

Machine Learning for Pixel and Object Segmentation

Robert Haase

With Material from

Deborah Schmidt, Jug Lab, MPI CBG

Uwe Schmidt, Myers Lab, MPI CBG

Martin Weigert, EPFL

Ignacio Arganda-Carreras, Universidad del Pais Vasco

Carsen Stringer, HHMI Janelia

Wei Ouyang, KTH Royal Institute of Technology, Stockholm and

The Scikit-Learn community

Overview

- Machine learning for Pixel and Object Classification
 - Random Forest Classifiers
- Python
 - scikit-learn / napari
 - Accelerated pixel and object classification (APOC)

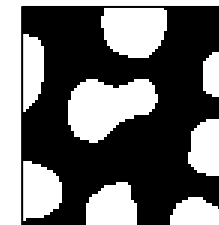
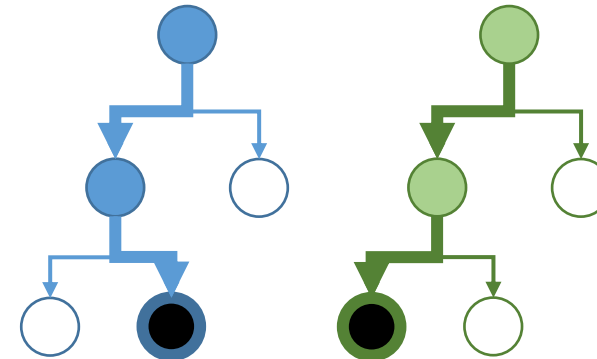
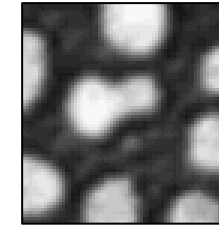
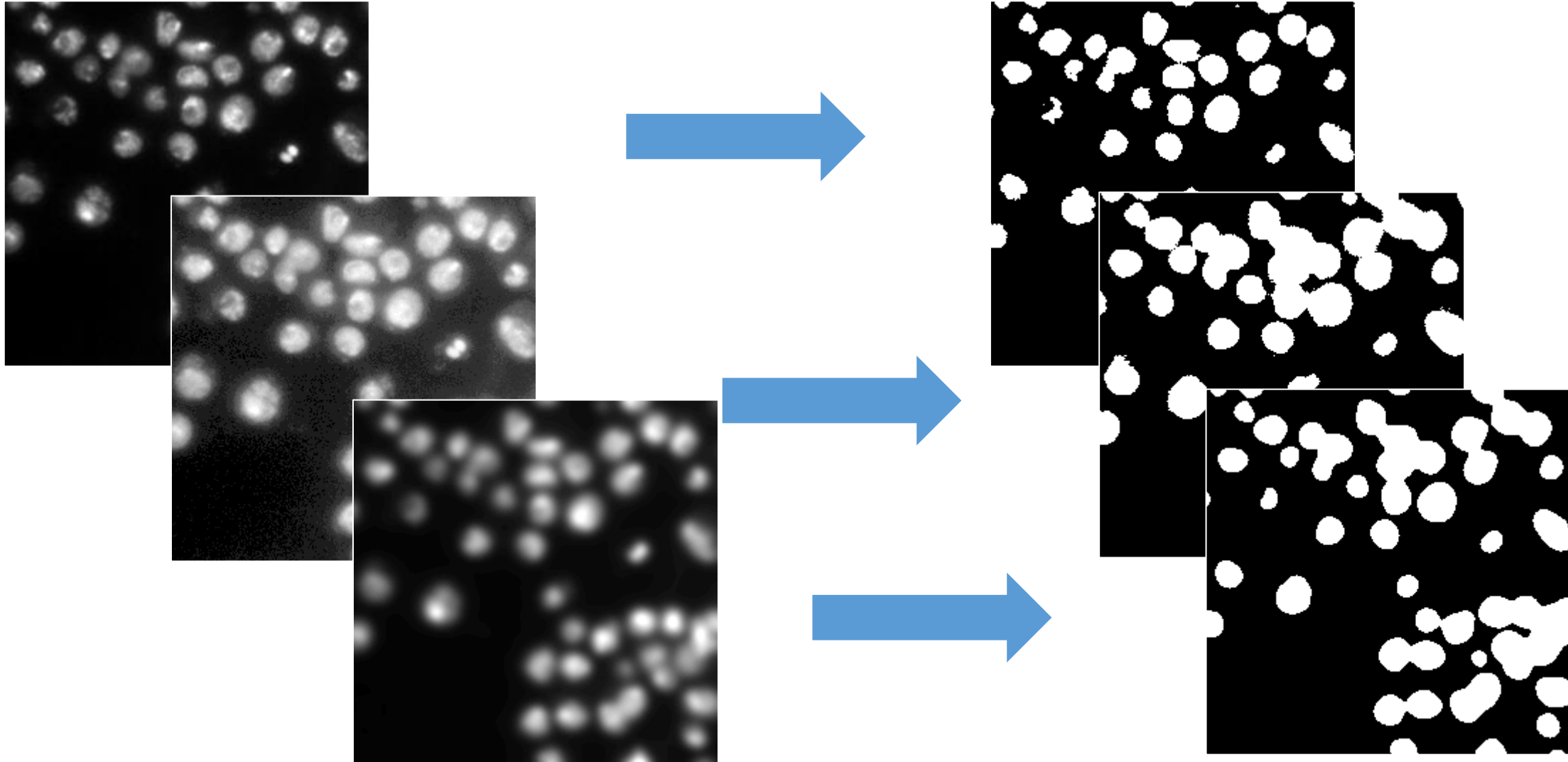
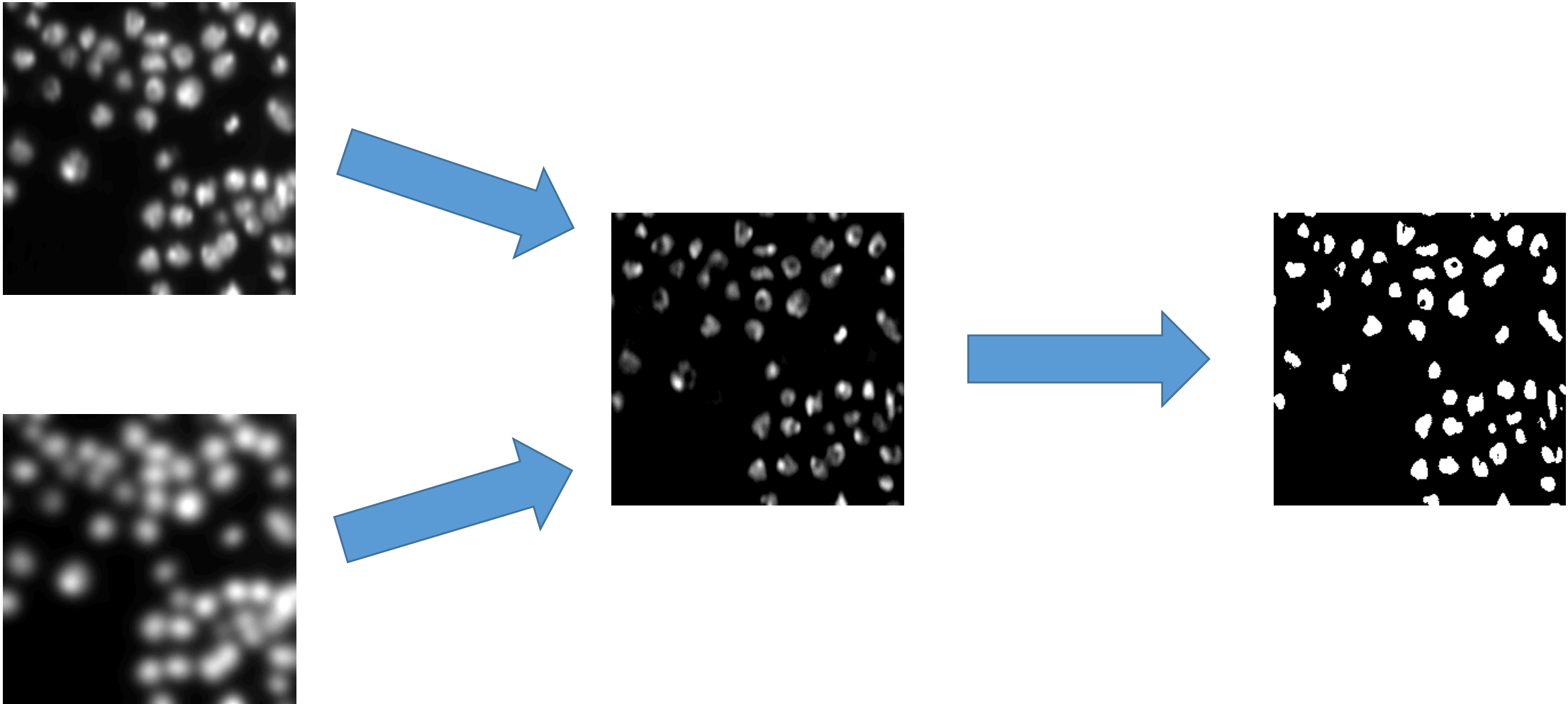


Image segmentation using thresholding

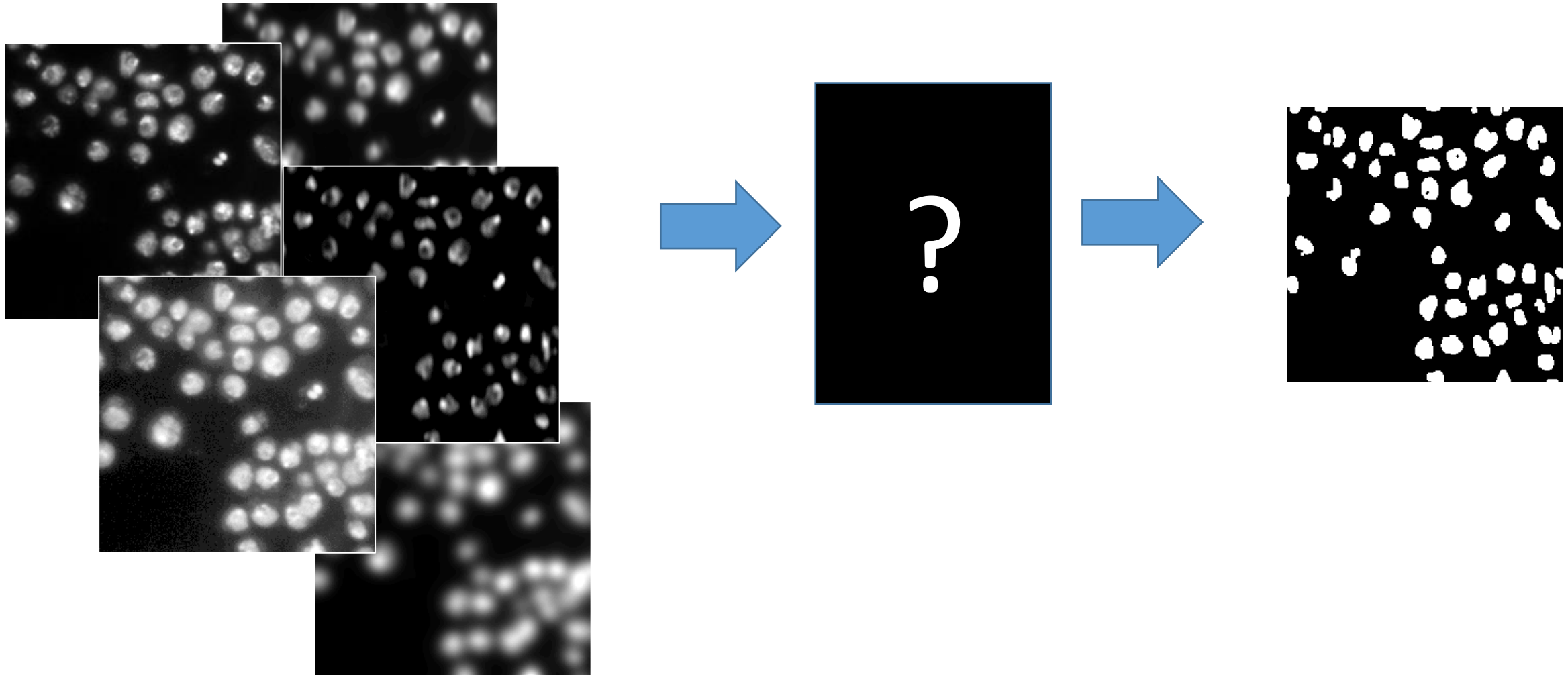
- Recap: Finding the right workflow towards a good segmentation takes time



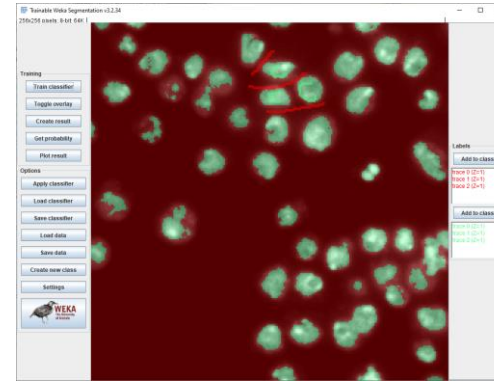
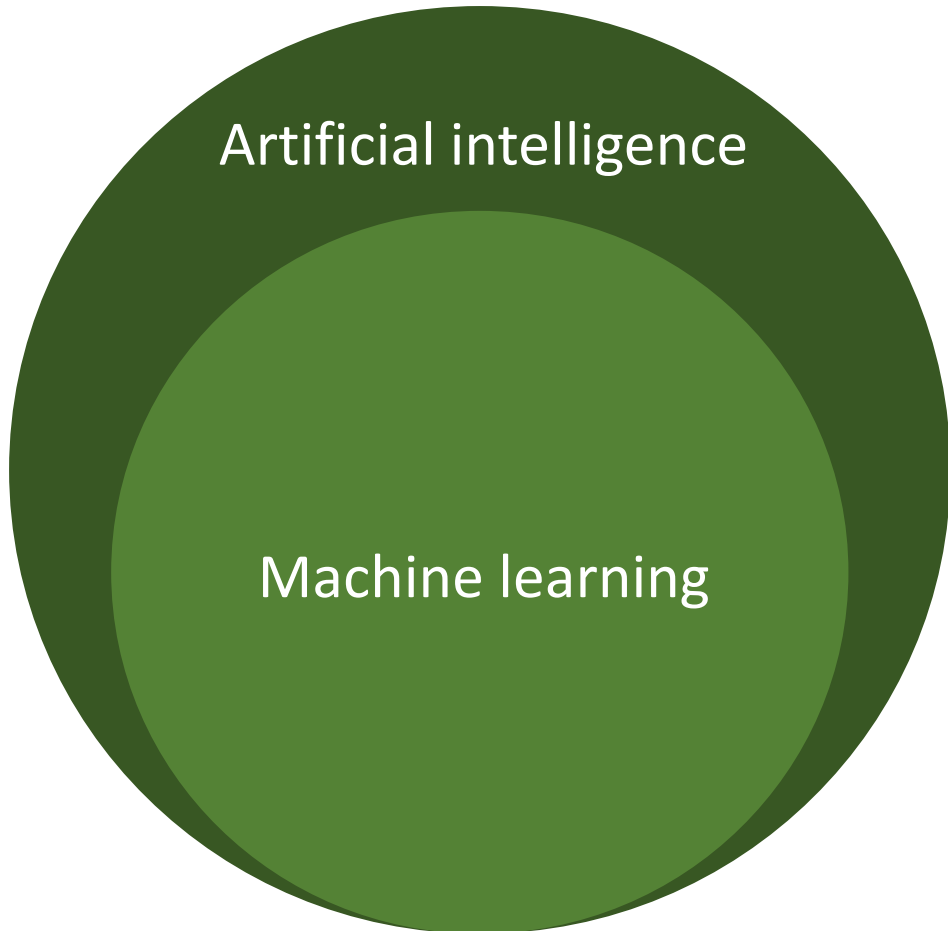
- Recap: Combining images, e.g. using Difference of Gaussian (DoG)



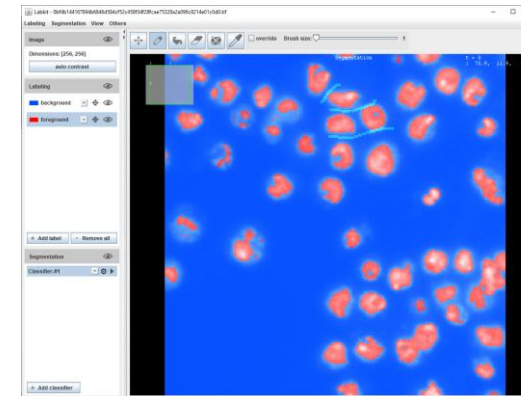
- Might there be a technology for optimization which combination of images can be used to get the best segmentation result?



- A research field in computer science
- Finds more and more applications, also in life sciences.

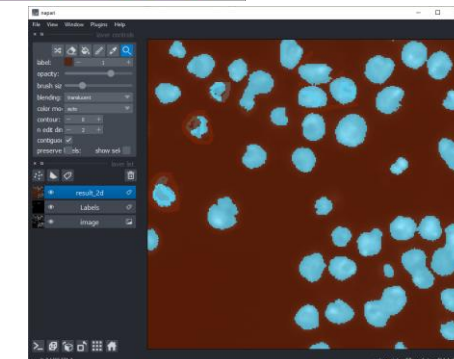


Trainable Weka Segmentation
<https://imagej.net/plugins/tws/>

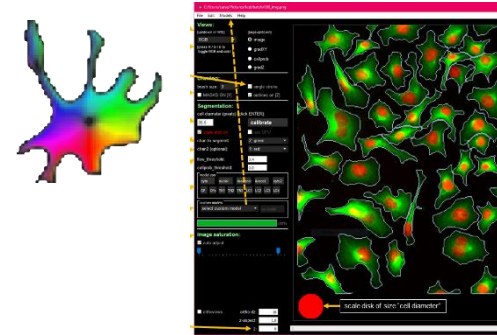
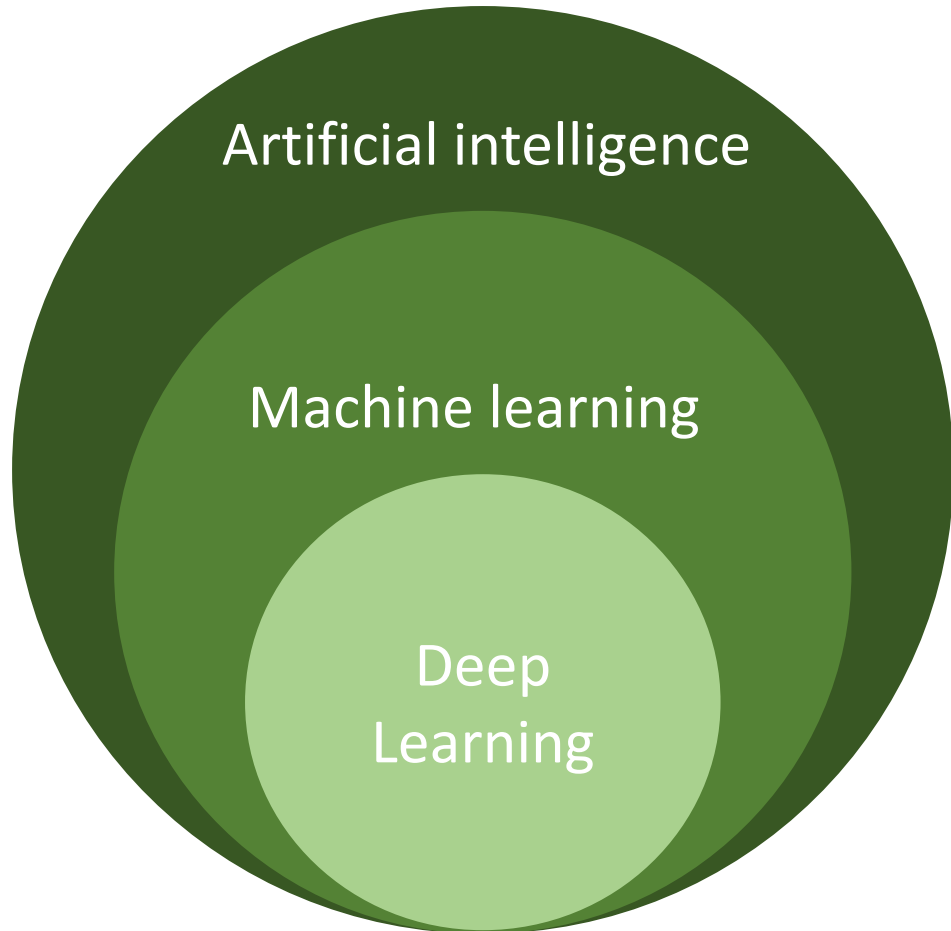


LabKit
<https://imagej.net/plugins/labkit/>

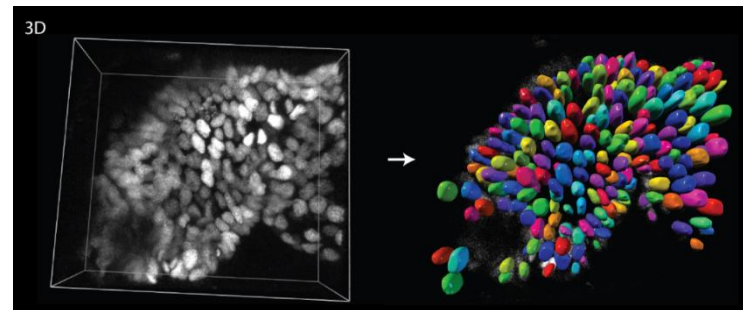
Python /
scikit-learn /
napari /
apoc



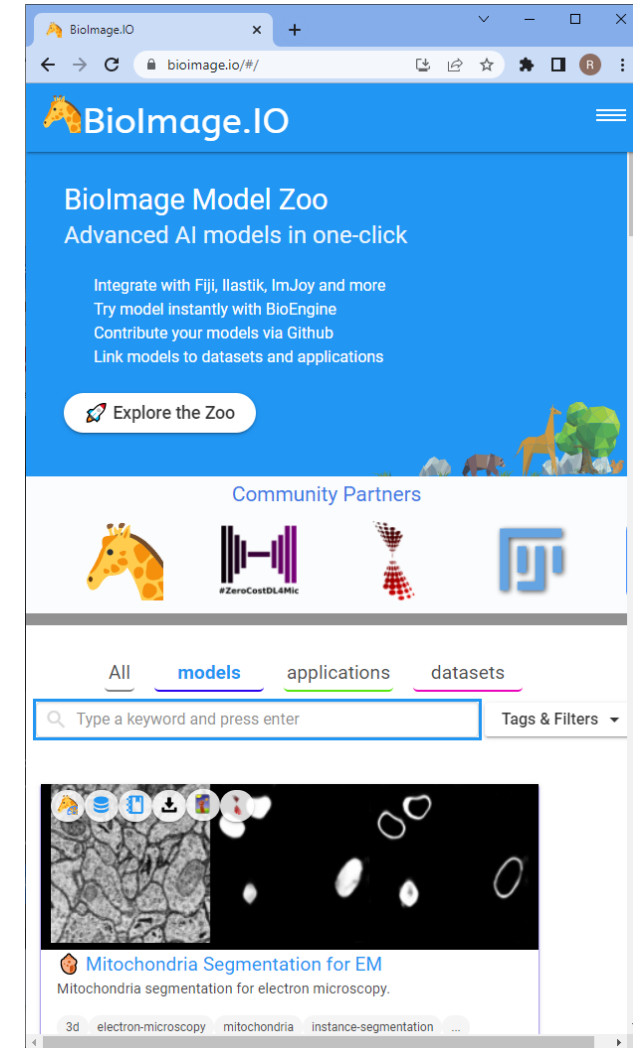
- A research field in computer science
- Finds more and more applications, also in life sciences.



www.cellpose.org/

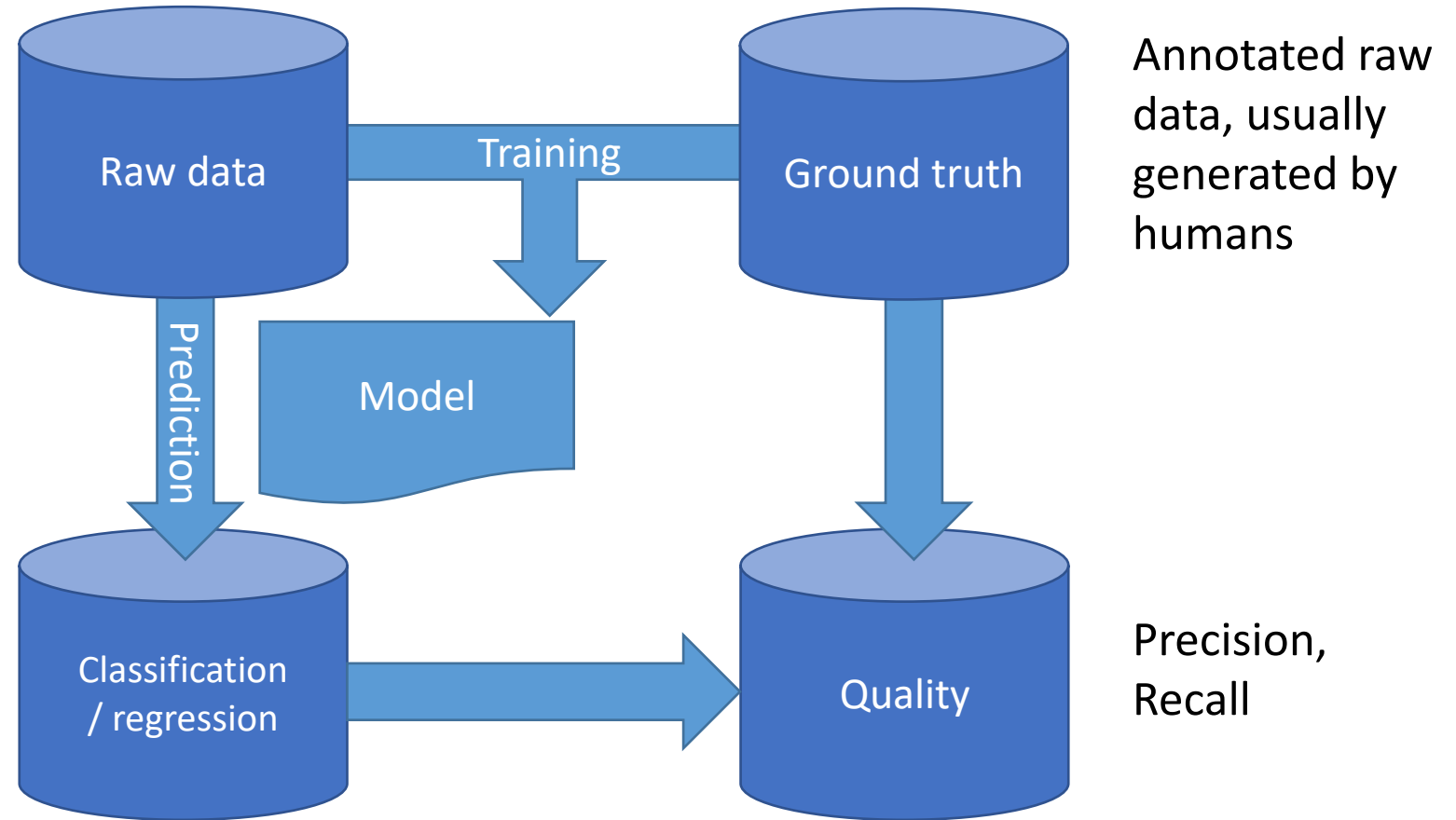
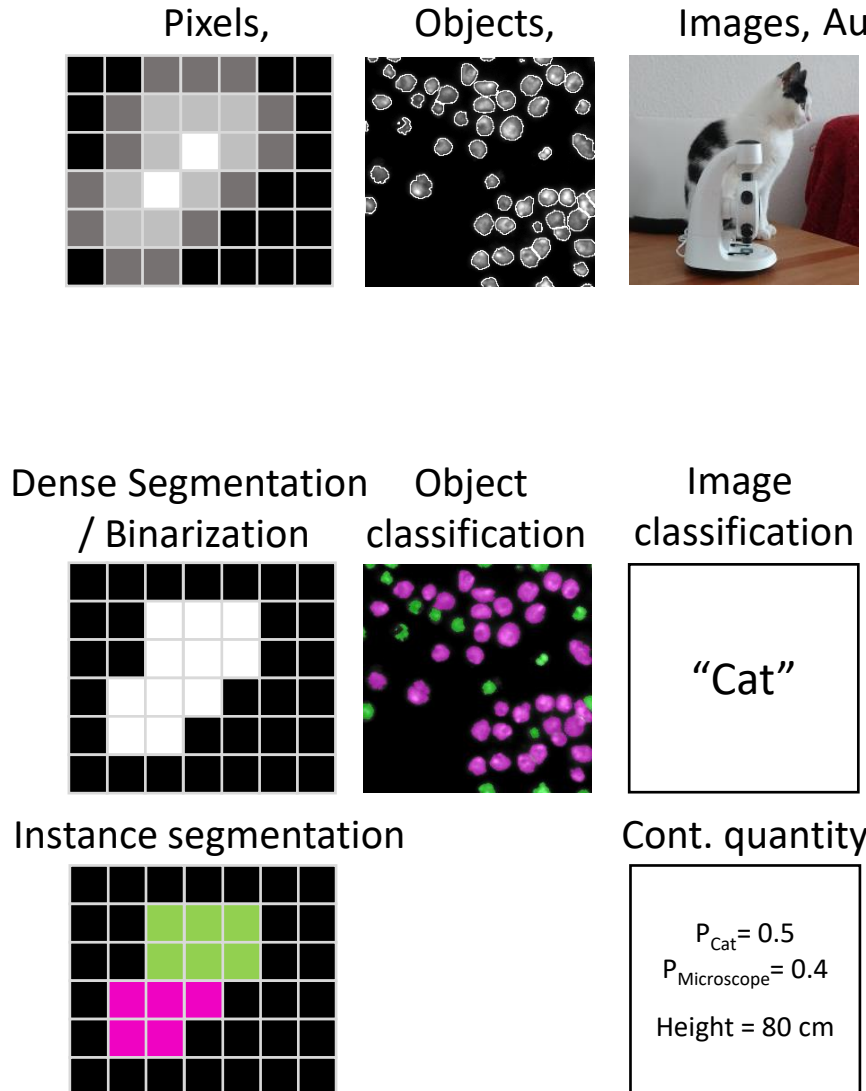


<https://github.com/stardist/stardist>

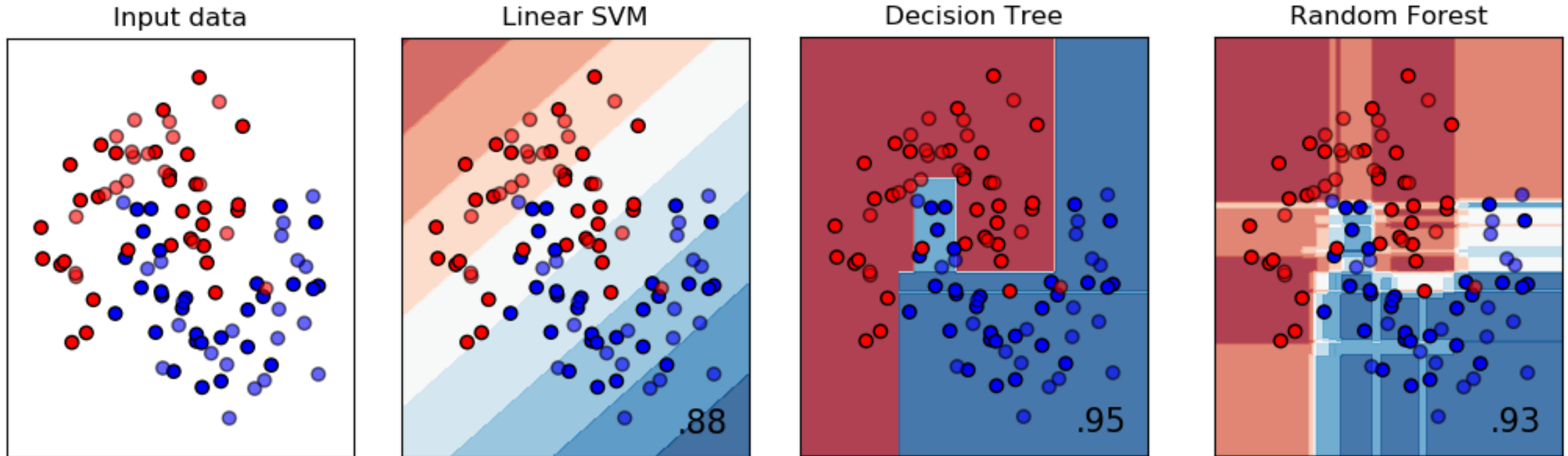


<https://bioimage.io/>

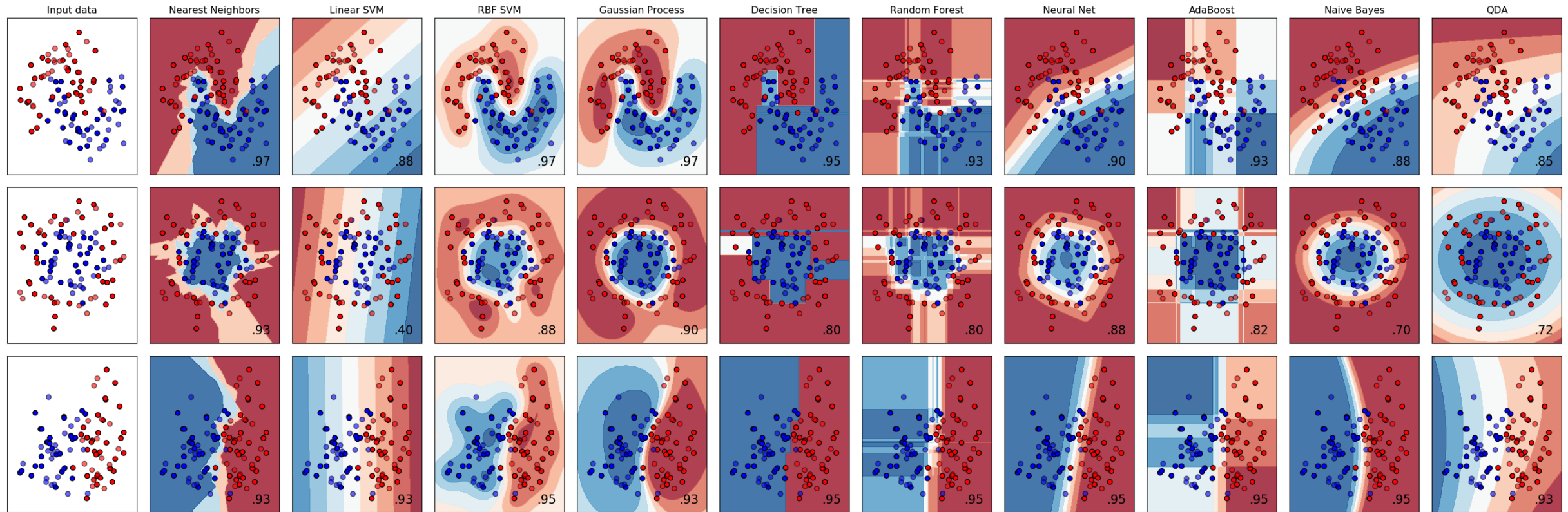
- Automatic construction of predictive models from given data



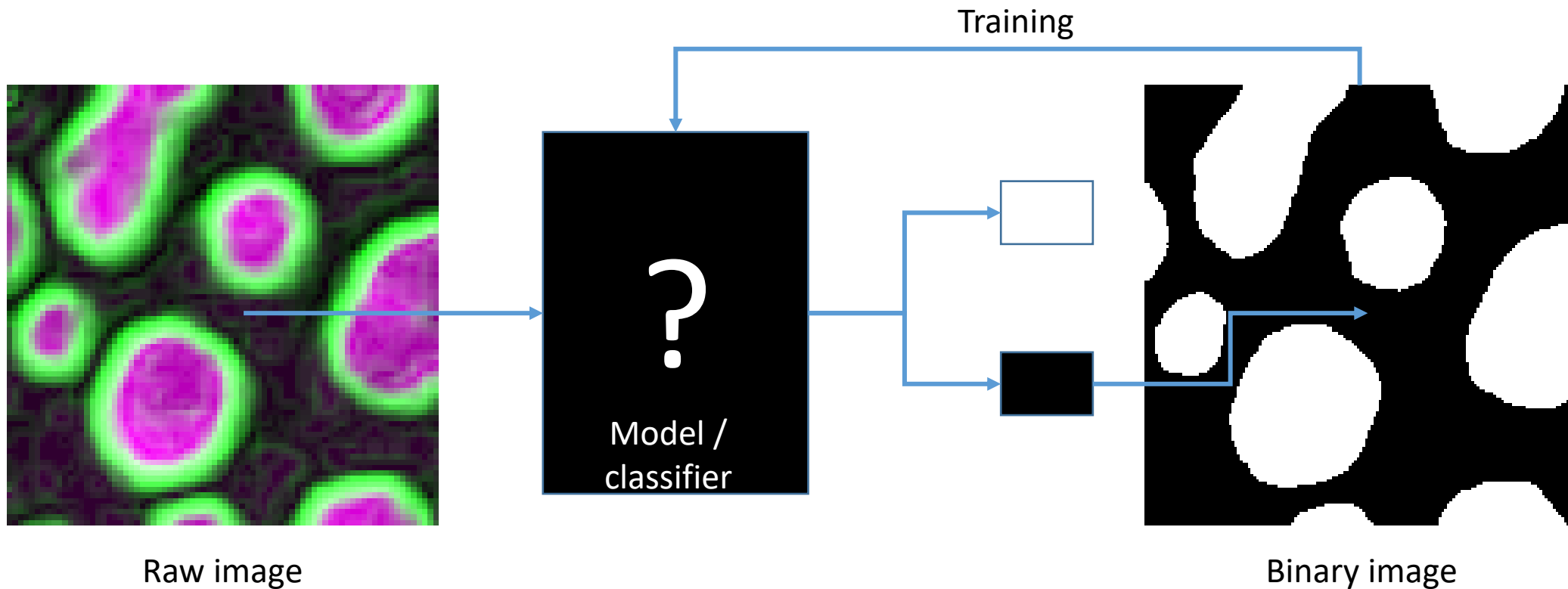
- Guess classification (color) from position of a sample in parameter space.



- The right approach depends on data, computational resources and desired quality

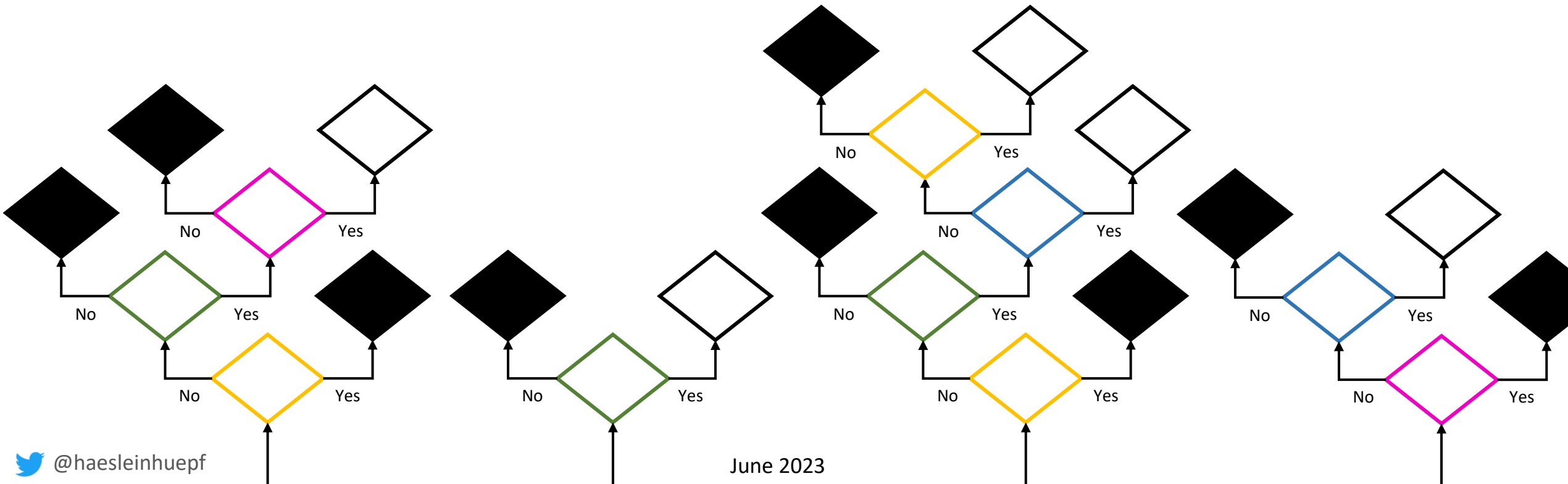


- *Supervised* machine learning: We give the computer some ground truth to learn from
- The computer derives a *model* or a *classifier* which can judge if a pixel should be foreground (white) or background (black)
- Example: Binary classifier

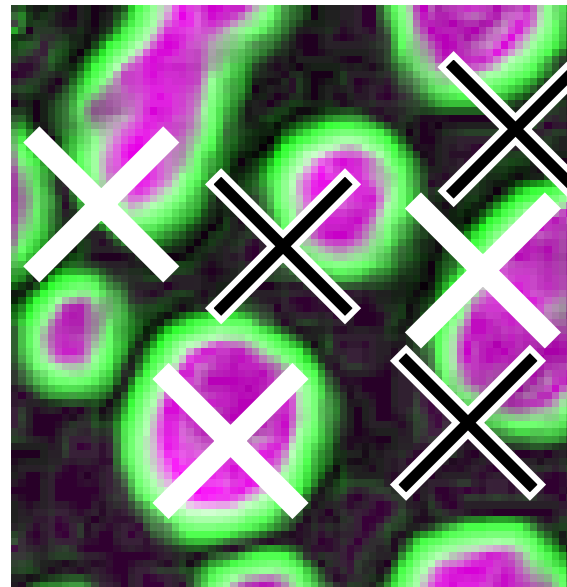
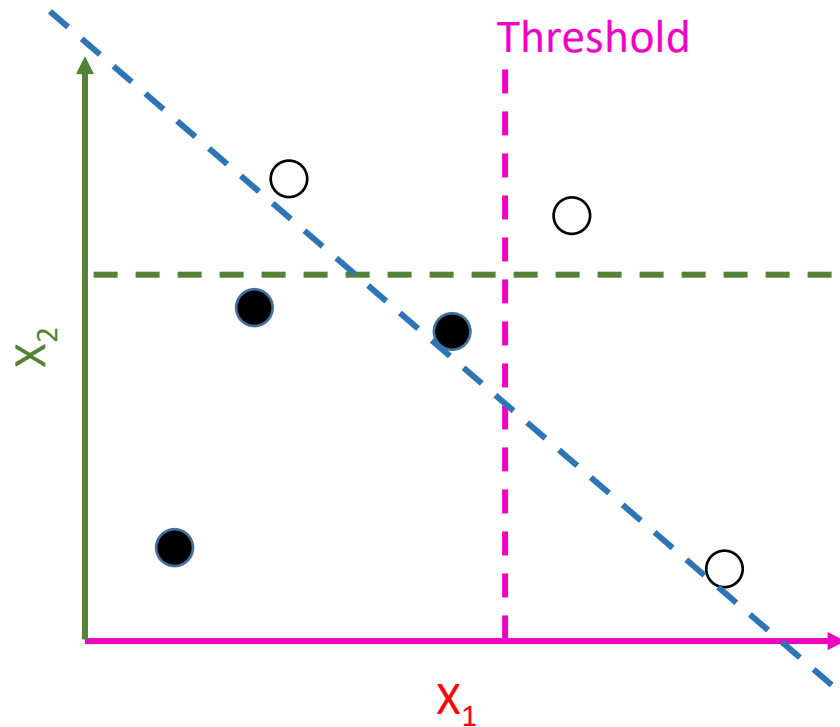


Random forest based image segmentation

- Decision trees are classifiers, they decide if a pixel should be white or black
- Random decision trees are randomly initialized, afterwards evaluated and selected
- Random forests consist of many random decision trees
- Example: Random forest of binary decision trees



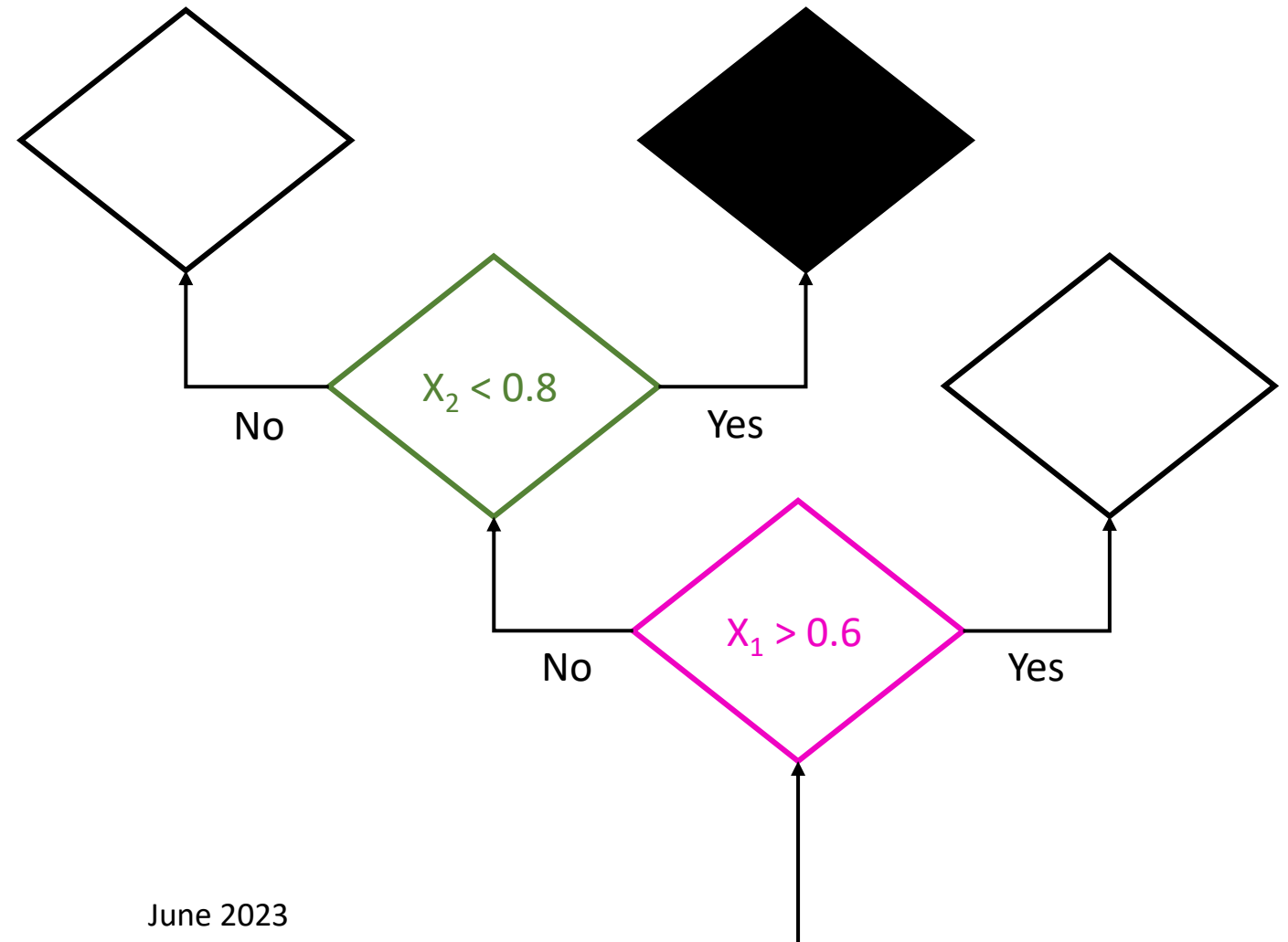
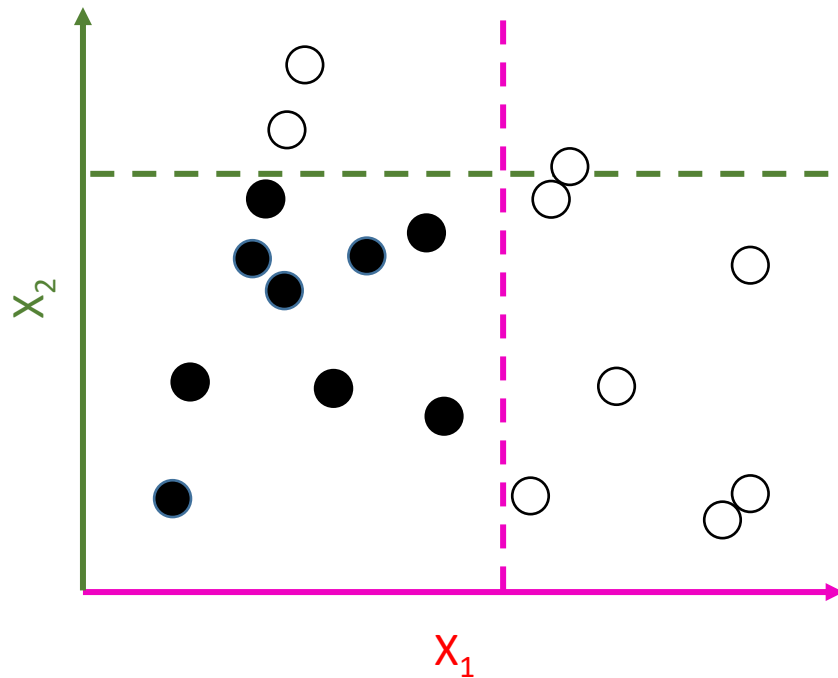
- For efficient processing, we randomly *sample* our data set
 - Individual pixels, their intensity and their classification



Note: You cannot use a single threshold to make the decision correctly

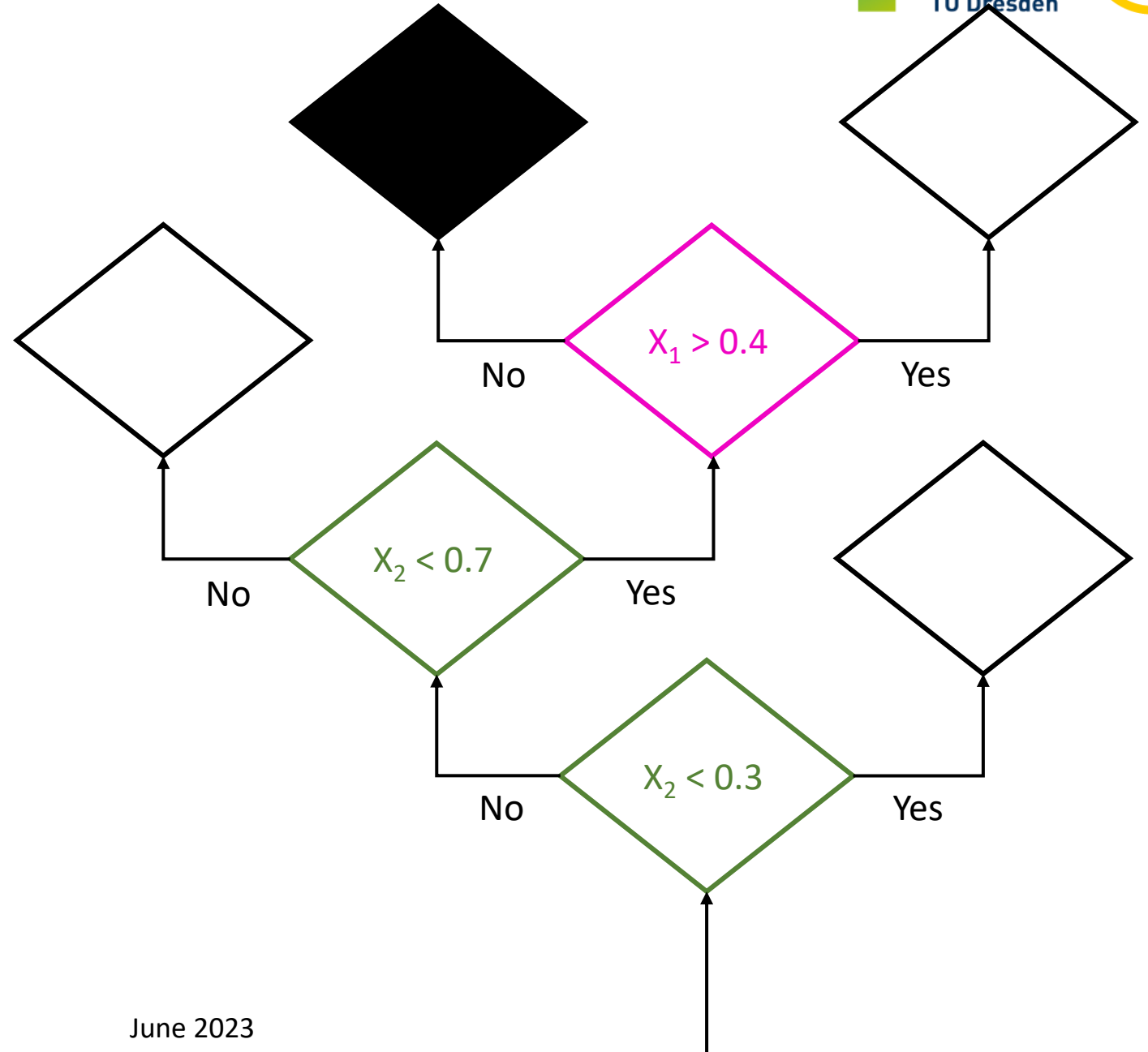
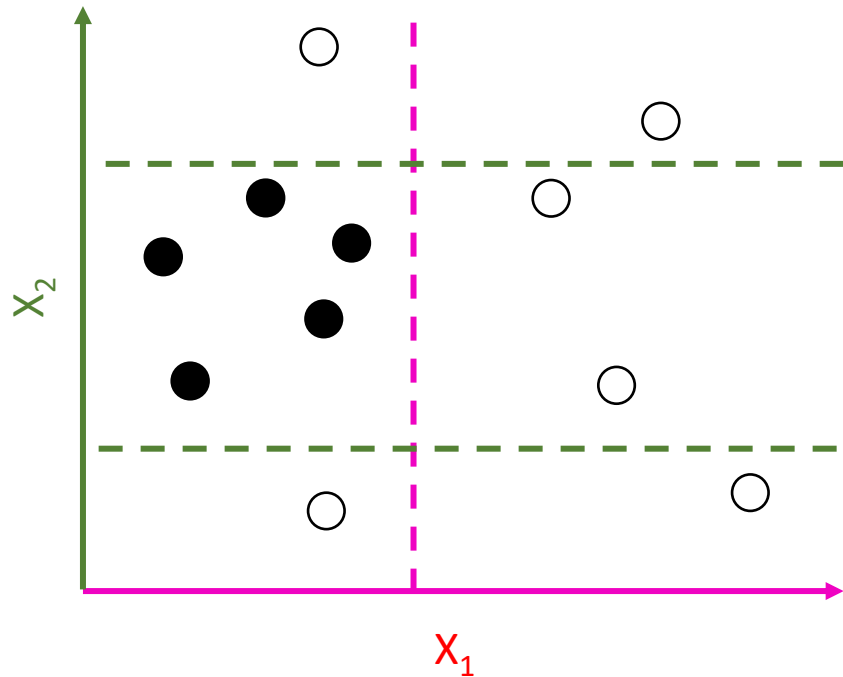
Deriving random decision trees

- Decision trees combine several thresholds on several parameters



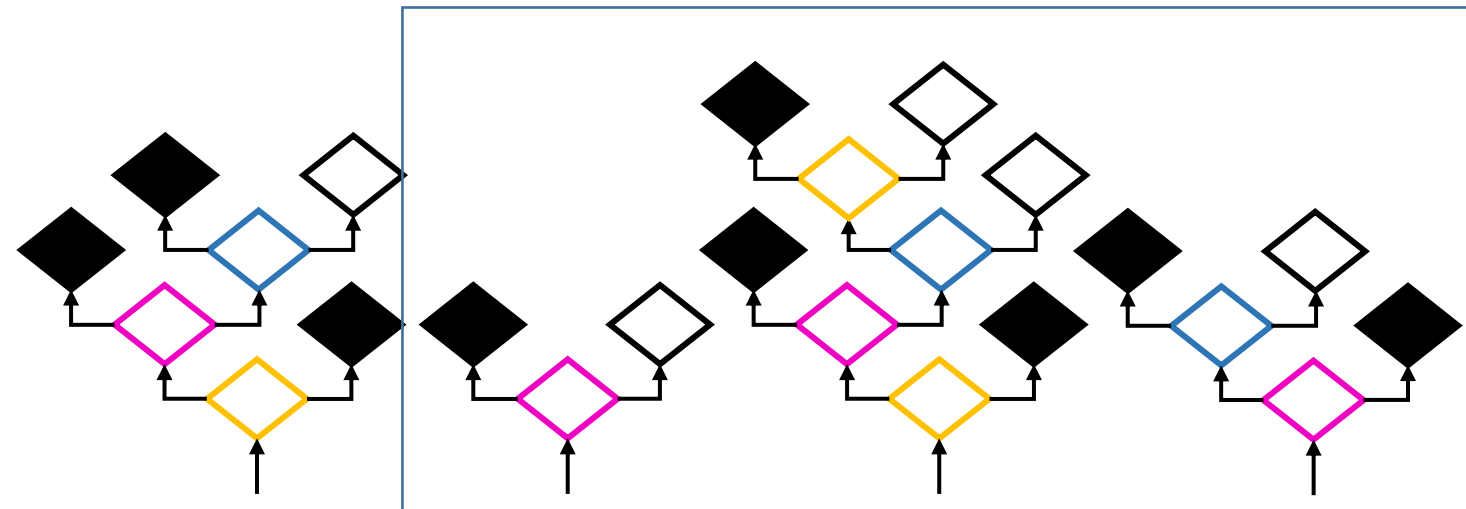
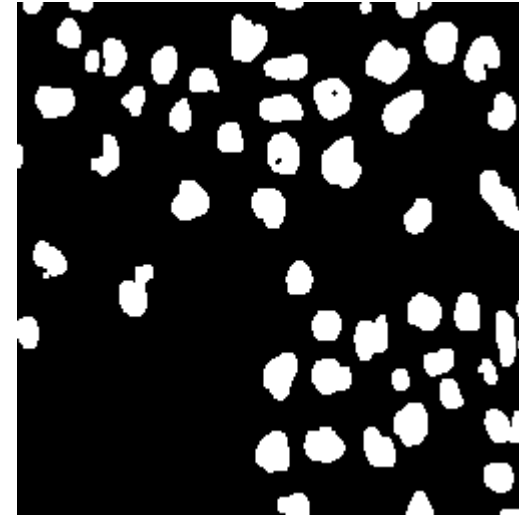
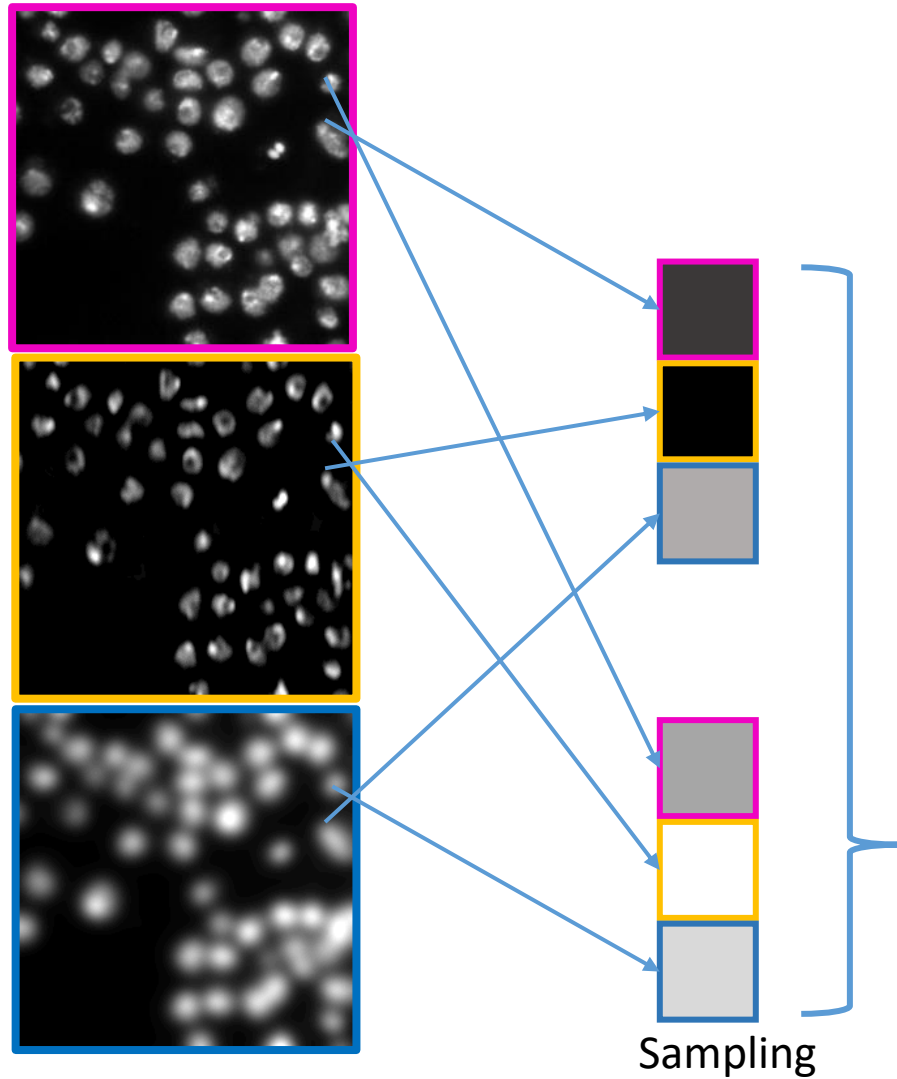
Deriving random decision trees

- Depending on sampling, the decision trees are different



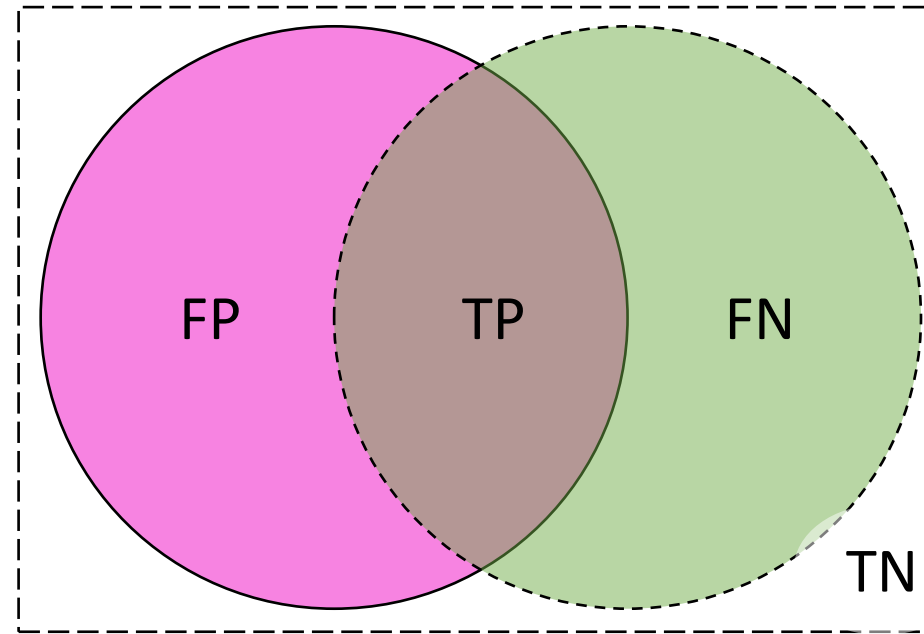
Random Forest Pixel Classifiers



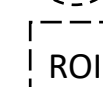



- By comparing performance of individual decision trees, good ones can be selected, bad ones excluded.



Segmentation quality estimation

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



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

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

How much do A and B overlap?

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

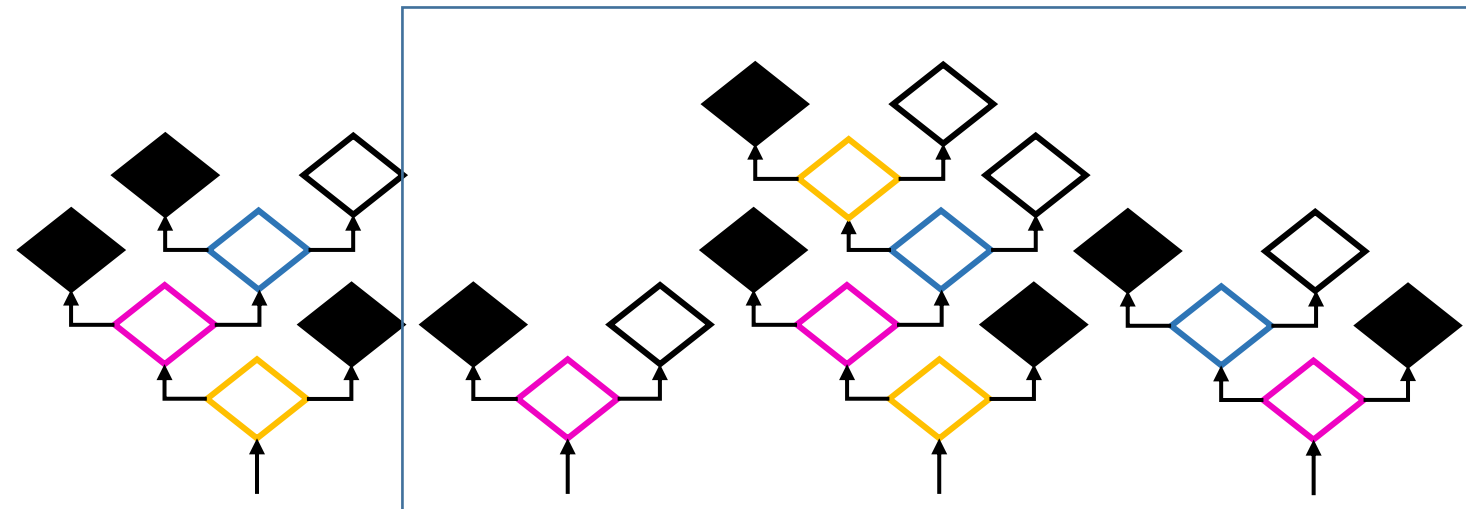
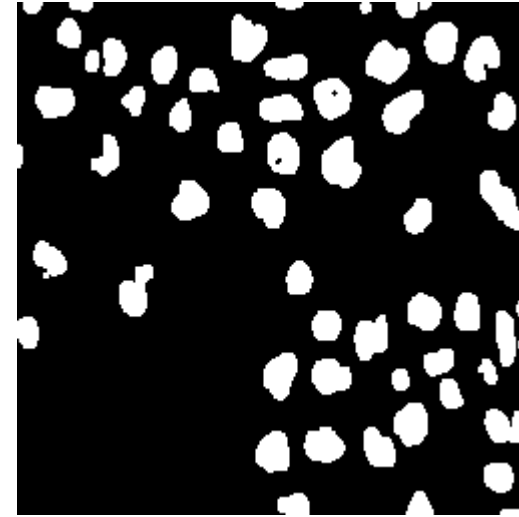
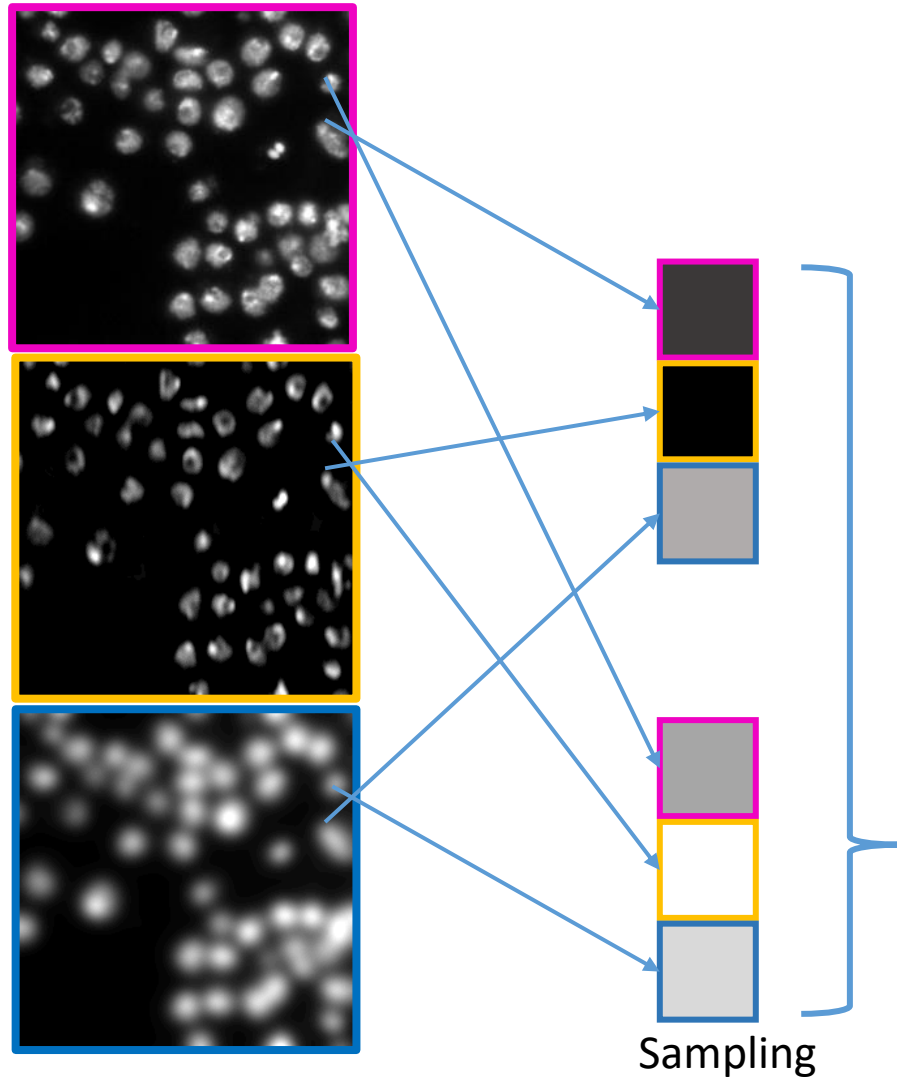
What fraction of points that were predicted as positives were really positive?

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

What fraction of positives points were predicted as positives?

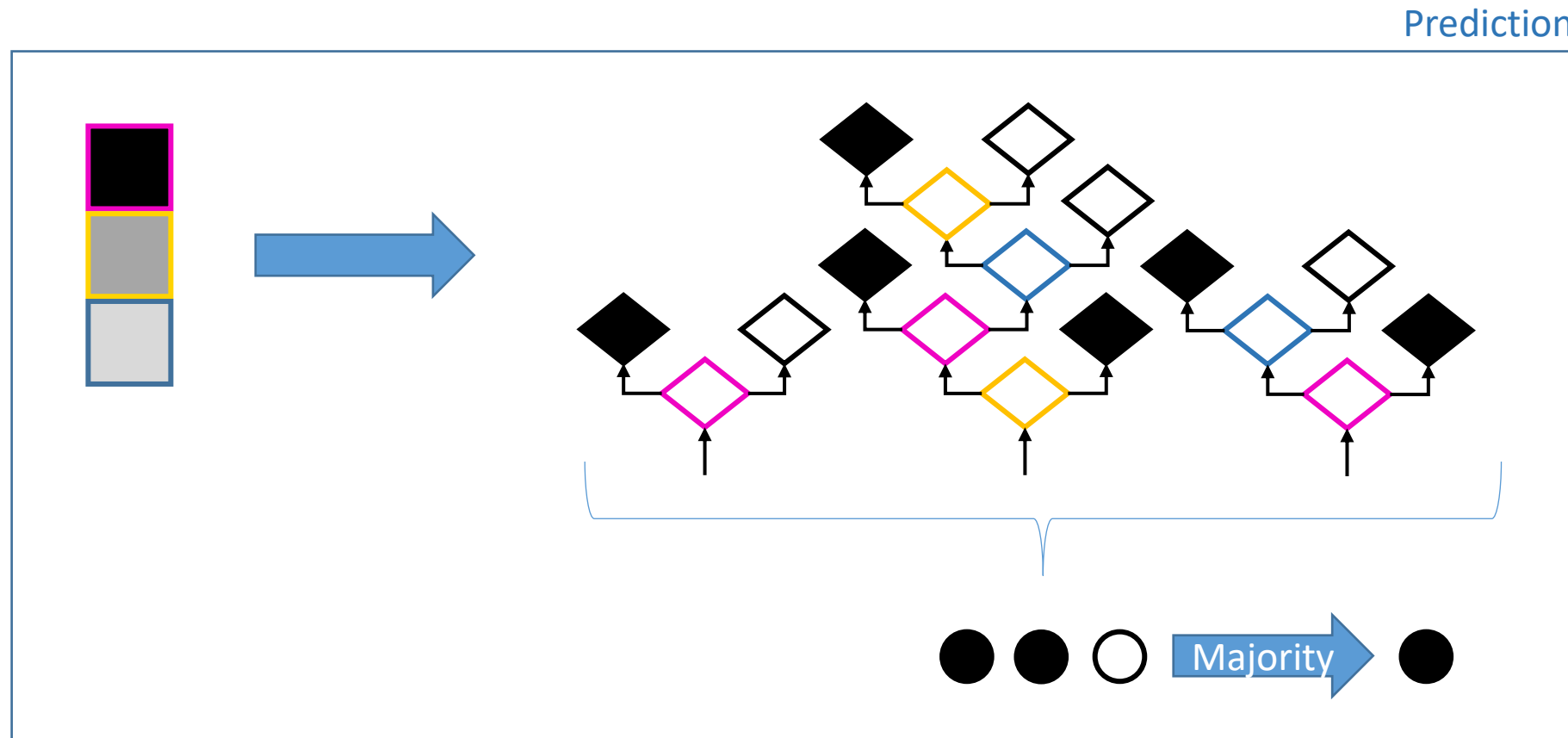
Random Forest Pixel Classifiers

- By comparing performance of individual decision trees, good ones can be selected, bad ones excluded.



Selection

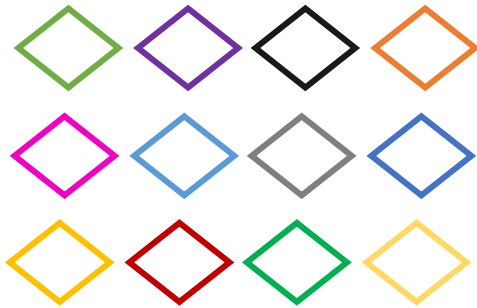
- Combination of individual tree decisions by voting or max / mean



Random Forest Pixel Classifiers

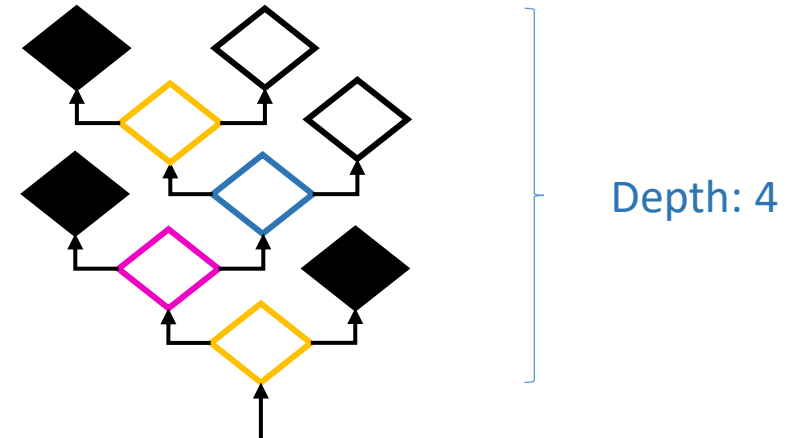
- Typical numbers for pixel classifiers in microscopy

Available features: > 20

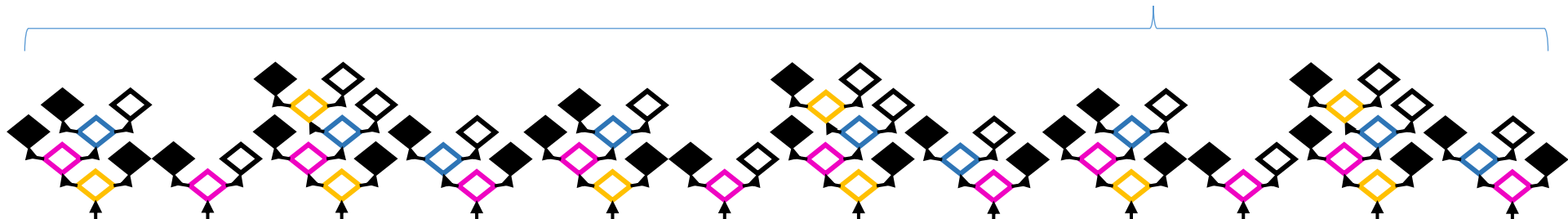


- Gaussian blur image
- DoG image
- LoG image
- Hessian
-

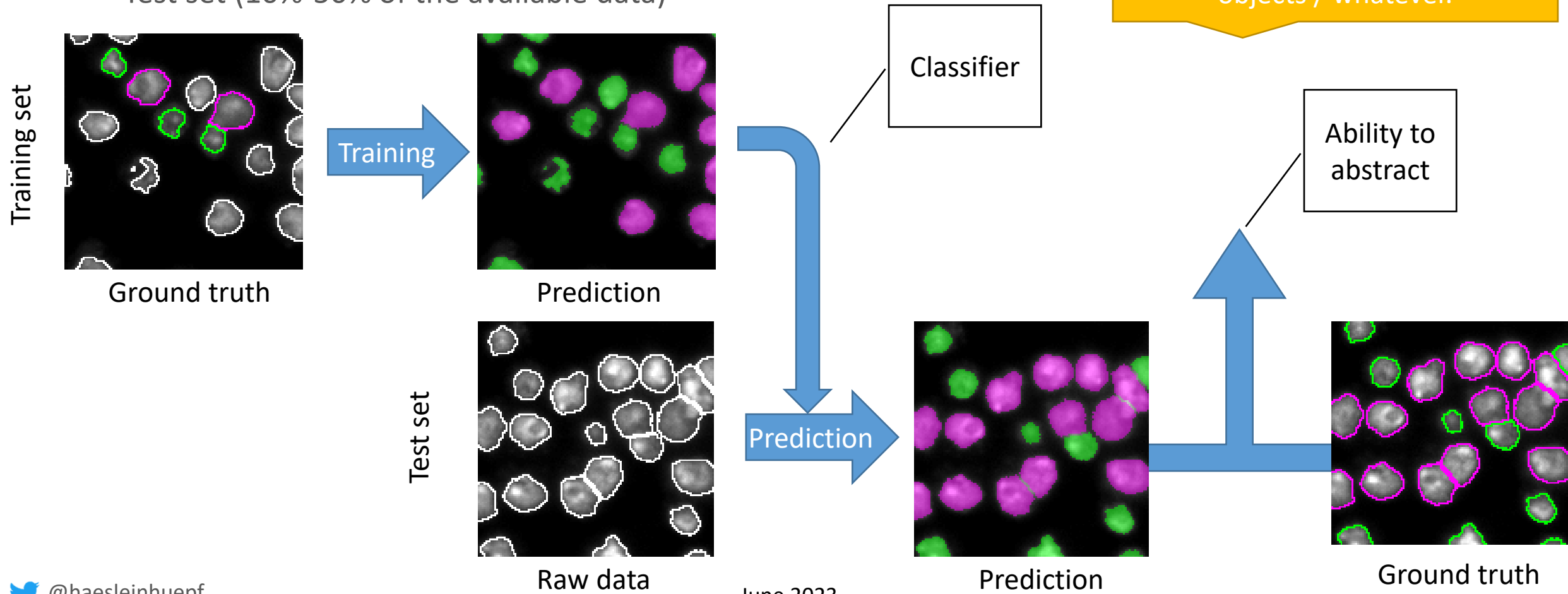
Selected features: \leq depth



Number of trees: > 100



- A good classifier is trained on a hand full of datasets and works on thousands similarly well.
- In order to assess that, we split the ground truth into two set
 - Training set (50%-90% of the available data)
 - Test set (10%-50% of the available data)



Train dataset (e.g. 80% of the data)

- Used for training directly

Validation dataset (10% of the data)

- After every iteration see if the model overfits

Test dataset (10% of the data)

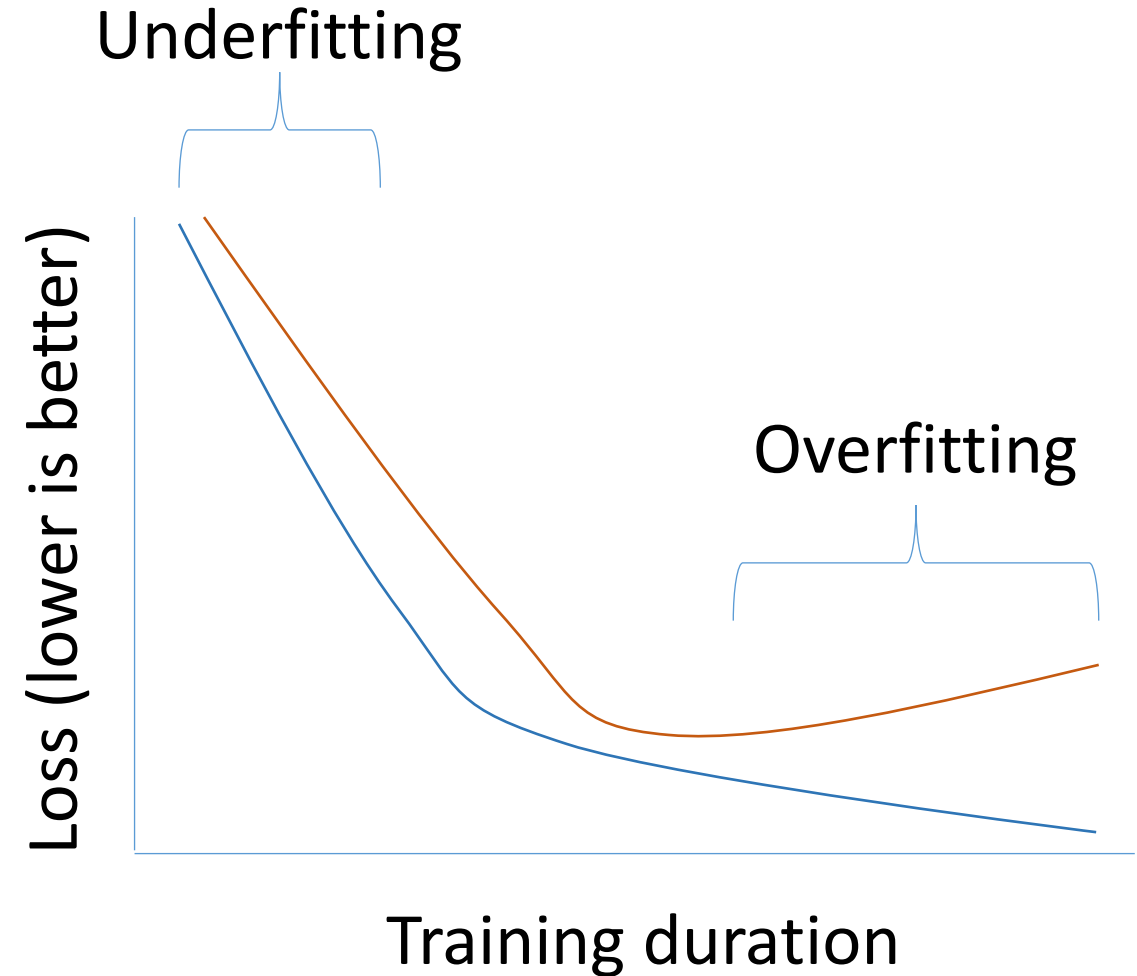
- Final evaluation after training is finished (once)

Underfitting

- A trained model that is not even able to properly process the data it was trained on

Overfitting

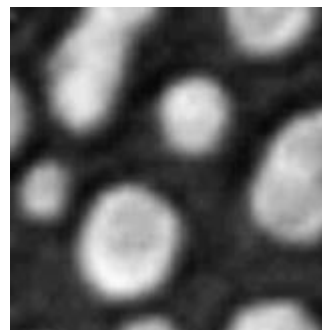
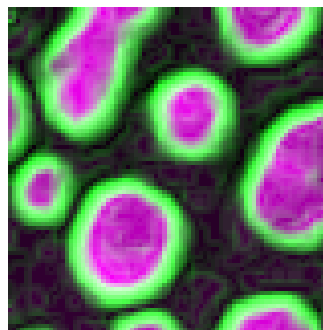
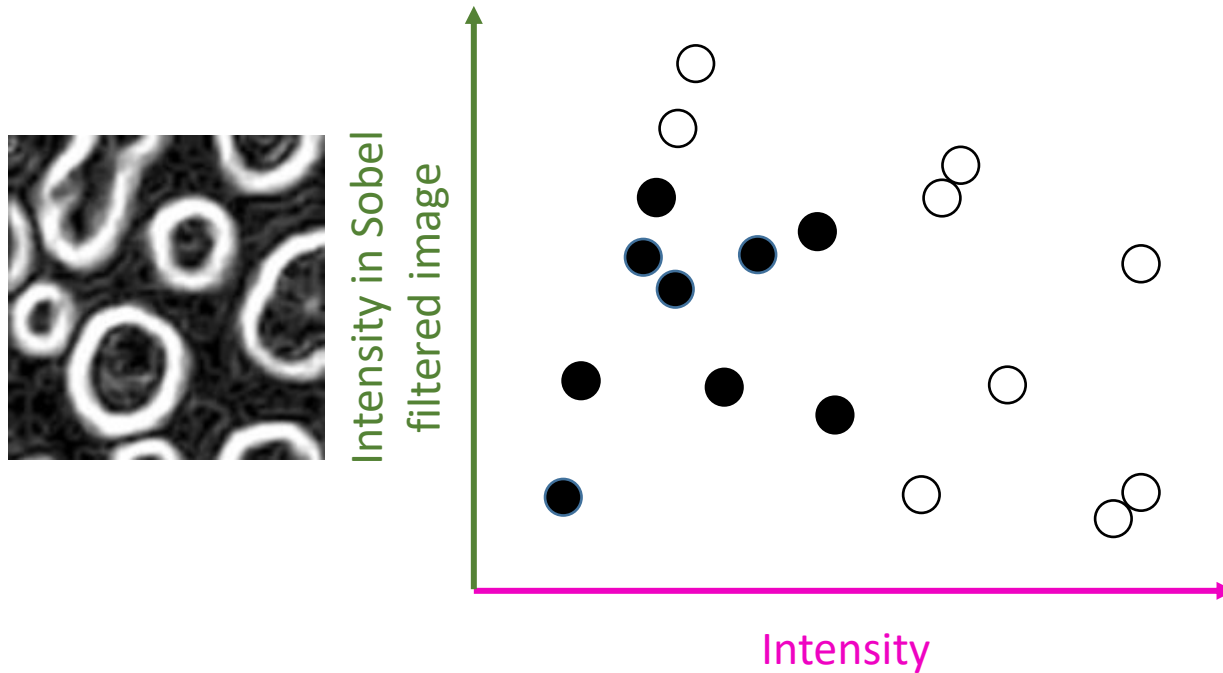
- A model that is able to process data it was trained on well
- It processes other data poorly



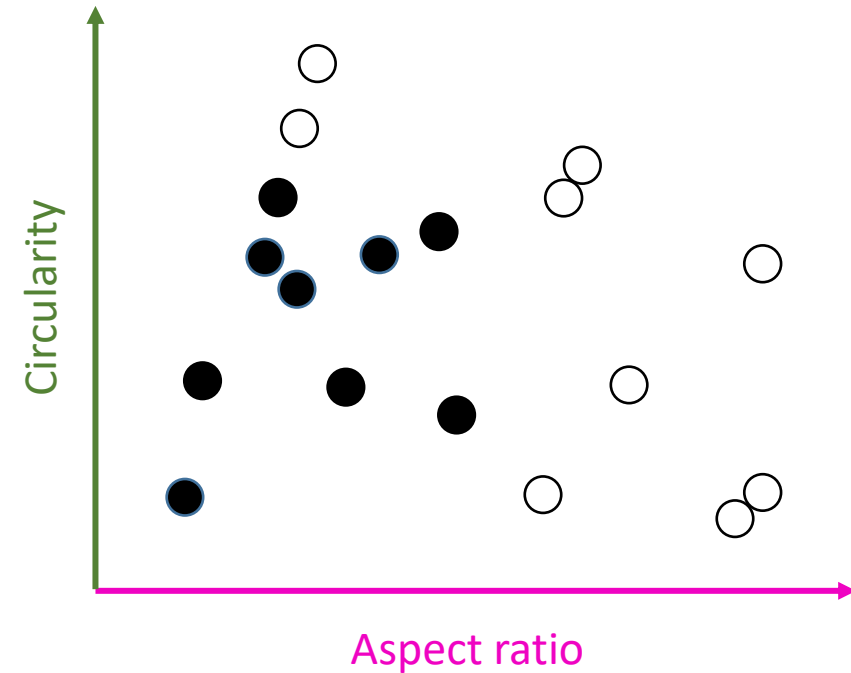
<https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c>

- What if we exchange pixel features with object features?

Pixel classification

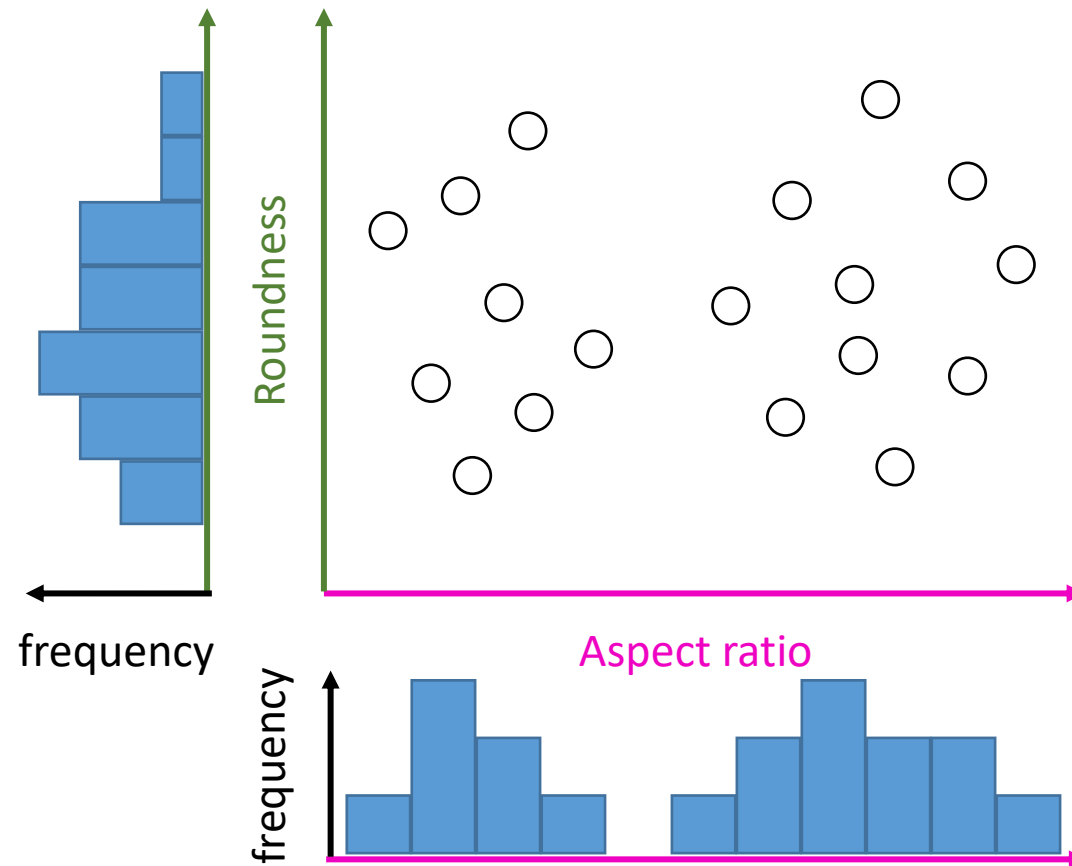


Object classification

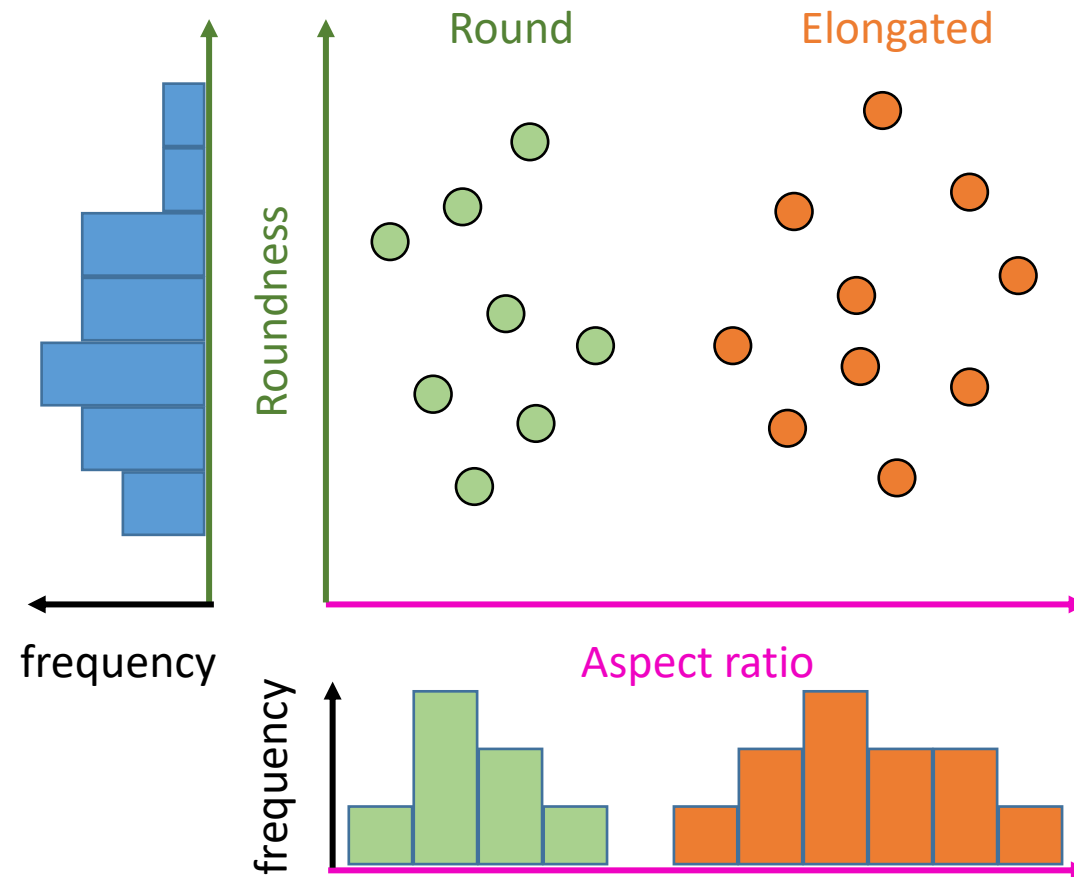


- The algorithms work the same but with different
 - Features
 - Number of features
 - Tree / forest parameters
 - Selection criteria

- If you don't provide ground truth, the algorithm is *unsupervised*.



- If you don't provide ground truth, the algorithm is unsupervised.
- Nevertheless, algorithms can tell us something about the data



Feature extraction

Robert Haase

With material from

Johannes Soltwedel, BiaPoL, PoL TU Dresden

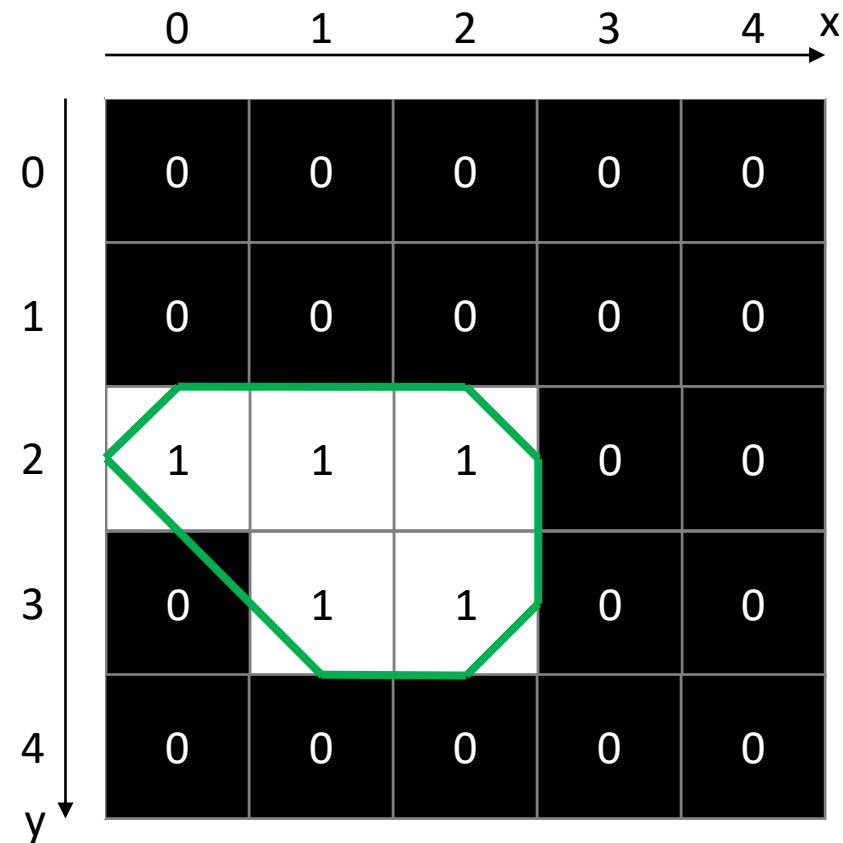
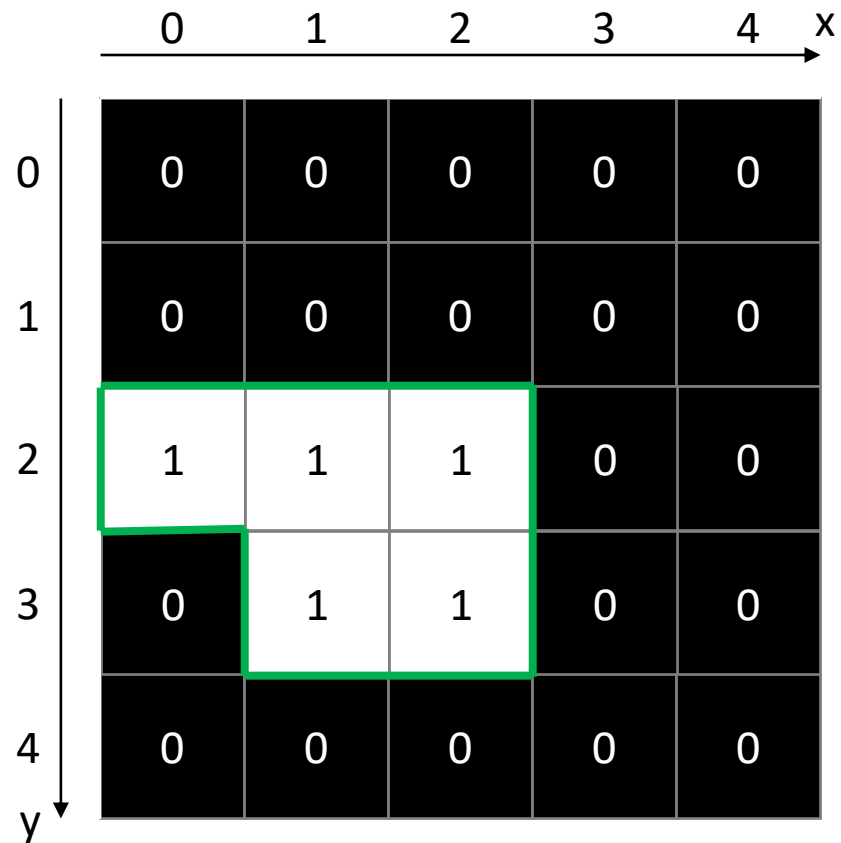
Marcelo Zoccoler, BiaPol, PoL, TU Dresden

June 2023

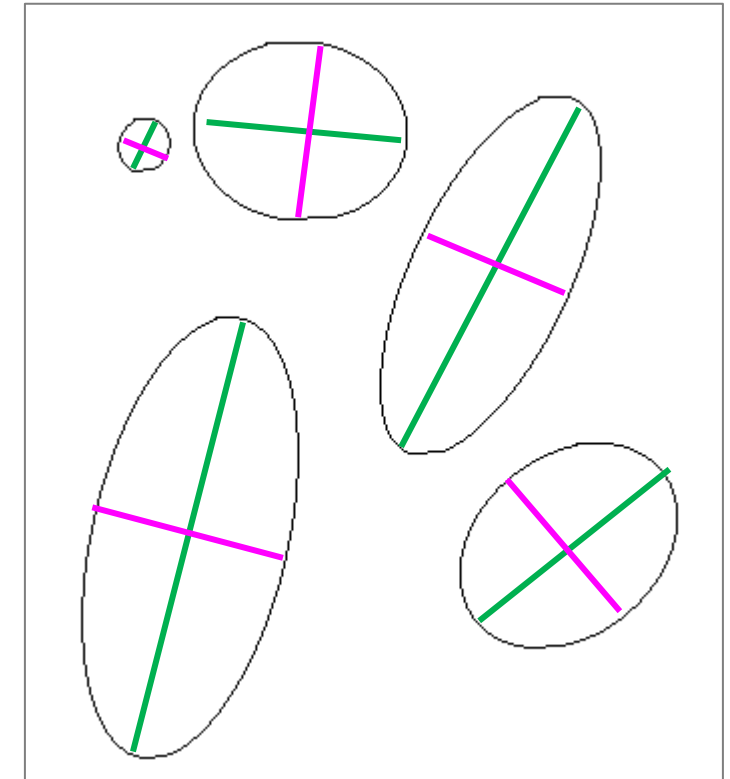
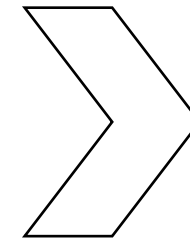
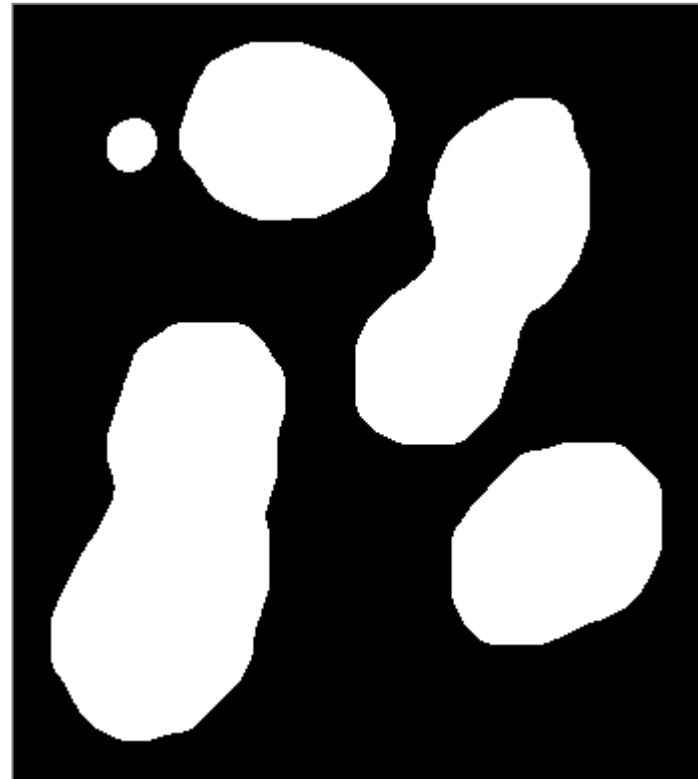
- A *feature* is a countable or measurable property of an image or object.
- Goal of feature extraction is finding a minimal set of features to describe an object well enough to differentiate it from other objects.
- **Intensity based**
 - Mean intensity
 - Standard deviation
 - Total intensity
 - Textures
- **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

Perimeter

- Length of the outline around an object
- Depends on the actual implementation

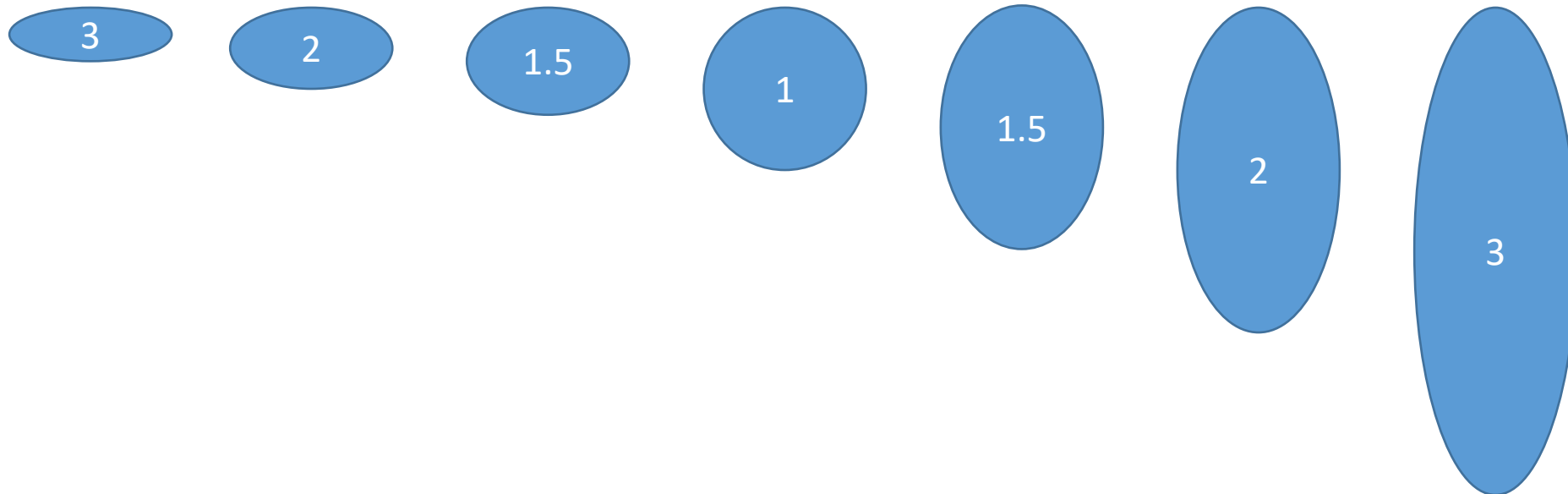


- For every object, find the optimal ellipse simplifying the object.
- Major axis ... long diameter
- Minor axis ... short diameter
- Major and minor axis are perpendicular to each other

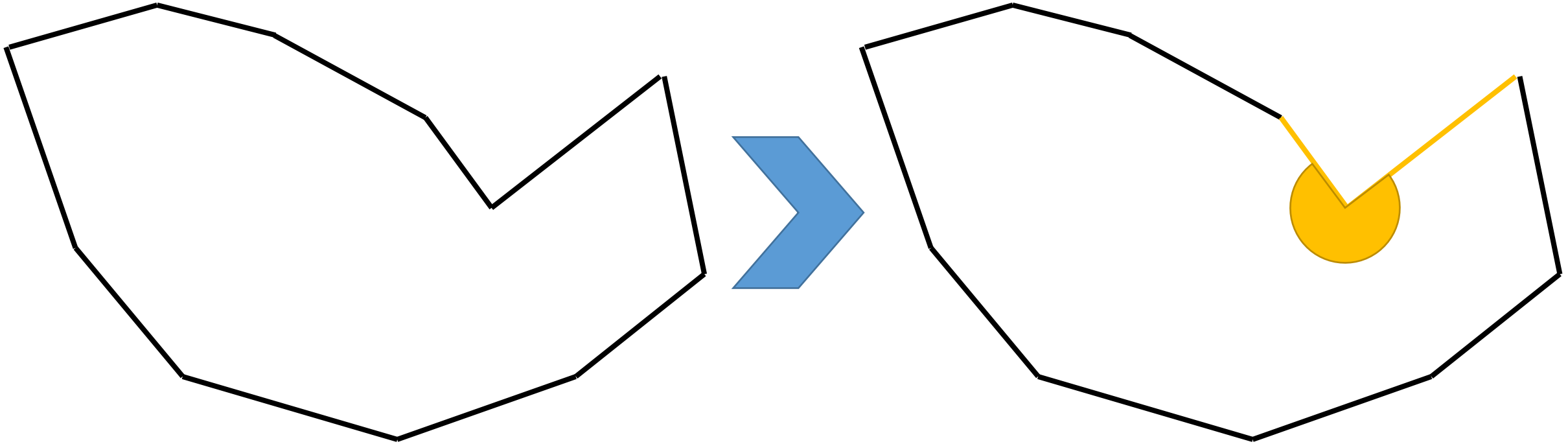


- The aspect ratio describes the elongation of an object.

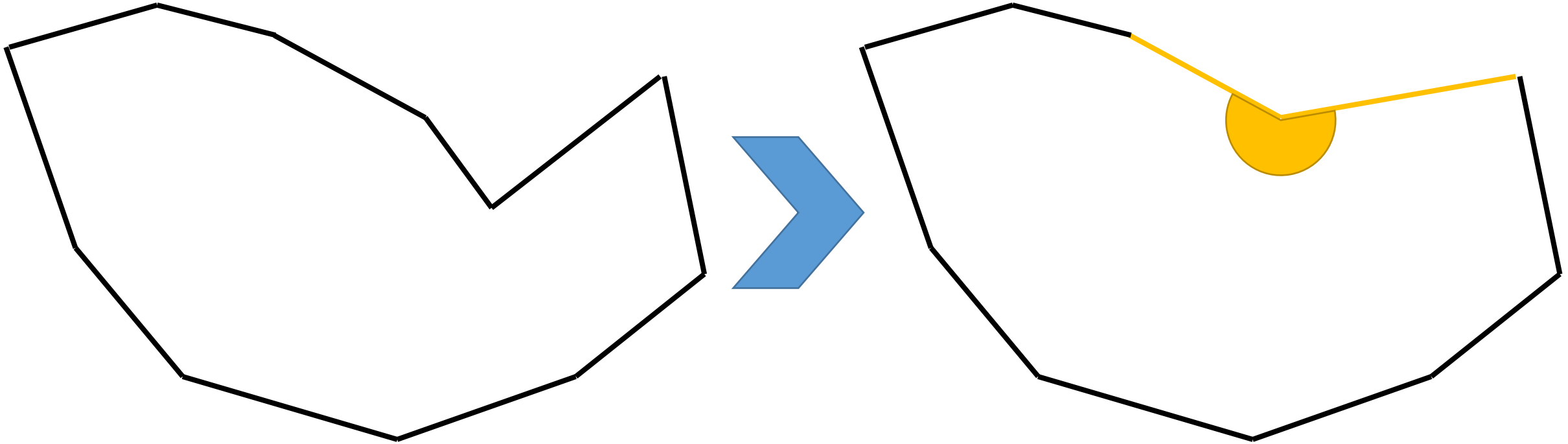
AR = major / minor



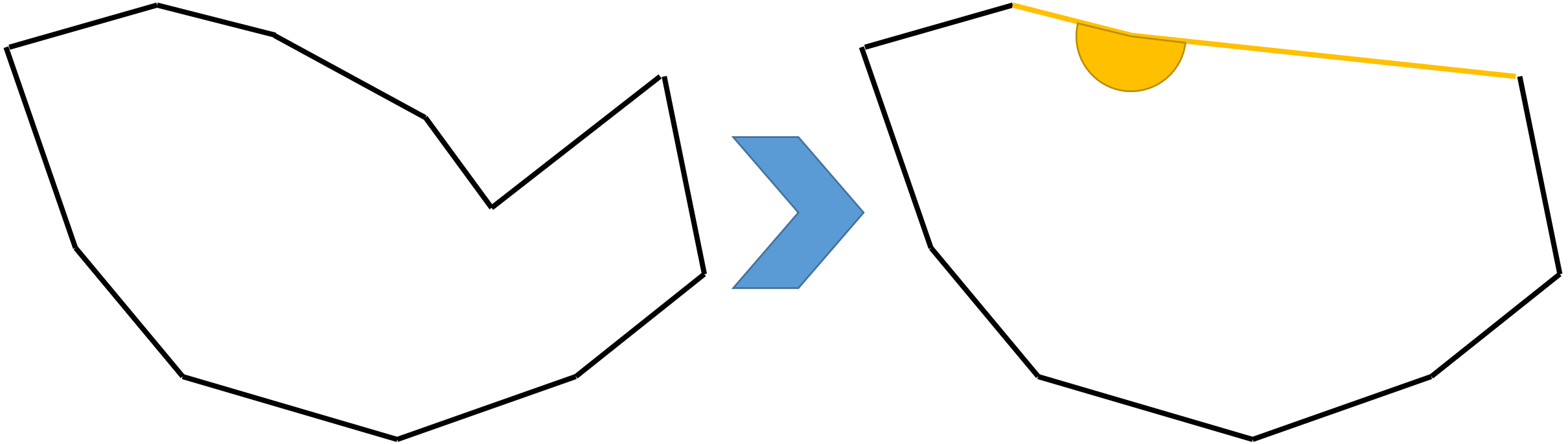
- By removing all concave corners of an object, we retrieve its **convex hull**.



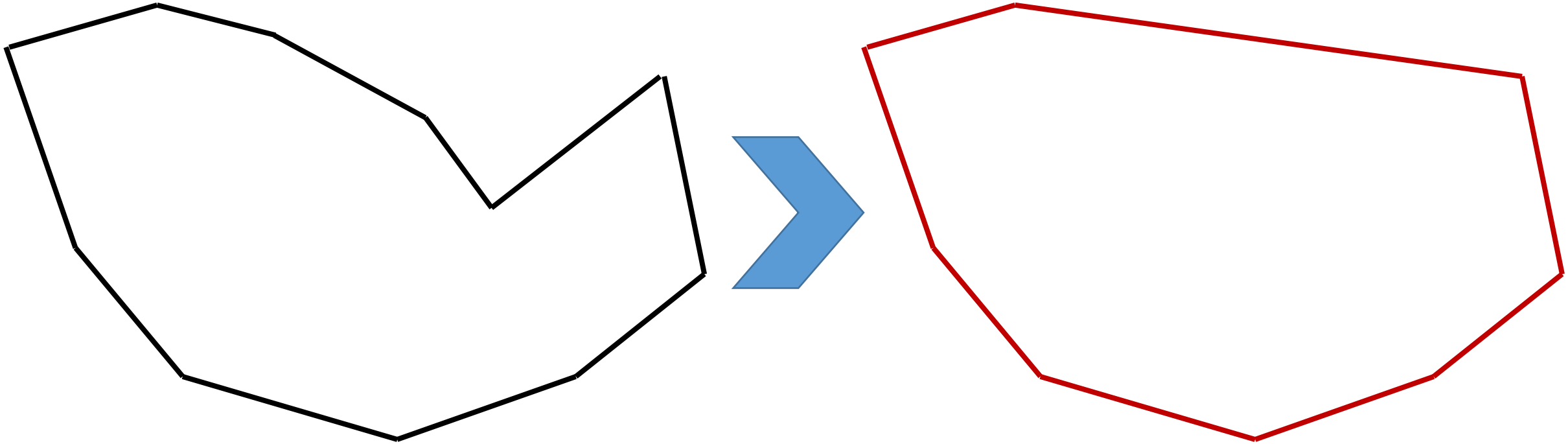
- By removing all concave corners of an object, we retrieve its **convex hull**.



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



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



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

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

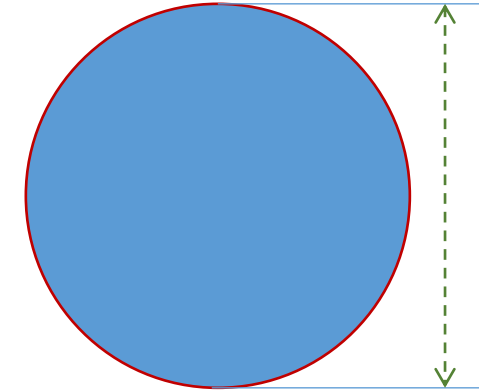
d

Circumference

$C = \pi d$

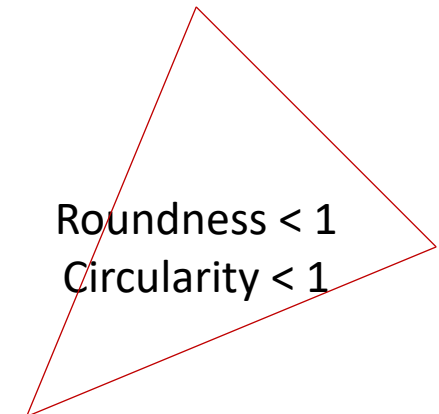
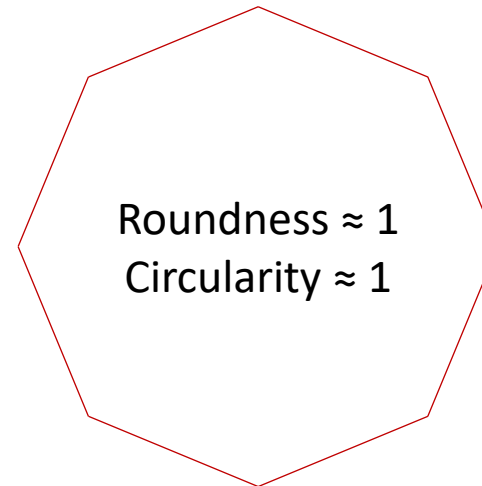
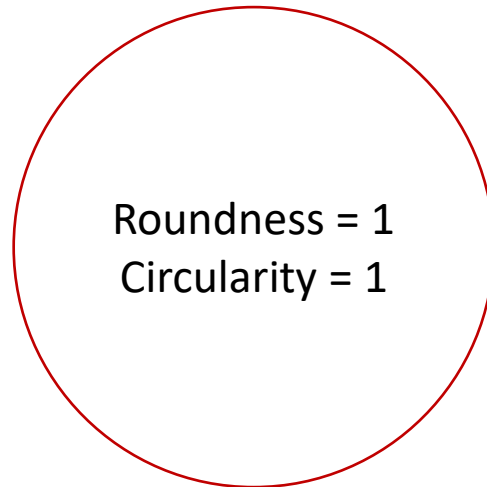
Area

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



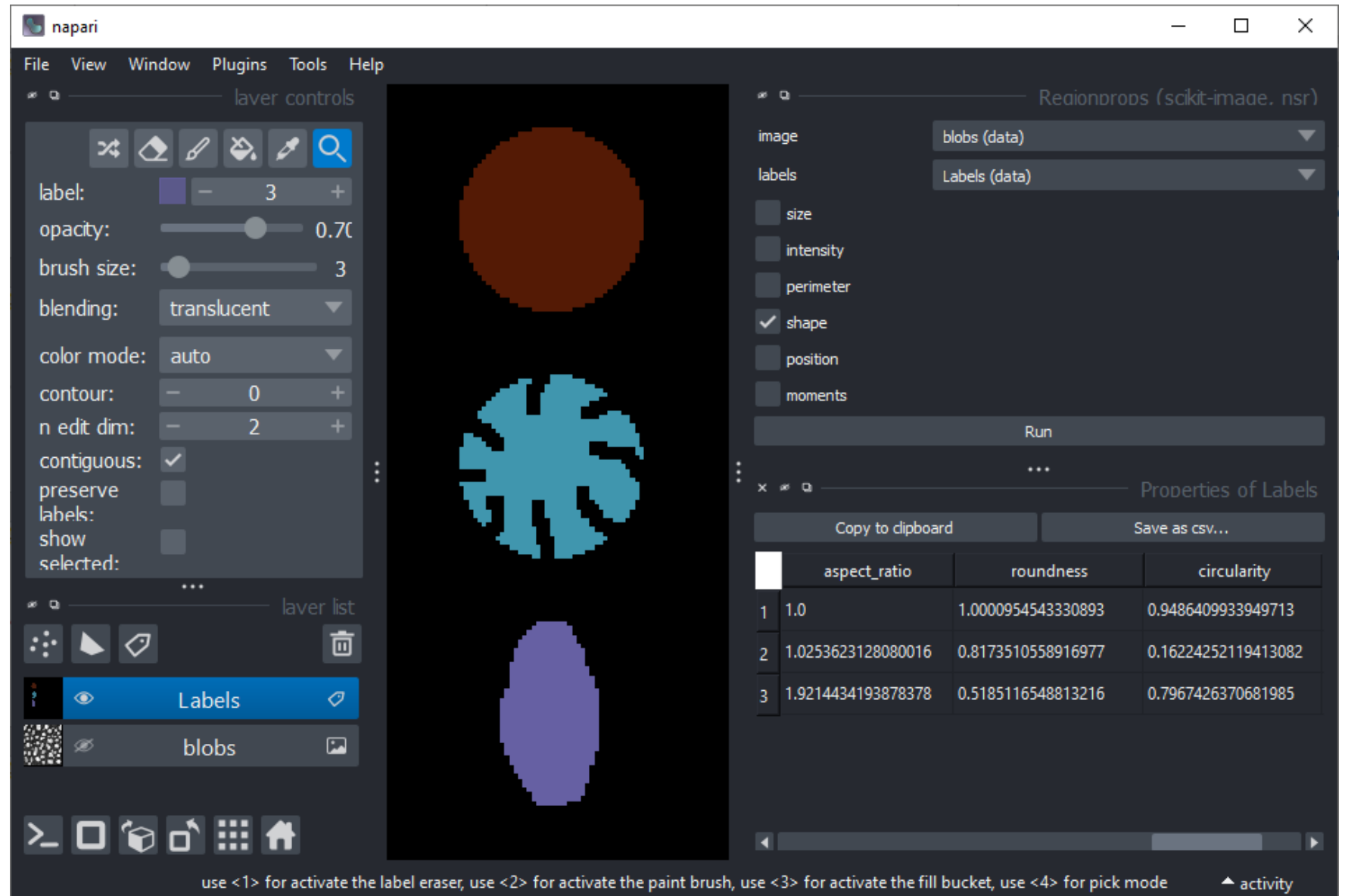
$$roundness = \frac{4 * A}{\pi major^2}$$

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



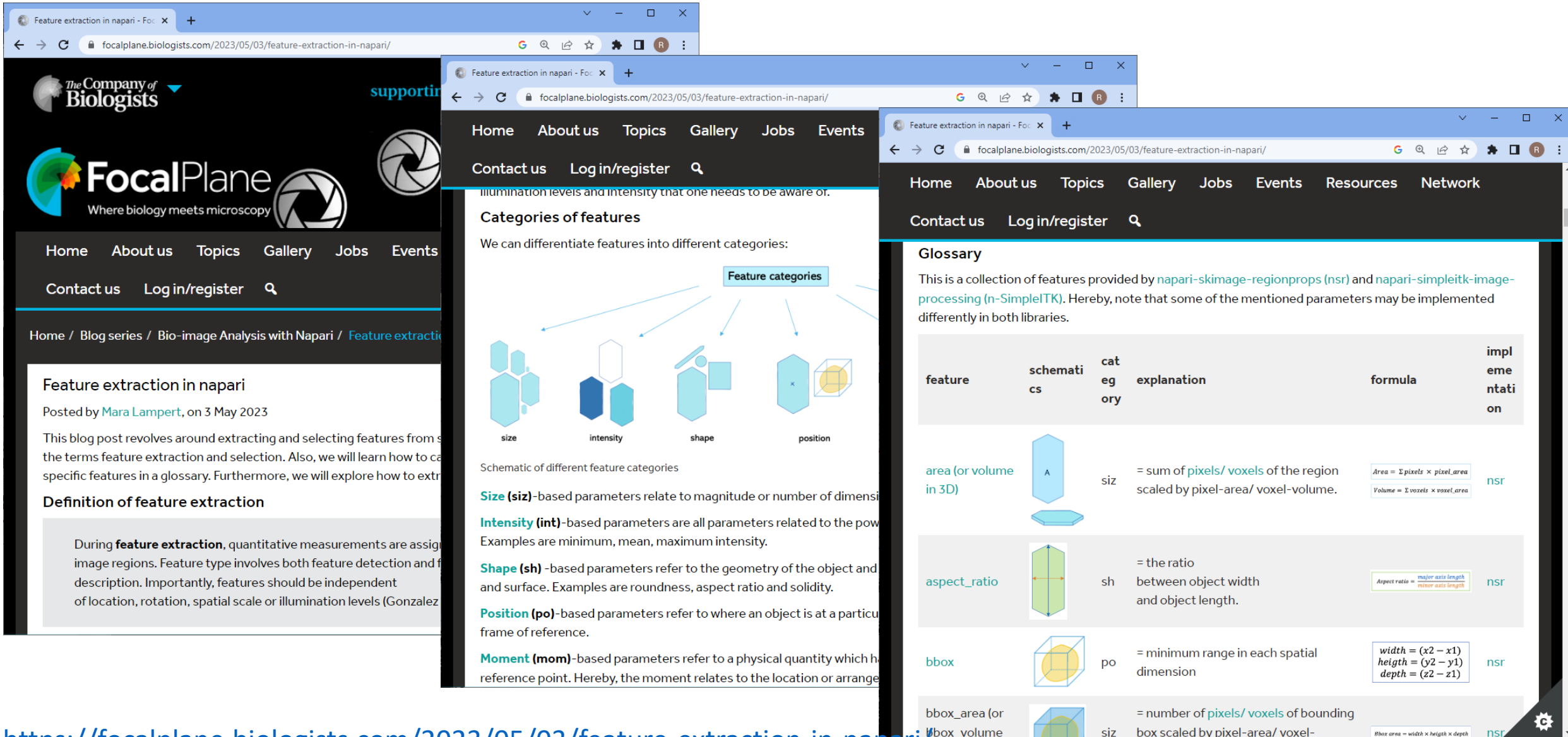
What's the difference between circularity and roundness?

- Draw representative shapes and study measurements



The screenshot shows the napari software interface with three labeled blobs in the center. The left panel shows layer controls for the 'Labels' layer, and the right panel shows the 'Regionprops (scikit-image, nsr)' panel with a table of properties for the three blobs.

	aspect_ratio	roundness	circularity
1	1.0	1.0000954543330893	0.9486409933949713
2	1.0253623128080016	0.8173510558916977	0.16224252119413082
3	1.9214434193878378	0.5185116548813216	0.7967426370681985



Feature extraction in napari

Posted by [Mara Lampert](#), on 3 May 2023

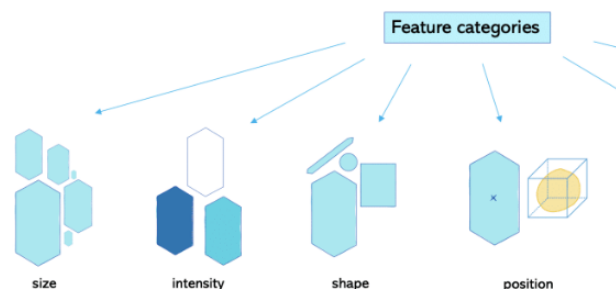
This blog post revolves around extracting and selecting features from images. In this post, we will use the terms feature extraction and selection. Also, we will learn how to categorize specific features in a glossary. Furthermore, we will explore how to extract features from images.

Definition of feature extraction

During **feature extraction**, quantitative measurements are assigned to image regions. Feature type involves both feature detection and feature description. Importantly, features should be independent of location, rotation, spatial scale or illumination levels (Gonzalez and Woods, 2008).

Categories of features

We can differentiate features into different categories:

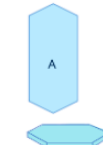
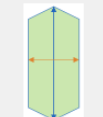

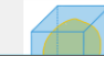


Schematic of different feature categories

- Size (siz)**-based parameters relate to magnitude or number of dimensions. Examples are area, volume, perimeter, and length.
- Intensity (int)**-based parameters are all parameters related to the power of the signal. Examples are minimum, mean, maximum intensity.
- Shape (sh)**-based parameters refer to the geometry of the object and its surface. Examples are roundness, aspect ratio and solidity.
- Position (po)**-based parameters refer to where an object is at a particular frame of reference.
- Moment (mom)**-based parameters refer to a physical quantity which has a reference point. Hereby, the moment relates to the location or arrangement of the object.

Glossary

This is a collection of features provided by [napari-skimage-regionprops \(nsr\)](#) and [napari-simpleitk-image-processing \(n-SimpleITK\)](#). Hereby, note that some of the mentioned parameters may be implemented differently in both libraries.

feature	schematic	category	explanation	formula	implementation
area (or volume in 3D)		siz	= sum of pixels/ voxels of the region scaled by pixel-area/ voxel-volume.	$\text{Area} = \sum \text{pixels} \times \text{pixel_area}$ $\text{Volume} = \sum \text{voxels} \times \text{voxel_area}$	nsr
aspect_ratio		sh	= the ratio between object width and object length.	$\text{Aspect ratio} = \frac{\text{major axis length}}{\text{minor axis length}}$	nsr
bbox		po	= minimum range in each spatial dimension	$\text{width} = (x2 - x1)$ $\text{height} = (y2 - y1)$ $\text{depth} = (z2 - z1)$	nsr
bbox_area (or bbox_volume)		siz	= number of pixels/ voxels of bounding box scaled by pixel-area/ voxel-volume	$\text{Bbox area} = \text{width} \times \text{height} \times \text{depth}$	nsr

<https://focalplane.biologists.com/2023/05/03/feature-extraction-in-napari/>



Pixel classification using scikit-learn

Robert Haase

June 2023

- Classify objects starting from feature vectors (table columns)

Raw data

	area	elongation
0	3.950088	2.848643
1	4.955912	3.390093
2	7.469852	5.575289
3	2.544467	3.017479
4	3.465662	1.463756
5	3.156507	3.232181
6	9.978705	6.676372
7	6.001683	5.047063
8	2.457139	3.416050
9	3.672295	3.407462
10	9.413702	7.598608

“Ground truth” annotation

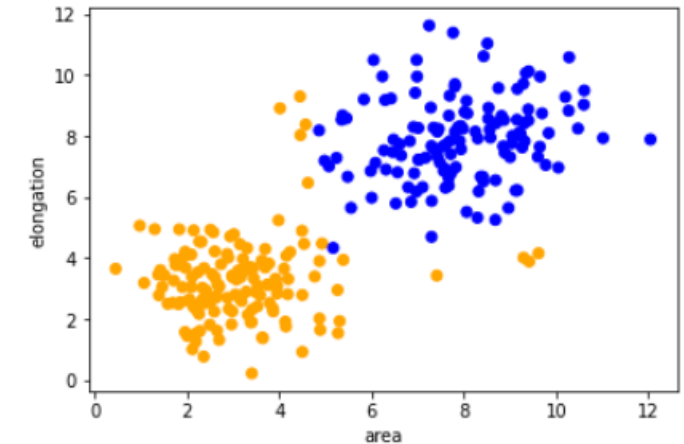
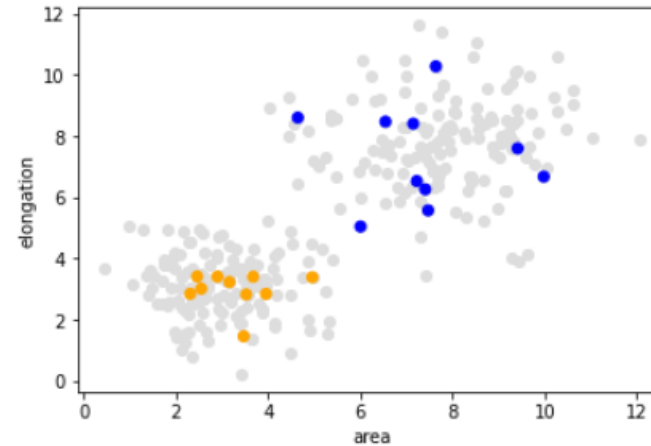
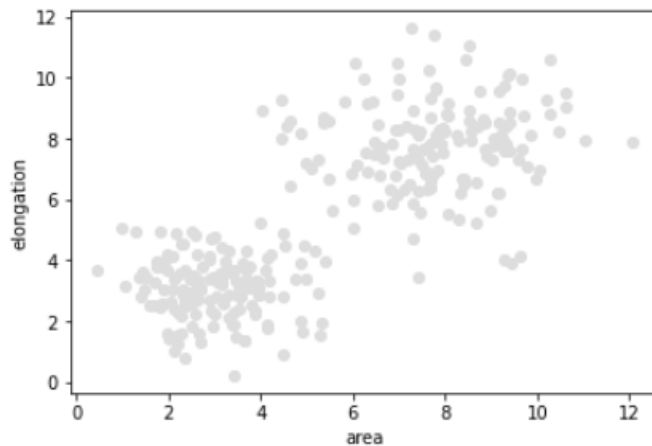
```
annotation = [1, 1, 2, 1, 1, 1, 2, 2,
```

Classifier training

```
classifier = RandomForestClassifier()  
classifier.fit(train_data, train_annotation)
```

Classifier prediction

```
result = classifier.predict(validation_data)
```



Interactive pixel classification

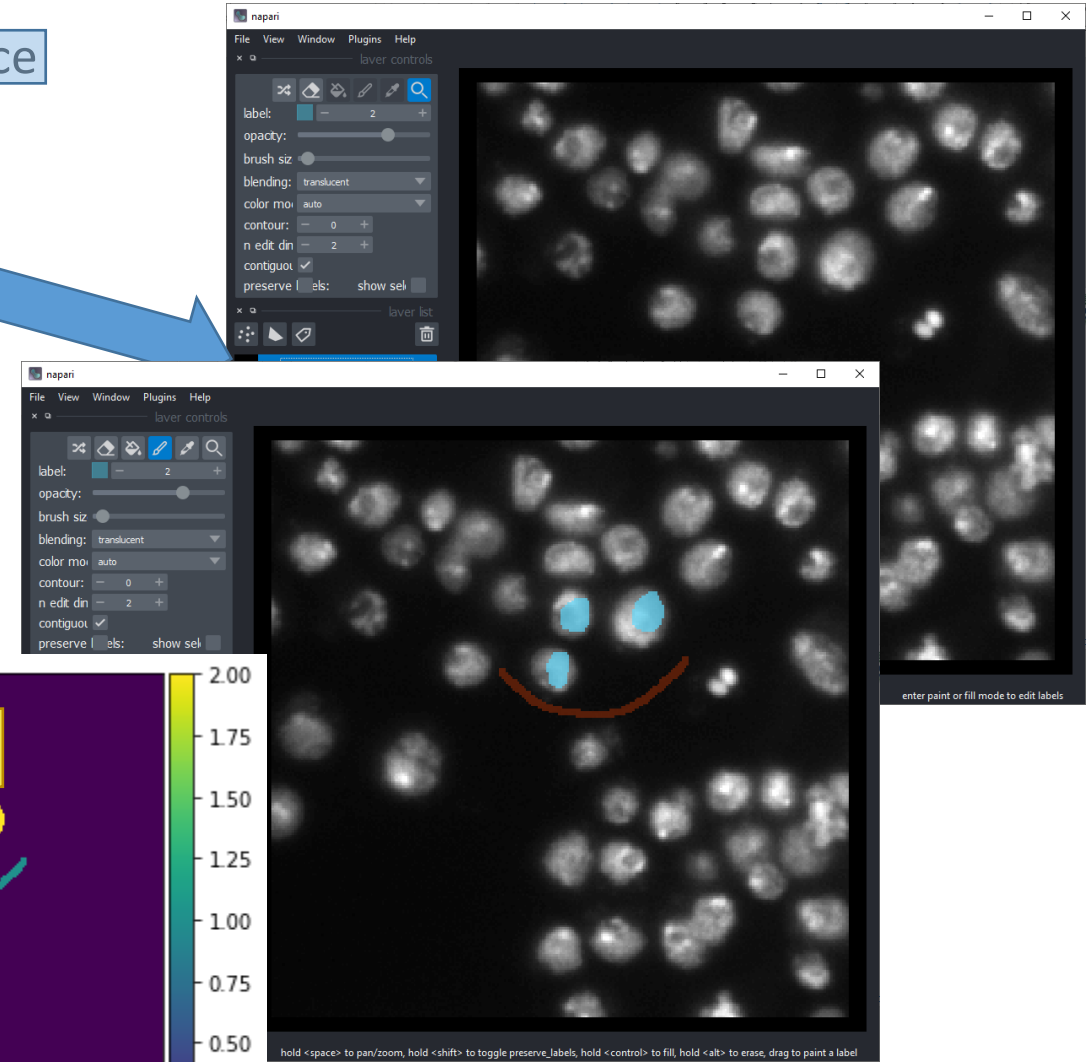
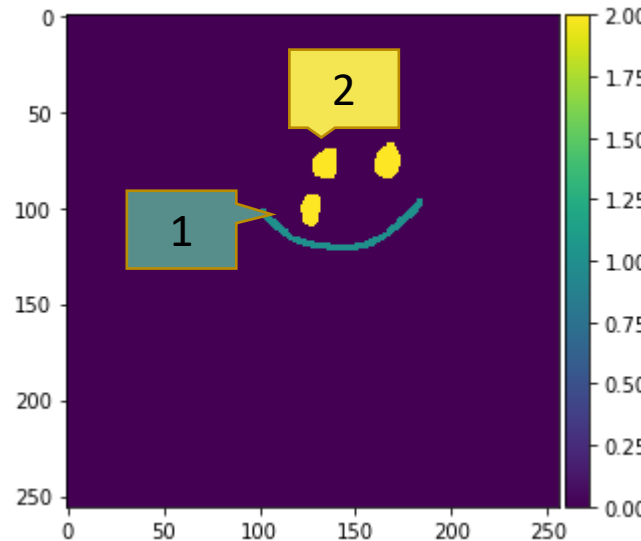
- Prepare an empty layer for annotations and keep a `reference`

```
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)
```



- Pixel classification using scikit-learn
 - Expects one-dimensional arrays for
 - every feature individually
 - ground truth

```
# train classifier
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(max_depth=2, random_state=0)
```

```
classifier.fit(X, y)
```

Image data

Ground truth /
annotation

Image data

```
y_ = classifier.predict(X)
```

prediction

- Pixel classification using scikit-learn
 - Expects one-dimensional arrays for
 - every feature individually
 - ground truth

```
# for training, we need to generate features
```

```
feature_stack = generate_feature_stack(image)
```

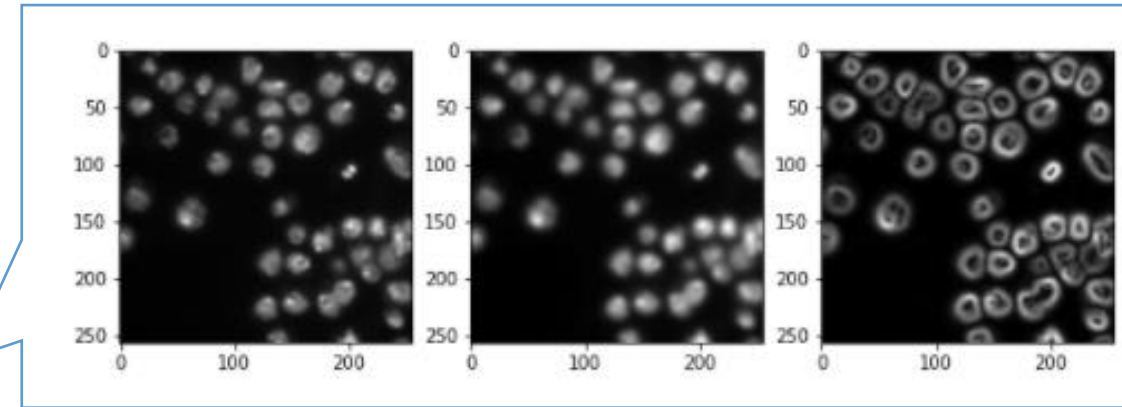
```
X, y = format_data(feature_stack, manual_annotations)
```

```
# train classifier
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(max_depth=2, random_state=0)
```

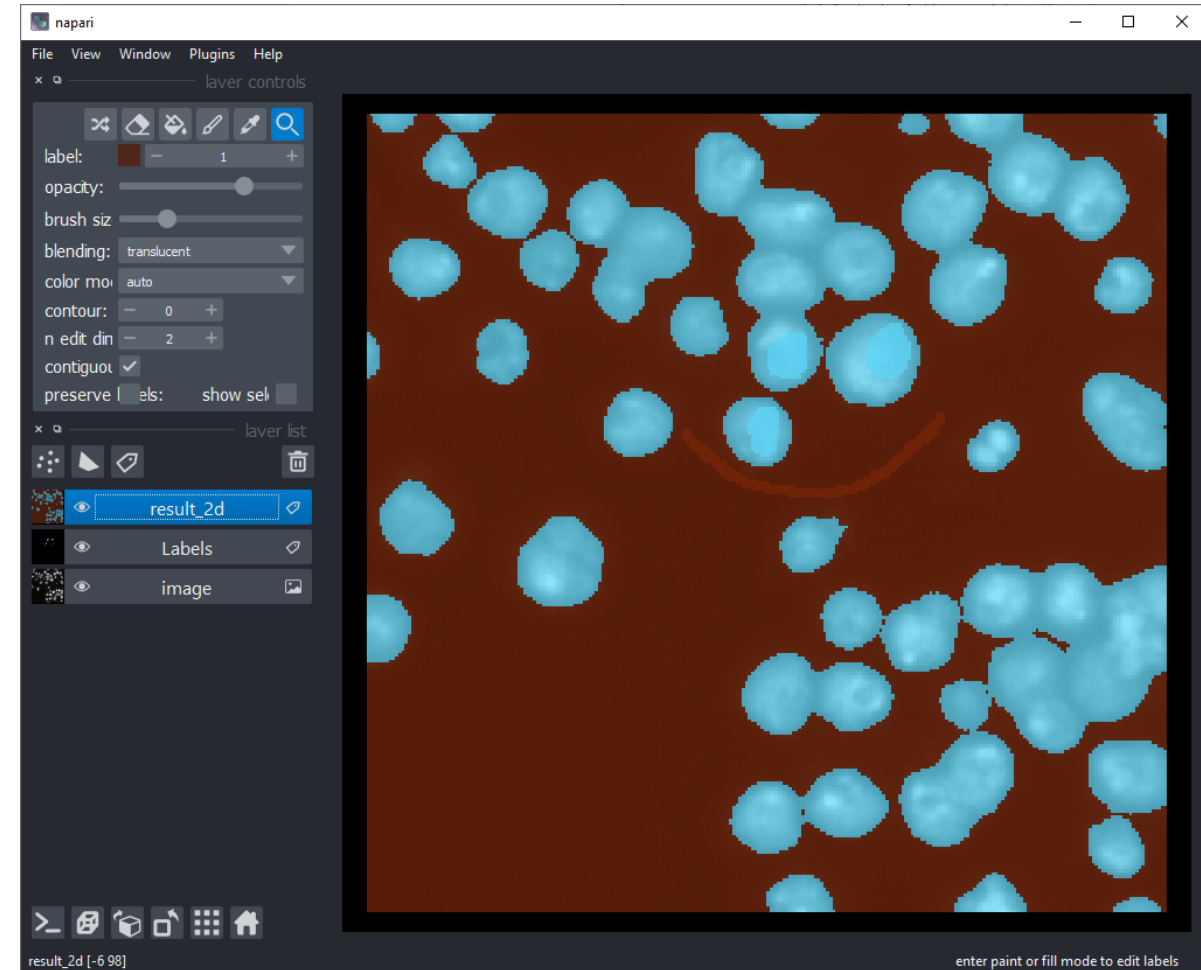
```
classifier.fit(X, y)
```



- Pixel classification using scikit-learn

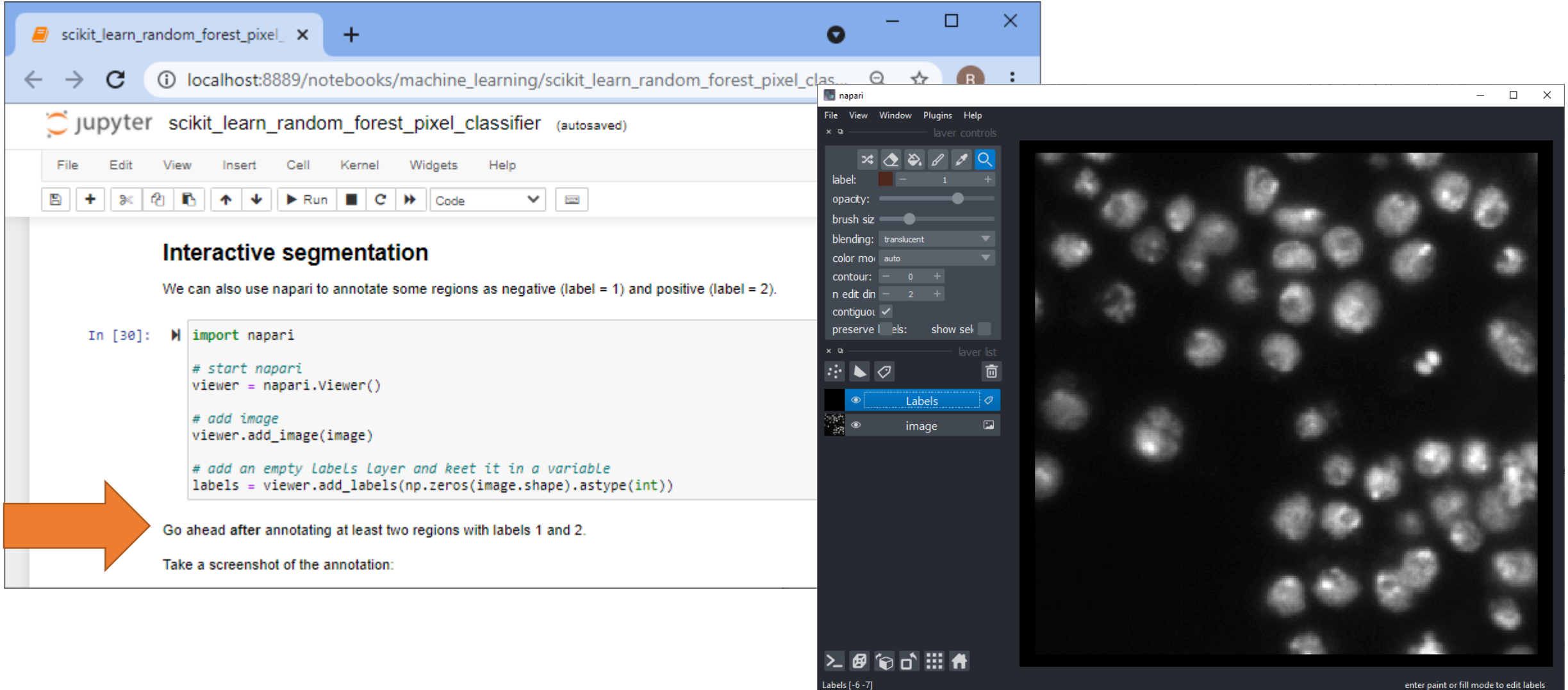
```
# process the whole image and show result
result_1d = classifier.predict(feature_stack.T)
result_2d = result_1d.reshape(image.shape)

viewer.add_labels(result_2d)
```



Interactive pixel classification

- Jupyter notebooks and napari side-by-side



The image shows a side-by-side view of a Jupyter Notebook and the napari application. The Jupyter Notebook on the left displays a code cell with the following Python code:

```
In [30]: import napari

# start napari
viewer = napari.Viewer()

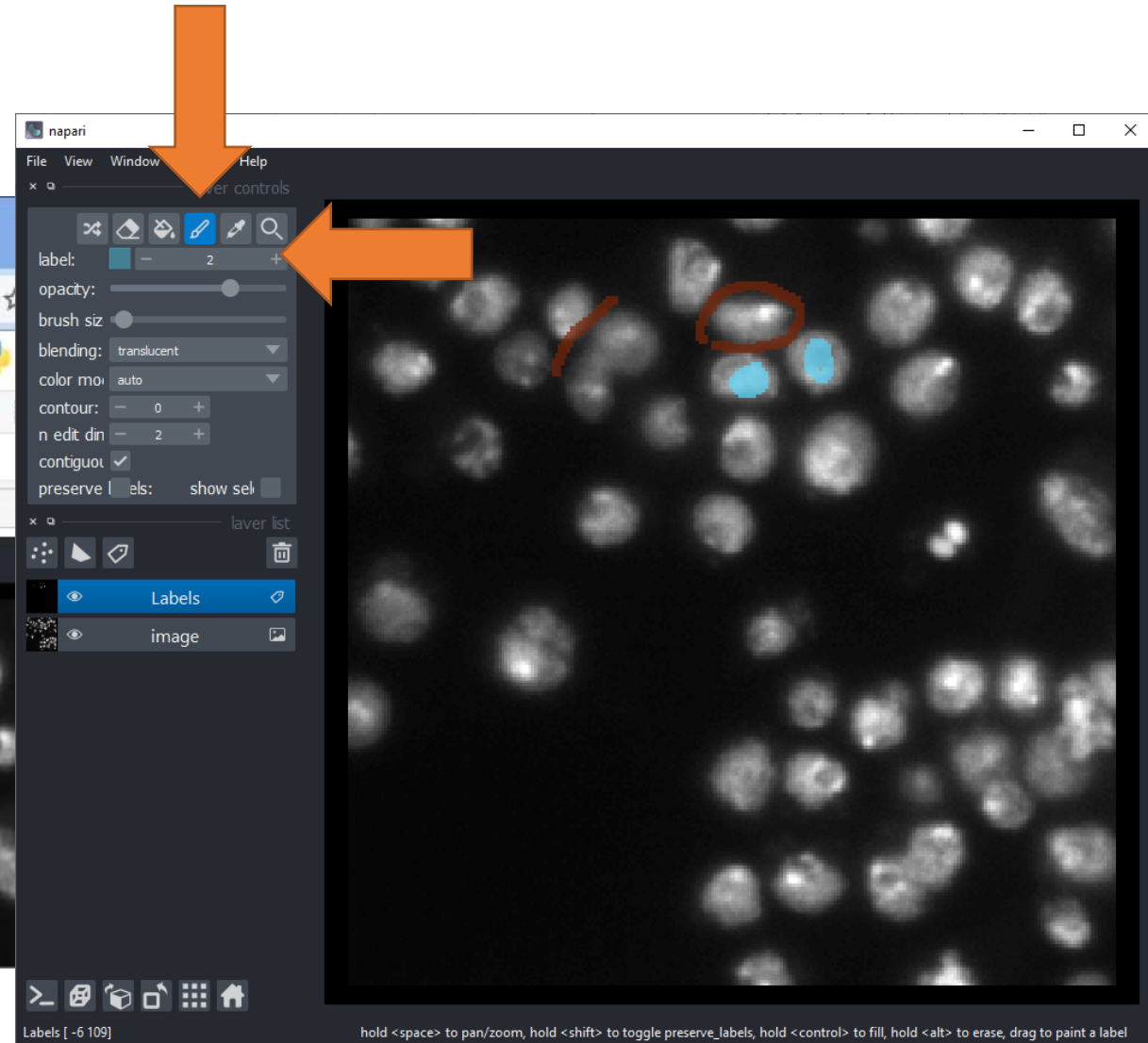
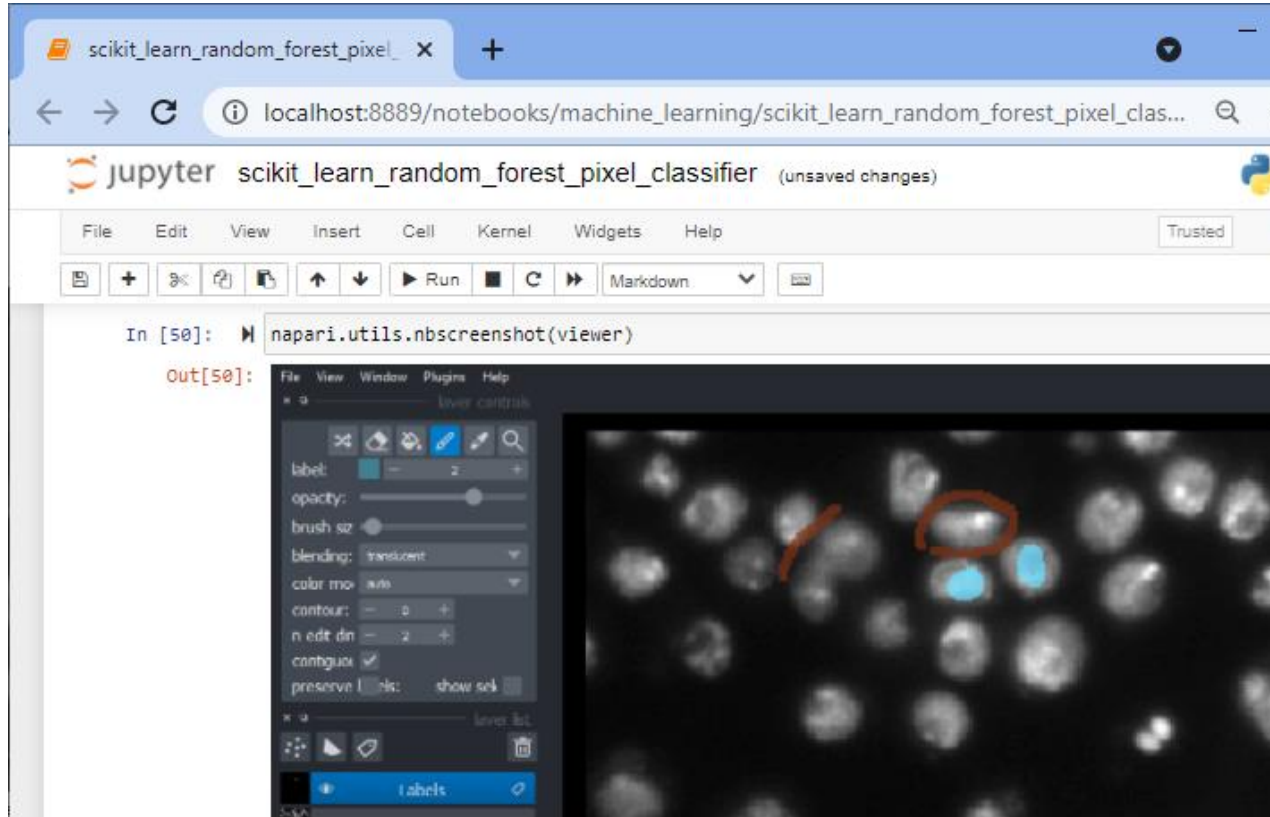
# add image
viewer.add_image(image)

# add an empty Labels Layer and keep it in a variable
labels = viewer.add_labels(np.zeros(image.shape).astype(int))
```

Below the code, an orange arrow points to the text: "Go ahead after annotating at least two regions with labels 1 and 2. Take a screenshot of the annotation:". The napari application on the right shows a grayscale image of biological cells. The interface includes a "layer controls" panel on the left with settings for label (1), opacity, brush size, blending (translucent), color mode (auto), contour (0), n edit dim (2), contiguous (checked), and preserve labels (show selected). The "layer list" at the bottom shows "Labels" and "image" layers. The status bar at the bottom of the napari window reads "Labels [-6 -7]" and "enter paint or fill mode to edit labels".

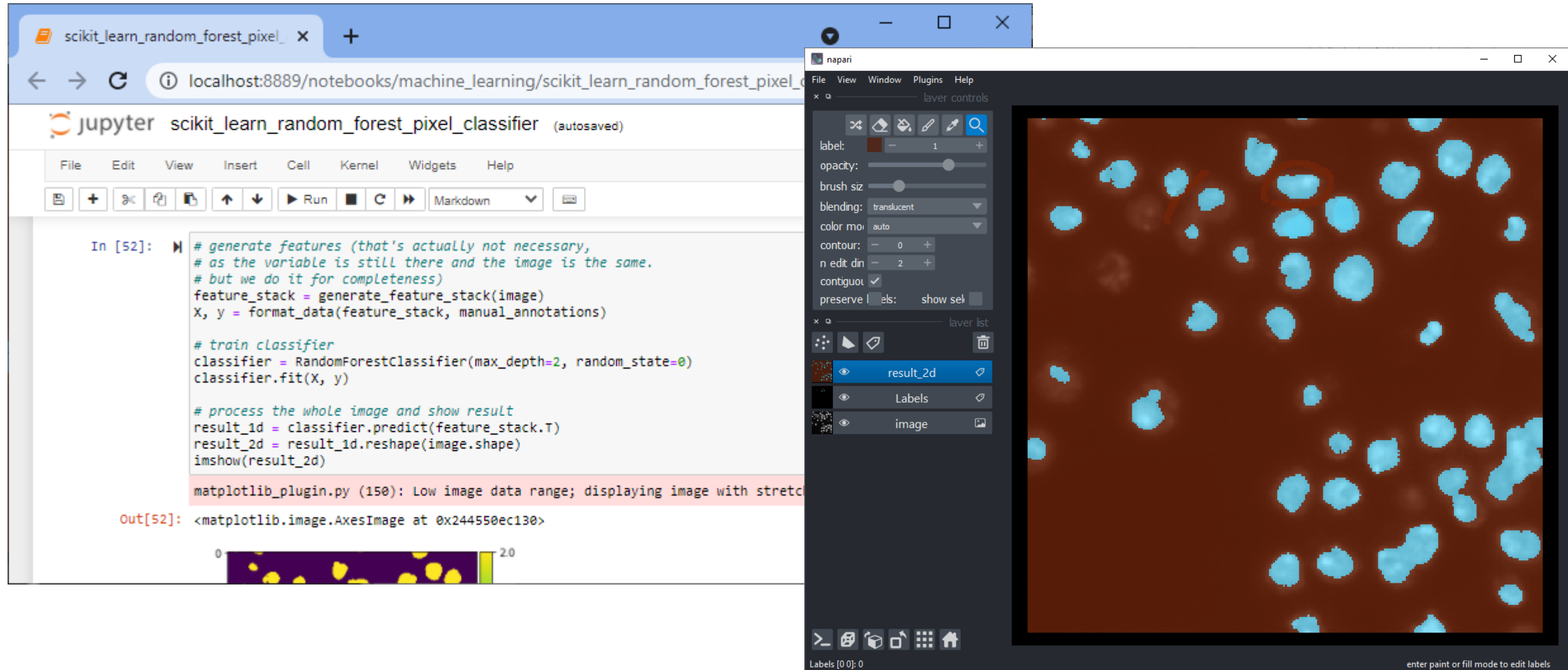
Interactive pixel classification

- Jupyter notebooks and napari side-by-side



Interactive pixel classification

- Jupyter notebooks and napari side-by-side



The image displays a side-by-side workflow. On the left, a Jupyter notebook titled "scikit_learn_random_forest_pixel_classifier" (autosaved) is open in a browser at localhost:8889. The notebook contains Python code for feature generation, classifier training, and image processing. The output shows a small thumbnail of a cell image with yellow spots on a dark background. On the right, the napari application is open, showing a larger view of the same image. The napari interface includes a layer controls panel on the left with settings for label, opacity, brush size, blending, color mode, contour, n edit, and contiguous. The main view shows the image with cyan-colored spots overlaid on a dark brown background, representing the classification results. The napari layer list on the left shows "result_2d", "Labels", and "image".

```
In [52]: # generate features (that's actually not necessary,  
# as the variable is still there and the image is the same.  
# but we do it for completeness)  
feature_stack = generate_feature_stack(image)  
X, y = format_data(feature_stack, manual_annotations)  
  
# train classifier  
classifier = RandomForestClassifier(max_depth=2, random_state=0)  
classifier.fit(X, y)  
  
# process the whole image and show result  
result_1d = classifier.predict(feature_stack.T)  
result_2d = result_1d.reshape(image.shape)  
imshow(result_2d)
```

matplotlib_plugin.py (150): Low image data range; displaying image with stretch

Out[52]: <matplotlib.image.AxesImage at 0x244550ec130>

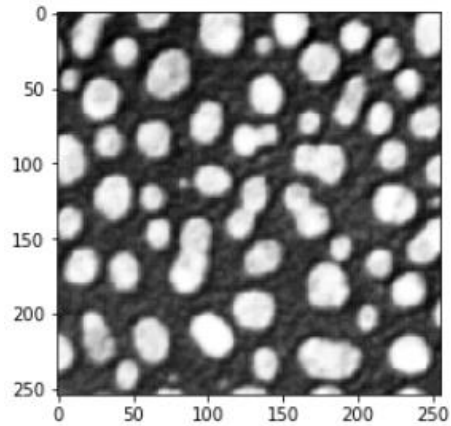
Accelerated pixel and object classification (APOC)

Robert Haase

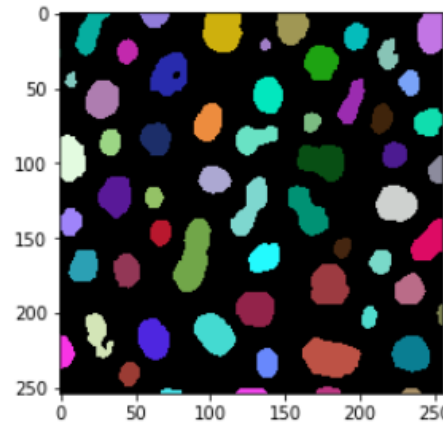
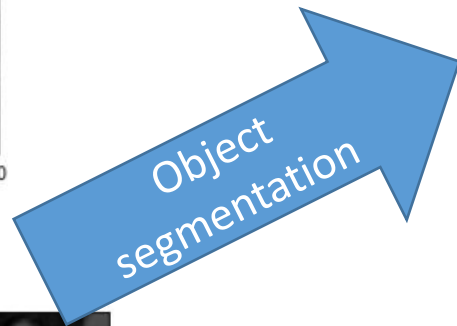
October 2022

Accelerated pixel and object classification

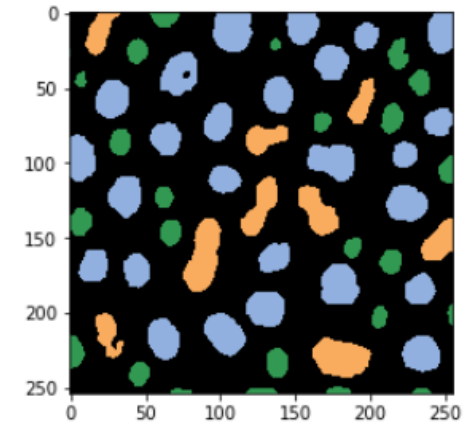
- APOC is a python library that makes use of OpenCL-compatible Graphics Cards to accelerate pixel and object classification



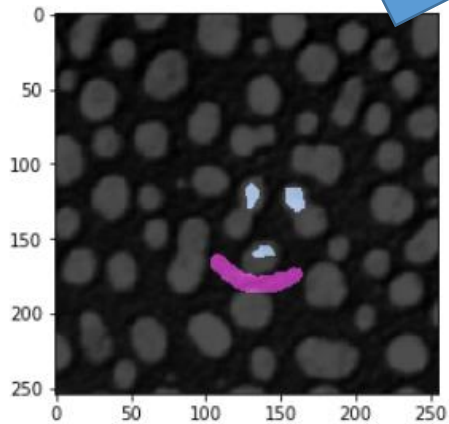
Raw image



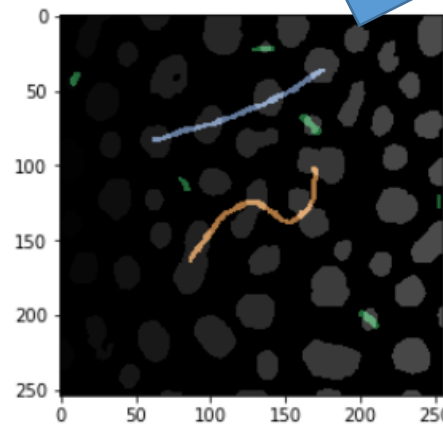
Object label image



Class label image

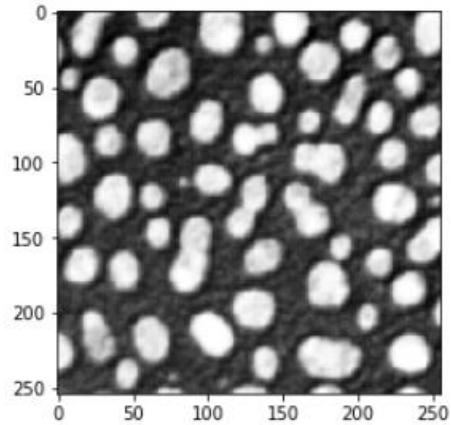


Pixel annotation

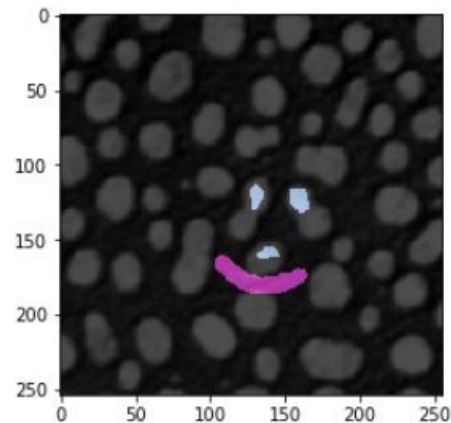


Object annotation

- Pixel classification + connected component labeling



Raw image



Pixel annotation

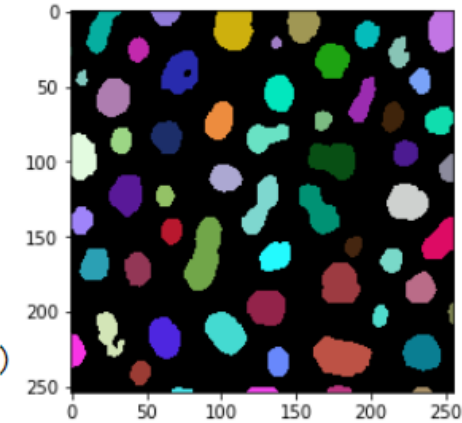
```
# define features
features = "gaussian_blur=1 gaussian_blur=5 sobel_of_gaussian_blur=1"

# this is where the model will be saved
cl_filename = 'my_object_segmenter.cl'

# delete classifier in case the file exists already
apoc.erase_classifier(cl_filename)

# train classifier
clf = apoc.ObjectSegmenter(opencl_filename=cl_filename, positive_class_identifier=2)
clf.train(features, manual_annotations, image)

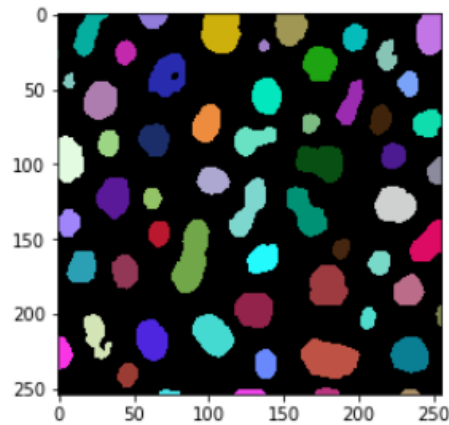
segmentation_result = clf.predict(features=features, image=image)
cle.imshow(segmentation_result, labels=True)
```



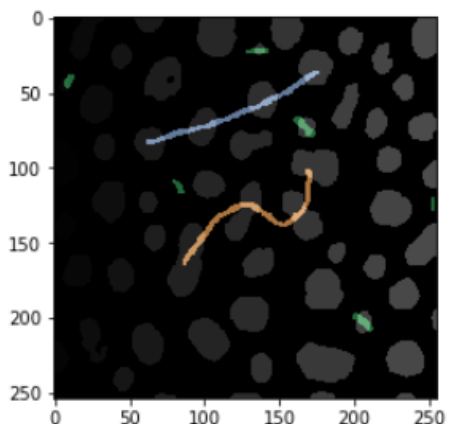
Object label image

Object segmentation

- Feature extraction + tabular classification



Object label image



Object annotation

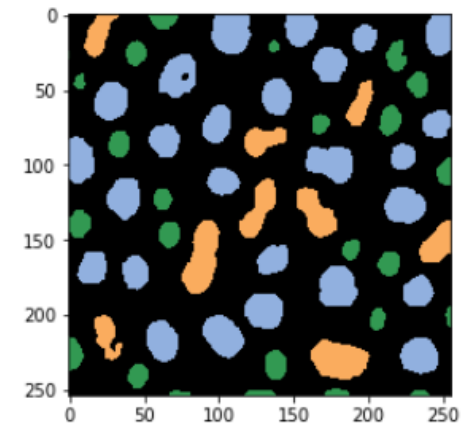
```
# for the classification we define size and shape as criteria  
features = 'area mean_max_distance_to_centroid_ratio'
```

```
# This is where the model will be saved  
cl_filename_object_classifier = "my_object_classifier.cl"
```

```
# delete classifier in case the file exists already  
apoc.erase_classifier(cl_filename_object_classifier)
```

```
# train the classifier  
classifier = apoc.ObjectClassifier(cl_filename_object_classifier)  
classifier.train(features, segmentation_result, annotation, image)
```

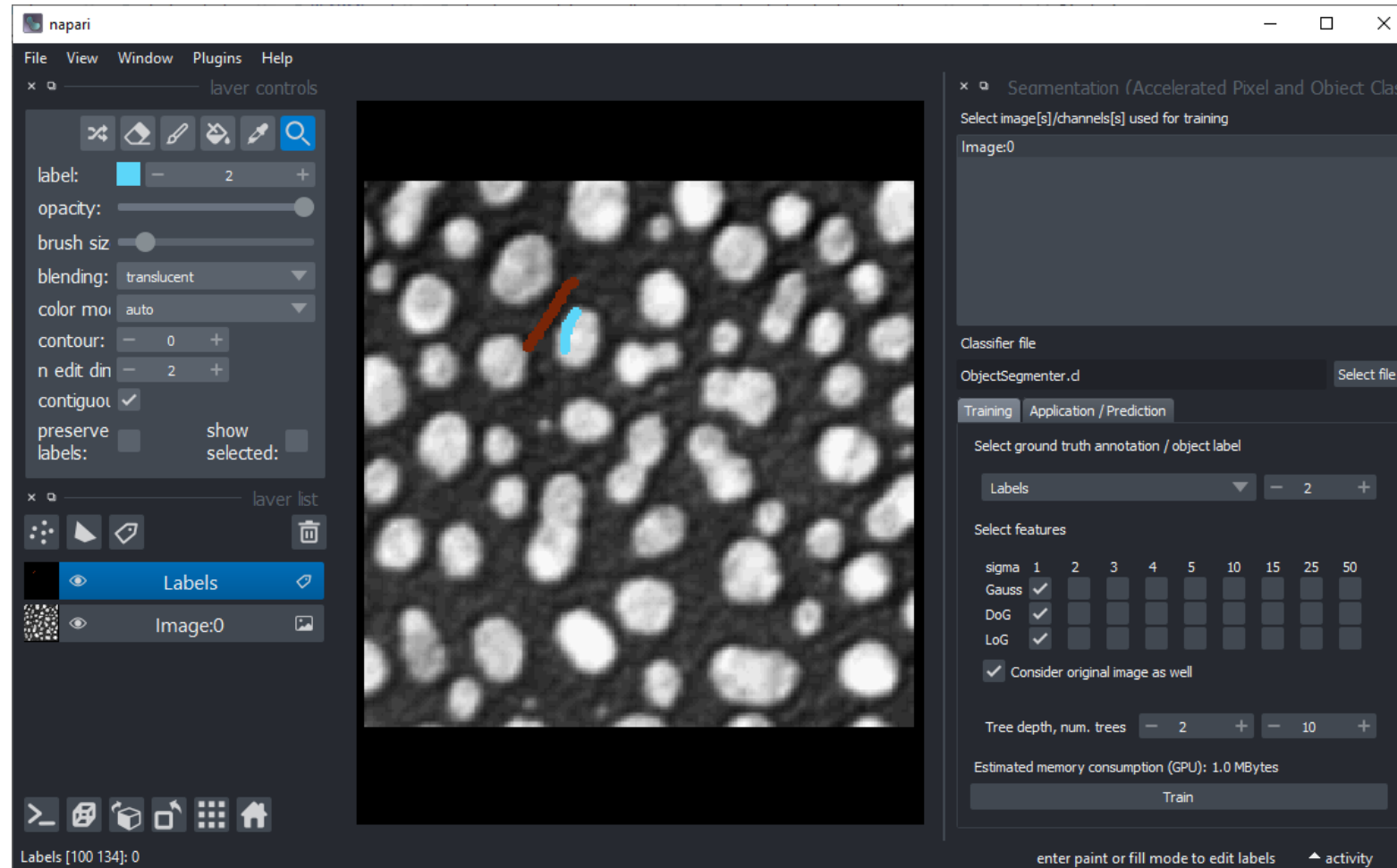
```
# determine object classification  
classification_result = classifier.predict(segmentation_result, image)  
cle.imshow(classification_result, labels=True)
```



Class label image

Object classification

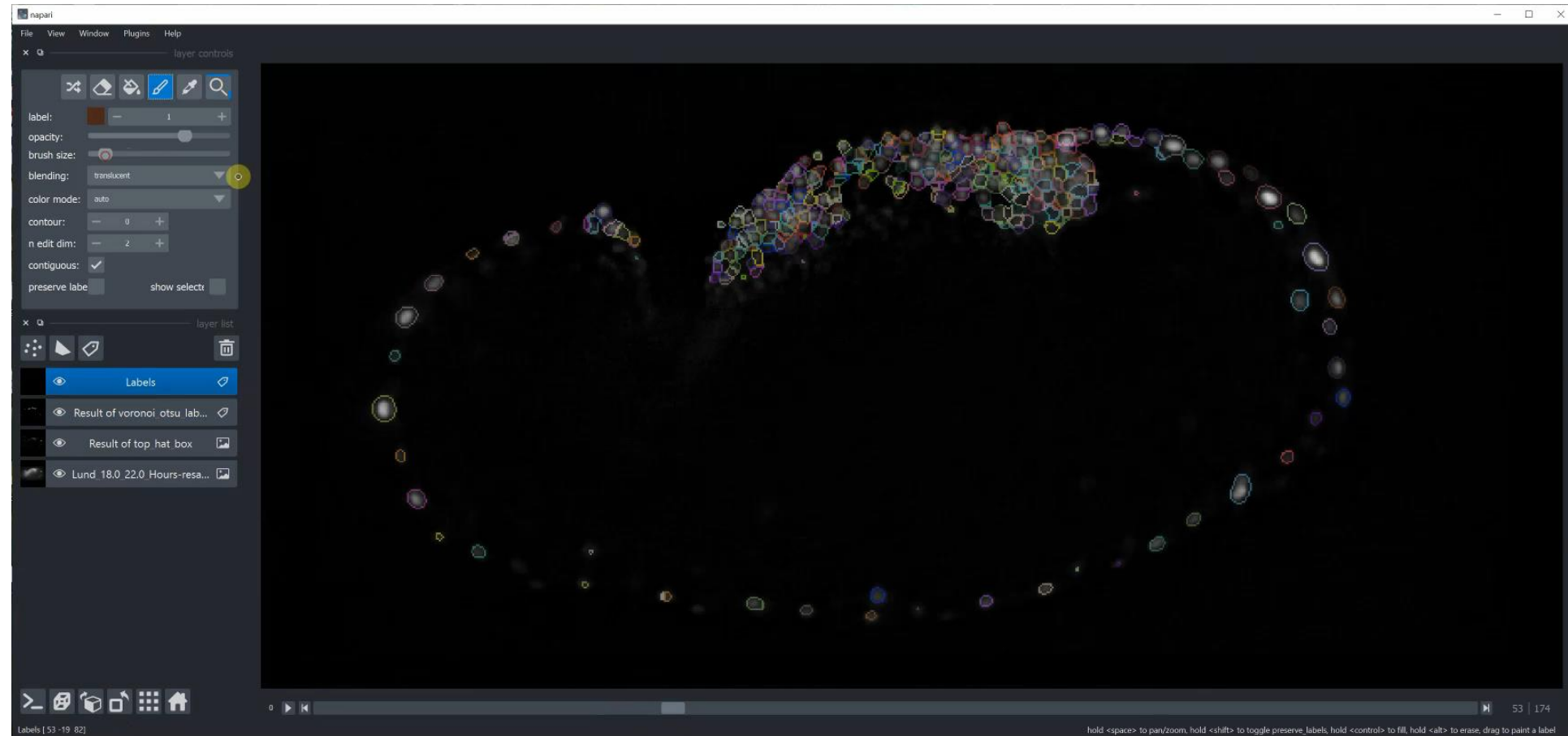
- Object segmentation
- <https://github.com/haesleinhuepf/napari-accelerated-pixel-and-object-classification#object-and-semantic-segmentation>



Supervised machine learning for tissue classification

Random Forest
Classifiers based on

- scikit-learn and
- clesperanto



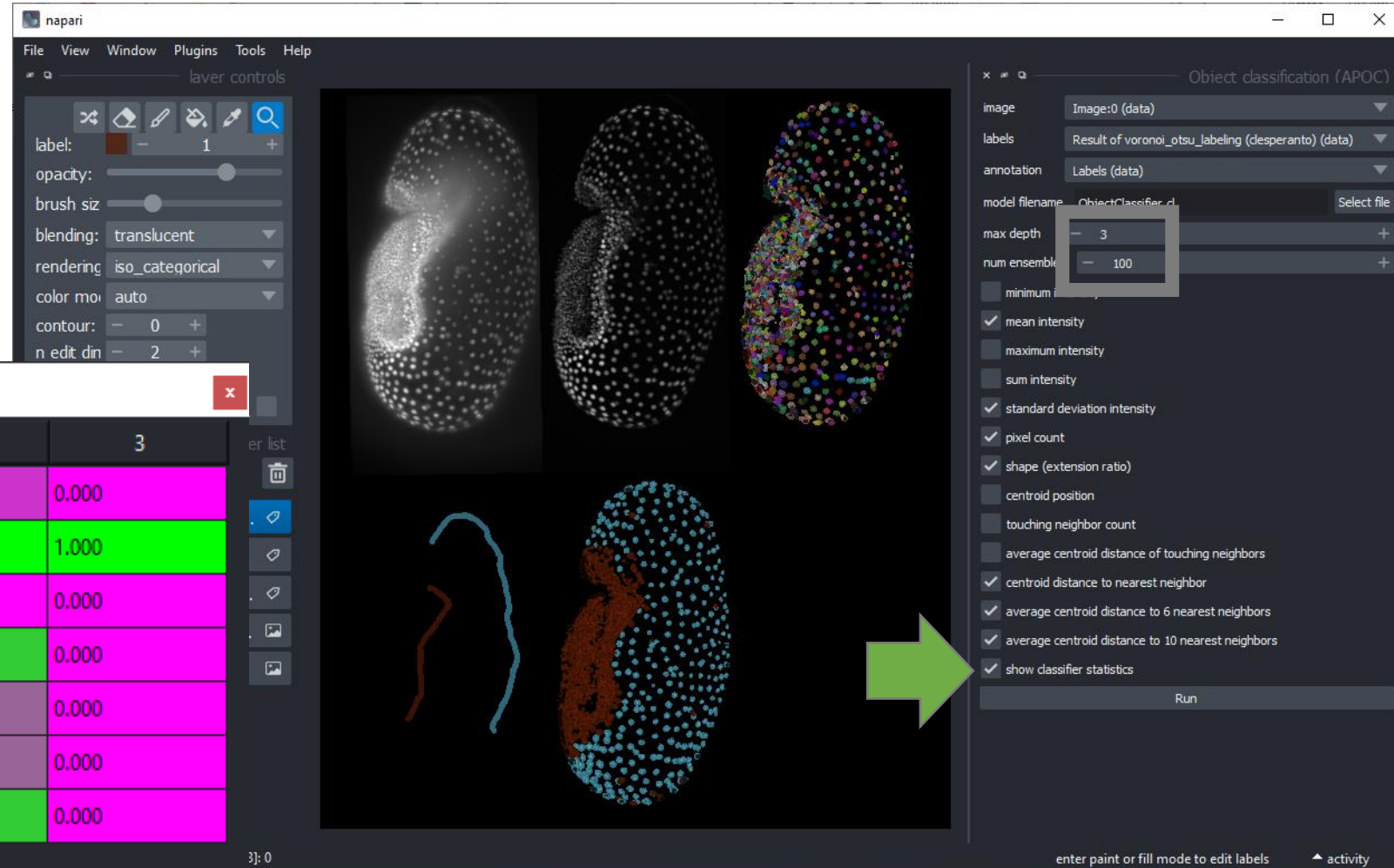
<https://github.com/haesleinhuepf/napari-accelerated-pixel-and-object-classification>

@haesleinhuepf
@PoLDresden

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

Data exploration / supervised machine learning

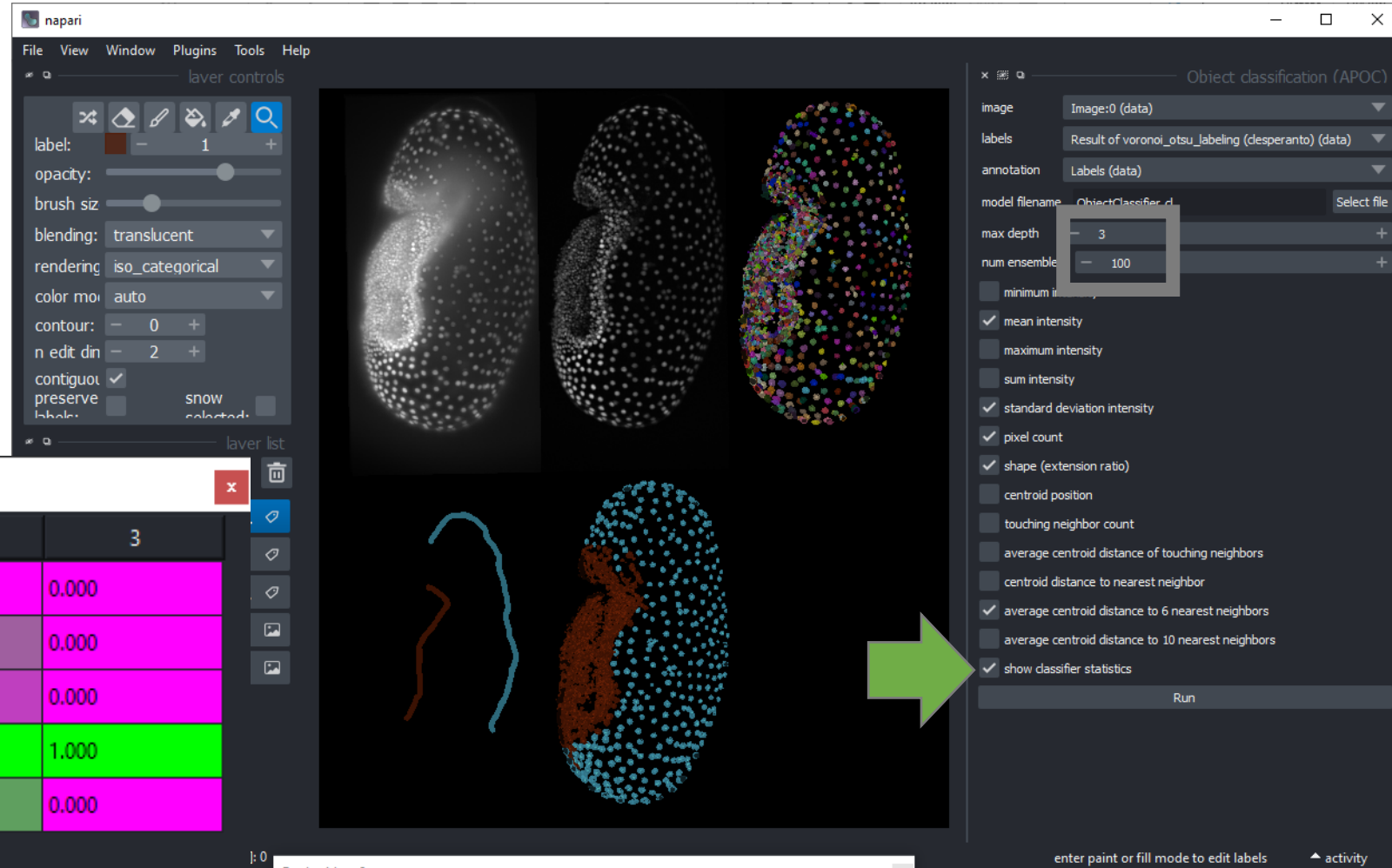
- Inspect how the random forest classifier makes decisions
- Note: Beware of correlated parameters!



	1	2	3
area	0.010	0.056	0.000
mean_intensity	0.200	0.278	1.000
standard_deviation_intensity	0.030	0.000	0.000
mean_max_distance_to_centroid_ratio	0.270	0.222	0.000
average_distance_of_n_nearest_neighbors=1	0.120	0.111	0.000
average_distance_of_n_nearest_neighbors=6	0.170	0.111	0.000
average_distance_of_n_nearest_neighbors=10	0.200	0.222	0.000

Data exploration / supervised machine learning

- Inspect how the random forest classifier makes decisions
- Note: Beware of correlated parameters!

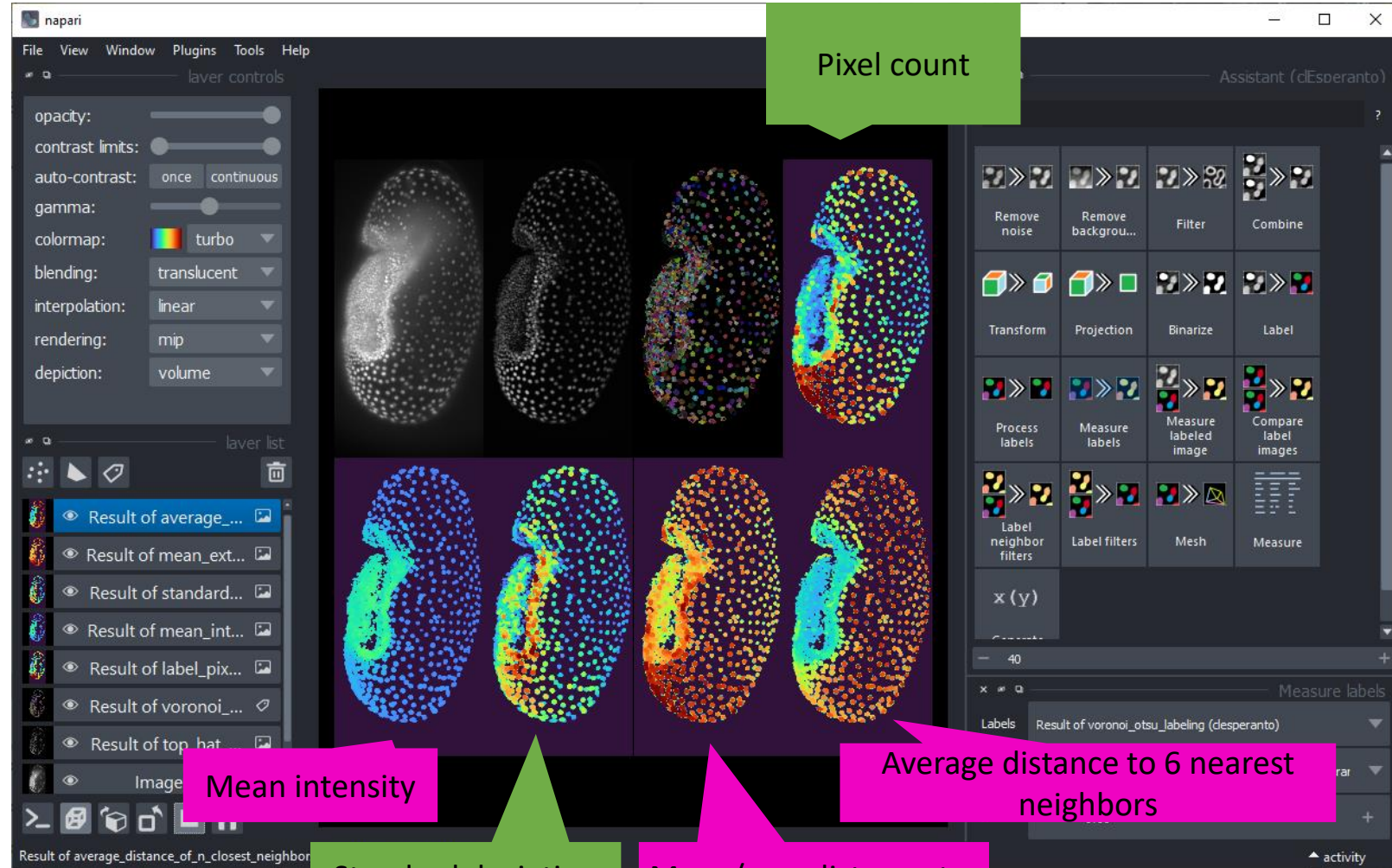


The screenshot shows the Napari interface with a central view of three kidney images: original, thresholded, and segmented. The right panel shows the 'Object classification (APOC)' settings, with 'max depth' set to 3 and 'num ensemble' set to 100. The bottom left panel shows a table of classifier statistics for three classes.

	1	2	3
area	0.060	0.000	0.000
mean_intensity	0.330	0.167	0.000
standard_deviation_intensity	0.040	0.111	0.000
mean_max_distance_to_centroid_ratio	0.260	0.444	1.000
average_distance_of_n_nearest_neighbors=6	0.310	0.278	0.000

Data exploration / supervised machine learning

- Inspect how the random forest classifier makes decisions
- Note: Beware of correlated parameters!



The screenshot shows the napari interface with a central view of four kidney images. The top row shows the original image, a thresholded binary image, a segmented image, and a pixel count map. The bottom row shows the segmented image with different color mappings. The dock widget on the left contains a table with the following data:

	1	2	3
area	0.060	0.000	0.000
mean_intensity	0.330	0.167	0.000
standard_deviation_intensity	0.040	0.111	0.000
mean_max_distance_to_centroid_ratio	0.260	0.444	1.000
average_distance_of_n_nearest_neighbors=6	0.310	0.278	0.000

Annotations in the image include:

- Pixel count** (green callout pointing to the top-right image)
- Mean intensity** (pink callout pointing to the bottom-left image)
- Standard deviation of intensity** (green callout pointing to the bottom-middle image)
- Mean/max distance-to-centroid ratio** (pink callout pointing to the bottom-right image)
- Average distance to 6 nearest neighbors** (pink callout pointing to the bottom-right image)

https://github.com/clEsperanto/napari_pyclesperanto_assistant

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

Exercises

- Use Napari to segment objects

Interactive pixel classification and object segmentation in Napari

In this exercise we will train a [Random Forest Classifier](#) for pixel classification and convert the result in an instance segmentation. We will use the napari plugin [napari-accelerated-pixel-and-object-classification](#).

Getting started

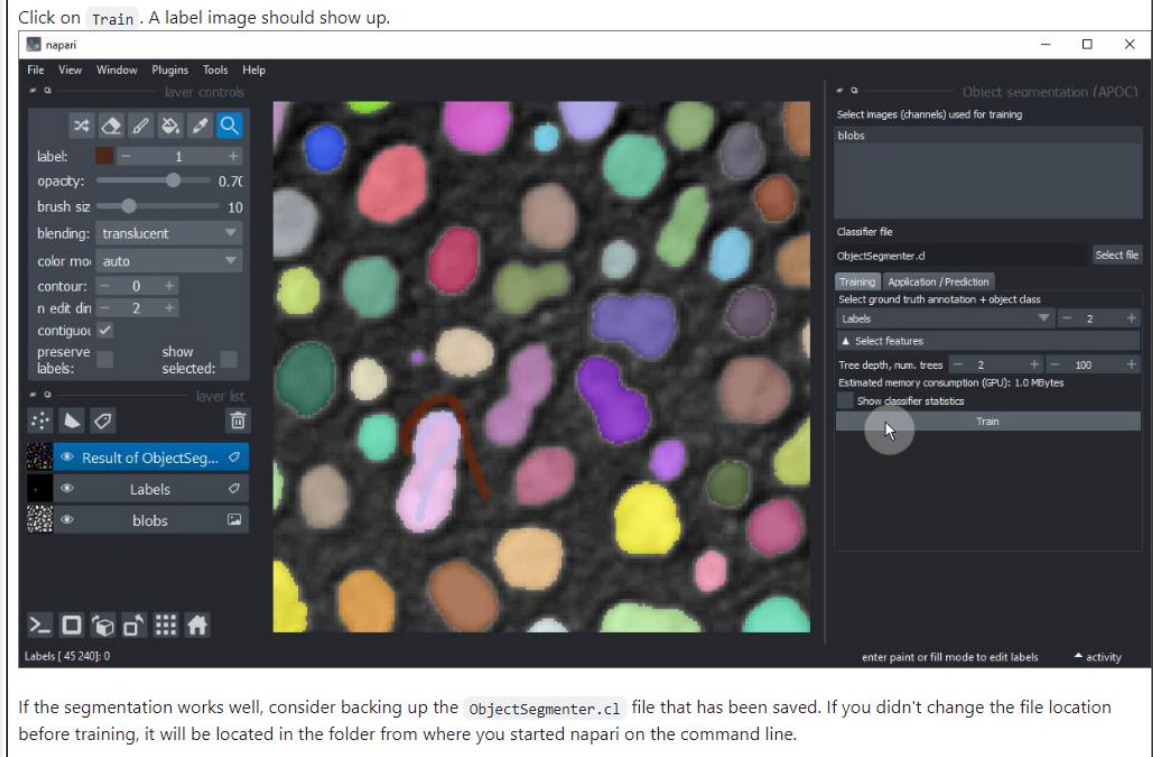
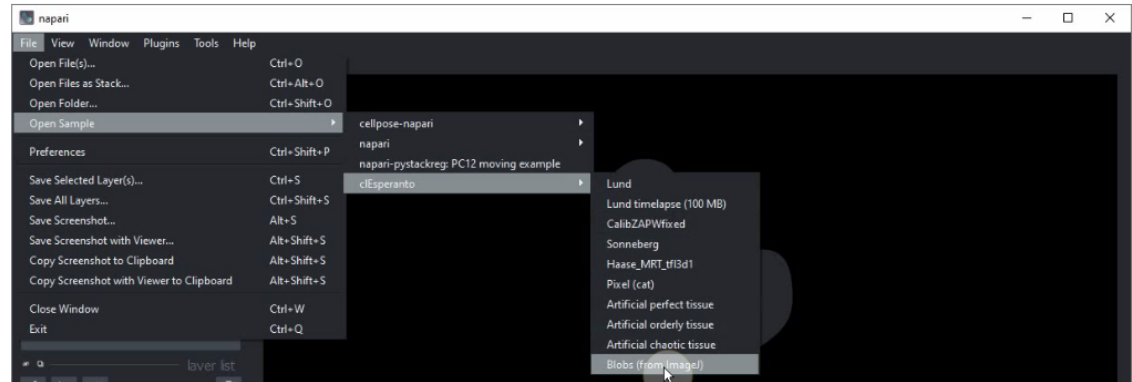
Open a terminal window and activate your conda environment:

```
conda activate devbio-napari-env
```

Afterwards, start up Napari:

```
napari
```

Load the "Blobs" example dataset from the menu `File > Open Sample > c1Esperanto > Blobs (from ImageJ)`



If the segmentation works well, consider backing up the `objectSegmenter.c1` file that has been saved. If you didn't change the file location before training, it will be located in the folder from where you started napari on the command line.

<https://github.com/BiAPoL/Bio-image Analysis with Python/blob/main/09 machine learning /interactive pixel classification/readme.md>

- Use Napari to group round and elongated objects

Interactive object classification in Napari

In this exercise we will train a [Random Forest Classifiers](#) for classifying segmented objects. We will use the napari plugin [napari-accelerated-pixel-and-object-classification](#).

Getting started

Open a terminal window and activate your conda environment:

```
conda activate devbio-napari-env
```

Afterwards, start up Napari:

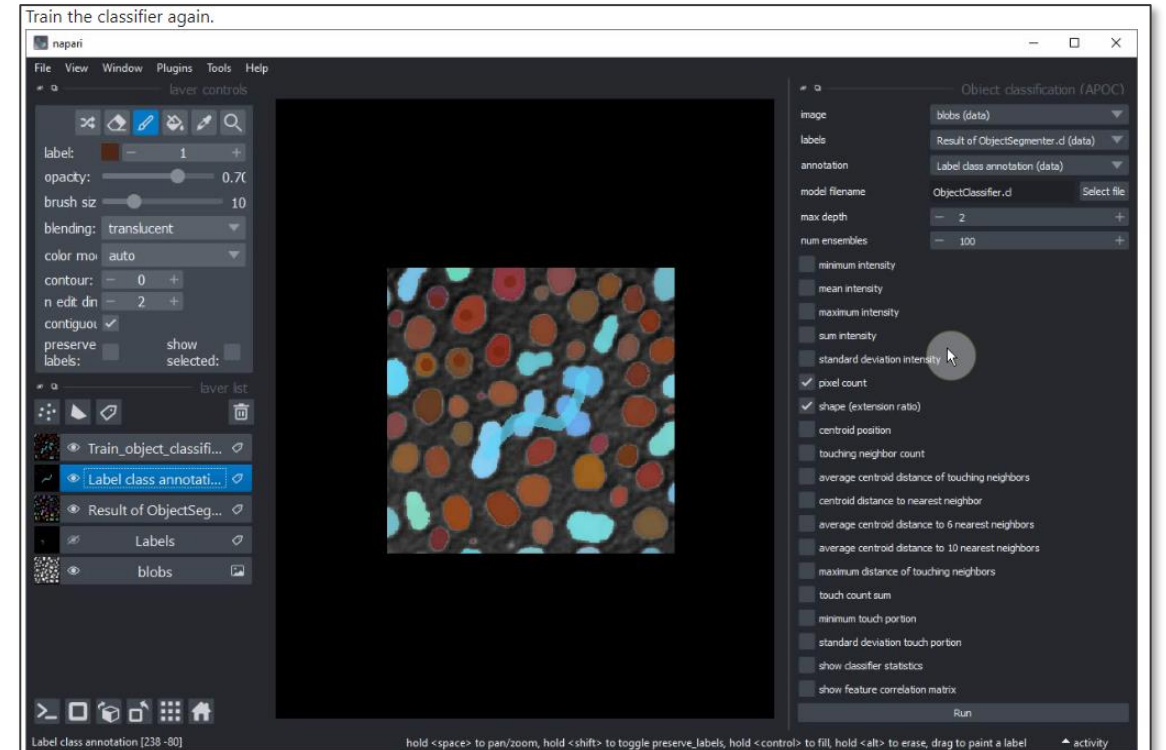
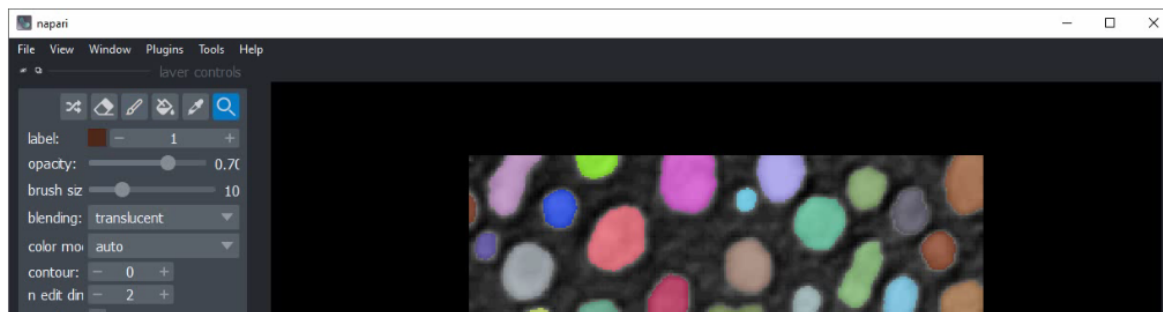
```
napari
```

Load the "Blobs" example dataset from the menu `File > Open Sample > c1Esperanto > Blobs (from ImageJ)`

We furthermore need a label image. You can create it using the pixel classifier trained earlier or using the menu `Tools > Segmentation / labeling > Gauss-Otsu Labeling (clesperanto)`.

Object classification

Our starting point is a loaded image and a label image with segmented objects. The following procedure is also shown in [this video](#).



If you are happy with the trained classifier, copy the file to a safe place. When training the next classifier this one might be overwritten.

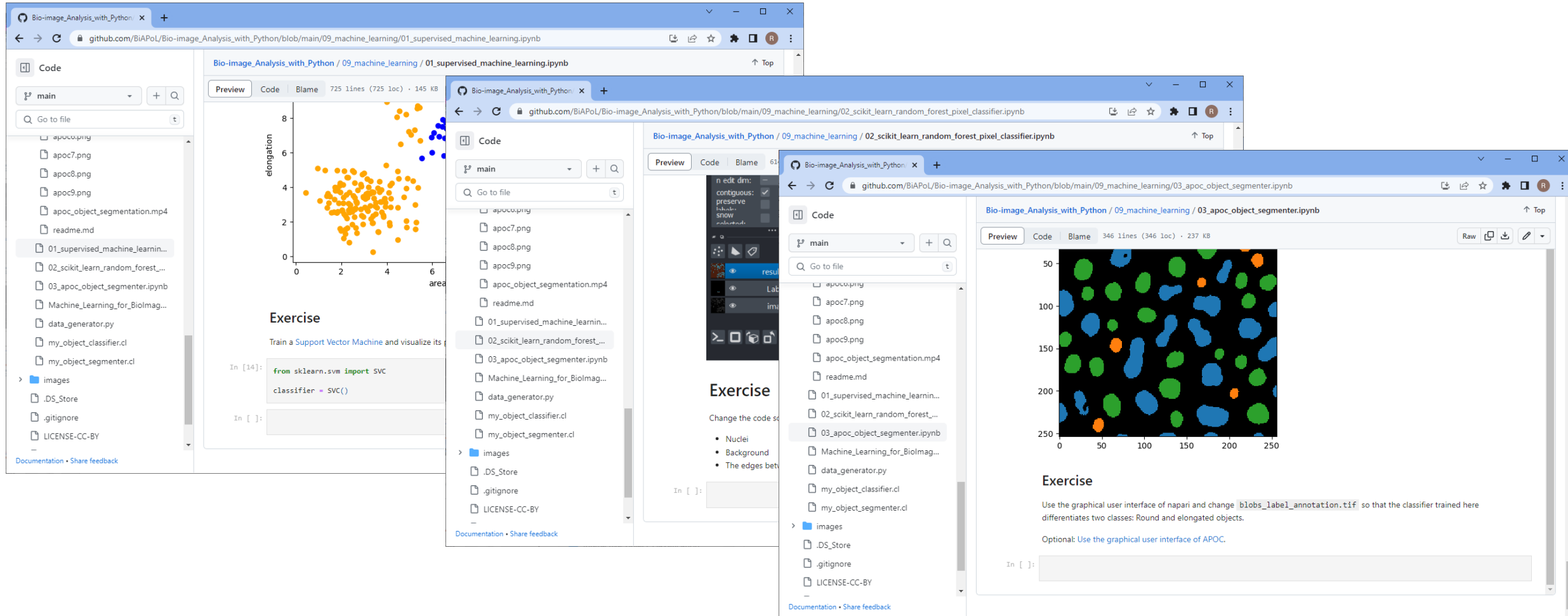
Extra exercise

Retrain the classifier so that it can differentiate three different classes:

- Small round objects
- Large round objects
- Large elongated objects

https://github.com/BiAPoL/Bio-image_Analysis_with_Python/blob/main/09_machine_learning/interactive_object_classification/readme.md

- Use scikit-learn and apoc in Jupyter Notebooks to train and apply Random Forest Classifiers and Support Vector Machines



The image displays three overlapping Jupyter Notebook windows from a GitHub repository. The leftmost window shows a scatter plot titled '01_supervised_machine_learning.ipynb' with 'elongation' on the y-axis and 'area' on the x-axis. The data points are colored orange and blue. Below the plot is an 'Exercise' section with the instruction: 'Train a Support Vector Machine and visualize its decision boundary'. The code cell shows:

```
In [14]: from sklearn.svm import SVC  
classifier = SVC()
```

```
In [ ]:
```

The middle window shows a code editor for '02_scikit_learn_random_forest_pixel_classifier.ipynb'. It features a file browser on the left and a code cell with the instruction: 'Change the code so that the classifier differentiates between: • Nuclei • Background • The edges between objects'. The code cell contains:

```
In [ ]:
```

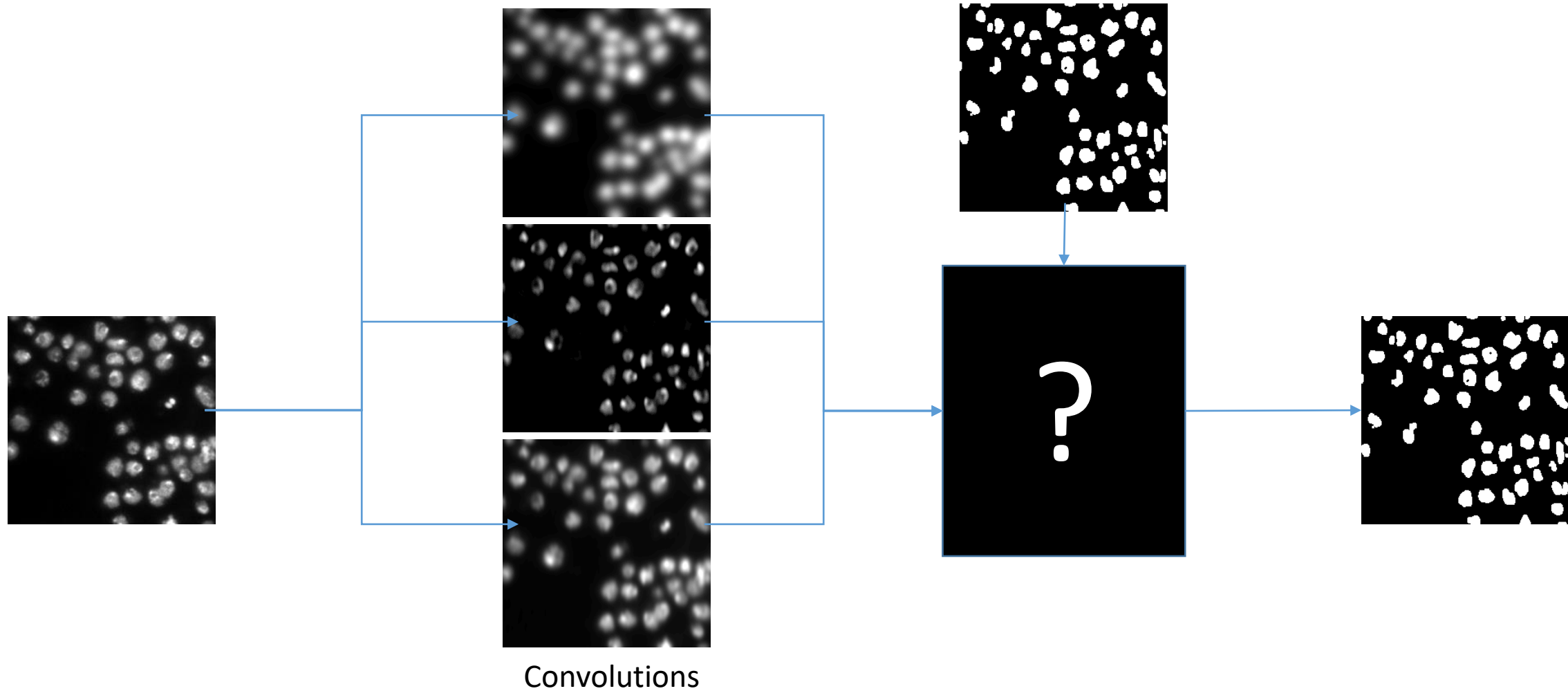
The rightmost window shows '03_apoc_object_segmenter.ipynb'. It displays a segmented image with green and blue objects on a black background. Below the image is an 'Exercise' section: 'Use the graphical user interface of napari and change blobs_label1_annotation.tif so that the classifier trained here differentiates two classes: Round and elongated objects. Optional: Use the graphical user interface of APOC.' The code cell contains:

```
In [ ]:
```

<https://github.com/BiAPoL/Bio-image Analysis with Python/blob/main/09 machine learning/>

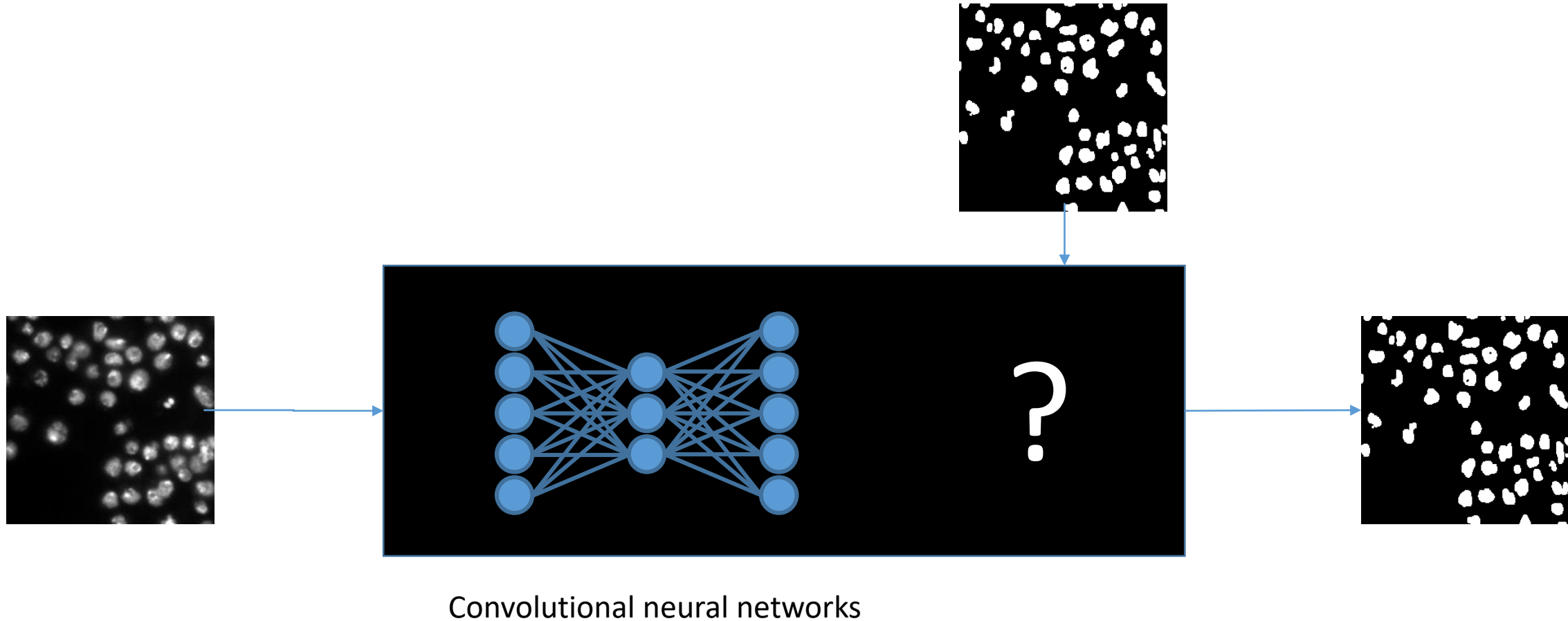
Summary & outlook

- In classical machine learning, we typically select features for training our classifier



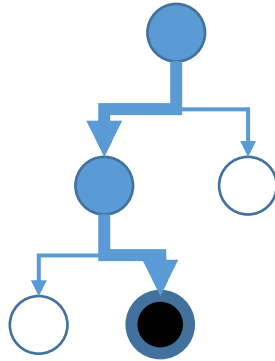
Outlook: Deep learning for image analysis

- In deep learning, this selection becomes part of the black box



Today, you learned

- Machine learning for Pixel and Object segmentation
- Python
 - Scikit-learn / napari
 - Accelerated pixel and object classifiers (APOC)



Coming up next:

- Unsupervised machine learning
- Deep learning

