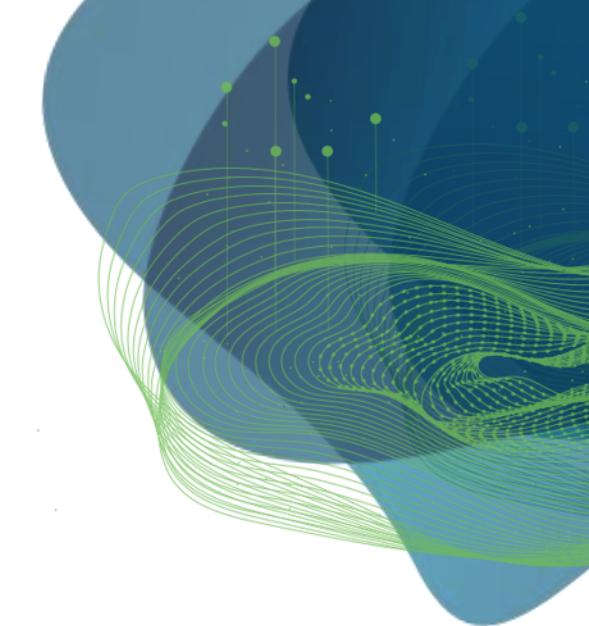




DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS  
AND ARTIFICIAL INTELLIGENCE



# Image segmentation

Robert Haase

Using materials from Ryan Savill George (PoL, TU Dresden)

GEFÖRDERT VOM



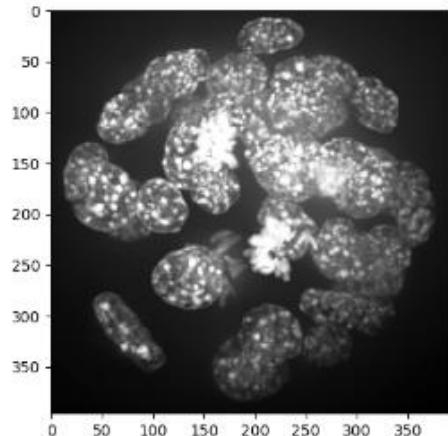
Bundesministerium  
für Bildung  
und Forschung

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.

# Quiz (recap)

- How is this operation called?

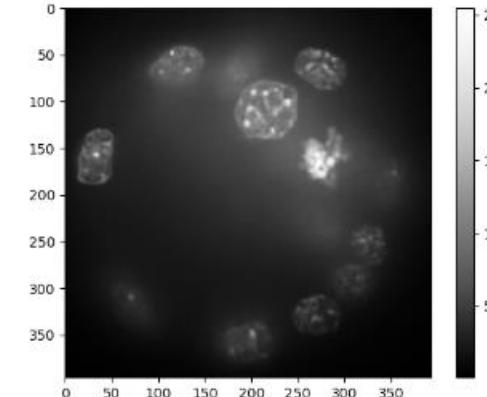
```
stackview.insight(image)
```



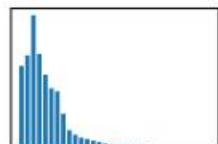
shape (86, 396, 393)  
dtype uint8  
size 12.8 MB  
min 0  
max 255

image [40]

```
new_image = image[40]  
stackview.insight(new_image)
```



shape (396, 393)  
dtype uint8  
size 152.0 kB  
min 1  
max 255



Slicing



Indexing



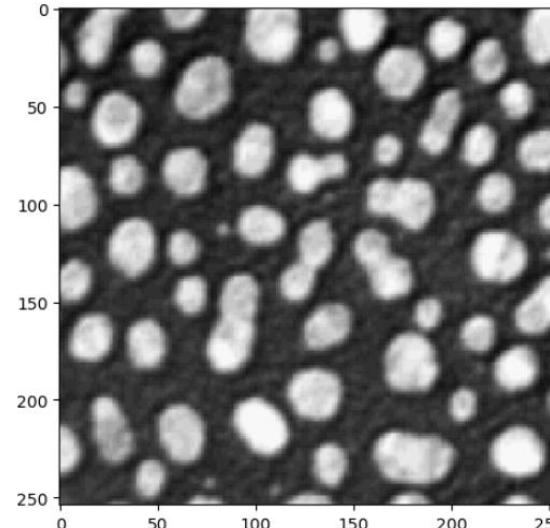
Cropping



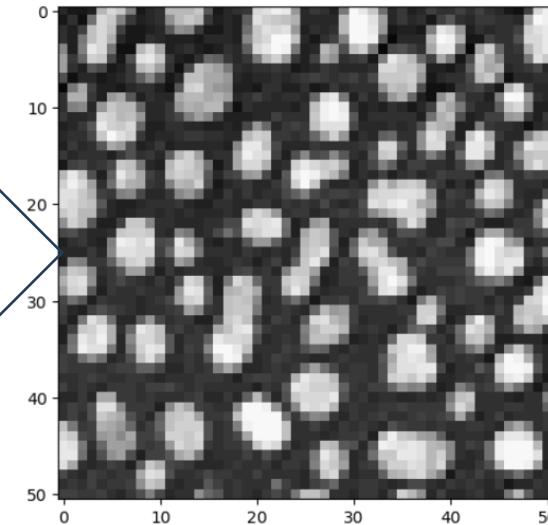
Sampling

# Quiz (recap)

- How is this operation called?



`image[::5, ::5]`



Slicing



Indexing



Cropping



Sampling

# Ask ChatGPT



Du

What does the code `image[::-5, ::5]` do?



ChatGPT

The code `image[::-5, ::5]` is used in Python, typically with libraries like NumPy or similar, to perform an operation known as "striding" on a 2D array, which is often used to represent an image. This particular snippet of code will downsample the image by a factor of 5 in both dimensions.

Here's a breakdown of what each part of the code does:

- `image`: This represents the 2D array (or image) being operated on.
- `[::-5, ::5]`: This is a slicing operation using the `:` operator, which is used to access elements of the array.
  - `::5` in each dimension tells Python to take every 5th element from the respective dimension. The first `::5` applies to rows, and the second `::5` applies to columns.

So, if the original image had a size of  $100 \times 100$  pixels, using `image[::-5, ::5]` would create a new image of size  $20 \times 20$  pixels, where each pixel in the new image corresponds to every fifth pixel of the original image in both horizontal and vertical directions. This method effectively reduces the resolution of the image, which can be useful for image processing tasks where lower resolution might be sufficient or to reduce the computational load.



# Ask ChatGPT



Du

What does the code `image[::5,::5]` do?



Du

Explain me like I'm 12 years old and keep it short

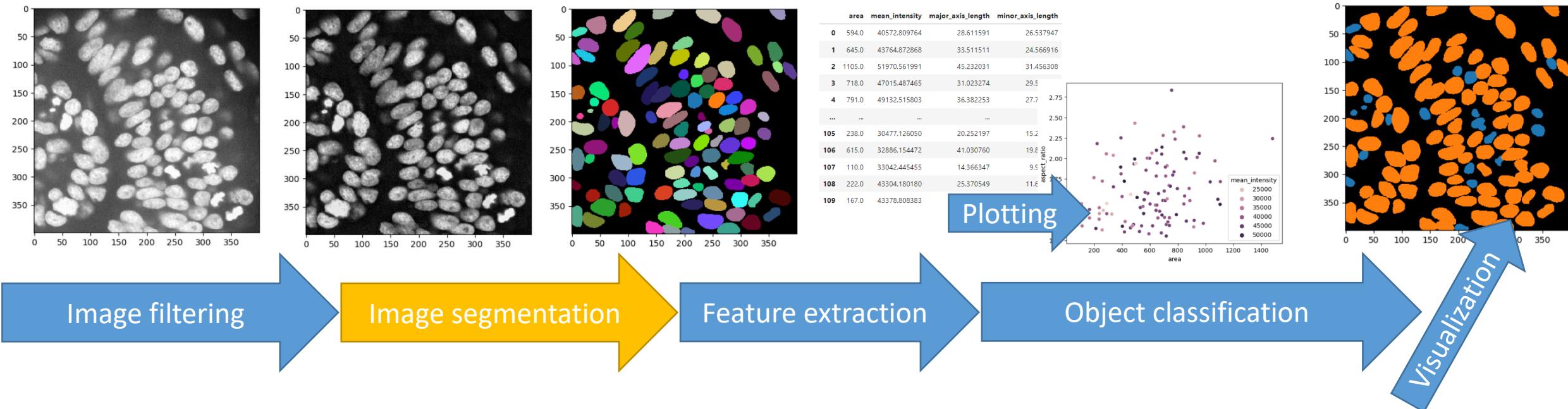


ChatGPT

Imagine you have a big grid of colored squares, which is a picture. If you take every fifth square from each row and each column, and put those squares together to make a smaller grid, that's what the code ``image[::5, ::5]`` does. It makes the big picture smaller by keeping only some of the squares and skipping a lot in between. This makes the picture smaller and simpler!

# Training overview: Bio-image Analysis

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

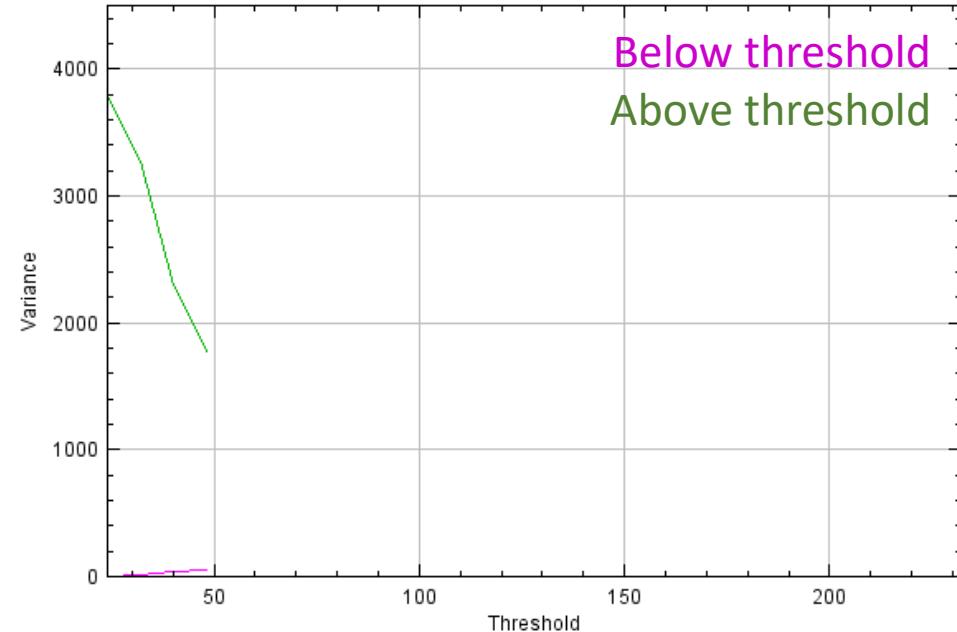
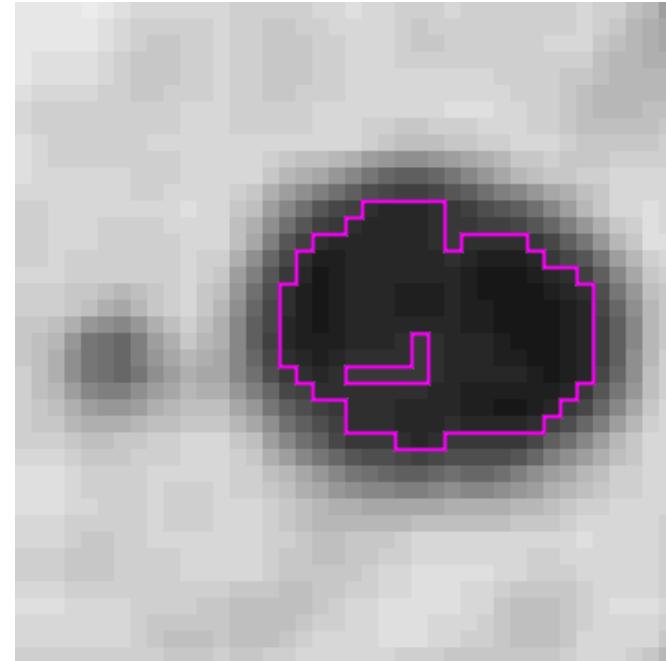


# Thresholding: Otsu's method

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

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

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



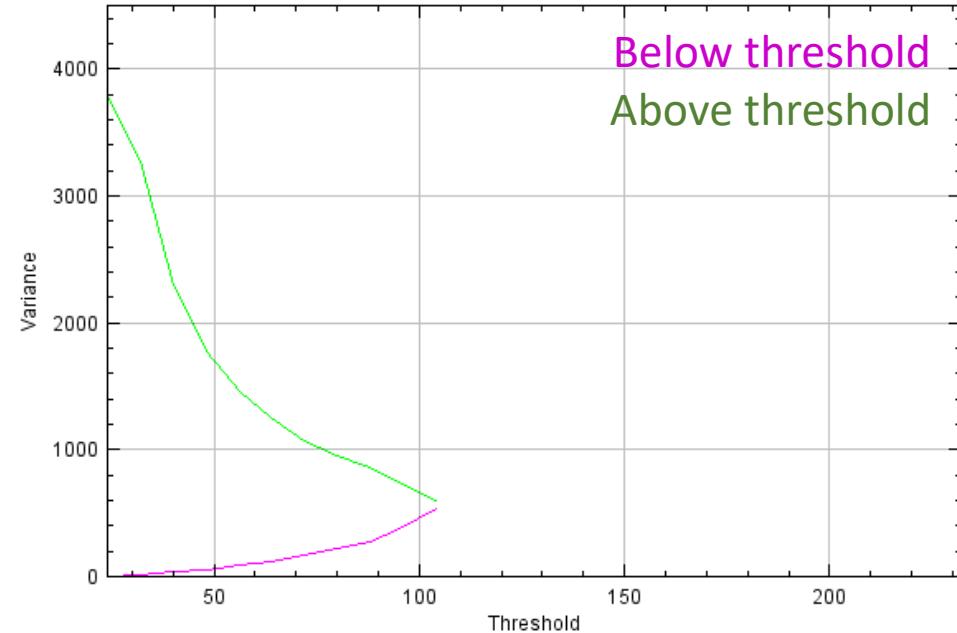
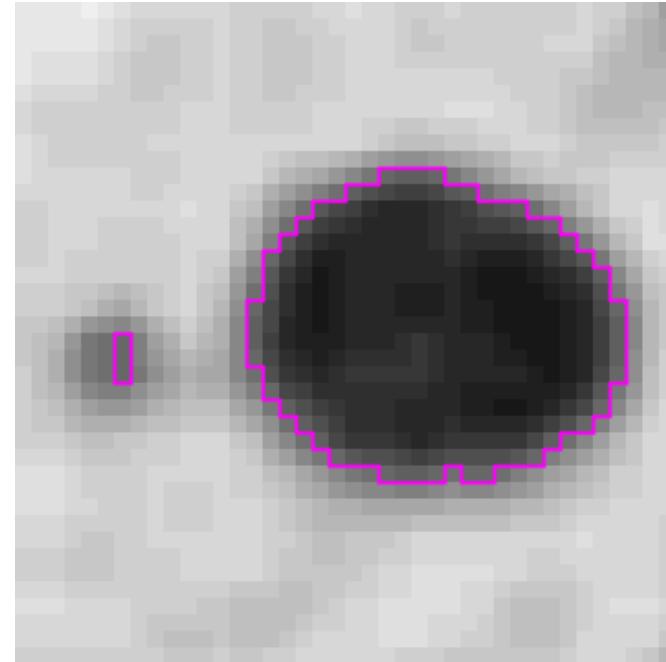
# Thresholding: Otsu's method

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

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

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

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



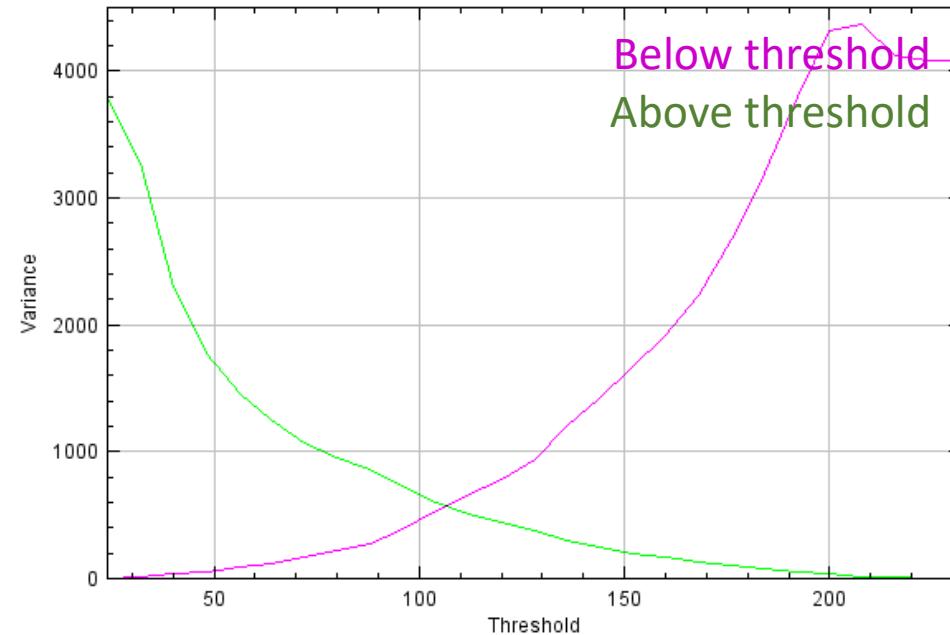
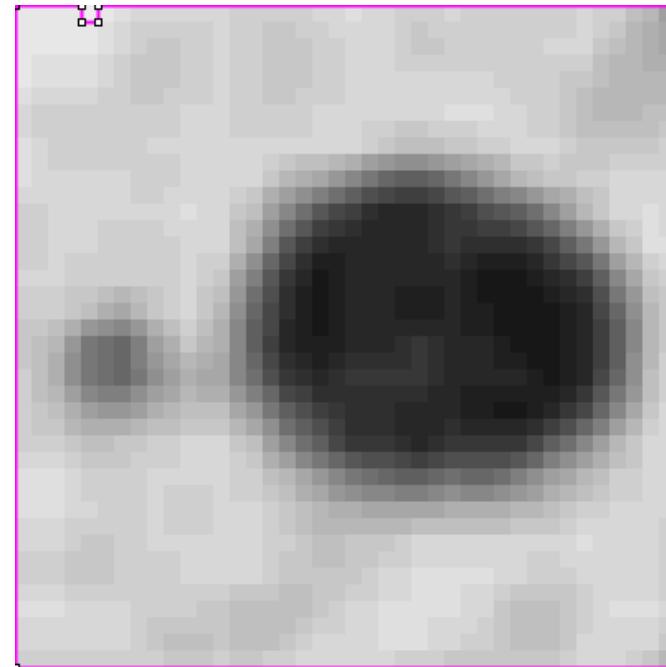
# Thresholding: Otsu's method

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

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

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

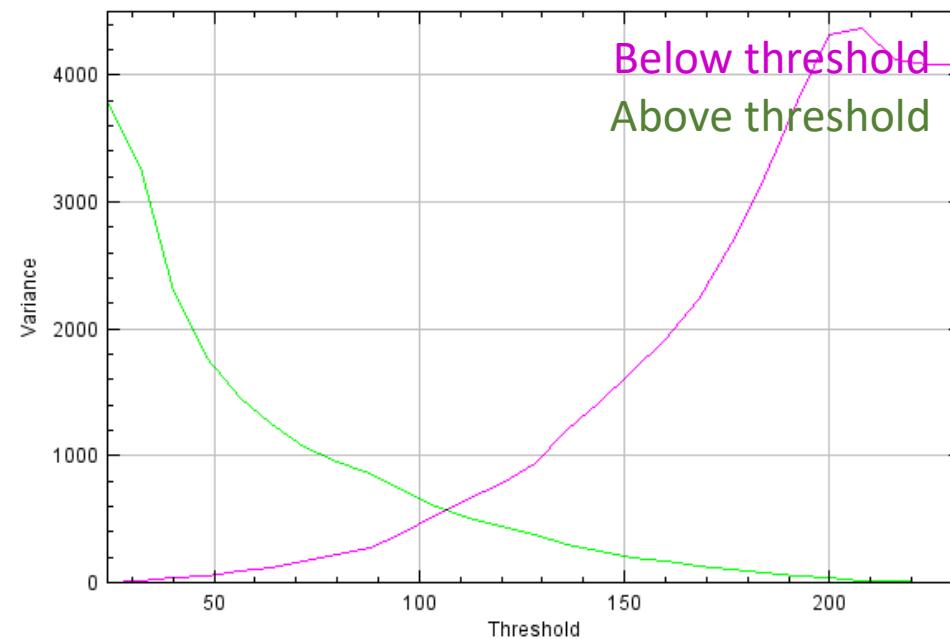
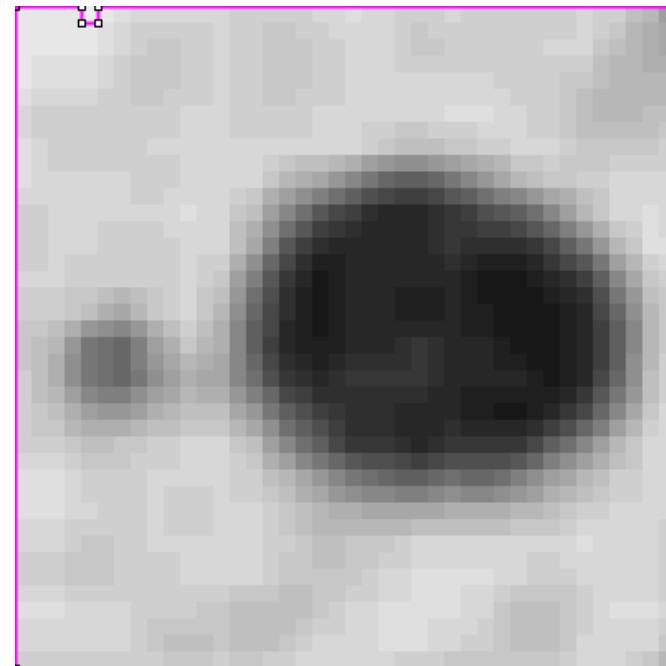
$Var(I)$  ... Variance in image I  
 $g_i$  ... grey value of a pixel i  
 $\bar{g}_I$  ... mean grey value of the whole image I  
 $n_I$  ... number of pixels in Image I



# Thresholding: Otsu's method

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

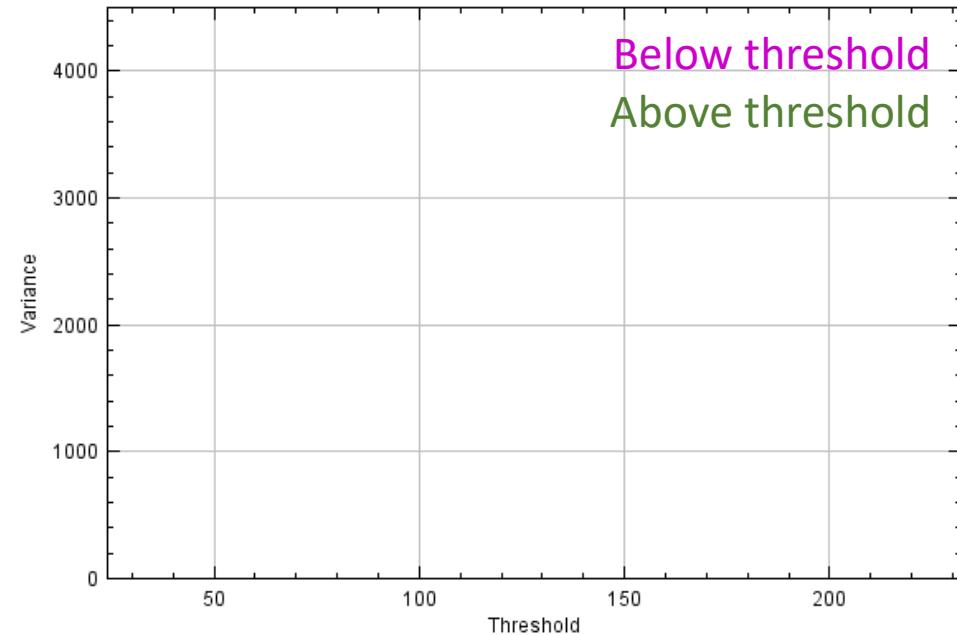
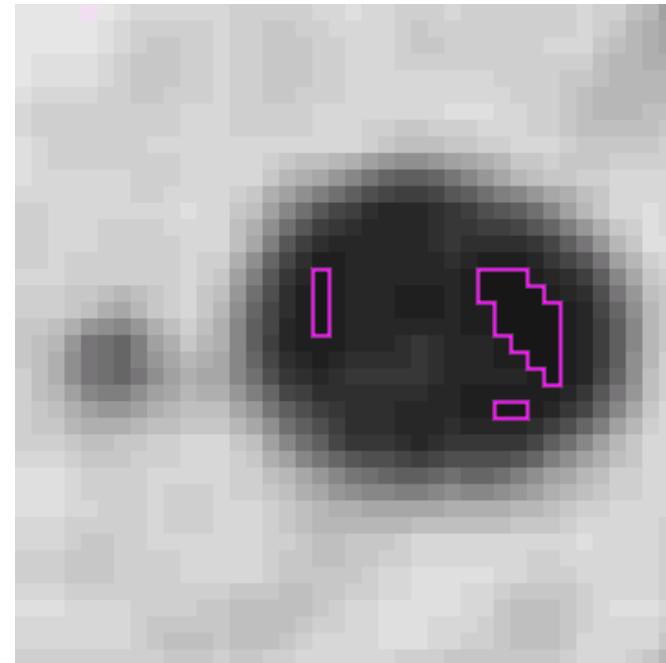
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



# Thresholding: Otsu's method

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

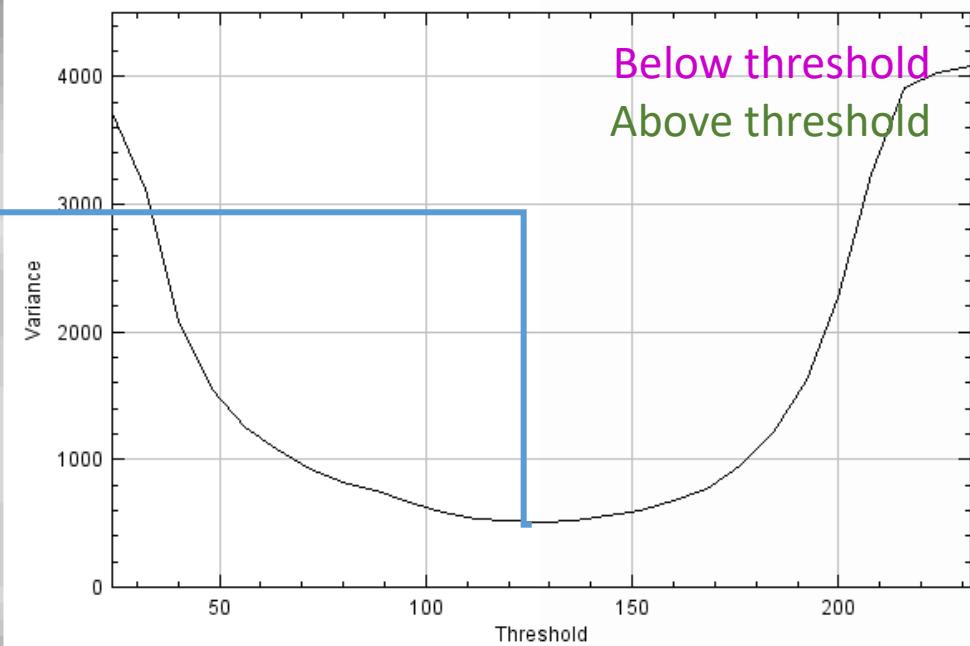
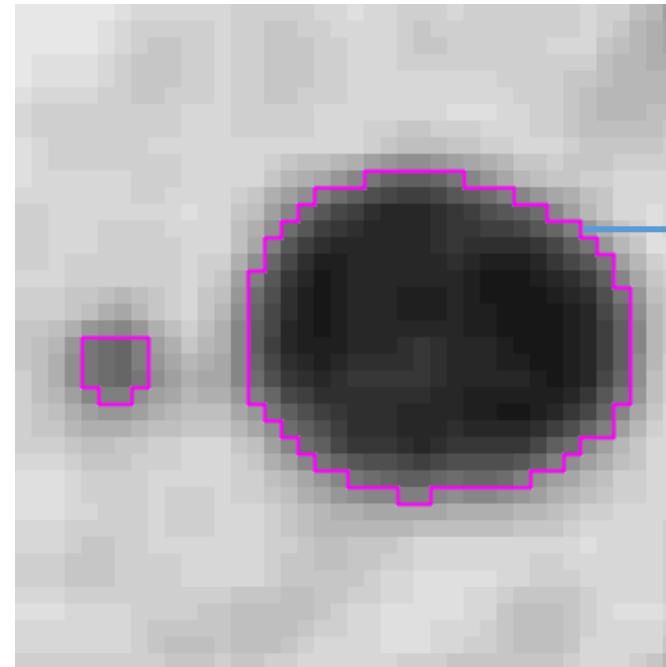
$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \quad I = A \cup B$$



# Thresholding: Otsu's method

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

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



# Thresholding: Citing

- Cite the thresholding method of your choice properly

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

IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. SMC-9, NO. 1, JANUARY 1979

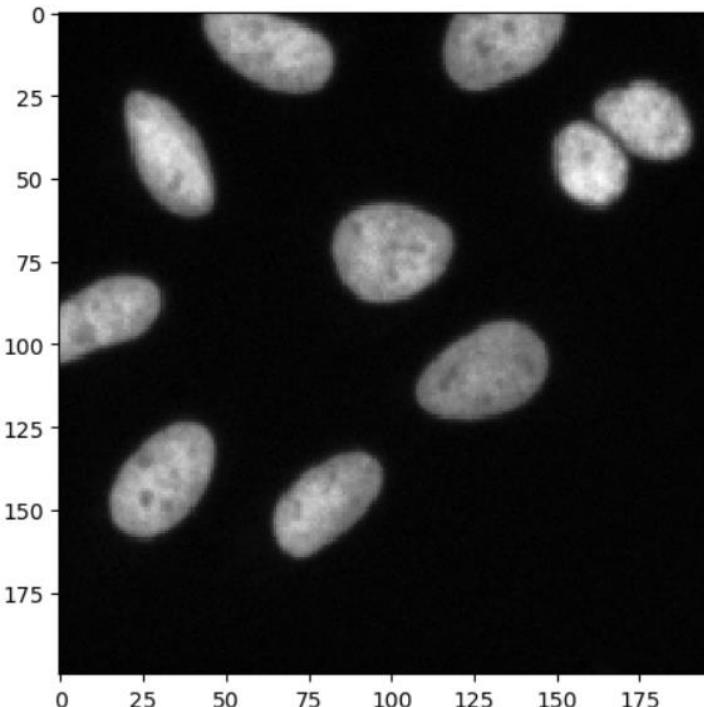
## A Threshold Selection Method from Gray-Level Histograms

NOBUYUKI OTSU

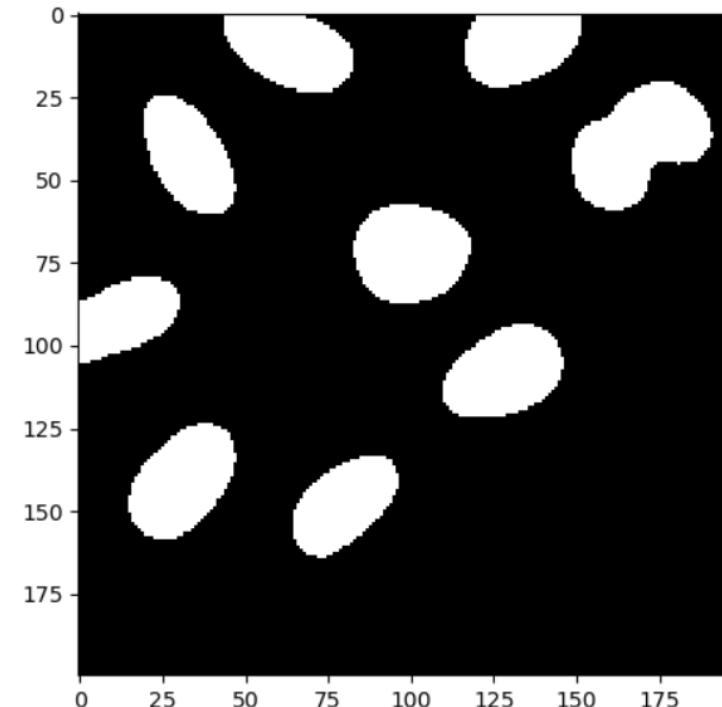
*Abstract*—A nonparametric and unsupervised method of automatic threshold selection for picture segmentation is presented. An optimal threshold is selected by the discriminant criterion, namely, so as to maximize the separability of the resultant classes in gray levels. The procedure is very simple, utilizing only the zeroth- and the first-order cumulative moments of the gray-level histogram. It is straightforward to extend the method to multithreshold problems. Several experimental results are also presented to support the validity of the method.

# Terminology

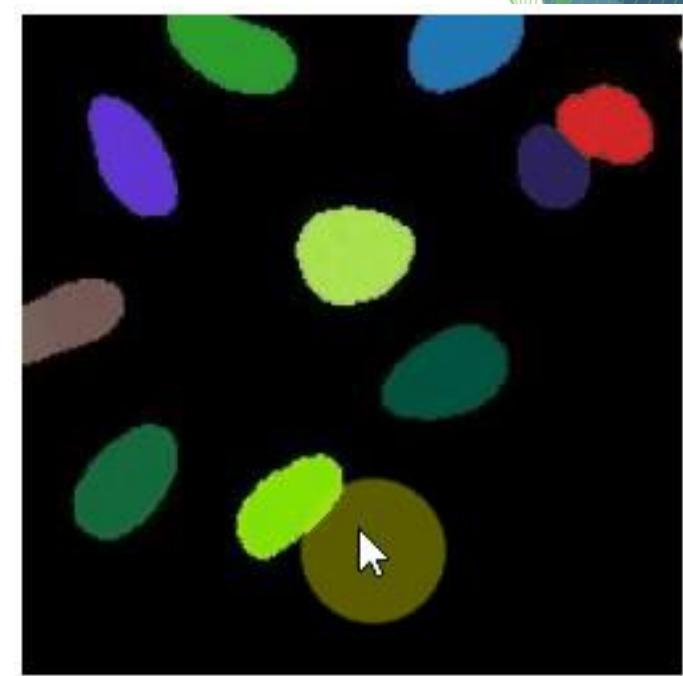
Intensity image



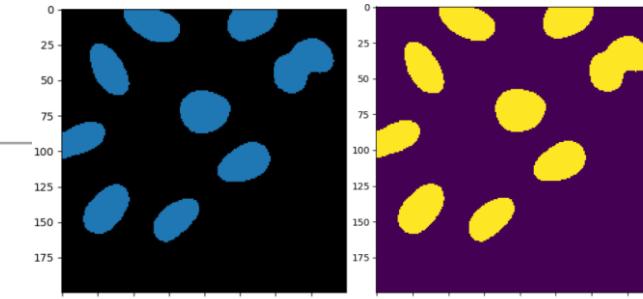
Binary image



Label image

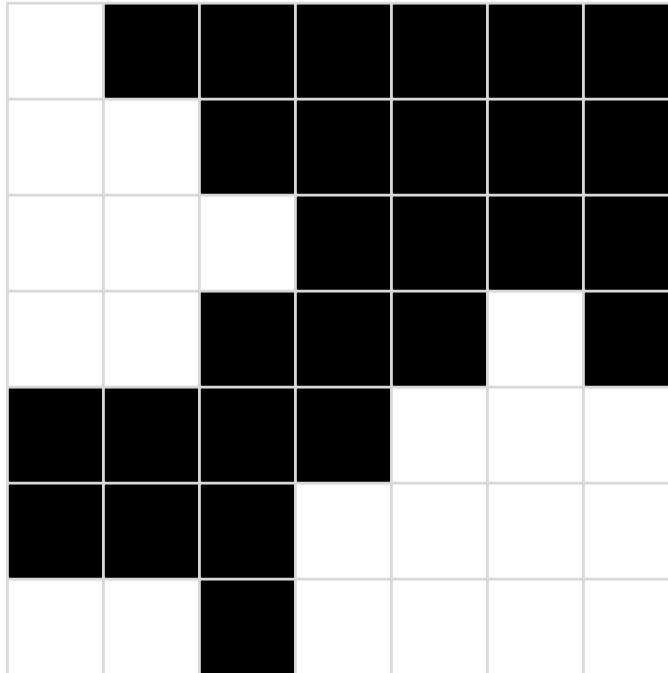


No matter how they are displayed



# Connected component labelling

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



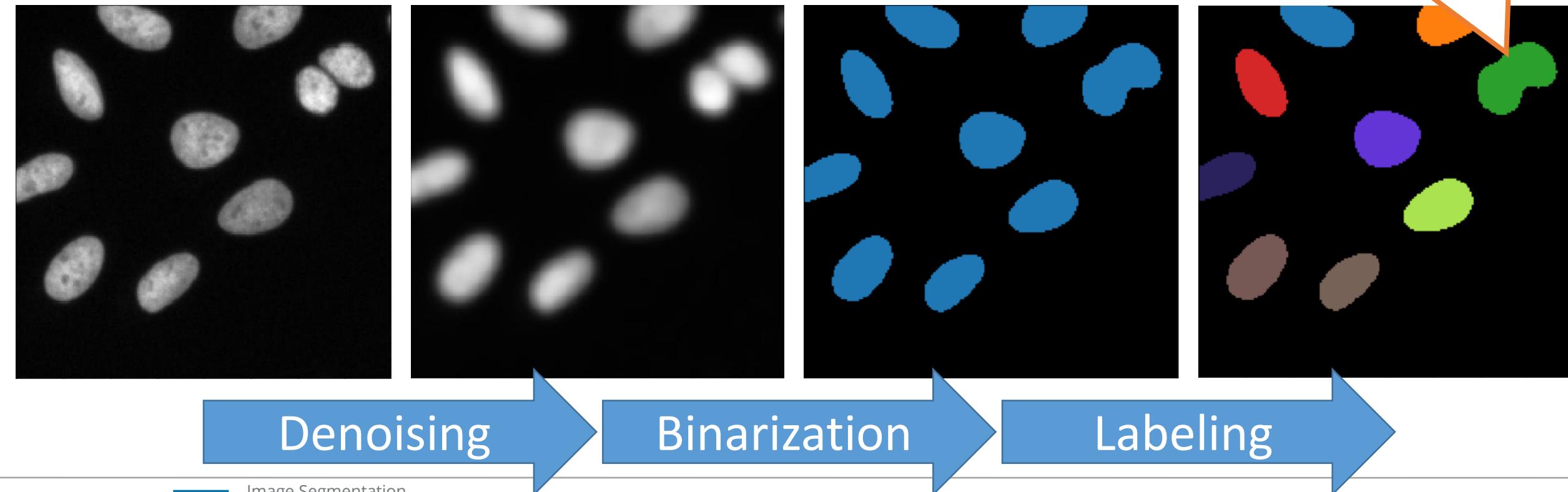
CCA

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

# Common image segmentation workflows

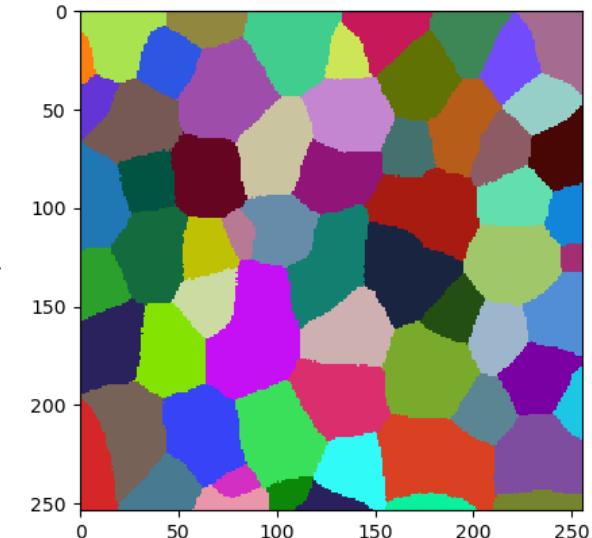
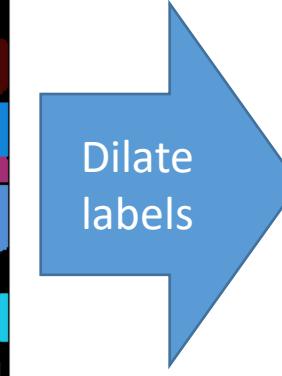
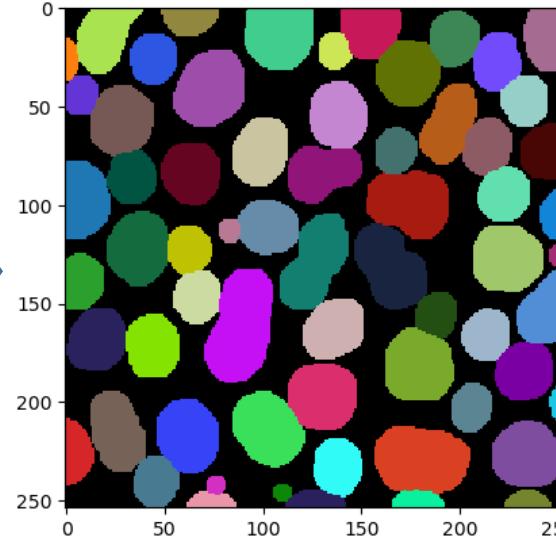
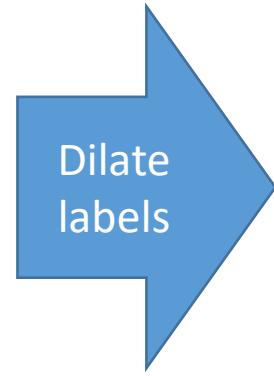
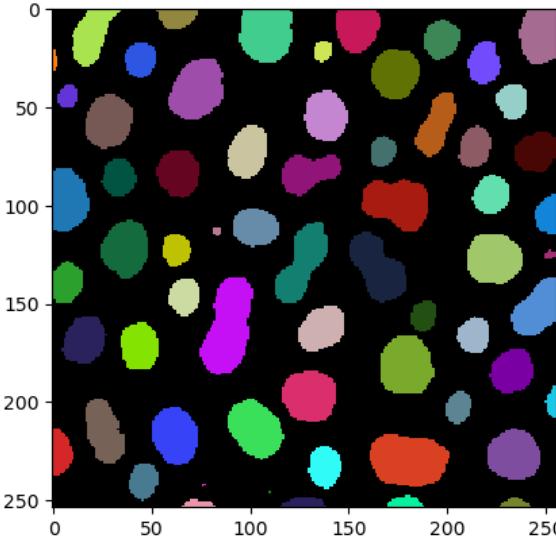
- Presumably the most common segmentation algorithm used for fluorescent microscopy images:
  - Gaussian blur, Otsu's Threshold, Connected Component Labeling

Limitation: Dense objects



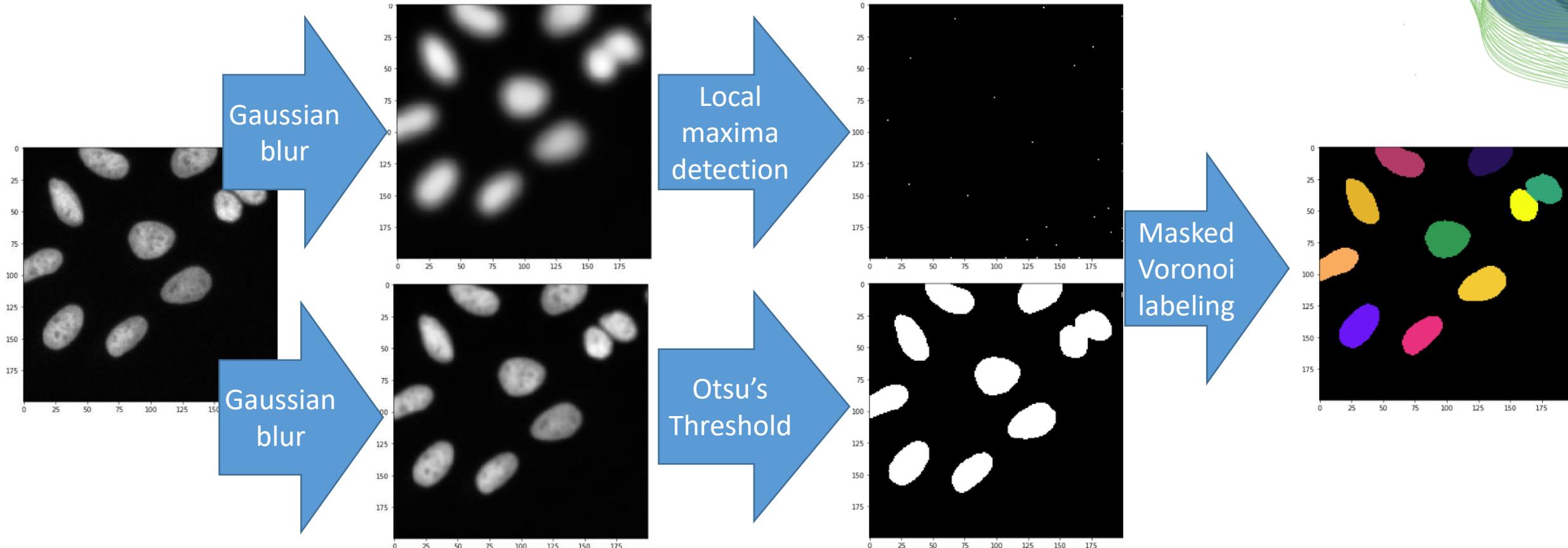
# Voronoi-Tesselation

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



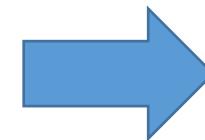
# Common image segmentation workflows

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



# Voronoi-Otsu-Labeling

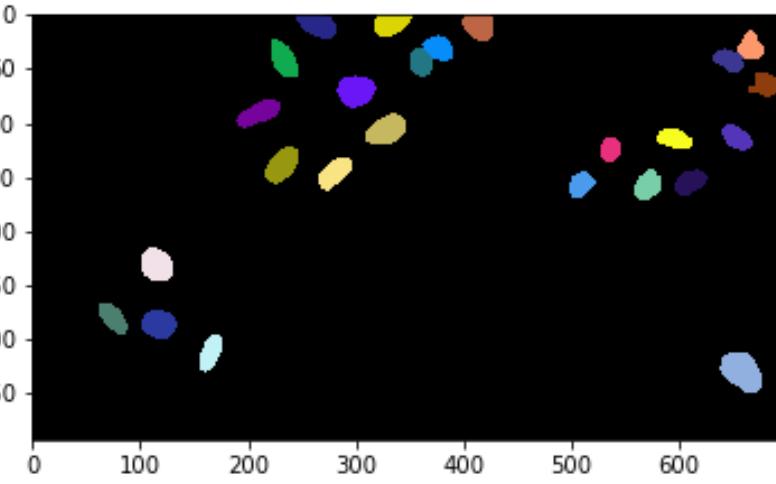
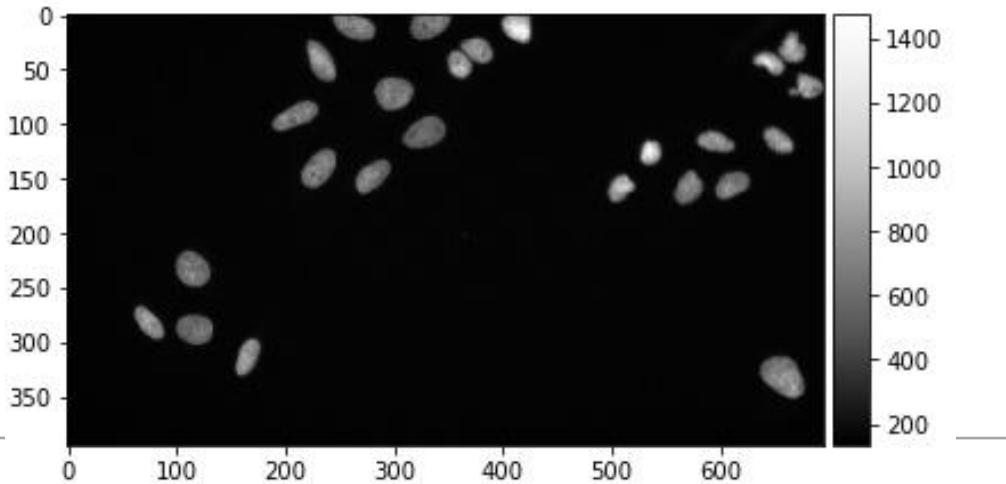
- Gaussian-Blur
- Otsu-Thresholding
- Spot-detection
- Watershed on the binary image



... in a single line of code:

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

segmented

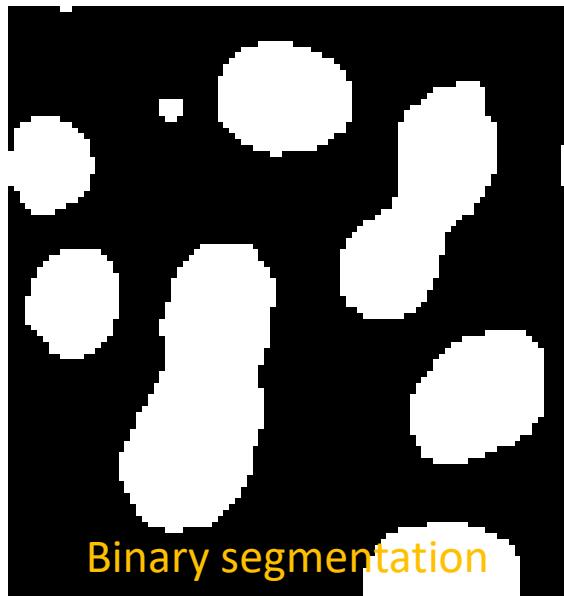


**nsbatwm made image**

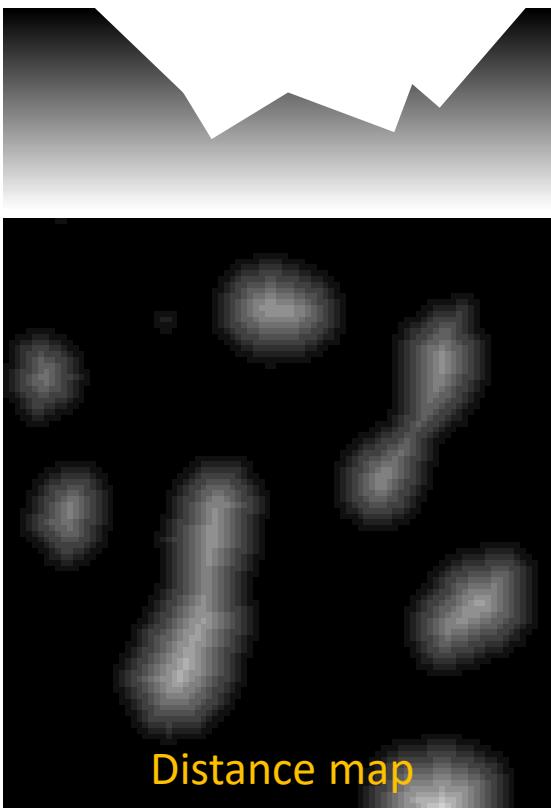
shape	(395, 695)
dtype	int32
size	1.0 MB
min	0
max	25

# Watershed

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



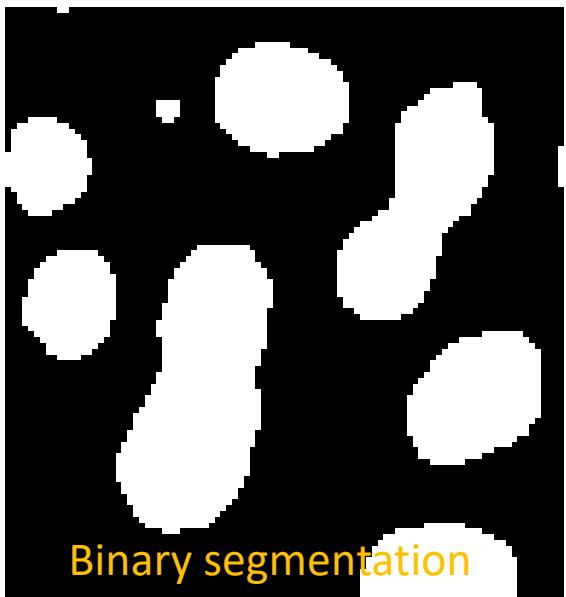
Binary segmentation



Distance map

# Watershed

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



Binary segmentation

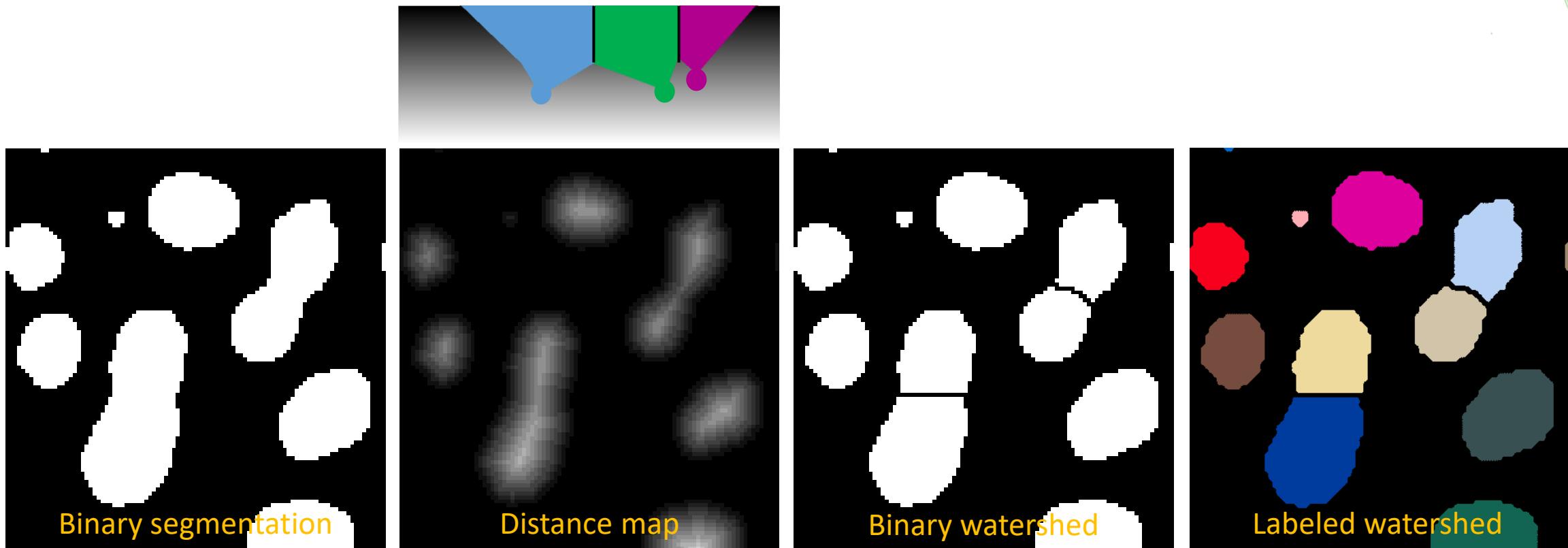


Distance map



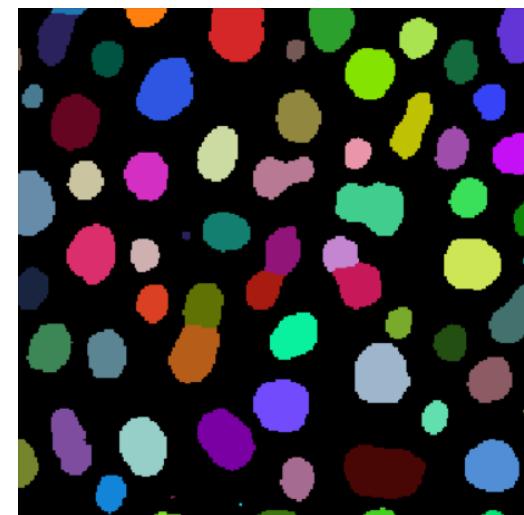
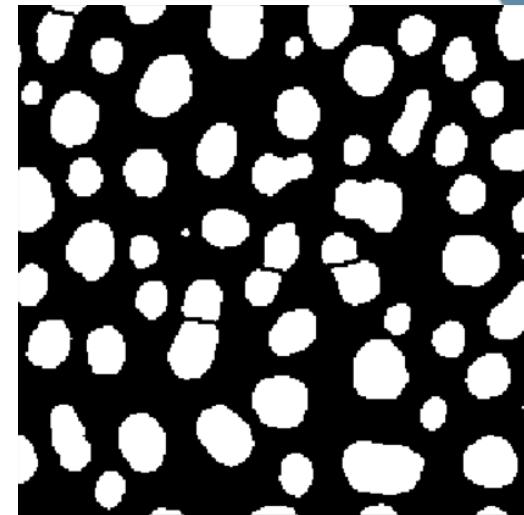
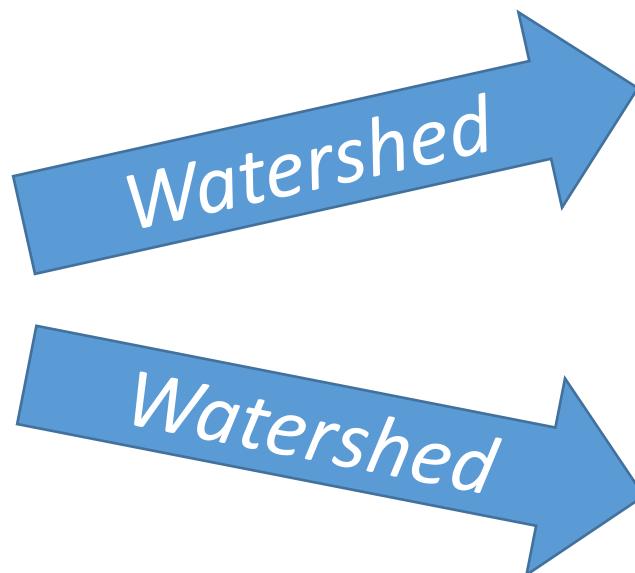
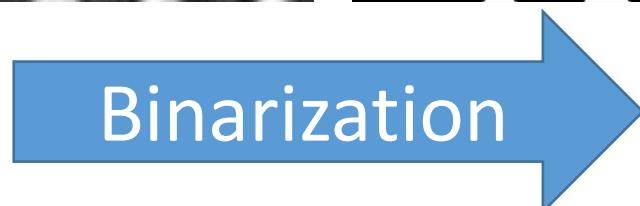
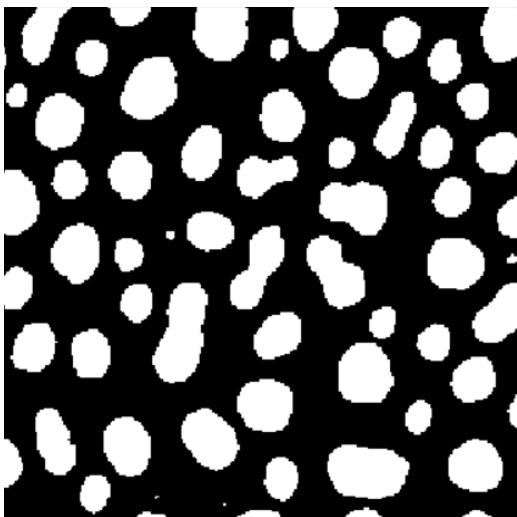
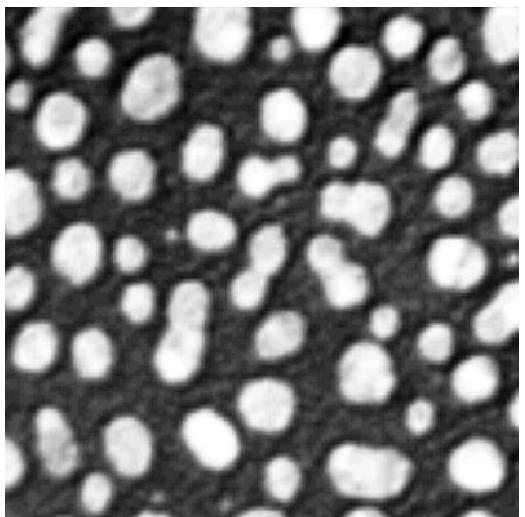
# Watershed

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



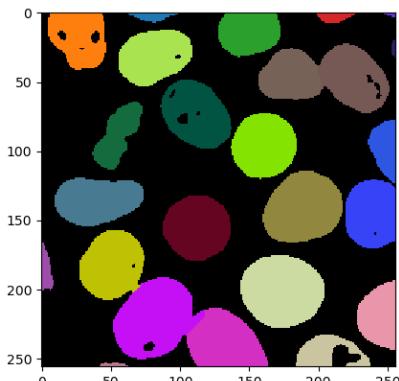
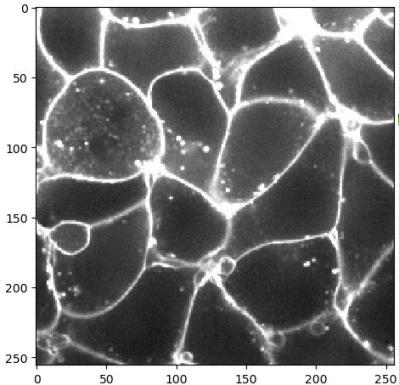
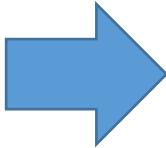
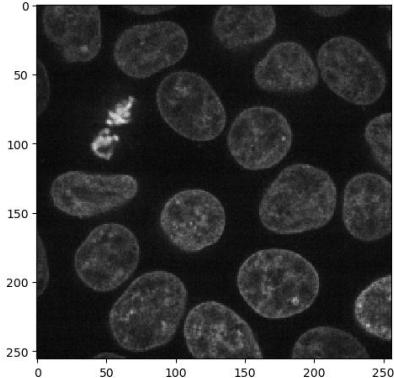
# Watershed use-cases

- Split dense objects



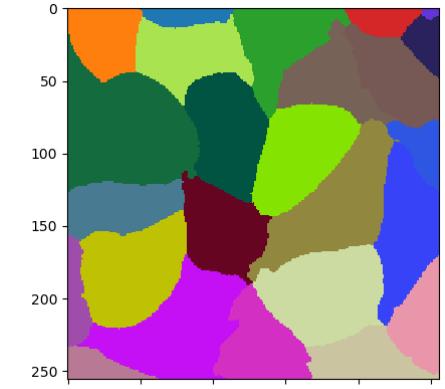
# Watershed use-cases

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



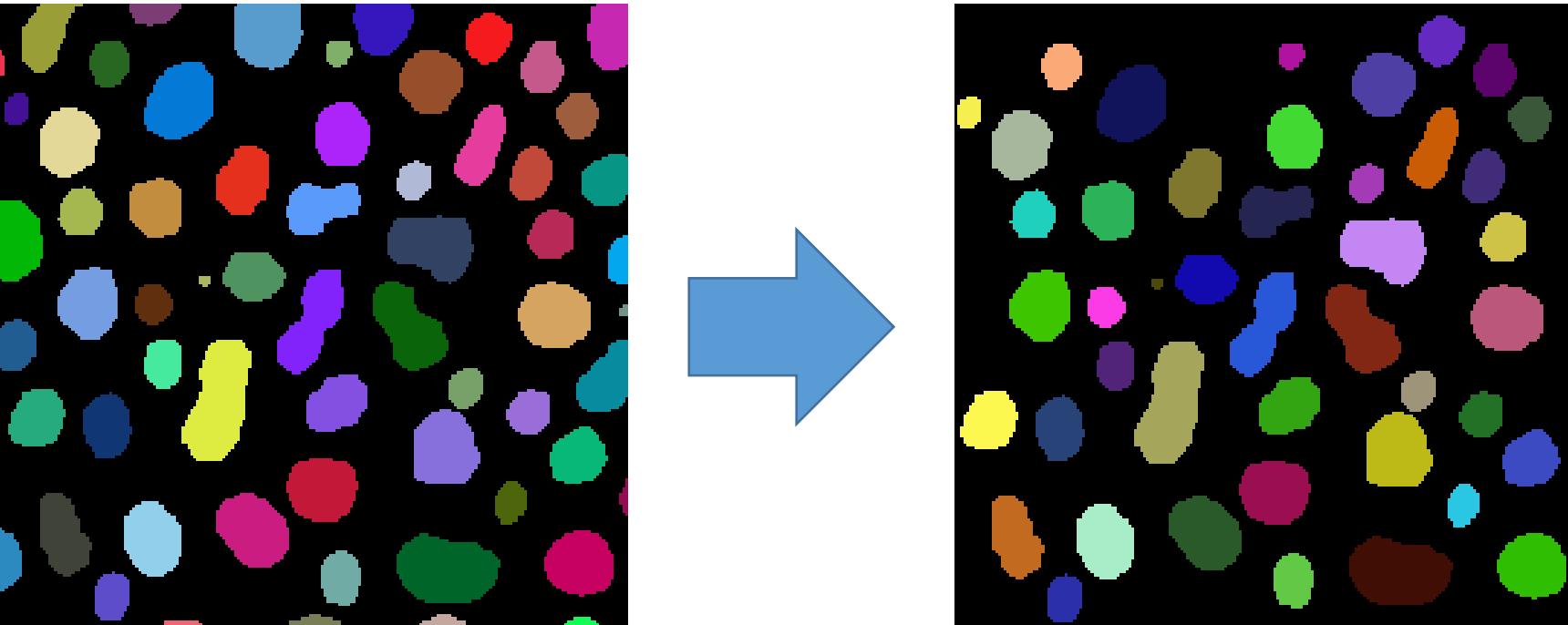
```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)
```

“Seeded” watershed



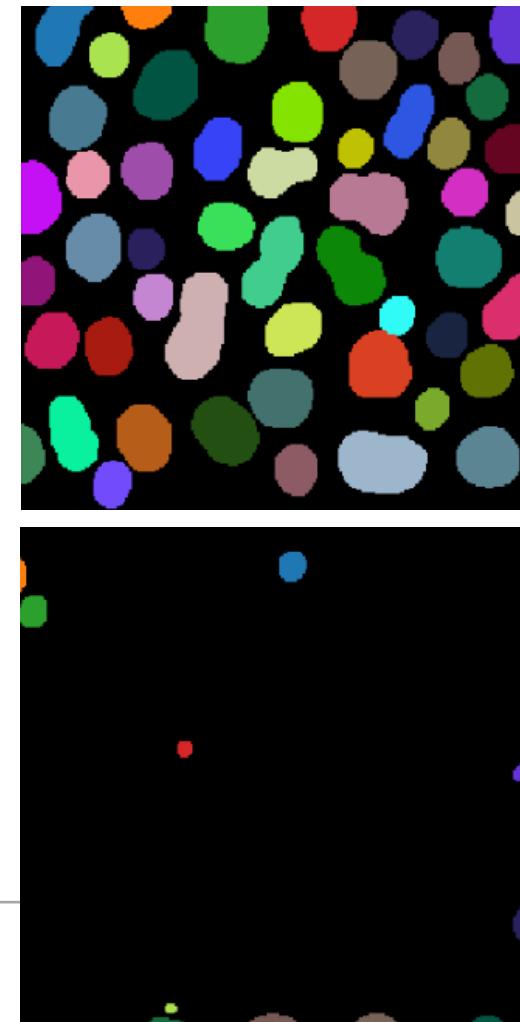
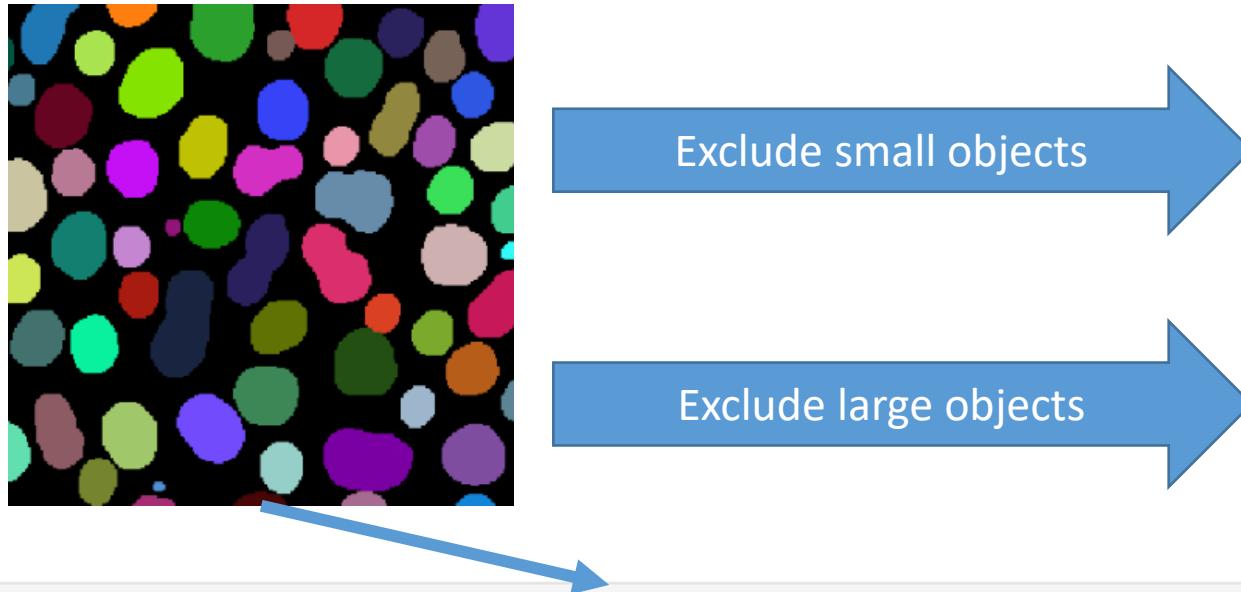
# Label post-processing / selections

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



# Label post-processing / selections

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





CENTER FOR SCALABLE DATA ANALYTICS AND  
ARTIFICIAL INTELLIGENCE

# Napari

## Robert Haase

Funded by



Bundesministerium  
für Bildung  
und Forschung

SACHSEN



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

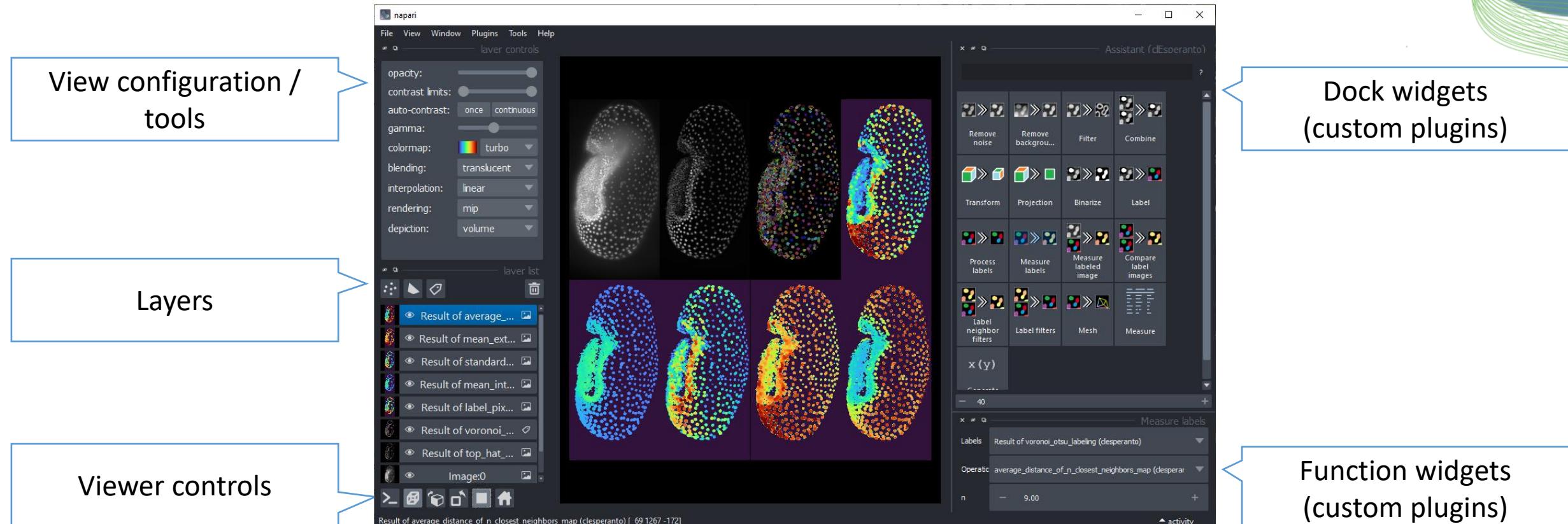
Chan  
Zuckerberg  
Initiative

# Napari

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

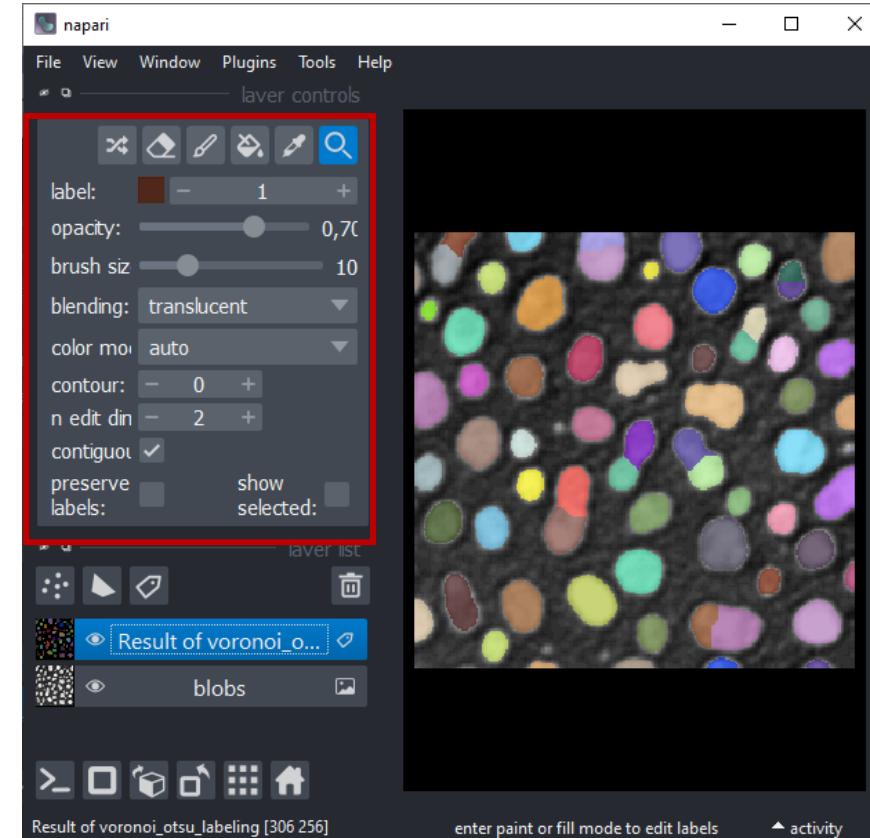
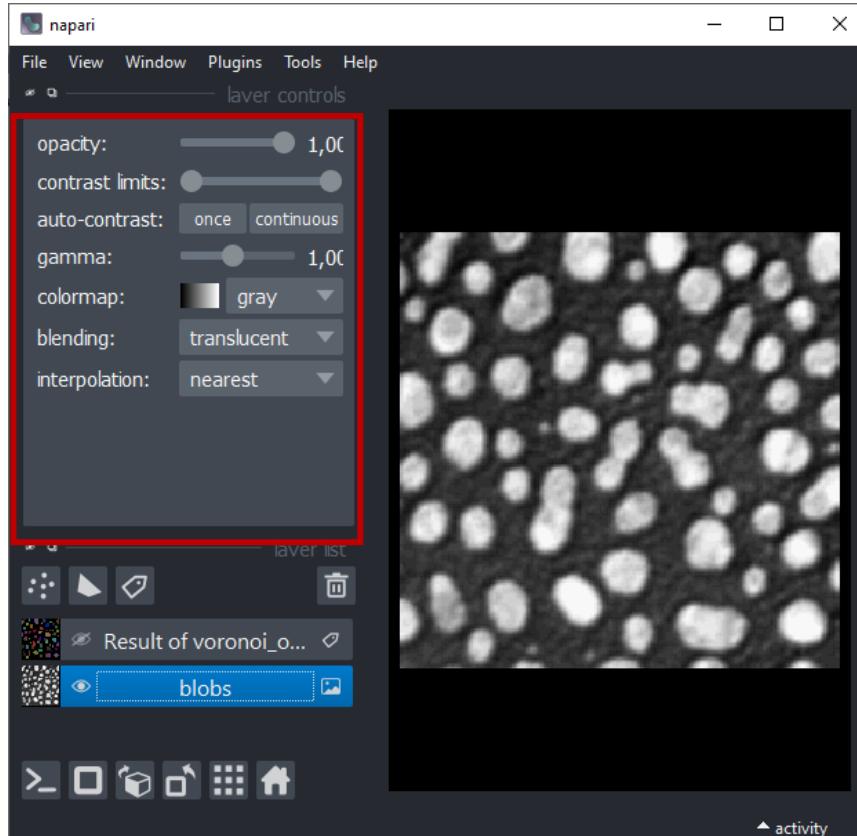


# Napari – Graphical User Interface



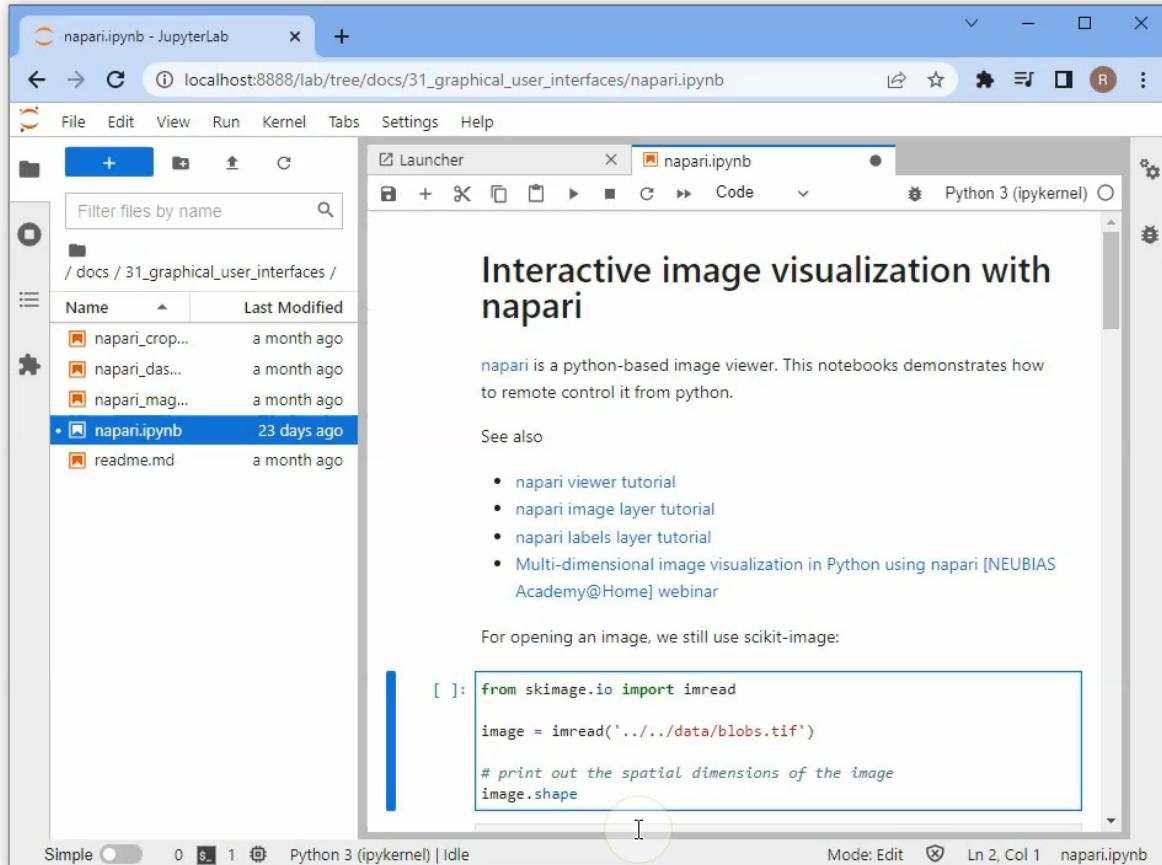
# Napari – Graphical User Interface

- Context / data type dependent tools



# Napari – Python Scripting

- Mixing interactivity and reproducibility



# Napari – Python Scripting

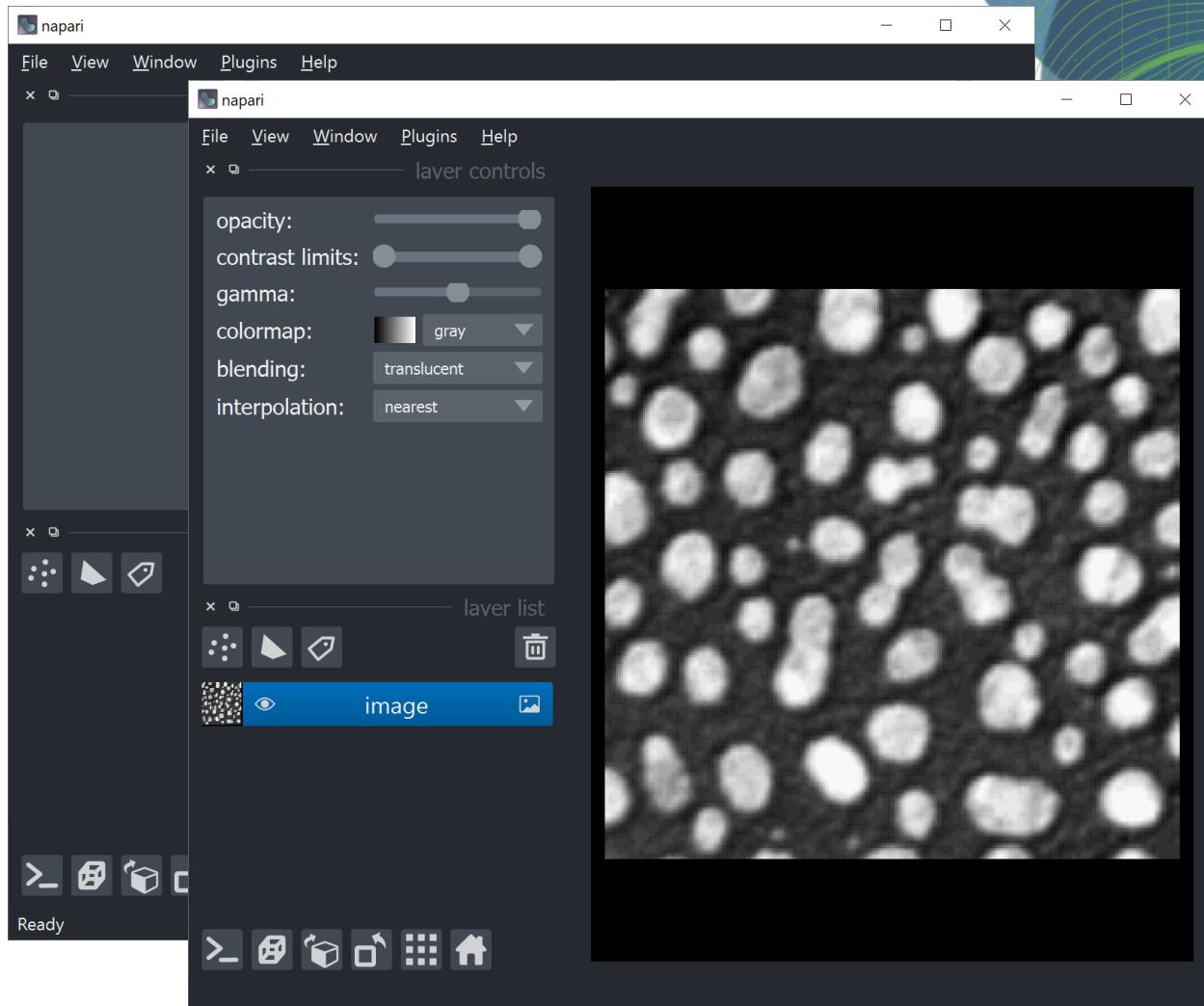
- Initialization

```
import napari
```

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

- Adding images

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



# Napari – Python Scripting

- Removing layers

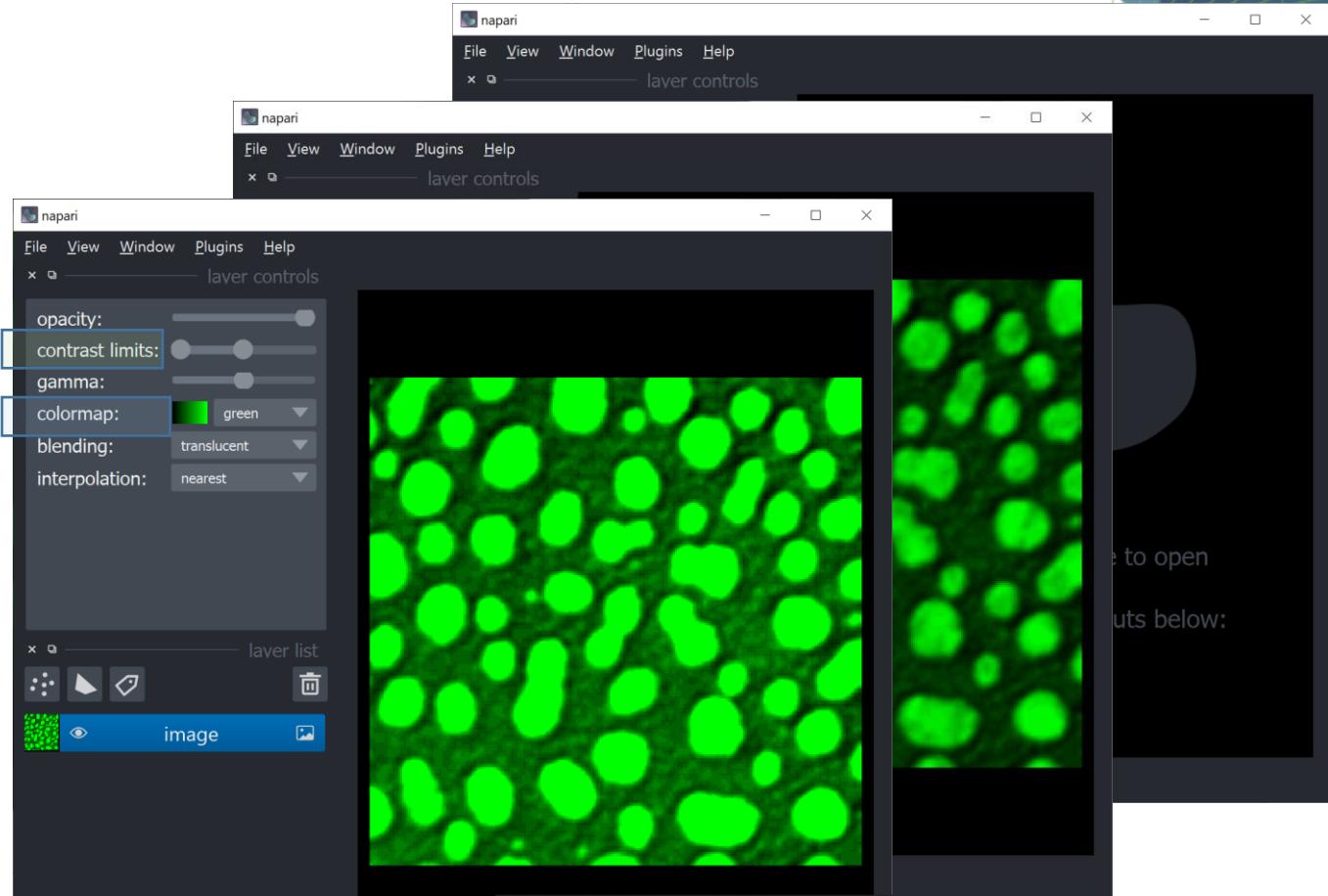
```
for l in viewer.layers:  
    viewer.layers.remove(l)
```

- Modify visualization while adding layers

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

- Modify layers after adding

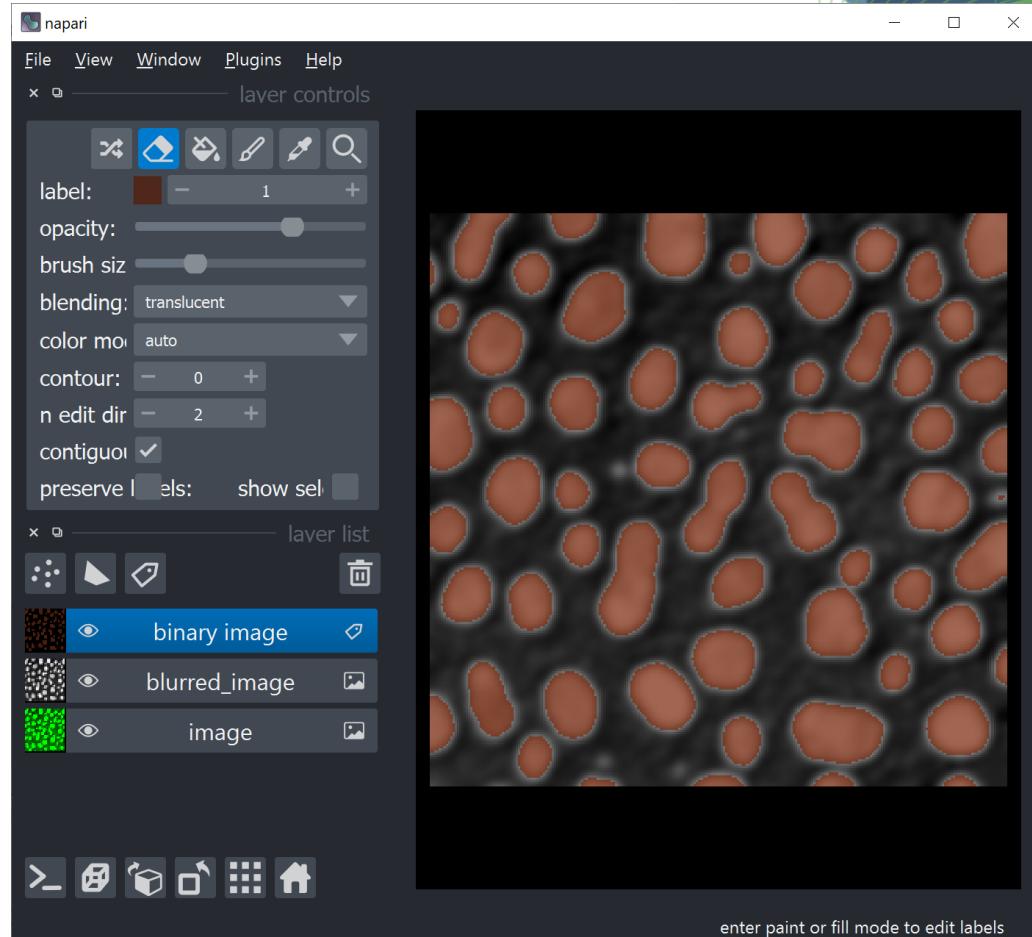
```
layer = viewer.add_image(image)  
layer.colormap = 'green'  
layer.contrast_limits = (0, 128)
```



# Napari – Python Scripting

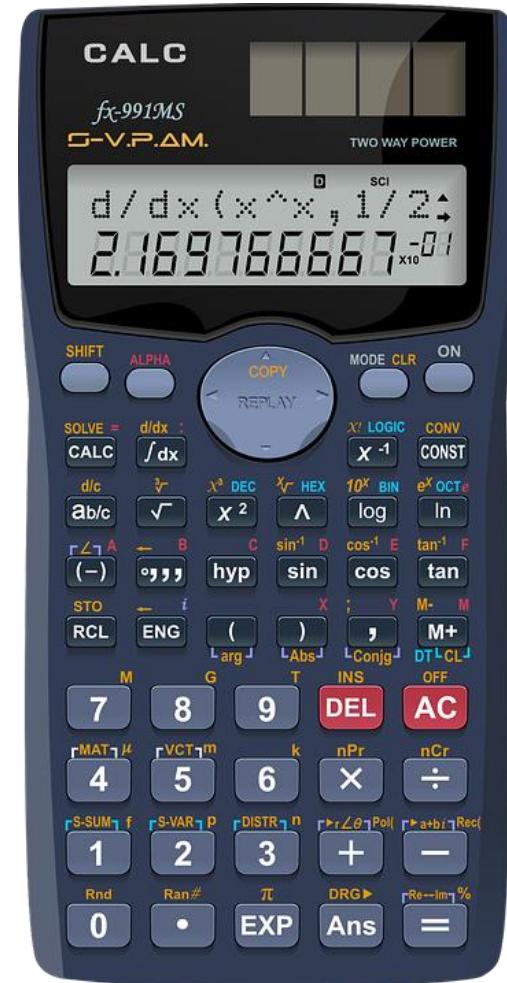
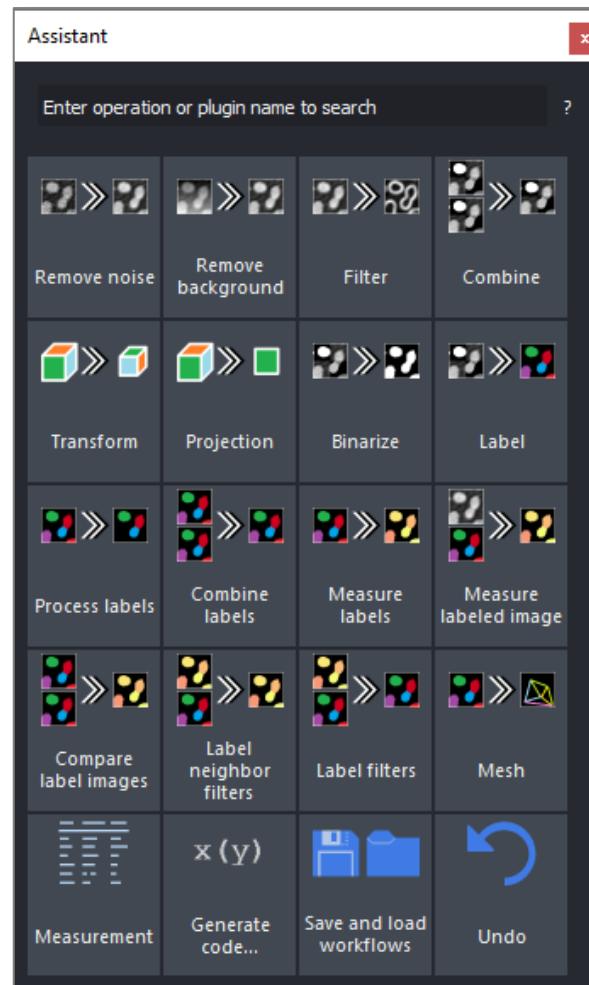
- Binary images and `label` images visualized as label layers

```
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

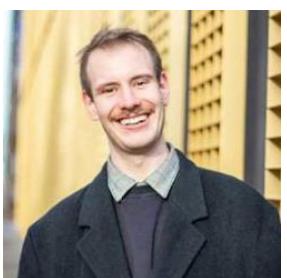
- A pocket-calculator-like interface to build image analysis workflows



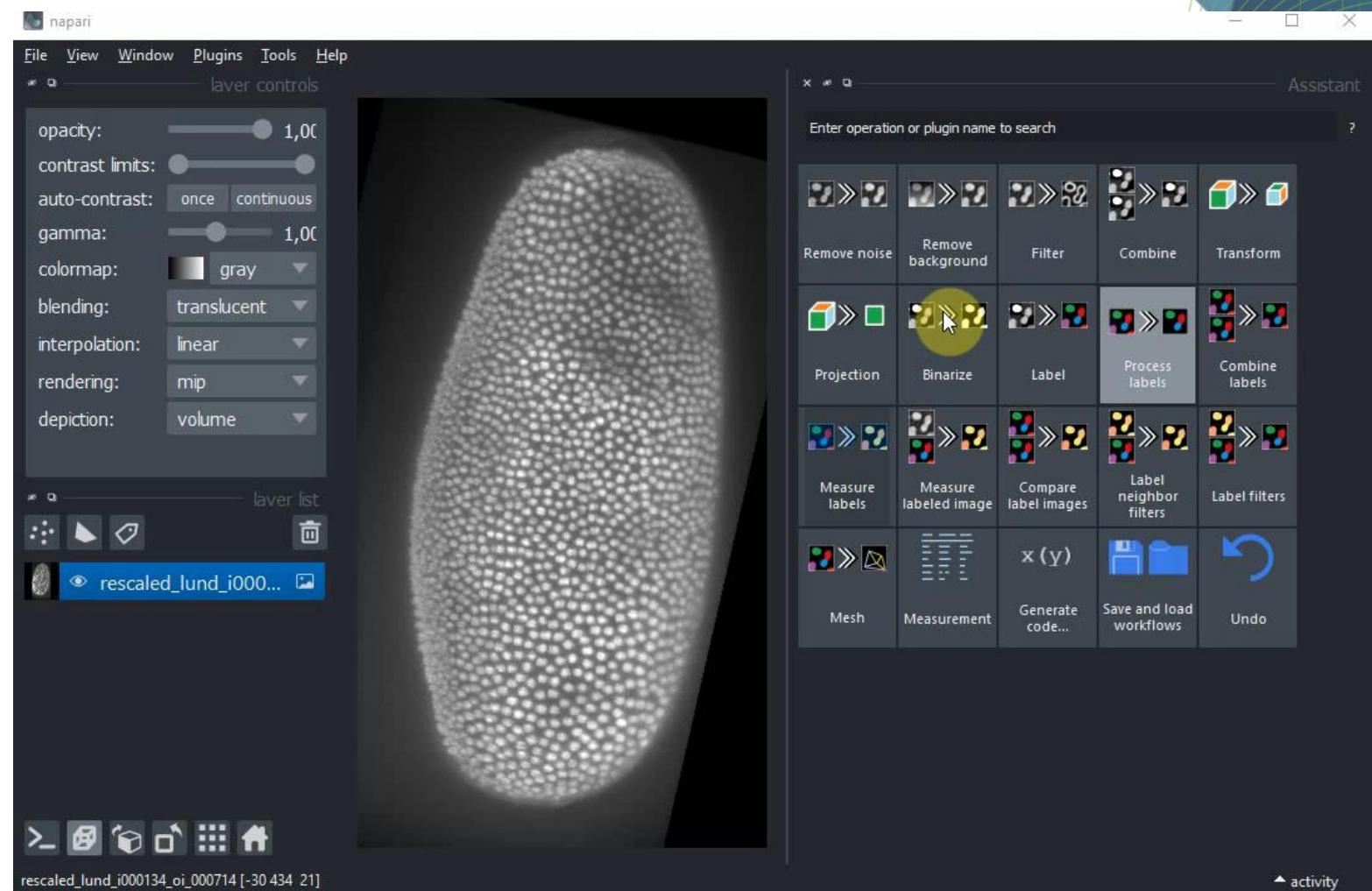
# The Napari Assistant

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

Big thanks to:

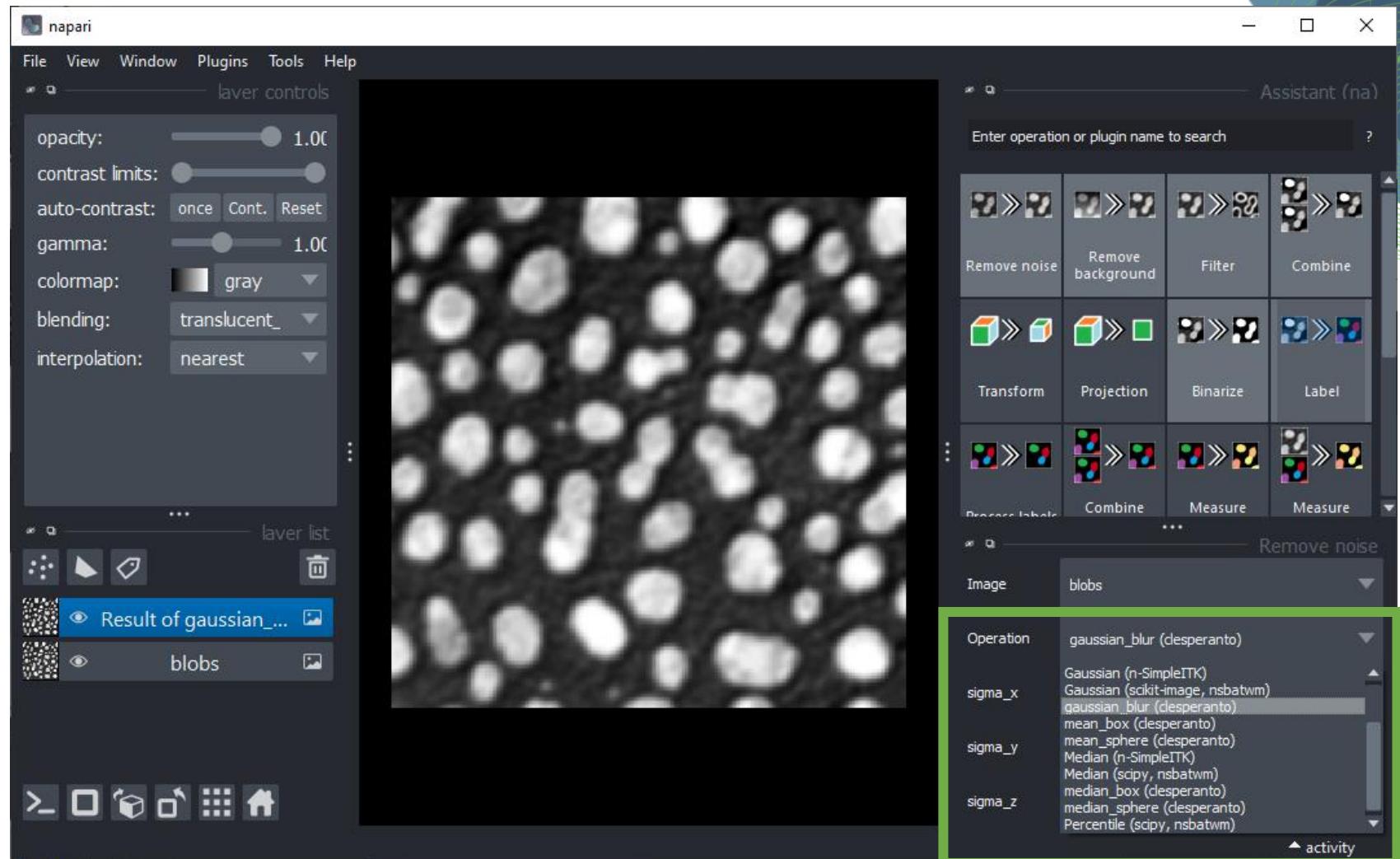


Ryan Savill  
@RyanSavill4



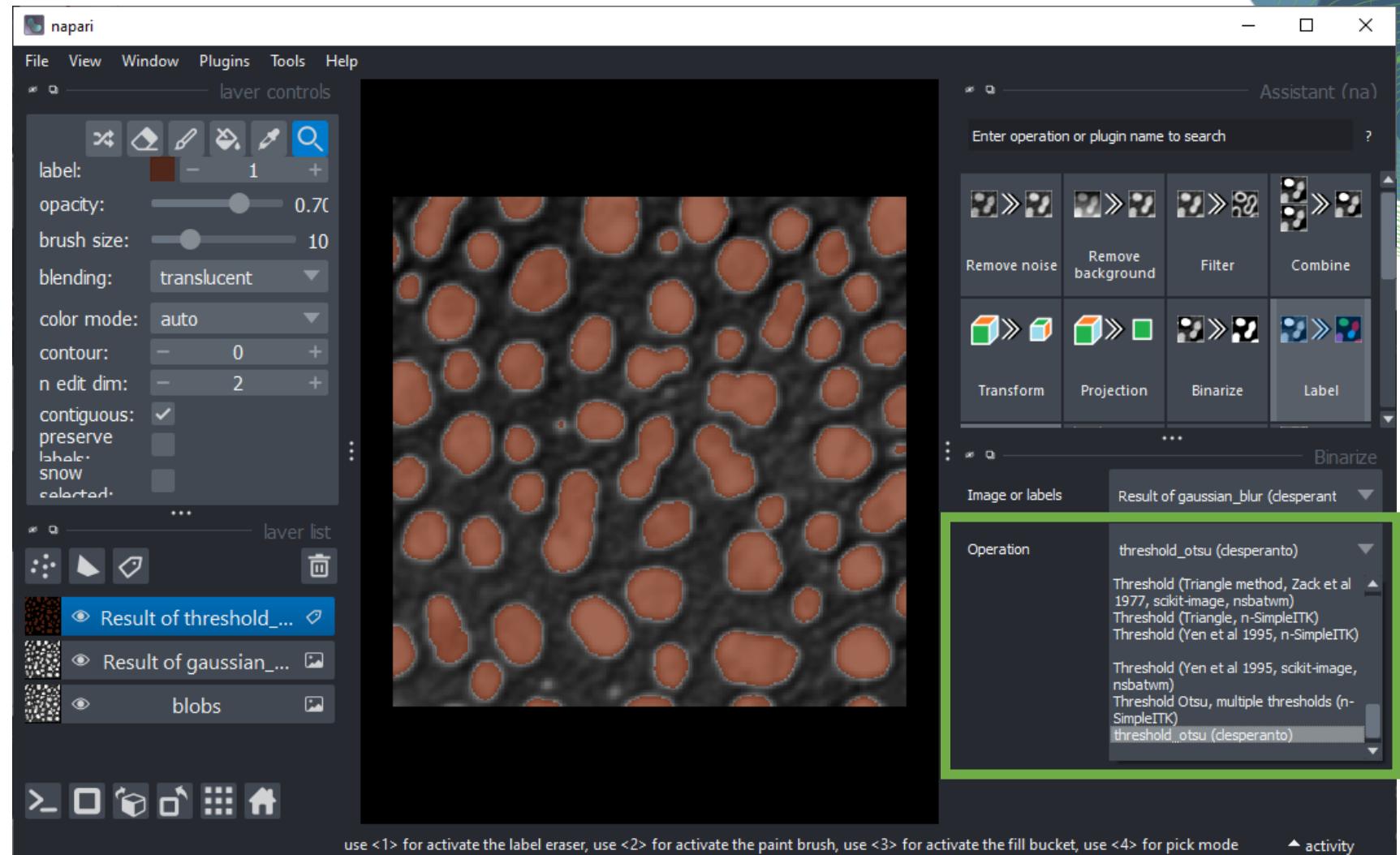
# Workflow building

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



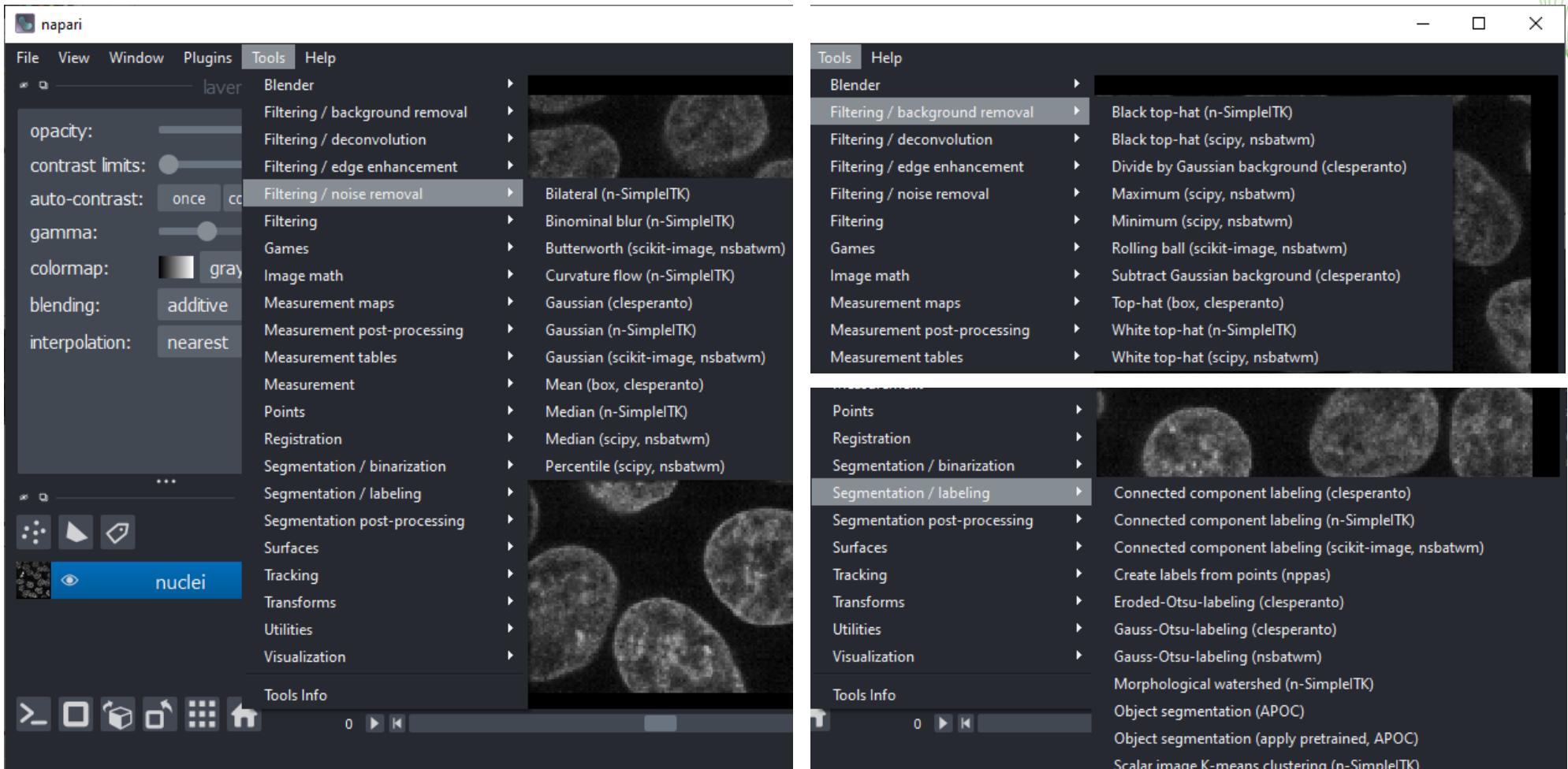
# Workflow building

- Try different binarization algorithms



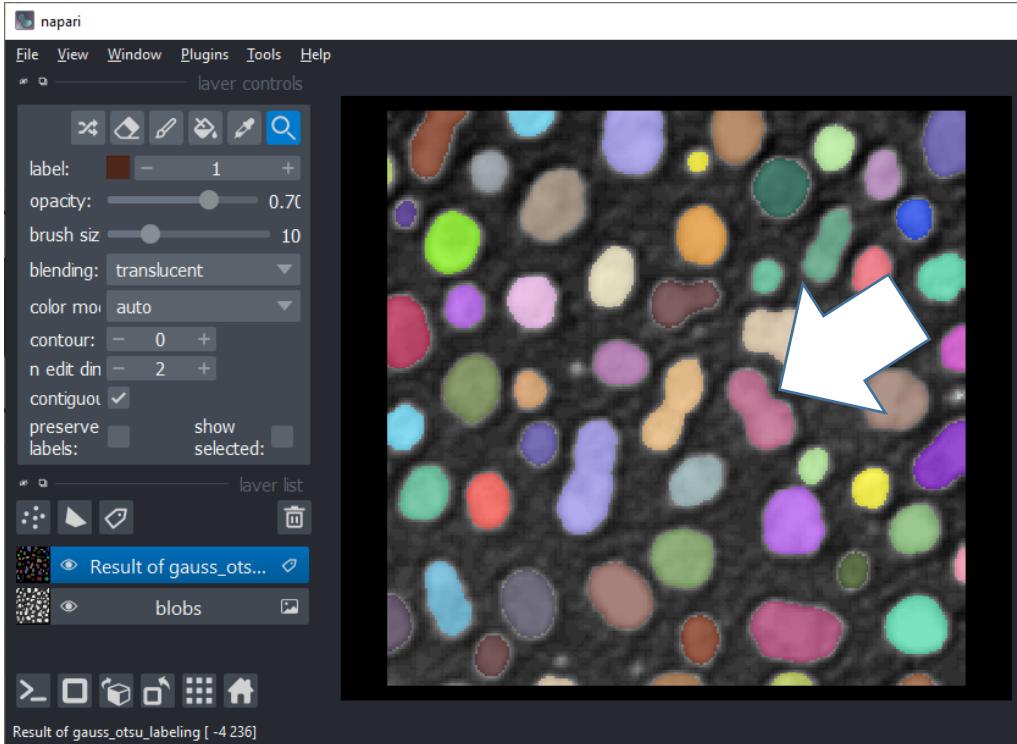
# The Tools menu

- Organized in categories

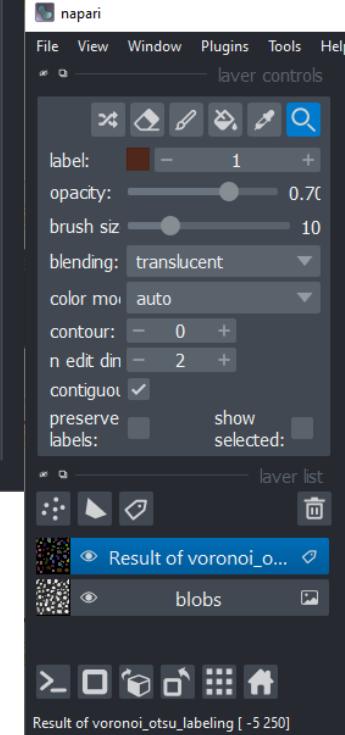
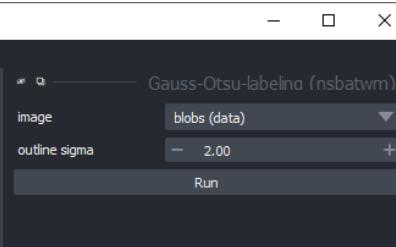


# Short-cuts: Voronoi-Otsu-Labeling

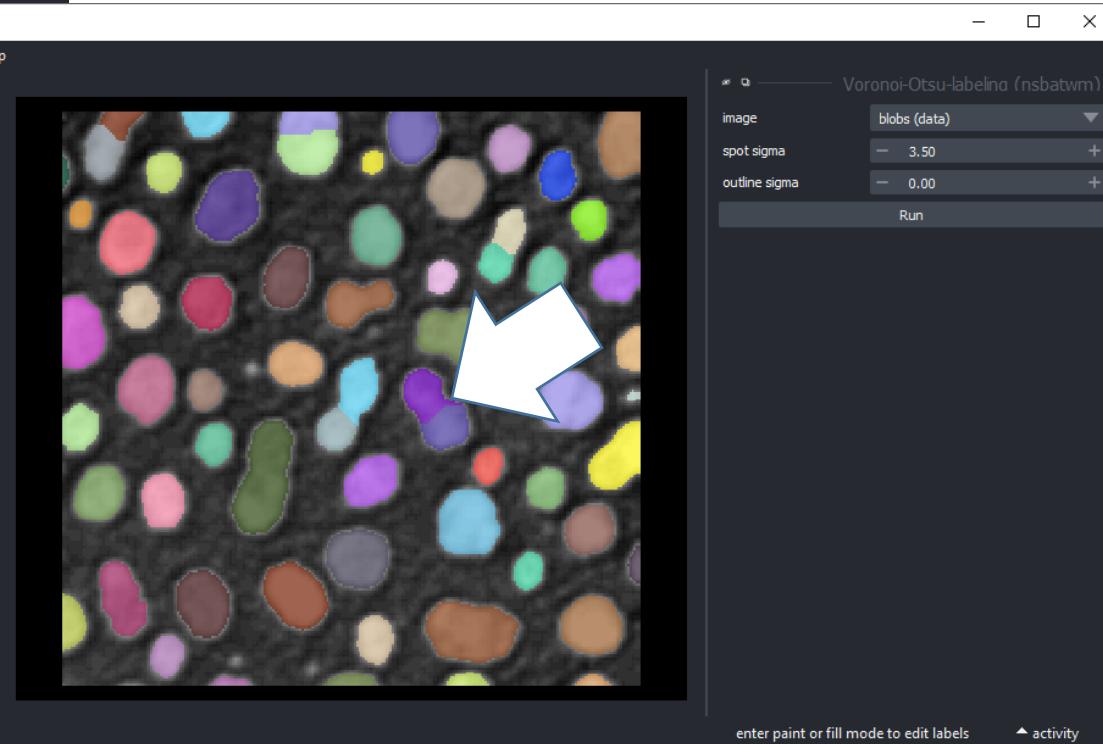
Also check out the Tools > Segmentation / labeling menu



Gauss-Otsu-Labeling

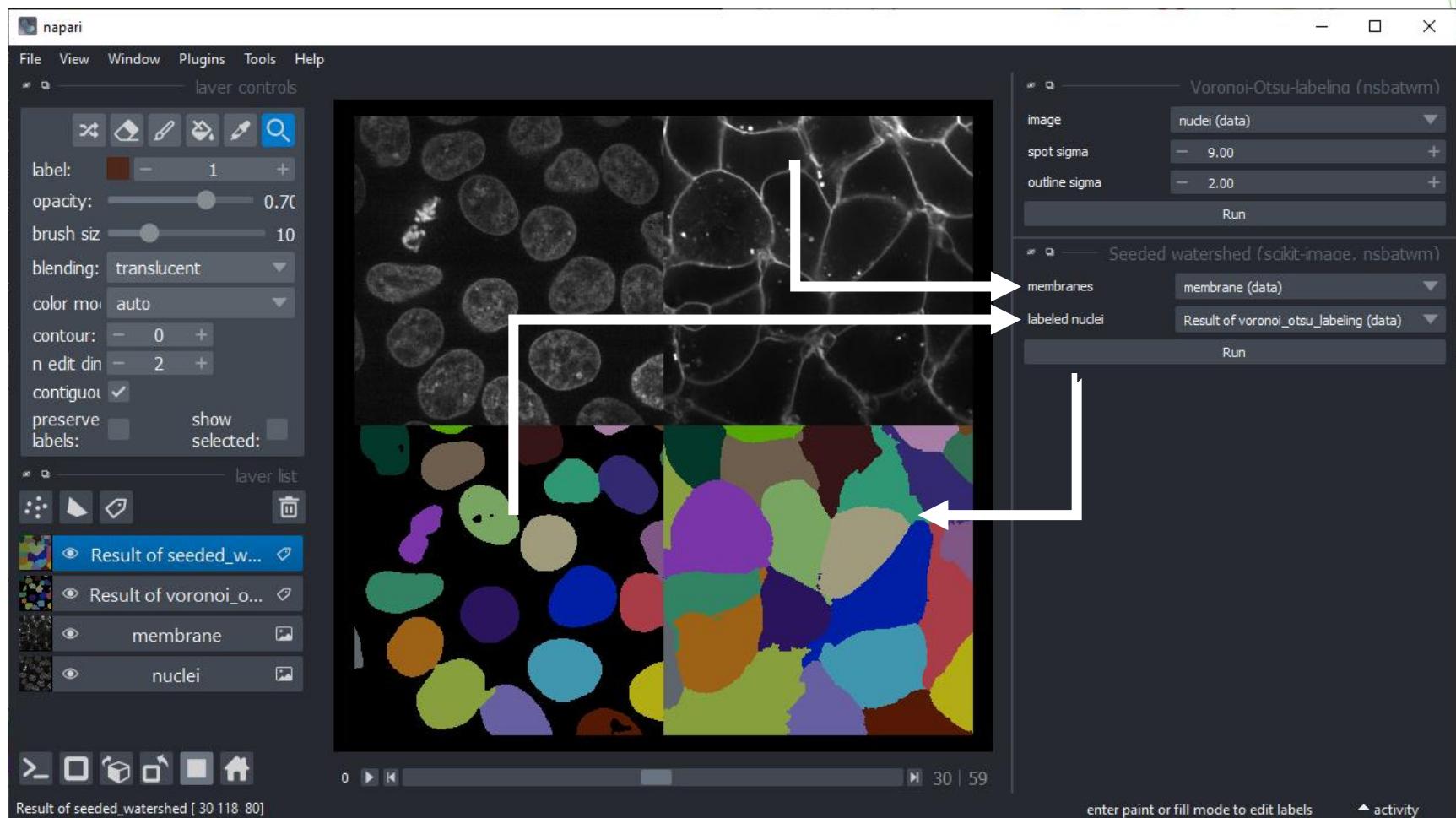


Voronoi-Otsu-Labeling



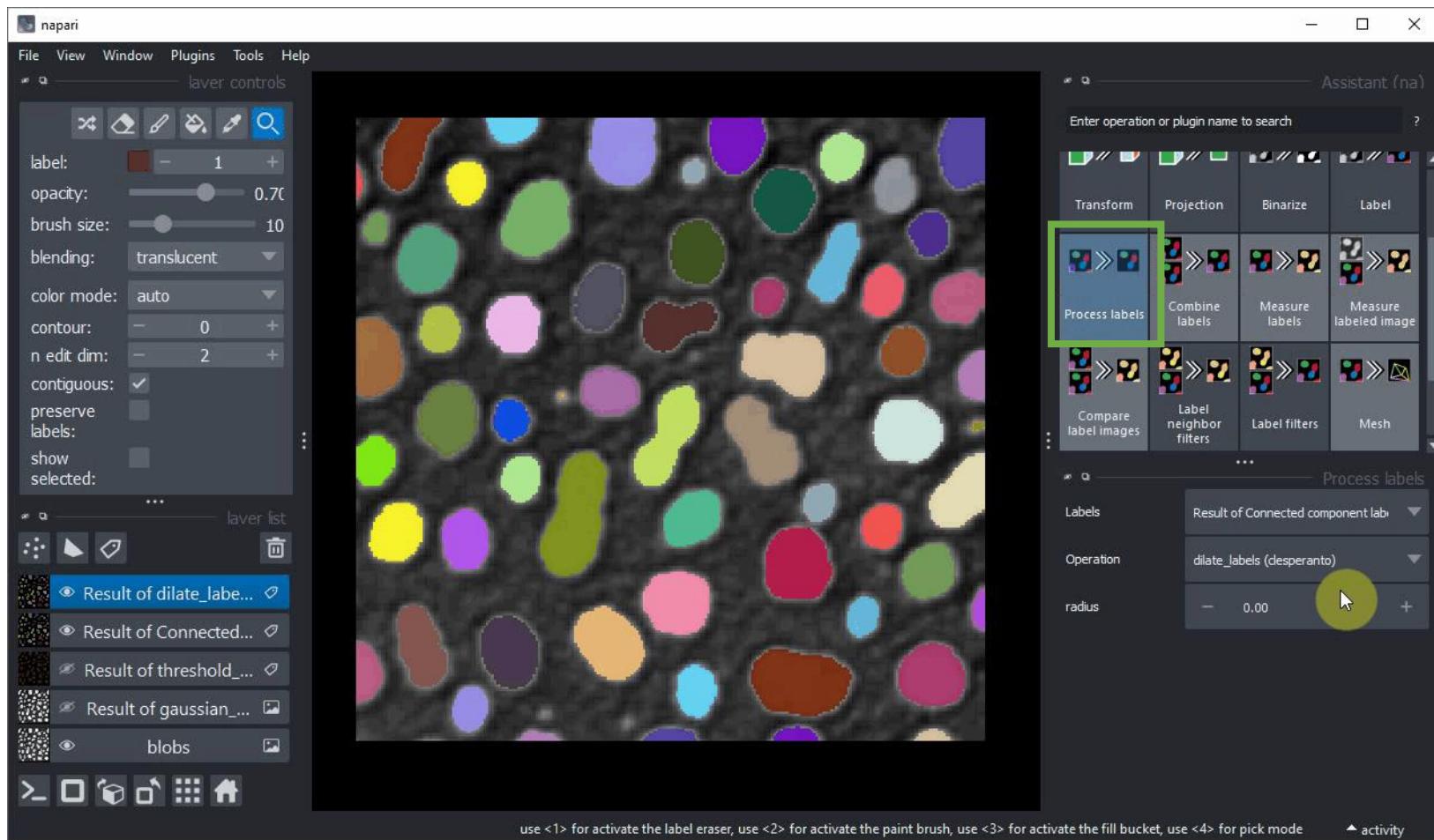
# Watershed

Also check out the Tools > Segmentation / labeling menu



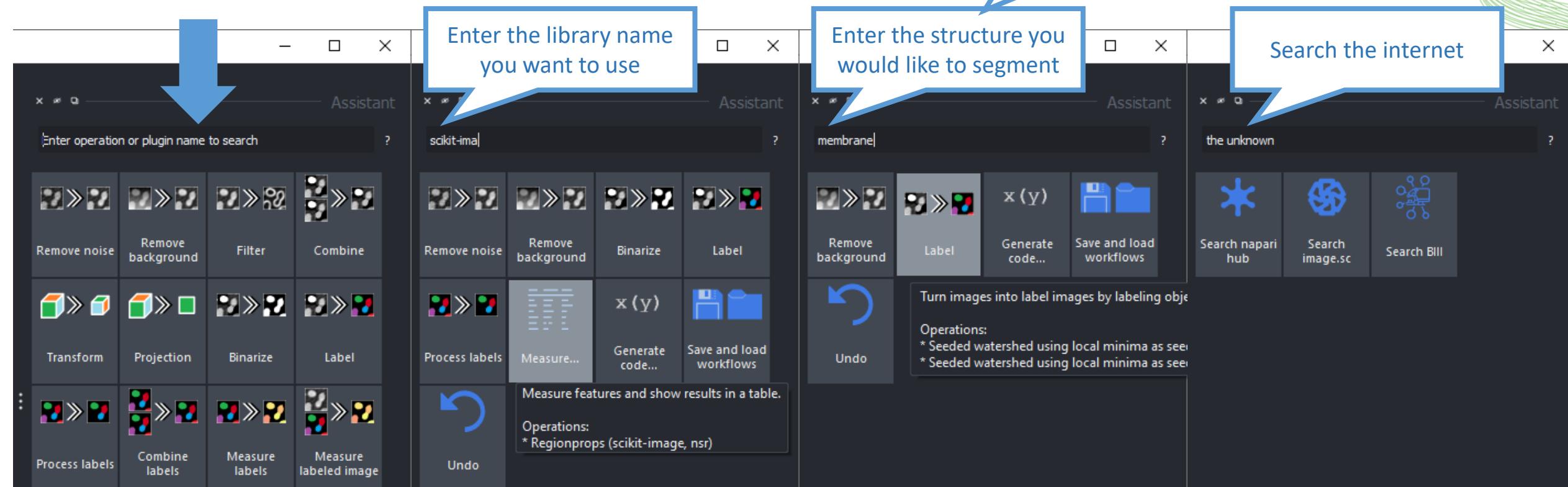
# Label erosion, dilation, opening, closing, ...

- In Napari Assistant: Process labels

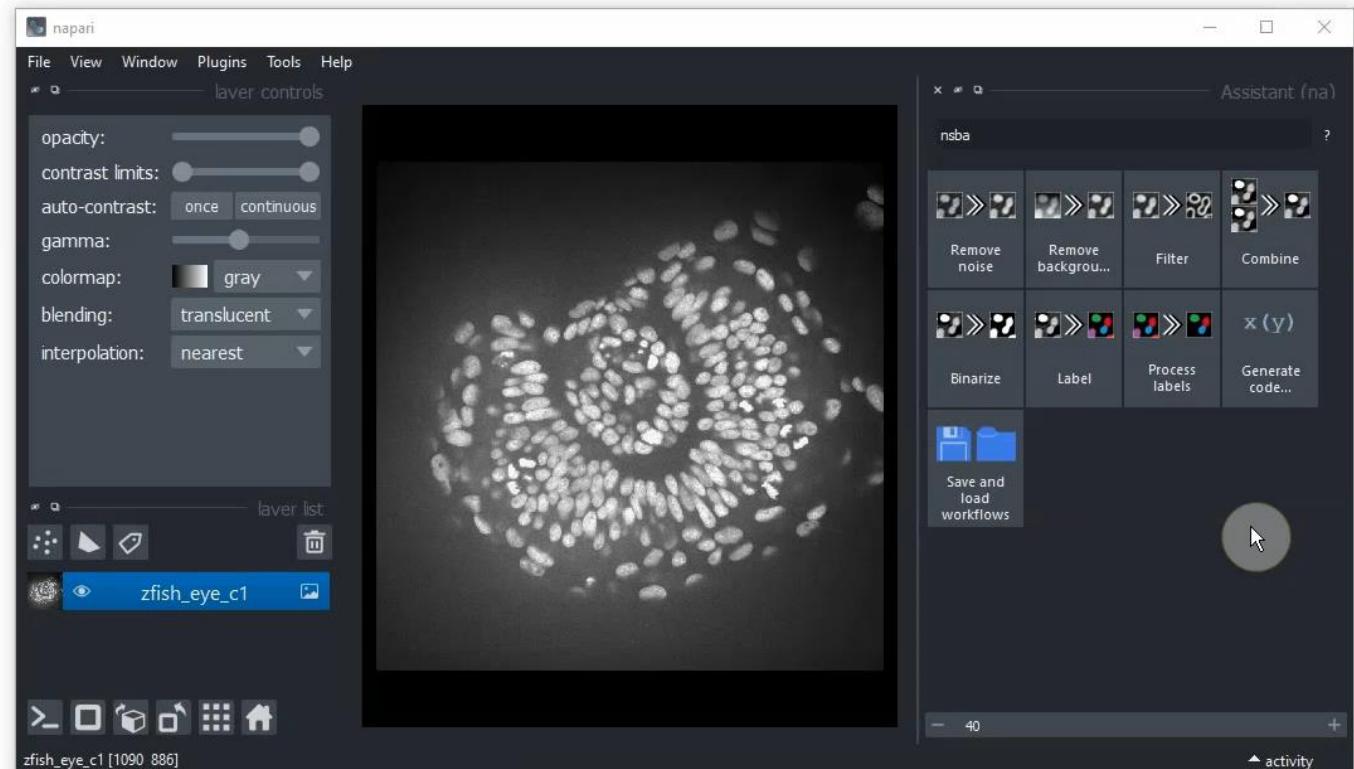


# Browse operations

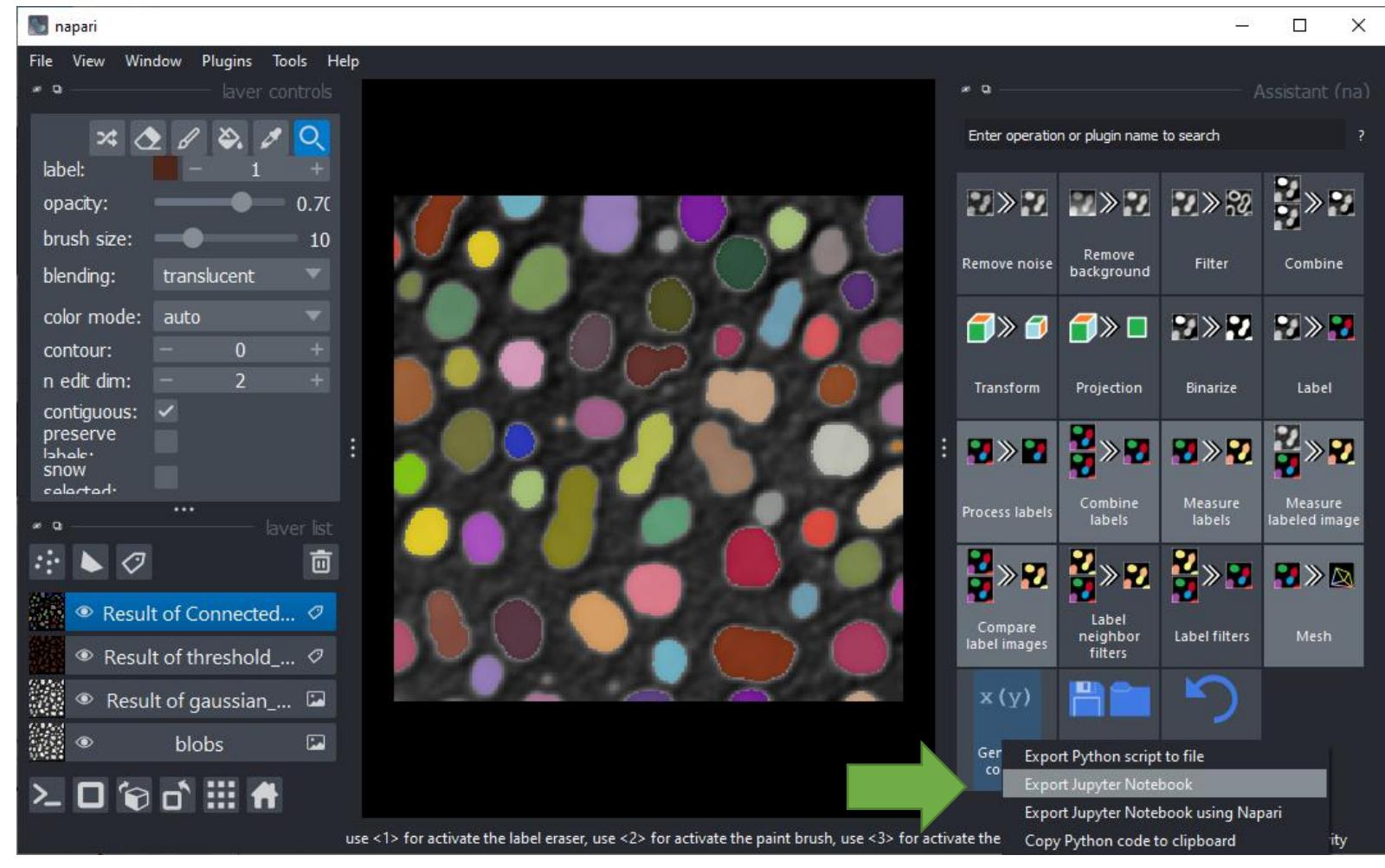
- Use the search...



# Export code to Jupyter Notebooks



# Export code to Jupyter Notebooks



The screenshot shows a Jupyter Notebook titled 'test.ipynb - JupyterLab'. The notebook contains the following code:

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

Below the code, there is a preview of a binary mask image with white blobs on a black background. To the right of the image, its properties are listed:

- cle.\_image
- shape (254, 256)
- dtype uint8
- size 63.5 kB
- min 0.0
- max 1.0

Further down the notebook, another section is titled 'connected component labeling' with the following code:

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

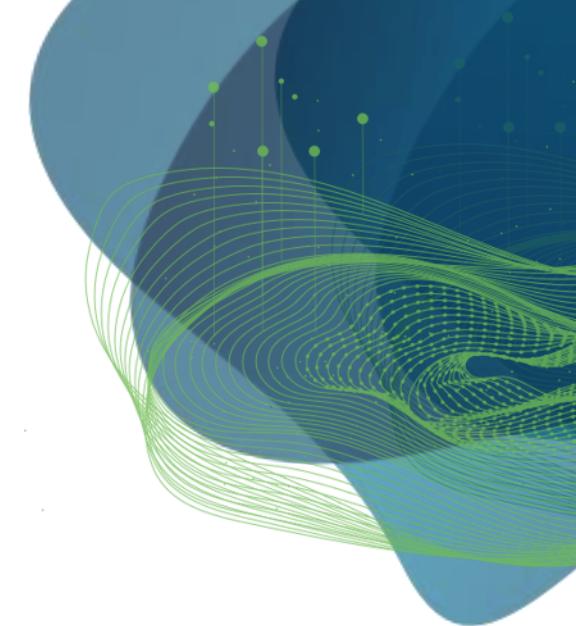
Below this, there is a preview of the original segmented image with colored blobs. To the right of the image, its properties are listed:

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



DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS  
AND ARTIFICIAL INTELLIGENCE



# Exercises

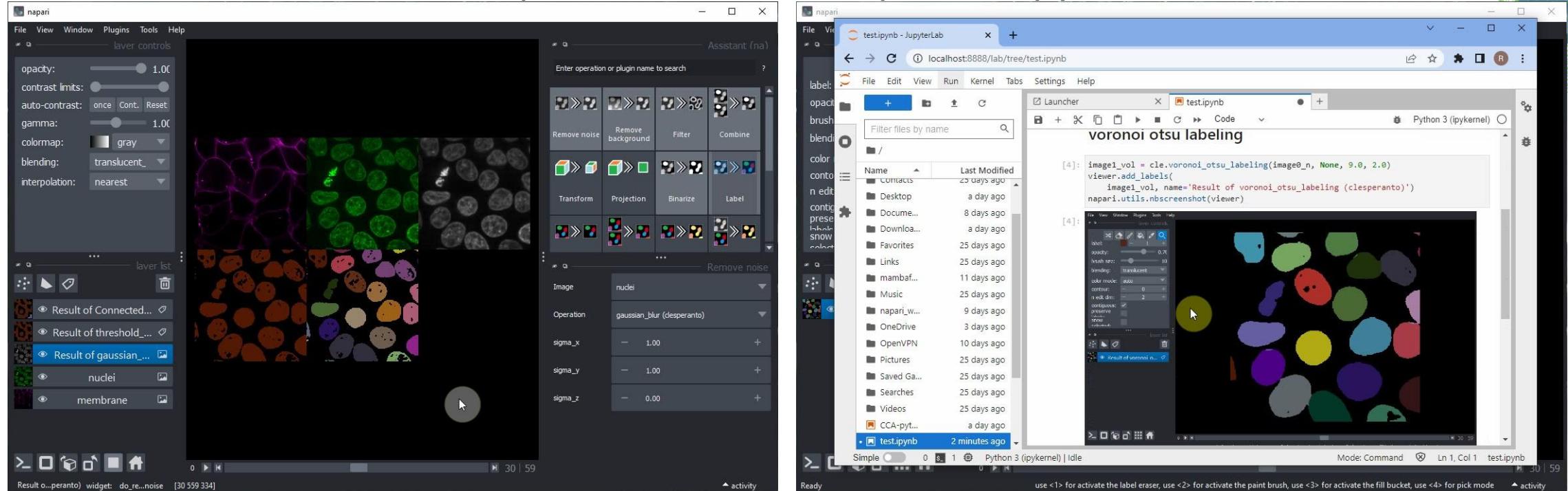
Robert Haase

GEFÖRDERT VOM



# Napari - Exercises

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



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

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

# Napari - Exercises

- Start using napari from Python

The image shows two side-by-side JupyterLab interfaces, each displaying a notebook titled "napari\_intro.ipynb".

**Left Notebook Content:**

- Title:** Opening the napari Viewer
- Text:** In order to open the viewer, we first have to import napari
- Code Cell [2]:** `import napari`
- Text:** Now, we can open the viewer with the following command:
- Code Cell [3]:** `viewer = napari.Viewer()`
- Text:** Napari should open in a separated window. Some warning messages in the cell above are normal.
- Text:** Let's show a screenshot of the viewer here. We pass the variable viewer to the function.
- Code Cell [4]:** `napari.utils.nbscreenshot(viewer)`
- Image:** A screenshot of the Napari viewer window showing a grayscale brain scan with a red segmentation mask overlaid.

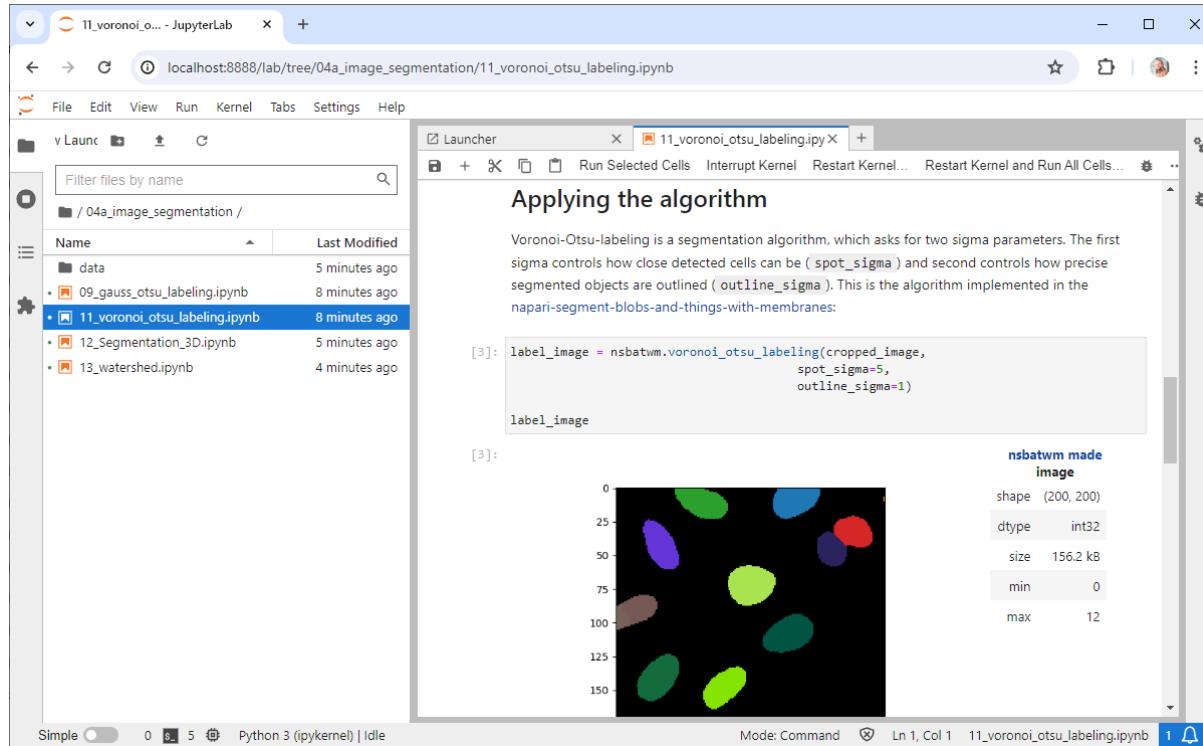
**Right Notebook Content:**

- Title:** Segmentation visualization
- Text:** You can also add a segmentation result to the viewer, which will get overlaid with the original image.
- Code Cell [13]:**

```
blurred = gaussian(mri, sigma=5)  
binary_image = blurred > threshold_otsu(blurred)  
viewer.add_labels(binary_image)  
napari.utils.nbscreenshot(viewer)
```
- Image:** A screenshot of the Napari viewer window showing a grayscale brain scan with a binary segmentation mask overlaid.

# Image segmentation exercises

- Try out segmentation algorithms and apply them to other datasets



localhost:8888/lab/tree/04a\_image\_segmentation/11\_voronoi\_otsu\_labeling.ipynb

File Edit View Run Kernel Tabs Settings Help

Applying the algorithm

```
label_image = nsbatwm.voronoi_otsu_labeling(cropped_image,
                                              spot_sigma=5,
                                              outline_sigma=1)
```

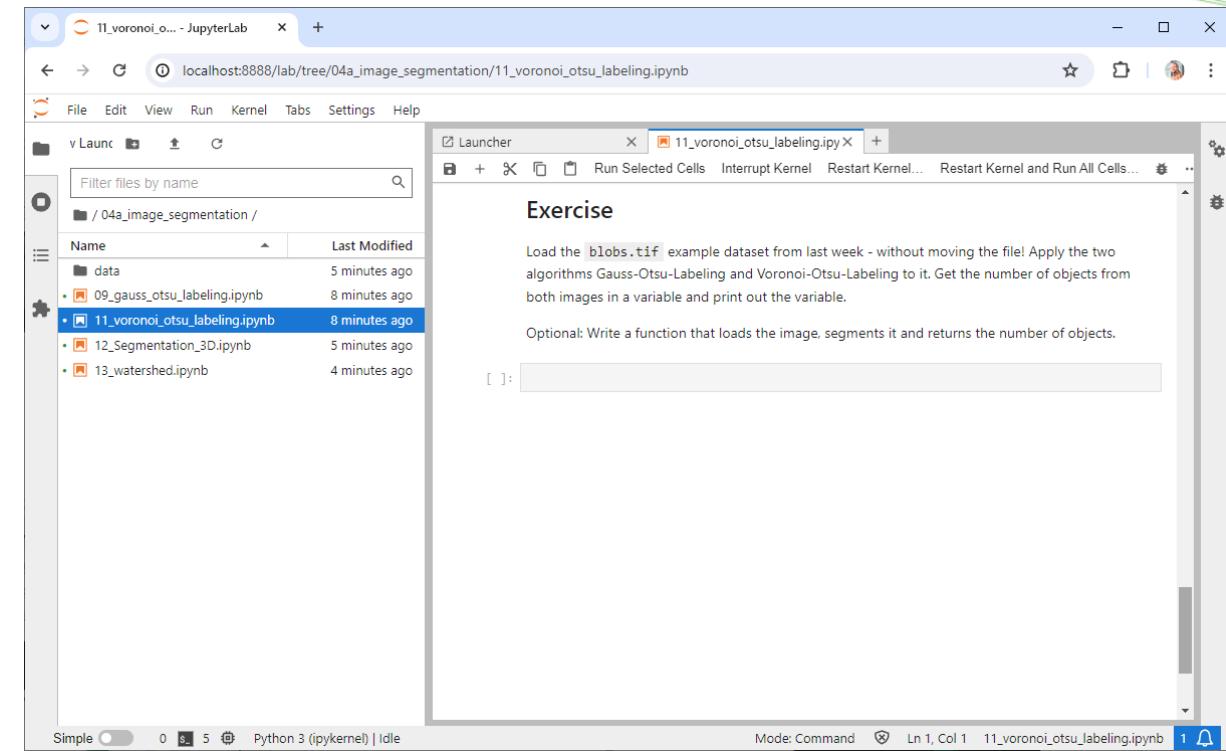
label\_image

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

0 25 50 75 100 125 150

0 25 50 75 100 125 150

Mode: Command Ln 1, Col 1 11\_voronoi\_otsu\_labeling.ipynb 1



localhost:8888/lab/tree/04a\_image\_segmentation/11\_voronoi\_otsu\_labeling.ipynb

File Edit View Run Kernel Tabs Settings Help

Exercise

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

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

[ ]:

Mode: Command Ln 1, Col 1 11\_voronoi\_otsu\_labeling.ipynb 1