PoL Physics of Life TU Dresden

CENTER FOR SYSTEMS BIOLOGY DRESDEN

Image segmentation

Robert Haase

With material from

Johannes Müller & Marcelo Zoccoler, BiAPol, PoL, TU Dresden

Benoit Lombardot, Scientific Computing Facility, MPI CBG

September 2022

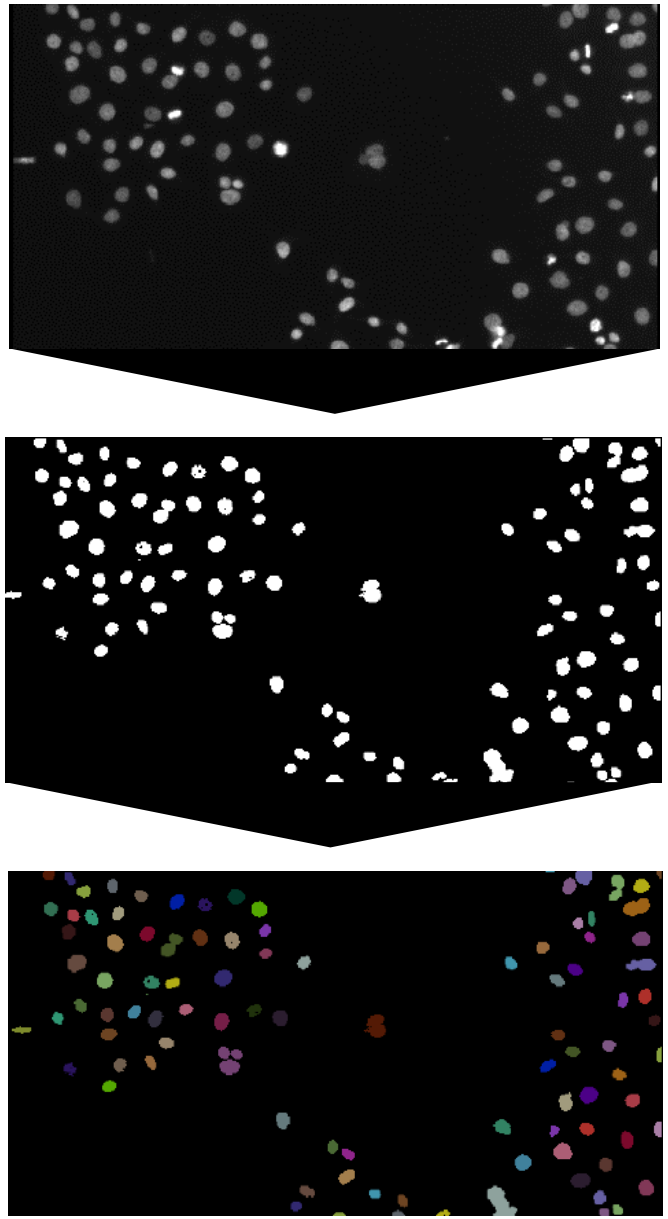
@haesleinhuepf

Lecture overview

- Creating binary masks from intensity images
  → Thresholding

- Create label images from binary masks
  → Connected-component analysis

- Short cuts

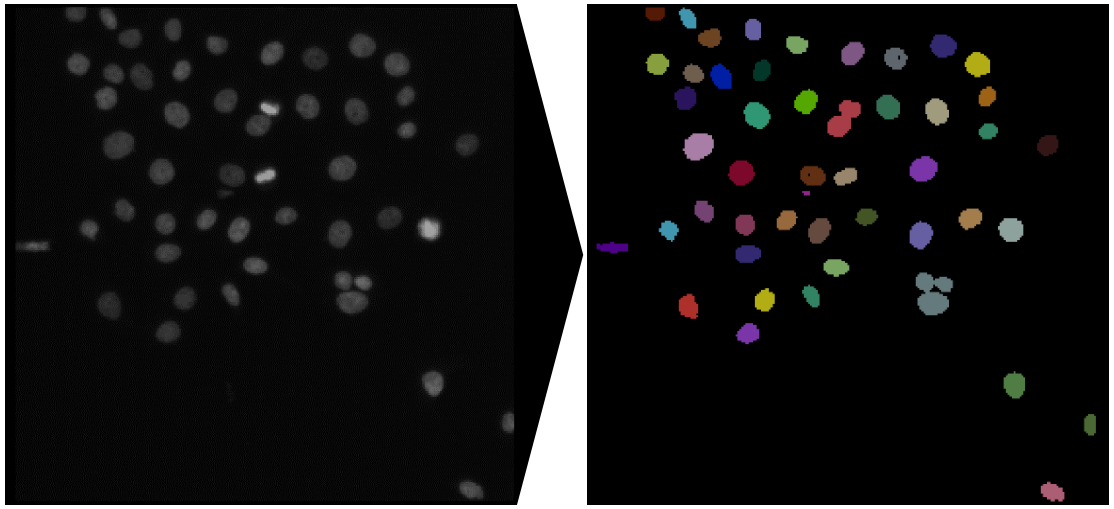


Label-image

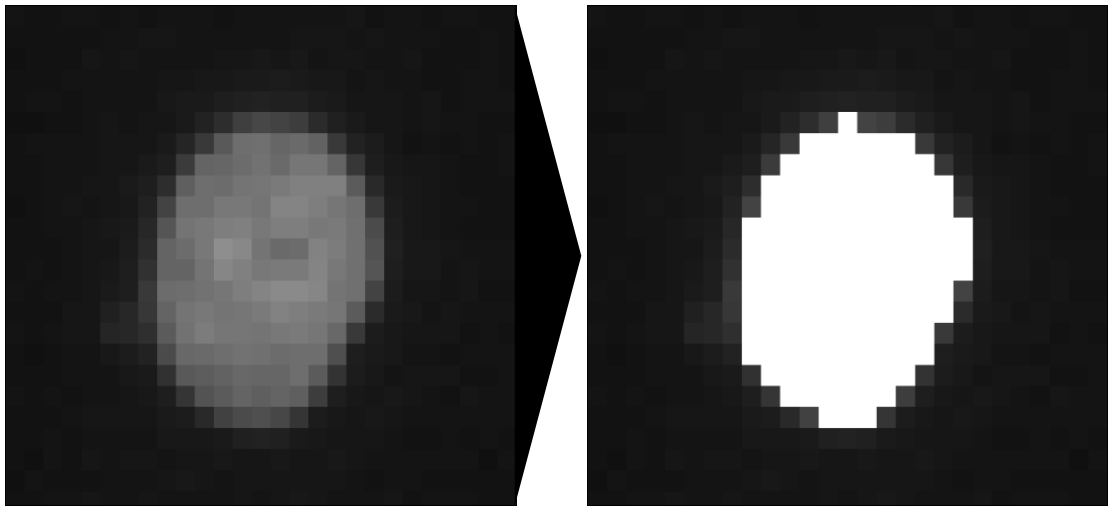
Segmentation

Vocabulary:

- **Segmentation:**
  Grouping pixels
- **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

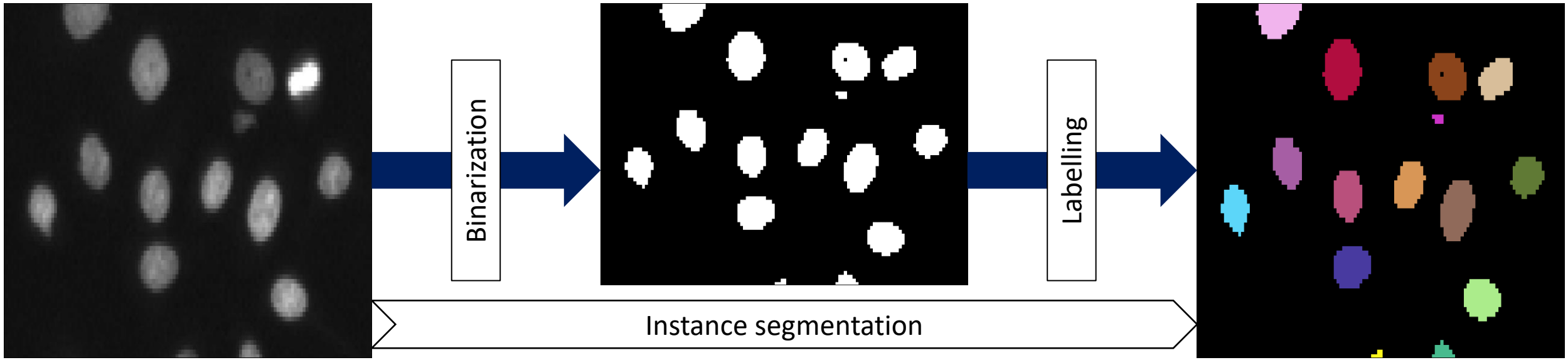


Semantic segmentation

Segmentation and labelling

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

- Methods
  - Thresholding + connected components labeling
  - Seeded watershed
  - Machine learning



Thresholding in Python

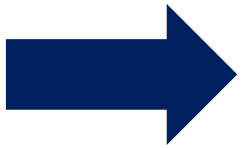
PoL Physics of Life TU Dresden

- 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)
```

```
False
True
False
```



```
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]])
```

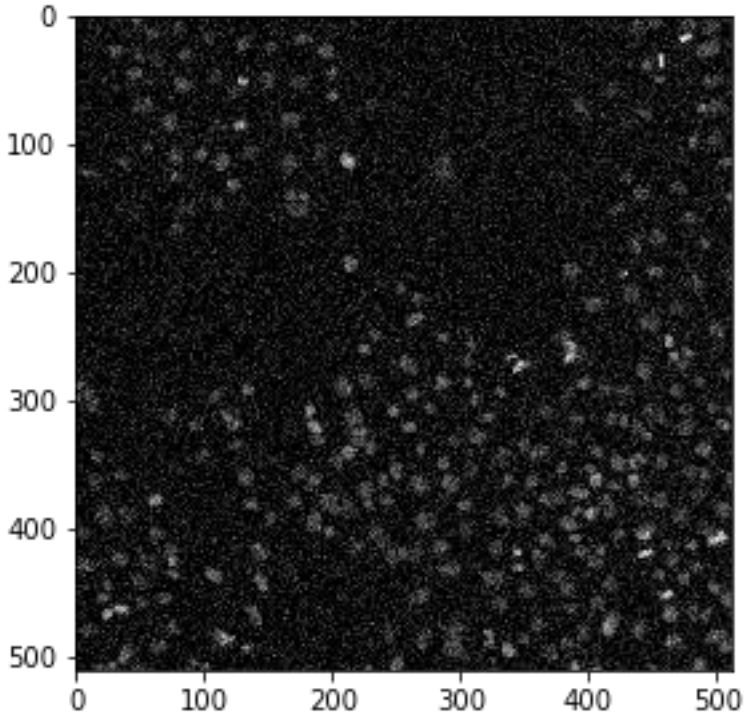
In this case, “image” is a *numpy array* → some operations are automatically applied to every pixel!

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

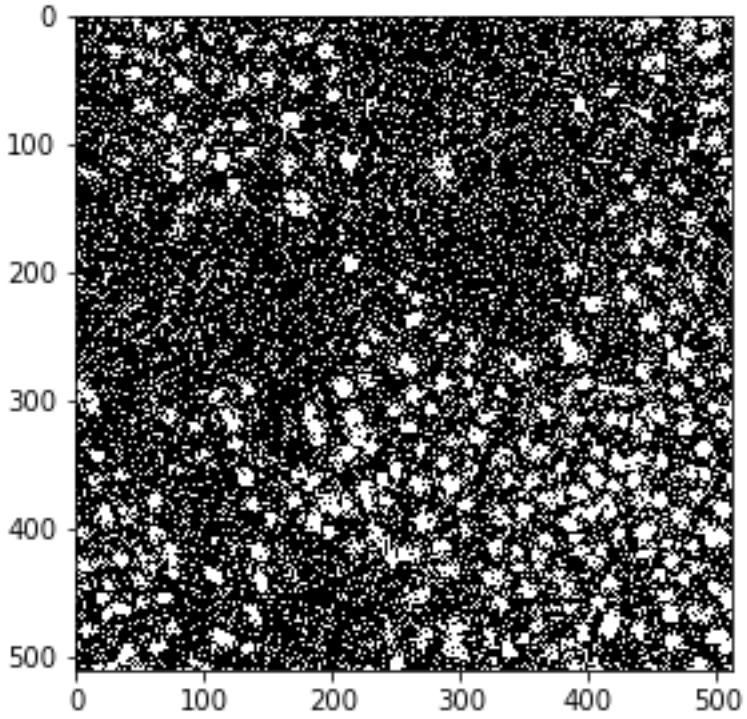
```
binary = image > threshold
```

Reminder: pre-processing!

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



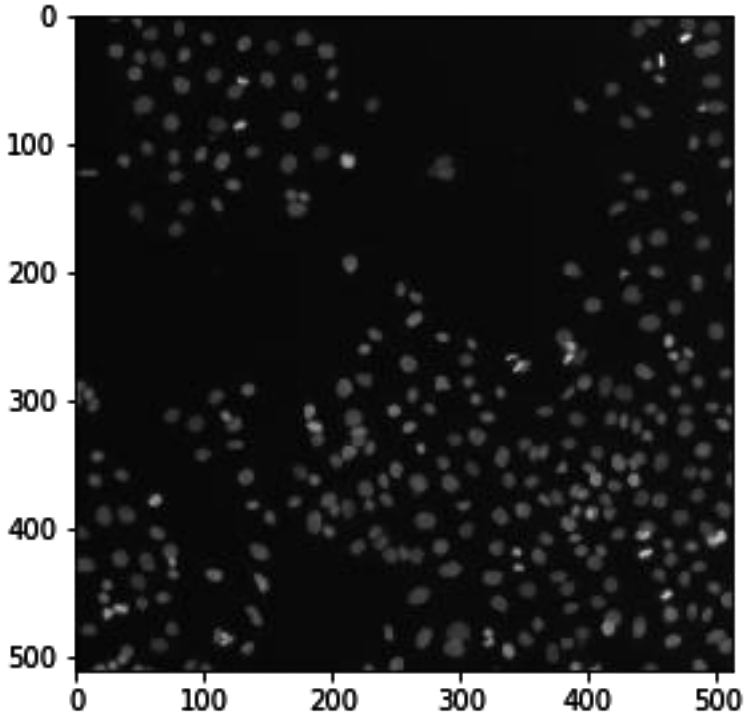
Noisy image



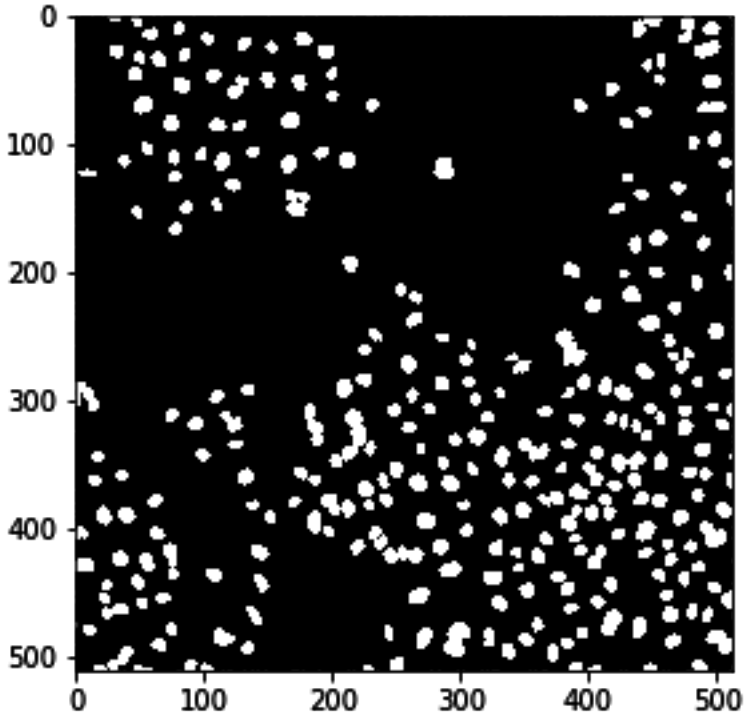
Thresholded image

Reminder: pre-processing!

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



Filtered image



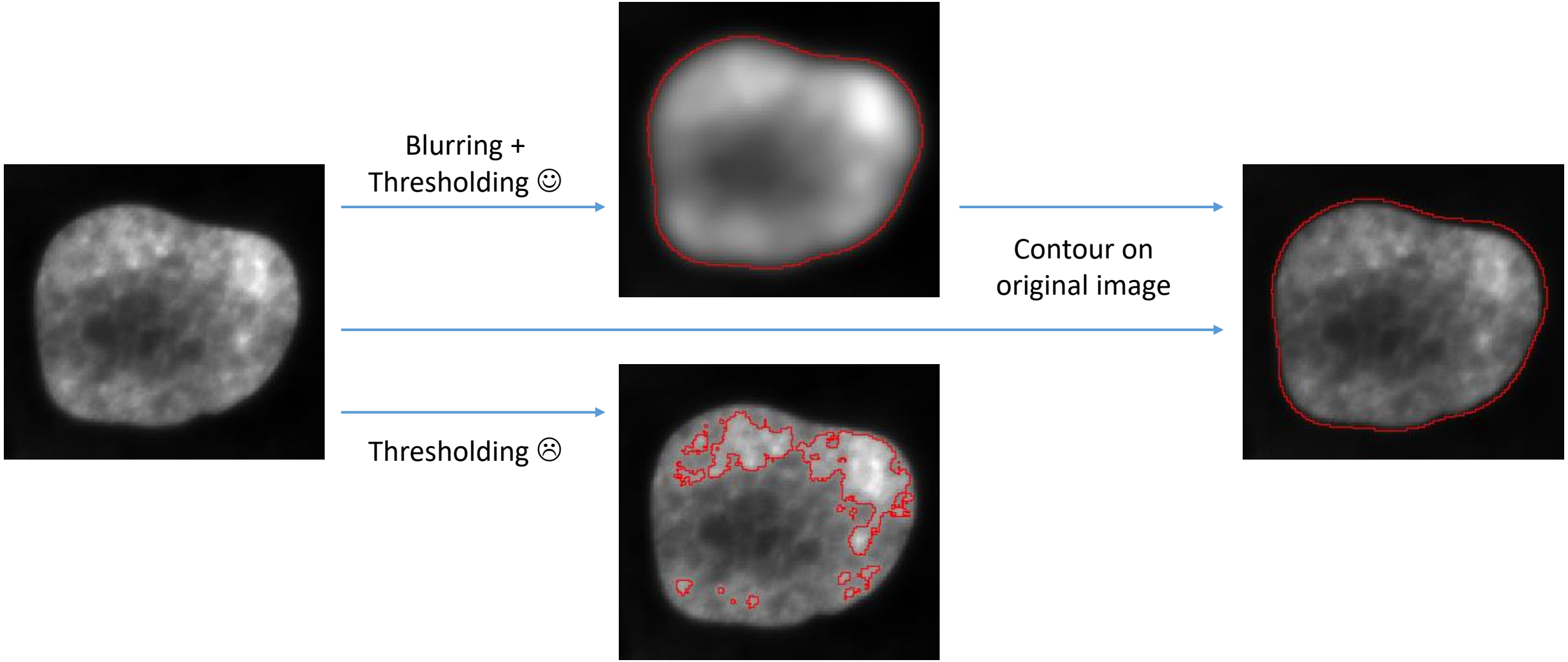
Thresholded image

```
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!

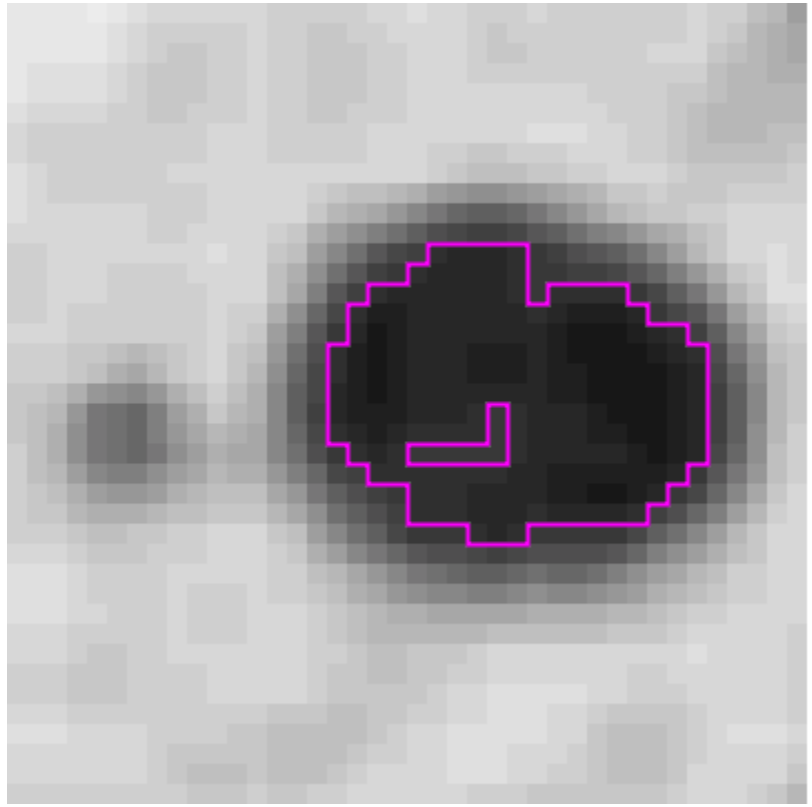


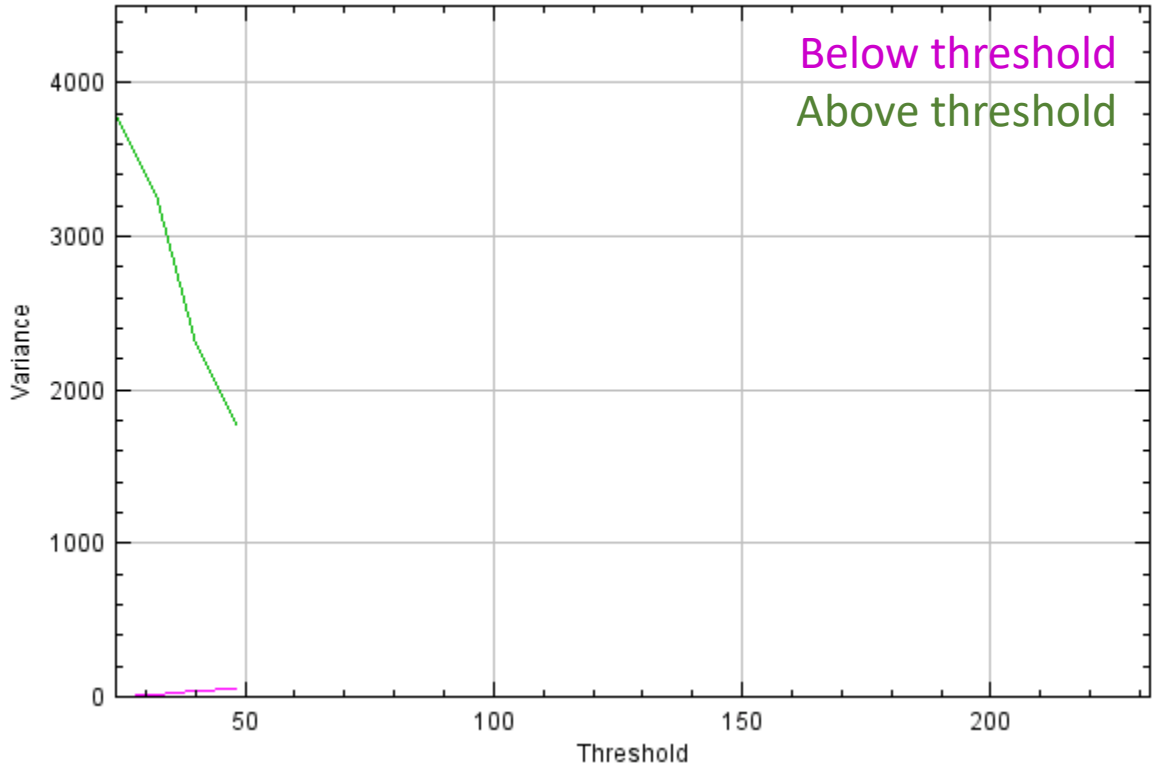
Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.

$$Var(I) = \sum_{i \in I} g_i - \bar{g}_I \qquad \bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$ … Variance in image I
g_i … grey value of a pixel i
$\bar{g}_I$ … mean grey value of the whole image I
n_I … number of pixels in Image I



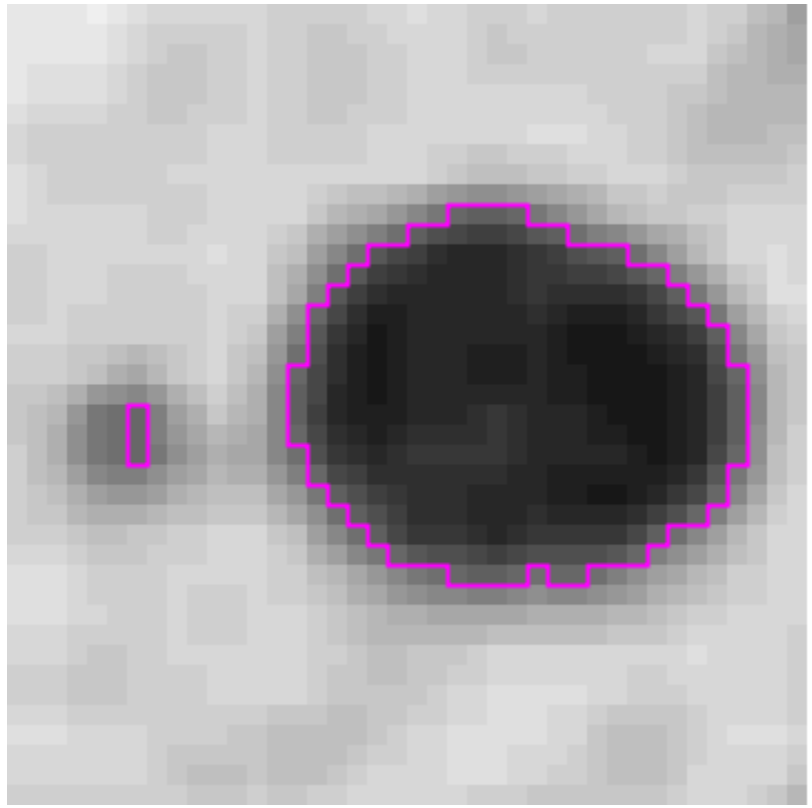


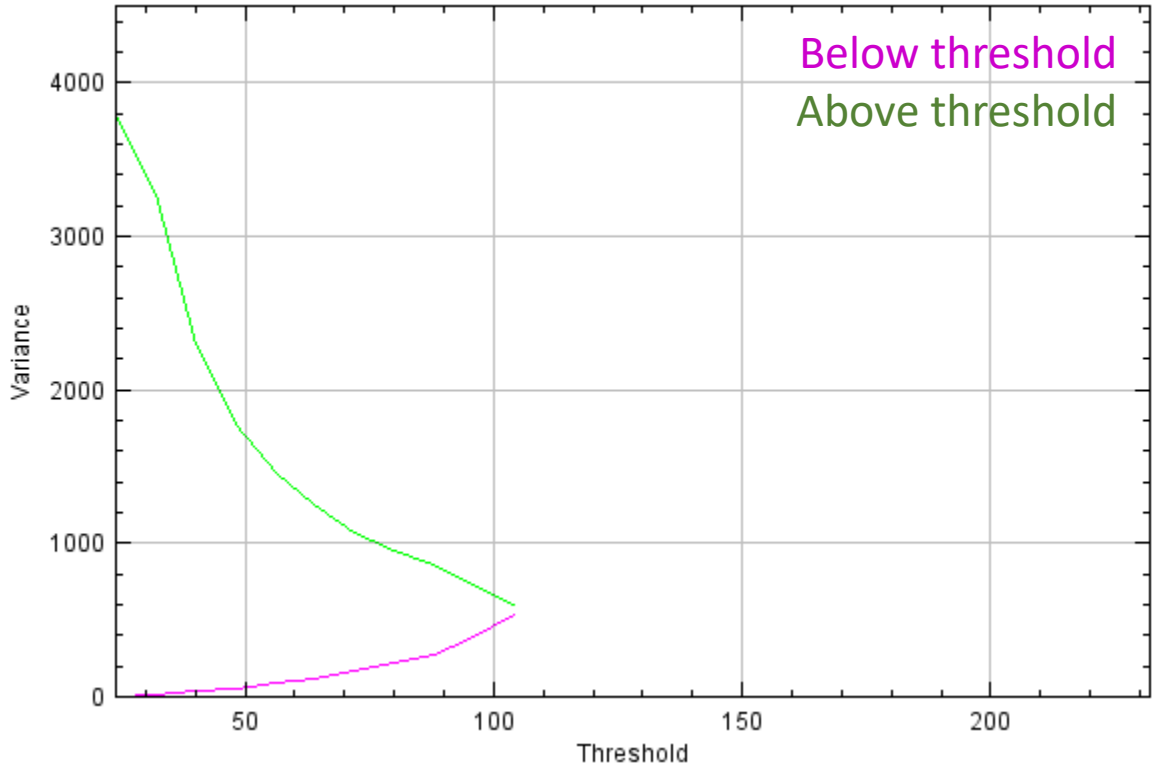
Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.

$$Var(I) = \sum_{i \in I} g_i - \bar{g}_I \qquad \bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$ … Variance in image I
g_i … grey value of a pixel i
$\bar{g}_I$ … mean grey value of the whole image I
n_I … number of pixels in Image I



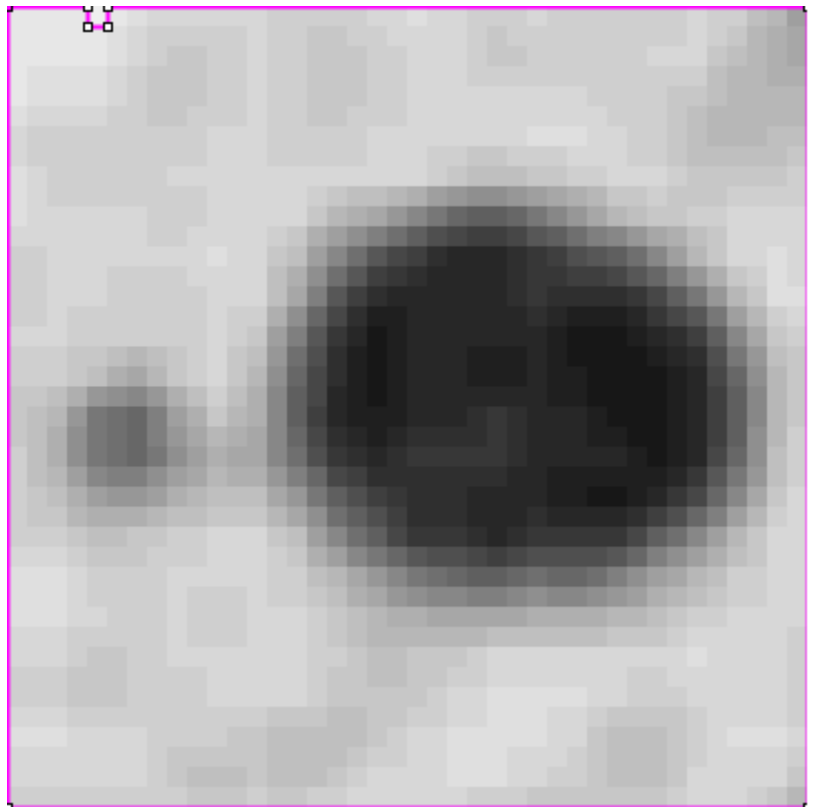


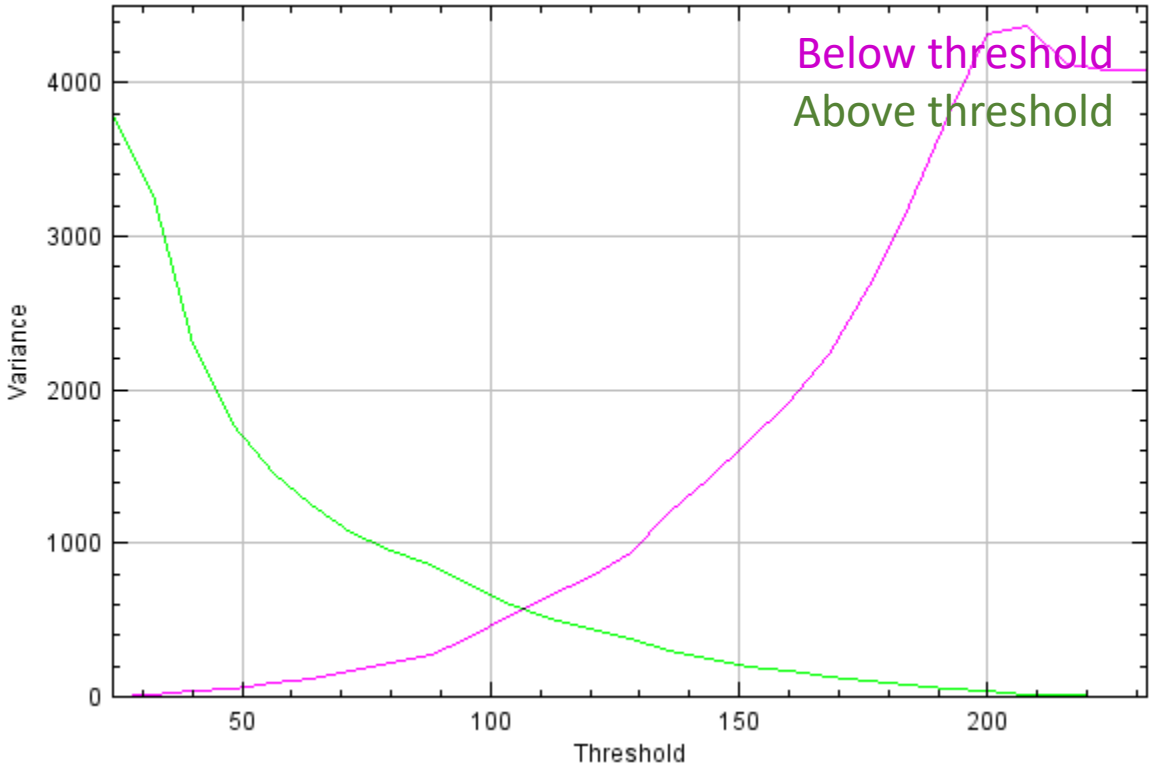
Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.

$$Var(I) = \sum_{i \in I} g_i - \bar{g}_I \qquad \bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$ … Variance in image I
g_i … grey value of a pixel i
$\bar{g}_I$ … mean grey value of the whole image I
n_I … number of pixels in Image I

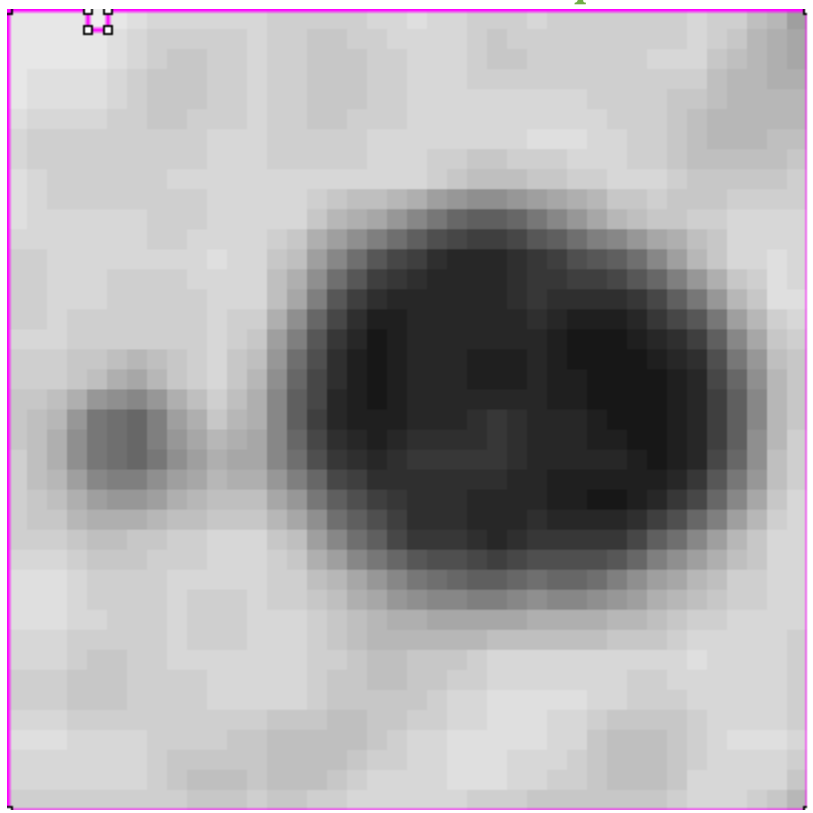


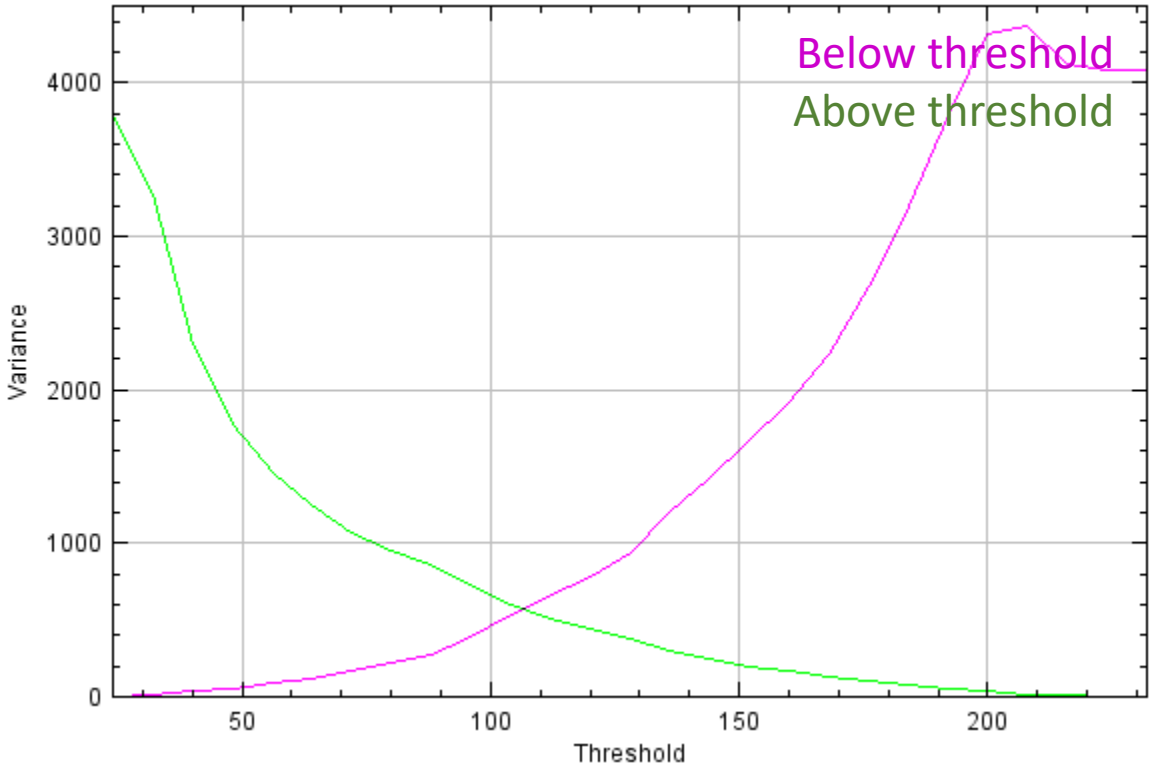


Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.
- Weighted (!) sum variance

$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \qquad I = A \cup B$$

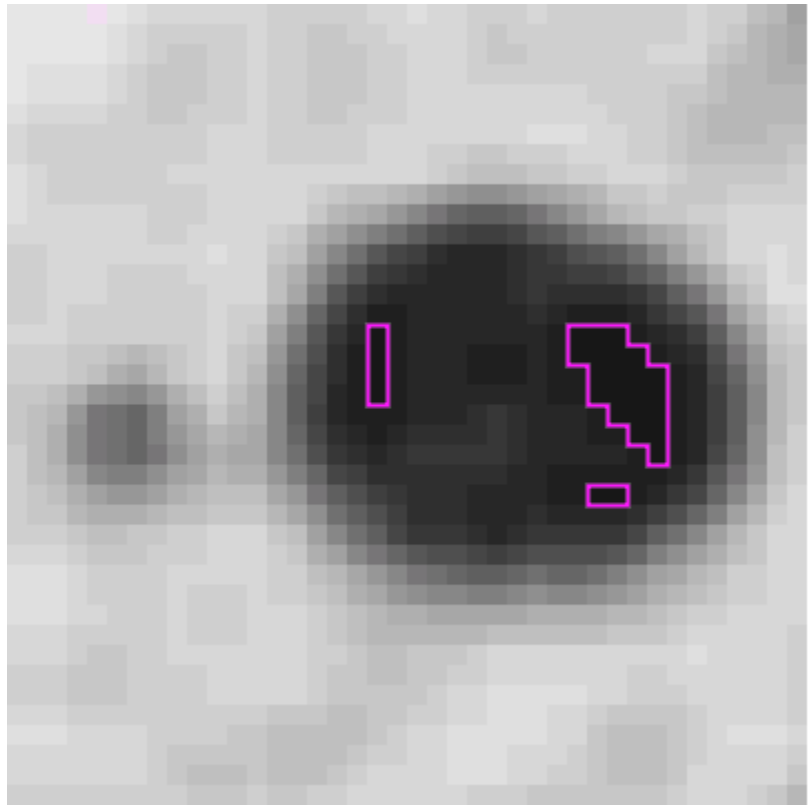


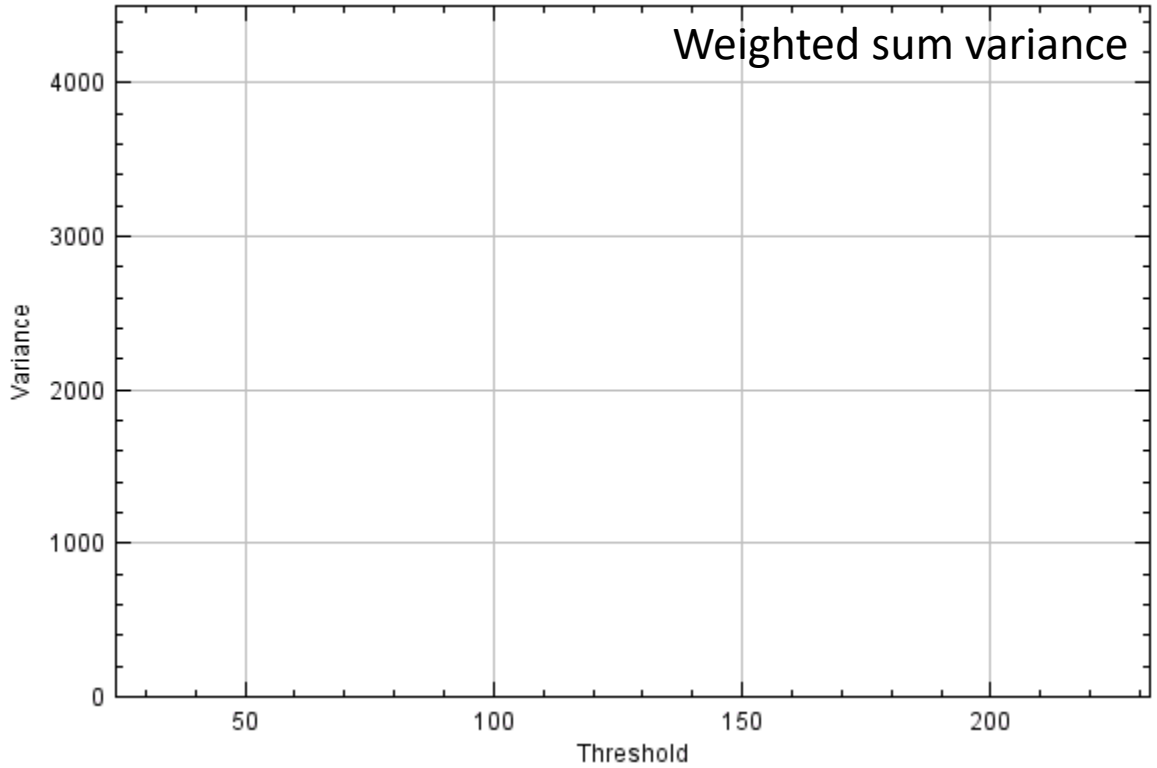


Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.
- Weighted (!) sum variance

$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \qquad I = A \cup B$$

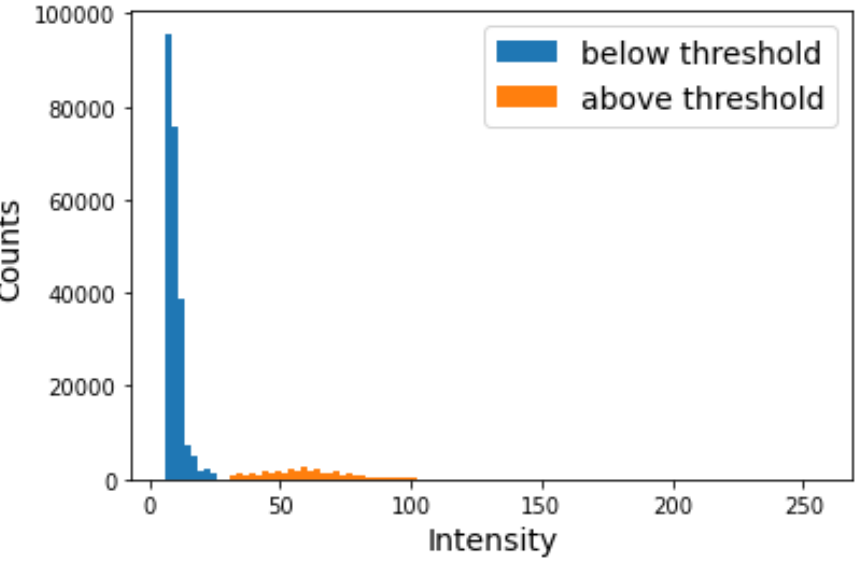


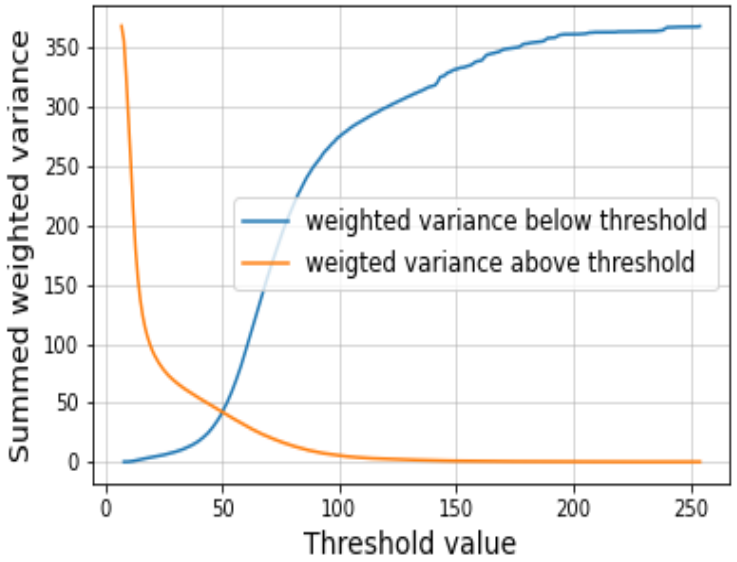


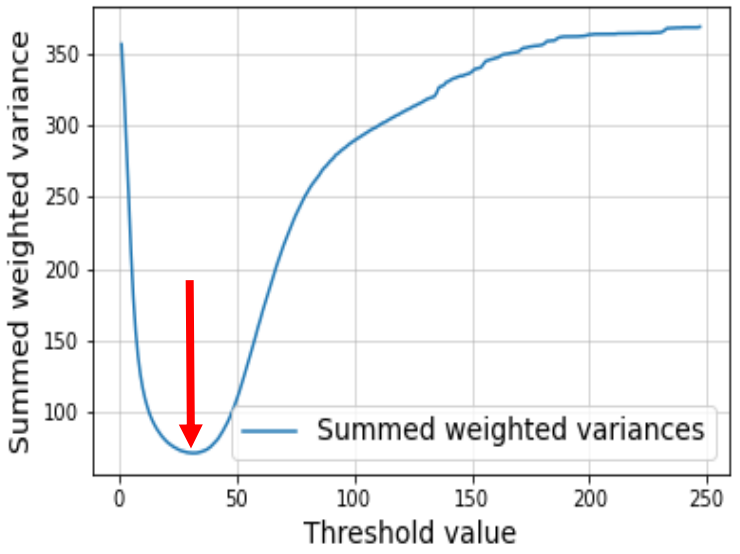
Threshold algorithms

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

$$Var(I) = \frac{1}{n_I}\sum\left(I - mean(I)\right)^2 \quad \rightarrow \quad Var_{w,sum} = \frac{n_A}{n_I} \cdot Var(A) + \frac{n_B}{n_I} \cdot Var(B)$$







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

Thresholding: Citing

- 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 Fiji (Schindelin et Al. 2012).

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

Thresholding with scikit-image

```
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)
```

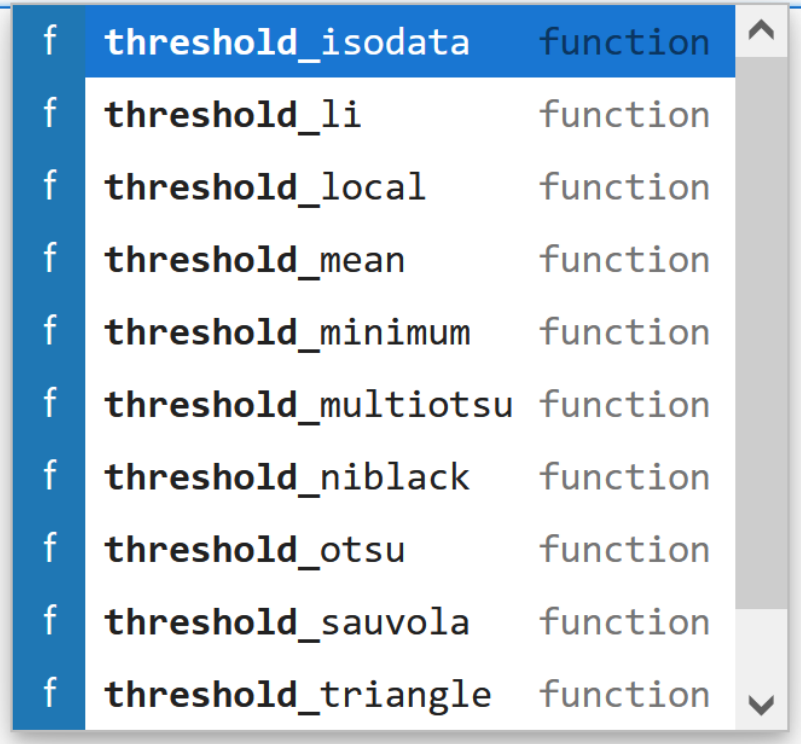
- **Statistical thresholding**: Pixels above statistical parameter of I belong to foreground. (Possibilities: Mean, Median, Quartiles, etc.)

```
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:

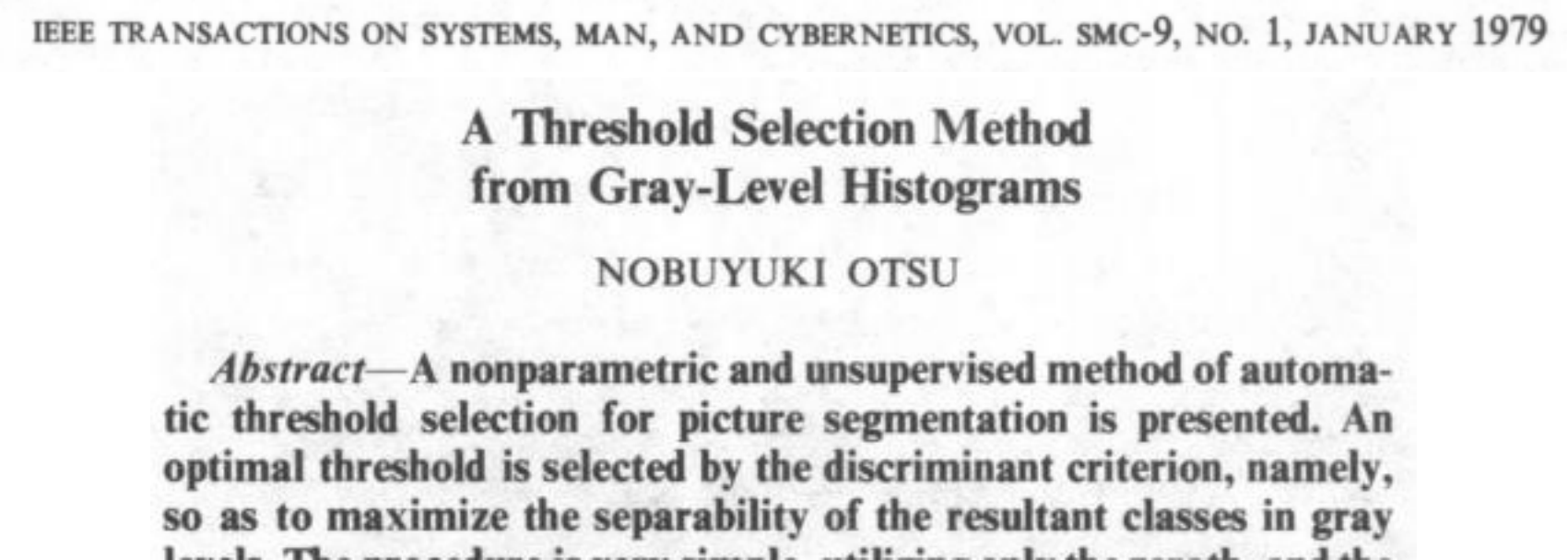
```
from skimage import filters

threshold = filters.threshold_
```



Thresholding: Citing

- 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**

Thresholding: Pitfalls

PoL Physics of Life TU Dresden

```
binary = image > a_good_threshold_value_of_my_choice
```

Never use manual thresholding!

Inter-observer variability

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

Intra-observer variability

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

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:

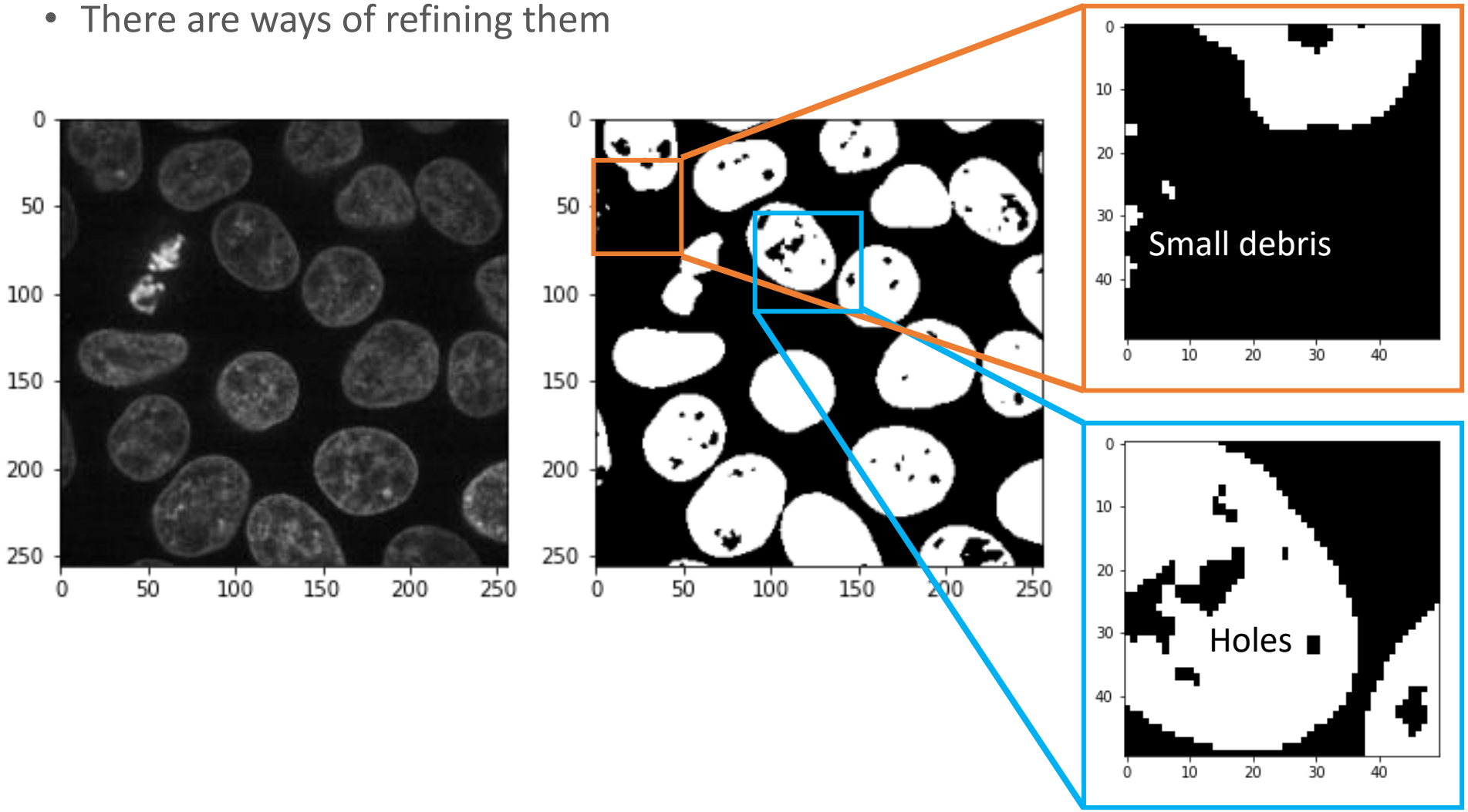
…and stick to it for the whole study. Using a new method for every image impairs reproducibility!

Do not over-engineer

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

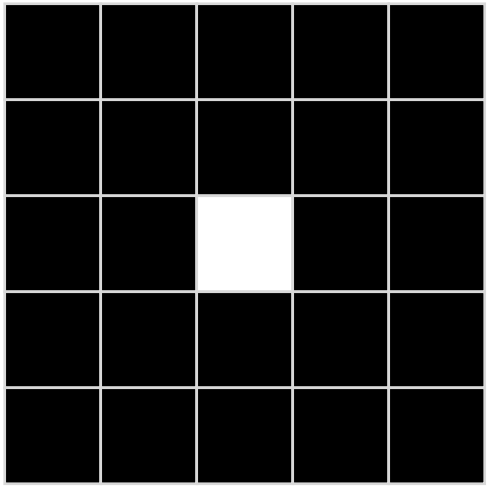
Refining masks

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



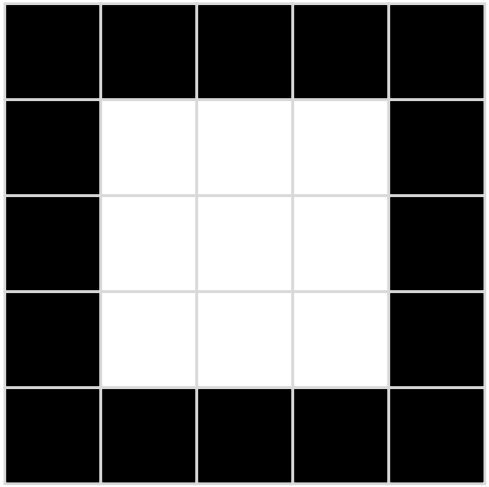
Refining masks: Dilation

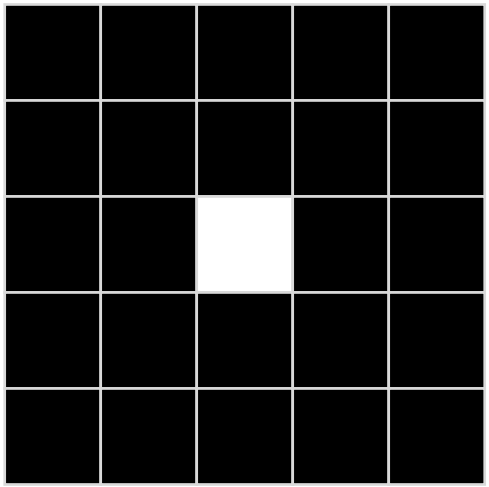
- Dilation: Every pixel with at least one white neighbor becomes white.



Dilation

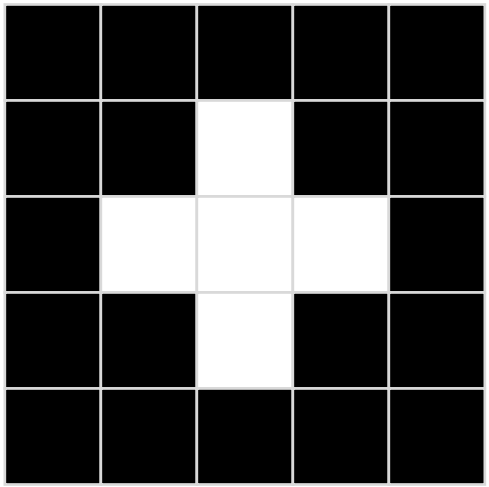
8-connected neighborhood
Moore-Neighborhood





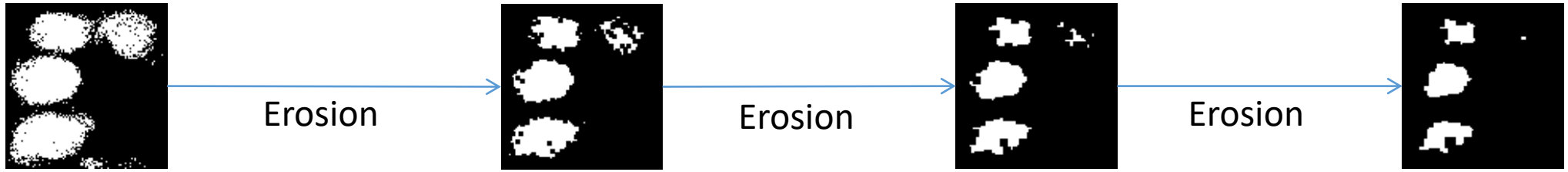
Dilation

4-connected neighborhood
von-Neumann-Neighborhood

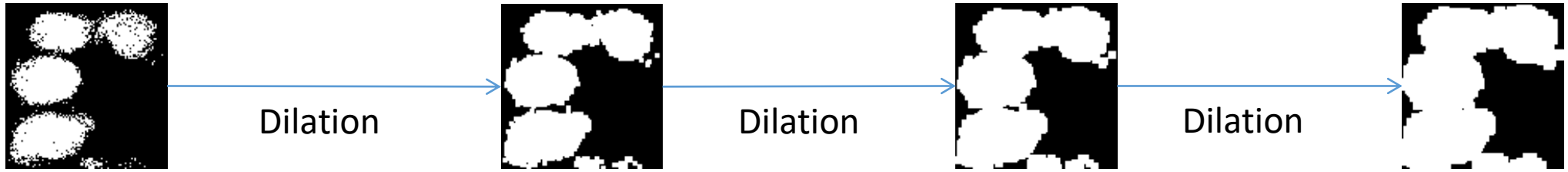


Recap: Filters on binary images

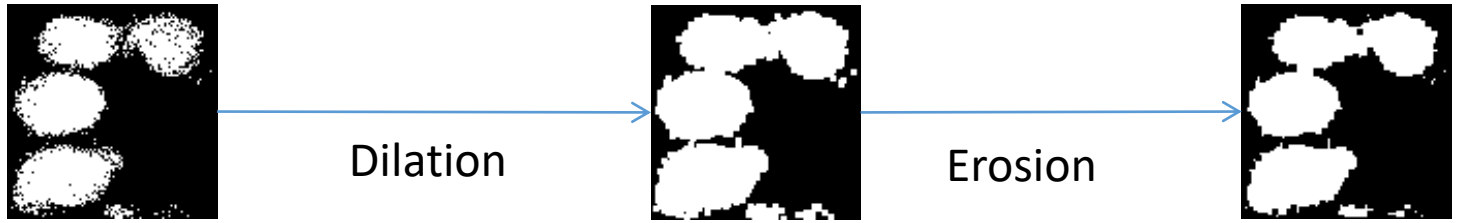
- Erosion: Set all pixels to black which have at least one black neighbor.



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



- Closing: Dilation + Erosion



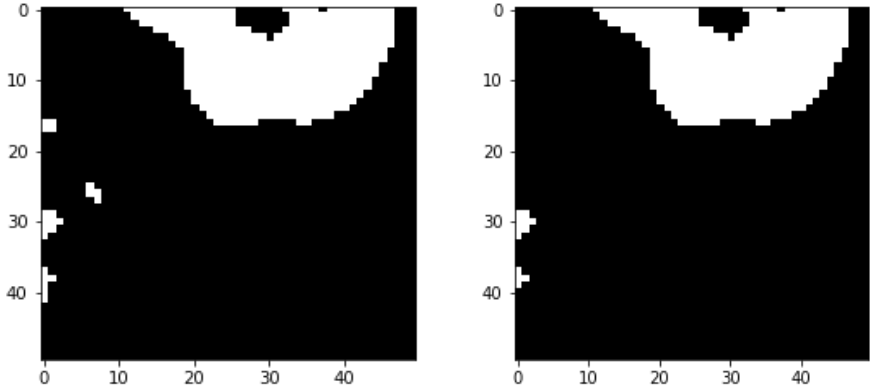
- Opening: Erosion + Dilation

Refining masks: Opening & Closing

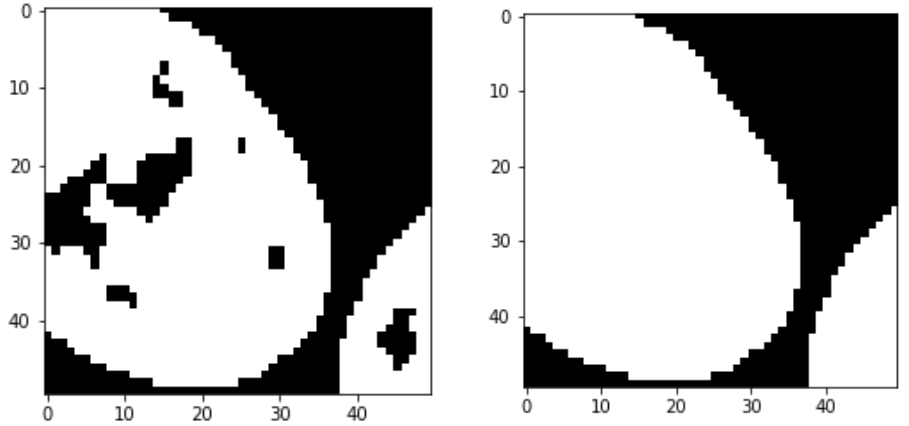
PoL Physics of Life TU Dresden

```
from skimage import morphology
```

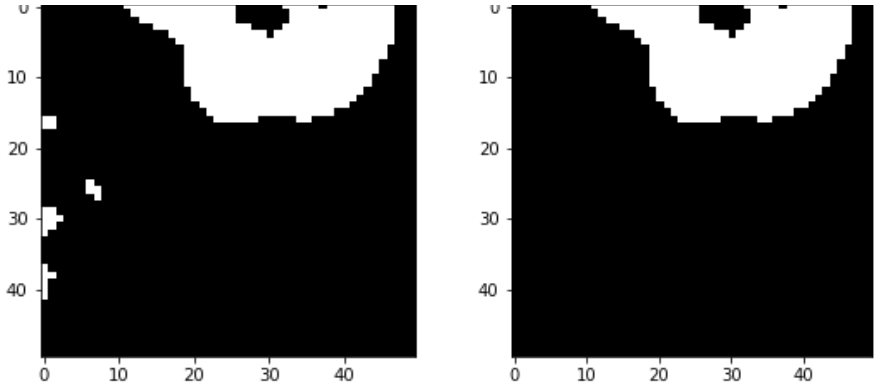
```
opened = morphology.binary_opening(binary)
```



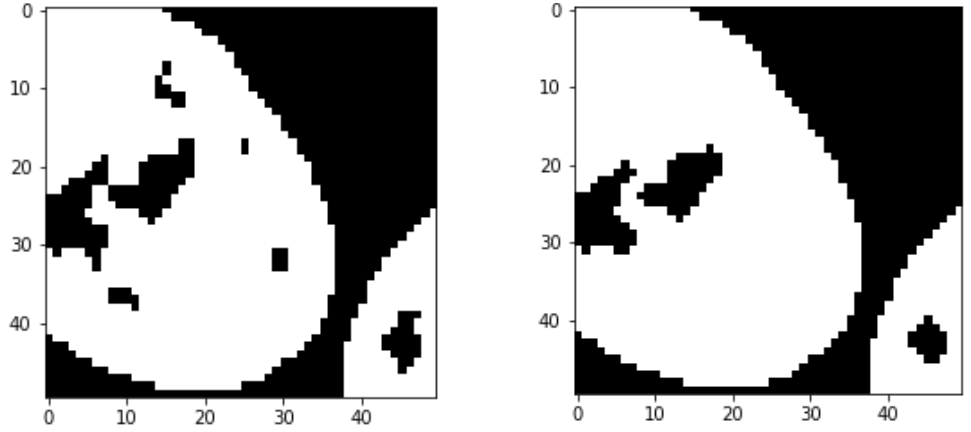
```
closed = morphology.area_closing(binary, area_threshold=100)
```



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

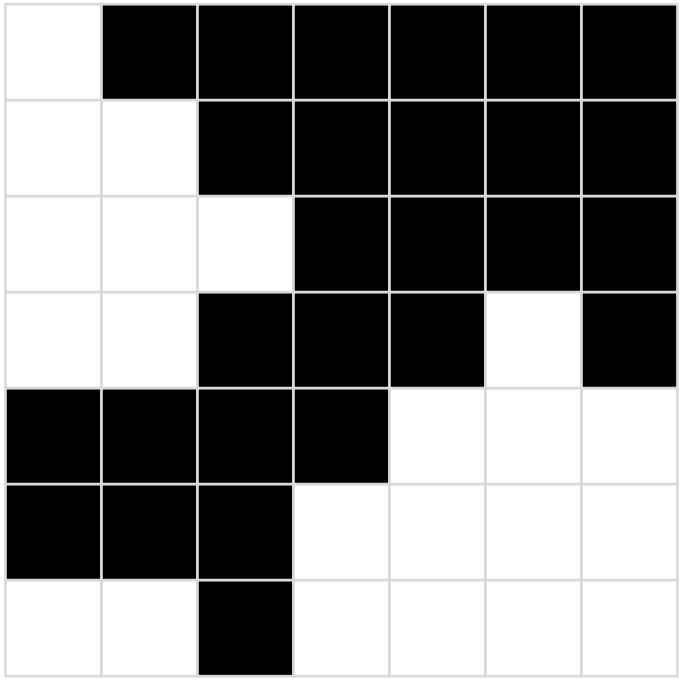


```
closed = morphology.binary_closing(binary)
```



Connected component labelling

- In order to allow the computer differentiating objects, connected component 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.



CCA

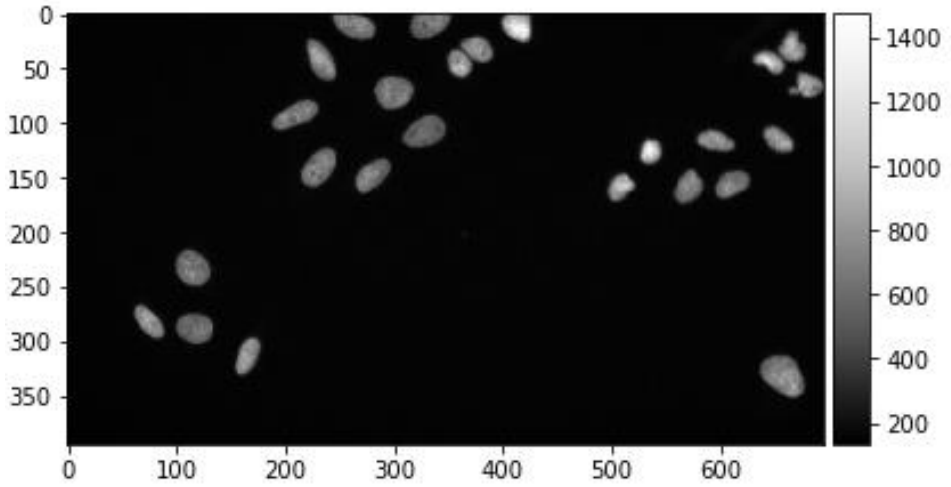
1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	1	0	0	0	3	0
0	0	0	0	3	3	3
0	0	0	3	3	3	3
2	2	0	3	3	3	3

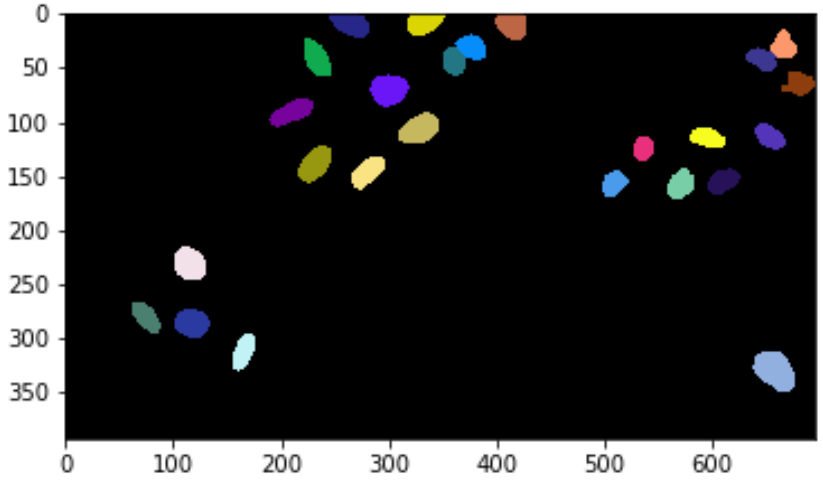
Short-cuts: Voronoi-Otsu-Labeling

- Gaussian-Blur
- Otsu-Thrsholding
- Spot-detection
- Watershed on the binary image

… in a single line of code:

```
segmented = nsbatwm.voronoi_otsu_labeling(input_image,
                                          spot_sigma=5,
                                          outline_sigma=1
                                         )
segmented
```





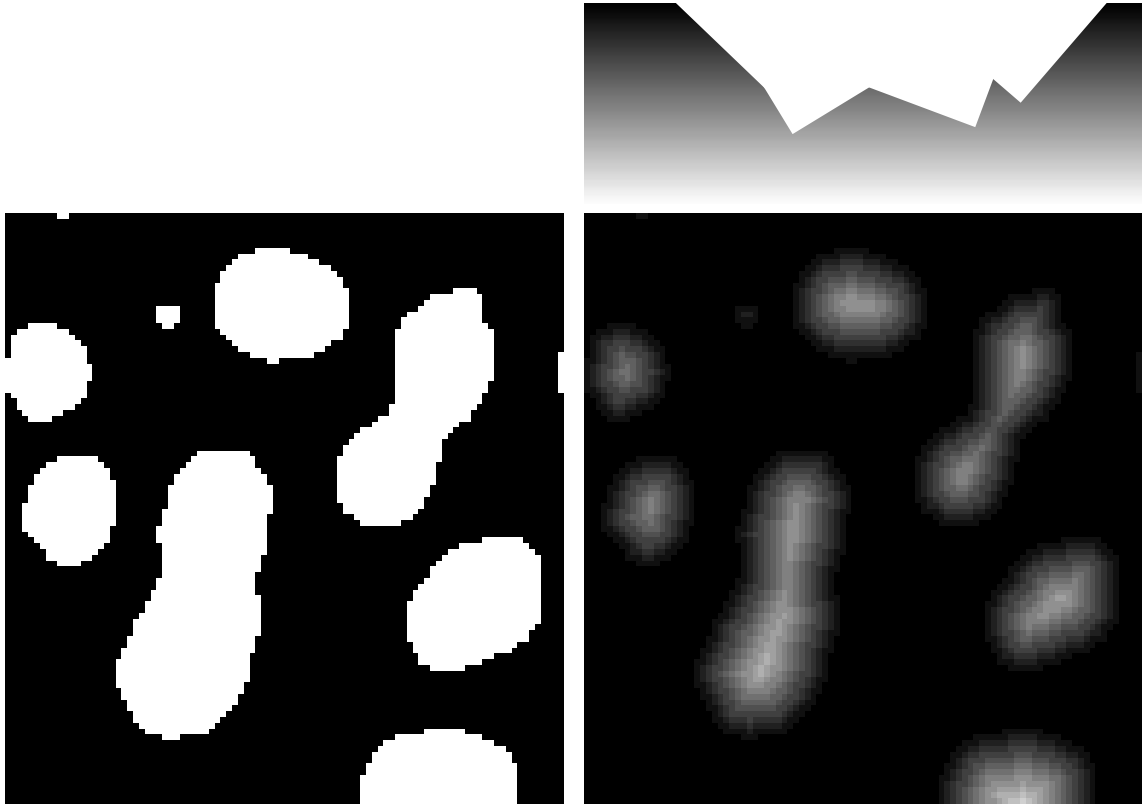
nsbatwm made image

shape	(395, 695)
dtype	int32
size	1.0 MB
min	0
max	25

https://haesleinhuepf.github.io/BioImageAnalysisNotebooks/20_image_segmentation/11_voronoi_otsu_labeling.html
https://www.youtube.com/watch?v=evgRgDfVuEc

Watershed

- The watershed algorithm for binary images allows cutting one object into tow where it’s reasonable.

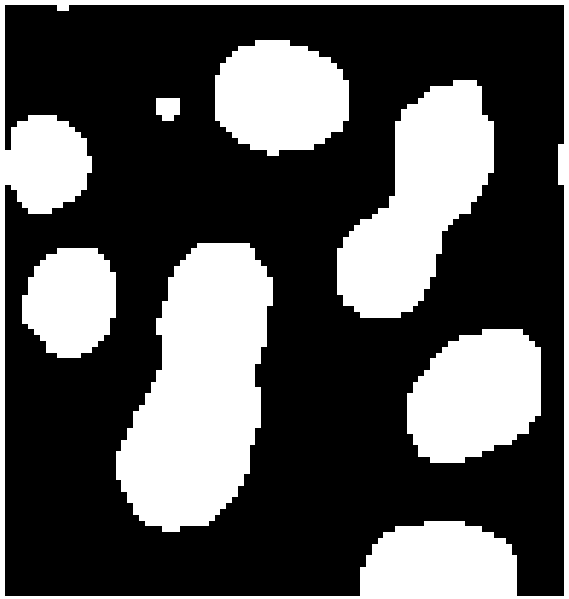


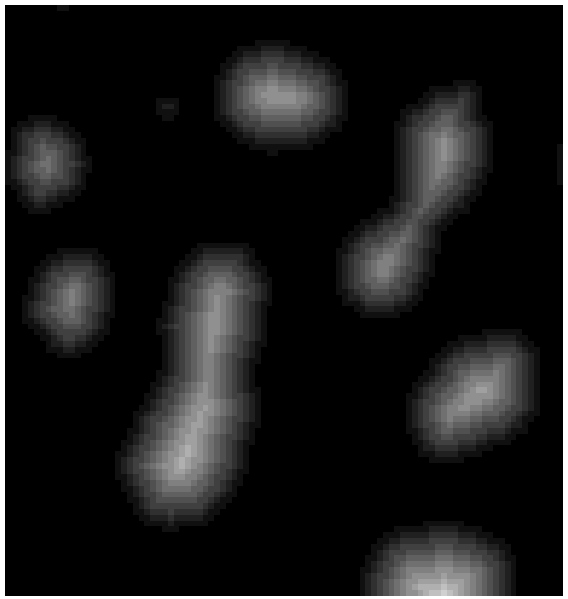
Binary segmentation

Distance map

Watershed

- The watershed algorithm for binary images allows cutting one object into tow where it’s reasonable.



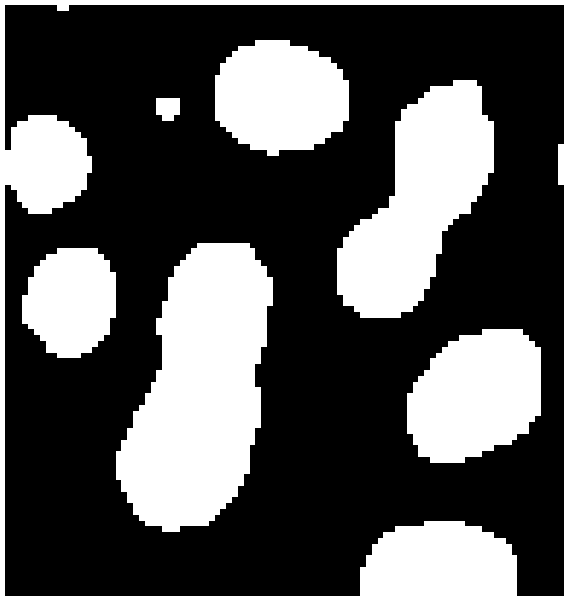


Binary segmentation

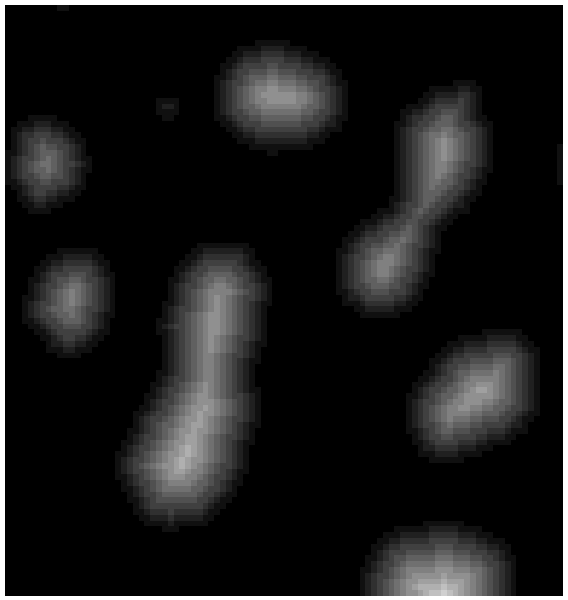
Distance map

Watershed

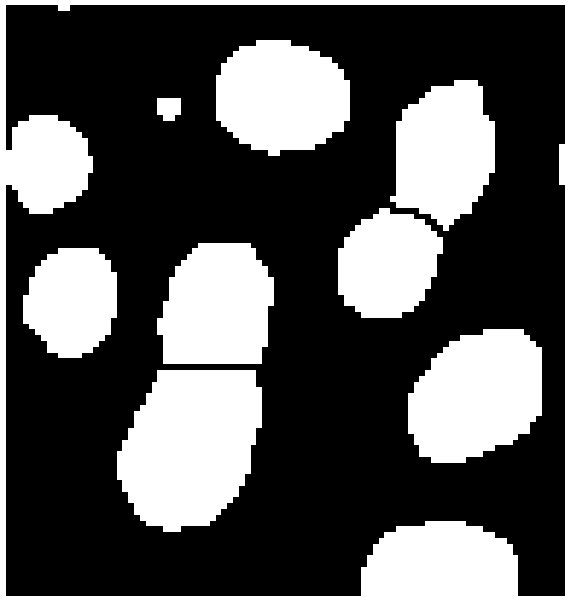
- The watershed algorithm for binary images allows cutting one object into tow where it’s reasonable.
- The watersheds are made from binary images. The algorithm does not take the original image into account!



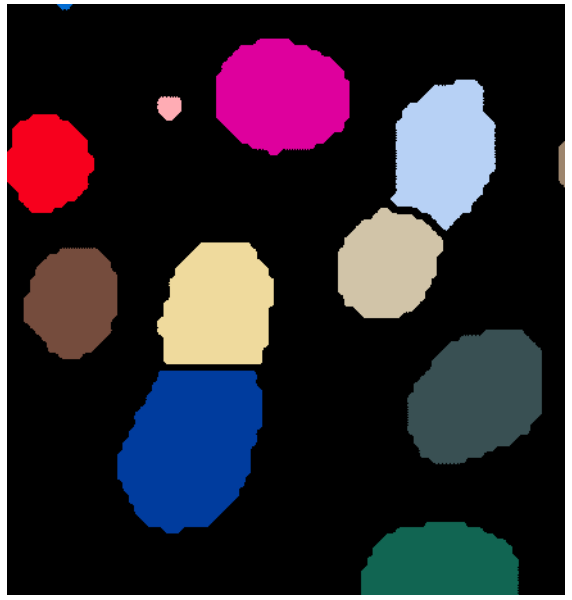
Binary segmentation



Distance map



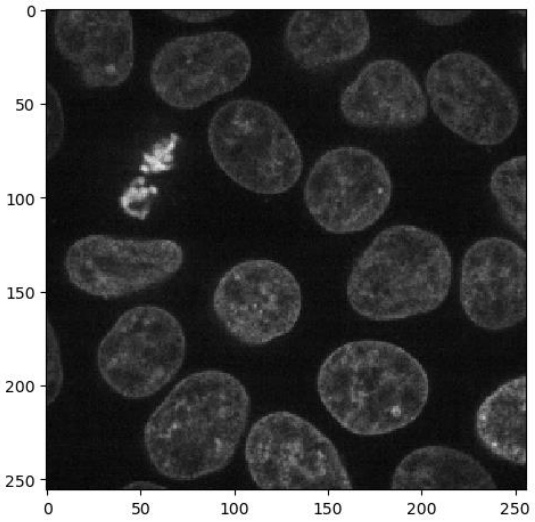
Binary watershed

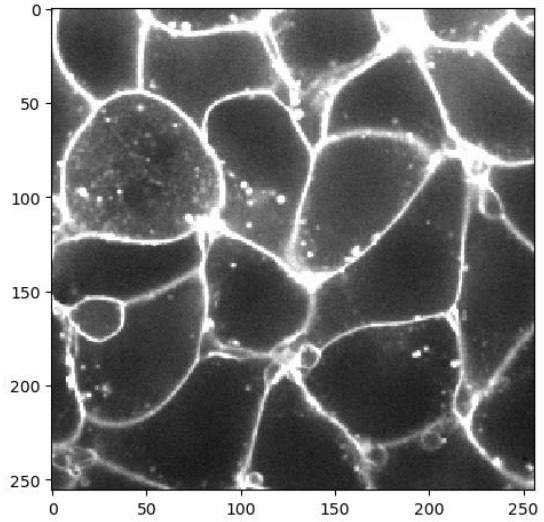


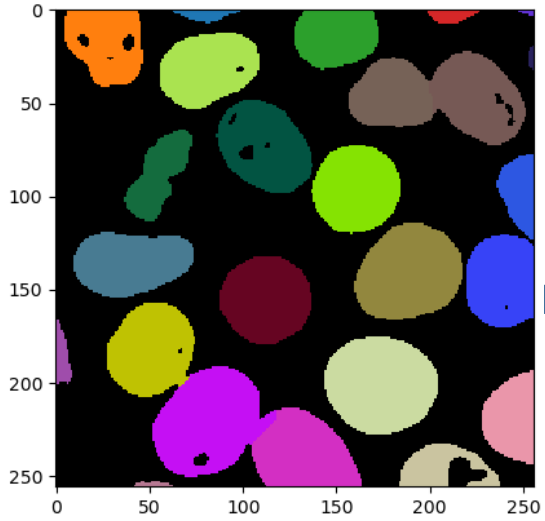
Labeled watershed

Watershed

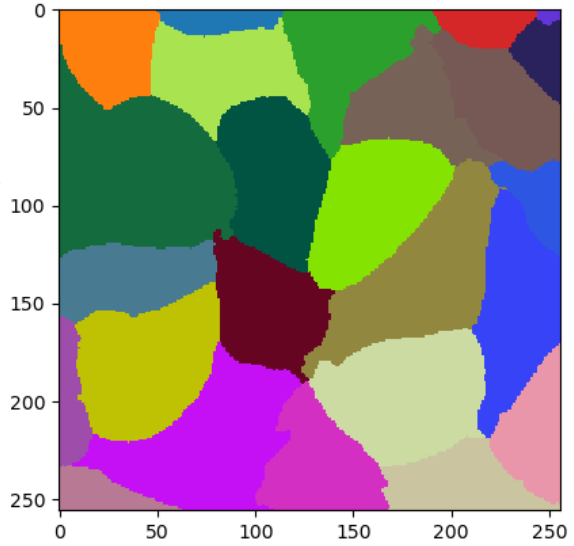
- … in Python practice







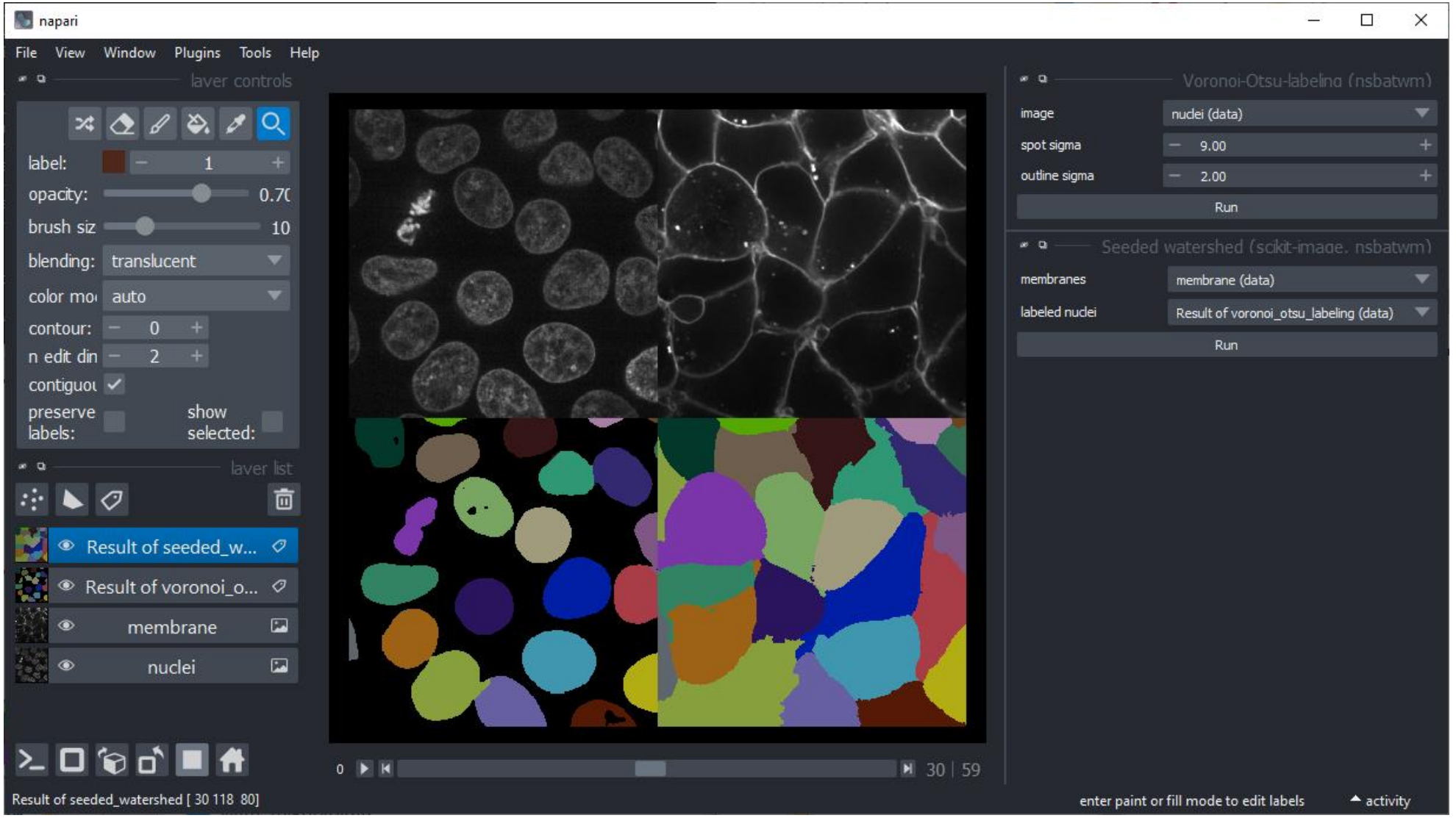

```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)

labeled_cells
```



https://www.napari-hub.org/plugins/napari-segment-blobs-and-things-with-membranes#seeded-watershed

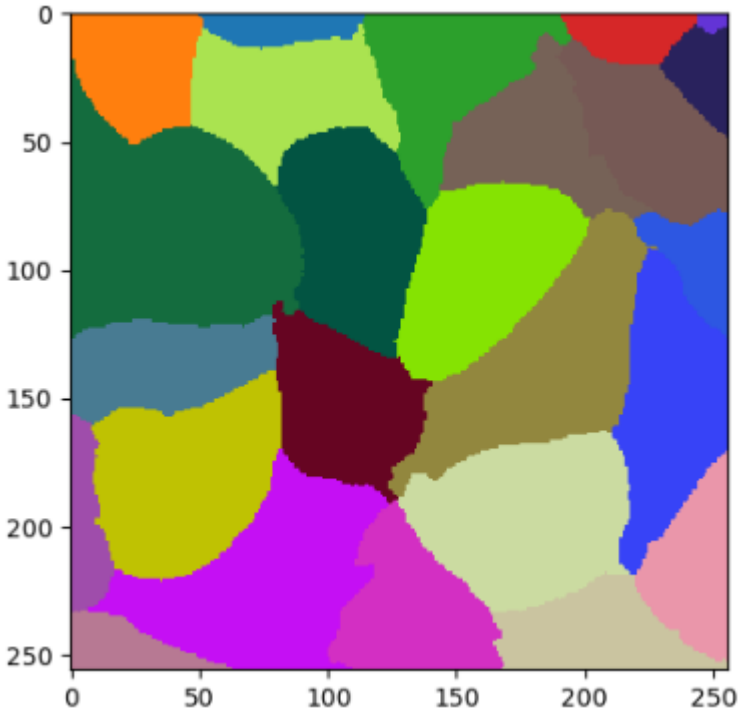
Watershed

- … in Napari practice: Tools > Segmentation / Labeling menu



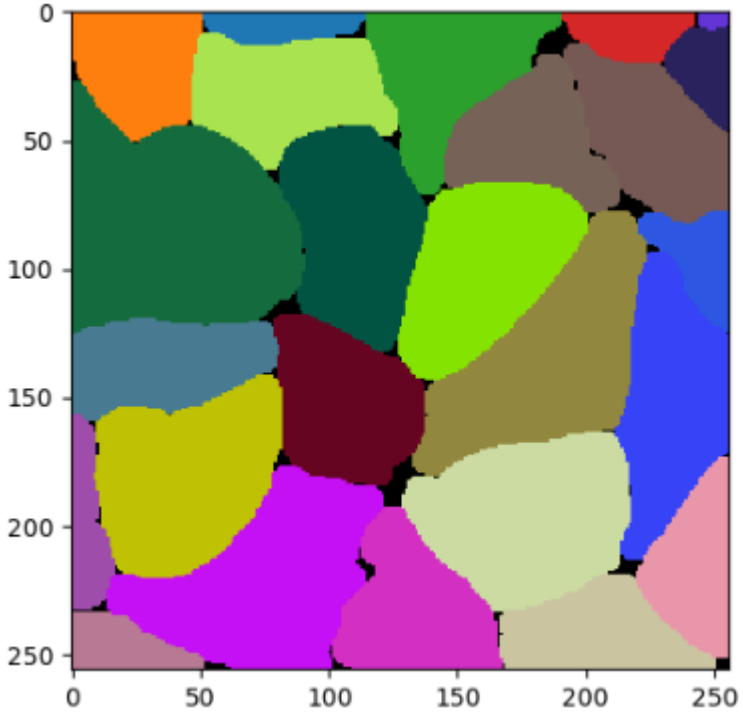
Label post-processing / morphological operations

- … similarily to morphological operations on binary images



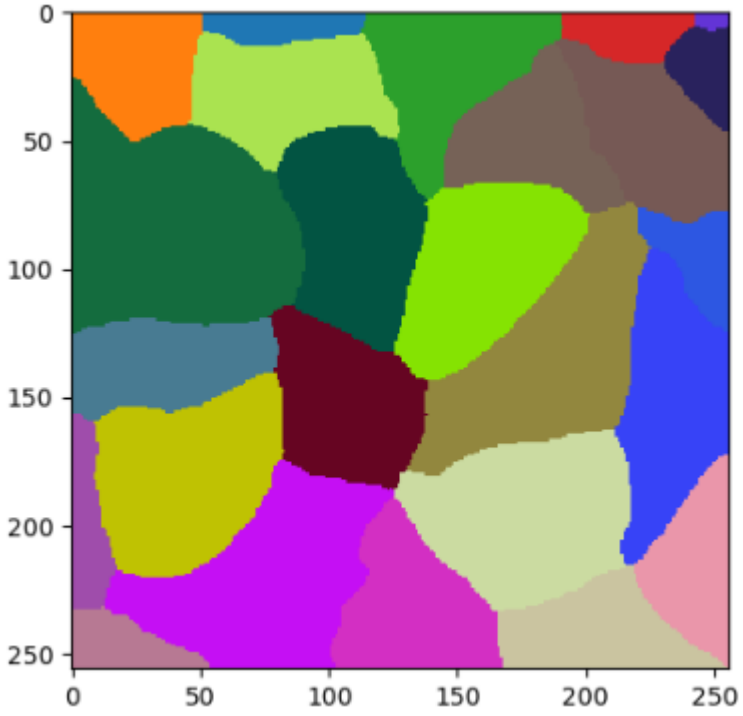
Original

```
cle.opening_labels(label_image, radius=5)
```



Opening Labels

```
cle.smooth_labels(label_image, radius = 5)
```



Smoothing Labels

Label post-processing / morphological operations

PoL Physics of Life TU Dresden

- In Napari menu Tools > Segmentation post-processing > Smooth labels (clEsperanto)