

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN



Bài tập Thực hành 1:
CÁC TOÁN TỬ HÌNH THÁI HỌC

Môn: Thị Giác Máy Tính Nâng Cao

Thực hiện: 21127240 – Nguyễn Phát Đạt

Lớp: HP2-K33

Giảng viên hướng dẫn: Lý Quốc Ngọc

Nguyễn Mạnh Hùng

Thành phố Hồ Chí Minh, tháng 7 năm 2024

Mục lục

1	Đánh giá.....	3
1.1	Thực hiện:.....	3
1.2	Kết quả:	3
2	Mô tả	3
2.1	Toán tử hình thái học nhị phân	5
2.1.1	Erosion	5
2.1.2	Dilation	6
2.1.3	Opening	8
2.1.4	Closing	9
2.1.5	Hit-or-miss.....	9
2.1.6	Boundary extraction.....	10
2.1.7	Hole filling.....	12
2.1.8	Extraction of Connected Components	13
2.1.9	Convex Hull.....	15
2.1.10	Thinning.....	16
2.1.11	Thickening.....	18
2.1.12	Skeletons	21
2.1.13	Pruning	26
2.1.14	Morphological Reconstruction	30
2.2	Toán tử hình thái học với ảnh độ xám	32
2.2.1	Erosion	32
2.2.2	Dilation	32

2.2.3	Opening	33
2.2.4	Closing	35
2.2.5	Morphological smoothing	36
2.2.6	Morphological gradient	36
2.2.7	Top-hat and bottom-hat transformations	37
2.2.8	Granulometry	38
2.2.9	Textural segmentation	38
2.2.10	Morphological Reconstruction	38
3	Kết luận	39
4	Tham khảo.....	39

1 Đánh giá

1.1 Thực hiện:

Mã số học viên	Họ tên	Lớp
21127240	Nguyễn Phát Đạt	HP2-K33

1.2 Kết quả:

Công việc	Hoàn thành
Cài đặt các toán tử hình thái học nhị phân: erosion, dilation, opening, closing, reconstruction...	100%
Cài đặt các toán tử hình thái học với ảnh độ xám: erosion, dilation, opening, closing, reconstruction...	80% (đã hoàn thành phần cài đặt một số phần không kịp viết báo cáo)
Bổ sung: Gồm một số phần như: <ul style="list-style-type: none">Thuật toán thinning Zhang-SuenBorder clearingLấp tất cả lỗ trong hình mà không cần biết điểm bắt đầuMorphological smoothingTop-hat by reconstruction	100%

2 Cách sử dụng chương trình

Lưu ý: do ở 2 cách xử lý ảnh nhị phân và ảnh xám có cách đọc dữ liệu và đầu ra khác nhau nên để thuận tiện sử dụng, chương trình sẽ được chia thành 2 phần riêng biệt:

- Xử lý ảnh nhị phân gồm:
 - ‘source/binary_main.py’: chứa chương trình thực thi xử lý ảnh nhị phân. Khi thực thi sẽ gọi các hàm thao tác toán tử hình thái được định nghĩa trong mô-đun: ‘morphological_operators/binary.py’
 - Thực thi:
`“python source/binary_main.py -i <input_path> -o <output_path> -p <operator> -t <wait_key_time>”`
 Trong đó:
 - input_path: Ảnh đầu vào (có thể là ảnh màu hoặc ảnh xám).
 - output_path: Ảnh đầu ra (là ảnh nhị phân nhưng ở định dạng ảnh xám với mỗi điểm ảnh mang giá trị 0 là đen hoặc 255 là trắng giúp đa số các trình đọc ảnh bình thường đều thể hiện rõ ràng được).
 - operator: Phép toán tử hình thái học được chọn để thực hiện.
 - wait_key_time: Thời gian đợi khi ảnh được thể hiện bởi hàm cv2.imshow(), mặc định mang giá trị 0 nghĩa là đợi đến khi người dùng nhấn phím bất kỳ mới tiếp tục.
- Xử lý ảnh độ xám:
 - ‘source/grayscale_main.py’: chứa chương trình thực thi xử lý ảnh nhị phân. Khi thực thi sẽ gọi các hàm thao tác toán tử hình thái được định nghĩa trong mô-đun: ‘morphological_operators/grayscale.py’.
 - Thực thi:
`“python source/binary_main.py -i <input_path> -o <output_path> -p <operator> -t <wait_key_time>”`
 - Các tham số có ý nghĩa tương tự đối với ảnh nhị phân ở trên, ngoại trừ ảnh đầu ra là ảnh xám (các điểm ảnh mang giá trị từ 0-255).

Vì lý do đã đề cập ở trên thì các hàm xử lý ảnh nhị phân sẽ thao tác luôn trên các số 0/255 thay vì 0/1 giúp hạn chế thời gian chuyển đổi sang ảnh đầu ra.

Ngoài ra, đối với các thuật toán cần nhiều vòng lặp thì để dễ dàng xem quá trình thực hiện của thuật toán thì chương trình cũng sẽ in ra các ảnh trong quá trình tính toán giúp ta theo dõi dễ hơn (các ảnh có tiêu đề là ‘in progress’).

3 Mô tả toán tử

3.1 Toán tử hình thái học nhị phân

3.1.1 Erosion

Toán tử erosion (xói mòn) là một phép toán hình thái học cơ bản trong xử lý ảnh, được sử dụng để làm mờ các vùng sáng trên ảnh. Nó có tác dụng loại bỏ các điểm ảnh biên và làm giảm kích thước của các vùng sáng.

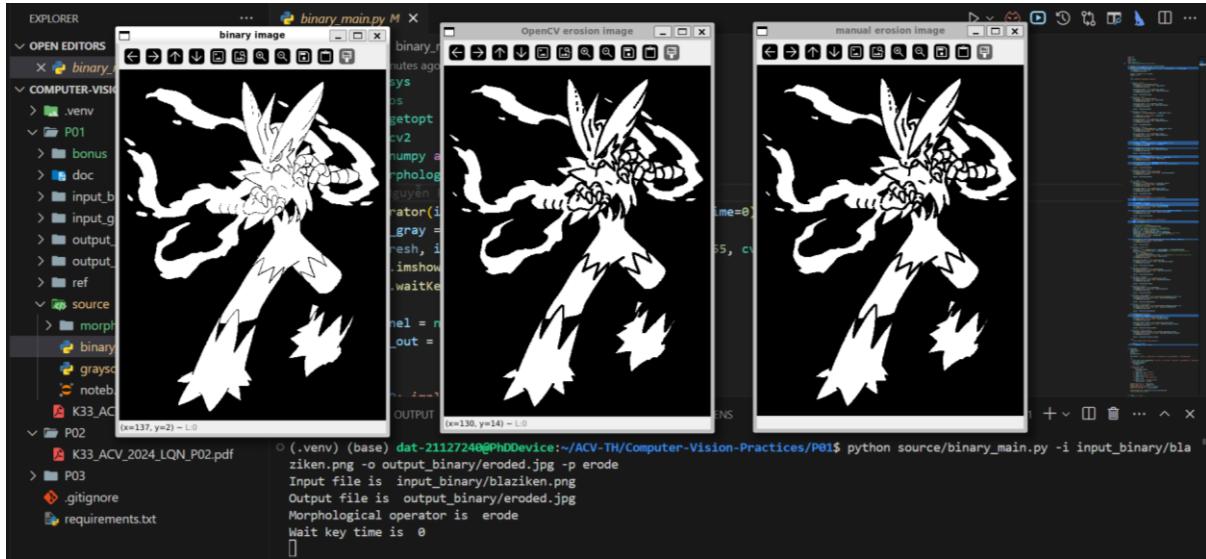
- **Giải thuật:**

- Đầu vào:
 - `img`: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
 - `kernel`: Phần tử cấu trúc (structuring element/kernel) với các giá trị 0/1 và sẽ mặc định tâm nằm ở giữa, ta sẽ gọi tắt là PTCT.
 - `n`: Số lần thực hiện phép toán.
- Thực thi:
 - Đệm (padding) ảnh thêm với phần nửa kích thước dài/rộng của phần tử cấu trúc (giúp đảm bảo có thể thực hiện được trên biên).
 - Khởi tạo ảnh đầu ra là kích thước bằng với `img` và mang giá trị 0.
 - Với mỗi điểm ảnh (i,j) trên ảnh gốc:
 - Xác định vùng ảnh cần xét ứng với PTCT, chỉ xét các vùng mang giá trị 1.
 - Nếu **tất cả** điểm ảnh đang xét trên ảnh gốc đều mang giá trị 255 thì gán giá trị 255 vào vị trí (i,j) trên ảnh đầu ra.

- Lặp lại thao tác trên bằng số lần mong muốn n.
- Trả về kết quả.

- **Kết quả:**

Lưu ý: Nếu không đề cập thì mặc định, chương trình sẽ sử dụng PTCT là hình vuông kích thước 3x3 mang toàn bộ giá trị 1 (các phép toán khác cũng tương tự).



Ảnh 1: Kết quả erosion

⇒ Các đường viền (khe hở) được mở rộng.

3.1.2 Dilation

Toán tử dilation (giãn nở) là một phép giúp mở rộng các vùng sáng (giá trị 255) trong ảnh nhị phân hoặc ảnh xám, làm cho các đối tượng trong ảnh trở nên to hơn và kết nối các vùng sáng gần nhau.

- **Giải thuật:**

- Đầu vào:
 - img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
 - kernel: Phần tử cấu trúc (structuring element/kernel) với các giá trị 0/1.

- n: Số lần thực hiện phép toán.
 - Thực thi:
 - Đệm (padding) ảnh thêm với phân nửa kích thước dài/rộng của phần tử cấu trúc (giúp đảm bảo có thể thực hiện được trên biên).
 - Khởi tạo ảnh đầu ra là kích thước bằng với img và mang giá trị 0.
 - Với mỗi điểm ảnh (i,j) trên ảnh gốc:
 - Xác định vùng ảnh cần xét ứng với PTCT, chỉ xét các vùng mang giá trị 1.
 - Nếu **bất kì** điểm ảnh đang xét trên ảnh gốc có mang giá trị 255 thì gán giá trị 255 vào vị trí (i,j) trên ảnh đầu ra.
 - Lặp lại thao tác trên bằng số lần mong muốn n.
 - Trả về kết quả.
- **Kết quả:**
-
- ```
(.venv) (base) dat-2112724@PhDDevice:~/ACV-TH/Computer-Vision-Practices/P01$ python source/binary_main.py -i input_binary/blaziken.png -o output_binary/dilated.jpg -p dilate
Input file is input_binary/blaziken.png
Output file is output_binary/dilated.jpg
Morphological operator is dilate
Wait key time is 0
```

Ảnh 2: Kết quả dilation

⇒ Các khe hở thu nhỏ lại (thậm chí biến mất), các điểm sáng nở ra rộng hơn.

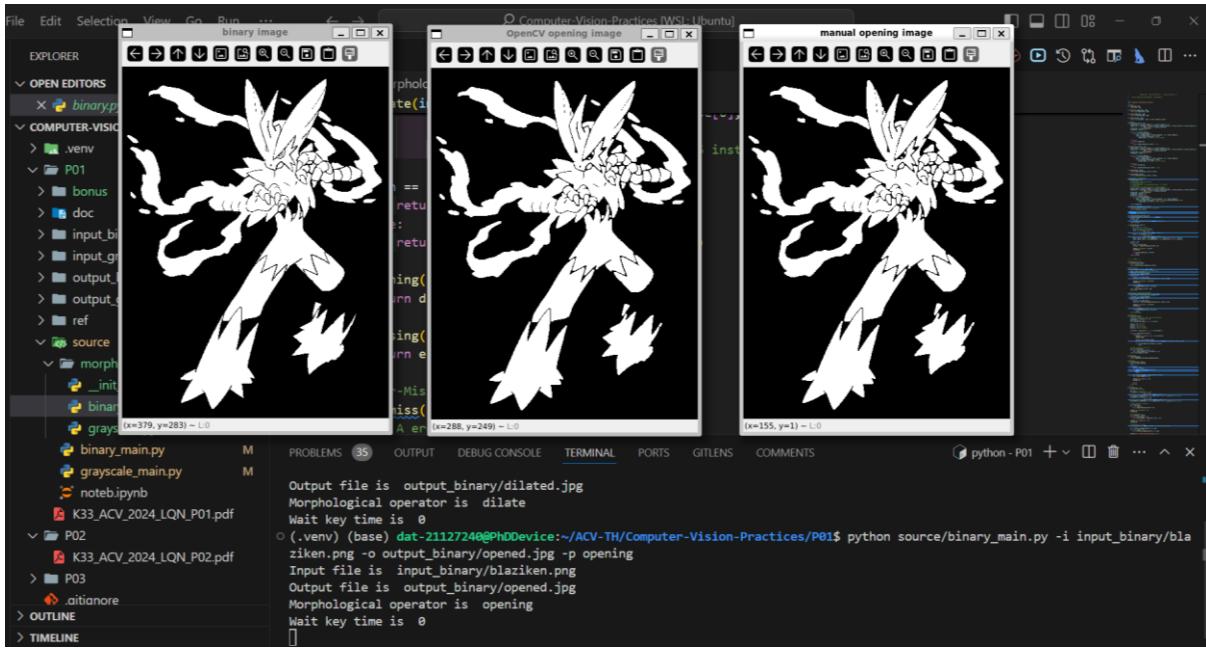
### 3.1.3 Opening

Toán tử **opening** (mở) là kết hợp hai phép toán cơ bản là **erosion** và **dilation**. Thường được sử dụng để loại bỏ nhiễu nhỏ và các đối tượng không mong muốn trong ảnh nhị phân, đồng thời giữ nguyên hình dạng và kích thước của các đối tượng lớn hơn. Nó có tác dụng làm mịn các đường viền của các đối tượng, tách các đối tượng nối liền nhau và loại bỏ các chi tiết nhỏ không liên quan.

- **Giải thuật:**

- Đầu vào:
  - **img:** Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
  - **kernel:** Phần tử cấu trúc (structuring element/kernel) với các giá trị 0/1.
  - **n:** Số lần thực hiện phép toán. (do đối với opening và closing thì chỉ có tác dụng 1 lần nếu cùng sử dụng 1 PTCT).
- Thực thi: Dilate kết quả erosion của ảnh img với PTCT là kernel.

- **Kết quả:**



Ảnh 3: Kết quả opening

⇒ Đa số các viền ngoài được giữ nguyên ngoại trừ các lỗ và khe ở trong được mở rộng hơn (mắt, mũi, các chi tiết).

### 3.1.4 Closing

Toán tử **closing** (mở) là cũng kết hợp hai phép toán cơ bản là **erosion** và **dilation** nhưng *theo thứ tự ngược lại*. Ngược lại với opening, closing thường được sử dụng để lấp đầy các lỗ nhỏ và các khoảng trống trong các đối tượng trong ảnh nhị phân, đồng thời giữ nguyên hình dạng và kích thước của các đối tượng lớn hơn. Nó có tác dụng làm mịn các đường viền của các đối tượng, kết nối các đối tượng gần nhau và lấp đầy các khe hở nhỏ trong các đối tượng.

- **Giải thuật:**

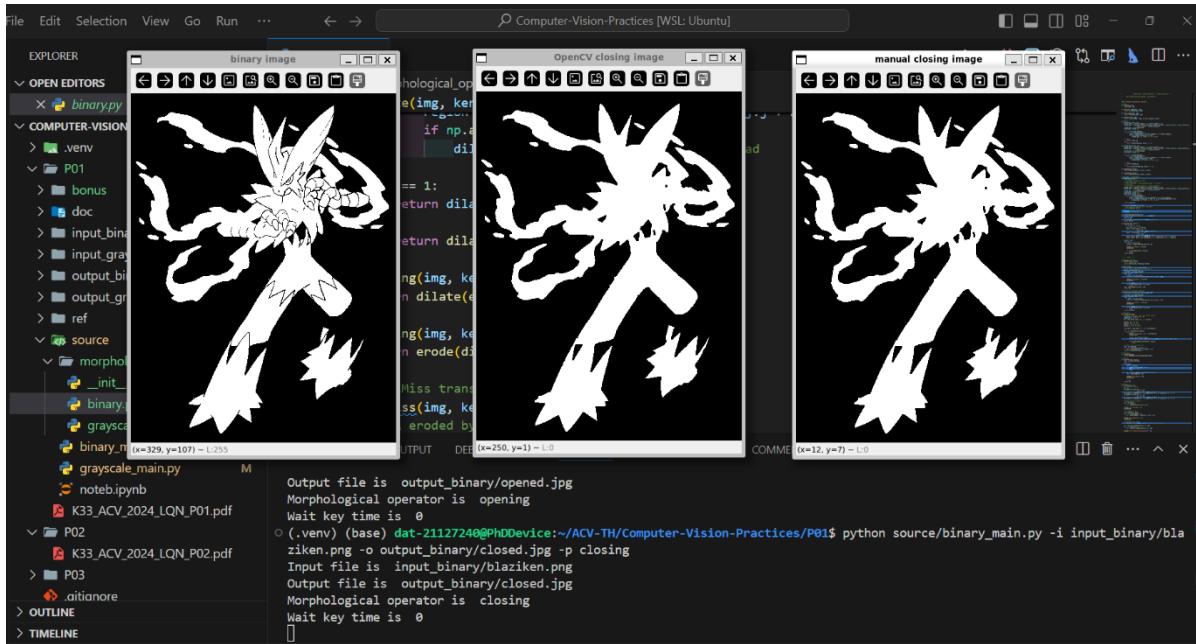
- Đầu vào:

- **img:** Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
- **kernel:** Phần tử cấu trúc (structuring element/kernel) với các giá trị 0/1.

- **n:** Số lần thực hiện phép toán.(do đối với opening và closing thì chỉ có tác dụng 1 lần nếu cùng sử dụng 1 PTCT).

- Thực thi: Erode kết quả dilation của ảnh img với PTCT là kernel.

- **Kết quả:**



Ảnh 4: Kết quả closing

- ⇒ Các viền ngoài cũng được giữ nguyên ngoại trừ các lỗ và khe hở bị lấp đi, khá tương tự với dilate nhưng hình dạng ảnh giống với ban đầu hơn chứ không nở ra như dilate.

### 3.1.5 Hit-or-miss

Phép toán **Hit-or-Miss** là một phép toán hình thái học đặc biệt trong xử lý ảnh, được sử dụng để tìm các cấu trúc hay mẫu cụ thể trong ảnh. Phép toán này kết hợp hai lần xói mòn (**erosion**) với hai phần tử cấu trúc (structuring elements) khác nhau để xác định các vị trí trong ảnh mà các cấu trúc hay mẫu đó chính xác xuất hiện.

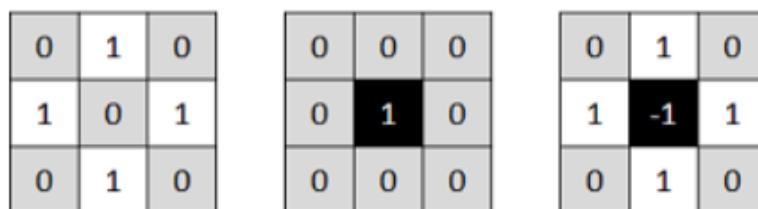
- **Giải thuật:**

- Đầu vào:

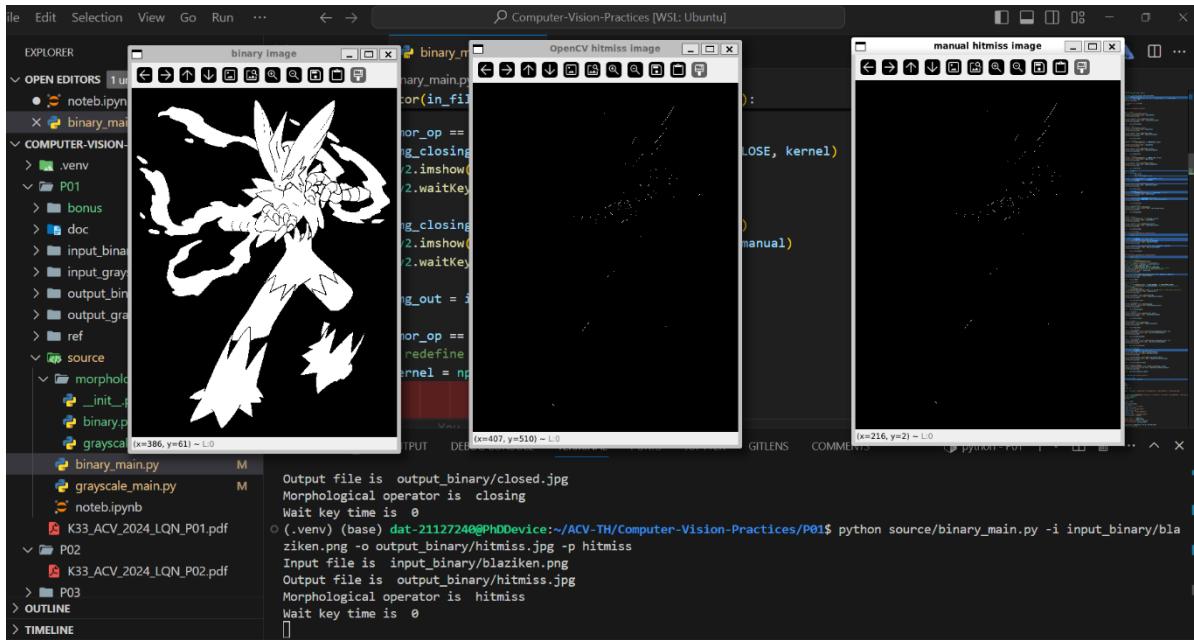
- **img:** Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
- **kernel:** Đặc biệt, phần tử cấu trúc lúc này mang giá trị -1/0/1 với:
  - -1: background
  - 1: foreground
  - 0: Không quan tâm
- Thực thi (cách 1) nếu truyền vào 2 kernel là  $B_1$  và  $B_2$ :
  - Erode ảnh với kernel  $B_1$  (các điểm xét là 1, -1 sẽ xem là 0)
  - Erode ảnh nghịch đảo ( $\text{img}_c$ ) với kernel  $B_2$  (các điểm xét là -1, 1 sẽ xem là 0).
  - Giao 2 kết quả trên lại ta được kết quả.
- Thực thi (cách 2) nếu truyền vào  $B$  là kết hợp của  $B_1$  và  $B_2$ :
  - Thực hiện tương tự như erosion nhưng 1 điểm  $(i,j)$  chỉ thỏa khi xét thêm điều kiện là tất cả các vị trí mang giá trị -1 trên kernel đều phải là 0 trên ảnh gốc.

- **Kết quả:**

Thay vì dùng kernel toàn số 1 như mặc định thì sẽ ra kết quả giống với erosion đơn thuần do không có background, ta dùng kernel sau (lần lượt là  $B_1$ ,  $B_2$  và  $B$ ):



Ảnh 5: PTCT ví dụ của hit-or-miss (nguồn ảnh: [docs.opencv.org](https://docs.opencv.org))



Ảnh 6: Kết quả hitmiss

⇒ Với kernel trên thì ta chỉ lấy các điểm ảnh mà tâm là đen mà các vùng ở 4 hướng (trái/phải/trên/dưới) là trắng. Lưu ý rằng toán tử hit-or-miss thường được dùng chủ yếu trong các toán tử khác để có nghĩa và công dụng cụ thể hơn.

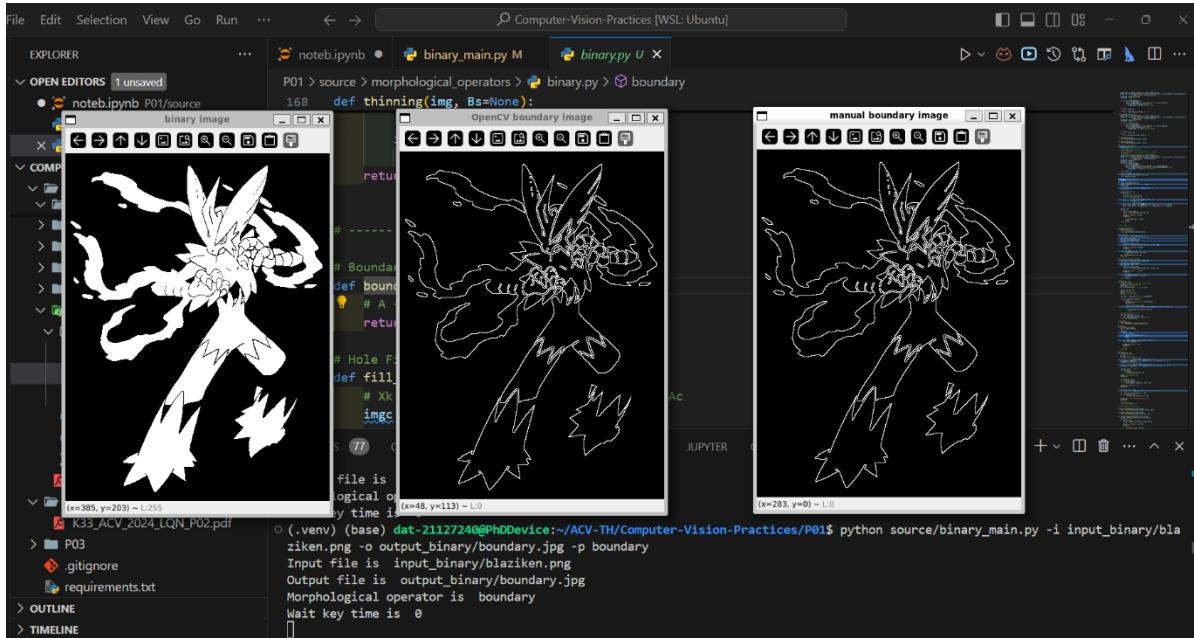
### 3.1.6 Boundary extraction

**Boundary extraction** là một thuật toán giúp lấy ra vùng biên của các đối tượng trên ảnh sử dụng toán tử erosion và phép trừ ma trận.

- **Giải thuật:**

- Đầu vào:
  - img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
  - kernel: PTCT mặc định là 0/1.
- Thực thi:
  - Trừ ảnh gốc với kết quả erosion của ảnh với PTCT là kernel.
  - Lưu ý lúc trừ ảnh ta giới hạn chặn trên dưới của kết quả không nằm ngoài 0 và 255.

- **Kết quả:**



Ảnh 7: Kết quả boundary extraction

⇒ Ta thu được viền của các đối tượng trong ảnh.

### 3.1.7 Hole filling

**Hole filling** giúp thu lấp đầy 1 lỗ bất kì trên ảnh với điểm bắt đầu được truyền vào. Lưu ý: thuật toán lấp đầy tất cả các lỗ trên ảnh tự động, sẽ được đề cập ở phần dưới.

- **Giải thuật:**

- Đầu vào:

- img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
    - kernel: PTCT mặc định là 0/1.
    - xpos, ypos: tọa độ điểm bắt đầu nằm trong lỗ cần lấp (nếu chọn nền thì sẽ lấp cả nền ảnh).

- Thực thi:

- Khởi tạo  $X_0$  là ma trận 0 bằng kích thước với ảnh đầu vào nhưng tại vị trí (xpos, ypos) mang giá trị là 1.

- Gán  $X_k$  là giao của *dilation* của  $X_{k-1}$  với PTCT là kernel với ảnh nghịch ( $\text{img}_c$ ), thực hiện cho đến khi  $X_k$  bằng  $X_{k-1}$  thì dừng.
- Thực hiện hợp  $X_k$  (cuối cùng) với ảnh gốc  $\text{img}$  ta được ảnh có lỗ xác định đã được lấp đầy.

- **Kết quả:**

Đối với các thuật toán yêu cầu tham số đầu vào bổ sung thì chương trình sẽ yêu cầu người dùng nhập vào chương trình trong console:

Chẳng hạn bên dưới, ta muốn che mắt của nhân vật (tọa độ có thể lấy bằng cách trỏ chuột vào ảnh gốc sau khi ảnh hiện ra), ở đây ta ví dụ ta chọn 1 điểm là (243,132) thì kết quả như sau:



*Ảnh 8: Kết quả hole filling*

⇒ Thuật toán phụ thuộc vào PTCT đầu vào, đối với kernel 3x3 chứa toàn số 1 thì thuật toán sẽ hoạt động theo cơ chế 8-way (các điểm được xem là kề nhau thuộc cùng 1 vùng lỗ nếu nó ở xung quanh điểm đó tính cả 4 góc), nếu muốn

dùng 4-way (không tính 4 góc ảnh) thì ta có thể thay đổi kernel thành dạng chữ thập (chứa 0 ở 4 góc).

Lưu ý: do không có hàm định nghĩa sẵn trong opencv, ảnh so sánh là thuật toán lấp đầy tất cả các lỗ tự động, ta vẫn có thể so sánh kết quả đối với lỗ mà ta muốn lấp.

### 3.1.8 Extraction of Connected Components

**Extraction of Connected Components** giúp ta thu được ảnh con chỉ chứa 1 thành phần liên thông mà ta chọn từ ảnh gốc từ 1 vị trí bắt đầu.

- **Giải thuật:**

- Đầu vào:

- img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
    - kernel: PTCT mặc định là 0/1.
    - xpos, ypos: tọa độ điểm bắt đầu nằm trong lỗ cần lấp (nếu chọn nền thì sẽ lấp cả nền ảnh).

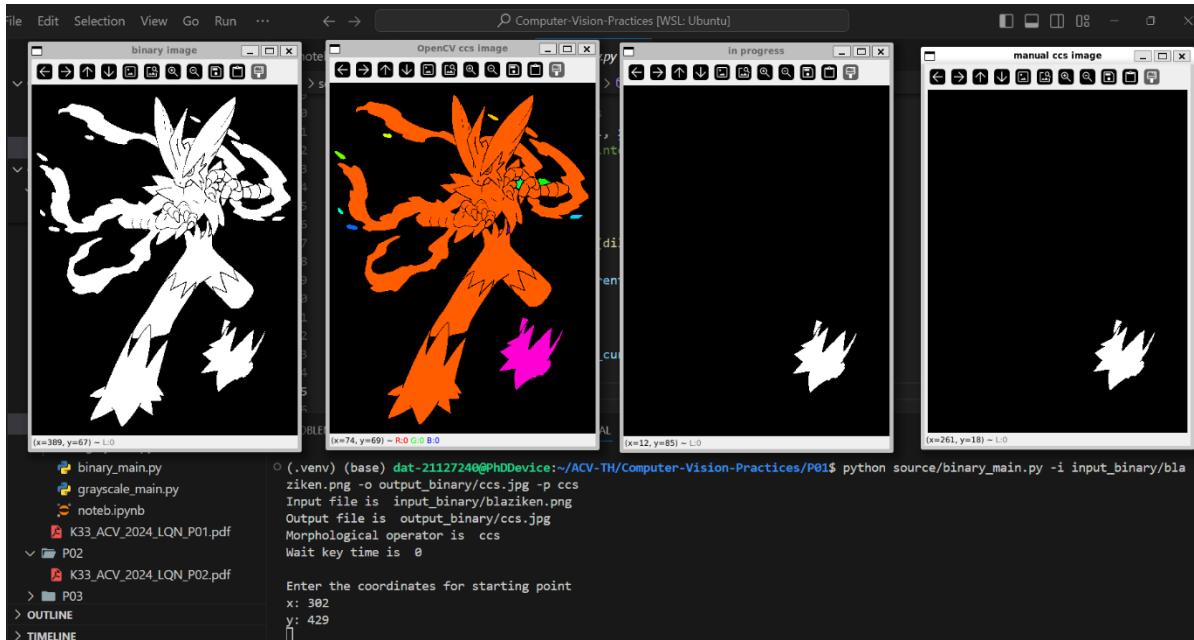
- Thực thi:

- Khởi tạo  $X_0$  là ma trận 0 bằng kích thước với ảnh đầu vào nhưng tại vị trí (xpos, ypos) mang giá trị là 1.
    - Gán  $X_k$  là giao của *dilation* của  $X_{k-1}$  với PTCT là kernel với ảnh gốc (img), thực hiện cho đến khi  $X_k$  bằng  $X_{k-1}$  thì dừng.
    - Trả về  $X_k$  (cuối cùng) ta thu được thành phần liên thông.

- **Kết quả:**

Tương tự như trên, chương trình cũng yêu cầu nhập điểm bắt đầu.

Chẳng hạn bên dưới, ta muốn lấy chân của nhân vật thì thu được kết quả như sau:



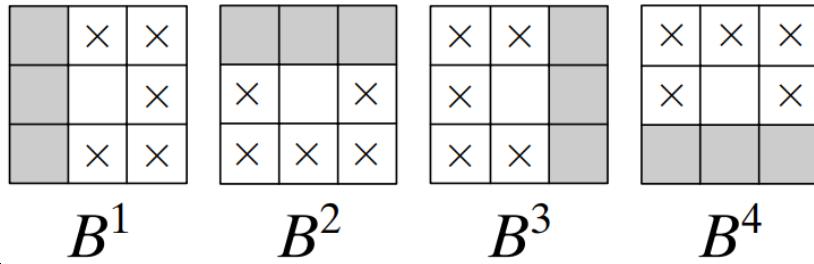
Ảnh 9: Kết quả connected component

⇒ Tương tự như hole filling, thành phần liên thông cho ra cũng phụ thuộc vào PTCT đầu vào, và kết quả của opencv là trích xuất tất cả các thành phần liên thông nên ở đây được tô màu giúp phân biệt nhóm các thành phần liên thông khác nhau.

### 3.1.9 Convex Hull

**Convex Hull** giúp ta thu được ảnh dùng để mô tả cụm điểm ảnh trong không gian hai chiều sao cho đường biên của nó không có độ uốn cong (convex).

- Giải thuật:
  - Đầu vào:
    - img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
    - kernel: PTCT. (không dùng kernel đầu vào do thuật toán yêu cầu 4 kernel khác để hoạt động), gồm:



Ảnh 10: Các PTCT đầu vào mặc định cho convex hull (nguồn ảnh: Digital Image Processing)

- Thực thi:

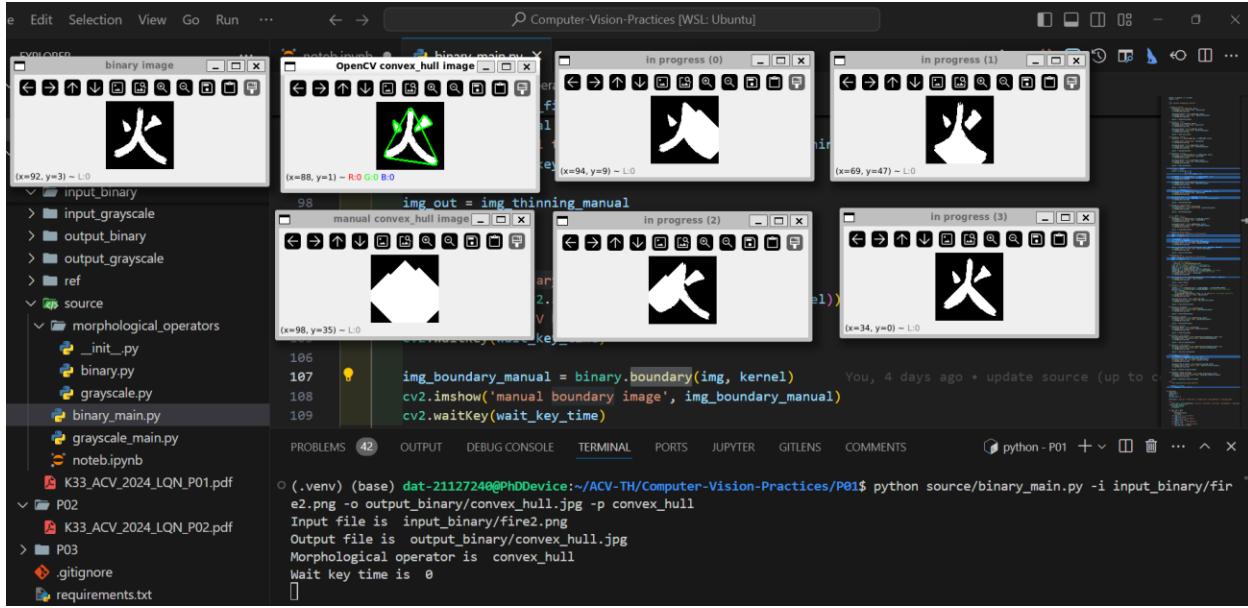
- Khởi tạo các PTCT ( $B_1, B_2, B_3, B_4$ ) như trên.
- Với  $X_0$  là ảnh đầu vào  $A$ , ta thực hiện hit-or-miss với kernel  $B_i$  rồi hợp với ảnh  $A$ , lặp lại cho đến khi không thay đổi ( $X_k = X_{k-1}$ ) và thực hiện như vậy 4 lần với mỗi PTCT khác nhau:

$$X_k^i = (X_{k-1} \circledast B^i) \cup A \quad i = 1, 2, 3, 4 \quad \text{and} \quad k = 1, 2, 3, \dots$$

- Cuối cùng hợp tất cả các  $X_k^i$  ( $i=1,2,3,4$ ) cuối cùng lại ta thu được kết quả.

- **Kết quả:**

Do thuật toán chạy khá lâu nên ta sẽ chọn ảnh đầu vào bé hơn:



Ảnh 11: Kết quả convex hull

⇒ Ta thu được kết quả là convex của cả hình, các ảnh in progress (0,1,2,3) lần lượt là quá trình tìm convex hướng bên phải, dưới, trái, trên. Thuật toán gấp khá nhiều vấn đề khi ảnh phức tạp và đầu ra chứa cả thành phần nhỏ trong ảnh. Kết quả của opencv cho ra kết quả tốt hơn do tách được 2 thành phần nhỏ và không đánh dấu tràn ra ngoài.

Lưu ý: ta có thể cải thiện miền convex tối đa để không tràn khỏi ảnh nhưng sẽ khiến thuật toán chạy lâu hơn khá nhiều nên ta sẽ không thực hiện ở đây.

### 3.1.10 Thinning

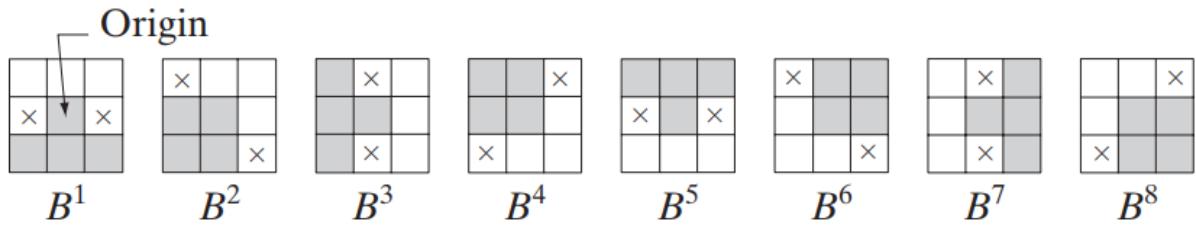
**Thinning** giúp làm mỏng các thành phần trong ảnh mà vẫn giữ được hình dáng gốc của ảnh, phù hợp đối với các ảnh đầu vào là chữ viết, hình dáng đơn giản...

- **Giải thuật:**

- Đầu vào:

- img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.

- Bs: danh sách các PTCT -1/0/1. Nếu không truyền vào thì sẽ sử dụng các thành phần sau:

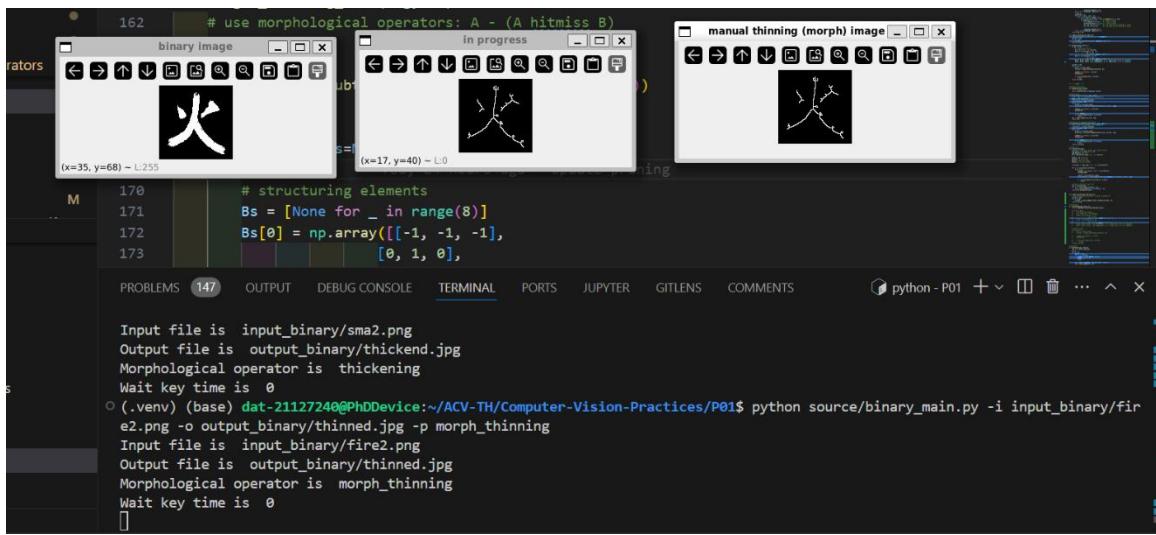


Ảnh 12: Các PTCT đầu vào mặc định cho thinning (nguồn ảnh: Digital Image Processing)

- Thực thi:
  - Khởi tạo các PTCT như trên nếu không được truyền vào.
  - Mỗi vòng lặp, thực hiện thinning qua từng PTCT trong danh sách Bs (lưu ý phải đi qua hết danh sách Bs ở mỗi vòng), lặp lại đến khi không thay đổi nữa thì dừng.
  - Trong đó, mỗi bước thinning thực hiện trừ ảnh đầu vào A với kết quả hit-or-miss của A với PTCT  $B_i$ :

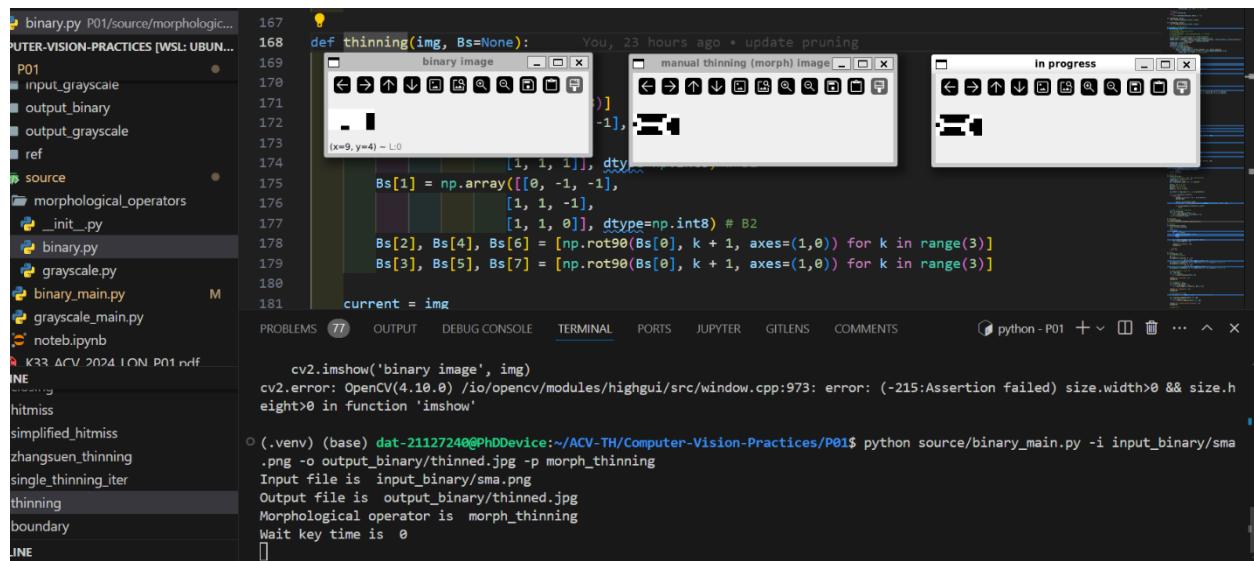
$$A \otimes B = A - (A \circledast B)$$

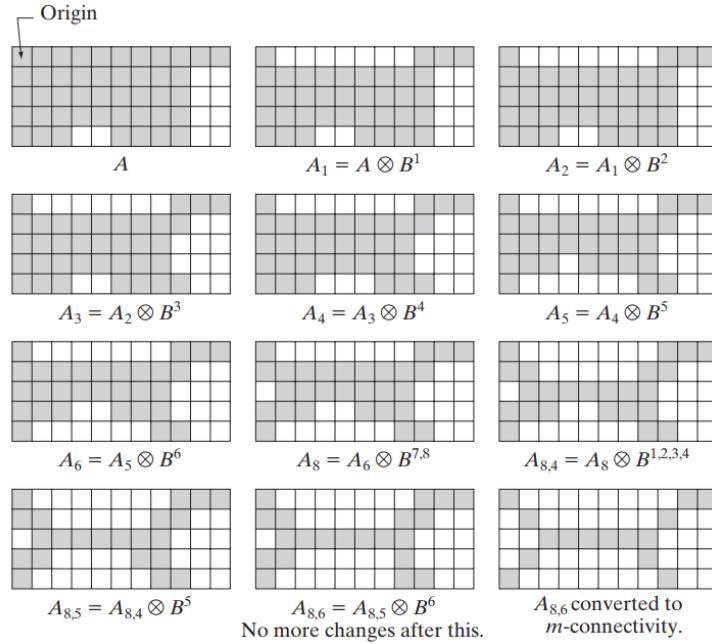
- **Kết quả:**



Ảnh 13: Kết quả thinning

⇒ Ta thu được ảnh mỏng hơn, có thể thấy thuật toán không hoạt động tốt với hình dạng ảnh có nhiều đường cong góc độ lớn dẫn đến có các chi tiết thừa. Ngoài ra, do thuật toán mặc định opencv cung cấp không có phương pháp dùng toán tử hình thái học (phương pháp được dùng là thuật toán Zhang-Suen sẽ được đề cập ở phần bổ sung) nên ta không so sánh với thư viện mà so sánh với kết quả ví dụ trong sách Digital Image Processing:





Ảnh 14: So sánh kết quả thinning (nguồn ảnh: Digital Image Processing)

### 3.1.11 Thickening

**Thickening**, ngược lại giúp làm dày các thành phần trong ảnh mà vẫn giữ được hình dáng gốc của ảnh.

- **Giải thuật:**

- Đầu vào:

- img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
- Bs: danh sách các PTCT -1/0/1. Nếu không truyền vào thì sẽ sử dụng các thành phần ngược lại với Bs mặc định ở thinning.

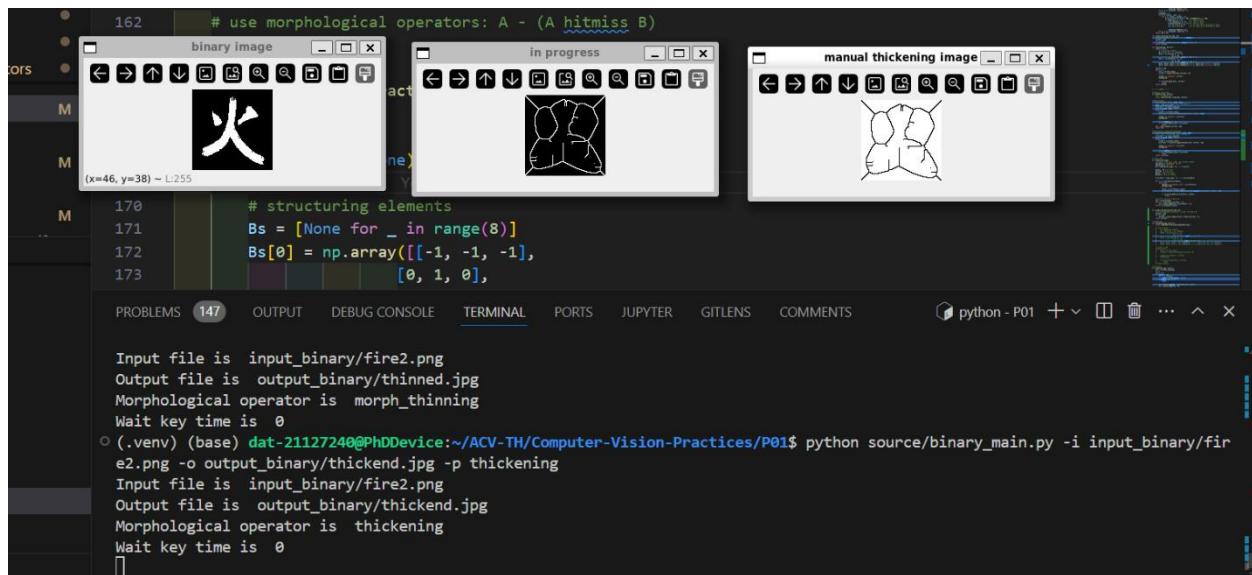
- Thực thi:

- Khởi tạo các PTCT như trên nếu không được truyền vào.
- Mỗi vòng lặp, thực hiện thickening qua từng PTCT trong danh sách Bs (lưu ý phải đi qua hết danh sách Bs ở mỗi vòng), lặp lại đến khi không thay đổi nữa thì dừng.

- Trong đó, mỗi bước thinning thực hiện hợp ảnh đầu vào A với kết quả hit-or-miss của A với PTCT B<sub>i</sub>:

$$A \odot B = A \cup (A \circledast B)$$

- Cách 2, tận dụng thuật toán thinning đã viết:
  - Do thickening là duality của thinning, ta có thể thực hiện thinning của ảnh nghịch (A<sub>c</sub>) rồi sau đó lấy ảnh nghịch của kết quả.
- **Kết quả:**



Ảnh 15: Kết quả thickening

- ⇒ Thu được hình ảnh dày hơn nhưng hình dạng khá bất thường do ảnh đầu vào có hình dáng bất thường:

```

binary image in progress manual thickening image
(x=4, y=0) ~ L:0 (x=6, y=1) ~ L:255

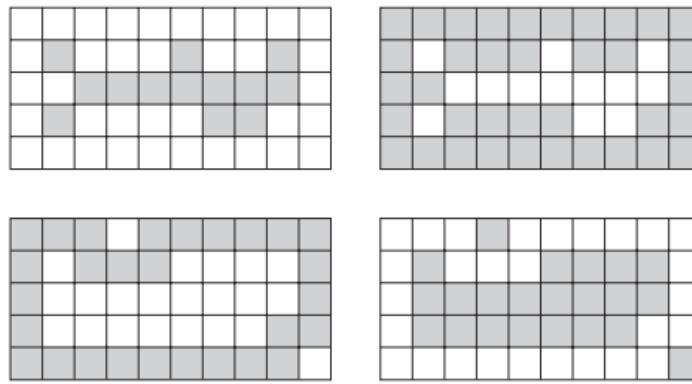
M 92 # return intersect_image(eroded_img, eroded_complement)
93 kernel_center = (kernel.shape[0] // 2, kernel.shape[1] // 2)
94 padded_img = np.pad(img, ((kernel_center[0], kernel_center[0]), (kernel_center[1], kernel_center[1])), mode='constant', constant_values=0)
95

PROBLEMS 45 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER GITLENS COMMENTS python - P01 ... x

```

Input file is input\_binary/fire2.png  
Output file is output\_binary/thickend.jpg  
Morphological operator is thickening  
Wait key time is 0

(.venv) (base) dat\_21127240@PhDDevice:~/ACV-TH/Computer-Vision-Practices/P01\$ python source/binary\_main.py -i input\_binary/sma2.png -o output\_binary/thickend.jpg -p thickening  
Input file is input\_binary/sma2.png  
Output file is output\_binary/thickend.jpg  
Morphological operator is thickening  
Wait key time is 0



Ảnh 16: So sánh kết quả thickening (nguồn ảnh: Digital Image Processing)

### 3.1.12 Skeletons

**Skeletons** giúp thu nhận khung xương của các vật thể trong ảnh.

- **Giải thuật:**

- Đầu vào:
  - img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
  - kernel: PTCT 0/1.
- Thực thi:

- Ảnh khung xương là hợp tất cả các  $S_k$  với  $K$  là  $k$  lớn nhất mà  $S_k$  không phải là ảnh chứa toàn số 0 (đến điều kiện này thì dừng):

$$S(A) = \bigcup_{k=0}^K S_k(A)$$

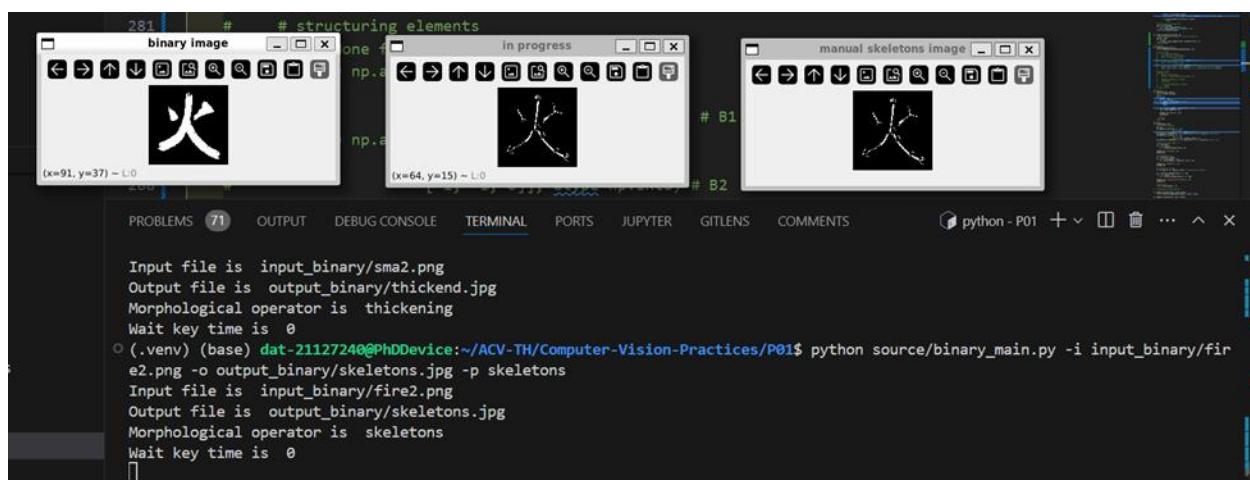
- Với  $S_k$  là:

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$$

- Trong đó  $\text{erode}_k$  tức là  $\text{erode}$  ảnh A với kernel B k lần:

$$(A \ominus kB) = (((\dots(((A \ominus B) \ominus B) \ominus \dots) \ominus B)$$

- Kết quả:

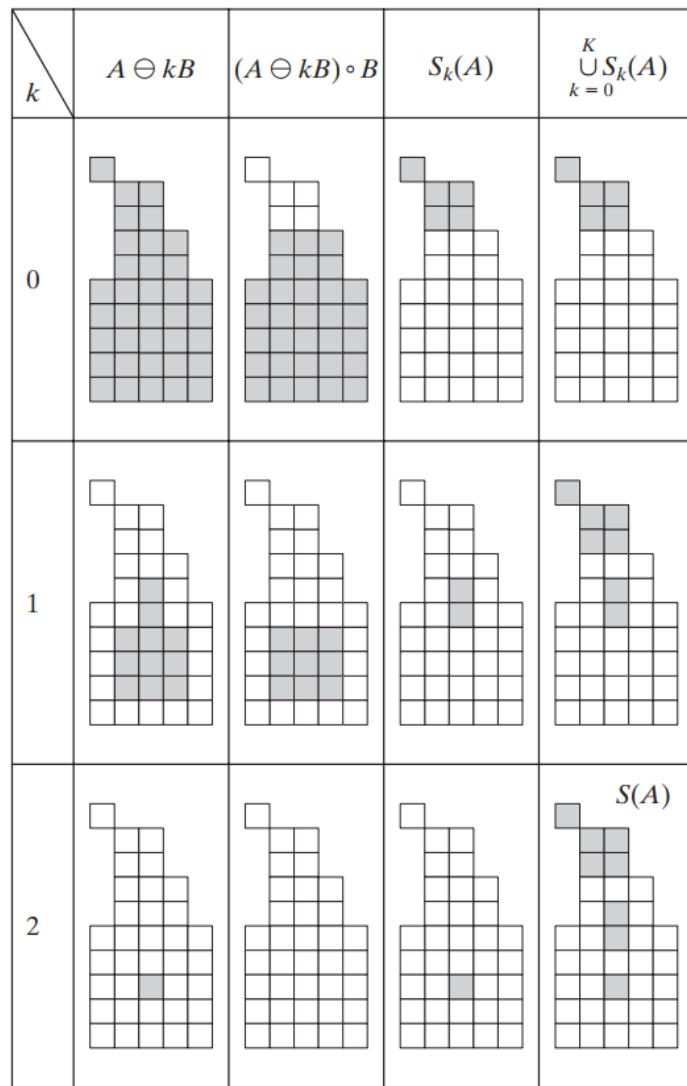


Ảnh 17: Kết quả skeletons

- ⇒ Thu được khung xương của ảnh, có thể thấy các đỉnh ngoài của ảnh nếu có hình dạng bầu hoặc to hơn thân sẽ khiến thuật toán rẽ nhánh thành nhiều khung xương khác nữa (do ta dùng kernel đầu vào nhỏ).

The screenshot shows a Jupyter Notebook environment with several windows:

- binary image**: Shows a binary image of a black silhouette of a person on a white background.
- img\_out = img\_thickening\_manual**: Shows the result of a manual thickening operation on the binary image.
- img\_skeletons\_manual**: Shows the result of a manual skeletonization operation on the binary image.
- manual skeletons image**: Shows the final manual skeletonized image.
- Code Editor**: Displays Python code for performing morphological operations like thickening and pruning on a binary image.
- Terminal**: Shows the command used to run the script: `python source/binary_main.py -i input_binary/sma3.png -o output_binary/skeletons.jpg -p skeletons`.



Ảnh 18: So sánh kết quả skeletons (nguồn ảnh: Digital Image Processing)

### 3.1.13 Pruning

**Pruning** giúp cắt bỏ các nhánh thừa và nhỏ của ảnh. Thường được dùng như bước hậu xử lý của thinning và skeletons do 2 thuật toán này thường sinh ra các nhánh thừa trong quá trình tạo ảnh đầu ra.

- **Giải thuật:**

- Đầu vào:

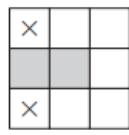
- img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
    - n: Số lần thực hiện các bước con (mặc định là 3).

- Thực thi:

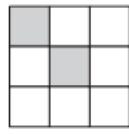
- Đầu tiên thực hiện thinning với PTCT là tập hợp B n lần:

$$X_1 = A \otimes \{B\}$$

- Tập hợp B được định nghĩa là, với B2, B3, B4 là phiên bản B1 được quay góc 90, 180, 270 độ theo chiều kim đồng hồ:



$$B^1, B^2, B^3, B^4 \text{ (rotated } 90^\circ)$$



$$B^5, B^6, B^7, B^8 \text{ (rotated } 90^\circ)$$

- Tiếp theo thực hiện giao 8 kết quả hitmiss của X1 với PTCT B<sub>k</sub> để lấy các điểm đầu đuôi mà của ảnh:

$$X_2 = \bigcup_{k=1}^8 (X_1 \circledast B^k)$$

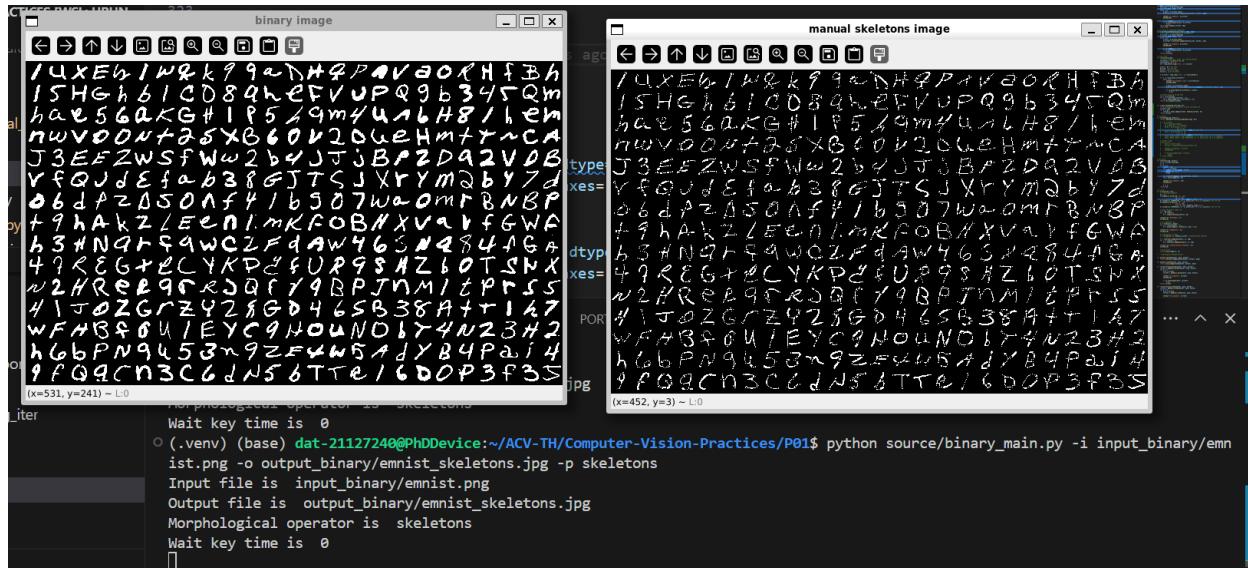
- Sau đó thực hiện giao kết quả dilation của X2 với H và ảnh gốc A, trong đó H là kernel 3x3 chứa toàn số 1, xây dựng ảnh lại từ điểm các đầu đuôi:

$$X_3 = (X_2 \oplus H) \cap A$$

- Cuối cùng là hợp X1 với X3 ta được kết quả.

- Kết quả:**

Để thấy ứng dụng thì đầu tiên giả sử có một ảnh đã thực hiện skeletons và sau đó thực hiện pruning để cho ra kết quả không có các khung xương không cần thiết.



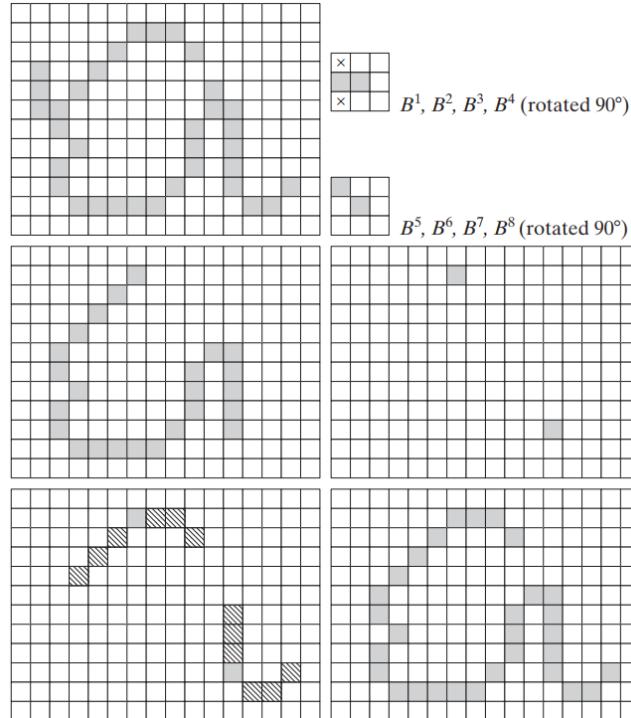
Ảnh 19: Kết quả skeletons



Ảnh 20: Kết quả pruning

⇒ Thu được ảnh đã loại bỏ các đường (khung xương) không đáng chú ý.

```
notebook.ipynb M binary_main.py M binary.py M
binary image X1 (thinning) X2 (endpoints) X3 (reconstruction dilation)
manual pruning image
Input file is output_binary/emnist_skeletons.jpg
Output file is output_binary/emnist_pruned.jpg
Morphological operator is pruning
Wait key time is 0
(.venv) (base) dat-21127240@PhDDevice:~/ACV-TH/Computer-Vision-Practices/P01$ python source/binary_main.py -i input_binary/letter_a.png -o output_binary/a_pruned.jpg -p pruning
Input file is input_binary/letter_a.png
Output file is output_binary/a_pruned.jpg
Morphological operator is pruning
Wait key time is 0
```



Ảnh 21: So sánh kết quả pruning (nguồn ảnh: Digital Image Processing)

### 3.1.14 Morphological Reconstruction

#### 1. Reconstruction erosion

- **Giải thuật:**

- Định nghĩa: Geodesic erosion là thực hiện erosion marker F với B rồi sau đó lấy kết quả đó giao với mask G, trong đó marker là ảnh trong quá trình reconstruction, mask là ảnh giới hạn quá trình reconstruction, B là PTCT:

$$D_G^{(1)}(F) = (F \oplus B) \cap G \quad D_G^{(n)}(F) = D_G^{(1)}[D_G^{(n-1)}(F)]$$

- Reconstruction erosion là thực hiện geodesic erosion cho đến khi kết quả không thay đổi nữa.

#### 2. Reconstruction dilation: tương tự

Do erosion và dilation nằm trong opening và closing nên ta sẽ thể hiện chúng ở opening và closing:

#### 3. Opening by reconstruction

Giúp khôi phục ảnh đã bị xói mòn.

- **Giải thuật:**

- Đầu vào:

- img: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
- kernel: PTCT mặc định là 0/1.
- n: số lần muốn thực hiện xói mòn ảnh.

- Thực thi:

- Thực hiện xói mòn (erode) ảnh n lần.
- Sau đó thực hiện reconstruction dilation.

- **Kết quả:**



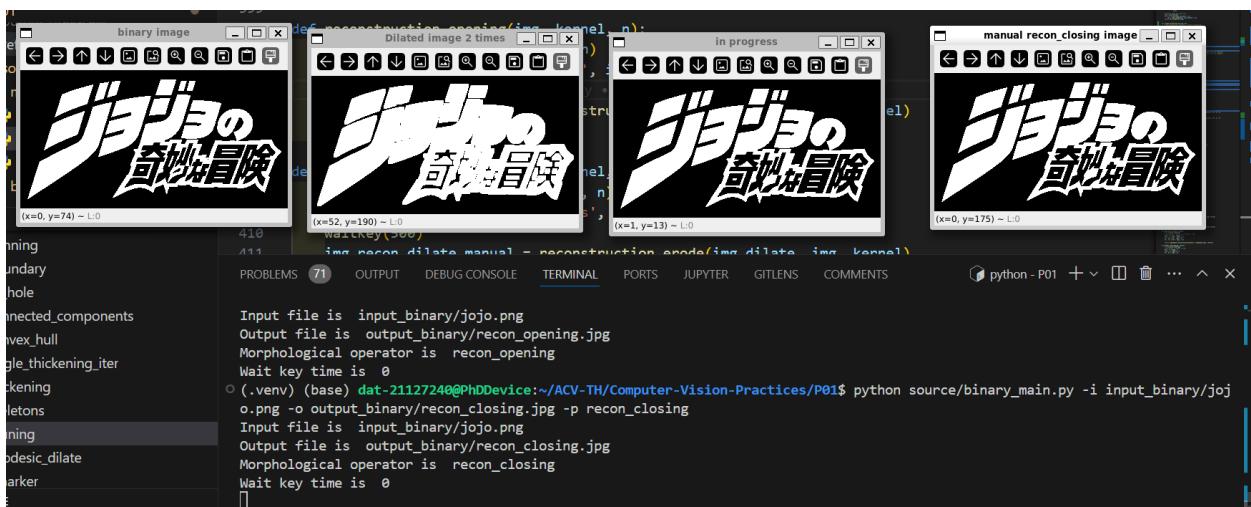
Ảnh 22: Kết quả reconstruction opening

⇒ Thuật toán có thể khôi phục hầu hết các chi tiết ảnh ngoại trừ các chi tiết nhỏ và không phù hợp với PTCT đầu vào đã bị erosion xóa mất thì không thể khôi phục được.

#### 4. Closing by reconstruction

Tương tự như trên

#### Kết quả:



Ảnh 23: Kết quả reconstruction closing

⇒ Tương tự, thuật toán có thể khôi phục hầu hết các chi tiết ảnh ngoại trừ các chi tiết đã bị dilation che lấp mất thì không thể khôi phục được (một số các lỗ và khe hở nhỏ).

## 3.2 Toán tử hình thái học với ảnh độ xám

### 3.2.1 Erosion

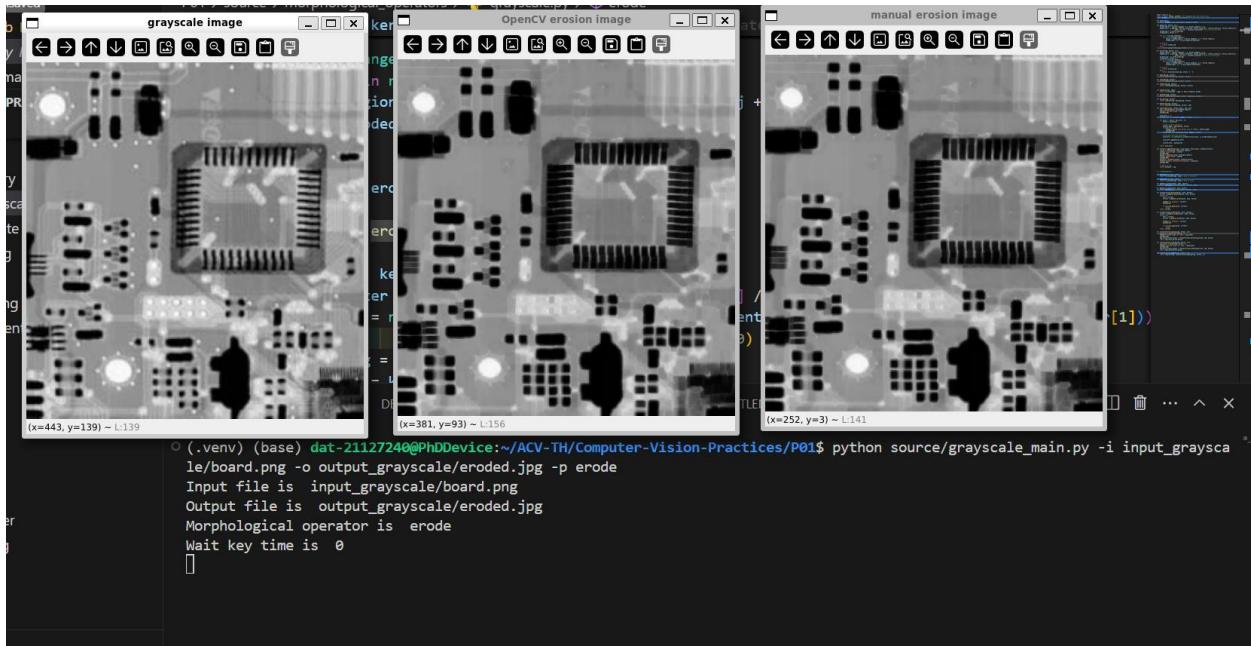
Toán tử erosion (xói mòn) là một phép toán hình thái học cơ bản trong xử lý ảnh, được sử dụng để làm mờ các vùng sáng trên ảnh. Nó có tác dụng loại bỏ các điểm ảnh biên và làm giảm kích thước của các vùng sáng.

- **Giải thuật:**

- Đầu vào:
  - `img`: Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
  - `kernel`: Phần tử cấu trúc (structuring element/kernel) với các giá trị 0/1 và sẽ mặc định tâm nằm ở giữa, ta sẽ gọi tắt là PTCT.
  - `n`: Số lần thực hiện phép toán.
- Thực thi:
  - Đệm (padding) ảnh thêm với phân nửa kích thước dài/rộng của phần tử cấu trúc (giúp đảm bảo có thể thực hiện được trên biên).
  - Khởi tạo ảnh đầu ra là kích thước bằng với `img` và mang giá trị 0.
  - Với mỗi điểm ảnh  $(i,j)$  trên ảnh gốc:
    - Xác định vùng ảnh cần xét ứng với PTCT, chỉ xét các vùng mang giá trị 1.
    - Gán vị trí  $(i,j)$  của ảnh đầu ra là **giá trị nhỏ nhất** của vùng ảnh đang xét.
- Lặp lại thao tác trên bằng số lần mong muốn `n`.
- Trả về kết quả.

- **Kết quả:**

Lưu ý: Nếu không đề cập thì mặc định, chương trình sẽ sử dụng PTCT là một hình đĩa phẳng kích thước  $3 \times 3$  mang toàn bộ (các phép toán khác cũng tương tự).



Ảnh 24: Kết quả erosion

⇒ Các vùng màu tối được mở rộng.

### 3.2.2 Dilation

Toán tử dilation (giãn nở) là một phép giúp mở rộng các vùng sáng trong ảnh nhị phân hoặc ảnh xám, làm cho các đối tượng trong ảnh trở nên to hơn và kết nối các vùng sáng gần nhau.

- **Giải thuật:**

- Đầu vào:
  - **img:** Ảnh nhị phân với các điểm ảnh mang giá trị 0/255.
  - **kernel:** Phần tử cấu trúc (structuring element/kernel) với các giá trị 0/1 và sẽ mặc định tâm nằm ở giữa, ta sẽ gọi tắt là PTCT.
  - **n:** Số lần thực hiện phép toán.

- Thực thi:
  - Đệm (padding) ảnh thêm với phân nửa kích thước dài/rộng của phần tử cấu trúc (giúp đảm bảo có thể thực hiện được trên biên).
  - Khởi tạo ảnh đầu ra là kích thước bằng với img và mang giá trị 0.
  - Với mỗi điểm ảnh (i,j) trên ảnh gốc:
    - Xác định vùng ảnh cần xét ứng với PTCT, chỉ xét các vùng mang giá trị 1.
    - Gán vị trí (i,j) của ảnh đầu ra là giá trị lớn nhất của vùng ảnh đang xét.
- Lặp lại thao tác trên bằng số lần mong muốn n.
- Trả về kết quả.

- **Kết quả:**



Ảnh 25: Kết quả erosion

⇒ Các vùng màu sáng được mở rộng.

### 3.2.3 Opening

Tương tự như binary

- Kết quả:



### 3.2.4 Closing

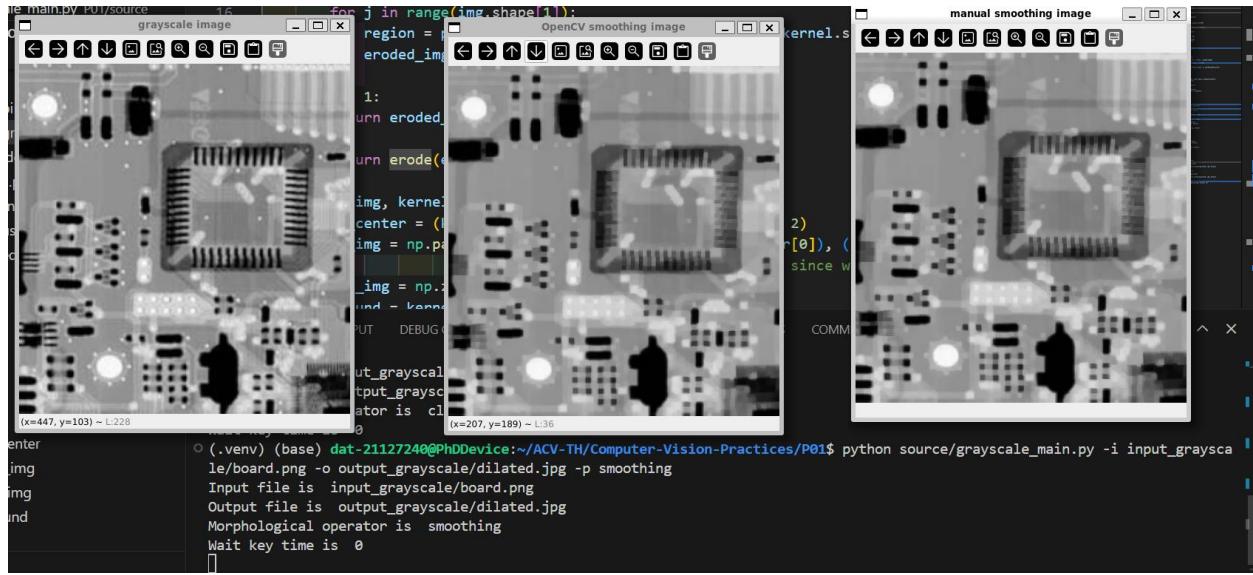
Tương tự

- Kết quả:



### 3.2.5 Morphological smoothing

- **Giải thuật:**
  - Thực hiện closing với ảnh sau khi opening
- **Kết quả:**



### 3.2.6 Morphological gradient

- **Giải thuật:**
  - Ảnh sau khi dilate trừ ảnh sau khi erode
- **Kết quả:**



### 3.2.7 Top-hat and bottom-hat transformations

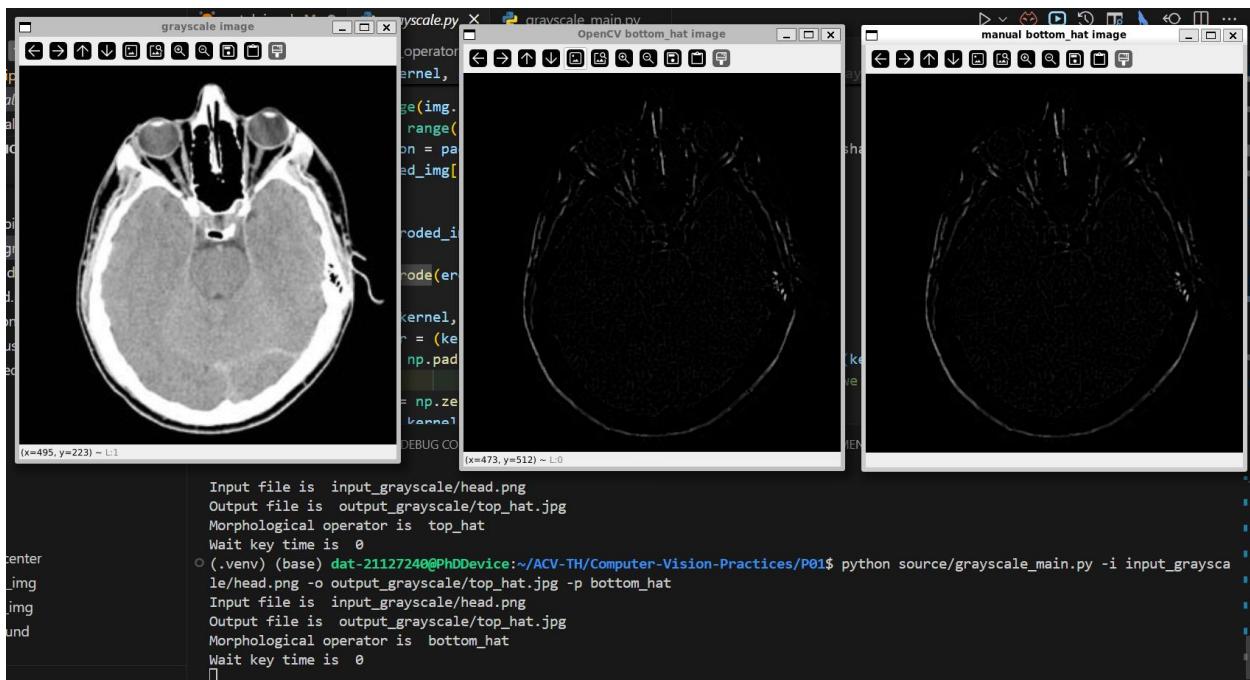
Tương tự

- Kết quả:

Top hat:



Bottom hat:



### 3.2.8 Granulometry

- **Kết quả:**

### 3.2.9 Textural segmentation

- Cách sử dụng:
- Giải thuật:
- Kết quả:

### 3.2.10 Morphological Reconstruction

- Cách sử dụng:
- Giải thuật:
- Kết quả:

## 4 Kết luận

## 5 Tham khảo

Rafael C. Gonzalez, R. E. (2008). *Digital Image Processing 3rd Edition.*