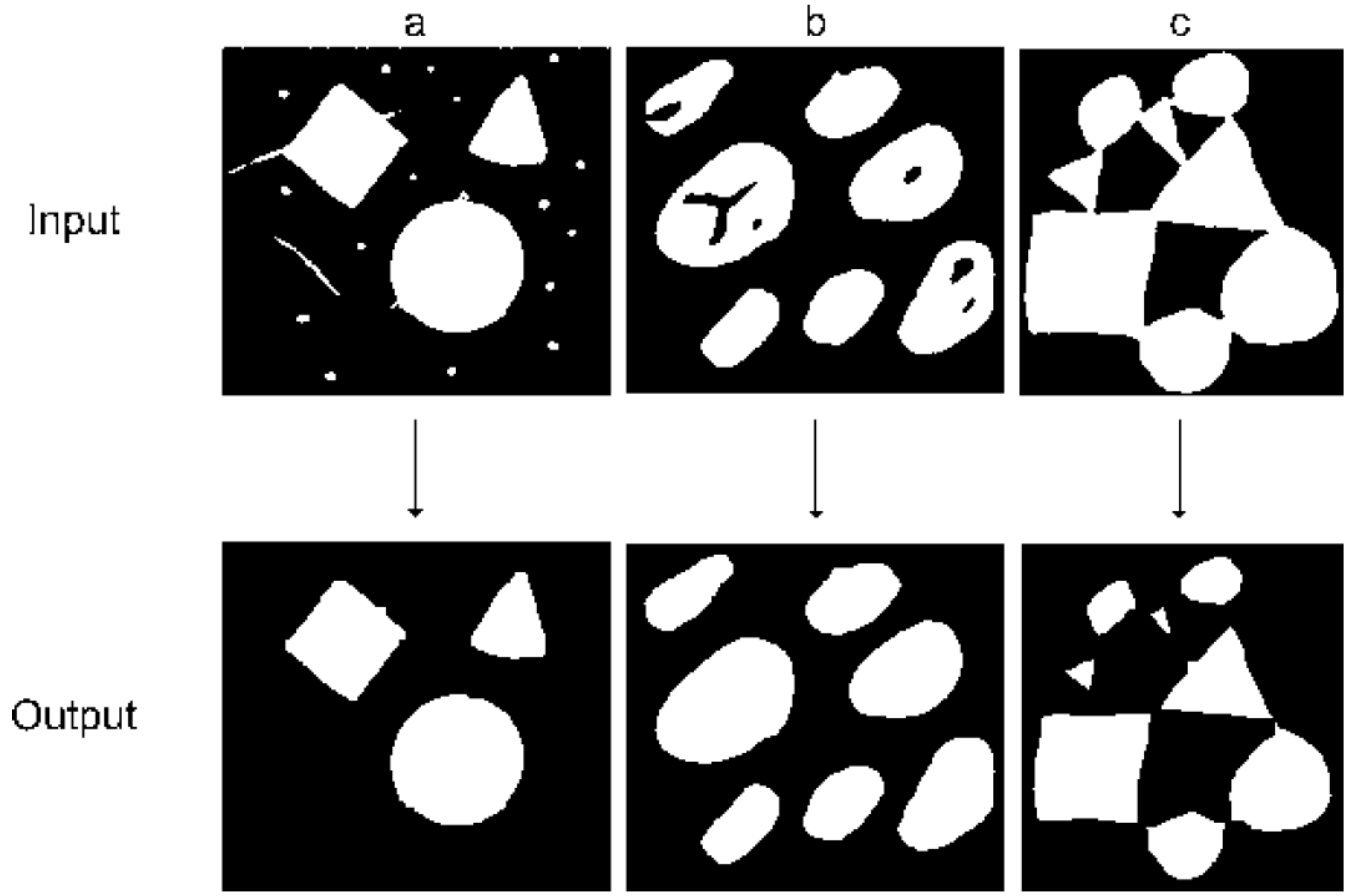
# Basics of Machine Vision

Primož Podržaj

Lecture 04

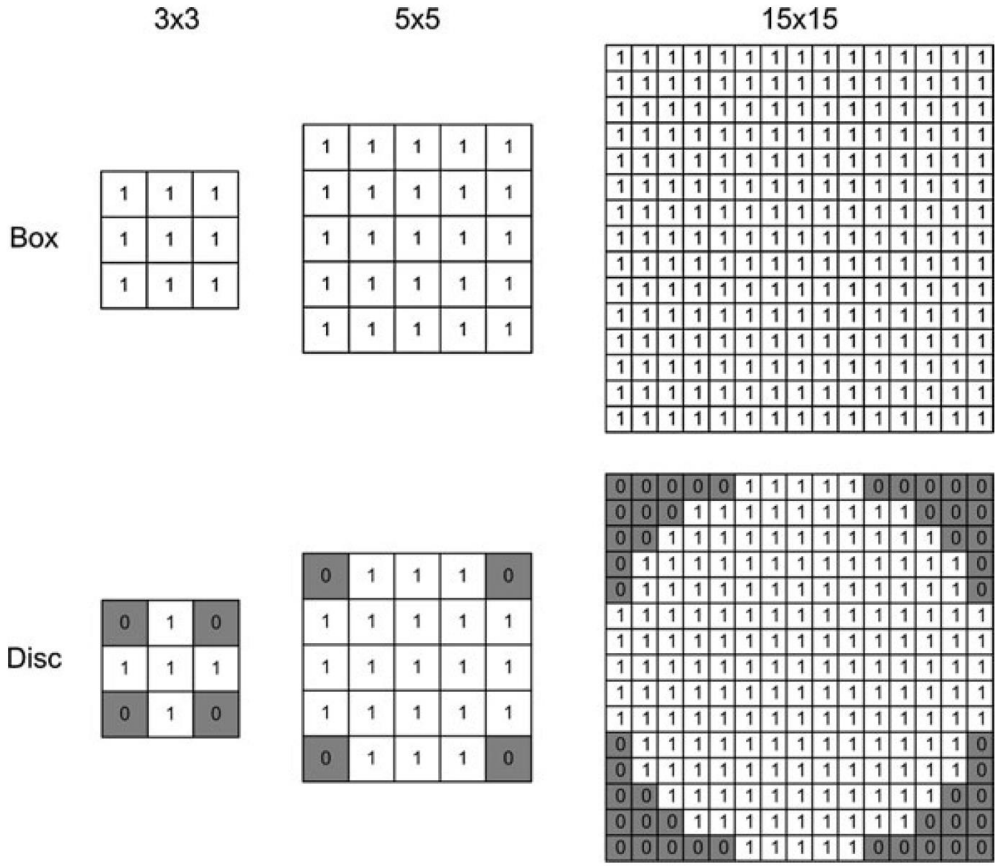
# Morphology

- (a) Removing small objects.
- (b) Filling holes.
- (c) Isolating objects



# Structuring elements

- In morphology, the kernel is denoted a **structuring element** and contains ‘0’s and ‘1’s.



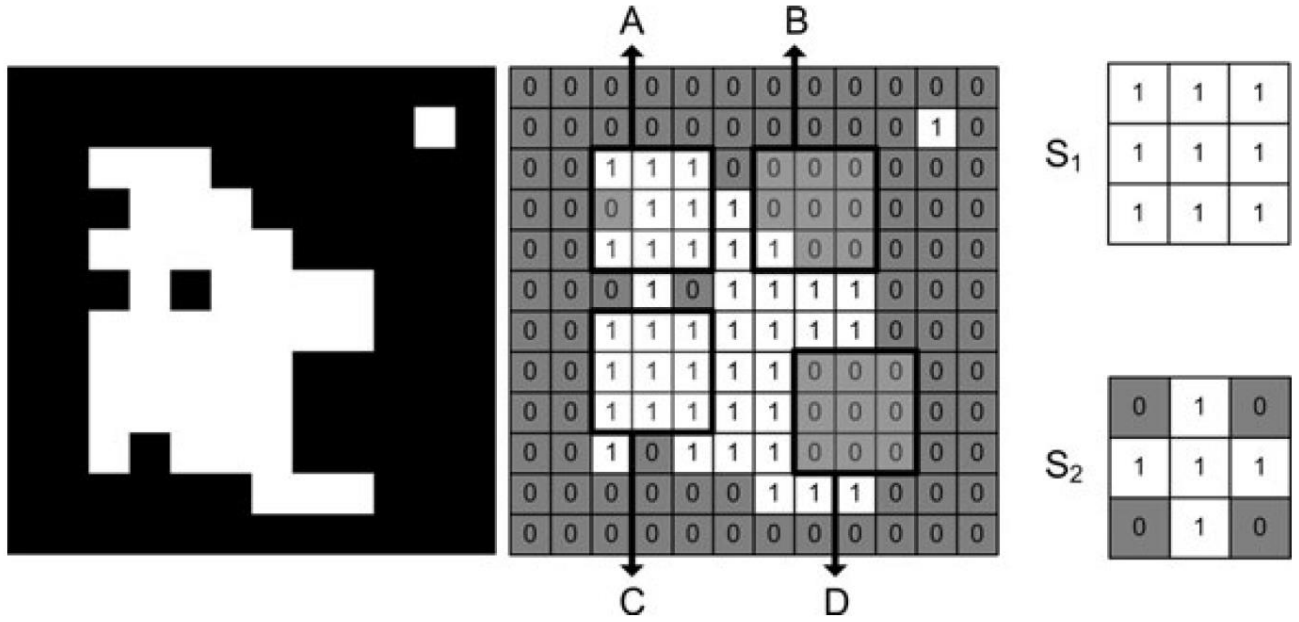
# Application of structuring elements

A structuring element is not applied in the same way as we saw before for  the kernels. Instead of using multiplications and additions in the calculations, a structuring element is applied using either a **Hit** or a **Fit** operation. Applying one of these operations to each pixel in an image is denoted **Dilation** and **Erosion**, respectively. Combining these two methods can result in powerful image processing tools known as **Compound Operations**.

# Hit and fit

- For each ‘1’ in the structuring element we investigate whether the pixel at the same position in the image is also a ‘1’. If this is the case for **just one** of the ‘1’s in the structuring element we say that the structuring element **hits** the image at the pixel position in question (the one on which the structuring element is centered). This pixel is therefore set to ‘1’ in the output image. Otherwise it is set to ‘0’.
- For each ‘1’ in the structuring element we investigate whether the pixel at the same position in the image is also a ‘1’. If this is the case **for all** the ‘1’s in the structuring element we say that the structuring element **fits** the image at the pixel position in question (the one on which the structuring element is centered). This pixel is therefore set to ‘1’ in the output image. Otherwise it is set to ‘0’.

# Graphical demonstration




| Position | *SE* | Fit | Hit |
|---|---|---|---|
| A | $S_1$ | No | Yes |
| A | $S_2$ | No | Yes |
| B | $S_1$ | No | Yes |
| B | $S_2$ | No | No |
| C | $S_1$ | Yes | Yes |
| C | $S_2$ | Yes | Yes |
| D | $S_1$ | No | No |
| D | $S_2$ | No | No |

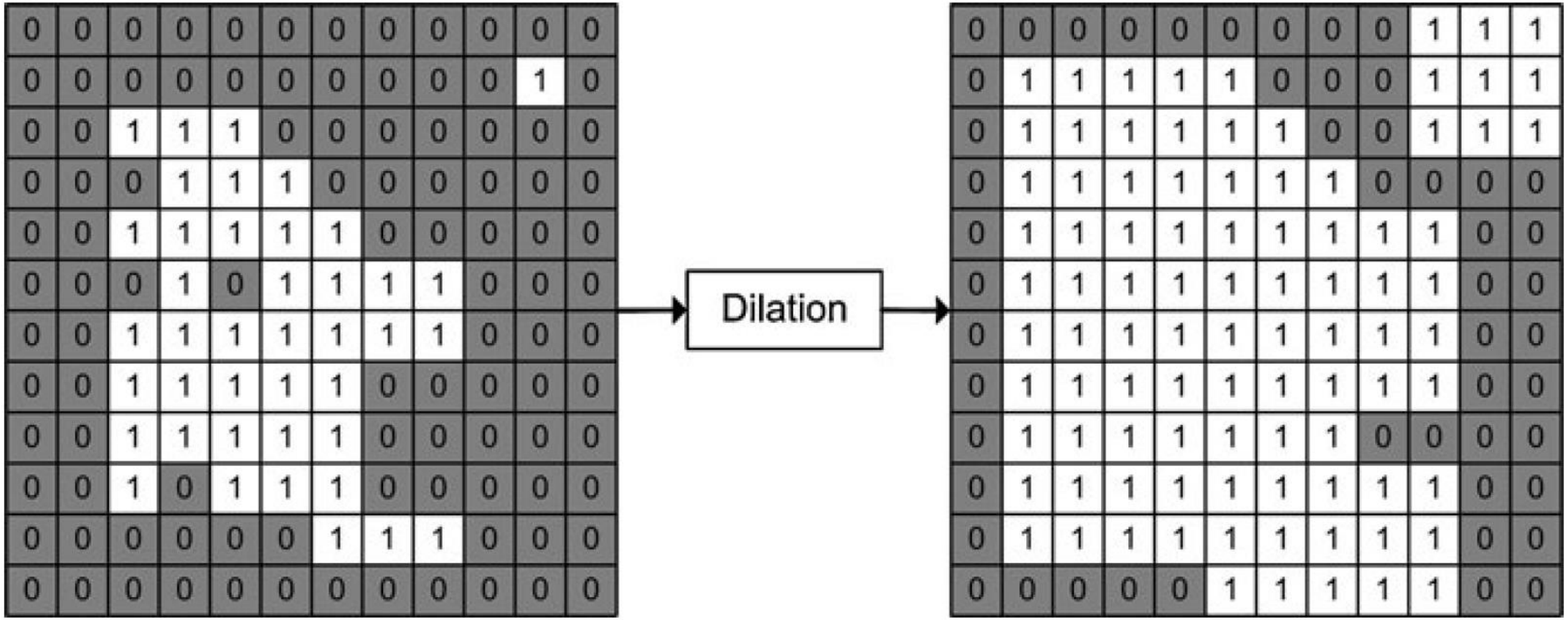
# Dilation

- Applying Hit to an entire image is denoted **Dilation** and is written as:

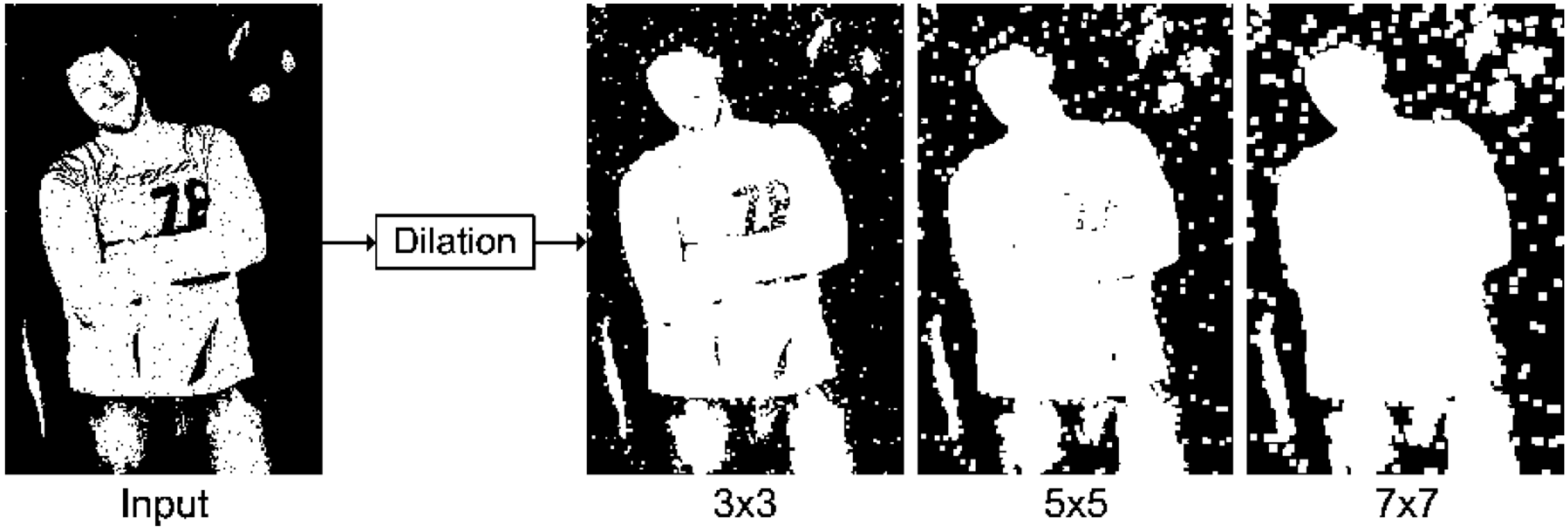
$$g(x, y) = f(x, y) \oplus SE$$

The term dilation refers to the fact that the object in the binary image is increased in size. In general, dilating an image results in objects becoming bigger, small holes being filled, and objects being merged. How big the effect is depends on the size of the structuring element. It should be noticed that a large structuring element can be implemented by iteratively applying a smaller structuring element.

# Dilation



# Dilation with different sized structuring elements



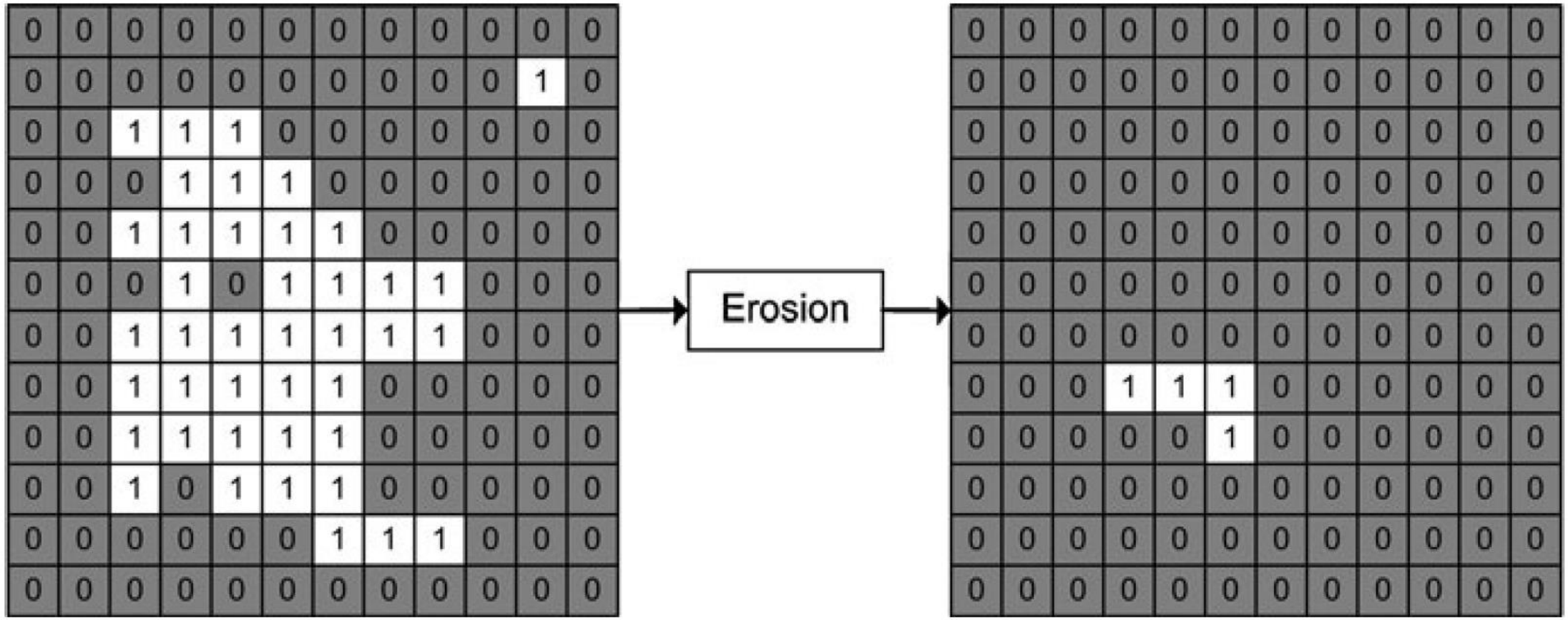
# Erosion

- Applying Fit to an entire image is denoted **Erosion** and is written as

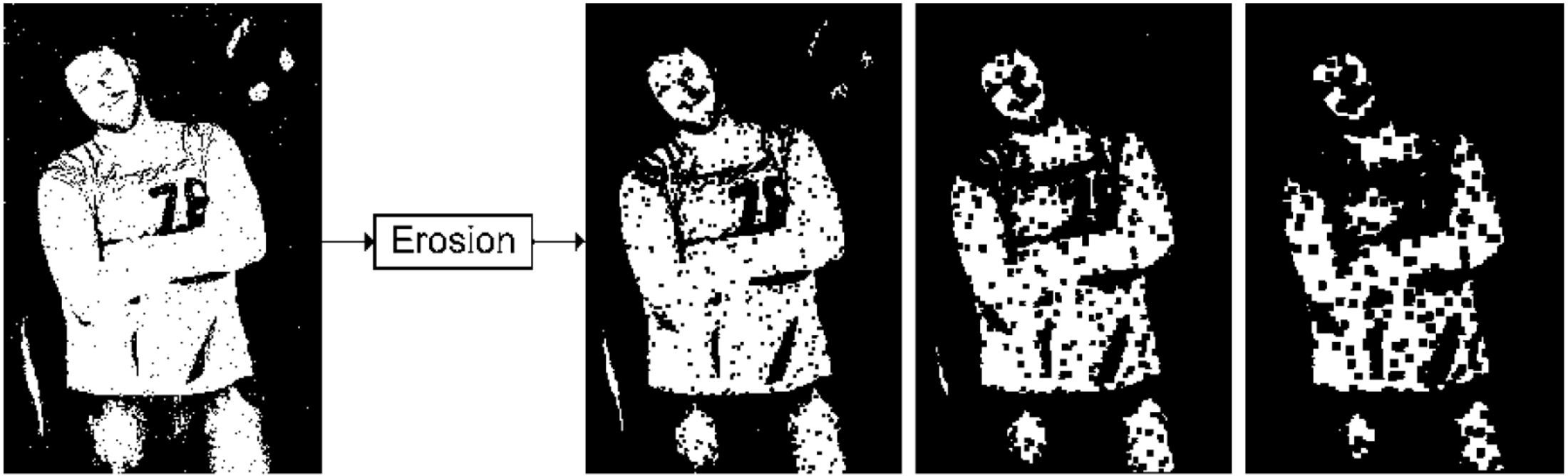
$$g(x, y) = f(x, y) \ominus SE$$

The term erosion refers to the fact that the object in the binary image is decreased in size. In general, erosion of an image results in objects becoming smaller, small objects disappearing, and larger objects splitting into smaller objects.

# Erosion

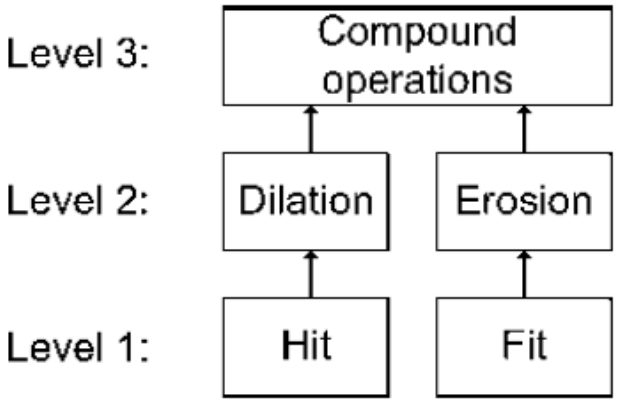


# Erosion with different sized structuring elements



# Compound operations

- The three levels of operation involved in Morphology



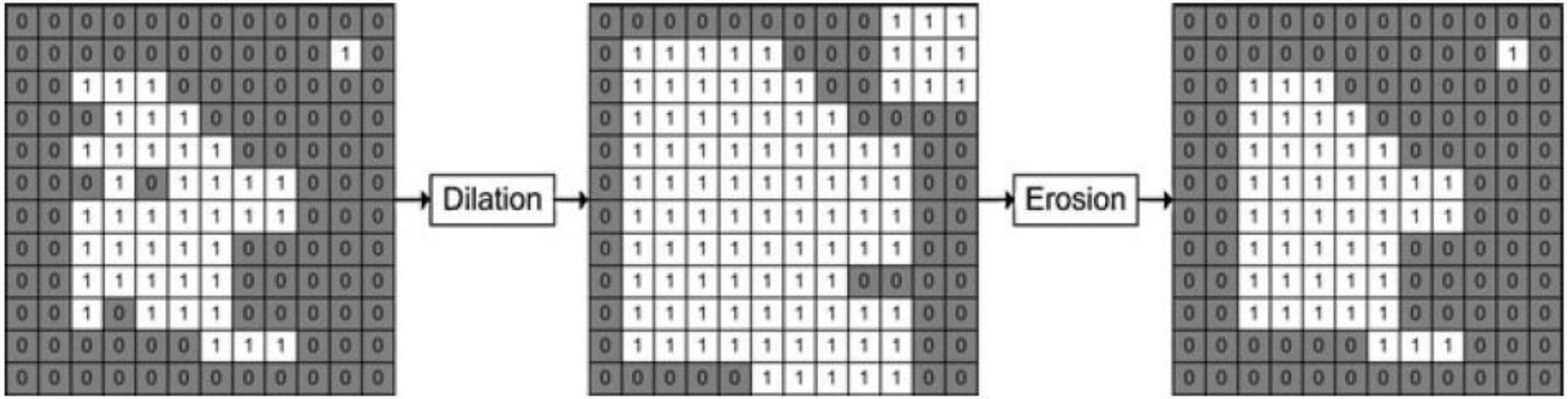

Combining dilation and erosion in different ways results in a number of different image processing tools. These are denoted compound operations. The three most common compound operations, namely **Opening**, **Closing**, and **Boundary Detection** will be presented.

# Closing

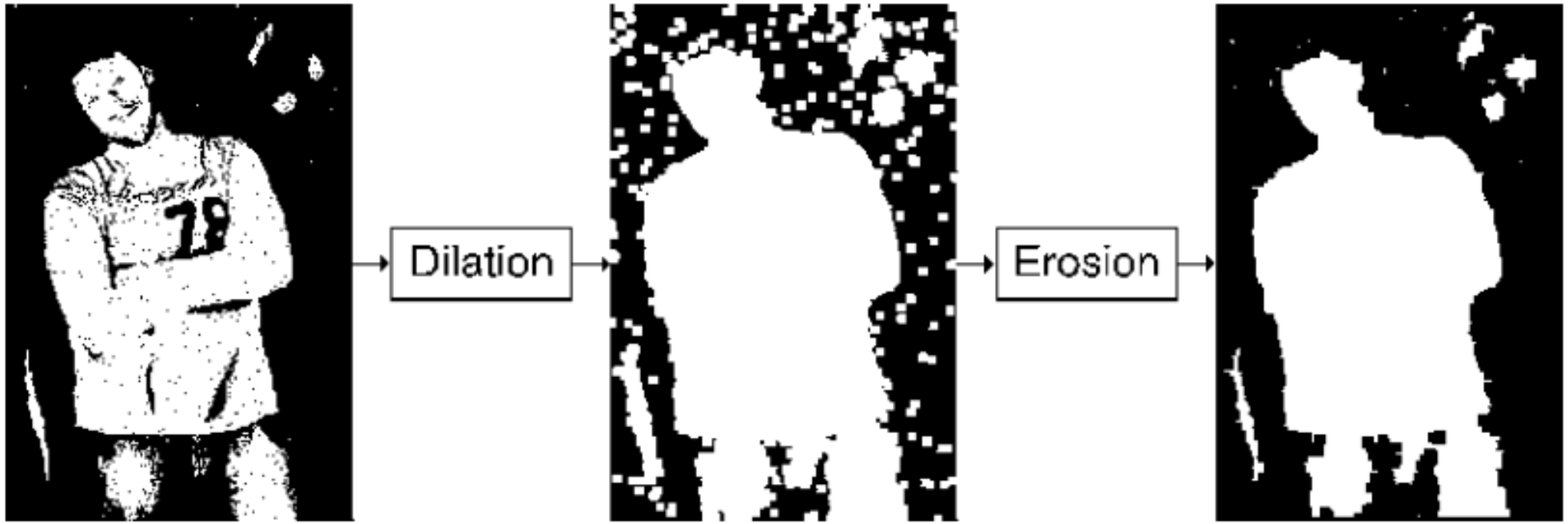
- Closing deals with the problem associated with dilation, namely that the objects increase in size when we use dilation to fill the holes in objects. This is a problem in situations where, for example, the size of the object (number of pixels) matters. The solution to this problem is luckily straightforward: we simply shrink the object by following the Dilation by an Erosion. This operation is denoted Closing and is written as:

$$g(x, y) = f(x, y) \bullet SE = \left(f(x, y) \oplus SE\right) \ominus SE$$

# Closing of the binary image



# Closing performed using 7 ×7 box-shaped structuring elements

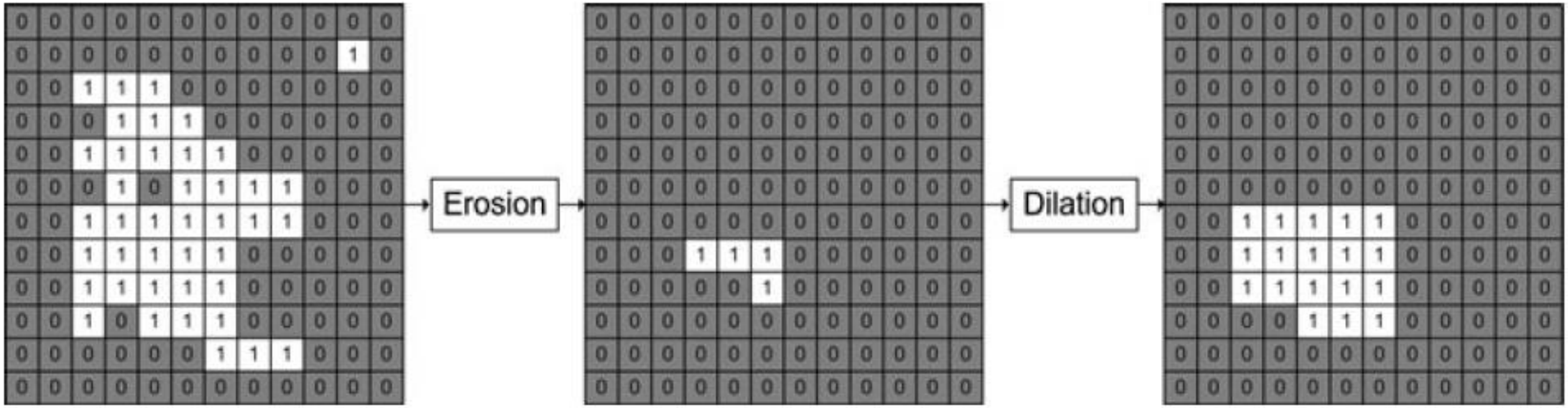


# Opening

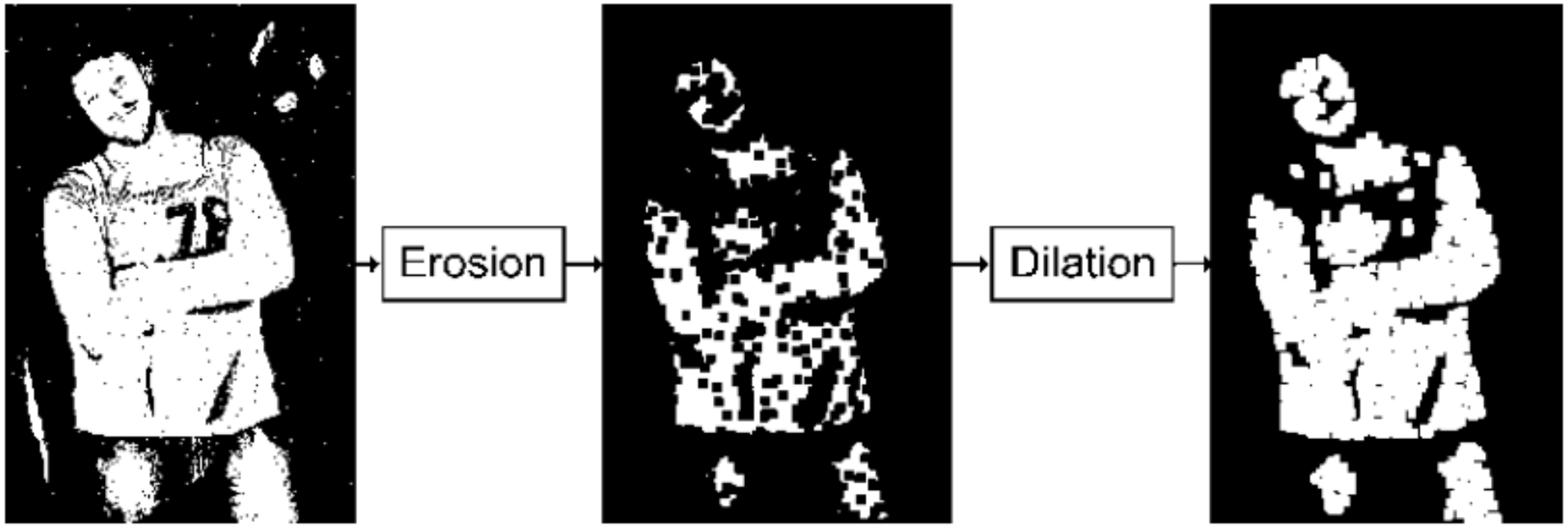
- Opening deals with the problem associated with erosion, namely that the objects decrease when we use erosion to erase small noisy objects or fractured parts of bigger objects. The decreasing object size is a problem in situations where for example the size of the object (number of pixels) matters. The solution to this problem is luckily straight forward, we simply enlarge the object by following the erosion by a dilation. This operation is denoted Opening and is written as:

$$g(x, y) = f(x, y) \circ SE = \left(f(x, y) \ominus SE\right) \oplus SE$$

# Opening of the binary image

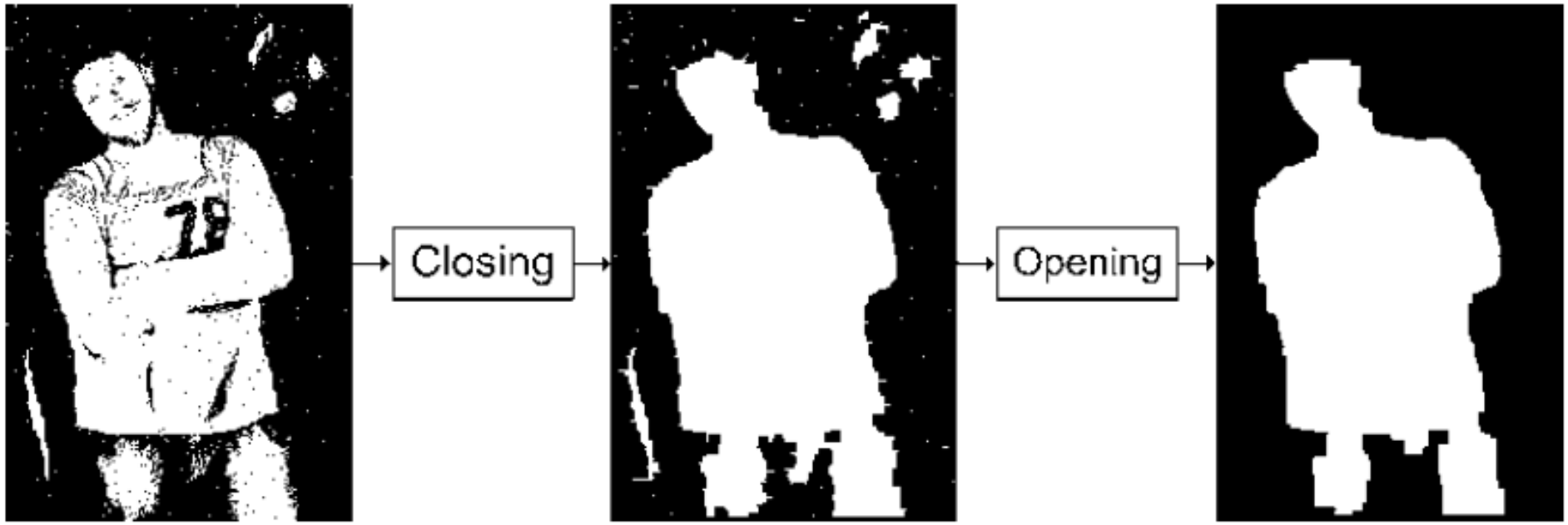


# Opening performed using a 7×7 box-shaped structuring element



# Combining Opening and Closing

In some situations we need to apply both opening and closing to an image. For example in cases where we both have holes inside the main object and small noisy objects.



# Boundary detection

- Doing edge detection in binary images is normally referred to as boundary detection and can be performed as described before. Morphology offers an alternative approach for binary images. The idea is to use erosion to make a smaller version of the object. By subtracting this from the input image only the difference stands out, namely the boundary:

$$g(x, y) = f(x, y) - \left(f(x, y) \ominus SE\right)$$

# Boundary detection - result

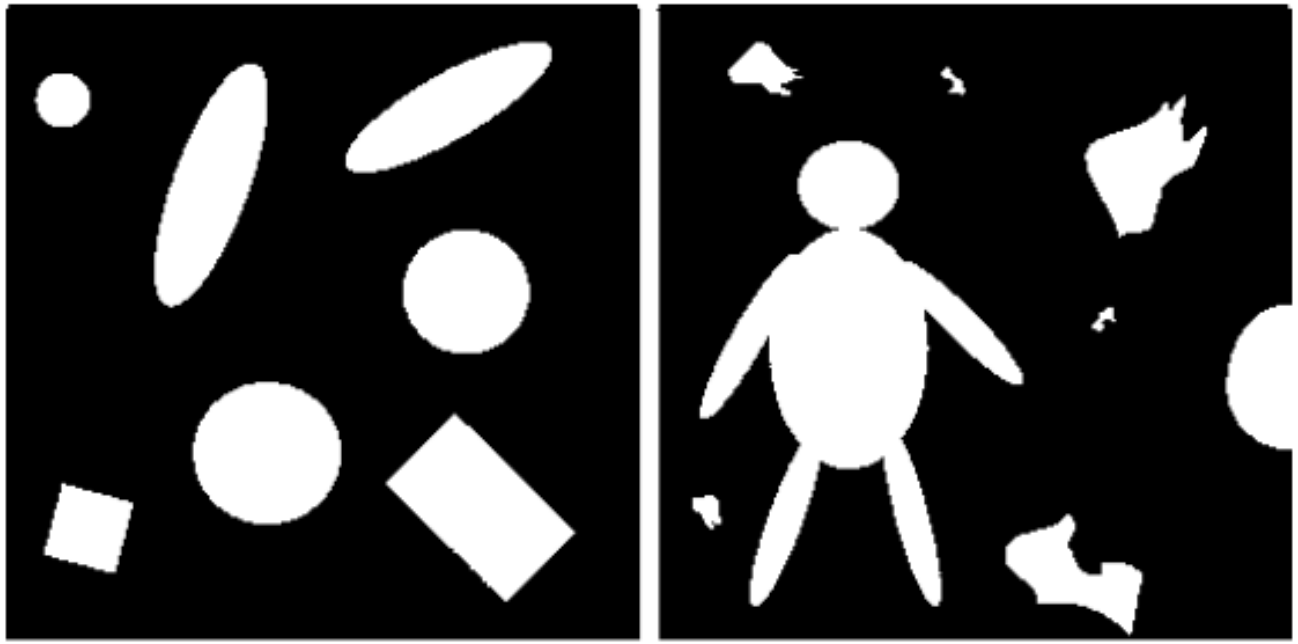
# Boundary detection - result

- If the task is only to locate the outer boundary, then the internal holes should first be filled using dilation or closing.

# BLOB Analysis

- **BLOB** stands for **B**inary **L**arge **Ob**ject and refers to a group of connected pixels in a binary image.
- The term refers to analyzing binary images by first extracting the BLOBs, then representing them compactly, and finally classifying the type of each BLOB.

# Examples of BLOBs



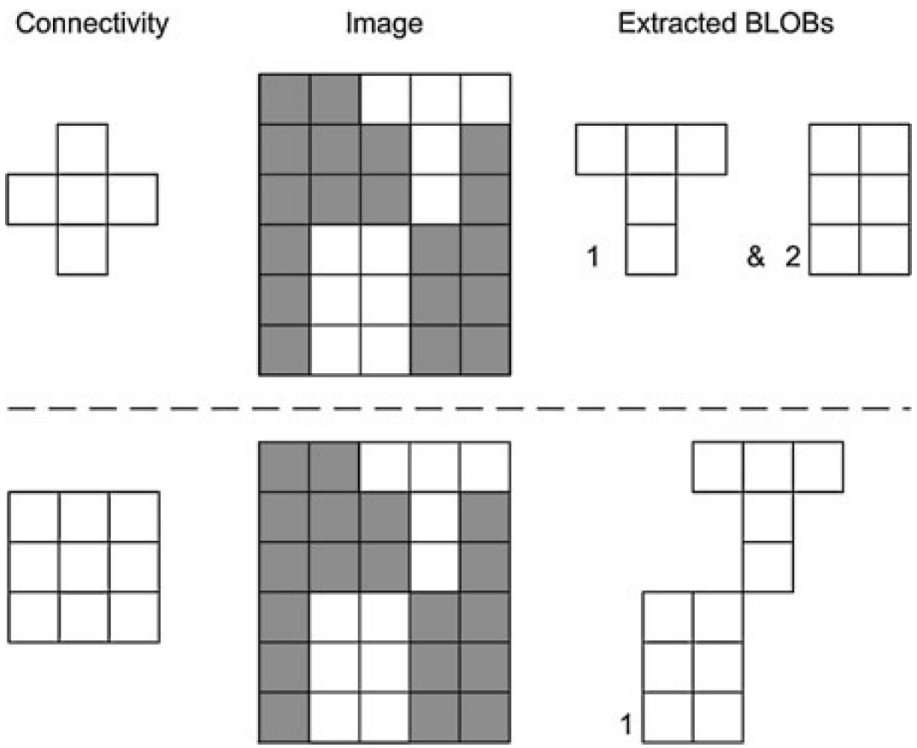
# BLOB extraction

The purpose of BLOB extraction is to isolate the BLOBs (objects) in a binary image. As mentioned above, a BLOB consists of a group of connected pixels. Whether or not two pixels are connected is defined by the connectivity, that is, which pixels are neighbors and which are not.

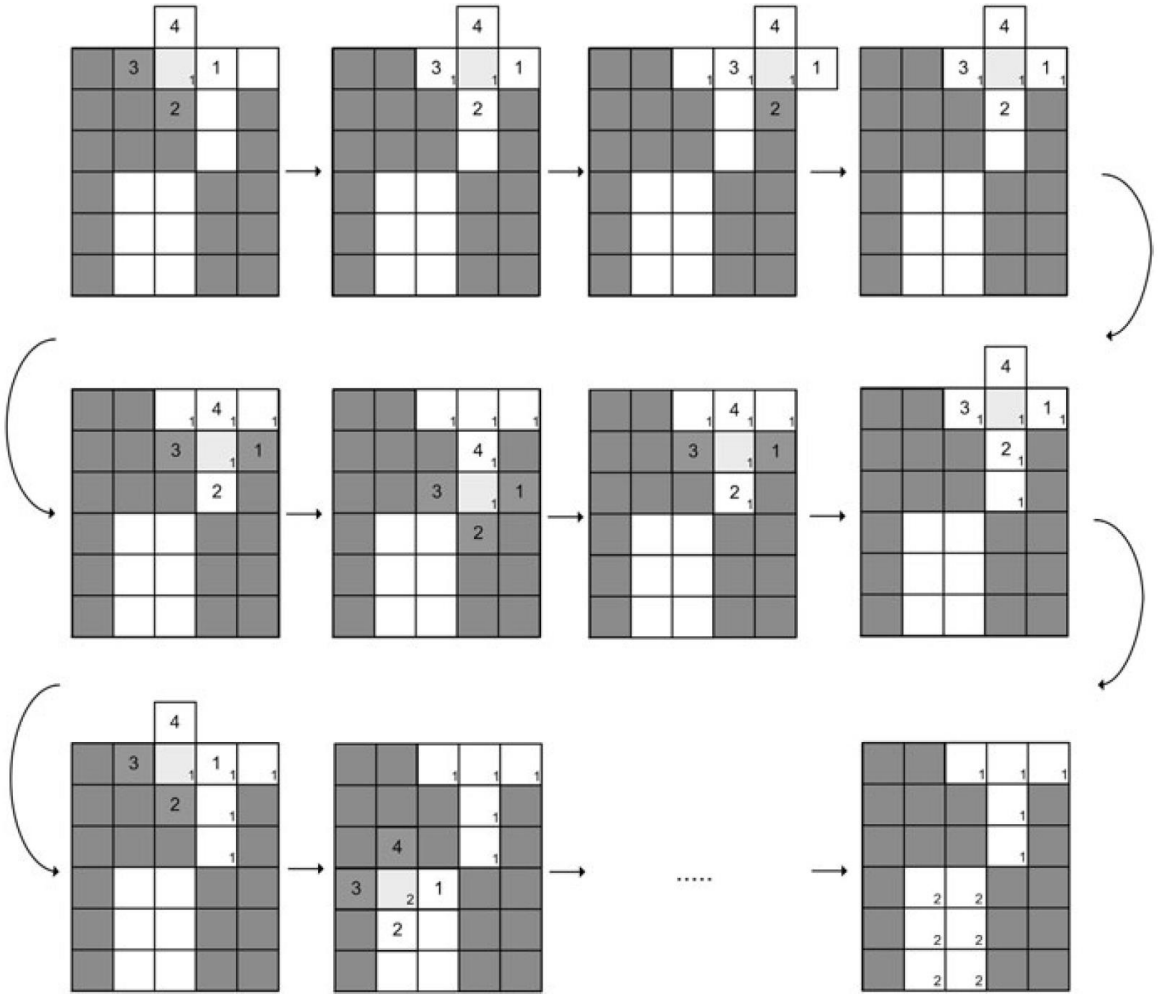
Among possible algorithms Grass-fire algorithm will be presented.

# Connectivity

- 4- and 8- connectivity

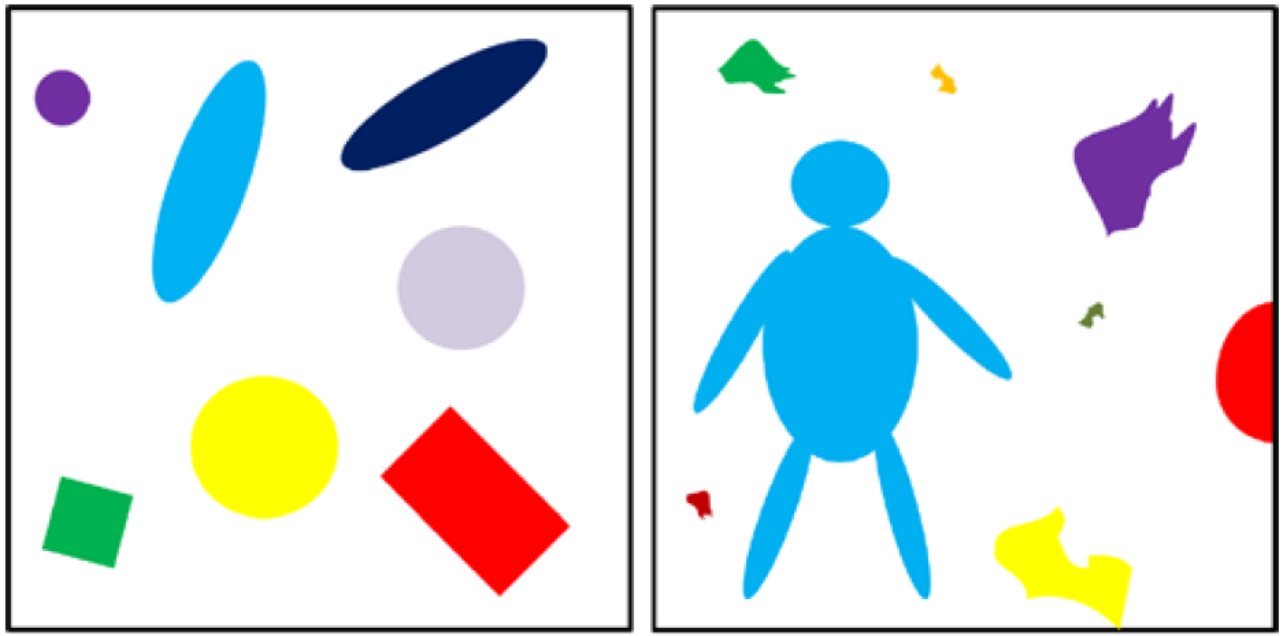


# Grass - fire algorithm



# Two examples of extracted BLOBs

- Each BLOB has a unique color



# BLOB Features

Extracting the BLOBs is the first step when confronted with examples like those presented above. The next step is to classify the different BLOBs. For the example with the circles, we want to classify each BLOB as either a circle or no a circle, and, for the other example, human vs. non-human BLOBs. The classification process consists of two steps. First, each BLOB is represented by a number of characteristics, denoted **features**, and second, some matching method is applied to compare the features of each BLOB with the features of the type of object we are looking for. For example, to find circles we could calculate the circularity of each BLOB and compare that to the circularity of a perfect circle.

# Typical features - Area

- **Area** of a BLOB is the number of pixels the BLOB consists of. This feature is often used to remove BLOBs that are too small or too big from the image. For example, in figure below the human can be segmented by simply saying that all BLOBs with an area smaller than a certain value are ignored.

# Typical features – Bounding box

- **Bounding box** of a BLOB is the minimum rectangle which contains the BLOB, see figure below. It is defined by going through all pixels for a BLOB and finding the four pixels with the minimum x-value, maximum x-value, minimum y-value and maximum y-value, respectively. From these values the width of the bounding box is given as $x_{max} - x_{min}$ and the height as $y_{max} - y_{min}$. A bounding box can be used as a ROI.

# Typical features – circle

- **Bounding circle** of a BLOB is the minimum circle which contains the BLOB, see figure below. It is found by first locating the center of the BLOB with one of the methods described below. Next we search from the center and outwards in one direction until we find the point where the BLOB ends. The distance between this point and the center is the radius in this direction. We do this for all possible directions (for example with an angular resolution of 10°) and the biggest radius defines the radius for the minimum circle.

# Typical features – covex hull

- **Convex hull** of a BLOB is the minimum convex polygon which contains the BLOB, see figure below. It corresponds to placing a rubber band around the BLOB. It can be found in the following manner. From the topmost pixel on the BLOB search to the right along a horizontal line. If no BLOB pixel is found increase (clockwise) the angle of the search line and repeat the search. When a BLOB pixel is found the first line of the polygon is defined and a new search is started based on the angle of the previous search line. When the search reappears at the topmost pixel, the convex hull is completed.

# Typical features - Bounding box ratio

- **Bounding box ratio** of a BLOB is defined as the height of the bounding box divided by the width. This feature indicates the elongation of the BLOB, i.e., is the BLOB long, high or neither.

# Typical features - Compactness

- **Compactness** of a BLOB is defined as the ratio of the BLOB's area to the area of the bounding box. This can be used to distinguish compact BLOBs from noncompact ones. For example, fist vs. a hand with outstretched fingers.

$$\text{Compactness} = \frac{\text{Area of BLOB}}{\text{width} \cdot \text{height}}$$

# Typical features - Center of mass

- **Center of mass** (or center of gravity or centroid) of a physical object is the location on the object where you should place your finger in order to balance the object. The center of mass for a binary image is similar. It is the average x- and y-positions of the binary object. It is defined as a point, whose x-value is calculated by summing the x-coordinates of all pixels in the BLOB and then dividing by the total number of pixels. Similarly for the y-value. In mathematical terms the center of mass, $(x_c, y_c)$ is calculated as where N is the number of pixels in the BLOB and $x_i$ and $y_i$ are the x and y coordinates of the N pixels, respectively.

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad y_c = \frac{1}{N}\sum_{i=1}^{N} y_i$$

# Typical features - Center of the bounding box

- Center of the bounding box is a fast approximation of the center of mass. In mathematical terms the center of the bounding box, ($x_{bb}$, $y_{bb}$) is calculated as:

$$x_{bb} = x_{\min} + \frac{x_{\max} - x_{\min}}{2} = x_{\min} + \frac{x_{\max}}{2} - \frac{x_{\min}}{2} = \frac{x_{\min} + x_{\max}}{2}$$

$$y_{bb} = y_{\min} + \frac{y_{\max} - y_{\min}}{2} = y_{\min} + \frac{y_{\max}}{2} - \frac{y_{\min}}{2} = \frac{y_{\min} + y_{\max}}{2}$$

# Typical features - Perimeter

- **Perimeter** of a BLOB is the length of the contour of the BLOB. This can be found by scanning along the rim (contour) of an object and summing the number of pixels encountered. A simple approximation of the perimeter is to first find the outer boundary using the method described before (or any another edge detection algorithm). Following this we simply count the number of white pixels in the image.

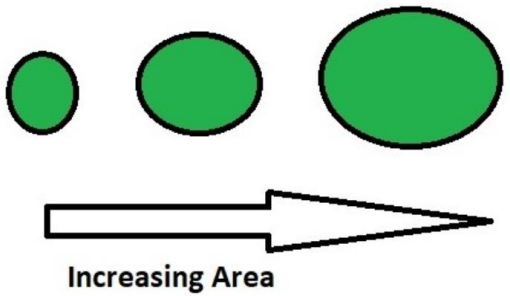
# Typical features - Circularity

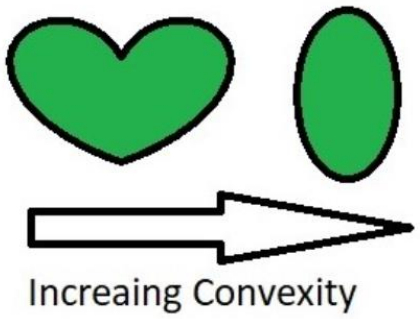
- **Circularity** of a BLOB defines how circular a BLOB is. Different definitions exist based on the perimeter and area of the BLOB. Heywood's circularity factor is, for example, defined as the ratio of the BLOB's perimeter to the perimeter of the circle with the same area:

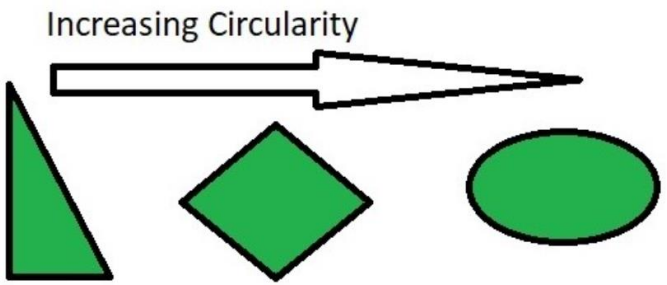
$$\text{Circularity} = \frac{\text{Perimeter of BLOB}}{2\sqrt{\pi \cdot \text{Area of BLOB}}}$$

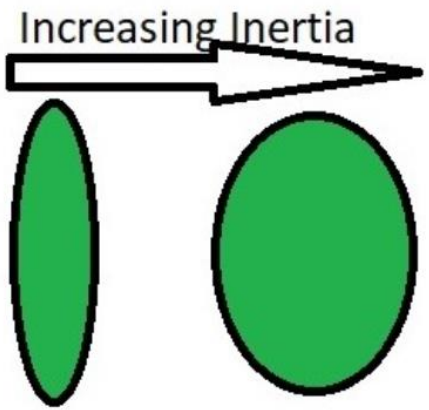
- Sometimes: $f_{\text{circ}} = \frac{4\pi A}{P^2}$

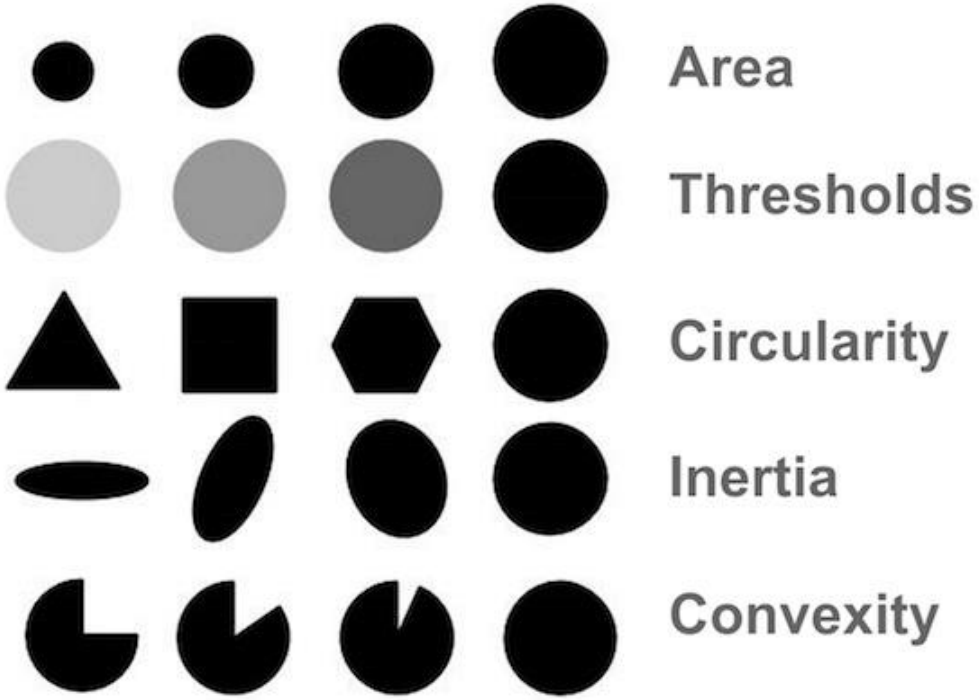
# Comparison


Increasing Area


Increaing Convexity

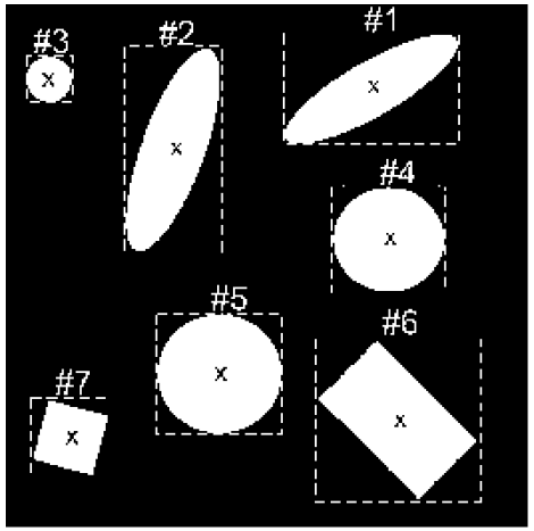

Increasing Circularity


Increasing Inertia


Area
Thresholds
Circularity
Inertia
Convexity

# Feature vector

- After extraction of features a binary image has been converted into a number of feature values for each BLOB. The feature values can be collected in a so-called **feature vector**.




| BLOB number | Circu-larity | Area (pixels) |
|---|---|---|
| 1 | 0.31 | 6561 |
| 2 | 0.40 | 6544 |
| 3 | 0.98 | 890 |
| 4 | 0.97 | 6607 |
| 5 | 0.99 | 6730 |
| 6 | 0.52 | 6611 |
| 7 | 0.75 | 2073 |

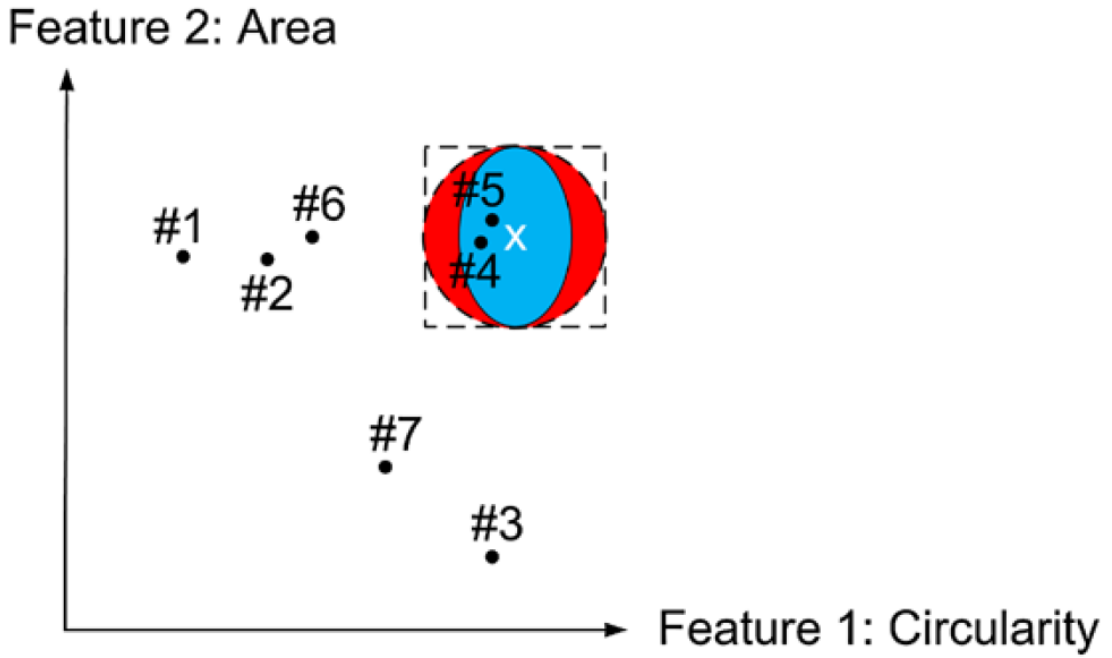
$$\vec{f}_1 = \begin{bmatrix} 0.31 \\ 6561 \end{bmatrix}$$

# BLOB Classification - 1

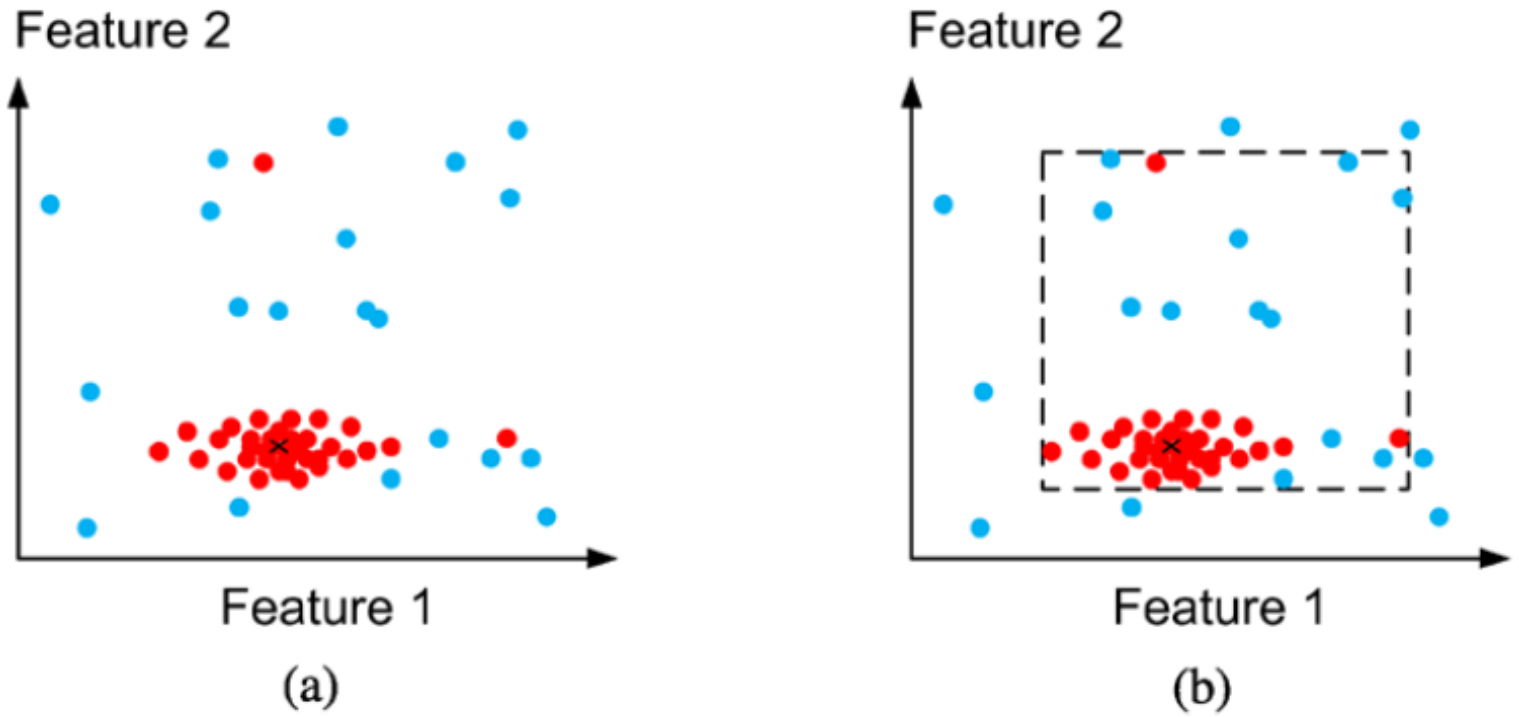
Let the task now be to determine which BLOB is a circle and which is not. As suggested above we can use the circularity feature for this purpose. In previous figure the values of the circularity of the different BLOBs are listed. The question is now how to define which BLOBs are circles and which are not based on their feature values. For this purpose we make a prototype model of the object we are looking for. That is, what are the feature values of a perfect circle and what kind of deviation will we accept? In a perfect world we will not accept any deviations from the prototype, but in practice the object or the segmentation of the object will not be perfect so we have to accept some deviations. For our example with the circles, we can define the prototype to have a circularity of 1 and a deviation of ±0.15, meaning that BLOBs with circularity values in the range [0.85, 1.15] are accepted as circles.

# BLOB Classification - 2

What if we are looking for large circles? For this task one feature is not sufficient and we therefore use both the circularity and the area. These two features span a 2-dimensional **feature space** as seen in the figure. The prototype model will now be two dimensional with each feature having an allowed range. Together these two ranges will form a rectangle and if a BLOB in an image has feature values inside the dashed rectangle, then it is a large circle otherwise it is not, see figure. This approach is known as a **box classifier** and the area of the rectangle is known as the **decision region**.



# Outlier problem



# Statistical classification

- A better approach is to express the spreading of the points using the variance. When we have the means and variances of the different features the box classifier should be replaced by a statistical classifier since this is a more accurate approach. In the box classifier we have a binary decision; is a new feature vector (BLOB) inside or outside the rectangle? In a statistical classifier we instead measure a distance between a new feature vector (BLOB) and the prototype.

# Weighted Euclidean distance - WED

$$\mathrm{WED}(\vec{f_i}, \text{prototype}) = \sqrt{\frac{(f_i(\text{cir}) - \text{mean}(\text{cir}))^2}{\text{variance}(\text{cir})} + \frac{(f_i(\text{area}) - \text{mean}(\text{area}))^2}{\text{variance}(\text{area})}}$$

- In general case

$$\mathrm{WED}(\vec{f_i}, \text{prototype}) = \sum_{j=1}^{p} \sqrt{\frac{(f_i(m_j) - \text{mean}(m_j))^2}{\text{variance}(m_j)}}$$

- It should be noticed that the three equations above assume that the scale of the features are the same. In our example the problem is that the area is measured in 1000 s and circularity is a value close to 1. This means that the area will dominate the distance measure completely. The solution is to normalize the features so they are scaled similarly and are in the same interval, e.g. [0, 1].