



Image segmentation

Robert Haase

Using materials from

Alba Villaronga Luque and Jesse Veenliet, MPI-CBG Dresden

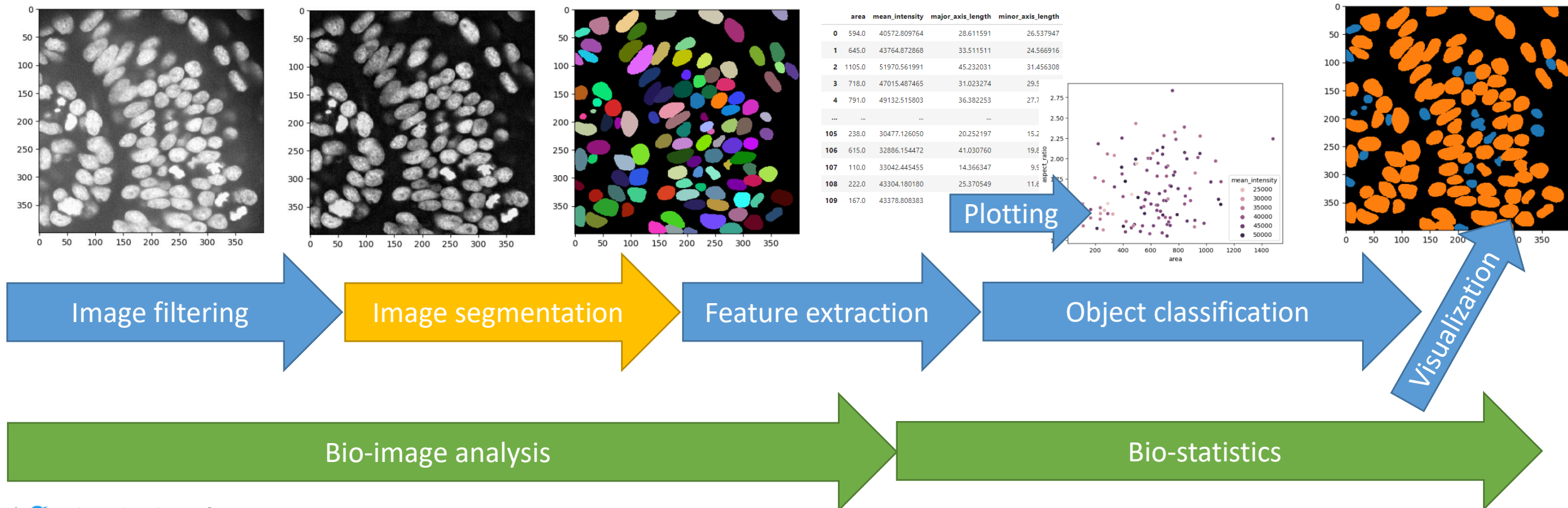
Ryan Savill George, MPI CBG Dresden

Johannes Soltwedel, PoL, TU Dresden

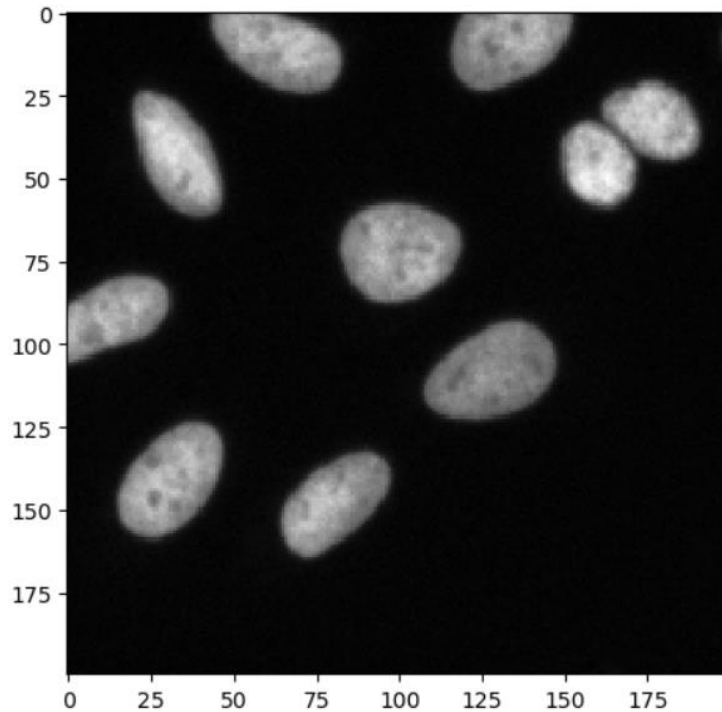
April 2023

Lecture overview: Bio-image Analysis

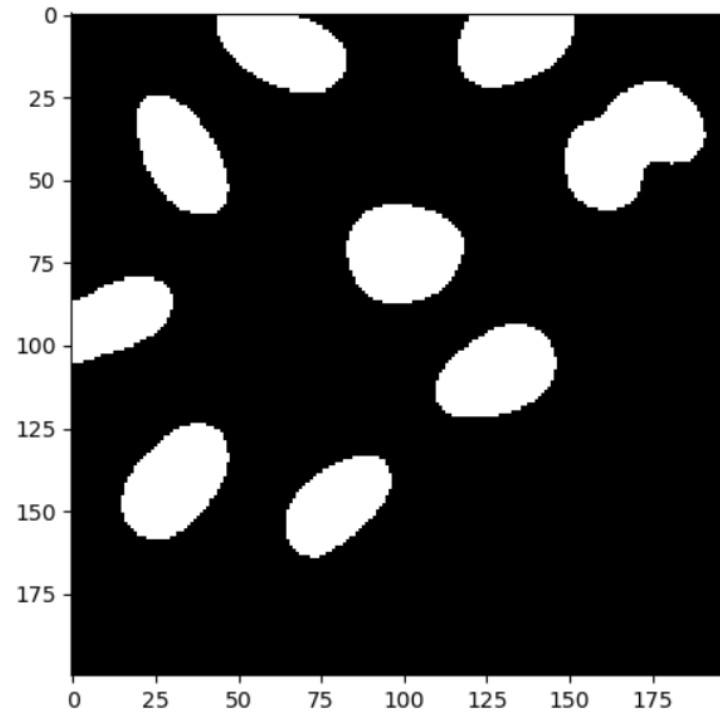
- Image Data Analysis workflows
- Goal: **Quantify observations, substantiate conclusions with numbers**



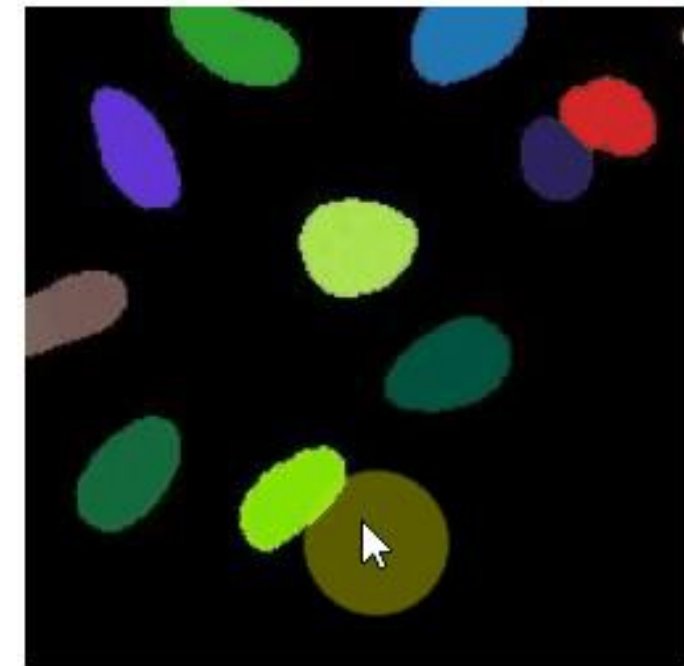
Intensity image



Binary image

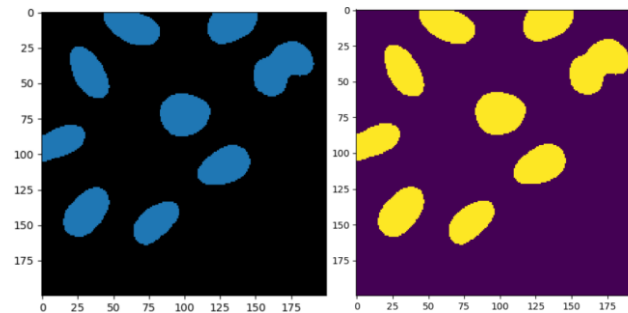


Label image



[y=152, x=92] = 0

No matter how they are displayed



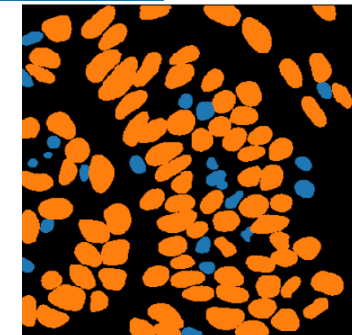
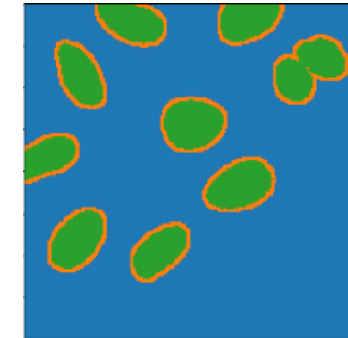
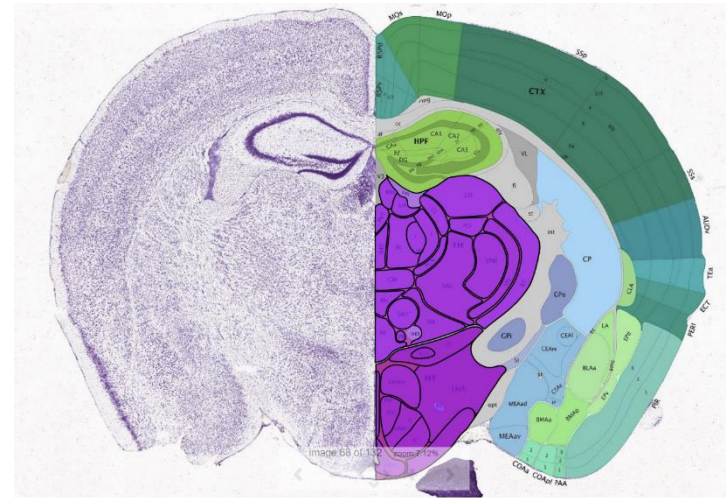
Instance segmentation



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

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

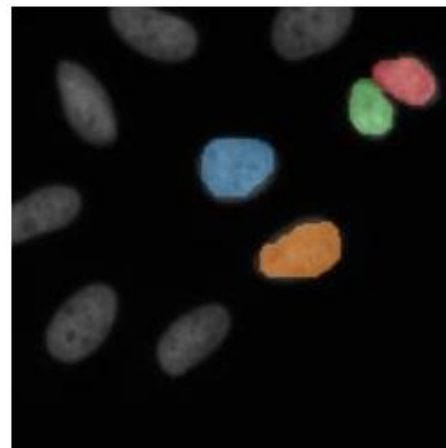
Instance
segmentation



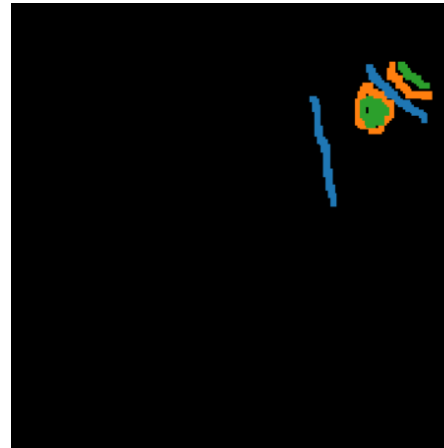
Semantic
segmentation



Sparse instance
annotation

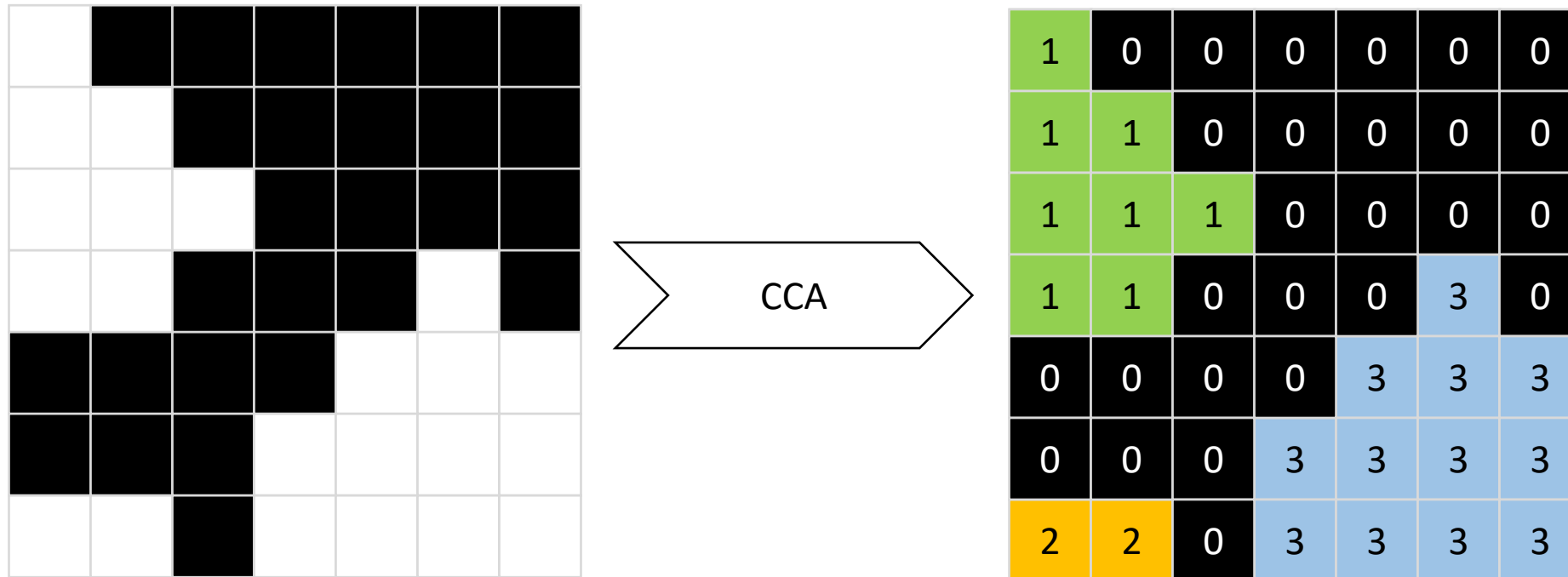


Sparse semantic
annotation



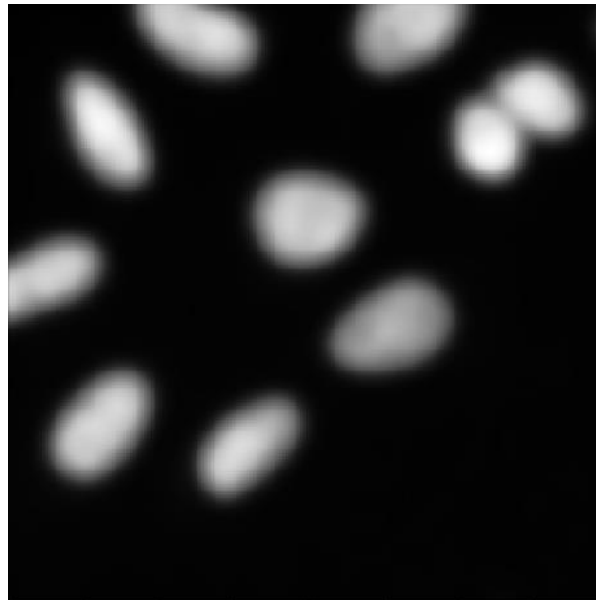
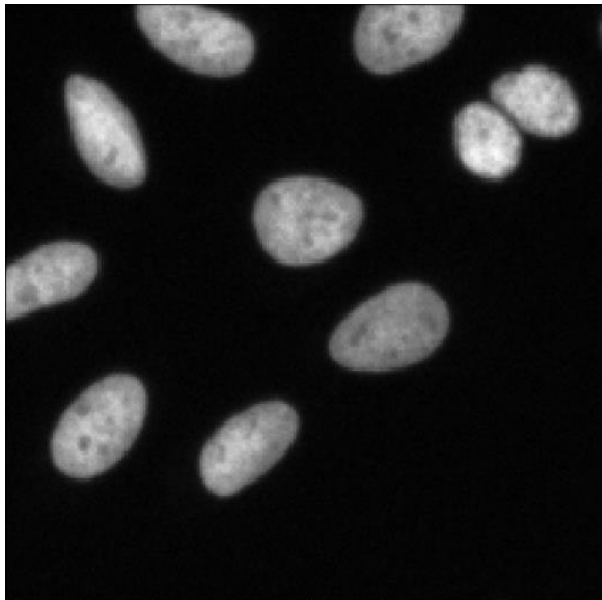
Connected component labelling

- 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.

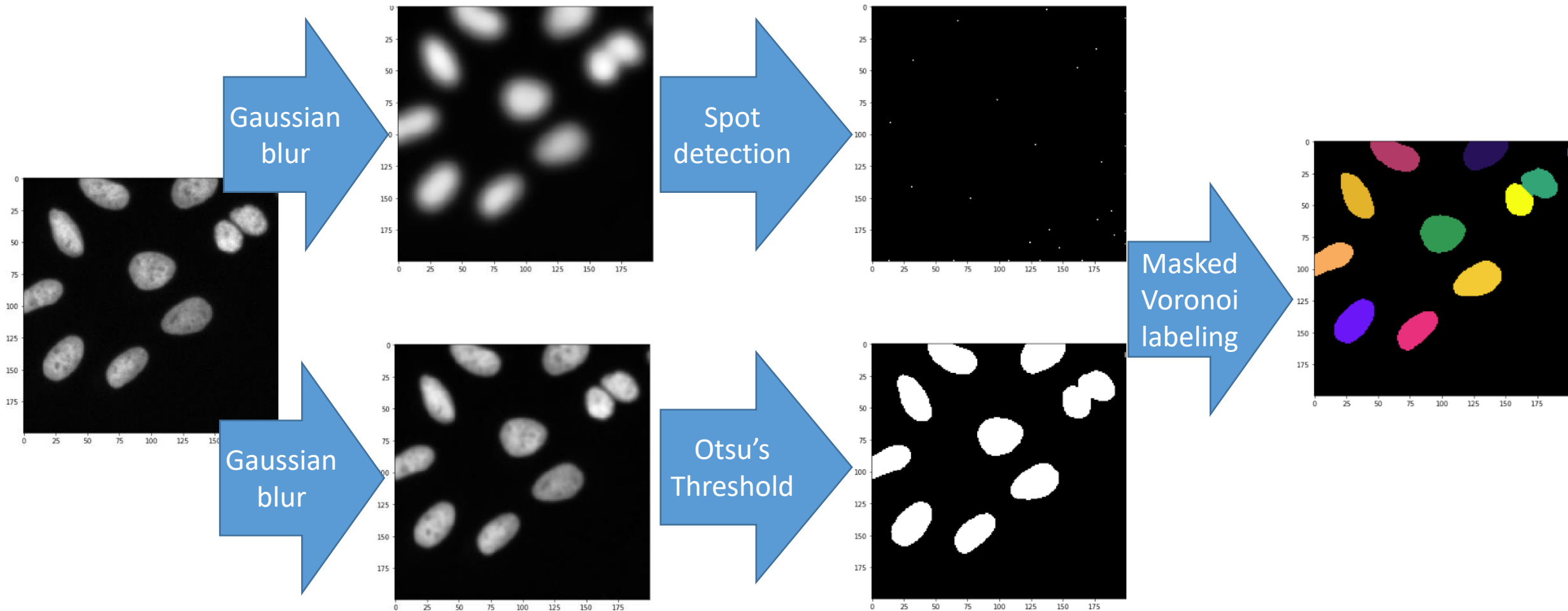


- Presumably the most common segmentation algorithm used for fluorescent microscopy images:
 - Gaussian blur, Otsu's Threshold, Connected Component Labeling

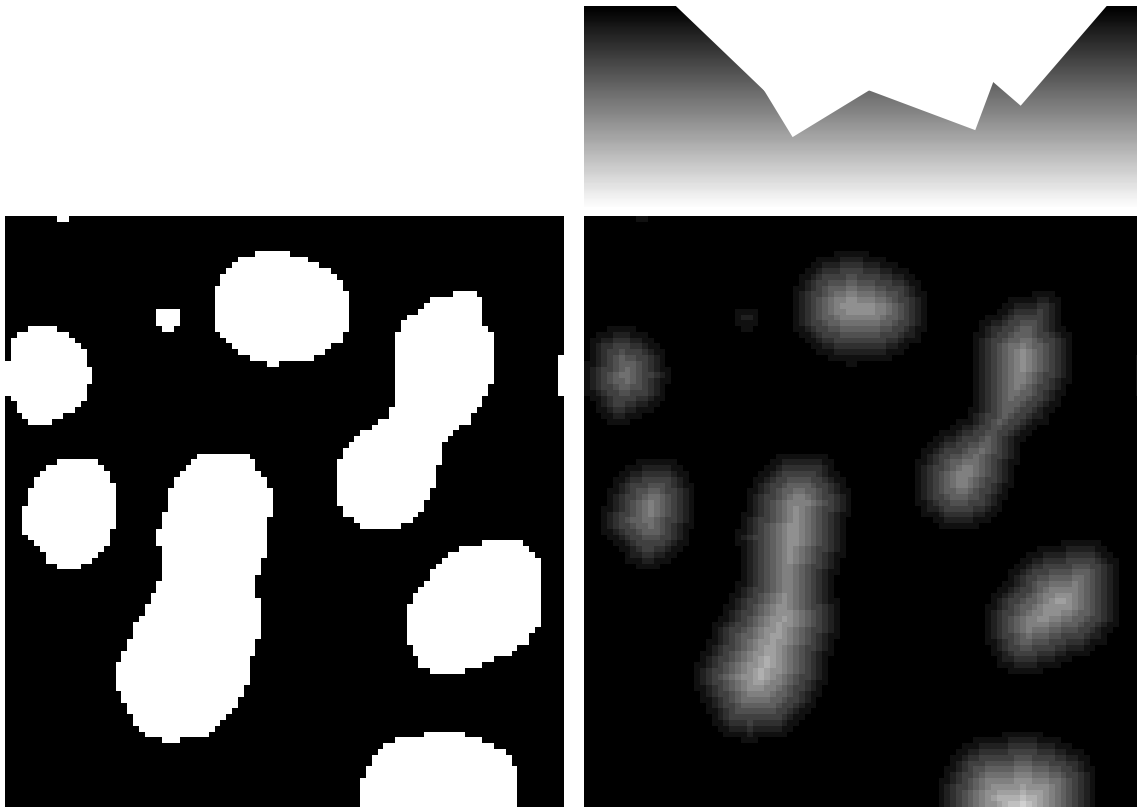
Limitation: Dense objects



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



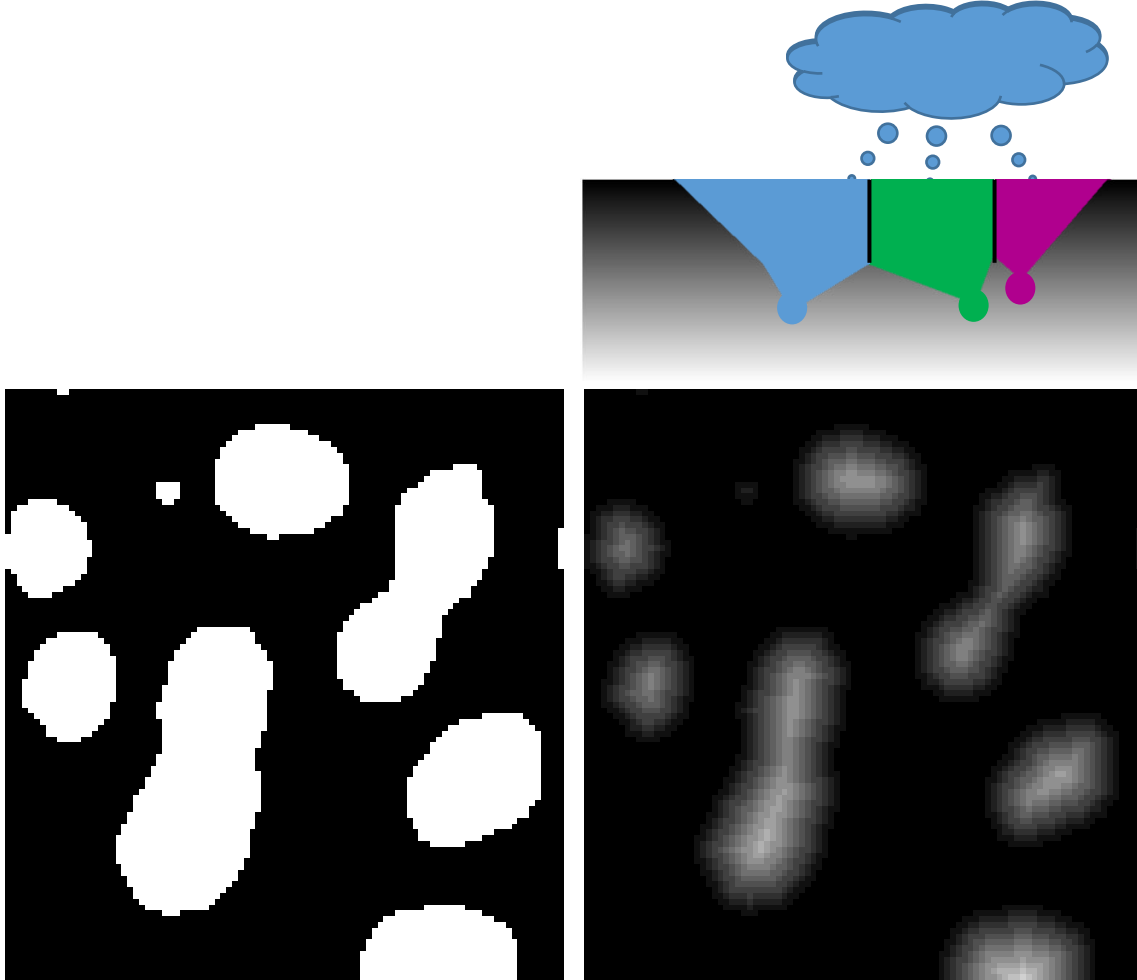
- The watershed algorithm for binary images allows cutting one object into two where it's reasonable.



Binary segmentation

Distance map

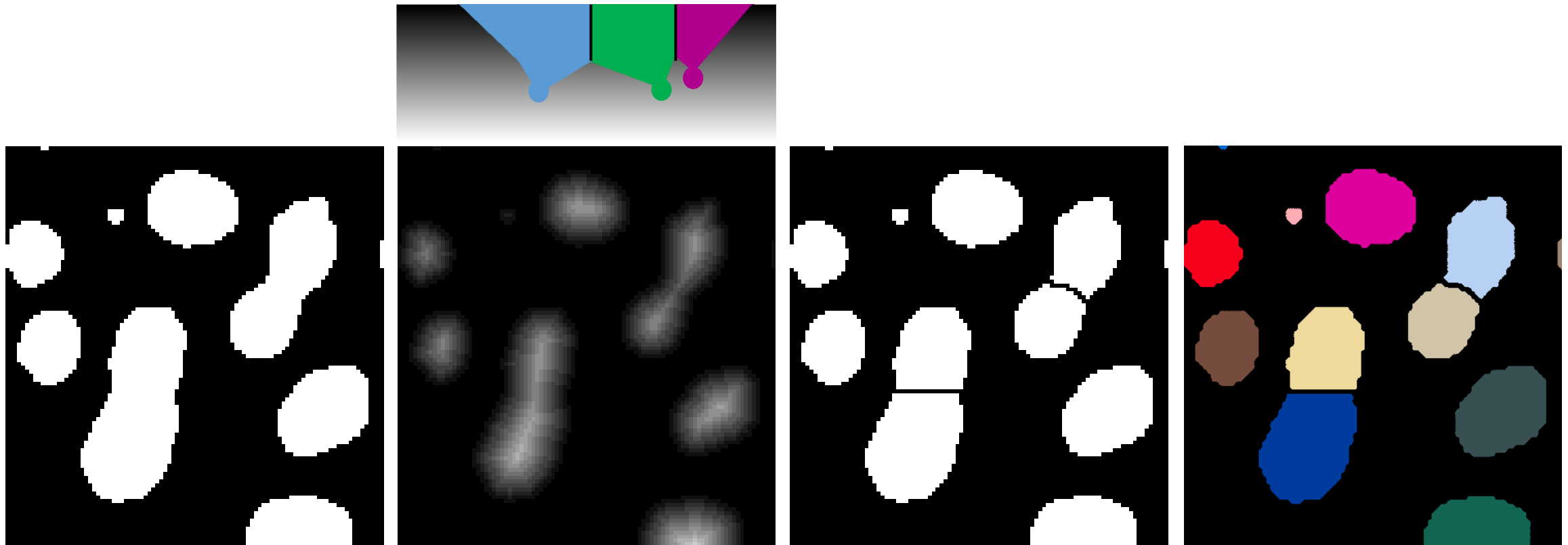
- The watershed algorithm for binary images allows cutting one object into two where it's reasonable.



Binary segmentation

Distance map

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



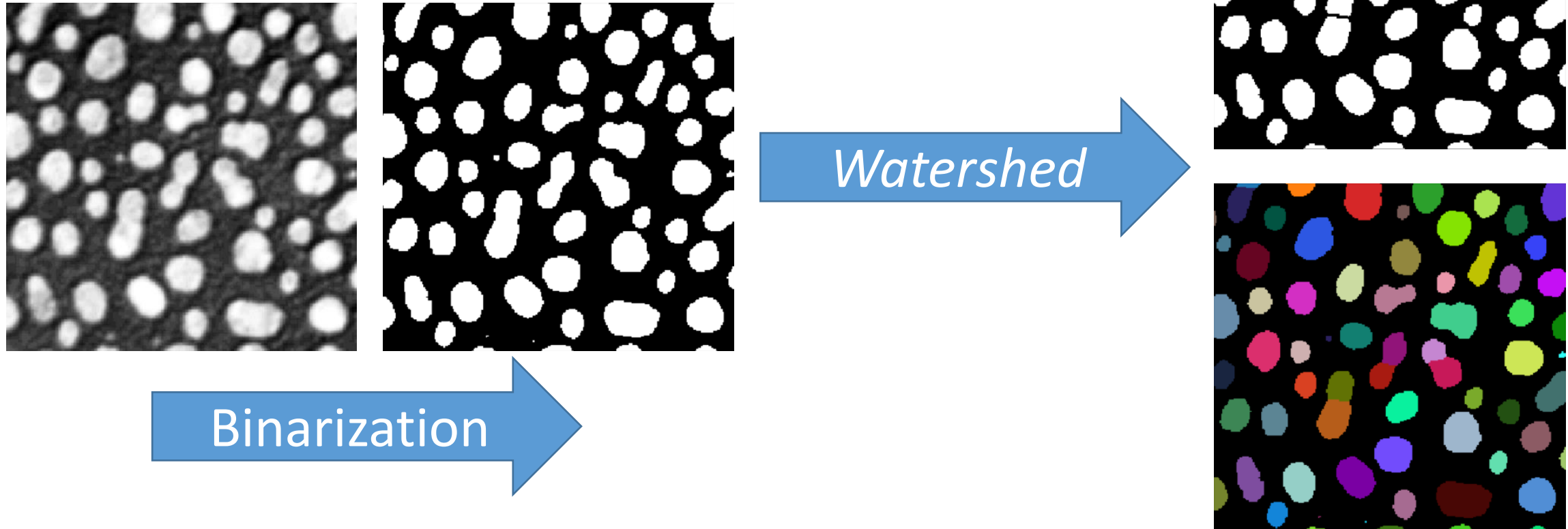
Binary segmentation

Distance map

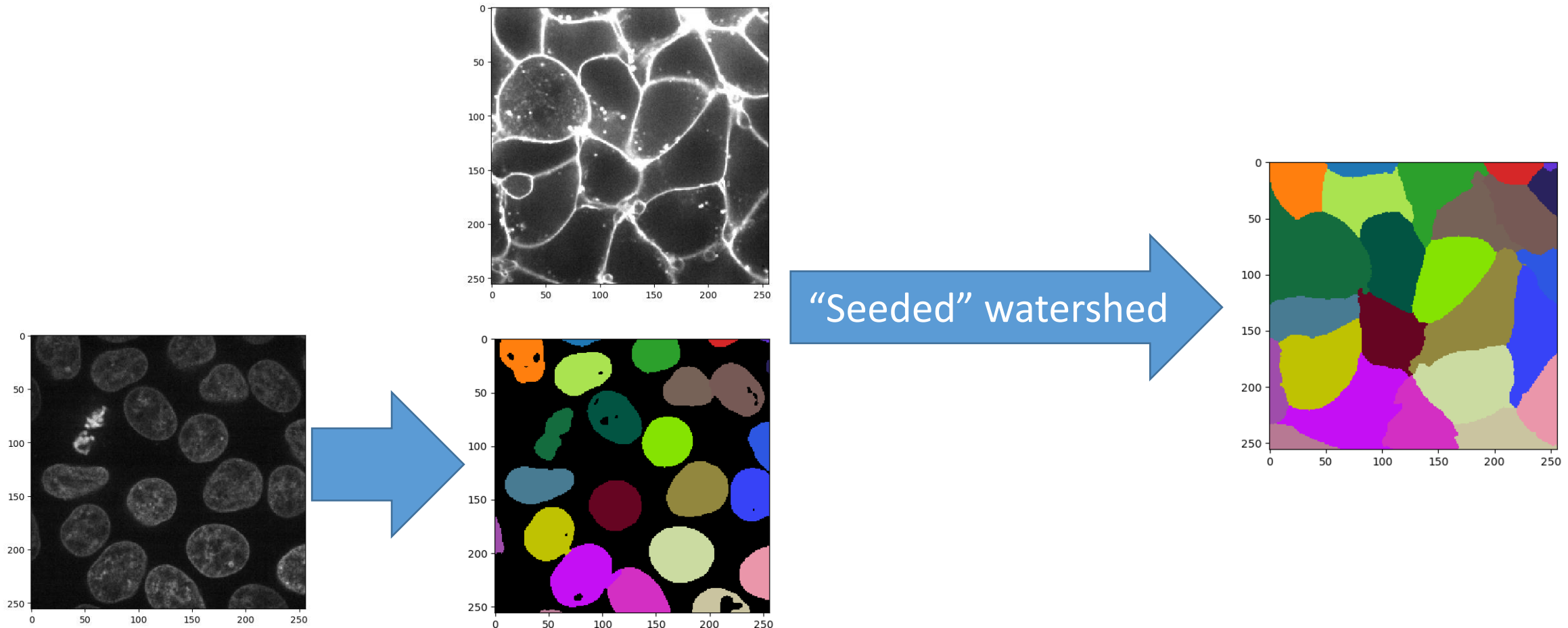
Binary watershed

Labeled watershed

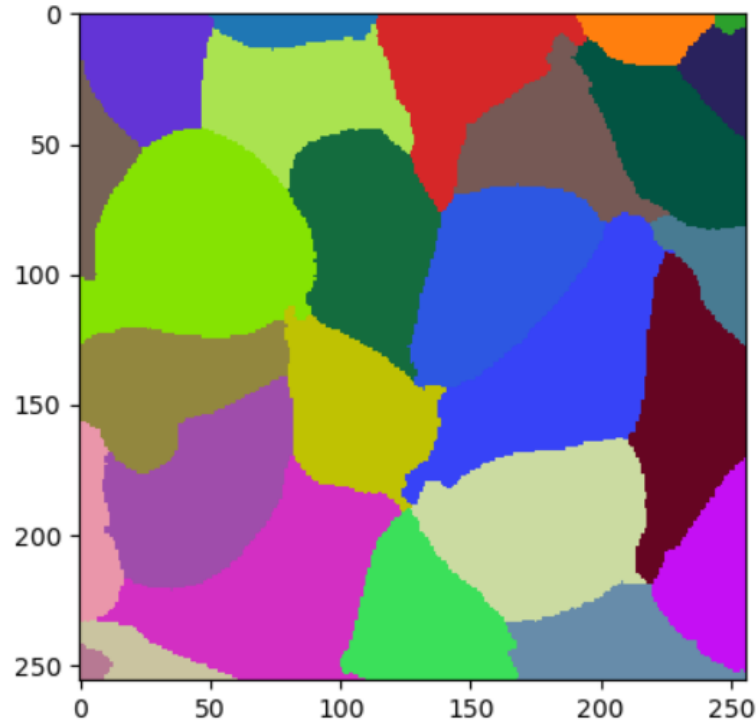
- Split dense objects



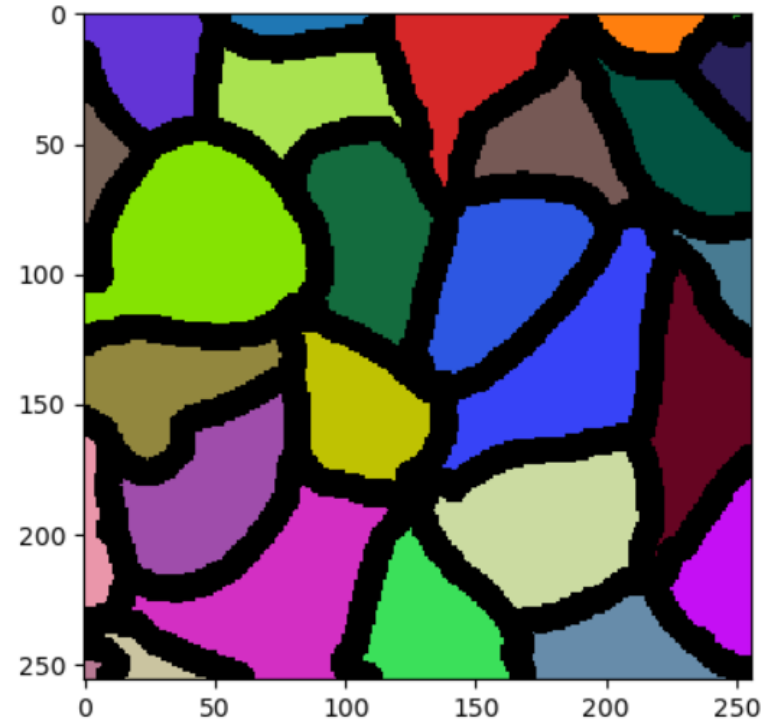
- Seeded watershed: Flood regions from pre-defined seeds
- Example: Flood cells from nuclei positions



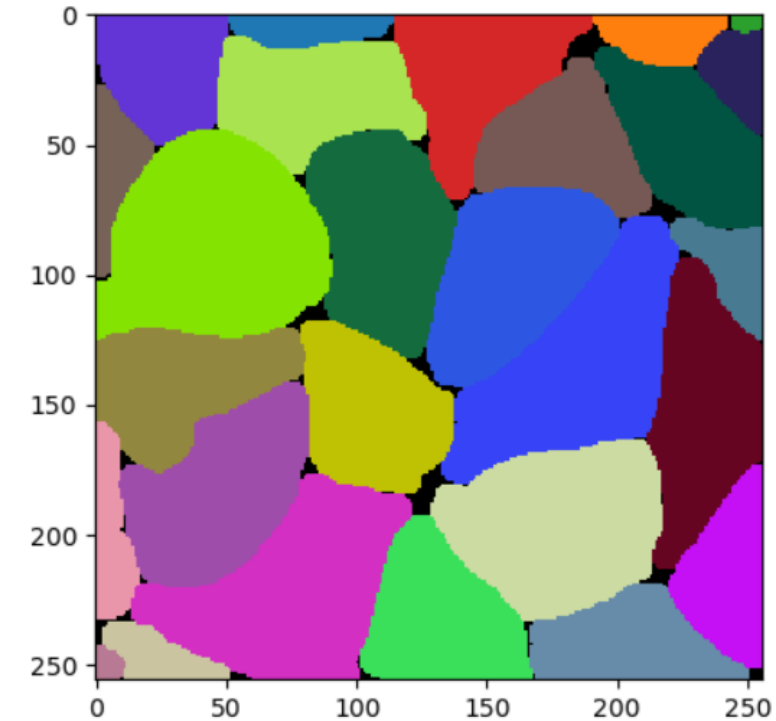
- ... similar to morphological operations on binary images



Original

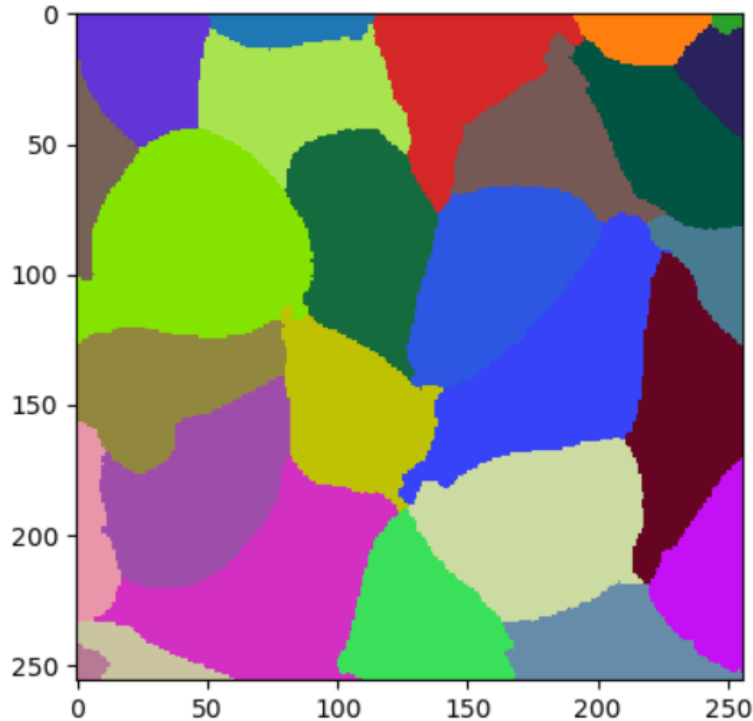


Eroding labels

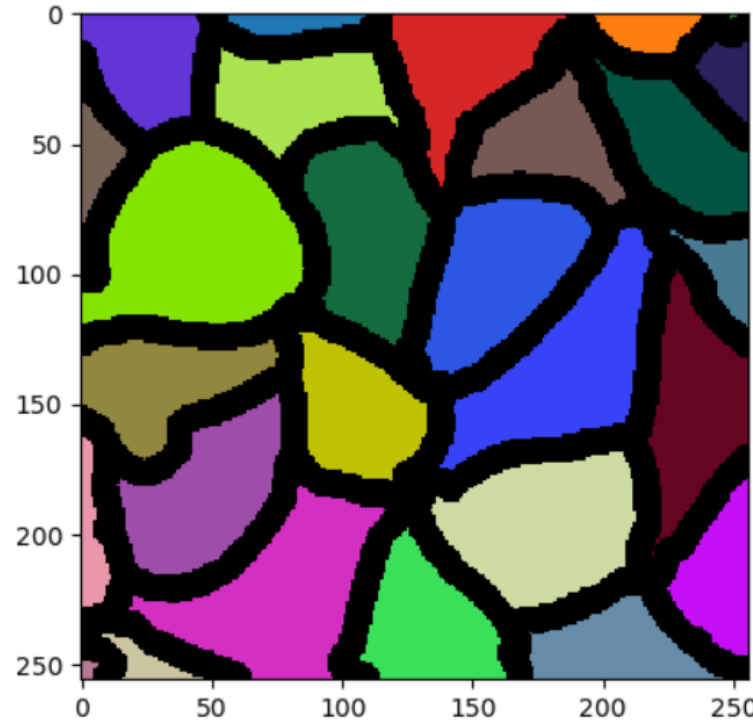


Dilating Labels

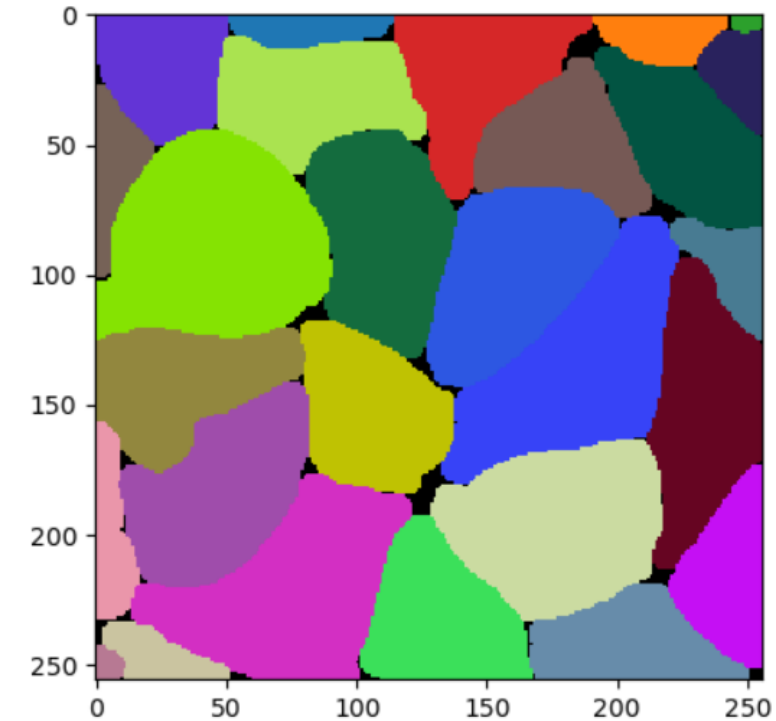
- ... similar to morphological operations on binary images



Original



Eroding labels



Dilating Labels

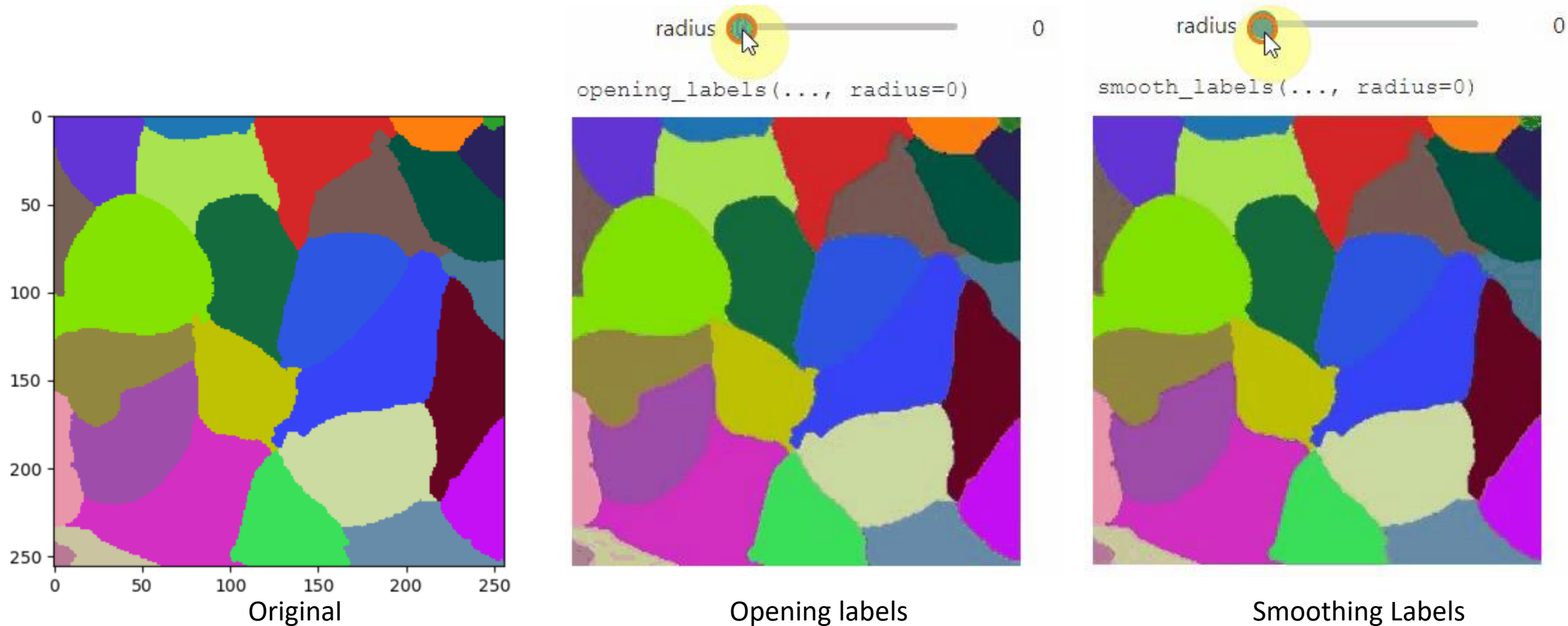
This combination is called

Opening

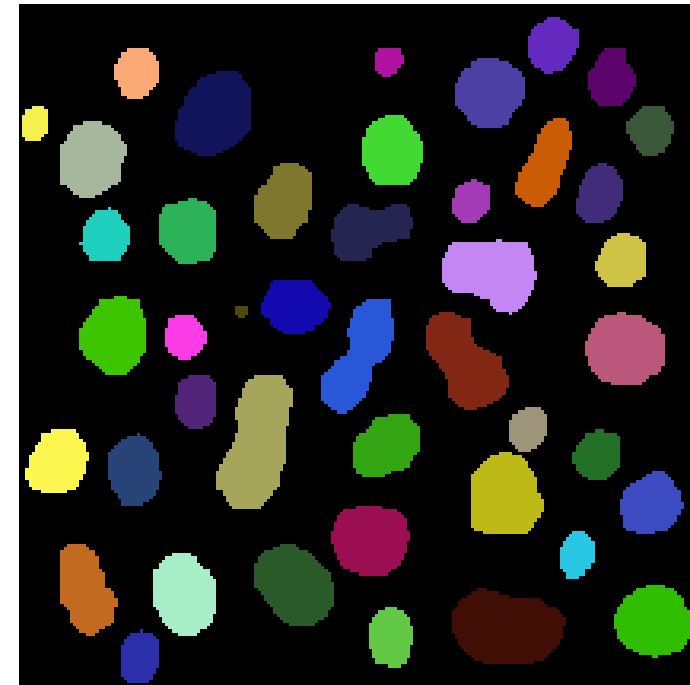
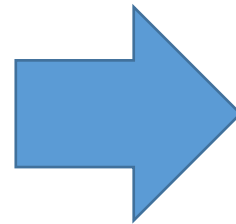
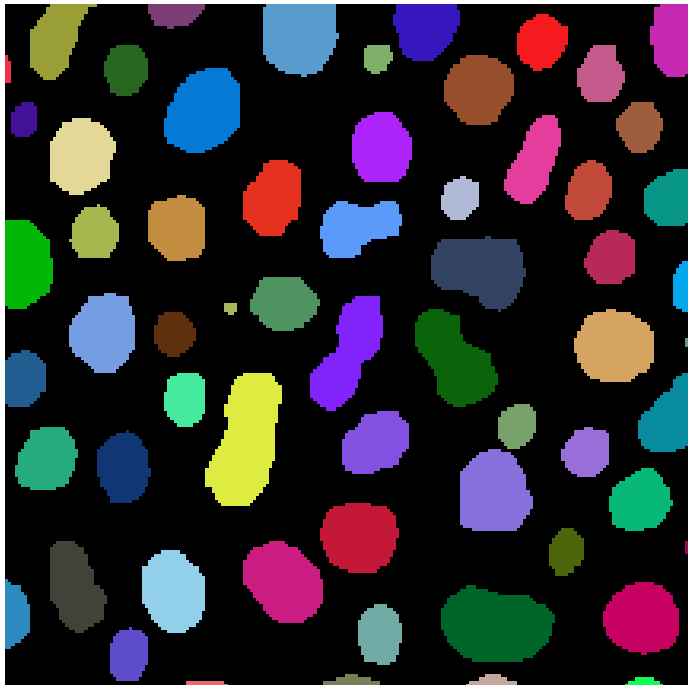
Closing

Epilepsy warning

- ... similar to morphological operations on binary images



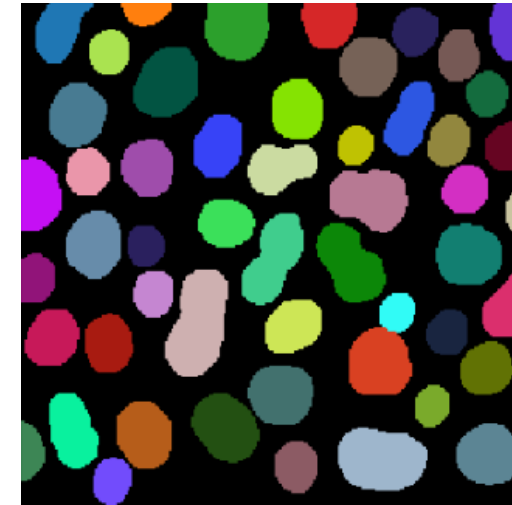
- Remove objects at the image border
- Their measurements (shape, size) would be misleading anyway



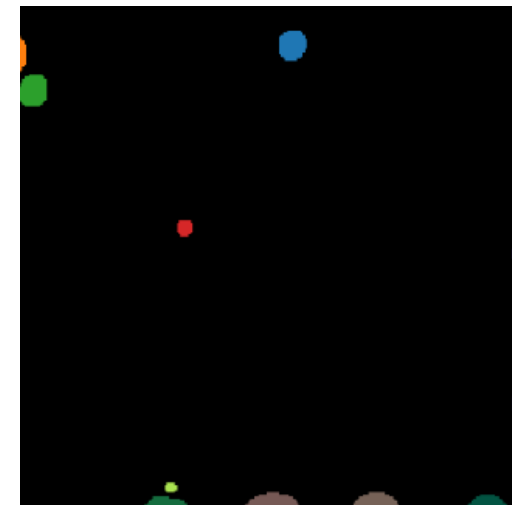
- Excluding small / large objects
- Common correction-step in case segmentations contain noise-related small particles



Exclude small objects



Exclude large objects



Surface reconstruction

Robert Haase

Using materials from

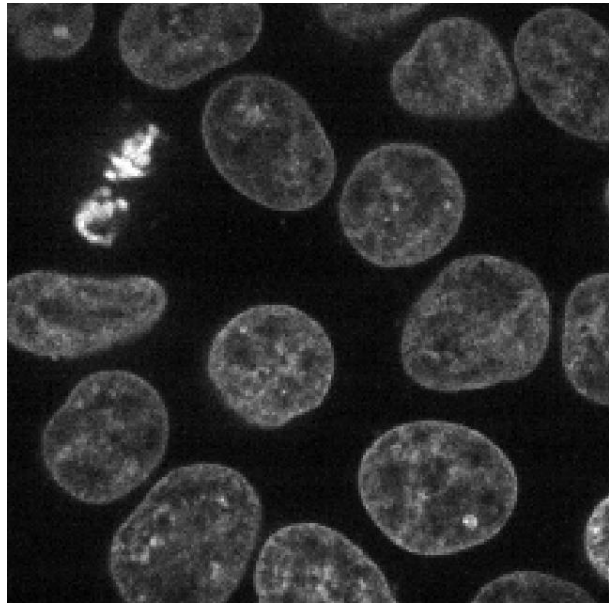
Alba Villaronga Luque and Jesse Veenliet, MPI-CBG Dresden

Johannes Soltwedel, PoL, TU Dresden

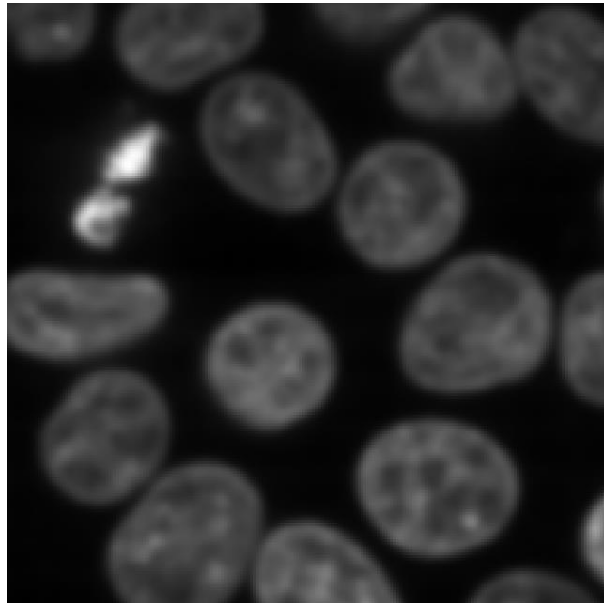
April 2023

Motivation: Surface reconstruction

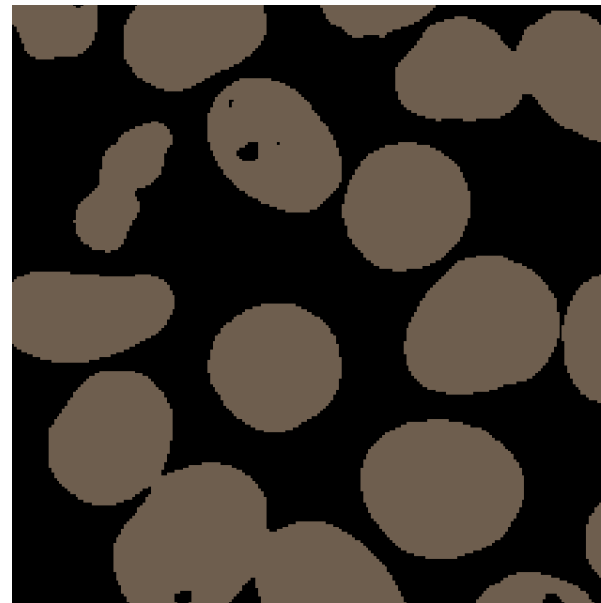
- Pixel and voxel arrays can be huge in memory
- Processing 3D arrays is time-consuming



1024 x 1024 x 100
16-bit image



1024 x 1024 x 100
16-bit image



1024 x 1024 x 100
8-bit image



1024 x 1024 x 100
16-bit image

How much memory does
this workflow cost?

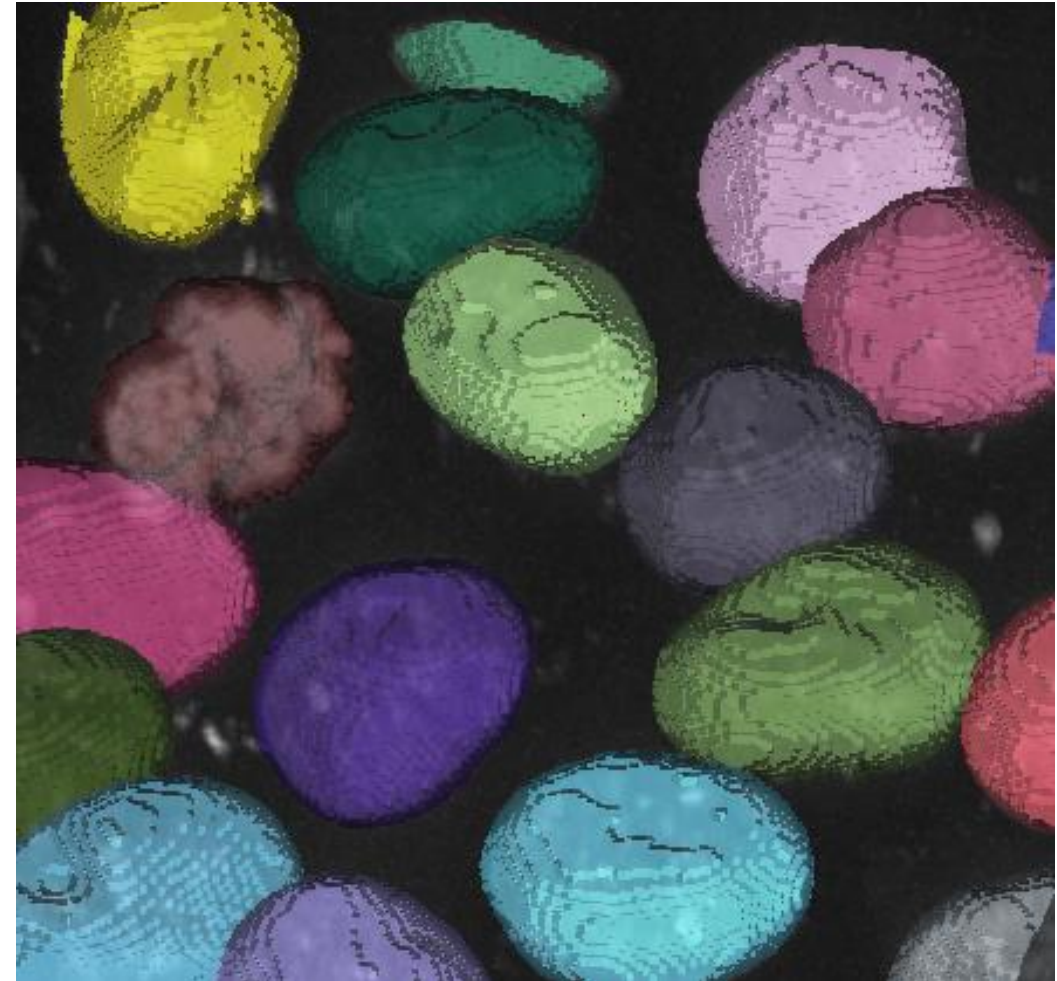
700 MB

400 MB

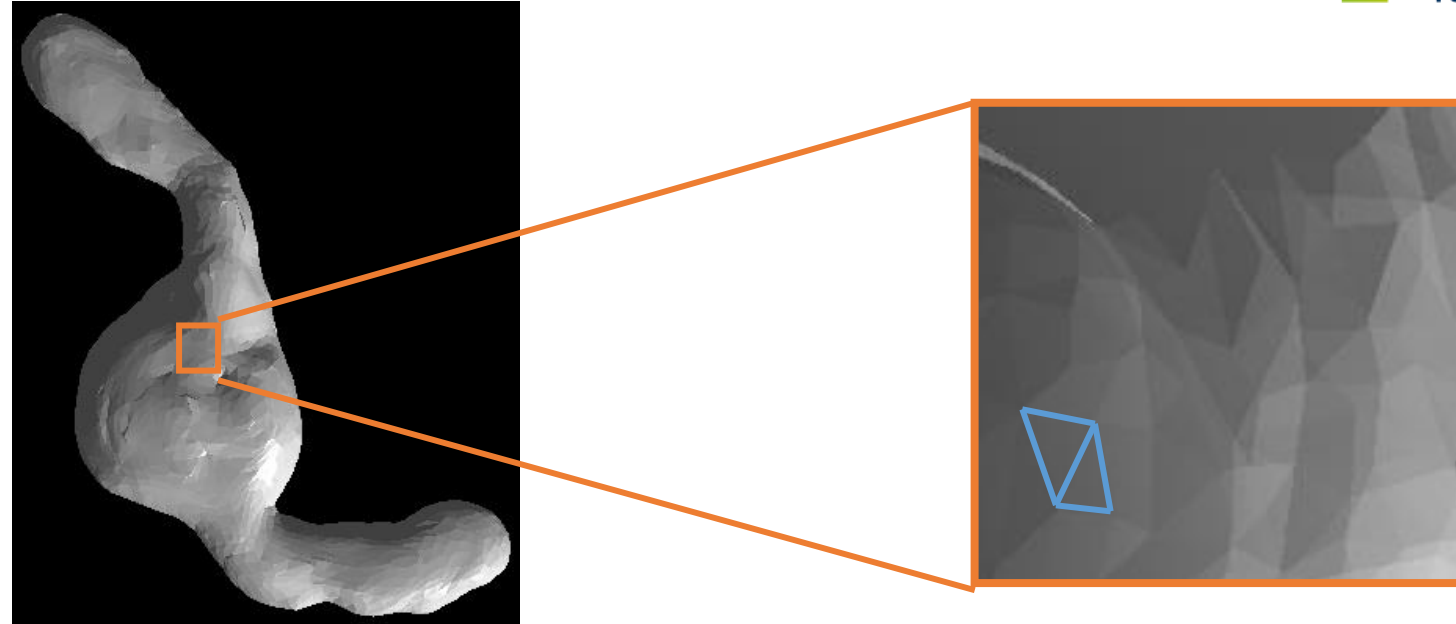
4 GB

7 GB

- Pixel and voxel borders introduce artifacts, potentially problematic for measurements, e.g. surface area



- Points on a surfaces connected by triangles forma a surface mesh

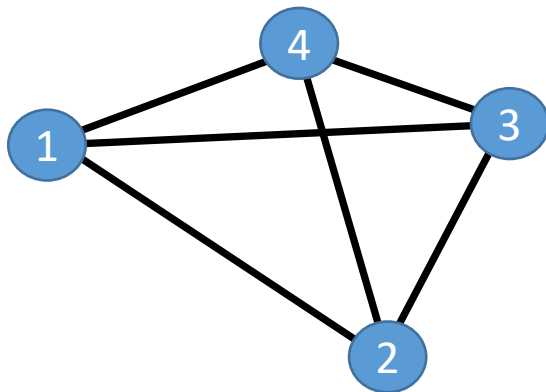


“Vertices” / points

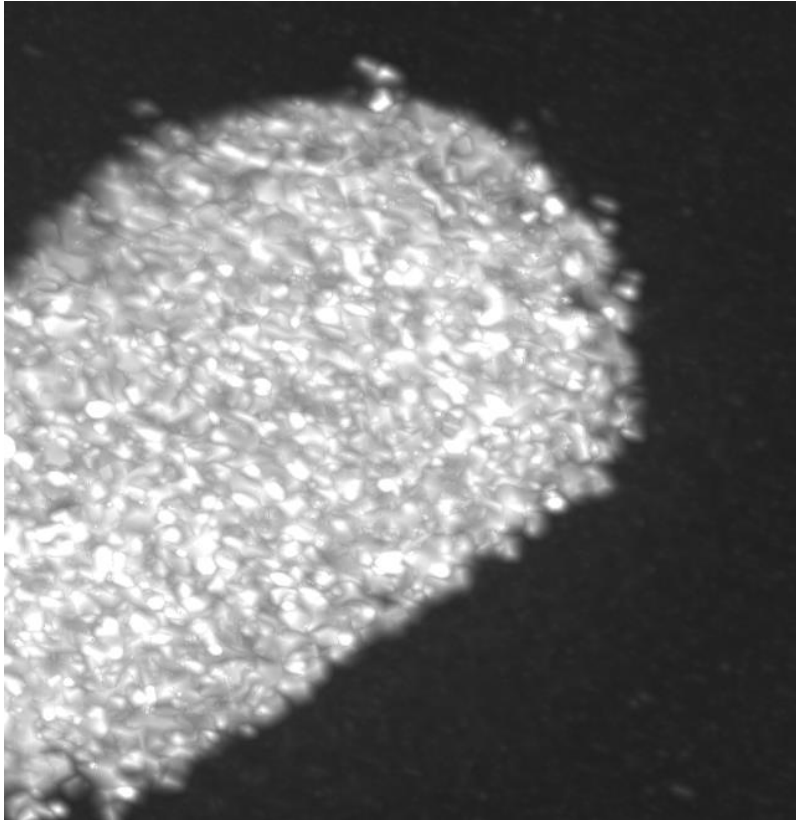
Point x	Point y	Point z
x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_3	z_3
x_4	y_4	z_4
...

“Faces” / Triangles

Point 1	Point 2	Point 3
1	2	3
1	2	4
2	3	4
1	3	4



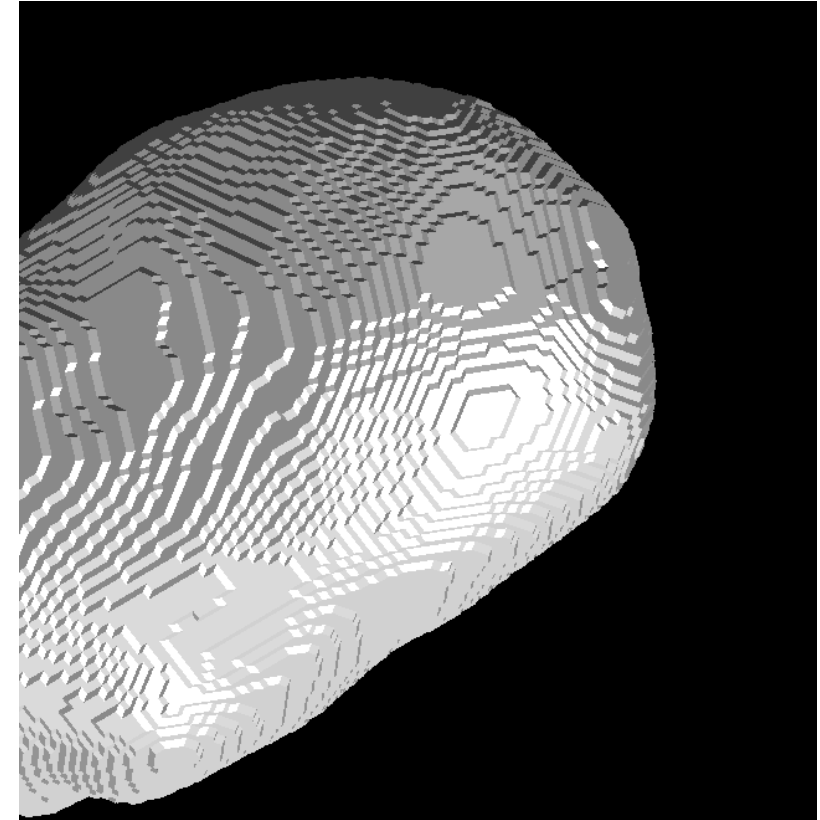
+



3D image of nuclei



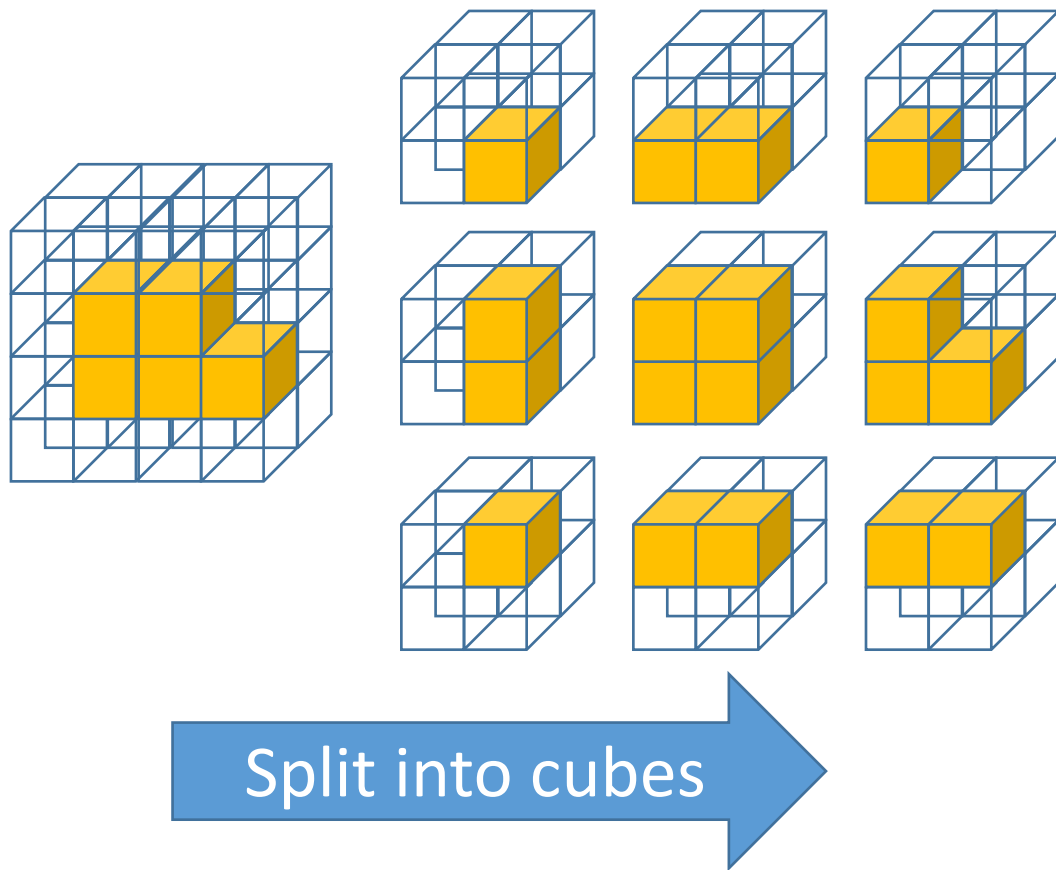
Gaussian filtered



Binary 3D image
(visualized as surface mesh)

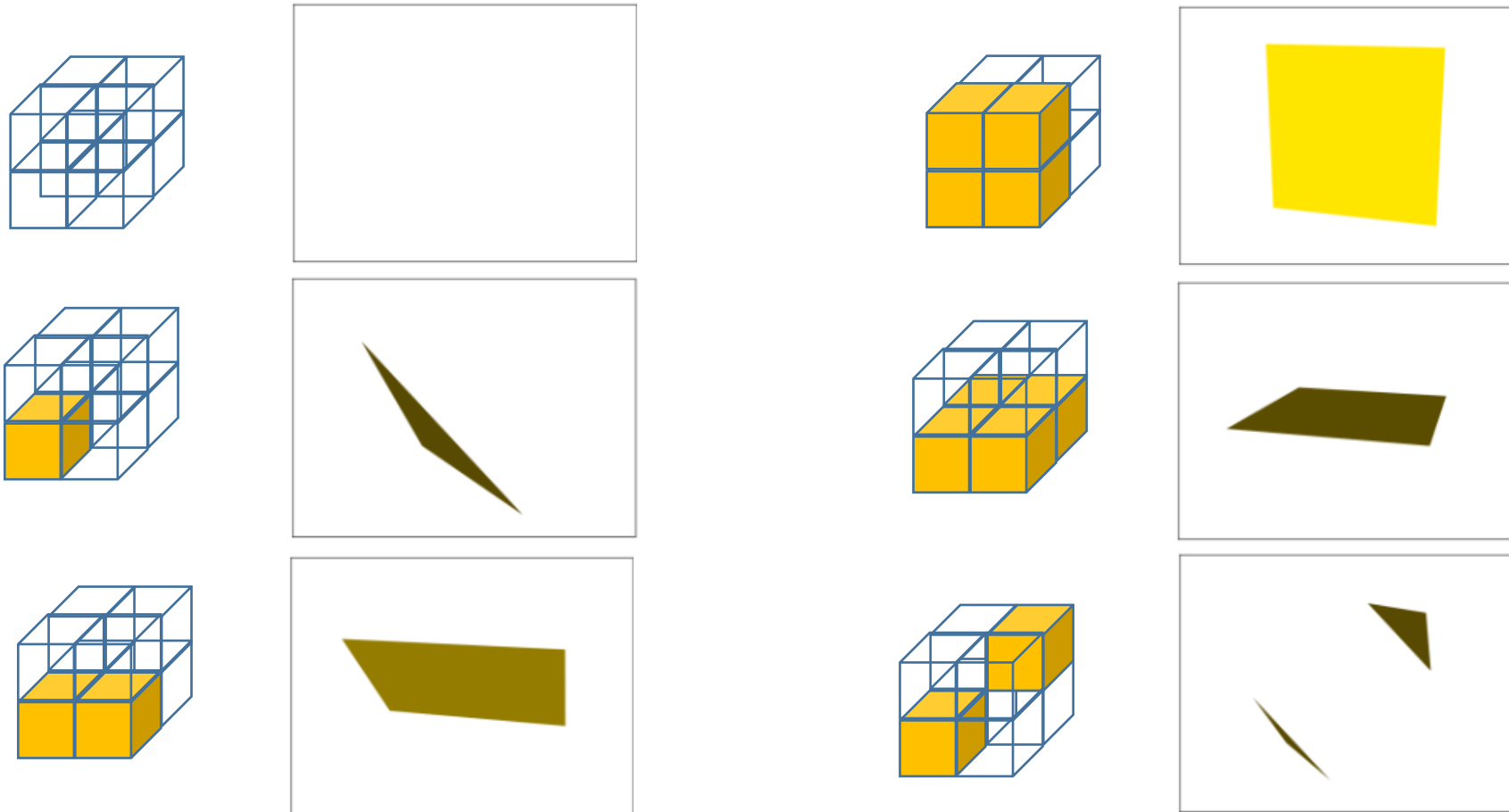
Marching cubes algorithm

- Starting point: 3D binary image
- Cuts the image in small cubes and iterates over them



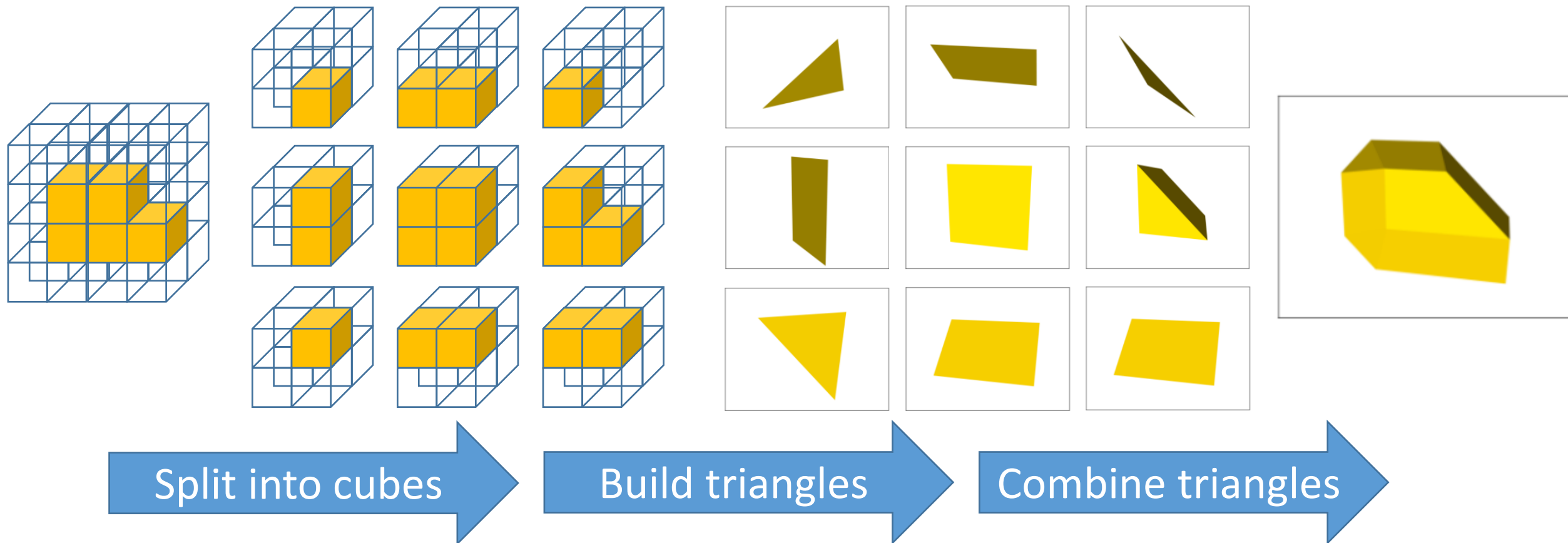
Marching cubes algorithm

- Starting point: 3D binary image
- Cuts the image in small cubes and iterates over them



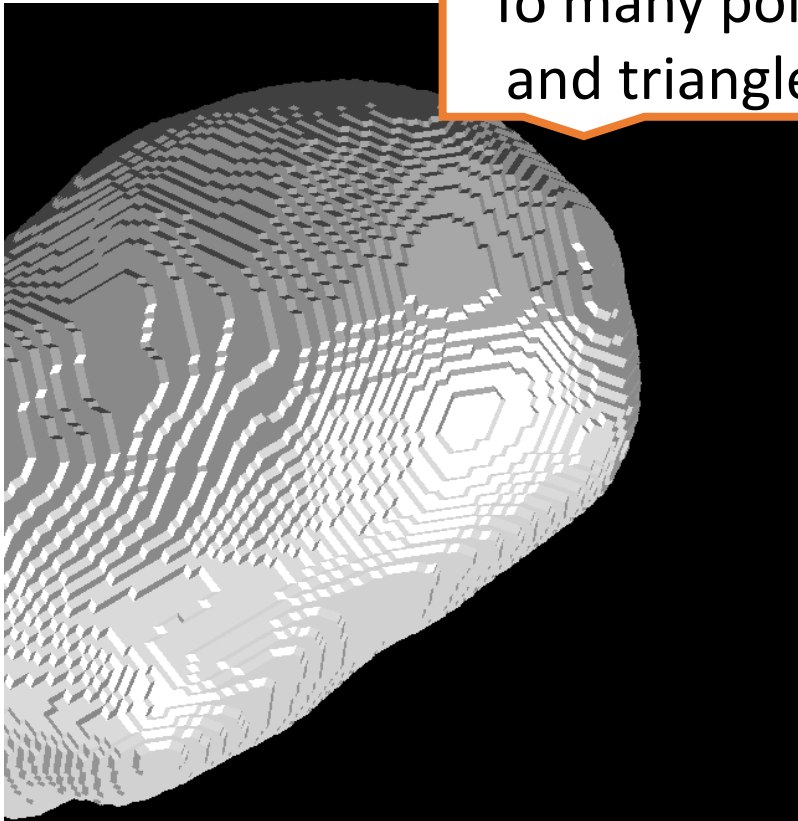
Marching cubes algorithm

- Starting point: 3D binary image
- Cuts the image in small cubes and iterates over them

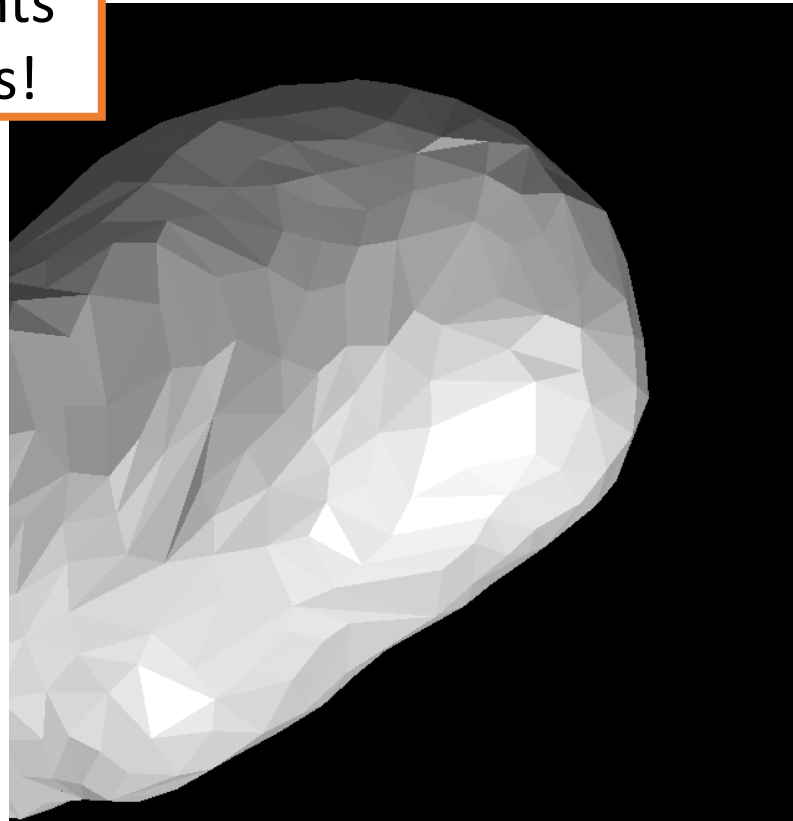


- Necessary to better match biological reality.

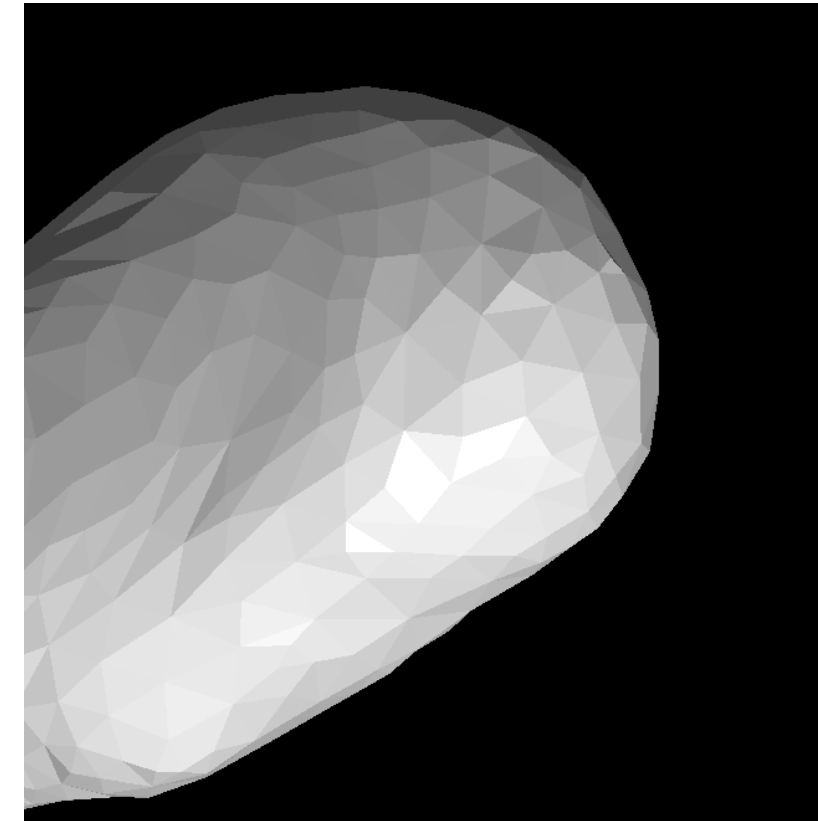
To many points
and triangles!



Marching cubes result



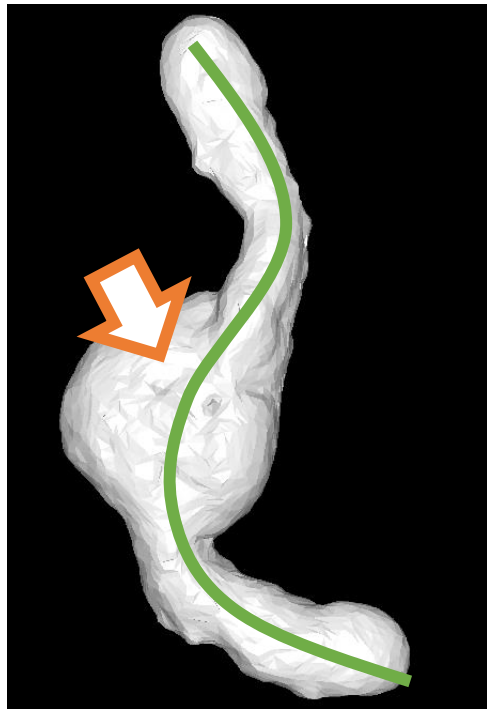
Simplified mesh
(less points, locally averaged)



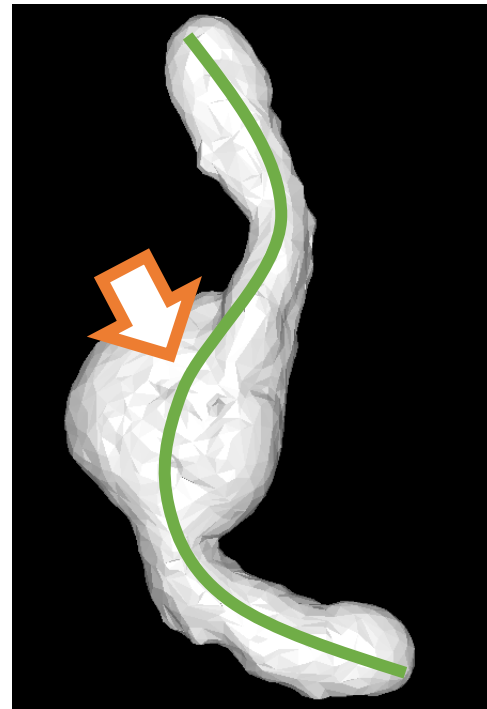
Smoothed mesh
(position locally planarized)

Data derived from [AV Luque and JV Veenliet \(2023\)](#)
licensed [CC-BY](#) : <https://zenodo.org/record/7603081>

- Every processing step has consequences errors of later measurements
- Depends on desired measurement



Surface mesh



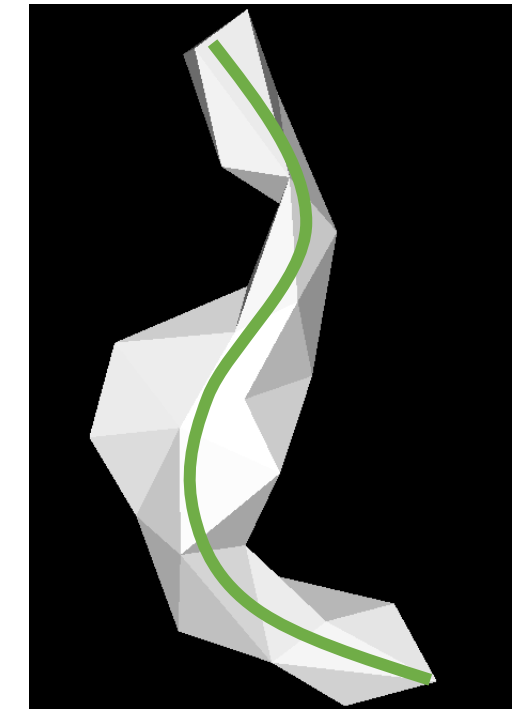
Simplified by factor 0.5

Number of small
concave regions



Simplified by factor 0.05

Total length

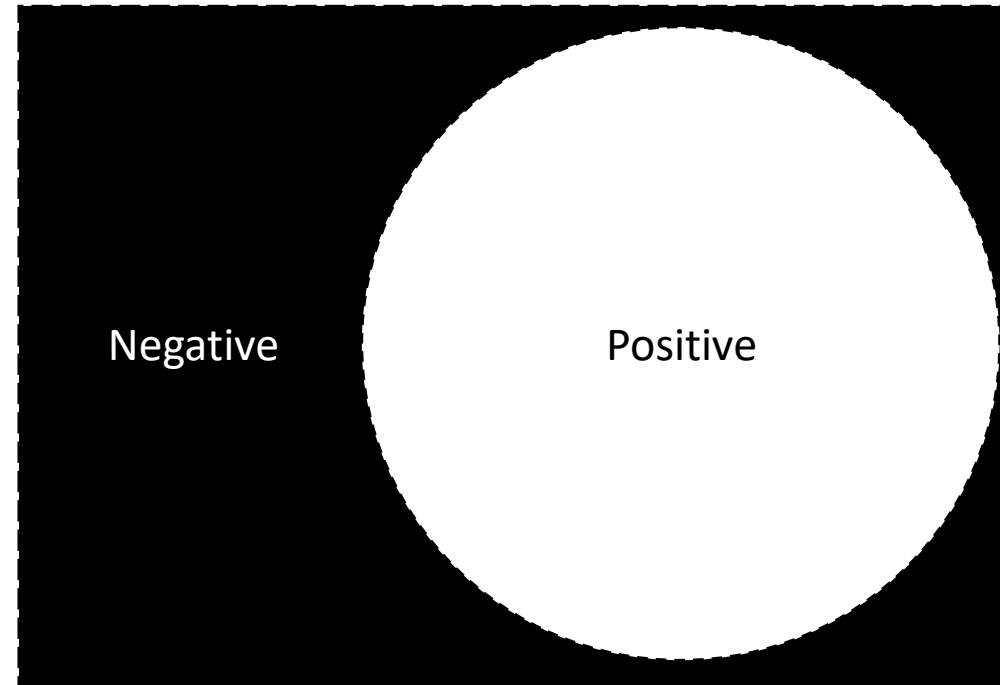


Simplified by factor 0.01

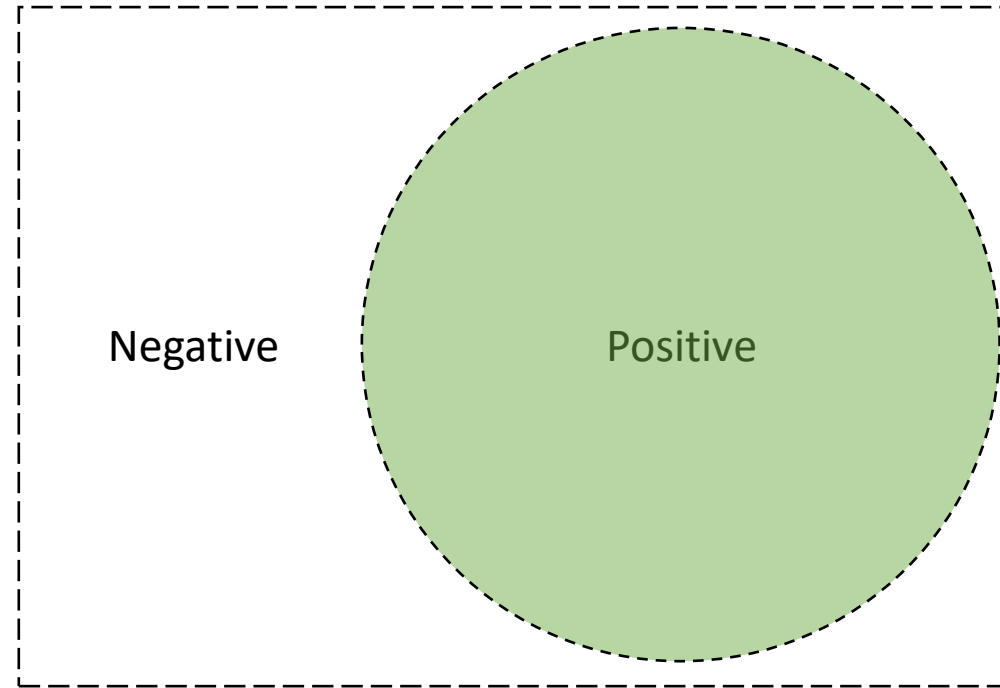
Segmentation quality estimation

Robert Haase

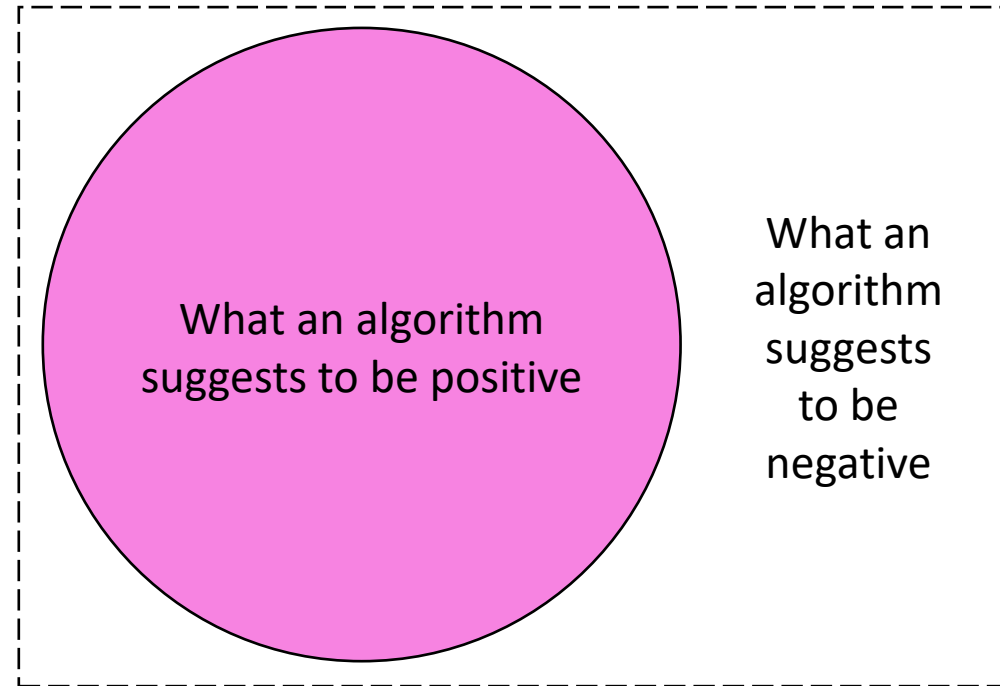
- In general
 - Define what's positive and what's negative.



- In general
 - Define what's positive and what's negative.

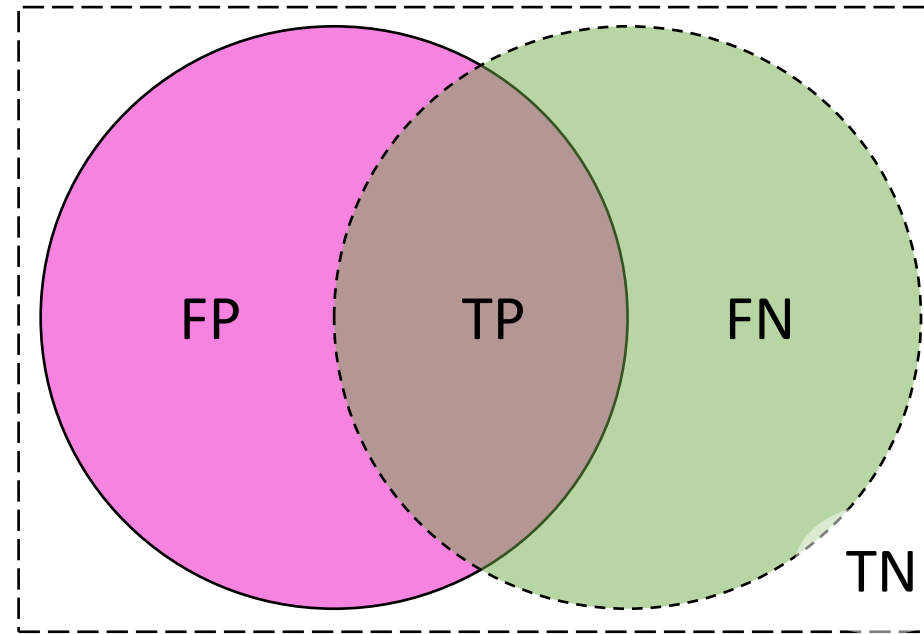




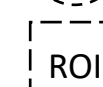



- In general
 - Define what's positive and what's negative.



Segmentation quality estimation

- In general
 - Define what's positive and what's negative.
 - Compare with a reference to figure out what was true and false
- Welcome to the Theory of Sets



-  A Prediction A
-  B Reference B (ground truth)
-  ROI Region of interest
-  TP True-positive
-  FN False-negative
-  FP False-positive
- TN True-negative

Overlap
(a.k.a. Jaccard index) $\frac{TP}{TP + FN + FP}$

How much do A and B overlap?

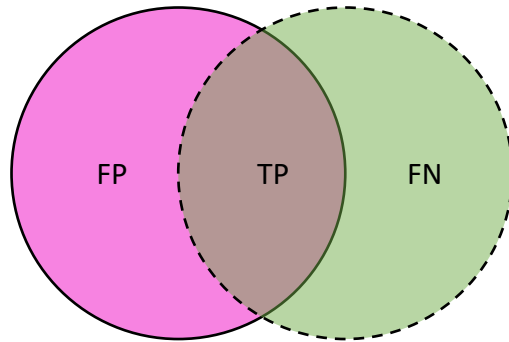
Precision $\frac{TP}{TP + FP}$

What fraction of points that were predicted as positives were really positive?

Recall
(a.k.a. sensitivity) $\frac{TP}{TP + FN}$

What fraction of positives points were predicted as positives?

- Pixel wise: Segmentation quality

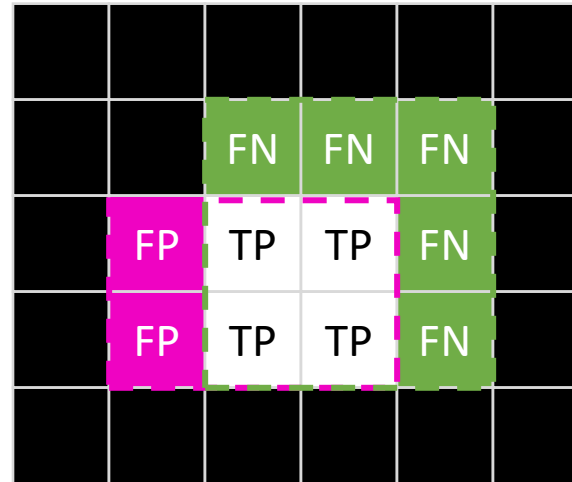


Precision

$$\frac{TP}{TP + FP}$$

Recall
(a.k.a. sensitivity)

$$\frac{TP}{TP + FN}$$



True-positive: 4

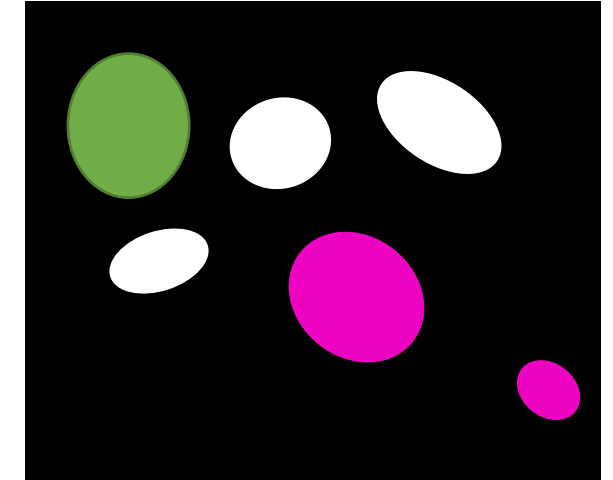
False-negative: 5

False-positive: 2

Precision: $4/6 = 66\%$

Recall: $4/9 = 44\%$

- Object wise: Detection quality



True-positive: 3

False-negative: 1

False-positive: 2

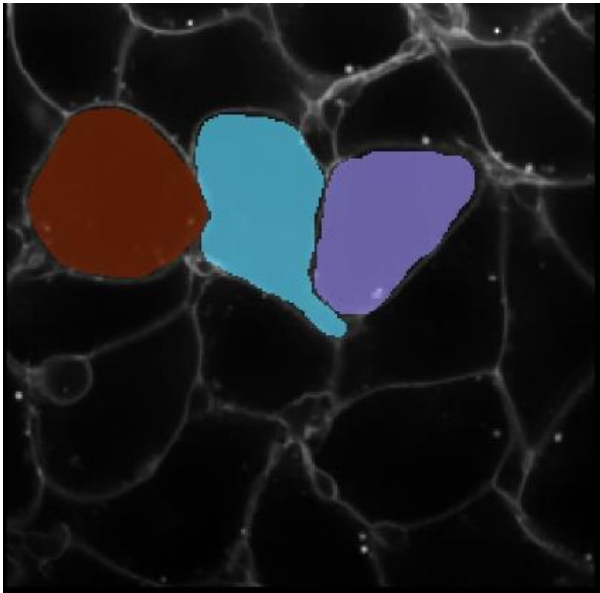
Precision: $3/4 = 75\%$

Recall: $3/5 = 60\%$

Pixel-wise versus Object-wise evaluation

- Average Overlap for all ground-truth objects
- <https://github.com/haesleinhuepf/the-segmentation-game>



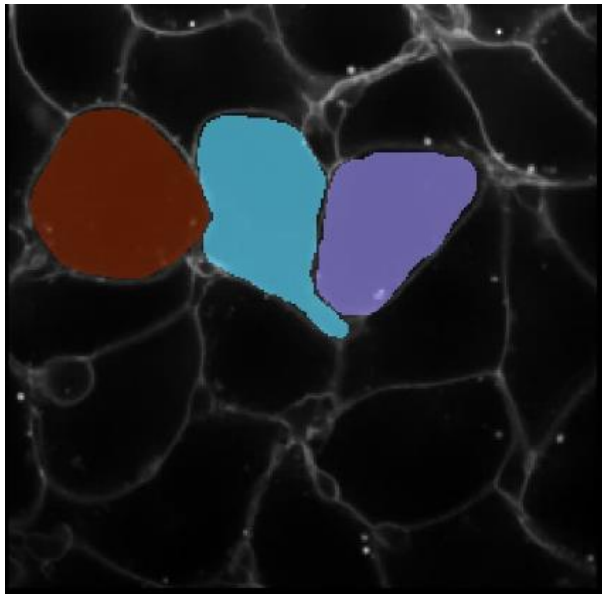


This is a ...

Sparse
instance
segmentation

Sparse
semantic
segmentation

- From some sparsely labels objects we can estimate segmentation quality



Sparse
instance
annotation



$J = 0.35$



$J = 0.66$



$J = 0.69$

- Voxel-wise Youden-Index

$$YI = p_{TP} + p_{TN} - 1$$

- Volume error

$$\Delta_V = V_A - V_B$$

$$\delta_V = \frac{\Delta_V}{V_B}$$

- Dice Index

$$DI(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

- Jaccard Index

$$JI(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{DI}{2 - DI}$$

- Contour distance

$$d_{e,min}(a, B) = \min(d_e(a, b) | b \in B)$$

$$\bar{d}_c(A, B) = \frac{\sum_{\forall a \in C(A)} d_{e,min}(a, C(B))}{|C(A)|}$$

$$\bar{d}_{bil,c}(A, B) = \frac{\bar{d}_c(A, B) + \bar{d}_c(B, A)}{2}$$

- Hausdorff distance

$$d_H(A, B) = \max(d_{e,min}(a, B) | a \in A)$$

$$d_{bil,H}(A, B) = \max(d_H(A, B), d_H(B, A))$$

- Simplified Hausdorff distance

$$d_H(A, B) = \max(d_{e,min}(a, C(B)) | a \in C(A))$$

- Volume standard deviation

$$\delta_{\bar{V}} = 2 \frac{|V_A - V_B|}{|V_A + V_B|}$$

- Classification error

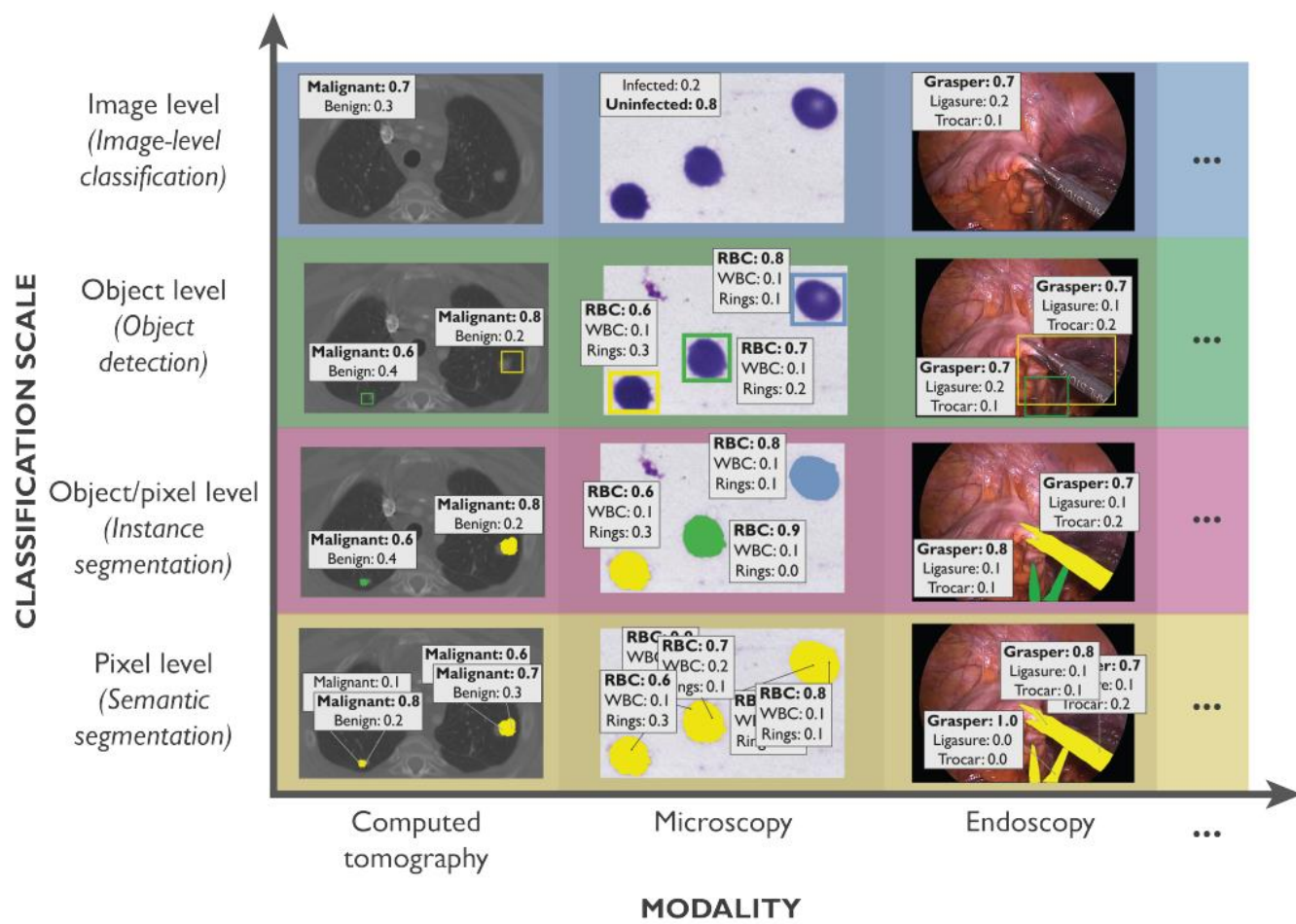
$$e_{Class} = \frac{H}{|TP| + |FN|}$$

- Hamming distance

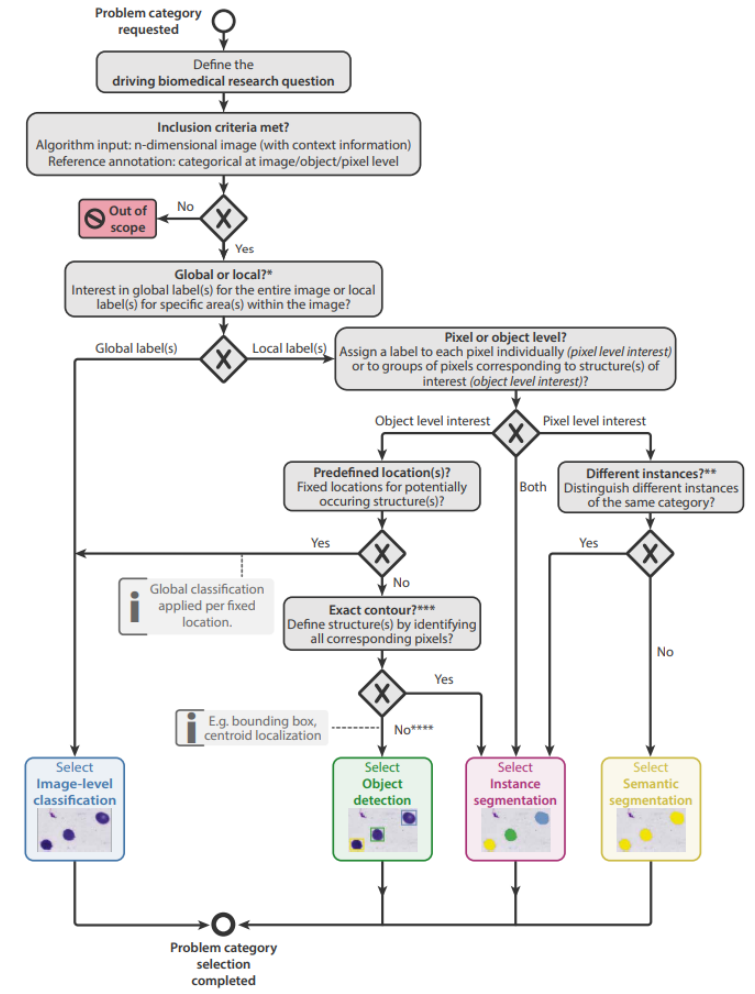
$$\begin{aligned} d_h &= |A \cup B| - |A \cap B| \\ &= |FP| + |FN| \end{aligned}$$

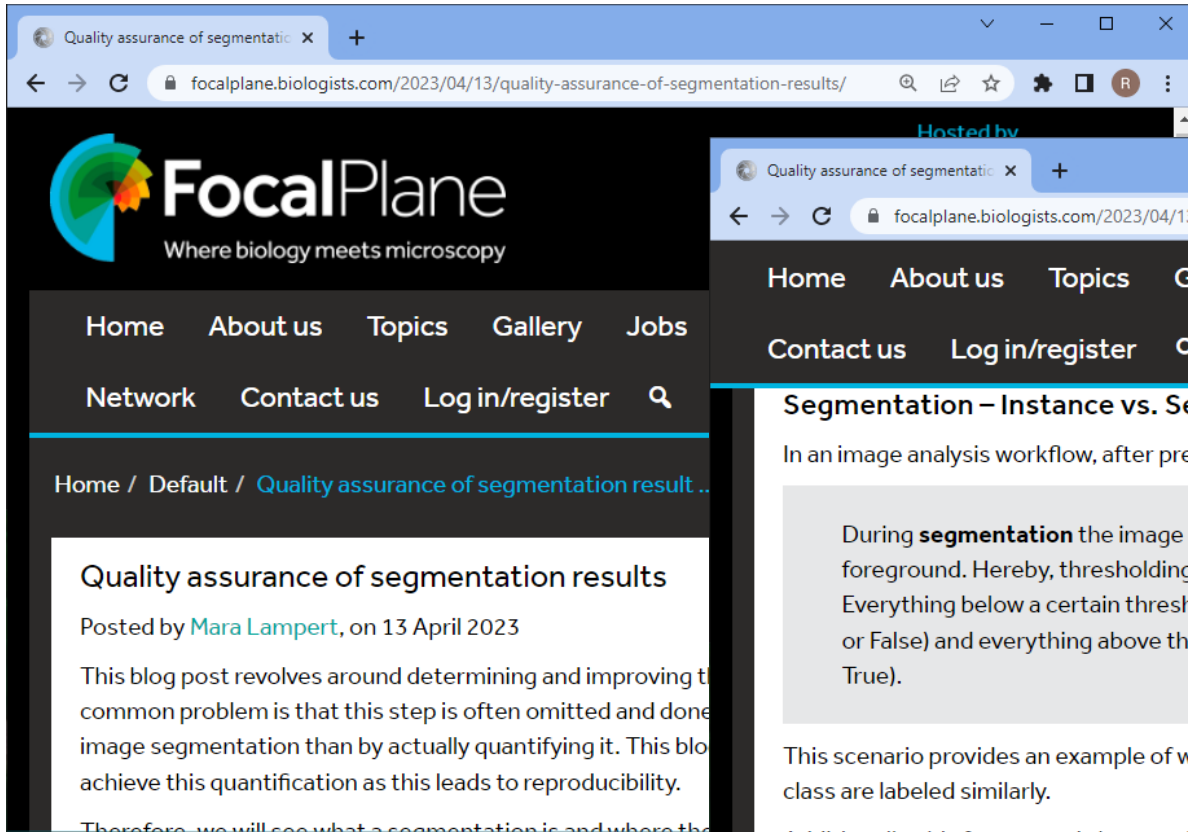
What metric to use when?

- “Metrics reloaded: Pitfalls and recommendations for image analysis validation”
Maier-Hein, Reinke et al. <https://arxiv.org/abs/2206.01653>



+ S1





Quality assurance of segmentatic x +

focalplane.biologists.com/2023/04/13/quality-assurance-of-segmentation-results/

Hosted by

FocalPlane
Where biology meets microscopy

Home About us Topics Gallery Jobs
Network Contact us Log in/register

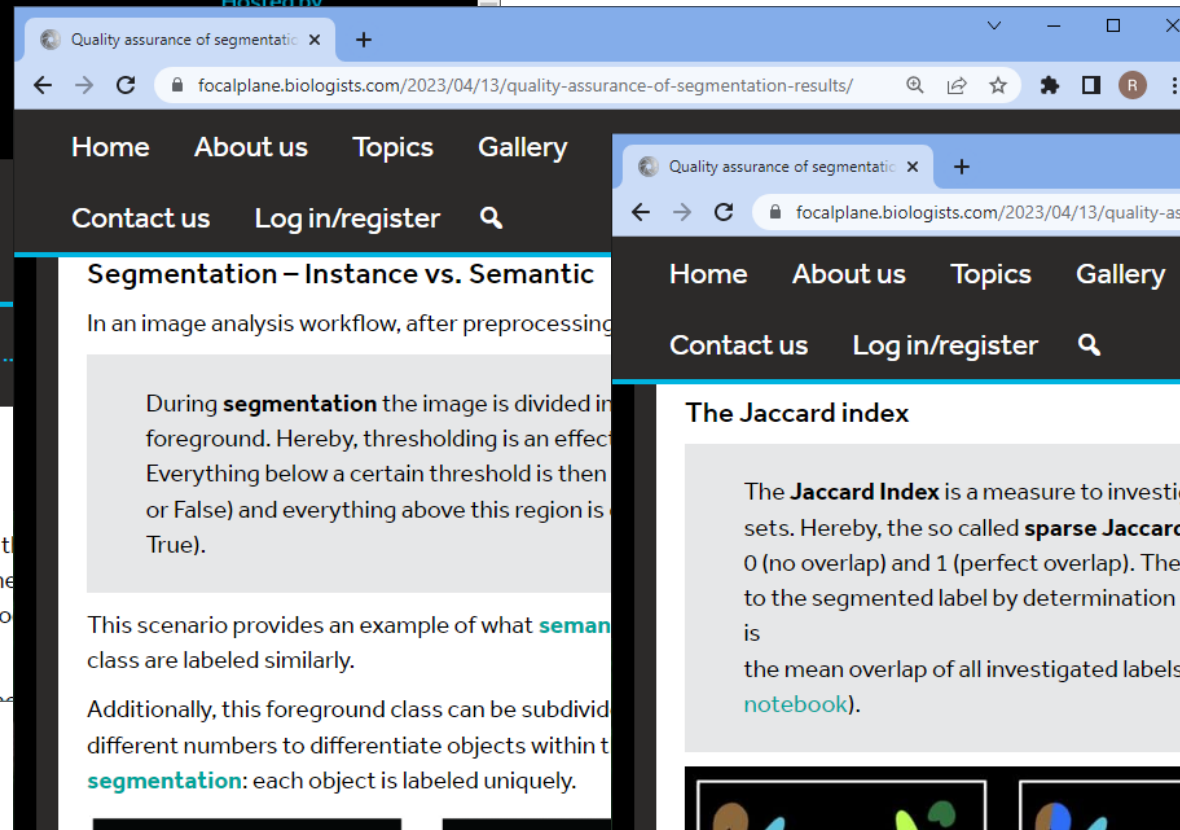
Home / Default / [Quality assurance of segmentation result...](#)

Quality assurance of segmentation results

Posted by [Mara Lampert](#), on 13 April 2023

This blog post revolves around determining and improving the quality of image segmentation. A common problem is that this step is often omitted and done visually instead of quantitatively. This blog post aims to achieve this quantification as this leads to reproducibility.

Therefore, we will see what a segmentation is and where the



Quality assurance of segmentatic x +

focalplane.biologists.com/2023/04/13/quality-assurance-of-segmentation-results/

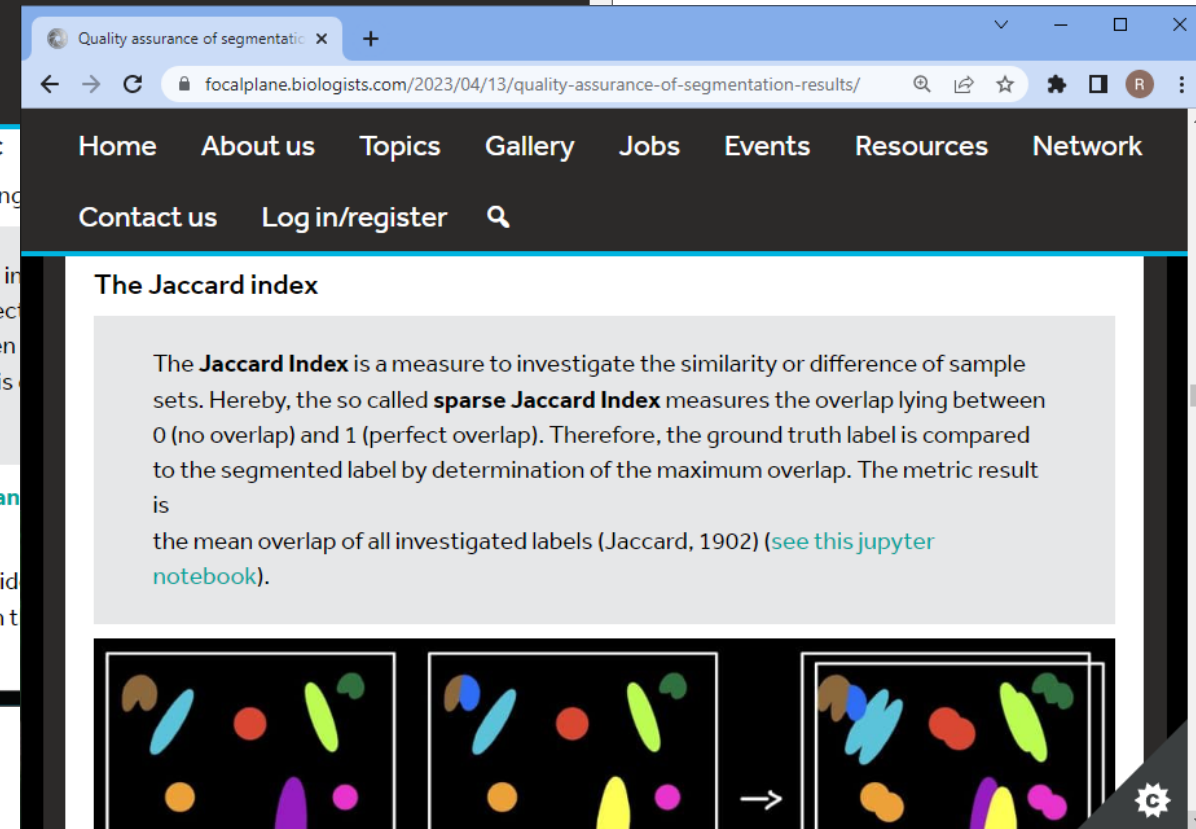
Home About us Topics Gallery
Contact us Log in/register

Segmentation – Instance vs. Semantic

In an image analysis workflow, after preprocessing the image, the next step is segmentation. During **segmentation** the image is divided into foreground and background. Hereby, thresholding is an effective method. Everything below a certain threshold is then considered as background (or False) and everything above this region is considered as foreground (or True).

This scenario provides an example of what **semantic segmentation** class are labeled similarly.

Additionally, this foreground class can be subdivided into different numbers to differentiate objects within the image. This is called **instance segmentation**: each object is labeled uniquely.



Quality assurance of segmentatic x +

focalplane.biologists.com/2023/04/13/quality-assurance-of-segmentation-results/

Home About us Topics Gallery Jobs Events Resources Network
Contact us Log in/register

The Jaccard index

The **Jaccard Index** is a measure to investigate the similarity or difference of sample sets. Hereby, the so called **sparse Jaccard Index** measures the overlap lying between 0 (no overlap) and 1 (perfect overlap). Therefore, the ground truth label is compared to the segmented label by determination of the maximum overlap. The metric result is the mean overlap of all investigated labels (Jaccard, 1902) (see [this jupyter notebook](#)).

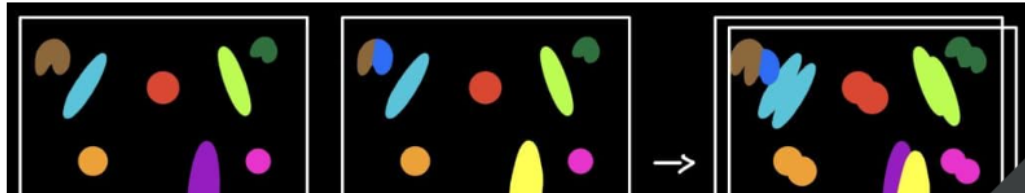


Image segmentation in Napari

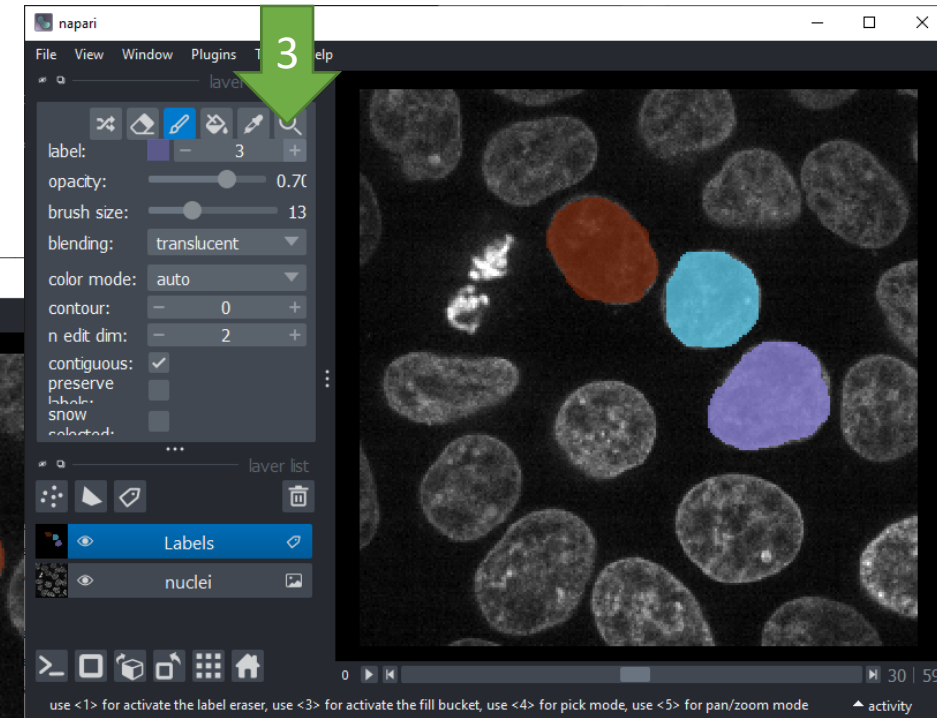
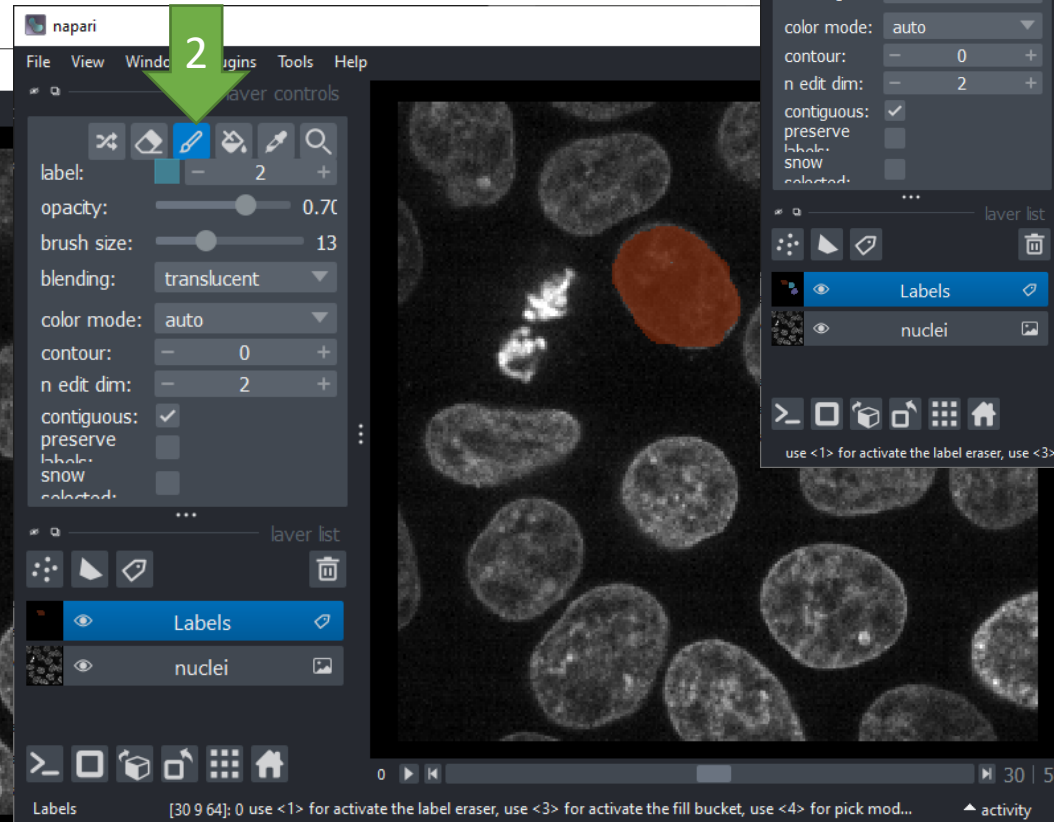
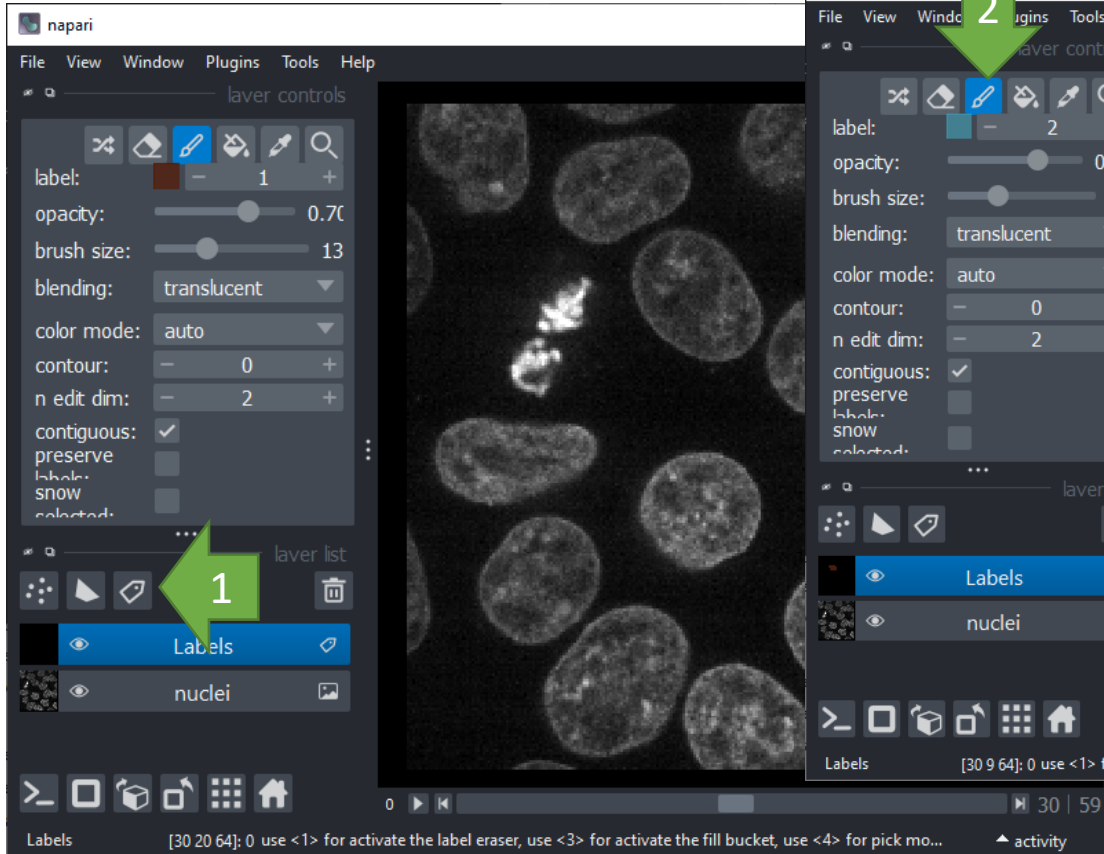
Robert Haase

Using materials from
Ryan Savill George, MPI CBG Dresden

April 2023

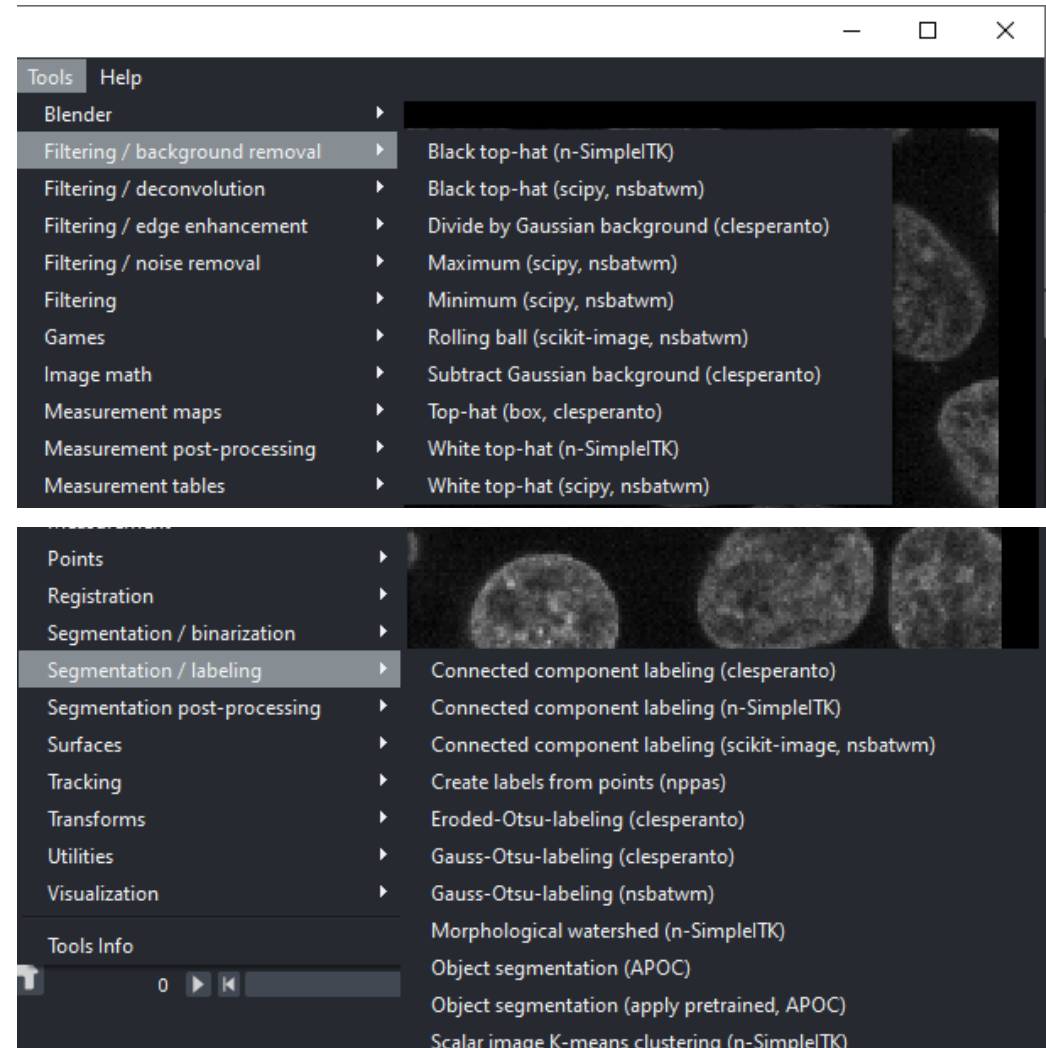
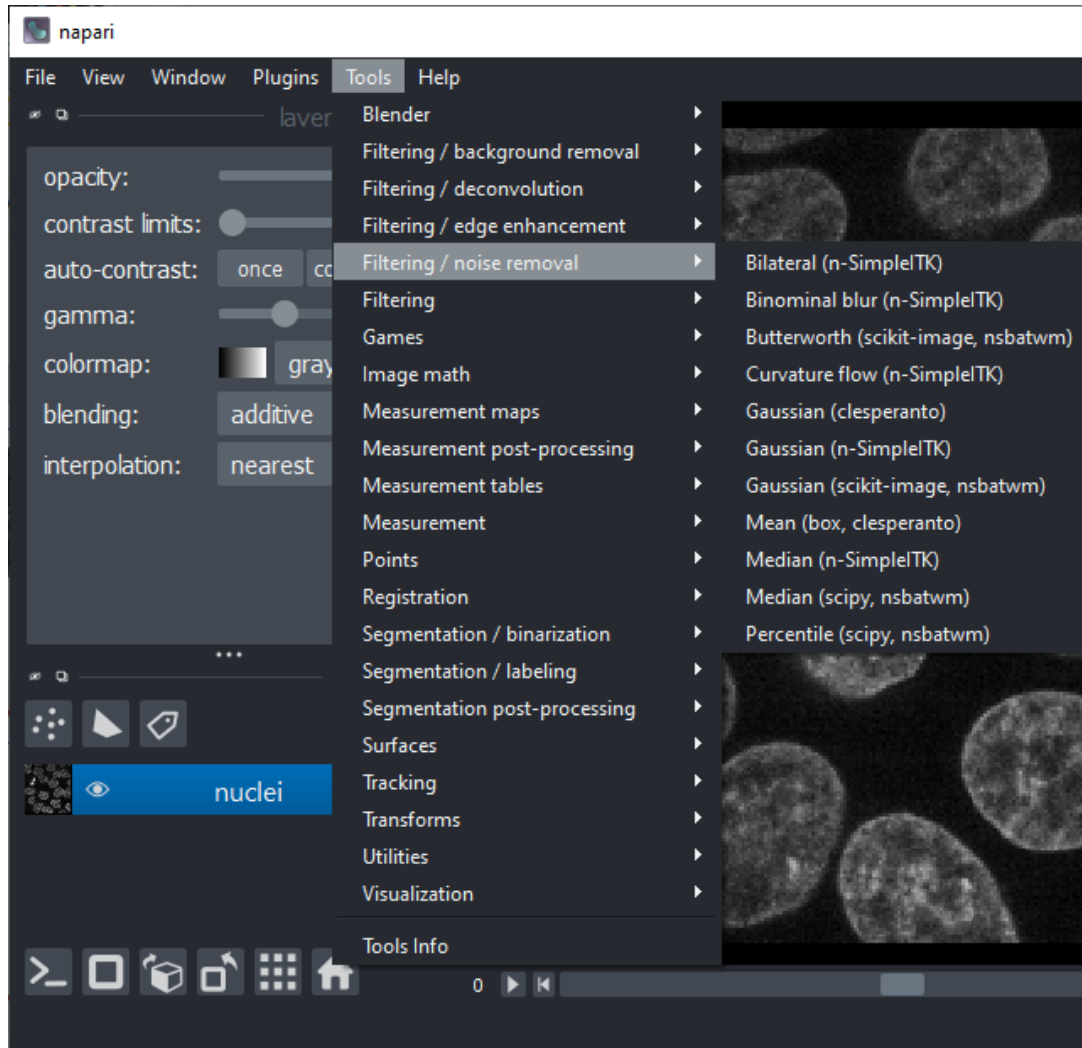
Manual annotation

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



The Tools menu

- Organized in categories similar to what you learned last week



The Napari Assistant

- Tools > Utilities > Assistant (na)

Viewer controls

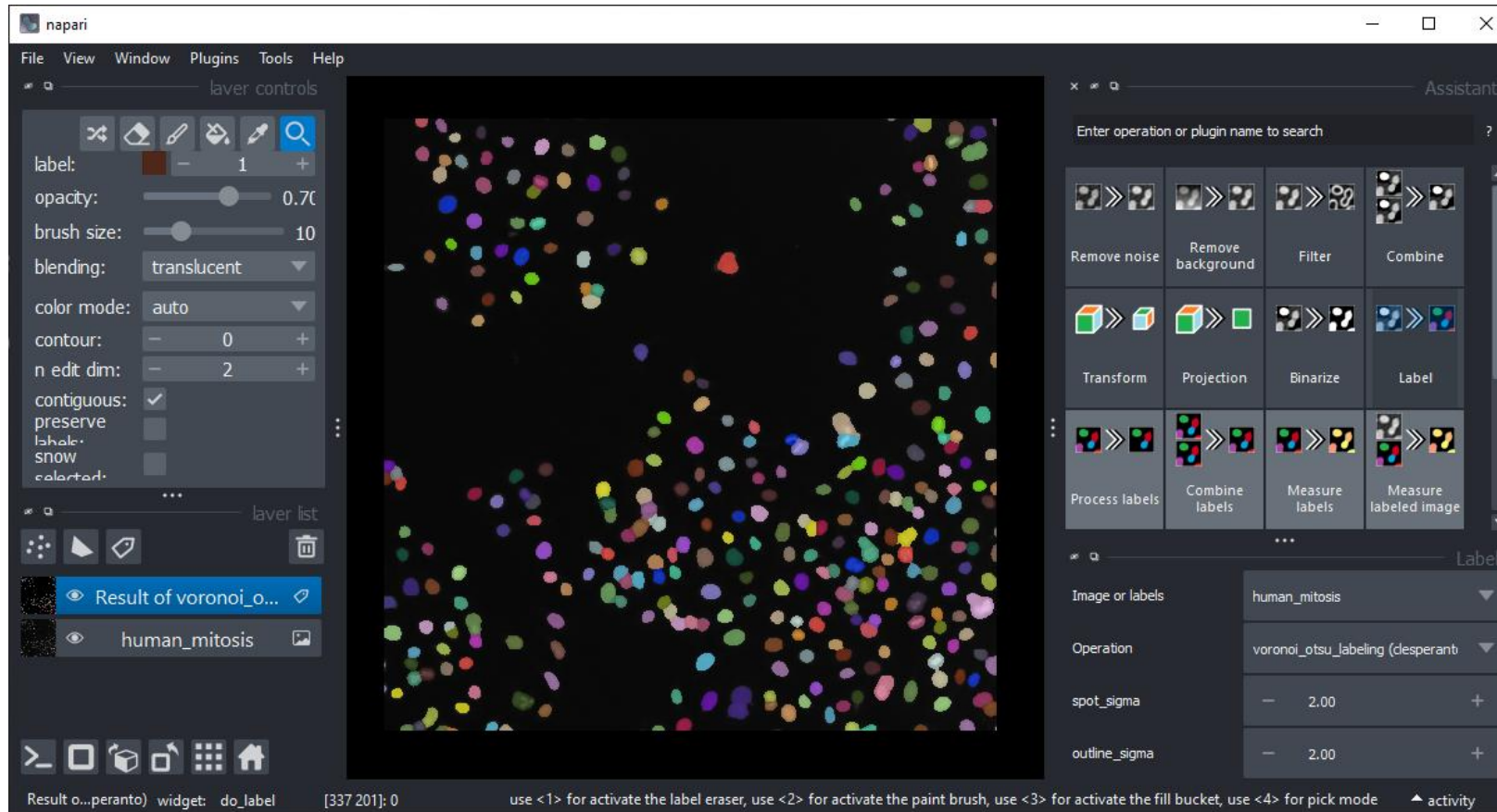
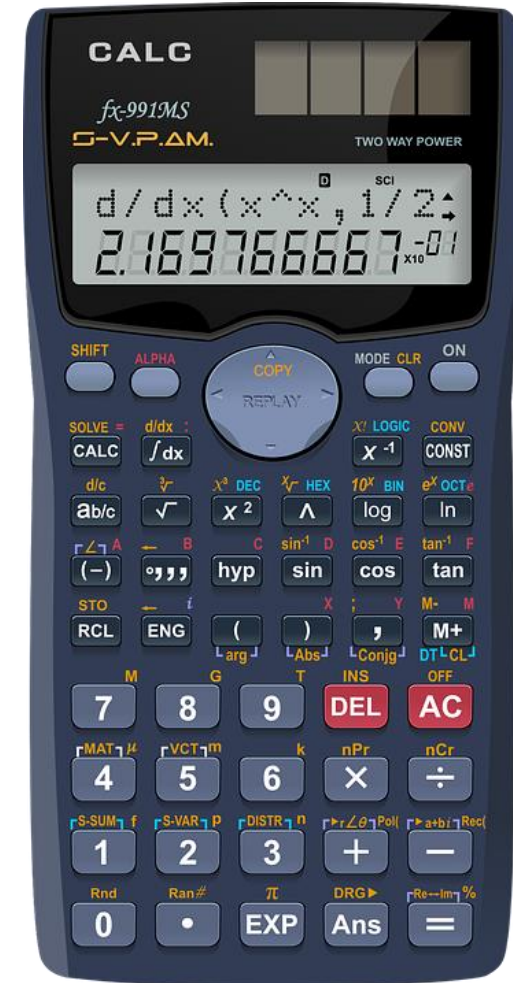
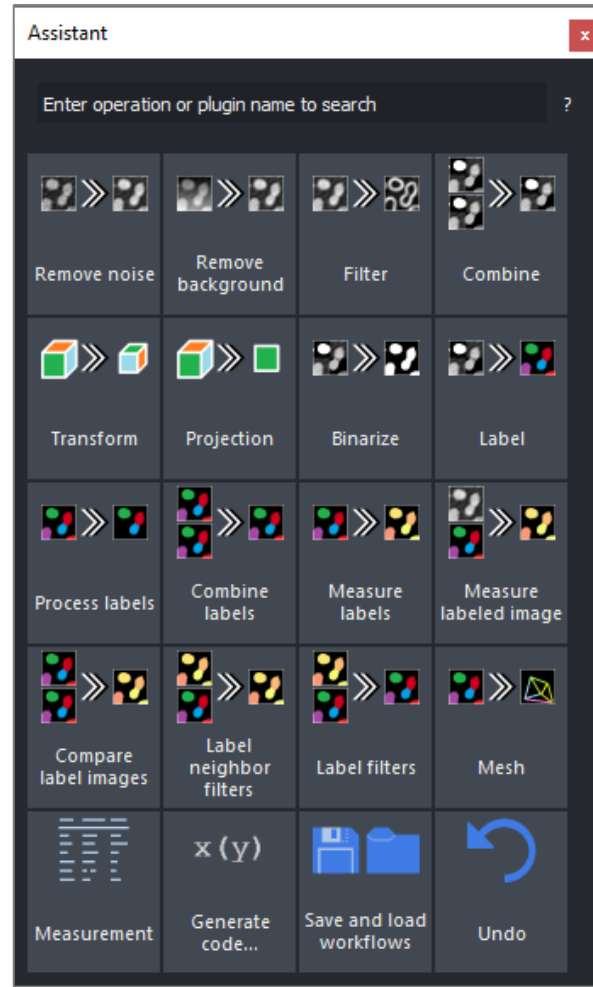


Image Processing

The Napari Assistant

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



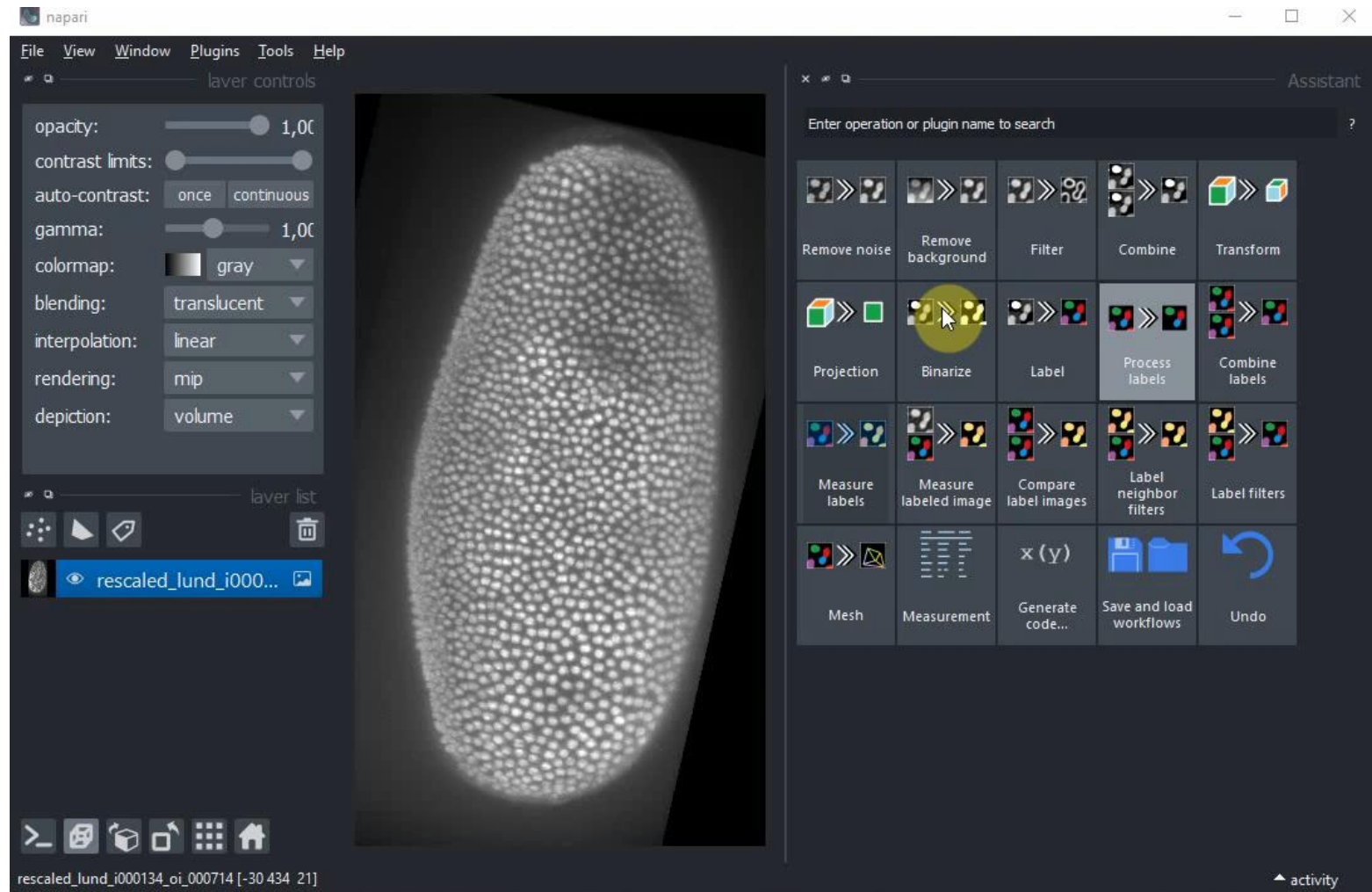
The Napari Assistant

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

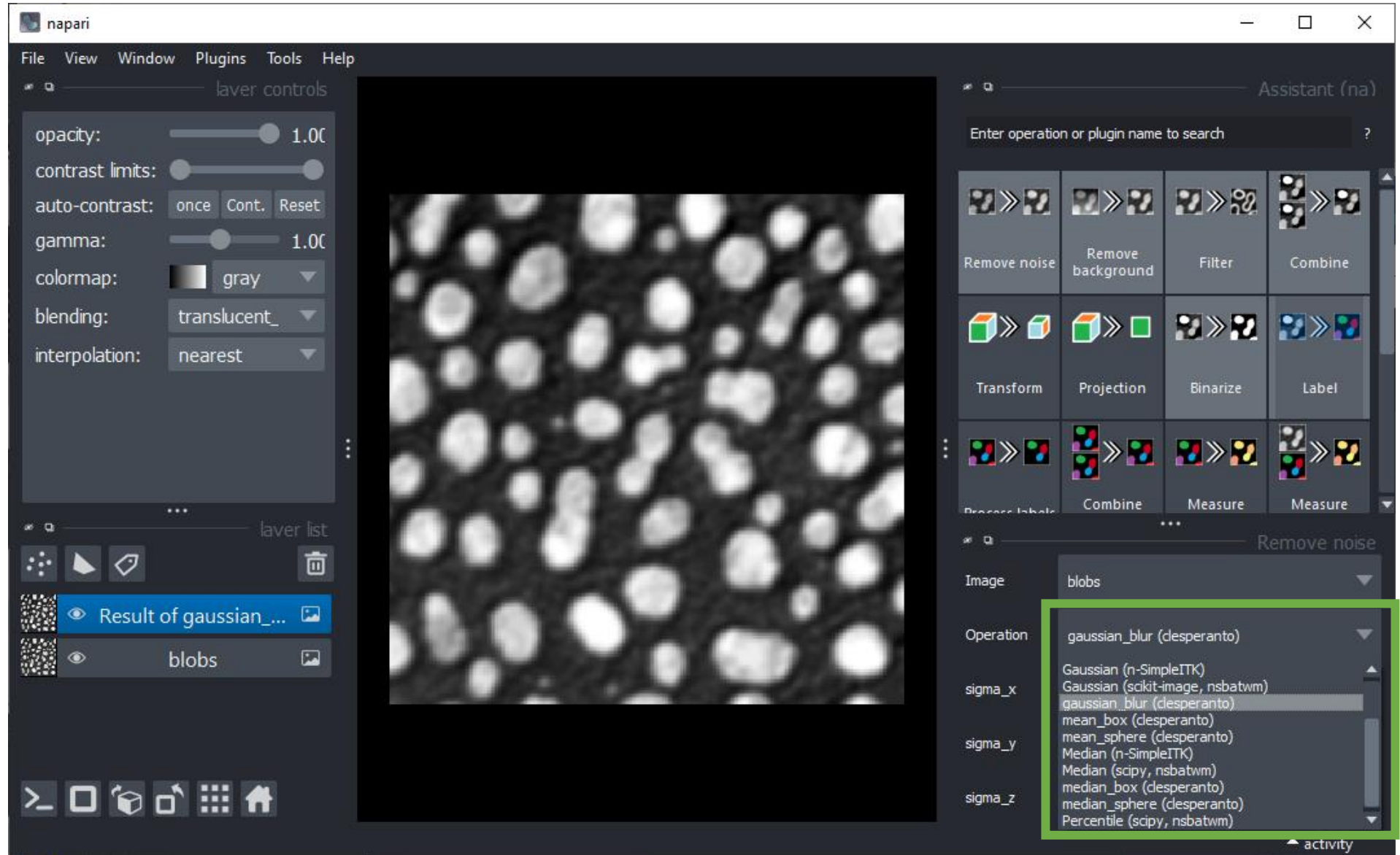
Big thanks to:



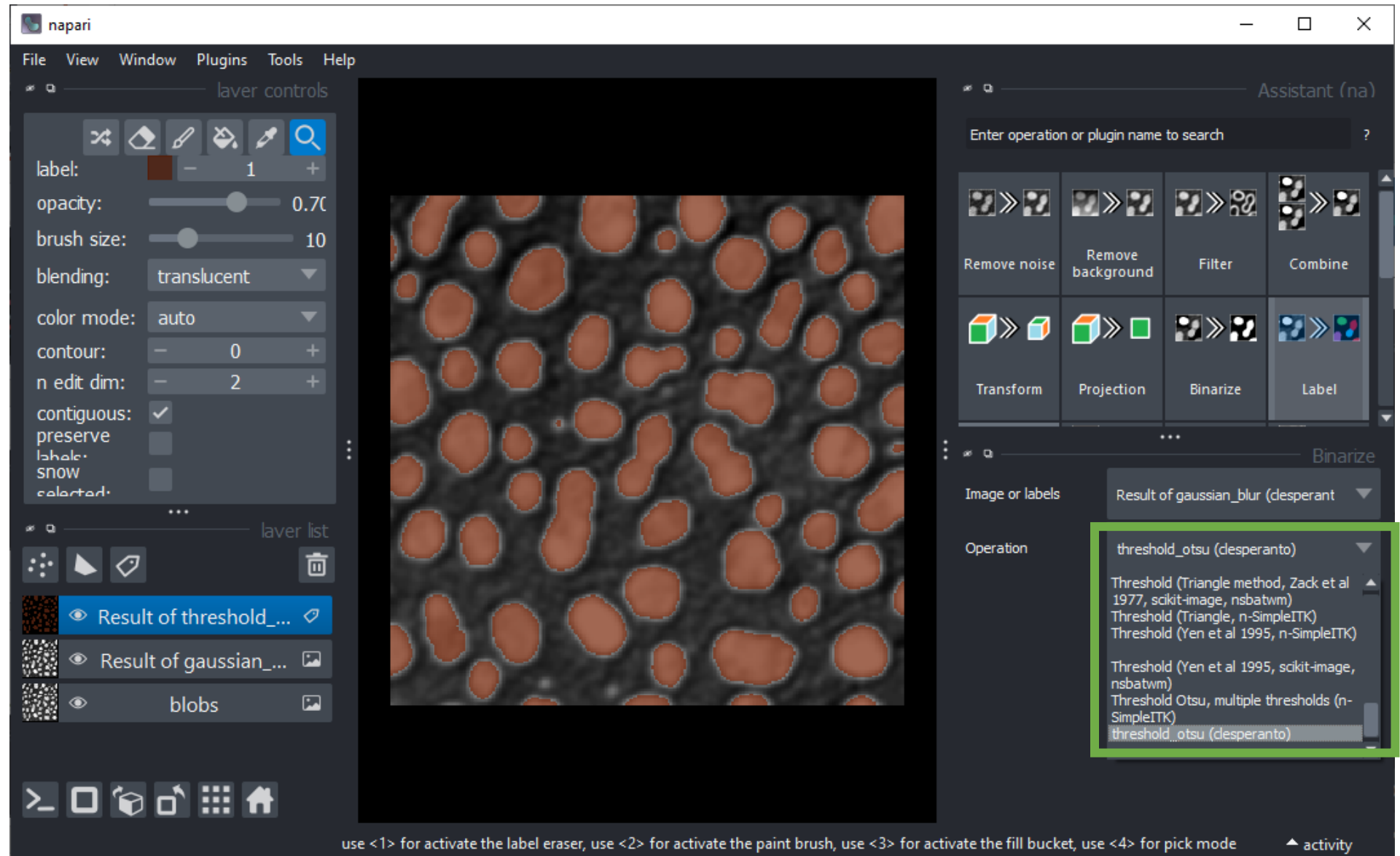
Ryan Savill
@RyanSavill4



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

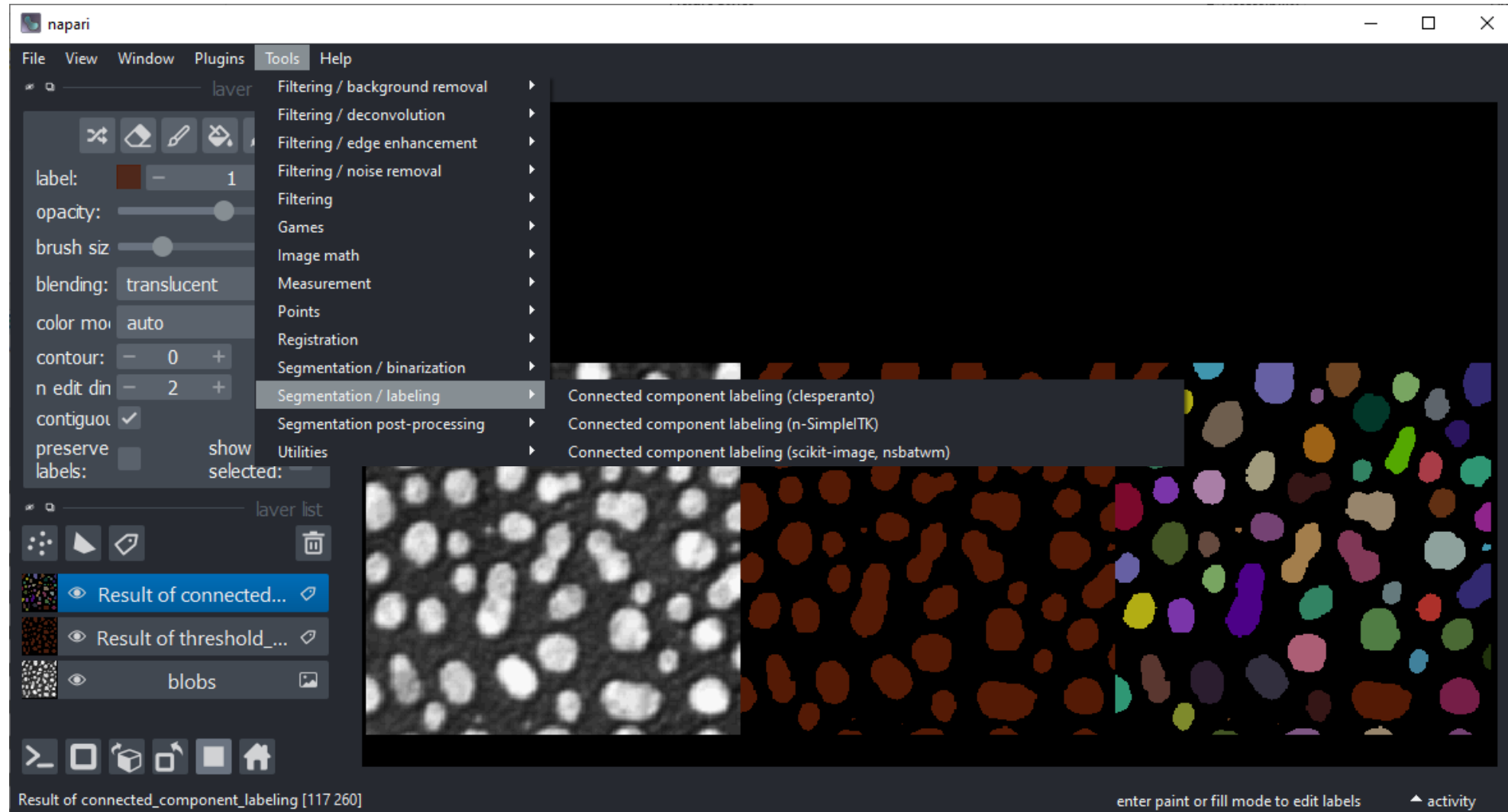


- Try different binarization algorithms

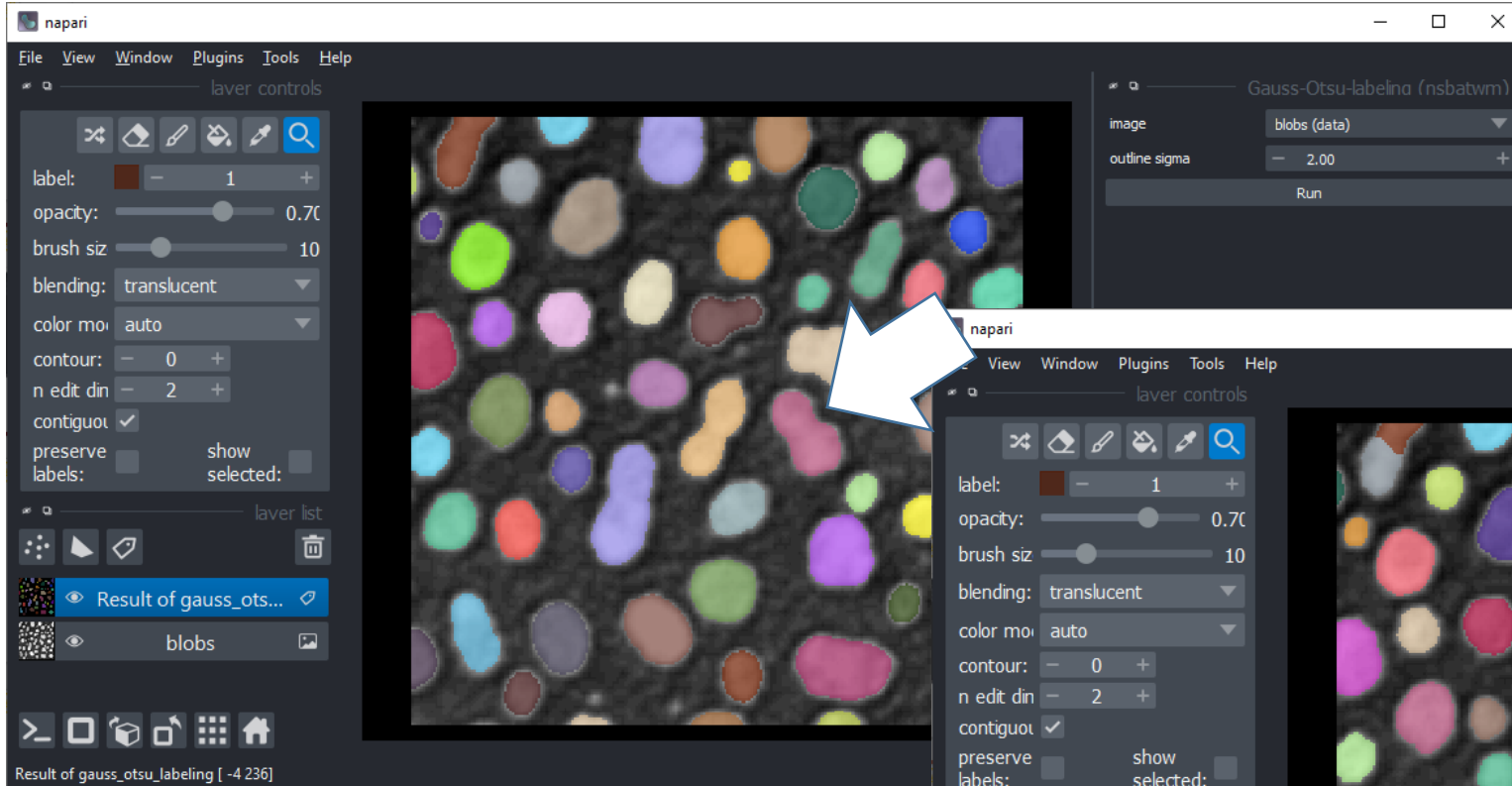


The screenshot shows the napari software interface with a central image of a cell-like pattern. On the left, the 'layer controls' panel is visible, showing settings for a layer named 'blobs'. On the right, the 'Assistant' panel is open, displaying a grid of operations. The 'Binarize' operation is selected, and a dropdown menu is open, showing a list of binarization algorithms. The 'threshold_otsu (desperanto)' algorithm is highlighted with a green box. At the bottom of the interface, there is a footer with keyboard shortcuts: 'use <1> for activate the label eraser, use <2> for activate the paint brush, use <3> for activate the fill bucket, use <4> for pick mode' and an 'activity' icon.

Also check out the Tools > Segmentation / labeling menu

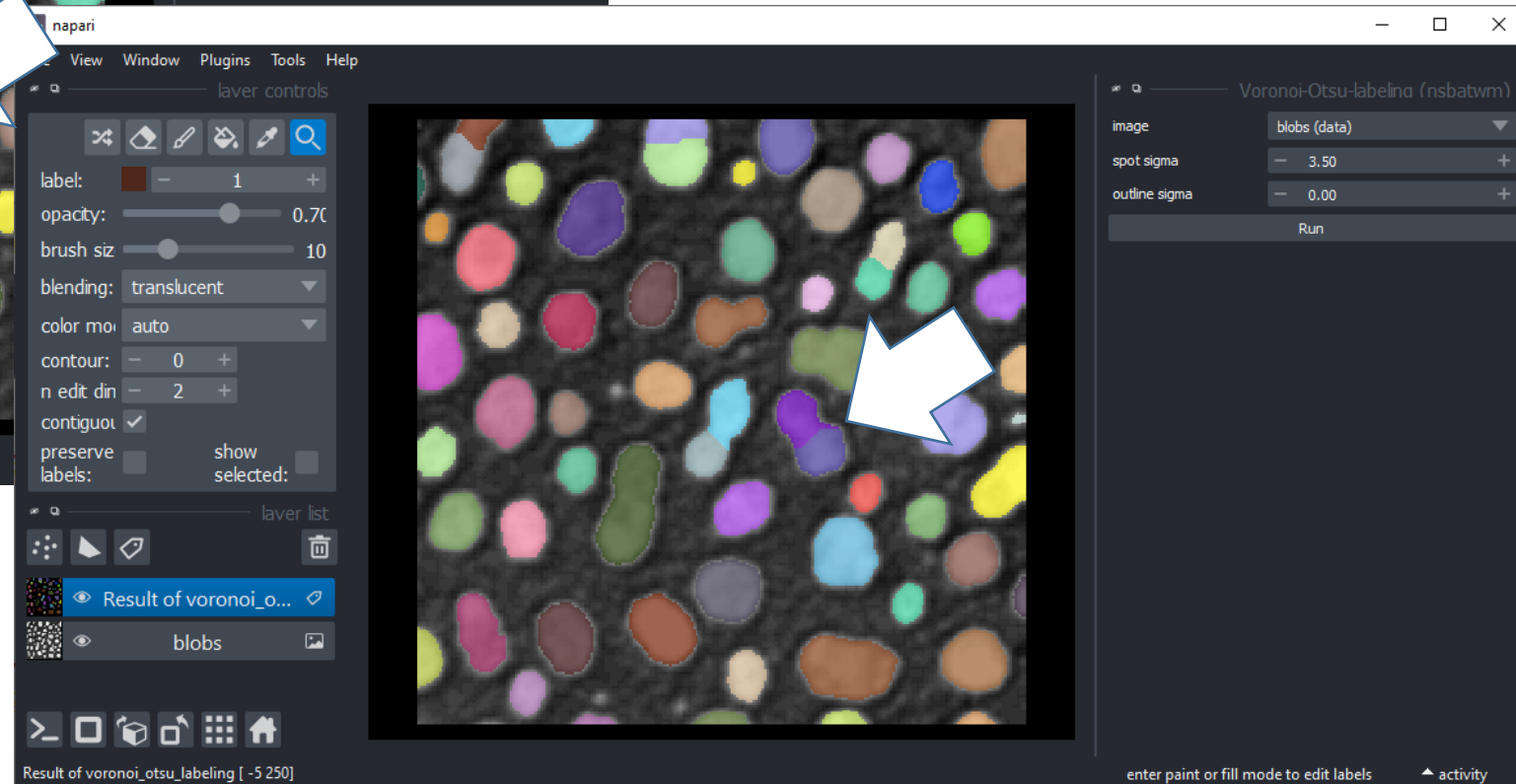


Also check out the Tools > Segmentation / labeling menu

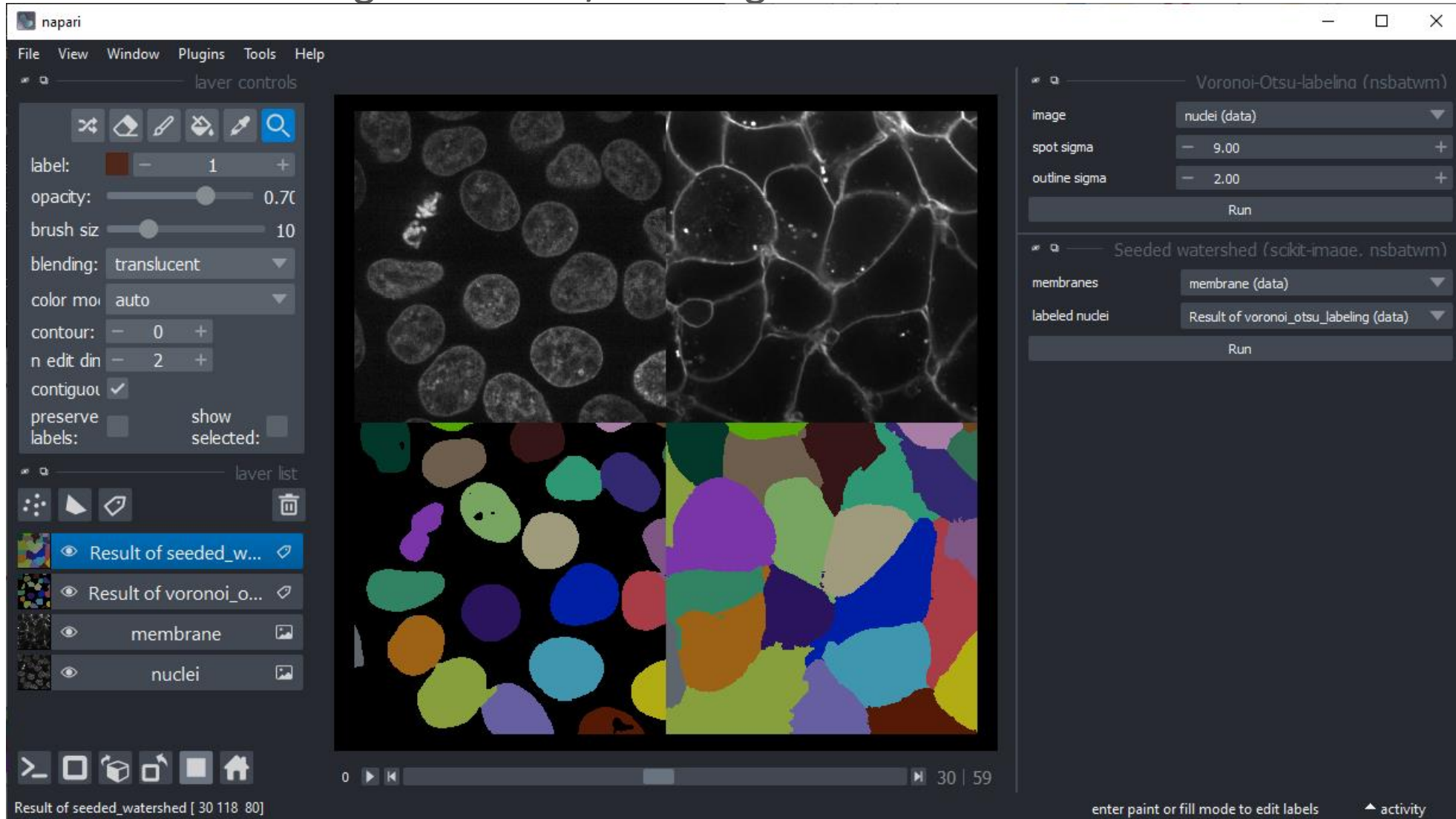


Gauss-Otsu-Labeling

Voronoi-Otsu-Labeling

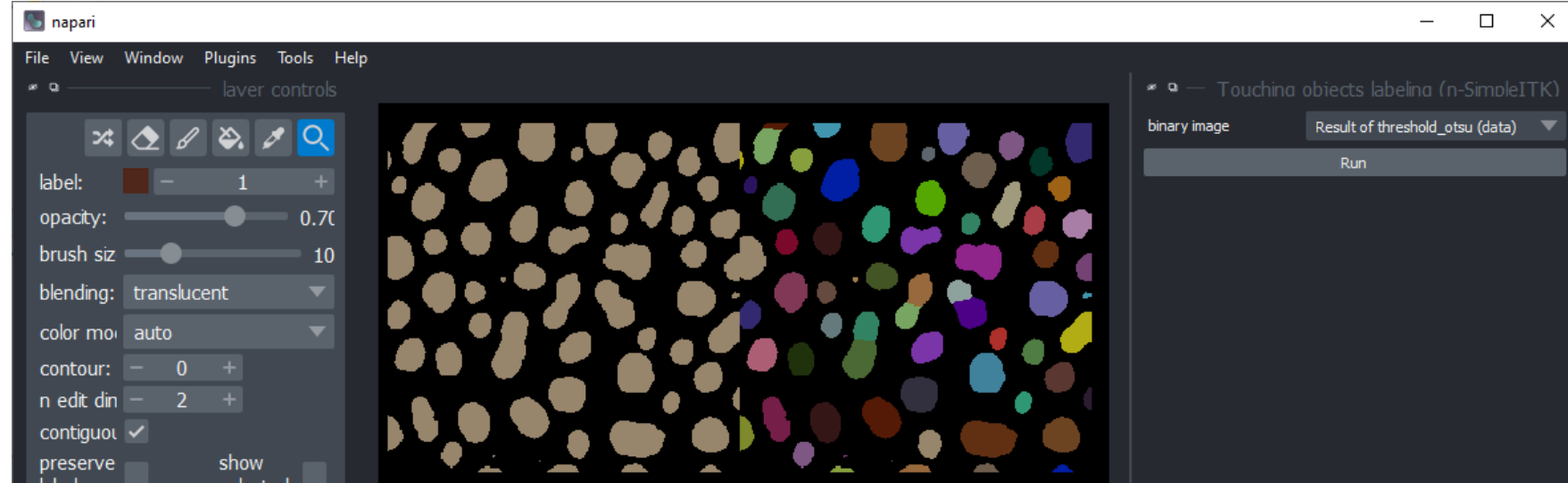


Also check out the Tools > Segmentation / labeling menu



- From binary images

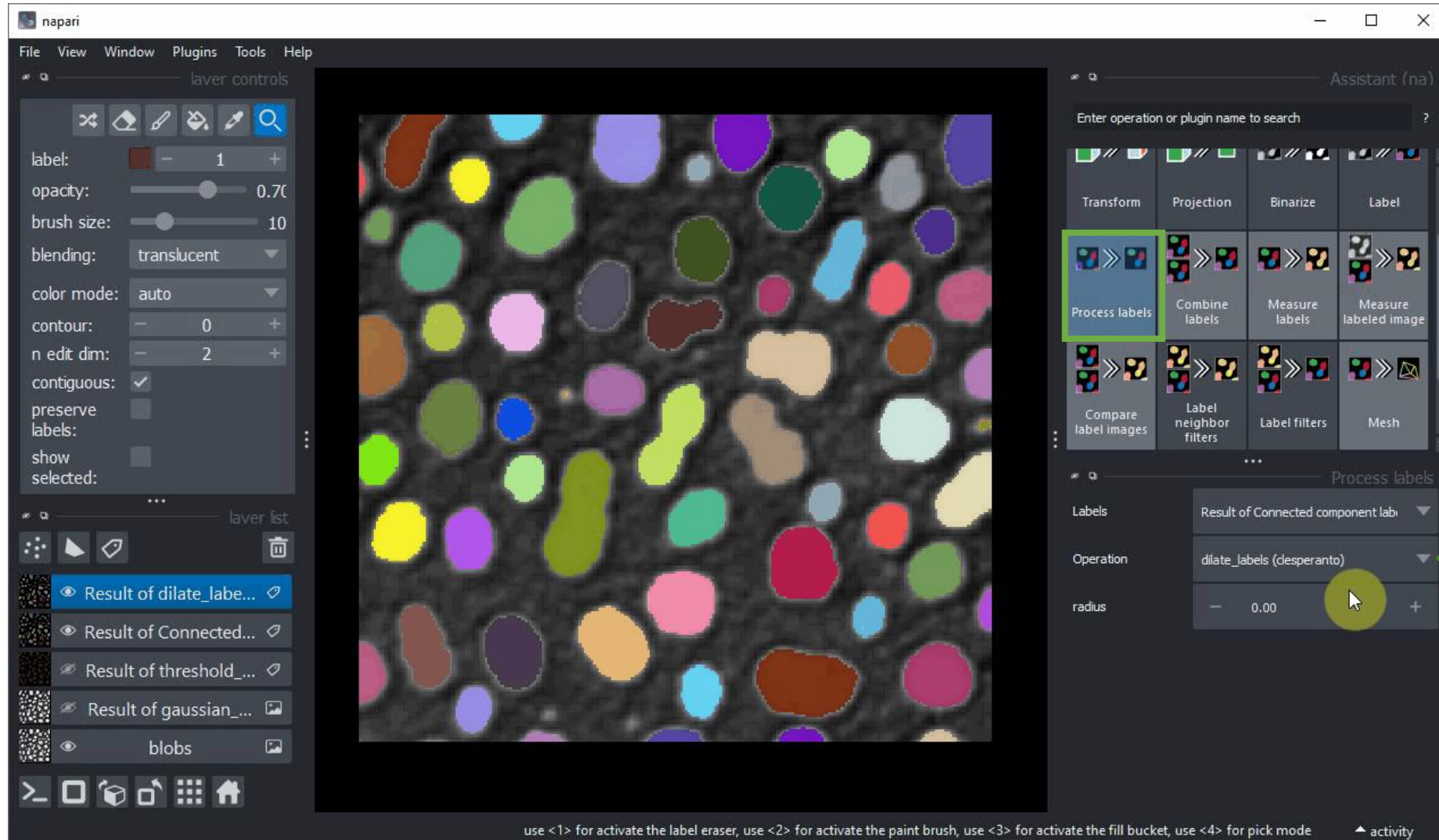
Tools > Segmentation / labeling >
Label touching objects



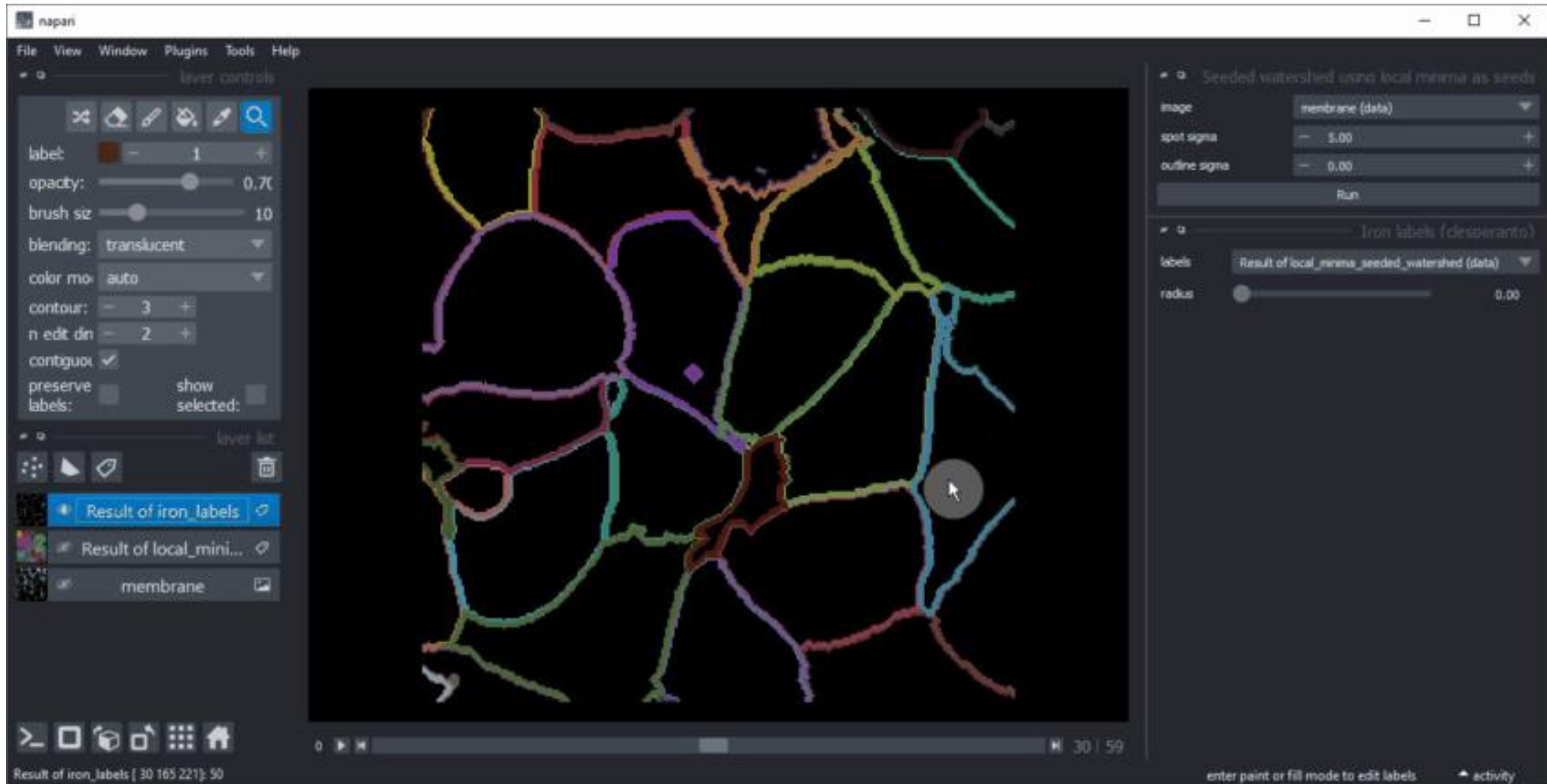
Tools > Segmentation post-
processing >
Split touching objects
(Similar to ImageJ's Watershed)



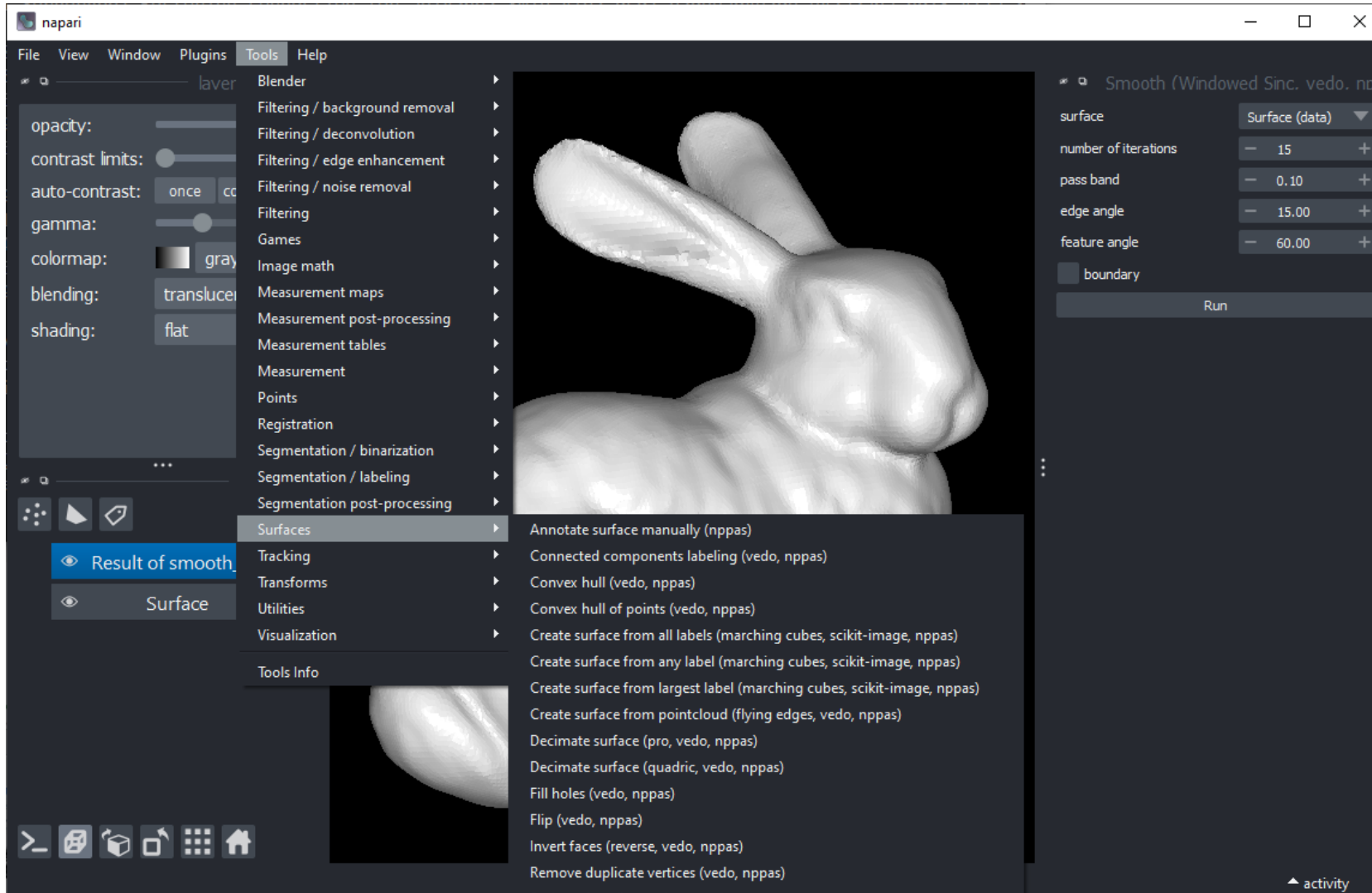
- In Napari Assistant: Process labels



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



- Tools > Surfaces > Create surface ...



You need to install an extra napari-plugin:
<https://github.com/haesleinhuepf/napari-process-points-and-surfaces>

Browse operations

- Use the search...

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

Enter the library name you want to use

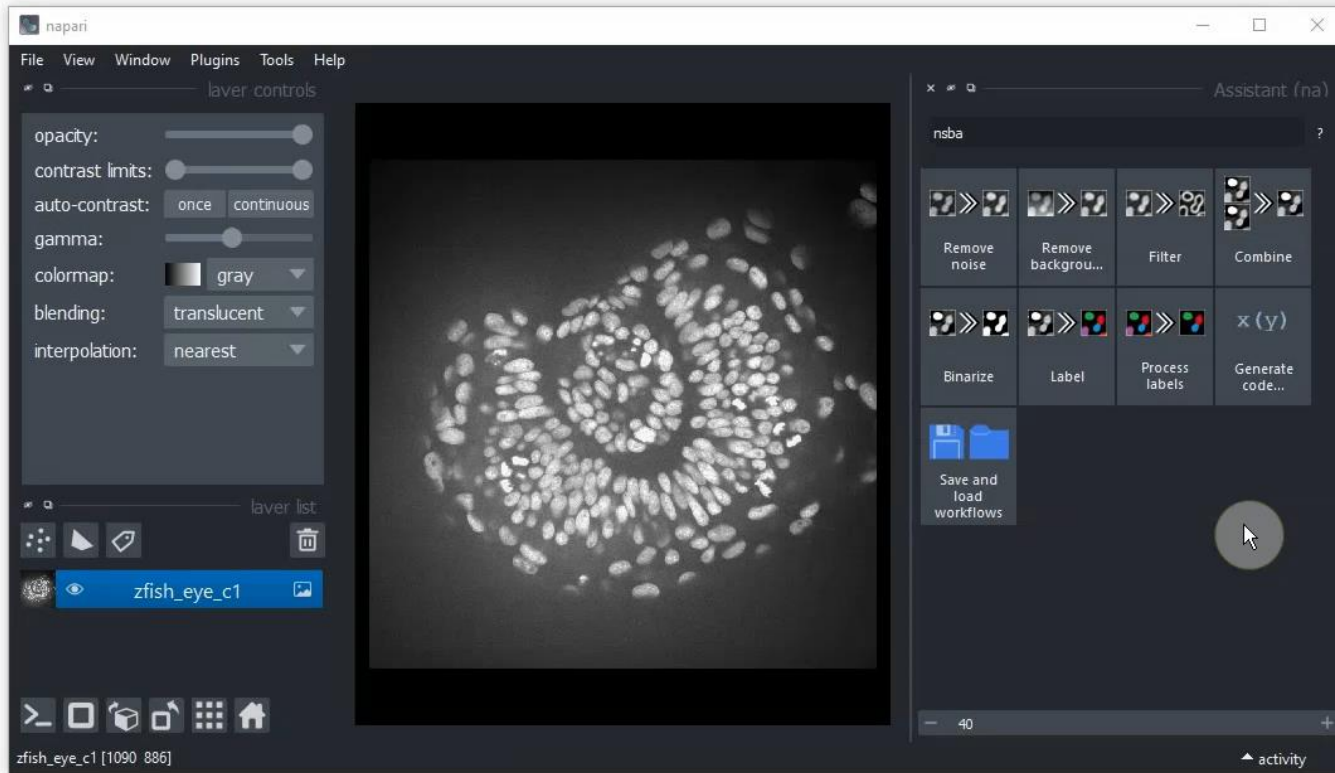
Enter the structure you would like to segment

Search the internet

The image displays four sequential screenshots of the Napari Assistant search interface, illustrating the process of finding operations based on different search criteria:

- Search 1:** The search bar contains the text "Enter operation or plugin name to search". The results grid shows various image processing operations such as "Remove noise", "Remove background", "Filter", "Combine", "Transform", "Projection", "Binarize", "Label", "Process labels", "Combine labels", "Measure labels", and "Measure labeled image".
- Search 2:** The search bar contains "scikit-ima". The results grid highlights operations from the scikit-image library, including "Remove noise", "Remove background", "Binarize", "Label", "Process labels", "Measure...", "Generate code...", "Save and load workflows", and "Undo". A tooltip for "Measure..." explains: "Measure features and show results in a table. Operations: * Regionprops (scikit-image, nsr)".
- Search 3:** The search bar contains "membrane". The results grid shows "Remove background", "Label", "Generate code...", "Save and load workflows", and "Undo". A tooltip for "Label" explains: "Turn images into label images by labeling objects. Operations: * Seeded watershed using local minima as seeds, * Seeded watershed using local minima as seeds".
- Search 4:** The search bar contains "the unknown". The results grid shows search options: "Search napari hub", "Search image.sc", and "Search Bili".

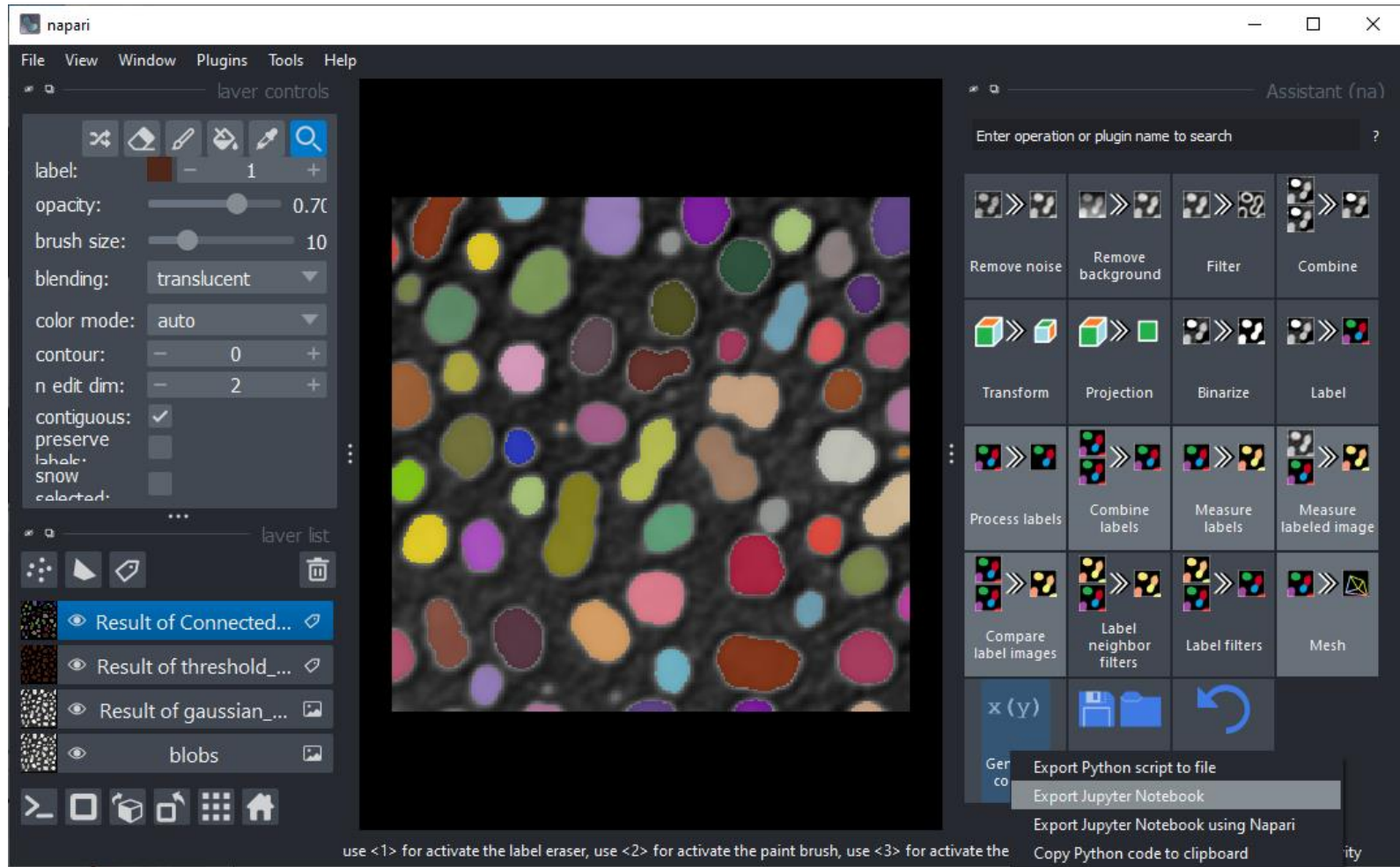
Export code to Jupyter Notebooks



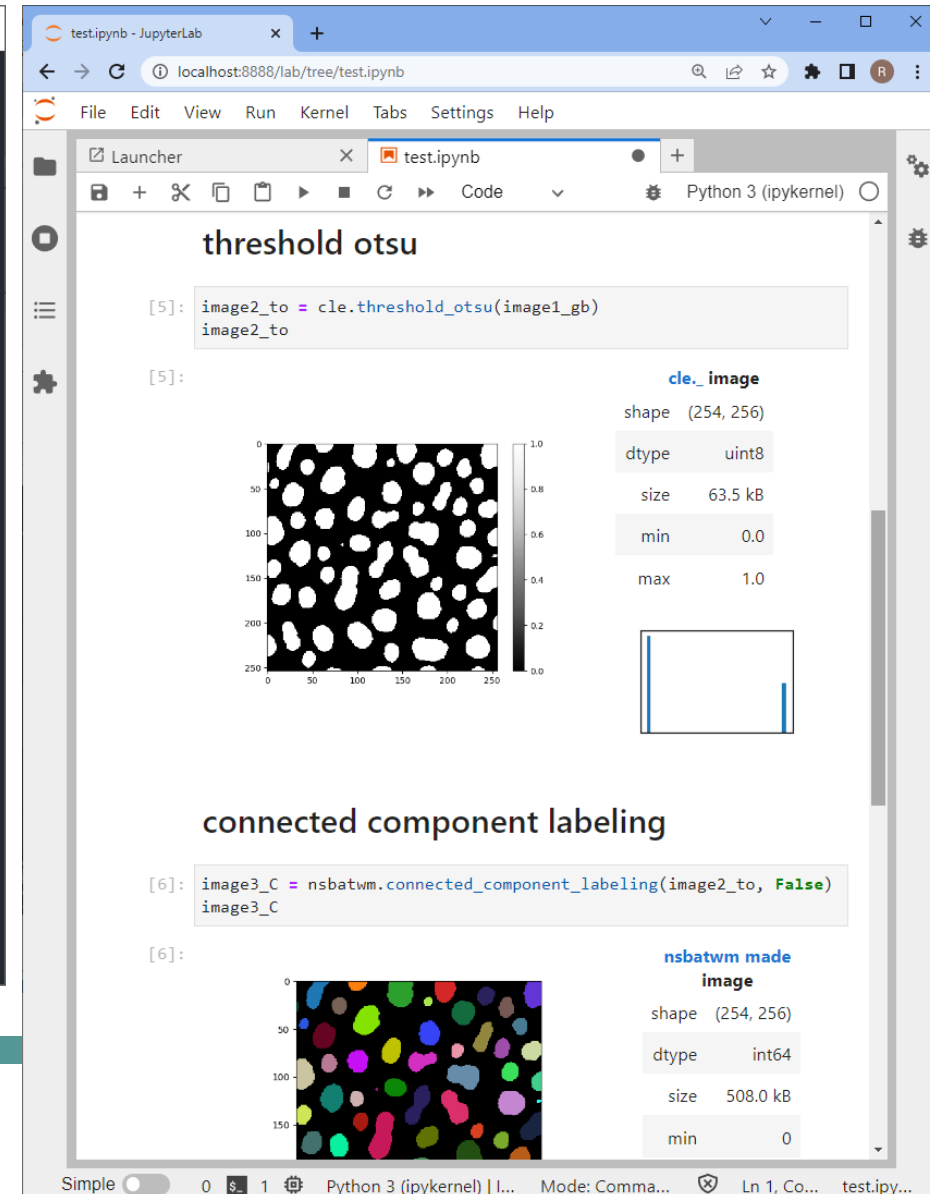
<https://github.com/haesleinhuepf/napari-assistant>

Image data source: Mauricio Rocha Martins, Norden lab, MPI CBG (now at IGC Oeiras)

Export code to Jupyter Notebooks



The screenshot shows the Napari Assistant interface. On the left is the Napari layer controls panel with various tool icons and settings like label, opacity, brush size, and blending. The center displays a multi-colored segmented image of cells. On the right is the Assistant panel with a search bar and a grid of operations such as 'Remove noise', 'Remove background', 'Filter', 'Combine', 'Transform', 'Projection', 'Binarize', 'Label', 'Process labels', 'Combine labels', 'Measure labels', 'Measure labeled image', 'Compare label images', 'Label neighbor filters', 'Label filters', and 'Mesh'. A context menu is open over the Assistant panel, listing options: 'Export Python script to file', 'Export Jupyter Notebook', 'Export Jupyter Notebook using Napari', and 'Copy Python code to clipboard'. At the bottom, a note reads: 'use <1> for activate the label eraser, use <2> for activate the paint brush, use <3> for activate the'.



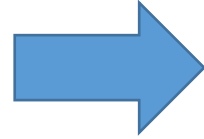
The screenshot shows a JupyterLab browser window. The top bar indicates 'testipynb - JupyterLab' and the URL is 'localhost:8888/lab/tree/test.ipynb'. The main area contains two code cells. The first cell, titled 'threshold otsu', contains the code `image2_to = cle.threshold_otsu(image1_gb)` and `image2_to`. The output shows a binary image with white spots on a black background and a metadata table for `cle_image` with shape (254, 256), dtype uint8, size 63.5 kB, min 0.0, and max 1.0. The second cell, titled 'connected component labeling', contains the code `image3_C = nsbatwm.connected_component_labeling(image2_to, False)` and `image3_C`. The output shows a multi-colored segmented image and a metadata table for `nsbatwm made image` with shape (254, 256), dtype int64, size 508.0 kB, and min 0. The bottom status bar shows 'Simple' mode, 'Python 3 (ipykernel) | 1...', and 'Ln 1, Co... test.ipynb'.

Image segmentation in Python

Robert Haase

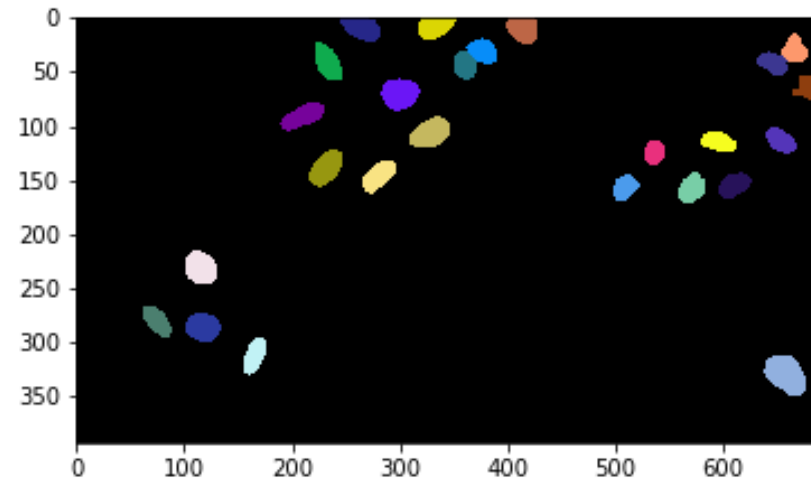
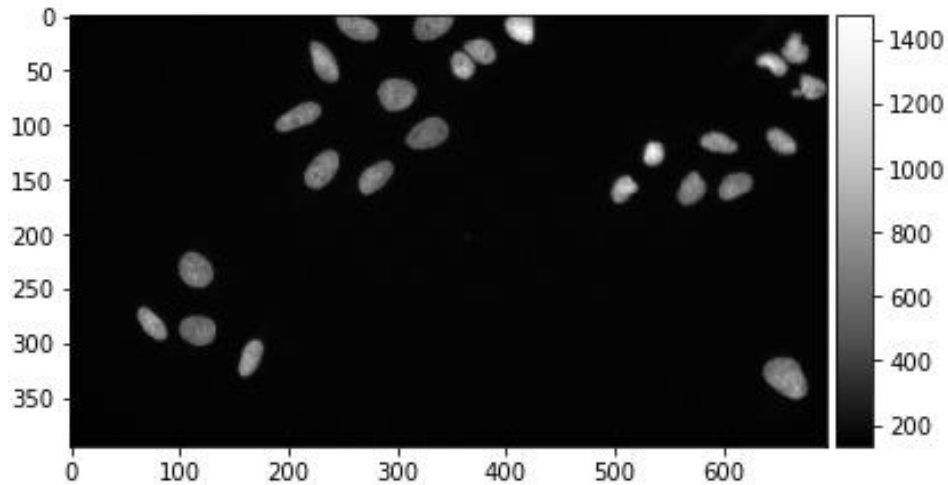
April 2023

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



... in a single line of code:

```
segmented = nsbatwm.voronoi_otсу_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

- 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

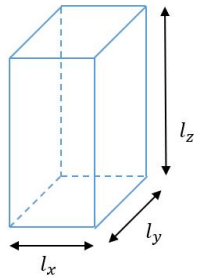
 Output image

spot sigma : float, optional

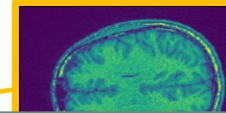
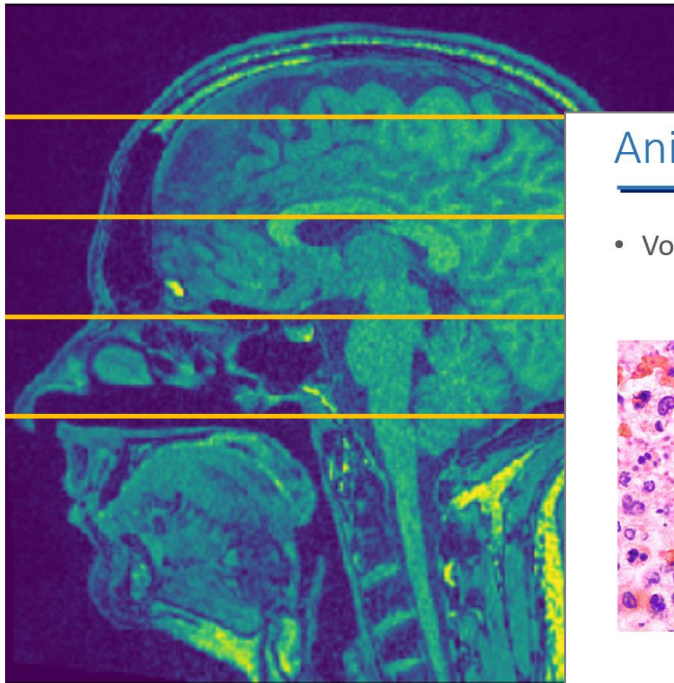
- Reminder: Anisotropic images might be tricky to process properly

Image stacks and voxels

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



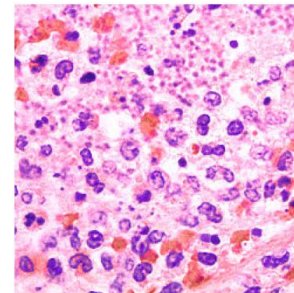
$$l_x = l_y \neq l_z$$



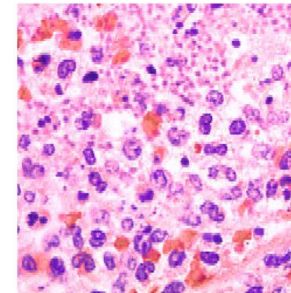
Anisotropy



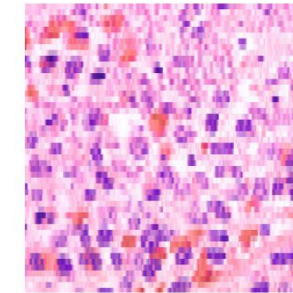
- Voxel size has immediate impact on image quality and thus, on processing / analysis results.



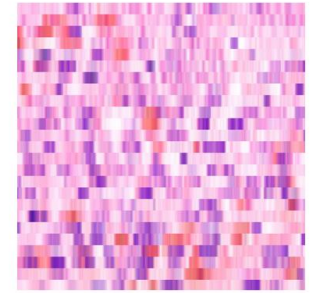
1:1
250 x 250 px



1:2
250 x 125 px



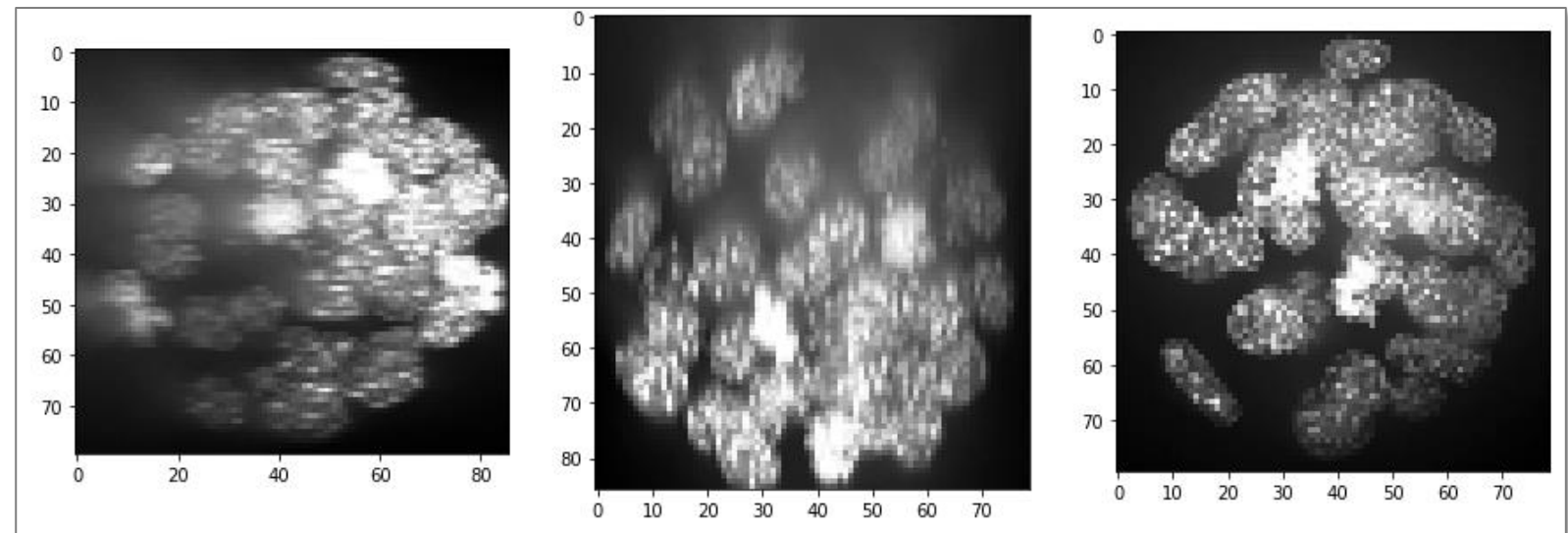
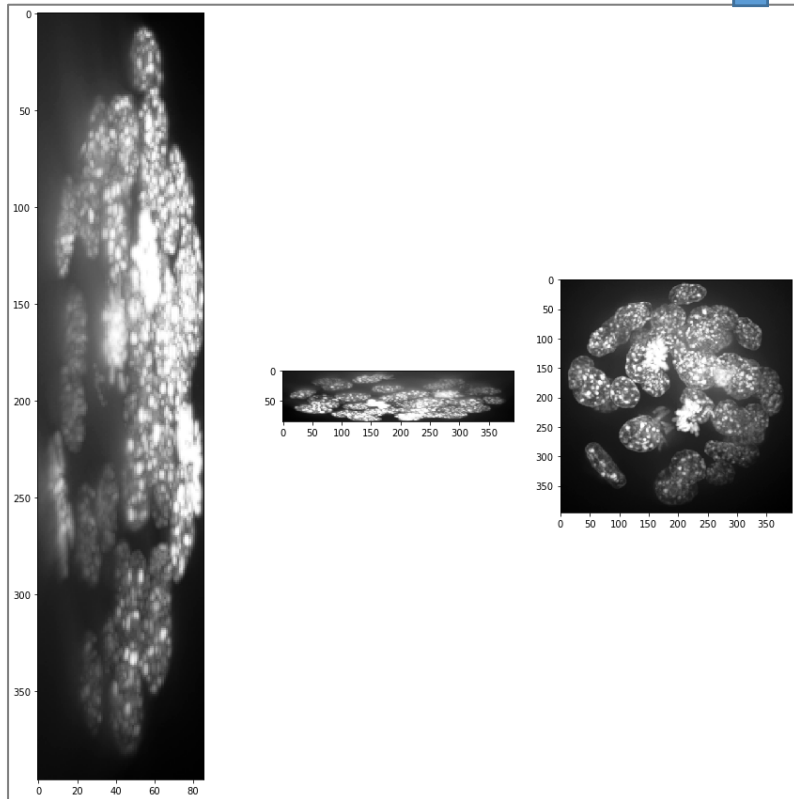
1:5
250 x 50 px



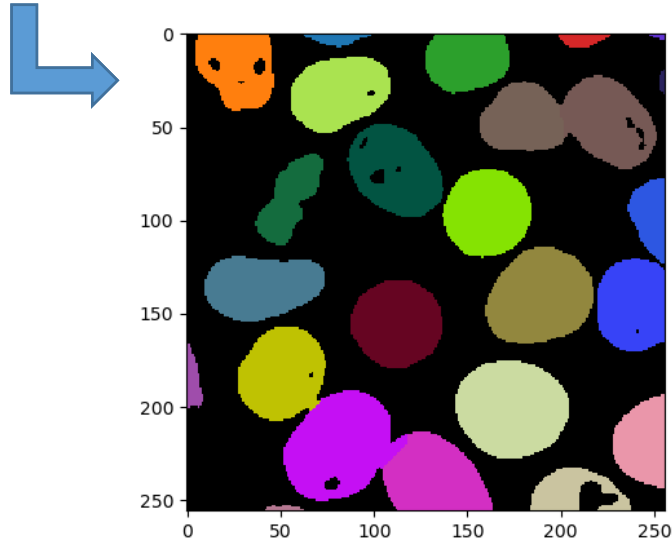
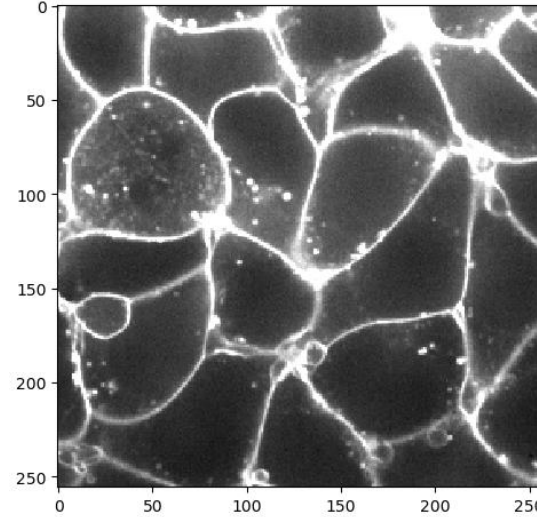
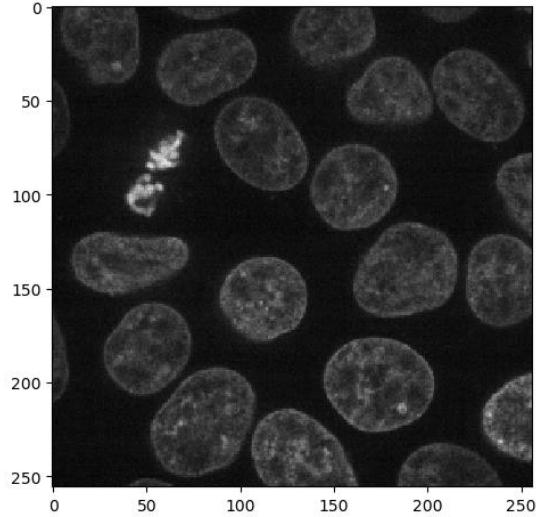
1:10
250 x 25 px

- Resample image data to a specific voxel size

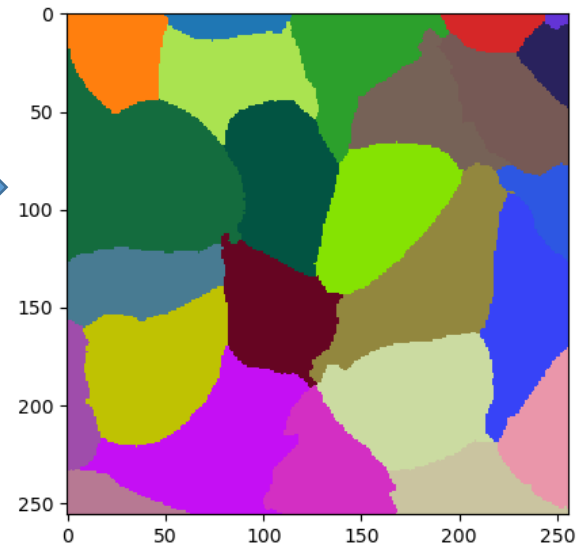
```
resampled = cle.scale(input_image, factor_x=voxel_size_x, factor_y=voxel_size_y, factor_z=voxel_size_z, auto_size=True)  
show(resampled)
```



- ... in Python practice

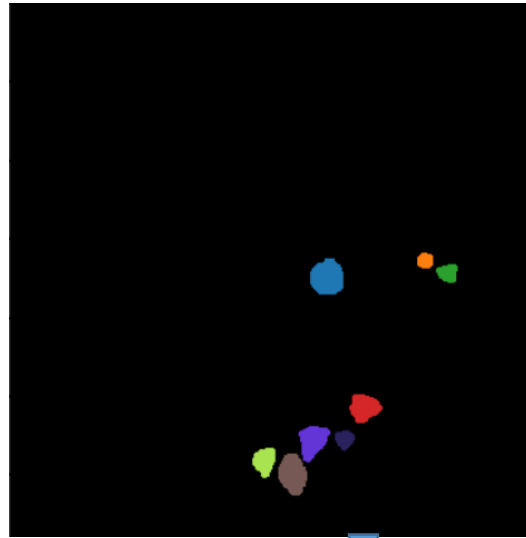


```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)  
labeled_cells
```

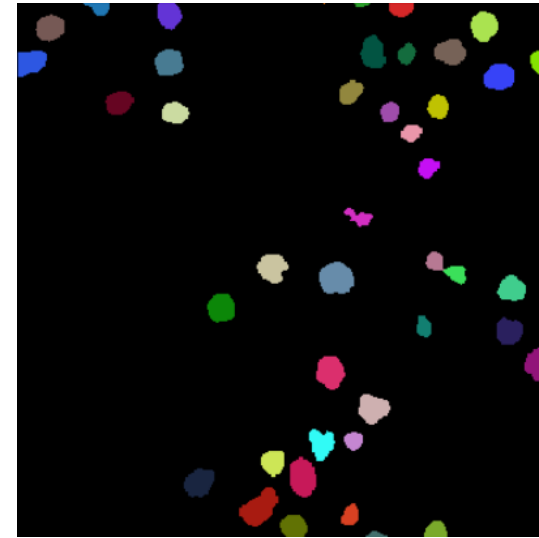


- Compare annotations with algorithm results

Sparse instance
annotation



Instance
segmentation



```
from the_segmentation_game import metrics
```

```
[10]: metrics.jaccard_index_sparse(annotation, labels)
```

```
←
```

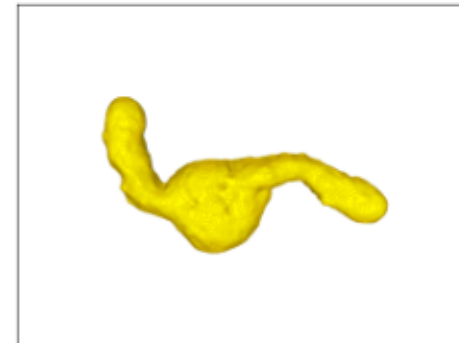
```
[10]: 0.8357392602053431
```

<https://github.com/haesleinhuepf/the-segmentation-game#segmentation-algorithm-comparison>

- Turn binary and/or label images into surface meshes

```
surface = nppas.all_labels_to_surface(binary_filled)
```

```
surface
```

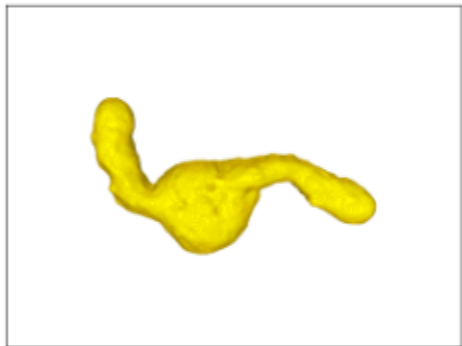


nppas.SurfaceTuple

origin (z/y/x)	[0. 0. 0.]
center of mass(z/y/x)	57.710,309.963,440.042
scale(z/y/x)	1.000,1.000,1.000
bounds (z/y/x)	12.500...113.500 111.500...461.500 169.500...807.500
average size	170.769
number of vertices	330776
number of faces	661548

- Surface mesh simplification
- To prevent the computer freezing

```
simplified_surface = nppas.decimate_quadric(surface, fraction=0.01)  
simplified_surface
```



nppas.SurfaceTuple

origin (z/y/x) [0. 0. 0.]

center of mass(z/y/x) 57.710,309.963,440.042

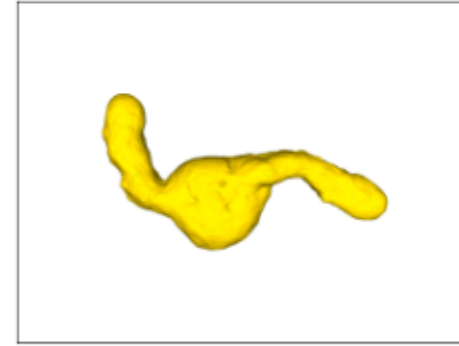
scale(z/y/x) 1.000,1.000,1.000

bounds (z/y/x) 12.500...113.500
111.500...461.500
169.500...807.500

average size 170.769

number of vertices 330776

number of faces 661548



nppas.SurfaceTuple

origin (z/y/x) [0. 0. 0.]

center of mass(z/y/x) 57.928,308.938,440.985

scale(z/y/x) 1.000,1.000,1.000

bounds (z/y/x) 13.231...113.510
111.642...461.602
170.022...806.468

average size 170.083

number of vertices 3310

number of faces 6615

- Surface mesh smoothing

```
smoothed_surface = nppas.smooth_surface(simplified_surface)
smoothed_surface
```

nppas.SurfaceTuple

origin (z/y/x) [0. 0. 0.]

center of mass(z/y/x) 57.928,308.938,440.985

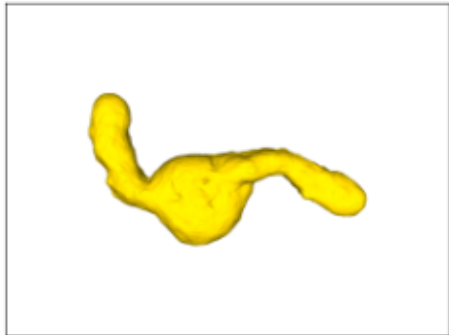
scale(z/y/x) 1.000,1.000,1.000

bounds (z/y/x) 13.231...113.510
111.642...461.602
170.022...806.468

average size 170.083

number of vertices 3310

number of faces 6615



nppas.SurfaceTuple

origin (z/y/x) [0. 0. 0.]

center of mass(z/y/x) 57.913,308.988,440.878

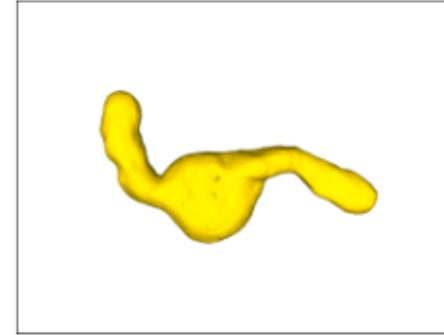
scale(z/y/x) 1.000,1.000,1.000

bounds (z/y/x) 13.901...113.627
110.982...461.191
169.711...807.193

average size 170.378

number of vertices 3310

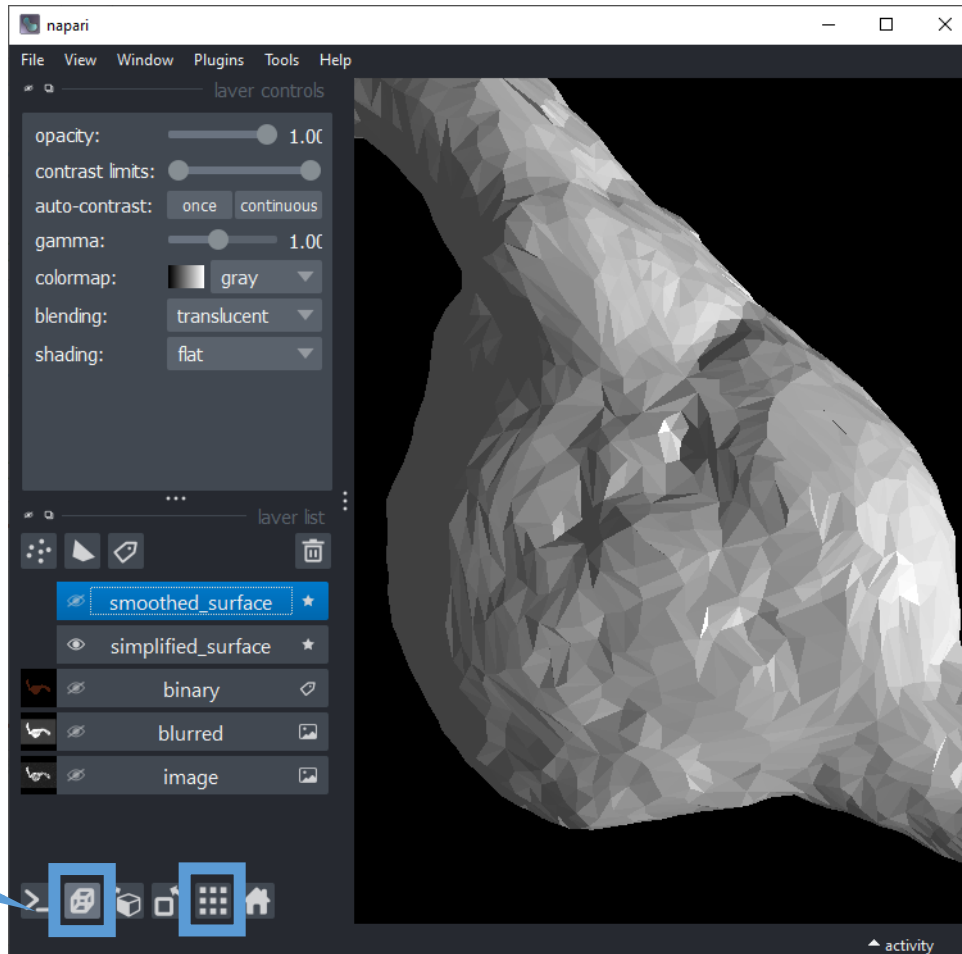
number of faces 6615



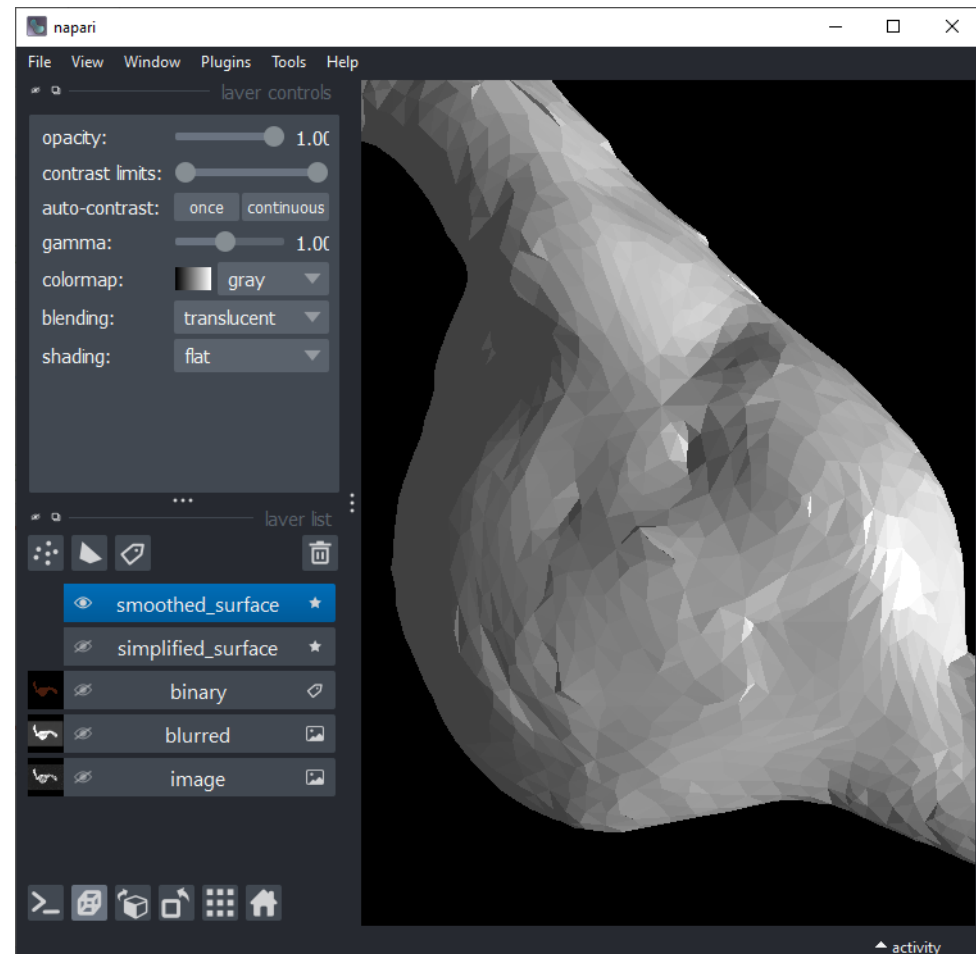
View surface meshes in Napari

```
viewer.add_surface(surface, scale=[zoom, zoom, zoom])  
viewer.add_surface(simplified_surface, scale=[zoom, zoom, zoom])  
viewer.add_surface(smoothed_surface, scale=[zoom, zoom, zoom])
```

In case your computer freezes, comment out this line



Toggle 3D view and grid mode





Exercises

Robert Haase



April 2023

Exercise

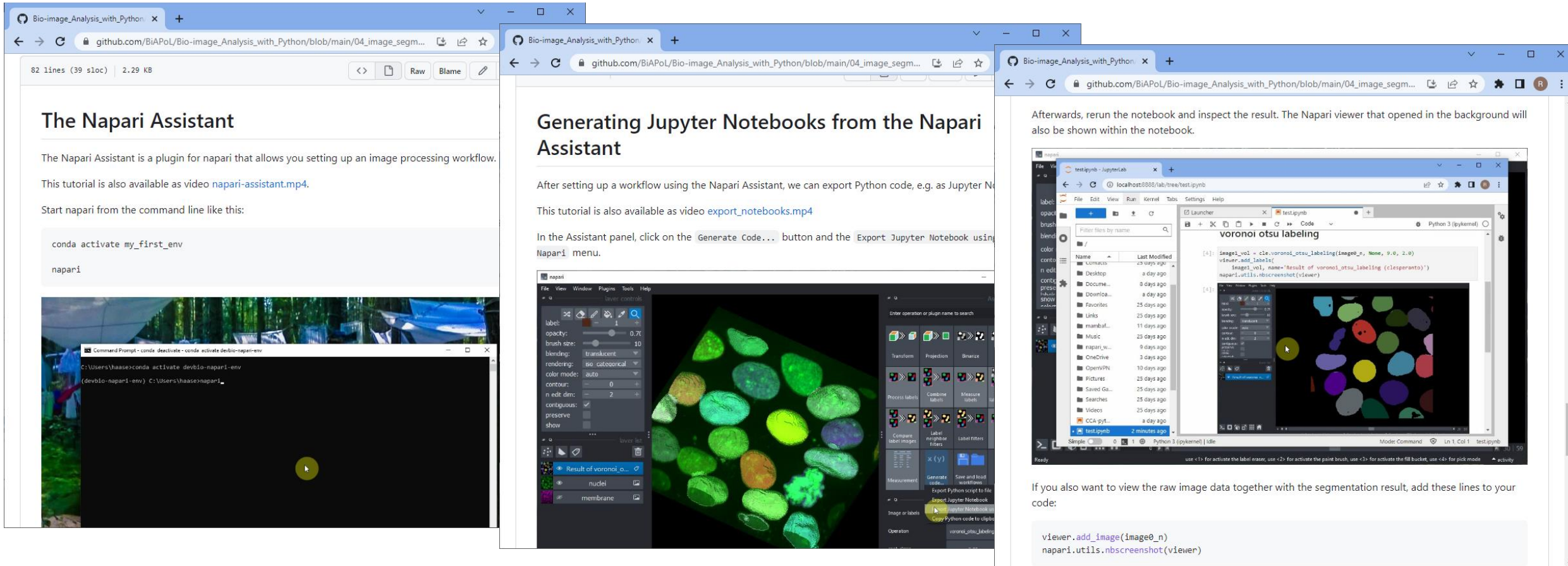
- Use the Napari Assistant to generate a Jupyter Notebook

The image displays a sequence of five screenshots illustrating the workflow for generating a Jupyter Notebook from a Napari session:

- Initial View:** A Napari window showing a microscopy image of cells. The left sidebar contains layer controls for 'nuclei' (green) and 'membrane' (purple).
- Segmentation:** The same image is shown with a Voronoi segmentation overlay. The layer list includes 'Result of voronoi_o...', 'nuclei', and 'membrane'.
- Assistant:** The Napari Assistant window is open, showing a search bar and a list of operations or plugins.
- Jupyter Notebook:** A JupyterLab interface is shown with a notebook titled 'voronoi otsu labeling'. The code in the notebook is:

```
image1_voi = cle.voronoi_otsu_labeling(image0_n, None, 9.0, 2.0)
viewer.add_labels(
    image1_voi, name='Result of voronoi_otsu_labeling (clesperanto)')
napari.utils.nbscreenshot(viewer)
```
- Final View:** The final result of the labeling process, showing the segmented image with different colors for each cell.

- Follow the online instructions
- https://github.com/BiAPoL/Bio-image_Analysis_with_Python/blob/main/04_image_segmentation/06_napari-assistant.md
- https://github.com/BiAPoL/Bio-image_Analysis_with_Python/blob/main/04_image_segmentation/07_notebook_export.md



The Napari Assistant

The Napari Assistant is a plugin for napari that allows you setting up an image processing workflow.

This tutorial is also available as video [napari-assistant.mp4](#).

Start napari from the command line like this:

```
conda activate my_first_env
napari
```

Generating Jupyter Notebooks from the Napari Assistant

After setting up a workflow using the Napari Assistant, we can export Python code, e.g. as Jupyter Notebook.

This tutorial is also available as video [export_notebooks.mp4](#)

In the Assistant panel, click on the `Generate Code...` button and the `Export Jupyter Notebook` using the `Napari` menu.

Afterwards, rerun the notebook and inspect the result. The Napari viewer that opened in the background will also be shown within the notebook.

```
image_vol = cle.voronoi_otsu_labeling(image0_n, None, 9.0, 2.0)
viewer.add_labels(
    image_vol, name='Result of voronoi_otsu_labeling (clesperanto)'
)
napari.utils.nbscreenshot(viewer)
```

If you also want to view the raw image data together with the segmentation result, add these lines to your code:

```
viewer.add_image(image0_n)
napari.utils.nbscreenshot(viewer)
```


- Measure the quality of a segmentation algorithm applied to a folder of images.
- <https://github.com/BiAPoL/Bio-image Analysis with Python/blob/main/04 image segmentation/17 segmentation quality estimation.ipynb>

```
metrics.jaccard_index_sparse(sparse_labels, labels)
```

```
0.8357392602053431
```

```
for image_filename in os.listdir(image_folder):  
    print(image_folder + image_filename)
```

```
../../../../data/BBBC007_batch/17P1_POS0013_D_1UL.tif  
../../../../data/BBBC007_batch/20P1_POS0005_D_1UL.tif  
../../../../data/BBBC007_batch/20P1_POS0007_D_1UL.tif  
../../../../data/BBBC007_batch/20P1_POS0010_D_1UL.tif  
../../../../data/BBBC007_batch/A9_p7d.tif  
../../../../data/BBBC007_batch/AS_09125_040701150004_A02f00d0.tif
```