Digital Image Processing

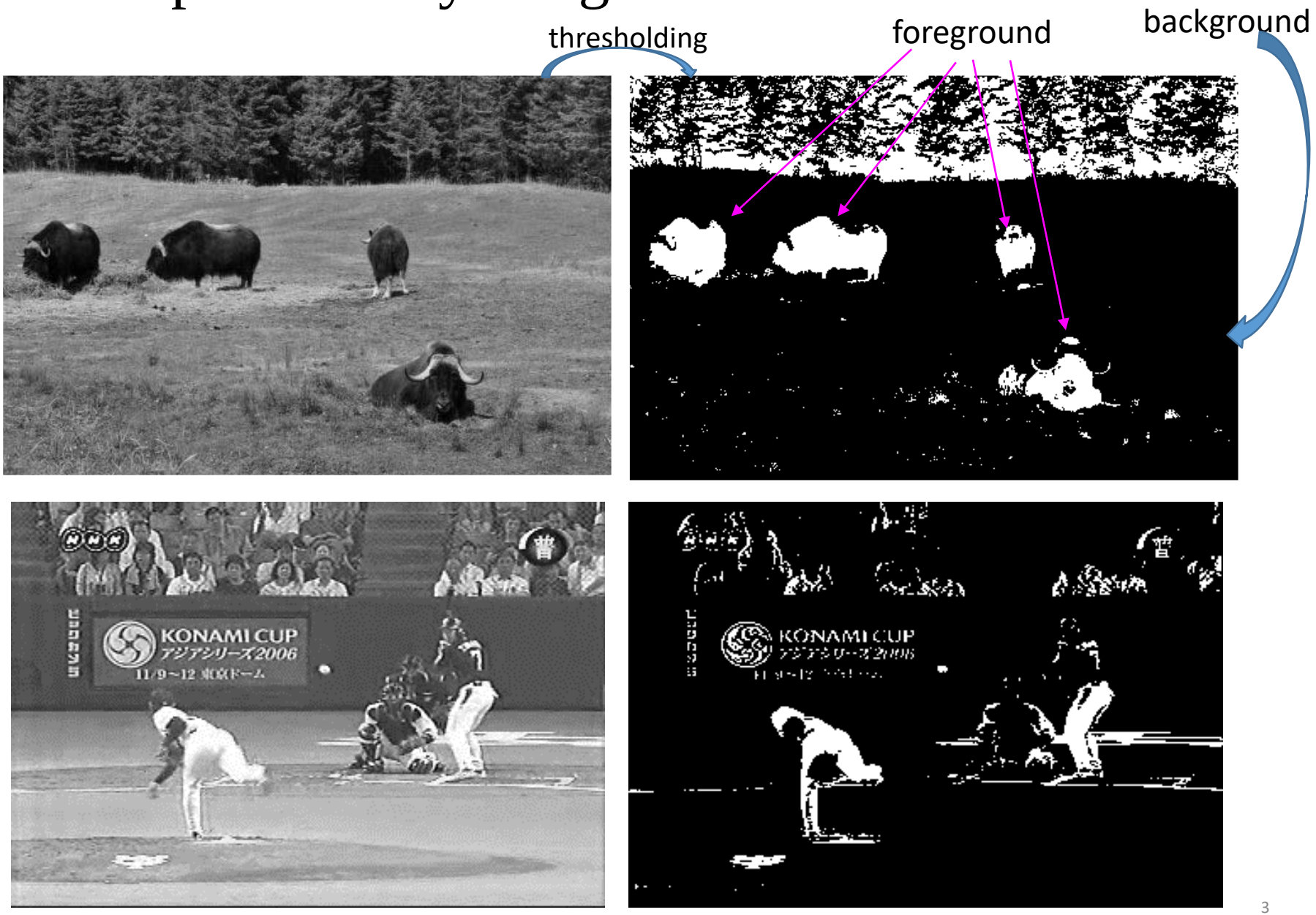
Chapter 4
Morphological Image Processing A

MingHan Tsai
Department of Information Engineering and Computer Science,
Feng Chia University

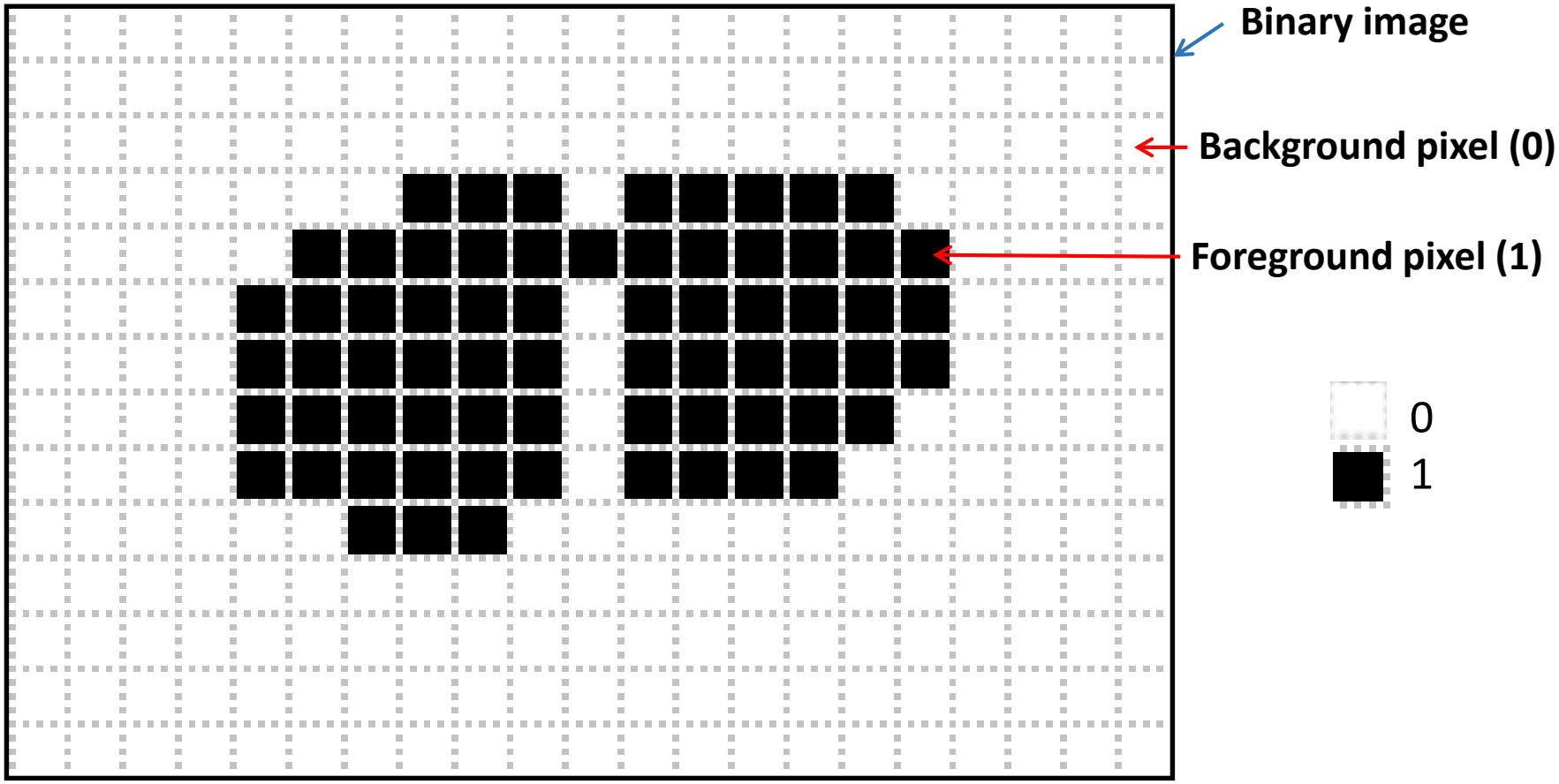
Morphological Image Processing

- Erosion(侵蝕) and Dilation(膨脹)
- Opening(斷開) and Closing(閉合)
- Boundary Extraction
- Hole Filling
- Hit-or-Miss Transformation
- Connected Component Analysis/ Labeling

Example – binary image



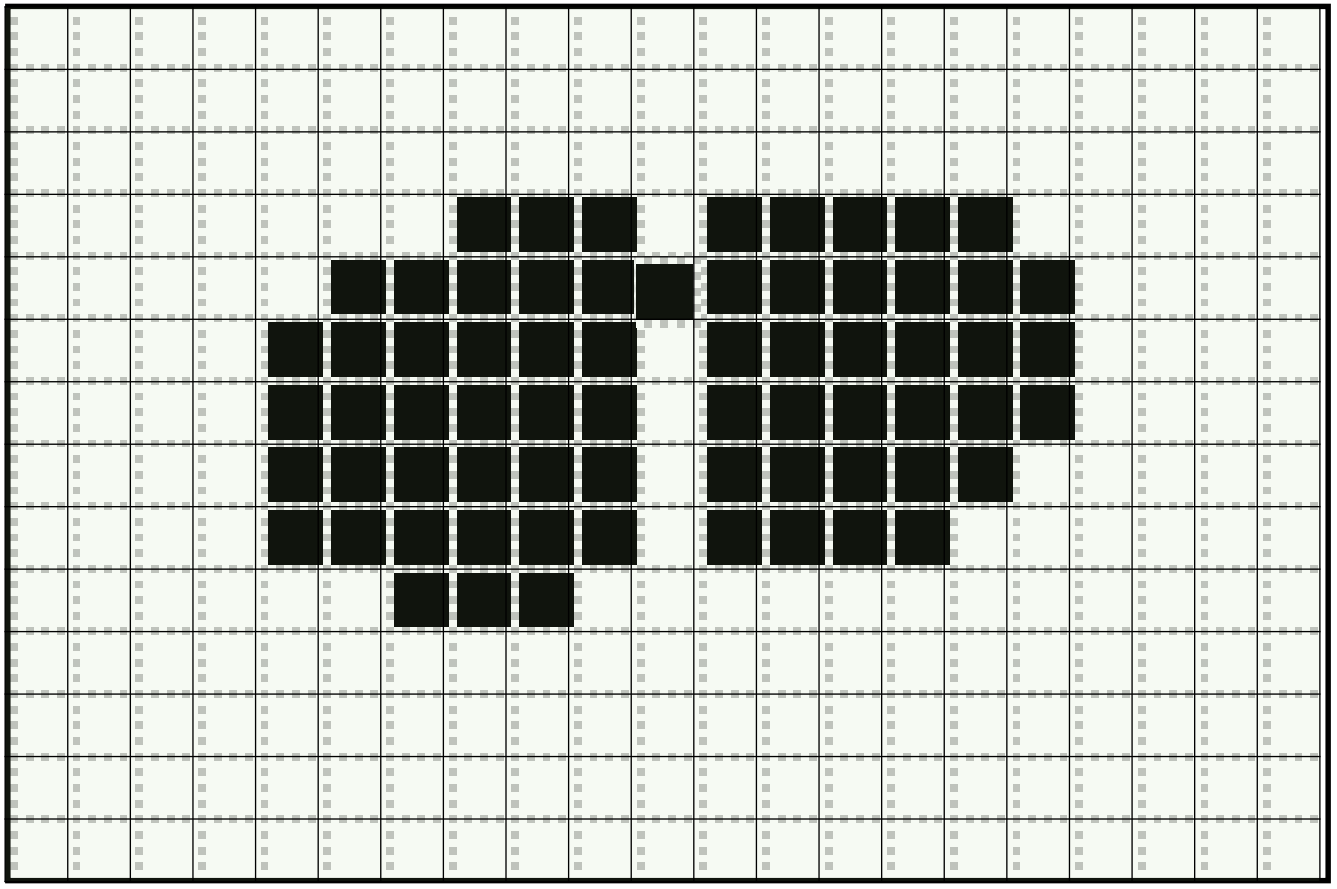
Morphological Operation



Morphological Operation

Morphological Dilation: take the maximum under the kernel

Morphological Erosion: take the minimum under the kernel



0

1

kernel

4-connected

	1	
1	1	1
	1	

☆: anchor

8-connected

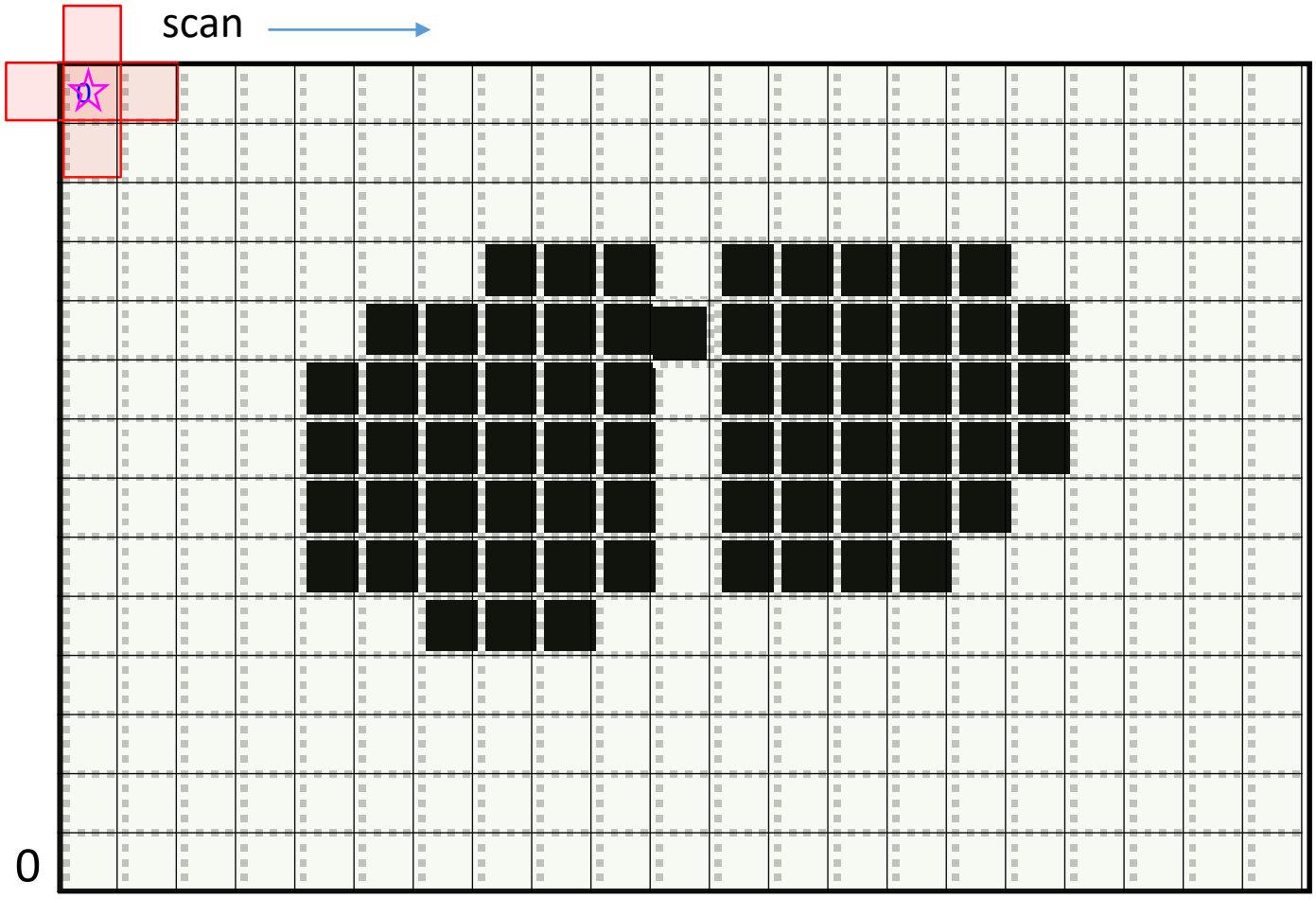
1	1	1
1	1	1
1	1	1

Kernel, Mask, Structuring Element

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

	1	
1	1	1
	1	

☆: anchor

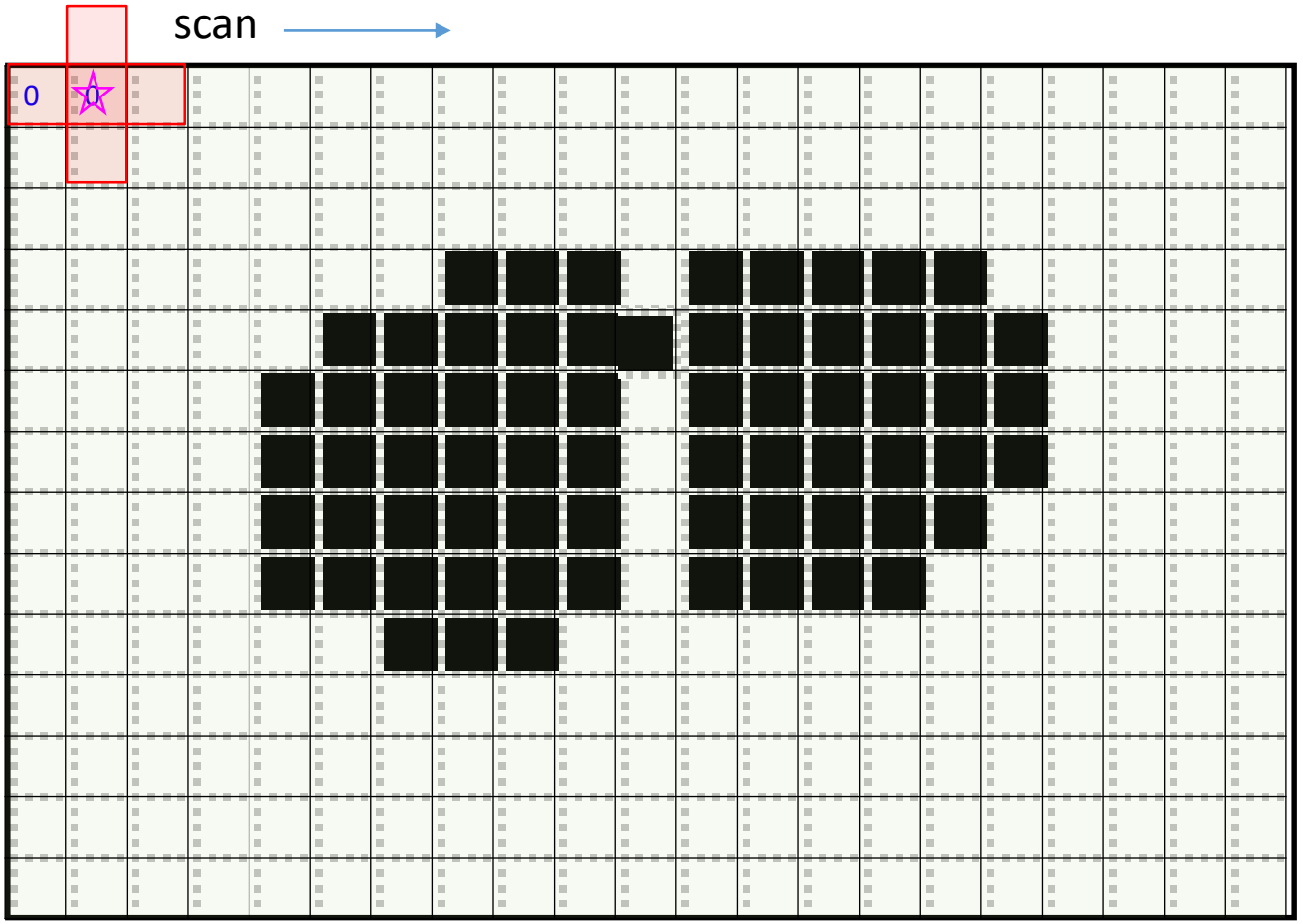
8-connected

1	1	1
1	1	1
1	1	1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

	1	
1	1	1
	1	

☆: anchor

8-connected

1	1	1
1	1	1
1	1	1

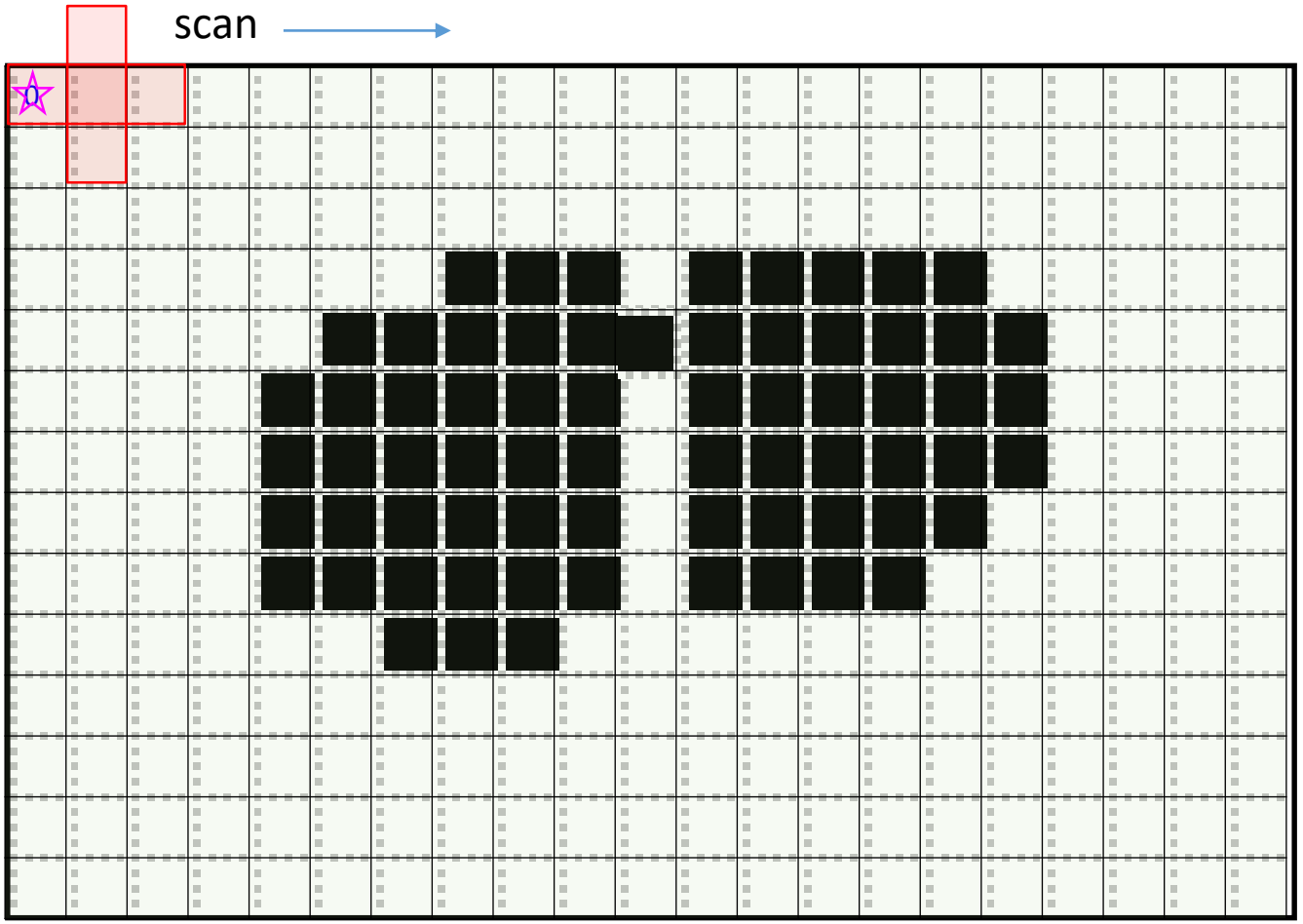
0

1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

	1	
1	1	1
	1	

☆: anchor

8-connected

1	1	1
1	1	1
1	1	1

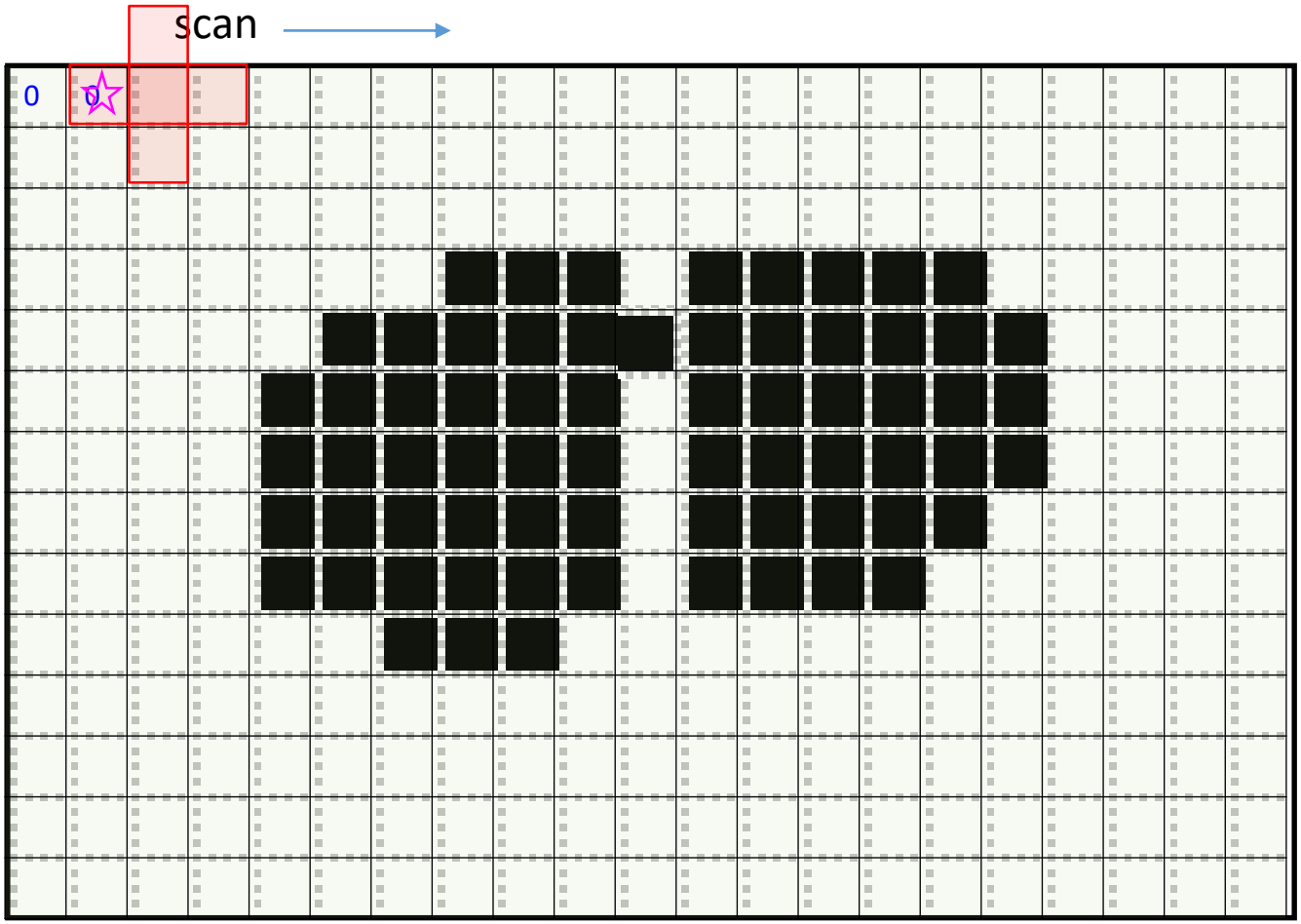
0

1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

	1	
1	1	1
	1	

☆: anchor

8-connected

1	1	1
1	1	1
1	1	1

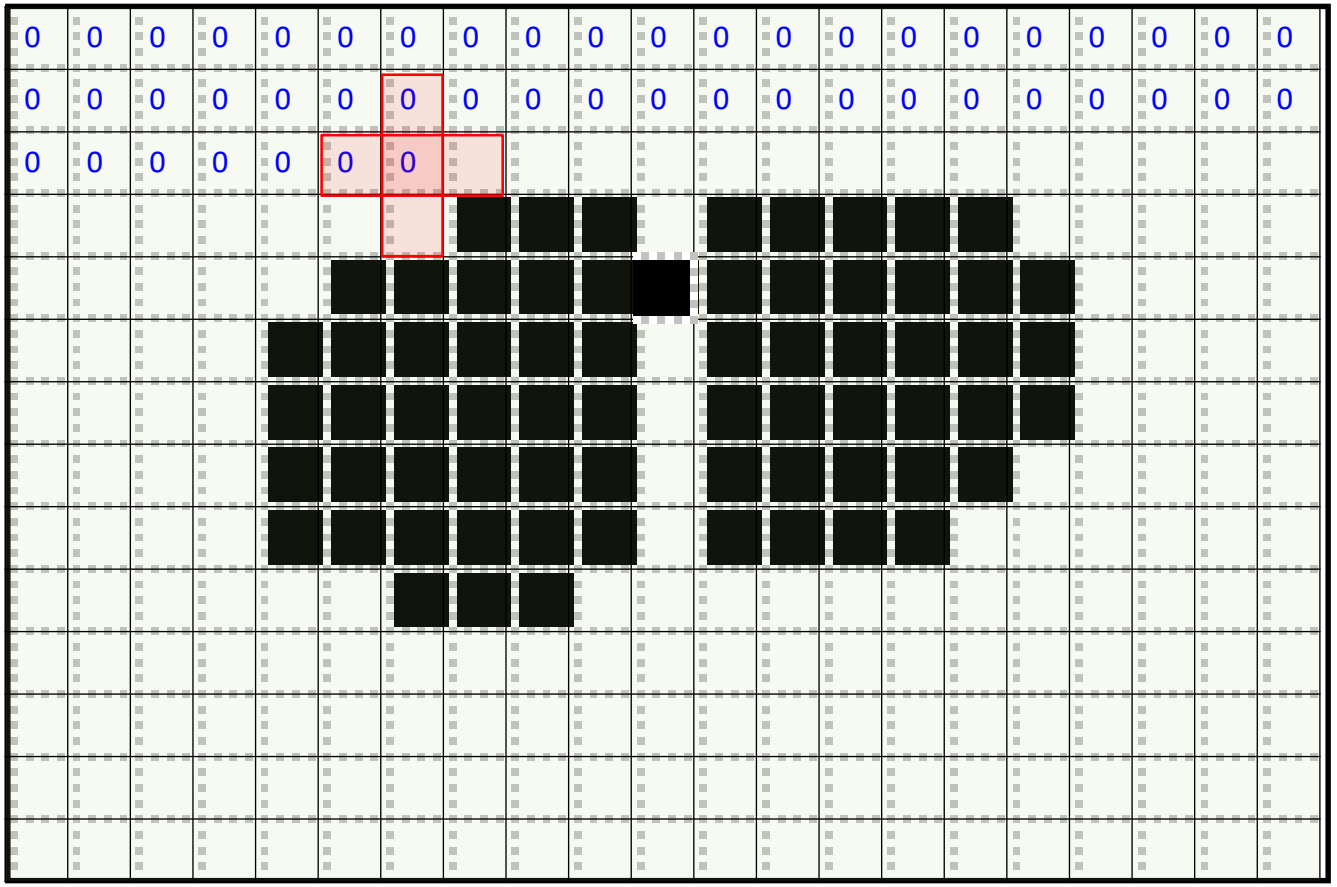
0

1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

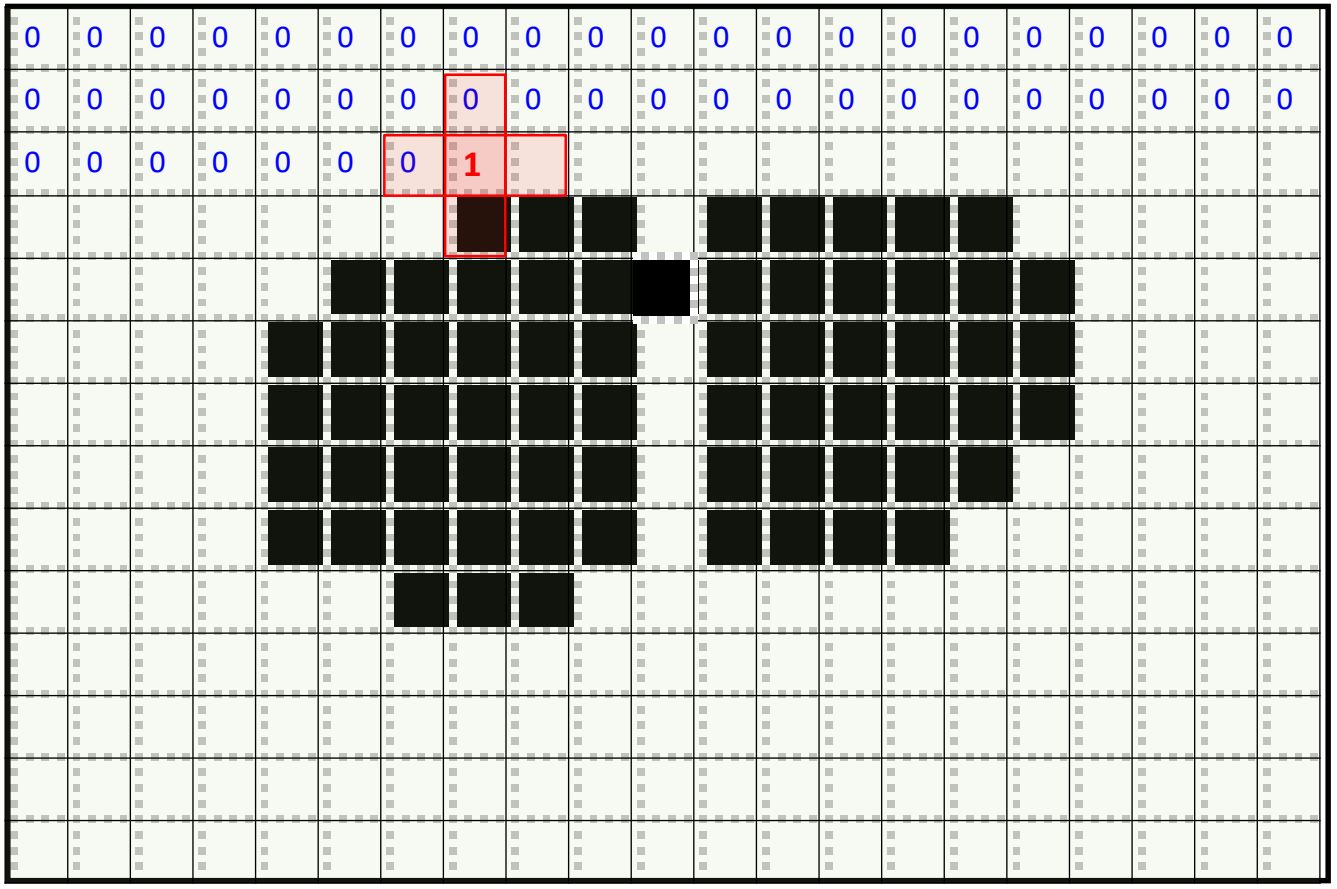
8-connected

1 1 1
1 1 1
1 1 1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

1
1 1 1
1

☆: anchor

8-connected

1 1 1
1 1 1
1 1 1

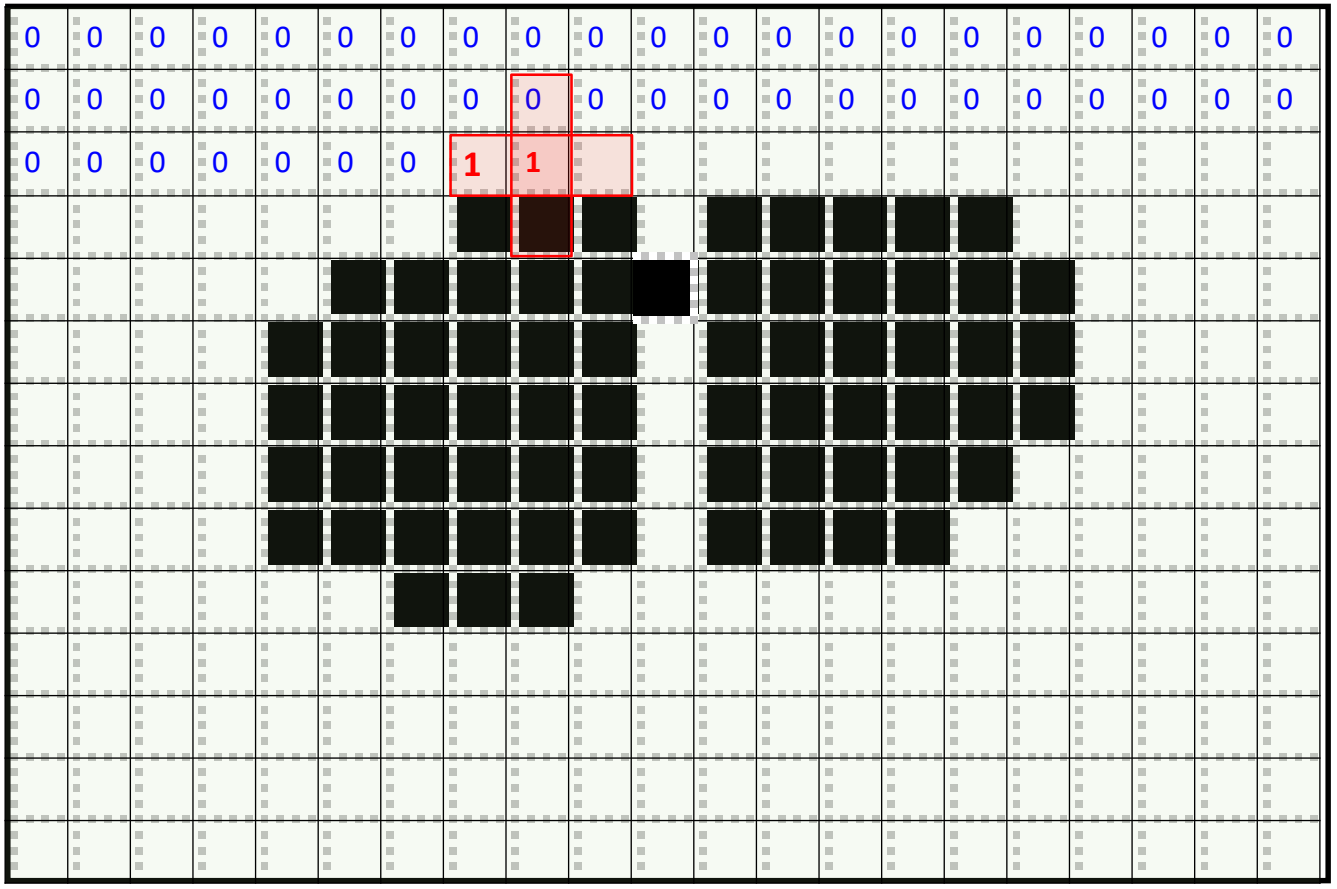
0

1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

1
1 1 1
1

☆: anchor

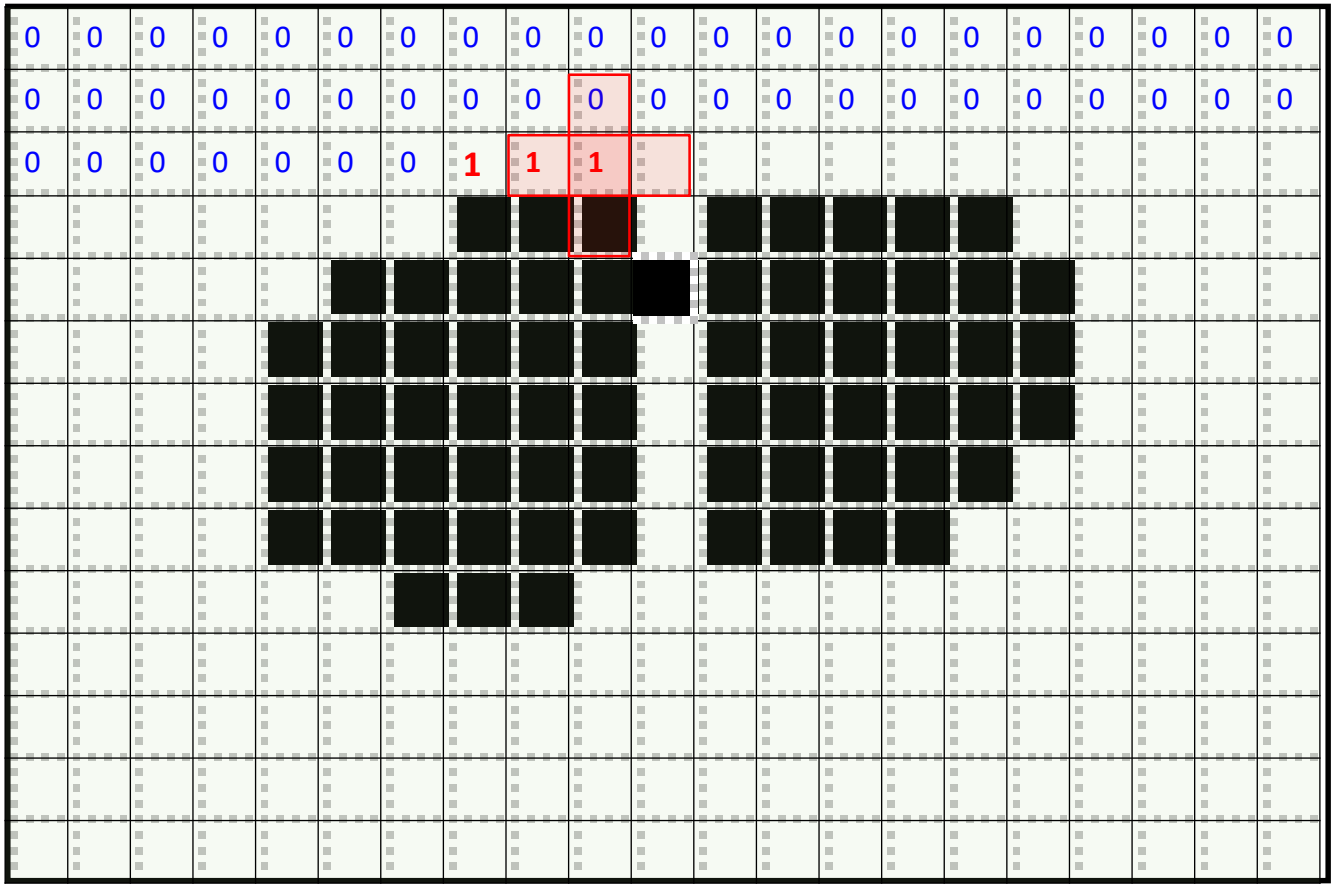
8-connected

1 1 1
1 1 1
1 1 1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

1
1 1 1
1

☆: anchor

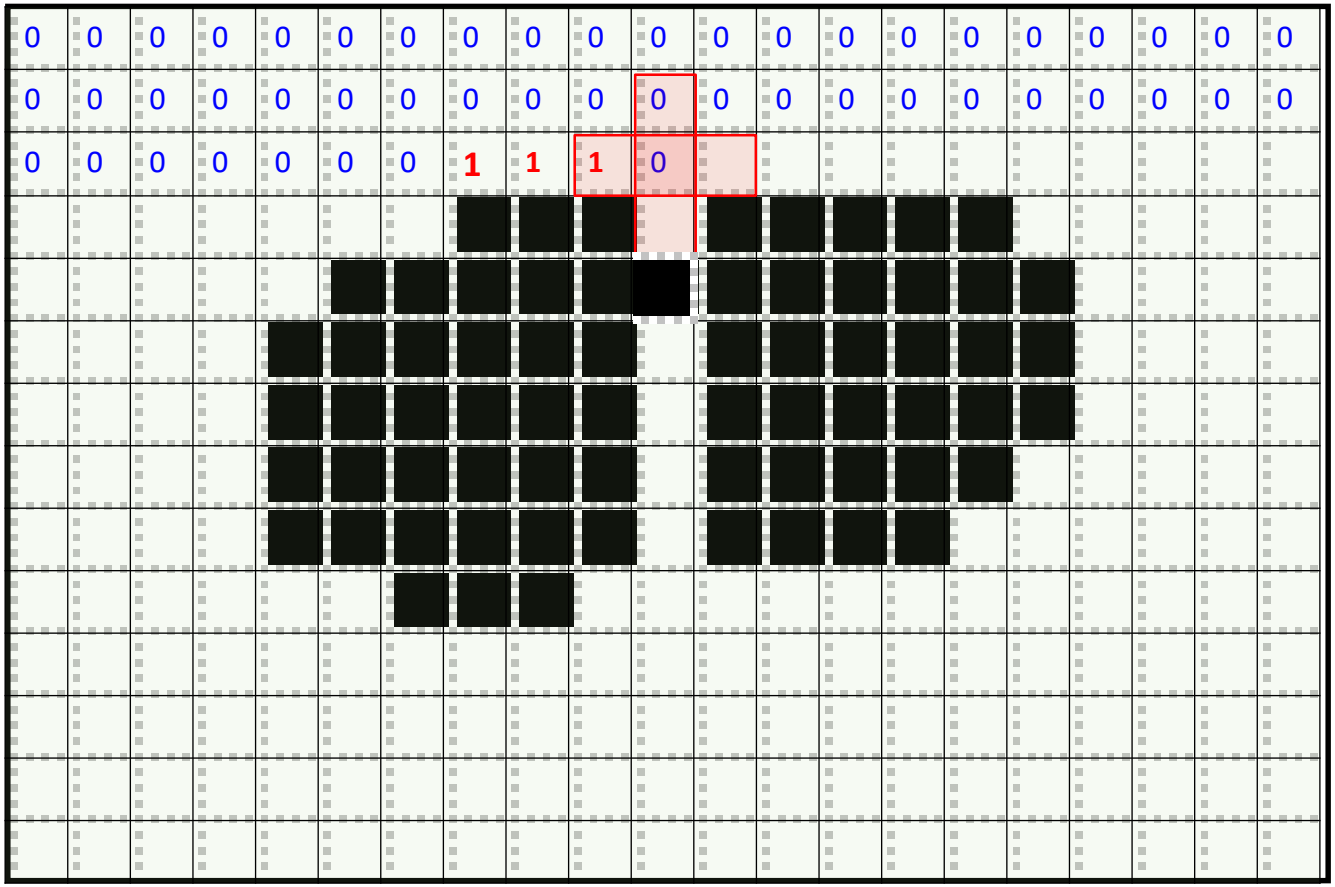
8-connected

1 1 1
1 1 1
1 1 1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



kernel

4-connected

1
1 1 1
1

☆: anchor

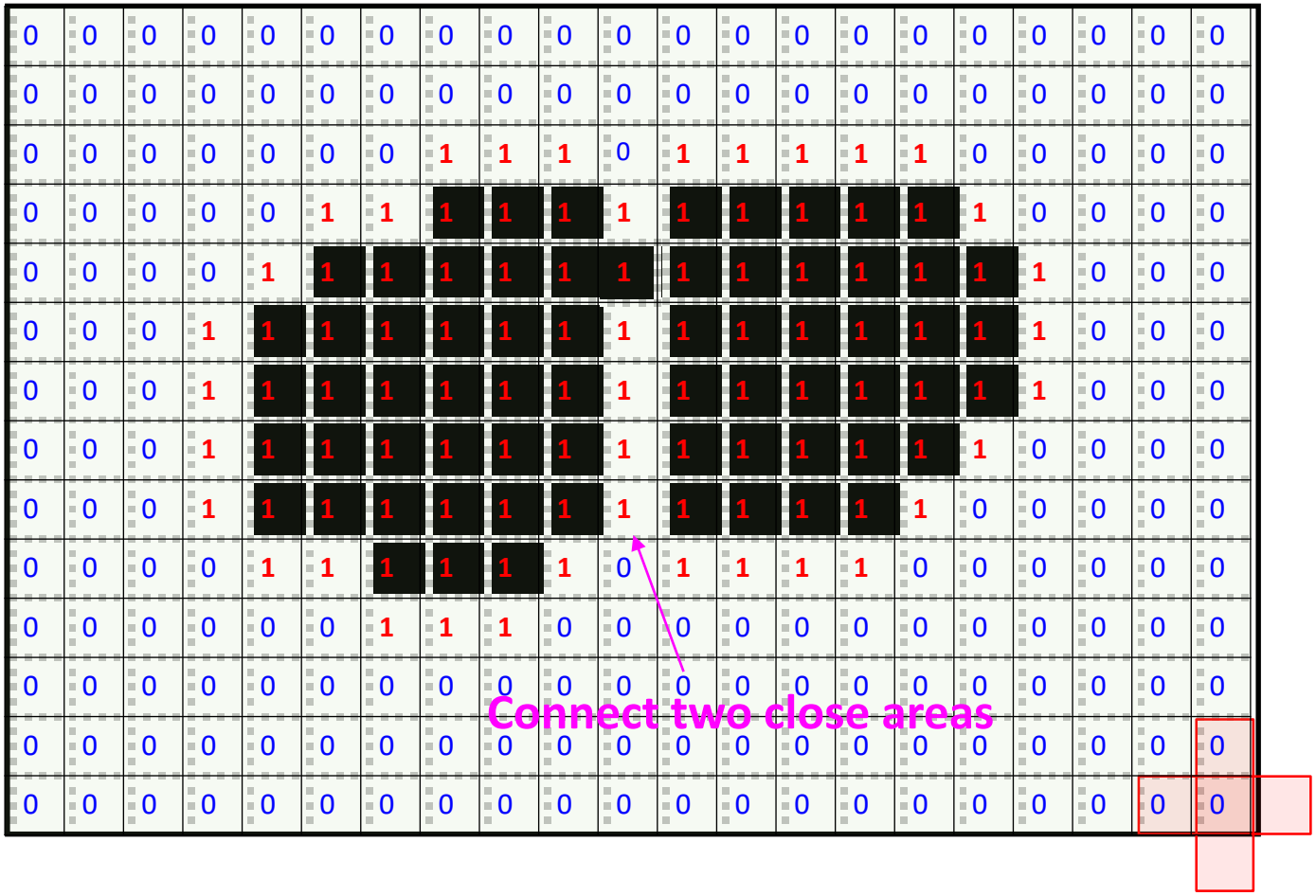
8-connected

1 1 1
1 1 1
1 1 1

Dilation $A \oplus B$

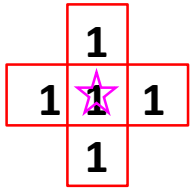
A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



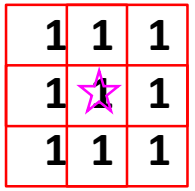
kernel

4-connected



☆: anchor

8-connected



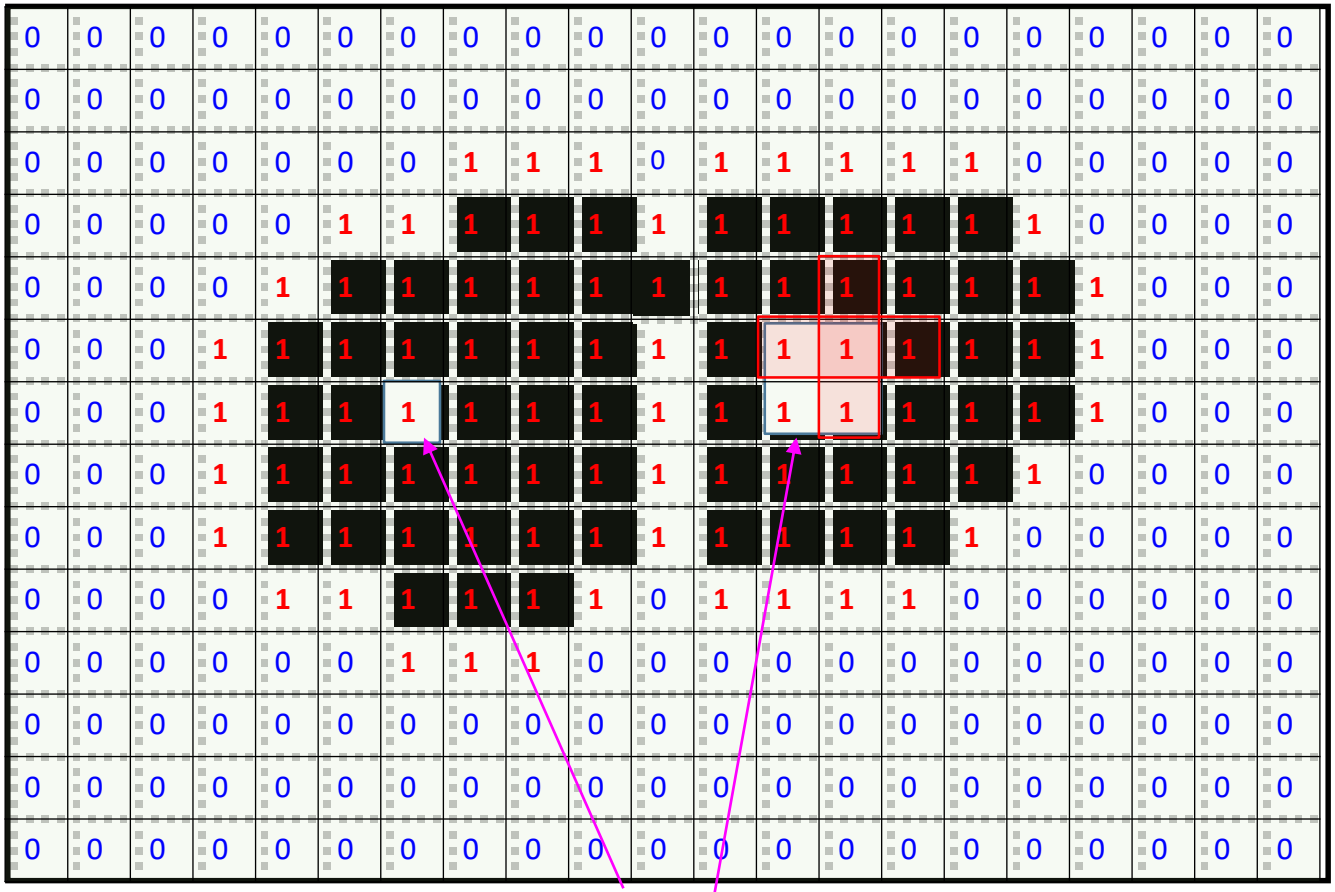
0

1

Dilation $A \oplus B$

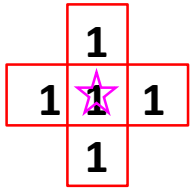
A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)



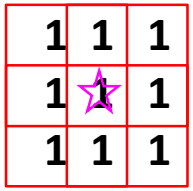
kernel

4-connected



☆: anchor

8-connected



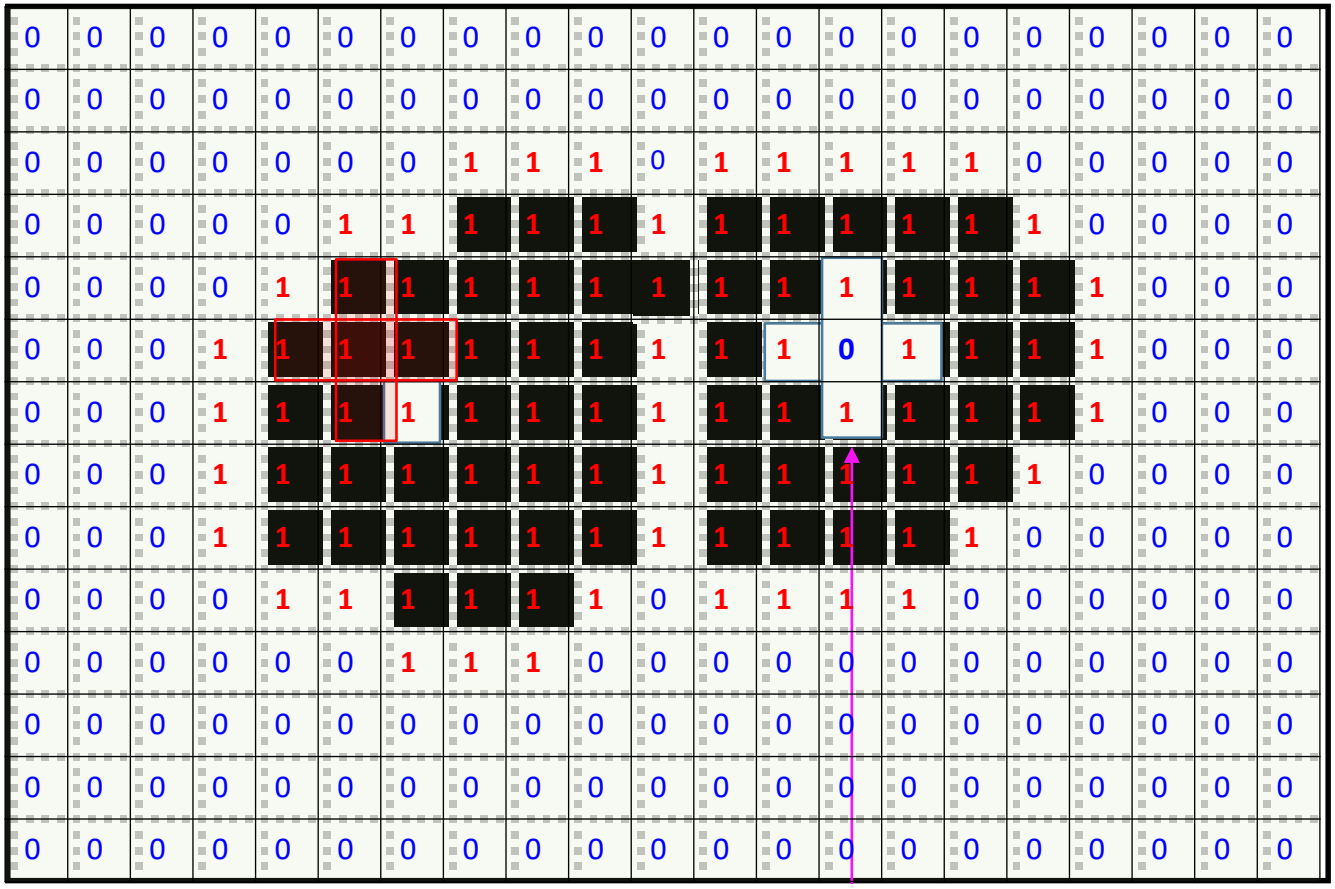
0

1

Dilation $A \oplus B$

A: image, B: kernal

Take the maximum under the kernel (op: pixel-wise AND, then OR)

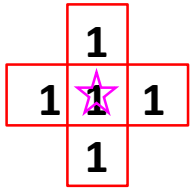


0

1

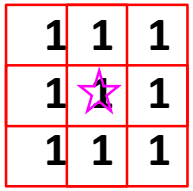
kernel

4-connected



☆: anchor

8-connected

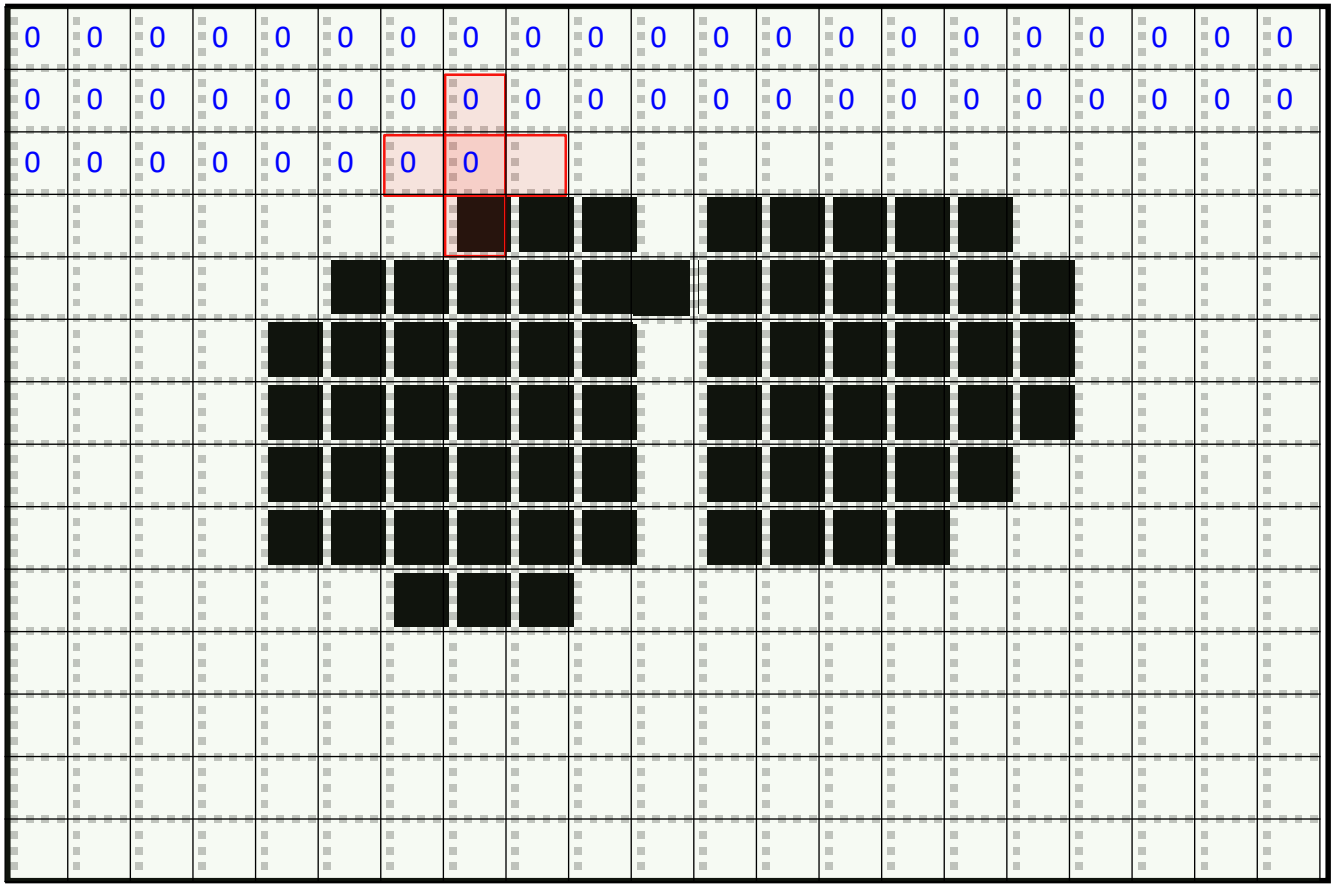


Cannot fill the hole

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

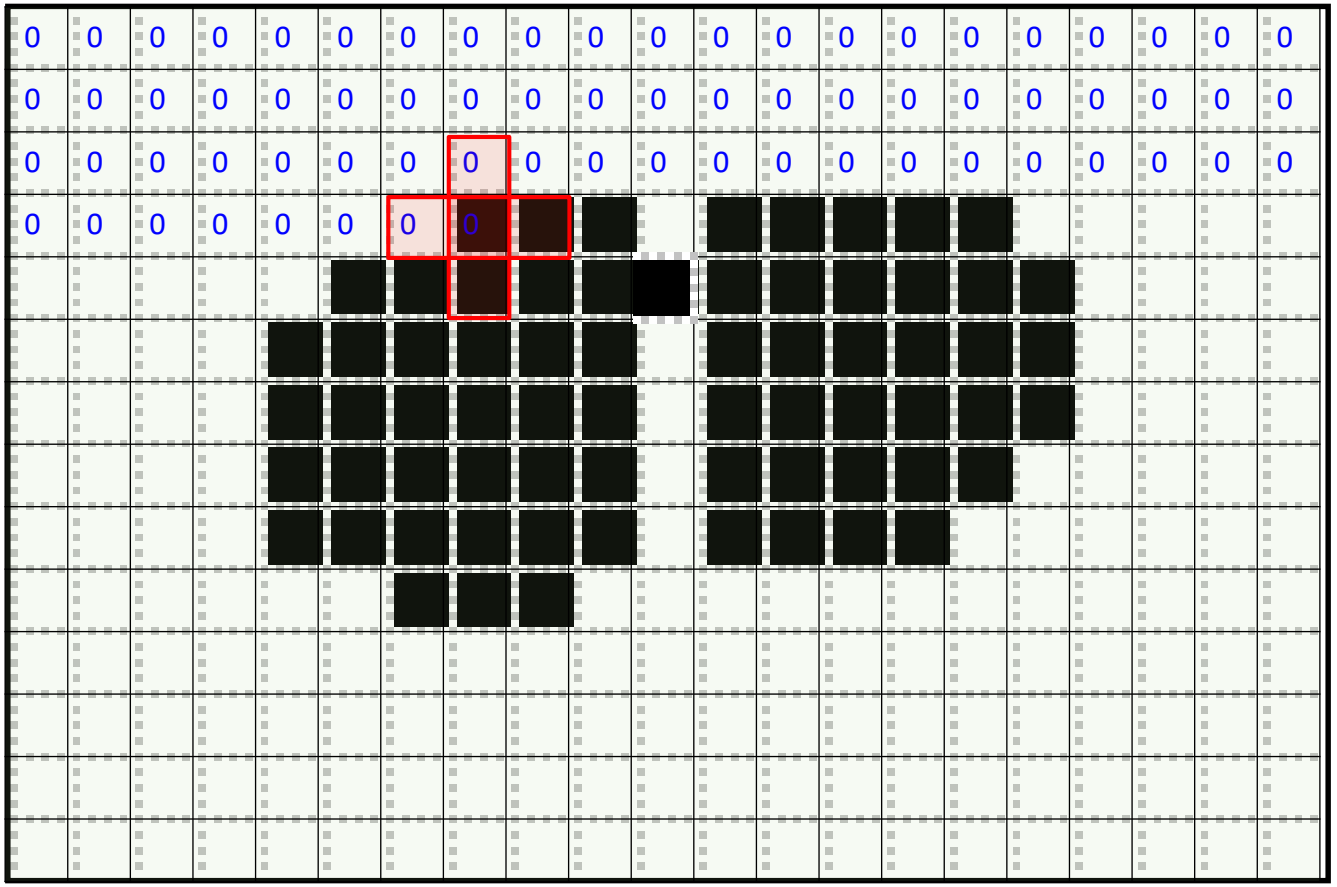
8-connected

1 1 1
1 1 1
1 1 1

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

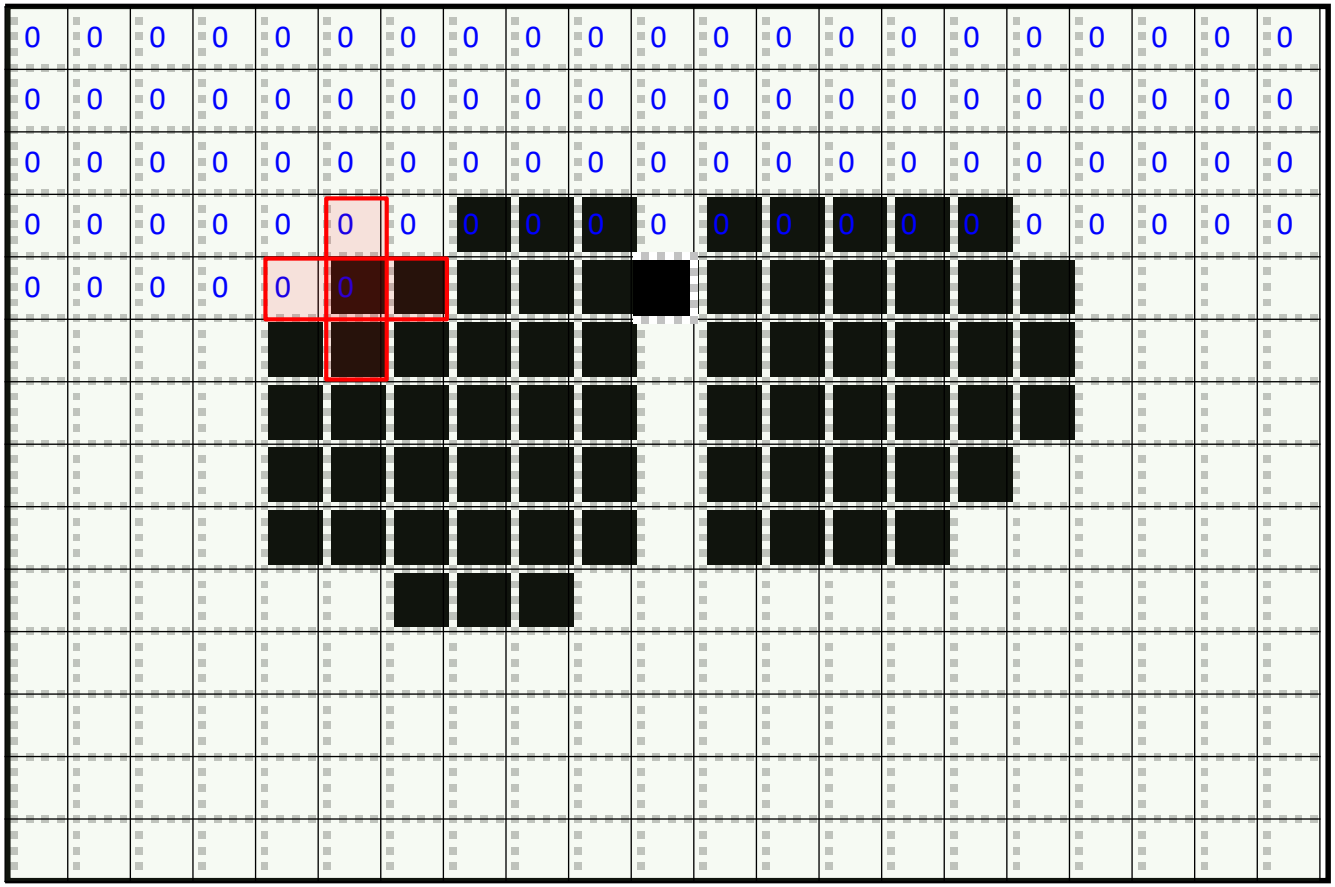
8-connected

1 1 1
1 1 1
1 1 1

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

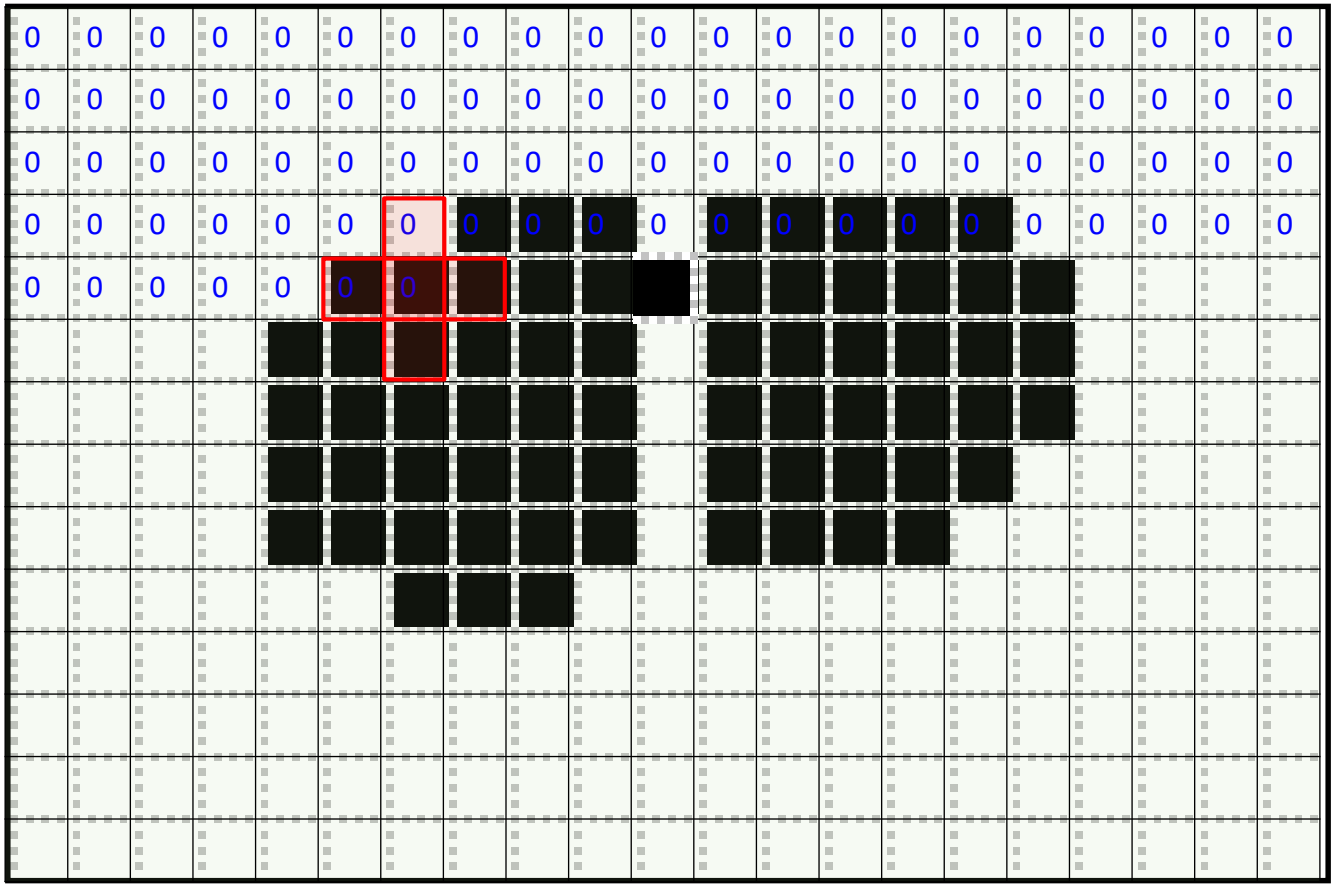
8-connected

1 1 1
1 1 1
1 1 1

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

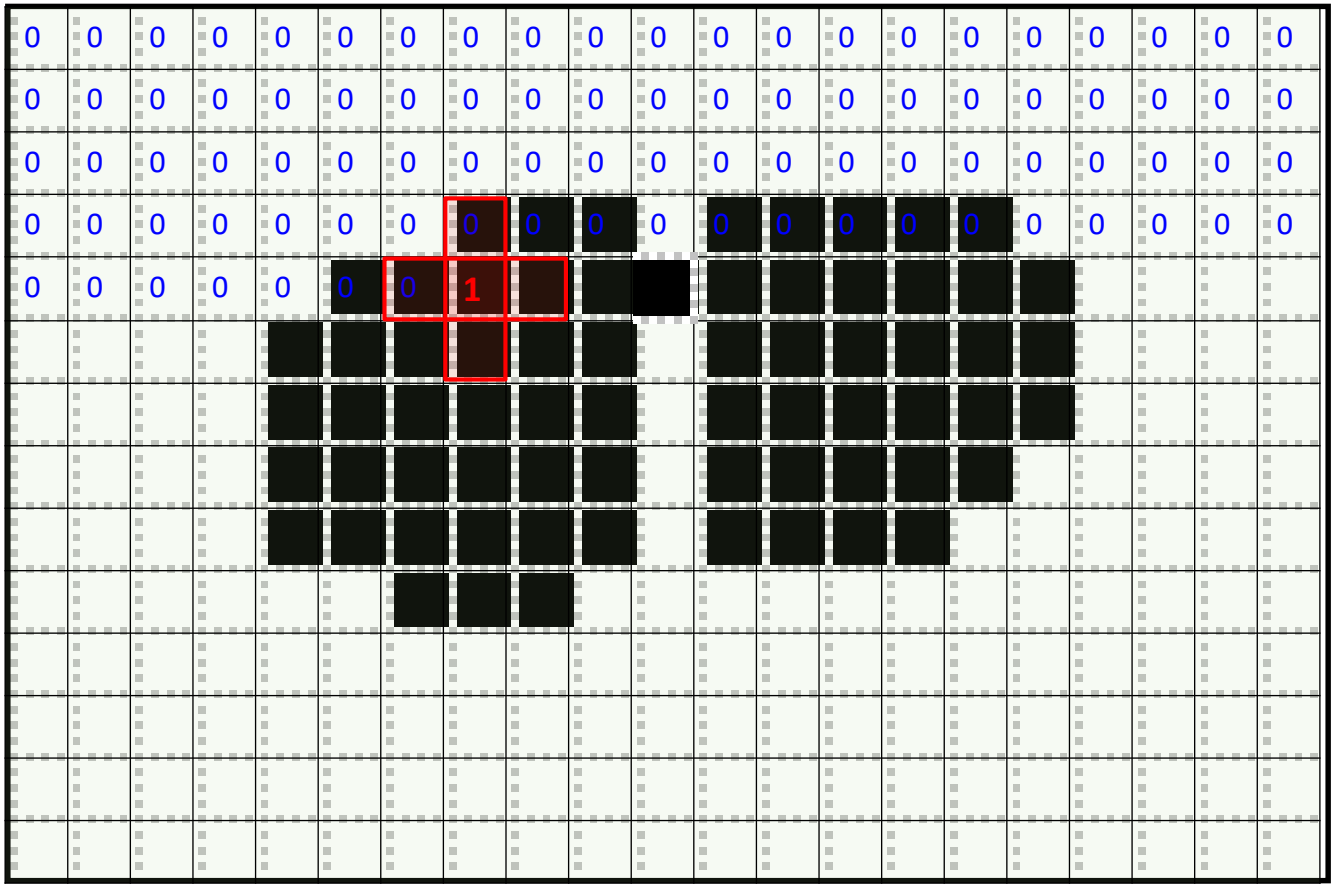
8-connected

1 1 1
1 1 1
1 1 1

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

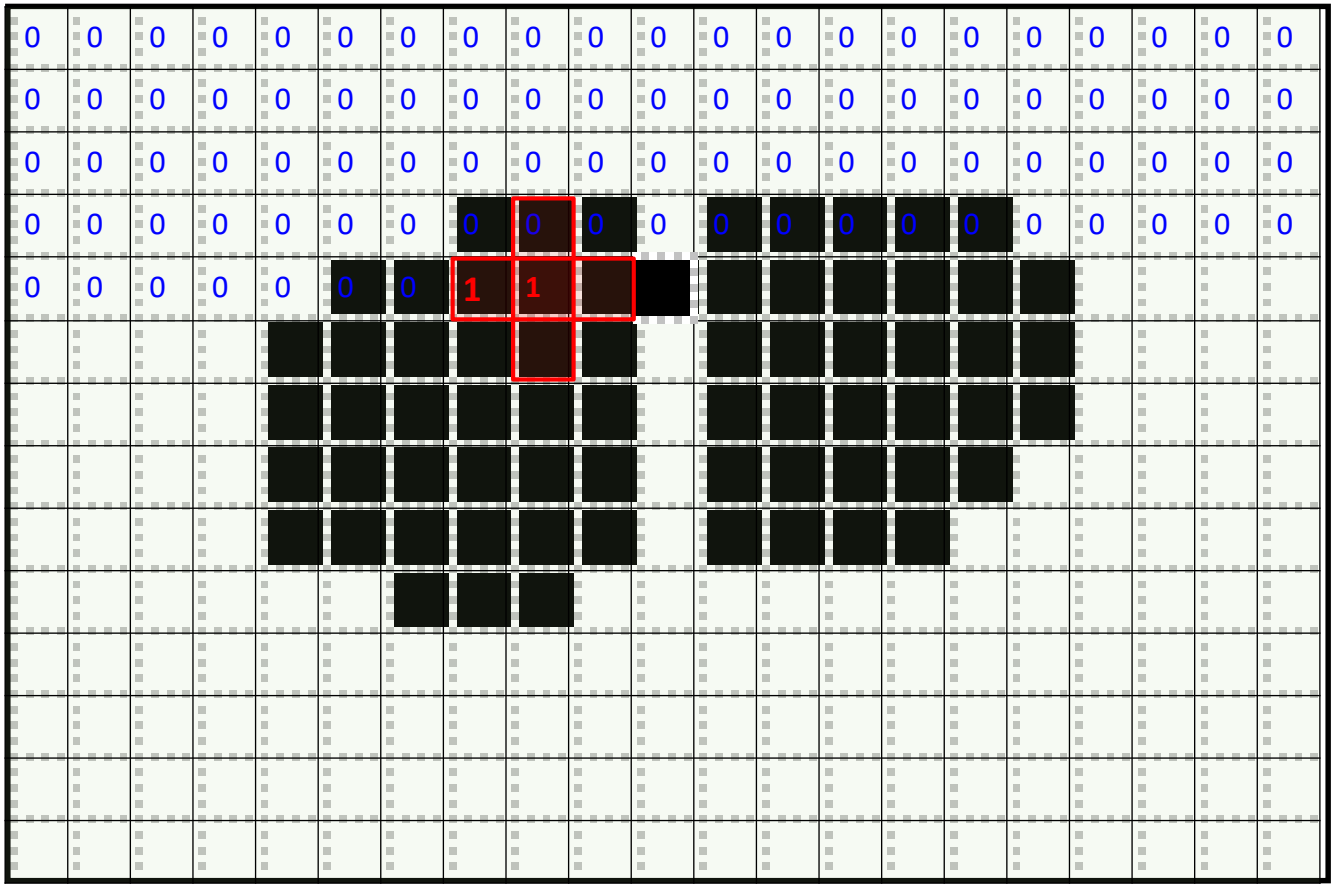
8-connected

1 1 1
1 1 1
1 1 1

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 1 1
1

☆: anchor

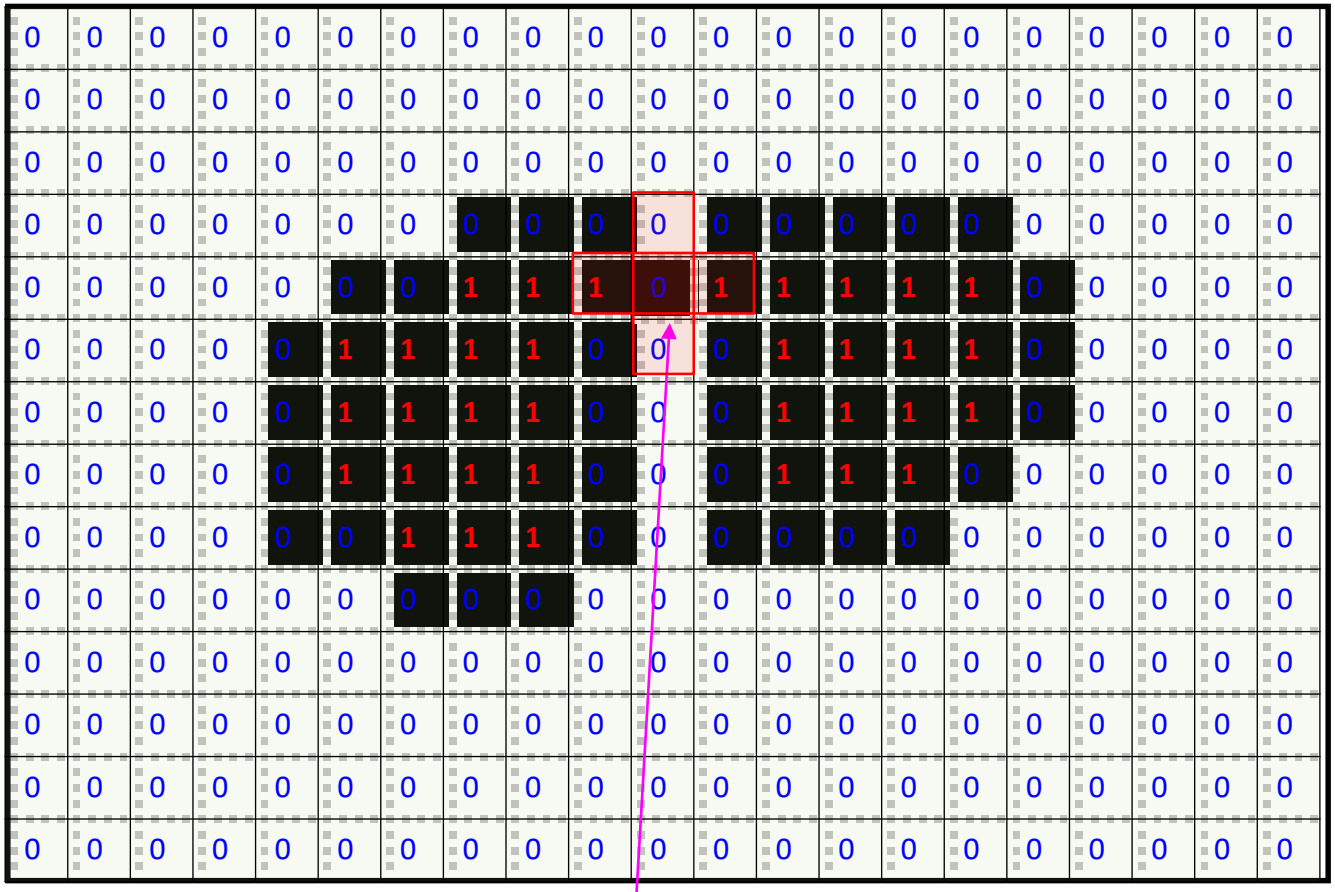
8-connected

1 1 1
1 1 1
1 1 1

Erosion $A \ominus B$

A: image, B: kernal

Take the minimum under the kernel (op: pixel-wise AND, then AND)



0

1

kernel

4-connected

1
1 ☆ 1
1

☆: anchor

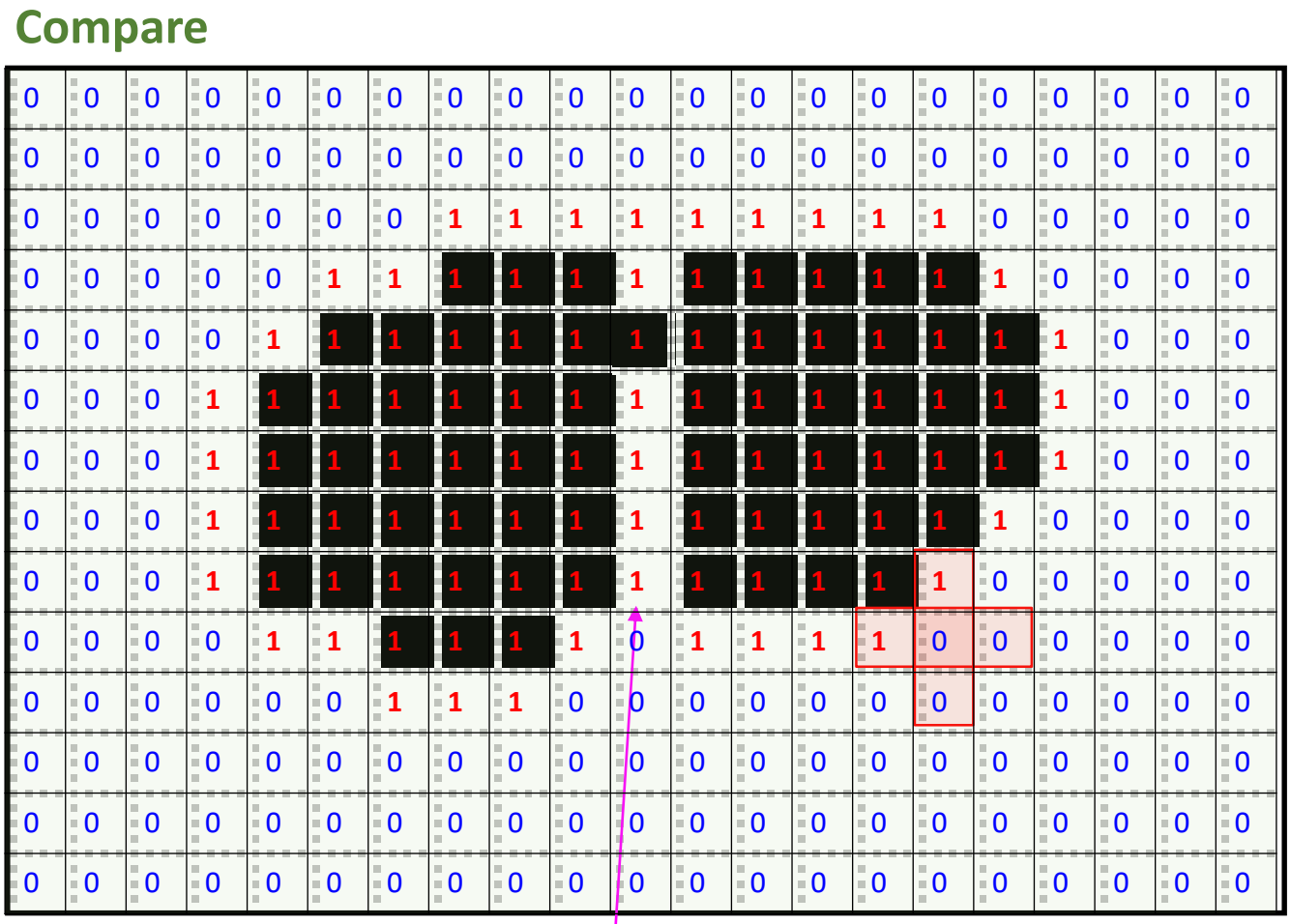
8-connected

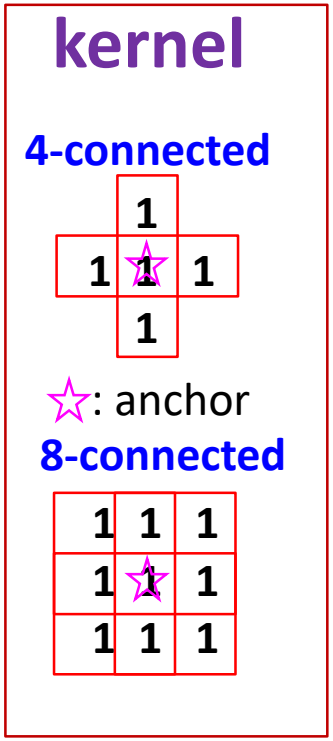
1 1 1
1 ☆ 1
1 1 1

Cut the thin connection

Compare with Dilation

Take the maximum under the kernel (op: pixel-wise AND, then OR)





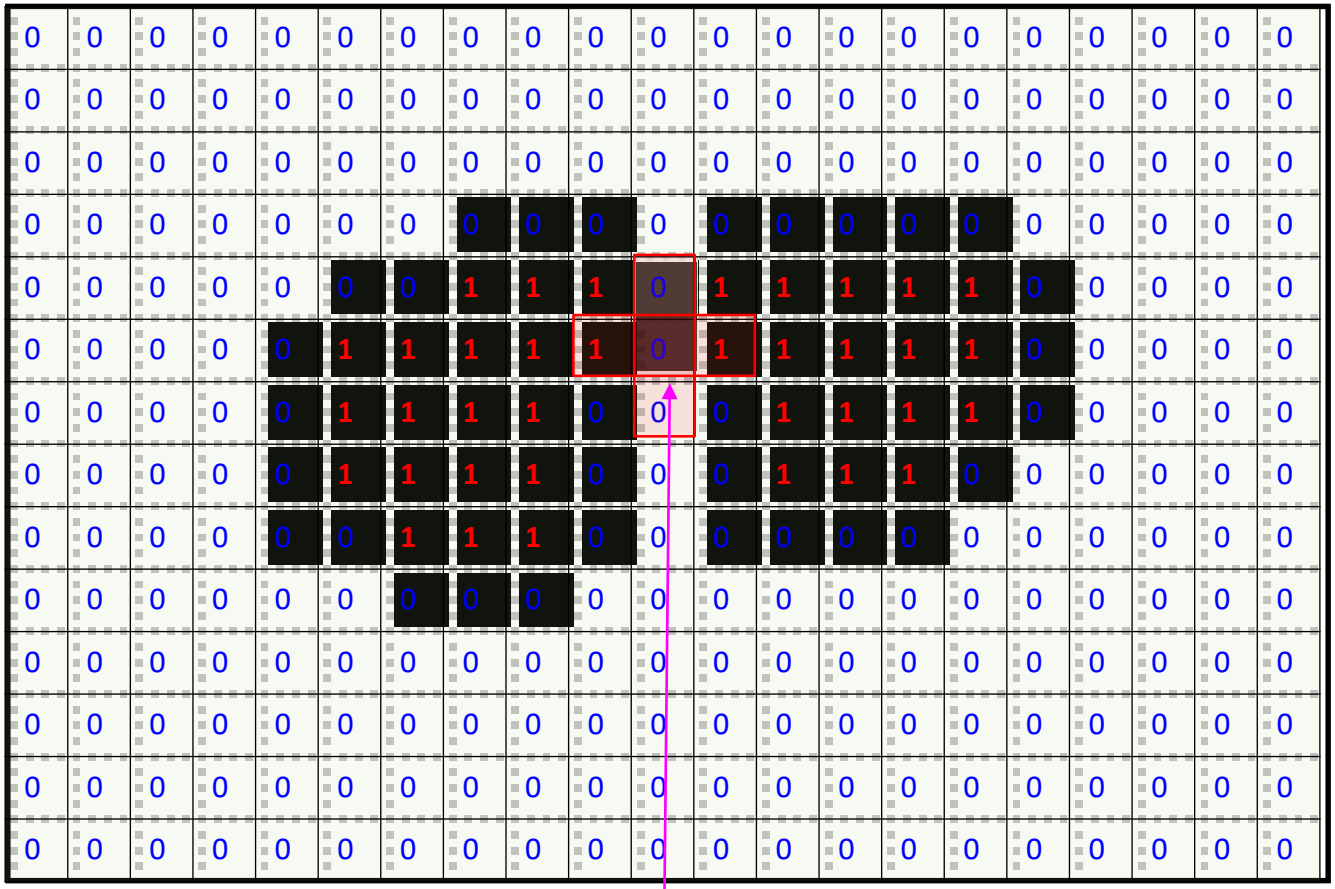
0

1

Erosion --- Another example

Take the minimum under the kernel (op: pixel-wise AND, then AND)

Another example



kernel

4-connected

1
1 1 1
1

☆: anchor

8-connected

1 1 1
1 1 1
1 1 1

0

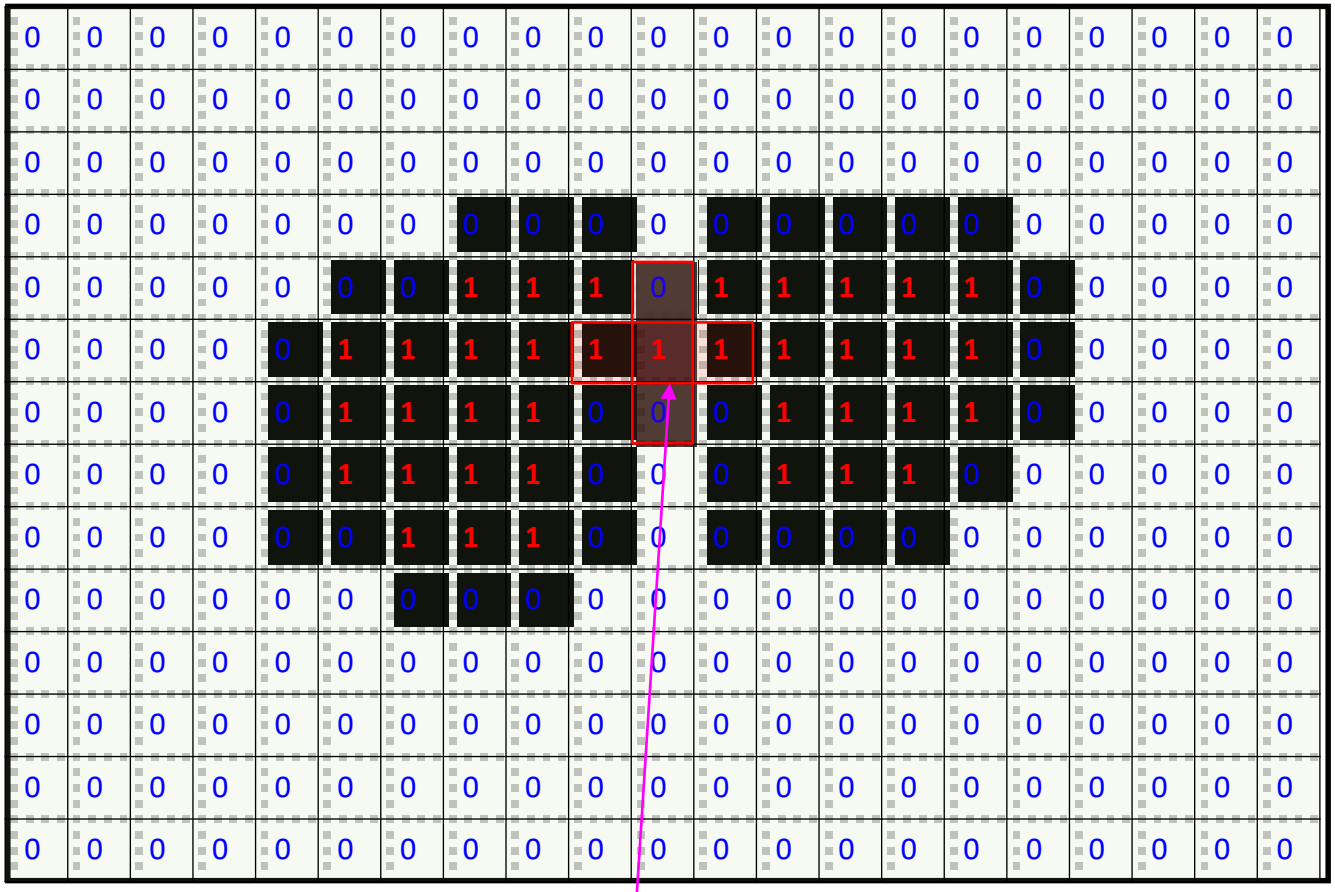
1

Cut the thin connection

Erosion --- Another example

Take the minimum under the kernel (op: pixel-wise AND, then AND)

Another example



Retained connection

0

1

kernel

4-connected

1
1 1 1
1

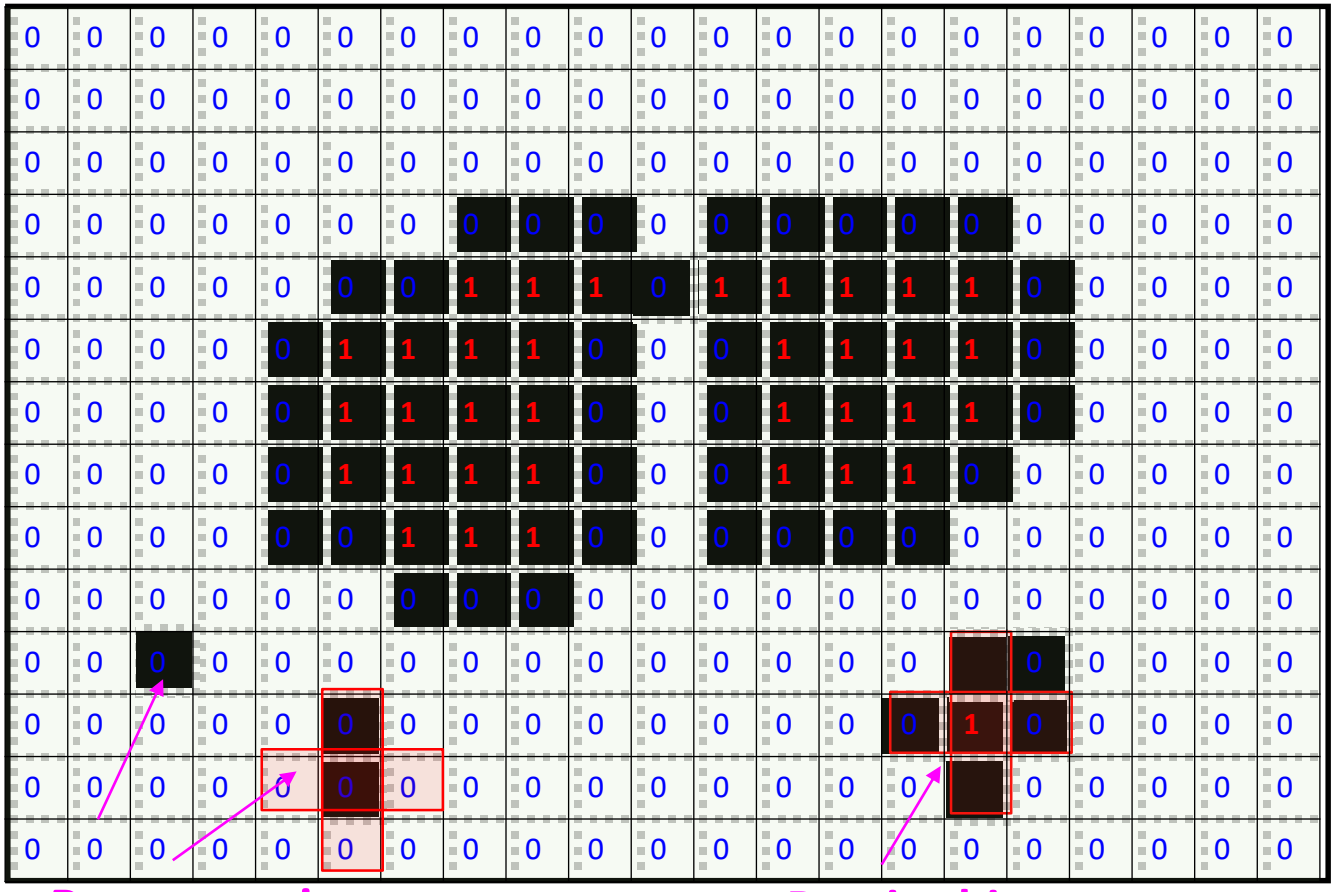
☆: anchor

8-connected

1 1 1
1 1 1
1 1 1

Erosion --- Another example

Take the minimum under the kernel (op: pixel-wise AND, then AND)



kernel

4-connected

1
1 1 1
1

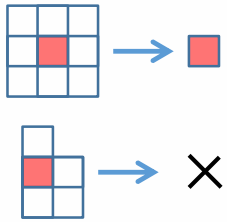
☆: anchor

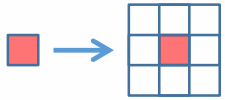
8-connected

1 1 1
1 1 1
1 1 1

0
1

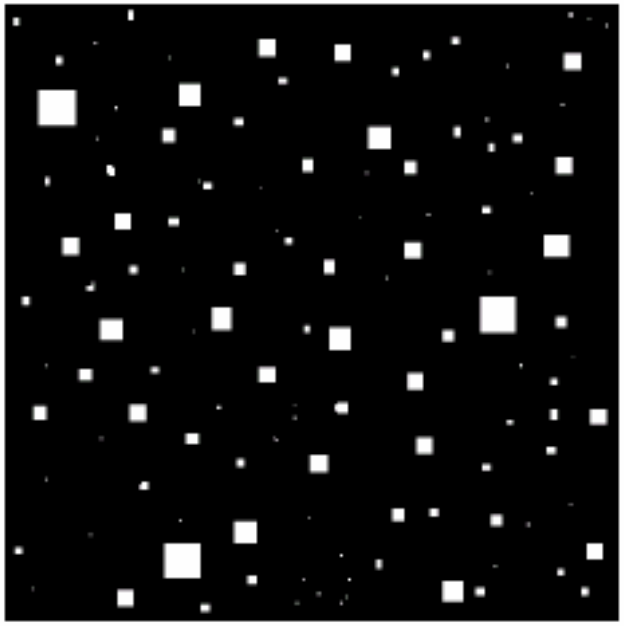
Application of erosion: eliminate irrelevant detail

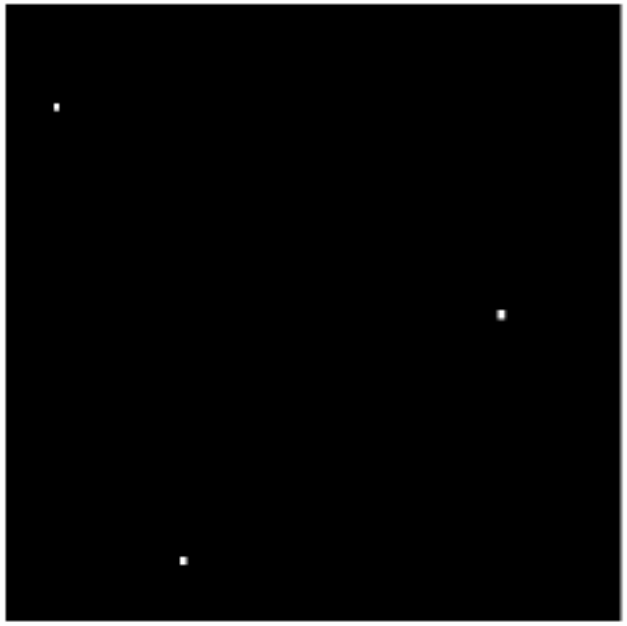


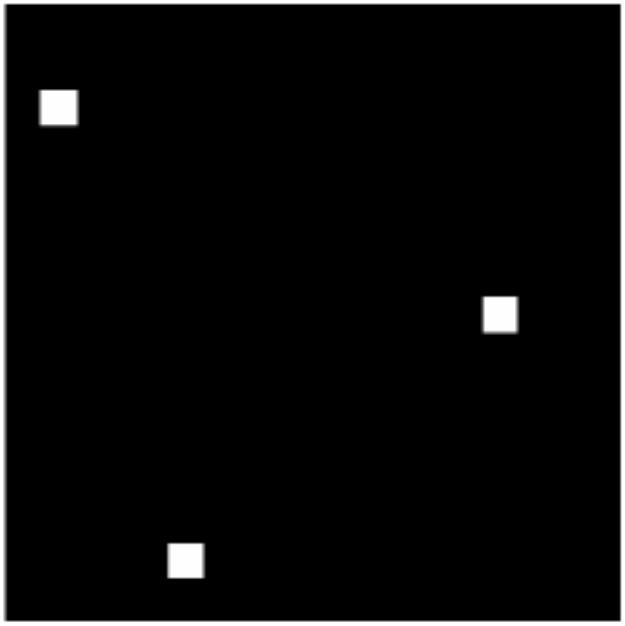


Squares of size
1,3,5,7,9,15 pels

Erode with
13x13 square







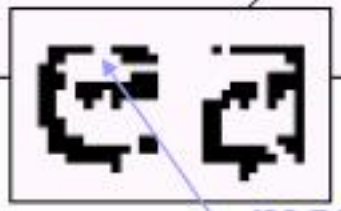
original image → erosion → dilation

Application of dilation: bridging gaps in images

0	1	0
1	1	1
0	1	0

Structuring element

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



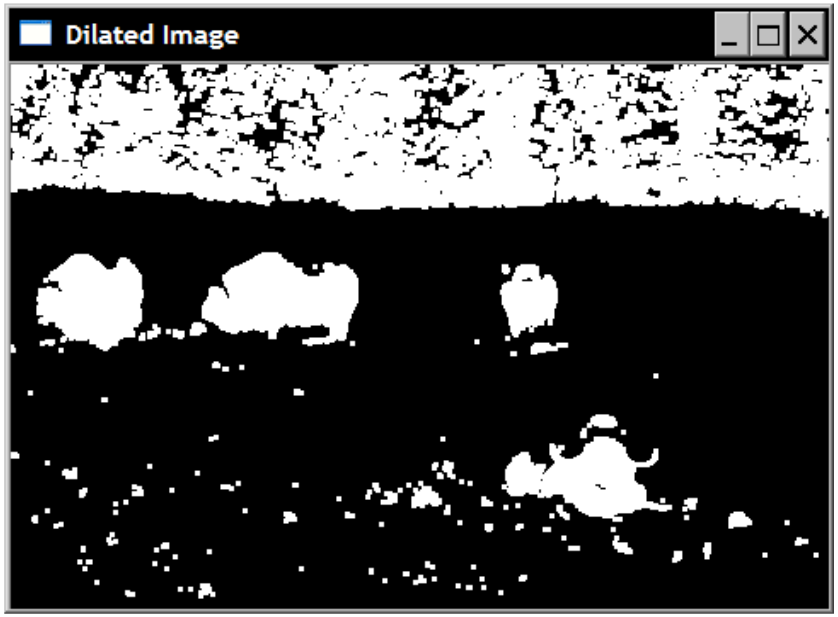
max. gap=2 pixels

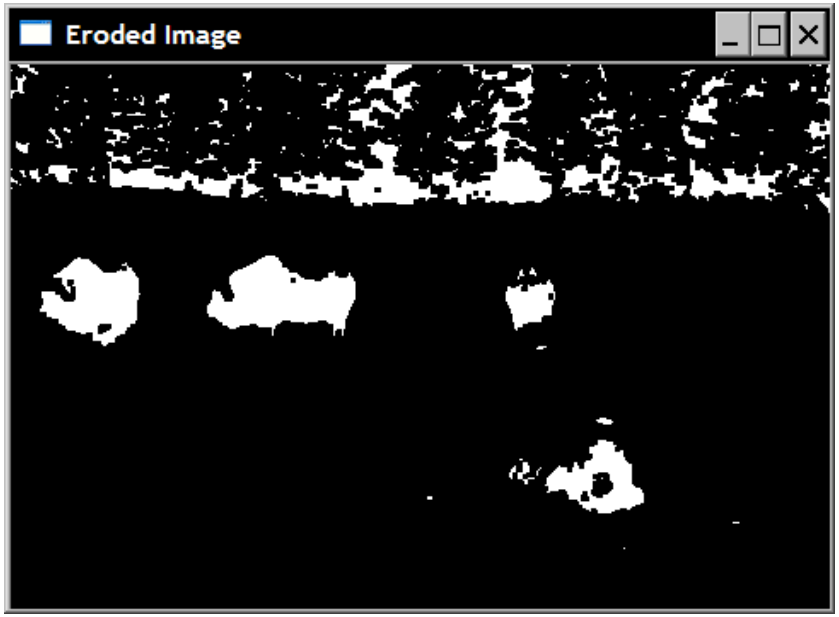
Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

Example – binary image

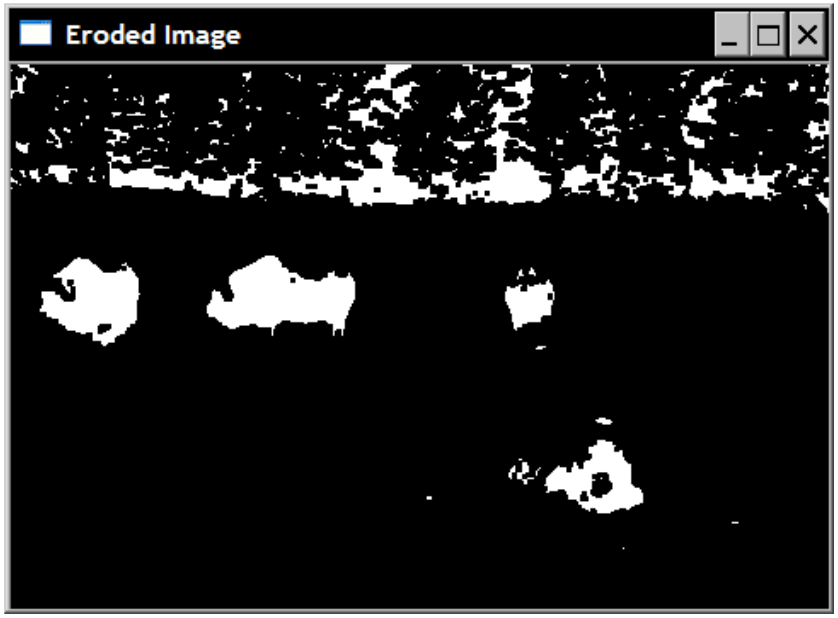
1	1	1
1	1	1
1	1	1

thresholding



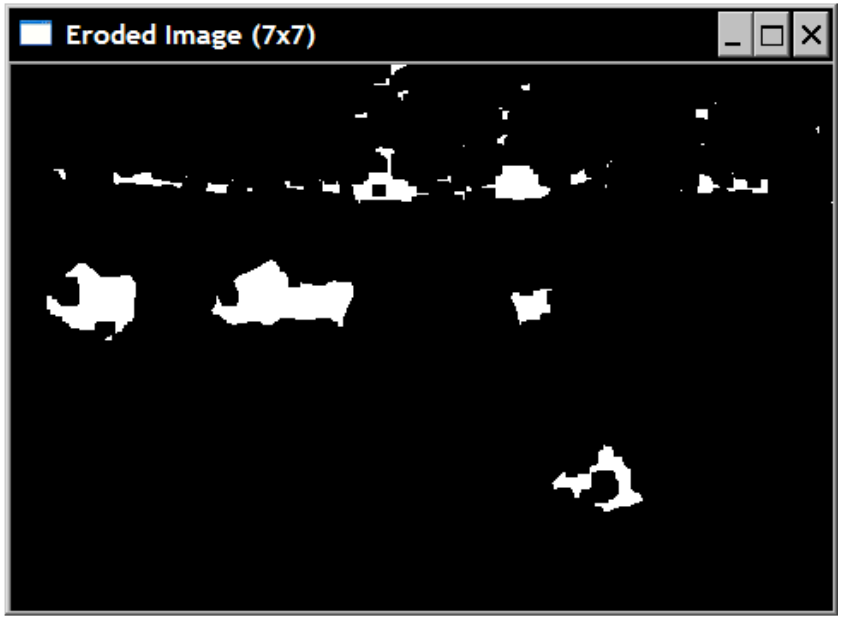


Example – binary image



With a 3 x 3 kernel (default)

1	1	1
1	1	1
1	1	1



With a 7 x 7 kernel

OpenCV Function -- dilate

- dilate(InputArray **src**,

  OutputArray **dst**,

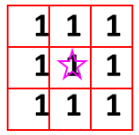
  InputArray **kernel**,

  Point **anchor**=Point(-1,-1),

  int **iterations**=1,

  int **borderType**=BORDER_CONSTANT,

  const Scalar& **borderValue** =
  morphologyDefaultBorderValue())

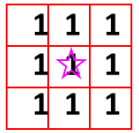


- dilate(src, dst, Mat(), Point(-1,-1), 1);

OpenCV Function -- dilate

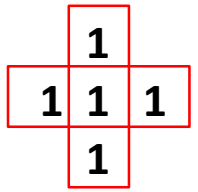
- dilate(src, dst, Mat(), Point(-1,-1), 1);

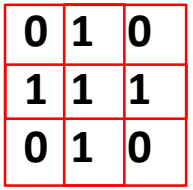
Kernel (structuring element)

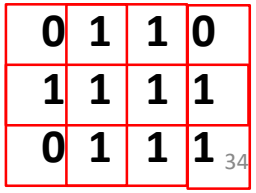


- By default, OpenCV uses a 3x3 square structuring element.
- This default structuring element is obtained when an empty matrix (that is cv::Mat()) is specified as the third argument in the function call.
- You can also specify a structuring element of the size and shape you want by providing a matrix in which the non-zero element defines the structuring element.
- For example, a 7x7 structuring element is applied:
  - Mat element(7,7,CV_8U,cv::Scalar(1));
  - dilate(image, dilated, element);

4-connected





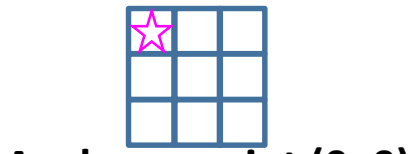


OpenCV Function -- dilate

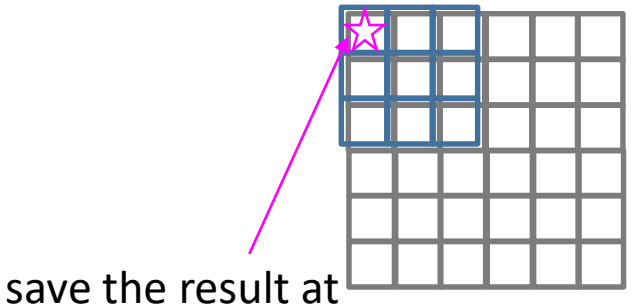
- dilate(src, dst, Mat(), Point(-1,-1), 1);

anchor

- The origin argument Point(-1,-1) means that the origin is at the center of the matrix (default).
- It can be defined anywhere on the structuring element.



Anchor = point (0, 0)



save the result at the achor point

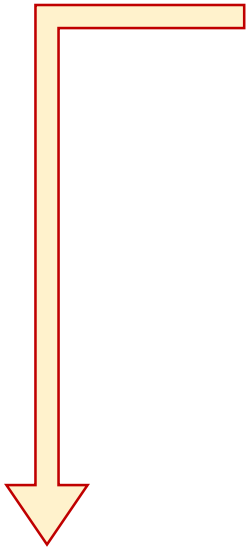
OpenCV Function -- erode

- erode(InputArray **src**,
  OutputArray **dst**,
  InputArray **kernel**,
  Point **anchor**=Point(-1,-1),
  int **iterations**=1,
  int **borderType**=BORDER_CONSTANT,
  const Scalar& **borderValue** =
  morphologyDefaultBorderValue())

erode(src, dst, Mat(), Point(-1,-1), 1);

OpenCV Function -- More Morphological Operations

- morphologyEx (InputArray **src**,
  OutputArray **dst**,
  Int **op,**
  InputArray **kernel**,
  Point **anchor**=Point(-1,-1),
  int **iterations**=1,
  int **borderType**=BORDER_CONSTANT,
  const Scalar& **borderValue** =
  morphologyDefaultBorderValue())



MORPH_OPEN - an opening operation
MORPH_CLOSE - a closing operation
MORPH_GRADIENT - a morphological gradient
MORPH_TOPHAT - "top hat"
MORPH_BLACKHAT - "black hat"

Try it !

Opening and Closing

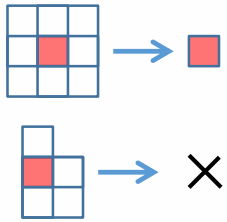
- Opening: Erosion then Dilation

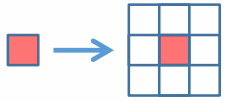
➔ $A \circ B = (A \ominus B) \oplus B$

- Closing: Dilation then Erosion

➔ $A \circ B = (A \oplus B) \ominus B$

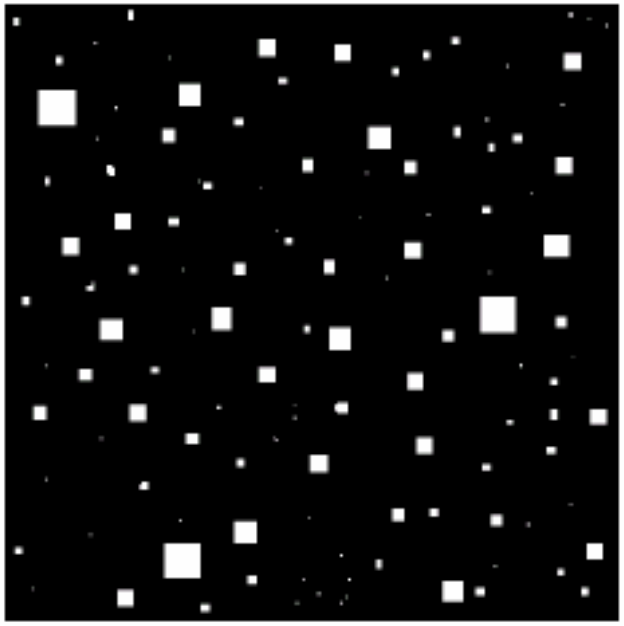
Application of erosion: eliminate irrelevant detail

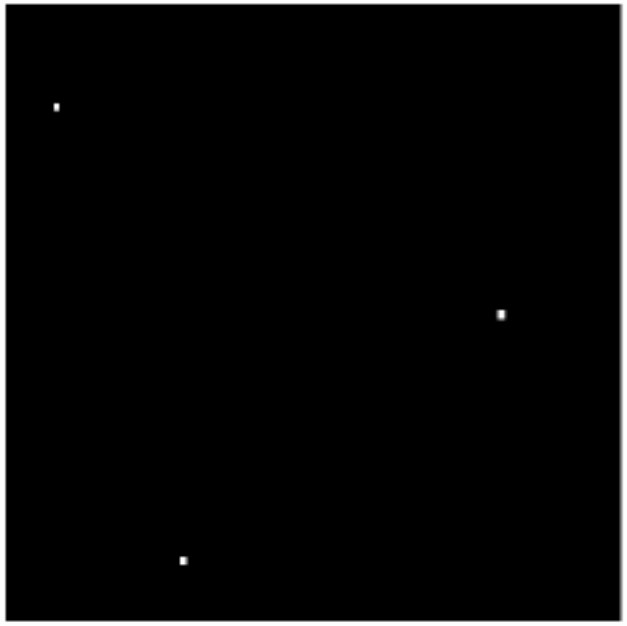


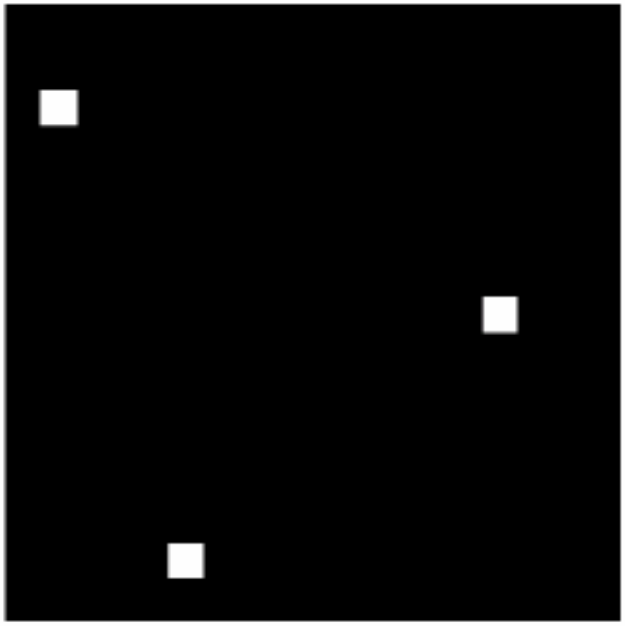


Squares of size
1,3,5,7,9,15 pels

Erode with
13x13 square



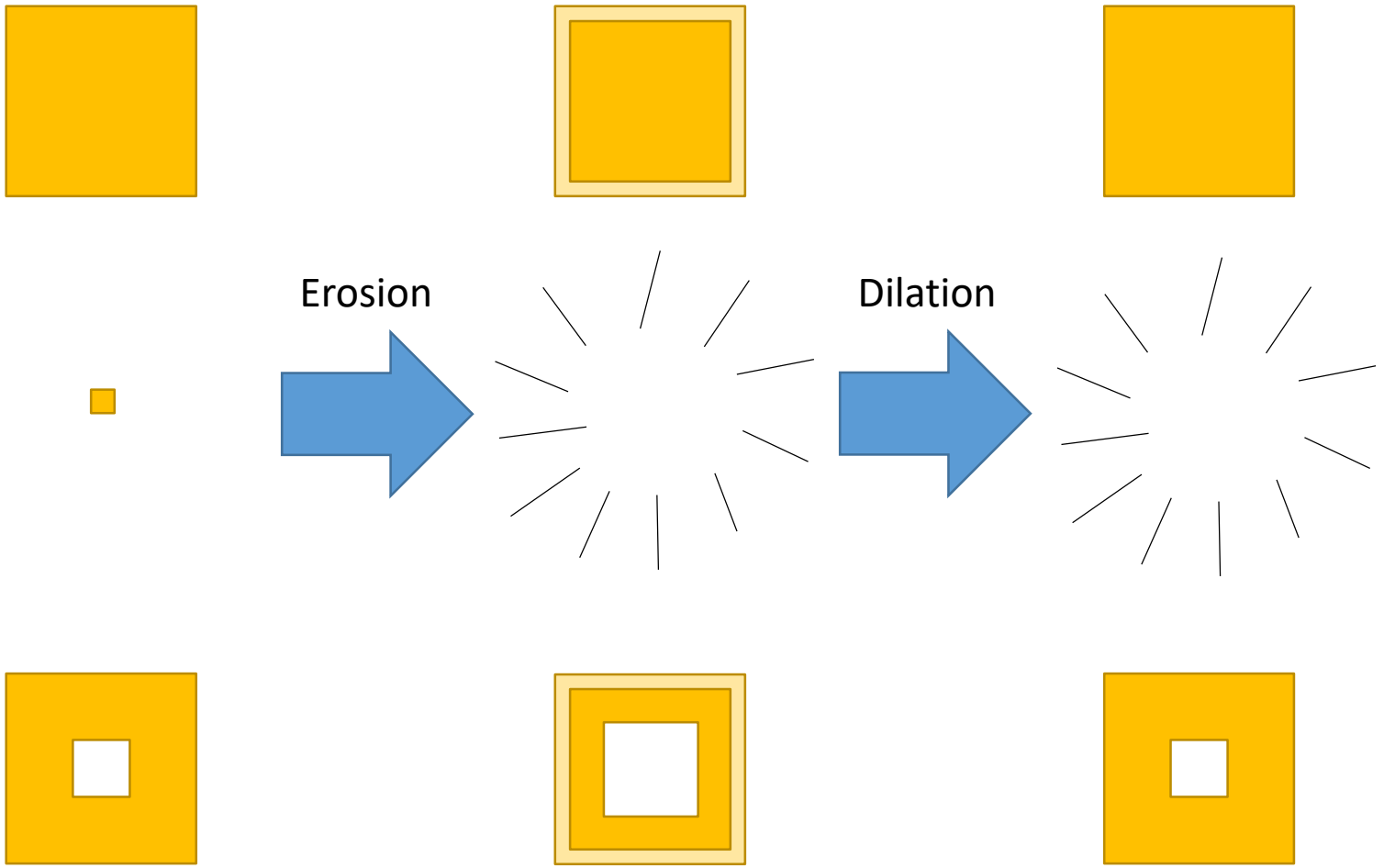




original image → erosion → dilation

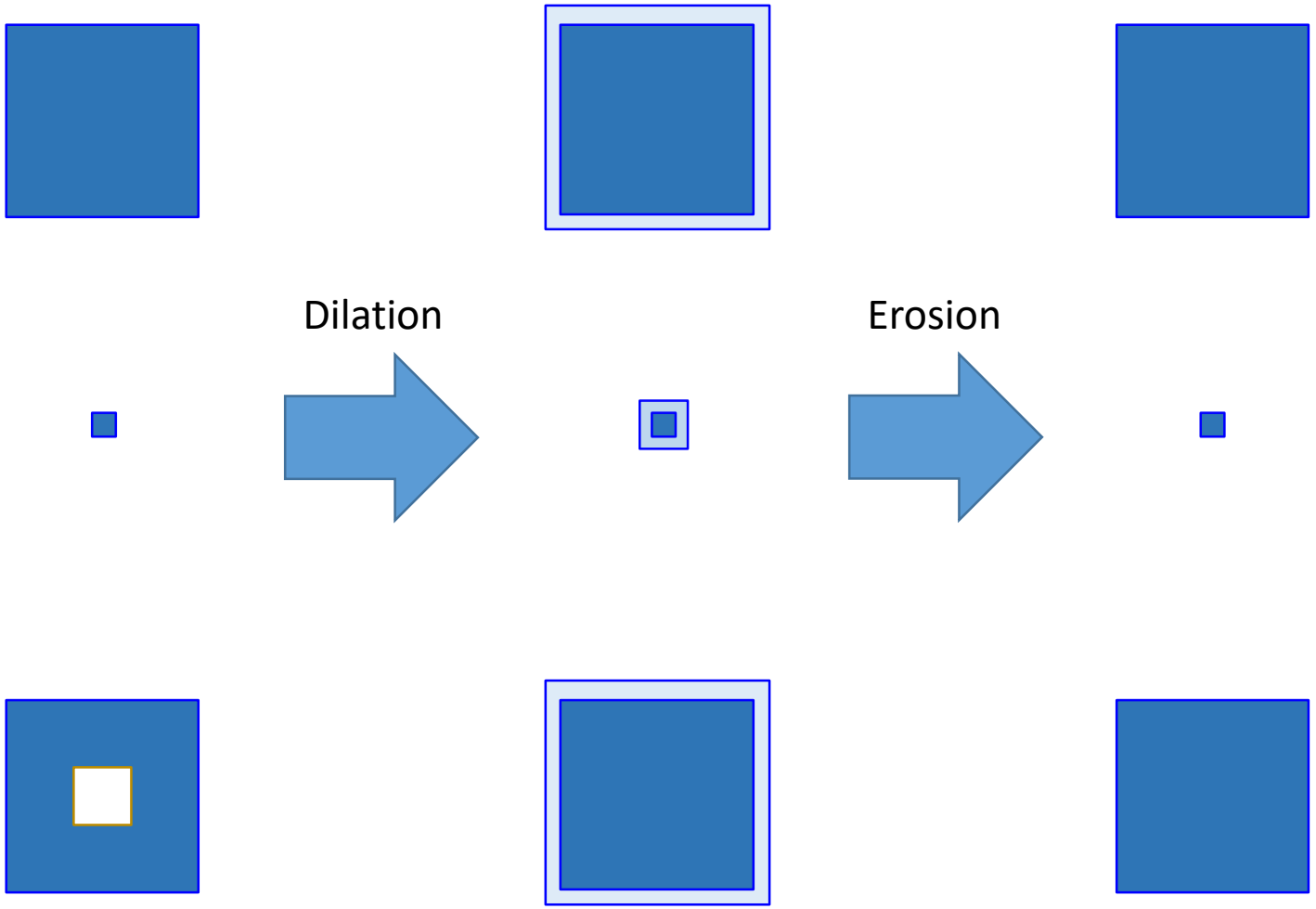
Opened Image

- Erosion then Dilation (先縮再擴)

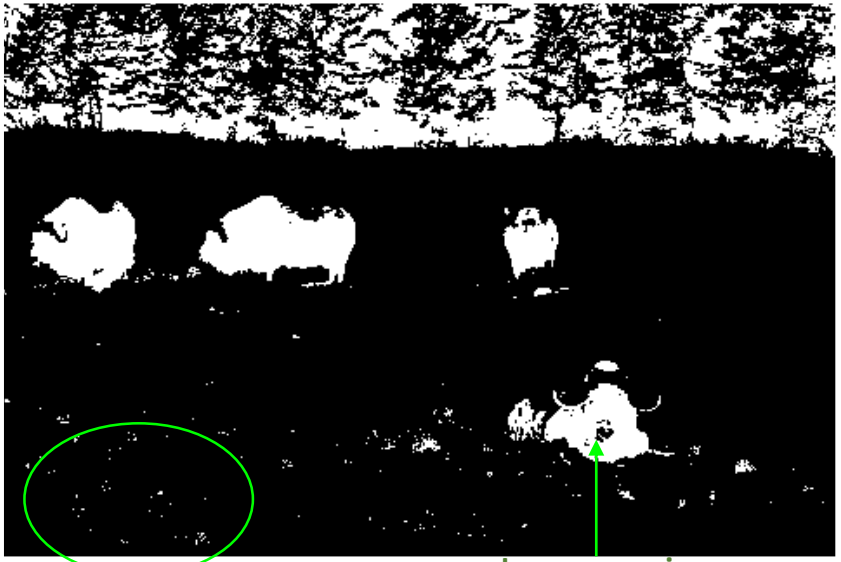


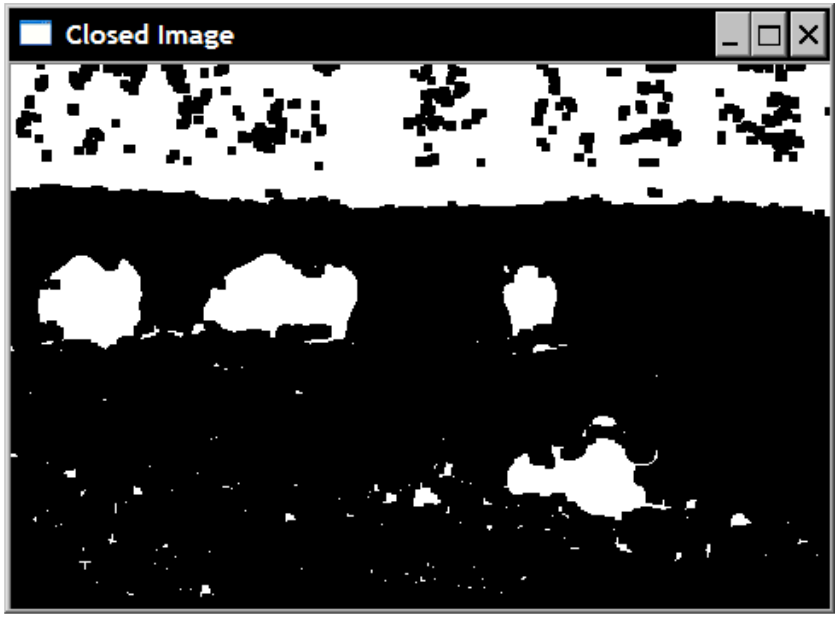
Closed Image

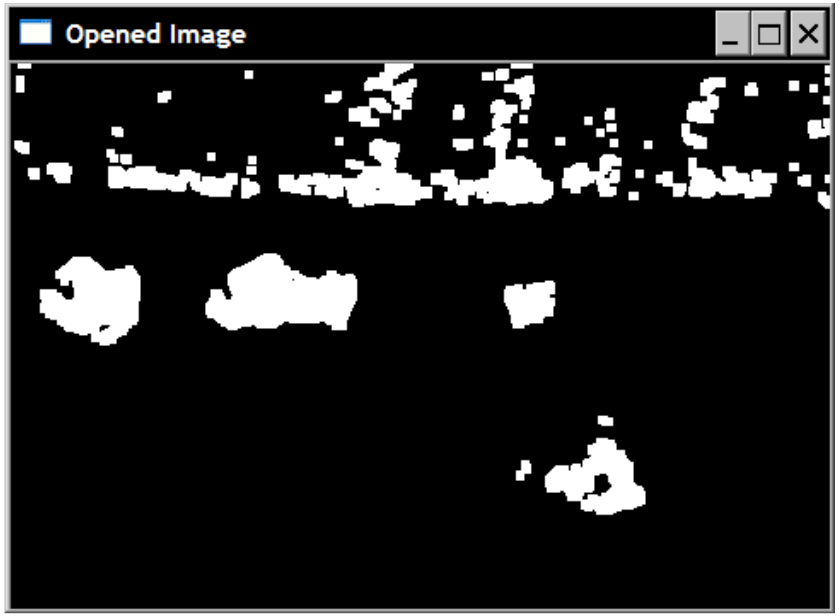
- Dilation then Erosion (先擴再縮)



Opening and Closing







Opening and Closing

Noisy image

A

$A \ominus B$

erode

			B
1	1	1	
1	1	1	
1	1	1	

(Opening) Remove outer noise

opening (erode+dilate)

dilate

$(A \ominus B) \oplus B = A \circ B$

$(A \circ B) \oplus B$

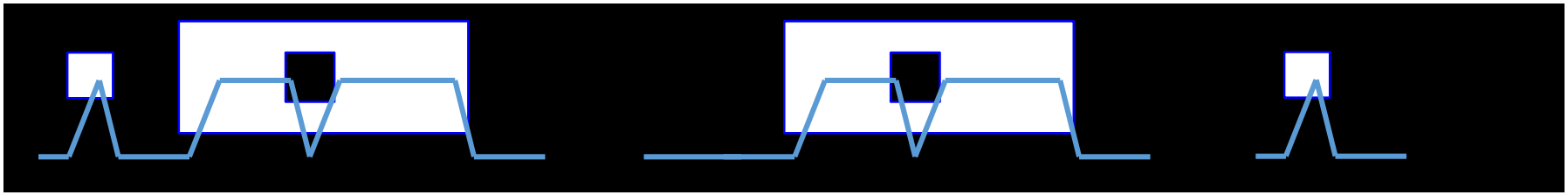
$[(A \circ B) \oplus B] \ominus B = (A \circ B) \circ B$

closing (dilate+erode)

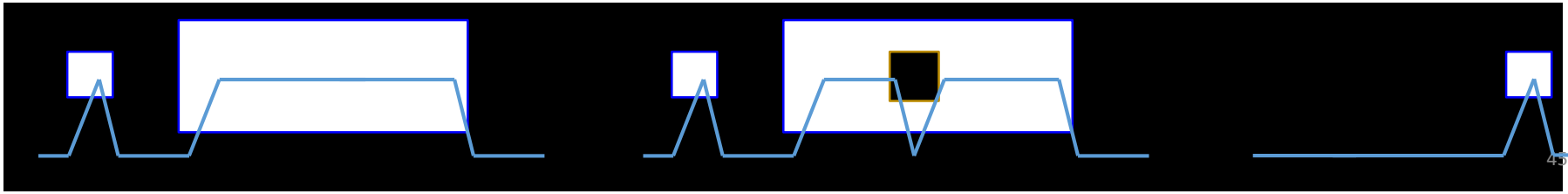
(Closing) Remove inner noise

Gradient, TopHat and BlackHat

- Gradient：Dilation – Erosion
  - Extract the edge / silhouette
- TopHat：Original – Opened Image
  - Extract the light region



- BlackHat：Closed Image – Original
  - Extract the dark region



Morphological Image Processing

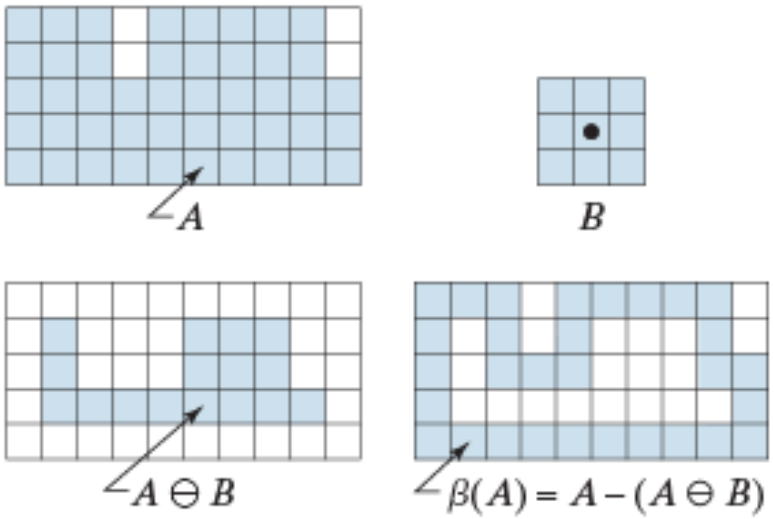
- 侵蝕 (Erode)與擴張 (Dilate)
- 斷開 (Open)與閉合 (Close)
- 邊界抽取 (Boundary Extraction)
- 填洞 (Hole Filling)
- 交離轉換(Hit-or-Miss Transformation)
- 連通成份分析(Connected Component Analysis/ Labeling)

Boundary Extraction

- β(A) ：The boundary of set A:

$$\beta(A) = A - (A \ominus B)$$

B: structuring element for boundary extraction



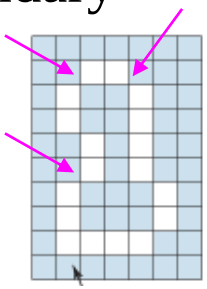
example

Hole Filling

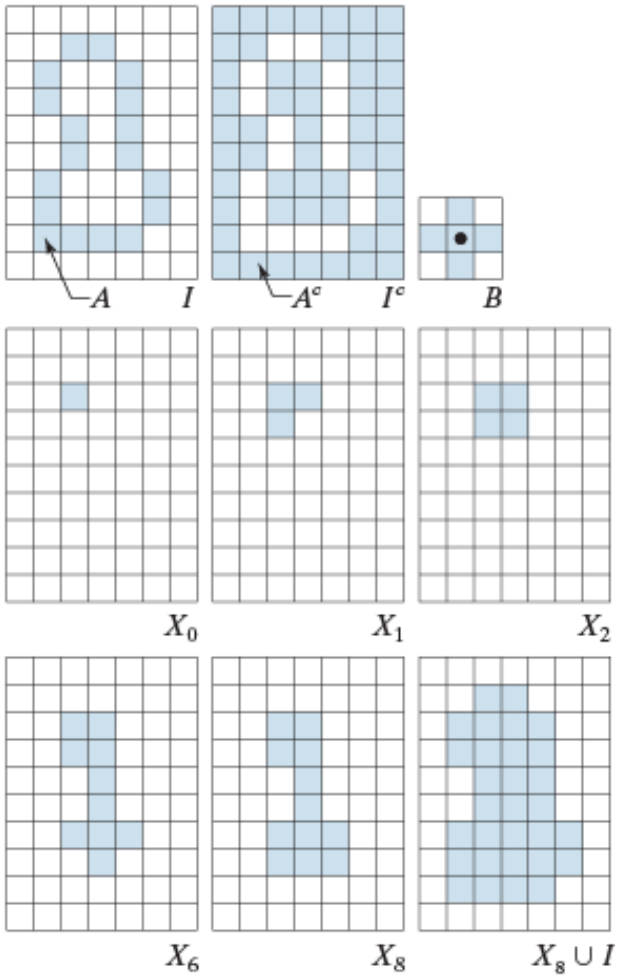
- Hole：A background region surrounded by connected foreground pixels
- A: a set of 8-connected pixels
- X_0: Initial：starting with only one hole pixel=1

$$X_k = (X_{k-1} \oplus B) \cap A^c$$

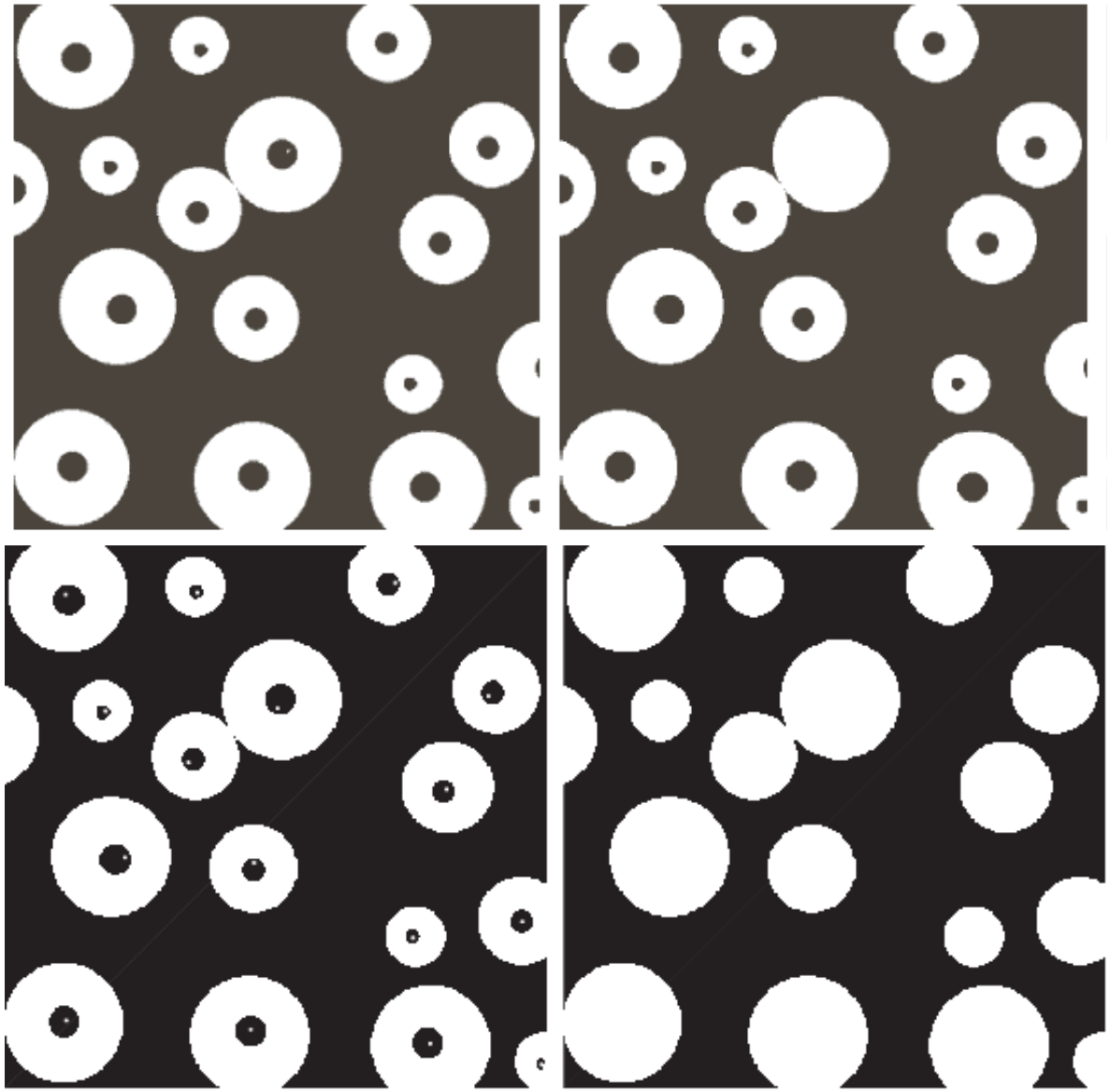
- *B:* symmetric structure element
- Interactively until $X_k = X_{k\text{-}1}$
- $\cap A^c$ limit the dilation inside the boundary
- Warning：If B is 8-connected…



Intersection: $f_{A\cap B}(x, y) = \min(f_A(x, y),\ f_B(x, y))$



Hole Filling --- example



Hit-or-Miss Transform (HMT)

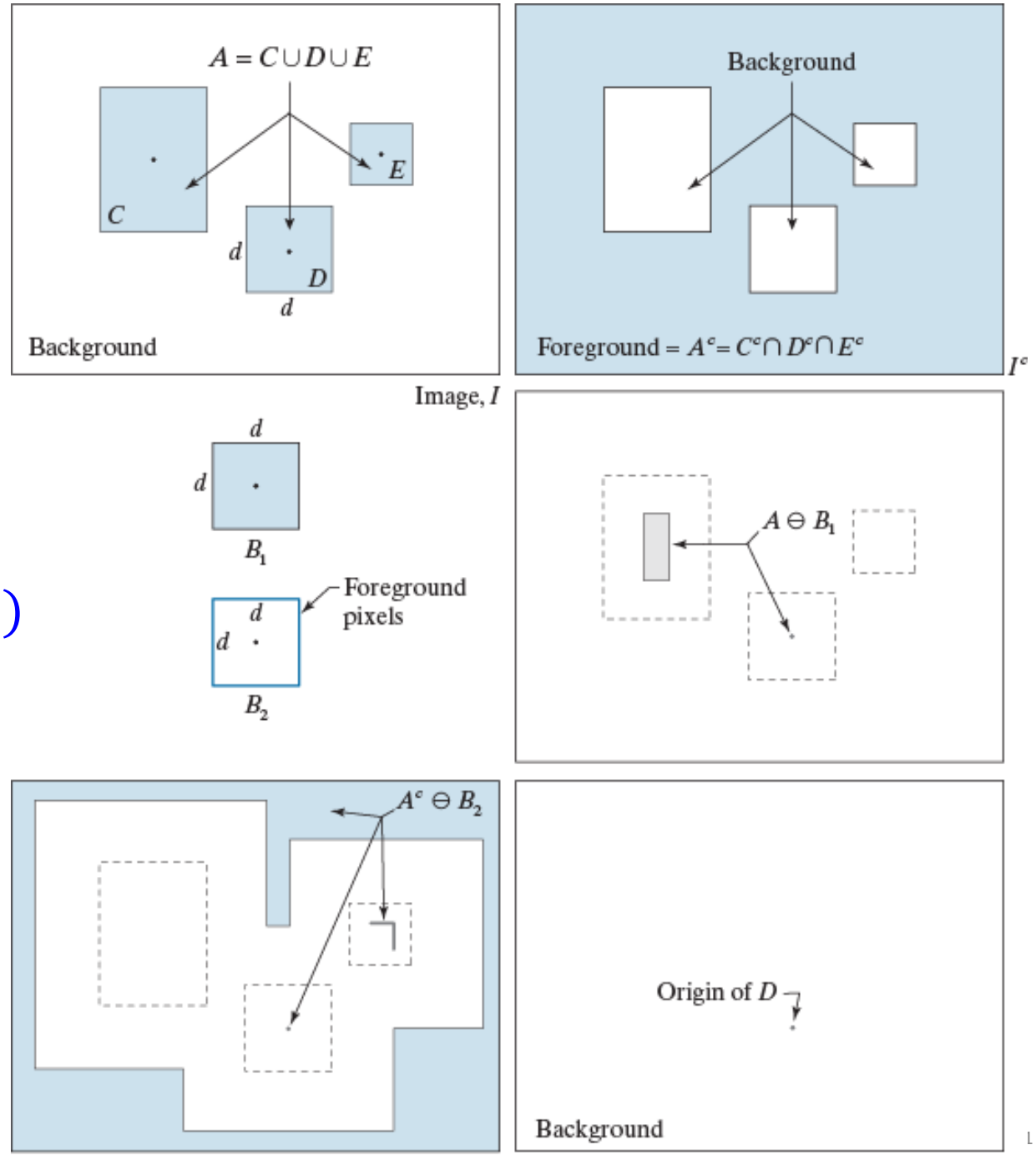
- Hit-or-Miss Transform
- Detect specific shape
- Two structuring elements
  - B_1: detecting shapes in the foreground
  - B_2: detecting shapes in the background

$$I \circledast B_{1,2} = (A \ominus B_1) \cap (A^c \ominus B_2)$$

Foreground background

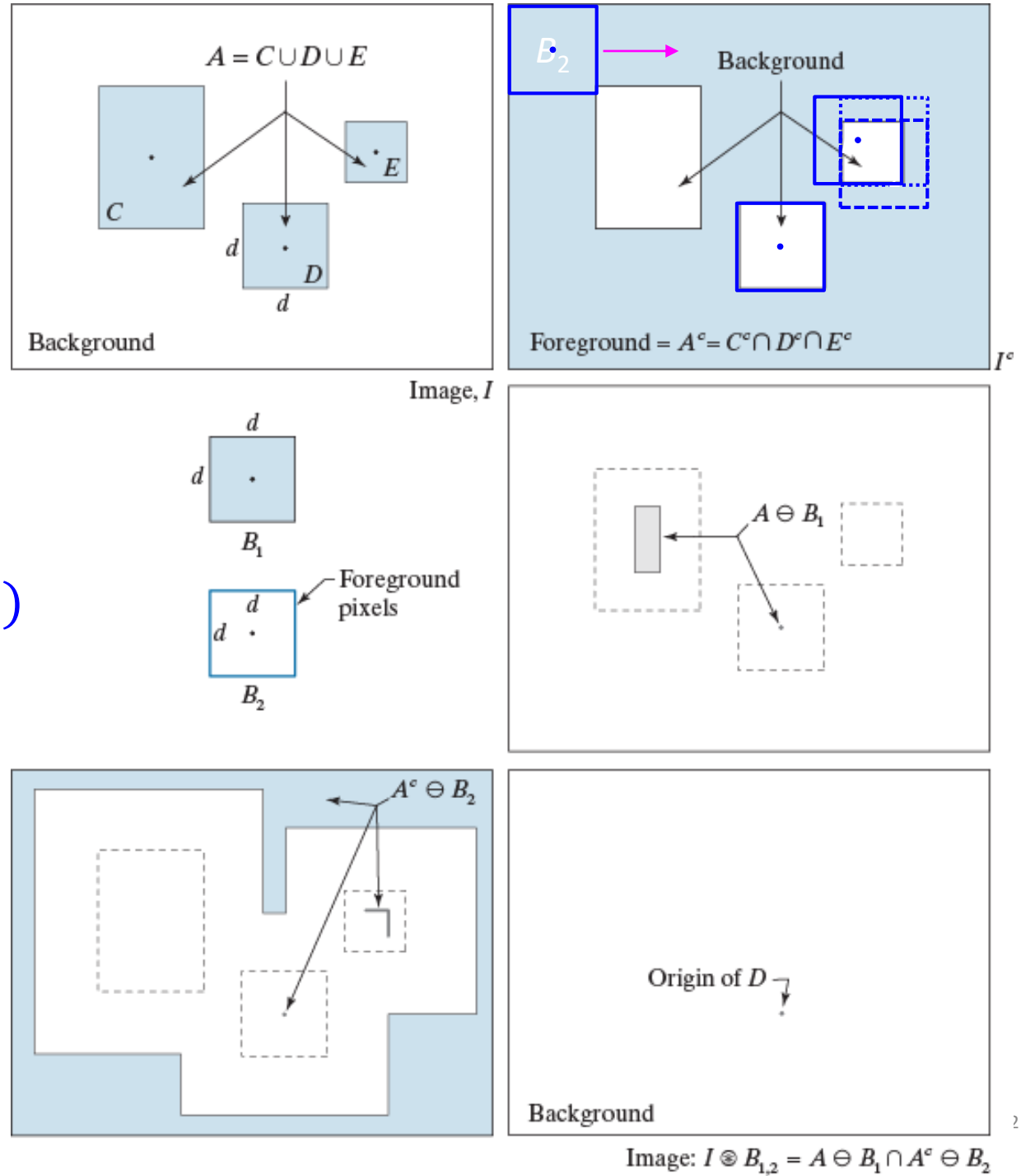
HMT

$$I \circledast B_{1,2} = (A \ominus B_1) \cap (A^c \ominus B_2)$$



HMT

$$I \circledast B_{1,2} = (A \ominus B_1) \cap (A^c \ominus B_2)$$



Connected component labeling
or
Connected component analysis

Connect Component Labeling

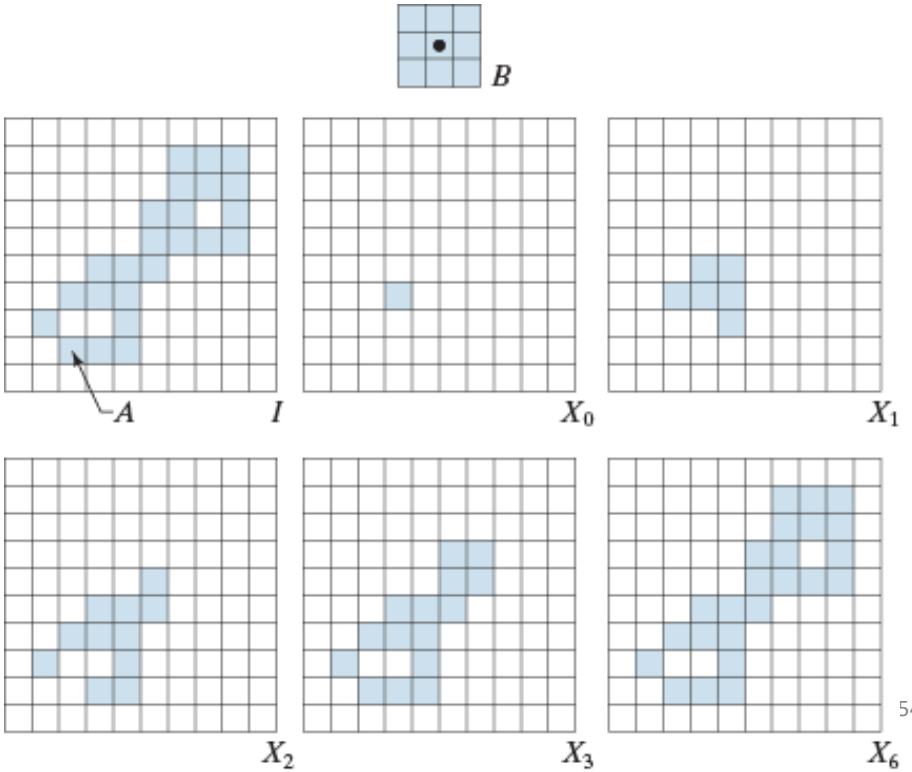
- Connected Component：A set of pixels which are connected (4-connected or 8-connected)
- A is a set of pixels containing a connected component
- X_0: Initial: only one foreground pixel is 1。

$$X_k = (X_{k-1} \oplus B) \cap A$$

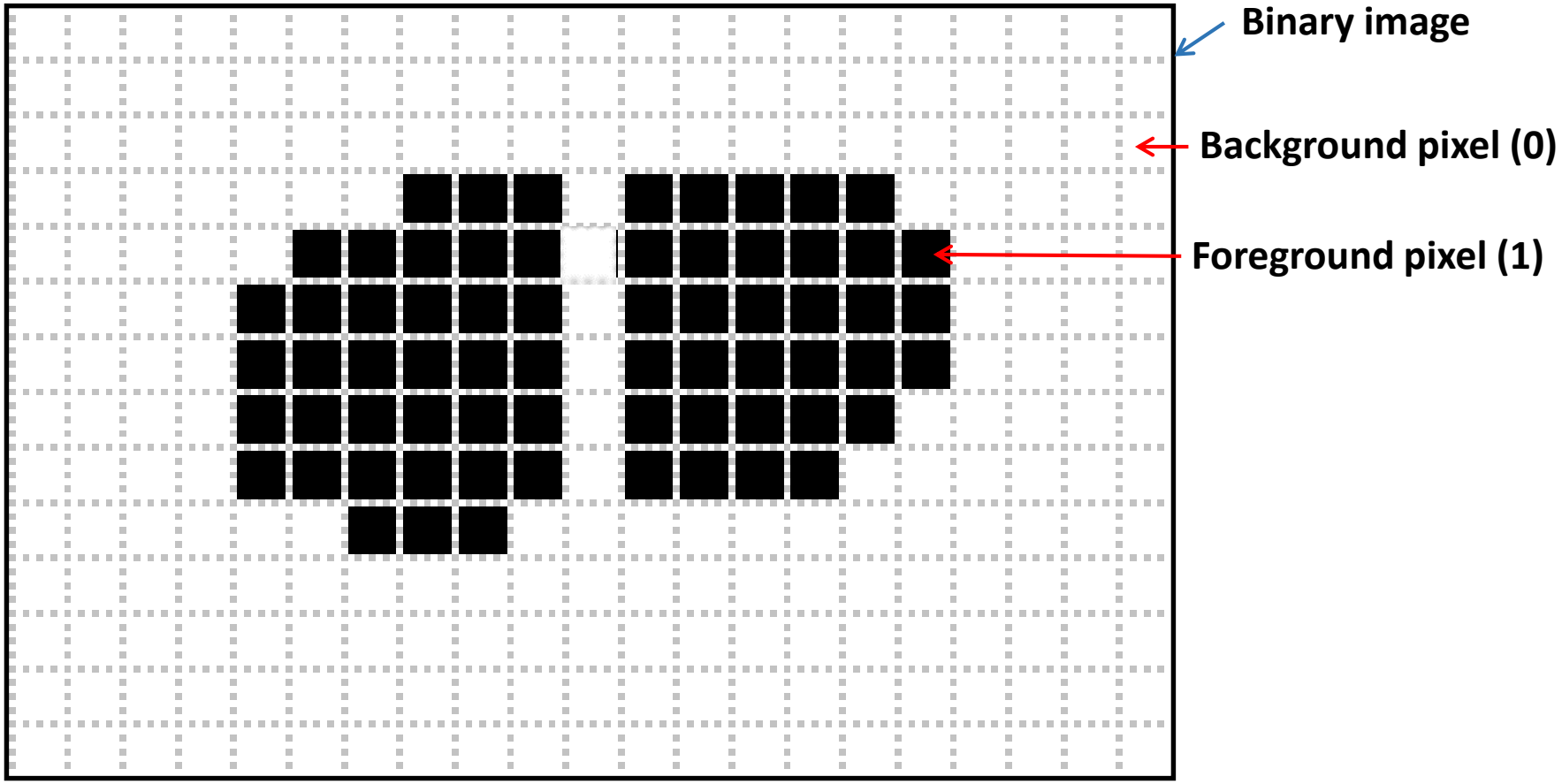
- B is symmetric S.E
- Iteratively until $X_k = X_{k\text{-}1}$
- Recall: ***hole filling***

$$X_k = (X_{k-1} \oplus B) \cap A^c$$

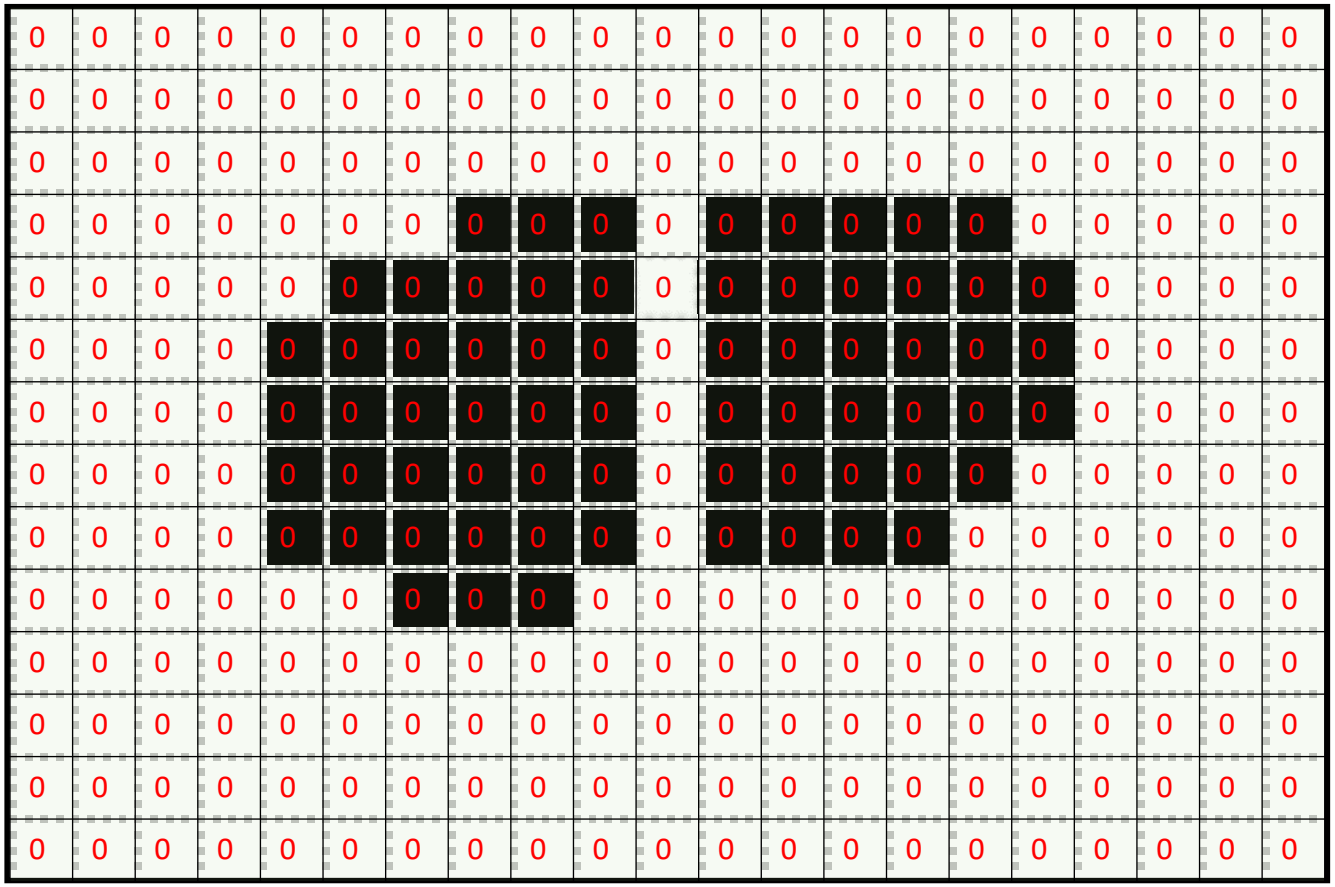
- C.C: foreground
  hole filling: background



Connect Component Labeling



Connect Component Labeling



Mask

0: not labeled

1: background

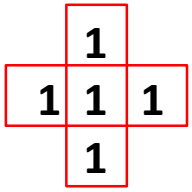
2: object #1

3: object #2

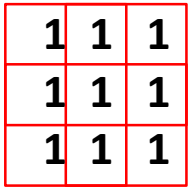
…

(need a counter)

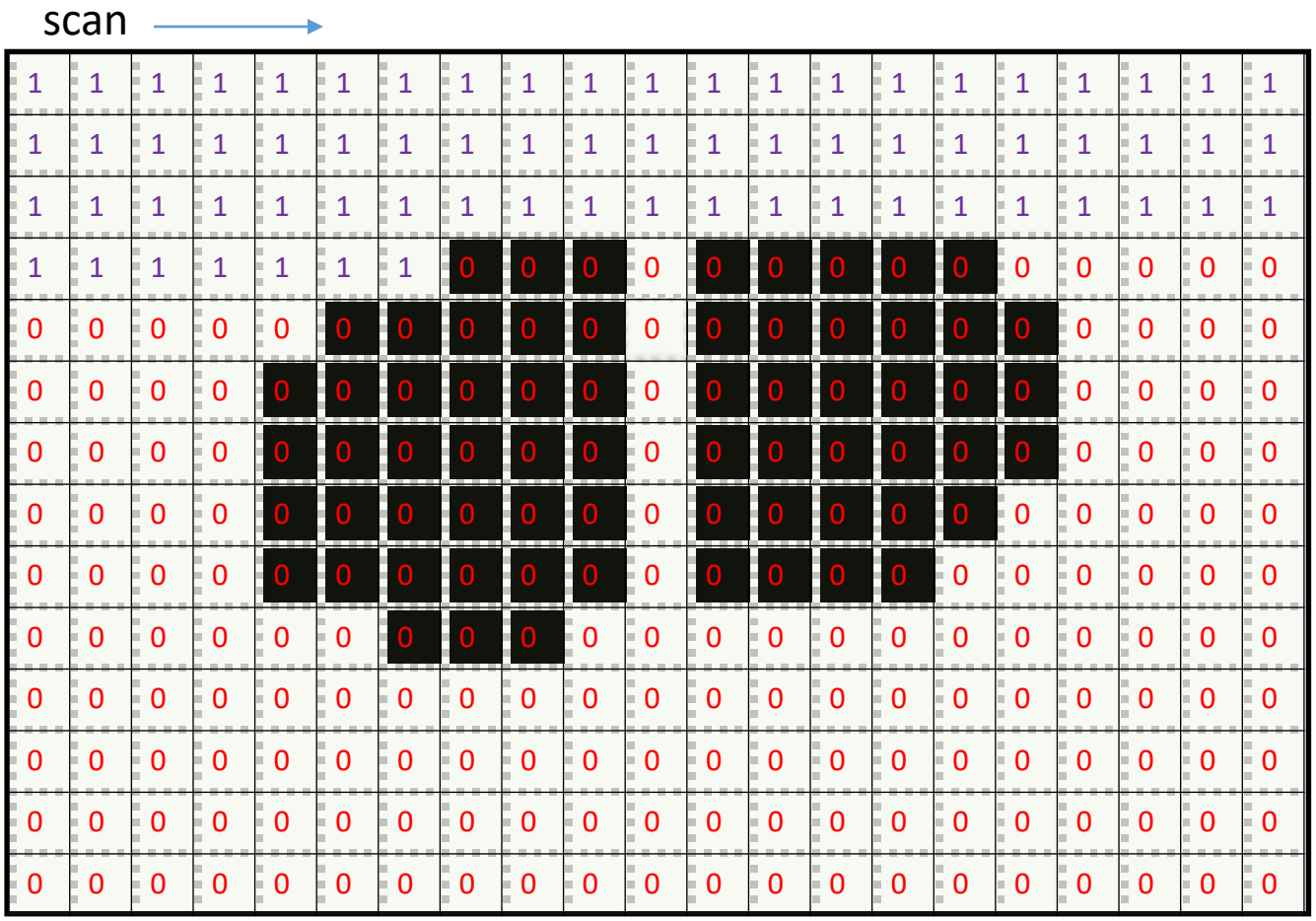
4-connected



8-connected



Connect Component Labeling



Mask

0: not labeled

1: background

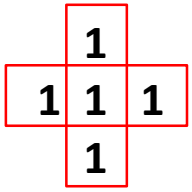
2: object #1

3: object #2

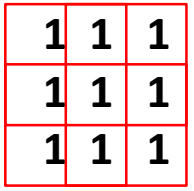
…

(need a counter)

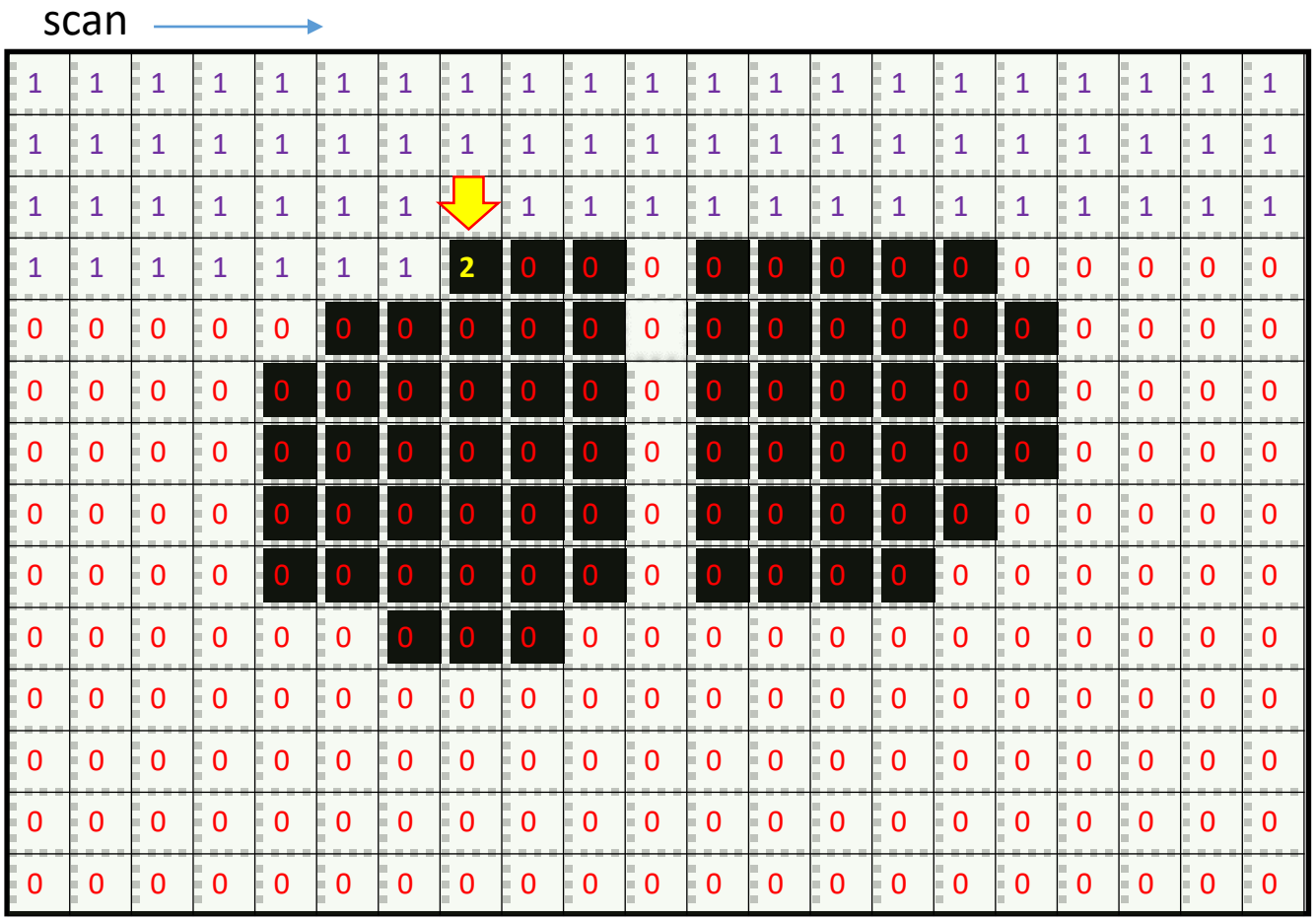
4-connected



8-connected



Connect Component Labeling



Mask

0: not labeled

1: background

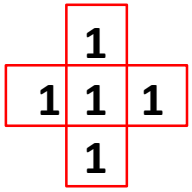
2: object #1

3: object #2

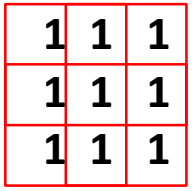
…

(need a counter)

4-connected



8-connected



Connect Component Labeling

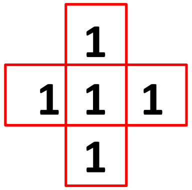
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Search order: up, right, down, left

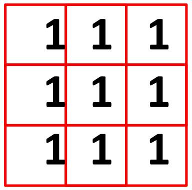
Mask

0: not labeled
1: background
2: object #1
3: object #2
…
(need a counter)

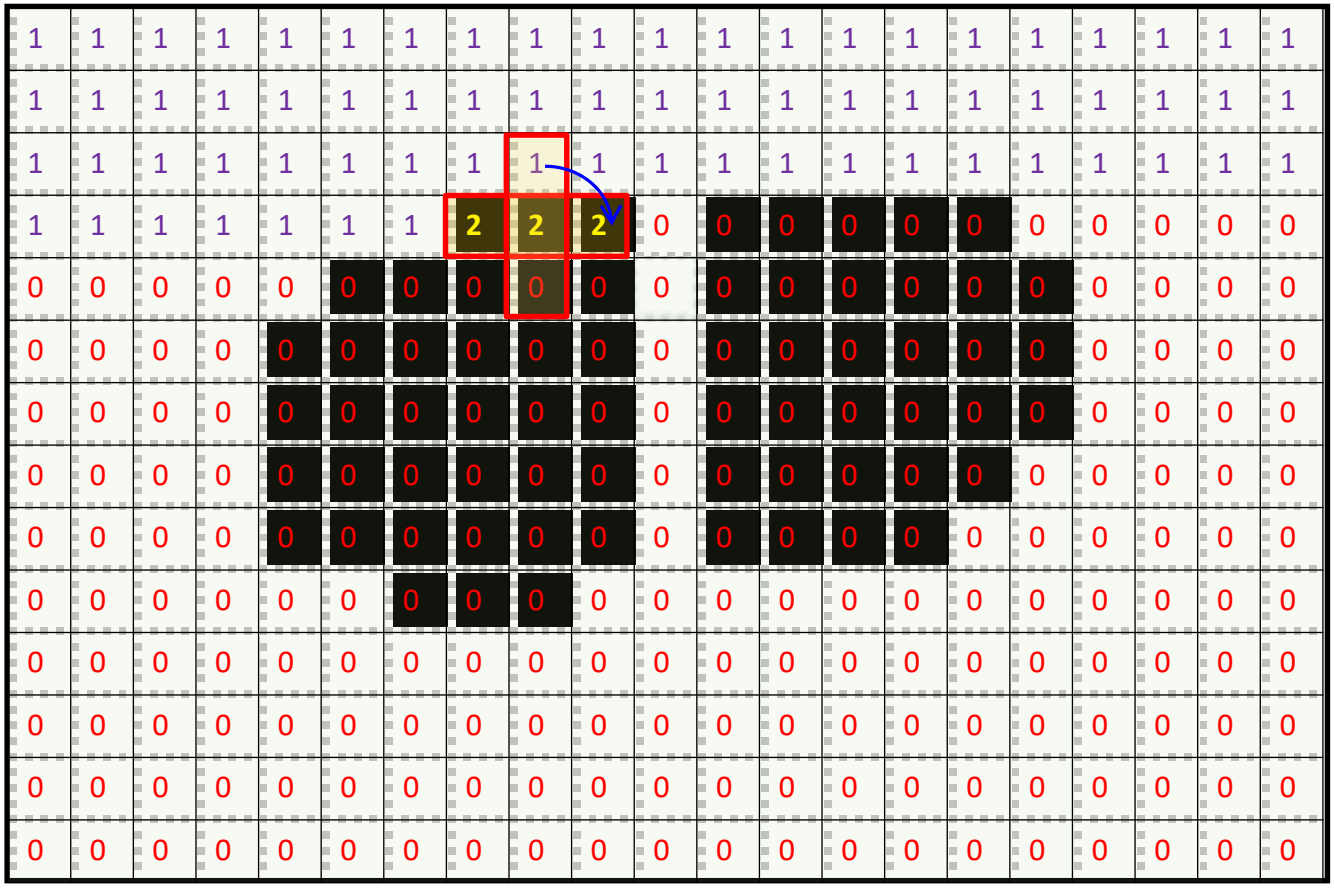
4-connected



8-connected



Connect Component Labeling



Search order: up, right, down, left

Mask

0: not labeled

1: background

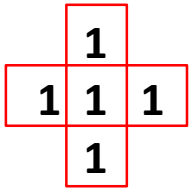
2: object #1

3: object #2

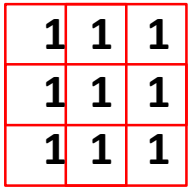
…

(need a counter)

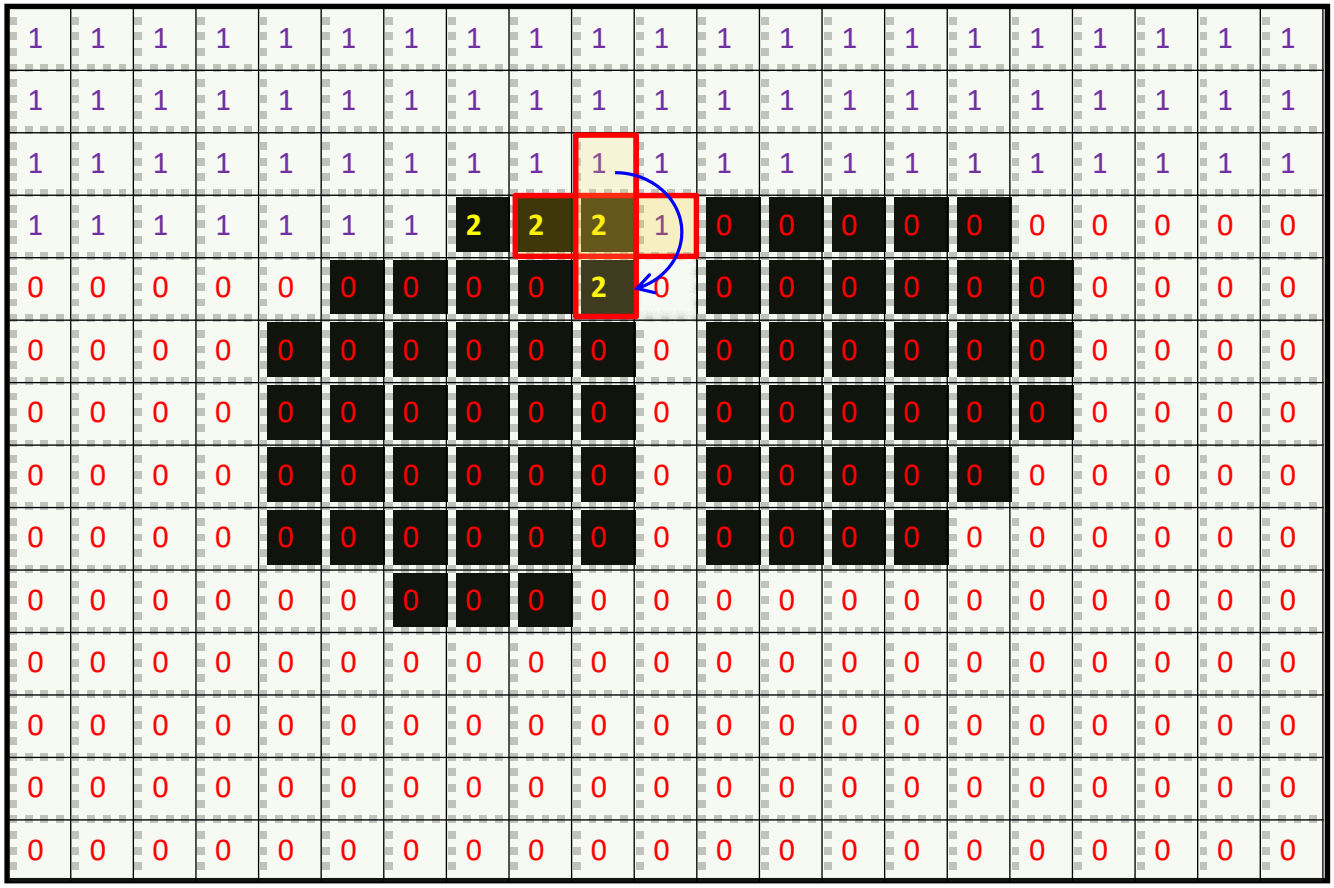
4-connected



8-connected



Connect Component Labeling



Search order: up, right, down, left

Mask

0: not labeled

1: background

2: object #1

3: object #2

…

(need a counter)

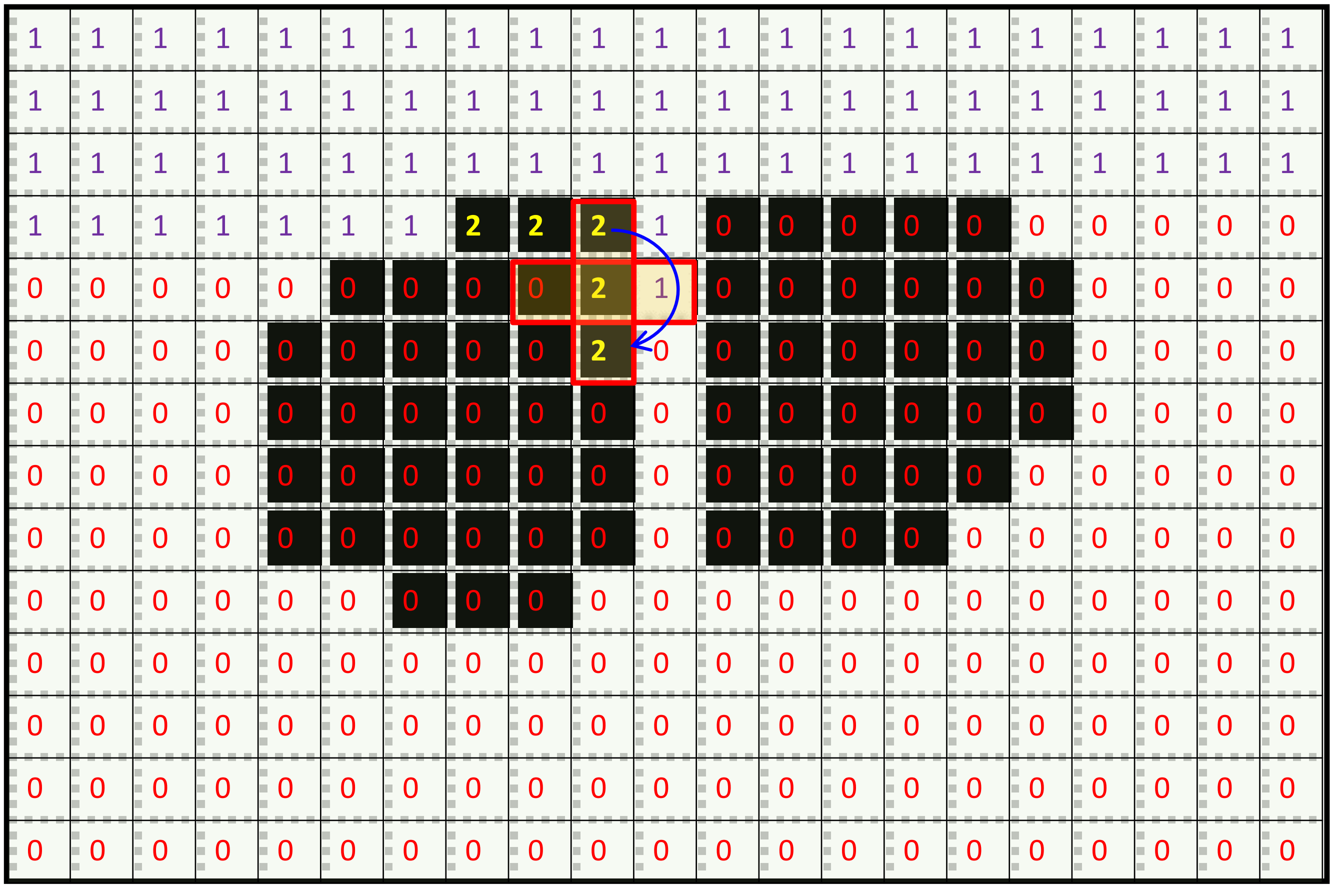
4-connected

	1	
1	1	1
	1	

8-connected

1	1	1
1	1	1
1	1	1

Connect Component Labeling

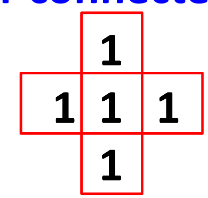


Mask

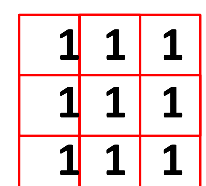
- 0: not labeled
- 1: background
- 2: object #1
- 3: object #2
- …

(need a counter)

4-connected

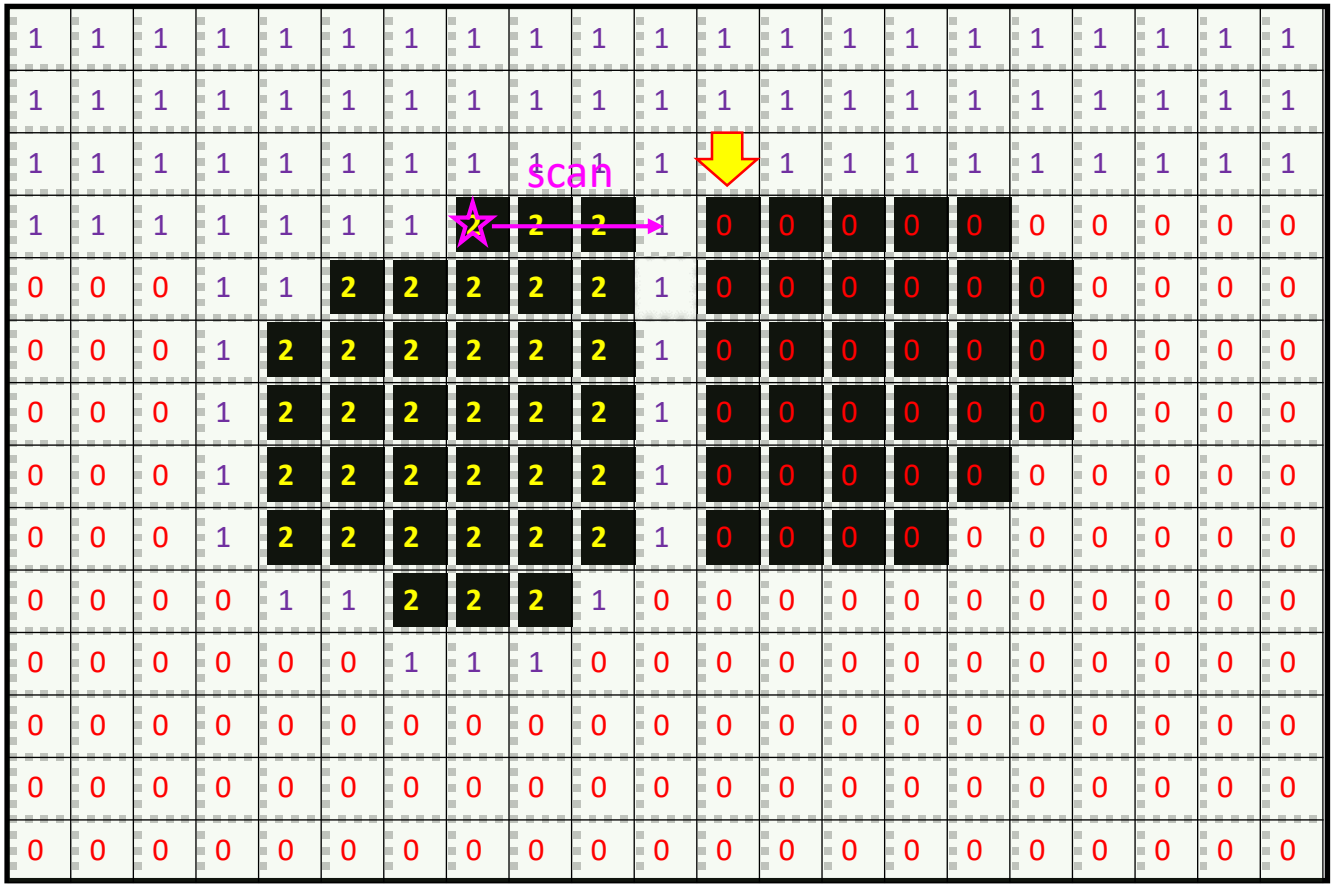


8-connected



Search order: up, right, down, left

Connect Component Labeling



Search order: up, right, down, left

Mask

0: not labeled

1: background

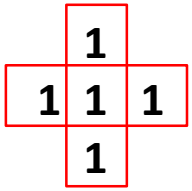
2: object #1

3: object #2

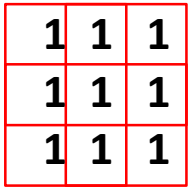
…

(need a counter)

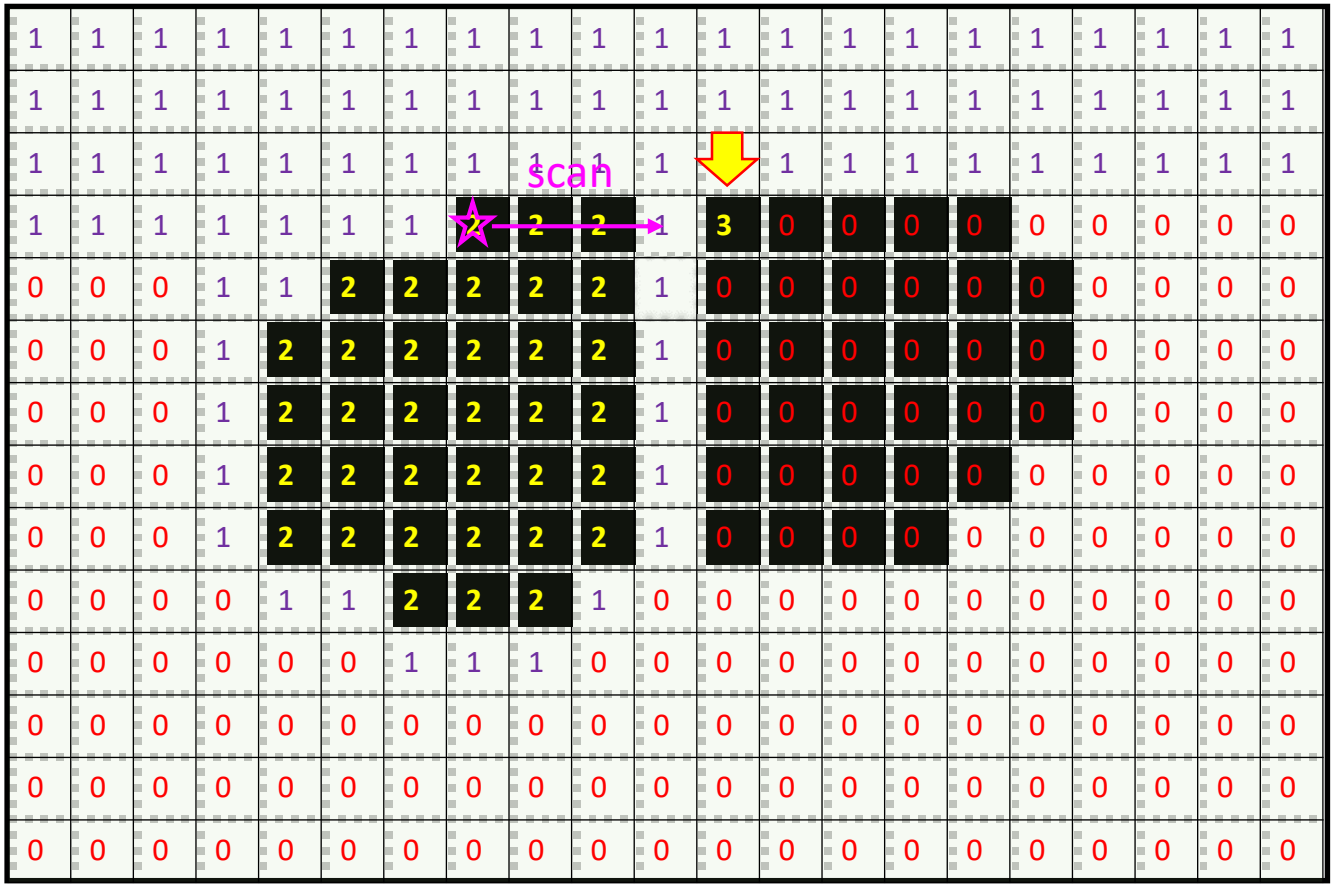
4-connected



8-connected



Connect Component Labeling



Search order: up, right, down, left

Mask

0: not labeled

1: background

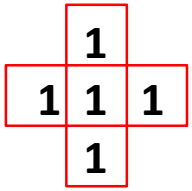
2: object #1

3: object #2

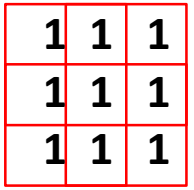
…

(need a counter)

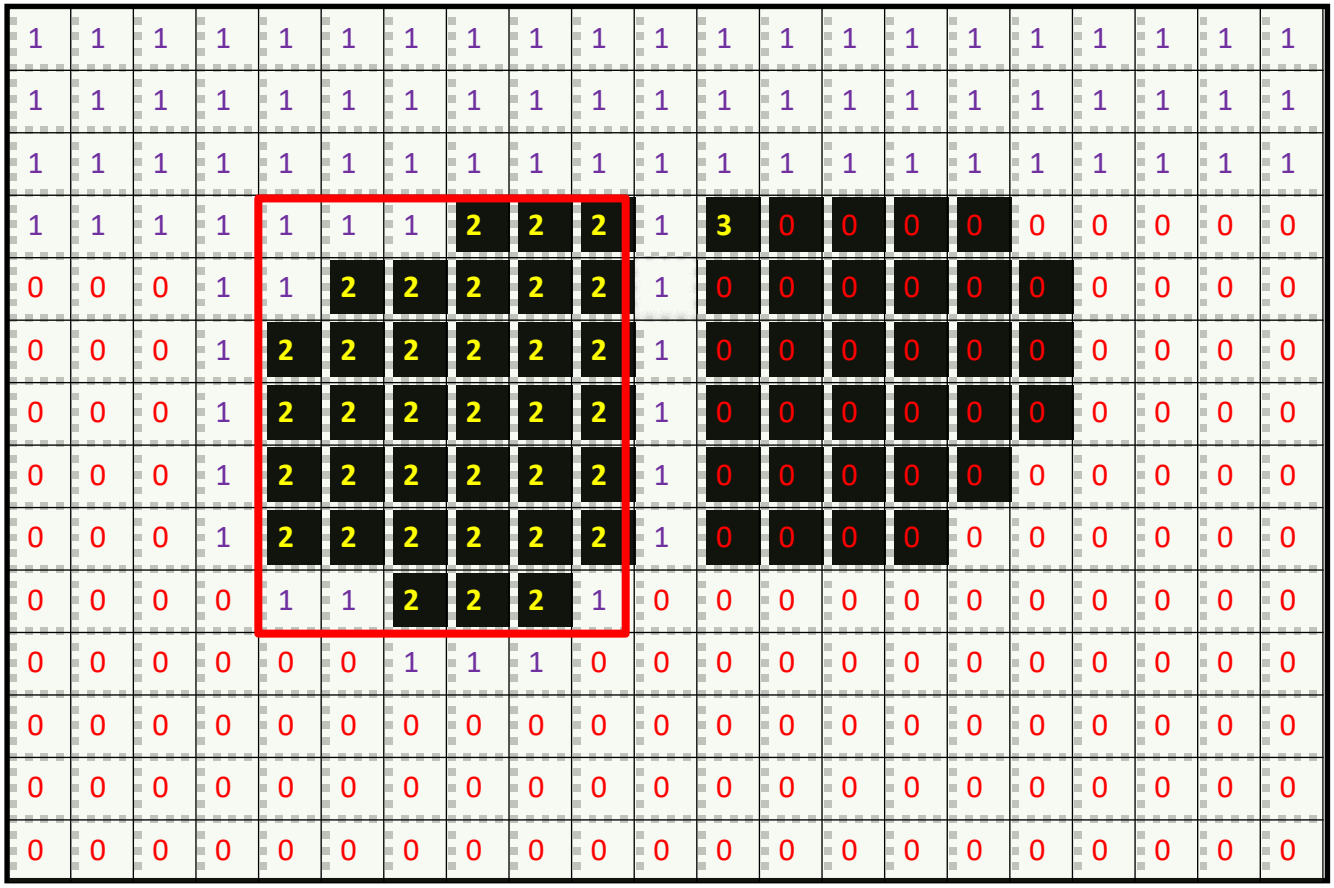
4-connected



8-connected



Bounding Box



Mask

0: not labeled

1: background

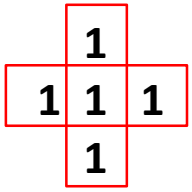
2: object #1

3: object #2

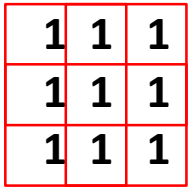
…

(need a counter)

4-connected



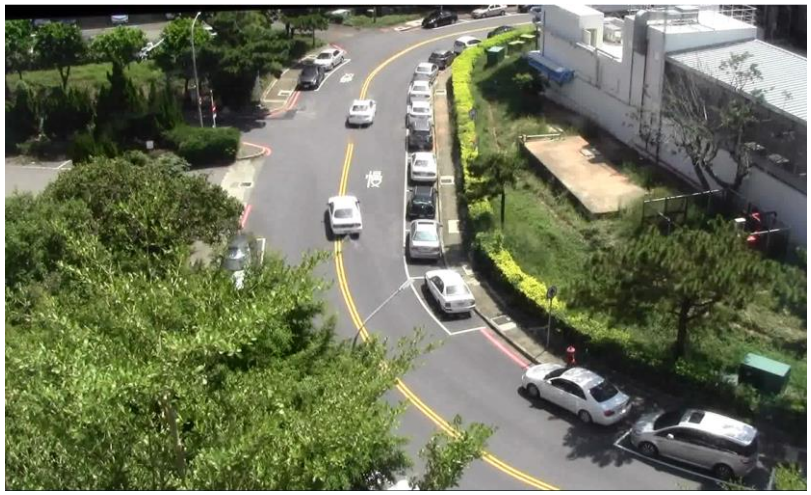
8-connected



When performing connected component labelling

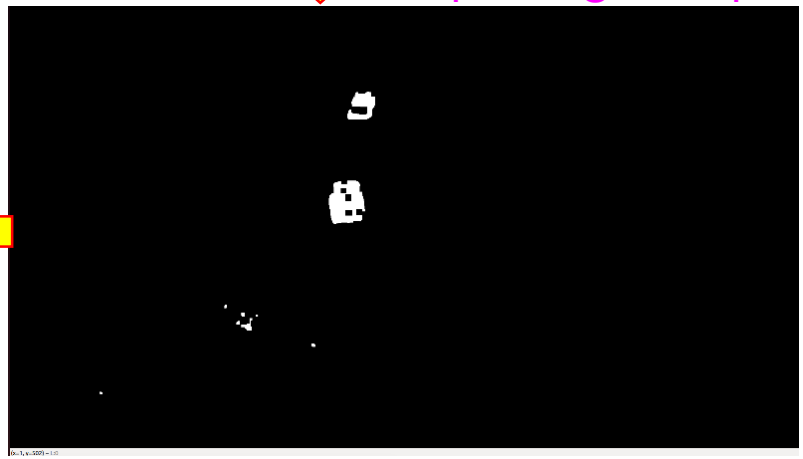
Record the max_x, max_y, min_x, min_y for each object

Example



Morphological operation

Bounding box



OpenCV Functions

- void findContours (InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

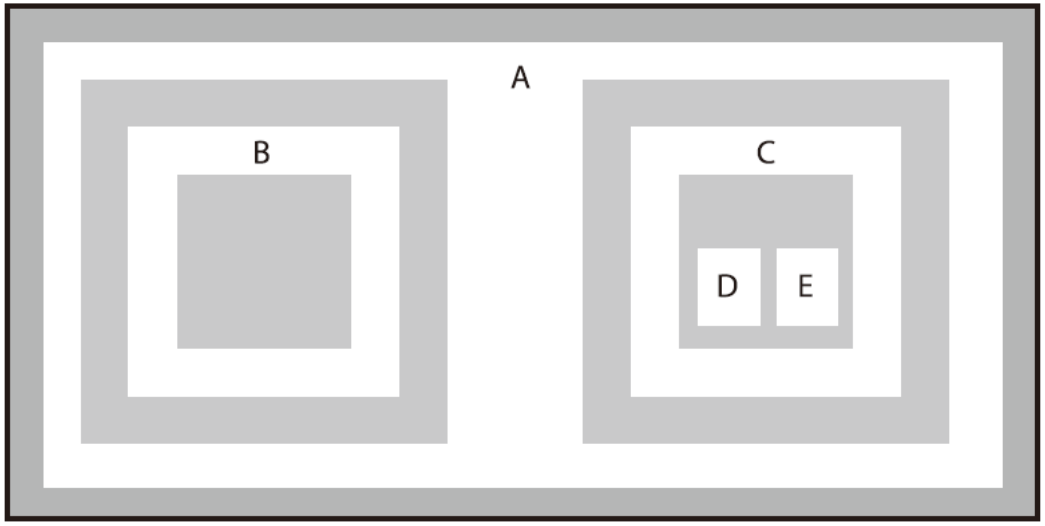
- void drawContours (InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())

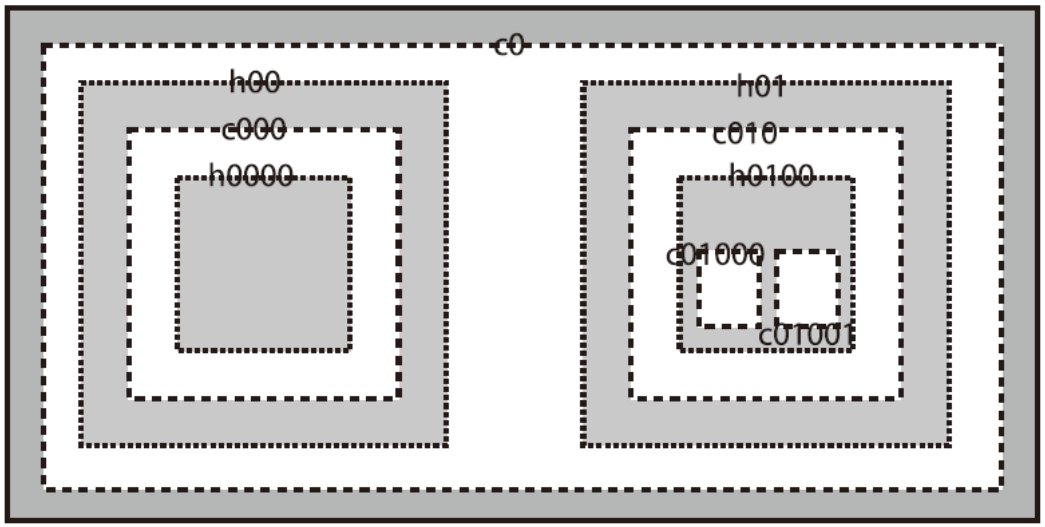
- Rect boundingRect (InputArray **points**)

OpenCV -- findContours

- void findContours (InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())
  - **image**: Source, an 8-bit single-channel image.
    - Non-zero pixels are treated as 1's.
    - Zero pixels remain 0's, so the image is treated as binary.
  - **contours**: Detected contours.
    - Each contour is stored as a vector of points.
  - **hierarchy**: Optional output vector, containing information about the image topology.
    - It has as many elements as the number of contours.
  - **mode**: Contour retrieval mode
  - **method**: Contour approximation method

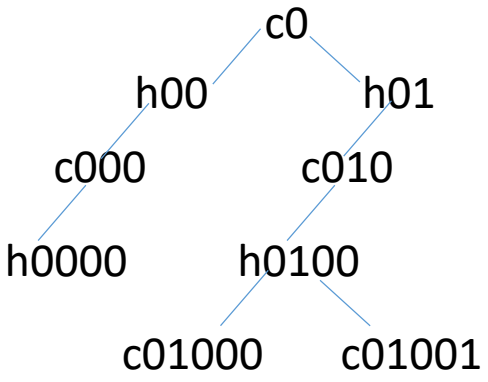
OpenCV -- findContours





Contour hierarchy

- White regions on a black background
- c: contour; h: hole
- Contours (dashed lines): *exterior boundaries*
- Contours (dotted lines): *interior boundaries*
- *Contour tree:*



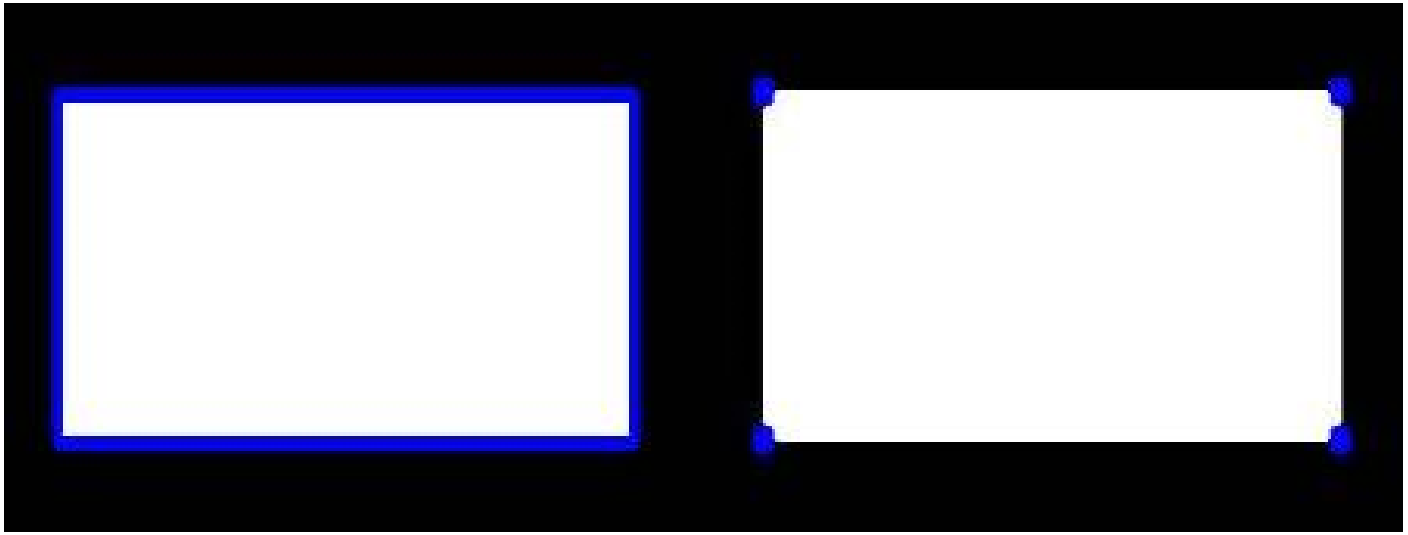
OpenCV -- findContours

- void findContours (InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())

- **mode**
  - CV_RETR_EXTERNAL：只取最外層的輪廓。
  - CV_RETR_LIST：取得所有輪廓，不建立階層(hierarchy)。
  - CV_RETR_CCOMP：取得所有輪廓，儲存成兩層的階層，首階層為物件外圍，第二階層為內部空心部分的輪廓，如果更內部有其餘物件，包含於首階層。
  - CV_RETR_TREE：取得所有輪廓，以全階層的方式儲存。

Enumerator	
RETR_EXTERNAL Python: cv.RETR_EXTERNAL	retrieves only the extreme outer contours. It sets `hierarchy[i][2]=hierarchy[i][3]=-1` for all the contours.
RETR_LIST Python: cv.RETR_LIST	retrieves all of the contours without establishing any hierarchical relationships.
RETR_CCOMP Python: cv.RETR_CCOMP	retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
RETR_TREE Python: cv.RETR_TREE	retrieves all of the contours and reconstructs a full hierarchy of nested contours.

- **method**：
  - CV_CHAIN_APPROX_NONE：stores absolutely all the contour points.
  - CV_CHAIN_APPROX_SIMPLE：compresses horizontal, vertical, and diagonal segments and leaves only their end points



Try it!

findContours(image, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);

OpenCV -- drawContours

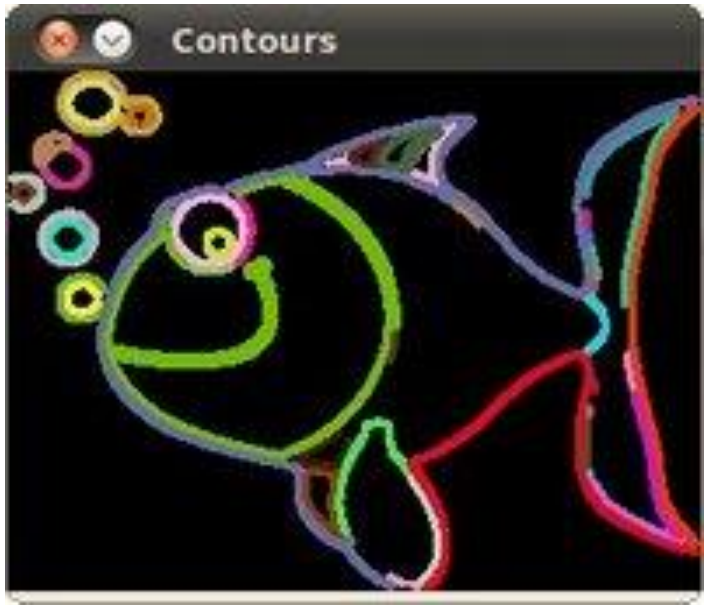
- void drawContours (InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())
  - **image** – Destination image.
  - **contours** – All the input contours.
    - Each contour is stored as a point vector.
  - **contourIdx** – Parameter indicating a contour to draw. If it is negative, all the contours are drawn.
  - **color** – Color of the contours.
  - **thickness** – Thickness of lines the contours are drawn with. If it is negative (e.g., thickness=CV_FILLED), the contour interiors are drawn.
  - **lineType** – Line connectivity. See line() for details.

OpenCV -- drawContours

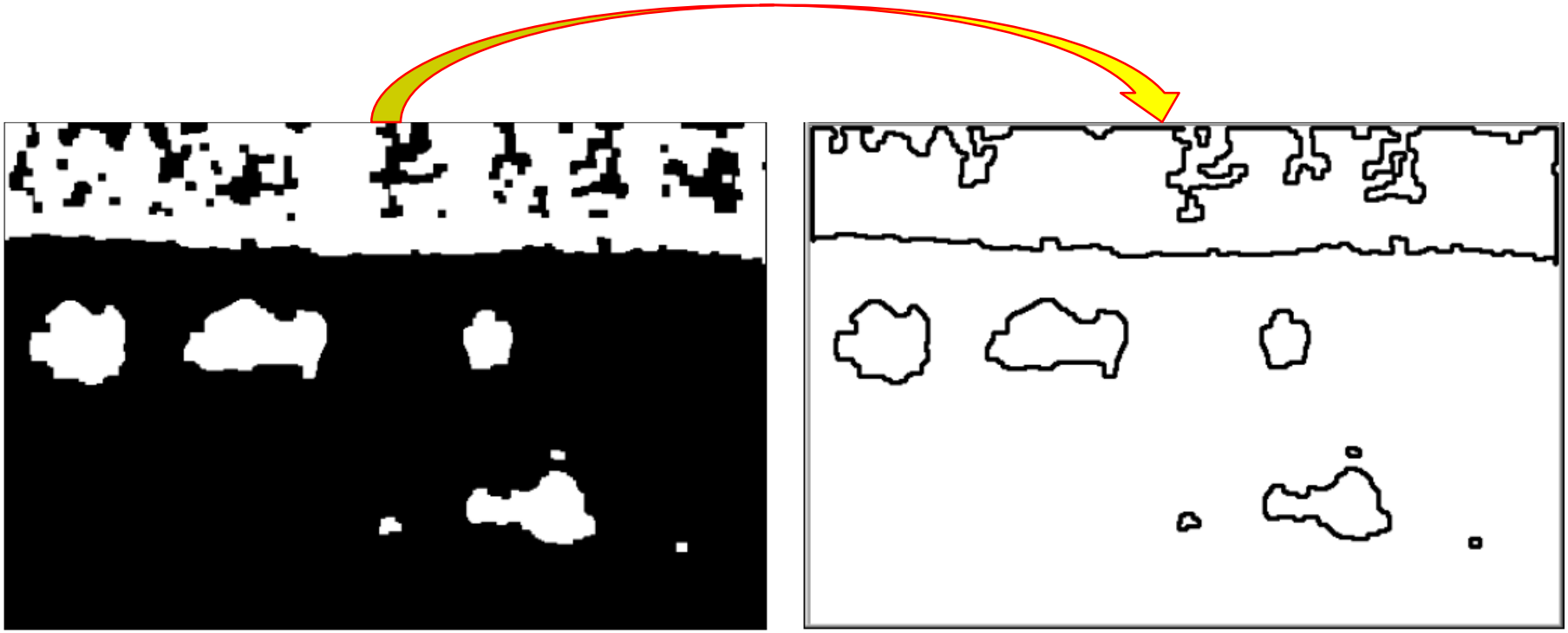
- void drawContours (InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())
  - **hierarchy** – Optional information about hierarchy.
    - It is only needed if you want to draw only some of the contours (see **maxLevel**).
  - **maxLevel** – Maximal level for drawn contours.
    - If it is 0, only the specified contour (**contourIdx**) is drawn.
    - If it is 1, the function draws the contour(s) and all the nested contours.
    - If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on.
    - This parameter is only taken into account when there is hierarchy available.

OpenCV -- findContours

- https://docs.opencv.org/3.4/df/d0d/tutorial_find_contours.html



OpenCV -- findContours



Example

void findContours (InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
RNG rng(12345);  // random number generator
findContours (src, contours, hierarchy, CV_RETR_TREE,
    CV_CHAIN_APPROX_NONE);
for(int i = 0; i<contours.size(); i++){
    Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0, 255), 255);
     drawContours(dst, contours, i, color, 2, 8, hierarchy);
}
imshow("origin", src);
imshow("result", dst);
```

取得所有輪廓，以全階層的方式儲存

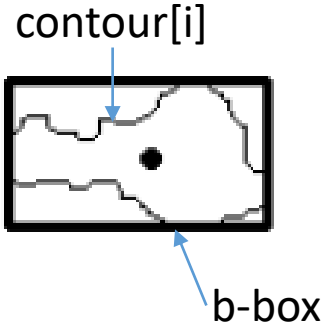
儲存所有輪廓點

contourIdx

thickness, lineType

void drawContours (InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())

OpenCV -- boundingRect



- Rect **boundingRect**(InputArray **points**)

→Rect b-box = boundingRect(contours[i]);

- **Draw a rectangle**

1. void **rectangle**(Mat& **img**, Rect **rec**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

→rectangle(dst, b-box, Scalar(0,0,255),2);

2. void **rectangle**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1,int **lineType**=8, int **shift**=0)

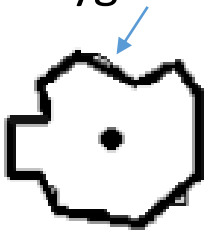
→rectangle(dst, Point(b-box.x, b-box.y),
Point((b-box.x+ b-box.width), (b-box.y+ b-box.height)), Scalar(0,0,255),2);

(**shift:** Number of fractional bits in the point coordinates.)

110011. -->shift =0
1100.11 -->shift =2

Shape descriptors

Polygonal approximation



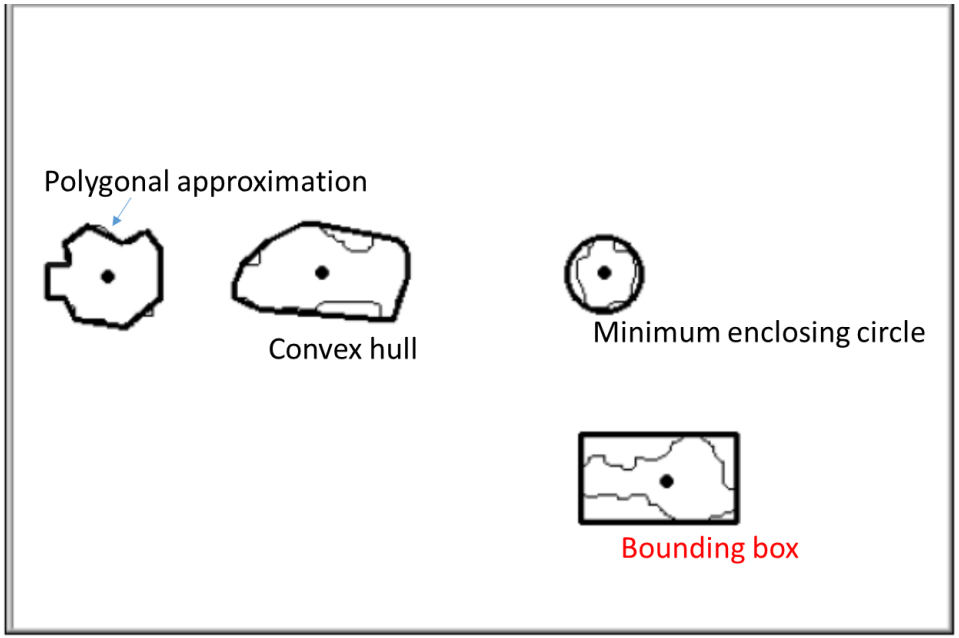
Convex hull

Minimum enclosing circle

Bounding box

OpenCV Functions

- void **approxPolyDP**(InputArray **curve**, OutputArray **approxCurve**, double **epsilon**, bool **closed**)
- void **convexHull**(InputArray **points**, OutputArray **hull**, bool **clockwise**=false, bool **returnPoints**=true)
- void **minEnclosingCircle** (InputArray **points**, Point2f& **center**, float& **radius**)



OpenCV Functions

- void **approxPolyDP** (InputArray **curve**, OutputArray **approxCurve**, double **epsilon**, bool **closed**)

- **curve** – Input vector of 2D points
- **approxCurve** – Result of the approximation. The type should match the type of the input curve.
- **epsilon** – Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.
- **closed** – If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

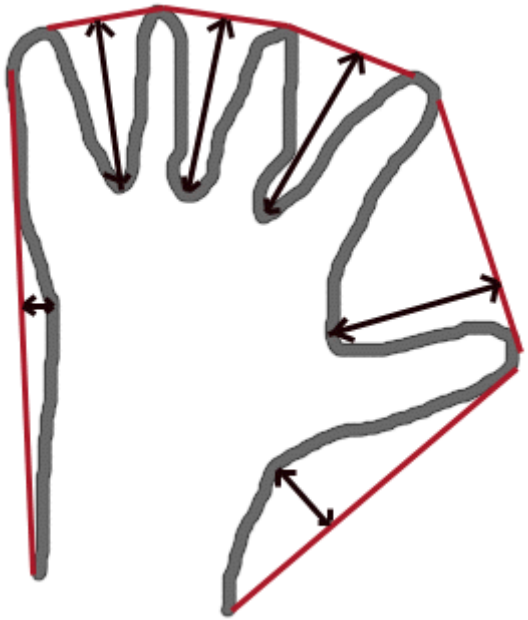
OpenCV Functions

- void **convexHull** (InputArray **points**, OutputArray **hull**, bool **clockwise**=false, bool **returnPoints**=true)

- **points** – Input 2D point set, stored in **std::vector** or **Mat**.
- **hull** – Output convex hull.
  It is either an integer vector of indices or vector of points.
  - In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). →從原本的點中找出幾個來表示
  - In the second case, hull elements are the convex hull points themselves.
- **clockwise** – Orientation flag. If it is true, the output convex hull is oriented clockwise.
- **returnPoints** – Operation flag.
  - In case of a **matrix**, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points.
  - When the output array is **std::vector**, the flag is ignored.

OpenCV Functions

- void **convexHull** (InputArray **points**, OutputArray **hull**, bool **clockwise**=false, bool **returnPoints**=true)

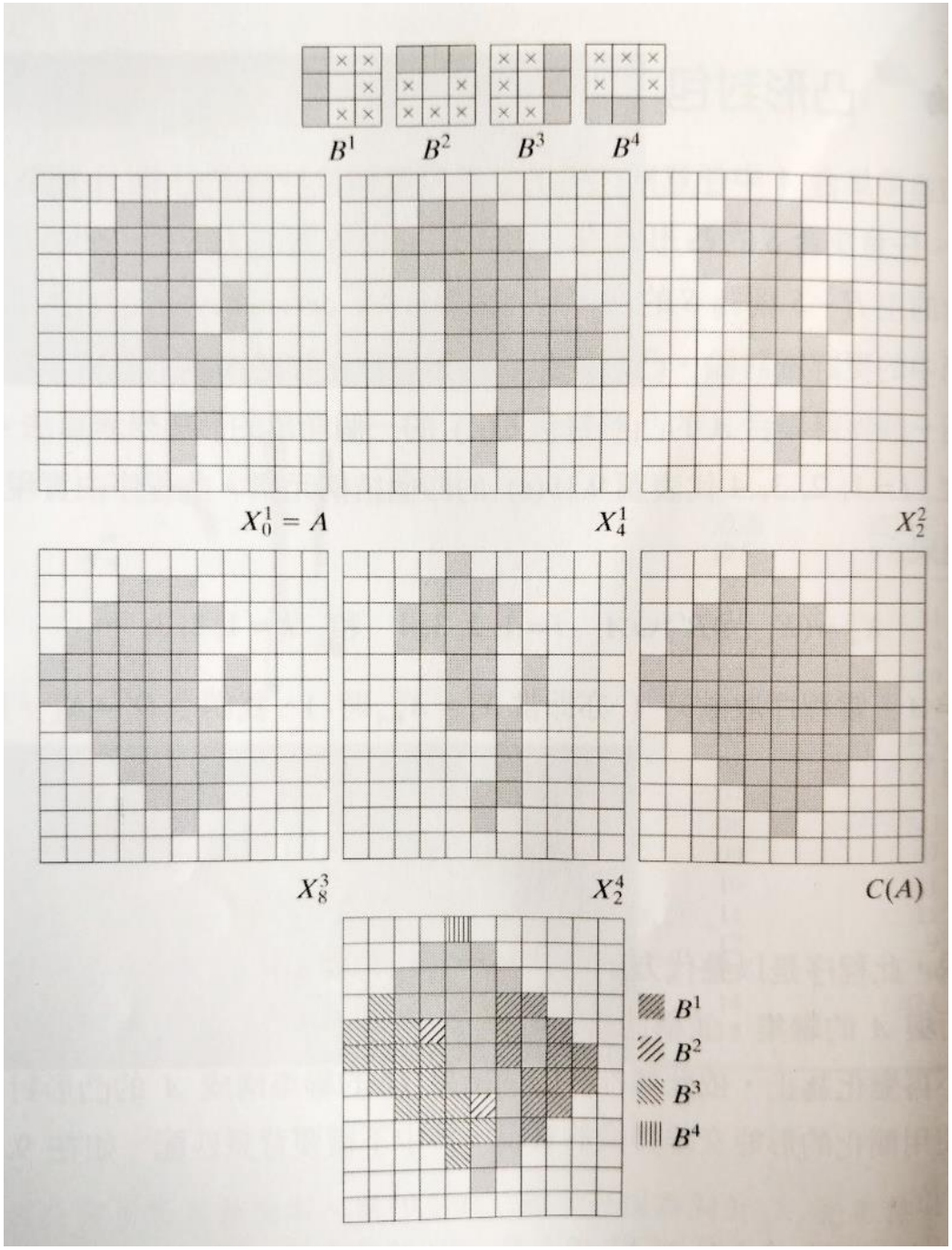


void **convexityDefects**(InputArray contour, InputArray convexhull, OutputArrayconvexityDefects

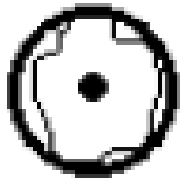
HMT: Hit or Miss 交離轉換

- $X_k^i = (X_{k-1} \circledast B^i) \cup A$
  - i = 1~4
  - k: Iteratively

- $C(A) = \bigcup_{i=1}^{4} D^i$



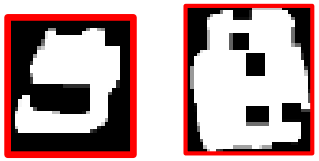
OpenCV Functions

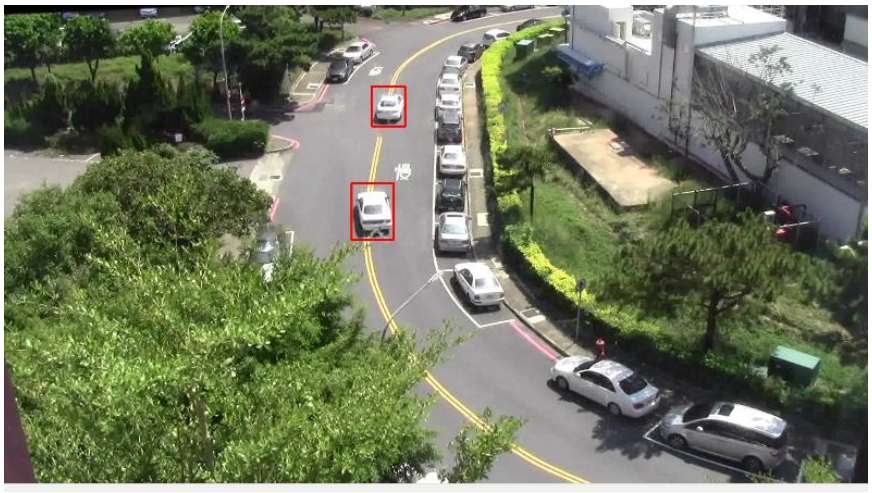


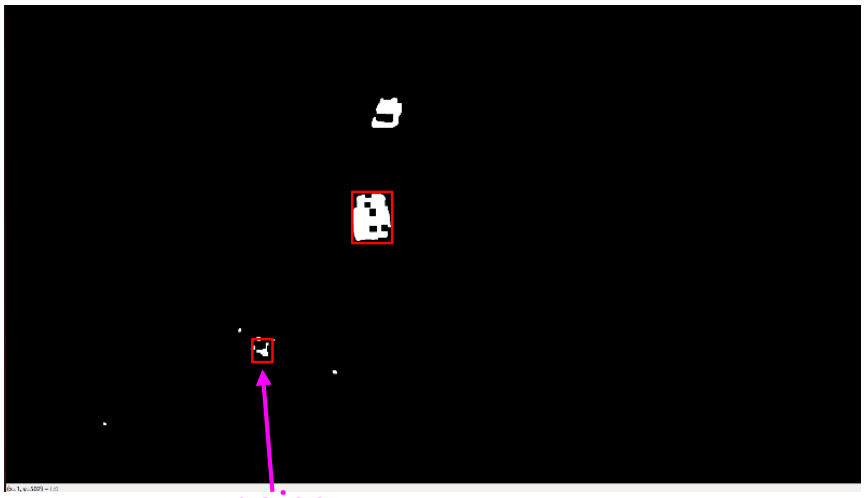
- void **minEnclosingCircle** (InputArray **points**, Point2f& **center**, float& **radius**)
- **points** – Input vector of 2D points
- **center** – Output center of the circle.
- **radius** – Output radius of the circle.

Object filtering

- Object Size
  - Area of the bounding box: width × height
  - Number of foreground pixels



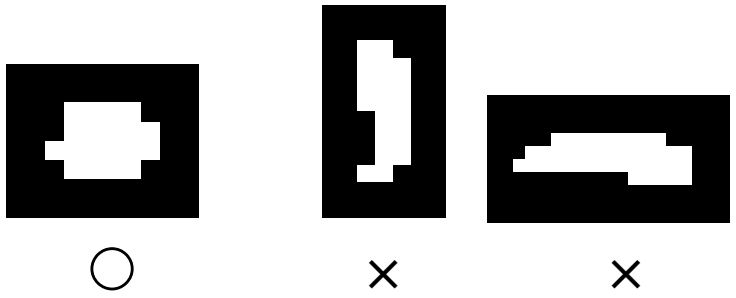




Object filtering

- Object Size
  - Area of the bounding box
  - Number of foreground pixels
- Shape
  - Aspect ratio of the bounding box

Pose classification

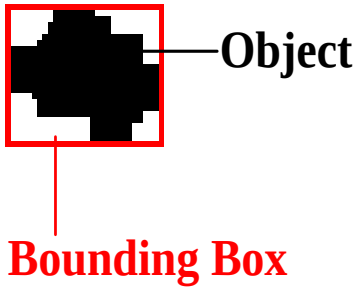


○ × ×

Ball detection (Aspect ratio ≈ 1)

Object filtering

- Object Size
  - Area of the bounding box
  - Number of foreground pixels
- Shape
  - Aspect ratio of the bounding box
- Compactness
  - Object_size/Bounding_box_area



You can design other features by yourselves.

Example --- baseball detection

Demo

- Q&A