những vùng có mức độ sáng khác nhau. Chúng ta giả định rằng các vùng được đề cập là đều đồng nhất để quá trình chuyển đổi giữa hai vùng có thể được xác định chỉ dựa trên sự gián đoạn ở mức xám. Một cạnh trong ảnh có thể được định nghĩa là sự gián đoạn hoặc thay đổi đột ngột về mức xám. Tuy nhiên các trường hợp lý tưởng này thường không xảy ra. Đồng thời, trong không gian hai chiều, các cạnh có thể xuất hiện ở bất kỳ hướng nào. Các cạnh thường không được biểu diễn gián đoạn một cách tuyệt đối. Vì vậy, các tác vụ trong việc nhận diện cạnh trở nên phức tạp hơn. Để nhận diện cạnh một cách hiệu quả người ta sử dụng vi phân bậc nhất và vi phân bậc hai. Ví dụ, độ lớn đạo hàm bậc nhất có thể được sử dụng để phát hiện cạnh; dấu của đạo hàm bậc hai có thể được sử dụng để nhận biết các pixel của cạnh nằm trên phần sáng hay tối của cạnh,...

Giao điểm 0 của đạo hàm bậc hai cung cấp một cách mạnh mẽ để định vị các cạnh



Hình 1.23: Một số cạnh lý tưởng.

trong ảnh. Ta ưu tiên sử dụng một kernel có kích thước nhỏ để phân biệt những sự thay đổi nhỏ trong mức xám. Mặt khác, ta sử dụng những kernel có bộ lọc lớn hơn để nhận biết những sự thay đổi lớn trong mức xám, đồng thời lọc nhiễu và các điểm bất thường khác. Do đó, chúng ta cần tìm kích thước mặt nạ, đây là sự dung hòa giữa hai yêu cầu đối lập này hoặc xác định nội dung cạnh bằng cách sử dụng các kích thước mặt nạ khác nhau.

Toán tử vi phân phổ biến nhất là đạo hàm.

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

Độ lớn và hướng của đạo hàm được xác định bởi công thức:

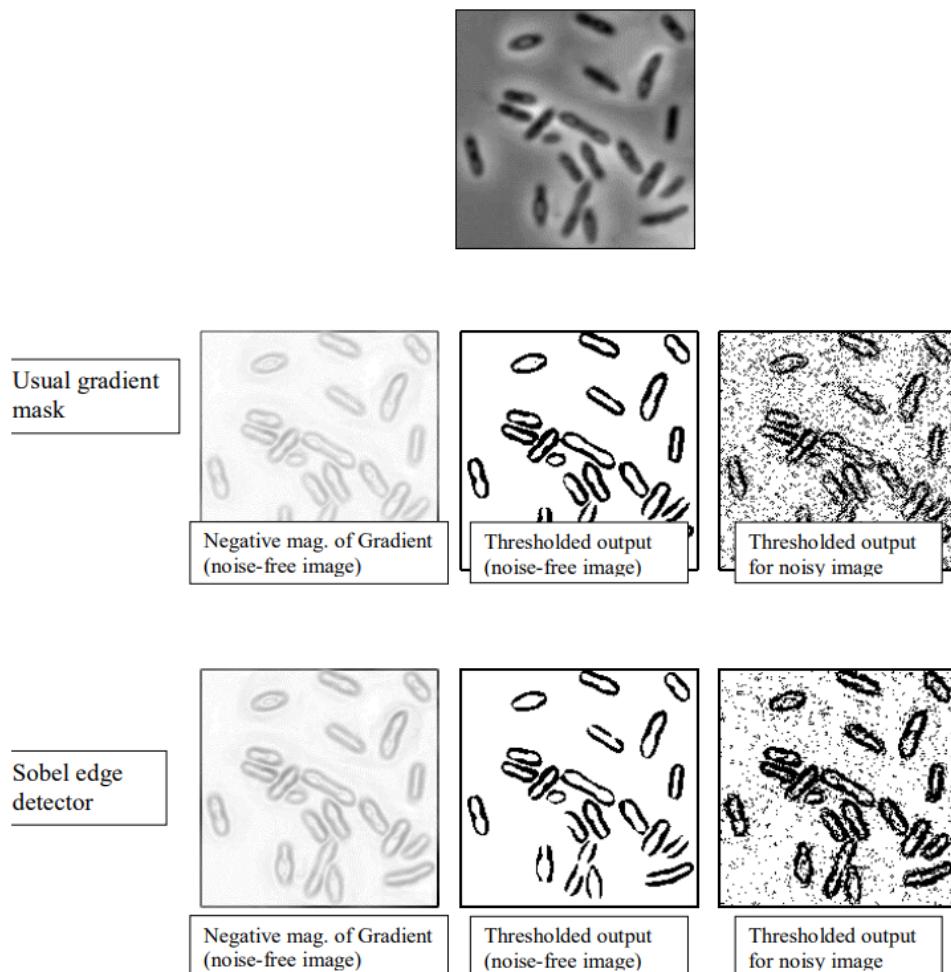
$$|\nabla f(x, y)| = \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2}$$

$$\Delta \nabla f(x, y) = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$$

Trong thực tế, chúng ta sử dụng những phép tính gần đúng rời rạc của đạo hàm riêng, được thực hiện bằng các sử dụng bộ lọc:

$$D_h = \frac{1}{2} \left\{ \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \right\} = \frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$D_v = \frac{1}{2} \left\{ \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \right\} = \frac{1}{2} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$



Hình 1.24: So sánh kết quả lọc cạnh có bộ lọc làm mịn và không có.

Bởi vì các đạo hàm tăng cường nhiễu, các toán tử trước đó có thể không cho kết quả tốt nếu hình ảnh đầu vào quá nhiễu. Một cách để chống lại ảnh hưởng của tiếng nhiễu

là sử dụng bộ lọc làm mịn. Bộ tách biên Sobel kết hợp phép toán làm mịn này cùng với phép toán đạo hàm cho ra các mặt nạ sau. Do phương pháp phát hiện cạnh gradient chỉ phụ thuộc vào độ lớn tương đối trong ảnh nên phép nhân vô hướng với các hệ số như 1/2 hoặc 1/8 không đóng vai trò thiết yếu. Điều này cũng đúng với các dấu hiệu của các mục của bộ lọc. Do đó, các bộ lọc tương ứng với cùng một bộ nhận diện, cụ thể là bộ nhận diện biên Sobel. Tuy nhiên, khi độ lớn chính xác là quan trọng thì nên sử dụng hệ số nhân vô hướng thích hợp. Tất cả các bộ lọc được xem xét cho đến nay đều có các mục có tổng bằng 0. Đây là điển hình của bất kỳ bộ lọc vi phân nào.

### **Nhận diện cạnh Laplacian:**

Trong nhiều ứng dụng, việc xây dựng các toán tử đạo hàm có tính *đẳng hướng* (bất biến phép quay) được đặc biệt quan tâm. Điều này có nghĩa là việc xoay ảnh  $f$  và áp dụng toán tử  $\nabla$  cho kết quả tương tự như việc áp dụng toán tử trên  $f$  rồi xoay kết quả. Ta muốn toán tử *đẳng hướng* vì chúng tôi muốn làm sắc nét các cạnh chạy theo bất kỳ hướng nào như nhau. Việc này có thể được thực hiện bằng một bộ nhận diện khác, dựa trên biến đổi Laplacian của hàm hai biến. Biến đổi Laplacian cho hàm hai biến được biểu diễn là:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Tuy nhiên, toán tử Laplacian có những nhược điểm sau: Nó là toán tử đạo hàm bậc hai và do đó cực kỳ nhạy cảm với nhiễu. Đồng thời toán tử Laplacian tạo ra các cạnh kép và không thể phát hiện hướng của cạnh. Laplacian thường đóng vai trò thứ yếu trong việc phát hiện cạnh. Nó đặc biệt hữu ích khi quá trình chuyển đổi mức xám ở biên không đột ngột mà diễn ra từ từ.

## Chương 2 Các kỹ thuật xử lý ảnh:

Các kỹ thuật trích xuất ảnh được trình bày ở phần trên có thể được thực hiện một cách đơn giản thông qua thư viện OpenCV. OpenCV là tên viết tắt cho Open Computer Vision library – có thể được hiểu là một thư viện mở cho máy tính. Cụ thể là một kho lưu trữ các mã nguồn mở thường được sử dụng để xử lý hình ảnh, phát triển độ họa trong thời gian thực. OpenCV là một phần mềm đa nhiệm, nó được sử dụng cho nhiều trường hợp khác nhau. Tất cả các công nghệ hiện đại ngày nay đều có sự góp mặt của OpenCV trong việc xử lý ảnh.

### 2.1 Các lệnh khai báo và đọc dữ liệu:

#### 2.1.1 Lệnh khai báo:

Để có thể sử dụng thư viện OpenCV trên Python, trước hết ta cần phải khai báo thư viện trong chương trình.

```
In [1]: import cv2
```

Trong lệnh khai báo có hai thành phần:

- *import*: Đây là từ khóa trong Python được sử dụng để nhập một module hoặc một thư viện vào chương trình..
- *cv2*: đây là tên của thư viện OpenCV. Khi ta *import cv2* có nghĩa là ta đang đưa toàn bộ thư viện OpenCV vào trong không gian làm việc của mình.

Sau câu lệnh, ta có thể sử dụng tất cả chức năng và lớp có sẵn trong thư viện OpenCV trong chương trình Python của mình.

#### 2.1.2 Đọc dữ liệu:

Để chương trình có thể biết được dữ liệu mà chúng ta cần xử lý, ta cần một câu lệnh để đọc trước dữ liệu cần được xử lý.

```
In [1]: img = cv2.imread('path/to/your/image.jpg')
```

Lệnh đọc dữ liệu gồm 3 phần:

- `cv2.imread()`: Đây là một hàm của thư viện OpenCV được sử dụng để đọc ảnh từ một tệp hình ảnh. Hàm này trả về một mảng NumPy biểu diễn hình ảnh.
  - `img`: đây là một biến có tên là `img` có tác dụng lưu trữ thông tin thu được sau quá trình đọc ảnh.
  - `'path/to/your/image.jpg'`: Đây là đường dẫn tới ảnh cần được đọc. Nếu không cung cấp đường dẫn nào, hàm sẽ trả về “None”.

Toàn bộ câu lệnh có tác dụng đọc và lưu lại dữ liệu từ một ảnh để sử dụng trong quá trình xử lý ảnh tiếp theo.

```
1 import cv2  
2 from matplotlib import pyplot as plt  
3  
1 img = cv2.imread('/content/PHENIKAA.jpg')  
2 plt.imshow(img)
```



Hình 2.1: Hình ảnh sau khi được đọc và dựng lại.

## 2.2 Các lệnh liên quan màu sắc, kích thước ảnh:

### 2.2.1 Các lệnh liên quan màu sắc:

#### Lệnh chuyển đổi giữa các không gian màu:

Trong quá trình xử lý ảnh, việc chuyển đổi giữa các không gian màu khác nhau là một tác vụ cơ bản và được sử dụng nhiều với nhiều mục đích khác nhau. Lệnh chuyển đổi hình ảnh có dạng:

```
1 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
2 plt.imshow(img)

<matplotlib.image.AxesImage at 0x7f4bb72dfcd0>

```

Hình 2.2: Hình ảnh sau khi được chuyển đổi sang không gian màu RGB

Lệnh này gồm 3 thành phần chính:

- *cv2.cvtColor()*: Là một hàm trong thư viện OpenCV được sử dụng để chuyển đổi không gian màu của ảnh.
- *img*: Là biến chứa dữ liệu của ảnh.
- *cv2.COLOR\_BGR2RGB*: Là một hằng số chỉ định loại chuyển đổi màu. Trong trường hợp này, *cv2.COLOR\_BGR2RGB* chỉ định chuyển đổi từ không gian màu BGR sang không gian màu RGB.

Lệnh này thực hiện việc chuyển đổi không gian màu. Chuyển đổi không gian màu thường được sử dụng khi ta muốn hiện thị hình ảnh bằng một thư viện hoặc chức năng Python khác, vì nhiều không gian sử dụng không gian màu khác nhau. Sau lệnh này, biến *img* sẽ chứa dữ liệu về ảnh với không gian màu được chuyển đổi sang RGB. [2]

### Lệnh chuyển đổi từ ảnh màu sang ảnh xám:

Tuy đây cũng là một trường hợp trong việc chuyển đổi không gian màu nhưng vì mức độ quan trọng của ảnh xám trong quá trình xử lý ảnh mà ta sẽ đưa nó ra thành một trường hợp riêng.

```
1 gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
2 plt.imshow(gray_img, cmap = 'gray')

<matplotlib.image.AxesImage at 0x7f4bb709e190>

```

Hình 2.3: Hình ảnh sau khi được chuyển sang ảnh xám.

Cũng tương tự như việc chuyển đổi màu từ không gian BGR sang RGB. Do trong không gian xám, mỗi điểm ảnh chỉ chứa một giá trị duy nhất biểu thị cường độ sáng. Kết quả là một ảnh xám có thể được sử dụng cho các nhiệm vụ xử lý ảnh như trích dẫn đặc trưng, phân tích cường độ, và nhận diện đối tượng mà không cần thông tin màu sắc chi tiết.[2]

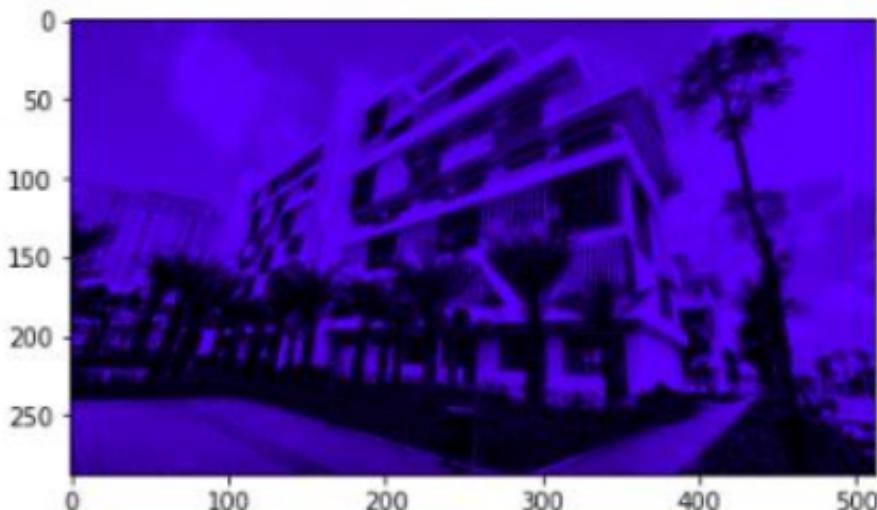
## Lệnh lấy ra một kênh màu của ảnh đầu vào:

Một hình ảnh màu số được cấu thành từ việc pha trộn ba màu là màu đỏ, xanh lục, và xanh dương. Do đó, việc biết cường độ mỗi màu cơ bản tại một pixel ảnh bất kỳ sẽ giúp ta biết được rõ hơn về bức ảnh. Trong cấu trúc lệnh lấy một kênh màu có hai phần:

- $img[:, :, [0, 1]]$ : Đây là cú pháp giúp truy cập vào tất cả pixel màu xanh lục và màu đỏ trong ảnh. Trong đó, phần  $[:, :, :]$  là để chia ra các cột và các hàng được chọn, ở đây không có chỉ số bắt đầu và kết thúc, có nghĩa là ta sẽ chọn tất cả các cột, hàng có trong dữ liệu ảnh lưu trữ trong  $img$ ; phần  $[0, 1]$  chỉ số chọn các kênh màu tương ứng. Do không gian màu ta đang sử dụng là RGB nên chỉ số màu đỏ và xanh lục sẽ lần lượt là 0 và 1.
- $= 0$ : việc này có nghĩa là tất cả giá trị màu xanh lục và đỏ thu được đều sẽ được đặt bằng 0.

```
1 a = img.copy()
2 a[:, :, [0, 1]] = 0
3 plt.imshow(a)
```

```
<matplotlib.image.AxesImage at 0x7f4bb7253390>
```



Hình 2.4: Hình ảnh chỉ lấy kênh màu xanh.

Việc đặt hai màu sắc này bằng 0 làm thay đổi màu sắc tổng thể của bức ảnh. Trong không gian ảnh thì mỗi màu sắc cơ bản đều góp phần mô tả màu sắc hình ảnh. Bằng cách đặt giá trị này bằng 0, ta sẽ giảm độ đỏ và lục của bức ảnh tạo ra một hiệu ứng màu sắc khác nhau trong ảnh. Việc lấy từng kênh màu riêng biệt giúp ta hiểu rõ hơn về màu sắc ảnh, có thể thay đổi, chỉnh sửa riêng từng màu một cách linh hoạt.

### Lệnh lấy giá trị từng màu trong không gian màu của 1 pixel:

Trong một số trường hợp, chúng ta cần biết chính xác màu sắc của từng pixel. Khi đó ta cần có giá trị về cường độ sáng của từng màu theo không gian màu tương ứng với ảnh hiện tại.. Dưới đây là một câu lệnh có tác dụng lấy giá trị cường độ màu sắc của một pixel tương ứng:

```
1 print(img[0, 0])
```

```
[ 9 110 188]
```

Hình 2.5: Cường độ màu sắc 1 pixel.

Hình trên là minh họa cho việc in ra cường độ sáng trong một pixel ảnh, cụ thể ở đây là pixel ở vị trí cột 0 và hàng 0. Kết quả thu được là một list gồm ba giá trị lần lượt là cường độ của màu đỏ, màu xanh lục và màu xanh dương. Việc này giúp ta hình dung được màu sắc của pixel và có thể tinh chỉnh màu một cách hiệu quả hơn.

#### 2.2.2 Lệnh liên quan đến kích thước ảnh:

##### Lệnh lấy ra kích thước ảnh:

Mỗi bức ảnh có một kích thước khác nhau và việc biết được kích thước ảnh sẽ giúp tách ảnh tốt hơn. Dưới đây là một cách để lấy ra kích thước ảnh:

```
1 print(img.shape)
```

```
(288, 512, 3)
```

Hình 2.6: Kích thước một ảnh

Lệnh này trả về thông tin về chiều cao, chiều rộng và số kênh màu của ảnh. Trong đó, giá trị đầu tiên là chiều cao của ảnh hay số pixel theo chiều dọc, trong ví dụ này là 288 pixel. Tương tự như thế, giá trị thứ hai là chiều rộng của ảnh hay số pixel theo chiều

ngang, trong ví dụ này là 512 pixel. Giá trị cuối là số kênh màu của ảnh, trong ví dụ này, ảnh đang ở không gian màu RGB nên số kênh màu tương ứng là 3.

### Lệnh thay đổi kích thước ảnh:

Trong một số trường hợp, ta cần thay đổi kích thước ảnh cho phù hợp với yêu cầu của công việc. Lệnh sử dụng để làm việc này là:

```
In [1]: img = cv2.resize(img, (300, 150))
```

Lệnh này gồm ba phần:

- *cv2.resize()*: Đây là hàm trong thư viện OpenCV có tác dụng thay đổi kích thước ảnh.
- *img*: Đây là biến chứa dữ liệu về ảnh.
- *(300, 150)*: Đây là kích thước mới của ảnh. Trong trường hợp này, chiều rộng là 300 pixel còn chiều cao sẽ là 150 pixel.

Việc thay đổi kích thước của ảnh sẽ làm thay đổi tỷ lệ khung hình và góc nhìn của ảnh.

## 2.3 Vẽ với OpenCV:

### 2.3.1 Các hình khối cơ bản:

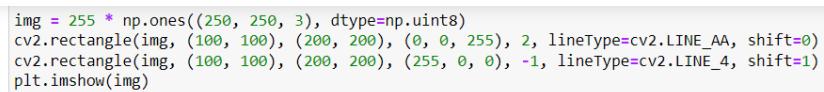
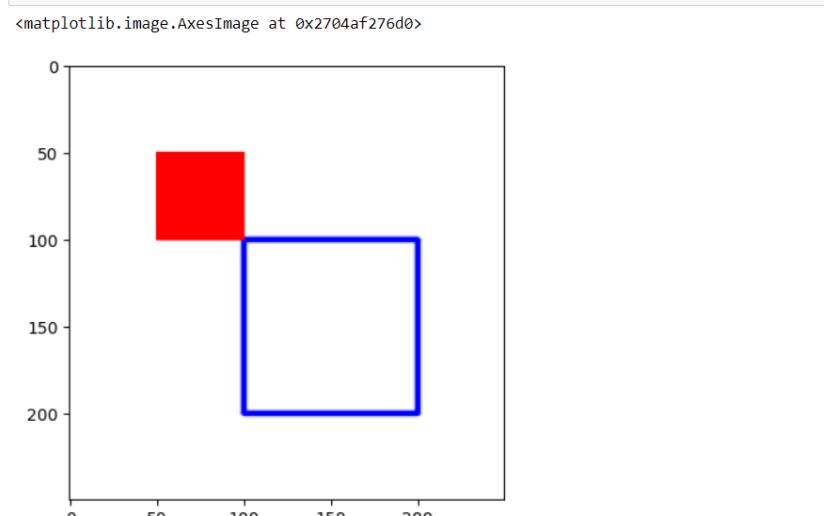
#### Hình chữ nhật:

Để vẽ một hình chữ nhật bằng OpenCV ta sẽ sử dụng lệnh *cv2.rectangle()*. Lệnh này gồm có 7 tham số lần lượt là:

- *img*: Ảnh trên đó bạn muốn vẽ hình chữ nhật. Đây là ảnh mà hình chữ nhật sẽ được vẽ lên.
- *pt1*: Điểm đầu tiên của hình chữ nhật, là một tuple (x, y) đại diện cho tọa độ của góc trái trên (hoặc dưới) của hình chữ nhật.

- *pt2*: Điểm thứ hai của hình chữ nhật, là một tuple (x, y) đại diện cho tọa độ của góc phải dưới (hoặc trên) của hình chữ nhật.
- *color*: Màu của hình chữ nhật. Đối với ảnh màu, màu có thể được biểu diễn dưới dạng tuple (B, G, R), trong đó B, G, R là giá trị màu xanh, màu lục và màu đỏ. Đối với ảnh xám, màu thường là một giá trị scalar (grayscale).
- *thickness*: Độ dày của đường viền của hình chữ nhật. Nếu bạn đặt giá trị là -1, hình chữ nhật sẽ được vẽ toàn bộ và sẽ không có đường viền.
- *lineType*: Kiểu đường vẽ. Có thể là 8-connected (mặc định), 4-connected, hoặc cv2.LINE\_AA cho vẽ đường liền mịn (anti-aliased).
- *shift*: Số bit dịch chuyển để nhân với các giá trị pixel của tọa độ. Thường được đặt là 0.

```

```

Out[24]: <matplotlib.image.AxesImage at 0x2704af276d0>

Hình 2.7: Vẽ hình vuông.

### Hình tròn:

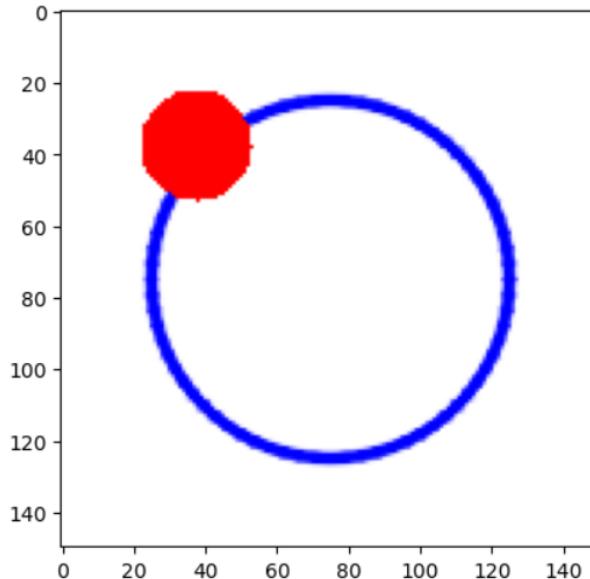
Để vẽ hình tròn, OpenCV cung cấp cho ta lệnh cv2.circle(). Lệnh này có 7 tham số lần lượt là:

- *img*: Ảnh trên đó bạn muốn vẽ hình tròn. Đây là ảnh mà hình tròn sẽ được vẽ lên.

- *center*: Tọa độ của trung tâm của hình tròn, là một tuple (x, y).
- *radius*: Bán kính của hình tròn, là một số nguyên dương.
- *color*: Màu của hình tròn. Đối với ảnh màu, màu có thể được biểu diễn dưới dạng tuple (B, G, R), trong đó B, G, R là giá trị màu xanh, màu lục và màu đỏ. Đối với ảnh xám, màu thường là một giá trị scalar (grayscale).
- *thickness*: Độ dày của đường viền của hình tròn. Nếu bạn đặt giá trị là -1, hình tròn sẽ được vẽ toàn bộ và sẽ không có đường viền.
- *lineType*: Kiểu đường vẽ. Có thể là 8-connected (mặc định), 4-connected, hoặc cv2.LINE\_AA cho vẽ đường liền mịn (anti-aliased).
- *shift*: Số bit dịch chuyển để nhân với các giá trị pixel của tọa độ. Thường được đặt là 0.

```
img = 255 * np.ones((150, 150, 3), dtype=np.uint8)
cv2.circle(img, (75, 75), 50, (0, 0, 255), 2, lineType=cv2.LINE_AA, shift=0)
cv2.circle(img, (75, 75), 30, (255, 0, 0), -1, lineType=cv2.LINE_4, shift=1)
plt.imshow(img)
```

: <matplotlib.image.AxesImage at 0x1fe928982d0>



Hình 2.8: Vẽ hình tròn.

## Văn bản:

Để đưa một văn bản vào hình ảnh, ta dùng lệnh cv2.putText(). Lệnh này thường được sử dụng khi ta muốn đưa thêm thông tin nhằm giải thích cho một sự vật hay sự kiện trong ảnh. cv2.putText() có 9 tham số đầu vào lần lượt là:

- *img*: Ảnh trên đó bạn muốn vẽ văn bản. Đây là ảnh mà văn bản sẽ được vẽ lên.
- *text*: Chuỗi văn bản cần được vẽ.
- *org*: Tọa độ của điểm gốc trái dưới của văn bản, là một tuple (x, y).
- *fontFace*: Kiểu font. Có thể sử dụng các hằng số có sẵn như cv2.FONT\_HERSHEY\_SIMPLEX cv2.FONT\_HERSHEY\_PLAIN, cv2.FONT\_HERSHEY\_DUPLEX, v.v.
- *fontSize*: Tỉ lệ kích thước của font.
- *color*: Màu của văn bản. Đôi với ảnh màu, màu có thể được biểu diễn dưới dạng tuple (B, G, R), trong đó B, G, R là giá trị màu xanh, màu lục và màu đỏ. Đôi với ảnh xám, màu thường là một giá trị scalar (grayscale).
- *thickness*: Độ dày của văn bản.
- *lineType*: Kiểu đường vẽ. Có thể là 8-connected (mặc định), 4-connected, hoặc cv2.LINE\_AA cho vẽ đường liền mịn (anti-aliased).
- *bottomLeftOrigin*: Nếu đặt là True, org sẽ được hiểu là góc dưới cùng bên trái của văn bản. Nếu đặt là False (mặc định), org sẽ là góc trái dưới của hộp giữ văn bản.

## Vẽ đường thẳng:

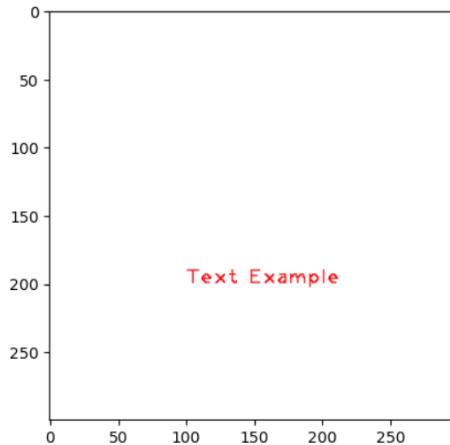
Để vẽ một đường thẳng vào ảnh, ta sẽ sử dụng lệnh cv2.line(). Lệnh này gồm 7 tham số như sau:

- *img*: Ảnh trên đó bạn muốn vẽ đường thẳng. Đây là ảnh mà đường thẳng sẽ được vẽ lên.
- *pt1*: Điểm đầu tiên của đường thẳng, là một tuple (x, y).

```



```



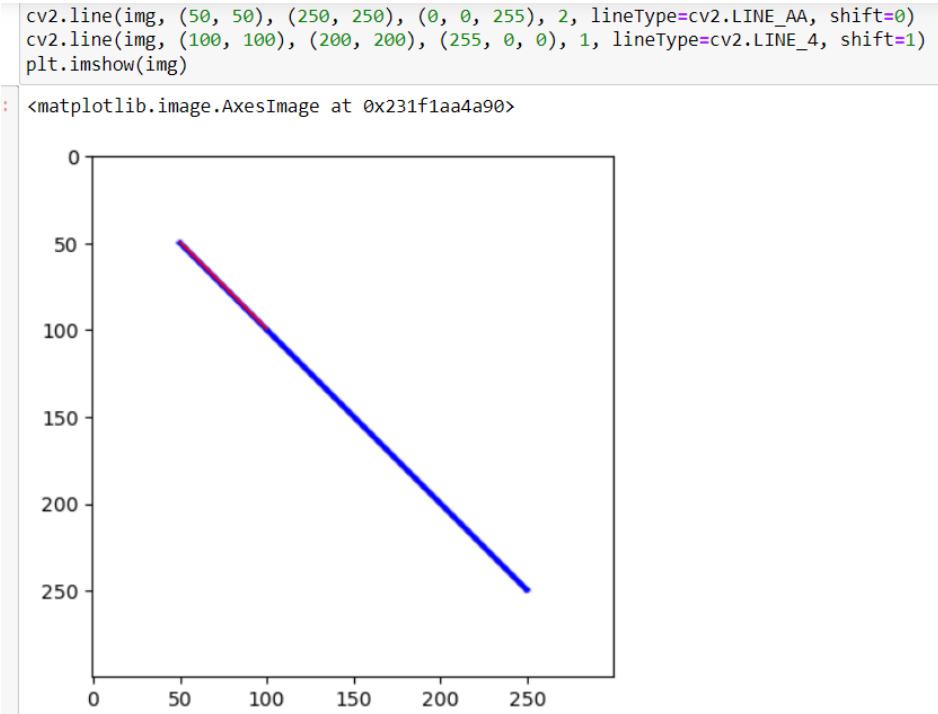
Hình 2.9: Thêm văn bản.

- *pt2*: Điểm thứ hai của đường thẳng, là một tuple (x, y).
- *color*: Màu của đường thẳng. Đối với ảnh màu, màu có thể được biểu diễn dưới dạng tuple (B, G, R), trong đó B, G, R là giá trị màu xanh, màu lục và màu đỏ. Đối với ảnh xám, màu thường là một giá trị scalar (grayscale).
- *thickness*: Độ dày của đường thẳng.
- *lineType*: Kiểu đường vẽ. Có thể là 8-connected (mặc định), 4-connected, hoặc cv2.LINE\_AA cho vẽ đường liền mịn (anti-aliased).
- *shift*: Số bit dịch chuyển để nhân với các giá trị pixel của tọa độ. Thường được đặt là 0.

### Vẽ đa giác:

Để vẽ một đa giác trên ảnh, ta dùng lệnh cv2.polyline(). Lệnh này gồm có 7 tham số như sau:

- *img*: Ảnh trên đó bạn muốn vẽ đa giác. Đây là ảnh mà đa giác sẽ được vẽ lên.
- *pts*: Một danh sách các điểm tạo nên đa giác. Mỗi điểm là một tuple (x, y).



Hình 2.10: Vẽ đường thẳng.

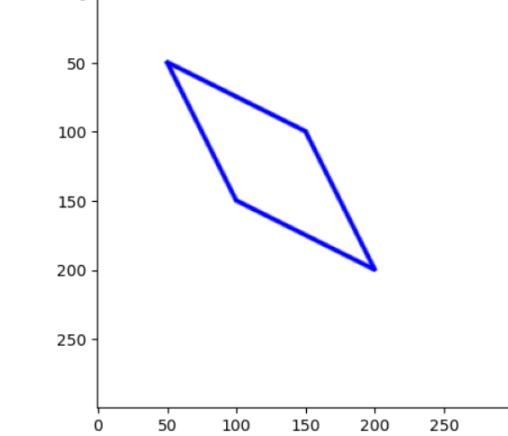
- *isClosed*: Một cờ (Boolean) xác định xem đa giác có được đóng hay không. Nếu là True, đa giác sẽ được đóng bằng cách nối điểm cuối với điểm đầu, tạo thành một hình với nhiều cạnh. Nếu là False, đa giác sẽ không được đóng và chỉ là một chuỗi các đoạn thẳng nối liền.
- *color*: Màu của đa giác. Đối với ảnh màu, màu có thể được biểu diễn dưới dạng tuple (B, G, R), trong đó B, G, R là giá trị màu xanh, màu lục và màu đỏ. Đối với ảnh xám, màu thường là một giá trị scalar (grayscale).
- *thickness*: Độ dày của đường vẽ.
- *lineType*: Kiểu đường vẽ. Có thể là 8-connected (mặc định), 4-connected, hoặc cv2.LINE\_AA cho vẽ đường liền mịn (anti-aliased).
- *shift*: Số bit dịch chuyển để nhân với các giá trị pixel của tọa độ. Thường được đặt là 0.

```

pts = np.array([[50, 50], [150, 100], [200, 200], [100, 150]], np.int32)
pts = pts.reshape((-1, 1, 2))
cv2.polylines(img, [pts], isClosed=True, color=(0, 0, 255), thickness=2, lineType=cv2.LINE_AA, shift=0)
plt.imshow(img)

```

[3]: <matplotlib.image.AxesImage at 0x231f1b22bd0>



Hình 2.11: Vẽ đa giác.

### 2.3.2 Các tác vụ cơ bản với ảnh:

#### Cộng ảnh:

Khi có hai hình ảnh cần ghép vào nhau, ta sử dụng lệnh cv2.addWeighted(). Hàm này gồm 6 tham số lần lượt là:

- *src1*: Ảnh đầu tiên hoặc ma trận.
- *alpha*: Trọng số cho ảnh *src1*.
- *src2*: Ảnh thứ hai hoặc ma trận.
- *beta*: Trọng số cho ảnh *src2*..
- *gamma*: Hằng số được cộng vào sau khi thực hiện phép cộng có trọng số.
- *dst*: Ảnh kết quả sau khi thực hiện phép cộng có trọng số.

Công thức toán học của phép cộng có trọng số là:  $dst = src1 \times alpha + src2 \times beta + gamma$ . Trong đó: *src1* $\times$ *alpha*: Phần tử-wise (từng pixel) của ảnh *src1* được nhân với trọng số *alpha*. *src2* $\times$ *beta*: Phần tử-wise của ảnh *src2* được nhân với trọng số *beta*. *gamma*: Hằng số được cộng vào cuối cùng.

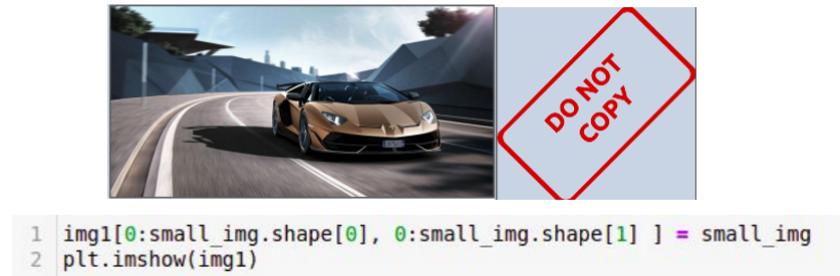


Hình 2.12: Cộng ảnh.

### Dán ảnh:

Ta có thể sử dụng đoạn mã: `img1[0: small_img.shape[0], 0: small_img.shape[1]] = small_img` để chép nội dung của ảnh `small_img` vào một phần của ảnh lớn `img1` tại một vị trí cụ thể. Dưới đây là giải thích chi tiết:

- `small_img.shape[0]` và `small_img.shape[1]`: Lấy chiều cao và chiều rộng của ảnh `small_img`. Đây là kích thước của phần ảnh cần chép.
- `img1[0: small_img.shape[0], 0: small_img.shape[1]]`: Đây là cú pháp slicing của NumPy, được sử dụng để chọn một phần của ảnh lớn `img1`. Trong trường hợp này, nó chọn một phần có kích thước bằng với ảnh `small_img` từ góc trái (0, 0) đến (chiều cao của `small_img`, chiều rộng của `small_img`).
- `= small_img`: Gán nội dung của ảnh `small_img` vào phần được chọn của ảnh `img1`. Điều này có tác dụng sao chép nội dung của ảnh nhỏ vào vị trí cụ thể của ảnh lớn.



Hình 2.13: Dán ảnh.

### Cắt ảnh:

Ta có thể cắt một phần hình ảnh trên ảnh gốc img1 và lưu vào biến roi. Trong đó:  $[400: 560]$  là dùng để cắt theo chiều dọc từ pixel thứ 400 đến pixel 559, tương tự với chiều ngang là từ pixel 600 đến pixel 954.



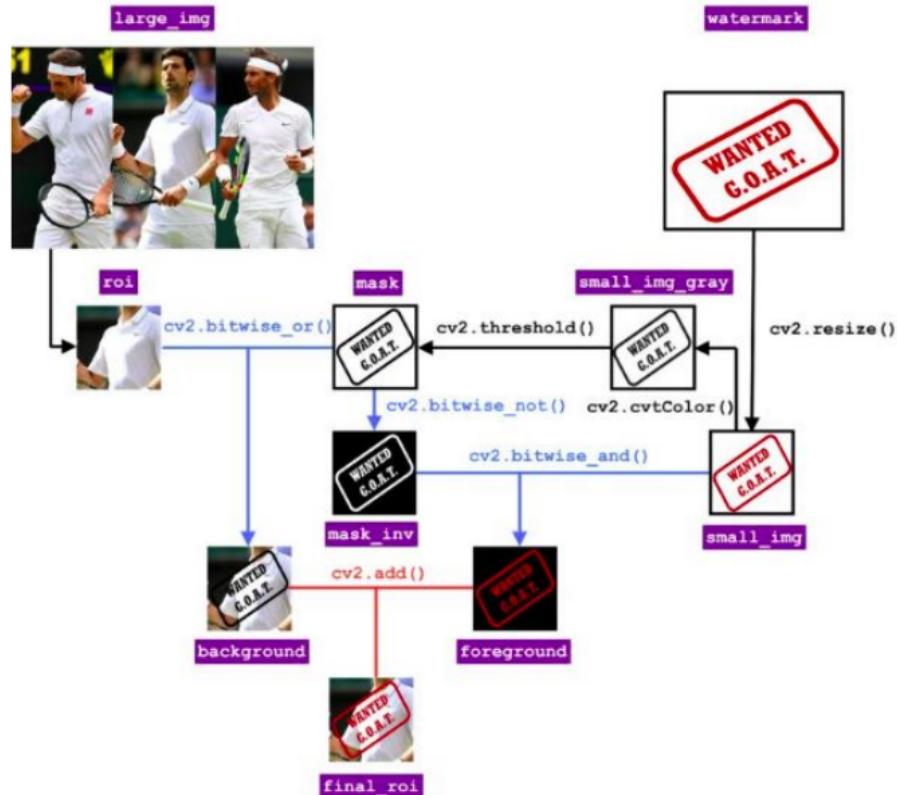
```
1 roi = img1[400:560, 600:955]  
2 plt.imshow(ro)
```



Hình 2.14: Cắt ảnh.

## Trộn ảnh:

Để có thể trộn các ảnh với nhau, ta sẽ vận dụng linh hoạt các phương thức xử lý ảnh ở trên. Dưới đây là một ví dụ cho loại này:



Hình 2.15: Trộn ảnh.

## 2.4 Biến đổi cường độ:

Các lý thuyết cơ bản về biến đổi cường độ đã được trình bày ở trên ở phần này ta chỉ đơn giản là xem cách viết chương trình biến đổi cường độ của ảnh và xem các sự thay đổi này một cách trực quan hơn.

### 2.4.1 Biến đổi Log:

```
In [7]: c = 255/(np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)
log_transformed = np.array(log_transformed, dtype = np.uint8)
```

Trong đó:



Hình 2.16: Biến đổi Log.

- $c = 255 / (\ln(1 + \max(img)))$ : Ở đây, c là một hằng số được tính toán để mở rộng động biên của hình ảnh sau khi áp dụng biến đổi logarit. Giá trị của c được tính bằng cách chia 255 cho logarit tự nhiên của ( $1 + \text{giá trị pixel lớn nhất trong hình ảnh}$ ).
- $\log_{transformed} = c * \ln(1 + img)$ :  $\log_{transformed}$  là hình ảnh đã được biến đổi, với giá trị pixel mới được tính toán bằng cách nhân c với logarit tự nhiên của ( $1 + \text{giá trị pixel gốc}$ ).
- $\log_{transformed} = \text{array}(\log_{transformed}, \text{dtype} = \text{uint8})$ : Hình ảnh biến đổi cuối cùng có kiểu dữ liệu uint8. Kiểu dữ liệu này được sử dụng để đảm bảo rằng tất cả các giá trị pixel đều nằm trong phạm vi từ 0 đến 255.

Biến đổi logarit được sử dụng trong xử lý hình ảnh để tăng cường các chi tiết màu sắc và độ tương phản của hình ảnh.

#### 2.4.2 Phép biến đổi lũy thừa:

```
In [7]: i = 1
for gamma in [0.1, 0.5, 1.2, 2.2]:
    # Apply gamma correction.
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')
    plt.subplot(2,2,i)
    plt.imshow(gamma_corrected)
    i += 1
```

Dưới đây là phần giải thích cho đoạn code trước đó:

- $i = 1$ : Khởi tạo biến đếm i với giá trị là 1.

- `for gamma in [0.1, 0.5, 1.2, 2.2]:` Vòng lặp này sẽ lặp qua một tập hợp các giá trị gamma (0.1, 0.5, 1.2, 2.2).
- `gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')`: Đây là công thức để áp dụng hiệu chỉnh gamma cho hình ảnh. Hình ảnh gốc (img) được chia cho 255 để chuẩn hóa các giá trị pixel về phạm vi từ 0 đến 1. Sau đó, lũy thừa của gamma được áp dụng cho hình ảnh đã chuẩn hóa này. Kết quả sau cùng được nhân lại với 255 và chuyển đổi thành kiểu dữ liệu uint8.
- `plt.subplot(2,2,i)`: Tạo một subplot trong một lưới 2x2 tại vị trí thứ i.
- `plt.imshow(gamma_corrected)`: Hiển thị hình ảnh đã được hiệu chỉnh gamma trên subplot.
- `i += 1`: Tăng biến đếm i lên 1 sau mỗi lần lặp.



Hình 2.17: Biến đổi lũy thừa.

#### 2.4.3 Biến đổi tuyến tính:

Dưới đây là phần giải thích code:

- `img = cv2.imread('path/to/your/image.img')`: Đọc hình ảnh từ đường dẫn đã cho và lưu vào biến img.

```
In [7]: img = cv2.imread('path/to/your/image.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def pixelVal(pix, r1, s1, r2 ,s2):
    if (0 <= pix and pix <= r1):
        return (s1 / r1)*pix
    elif (r1 < pix and pix <= r2):
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (pix - r2) + s2

r1 = 70
s1 = 0
r2 = 140
s2 = 255

pixelVal_vec = np.vectorize(pixelVal)
contrast_stretched = pixelVal_vec(img,r1,s1,r2,s2)|
```



Hình 2.18: Biến đổi tuyến tính.

- $img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)$ : Chuyển đổi hình ảnh sang ảnh xám.
- $def pixelVal(pix, r1, s1, r2 ,s2)$ : Định nghĩa hàm pixelVal để thực hiện kéo giãn tương phản dựa trên các tham số đã cho ( $r1, s1, r2, s2$ ).
- $if (0 <= pix and pix <= r1)$ : Nếu giá trị pixel nằm trong khoảng từ 0 đến  $r1$ , hàm sẽ trả về giá trị mới được tính bằng cách nhân giá trị pixel với tỷ lệ ( $s1 / r1$ ).
- $elif (r1 < pix and pix <= r2)$ : Nếu giá trị pixel nằm trong khoảng từ  $r1$  đến  $r2$ , hàm sẽ trả về giá trị mới được tính bằng cách lấy giá trị pixel trừ đi  $r1$ , nhân với tỷ lệ  $((s2 - s1)/(r2 - r1))$ , sau đó cộng thêm  $s1$ .
- $else$ : Nếu giá trị pixel lớn hơn  $r2$ , hàm sẽ trả về giá trị mới được tính bằng cách lấy giá trị pixel trừ đi  $r2$ , nhân với tỷ lệ  $((255 - s2)/(255 - r2))$ , sau đó cộng thêm  $s2$ .

- $r1 = 70, s1 = 0, r2 = 140, s2 = 255$ : Định nghĩa các tham số cho việc kéo giãn tương phản.
- $pixelVal\_vec = np.vectorize(pixelVal)$ : Vector hóa hàm pixelVal để áp dụng nó cho từng giá trị trong mảng NumPy của hình ảnh.
- $contrast\_stretched = pixelVal\_vec(img, r1, s1, r2, s2)$ : Áp dụng kéo giãn tương phản cho hình ảnh.

## 2.5 Làm mịn ảnh:

### 2.5.1 Blur:

```
In [7]: img = cv2.imread('path/to/your/image.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)      # convert BGR sang RGB
img_blur = cv2.blur(img, (10,10))
```



Hình 2.19: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép làm mịn blur.

Đoạn mã trên bao gồm 3 phần lần lượt là đọc dữ liệu ảnh, chuyển đổi không gian màu, và áp dụng phép lọc blur, dưới đây là giải thích chi tiết cho từng lệnh ở đoạn code trên:

- $img = cv2.imread('path/to/your/image.jpg')$ : Đọc hình ảnh từ đường dẫn đã cho và lưu vào biến img.
- $img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)$ : Chuyển đổi không gian màu của hình ảnh từ BGR sang RGB. BGR là không gian màu mặc định khi đọc hình ảnh bằng OpenCV nhưng RGB thì phổ biến hơn trong các ứng dụng khác.

- $img\_blur = cv2.blur(img, (10,10))$ : Áp dụng hiệu ứng làm mờ cho hình ảnh bằng cách sử dụng một kernel kích thước 10x10 pixel.

Lệnh `cv2.blur()` có tác dụng làm mờ ảnh. Lệnh này gồm có 4 tham số là:

- *src*: Đây là hình ảnh gốc mà bạn muốn thực hiện phép làm mờ lên.
- *ksize*: Đây là kích thước của kernel (bộ lọc) mà bạn sử dụng để thực hiện phép làm mờ. Kernel này xác định cách các pixel lân cận trong hình ảnh được trung bình hoặc làm mờ. *ksize* là một tuple chứa hai giá trị (*kwidth*, *kheight*) để xác định chiều rộng và chiều cao của kernel.
- *anchor (tùy chọn)*: Điểm neo (anchor) trong kernel. Mặc định là -1, -1, nghĩa là điểm neo nằm ở trung tâm của kernel. Điểm neo quyết định vị trí của kernel so với mỗi pixel trong hình ảnh.
- *borderType (tùy chọn)*: Cách xử lý biên giới của hình ảnh khi thực hiện phép làm mờ. Có các tùy chọn như `cv2.BORDER_CONSTANT`, `cv2.BORDER_REPLICATE`, `cv2.BORDER_REFLECT`, v.v.

Khi gọi hàm `cv2.blur()`, nó sẽ sử dụng kernel với kích thước *ksize* để làm mờ hình ảnh *src*. Kernel này di chuyển qua từng pixel trong hình ảnh và tính trung bình của các giá trị màu tại các pixel lân cận, sau đó gán giá trị trung bình đó cho pixel tại vị trí hiện tại, tạo ra hiệu ứng làm mờ. Sử dụng các tham số khác nhau cho *ksize*, *anchor*, và *borderType* cho phép bạn kiểm soát cách phép làm mờ được thực hiện và độ mờ của hình ảnh đầu ra. Trong ví dụ trên, ta đã sử dụng một bộ lọc có kích thước (10,10) lên hình ảnh *img*, nó sẽ di chuyển qua từng pixel trong hình ảnh và tính giá trị trung bình của tất cả các giá trị màu tại các pixel lân cận trong kernel. Sau đó, giá trị trung bình đó sẽ được gán cho pixel tại vị trí hiện tại. Điều này tạo ra hiệu ứng làm mờ, làm cho hình ảnh trở nên mịn hơn và giảm đi chi tiết. Độ lớn của kích thước kernel ((10, 10) trong trường hợp này) ảnh hưởng đến mức độ làm mờ. Kernel lớn hơn sẽ tạo ra hiệu ứng làm mờ mạnh hơn.[2]

```
In [7]: img = cv2.imread('path/to/your/image.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # convert BGR sang RGB
img_blur = cv2.GaussianBlur(img, (3,3), 0)
```



Hình 2.20: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép làm mịn Gaussian Blur.

### 2.5.2 Gaussian Blur:

Lệnh `cv2.GaussianBlur()` cũng có tác dụng làm mờ ảnh, phép làm mờ Gaussian sử dụng kernel Gaussian để làm mờ hình ảnh, tạo ra hiệu ứng làm mờ mượt, trong đó giá trị trung bình của các pixel xung quanh được tính dựa trên phân phối Gaussian. Lệnh này gồm có 6 tham số là:

- *src*: Đây là hình ảnh gốc mà bạn muốn thực hiện phép làm mờ Gaussian lên.
- *ksize*: Đây là kích thước của kernel Gaussian, xác định độ rộng của phân phối Gaussian. *ksize* là một tuple chứa hai giá trị (*kwidth*, *kheight*) để xác định chiều rộng và chiều cao của kernel. Kernel Gaussian sẽ có kích thước *kwidth* x *kheight*.
- *sigmaX*: Đây là tham số xác định độ phân tán (spread) của phân phối Gaussian theo chiều ngang. Giá trị này kiểm soát độ rộng của đỉnh của phân phối Gaussian. Giá trị càng lớn, đỉnh càng rộng.
- *sigmaY (tùy chọn)*: Đây là tham số tương tự như *sigmaX*, nhưng áp dụng theo chiều dọc. Nếu không được cung cấp, thì nó sẽ bằng *sigmaX*, tức là phân phối Gaussian là hình tròn.
- *dst (tùy chọn)*: Đây là hình ảnh đích (destination image) để lưu kết quả làm mờ. Nếu bạn không cung cấp giá trị này, kết quả sẽ được lưu trực tiếp vào hình ảnh gốc (*src*).

- *borderType* (*tùy chọn*): Cách xử lý biên giới của hình ảnh khi thực hiện phép làm mờ. Tương tự như cv2.blur(), bạn có thể sử dụng các tùy chọn như cv2.BORDER\_CONSTANT, cv2.BORDER\_REPLICATE, cv2.BORDER\_REFLECT, v.v.

Lệnh trong ví dụ dùng để thực hiện phép làm mờ Gaussian lên hình ảnh img bằng cách sử dụng một *kernel Gaussian* có kích thước (3, 3) và *sigmaX* được đặt là 0. Khi bạn áp dụng phép làm mờ Gaussian với kernel có kích thước (3, 3) và *sigmaX* được tính dựa trên kích thước kernel, nó sẽ sử dụng kernel này để làm mờ hình ảnh *img*. Kernel này sẽ di chuyển qua từng pixel trong hình ảnh và tính giá trị trung bình của các giá trị màu tại các pixel lân cận, dựa trên phân phối Gaussian. Kết quả là một hình ảnh được làm mờ, giúp giảm nhiễu và làm mờ chi tiết không mong muốn.

*cv2.GaussianBlur()* sử dụng kernel Gaussian để làm mờ. Kernel Gaussian sử dụng một hàm Gauss để tính giá trị trung bình của các pixel lân cận, với giá trị trung bình tăng dần từ trung tâm kernel đến ngoài, tạo ra hiệu ứng làm mờ mượt và tự nhiên. Cung cấp sự kiểm soát độ phân tán (*sigmaX*) của phân phối Gaussian, cho phép điều chỉnh mức độ làm mờ và chi tiết được bảo toàn. Thường được sử dụng để làm mờ hình ảnh một cách mượt mà và kiểm soát mức độ làm mờ.[2]

### 2.5.3 Median Blur:

```
In [7]: img = cv2.imread('path/to/your/image.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)           # convert BGR sang RGB
img_median = cv2.medianBlur(img,3)
```



Hình 2.21: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép làm mịn MedianBlur.

Lệnh `cv2.medianBlur()` trong thư viện OpenCV được sử dụng để thực hiện phép làm mờ trung vị (median blur) trên hình ảnh. Phép làm mờ trung vị sử dụng một kernel hình chữ nhật hoặc hình vuông để làm mờ hình ảnh bằng cách thay giá trị của mỗi pixel bằng giá trị trung vị của các pixel lân cận nằm trong kernel. Các tham số của hàm `cv2.medianBlur()`:

- *src*: Đây là hình ảnh gốc mà bạn muốn thực hiện phép làm mờ trung vị lên.
- *ksize*: Đây là kích thước của kernel trung vị, xác định chiều rộng và chiều cao của kernel hình chữ nhật hoặc hình vuông. Kernel này di chuyển qua từng pixel trong hình ảnh và tính giá trị trung vị của các pixel lân cận nằm trong kernel.

Ở ví dụ trên, phép làm mờ trung vị sẽ sử dụng một kernel hình chữ nhật có kích thước 3x3 pixel (kích thước này được xác định bởi tham số 3). Nó tính giá trị trung vị của các pixel trong kernel và gán giá trị đó cho pixel tại vị trí tương ứng. Điều này giúp làm mờ và giảm nhiễu trong hình ảnh.[2]

#### 2.5.4 *Bilateral Blur*:

```
In [7]: img = cv2.imread('path/to/your/image.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)      # convert BGR sang RGB
img_bila = cv2.bilateralFilter(img, 5, 180, 180)
```



Hình 2.22: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép làm mờ BilateralBlur.

`cv2.bilateralFilter()` là một hàm trong thư viện OpenCV được sử dụng để thực hiện phép làm mờ đối xứng (bilateral blur) trên hình ảnh. Đây là một phương pháp làm mờ cực kỳ mạnh mẽ và phổ biến để làm mờ hình ảnh trong thế giới xử lý ảnh. Phép làm

mờ đối xứng (bilateral blur) không chỉ xem xét giá trị màu tại các pixel xung quanh mục tiêu, mà còn xem xét sự tương quan giữa giá trị màu tại pixel đó và các pixel lân cận. Điều này có nghĩa là nó giữ lại các cạnh (biên giới) trong hình ảnh trong khi làm mờ các vùng màu đồng nhất. Hàm *cv2.bilateralFilter()* chấp nhận các tham số như sau:

- Hình ảnh gốc mà bạn muốn làm mờ.
- Kích thước của kernel, tức là chiều rộng của cửa sổ xem xét pixel xung quanh.
- Tham số sigmaColor: Đây là độ phân tán màu sắc, kiểm soát mức độ xem xét tương quan màu giữa các pixel. Giá trị cao cho phép nhiều màu sắc giữa các pixel được xem xét, trong khi giá trị thấp giữ lại các biên giới màu.
- Tham số sigmaSpace: Đây là độ phân tán không gian, kiểm soát mức độ xem xét tương quan vị trí giữa các pixel. Giá trị cao cho phép nhiều pixel ở xa nhau được xem xét, trong khi giá trị thấp chỉ xem xét các pixel gần.

Về ưu điểm, bilateral blur giữ lại cạnh và biên giới màu sắc rất tốt, do đó nó thường được sử dụng trong các ứng dụng cần làm mờ hình ảnh nhưng vẫn bảo toàn chi tiết quan trọng, chẳng hạn như xử lý hình ảnh y tế, làm mờ ảnh nghiệp ảnh hoặc làm sạch ảnh máy ảnh. Tuy nhiên nó cũng còn tồn tại những nhược điểm như là phép làm mờ đối xứng có thể tốn nhiều thời gian tính toán hơn so với các phương pháp làm mờ khác, vì nó xem xét cả màu sắc và không gian. Tùy thuộc vào kích thước hình ảnh và tham số, việc tính toán có thể trở nên chậm.

Trong ví dụ thì lệnh này sẽ áp dụng phép làm mờ đối xứng với kernel kích thước 5x5 và độ phân tán màu sắc và không gian là 180. Kết quả là hình ảnh sau khi đã làm mờ và chi tiết màu sắc và vị trí được bảo toàn tốt hơn.[2]

## 2.6 Phép biến đổi hình thái:

### 2.6.1 Đọc, tạo bộ lọc:

```
In [7]: img = cv2.imread('path/to/your/image.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img_gray, cmap = 'gray')
```

Đây là câu lệnh sử dụng thư viện OpenCV và Matplotlib để đọc một hình ảnh từ tệp, chuyển đổi nó thành ảnh xám, và sau đó hiển thị nó. Trong đó, từng câu lệnh được giải thích như sau:

- `img = cv2.imread('path/to/your/image.jpg')`: Dòng này đọc một hình ảnh từ đường dẫn đã chỉ định và lưu trữ nó trong biến img.
- `img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`: Màu sắc của hình ảnh được lưu trữ trong img được chuyển đổi thành ảnh xám bằng cách sử dụng hàm cvtColor của OpenCV và lưu trữ trong biến img\_gray.
- `plt.imshow(img_gray, cmap = 'gray')`: Sử dụng hàm imshow của Matplotlib để hiển thị hình ảnh xám với bản đồ màu xám.

```
In [2]: kernel = np.ones((5,5),np.uint8)
print(kernel)
```

Đây là một đoạn mã Python tạo ra một ma trận (kernel) 5x5 gồm toàn số 1 và sau đó in ma trận này ra. Mã này sử dụng thư viện NumPy của Python để tạo ra ma trận. Hàm `np.ones()` được sử dụng để tạo ra một mảng chứa toàn số 1, và (5,5) xác định kích thước của mảng. `np.uint8` đặt kiểu dữ liệu của các phần tử trong mảng thành số nguyên không dấu 8 bit. Kết quả thu được sau lệnh `print(kernel)` là một ma trận ma trận 5x5 với tất cả phần tử là số 1.

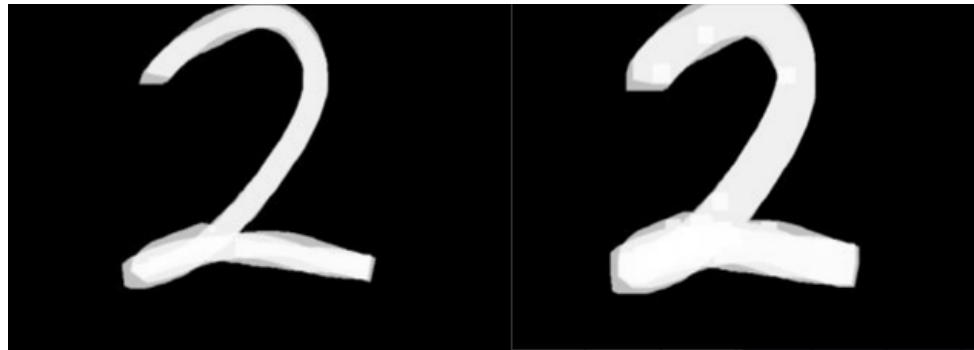
### 2.6.2 Giãn nở hình ảnh:

```
In [7]: dilation = cv2.dilate(img_gray,kernel,iterations = 5)
plt.imshow(dilation, cmap = 'gray')
```

Lệnh sử dụng thư viện OpenCV (cv2) để thực hiện phép toán dilation trên một ảnh màu xám (img\_gray). Dưới đây là phần giải thích chi tiết cho từng lệnh `cv2.dilate()`:

- `cv2.dilate`: Là hàm thực hiện phép toán dilation (giãn nở) trong xử lý ảnh. Phép toán này sẽ “mở rộng” các vùng trắng và “thu hẹp” các vùng đen trong ảnh màu xám.

- *img\_gray*: Là ảnh đầu vào cần thực hiện phép dilation. Ảnh này phải là ảnh màu xám.
- *kernel*: Là ma trận nhị phân dùng để quyết định “hình dạng” của phép dilation.
- *iterations = 5*: Là số lần lặp lại phép dilation. Số lần lặp càng nhiều thì vùng trắng trong ảnh sẽ càng được mở rộng.



Hình 2.23: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép giãn nở Dilation

Đây là hình ảnh thu được sau khi sử dụng phép giãn nở dilation, ta thu được một hình ảnh có phần màu trắng lớn hơn nhiều so với ảnh gốc.[2]

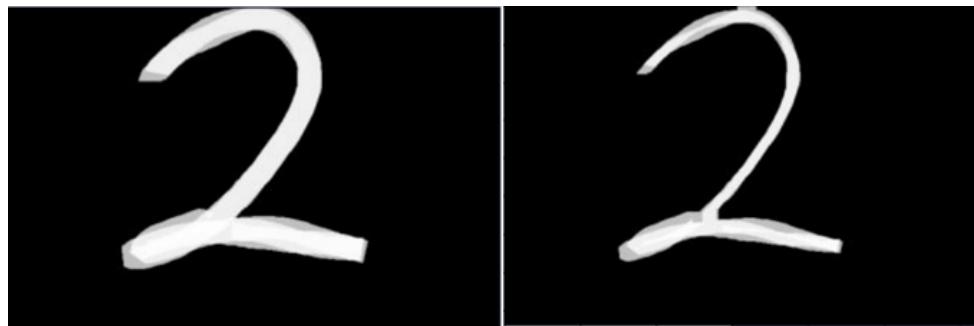
### 2.6.3 Co rút hình ảnh:

```
In [7]: erosion = cv2.erode(img_gray, kernel, iterations=3)
print(erosion.shape)
plt.imshow(erosion, cmap='gray')
```

Đoạn code trên sử dụng lệnh `cv2.erode()` để thực hiện một phép toán co lại trong xử lý. Dưới đây sẽ là phần giải thích chi tiết cho đoạn lệnh trên:

- *cv2.erode*: Là hàm thực hiện phép toán erosion (co lại) trong xử lý ảnh. Phép toán này sẽ “thu hẹp” các vùng trắng và “mở rộng” các vùng đen trong ảnh màu xám.
- *img\_gray*: Là ảnh đầu vào cần thực hiện phép erosion. Ảnh này phải là ảnh màu xám.
- *kernel*: Là ma trận nhị phân dùng để quyết định “hình dạng” của phép erosion.

- *iterations* = 3: Là số lần lặp lại phép erosion. Số lần lặp càng nhiều thì vùng đen trong ảnh sẽ càng được mở rộng.



Hình 2.24: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép co rút Erosion.

Hình ảnh thu được có phần trắng trở nên nhỏ, mảnh hơn so với hình ảnh gốc.[2]

#### 2.6.4 Phép mở:

```
In [7]: opening = cv2.morphologyEx(img_gray, cv2.MORPH_OPEN, kernel)
plt.imshow(opening, cmap='gray')
```

Đoạn code trên sử dụng phép biến đổi hình thái opening trên một hình ảnh xám. Dưới đây là giải thích cho đoạn code trên:

- *cv2.morphologyEx*: Là hàm thực hiện các phép toán morphological transformations. Trong trường hợp này, nó thực hiện phép morphological opening.
- *img\_gray*: Là ảnh đầu vào cần thực hiện phép morphological opening. Ảnh này phải là ảnh màu xám.
- *cv2.MORPH\_OPEN*: Là tham số chỉ định loại phép toán morphological transformations. cv2.MORPH\_OPEN chỉ định phép morphological opening, một phép toán giúp loại bỏ nhiễu trong ảnh.
- *kernel*: Là ma trận nhị phân dùng để quyết định “hình dạng” của phép morphological opening.

Phép biến đổi hình thái mở (opening) không được coi là thuật ngữ tiêu chuẩn trong xử lý ảnh. Tuy nhiên, ta có thể hiểu thông qua các phép biến đổi cơ bản. Với trường hợp

phép mở, thì đây là sự kết hợp của 2 phép biến đổi hình thái cơ bản đã nói ở trên là phép giãn nở và co rút, trong đó ta thực hiện co rút trước rồi sẽ thực hiện phép giãn nở. Phép mở thường được sử dụng để loại bỏ các nhiễu ngoài đường bao vật thể và chia tách các vật thể gần với nhau. Phép co rút được sử dụng trước nhằm thu nhỏ các vật thể và loại bỏ các phần nhỏ, sau đó sẽ sử dụng phép giãn nở để mở rộng các vật thể trở lại những các nhiễu và chi tiết nhỏ đã bị loại bỏ.



Hình 2.25: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép mở Opening.

Như ví dụ trên, sau khi áp dụng phép mở, ta thu được lại một hình ảnh không còn các điểm nhỏ màu trắng nữa.[2]

#### 2.6.5 Phép đóng:

```
In [7]: closing = cv2.morphologyEx(img_gray, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing, cmap='gray')
```

Đây là một đoạn lệnh thực hiện phép đóng (Closing) trên một ảnh xám. Đây là phần giải thích cho đoạn lệnh này:

- *cv2.morphologyEx*: Là hàm thực hiện các phép toán morphological transformations. Trong trường hợp này, nó thực hiện phép morphological closing.

- *img\_gray*: Là ảnh đầu vào cần thực hiện phép morphological closing. Ảnh này phải là ảnh màu xám.
- *cv2.MORPH\_CLOSE*: Là tham số chỉ định loại phép toán morphological transformations. cv2.MORPH\_CLOSE chỉ định phép morphological closing, một phép toán giúp loại bỏ nhiễu trong ảnh.
- *kernel*: Là ma trận nhị phân dùng để quyết định “hình dạng” của phép morphological closing.

Tương tự như phép mở, phép đóng không được coi là một thuật ngữ tiêu chuẩn trong việc xử lý ảnh. Tuy nhiên, ta cũng có thể tìm hiểu phép đóng thông qua các phép biến đổi cơ bản. Phép đóng cũng là sự kết hợp của 2 phép biến đổi hình thái cơ bản đã nói ở trên là phép giãn nở và co rút, trong đó ta thực hiện giãn nở trước rồi sẽ thực hiện phép co rút. Phép đóng được sử dụng khi ta muốn đóng các lỗ nhỏ trong vật thể và các vật thể bị tách nhau một chút. Phép giãn nở được áp dụng để mở rộng các vật thể và lắp đầy các lỗ nhỏ, sau đó phép co rút được áp dụng để thu nhỏ các vật thể trong khi vẫn giữ nguyên hình dạng tổng thể.



Hình 2.26: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép đóng Closing.

Như ví dụ trên, ta thấy hình ảnh thu được đã không còn các điểm nhiễu màu đen ở trong vật thể và không có khoảng trống trong vật thể.[2]

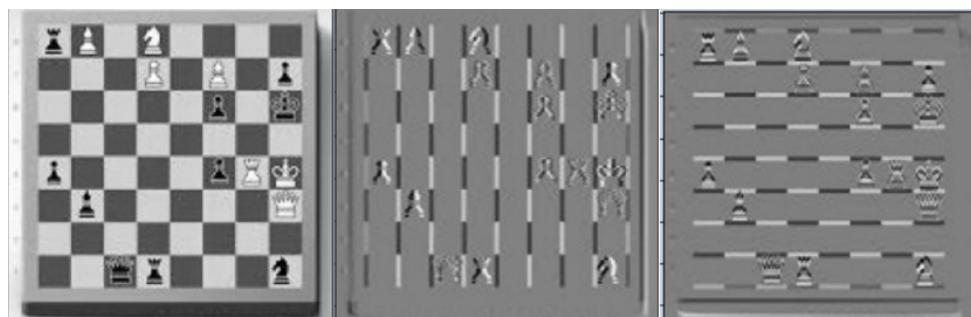
## 2.7 Đạo hàm ảnh:

Trong ngữ cảnh xử lý ảnh, đạo hàm hay gradient ảnh là sự thay đổi hướng trong cường độ ảnh hoặc màu sắc một ảnh. Gradient của ảnh là một trong những khối xây dựng cơ bản trong xử lý ảnh. Chúng ta có thể sử dụng đạo hàm để nhận diện cạnh trong hình ảnh, việc này giúp ta xác định được các đường bao và rìa của vật thể trong ảnh.

### 2.7.1 Đạo hàm theo x và y:

```
In [ ]: sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=1)
plt.show(sobelx, cmap = 'gray')
```

```
In [ ]: sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=1)
plt.show(sobely, cmap = 'gray')
```



Hình 2.27: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép đạo hàm Sobel.

Hai dòng code trên sử dụng hàm `cv2.Sobel()` để thực hiện phép lọc trên hình ảnh xám `img`. Dưới đây là giải thích về các tham số trong hàm này:

- *img*: Đây là hình ảnh xám mà bạn muốn áp dụng phép lọc Sobel lên.
- *cv2.CV\_64F*: Đây là kiểu dữ liệu của ảnh kết quả, trong trường hợp này là float64 (64-bit floating-point). Sobel thường tạo ra các giá trị gradient là số thực, do đó chúng được lưu trữ trong một ma trận số thực để bảo toàn thông tin.

- $(1, 0)$  hay  $(0, 1)$ : Đây là tham số của hàm Sobel, xác định hướng của đạo hàm theo trục x (ngang) và y (dọc). Trong trường hợp  $(1, 0)$ , 1 cho biết đạo hàm theo trục x, và 0 cho biết không có đạo hàm theo trục y. Tương tự trường hợp  $(0, 1)$  cho đạo hàm theo trục y và không có đạo hàm trên trục x.
- $ksize=1$ : Đây là kích thước của kernel được sử dụng trong phép lọc. Trong trường hợp này, kernel có kích thước là  $1 \times 1$ . Kích thước của kernel ảnh hưởng đến cách Sobel đo lường độ lớn của gradient. Kích thước lớn hơn có thể tạo ra một hiệu ứng làm mờ độ lớn gradient.

Phía bên trái của hình ảnh này cho thấy một bức ảnh xám của một bàn cờ với các quân cờ khác nhau được đặt trên đó. Các quân cờ có thể phân biệt rõ ràng với các hình dạng khác nhau đại diện cho các loại khác nhau như vua, hậu, tượng, v.v. Hai hình còn lại hiển thị một phiên bản phát hiện cạnh của bàn cờ đó sử dụng phương pháp lọc Sobel, làm nổi bật các gradient cường độ hoặc cạnh trong bức ảnh xám gốc đó. Trên phiên bản phát hiện cạnh này, các đường ngang hoặc dọc nổi bật cho thấy nơi có sự thay đổi đáng kể về cường độ trên bức ảnh gốc.[2]

### 2.7.2 Đạo hàm ảnh có trọng số:

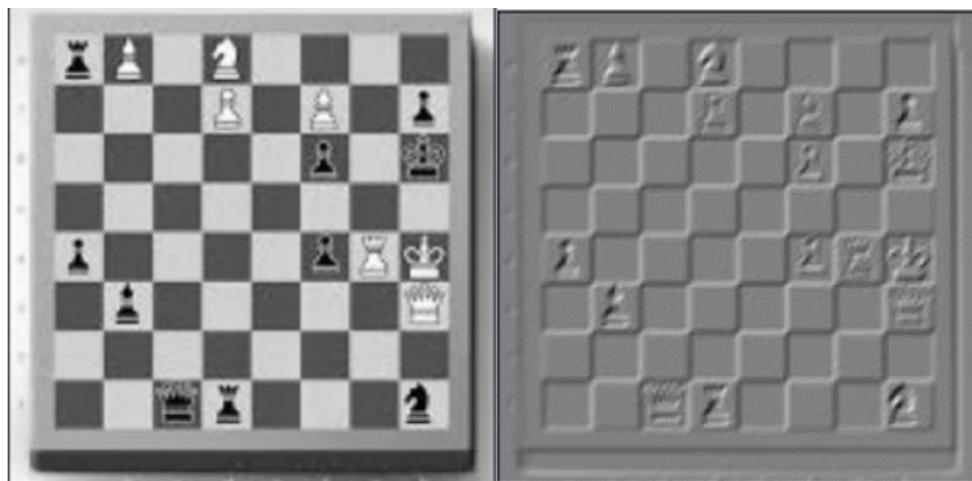
```
In [ ]: dst = cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)
plt.show(dst, cmap = 'gray')
```

Dòng mã `dst = cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)` sử dụng hàm `addWeighted` trong OpenCV để kết hợp hai ảnh đã được xử lý bằng phép lọc Sobel theo hướng ngang (`sobelx`) và theo hướng dọc (`sobely`). Dưới đây là giải thích về dòng mã này và các tham số của hàm:

- `sobelx`: Đây là ảnh kết quả sau khi áp dụng phép lọc Sobel theo hướng ngang. Đây thường là ảnh chứa thông tin về gradient theo trục ngang của hình ảnh.
- `0.5`: Đây là trọng số được áp dụng cho ảnh `sobelx` trong quá trình kết hợp. Trọng số này xác định độ quan trọng của đạo hàm theo trục ngang trong ảnh kết quả cuối cùng.

- *sobely*: Đây là ảnh kết quả sau khi áp dụng phép lọc Sobel theo hướng dọc. Đây thường là ảnh chứa thông tin về gradient theo trục dọc của hình ảnh.
- *0.5*: Đây là trọng số được áp dụng cho ảnh *sobely* trong quá trình kết hợp. Trọng số này xác định độ quan trọng của đạo hàm theo trục dọc trong ảnh kết quả cuối cùng.
- *0*: Đây là giá trị thêm vào ảnh cuối cùng sau khi kết hợp. Trong trường hợp này, không có giá trị được thêm vào.

Kết quả của dòng mã này là ảnh *dst*, là sự kết hợp tuyến tính của hai ảnh *sobelx* và *sobely* với trọng số tương ứng. Điều này giúp tạo ra một ảnh kết quả cuối cùng có thông tin gradient từ cả hai hướng, thường được sử dụng để làm nổi bật các đặc trưng và biên trong hình ảnh.[2]



Hình 2.28: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép đạo hàm có trọng số.

### 2.7.3 Đạo hàm Laplacian:

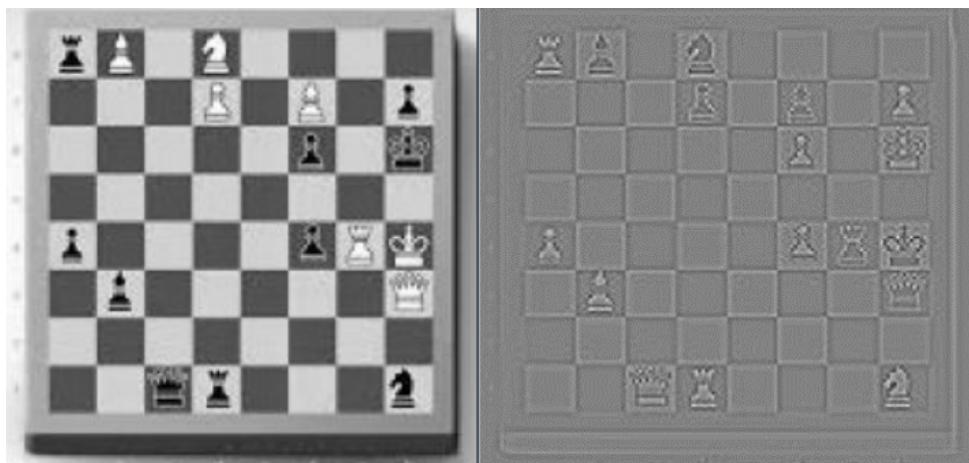
```
In [ ]: laplacian = cv2.Laplacian(img, cv2.CV_64F)
plt.show(laplacian, cmap = 'gray')
```

Dòng mã *laplacian = cv2.Laplacian(img, cv2.CV\_64F)* sử dụng hàm *cv2.Laplacian* trong OpenCV để áp dụng phép lọc Laplacian lên hình ảnh xám *img*. Dưới đây là giải thích về dòng mã này và các tham số của hàm:

- *img*: Đây là hình ảnh xám mà bạn muốn áp dụng phép lọc Laplacian lên.
- *cv2.CV\_64F*: Đây là kiểu dữ liệu của ảnh kết quả, trong trường hợp này là float64 (64-bit floating-point). Phép lọc Laplacian thường tạo ra các giá trị gradient là số thực, do đó chúng được lưu trữ trong một ma trận số thực để bảo toàn thông tin.

Kết quả của dòng mã này là *laplacian*, là ảnh chứa thông tin về gradient Laplacian của hình ảnh gốc. Phép lọc Laplacian giúp làm nổi bật các vùng có sự thay đổi cường độ đột ngột trong hình ảnh, thường được sử dụng để phát hiện các điểm cực trị và các đặc trưng cạnh. Tóm lại, dòng mã này thực hiện phép lọc Laplacian trên hình ảnh xám và tạo ra một ảnh mới chứa thông tin về độ brusque của hình ảnh, giúp làm nổi bật các đặc trưng và cạnh trong hình ảnh.

Độ brusque hay độ đột ngột của một hình ảnh thường được liên kết với sự thay đổi cường độ đột ngột tại các vùng cạnh hoặc biên giữa các vùng khác nhau trong hình ảnh. Trong ngữ cảnh của phép lọc Laplacian mà bạn đang đề cập, độ đột ngột này thường được đo lường bằng gradient của hình ảnh. Nói chung, độ đột ngột của hình ảnh thể hiện mức độ thay đổi cường độ giữa các điểm ảnh gần nhau. Các vùng có độ đột ngột lớn thường là những vùng chứa biên, cạnh, hoặc các đối tượng quan trọng trong hình ảnh.



Hình 2.29: Hình ảnh gốc và hình ảnh thu được sau khi sử dụng phép đạo hàm Laplacian.

## 2.8 Làm sắc nét ảnh:

Phép làm sắc nét ảnh (Image Sharpening) là quá trình tăng cường độ tương phản giữa các điểm ảnh liền kề nhau để làm cho các cạnh và chi tiết trong ảnh trở nên rõ ràng hơn. Có nhiều cách làm sắc nét một hình ảnh, tùy thuộc vào loại hình ảnh và các yếu tố khác của ảnh mà ta có thể chọn một phương pháp để có được hiệu quả cao nhất. Về cơ bản, để làm sắc nét một hình ảnh, ta cần xác định được các đường giới hạn, đường bao của vật thể và phân biệt được chúng với tổng thể ảnh. Để có thể làm được như vậy chúng ta có một vài cách làm như sau:

- *Phương Pháp Unsharp Masking*: Đây là một phương pháp phổ biến để làm sắc nét ảnh. Nó hoạt động bằng cách tạo ra một bản sao mờ của ảnh gốc, sau đó trừ đi bản sao mờ từ ảnh gốc để tạo ra một “mask”. Mask này sau đó được cộng vào ảnh gốc để tạo ra ảnh sắc nét.
- *Sử Dụng Phương Pháp Wiener Deconvolution*: Nếu ảnh bị nhòe do quá trình mất mát thông tin trong quá trình chụp hình, có thể sử dụng phương pháp Wiener deconvolution để khôi phục thông tin bị mất và làm sắc nét ảnh.
- *Tăng Cường Độ Tương Phản*: Tăng cường độ tương phản của hình ảnh có thể làm nổi bật sự chênh lệch giữa các vùng cường độ, làm tăng sự sắc nét.
- *Áp Dụng Phương Pháp Super-Resolution*: Trong một số trường hợp, có thể sử dụng phương pháp super-resolution để nâng cao độ phân giải của hình ảnh và làm sắc nét hơn.
- *High Pass Filtering*: Phương pháp này hoạt động bằng cách loại bỏ các thành phần tần số thấp (mà thường tương ứng với các vùng mờ) từ ảnh. Kết quả là ảnh chỉ giữ lại các thành phần tần số cao, tạo ra ảnh sắc nét hơn.
- *Edge Enhancement*: Phương pháp này hoạt động bằng cách tăng cường độ tương phản giữa các cạnh trong ảnh, thường là bằng cách sử dụng các bộ lọc như Sobel hoặc Laplacian.
- *Học Máy và Học Sâu*: Các phương pháp dựa trên học máy và học sâu, như mạng nơ-ron tích chập (CNNs), có thể được huấn luyện để thực hiện nhiệm vụ làm sắc

nét ảnh. Các mô hình này có thể học cách “hiểu” cấu trúc và nội dung của ảnh, và do đó có thể tạo ra kết quả tốt hơn so với các phương pháp truyền thống.

Các phương pháp trên có thể được lựa chọn tùy vào loại nhiễu và không có một phương pháp nào là hoàn hảo. Có thể là với một ảnh phương pháp này là hiệu quả tuy nhiên không phải nó cũng sẽ hiểu quả với những ảnh khác. Dưới đây là một ví dụ làm sắc nét ảnh sử dụng phương pháp phổ biến nhất là Unsharp Masking:

```
In [ ]: img = cv2.imread("path/to/your/image.jpg")
img_gray = cv2.cvtColor(img, BGR2GRAY)

plt.figure(figsize = (10, 10))
plt.imshow(img_gray, cmap = 'gray')
```

Đây là đoạn code giúp đọc dữ liệu và chuyển ảnh từ ảnh màu sang ánh xám.



Hình 2.30: Hình ảnh cần làm nét..

```
In [ ]: img.blur = cv2.blur(img, (5, 5))
plt.figure(figsize = (10, 10))
plt.imshow(img.blur, cmap = 'gray')
```

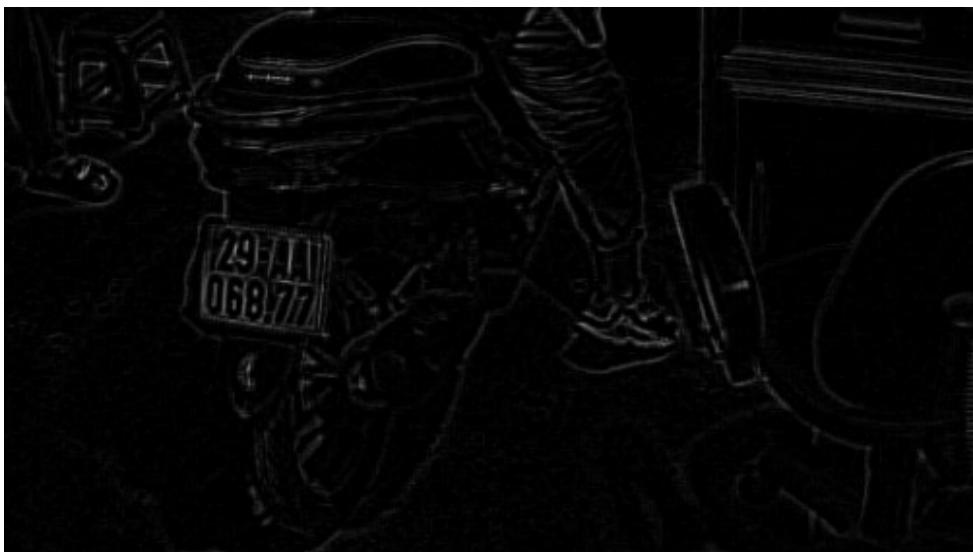
Sau khi chuyển ảnh sang ánh xám thì ta sẽ làm mờ ảnh bằng lệnh `cv2.blur()`.

Bước kế tiếp, ta thực hiện phép trừ để tìm ra sự khác biệt giữa ảnh gốc với ảnh được làm mờ chung làm nổi bật hơn các chi tiết có nhiều sự thay đổi cường độ. Hàm được sử dụng ở đây là hàm `cv2.subtract()`



Hình 2.31: Hình ảnh gốc đã được làm mịn.

```
In [ ]: diff = cv2.subtract(img_gray, img_blur)
plt.figure(figsize = (10, 10))
plt.imshow(diff, cmap = 'gray')
```



Hình 2.32: Hình ảnh thu được sau khi thực hiện phép trừ.

```
In [ ]: final = cv2.addWeighted(img_gray, 0.1, diff, 0.9)
plt.figure(figsize = (10, 10))
plt.imshow(final, cmap = 'gray')
```

Cuối cùng, ta kết hợp hai ảnh lại với điều kiện ảnh gốc giữ trọng số nhỏ trong khi ảnh lưu trữ những sự riêng biệt của các chi tiết có trọng số lớn hơn. Việc này giúp các chi tiết trở nên sắc nét hơn.



Hình 2.33: Hình ảnh cuối.

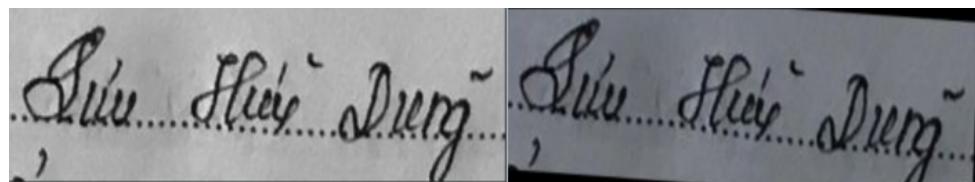
## 2.9 Quay hình:

```
In [ ]: (h,w)=img.shape[:2]
(cx,cy)=(w//2,h//2)
M=cv2.getRotationMatrix2D((cx,cy),randint(-5,5),1.0)
rotated=cv2.warpAffine(img,M,(w,h))
plt.imshow(rotated)
```

Đoạn code trên dùng để xoay hình một cách ngẫu nhiên trong phạm vi từ -5 đến 5 độ. Sau đây là phần giải thích cho từng đoạn code:

- $(h,w)=img.shape[:2]$ : Dòng này trích xuất chiều cao và chiều rộng của hình ảnh img.
- $(cx,cy)=(w//2,h//2)$ : Dòng này tính toán tọa độ trung tâm của hình ảnh.
- $M=cv2.getRotationMatrix2D((cx,cy),randint(-5,5),1.0)$ : Dòng này tạo ra một ma trận xoay với một góc ngẫu nhiên từ -5 đến 5 độ xung quanh trung tâm của hình ảnh.
- $rotated=cv2.warpAffine(img,M,(w,h))$ : Dòng này áp dụng biến đổi affine để xoay hình ảnh bằng cách sử dụng ma trận xoay đã tạo.

Tùy thuộc vào yêu cầu bài toán mà ta lựa chọn góc độ xoay hình phụ hợp.[2]



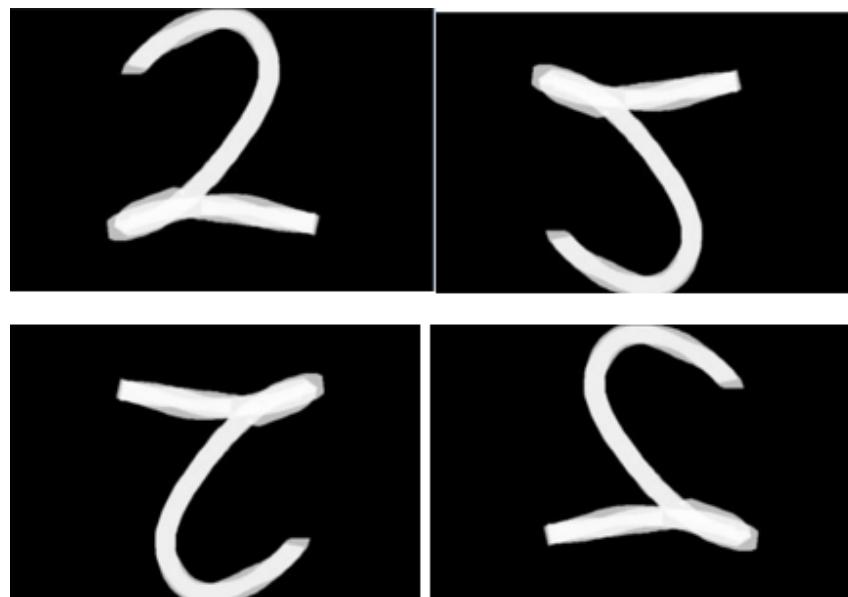
Hình 2.34: Hình ảnh lúc đầu và hình ảnh thu được sau khi xoay.

## 2.10 Lật ảnh:

```
In [ ]: a = cv2.flip(img, 1)  
plt.imshow(a)
```

Để lật một hình ảnh, ta có lệnh `cv2.flip()` của thư viện OpenCV để thực hiện tác vụ này. Dưới đây là giải thích lệnh và các tham số của lệnh này:[2]

- *img*: Đây là hình ảnh cần lật.
- *flipCode*: Một cờ để chỉ định cách lật mảng: 0 để lật hình ảnh theo trục x (tức là lật theo chiều dọc). Số dương (ví dụ: 1) để lật hình ảnh theo trục y (tức là lật theo chiều ngang). Số âm (ví dụ: -1) để lật hình ảnh cả hai trục x và y.
- *dst (tùy chọn)*: Mảng đầu ra có cùng kích thước và loại như *img*.



Hình 2.35: Hình ảnh lúc đầu và hình ảnh được lật với các *flipCode* khác nhau.

## 2.11 Nhận diện các phần của vật thể:

### 2.11.1 Nhận diện góc:

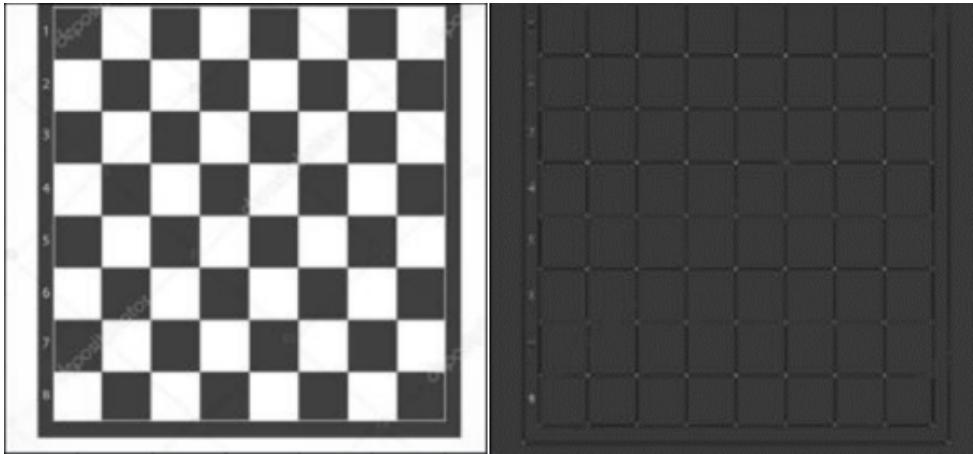
Nhận diện góc là một trong những công việc quan trọng trong xử lý ảnh và thị giác máy tính vì nó mang lại nhiều lợi ích cho các công việc sau này. Việc nhận diện này có thể được sử dụng để nhận diện đối tượng, định vị một đối tượng, điều chỉnh vật thể trong không gian,... Dưới đây là cách để có thể nhận diện góc sử dụng thư viện OpenCV:

```
In [18]: corner = cv2.cornerHarris(src=gray, blockSize=3, ksize=3, k=0.06)
plt.imshow(corner, cmap='gray')
```

Lệnh trên sử dụng hàm *cv2.cornerHarris()* của thư viện OpenCV để tìm ra các góc Harris của ảnh xám gray. Lệnh này yêu cầu một số tham số là:

- *gray*: Đây là hình ảnh xám mà bạn muốn phát hiện các góc trong đó.
- *blockSize=3*: Đây là kích thước của cửa sổ (window) được sử dụng để tính đạo hàm và xác định các vùng có sự thay đổi cường độ đột ngột. Trong trường hợp này, cửa sổ có kích thước 3x3.
- *ksize=3*: Đây là kích thước của kernel Sobel được sử dụng để tính đạo hàm tại mỗi điểm ảnh. Kích thước này ảnh hưởng đến độ nhạy của thuật toán đối với các biên cạnh và góc. Trong trường hợp này, kernel Sobel có kích thước 3x3.
- *k=0.04*: Đây là tham số k trong công thức tính ma trận Harris. Tham số này ảnh hưởng đến độ nhạy của thuật toán đối với sự thay đổi cường độ. Giá trị thường được lựa chọn trong khoảng 0.04 đến 0.06.

Kết quả của dòng mã này là *corner*, là một ảnh chứa các giá trị phản hồi Harris tại mỗi điểm ảnh. Các giá trị cao của *corner* thường tương ứng với các vùng chứa góc trong hình ảnh. Thuật toán Harris Corner Detection được sử dụng để phát hiện các điểm góc quan trọng trong hình ảnh.[2]



Hình 2.36: Hình ảnh xám và hình ảnh chứa các góc Harris.

### 2.11.2 Nhận diện cạnh:

Nhận diện cạnh là một trong những lĩnh vực quan trọng của xử lý ảnh. Để có thể nhận diện cạnh trong một ảnh, thư viện OpenCV cung cấp cho ta câu lệnh cv2.Canny(). Phương pháp nhận diện cạnh này trả về một hình ảnh đen trắng, nơi các pixel trắng chỉ ra một phần của cạnh.

```
In [18]: edges = cv2.Canny(img, 100,200)
plt.imshow(edges, cmap='gray')
```

Dưới đây là mô tả về các tham số có trong lệnh *cv2.Canny()*:

- *img*: Ảnh đầu vào. Hàm sẽ thực hiện phát hiện cạnh trên ảnh này.
- *threshold1*: Nguồn dưới, là một giá trị để xác định cạnh. Nếu gradient cường độ tại một điểm nào đó vượt qua ngưỡng này, nó được coi là cạnh có thể.
- *threshold2*: Ngưỡng trên, giá trị này được sử dụng để xác định cạnh chắc chắn. Nếu gradient cường độ vượt qua ngưỡng này, nó được coi là cạnh chắc chắn.
- *edges (tùy chọn)*: Mảng NumPy kết quả chứa các pixel biên (cạnh) sau khi phát hiện.
- *apertureSize (tùy chọn)*: Kích thước của bộ lọc Sobel được sử dụng để tính gradient. Mặc định là 3.

- *L2gradient* (tùy chọn): Nếu True, sử dụng  $L_2$  norm (Euclidean norm) để tính toán gradient. Nếu False, sử dụng  $L_1$  norm. Mặc định là False.



Hình 2.37: Hình ảnh gốc và hình ảnh thu được sau lệnh Canny.

Có thể kết hợp với một số lệnh khác để thu được kết quả như mong muốn. Như ví dụ trên, nếu chúng ta không muốn có nhiều chấm trắng bên trong vật thể, ta có thể làm mờ ảnh ban đầu làm cho giá trị các pixel ở vị trí đó giảm đi từ đó sẽ thu được một hình ảnh chỉ bao gồm các cạnh chính của vật thể.[2]

### 2.11.3 Nhận diện đường bao:

Để nhận diện đường bao trong ảnh, ta sẽ sử dụng lệnh *cv2.findContour()*. Hàm này trả về 2 giá trị là:

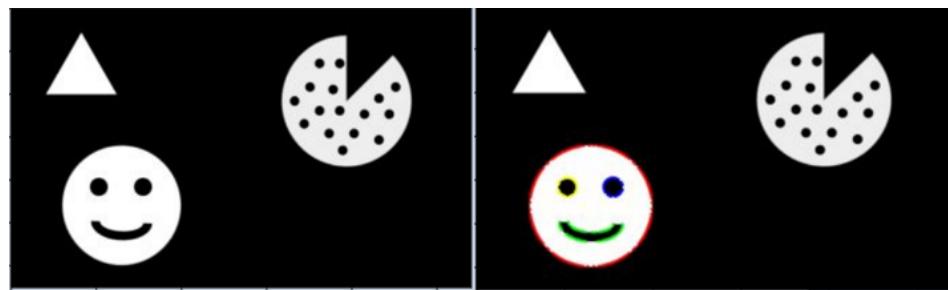
- *contours*: Danh sách các contours được tìm thấy. Mỗi contour là một mảng các điểm (x, y).
- *hierarchy*: Cấu trúc cây hierarchy của contours. Nó mô tả quan hệ giữa các contours (cha - con).

```
test = img.copy()
cv2.polylines(test, contour[0], isClosed = True, color = [255,0,0], thickness = 5)
cv2.polylines(test, contour[1], isClosed = True, color = [0,255,0], thickness = 5)
cv2.polylines(test, contour[2], isClosed = True,color = [0 ,0 ,255] ,thickness=5)
cv2.polylines(test ,contour[3],isClosed=True,color=[255 ,255 ,0] ,thickness=5)
plt.imshow(test)
```

Để hiện thị các đường bao này, ta cần vẽ những đường bao này lên hình ảnh. Phía đây là một đoạn mã giúp làm và phần giải thích cho các lệnh ở trên:

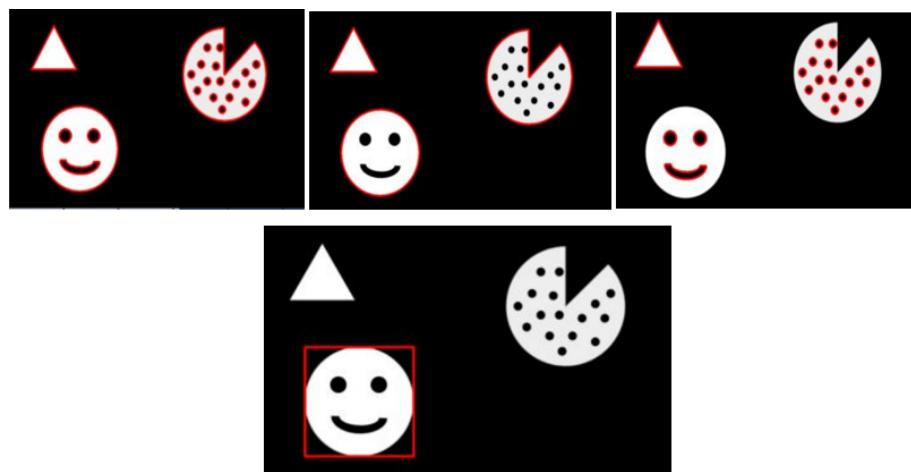
- *test = img.copy()*: Tạo một bản sao của ảnh gốc để vẽ contours lên đó mà không làm thay đổi ảnh gốc.

- `cv2.polyline()`: Vẽ đường đa giác nối các điểm trong contours.
- `contour[0], contour[1], contour[2], contour[3]`: Các contours cụ thể được chọn để vẽ. Đây có thể là các contours tìm được từ `cv2.findContours()` (thường được sắp xếp theo diện tích, vùng lớn đến vùng nhỏ).
- `isClosed=True`: Chọn True nếu bạn muốn vẽ một đa giác đóng (vòng tròn, hình vuông), False nếu bạn chỉ muốn vẽ các đoạn thẳng mở.
- `color`: Màu của đường đa giác (ở đây, là một list chứa giá trị RGB).
- `thickness`: Độ dày của đường vẽ.



Hình 2.38: Hình ảnh gốc và hình ảnh thu được sau lệnh `findContour()`.

Với hai thông số thu được từ lệnh `cv2.findContour()`, ta có thể yêu cầu OpenCV làm nhiều yêu cầu như vẽ đường bao cha, đường bao con, đường bao thứ bao nhiêu thông qua việc điều chỉnh hai tham số này.[2]



Hình 2.39: Một số tác vụ khác với Contour và Hierarchy.

## Chương 3 Diffusion Model

Diffusion là các mô hình tự sinh đã trở nên phổ biến đáng kể trong vài năm qua và vì lý do chính đáng. Chỉ riêng một số bài báo chuyên đề được phát hành trong những năm 2020 đã cho thấy khả năng của các mô hình Khuếch tán, chẳng hạn như đánh bại GAN về tổng hợp hình ảnh. Gần đây nhất, chúng ta sẽ thấy Diffusion Model được sử dụng trong DALL-E 2, mô hình tạo hình ảnh của OpenAI.

### 3.1 Tổng quan về hoạt động của mô hình:

Diffusion Model là mô hình tự sinh, nghĩa là chúng được sử dụng để tạo ra dữ liệu tương tự với dữ liệu mà chúng được đào tạo. Về cơ bản, Diffusion Model hoạt động bằng cách biến dữ liệu huấn luyện thành nhiều thông qua việc bổ sung liên tiếp nhiều Gaussian, sau đó học cách khôi phục dữ liệu bằng cách đảo ngược quá trình này. Sau khi đào tạo, chúng ta có thể sử dụng Diffusion Model để tạo dữ liệu bằng cách chuyển nhiều được lấy mẫu ngẫu nhiên thông qua quá trình khử nhiễu đã học.[3]

Mục đích chính của mô hình khuếch tán trong học máy là tìm hiểu quy trình khuếch tán tạo ra phân bố xác suất của một tập dữ liệu nhất định. Chúng được sử dụng để tìm hiểu cấu trúc tiềm ẩn của tập dữ liệu bằng cách mô hình hóa cách các điểm dữ liệu khuếch tán trong không gian tiềm ẩn của chúng.

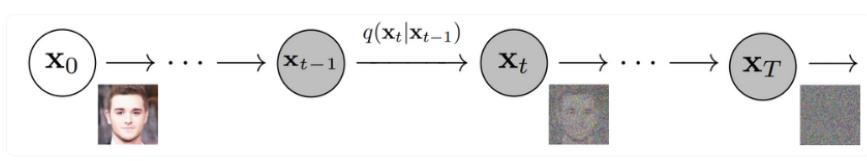
Chúng bao gồm hai thành phần chính: quy trình chuyển tiếp(Forward Process), quy trình ngược(Reverse Process)

#### 3.1.1 Forward Process:

Quy trình chuyển tiếp hay Forward Process trong Diffusion Model còn được gọi là quá trình khuếch tán(Diffusion Process) được định nghĩa là quá trình thêm các nhiễu Gaussian vào hình ảnh. Các nhiễu Gaussian được thêm từng lượng nhỏ vào ảnh trong một số bước thời gian nhất định. Trong mỗi bước thời gian có phương sai nhất định. [3]

Đây là một giải thích đơn giản các hoạt động của Forward Process:

- *Bắt đầu với ảnh:* Quá trình này xuất phát từ một ảnh sạch.
- *Thêm nhiều tuần hoàn:* Mô hình lấy hình ảnh này và bắt đầu lặp đi lặp lại thêm nhiều Gaussian vào nó. Ở mỗi bước, mô hình đưa ra dự đoán về hình ảnh bị nhiễu sẽ trông như thế nào dựa trên trạng thái hiện tại của nó.
- *Kết thúc là nhiễu:* Sau nhiều lần thêm nhiễu, hình ảnh đã được chuyển hoàn toàn thành nhiễu Gaussian thuần túy.



Hình 3.1: Forward Process.

Forward Process là một phần quan trọng trong hoạt động của Diffusion Model, nhưng đây không phải là phần được học trong quá trình huấn luyện mô hình.

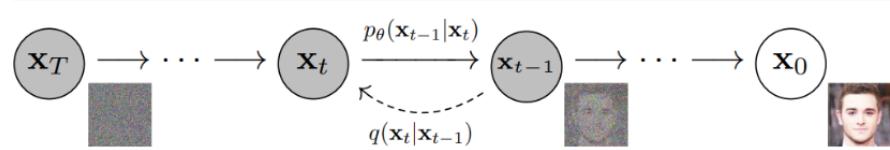
### 3.1.2 Reverse Process:

Quá trình ngược hay Reverse Process trong Diffusion Model còn được gọi là quá trình khử nhiễu hay quá trình khuếch tán ngược, đây là phần mô hình học trong quá trình huấn luyện. Ngược lại với Forward Process, quá trình này gỡ nhiễu từ bức ảnh thuần nhiễu mà Forward Process tạo ra. [3]

Dưới đây là một giải thích cơ bản cách hoạt động của quá trình ngược:

- *Bắt đầu là nhiễu:* Quá trình bắt đầu với một hình ảnh nhiễu, có thể là một hình ảnh đã được thêm nhiễu trong quá trình chuyển tiếp hoặc có thể là một hình ảnh nhiễu mới mà mô hình chưa từng thấy trước đây.
- *Khử nhiễu lặp lại:* Mô hình khuếch tán được đào tạo sau đó lấy hình ảnh nhiễu này và bắt đầu khử nhiễu lặp đi lặp lại. Ở mỗi bước, mô hình đưa ra dự đoán về hình ảnh được khử nhiễu sẽ trông như thế nào dựa trên trạng thái hiện tại của nó.

- *Kết thúc là hình ảnh sạch:* Sau nhiều lần lặp lại, nhiễu đã được loại bỏ hoàn toàn và những gì còn lại là một hình ảnh sạch sẽ. Hình ảnh này phải giống với loại hình ảnh mà người mẫu đã được đào tạo.



Hình 3.2: Reverse Process.

Quá trình ngược là một phần quan trọng trong Diffusion Model. Đây là phần cho phép mô hình tạo ra những dữ liệu mới bằng cách khôi phục lại dữ liệu gốc từ phiên bản nhiễu của nó. Reverse Process được học trong quá trình đào tạo, trong đó mô hình học cách ánh xạ từ hình ảnh nhiễu trở lại hình ảnh sạch.

### 3.2 Mục đích sử dụng:

Trong bối cảnh thị giác máy tính, các mô hình khuếch tán có thể được áp dụng cho nhiều nhiệm vụ khác nhau, bao gồm khử nhiễu hình ảnh, inpainting, siêu phân giải và tạo hình ảnh. Các mô hình khuếch tán thường được xây dựng dưới dạng chuỗi Markov và được huấn luyện bằng cách sử dụng suy luận biến phân. Ví dụ về các khung Diffusion Model chung được sử dụng trong thị giác máy tính là mô hình xác suất khuếch tán khử nhiễu, mạng điểm có điều kiện nhiễu và phương trình vi phân ngẫu nhiên. Trong đó, ba lĩnh vực nổi bật nhất trong thị giác máy tính của Diffusion Model là giảm nhiễu, tạo ảnh mới và tăng độ phân giải ảnh.

#### 3.2.1 Giảm nhiễu:

Denoising Diffusion Probabilistic Models (DDPM) là một loại mô hình khuếch tán được sử dụng để khử nhiễu hình ảnh. Chúng hoạt động bằng cách khuếch tán dần dần tiếng ồn có trong ảnh gốc. Dưới đây là giải thích từng bước về cách hoạt động của DDPM:

- *Forward Process:* Mô hình bắt đầu với một hình ảnh rõ ràng và thêm nhiễu Gaussian ở mỗi bước thời gian, dần dần chuyển hình ảnh thành nhiễu thuần túy.

- *Reverse Process*: Mô hình học cách đảo ngược forward process, bắt đầu từ hình ảnh nhiễu và loại bỏ dần nhiễu để khôi phục hình ảnh sạch ban đầu. Điều này được thực hiện bằng cách huấn luyện mạng lưới thần kinh để dự đoán hình ảnh sạch ở bước thời gian tiếp theo với hình ảnh nhiễu hiện tại.
- *Mục tiêu đào tạo & Loss Function*: Mô hình được đào tạo bằng cách sử dụng loss function nhằm khuyến khích hình ảnh sạch được dự đoán gần với hình ảnh sạch thực tế. Điều này thường được thực hiện bằng cách sử dụng một dạng suy luận biến phân.
- *Khử nhiễu*: Sau khi mô hình được huấn luyện, nó có thể được sử dụng để khử nhiễu bằng cách áp dụng quy trình ngược lại cho hình ảnh bị nhiễu. Mô hình sẽ lặp lại dự đoán hình ảnh sạch ở bước tiếp theo, loại bỏ dần nhiễu cho đến khi khôi phục được hình ảnh sạch ban đầu.

Tóm lại, DDPM sử dụng quy trình khuếch tán kết hợp với mạng lưới thần kinh để tìm hiểu cách phân phối hình ảnh và khử nhiễu hình ảnh bằng cách đảo ngược quá trình này. Họ đã thể hiện nhiều hứa hẹn trong việc tạo ra hình ảnh và đã đạt được độ trung thực của hình ảnh sánh ngang với Mạng đối thủ sáng tạo (GAN). [4] [5]

### 3.2.2 Tao ảnh:

Diffusion Model đã được sử dụng để tạo ra hình ảnh với kết quả ấn tượng. Bắt đầu với một hình ảnh nhiễu ngẫu nhiên và dần dần tinh chỉnh nó thành một hình ảnh mạch lạc. Sau nhiều lần lặp lại, nhiễu đã được loại bỏ hoàn toàn và những gì còn lại là hình ảnh được tạo ra. Hình ảnh này phải giống với loại hình ảnh mà mô hình đã được huấn luyện. Có một số mô hình khuếch tán nổi tiếng để tạo hình ảnh:

- *DALL-E 2*: Một mô hình OpenAI tạo hình ảnh từ lời nhắc văn bản.
- *Imagen*: Một mô hình của Google Research xây dựng dựa trên sức mạnh của các mô hình ngôn ngữ biến đổi lớn trong việc hiểu văn bản và xoay quanh sức mạnh của các mô hình khuếch tán trong việc tạo ra hình ảnh có độ trung thực cao.

- *Stable Diffusion*: Mô hình StabilityAI tạo ra hình ảnh chất lượng cao từ các lời nhắc văn bản đơn giản và cấu hình tối thiểu.
- *Midjourney*: Một mô hình khuếch tán phổ biến khác để tạo hình ảnh. .

Những mô hình này đã được sử dụng để tạo ra nhiều hình ảnh khác nhau, từ những đồ vật hàng ngày đến những sinh vật kỳ ảo và thậm chí cả những khái niệm trừu tượng. Họ đã thể hiện nhiều hứa hẹn trong lĩnh vực sáng tạo nghệ thuật AI. [5]

### 3.2.3 *Tăng cường độ phân giải:*

Super-resolution diffusion models là một loại Diffusion Models được sử dụng cho hình ảnh siêu phân giải. Chúng hoạt động bằng cách tinh chỉnh dần dần hình ảnh có độ phân giải thấp thành hình ảnh có độ phân giải cao. Đây là một giải thích về các bước hoạt động của mô hình này:

- *Forward Process*: Mô hình bắt đầu với hình ảnh có độ phân giải cao và thêm nhiễu Gaussian ở mỗi bước thời gian, dần dần chuyển đổi hình ảnh thành hình ảnh có độ phân giải thấp.
- *Reverse Process*: Mô hình học cách đảo ngược forward process, bắt đầu từ hình ảnh có độ phân giải thấp và tinh chỉnh dần dần để khôi phục hình ảnh có độ phân giải cao ban đầu. Điều này được thực hiện bằng cách huấn luyện mạng lưới thần kinh để dự đoán hình ảnh có độ phân giải cao ở bước tiếp theo dựa trên hình ảnh có độ phân giải thấp hiện tại.
- *Mục tiêu đào tạo & Loss Function*: Mô hình được đào tạo bằng cách sử dụng loss function nhằm khuyến khích hình ảnh có độ phân giải cao được dự đoán gần với hình ảnh có độ phân giải cao thực tế. Điều này thường được thực hiện bằng cách sử dụng một dạng suy luận biến phân.
- *Độ phân giải siêu cao*: Sau khi mô hình được huấn luyện, mô hình có thể được sử dụng cho độ phân giải siêu cao bằng cách áp dụng quy trình ngược lại cho hình ảnh có độ phân giải thấp. Mô hình sẽ lặp lại dự đoán hình ảnh có độ phân giải cao

ở bước tiếp theo, dần dần tinh chỉnh hình ảnh cho đến khi khôi phục được hình ảnh có độ phân giải cao ban đầu.

Các Diffusion Model đã được sử dụng cho các nhiệm vụ có độ phân giải siêu cao với những tiến bộ đáng kể. Chúng điều chỉnh chất lượng hình ảnh kỹ thuật chặt chẽ hơn với sở thích của con người và mở rộng các ứng dụng có độ phân giải siêu cao. Các Diffusion Model giải quyết các hạn chế quan trọng của các phương pháp trước đó, nâng cao tính chân thực tổng thể và chi tiết trong hình ảnh siêu phân giải.

Một số mô hình tăng cường độ phân giải là:

- *SRDiff*: Mô hình này là first diffusion - based model cho Độ phân giải siêu hình ảnh đơn (SISR). SRDiff được tối ưu hóa với một biến thể của giới hạn biến thiên về khả năng dữ liệu và có thể đưa ra các dự đoán siêu phân giải đa dạng và thực tế bằng cách chuyển dần nhiều Gaussian thành hình ảnh siêu phân giải được điều chỉnh trên đầu vào có độ phân giải thấp thông qua chuỗi Markov. [6]
- *Implicit Diffusion Model(IDM)*: IDM tích hợp biểu diễn thần kinh tiềm ẩn và diffusion model khử nhiễu trong khuôn khổ thống nhất từ đầu đến cuối để có độ phân giải siêu hình ảnh liên tục có độ trung thực cao.
- *Efficient Hybrid Diffusion Model*: Mô hình này sử dụng khả năng tổng hợp mạnh mẽ của diffusion model để hiểu đầy đủ thông tin hình ảnh, giải quyết những thiếu sót của các phương pháp siêu phân giải viễn thám dựa trên mạng thần kinh trước đây thường không thu được hình ảnh chi tiết có độ trung thực cao tại độ phóng đại cao.

Những mô hình này đã cho thấy nhiều hứa hẹn trong lĩnh vực siêu phân giải, cung cấp hình ảnh chất lượng cao từ đầu vào có độ phân giải thấp.

### **3.3 Dữ liệu cần:**

#### **3.3.1 Dữ liệu huấn luyện:**

##### **Giảm nhiễu:**

Với một mô hình giảm nhiễu, để có một mô hình hoạt động tốt, dữ liệu huấn luyện nên bao gồm nhiều loại ảnh với nhiều loại nhiễu khác nhau như nhiễu Gaussian, nhiễu hạt tiêu, muối, nhiễu răng cưa,...

Đồng thời cũng rất tốt nếu ta có thể có cả phiên bản nhiễu và sạch của bức ảnh đó. Mô hình sẽ học để ánh xạ từ nhiễu thành ảnh sạch

##### **Tạo ảnh:**

Để tạo ảnh được đa dạng thì, dữ liệu dùng để huấn luyện nên lớn và đa dạng. Đồng thời chứa nhiều ảnh ở các dạng mà ta muốn mô hình tạo ra.

Nếu mô hình được điều chỉnh bằng văn bản, bộ dữ liệu huấn luyện nên chứa theo kiểu cặp ảnh - chữ. Mỗi ảnh nên được cặp với một văn bản mô tả để model có thể học được mối quan hệ của văn bản và ảnh hiện thị

##### **Tăng chất lượng ảnh:**

Dữ liệu nên là một cặp gồm ảnh chất lượng thấp và cao. Đồng thời tập dữ liệu này cũng cần lớn và đa dạng để mô hình có thể giải quyết được nhiều loại hình ảnh khác nhau.

#### **3.3.2 Dữ liệu kiểm tra mô hình:**

##### **Giảm nhiễu:**

Với tệp kiểm tra thì ta cần đưa vào mô hình được huấn luyện một hình ảnh bị nhiễu và chuẩn bị một hình ảnh sạch của nó để kiểm tra mức độ giảm nhiễu của mô hình và điều chỉnh các tham số theo yêu cầu của từng mô hình.

### **Tạo ảnh:**

Có thể đưa vào mô hình là những ảnh không quá rõ ràng, hoặc nhiễu đơn thuần, với các mô hình có văn bản thì ta cần đưa thêm yêu cầu cho mô hình dưới dạng văn bản và điều chỉnh các tham số theo yêu cầu của từng mô hình.

### **Tăng chất lượng ảnh:**

Ta đưa vào trong mô hình những ảnh chất lượng thấp và điều chỉnh các tham số theo yêu cầu của từng mô hình để thu được hình ảnh chất lượng cao.

### **3.4 Các ứng dụng chính của Diffusion Model:**

- *Generation:* Diffusion Model có thể tạo ra các mẫu mới một cách chính xác với phân phối đầu vào. Điều này đặc biệt hữu ích trong việc tạo ra hình ảnh 2D, video, đối tượng 3D, di chuyển, và cảnh 4D có tính thực tế. Mô hình bắt đầu từ một phân phối đơn giản và dần dần biến đổi nó để phù hợp với phân phối mục tiêu thông qua quá trình lan truyền.[7]
- *Editing:* Diffusion Model cũng có thể được sử dụng để chỉnh sửa các mẫu hiện tại. Điều này được thực hiện bằng cách ánh xạ mẫu vào không gian ẩn, áp dụng các thay đổi trong không gian ẩn, và sau đó ánh xạ nó trở lại không gian dữ liệu. Điều này cho phép kiểm soát chính xác quá trình chỉnh sửa.[7]
- *Reconstruction:* Diffusion Model có thể tái tạo các mẫu từ biểu diễn ẩn. Điều này hữu ích trong các nhiệm vụ như tái tạo hình ảnh nơi mục tiêu là tái tạo một hình ảnh từ một phiên bản nén hoặc mã hóa. Các mô hình có thể ánh xạ biểu diễn ẩn trở lại không gian dữ liệu, hiệu quả tái tạo mẫu gốc.[7]
- *Personalization:* Diffusion Model có thể được cá nhân hóa để tạo ra các mẫu được tùy chỉnh theo yêu cầu hoặc sở thích cụ thể. Điều này được thực hiện bằng cách điều kiện mô hình trên các biến cụ thể trong quá trình tạo sinh.[7]
- *Conditioning:* Diffusion Model có thể được điều kiện trên các biến cụ thể để tạo ra các mẫu thỏa mãn điều kiện nhất định. Điều này cho phép mô hình tạo ra các

mẫu không chỉ phù hợp với phân phối đầu vào mà còn đáp ứng các ràng buộc nhất định.[7]

- *Inversion:* Diffusion Model có thể thực hiện nghịch đảo, bao gồm việc ánh xạ một mẫu trở lại điểm tương ứng của nó trong không gian ẩn. Điều này hữu ích để hiểu cấu trúc của không gian ẩn và mối quan hệ giữa các mẫu khác nhau.[7]

## Kết Luận

Trong bài tiểu luận này, chúng ta đã xem xét sâu rộng về hai chủ đề quan trọng là các phương pháp xử lý ảnh và mô hình Diffusion.

Về phần "Các phương pháp xử lý ảnh", chúng ta đã thấy rõ được bản chất của các thuật toán và kỹ thuật xử lý ảnh. Từ việc tiền xử lý đến nhận dạng đối tượng và phân loại ảnh, các nghiên cứu trong lĩnh vực này không chỉ giúp cải thiện hiệu suất của các hệ thống thị giác máy tính mà còn mở ra nhiều ứng dụng mới trong thực tế. Việc tích hợp trí tuệ nhân tạo vào xử lý ảnh đã mở đường cho nhiều tiến bộ, đặc biệt là trong các lĩnh vực như ô tô tự lái, y tế, và giám sát an ninh.

Mô hình Diffusion không chỉ giúp chúng ta hiểu rõ hơn về cách mà thông tin, màu sắc, và các đặc trưng của hình ảnh được truyền tải trong không gian, mà còn mở ra nhiều cơ hội để cải thiện quá trình xử lý ảnh. Áp dụng mô hình Diffusion vào lĩnh vực xử lý ảnh có thể tạo ra những thuật toán mới để tái tạo hình ảnh, làm mịn đồng đều màu sắc, và giảm nhiễu một cách hiệu quả. Trong các ứng dụng thực tế, mô hình Diffusion có thể được sử dụng để nâng cao chất lượng ảnh, đặc biệt là trong bối cảnh y tế, nơi độ chính xác của hình ảnh đóng vai trò quan trọng trong việc chẩn đoán và theo dõi các bệnh lý. Việc áp dụng các phương pháp Diffusion trong quá trình xử lý hình ảnh y tế có thể giúp làm sáng tỏ các chi tiết, giảm nhiễu và tăng độ tương phản, đồng thời tạo ra những ảnh có chất lượng hơn.

Tích hợp giữa các phương pháp xử lý ảnh và mô hình Diffusion đã tạo ra những cơ hội mới và đưa ra những giải pháp sáng tạo. Các ứng dụng như xử lý ảnh y tế với mô hình Diffusion để phát hiện bất thường, hay ứng dụng trong nghệ thuật số với việc tái tạo ảnh thông qua các quy luật phân tán, là những ví dụ rõ ràng về sức mạnh của việc kết hợp hai lĩnh vực này.

Tóm lại, bài tiểu luận đã đề cập đến sự phát triển và tiềm năng lớn của các phương pháp xử lý ảnh và mô hình Diffusion. Sự hiểu biết sâu sắc về cả hai chủ đề này không chỉ mở ra những hướng nghiên cứu mới mà còn giúp chúng ta hiểu rõ hơn về cách chúng tương tác và có thể cộng tác để tạo ra những giải pháp hiệu quả trong tương lai.

## Tài liệu tham khảo

- [1] Rafael C Gonzalez. *Digital image processing*. Pearson education india, 2009.
- [2] Alexander Mordvintsev and K Abid. “Opencv-python tutorials documentation”. In: *Obtenido de https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf* (2014).
- [3] Florinel-Alin Croitoru, Vlad Hondu, Radu Tudor Ionescu, et al. “Diffusion models in vision: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).
- [4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.
- [5] Prafulla Dhariwal and Alexander Nichol. “Diffusion models beat gans on image synthesis”. In: *Advances in neural information processing systems* 34 (2021), pp. 8780–8794.
- [6] Haoying Li, Yifan Yang, Meng Chang, et al. “Srdiff: Single image super-resolution with diffusion probabilistic models”. In: *Neurocomputing* 479 (2022), pp. 47–59.
- [7] Ryan Po, Wang Yifan, Vladislav Golyanik, et al. “State of the art on diffusion models for visual computing”. In: *arXiv preprint arXiv:2310.07204* (2023).