



DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS
AND ARTIFICIAL INTELLIGENCE

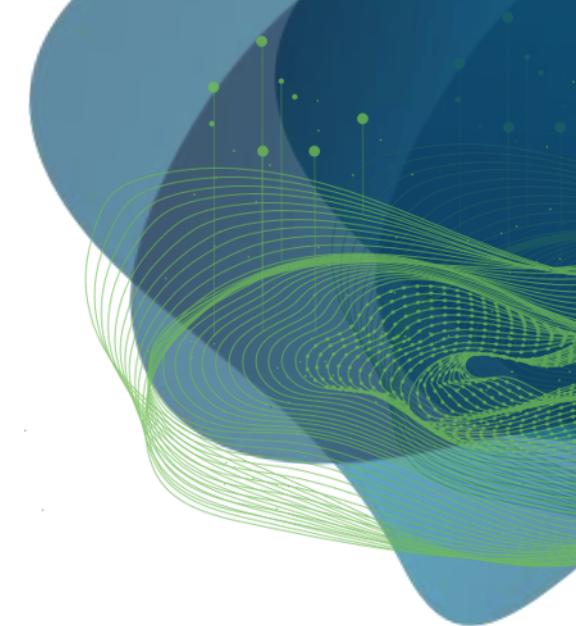


Image segmentation

Robert Haase

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

GEFÖRDERT VOM



Bundesministerium
für Bildung
und Forschung

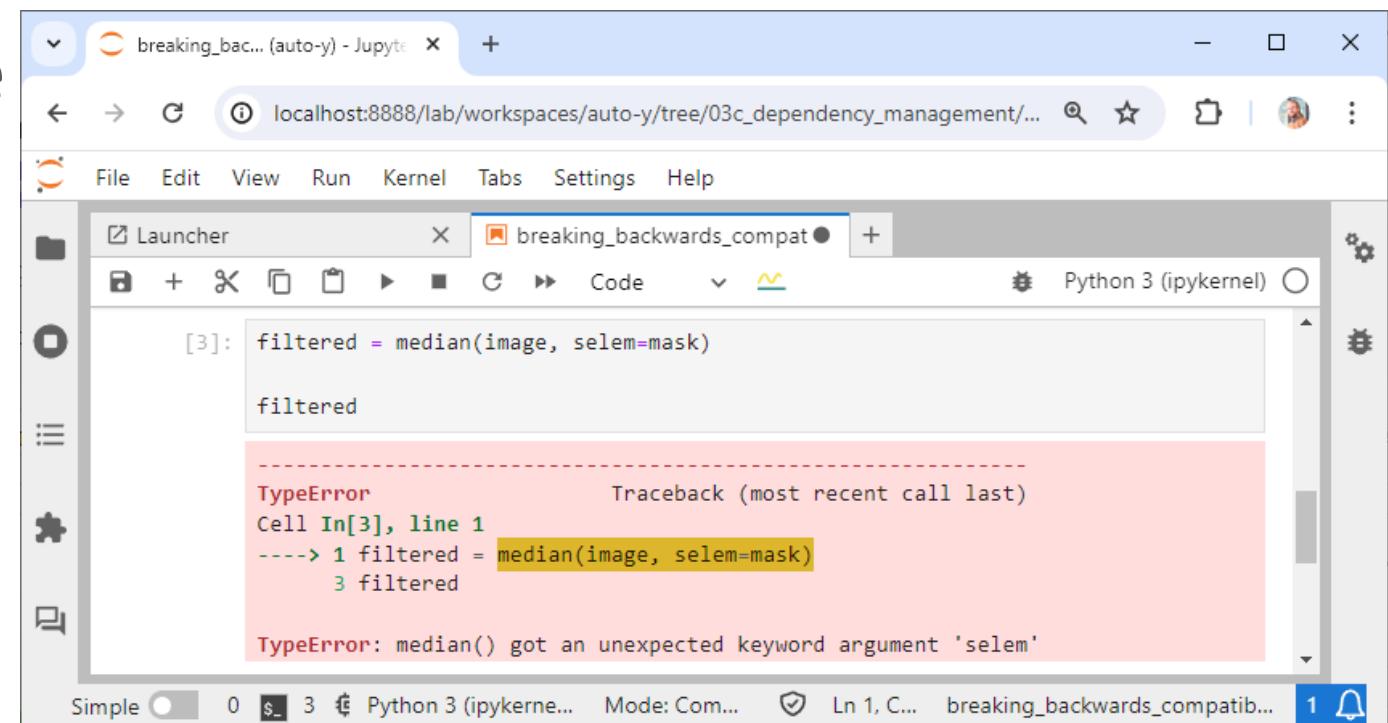


Diese Maßnahme wird gefördert durch die Bundesregierung
aufgrund eines Beschlusses des Deutschen Bundestages.
Diese Maßnahme wird mitfinanziert durch Steuermittel auf
der Grundlage des von den Abgeordneten des Sächsischen
Landtags beschlossenen Haushaltes.

Recap

How can one solve this problem?

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



The screenshot shows a Jupyter Notebook interface with a single code cell. The code in the cell is:filtered = median(image, selem=mask)
filteredA red box highlights the error message in the output area:

```
-----  
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

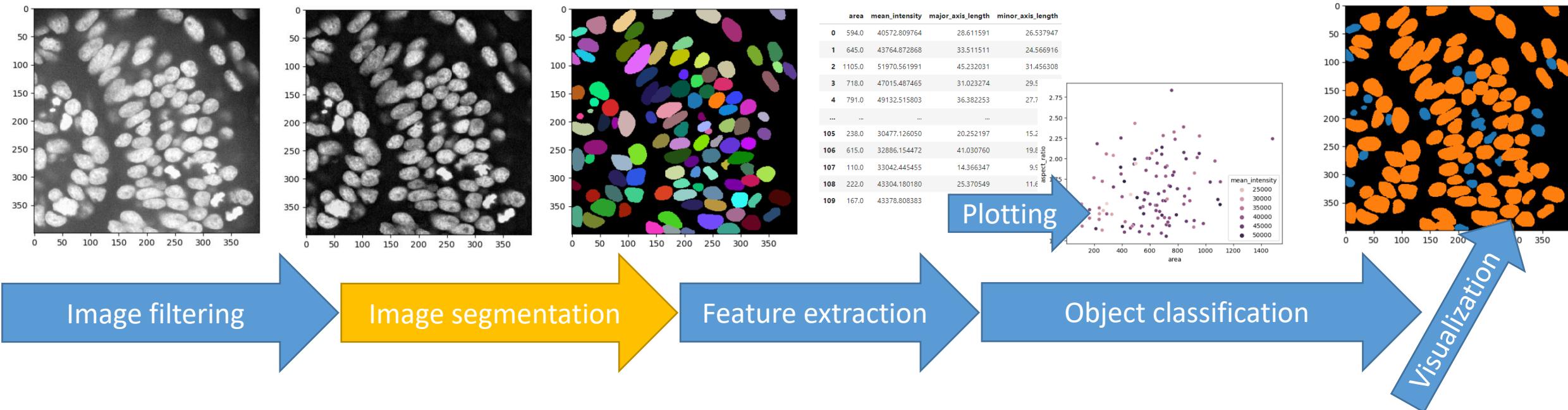


Difference
of Gaussian



Lecture overview: Bio-image Analysis

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



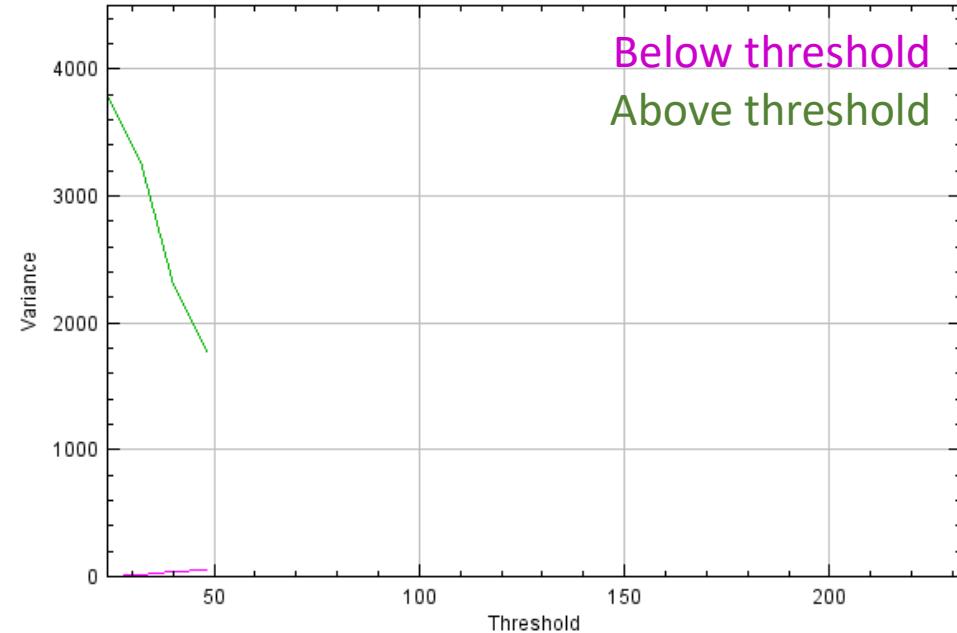
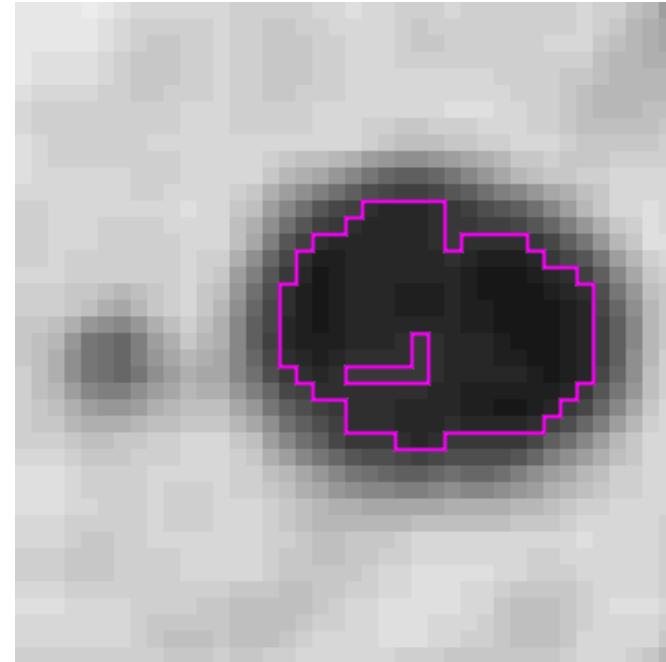
Thresholding: Otsu's method

- 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



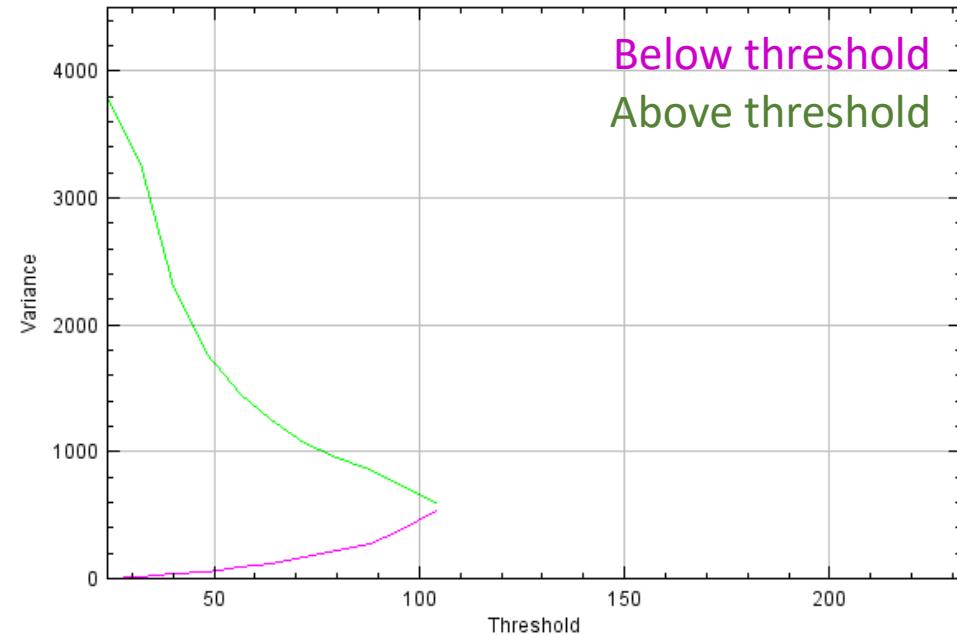
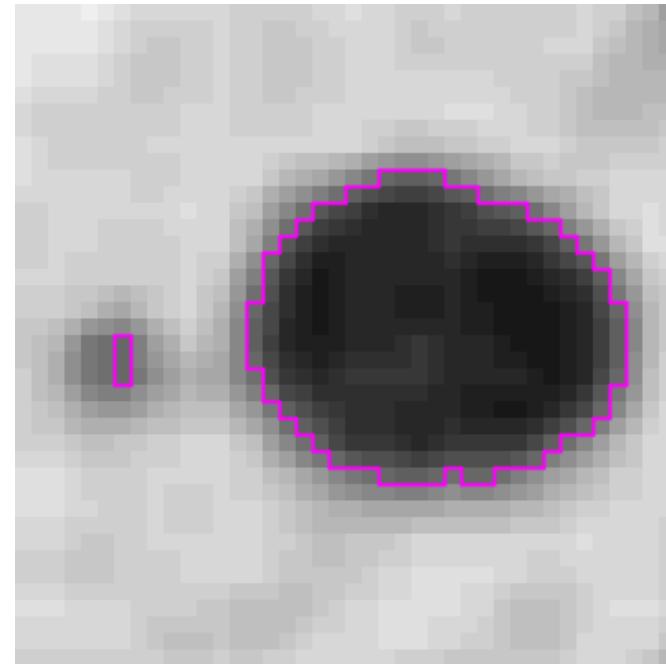
Thresholding: Otsu's method

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

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

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

$Var(I)$... 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



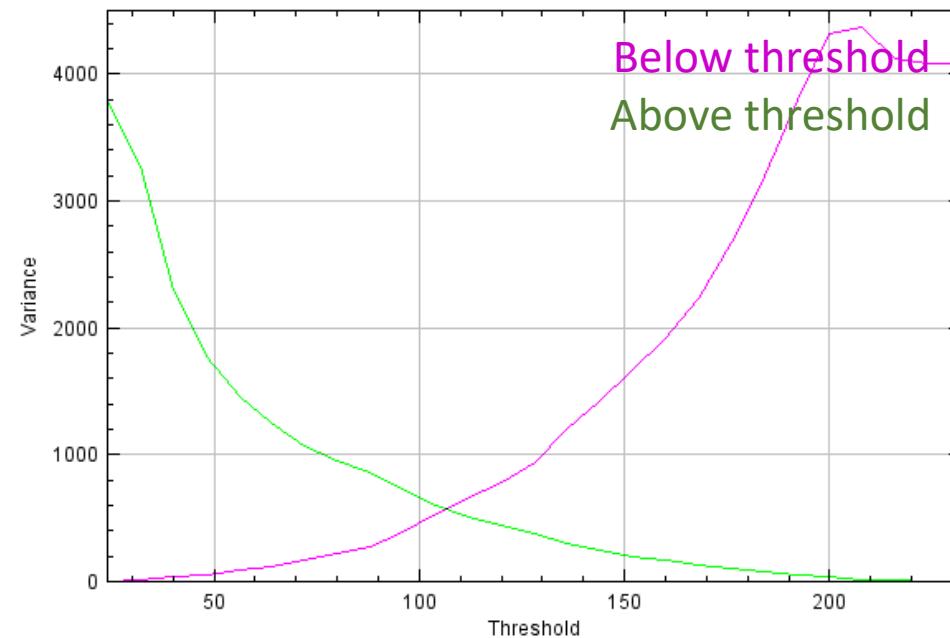
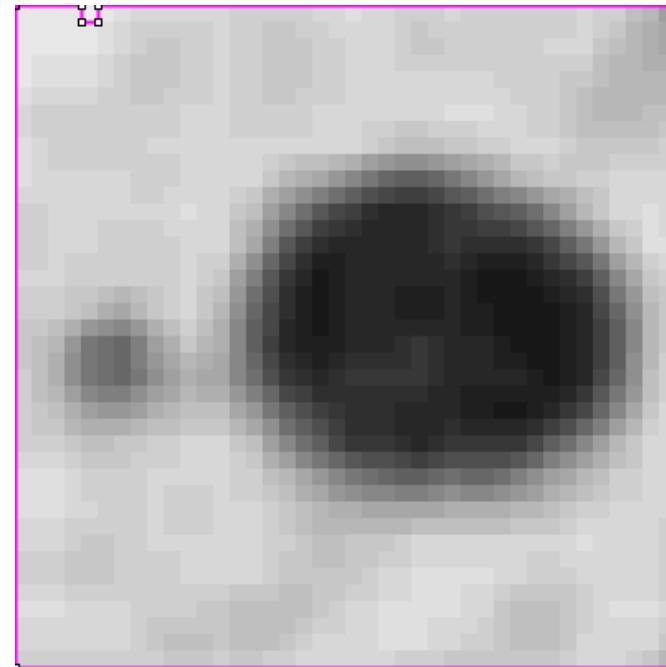
Thresholding: Otsu's method

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

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

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

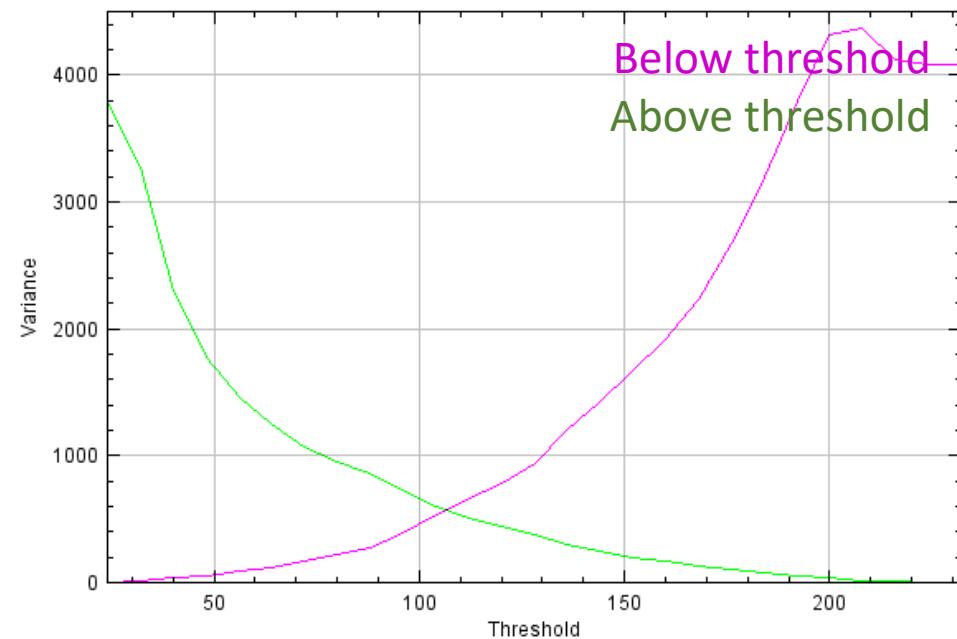
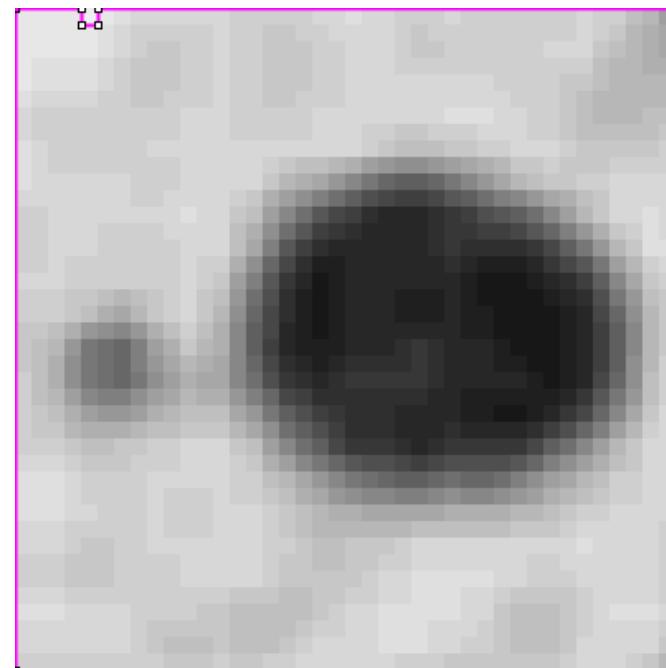
$Var(I)$... Variance in image I
 g_i ... grey value of a pixel i
 \bar{g}_I ... mean grey value of the whole image I
 n_I ... number of pixels in Image I



Thresholding: Otsu's method

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

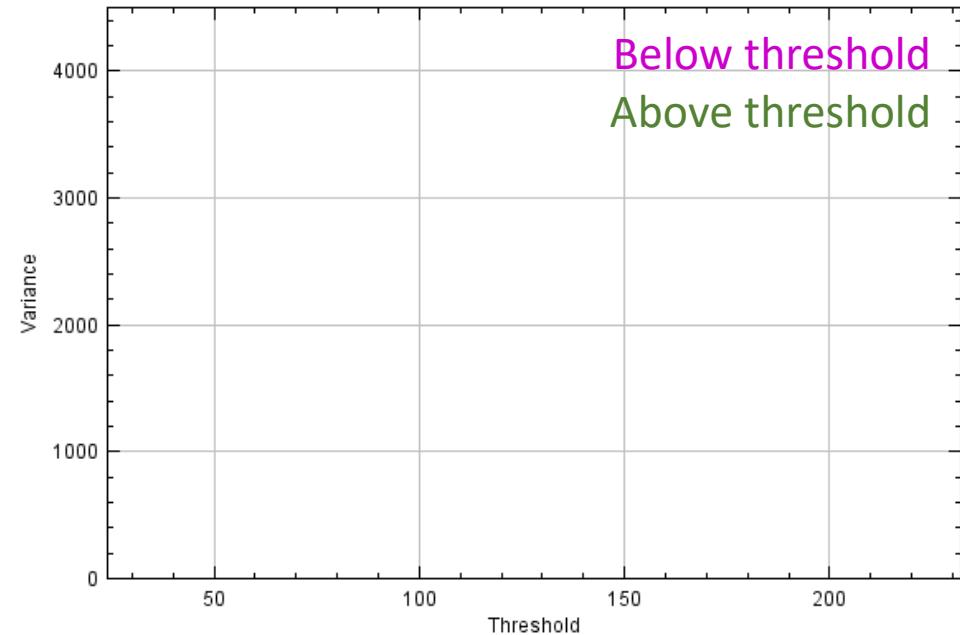
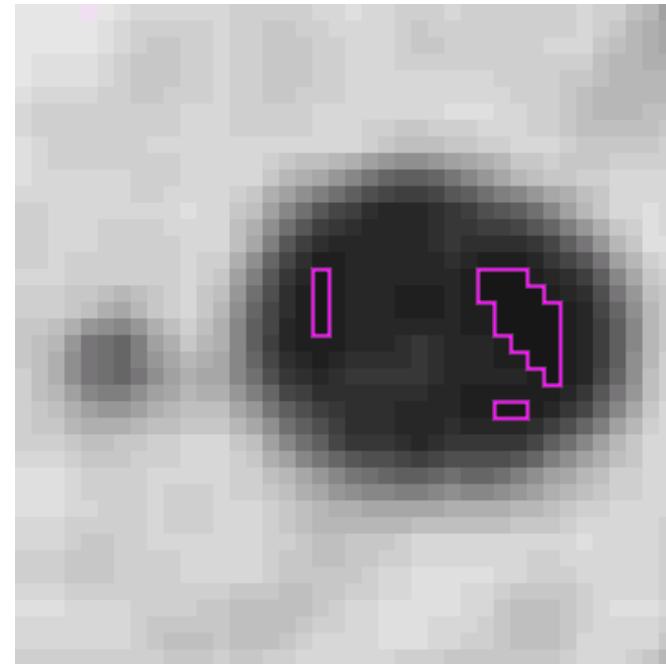
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



Thresholding: Otsu's method

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

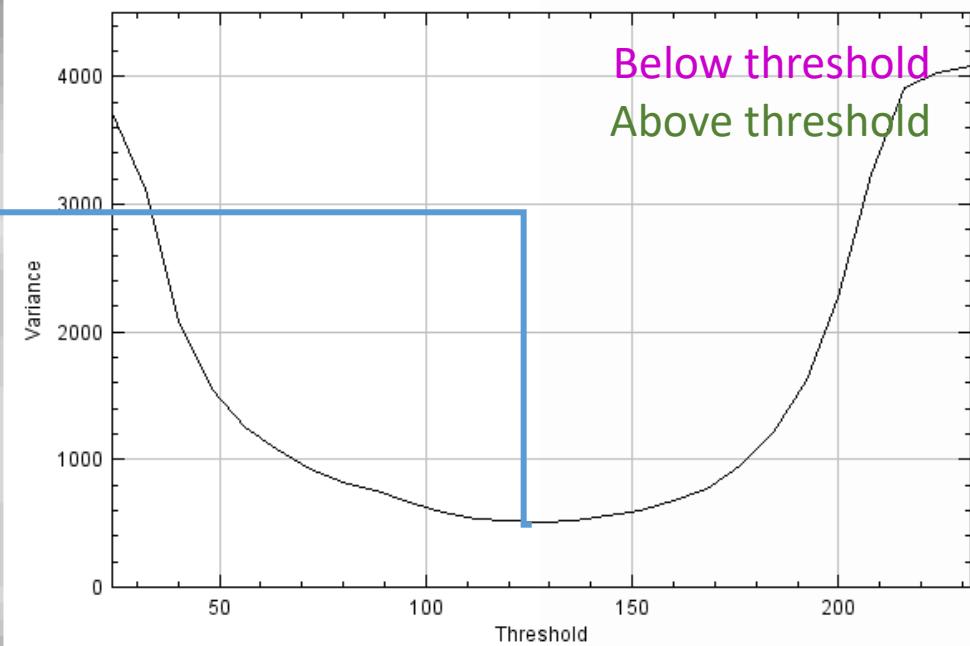
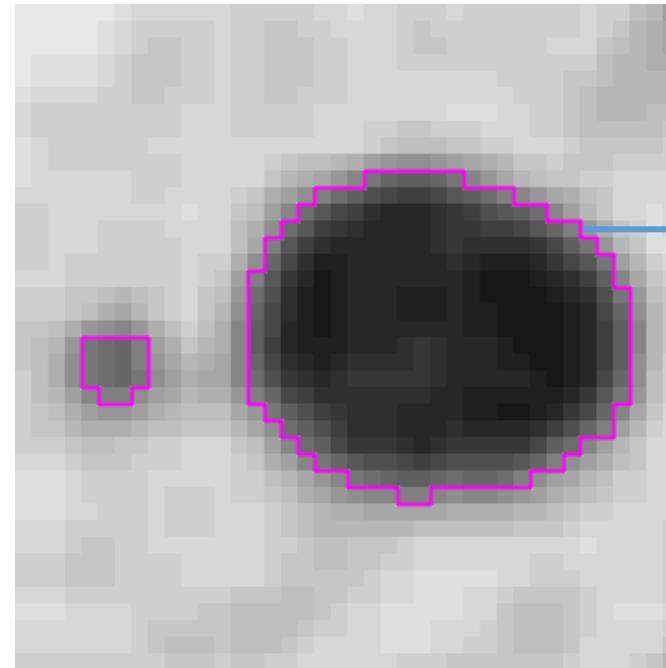
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



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.

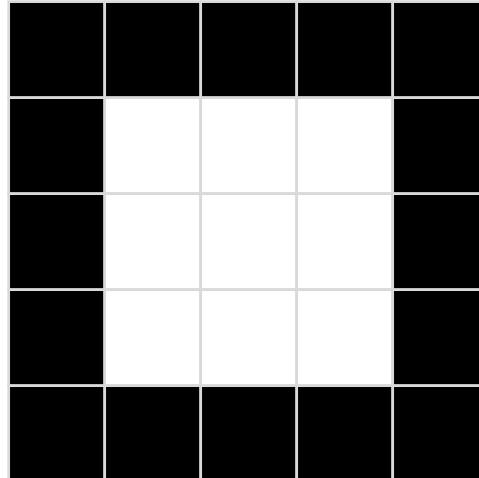
Refining masks

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

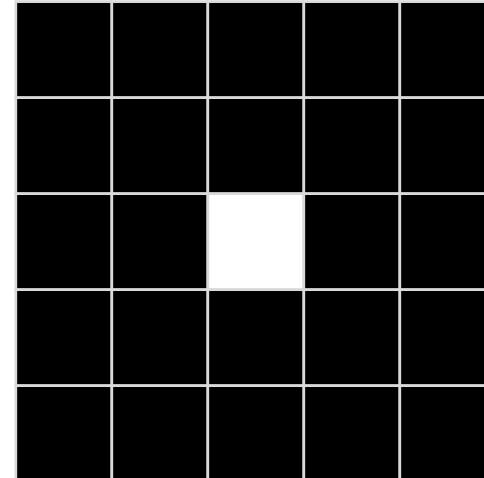


Erosion

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



Erosion



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

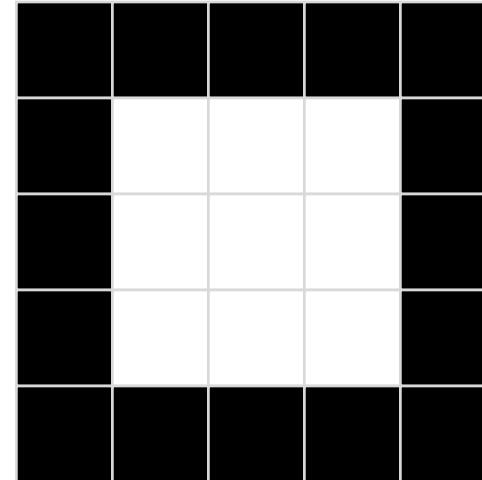
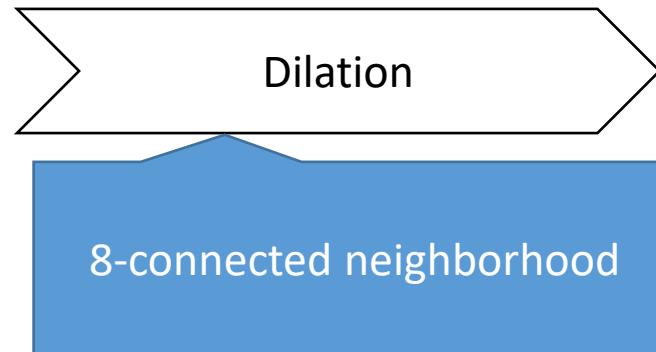
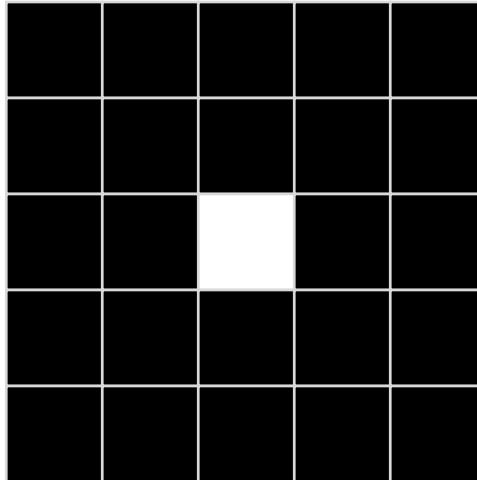


Maximum



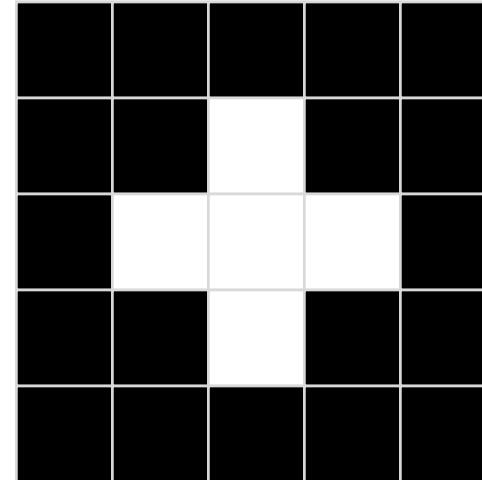
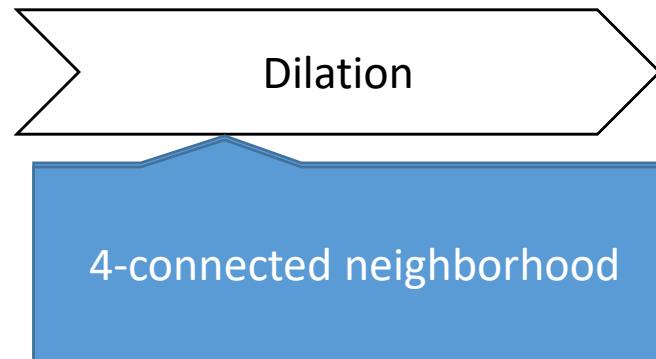
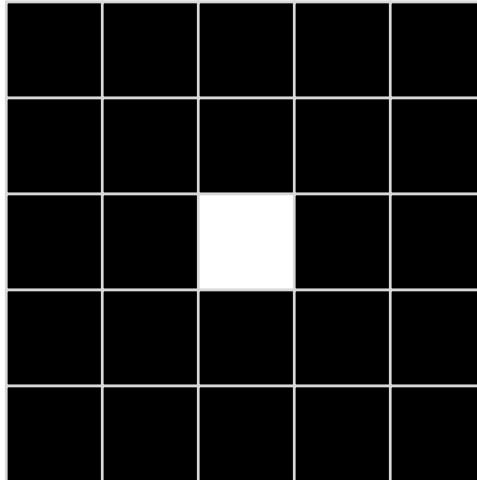
Dilation

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



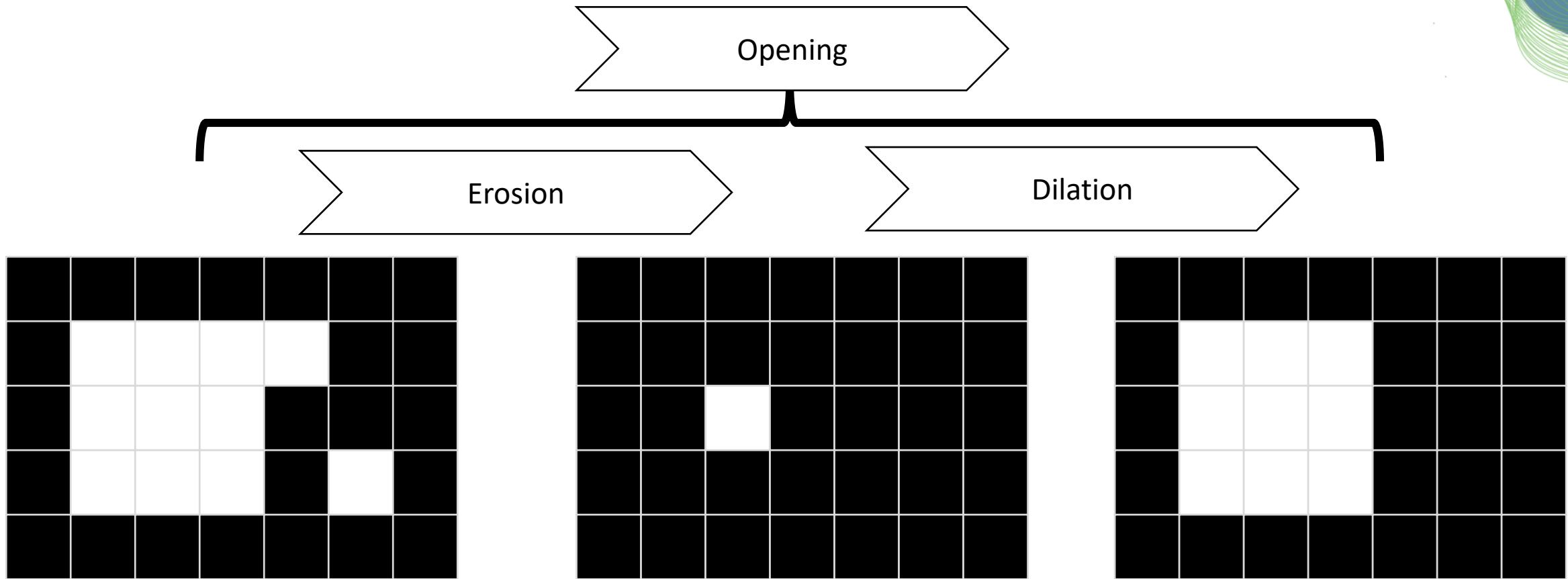
Dilation

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



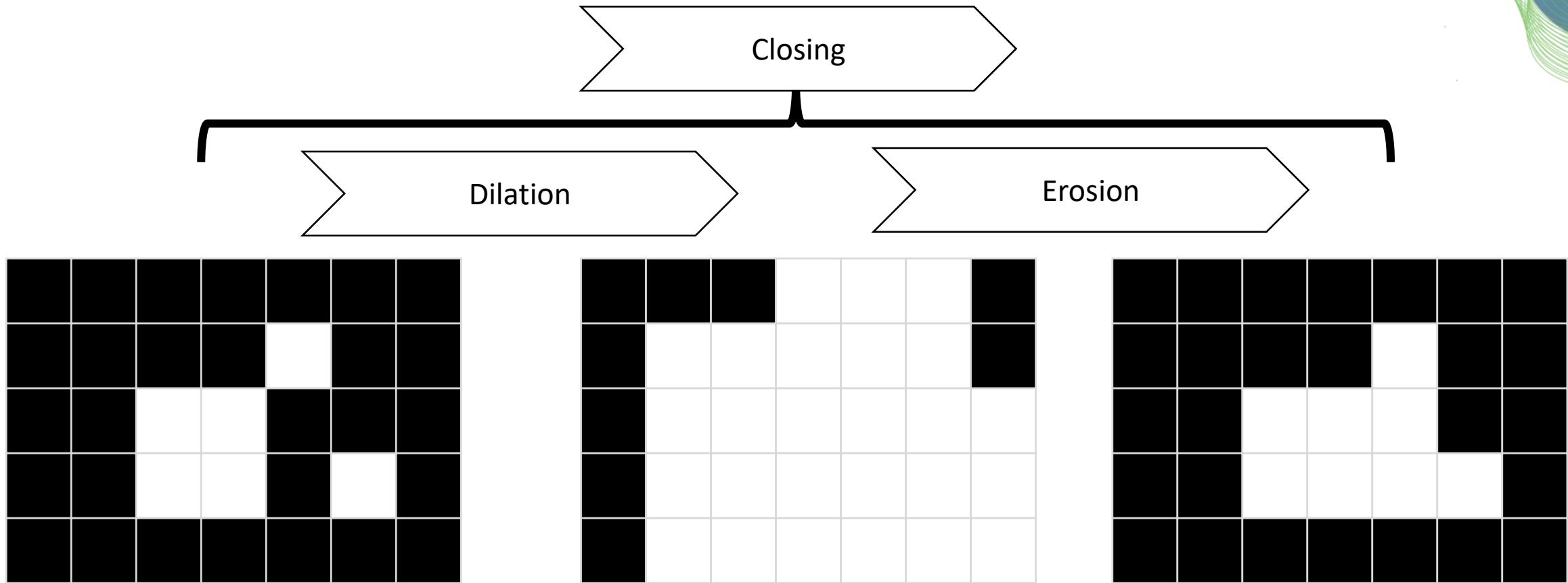
Opening

- Erosion and dilation combined allow correcting outlines.



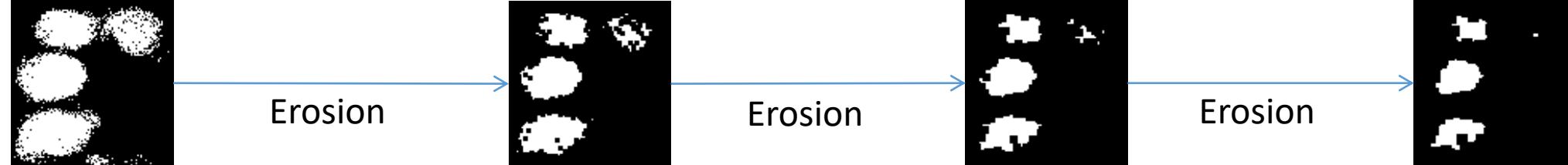
Closing

- 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.



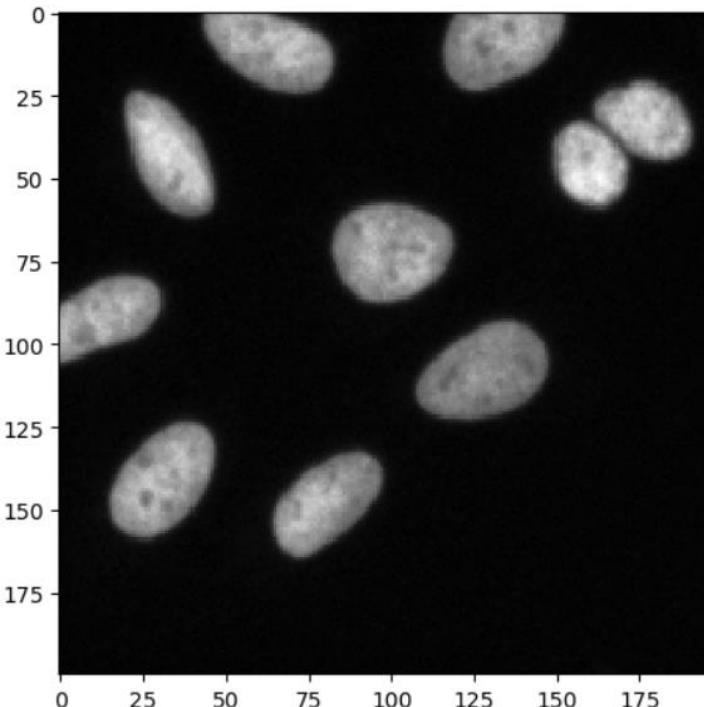
- Closing: Dilation + Erosion



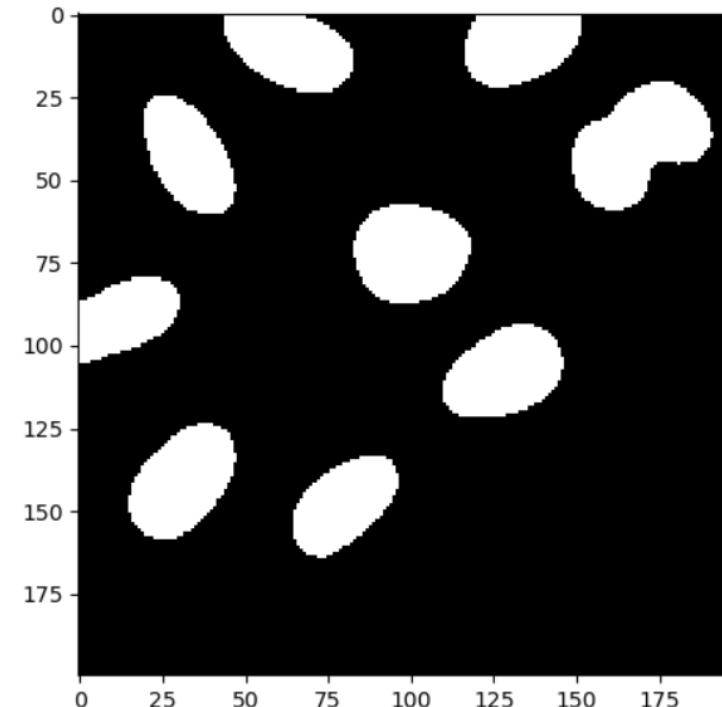
- Opening: Erosion + Dilation

Terminology

Intensity image



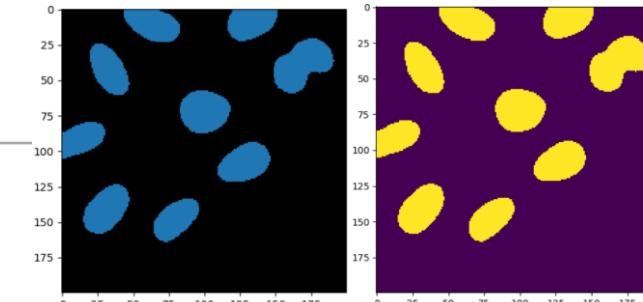
Binary image



Label image



No matter how they are displayed



Terminology

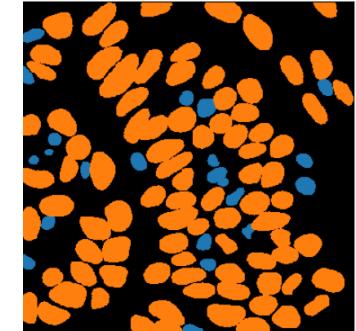
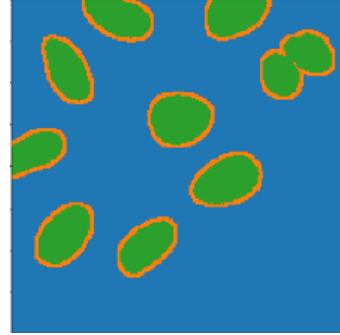
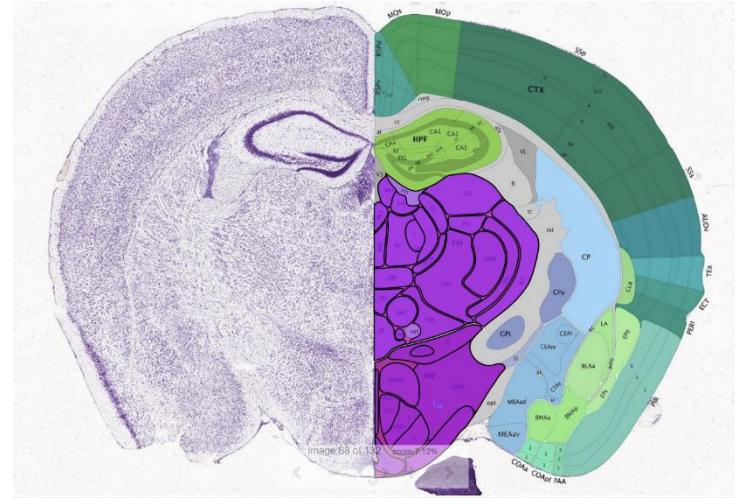
Instance segmentation



Instances:

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

Semantic segmentation



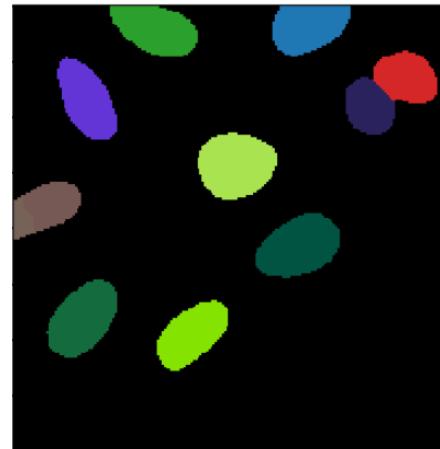
Regions:

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

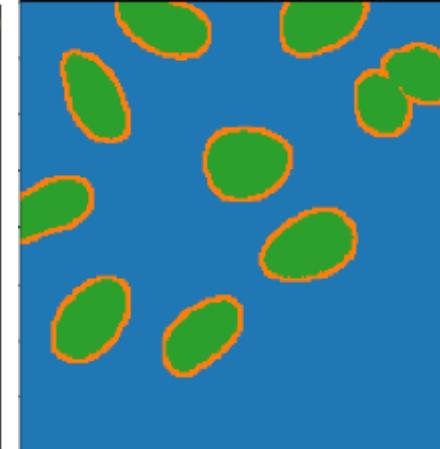
Terminology

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

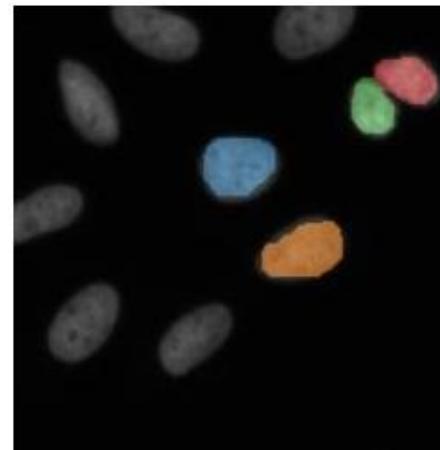
Instance
segmentation



Semantic
segmentation



Sparse instance
annotation

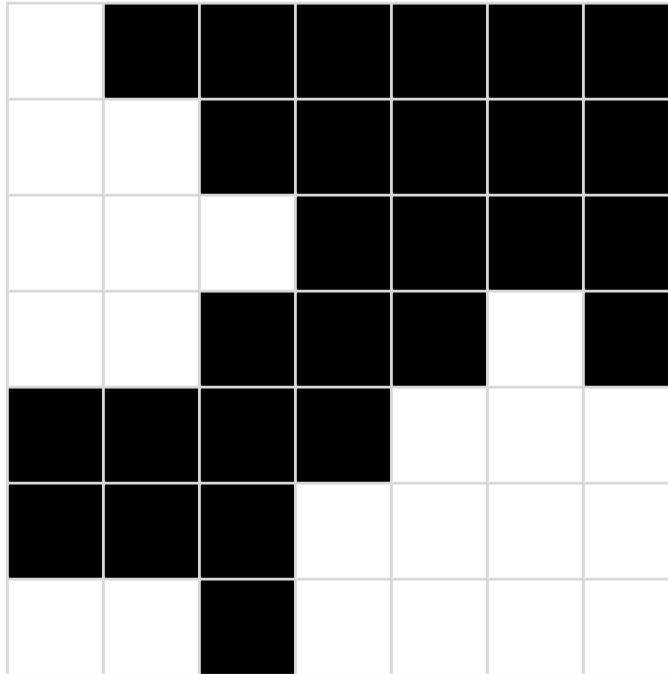


Sparse semantic
annotation



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.



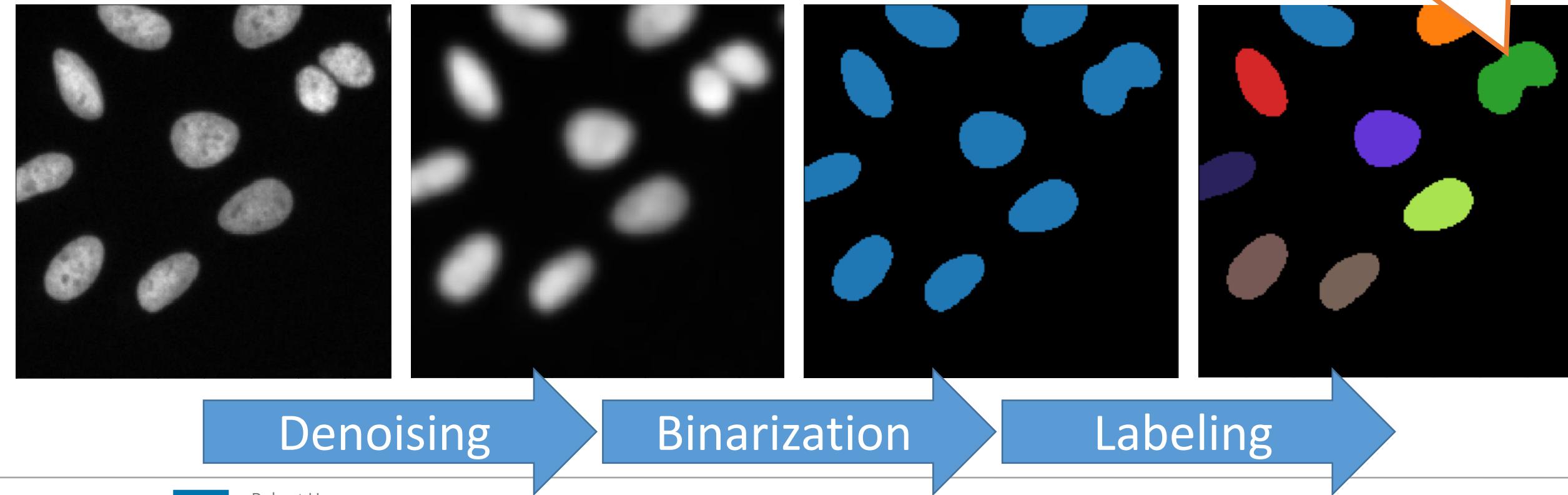
CCA

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

Common image segmentation workflows

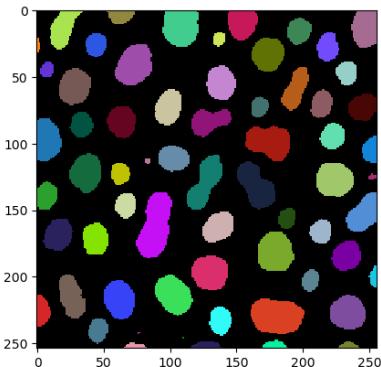
- Presumably the most common segmentation algorithm used for fluorescent microscopy images:
 - Gaussian blur, Otsu's Threshold, Connected Component Labeling

Limitation: Dense objects

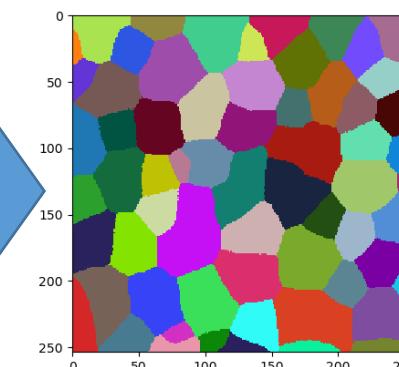
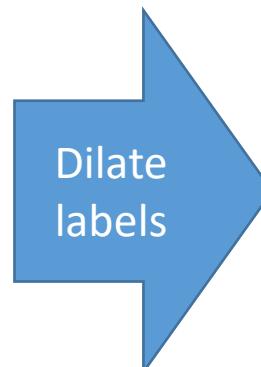
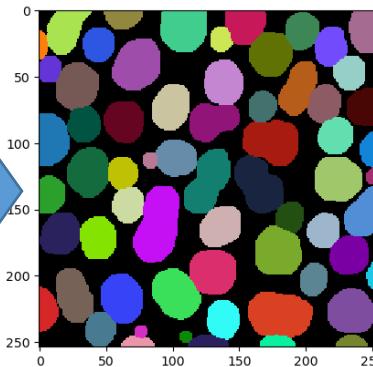
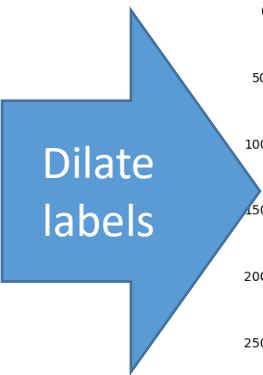


Voronoi-Tesselation

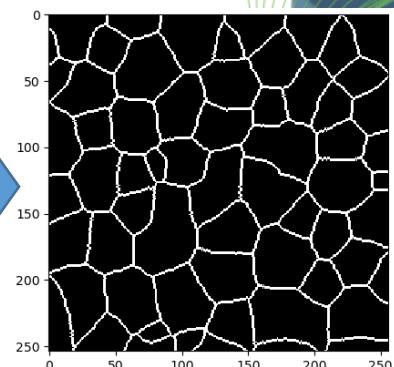
- For separating objects using spatial constraints (not intensity-based)



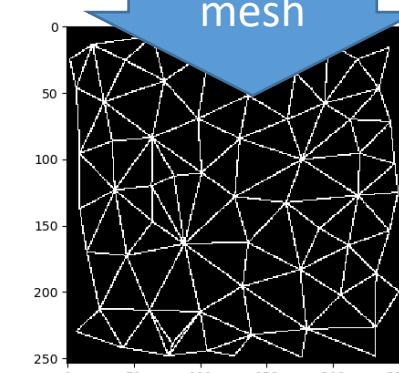
Label-image



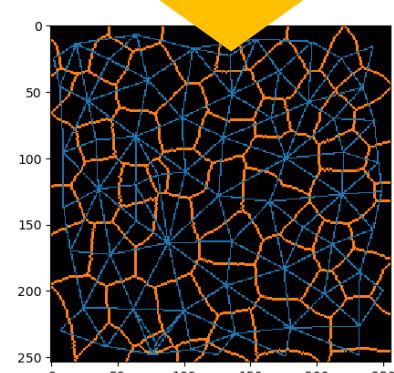
Voronoi-label-image
Centroid mesh



Label-edge-image



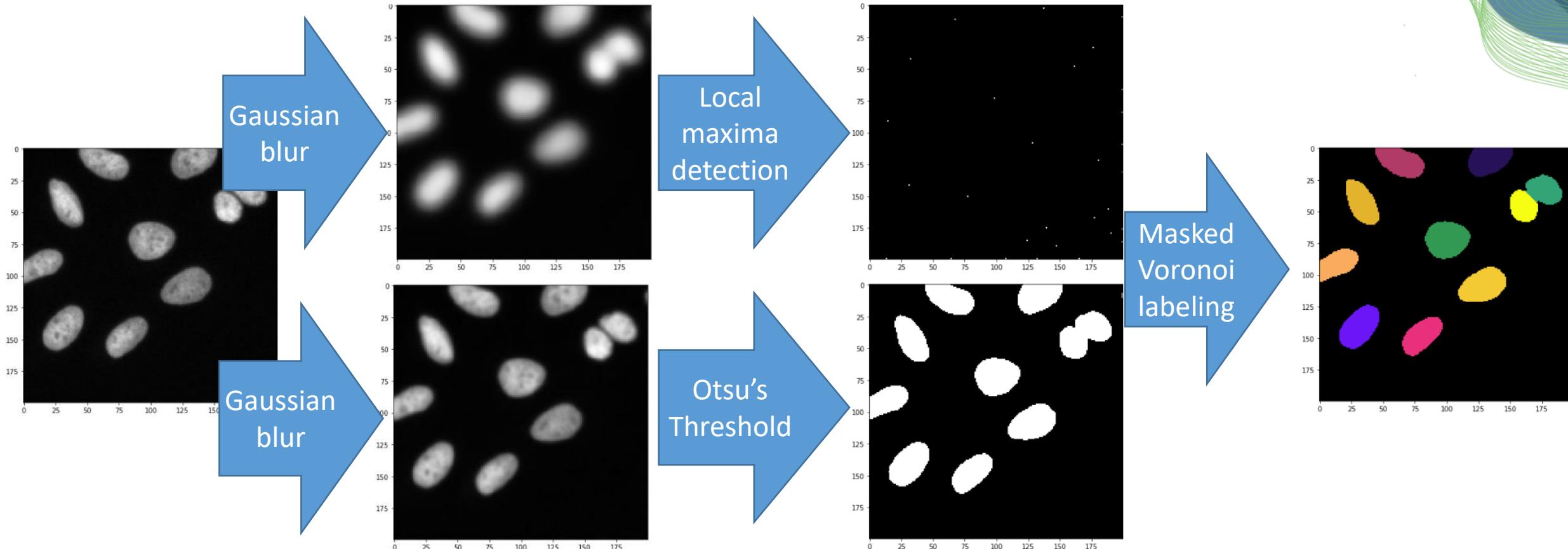
Touching neighbor mesh



Voronoi-Tesselation
Delauney-Triangulation

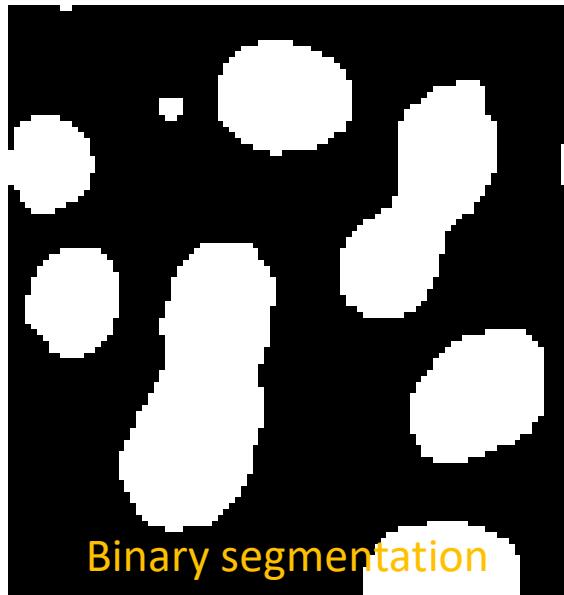
Common image segmentation workflows

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

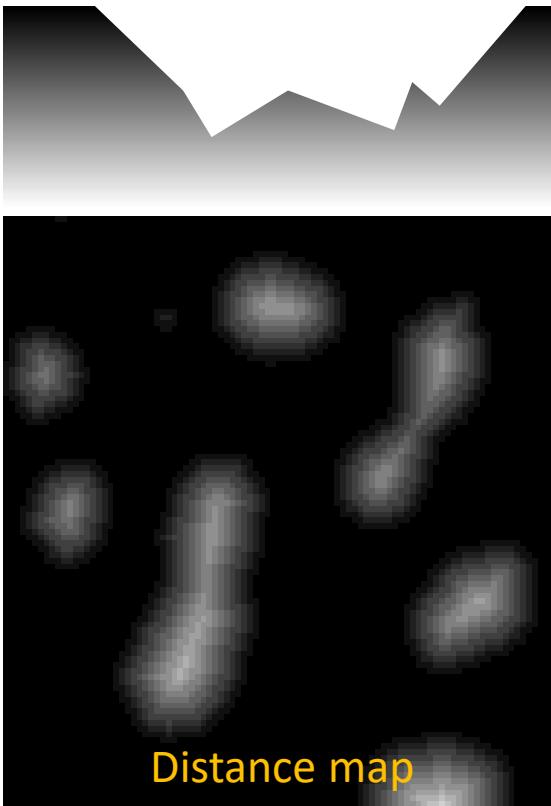


Watershed

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



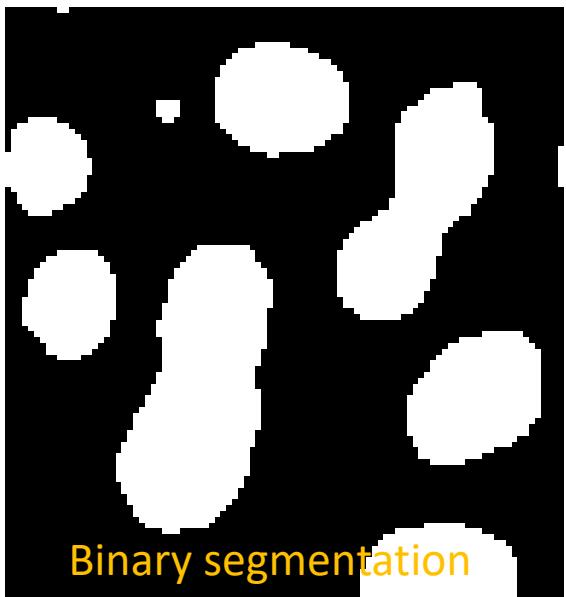
Binary segmentation



Distance map

Watershed

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



Binary segmentation

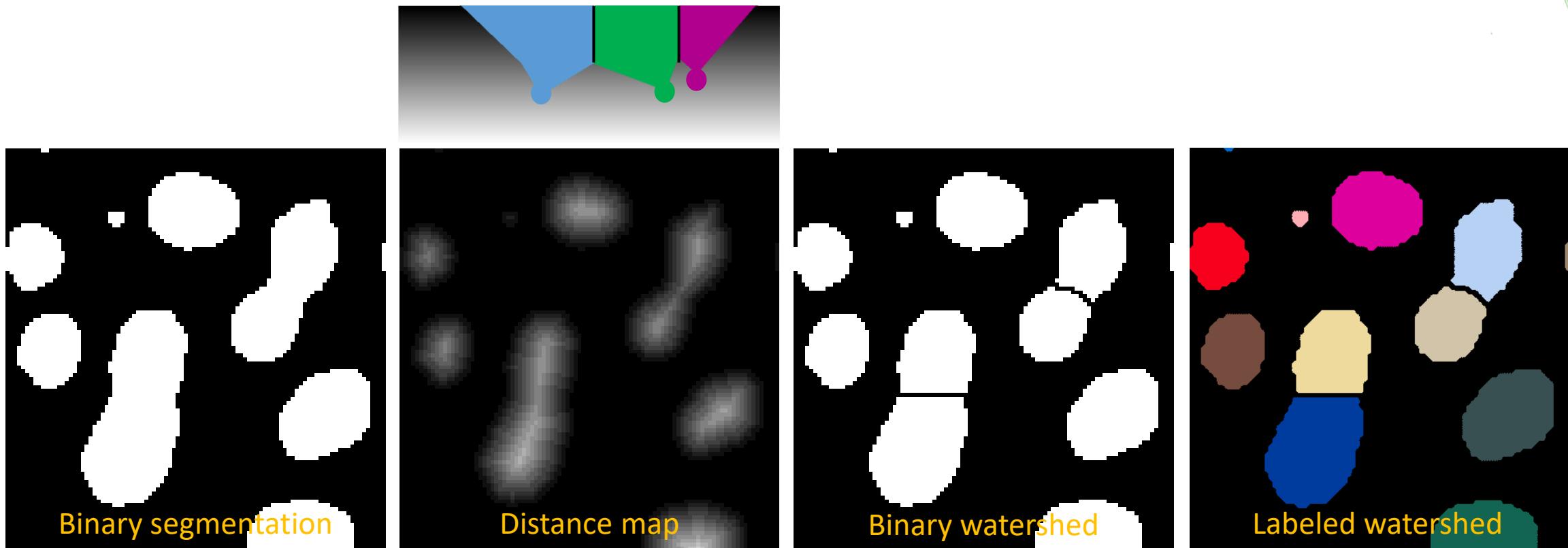


Distance map



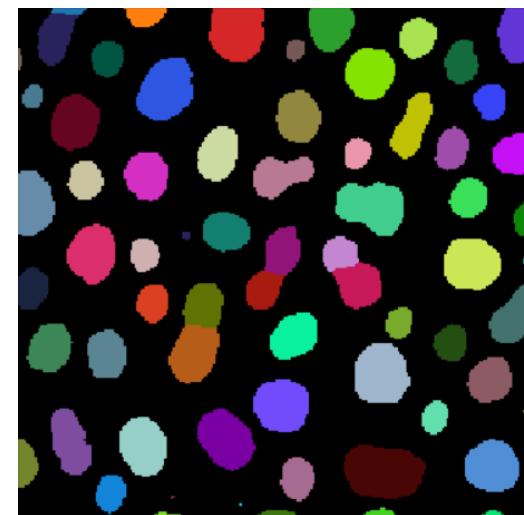
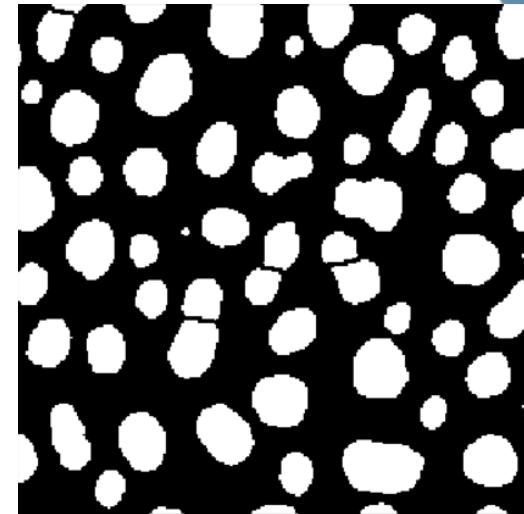
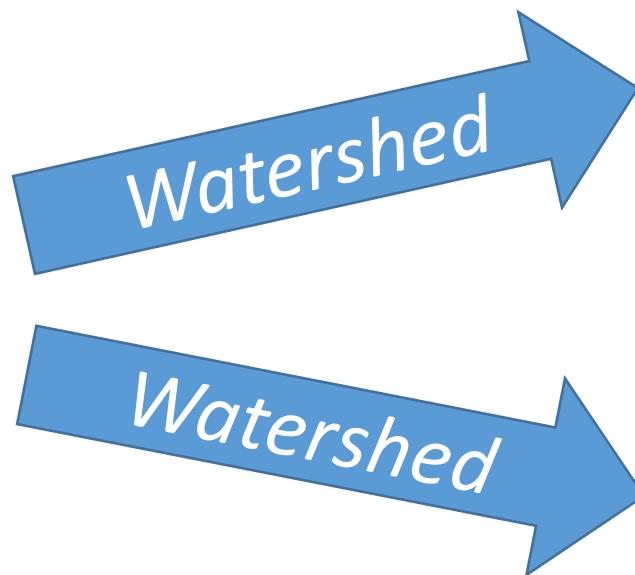
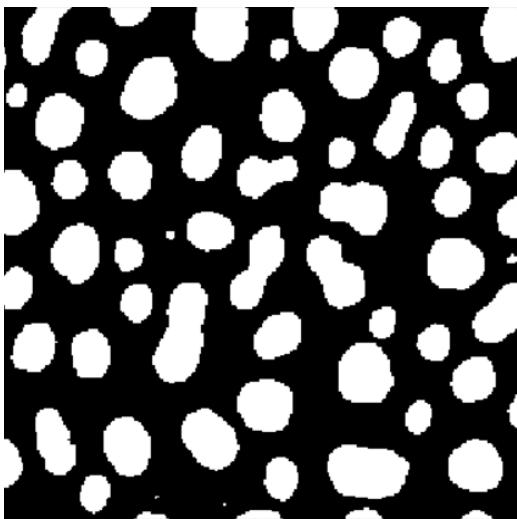
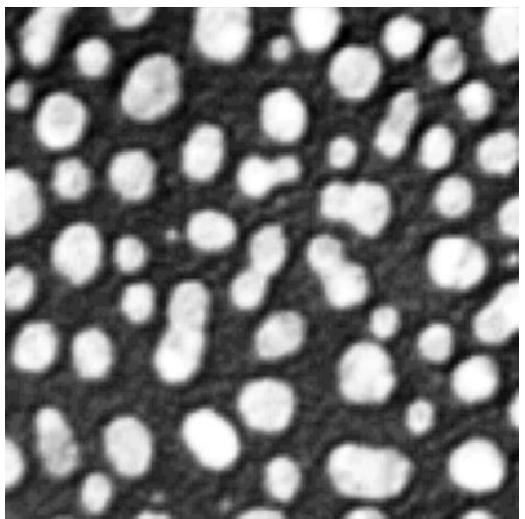
Watershed

- 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!



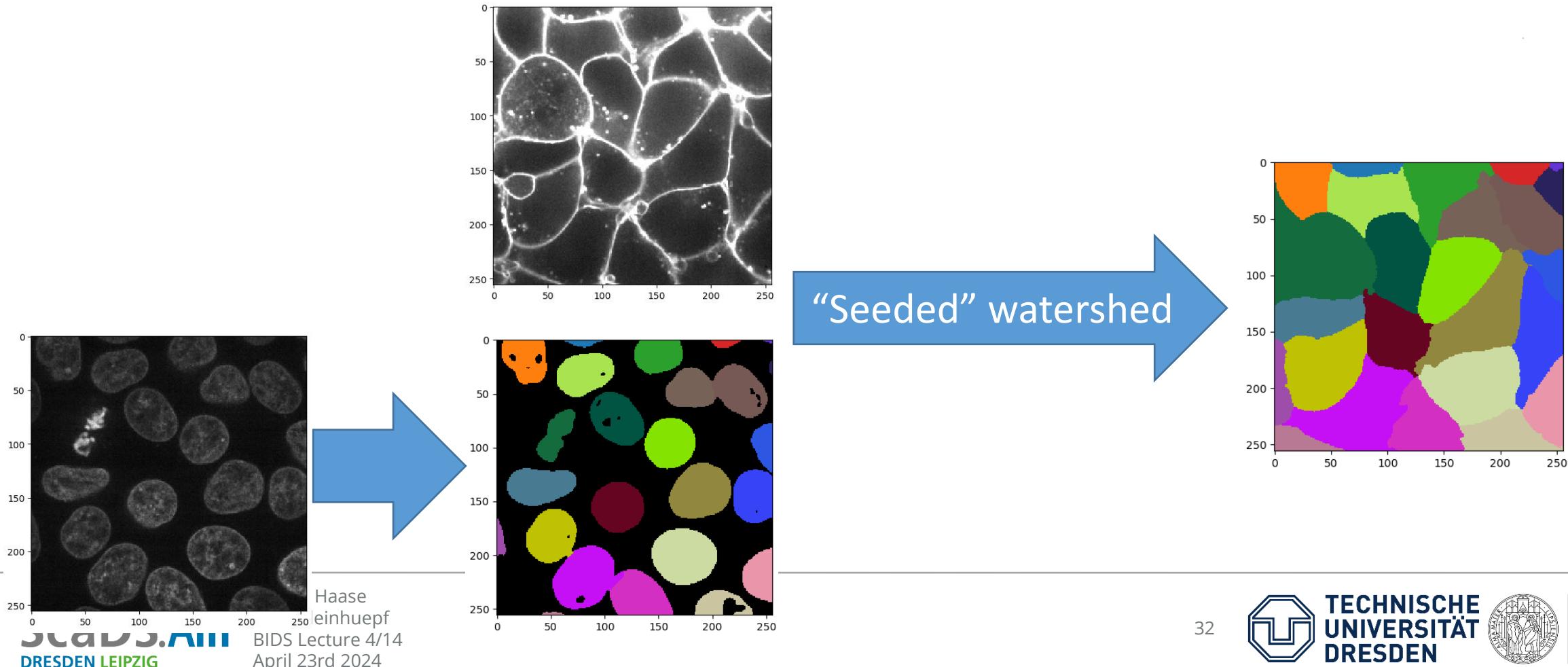
Watershed use-cases

- Split dense objects



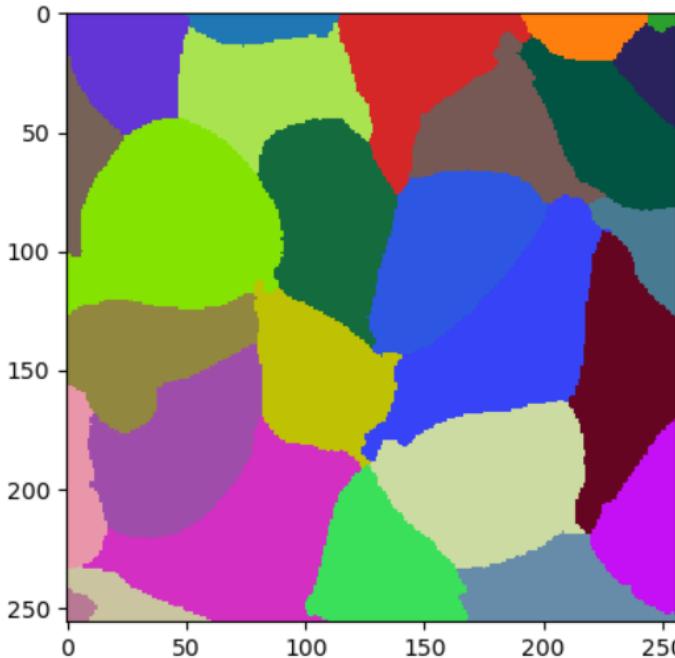
Watershed use-cases

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

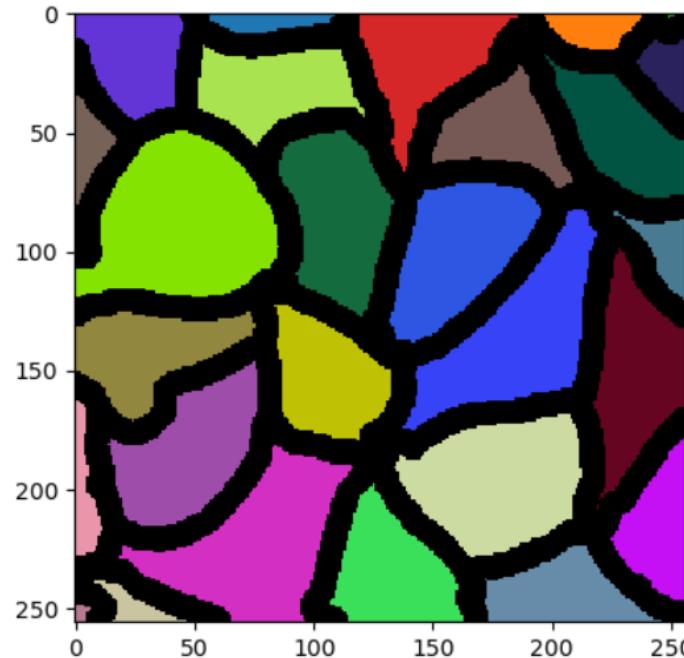


Label post-processing / morphological operations

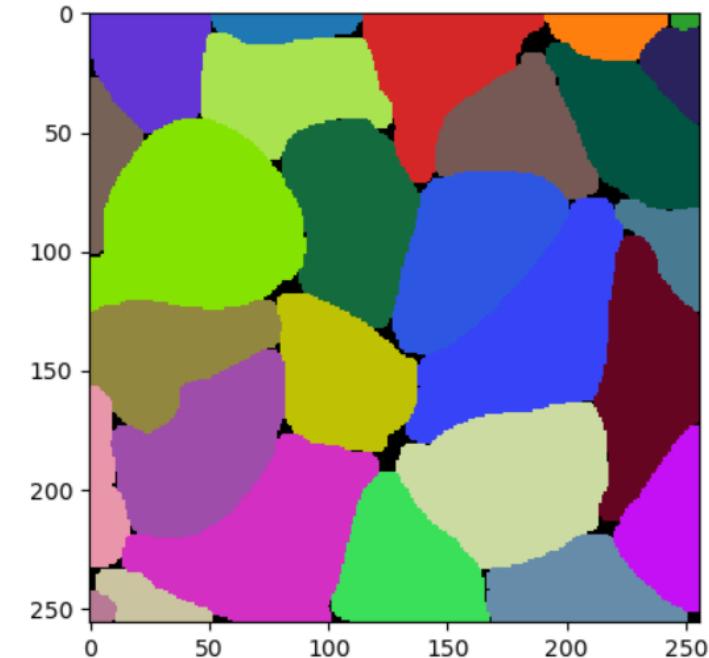
- ... similar to morphological operations on binary images



Original



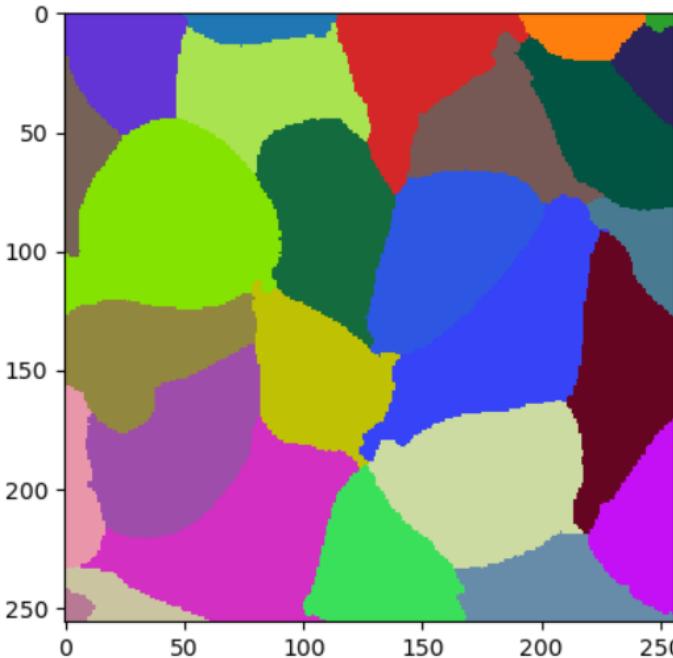
Eroding labels



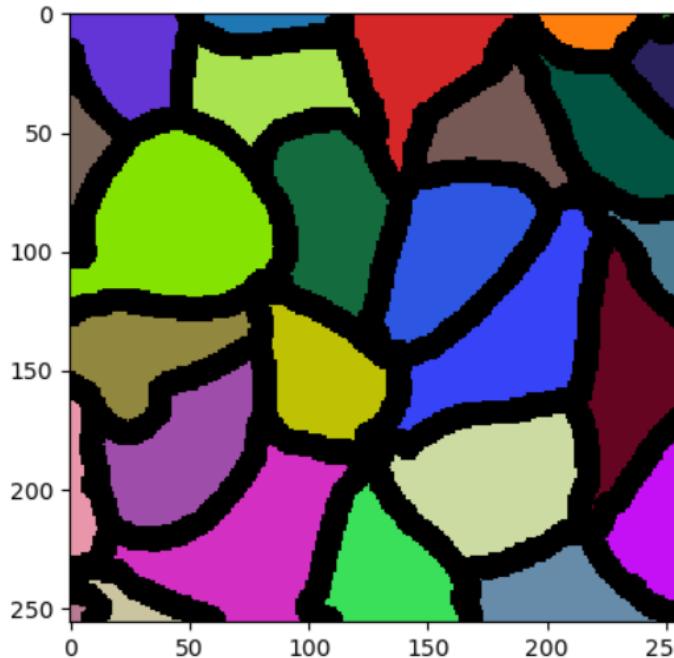
Dilating Labels

Label post-processing / morphological operations

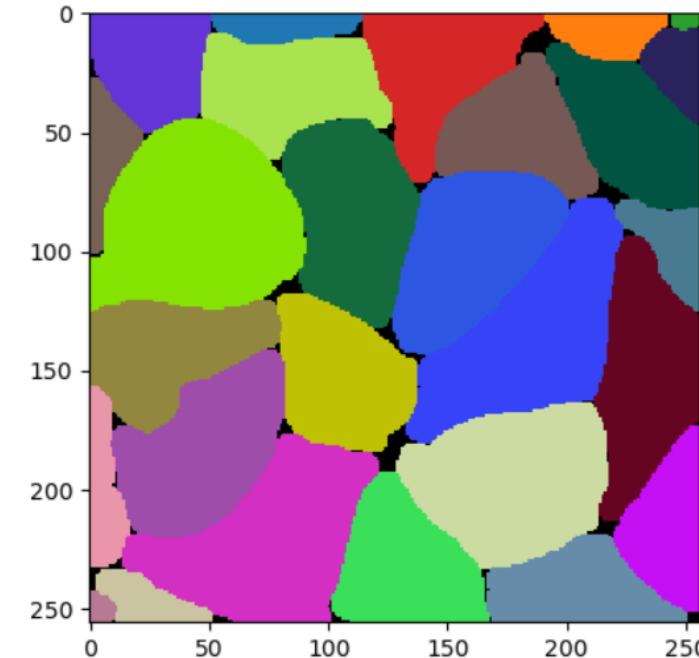
- ... similar to morphological operations on binary images



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



Original



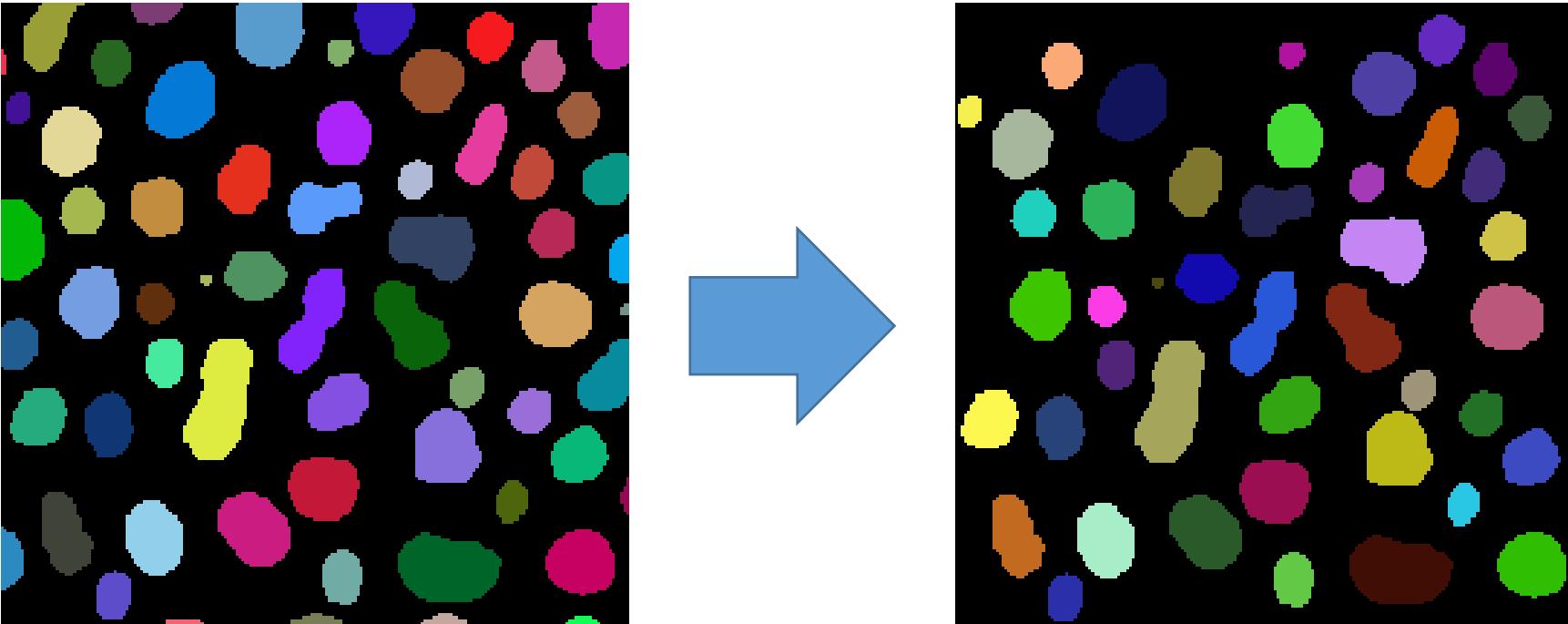
Opening labels



Smoothing Labels

Label post-processing / selections

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

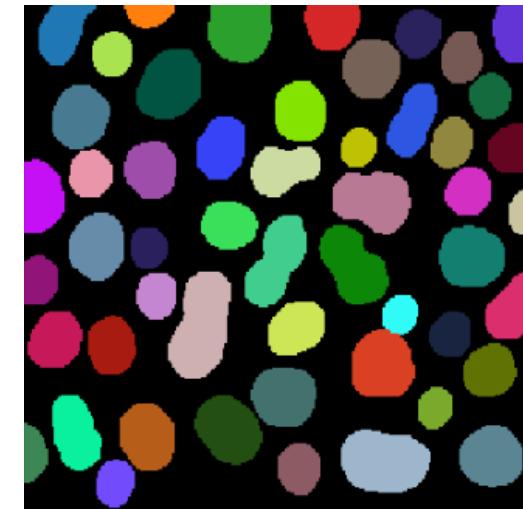


Label post-processing / selections

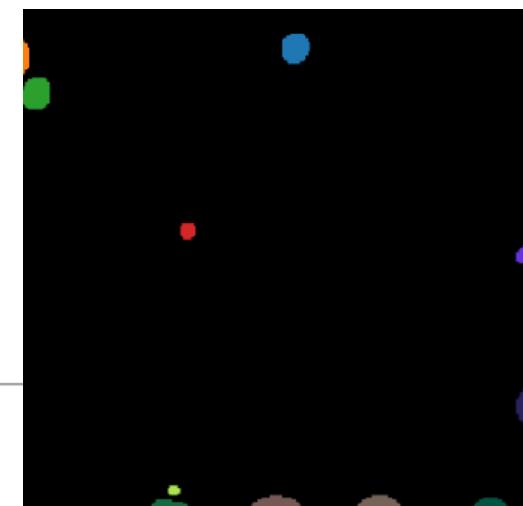
- Excluding small / large objects
- Common correction-step in case segmentations contain noise-related small particles



Exclude small objects

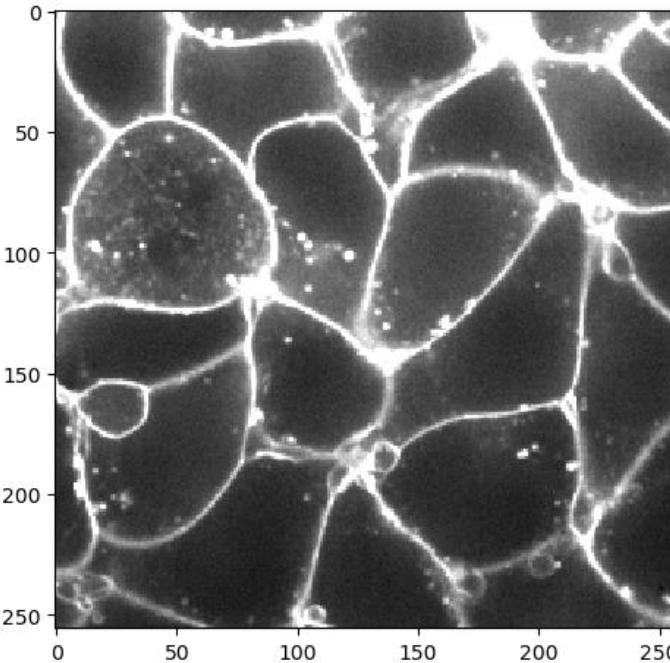


Exclude large objects



Quiz

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



Thresholding Watershed



Machine-
Learning



CENTER FOR SCALABLE DATA ANALYTICS AND
ARTIFICIAL INTELLIGENCE

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.

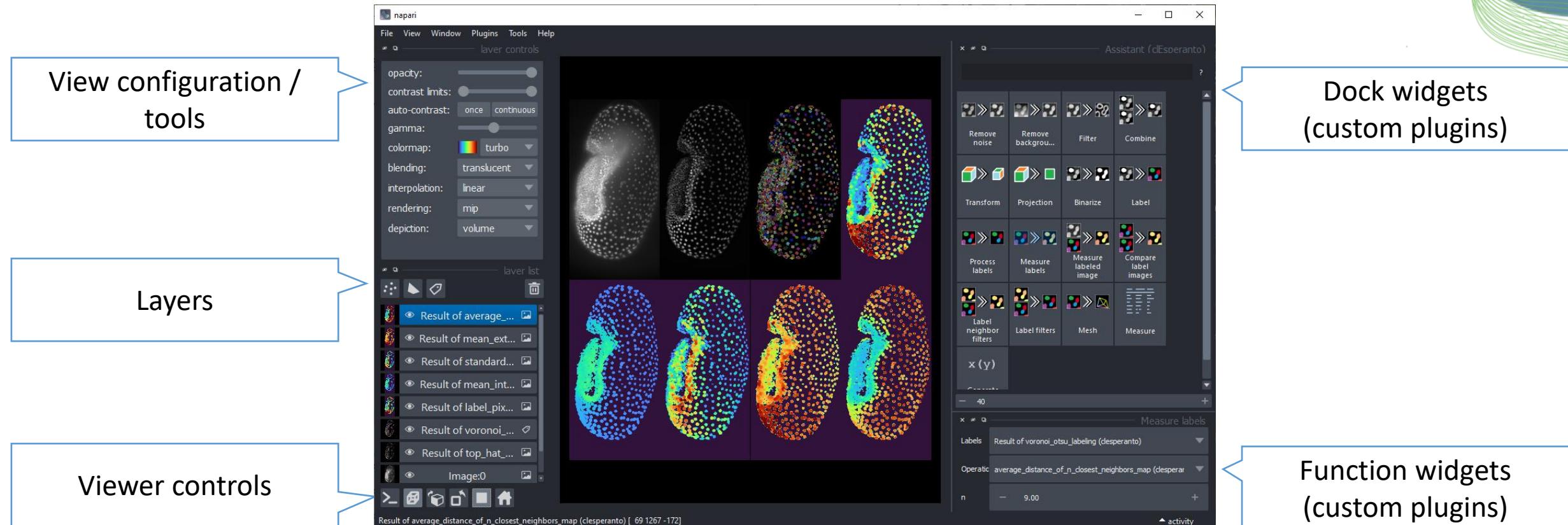
Chan
Zuckerberg
Initiative

Napari

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

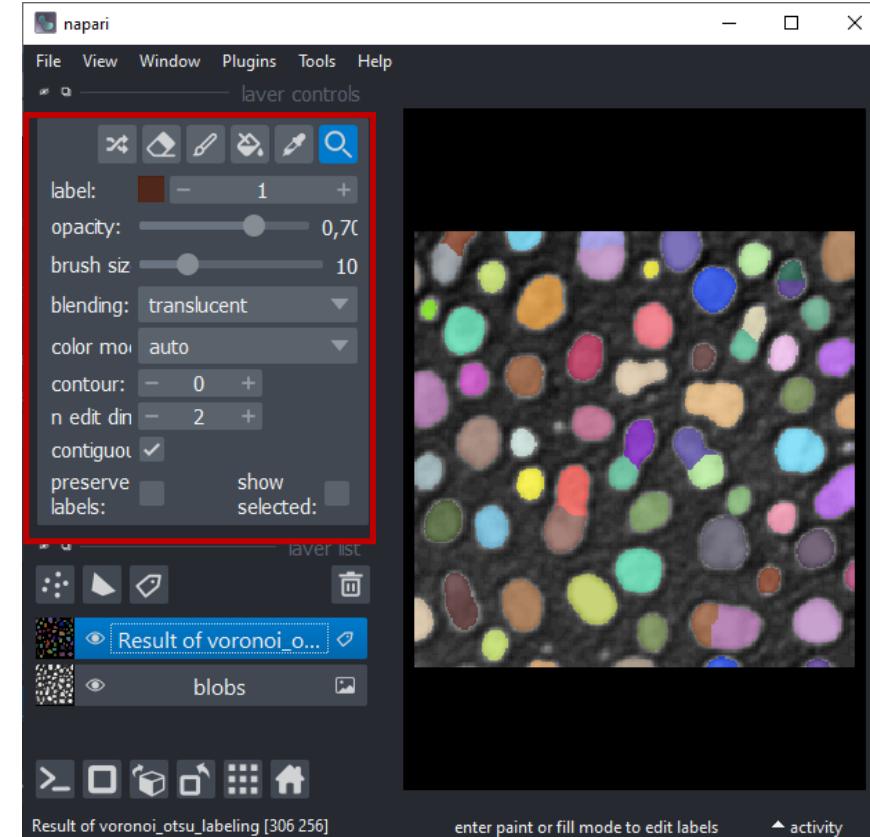
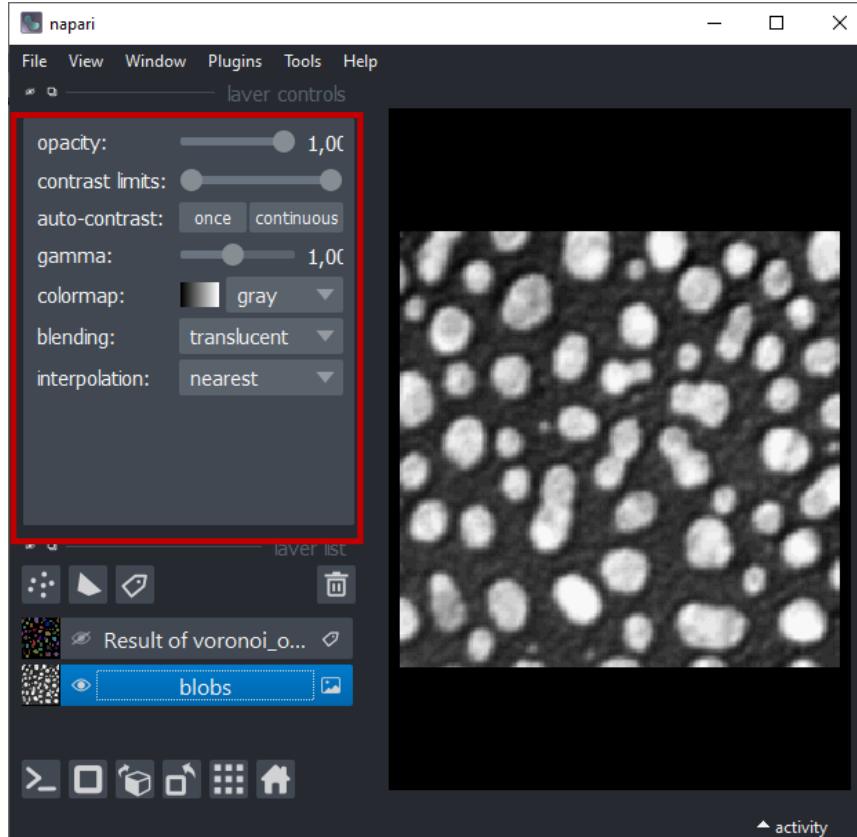


Napari – Graphical User Interface



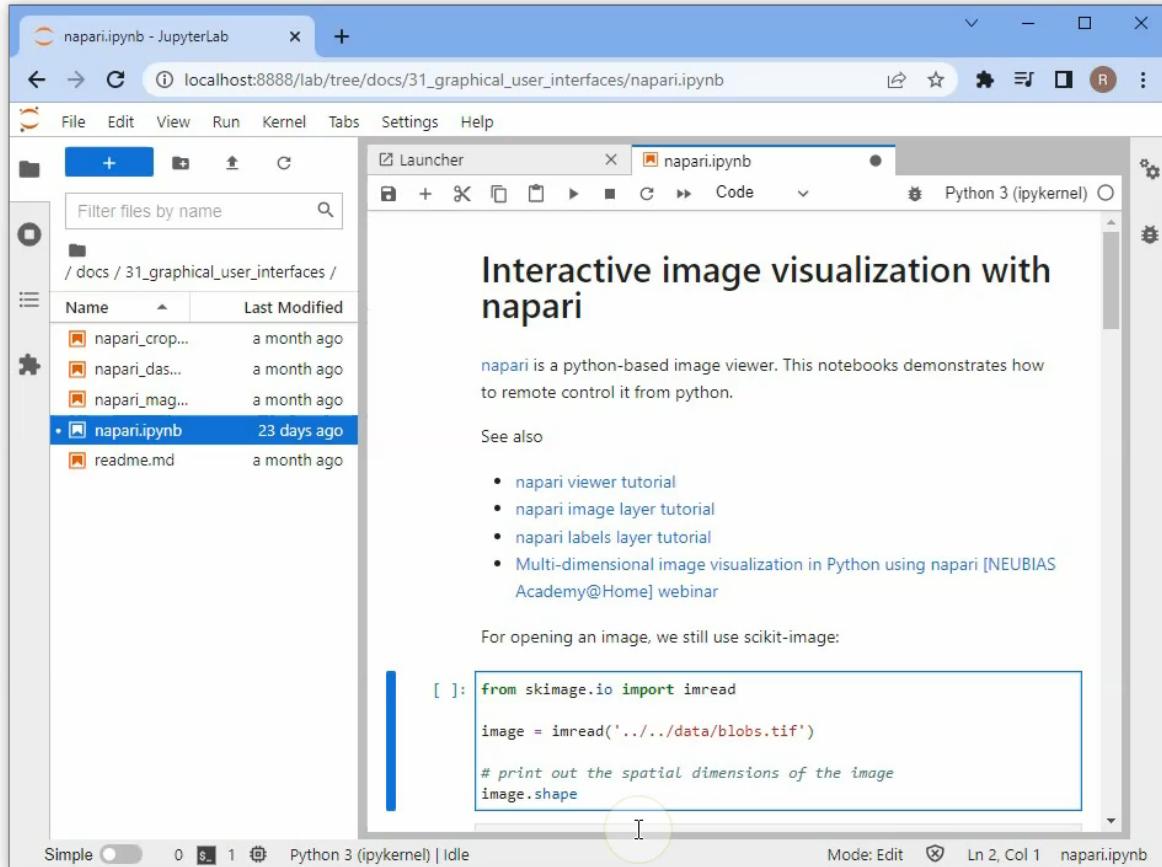
Napari – Graphical User Interface

- Context / data type dependent tools



Napari – Python Scripting

- Mixing interactivity and reproducibility



Napari – Python Scripting

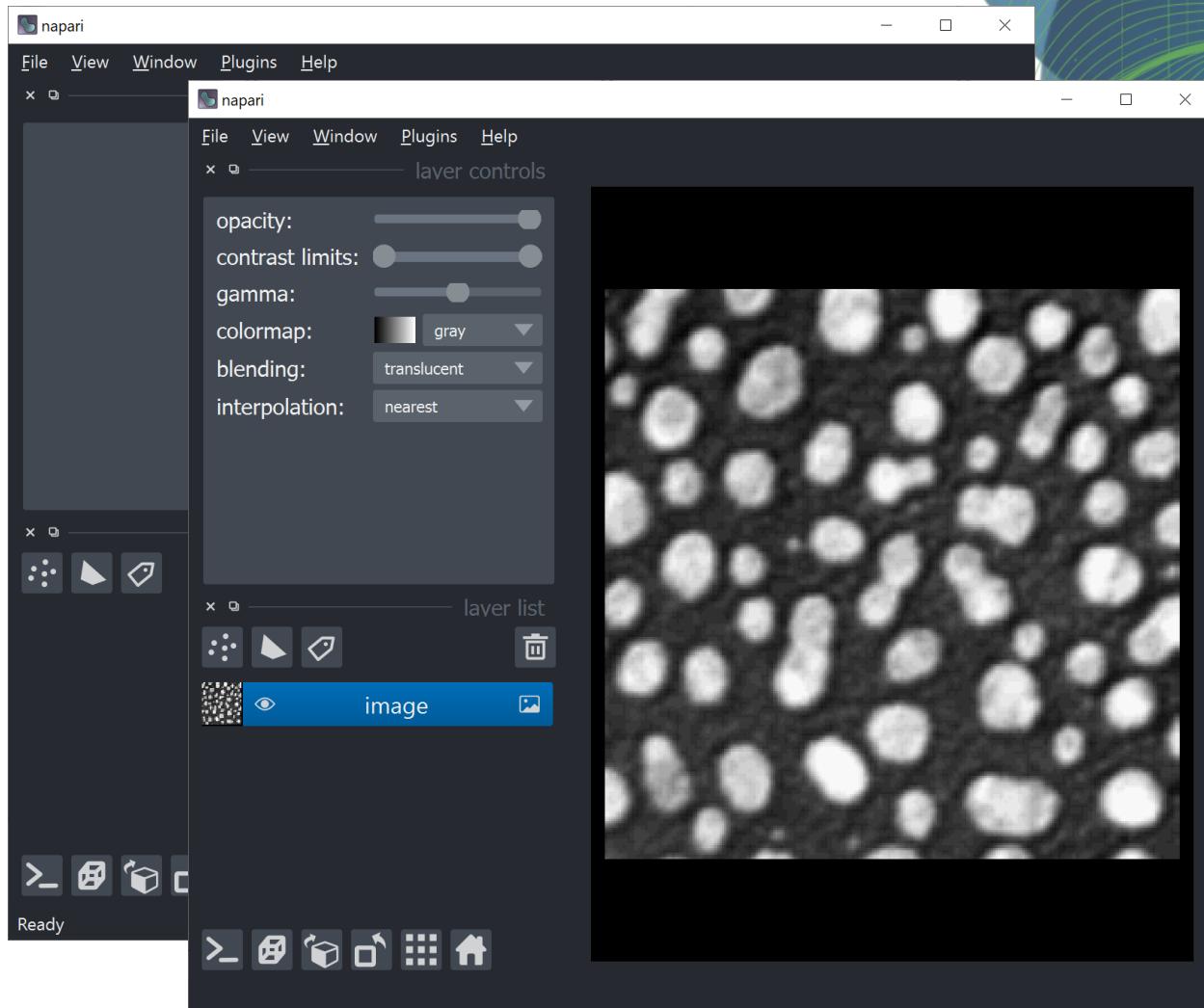
- Initialization

```
import napari
```

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

- Adding images

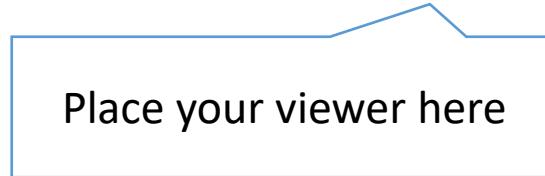
```
viewer.add_image(image)
```



Napari – Python Scripting

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

`napari.utils.nbscreenshot(viewer)`



The screenshot shows a Jupyter Notebook interface with the title "01_napari - Jupyter Notebook". The code cell In [1] contains:

```
import napari  
  
# Create an empty viewer  
viewer = napari.Viewer()  
  
# Start it  
napari.run()
```

The code cell In [3] contains:

```
# Add a new Layer containing an image  
viewer.add_image(image)
```

The output cell Out[3] shows:

```
<Image layer 'image' at 0x1a72ea05580>
```

To the right of the notebook, a small napari viewer window is displayed with a grayscale image of cells and various adjustment sliders for opacity, contrast limits, gamma, colormap, blending, and interpolation.

Napari – Python Scripting

- Removing layers

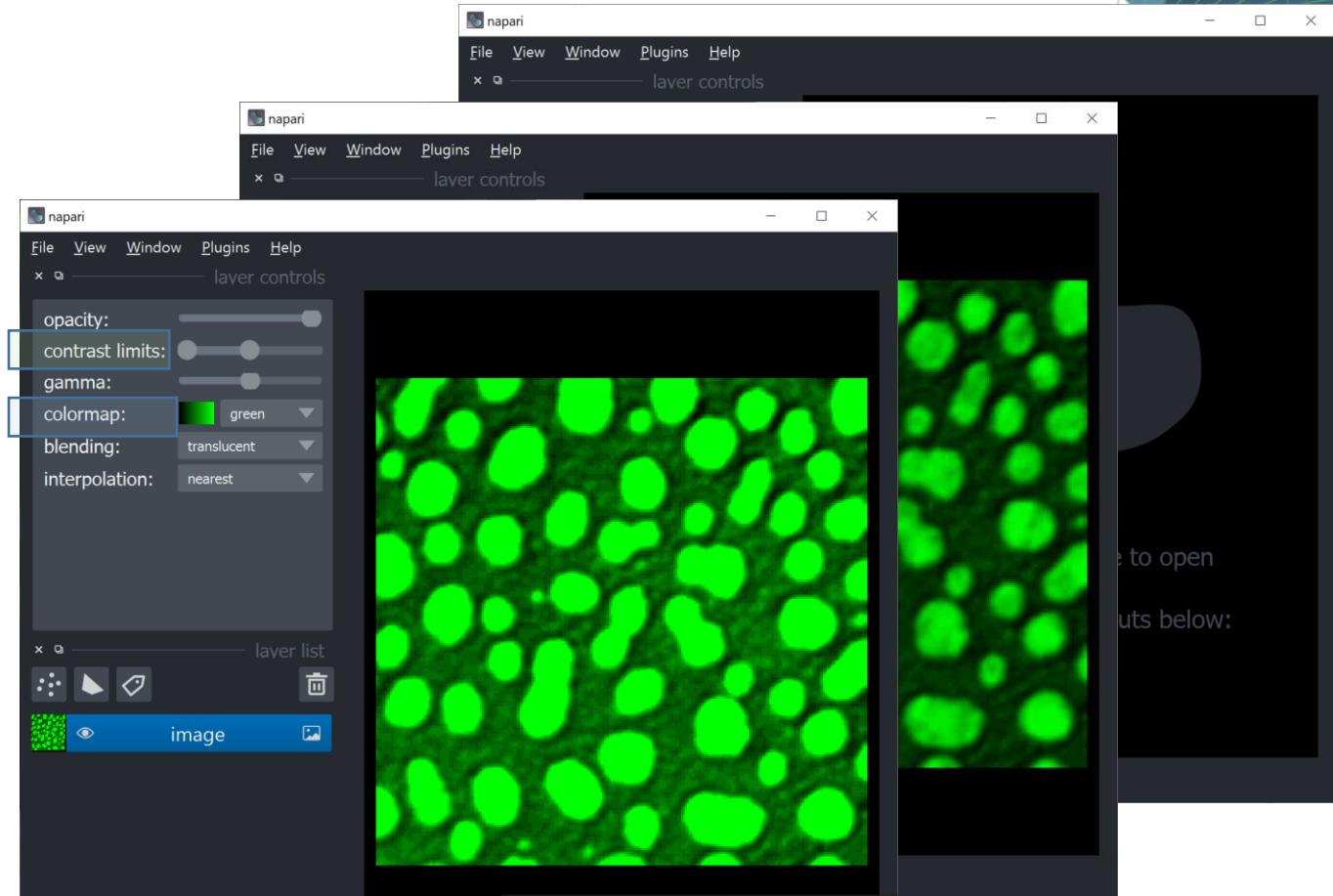
```
for l in viewer.layers:  
    viewer.layers.remove(l)
```

- Modify visualization while adding layers

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

- Modify layers after adding

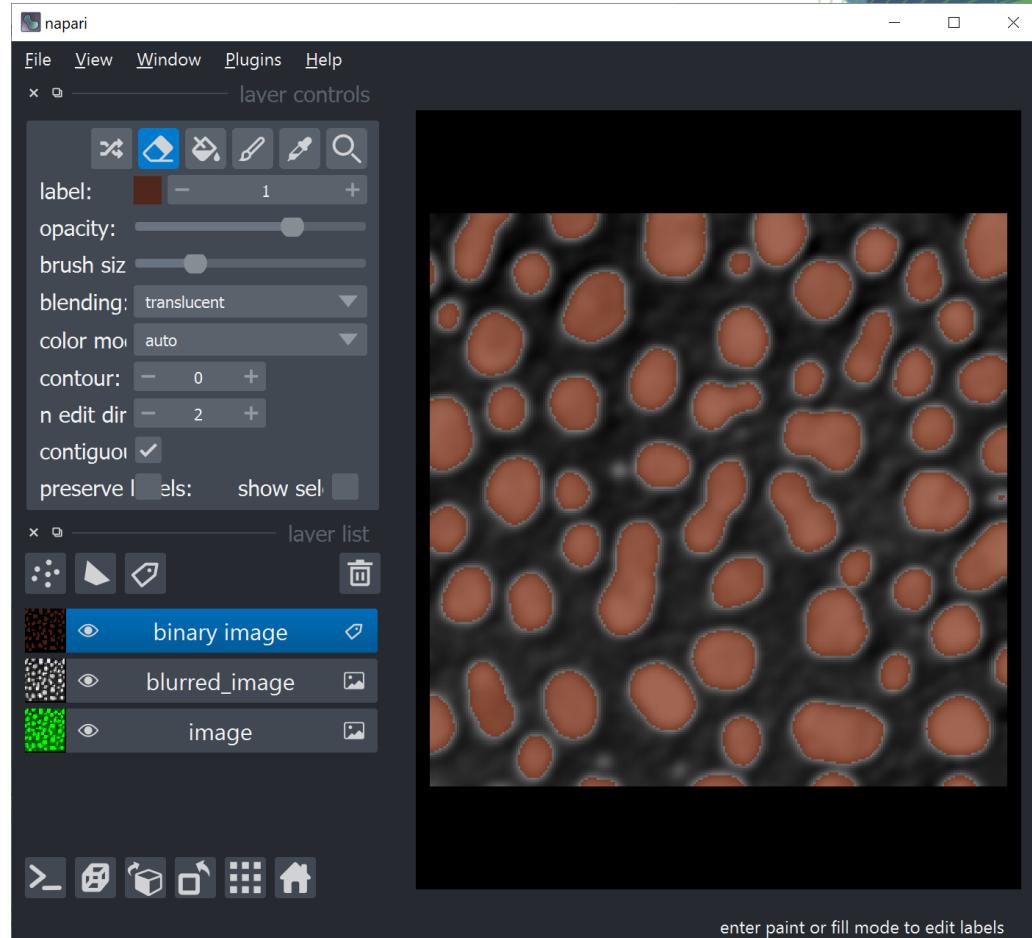
```
layer = viewer.add_image(image)  
layer.colormap = 'green'  
layer.contrast_limits = (0, 128)
```



Napari – Python Scripting

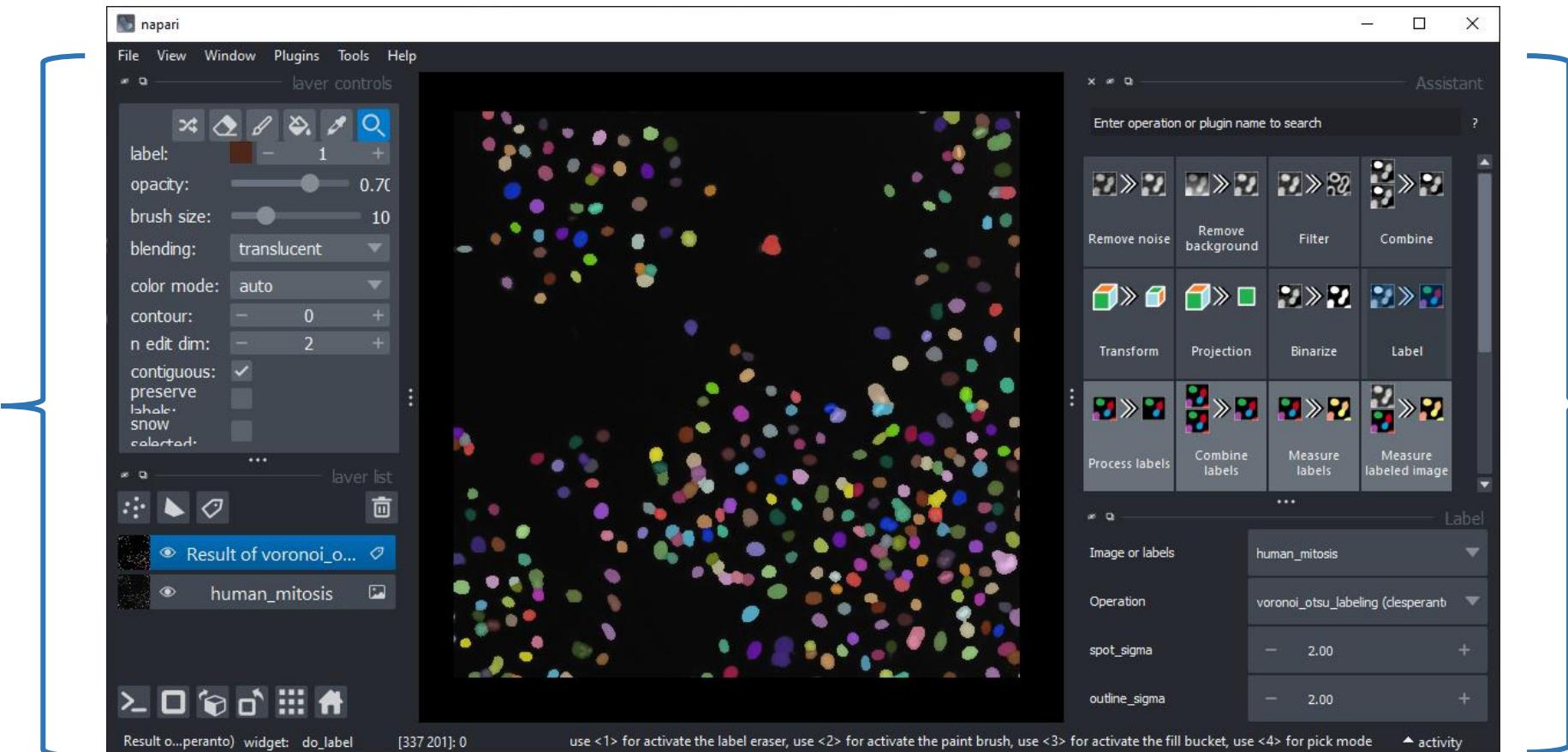
- Binary images and **label** images visualized as label layers

```
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)
```



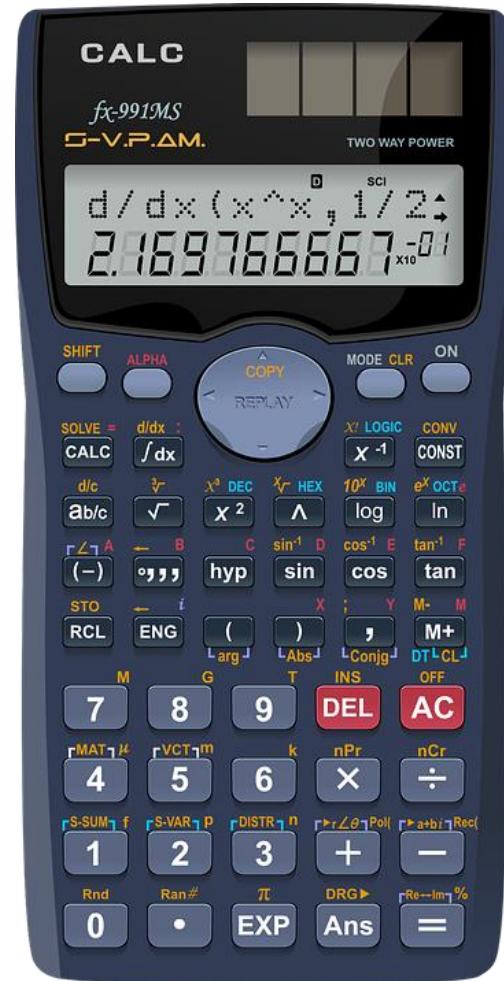
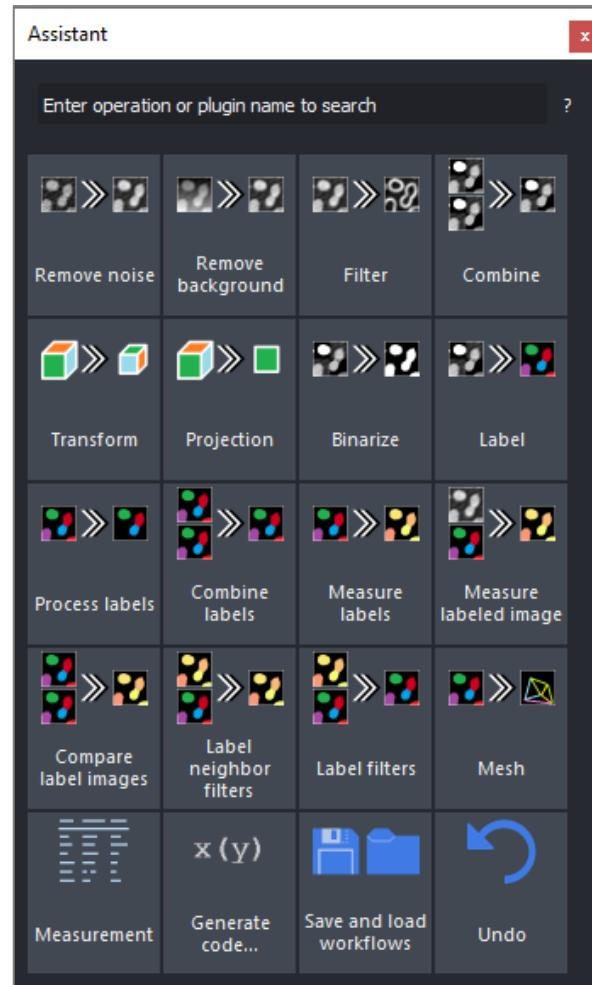
The Napari Assistant

- Tools > Utilities > Assistant (na)



The Napari Assistant

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



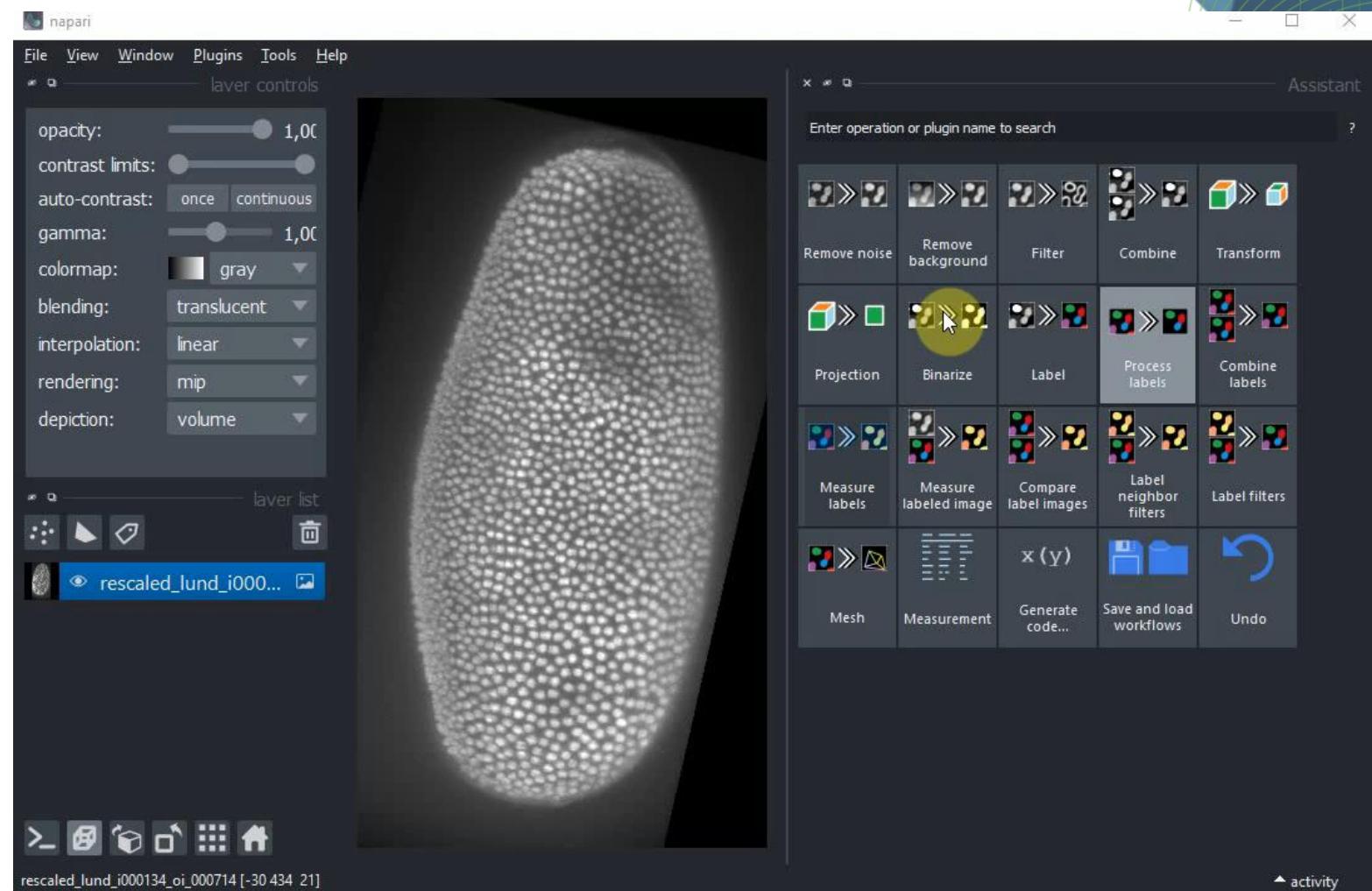
The Napari Assistant

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

Big thanks to:

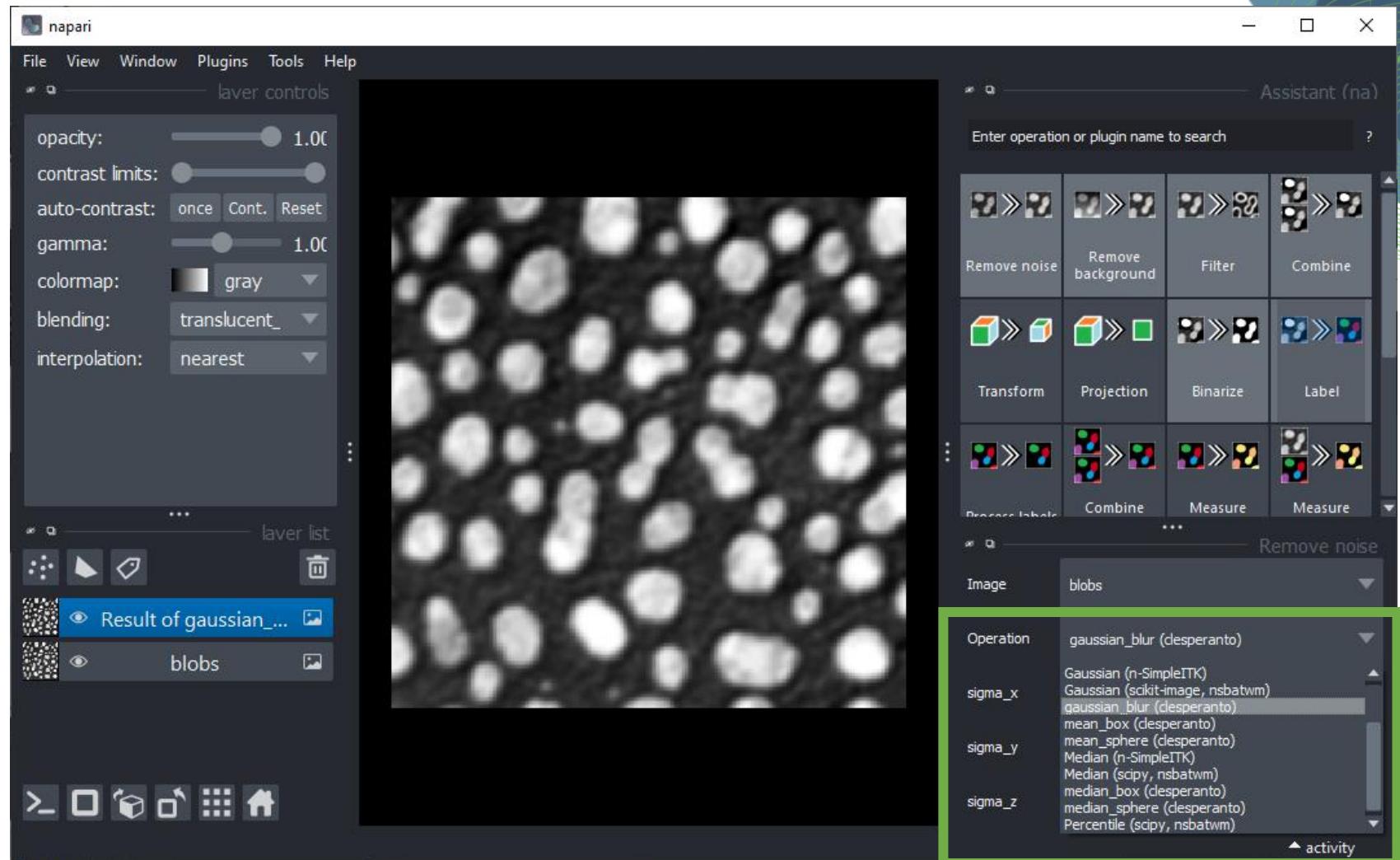


Ryan Savill
@RyanSavill4



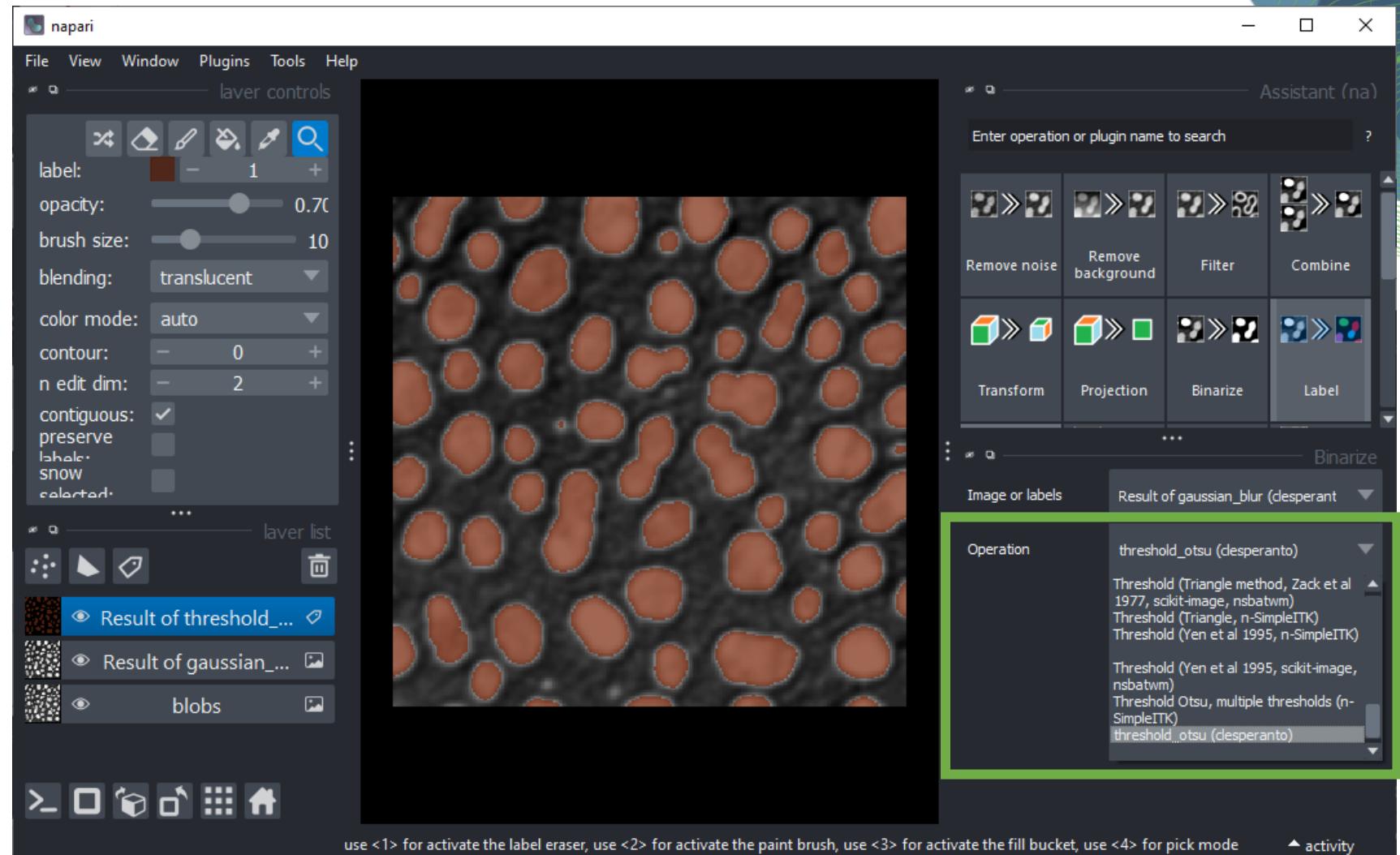
Workflow building

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



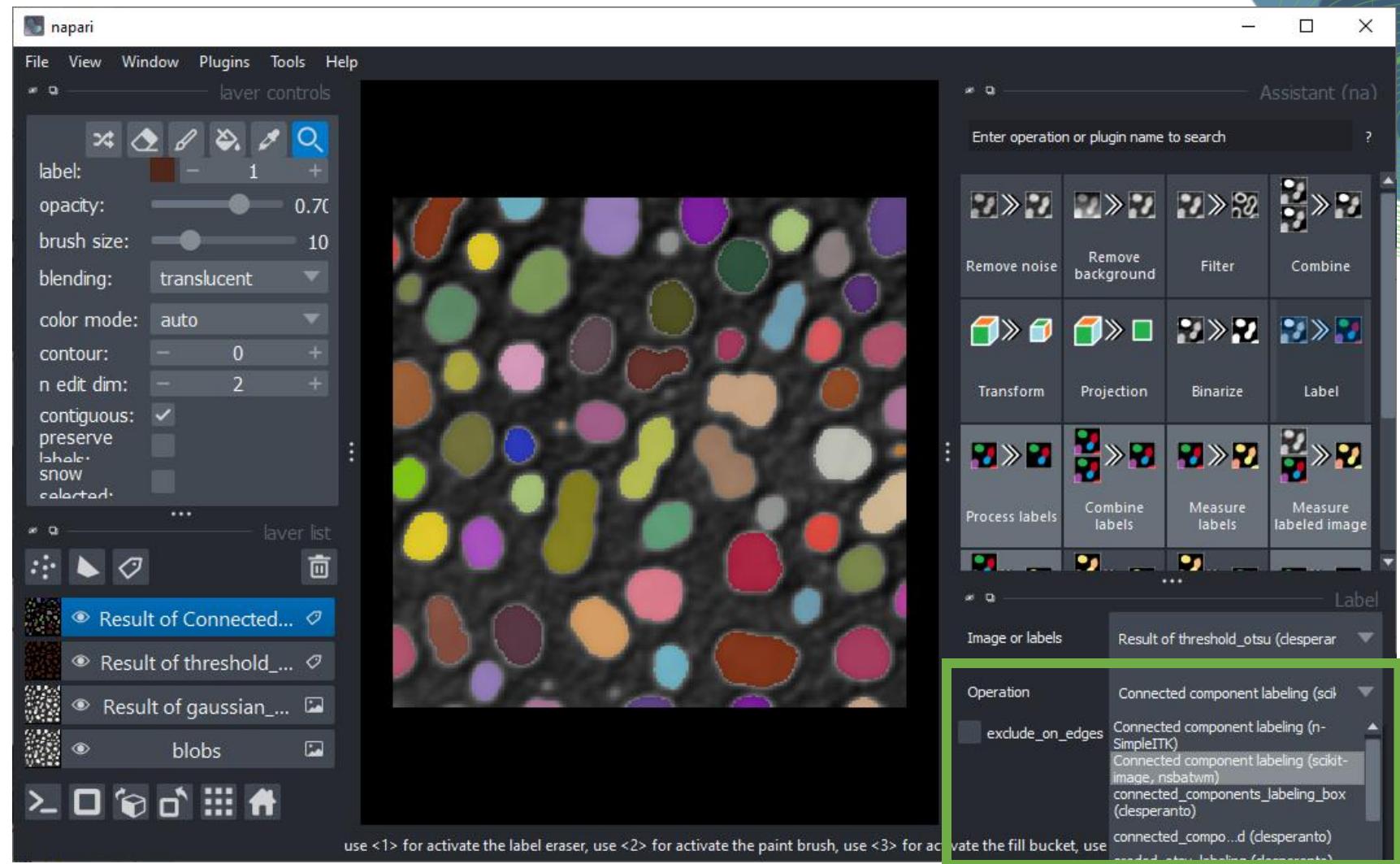
Workflow building

- Try different binarization algorithms



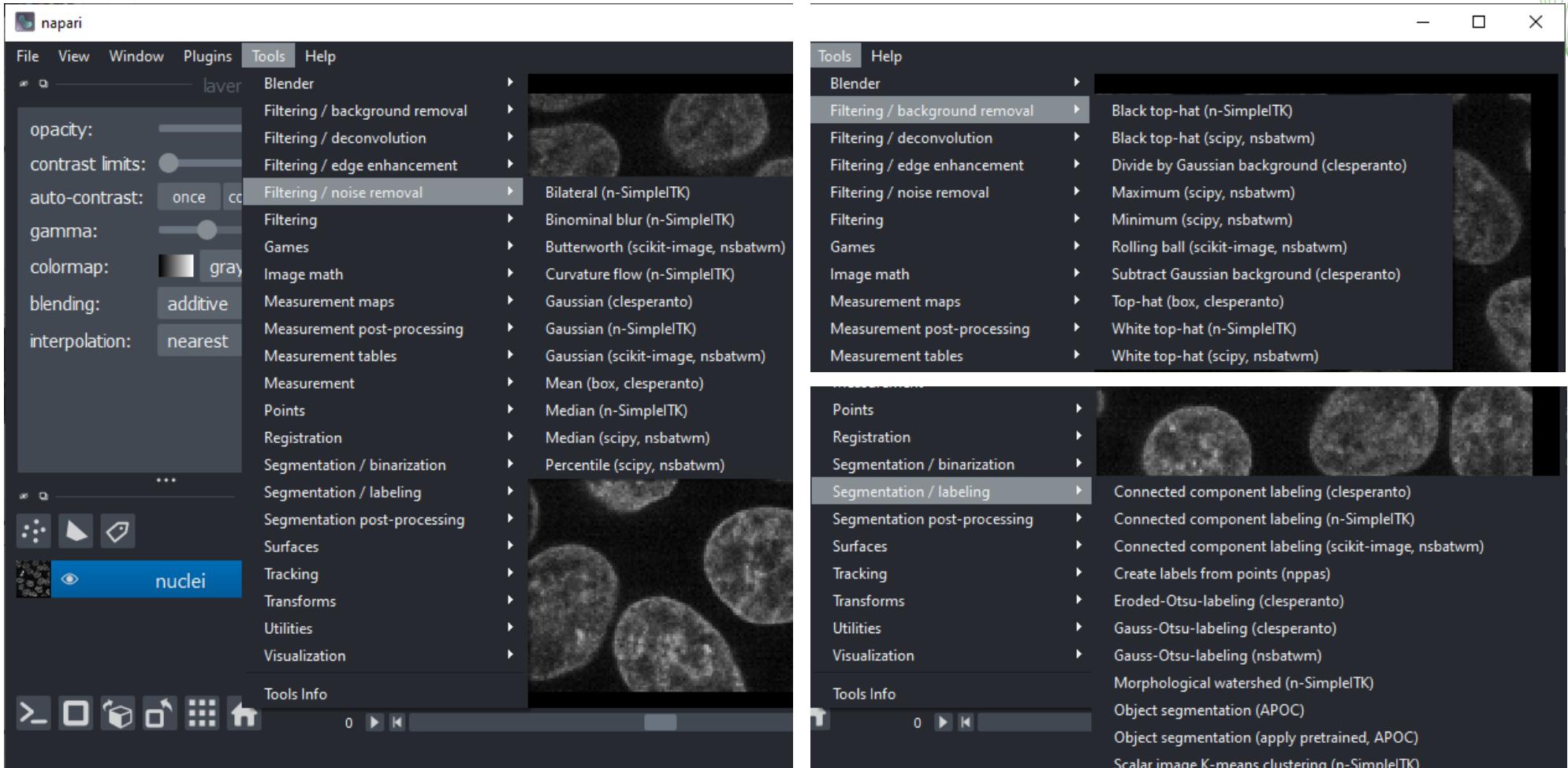
Workflow building

- Try different labeling algorithms



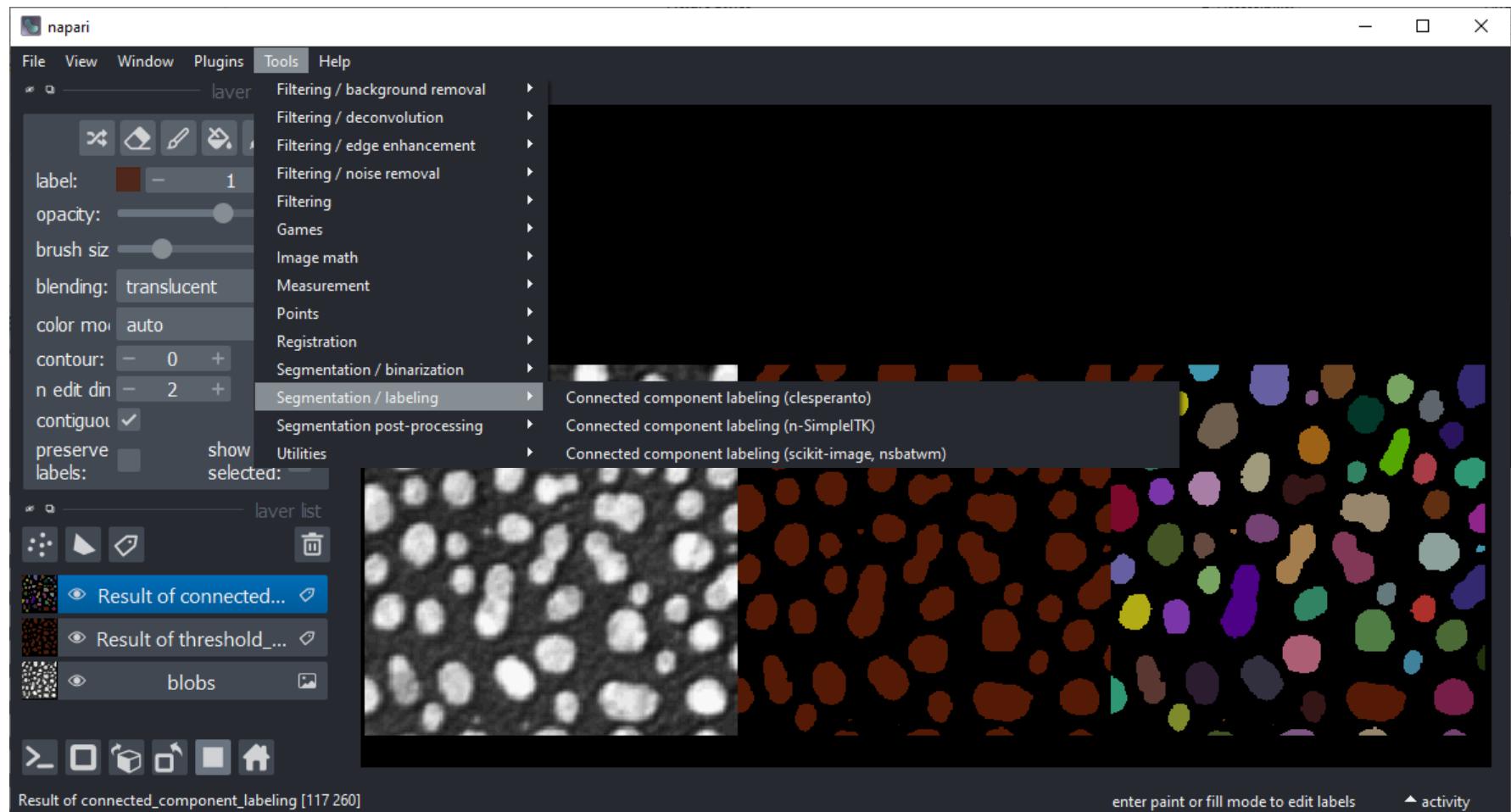
The Tools menu

- Organized in categories



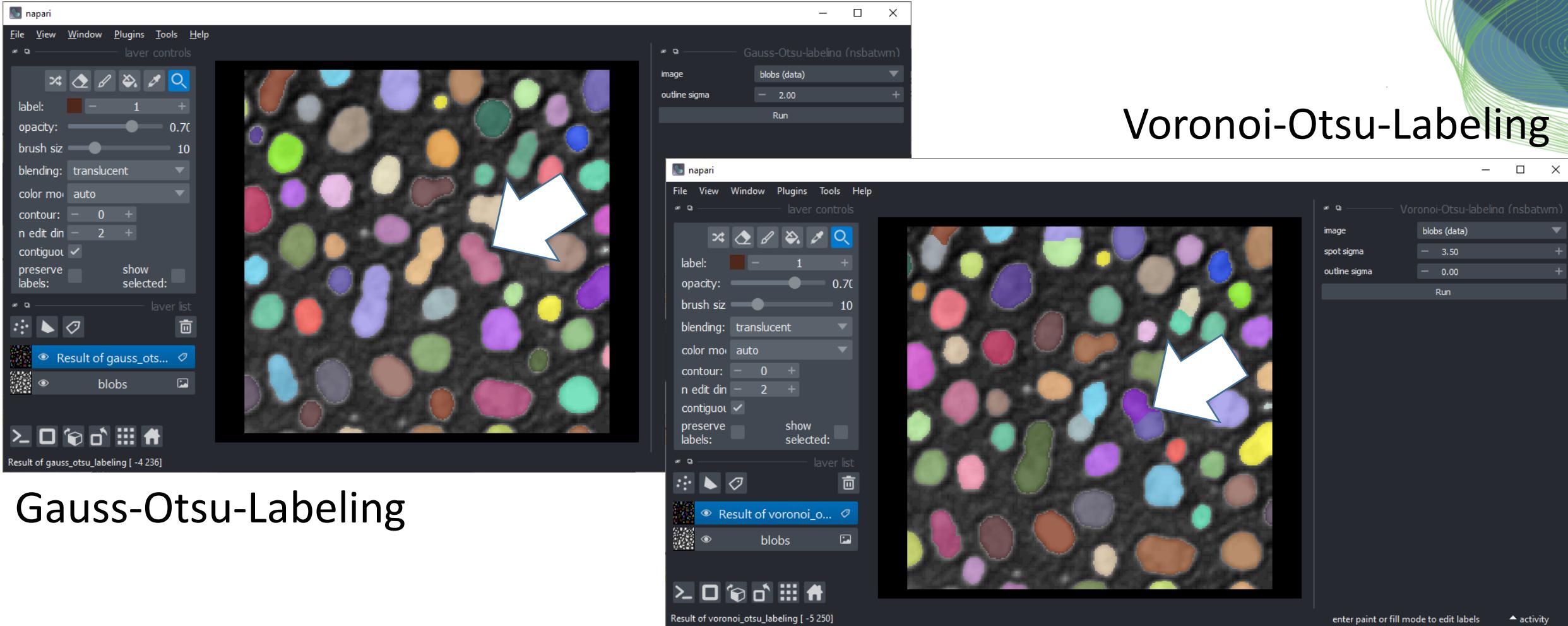
Workflow building

Also check out the Tools > Segmentation / labeling menu



Short-cuts: Voronoi-Otsu-Labeling

Also check out the Tools > Segmentation / labeling menu

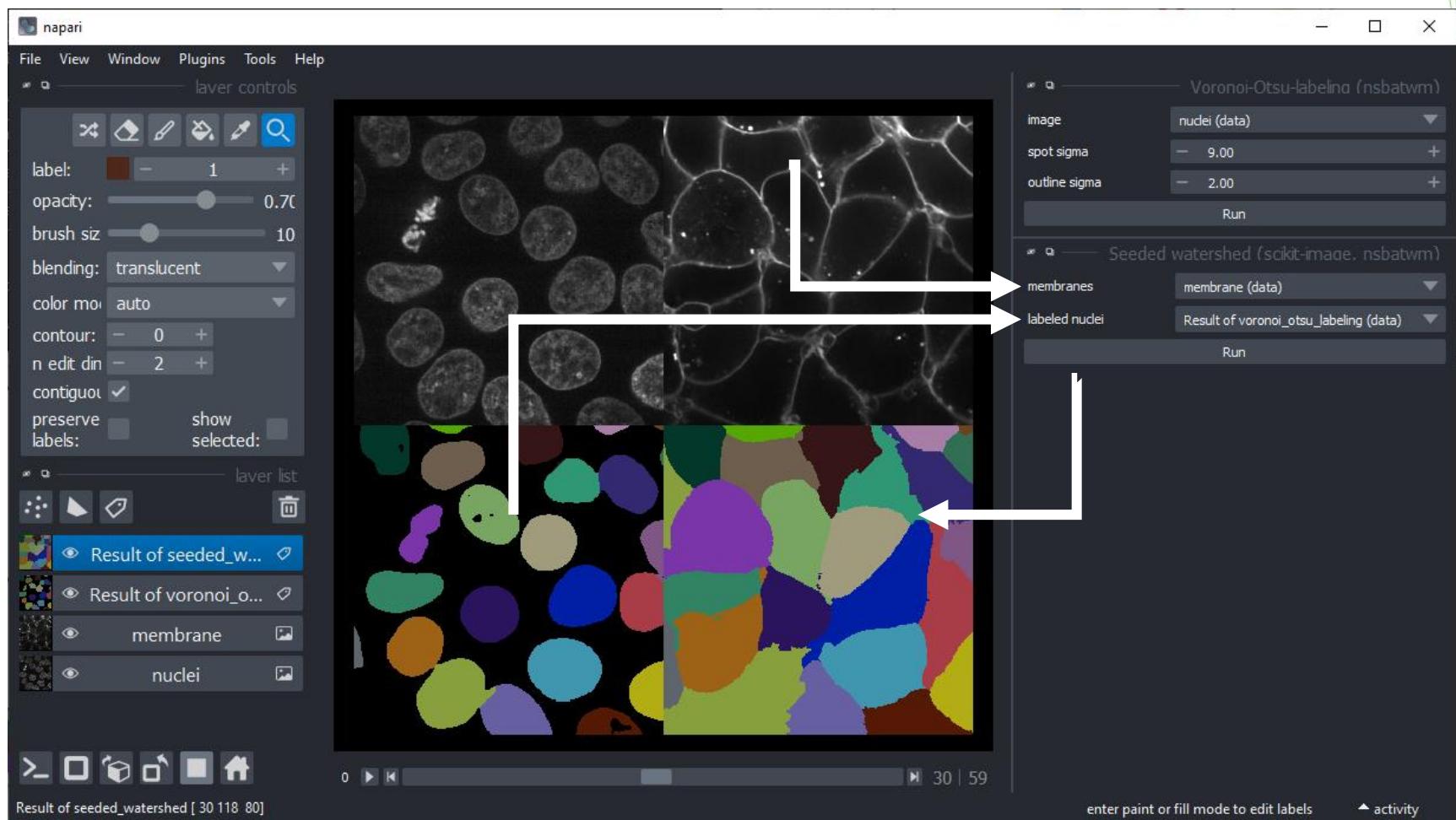


Gauss-Otsu-Labeling

Voronoi-Otsu-Labeling

Watershed

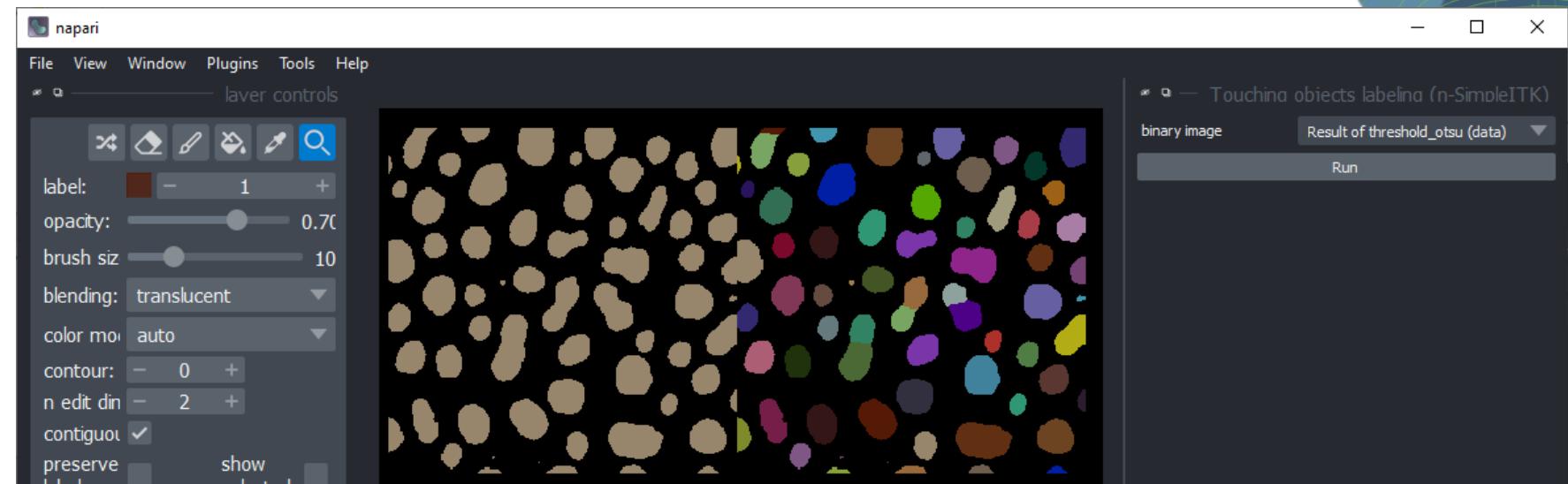
Also check out the Tools > Segmentation / labeling menu



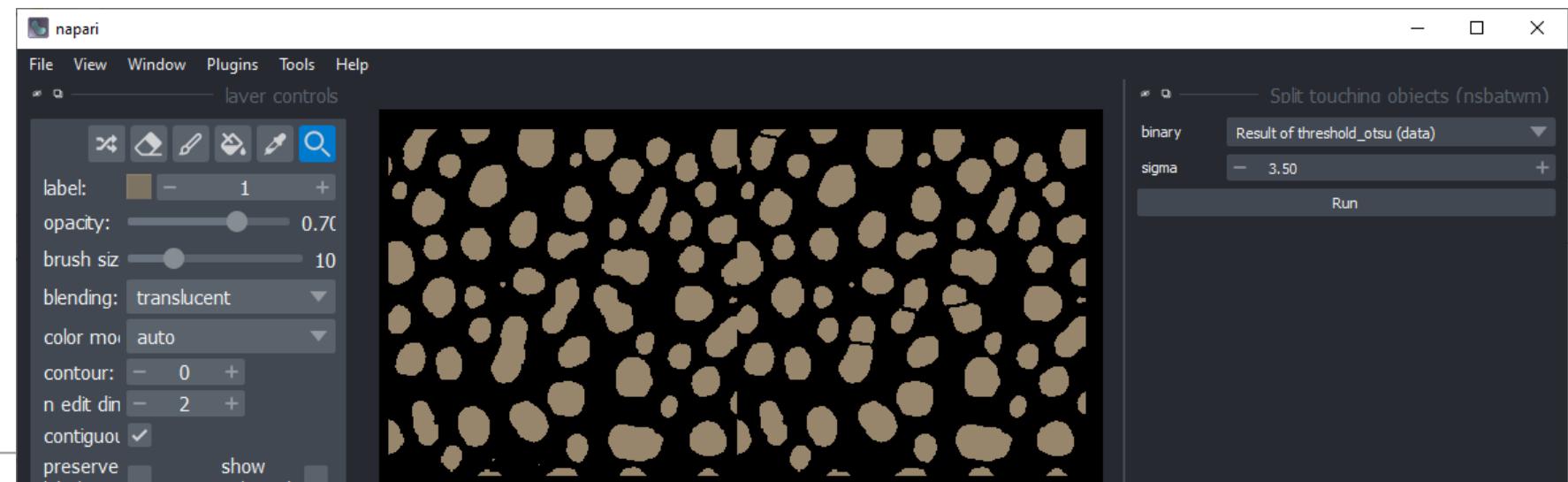
Watershed

- From binary images

Tools > Segmentation / labeling >
Label touching objects

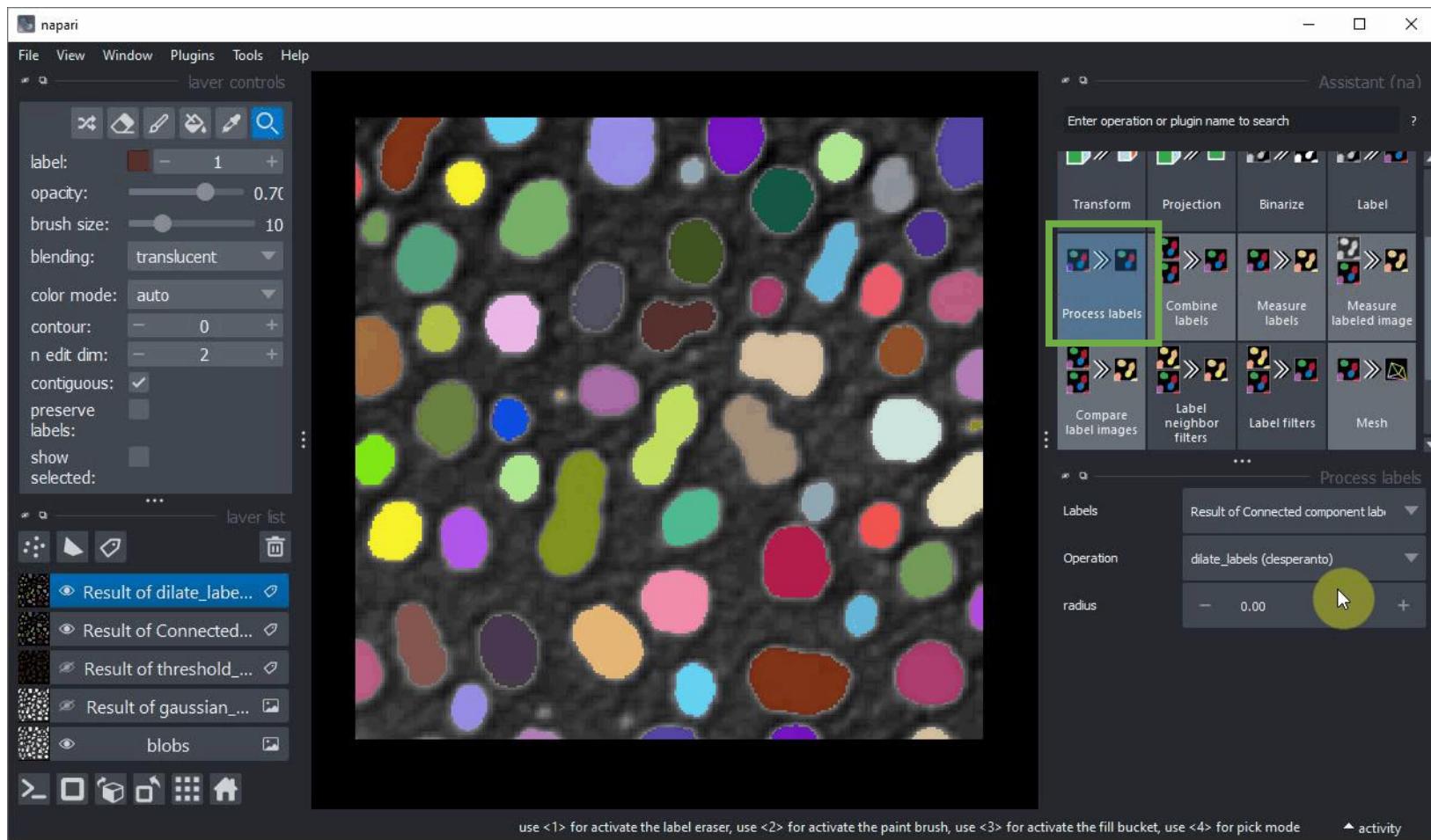


Tools > Segmentation post-processing >
Split touching objects
(Similar to ImageJ's Watershed)



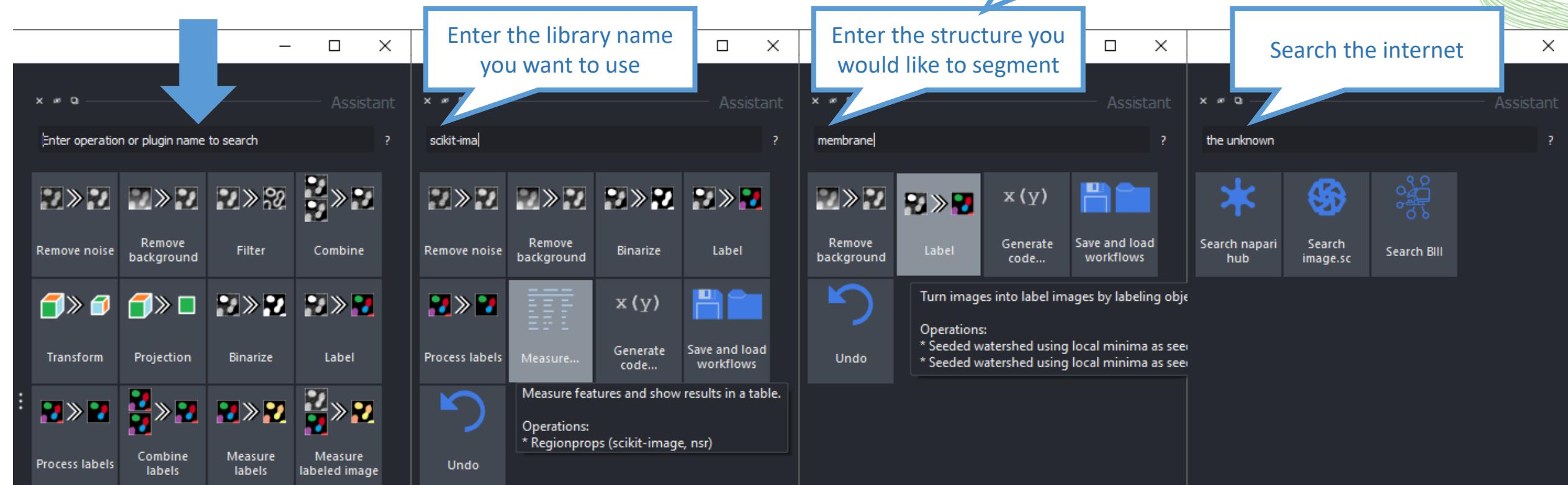
Label erosion, dilation, opening, closing, ...

- In Napari Assistant: Process labels

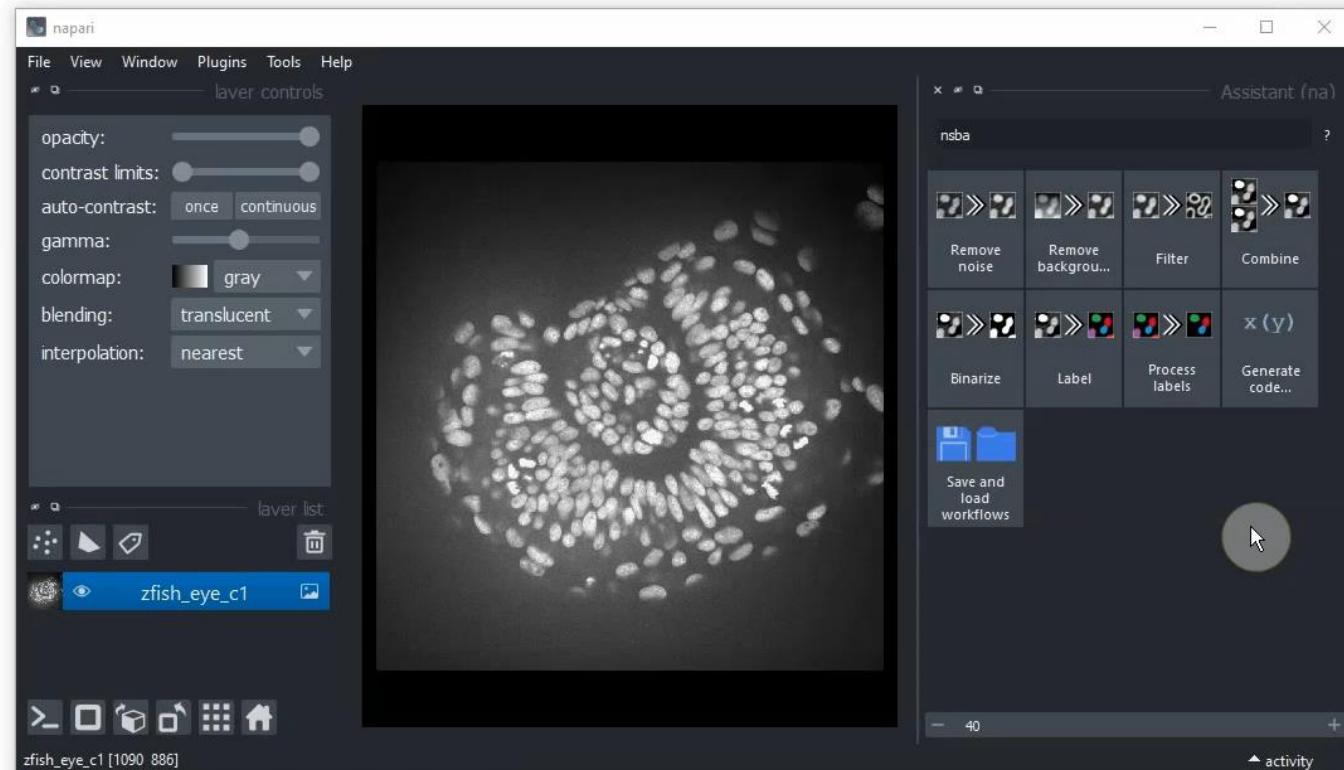


Browse operations

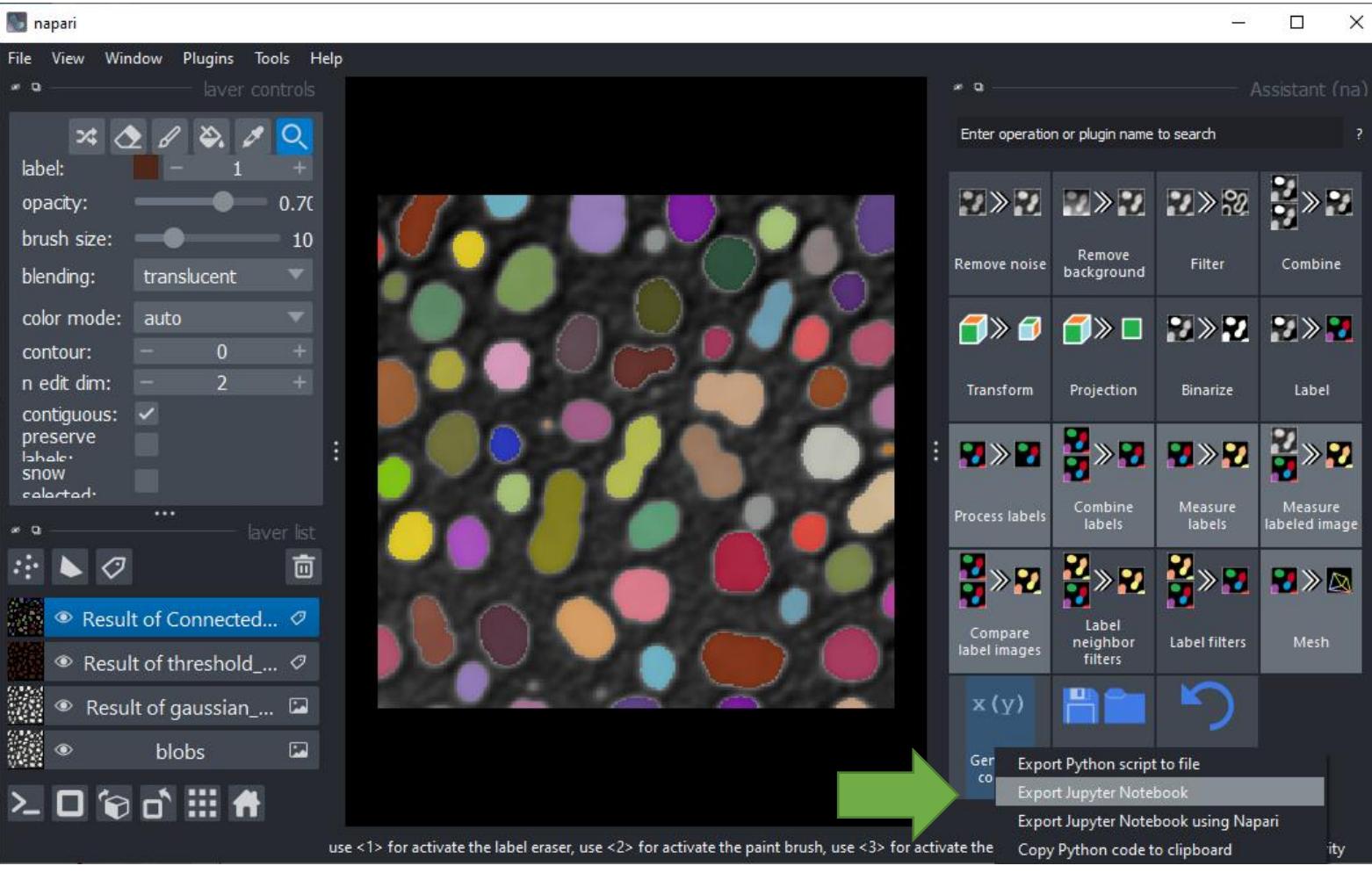
- Use the search...



Export code to Jupyter Notebooks



Export code to Jupyter Notebooks



The screenshot shows the napari Assistant application interface. On the left, there is a segmented image of various colored blobs on a black background. On the right, there is a grid of operation icons. A green arrow points from the bottom right towards the export options at the bottom of the interface. The export options include:

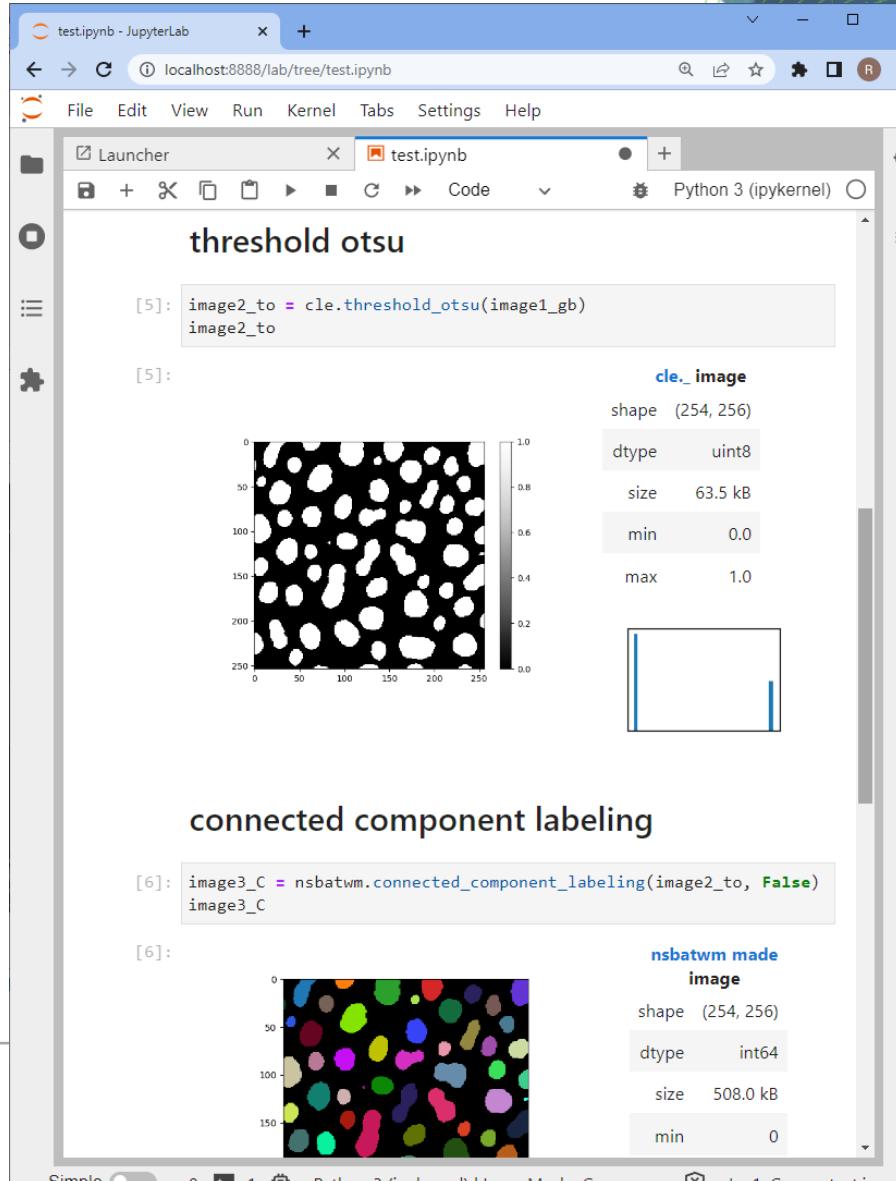
- Export Python script to file
- Export Jupyter Notebook** (highlighted)
- Export Jupyter Notebook using Napari
- Copy Python code to clipboard

On the right side of the image, there is a Jupyter Notebook window titled "test.ipynb - JupyterLab". It contains the following code and output:

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

Output:

```
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
```

Output:

```
nsbatwm made
image
shape (254, 256)
dtype int64
size 508.0 kB
min 0
```

At the bottom of the slide, there is a footer with the ScaDS.AI logo and some text:

Robert Haase
@haesleinhuepf
BIDS Lecture 4/14
April 23rd 2024

<https://github.com/haesleinhuepf/napari-assistant>
#code-generation



DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS
AND ARTIFICIAL INTELLIGENCE

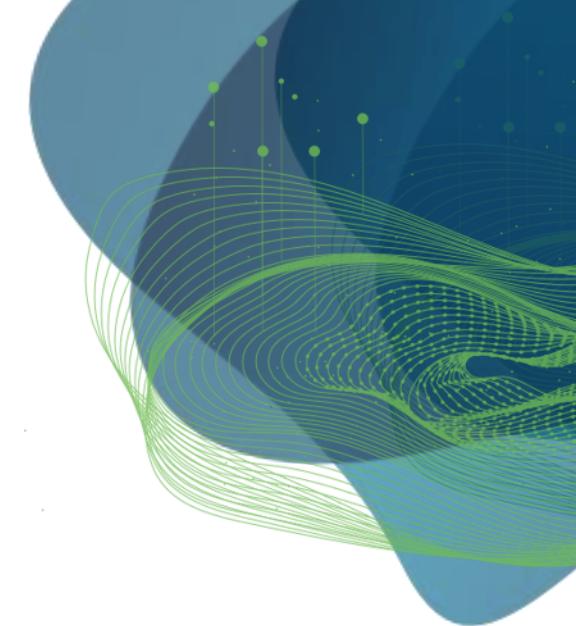


Image segmentation in Python

Robert Haase

GEFÖRDERT VOM

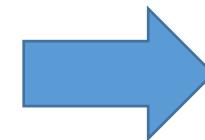


Bundesministerium
für Bildung
und Forschung

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

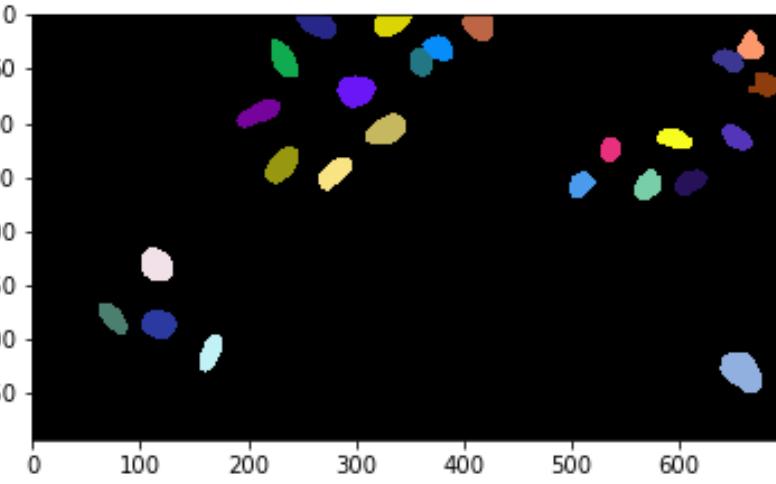
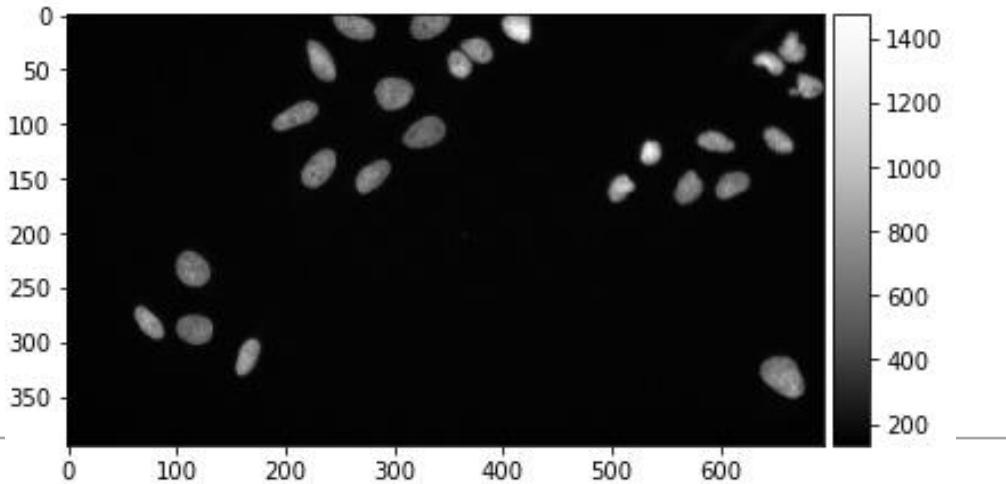
- Gaussian-Blur
- Otsu-Thresholding
- Spot-detection
- Watershed on the binary image



... 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

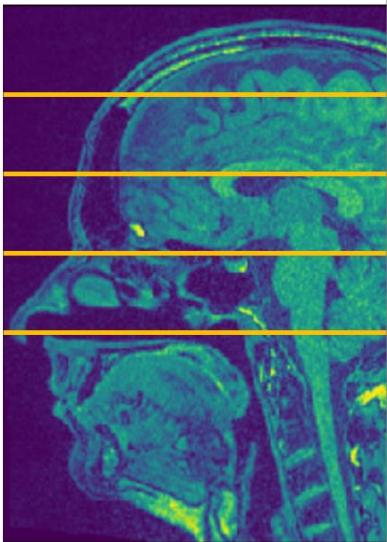
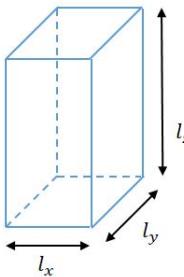
spot_sigma : float, optional

Anisotropy

- Reminder: Anisotropic images might be tricky to process properly

Image stacks and voxels

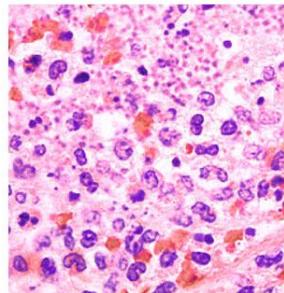
- 3-dimensional images consisting of voxels
- "Image stack"
- Often *anisotropic* (not equally large in all directions)



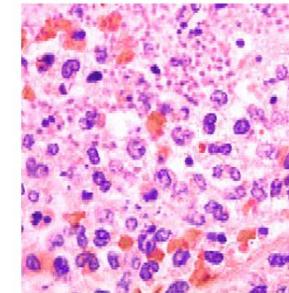
ScaDS.AI
DRESDEN LEIPZIG
 $l_x \neq l_y \neq l_z$
Robert Haase
@haesleinhuepf
BIDS Lecture 3/14
April 16th 2024

Anisotropy

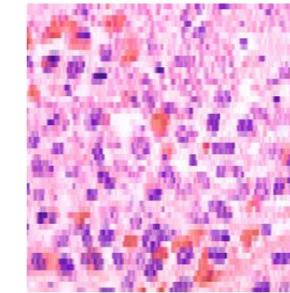
- Voxel size has immediate impact on image quality and thus, on processing / analysis results.



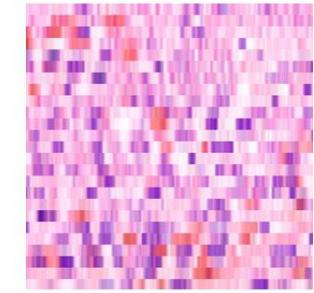
1:1
250 x 250 px



1:2
250 x 125 px



1:5
250 x 50 px



1:10
250 x 25 px

Image source: cropped from
https://de.m.wikipedia.org/wiki/Datei:Histo_Lungenpest.jpg

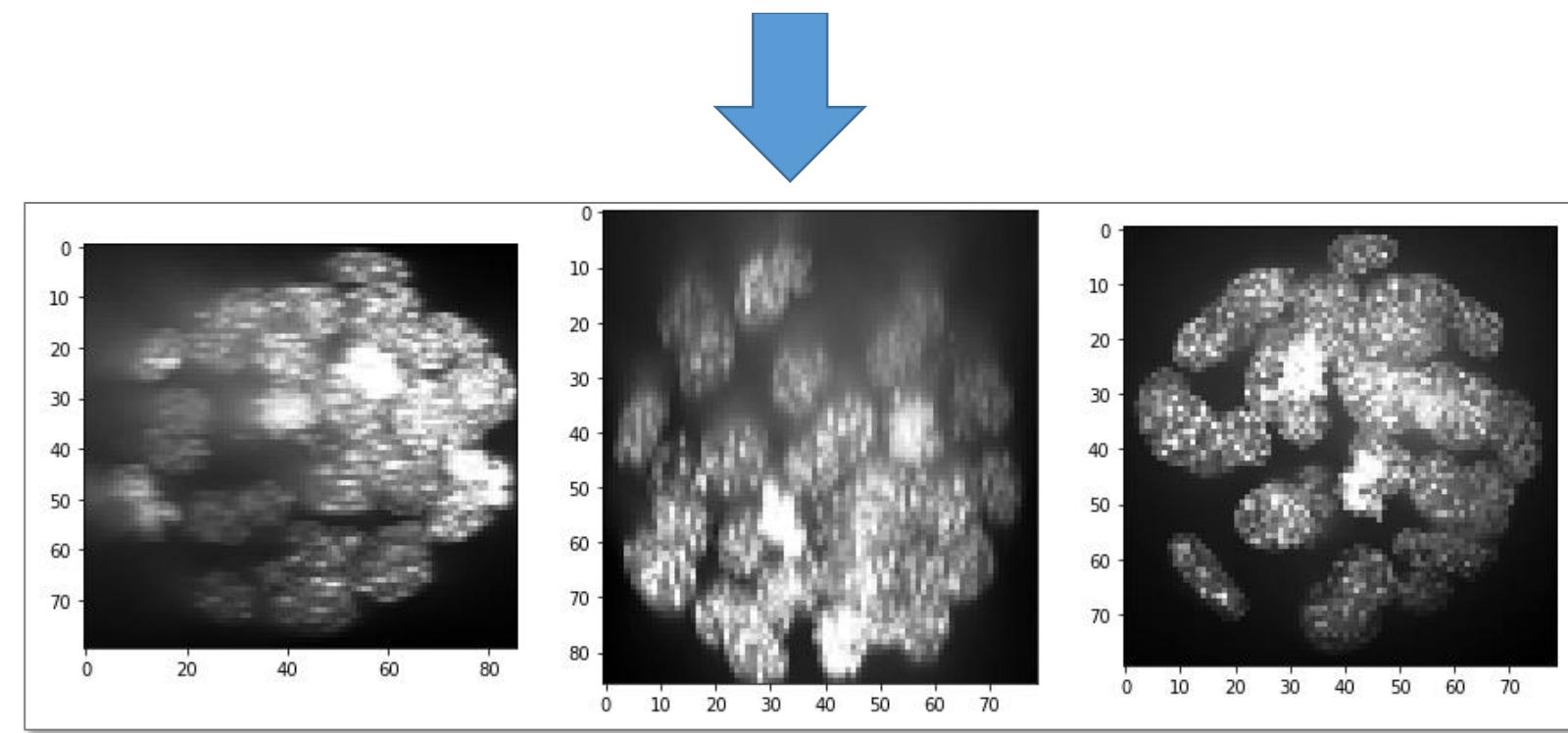
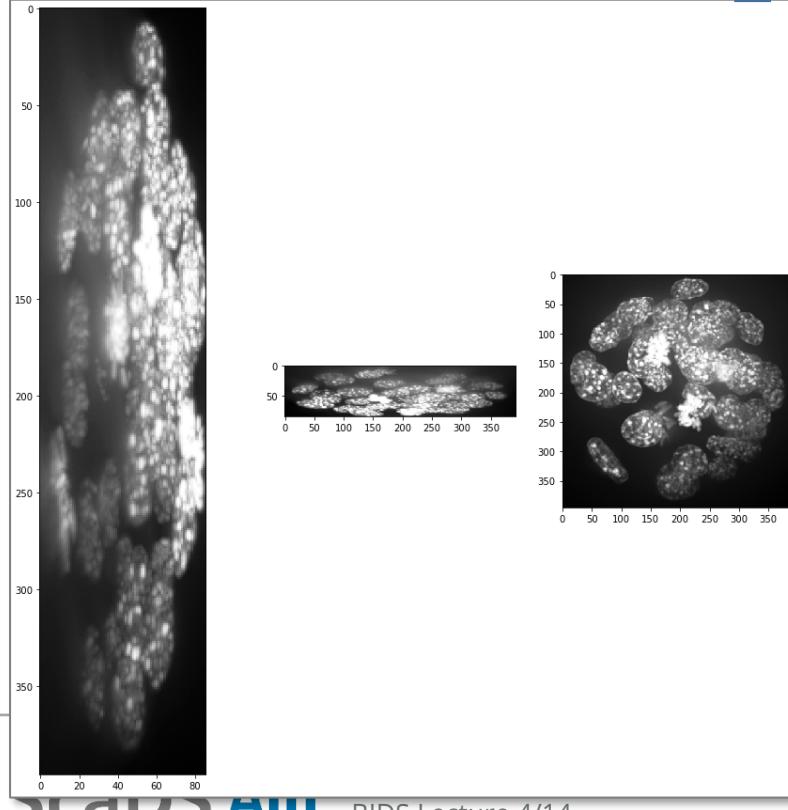
48

 TECHNISCHE
UNIVERSITÄT
DRESDEN  UNIVERSITÄT
LEIPZIG

Reslicing / scaling / sampling

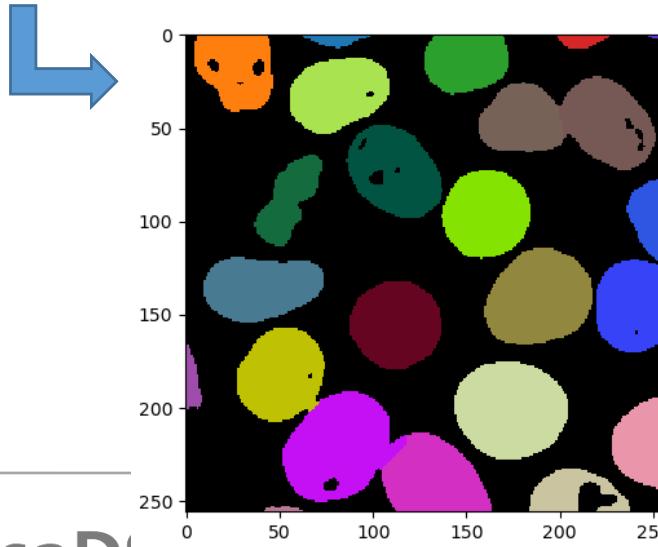
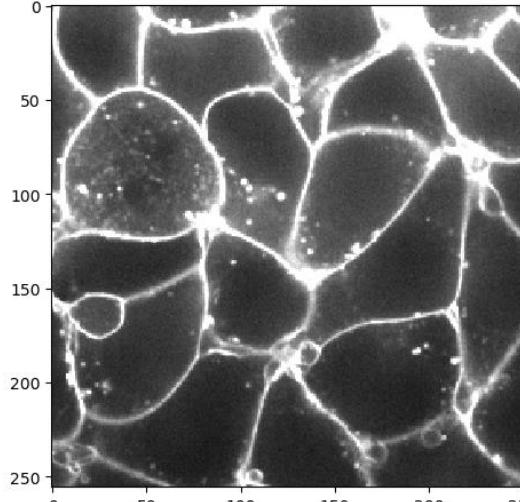
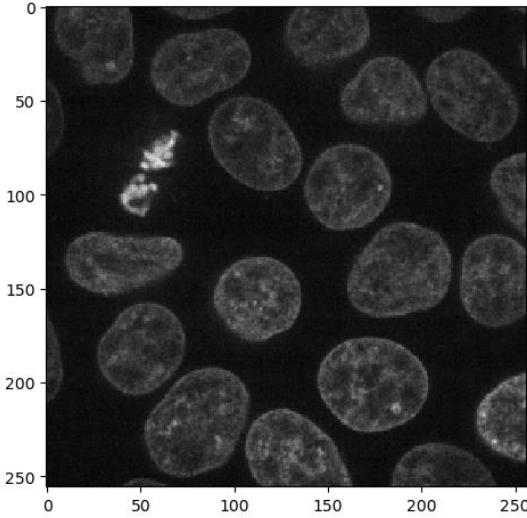
- Resample image data to a specific voxel size

```
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)
```

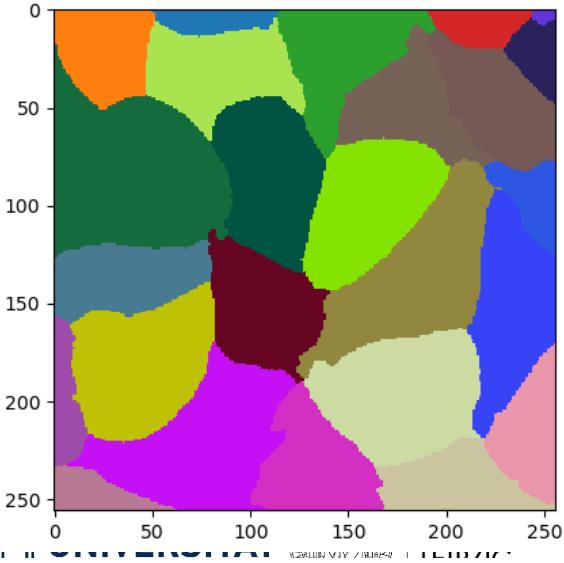


Watershed

- ... in Python practice



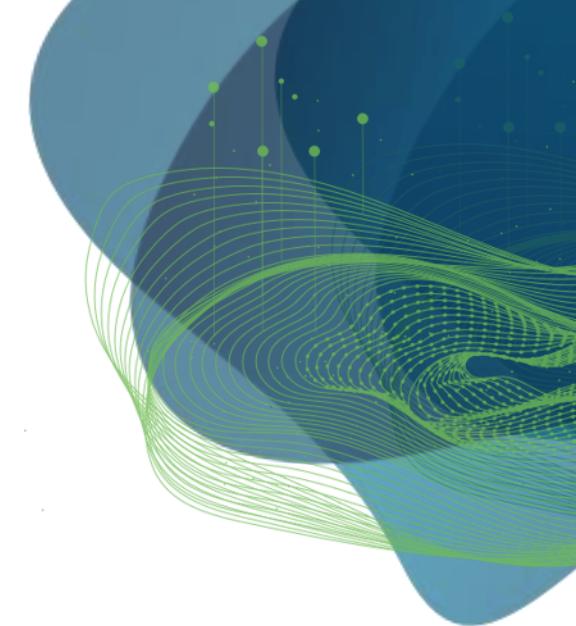
```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)  
labeled_cells
```





DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS
AND ARTIFICIAL INTELLIGENCE



Exercises

Robert Haase

Image segmentation exercises

- Try out segmentation algorithms and apply them to other datasets

The screenshot shows a JupyterLab interface with two panes. The left pane is a file browser showing a directory structure for '04a_image_segmentation' with files like '09_gauss_otsu_labeling.ipynb', '11_voronoi_otsu_labeling.ipynb' (selected), '12_Segmentation_3D.ipynb', and '13_watershed.ipynb'. The right pane contains a code cell and its output. The code cell runs:

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

The output shows the variable `label_image` with the following properties:

nsbatwm made image
shape: (200, 200)
dtype: int32
size: 156.2 kB
min: 0
max: 12

Below the properties is a 200x200 pixel grayscale image where different objects are labeled with distinct colors (purple, green, blue, red, yellow).

The screenshot shows a JupyterLab interface with two panes. The left pane is a file browser showing a directory structure for '04a_image_segmentation' with files like '09_gauss_otsu_labeling.ipynb', '11_voronoi_otsu_labeling.ipynb' (selected), '12_Segmentation_3D.ipynb', and '13_watershed.ipynb'. The right pane contains a code cell and its output. The code cell runs:

```
[ ]:
```

The output is a text block titled 'Exercise' with instructions:

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

The figure consists of four side-by-side screenshots of a JupyterLab interface. Each screenshot shows a file tree on the left and a code editor with a terminal on the right.

- Screenshot 1:** Shows the code for "12_Segmentation_3D.ipynb". It imports necessary libraries and demonstrates a workflow for 3D processing using a GeForce RTX 3050 Ti Laptop GPU.
- Screenshot 2:** Shows the code for "12_Segmentation_3D.ipynb" again, with a different execution history. It includes a note about using isotropic voxels for the Voronoi-Otsu labeling algorithm.
- Screenshot 3:** Shows the resulting visualization of segmented nuclei in a 3D volume. The code used is:

```
segmented = cle.voronoi_otsu_labeling(background_subtracted, spot_sigma=3, outline_sigma=3)
show(segmented, labels=True)
```
- Screenshot 4:** Shows an exercise section titled "Exercise". It asks to load the `skimage.data.cells3d` dataset, extract the nuclei channel, and segment the nuclei. It also asks to print the number of nuclei that do not touch the image border.

Napari - Exercises

- Start using napari from Python

The image shows two side-by-side screenshots of a JupyterLab interface. Both screenshots feature a sidebar on the left with a file browser and a central code editor area.

Left Screenshot: The title bar says "napari_intro.ipynb - JupyterLab". The code editor contains the following code:

```
[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)
```

The output of cell [4] shows a dark napari viewer window with a single blue segmented region.

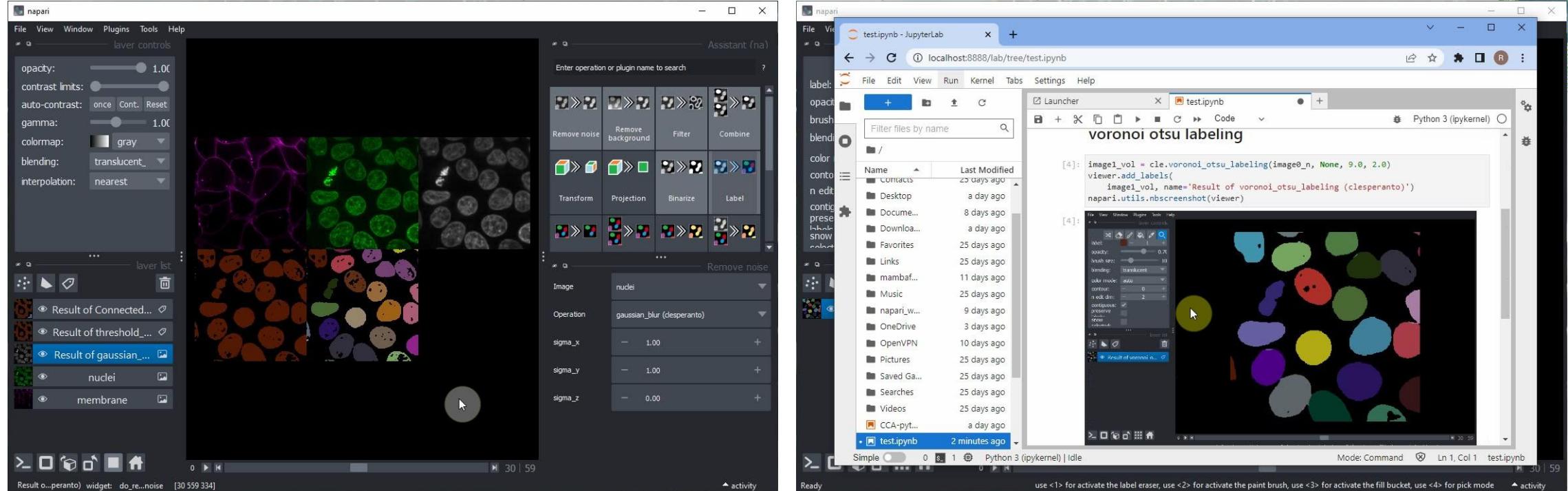
Right Screenshot: The title bar says "napari_intro.ipynb - JupyterLab". The code editor contains the following code:

```
[13]: blurred = gaussian(mri, sigma=5)  
binary_image = blurred > threshold_otsu(blurred)  
viewer.add_labels(binary_image)  
napari.utils.nbscreenshot(viewer)
```

The output of cell [13] shows a napari viewer window displaying a grayscale MRI scan with a red segmented region overlay.

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/notebook_export.md