

Classic Segmentation with Python

Eva de la Serna, PhD

Associate Director of the Core for
Imaging Technology & Education

Slides & feedback from the BoBiAC Teaching Team & CITE!



Federico Gasparoli, PhD



Eva de la Serna, PhD



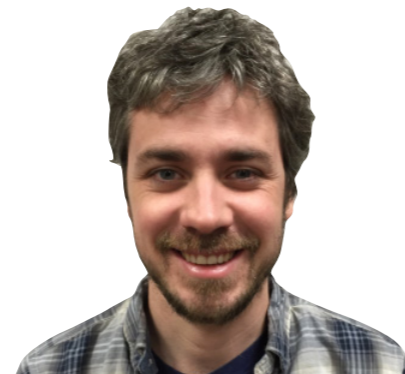
Max Brambach, PhD



Amelie Andreas



Jennifer Waters
Director



Talley Lambert
*Associate Director
Imaging Technology*



Anna Jost
*Associate Director
Imaging Education*



Maria Theiss
Postdoc Fellow






Biological Question hypothesis



Model System



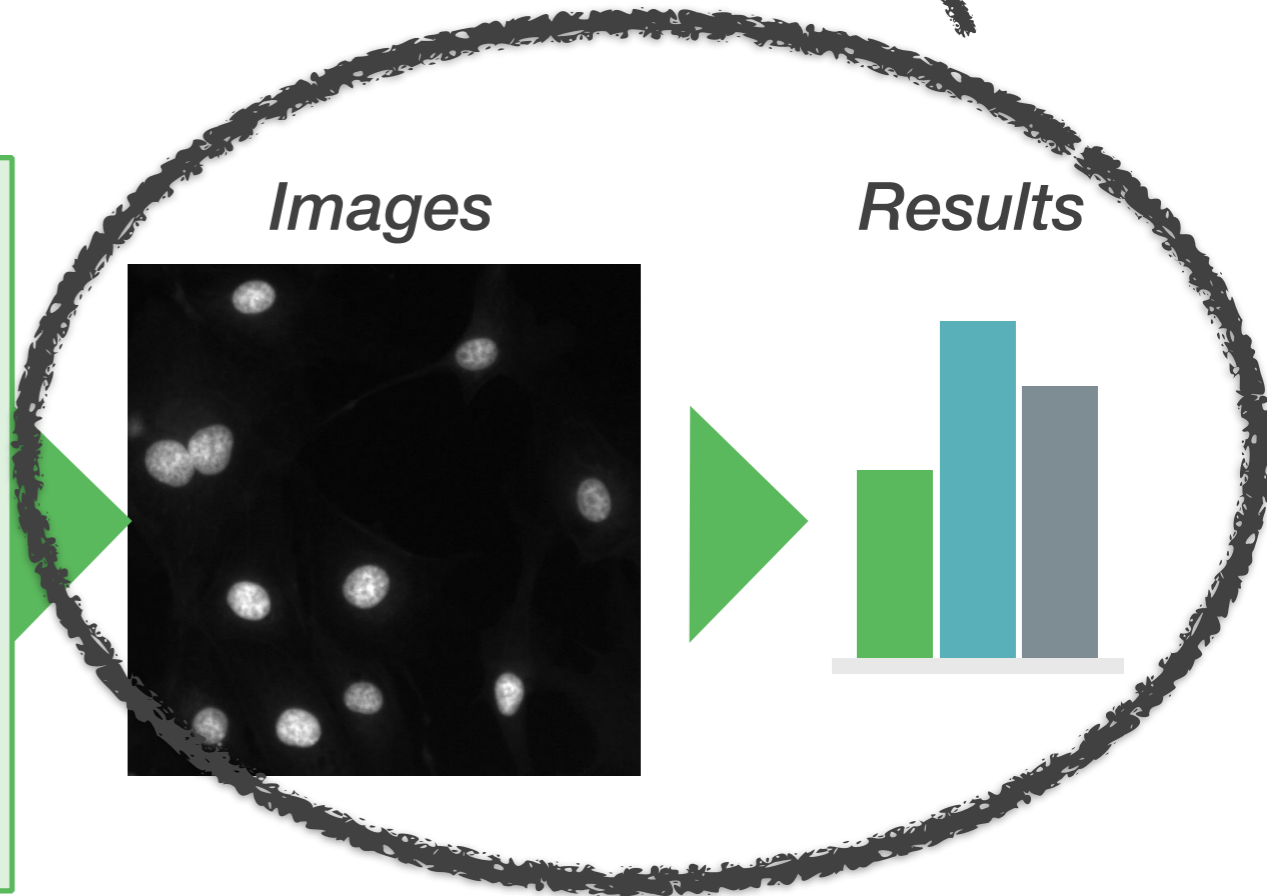
What sample can I prepare that will let me address my question?



Experiment

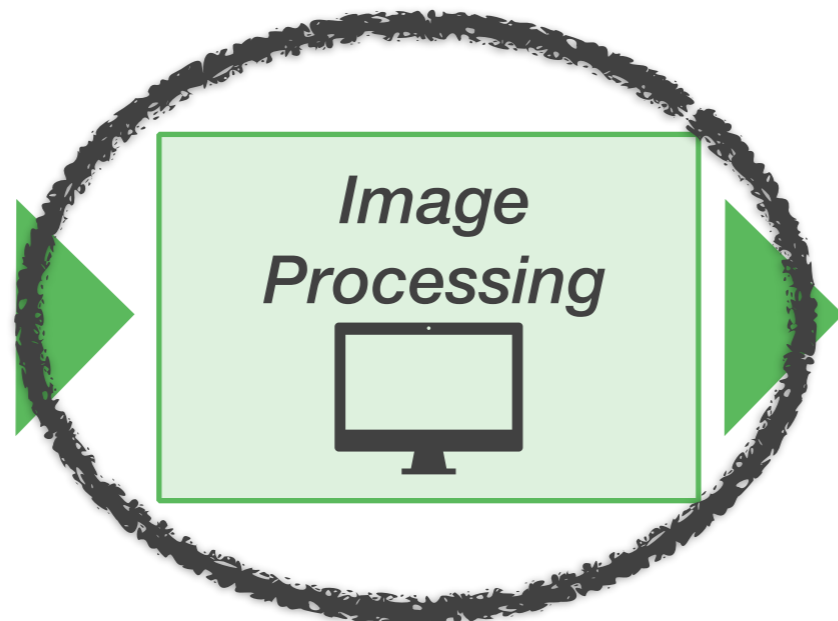
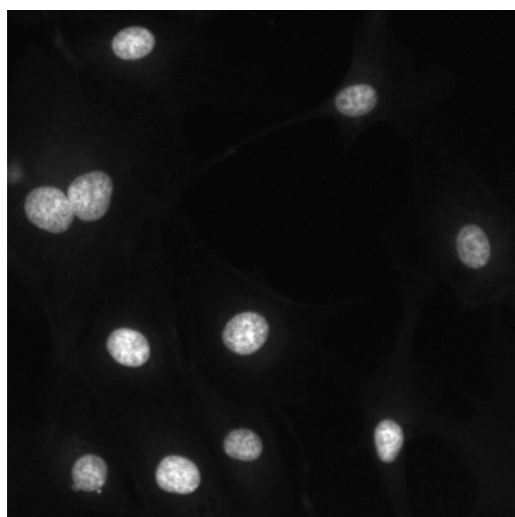


Can I design an experiment using fluorescence microscopy to address my question?





Images



*Image
Processing*



*Image
Analysis*



Results



Today's Focus:

Analysis Goal: Make measurements on objects in images

Processing Goal: *Select* individual objects in images





Example Dataset



WHAT

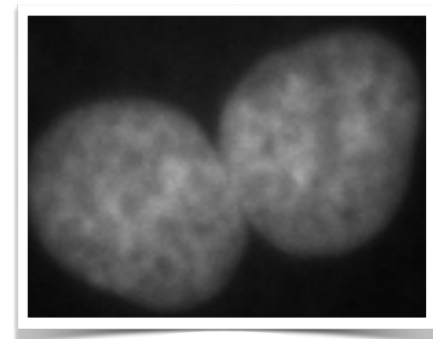
Fluorescence images of mammalian cell monolayer

LABEL

DAPI: a fluorophore that binds to minor grooves of dsDNA

DATASET

Widefield fluorescence images



DAPI is a nuclear marker!

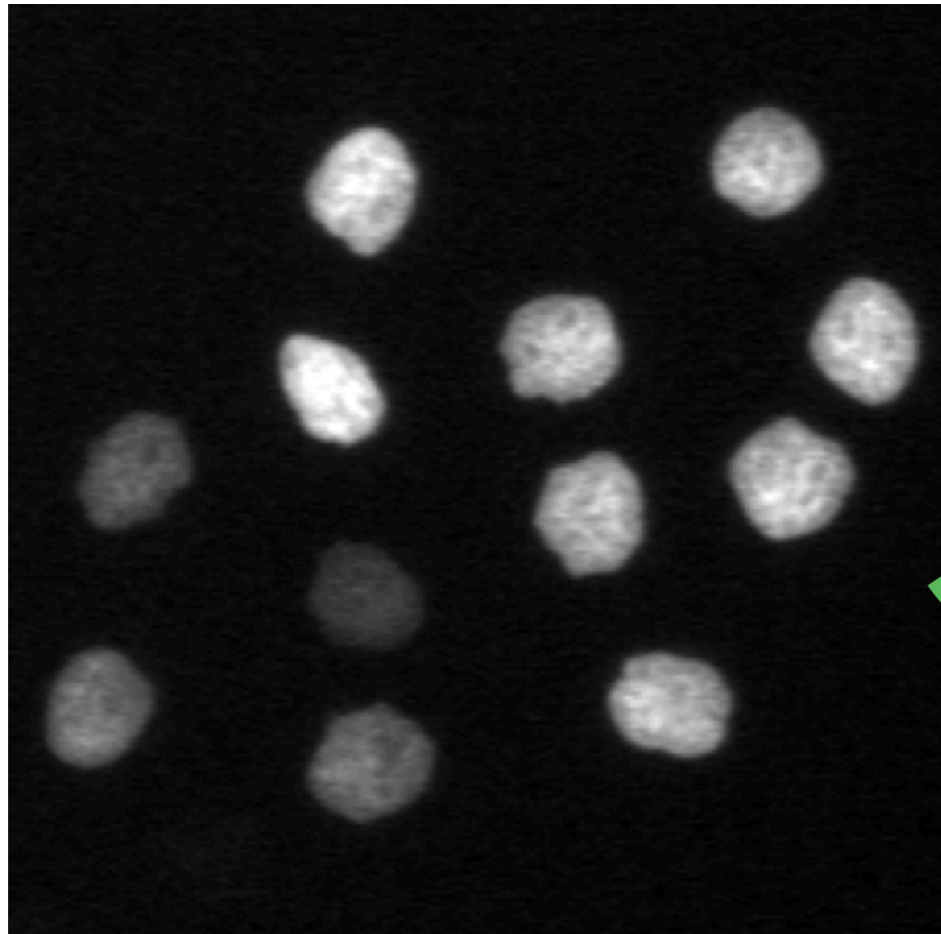




Analysis Goal: Make a measurement on images of nuclei



Original Image



Segmented Image



To measure nuclei properties, we need to ***select*** each individual nucleus. This process is called ***image segmentation***.

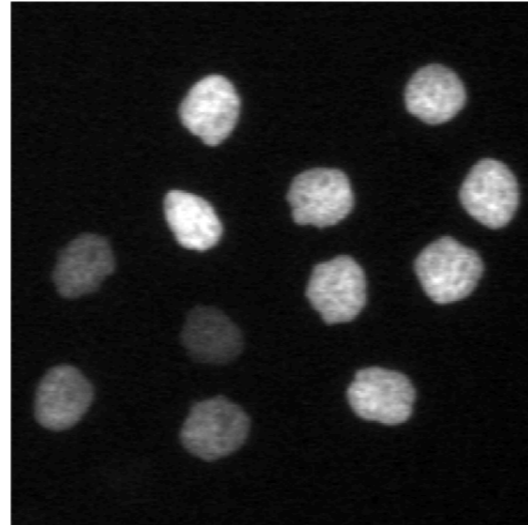




2 types of segmentation: Semantic & Instance



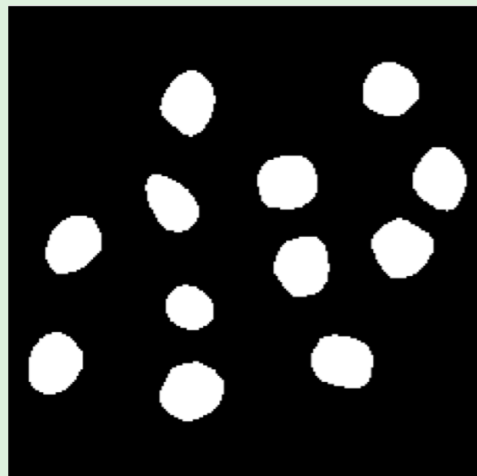
Raw Image



Semantic Segmentation

All objects treated as the same category

Example:



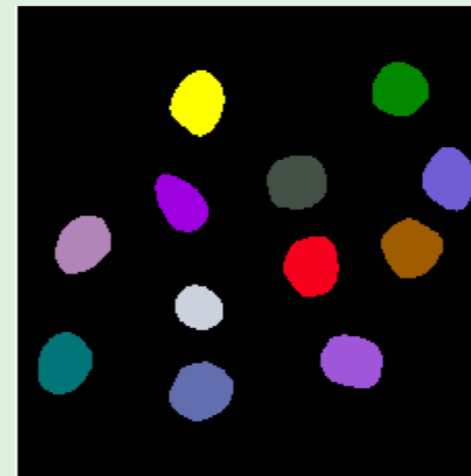
Categories:

Nucleus

Instance Segmentation

Each object is distinguished as separate and has a unique label

Example:



Categories:

Nucleus 1

Nucleus 2

Nucleus 3

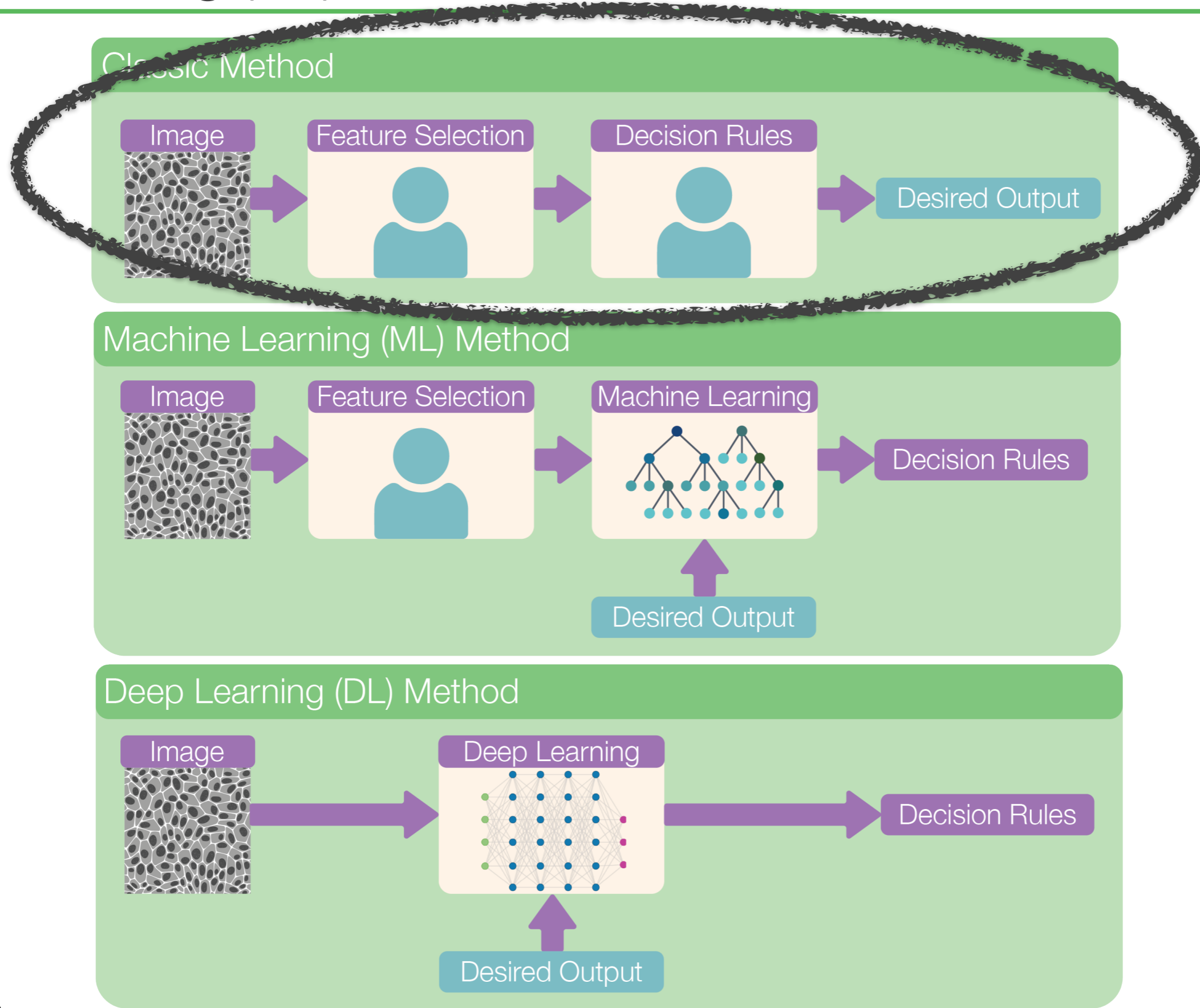
...

We can achieve either segmentation type with different *pipelines*.



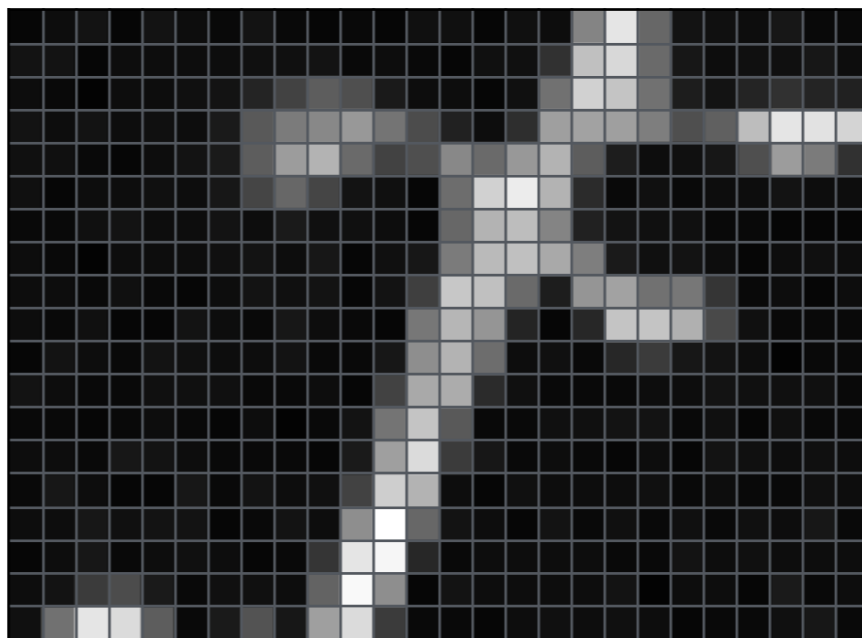
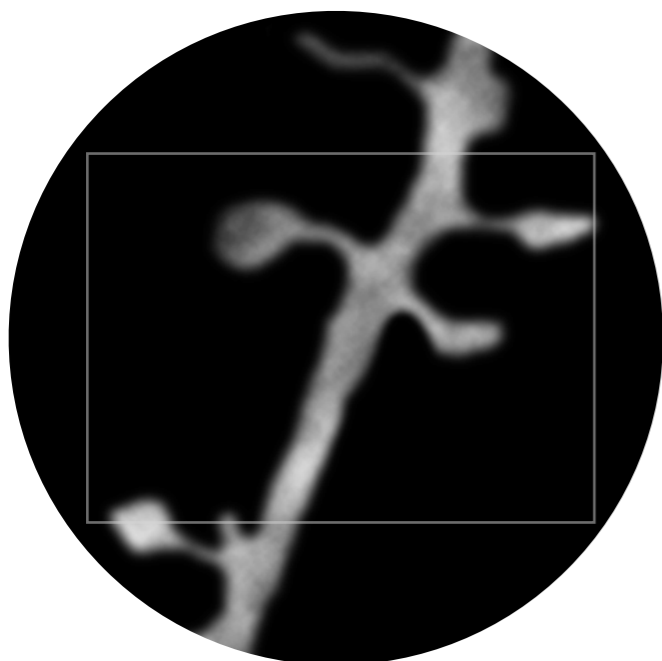


3 pipelines: Classic, Machine Learning (ML), & Deep Learning (DL)





The **classic pipeline** takes advantage of an image's spatial organization of intensity values



6	13	19	6	19	13	9	19	9	6	9	6	16	16	6	16	13	132	229	103	19	16	13	23	9	9
19	19	6	13	13	13	13	16	16	19	9	13	9	6	16	16	49	192	216	106	23	13	16	16	23	13
13	9	4	13	13	16	19	36	66	93	79	26	13	13	6	16	113	209	196	113	29	19	36	49	36	33
19	13	19	13	16	13	26	89	123	136	152	116	76	33	13	46	159	162	159	126	79	96	189	229	226	212
16	16	9	6	13	19	26	93	156	179	106	66	79	136	106	152	179	93	29	13	16	23	79	156	123	49
16	6	13	13	16	13	23	69	103	69	19	16	6	109	209	236	179	43	9	16	9	13	13	19	13	13
9	9	16	19	13	13	19	13	26	16	16	13	6	103	179	189	132	33	19	16	16	9	9	6	6	6
13	9	4	13	13	13	16	19	13	23	6	16	23	123	186	192	169	126	26	16	19	13	6	13	16	13
13	13	9	16	9	6	13	19	16	19	6	19	63	199	192	106	29	149	162	113	119	53	9	13	6	13
13	9	16	6	6	19	13	9	23	13	9	6	119	182	149	36	6	39	196	196	176	73	16	9	9	9
6	19	13	9	19	16	13	13	19	9	9	23	142	179	109	13	16	9	39	59	23	19	13	4	9	9
19	13	9	9	16	16	16	9	9	13	6	66	169	172	43	16	9	9	9	13	13	19	16	16	16	9
9	9	6	9	13	9	6	13	4	9	19	116	196	89	9	9	16	16	19	19	9	16	6	16	9	9
13	13	9	23	19	13	9	9	9	6	26	159	219	59	23	9	13	9	6	13	6	19	16	13	16	13
9	23	13	6	6	23	9	19	13	16	66	206	179	13	6	16	13	13	13	16	9	13	9	9	16	13
13	13	23	16	19	19	6	9	19	13	142	255	103	19	13	6	19	9	16	9	16	9	16	13	23	9
6	13	23	9	13	16	13	6	9	53	229	246	39	9	13	13	13	13	9	9	19	13	16	13	13	13
13	19	59	76	26	9	16	16	13	99	249	142	6	19	13	13	13	13	19	4	13	13	6	26	9	13
16	113	229	219	93	9	26	83	23	159	219	59	9	9	6	13	16	13	16	13	6	9	9	16	23	9

photons
optical image



intensity values
digital image

intensity values \neq photons!!



classic segmentation with Python





classic segmentation with Python

thresholding

filtering

labeling a binary mask

refining a binary mask





classic segmentation with Python

thresholding

filtering

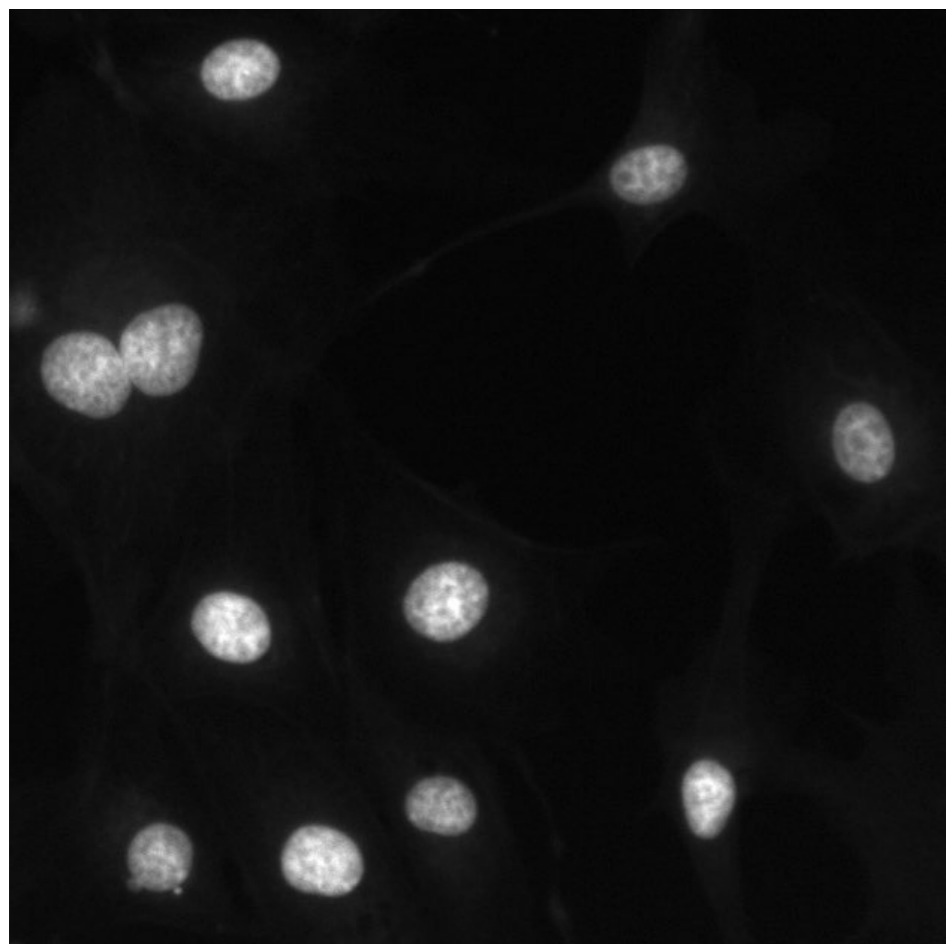
labeling a binary mask

refining a binary mask





Original Image



Nuclei regions have higher intensity values than non-nuclei regions in the image



thresholding

select a range of digital values in the image

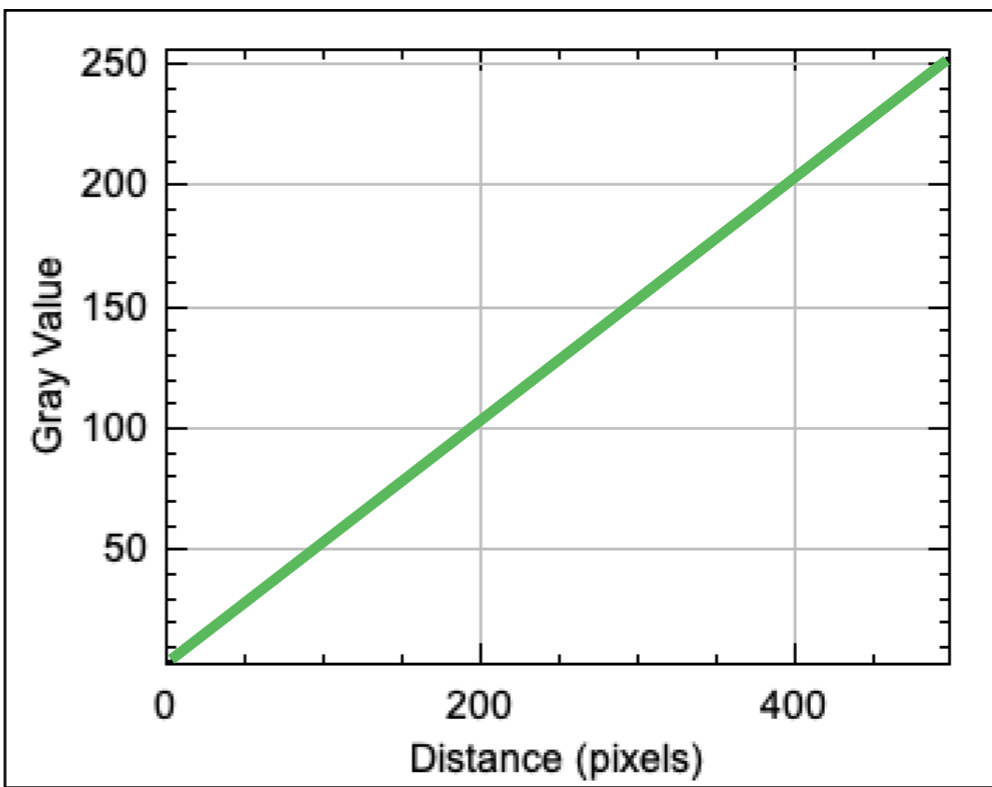
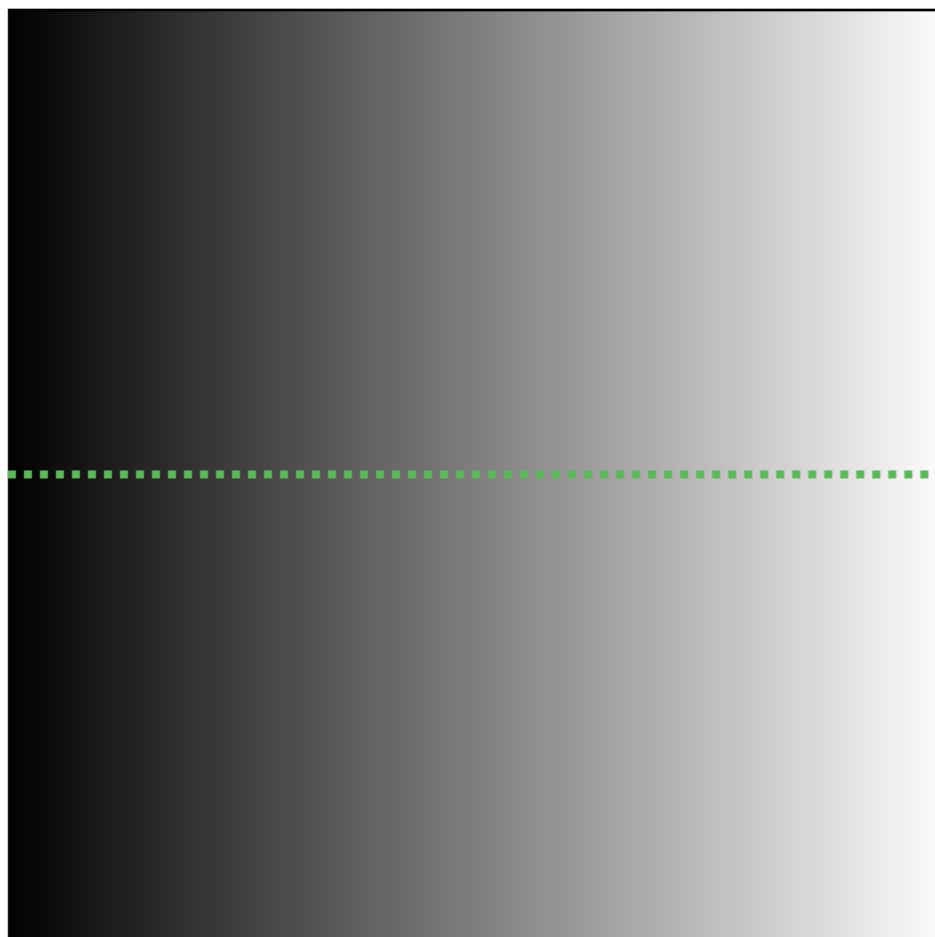




thresholding

select a range of digital values in the image

8 bit image (0 - 255)*



*8 bit image = each pixel can have 2^8 grey values = 256 grey values = range 0-255



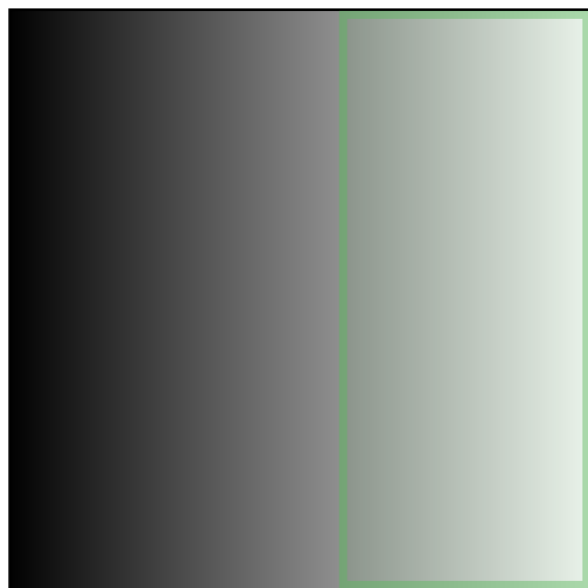


The result of the thresholding process is a *binary mask*



A **binary mask** is an image that has only 2 pixel values

8 bit image (0 - 255)



binary_mask

F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T

False: “discarded” pixels

True: selected pixels

FIJI’s binary masks have 0 and 255 values.

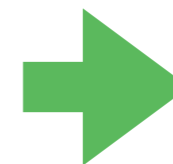
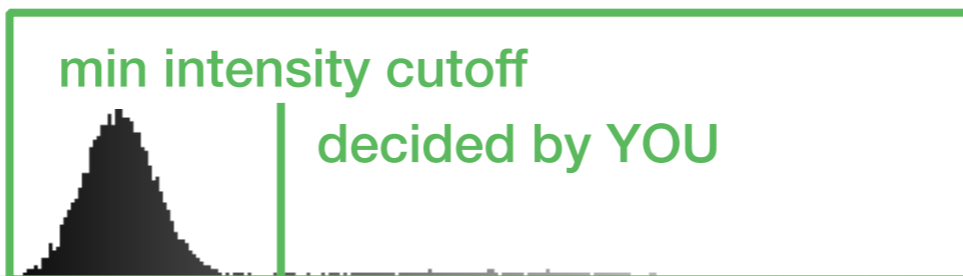


2 ways to set a threshold



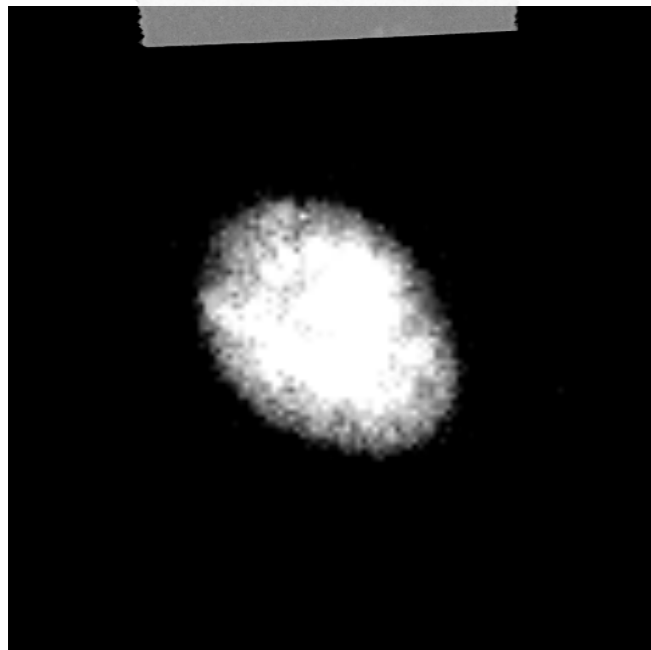
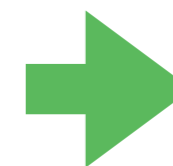
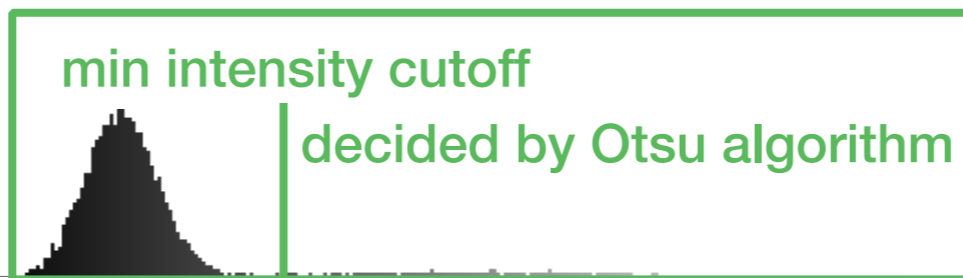
Thresholding by **manually** setting a min intensity count

Image Histogram



Thresholding by **automatically** setting a min intensity count using a thresholding algorithm

Image Histogram



Use thresholding algorithms so that thresholding will be reproducible across many images.

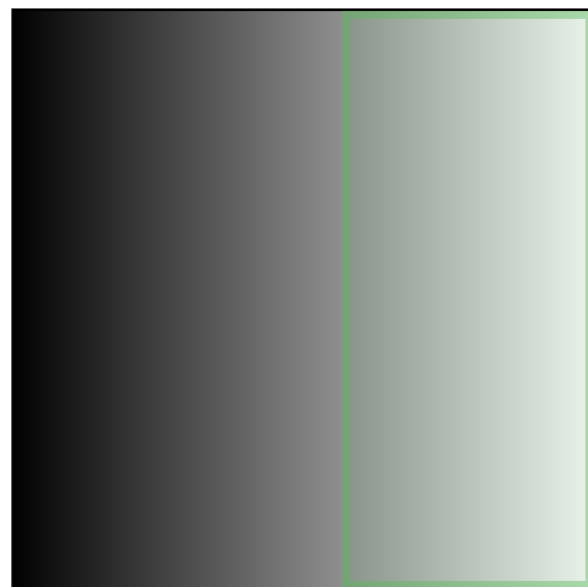
Thresholding in Python




Thresholding by **automatically** setting a min intensity count using a thresholding algorithm

```
[1]: # use Otsu thresholding algorithm
from skimage.filters import threshold_otsu
binary_mask = raw_image >
              threshold_otsu(raw_image)
```

raw_image



binary_mask



F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T

False: "discarded" pixels
True: selected pixels

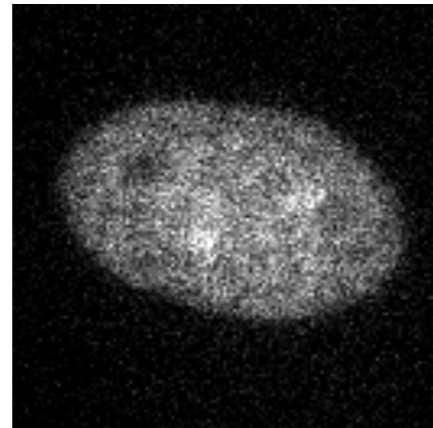




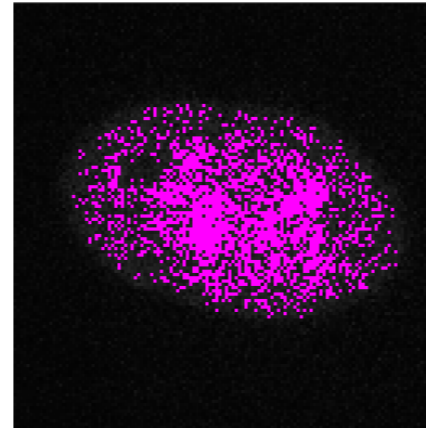
Usually if you apply thresholding to the original image, you won't precisely select all or only pixels of interest



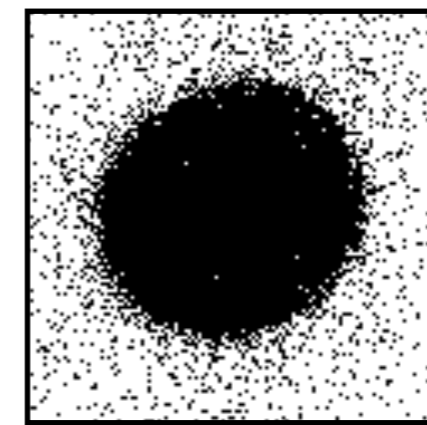
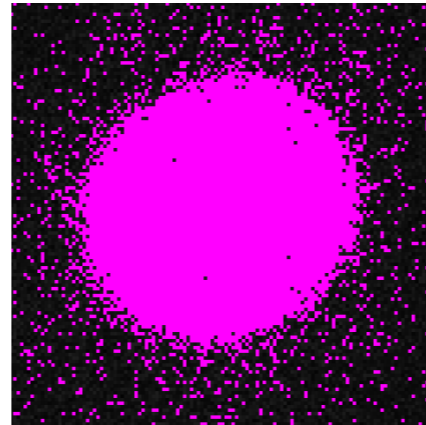
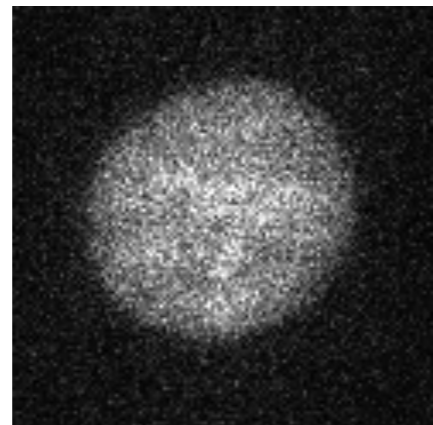
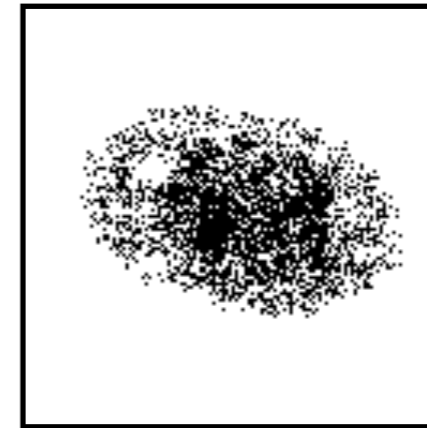
Image



Threshold



Binary Mask



Many factors can contribute to variance in pixel values:

fluorescence label (e.g. DAPI)

background (uneven illumination, out of focus light, aberration)

detector offset

noise (detector read noise, poisson noise, etc.)



classic segmentation with Python

thresholding

~~filtering~~

labeling a binary mask

refining a binary mask





classic segmentation with Python

filtering

thresholding

~~filtering~~

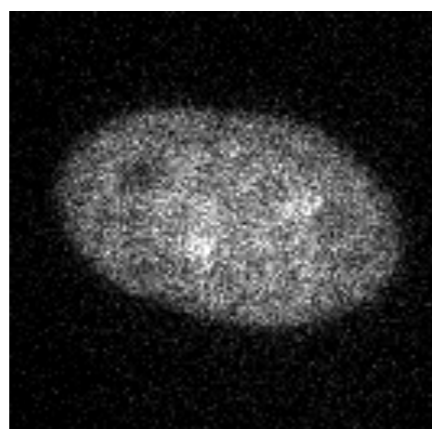
labeling a binary mask

refining a binary mask

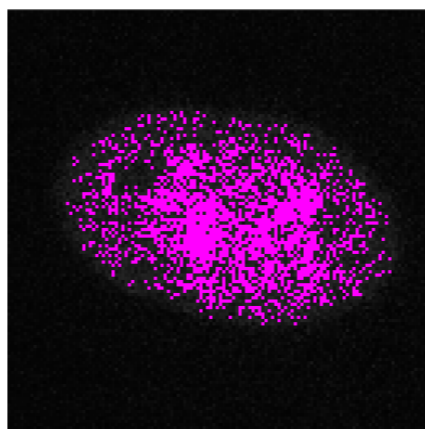




Image



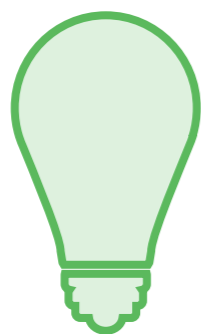
Threshold



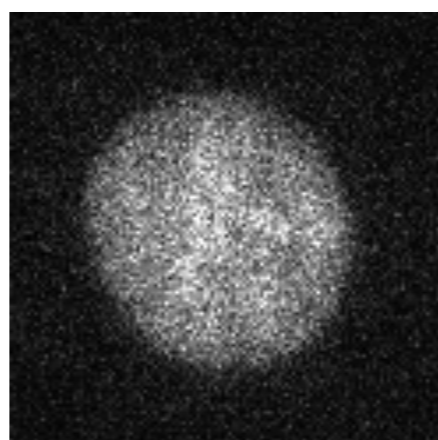
Binary Mask



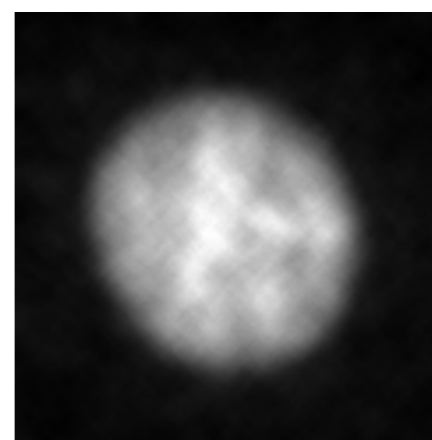
What if we apply an image filter before thresholding?



Image



Filtered Image



filtering

Change image pixel values using a ***mathematical operation*** to smooth and reduce noise from images.



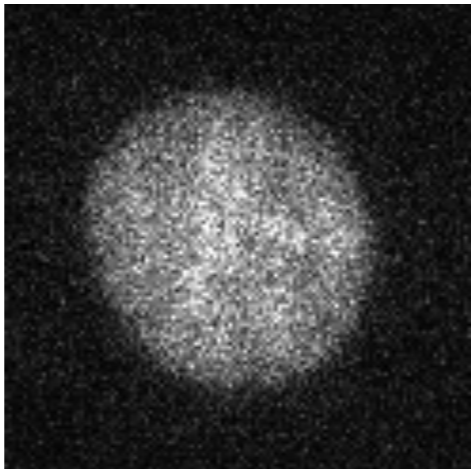


filtering

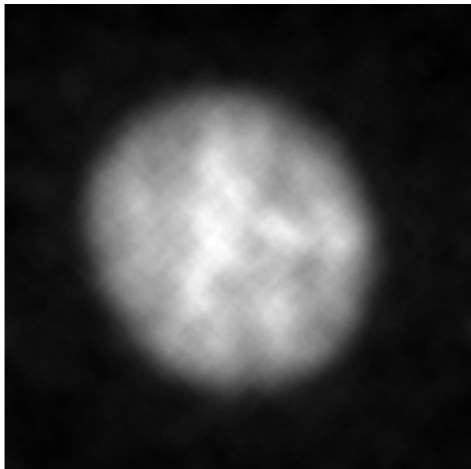
Change image pixel values using a **mathematical operation** to smooth and reduce noise from images.

we are mathematically changing this image's pixel values when we apply a filter.

Image



Filtered Image

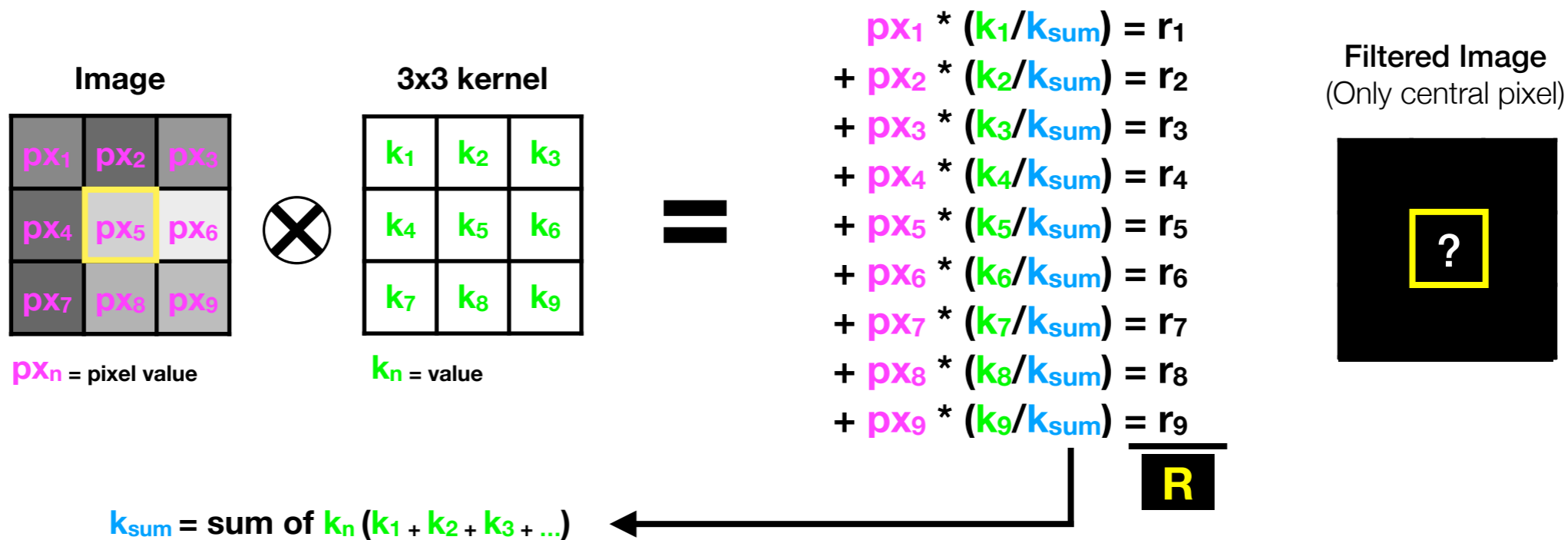




How most filters work mathematically

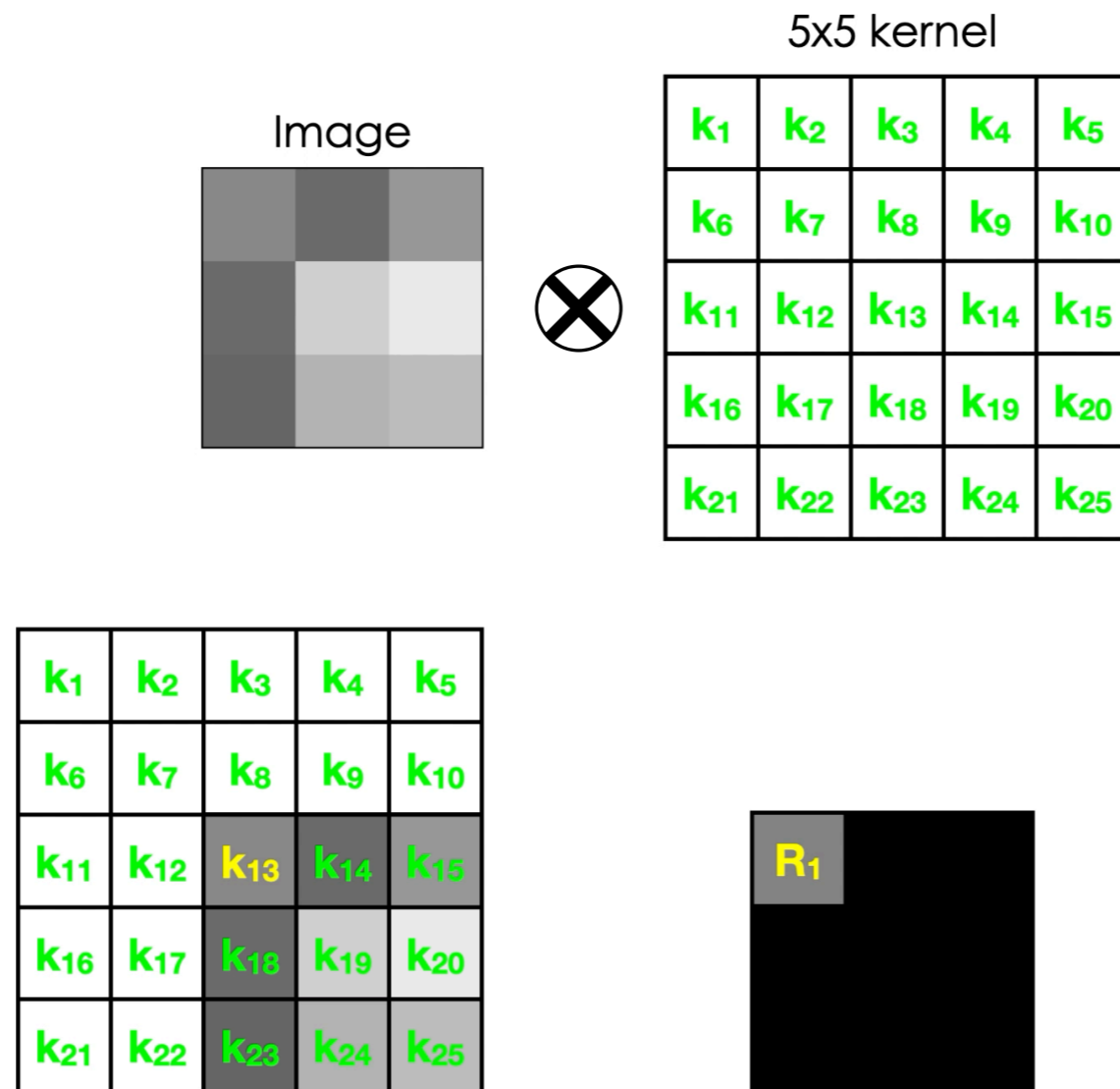


Convolve an image with a **3x3 kernel**





Convolve an image with a *5x5 kernel*



When you apply a filter, you can specify the kernel size you are convolving with.



examples of filters good at reducing noise

mean filter

Gaussian blur filter

median filter





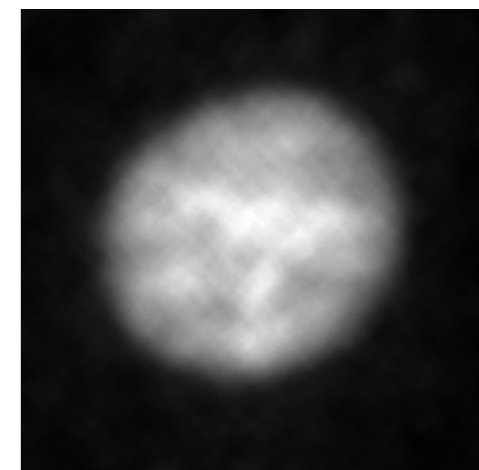
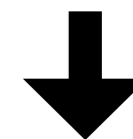
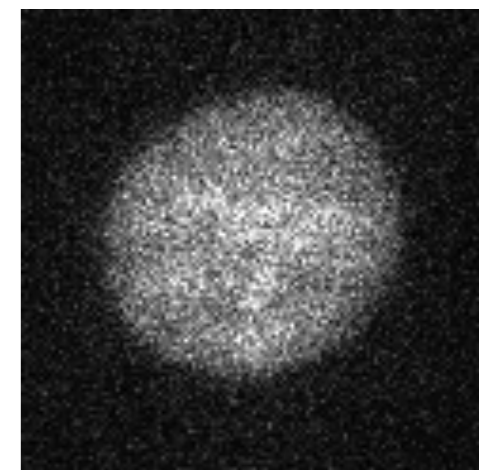
mean filter

sum values in a list and then divide by total number of values

Footprint refers to the kernel size.

3x3 kernel

1	1	1
1	1	1
1	1	1



Larger Footprint = Bigger Kernel = Higher Blur



examples of filters good at reducing noise



mean filter

Gaussian blur filter

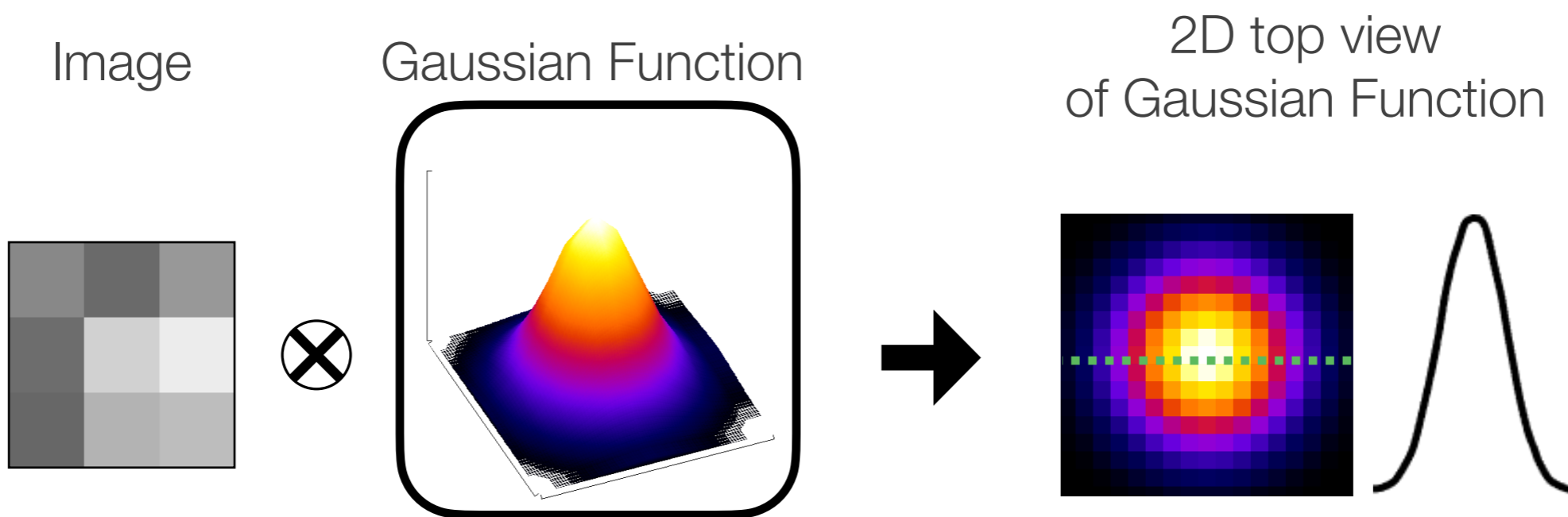
median filter





Gaussian blur filter

multiply each value by Gaussian profile weighting, then divide by total number of values

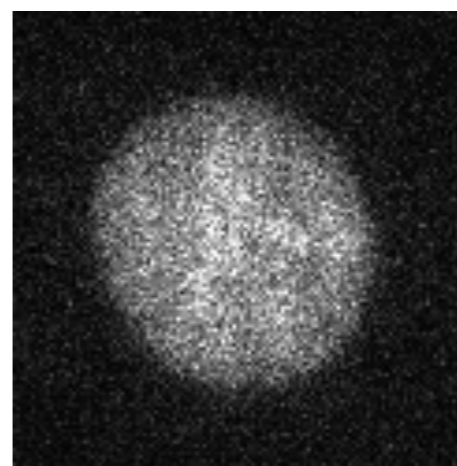


calculated using Gaussian function

sigma = 0.5 = 5x5 kernel



1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1



Higher Sigma = Bigger Kernel = Higher Blur



examples of filters good at reducing noise



mean filter



Gaussian blur filter

median filter

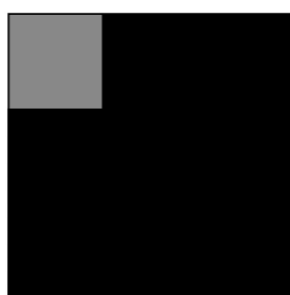
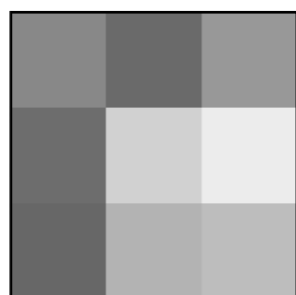




median filter

take the middle number in a sorted list of numbers

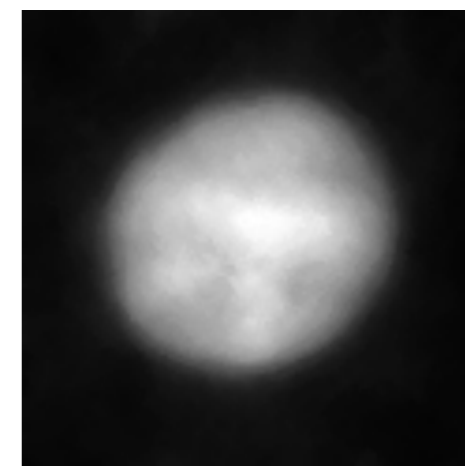
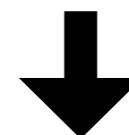
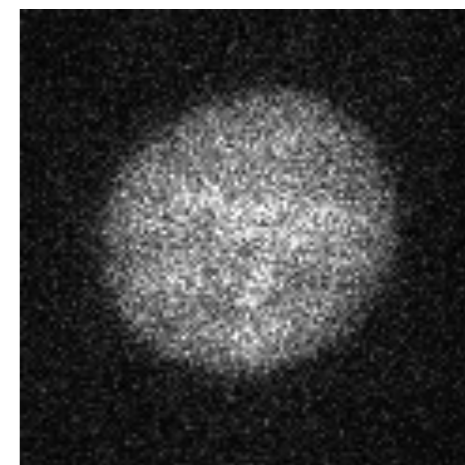
3x3 kernel



136	136	106	
136	136	106	152
109	109	209	236
	103	179	189

136		

↓
[106, 106, 109, 109, 136, 136, 136, 136, 209]

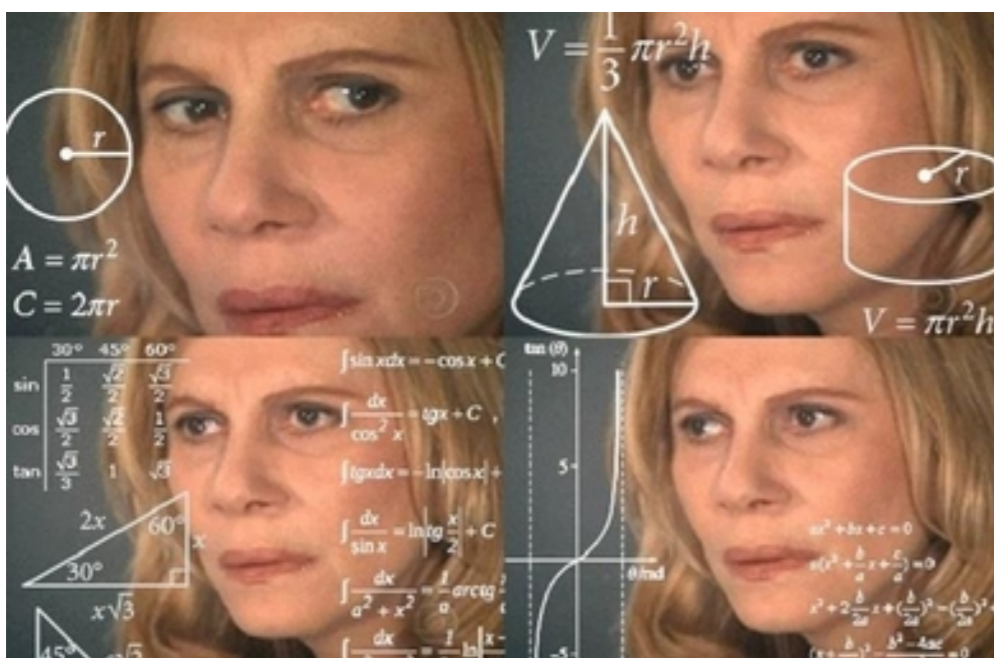


Bigger Kernel = Higher Blur
Median filters don't use convolution (non-linear filter!)



examples of filters good at reducing noise

- ✓ mean filter
- ✓ Gaussian blur filter
- ✓ median filter



Thinking about filter math can take some time to get used to.



Choose the filter parameters that give you the best binary mask result

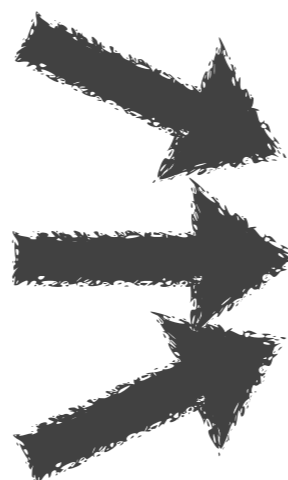


test different filters

mean filter

gaussian blur filter

median filter



...and different kernel sizes

choose the combination that gives you the best binary mask result





Filtering in Python



Gaussian blur filter

```
[ ]: # use Gaussian blur filter  
from skimage.filters import gaussian  
filtered_image = gaussian(raw_image,  
                           sigma = 1)
```

thresholding algorithm

```
[ ]: # use Otsu thresholding algorithm  
binary_mask = filtered_image >  
              threshold_otsu(filtered_image)
```





Now what do we do with the binary mask?

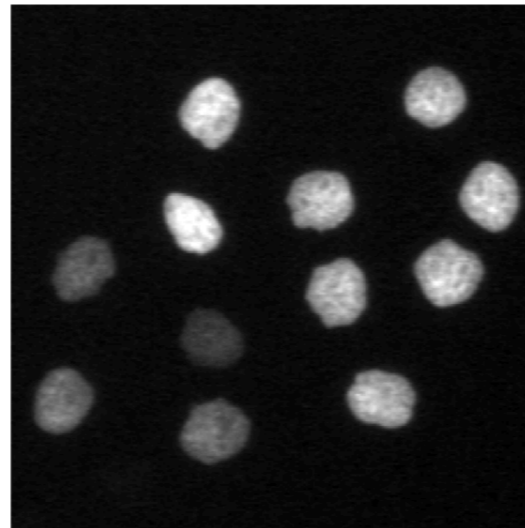




2 types of segmentation: Semantic & Instance



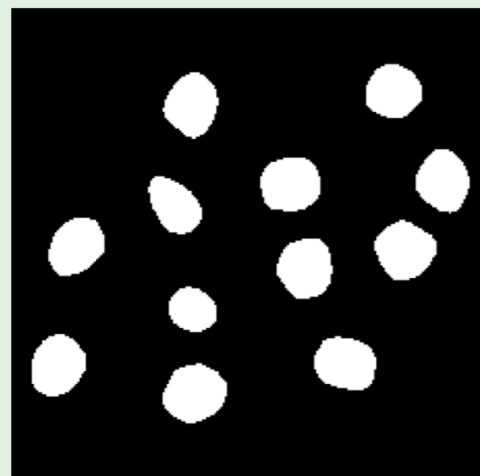
Raw Image



Semantic Segmentation

All objects treated as the same category

Example:



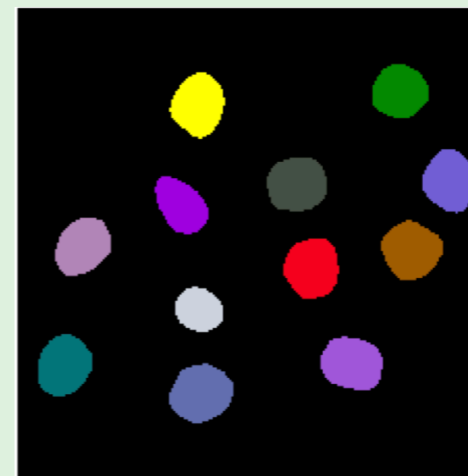
Categories:

Nuclei

Instance Segmentation

Each object is distinguished as separate and has a unique label

Example:



Categories:

Nucleus 1

Nucleus 2

Nucleus 3

...

We need to do another step to accomplish this.





classic segmentation with Python

filtering

thresholding

~~filtering~~

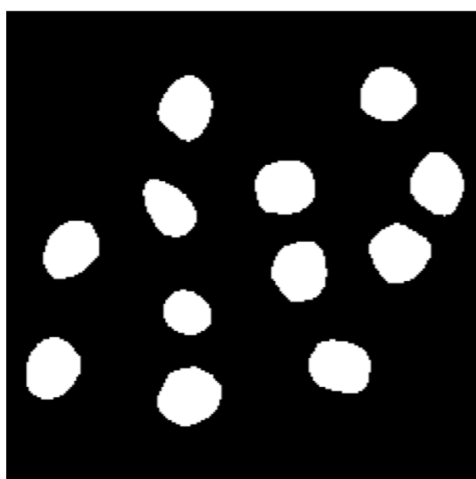
labeling a binary mask

refining a binary mask

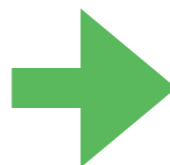




Now that we have a binary mask, we need a way to *distinguish individual objects of interest* in the mask



Categories:
Nucleus



Categories:
Nucleus 1
Nucleus 2
Nucleus 3
...





Labeling a mask in Python



labeling a mask

```
[ ]: from skimage.measure import label  
labeled_image = label(binary_mask)
```



all pixel
values are the
label value





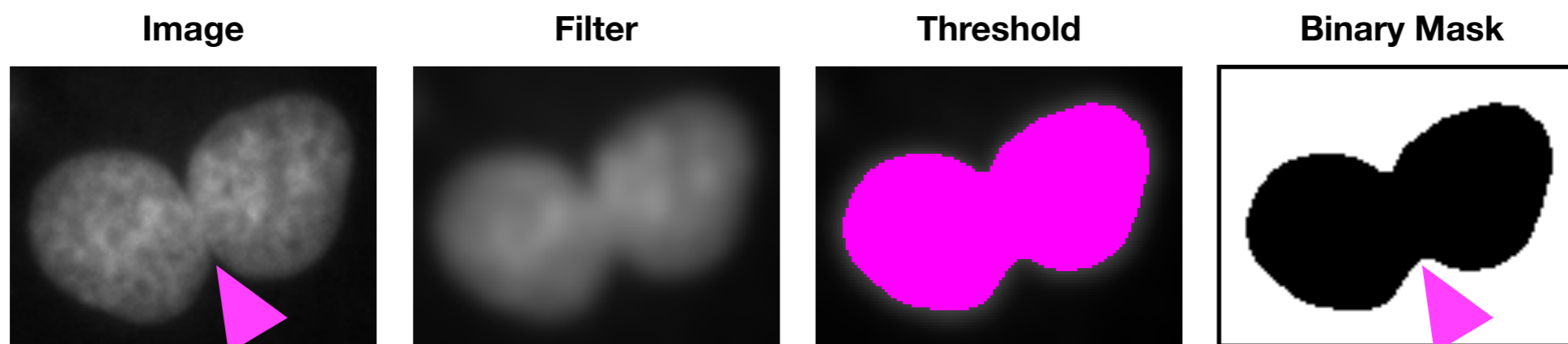
Lab:

Classic Segmentation Notebook, Steps 0-4





What if the binary mask isn't perfect?



2 or more nuclei in the binary mask image are touching each other, resulting in them being considered as a single object.





classic segmentation with Python

filtering

thresholding

~~filtering~~

labeling a binary mask

refining a binary mask





Sometimes binary masks need to be *refined*



mask refinement: additional processing steps applied to a binary mask to more accurately match the image foreground.

common problems:

miscellaneous schmutz



holes inside objects



touching round objects



Morphological operations: a family of algorithms that are helpful for modifying object shapes

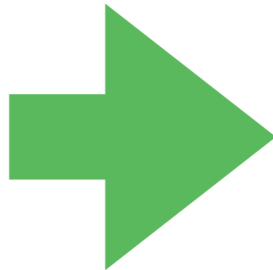
Watershed Algorithm: useful for separating touching objects.





miscellaneous schmutz

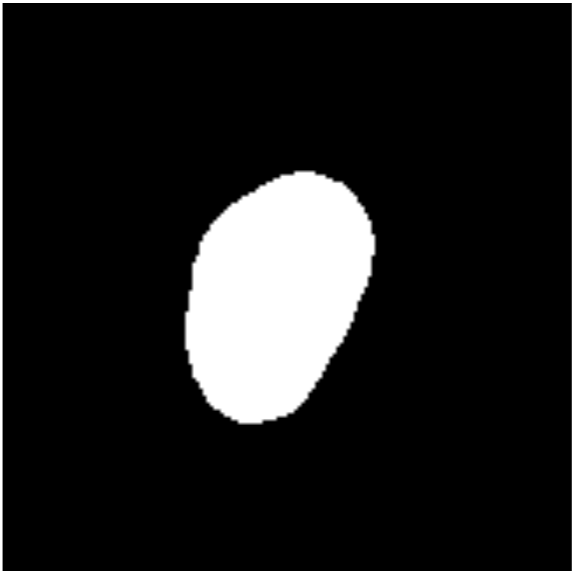
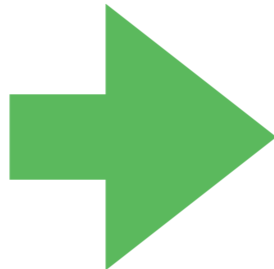
Morphological operations: a family of algorithms that are helpful for modifying object shapes



opening

~ removes small objects from foreground

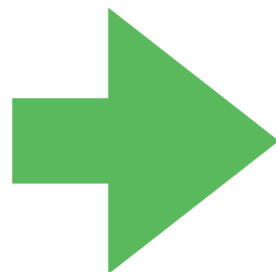
```
[ ]: from skimage.morphology import remove_small_objects  
binary_mask_sized = remove_small_objects(  
    binary_mask, min_size=10)
```





holes inside objects

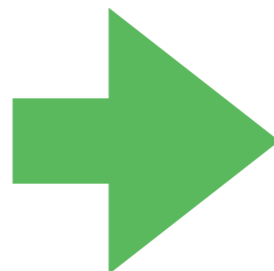
Morphological operations: a family of algorithms that are helpful for modifying object shapes



closing

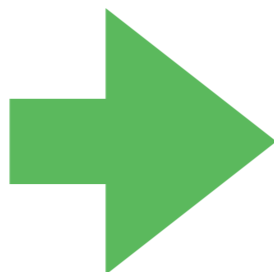
~ fills small holes in foreground

```
[ ]: from skimage.morphology import closing, disk  
binary_mask_sized = closing(binary_mask, disk(1))
```





touching objects

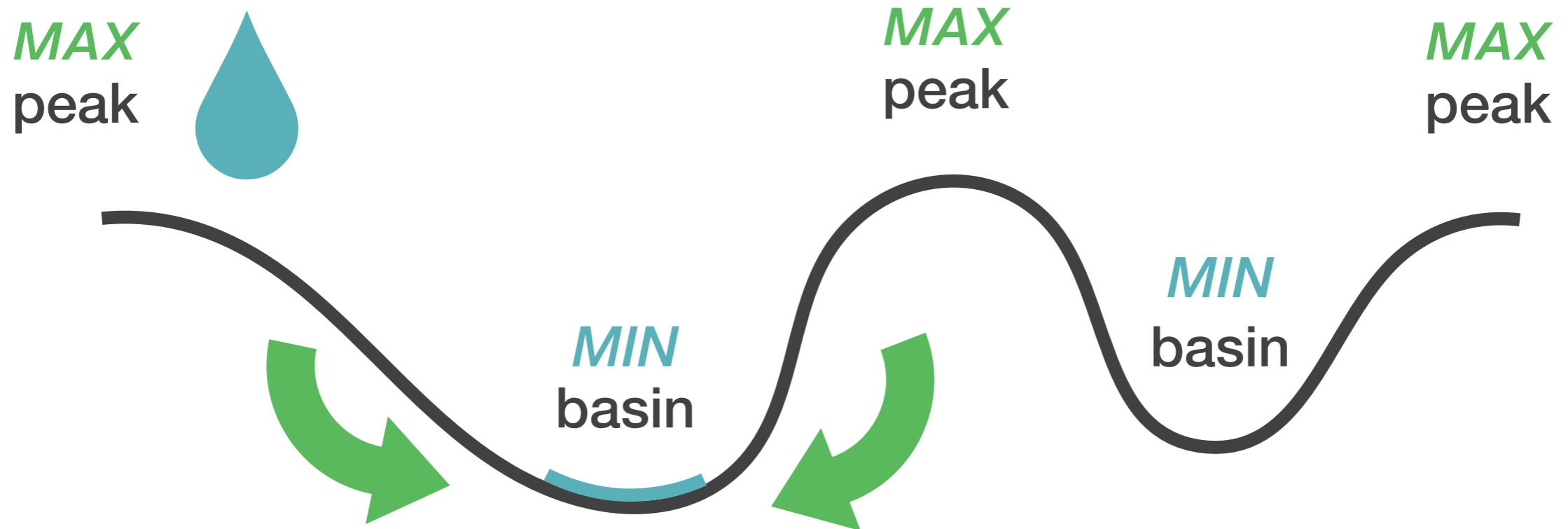


Watershed Algorithm: useful for separating touching round objects.



What *is* the watershed algorithm?

The name **watershed** is inspired by how a drop of water falls along a surface



the drop flows to the nearest low point, called a **basin**



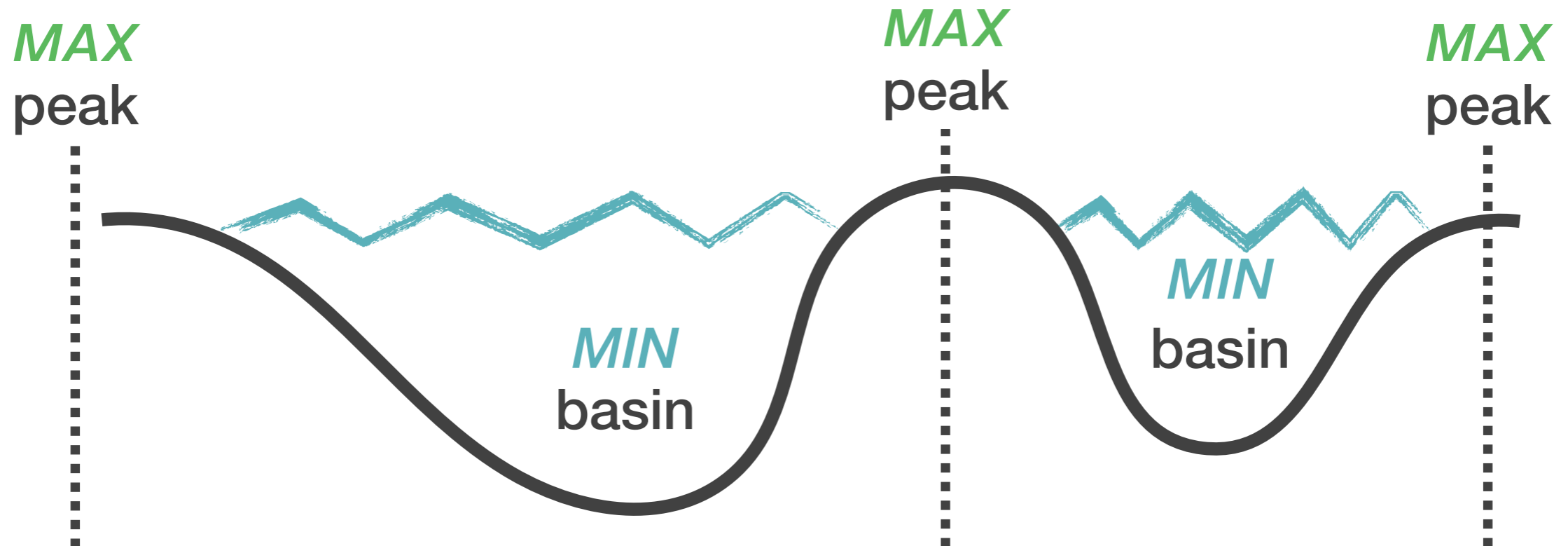


What *is* the watershed algorithm?

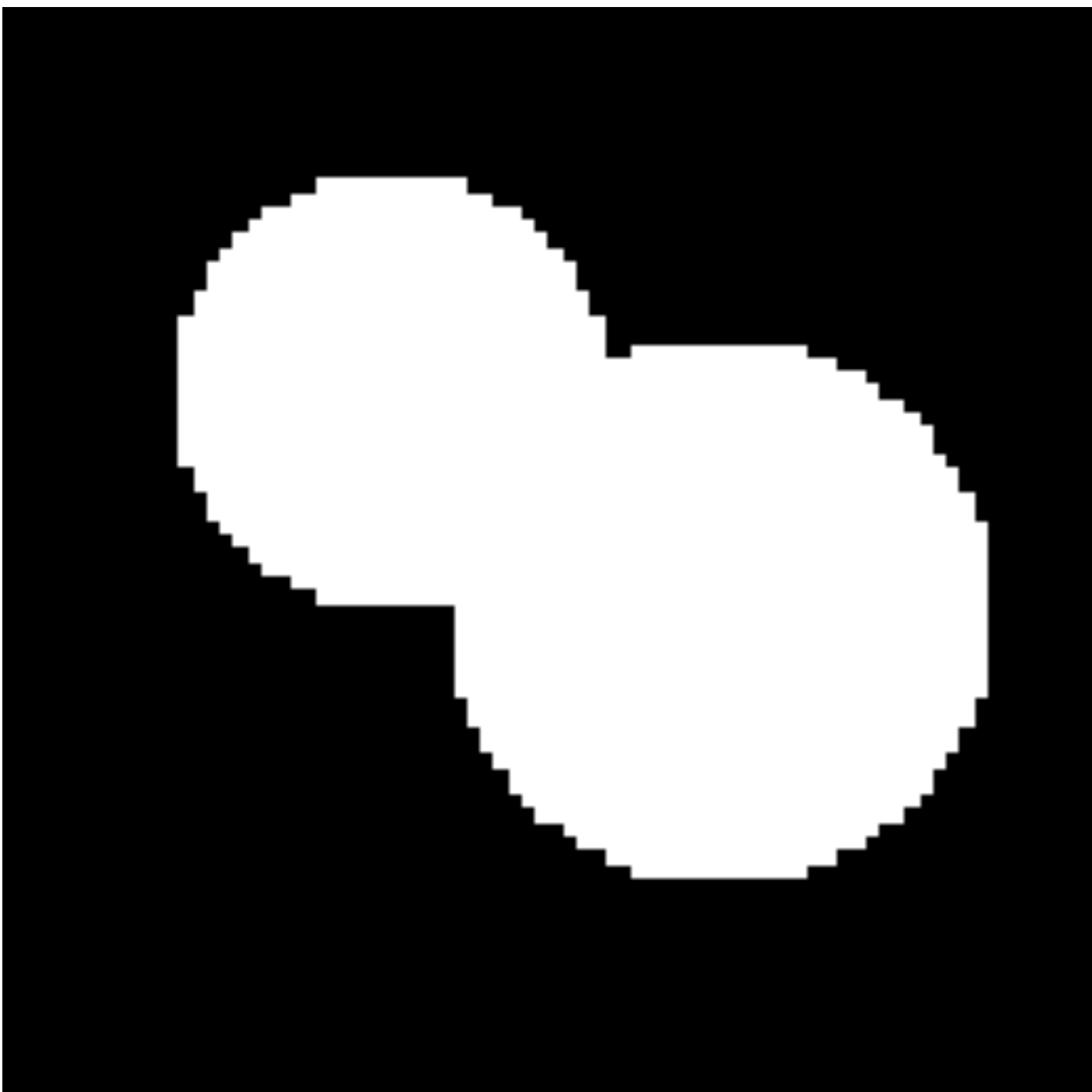


The name **watershed** is inspired by how a drop of water falls along a surface

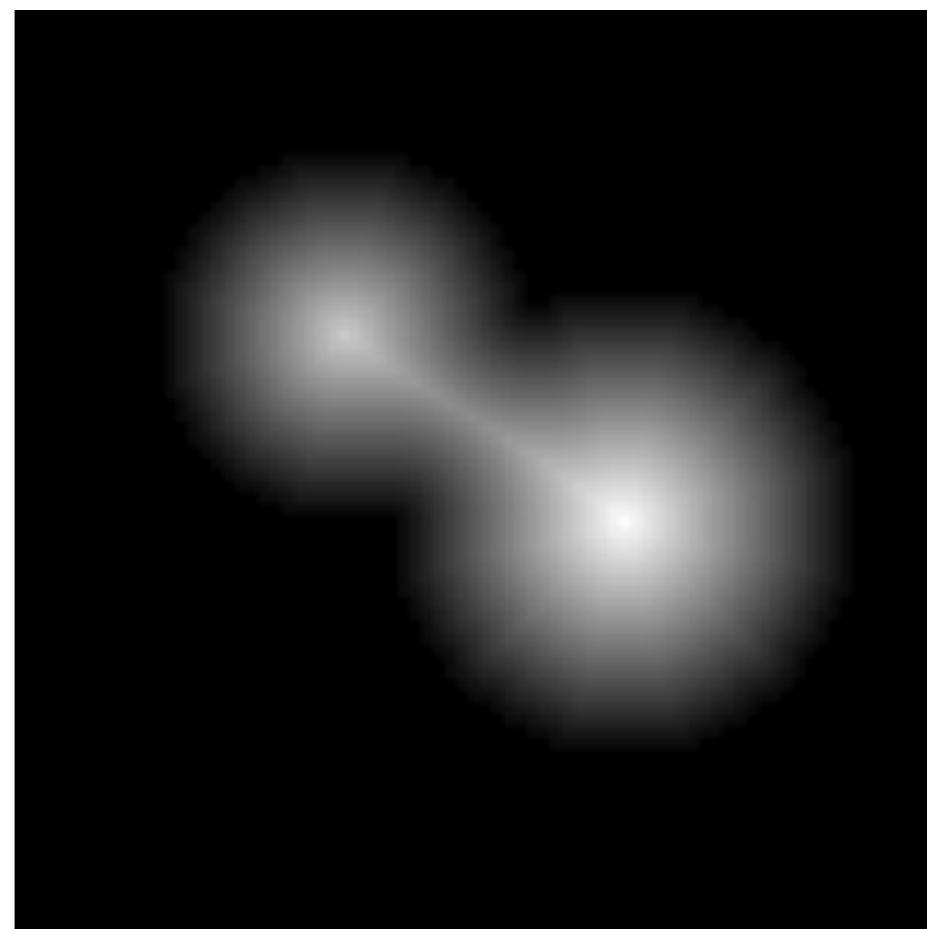
the **watershed line** separates which basin the water will go to



What *is* the watershed algorithm?



Calculate how far each white pixel is from the nearest black pixel



the resulting image is called a *distance transform*

What *is* the watershed algorithm?



Let's study this distance transform...

The largest distance values are in the *centers* of the 2 objects

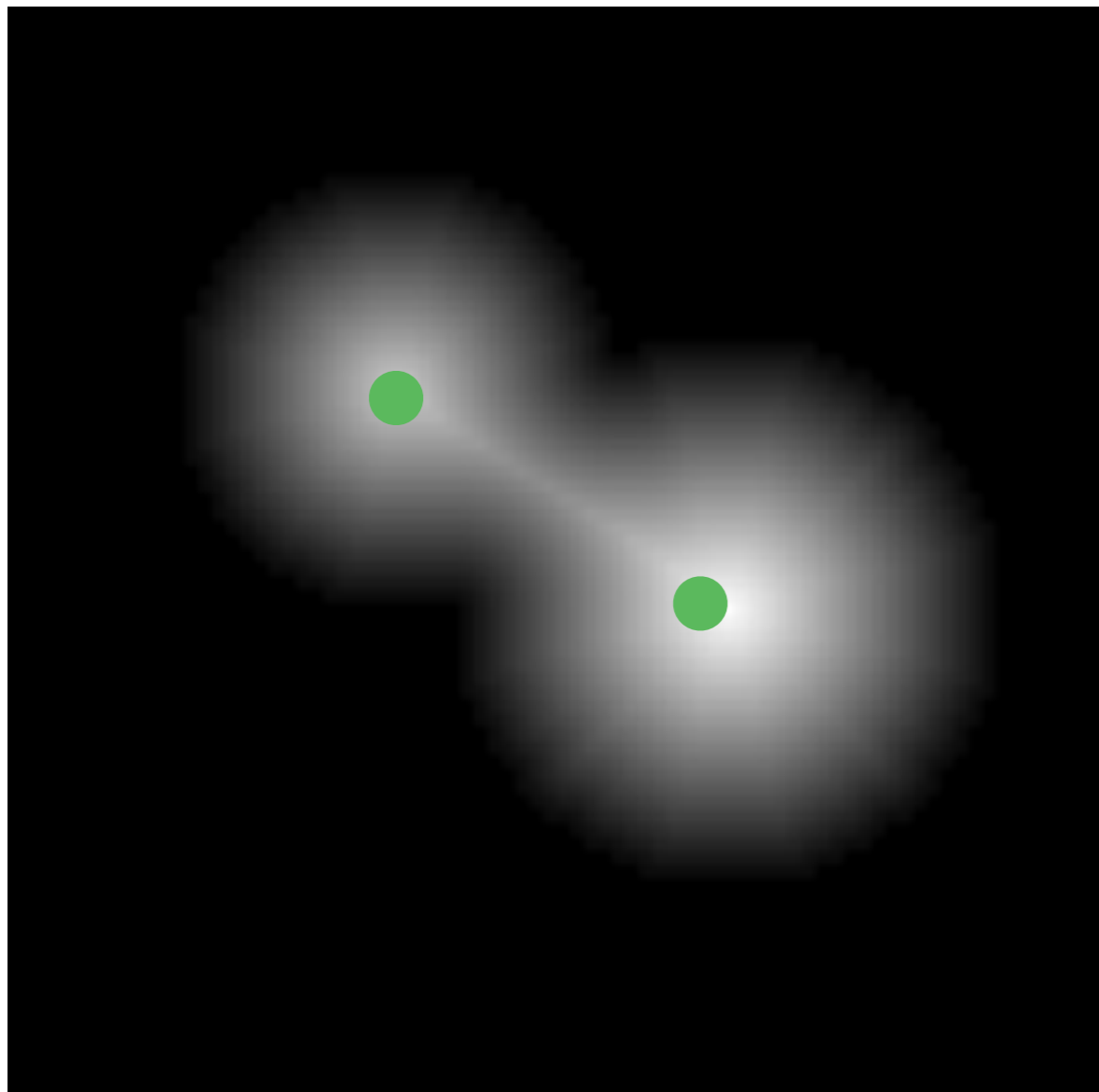
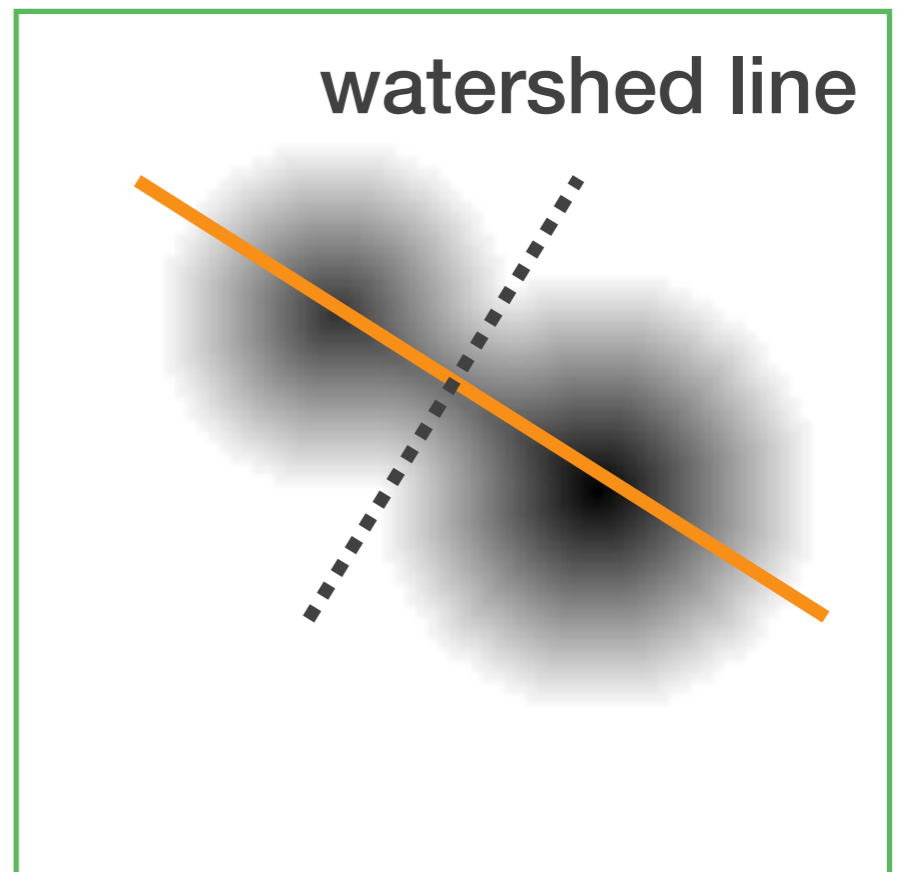
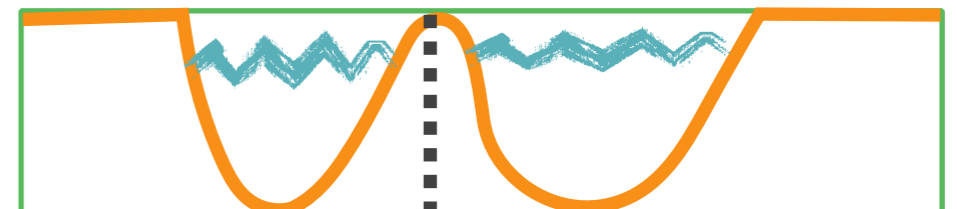


image of *inverted distances*



profile



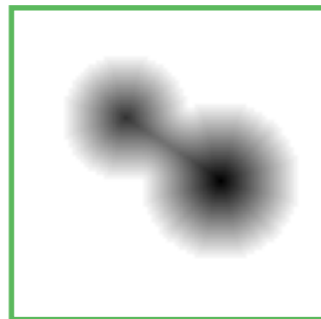
What *is* the watershed algorithm?



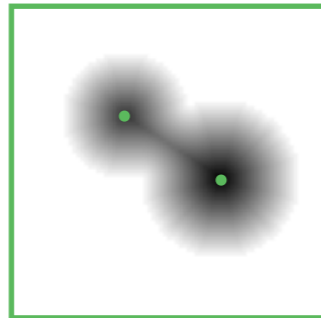
the **watershed algorithm**, in summary:

inputs

-distance
transform



labeled
peaks
(seeds)

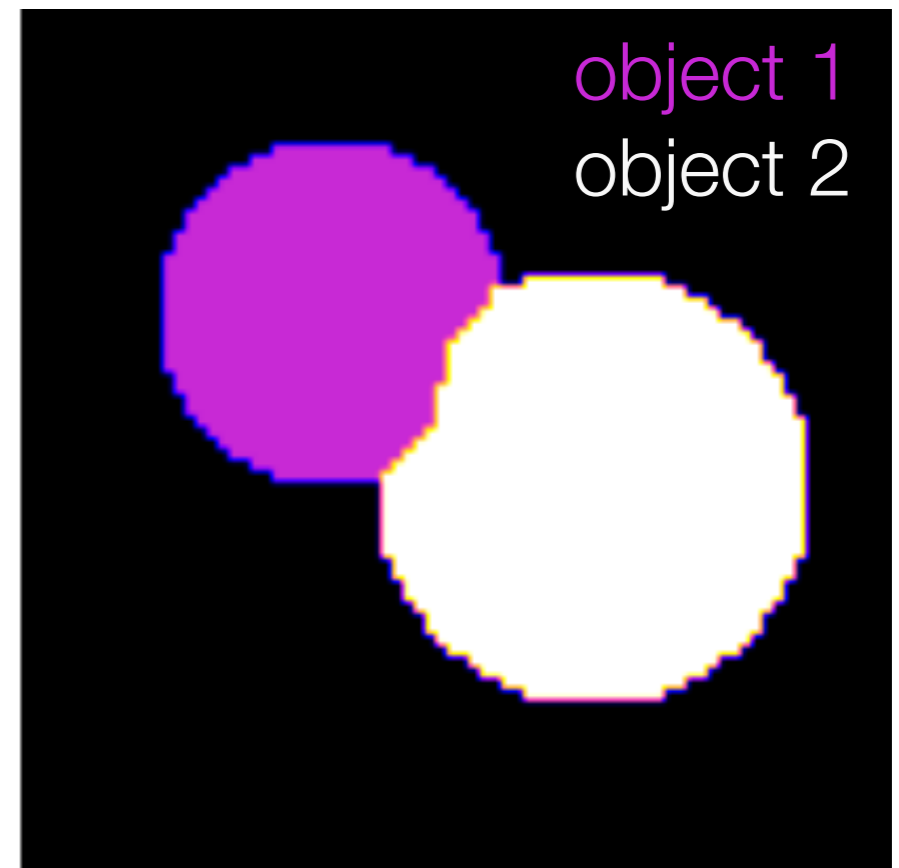


binary
mask



output

labeled image with
separated objects



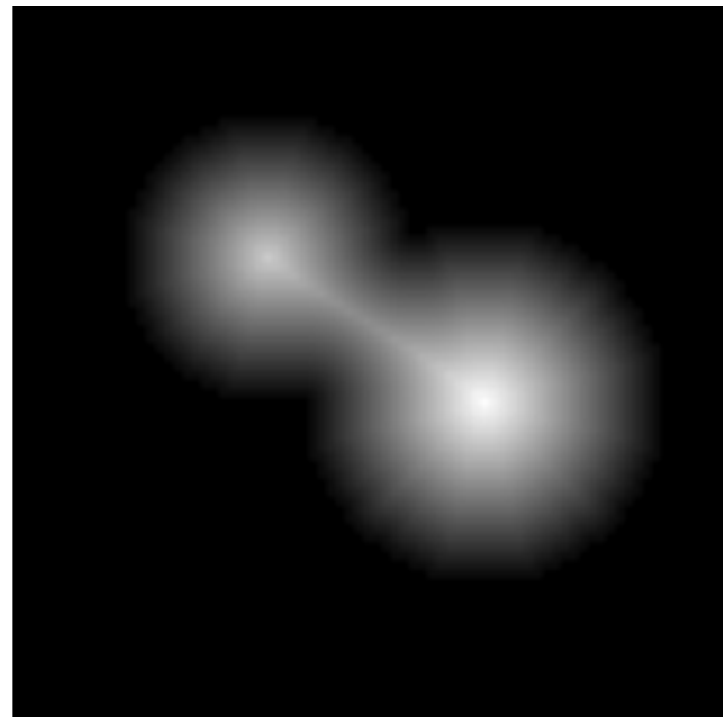


Watershed in Python



calculate the distance transform

```
[1]: # calculate distance transform
from scipy.ndimage import distance_transform_edt
distance_transform =
        distance_transform_edt(binary_image)
```





Watershed in Python



find the peak coordinates in the distance transform

```
[ ]: # find the peak distances in distance_transform
from skimage.feature import peak_local_max
import numpy as np
peak_coords = peak_local_max(distance_transform,
                             footprint=np.ones((25,25)),
                             min_distance=10)
```





Watershed in Python



create a labeled image with the peaks

```
[ ]: # create img that's same size as binary_mask  
local_maxima_image = np.zeros_like(  
    binary_mask, dtype=bool)  
# add peak_coords to img  
local_maxima_image[  
    tuple(local_maxima_coords.T)] = True
```





Watershed in Python



label the image with the peaks to create seeds

```
[ ]: # label local_maxima_image  
seeds = label(local_maxima_image)
```





Watershed in Python



give the watershed algorithm the distance transform, the seeds, and the binary mask

```
[ ]: # perform watershed  
from skimage.segmentation import watershed  
labeled_image = watershed(  
    -distance_transform,  
    seeds,  
    mask = binary_mask)
```

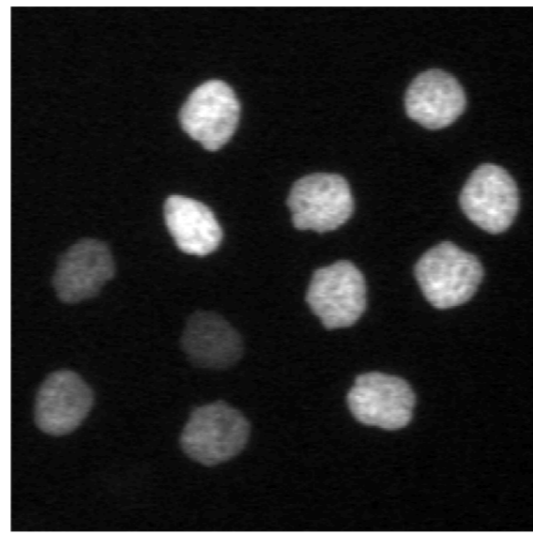




2 types of segmentation: Semantic & Instance



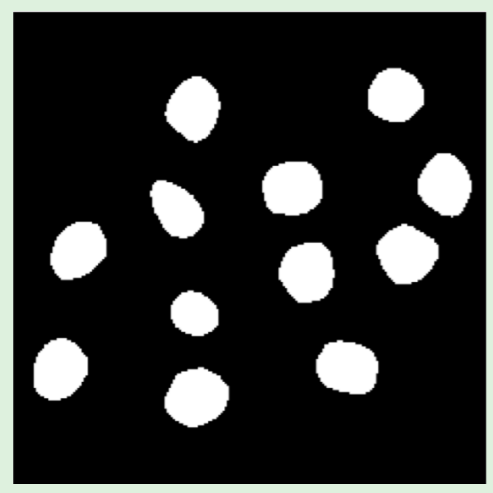
Raw Image



Semantic Segmentation

All objects treated as the same category

Example:



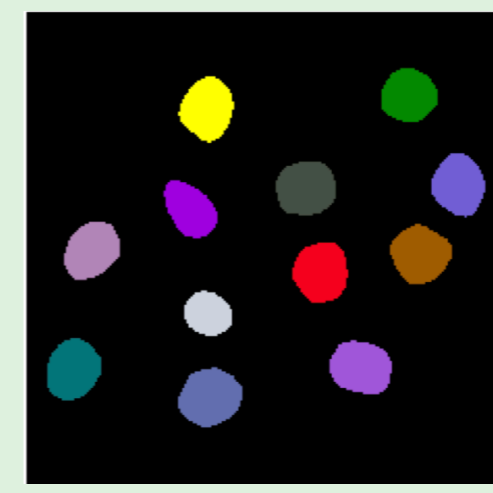
Categories:
Nuclei



Instance Segmentation

Each object is distinguished as separate and has a unique label

Example:



Categories:
Nucleus 1
Nucleus 2
Nucleus 3
...





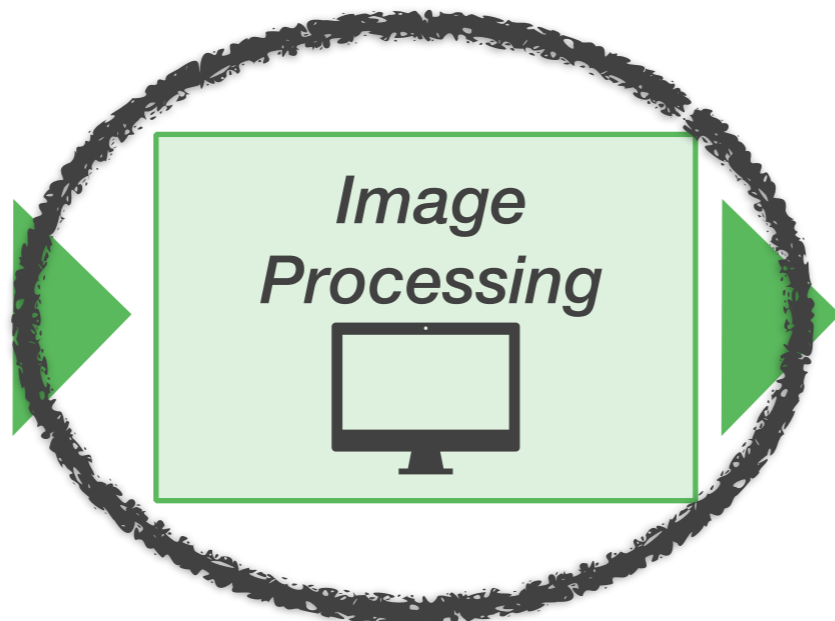
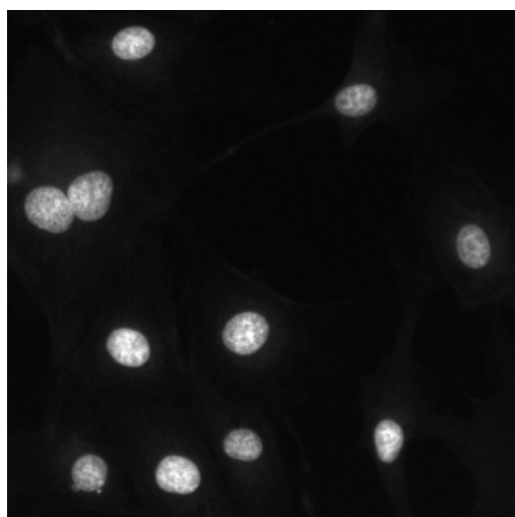
Lab:

Classic Segmentation Notebook, Step 5





Images



Results



Statistically relevant & reproducible measurements come from analyzing ***many fluorescence images.***





We need to analyze enough images to represent an entire cell population

12 widefield images to analyze





classic segmentation with Python

processing many images





Processing many images in Python



We use a for loop to run same processing steps on each image!





Processing many images in Python



Organize all images to process in 1 folder.

Use a **for** loop to loop through image paths.

```
[ ]: # loop through image paths to get each image  
from pathlib import Path  
folder_dir = Path("/Users/edelase/bobiac")  
for image_path in folder_dir.iterdir():  
    # do classic processing steps
```





Processing many images in Python



Use a **for** loop to loop through *only tif* image paths in a folder.

```
[ ]: # loop through only tif image paths  
from pathlib import Path  
folder_dir = Path("/Users/edelase/bobiac")  
for image_path in folder_dir.glob("*.tif"):  
    # do classic processing steps
```





what about saving images?





Saving images in Python



Use a `tifffile.imwrite()` to save output images.

```
[ ]: # save an image
import tifffile
from pathlib import Path
output_dir = Path("/Users/edelase/output")
output_filepath = output_dir /
                  "output_file.tif"
tifffile.imwrite(output_filepath,
                 data=image.astype("uint32"))
```

Why *won't* this work in our image paths for loop?





Saving images in Python



Use `f"{image_path.stem}_.tif"` to automatically generate file names for each loop

```
[ ]: # save an image
import tifffile
from pathlib import Path

input_dir = Path("/Users/edelase/input")
output_dir = Path("/Users/edelase/output")
for image_path in input_dir.glob("*.tif"):
    output_filepath = output_dir /
        f"{image_path.stem}_.tif"
    tifffile.imwrite(output_filepath,
        data=image.astype("uint32"))
```





Lab:

Classic Segmentation Notebook, Step 6



We made it!



classic segmentation with Python

processing many images





BoBiAC Exercises: Classic Segmentation



General resources



Documentation



scikit-image.org

Image Analysis Forum



<https://forum.image.sc/>

Learning Resource



[https://
bioimagebook.github.io/
index.html](https://bioimagebook.github.io/index.html)





Questions?

