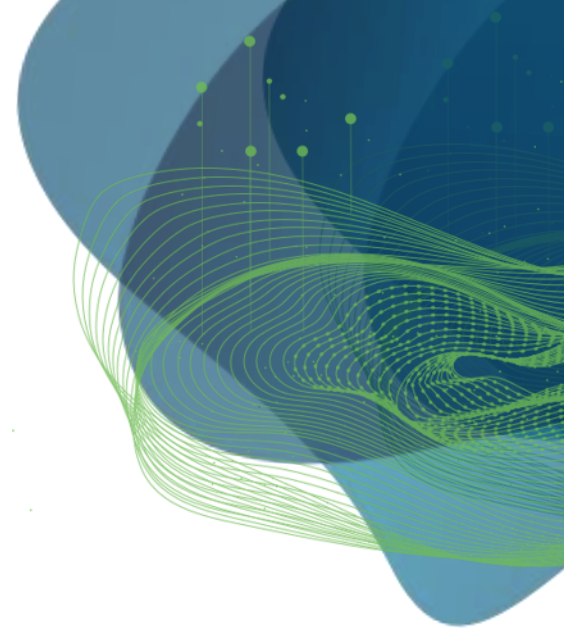


# Image segmentation

Robert Haase

Using materials from Marcelo Leomil Zoccoler and Johannes Soltwedel,  
PoL, TU Dresden



GEFÖRDERT VOM



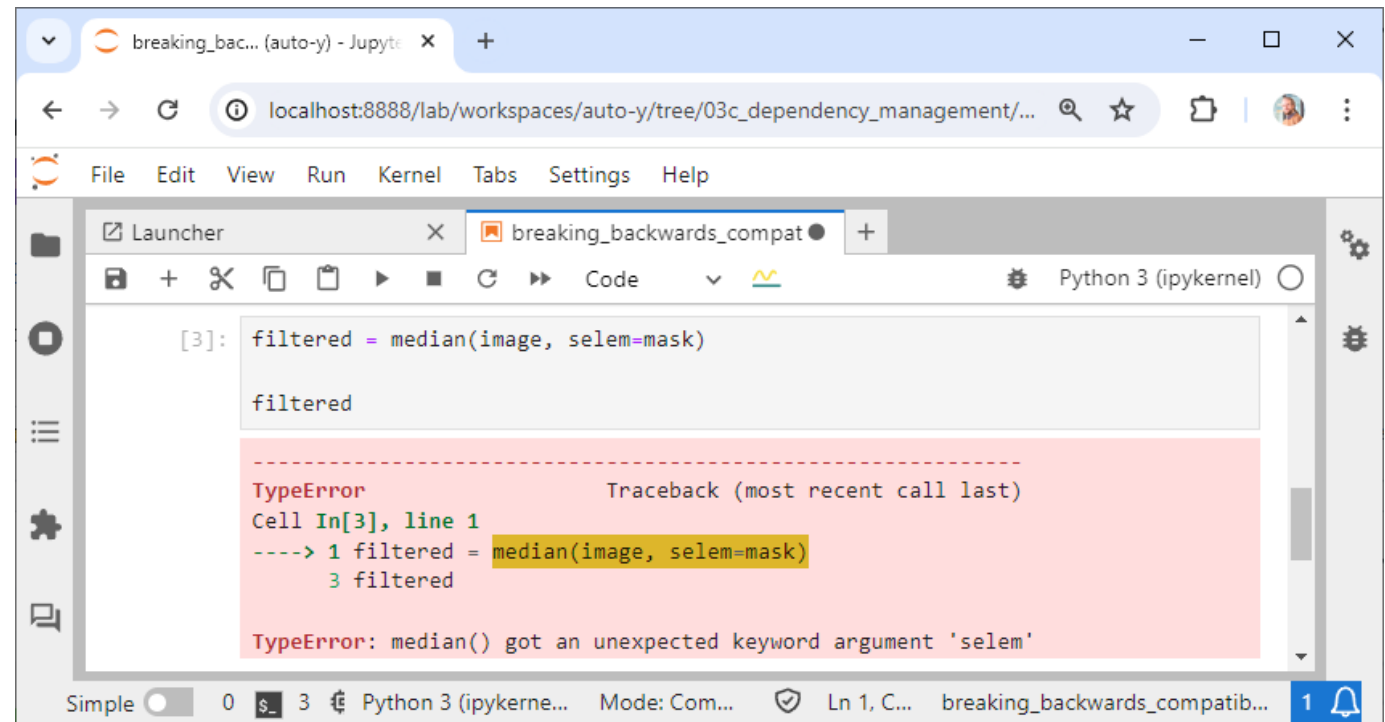
Bundesministerium  
für Bildung  
und Forschung

Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

# Recap

How can one solve this problem?

- A) By modifying the code
- B) By not modifying the code



```
[3]: filtered = median(image, selem=mask)

filtered

-----
TypeError                                Traceback (most recent call last)
Cell In[3], line 1
----> 1 filtered = median(image, selem=mask)
      3 filtered

TypeError: median() got an unexpected keyword argument 'selem'
```

# Quiz (recap)

- Which of the following is a band-pass filter?

Gaussian



Median



Top-hat



Difference  
of Gaussian



# Quiz (recap)

- Which of the following is a denoising filter?

Gaussian



Median



Top-hat

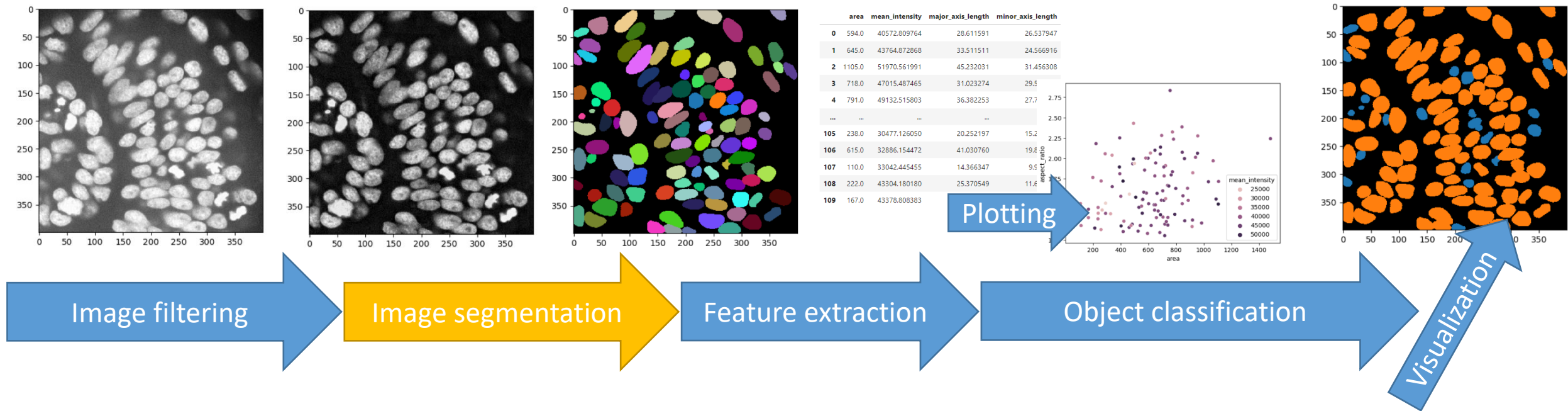


Difference  
of Gaussian



# Lecture overview: Bio-image Analysis

- Image Data Analysis workflows
- Goal: Quantify observations, substantiate conclusions with numbers



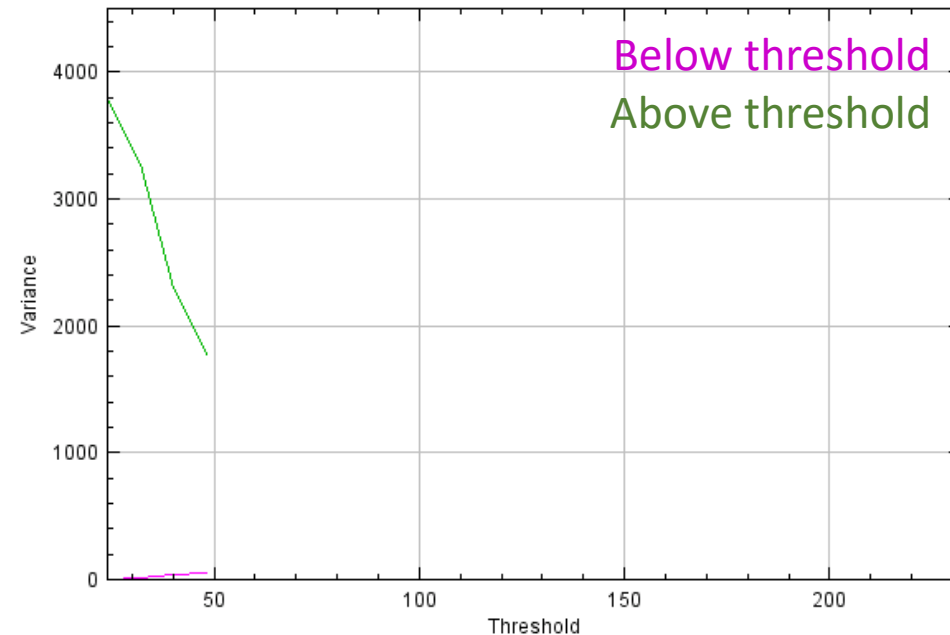
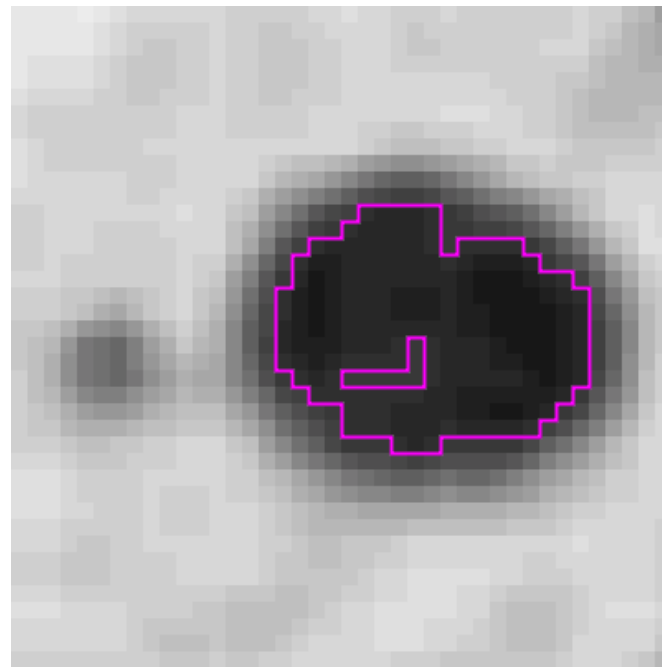
# Thresholding: Otsu's method

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

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

$$\bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$  ... 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



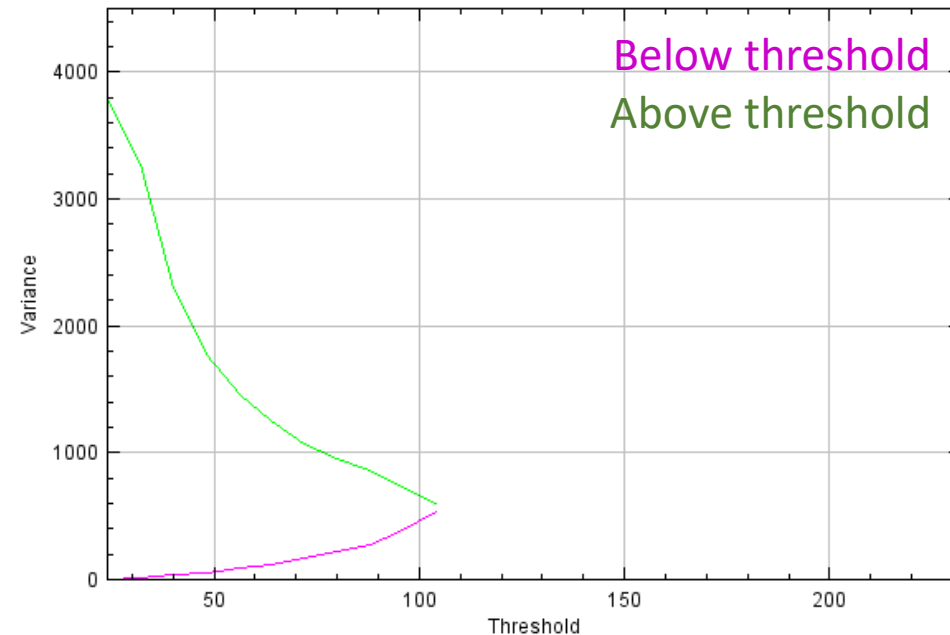
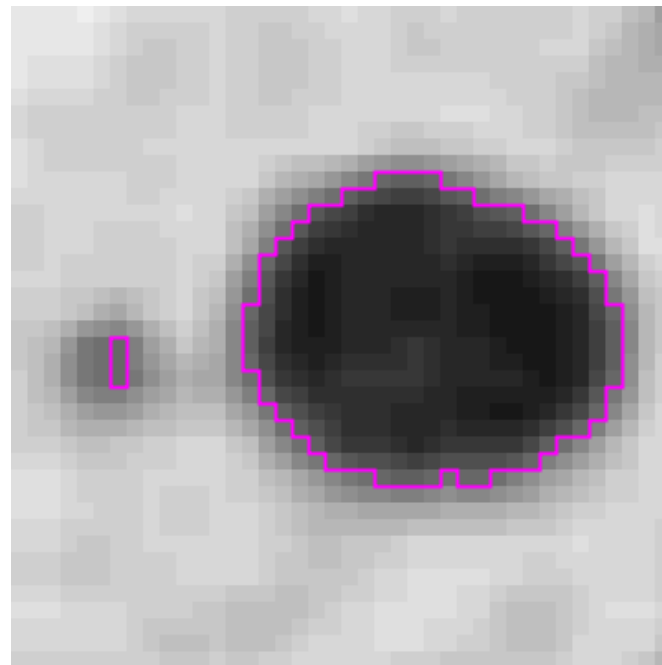
# Thresholding: Otsu's method

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

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

$$\bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$  ... 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





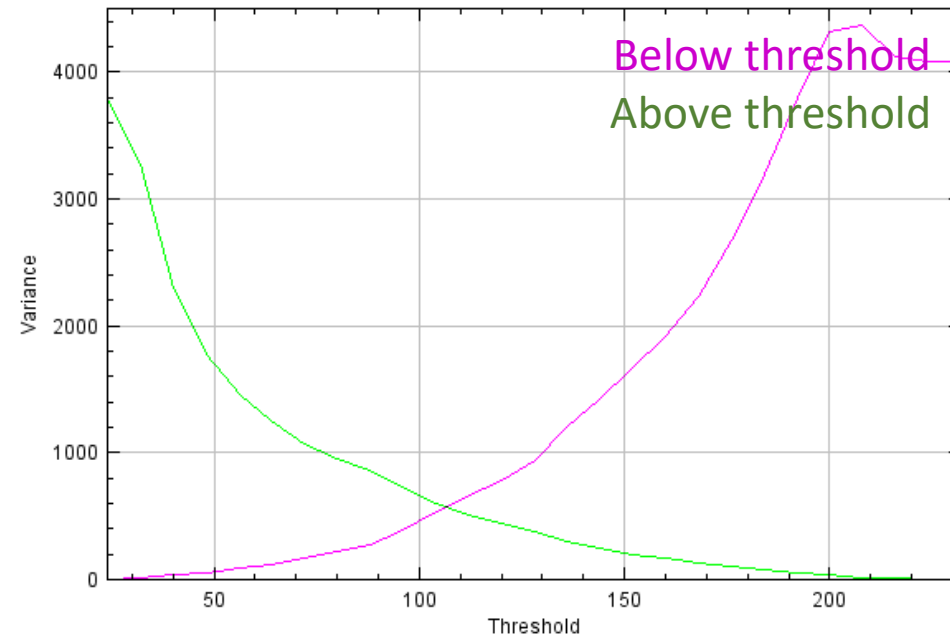
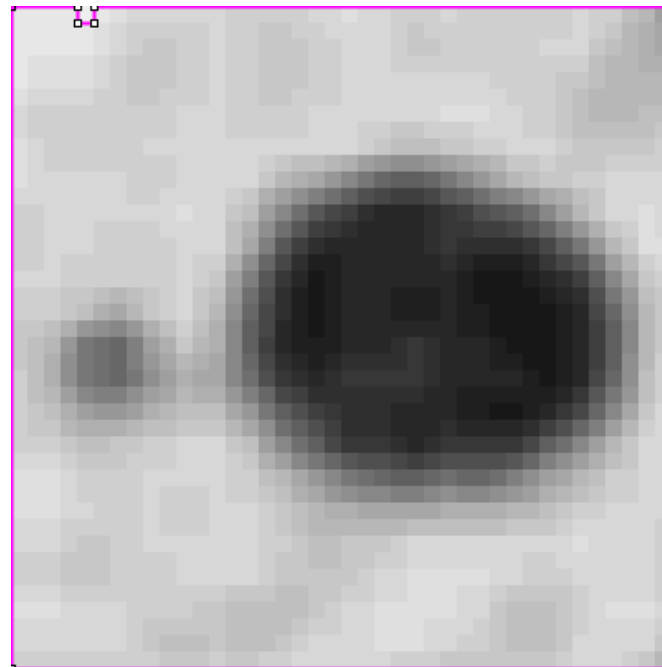
# Thresholding: Otsu's method

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

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

$$\bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$  ... 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



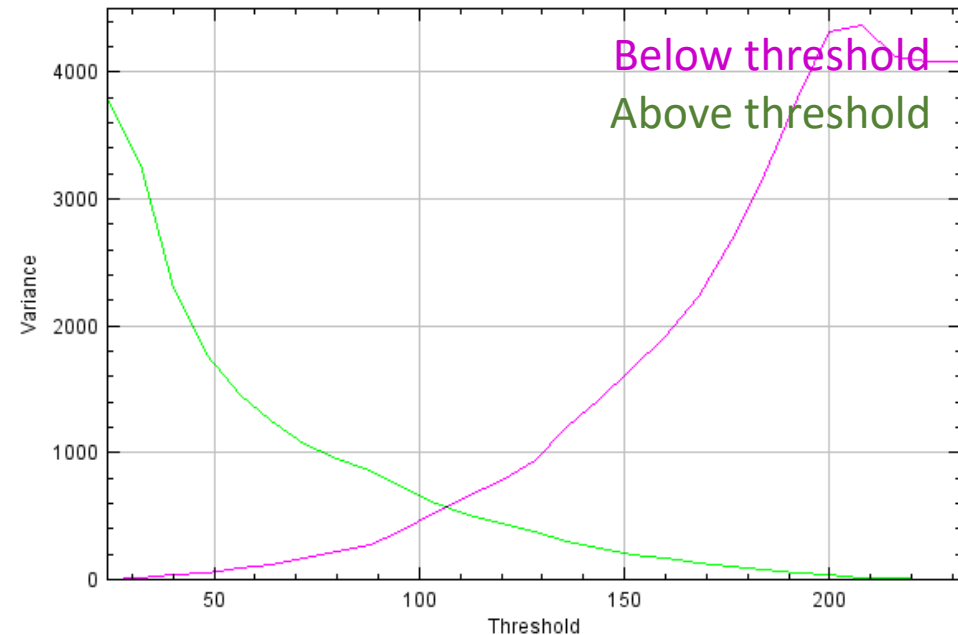
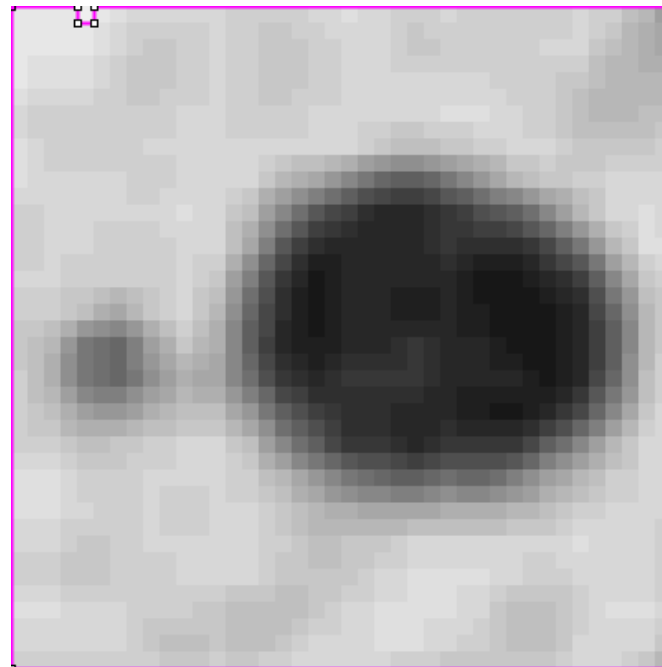


# Thresholding: Otsu's method

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

- Weighted (!)  
sum variance

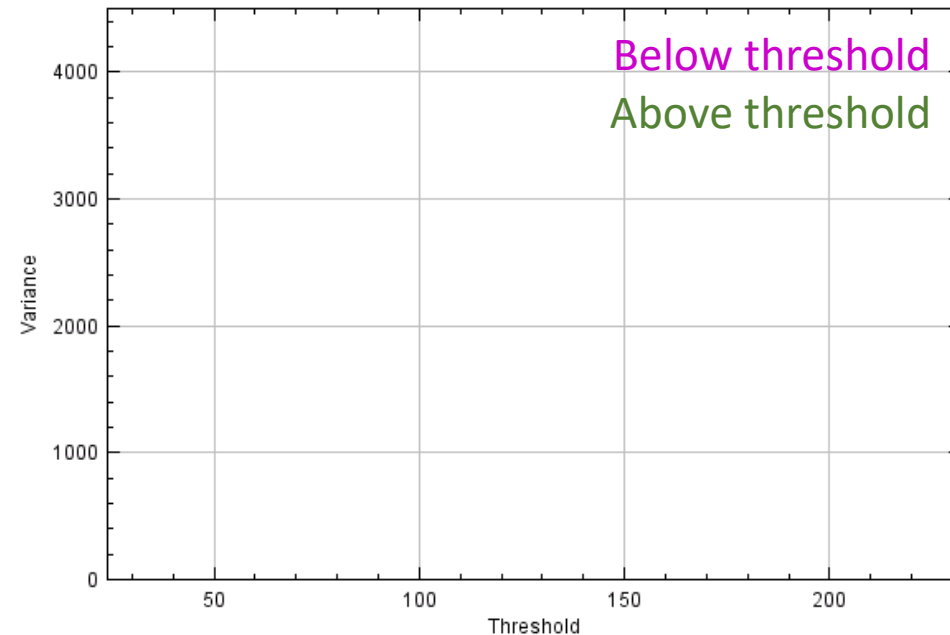
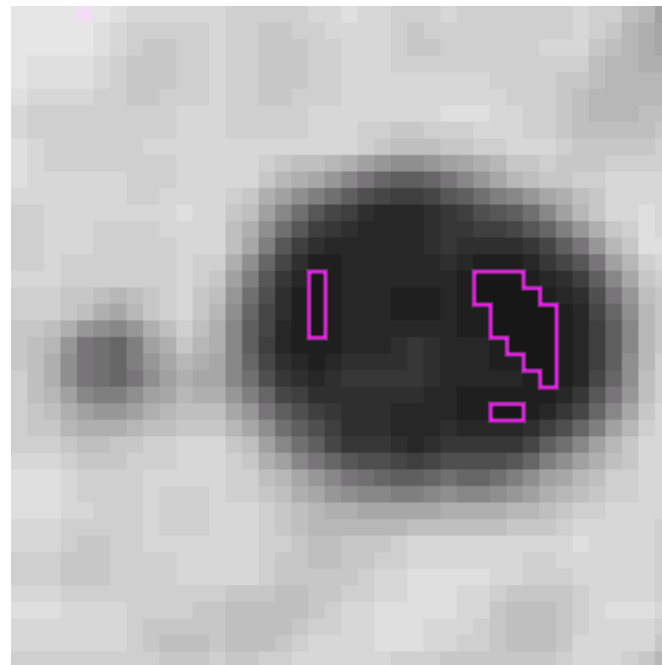
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



# Thresholding: Otsu's method

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

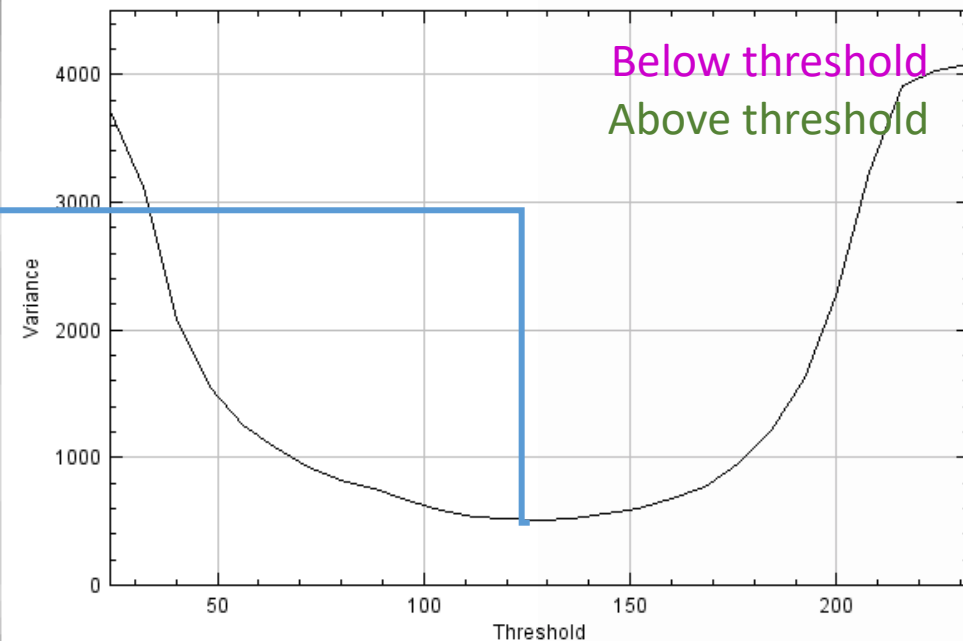
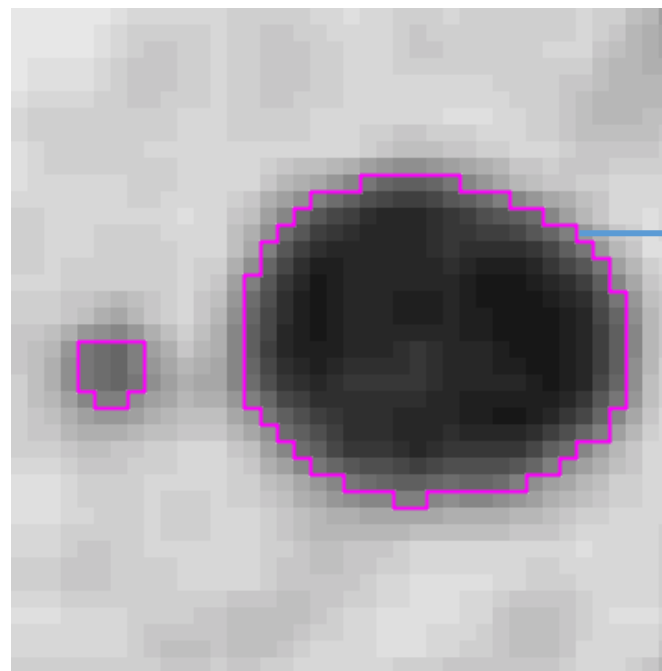
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



# Thresholding: Otsu's method

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

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



# Thresholding: Citing

- Cite the thresholding method of your choice properly

*“We segmented the cell nuclei in the images using Otsu’s 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

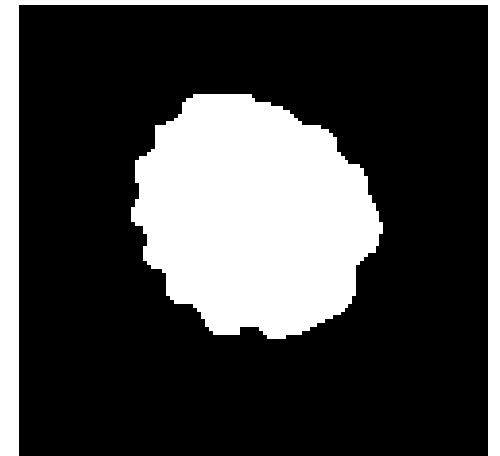
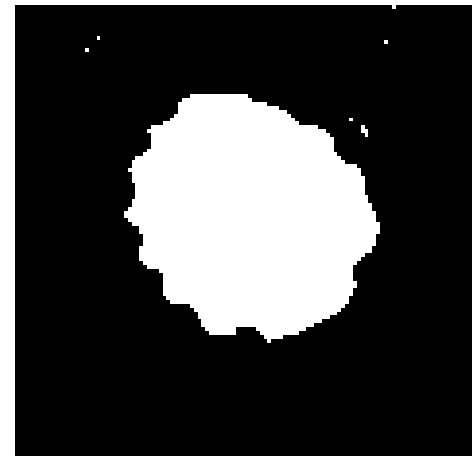
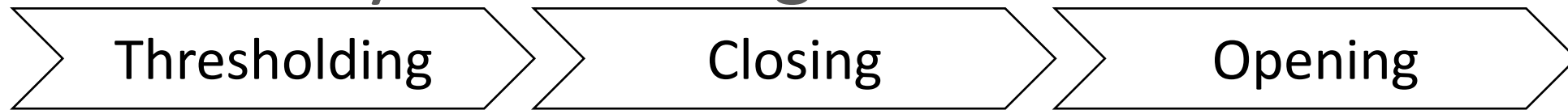
## A Threshold Selection Method from Gray-Level Histograms

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 zeroth- and the first-order cumulative moments of the gray-level histogram. It is straightforward to extend the method to multithreshold problems. Several experimental results are also presented to support the validity of the method.

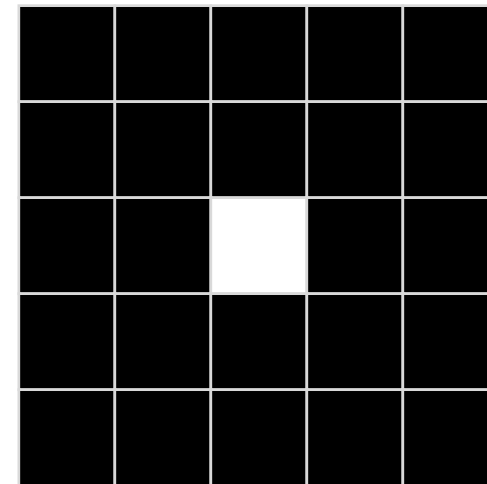
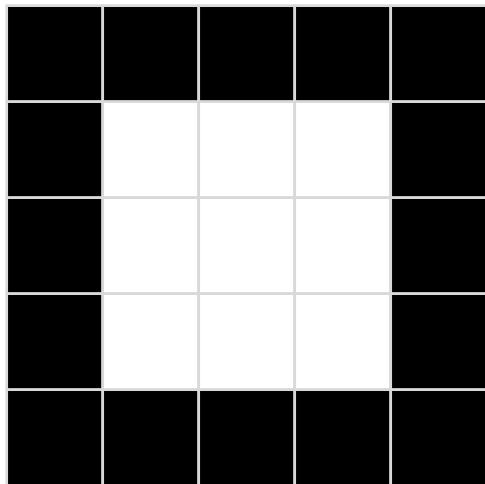
# Refining masks

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



# Erosion

- Erosion: Every pixel with at least one black neighbor becomes black.



# Quiz

- Binary erosion is identical with which filter?

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0



0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

Mean

Median

Minimum

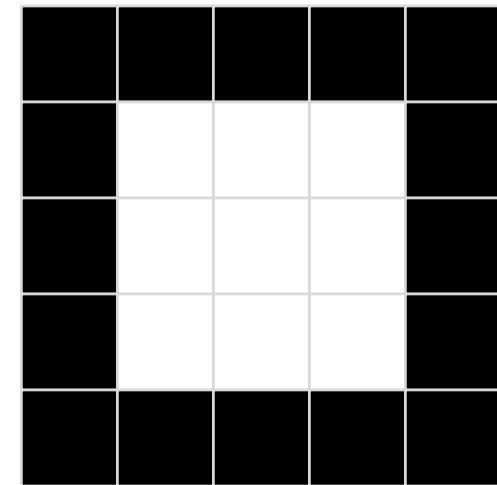
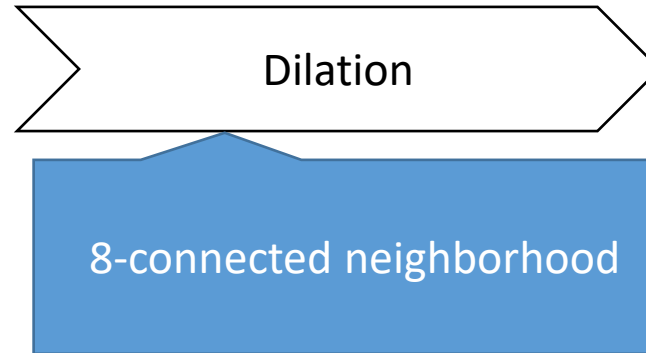
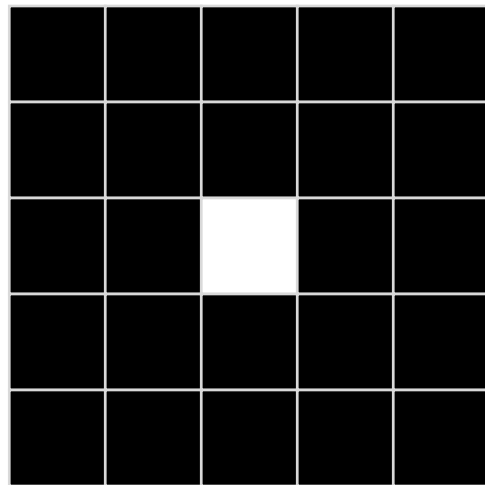
Maximum





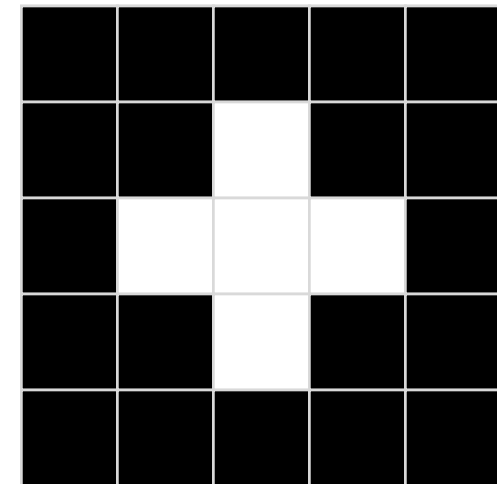
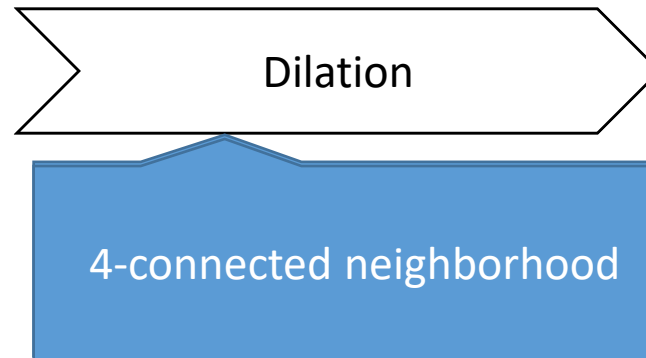
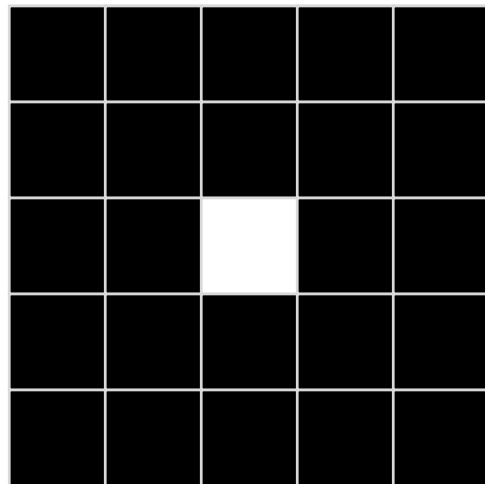
# Dilation

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



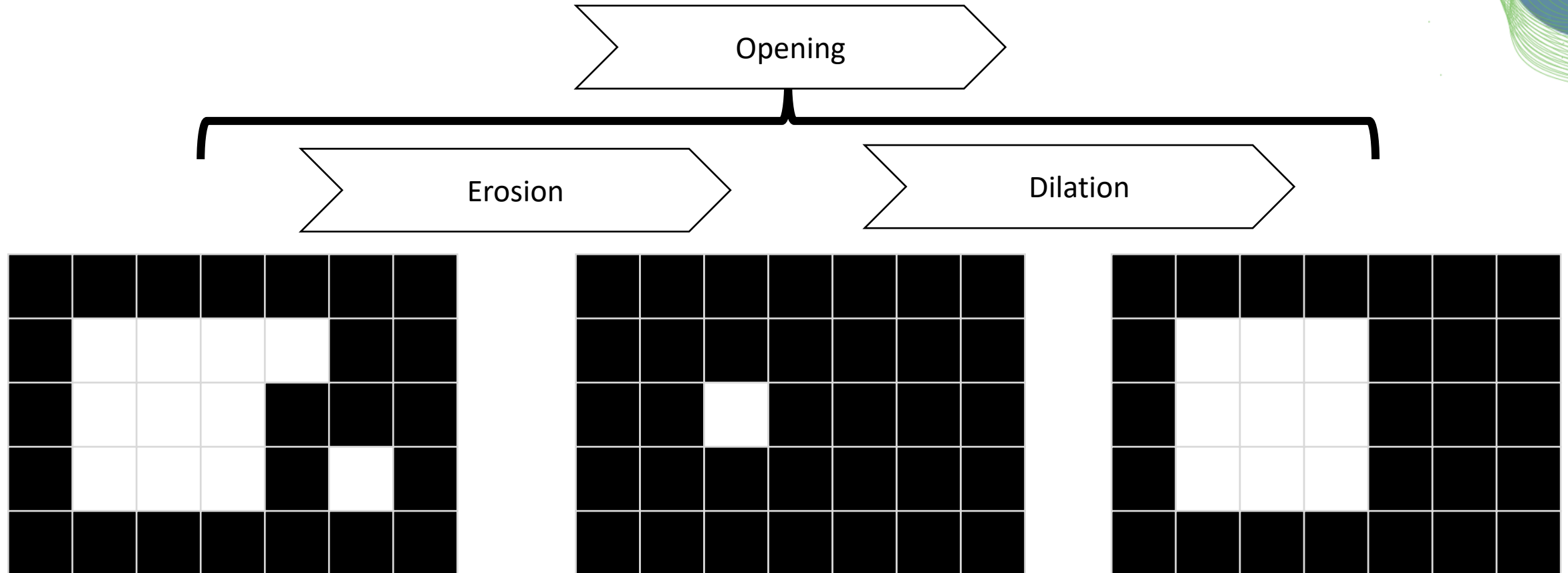
# Dilation

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



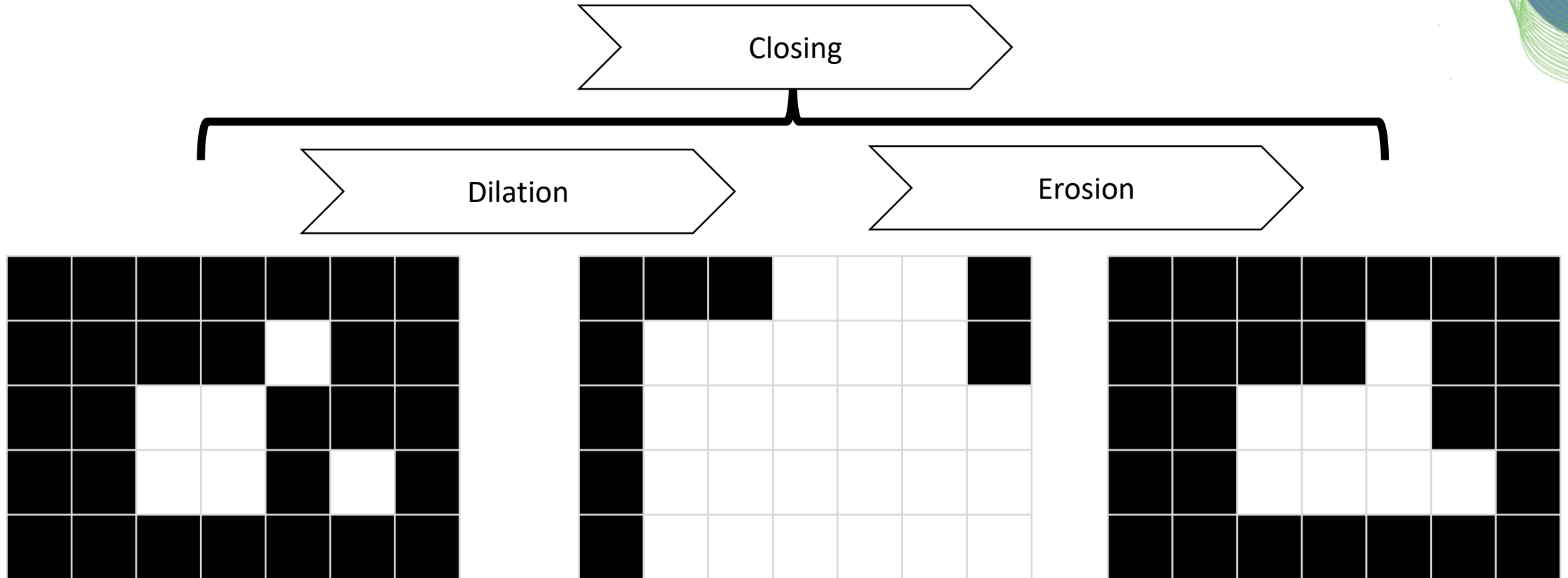
# Opening

- Erosion and dilation combined allow correcting outlines.



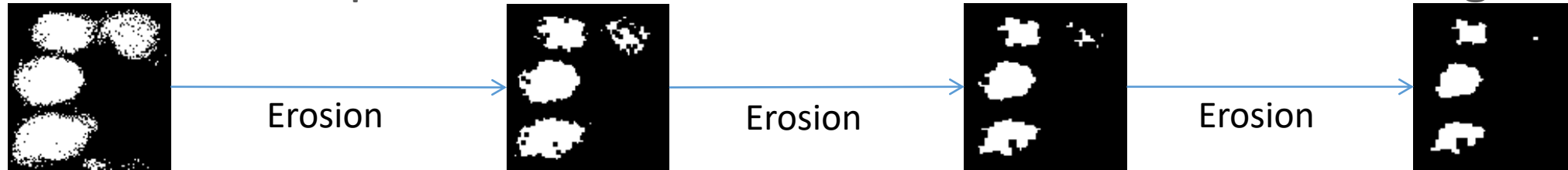
# Closing

- Erosion and dilation combined allow correcting outlines.

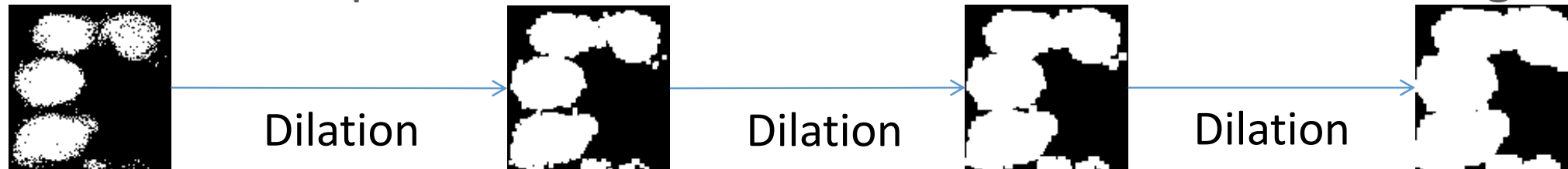


# Chaining erosion and dilation

- 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.



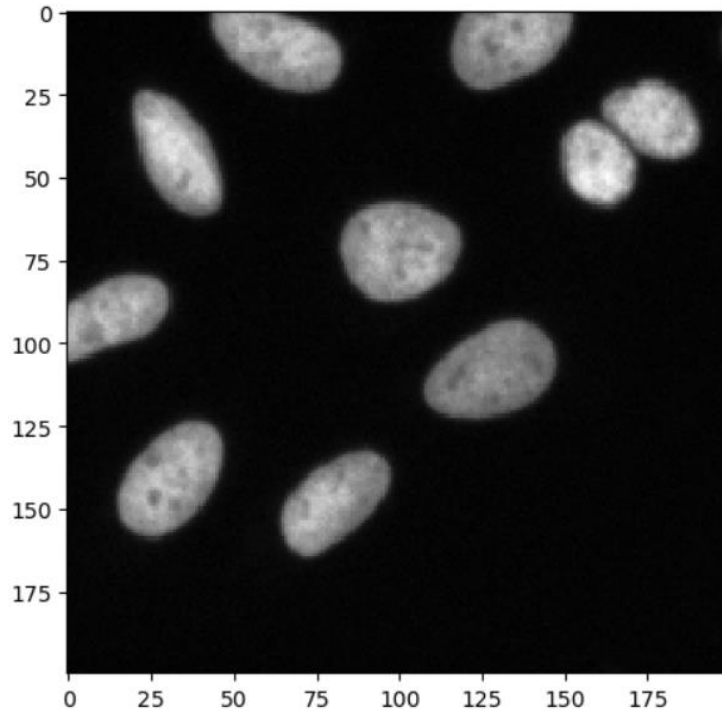
- Closing: Dilation + Erosion



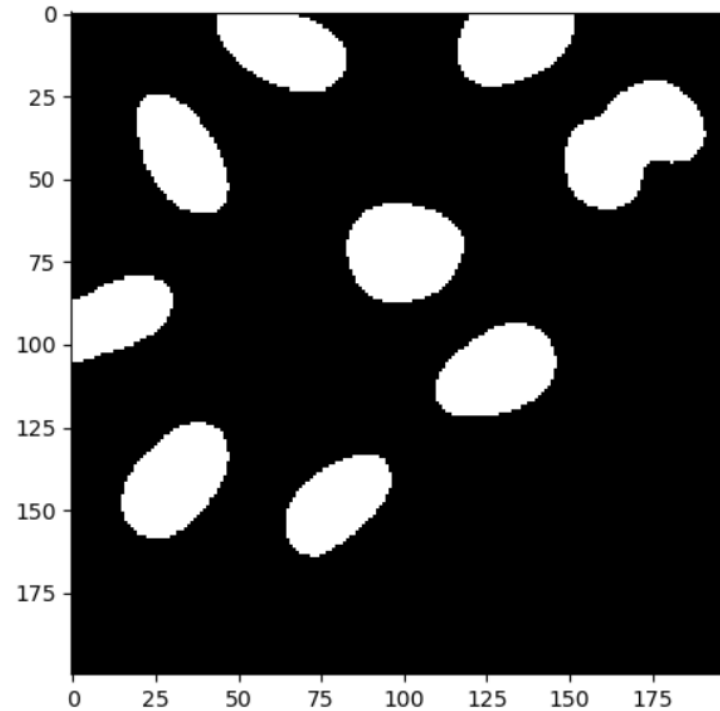
- Opening: Erosion + Dilation

# Terminology

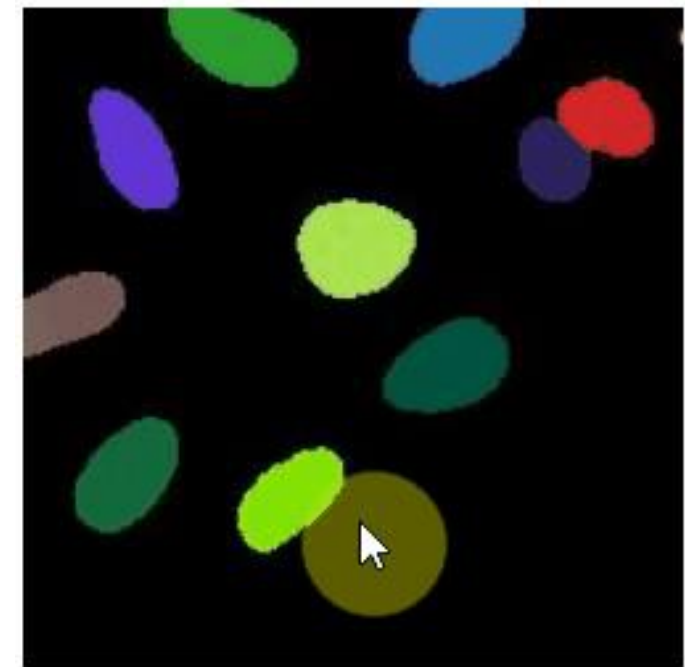
Intensity image



Binary image

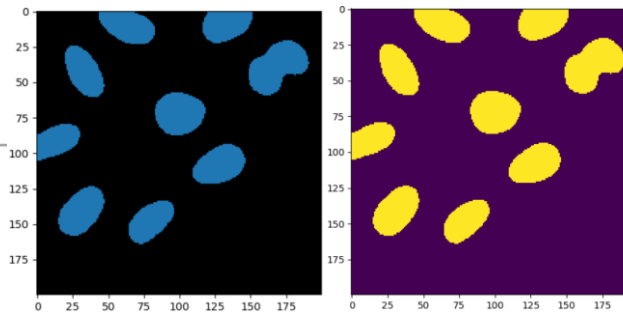


Label image



$[y=152, x=92] = 0$

No matter how they are displayed



# Terminology

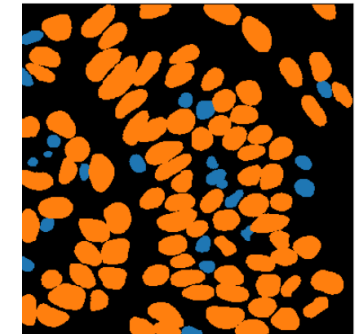
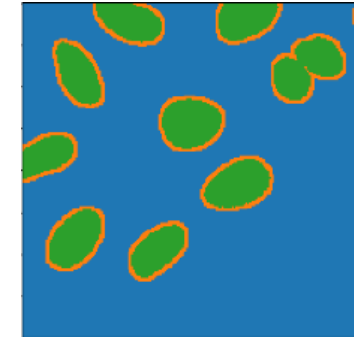
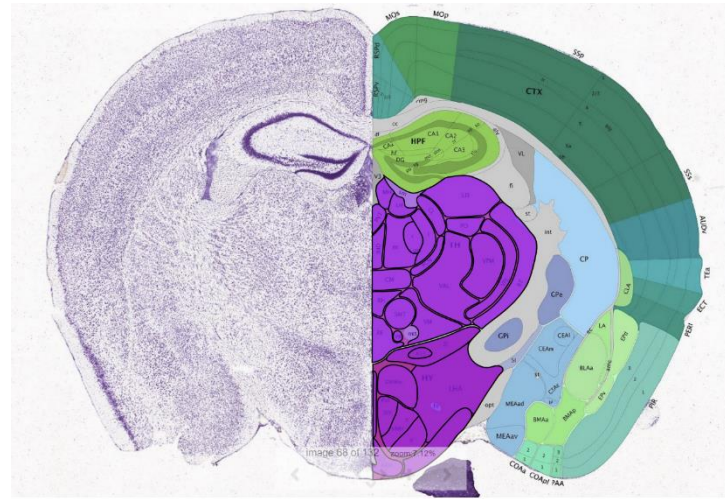
## Instance segmentation



Instances:

- Cells, nuclei, cats, dogs, cars, trees

## Semantic segmentation



Regions:

- Anatomical, geographical
- All pixels belonging to the same type of object have the same value



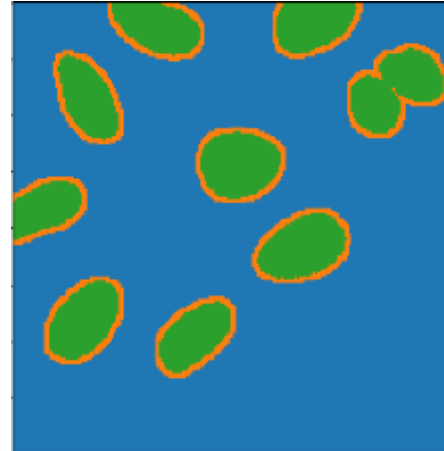
# Terminology

- Annotations are typically drawn by humans (e.g. to train machine learning models)

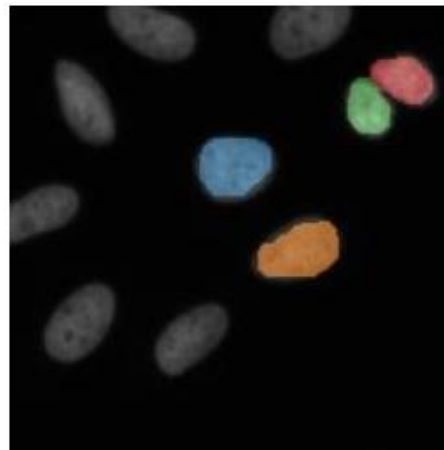
Instance segmentation



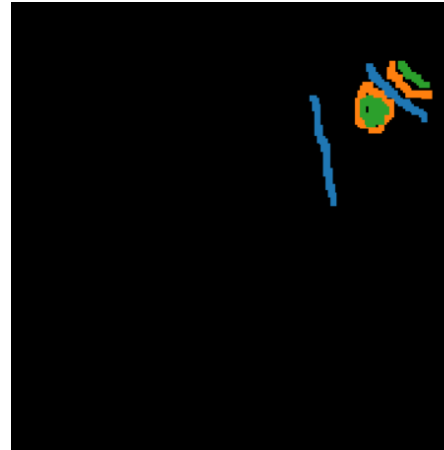
Semantic segmentation



Sparse instance annotation

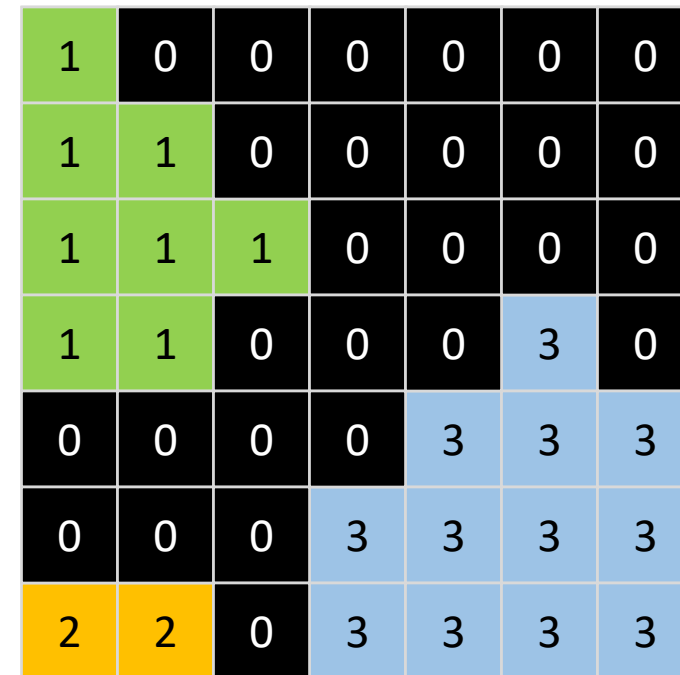
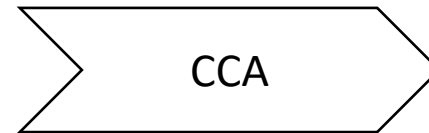
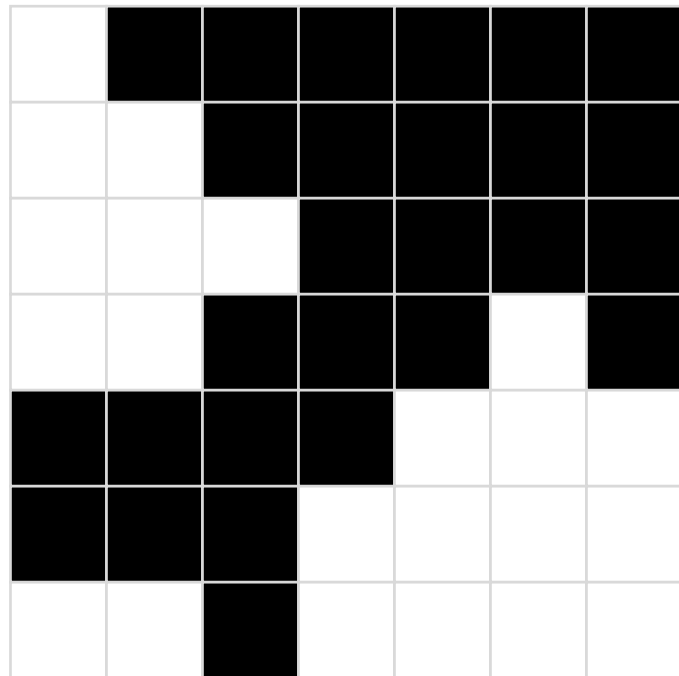


Sparse semantic annotation



# Connected component labelling

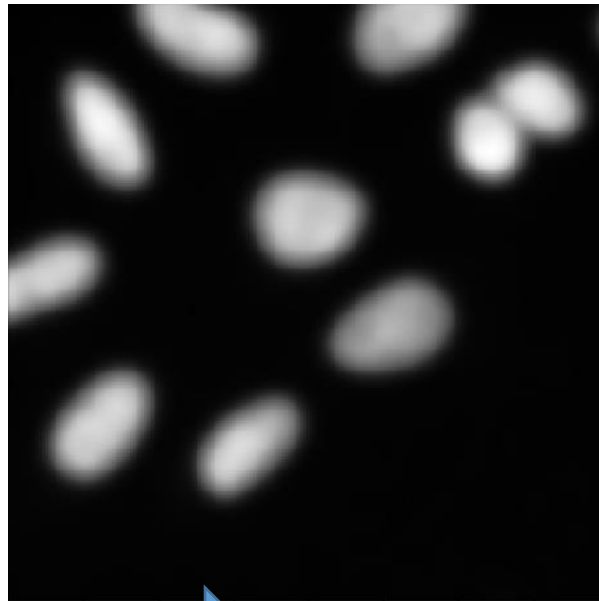
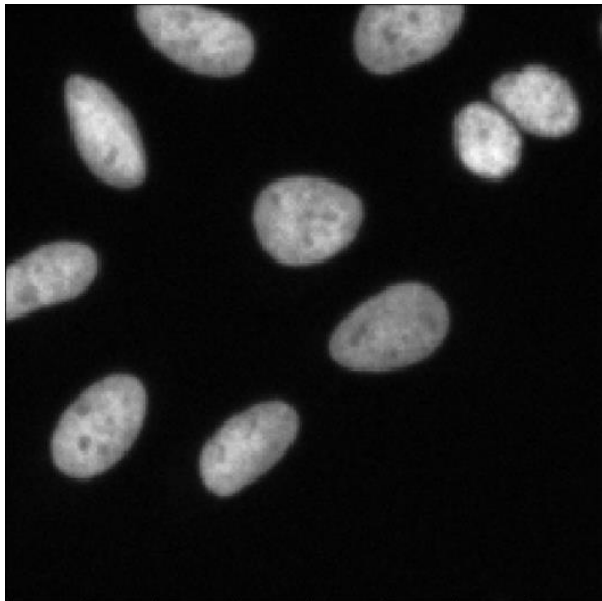
- In order to allow the computer differentiating objects, connected component analysis (CCA) is used to mark pixels belonging to different objects with different numbers
- Background pixels are marked with 0.
- The maximum intensity of a labelled map corresponds to the number of objects.



# Common image segmentation workflows

- Presumably the most common segmentation algorithm used for fluorescent microscopy images:
  - Gaussian blur, Otsu's Threshold, Connected Component Labeling

Limitation: Dense objects



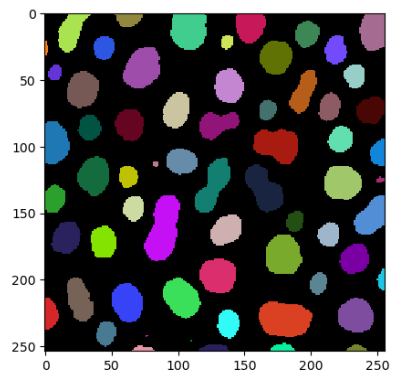
Denoising

Binarization

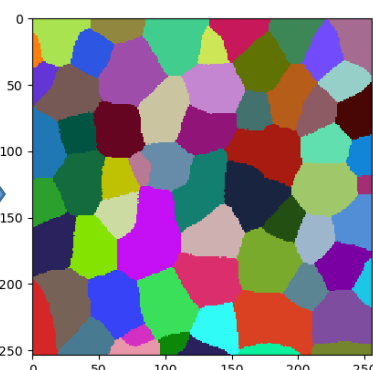
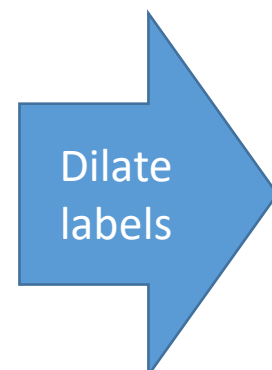
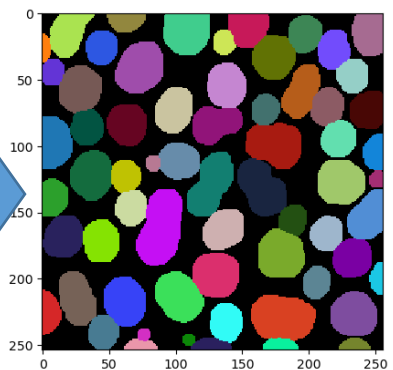
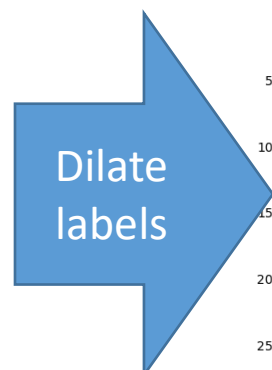
Labeling

# Voronoi-Tessellation

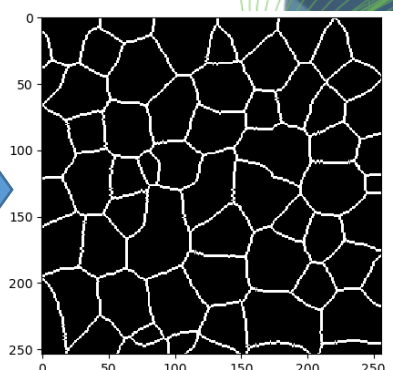
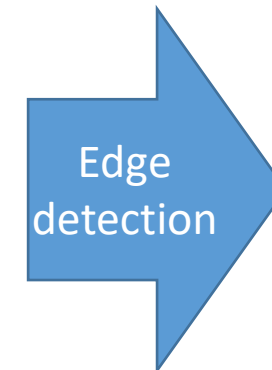
- For separating objects using spatial constraints (not intensity-based)



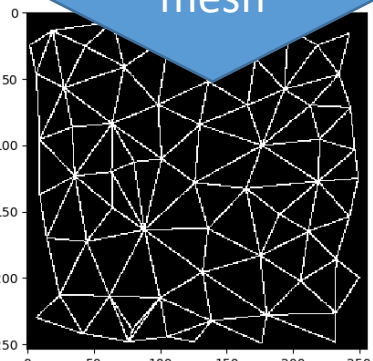
Label-image



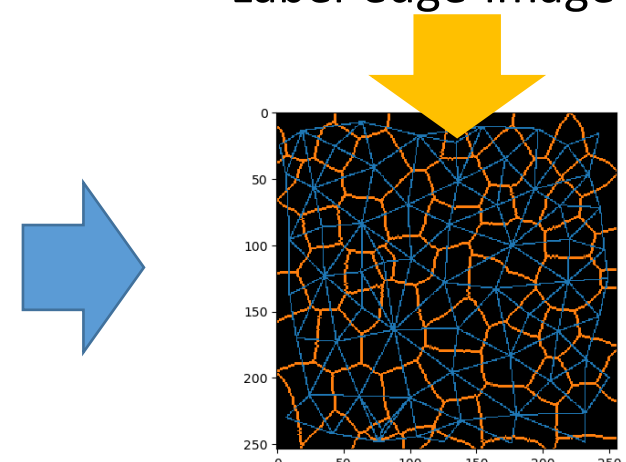
Voronoi-label-image



Label-edge-image



Touching neighbor mesh

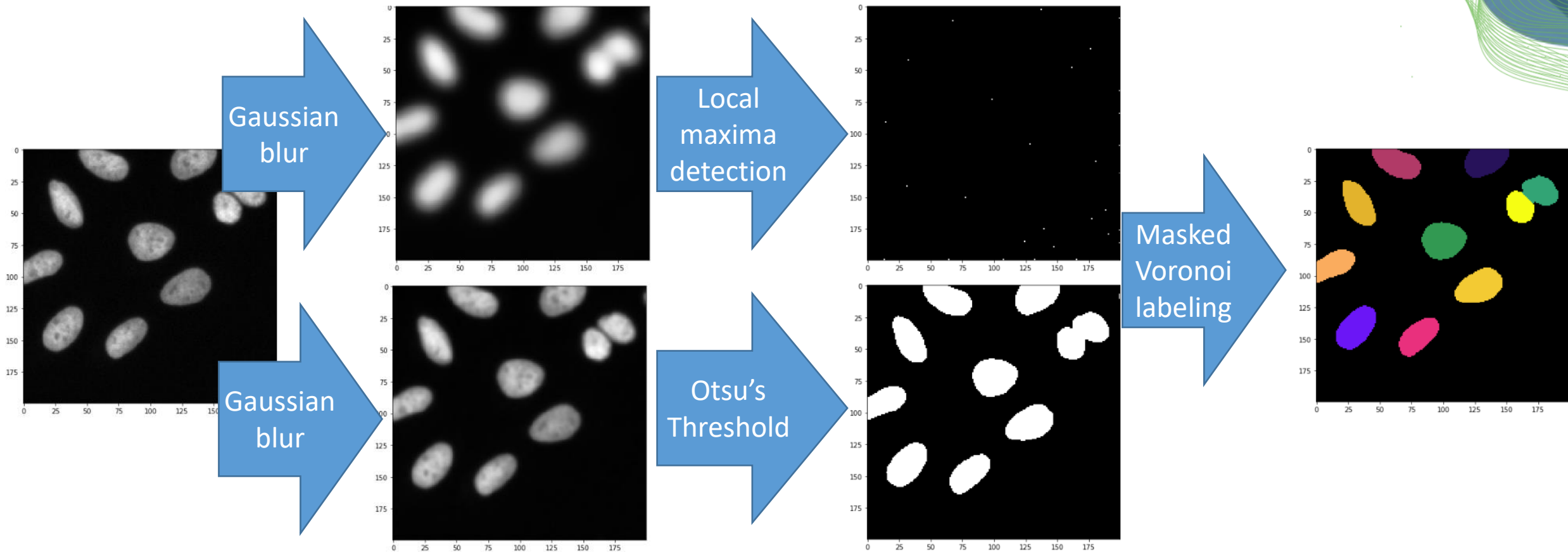


Voronoi-Tessellation

Delauney-Triangulation

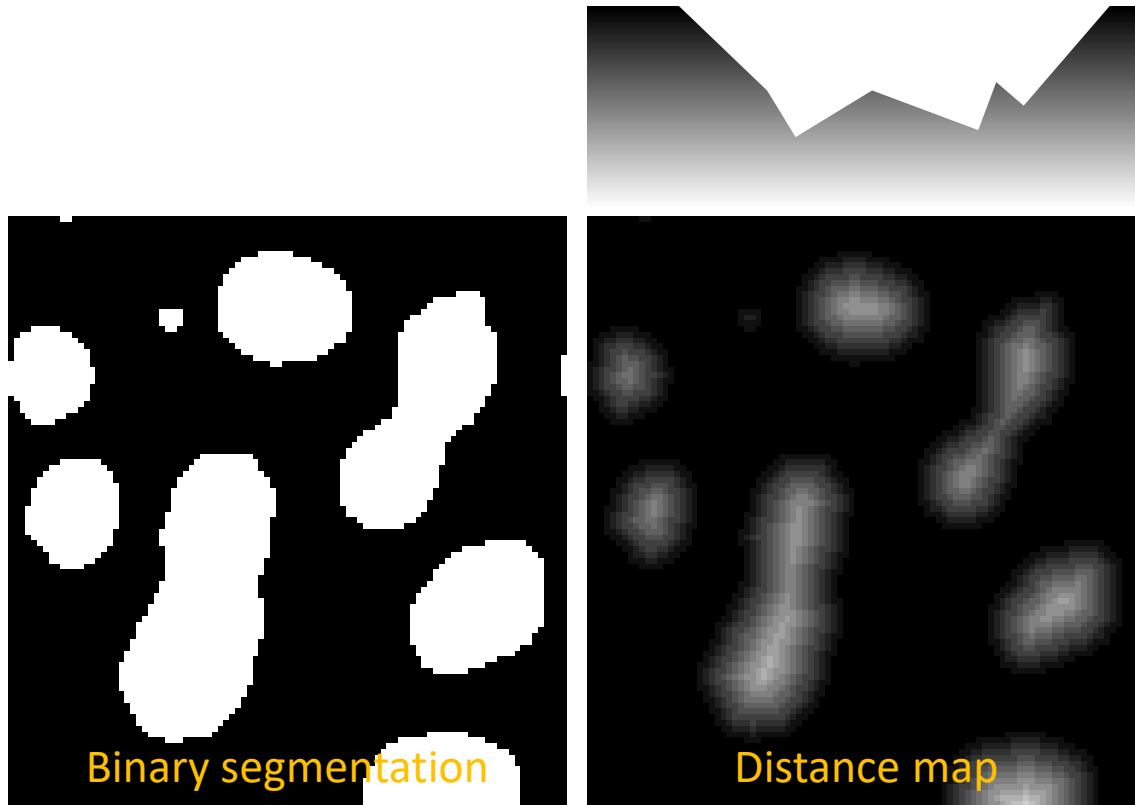
# Common image segmentation workflows

- Combination of Gaussian blur, Otsu's Threshold and Voronoi-labeling



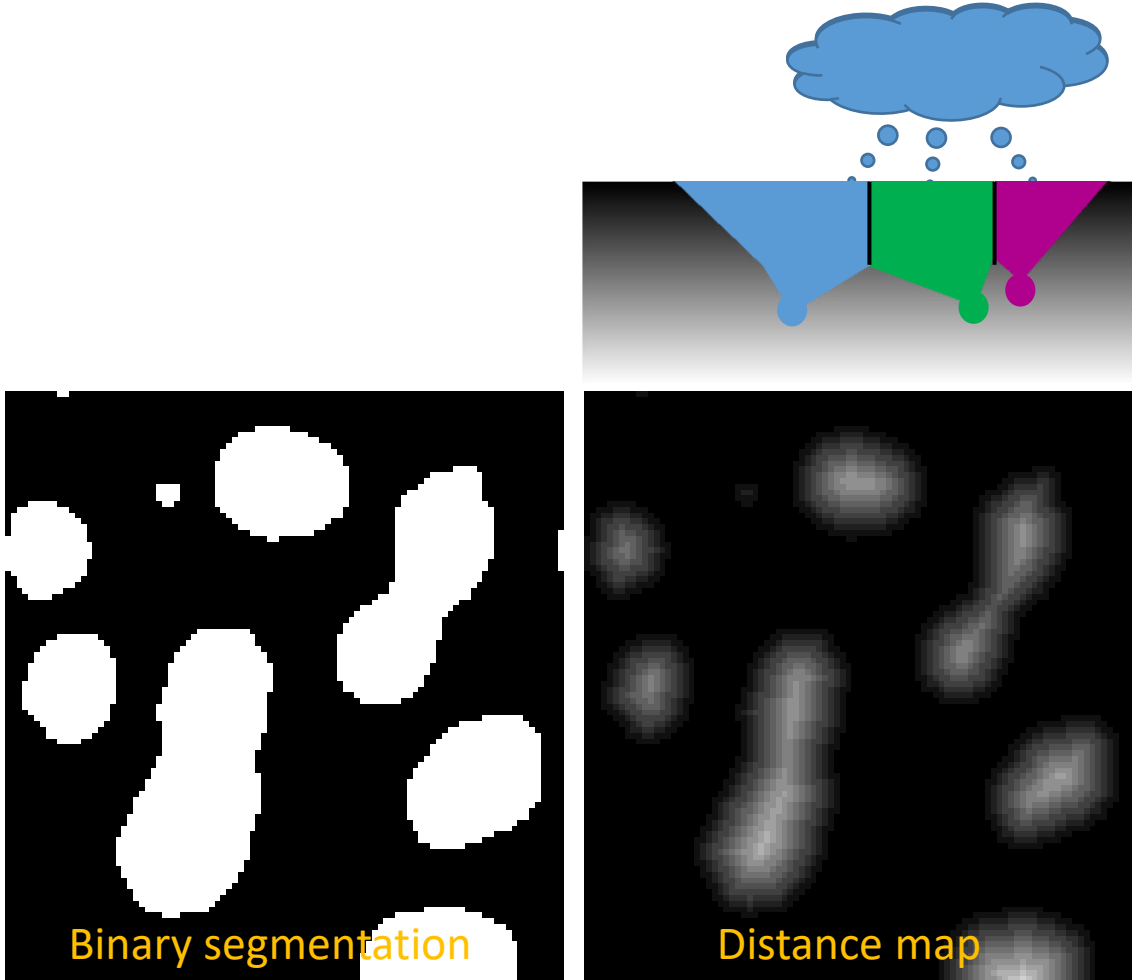
# Watershed

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



# Watershed

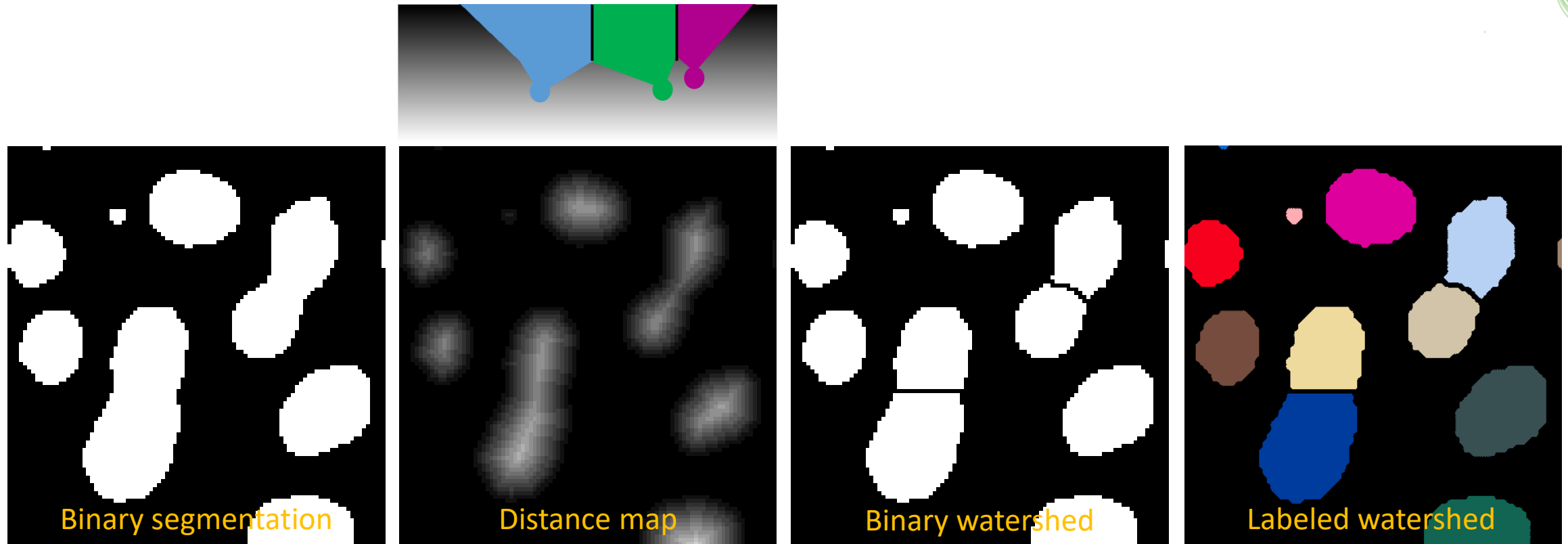
- The watershed algorithm for binary images allows cutting one object into two where it's reasonable.





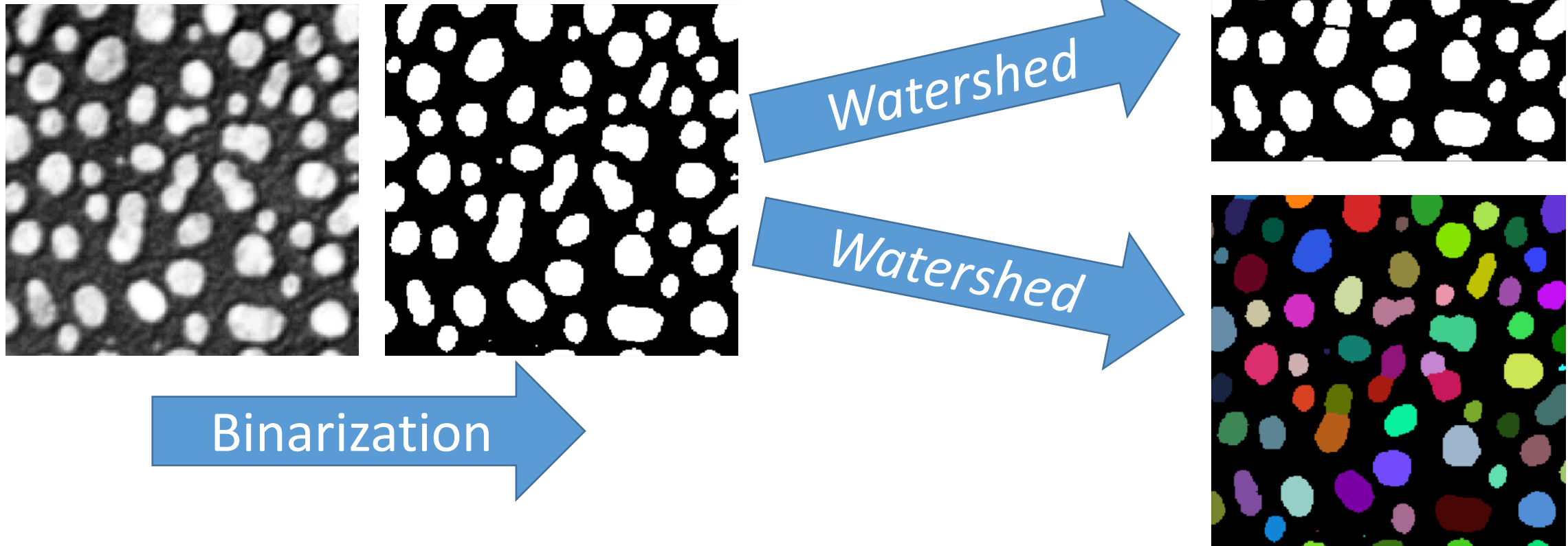
# Watershed

- The watershed algorithm for binary images allows cutting one object into two where it's reasonable.
- The distance-maps are typically made from binary images. It does not take the original image into account!



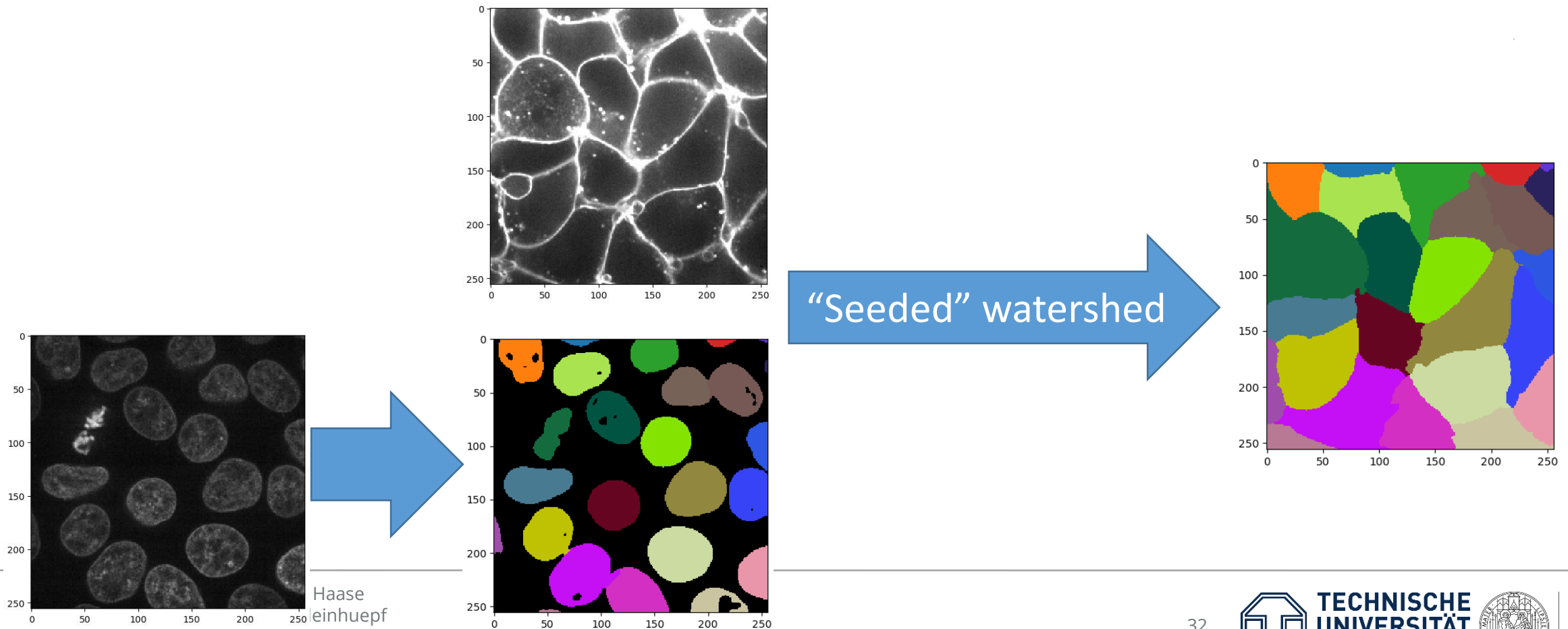
# Watershed use-cases

- Split dense objects



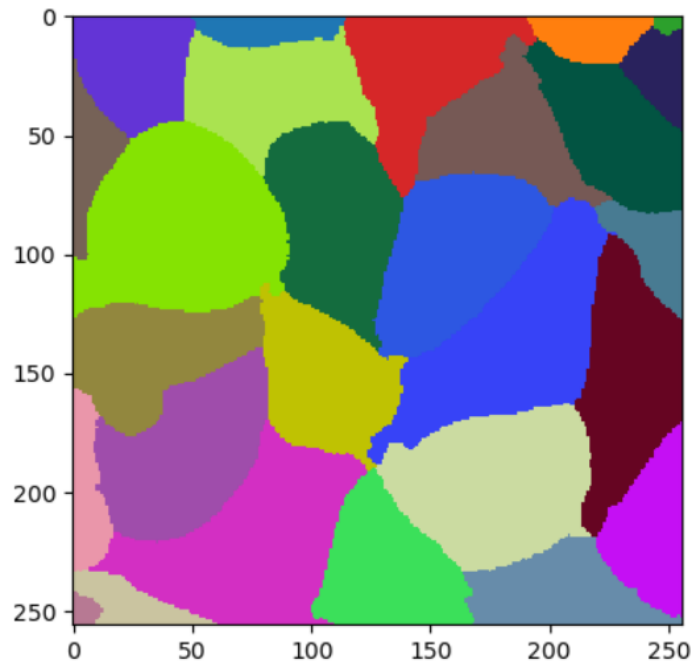
# Watershed use-cases

- Seeded watershed: Flood regions from pre-defined seeds
- Example: Flood cells from nuclei positions

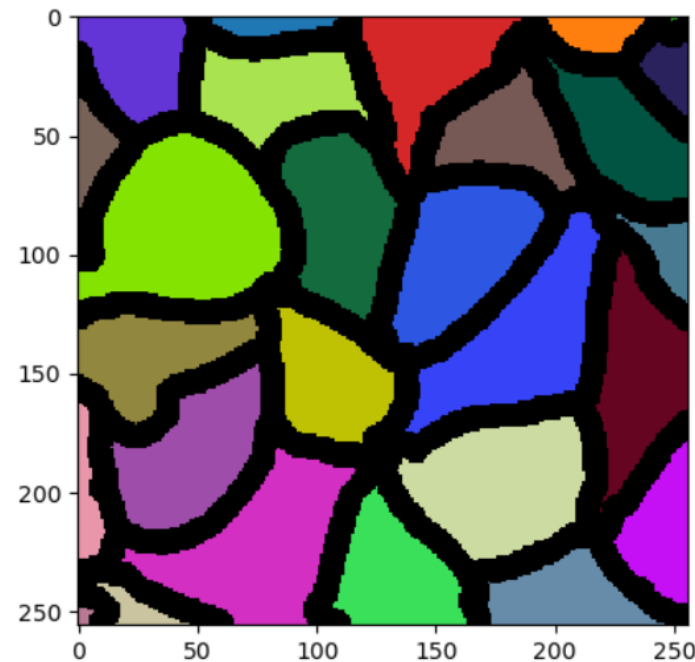


# Label post-processing / morphological operations

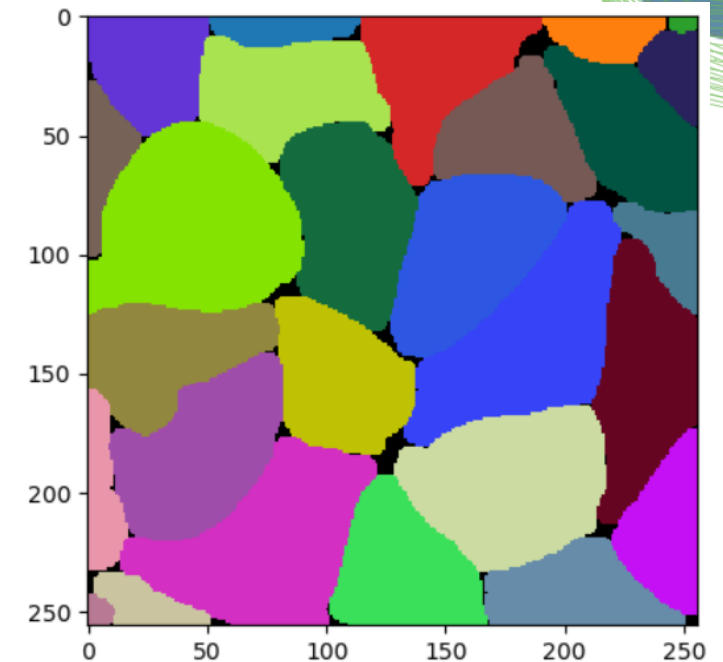
- ... similar to morphological operations on binary images



Original



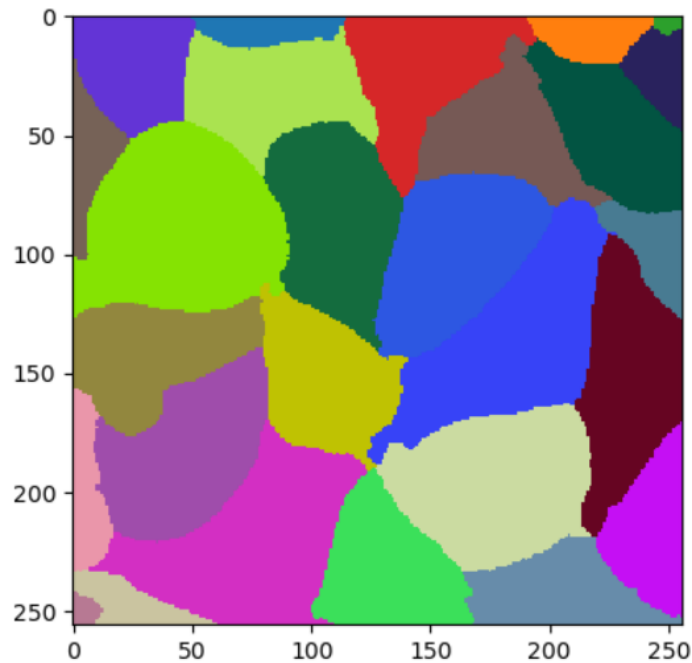
Eroding labels



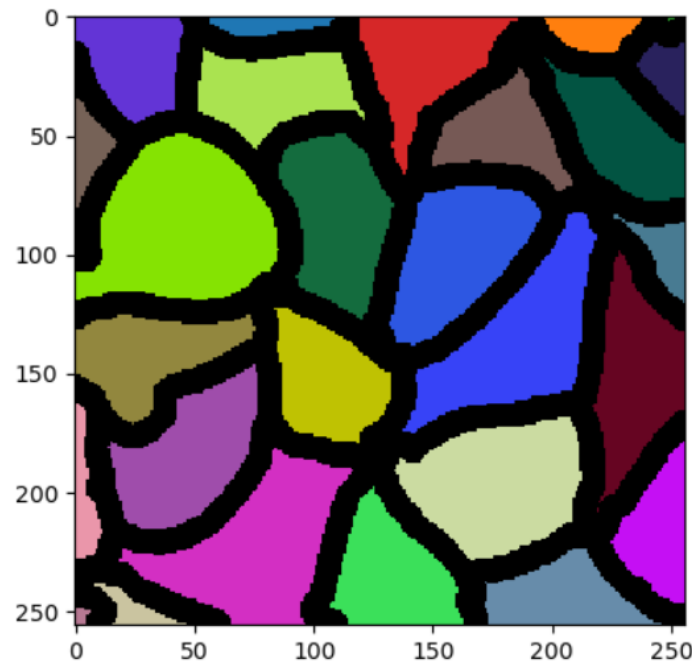
Dilating Labels

# Label post-processing / morphological operations

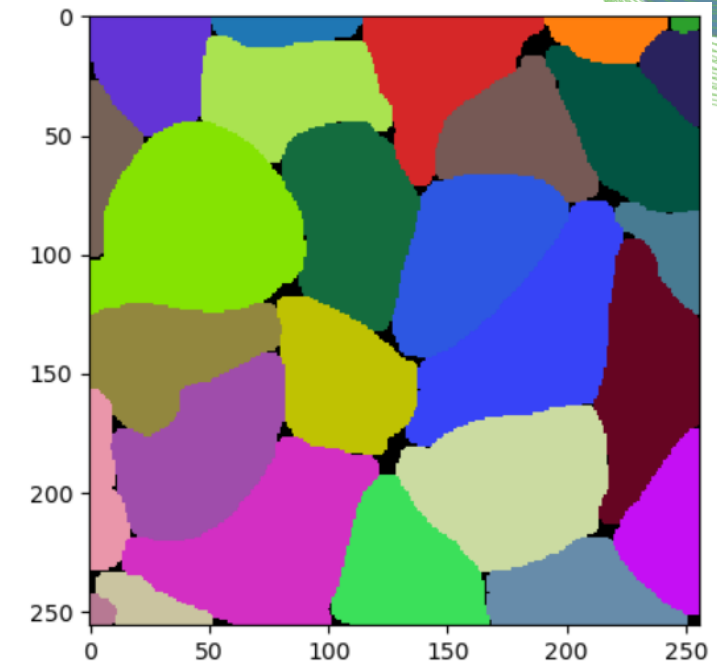
- ... similar to morphological operations on binary images



Original



Eroding labels



Dilating Labels

This combination is called ...?

Opening

Closing

Epilepsy warning

# Label post-processing / morphological operations

- ... similar to morphological operations on binary images

radius

`opening_labels(..., radius=0)`



Original

radius

`smooth_labels(..., radius=0)`



Opening labels

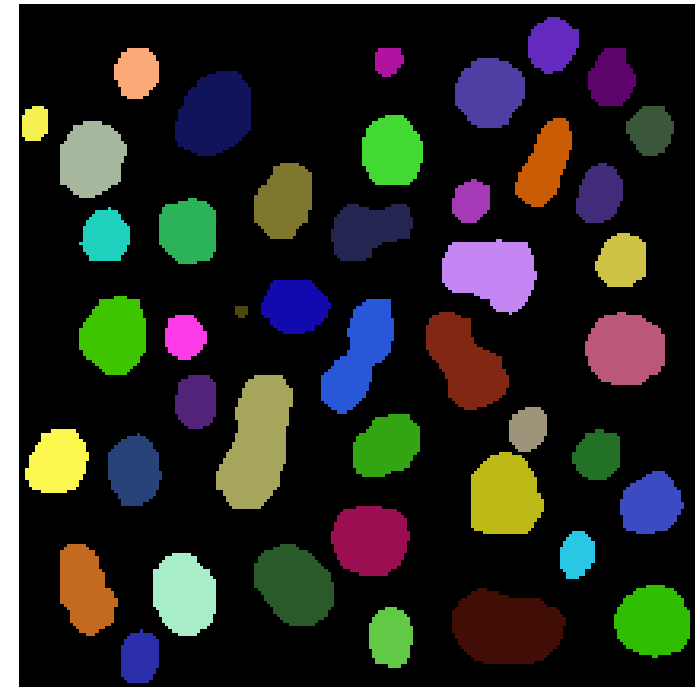
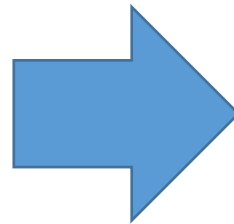
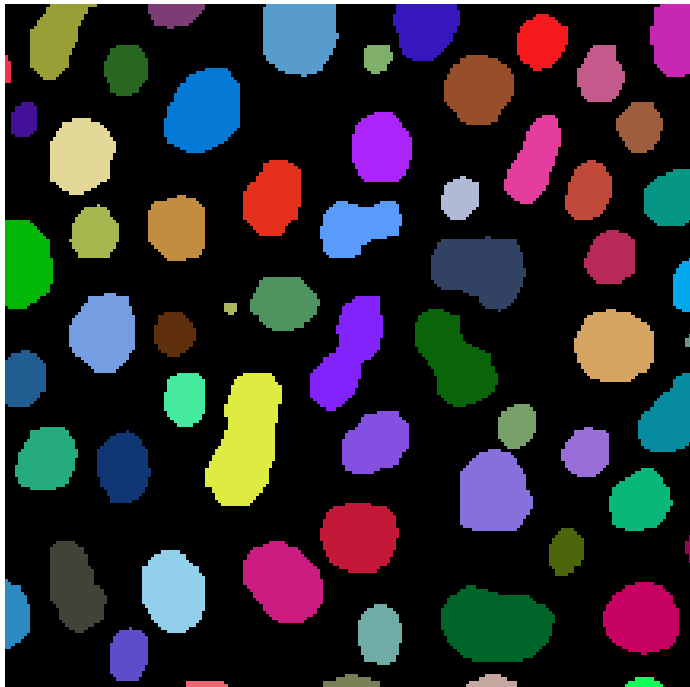


Smoothing Labels



# Label post-processing / selections

- Remove objects at the image border
- Their measurements (shape, size) would be misleading anyway

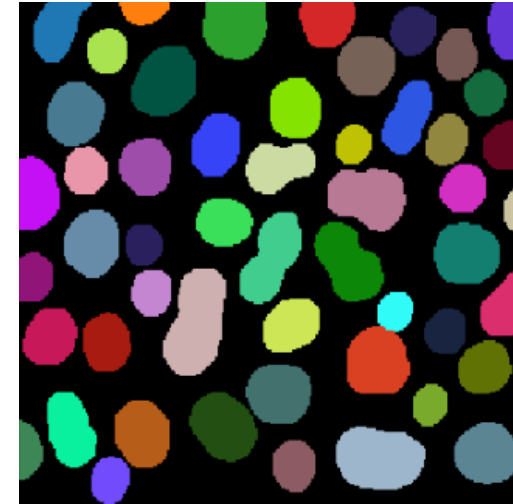


# Label post-processing / selections

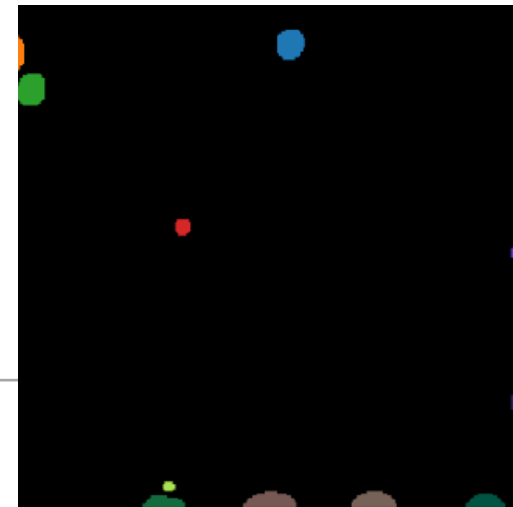
- Excluding small / large objects
- Common correction-step in case segmentations contain noise-related small particles



Exclude small objects

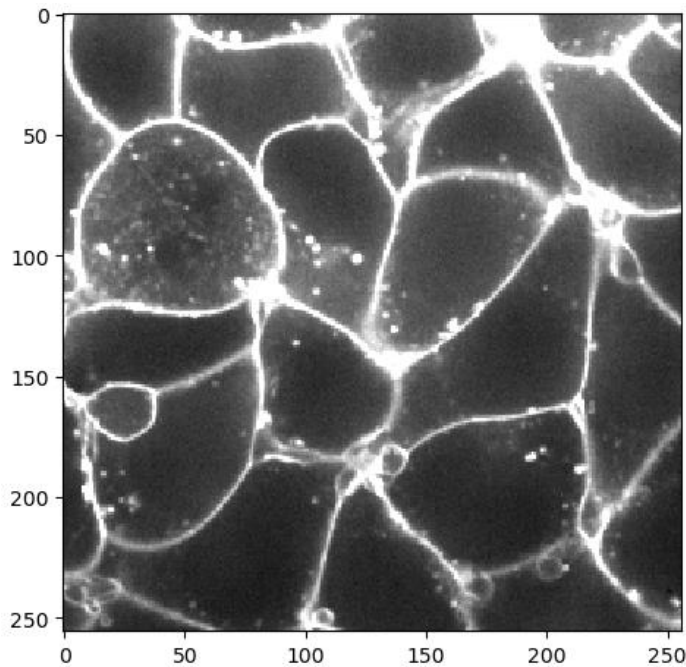


Exclude large objects



# Quiz

- What's a reasonable approach to process such an image?



Thresholding



Watershed



Machine-  
Learning



# Napari

## Robert Haase

Funded by



Bundesministerium  
für Bildung  
und Forschung

SACHSEN



Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Chan  
Zuckerberg  
Initiative 

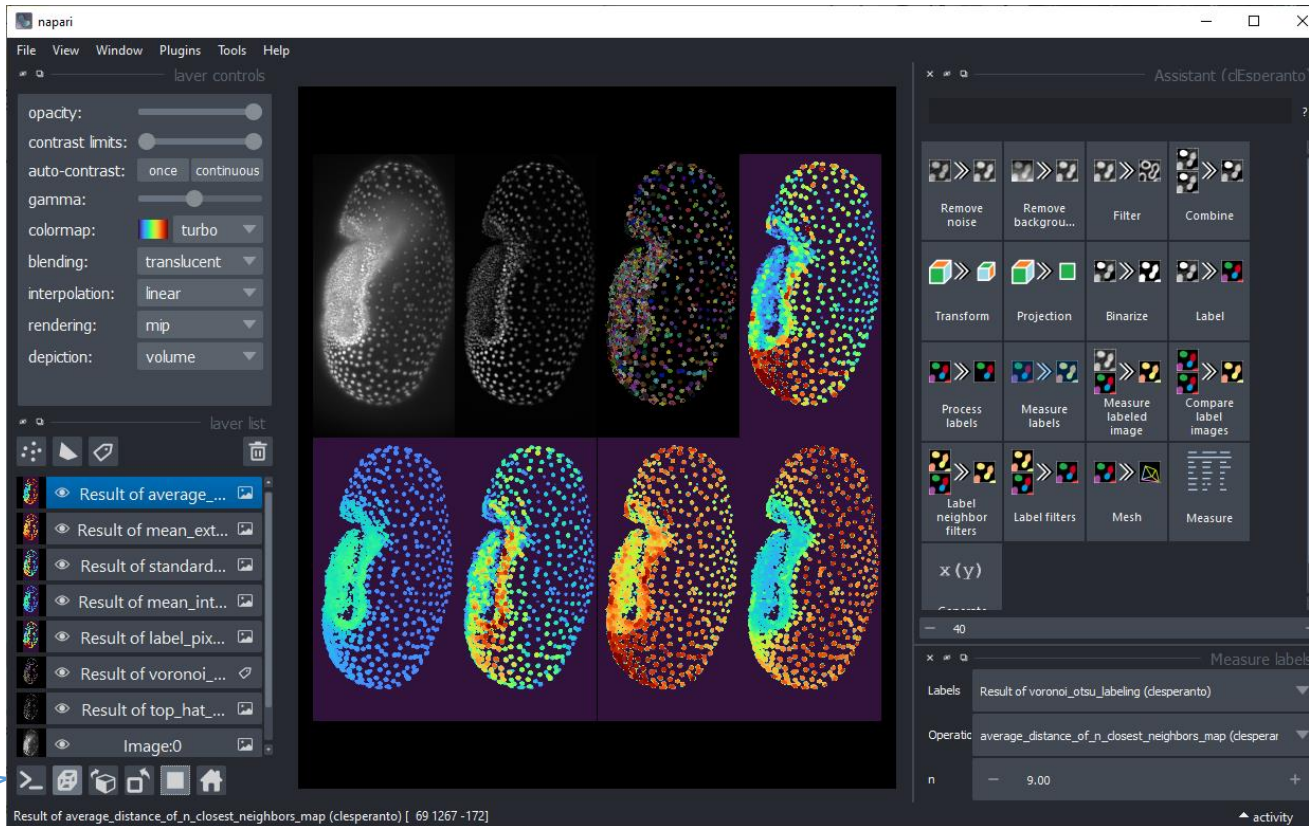
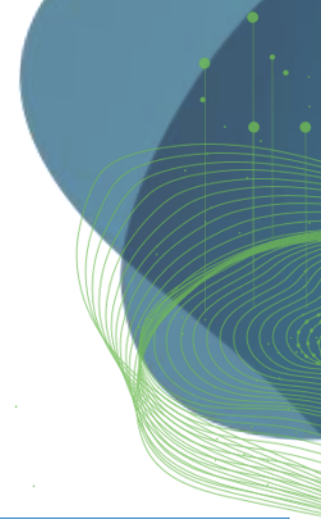
# Napari

- A viewer for n-dimensional image data written in Python





# Napari – Graphical User Interface



View configuration / tools

Layers

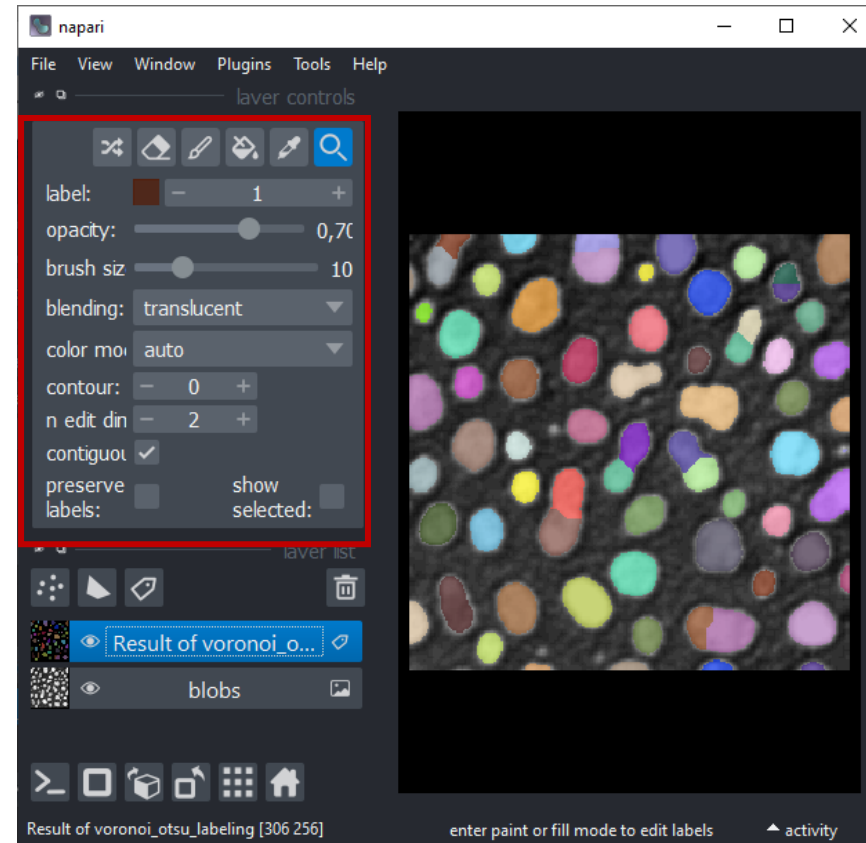
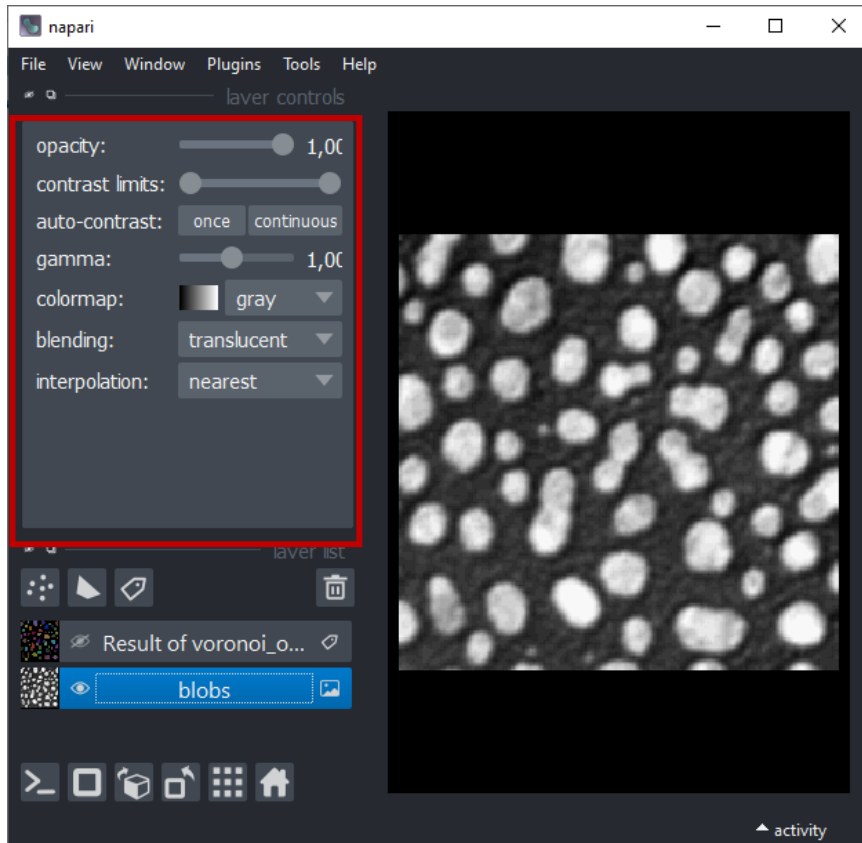
Viewer controls

Dock widgets (custom plugins)

Function widgets (custom plugins)

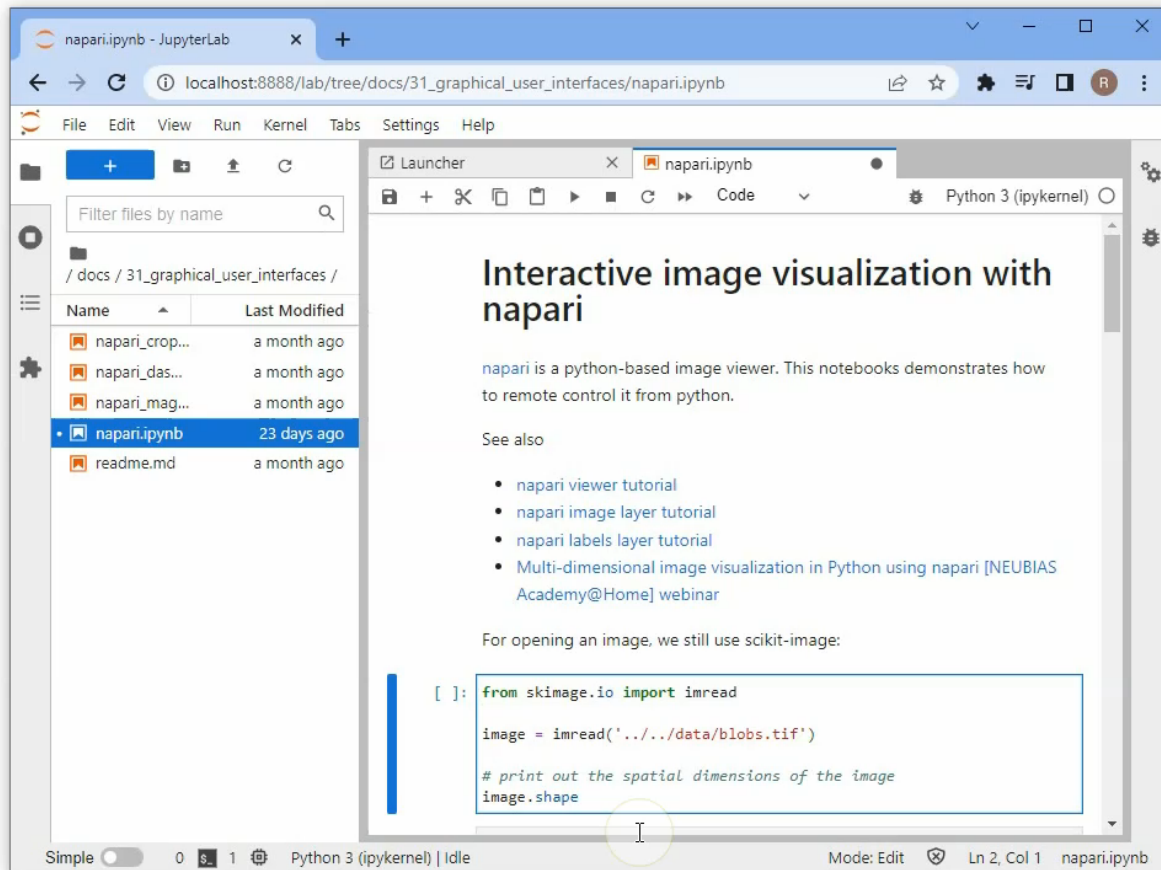
# Napari – Graphical User Interface

- Context / data type dependent tools



# Napari – Python Scripting

- Mixing interactivity and reproducibility



The screenshot shows a JupyterLab environment with a file browser on the left and a notebook editor on the right. The notebook content is as follows:

## Interactive image visualization with napari

napari is a python-based image viewer. This notebooks demonstrates how to remote control it from python.

See also

- [napari viewer tutorial](#)
- [napari image layer tutorial](#)
- [napari labels layer tutorial](#)
- [Multi-dimensional image visualization in Python using napari \[NEUBIAS Academy@Home\] webinar](#)

For opening an image, we still use skimage-image:

```
[ ]: from skimage.io import imread
      image = imread('../data/blobs.tif')
      # print out the spatial dimensions of the image
      image.shape
```



# Napari – Python Scripting

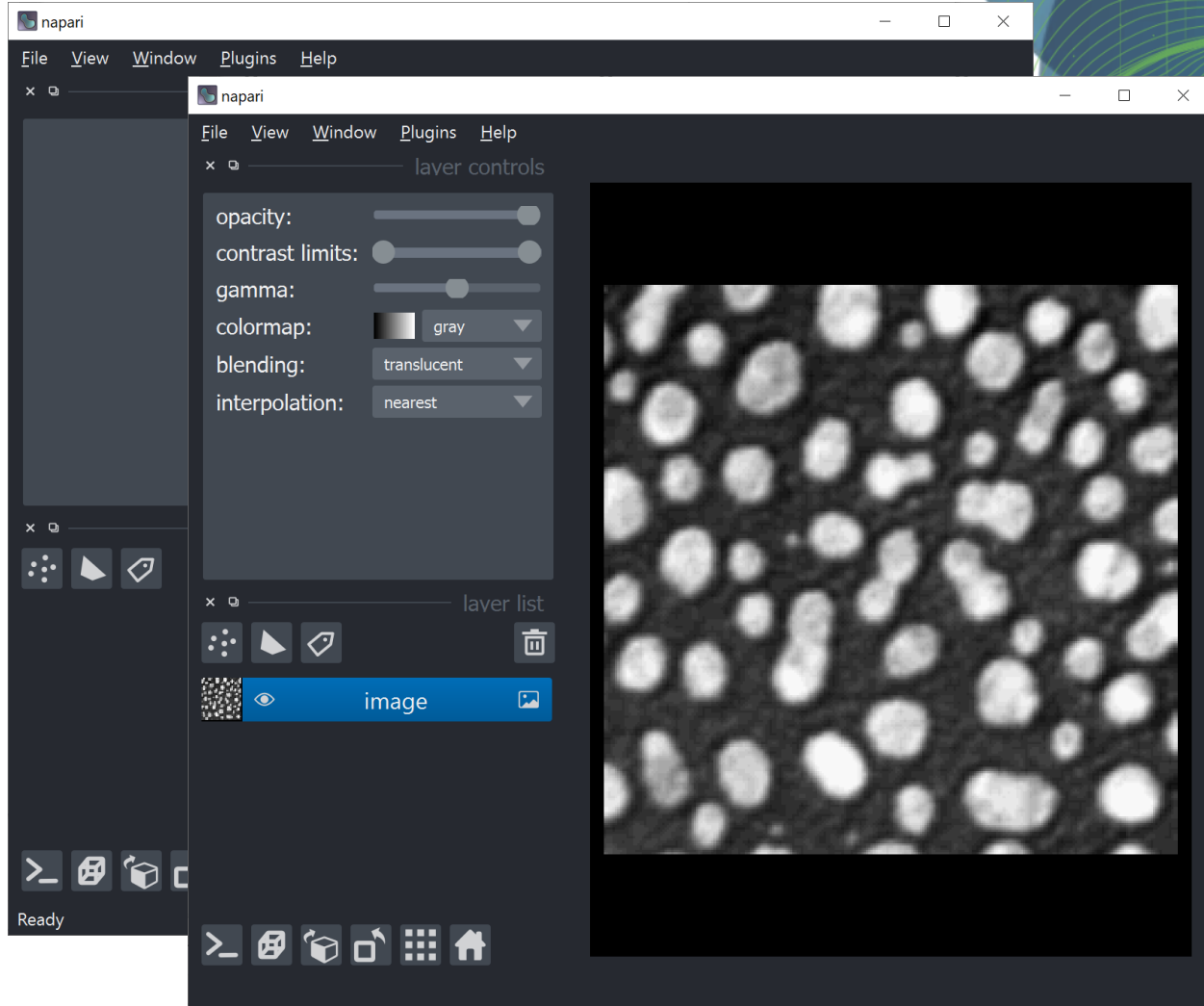
- Initialization

```
import napari
```

```
# Create an empty viewer  
viewer = napari.Viewer()
```

- Adding images

```
viewer.add_image(image)
```

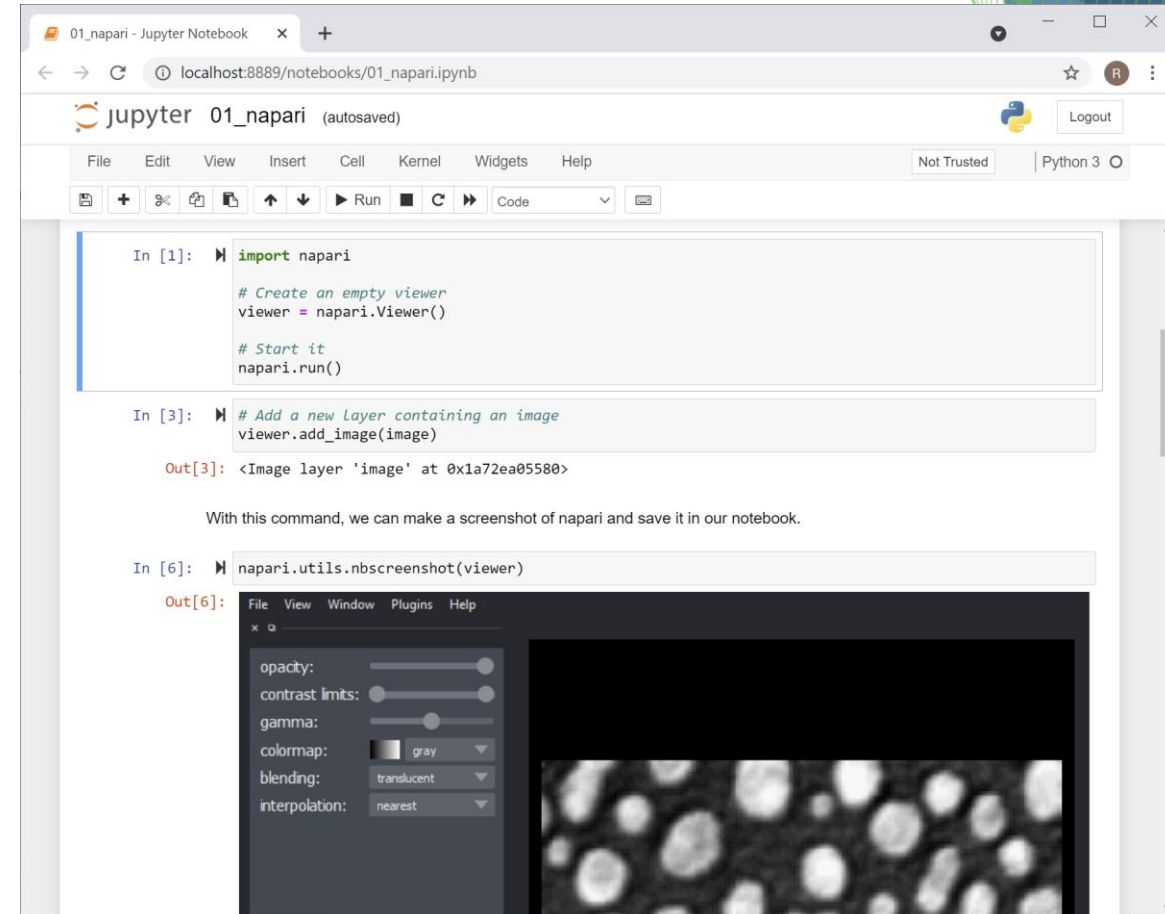


# Napari – Python Scripting

- Make screenshots from napari and put them in your jupyter notebook

```
napari.utils.nbscreenshot(viewer)
```

Place your viewer here



# Napari – Python Scripting

- Removing layers

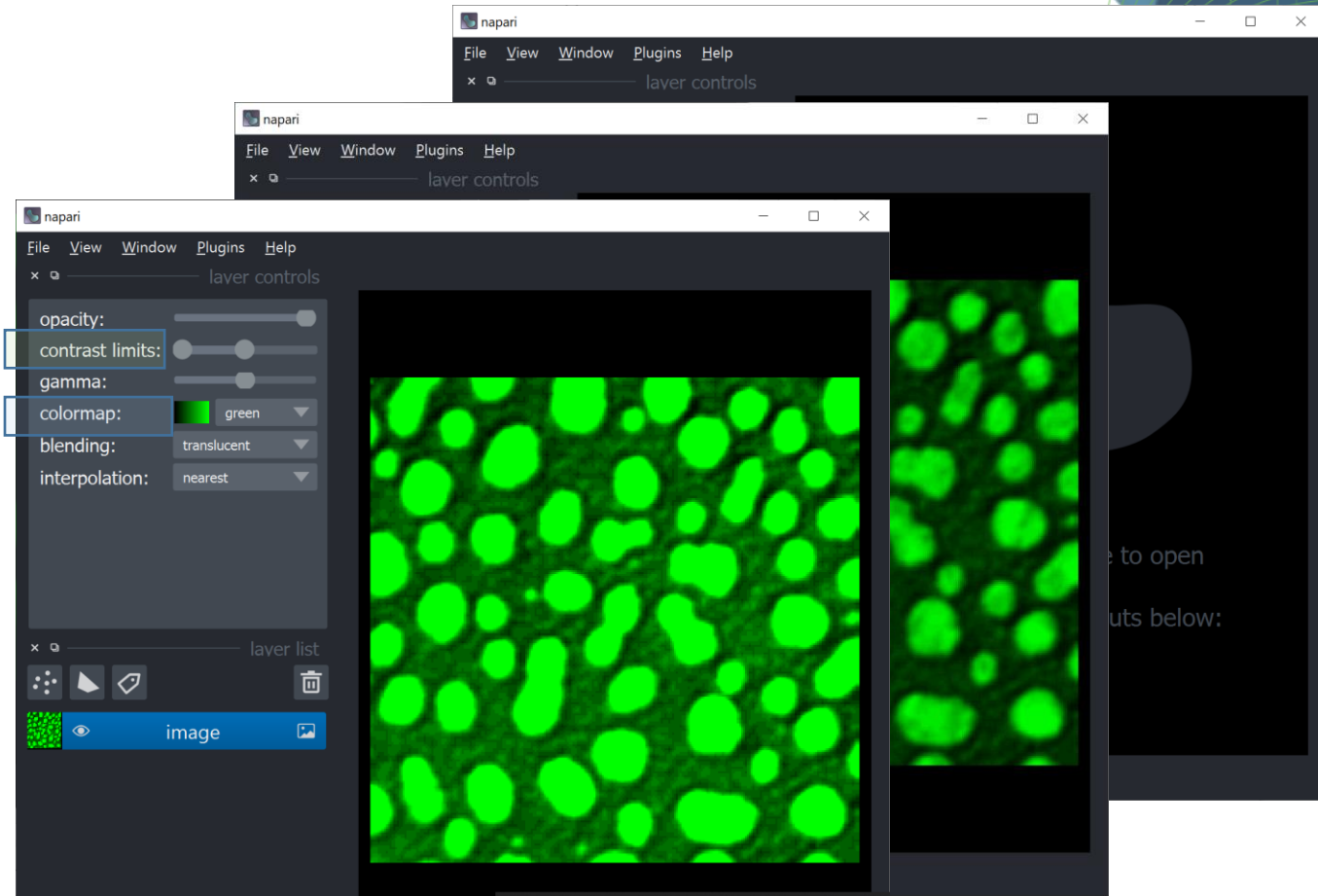
```
for l in viewer.layers:  
    viewer.layers.remove(l)
```

- Modify visualization while adding layers

```
viewer.add_image(image,  
                 colormap='green')
```

- Modify layers after adding

```
layer = viewer.add_image(image)  
layer.colormap = 'green'  
layer.contrast_limits = (0, 128)
```

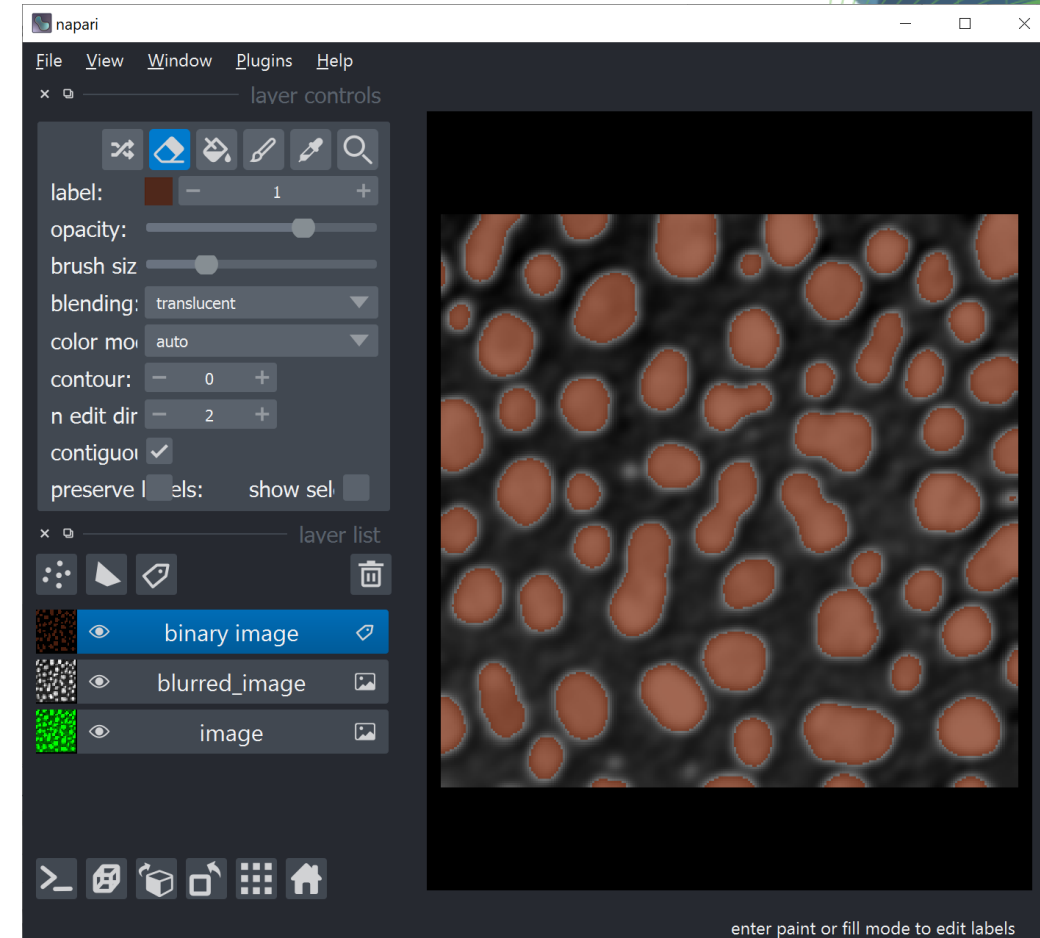


# Napari – Python Scripting

- Binary images and `label` images visualized as label layers

```
from skimage.filters import threshold_otsu
threshold = threshold_otsu(blurred_image)
binary_image = blurred_image > threshold

# Add a new labels layer containing an image
viewer.add_labels(binary_image)
```



# The Napari Assistant

- Tools > Utilities > Assistant (na)

Viewer controls

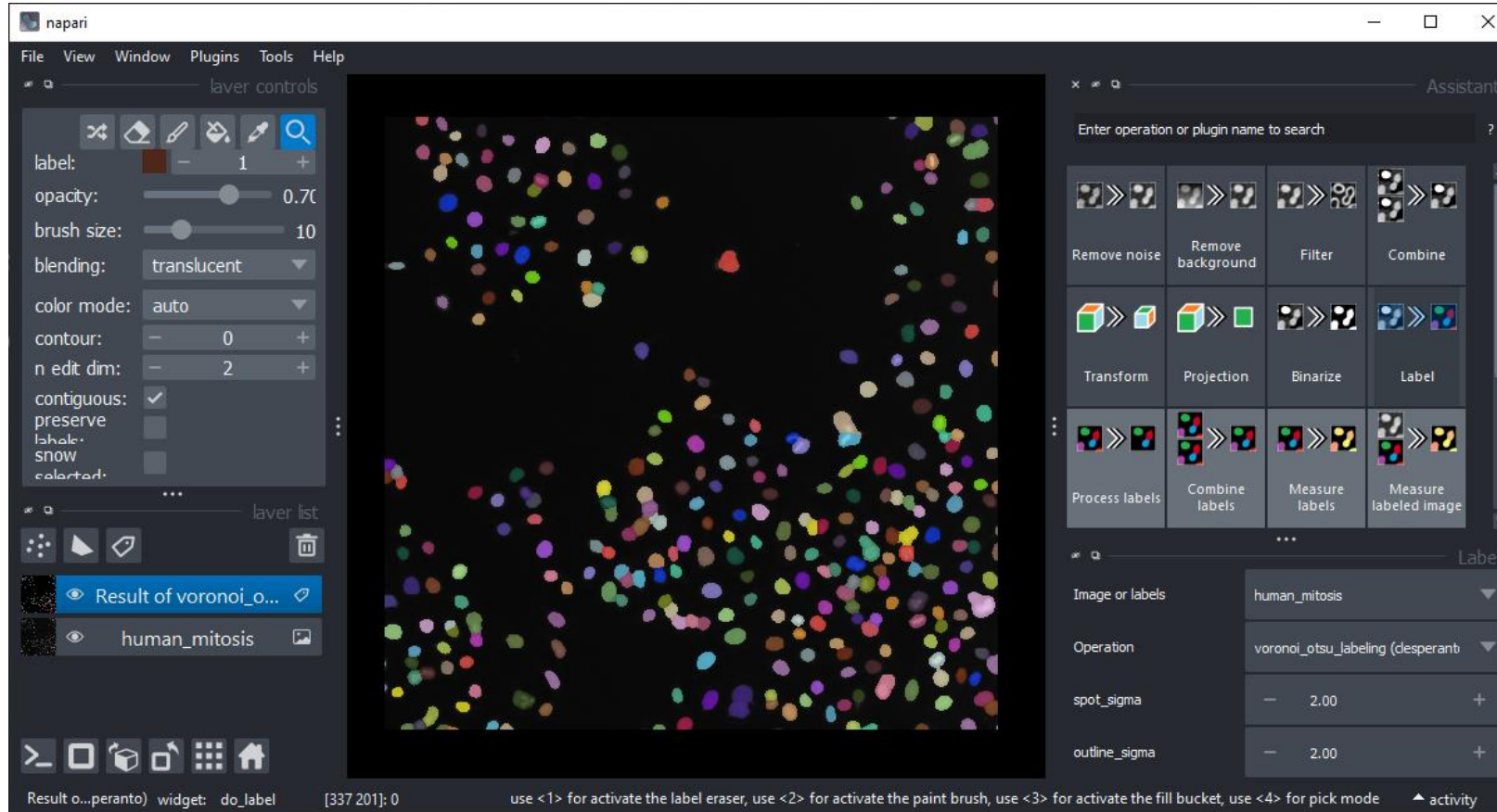
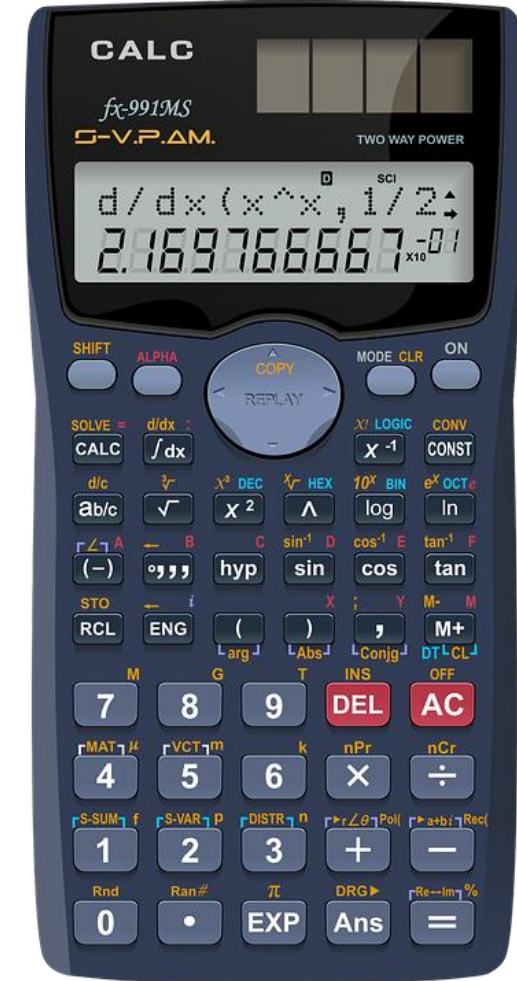


Image Processing



# The Napari Assistant

- A pocket-calculator-like interface to build image analysis workflows



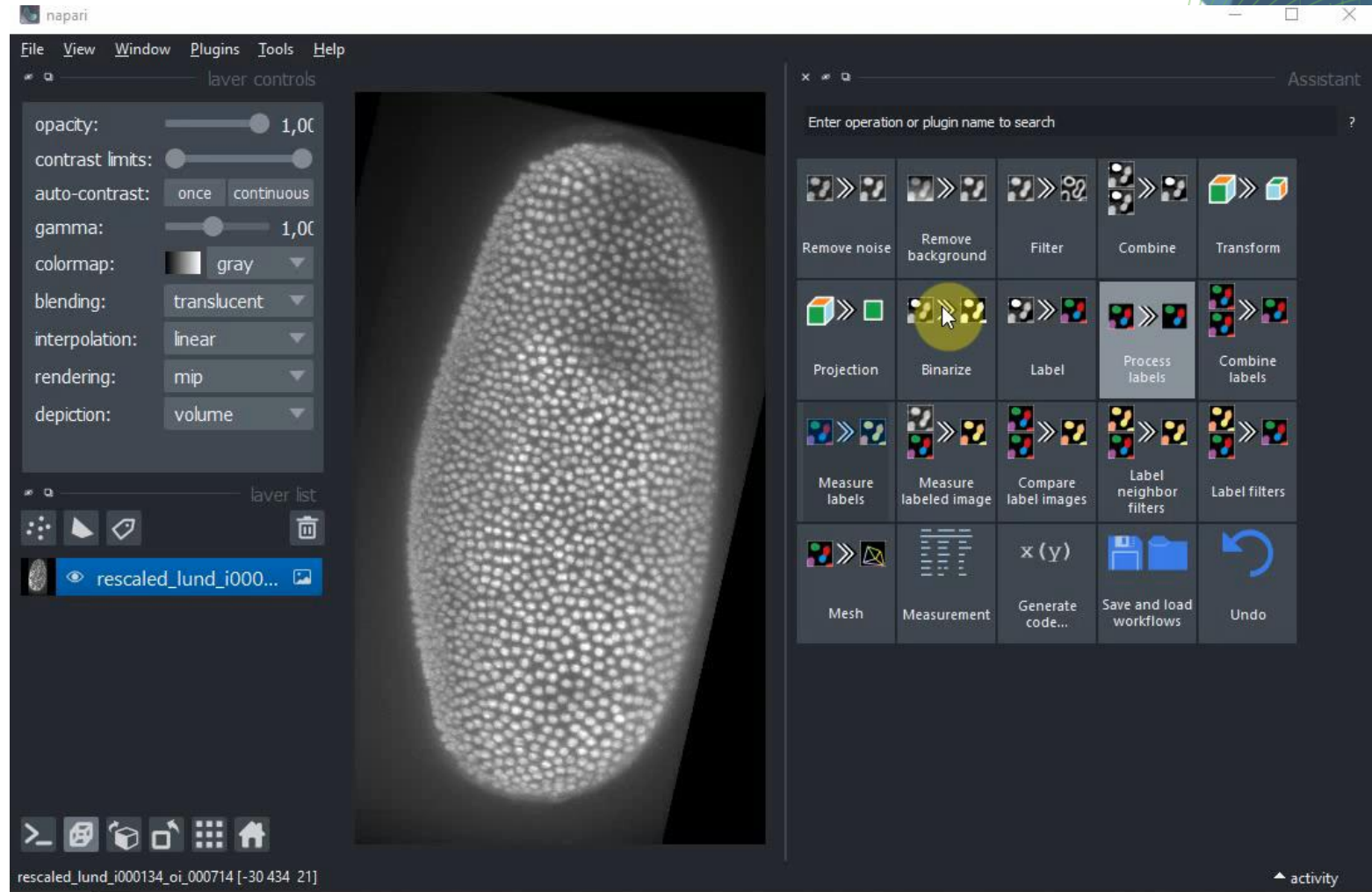
# The Napari Assistant

- Classical image processing operations + advanced tools
- Saving&loading supported
- Undo [redo]
- Hints for next steps
- ...

Big thanks to:

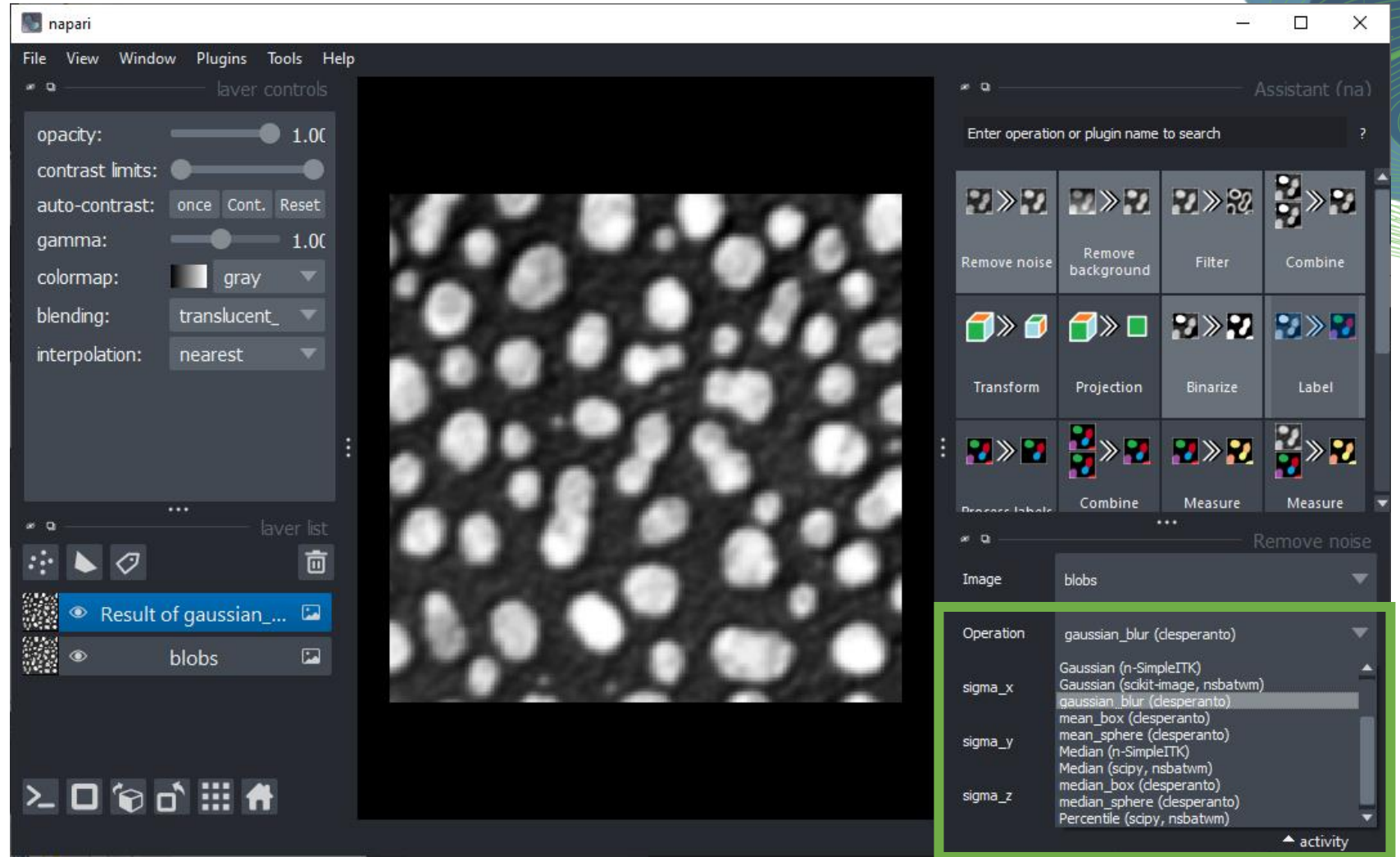


Ryan Savill  
@RyanSavill4



# Workflow building

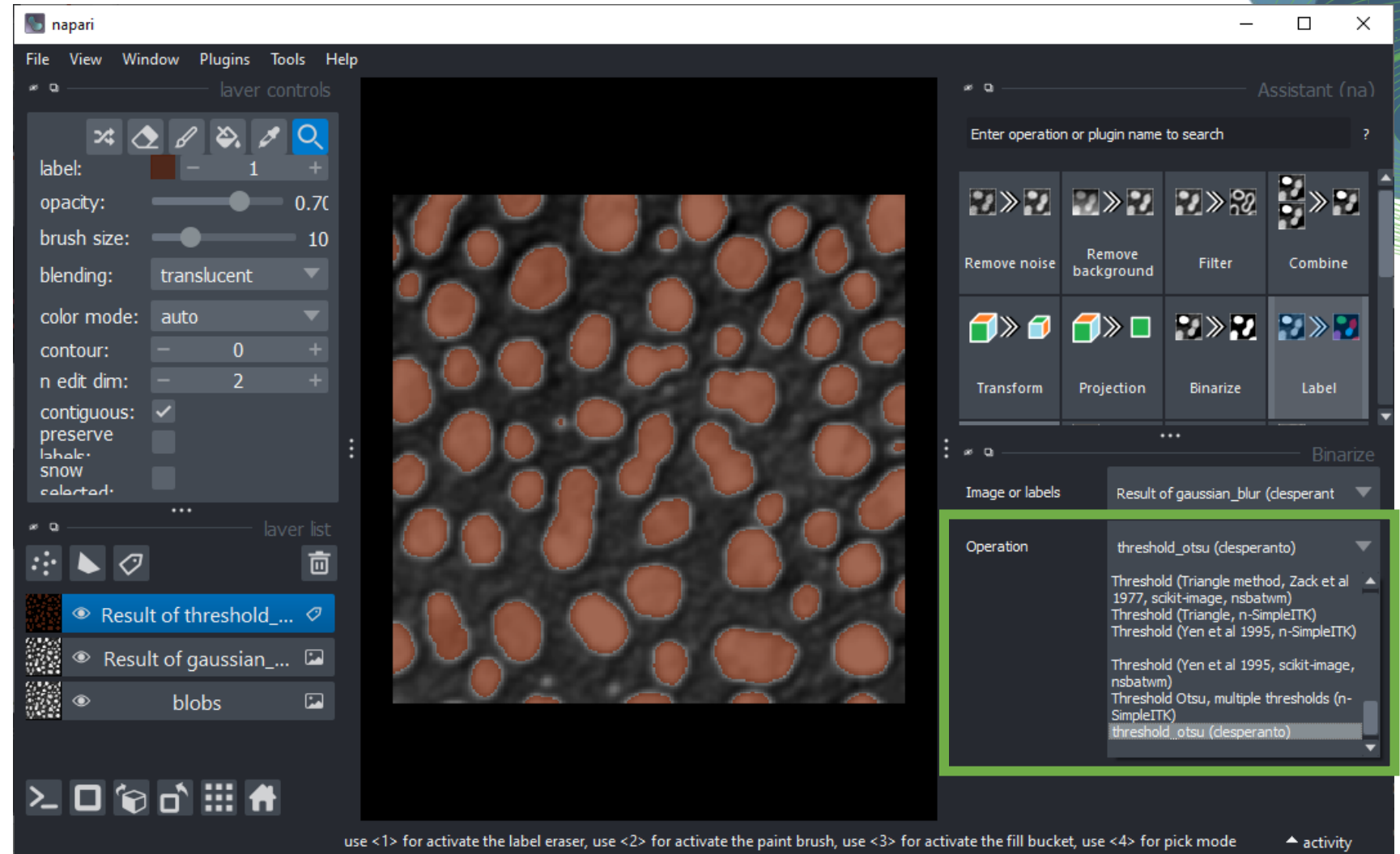
- Try different algorithms, e.g. for removing noise
- Find them in the pulldown





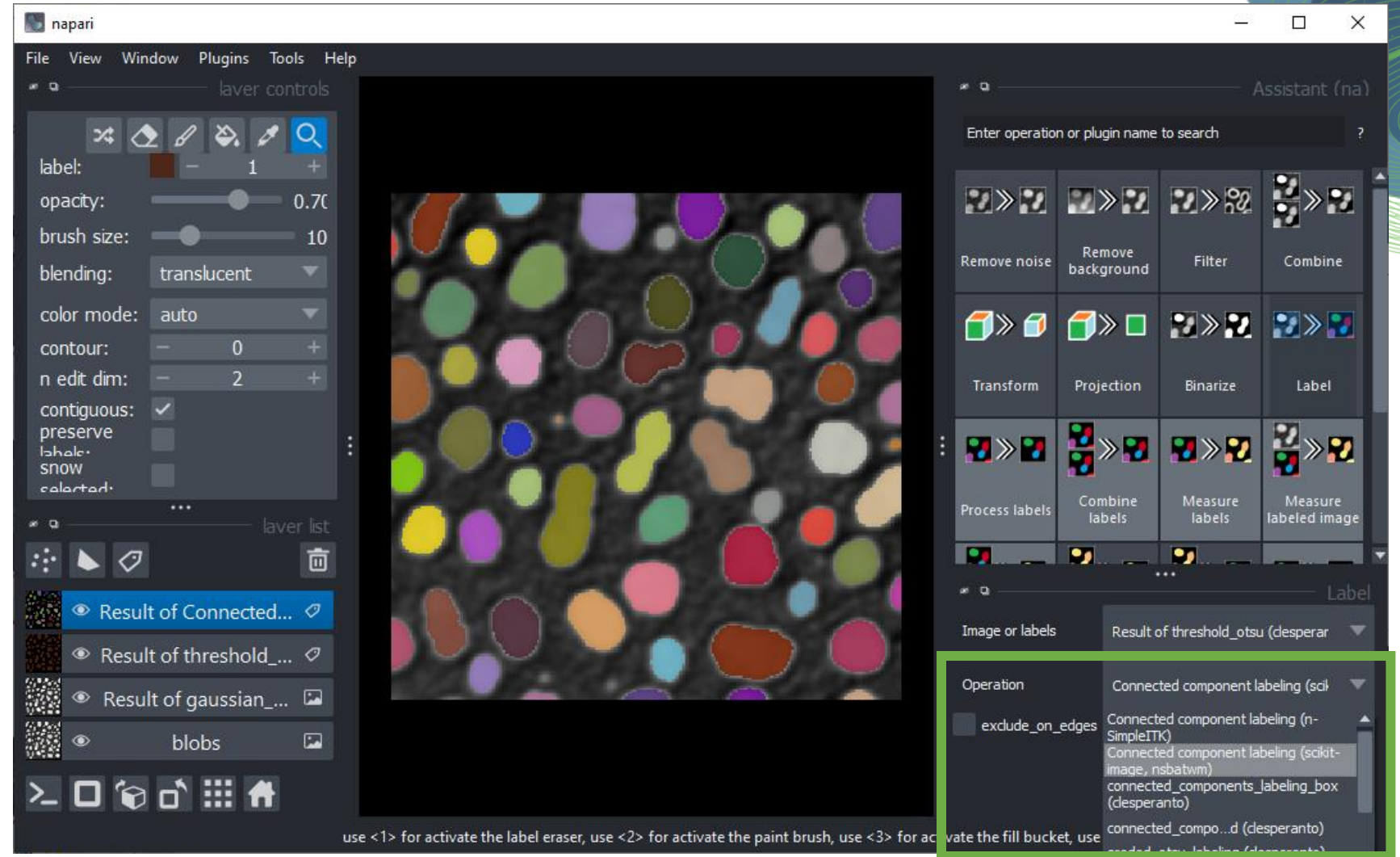
# Workflow building

- Try different binarization algorithms



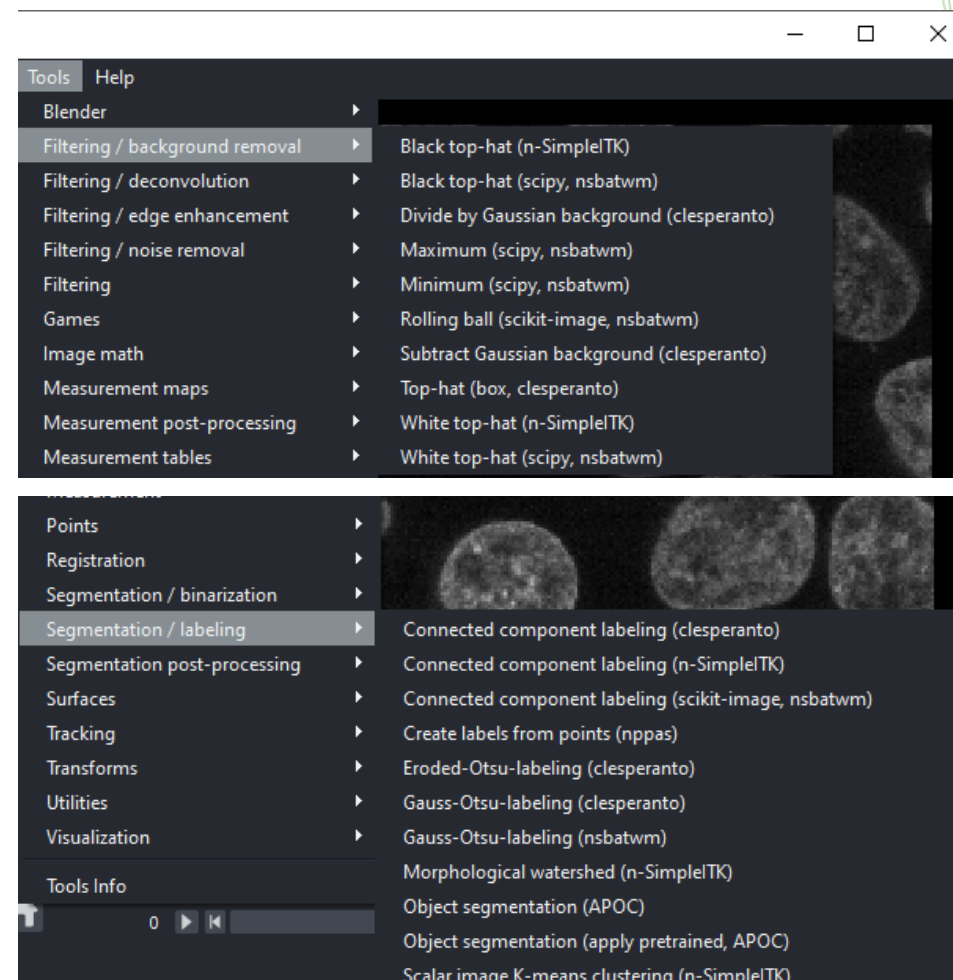
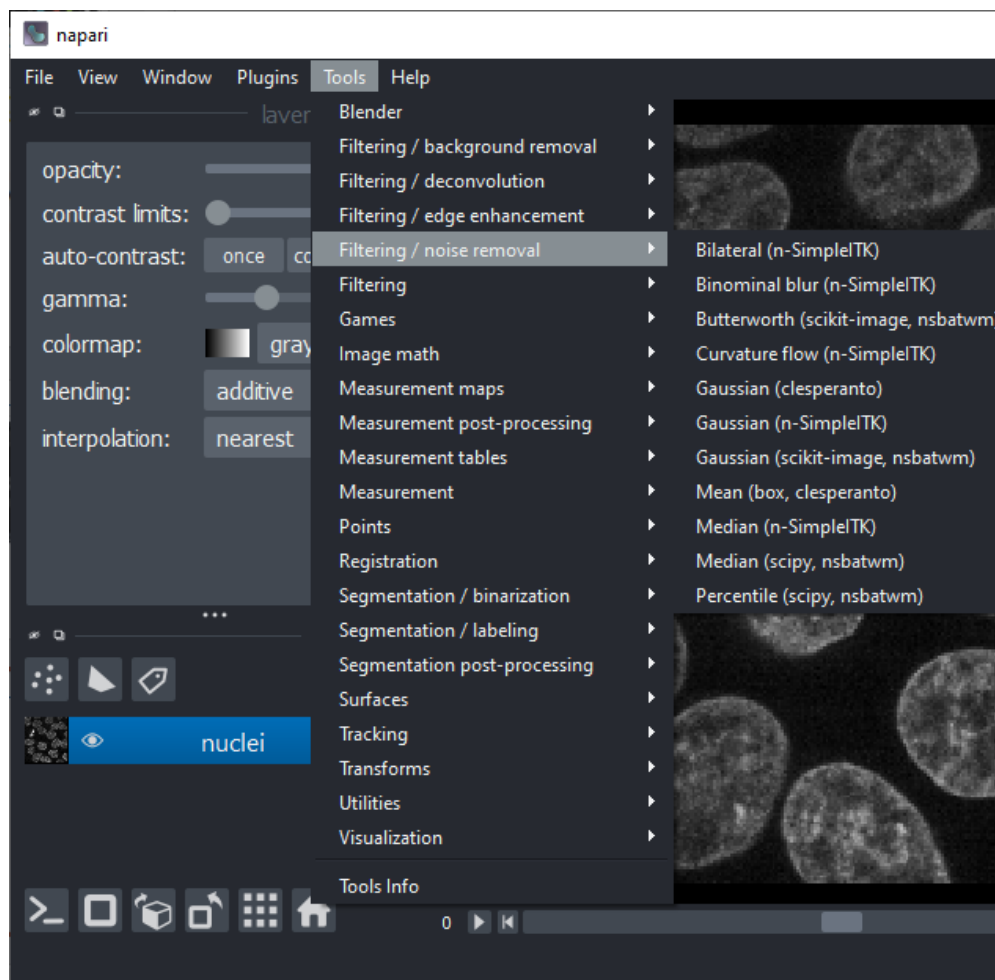
# Workflow building

- Try different labeling algorithms



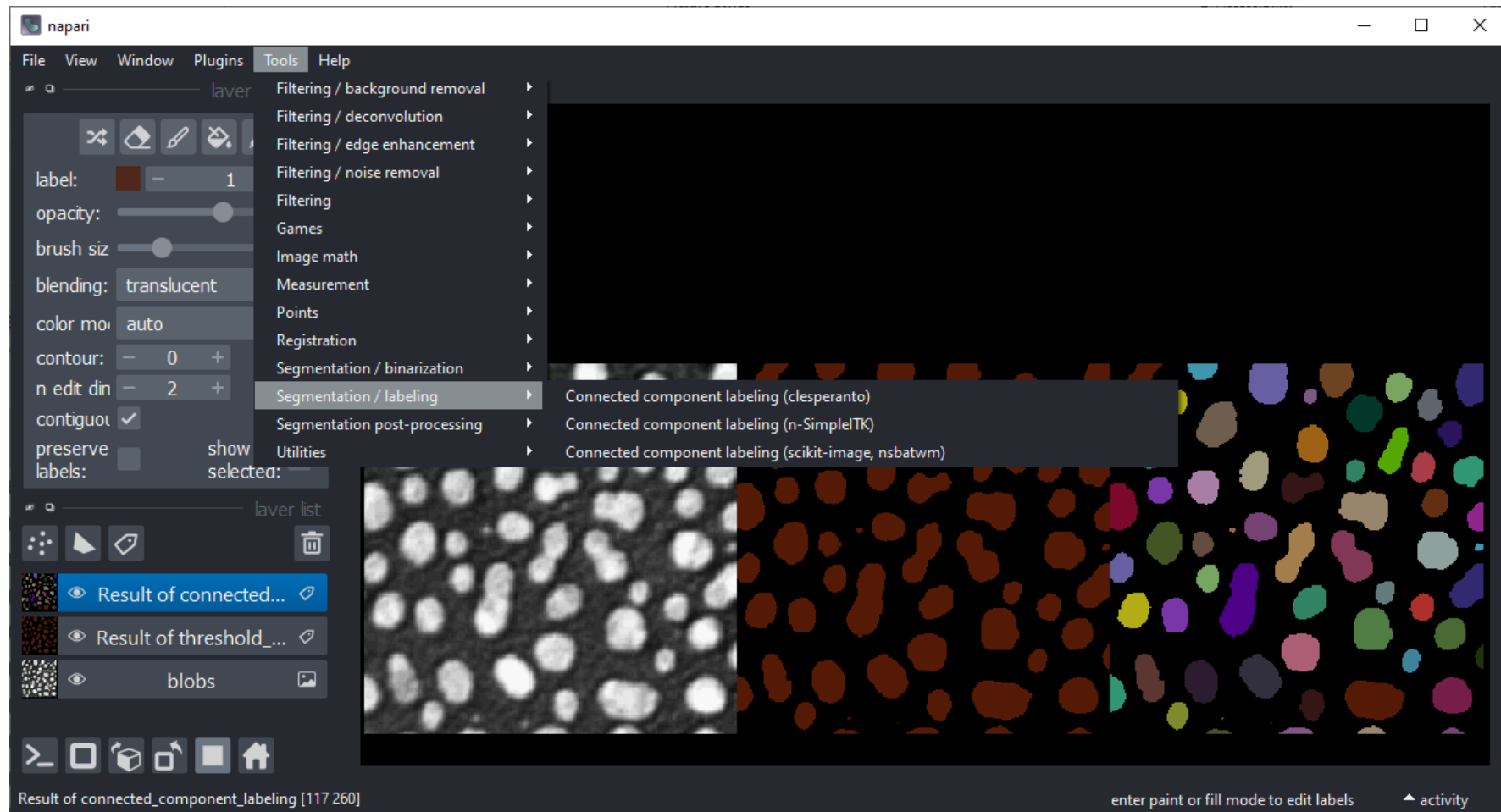
# The Tools menu

- Organized in categories



# Workflow building

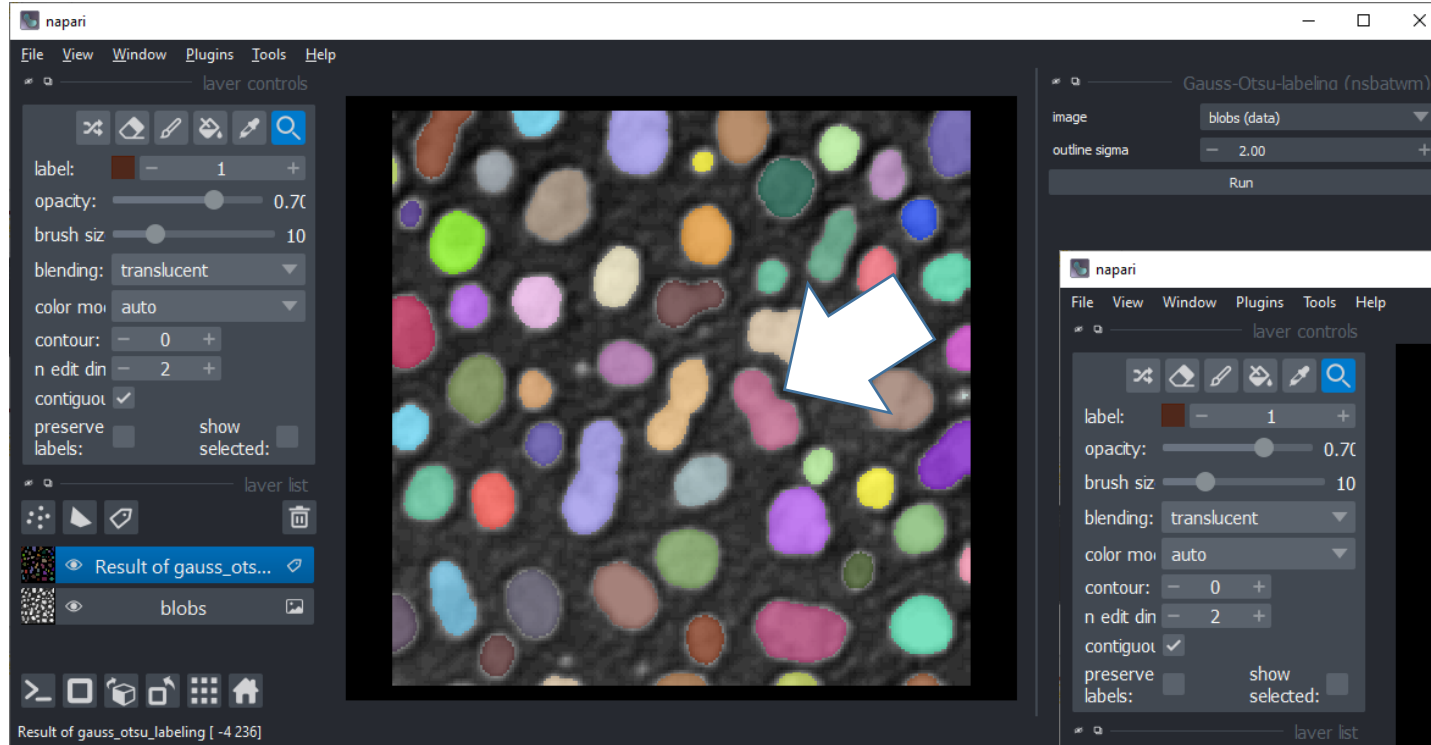
Also check out the Tools > Segmentation / labeling menu



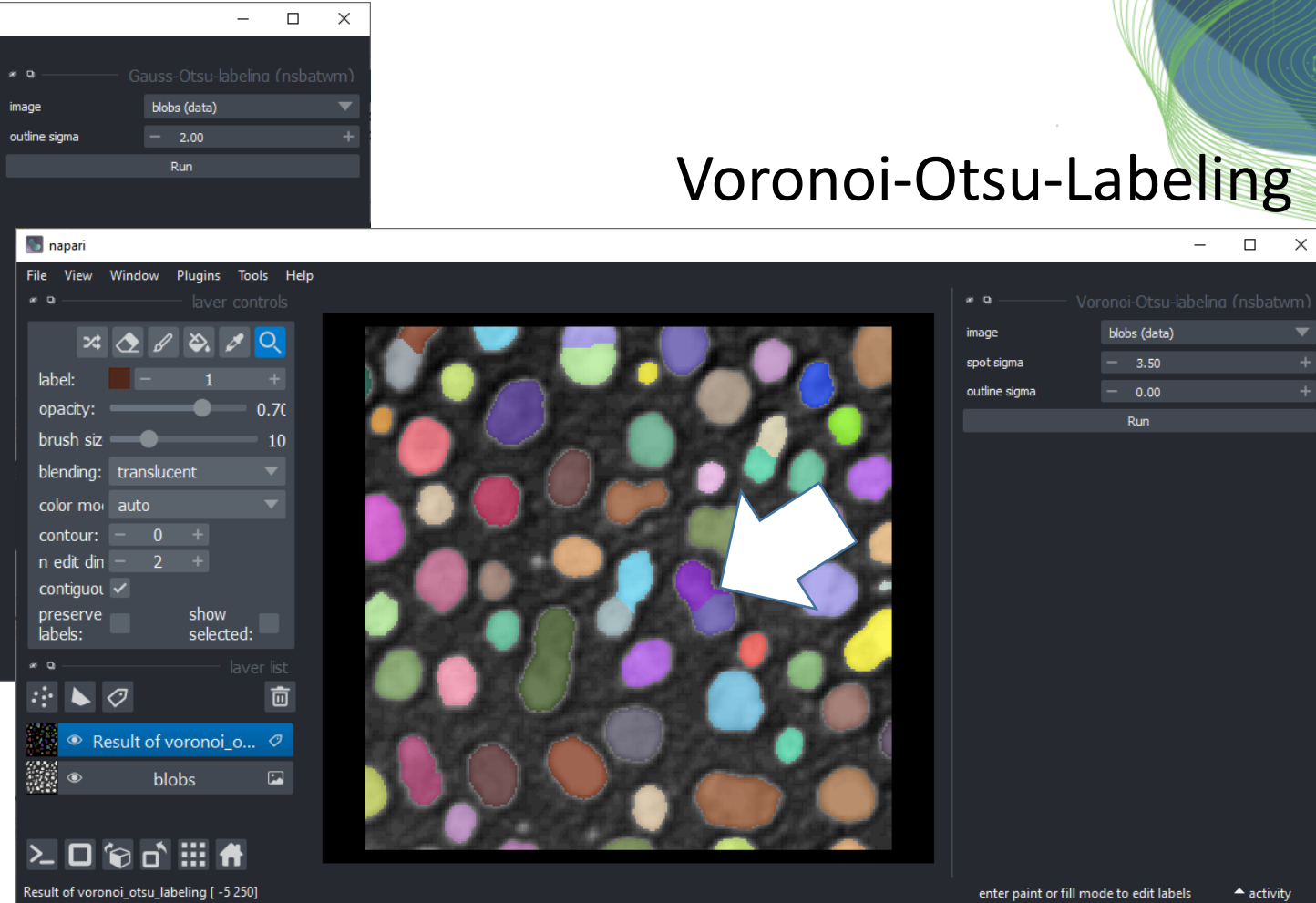


# Short-cuts: Voronoi-Otsu-Labeling

Also check out the Tools > Segmentation / labeling menu



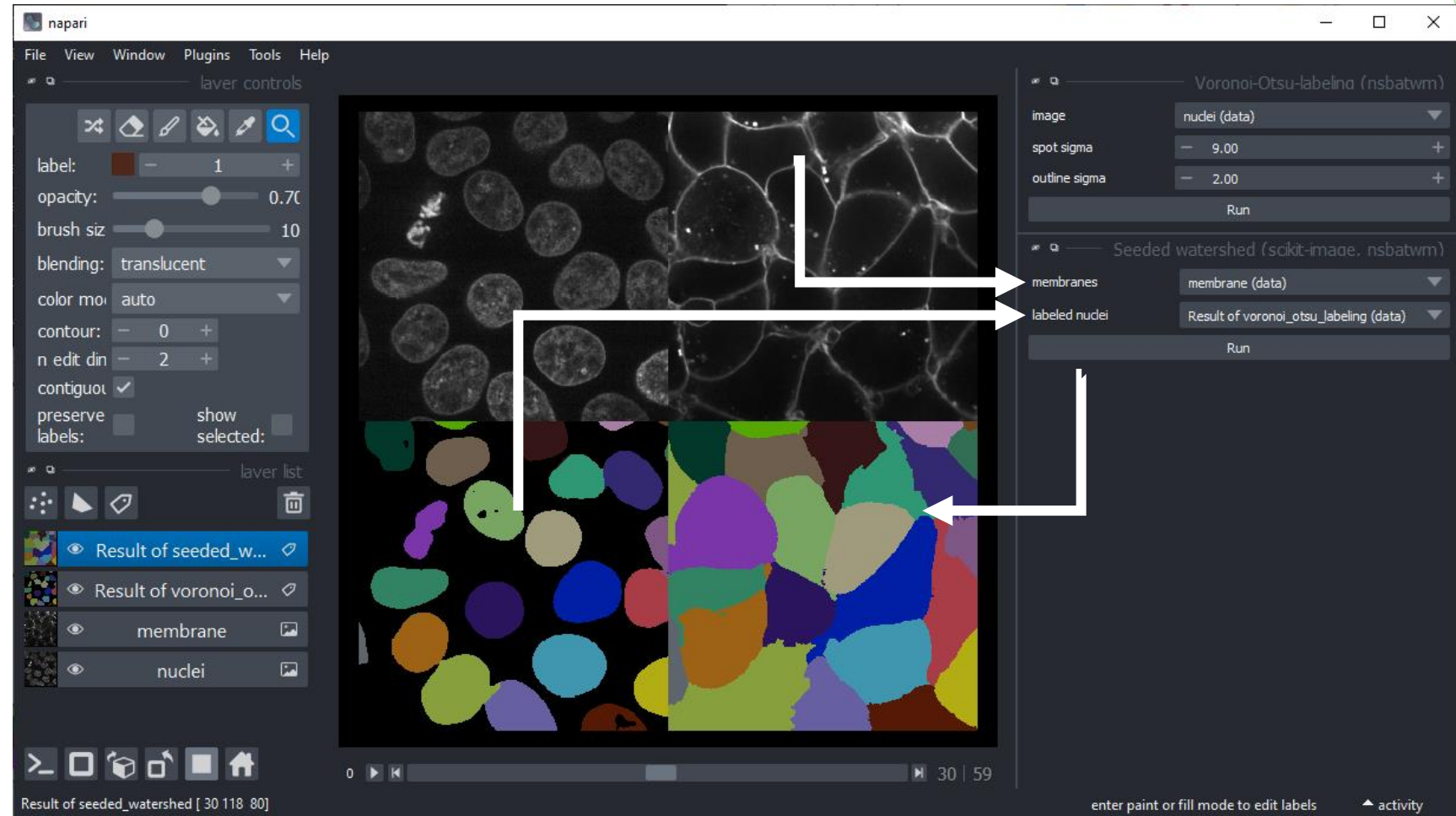
Gauss-Otsu-Labeling



Voronoi-Otsu-Labeling

# Watershed

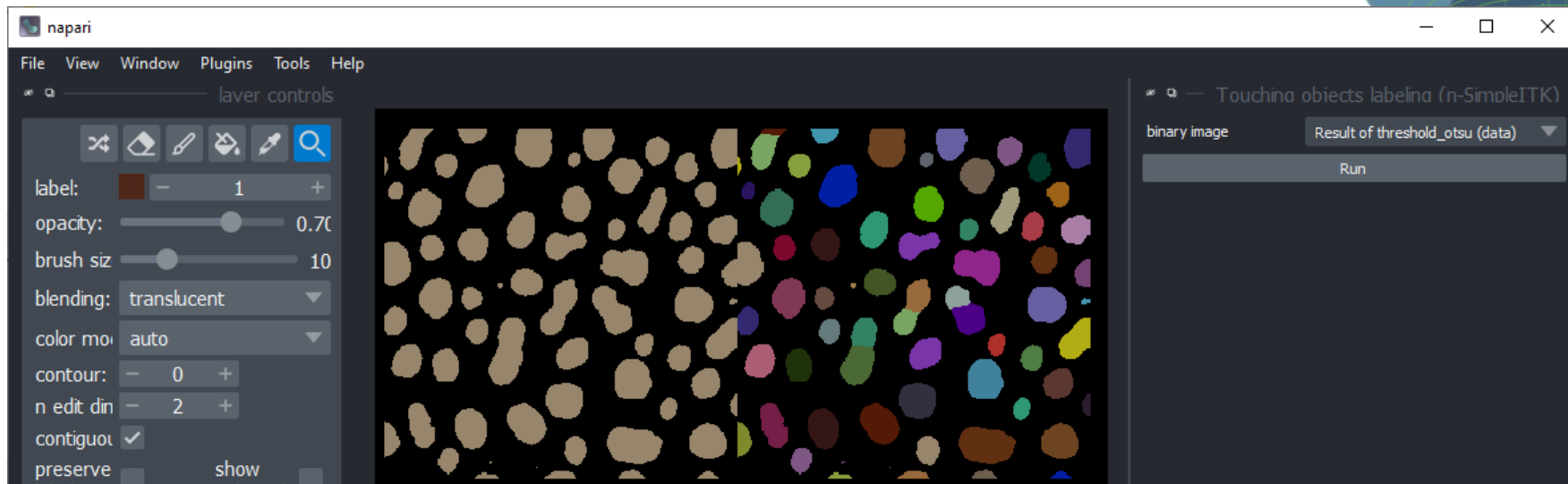
Also check out the Tools > Segmentation / labeling menu



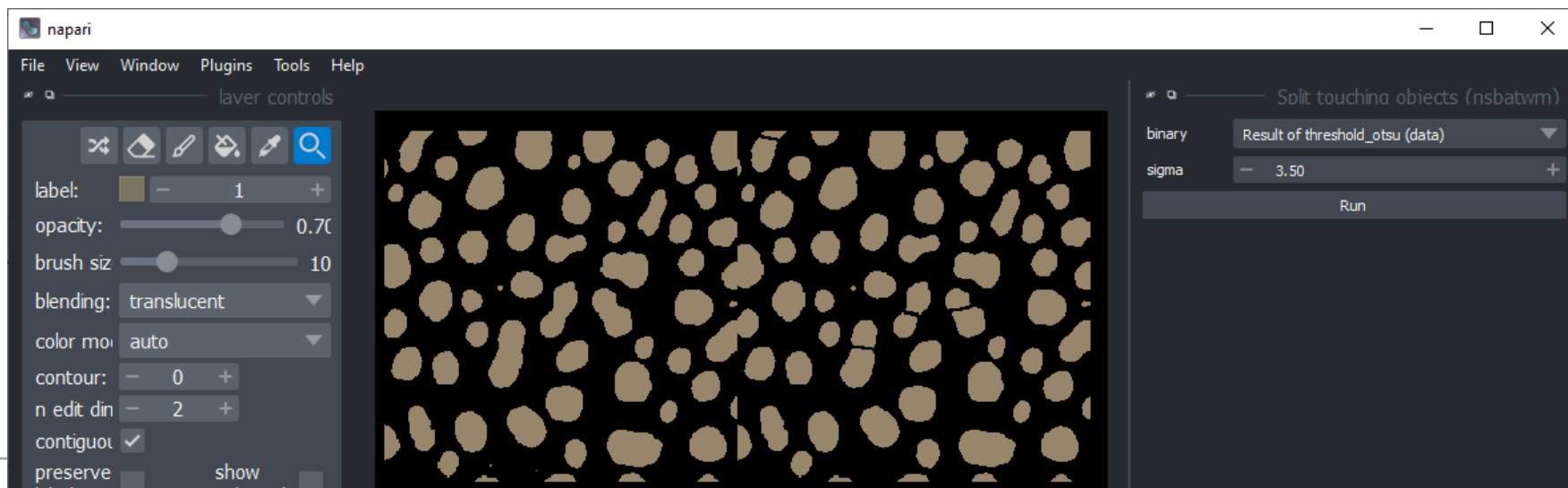
# Watershed

- From binary images

Tools > Segmentation / labeling >  
Label touching objects

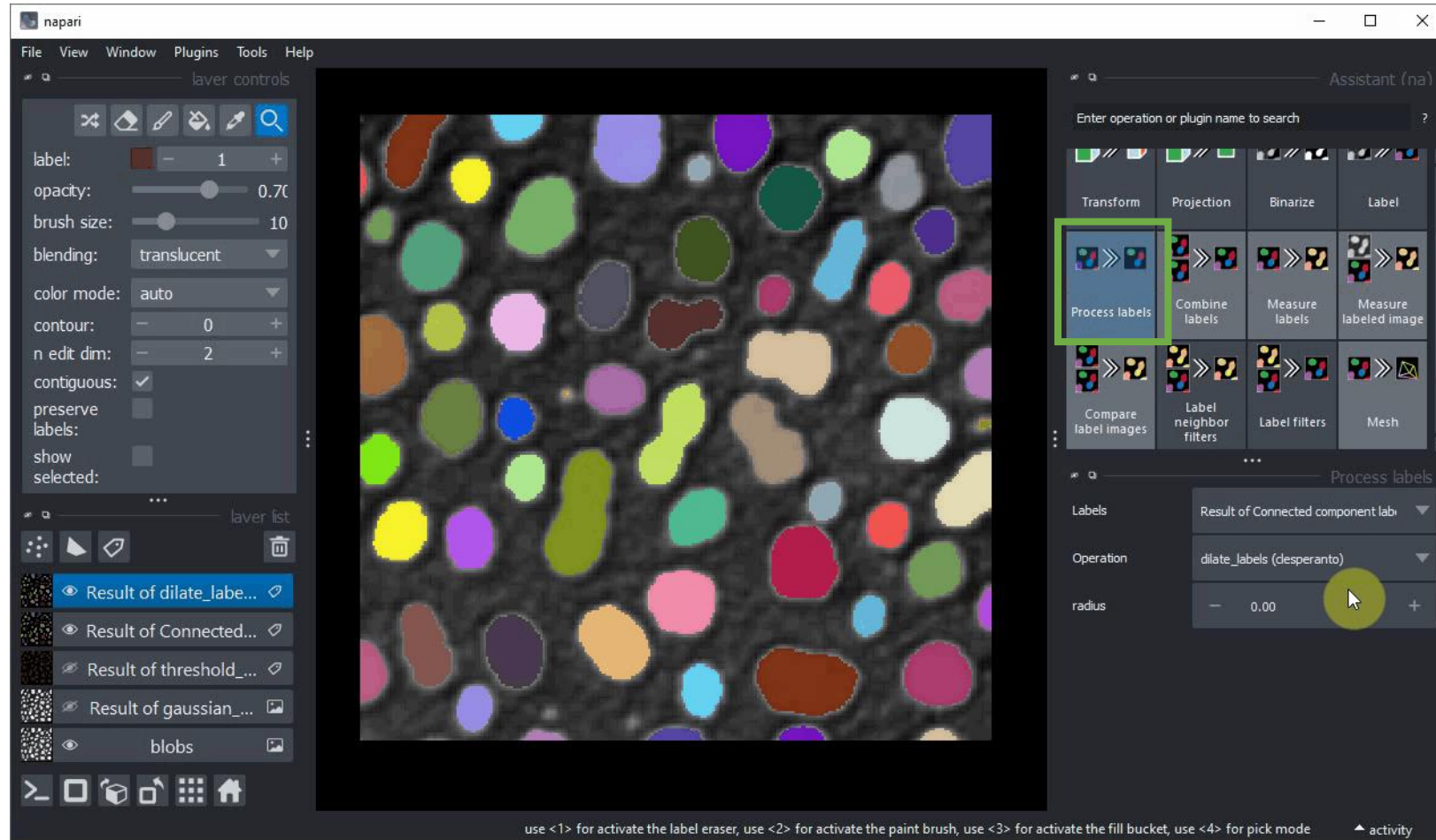


Tools > Segmentation post-  
processing >  
Split touching objects  
(Similar to ImageJ's Watershed)



# Label erosion, dilation, opening, closing, ...

- In Napari Assistant: Process labels





# Browse operations

- Use the search...

This only works if developers documents their plugins well ;-)

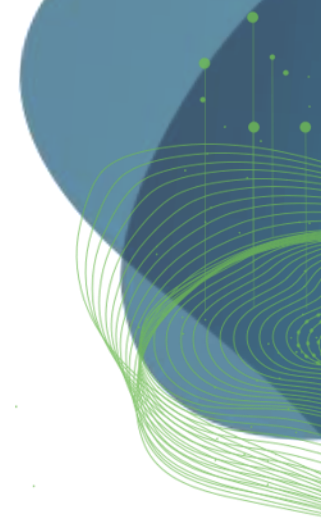
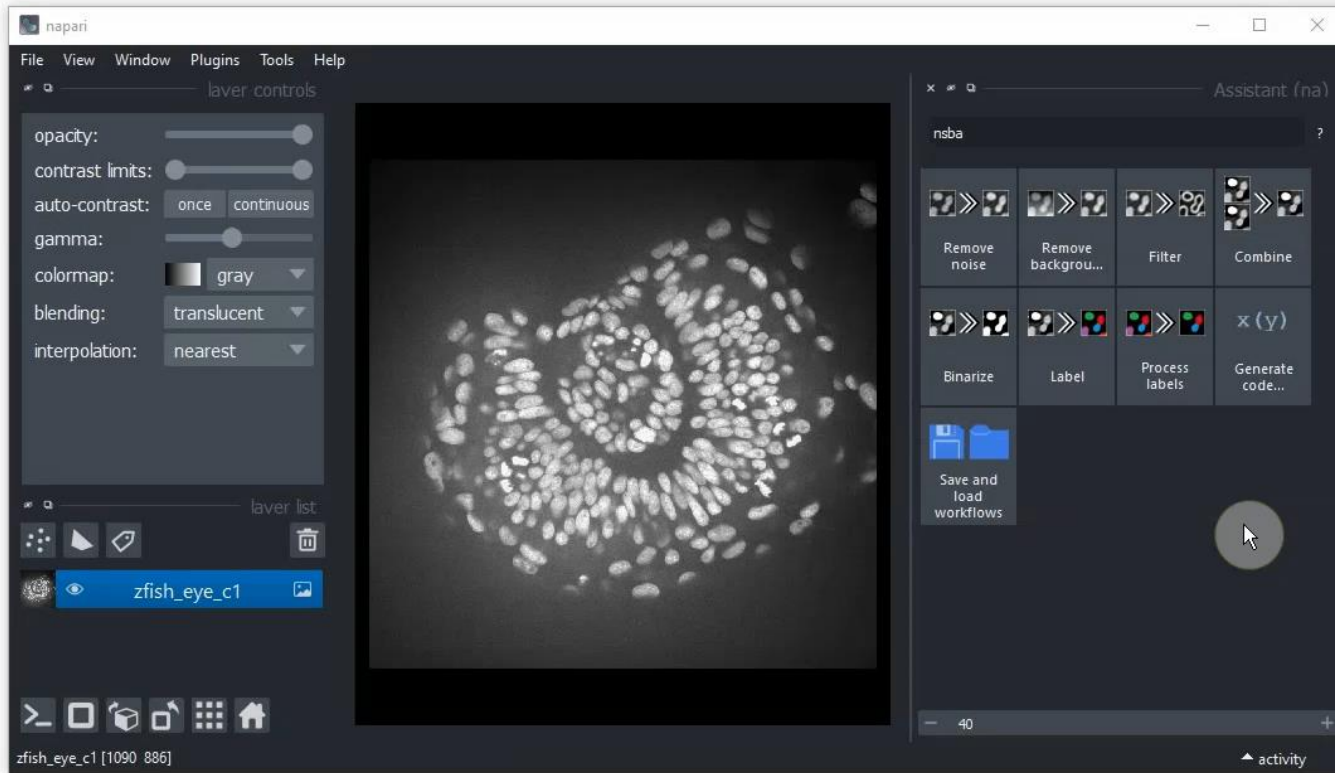
Enter the library name you want to use

Enter the structure you would like to segment

Search the internet

The image displays four sequential screenshots of the Napari Assistant search interface. Each window has a search bar at the top and a grid of operation icons below. The first window shows a search for 'scikit-ima' with results like 'Remove noise', 'Remove background', 'Filter', and 'Combine'. The second window shows a search for 'membrane' with results like 'Remove background', 'Label', 'Generate code...', and 'Save and load workflows'. The third window shows a search for 'the unknown' with results like 'Search napari hub', 'Search image.sc', and 'Search Bili'. A large blue arrow points to the search bar in the first window.

# Export code to Jupyter Notebooks



# Export code to Jupyter Notebooks

napari Assistant (na)

Enter operation or plugin name to search ?

Remove noise Remove background Filter Combine

Transform Projection Binarize Label

Process labels Combine labels Measure labels Measure labeled image

Compare label images Label neighbor filters Label filters Mesh

x (y) Ger co

Export Python script to file  
Export Jupyter Notebook  
Export Jupyter Notebook using Napari  
Copy Python code to clipboard

use <1> for activate the label eraser, use <2> for activate the paint brush, use <3> for activate the

testipynb - JupyterLab

localhost:8888/lab/tree/test.ipynb

File Edit View Run Kernel Tabs Settings Help

Launcher test.ipynb Python 3 (ipykernel)

### threshold otsu

```
[5]: image2_to = cle.threshold_otsu(image1_gb)
image2_to
```

```
[5]:
```

**cle\_image**

shape (254, 256)  
dtype uint8  
size 63.5 kB  
min 0.0  
max 1.0

### connected component labeling

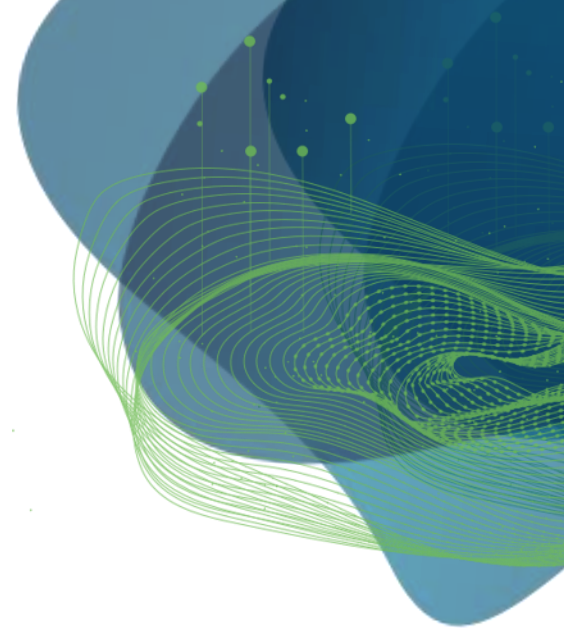
```
[6]: image3_C = nsbatwm.connected_component_labeling(image2_to, False)
image3_C
```

```
[6]:
```

**nsbatwm made image**

shape (254, 256)  
dtype int64  
size 508.0 kB  
min 0

Simple 0 1 Python 3 (ipykernel) |... Mode: Comma... Ln 1, Co... test.ipy...



# Image segmentation in Python

Robert Haase

GEFÖRDERT VOM

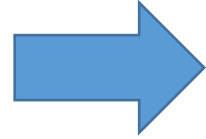


Bundesministerium  
für Bildung  
und Forschung

Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

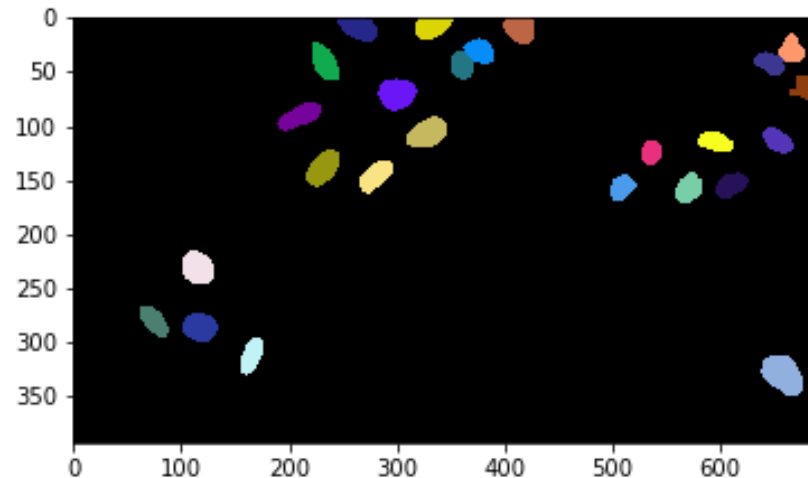
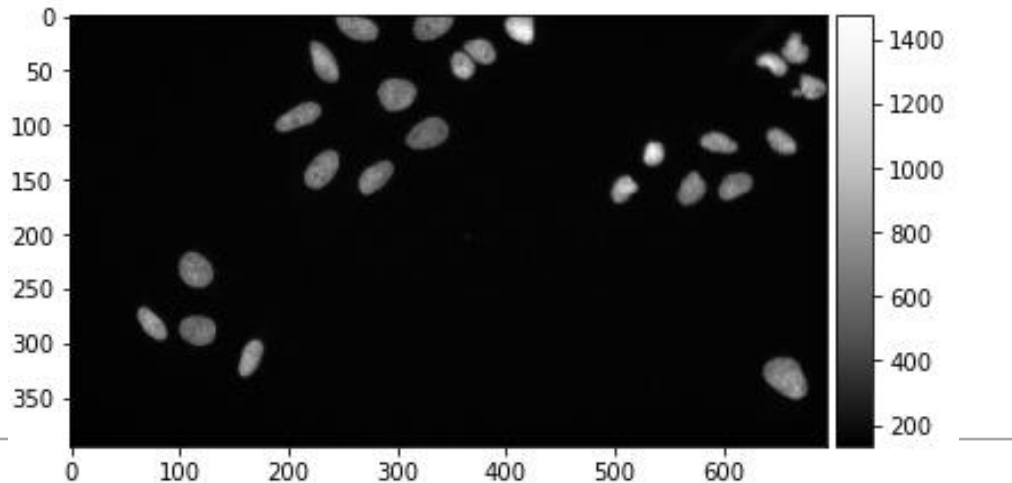
# Voronoi-Otsu-Labeling

- Gaussian-Blur
- Otsu-Thresholding
- 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



# Anisotropy

- Some [segmentation] algorithms have prerequisites...

```
[1]: import pyclesperanto_prototype as cle
```

```
[ ]: cle.voronoi_otsu_labeling(  
[ ]:
```

## Docstring:

Labels objects directly from grey-value images.

The two sigma parameters allow tuning the segmentation result. Under the hood, this filter applies two Gaussian blurs, spot detection, Otsu-thresholding [2] and Voronoi-labeling [3]. The thresholded binary image is flooded using the Voronoi tessellation approach starting from the found local maxima.

## Notes

-----

\* This operation assumes input images are isotropic.

## Parameters

-----

source : Image

Input grey-value image

label\_image\_destination : Image, optional

Output image

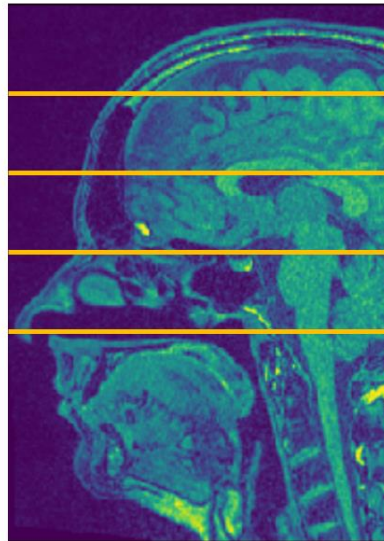
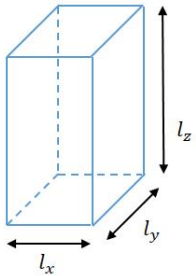
spot sigma : float, optional

# Anisotropy

- Reminder: Anisotropic images might be tricky to process properly

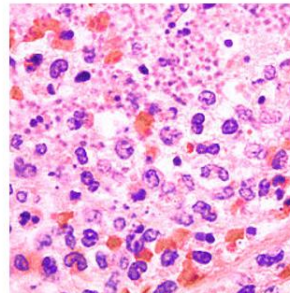
## Image stacks and voxels

- 3-dimensional images consisting of voxels
- “Image stack”
- Often *anisotropic* (not equally large in all directions)



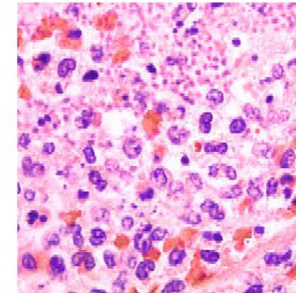
## Anisotropy

- Voxel size has immediate impact on image quality and thus, on processing / analysis results.



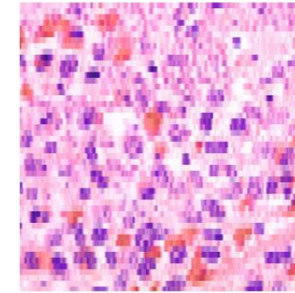
1:1

250 x 250 px



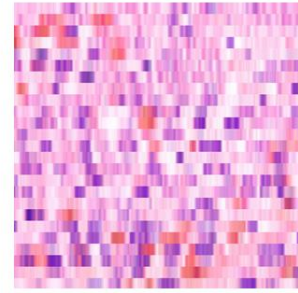
1:2

250 x 125 px



1:5

250 x 50 px



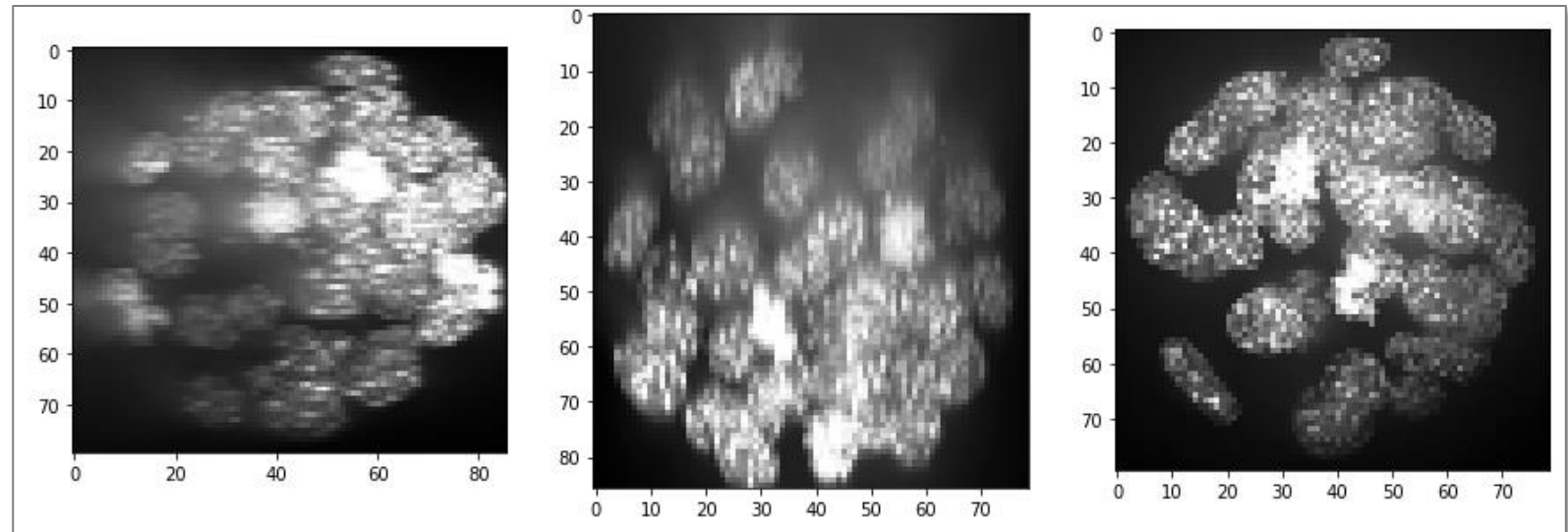
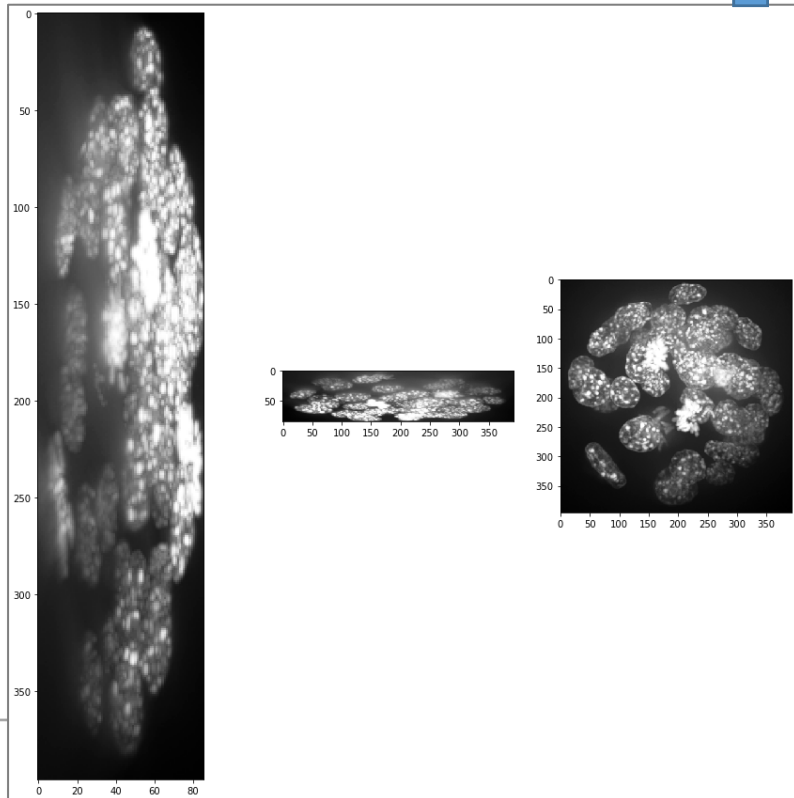
1:10

250 x 25 px

# Reslicing / scaling / sampling

- Resample image data to a specific voxel size

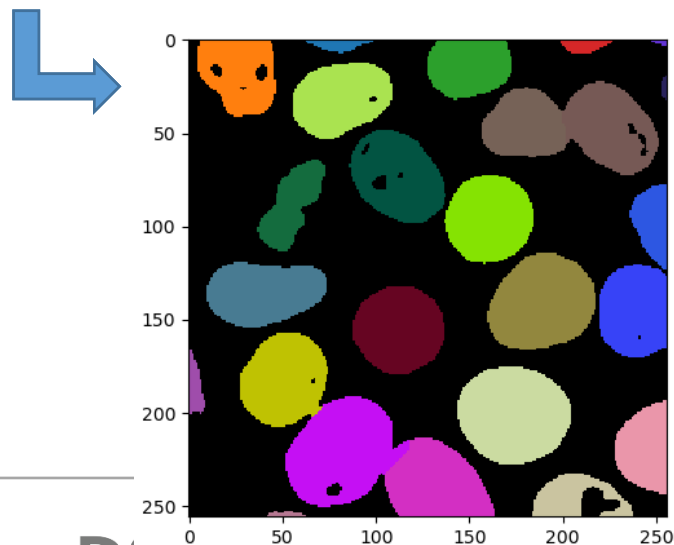
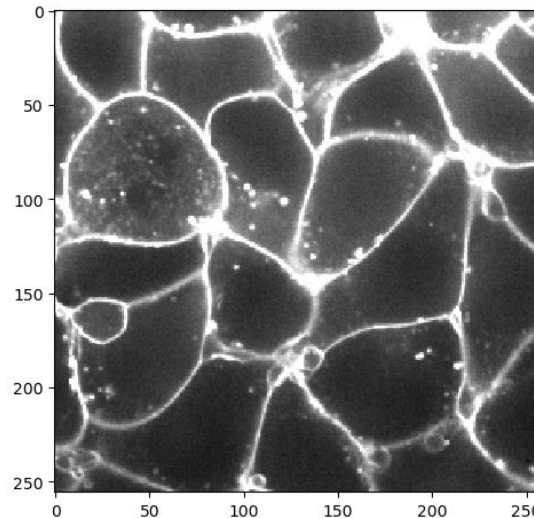
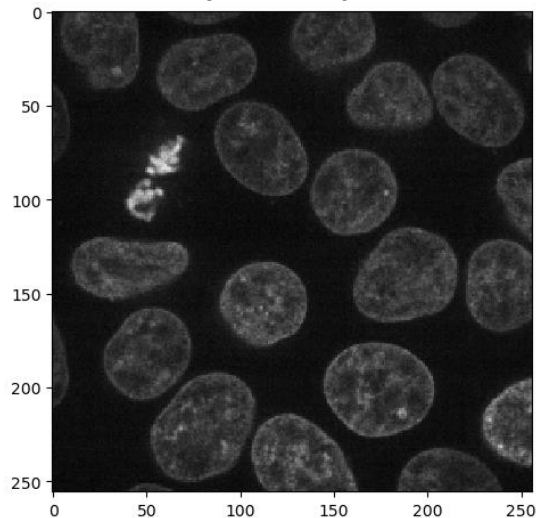
```
resampled = cle.scale(input_image, factor_x=voxel_size_x, factor_y=voxel_size_y, factor_z=voxel_size_z, auto_size=True)  
show(resampled)
```



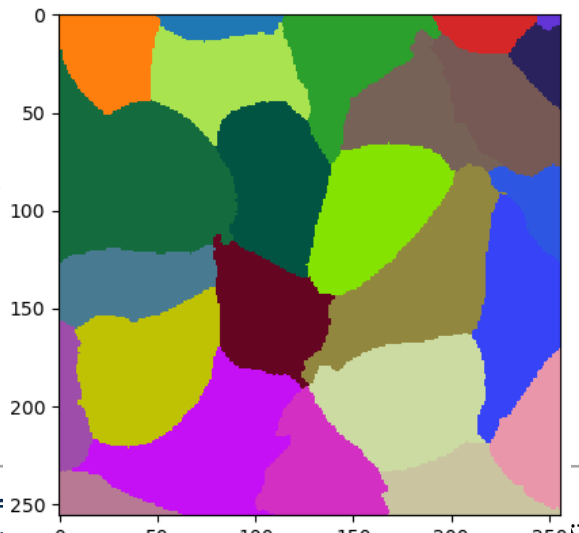


# Watershed

- ... in Python practice

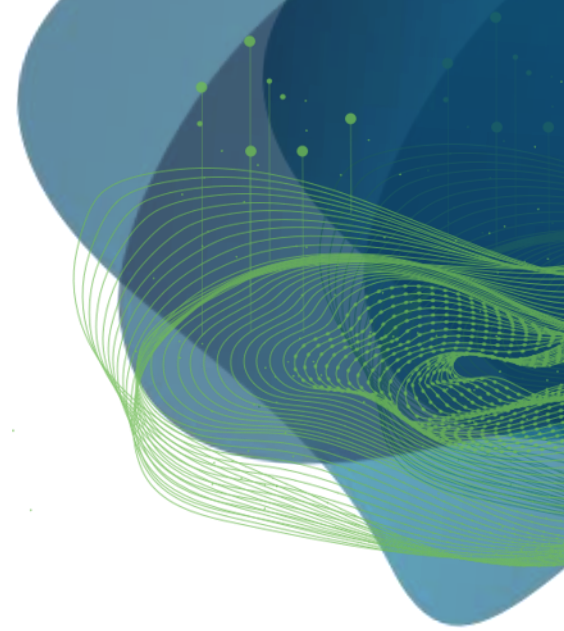


```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)
labeled_cells
```



# Exercises

Robert Haase



GEFÖRDERT VOM



Bundesministerium  
für Bildung  
und Forschung



Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

# Image segmentation exercises

- Try out segmentation algorithms and apply them to other datasets

Applying the algorithm

Voronoi-Otsu-labeling is a segmentation algorithm, which asks for two sigma parameters. The first sigma controls how close detected cells can be ( `spot_sigma` ) and second controls how precise segmented objects are outlined ( `outline_sigma` ). This is the algorithm implemented in the `napari-segment-blobs-and-things-with-membranes`:

```
[3]: label_image = nsbatwm.voronoi_otsu_labeling(cropped_image,
                                             spot_sigma=5,
                                             outline_sigma=1)

label_image
```

[3]:

nsbatwm made image

shape	(200, 200)
dtype	int32
size	156.2 kB
min	0
max	12

Exercise

Load the `blobs.tif` example dataset from last week - without moving the file! Apply the two algorithms Gauss-Otsu-Labeling and Voronoi-Otsu-Labeling to it. Get the number of objects from both images in a variable and print out the variable.

Optional: Write a function that loads the image, segments it and returns the number of objects.

```
[ ]:
```

# Image segmentation exercises

- 3D image processing may require GPU-acceleration
- In case there are errors:

[https://github.com/clEsperanto/pyclesperanto\\_prototype/?tab=readme-ov-file#troubleshooting-graphics-cards-drivers](https://github.com/clEsperanto/pyclesperanto_prototype/?tab=readme-ov-file#troubleshooting-graphics-cards-drivers)

3D Image Segmentation

Image segmentation in 3D is challenging for several reasons: In many microscopy imaging techniques, image quality varies in space: For example intensity and/or contrast degrades the deeper you image inside a sample. Furthermore, touching nuclei are hard to differentiate in an automated way. Last but not least, anisotropy is difficult to handle depending on the applied algorithms and respective given parameters. Some algorithms, like the Voronoi-Otsu-Labeling approach demonstrated here, only work for isotropic data.

```
[1]: from skimage.io import imread
from pyclesperanto_prototype import imshow
import pyclesperanto_prototype as cle
import matplotlib.pyplot as plt

import napari
from napari.utils import nbscreenshot

# For 3D processing, powerful graphics processing units might be necessary
# Here we select a GTX or RTX graphics card if available. If none is available
# the following code might still work, but it may be a bit slow
cle.select_device('TX')
```

[1]: <NVIDIA GeForce RTX 3050 Ti Laptop GPU on Platform: NVIDIA CUDA (1 refs)>

To demonstrate the workflow, we're using cropped and resampled image data from the Broad Bio Image Challenge: Lissa V. Sokolnicki, K.L. Carpenter, A.E. (2012). Annotated.binh.throughout

Segmentation

For segmenting the nuclei in 3D we use the Voronoi-Otsu-Labeling technique we have seen earlier. The algorithm works in 3D as in 2D, but only with isotropic voxels. Thanks to the rescaling applied above, our data is isotropic.

```
[6]: segmented = cle.voronoi_otsu_labeling(background_subtracted, spot_sigma=3, outline_sigma=1
show(segmented, labels=True)
```

Visualize

Exercise

Load the `skimage.data.cells3d` dataset, extract the nuclei channel and segment the nuclei. Print out the number of nuclei, which do not touch the image border.

```
[ ]:
```

# Napari - Exercises

- Start using napari from Python

Opening the napari Viewer

In order to open the viewer, we first have to import napari

```
[2]: import napari
```

Now, we can open the viewer with the following command:

```
[3]: viewer = napari.Viewer()
```

Napari should open in a separated window. Some warning messages in the cell above are normal.

Let's show a screenshot of the viewer here. We pass the variable viewer to the function.

```
[4]: napari.utils.nbscreenshot(viewer)
```

```
[4]:
```

Segmentation visualization

You can also add a segmentation result to the viewer, which will get overlayed with the original image.

```
[13]: blurred = gaussian(mri, sigma=5)
```

```
binary_image = blurred > threshold_otsu(blurred)
```

```
viewer.add_labels(binary_image)
```

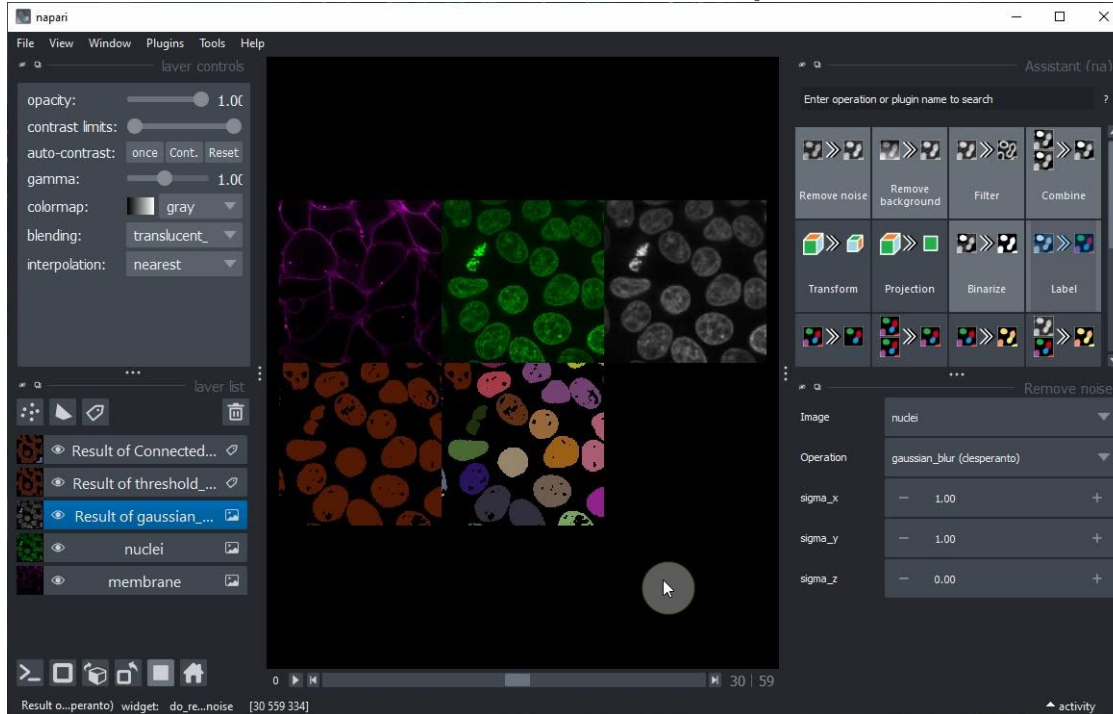
```
napari.utils.nbscreenshot(viewer)
```

```
[13]:
```

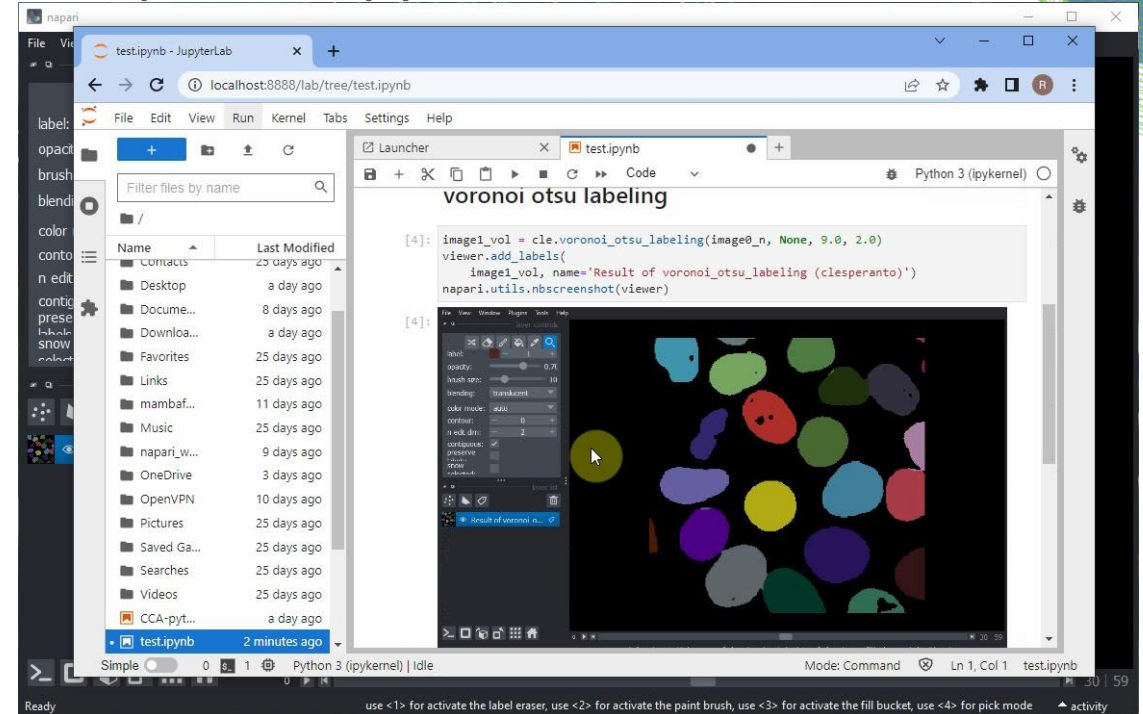


# Napari - Exercises

- Start napari from the terminal!
- Follow the instructions to set up a workflow and export a Jupyter notebook



[https://github.com/ScaDS/BIDS-lecture-2024/blob/main/04b\\_napari\\_notebooks/napari-assistant.md](https://github.com/ScaDS/BIDS-lecture-2024/blob/main/04b_napari_notebooks/napari-assistant.md)



[https://github.com/ScaDS/BIDS-lecture-2024/blob/main/04b\\_napari\\_notebooks/notebook\\_export.md](https://github.com/ScaDS/BIDS-lecture-2024/blob/main/04b_napari_notebooks/notebook_export.md)