Họ tên : Nguyễn Quang Anh

Mã sinh viên : B21DCDT036

CHƯƠNG 6 : XỬ LÝ ẢNH HÌNH THÁI HỌC ( Morphological Image Processing )

Trong chương này, chúng ta sẽ thảo luận về toán học hình thái học và xử lý ảnh hình thái học. **Xử lý ảnh hình thái học** là tập hợp các phép toán phi tuyến liên quan đến hình dạng hoặc cấu trúc của các đặc điểm trong ảnh. Những phép toán này đặc biệt phù hợp với việc xử lý **ảnh nhị phân** (nơi các pixel được biểu diễn dưới dạng 0 hoặc 1, với quy ước: đối tượng tiền cảnh = 1 hoặc màu trắng, và nền = 0 hoặc màu đen), mặc dù chúng cũng có thể được mở rộng để áp dụng trên ảnh mức xám.

THUẬT TOÁN 1 : PHÉP CO ẢNH ( EROSION )

**1. Giới thiệu thuật toán Co rút (Erosion):**

* Co rút là một phép toán hình thái học giúp giảm kích thước của vùng tiền cảnh (foreground) trong ảnh nhị phân.
* Nguyên tắc chính:
  + Với mỗi pixel trong ảnh, so sánh vùng lân cận của nó (được xác định bởi phần tử cấu trúc) với phần tử cấu trúc.
  + Nếu toàn bộ phần tử cấu trúc **khớp hoàn toàn** với vùng lân cận, pixel trung tâm sẽ giữ nguyên giá trị.
  + Ngược lại, pixel trung tâm sẽ bị đặt về giá trị 0 (nền).
* Ứng dụng:
  + Loại bỏ các chi tiết nhỏ không mong muốn.
  + Làm mượt biên của đối tượng.
  + Chuẩn bị cho các phép toán hình thái học khác như mở rộng (dilation), mở (opening), và đóng (closing).

**Phần tử cấu trúc (Structuring Element):**

* Là một mẫu nhỏ, có dạng hình học (ví dụ: hình chữ nhật, hình vuông, hình tròn).
* Trong ví dụ trên:
  + Phần tử cấu trúc là các hình chữ nhật kích thước **(1,5)** và **(1,15)**.
  + Hình chữ nhật dọc hẹp loại bỏ các vạch nhỏ và cao hơn thì xóa cả các kim đồng hồ.

#### ****2. Các bước thực hiện thuật toán Co rút:****

**B1 : Chuẩn bị ảnh:**

* + Chuyển ảnh về ảnh nhị phân (binary) bằng cách áp dụng ngưỡng cố định (thresholding). Trong ví dụ: im[im <= 0.5] = 0, im[im > 0.5] = 1.

**B2: Xác định phần tử cấu trúc:**

* + Dùng phần tử cấu trúc phù hợp với đặc điểm cần xử lý. Trong ví dụ: rectangle(1, 5) và rectangle(1, 15).

**B3: Áp dụng phép toán:**

* + Dùng hàm **binary\_erosion()** để thực hiện phép toán co rút:
    - Kiểm tra vùng lân cận của từng pixel có khớp với phần tử cấu trúc hay không.
    - Nếu không khớp, pixel trung tâm bị đặt thành giá trị nền (0).

**B4: Hiển thị kết quả:**

* + So sánh ảnh gốc và ảnh sau khi xử lý với các phần tử cấu trúc khác nhau.

**3. Giải thích đoạn mã nguồn :**

from skimage.io import imread  
from skimage.color import rgb2gray  
import matplotlib.pylab as pylab  
from skimage.morphology import binary\_erosion, rectangle  
def plot\_image(image, title=''):  
 pylab.title(title, size=20), pylab.imshow(image)  
 pylab.axis('off') # comment this line if you want axis ticks  
im = rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\clock2.jpg"))  
im[im <= 0.5] = 0 # create binary image with fixed threshold 0.5  
im[im > 0.5] = 1  
pylab.gray()  
pylab.figure(figsize=(20,10))  
pylab.subplot(1,3,1), plot\_image(im, 'original')  
im1 = binary\_erosion(im, rectangle(1,5))  
pylab.subplot(1,3,2), plot\_image(im1, 'erosion with rectangle size (1,5)')  
im1 = binary\_erosion(im, rectangle(1,15))  
pylab.subplot(1,3,3), plot\_image(im1, 'erosion with rectangle size (1,15)')  
pylab.show()

**Import thư viện và hàm :**

from skimage.morphology import binary\_erosion, rectangle

* **imread**: Đọc ảnh từ file.
* **rgb2gray**: Chuyển ảnh màu RGB sang ảnh xám.
* **binary\_erosion**: Hàm thực hiện phép toán co rút (erosion) trên ảnh nhị phân.
* **rectangle**: Tạo phần tử cấu trúc (structuring element) có dạng hình chữ nhật.
* **pylab**: Dùng để hiển thị ảnh và đồ họa.

**.Định nghĩa hàm hiển thị ảnh :**

* **plot\_image**: Hàm hiển thị ảnh với tiêu đề.
  + **pylab.title()**: Đặt tiêu đề cho ảnh.
  + **pylab.imshow()**: Hiển thị ảnh.
  + **pylab.axis('off')**: Ẩn trục tọa độ của đồ thị.

**Đọc và tiền xử lý ảnh :**

* **imread**: Đọc ảnh gốc từ đường dẫn.
* **rgb2gray**: Chuyển ảnh từ RGB sang ảnh xám.
* **Tạo ảnh nhị phân**:
  + **Pixel ≤ 0.5**: Gán giá trị 0 (màu đen - nền).
  + **Pixel > 0.5**: Gán giá trị 1 (màu trắng - tiền cảnh).

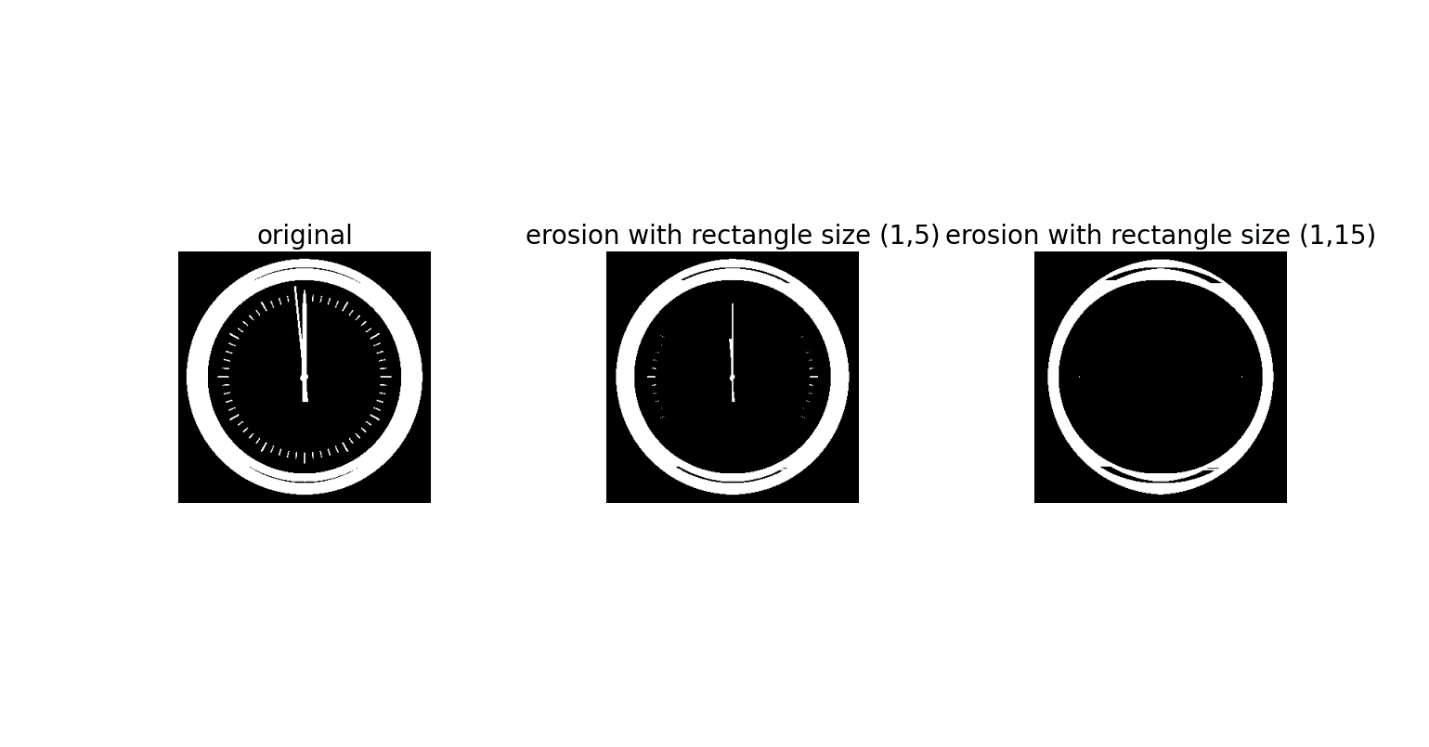
**Hiển thị ảnh gốc :**

* **pylab.gray()**: Thiết lập chế độ hiển thị ảnh dạng xám.
* **pylab.figure(figsize=(20, 10))**: Tạo khung hiển thị với kích thước 20x10 inch.
* **pylab.subplot(1, 3, 1)**: Định nghĩa vùng hiển thị đầu tiên trong lưới (1 hàng, 3 cột, vị trí 1).
* **plot\_image(im, 'original')**: Hiển thị ảnh gốc.

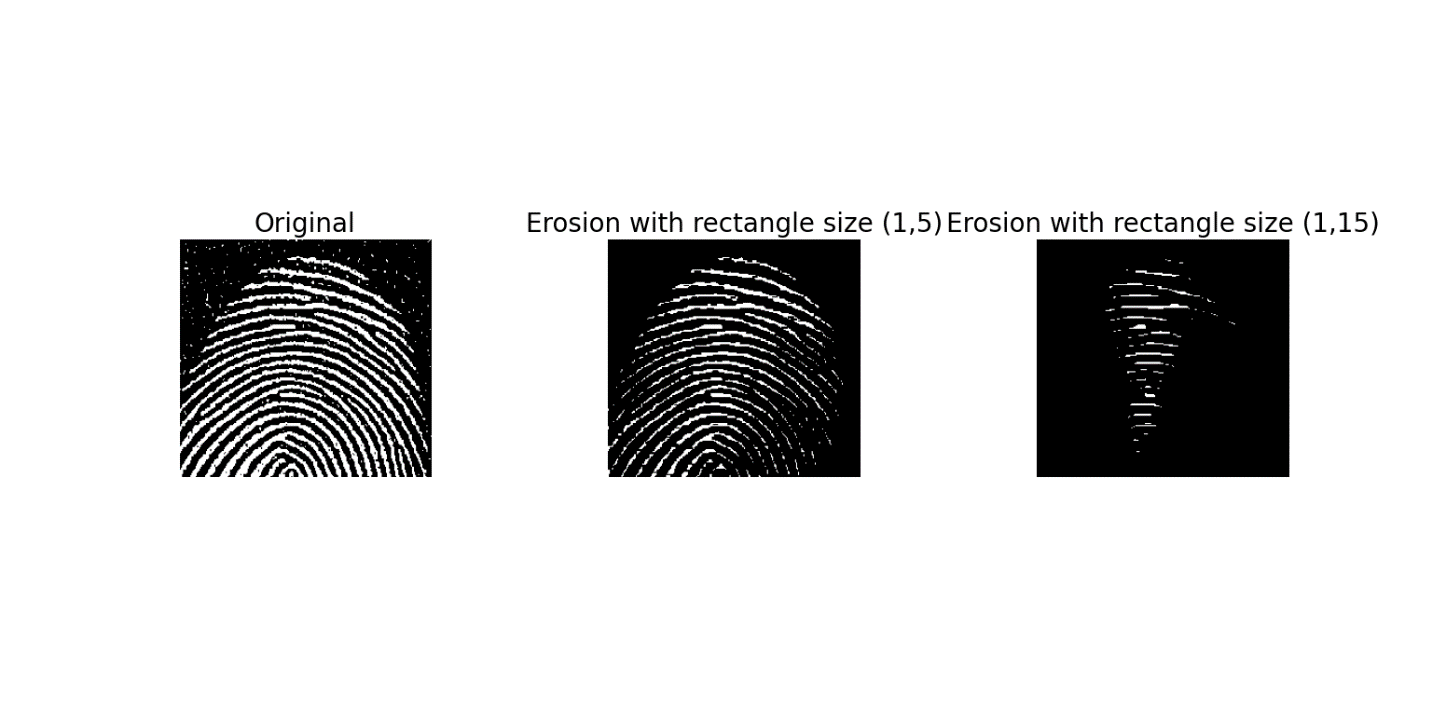
**Áp dụng phép toán co rút (erosion) :**

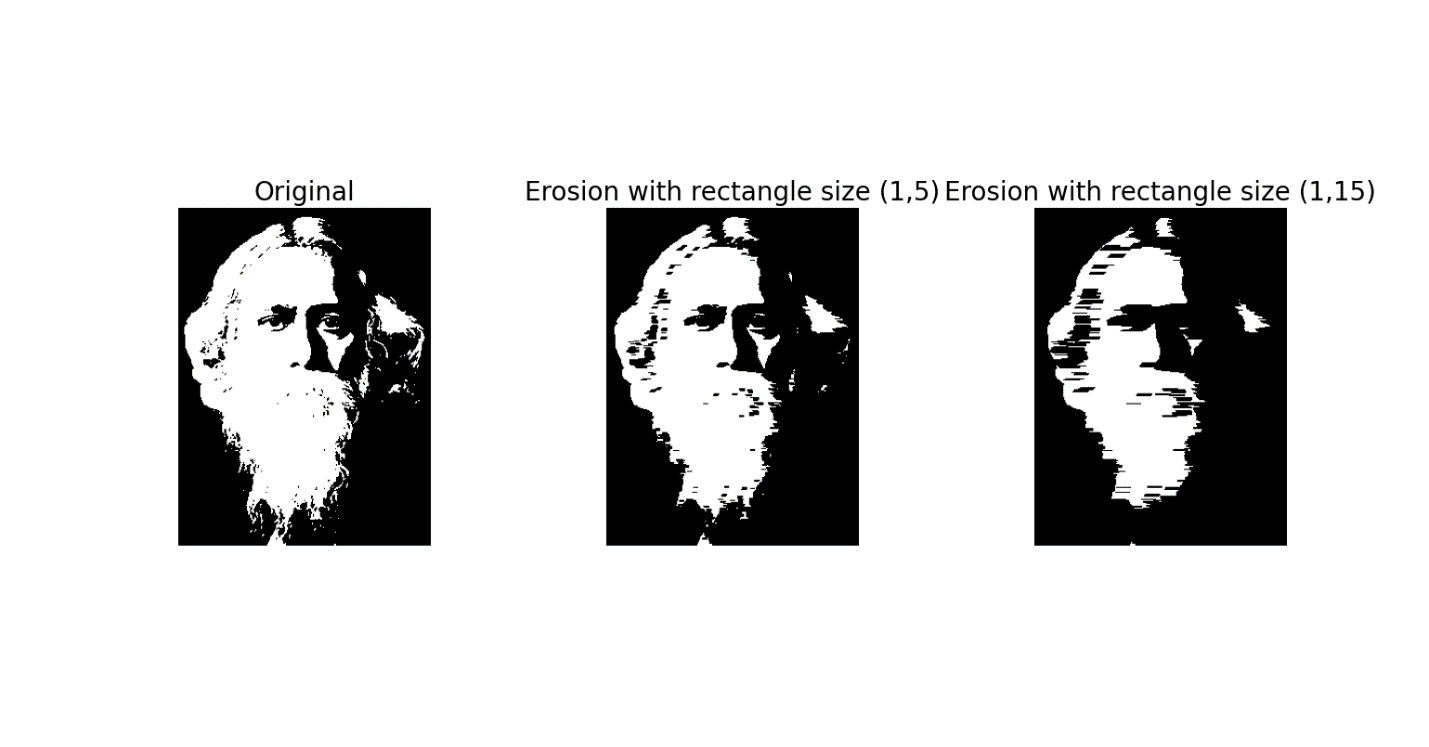
* **binary\_erosion(im, rectangle(1, 5))**:
  + Thực hiện phép co rút trên ảnh nhị phân.
  + Sử dụng phần tử cấu trúc hình chữ nhật kích thước **1x5** (mỏng và dọc).
* **binary\_erosion(im, rectangle(1, 15))**:
  + Tương tự nhưng với phần tử cấu trúc lớn hơn, kích thước **1x15**.
* **Hiển thị kết quả**:
  + **pylab.subplot(1, 3, 2)**: Hiển thị kết quả của phép co rút với hình chữ nhật **1x5**.
  + **pylab.subplot(1, 3, 3)**: Hiển thị kết quả của phép co rút với hình chữ nhật **1x15**.

**4,Kết quả sau khi thực hiện thuật toán :**

Ví dụ 1 : 

Ví dụ 2 :



Ví dụ 3 : 

Nhận xét ảnh sau khi thực hiện :

Sau khi thực hiện thuật toán **erosion** (xói mòn) trên ảnh nhị phân với các kích thước kernel khác nhau, dưới đây là các nhận xét về ảnh sau khi áp dụng:

**1. Ảnh gốc (Original):**

* Ảnh gốc đã được chuyển từ ảnh màu (RGB) sang ảnh xám (grayscale), và sau đó chuyển thành ảnh nhị phân với ngưỡng cố định (0.5).
* Các pixel trong ảnh có giá trị là 0 (đen) hoặc 1 (trắng), tùy thuộc vào độ sáng của pixel gốc.
* Ảnh gốc sẽ có các đối tượng sáng (trắng) trên nền tối (đen), tuy nhiên, ảnh nhị phân có thể thiếu chi tiết nếu có các vùng sáng mờ nhạt.

**2. Erosion với hình chữ nhật (1, 5):**

* **Erosion** làm giảm kích thước của các đối tượng sáng (trắng) trong ảnh.
* Với kernel (1, 5), ảnh sẽ bị xói mòn nhẹ theo chiều ngang, tức là các chi tiết sáng sẽ bị thu hẹp một chút theo chiều ngang, nhưng ảnh vẫn giữ được hình dạng tổng thể.
* Các đối tượng nhỏ sẽ bị loại bỏ hoặc thu nhỏ lại, nhưng ảnh sẽ không thay đổi quá nhiều.

**3. Erosion với hình chữ nhật (1, 15):**

* Với kernel (1, 15), **erosion** sẽ mạnh mẽ hơn vì kernel có chiều rộng lớn hơn.
* Điều này có nghĩa là các đối tượng sáng (trắng) sẽ bị xói mòn mạnh mẽ hơn, đặc biệt là theo chiều ngang. Các chi tiết nhỏ sẽ bị loại bỏ hoàn toàn hoặc trở nên rất mảnh.
* Các đối tượng sáng lớn cũng sẽ bị thu hẹp đáng kể, và hình dạng ban đầu của các đối tượng sẽ bị thay đổi nhiều hơn so với trường hợp kernel nhỏ (1, 5).

THUẬT TOÁN 2 : PHÉP DÃN NỞ ( DILATION )

1, Giới thiệu thuật toán :

Giãn nở là một thao tác hình thái học cơ bản giúp mở rộng kích thước của các đối tượng nền trước, làm mượt biên của các đối tượng và lấp đầy các lỗ hoặc khoảng trống trong ảnh nhị phân. Đây là một thao tác đối ngẫu với xói mòn (erosion).

**Nguyên lý cơ bản**

Giãn nở là một phép toán hình thái trong xử lý ảnh nhị phân, hoạt động bằng cách:

1. **Phát triển kích thước các đối tượng foreground (các pixel có giá trị 1)**.
2. **Làm mượt ranh giới** của các đối tượng.
3. **Lấp đầy các khoảng trống nhỏ hoặc các lỗ** bên trong đối tượng.

Phép toán này sử dụng **phần tử cấu trúc (structuring element)** để quét qua ảnh nhị phân và xác định sự thay đổi giá trị pixel dựa trên quan hệ không gian.

2,Các bước thực hiện thuật toán :

B1: **Chuyển đổi ảnh sang ảnh nhị phân**:

* Sử dụng giá trị ngưỡng để phân chia ảnh thành hai phần: foreground (1) và background (0).
* Trong đoạn mã trên, kênh alpha được dùng làm đầu vào, và các giá trị được chuẩn hóa với ngưỡng 0.5.

 B2: **Khởi tạo phần tử cấu trúc (structuring element)**:

* **Phần tử cấu trúc** là một ma trận nhỏ (ví dụ: hình đĩa, hình vuông, hoặc hình chữ thập) dùng để định nghĩa vùng lân cận trong quá trình giãn nở.
* Trong đoạn mã trên, phần tử dạng **đĩa (disk)** có bán kính được tạo ra với disk(2\*d).

B3: **Thực hiện giãn nở**:

* Với mỗi pixel trong ảnh nhị phân, xét vùng lân cận được xác định bởi phần tử cấu trúc.
* Nếu bất kỳ pixel nào trong vùng lân cận là **foreground (1)**, thì pixel trung tâm cũng được gán giá trị 1.
* Điều này giúp mở rộng các vùng foreground.

B4: **Lặp lại với các kích thước khác nhau**:

* Bằng cách thay đổi kích thước của phần tử cấu trúc, có thể điều chỉnh mức độ giãn nở. Trong đoạn mã, bán kính đĩa được tăng từ 2 đến 4.

B5: **Hiển thị kết quả**:

* So sánh ảnh gốc và ảnh sau khi giãn nở với các bán kính khác nhau để thấy rõ sự khác biệt.

3,Giải thích đoạn mã nguồn :

from skimage.morphology import binary\_dilation, disk  
from skimage import img\_as\_float  
import matplotlib.pylab as pylab  
from skimage.io import imread  
  
# Đọc ảnh  
im = img\_as\_float(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\triangle\_circle.png"))  
  
# Chuyển ảnh về dạng nhị phân  
im = 1 - im[..., 3] # Chọn kênh alpha nếu ảnh có 4 kênh  
im[im <= 0.5] = 0 # Gán giá trị 0 cho các pixel có độ sáng <= 0.5  
im[im > 0.5] = 1 # Gán giá trị 1 cho các pixel có độ sáng > 0.5  
  
# Hiển thị các kết quả  
pylab.gray()  
pylab.figure(figsize=(18, 9))  
  
# Ảnh gốc  
pylab.subplot(131)  
pylab.imshow(im)  
pylab.title('Original', size=20)  
pylab.axis('off')  
  
# Erosion với dilation với các kích thước khác nhau  
for d in range(1, 3):  
 pylab.subplot(1, 3, d + 1)  
 im1 = binary\_dilation(im, disk(2 \* d)) # Tạo dilation với hình đĩa (disk) có bán kính 2\*d  
 pylab.imshow(im1)  
 pylab.title('Dilation with disk size ' + str(2 \* d), size=20)  
 pylab.axis('off')  
  
# Hiển thị kết quả  
pylab.show()

**Import các thư viện cần thiết**

* **binary\_dilation**: Hàm thực hiện giãn nở trên ảnh nhị phân.
* **disk**: Tạo phần tử cấu trúc (structuring element) dạng đĩa với bán kính xác định.
* **img\_as\_float**: Chuyển đổi ảnh sang định dạng float với giá trị nằm trong khoảng [0, 1].
* **pylab**: Công cụ từ matplotlib để hiển thị hình ảnh và biểu đồ.
* **imread**: Dùng để đọc ảnh từ tệp.

**Đọc ảnh và chuyển đổi thành ảnh nhị phân**

**Đọc ảnh:**

* Hàm imread đọc ảnh từ đường dẫn chỉ định.
* Hàm img\_as\_float chuẩn hóa giá trị pixel của ảnh về dạng số thực trong khoảng [0, 1].

**Chuyển đổi thành ảnh nhị phân:**

* **1 - im[..., 3]**: Nếu ảnh có 4 kênh (RGBA), kênh alpha (độ trong suốt) được chọn để làm đầu vào. Sau đó giá trị được đảo ngược (chuyển từ trong suốt sang đục).
* **Phân ngưỡng (thresholding):**
  + Các pixel có giá trị **≤ 0.5** được gán là **nền (0)**.
  + Các pixel có giá trị **> 0.5** được gán là **đối tượng foreground (1)**.
* Kết quả là một ảnh nhị phân chỉ chứa hai giá trị: **0** và **1**.

**Hiển thị ảnh gốc**

* **pylab.gray()**: Cài đặt chế độ hiển thị ảnh theo thang độ xám (grayscale).
* **pylab.figure(figsize=(18, 9))**: Tạo khung hiển thị với kích thước 18x9 inch.
* **pylab.subplot(131)**: Chia khung hiển thị thành 3 cột, hiển thị ảnh gốc ở vị trí đầu tiên (1/3).
* **pylab.imshow(im)**: Hiển thị ảnh nhị phân gốc.
* **pylab.title('Original', size=20)**: Đặt tiêu đề là "Original".
* **pylab.axis('off')**: Tắt hiển thị trục tọa độ.

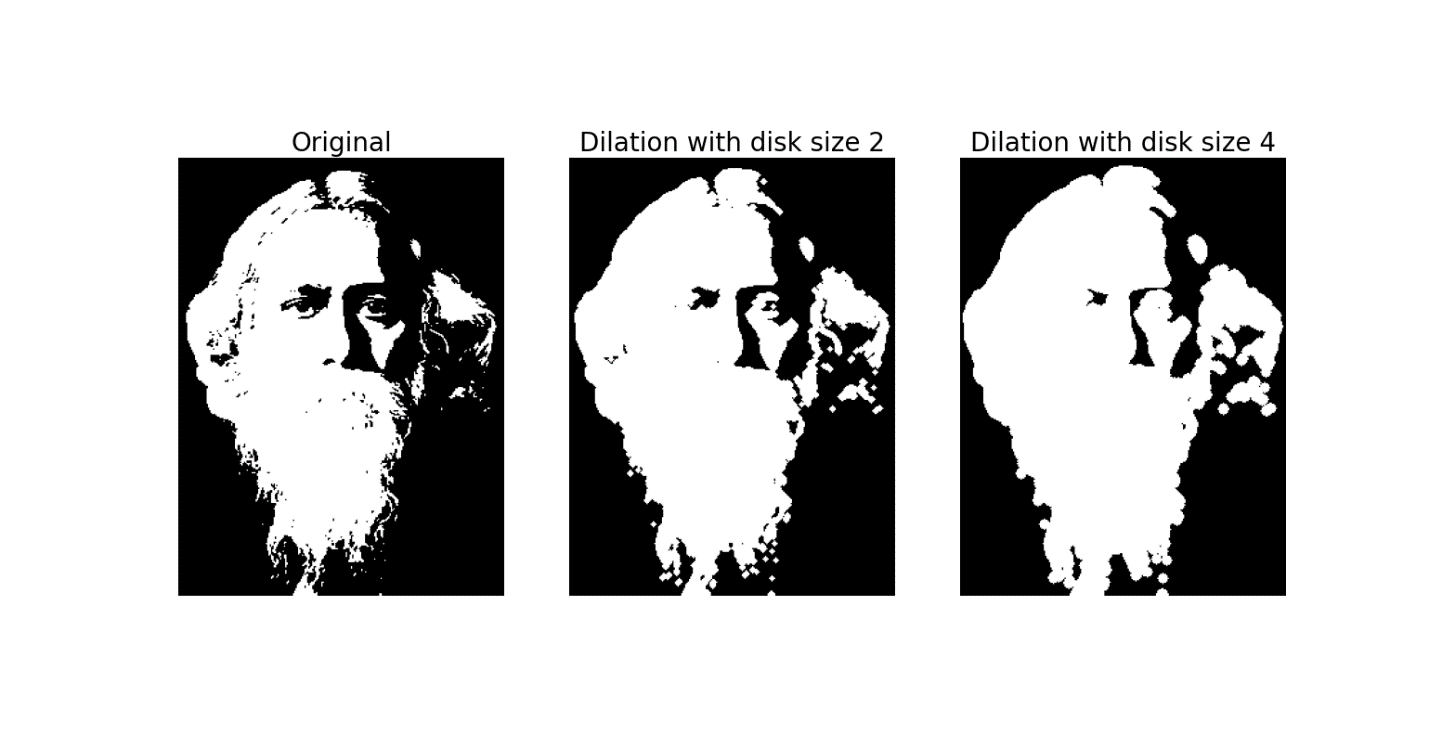
**Thực hiện giãn nở (dilation)**

* **Vòng lặp for**:
  + Thực hiện giãn nở với các phần tử cấu trúc dạng đĩa có bán kính khác nhau: radius=2⋅d\text{radius} = 2 \cdot dradius=2⋅d (tăng từ 2 đến 4).
* **binary\_dilation(im, disk(2 \* d))**:
  + Áp dụng giãn nở trên ảnh nhị phân im.
  + **disk(2 \* d)**: Tạo phần tử cấu trúc dạng đĩa có bán kính 2⋅d2 \cdot d2⋅d.
* **pylab.subplot(1, 3, d + 1)**:
  + Đặt ảnh đã giãn nở vào các cột 2 và 3 của khung hiển thị.
* **pylab.imshow(im1)**: Hiển thị ảnh sau khi giãn nở.
* **pylab.title('Dilation with disk size ' + str(2 \* d), size=20)**: Đặt tiêu đề cho từng ảnh, thể hiện bán kính phần tử cấu trúc được sử dụng.
* **pylab.axis('off')**: Tắt hiển thị trục tọa độ.

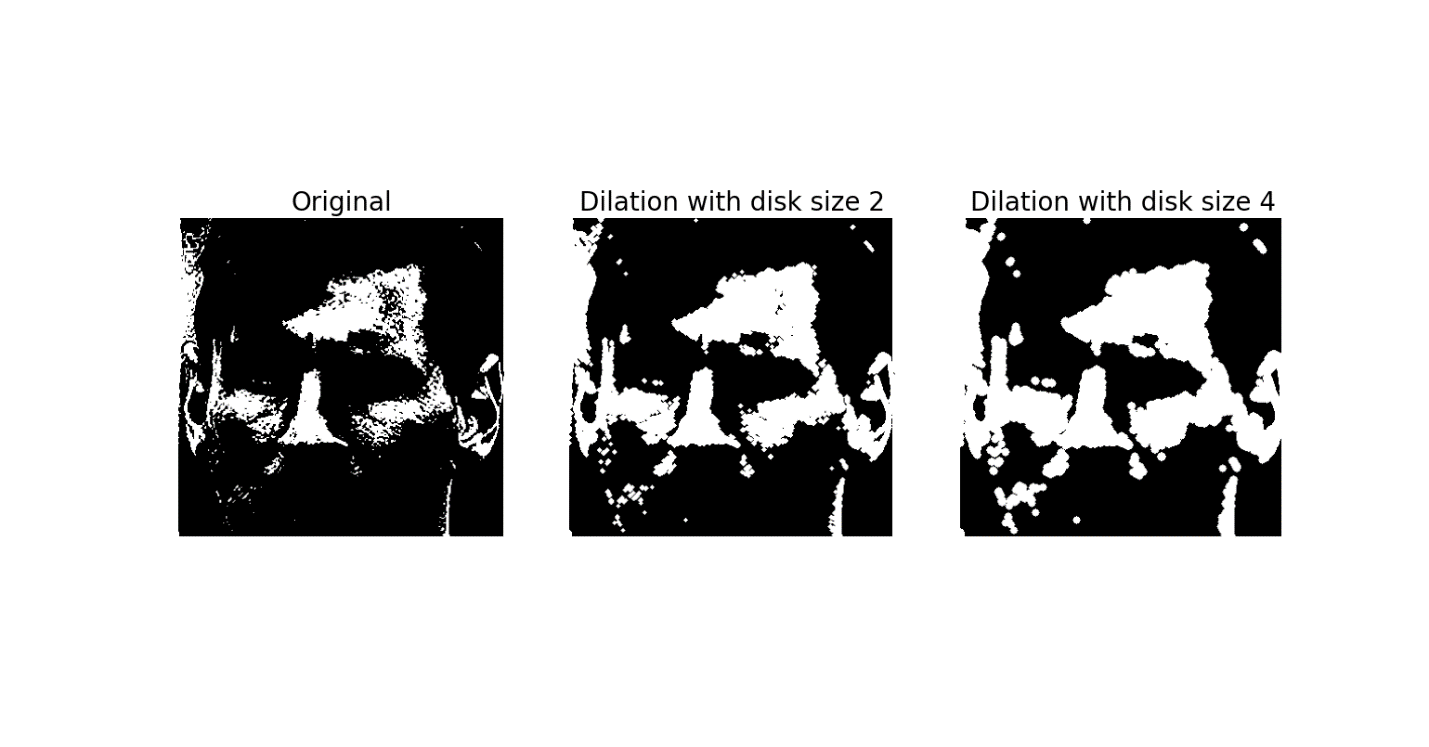
**Hiển thị kết quả**

* **pylab.show()**: Hiển thị tất cả các hình ảnh đã được chuẩn bị trong khung hình.

4, Kết quả sau khi thực hiện :

Ví dụ 1 : 

Ví dụ 2 :



Nhận xét ảnh sau khi thực hiện :

Đoạn mã đã xử lý ảnh nhị phân bằng phép **giãn nở (dilation)** và cho ra ba ảnh để so sánh:

1. **Ảnh gốc**:
   * Đây là ảnh nhị phân được tạo ra sau khi phân ngưỡng (thresholding) từ ảnh ban đầu.
   * Các pixel có giá trị sáng (foreground) và tối (background) đã được phân biệt rõ ràng.
   * Ảnh có thể vẫn chứa các khoảng trống hoặc ranh giới không liên tục.
2. **Ảnh giãn nở với phần tử cấu trúc nhỏ (disk size 2)**:
   * Các vùng sáng (foreground) được mở rộng nhẹ.
   * Một số khoảng trống nhỏ bên trong các vùng foreground đã được lấp đầy.
   * Các ranh giới của đối tượng trở nên mượt hơn.
   * Một số chi tiết nhỏ (nếu có) vẫn còn giữ lại.
3. **Ảnh giãn nở với phần tử cấu trúc lớn (disk size 4)**:
   * Hiệu ứng giãn nở mạnh hơn:
     + Tất cả các khoảng trống nhỏ đã được lấp đầy.
     + Các vùng foreground nhỏ lẻ có thể đã hợp nhất thành các vùng lớn hơn.
   * Tuy nhiên, chi tiết nhỏ trên đối tượng có thể bị mất đi.
   * Ranh giới của các đối tượng trở nên mượt hơn nhưng ít chi tiết.

THUẬT TOÁN 3 : LÀM MỎNG XƯƠNG ( Skeletonizing )

**1 ,Giới thiệu thuật toán**

**Skeletonizing (Làm mỏng xương)** là một kỹ thuật xử lý ảnh thường được sử dụng trong nhận diện hình dạng và phân tích cấu trúc. Mục tiêu chính là giảm toàn bộ các thành phần liên thông (connected components) trong một ảnh nhị phân xuống một đường hoặc bộ xương rộng một pixel, mà vẫn giữ được cấu trúc hình học ban đầu.

**Ứng dụng của Skeletonizing:**

* Nhận diện đường biên.
* Tách các đối tượng trong ảnh.
* Phân tích cấu trúc hình học hoặc mạng lưới (ví dụ: mạng mạch máu).

### **2,Các bước thực hiện thuật toán**

**B1 :Đầu vào:**

* + Một ảnh nhị phân, trong đó giá trị 1 đại diện cho nền trước (foreground) và 0 đại diện cho nền (background).

**B2 :Làm mỏng hình thái học (Morphological Thinning):**

* + Phép toán được áp dụng lặp đi lặp lại để loại bỏ các pixel rìa mà không phá vỡ kết nối của cấu trúc hình học ban đầu.
  + Quá trình dừng lại khi không còn pixel nào có thể bị loại bỏ.

**B3 : Đầu ra:**

* + Một ảnh nhị phân, nơi mỗi đối tượng được biểu diễn dưới dạng bộ xương rộng một pixel.

**3 ,Giải thích đoạn mã nguồn**

import matplotlib.pylab as pylab  
from skimage.morphology import skeletonize  
from skimage import img\_as\_float  
from skimage.io import imread  
  
# Hàm hiển thị hình ảnh  
def plot\_image(image, title):  
 pylab.imshow(image, cmap='gray') # Hiển thị ảnh dạng thang độ xám  
 pylab.title(title)  
 pylab.axis('off') # Ẩn trục  
  
def plot\_images\_horizontally(original, filtered, filter\_name, sz=(18, 7)):  
 pylab.gray() # Thiết lập chế độ hiển thị ảnh xám  
 pylab.figure(figsize=sz) # Kích thước cửa sổ hiển thị  
 pylab.subplot(1, 2, 1) # Chia cửa sổ hiển thị thành 1 hàng, 2 cột  
 plot\_image(original, 'original') # Hiển thị ảnh gốc  
 pylab.subplot(1, 2, 2) # Vị trí hiển thị ảnh thứ 2  
 plot\_image(filtered, filter\_name) # Hiển thị ảnh sau xử lý  
 pylab.show()  
  
# Đọc ảnh và xử lý nhị phân  
im = img\_as\_float(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\horse-dog.jpg", as\_gray=True))  
threshold = 0.5  
im[im <= threshold] = 0  
im[im > threshold] = 1  
  
# Tạo skeleton  
skeleton = skeletonize(im)  
  
# Hiển thị ảnh gốc và skeleton  
plot\_images\_horizontally(im, skeleton, 'skeleton', sz=(18, 9))

**Import thư viện**

* pylab: Thư viện để hiển thị hình ảnh.
* skeletonize: Hàm từ skimage.morphology để làm mỏng xương ảnh nhị phân.
* img\_as\_float: Chuyển đổi ảnh sang dạng số thực với giá trị từ 0 đến 1.
* imread: Đọc ảnh từ tệp.

**Hàm plot\_image**

* Hiển thị một ảnh đơn với tiêu đề.

**Hàm plot\_images\_horizontally**

* Hiển thị hai ảnh cạnh nhau (ảnh gốc và ảnh sau xử lý).

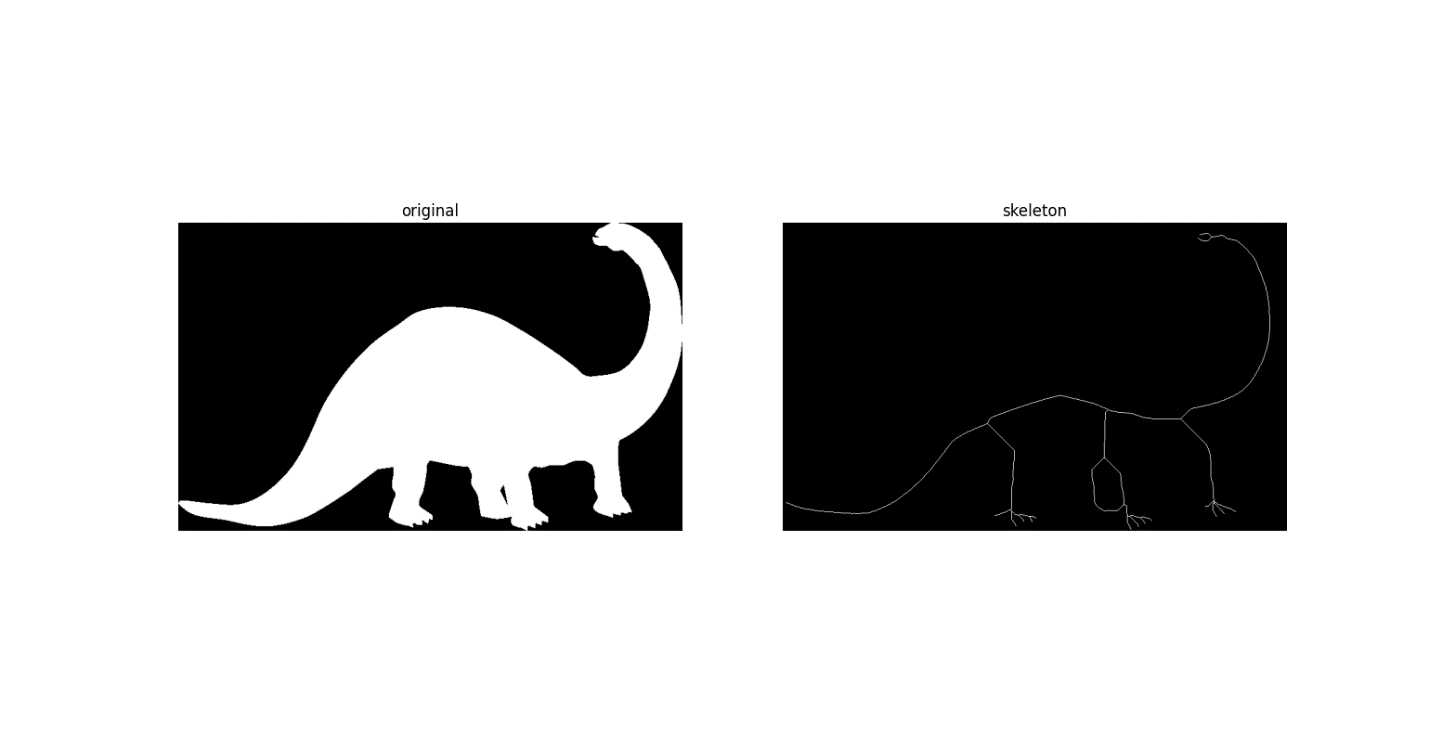
**Xử lý ảnh đầu vào**

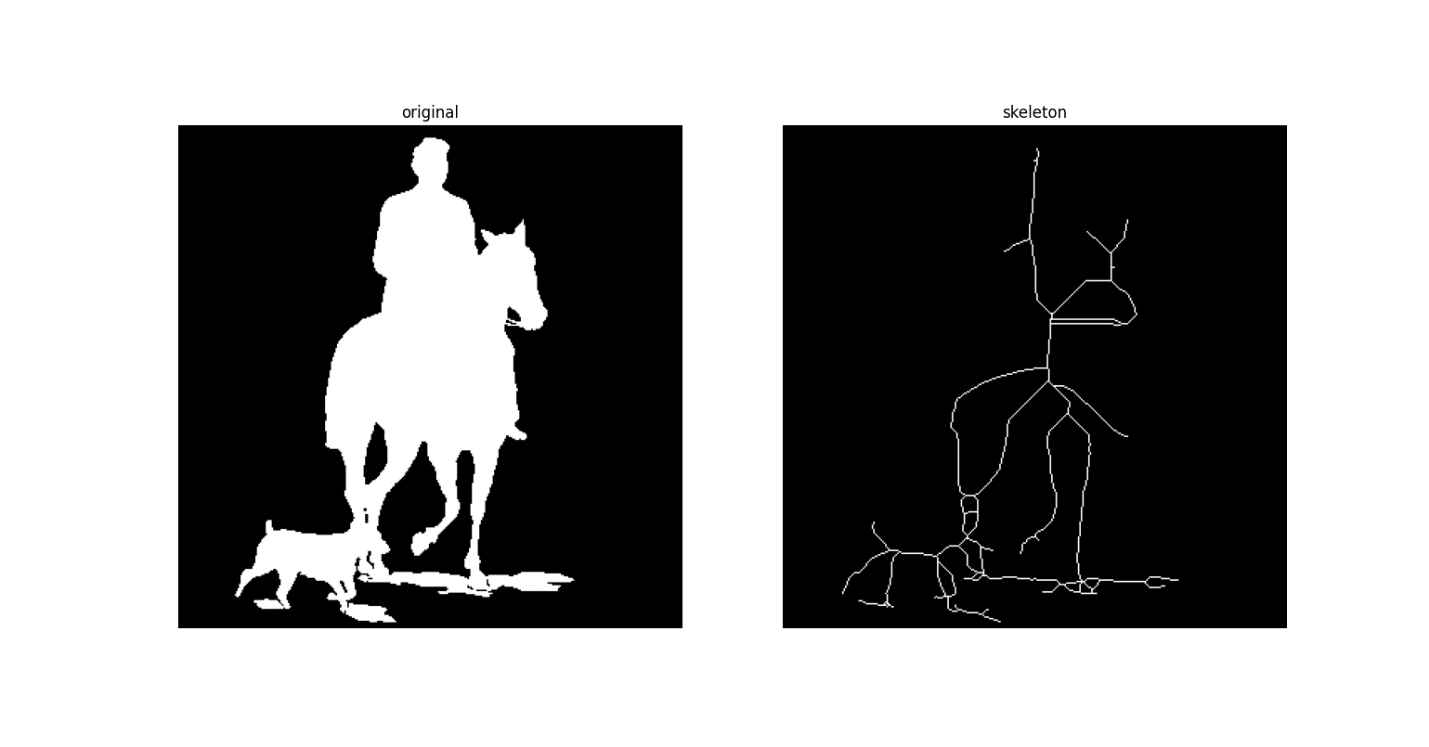
1. imread: Đọc ảnh từ tệp.
2. [..., 3]: Chọn kênh alpha (kênh thứ 4) từ ảnh RGBA.
3. threshold = 0.5: Ngưỡng phân đoạn (threshold) để tạo ảnh nhị phân.
   * Pixel ≤ 0.5 được gán giá trị 0.
   * Pixel > 0.5 được gán giá trị 1.

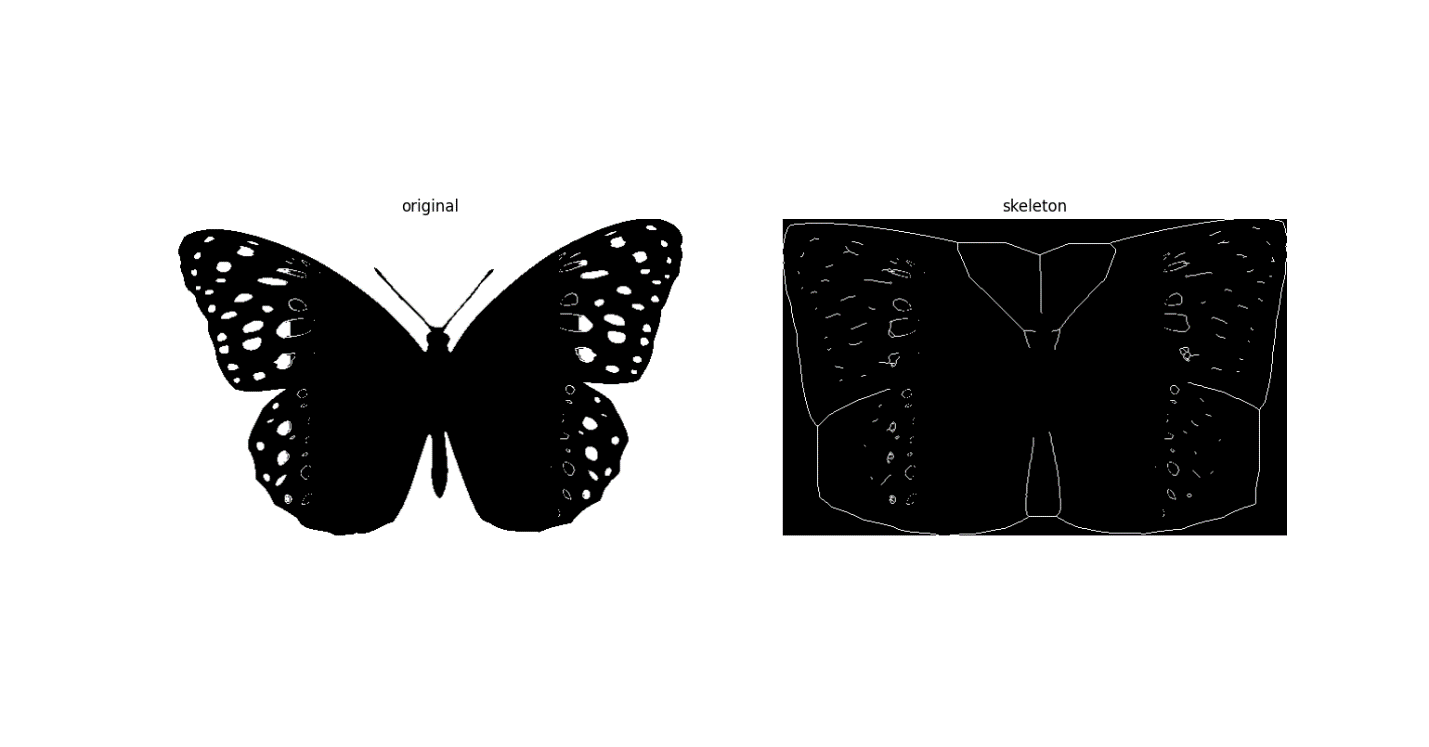
**Làm mỏng xương**

* Áp dụng thuật toán skeletonizing để tạo bộ xương ảnh.

**4,Kết quả sau khi thực hiện thuật toán :**

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 : 

Nhận xét ảnh đầu ra :

* **Ảnh gốc (original)**:  
  Là ảnh nhị phân được xử lý từ ảnh đầu vào. Các vùng nền trước có giá trị 1 (trắng), và nền có giá trị 0 (đen).
* **Ảnh bộ xương (skeleton)**:  
  Các đối tượng trong ảnh đã được giảm xuống một bộ xương rộng một pixel, giữ nguyên hình dạng tổng thể nhưng loại bỏ các pixel thừa ở biên.

**Ưu điểm:**

* Giảm kích thước dữ liệu trong ảnh mà vẫn bảo toàn cấu trúc hình học chính.
* Phù hợp để phân tích cấu trúc và nhận diện hình dạng.

**Nhược điểm:**

* Kết quả có thể bị nhiễu nếu ảnh gốc không được làm mịn hoặc có nhiều nhiễu nền.