

Bio-image Analysis with Python and Napari

Marcelo Leomil Zoccoler, Johannes Müller

Schedule

13:00 Introduction to Image Analysis with Python

- Bio-Image Analysis
- Napari
- Jupyter Lab

14:00 Python basics

- Basic Python syntax (strings, numbers, put assign variables, indexing)
- Variables and objects (lists, nd-arrays)
- Loops & conditions (for, if, functions)

15:00 Image Analysis

- Images
- Filtering
- Background subtraction
- Morphological operators

16:00 Image Segmentation

17:00 Feature extraction

Short breaks after
every session

Introduction to Bio-Image Analysis

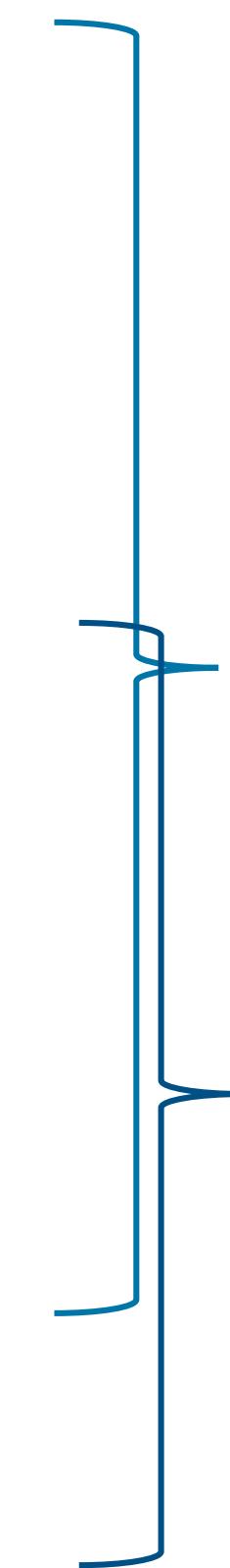
Marcelo Leomil Zoccoler

With material from:

Robert Haase, TU Dresden and Benoit Lombardot, MPI CBG

Quantitative bio-image analysis

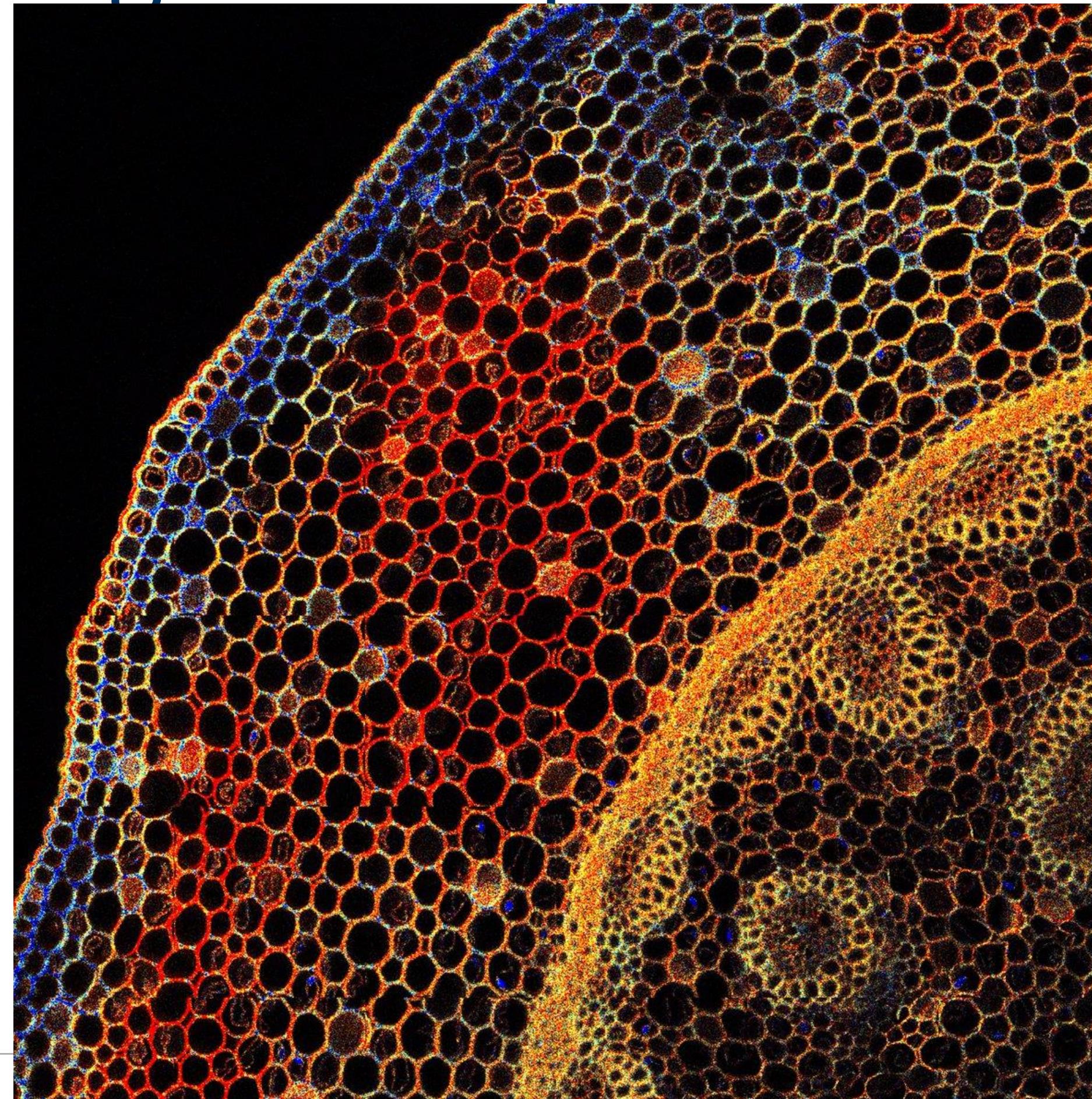
Deriving quantitative information from images of biological samples taken with microscopes



cat height = 1.5 x microscope height

Quantitative bio-image analysis

Deriving quantitative information from images of biological samples taken with microscopes



How many cells are there in the outer ring?

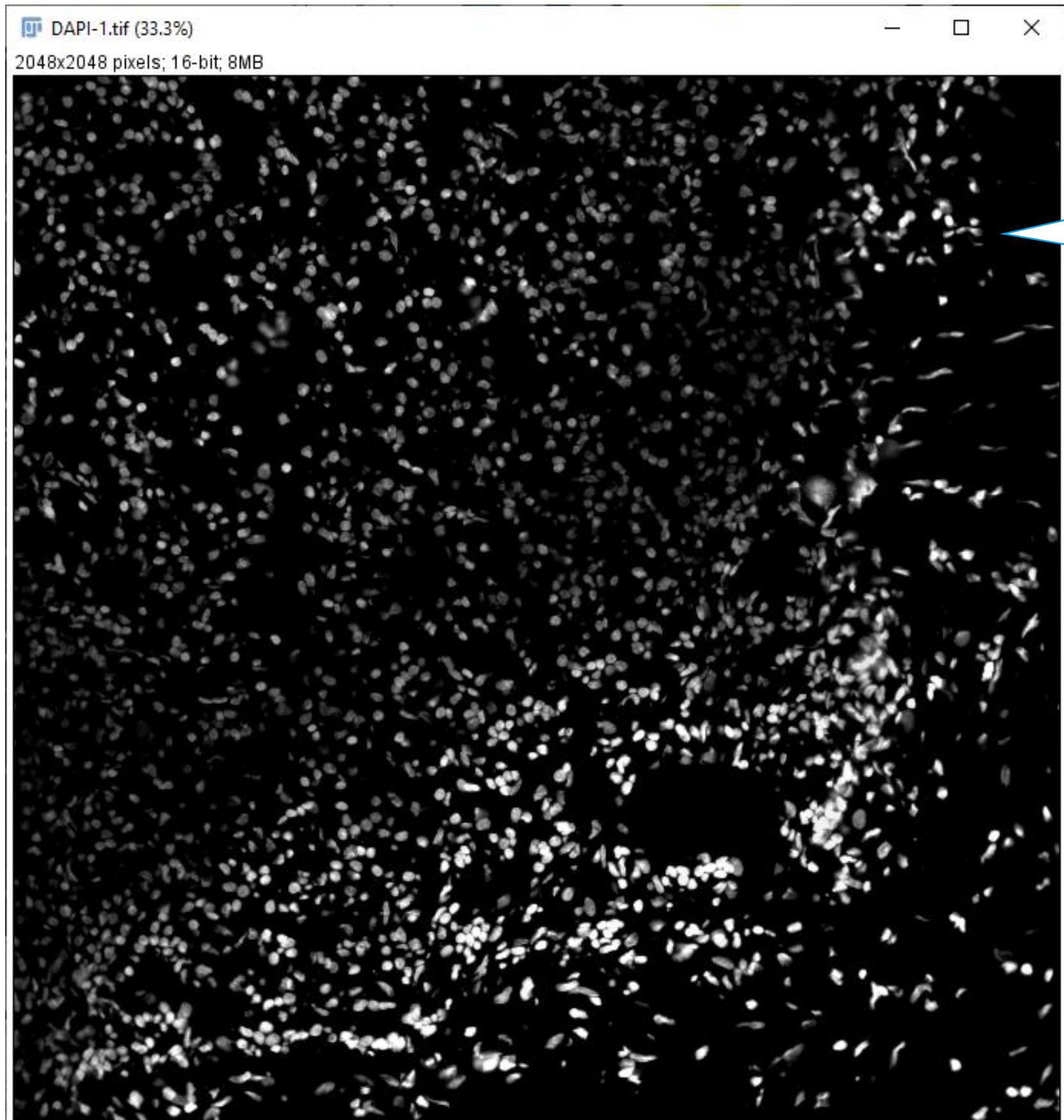
How high is the intensity in the inner ring?

How is the signal in the blue channel distributed?

Image data source: Lorenzo Scipioni, U. California / Irvine @LorenzoScipion3

Objective bio-image analysis

Measurements should be objective, not influenced by human interpretation



Nuclei in this
image are ...

... more dense than
in this image.

Use automation for
less subjective
analysis.

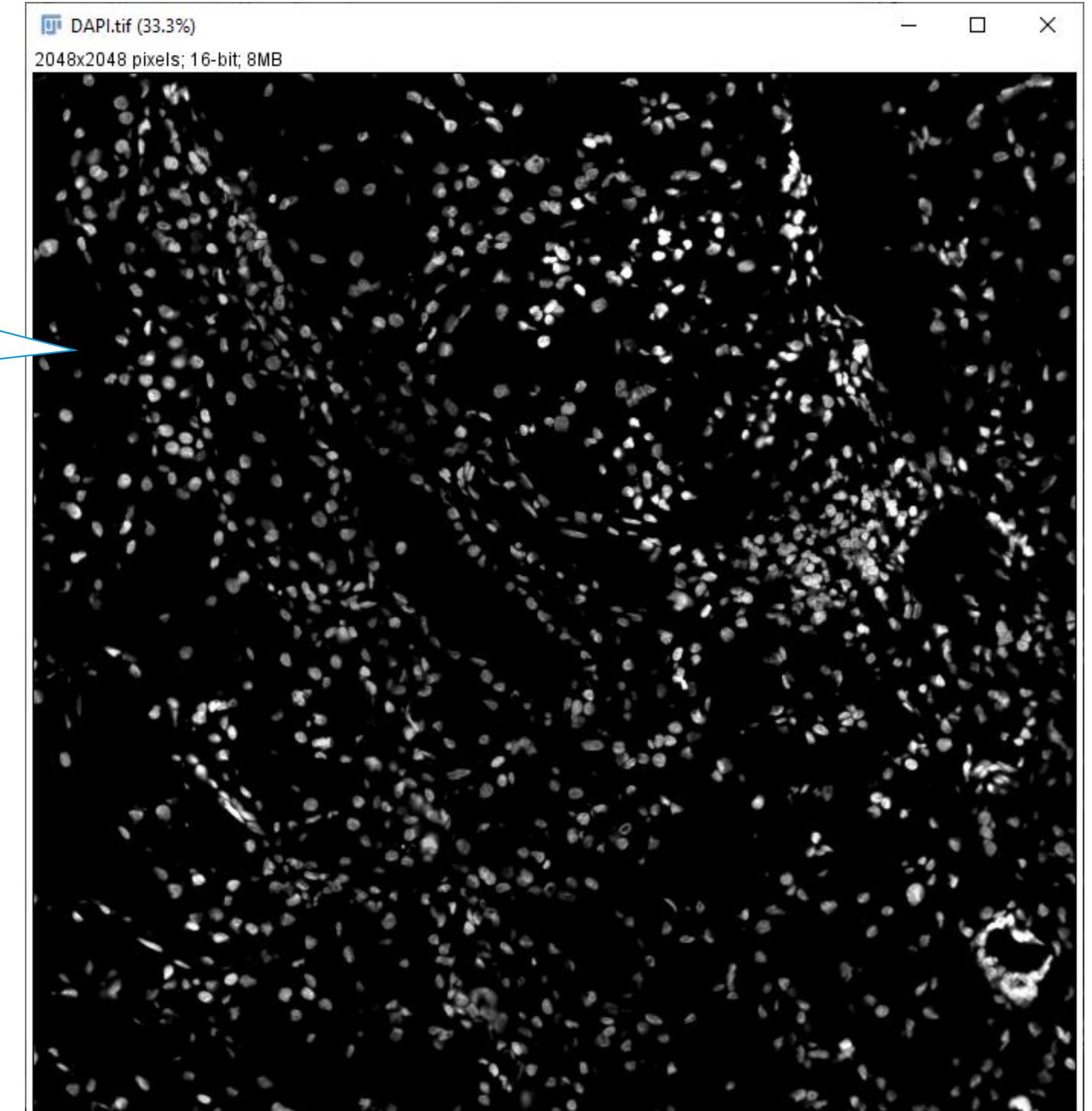
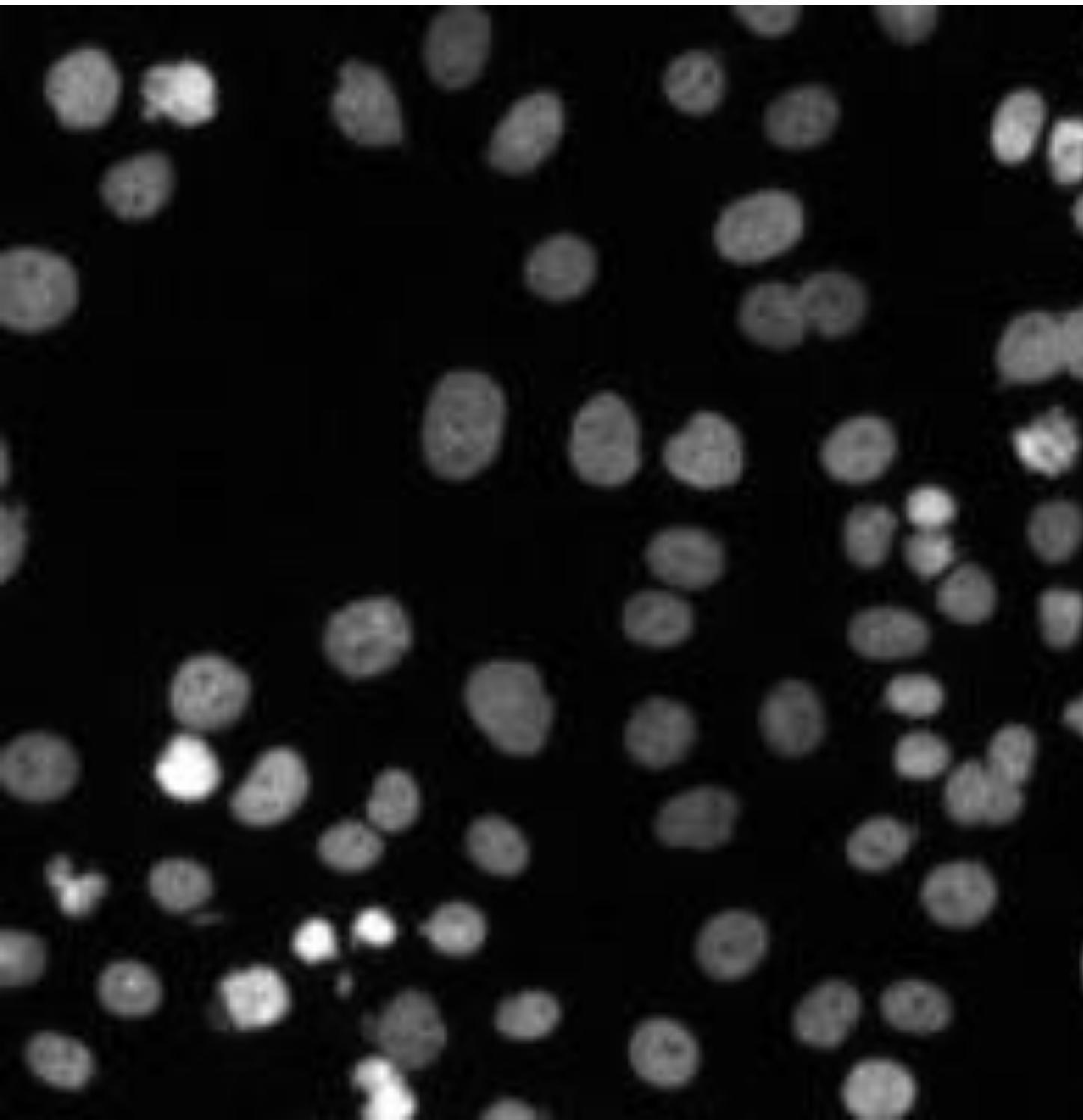


Image data source: Pascual-Reguant, Anna. (2021). Immunofluorescence staining of a human kidney (#2, peri-tumor area) obtained by MELC [Data set]. Zenodo. <http://doi.org/10.5281/zenodo.4434462> licensed CC-BY 4.0

Reliable bio-image analysis

Algorithms must be reliable (trustworthy). Visualization helps gaining trust in automated methods.

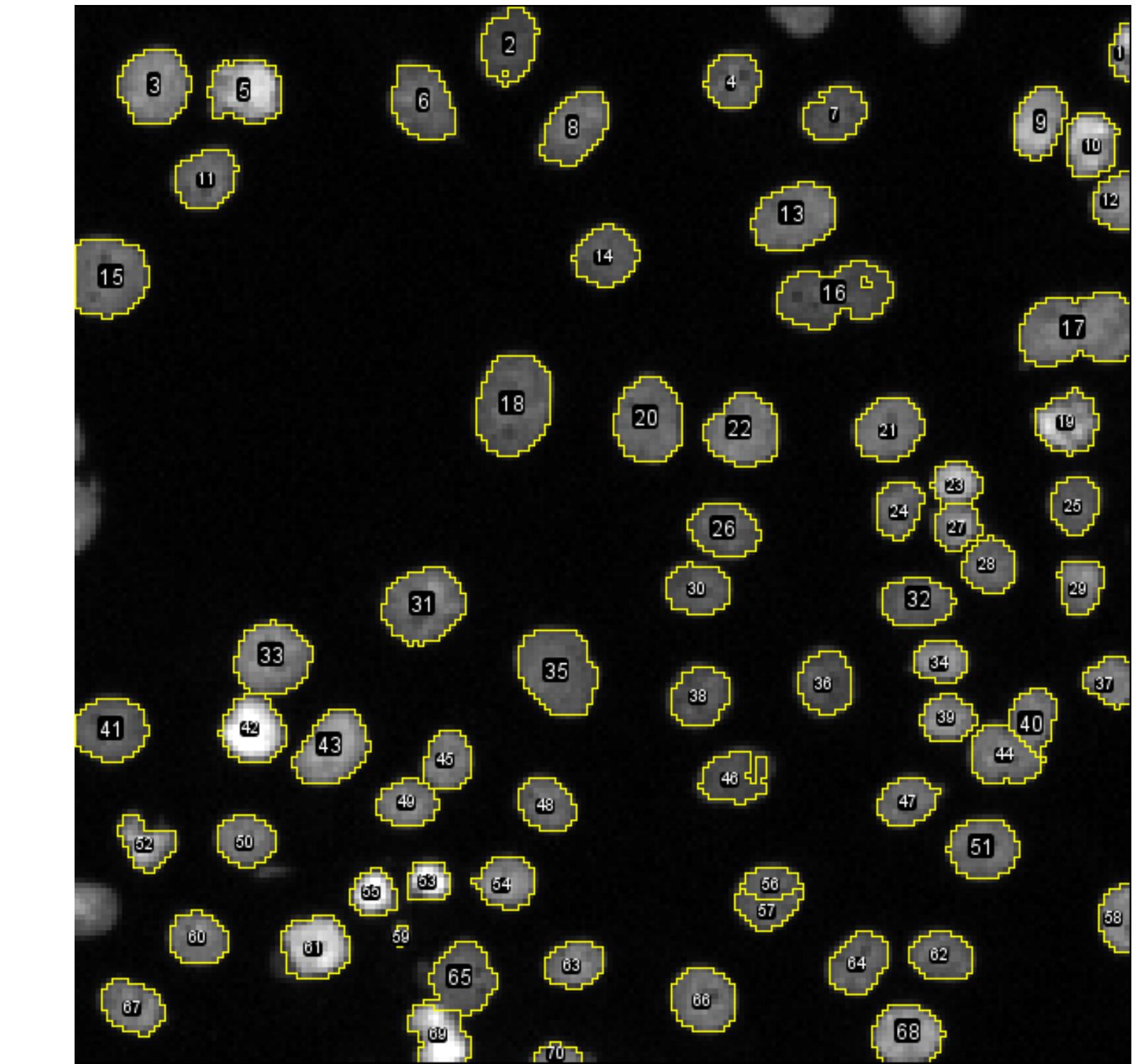
Original image



Label image



Overlay



There are 70
nuclei in this
image.

Reproducible bio-image analysis

"The image data was analyzed with ImageJ."

Can you reproduce what they did?

Can you reproduce what they did?

Trailer: Bio-image Analysis with Python

In the following chapters we will dive into image analysis, machine learning and bio-statistics using Python. This first notebook serves as a trailer of what we will be doing.

Python notebooks typically start with the imports of Python libraries which the notebook will use. The reader can check first if all those libraries are installed before going through the whole notebook.

```
import numpy as np
from skimage.io import imread, imshow
import pyclesperanto_prototype as cle
from cellpose import models, io
from skimage import measure
import pandas as pd
import apoc
```

Working with image data

We start with loading the image data of interest. In this example we load an image showing a zebrafish eye, courtesy of Mauricio Rocha Martins, Norden lab, MPI CBG Dresden.

```
# open an image file
multichannel_image = imread("../data/zfish_eye.tif")
print("Image size", multichannel_image.shape)
```

Image size (1024, 1024, 3)

Image analysis is part of the experiment

Think about how to analyze your images before starting the experiment.

- Consider adapting your experiment so that quantitative image analysis can be performed easily.

Think about controls, counter-proves, an easy to falsify null-hypothesis.

- Be a lazy scientist. Do simple experiments.

How can you exclude yourself from the experiment?

- Think of blinding yourself or fully automate analysis.

One experiment usually answers just one or less questions.

Image analysis is part of the experiment

Talk to image-analysis / data-science experts early during your project.

Not one week before the
submission deadline.

Napari

Marcelo Leomil Zoccoler

With material from:

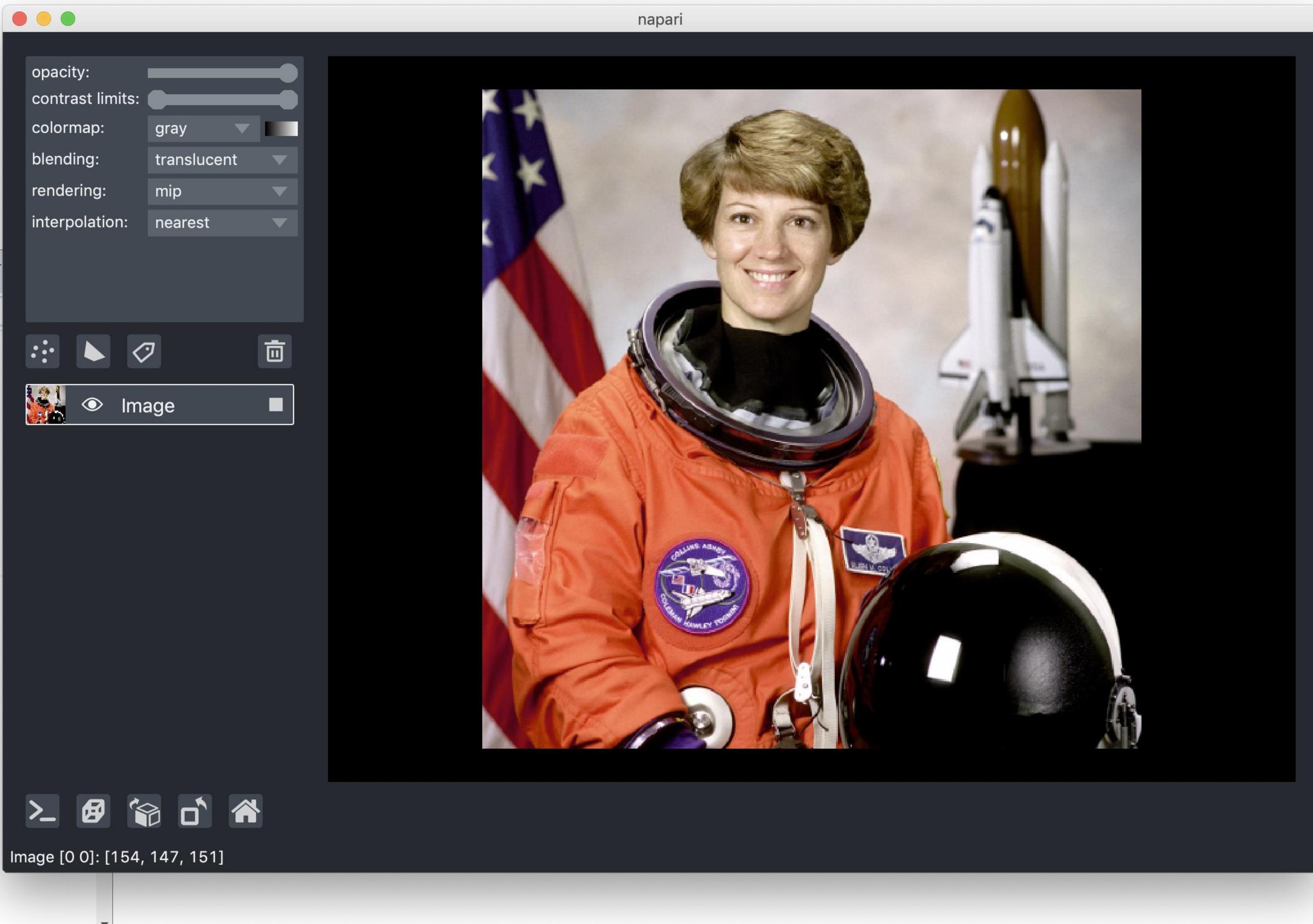
Robert Haase, PoL

Napari: 3D viewer for Python

Multi-dimensional image viewer in python

<https://napari.org/>

The screenshot shows the official website for napari. At the top, there's a navigation bar with links for Home, Forum, License, Build, Codecov, Python versions, PyPI version, Downloads, Status, Code Style, and DOI. Below the navigation is a large logo featuring a stylized blue and green blob on a purple square. The main content area has a dark background with white text. It features the word "napari" in a large font, followed by "multi-dimensional image viewer for python". Below this, there's a brief description of what napari is, mentioning it's a fast, interactive, multi-dimensional image viewer for Python, built on Qt, Vispy, and the scientific Python stack (numpy, scipy). It also notes that the project is in an alpha stage and subject to breaking changes. A sidebar on the left contains links for Community, Tutorials, Plugins, Release Notes, API Reference, Roadmaps, and Developer Guides.



Napari: 3D viewer for Python



Image data source: Daniela Vorkel, Myers lab, MPI-CBG/CSBD

Napari user interface

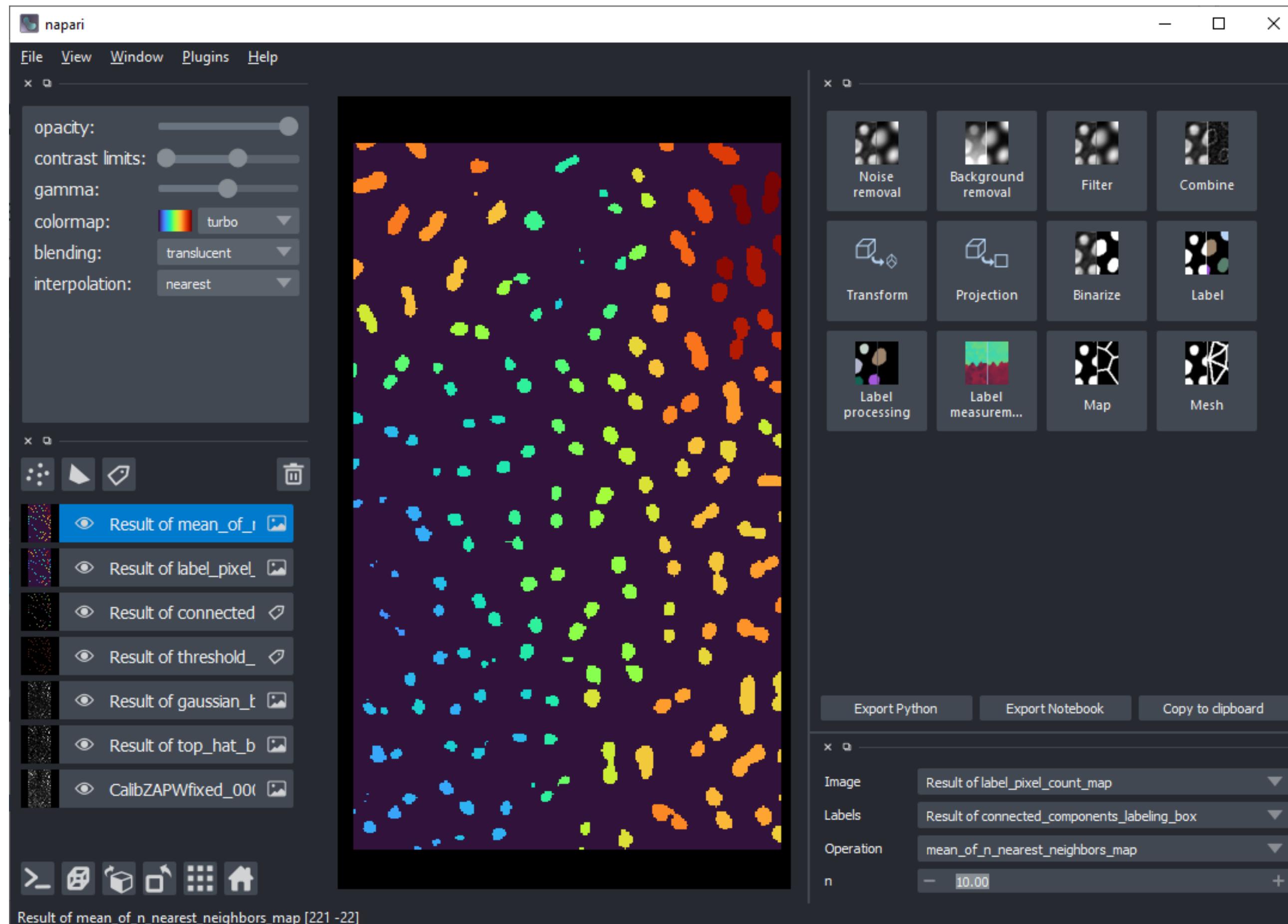
View configuration / tools

```
layer.opacity = 0.5
```

Layers

```
layer.visible = False
```

Viewer controls



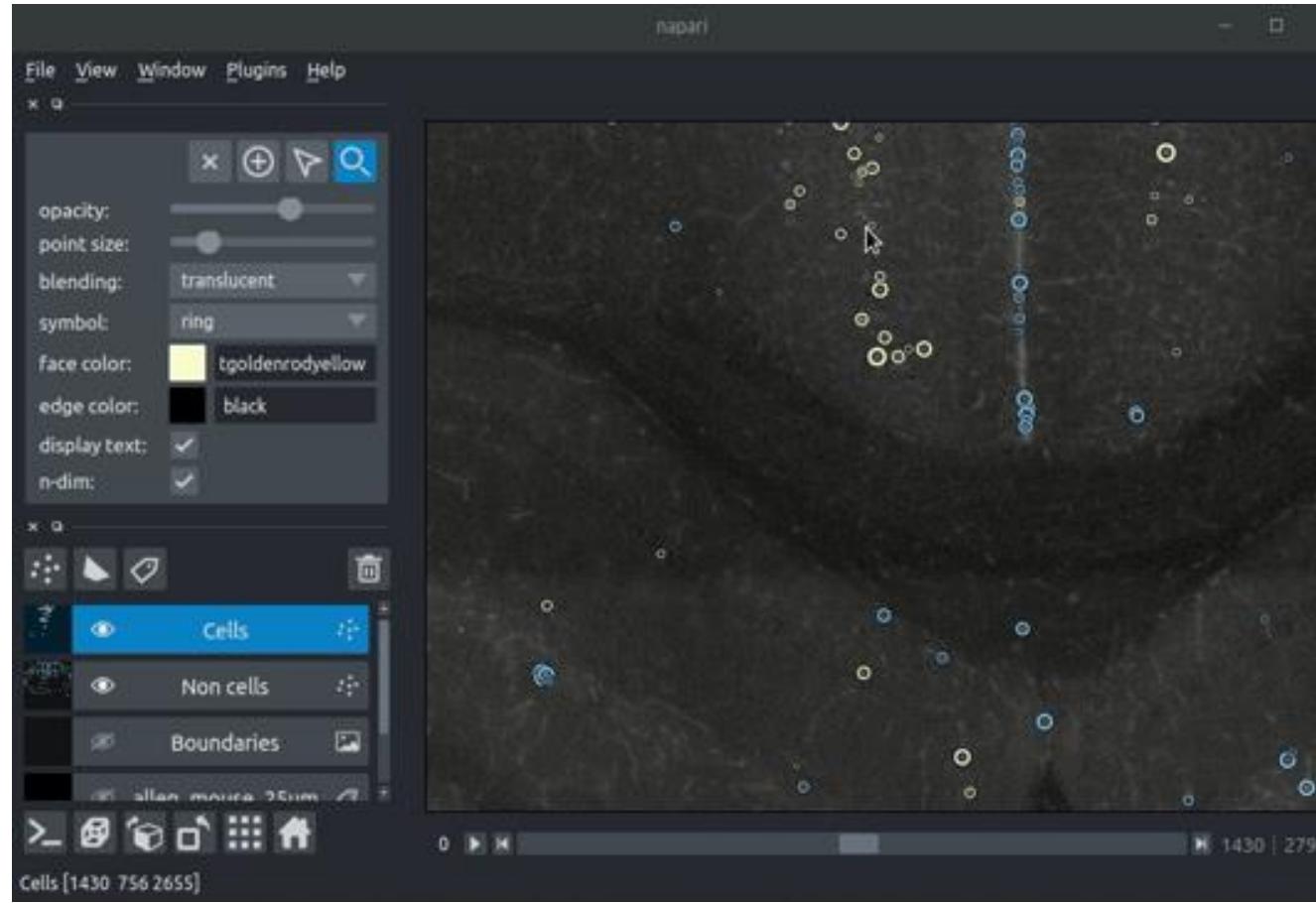
Dock widgets
(custom plugins)

Function widgets
(custom plugins)

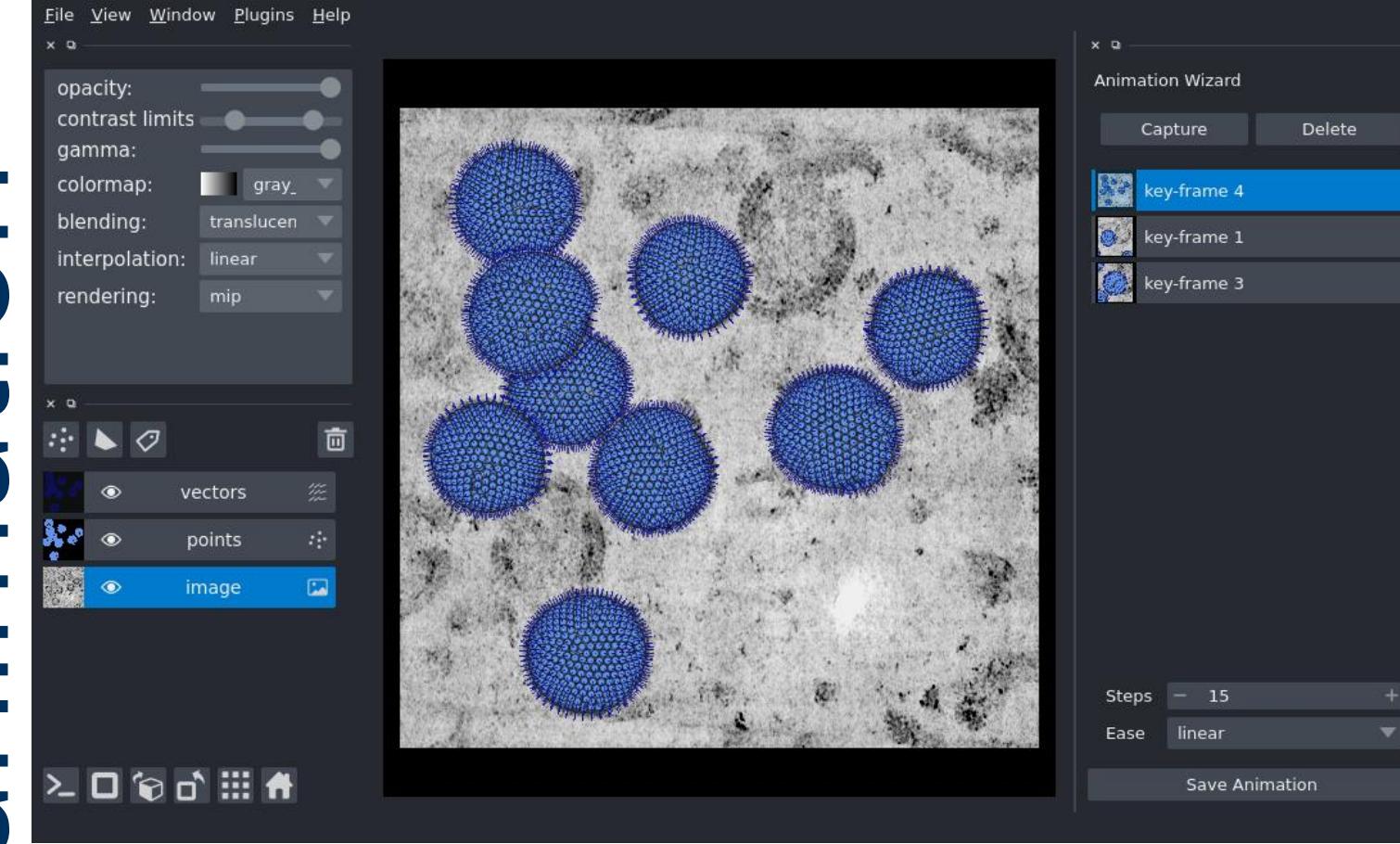
<https://napari.org/tutorials/fundamentals/viewer.html>

The era of napari plugins has just begun

cellfinder

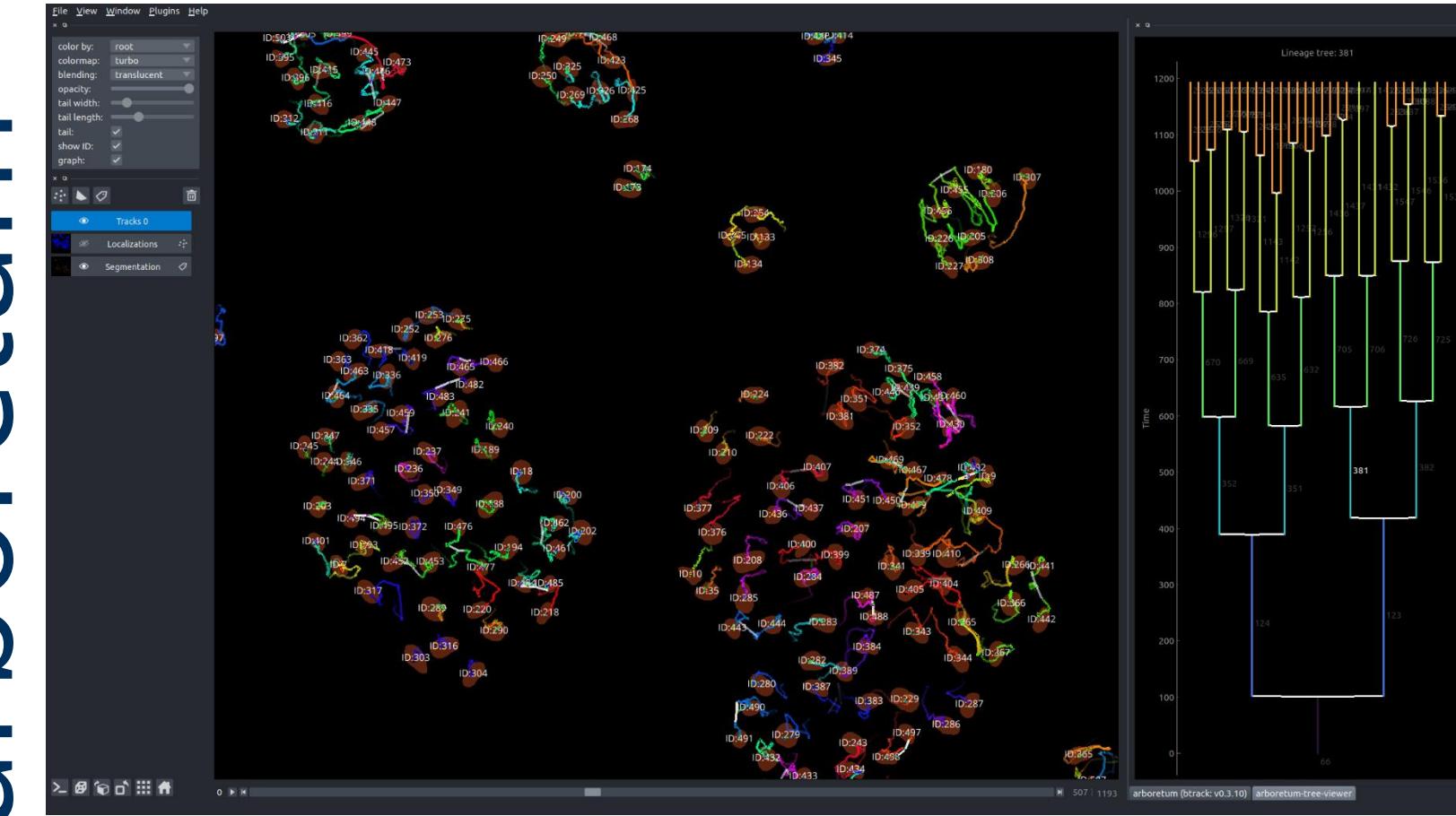


animation



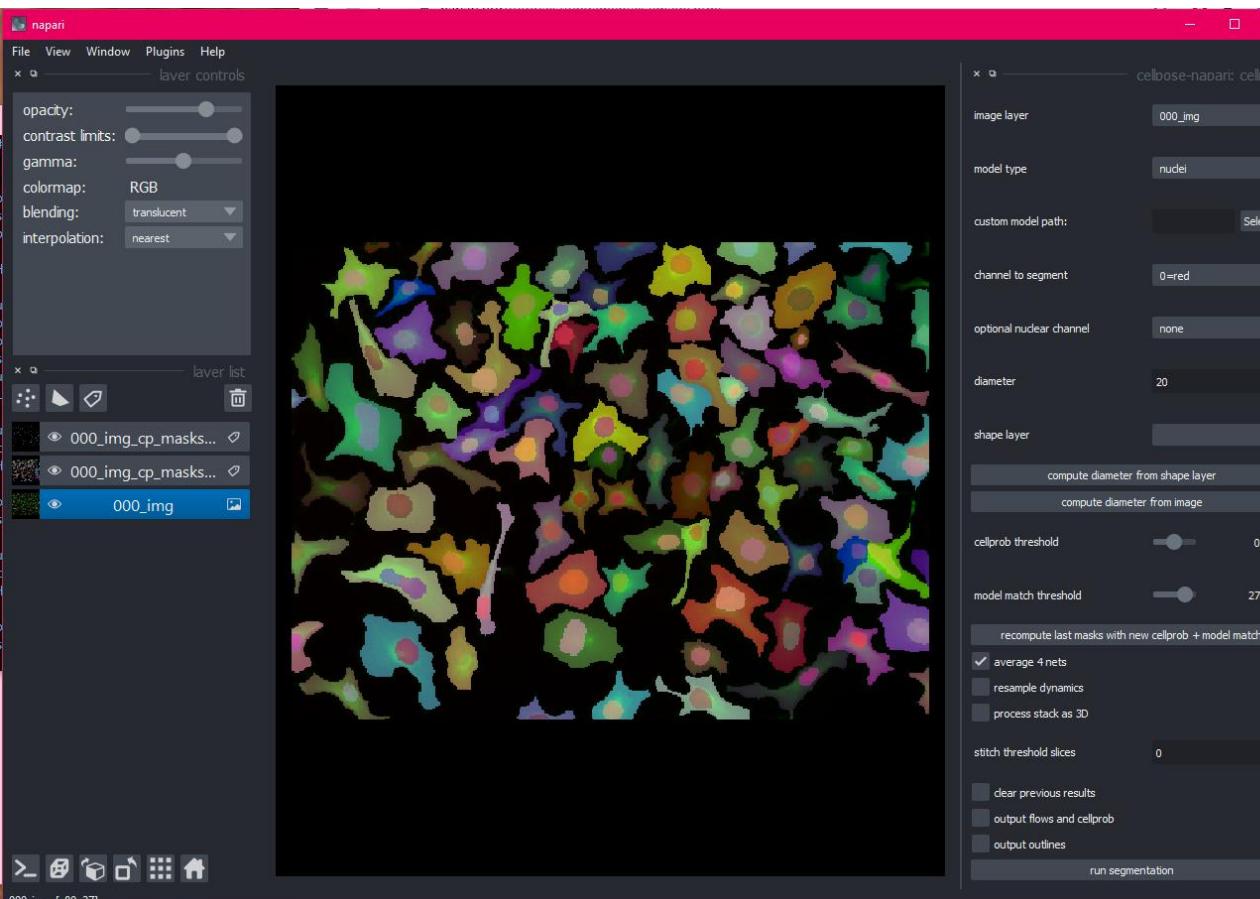
<https://github.com/napari/napari-animation>

arboretum



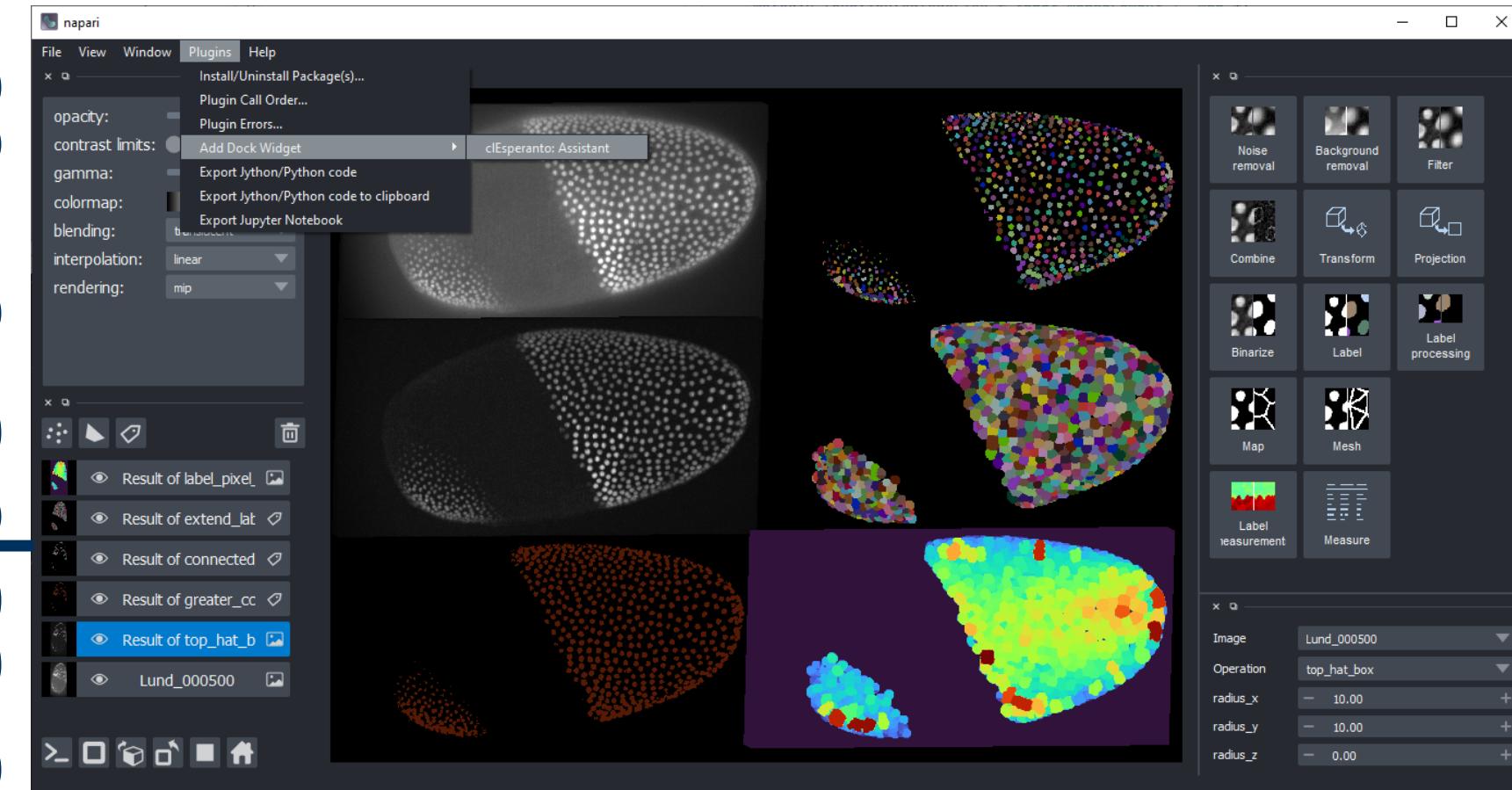
<https://github.com/quantumjot/arboretum>

cellpose



<https://cellpose-napari.readthedocs.io/en/latest/>

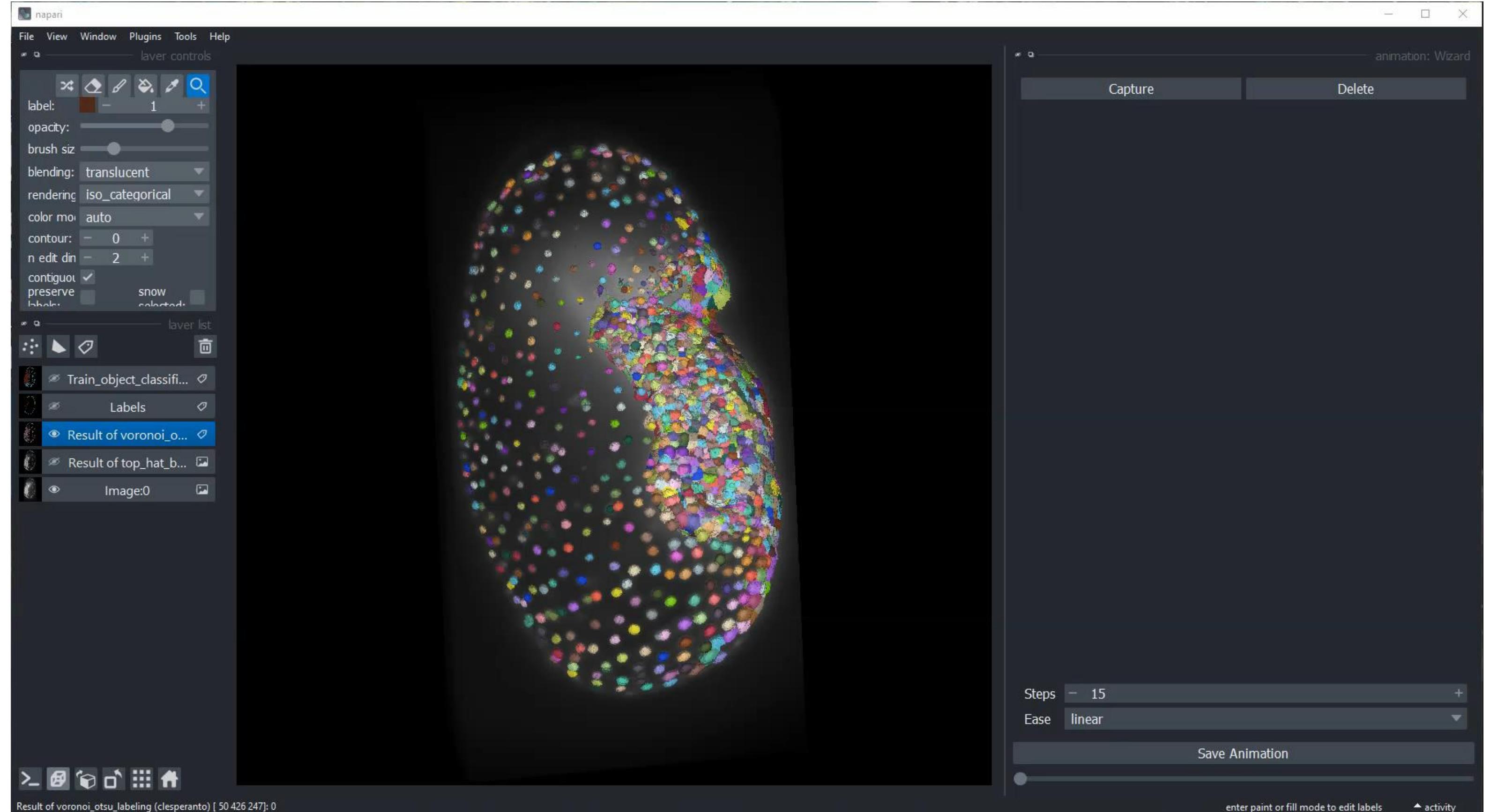
clesperanto



https://github.com/cI Esperanto/napari_pyclesperanto_assistant

Napari-animation

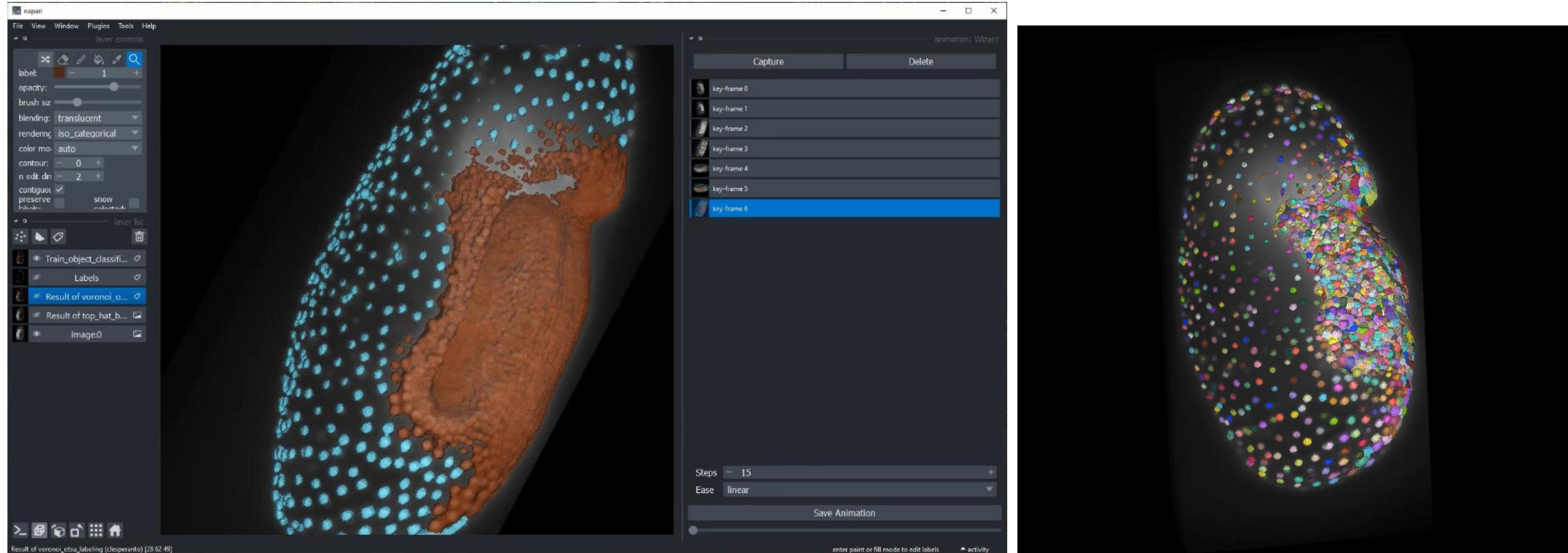
Making animations – as easy as it gets



Work by Alister Burt, Nicholas Sofroniew, et al (BSD3 licensed)
<https://github.com/napari/napari-animation>

Napari-animation

Making animations – as easy as it gets



Work by Alister Burt, Nicholas Sofroniew, et al
<https://github.com/napari/napari-animation>

Napari, segment blobs and things with membranes!

- Filtering,
- thresholding,
- spot detection,
- seeded watershed segmentation,
- Voronoi-Otsu-labeling

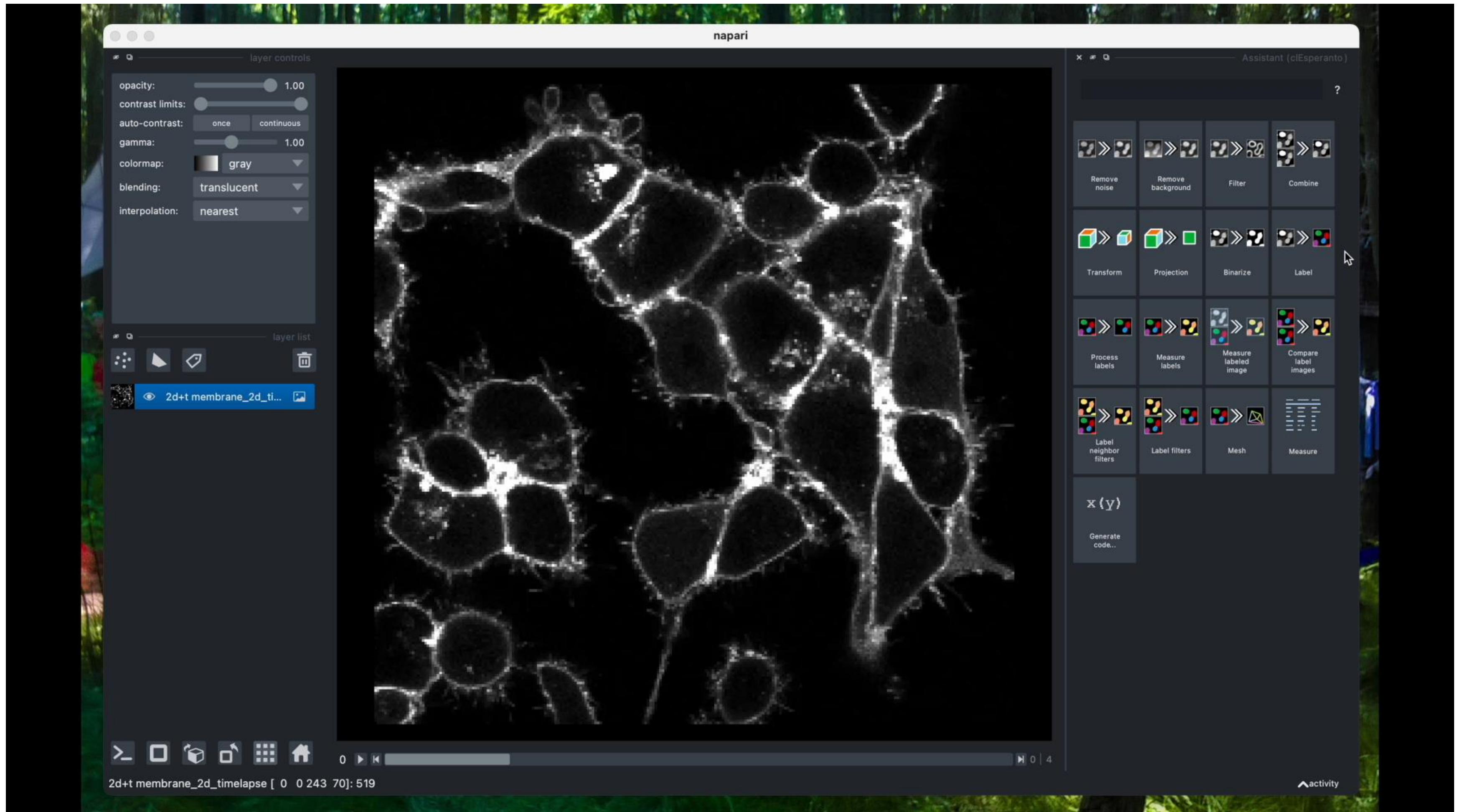
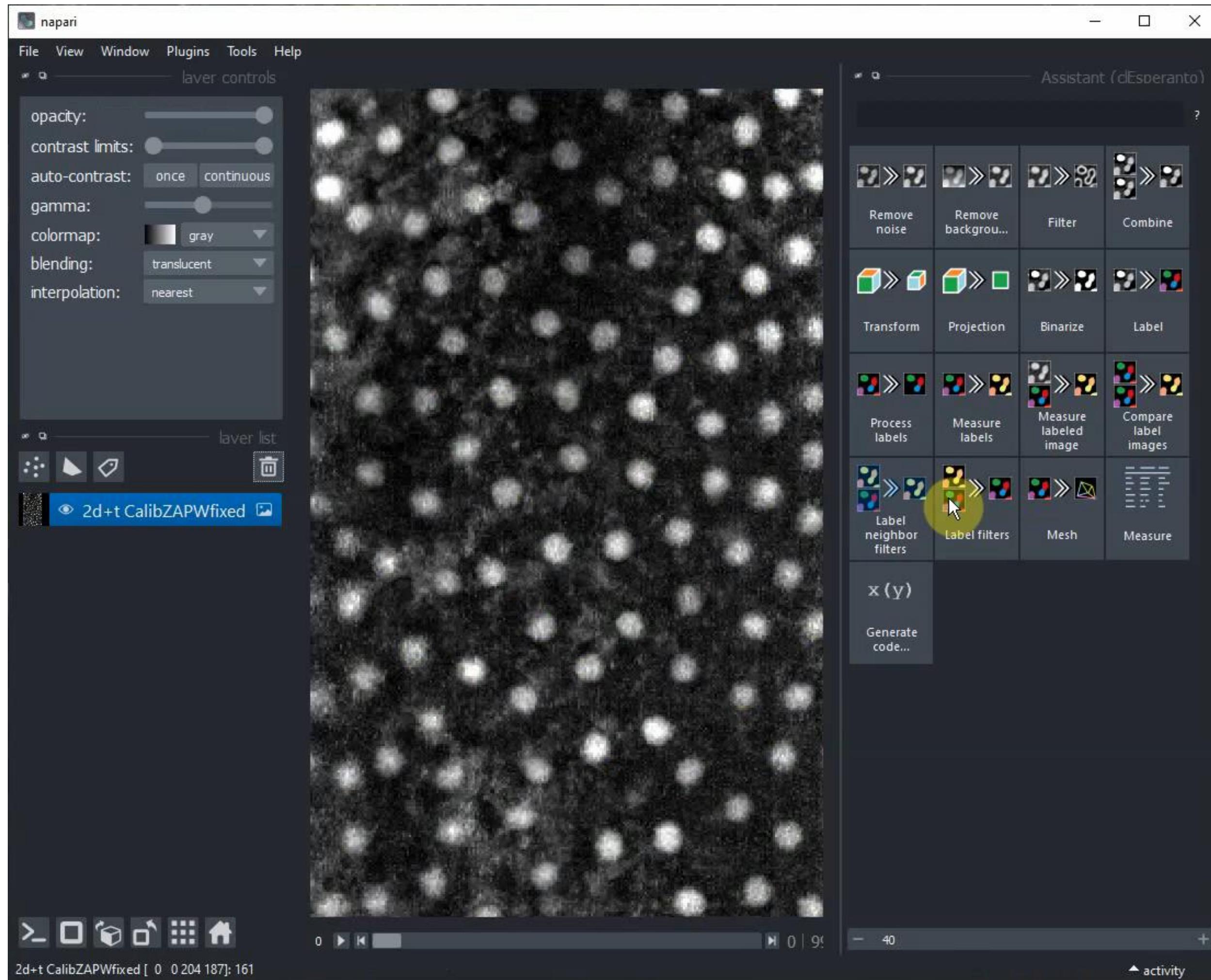


Image data source: Sascha Kuhn,
Nadler lab, MPI-CBG

<https://www.napari-hub.org/plugins/napari-segment-blobs-and-things-with-membranes>

Napari + pyclesperanto + assistant

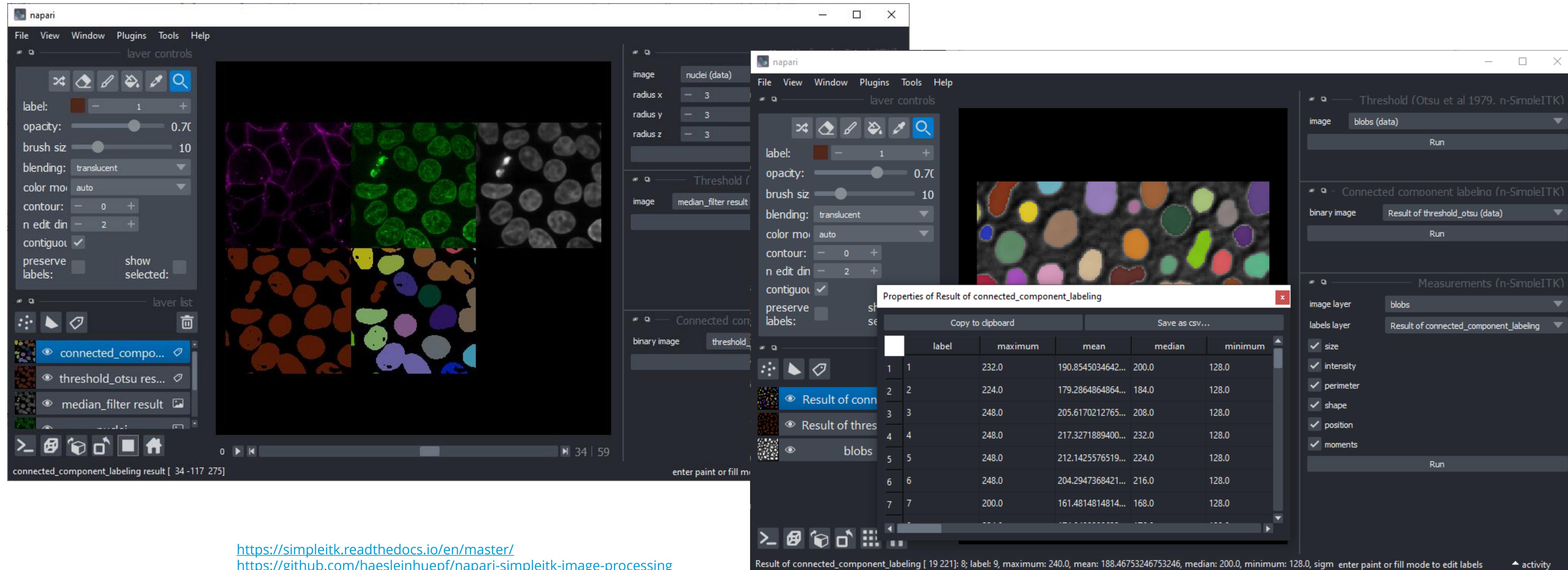
GPU-accelerated image processing
with a pocket-calculator like graphical user
interface



https://github.com/clEsperanto/napari_pyclesperanto_assistant

napari-simpleitk-image-processing

Recommended for 3D-measurements, based on the SimpleITK-project

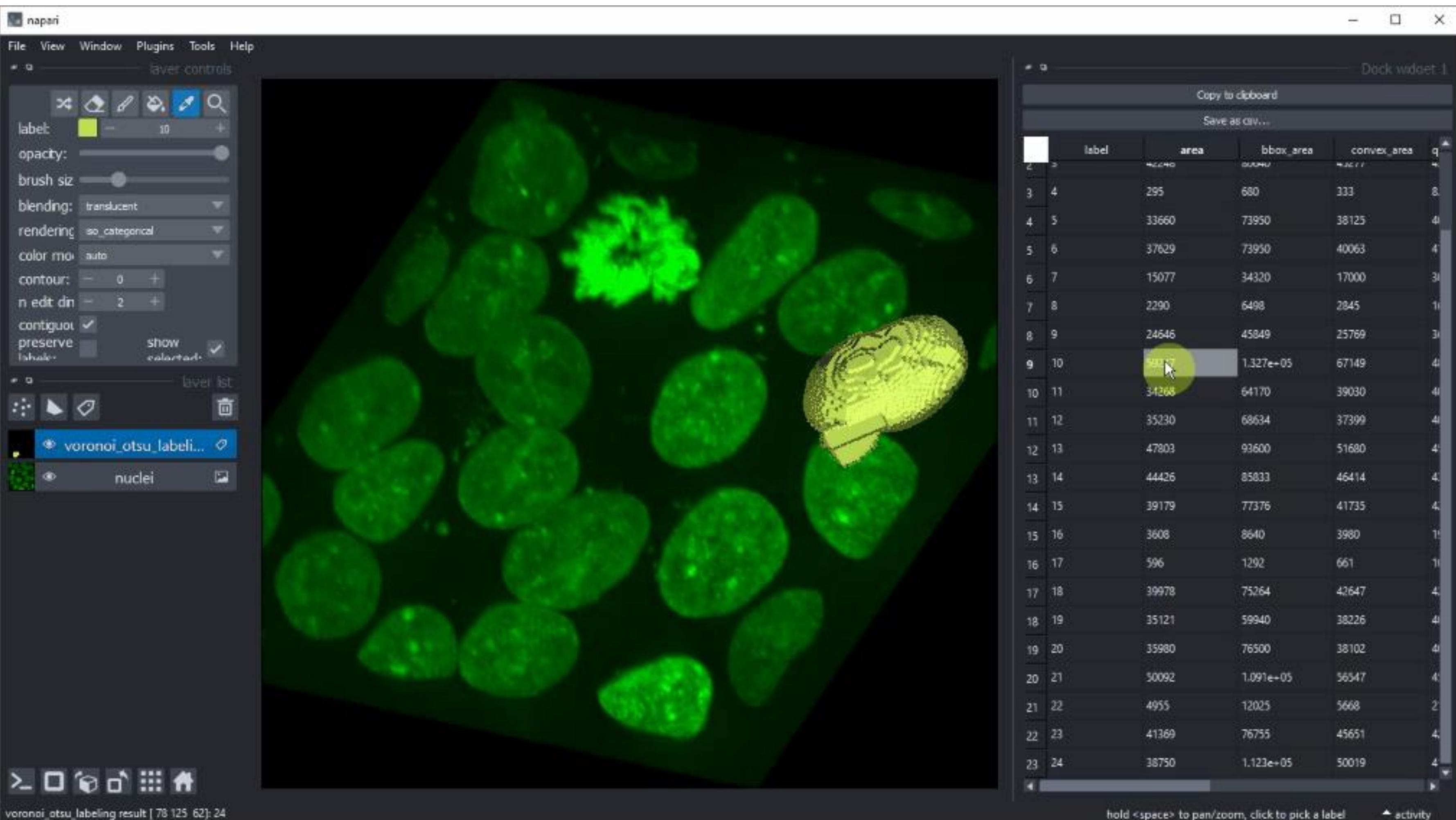


<https://simpleitk.readthedocs.io/en/master/>

<https://github.com/haesleinhuepf/napari-simpleitk-image-processing>

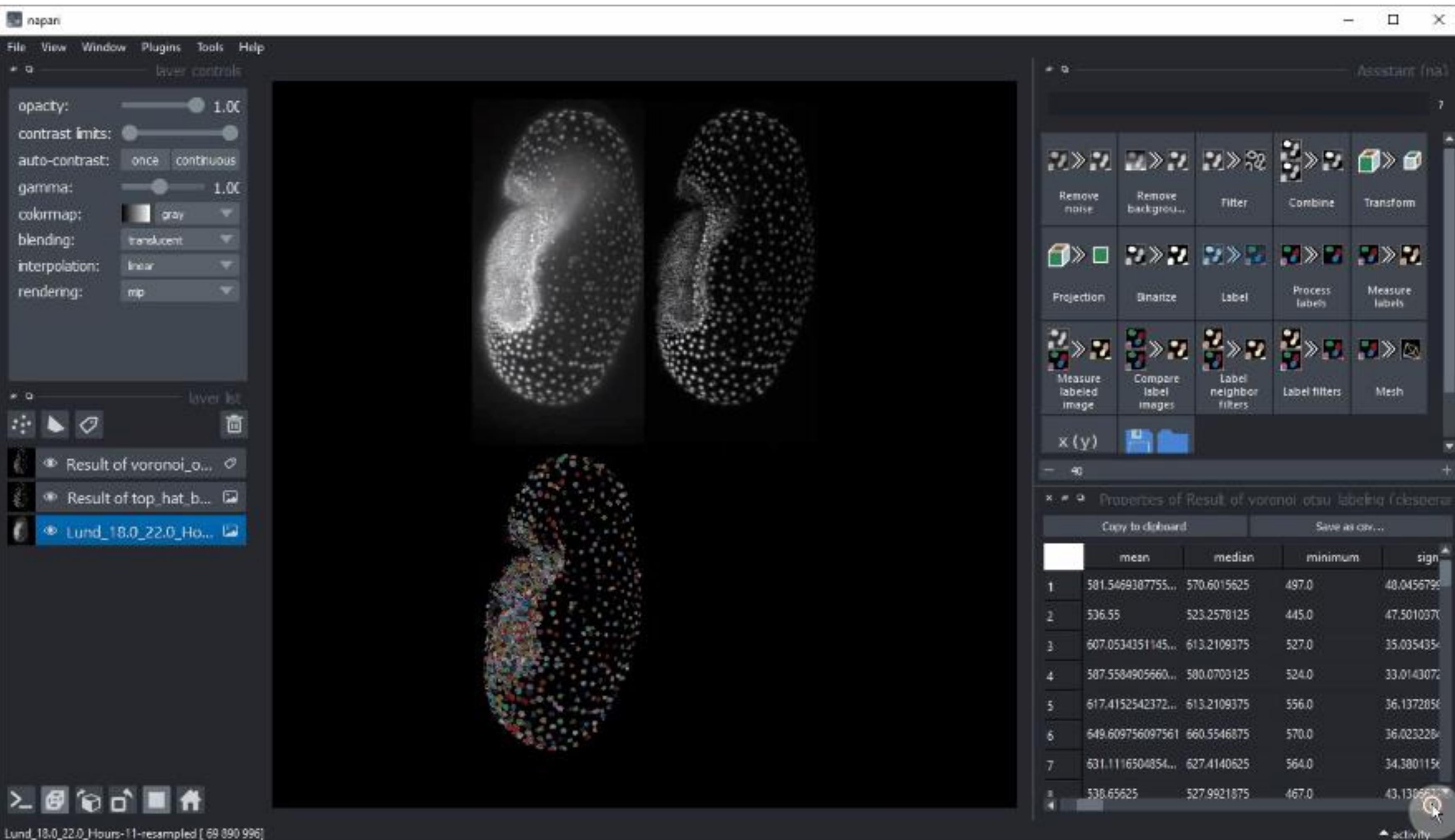
Data exploration

- Click on a cell to view the object the measurement belongs to



Data exploration

- Double-click on a column of measurements to view a parametric image



<https://github.com/haesleinhuepf/napari-skimage-regionprops>

Data exploration

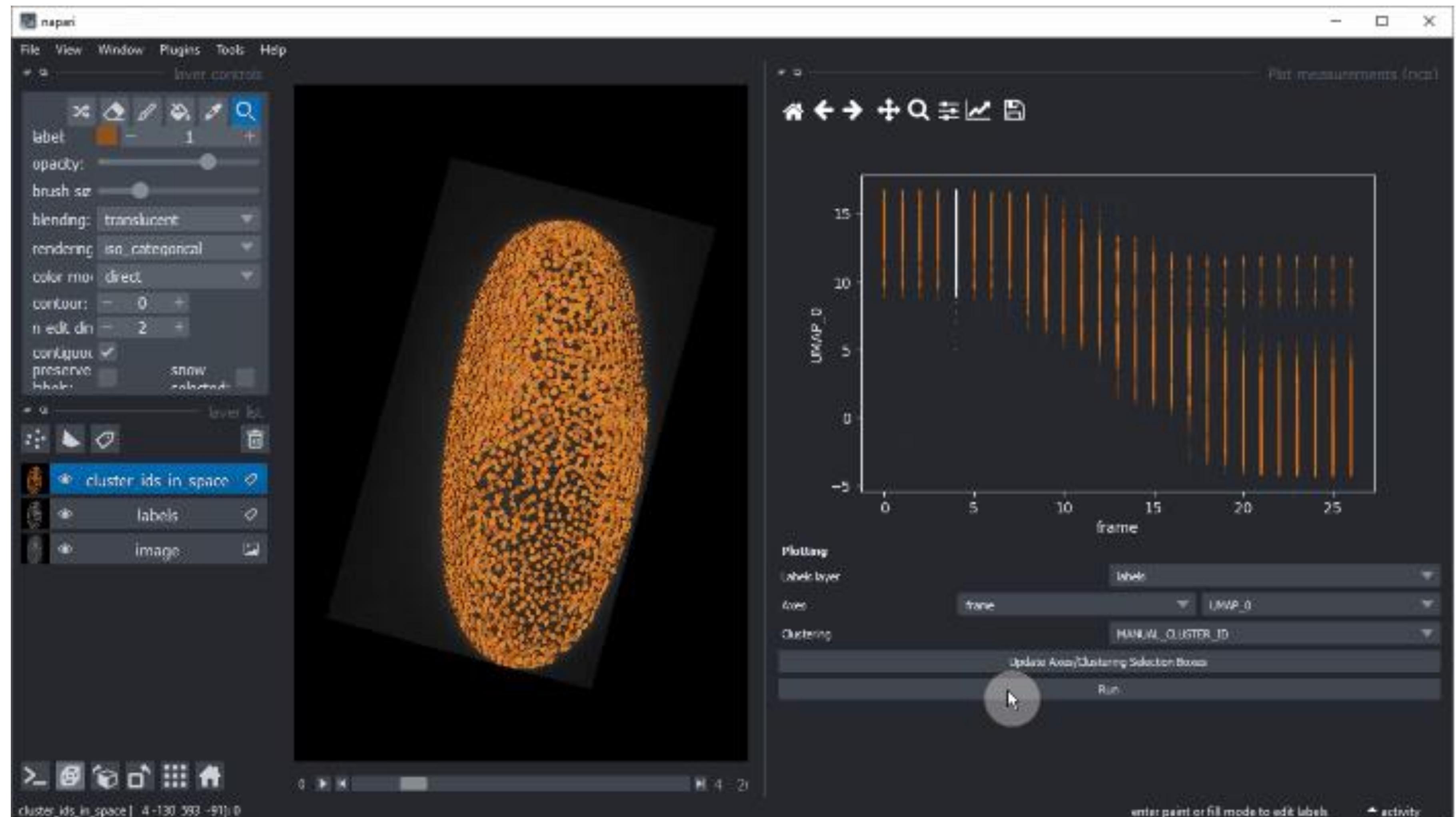
- Manual clustering to gain deeper insights in relationships between measured parameters



Laura Žigutytė
@zigutyte

Ryan Savill
@RyanSavill4

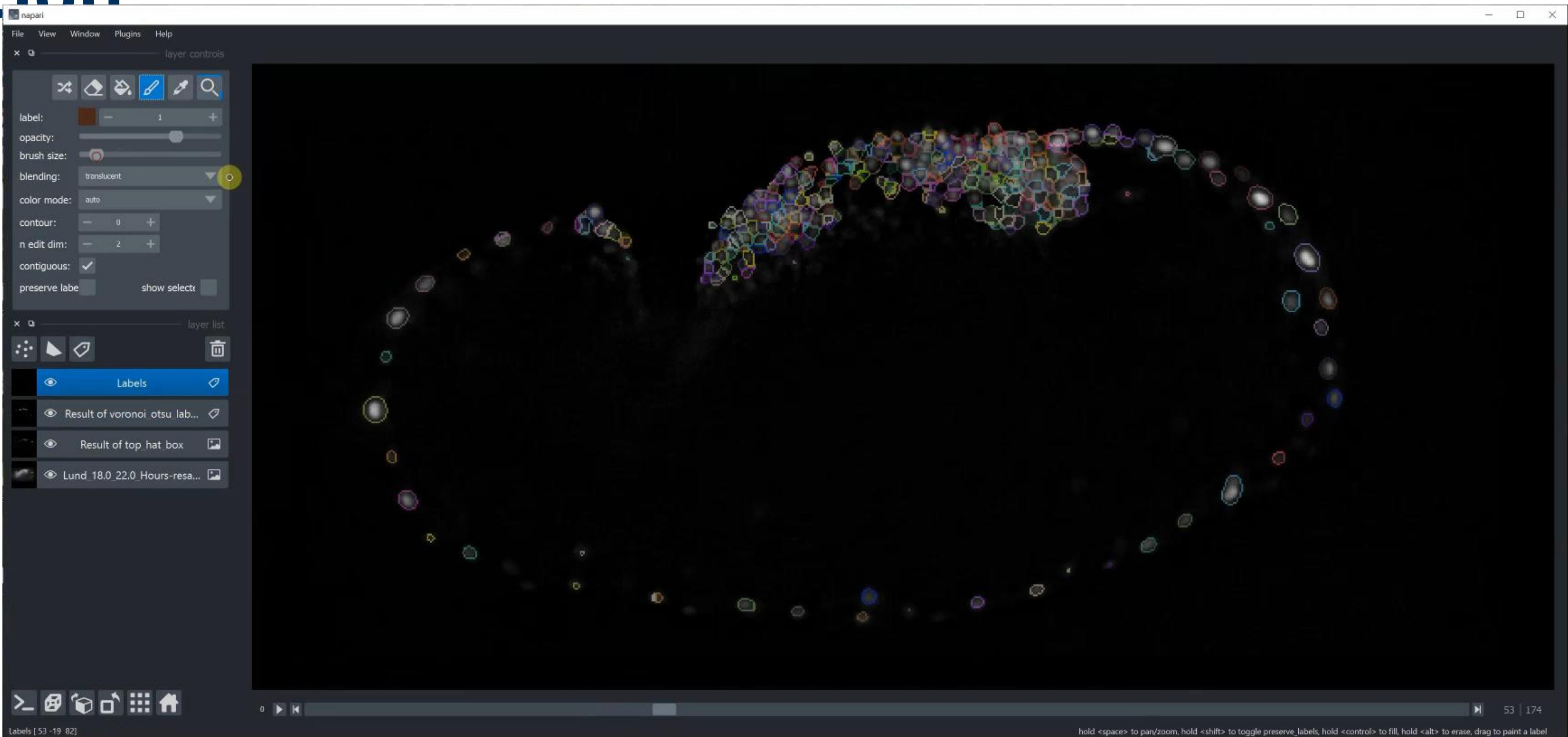
Marcelo Zoccoler
@zoccolermarcelo



Supervised machine learning for tissue classification

GPU-accelerated
Random Forest
Classifiers based on

- scikit-learn and
- clesperanto



<https://github.com/haesleinhuepf/napari-accelerated-pixel-and-object-classification>

The Napari Hub

Search engine for napari plugins

The Napari Hub homepage features a large blue header with the text "Discover, install, and share napari plugins". Below the header is a circular icon with a puzzle piece in the center, surrounded by four user icons. A list of bullet points includes "Discover plugins that solve your image analysis challenges", "Learn how to install into napari", and "Share your image analysis tools with napari's growing community". The main search bar is labeled "Search for a plugin by keyword or author" and has a "Plugin name" filter selected. The footer contains links for "napari hub", "About", and "FAQ".

The search results page shows a search bar with the query "cell segmentation". The results are sorted by relevance, displaying 26 plugins. The first result is "napari-cellseg3d", described as a plugin for cell segmentation by Cyril Achard, Maxime Vidal, Jessy Lauer, and Mackenzie Mathis. The second result is "vollseg-napari", described as a tool for irregular cell shape segmentation using VollSeg by Varun Kapoor.

Search for a plugin by keyword or author
cell segmentation

SORT
 Relevance
 Plugin name
 Recently updated
 Newest

Browse plugins: 26

napari-cellseg3d
plugin for cell segmentation
Cyril Achard, Maxime Vidal, Jessy Lauer, Mackenzie Mathis
...3D: a napari plug-in for direct 3D cell segmentation with deep learning. A napari plug...

vollseg-napari
Irregular cell shape segmentation using VollSeg
Varun Kapoor
...llSeg, a deep learning based 2D and 3D segmentation tool for irregular shaped cells. VollSe...

SORT
 Relevance
 Plugin name
 Recently updated
 Newest

Browse plugins: 8

napari-features
version 0.1.4
release date 24 August 2021
license MIT
python version >=3.7
operating system All
extensible, general-purpose feature extraction

Allen Goodman
... An extensible, general-purpose feature extraction plug-in for the Napari image viewer. Fe...

napari-blob-detection
version 0.0.2
release date 22 April 2022
license BSD-3-Clause
python version >=3.8
operating system All
Detects blobs in images

Andy Sweet, Chi-Li Chiu
...s to labels, users can leverage feature extraction functions that are available to labels...

napari-skimage-regionprops
version 0.5.3
release date 10 July 2022
license BSD-3-Clause
python version >=3.8
A regionprops table widget

Exercise: Install devbio-napari

A collection of napari plugins for developmental biologists

The screenshot shows the GitHub README page for the devbio-napari repository. It features a dark-themed interface with a sidebar containing links to various sections like 'File input/output plugins', 'Image exploration and visualization tools', 'Image processing', and 'Image segmentation plugins'. At the bottom, there's a list of contributing authors.

This screenshot shows the 'Installation' section of the README. It provides instructions for installing via conda and pip, along with specific command examples. It also includes instructions for Mac users and a note about running the command line afterwards.

The screenshot shows the 'devbio-napari' graphical user interface (Assistant window). It displays a dark-themed window with a central image viewer and a sidebar with various processing tools. A message at the bottom indicates that the window has opened successfully.



Marcelo Leomil Zoccoler

With material from:
Robert Haase

“Classic” software vs. notebooks

The image shows a desktop environment with several windows open:

- A terminal window titled "gw18g940 — bash — 72x21" containing Python code for detecting nuclei in an image.
- A file browser window showing files related to the segmentation process.
- A scatter plot titled "Figure 1" showing Mean intensity [AU] versus Area [px].
- A scatter plot showing segmented nuclei in various colors on a black background.

Code in the terminal window:

```
course_functions.py
course_functions.py
1 import skimage.morphology
2 import skimage.filters
3 import numpy as np
4 from skimage.measure import label, regionprops_table
5 import matplotlib
6
7 def detect_nuclei(image, si
8
9     # filtering
10    image = skimage.filters
11    # local thresholding
12    image_local_threshold =
13    image_local = image > i
14    # remove tiny features
15    image_local_open = skim
16    # label image
17    image_labeled = label(i
18    # analyze regions
19    our_regions = regionpro
20    # create a new mask with
21    newimage = np.zeros(im
22    # fill in using region
23    for x in range(len(our_
24        if our_regions['are
25        newimage[our_re
26
27    return newimage
28
```

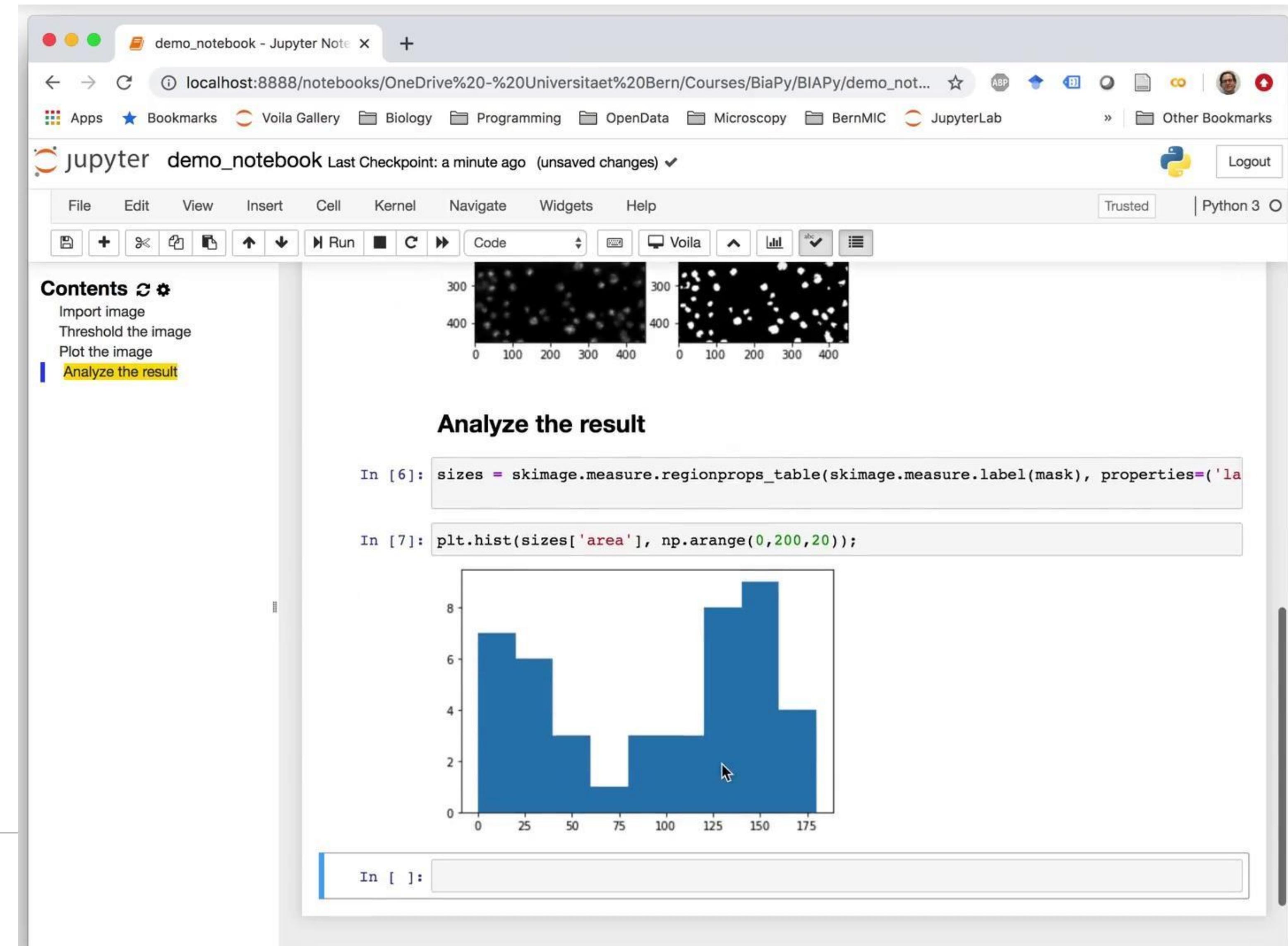
“Classic” software vs. notebooks

Code divided in parts

Dynamic: easy to test

Rich output: processing + visualisation + analysis in one place

Code + formatted text: easy documentation



What is a jupyter notebook?

A text file (easily sent around)

Rendered by Jupyter in the browser

Split into sections called cells

The screenshot shows a Jupyter Notebook interface in a web browser. The title bar says "demo_notebook - Jupyter Note". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Trusted, and Python 3. The toolbar has icons for New, Run, Cell, Code, Voila, and other utilities.

The notebook contains the following code cells:

- Import image**
In [1]:

```
import numpy as np
import skimage
import skimage.io
import skimage.measure
import matplotlib.pyplot as plt
```
- Threshold the image**
In [2]:

```
image = skimage.io.imread('../Data/BBBC007_v1_images/A9/A9_p10d.tif')
```
- Define a threshold value:**
In [8]:

```
threshold = 100
```
- Plot the image**
In [9]:

```
mask = image > threshold
```
- In [10]:

```
fig, ax = plt.subplots(1,2)
ax[0].imshow(image, cmap = 'gray')
ax[1].imshow(mask, cmap = 'gray')
```

Out[10]: `<matplotlib.image.AxesImage at 0x1202af510>`

Below the code, two grayscale plots are shown side-by-side. The left plot shows a noisy image with a grid of values from 0 to 400 on both axes. The right plot shows a binary mask where the noise is thresholded, with values 0 and 1.

What is a jupyter notebook?

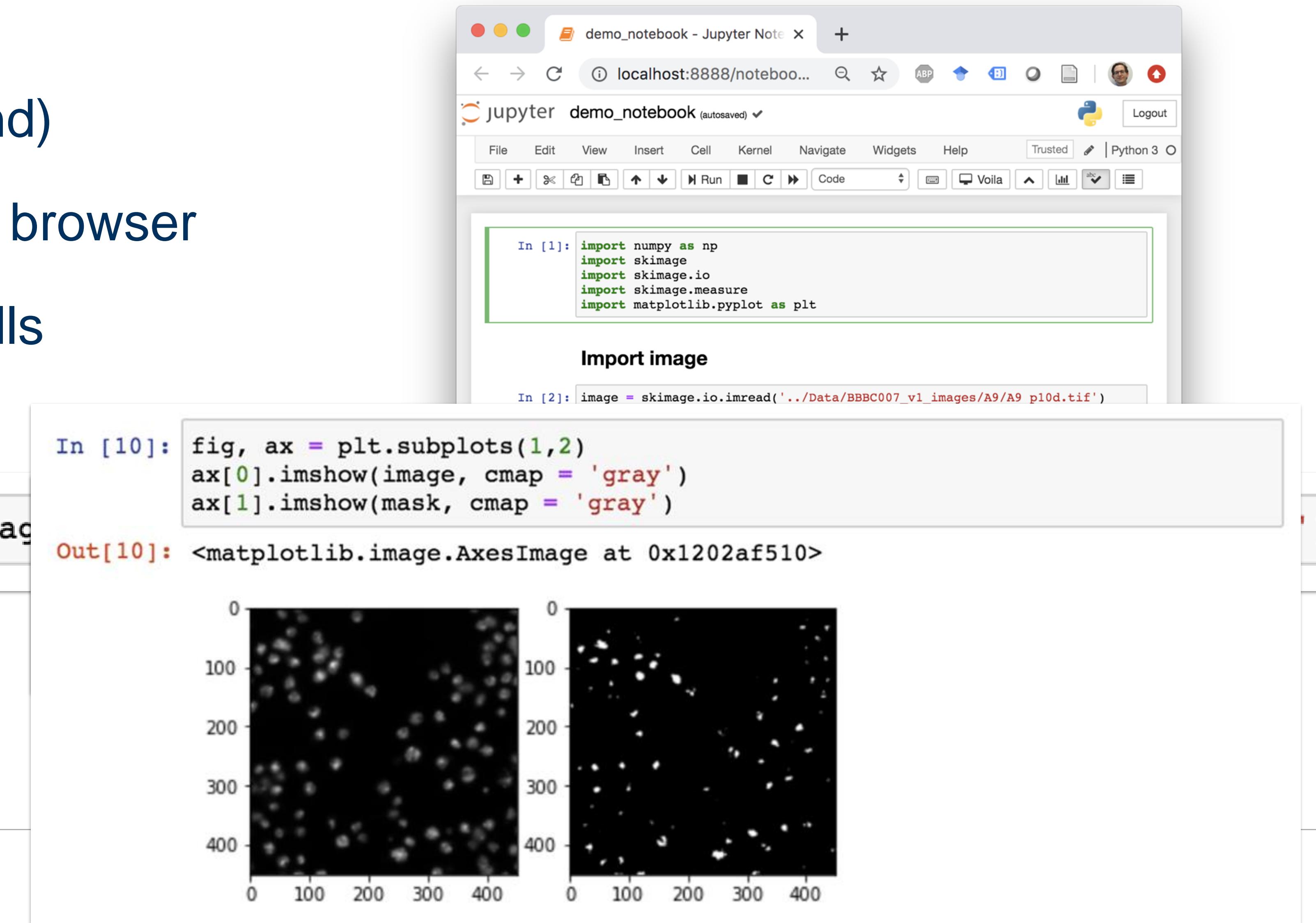
A text file (easily sent around)

Rendered by Jupyter in the browser

Split into sections called cells

Cells can contain:

- Code
- Formatted text
- Rich output



The screenshot shows a Jupyter Notebook interface running in a web browser. The title bar says "demo_notebook - Jupyter Note". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, and Trusted Python 3. The toolbar has icons for New, Run, Cell, Code, Voila, and other utilities.

The notebook contains two cells:

- In [1]:**

```
import numpy as np
import skimage
import skimage.io
import skimage.measure
import matplotlib.pyplot as plt
```

Import image
- In [2]:**

```
image = skimage.io.imread('../Data/BBBC007_v1_images/A9/A9_p10d.tif')
```

Out[2]: <matplotlib.image.AxesImage at 0x1202af510>

Two grayscale images are displayed below the code cell. Both images show a distribution of small, bright, irregular spots on a black background. The axes for both plots range from 0 to 400, with major ticks at 0, 100, 200, 300, and 400.

Why using notebooks?

Documenting (for your-(future)-self and for others) and enhanced reproducibility

“First, we applied a Gaussian filter from scikit-image with sigma = 1. Then, we applied another Gaussian filter to the original image with sigma = 10. After that, we subtracted the first result from the second...

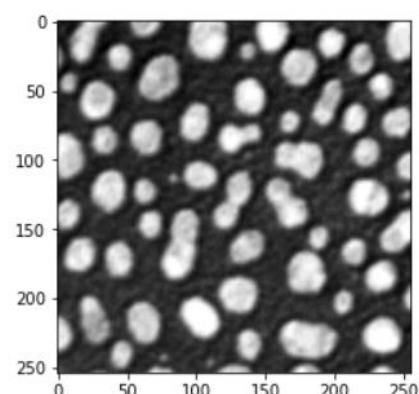
Image Processing Workflow

```
[6]: from skimage.filters import gaussian
from skimage.io import imread
import matplotlib.pyplot as plt

*[5]: image = imread(r'..\data\image.tif')

[7]: plt.imshow(image, cmap = 'gray')

[7]: <matplotlib.image.AxesImage at 0x1900ad27b50>
```



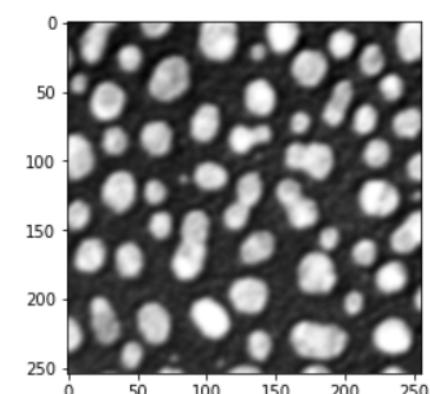
1. Apply a gaussian filter with a small sigma

We used `sigma = 1` in this example.

```
[14]: image_gauss1 = gaussian(image, sigma = 1)

[15]: plt.imshow(image_gauss1, cmap = 'gray')

[15]: <matplotlib.image.AxesImage at 0x1900bf56670>
```



2. Apply a gaussian filter with a large sigma

We used `sigma = 10` in this example.

```
[16]: image_gauss2 = gaussian(image, sigma = 10)

[17]: plt.imshow(image_gauss2, cmap = 'gray')

[17]: <matplotlib.image.AxesImage at 0x1900bfb5b0>
```



Why using notebooks?

Sharing

- Send a single file with code, intermediary outputs and rich text explanations

This screenshot shows a GitHub page for a repository named 'neubias_academy.biipy'. It displays the contents of a file named '02-Numpy_images.ipynb'. The notebook includes a section titled '2. Numpy with images' with explanatory text and code cells. One cell contains the command `import numpy as np`. Another cell shows the output of running this command, which is a grayscale image of a neuron.

GitHub rendering

This screenshot shows an nbviewer rendering of the same Jupyter notebook. The interface is similar to GitHub's, showing the notebook's structure and content. The '2. Numpy with images' section is visible, along with the code cell containing `import numpy as np` and its output.

Nbviewer rendering

This screenshot shows a Jupyter lab session for the same notebook. The interface is more complex, showing multiple panes. One pane displays the '2. Numpy with images' section, while another shows the code cell `import numpy as np` and its output. The bottom pane shows the command line interface.

Jupyter lab (local or Binder)

This screenshot shows a Google Colab session for the notebook. The interface is clean and modern. It displays the '2. Numpy with images' section and the code cell `import numpy as np`. The output pane shows the resulting grayscale image of a neuron.

Google Colab

Why using notebooks?

Keep all the benefits from using code:

- Batch processing
- Running python functions/tools that do not have UI

Why using notebooks with napari?

- Easy data interaction and visualization with napari:
 - Great for visualizing 3D (and more) data
 - Each processing step result can be displayed as a separate layer
- Data annotation

Scripting napari in notebooks

Start napari from a jupyter

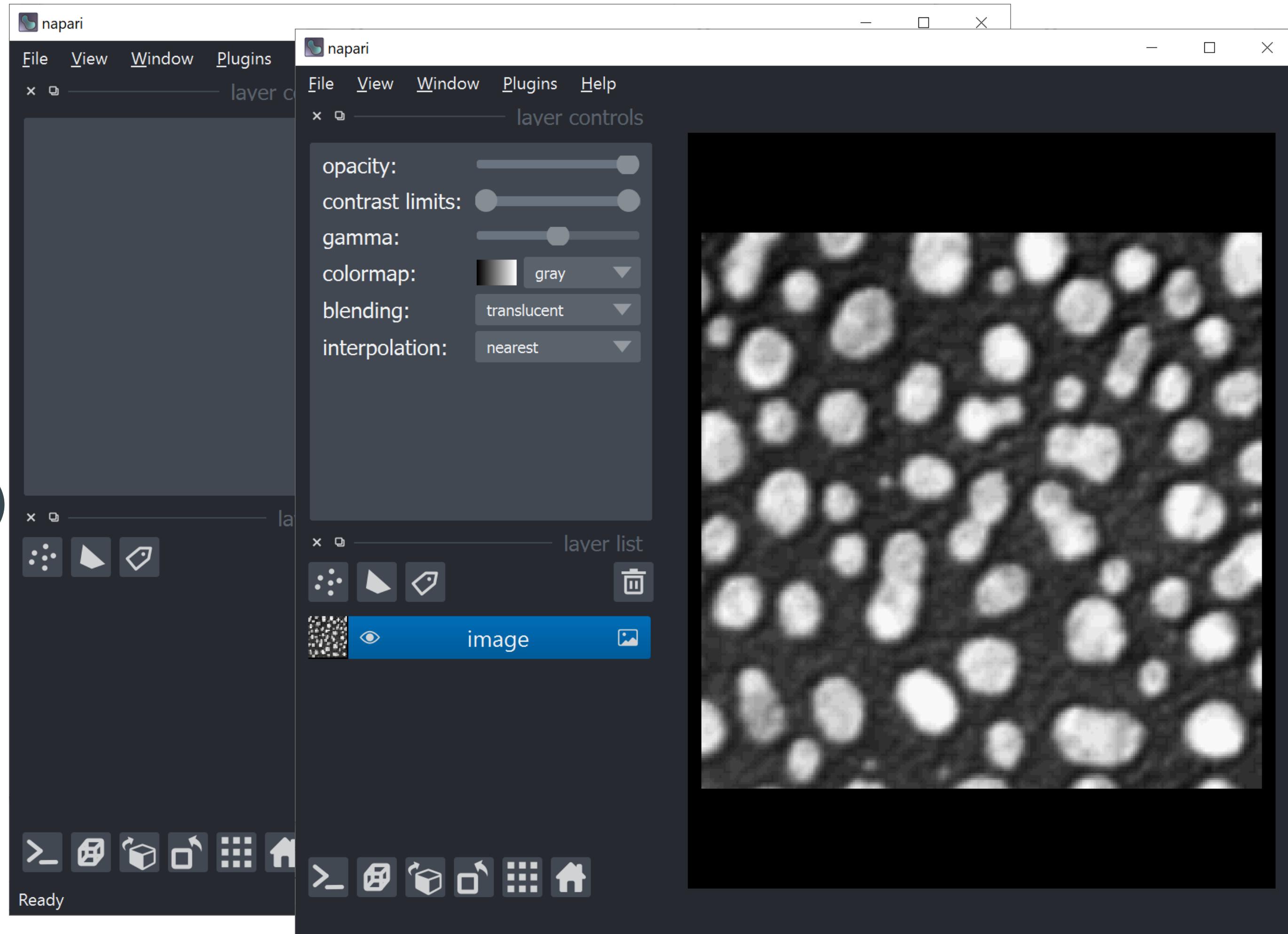
```
import napari  
  
# Create an empty viewer  
viewer = napari.Viewer()
```

```
viewer.add_image(image, name = 'image')
```

Load image from notebook

to napari (and back): flexibility!

```
image = viewer.layers['image'].data
```



Scripting napari in notebooks

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". In the code editor, there are two code cells:

```
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)
```

The output of the second cell is:

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

Below the code cells, a note states: "With this command, we can make a screenshot of napari and save it in our notebook." The next cell shows the result of running the `napari.utils.nbscreenshot` command:

```
In [6]: napari.utils.nbscreenshot(viewer)
```

The output is a screenshot of the napari viewer interface, which includes a control panel on the left with sliders for opacity, contrast limits, gamma, colormap, blending, and interpolation, and a main canvas on the right displaying a grayscale image of cellular structures.

Scripting napari in notebooks

- Add layers to napari to visualize intermediate processing results on top of each other or side by side.
- Change layer visualization within napari...

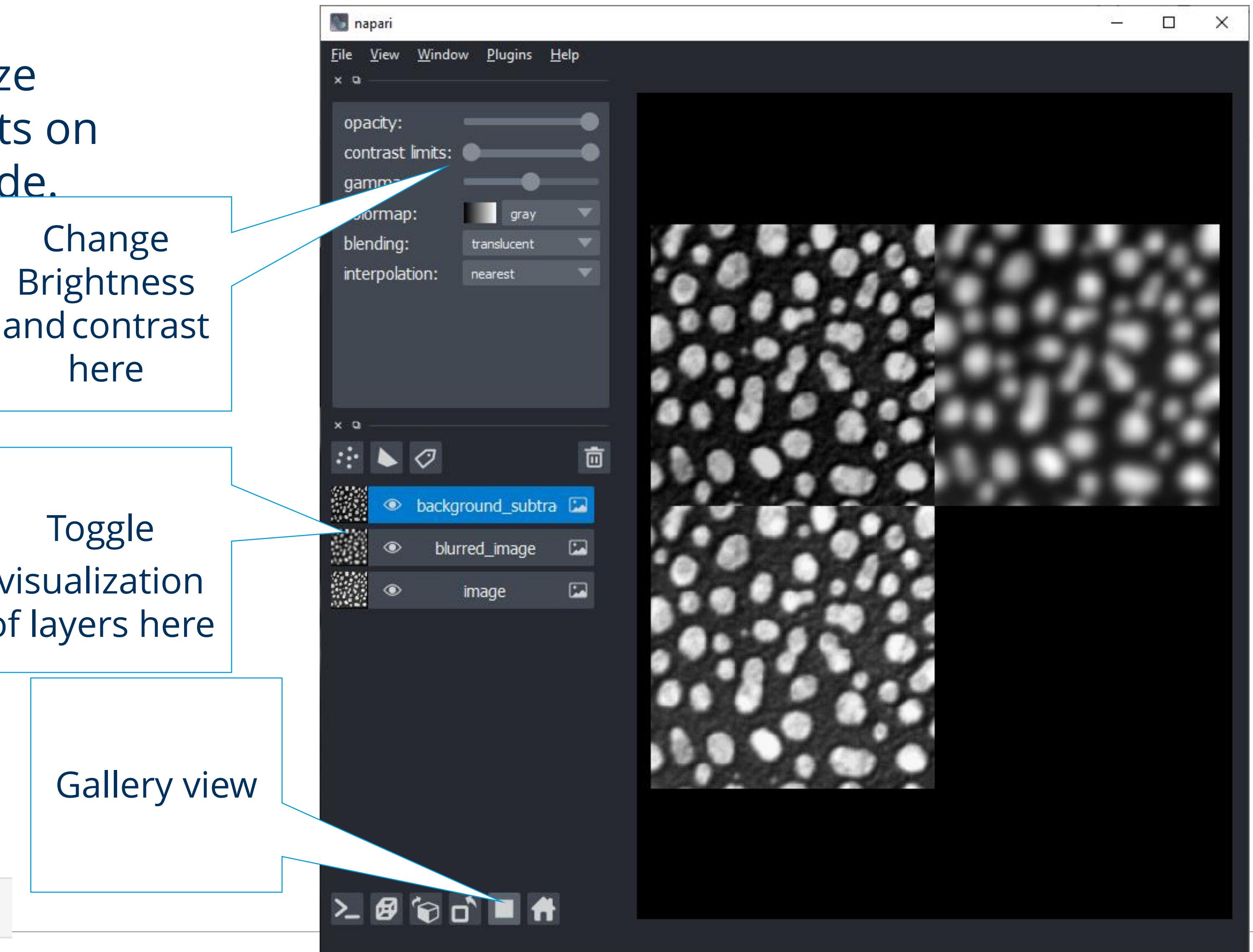
... or via code in a jupyter notebook:

```
viewer.layers[0].contrast_limits
```

[0, 255]

1. Access the viewer
2. Access the layers
3. Choose a layer (by index or name)
4. "Press TAB" and check out available properties

```
viewer.layers[0].contrast_limits = [30, 170]
```

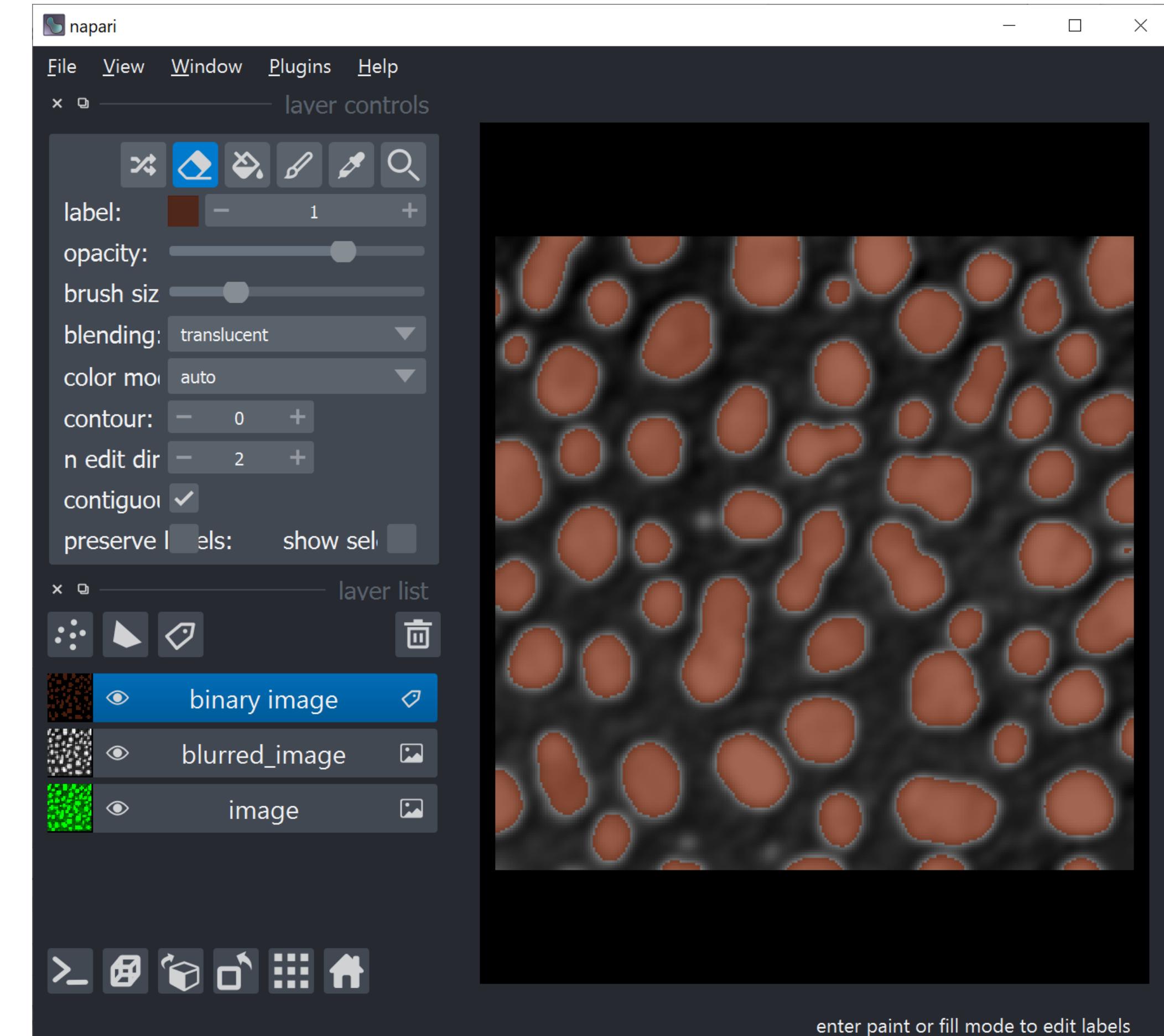


Visualizing image segmentation

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,  
                  name="binary image")
```

Name your layers to keep track of what they contain



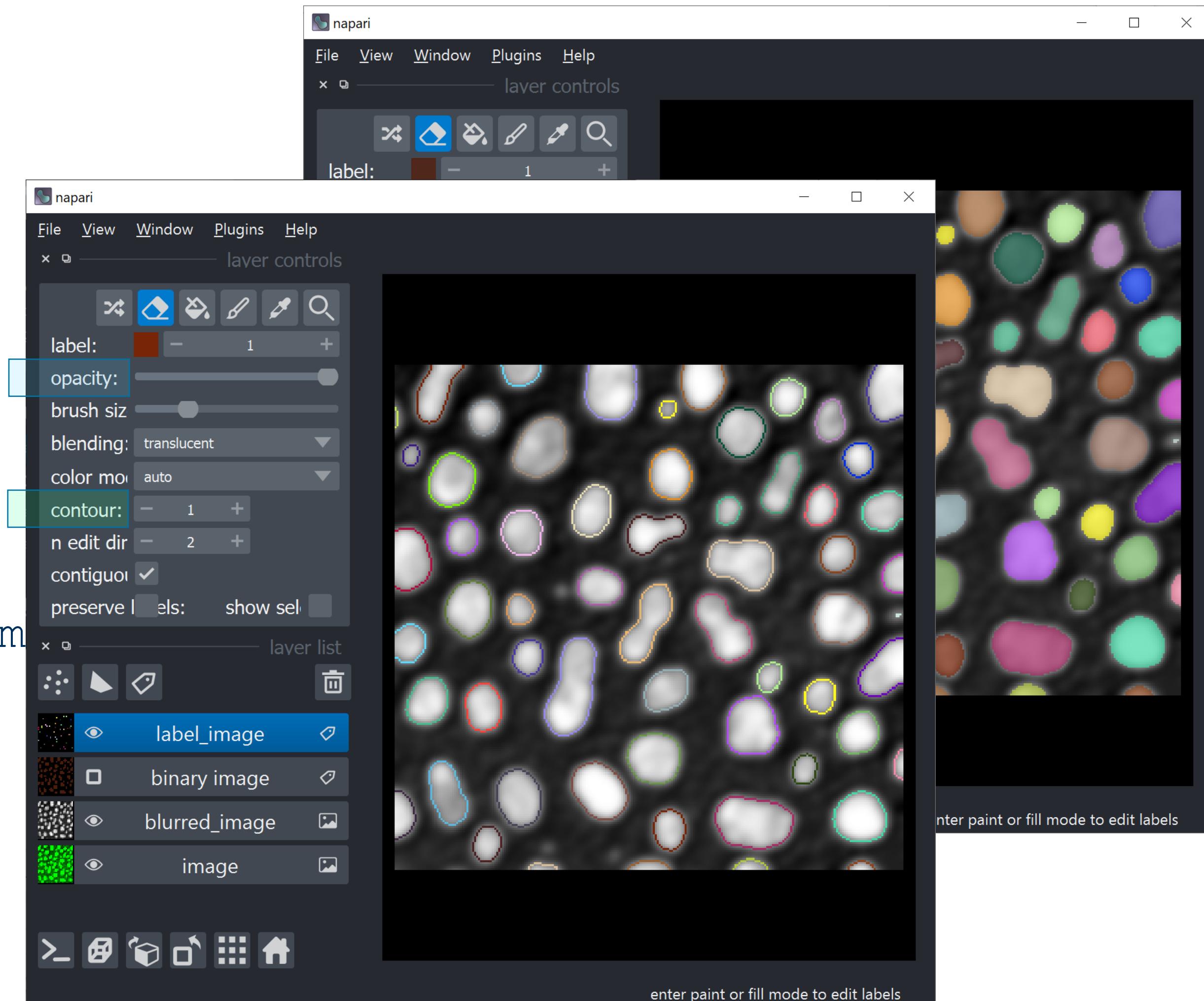
Visualizing image segmentation

Binary images and label images visualized as label layers

```
from skimage.measure import label  
  
label_image = label(binary_image)  
  
# add labels to viewer  
  
label_layer = viewer.add_labels(label_im
```

Visualize contours instead of the overlay

```
label_layer.contour = 1  
  
label_layer.opacity = 1
```

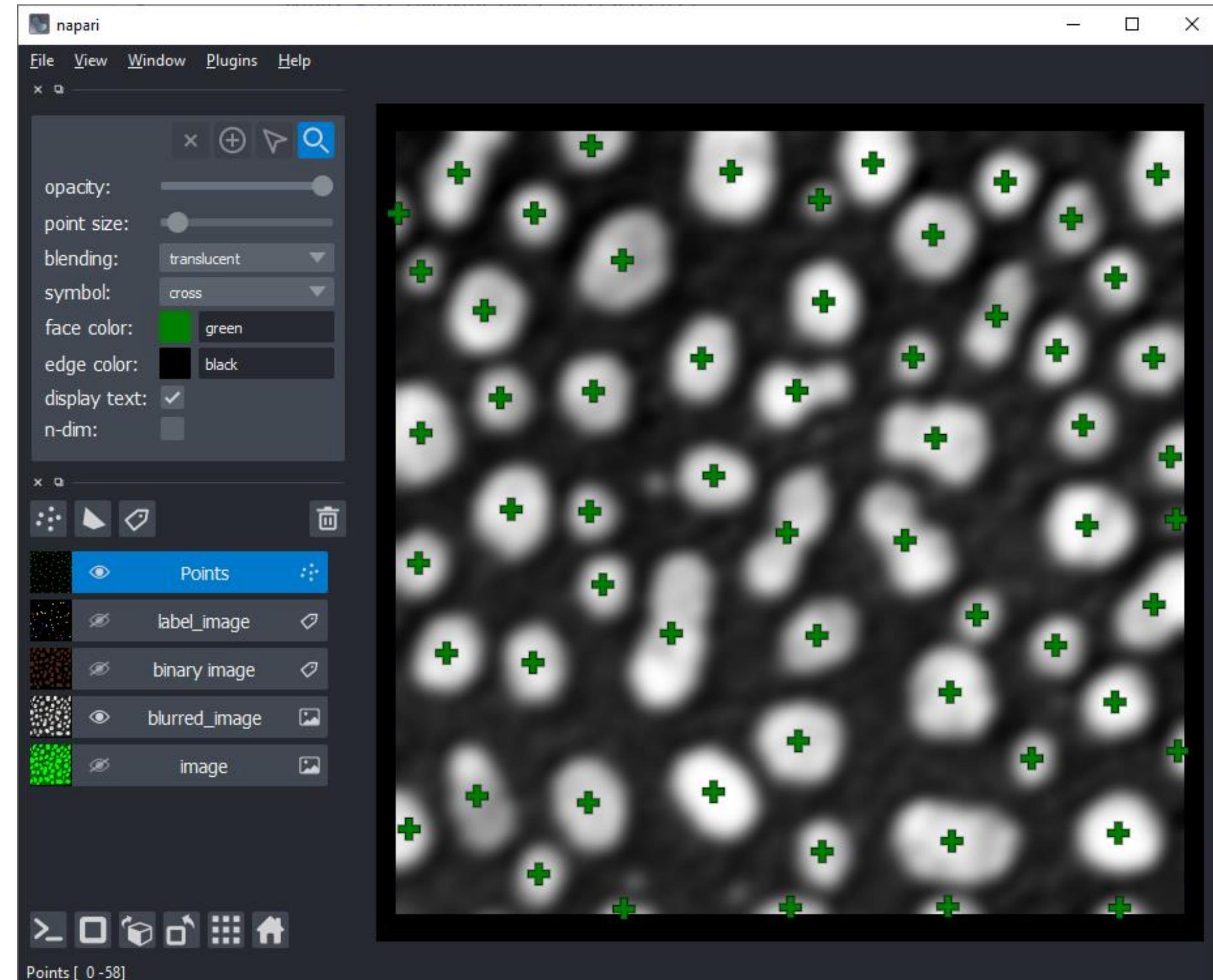


Points layers

There is also other layer types

- Shapes
- Points
- Surfaces
- Tracks
- Vectors

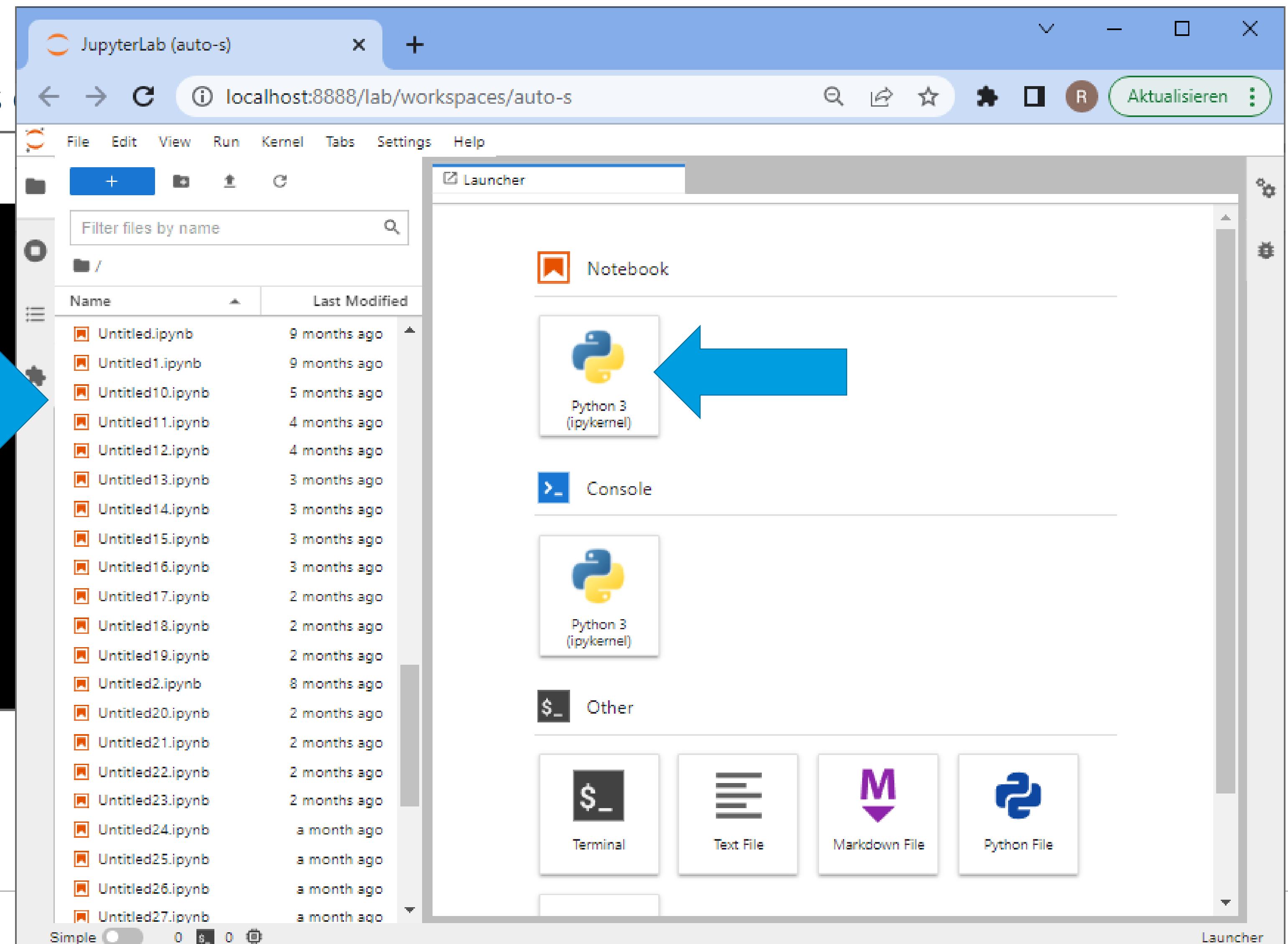
```
from skimage.measure import regionprops  
  
statistics = regionprops(label_image)  
  
points = [s.centroid for s in statistics]  
  
# add points to viewer  
label_layer = viewer.add_points(points, face_color='green', symbol='cross', size=5)
```



Jupyter lab

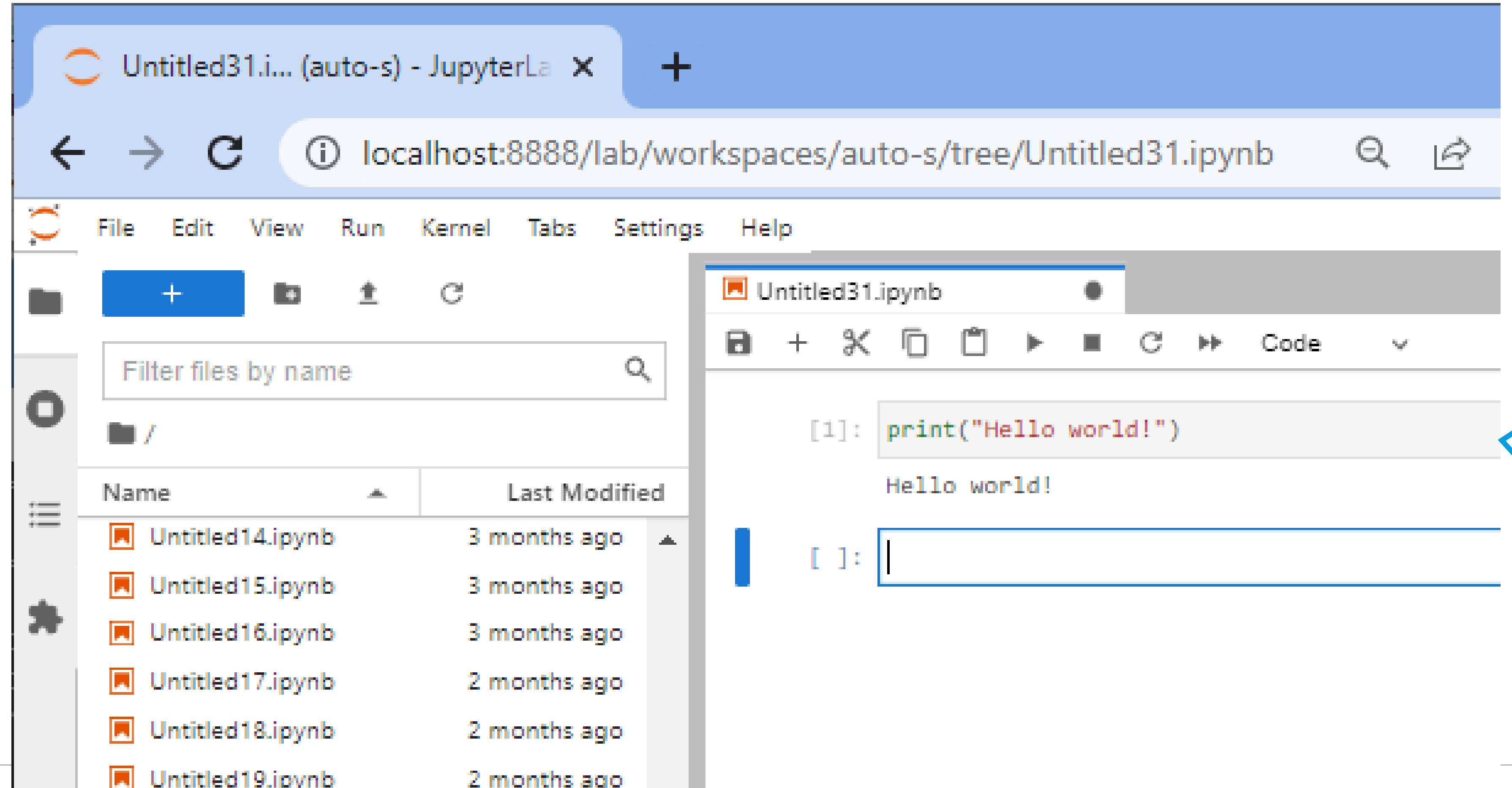
Our programming environment for this

```
Command Prompt - conda deactivate - cond...
c:\Users\rober>conda activate bio_39
(bio_39) c:\Users\rober>jupyter lab
```



Jupyter lab

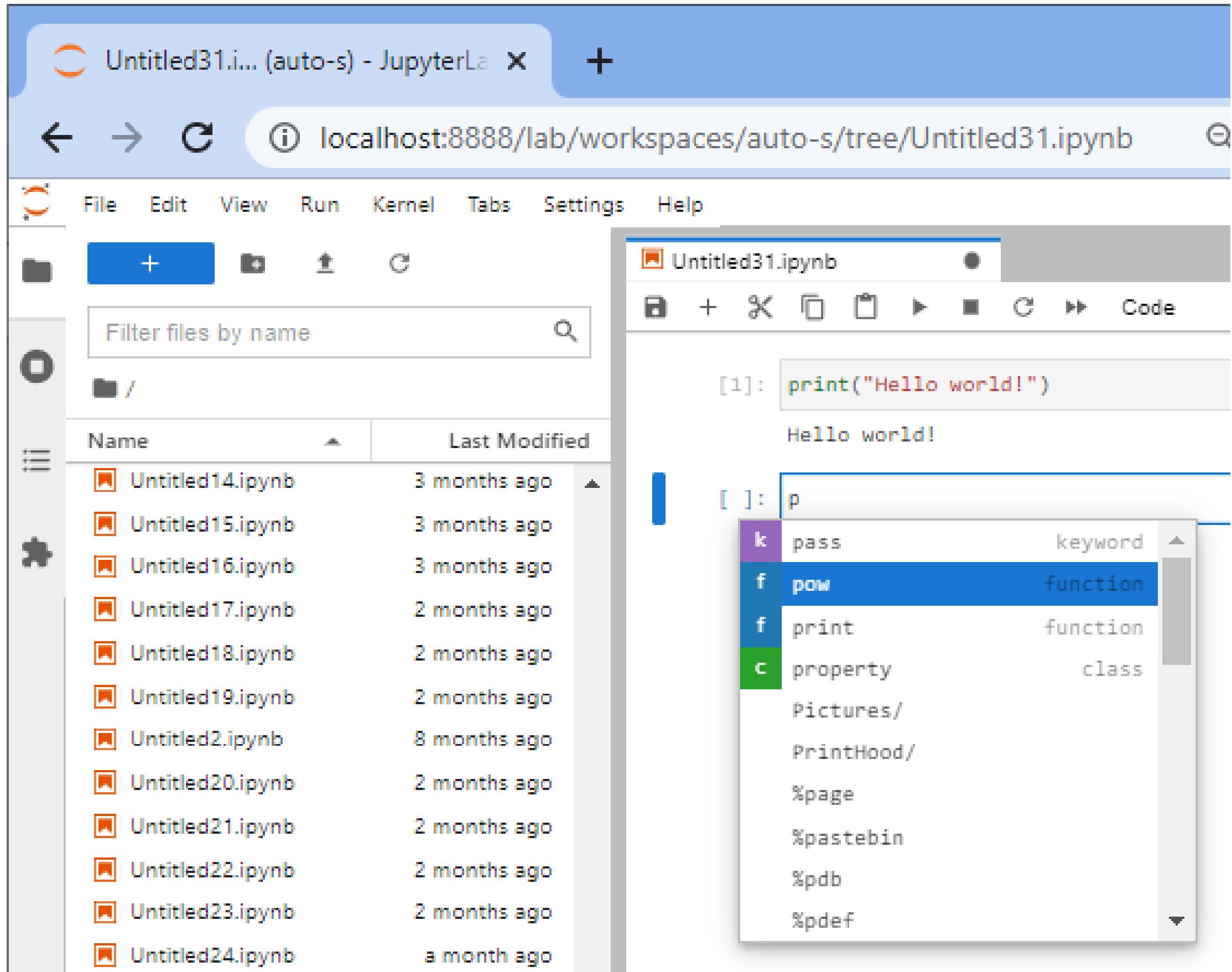
Execute code cell-by-cell and see results instantaneously



SHIFT +
ENTER to
execute a
code cell

Jupyter lab

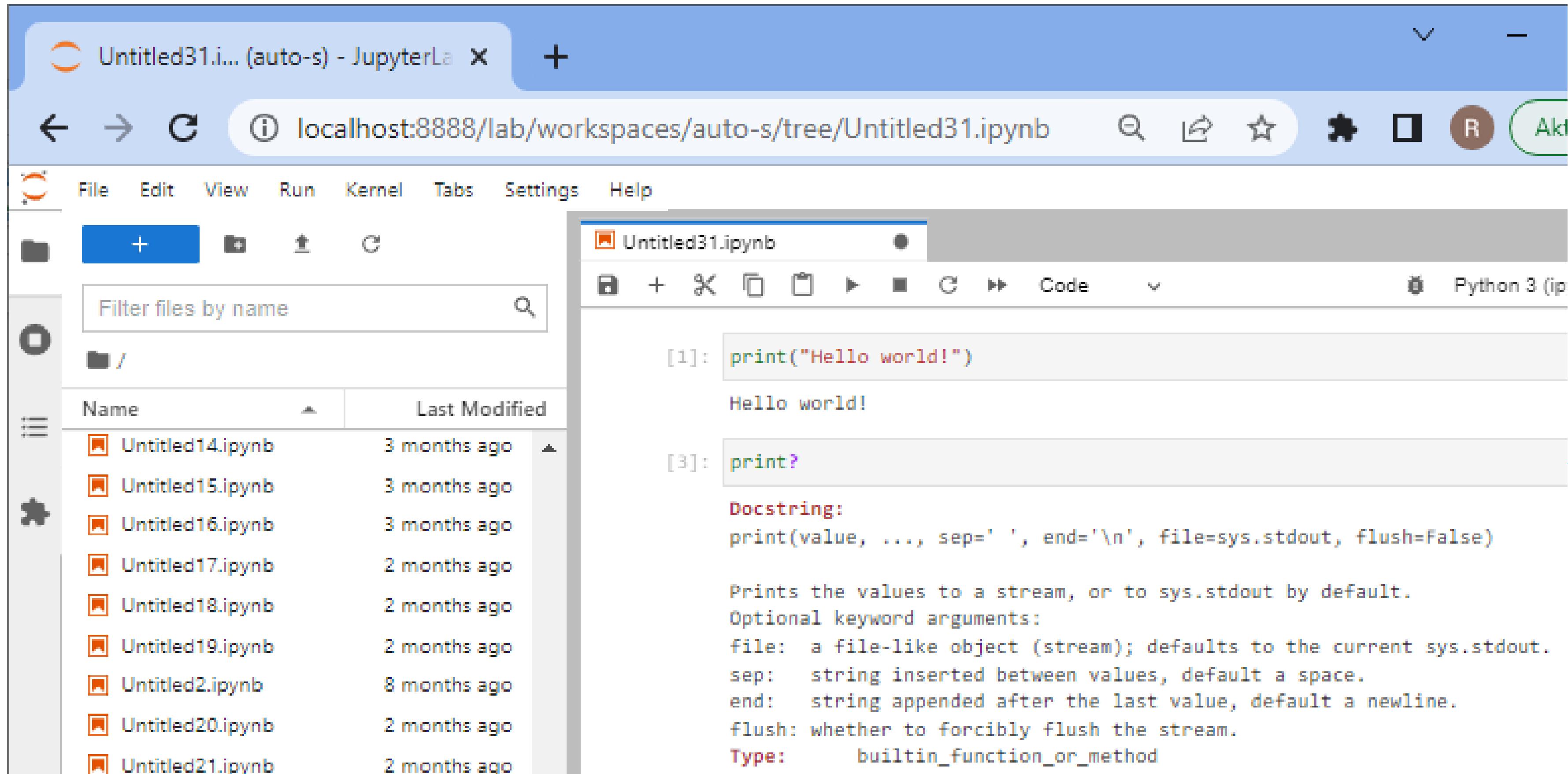
Context-specific help, auto-completion



TAB
to open
auto-
completion

Jupyter lab

Help / "docstrings"



The screenshot shows the Jupyter Lab interface. On the left, there's a file browser with a sidebar for filtering files by name. In the main area, a code cell contains the following Python code:

```
[1]: print("Hello world!")
```

The output of the cell is:

```
Hello world!
```

Below the cell, another code cell starts with:

```
[3]: print?
```

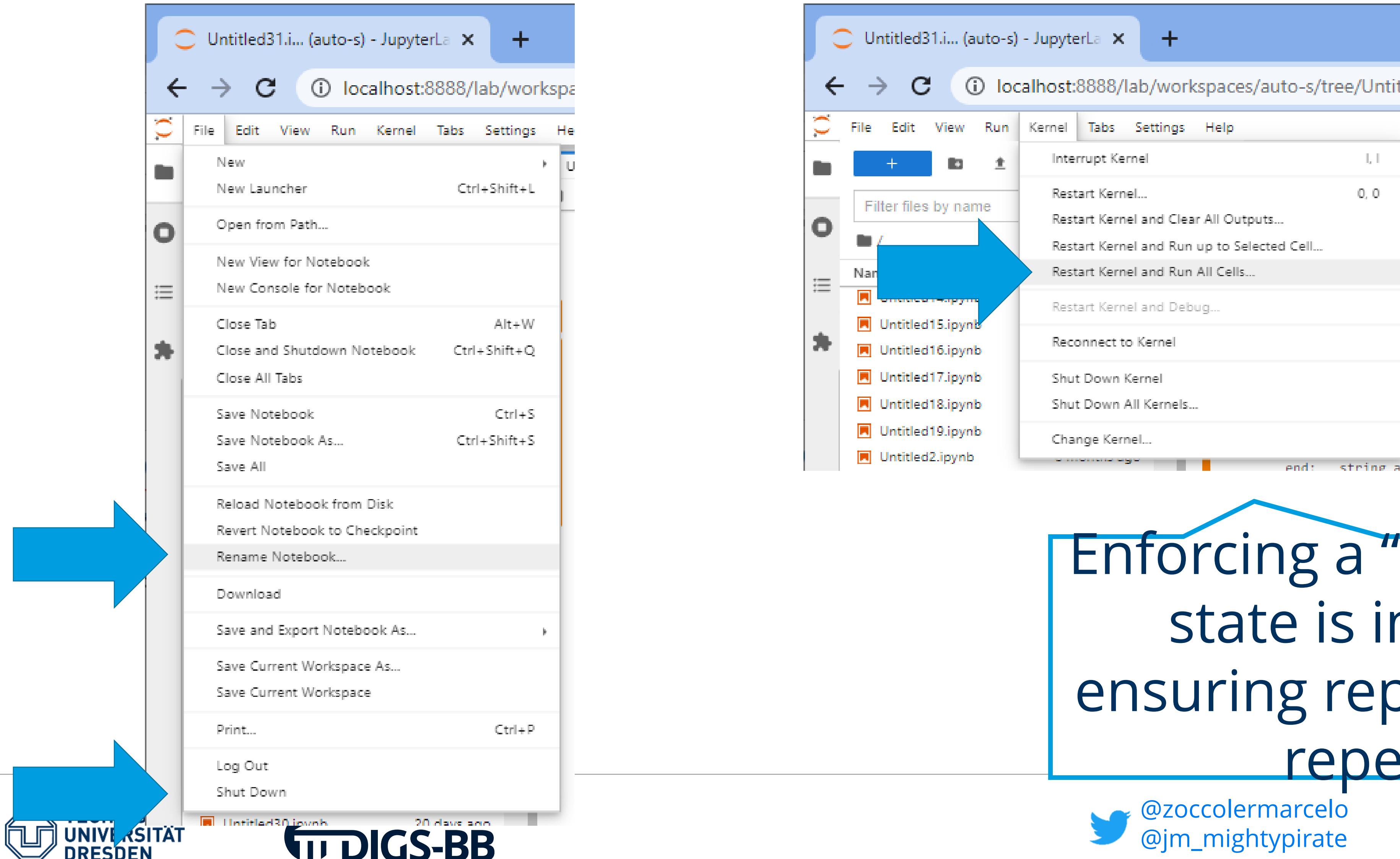
The output shows the docstring for the `print` function:

```
Docstring:  
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)  
  
Prints the values to a stream, or to sys.stdout by default.  
Optional keyword arguments:  
file: a file-like object (stream); defaults to the current sys.stdout.  
sep: string inserted between values, default a space.  
end: string appended after the last value, default a newline.  
flush: whether to forcibly flush the stream.  
Type: builtin_function_or_method
```

?
to read what a
function does

Jupyter lab

Saving / renaming / closing



Enforcing a "clean" execution state is important for ensuring reproducibility and repeatability

Python Data structures

Variables

Variables are memory blocks where you can store stuff

```
measurement = 5
```

```
name = "Drosophila"
```

```
combination = name + str(measurement)
```

Function(argument)

Computer memory

measurement

5

name

"Drosophila"

combination

"Drosophila5"

Arrays

Arrays are variables, where you can store multiple values

Give me a "0", five times!

```
array = [0] * 5
```

```
array[0] = 1
```

```
array[2] = 5
```

```
array[4] = "hi"
```

Computer memory

array

0	0	0	0	0
---	---	---	---	---

1	0	0	0	0
---	---	---	---	---

1	0	5	0	0
---	---	---	---	---

1	0	5	0	"hi"
---	---	---	---	------

Subsets

```
# Arrays  
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
print(numbers)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Creating subsets of arrays

Start

End

Step

```
subset = numbers[2:4]  
print(subset)
```

```
[2, 3]
```

```
subset_with_gaps = arr[1:8:2]  
print(subset_with_gaps)
```

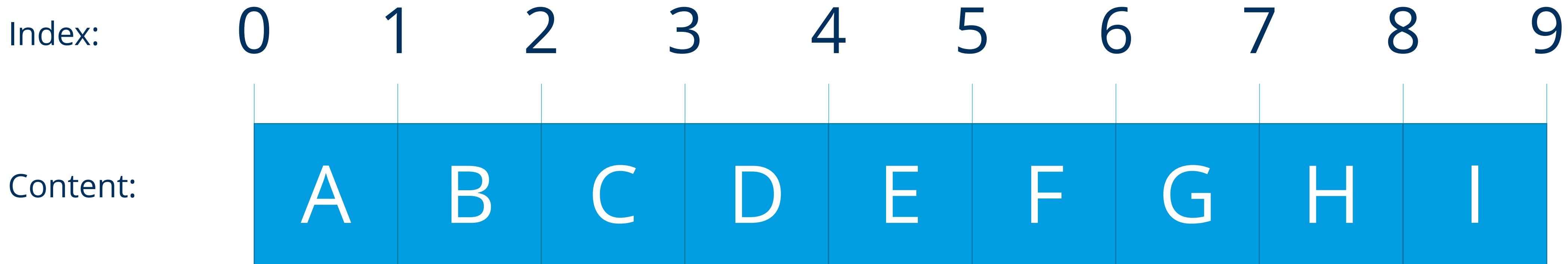
```
[1, 3, 5, 7]
```

data[start:stop:step]

Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

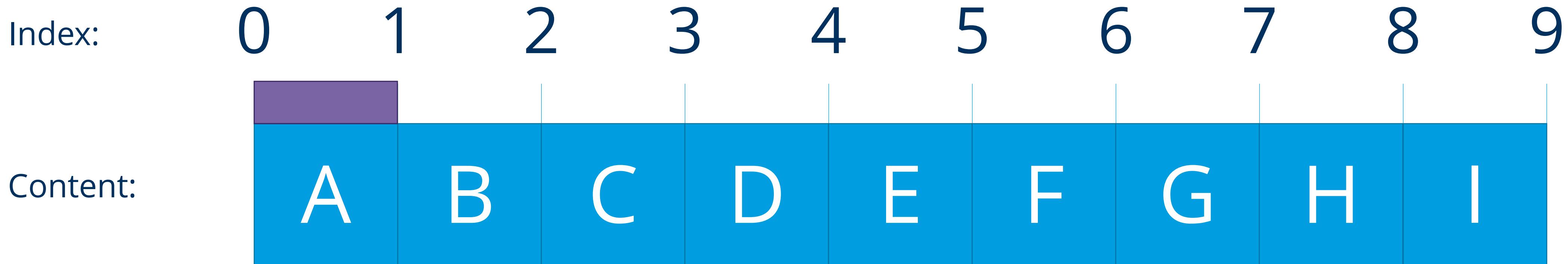
```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



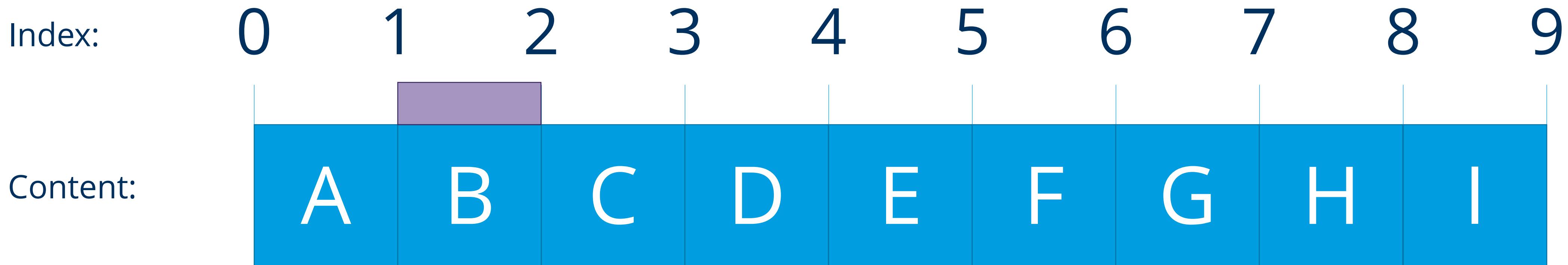
```
data[0]
```

'A'

Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

data[1]

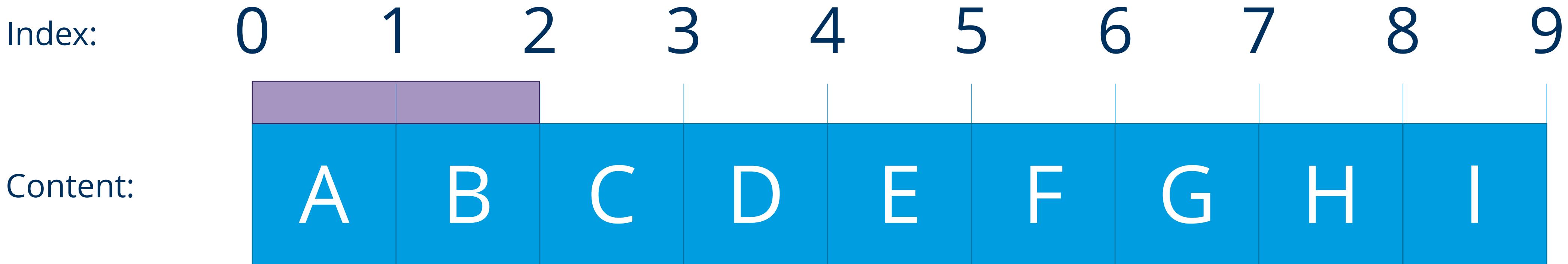
'A'

'B'

Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

data[1]

data[0:2]

'A'

'B'

['A', 'B']

Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index:

0 1 2 3 4 5 6 7 8 9

Content:



data[0]

data[1]

data[0:2]

data[0:3]

data[1:2]

len(data)

'A'

'B'

['A', 'B']

['A', 'B', 'C']

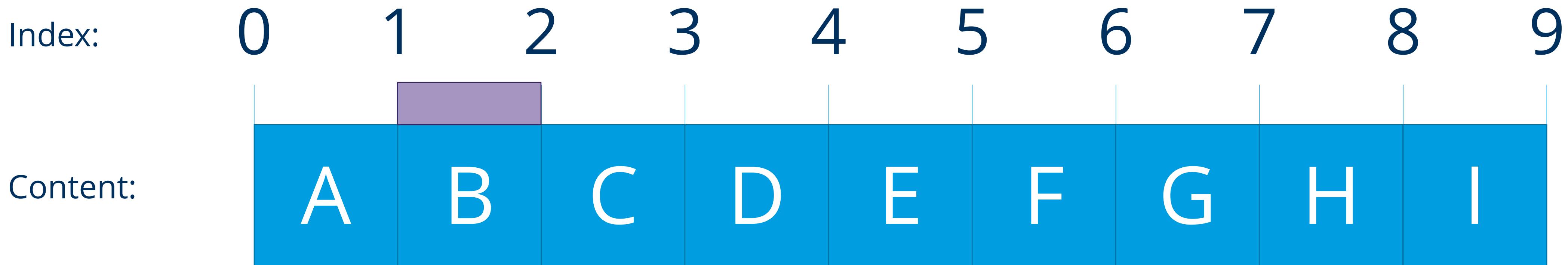
['B']

9

Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

data[1]

data[0:2]

data[0:3]

data[1:2]

'A'

'B'

['A', 'B']

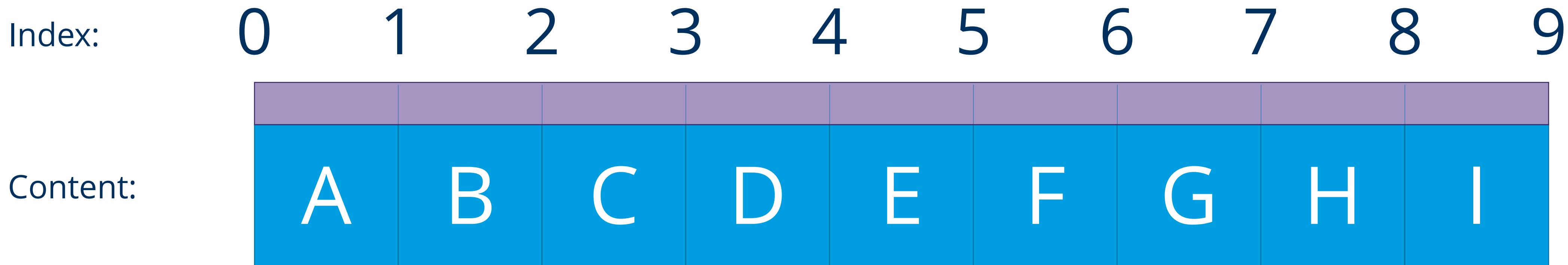
['A', 'B', 'C']

['B']

Indexing, cropping, subsets

“Indexing” is addressing certain elements in arrays. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

data[1]

data[0:2]

data[0:3]

data[1:2]

len(data)

'A'

'B'

['A', 'B']

['A', 'B', 'C']

['B']

9

Indexing, cropping, subsets

You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



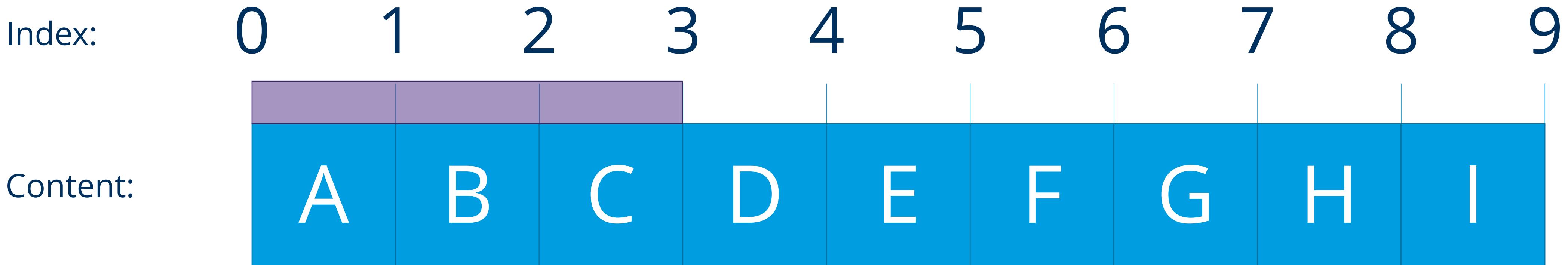
```
data[:2]
```

```
['A', 'B']
```

Indexing, cropping, subsets

You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
data[:3]
```

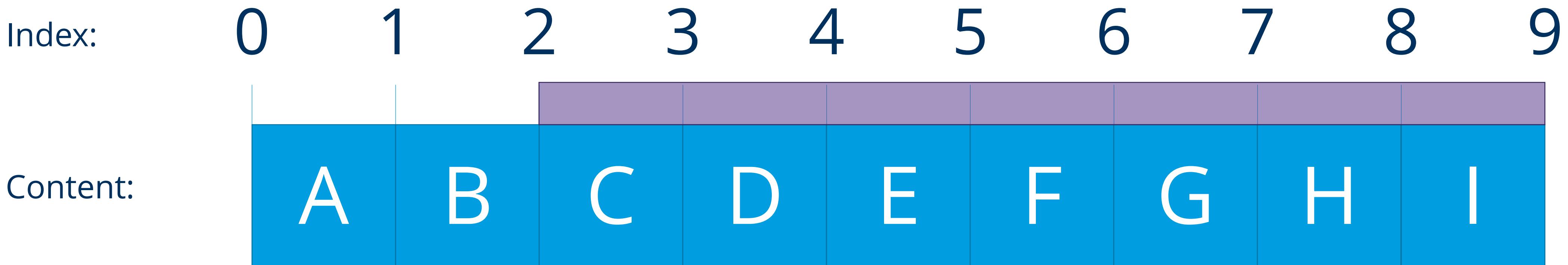
```
['A', 'B']
```

```
['A', 'B', 'C']
```

Indexing, cropping, subsets

You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
data[:3]
```

```
data[2:]
```

```
['A', 'B']
```

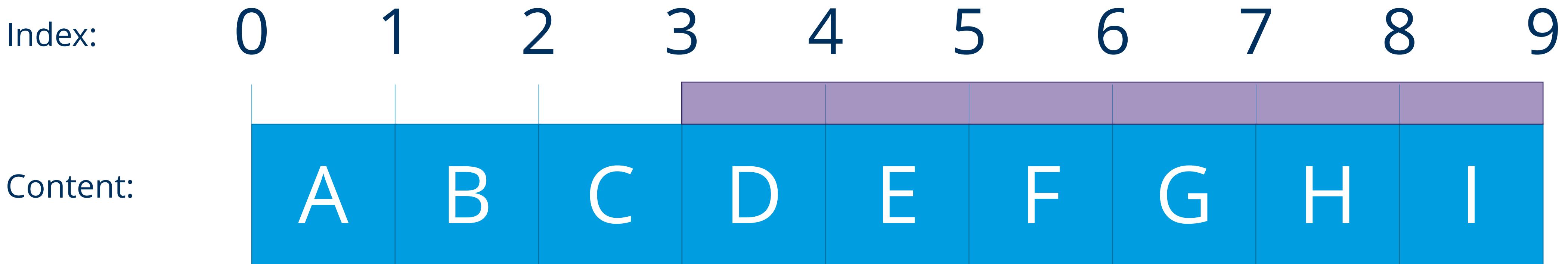
```
['A', 'B', 'C']
```

```
['C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Indexing, cropping, subsets

You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
data[:3]
```

```
data[2:]
```

```
data[3:]
```

```
['A', 'B']
```

```
['A', 'B', 'C']
```

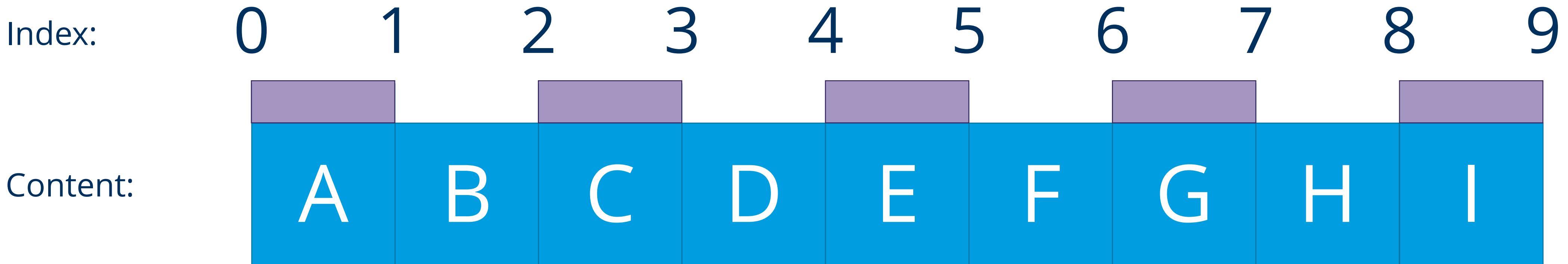
```
['C', 'D', 'E', 'F', 'G', 'H', 'I']
```

```
['D', 'E', 'F', 'G', 'H', 'I']
```

Indexing, cropping, subsets

The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



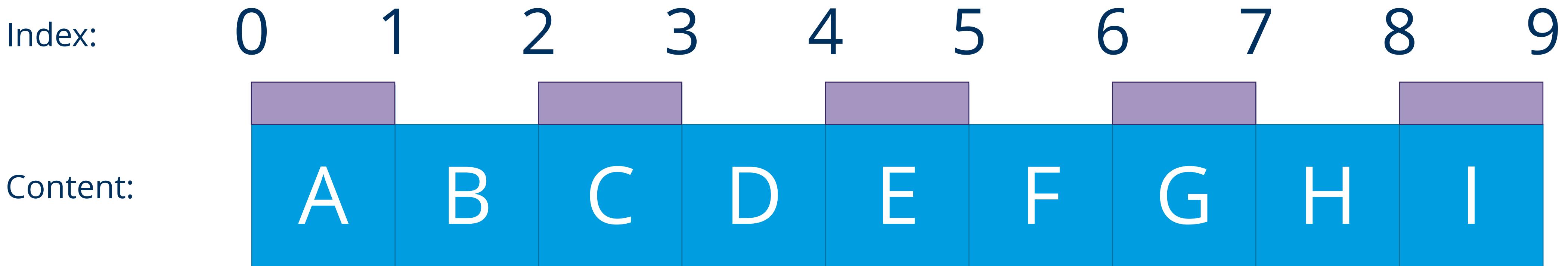
```
data[0:10:2]
```

```
['A', 'C', 'E', 'G', 'I']
```

Indexing, cropping, subsets

The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

```
data[::2]
```

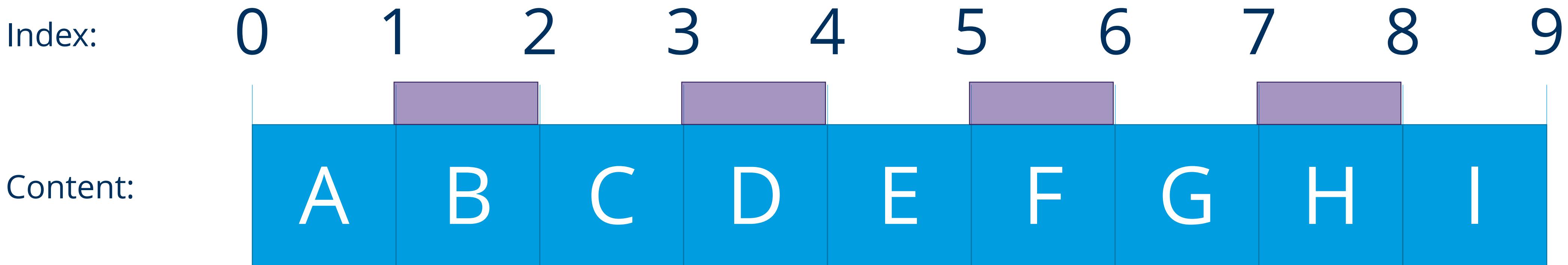
```
['A', 'C', 'E', 'G', 'I']
```

```
['A', 'C', 'E', 'G', 'I']
```

Indexing, cropping, subsets

The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

```
data[::-2]
```

```
data[1::2]
```

```
['A', 'C', 'E', 'G', 'I']
```

```
['A', 'C', 'E', 'G', 'I']
```

```
['B', 'D', 'F', 'H']
```

Indexing, cropping, subsets

Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[-2:]
```

```
['H', 'I']
```

Indexing, cropping, subsets

Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index: -9 -8 -7 -6 -5 -4 -3 -2 -1



Content:

A B C D E F G H I

```
data[-2:]
```

```
data[:-2]
```

['H', 'I']

['A', 'B', 'C', 'D', 'E', 'F', 'G']

Indexing, cropping, subsets

Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index: -9 -8 -7 -6 -5 -4 -3 -2 -1

Content: A B C D E F G H I

```
data[-2:]
```

```
data[:-2]
```

```
data[-7:-5]
```

```
['H', 'I']
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

```
['C', 'D']
```

Indexing, cropping, subsets

Negative stepping also works

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



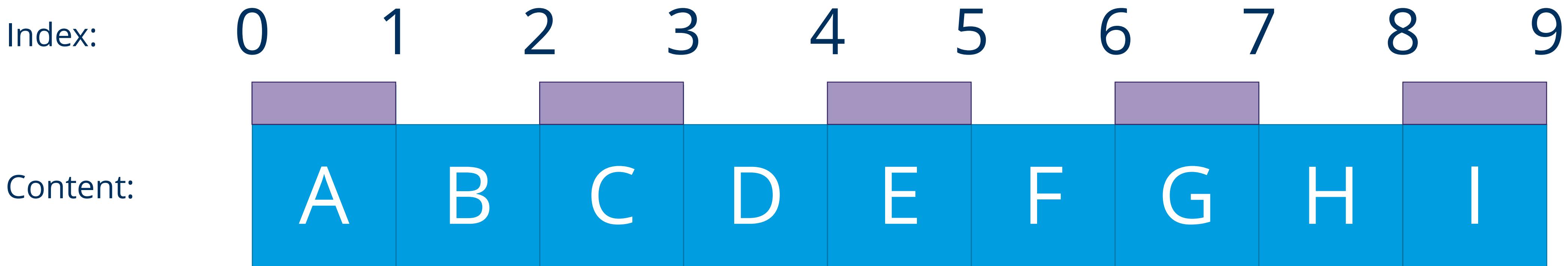
```
data[:: -1]
```

```
['I', 'H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
```

Indexing, cropping, subsets

Negative stepping also works

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:::-1]
```

```
['I', 'H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
```

```
data[:::-2]
```

```
['I', 'G', 'E', 'C', 'A']
```

Arrays in Python

Modifying array elements

```
▶ numbers = [0, 1, 2, 3, 4]  
  
# write in one array element  
numbers[1] = 5  
  
print(numbers)  
  
[0, 5, 2, 3, 4]
```

Note: The first element has index 0!

- Creating arrays of defined size

What?

How many?

```
▶ zeros = [0] * 10  
print(zeros)
```

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

- Concatenating arrays

```
▶ ones = [1, 1, 1]  
twos = [2, 2, 2]  
  
# concatenate arrays  
numbers = ones + twos  
  
print(numbers)
```

[1, 1, 1, 2, 2, 2]

+ means appending

Arrays: Lists versus Tuples

Lists can be modified

```
measurements = [5.5, 6.3, 7.2, 8.0, 8.8]
```

Create

```
measurements.append(10.2)
```

Add values

```
measurements[1] = 25
```

Replace values

```
measurements
```

Inspect content

```
[5.5, 25, 7.2, 8.0, 8.8, 10.2]
```

Tuples not

```
immutable = (4, 3, 7.8)
```

Create

```
immutable[1] = 5
```

TypeError

```
<ipython-input-49-a01b13633c23> in <module>
----> 1 immutable[1] = 5
```

Traceback (most recent call last)

```
TypeError: 'tuple' object does not support item assignment
```

Check the data type if uncertain:

```
type(measurements)
```

list

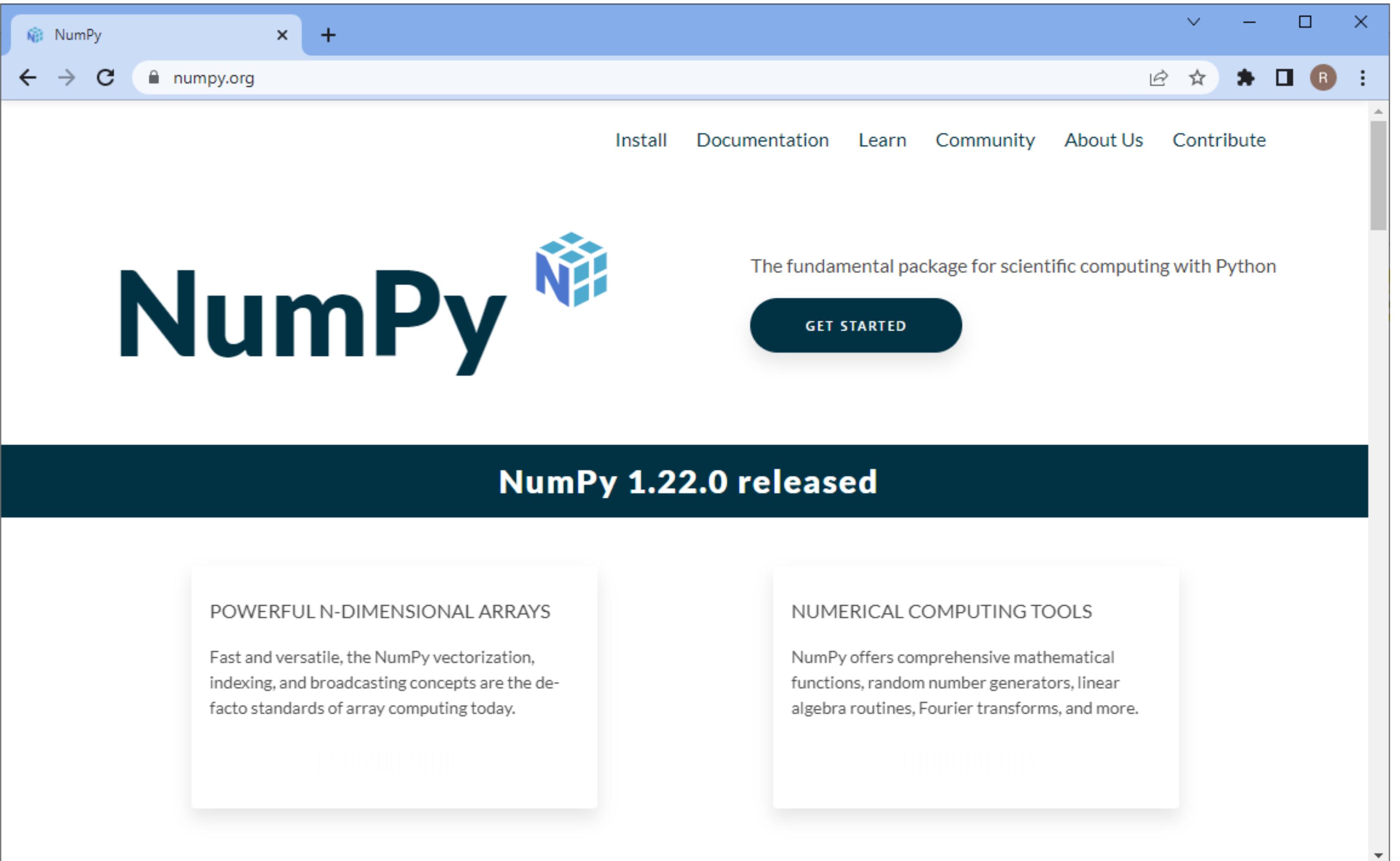
```
type(immutable)
```

tuple

numpy

The fundamental package for scientific computing with python.

conda install numpy



<https://numpy.org/>

numpy

Simplifying mathematical operations on n-dimensional arrays

- Python arrays of arrays (lists of lists)

```
► # multidimensional arrays  
matrix = [  
    [1, 2, 3],  
    [2, 3, 4],  
    [3, 4, 5]  
]  
  
print(matrix)  
[[1, 2, 3], [2, 3, 4], [3, 4, 5]]
```

```
► result = matrix * 2  
print(result)  
[[1, 2, 3], [2, 3, 4], [3, 4, 5], [1, 2, 3], [2, 3, 4], [3, 4, 5]]
```

- numpy arrays

```
► import numpy as np
```

```
np_matrix = np.asarray(matrix)  
print(np_matrix)
```

```
[[1 2 3]  
 [2 3 4]  
 [3 4 5]]
```

```
► np_result = np_matrix * 2  
print(np_result)
```

```
[[ 2  4  6]  
 [ 4  6  8]  
 [ 6  8 10]]
```

Tell python that
you want to use a
library called
numpy

If "numpy" is too
long, you can give
an alias "np"

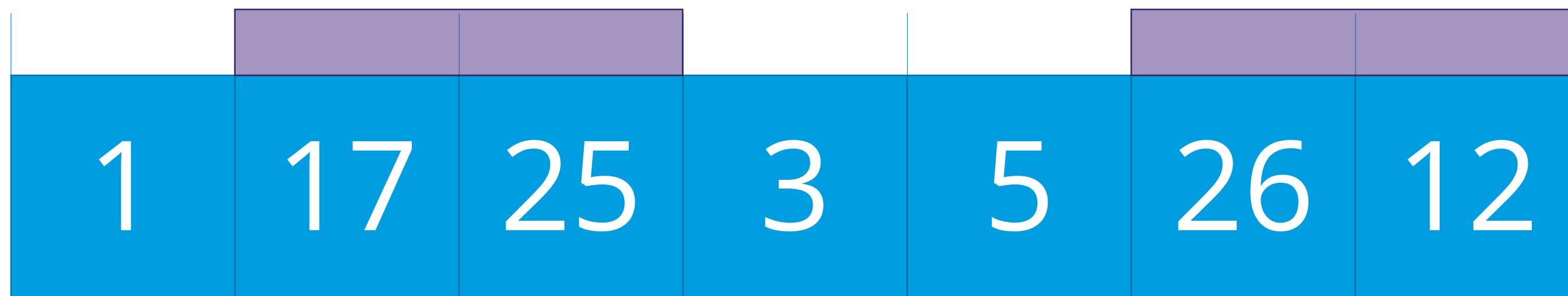
Masking (a.k.a. logical indexing)

“Masking” is addressing certain elements in numpy arrays, e.g. depending on their content

```
import numpy  
measurements = numpy.asarray([1, 17, 25, 3, 5, 26, 12])  
measurements
```

```
array([ 1, 17, 25, 3, 5, 26, 12])
```

Content:



```
mask = measurements > 10
```

```
mask
```

```
measurements[mask]
```

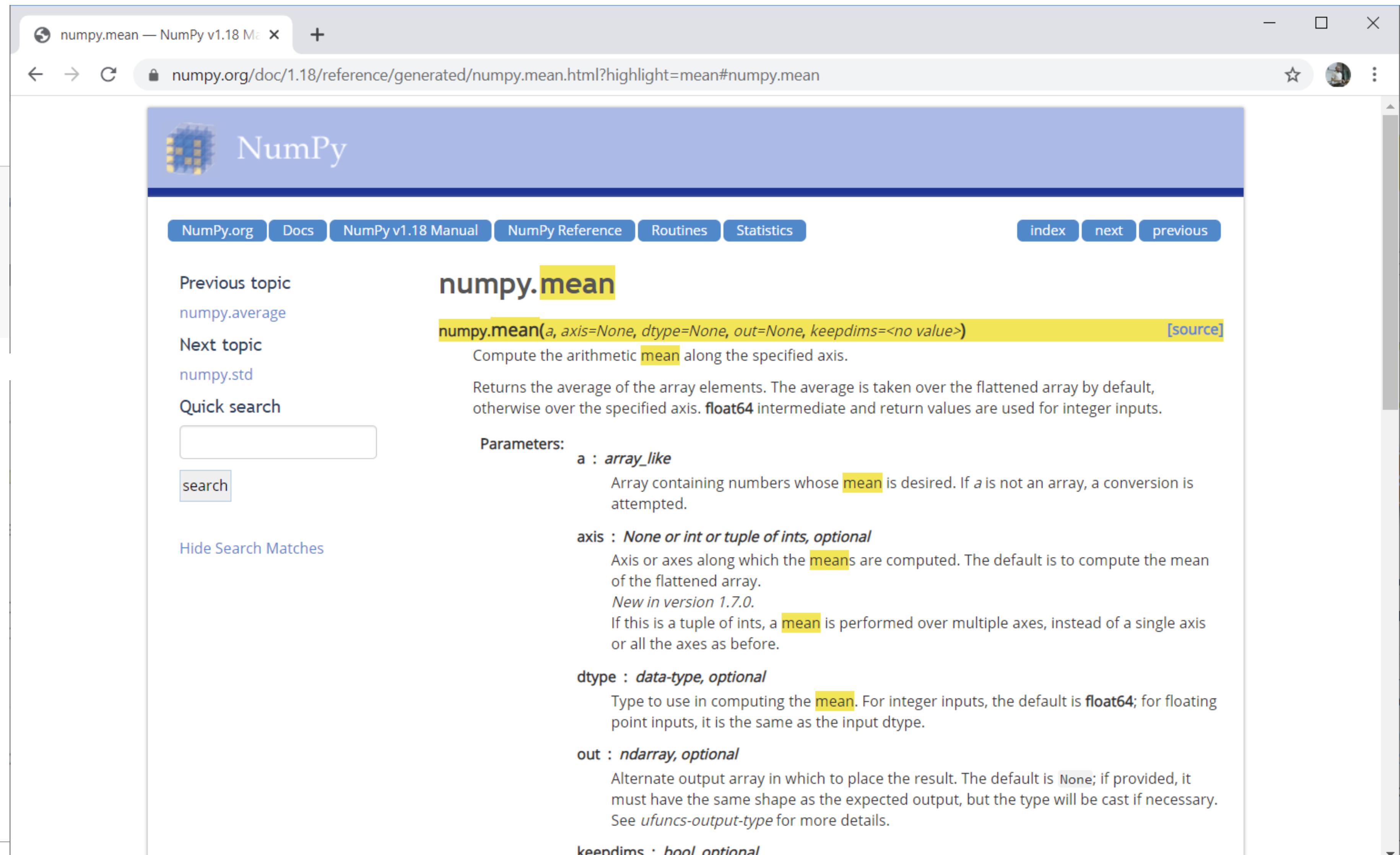
```
array([False, True, True, False, False, True, True])
```

```
array([17, 25, 26, 12])
```

Basic descriptive statistics using numpy

Basic descriptive statistics

```
▶ import numpy as np  
  
measurements = [1, 4, 6, 7, 2]  
  
mean = np.mean(measurements)  
print("Mean: " + str(mean))  
  
Mean: 4.0
```



The screenshot shows a web browser window displaying the NumPy documentation for the `numpy.mean` function. The URL in the address bar is `numpy.org/doc/1.18/reference/generated/numpy.mean.html?highlight=mean#numpy.mean`. The page title is "NumPy". On the left, there is a sidebar with links to "NumPy.org", "Docs", "NumPy v1.18 Manual", "NumPy Reference", "Routines", "Statistics", "index", "next", and "previous". Below these are links to "Previous topic" (`numpy.average`), "Next topic" (`numpy.std`), and a "Quick search" input field with a "search" button. A "Hide Search Matches" link is also present. The main content area is titled "numpy.mean" and contains the function signature `numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)`. It includes a "[source]" link. The function is described as computing the arithmetic mean along the specified axis, returning the average of array elements. The parameters are detailed as follows:

- a : array_like**: Array containing numbers whose mean is desired. If `a` is not an array, a conversion is attempted.
- axis : None or int or tuple of ints, optional**: Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. *New in version 1.7.0.* If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.
- dtype : data-type, optional**: Type to use in computing the mean. For integer inputs, the default is `float64`; for floating point inputs, it is the same as the input `dtype`.
- out : ndarray, optional**: Alternate output array in which to place the result. The default is `None`; if provided, it must have the same shape as the expected output, but the type will be cast if necessary. See `ufuncs-output-type` for more details.
- keepdims : bool, optional**: If `True`, the axes which are reduced are left in the result as dimensions with size one. In this case, the result is broadcasted with the original arrays.

scikit-image

scikit-image is a collection of algorithms for image processing.

conda install scikit-image

The screenshot shows a web browser window displaying the official website for scikit-image (<https://scikit-image.org/>). The page features a header with the scikit-image logo and navigation links for Installation, Gallery, Documentation, Community, and Source. A search bar is also present. The main content area includes sections for 'Stable' (version 0.19.2) and 'Development' (pre-0.20) releases, each with a download button. A large central box highlights 'Image processing in Python' as a free and open-source project. Below this, a light blue box provides citation information and links to GitHub, Contribute, Get Help, Discuss, and StackOverflow. A 'News' section at the bottom announces the release of version 0.19.2 on February 17, 2022. A green 'OPEN CHAT' button is located in the bottom right corner.

<https://scikit-image.org/>

Working with images in python

Open images

```
from skimage.io import imread  
  
image = imread("blobs.tif")
```

```
image
```

```
array([[[ 40,  32,  24, ..., 216, 200, 200],  
       [[ 56,  40,  24, ..., 232, 216, 216]],  
       [[ 64,  48,  24, ..., 240, 232, 232]],  
       ...,  
       [[ 72,  80,  80, ..., 48,  48,  48],  
        [ 80,  80,  80, ..., 48,  48,  48],  
        [ 96,  88,  80, ..., 48,  48,  48]]], dtype=uint8)
```

Images are *just* multi-dimensional arrays or “arrays of arrays”.

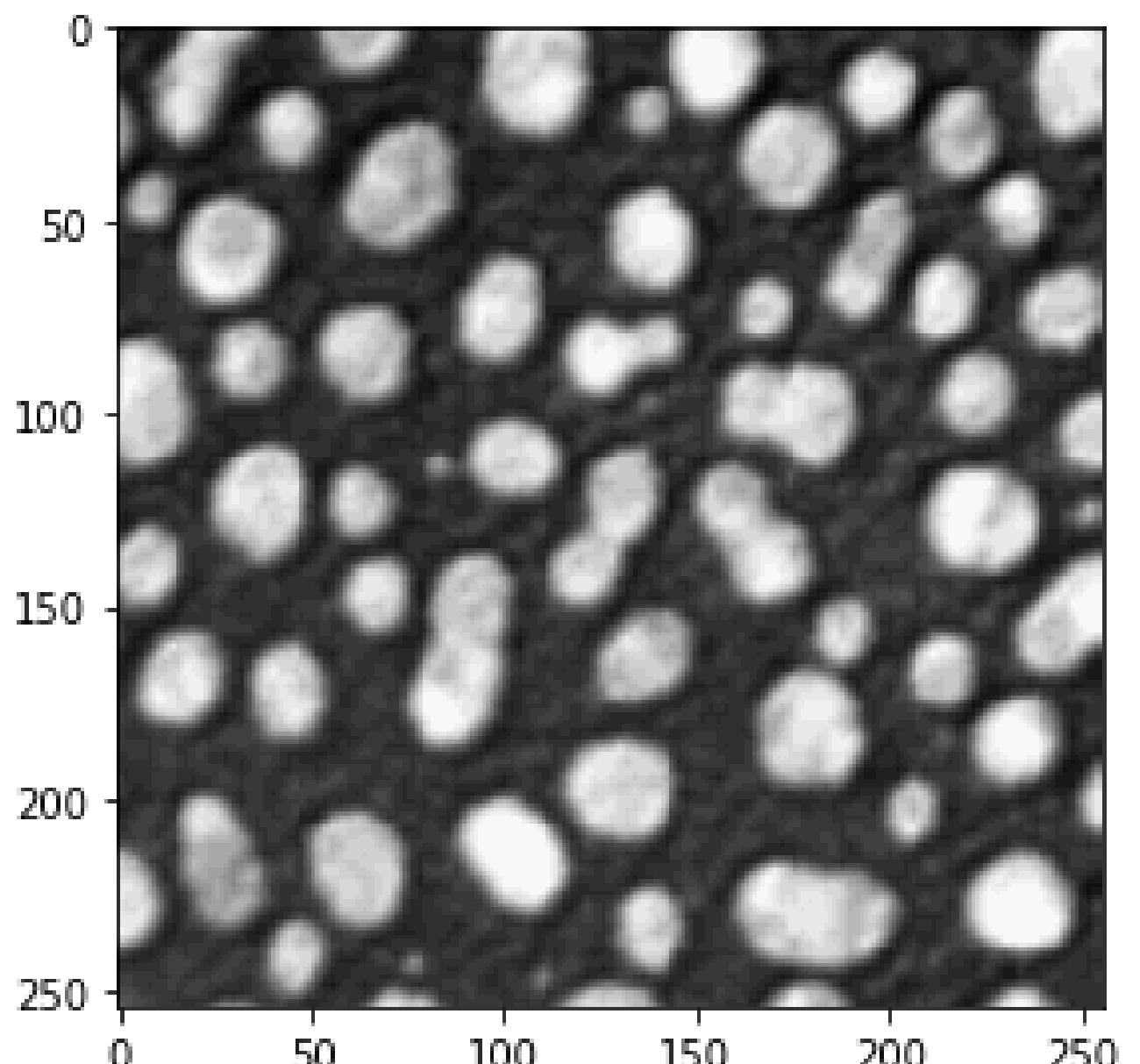
Working with images in python

Open images

```
from skimage.io import imread  
  
image = imread("blobs.tif")
```

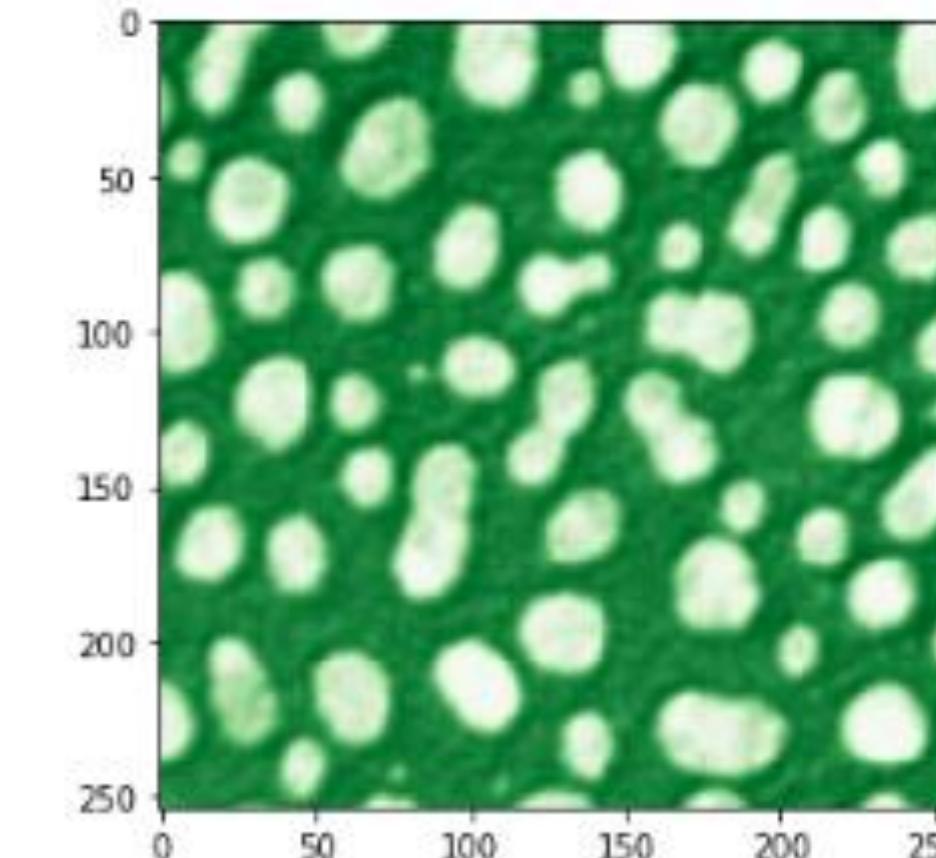
• Visualize images

```
from skimage.io import imshow  
  
imshow(image)  
  
<matplotlib.image.AxesImage at 0x245e7e>
```



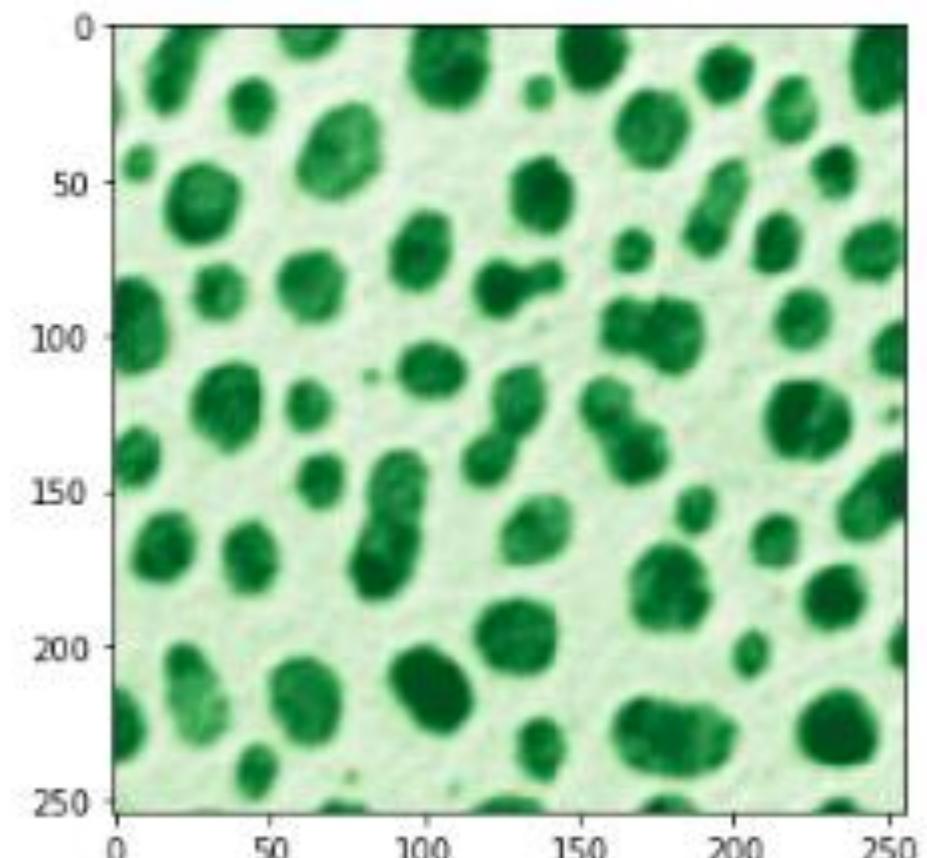
```
imshow(image, cmap="Greens_r")
```

```
<matplotlib.image.AxesImage at 0:0>
```



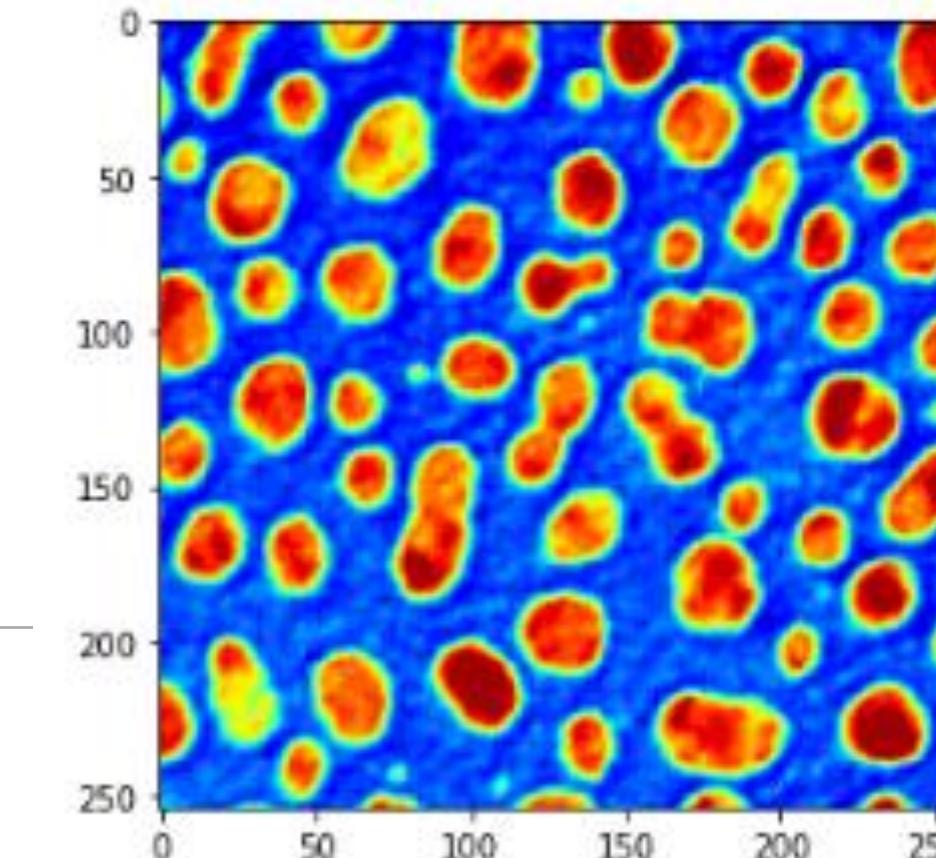
```
imshow(image, cmap="Greens")
```

```
<matplotlib.image.AxesImage at 0:0>
```



```
imshow(image, cmap="jet")
```

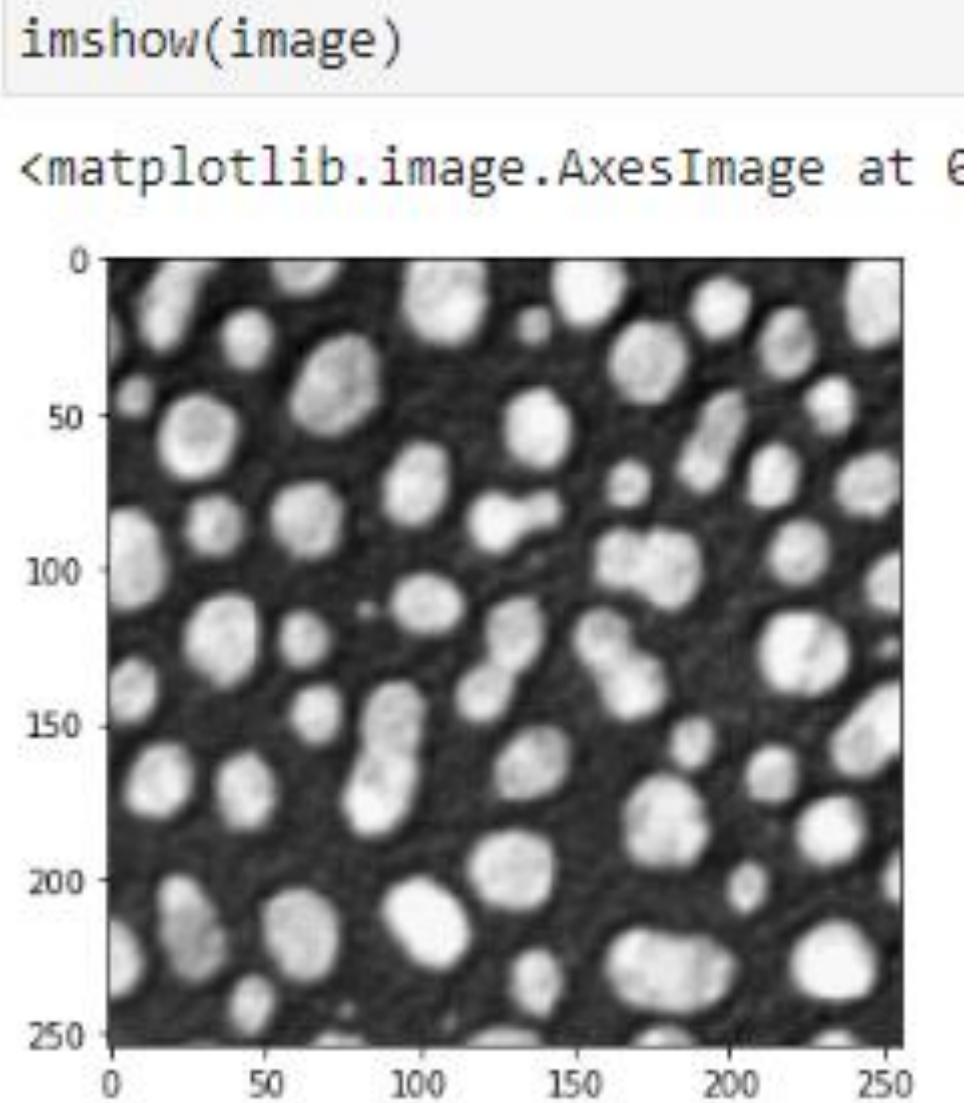
```
<matplotlib.image.AxesImage at 0:0>
```



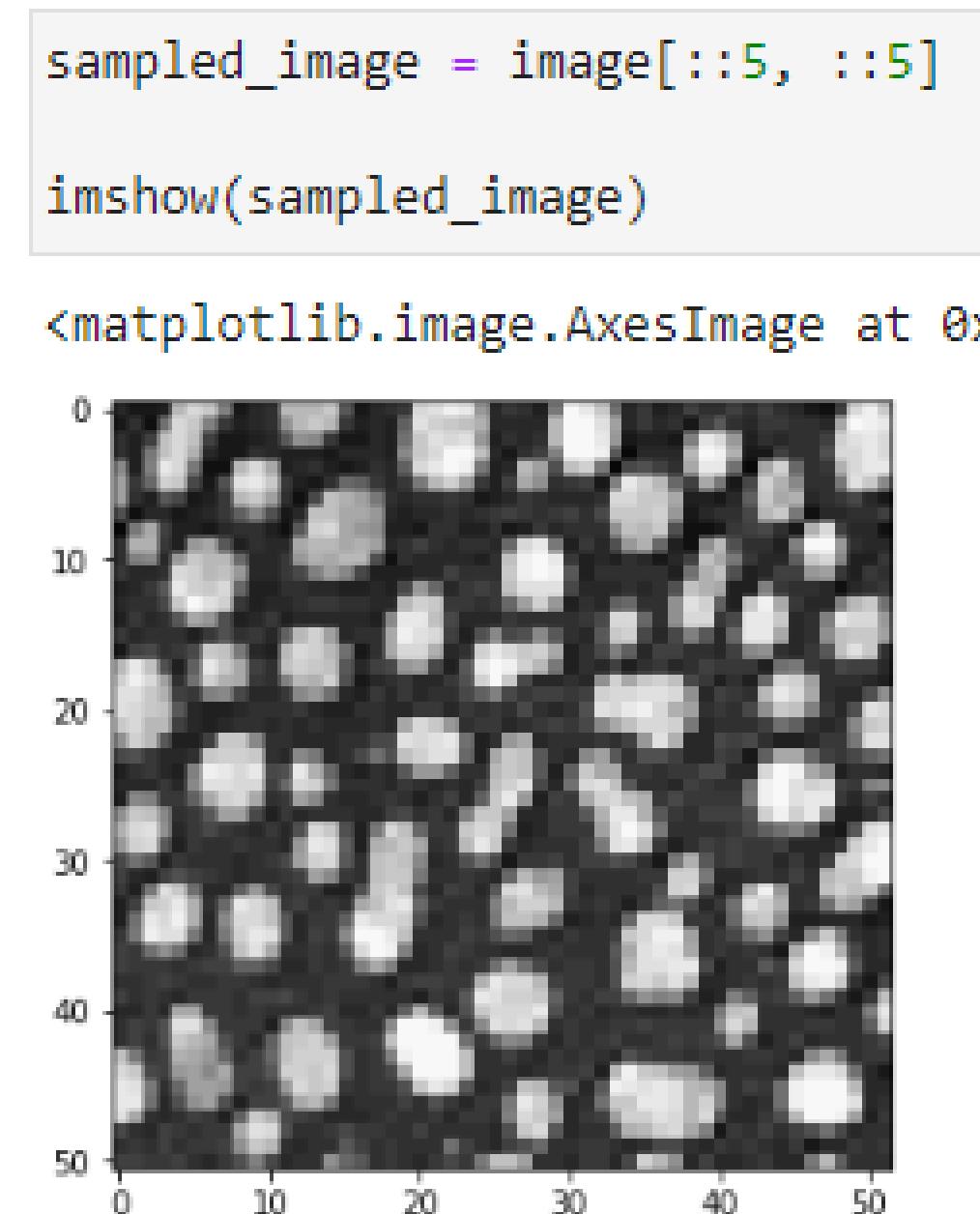
This does not modify the image data. The images are just shown with different colors representing the same values.

Cropping, sampling and flipping images

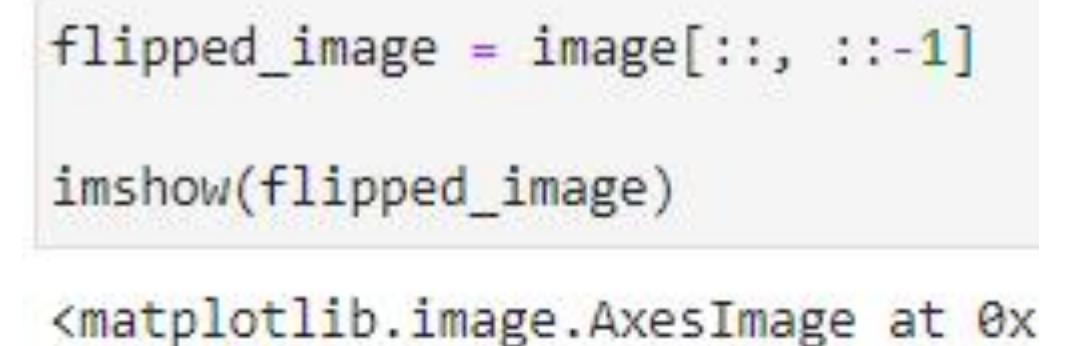
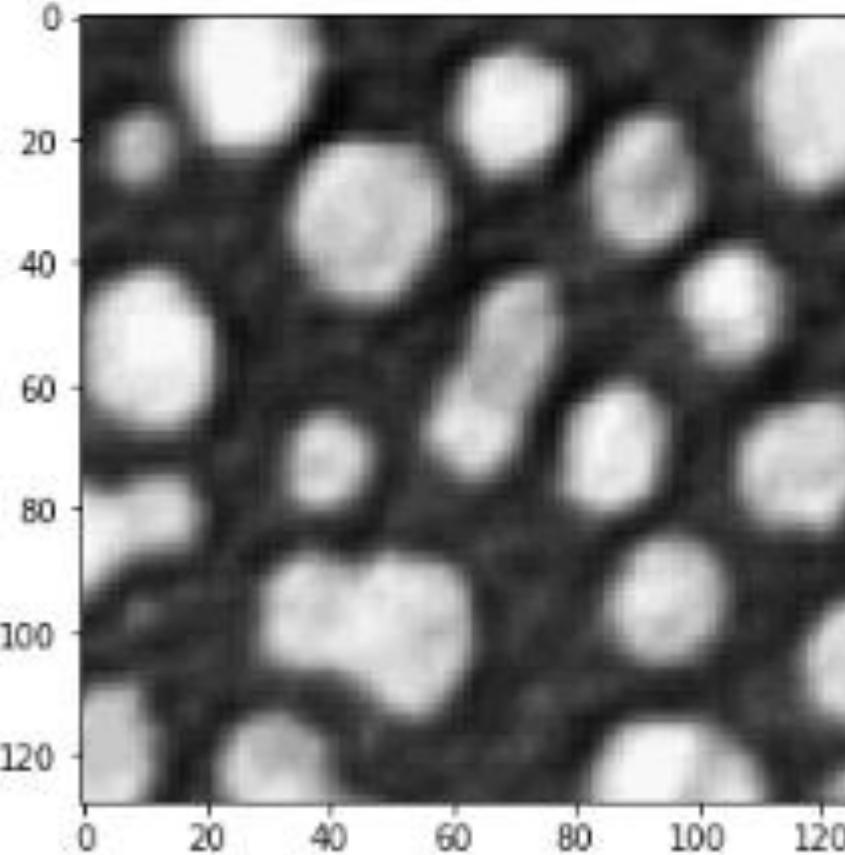
Indexing and cropping *numpy-arrays* works like with python lists.



Original image



Sub-sampled image



Flipped image

Troubleshooting

If your program throws error messages:

- Don't panic.
- *"There are two ways to write error-free programs; only the third one works."*

Alan J. Perlis, Yale University

- Read where the error happened.
 - You may see your fault immediately, when looking at the right point.
- Read what appears to be wrong.
 - If you know, what's missing, you may see it, even if it's missing in a slightly different place.
 - Sometimes, something related is missing

```
▶ print(round(4.5)
      File "<ipython-input-15-09a9be4a90c5>", line 1
            print(round(4.5)
                  ^
SyntaxError: unexpected EOF while parsing
```

Summary

Take home messages

Arrays can be accessed like this:

data[start:stop:step]

Strings are arrays

Lists are arrays

Tuples are arrays

Dictionaries are arrays with named elements

Columns in tables are arrays

Images are multi-dimensional arrays

Learning how to deal with arrays in Python is key.

Coming up next

- Loops
- Conditions
- Functions
- Libraries

```
► animal_set = ["Cat", "Dog", "Mouse"]  
  
for animal in animal_set:  
    print(animal)
```

Cat
Dog
Mouse

Python Algorithms

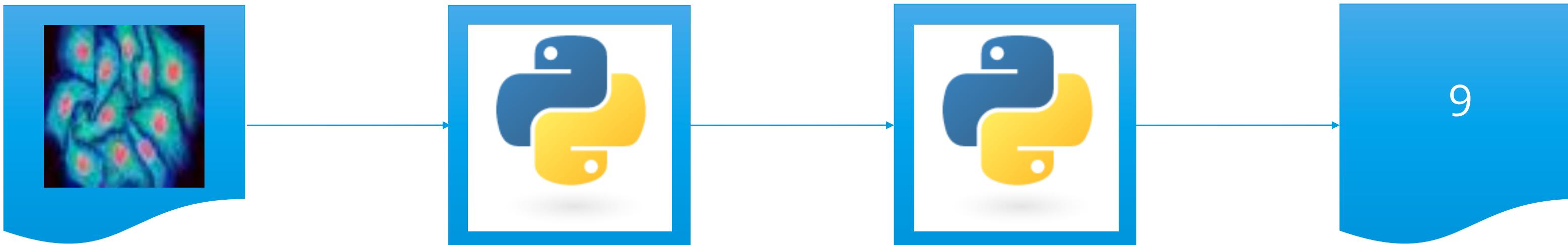
Conditions, loops, functions and custom libraries

Johannes Müller

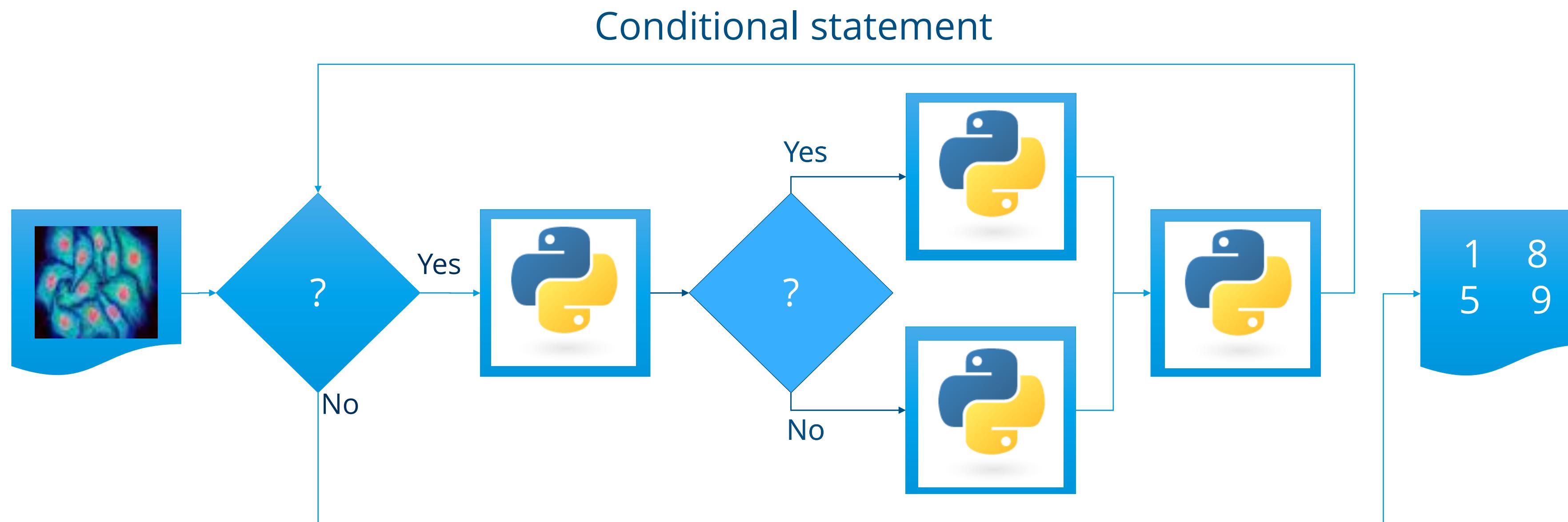
Using material from Robert Haase & Benoit Lombardot,
Scientific Computing Facility, MPI CBG

Conditions

Data science workflows *rarely* look like this:



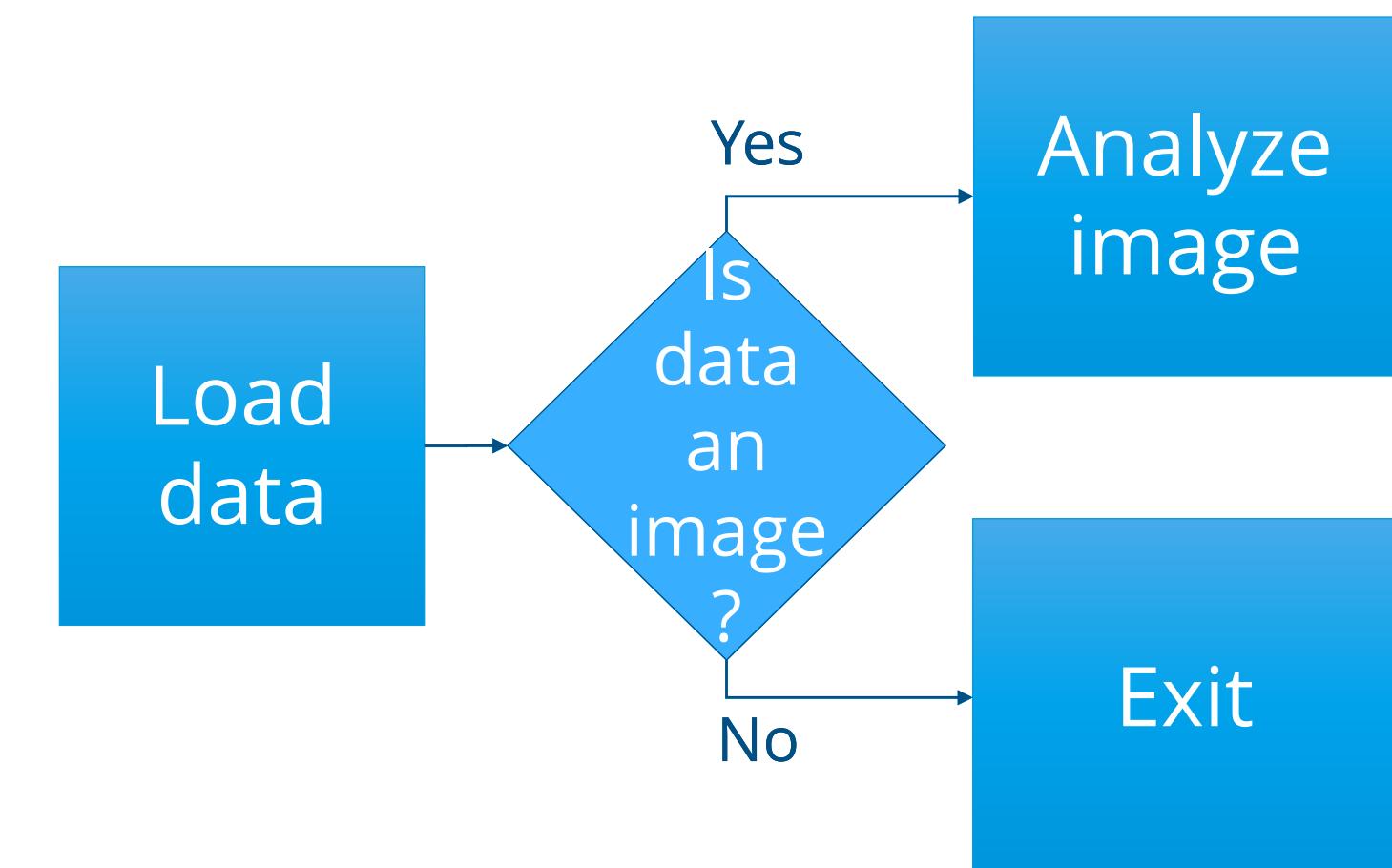
Data science workflows *rather* look like this:



Conditions

Conditional statements can be used to

- Check if pre-requisites are met
- Check if data has the right format
- Check if processing results are within an expected range
- Check for errors



If-statement

Depending on a condition, some lines of code are executed or not.

```
# do something  
if condition:  
    # do something only if condition is true  
# proceed
```

```
# do something  
if condition:  
    # do something only if condition is true  
# proceed
```

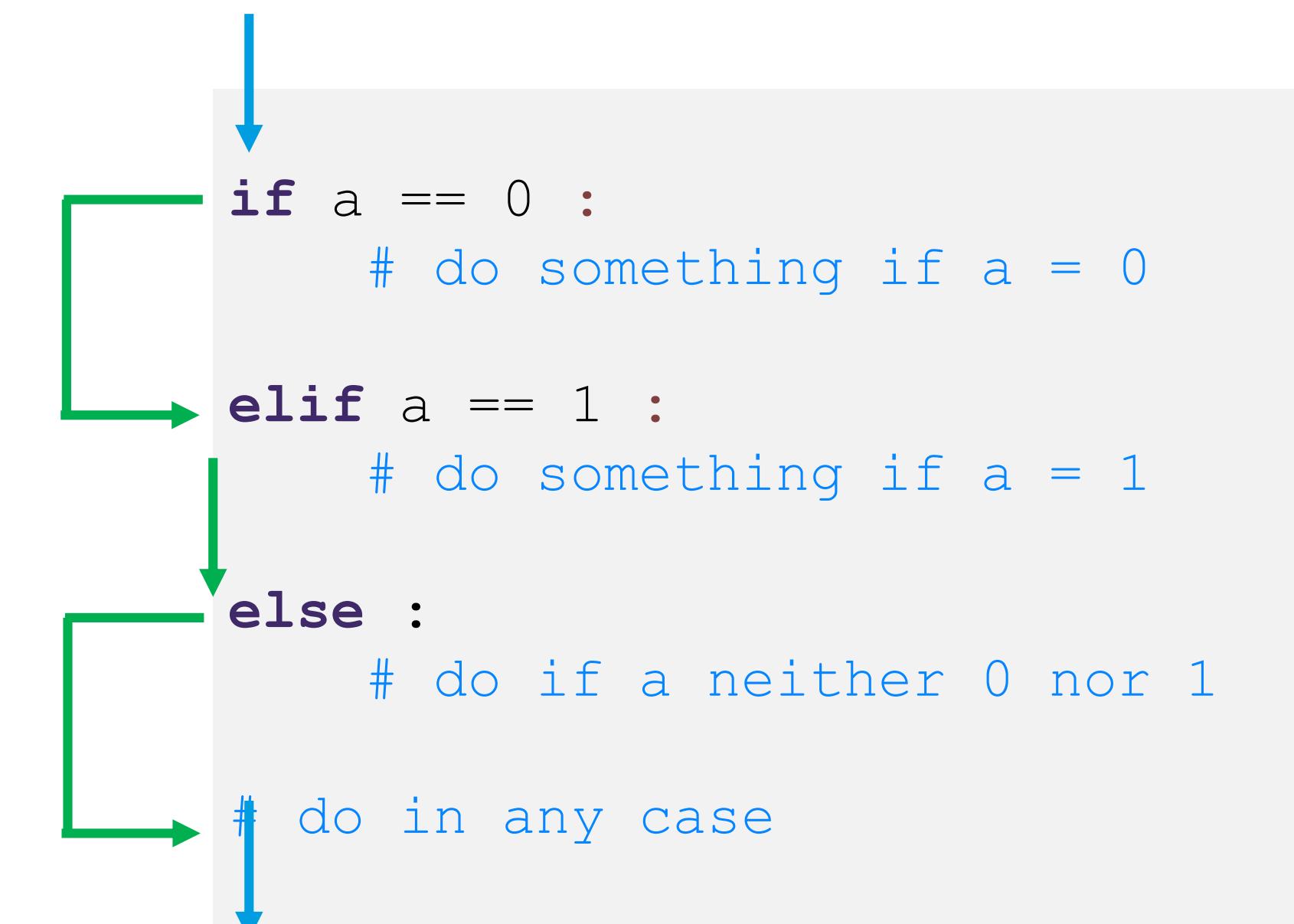
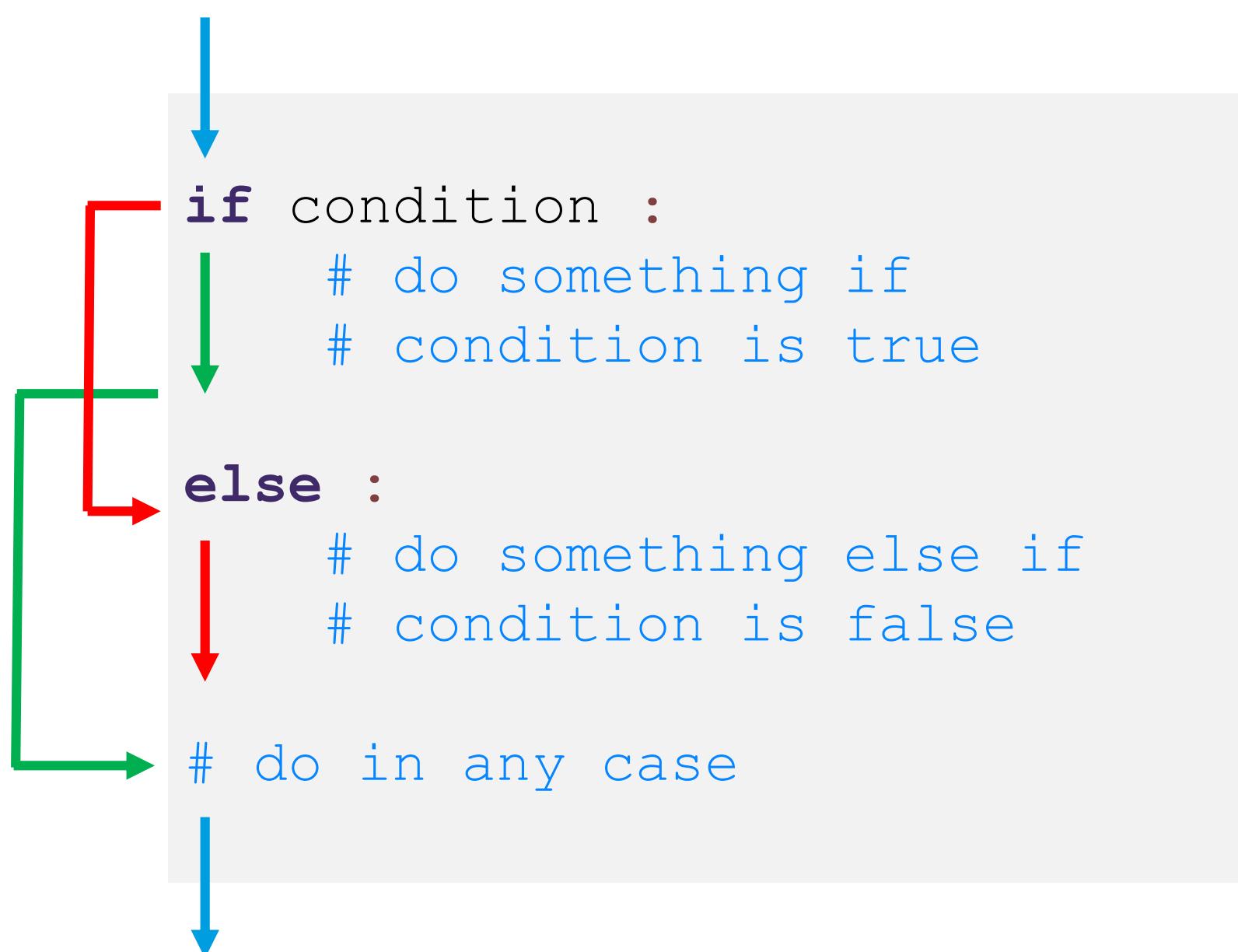
If-statement

The **if** / **elif** / **else** statement allows to program alternatives.

Depending on conditions, only one block is computed

Indentation is used to mark where a block starts and ends.

Indentation helps reading blocks,



If-statement

Comparison operators always have True (1) or False (0) as results.

```
# initialise program
quality = 99.5

# evaluate result
if quality > 99.9 :
    print("Everything is fine.")
else :
    print("We need to improve!")
```

In [1]: a = 4
if a = 5:
 print("Hello world")

File "<ipython-input-1-13fb587c9332>", line 3
 if a = 5:
 ^
SyntaxError: invalid syntax

Note: These are two equal signs!

Operator	Description	Example
<, <=	smaller than, smaller or equal	a < b

Combined conditions

Logic operators always take conditions as operands and result in a condition.

- and
- or
- not

Also combined conditions can be either True (1) or False (0).

```
# initialise program
quality = 99.9
age = 3

if quality >= 99.9 and age > 5 :
    print("The item is ok.")
```

```
# initialise program
quality = 99.9

if not quality < 99.9 :
    print("The item is ok.")
```

Conditions with arrays

Checking contents of lists can be done intuitively using the `in` statement

```
# initialise program
my_list = [1, 5, 7, 8]
item = 3

if item in my_list :
    print("The item is in the list.")
else :
    print("There is no", item, "in", my_list )
```

Works for strings, too!

```
cellline = 'PDX_10X_gH2AX_M23'
'PDX' in cellline
```

True

```
# initialise program
my_list = [1, 5, 7, 8]
item = 3

if item not in my_list :
    print("There is no", item, "in", my_list )
else :
    print("The item is in the list.")
```

Readable code

Rules for readable code

- Every command belongs on its own line

- Insert empty lines to separate important processing steps

- Put spaces between operators and operands, because:

This is easier to read than that, or isn't it?

- Indent every conditional block (if/else) using the TAB key

```
# initialise program
a = 5
b = 3
c = 8

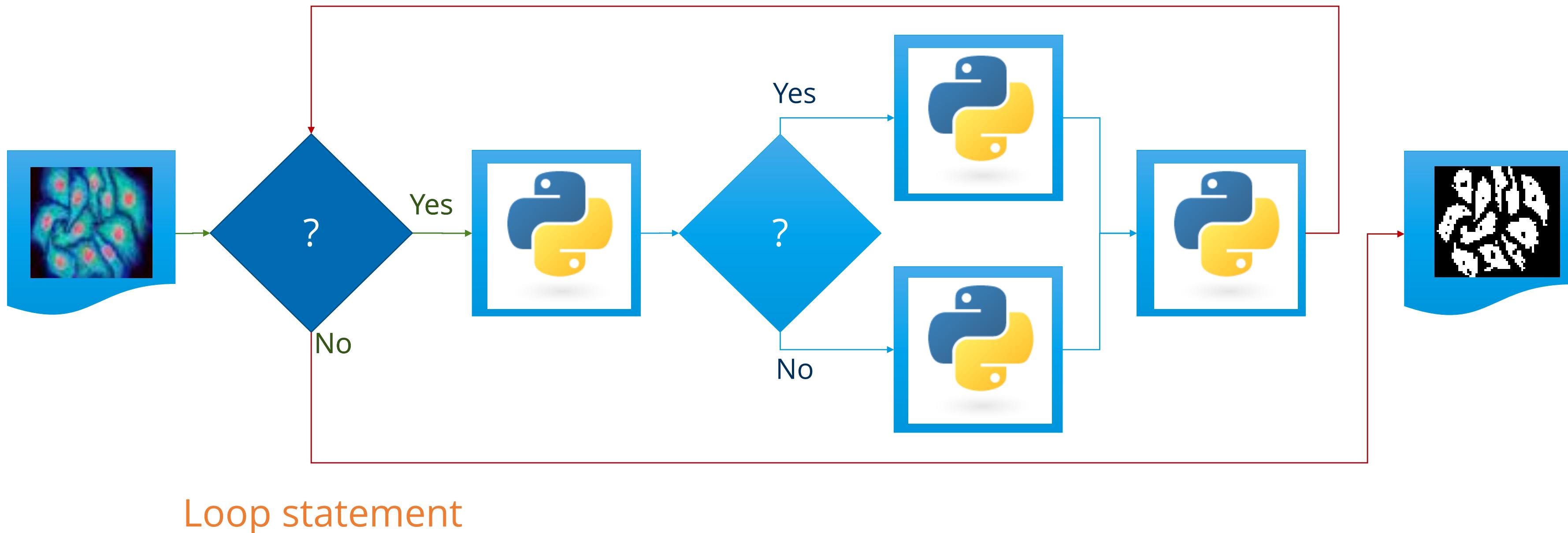
# execute algorithm
d = (a + b) / c

# evaluate result

if a == 5 :
    print("Yin")
else :
    print("Yang")
```

Loops

To repeat actions, you run code in loops



Loops

The `for` statement allows us to execute some lines of code *for* several times,
typically for all items in an array-like thing (lists, tuples, images)

```
# do something  
for <variable> in <array> :  
    # do something repeatedly  
  
# do something
```

Example list : `range(start, stop, step)`

```
# for loops  
for i in range(0, 5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

```
animal_set = ["Cat", "Dog", "Mouse"]  
  
for animal in animal_set:  
    print(animal)
```

Cat
Dog
Mouse

For-loops

- Indentation *means* combining operations to a block

```
▶ # for loops
  for i in range(0, 5):
    print(i)
```

```
File "<ipython-input-15-59c457ae0ac9>", line 3
  print(i)
  ^

```

IndentationError: expected an indented block

Don't forget to indent!

- Colon necessary

```
▶ # for loops
  for i in range(0, 5)
    print(i)
```

```
File "<ipython-input-13-23157c0ed137>", line 2
  for i in range(0, 5)
  ^

```

SyntaxError: invalid syntax

Don't forget the colon!

Generating arrays within for-loops

Also a combination with the if-statement is possible

```
# we start with an empty list
numbers = []

# and add elements
for i in range(0, 5):
    # check if the number is odd
    if i % 2:
        numbers.append(i * 2)

print(numbers)
```

[2, 6]

While-loops

While loops keep executing indented code as long as a condition is met:

```
number = 1024
```

```
while (number > 1):
    number = number / 2
    print(number)
```

```
512.0
256.0
128.0
64.0
32.0
16.0
8.0
4.0
2.0
1.0
```

```
number = 1024
```

```
while (True):
    number = number / 2
    print(number)

    if number < 1:
        break
```

```
512.0
256.0
128.0
64.0
32.0
16.0
8.0
4.0
2.0
1.0
0.5
```

Using the **break** statement, you can leave a loop

The **continue** statement allows to skip iterations

```
for i in range(0, 10):
    if i >= 3 and i <= 6:
        continue
    print(i)
```

```
0
1
2
7
8
9
```

Functions

In case repetitive tasks appear that cannot be handled in a loop, *functions* help:

- re-use code in different contexts
- Indentation is crucial
- Functions must be *defined* before called

Definition

```
def sum_numbers(a, b):  
    result = a + b  
    return result
```

function_name (parameters)

body commands

return statement
(optional)

Call

```
c = sum_numbers(4, 5)  
print(c)
```

9

```
sum_numbers(5, 6)
```

11

```
sum_numbers(3, 4)
```

7

Note: Variables defined inside a function are unknown outside of the function!

Functions

In case repetitive tasks appear that cannot be handled in a loop, *functions* help:

- re-use code in different contexts
- Indentation is crucial
- Functions must be *defined* before called

Definition

```
def sum_numbers(a, b):  
    result = a + b  
    return result
```

Call

```
c = sum_numbers(4, 5)  
print(c)
```

9

Functions

Document your functions to keep track of what they do.

Describe what the functions does and what the parameters are meant to be

```
def square(number):
    """
    Squares a number by multiplying it with itself and returns its result.
    """

    return number * number
```

- You can then later print the *documentation* if you can't recall how a function works.

```
square?
```

Signature: square(number)

Docstring: Squares a number by multiplying it with itself and returns its result.

Image Processing and Filtering

Marcelo Leomil Zoccoler

With material from
Robert Haase, PoL
Mauricio Rocha Martins, Norden lab, MPI CBG
Dominic Waithe, Oxford University
Nuno P Martins, IGC Lisbon
Paulo Aguiar, INEB, Porto
Sebastian Tosi, IRB Barcelona
Benoit Lombardot, Scientific Computing Facility, MPI CBG
Alex Bird, Dan White, MPI CBG

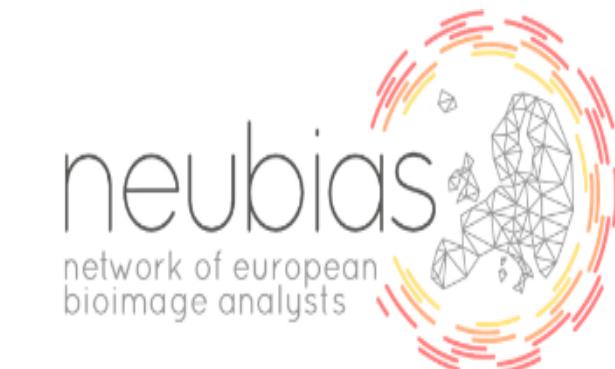


Image Visualization

Marcelo Leomil Zoccoler

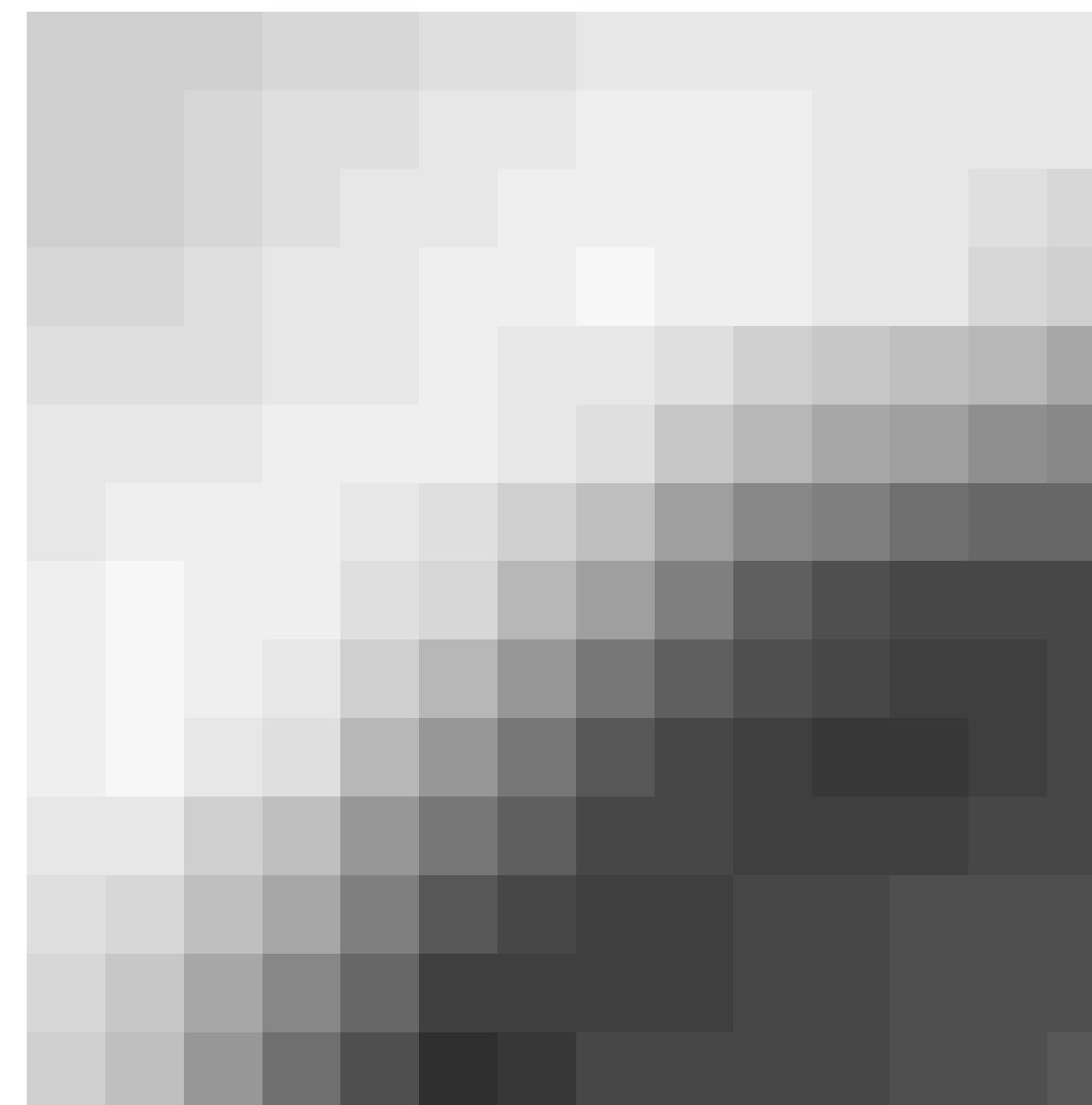
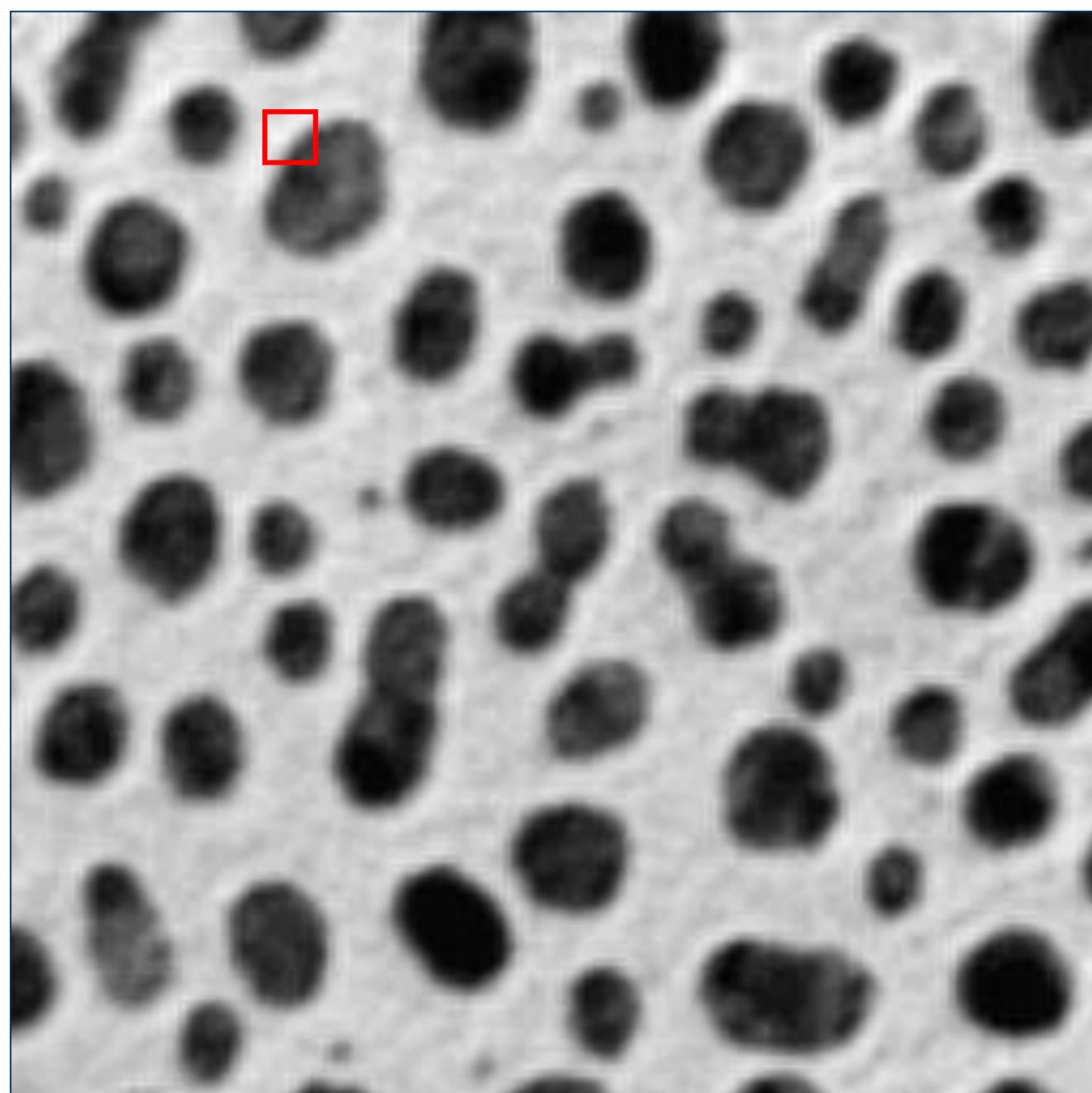
With material from:

Images and pixels

An image is just a matrix of numbers

Pixel: “picture element”

The edges between pixels are an artefact. In reality, they don't exist!

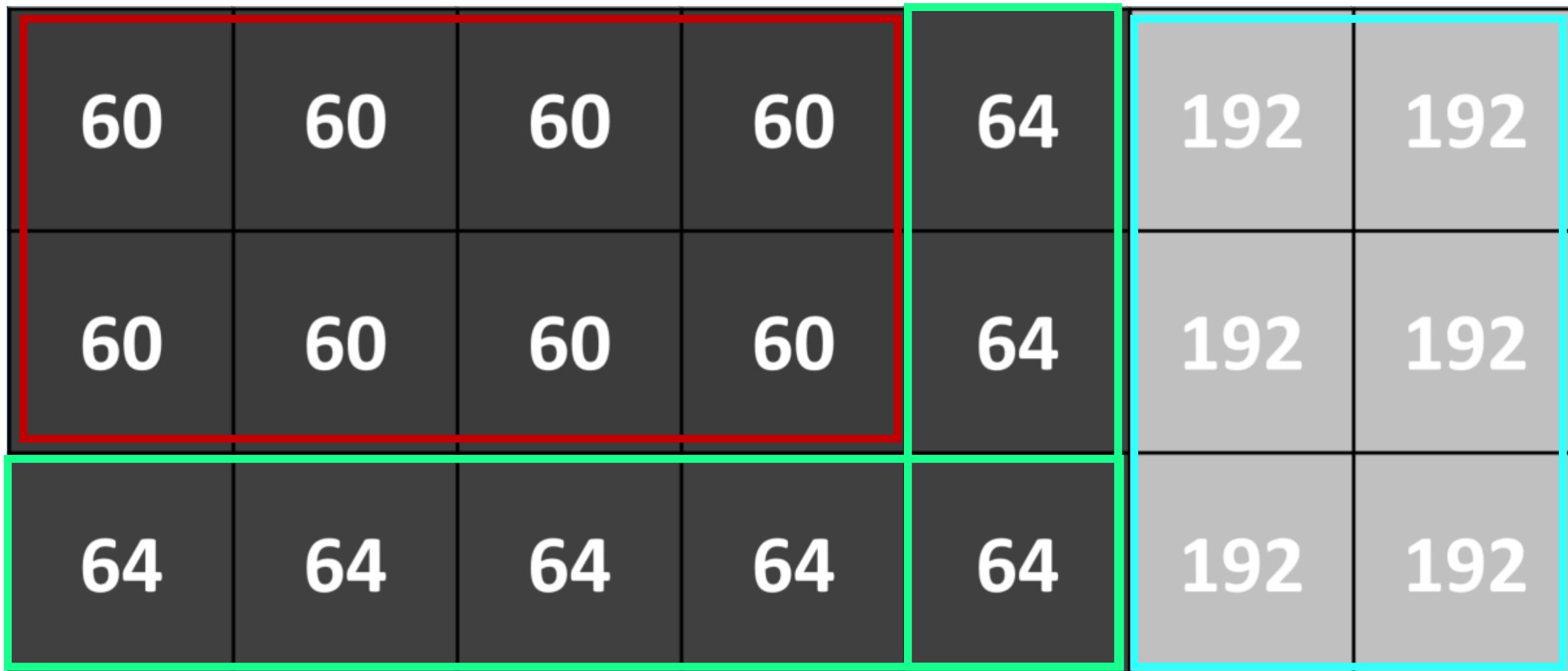


48	48	48	40	40	32	32	24	24	24	24	24	24	24	24
48	48	40	32	32	24	24	16	16	16	16	24	24	24	24
48	48	40	32	24	24	16	16	16	16	16	24	24	32	40
40	40	32	24	24	16	16	8	16	16	16	24	24	40	48
32	32	32	24	24	16	24	24	32	48	56	64	72	88	
24	24	24	16	16	16	24	32	56	72	88	96	112	120	
24	16	16	16	24	32	48	64	96	120	128	144	152	152	
16	8	16	16	32	40	72	96	128	160	176	184	184	184	
16	8	16	24	48	72	104	136	160	176	184	192	192	184	
16	8	24	32	72	104	136	168	184	192	200	200	192	184	
24	24	48	64	104	136	160	184	184	192	192	192	184	184	
32	40	64	88	128	168	184	192	192	184	184	176	176	176	
40	56	88	120	152	192	192	192	184	184	176	176	176	176	
48	64	104	144	176	208	200	184	184	184	176	176	176	168	



Histogram

Consider the following image:



How many elements does it have?

Which pixel value occurs more often?

If we may be mistaken in such a small image,
imagine for big images!

The histogram answers these questions at a glance!

Histogram: a graph that shows how frequent values are in an array.

```
from skimage.exposure import histogram
```

```
pixel_counts, pixel_values = histogram(array)
```

```
plt.bar(pixel_values, pixel_counts)
```

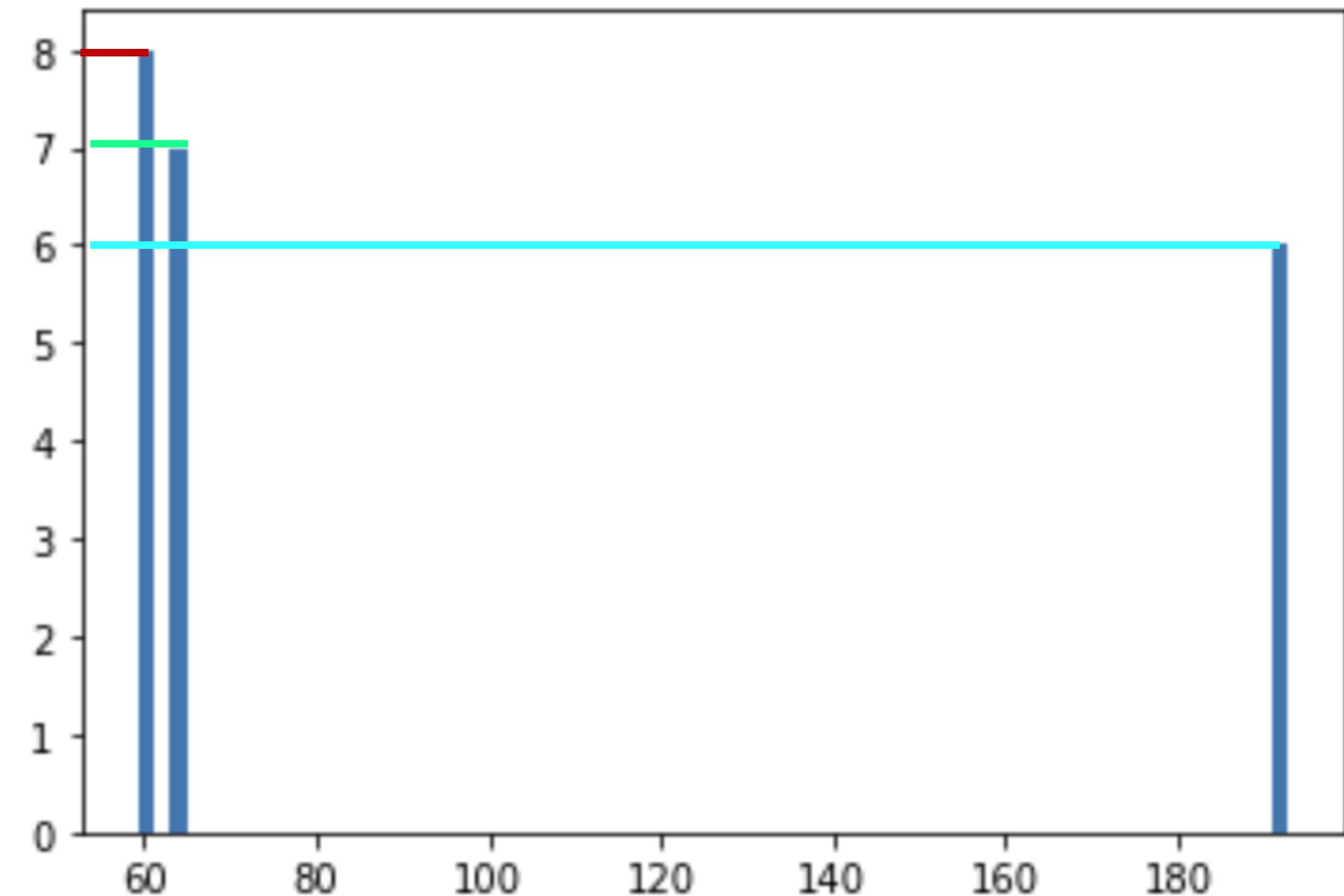
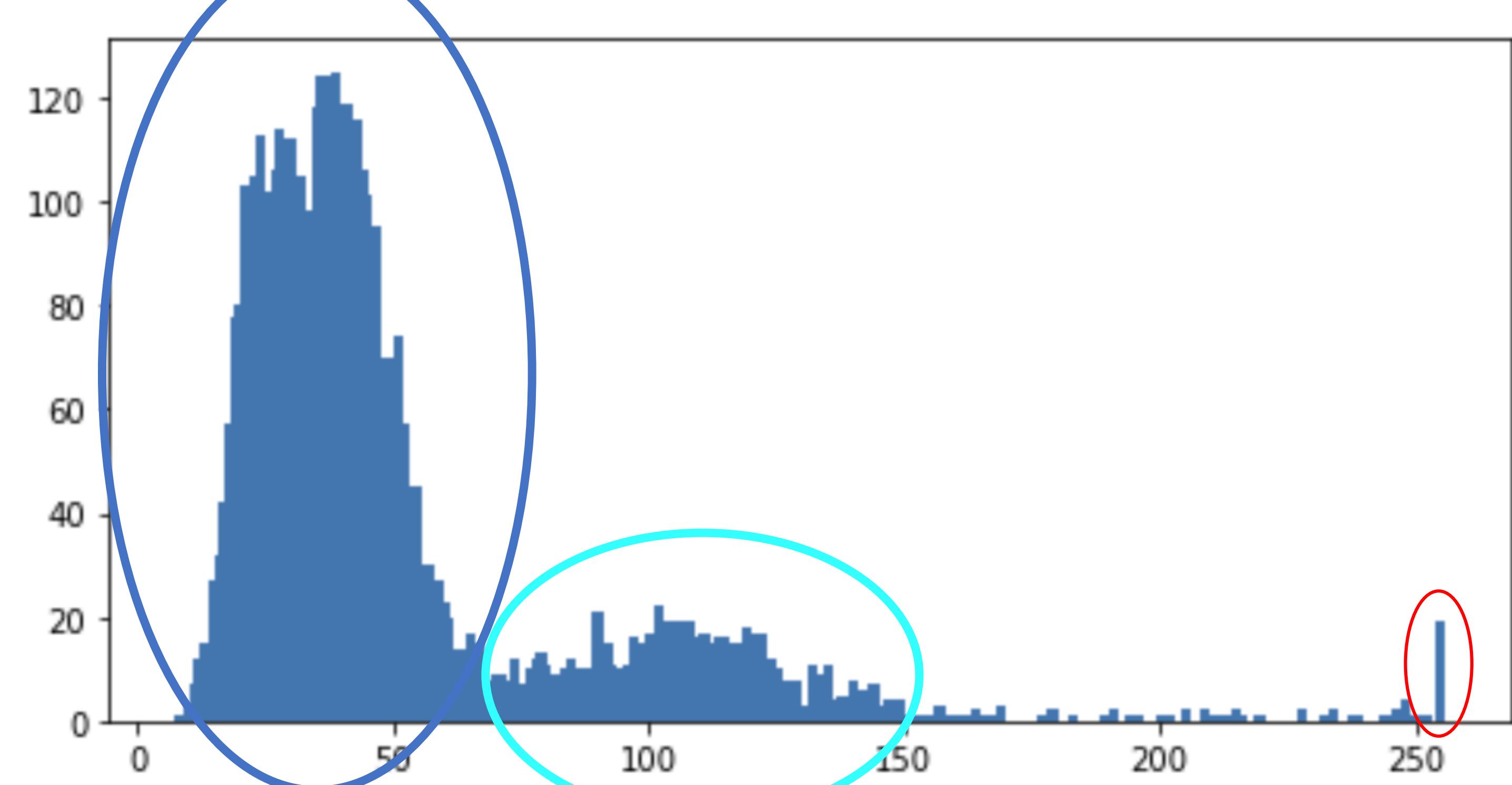
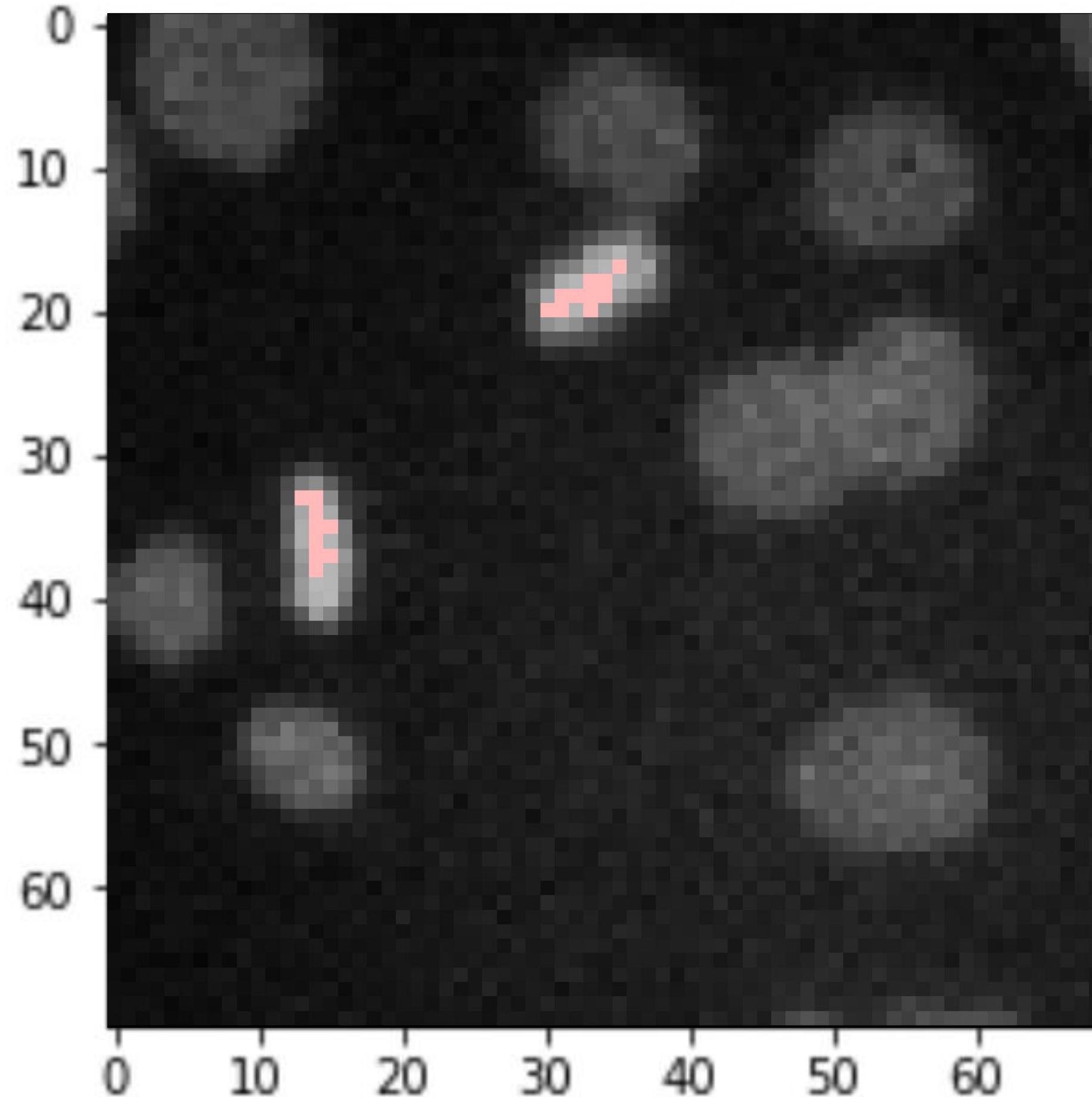


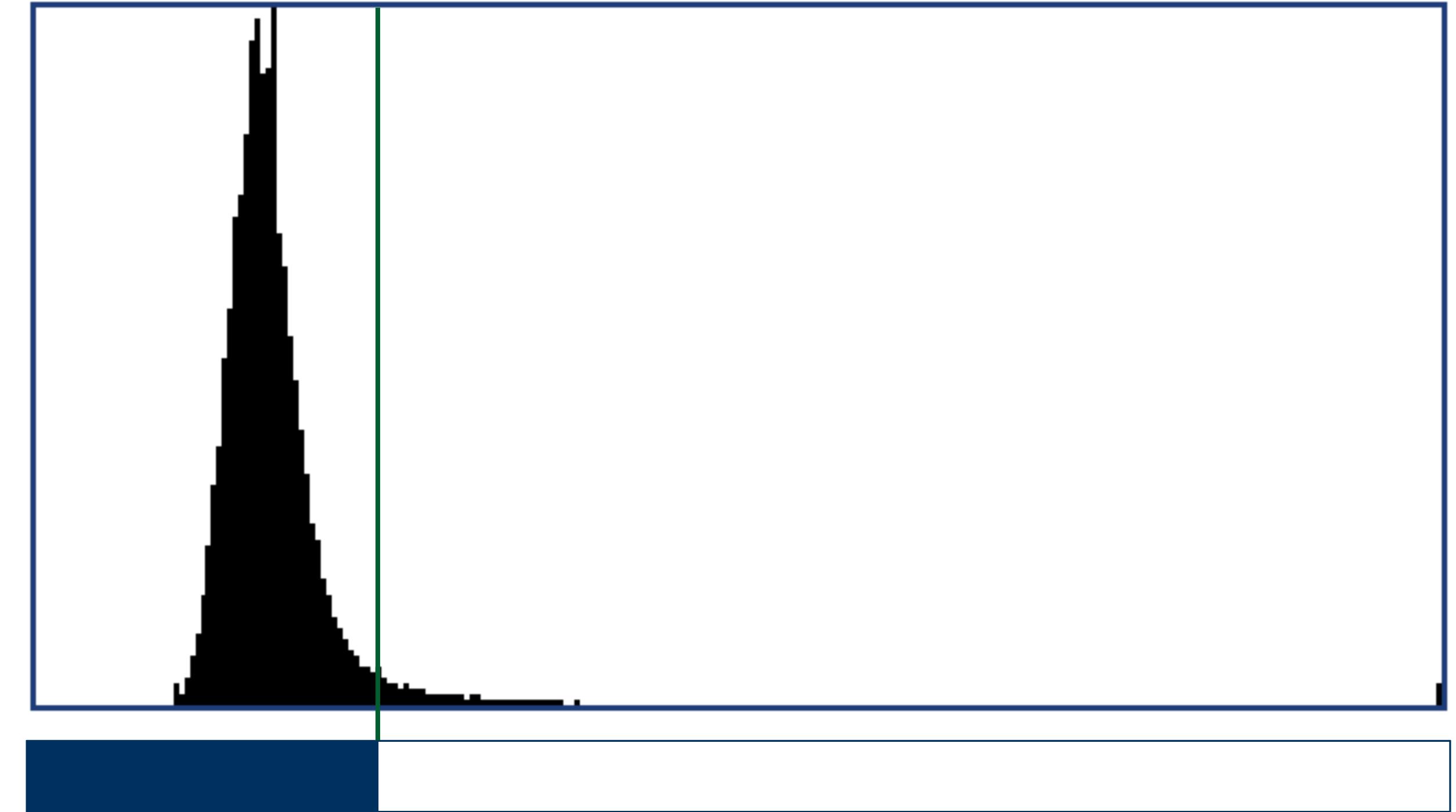
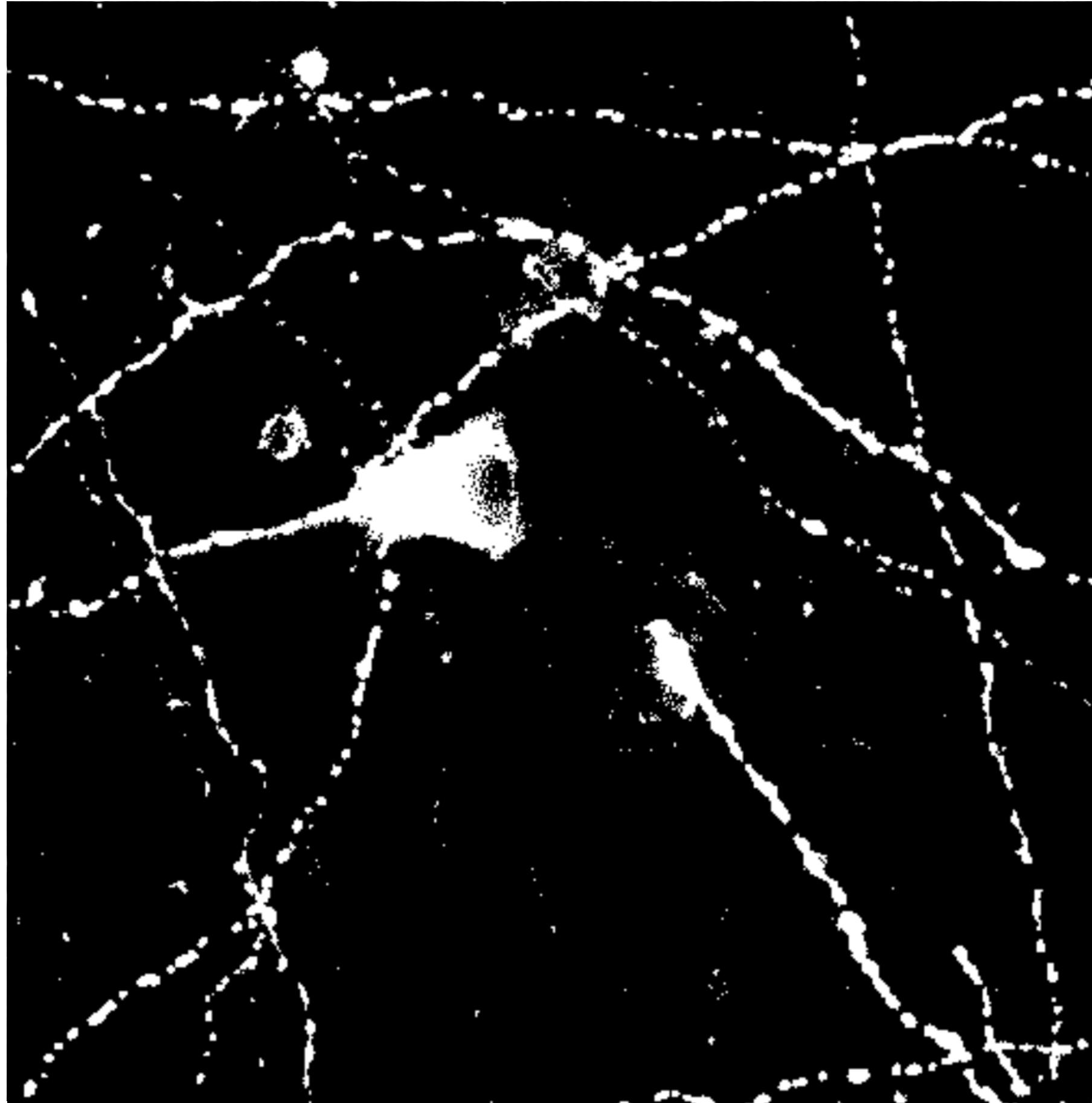
Image Histogram

Example with a slightly larger image:



Binarization: a quick peek

Values above a threshold value are replaced by 1 (or True).
Values below or equal threshold are replaced by 0 (or False)



```
image_binary = image > value
```

Plotting with matplotlib: a quick peek

Use matplotlib to put images side-by-side

```
median_filtered = filters.median(noisy_mri, disk(1))
mean_filtered = filters.rank.mean(noisy_mri, disk(1))
gaussian_filtered = filters.gaussian(noisy_mri, sigma=1)

fig, axs = plt.subplots(2, 3, figsize=(15,10))

# first row
axs[0, 0].imshow(median_filtered)
axs[0, 0].set_title("Median")
axs[0, 1].imshow(mean_filtered)
axs[0, 1].set_title("Mean")
axs[0, 2].imshow(gaussian_filtered)
axs[0, 2].set_title("Gaussian")

# second row
axs[1, 0].imshow(median_filtered[50:100, 50:100])
axs[1, 1].imshow(mean_filtered[50:100, 50:100])
axs[1, 2].imshow(gaussian_filtered[50:100, 50:100])
```

row column

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

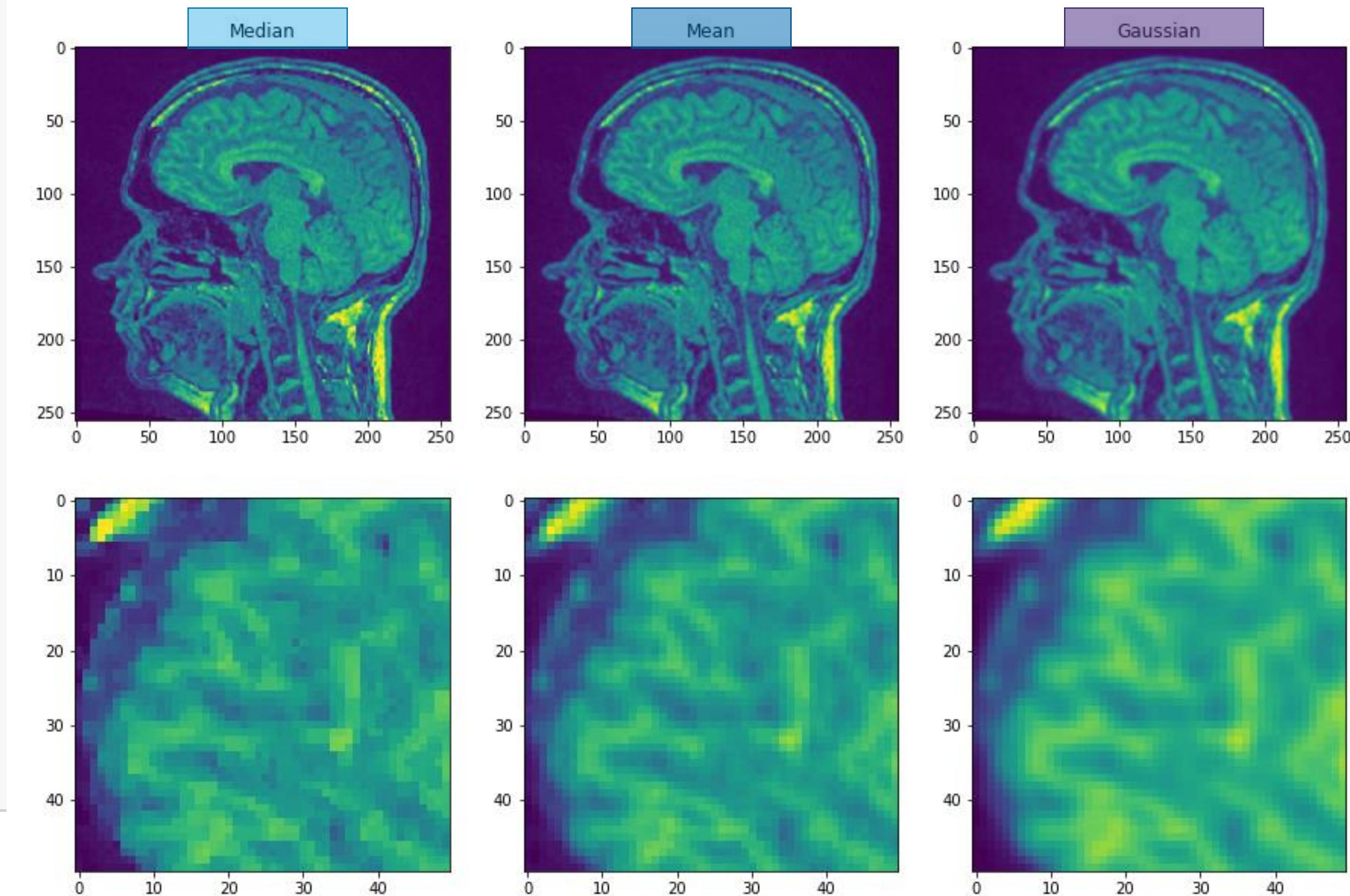


Image Processing: Filters

Marcelo Leomil Zoccoler

With material from:

Robert Haase, PoL

Mauricio Rocha Martins, Norden lab, MPI CBG

Filters

An image processing filter is an operation on an image.

It takes an image and produces a new image out of it.

Filters change pixel values.

There is no “best” filter. Which filter fits your needs, depends on the context.

Filters do not do magic. They can not make things visible which are not in the image.

Application examples

- Noise-reduction
- Artefact-removal
- Contrast enhancement
- Correct uneven illumination

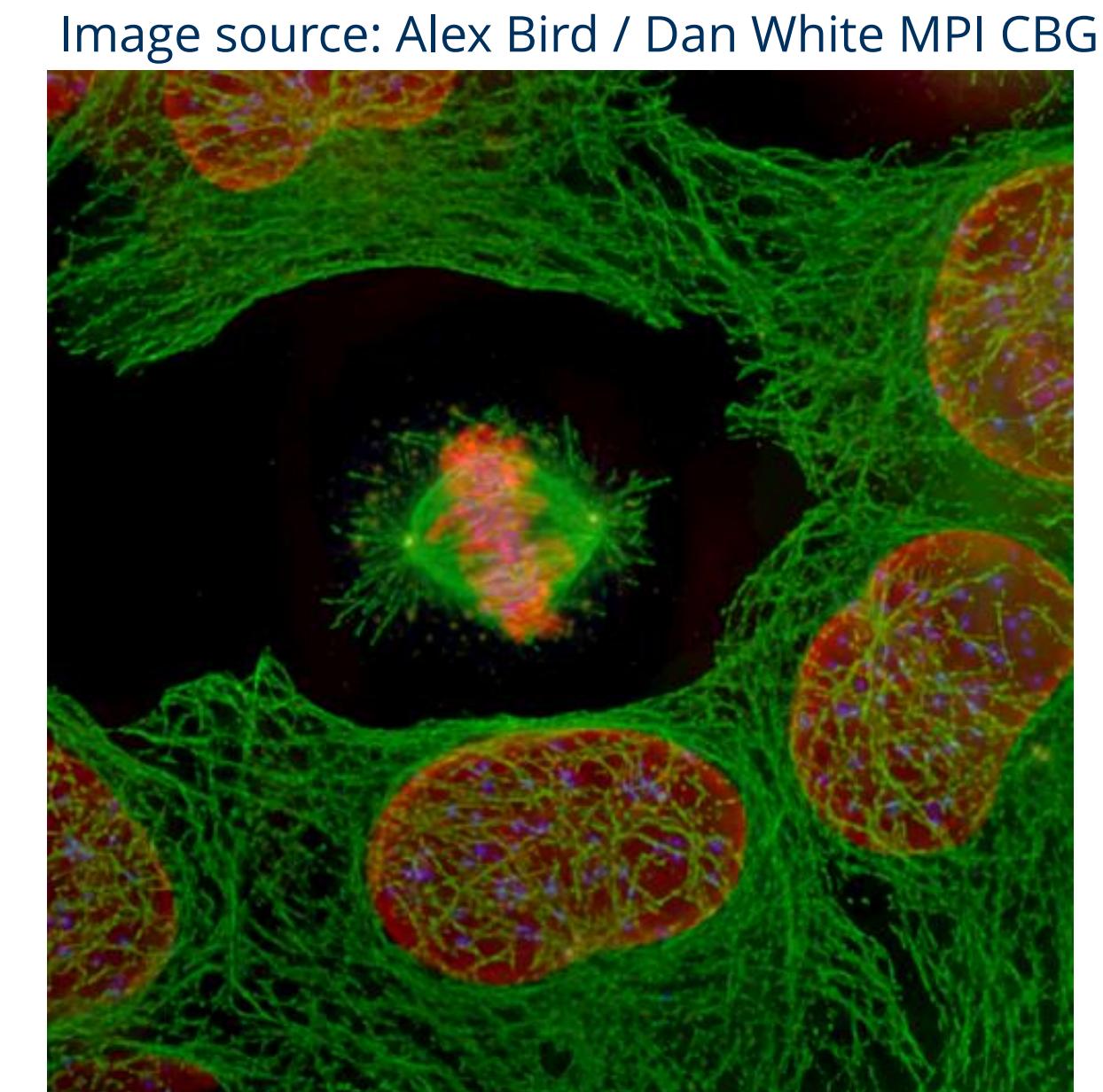
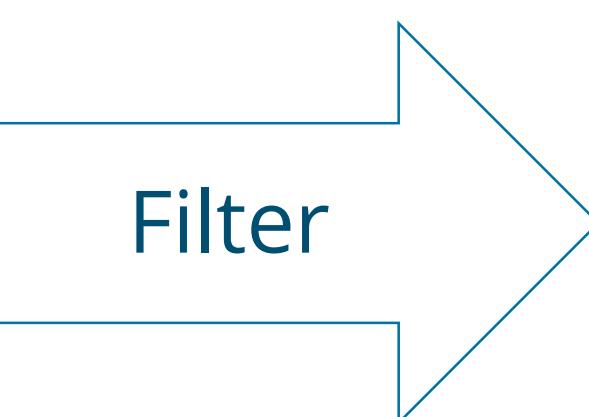
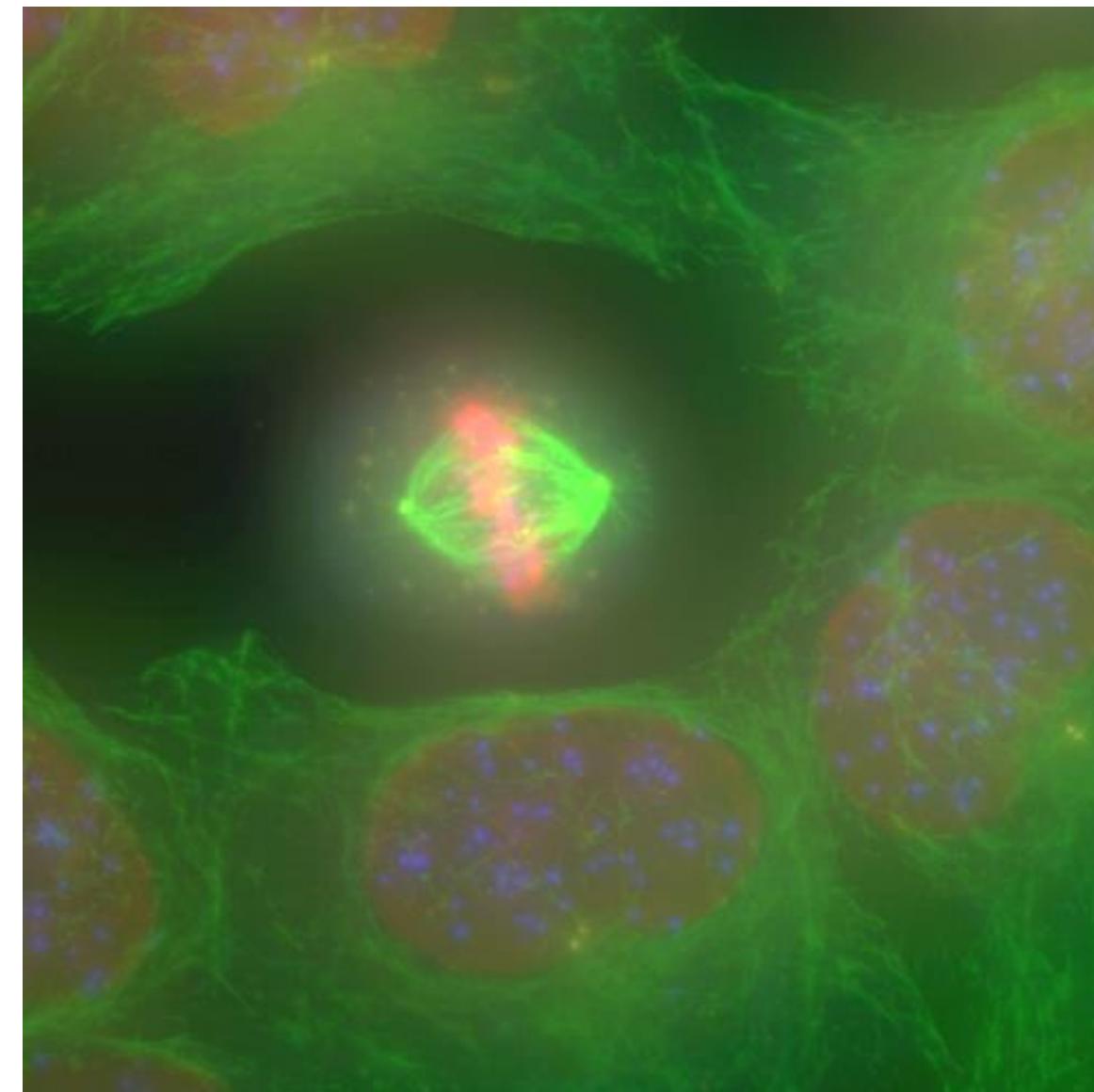


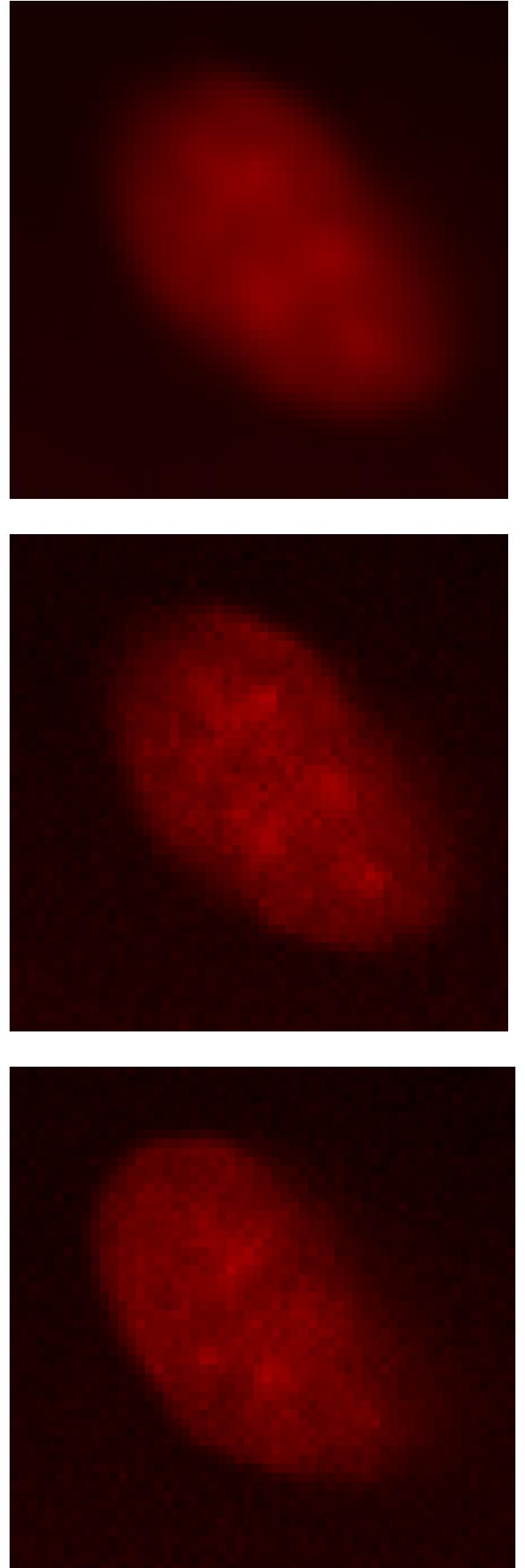
Image source: Alex Bird / Dan White MPI CBG

Effects harming image quality

Noise is a general term for unwanted modifications that a signal may suffer during capture, storage, transmission, processing, or conversion.

In microscopy, image quality suffers from

- shot noise: Statistical variation of the photons arriving at the camera
- dark noise: Statistical variation of how many electrons are generated if a photon arrives in a camera pixel (temperature dependent).
- read out noise: introduced by the electronics, especially the analog-digital-converter
- Physical/optical effects: aberrations, defocus
- Biological/physiological/structural effects: motion, diffusion



Noise

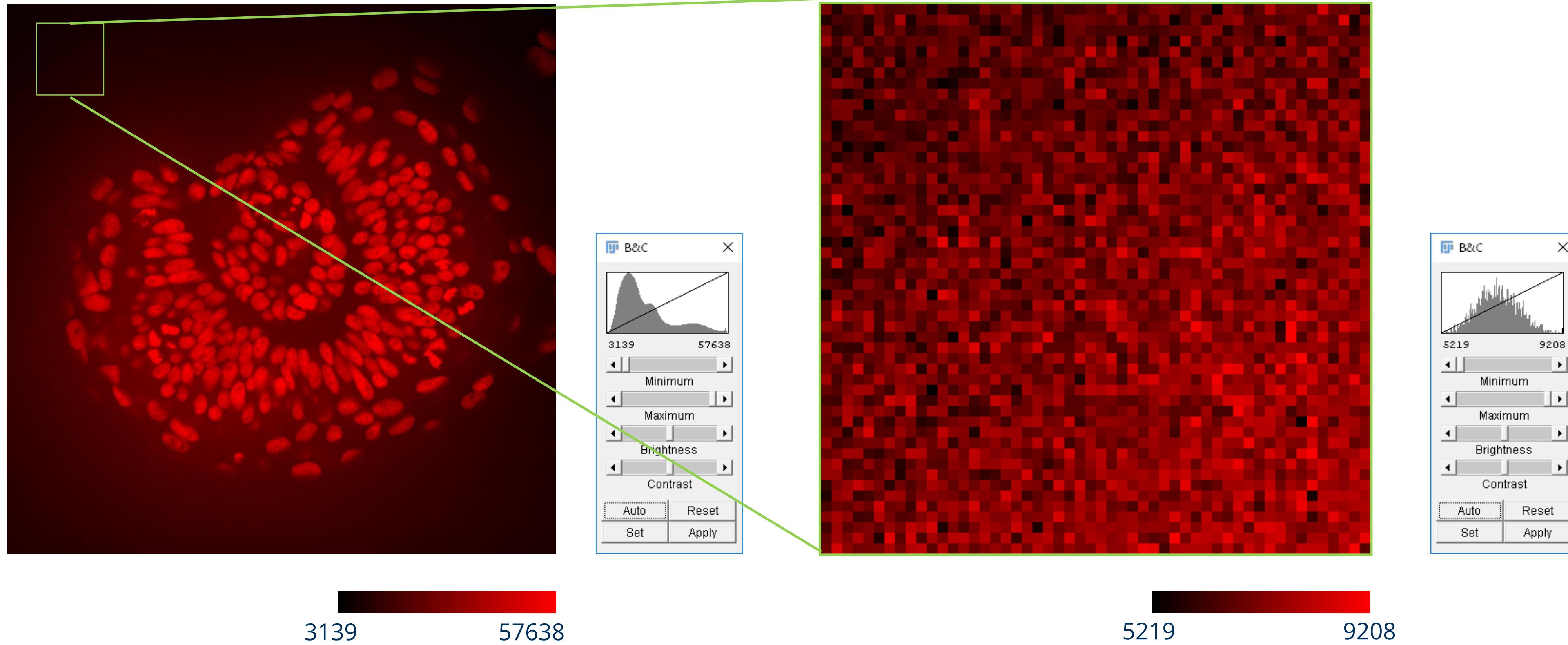
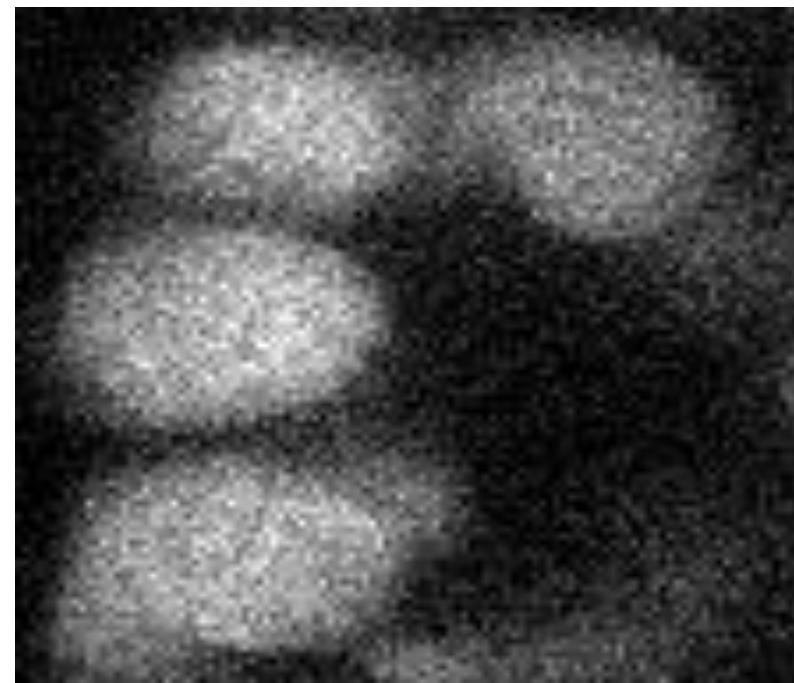


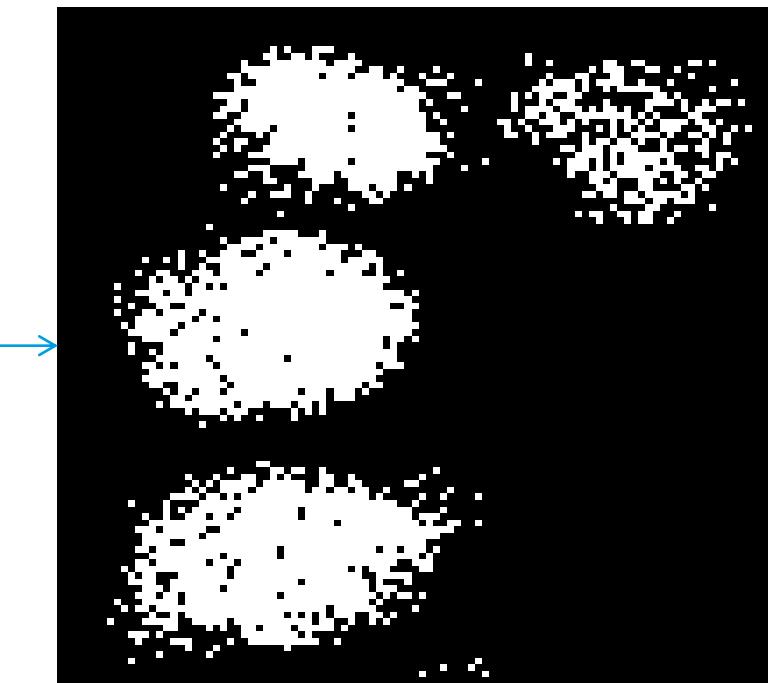
Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

Image correction / noise removal

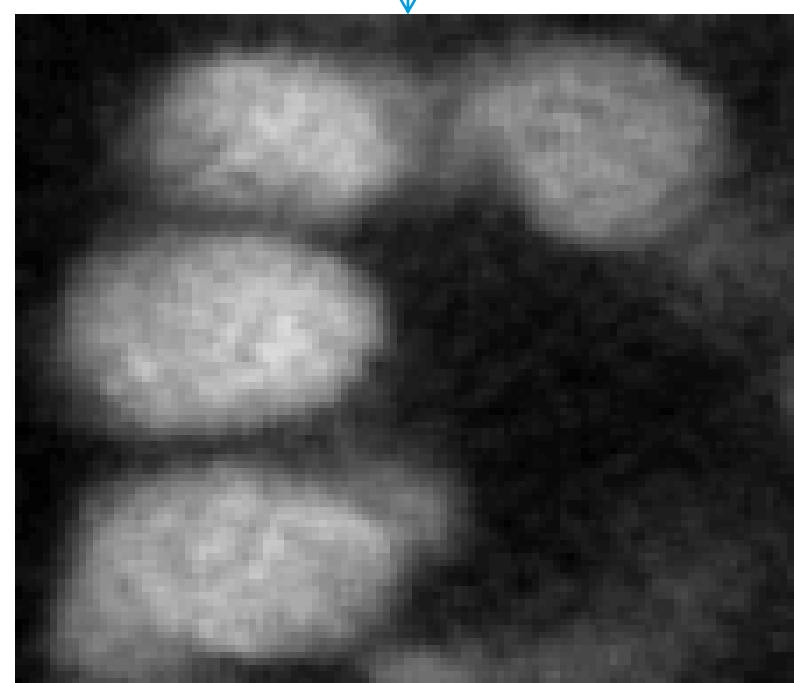
We need to remove the noise to help the computer *interpreting* the image



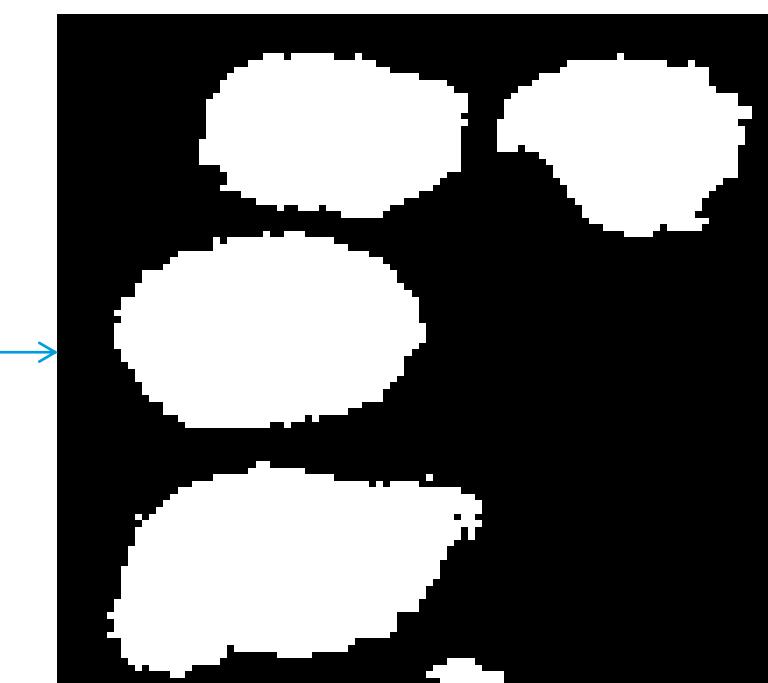
Thresholding



Oh no! I see
thousands of tiny
white objects!



Thresholding

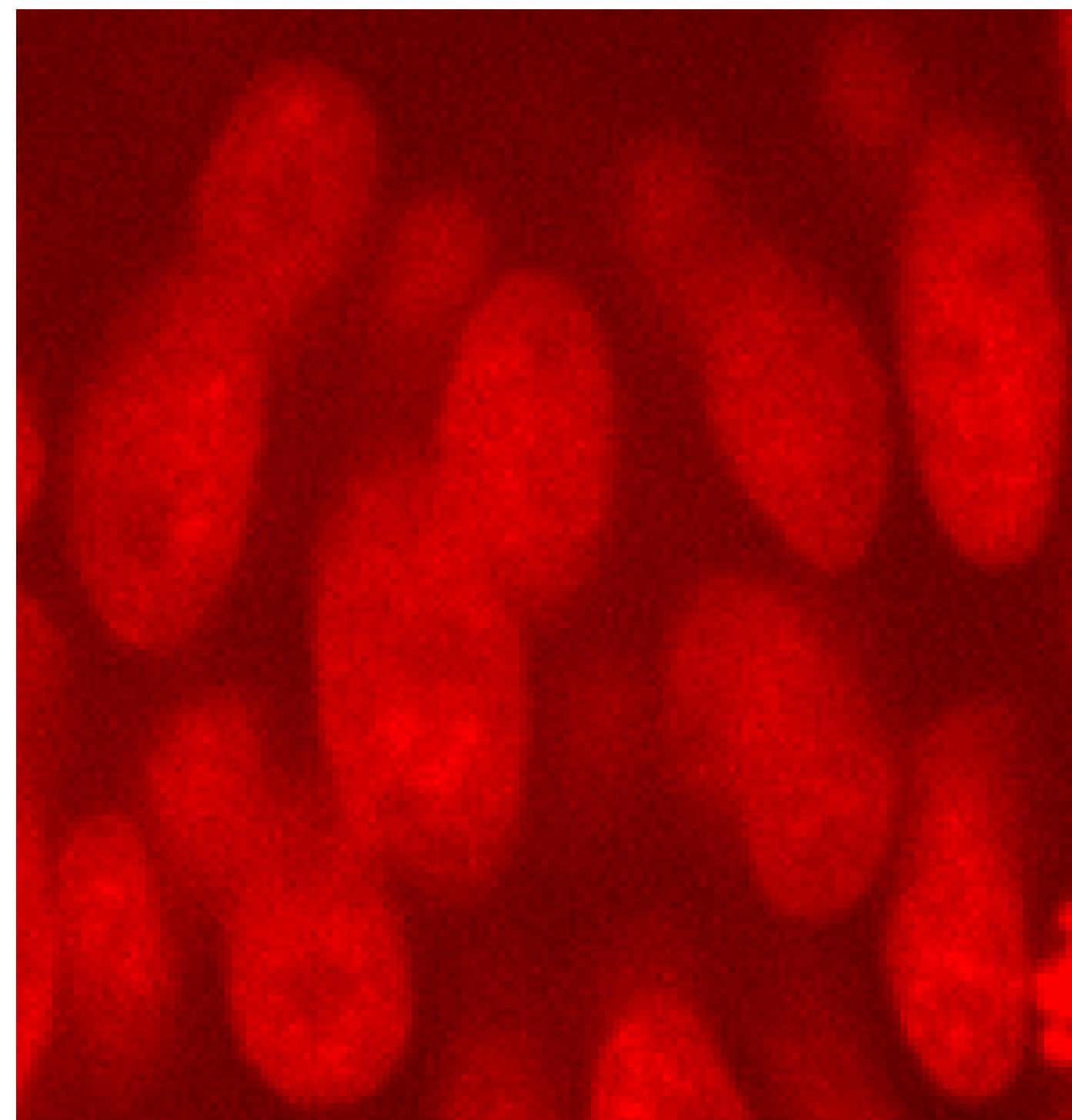


Ok, it's just 4 objects.

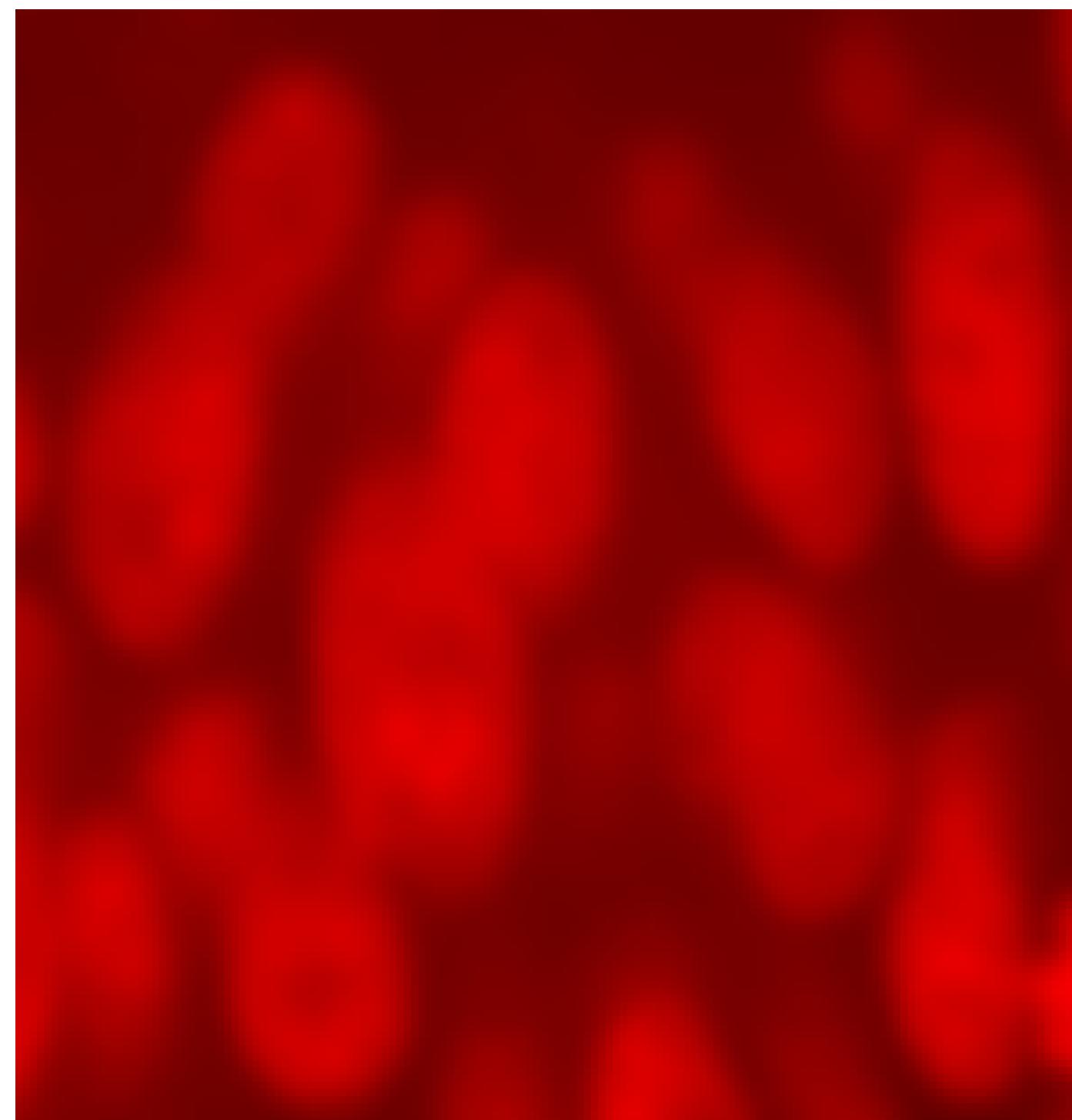


Image correction / noise removal

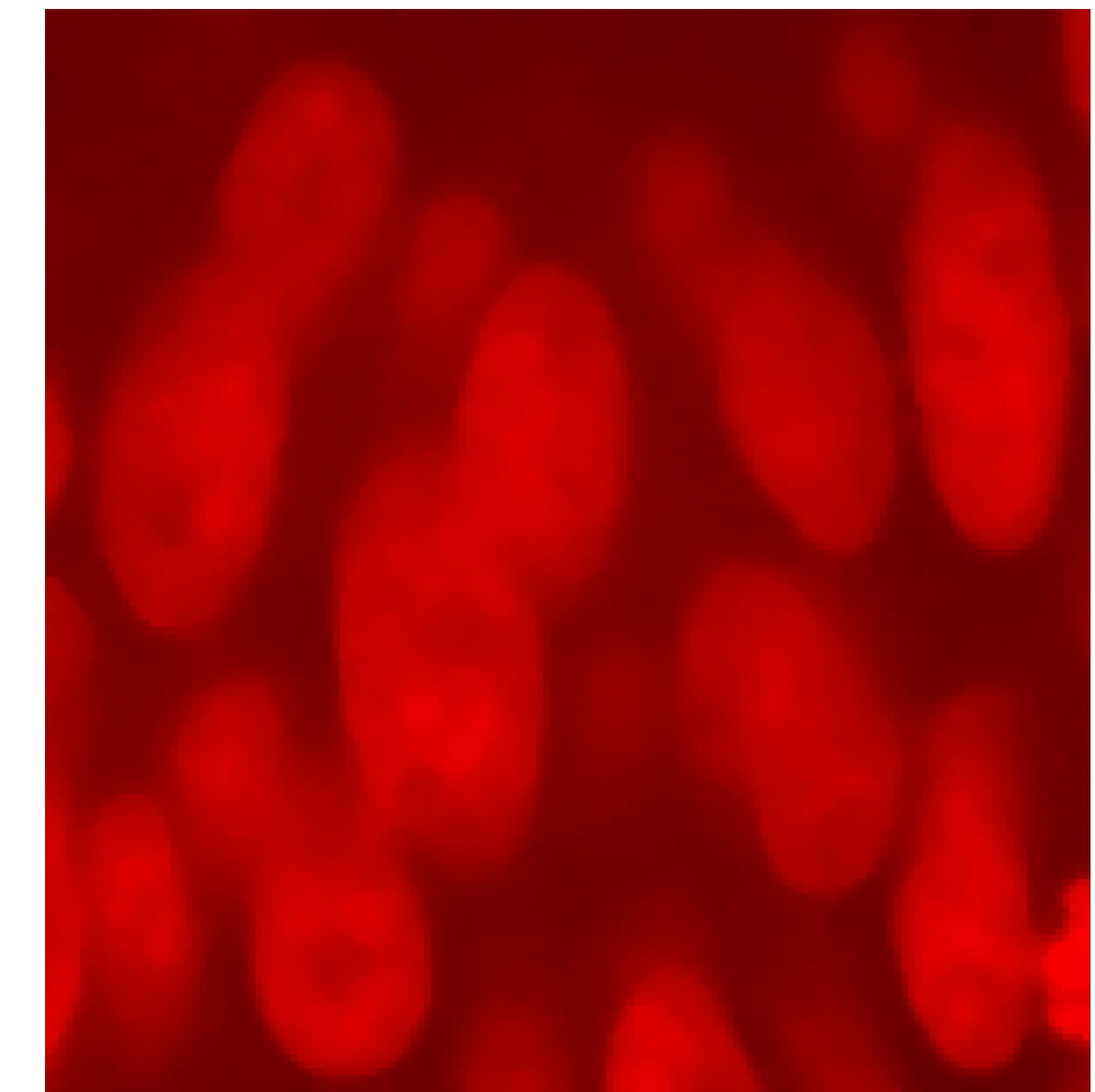
Filters can remove noise



Original image



Gaussian blur filtered



Median filtered

Linear Filters

- Linear filters replace each pixel value with a linear combination of surrounding pixels
- Kernels are matrices describing a linear filter

Mean filter 3x3 kernel =
$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Kernel									
32	48	64	96	120	128	144	152	152	152
40	72	96	128	160	176	184	184	184	184
72	104	136	160	176	184	192	192	184	184
104	136	168	184	192	200	200	192	184	184
136	160	184	184	192	192	192	184	184	184
168	184	192	192	184	184	176	176	176	176
192	192	192	192	184	184	176	176	176	176
208	200	184	184	184	184	176	176	176	168

$$\text{New_value} = (1/9)*32 + (1/9)*48 + (1/9)*64 + (1/9)*40 + (1/9)*72 + (1/9)*96 + (1/9)*72 + (1/9)*104 + (1/9)*136$$

$$\text{New_value} = 73.8$$

What operation is this?

Linear Filters

- Linear filters replace each pixel value with a linear combination of surrounding pixels
- Kernels are matrices describing a linear filter

Mean filter 3x3 kernel =
$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Kernel									
32	48	64	96	120	128	144	152	152	152
40	72	96	128	160	176	184	184	184	184
72	104	136	160	176	184	192	192	184	184
104	136	168	184	192	200	200	192	184	184
136	160	184	184	192	192	192	184	184	184
168	184	192	192	184	184	176	176	176	176
192	192	192	192	184	184	176	176	176	176
208	200	184	184	184	184	184	176	176	168

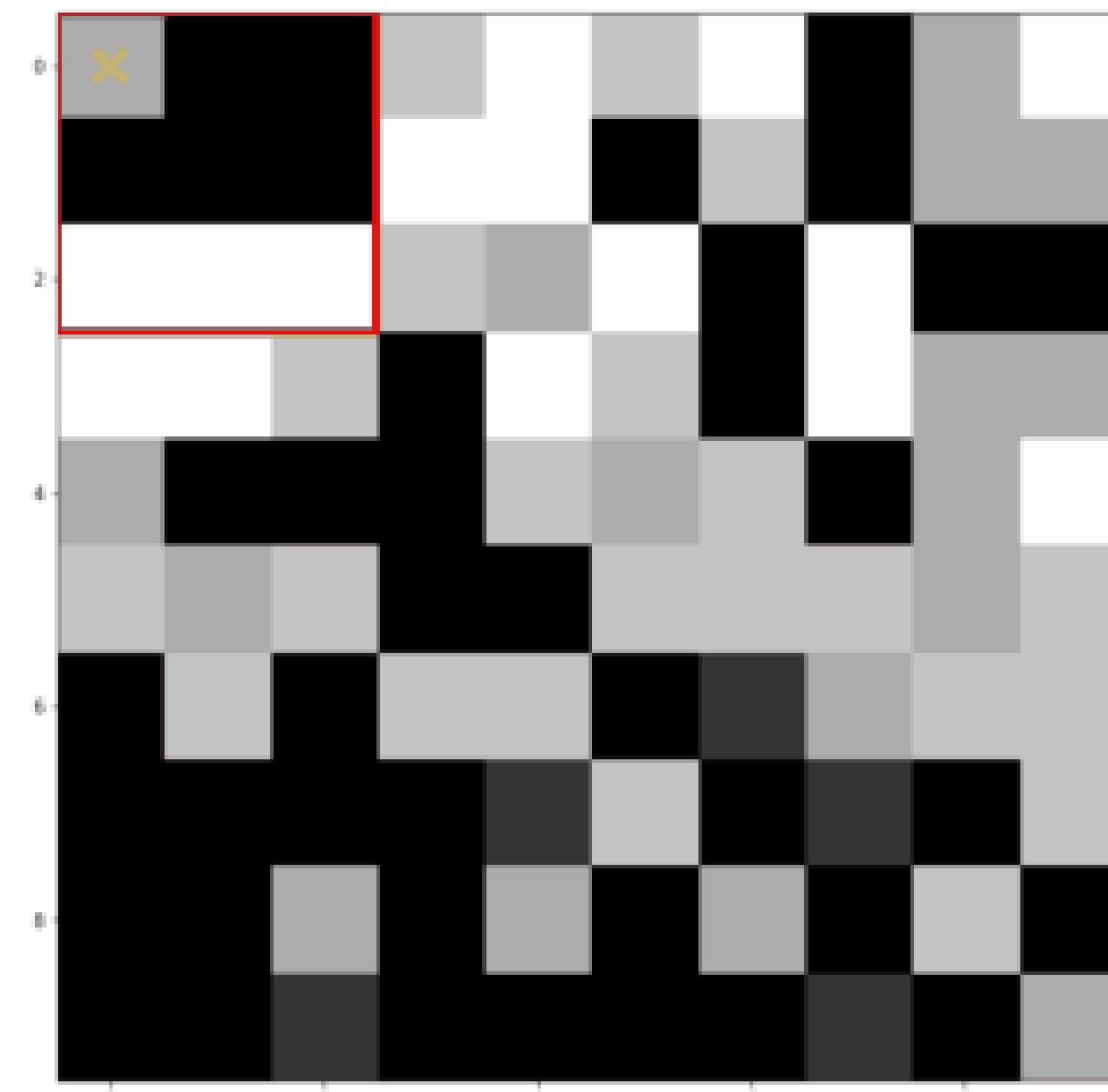
$$\text{New_value} = (1/9)*48 + (1/9)*64 + (1/9)*96 + (1/9)*72 + (1/9)*96 + (1/9)*128 + (1/9)*104 + (1/9)*136 + (1/9)*160$$

$$\text{New_value} = 100.4$$

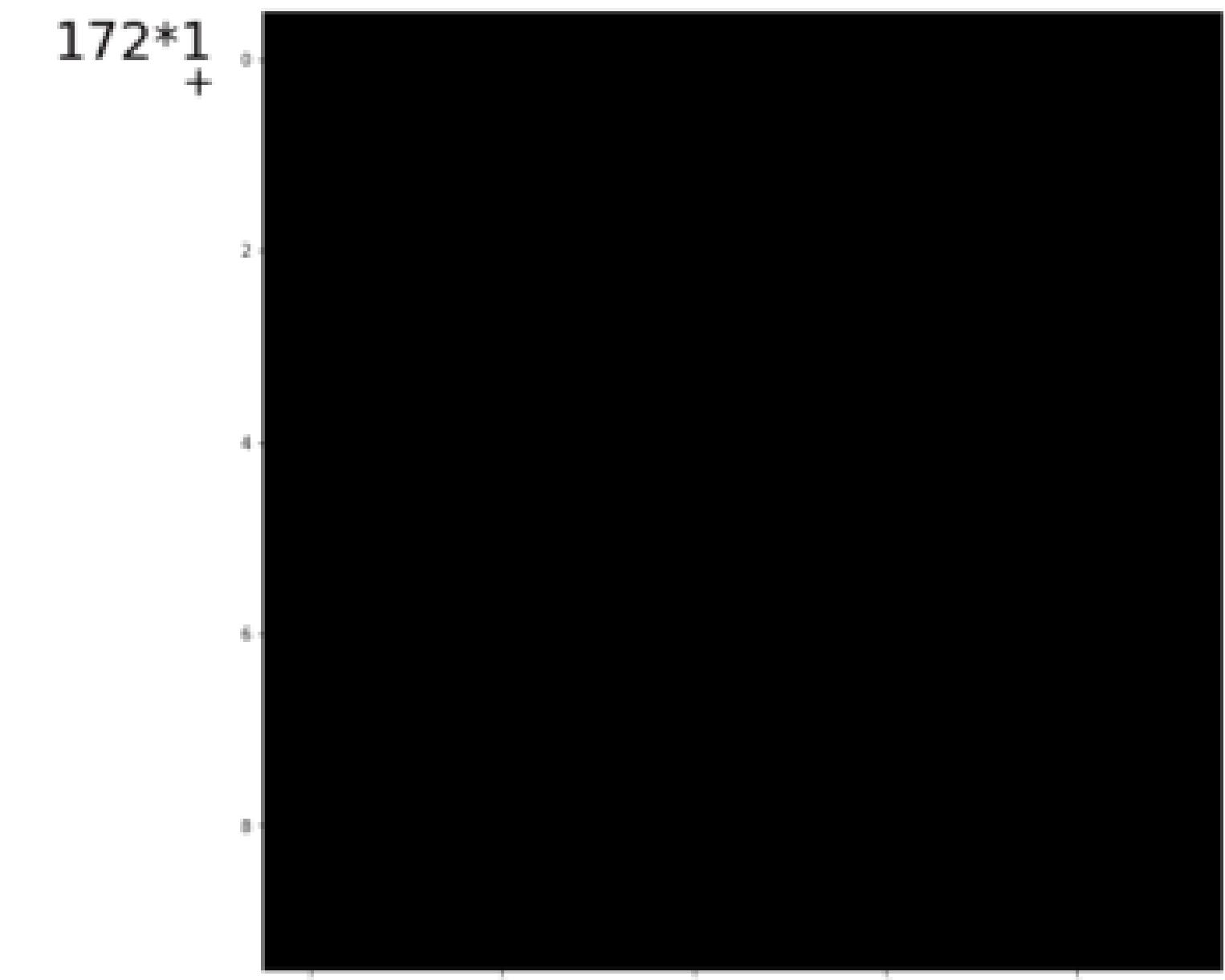
What operation is this?

Linear Filters

- Basically, linear filtering is a Convolution
- It needs a kernel (weight template)
- Result: new image where each pixel is replaced by the weighted sum of pixel values in the neighborhood



Image



Output_image

Terminology:

- “We convolve an image with a kernel.”
- Convolution operator: *

$$\text{Output_image} = \text{Image} * \text{Kernel}$$

Linear filters

Different linear filters use different kernels

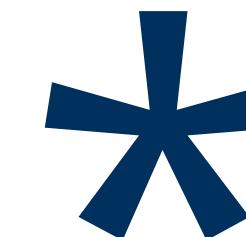
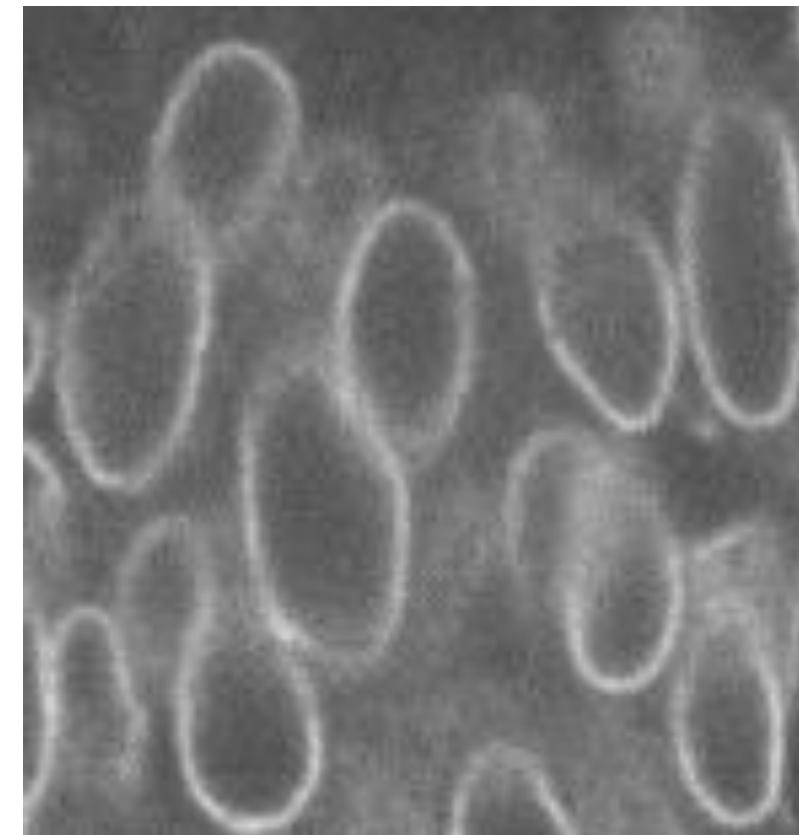
Other Examples

- Mean

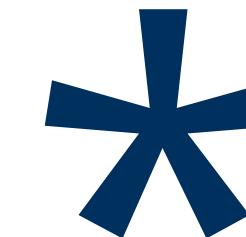
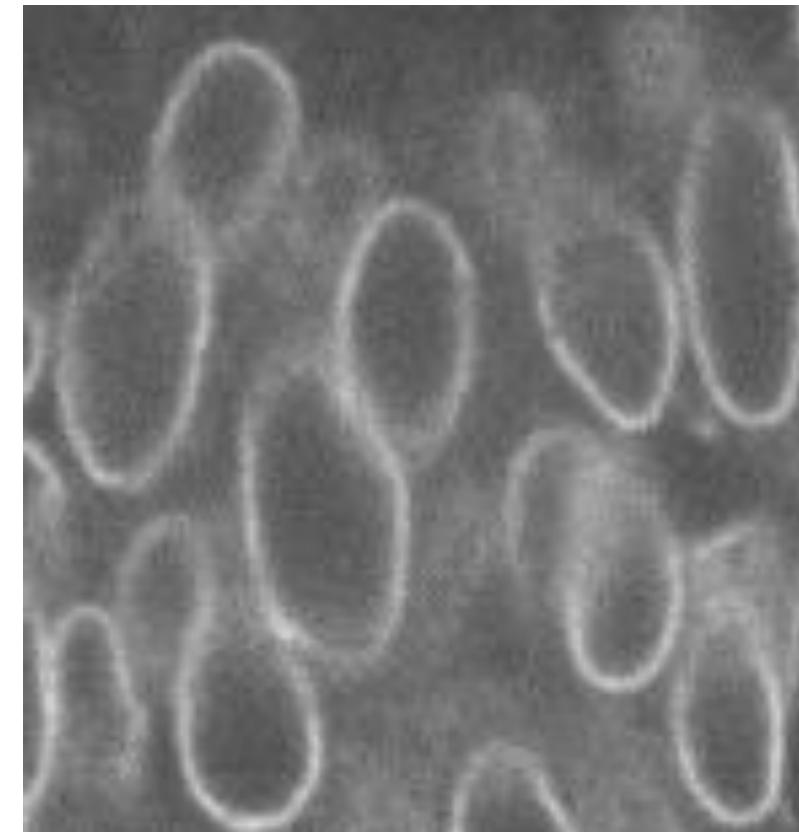
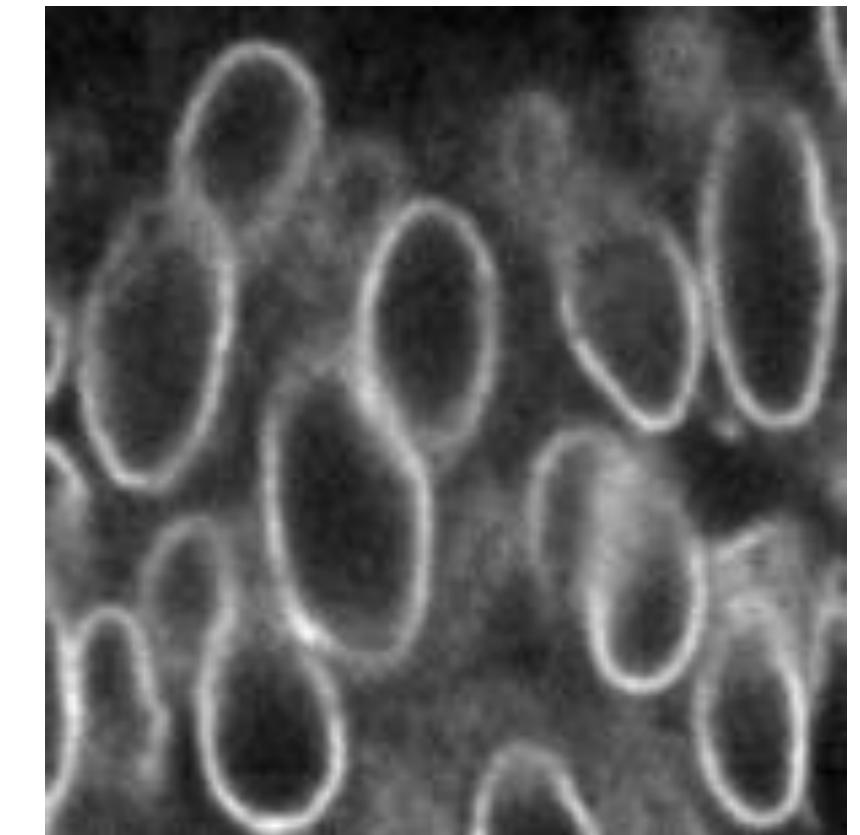
- Gaussian blur

- Sobel-operator

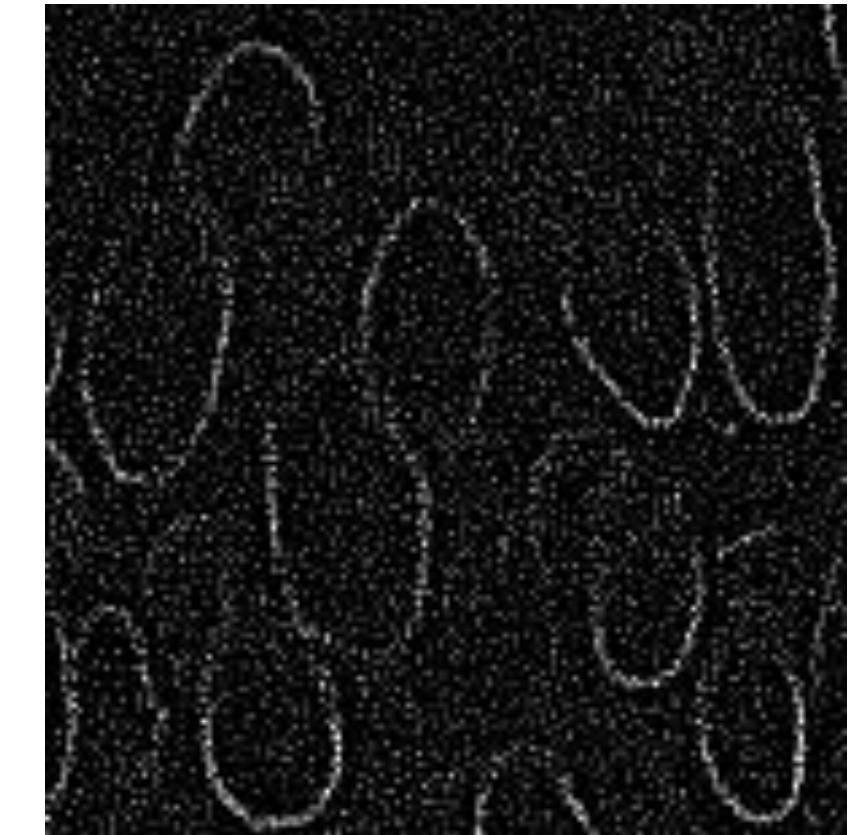
- Laplace-filter



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

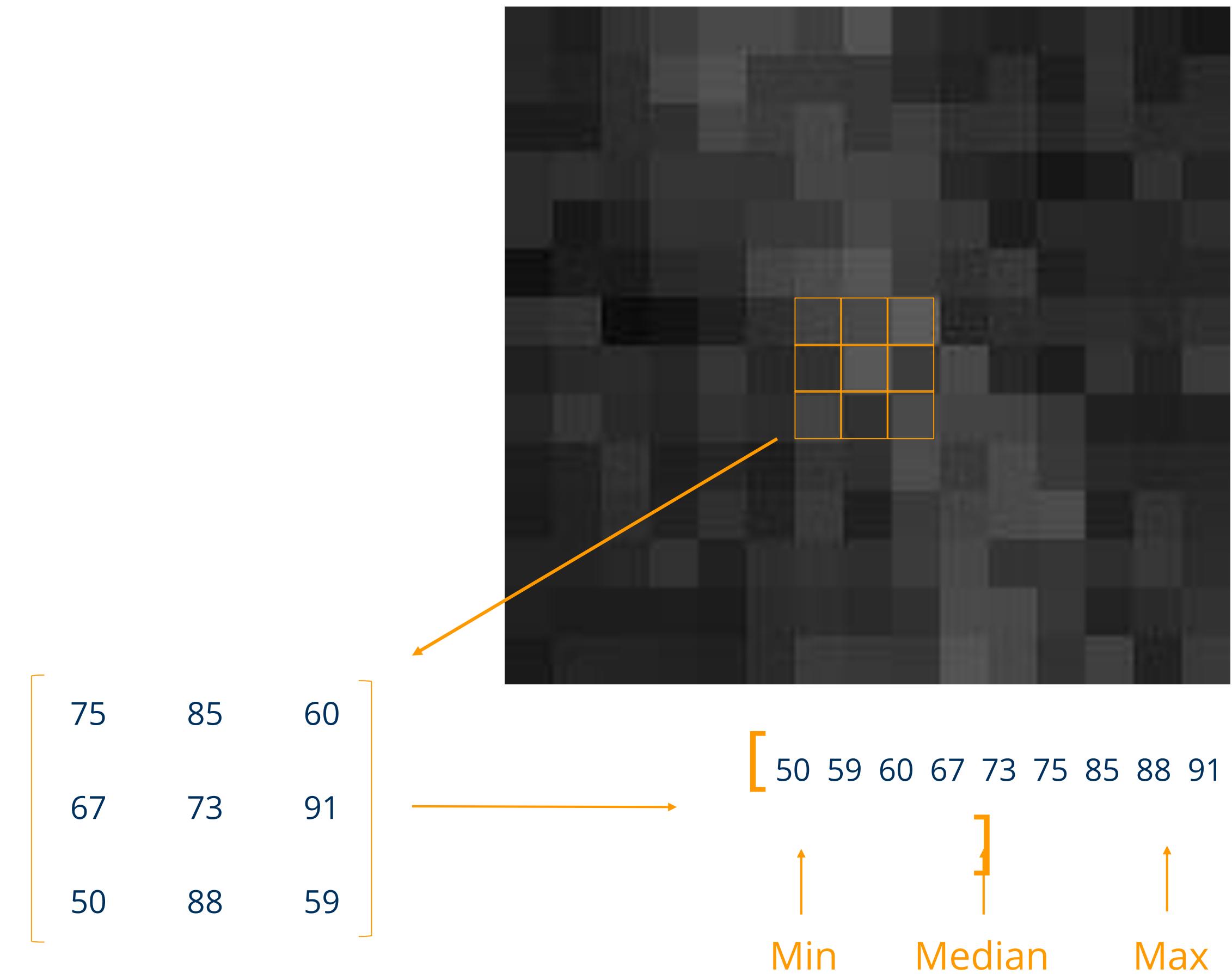


Nonlinear Filters

- Nonlinear filters also replace pixel value inside as rolling window but in a non linear function.

Examples: order statistics filters

- Min
- Median
- Max
- Variance
- Standard deviation



Filters

In python, for linear filters, we could apply a kernel to an image like this:

```
kernel = np.array(  
    [[1/9, 1/9, 1/9],  
     [1/9, 1/9, 1/9],  
     [1/9, 1/9, 1/9]])
```

```
from scipy.ndimage import convolve  
  
output = convolve(image, kernel)
```

But scikit-image already has several of these filters implemented and optimized.

```
from skimage.filters import gaussian  
  
output = gaussian(image, sigma = 1)
```

Then there is no need to provide a kernel. And you can also directly apply non-linear filters.

```
from skimage.filters import median  
  
output = median(image)
```

Available filters can be searched like this:

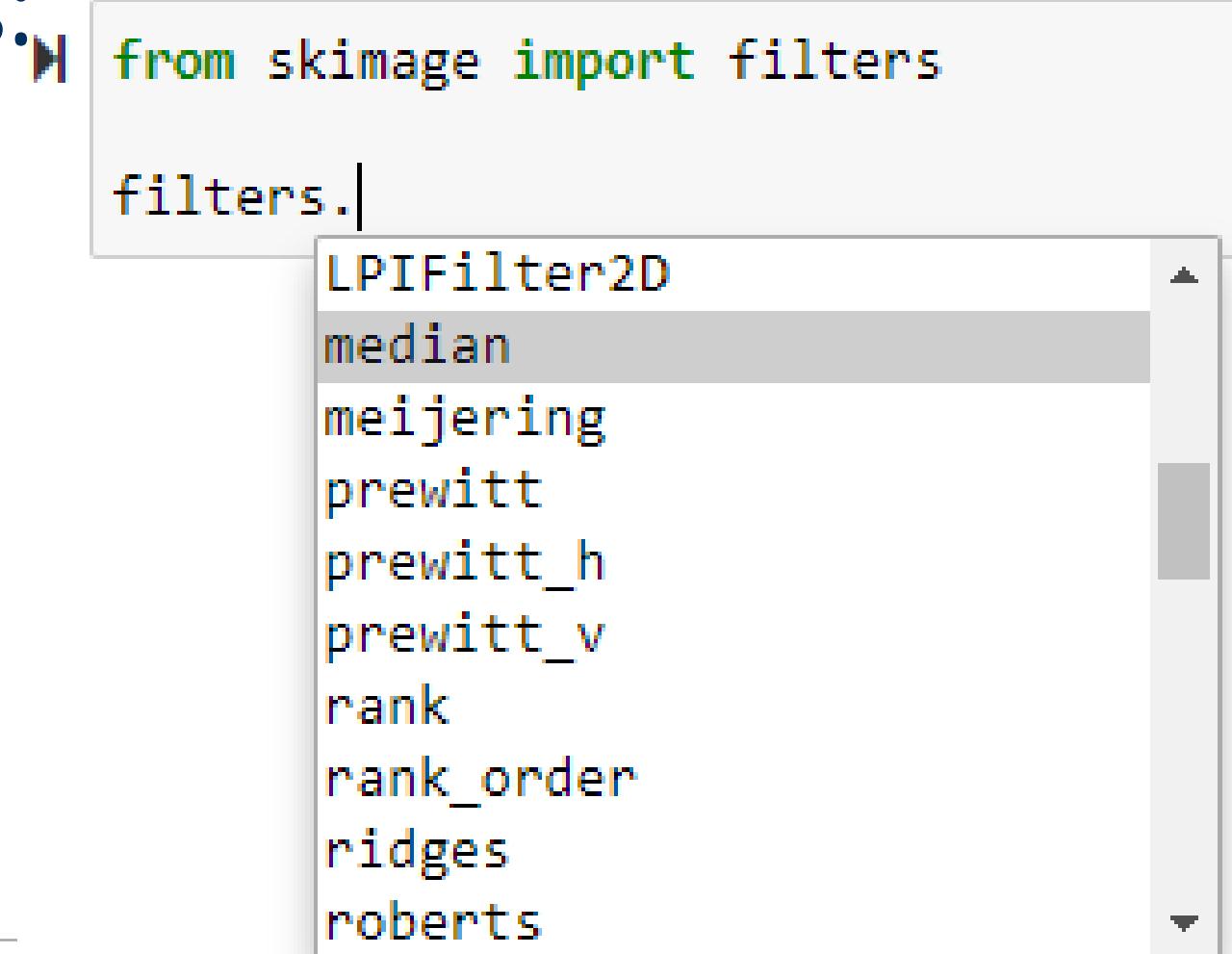


Image Processing: Morphological Operations

Marcelo Leomil Zoccoler

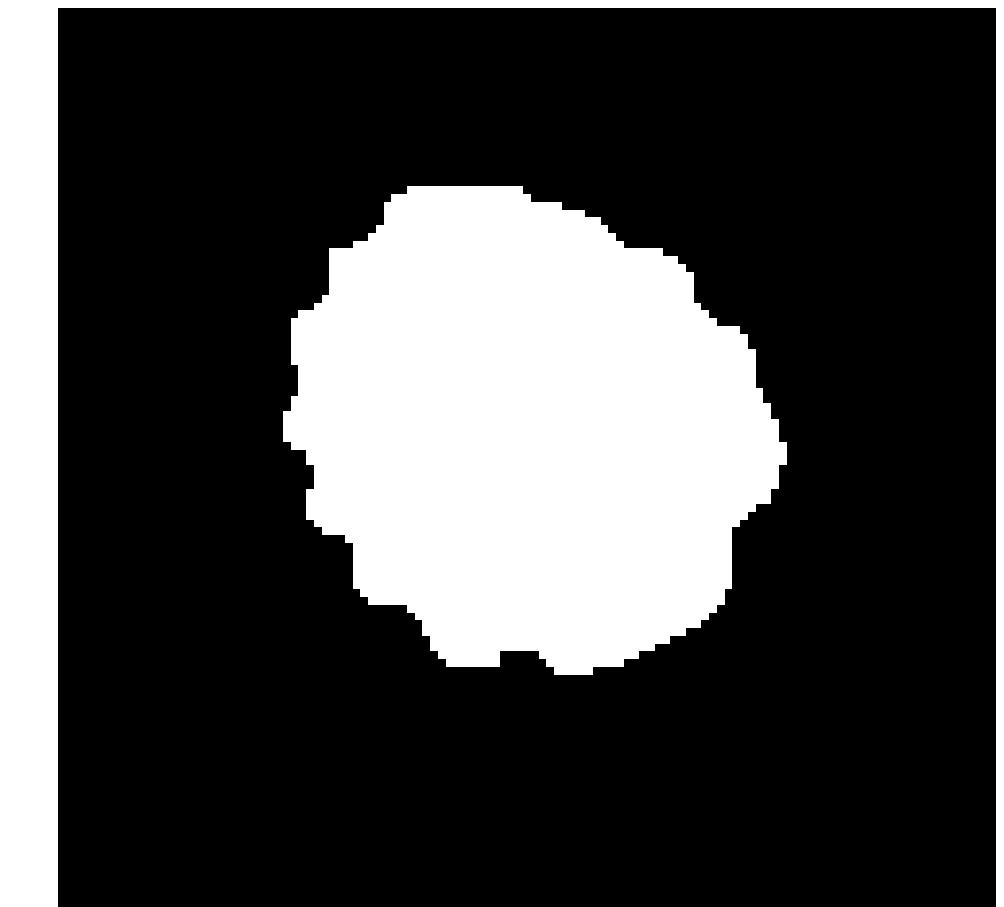
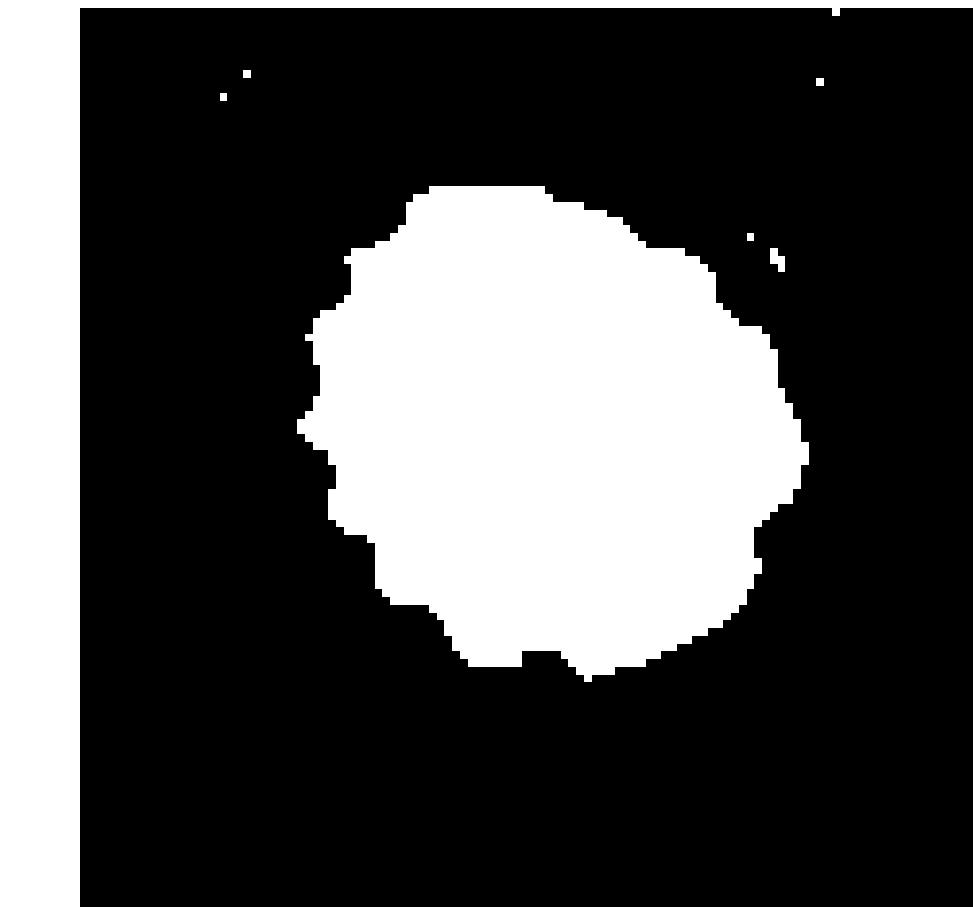
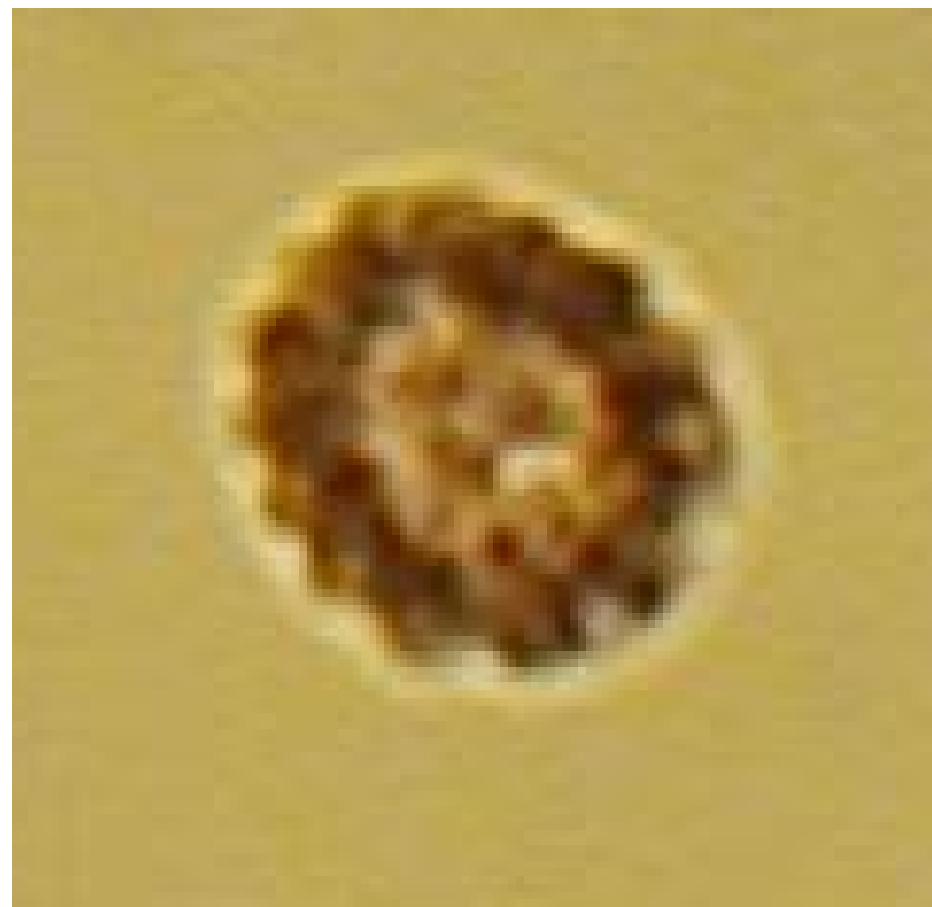
With material from
Robert Haase, PoL

Mauricio Rocha Martins, Norden lab, MPI CBG

Refining masks

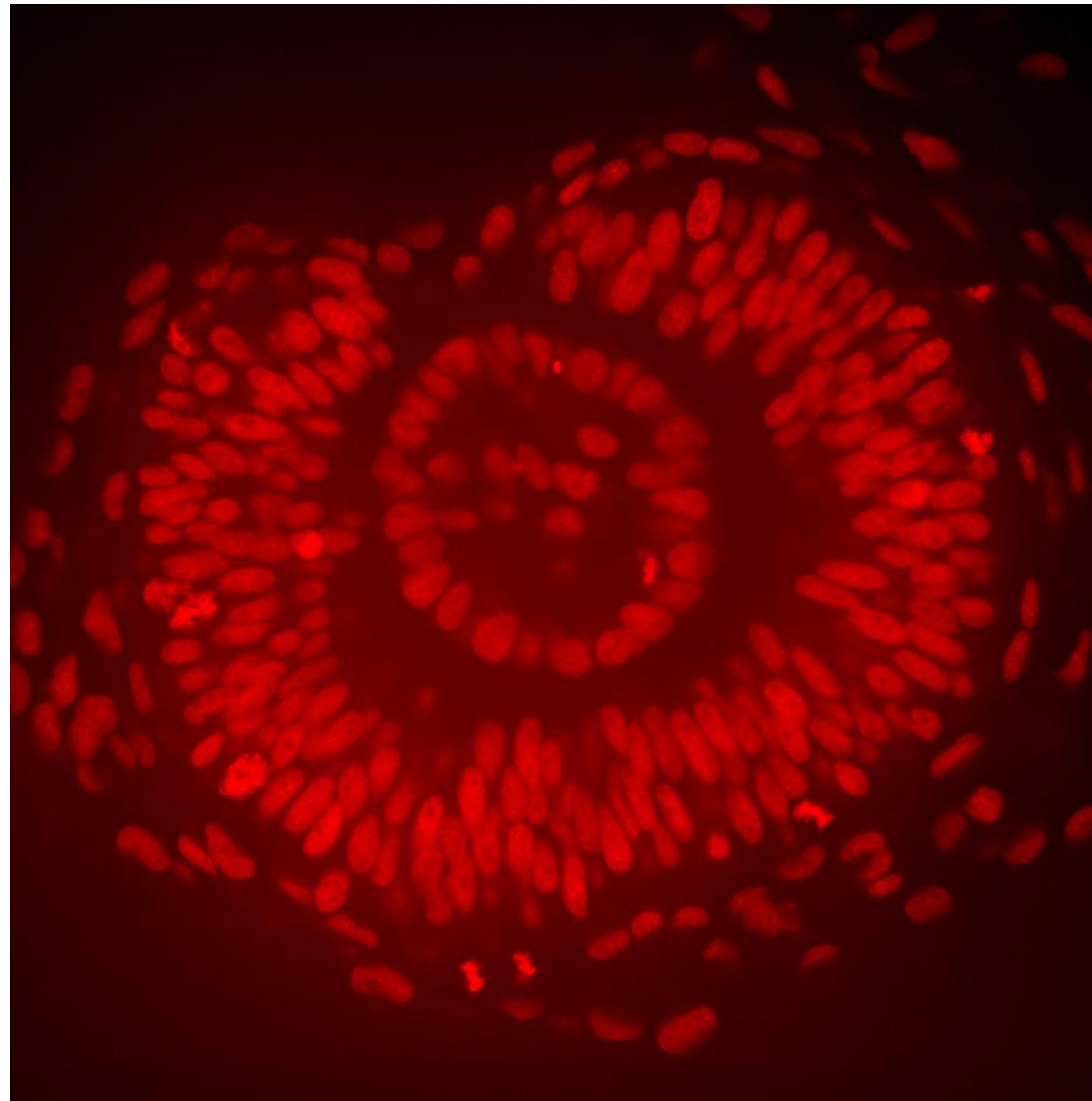
Binary mask images may not be perfect immediately after thresholding.

There are ways of refining them

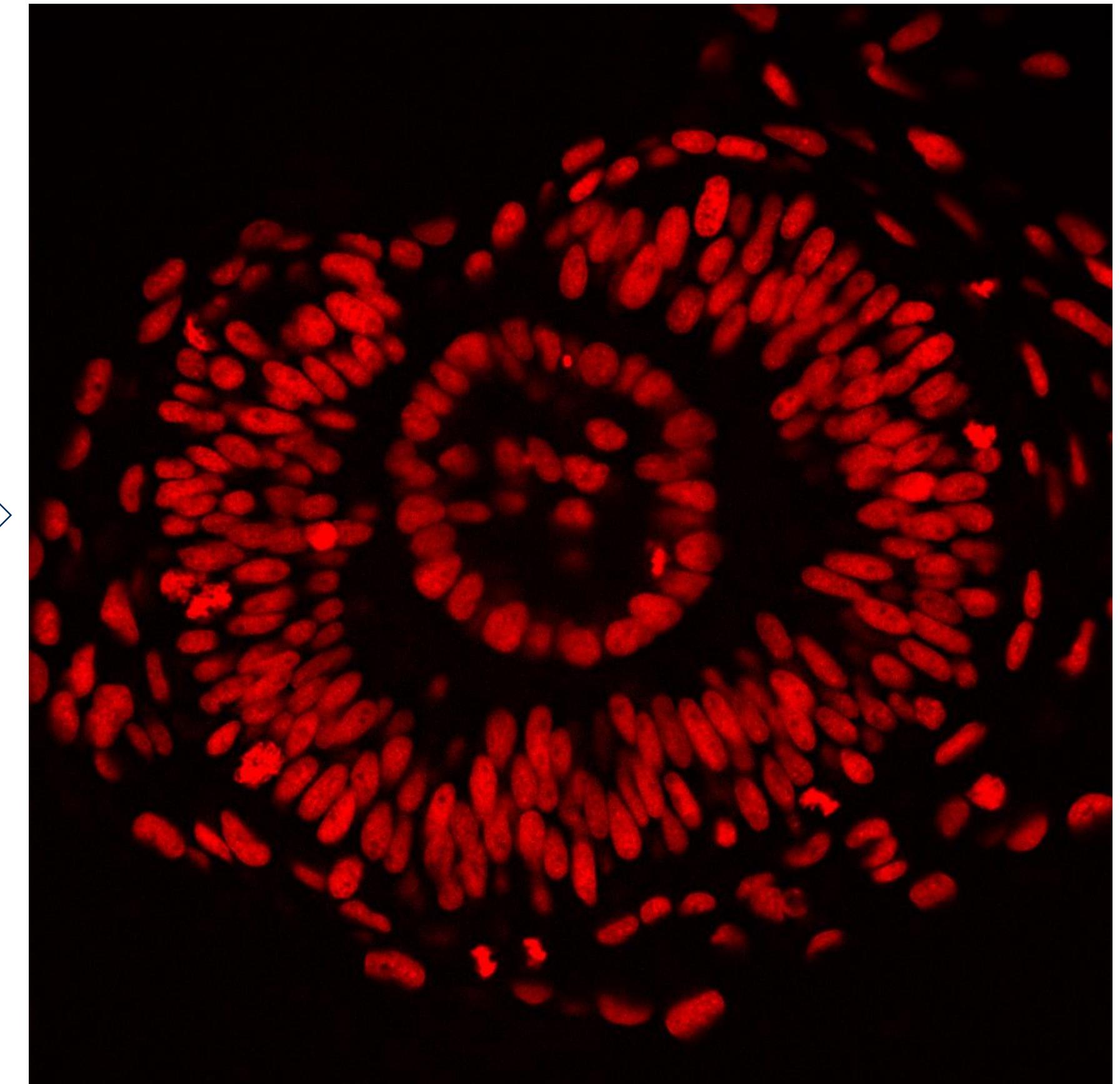


Background removal

Differentiating objects is easier if their background intensity is equal.



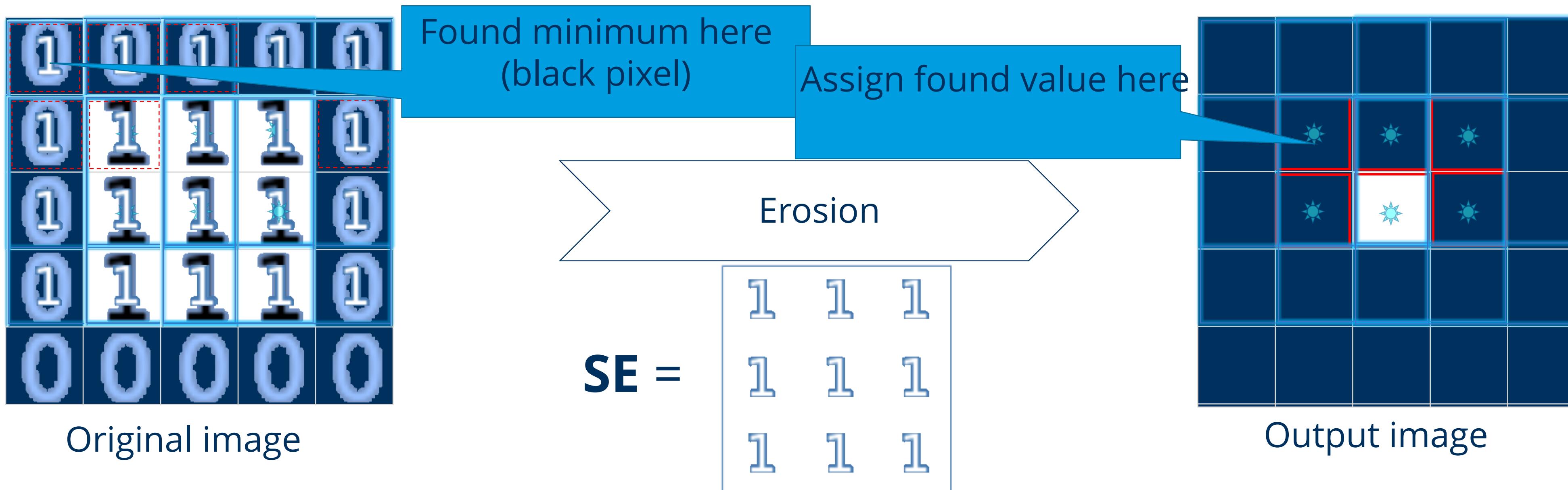
Subtract
background



Refining masks: Erosion

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

For an example with a grayscale image, check out this gif:
[https://en.wikipedia.org/wiki/Erosion_\(morphology\)#/media
/File:Grayscale_Morphological_Erosion.gif](https://en.wikipedia.org/wiki/Erosion_(morphology)#/media/File:Grayscale_Morphological_Erosion.gif)



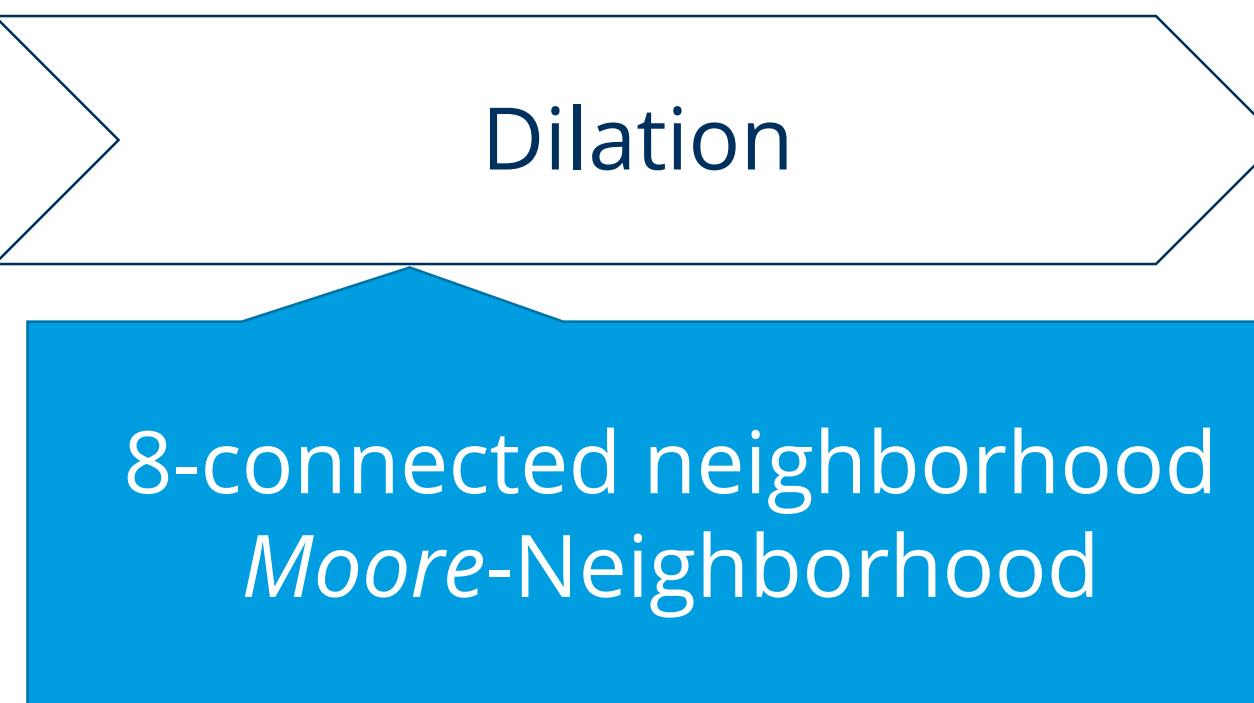
Erosion is essentially a minimum filter whose extent is defined by the kernel (structural element or **SE**) size and shape.

Dilation is essentially a maximum filter whose extent is defined by the kernel (structural element or **SE**) size and shape.

Refining masks: Dilation

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

1	1	1		
1	1	1		
1	1	1		

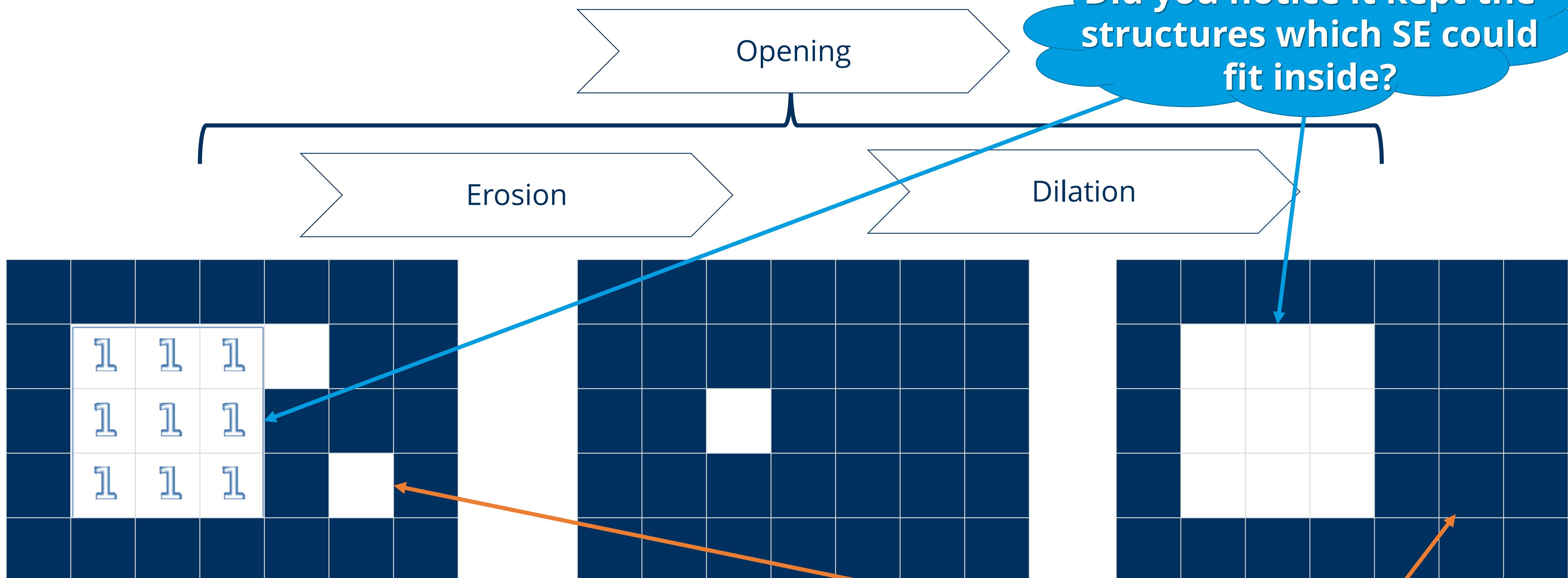


0	1	0		
1	1	1		
0	1	0		



Refining masks: Erosion & Dilation

Erosion and dilation combined allow correcting outlines.

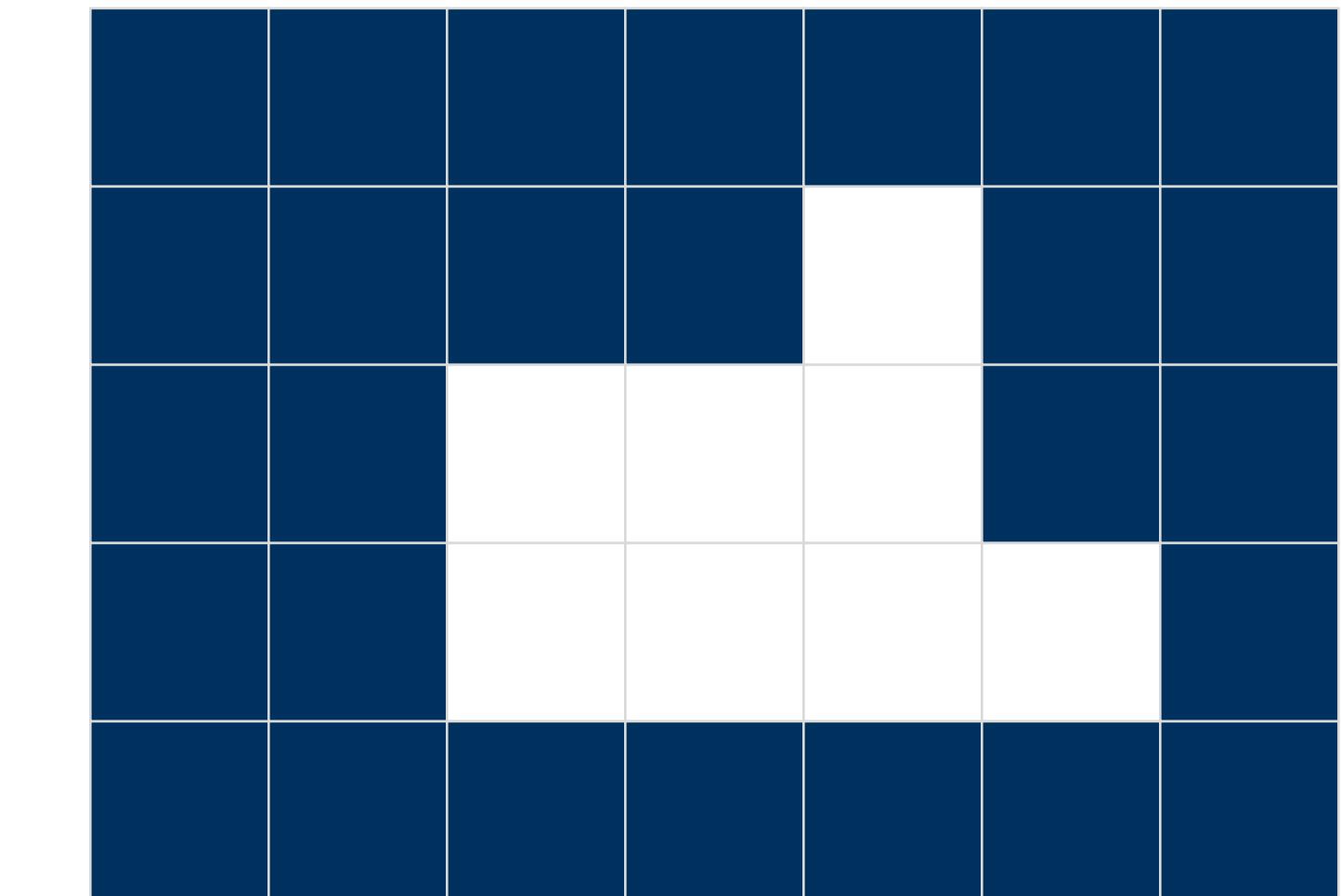
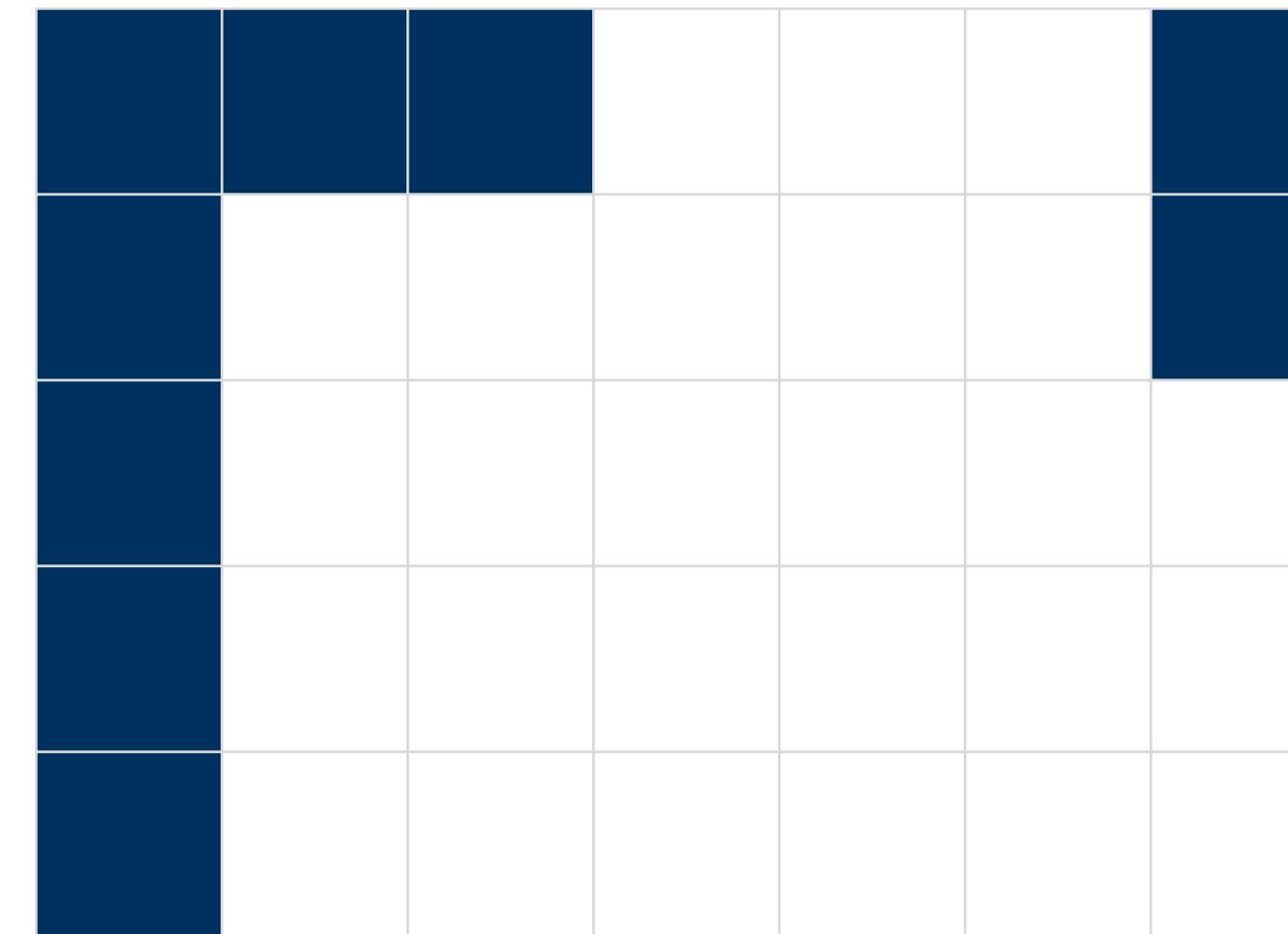
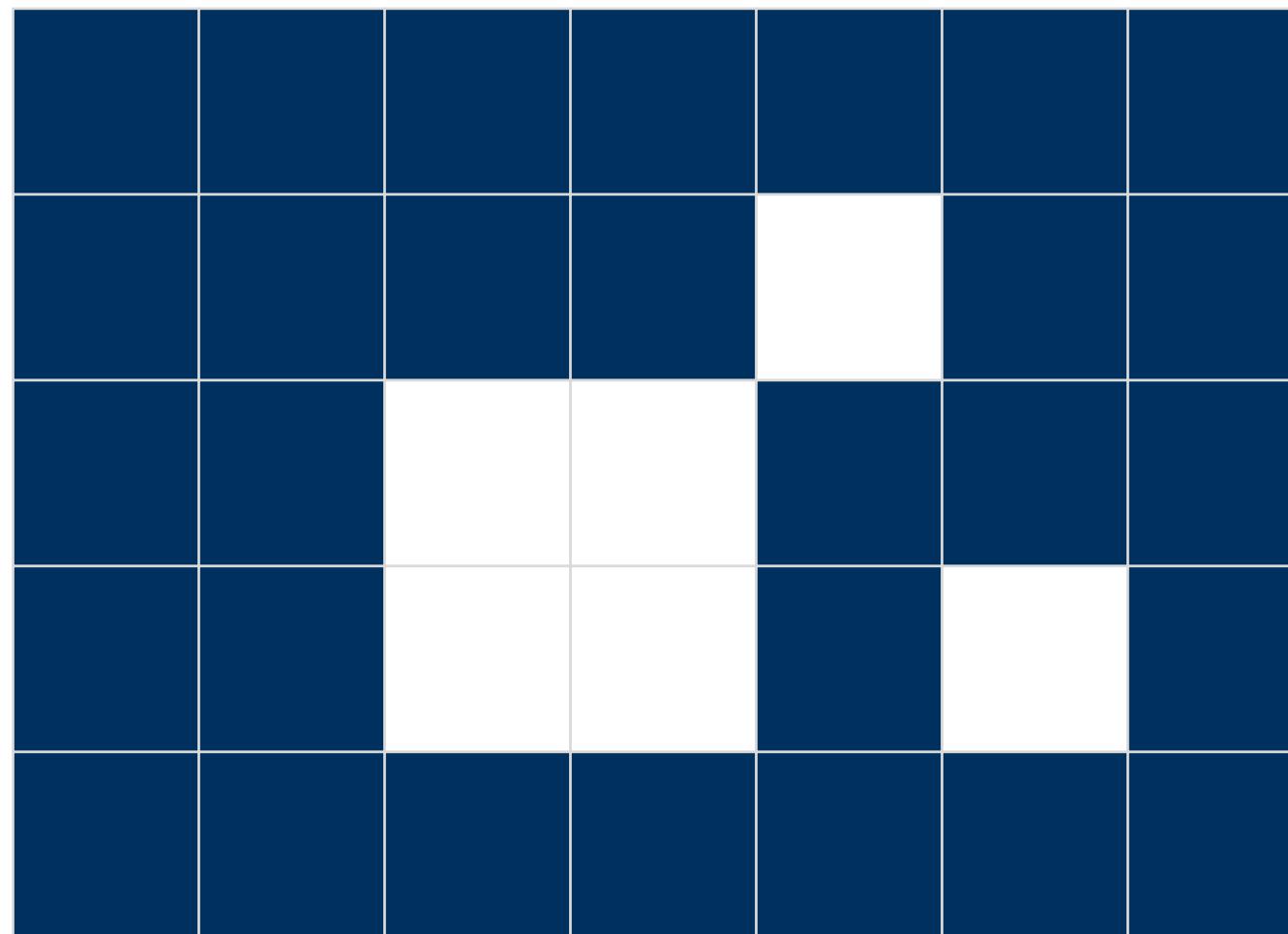
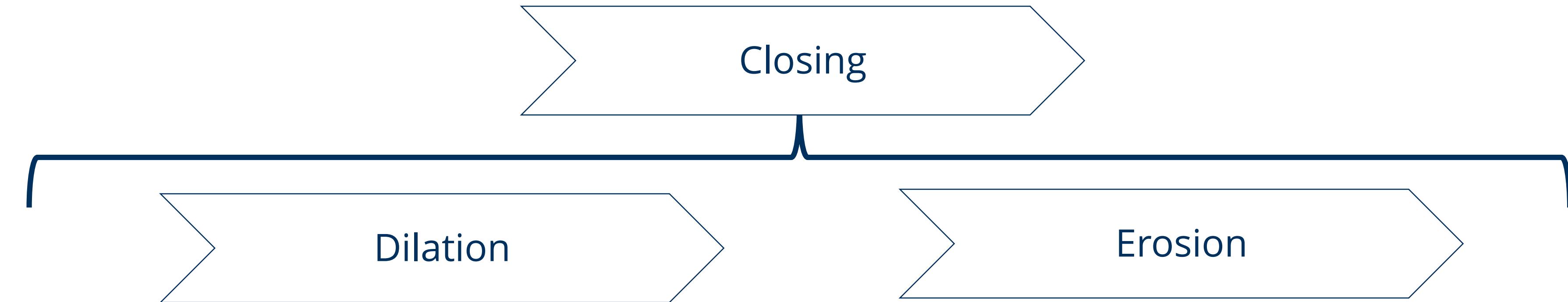


- It can separate white (high intensity) structures that are weakly connected
- It may erase small white structures
- It tends to better preserve area of structures

Did you notice it kept the structures which SE could fit inside?

And deletes structures smaller than SE?

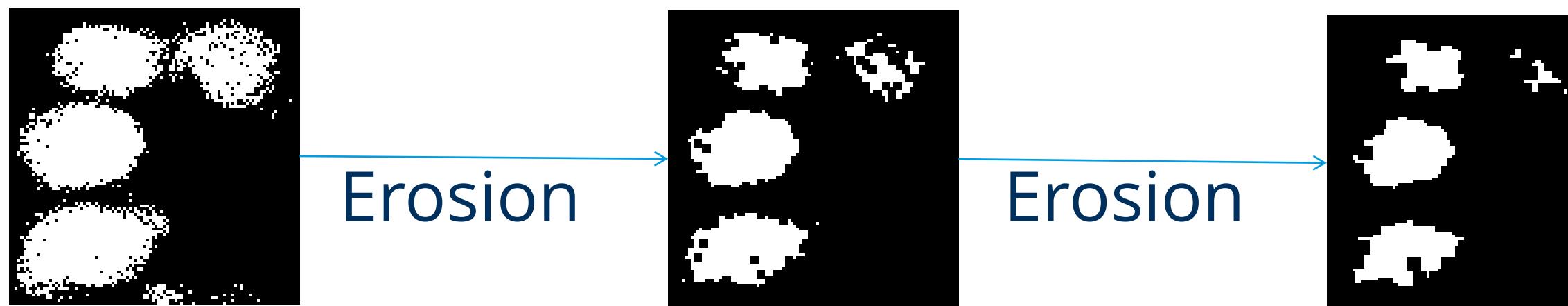
Refining masks: Erosion & Dilation



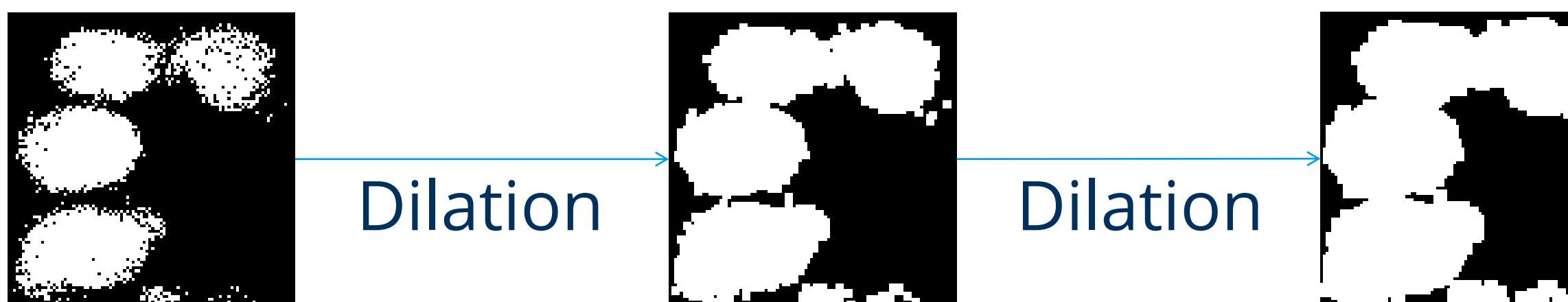
- It can connect white (high intensity) structures that are nearby
- It may close small holes inside structures
- It tends to better preserve area of structures

Refining masks: Erosion / dilation

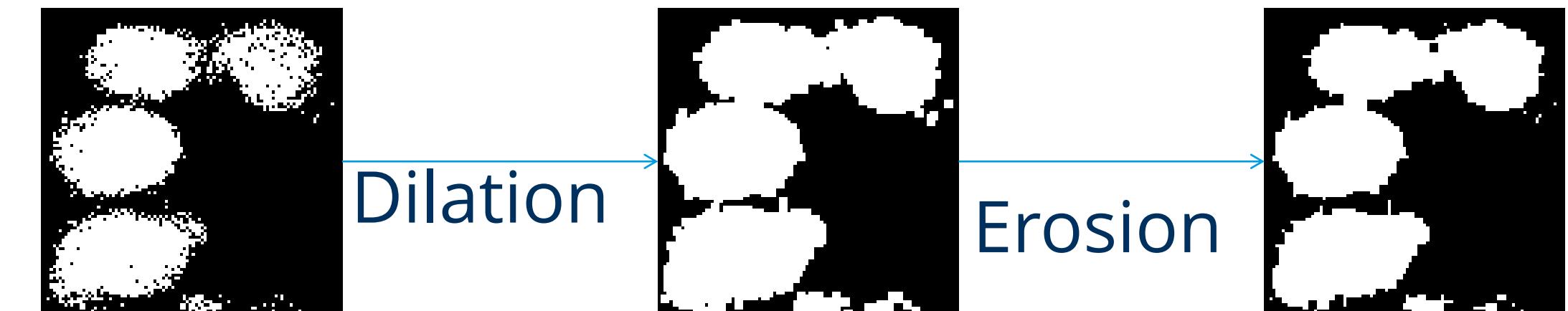
Dilation: Set all pixels to white which have at least one white neighbor.



Dilation: Set all pixels to white which have at least one white neighbor.



Closing: Dilation + Erosion



Opening: Erosion + Dilation

Slide adapted from Haase, Lombardot, Scientific Computing Facility, MPI CBG, licensed [CC BY-NC 4.0](#)

Morphological Operations in Python

Scikit-image has a sub-package called morphology

```
from skimage import morphology
```

You must define a SE first (also called footprint):

```
SE = morphology.square(3)
SE
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]], dtype=uint8)
```

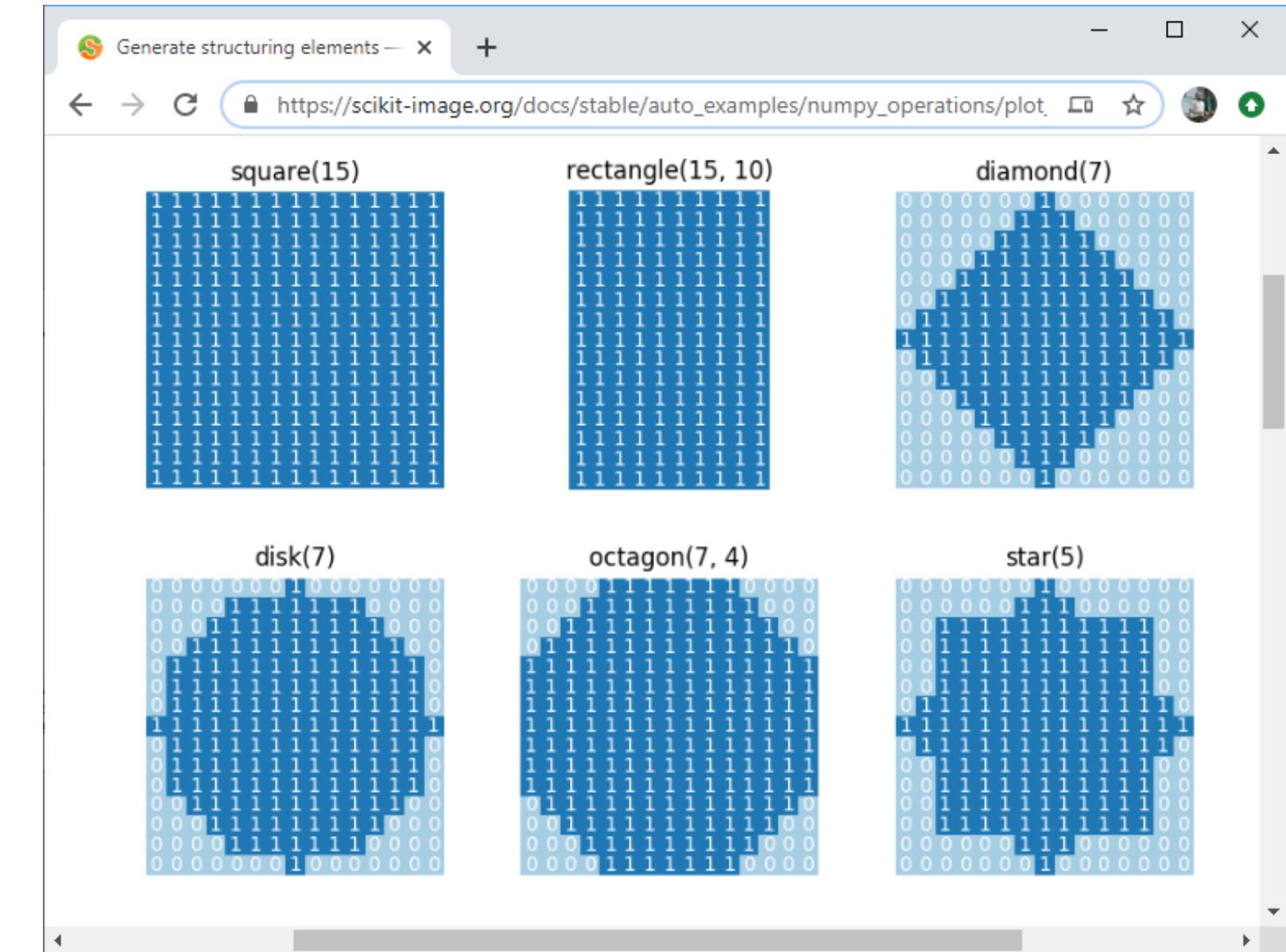
For a binary image, you apply it like this:

```
output = morphology.binary_dilation(binary_image, SE)
```

For a grayscale image, you apply it like this:

```
output = morphology.dilation(image, SE)
```

It can have other shapes/sizes:



https://scikit-image.org/docs/stable/auto_examples/numpy_operations/plot_structuring_elements.html#sphx-glr-auto-examples-numpy-operations-plot-structuring-elements-py

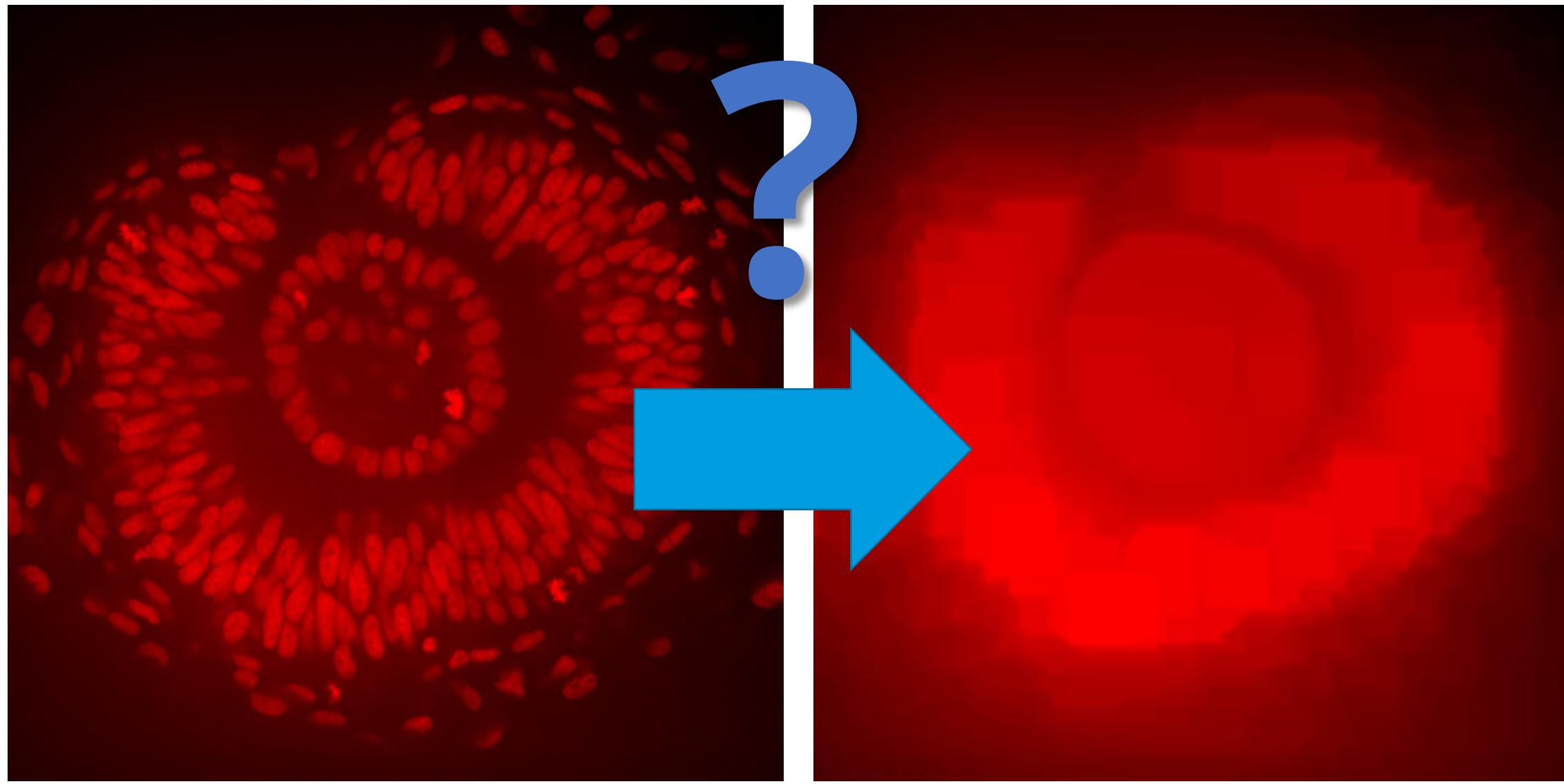
Background removal

Depending on the effect we want to correct for, it might make sense to divide an image by its background.

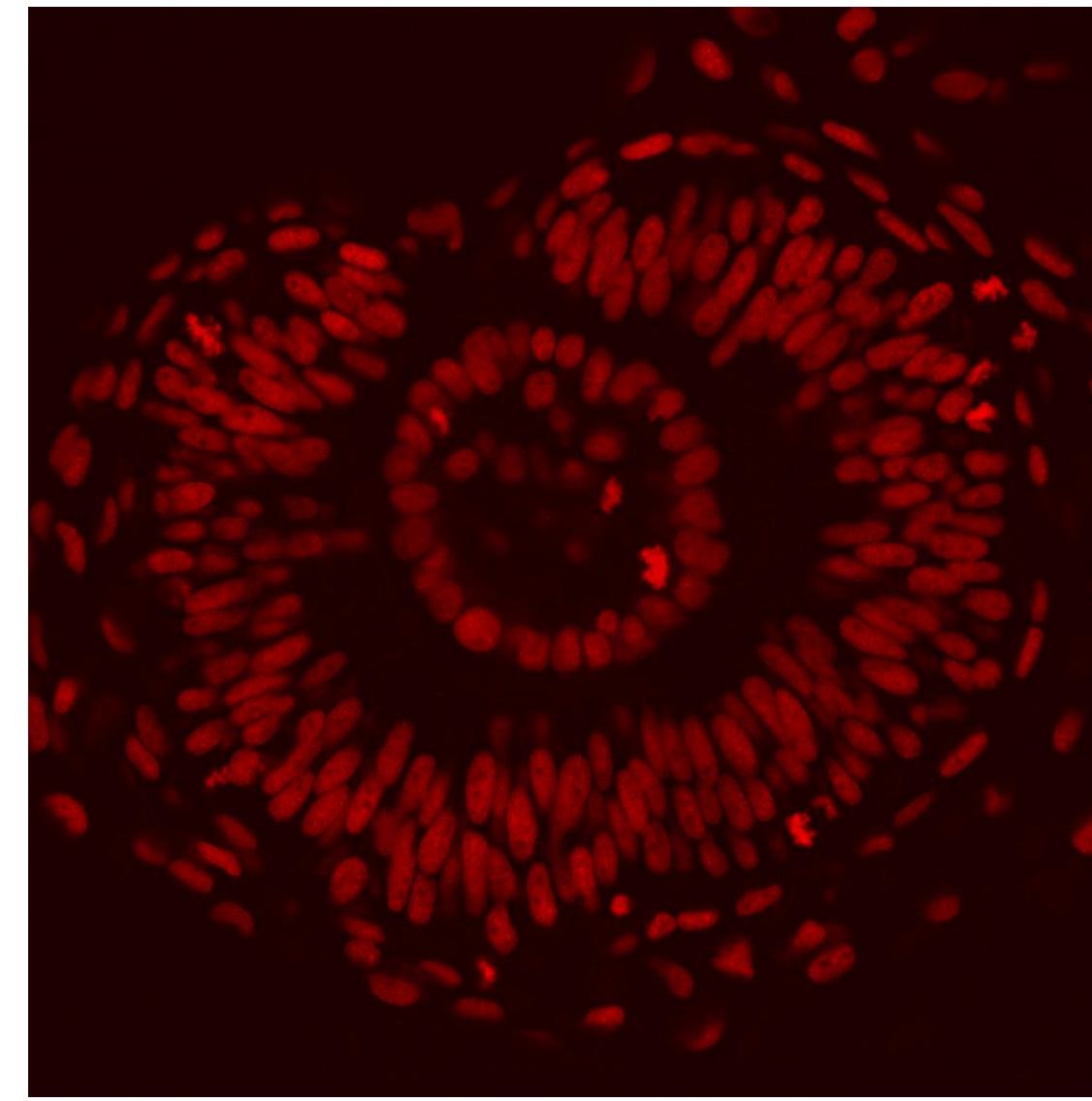
Do you remember that
opening kept the structures
which the SE could fit inside?

And do you remember that
opening deletes structures
smaller than SE?

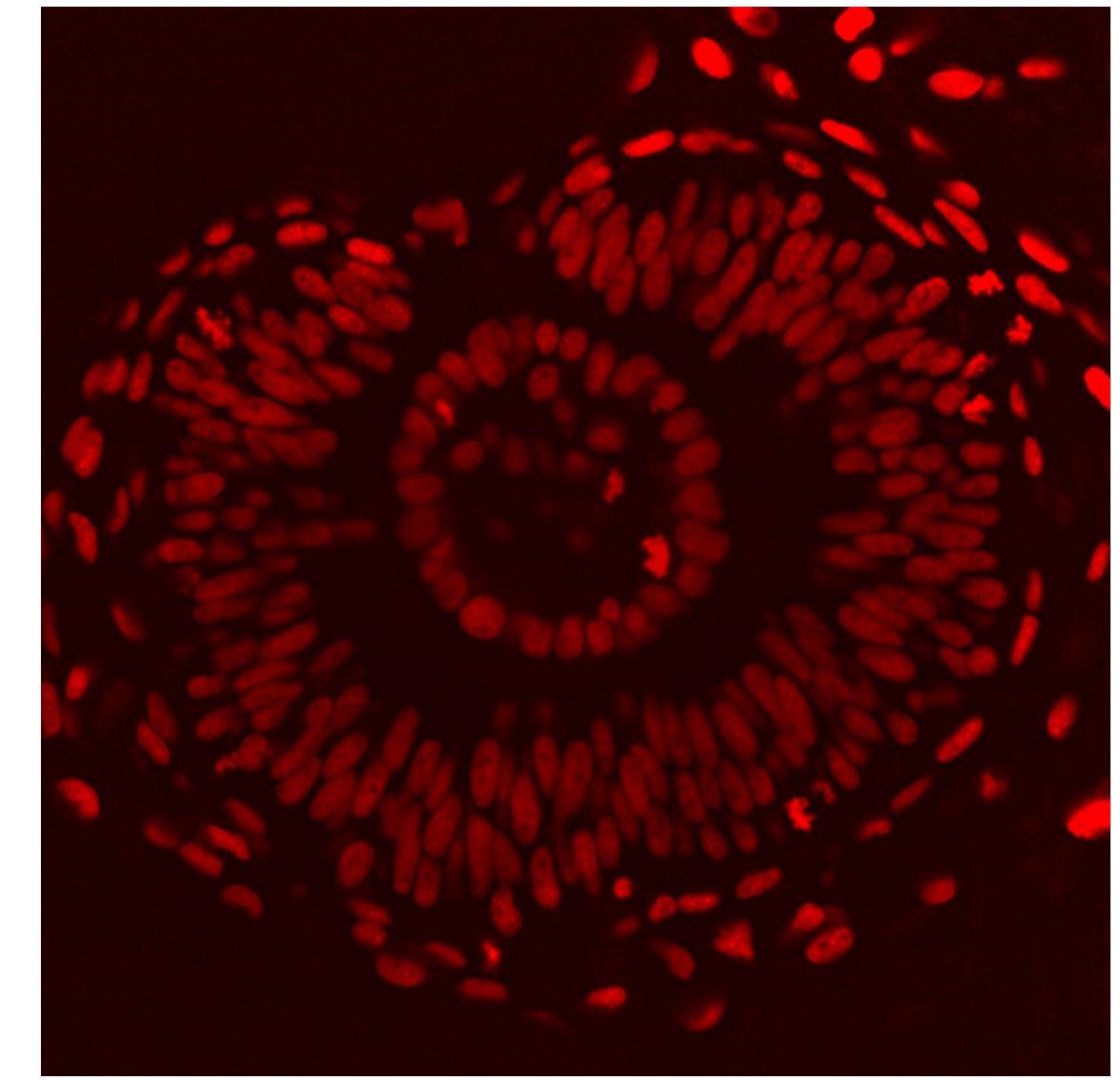
Image source: Mauricio Rocha
Martins (Norden/Myers lab, MP
CBG)



Original image



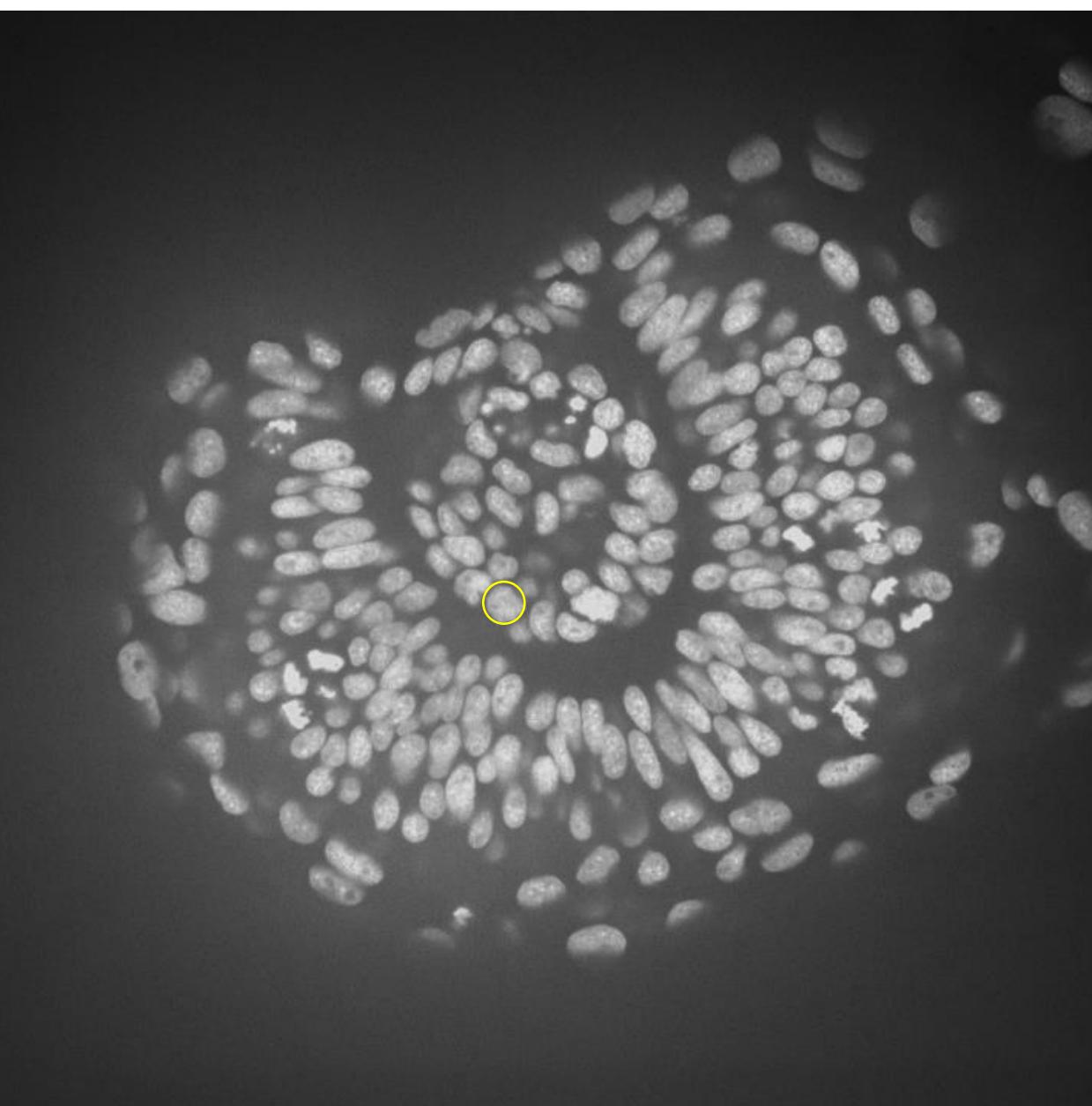
Extracted
background



Subtracted
background

Image divided by
background

Background removal



Original image

Structures have a radius ≈ 12

Opening

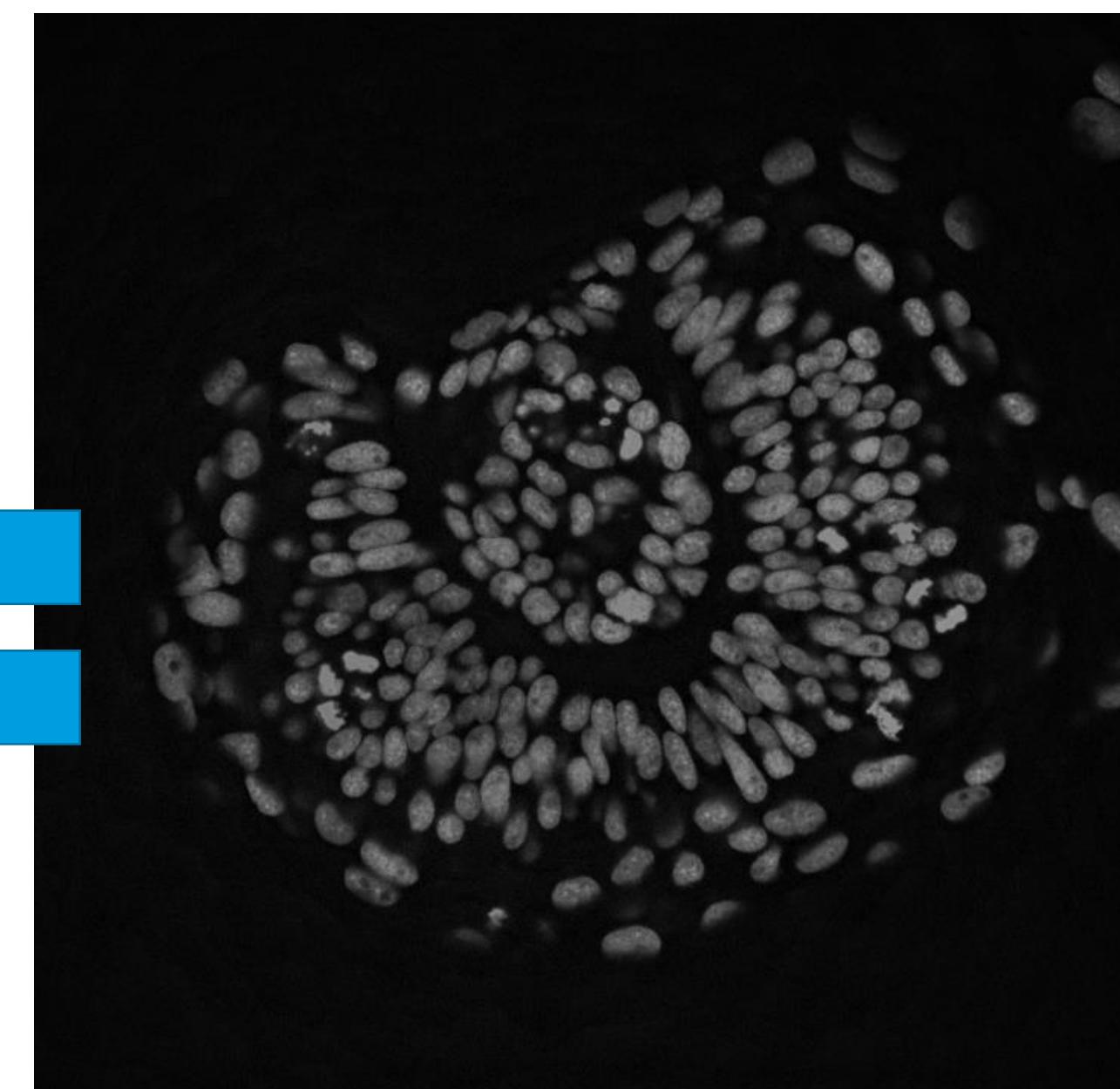
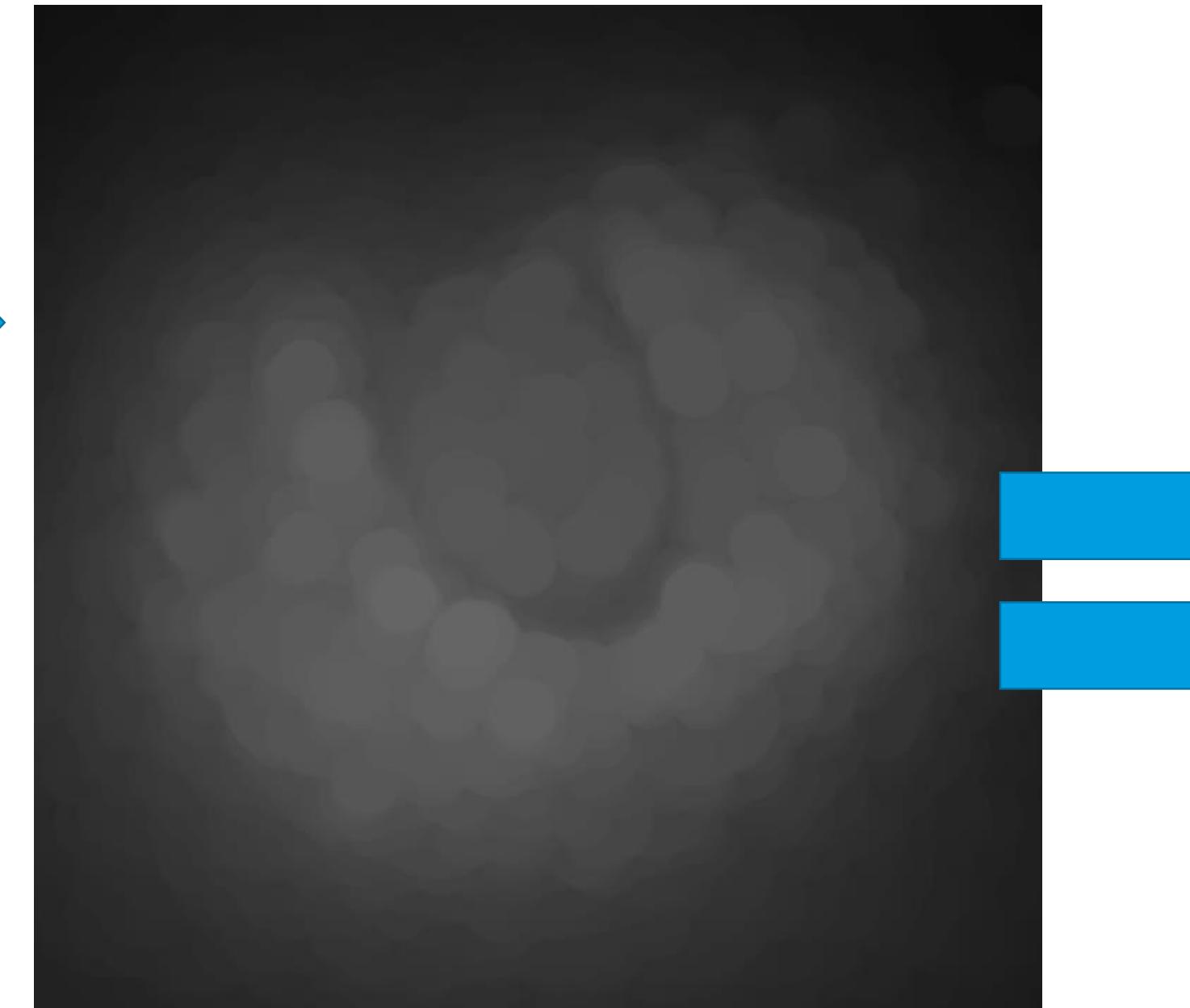
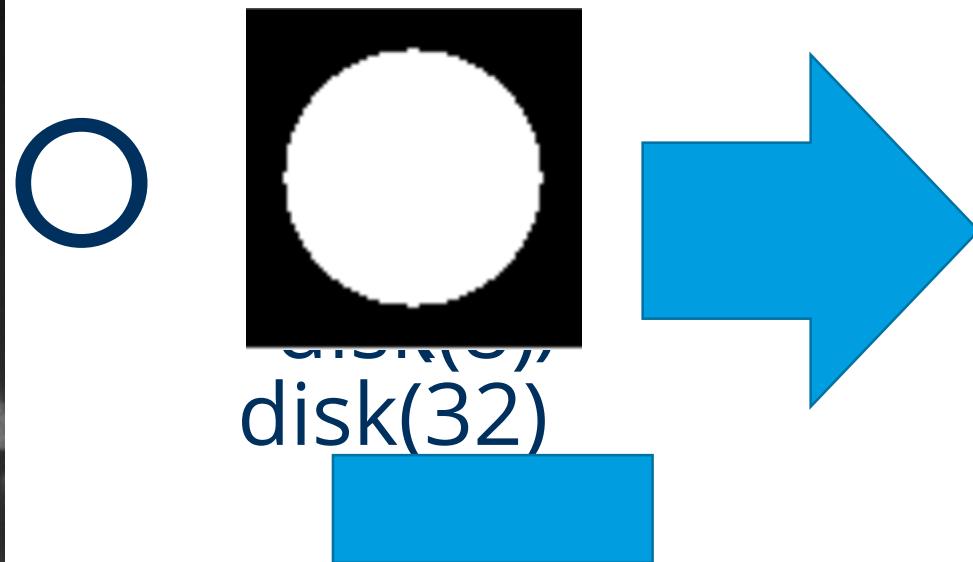


Image source: Mauricio Rocha Martins
(Norden/Myers lab, MPI CBG)

This is a good estimation of the background

You have just learned the white tophat filter!

Exercises

Image thresholding and feature extraction

Johannes Müller

With material from

Robert Haase, BiaPoL, PoL TU Dresden

Marcelo Zoccoler, BiaPol, PoL, TU Dresden

Benoit Lombardot, Scientific Computing Facility, MPI CBG

Lecture overview

Today's content:

Complete workflow from raw images toward extracting quantitative features from image data

Creating binary masks from intensity images (**Segmentation**)

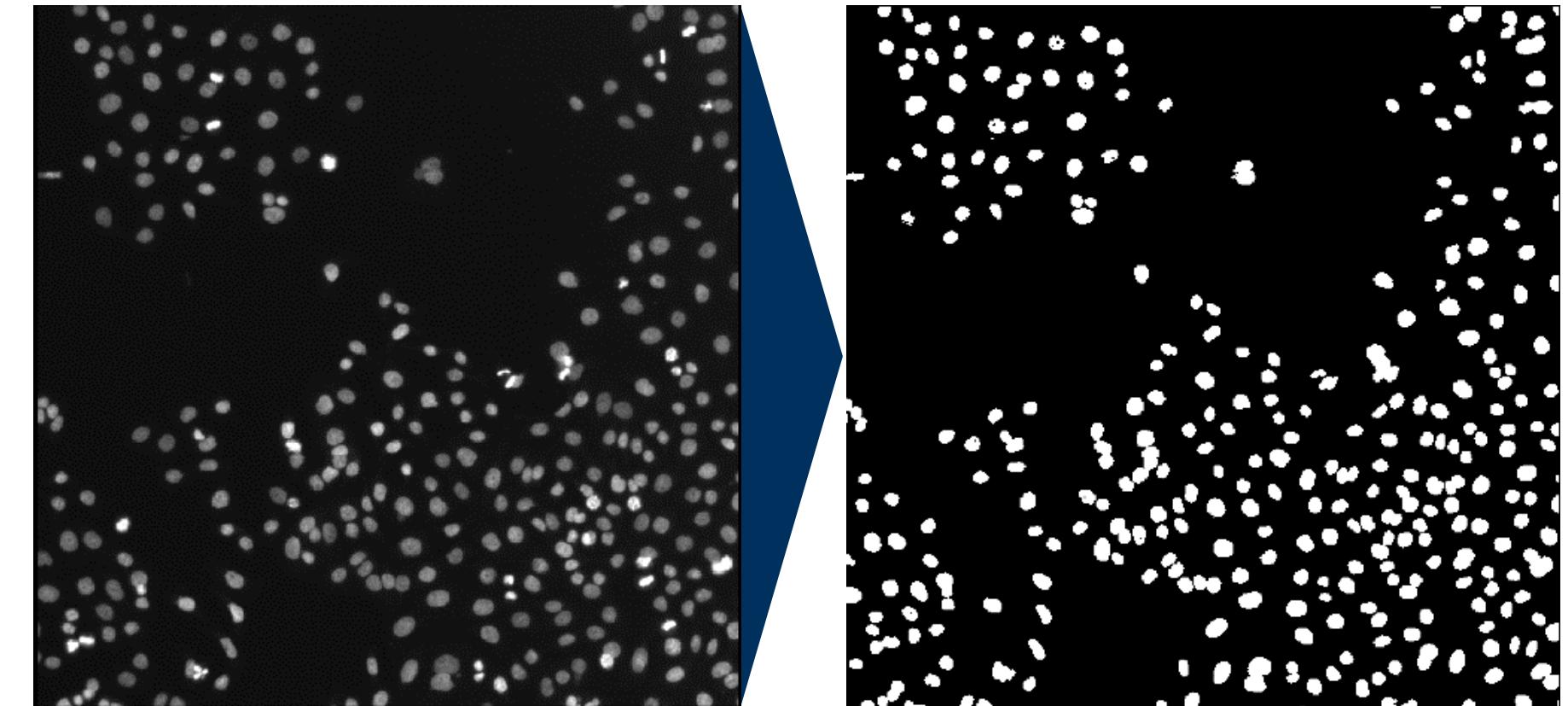
- Thresholding

Create label images from binary masks

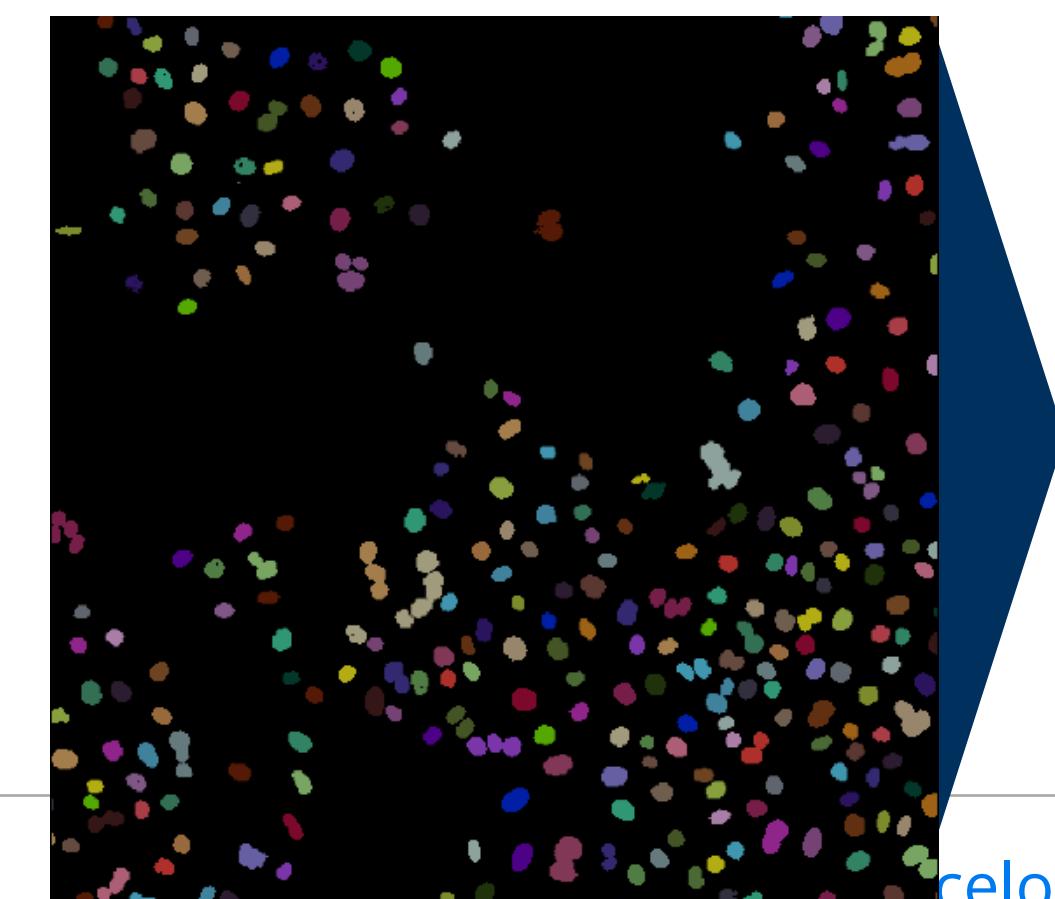
- Connected-component analysis

Measure features from label images

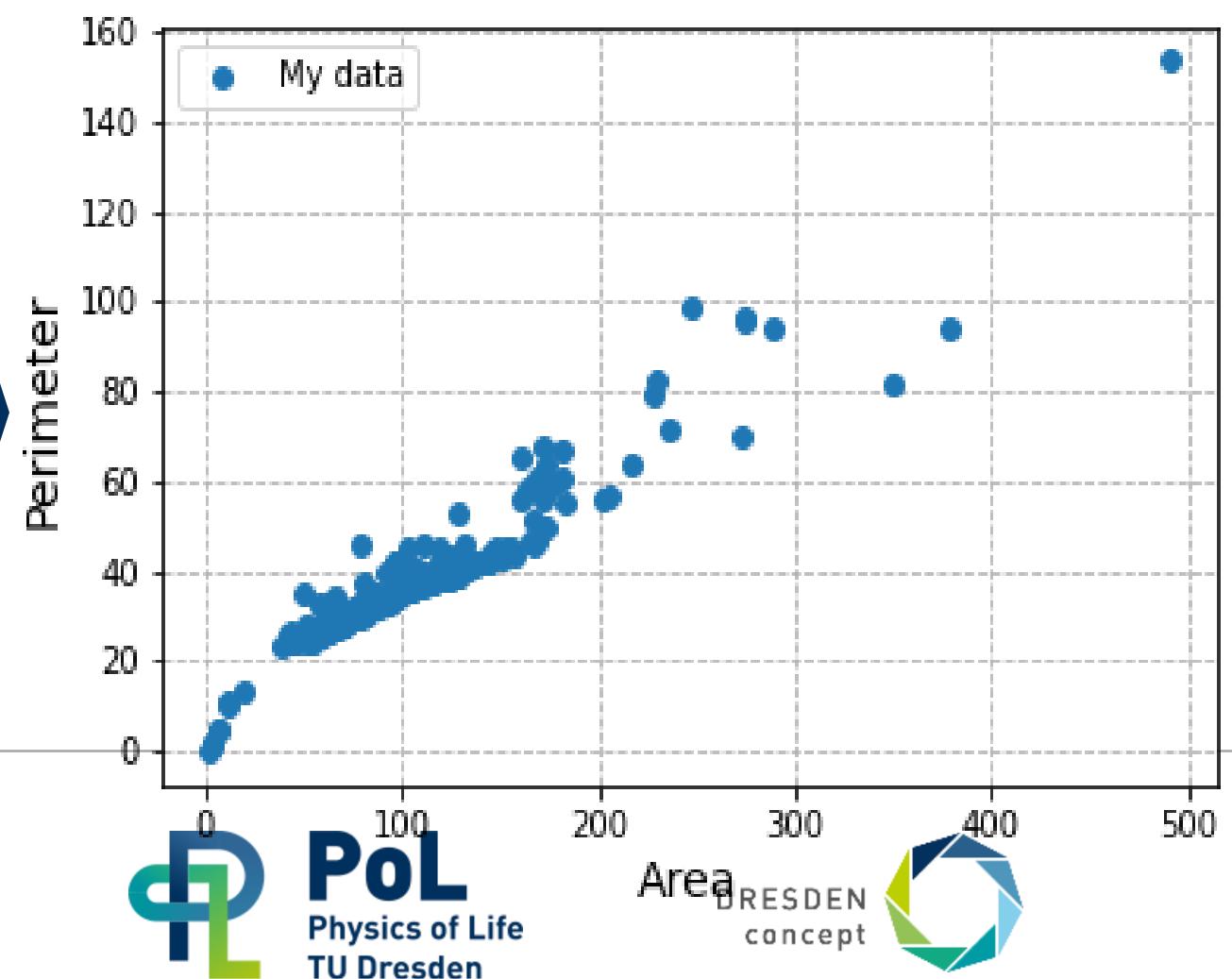
- Shape, Intensity, etc.
- Visualize measured features



Labelled image



Measurement



Segmentation

Aim:

Separate background from foreground

Vocabulary:

-Segmentation:

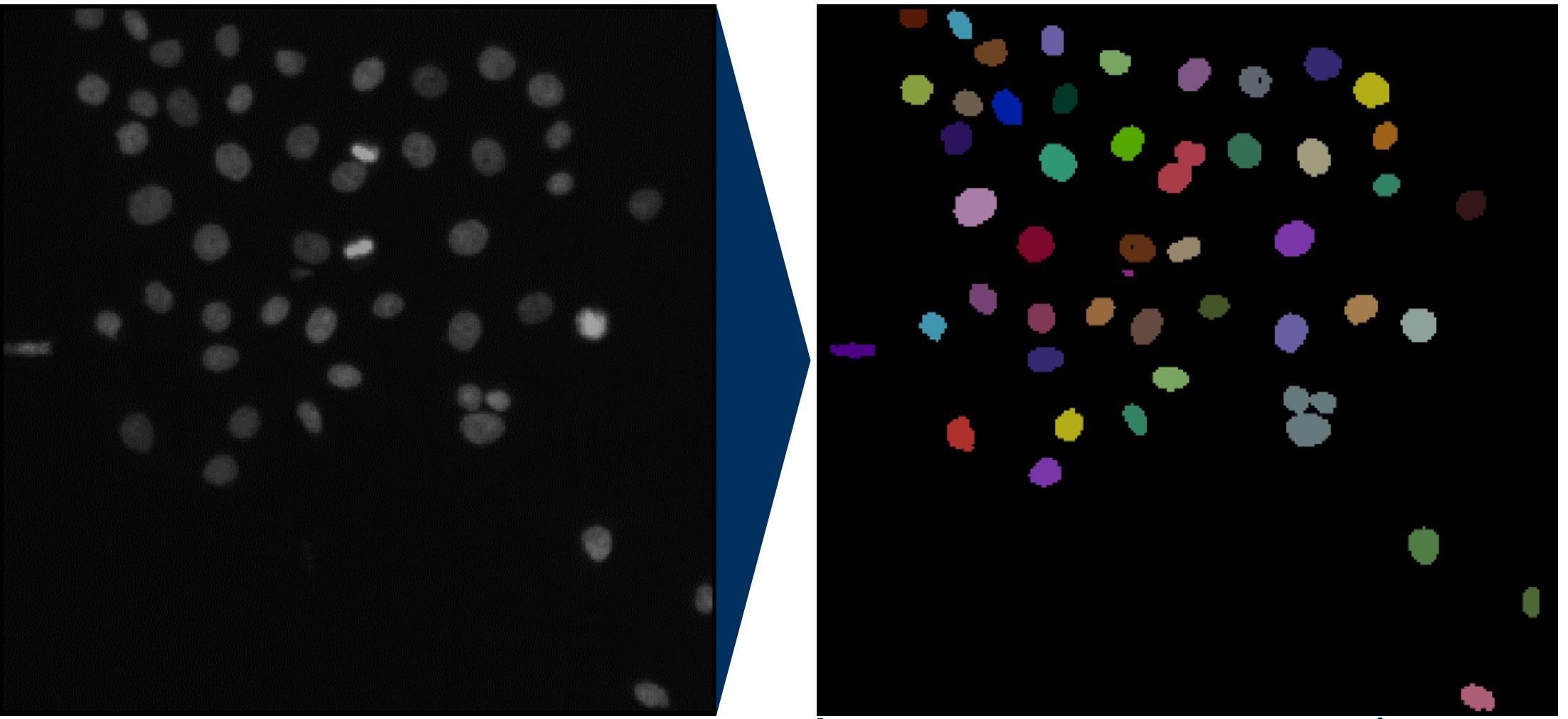
Assigning a meaningful *label* to each pixel

-Semantic segmentation:

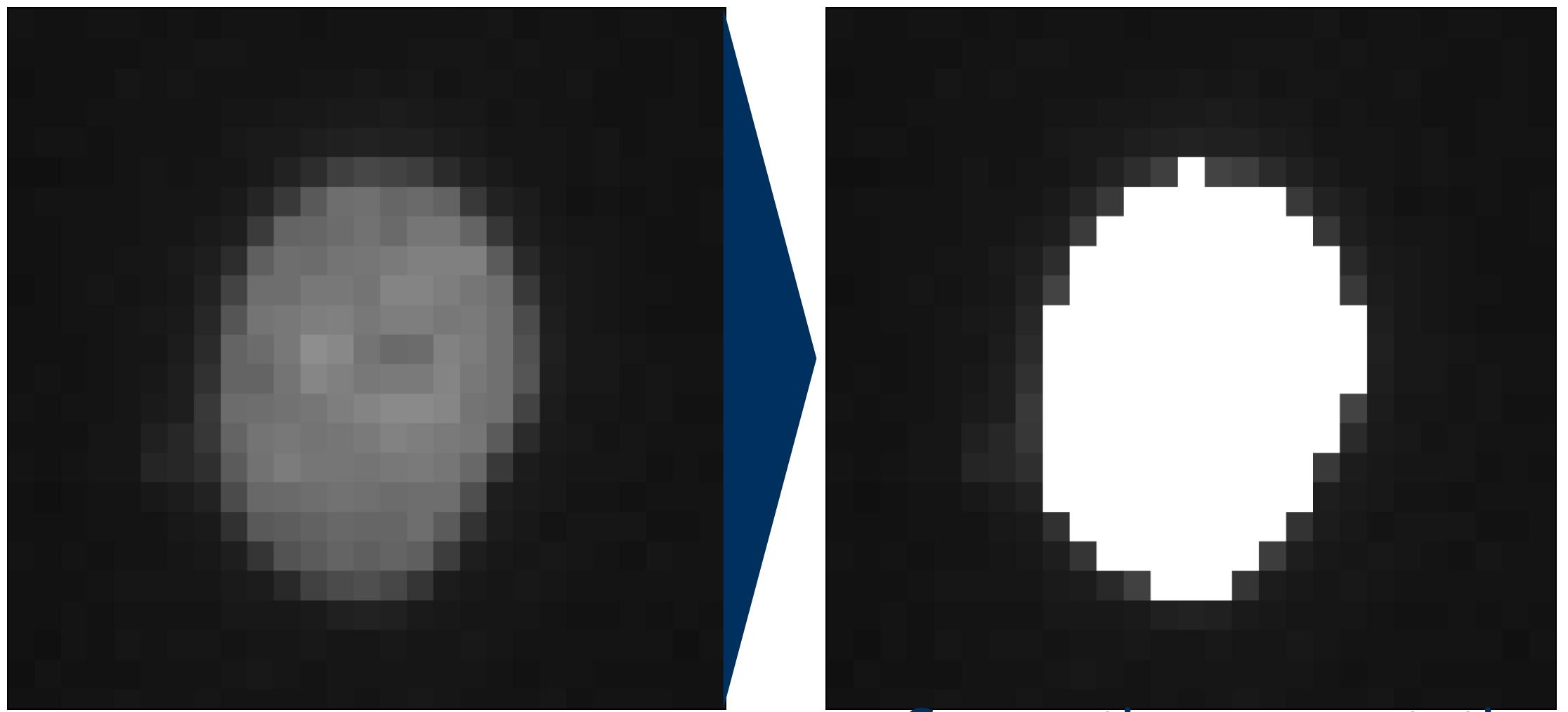
Differentiate pixels into multiple *classes* (e.g., membrane, nucleus, cytosol, etc.)

-Instance segmentation:

Differentiate multiple occurrences of the same class into separate instances of this class (e.g., separate *label* for each cell in image)



Instance segmentation



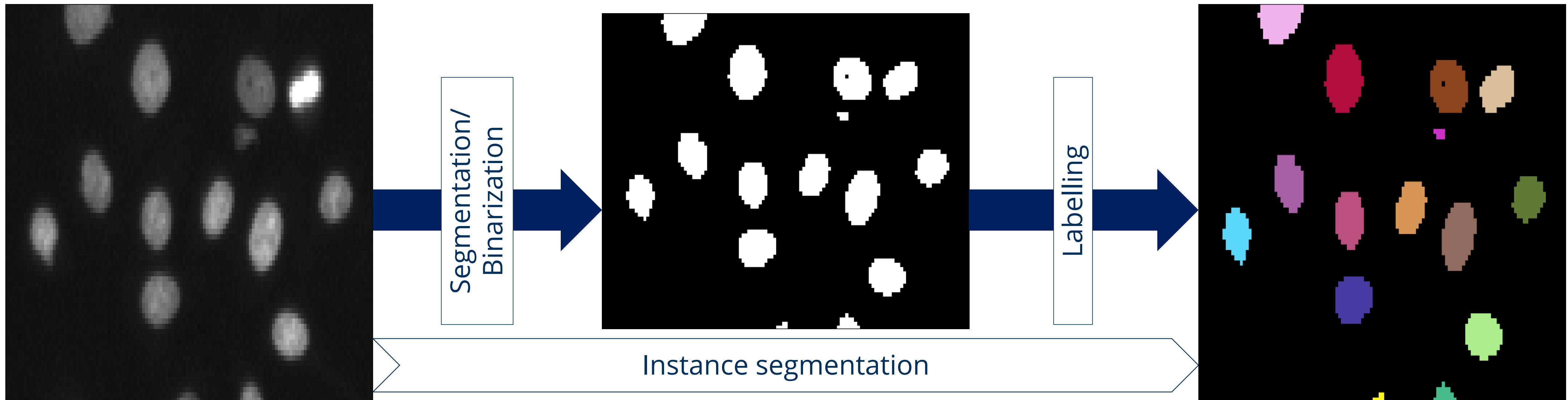
Semantic segmentation

Segmentation and labelling

Analyzing properties (*features*) of individual objects in images requires instance segmentation

Methods

- Thresholding + connected components labeling
- Spot detection + seeded watershed
- Edge detection based
- Machine learning



Thresholding in Python

Applying a threshold to an image requires to compare every pixel to the threshold value

We can compare values in Python with:

```
a = 5  
b = 6  
print(a > b)  
print(a < b)  
print(a==b)
```

```
False  
True  
False
```

se, “image” is a

```
image > threshold
```

```
array([[False, False, False, ..., False, False],  
       [False, False, False, ..., False, False],  
       [False, False, False, ..., False, False],  
       ...,  
       [False, False, False, ..., False, False],  
       [False, False, False, ..., False, False],  
       [False, False, False, ..., False, False]])
```

applied to every pixel!

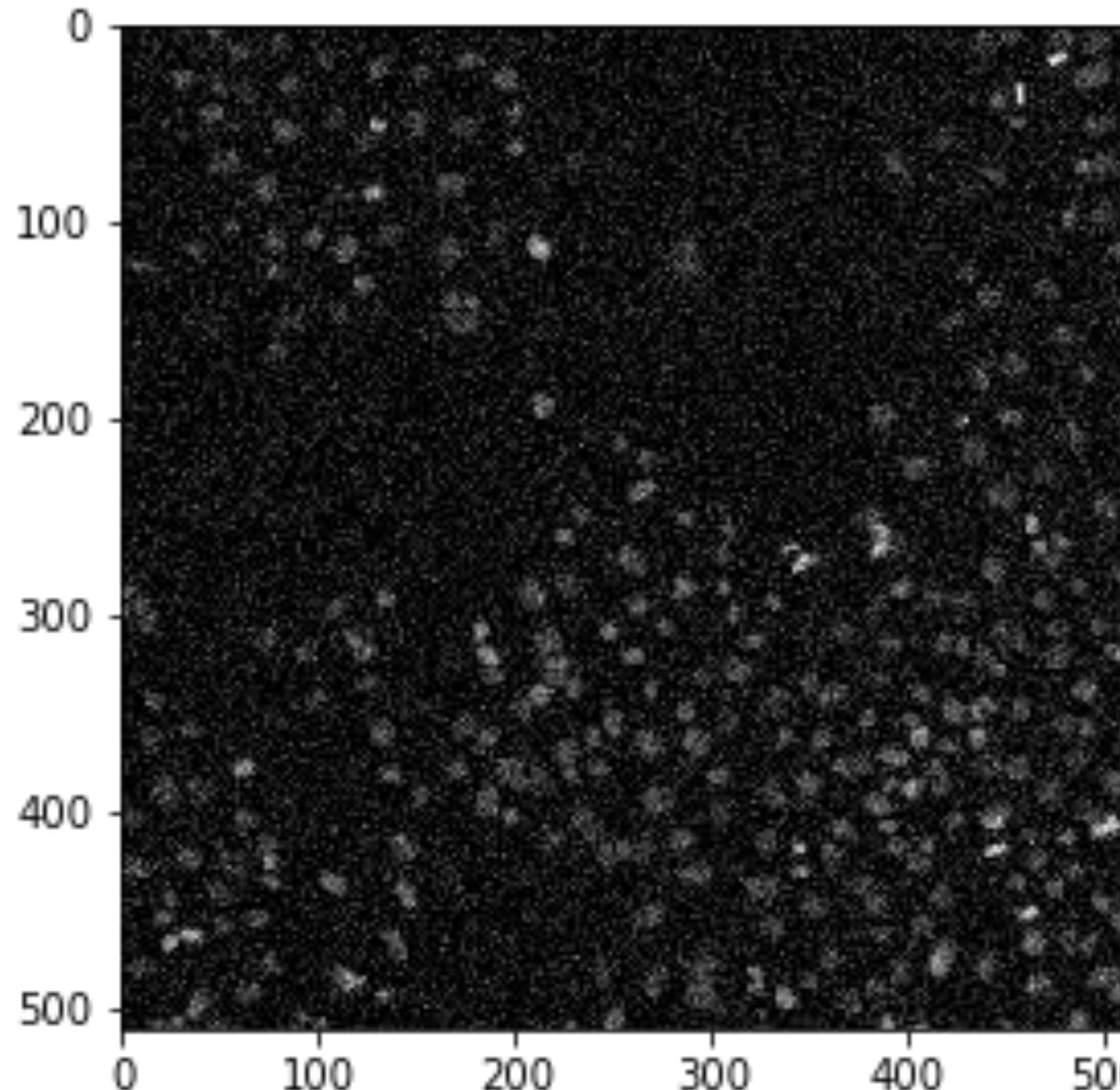
We can then simply store the output of this element-wise comparison in a new variable:

```
binary = image > threshold
```

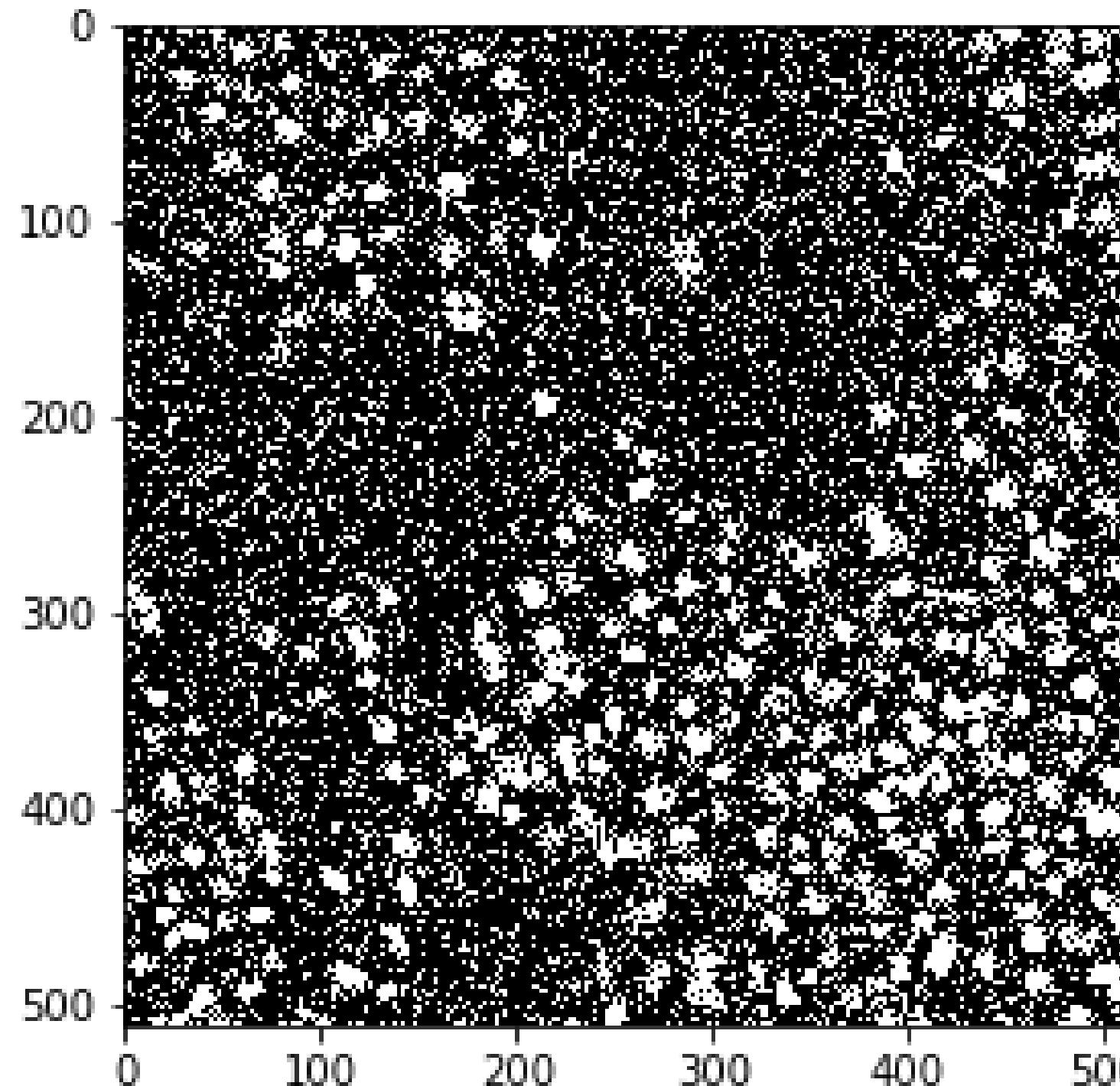
Reminder: pre-processing!

Before we can create masks, we need to pre-process images.

- Noise removal
- Background subtraction



Noisy image



Thresholded image

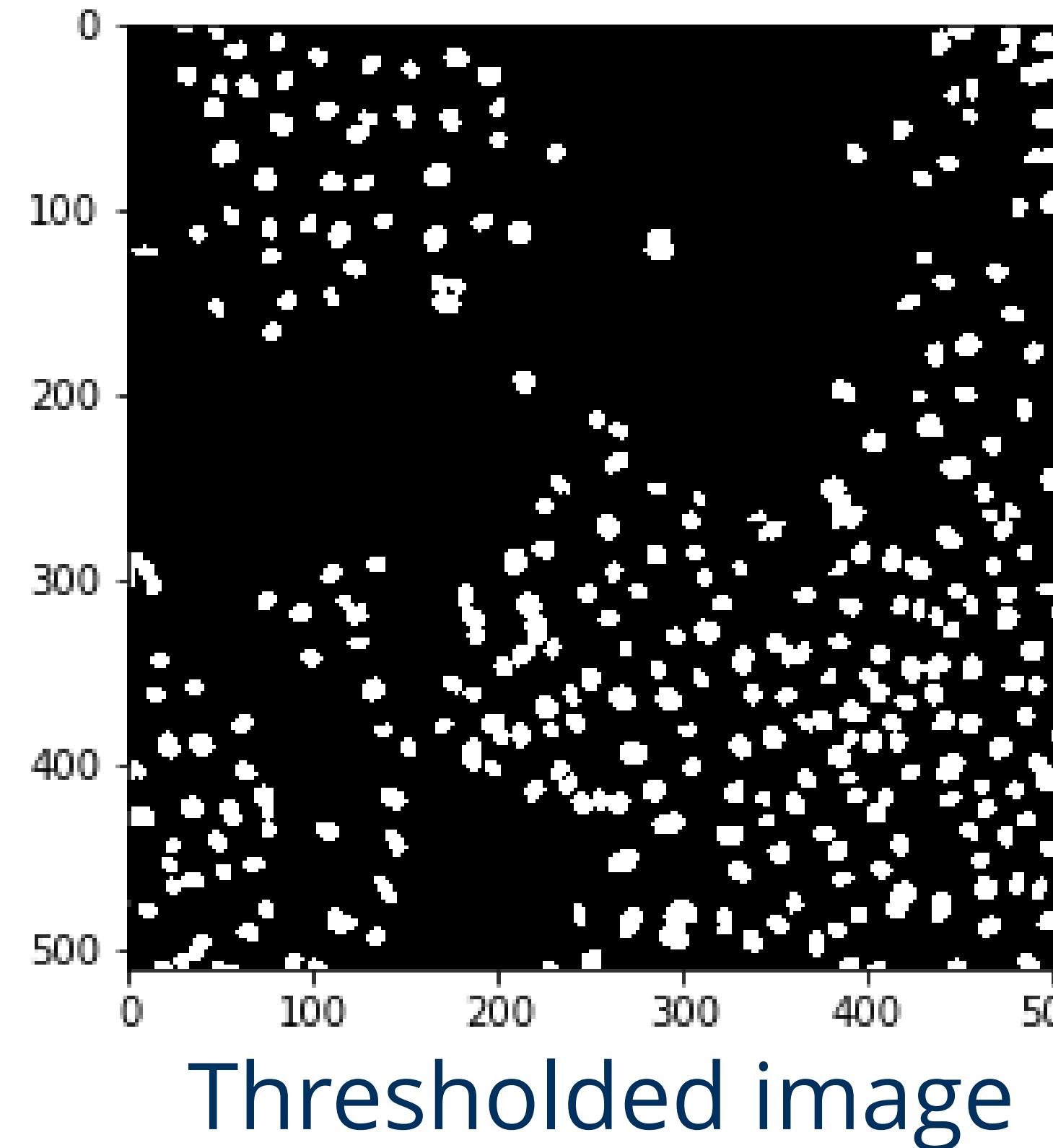
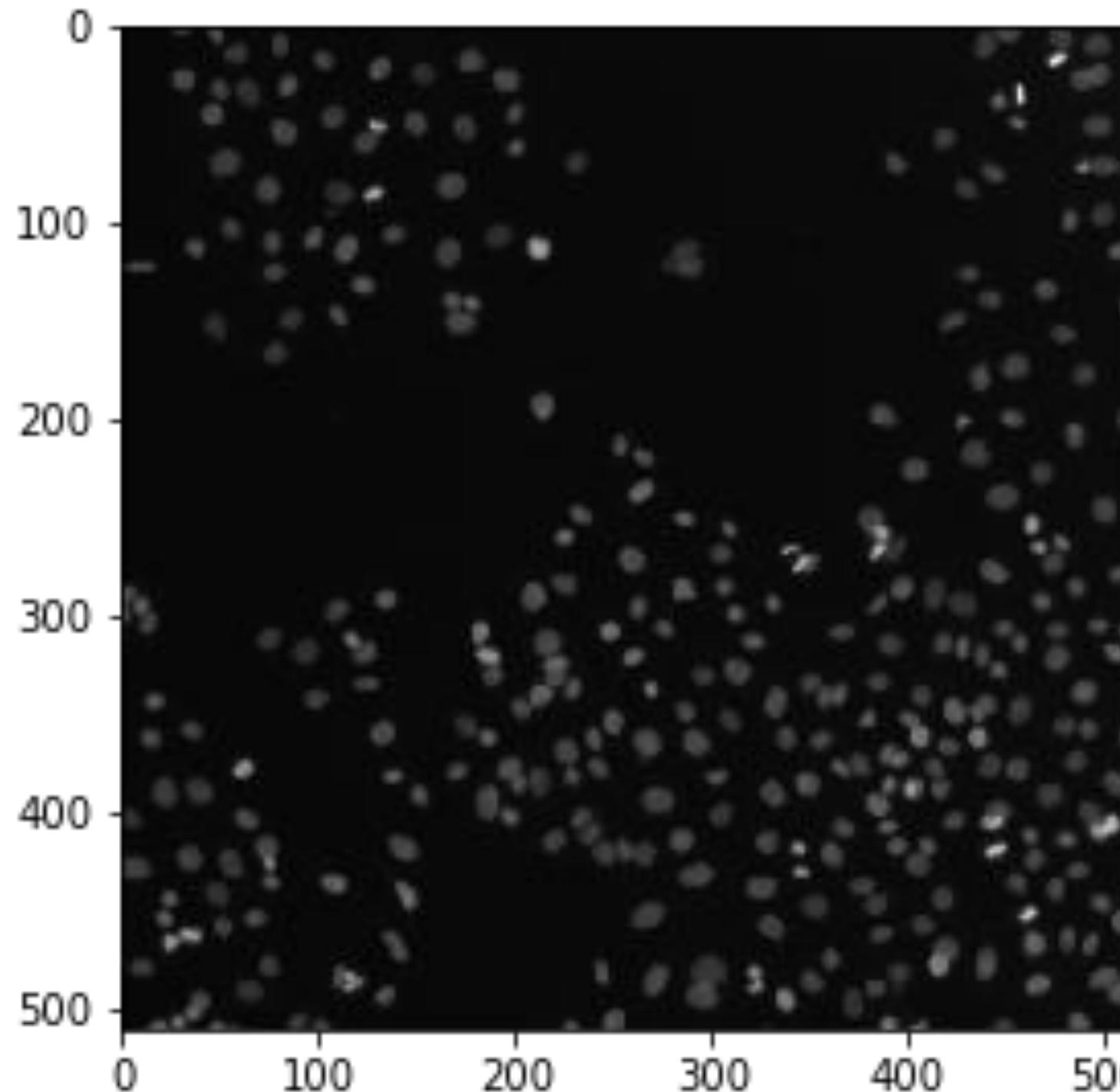
```
filtered = filters.median(image)
```

Image filtering *filters* relevant information for subsequent operations from the image!

Reminder: pre-processing!

Before we can create masks, we need to pre-process images.

- Noise removal
- Background subtraction



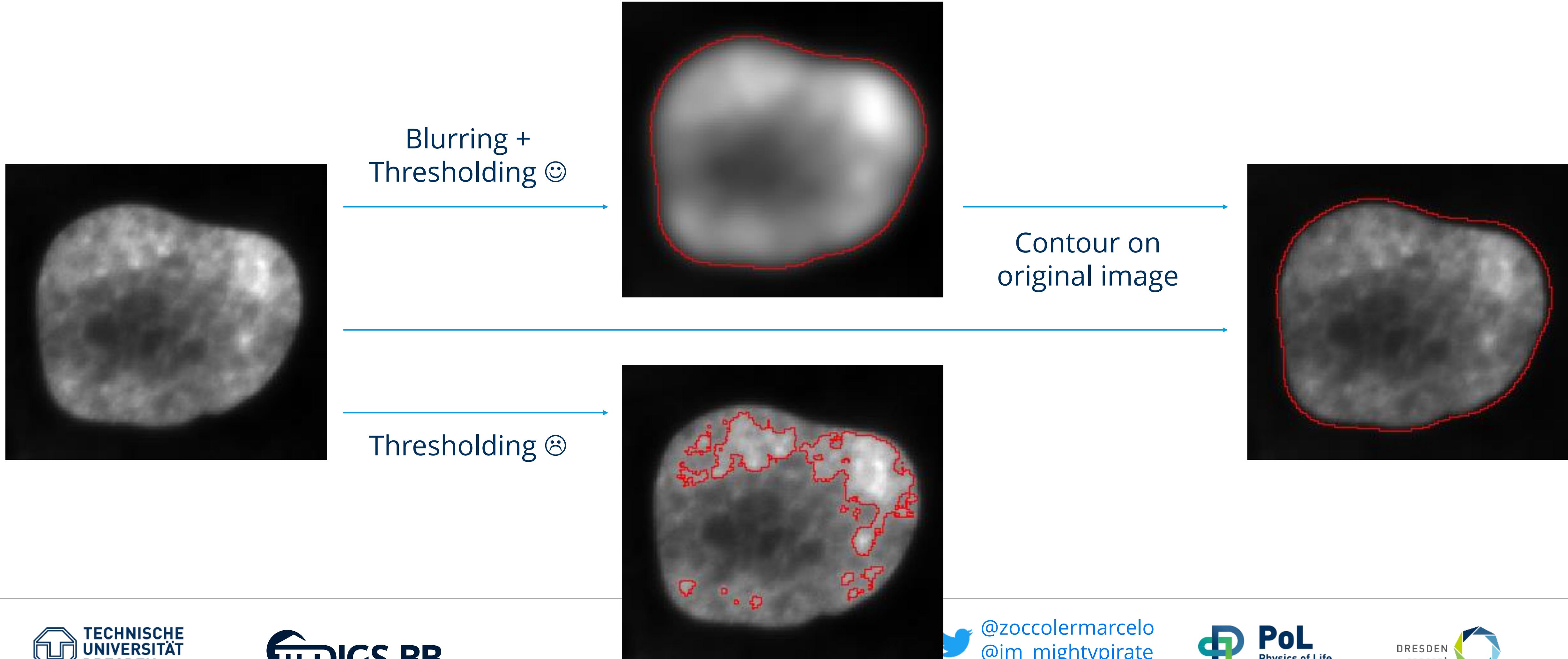
```
filtered = filters.median(image)
```

Image filtering *filters* relevant information for subsequent operations from the image!

Low-pass filtering to improve thresholding results

In case thresholding algorithms outline the wrong structure, blurring in advance may help.

However: **Do not** continue processing the blurred image, continue with the original!

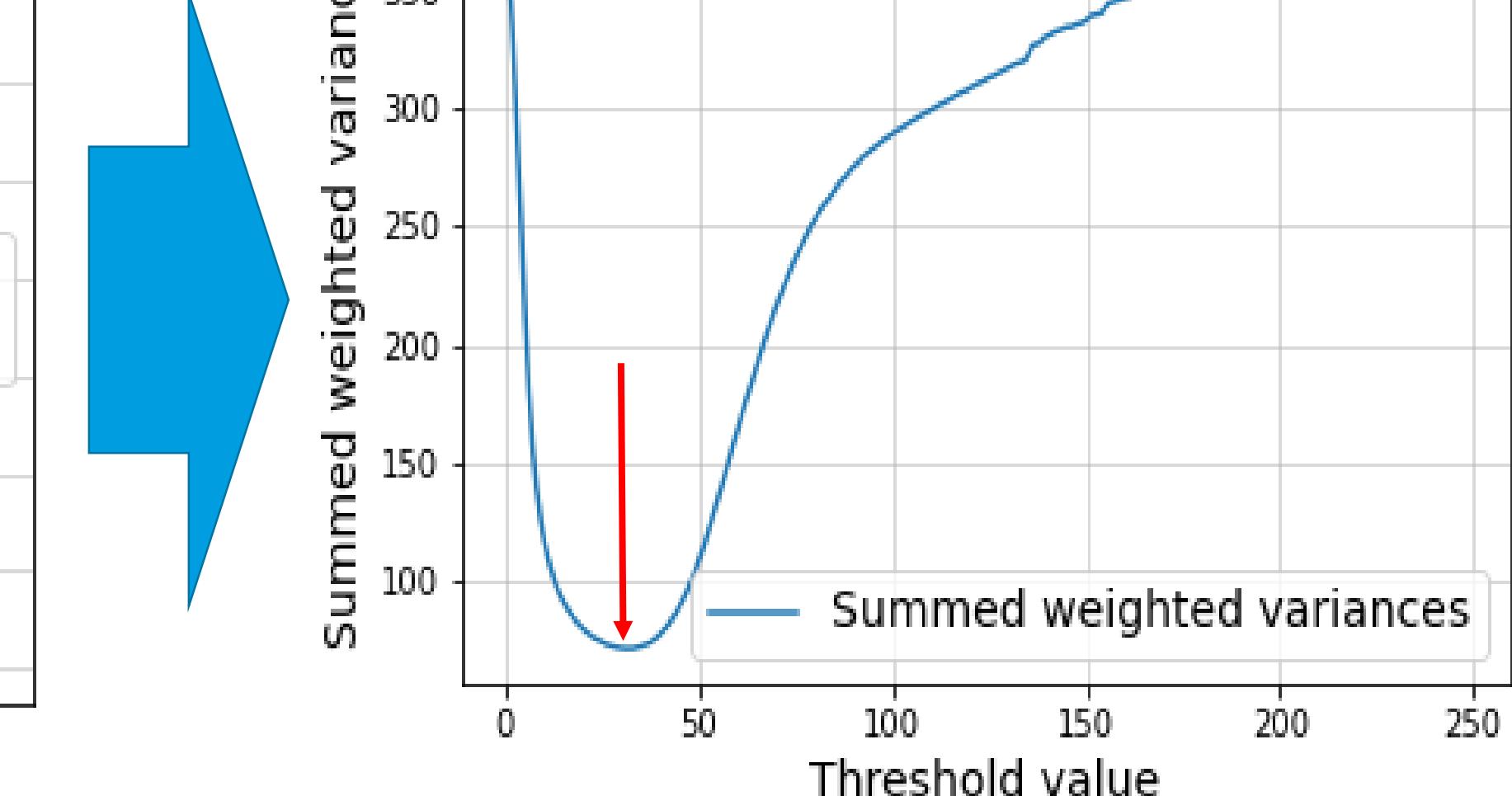
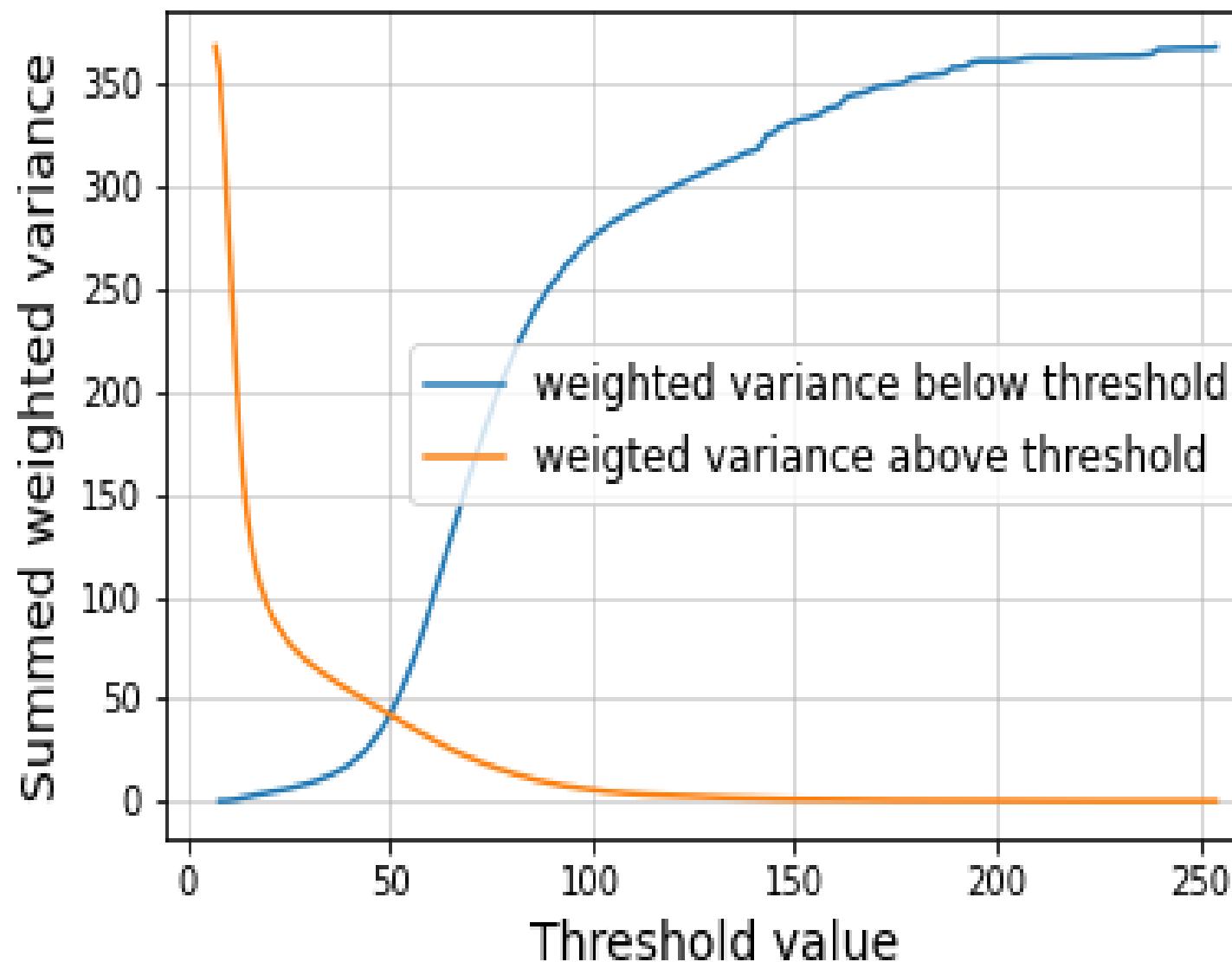
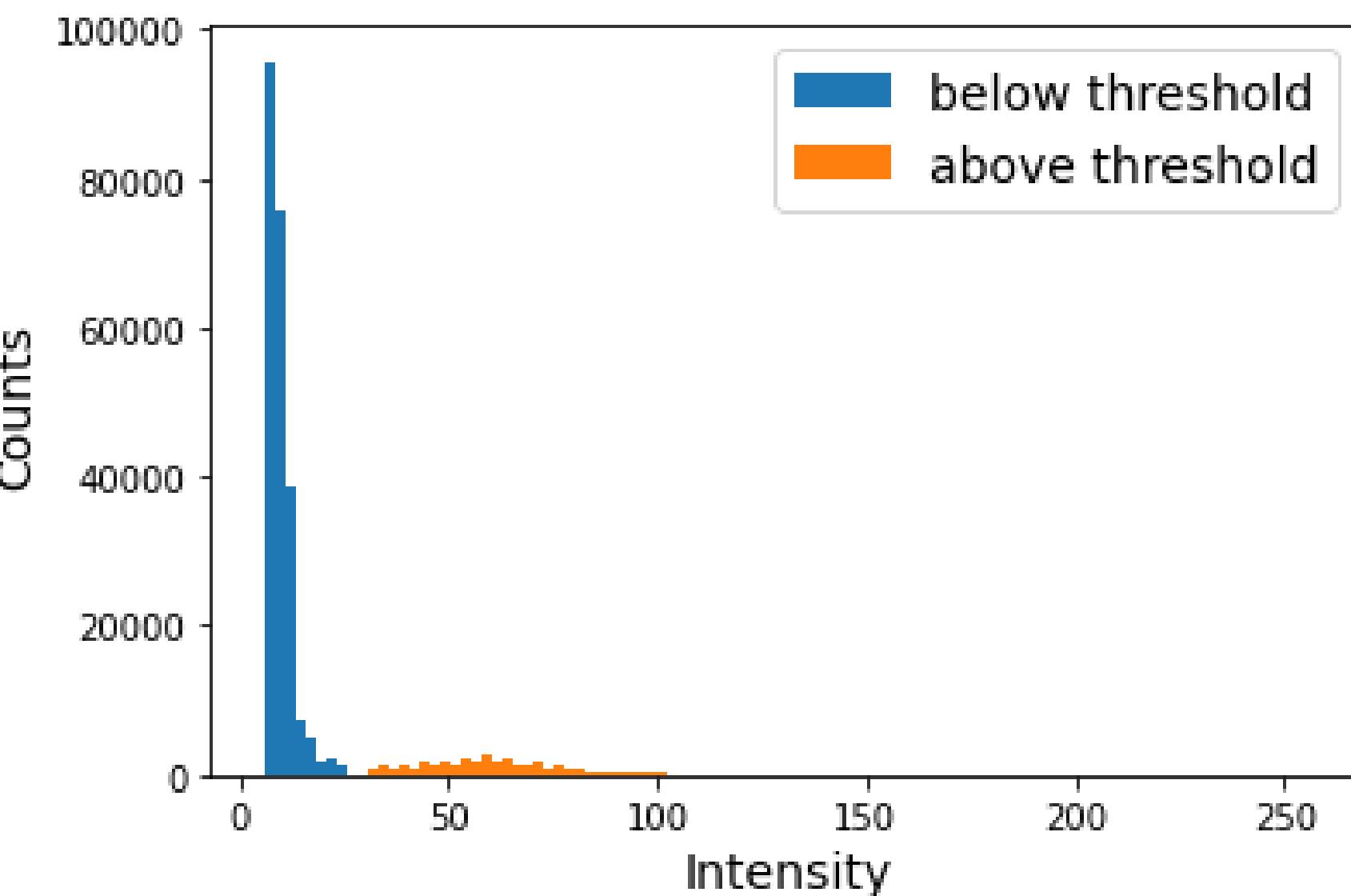


Threshold algorithms

Otsu-thresholding (Otsu et Al. 1979): Find threshold so that the summed, weighted variance $Var_{w,sum}$ becomes minimal:

$$Var(I) = \frac{1}{n_I} \sum (I - mean(I))^2 \rightarrow$$

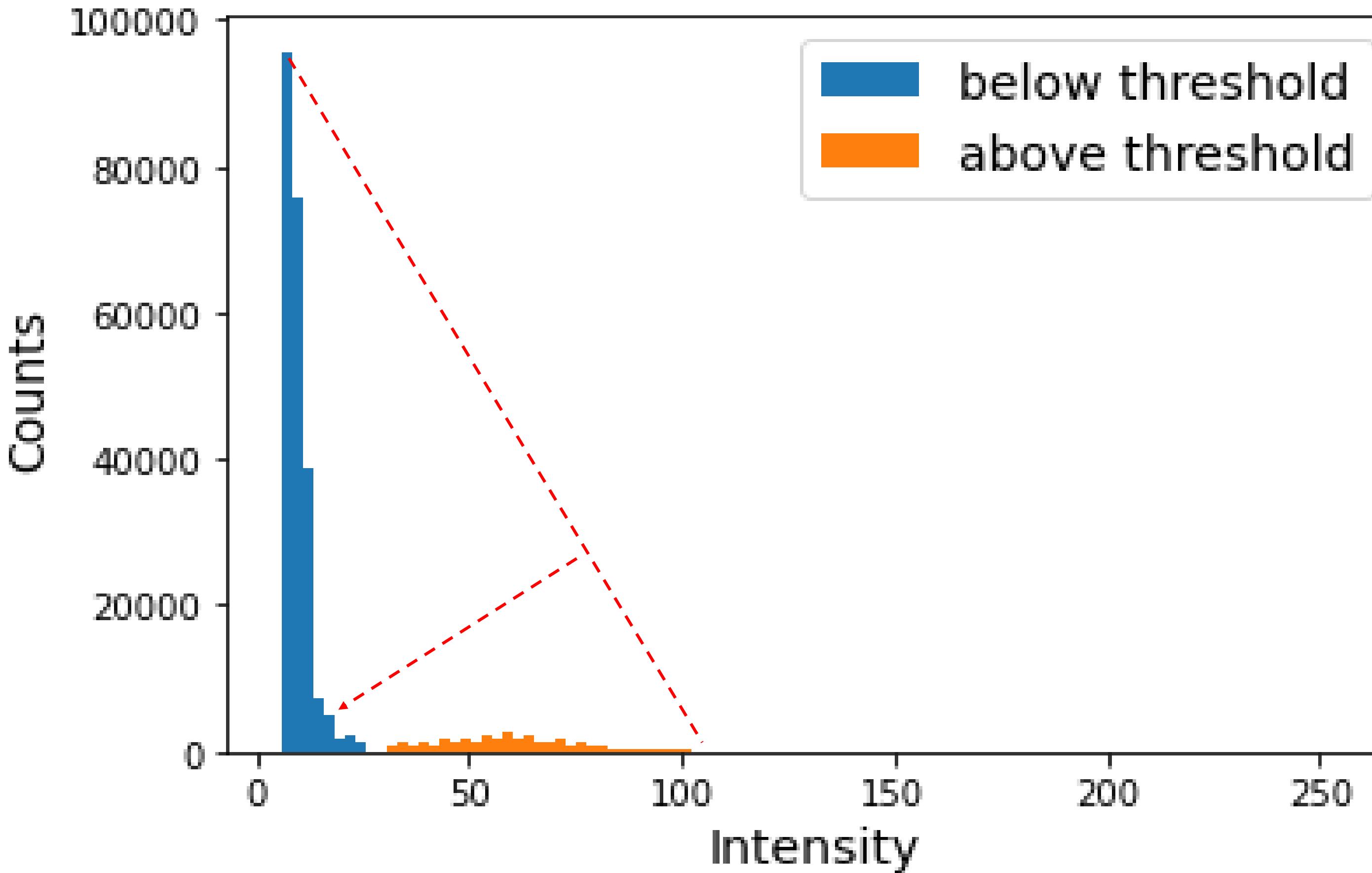
$$Var_{w,sum} = \frac{n_A}{n_I} \cdot Var(A) + \frac{n_B}{n_I} \cdot Var(B)$$



Statistical thresholding: Pixels above statistical parameter of I belong to foreground. (Possibilities: Mean, Median, Quartiles, etc.)

Threshold algorithms

Triangle thresholding: Draw a line between histogram point with max. counts and max. intensity and find point in histogram with maximal distance to this line (----)



Thresholding with scikit-image

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

Otsu-thresholding (Otsu et Al. 1979): Find threshold so that the summed, weighted variance $Var_{w,sum}$ becomes minimal.

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

Statistical thresholding: Pixels above statistical parameter of I belong to foreground.
(Possibilities: Mean, Median, Quartiles, etc.)

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

Triangle thresholding: Draw a line between histogram point with max. counts and max. intensity and find point in histogram with maximal distance to this line.

Explore more threshold options in scikit-image with:

```
from skimage import filters
```

```
threshold = filters.threshold_
```

f	threshold_isodata	function
f	threshold_li	function
f	threshold_local	function
f	threshold_mean	function
f	threshold_minimum	function
f	threshold_multithreshold	function
f	threshold_niblack	function
f	threshold_otsu	function
f	threshold_sauvola	function
f	threshold_triangle	function

Thresholding: Citing

Cite the thresholding method of your choice properly

We segmented the cell nuclei in the images using the Otsu 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

Thresholding: Pitfalls

```
binary = image > a_good_threshold_value_of_my_choice
```

Never use manual thresholding!

- Different observers come to different results when selecting a “good” threshold value
- You may come to different results when selecting a threshold value repeatedly

```
binary = image > threshold  
intensities = some_function_to_measure_intensities(binary, image)
```

Inter-observer variability

Intra-observer variability

Avoid thresholding an image and afterwards measure intensities in the same image

- You would measure the threshold you entered

```
binary_1 = image_1 > threshold_1(image_1)  
binary_2 = image_2 > threshold_2(image_2)
```

Chose one threshold algorithm:

...and stick to it for the whole study. Using a new method for every image impairs reproducibility!

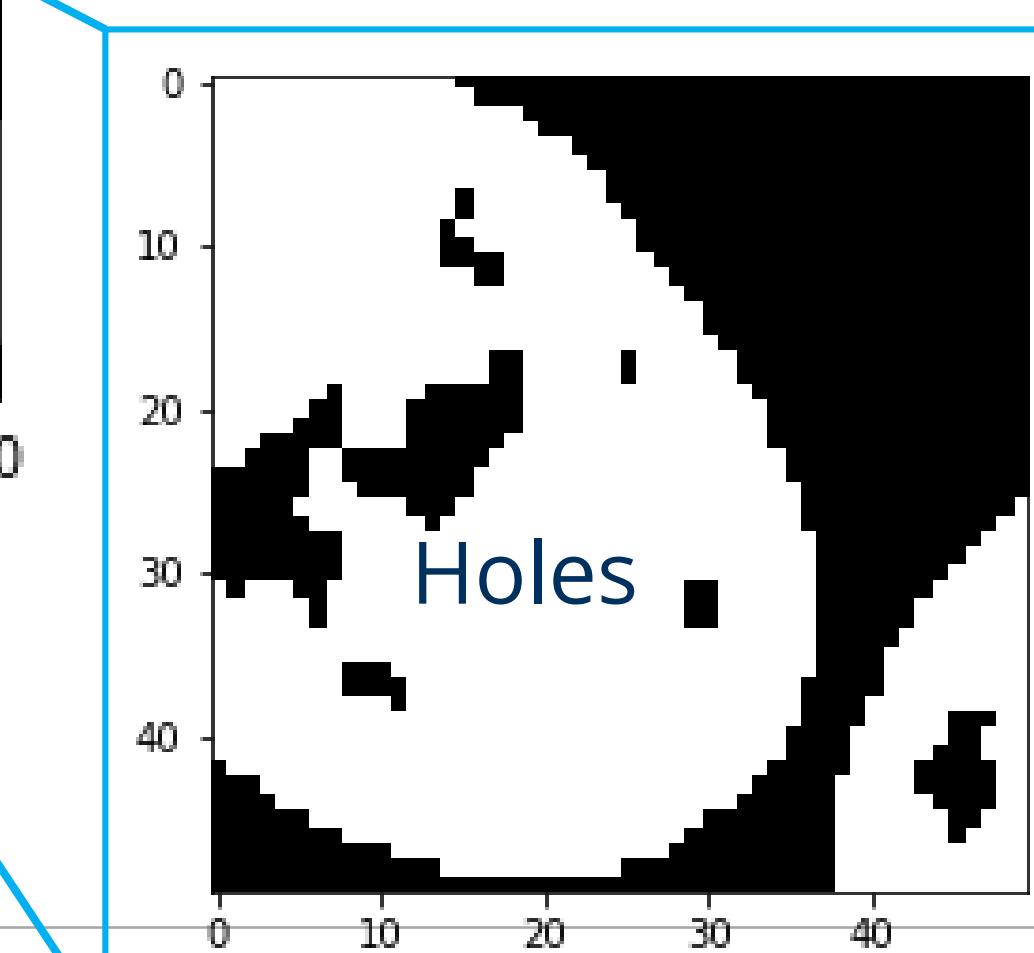
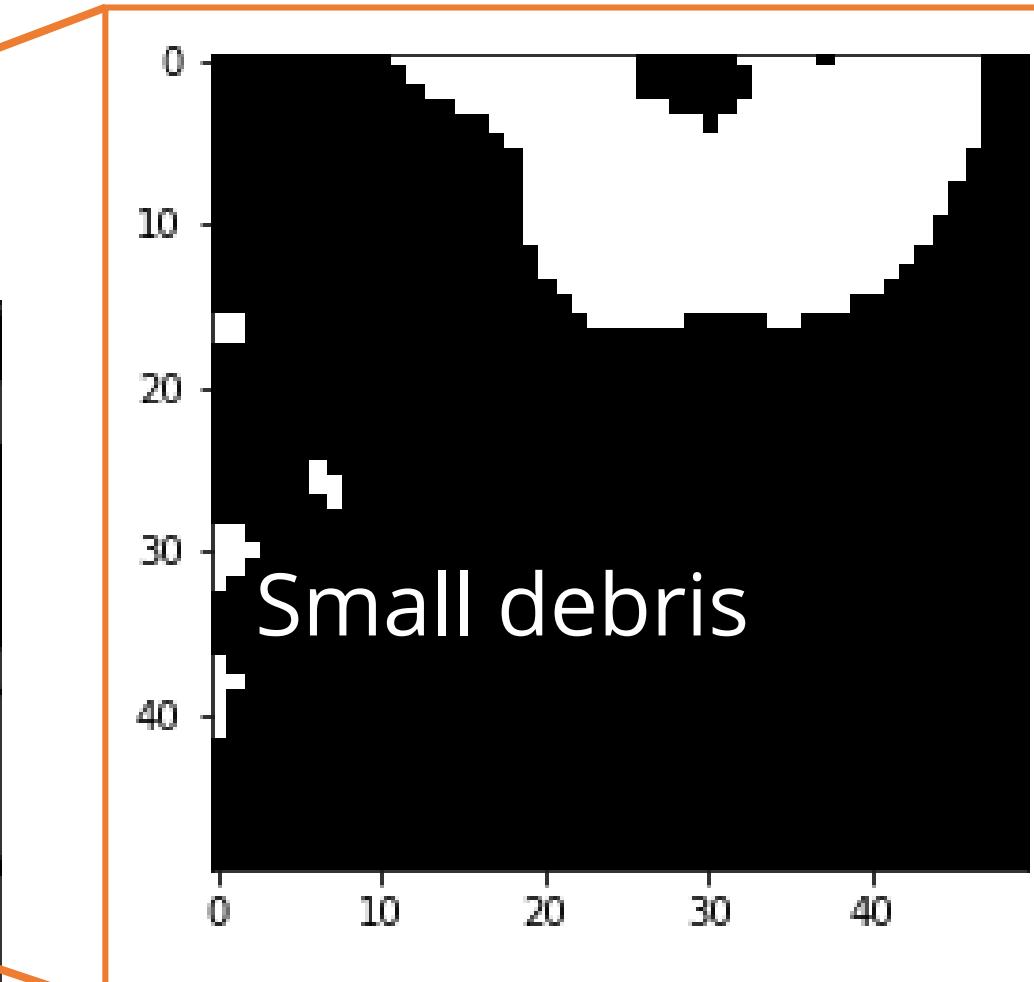
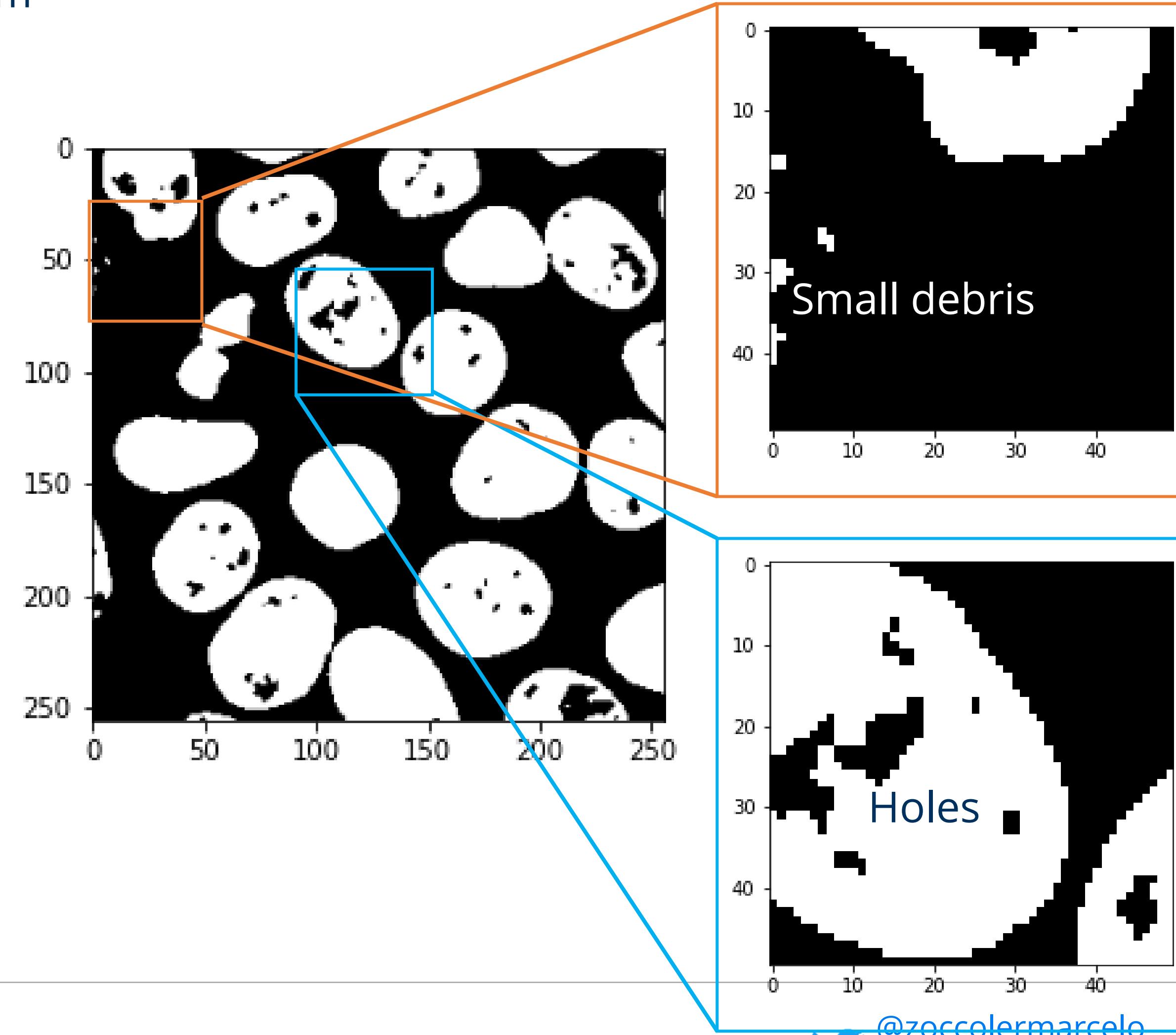
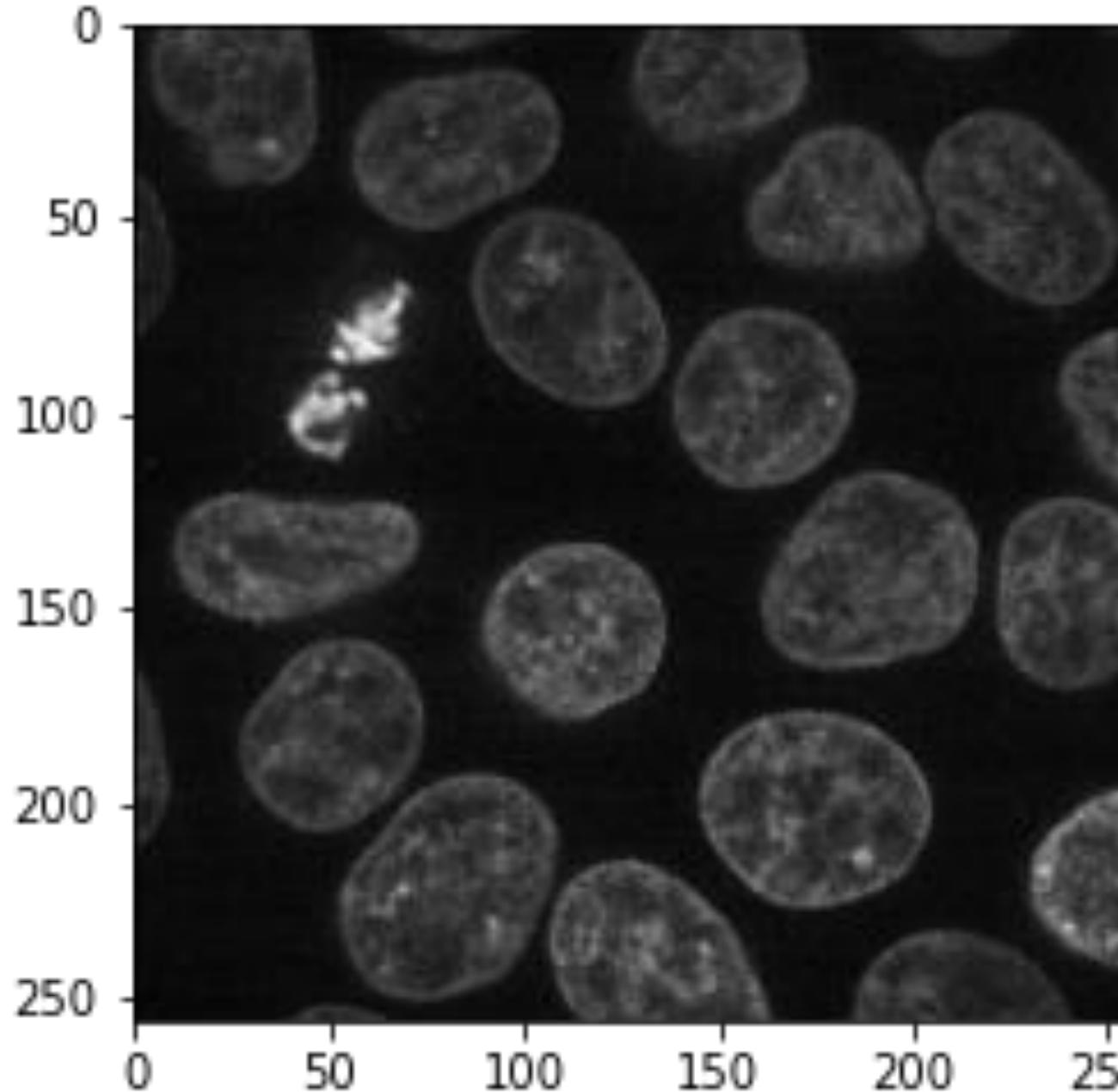
Do not over-engineer

There will be always images where thresholding fails – better report the errors!

Refining masks

Binary mask images may not be perfect immediately after thresholding.

There are ways of refining them

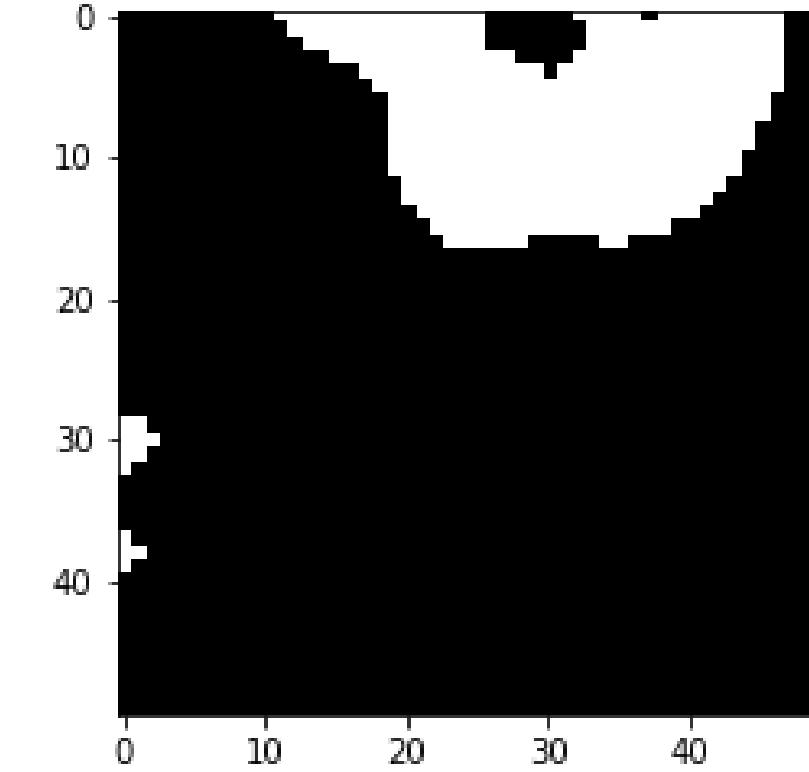
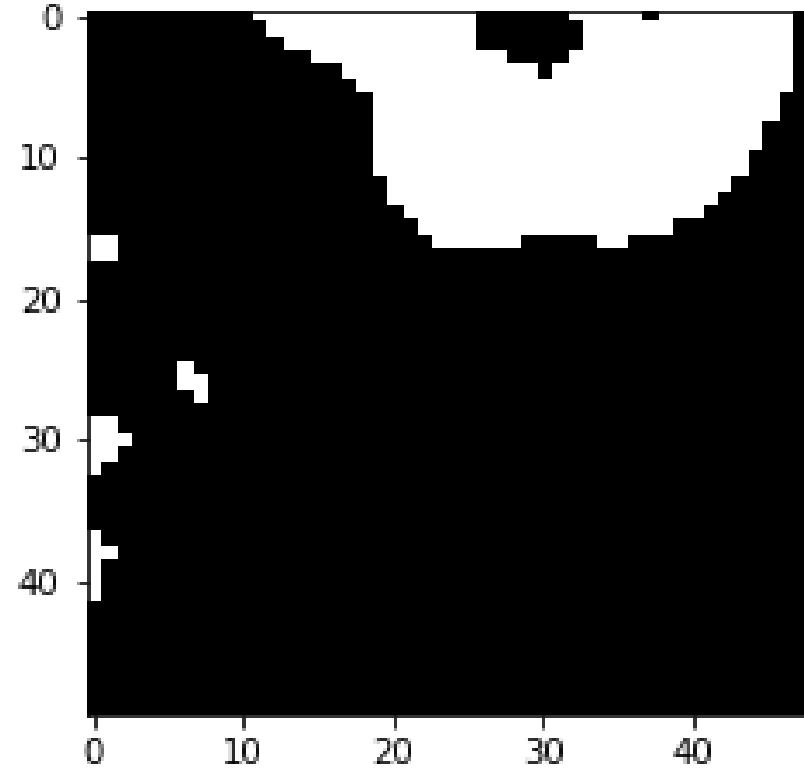


@zoccolermarcelo
@jm_mightypirate

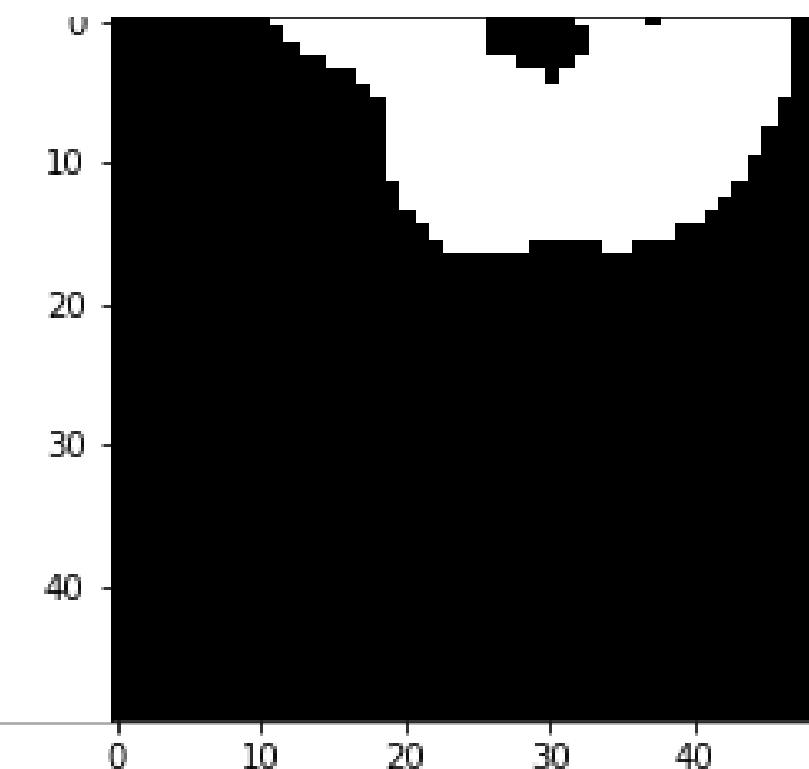
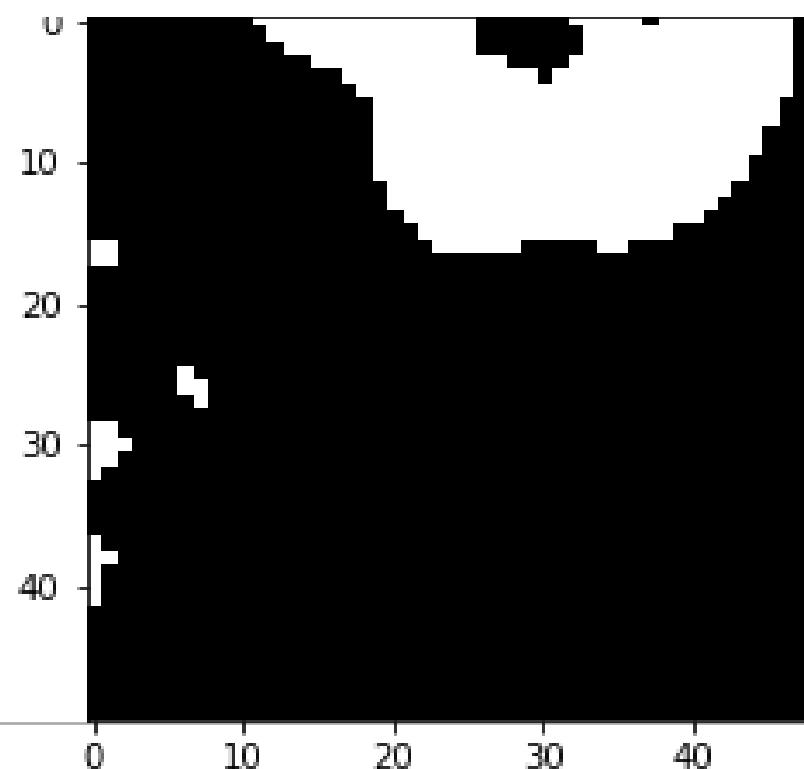
Refining masks: Opening & Closing

```
from skimage import morphology
```

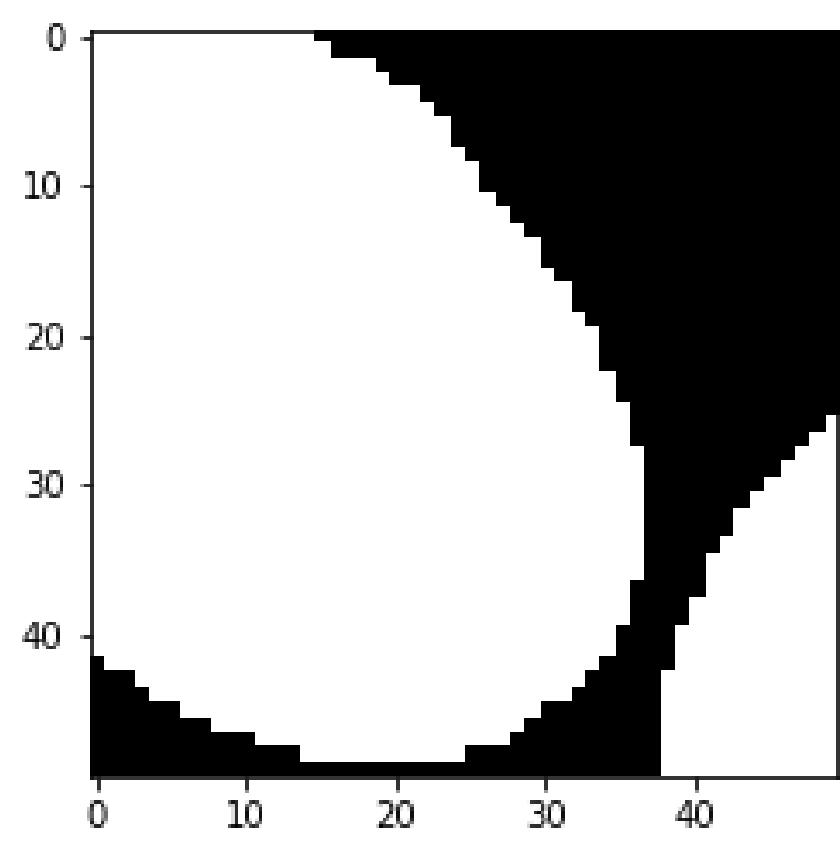
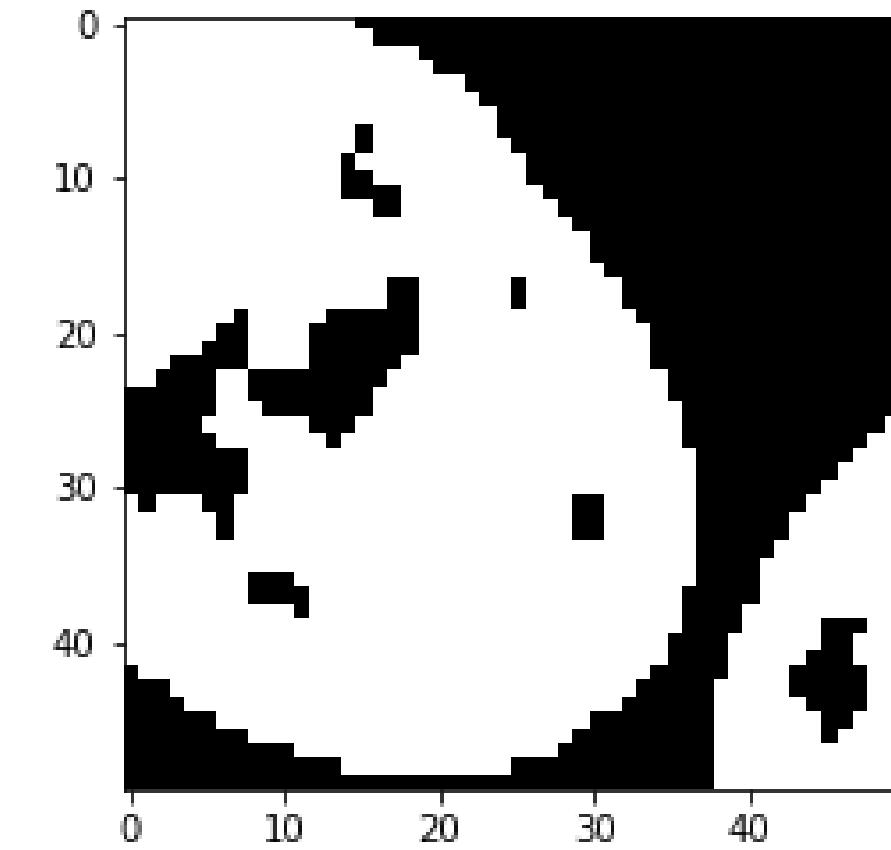
```
opened = morphology.binary_opening(binary)
```



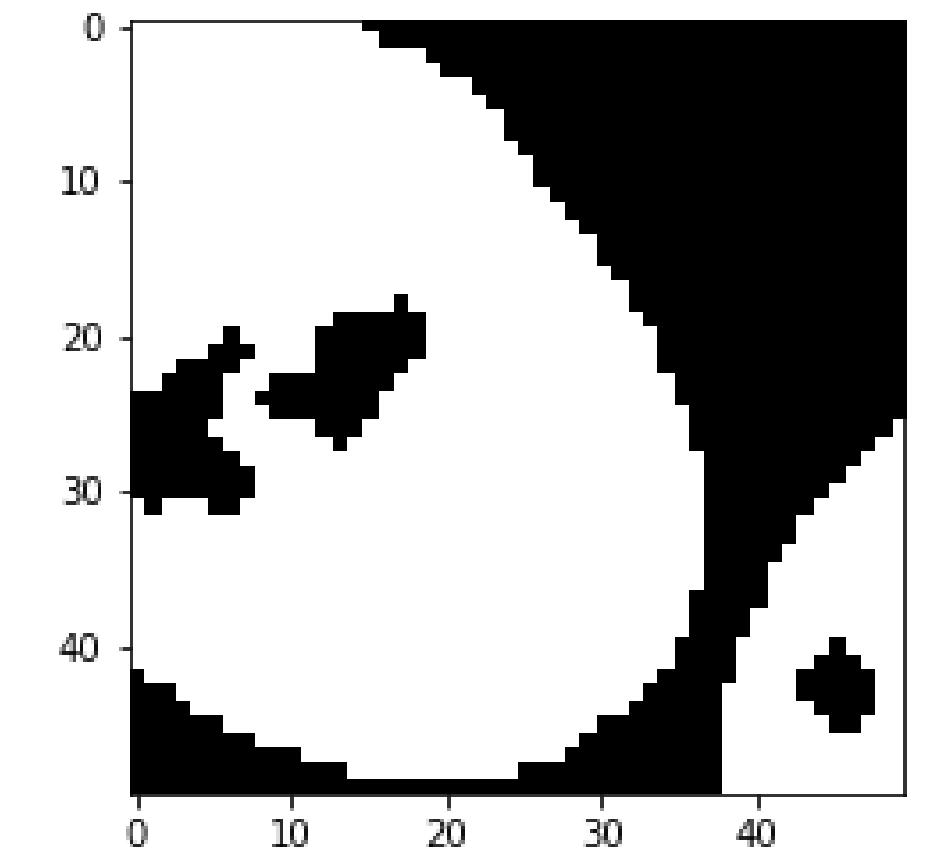
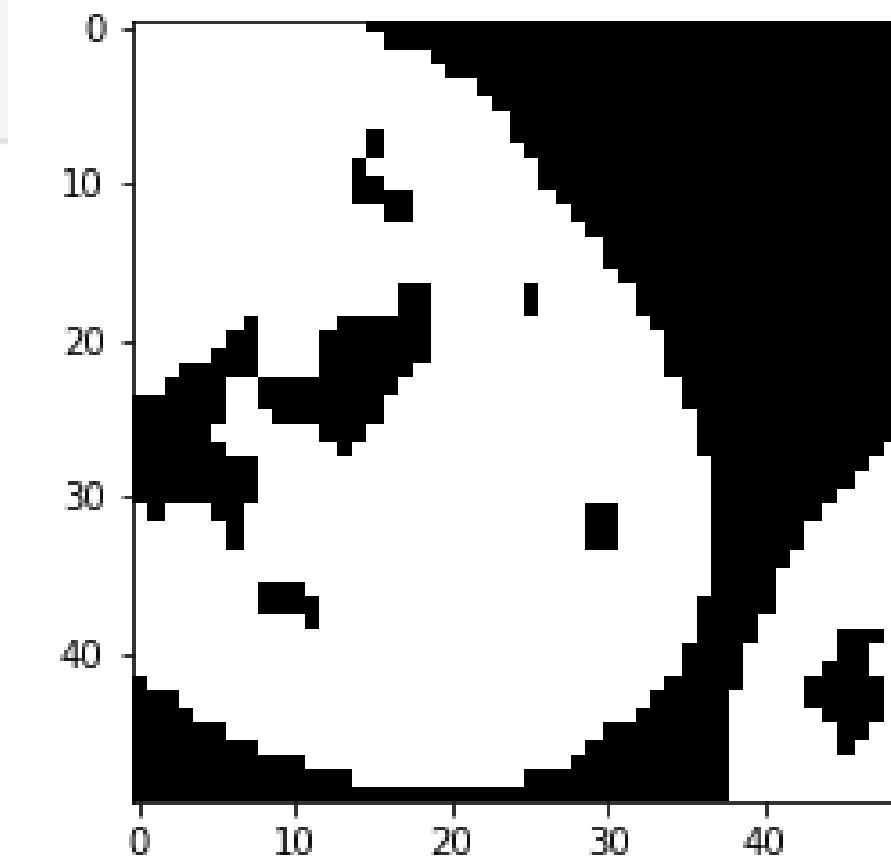
```
opened = morphology.area_opening(binary, area_threshold=20)
```



```
closed = morphology.area_closing(binary, area_threshold=100)
```



```
closed = morphology.binary_closing(binary)
```

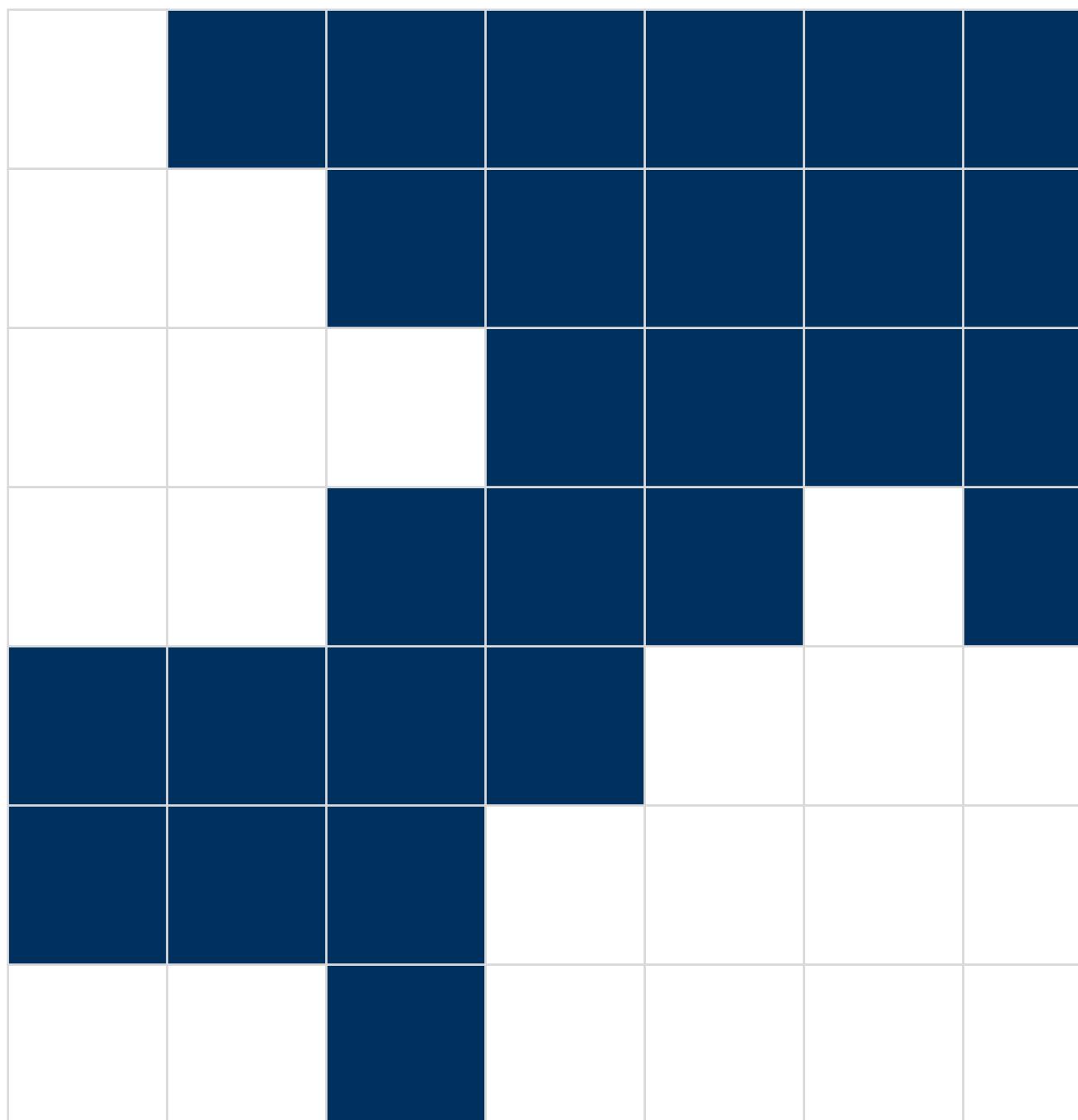


Connected components labelling

In order to allow the computer differentiating objects, connected components analysis (CCA) is used to mark pixels belonging to different objects with different numbers

Background pixels are marked with 0.

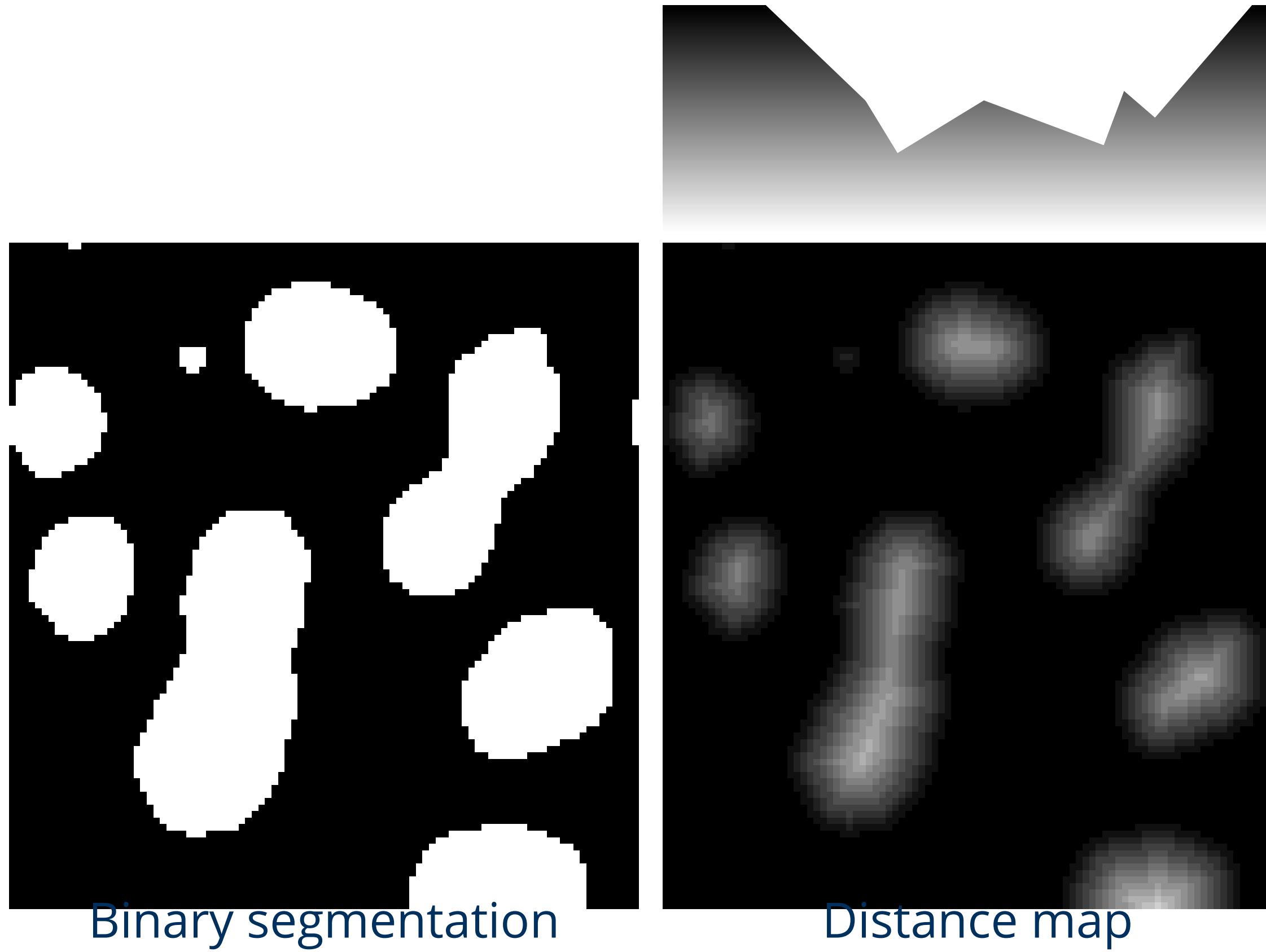
The maximum intensity of a labelled map corresponds to the number of objects.



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

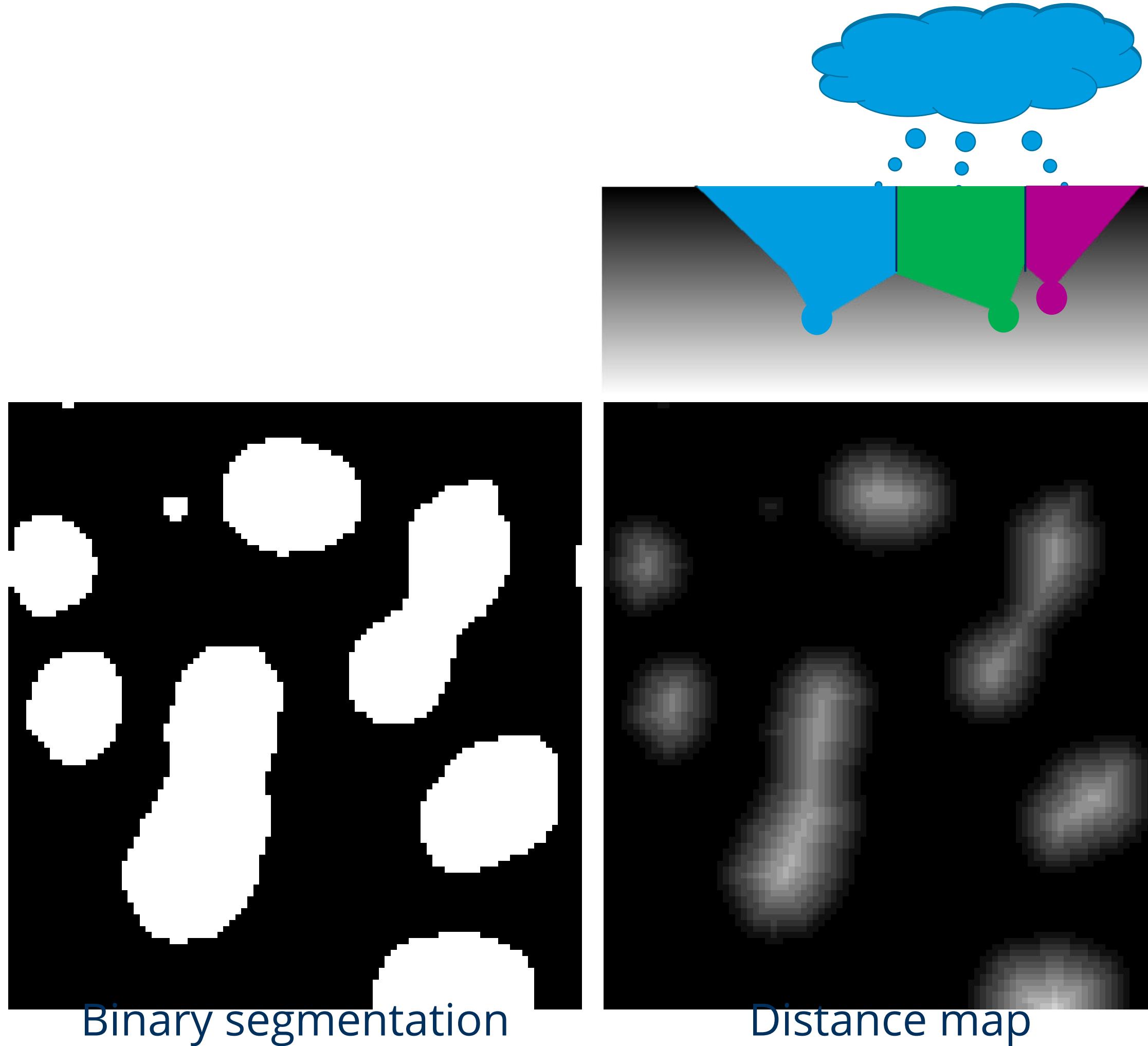
Watershed

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



Watershed

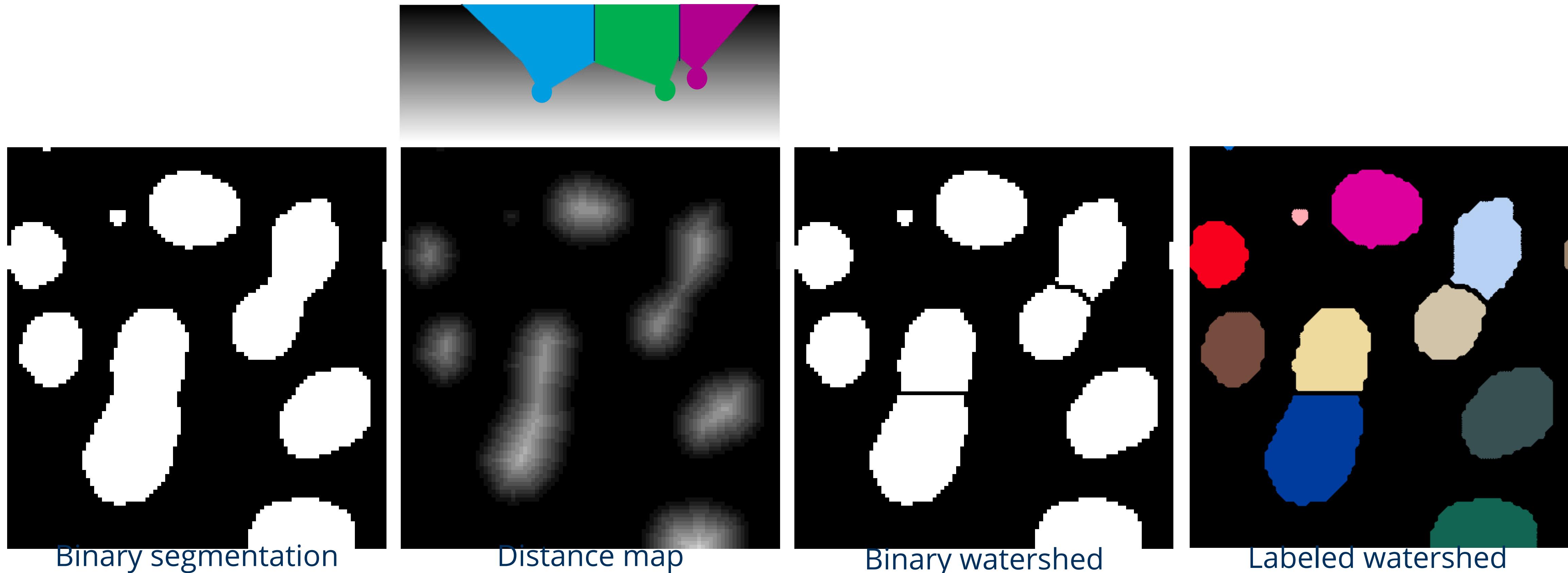
The watershed algorithm for binary images allows cutting one object into two where it's reasonable.



Watershed

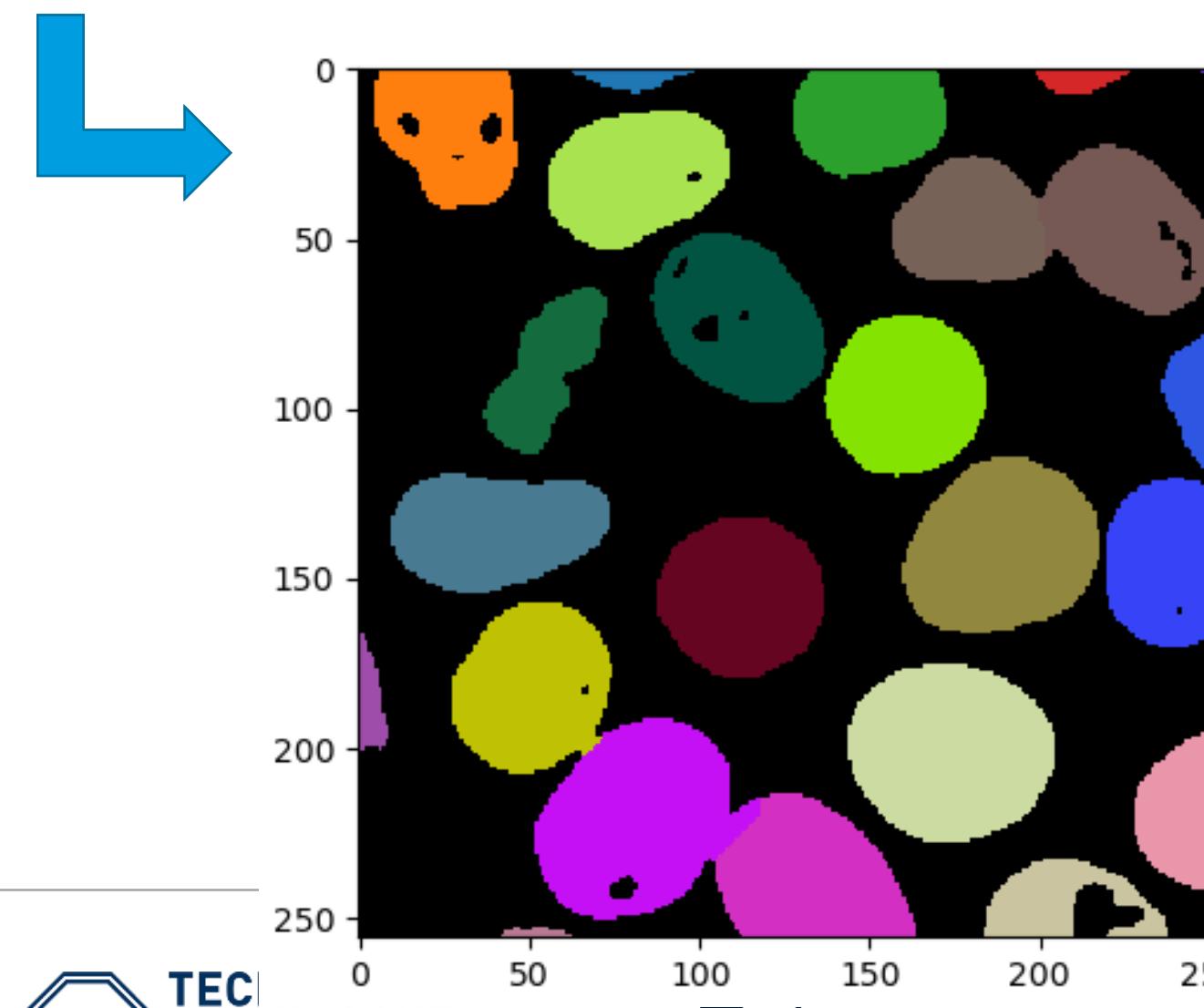
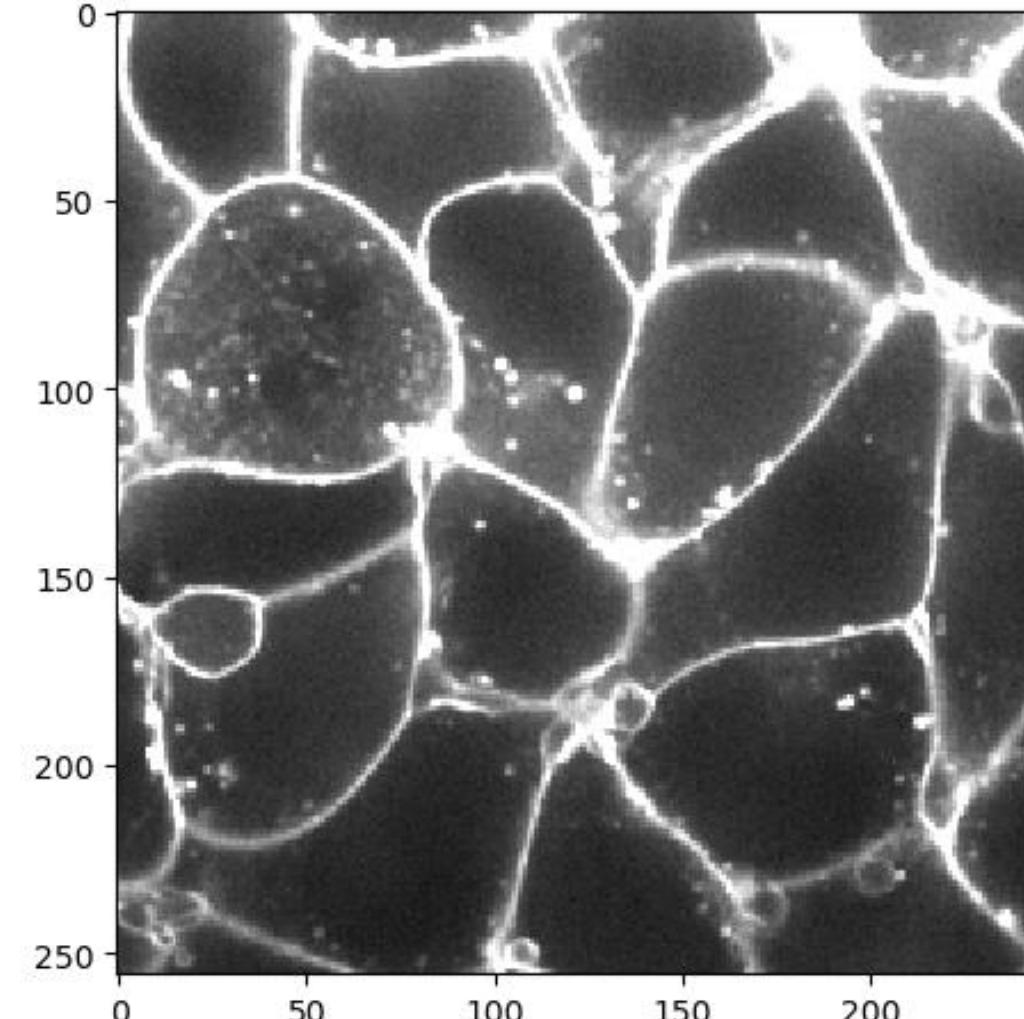
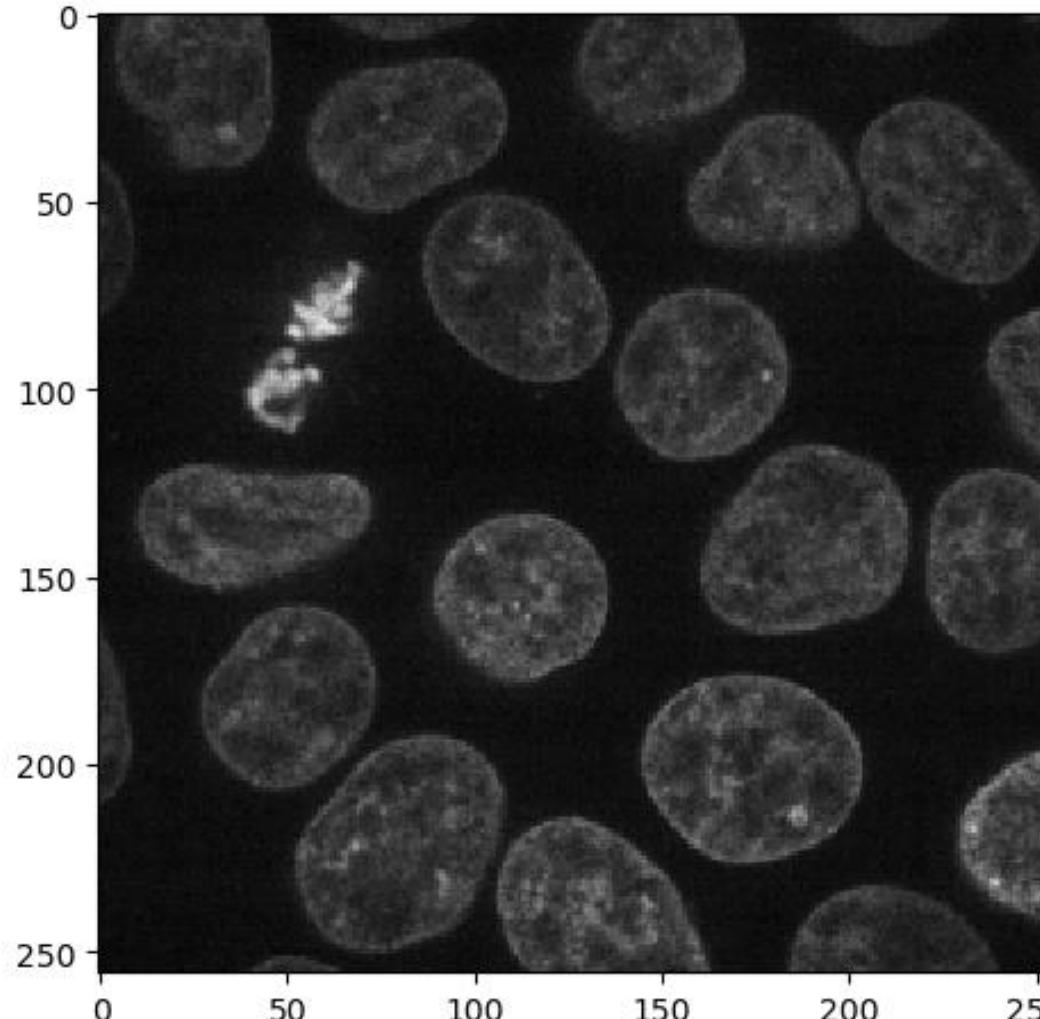
The watershed algorithm for binary images allows cutting one object into two where it's reasonable.

The watersheds are made from binary images. The algorithm does not take the original image into account!

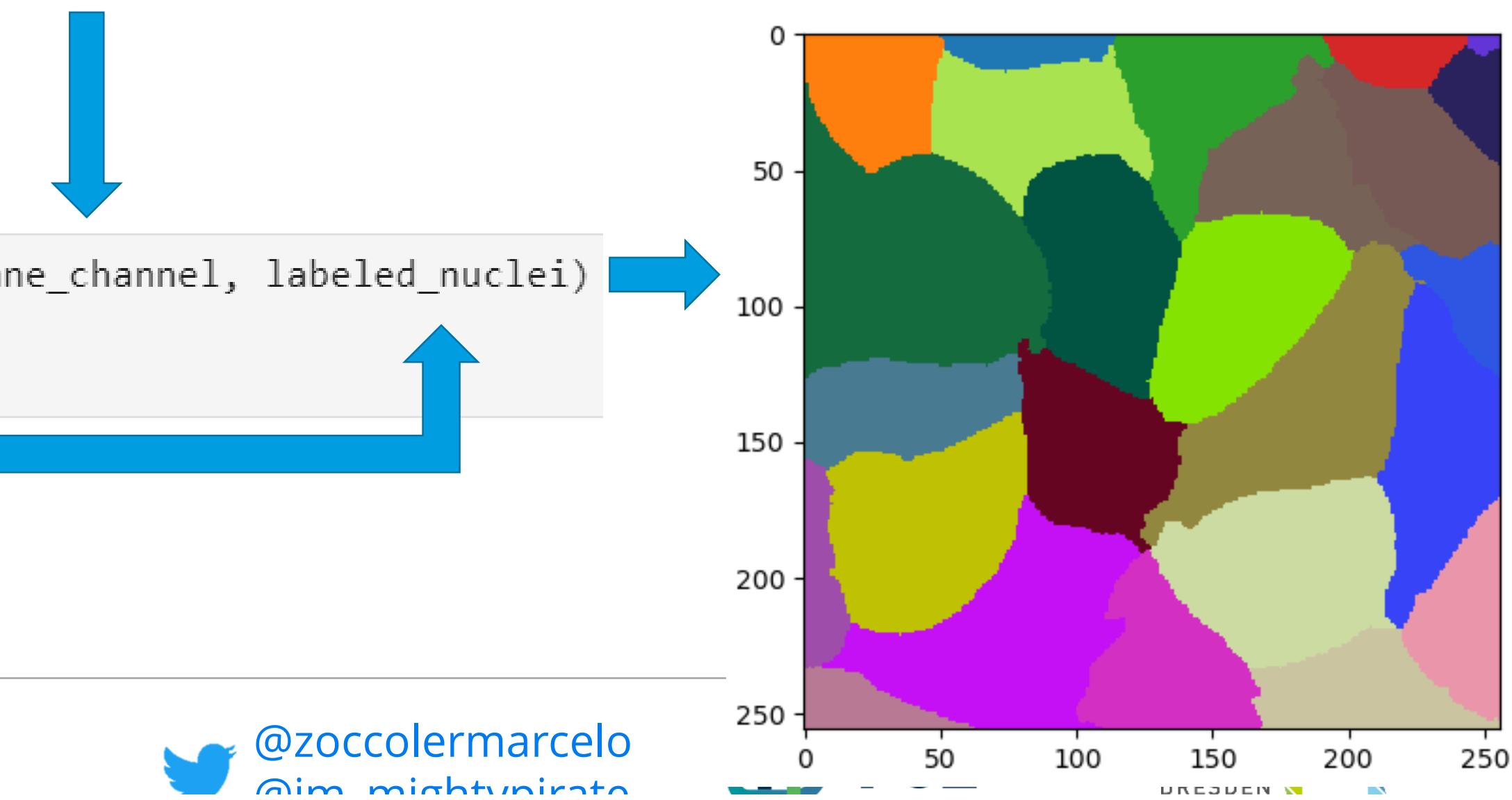


Watershed

... in Python practice



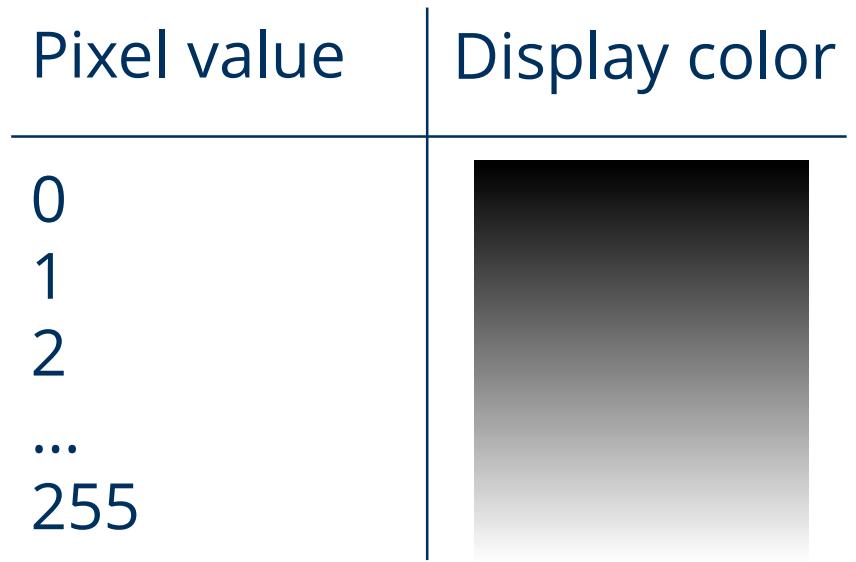
```
labeled_cells = seeded_watershed(membrane_channel, labeled_nuclei)  
labeled_cells
```



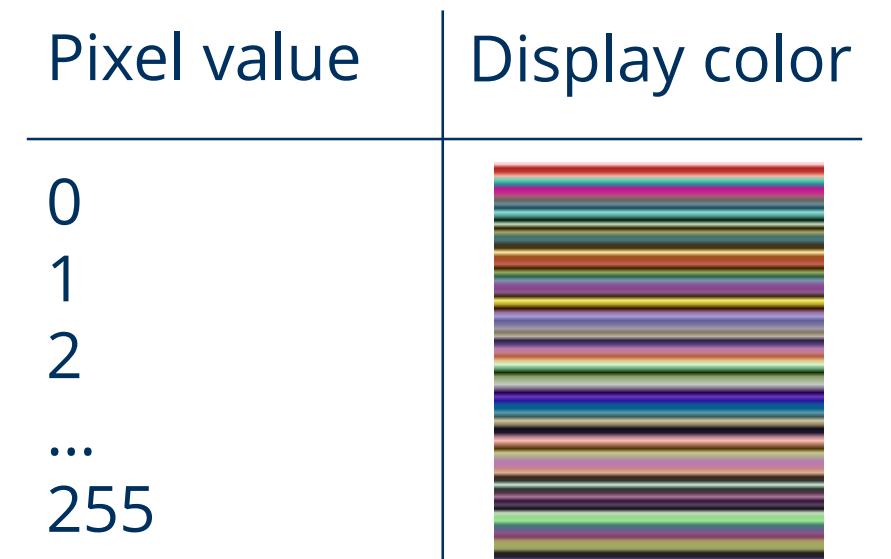
Label map visualisation

Label maps can be nicely visualized with the right lookup table

Grey



Glasbey



Reminder: Vocabulary

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

Instance segmentation

Label 1 = Nucleus #1

Label 2 = Nucleus #2

Label 3 = Nucleus #3

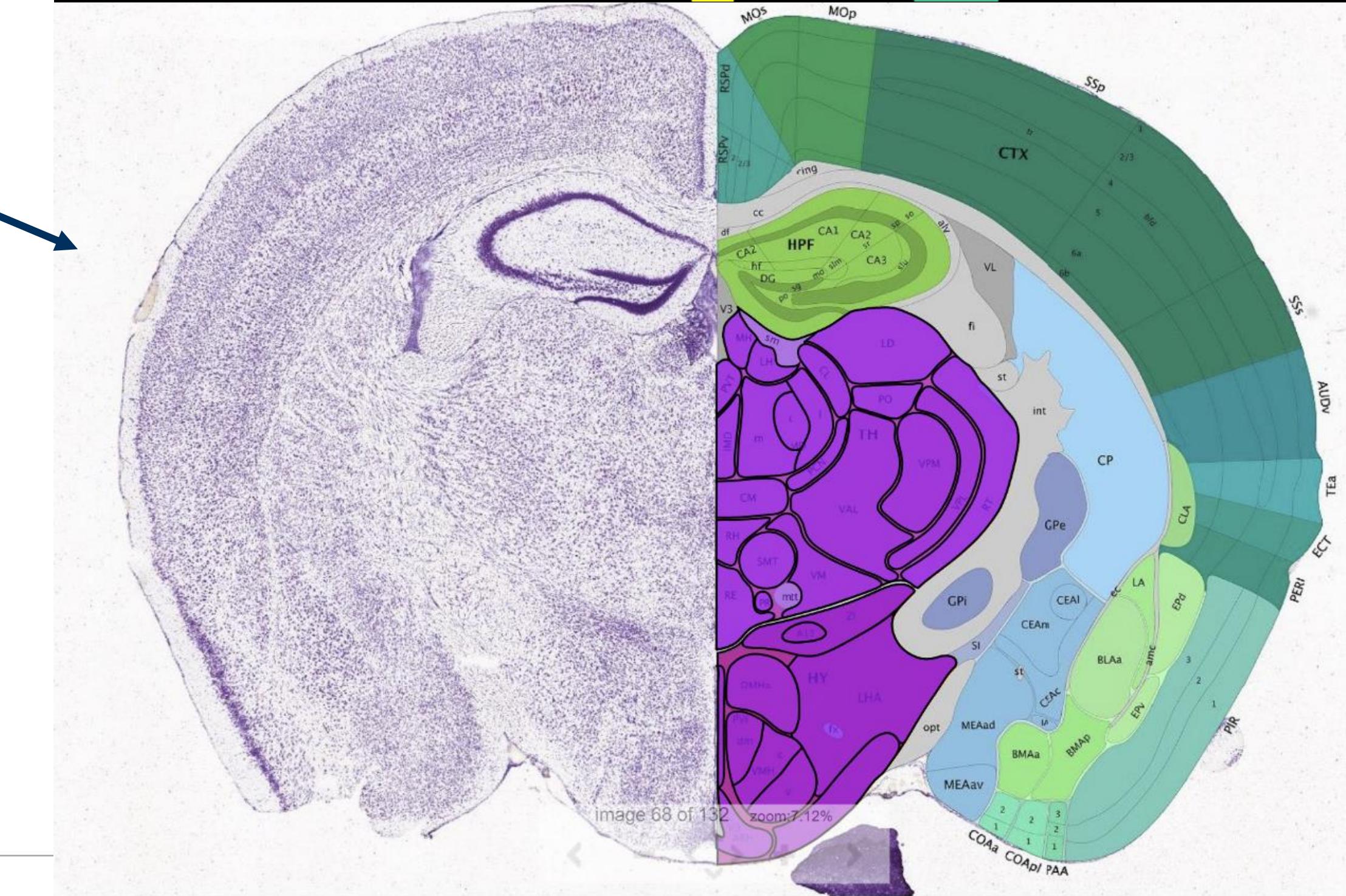


Semantic segmentation

Label 1 = Cortex

Label 2 = Cerebellum

Label 3 = Hippocampus



Allen adult mouse atlases,
<https://atlas.brain-map.org>

Feature extraction

A *feature* is a countable or measurable property of an image or object.

Goal of feature extraction is finding a minimal set of features to describe an object well enough to differentiate it from other objects.

- **Intensity based**

- Mean intensity
- Standard deviation
- Total intensity
- Textures

- **Shape based /spatial**

- Area / Volume
- Roundness
- Solidity
- Circularity / Sphericity
- Elongation
- Centroid
- Bounding box

- **Spatio-temporal**

- Displacement,
- Speed,
- Acceleration

- **Others**

- Overlap
- Colocalization
- Neighborhood

- **Mixed features**

- Center of mass
- Local minima / maxima

Intensity based features

Min / max

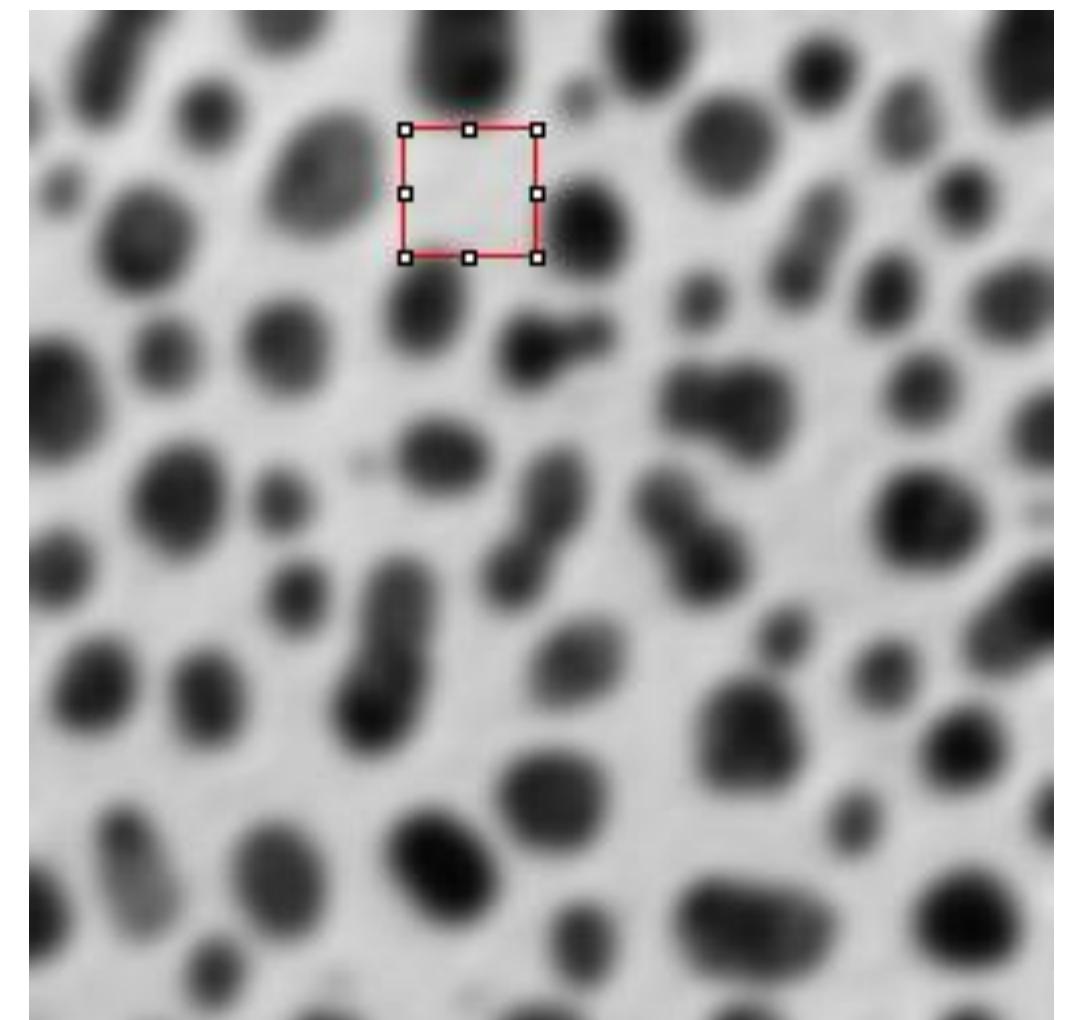
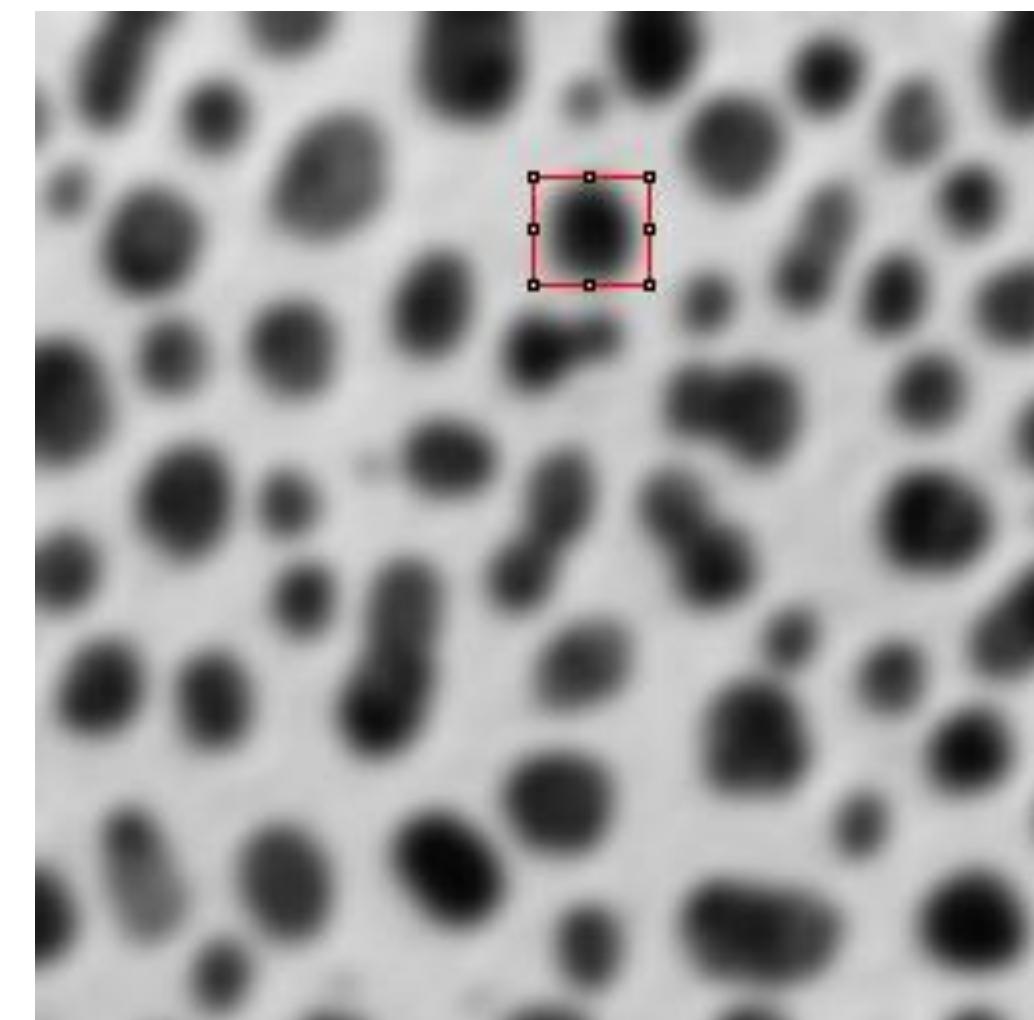
Median

Mean

Mode

Variance

Standard deviation

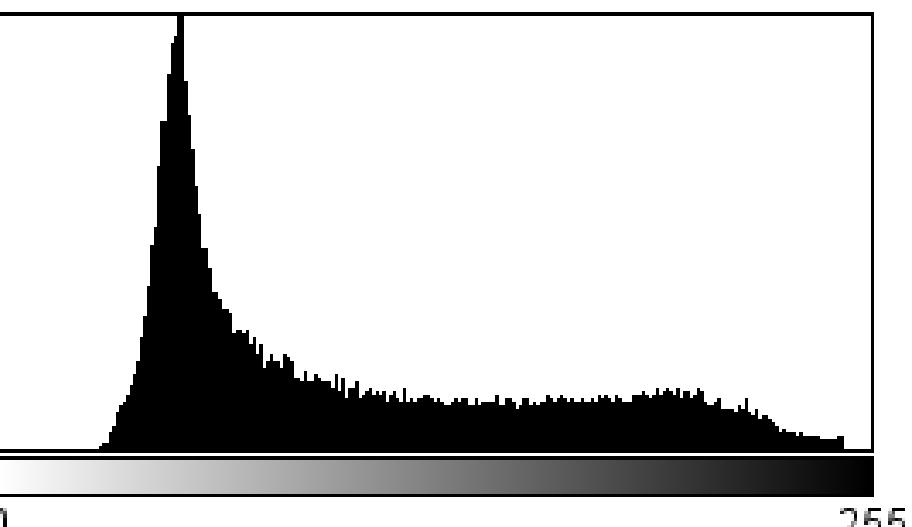


Can be derived from pixel values

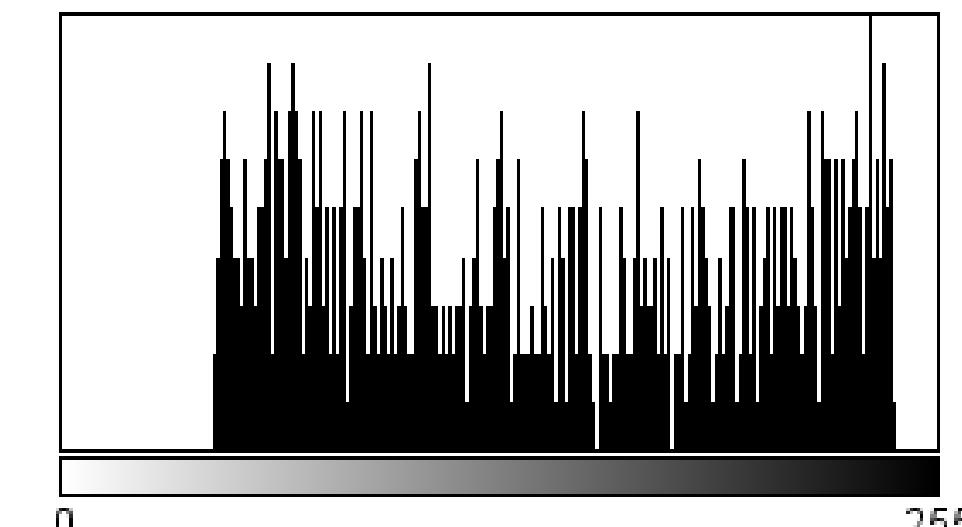
Don't take spatial relationship of pixels
into account

See also:

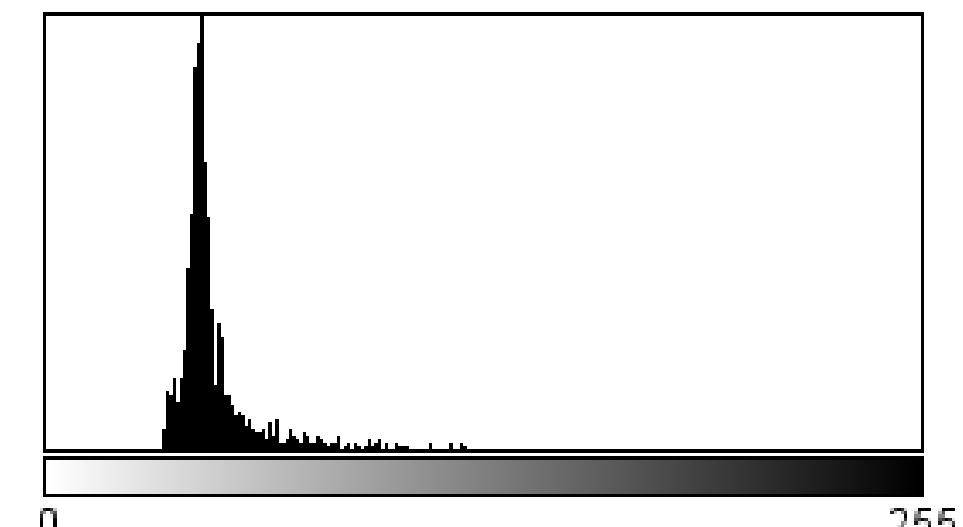
- descriptive statistics
- histogram



Count: 65024 Min: 29
Mean: 103.301 Max: 248
StdDev: 57.991 Mode: 53 (1663)



Count: 783 Min: 44
Mean: 141.308 Max: 243
StdDev: 61.876 Mode: 236 (9)



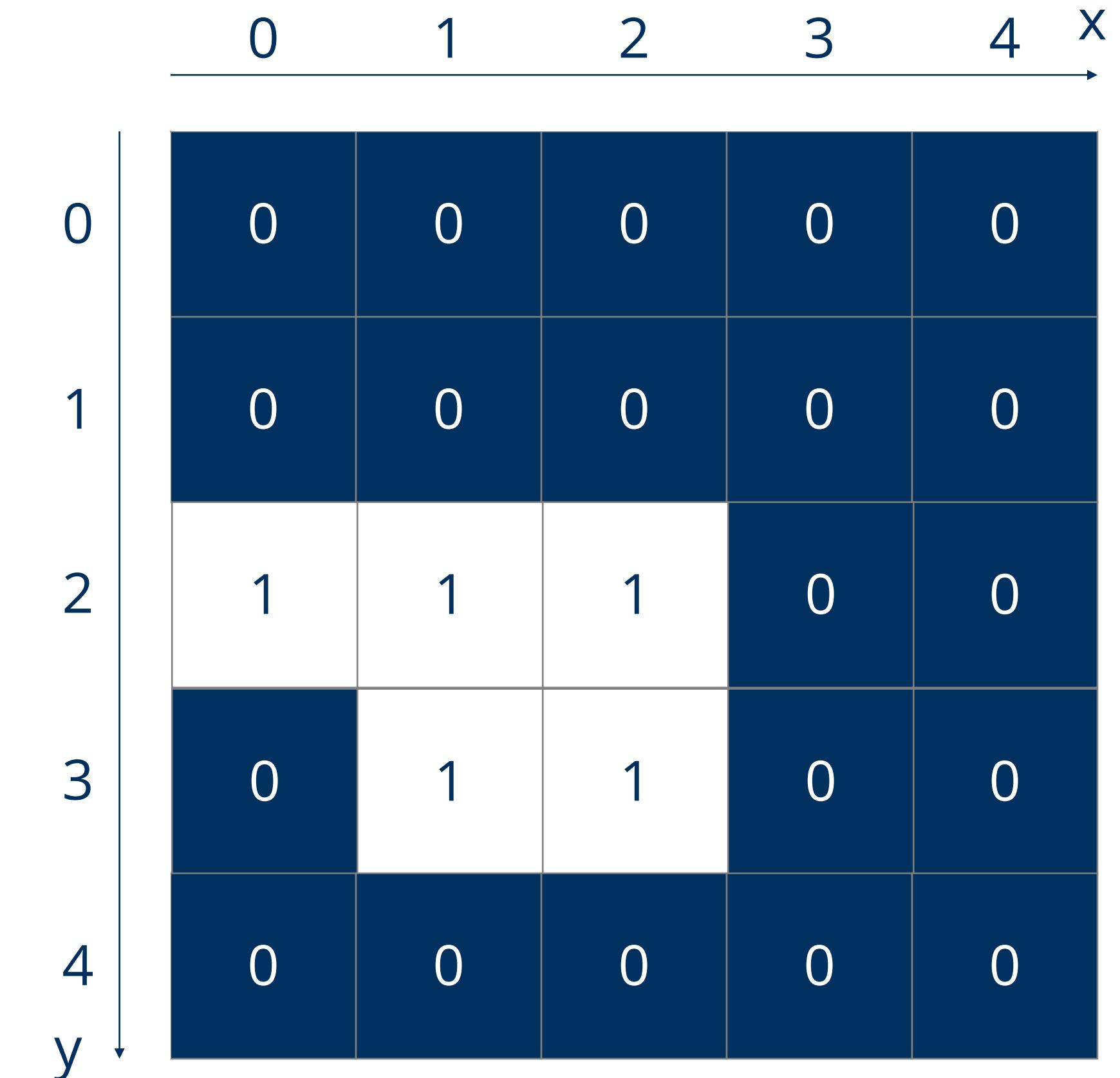
Count: 1056 Min: 34
Mean: 49.016 Max: 122
StdDev: 12.685 Mode: 45 (120)

Bounding rectangle / bounding box

Position and size of the smallest rectangle containing all pixels of an object

- x_b, y_b ... position of the bounding box
- w_b ... width of the bounding box
- h_b ... height of the bounding box

variable	value
x_b	0
y_b	2
w_b	3



Center of mass

Relative position in an image weighted by pixel intensities

- x, y ... pixel coordinates
- w ... image width
- h ... image height
- μ ... mean intensity
- $g_{x,y}$... pixel grey value
- x_m, y_m ... center of mass coordinates

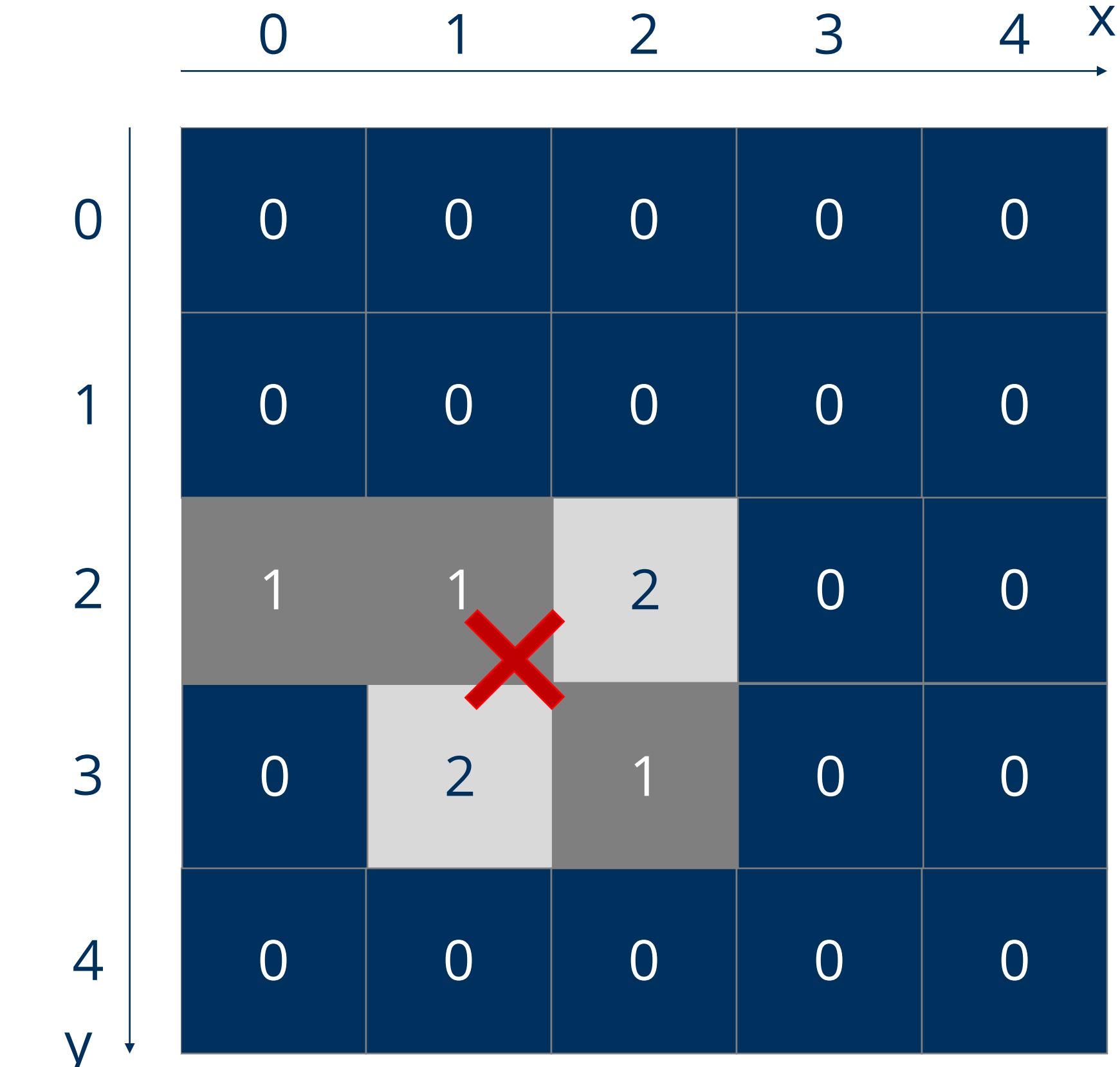
$$\mu = \frac{1}{wh} \sum_{y=0}^{h-1} \sum_{x=0}^{w-1} g_{x,y}$$

$$x_m = \frac{1}{wh\mu} \sum_{y=0}^{h-1} \sum_{x=0}^{w-1} x g_{x,y}$$

“sum intensity”

“total intensity”

$$y_m = \frac{1}{wh\mu} \sum_{y=0}^{h-1} \sum_{x=0}^{w-1} y g_{x,y}$$



$$x_m = 1/7 (1 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 + 2 \cdot 1 + 1 \cdot 2) = 1.3$$

$$y_m = 1/7 (1 \cdot 2 + 1 \cdot 2 + 2 \cdot 3 + 2 \cdot 2 + 1 \cdot 3) = 2.4$$

Center of geometry / centroid

Relative position in an image weighted by pixel intensities

Special case of center of mass for binary images

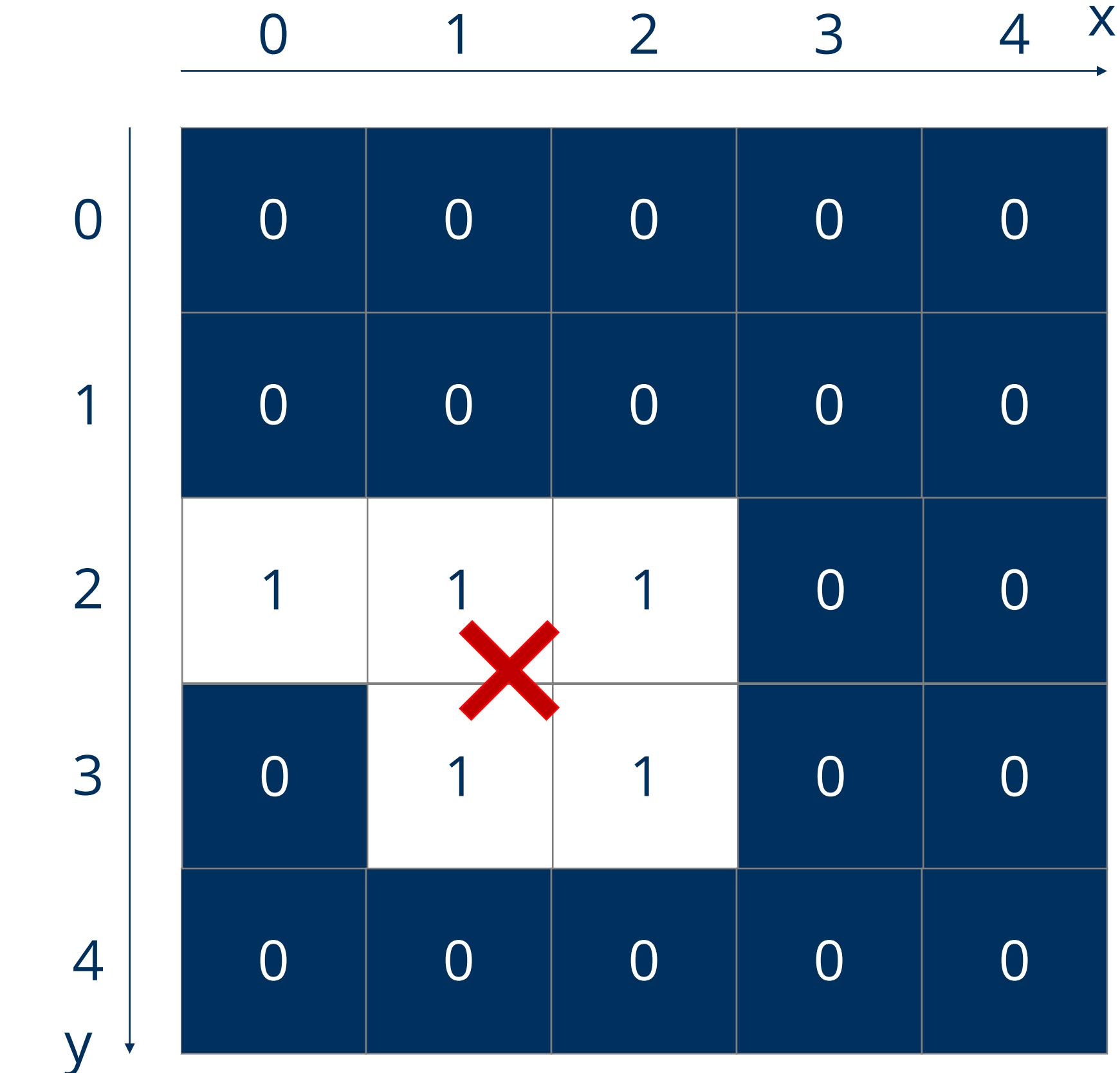
- x, y ... pixel coordinates
- w ... image width
- h ... image height
- μ ... mean intensity
- $g_{x,y}$... pixel grey value, integer in range [0;1]
- x_m, y_m ... center of mass coordinates

$$\mu = \frac{1}{wh} \sum_{y=0}^{h-1} \sum_{x=0}^{w-1} g_{x,y}$$

$$x_m = \frac{1}{wh\mu} \sum_{y=0}^{h-1} \sum_{x=0}^{w-1} x g_{x,y}$$

Number of white pixels

$$y_m = \frac{1}{wh\mu} \sum_{y=0}^{h-1} \sum_{x=0}^{w-1} y g_{x,y}$$



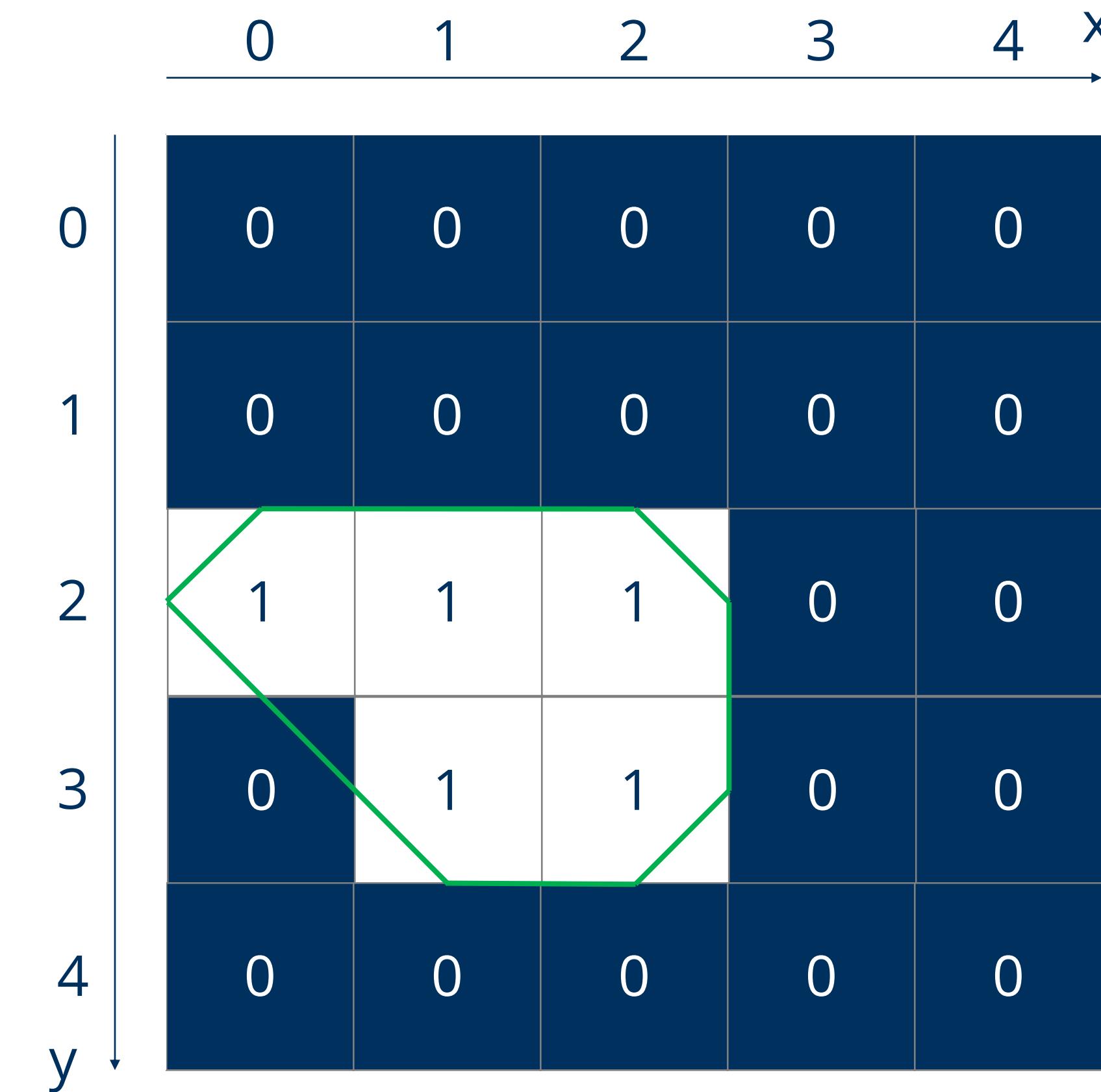
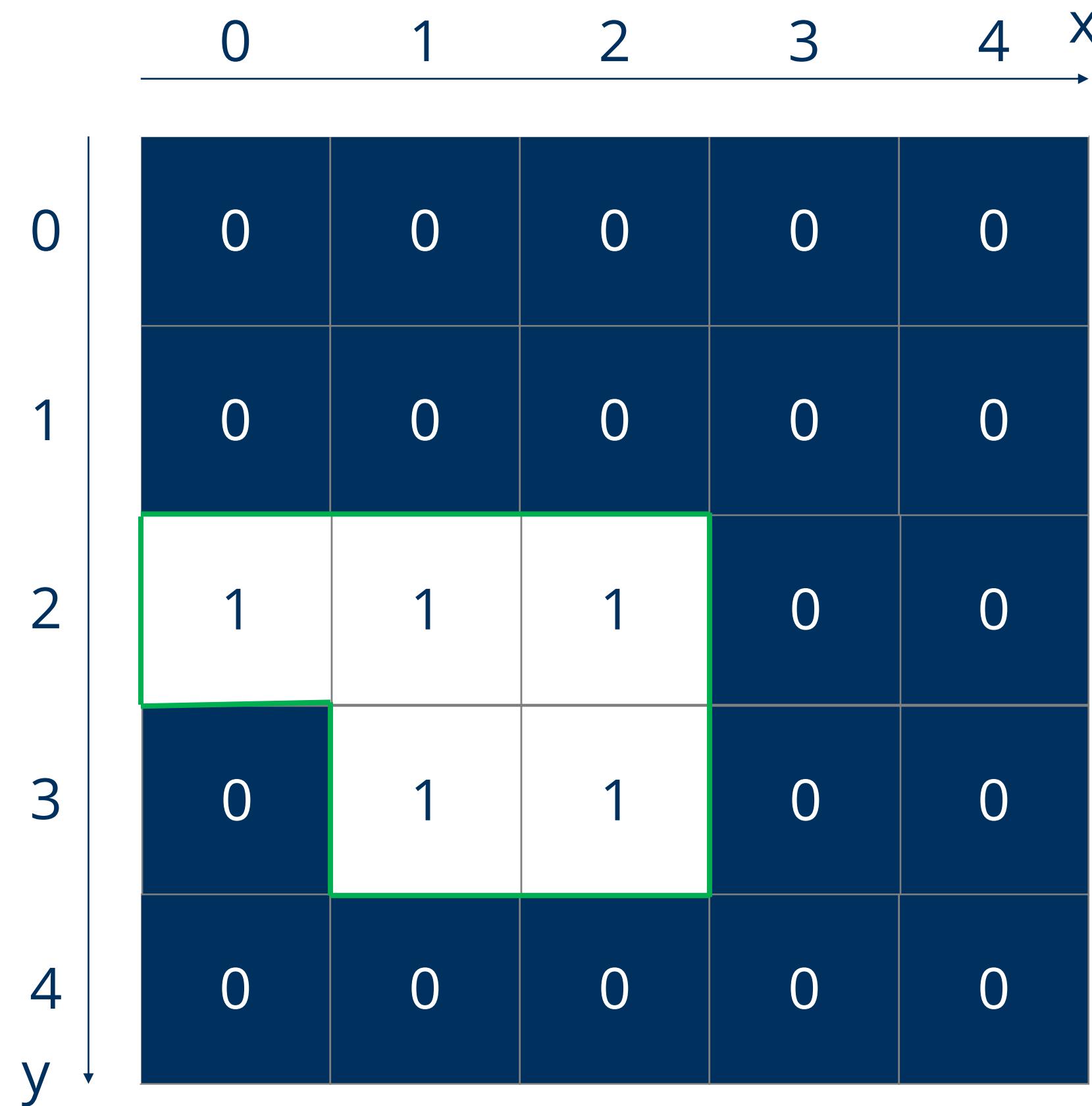
$$x_m = 1/5 (1 \cdot 0 + 1 \cdot 1 + 1 \cdot 2 + 1 \cdot 1 + 1 \cdot 2) = 1.2$$

$$y_m = 1/5 (1 \cdot 2 + 1 \cdot 2 + 1 \cdot 3 + 1 \cdot 2 + 1 \cdot 3) = 2.4$$

Perimeter

Length of the outline around an object

Depends on the actual implementation



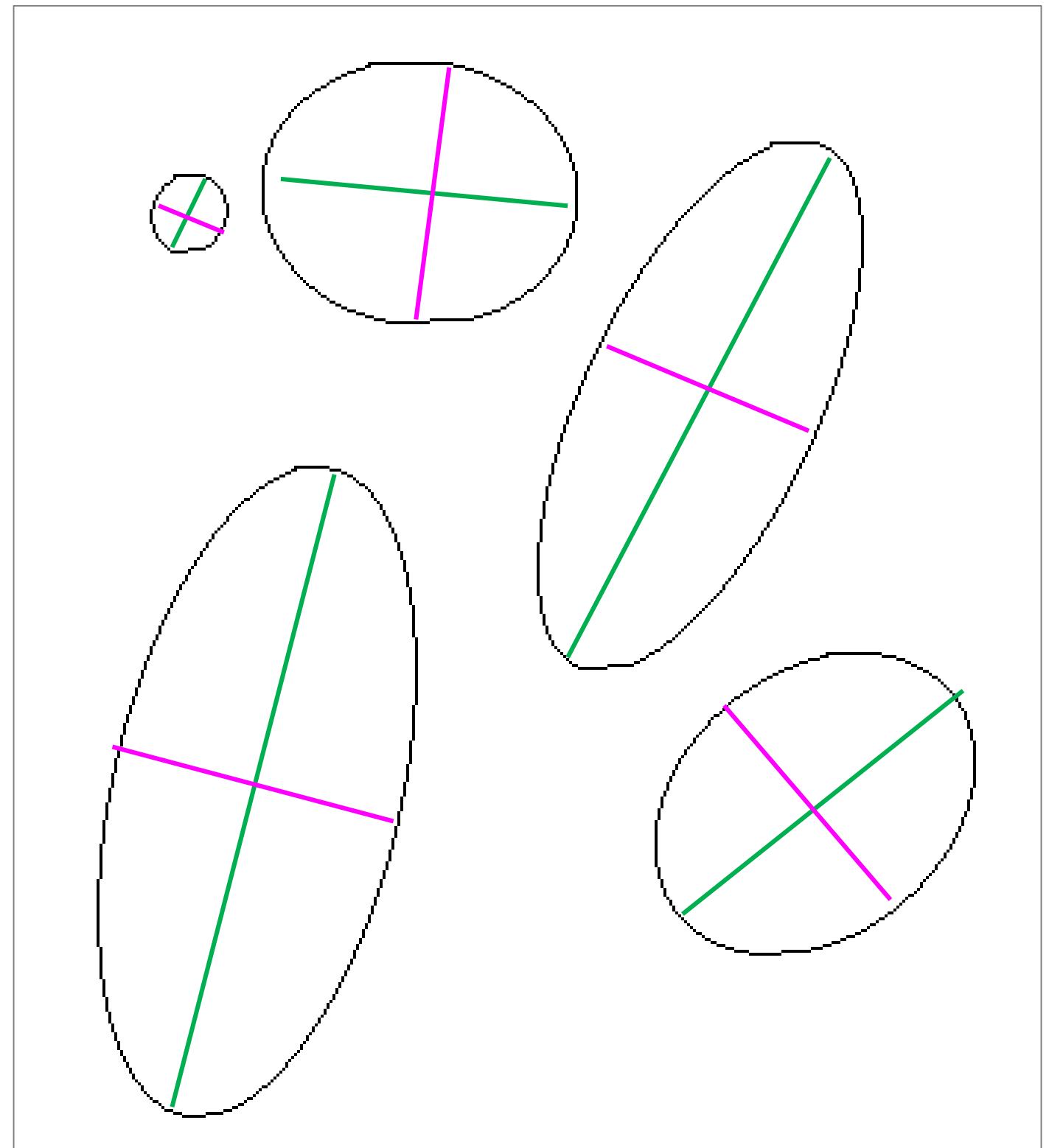
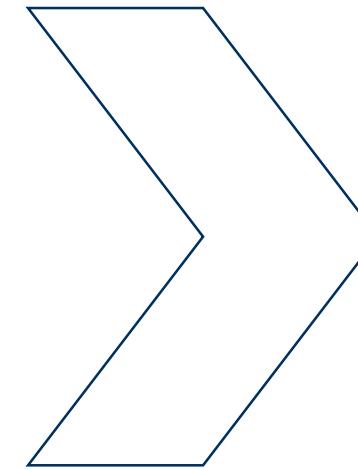
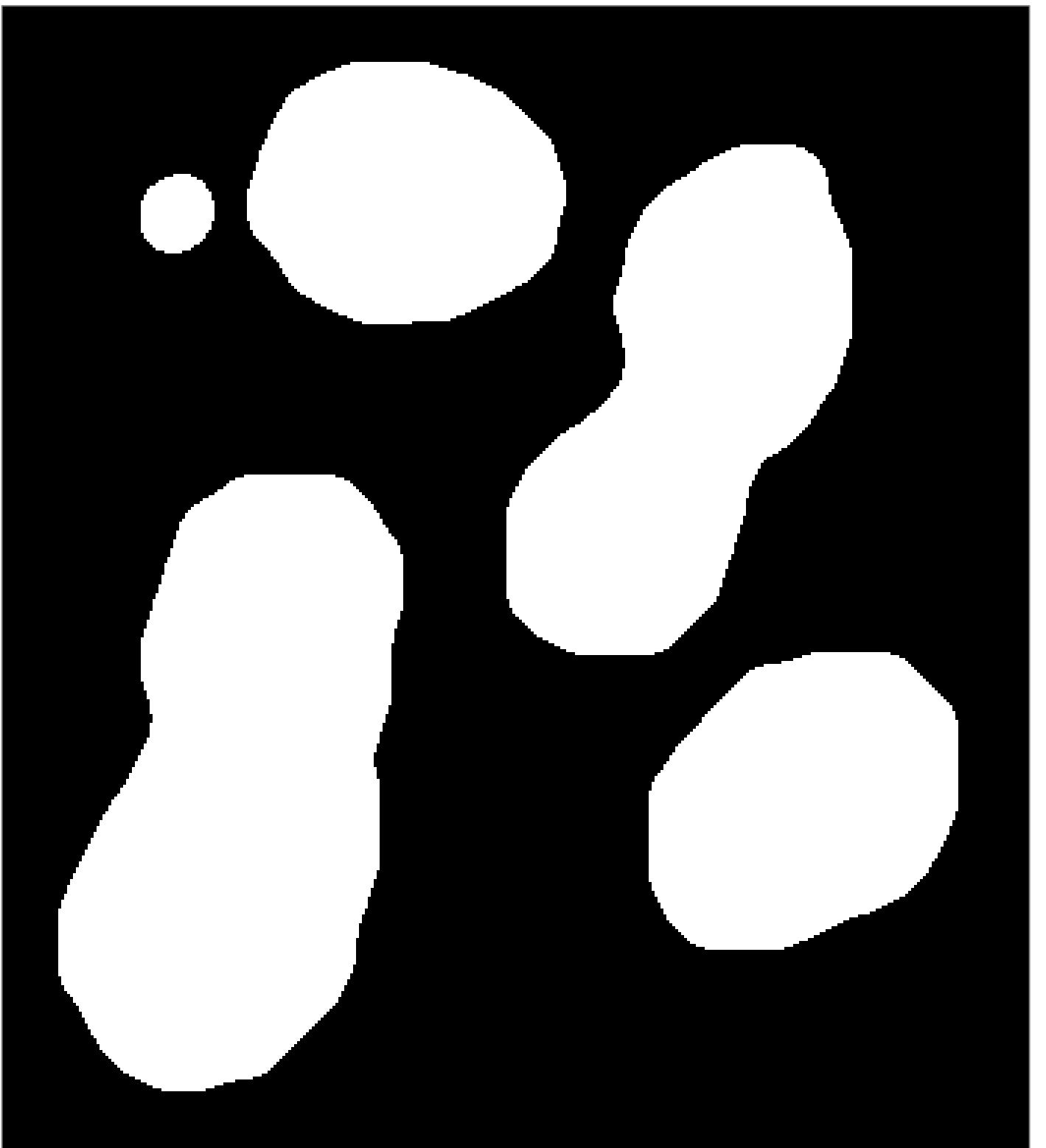
Fit ellipse

For every object, find the optimal ellipse simplifying the object.

Major axis ... long diameter

Minor axis ... short diameter

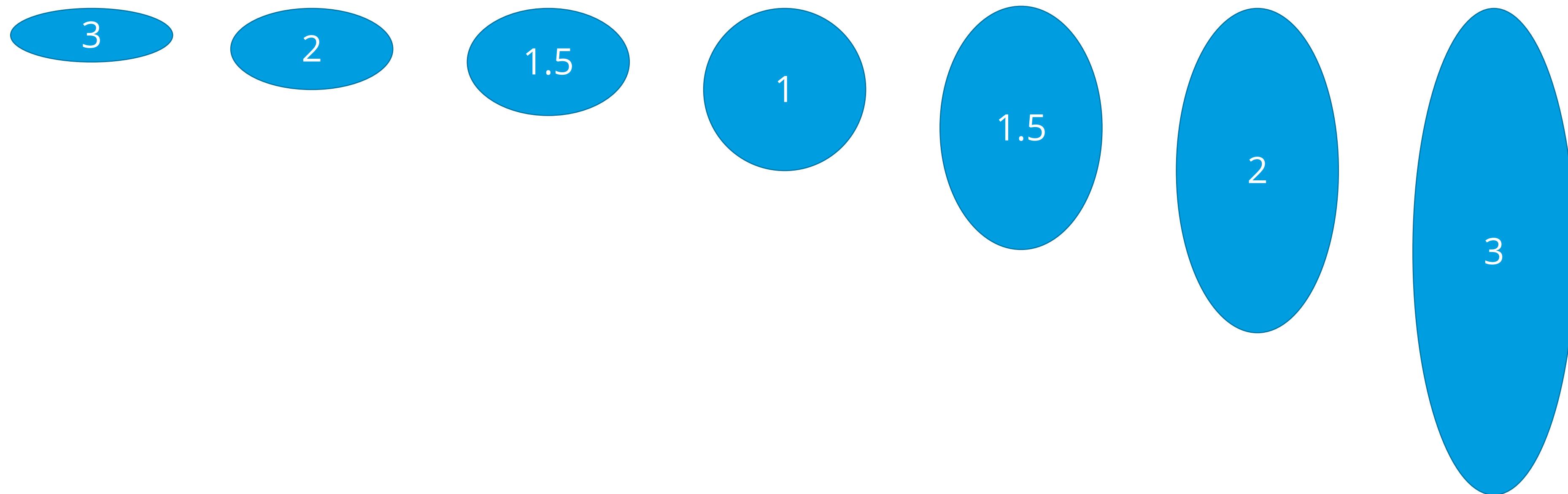
Major and minor axis are
perpendicular to each other



Aspect ratio

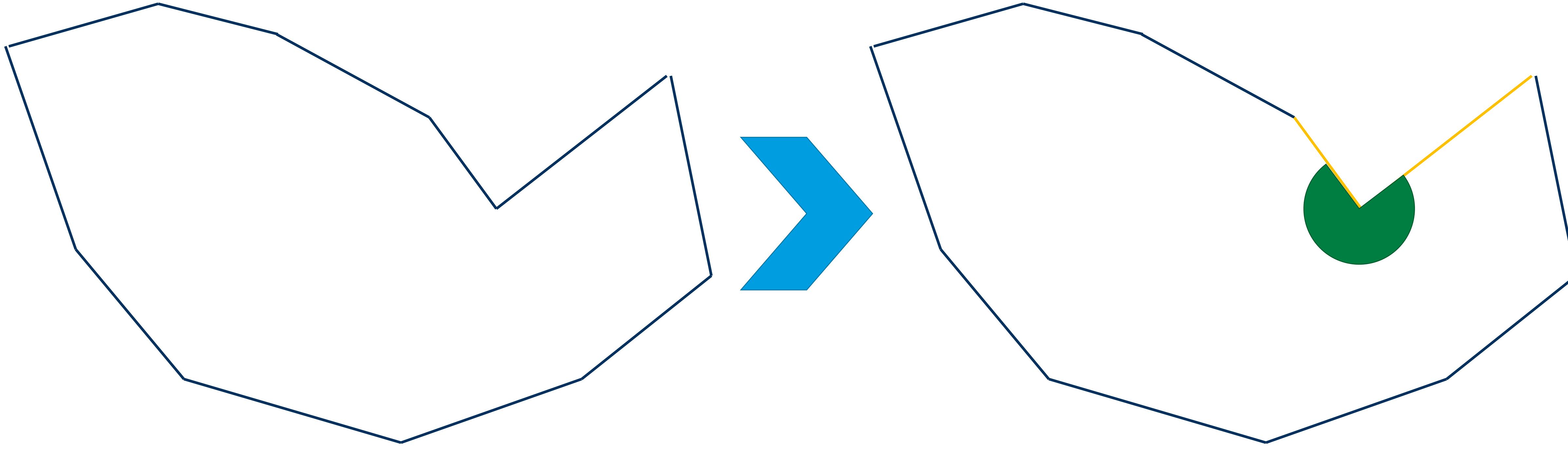
The aspect ratio describes the elongation of an object.

$$AR = \text{major} / \text{minor}$$



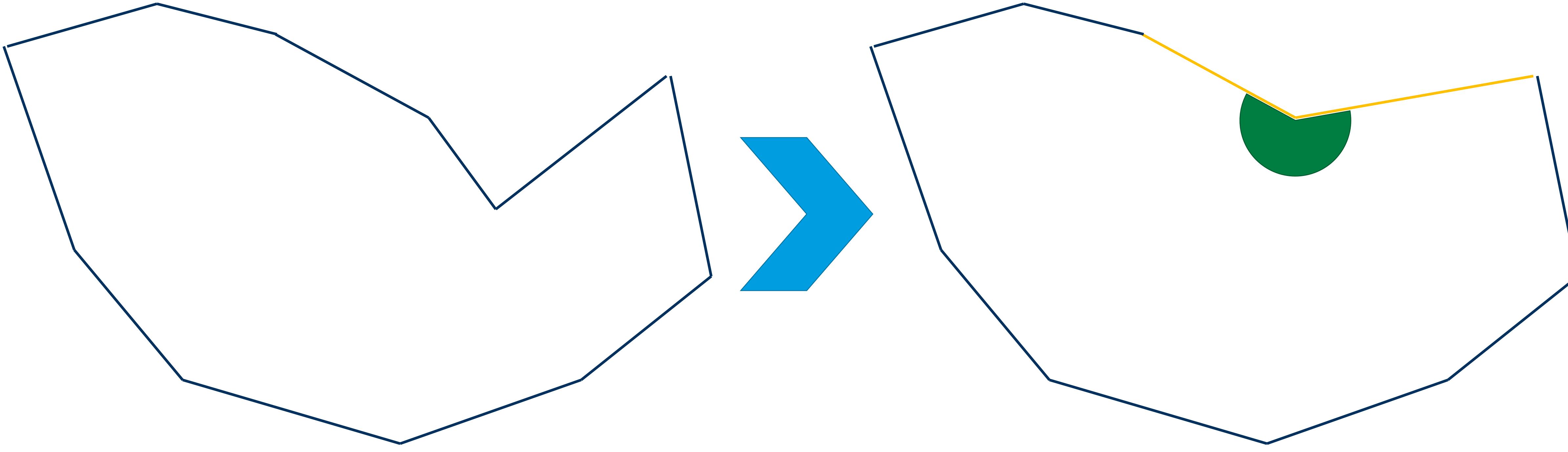
Convex hull

By removing all concave corners of an object, we retrieve its **convex hull**.



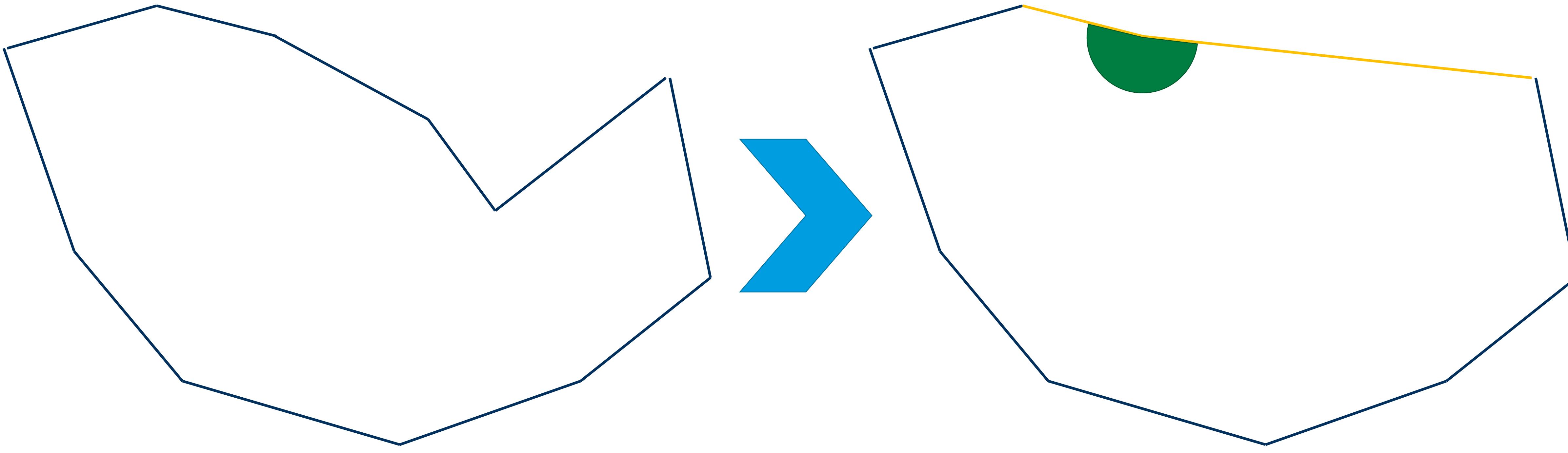
Convex hull

By removing all concave corners of an object, we retrieve its **convex hull**.



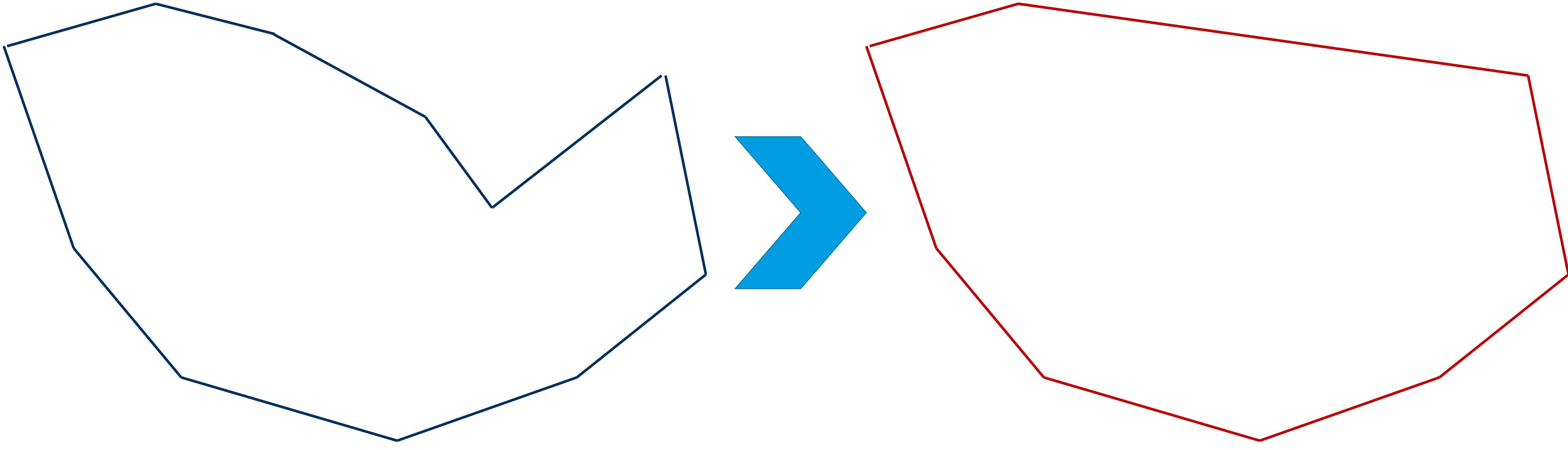
Convex hull

By removing all concave corners of an object, we retrieve its **convex hull**.



Convex hull

By removing all concave corners of an object, we retrieve its **convex hull**.



$$solidity = \frac{A}{A_{convexHull}}$$

Roundness and circularity

The definition of a circle leads us to measurements of circularity and roundness.

In case you use these measures, define them correctly. They are not standardized!

Diameter

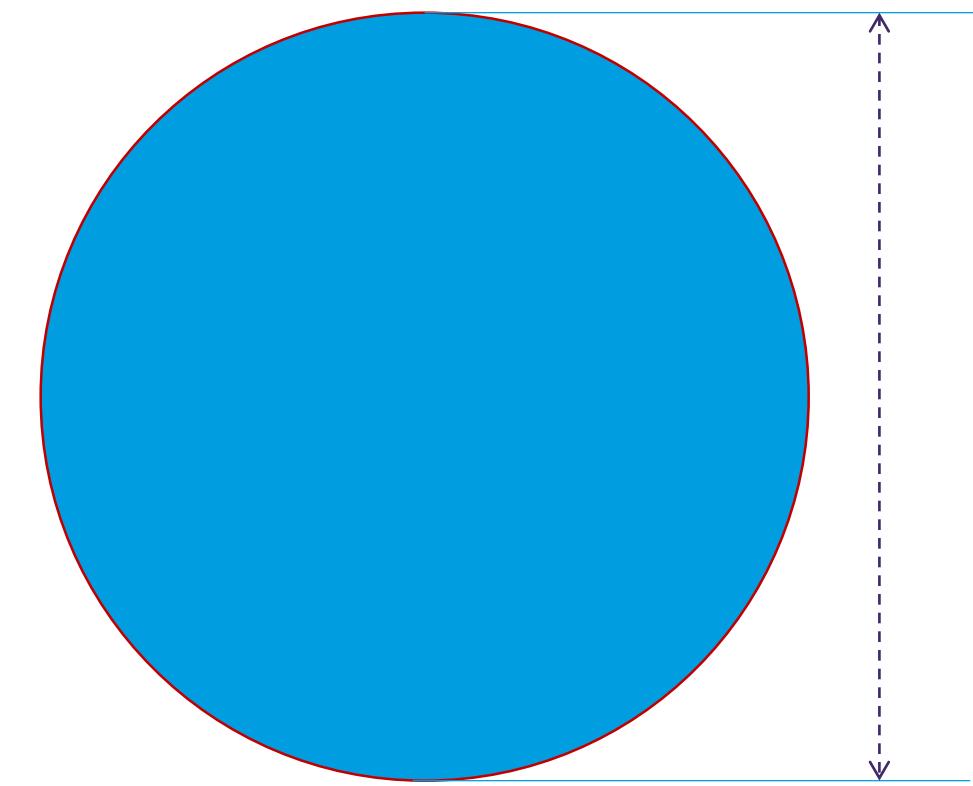
Circumference

Area

d

$$C = \pi d$$

$$A = \frac{\pi d^2}{4}$$



$$\text{roundness} = \frac{4 * A}{\pi \text{ major}^2}$$

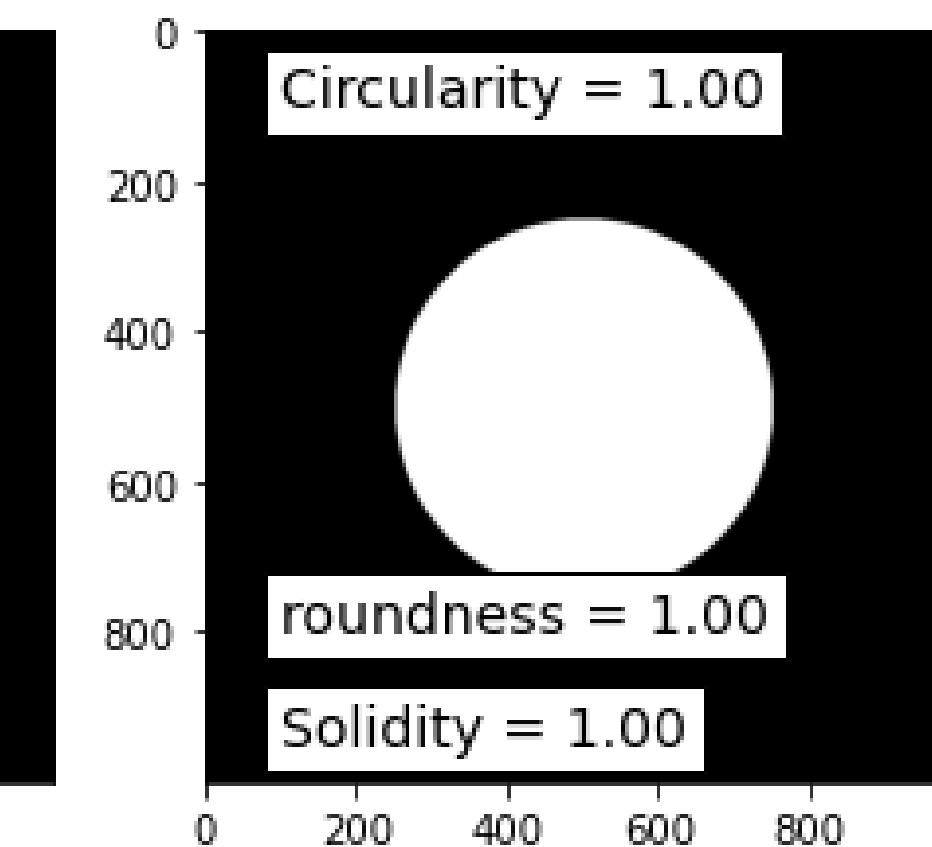
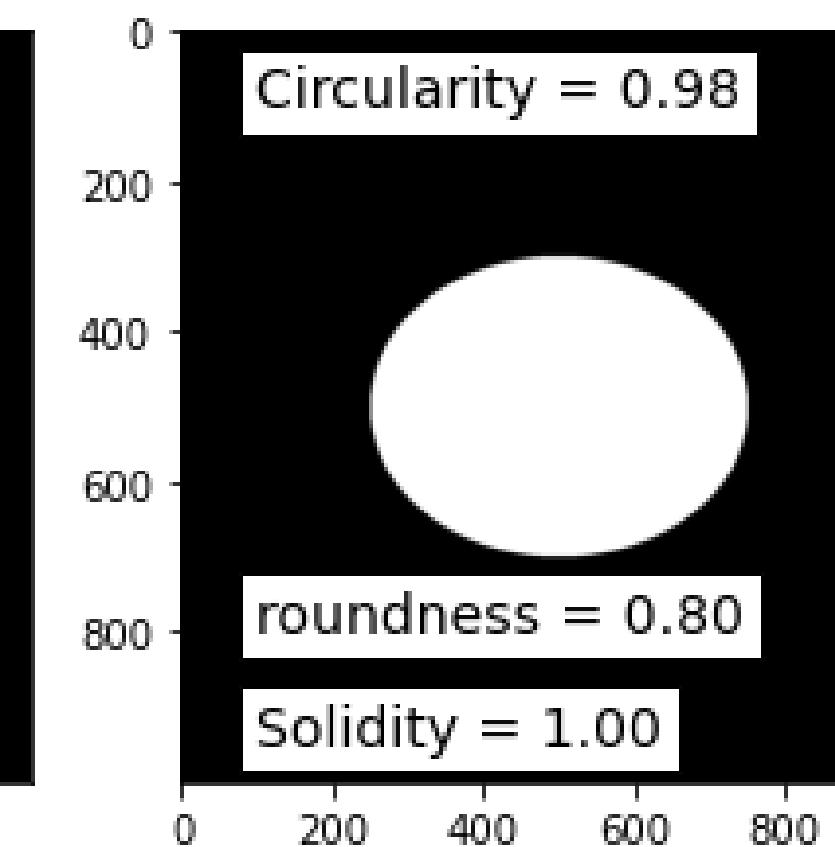
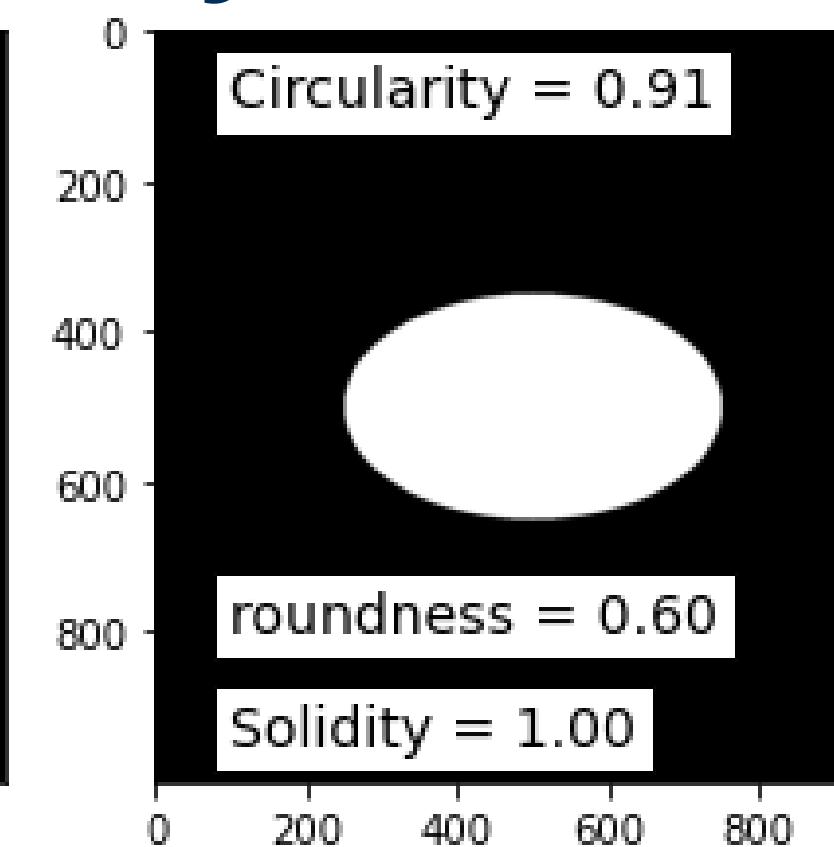
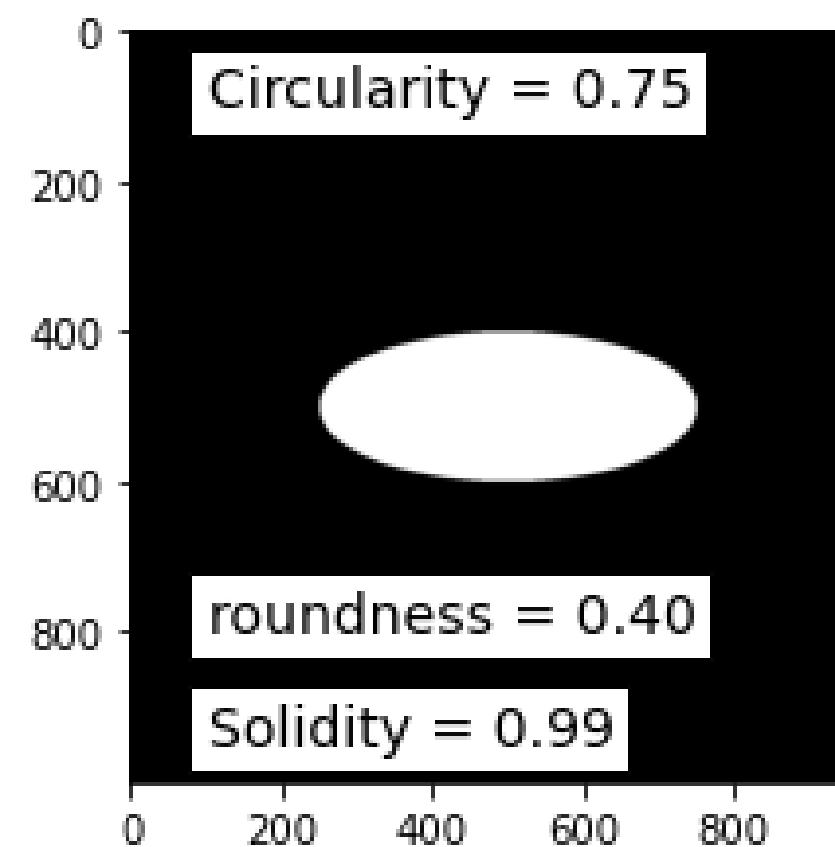
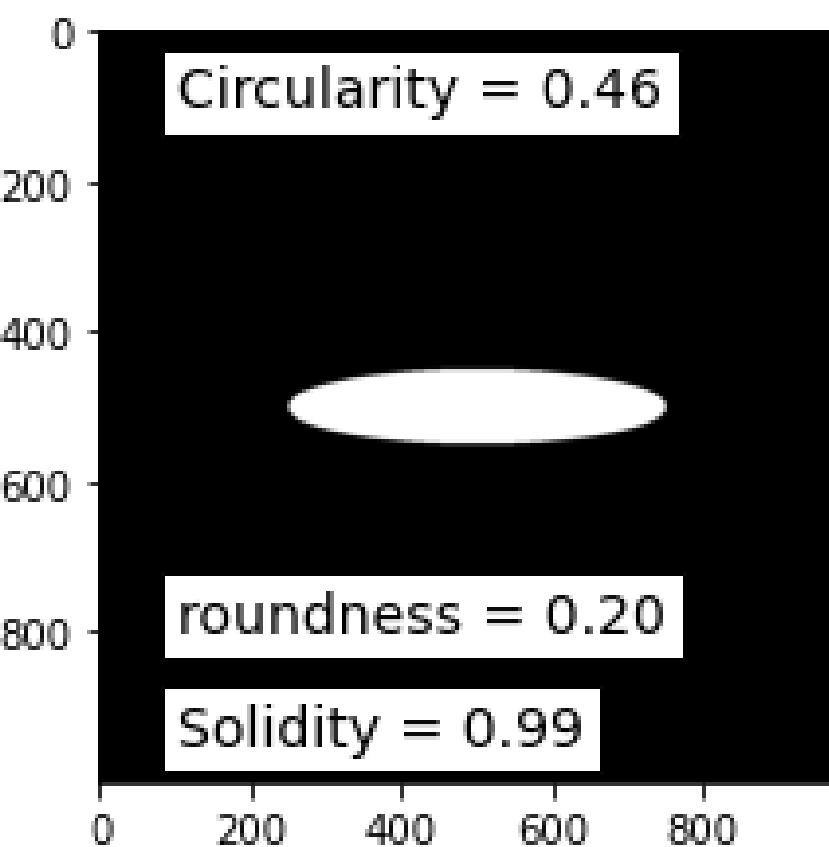
$$\text{circularity} = \frac{4\pi * A}{\text{perimeter}^2}$$

Roundness = 1
Circularity = 1

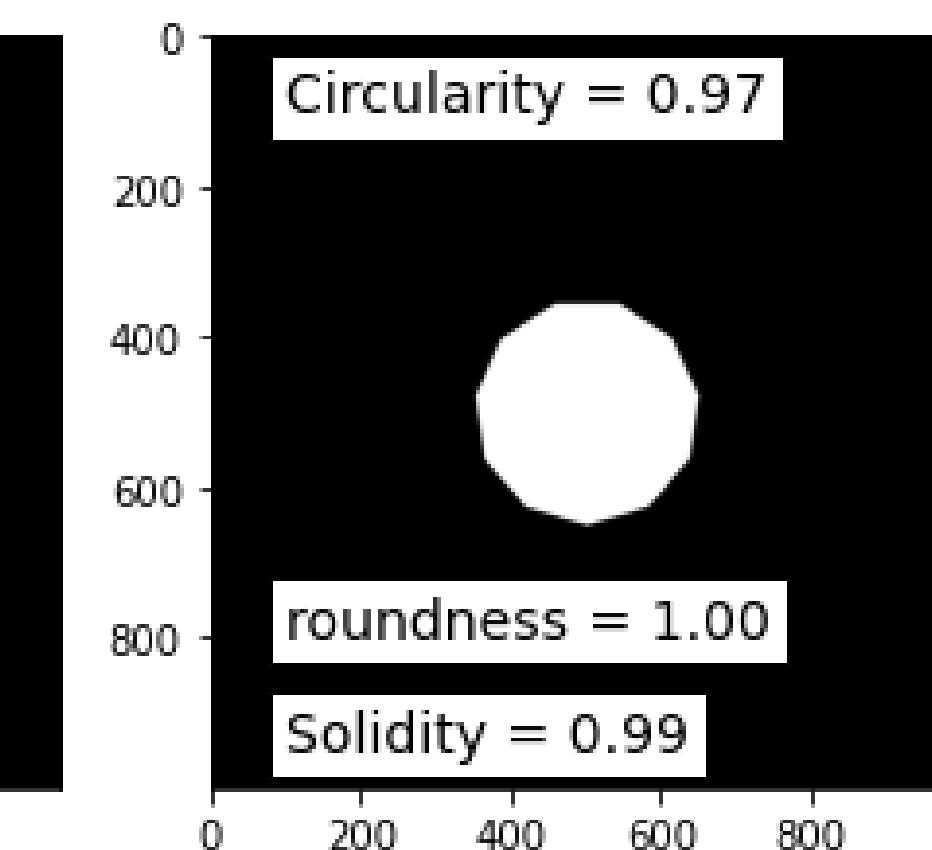
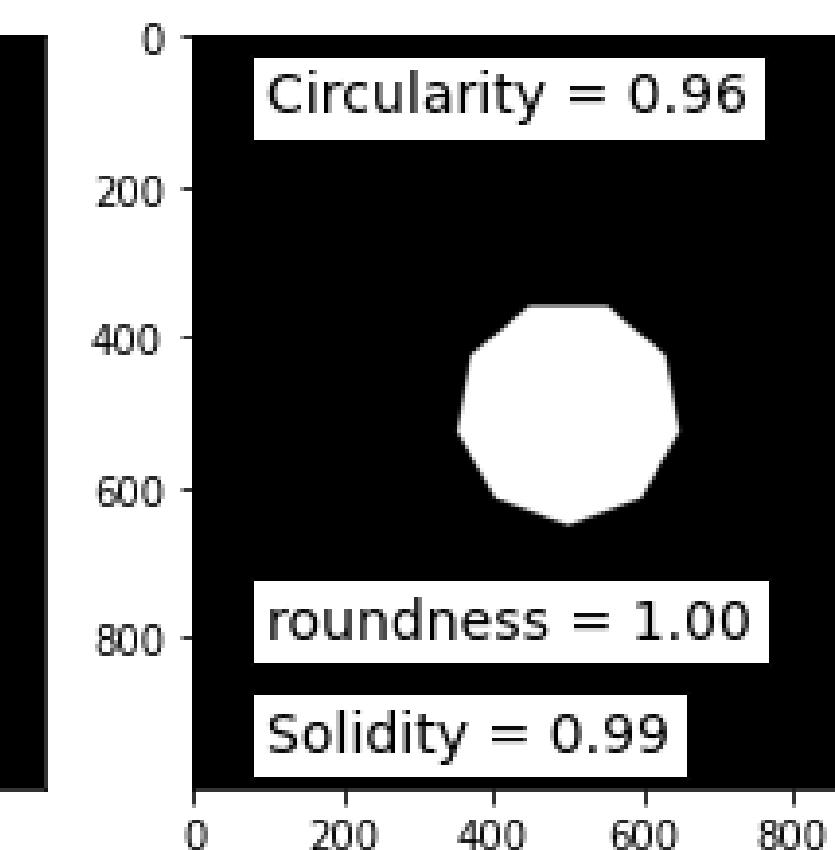
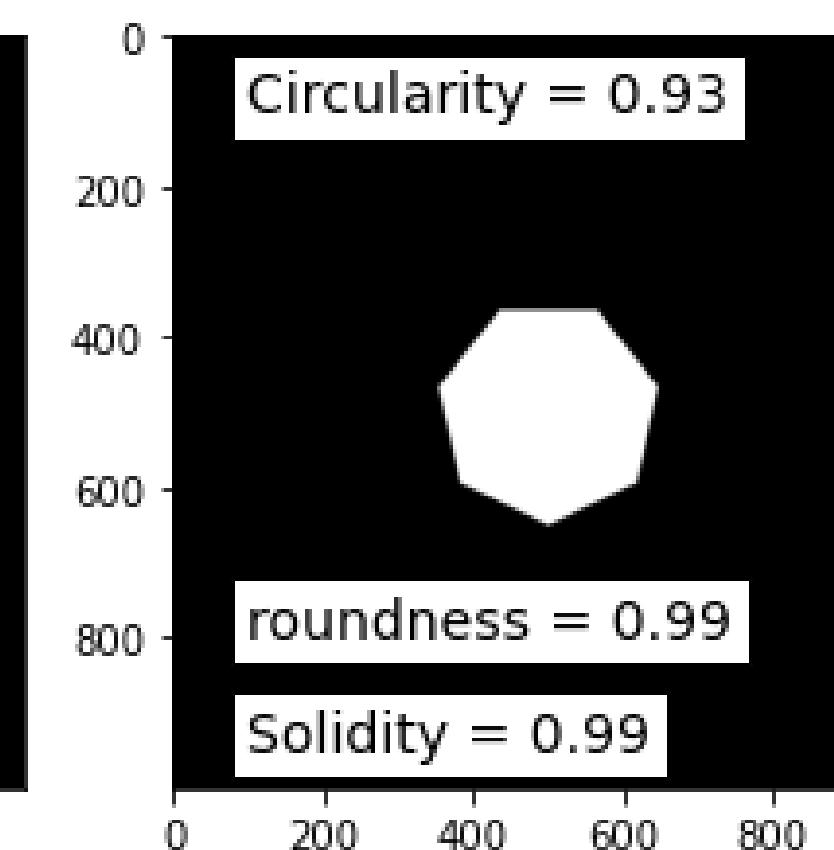
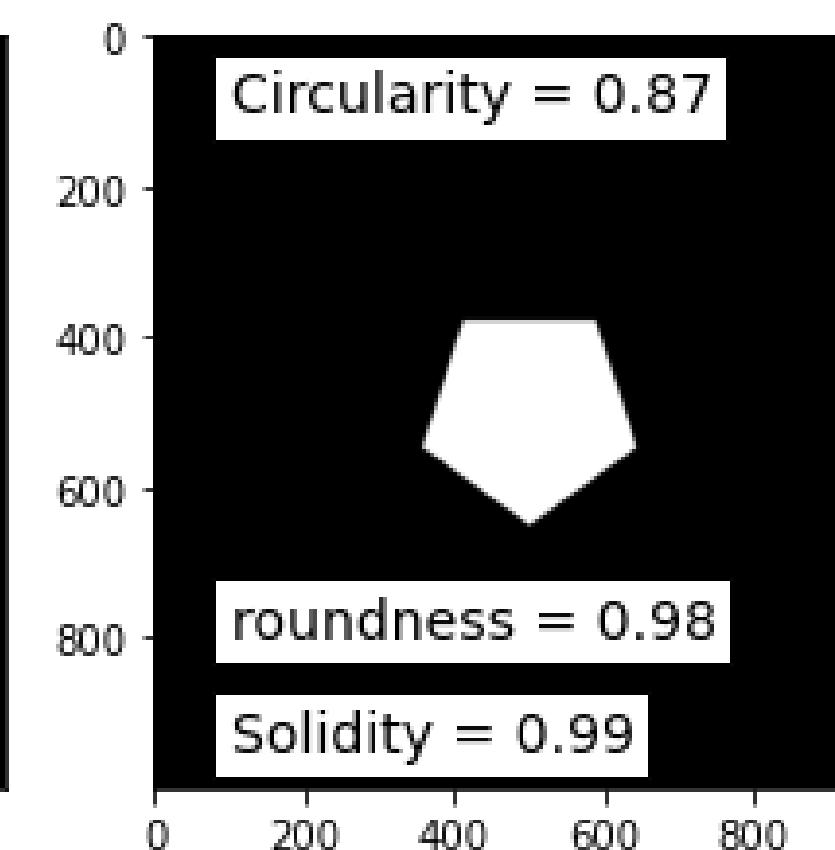
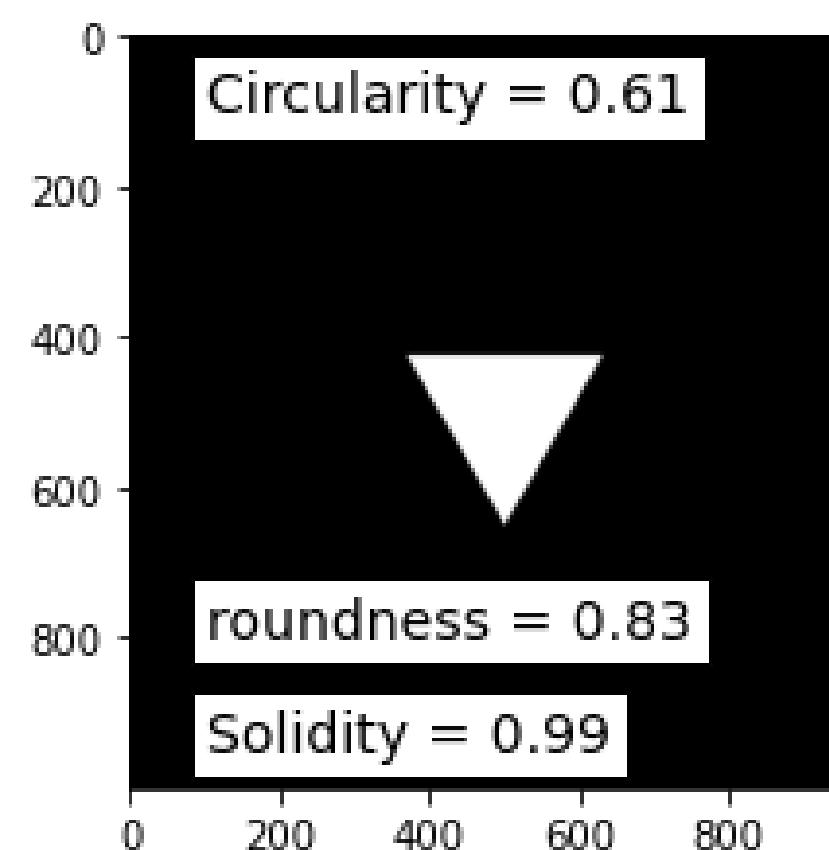
Roundness ≈ 1
Circularity ≈ 1

Roundness < 1
Circularity < 1

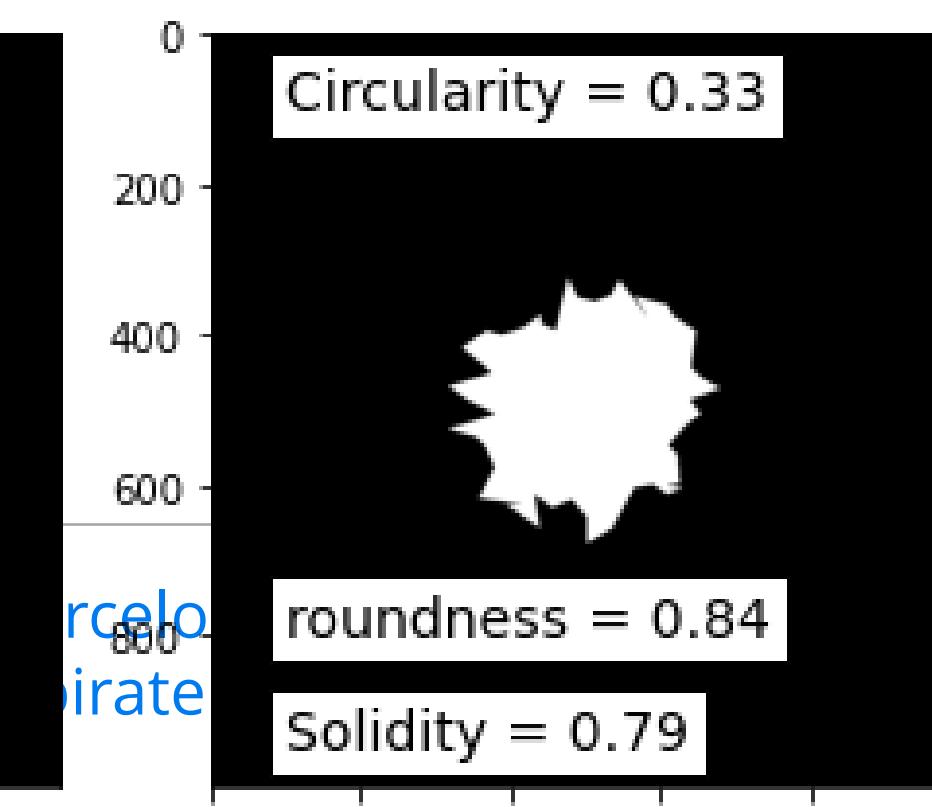
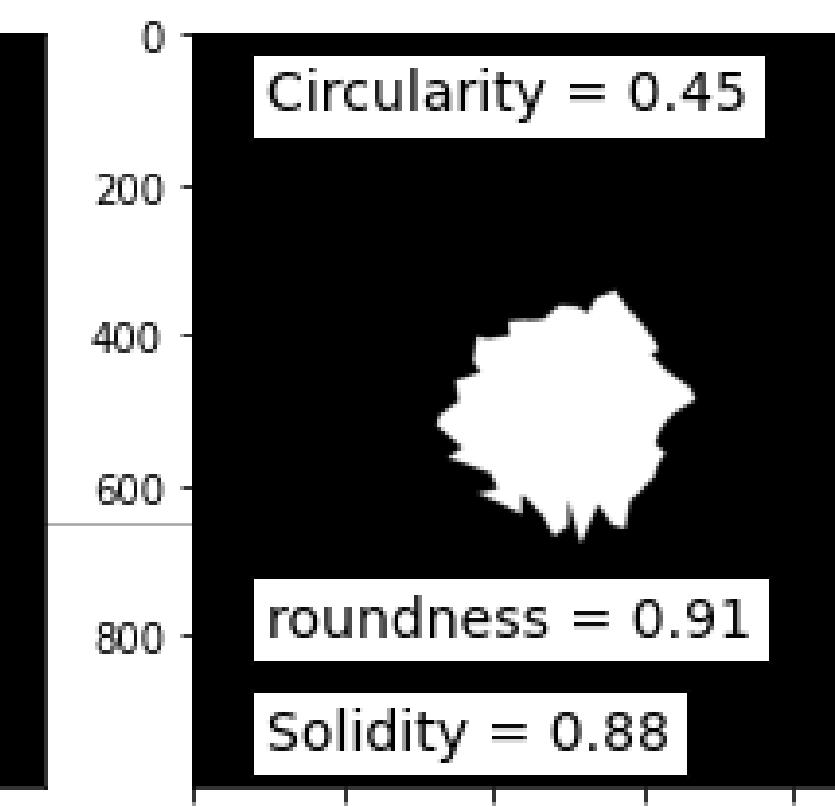
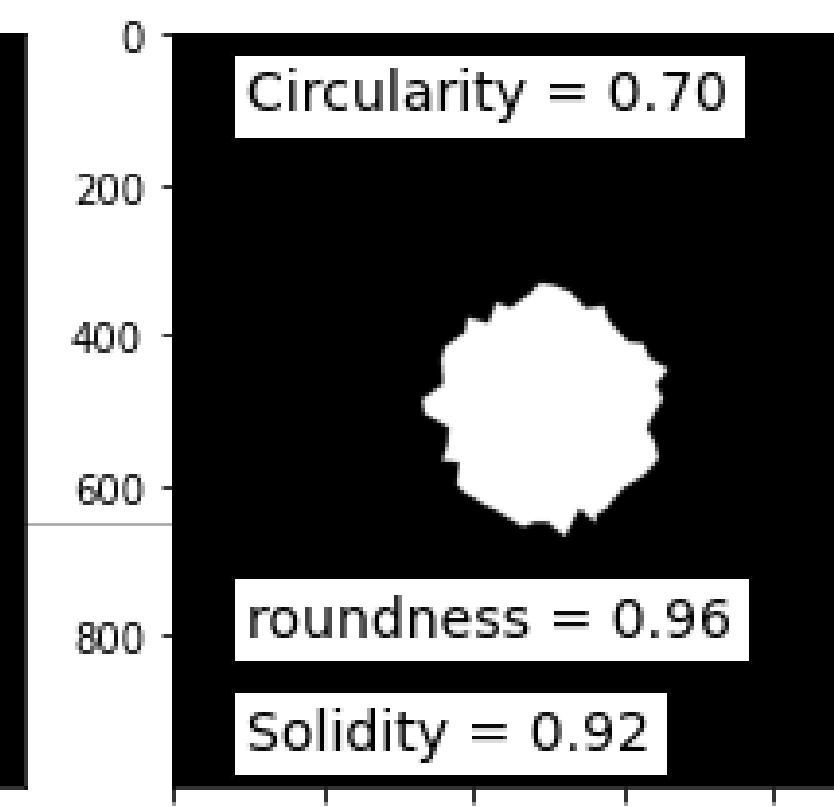
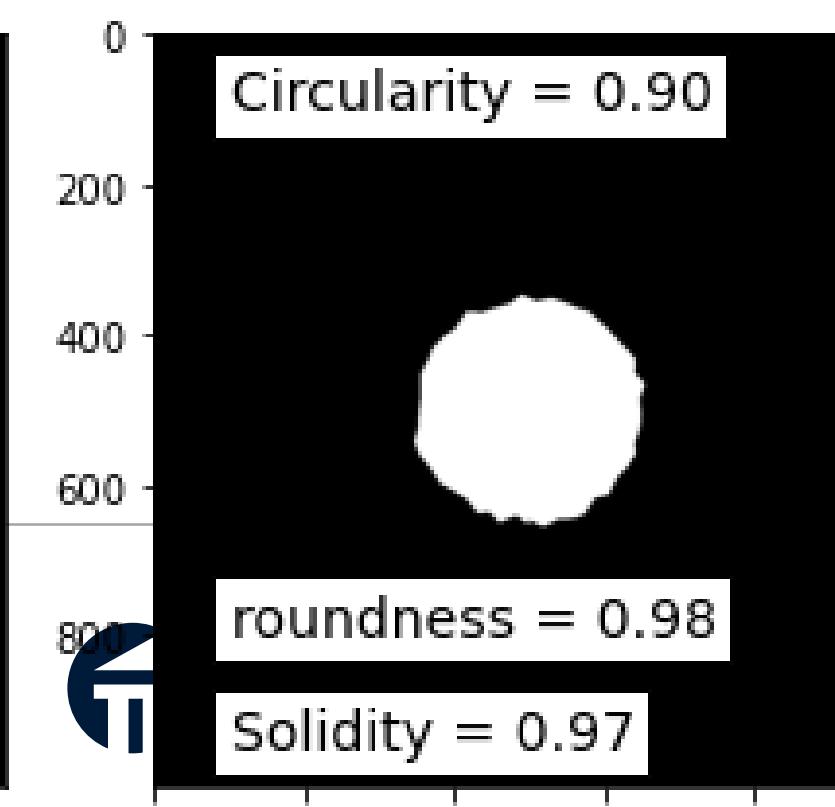
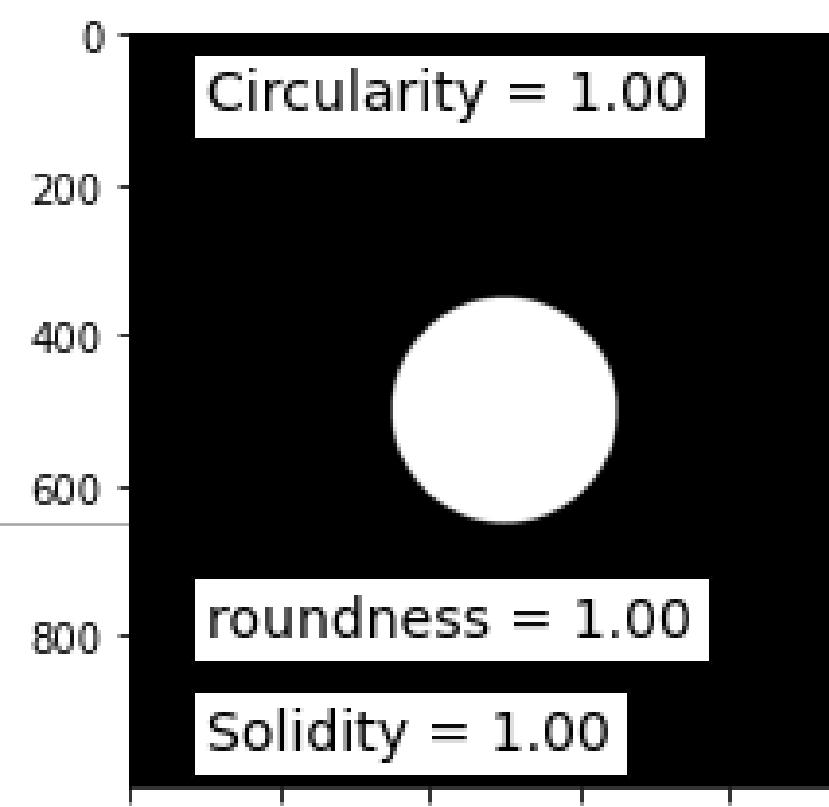
Roundness and circularity



$$roundness = \frac{4 * A}{\pi \text{ major}^2}$$



$$circularity = \frac{4\pi * A}{perimeter^2}$$



$$solidity = \frac{A}{A_{convexHull}}$$

Feature extraction in Python

In Fiji: Analyze > Analyze Particles...

In Python:

```
from skimage import measure
```

<https://scikit-image.org/docs/stable/api/skimage.measure.html>

`skimage.measure.blur_effect (image[, h_size, ...])`

Compute a metric that indicates the strength of blur in an image (0 for no blur, 1 for maximal blur).

`skimage.measure.euler_number (image[, ...])`

Calculate the Euler characteristic in binary image.

`skimage.measure.find_contours (image[, ...])`

Find iso-valued contours in a 2D array for a given level value.

`skimage.measure.grid_points_in_poly (shape, verts)` Test whether points on a specified grid are inside a polygon.

`skimage.measure.inertia_tensor (image[, mu])`

Compute the inertia tensor of the input image.

`skimage.measure.inertia_tensor_eigvals (image)`

Compute the eigenvalues of the inertia tensor of the image.

`skimage.measure.label (label_image[, ...])`

Label connected regions of an integer array.

`skimage.measure.regionprops (label_image[, ...])`

Measure properties of labeled image regions.

`skimage.measure.regionprops_table (label_image)`

Compute image properties and return them as a pandas-compatible table.

area : int

Number of pixels of the region.

area_bbox : int

Number of pixels of bounding box.

area_convex : int

Number of pixels of convex hull image, which is the smallest convex polygon that encloses the region.

area_filled : int

Number of pixels of the region will all the holes filled in. Describes the area of the innermost hole.

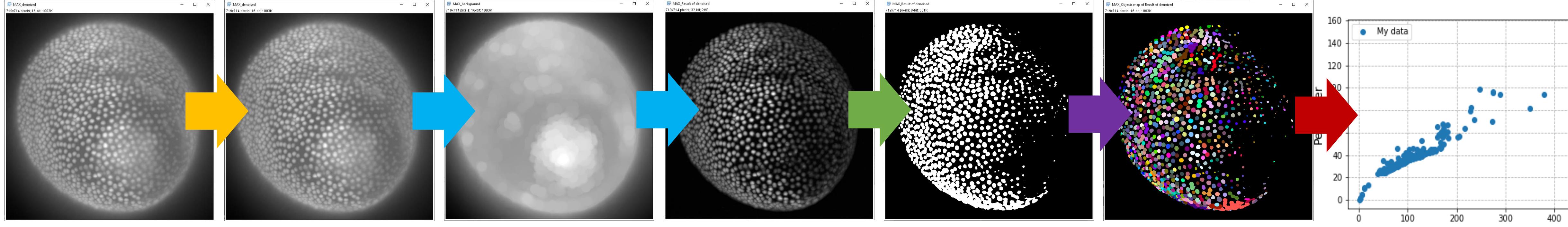
axis_major_length : float

The length of the major axis of the ellipse that has the same normalized second central moment as the region.

axis_minor_length : float

The length of the minor axis of the ellipse that has the same normalized second central moment as the region.

Summary



```
filtered = filters.median(image)
```

```
filtered = filters.gaussian(image, sigma=5)
```

Filtering the image reduces pixel noise

```
bg_subtracted = morphology.white_tophat(image, footprint=footprint)
```

Top-hat filtering removes the background

Thresholding binarizes the image

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

Connected-components analysis groups pixels to objects

```
labels = measure.label(binary)
```

Feature extraction allows descriptive statistics

```
measurements = measure.regionprops_table(labels, properties=properties)
```

Image data source: Daniela Vorkel, Myers lab, MPI CBG