



DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS
AND ARTIFICIAL INTELLIGENCE

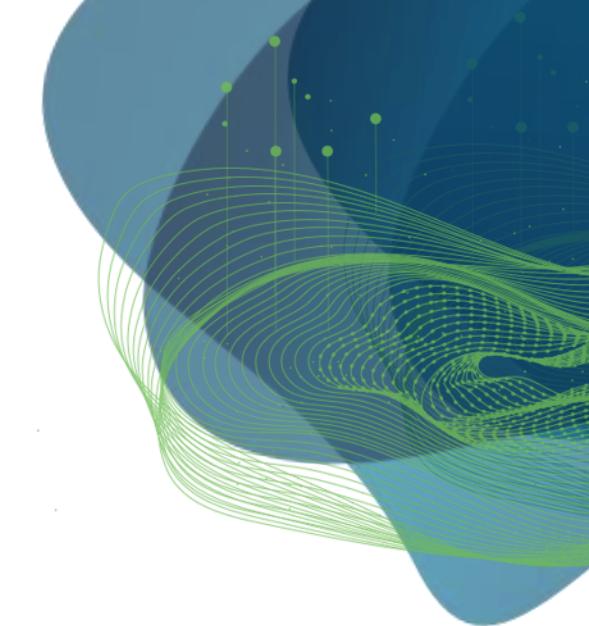


Image segmentation

Robert Haase

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

GEFÖRDERT VOM



Bundesministerium
für Bildung
und Forschung

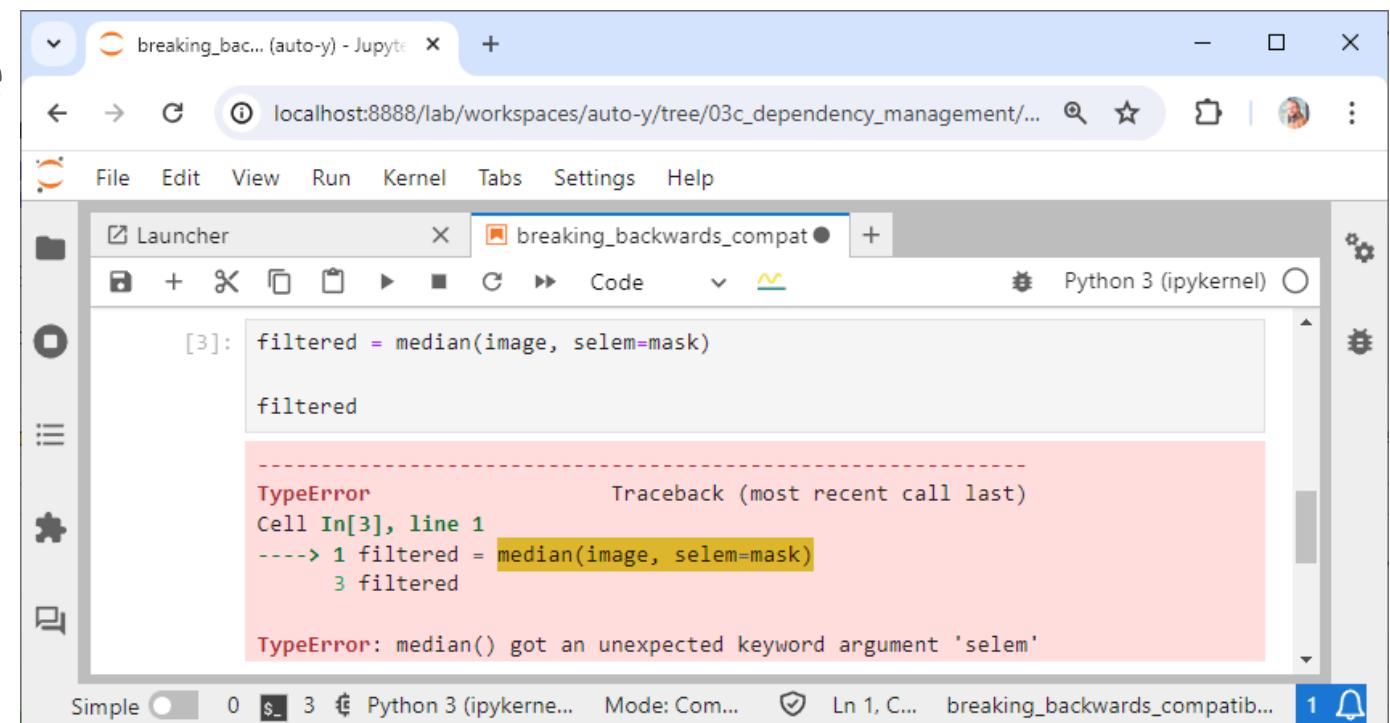


Diese Maßnahme wird gefördert durch die Bundesregierung
aufgrund eines Beschlusses des Deutschen Bundestages.
Diese Maßnahme wird mitfinanziert durch Steuermittel auf
der Grundlage des von den Abgeordneten des Sächsischen
Landtags beschlossenen Haushaltes.

Recap

How can one solve this problem?

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

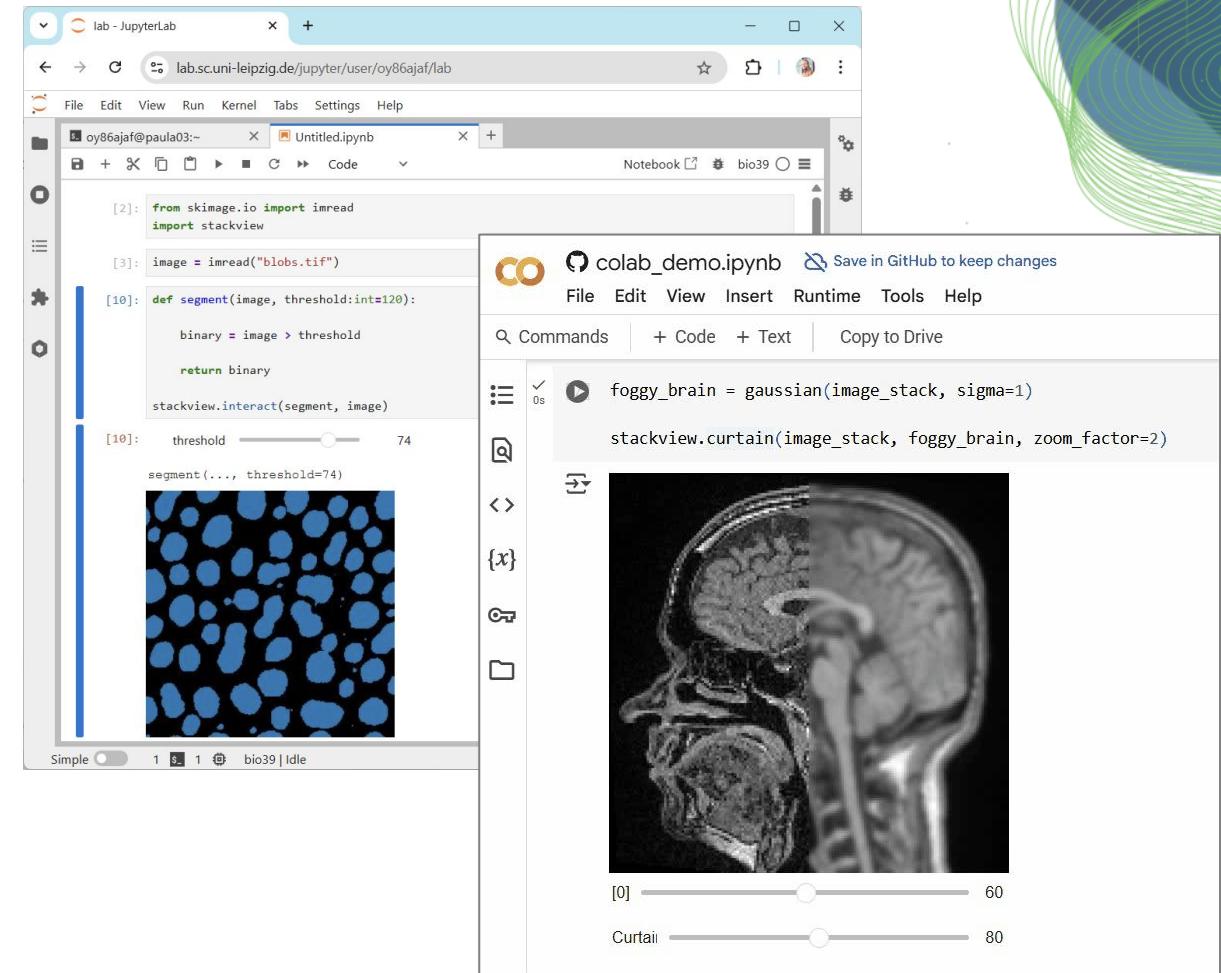


The screenshot shows a Jupyter Notebook interface with a single code cell. The code in the cell is:filtered = median(image, selem=mask)
filteredA red box highlights the error message in the output area:

```
-----  
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'
```

Recap

- The stackview installation on the Jupyter Hub was broken. The SC/IT had to install ipywidgets in the GUI.
- For google colab there is a new stackview 0.16.0 release
- Both environments *should* work now.



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

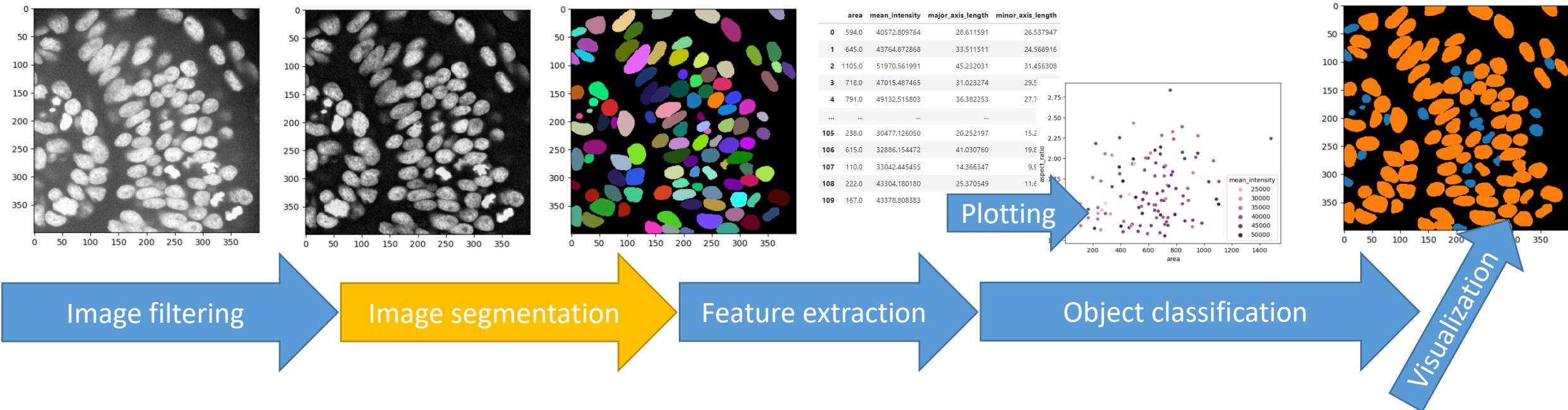


Difference
of Gaussian

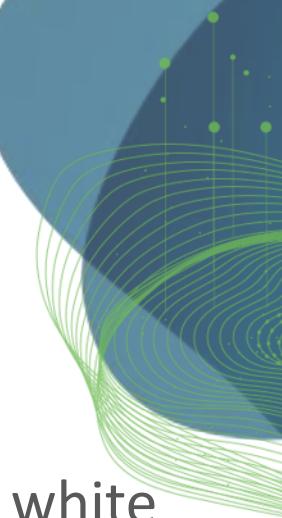


Lecture overview: Bio-image Analysis

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



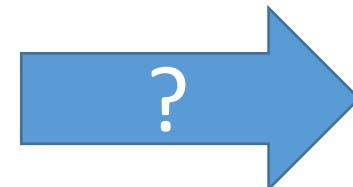
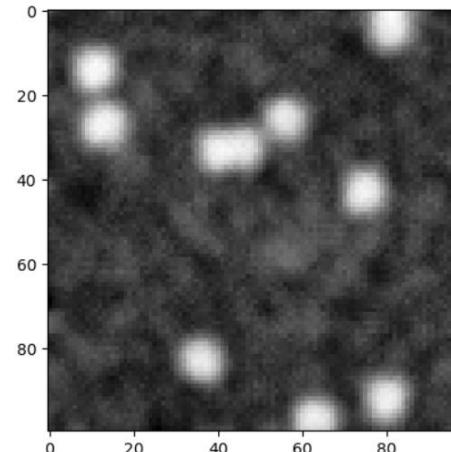
Segmentation: Binarization



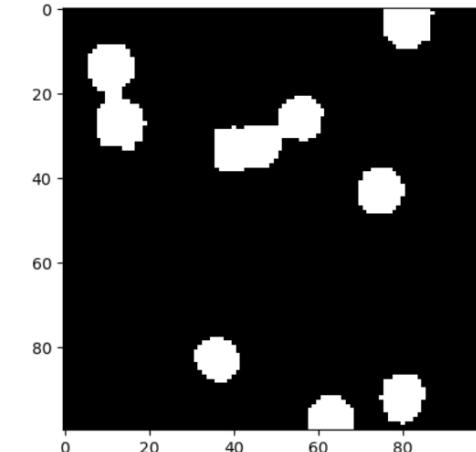
Binarization, e.g. via thresholding

- Very basic and yet efficient segmentation technique
- Histogram based, to determine an intensity threshold above which pixels become white
- Not state-of-the-art in many fields anymore

Intensity image



Binary image



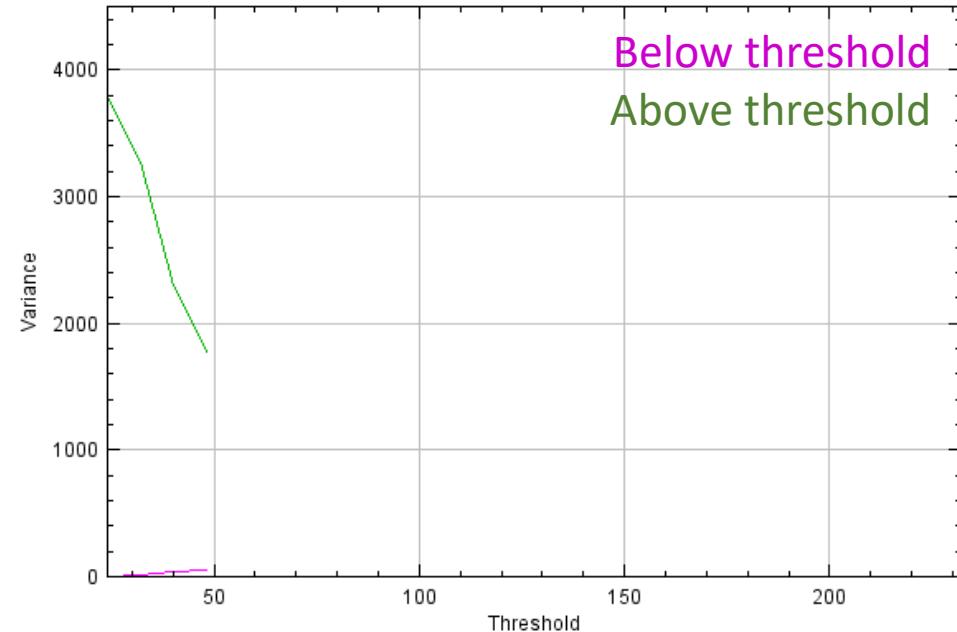
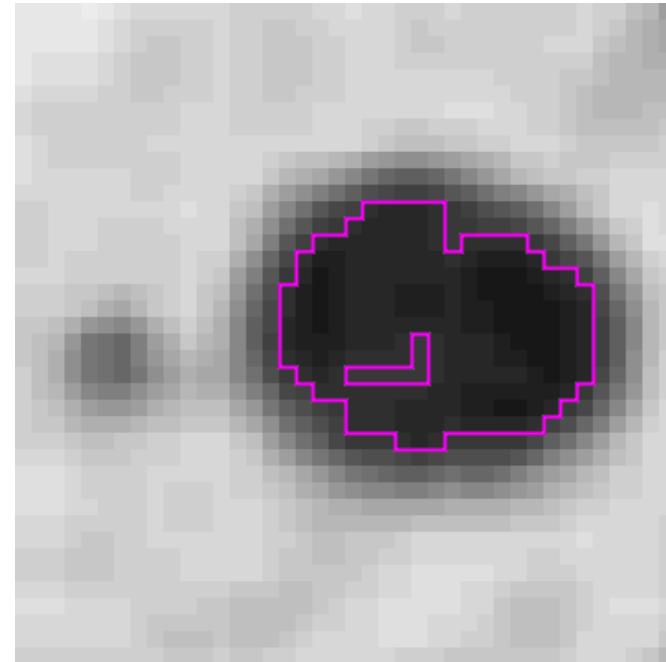
Thresholding: Otsu's method

- 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



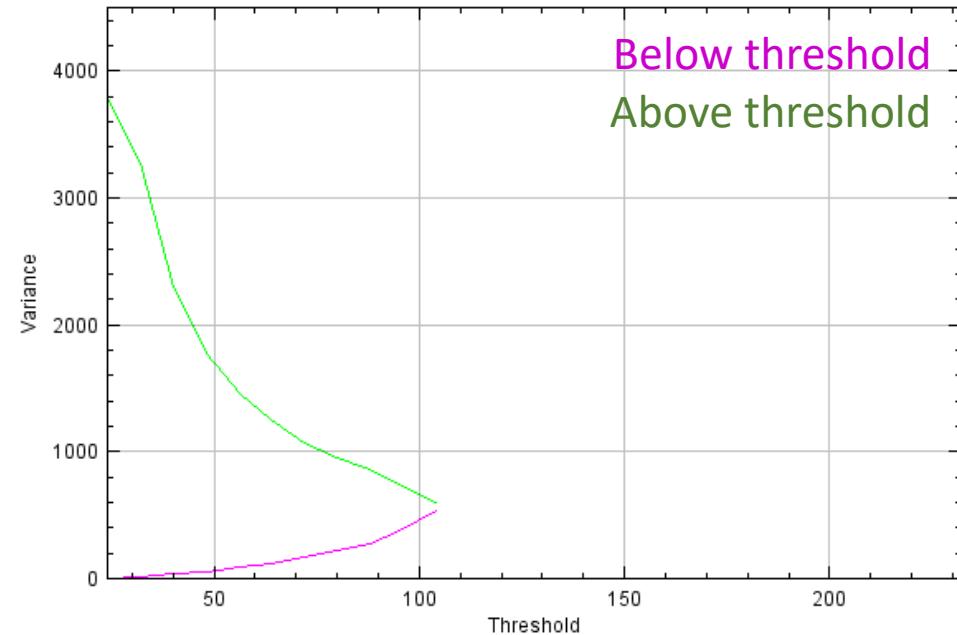
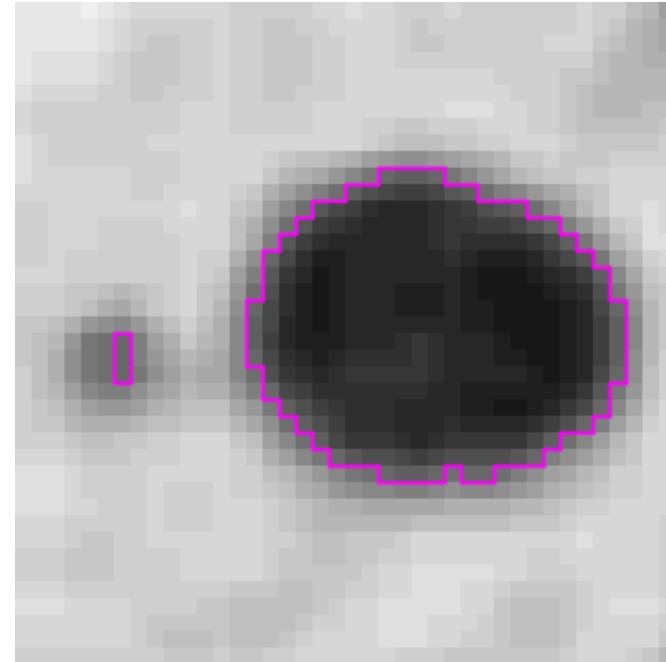
Thresholding: Otsu's method

- 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



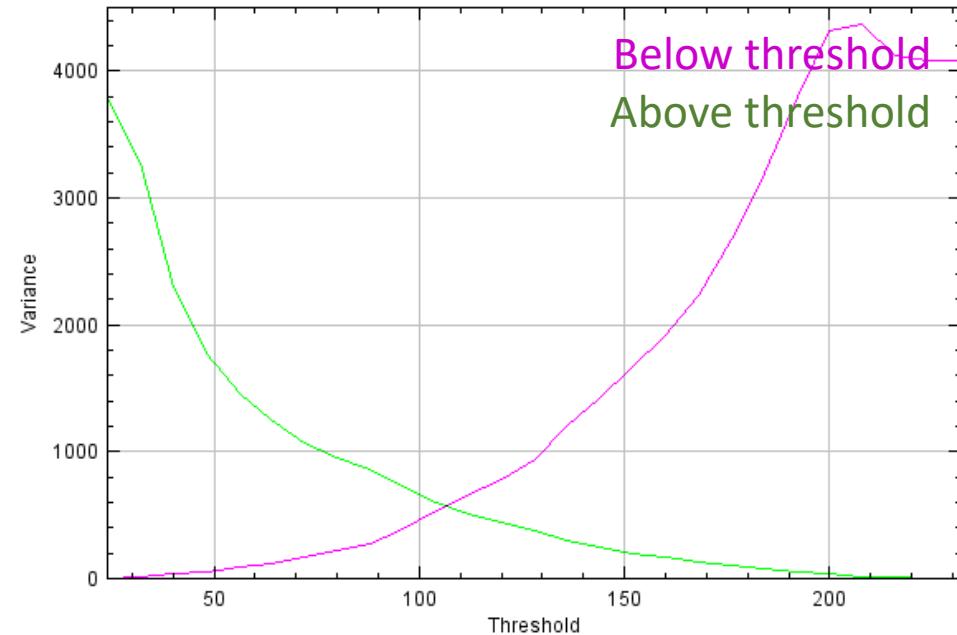
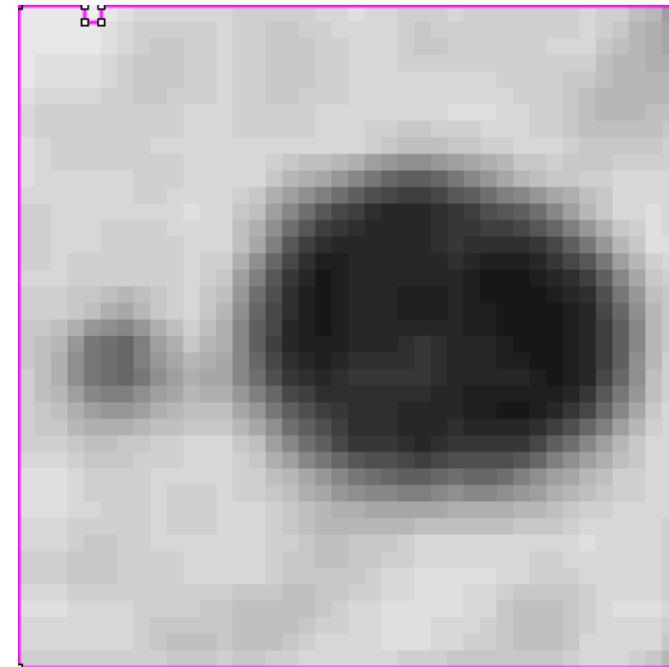
Thresholding: Otsu's method

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

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

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

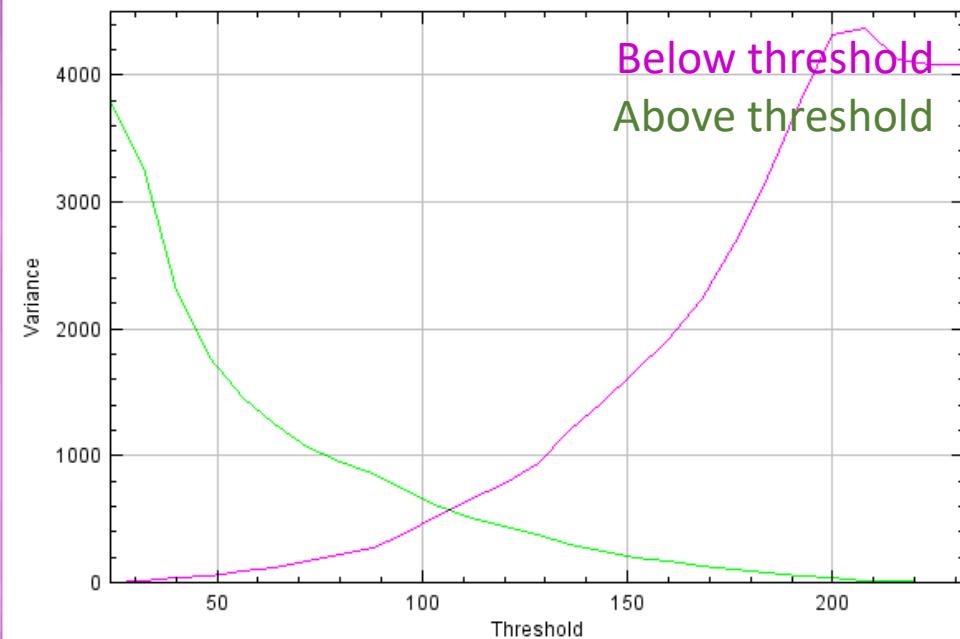
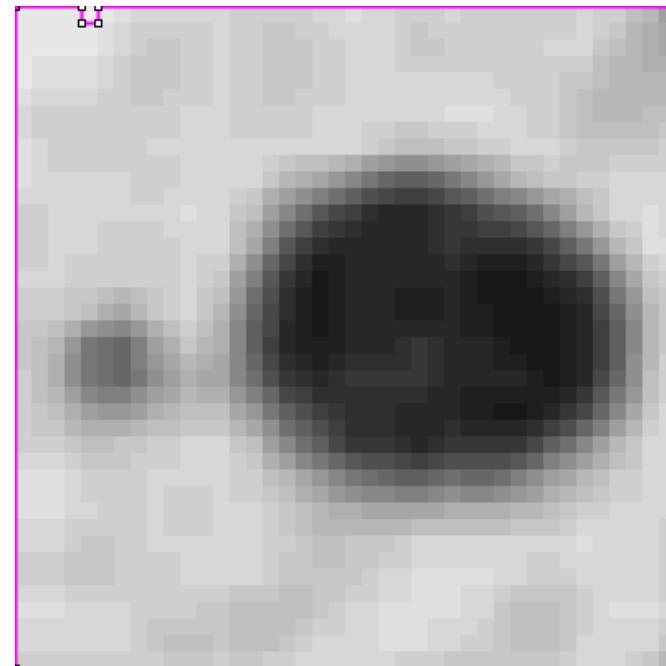
$Var(I)$... Variance in image I
 g_i ... grey value of a pixel i
 \bar{g}_I ... mean grey value of the whole image I
 n_I ... number of pixels in Image I



Thresholding: Otsu's method

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

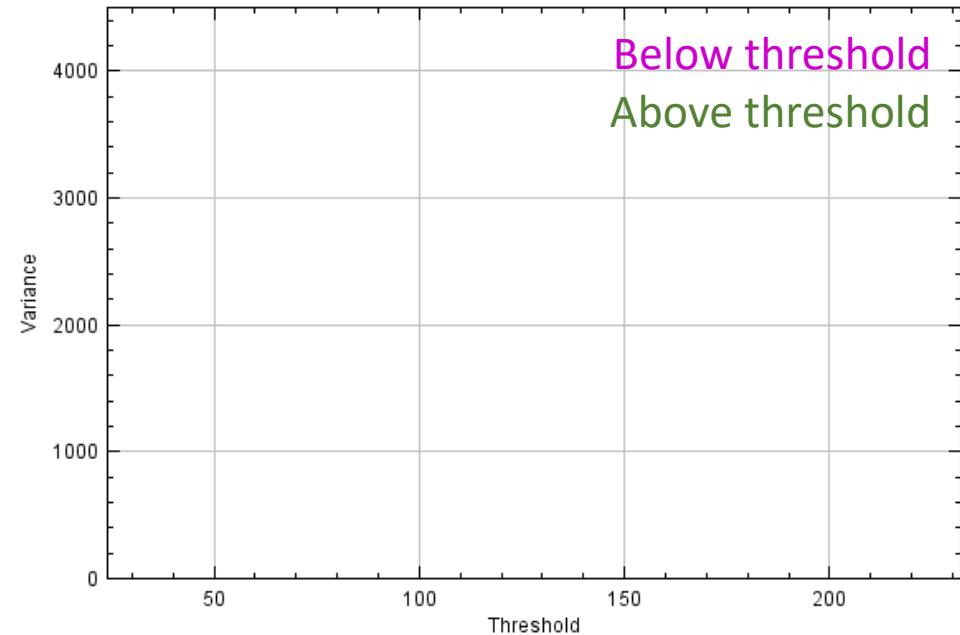
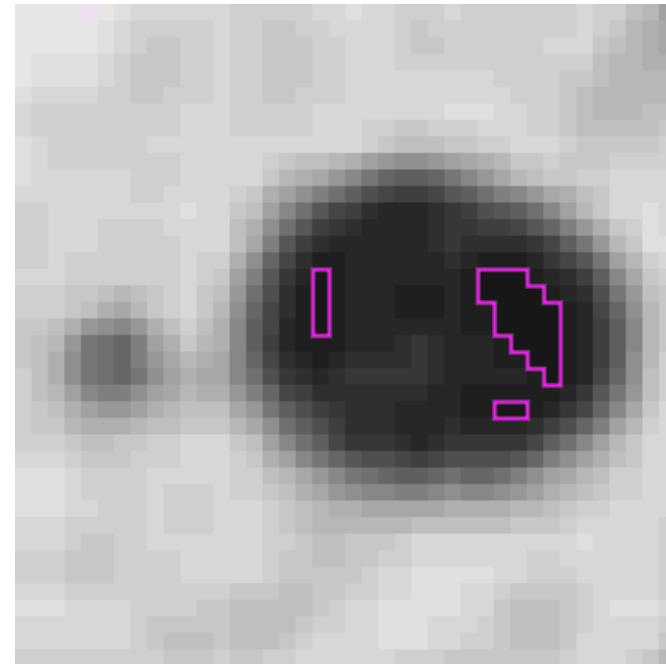
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



Thresholding: Otsu's method

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

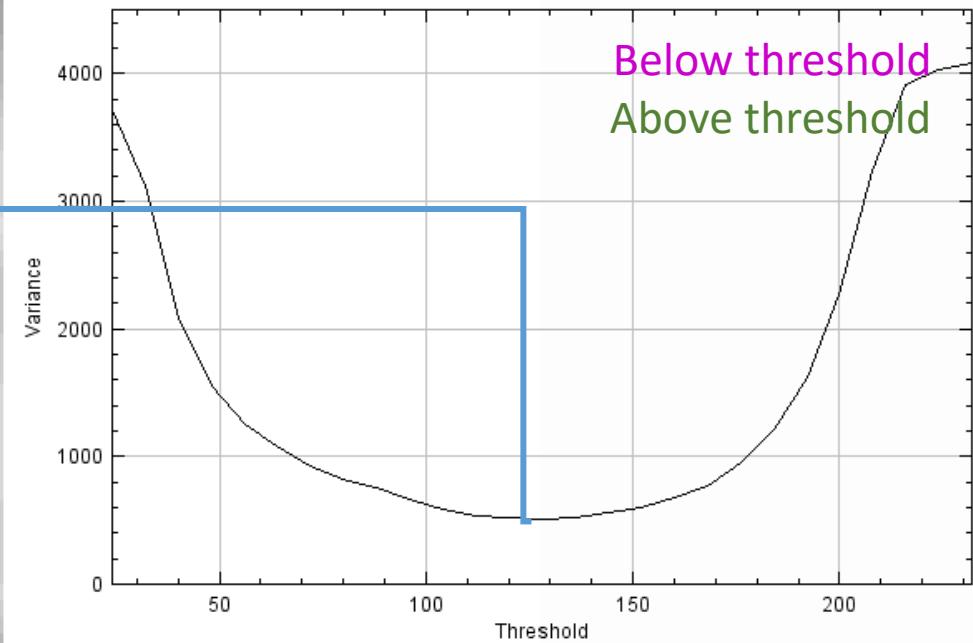
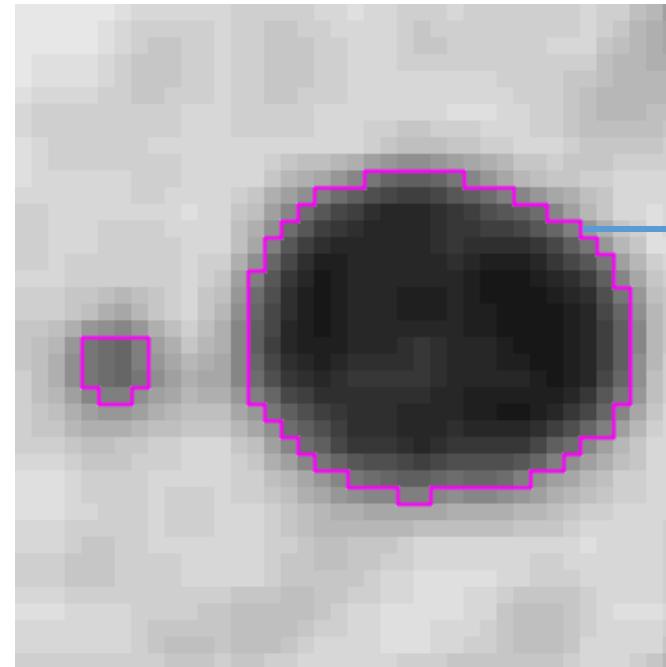
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



Thresholding: Otsu's method

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

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



Citing Otsu's method / implementation

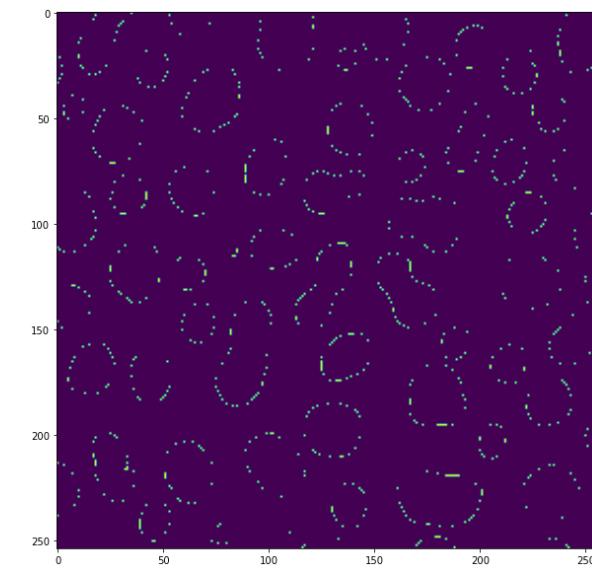
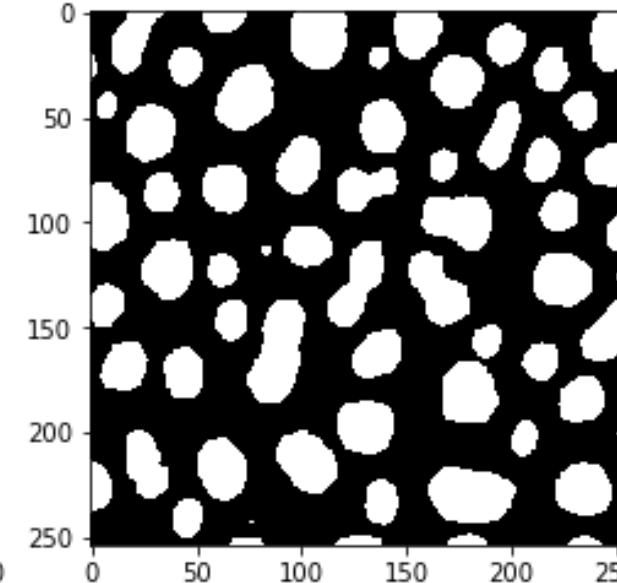
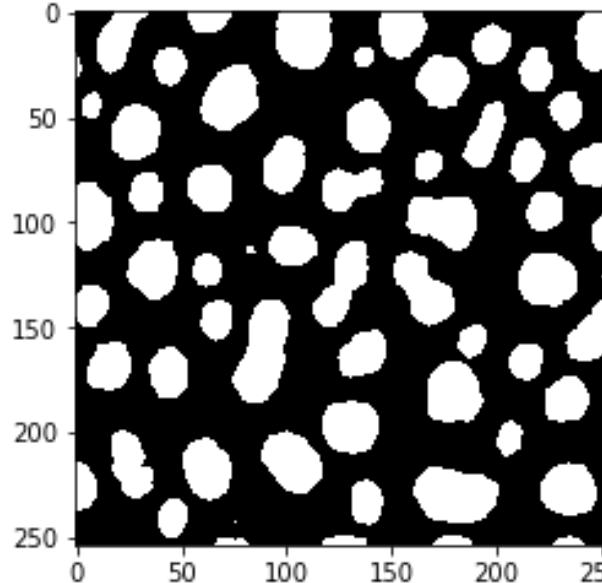
- Cite the thresholding method of your choice properly

“We segmented the cell nuclei in the images using Otsu’s thresholding method (Otsu et Al. 1979) implemented in scikit-image 0.19.3 (van der Walt et al. 2014).”

Quiz: Why do we
need to report this
so detailed?

Reproducibility of Otsu's method

- Comparing scikit-image versus ImageJ



Different pixels:

`np.sum(difference)`

830

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

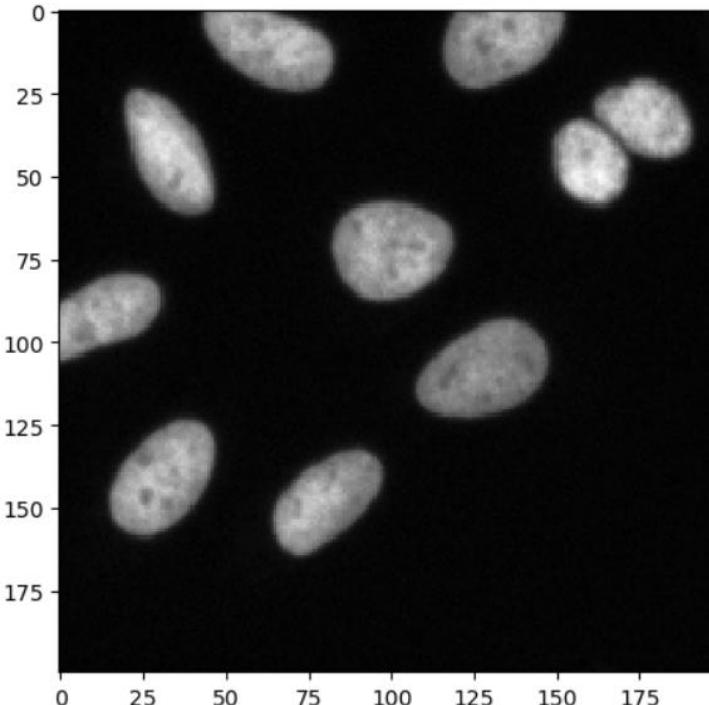


Maximum

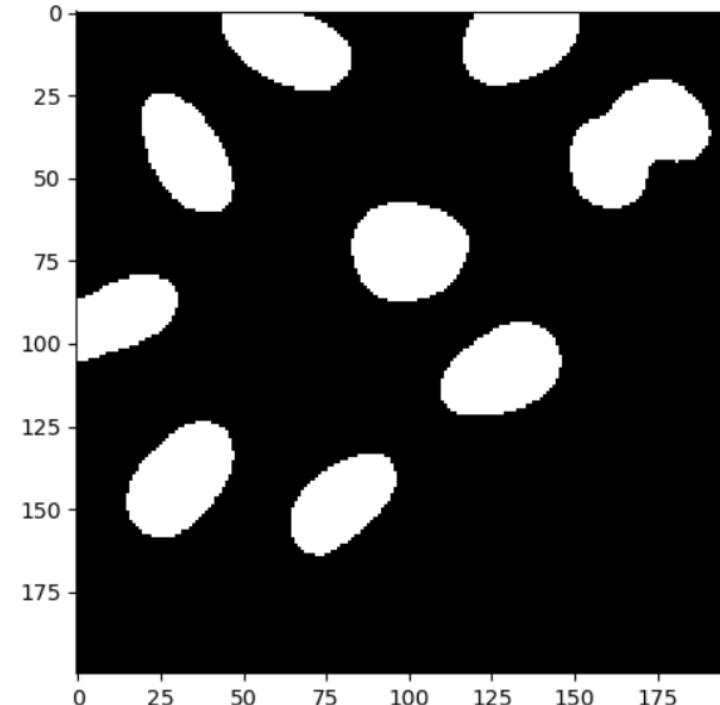


Terminology

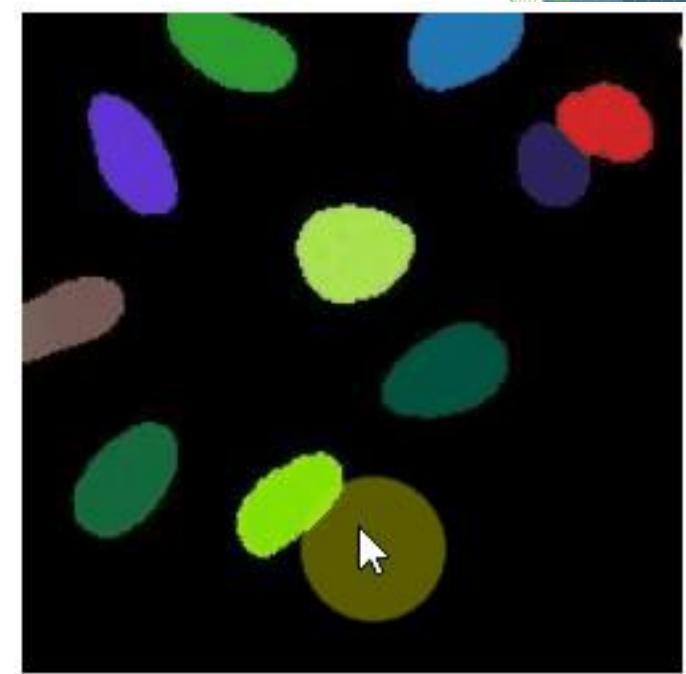
Intensity image



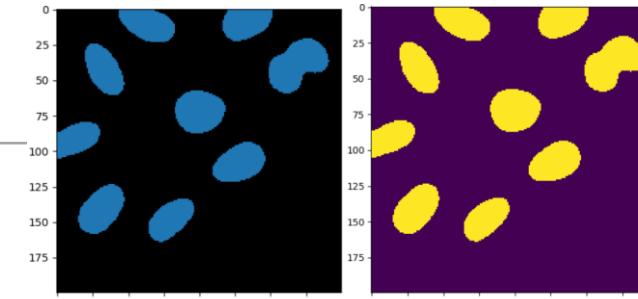
Binary image



Label image



No matter how they are displayed



Terminology

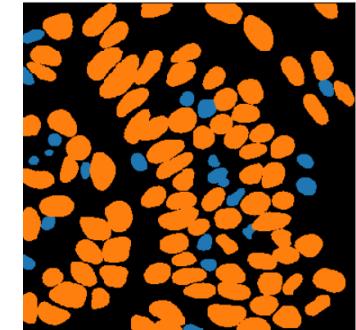
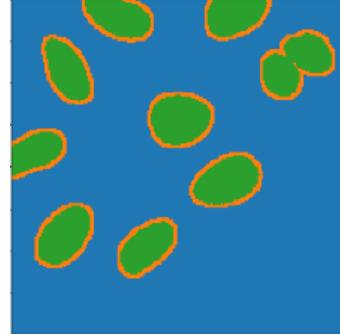
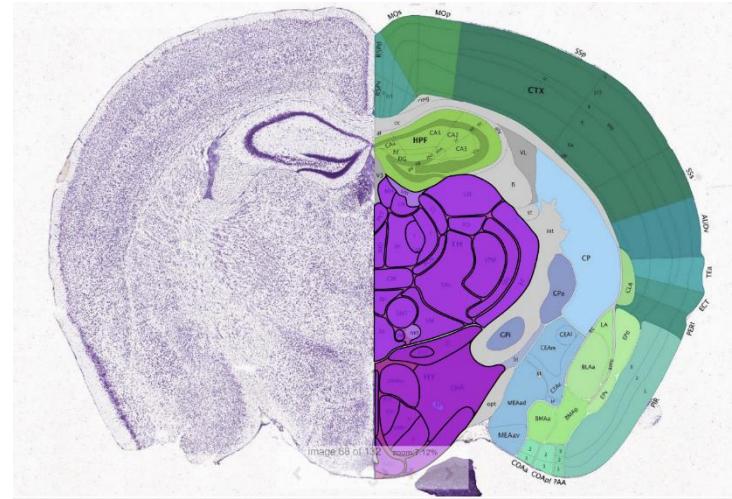
Instance segmentation



Instances:

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

Semantic segmentation



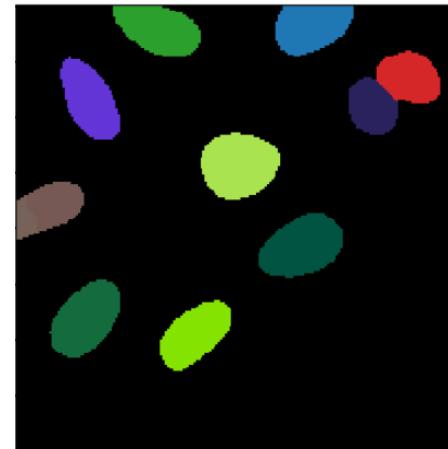
Regions:

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

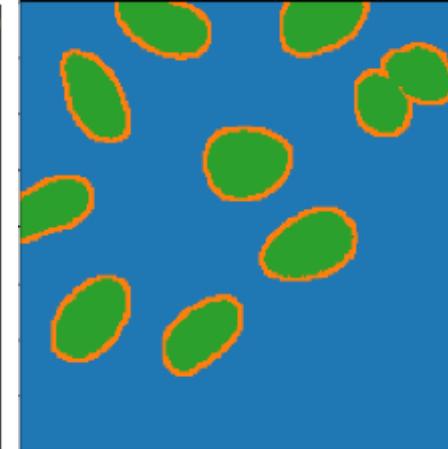
Terminology

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

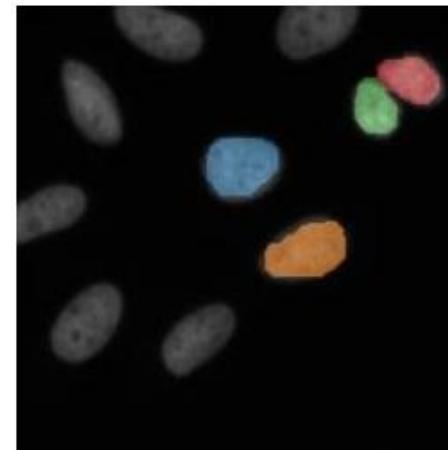
Instance
segmentation



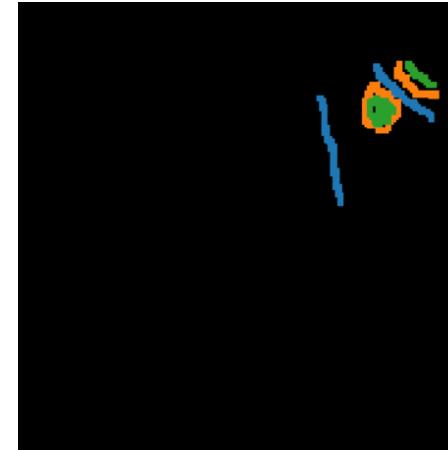
Semantic
segmentation



Sparse instance
annotation

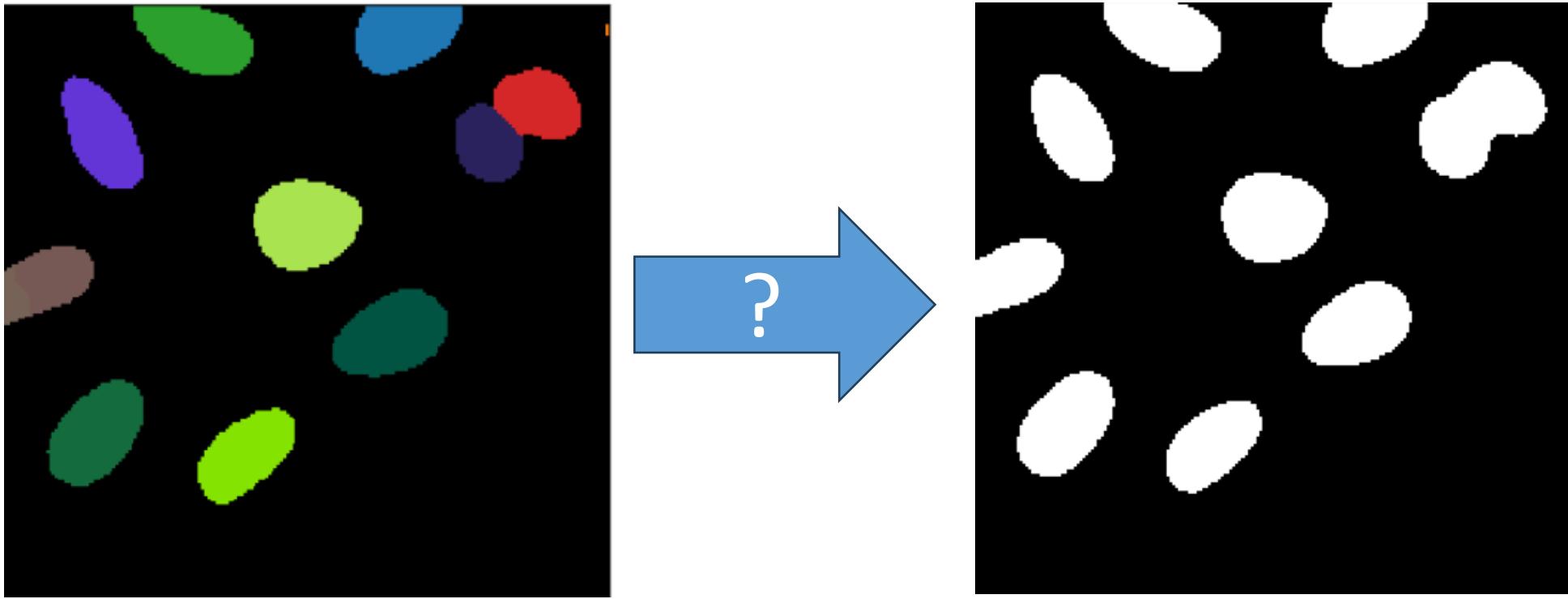


Sparse semantic
annotation



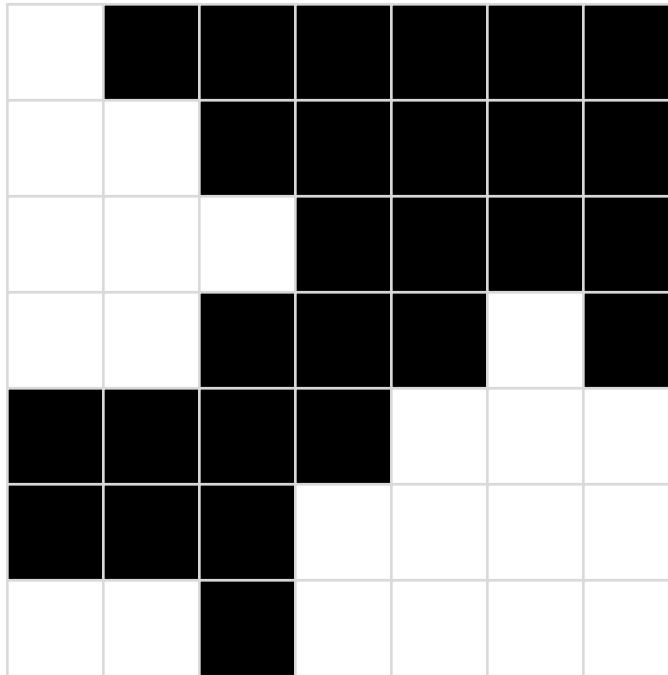
Quiz

- How could I turn a label image into a binary image?



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 such a label map corresponds to the number of objects.



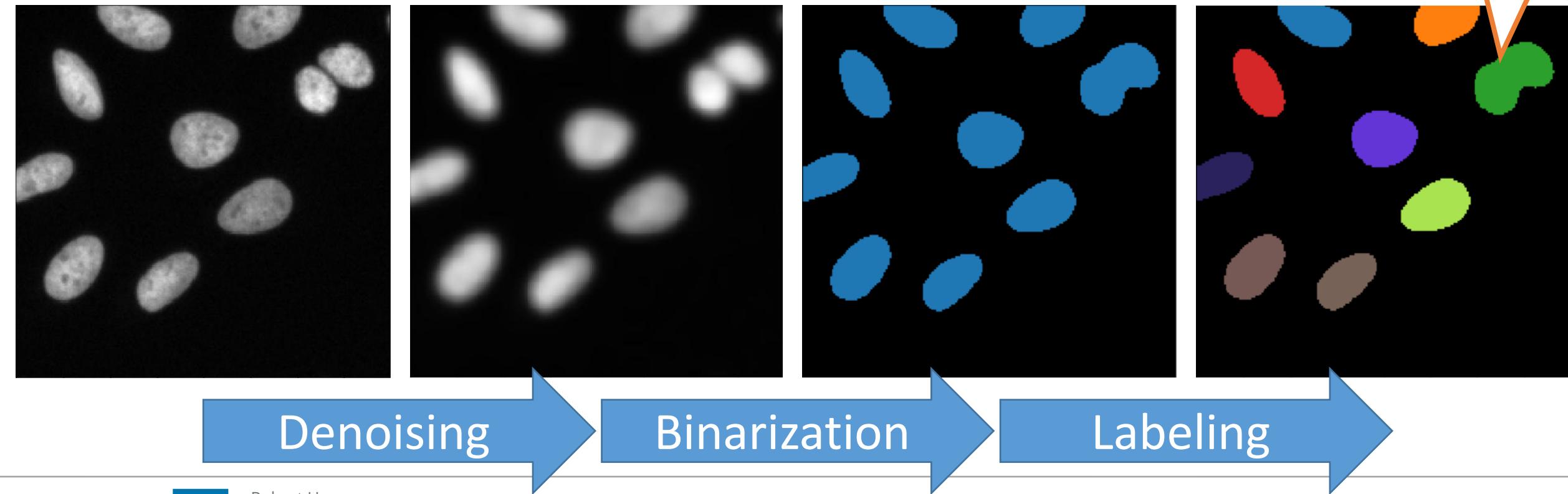
CCA

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

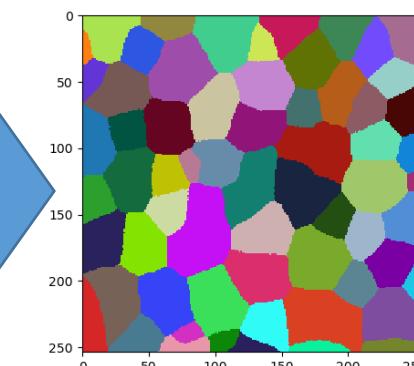
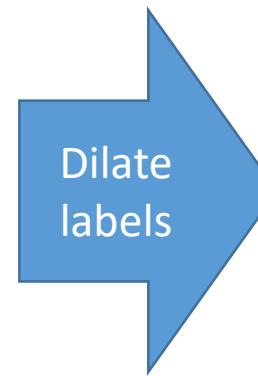
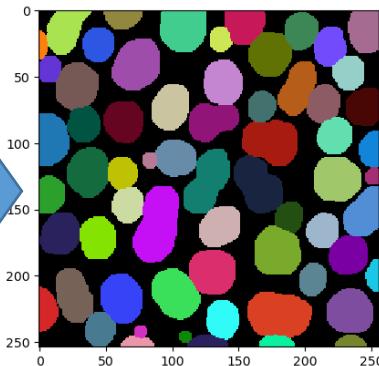
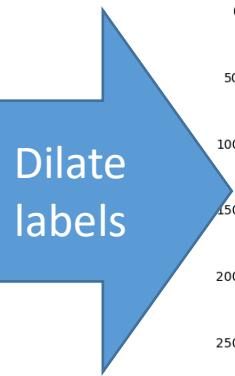
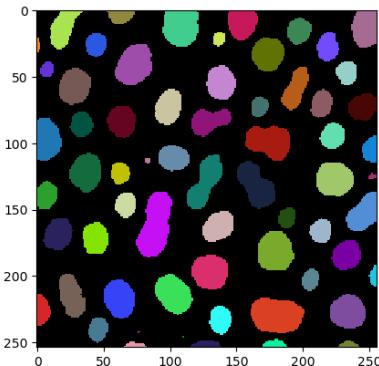
- Presumably the most common segmentation algorithm used for fluorescence microscopy images:
 - Gaussian blur, Otsu's Threshold, Connected Component Labeling

Limitation:
Dense
objects

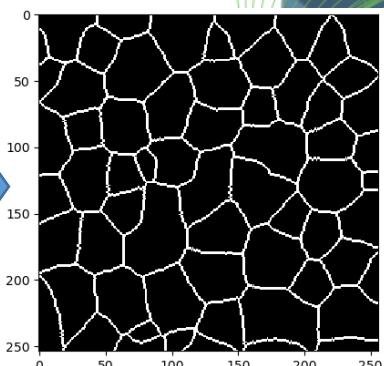
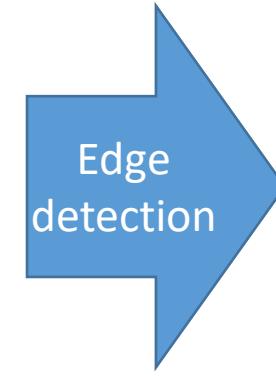


Voronoi-Tesselation

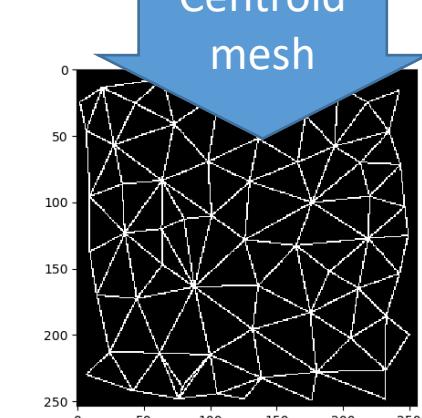
- For separating objects using spatial constraints (not intensity-based)



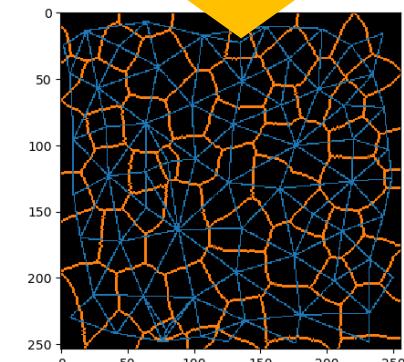
Voronoi-label-image
Centroid mesh



Label-edge-image



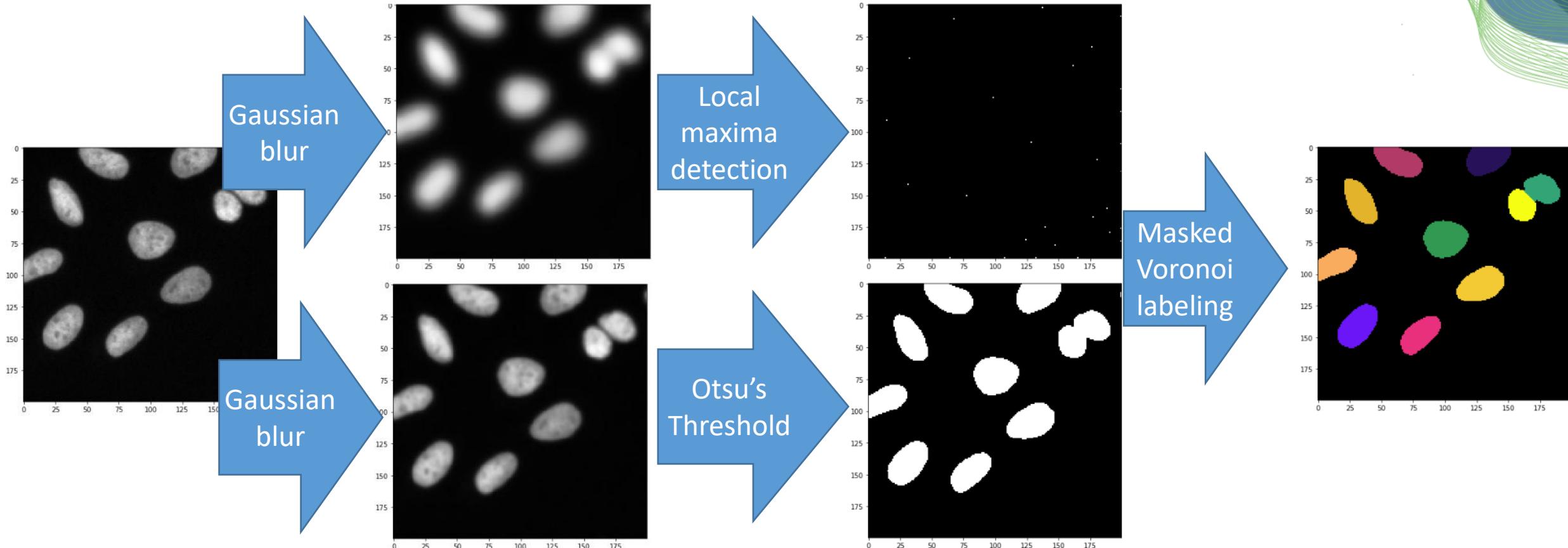
Touching neighbor mesh



Voronoi-Tesselation
Delauney-Triangulation

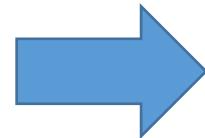
Common image segmentation workflows

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



Voronoi-Otsu-Labeling

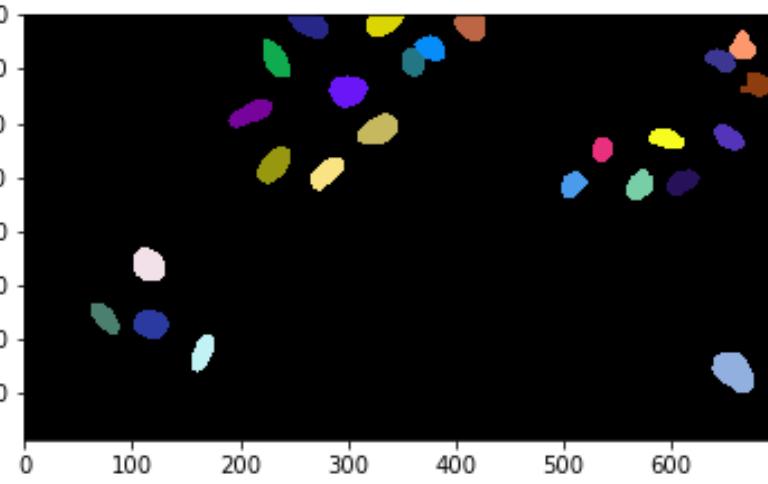
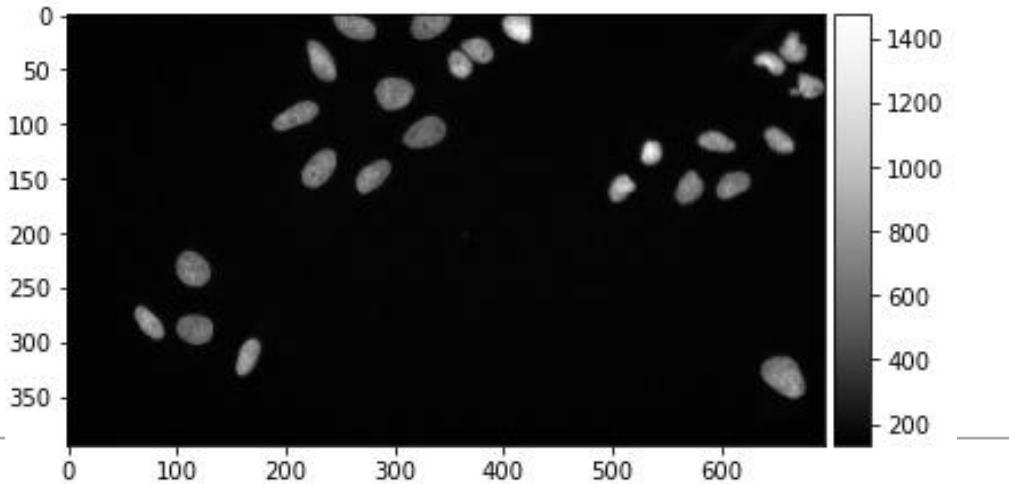
- Gaussian-Blur
- Otsu-Thresholding
- Spot-detection
- Watershed on the binary image



... 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

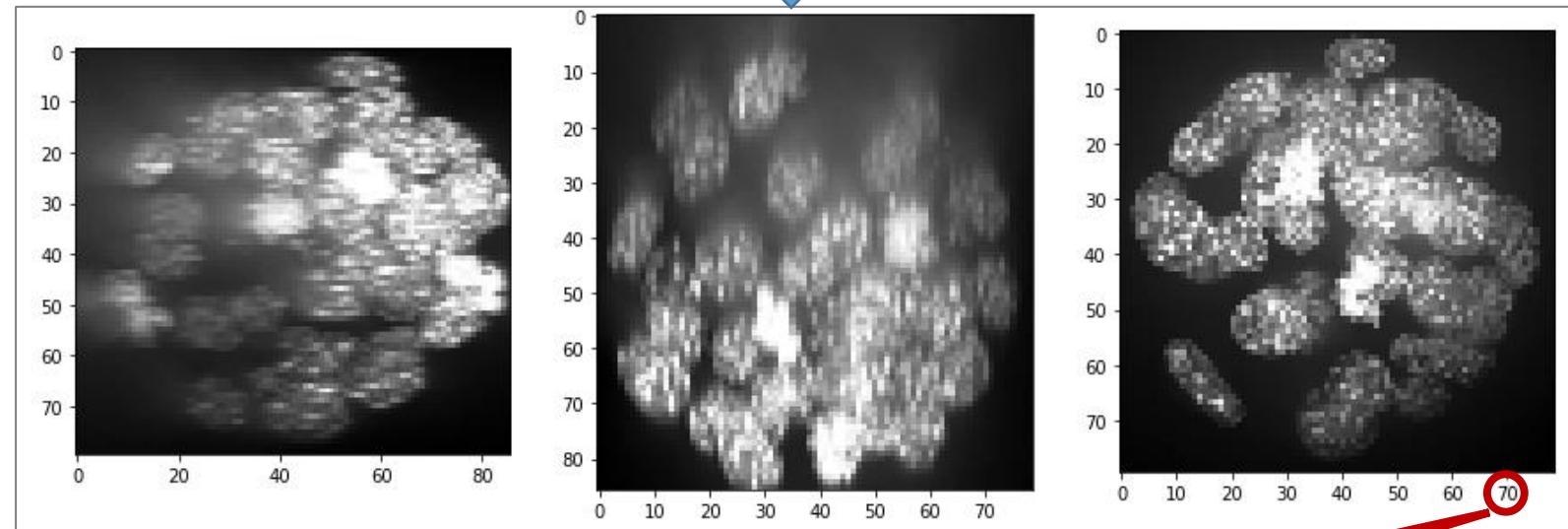
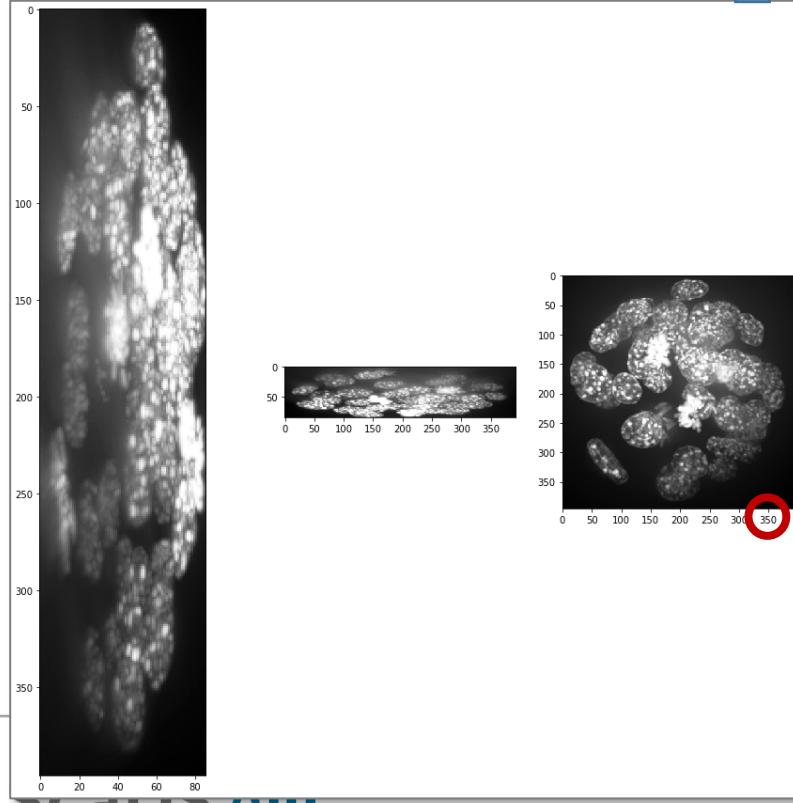
- Some [segmentation] algorithms have prerequisites...

```
[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
```

Reslicing / scaling / sampling

- Resample image data to a specific voxel size

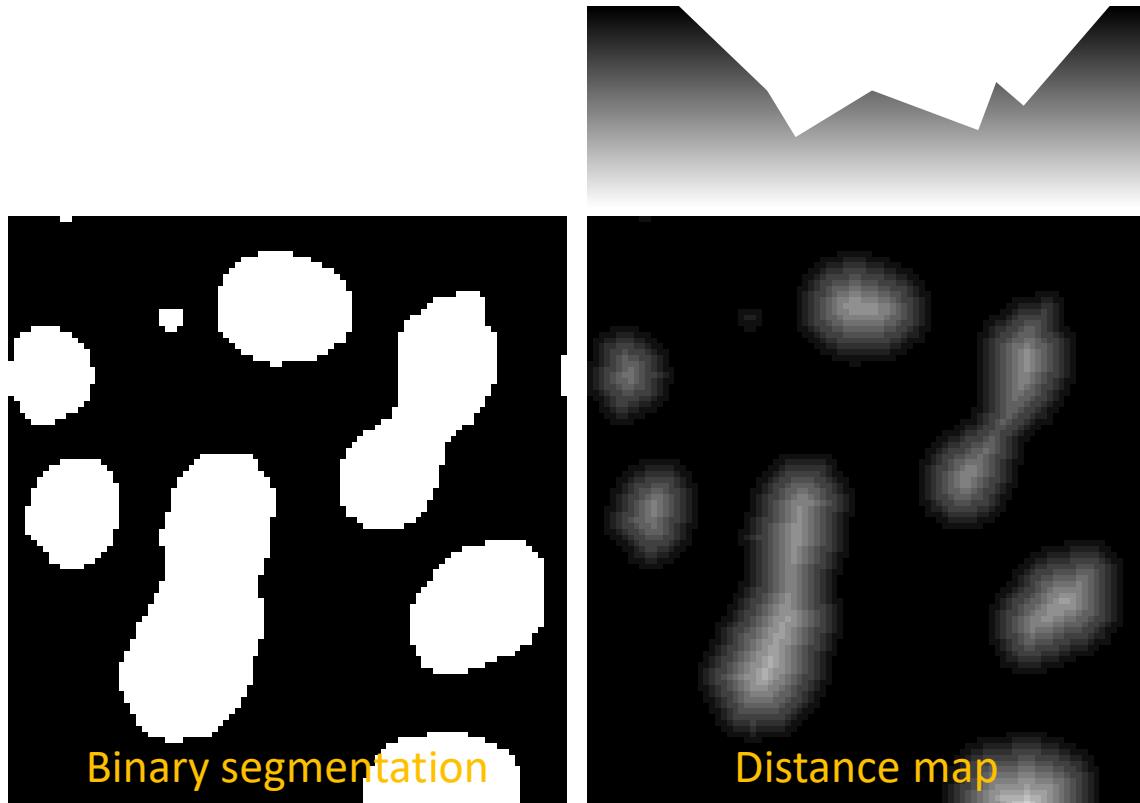
```
resampled = cle.scale(input_image, factor_x=voxel_size_x, factor_y=voxel_size_y, factor_z=voxel_size_z, auto_size=True)  
show(resampled)
```



Down-sampling versus up-sampling,
in this case:
670kByte versus 89mByte

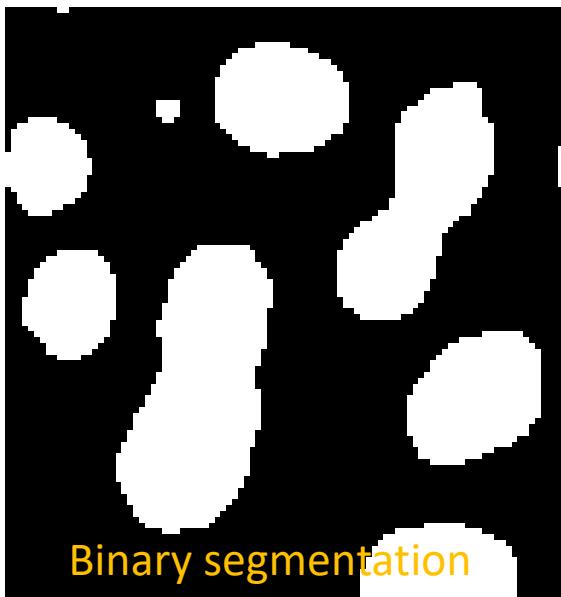
Watershed

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

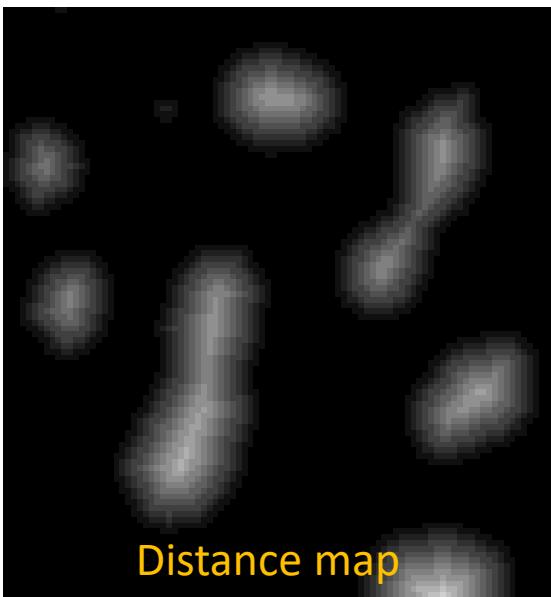


Watershed

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



Binary segmentation

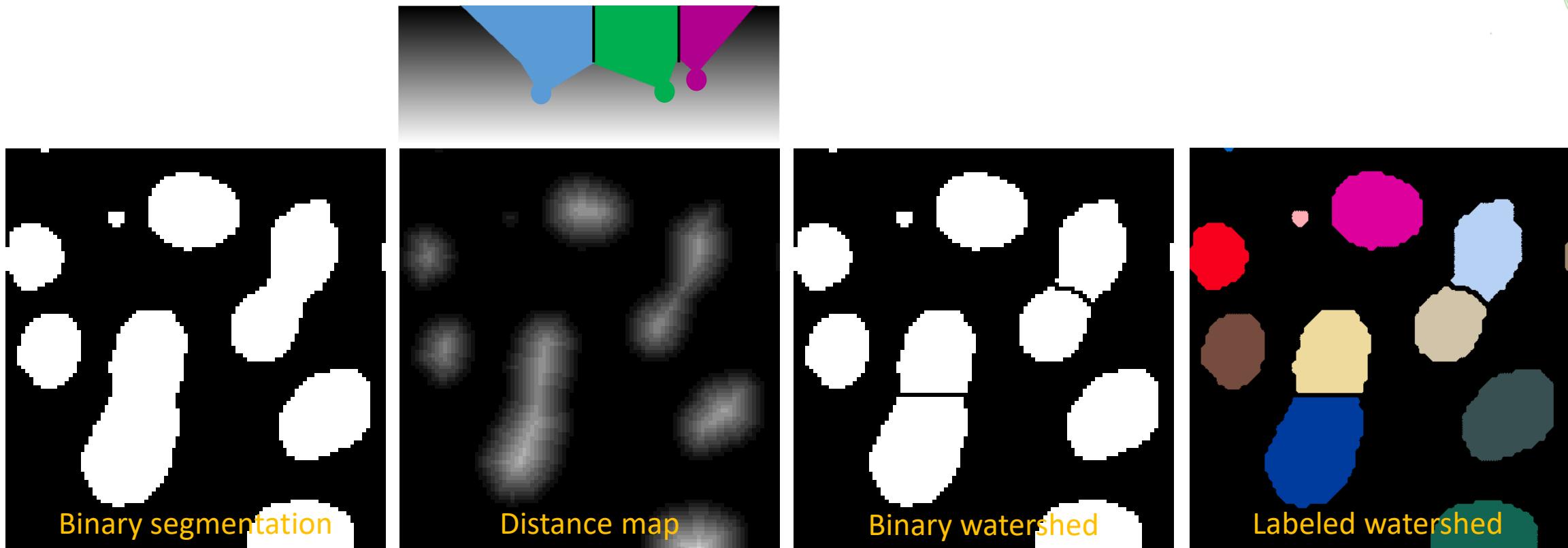


Distance map



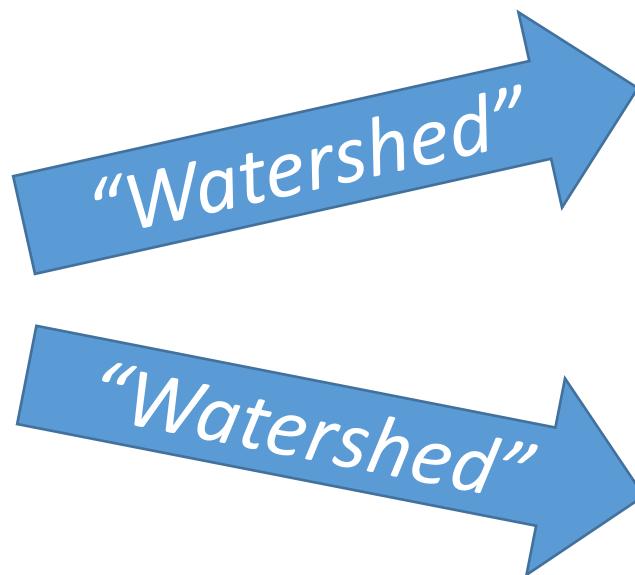
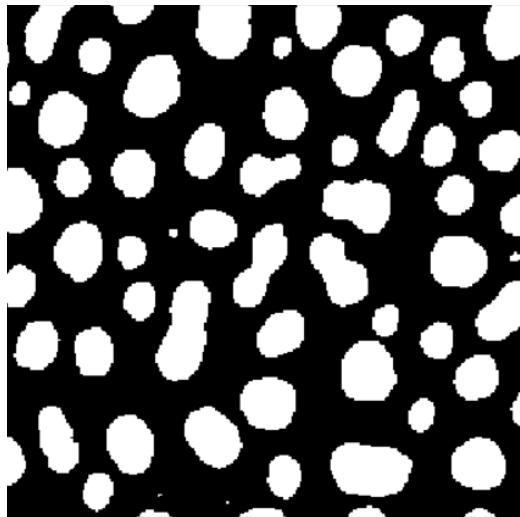
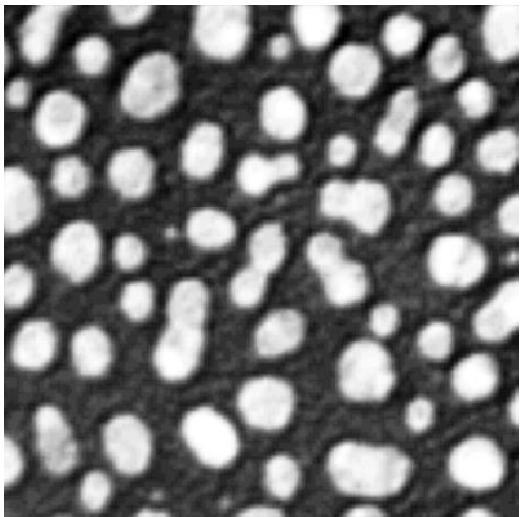
Watershed

- 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!

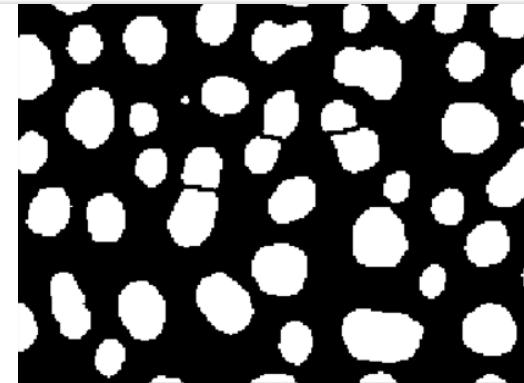


Watershed use-cases

- Split dense objects



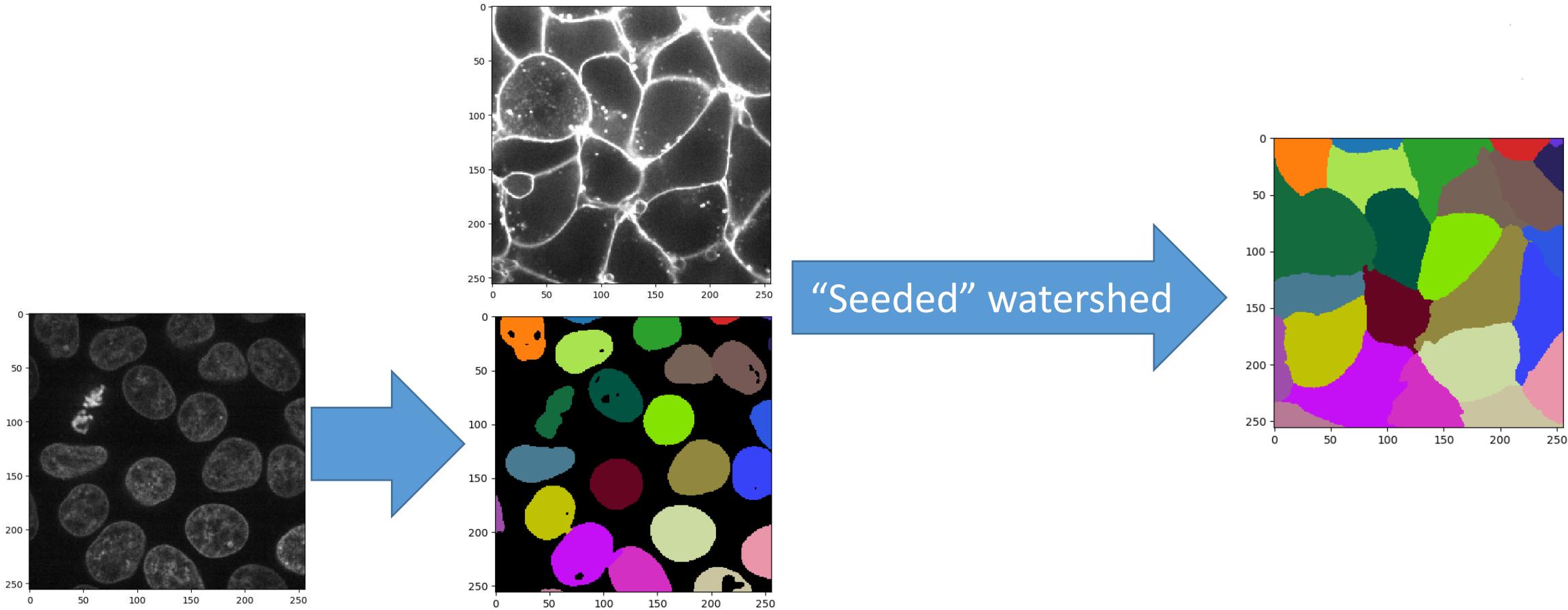
```
split_objects = nsbatwm.split_touching_objects(binary)  
split_objects
```



```
touching_labels = nsitk.touching_objects_labeling(binary)  
touching_labels
```

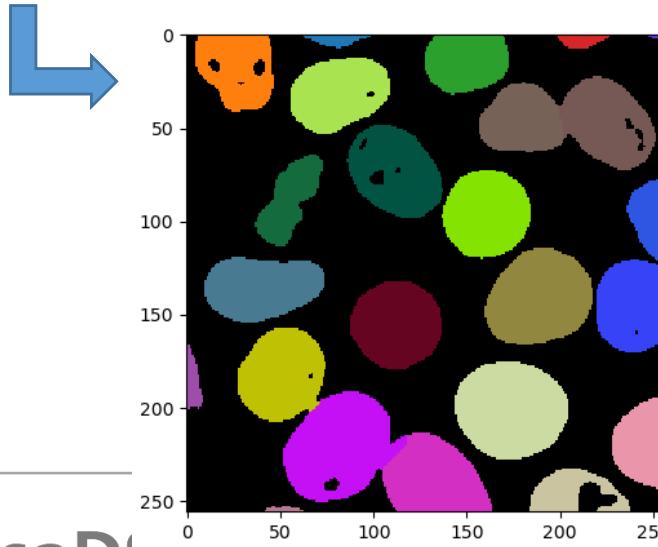
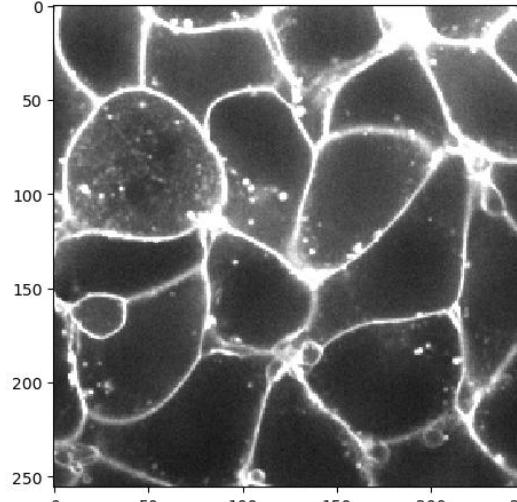
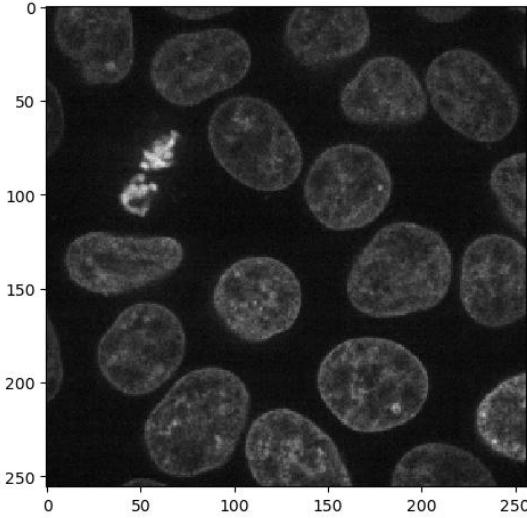
Watershed use-cases

- Seeded watershed: Flood regions from pre-defined seeds
- Example: Flood membrane image from nuclei positions (no distance map)

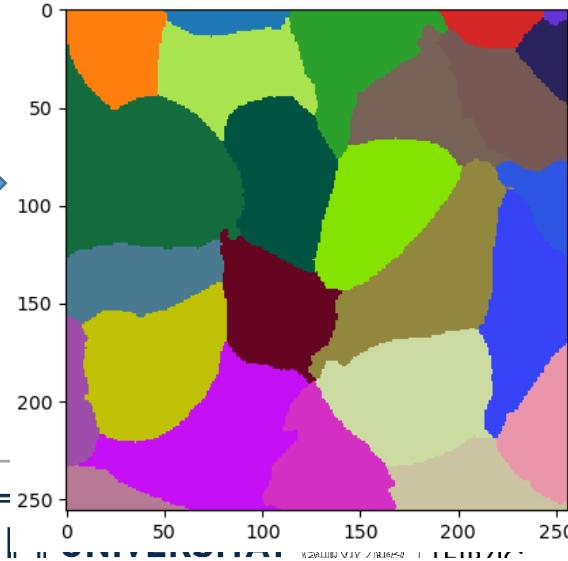


Watershed

- ... in Python practice

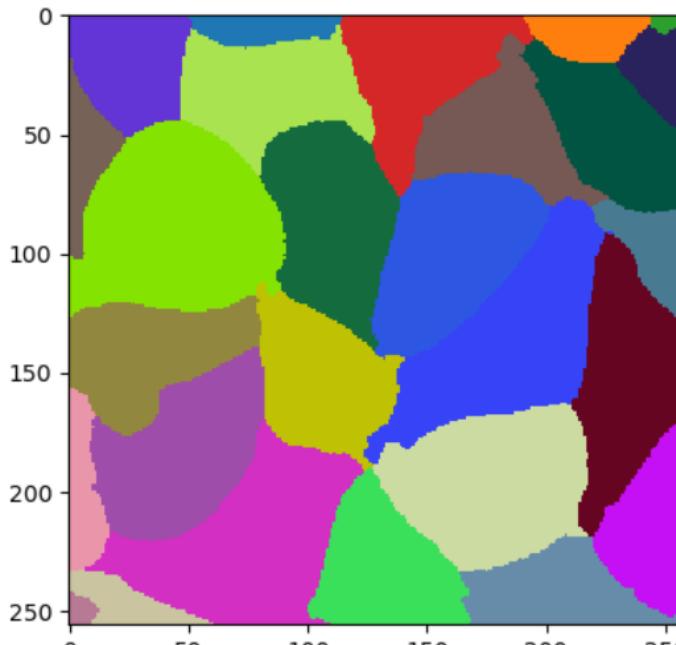


```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)  
labeled_cells
```

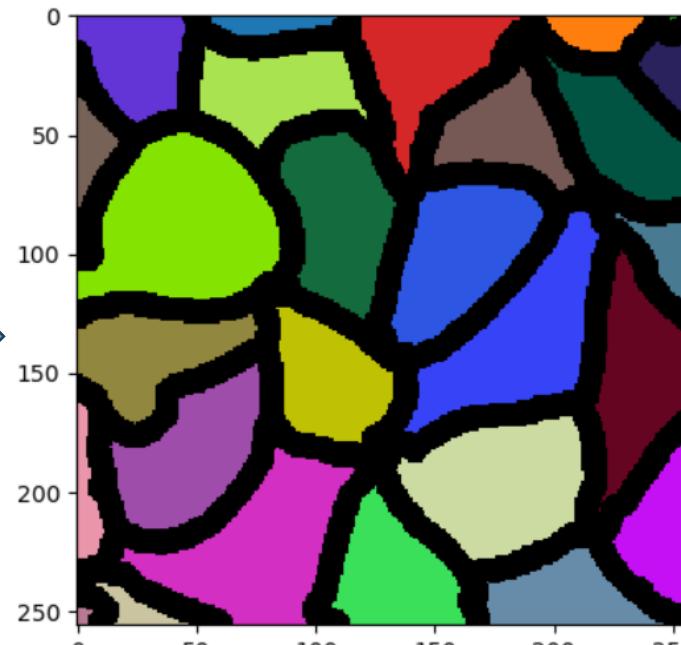
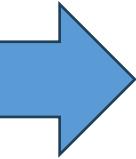


Label post-processing / morphological operations

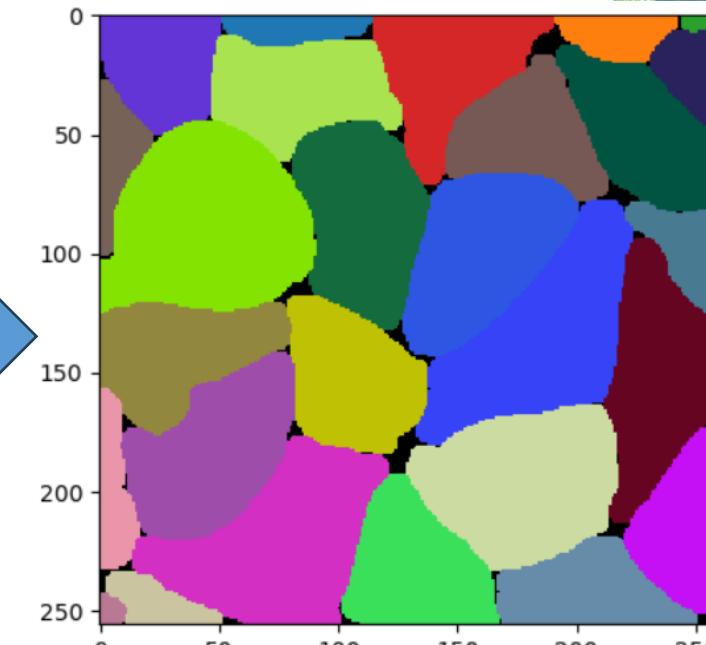
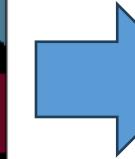
- ... similar to morphological operations on binary images



Original



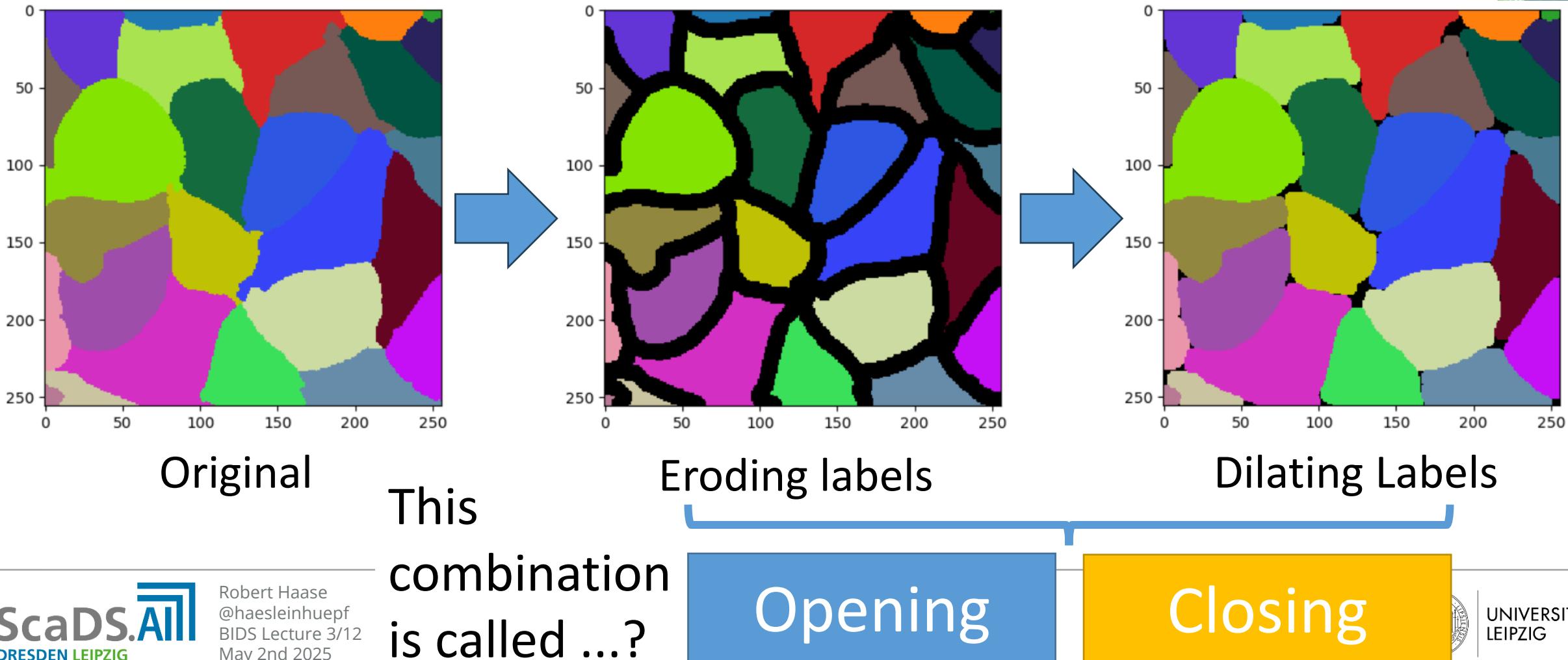
Eroding labels



Dilating Labels

Label post-processing / morphological operations

- ... similar to morphological operations on binary images



Label post-processing / morphological operations

- ... similar to morphological operations on binary images



Original



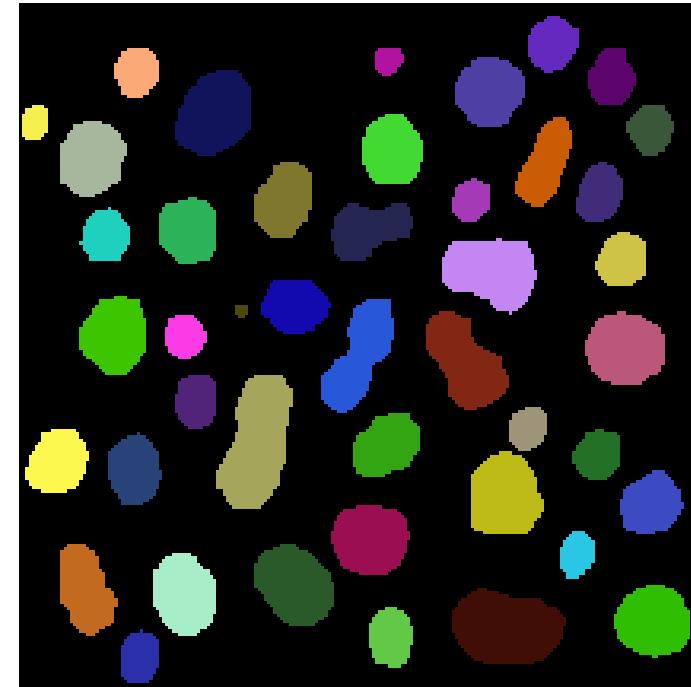
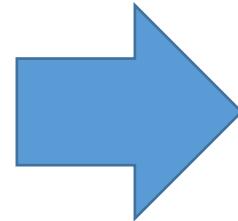
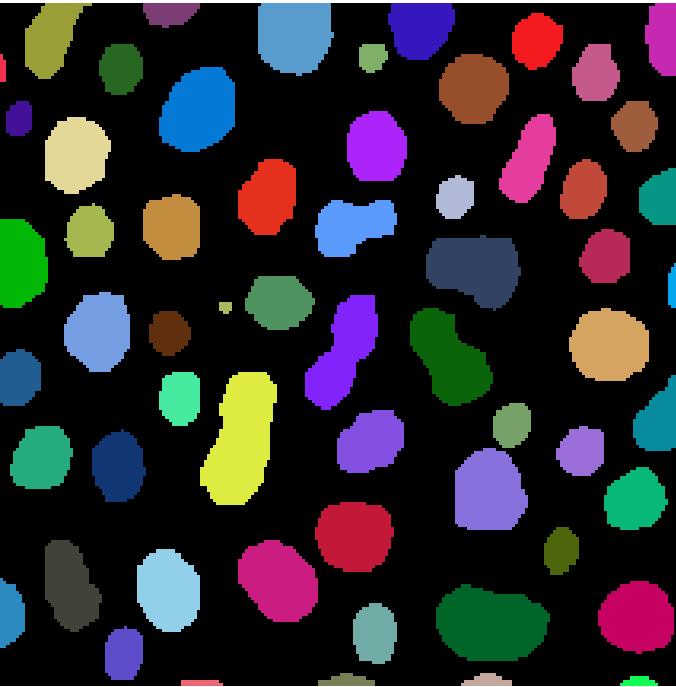
Opening labels



Smoothing Labels

Label post-processing / selections

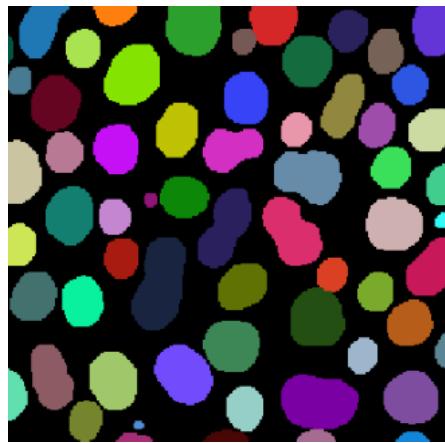
- Remove objects at the image border



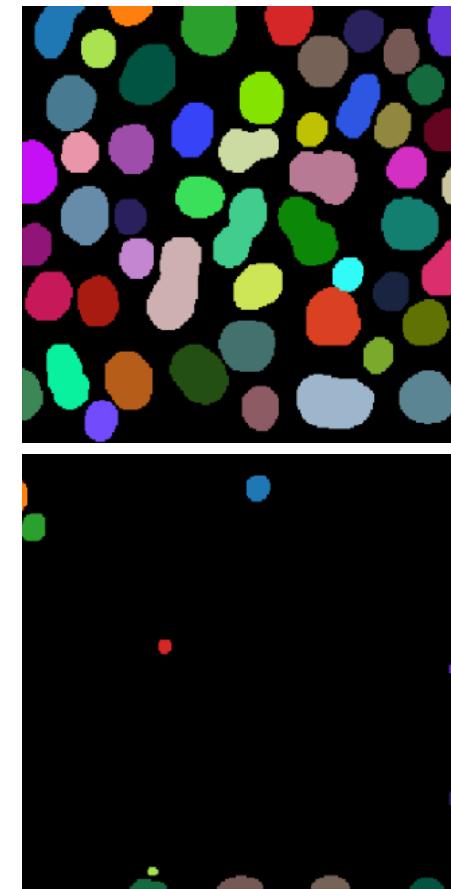
Quiz: Why is it a good idea to do this before measuring object properties?

Label post-processing / selections

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



Exclude small objects

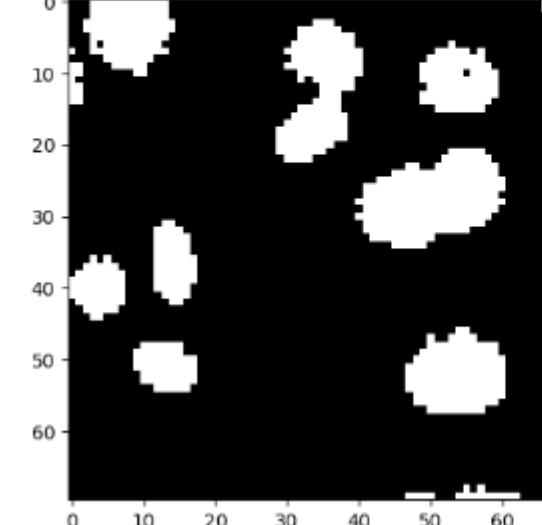


Exclude large objects

Pitfalls in Python

- Image results of some libraries do not need to be imshow'ed

```
binary_image = cle.threshold_otsu(image)
binary_image
```



`cle._image`

shape (70, 70)

dtype uint8

size 4.8 kB

min 0.0

max 1.0



Pitfalls in Python

- Some threshold functions return numbers, others return images.

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

```
threshold
```

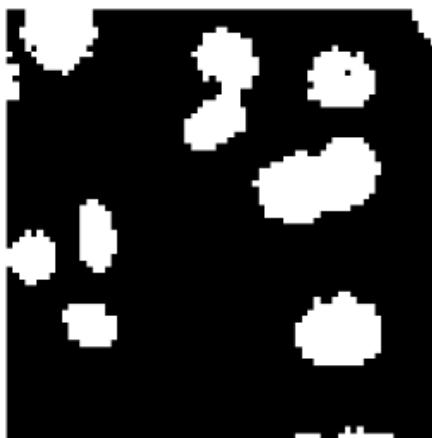
```
77
```

```
binary_image = image > threshold
```

```
stackview.insight(binary_image)
```

```
c:\structure\code\stackview\stackview\_static_view.py:101: RuntimeWarning: Converting input from bool to <class 'numpy.uint8'> for compatibility.
```

```
h, _ = np.histogram(self.obj, bins=num_bins)
```



shape (70, 70)

dtype bool

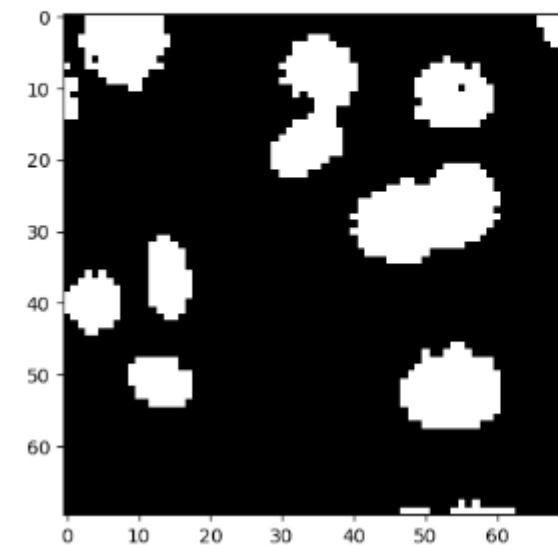
size 4.8 kB

min False

max True

```
binary_image = cle.threshold_otsu(image)
```

```
binary_image
```



cle._image

shape (70, 70)

dtype uint8

size 4.8 kB

min 0.0

max 1.0

Pitfalls in Python

```
image = imread("data/mitosis_mod.tif")

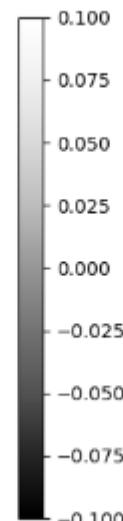
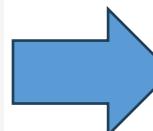
# determine a threshold
threshold = filters.threshold_otsu(image)

# blur the image to make segmentation outlines smoother
blurred_image = filters.gaussian(image, sigma=2)

# threshold the image
binary_image = blurred_image > threshold

stackview.insight(binary_image)
```

Quiz: Can anyone
spot the issue?



shape (70, 70)
dtype bool
size 4.8 kB
min False
max False



Background: gaussian() normalizes the intensity range.
Mathematical nonsense, but default unfortunately.

- set `preserve_range=True` or
- Determine the **threshold** on the same image where you also apply the threshold.

`image.min(), image.max(), image.dtype`

(8, 255, `dtype('uint8')`)

`blurred_image.min(), blurred_image.max(), blurred_image.dtype`

(0.06732543055888668, 0.7405626105467442, `dtype('float64')`)

Pitfalls in Python

- 3D image processing may require GPU-acceleration
- In case there are errors like PLATFORM_NOT_FOUND or BUILD_PROGRAM_FAILURE:

https://github.com/clEsperanto/pyclesperanto_prototype/?tab=readme-ov-file#troubleshooting-graphics-cards-drivers

Interactive image processing using Napari

Robert Haase

Funded by



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.

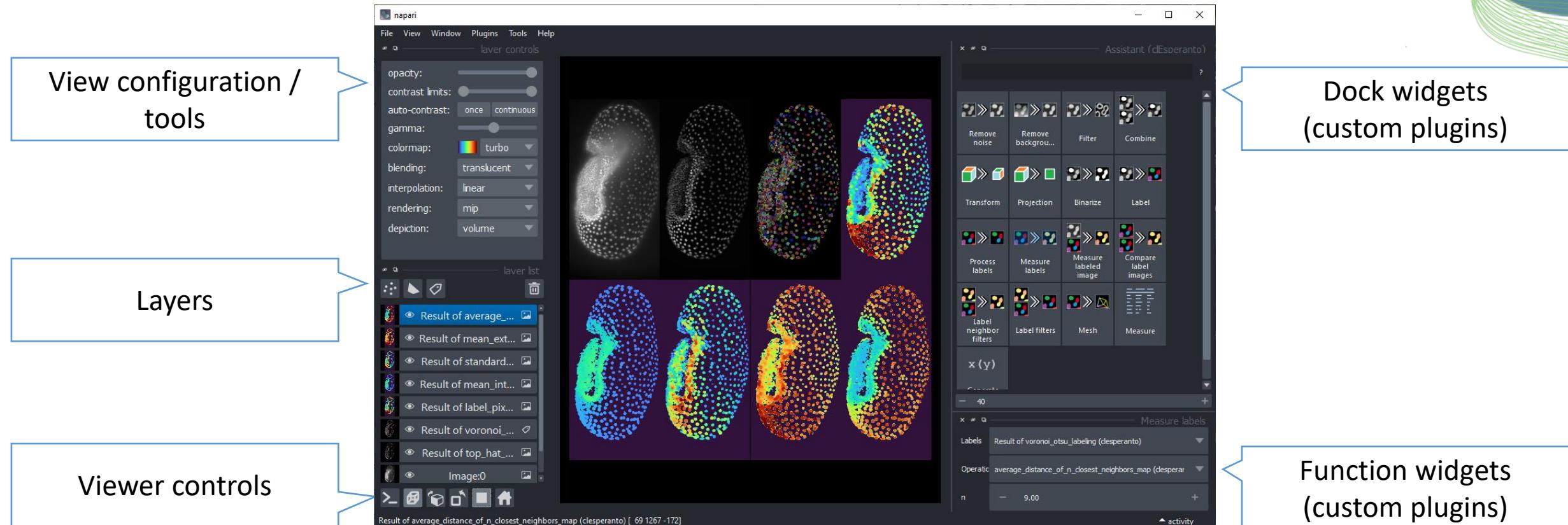
Chan
Zuckerberg
Initiative 

Napari

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

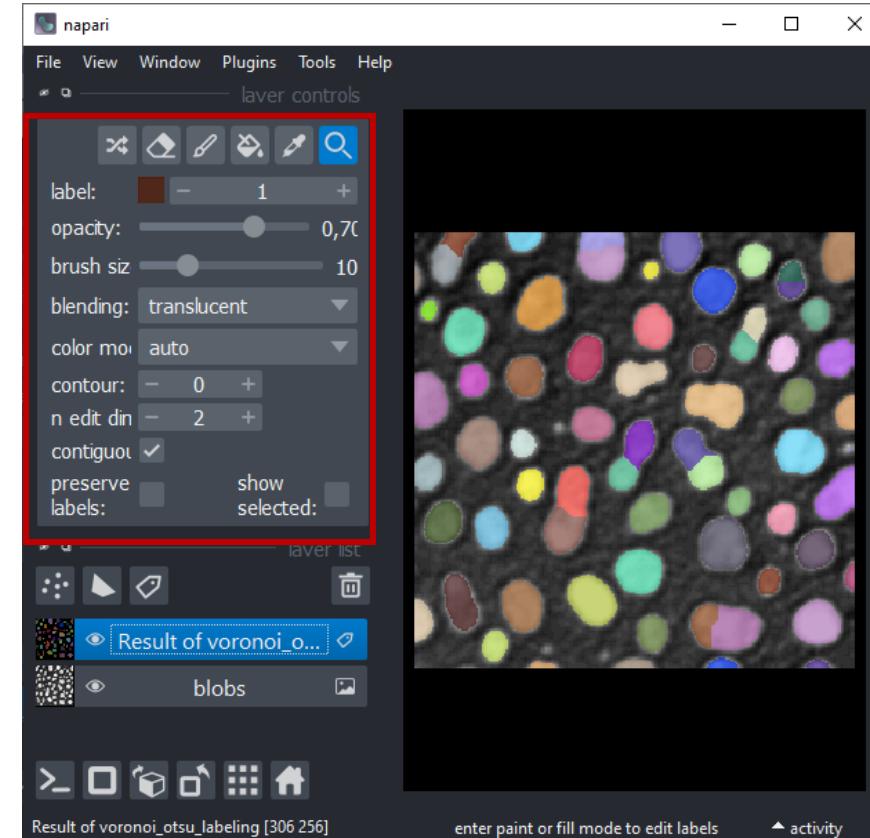
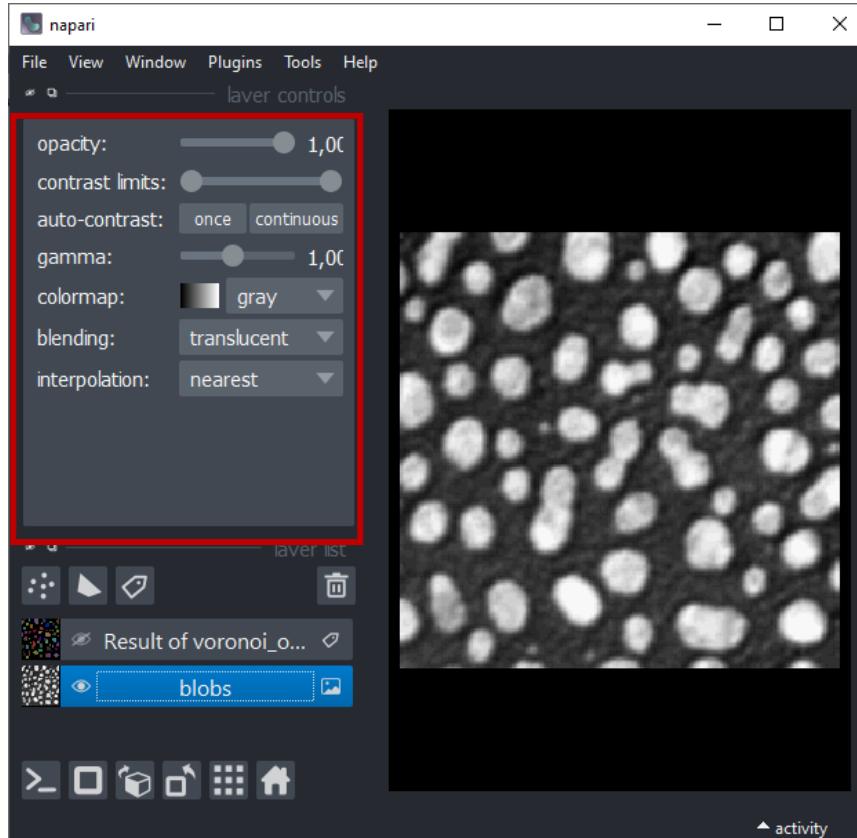


Napari – Graphical User Interface



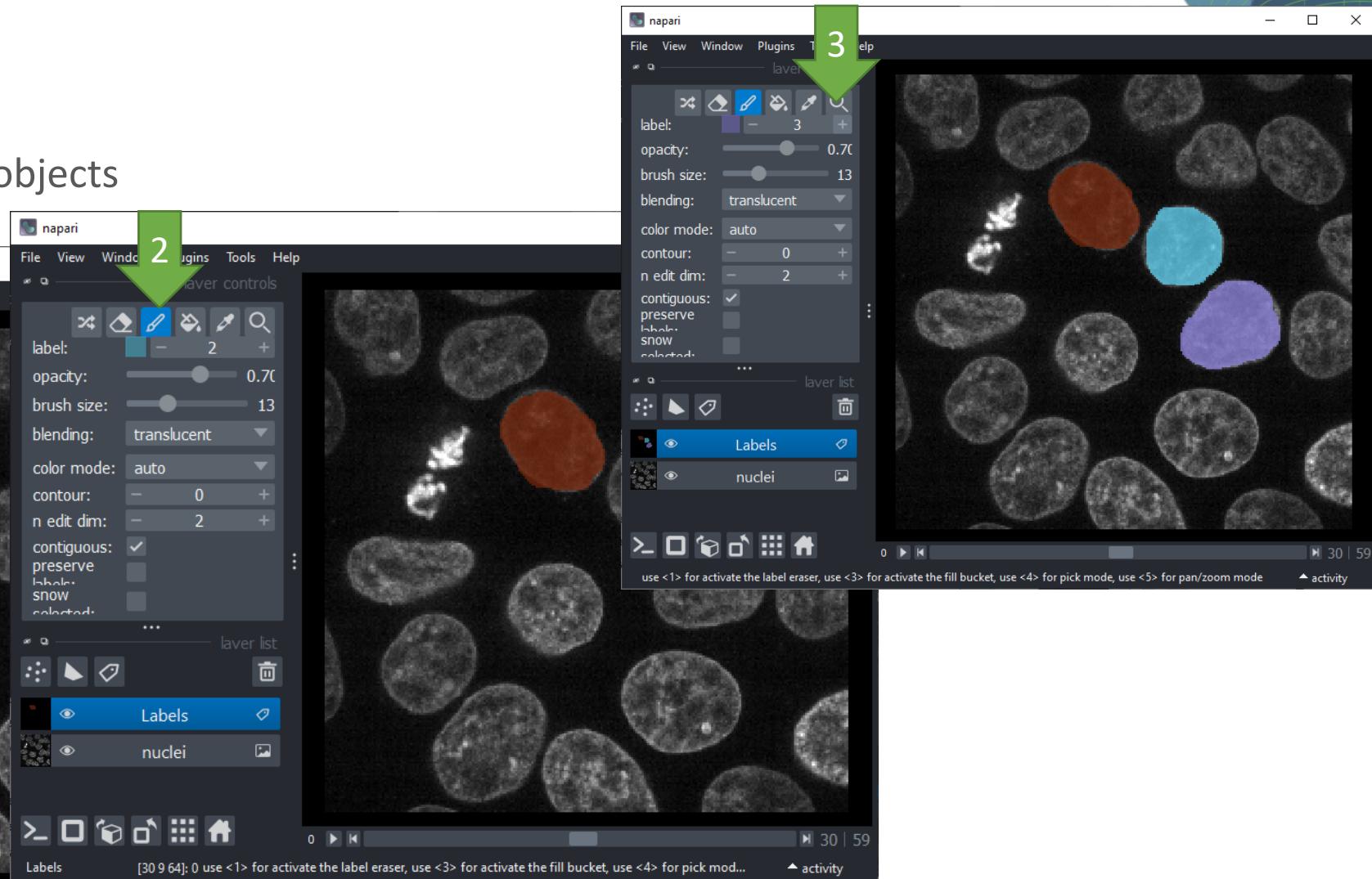
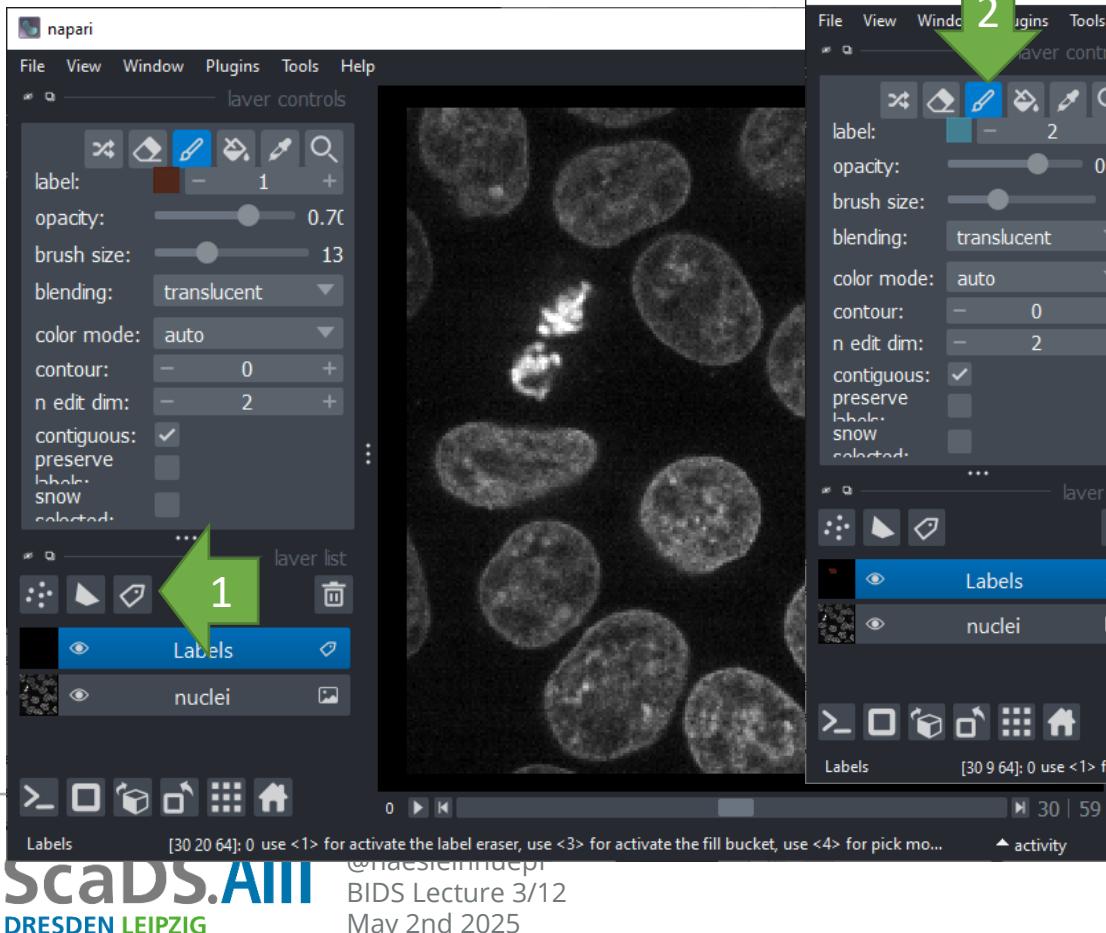
Napari – Graphical User Interface

- Context / data type dependent tools



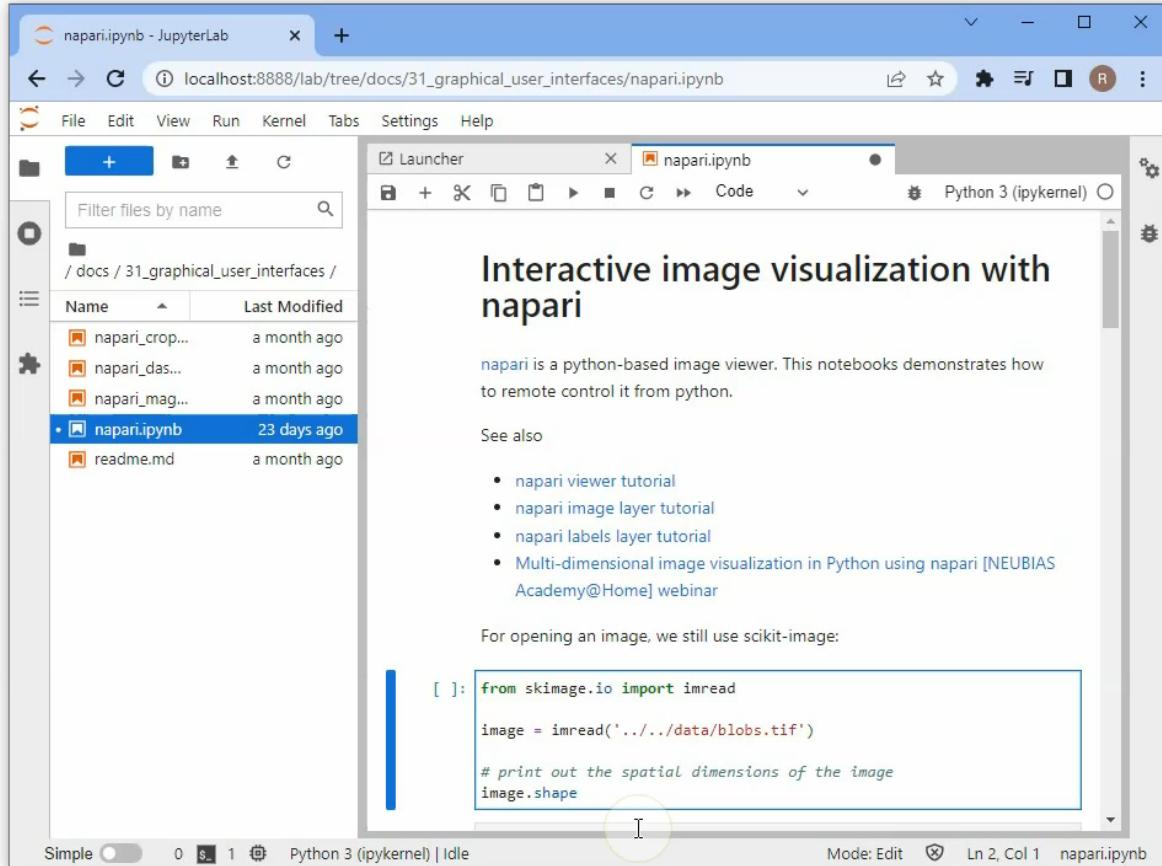
Manual annotation

1. Create a labels layer
2. Annotate first object
3. Increase label, annotate more objects



Napari – Python Scripting

- Mixing interactivity and reproducibility



Napari – Python Scripting

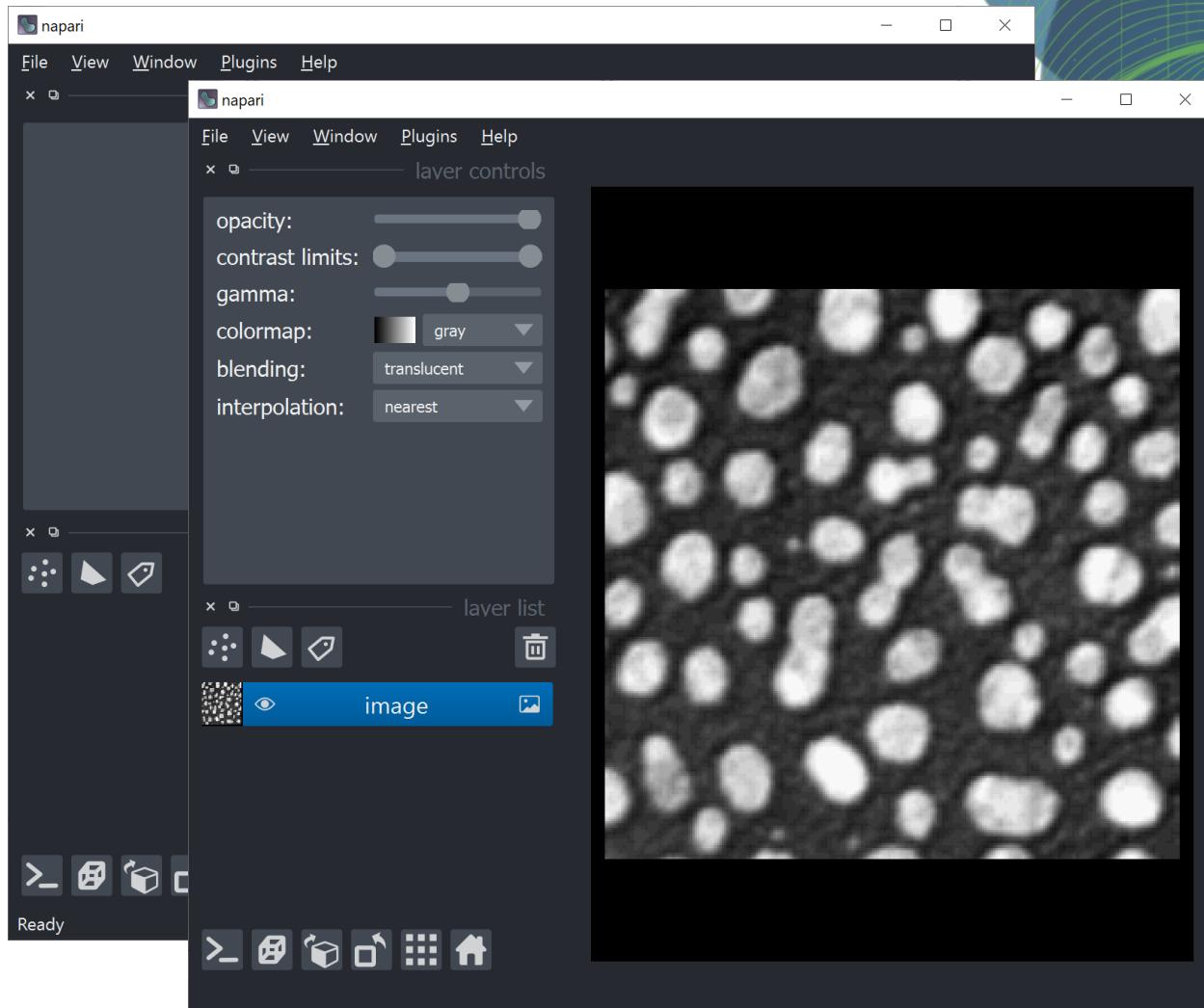
- Initialization

```
import napari
```

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

- Adding images

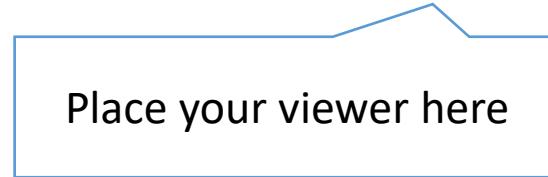
```
viewer.add_image(image)
```



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 URL bar indicates the notebook is running at `localhost:8889/notebooks/01_napari.ipynb`. The notebook has a Python 3 kernel and is set to "Not Trusted".

In [1]:

```
import napari
```

Create an empty viewer
viewer = napari.Viewer()

Start it
napari.run()

In [3]:

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

Out[3]: <Image layer 'image' at 0x1a72ea05580>

With this command, we can make a screenshot of napari and save it in our notebook.

In [6]:

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

Out[6]:

The viewer interface shows a grayscale image of cells. On the left, there is a control panel with sliders for opacity, contrast limits, gamma, colormap (set to gray), blending (set to translucent), and interpolation (set to nearest). The main area displays the image.

Napari – Python Scripting

- Removing layers

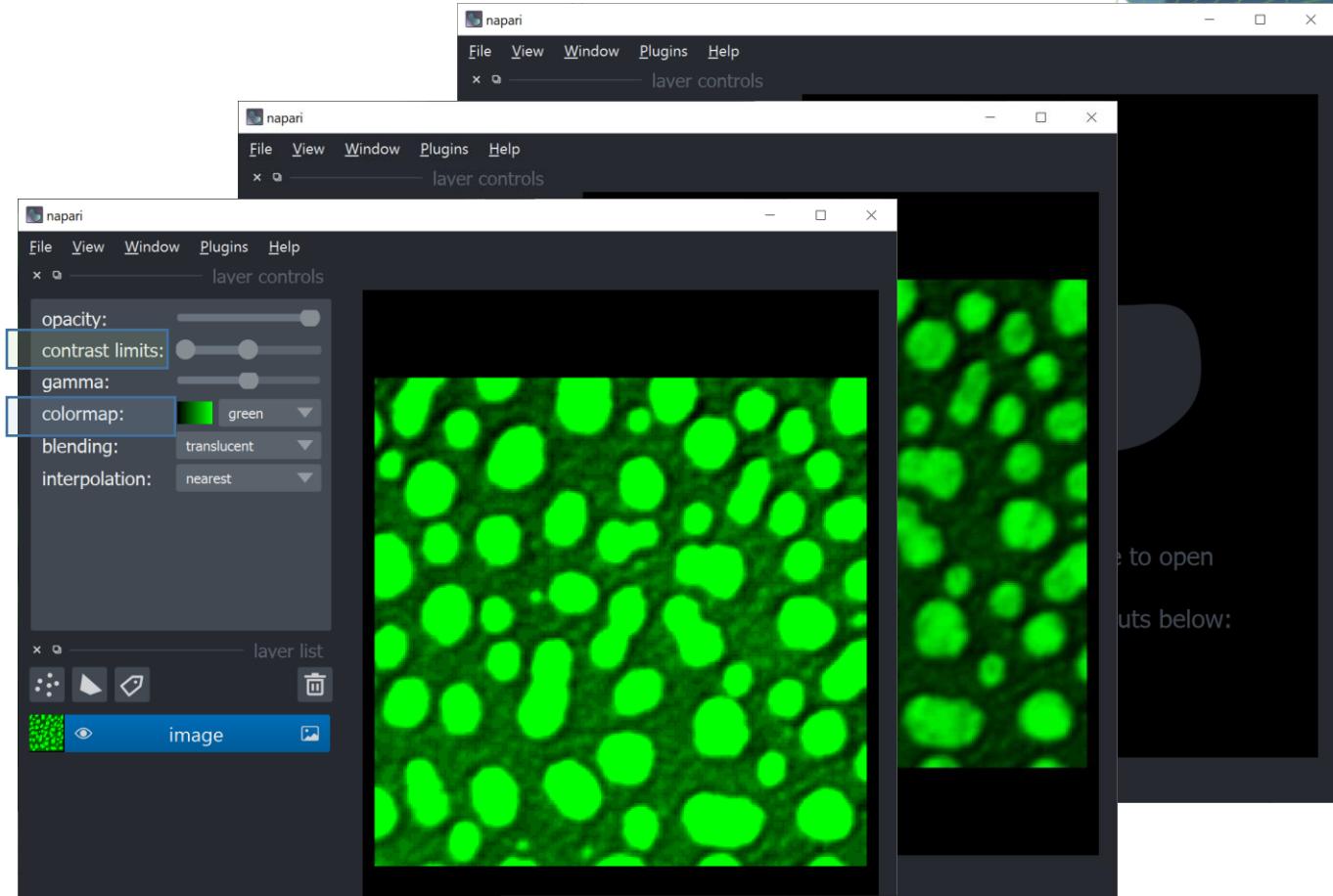
```
for l in viewer.layers:  
    viewer.layers.remove(l)
```

- Modify visualization while adding layers

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

- Modify layers after adding

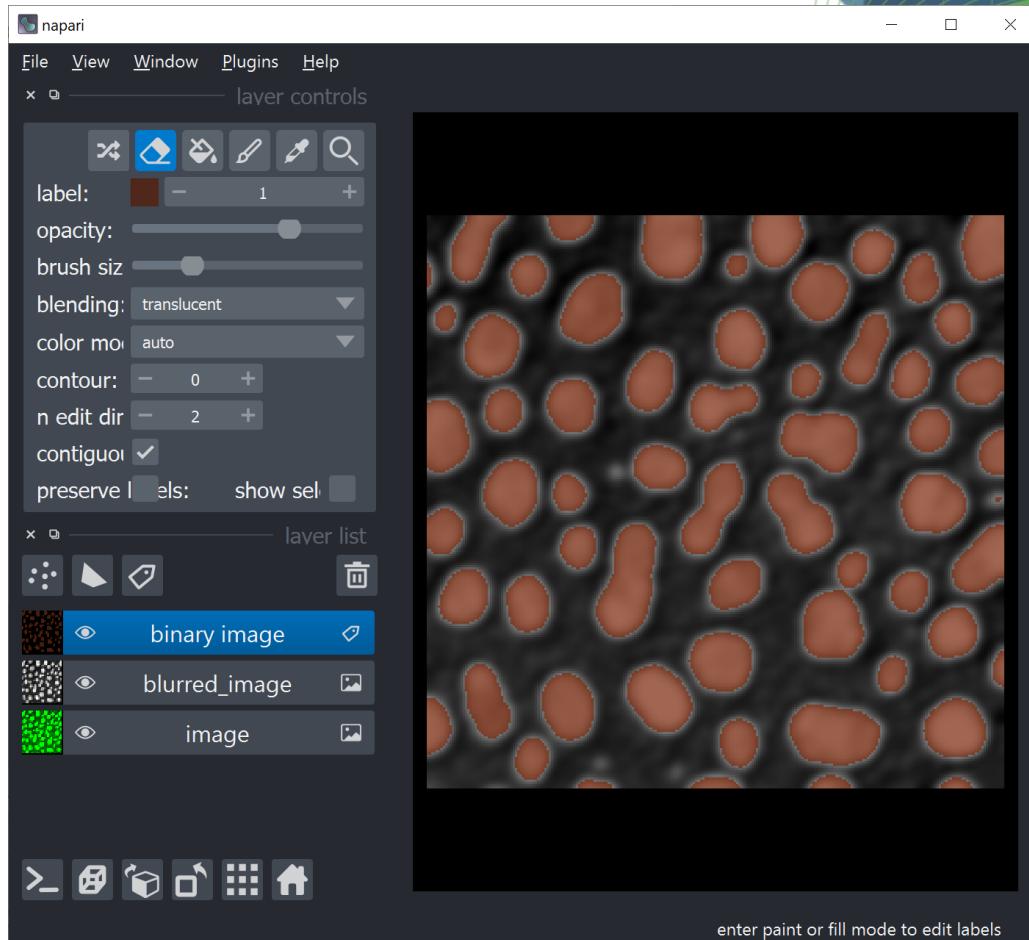
```
layer = viewer.add_image(image)  
layer.colormap = 'green'  
layer.contrast_limits = (0, 128)
```



Napari – Python Scripting

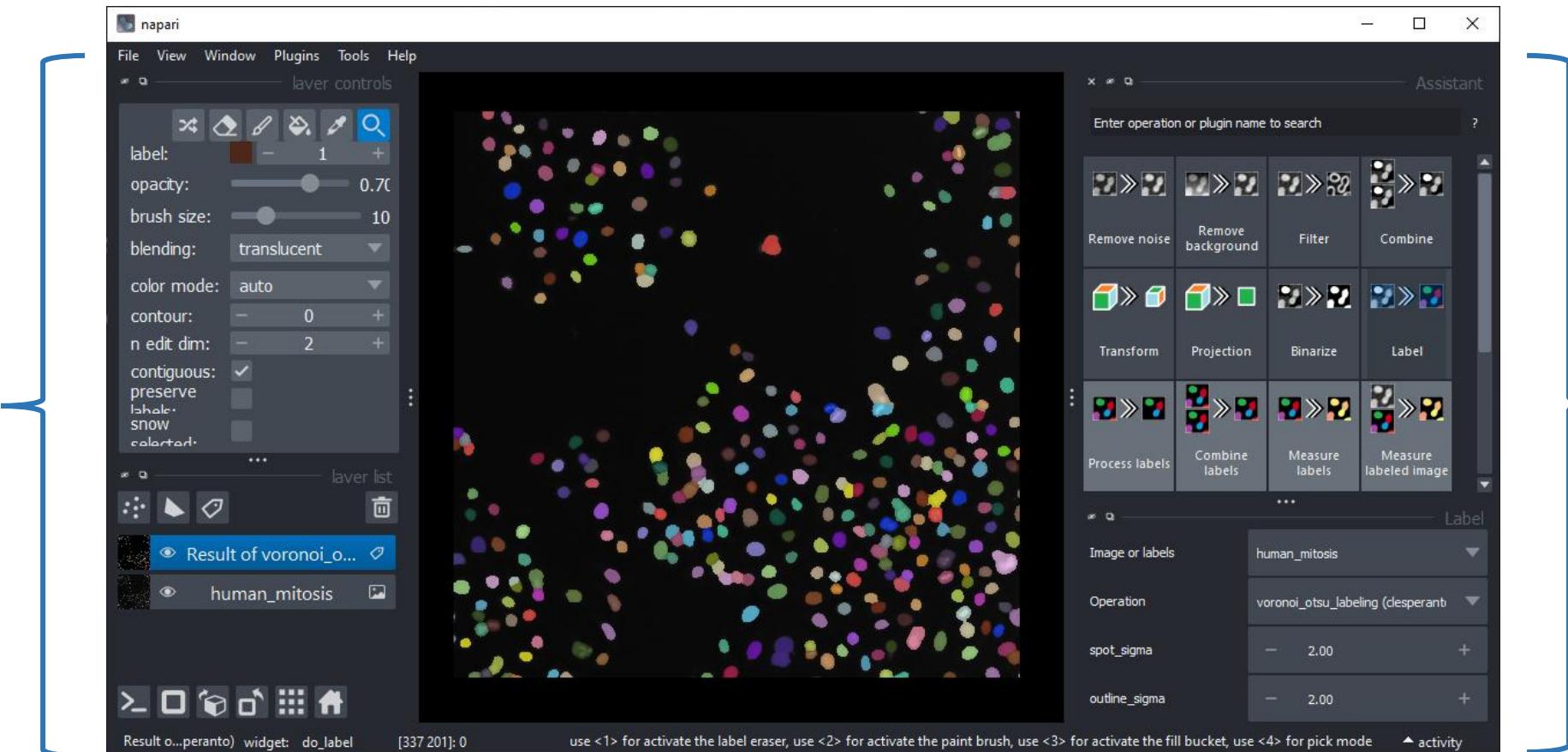
- Binary images and **label** images visualized as label layers

```
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)
```



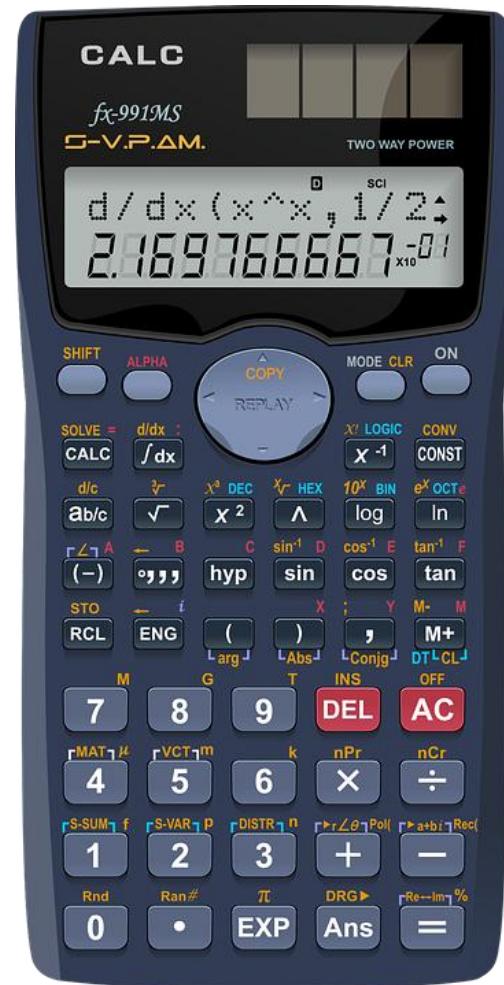
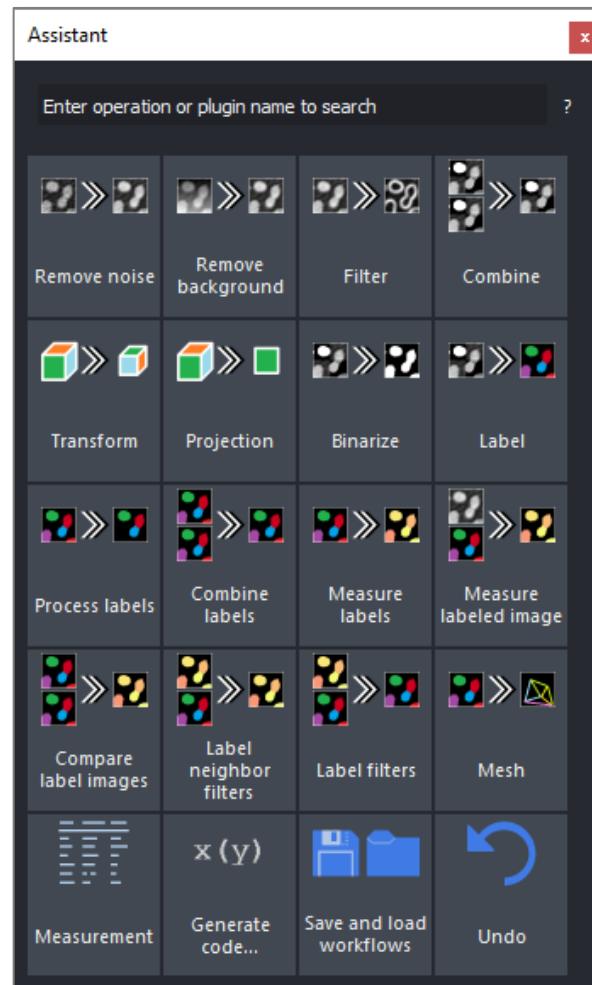
The Napari Assistant

- Tools > Utilities > Assistant (na)



The Napari Assistant

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

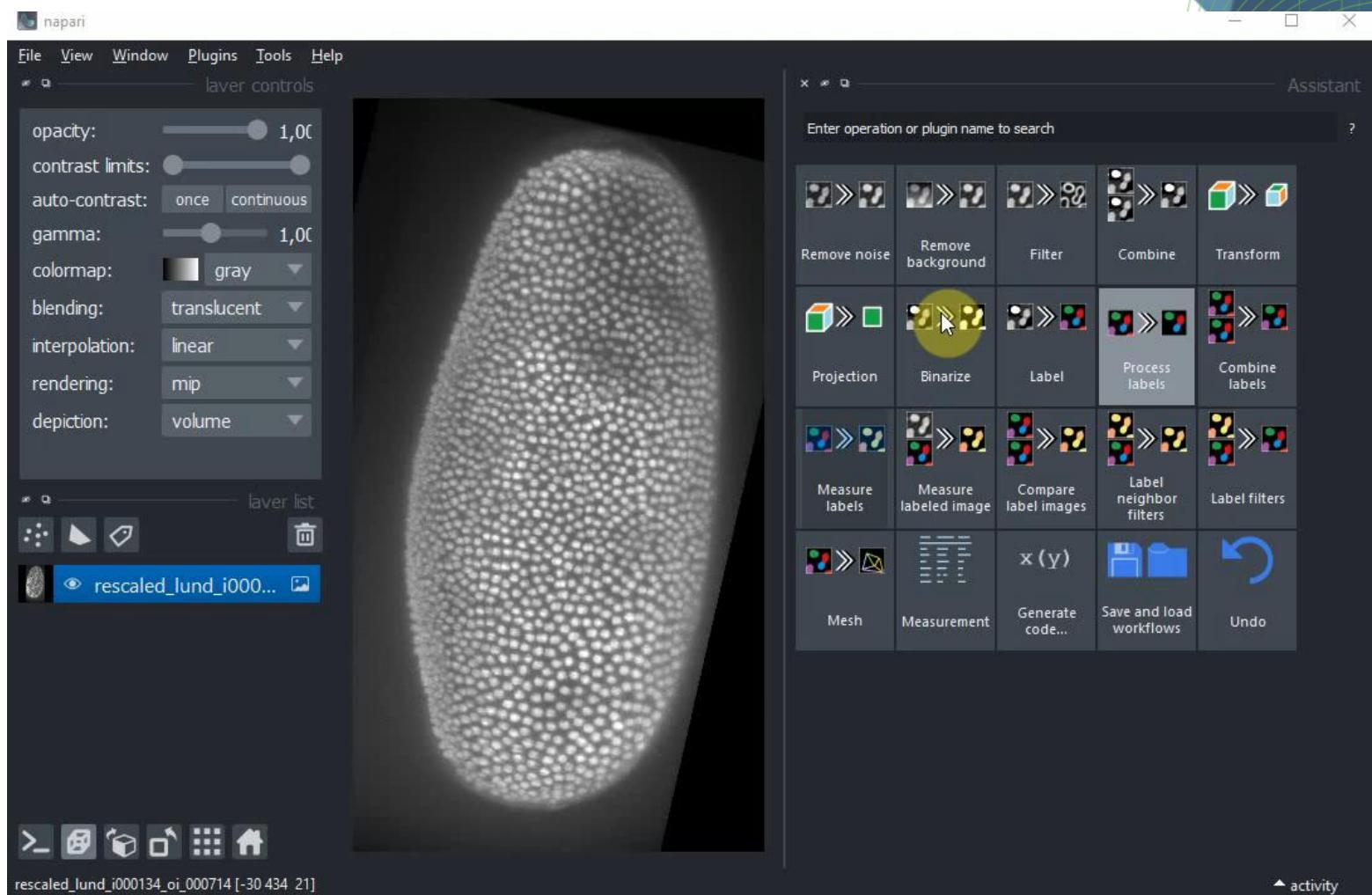


The Napari Assistant

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

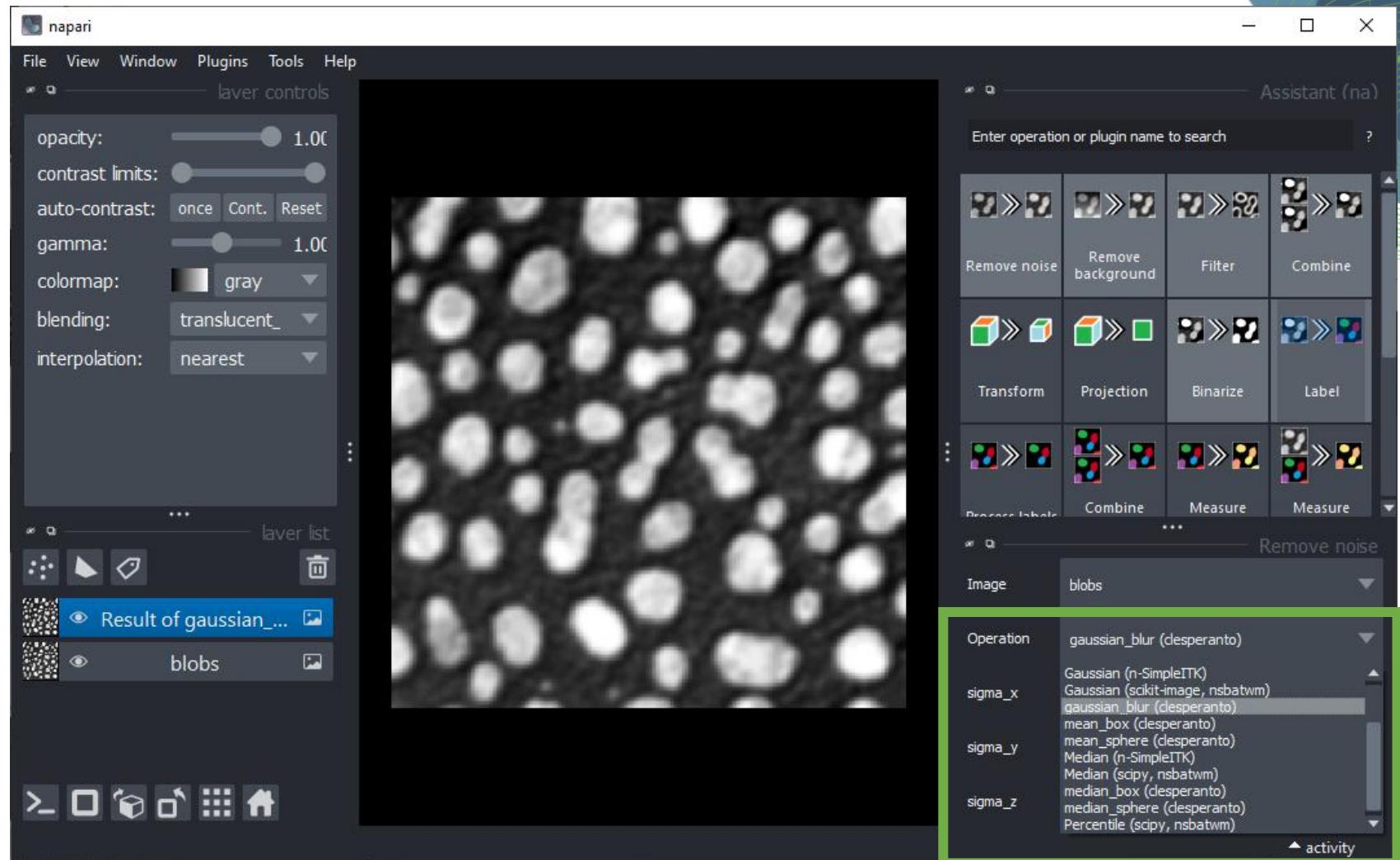


Ryan Savill ()
@RyanSavill4



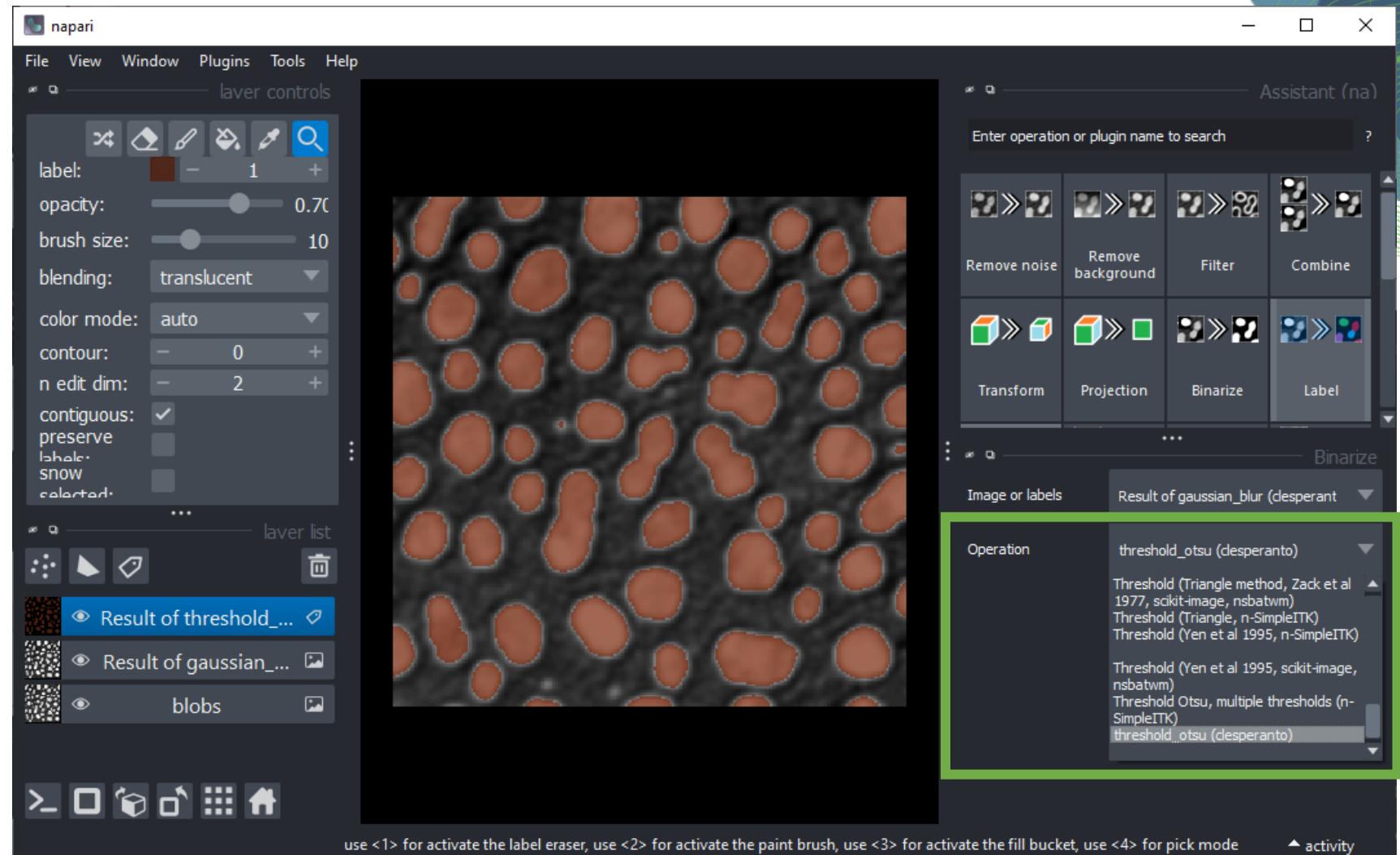
Workflow building

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



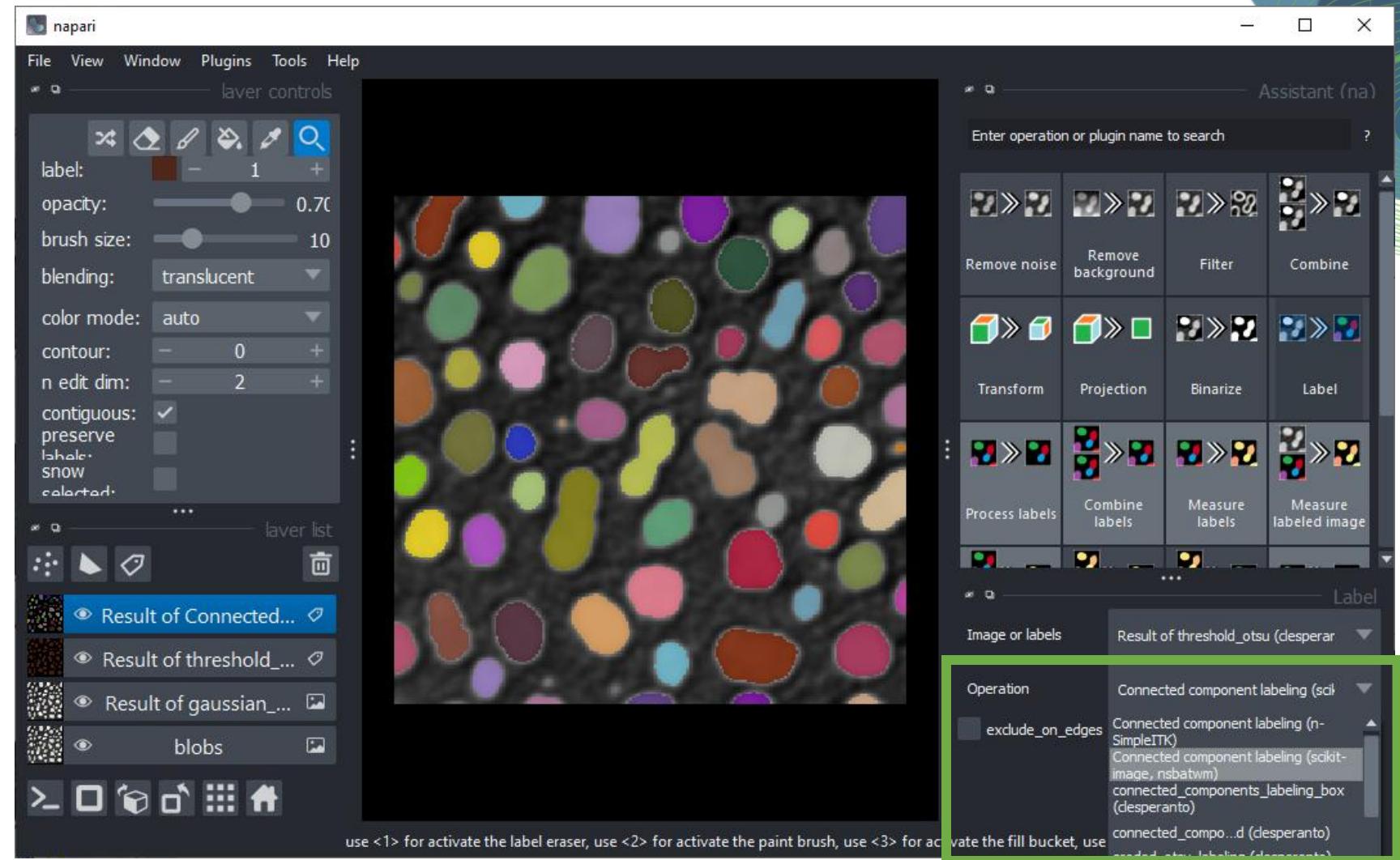
Workflow building

- Try different binarization algorithms



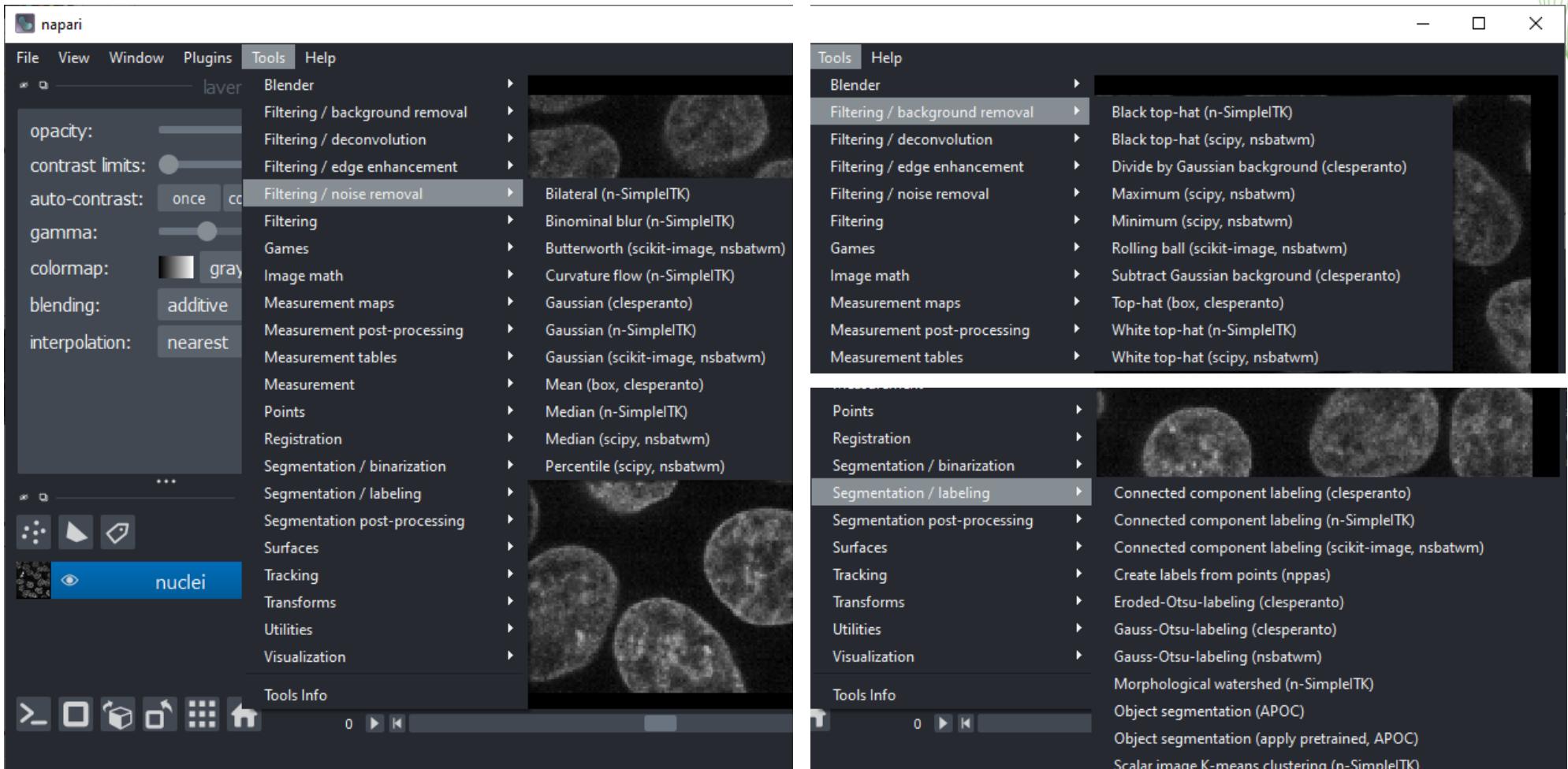
Workflow building

- Multiple labeling algorithms available



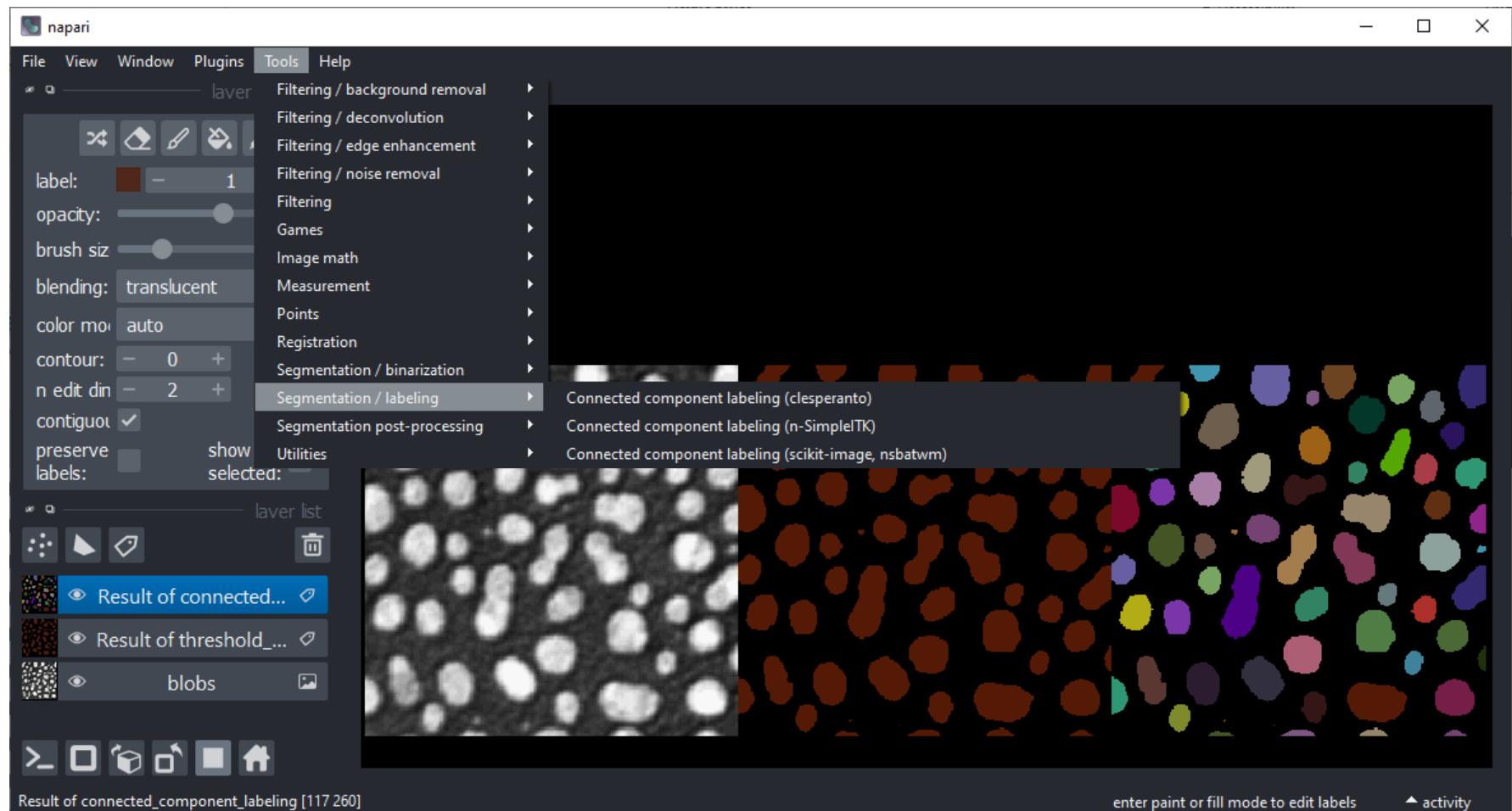
The Tools menu

- Organized in categories



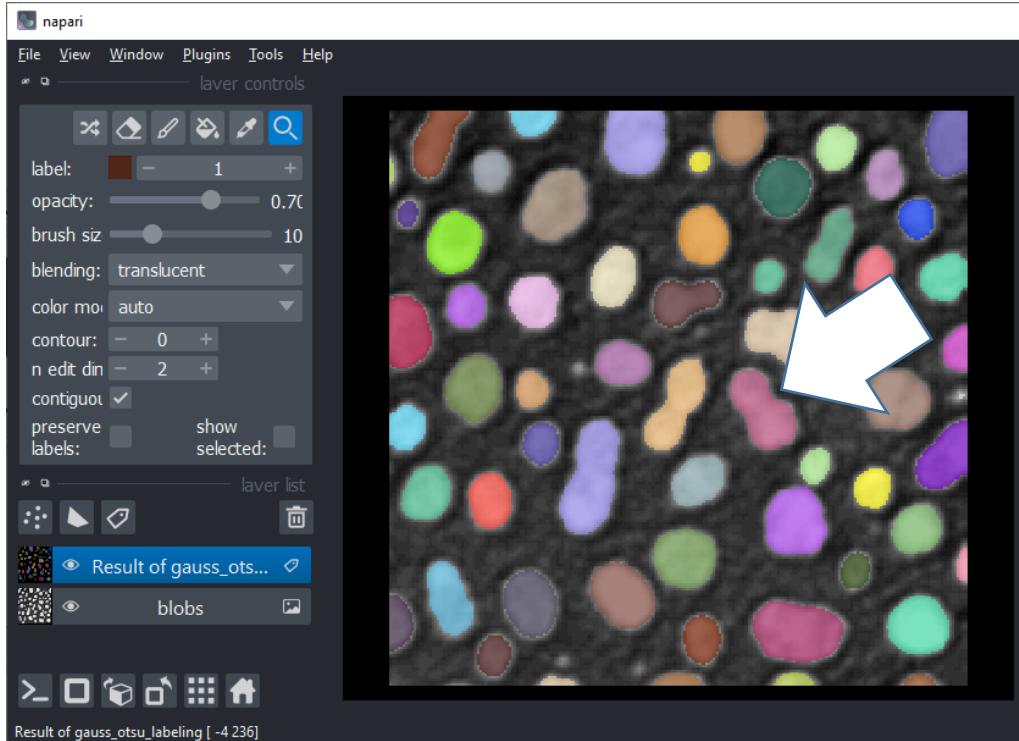
Workflow building

Also check out the Tools > Segmentation / labeling menu

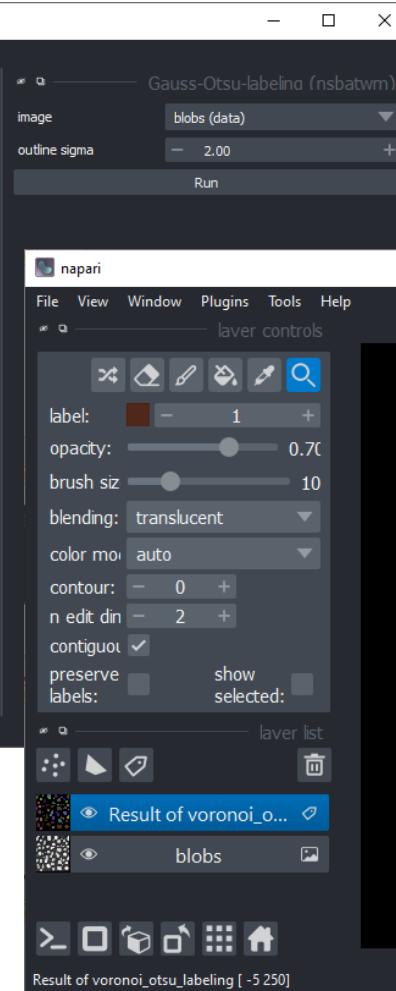


Short-cuts: Voronoi-Otsu-Labeling

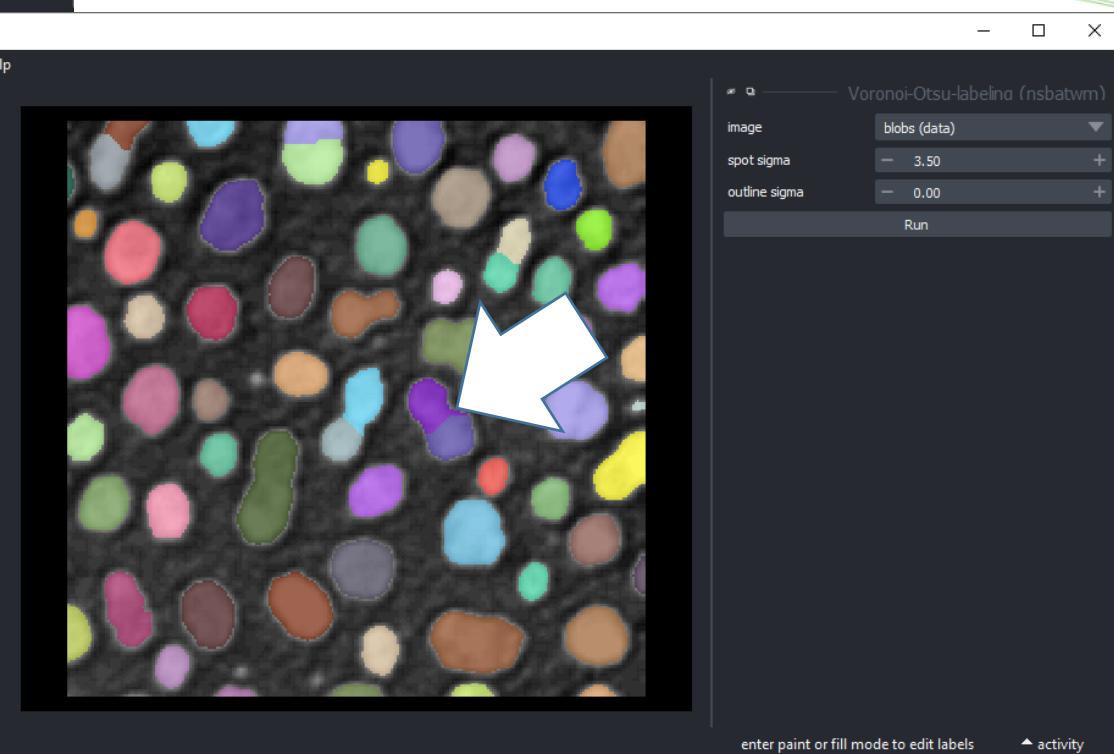
Also check out the Tools > Segmentation / labeling menu



Gauss-Otsu-Labeling

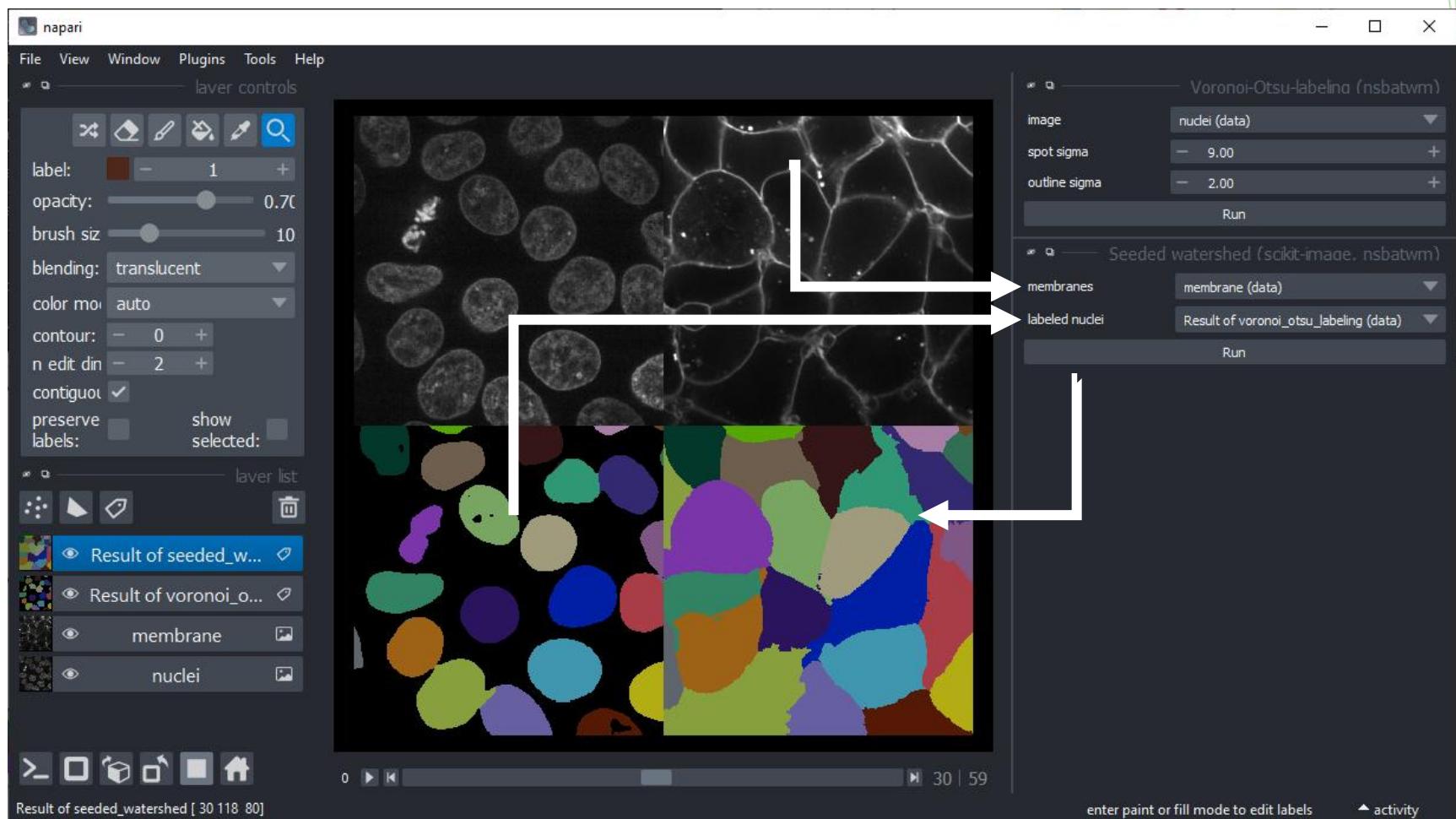


Voronoi-Otsu-Labeling



Watershed

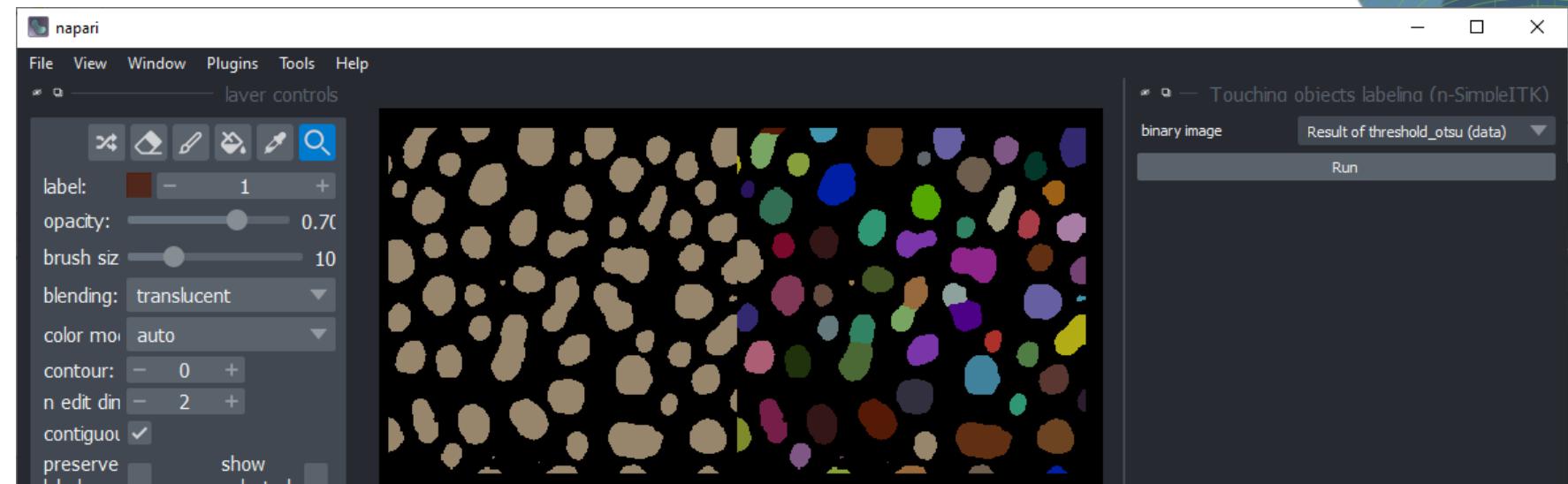
Also check out the Tools > Segmentation / labeling menu



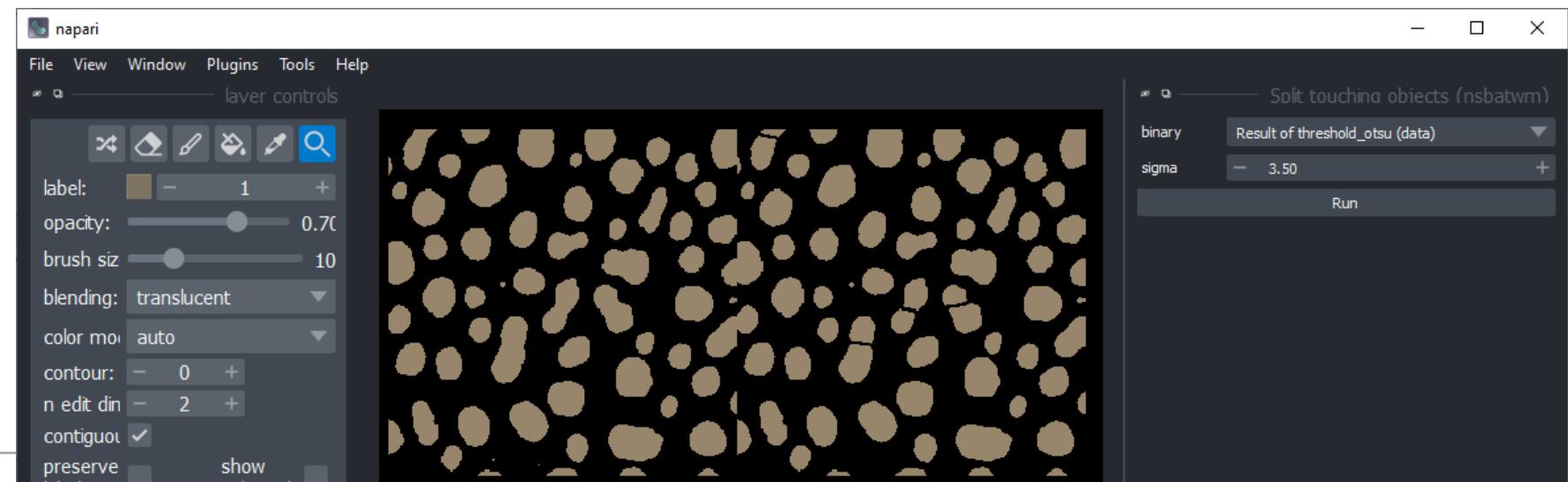
Watershed

- From binary images

Tools > Segmentation / labeling >
Label touching objects

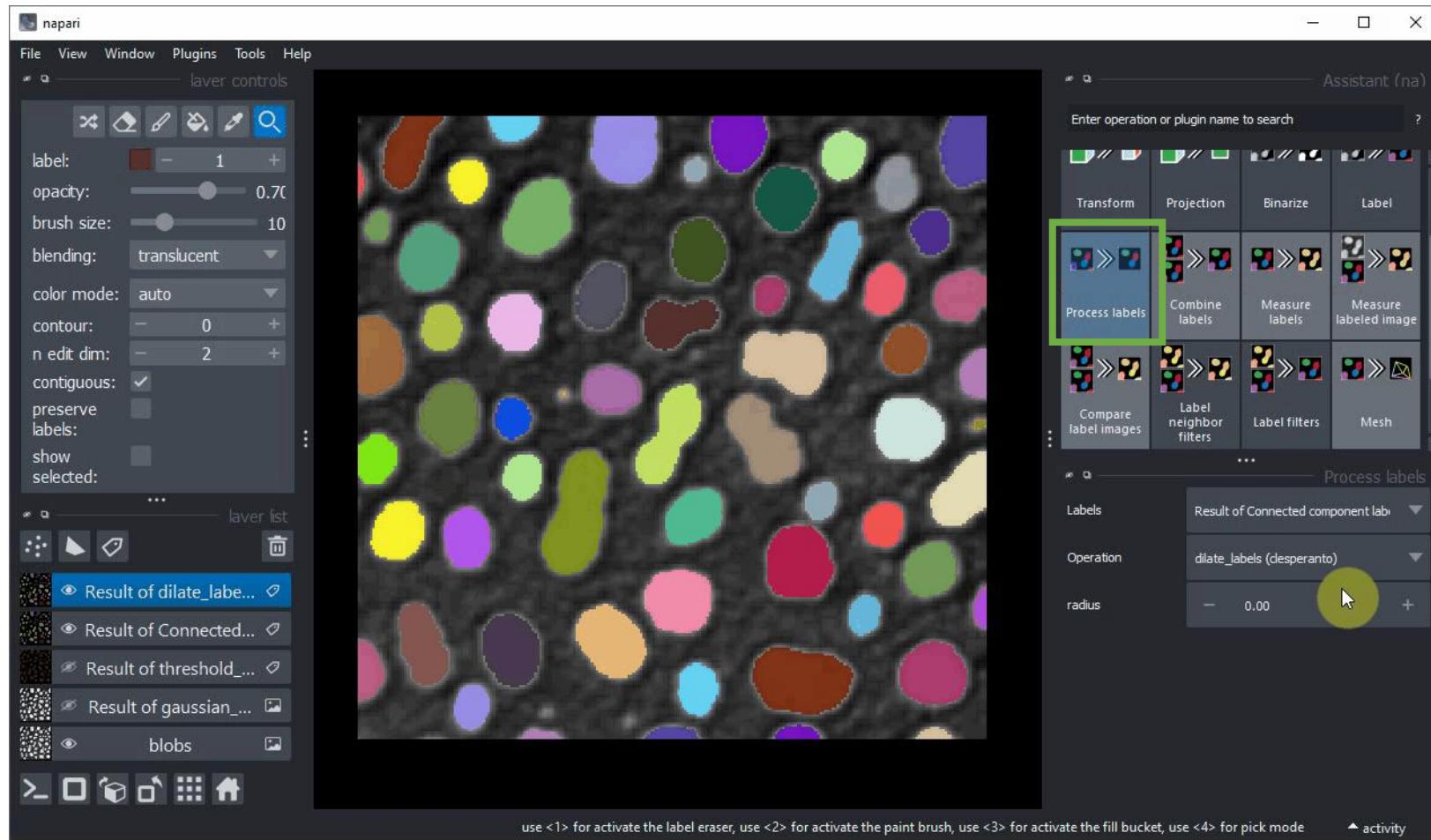


Tools > Segmentation post-processing >
Split touching objects
(Similar to ImageJ's Watershed)



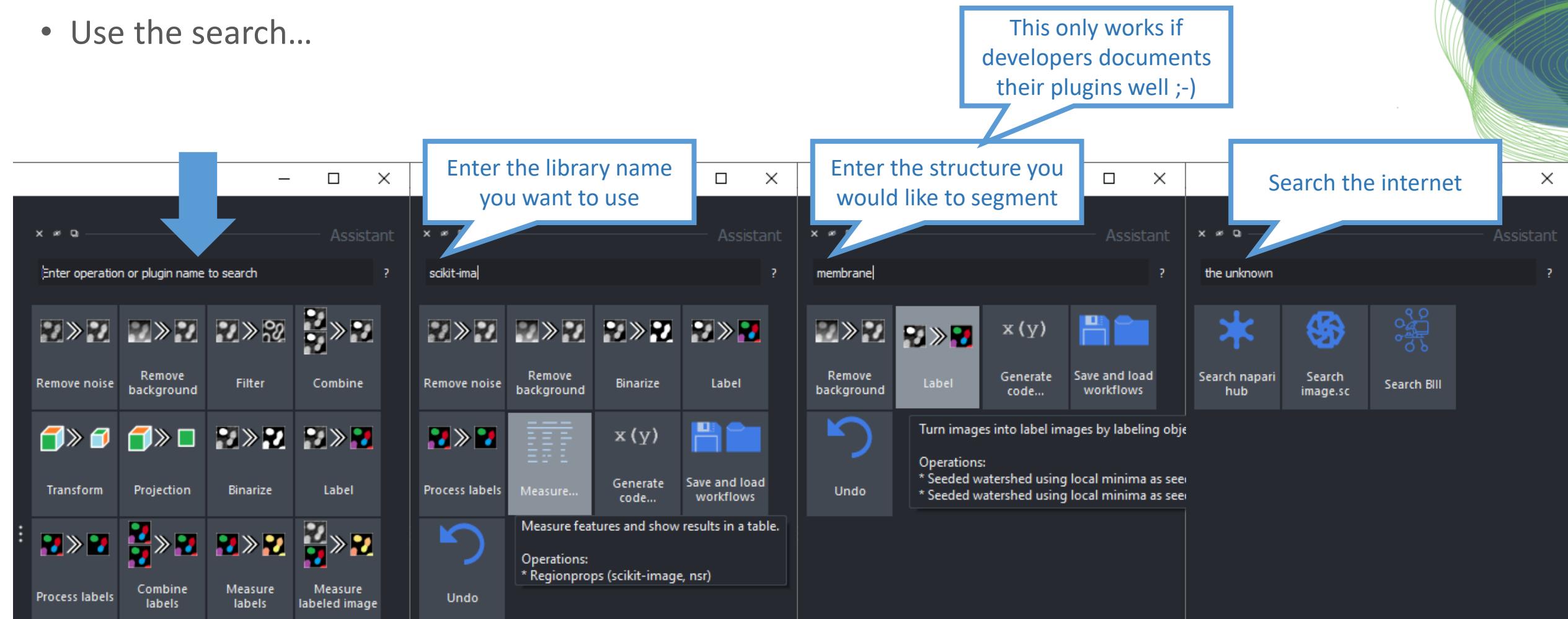
Label erosion, dilation, opening, closing, ...

- In Napari Assistant: Process labels

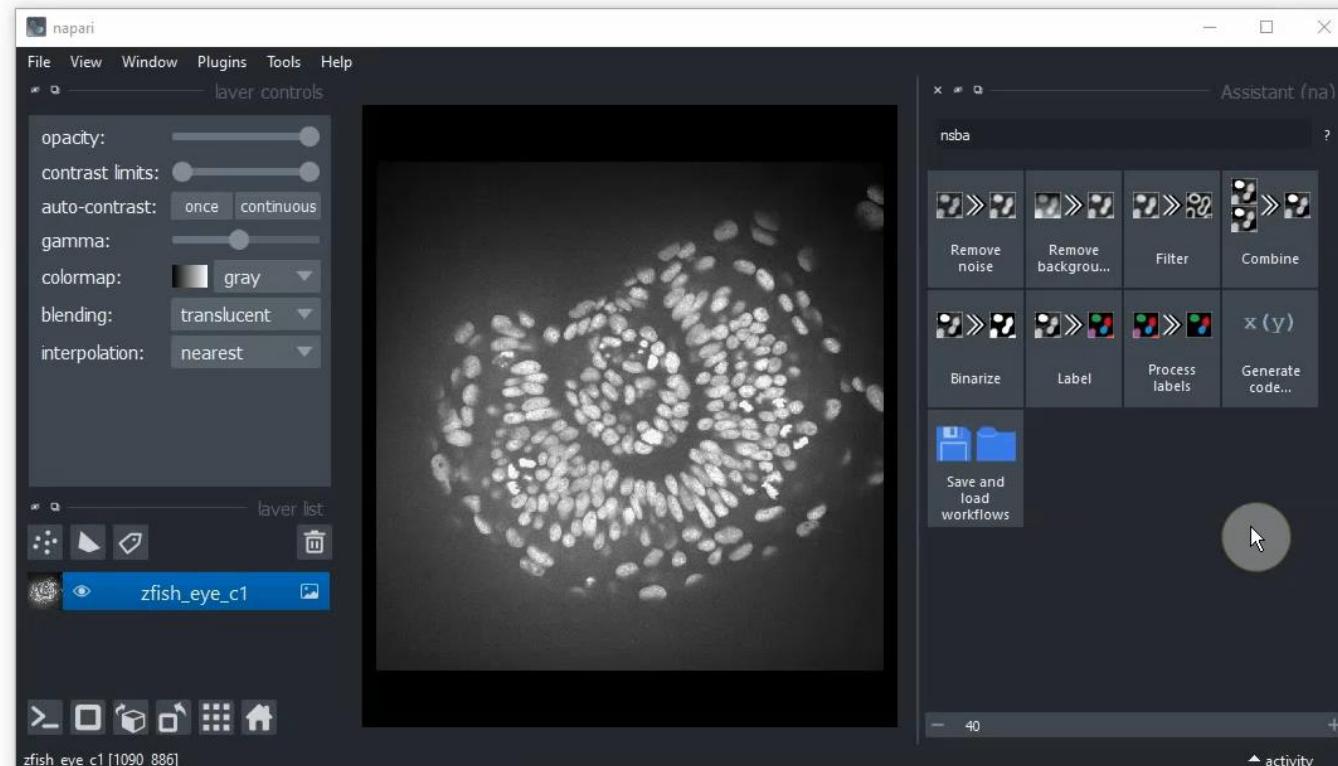


Browse operations

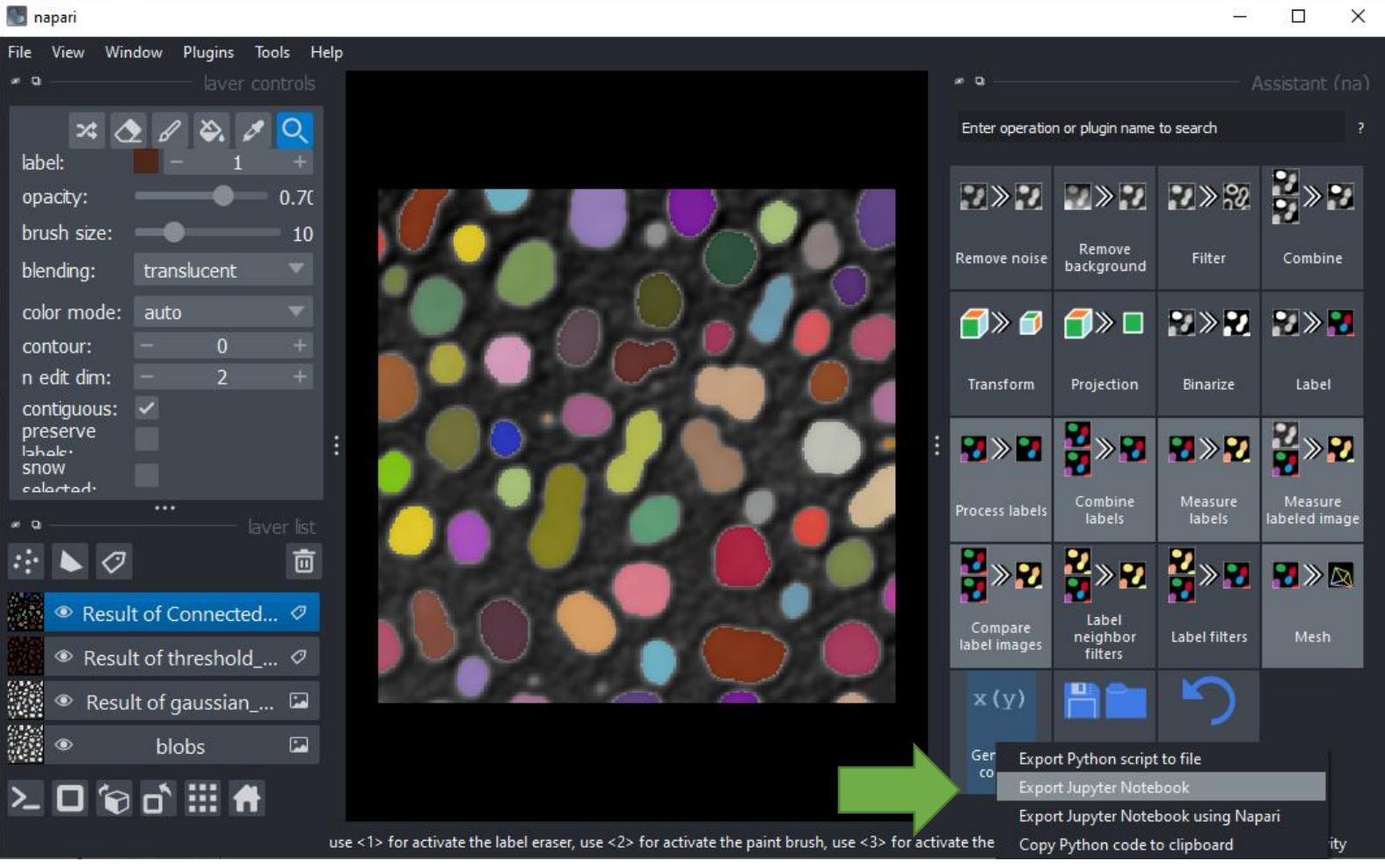
- Use the search...



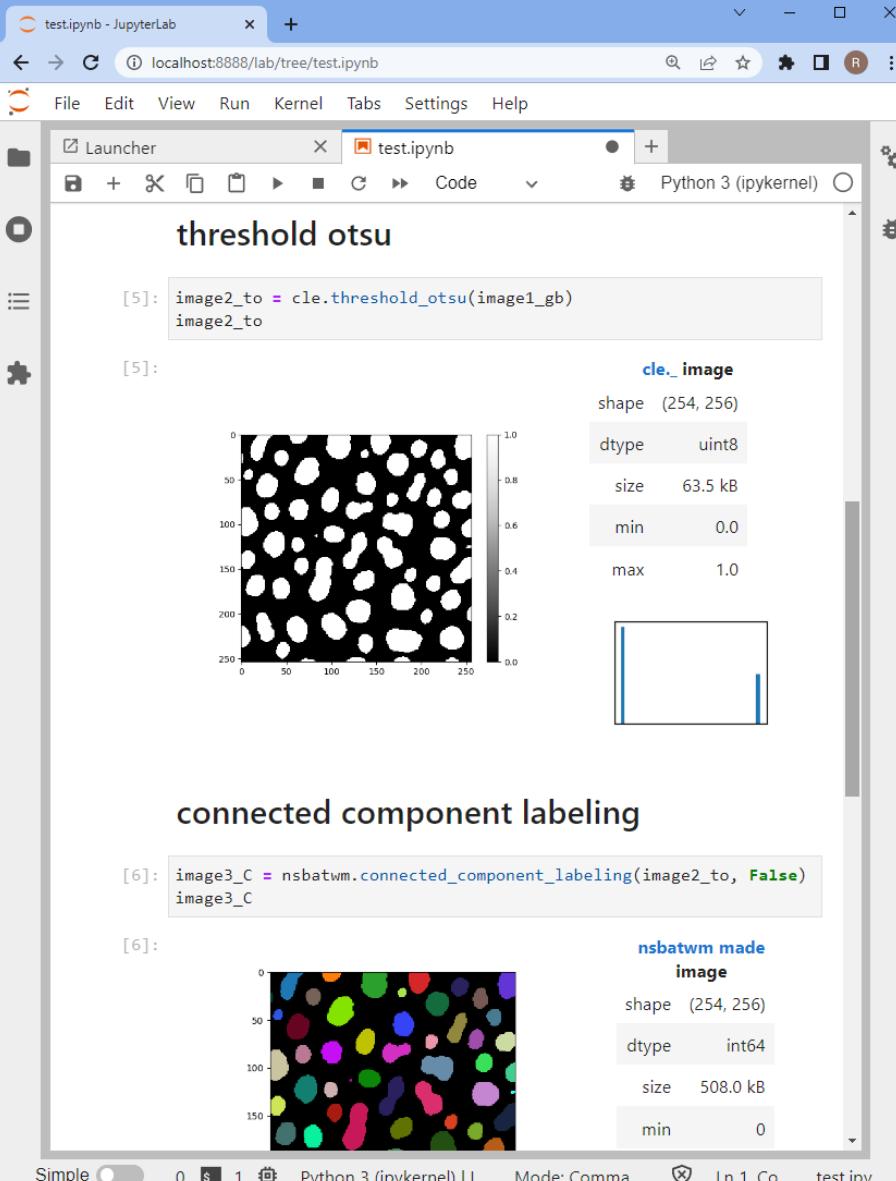
Export code to Jupyter Notebooks



Export code to Jupyter Notebooks



The screenshot shows the napari interface with the Assistant extension open. On the left, a segmented image of colored blobs is displayed. The Assistant panel on the right contains various operations like Remove noise, Remove background, Filter, and Combine. A green arrow points from the napari interface to the bottom right of the Assistant panel, where the "Export Jupyter Notebook" option is highlighted. Below the Assistant panel, a context menu lists "Export Python script to file", "Export Jupyter Notebook", "Export Jupyter Notebook using Napari", and "Copy Python code to clipboard".



The screenshot shows a Jupyter Notebook titled "test.ipynb" running in a browser. The code cell [5] contains the command `image2_to = cle.threshold_otsu(image1_gb)`. To the right, the variable `cle._image` is displayed with its properties: shape (254, 256), dtype uint8, size 63.5 kB, min 0.0, and max 1.0. Below it, a binary mask image is shown. The code cell [6] contains `image3_C = nsbatwm.connected_component_labeling(image2_to, False)`. To the right, the variable `nsbatwm.made_image` is displayed with its properties: shape (254, 256), dtype int64, size 508.0 kB, and min 0. Below it, the original colored blobs image is shown.

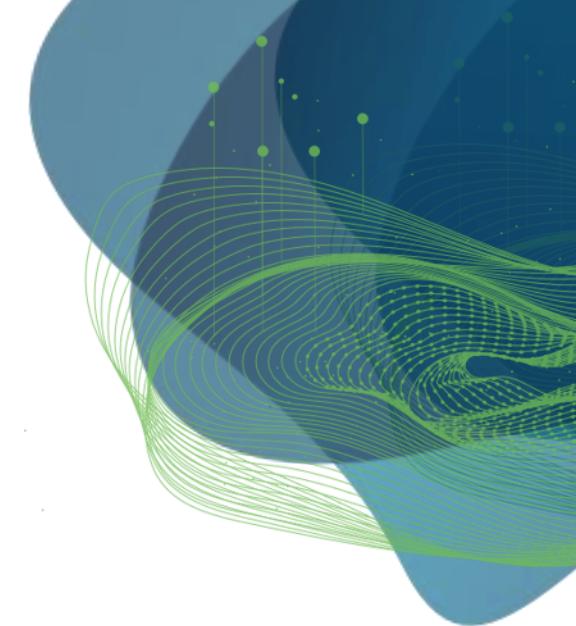
Robert Haase
@haesleinhuepf
BIDS Lecture 4/14
April 23rd 2024

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



DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS
AND ARTIFICIAL INTELLIGENCE



Exercises

Robert Haase

Image segmentation exercises

- Try out segmentation algorithms and apply them to other datasets

The screenshot shows a JupyterLab interface with a notebook titled "11_voronoi_otsu_labeling.ipynb". In the left sidebar, there's a file browser with a list of files under "/04a_image_segmentation/". The main area contains a section titled "Applying the algorithm" which explains the Voronoi-Otsu labeling algorithm and its parameters. Below this, a code cell shows the execution of the algorithm on a cropped image, resulting in a labeled image. A preview of the labeled image is shown, and its properties are displayed: shape (200, 200), dtype int32, size 156.2 kB, min 0, max 12.

The screenshot shows the continuation of the JupyterLab session. The notebook is still titled "11_voronoi_otsu_labeling.ipynb". The main area now contains an "Exercise" section 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." An optional task is also mentioned: "Optional: Write a function that loads the image, segments it and returns the number of objects." A code cell is present at the bottom for further work.

Image segmentation exercises

- 3D-segmentation using GPUs.
- Consider using UL's Jupyter Hub.

A screenshot of a JupyterLab interface. On the left, there is a file tree showing a directory structure under '/04a_image_segmentation/'. The current file is '12_Segmentation_3D.ipynb'. The main area displays a code cell with Python code for 3D image segmentation using skimage, pyclesperanto_prototype, napari, and matplotlib. The code includes imports and a comment about selecting a GPU device ('cle.select_device('TX')'). Below the code cell, a note mentions using cropped and resampled image data from the Broad BioImage Challenge.

```
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')

<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 BioImage Challenge (Luisa V. Sokolnicki KL, Carpenter AE (2012). Annotated high-throughput
```

A screenshot of the JupyterHub resource selection interface. It shows various configuration options: Session length (4 hours), Memory (16 GB), Number of CPUs (1 CPU), Partition (paula), and GPU (Tesla A30). Three large green arrows point to the 'Memory' (16 GB), 'Partition' (paula), and 'GPU' (Tesla A30) fields, indicating they are the primary focus for this exercise.

Resource selection

Session length
4 hours

Memory
16 GB

Number of CPUs
1 CPU

Partition
paula

GPU
Tesla A30

Napari - Exercises

- Start using napari from Python

Napari will not work on Jupyter Hub because it's a Window-based app.

The image shows two side-by-side screenshots of a JupyterLab interface. Both screenshots feature a top navigation bar with tabs for 'File', 'Edit', 'View', 'Run', 'Kernel', 'Tabs', 'Settings', and 'Help'. Below the navigation bar is a file browser on the left and a code editor on the right.

Screenshot 1: Opening the napari Viewer

- File Browser:** Shows a directory structure with 'data' and 'images' folders, and files 'napari_intro.ipynb', 'napari-assistant.md', and 'notebook_export.md'. 'napari_intro.ipynb' is selected.
- Code Editor:**
 - Cell [2]: `import napari`
 - Cell [3]: `viewer = napari.Viewer()`
 - Cell [4]: `napari.utils.nbscreenshot(viewer)`A preview window on the right shows a dark image with a single blue blob.

Screenshot 2: Segmentation visualization

- File Browser:** Shows the same directory structure, with 'napari_intro.ipynb' selected.
- Code Editor:**
 - Cell [13]:

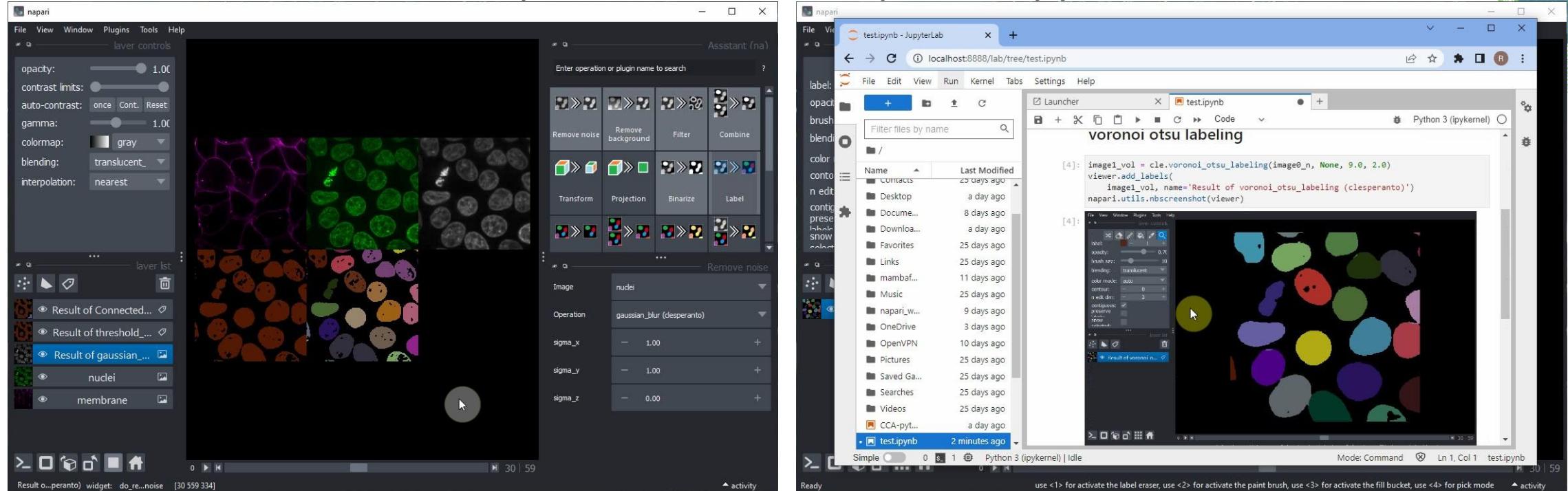
```
blurred = gaussian(mri, sigma=5)  
binary_image = blurred > threshold_otsu(blurred)  
viewer.add_labels(binary_image)  
napari.utils.nbscreenshot(viewer)
```

A preview window on the right shows a brain scan with a segmented region highlighted in red.

Napari - Exercises

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

Napari will not work on Jupyter Hub because it's a Window-based app.



https://github.com/ScaDS/BIDS-lecture-2025/blob/main/03b_napari_notebooks/napari-assistant.md

https://github.com/ScaDS/BIDS-lecture-2025/blob/main/03b_napari_notebooks/notebook_export.md