TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

···🙞🙜🕮🙞🙜···

A yellow and red logo

Description automatically generated

BÁO CÁO BÀI TẬP LỚN

Học phần: XỬ LÍ ẢNH

TÌM HIỂU, CÀI ĐẶT, ỨNG DỤNG CÁC

PHÉP TOÁN HÌNH THÁI TRONG XỬ LÝ ẢNH

|  |  |
| --- | --- |
| GVHD: | ThS. Lê Thị Thủy |
| Nhóm - Lớp: | 8 - 20232IT6072002 |
| Thành viên: |  |
| 2021602059 | Ngô Quý Điệp |
| 2021600454 | Nguyễn Mạnh Đức |
| 2021601158 | Đặng Quốc Diệu |
| 2021606475 | Hoàng Lan Hương |
| 2021608136 | Nguyễn Thị Linh |

**Hà Nội - Năm 2024**

# LỜI MỞ ĐẦU

Xử lý ảnh là một môn học có rất nhiều ứng dụng trong thực tế. Nó là một ngành khoa học mới mẻ so với nhiều ngành khoa học khác nhưng tốc độ phát triển của nó rất nhanh, kích thích các trung tâm nghiên cứu, ứng dụng, đặc biệt là máy tính chuyên dụng riêng cho nó. Chúng em đã chọn đề tài "***Tìm hiểu, cài đặt ứng dụng các phép toán hình thái trong xử lý ảnh***" vì thấy đây là một đề tài rất hay, đặc biệt có liên quan đến vấn đề phục hồi và nhận dạng, một kỹ thuật rất được quan tâm bây giờ.

Chính nhờ quá trình nỗ lực làm bài tập lớn, chúng em đã xây dựng được những mô đun và tích luỹ được những kinh nghiệm hết sức quý báu. Và vì vậy nên do nhận thấy sự liên quan, hỗ trợ lẫn nhau giữa các đề tài nên chúng em xin được phát triển thêm một số tính năng của chương trình bằng cách tìm hiểu và giải quyết thêm một số đề tài khác. Nhờ vậy mà khả năng làm việc theo nhóm của chúng em đã tiến bộ rõ rệt. Tuy đã cố gắng hết sức nhưng bài viết chắc chắn không thể tránh khỏi những sai sót. Vì vậy rất mong nhận được sự đóng góp của thầy cô để chúng em hoàn thiện đề tài.

Chúng em xin cảm ơn cô đã tận tình hướng dẫn và dành thời gian đọc bản báo cáo này.

# MỤC LỤC

[LỜI MỞ ĐẦU 1](#_Toc169899683)

[MỤC LỤC 2](#_Toc169899684)

[CHƯƠNG 1: LÝ THUYẾT PHÉP TOÁN HÌNH THÁI 4](#_Toc169899685)

[1.1. Phép co (Erosion) 4](#_Toc169899686)

[1.1.1. Định nghĩa 4](#_Toc169899687)

[1.1.2. Công thức 4](#_Toc169899688)

[1.1.3. Ý nghĩa 6](#_Toc169899689)

[1.1.4. Ứng dụng 6](#_Toc169899690)

[1.2. Phép giãn (Dilation) 6](#_Toc169899691)

[1.2.1. Định nghĩa 6](#_Toc169899692)

[1.2.2. Công thức 6](#_Toc169899693)

[1.2.3. Ý nghĩa 7](#_Toc169899694)

[1.2.4. Ứng dụng 7](#_Toc169899695)

[1.3. Phép mở (Opening) 7](#_Toc169899696)

[1.3.1. Định nghĩa 7](#_Toc169899697)

[1.3.2. Công thức 8](#_Toc169899698)

[1.3.3. Ý nghĩa 8](#_Toc169899699)

[1.3.4. Ứng dụng 8](#_Toc169899700)

[1.4. Phép đóng (Closing) 9](#_Toc169899701)

[1.4.1. Định nghĩa 9](#_Toc169899702)

[1.4.2. Công thức 9](#_Toc169899703)

[1.4.3. Ý nghĩa 9](#_Toc169899704)

[1.4.4. Ứng dụng 10](#_Toc169899705)

[1.5. Phép Hit-or-miss Transformation 10](#_Toc169899706)

[1.5.1. Định nghĩa 10](#_Toc169899707)

[1.5.2. Công thức 10](#_Toc169899708)

[1.5.3. Ý nghĩa 13](#_Toc169899709)

[1.5.4. Ứng dụng 13](#_Toc169899710)

[CHƯƠNG 2: ỨNG DỤNG VÀ CHƯƠNG TRÌNH 14](#_Toc169899711)

[2.1. Các hàm dùng chung trong BTL 14](#_Toc169899712)

[2.1.1. Phép toán hình thái co (erosion) 14](#_Toc169899713)

[2.1.2. Phép toán hình thái giãn 14](#_Toc169899714)

[2.1.3. Hàm check erosion có hợp lệ hay không 15](#_Toc169899715)

[2.1.4. Hàm phân ngưỡng tự động 16](#_Toc169899716)

[2.2. Ứng dụng các phép toán hình thái xử lý nhiễu biển số xe 17](#_Toc169899717)

[2.2.1. Tư tưởng 17](#_Toc169899718)

[2.2.2. Source code 17](#_Toc169899719)

[2.2.3. Kết quả 18](#_Toc169899720)

[2.3. Ứng dụng các phép toán hình thái đếm số lượng đối tượng chồng lên nhau 18](#_Toc169899721)

[2.3.1. Tư tưởng 18](#_Toc169899722)

[2.3.2. Source code 18](#_Toc169899723)

[2.3.3. Kết quả 19](#_Toc169899724)

[2.4. Ứng dụng các phép toán hình thái xử lý ảnh vân tay 19](#_Toc169899725)

[2.4.1. Tư tưởng 19](#_Toc169899726)

[2.4.2. Source code 20](#_Toc169899727)

[2.4.3. Kết quả 20](#_Toc169899728)

[2.5. Ứng dụng các phép toán hình thái tìm khung xương (skeleton) 21](#_Toc169899729)

[2.5.1. Tư tưởng 21](#_Toc169899730)

[2.5.2. Source code 21](#_Toc169899731)

[2.5.3. Kết quả 22](#_Toc169899732)

[2.6. Ứng dụng các phép toán hình thái làm dày đối tượng (Thickening) 23](#_Toc169899733)

[2.6.1. Tư tưởng 23](#_Toc169899734)

[2.6.2. Source code 24](#_Toc169899735)

[2.6.3. Kết quả 25](#_Toc169899736)

[TÀI LIỆU THAM KHẢO 26](#_Toc169899737)

# CHƯƠNG 1: LÝ THUYẾT PHÉP TOÁN HÌNH THÁI

## Phép co (Erosion)

### Định nghĩa

Xét tập hợp A và tập hợp B (phần tử cấu trúc), phép co ảnh nhị phân của tập hợp A bởi phần tử cấu trúc B được kí hiệu là 

### Công thức



*Hình 1.1.1 Công thức phép co ảnh*

Phép co ảnh nhị phần của tập hợp A bởi phần tử cấu trúc B là tập hợp các điểm z (z nằm ở tâm điểm của phần tử cấu trúc B) sao cho Bz là tập con của A.

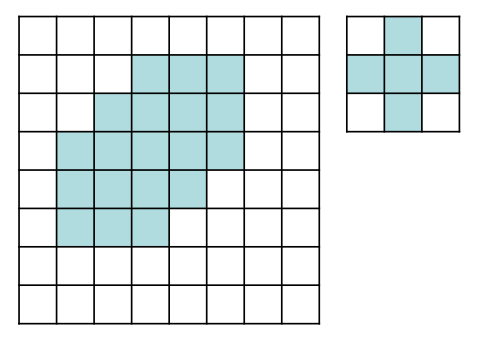
Mô tả thuật toán:

- Dịch chuyển gốc của phần tử cấu trúc B trên từng điểm ảnh của tập A.

+ Nếu B nằm hoàn toàn trong A thì tại vị trí gốc của B sẽ có giá trị 1 (giá trị đối tượng)

+ Ngược lại thì sẽ có giá trị 0 (giá trị nền)

***Thực hiện***

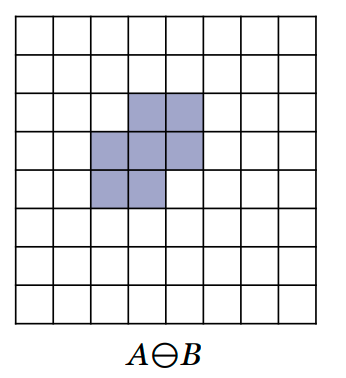


a) b)

*Hình 1.1.2 a) Mô phỏng ma trận ảnh*

*b) Phần tử cấu trúc*

***Kết quả:***



*Hình 1.1.3 Kết quả sau khi thực hiện phép co*

### Ý nghĩa

Có tác dụng mở rộng các lỗ, phá vỡ các bộ phận mỏng và làm co hẹp các đối tượng.

### Ứng dụng

• Loại bỏ nhiễu trong ảnh.

• Đếm số lượng đối tượng khi bị đè lên nhau.

## Phép giãn (Dilation)

### Định nghĩa

Phép toán Dilation hay còn gọi là sự giãn nở là một trong hai phép toán cơ bản trong lĩnh vực hình thái toán học của xử lý ảnh.

Khác với Erosion là một hoạt động co nhỏ hoặc làm mỏng đối tượng, Dilation được sử dụng để mở rộng, làm mịn ảnh hoặc nối các đường nét đứt trong ảnh.

Cách thức và mức độ giãn nở được kiểm soát bởi hình dạng của phần tử cấu trúc.

### Công thức

Dilation của X theo B là hợp của tất cả các Bx với x thuộc X

X B =

*Hình 1.2.1 Công thức phép giãn ảnh*

Trong đó:  
 + X: Đối tượng ảnh cần thực hiện phép giãn nở

+ B: Phần tử cấu trúc: Có 1 tâm và có hình dạng cơ sở

+ Bx: Là sự dịch chuyển của B tới vị trí x thuộc X.

Ví dụ minh hoạ:

X =  
 =

Mô tả thuật toán:

* Đưa ma trận kernel vào trên ảnh, và đặt tâm của kernel lên một điểm cụ thể trong ảnh.
* Tại điểm đó, kiểm tra xem các pixel trong kernel có giá trị là 1 hay không. Nếu có, ta coi điểm này là một điểm trắng trong ảnh đầu ra.
* Di chuyển kernel đến một điểm khác trên trong ảnh, và lặp lại quá trình trên cho tất cả các điểm khác.
* Sau khi thực hiện quá trình trên cho tất cả các điểm, ta sẽ có một ảnh đầu ra mới với vùng được chọn đã được mở rộng.

### Ý nghĩa

Ý nghĩa của thuật toán Dilation là mở rộng các đối tượng trong ảnh, làm cho chúng trở nên lớn hơn và lấp đầy các lỗ hổng nhỏ. Khi thực hiện Dilation trên một ảnh, các đối tượng trong ảnh sẽ được mở rộng ra và trở nên rõ ràng hơn. Điều này giúp cải thiện chất lượng của ảnh và làm cho các đối tượng trở nên dễ nhận dạng và phân loại hơn trong các ứng dụng thị giác máy tính.

### Ứng dụng

Các ứng dụng của thuật toán Dilation như:

• Cải thiện chất lượng ảnh

• Loại bỏ nhiễu

• Xử lý ảnh y tế (skeleton)

## Phép mở (Opening)

### Định nghĩa

Phép opening là một phép biến đổi hình học trong đó một hình ảnh được làm mịn và giảm nhiễu bằng cách áp dụng một bộ lọc nhất định sau đó thực hiện phép giãn nhỏ để tách các đối tượng lớn thành các đối tượng nhỏ hơn và loại bỏ các đối tượng nhỏ.

Phép opening thường được sử dụng trong xử lý hình ảnh và phân tích hình ảnh để làm sạch hình ảnh, tách các đối tượng và loại bỏ các chi tiết không mong muốn. Nó thường được sử dụng để cải thiện chất lượng hình ảnh và giúp cho các thuật toán xử lý hình ảnh hoạt động hiệu quả hơn.

### Công thức



*Hình 1.3.1 Công thức phép mở (Opening)*

Các bước thực hiện phép opening trên một hình ảnh là:

1. Áp dụng một bộ lọc (kernel) nhất định lên hình ảnh để làm mịn và giảm nhiễu. Bộ lọc thường được sử dụng là bộ lọc trung bình (median filter) hoặc bộ lọc Gauss (Gaussian filter).

2. Thực hiện phép giãn nhỏ (erosion) trên hình ảnh đã được lọc để tách các đối tượng lớn thành các đối tượng nhỏ hơn. Phép giãn nhỏ thực hiện bằng cách trượt một kernel qua toàn bộ hình ảnh và chỉ giữ lại các pixel có giá trị lớn nhất trong vùng kernel.

3. Thực hiện phép mở rộng (dilation) trên hình ảnh đã được giãn nhỏ để loại bỏ các đối tượng nhỏ và làm cho các đối tượng còn lại trở nên rõ nét hơn. Phép mở rộng thực hiện bằng cách trượt một kernel qua toàn bộ hình ảnh và giữ lại các pixel có giá trị lớn nhất trong vùng kernel.

### Ý nghĩa

Quá trình phép opening giúp làm sạch hình ảnh, tách các đối tượng và loại bỏ các chi tiết không mong muốn. Kết quả cuối cùng là một hình ảnh được mịn và rõ nét hơn, giúp cho các thuật toán xử lý hình ảnh hoạt động hiệu quả hơn.

### Ứng dụng

• Lấy nét vân tay

• Xóa nhiễu

• Cải thiện chất lượng ảnh

Tóm lại, phép opening là một phép biến đổi hình học quan trọng trong xử lý hình ảnh và có rất nhiều ứng dụng trong nhiều lĩnh vực khác nhau.

## Phép đóng (Closing)

### Định nghĩa

Closing là thực hiện phép giãn rồi bắt đầu thực hiện phép co với một cấu trúc. Giúp là mượt các đường viền, loại bỏ các lỗ nhỏ, làm mượt các đỉnh khe hẹp

Cụ thể, phép closing được thực hiện bằng cách áp dụng phép dilation sau đó là phép erosion lên ảnh đầu vào. Phép dilation được sử dụng để mở rộng các vùng trắng, trong khi phép erosion được sử dụng để thu nhỏ chúng. Khi hai phép toán này được kết hợp với nhau, các lỗ hổng nhỏ sẽ được lấp đầy và các vật thể sẽ trở nên liền mạch hơn.

### Công thức

A B = (A B) B

*Hình 1.4.1 Công thức phép đóng (closing)*

Trong đó: A là ảnh đầu vào

B là cấu trúc hình, thường là một hình vuông hoặc tròn

Là phép Dilation (Giãn)

Là phép Erosion(Co)

Ví dụ minh hoạ:

Ta có ma trận nhị phân sau:

Sau khi thực hiện Closing với một ma trận vuông 3x3

Kết quả của phép Closing:

### Ý nghĩa

Phép closing giúp loại bỏ các lỗ hổng nhỏ và liền mạch hoá các vật thể trong ảnh từ đó cải thiện độ chính xác và độ tin cậy của các kết quả phân tích ảnh

### Ứng dụng

Phép closing là một trong những phép toán quan trọng trong xử lý ảnh và có nhiều ứng dụng trong nhiều lĩnh vực khác nhau bao gồm:

• Xử lý nhiễu ảnh

• Nối vật thể bị đứt đoạn

• Cải thiện chất lượng ảnh

## Phép Hit-or-miss Transformation

### Định nghĩa

Hit-or-miss Transformation hay Phép biến đổi trúng hay trượt là một phép toán hình thái có khả năng phát hiện một cấu hình (hoặc mẫu) nhất định trong một hình ảnh nhị phân. Phép biến đổi sử dụng phép toán tương tự nhưng mở rộng hơn một chút so với phép toán sử dụng cho phép co và dãn, hoạt động với một cặp phần tử cấu trúc (Structuring Element) rời rạc – thường đặt là B1, B2.

Phép biến đổi Hit-or-Miss tìm thấy các pixel có vùng lân cận khớp với hình dạng của phần tử cấu trúc thứ nhất B1 trong khi không khớp với hình dạng của phần tử cấu trúc thứ hai B2 cùng một lúc. B1 và B2 là các phần tử cấu trúc biệt lập (hai tập hợp được gọi là biệt lập nếu B1 ∩ B2 = Ø, tập hợp rỗng), bị chặn (một tập hợp bị chặn khi tồn tại một tập hợp khác chứa nó).

### Công thức

Về mặt toán học, thao tác được áp dụng cho hình ảnh A có thể được biểu thị như sau:

https://kipalog.kaopiz.com/uploads/5bc9/cbaa/Screenshot%20from%202020-01-22%2011-06-49.png

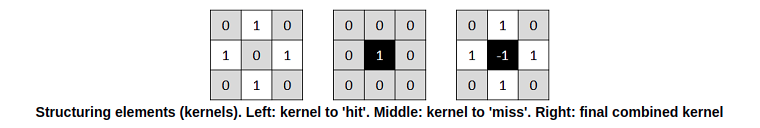
*Hình 1.5.1 Công thức phép Hit – or – miss*

Trong đó: Aᶜ là tập hợp phần bù của A – tức là tập hợp các phần tử không thuộc A. Trong trường hợp xử lý ảnh, Aᶜ là một ảnh A với foreground và background ngược lại so với A.

Các phần tử cấu trúc B1 và B2 có thể được kết hợp thành một phần tử

B.

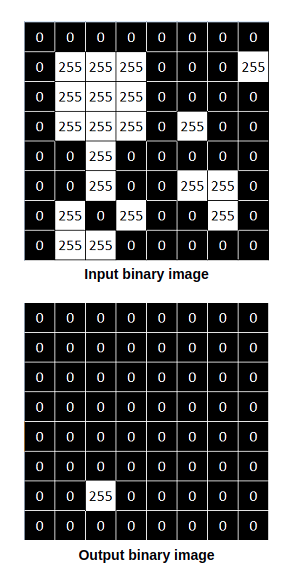
Ví dụ:



*Hình 1.5.2 Các phần tử cấu trúc*

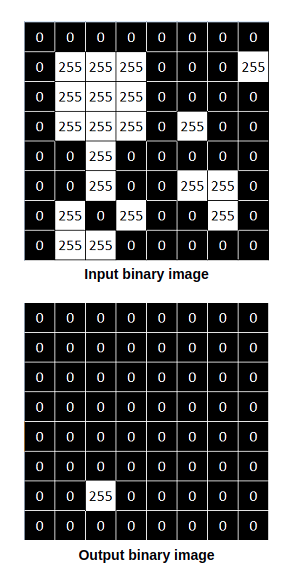
Trong đó: SE bên trái là B1, ở giữa là B2 và bên phải là B

Ví dụ, ta tìm kiếm một mẫu trong đó pixel trung tâm thuộc về background (có giá trị điểm ảnh bằng 0) trong khi các pixel phía bắc, nam, đông và tây thuộc về foreground (có giá trị điểm ảnh bằng 255). Phần còn lại của pixel trong vùng lân cận có thể là bất kỳ loại nào. Áp dụng kernel này cho một hình ảnh đầu vào như sau:



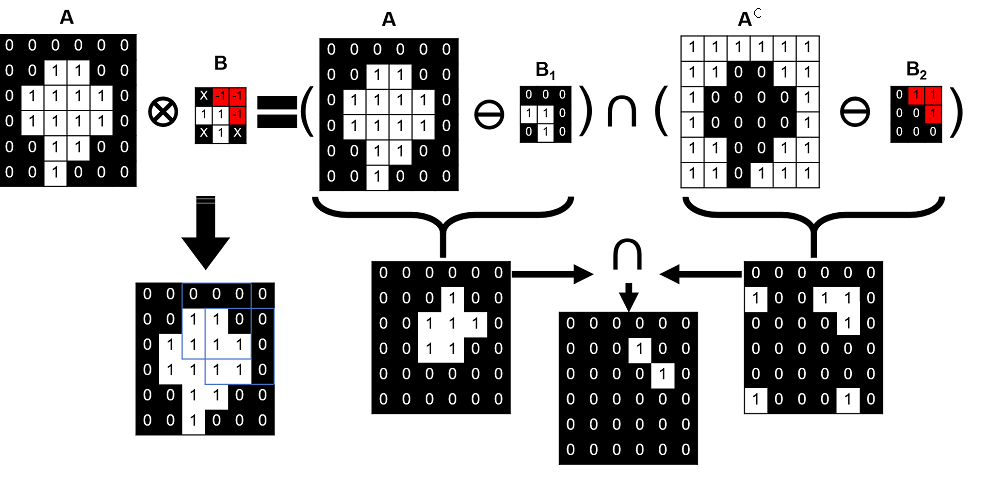
*Hình 1.5.3 Ảnh nhị phân input*

Sau khi áp dụng kernel B1, B2 vào hình ảnh, ta được:



*Hình 1.5.4 Ảnh sau khi thực hiện phép chuyển đổi (output)*

Cụ thể hơn, ta có:



*Hình 1.5.5 Mô tả cụ thể từng bước của phép biến đổi*

Ta có thể thấy, phần tử cấu trúc B (tạo nên từ 2 phần tử cấu trúc B1, B2) có 3 mức giá trị: -1, 0, 1. Kết quả cuối cùng của mỗi điểm ảnh mà SE quét qua sẽ bằng 1 nếu như giá trị điểm ảnh ở những những vị trí có giá trị -1 phải bằng 0, vị trí có giá trị 1 phải bằng 1 và vị trí 0 không bắt buộc.

### Ý nghĩa

Phép biển đổi trúng hay trượt là cơ sở của các hoạt động hình thái tân tiến hơn như tỉa thưa hoặc cắt tỉa.

Phép biến đổi trúng hay trượt rất hữu ích trong việc tìm kiếm các mẫu trong ảnh nhị phân. Đồng thời phép biến đổi có thể được biến đổi để sử dụng như phép co, dãn, đóng, mở, làm mỏng hay làm dày thông qua việc cài đặt một số toán tử nhất định.

Có thể coi phép biến đổi là khởi nguồn của gần như mọi phép toán hình thái khác.

### Ứng dụng

Phép biến đổi trúng hay trượt có thể coi là khởi nguồn của mọi phép toán hình thái khác, nên nó có thể ứng dụng vào đa số các lĩnh vực xử lý ảnh sử dụng đến phép toán hình thái.

Một số ứng dụng có thể kể đến như:

• Cải thiện chất lượng ảnh

• Làm dày ảnh (thickening)

• Loại bỏ nhiễu

# CHƯƠNG 2: ỨNG DỤNG VÀ CHƯƠNG TRÌNH

## Các hàm dùng chung trong BTL

### Phép toán hình thái co (erosion)

def erosion(image, kernel):

rows, cols = image.shape

kernel\_size = kernel.shape[0]

p = kernel\_size // 2

tmp = np.zeros(((rows + 2 \* p), (cols + 2 \* p)), dtype=int)

tmp[p:-p, p:-p] = image.copy()

output = np.zeros\_like(image)

for row in range(rows):

for col in range(cols):

if tmp[row + p,col + p] != 0:

s = tmp[row:row + 2 \* p + 1, col:col + 2 \* p + 1]

check = ((s & kernel) == kernel).all()

output[row, col] = int(check)

return output.astype(np.uint8)

### Phép toán hình thái giãn

def dilation(image, kernel):

rows, cols = image.shape

kernel\_size = kernel.shape[0]

p = kernel\_size // 2

tmp = np.zeros(((rows + 2 \* p), (cols + 2 \* p)), dtype=int)

tmp[p:-p, p:-p] = image.copy()

output = np.zeros\_like(tmp)

for row in range(rows):

for col in range(cols):

if tmp[row + p][col + p] != 0:

s = tmp[row: row + 2 \* p + 1, col: col + 2 \* p + 1]

output[row: row + 2 \* p + 1, col: col + 2 \* p + 1] = s | kernel

return output[p:-p, p:-p].astype(np.uint8)

### Hàm check erosion có hợp lệ hay không

def check\_erosion(kernel, img):

    if kernel.shape != img.shape:

        return False

    maskA = np.zeros\_like(kernel, dtype=bool)

    maskB = np.zeros\_like(img, dtype=bool)

    w, h = maskA.shape

    for i in range(w):

        for j in range(h):

            if kernel[i,j] != 0:

                maskA[i,j] = True

            if img[i,j] != 0:

                maskB[i,j] = True

    maskB = ~maskB

    for i in range(w):

        for j in range(h):

            if maskA[i, j] == maskB[i, j]:

                maskB[i, j] = True

    maskB = ~maskB

    if np.array\_equal(maskA, maskB):

        return True

    return False

### Hàm phân ngưỡng tự động

def threshold\_automatic(image):

n, m = image.shape

bins, his = np.unique(image, return\_counts=True)

t\_g = np.cumsum(his) # Tính t(g)

his\_times\_bins = his \* bins # Tính g \* h(g)

m\_g = (1 / t\_g) \* np.cumsum(his\_times\_bins)

m\_G = max(m\_g)

f\_g = []

for i in range(len(bins)):

if n \* m - t\_g[i] != 0:

temp = t\_g[i] / (m \* n - t\_g[i]) \* (m\_g[i] - m\_G)\*\*2

f\_g.append(temp)

else:

f\_g.append(0)

index = np.where(f\_g == max(f\_g))

return bins[index].item()

def segment\_with\_threshold(image):

threshold = threshold\_automatic(image)

res = np.where(image >= threshold, 1, 0)

return res

## Ứng dụng các phép toán hình thái xử lý nhiễu biển số xe

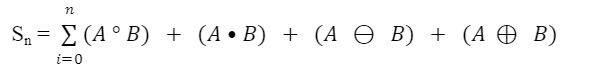
### Tư tưởng

Trong bài toán xử lý ảnh biển số xe, các phép biến đổi này có thể được sử dụng để tiền xử lý ảnh và tăng khả năng nhận dạng biển số.

Phép co và giãn có thể được sử dụng để thay đổi kích thước của ảnh biển số xe. Khi kích thước của ảnh quá lớn hoặc quá nhỏ, nó có thể làm giảm độ chính xác của quá trình nhận dạng.

Việc sử dụng các phép toán hình thái có thể được sử dụng để loại bỏ các đối tượng không phải biển số xe trong ảnh, giúp tăng độ chính xác của quá trình nhận dạng.

Kí hiệu:



Trong đó:

* A là ảnh đầu vào
* B là phần tử cấu trúc
* Sn là kết quả của ảnh biển số sau khi đã xử lý qua n phép toán hình thái để đạt kết quả tốt nhất

### Source code

def restore\_license\_plate(image, kernel):

    image\_binary = segment\_with\_threshold(image)

    eroded = erosion(image\_binary, kernel)

    opened = open\_image(eroded, kernel)

    return opened

### Kết quả

 *Hình 2.2.1 Ảnh trước khi xử lý* *Hình 2.2.2 Ảnh sau khi xử lý*

## Ứng dụng các phép toán hình thái đếm số lượng đối tượng chồng lên nhau

### Tư tưởng

* Chuyển ảnh về ảnh đa cấp xám, phân ngưỡng để phân ra đối tượng và nền.
* Co các đối tượng chồng chéo lên nhau để tách thành các đối tượng riêng.
* Thực hiện đếm các đối tượng riêng rồi trả về kết quả.

### Source code

def count\_object(gray):

    kernel = np.array([[0,1,1,1,0],

[1,1,1,1,1],

[1,1,1,1,1],

[1,1,1,1,1],

[0,1,1,1,0], dtype = int)

    img\_erode = erosion(gray, kernel)

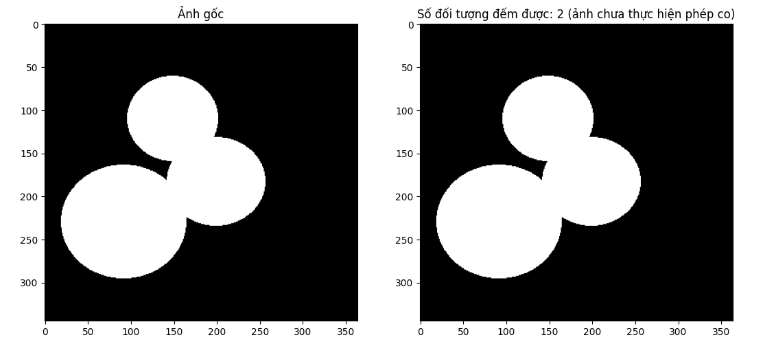
    for i in range(n):

        img\_erode = erosion(img\_erode, kernel)

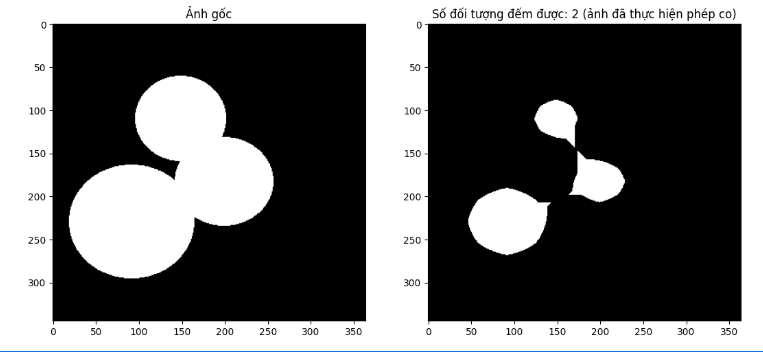
    ret, labels = cv2.connectedComponents(img\_erode)

return ret, img\_erode

### Kết quả



*Hình 2.3.1 Kết quả đếm trước khi thực hiện co*



*Hình 2.3.1 Kết quả đếm sau khi thực hiện co*

## Ứng dụng các phép toán hình thái xử lý ảnh vân tay

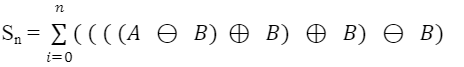
### Tư tưởng

Trong bài toán xử lý ảnh vân tay, các phép biến đổi này có thể được sử dụng để tiền xử lý ảnh và tăng khả năng nhận dạng vân tay.

Phép co và giãn có thể được sử dụng để thay đổi kích thước của ảnh vân tay. Khi kích thước của ảnh quá lớn hoặc quá nhỏ, nó có thể làm giảm độ chính xác của quá trình nhận dạng.

Việc sử dụng các phép toán hình thái có thể được sử dụng để loại bỏ các đối tượng không phải vân tay trong ảnh, giúp tăng độ chính xác của quá trình nhận dạng.

Kí hiệu:



Trong đó:

* A là ảnh đầu vào
* B là phần tử cấu trúc
* Sn là kết quả của ảnh vân tay sau khi đã xử lý qua n phép toán hình thái để đạt kết quả tốt nhất

### Source code

def preprocess\_fingerprint(image, kernel):

    eroded = erosion(image, kernel)

    dilated = dilation(eroded, kernel)

    dilated = dilation(dilated, kernel)

    eroded = erosion(dilated, kernel)

    return eroded

### Kết quả

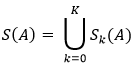
 

*Hình 2.4.1 Ảnh vân tay trước khi xóa nhiễu Hình 2.4.2 Ảnh vân tay sau khi xử lý nhiễu*

## Ứng dụng các phép toán hình thái tìm khung xương (skeleton)

### Tư tưởng

Toán tử rút xương (skeleton) có thể được biểu diễn bằng tập hữu hạn các phép co (erosion) và mở (opening).

Định nghĩa

Với:

* ss_17_Eqn
* ss_19_EqnB là phần tử cấu trúc và ss_20_Eqn  là kết quả của phép thực hiện erosion k lần với B của A.
* K là lần thực hiện rút xương cuối cùng, mà bước tiếp theo ss_21_Eqn sẽ cho kết quả rỗng.ss_22_Eqn

### Source code

def skeleton(img, kernel):

    size = np.size(img)

    skel = np.zeros(img.shape, dtype=np.uint8)

    img = segment\_with\_threshold(img)

    img = img.astype(np.uint8)

    done = False

    while (not done):

        eroded = erosion(img, kernel)

        temp = dilation(eroded, kernel)

        temp = temp.astype(np.uint8)

        temp = cv2.subtract(img, temp)

        skel = skel | temp

        img = eroded.copy()

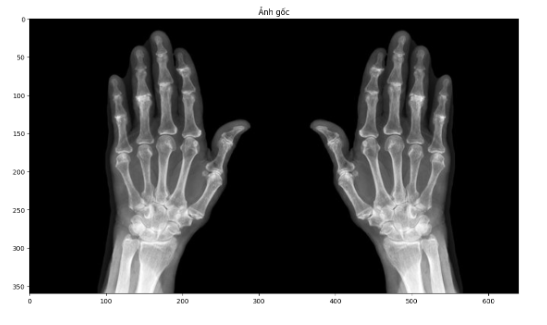
        zeros = size – np.count\_nonzero(img)

        if zeros == size:

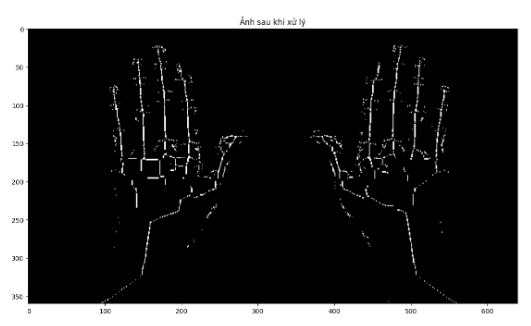
            done = True

    return skel

### Kết quả



*Hình 2.4.1 Ảnh gốc (ảnh x-quang)*



*Hình 2.4.2 Ảnh khung xương (skeleton)*

## Ứng dụng các phép toán hình thái làm dày đối tượng (Thickening)

### Tư tưởng

Toán tử Thickening có thể được biểu diễn thông qua phép biến đổi hit-or-miss:



Hay:



Trong đó:

* + B là phần tử cấu trúc phù hợp cho việc làm dày.
  +  là phép biến đổi hit or miss với A và kernel B, bản chất là giao của hai phép co của A và Aᶜ lần lượt với B1 và B2.

Toán tử làm dày được thể hiện thông qua một vòng lặp với các hình ảnh A thay đổi liên tục:



Với mỗi phần tử A mới được tính toán, ta thực hiện phép biến đổi mới cho đến khi sự thay đổi qua mỗi bước trở nên không đáng kể, thuật toán lúc đó sẽ được dừng lại.

### Source code

def add(matrix, result):

    for i in range(result.shape[0]):

        for j in range(result.shape[1]):

            if result[i, j] != 0:

                matrix[i, j] = 255

    return matrix

def thickening(matrix, kernel1, kernel2):

    matrix\_tmp = matrix.copy()

    thick = np.ones(matrix.shape, dtype=np.uint8)

    while np.count\_nonzero(thick) > 400:

condition = np.count\_nonzero(thick)

        matrix\_complement = ~matrix\_tmp

        result\_1 = erosion\_check(matrix\_tmp, kernel1)

        result\_2 = erosion\_check(matrix\_complement, kernel2)

        result = cv2.bitwise\_and(result\_1, result\_2)

        thick = result.copy()

print(np.count\_nonzero(thick))

        count = np.count\_nonzero(matrix\_tmp)

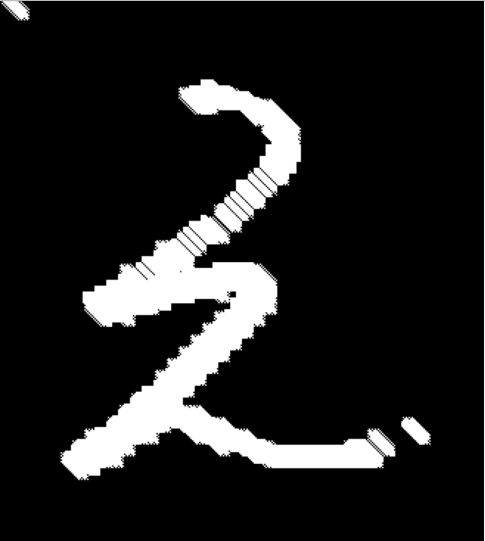
        matrix\_tmp = add(matrix\_tmp, result)

        if condition – np.count\_nonzero(thick) <= 10:

            break

    return matrix\_tmp

### Kết quả

*Hình 2.6.1 Ảnh trước khi làm dày Hình 2.6.2 Ảnh sau khi làm dày (thickening)*

# TÀI LIỆU THAM KHẢO

1. <https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm>
2. <https://homepages.inf.ed.ac.uk/rbf/HIPR2/hitmiss.htm>
3. Morphological-Image-Processing-Combined <https://web.stanford.edu/class/ee368/Handouts/Lectures/2014_Spring/Combined_Slides/7-Morphological-Image-Processing-Combined.pdf>
4. Morphological-skeleton <https://en.wikipedia.org/wiki/Morphological_skeleton>
5. Skeleton Binary Morphology <https://www.iostream.vn/article/toan-tu-khung-xuong-trong-anh-nhi-phan-skeleton-binary-morphology-0Iu1g>
6. Lê Thị Thuỷ (chủ biên), Ngô Thị Bích Thuỷ, Trần Hùng Cường, Giáo trình Xử Lý Ảnh, Trường đại học Công nghiệp Hà Nội, Nhà xuất bản thống kê - 2019.