

Image thresholding and feature extraction

Johannes Müller

With material from

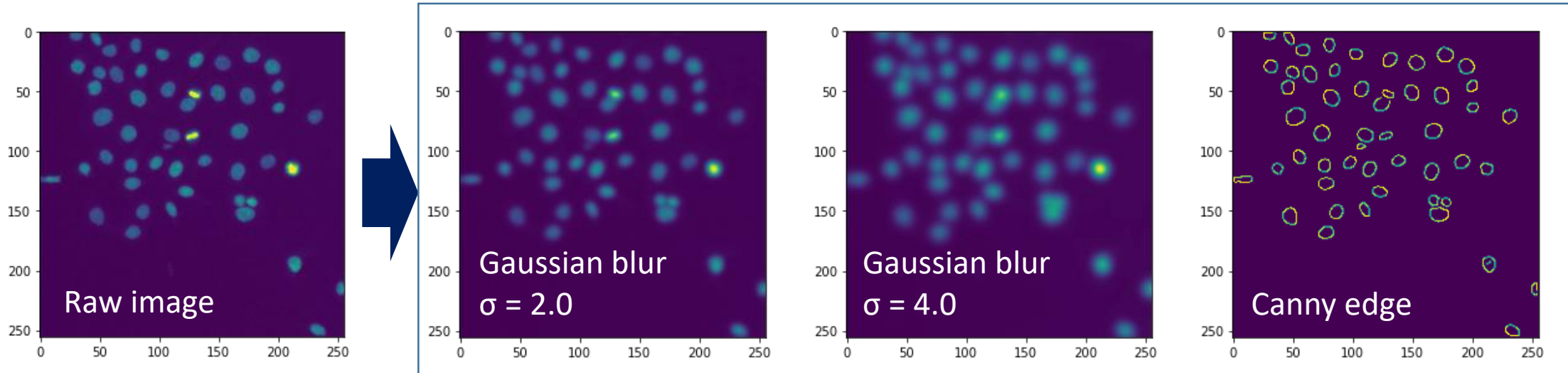
Robert Haase, BiaPoL, PoL TU Dresden

Marcelo Zoccoler, BiaPol, PoL, TU Dresden

Benoit Lombardot, Scientific Computing Facility, MPI CBG

May 2022

- Image filtering

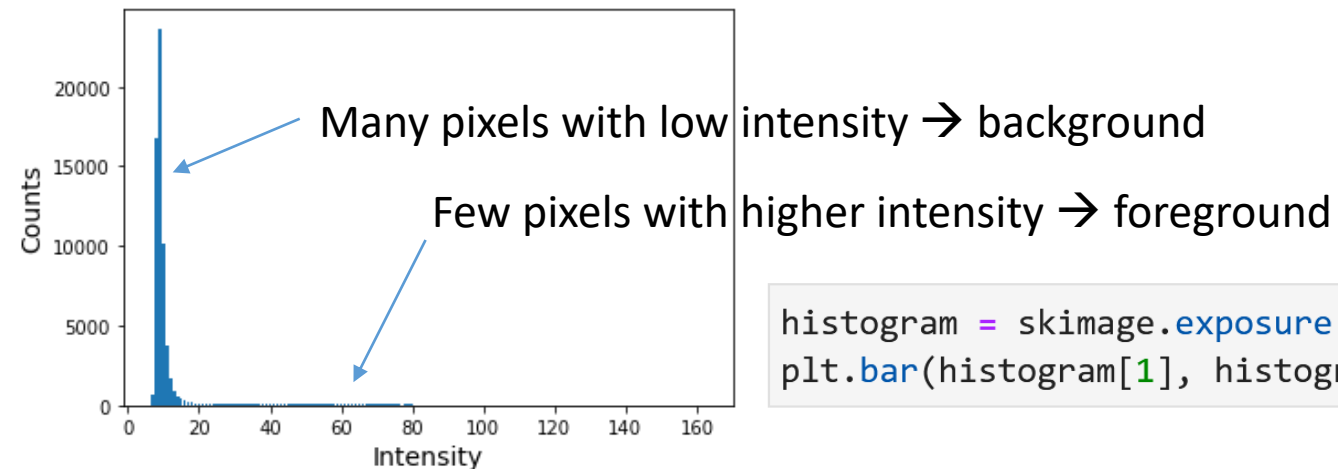


```
from skimage import filters, feature
```

<https://scikit-image.org/docs/stable/api/skimage.filters.html>
<https://scikit-image.org/docs/stable/api/skimage.feature.html>

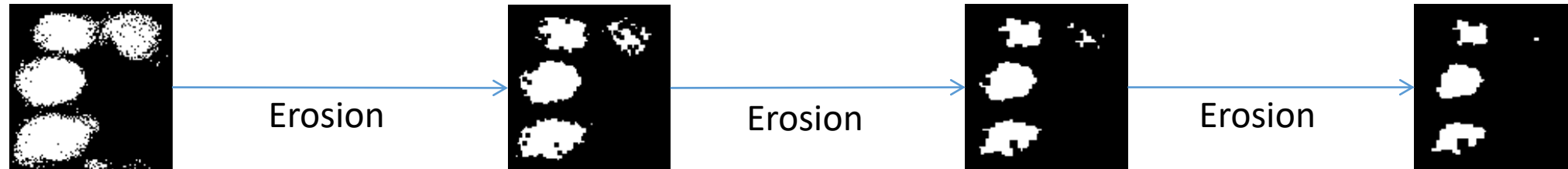
- Histograms

Count number of pixels within specific ranges ("bins") of intensity

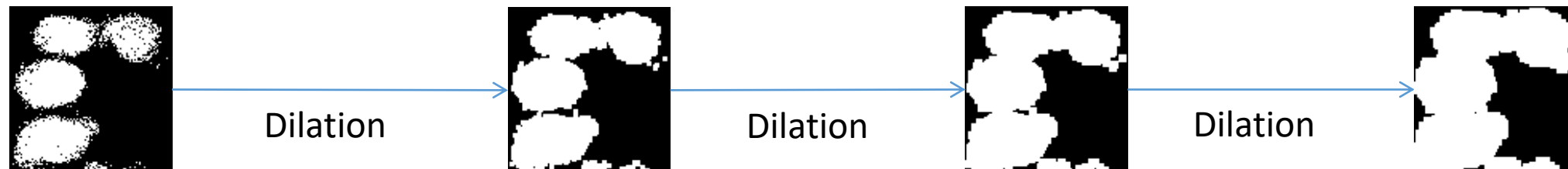


```
histogram = skimage.exposure.histogram(image)  
plt.bar(histogram[1], histogram[0])
```

- Erosion: Set all pixels to black which have at least one black neighbor.



- Dilation: Set all pixels to white which have at least one white neighbor.



- Closing: Dilation + Erosion



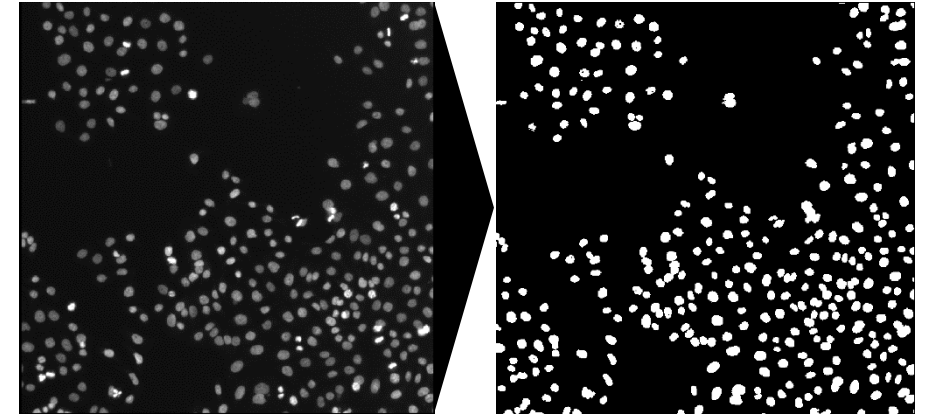
- Opening: Erosion + Dilation

Today's content:

Complete workflow from raw images toward extracting quantitative features from image data

- Creating binary masks from intensity images (**Segmentation**)

→ Thresholding



- Create label images from binary masks

→ Connected-component analysis

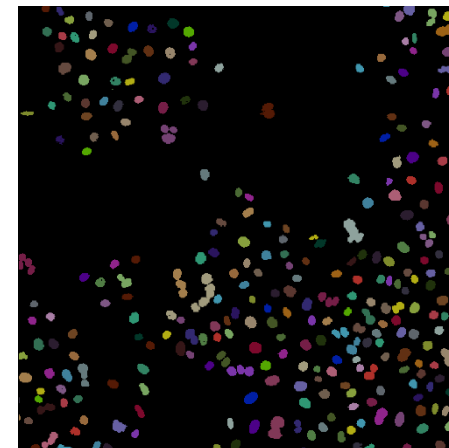
→ Watershed algorithm

- Measure features from label images

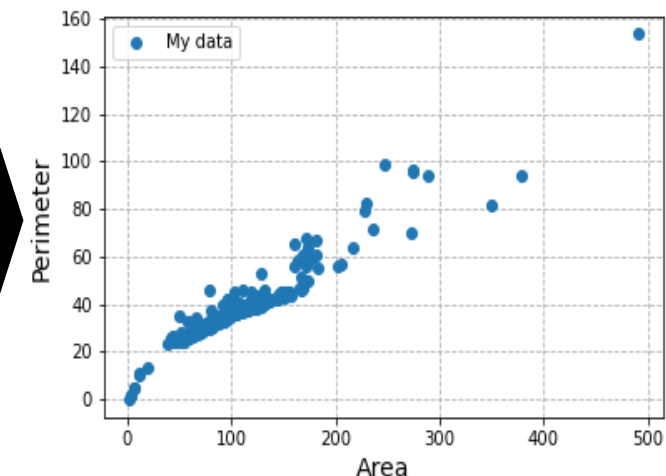
→ Shape, Intensity, etc.

→ Visualize measured features

Labelled image



Measurement

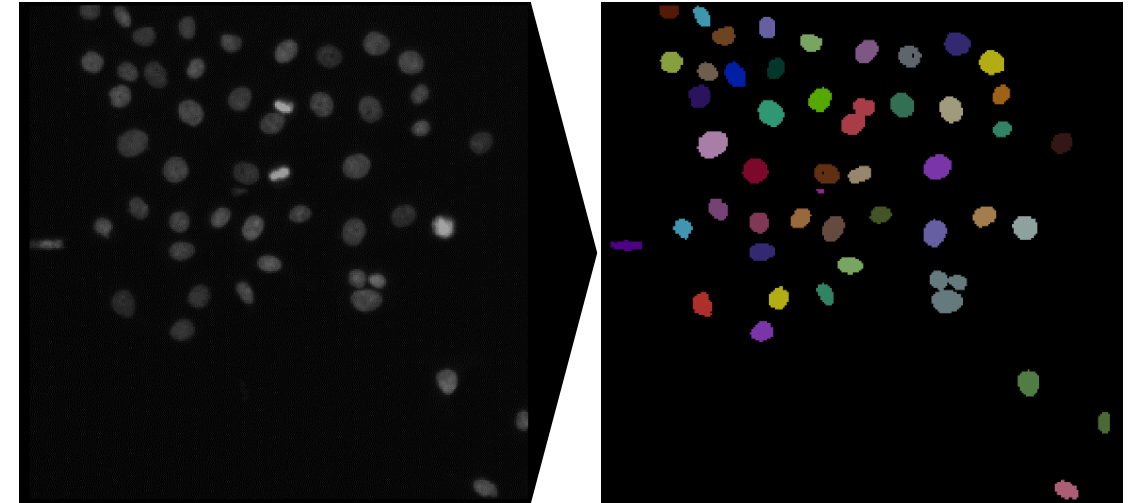


Aim:

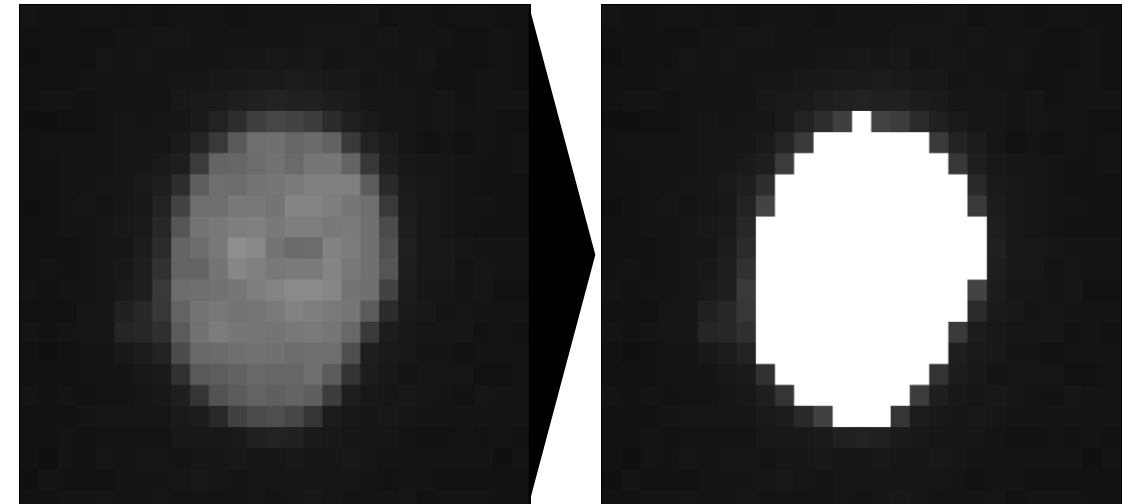
Separate background from foreground

Vocabulary:

- **Segmentation:**
Assigning a meaningful *label* to each pixel
- **Semantic segmentation:**
Differentiate pixels into multiple *classes* (e.g., membrane, nucleus, cytosol, etc.)
- **Instance segmentation:**
Differentiate multiple occurrences of the same class into separate instances of this class (e.g., separate *label* for each cell in image)



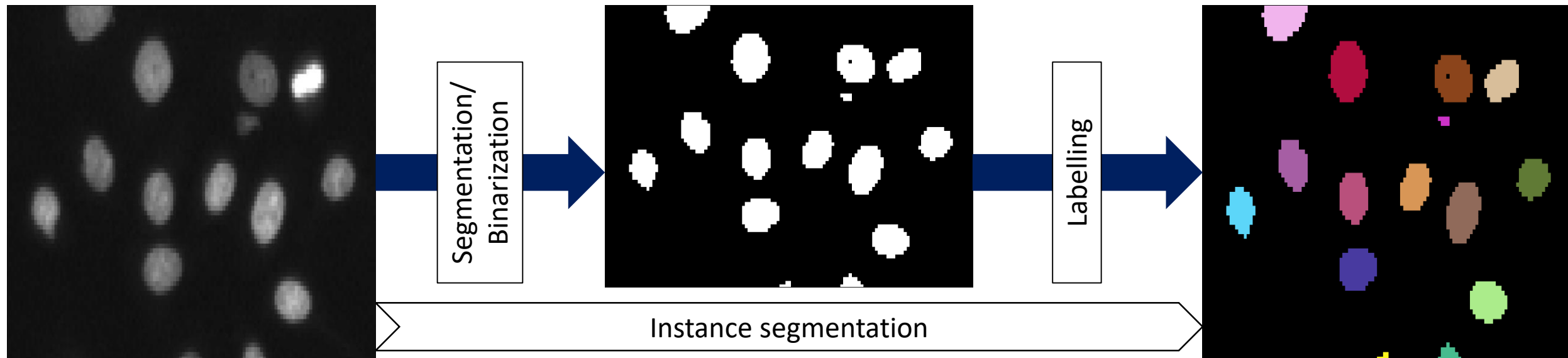
Instance segmentation



Semantic segmentation

Analyzing properties (*features*) of individual objects in images requires instance segmentation

- Methods
 - Thresholding + connected components labeling
 - Spot detection + seeded watershed
 - Edge detection based
 - Machine learning



- Applying a threshold to an image requires to compare every pixel to the threshold value
- We can compare values in Python with:

```
a = 5  
b = 6  
print(a > b)  
print(a < b)  
print(a == b)
```



```
image > threshold
```

```
array([[False, False, False, ..., False, False, False],  
       [False, False, False, ..., False, False, False],  
       [False, False, False, ..., False, False, False],  
       ...,  
       [False, False, False, ..., False, False, False],  
       [False, False, False, ..., False, False, False],  
       [False, False, False, ..., False, False, False]])
```

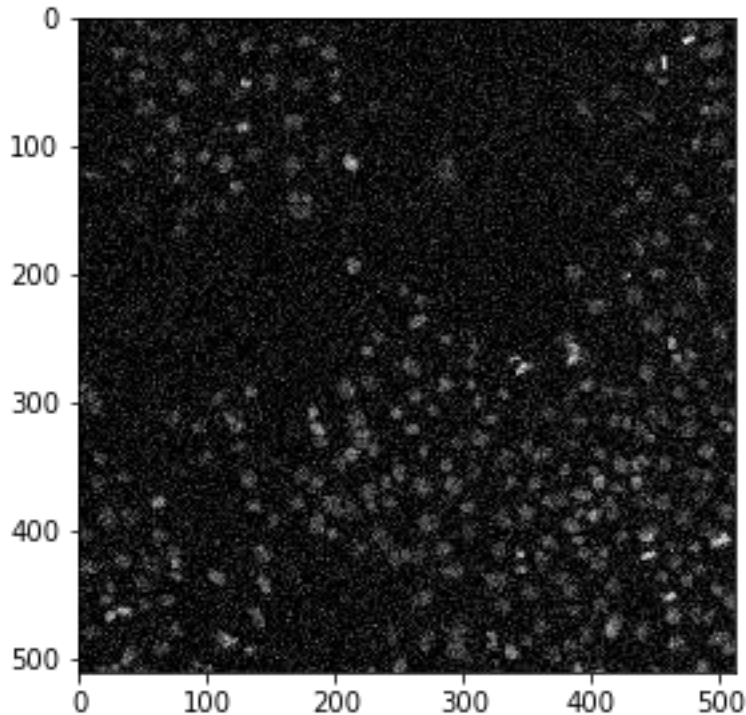
In this case, “image” is a *numpy array* → some operations are automatically applied to every pixel!

- We can then simply store the output of this element-wise comparison in a new variable:

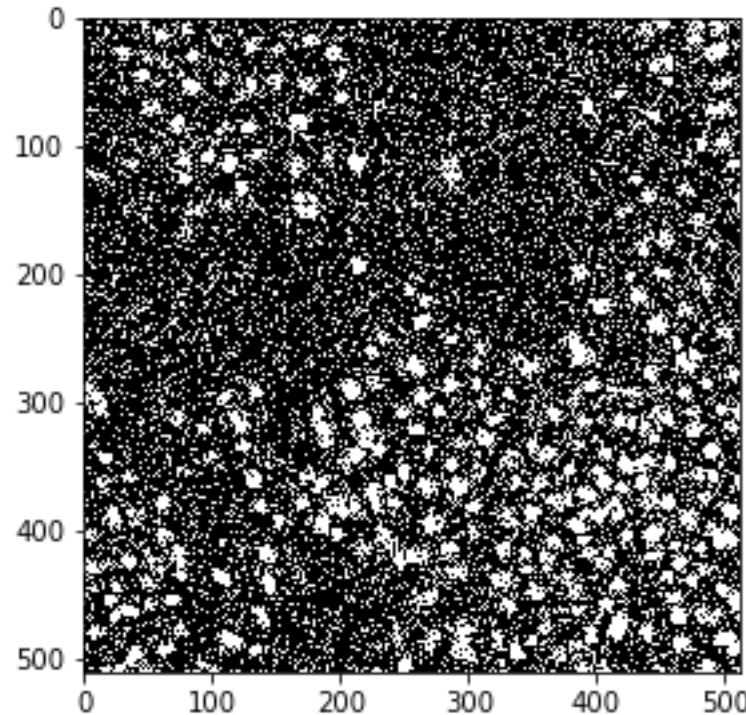
```
binary = image > threshold
```


Reminder: pre-processing!

- Before we can create masks, we need to pre-process images.
 - Noise removal
 - Background subtraction



Noisy image



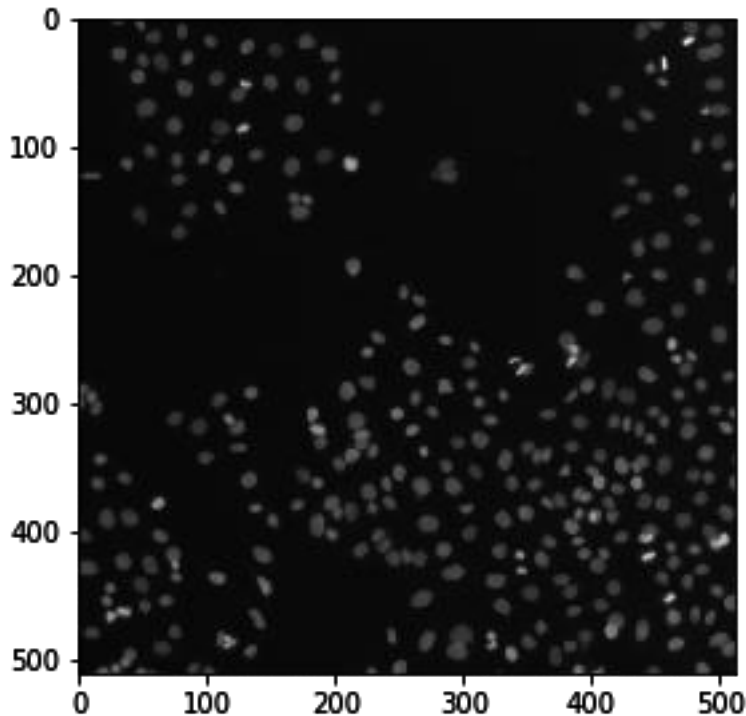
Thresholded image

```
filtered = filters.median(image)
```

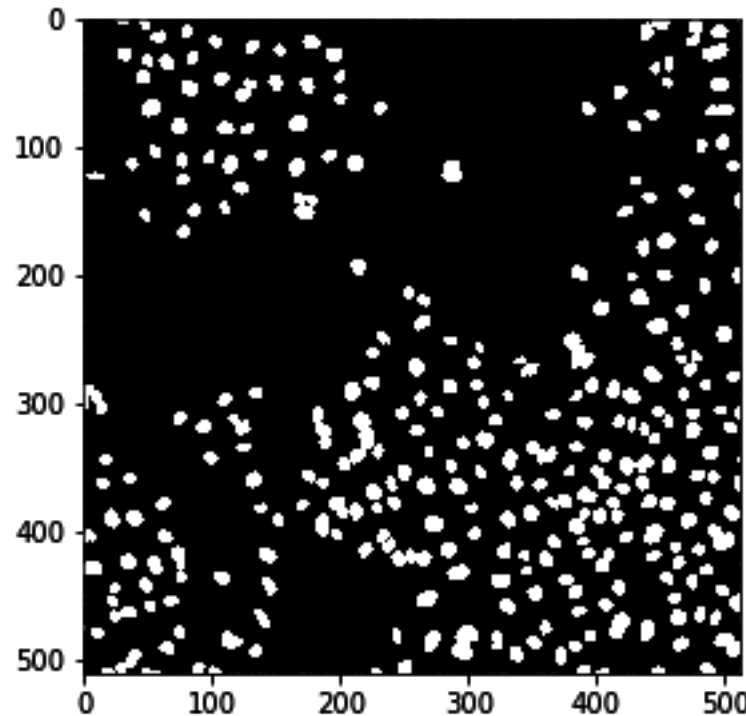
Image filtering *filters* relevant information for subsequent operations from the image!

Reminder: pre-processing!

- Before we can create masks, we need to pre-process images.
 - Noise removal
 - Background subtraction



Filtered image



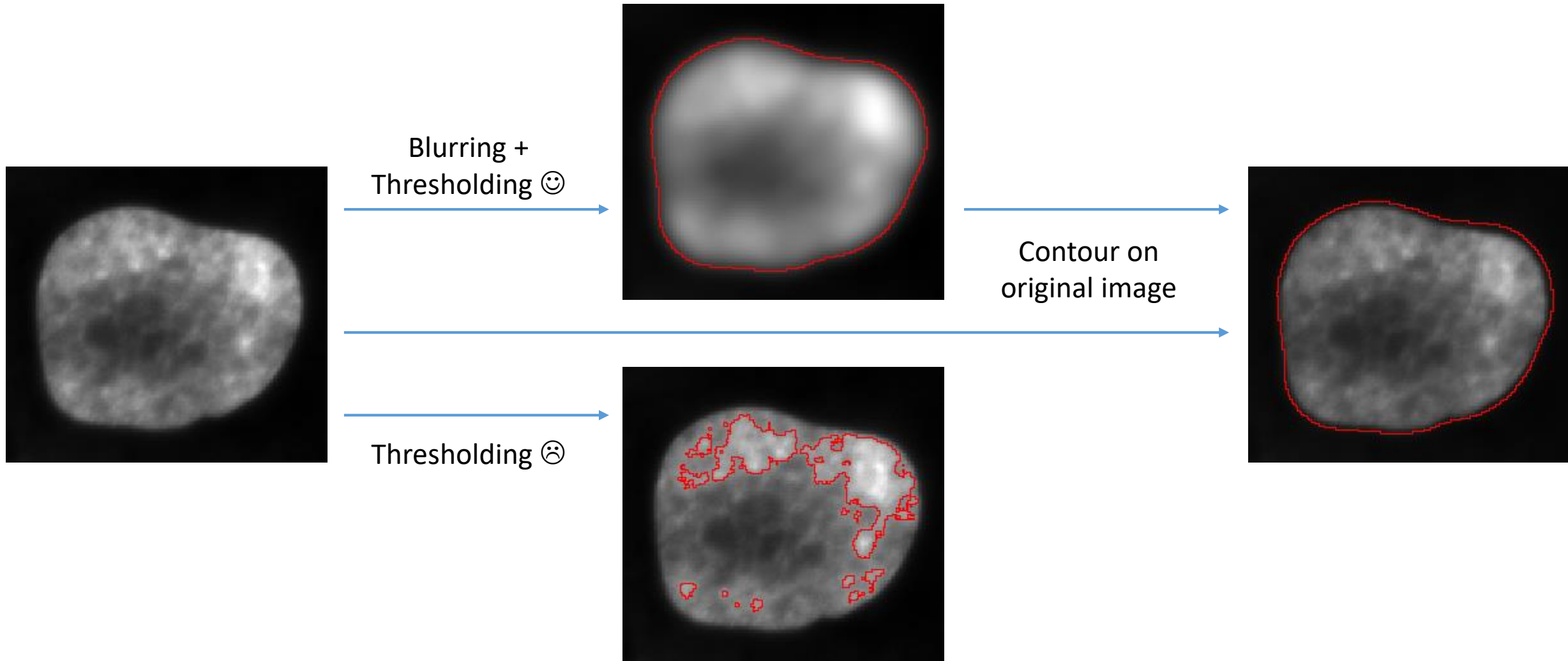
Thresholded image

```
filtered = filters.median(image)
```

Image filtering *filters* relevant information for subsequent operations from the image!

Low-pass filtering to improve thresholding results

- In case thresholding algorithms outline the wrong structure, blurring in advance may help.
- However: **Do not** continue processing the blurred image, continue with the original!

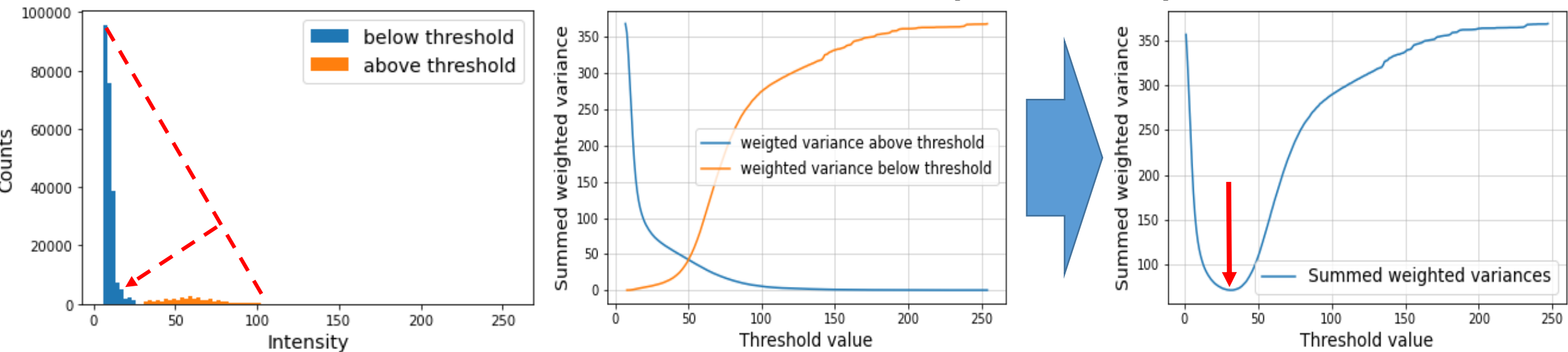


- **Otsu-thresholding** (Otsu et Al. 1979): Find threshold so that the summed, weighted variance $Var_{w,sum}$ becomes minimal:

$$Var(I) = \sum (I - mean(I))^2$$



$$Var_{w,sum} = \frac{n_A}{n_I} \cdot Var(A) + \frac{n_B}{n_I} \cdot Var(B)$$



- **Statistical thresholding:** Pixels above statistical parameter of I belong to foreground. (Possibilities: Mean, Median, Quartiles, etc.)
- **Triangle thresholding:** Draw a line between histogram point with max. counts and max. intensity and find point in histogram with maximal distance to this line (----)

```
threshold = filters.threshold_otsu(image)
```

- **Otsu-thresholding** (Otsu et Al. 1979): Find threshold so that the summed, weighted variance $Var_{w,sum}$ becomes minimal.

```
threshold = filters.threshold_mean(image)
```

- **Statistical thresholding**: Pixels above statistical parameter of I belong to foreground.
(Possibilities: Mean, Median, Quartiles, etc.)

```
threshold = filters.threshold_triangle(image)
```

- **Triangle thresholding**: Draw a line between histogram point with max. counts and max. intensity and find point in histogram with maximal distance to this line.

Explore more threshold options in scikit-image with:

```
from skimage import filters
```

```
threshold = filters.threshold_
```

f	threshold_isodata	function
f	threshold_li	function
f	threshold_local	function
f	threshold_mean	function
f	threshold_minimum	function
f	threshold_multiotsu	function
f	threshold_niblack	function
f	threshold_otsu	function
f	threshold_sauvola	function
f	threshold_triangle	function

- Cite the thresholding method of your choice properly

We segmented the cell nuclei in the images using the Otsu thresholding method (Otsu et Al. 1979) implemented in scikit-image (van der Walt et Al. 2014).

IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. SMC-9, NO. 1, JANUARY 1979

A Threshold Selection Method from Gray-Level Histograms

NOBUYUKI OTSU

Abstract—A nonparametric and unsupervised method of automatic threshold selection for picture segmentation is presented. An optimal threshold is selected by the discriminant criterion, namely, so as to maximize the separability of the resultant classes in gray levels. The procedure is very simple, utilizing only the gray levels and the

```
binary = image > a_good_threshold_value_of_my_choice
```

Never use manual thresholding!

- Different observers come to different results when selecting a “good” threshold value
- You may come to different results when selecting a threshold value repeatedly

Inter-observer
variability

```
binary = image > threshold  
intensities = some_function_to_measure_intensities(binary, image)
```

Intra-observer
variability

Avoid thresholding an image and afterwards measure intensities in the same image

- You would measure the threshold you entered

```
binary_1 = image_1 > threshold_1(image_1)  
binary_2 = image_2 > threshold_2(image_2)
```

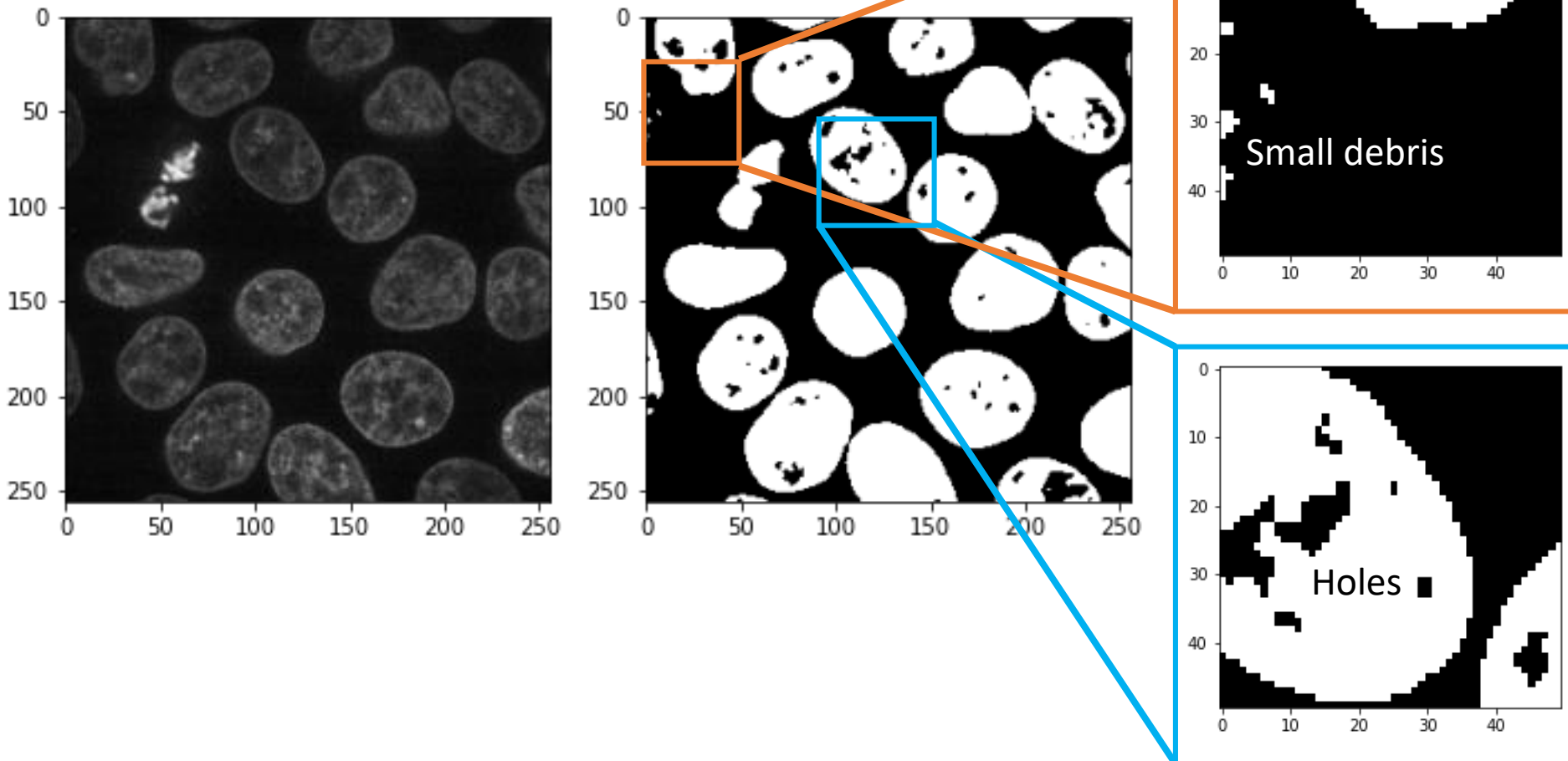
Chose one threshold algorithm:

...and stick to it for the whole study. Using a new method for every image impairs reproducibility!

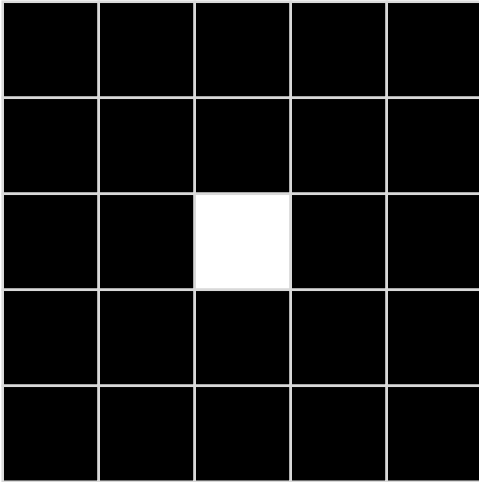
Do not over-engineer

There will be always images where thresholding fails – better report the errors!

- Binary mask images may not be perfect immediately after thresholding.
- There are ways of refining them

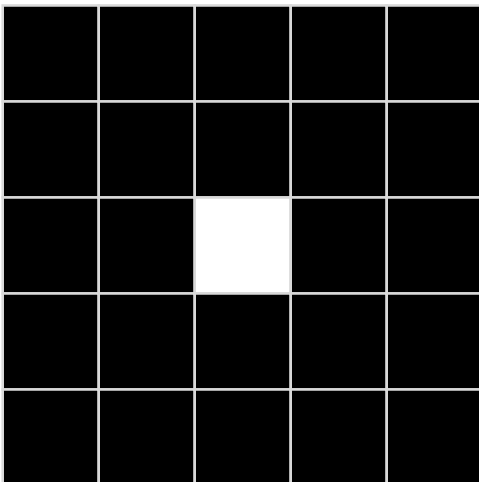
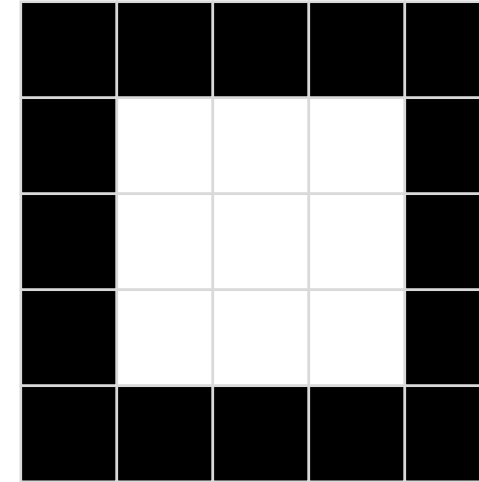


- Dilation: Every pixel with at least one white neighbor becomes white.



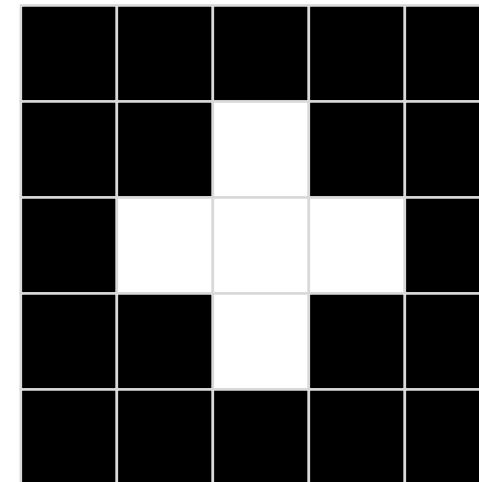
Dilation

8-connected neighborhood
Moore-Neighborhood



Dilation

4-connected neighborhood
von-Neumann-Neighborhood

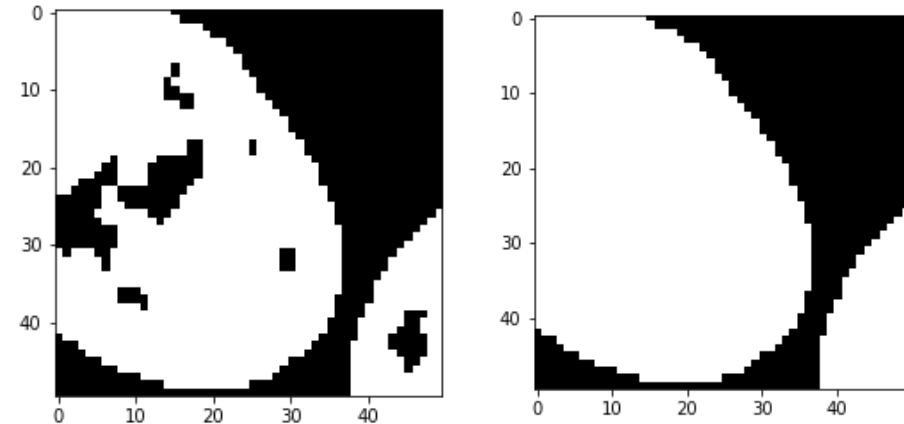
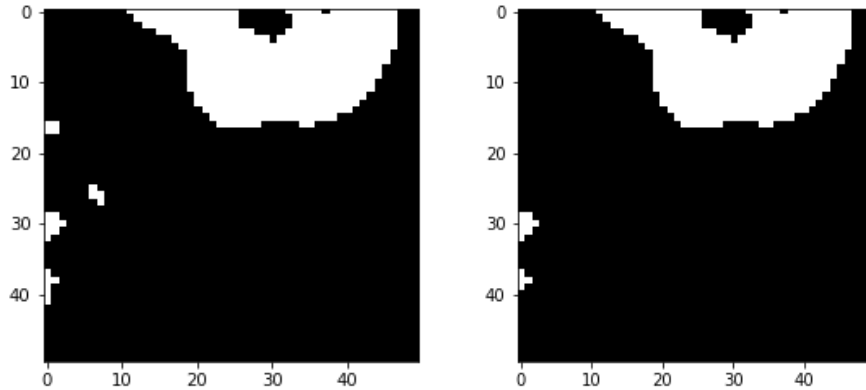


Refining masks: Opening & Closing

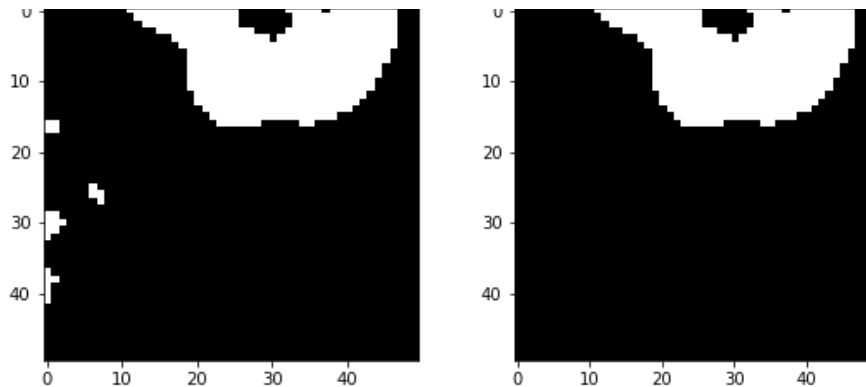
```
from skimage import morphology
```

```
closed = morphology.area_closing(binary, area_threshold=100)
```

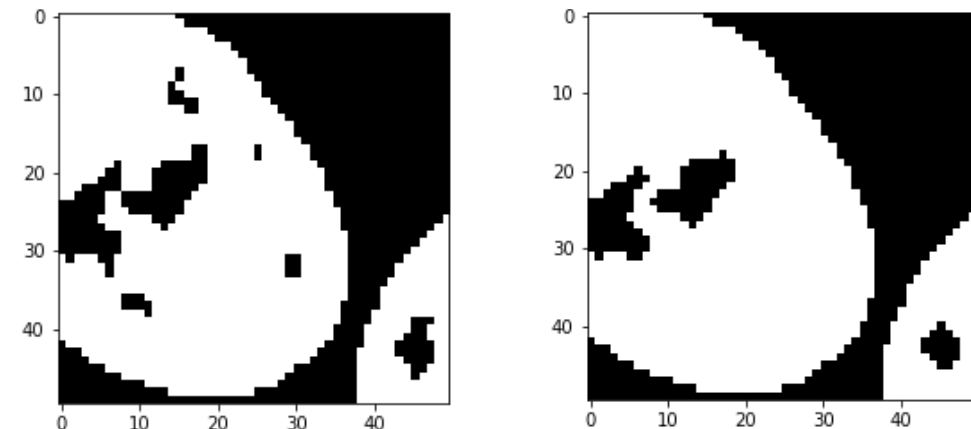
```
opened = morphology.binary_opening(binary)
```



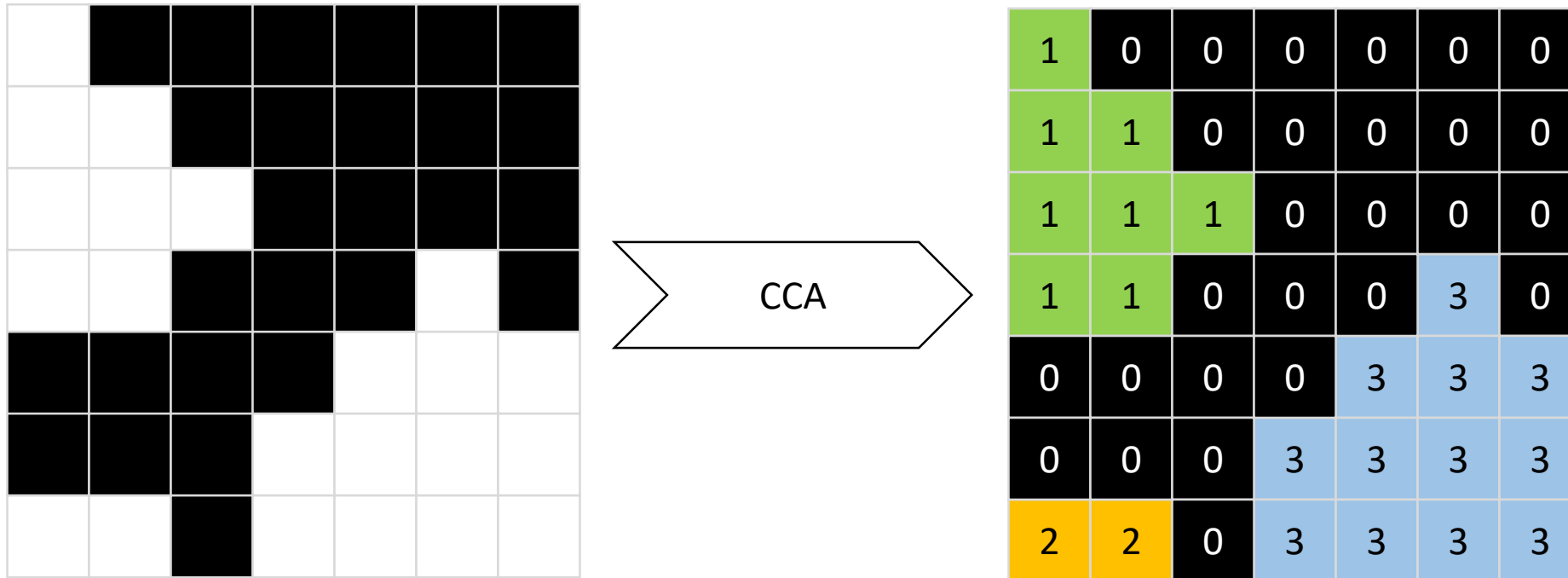
```
opened = morphology.area_opening(binary, area_threshold=20)
```



```
closed = morphology.binary_closing(binary)
```




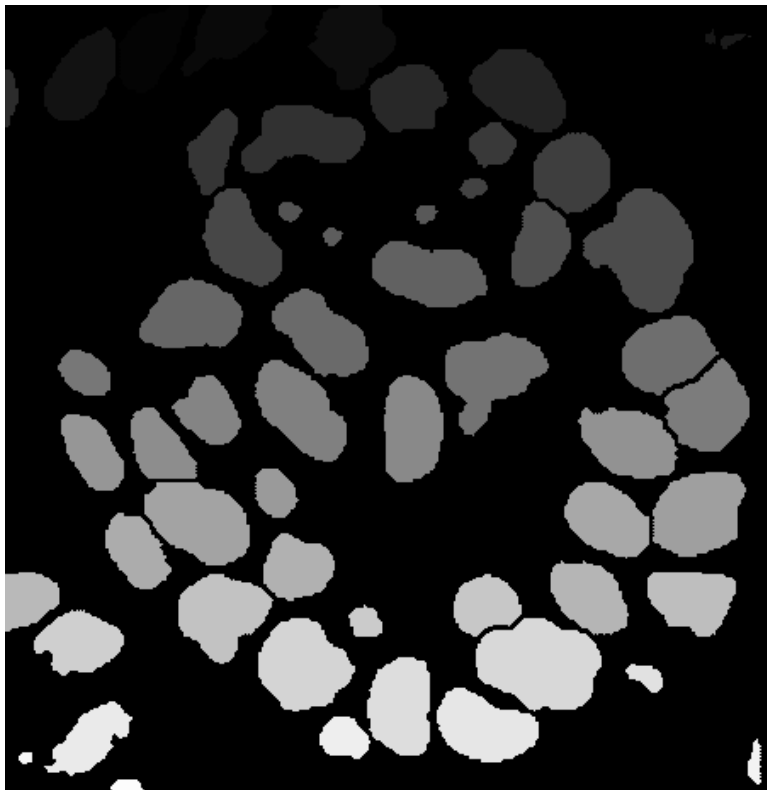
- In order to allow the computer differentiating objects, connected components analysis (CCA) is used to mark pixels belonging to different objects with different numbers
- Background pixels are marked with 0.
- The maximum intensity of a labelled map corresponds to the number of objects.




- Label maps can be nicely visualized with the right lookup table

Grey

Pixel value	Display color
0	
1	
2	
...	
255	



Glasbey

Pixel value	Display color
0	
1	
2	
...	
255	



1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	1	0	0	0	3	0
0	0	0	0	3	3	3
0	0	0	3	3	3	3
2	2	0	3	3	3	3

Case 1:

Label 1 = Nucleus #1

Label 2 = Nucleus #2

Label 3 = Nucleus #3



Instance segmentation

Case 2:

Label 1 = Nucleus

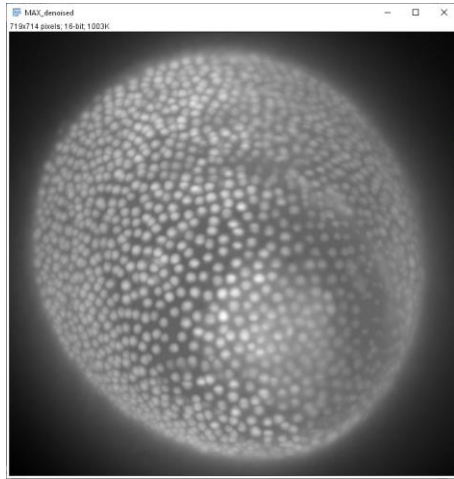
Label 2 = Cytosol

Label 3 = Membrane

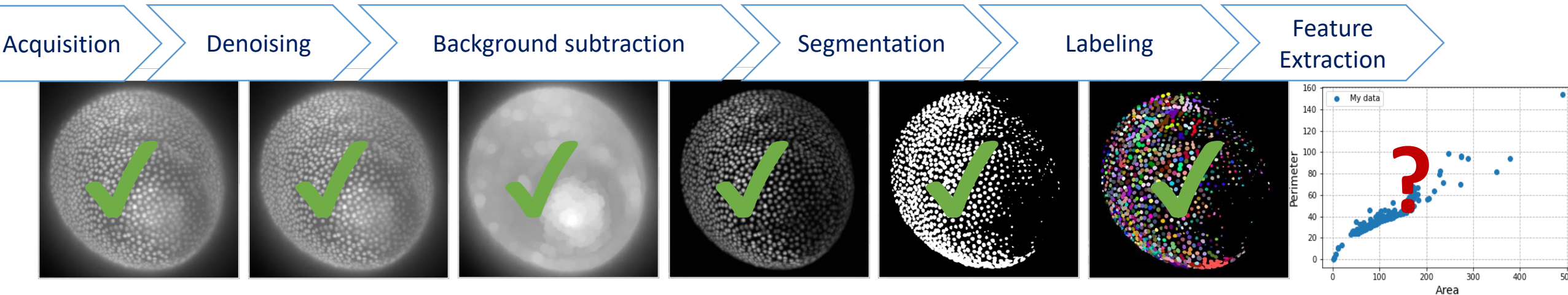
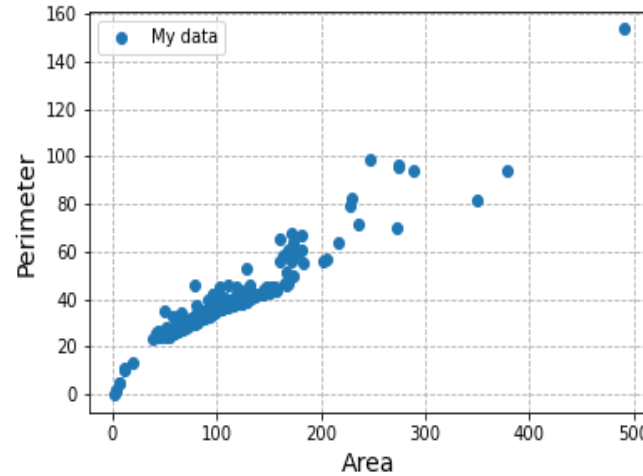


Semantic segmentation

- Feature extraction a *late* processing step in image analysis.
- It can be used for images, or segmented/labelled images

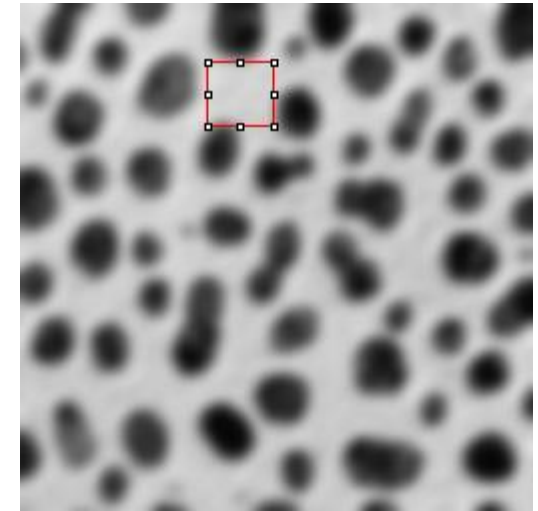
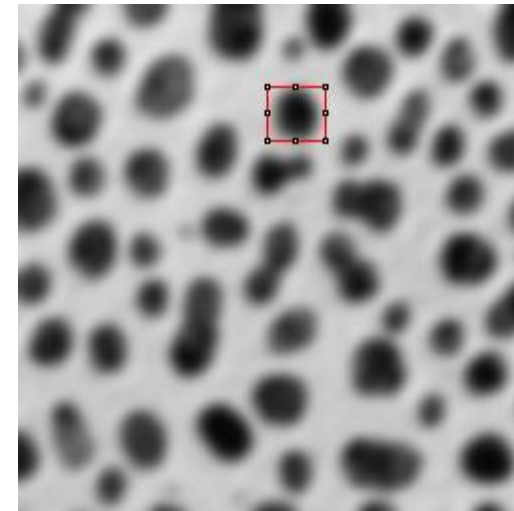
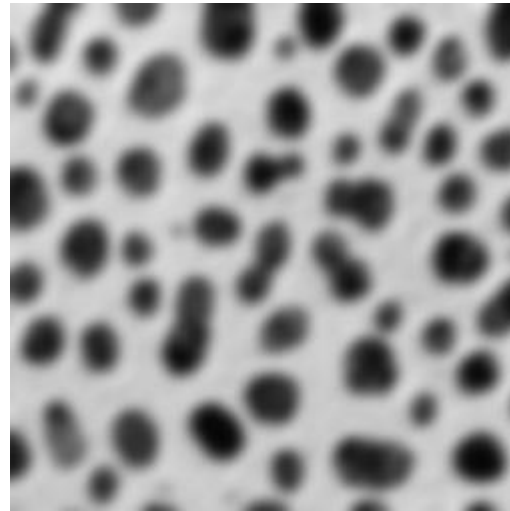


Feature
Extraction

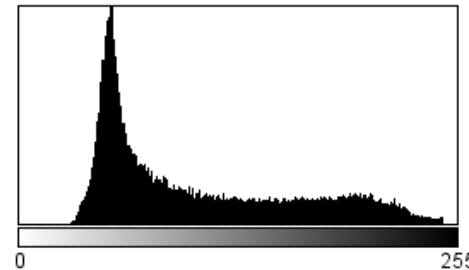


- A *feature* is a countable or measurable property of an image or object.
- Goal of feature extraction is finding a minimal set of features to describe an object well enough to differentiate it from other objects.
- **Intensity based**
 - Mean intensity
 - Standard deviation
 - Total intensity
 - Textures
- **Shape based /spatial**
 - Area / Volume
 - Roundness
 - Solidity
 - Circularity / Sphericity
 - Elongation
 - Centroid
 - Bounding box
- **Spatio-temporal**
 - Displacement,
 - Speed,
 - Acceleration
- **Others**
 - Overlap
 - Colocalization
 - Neighborhood
- **Mixed features**
 - Center of mass
 - Local minima / maxima

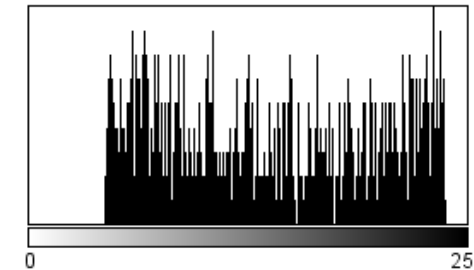
- Min / max
- Median
- Mean
- Mode
- Variance
- Standard deviation



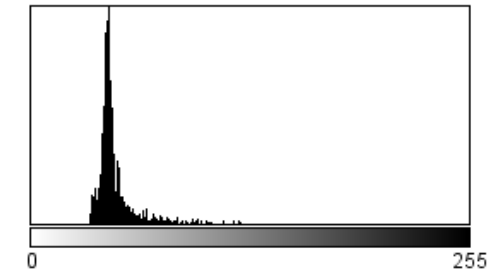
- Can be derived from pixel values
- Don't take spatial relationship of pixels into account
- See also:
 - descriptive statistics
 - histogram



Count: 65024
Mean: 103.301
StdDev: 57.991
Min: 29
Max: 248
Mode: 53 (1663)



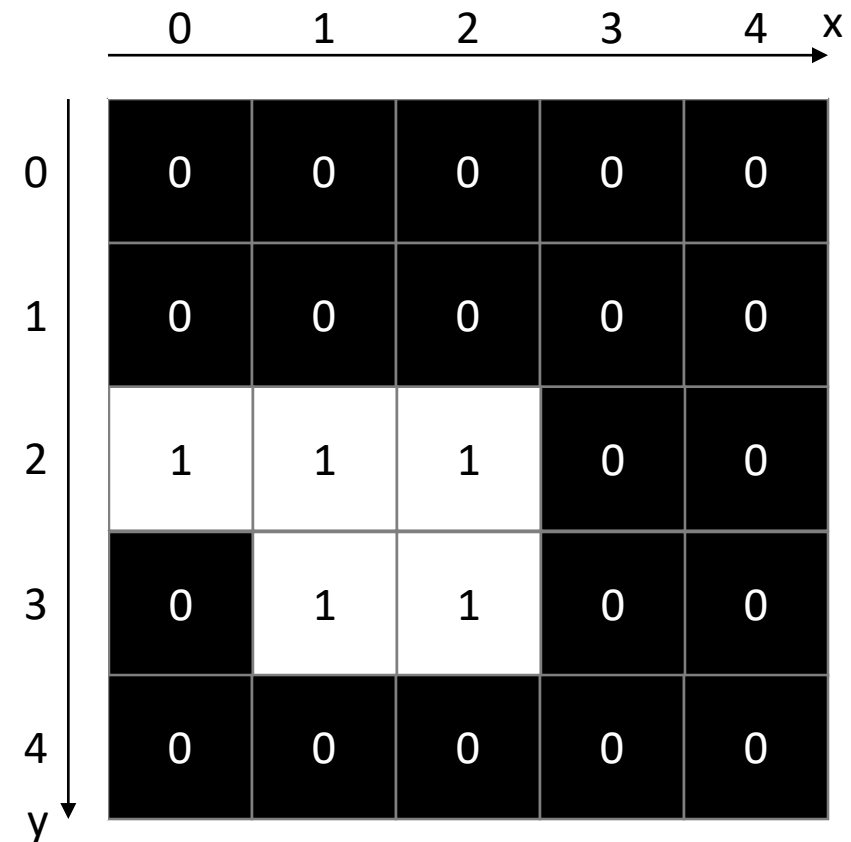
Count: 783
Mean: 141.308
StdDev: 61.876
Min: 44
Max: 243
Mode: 236 (9)



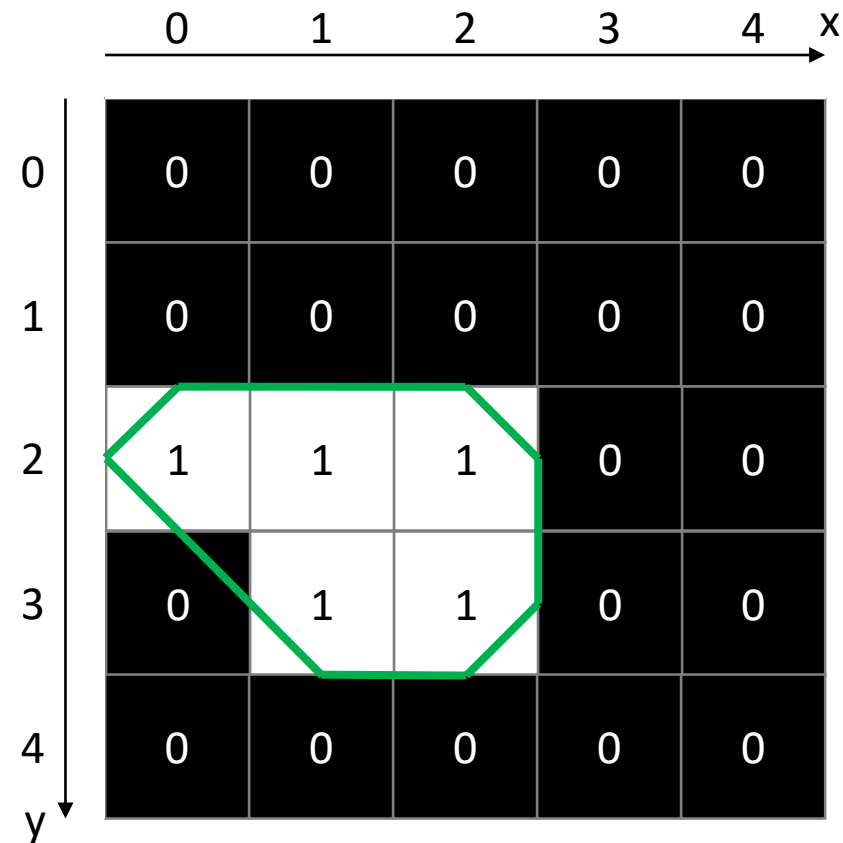
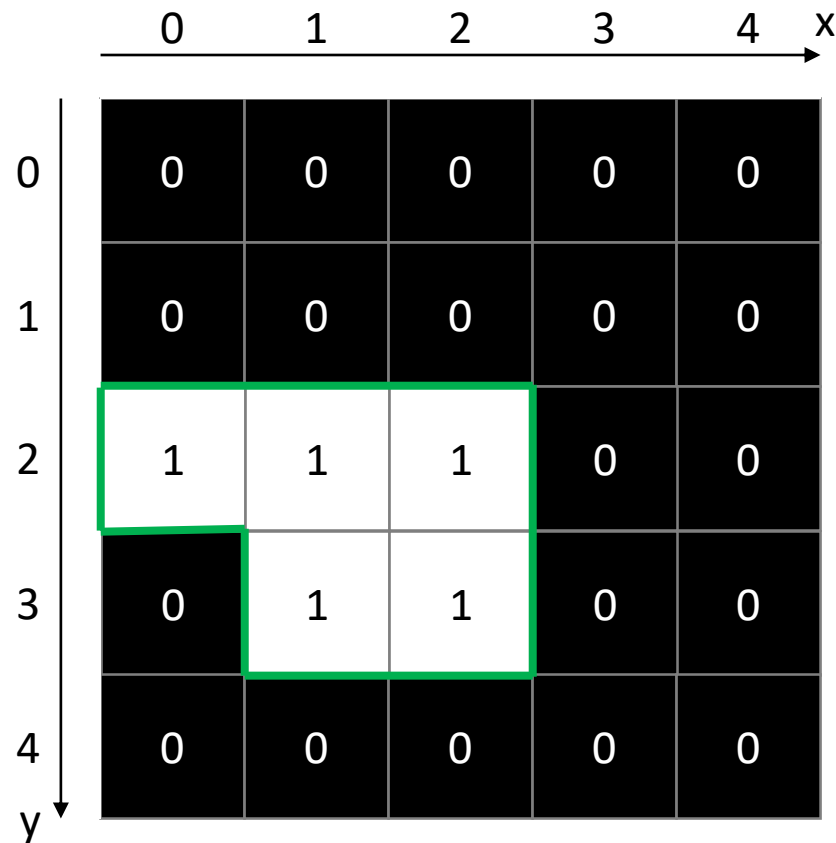
Count: 1056
Mean: 49.016
StdDev: 12.685
Min: 34
Max: 122
Mode: 45 (120)

- Position and size of the smallest rectangle containing all pixels of an object
 - x_b, y_b ... position of the bounding box
 - w_b ... width of the bounding box
 - h_b ... height of the bounding box

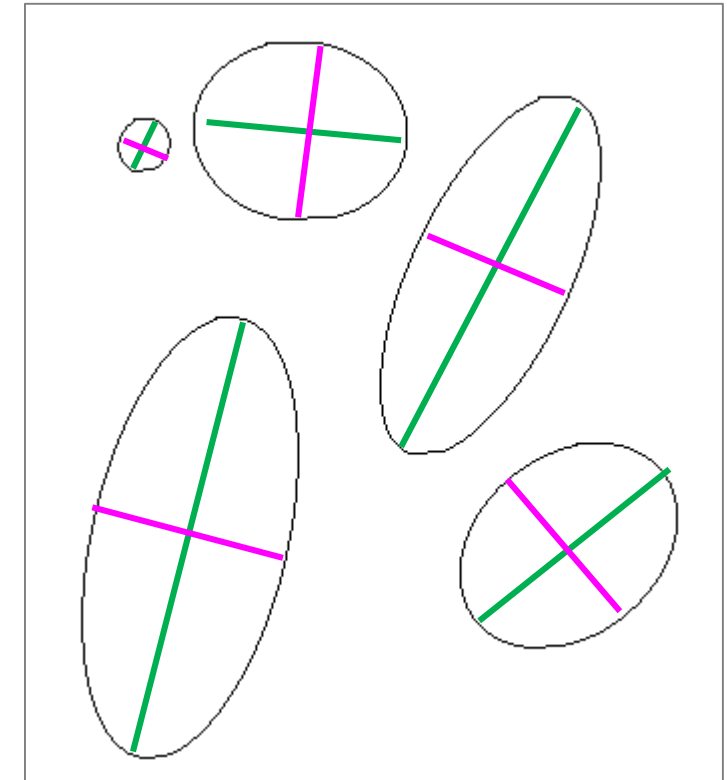
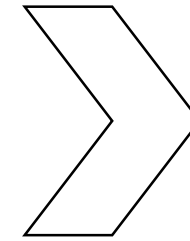
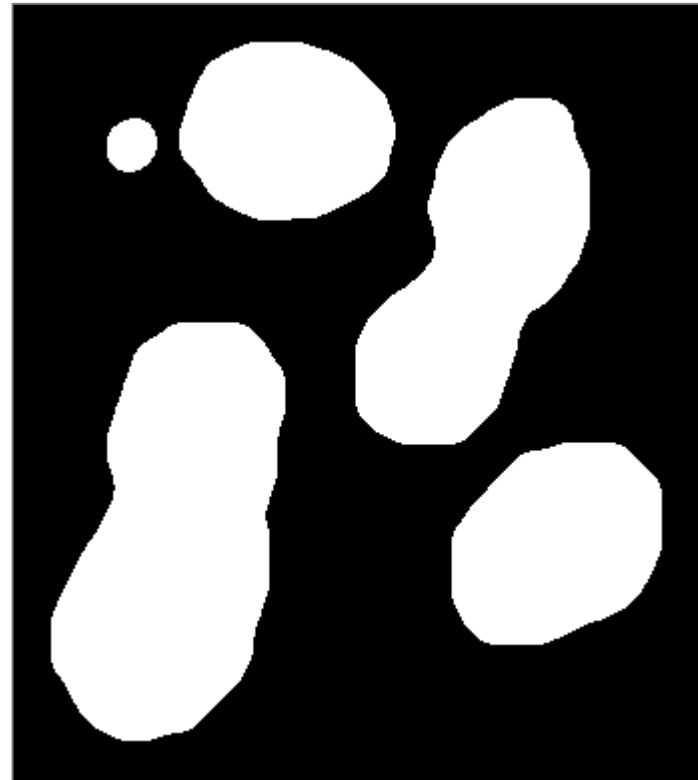
variable	value
x_b	0
y_b	2
w_b	3
h_b	2



- Length of the outline around an object
- Depends on the actual implementation

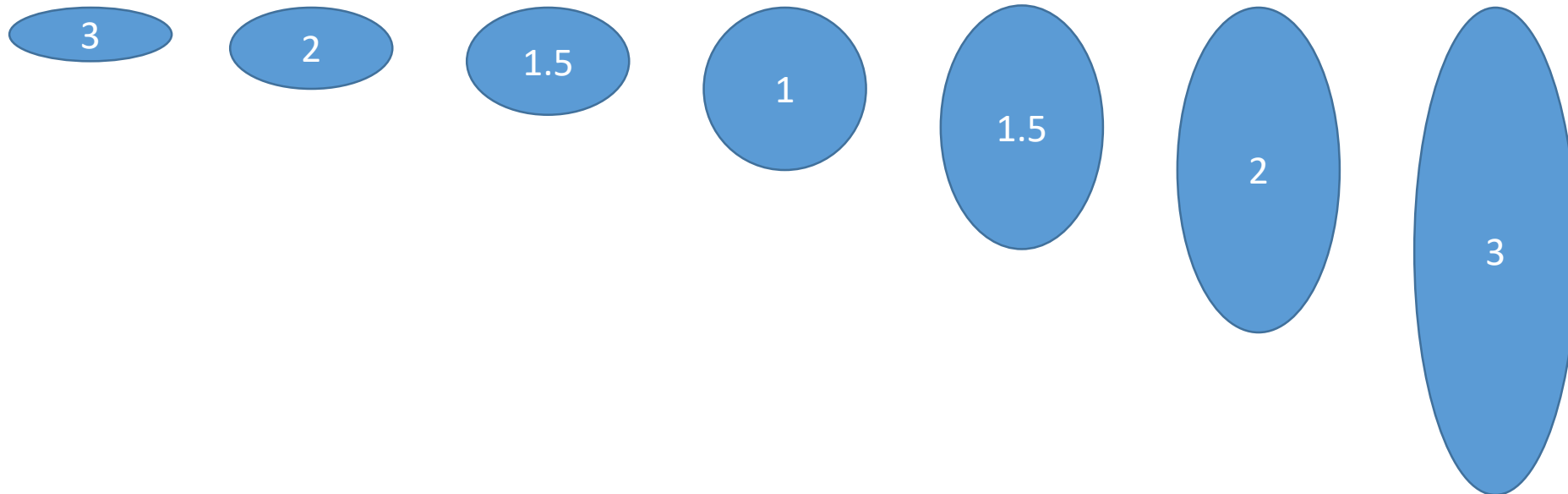


- For every object, find the optimal ellipse simplifying the object.
- Major axis ... long diameter
- Minor axis ... short diameter
- Major and minor axis are perpendicular to each other

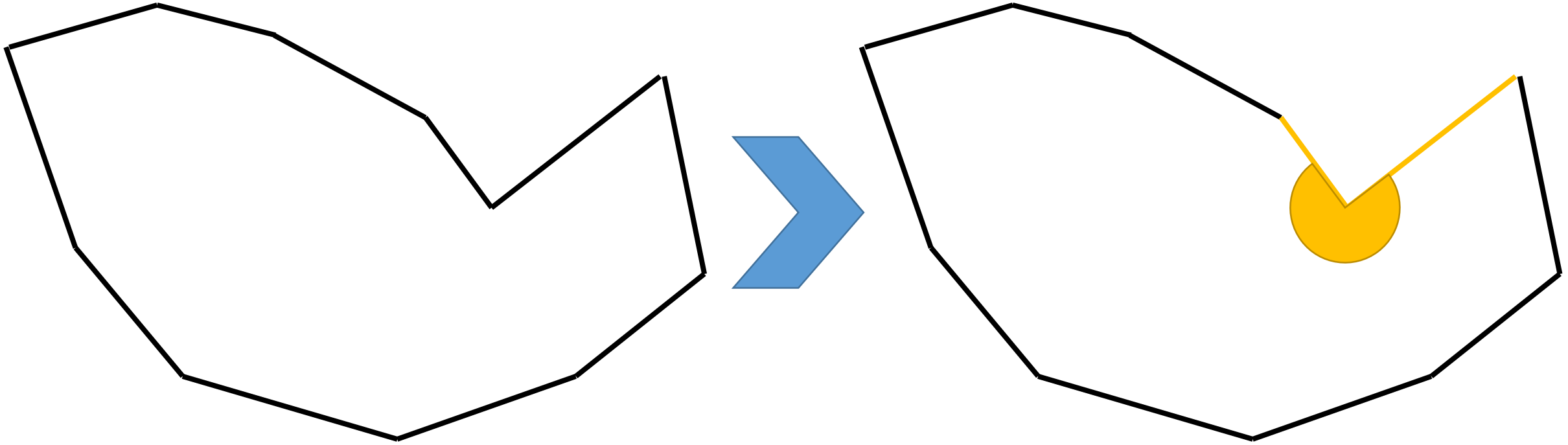


- The aspect ratio describes the elongation of an object.

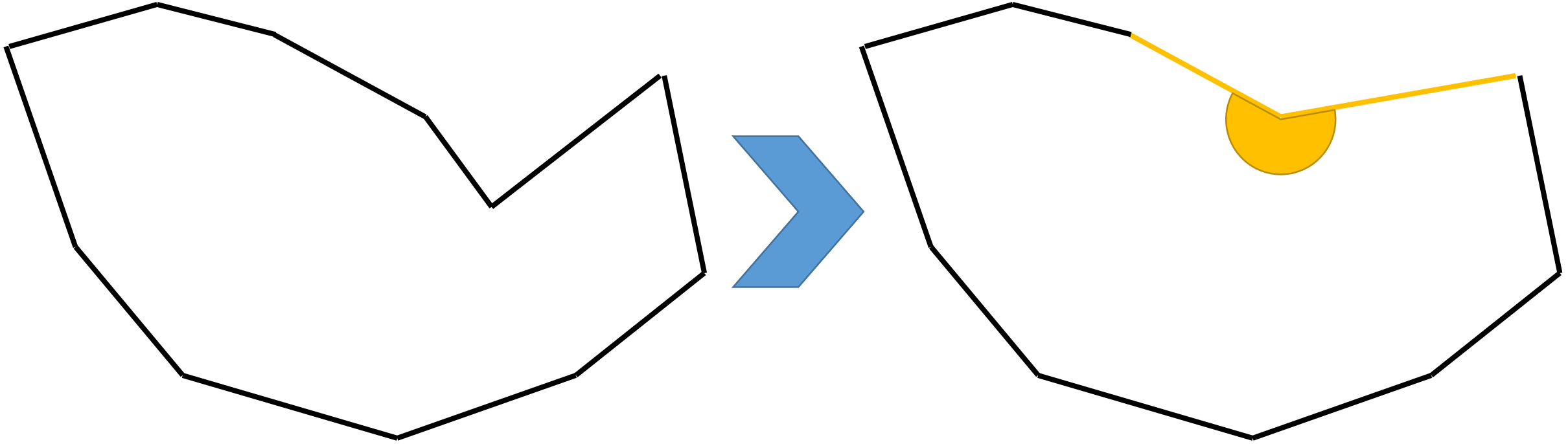
$$AR = \text{major} / \text{minor}$$



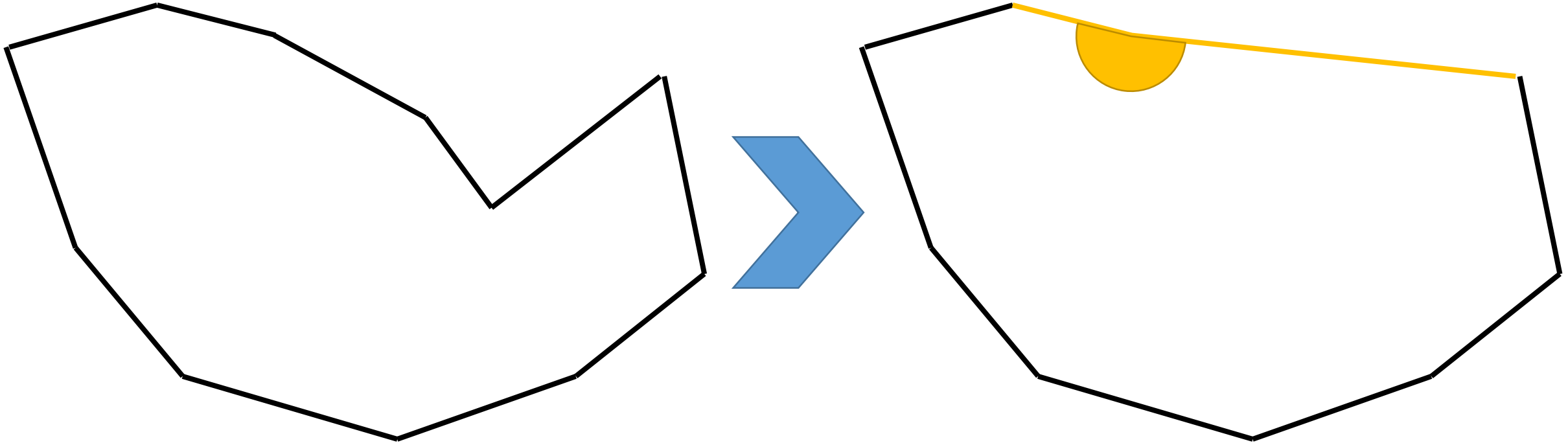
- By removing all concave corners of an object, we retrieve its **convex hull**.



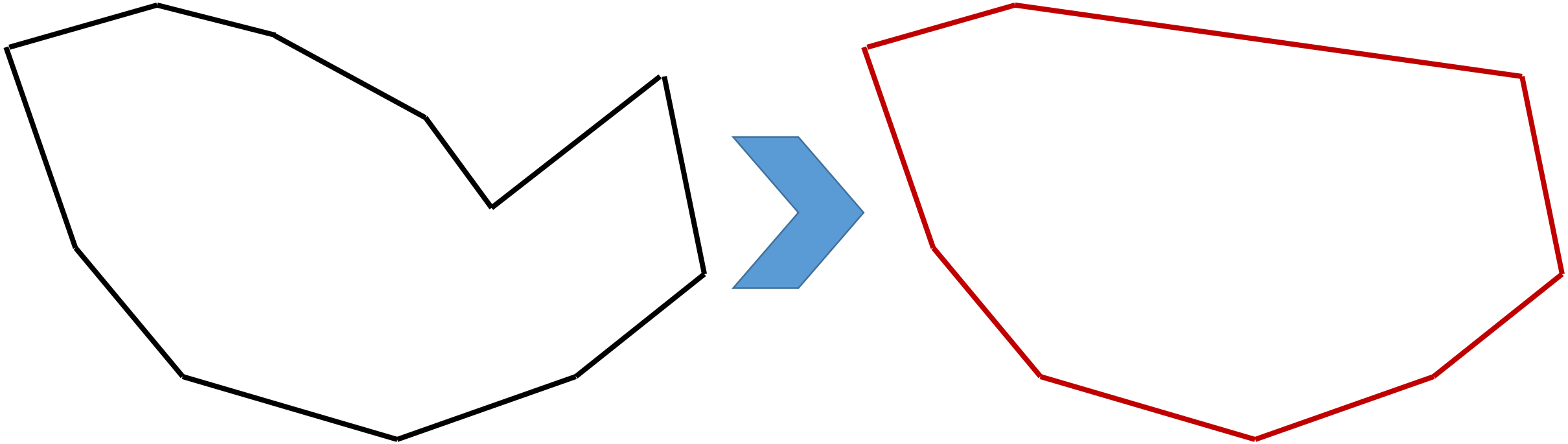
- By removing all concave corners of an object, we retrieve its **convex hull**.



- By removing all concave corners of an object, we retrieve its **convex hull**.



- By removing all concave corners of an object, we retrieve its **convex hull**.



$$solidity = \frac{A}{A_{convexHull}}$$

Roundness and circularity

- The definition of a circle leads us to measurements of circularity and roundness.
- In case you use these measures, define them correctly. They are not standardized!

Diameter

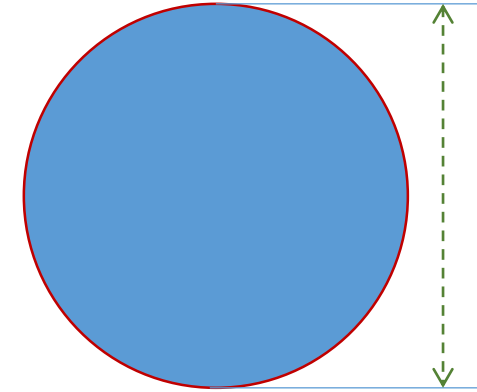
d

Circumference

$$C = \pi d$$

Area

$$A = \frac{\pi d^2}{4}$$



$$roundness = \frac{4 * A}{\pi major^2}$$

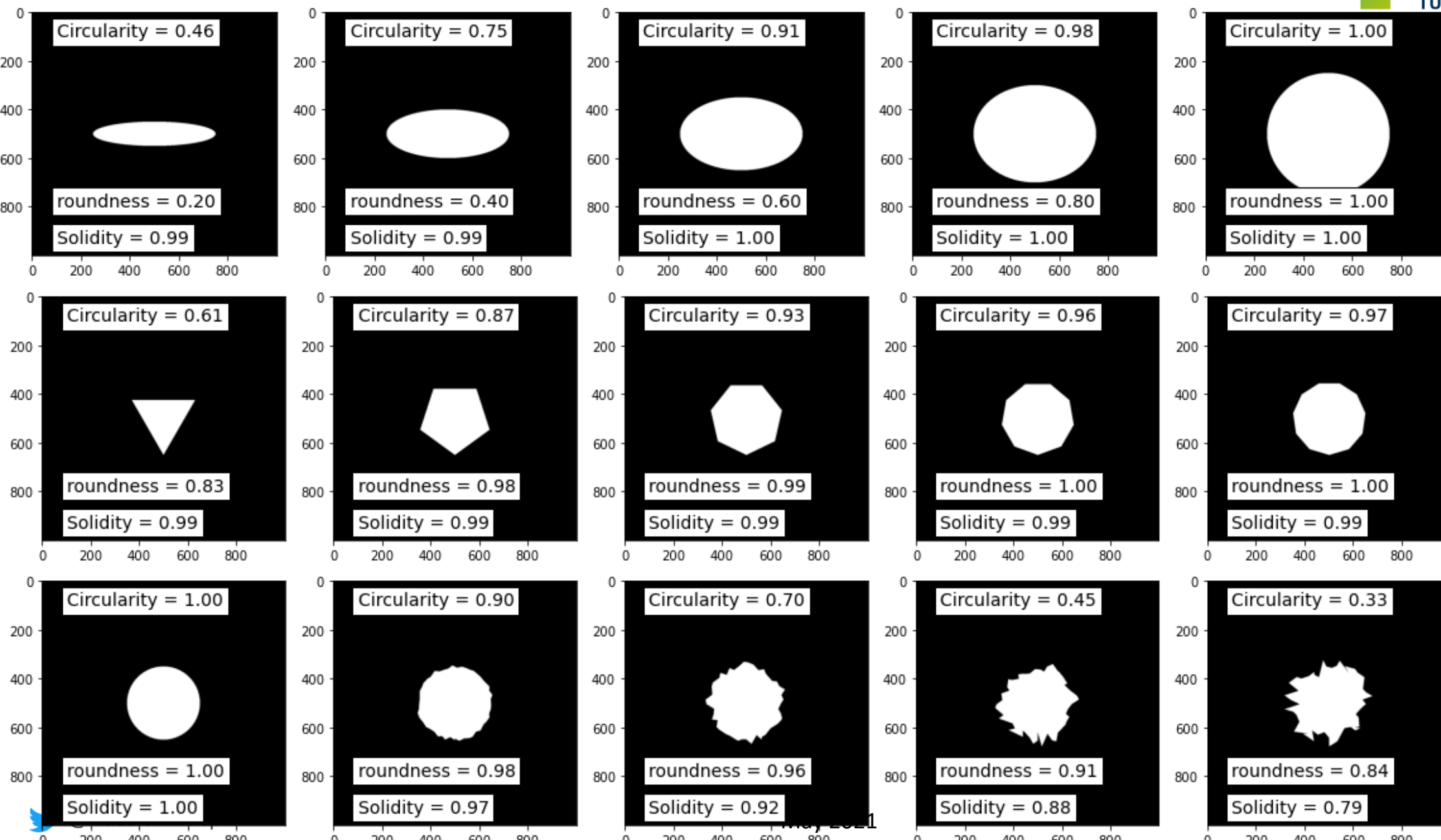
$$circularity = \frac{4\pi * A}{perimeter^2}$$

Roundness = 1
Circularity = 1

Roundness \approx 1
Circularity \approx 1

Roundness < 1
Circularity < 1

Roundness and circularity



$$roundness = \frac{4 * A}{\pi major^2}$$

$$circularity = \frac{4\pi * A}{perimeter^2}$$

$$solidity = \frac{A}{A_{convexHull}}$$

- In Fiji: *Analyze > Analyze Particles...*
- In Python: `from skimage import measure`

<https://scikit-image.org/docs/stable/api/skimage.measure.html>

`skimage.measure.blur_effect` (image[, h_size, ...]) Compute a metric that indicates the strength of blur in an image (0 for no blur, 1 for maximal blur).

`skimage.measure.euler_number` (image[, ...]) Calculate the Euler characteristic in binary image.

`skimage.measure.find_contours` (image[, ...]) Find iso-valued contours in a 2D array for a given level value.

`skimage.measure.grid_points_in_poly` (shape, verts) Test whether points on a specified grid are inside a polygon.

`skimage.measure.inertia_tensor` (image[, mu]) Compute the inertia tensor of the input image.

`skimage.measure.inertia_tensor_eigvals` (image) Compute the eigenvalues of the inertia tensor of the image.

`skimage.measure.label` (label_image[, ...]) Label connected regions of an integer array.

`skimage.measure.regionprops` (label_image[, ...]) Measure properties of labeled image regions.

`skimage.measure.regionprops_table` (label_image) Compute image properties and return them as a pandas-compatible table.

area : int

Number of pixels of the region.

area_bbox : int

Number of pixels of bounding box.

area_convex : int

Number of pixels of convex hull image, which is the smallest convex polygon that encloses the region.

area_filled : int

Number of pixels of the region with all the holes filled in. Describes the area of the region.

axis_major_length : float

The length of the major axis of the ellipse that has the same normalized second central moments as the region.

axis_minor_length : float

The length of the minor axis of the ellipse that has the same normalized second central moments as the region.

