

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)

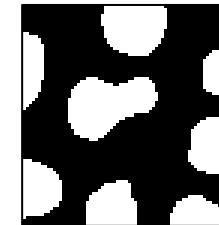
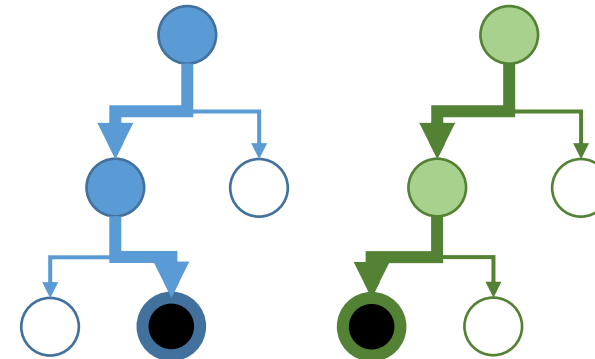
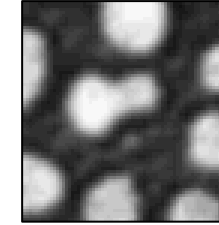
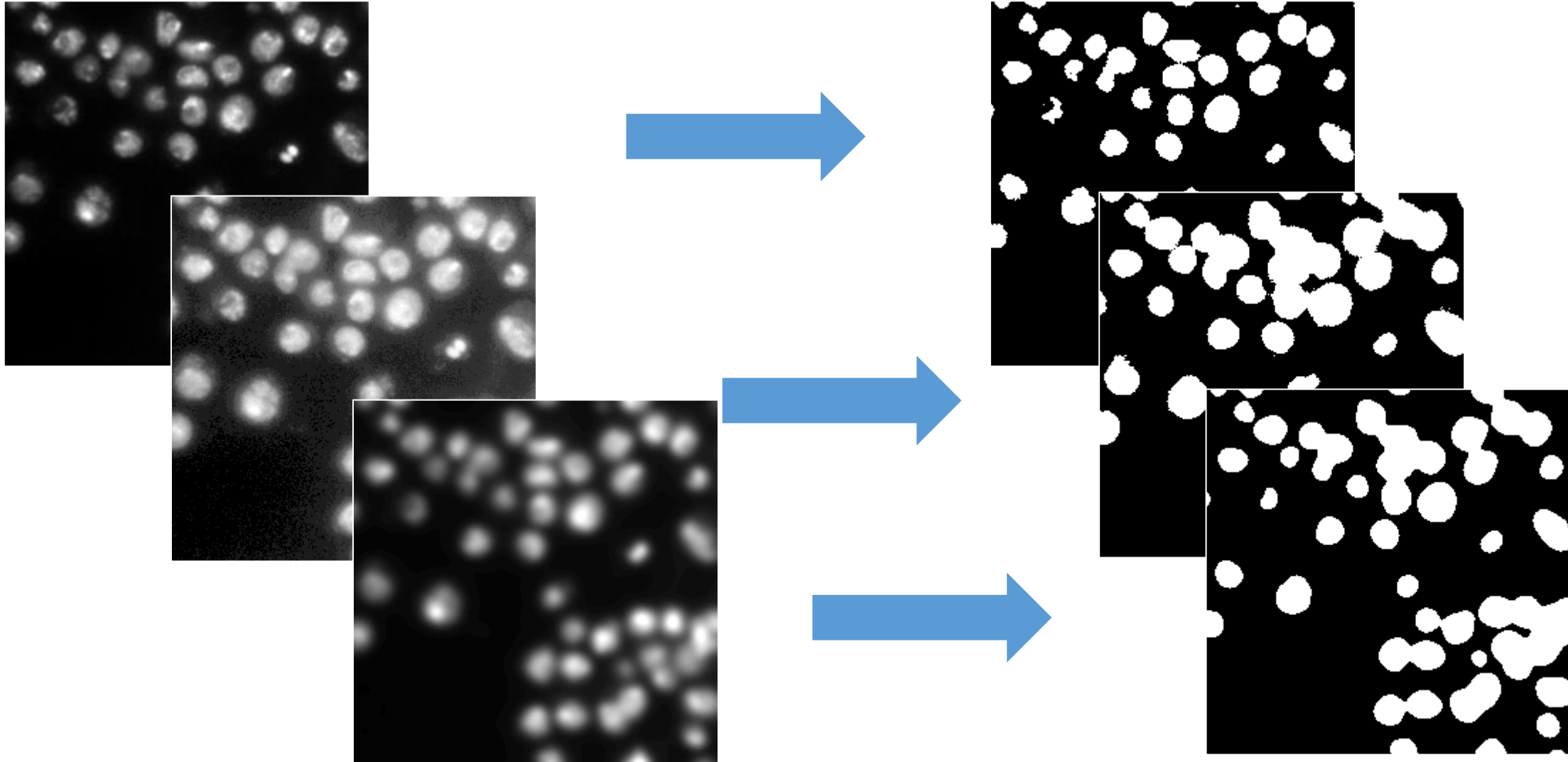
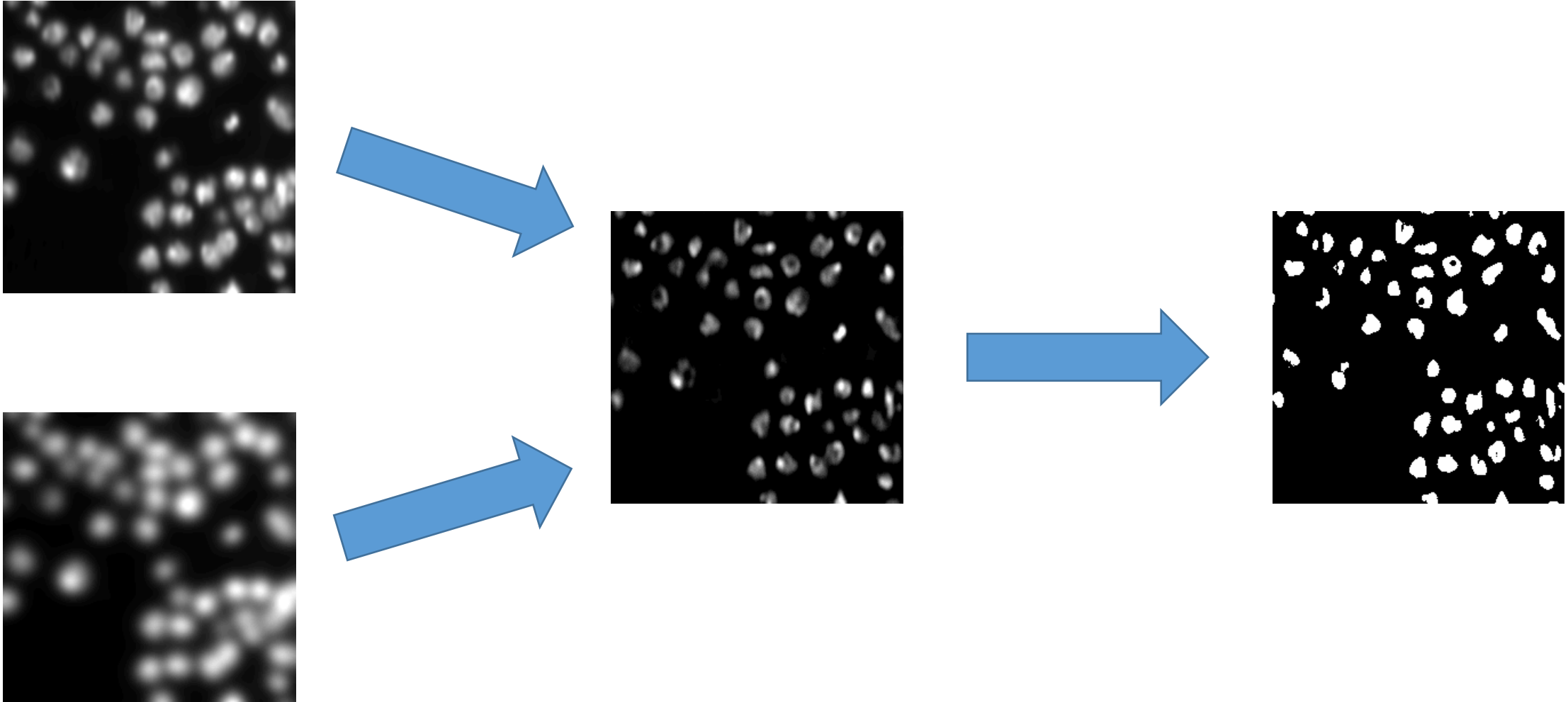


Image segmentation using thresholding

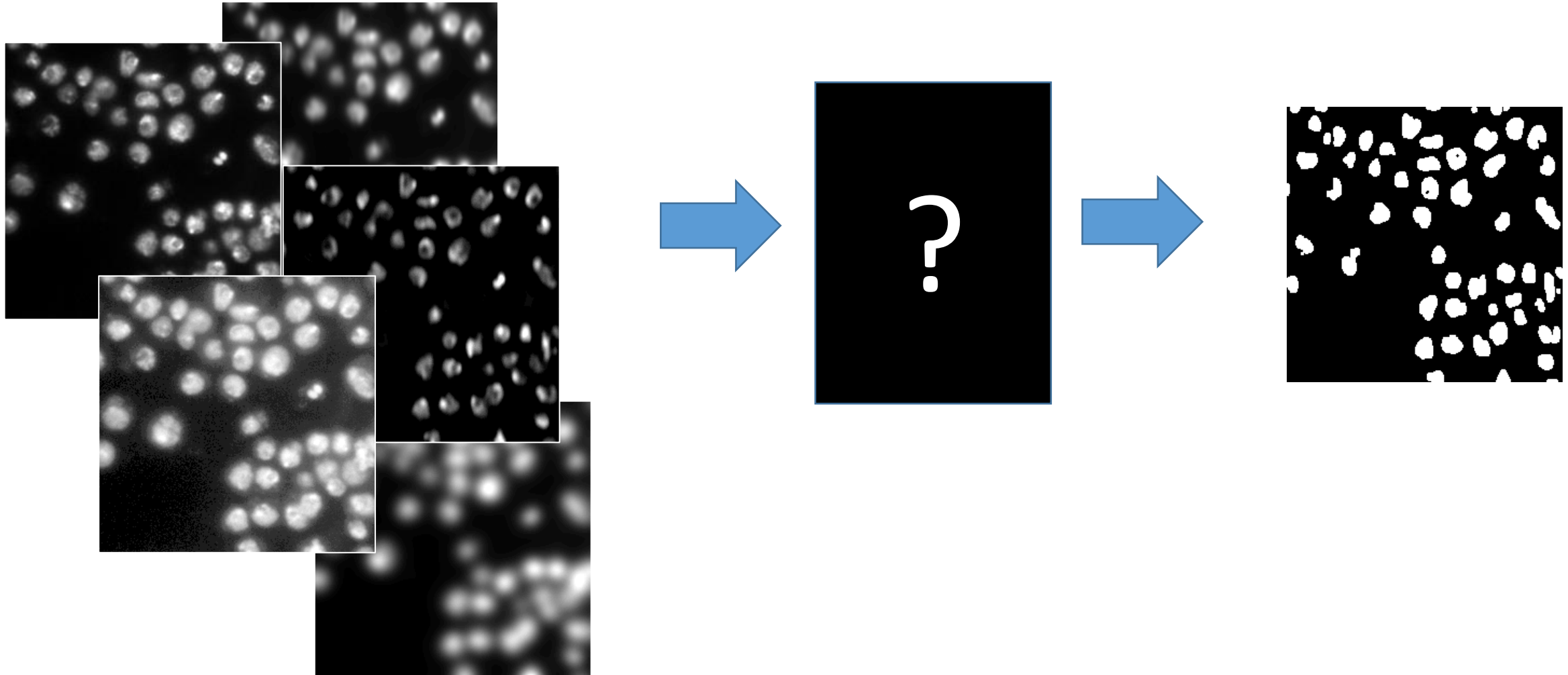
- Recap: Finding the right workflow towards a good segmentation takes time



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



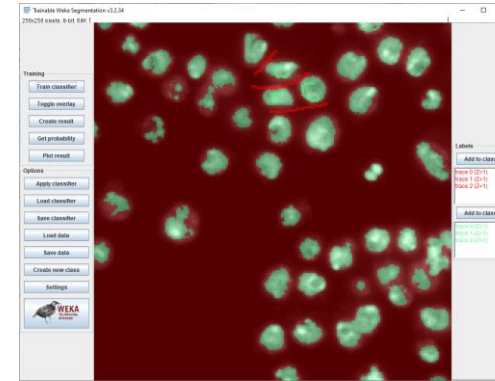
- Might there be a technology for optimization which combination of images can be used to get the best segmentation result?



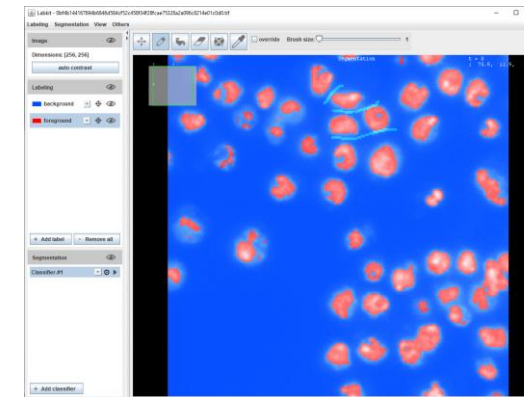
- A research field in computer science
- Finds more and more applications, also in life sciences.

Artificial intelligence

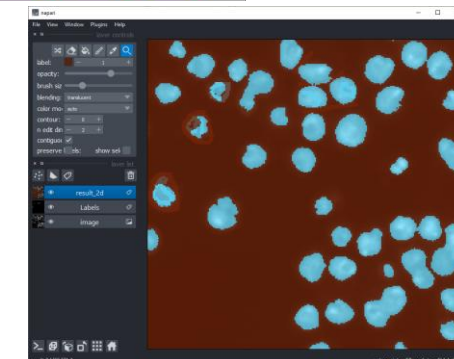
Machine learning



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

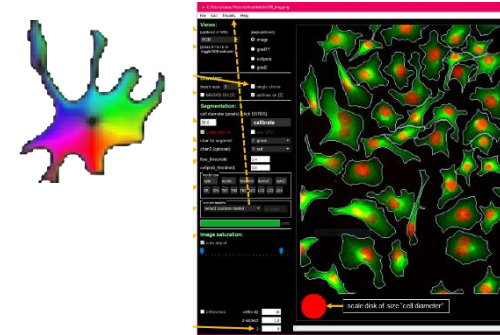
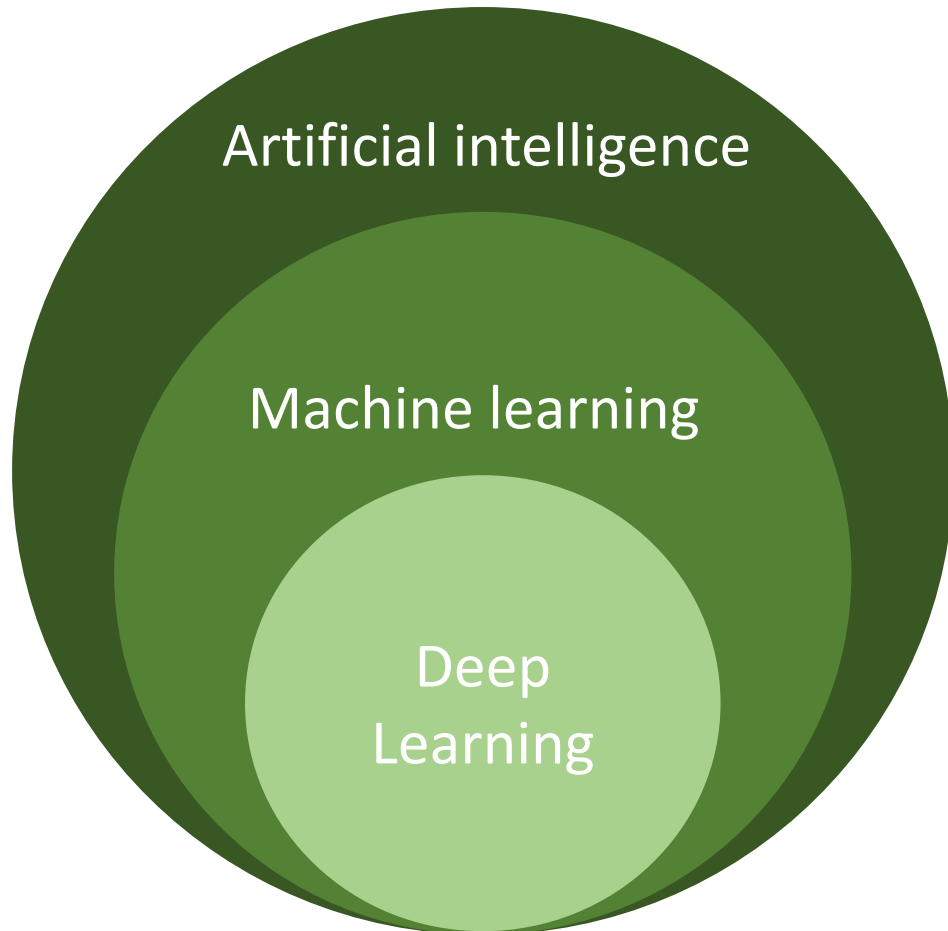


LabKit
<https://imagej.net/plugins/labkit/>

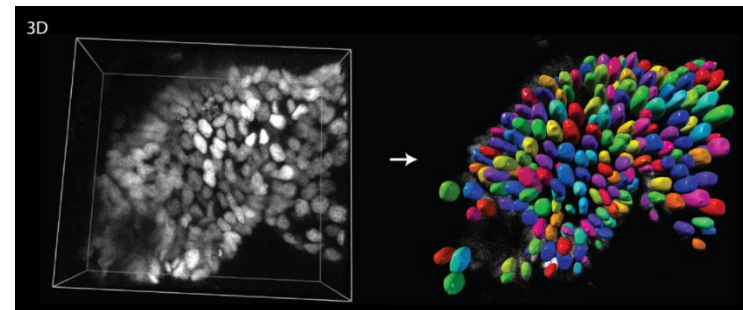


Python /
scikit-learn /
napari /
apoc

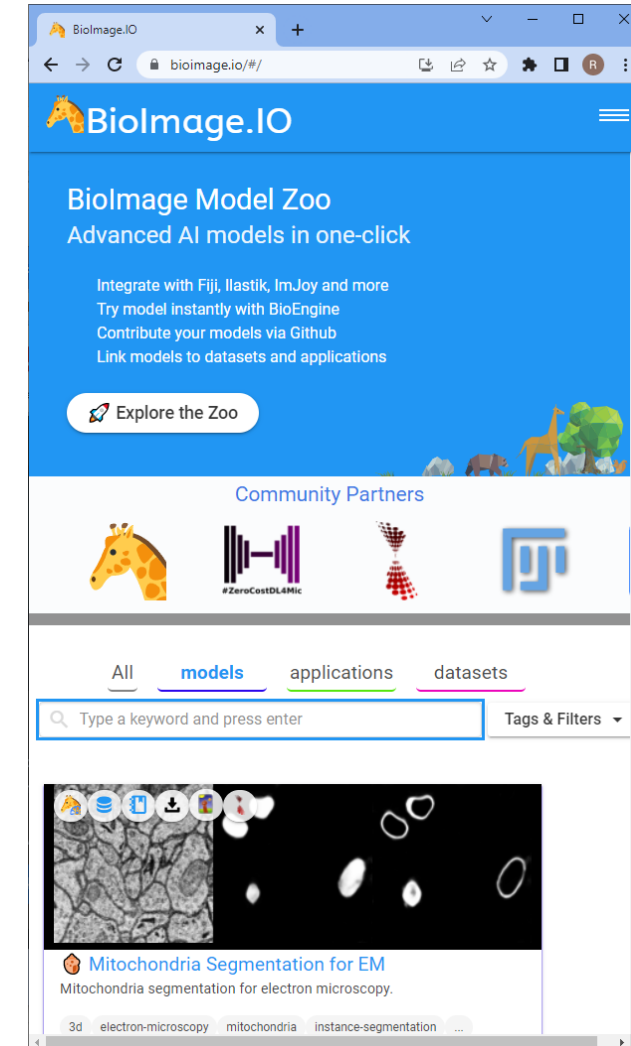
- A research field in computer science
- Finds more and more applications, also in life sciences.



www.cellpose.org/

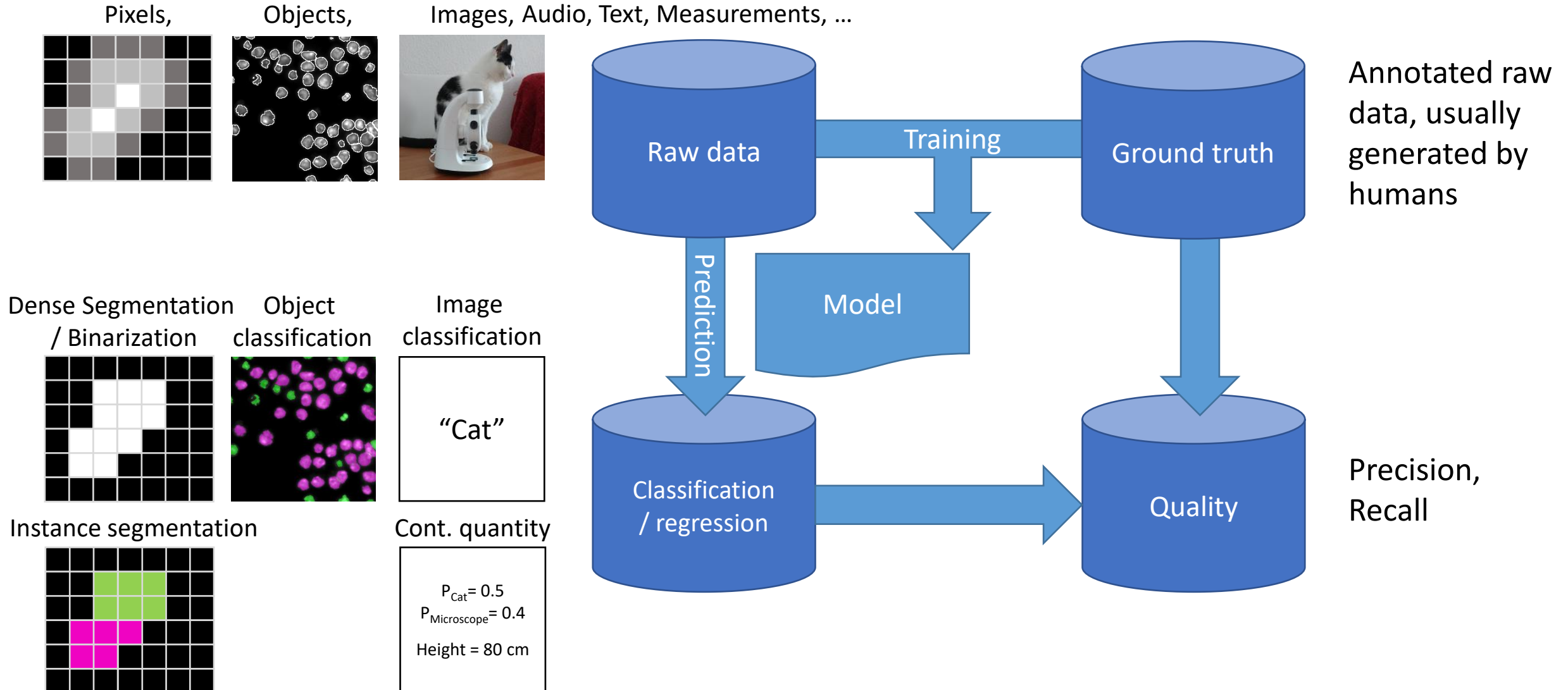


<https://github.com/stardist/stardist>

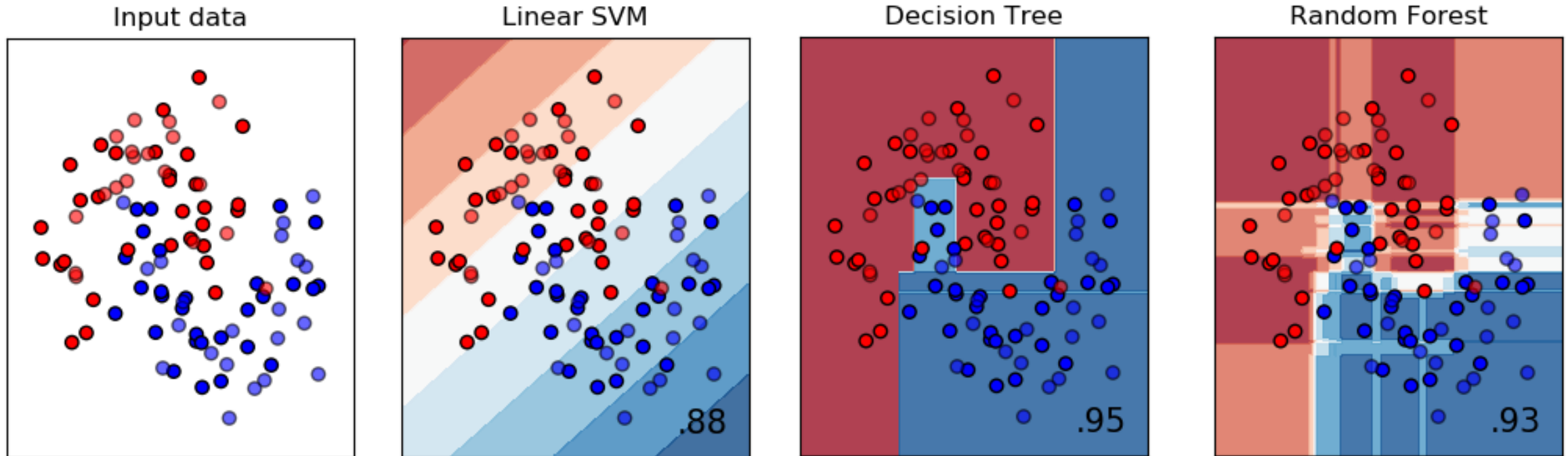


<https://bioimage.io/>

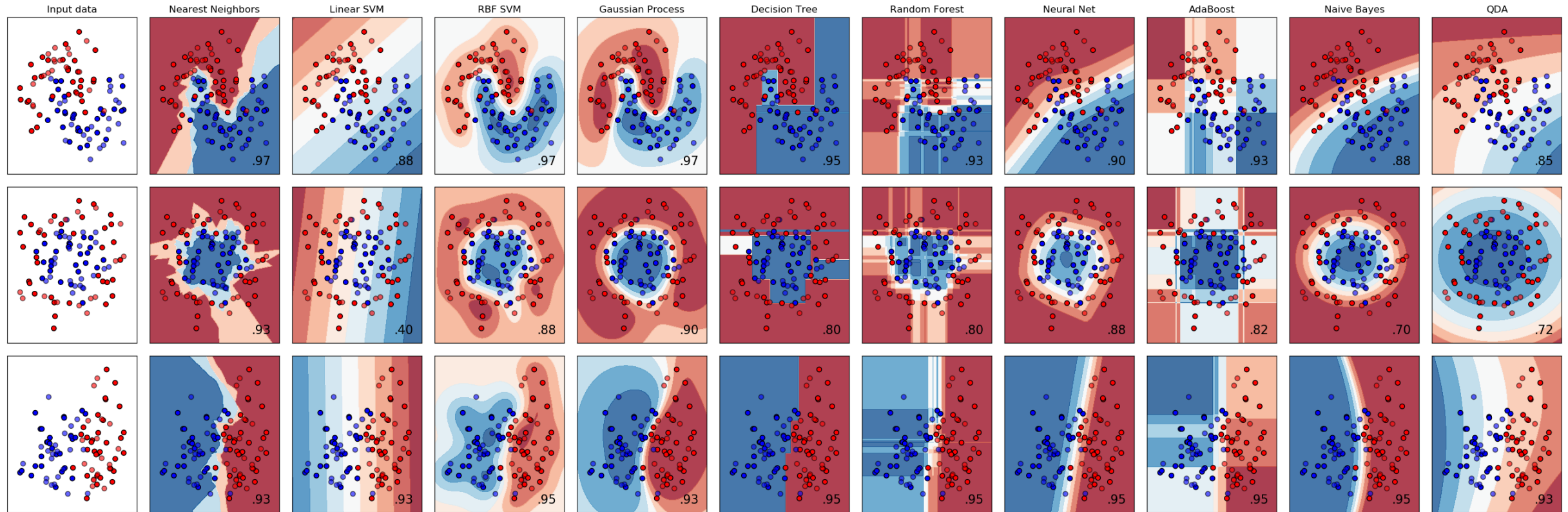
- Automatic construction of predictive models from given data



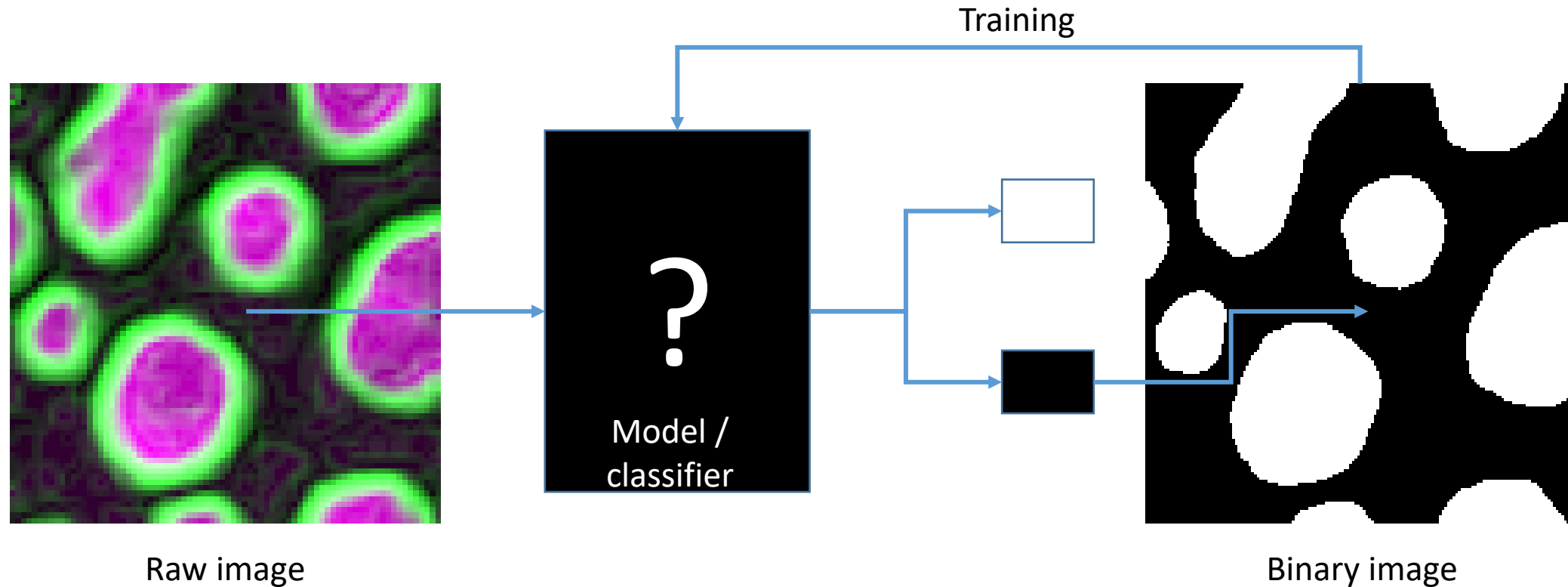
- Guess classification (color) from position of a sample in parameter space.



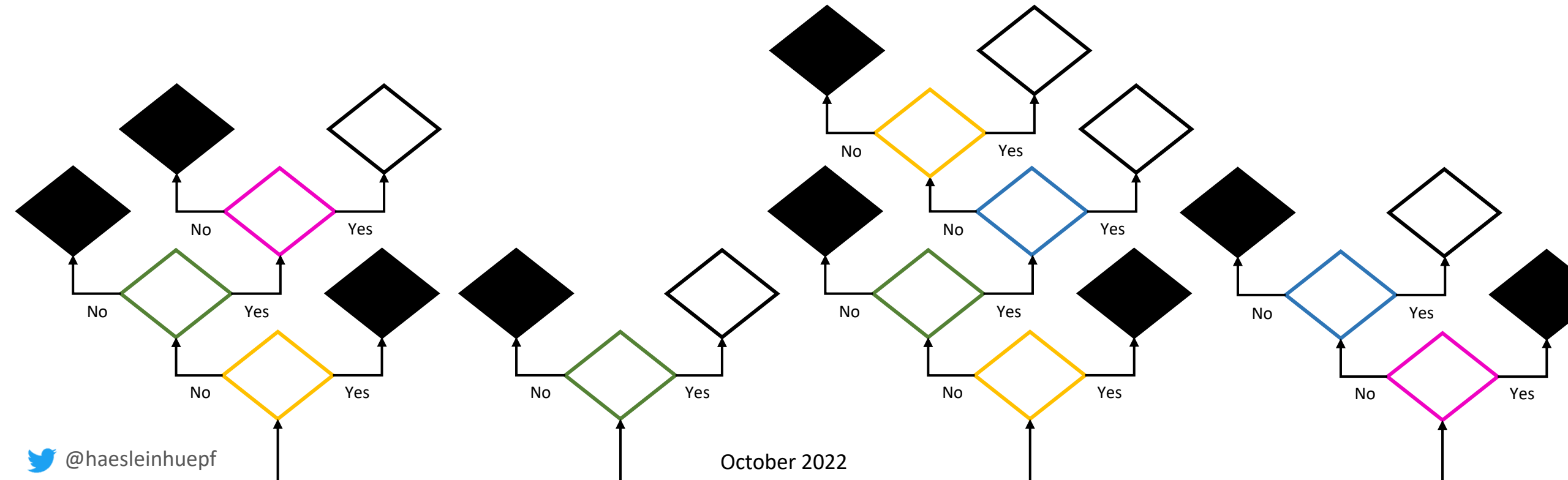
- The right approach depends on data, computational resources and desired quality



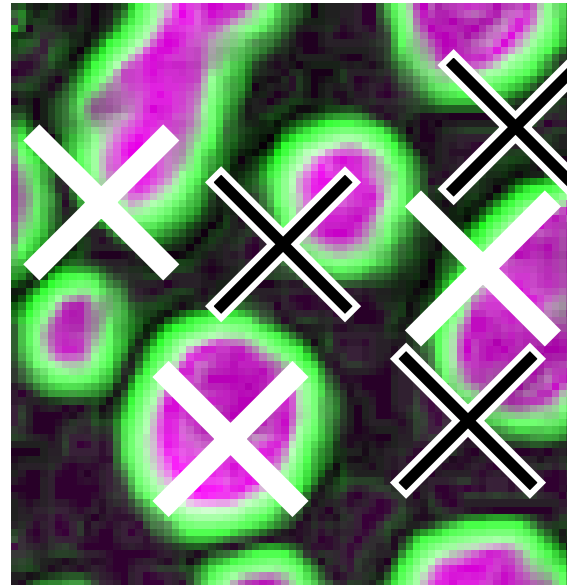
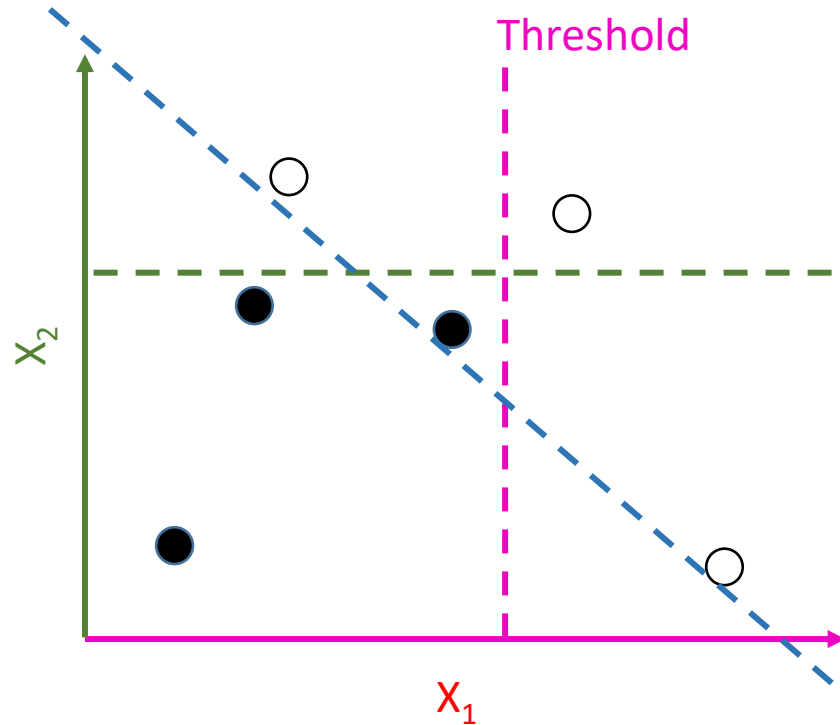
- *Supervised* machine learning: We give the computer some ground truth to learn from
- The computer derives a *model* or a *classifier* which can judge if a pixel should be foreground (white) or background (black)
- Example: Binary classifier



- 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

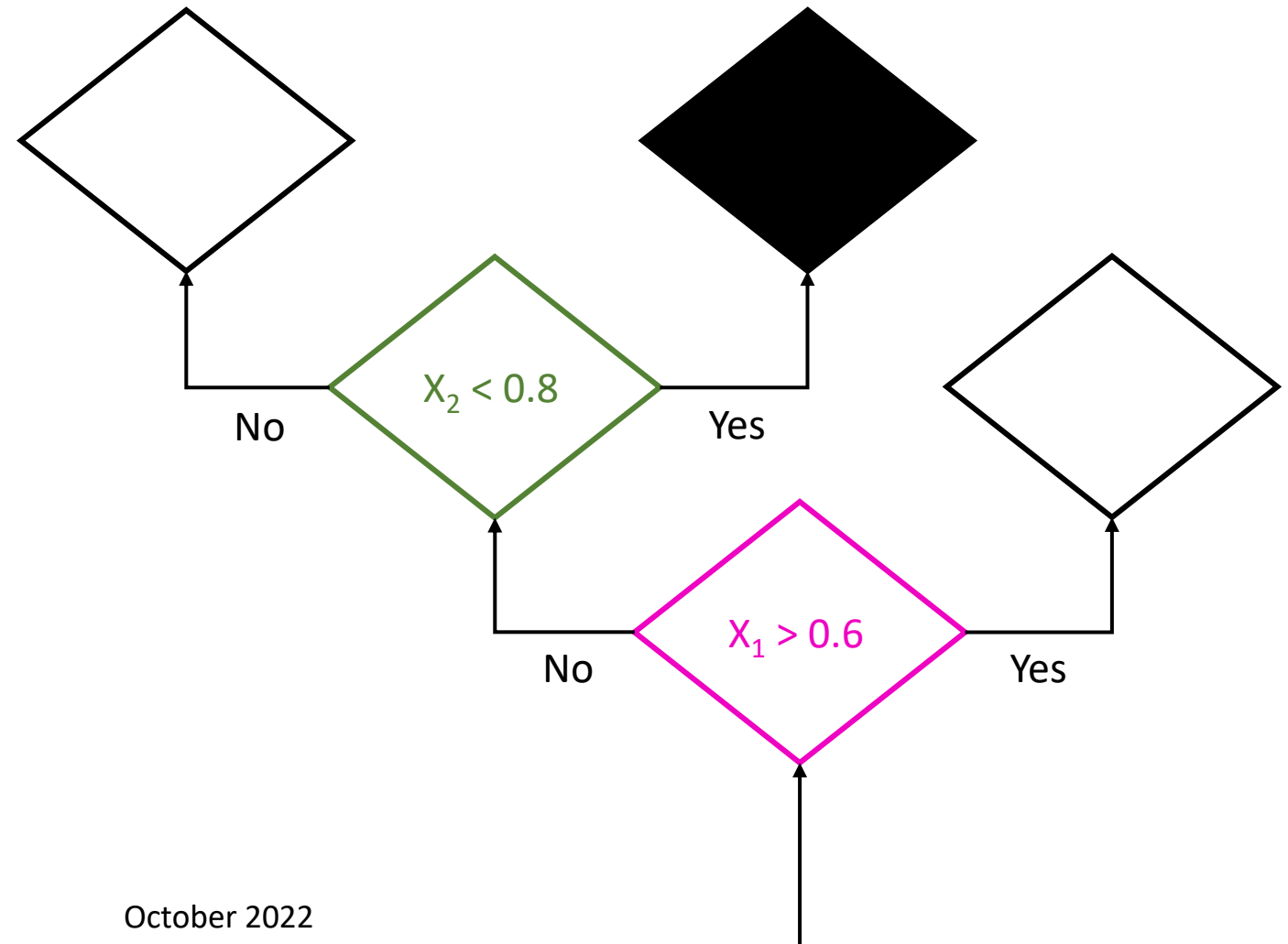
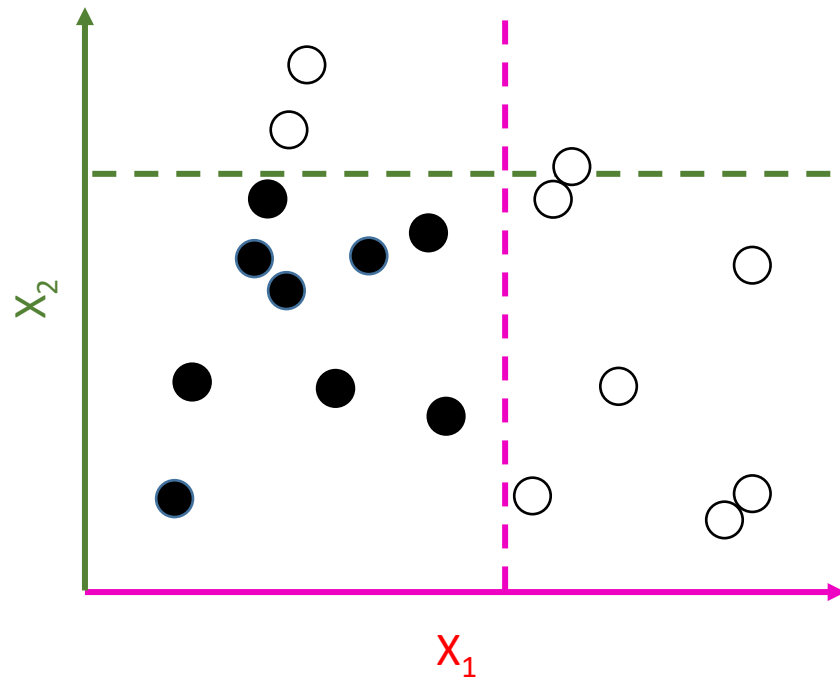


- For efficient processing, we randomly *sample* our data set
 - Individual pixels, their intensity and their classification



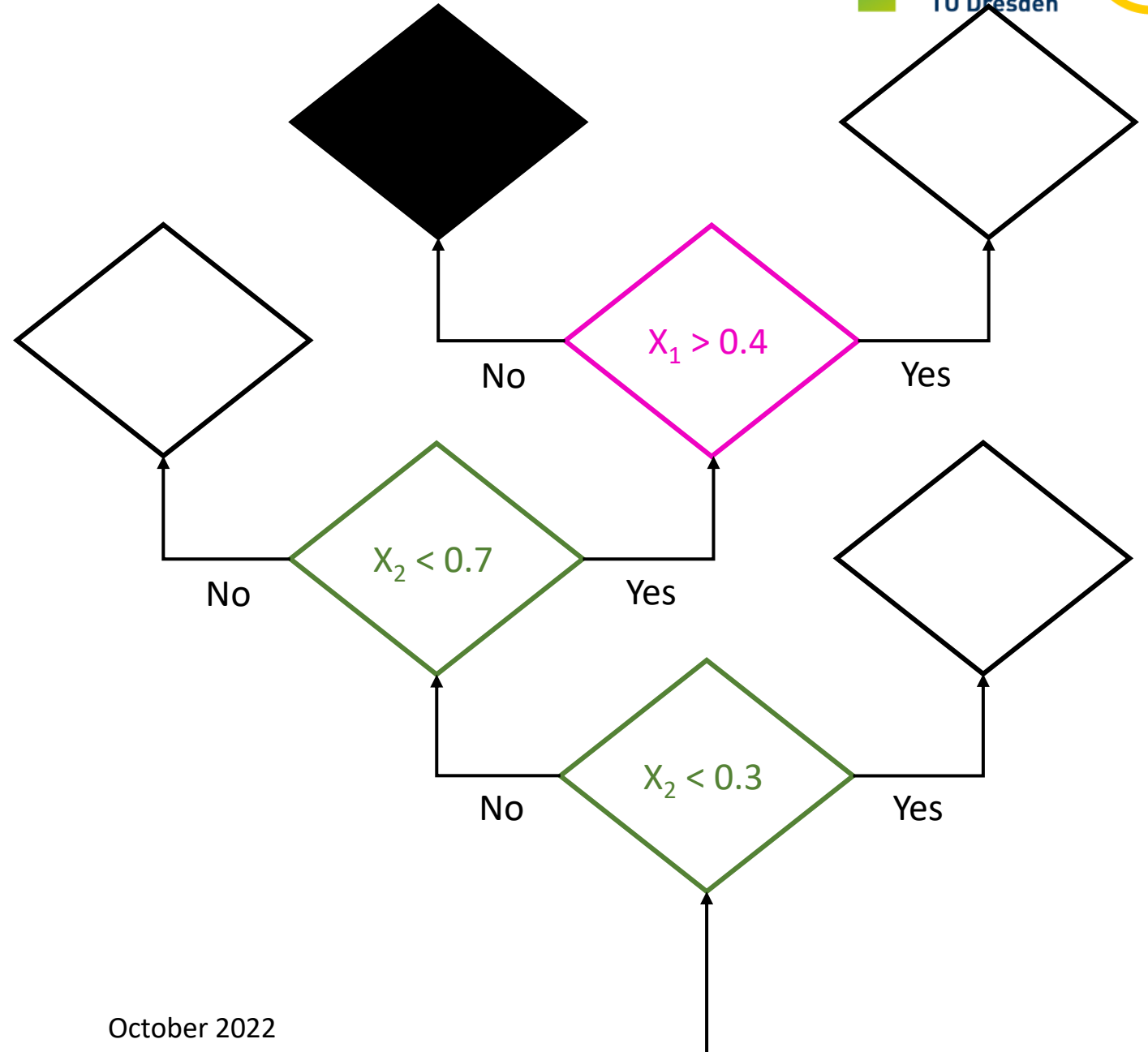
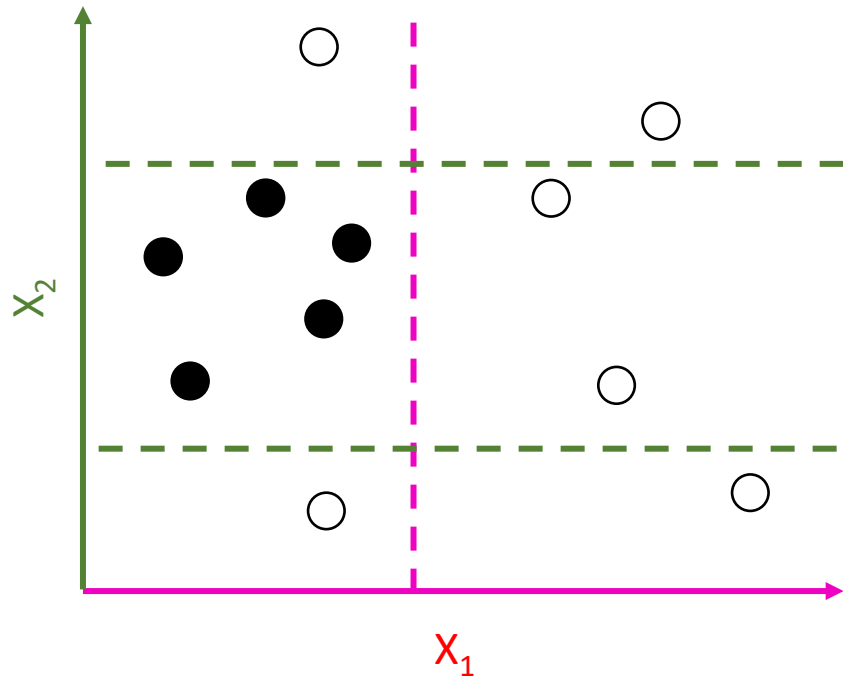
Note: You cannot use a single threshold to make the decision correctly

- Decision trees combine several thresholds on several parameters

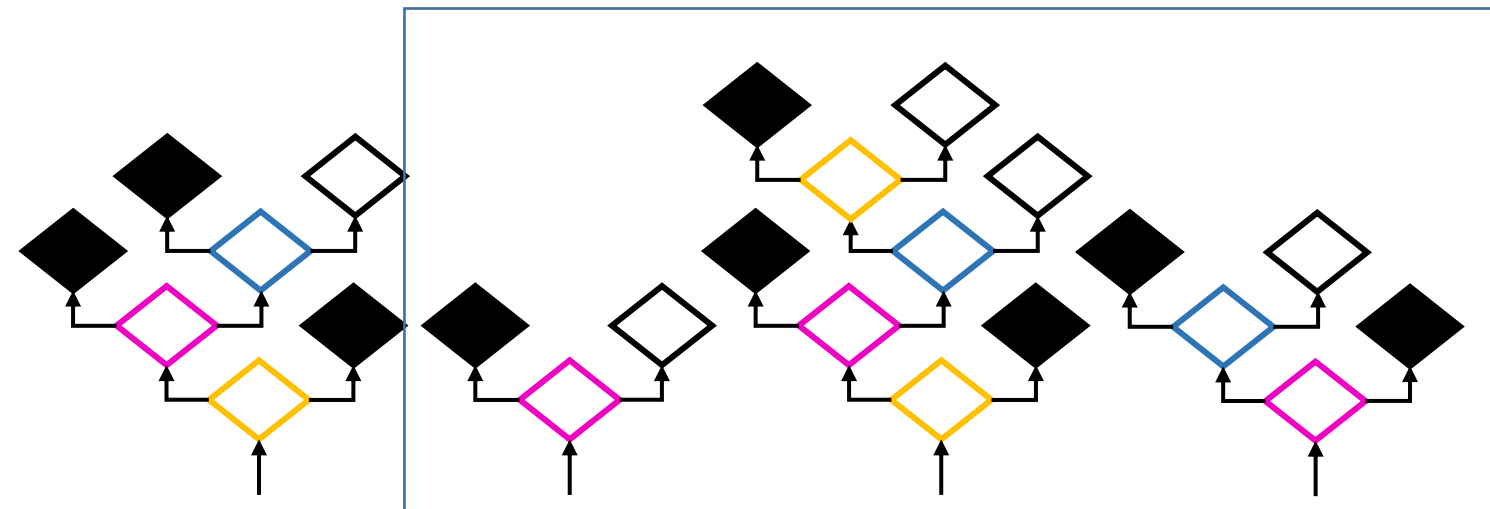
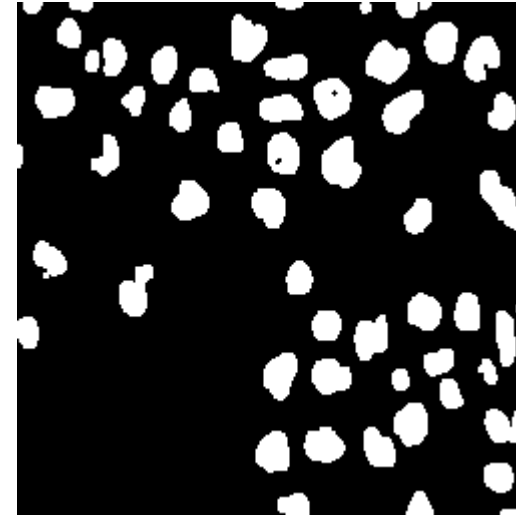
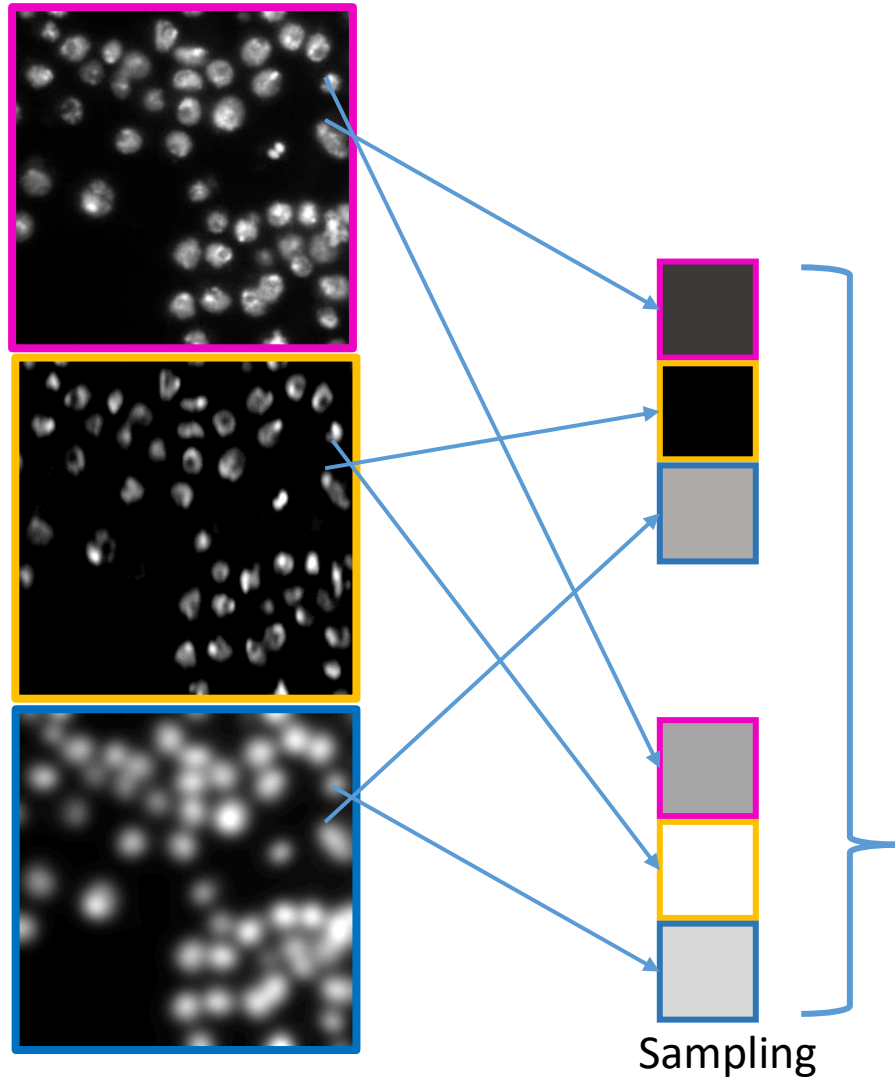


Deriving random decision trees

- Depending on sampling, the decision trees are different

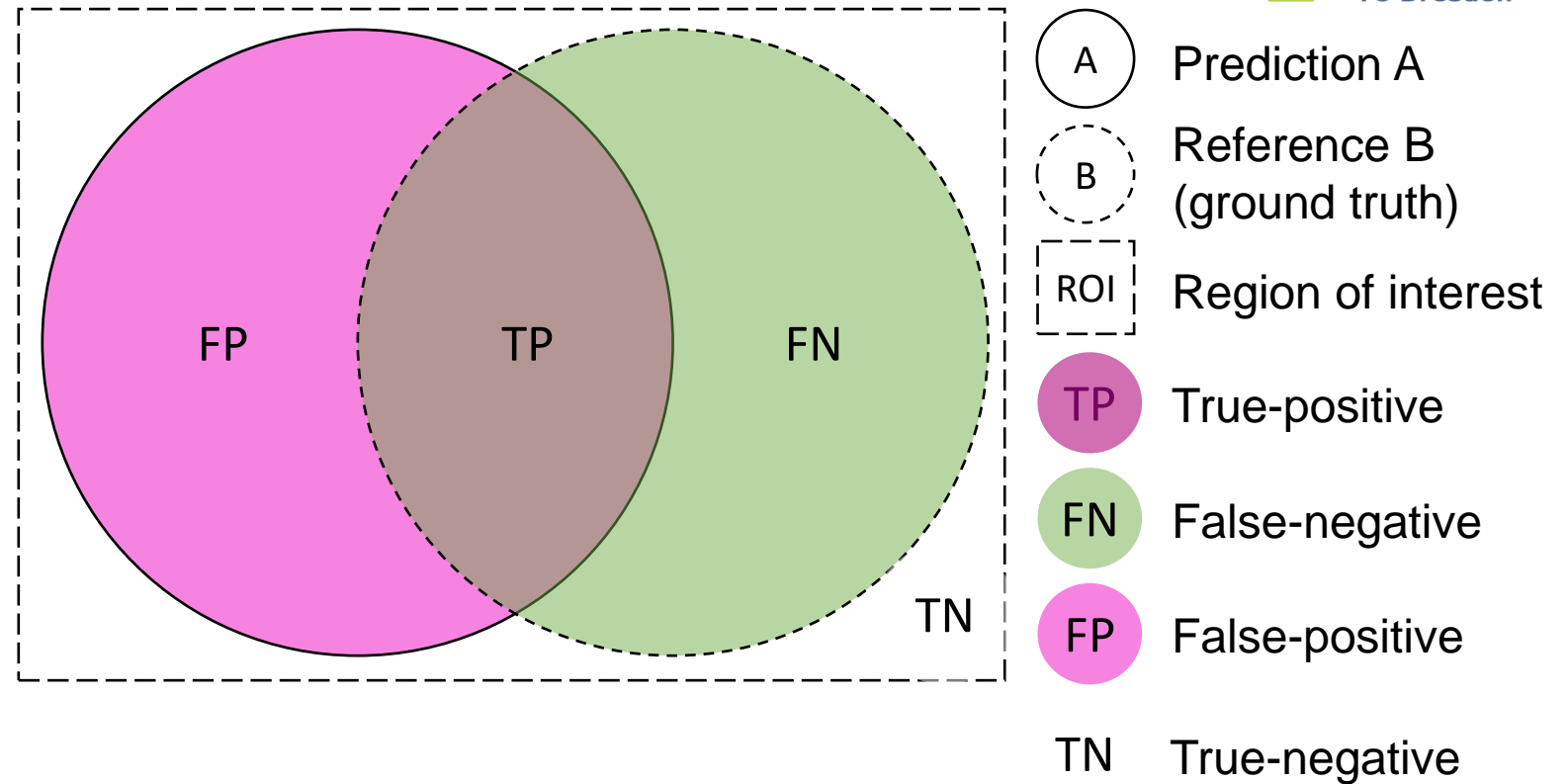


- By comparing performance of individual decision trees, good ones can be selected, bad ones excluded.



Recap: Algorithm evaluation

- 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



Precision

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

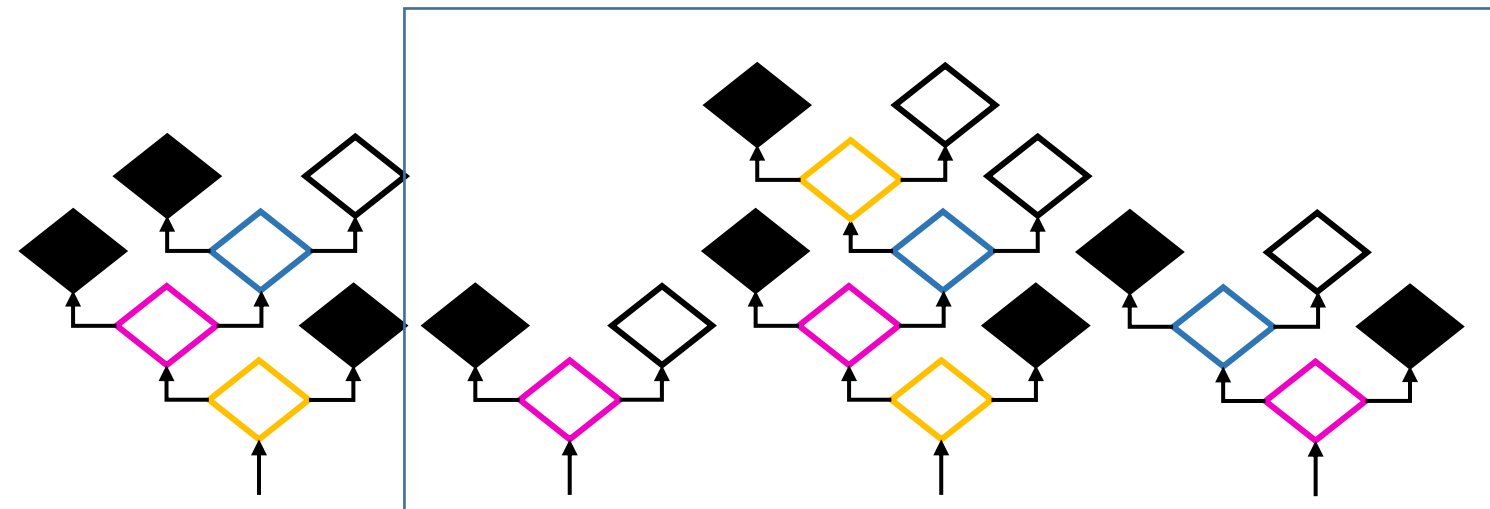
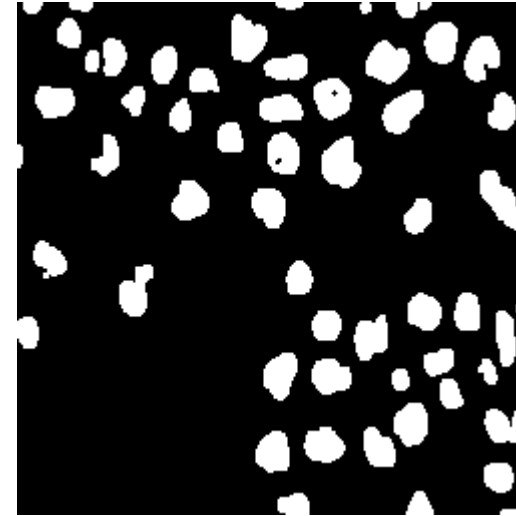
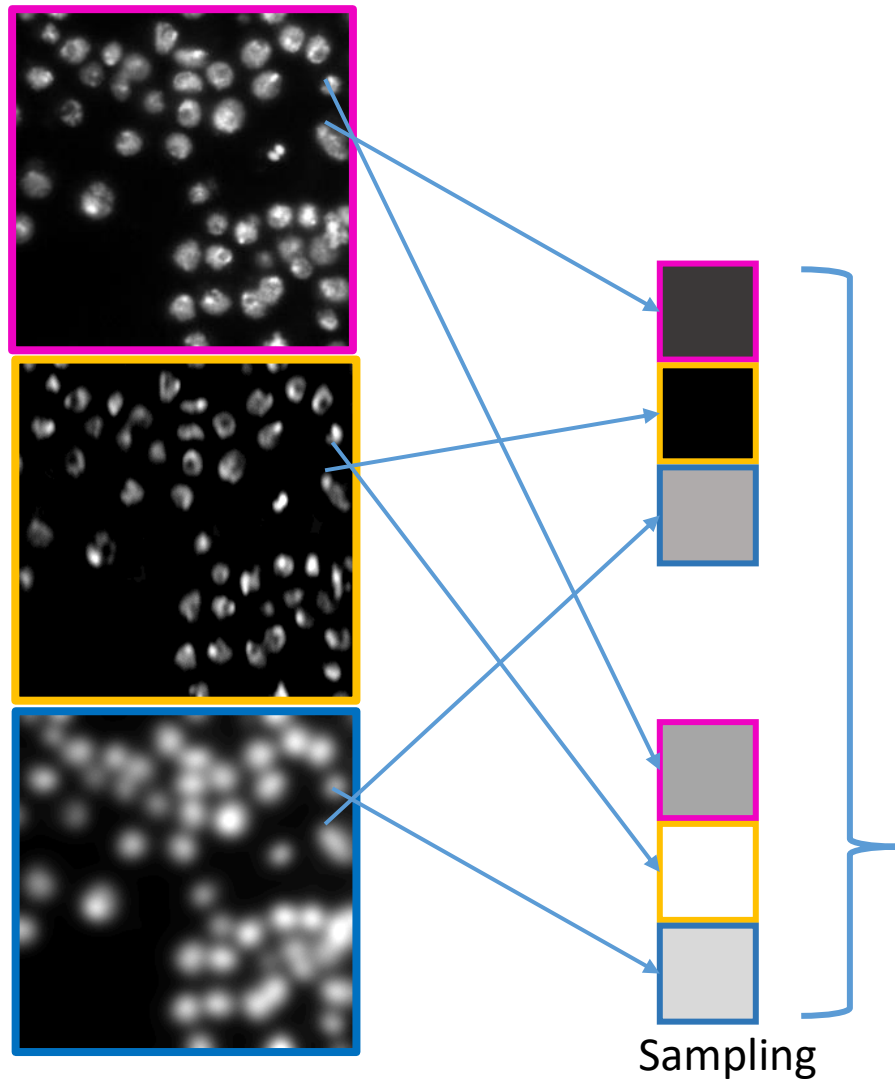
What fraction of points that were predicted as positives were really positive?

Recall
(a.k.a. sensitivity)

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

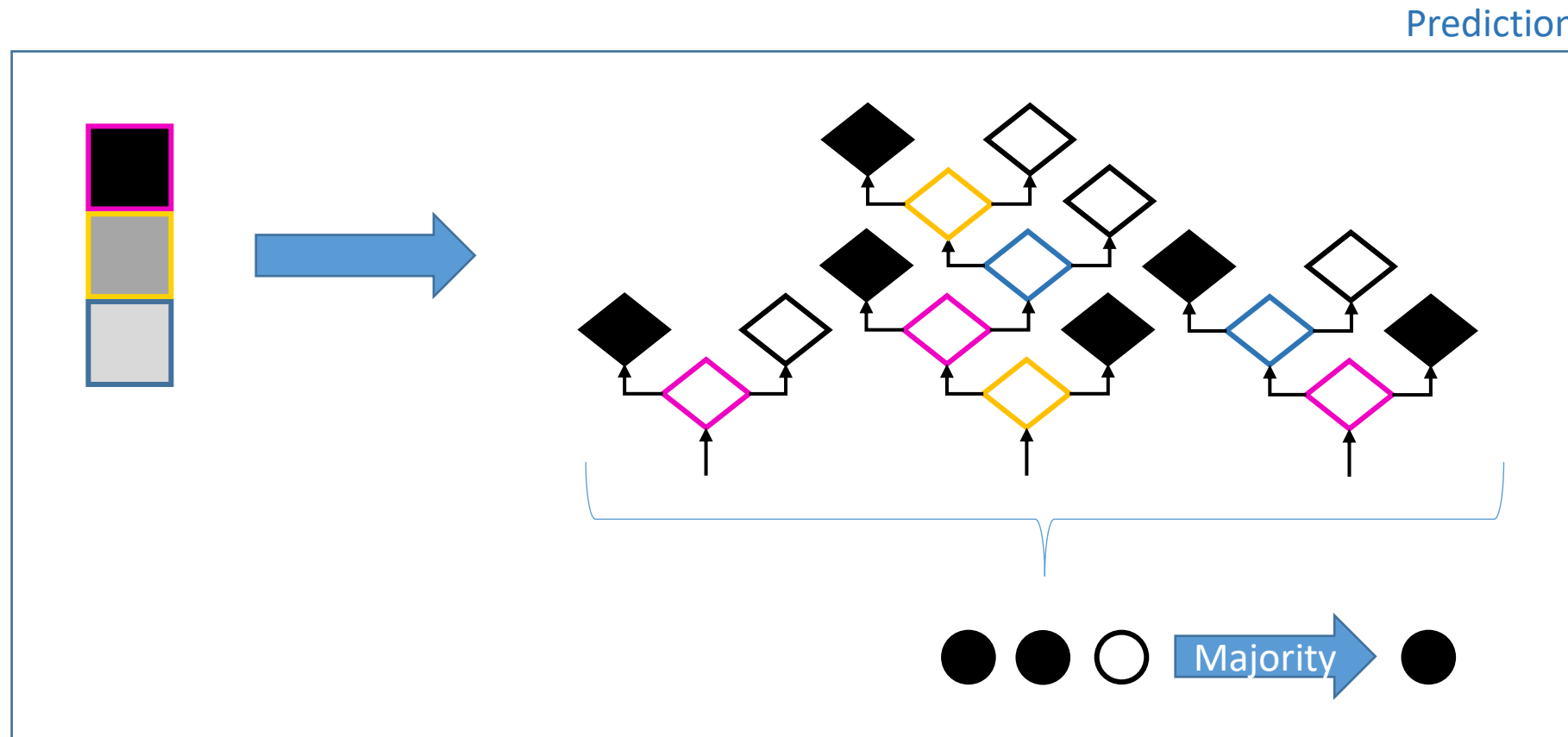
What fraction of positives points were predicted as positives?

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



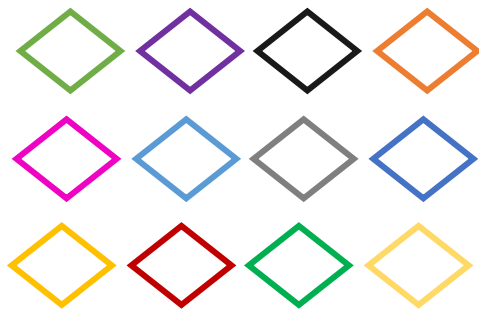
October 2022

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



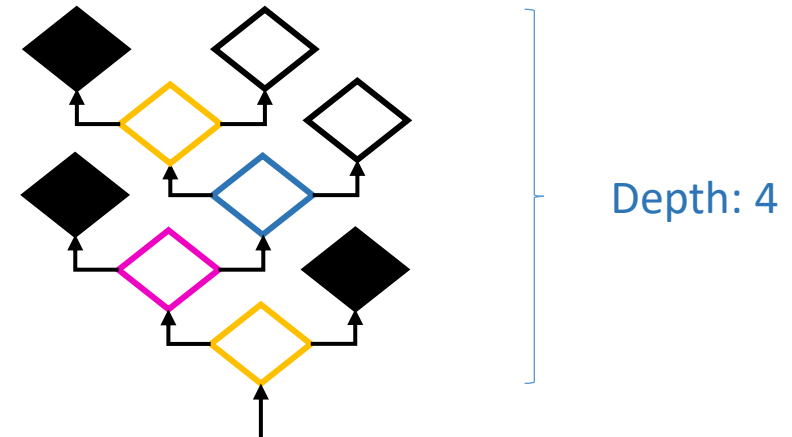
- Typical numbers for pixel classifiers in microscopy

Available features: > 20

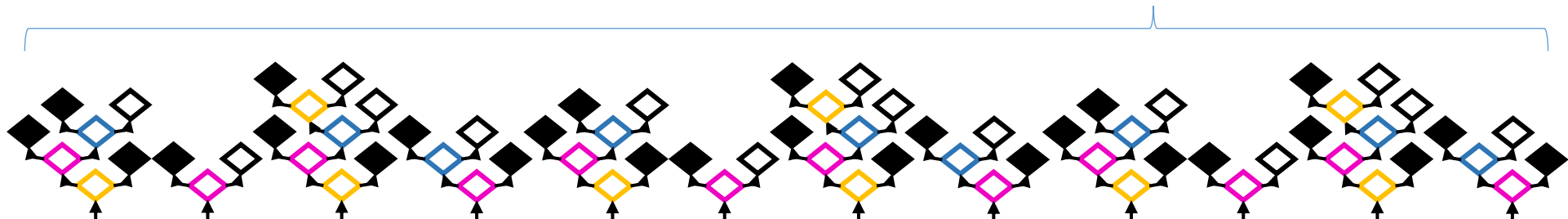


- Gaussian blur image
- DoG image
- LoG image
- Hessian
-

Selected features: $\leq \text{depth}$

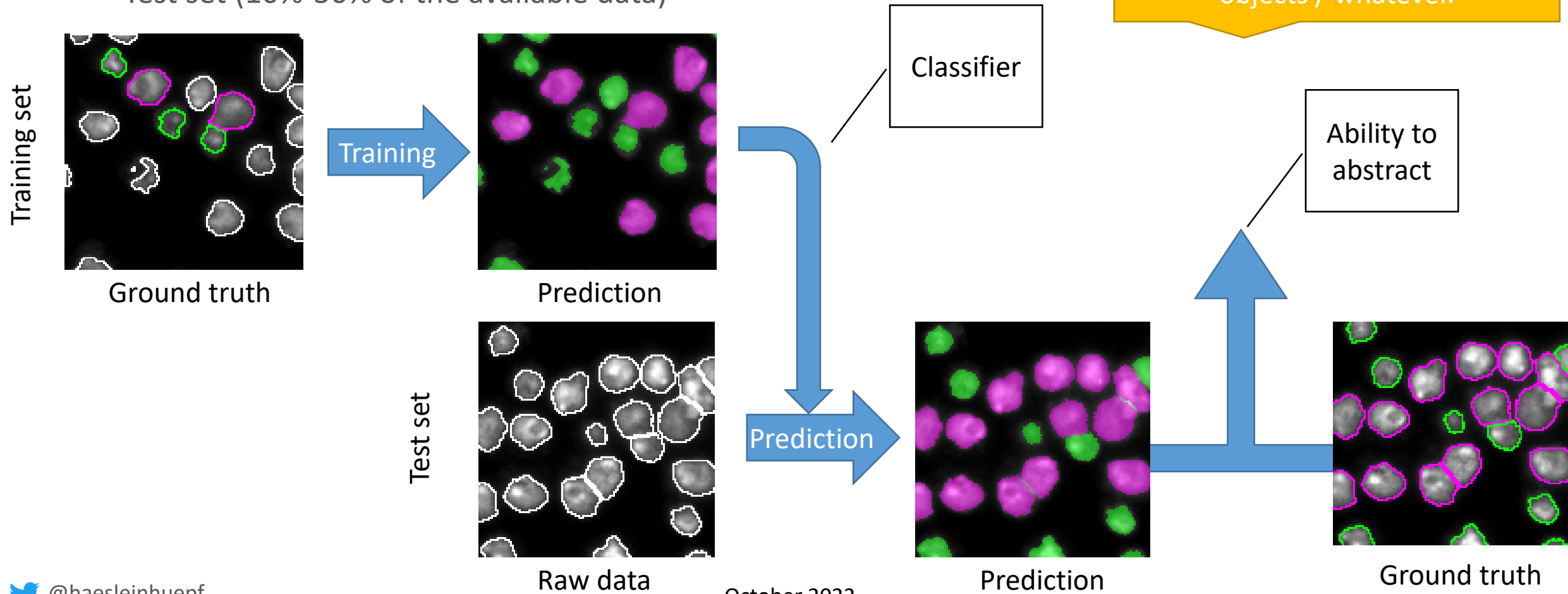


Number of trees: > 100



- 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

- 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)





Pixel classification using scikit-learn

Robert Haase

October 2022

- 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

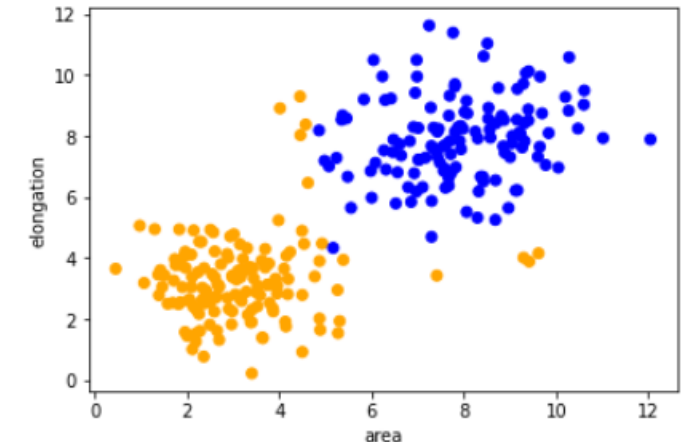
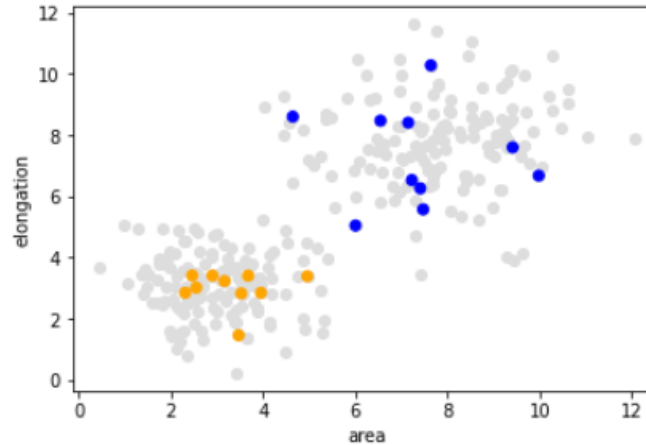
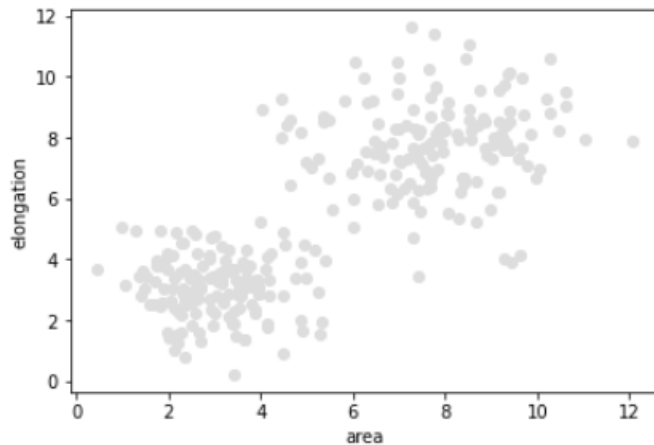
```
annotation = [1, 1, 2, 1, 1, 1, 2, 2,
```

Classifier training

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

Classifier prediction

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



Interactive pixel classification

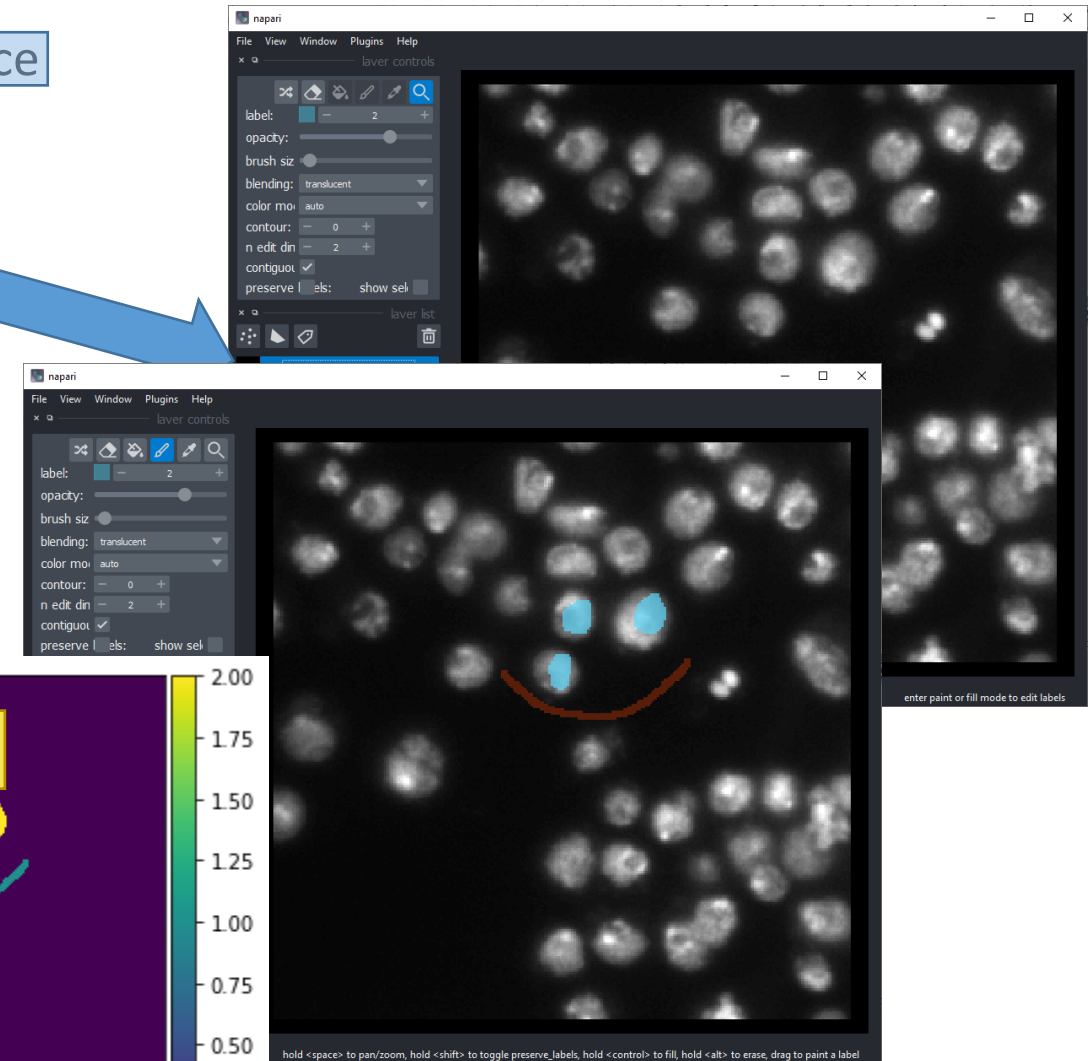
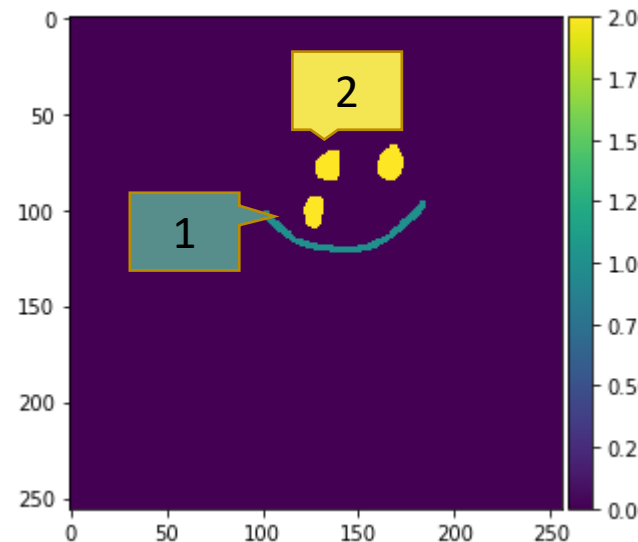
- Prepare an empty layer for annotations and keep a **reference**

```
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)
```

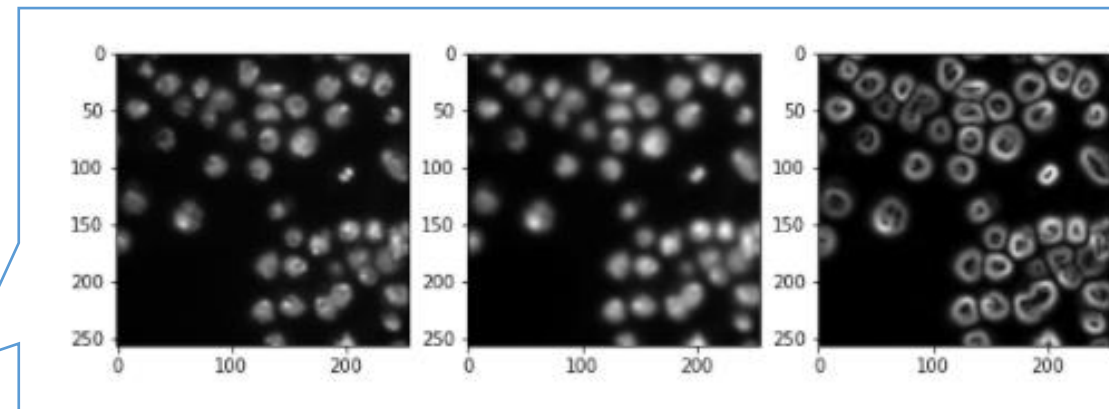
```
X, y = format_data(feature_stack, manual_annotations)
```

```
# train classifier
```

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

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

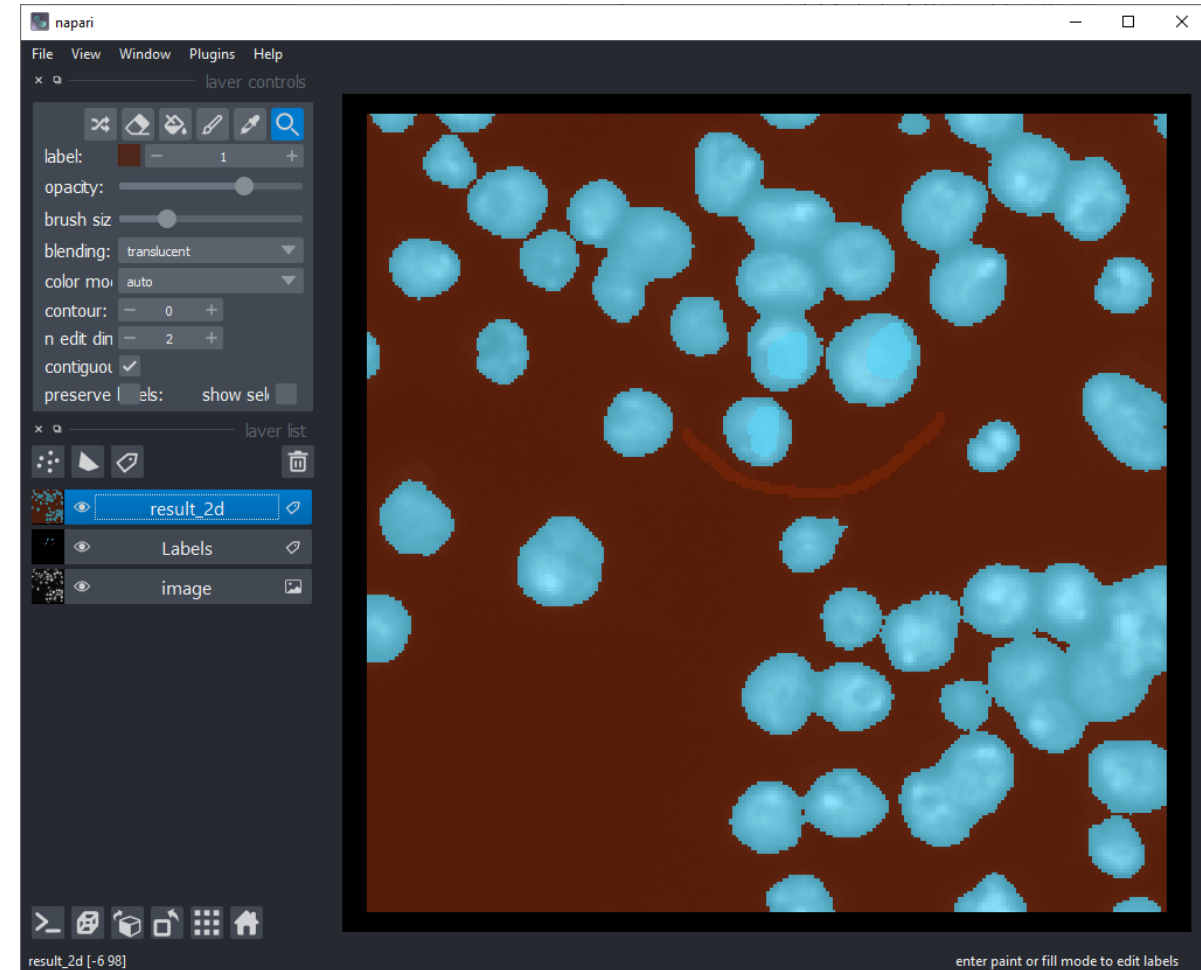
```
classifier.fit(X, y)
```



- Pixel classification using scikit-learn

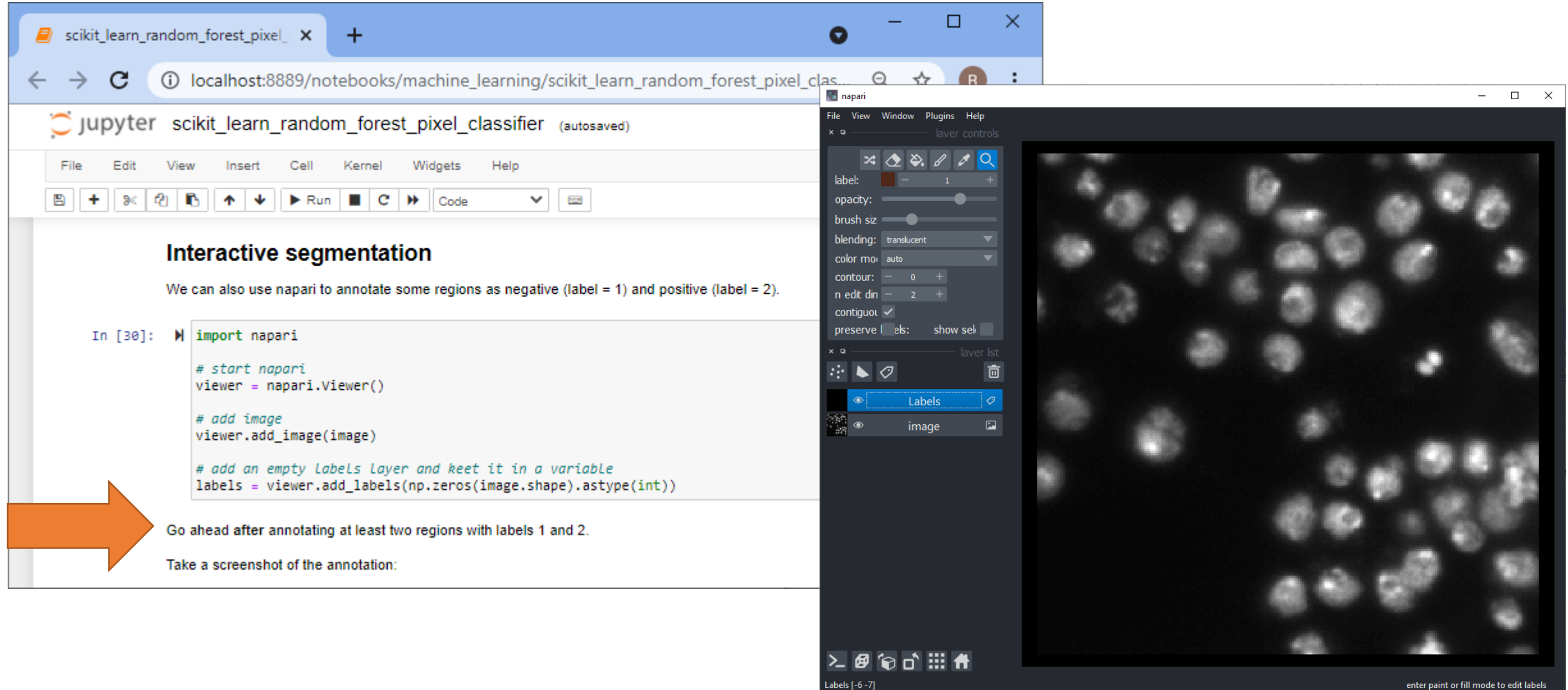
```
# process the whole image and show result
result_1d = classifier.predict(feature_stack.T)
result_2d = result_1d.reshape(image.shape)

viewer.add_labels(result_2d)
```



Interactive pixel classification

- Jupyter notebooks and napari side-by-side



The screenshot shows a Jupyter notebook titled "scikit_learn_random_forest_pixel_classifier" (autosaved) running on localhost:8889. The notebook content includes the title "Interactive segmentation" and a text block stating: "We can also use napari to annotate some regions as negative (label = 1) and positive (label = 2)." Below this, a code cell shows the following Python code:

```
In [30]: import napari

# start napari
viewer = napari.Viewer()

# add image
viewer.add_image(image)

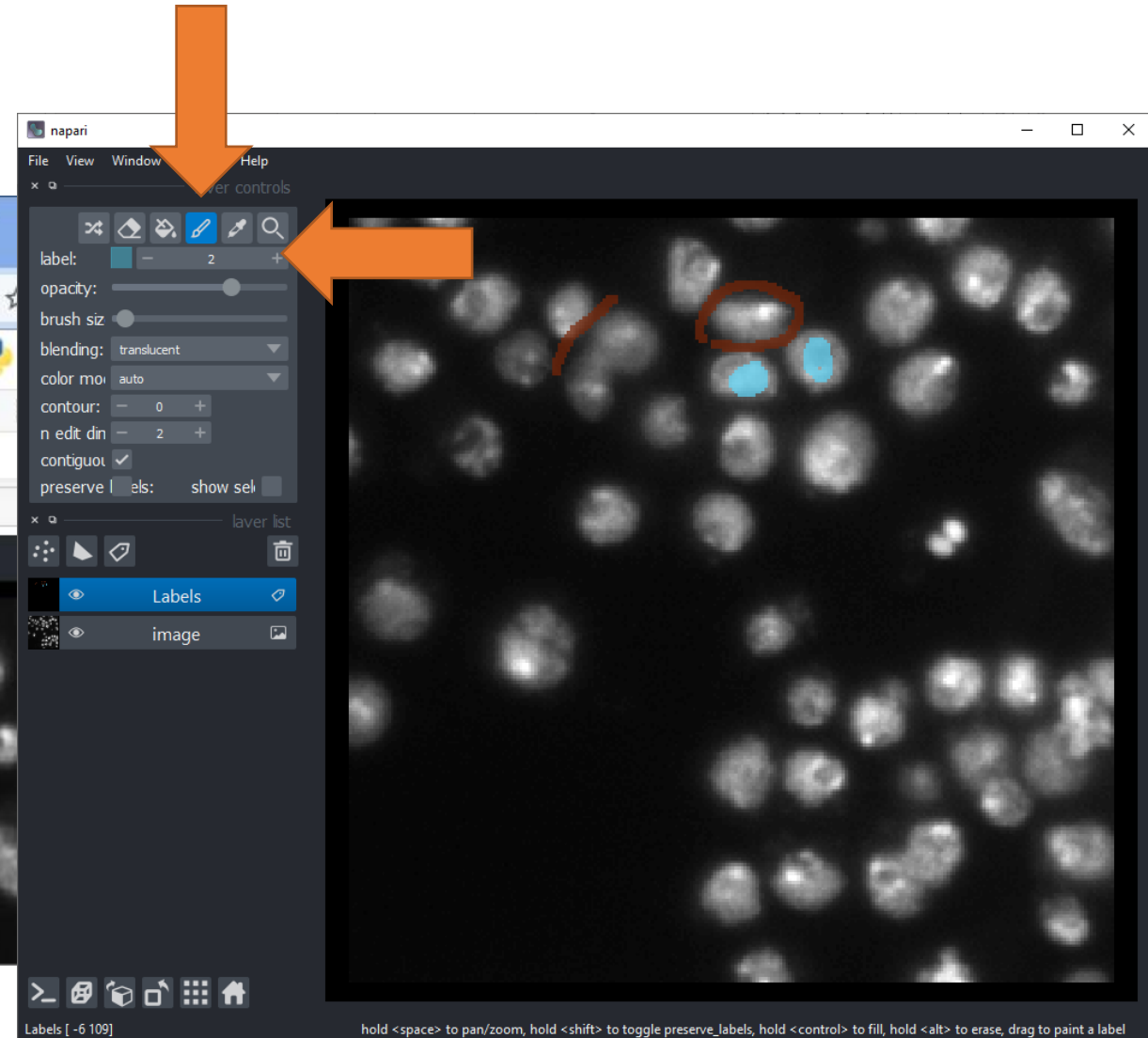
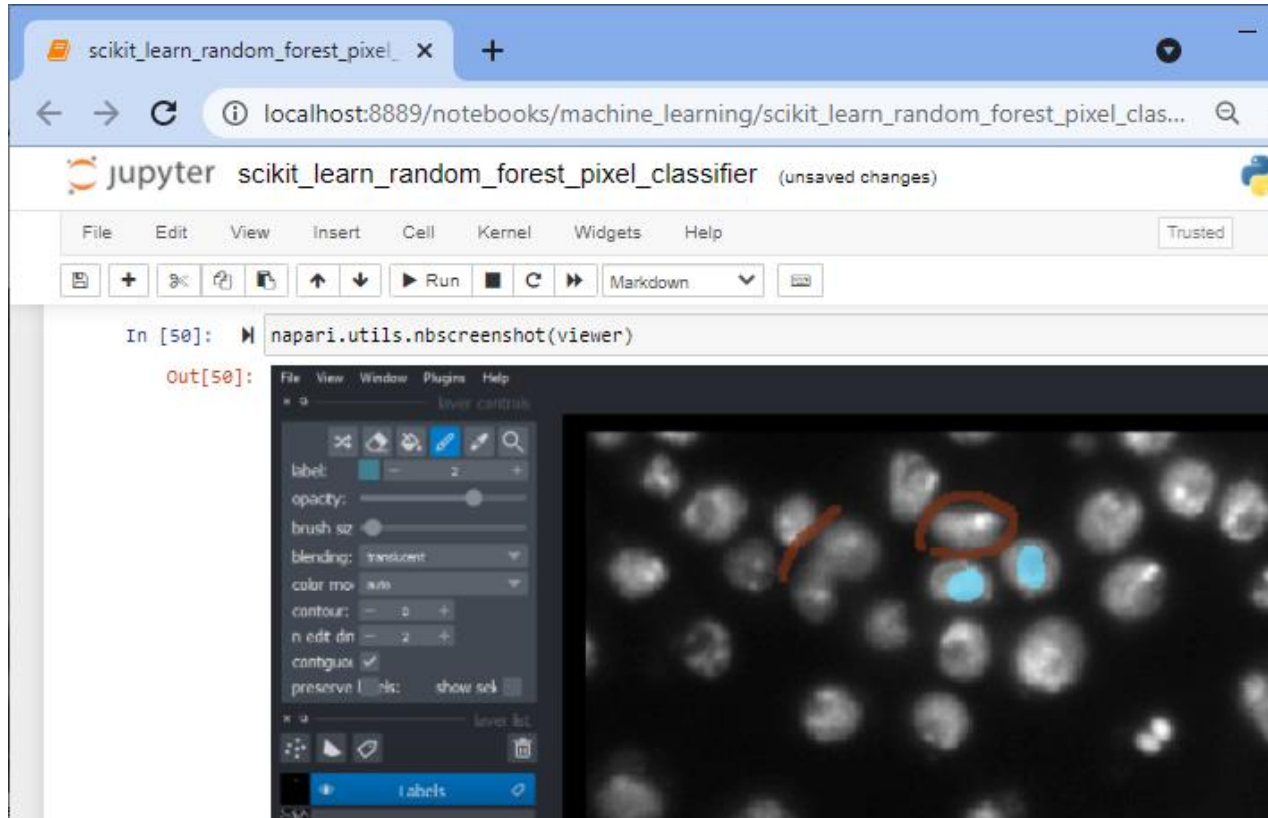
# add an empty Labels Layer and keep it in a variable
labels = viewer.add_labels(np.zeros(image.shape).astype(int))
```

An orange arrow points from the code cell to a text block that says: "Go ahead after annotating at least two regions with labels 1 and 2. Take a screenshot of the annotation:"

The napari viewer window is open to the right, showing a grayscale image of cells. The viewer controls on the left include a layer list with "Labels" and "image" layers. The "Labels" layer is selected, and the viewer shows a grid of cells with some regions highlighted in blue and red, indicating segmentation results.

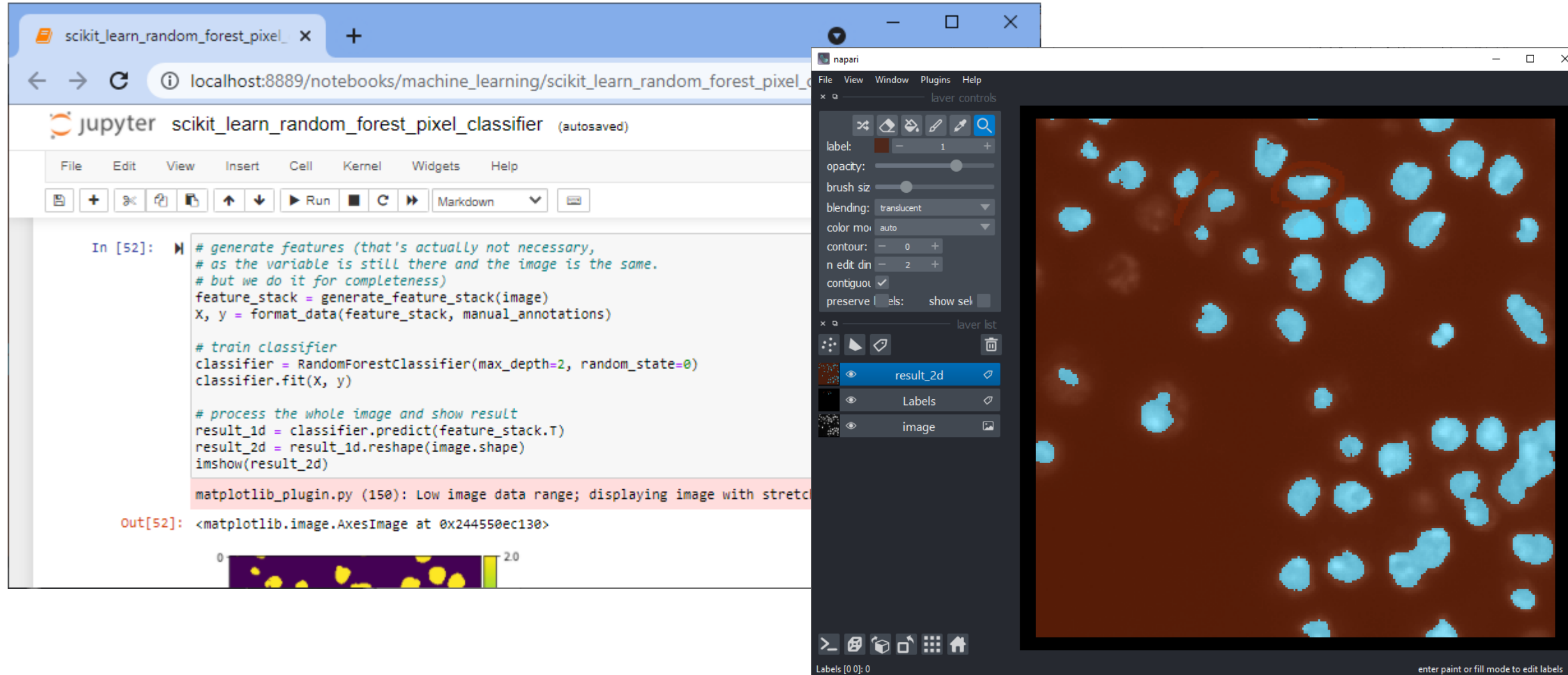
Interactive pixel classification

- Jupyter notebooks and napari side-by-side



Interactive pixel classification

- Jupyter notebooks and napari side-by-side



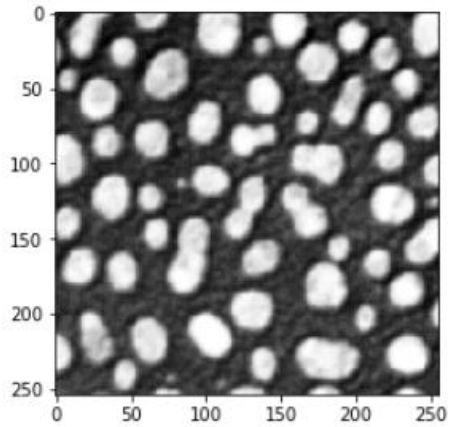
Accelerated pixel and object classification (APOC)

Robert Haase

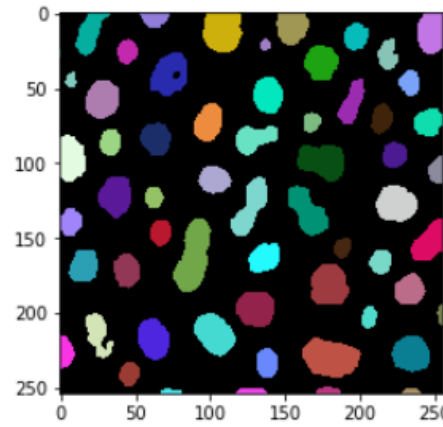
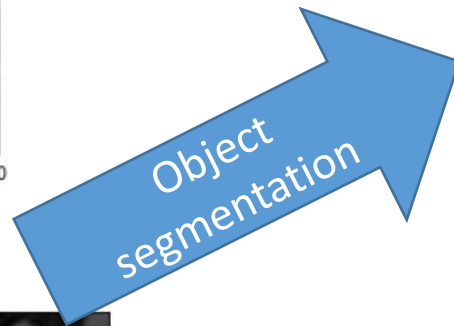
October 2022

Accelerated pixel and object classification

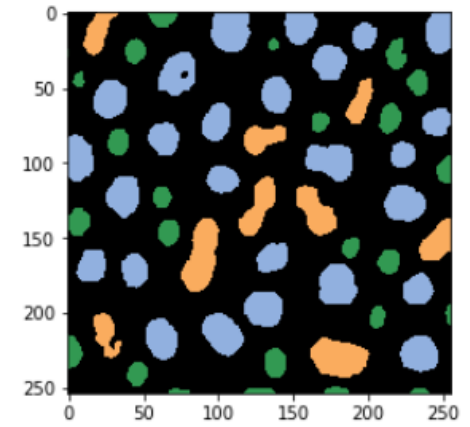
- APOC is a python library that makes use of OpenCL-compatible Graphics Cards to accelerate pixel and object classification



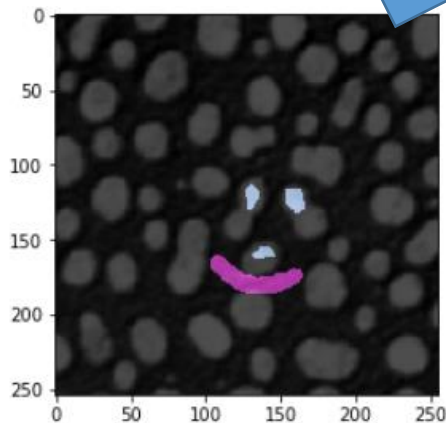
Raw image



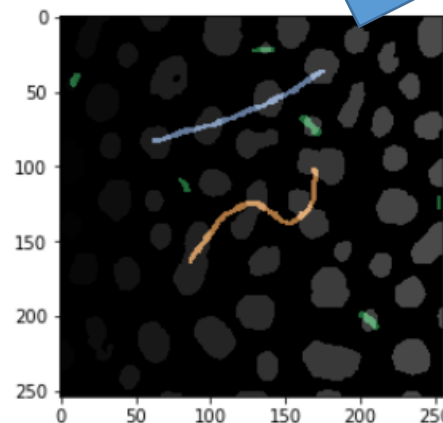
Object label image



Class label image

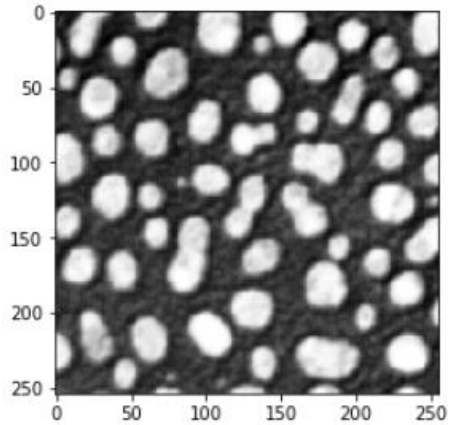


Pixel annotation

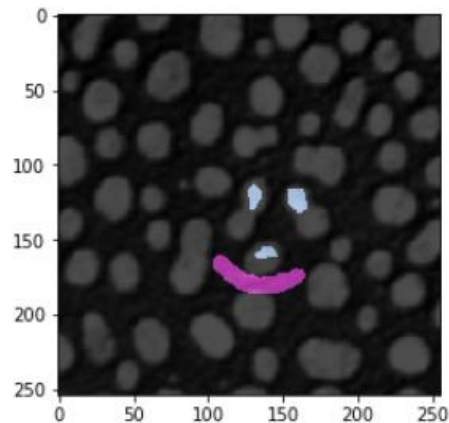


Object annotation

- Pixel classification + connected component labeling



Raw image



Pixel annotation

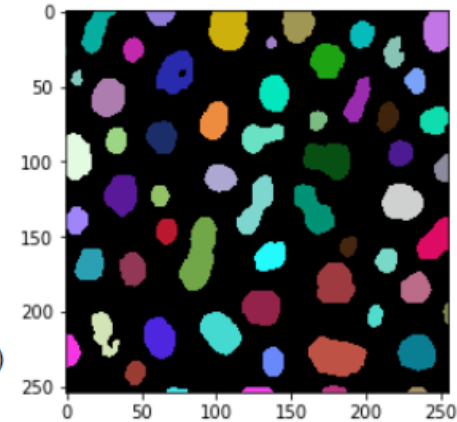
```
# define features
features = "gaussian_blur=1 gaussian_blur=5 sobel_of_gaussian_blur=1"

# 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)

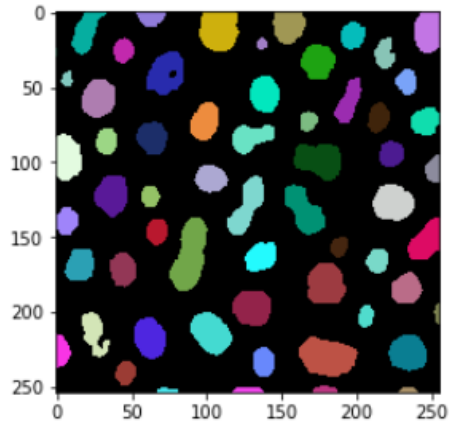
segmentation_result = clf.predict(features=features, image=image)
cle.imshow(segmentation_result, labels=True)
```



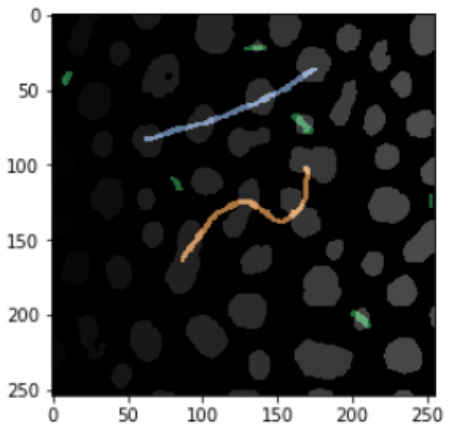
Object label image

Object segmentation

- Feature extraction + tabular classification



Object label image



Object annotation

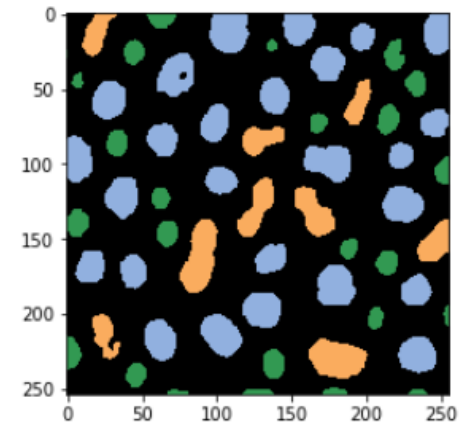
```
# for the classification we define size and shape as criteria  
features = 'area mean_max_distance_to_centroid_ratio'
```

```
# 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)
```

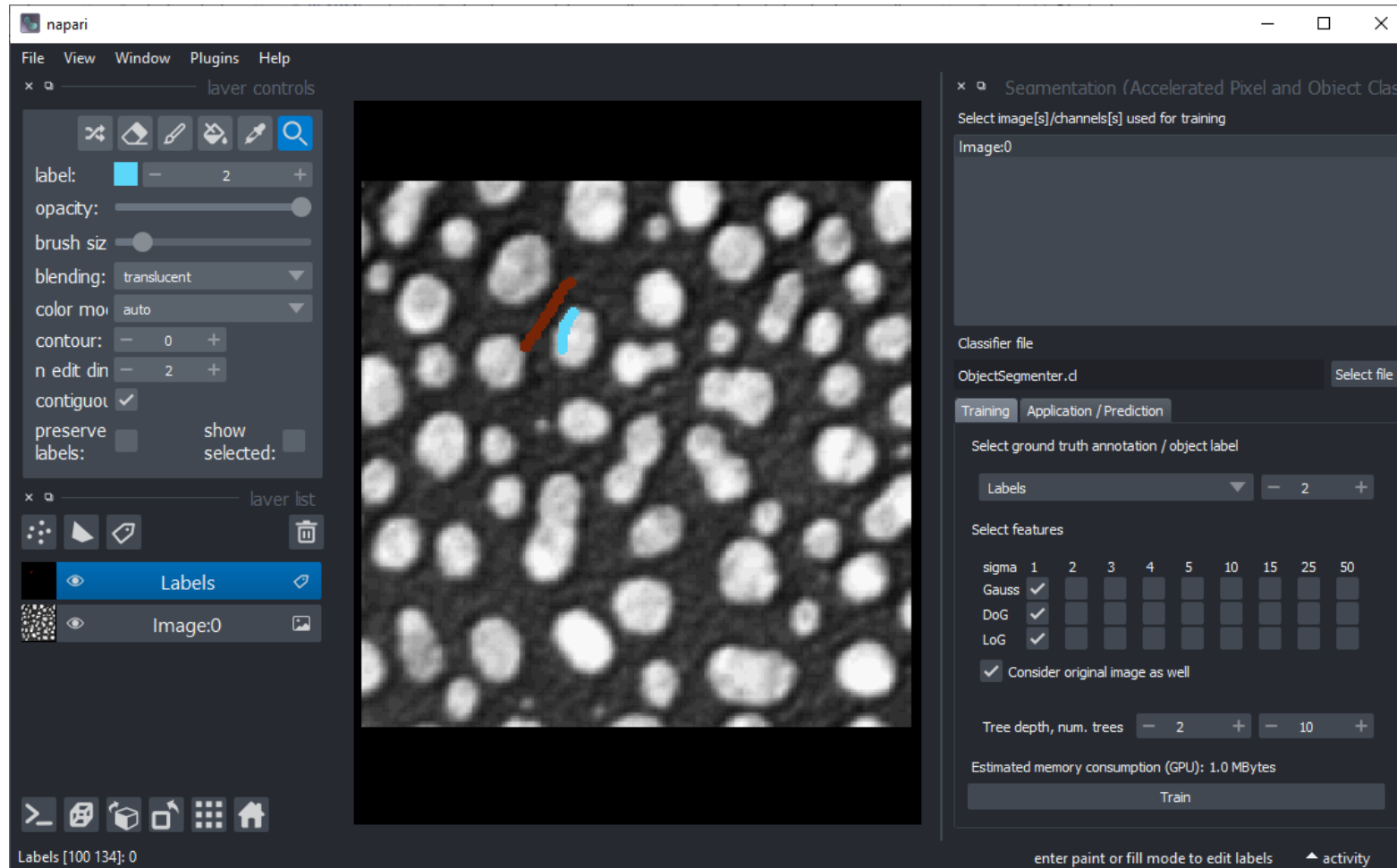
```
# determine object classification  
classification_result = classifier.predict(segmentation_result, image)  
cle.imshow(classification_result, labels=True)
```



Class label image

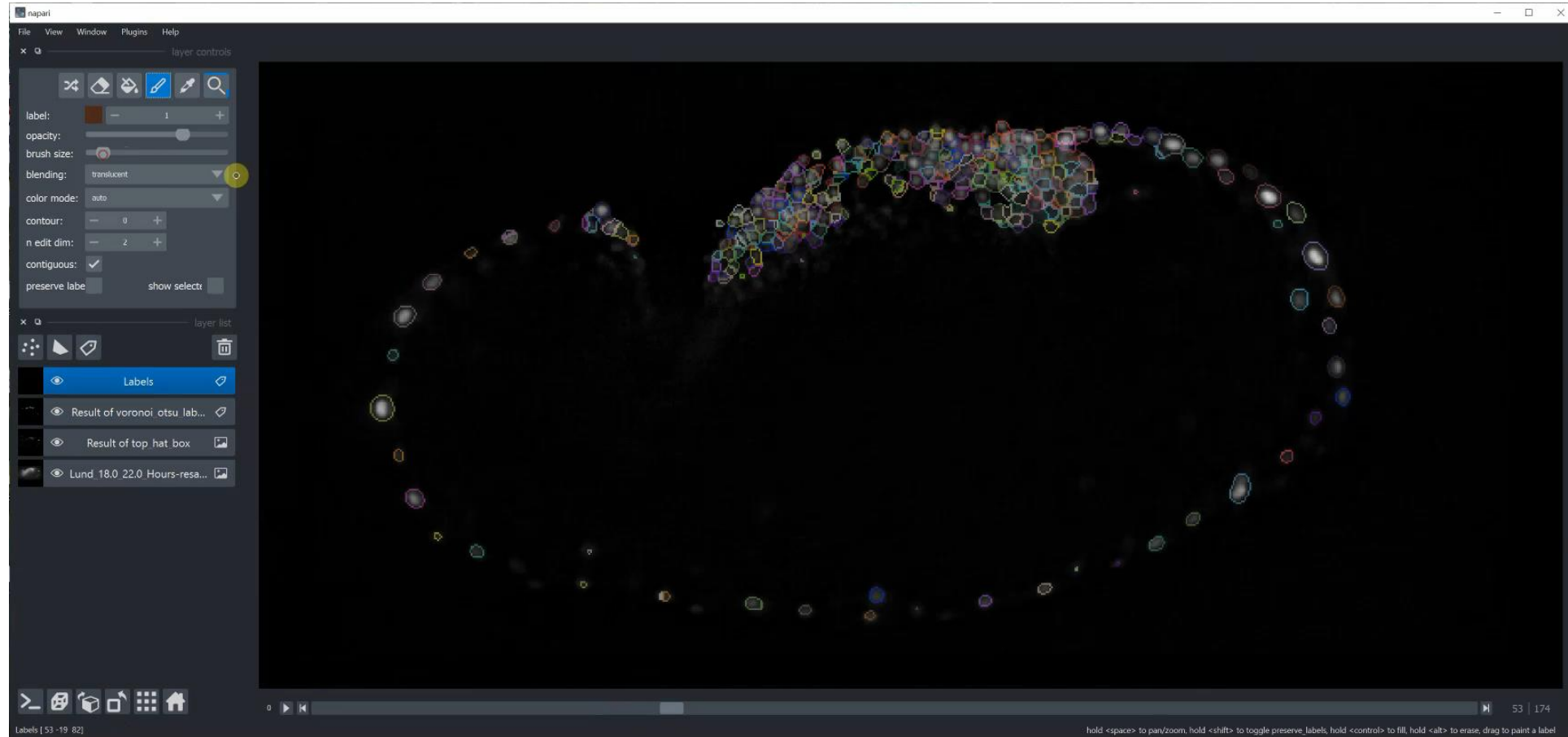
Object classification

- 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>

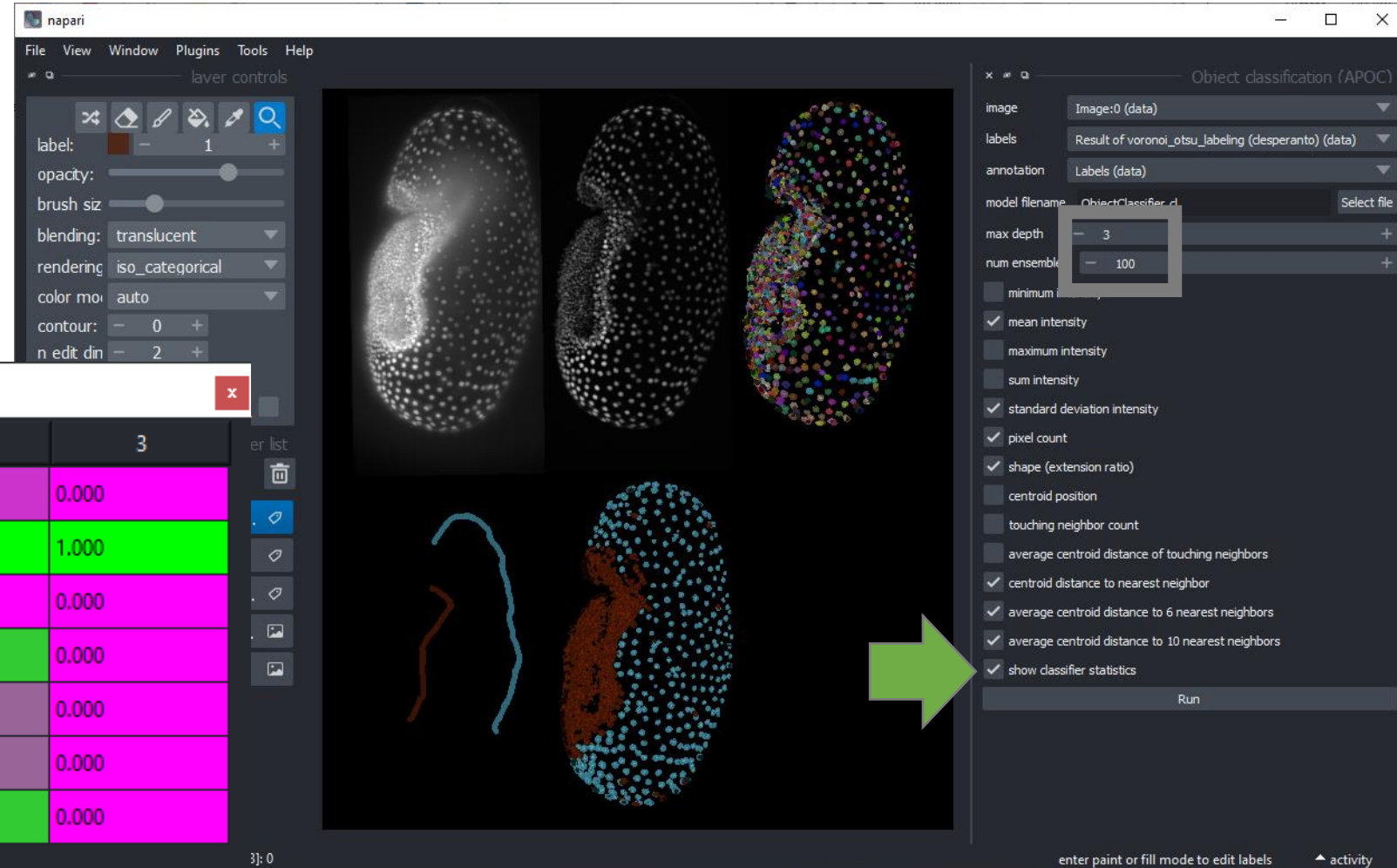


@haesleinhuepf
@PoLDresden

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

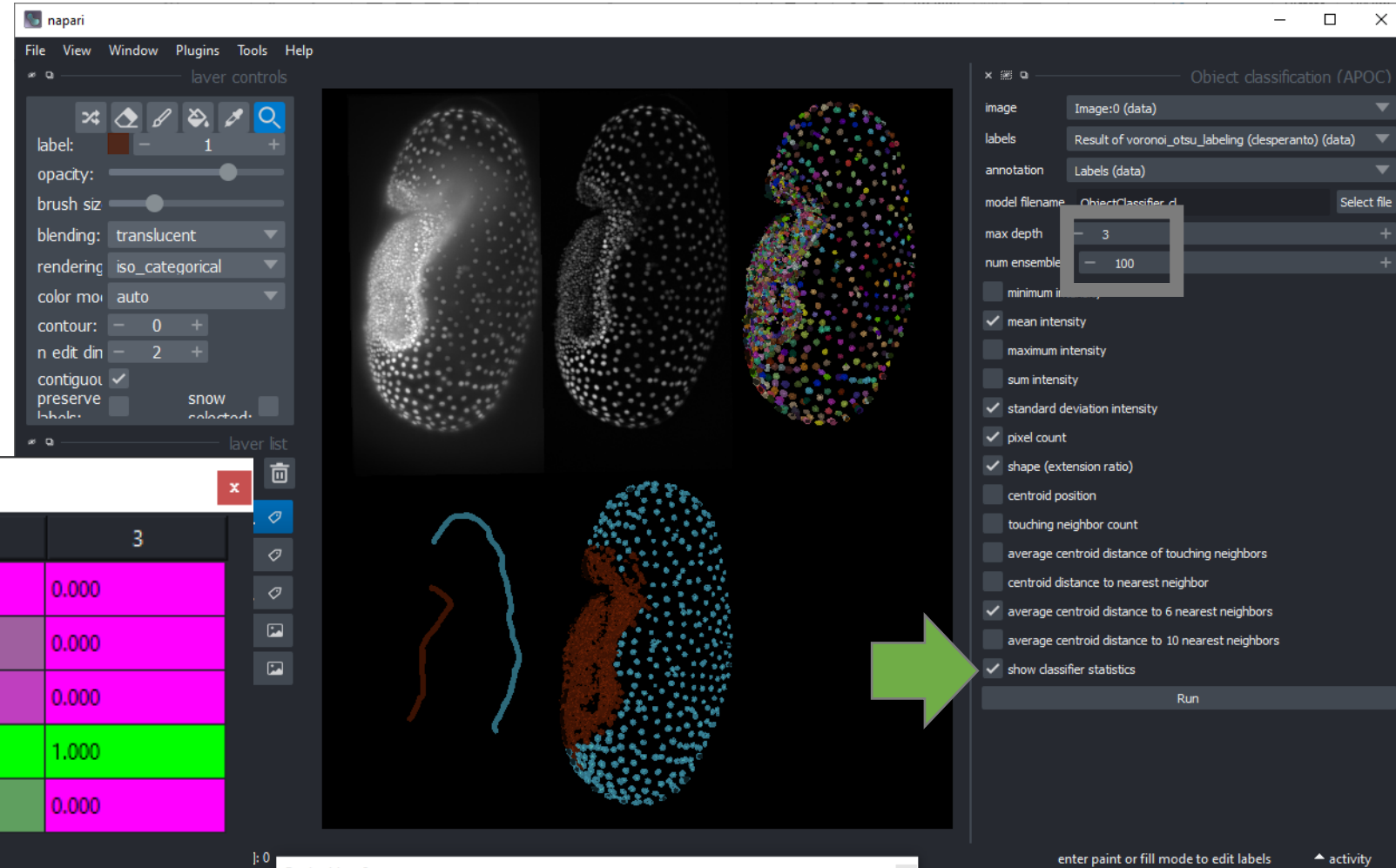
Data exploration / supervised machine learning

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



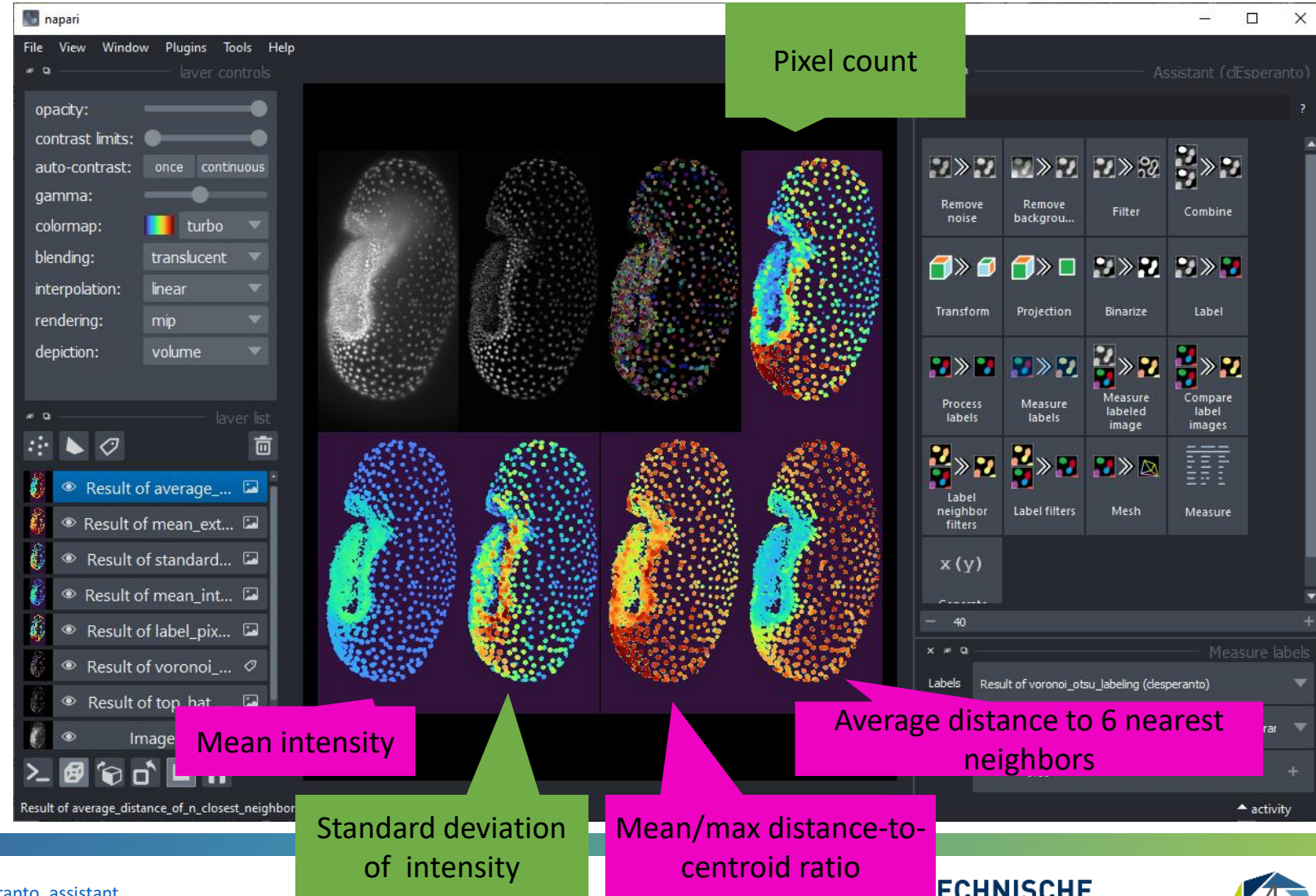
Data exploration / supervised machine learning

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



Data exploration / supervised machine learning

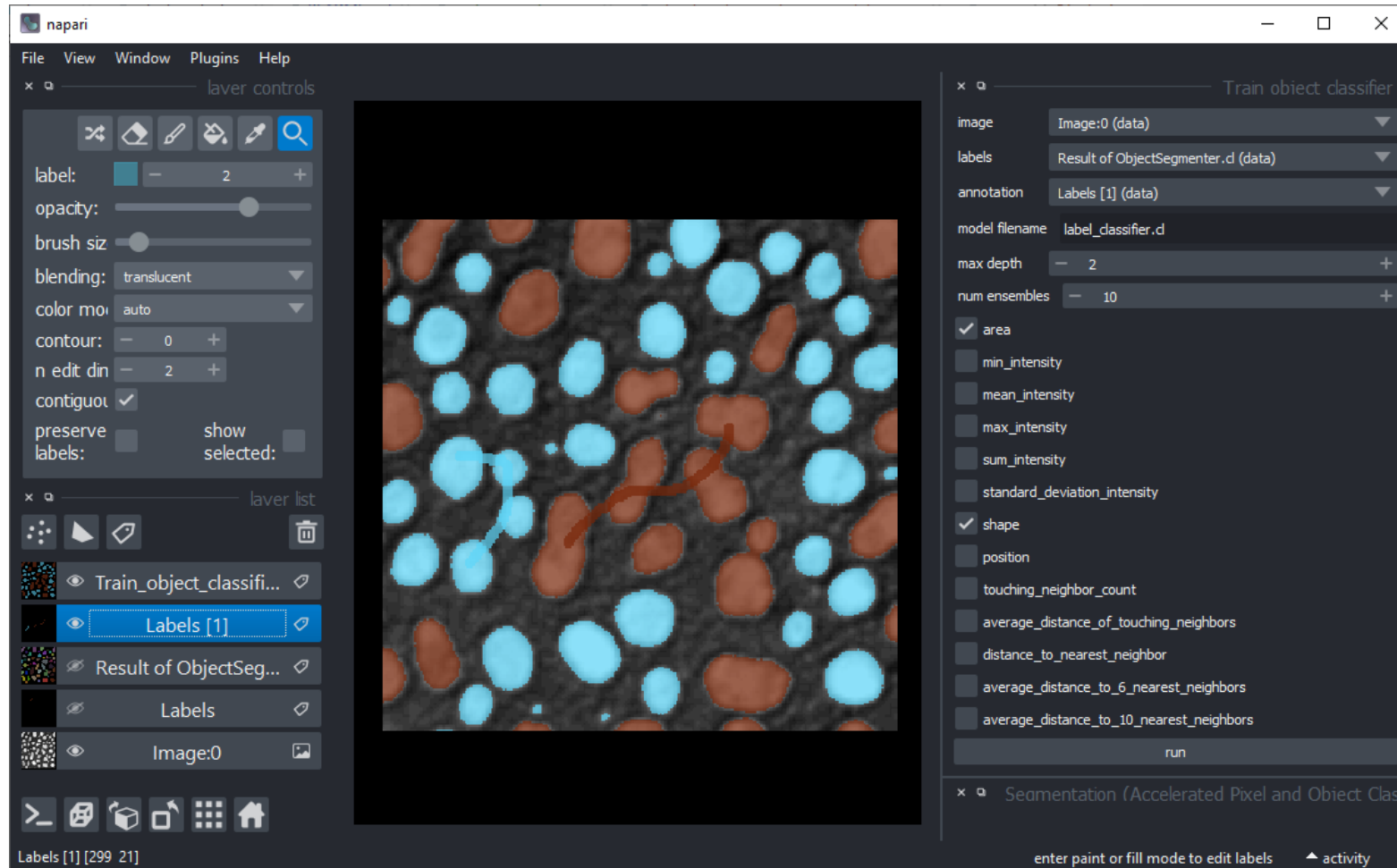
- Inspect how the random forest classifier makes decisions
- Note: Beware of correlated parameters!



https://github.com/clEsperanto/napari_pyclesperanto_assistant

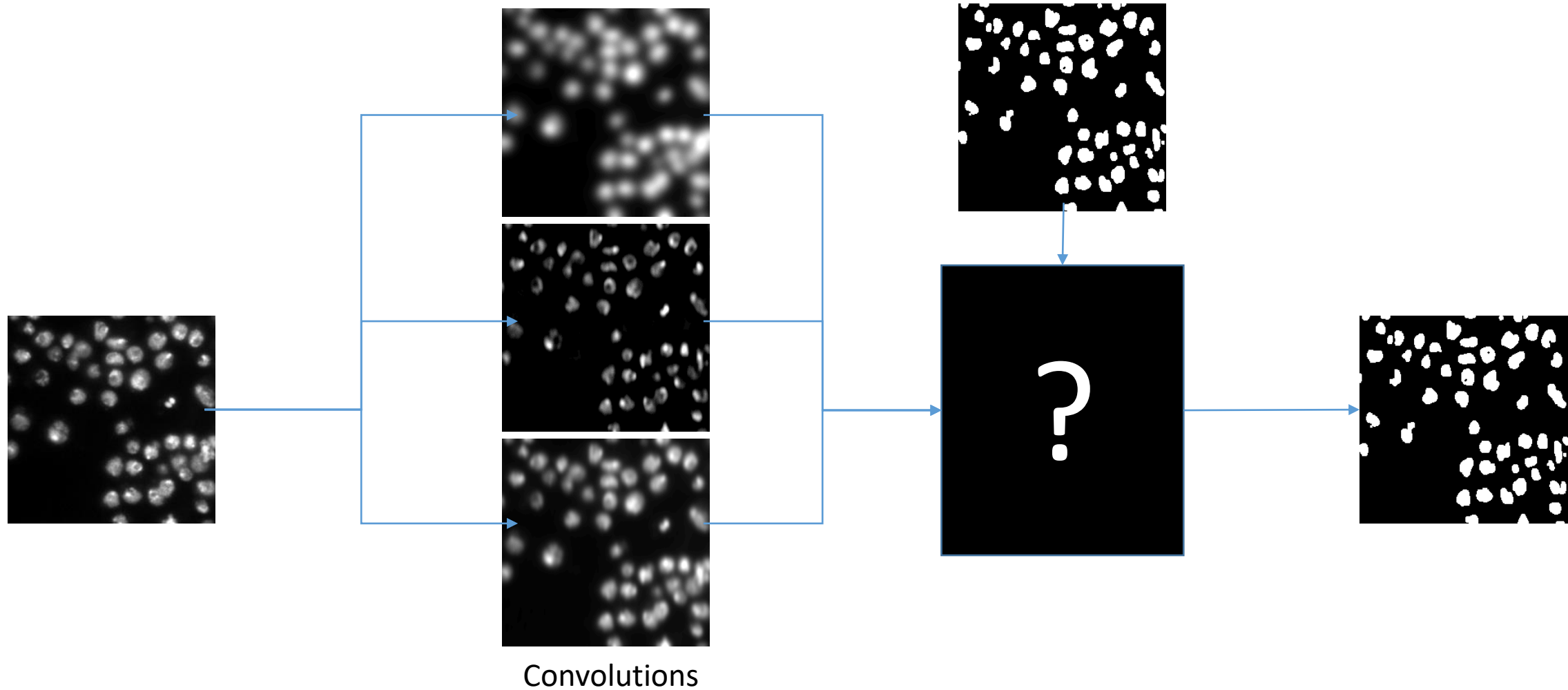
Image data source: Daniela Vorkel, Myers lab, MPI-CBG/CSBD

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



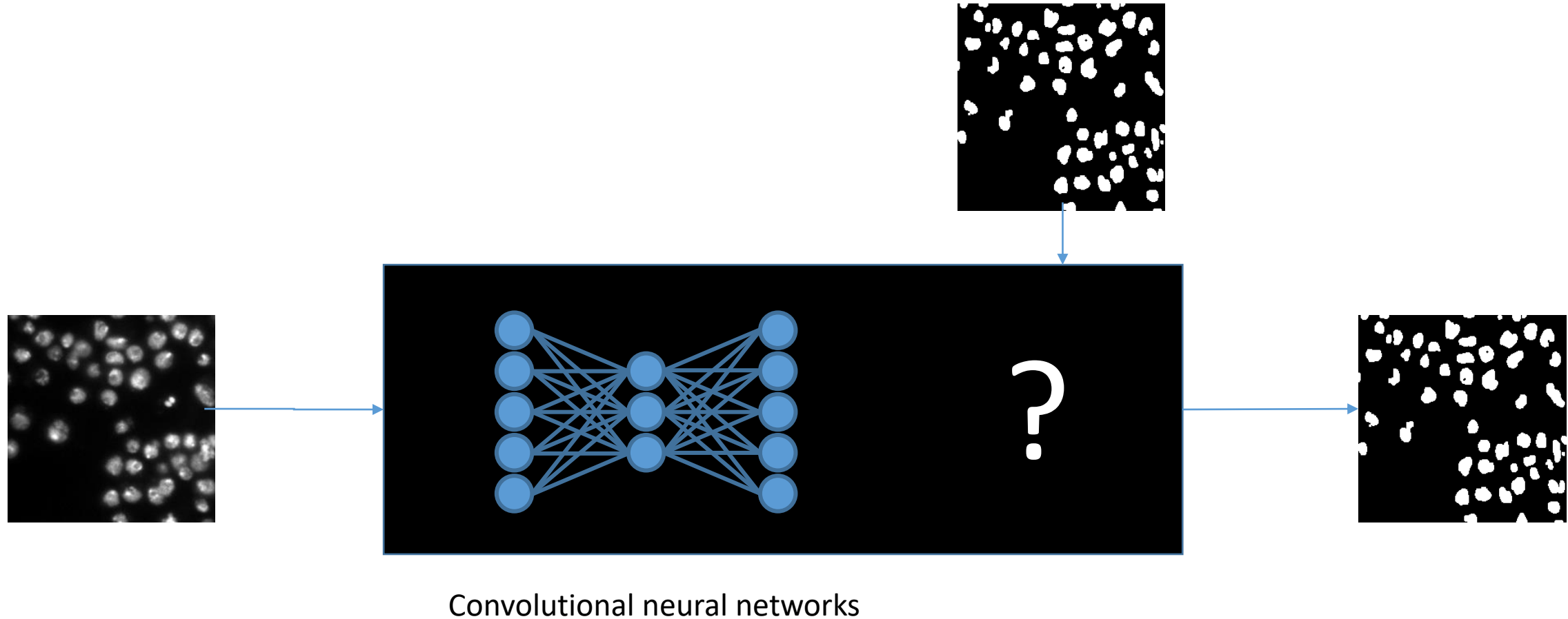
Summary & outlook

- 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



- Manual workflow design

Explicit definition of what's analysed

We specify features and how to combine them

Workload on human side is high

We can inspect code / models to understand how a decision is made

Uses neural networks

- Machine learning

Semi-automatic feature selection

We need to check carefully that the algorithm doesn't learn misleading features

Training can take long time

Allows us to analyze data we cannot manage manually

- Deep learning

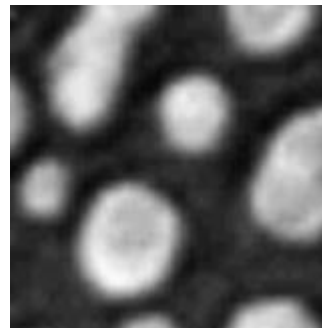
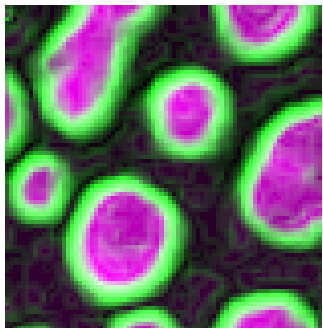
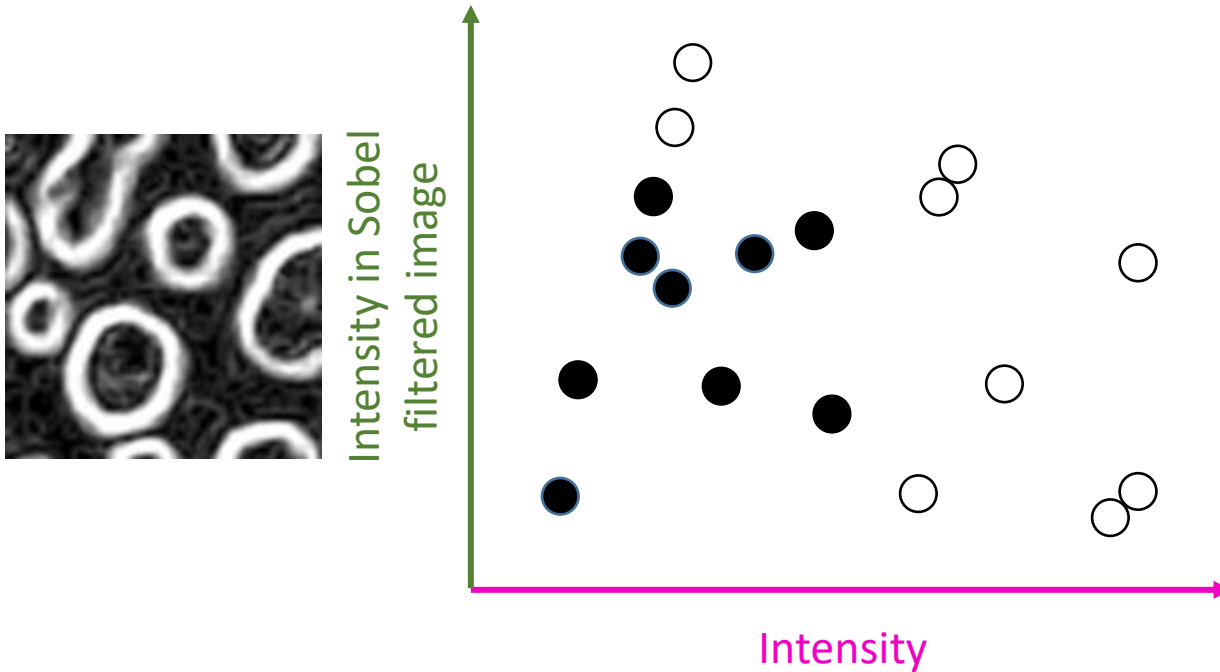
Fully automatic feature generation

Uses artificial neural networks

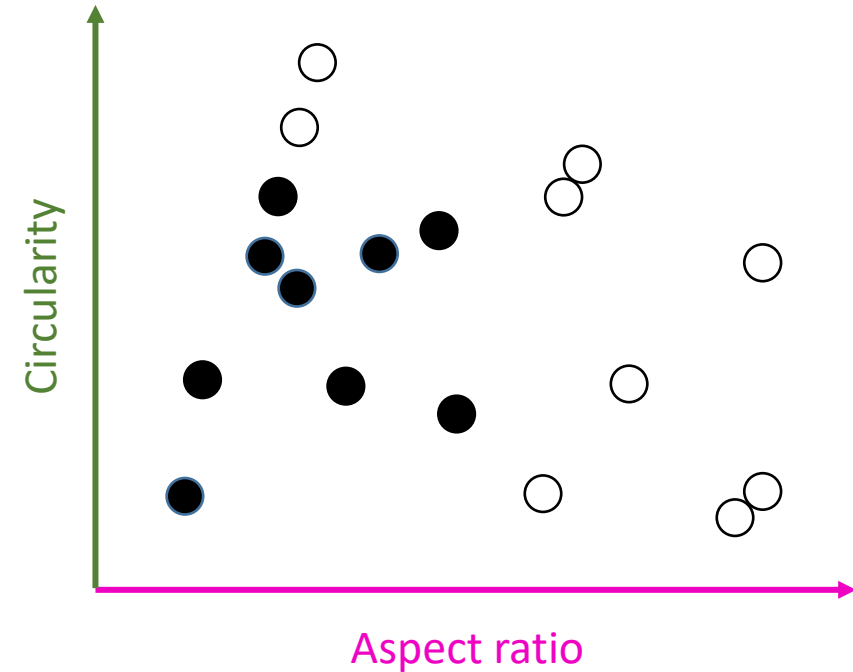
Needs training

- What if we exchange pixel features with object features?

Pixel classification

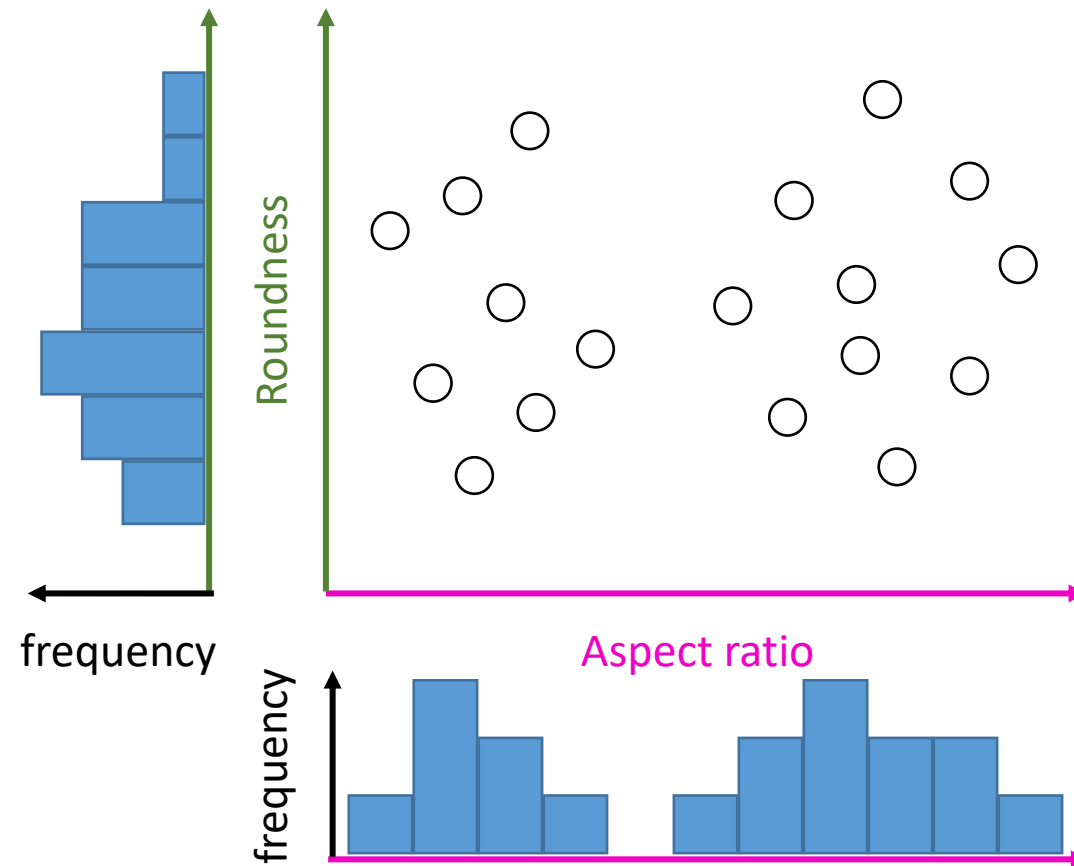


Object classification

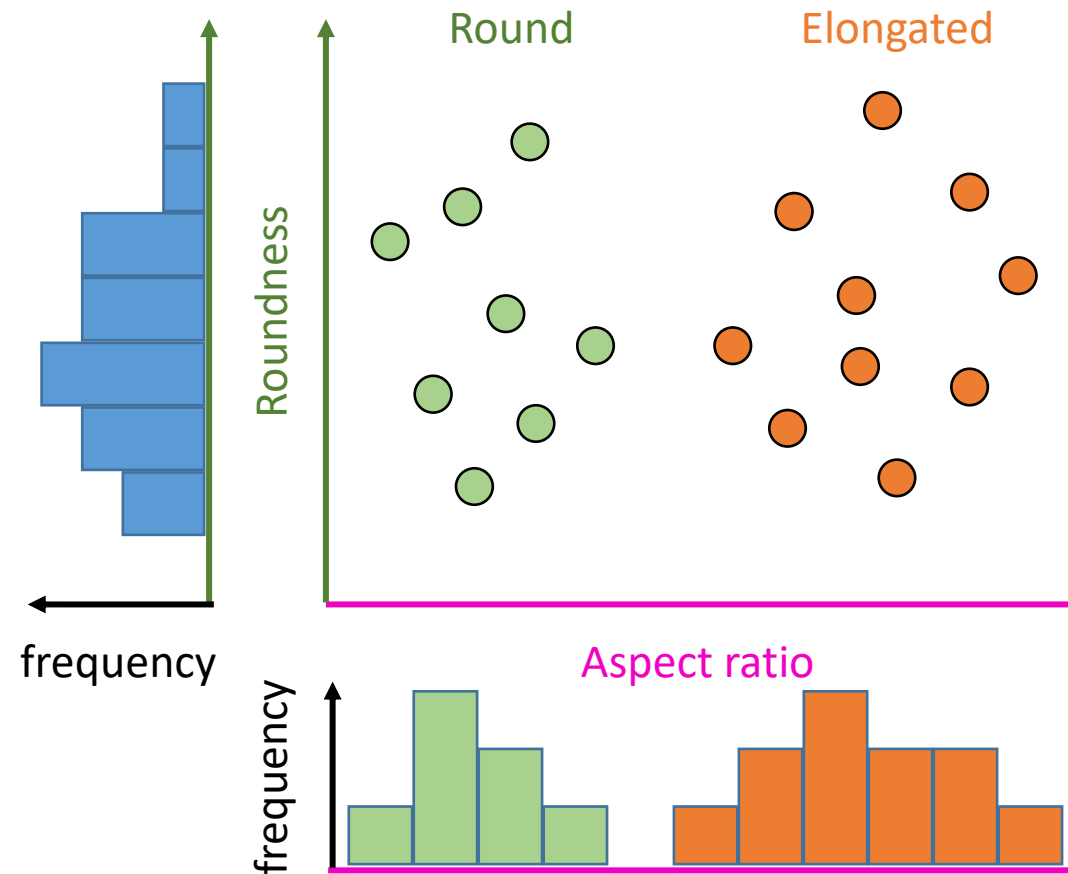


- The algorithms work the same but with different
 - Features
 - Number of features
 - Tree / forest parameters
 - Selection criteria

- 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

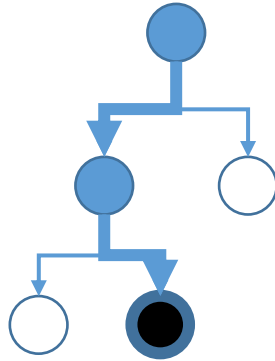


Further reading:

- Principal component analysis
- Cluster analysis

Today, you learned

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



Coming up next:

- Deep learning

