**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA   
KHOA ĐIỆN – ĐIỆN TỬ**

**BỘ MÔN VIỄN THÔNG  
-------\*-------  
  
BÁO CÁO BÀI TẬP LỚN XỬ LÝ ẢNH**

**Đề tài: Hình thái học (Morphological Filtering)**

**Nhóm: L01\_06**

**GVHD: VÕ TUẤN KIỆT**

**Sinh viên thực hiện:**

**NGUYỄN QUANG SÁNG MSSV: 1813822**

**LÊ TRUNG QUÂN MSSV: 1813701**

**TRẦN MINH SANG MSSV: 1813815**

**TP HỒ CHÍ MINH, THÁNG 3/2022**

**MỤC LỤC**

[1. Giới thiệu 2](#_Toc99353864)

[2. Các thuật toán xử lý hình thái học 4](#_Toc99353865)

[2.1. Phần tử cấu trúc (Structuring element) 4](#_Toc99353866)

[2.2. Phép toán xói mòn (Erosion) 6](#_Toc99353867)

[2.3. Phép toán giãn nở (Dilation) 8](#_Toc99353868)

[2.4. Phép toán mở (Opening): 11](#_Toc99353869)

[2.5. Phép toán đóng (Closing): 14](#_Toc99353870)

[2.6. Lấp đầy vùng (Region fill) : 17](#_Toc99353871)

[2.7. Hit-and-miss transformation: 20](#_Toc99353872)

[2.8. Làm mỏng (Thinning): 25](#_Toc99353873)

[2.9. Skeletonization: 29](#_Toc99353874)

[2.10. Làm dày (Thickening): 35](#_Toc99353875)

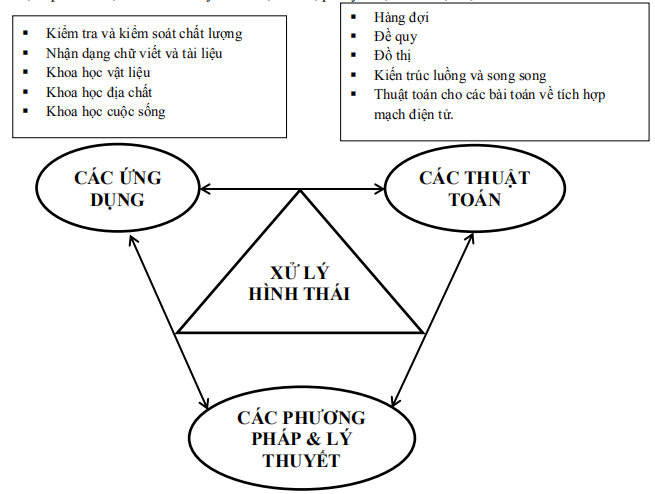
[2.11. White tophat: 39](#_Toc99353876)

[2.12. Black hat: 41](#_Toc99353877)

# 1. Giới thiệu

Các kỹ thuật xử lý ảnh số trên máy tính đang được rất nhiều nhà nghiên cứu quan tâm và phát triển. Sự phát triển của hình thái học là sự kết hợp giữa lý thuyết, ứng dụng, phương pháp và các thuật toán. Trong đó, các phương pháp mới được đưa ra nhằm giải quyết những vấn đề trong thực tế, lý thuyết kiểm chứng tính chính xác của các phương pháp, và phát triển các thiết bị phần cứng chuyên dụng hoặc các thuật toán hiệu quả để thực thi trên máy tính. Sự kết hợp này được thể hiện tại hình 1.

Xử lý hình thái học trên ảnh cung cấp cho chúng ta những mô tả định lượng về cấu trúc và hình dạng hình học của các đối tượng trong ảnh dựa trên những lý thuyết trong toán học như lý thuyết tập hợp, hình học tôpô, xác suất,...và nó đang được ứng dụng rộng rãi trong nhiều ứng dụng như phát hiện biên, phân đoạn đối tượng, giảm nhiễu, tìm xương ảnh ... Trong các ứng dụng thị giác máy tính, xử lý hình thái học có thể được sử dụng để nhận dạng đối tượng, nâng cao chất lượng ảnh, phân đoạn ảnh và kiểm tra khuyết điểm trên ảnh. Các phép toán xử lý hình thái học được thực hiện chủ yếu trên ảnh nhị phân và ảnh xám.



* Lọc ảnh
* Phân đoạn và phân lớp ảnh
* Đo lường ảnh
* Nhận dạng mẫu
* Phân tích kết cấu

*Hình 1: Sự phát triển của xử lý hình thái được đặc trưng bởi sự kết hợp kiết hợp giữa lý thuyết, ứng dụng, phương pháp và các thuật toán*

Trong các ứng dụng thị giác máy tính, xử lý hình thái học có thể được sử dụng để

nhận dạng đối tượng, nâng cao chất lượng ảnh, phân đoạn ảnh và kiểm tra khuyết điểm

trên ảnh . Các phép toán xử lý hình thái học được thực hiện chủ yếu trên ảnh nhị phân và ảnh xám. Ảnh nhị phân hay còn được gọi là ảnh đen trắng tương ứng với hai giá trị 0 (màu trắng) và 1 (màu đen). Ảnh xám là ảnh mà tại mỗi điểm ảnh có giá trị cường độ sáng nằm trong khoảng [0,255].

# 2. Các thuật toán xử lý hình thái học

Hai phép toán thường dùng là phép giãn nở (Dilation) và phép co (Erosion). Từ hai phép toán cơ bản này phát triển thành một số phép toán như phép đóng (Closing) và phép mở (Opening).

* Dilation gọi là D(i): giãn nở.
* Erosion gọi là E(i): xói mòn.
* Một chu trình E(i)-D(i) gọi là phép mở (Opening).
* Một chu trình D(i)-E(i) gọi là phép đóng (Closing).

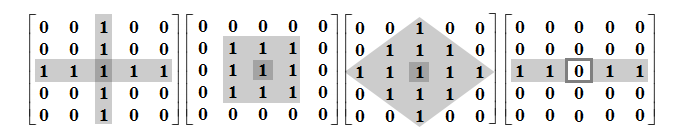
Yếu tố quan trọng trong các phép toán này là lựa chọn một phần tử cấu trúc có hình dáng phù hợp.

## 2.1. Phần tử cấu trúc (Structuring element)

Phần tử cấu trúc ảnh (Image structuring element) là một hình khối được định nghĩa sẵn nhằm tương tác với ảnh xem nó có thỏa mãn một số tính chất nào đó. Là một ma trận nhỏ có hai giá trị 0 và 1, các giá trị 0 được bỏ qua trong quá trình tính toán.

Một số hình dáng của phần tử cấu trúc thường xuyên sử dụng trên ảnh nhị phân:

* Dạng đường theo chiều ngang và dọc.
* Dạng chữ thập.
* Dạng hình vuông.
* Dạng hình ellipse.



Các thành phần với một phần tử cấu trúc:

1. Kích thước của ma trận phần tử cấu trúc.
2. Hình dáng của phần tử cấu trúc.
3. Gốc của phần tử (Ogirin). Thông thường thì gốc của phần tử mang giá trị 1 nhưng trong một số trường hợp thì gốc phần tử mang giá trị C (Phần tử cấu trúc không có gốc).

Trong các phép toán hình thái học, một phần tử cấu trúc có kích thước (NxN) được di chuyển khắp ảnh và thực hiện phép tính toán với từng điểm ảnh (Pixel) của ảnh với   
(N2 -1) điểm ảnh (Pixel) lân cận (Không tính điểm ở tâm). Phép tính toán ở đây tùy thuộc vào nội dung của phép toán hình thái học mà từ đó cho ra một kết quả phù hợp:

* Nếu gốc của phần tử cấu trúc (Ogirin) nằm ở phía bên trái thì ảnh sẽ có xu hướng co và giãn nở về phía bên phải.
* Nếu gốc của phần tử cấu trúc (Ogirin) nằm ở phía bên phải thì ảnh sẽ có xu hướng co và giãn nở về phía bên trái.

Vì vậy: Trong các kết quả của phép toán xử lý hình thái học thì yếu tố quan trọng là phần tử cấu trúc.

## 2.2. Phép toán xói mòn (Erosion)

Phép toán xói mòn (Erosion) là một trong những phép toán cơ bản hình thái học có ứng dụng trong việc giảm kích thước của đối tượng, tách rời các đối tượng gần nhau, làm mảnh và tìm xương của đối tượng.

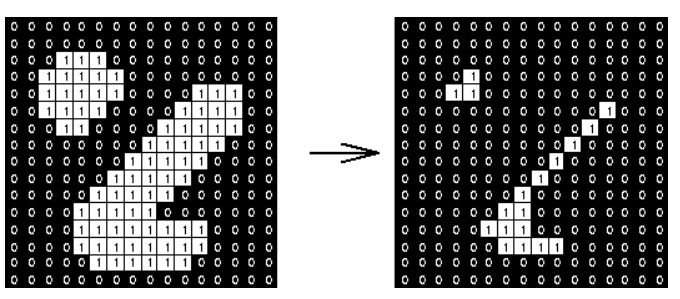
Công thức:

  
Giải thích: Shift B theo z, nếu B nằm hoàn toàn ( fit ) trong A thì output bằng 1 tại tâm của kernel.

Trong đó:

A: Ma trận điểm ảnh của ảnh nhị phân

B: Là phần tử cấu trúc ( Kernel )



Xói mòn dùng phần tử cấu trúc vuông 3x3

Ứng dụng: Tách các thành phần trong ảnh, triệt nhiễu muối,…

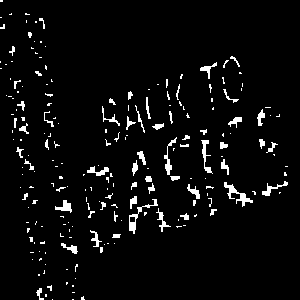


Nhiễu muối được loại bỏ, tuy nhiên ảnh bị giảm chất lượng nhiều.

**Code mô phỏng:**

import cv2  
import numpy as np  
# Đọc ảnh và xử lý ảnh ( biến ảnh màu thành ảnh nhị phân)  
img1 = cv2.imread('picture/basic.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img1, (300, 300))  
r, img = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
cv2.imshow("Original", img)  
cv2.imwrite('result/Original.png', img)  
  
#tạo một kernel là một ma trận 5x5 có tất cả các phần tử là 1  
kernel = np.ones((5,5),np.uint8)  
  
#Thực hiện phép toán erosion  
eroded = cv2.erode(img,kernel,iterations = 1)  
cv2.imshow("Erosion", eroded)  
cv2.imwrite('result/Erosion.png', eroded)  
cv2.waitKey(0)

**Kết quả:**

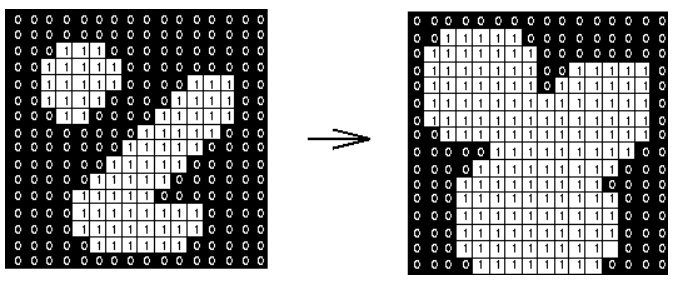
** **

## 2.3. Phép toán giãn nở (Dilation)

Là một trong các hoạt động cơ bản trong hình thái toán học. Phép toán này có tác dụng làm cho đối tượng ban đầu trong ảnh tăng lên về kích thước (Giãn nở ra).

Công thức:

  
Giải thích: Reflect B, Shift theo z, nếu ít nhất 1 pixels trong trùng với phần cảnh trong A ( B hit A ) thì output bằng 1 tại tâm của kernel.



Giãn nỡ dùng phần tử cấu trúc vuông 3x3

Ứng dụng: Lấp những lỗ trống trong ảnh, triệt nhiễu tiêu,…

Nhiễu tiêu được loại bỏ nhưng ảnh bị giảm chất lượng đáng kể.

**Code mô phỏng:**

import cv2  
import numpy as np  
# Đọc ảnh và xử lý ảnh ( biến ảnh màu thành ảnh nhị phân)  
img1 = cv2.imread('picture/basic.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img1, (300, 300))  
r, img = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
cv2.imshow("Original", img)  
cv2.imwrite('result/Original.png', img)  
#tạo một kernel là một ma trận 5x5 có tất cả các phần tử là 1  
kernel = np.ones((5,5),np.uint8)  
  
#Thực hiện phép toán dilation  
dilated = cv2.dilate(img,kernel,iterations = 1)  
cv2.imshow("Dilation", dilated)  
cv2.imwrite('result/Dilation.png', dilated)  
cv2.waitKey(0)

**Kết quả mô phỏng:**

** **

**⁕ Tính đối ngẫu giữa xói mòn và giản nở:**



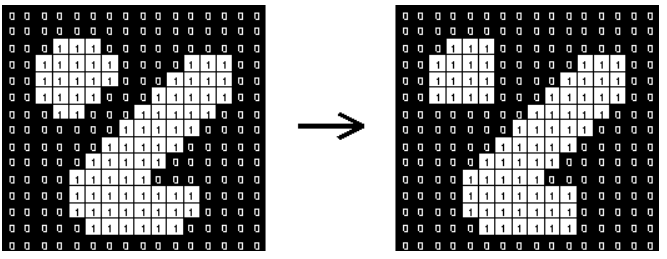
## 2.4. Phép toán mở (Opening):

Phép toán mở (Opening) và phép toán đóng (Closing) là sự kết hợp của phép xói mòn (Erosion) và phép giãn nở (Dilation) như trên.

Thực hiện phép co (Erosion) trước, sau đó thực hiện phép giãn nở (Dilation) với cùng 1 phần tử cấu trúc.

Công thức:

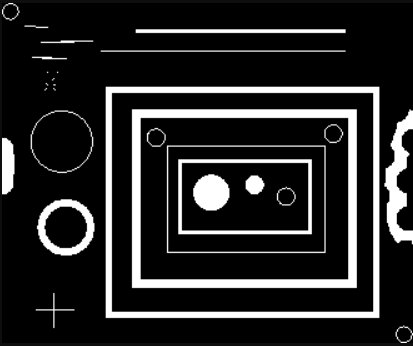
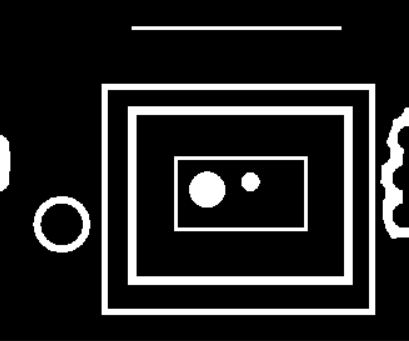




Phép mở rộng với phần tử cấu trúc vuông 3x3

Ứng dụng:

Phép toán mở (Opening) được ứng dụng trong việc loại bỏ các phần lồi lõm và làm cho đường bao các đối tượng trong ảnh trở nên mượt mà hơn.

Loại bỏ những phần tử nhỏ, giữ lại những phần tử lớn với kích thước không đổi



Loại bỏ nhiễu muối, ảnh ít bị suy giảm chất lượng so với khi dùng phép xói mòn (Erosion).

**Code mô phỏng:**

import cv2  
import numpy as np  
# Đọc ảnh và xử lý ảnh ( biến ảnh màu thành ảnh nhị phân)  
img1 = cv2.imread('picture/basic.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img1, (300, 300))  
r,img = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
cv2.imshow("Original", img)  
cv2.imwrite('result/Original.png', img)  
  
#tạo một kernel là một ma trận 5x5 có tất cả các phần tử là 1  
kernel = np.ones((5,5),np.uint8)  
  
#Thực hiện phép toán Opening  
opened = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)  
cv2.imshow("Opening", opened)  
cv2.imwrite('result/Opening.png', opened)  
cv2.waitKey(0)

**Kết quả mô phỏng:**

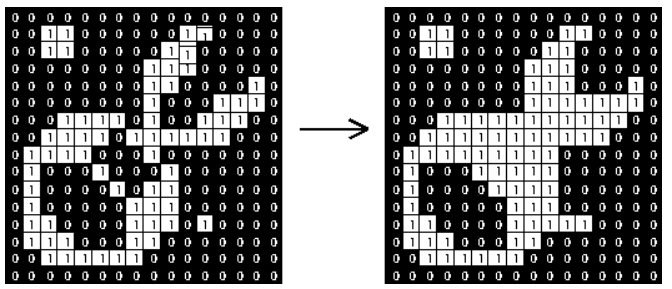
** **

## 2.5. Phép toán đóng (Closing):

Thực hiện phép giãn nở (Dilation) trước sau đó mới thực hiện phép co (Erosion).

Công thức:

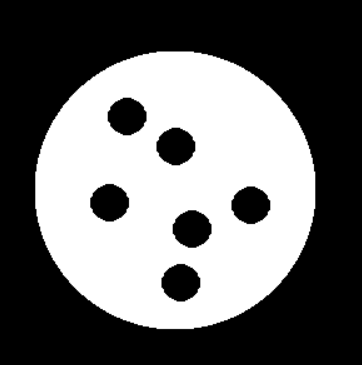
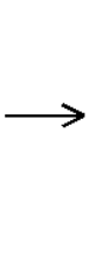
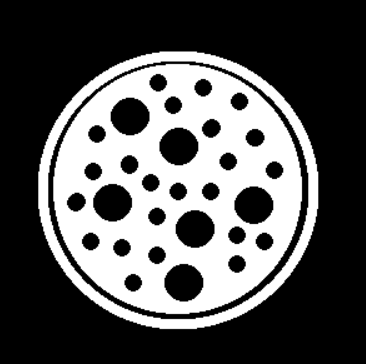
ss_12

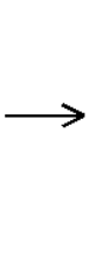


Phép toán đóng dùng phần tử cấu trúc vuông 3x3

Ứng dụng:

Phép toán đóng (Closing) được dùng trong ứng dụng làm trơn đường bao các đối tượng, lấp đầy các khoảng trống biên và loại bỏ những hố nhỏ.





Loại bỏ nhiễu tiêu, ảnh ít bị suy giảm chất lượng so với khi dùng phép giãn nở (Dilation).

**Code mô phỏng:**

import cv2  
import numpy as np  
# Đọc ảnh và xử lý ảnh ( biến ảnh màu thành ảnh nhị phân)  
img1 = cv2.imread('picture/basic.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img1, (300, 300))  
r,img = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
cv2.imshow("Original", img)  
cv2.imwrite('result/Original.png', img)  
  
#tạo một kernel là một ma trận 5x5 có tất cả các phần tử là 1  
kernel = np.ones((5,5),np.uint8)  
  
##Thực hiện phép toán Closing  
closed = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)  
cv2.imshow("Closing", closed)  
cv2.imwrite('result/Closing.png', closed)  
cv2.waitKey(0)

**Kết quả mô phỏng:**

** **

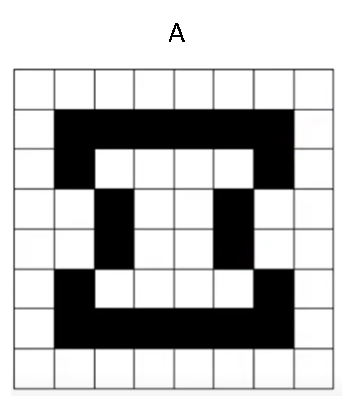
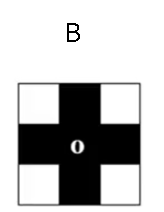
## 2.6. Lấp đầy vùng (Region fill) :

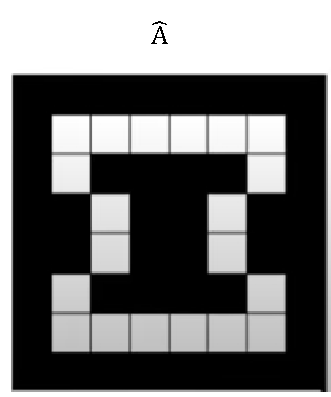
Là thuật toán hình thái học dùng để lấp đầy các vùng trong ảnh, dựa trên các phép giãn nở, đảo ảnh, giao ảnh.

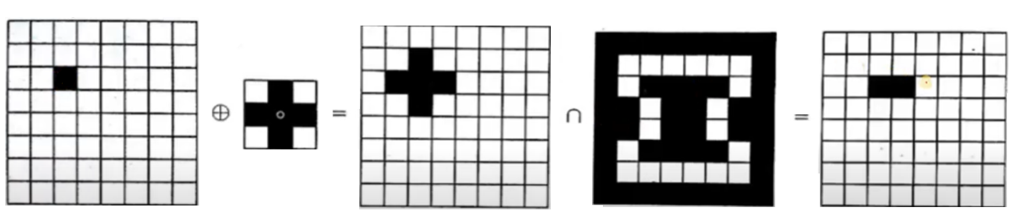
Công thức:

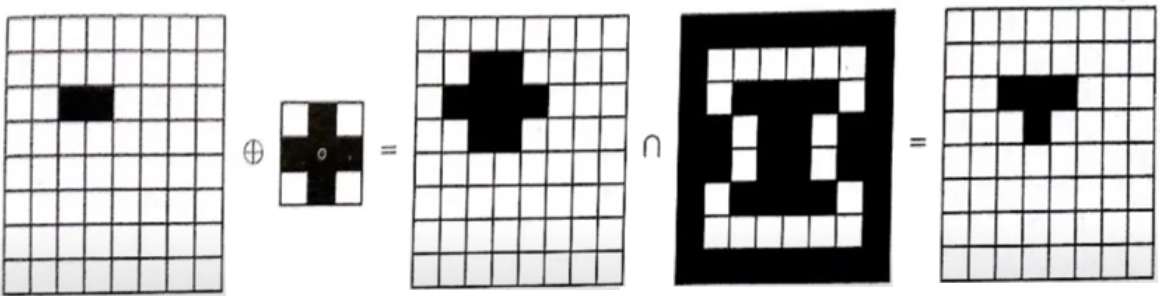


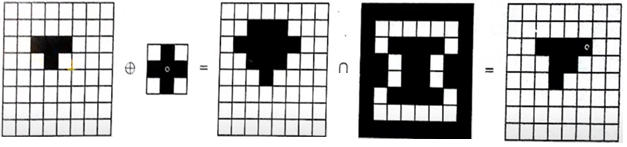
Phép toán dừng tại vòng lặp thứ k khi Xk = Xk-1.

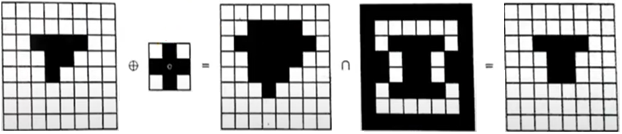
  

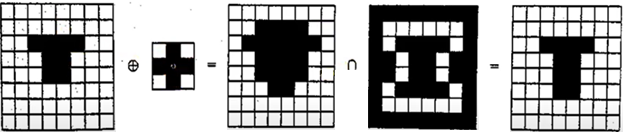
****

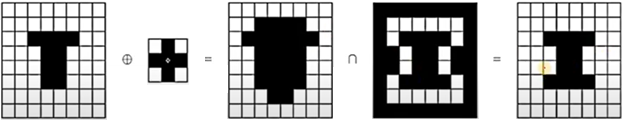
****

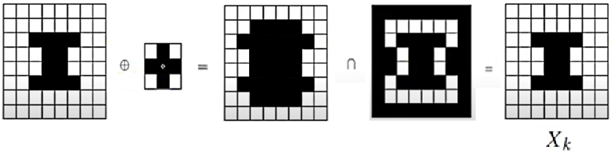
****

****

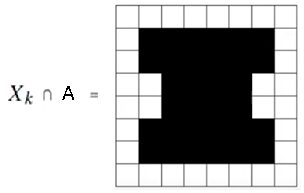
****

****

****

****

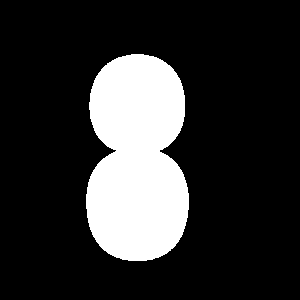
Lúc này giao của X­­k với A cho ảnh ngõ ra được lấp đầy vùng trống:



**Code mô phỏng:**

import cv2  
import numpy as np  
  
img = cv2.imread("picture/eight.png", cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img, (300, 300))  
thresh, biimg = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
  
floodimg = biimg.copy()  
  
y, x = biimg.shape[:2]  
kernel = np.zeros((y+2, x+2), np.uint8)  
  
cv2.floodFill(floodimg, kernel, (0,0), 255)  
invfloodimg = cv2.bitwise\_not(floodimg)  
outimg = cv2.bitwise\_or(biimg,invfloodimg)  
  
cv2.imshow("Original image", img)  
cv2.imwrite('result/Original.png', img)  
cv2.imshow("Output filled image", outimg)  
cv2.imwrite('result/RegionFill.png', outimg)  
  
cv2.waitKey(0)

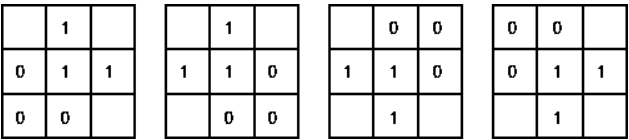
**Kết quả mô phỏng:**

## 2.7. Hit-and-miss transformation:

Thuật toán hit-and-miss là phép biến đổi hình thái học có thể dùng để tìm những chuỗi cụ thể từ nền và cảnh trong ảnh, là phép toán cơ bản của hình thái học, gần như tất cả các phép toán khác đều có thể được xây dựng từ đây.

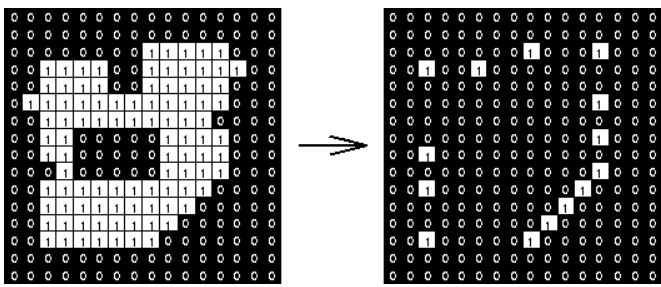
Nguyên lý: Khác với phần tử cấu trúc trong Erosion và Dilation chỉ quan tâm đến các giá trị “1” còn các giá trị “0” (xem như các ô trống) được bỏ qua, phần tử cấu trúc trong hit-and-miss chứa cả pixels nền và cảnh, nghĩa là quan tâm đến cả 2 các giá trị “1” và “0”.



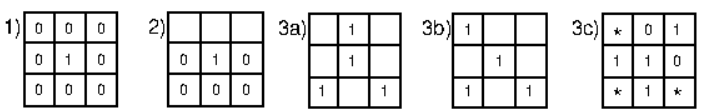
Ví dụ phần tử cấu trúc của hit-and-miss để xác định các pixel ở góc vuông, “0” là nền, “1” là cảnh, không quan tâm đến các ô trống. Các phần tử cấu trúc là như nhau nhưng đươc quay các góc khác nhau.

Dịch chuyển phần tử cấu trúc đến tất cả điểm ảnh, nếu giá trị nền và cảnh trong phần tử cấu trúc trùng hoàn toàn (fit) với giá trị nền và cảnh trong ảnh thì pixel của ảnh ở tâm phần tử cấu trúc được gán giá trị của cảnh, ngược lại gán cho pixel đó giá trị nền.

Sau khi áp dụng 4 phần tử cấu trúc cho ảnh, chúng ta chỉ cần OR những ảnh ngõ ra với nhau để được kết quả mong muốn.



Phát hiện các pixel ở góc vuông (3 mặt giáp nền) dùng hit-and-miss.



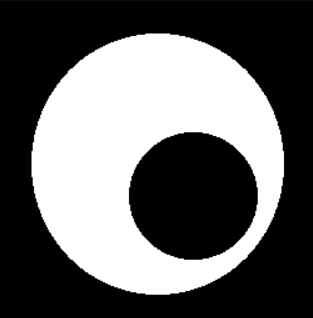
**1** được dùng để xác định những điểm tách biệt trong ảnh nhị phân.

**2** được để xác định các điểm cuối trong ảnh skeleton.

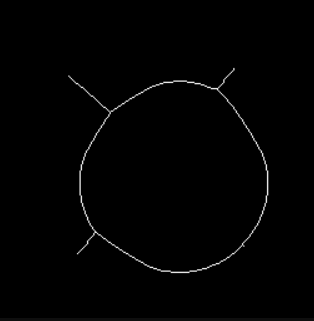
**3a**, **3b** và **3c** dùng để xác định các điểm nối 3 nhánh trong 1 skeleton.

Với mỗi phần tử cấu trúc, ta có 4 phiên bản của nó được xoay với các góc khác nhau. Áp dụng tất cả các phần tử này lên ảnh, tức là với mỗi phần tử cấu trúc ta thực hiện 4 lần hit-and-miss.

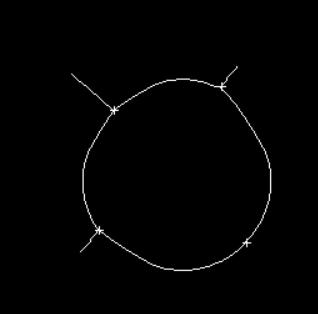
Ví dụ cho ảnh:



Có skeleton:



Ảnh ngõ ra:



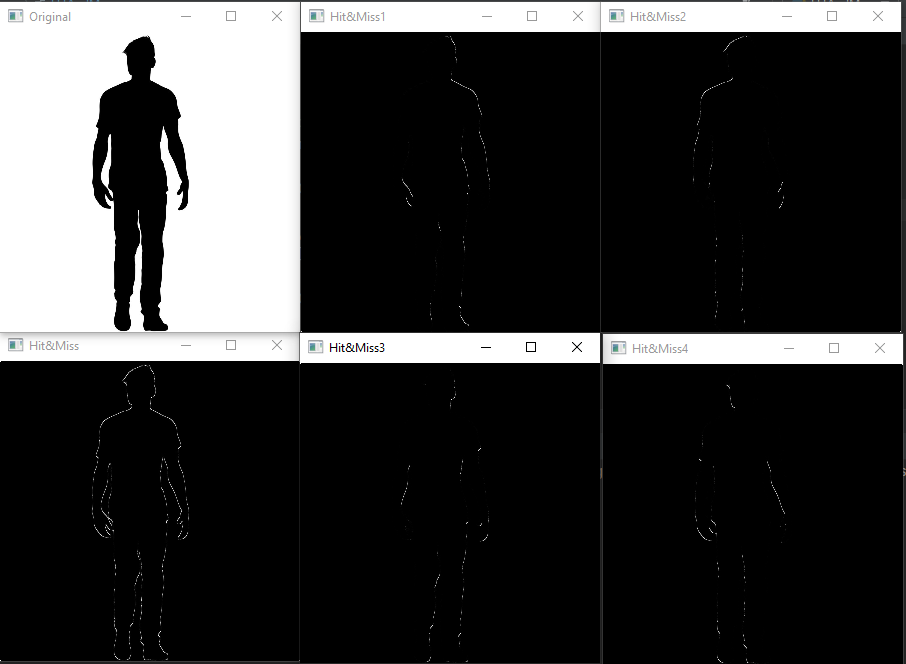
cho thấy các điểm nối 3 nhánh của skeleton trên, trong hình này các điểm nối 3 nhánh sau khi được trích xuất ra thì được Dilated sau đó OR với skeleton gốc để thuận tiện cho quan sát.

**Code mô phỏng:**

import cv2  
import numpy as np  
  
img = cv2.imread('picture/man.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img, (300, 300))  
  
kernel1 = np.array((  
 [0, 1, 0],  
 [-1, 1, 1],  
 [-1, -1, 0]), dtype="int")  
  
kernel2 = np.array((  
 [0, 1, 0],  
 [1, 1, -1],  
 [0, -1, -1]), dtype="int")  
  
  
kernel3 = np.array((  
 [-1, -1, 0],  
 [-1, 1, 1],  
 [0, 1, 0]), dtype="int")  
  
kernel4 = np.array((  
 [0, -1, -1],  
 [1, 1, -1],  
 [0, 1, 0]), dtype="int")

outimg1 = cv2.morphologyEx(img, cv2.MORPH\_HITMISS, kernel1)  
outimg2 = cv2.morphologyEx(img, cv2.MORPH\_HITMISS, kernel2)  
outimg3 = cv2.morphologyEx(img, cv2.MORPH\_HITMISS, kernel3)  
outimg4 = cv2.morphologyEx(img, cv2.MORPH\_HITMISS, kernel4)  
  
outimg = outimg1 + outimg2 + outimg3 + outimg4  
  
cv2.imshow("Original", img)  
cv2.imshow("Hit&Miss1", outimg1)  
cv2.imshow("Hit&Miss2", outimg2)  
cv2.imshow("Hit&Miss3", outimg3)  
cv2.imshow("Hit&Miss4", outimg4)  
cv2.imshow("Hit&Miss", outimg)  
  
cv2.waitKey(0)

**Kết quả mô phỏng:**



## 2.8. Làm mỏng (Thinning):

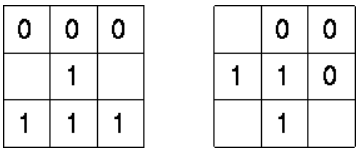
Làm mỏng là phép toán hình thái học dùng để loại bỏ những pixel được chọn của vùng cảnh từ ảnh nhị phân.

Công thức: Cho ảnh I và phần tử cấu trúc J.

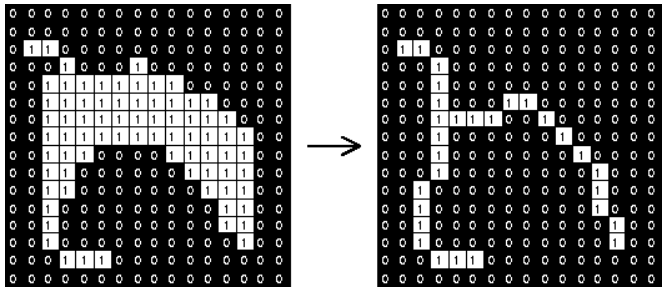


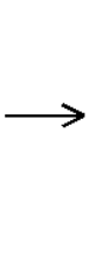
Giải thích: Dịch chuyển phần tử cấu trúc, nếu trùng hoàn toàn với mỗi pixel trong ảnh, gán giá trị nền tại tâm của phần tử cấu trúc, ngược lại gán giá trị cảnh.

Việc lựa chọn phần tử cấu trúc quyết định tác dụng cụ thể của thuật toán làm mỏng.

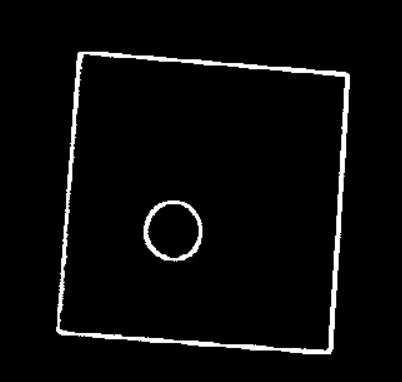
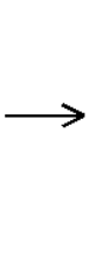
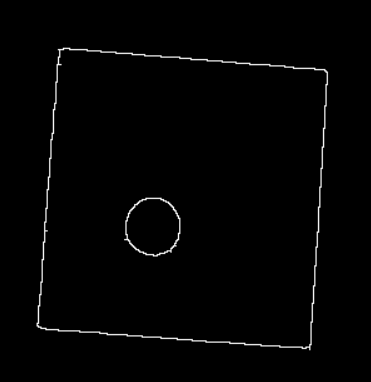


Phần tử cấu trúc dùng để thực hiện skeletonization bằng thuật toán làm mỏng. Tại mỗi lần thực hiện, lần lượt áp dụng phần tử bên trái, sau đó đến là bên phải, thực hiện đủ số vong flặp cho 4 phiên bản xoay của mỗi phần tử.



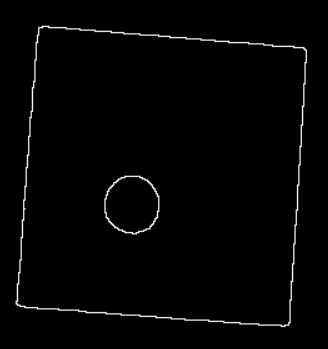
  

Ví dụ về skeletonization dùng thuật toán làm mỏng trên ảnh nhị phân, sử dụng các phần tử cấu trúc trên.

Dùng thuật toán làm mỏng lặp lại liên tục đến khi ảnh ngõ ra không thay đổi (hội tụ).

Thuật toán làm mỏng có ứng dụng rộng rãi. Trong một số trường hợp thuật toán gây ra những spur không mong muốn, khi đó cần áp dụng prunning cũng một số thuật toán khác như Dilation để loại bỏ các phần tử này.

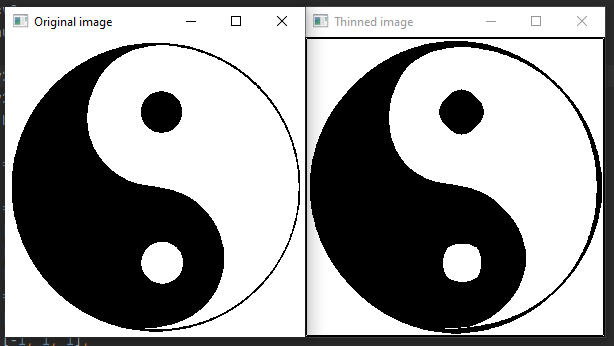


Kết quả của việc áp dụng prunning trên ảnh sau khi làm mỏng, các điểm cảnh không mong muốn bị loại bỏ.

**Code mô phỏng:**

import cv2  
import numpy as np  
  
img = cv2.imread("picture/YinYang.png", cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img, (300, 300))  
thresh, biimg = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
  
thinimg = biimg.copy()  
  
kernel1 = np.array((  
 [-1, -1, -1],  
 [0, 1, 0],  
 [1, 1, 1]), dtype="int")  
kernel2 = np.array((  
 [-1, 0, 1],  
 [-1, 1, 1],  
 [-1, 0, 1]), dtype="int")  
kernel3 = np.array((  
 [1, 1, 1],  
 [0, 1, 0],  
 [-1, -1, -1]), dtype="int")  
kernel4 = np.array((  
 [1, 0, -1],  
 [1, 1, -1],  
 [1, 0, -1]), dtype="int")  
  
for i in range(1,3):  
 hm1 = cv2.morphologyEx(thinimg, cv2.MORPH\_HITMISS, kernel1)  
 hm2 = cv2.morphologyEx(thinimg, cv2.MORPH\_HITMISS, kernel2)  
 hm3 = cv2.morphologyEx(thinimg, cv2.MORPH\_HITMISS, kernel3)  
 hm4 = cv2.morphologyEx(thinimg, cv2.MORPH\_HITMISS, kernel4)  
 hm = hm1 + hm2 + hm3 + hm4  
 thinimg = thinimg - hm  
  
cv2.imshow('Original image', biimg)  
cv2.imshow('Thinned image', thinimg)  
  
cv2.waitKey(0)

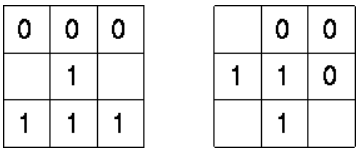
**Kết quả mô phỏng:**



## 2.9. Skeletonization:

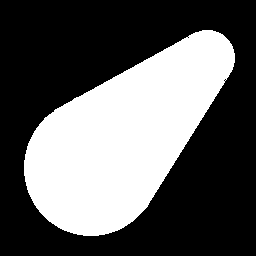
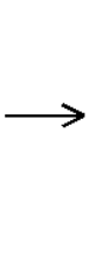
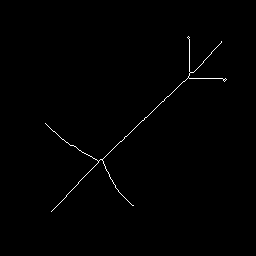
Skeletonization là kỹ thuật làm tiêu giảm vùng cản trong ảnh nhị phân thành 1 ảnh chỉ giữ lại phần “xương”, nghĩa là giữ nguyên độ dài và liên kết giữ các phần của cảnh.

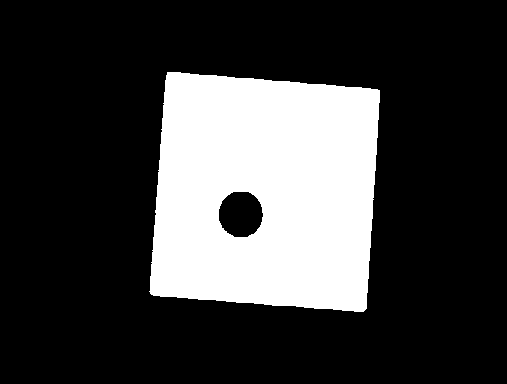
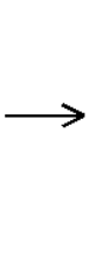
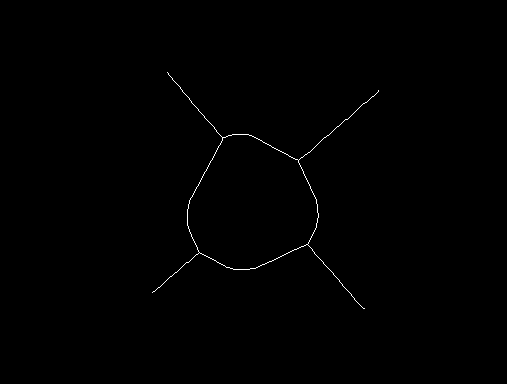
Thuật toán này được thực hiện bằng thuật toán làm mỏng (như đã trình bày sơ lược ở phần trước).

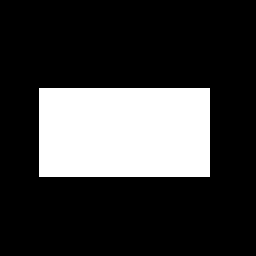
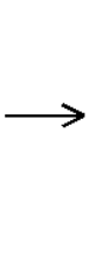
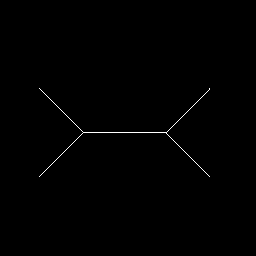


Phần tử cấu trúc dùng để thực hiện skeletonization bằng thuật toán làm mỏng. Tại mỗi lần thực hiện, lần lượt áp dụng phần tử bên trái, sau đó đến là bên phải, thực hiện đủ số vong flặp cho 4 phiên bản xoay của mỗi phần tử.

Ứng dụng: Skeletonization biểu diễn vật thể một cách đơn giản và súc tích, giữ lại những đặc tính về kích thước và cấu trúc liên kết (topology).

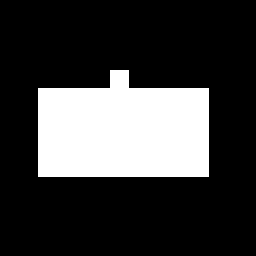
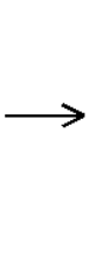
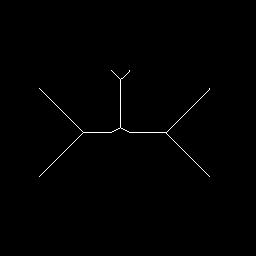
  

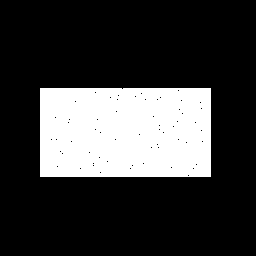
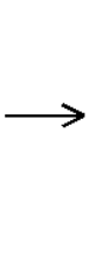
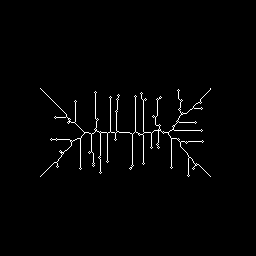
Một vài ví dụ cho biến đổi skeleton.

Skeletonization rất nhạy cảm với nhiễu và các thay đổi nhỏ trong ảnh. Sự khác biệt nhỏ trong ảnh có thể tạo ra sự khác biệt lớn ở kết quả.

Ví dụ, với 1 chút thay đổi ở hình vuông trên, ngõ ra sẽ có sự khác biệt đáng kể.

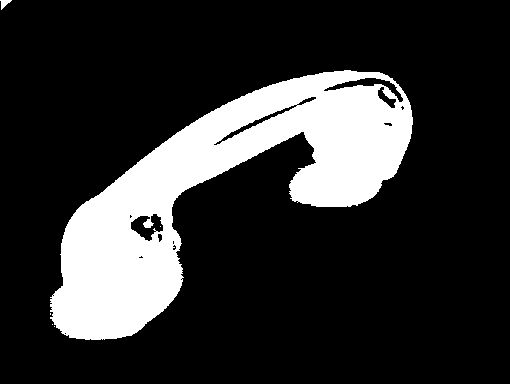
  

Với 1 hình ảnh bị nhiễu.

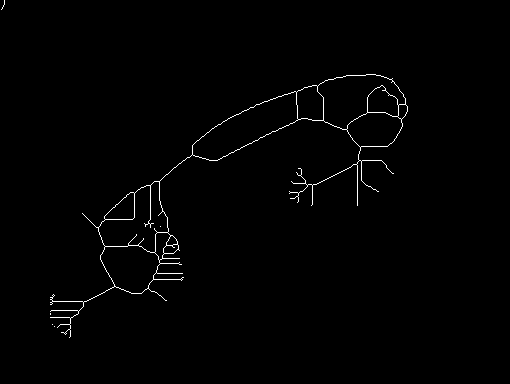
  

Với ảnh xám hay ảnh màu, việc phân ngưỡng ảnh để có ảnh nhị phân khiến cho skeleton của ảnh có cấu trúc phức tạp.

Ảnh gốc: Ảnh grayscale:

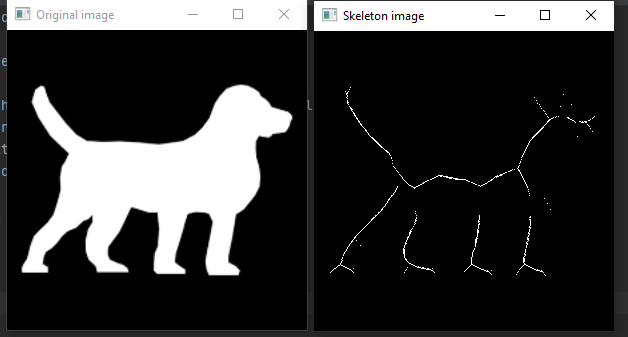
Skeleton:



**Code mô phỏng:**

import cv2  
import numpy as np  
  
img = cv2.imread('picture/dog.png', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img, (300, 300))  
thresh,img1 = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
  
skelimg = np.zeros(img.shape, np.uint8)  
  
kernel = cv2.getStructuringElement(cv2.MORPH\_CROSS, (3,3))  
  
while (cv2.countNonZero(img1) != 0):  
  
 white = cv2.morphologyEx(img1, cv2.MORPH\_TOPHAT, kernel)  
 erodeimg = cv2.erode(img1, kernel)  
 skelimg = cv2.bitwise\_or(skelimg,white)  
 img1 = erodeimg.copy()  
  
cv2.imshow("Original image", img)  
cv2.imshow("Skeleton image",skelimg)  
  
cv2.waitKey(0)

**Kết quả mô phỏng:**



## 2.10. Làm dày (Thickening):

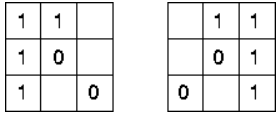
Làm dày là thuật toán hình thái học dùng để gia tăng kích thước vùng cảnh được chọn. Khá giống với thuật toán giãn nỡ (Dilation) và đóng (Closing).

Công thức: Với ảnh I và phần tử cấu trúc J.

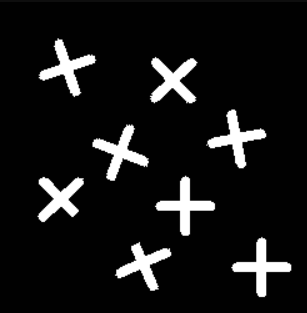
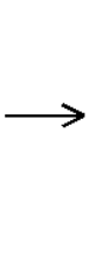
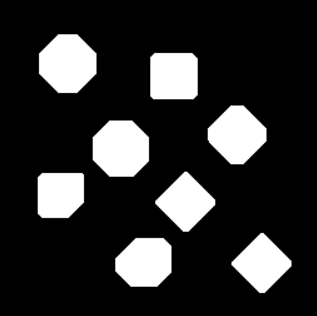


Dịch chuyển phần tử cấu trúc đến tất cả các pixel trong ảnh, nếu trùng khớp hoàn toàn, gán gia trị cảnh cho pixel tại tâm của phần tử cấu trúc, ngược lại giữ nguyên giá trị của pixel trong ảnh. Tâm phần tử cấu trúc của thuật toán làm dày có giá trị nền để mở rộng phần cảnh bằng cách gán giá trị cảnh cho nền nếu thỏa điều kiện (fit).

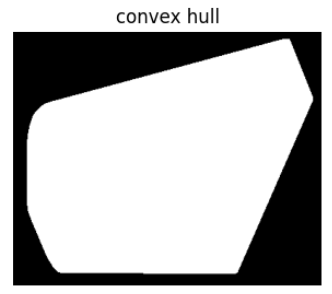
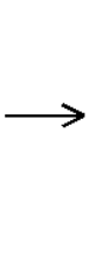
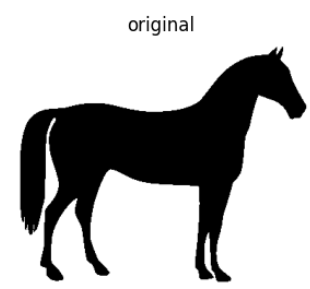
Ứng dụng: Bao phần lõm (convex hull), và skeleton vùng ảnh hưởng.



Ví dụ cho phần tử cấu trúc dùng cho convex hull.

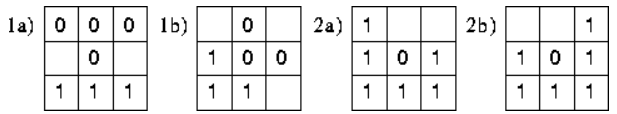
  

Làm dày ảnh dùng 2 phần tử cấu trúc ở trên. Lặp lại với 4 phiên bản xoay của mỗi phần tử.

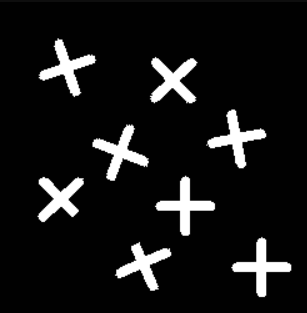
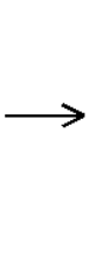
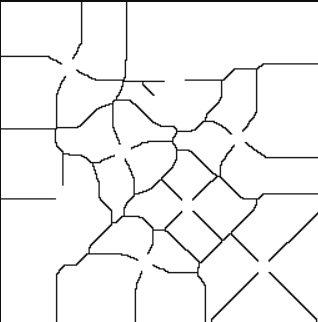


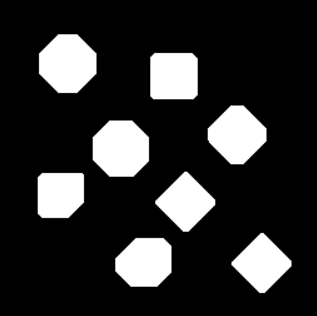
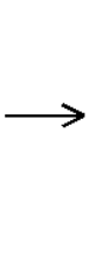
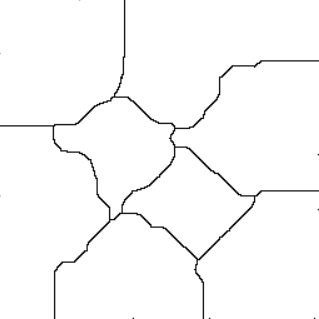
Một ví dụ khác về Convex hull

Convex hull giống như bọc một vật thể thực tế bằng giấy nilon, kết quả là ảnh chụp của vật thể với phần bên trong đường bao được lấp đầy. Phàn lõm được lấp, đường bao là đường nối giữa các đỉnh nhô ra của vật thể.



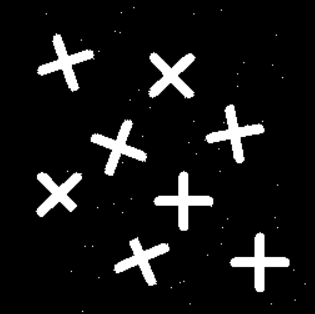
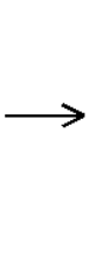
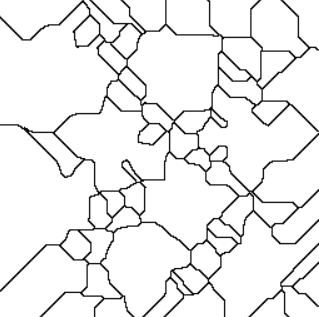
Phần tử cấu trúc để thực hiện skeleton vùng ảnh hưởng.

Ví dụ về skeleton vùng ảnh hưởng. (SKIZ) 1a và 1b dùng để làm dày ảnh và được lặp đến khi hội tụ. Sau đó, 2a và 2b được dùng để prun đến khi hội tụ để được ảnh SKIZ.

Skeleton vùng ảnh huởng xem mỗi pixel cảnh là 1 vật thể (object), mỗi vật thể được gắn với 1 vùng, nên rất nhạy với nhiễu.

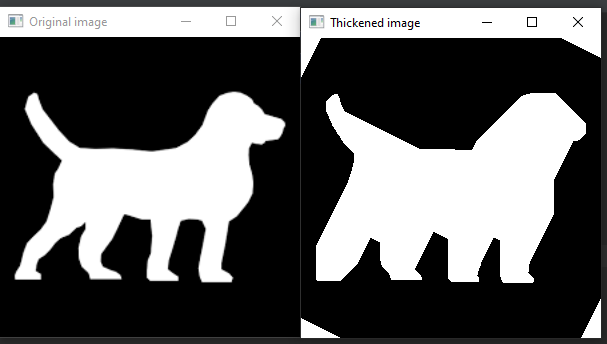
  

Ví dụ về SKIZ của ảnh bị nhiễu muối.

**Code mô phỏng:**

import cv2  
import numpy as np  
  
img = cv2.imread("picture/dog.png", cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img, (300, 300))  
thresh, biimg = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
  
thickimg = biimg.copy()  
  
kernel1 = np.array((  
 [1, 1, 0],  
 [1, -1, 0],  
 [1, 0, -1]), dtype="int")  
kernel2 = np.array((  
 [0, 0, -1],  
 [1, -1, 0],  
 [1, 1, 1]), dtype="int")  
kernel3 = np.array((  
 [-1, 0, 1],  
 [0, -1, 1],  
 [0, 1, 1]), dtype="int")  
kernel4 = np.array((  
 [1, 1, 1],  
 [0, -1, 1],  
 [-1, 0, 0]), dtype="int")  
  
for i in range(1,40):  
 hm1 = cv2.morphologyEx(thickimg, cv2.MORPH\_HITMISS, kernel1)  
 hm2 = cv2.morphologyEx(thickimg, cv2.MORPH\_HITMISS, kernel2)  
 hm3 = cv2.morphologyEx(thickimg, cv2.MORPH\_HITMISS, kernel3)  
 hm4 = cv2.morphologyEx(thickimg, cv2.MORPH\_HITMISS, kernel4)  
 hm = hm1 + hm2 + hm3 + hm4  
 thickimg = cv2.bitwise\_or(thickimg, hm)  
  
cv2.imshow('Original image', img)  
cv2.imshow('Thickened image', thickimg)  
  
cv2.waitKey(0)

**Kết quả mô phỏng:**

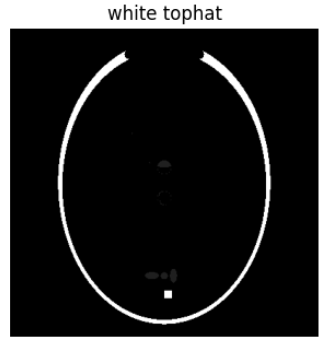
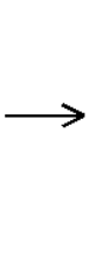
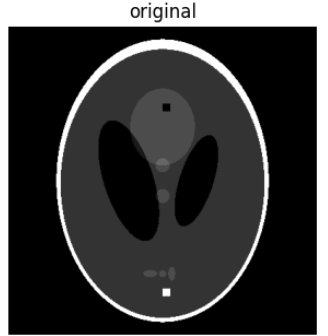


## 2.11. White tophat:

Thuật toán White-hat áp dụng phép mở (Opening) lên ảnh, ảnh ngõ ra là kết quả của ảnh gốc trừ cho ảnh thu được từ phép mở.

Công thức: Cho ảnh A, phần tử cấu trúc B.



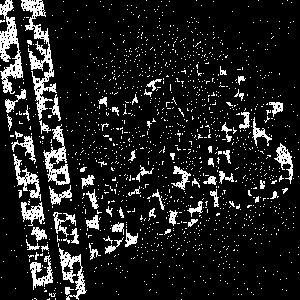


Áp dụng phép Whitehat lên ảnh, ảnh ngõ ra cho ta chỉ những thành phần sáng có kích thước nhỏ hơn 10 pixel (kích thước của phần tử cấu trúc).

**Code mô phỏng:**

import cv2  
import numpy as np  
# Đọc ảnh và xử lý ảnh ( biến ảnh màu thành ảnh nhị phân)  
img1 = cv2.imread('picture/basic.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img1, (300, 300))  
r,img = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
cv2.imshow("Original", img)  
cv2.imwrite('result/Original.png', img)  
  
#tạo một kernel là một ma trận 5x5 có tất cả các phần tử là 1  
kernel = np.ones((5,5),np.uint8)  
  
#Thực hiện phép toán Top hat  
top\_hat = cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, kernel)  
cv2.imshow("Top Hat", top\_hat)  
cv2.imwrite('result/Top\_hat.png', top\_hat)  
cv2.waitKey(0)

**Kết quả mô phỏng:**

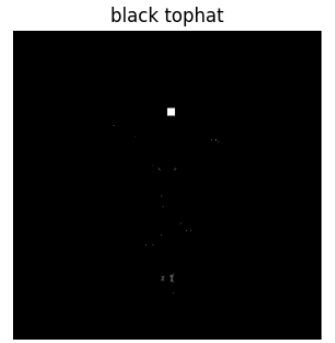
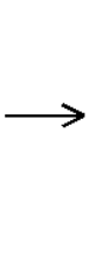
** **

## 2.12. Black hat:

Thuật toán Black-hat áp dụg phép đóng (Closing) lên ảnh. Ảnh ngõ ra là kết quả của ảnh có được từ phép đóng trừ cho ảnh gốc.

Công thức: Cho ảnh A, phần tử cấu trúc B.



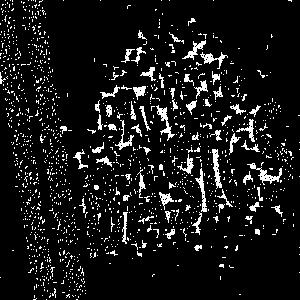


Áp dụng phép Black-tophat lên ảnh, ảnh ngõ ra chi hiển thị những thành phần tối có kích thước nhỏ hơn 10 pixel (kích thước của phần tử cấu trúc).

**Code mô phỏng:**

import cv2  
import numpy as np  
# Đọc ảnh và xử lý ảnh ( biến ảnh màu thành ảnh nhị phân)  
img1 = cv2.imread('picture/basic.jpg', cv2.IMREAD\_GRAYSCALE)  
img = cv2.resize(img1, (300, 300))  
r,img = cv2.threshold(img, 128, 255, cv2.THRESH\_OTSU)  
cv2.imshow("Original", img)  
cv2.imwrite('result/Original.png', img)  
  
#tạo một kernel là một ma trận 5x5 có tất cả các phần tử là 1  
kernel = np.ones((5,5),np.uint8)  
  
#Thực hiện phép toán Black Hat  
black\_hat = cv2.morphologyEx(img, cv2.MORPH\_BLACKHAT, kernel)  
cv2.imshow("Black Hat Image", black\_hat)  
cv2.imwrite('result/Black\_hat.png', black\_hat)  
  
cv2.waitKey(0)

**Kết quả mô phỏng:**

****  ****

--HẾT--