PoL
Physics of Life
TU Dresden

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

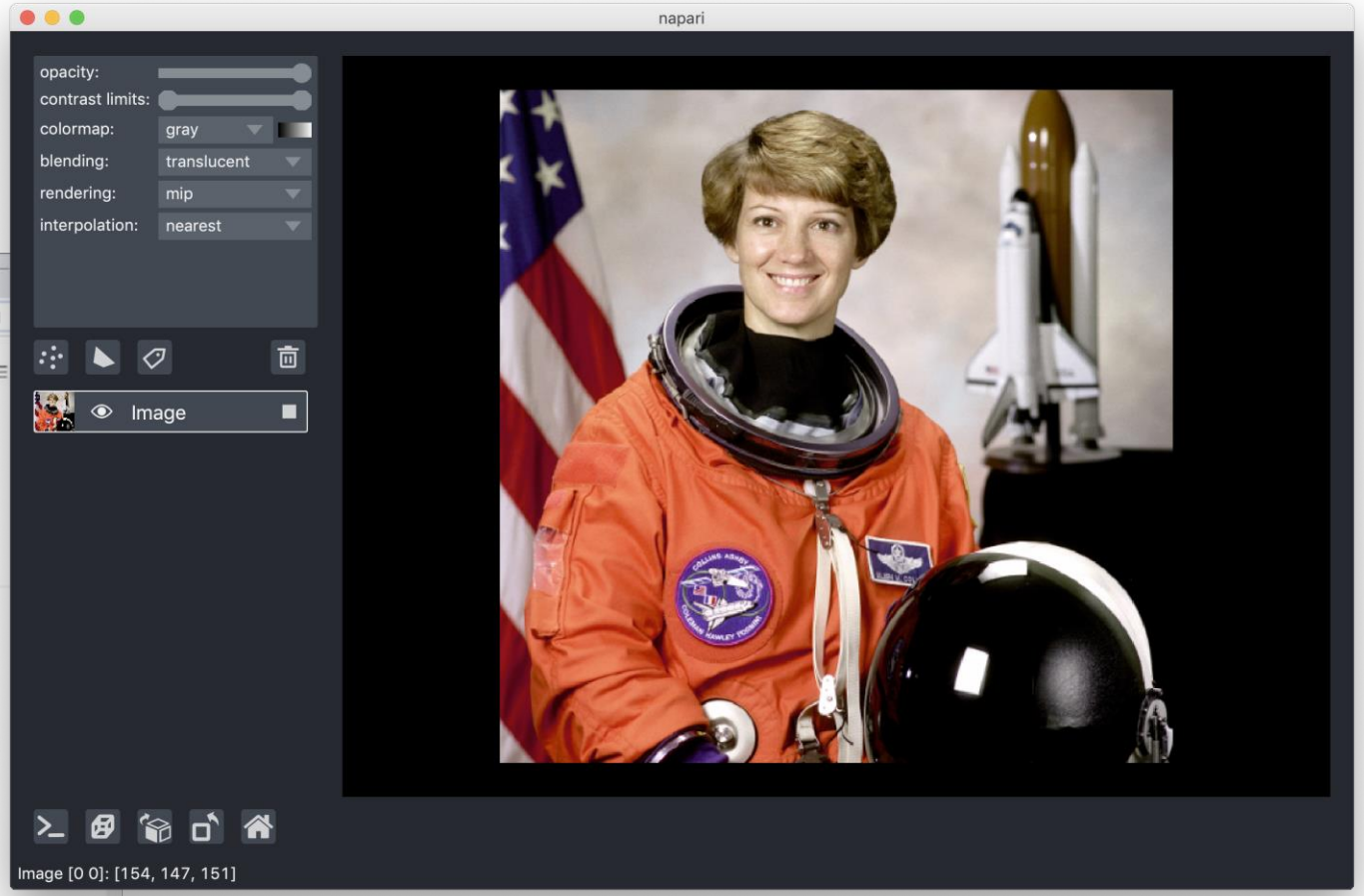
On-the-fly image processing with Python and napari

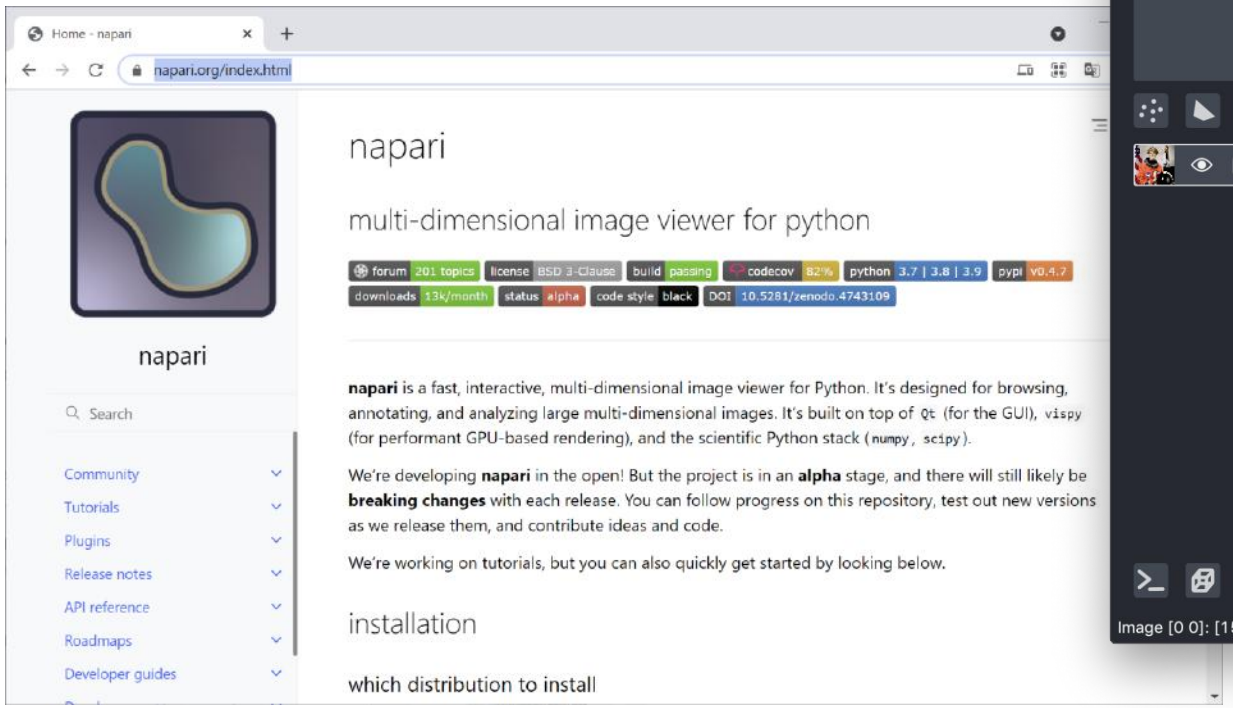
Robert Haase

Partly based on napari community materials https://napari.org/

napari

- Multi-dimensional image viewer in python
- https://napari.org/





napari user interface

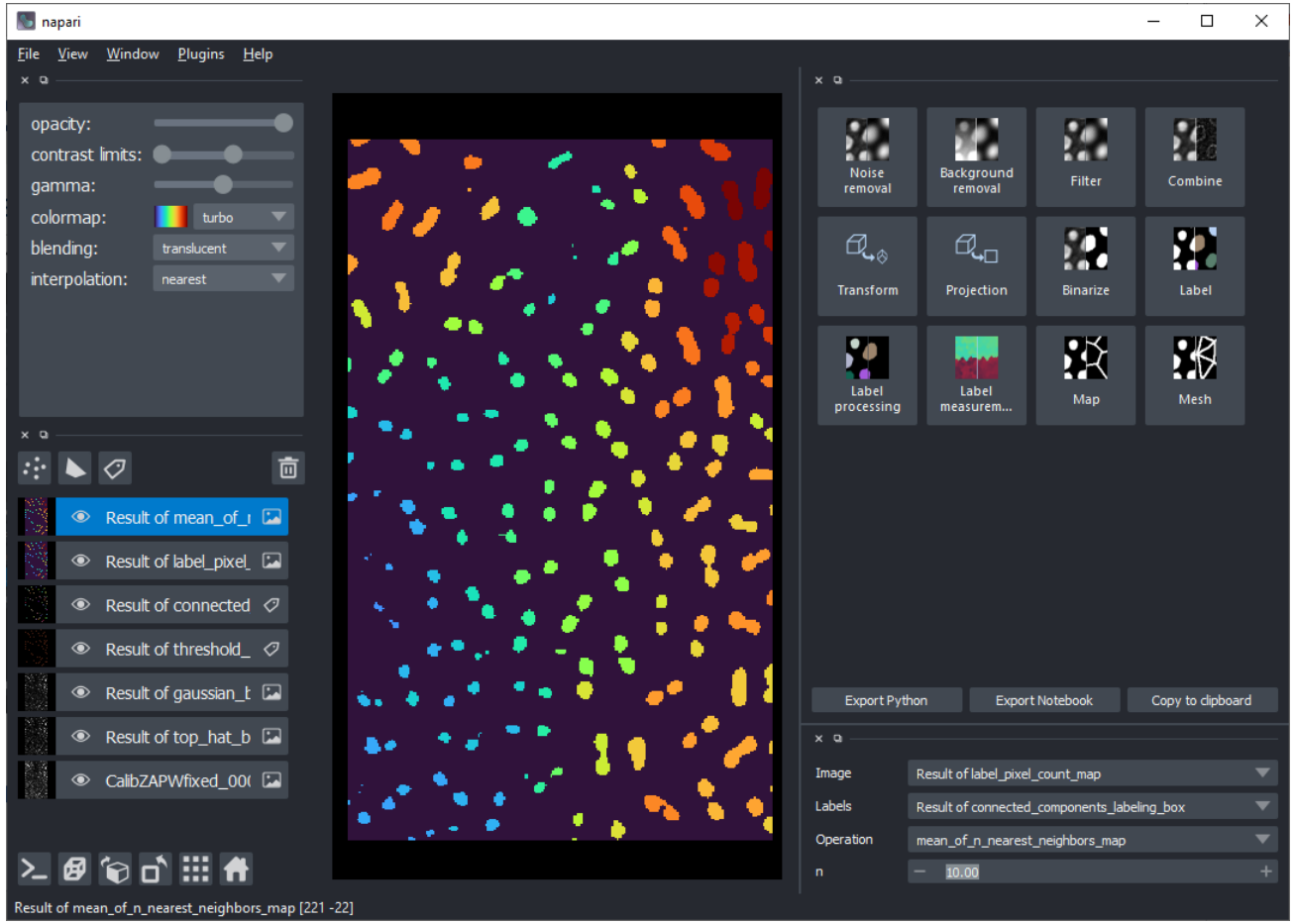
View configuration / tools

```
layer.opacity = 0.5
```

Layers

```
layer.visible = False
```

Viewer controls



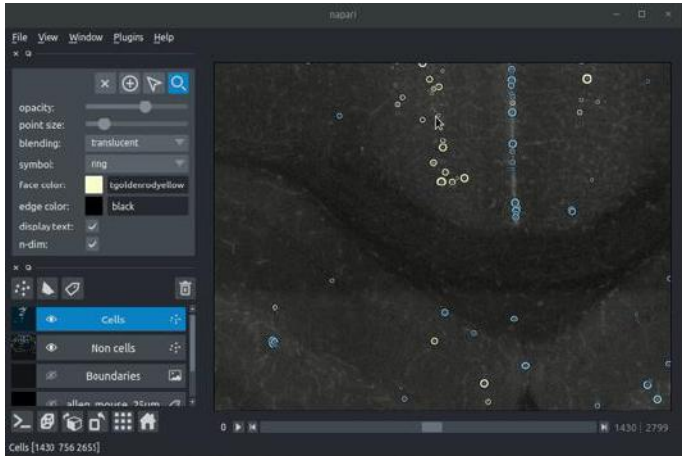
Dock widgets (custom plugins)

Function widgets (custom plugins)

The era of napari plugins has just begun

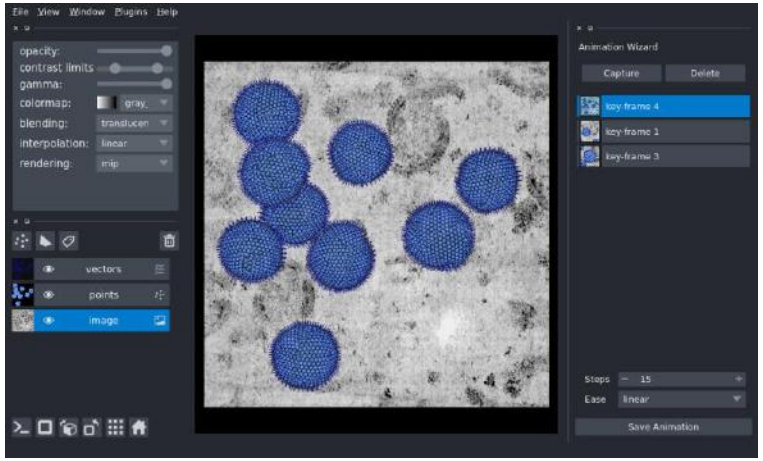
PoL Physics of Life TU Dresden

cellfinder



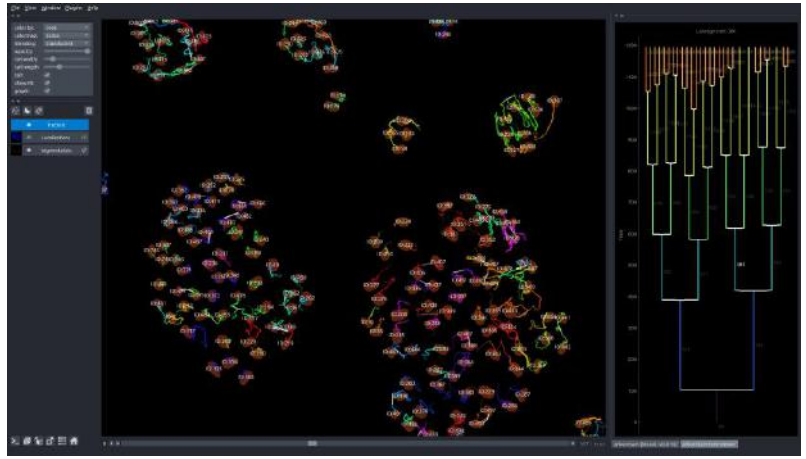
https://github.com/brainglobe/napari-cellfinder

animation



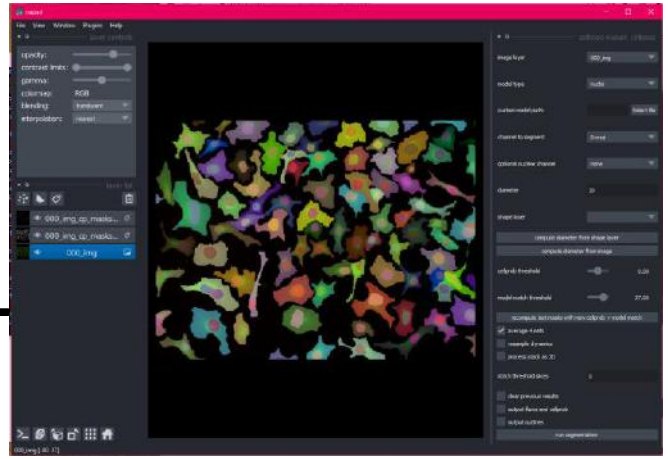
https://github.com/napari/napari-animation

arboretum



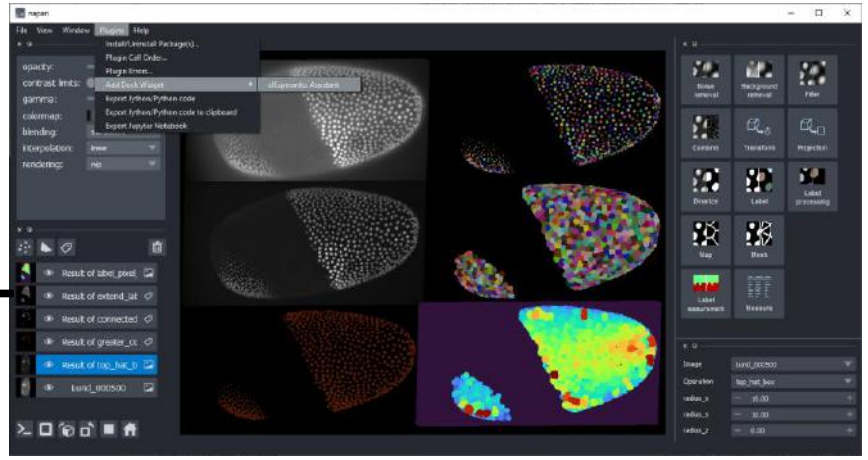
https://github.com/quantumjot/arboretum

cellpose



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

clesperanto



https://github.com/clEsperanto/napari_pyclesperanto_assistant

In development: https://github.com/topics/napari-plugin

Released: https://pypi.org/search/?q=&o=&c=Framework+%3A%3A+napari

Installation

- Install napari via pip

```
pip install napari[all]==0.4.8rc3
```

"all" -> default framework
Alternatively:
"pyqt5" or "pyside2"

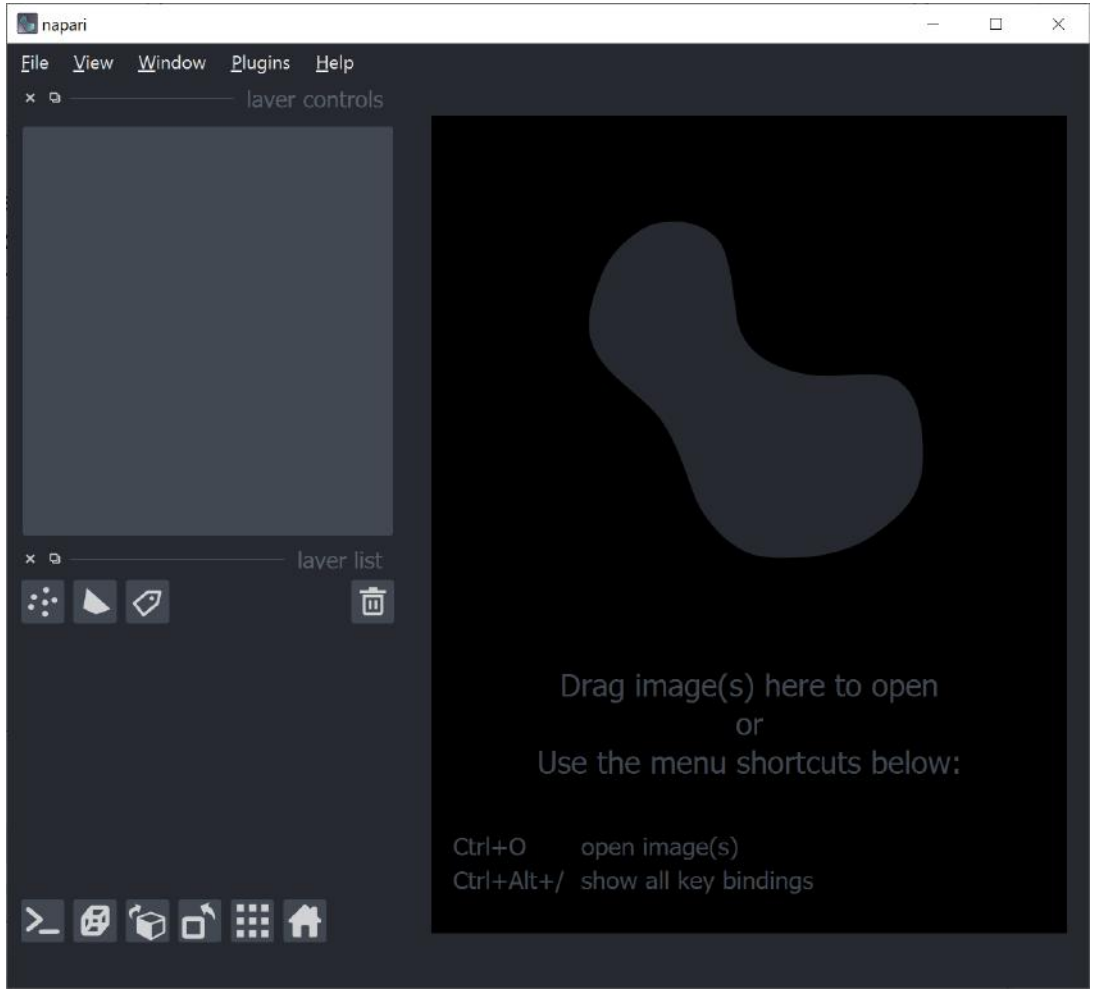
For today only ;-)

- Install napari via conda

```
conda install -c conda-forge napari
```

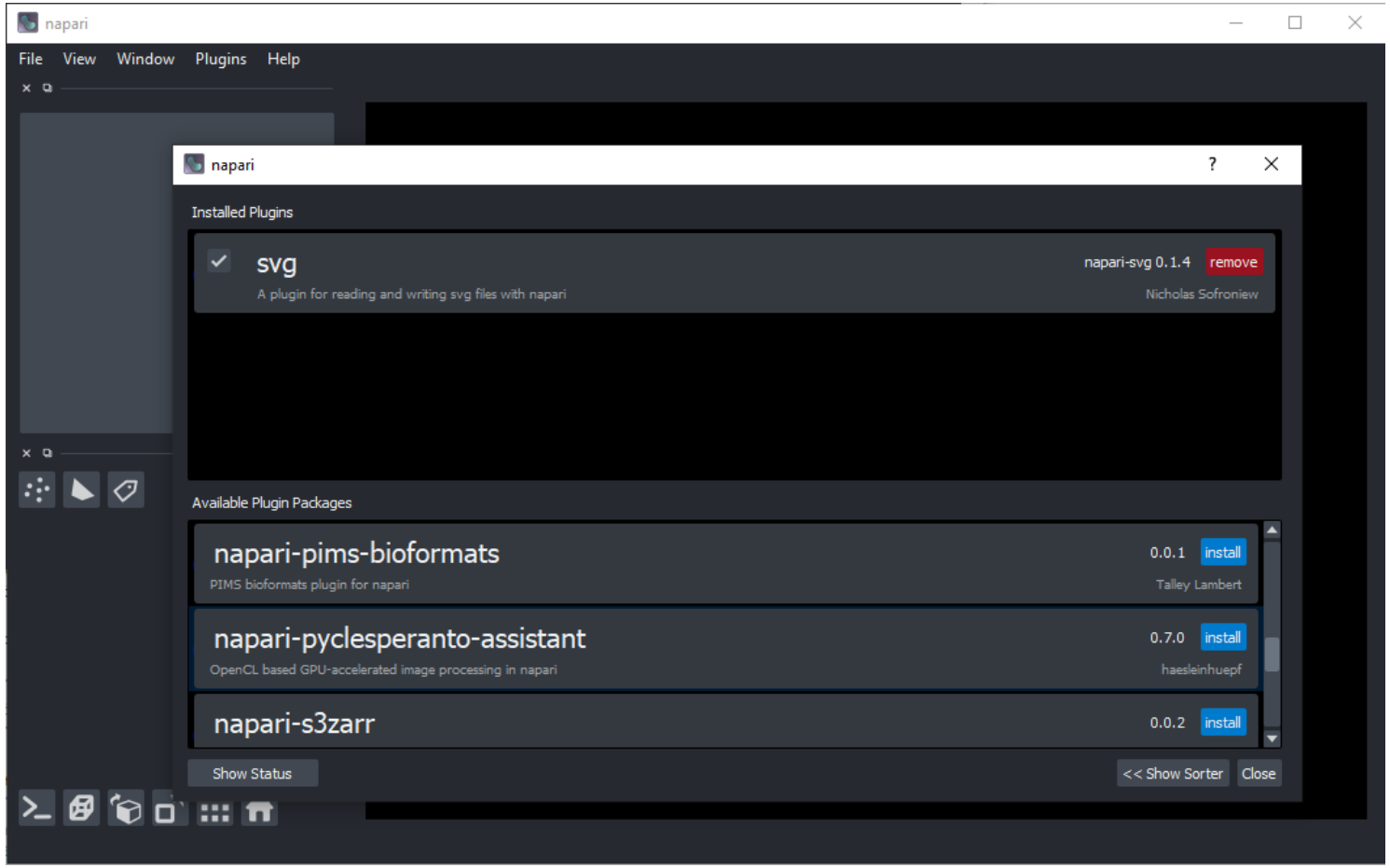
- Test if napari works

```
napari
```



napari plugins

- Install them via conda, pip *or* the menu *Plugins > Install / Uninstall packages…*



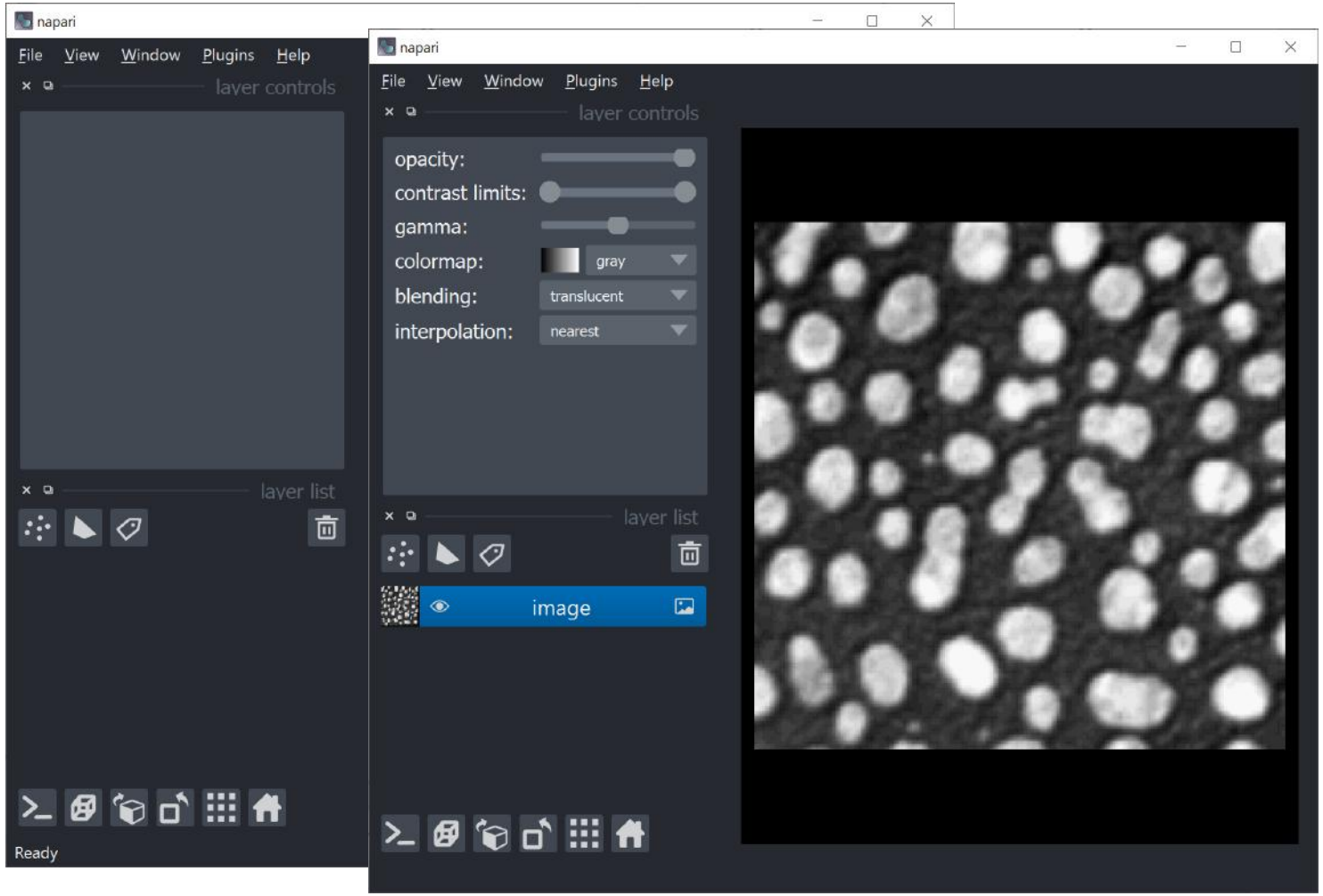
Scripting napari

- Initialization

```
import napari

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

# Start it
napari.run()
```

- Adding images

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



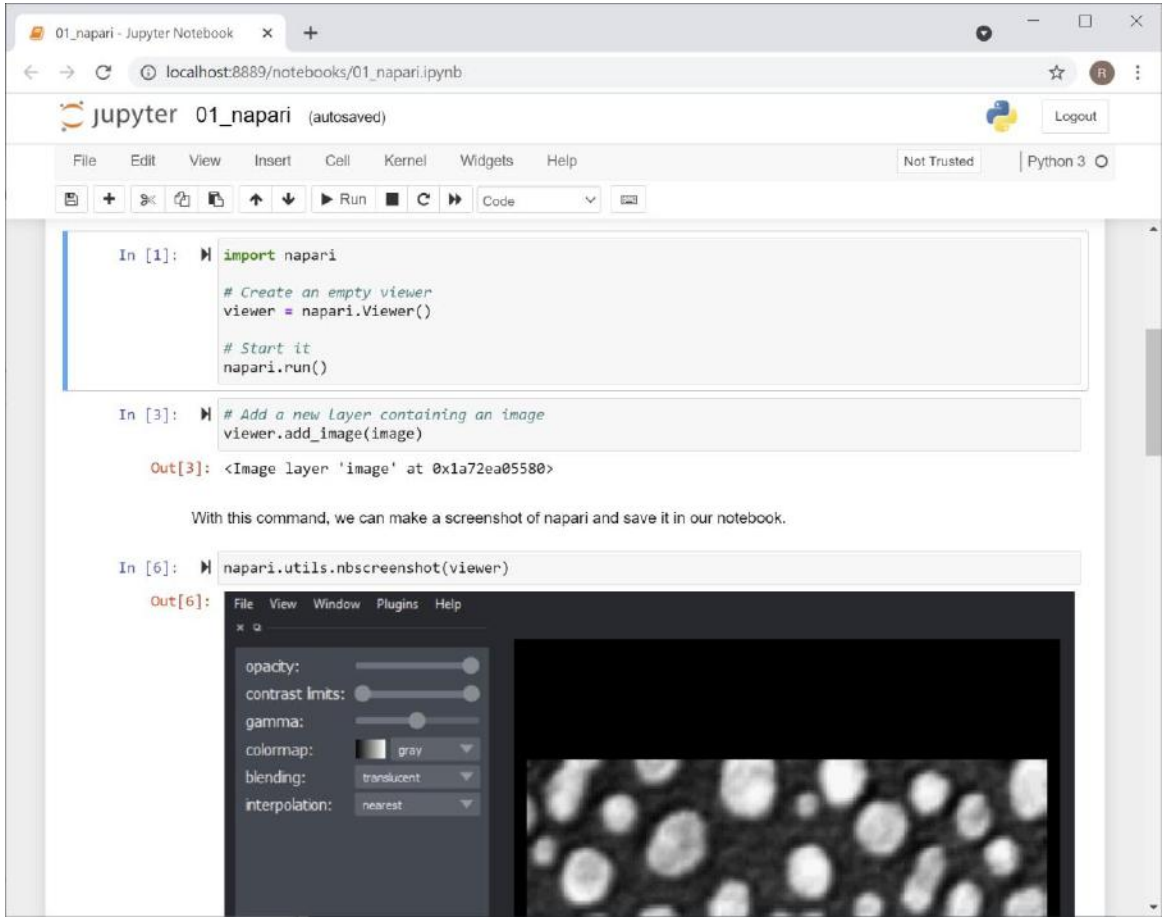
Scripting napari in notebooks

PoL Physics of Life TU Dresden

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

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

Place your viewer here



Working with layers

- Removing layers

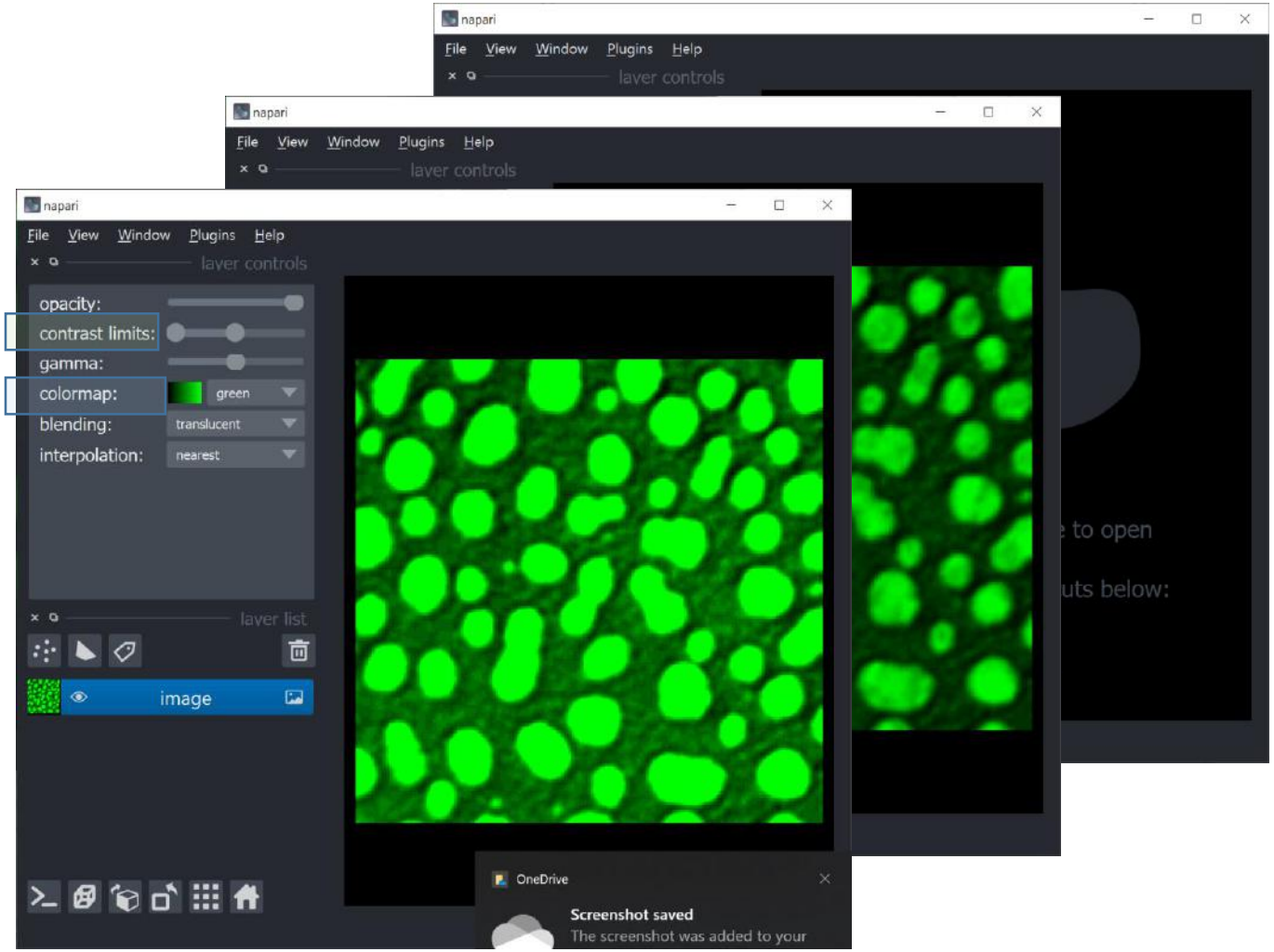
```
for l in viewer.layers:
    viewer.layers.remove(l)
```

- Modify visualization while adding layers

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

- Modify layers after adding

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



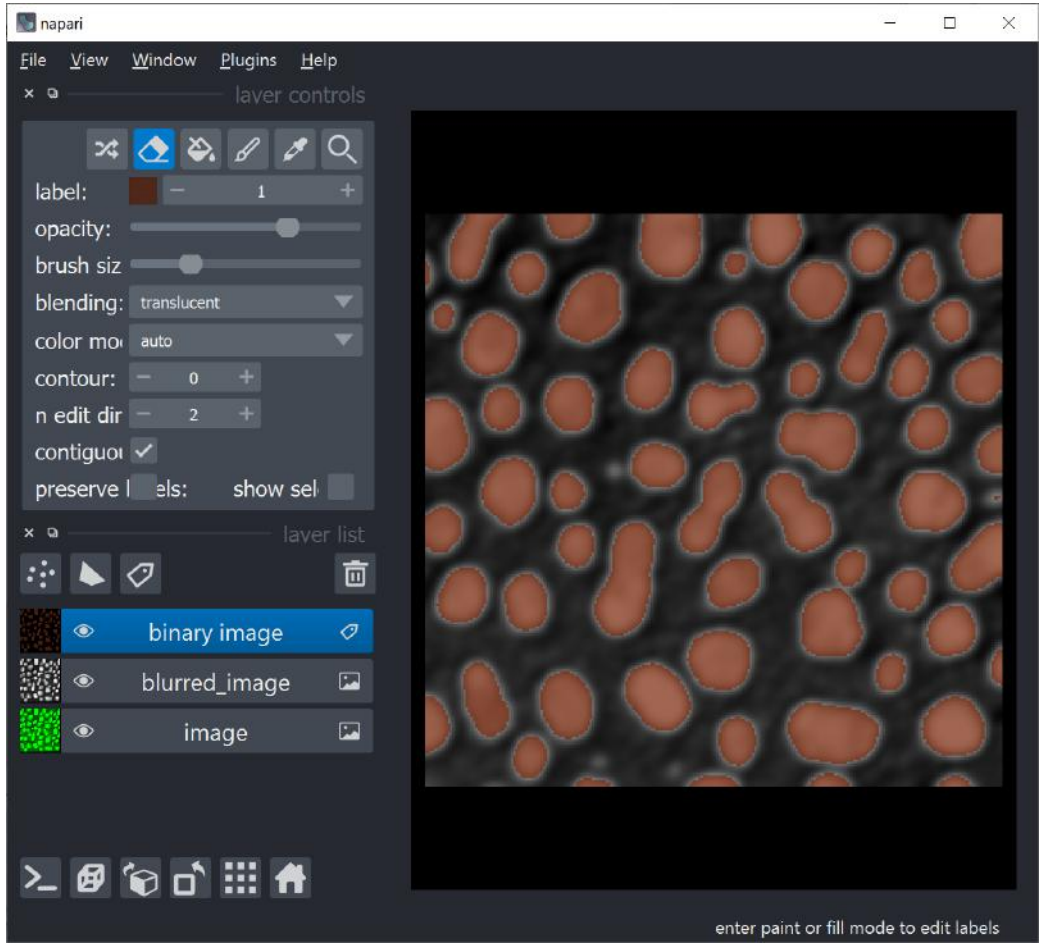
Visualizing image segmentation

PoL Physics of Life TU Dresden

- 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



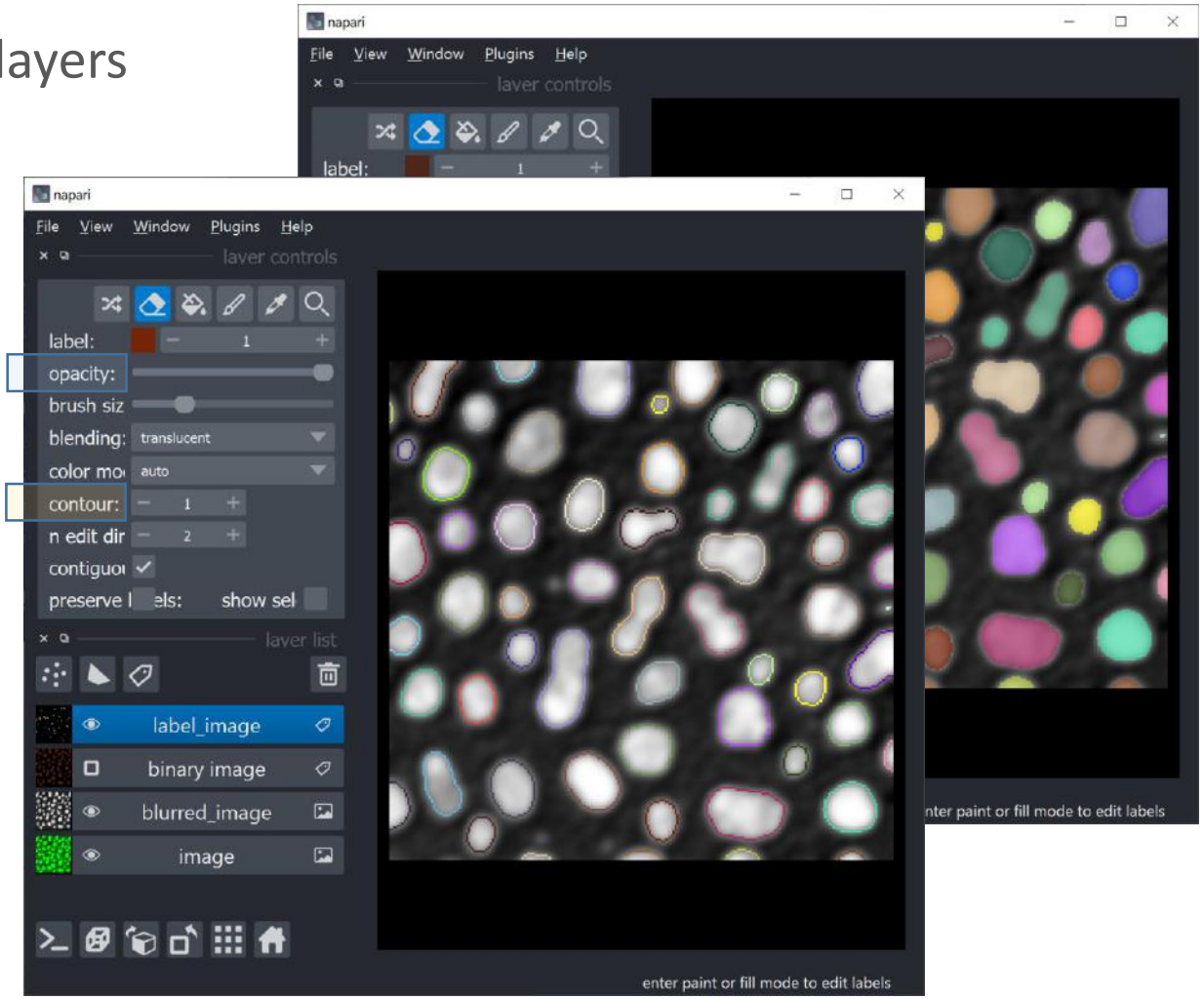
http://nbviewer.jupyter.org/github/BiAPoL/on_the_fly_image_processing_napari/blob/master/01_napari.ipynb

Visualizing image segmentation

PoL
Physics of Life
TU Dresden

- 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_image)
```



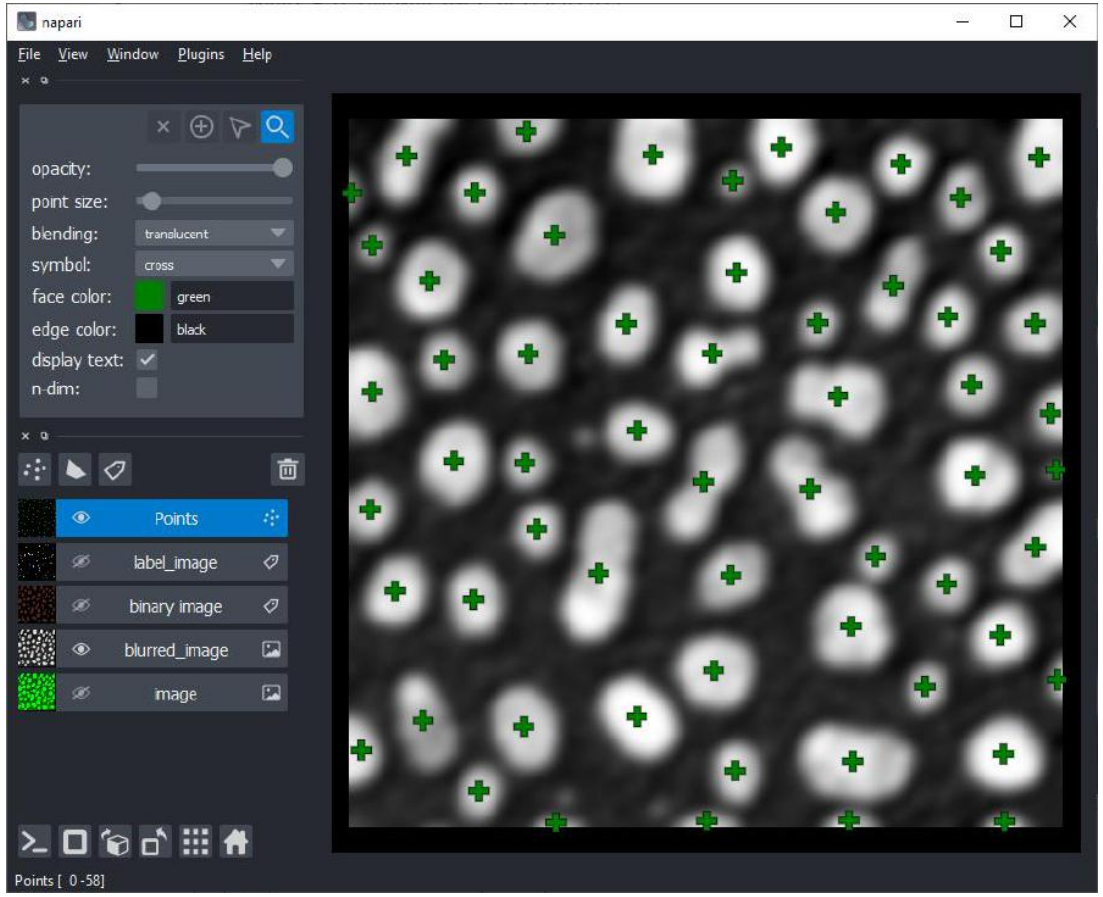
- Visualize contours instead of the overlay

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


http://nbviewer.jupyter.org/github/BiAPoL/on_the_fly_image_processing_napari/blob/master/01_napari.ipynb

Points layers

- There is also other layer types
  - Shapes
  - Points
  - Surfaces
  - Tracks
  - Vectors

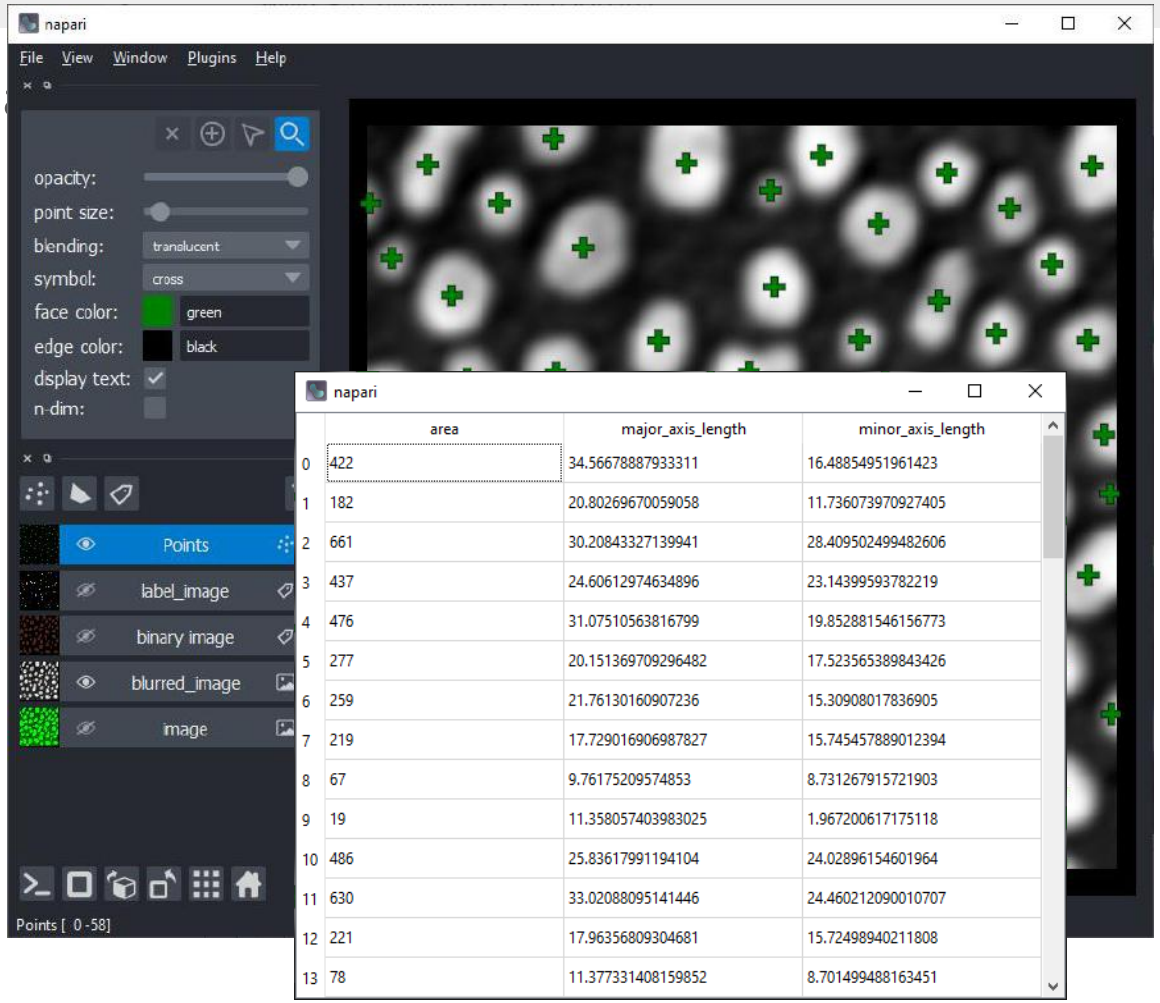



```
from skimage.measure import regionprops

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)
```

Tables

- Support for tables was added recently. Basic function

```
from skimage.measure import regionprops_table


statistics = regionprops_table(label_image,
                               properties=['area',
                              'major_axis_length',
                              'minor_axis_length'])


# Show statistics as table on screen
from magicgui.widgets import Table
Table(statistics).show()
```

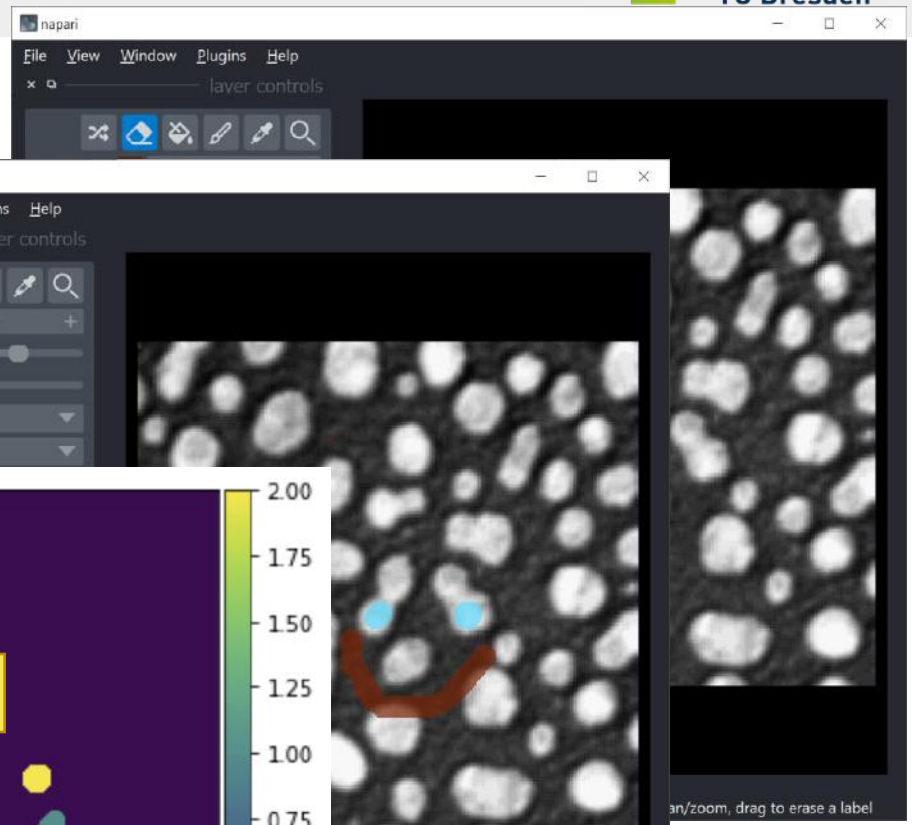


Interactive pixel classification

PoL Physics of Life TU Dresden

- Prepare an empty layer for annotations

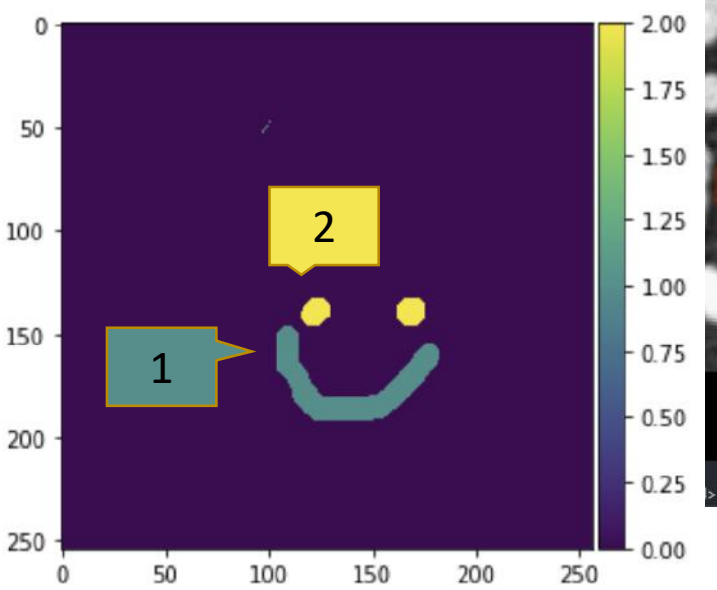
```
labels = viewer.add_labels(
    np.zeros(image.shape).astype(int))
```



- Read annotations

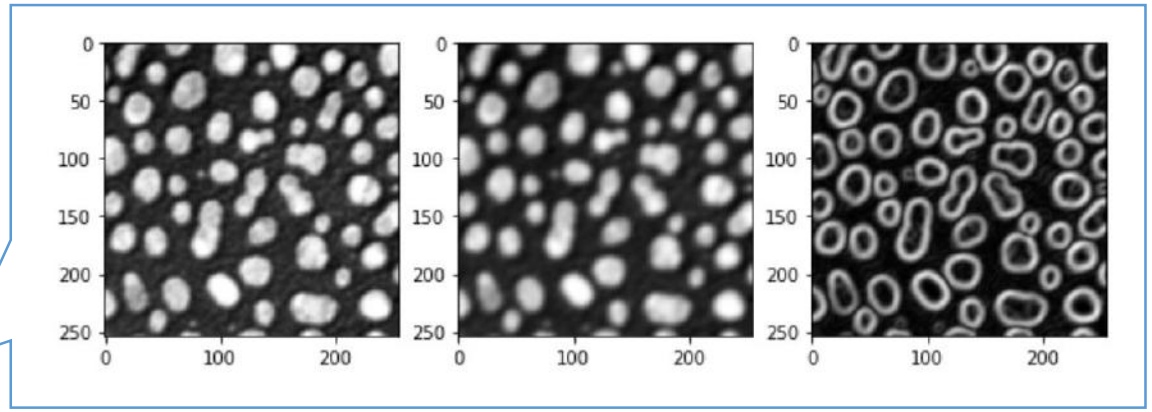
```
manual_annotations = labels.data
```

```
from skimage.io import imshow
imshow(manual_annotations,
               vmin=0, vmax=2)
```



Interactive pixel classification

- Pixel classification using scikit-learn

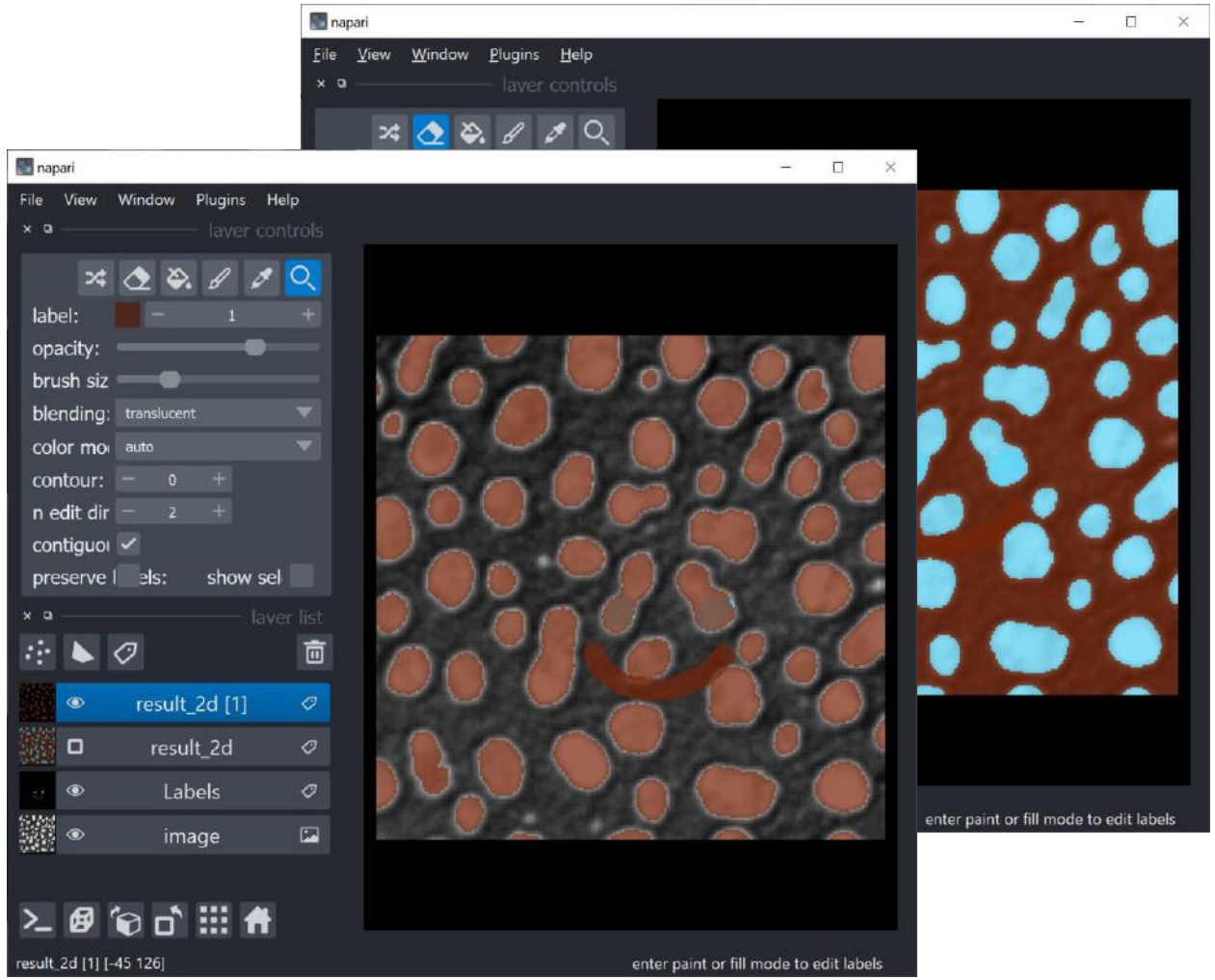


```
# for training, we need to generate features
feature_stack = generate_feature_stack(image)
X, y = format_data(feature_stack, manual_annotations)

# train classifier
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X, y)
```

Interactive pixel classification

- Pixel classification using scikit-learn

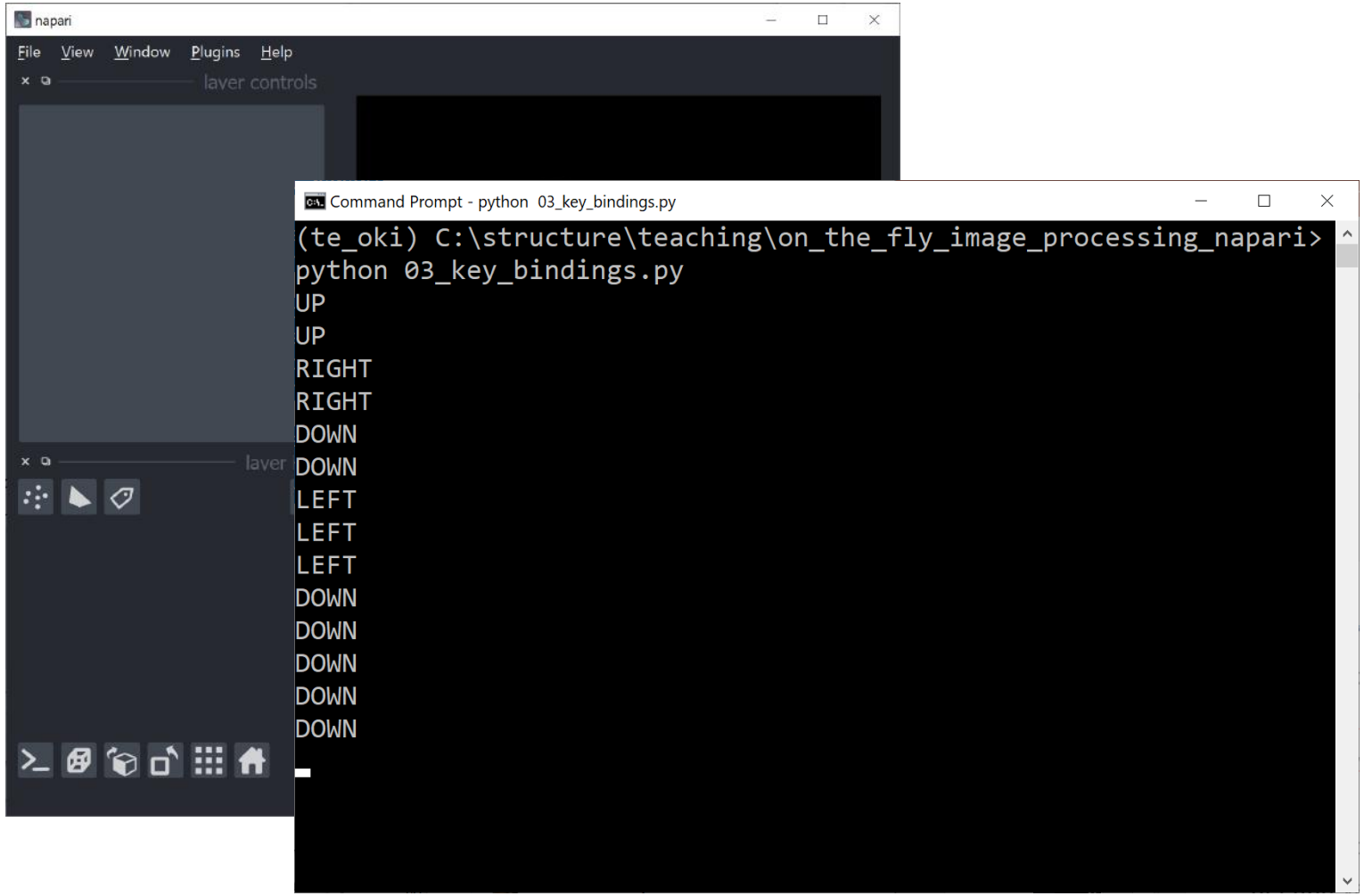
```
# process the whole image and show result
result_1d = classifier.predict(feature_stack.T)
result_2d = result_1d.reshape(image.shape)


# make sure background = 0
result_2d = result_2d - 1

viewer.add_labels(result_2d)
```



Key bindings

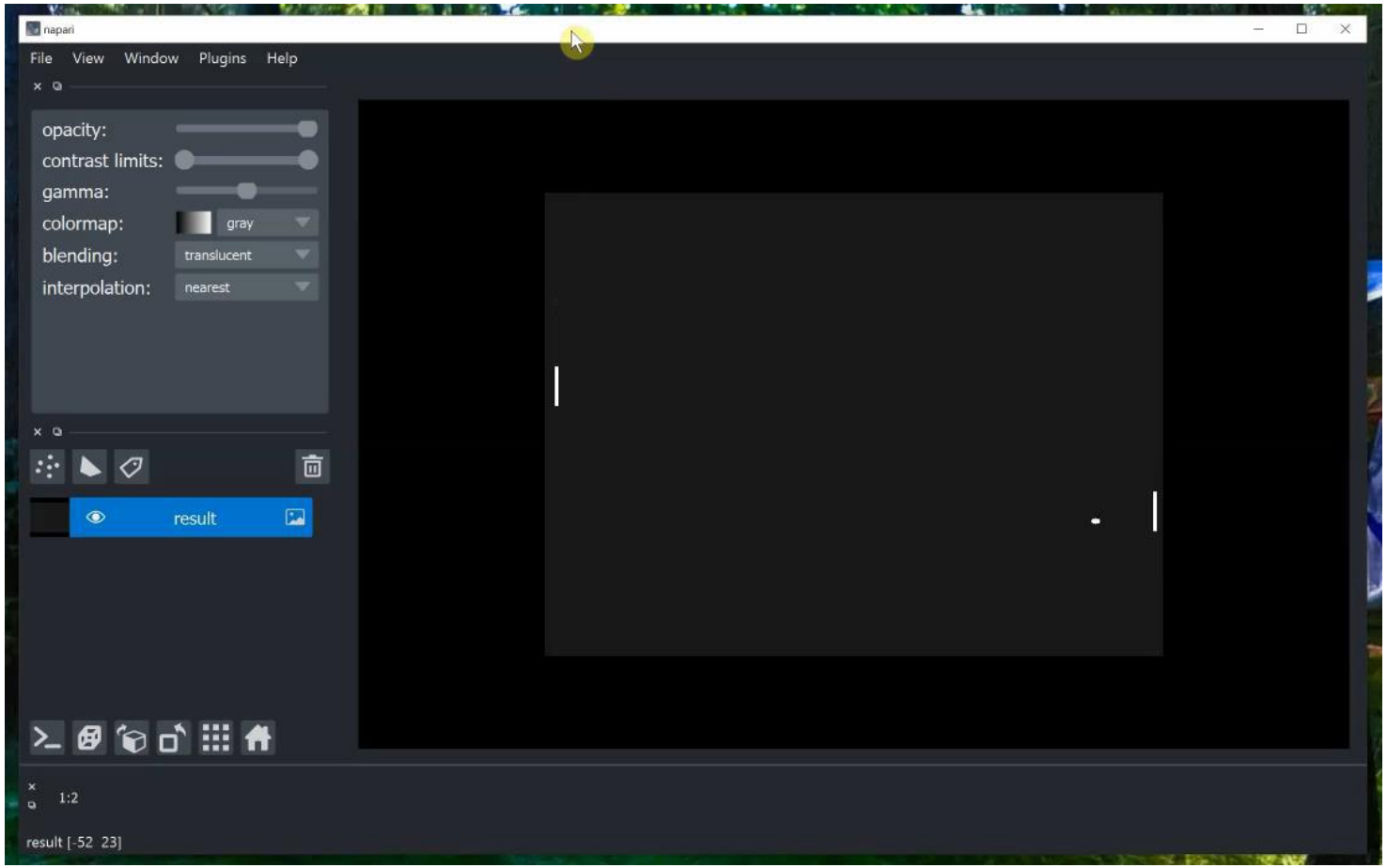
- Bind short-cuts to operations

```
import napari
viewer = napari.Viewer()


@viewer.bind_key("w")
def up(event):
    print("UP")


def left(event):
    print("LEFT")
viewer.bind_key("a", left)
```



Key bindings

- Control microscope stages, or



Multi-threading

- Python code typically runs in the main thread
- Multi-threading might be necessary for
  - User-interaction
  - Parallel processing
  - Device handling (camera, stage,…)

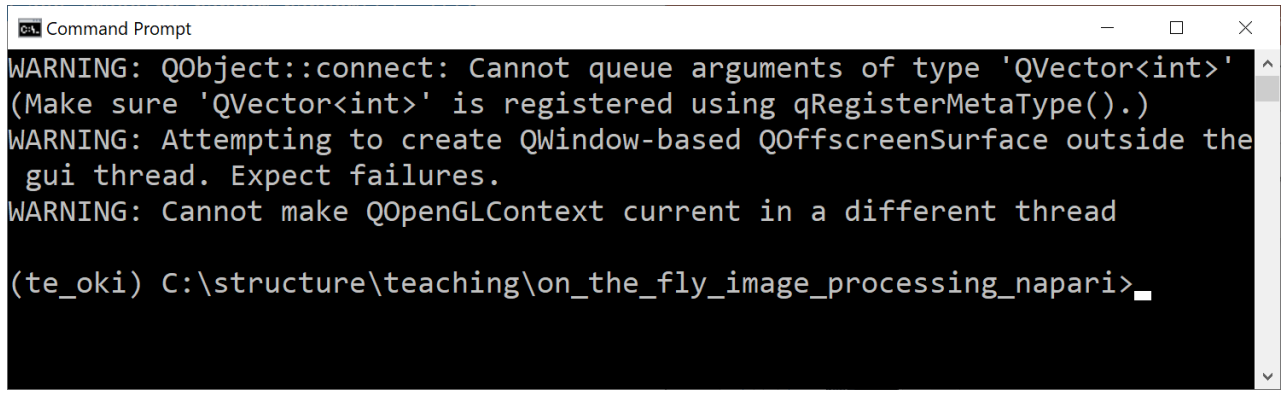
- Read more: https://napari.org/guides/stable/threading.html

```python
import napari
import time
from napari._qt.qthreading import thread_worker
viewer = napari.Viewer()

@thread_worker
def loop_run():
    while True: # endless loop
        print("Hello world", time.time())
        time.sleep(0.5)
        yield

# Start the loop
worker = loop_run()
worker.start()

# Start napari
napari.run()
```

Stops here once napari closes

Multi-threading: processing images

- Pitfall: You cannot access the viewer from other threads but the main thread.




```python
import napari
import time
from napari._qt.qthreading import thread_worker
viewer = napari.Viewer()

@thread_worker
def loop_run():
    while True: # endless loop
        viewer.add_image(np.random.random((2, 2)))
        time.sleep(0.5)
        yield

# Start the loop
worker = loop_run()
worker.start()

# Start napari
napari.run()
```

Multi-threading: processing images

- Solution: yield results
  - “Return something but continue processing”
  - a.k.a. Generator pattern
  - https://wiki.python.org/moin/Generators

- Add/update layers on demand

```
def update_layer(image):
    """
    Updates the image in the layer 'result'
    or adds this layer.
    """
    try:
        viewer.layers['result'].data = image
    except KeyError:
        viewer.add_image(image, name='result')
```

```
import napari
import time
from napari._qt.qthreading import thread_worker
viewer = napari.Viewer()

@thread_worker
def loop_run():
    while True: # endless loop
        yield np.random.random((2, 2))
        time.sleep(0.5)

# Start the loop
worker = loop_run()
worker.yielded.connect(update_layer)
worker.start()
```

Big thanks to

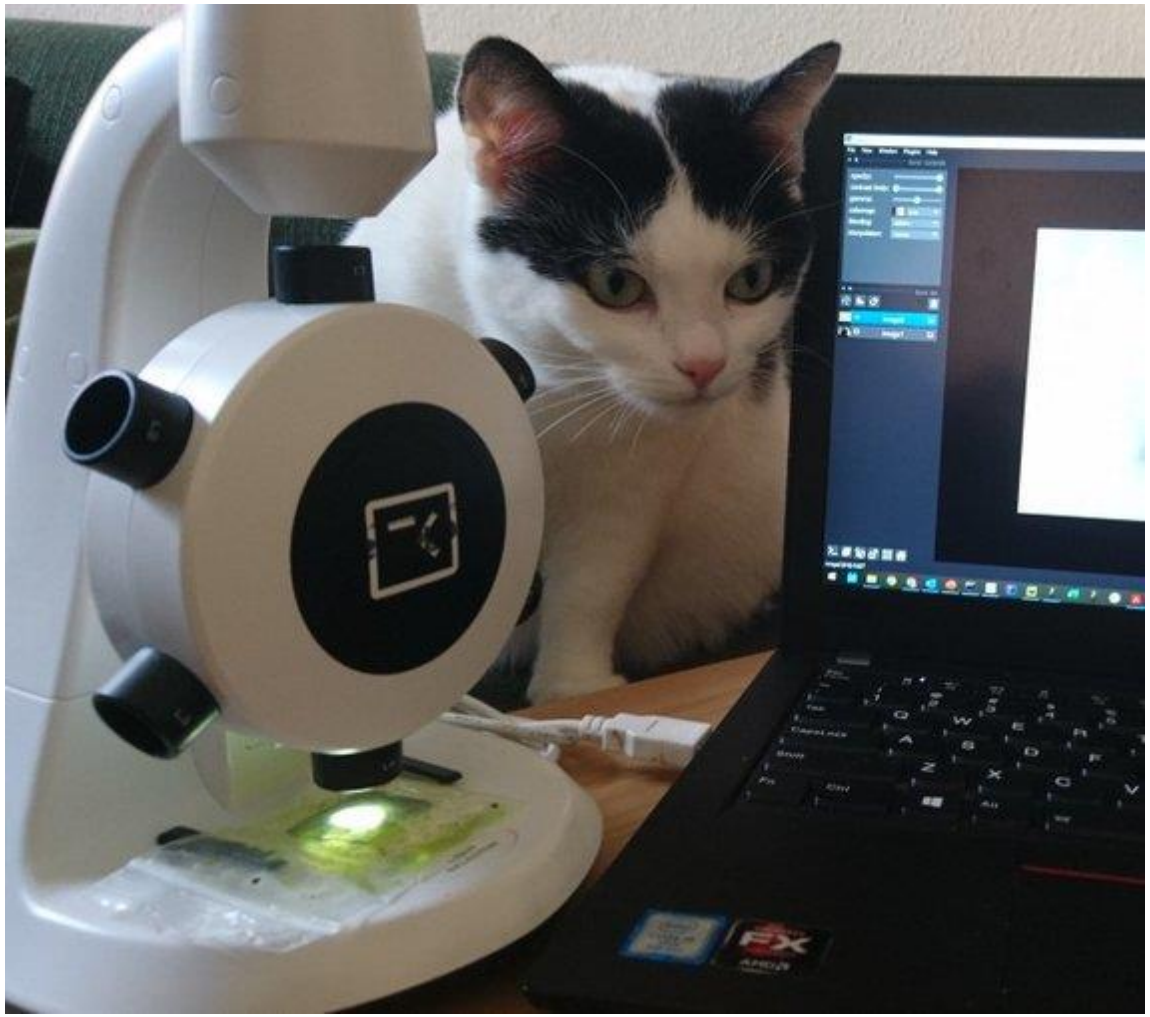
Talley J. Lambert (HMS) @TalleyJLambert

Reading camera output

PoL Physics of Life TU Dresden

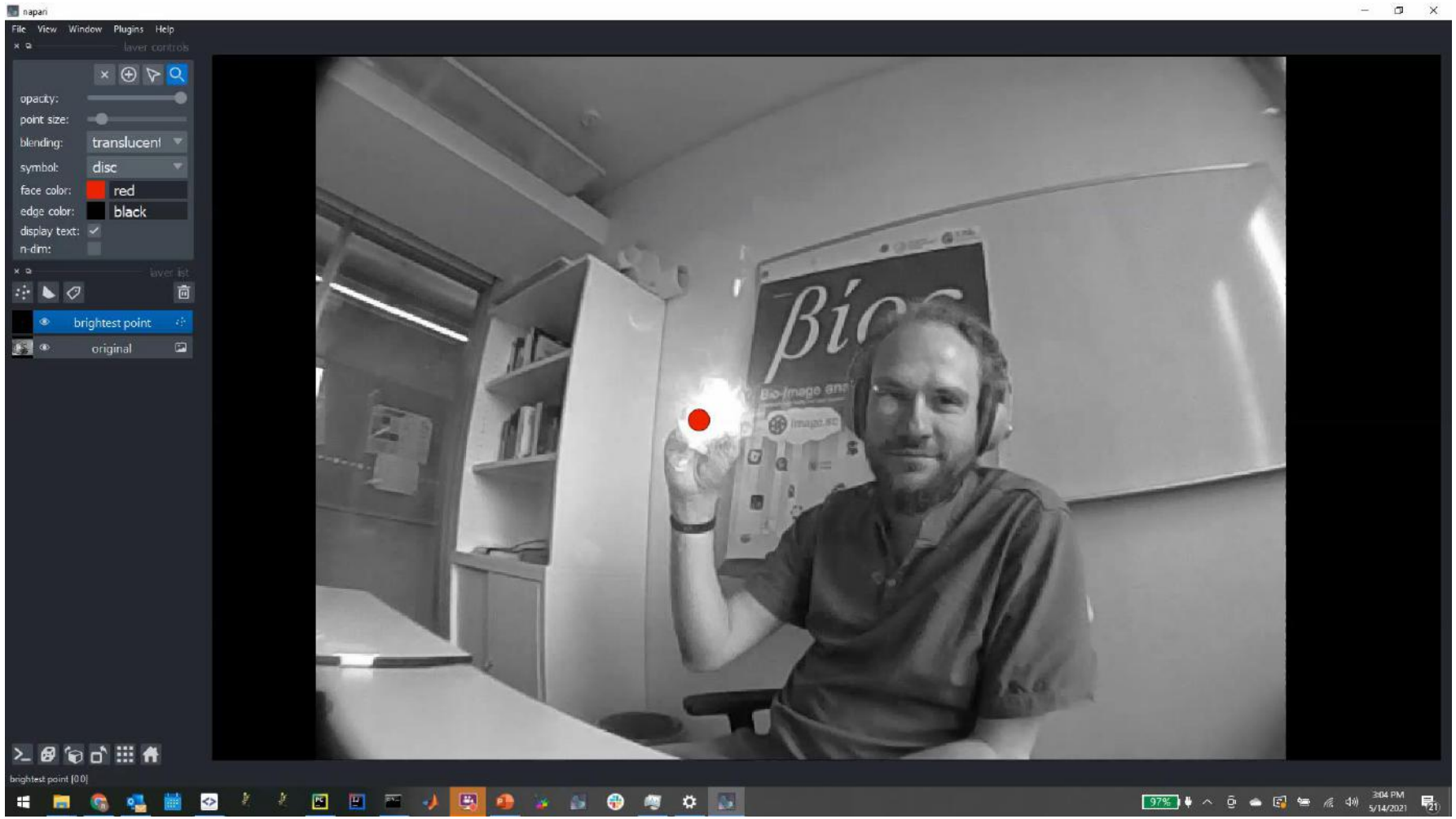
- Image acquisition: OpenCV

```
from cv2.cv2 import VideoCapture
from skimage.color import rgb2gray

camera = VideoCapture(camera_index)

def acquire_image(camera : VideoCapture):
    _, picture = camera.read()
    if picture is None:
        return

    # convert to single channel image
    return rgb2gray(picture)
```



Multi-threading: processing camera images

PoL
Physics of Life
TU Dresden



magicgui

- magicgui automatically generates a user interface for your function
- Read more: https://napari.org/magicgui/

```
def gaussian_blur(image, sigma = 2):
    """
    Apply a gaussian blur to an image.
    """
    return gaussian(image, sigma)
```

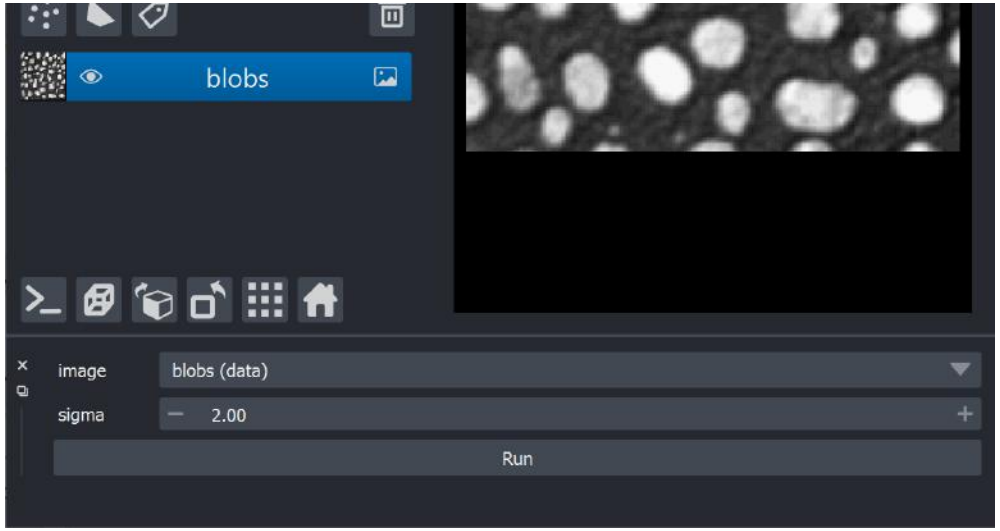
magicgui

- magicgui automatically generates a user interface for your function
- Read more: https://napari.org/magicgui/

Input parameters

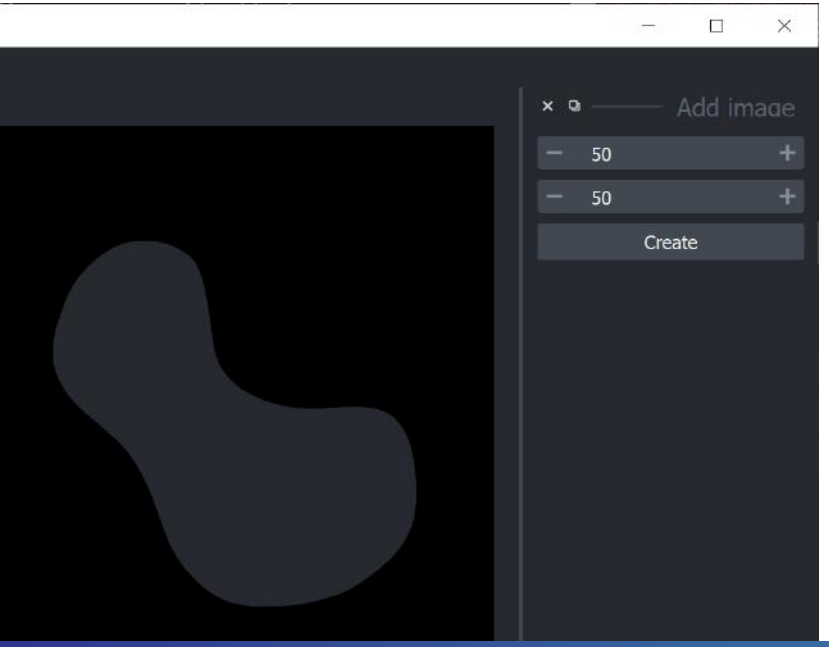
Return type

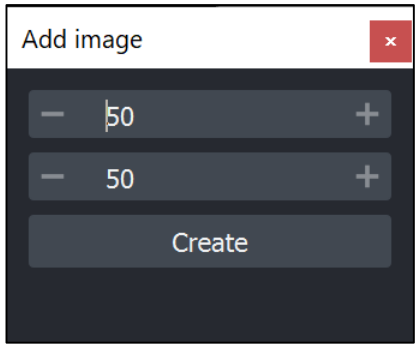
```
@magicgui(call_button='Run')
def gaussian_blur(image : ImageData, sigma : float = 2) -> ImageData:
    """
    Apply a gaussian blur to an image.
    """
    return gaussian(image, sigma)

# add user interface to napari viewer
viewer.window.add_dock_widget(gaussian_blur)
```



Dock widgets

- Dock widgets are user-interface plugins that snap to napari's window
- They have a title





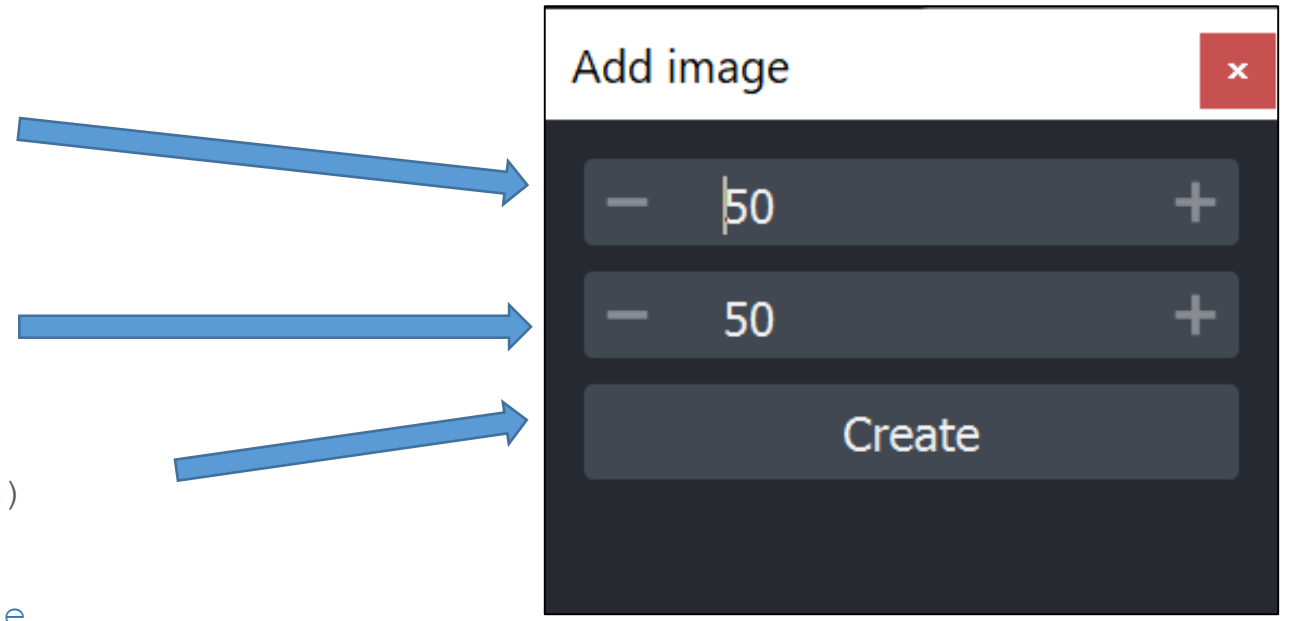
Undockable ;-)

```
import napari
from qtpy.QtWidgets import QWidget

viewer = napari.Viewer()

class AddImageWidget(QWidget):
    def __init__(self, napari_viewer: napari.Viewer):
        super().__init__()
        self._viewer = napari_viewer
        self.setObjectName("Add image")

# add the dock widget to the viewer
viewer.window.add_dock_widget(AddImageWidget(viewer))
```

Dock widgets

- User interfaces are made with Qt for python. Qt is the “cute” library. Read more: https://doc.qt.io/qtforpython/

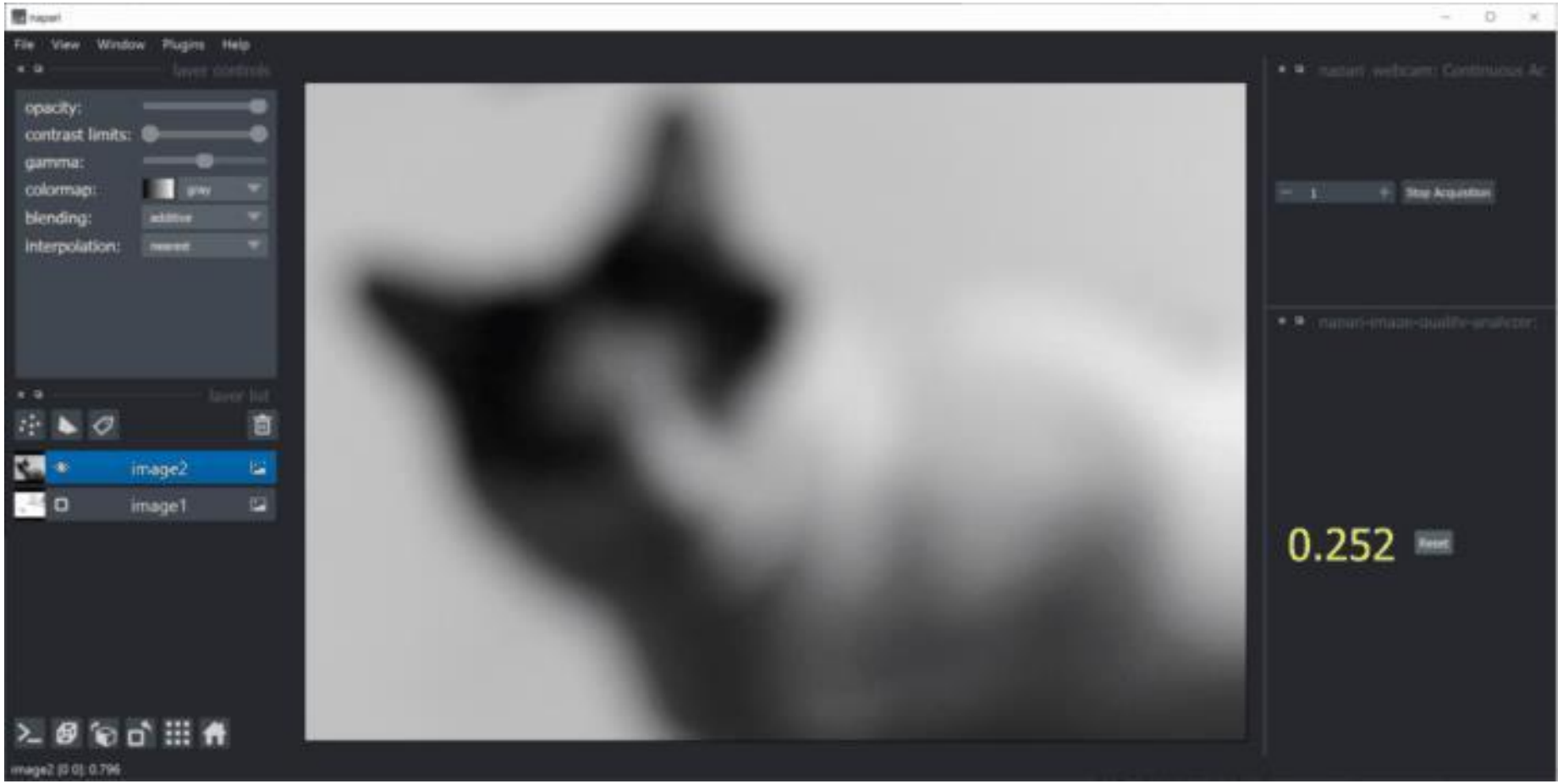
```
self.width_spinner = QSpinBox()
self.width_spinner.setValue(50)

self.height_spinner = QSpinBox()
self.height_spinner.setValue(50)

self.create_button = QPushButton("Create")
self.create_button.clicked.connect(self._create)

# put spinners and buttons in the user interface
self.setLayout(QVBoxLayout(self))
self.layout().addWidget(self.width_spinner)
self.layout().addWidget(self.height_spinner)
self.layout().addWidget(self.create_button)
```



On the fly processing + dock widgets

PoL Physics of Life TU Dresden

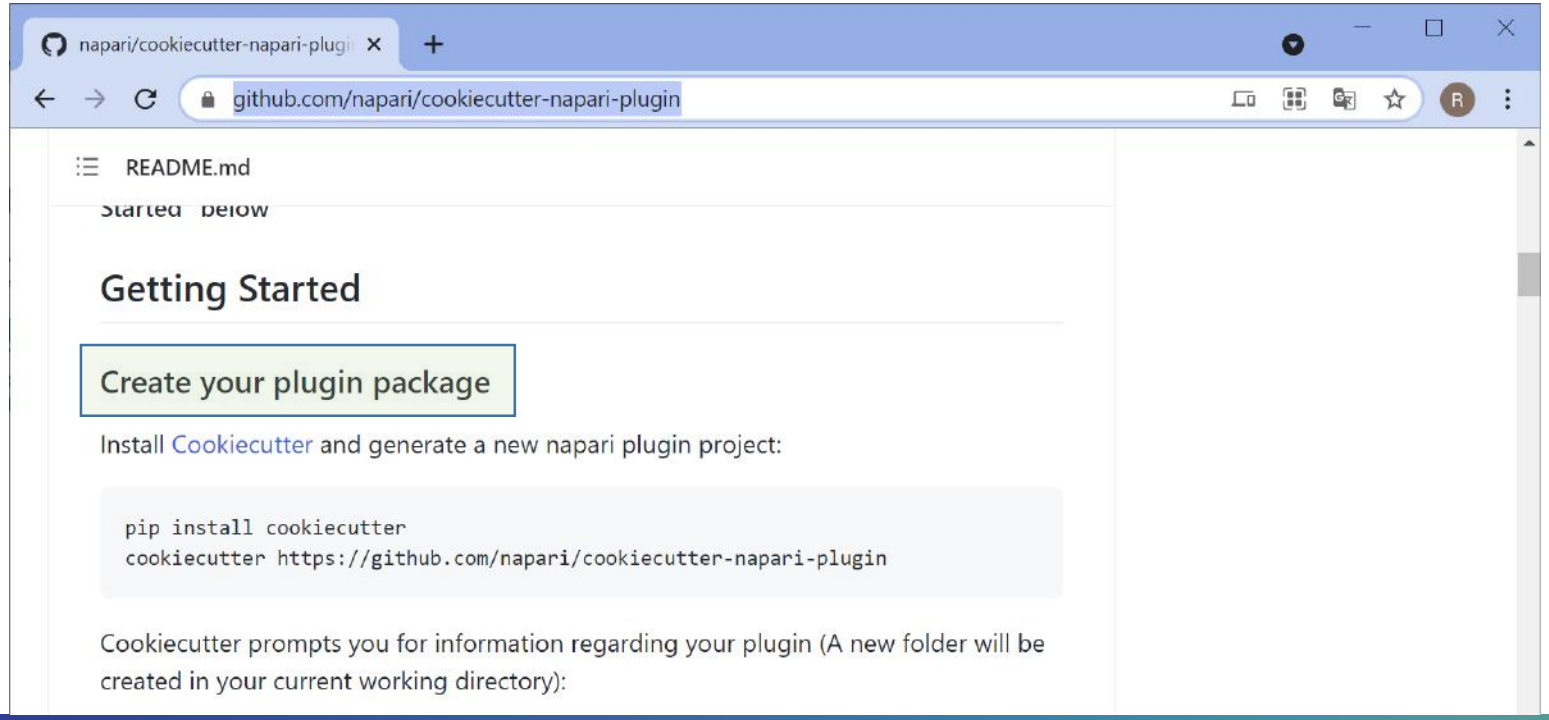
- Measure image quality on the fly



https://github.com/haesleinhuepf/napari-webcam
https://github.com/haesleinhuepf/napari-image-quality-analyzer

Napari plugins

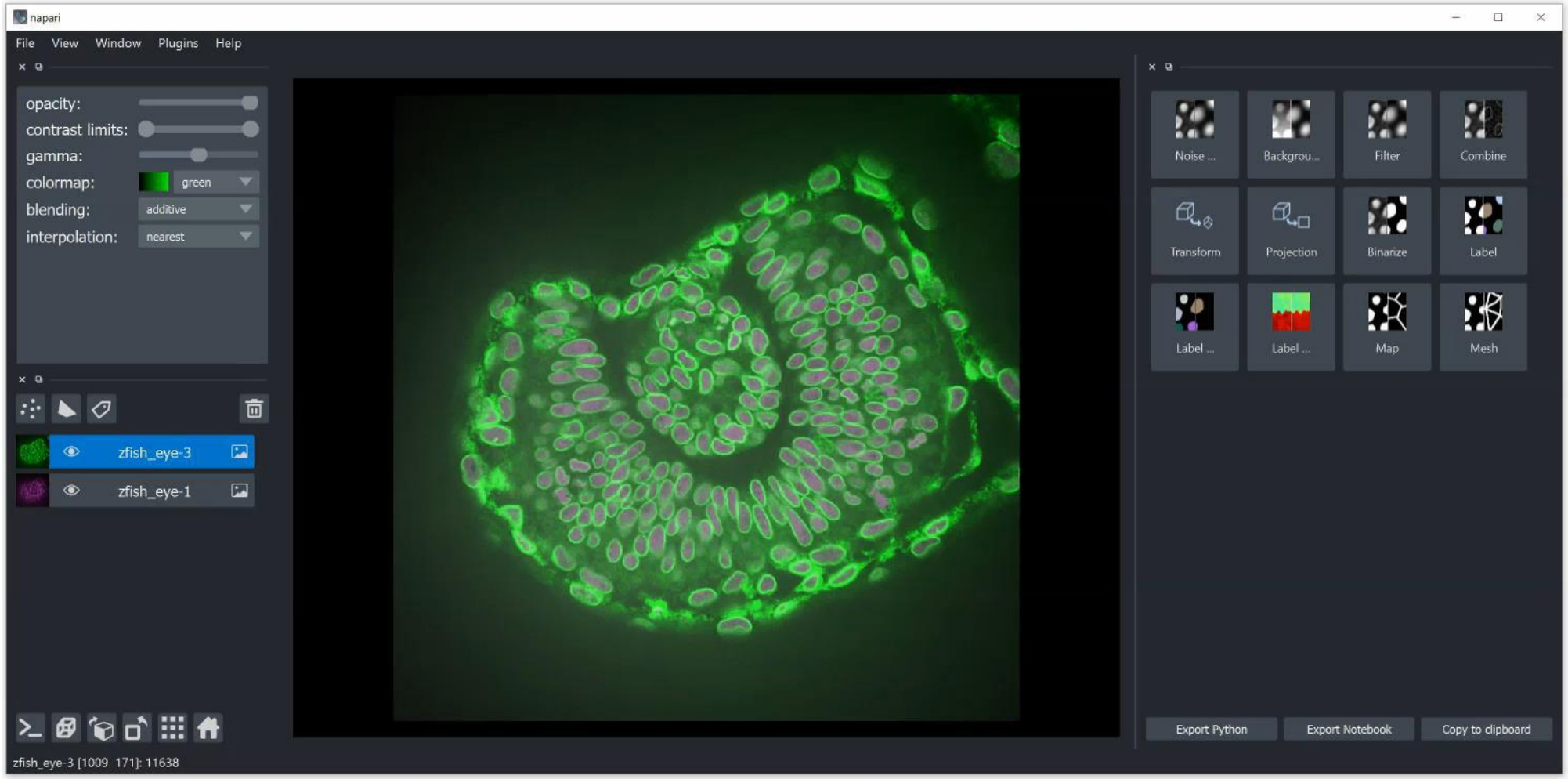
- Instead of adding functions and dock widgets to the napari viewer:
  - Make plugins!
  - Use the cookiecutter!



Learn from Rafa in our session on Friday!

clEsperanto: Building image data flow graphs in napari

PoL Physics of Life TU Dresden



Fantastic support by

Talley J. Lambert (HMS) @TalleyJLambert

@haesleinhuepf
@PoLDresden

https://github.com/clesperanto/napari_pyclesperanto_assistant

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

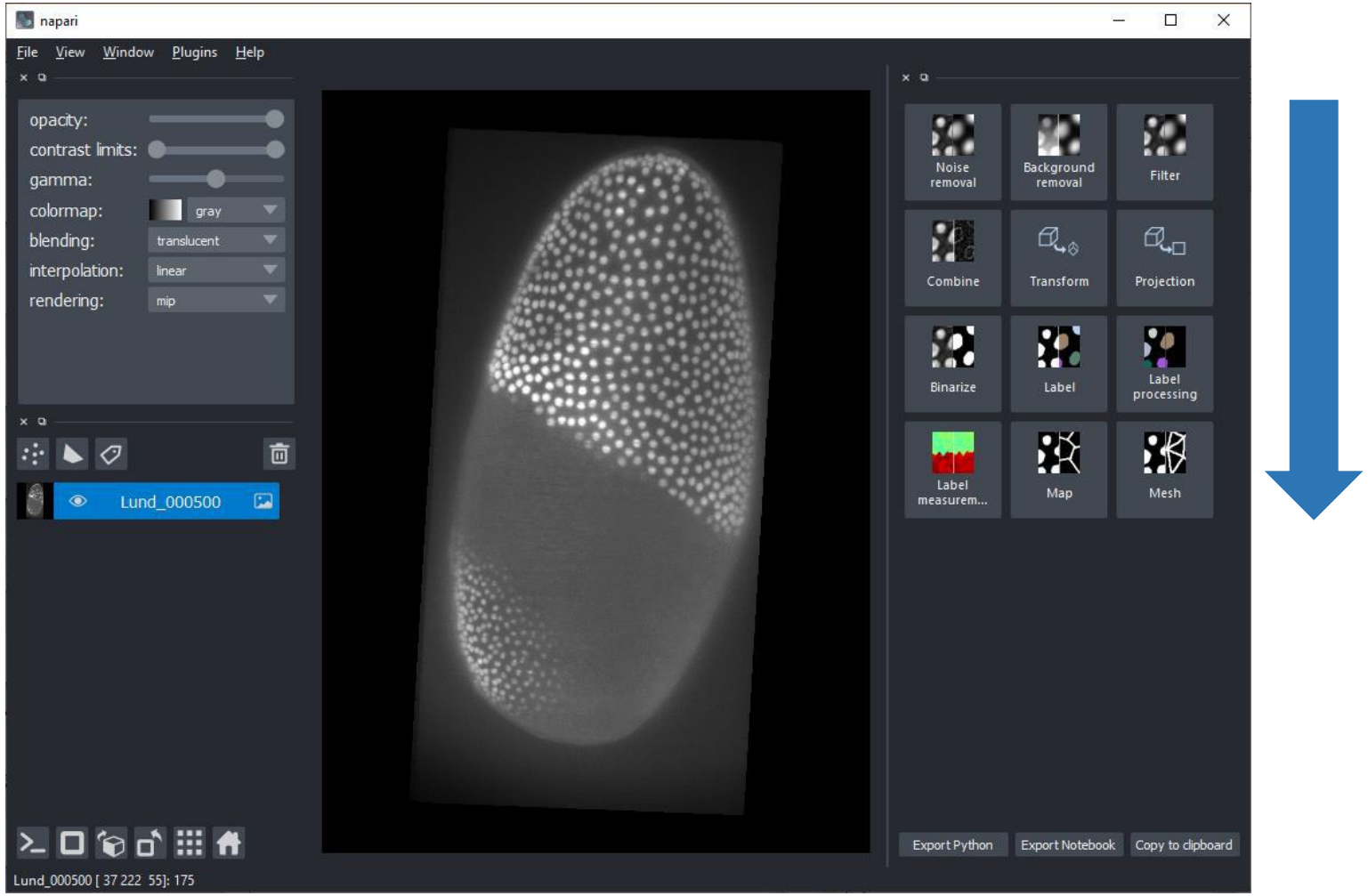
TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

clEsperanto: Building image data flow graphs in napari

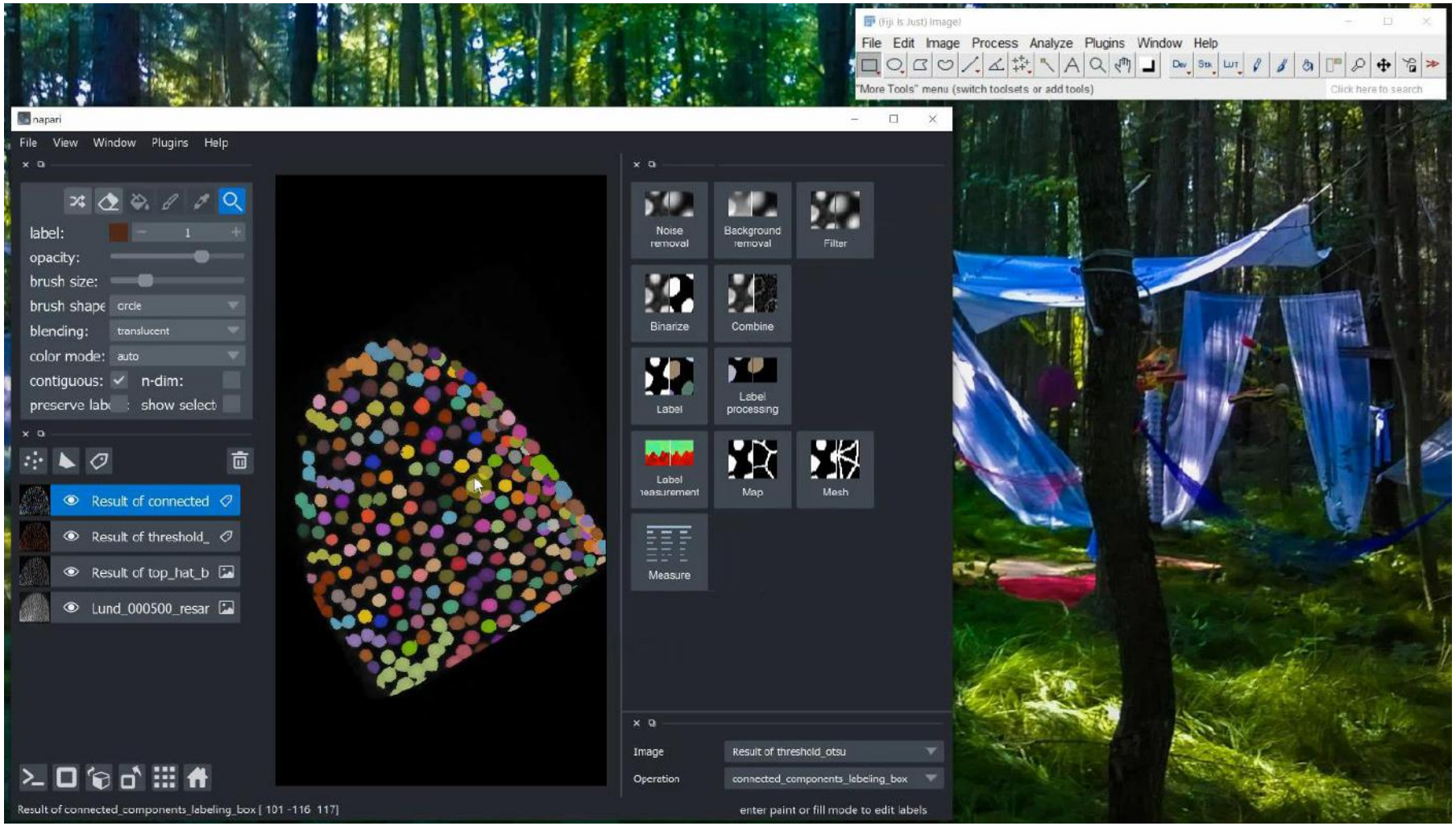
PoL Physics of Life TU Dresden

- The button order is intentional.
- Work from the top left to the bottom right



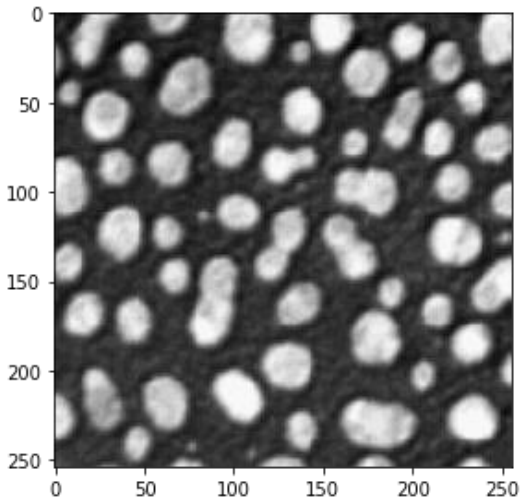
Code export

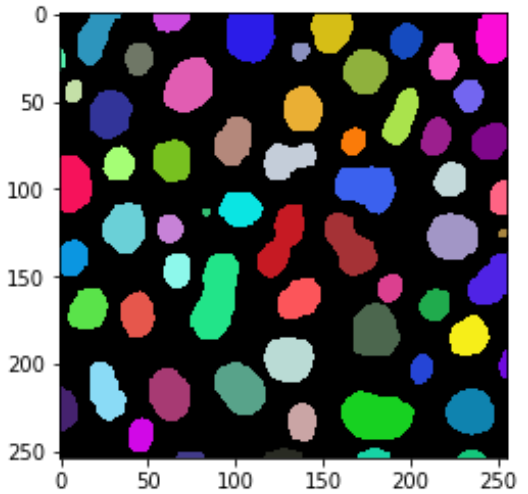
- Export code from napari and
  - run it in Fiji
  - run it in Python



The cle gateway

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

import pyclesperanto_prototype as cle

# noise removal
blurred = cle.gaussian_blur(image, sigma_x=1, sigma_y=1)
# binarization
binary = cle.threshold_otsu(blurred)
# labeling
labels = cle.connected_components_labeling_box(binary)
# visualize results
cle.imshow(labels, labels=True)
```





Fantastic support by

Juan Nunez-Iglesias (Monash) @jnuneziglesias

Talley J. Lambert (HMS) @TalleyJLambert

The `cle` gateway

PoL Physics of Life TU Dresden

- … allows you to explore available functions for given purposes

```
cle.operations('denoise').keys()
```

```
dict_keys(['gaussian_blur', 'mean_box', 'mean_sphere'])
```

```
cle.operations('background removal').keys()
```

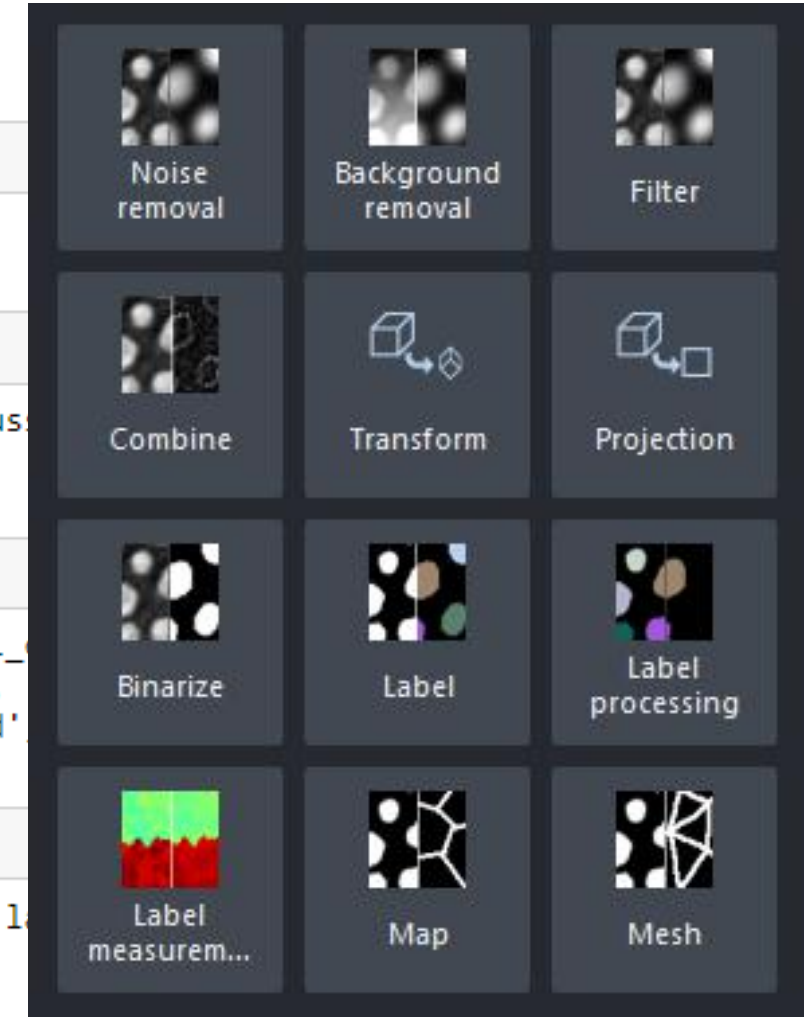
```
dict_keys(['bottom_hat_box', 'bottom_hat_sphere', 'difference_of_gaussian', 'divide_by_gaus
_background', 'top_hat_box', 'top_hat_sphere'])
```

```
cle.operations('binarize').keys()
```

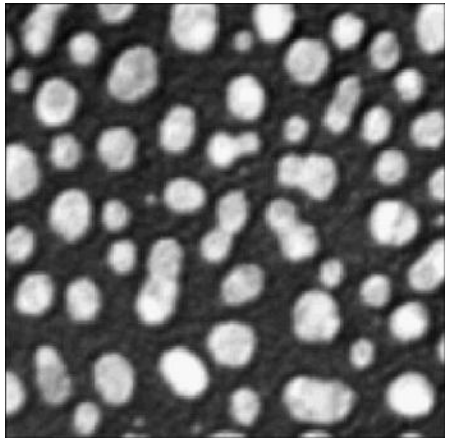
```
dict_keys(['detect_label_edges', 'detect_maxima_box', 'detect_minima_box', 'equal', 'equal_
ant', 'greater_or_equal', 'greater_or_equal_constant', 'label_to_mask', 'local_threshold',
'smaller', 'smaller_constant', 'smaller_or_equal', 'smaller_or_equal_constant', 'threshold'
```

```
cle.operations('label').keys()
```

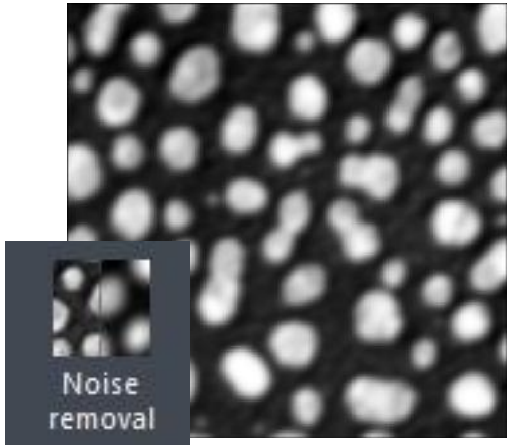
```
dict_keys(['connected_components_labeling_box', 'connected_components_labeling_diamond', 'l
g', 'voronoi_labeling', 'voronoi_otsu_labeling'])
```



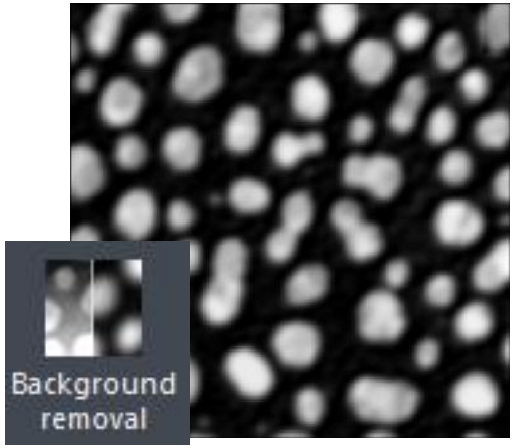
clEsperanto categories



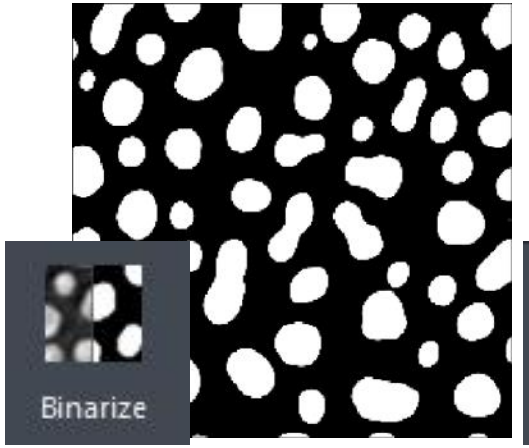
Original



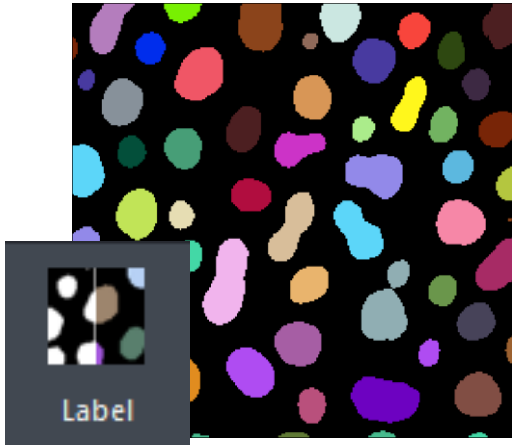
Gaussian blur



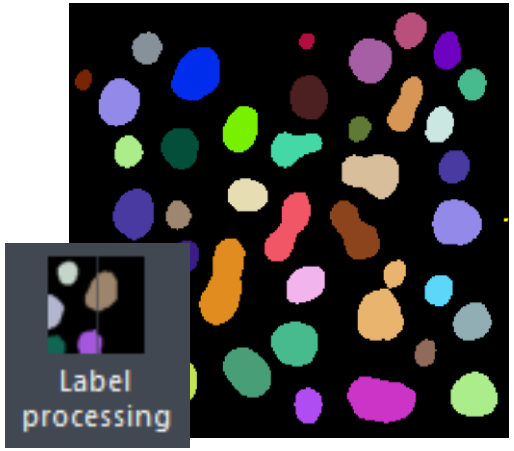
Top hat



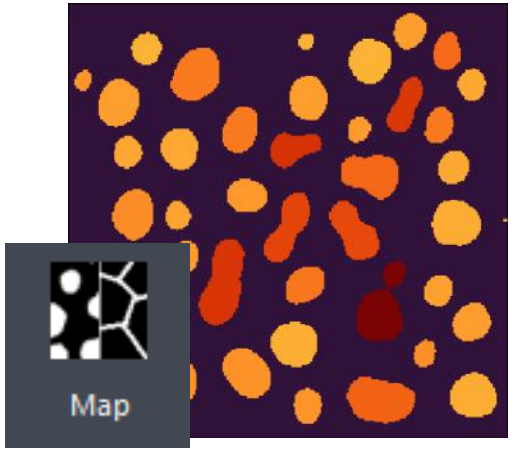
Threshold Otsu



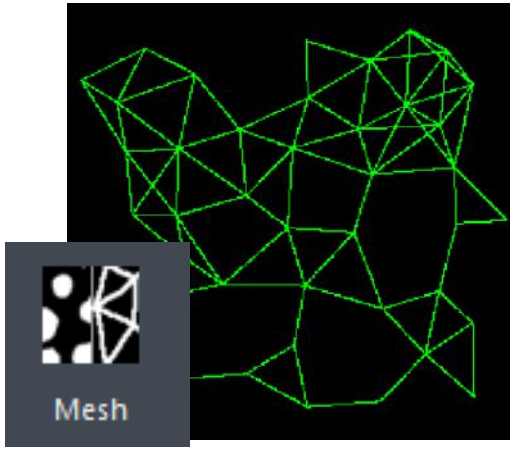
Connected Component



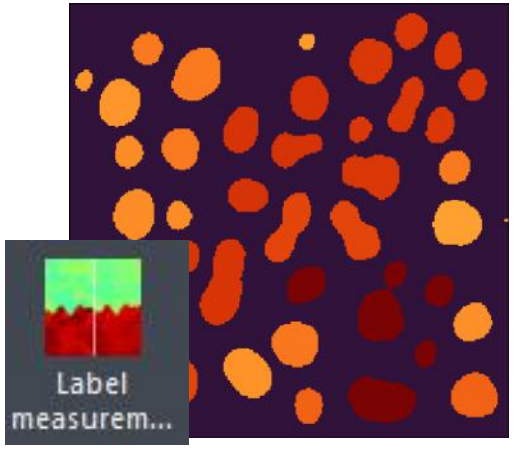
Exclude edges



Extension ratio



Mesh proximal neighbors



Maximum of n-nearest n.

clEsperanto: interoperability

PoL Physics of Life TU Dresden

- clEsperanto's images are numpy-compatible arrays.

```
from skimage.measure import regionprops
statistics = regionprops(labels)

import numpy as np
np.mean([s.area for s in statistics])
```

- Sometimes, they have to be converted to numpy arrays:

```
from skimage.io import imsave
imsave('output.tif', np.asarray(labels))
```

clEsperanto: GPU-accelerated image processing

PoL Physics of Life TU Dresden

- Why?
- Because operations can run orders of magnitude faster!

CUDA based GPU-acceleration:
https://cupy.dev/
https://github.com/rapidsai/cucim

Scipy

```
import time
from scipy import ndimage as sndi

scaled = np.ndarray(output_shape)
for i in range(0, 10):
    start_time = time.time()
    sndi.affine_transform(image, matrix, output=scaled, out
    print("scipy affine transform duration: " + str(time.ti

imshow(scaled[120])
```

```
scipy affine transform duration: 9.005393266677856
scipy affine transform duration: 9.04822826385498
scipy affine transform duration: 9.089797019958496
scipy affine transform duration: 8.864994764328003
scipy affine transform duration: 9.063815832138062
scipy affine transform duration: 8.975829362869263
scipy affine transform duration: 8.582808256149292
scipy affine transform duration: 9.115242958068848
scipy affine transform duration: 9.170432567596436
scipy affine transform duration: 9.086284637451172
```

clEsperanto

```
ocl_image = cle.push(image)

ocl_scaled = cle.create(output_shape)
for i in range(0, 10):
    start_time = time.time()
    cle.affine_transform(ocl_image, ocl_scaled, transform=np.lina
    print("clEsperanto affine transform duration: " + str(time.ti

result = cle.pull(ocl_scaled)
imshow(result[120])
```

```
clEsperanto affine transform duration: 0.036901473999023 44
clEsperanto affine transform duration: 0.003954648971557617
clEsperanto affine transform duration: 0.0029942989349365234
clEsperanto affine transform duration: 0.00398564338684082
clEsperanto affine transform duration: 0.0030252933502197266
clEsperanto affine transform duration: 0.002960205078125
clEsperanto affine transform duration: 0.003988981246948242
clEsperanto affine transform duration: 0.003989696502685547
clEsperanto affine transform duration: 0.0030260086059570312
clEsperanto affine transform duration: 0.003954887390136719
```

cupy

```
import time
import cupy
from cupyx.scipy import ndimage as ndi

cuda_image = cupy.asarray(image)

cuda_scaled = cupy.ndarray(output_shape)
for i in range(0, 10):
    start_time = time.time()
    ndi.affine_transform(cuda_image, cupy.asarray(matrix), output=cuda_scaled, output_shape=output_shape)
    cupy.cuda.stream.get_current_stream().synchronize() # we need to wait here to measure time properly
    print("cupy affine transform duration: " + str(time.time() - start_time))

result = cupy.asnumpy(cuda_scaled)
imshow(result[120])
```

```
C:\Programs\miniconda3\envs\beetlesafari\lib\site-packages\cupyx\scipy\ndimage\interpolation.py:15: UserWa
he default order of affine_transform is 1. It is different from scipy.ndimage and can change in the future
  warnings.warn('In the current feature the default order of {} is 1. '
cupy affine transform duration: 0.4258244037628174
cupy affine transform duration: 0.01000523567199707
cupy affine transform duration: 0.010988473892211914
cupy affine transform duration: 0.010972261428833008
cupy affine transform duration: 0.010921955108642578
cupy affine transform duration: 0.011012792587280273
cupy affine transform duration: 0.010975122451782227
cupy affine transform duration: 0.009923934936523438
cupy affine transform duration: 0.011012554168701172
cupy affine transform duration: 0.010970354080200195
```

Note: This is just a snapshot! GPU-acceleration speedup / performance depends on many aspects such as image size, operation, parameters, used GPU, repetions, …

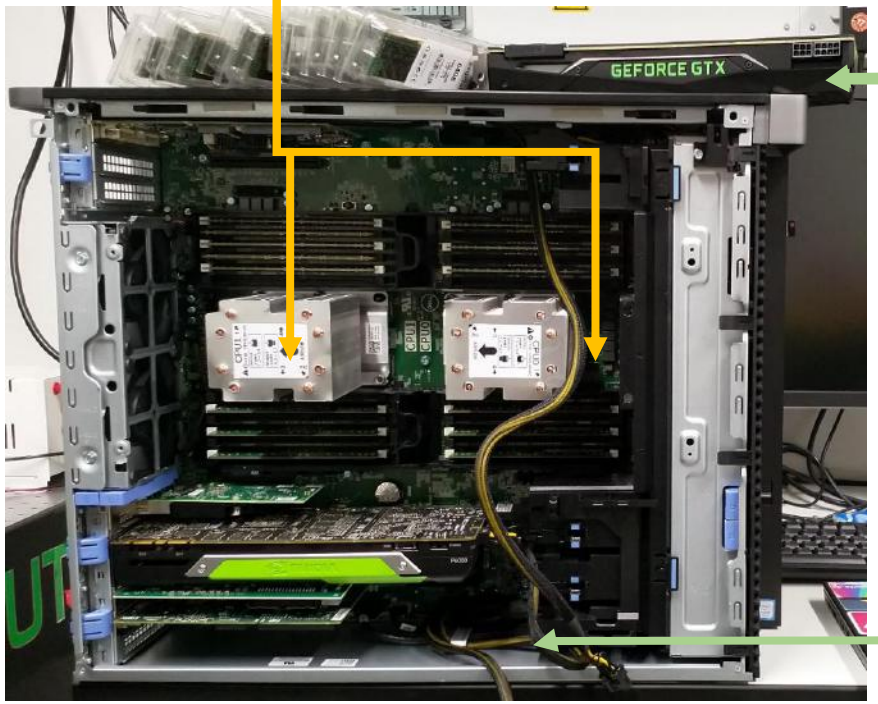
Graphics Processing Units (GPUs)

PoL Physics of Life TU Dresden

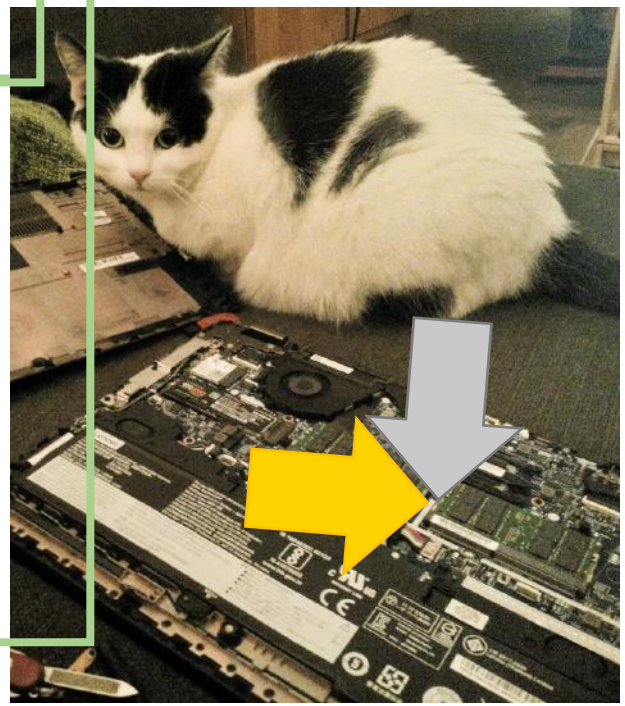
- … don’t have to be expensive.

Central Processing Unit (CPU)

Graphics Processing Unit (GPU)

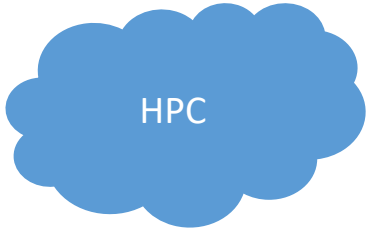


dedicated GPUs



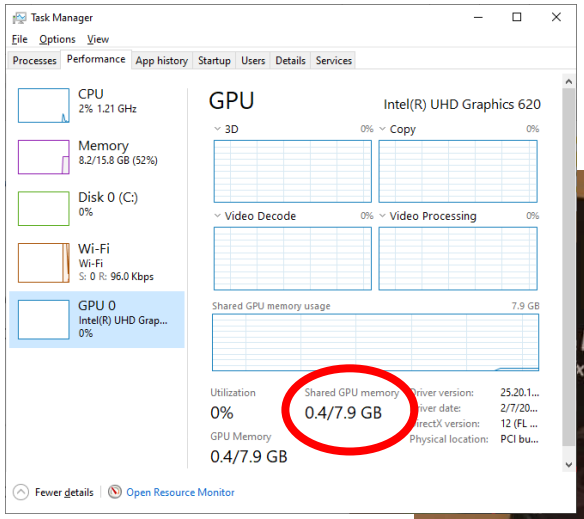
integrated GPUs

external “eGPUs”



Checklist

When does GPU-accelerated image processing make sense?



You need

- a workflow that is slow, ideally:
  - image processing time 10 : 1 image loading time,
- Enough graphics card memory,
  - > 5 times image size
- Basic coding experience
- A recent Intel, AMD or Nvidia GPU (integrated, dedicated, external)

If you really want to get the most out of it:

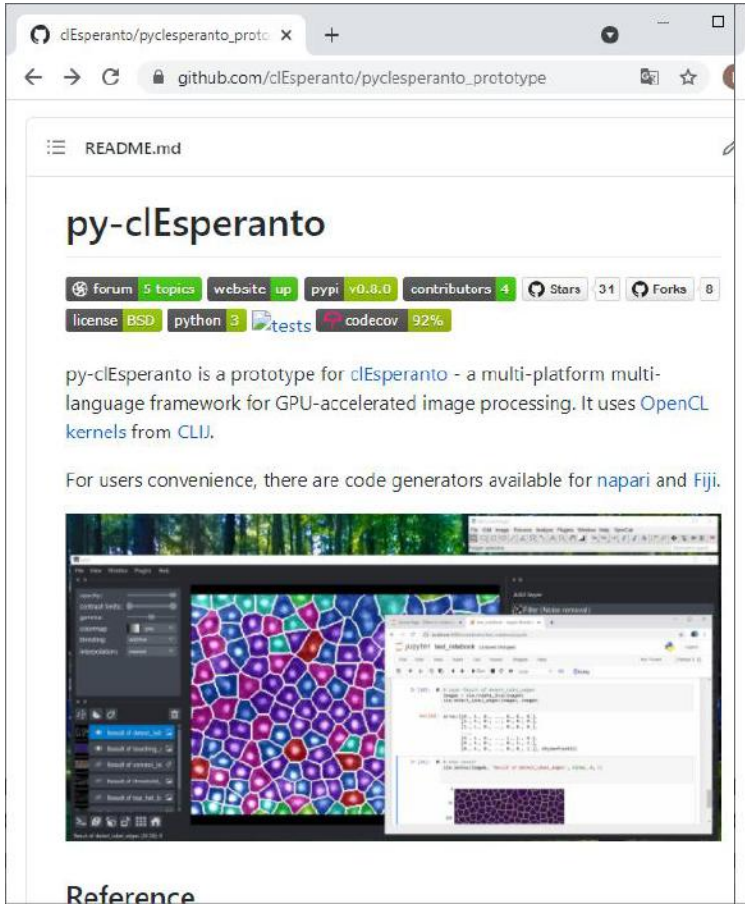
- a graphics card with GDDR6 memory (memory bandwidth > 400 Gb/s).

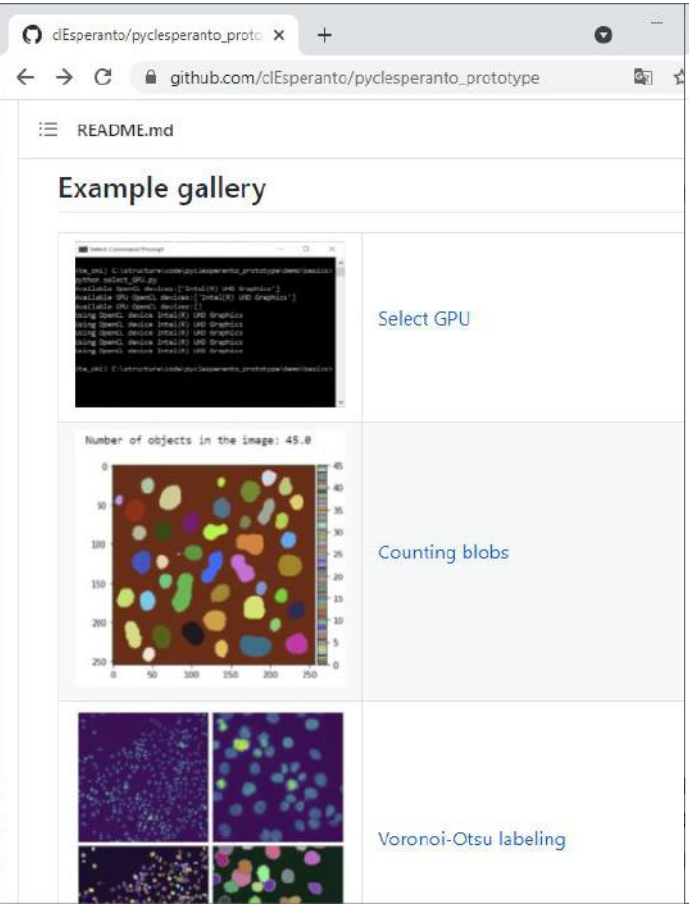
Online tutorials: clEsperanto

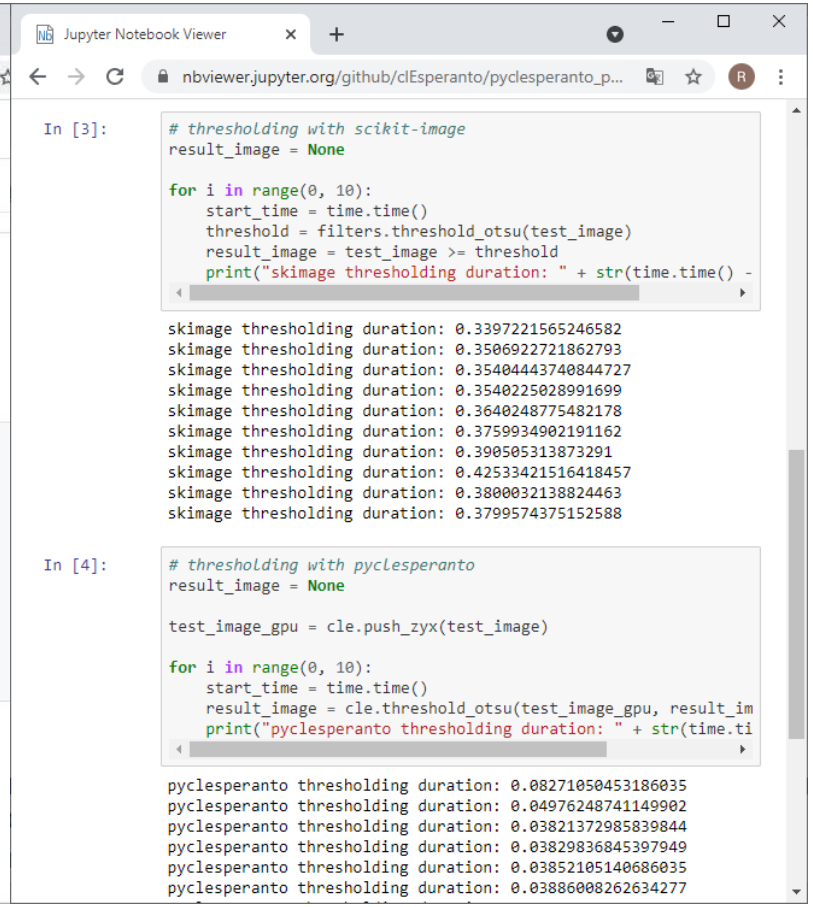
PoL Physics of Life TU Dresden

- The py-clEsperanto website offers examples and simple benchmarking for snake county.

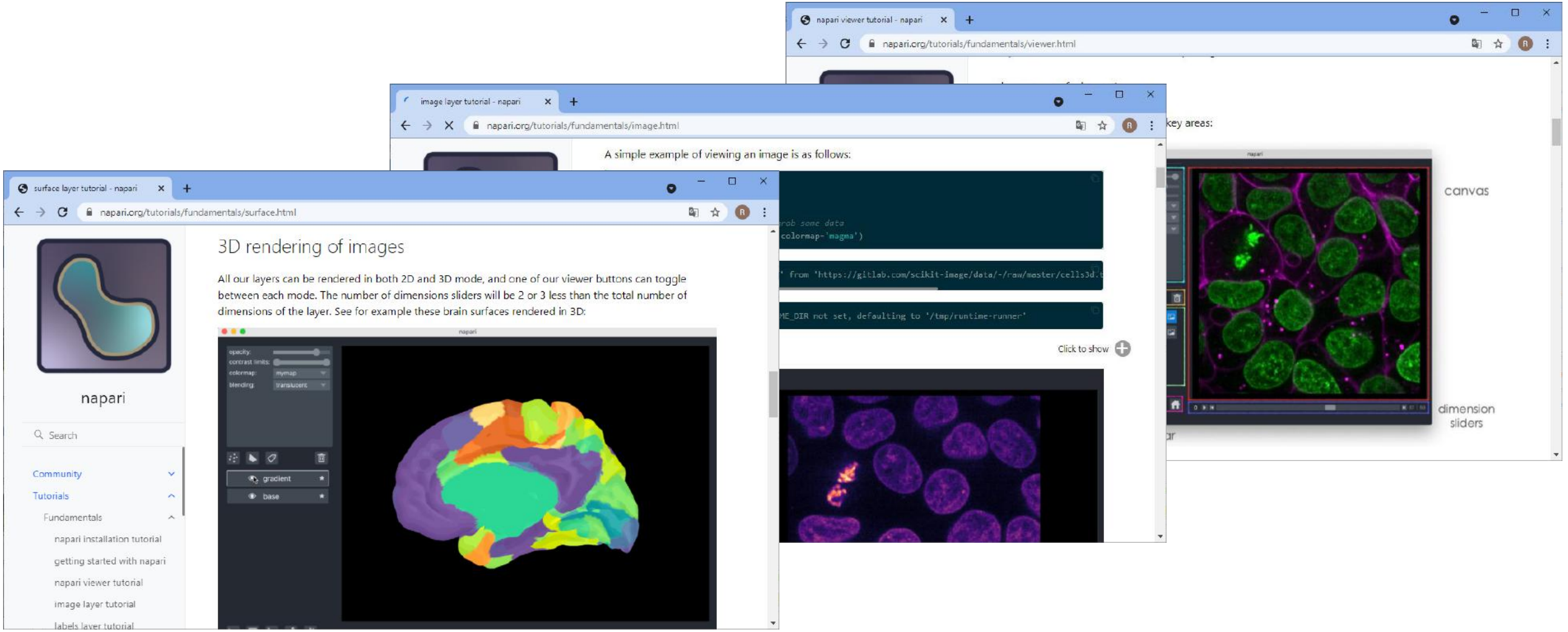
cle._







Online tutorials: napari

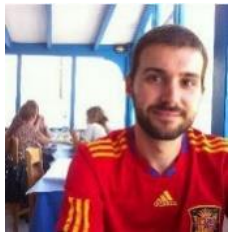


Acknowledgements

Dani Vorkel (Myers lab) @happifocus

Pradeep Rajasekhar (Monash U) @pr4deepr

Stéphane Rigaud (Institut Pasteur) @StRigaud



Juan Nunez-Iglesias (Monash) @jnuneziglesias

Talley J. Lambert (HMS) @TalleyJLambert

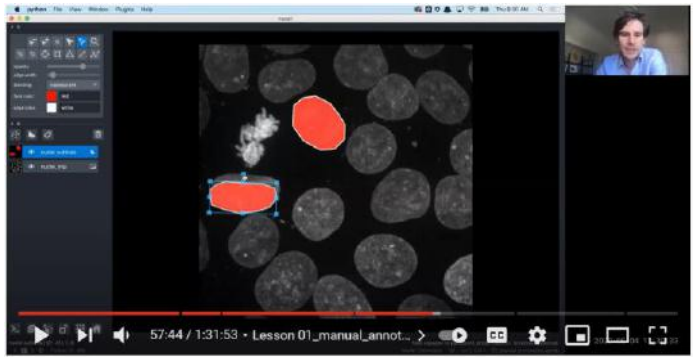
Nicholas Sofroniew (CZI) @sofroniewn

https://napari.dev

https://image.sc

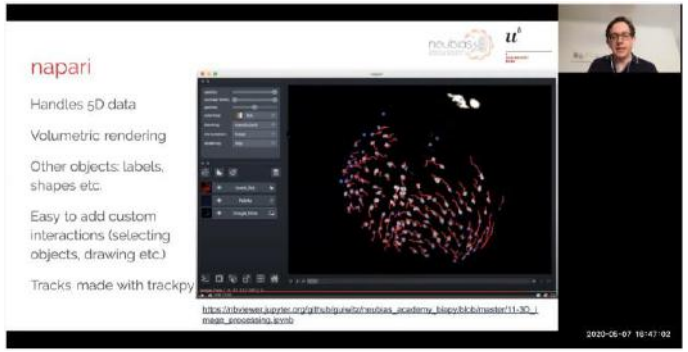
http://eubias.org/NEUBIAS/

#OpenKnowledge



https://www.youtube.com/watch?v=VgvDSq5aCDQ

https://www.youtube.com/watch?v=rOO6tK7z6yk



https://www.youtube.com/watch?v=2KF8vBrp3Zw

Funding:

DFG public

DFG 2020 FÜR DAS WISSEN ENTSCHEIDEN

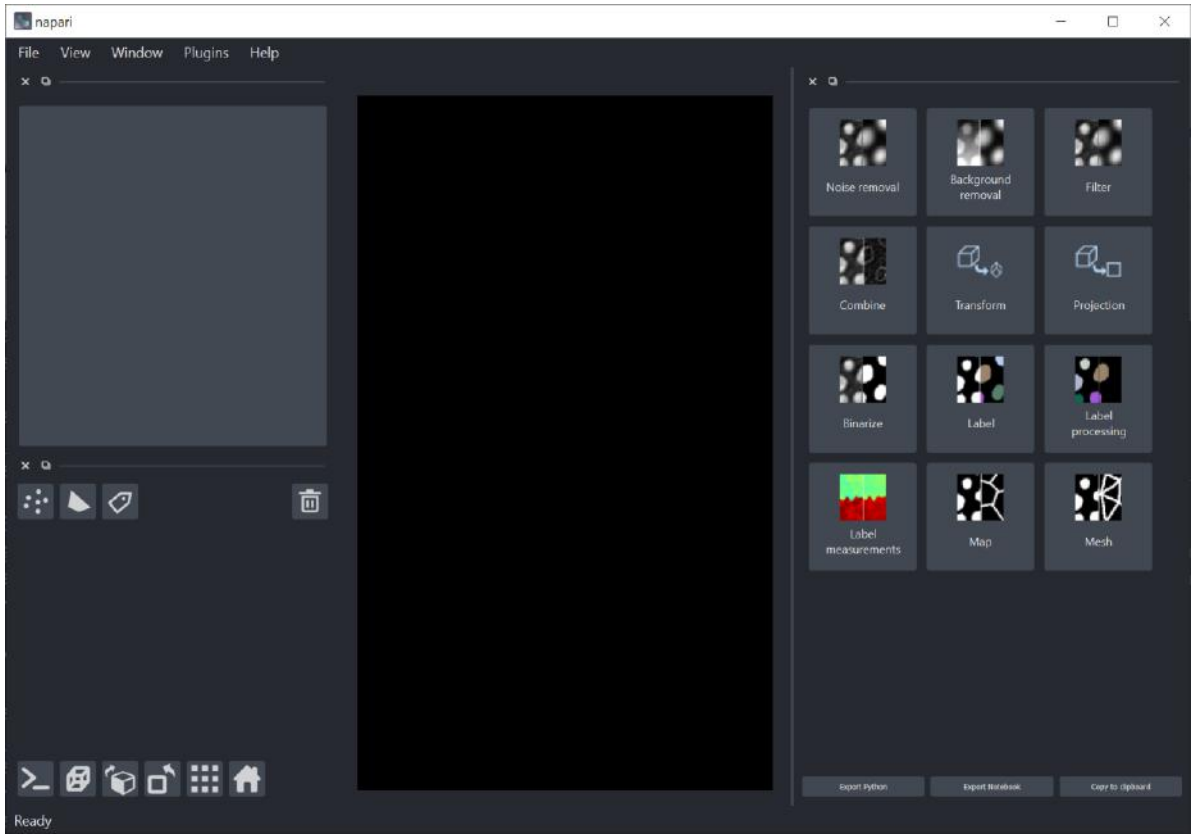
R.H. was supported by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy – EXC2068 - Cluster of Excellence Physics of Life of TU Dresden.

Let's take a break

```
git clone https://github.com/BiAPoL/on_the_fly_image_processing_napari.git
cd on_the_fly_image_processing_napari
pip install -r requirements.txt
```

```
conda install -c conda-forge pyopencl
pip install napari-pyclesperanto-assistant
```

```
napari --with clEsperanto Assistant
```



Linux users: you may need to install OpenCL, e.g.

```
apt-get install ocl-icd-devel
```