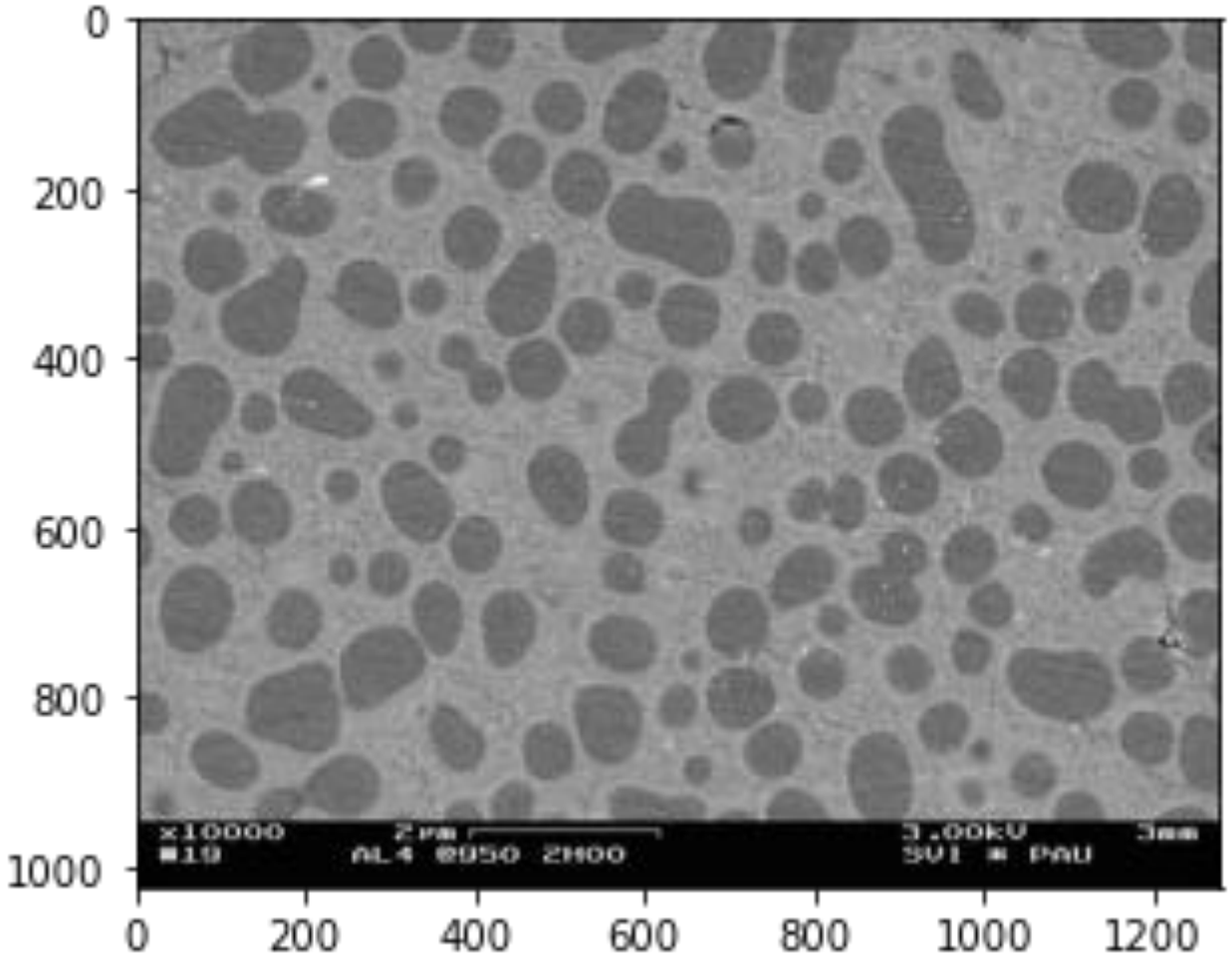
REGIONS IN IMAGE

Nada Fitrieyatul Hikmah

Image segmentation: extracting objects from images



1) Open image : electron microscopy image

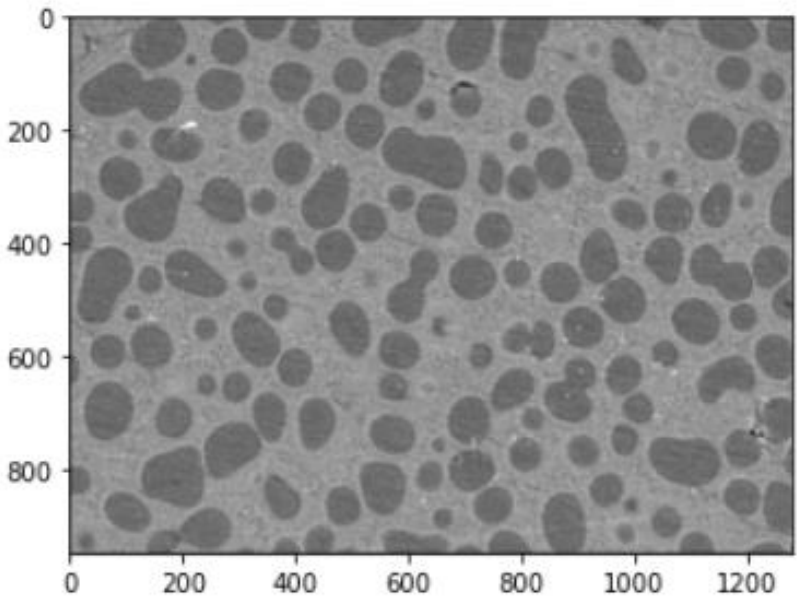
```
%matplotlib inline
import imageio
from matplotlib import pyplot as plt
import numpy as np

im = imageio.imread('D:\Teaching\Pengolahan Citra Me
plt.imshow(im, cmap='gray')
plt.show()
im.dtype, im.shape
```

Pre-processing (1)

Removes the information bar at the bottom, in order to retain only the region of the image with the blobs of interest.

```
1 phase_separation = im[:947]
2 plt.imshow(phase_separation, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1498f616be0>
```



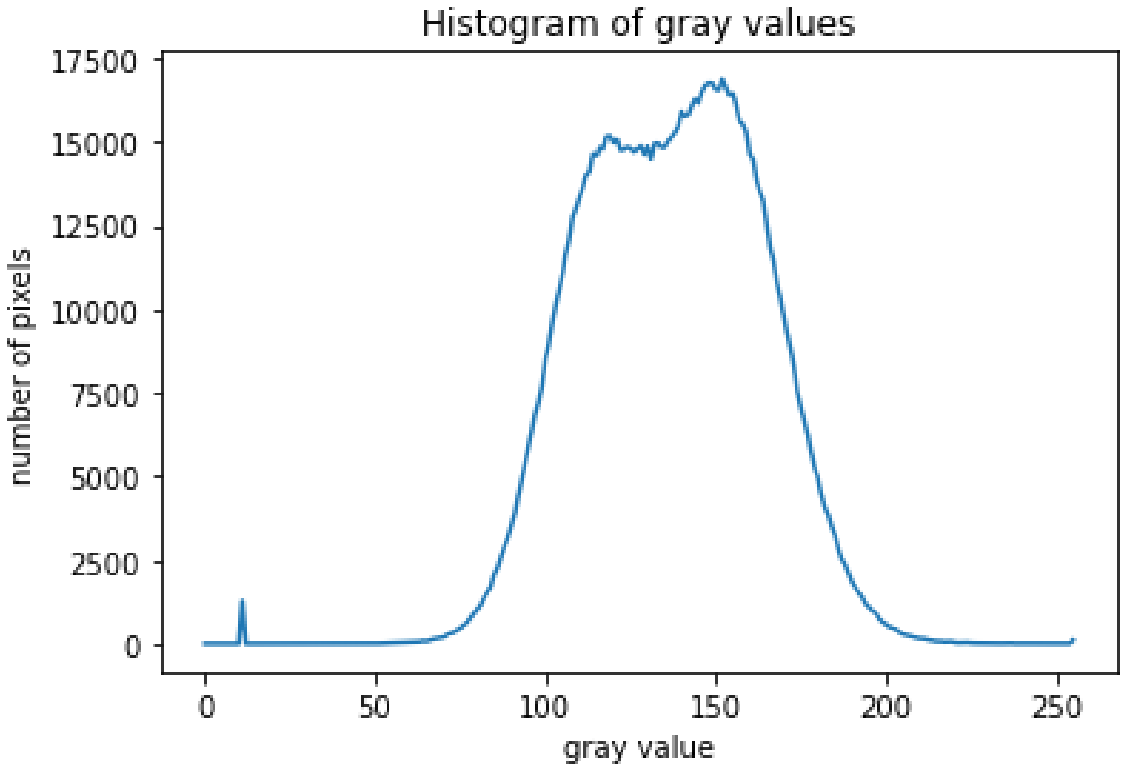
Pre-processing (2) - Histogram

- In order to separate blobs from the background, a simple idea is to use the gray values of pixels: blobs are typically darker than the background.
- In order to check this impression, look at the histogram of pixel values of the image.

```python
from skimage import exposure
import scipy.ndimage as ndi

histogram = ndi.histogram(phase_separation, min=0, max=255, bins=256)

#histogram = exposure.histogram(phase_separation)
plt.plot(histogram)
plt.xlabel('gray value')
plt.ylabel('number of pixels')
plt.title('Histogram of gray values')
```

Pre-processing (3) - Histogram

- Two peaks are clearly visible in the histogram, but they have a strong overlap. What happens if we try to threshold the image at a value that separates the two peaks?
- For an automatic computation of the thresholding values, we use Otsu's thresholding, an operation that chooses the threshold in order to have a good separation between gray values of background and foreground.



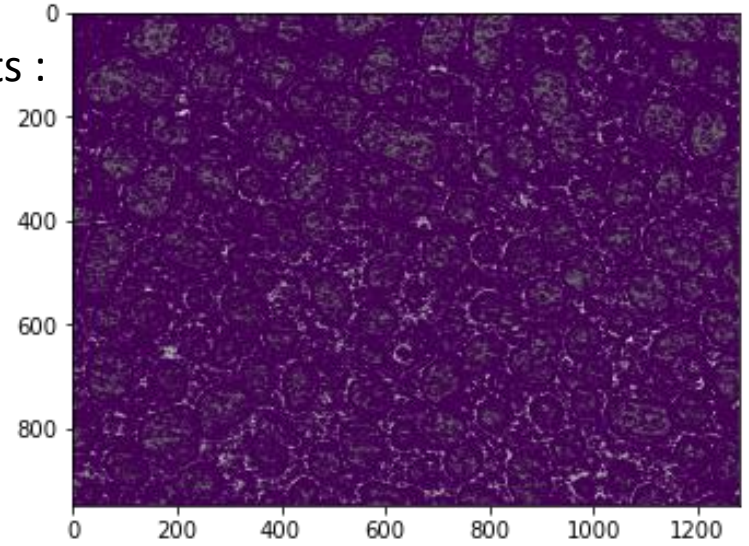
Pre-processing (4) – Thresholding

```
1  from skimage import filters
2
3  threshold = filters.threshold_otsu(phase_separation) #deteksi otomatis nilai threshold yg pas
4  threshold
```

```
136
```

```
1  fig, ax = plt.subplots(ncols=2, figsize=(12, 8))
2  ax[0].imshow(phase_separation, cmap='gray')
3  ax[0].contour(phase_separation, [threshold]) #contour pada daerah threshold
4  ax[1].imshow(phase_separation < threshold, cmap='gray')
```

Segmentation results :
(Good enough?)



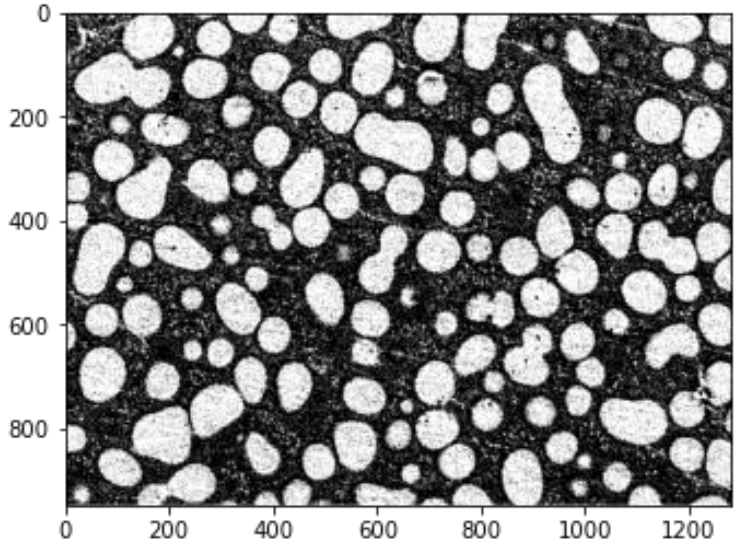


Image Denoising (1)

- In order to improve the thresholding, we will try first to filter the image so that gray values are more uniform inside the two phases, and more separated. Filters used to this aim are called *denoising filters*, since their action amounts to reducing the intensity of the noise on the image.
- Zooming on a part of the image that should be uniform illustrates well the concept of noise: the image has random variations of gray levels that originate from the imaging process. Noise can be due to low photon-counting, or to electronic noise on the sensor, although other sources of noise are possible as well.

```
1 plt.imshow(phase_separation[390:410, 820:840], cmap='gray')
2 plt.colorbar()
3 print(phase_separation[390:410, 820:840].std())
```

```
17.67220911487865
```

Image Denoising (2) – Check noise!

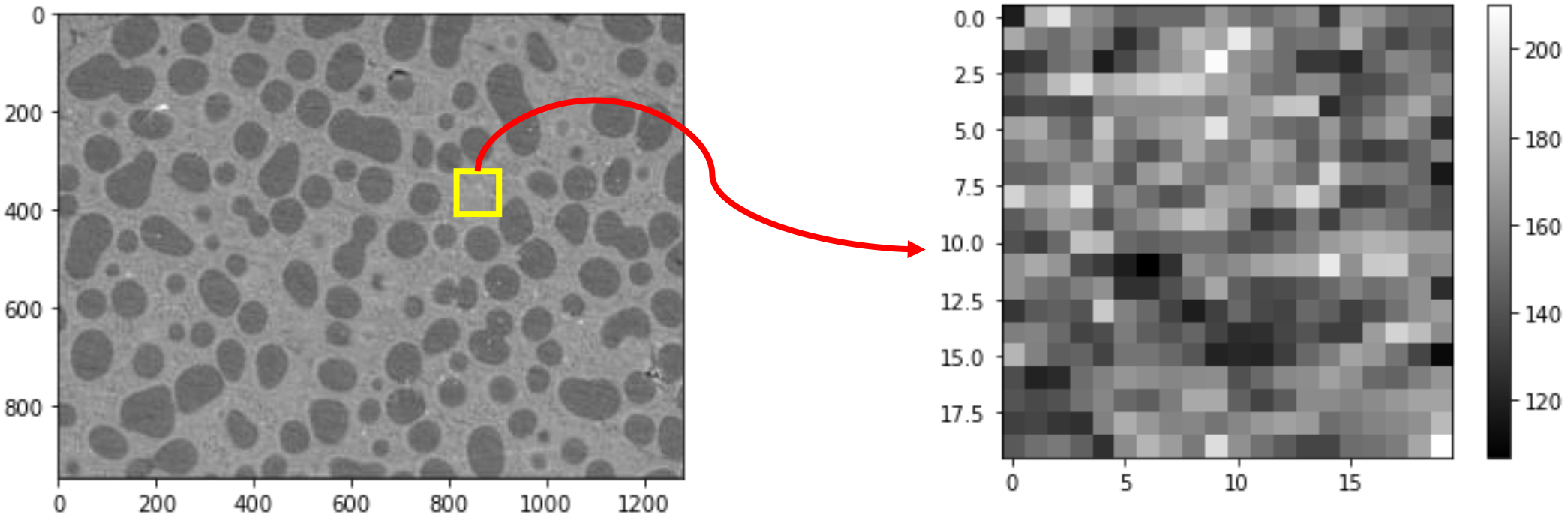


Image Denoising (3)

- Several denoising filters average together pixels that are close to each other. If the noise is not spatially correlated, random noise fluctuations will be strongly attenuated by this averaging.
- *Median filter* : it replaces the value of a pixel by the median gray value inside a neighbourhood of the pixel. Taking the median gray value preserves edges much better than taking the mean gray value.
- Use a square neighbourhood of size 7x7: the larger the window size, the larger the attenuation of the noise, but this may come at the expense of precision for the location of boundaries. Choosing a window size therefore represents a trade-off between denoising and accuracy.

```
1  from skimage import filters
2
3  median_filtered = filters.median(phase_separation, np.ones((7, 7)))
4  plt.imshow(median_filtered, cmap='gray')
5
6  cek = np.ones((7,7))
7  cek
```

```
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

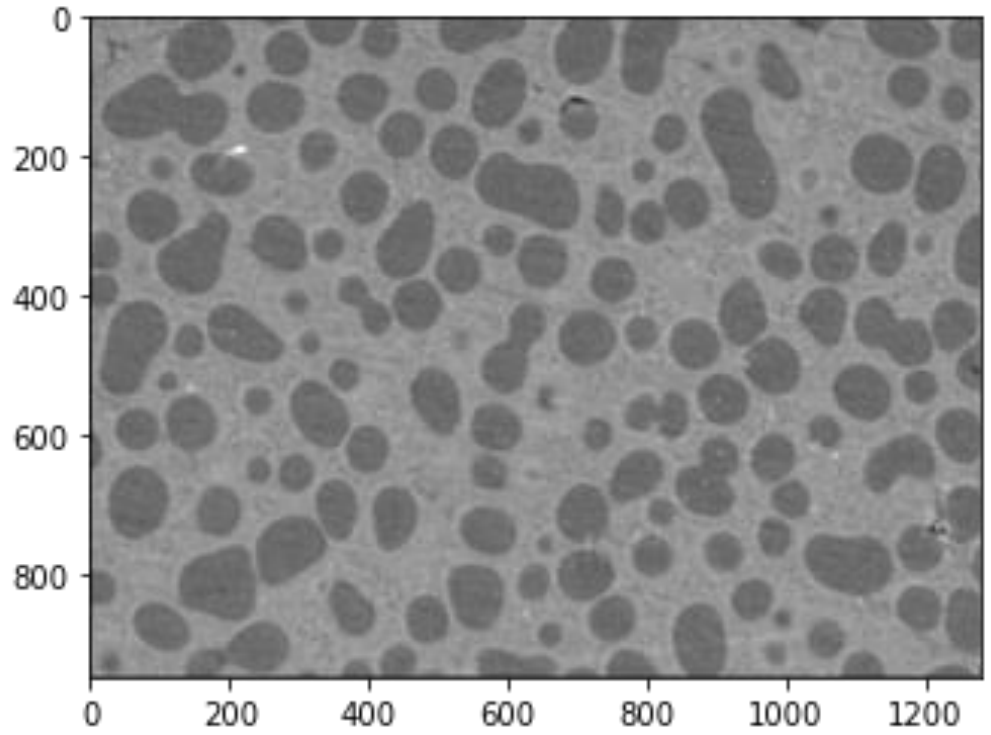


Image Denoising (4) – Check filter effect!

- Variations of gray levels inside zones that should be uniform are now smaller in range, and also spatially smoother.

```
plt.imshow(median_filtered[390:410, 820:840], cmap='gray',
           interpolation='nearest')
plt.colorbar()
print(median_filtered[390:410, 820:840].std())
```

```
5.533660632890312
```

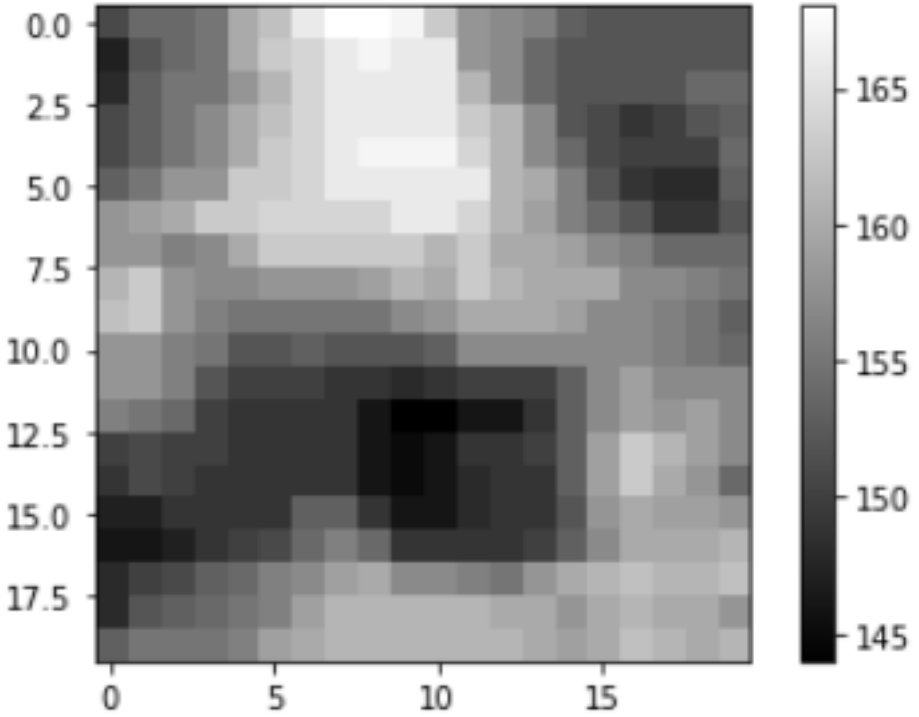
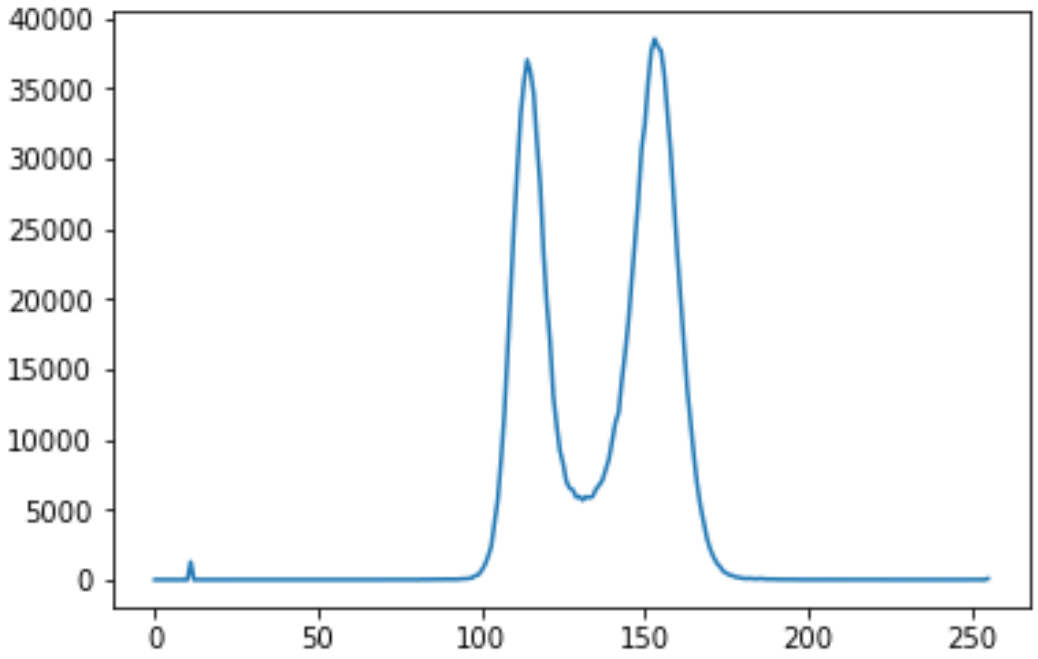


Image Denoising (5) – Check filter effect!

- Plotting the histogram of the denoised image shows that the gray levels of the two phases are now better separated.



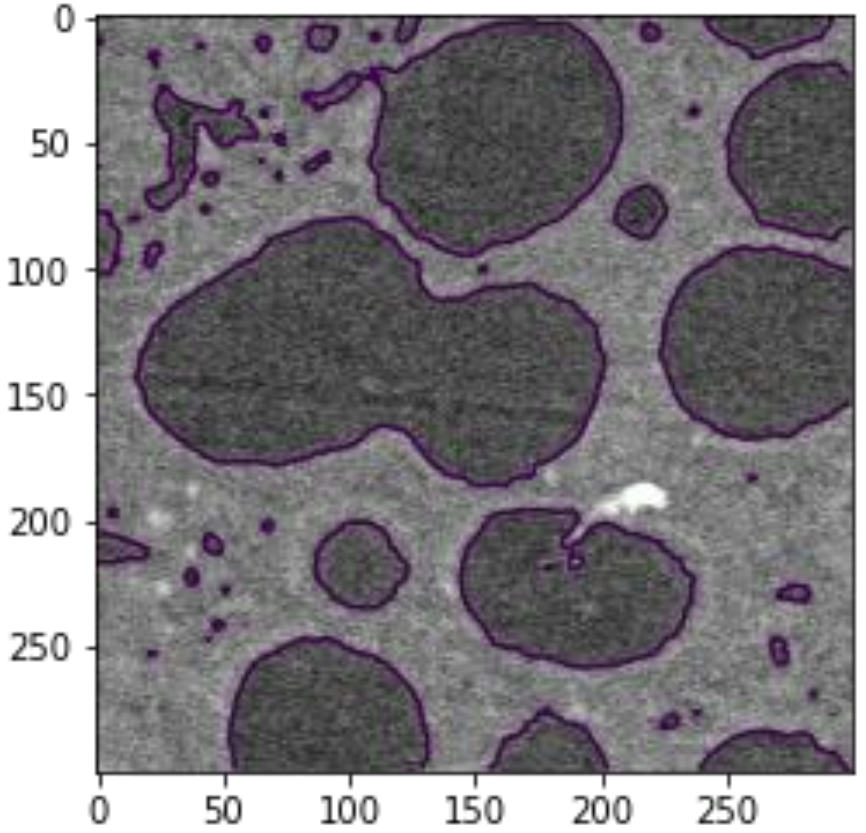

```
histo_median = ndi.histogram(median_filtered, min=0, max=255, bins=256)
plt.plot(histo_median)
```

Image Segmentation

- As a consequence, Otsu thresholding now results in a much better segmentation.
- For images with non-uniform illumination, it is possible to extend Otsu's method to the case for which different thresholds are used in different regions of space.

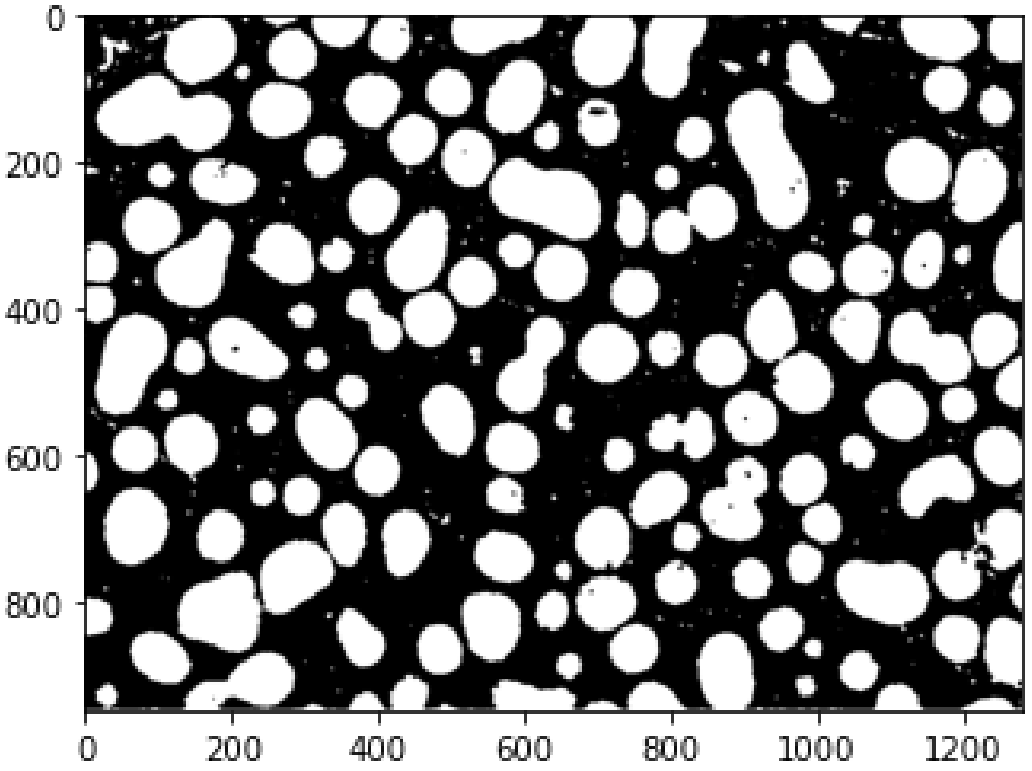
```
threshold2 = filters.threshold_otsu(median_filtered)
plt.imshow(phase_separation[:300, :300], cmap='gray')
plt.contour(median_filtered[:300, :300], [threshold2])
threshold2
```

134



Binary Image from Thresholding

```
1 binary_image = median_filtered < filters.threshold_otsu(median_filtered)
2 plt.imshow(binary_image, cmap='gray')
```



Small objects and small holes also segmented! ➔ need removed

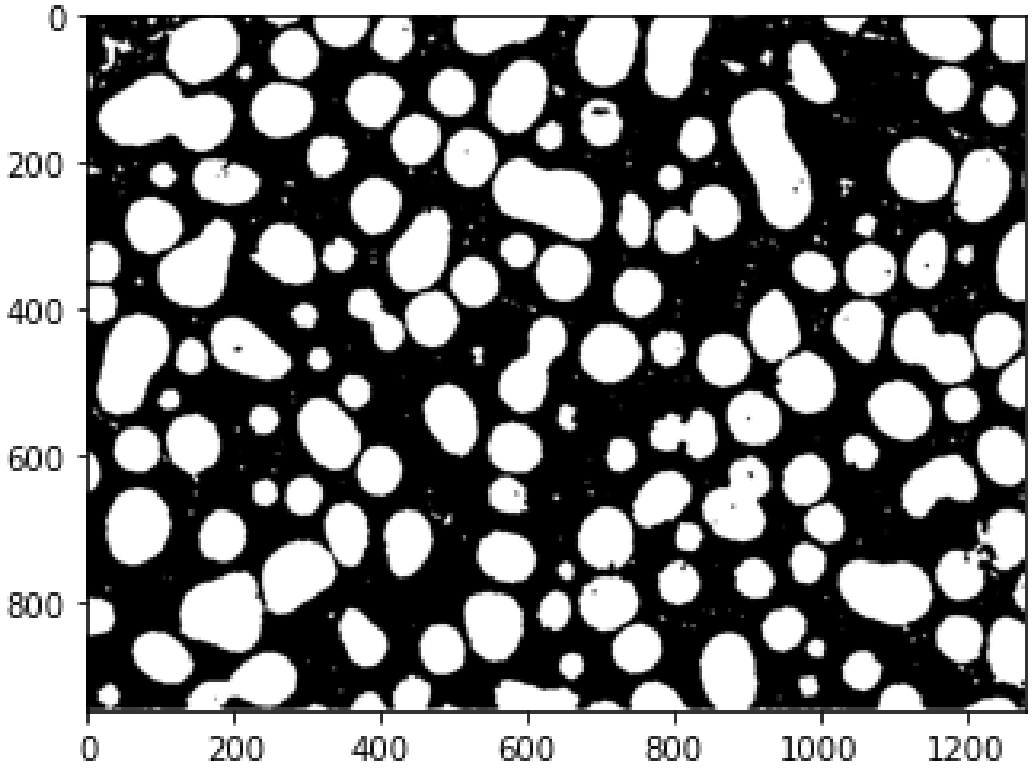
Image Cleaning: Morphological Filtering

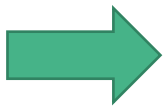
- If we use the denoising + thresholding approach, the result of the thresholding is not completely what we want: small objects are detected, and small holes exist in the objects. Such defects of the segmentation can be amended, using the knowledge that no small holes should exist, and that blobs have a minimal size.
- Morphological Filtering: https://scikit-image.org/docs/stable/auto_examples/applications/plot_morphology.html

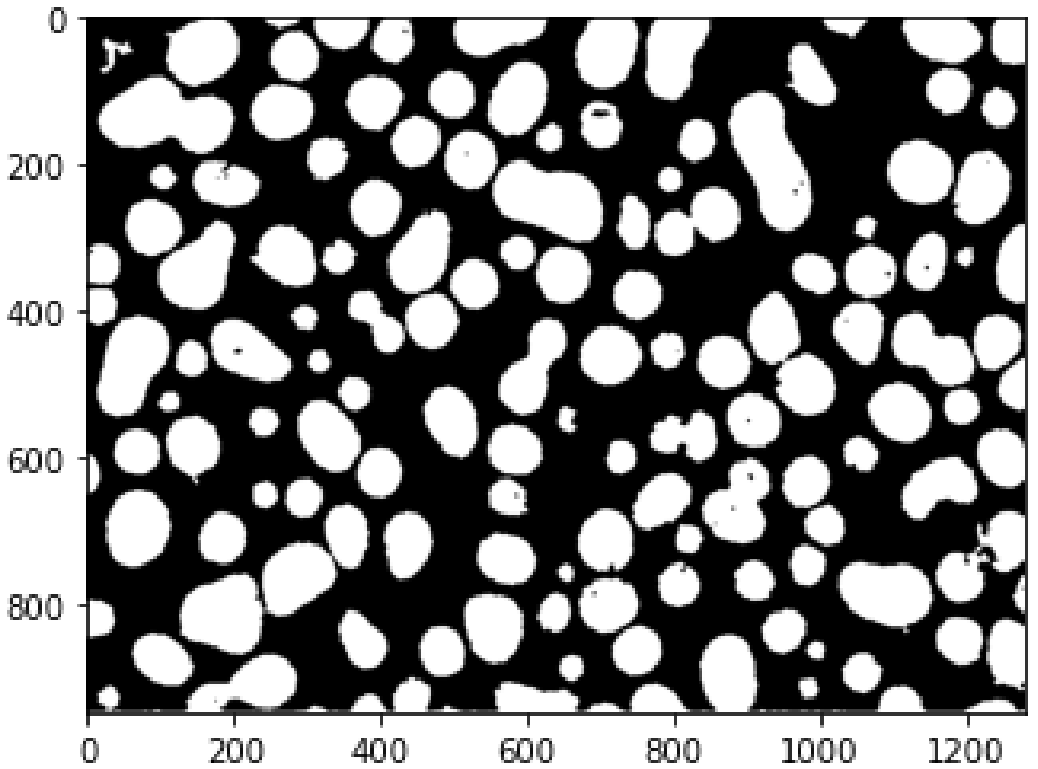
Morphological Filtering (1)

- Remove small objects

```
from skimage import morphology

#Remove small objects
only_large_blobs = morphology.remove_small_objects(binary_image,
                                                   min_size=300)
plt.imshow(only_large_blobs, cmap='gray')
```



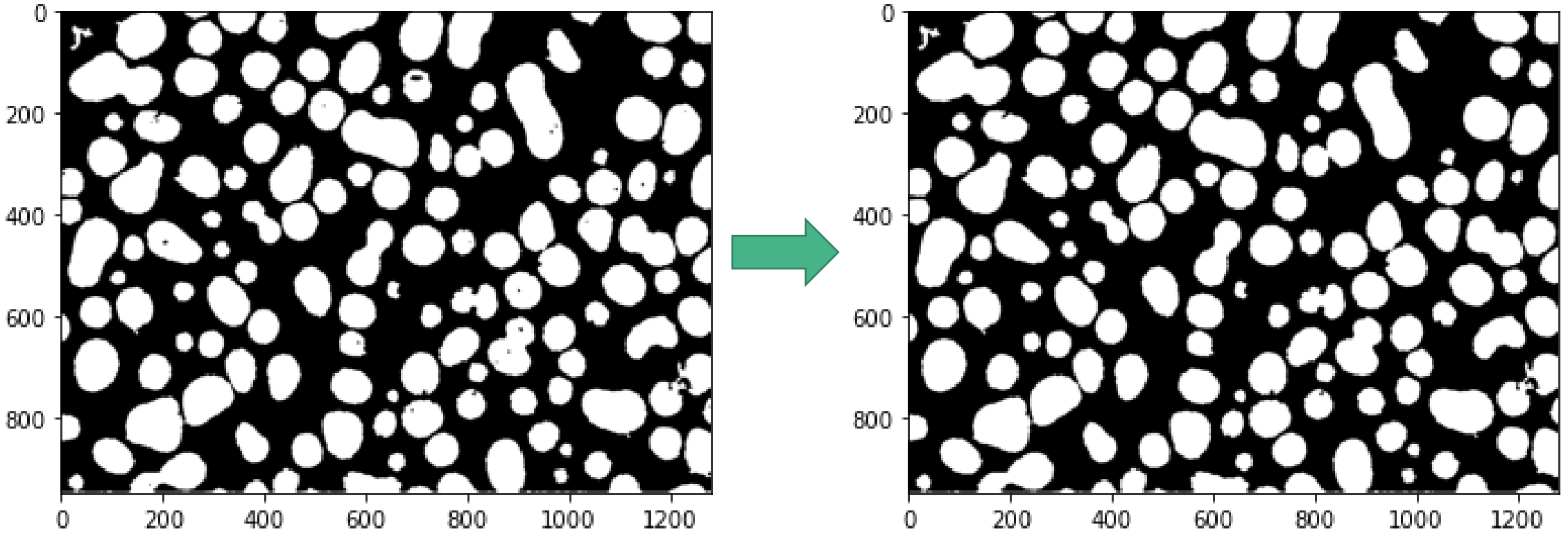




Morphological Filtering (2)

- Fill small holes

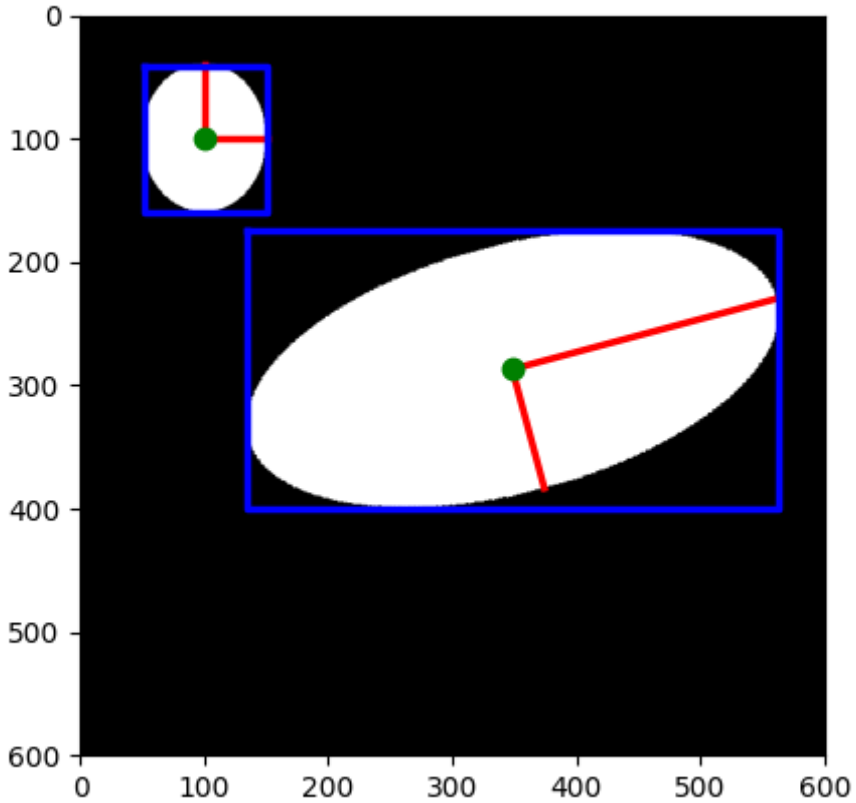
```
#Fill small holes
only_large = np.logical_not(morphology.remove_small_objects(
                                        np.logical_not(only_large_blobs),
                                                       min_size=300))
plt.imshow(only_large, cmap='gray')
```



Measuring region properties

- The segmentation of foreground (objects) and background results in a binary image. In order to measure the properties of the different blobs, one must first attribute a different label to each blob (identified as a connected component of the foreground phase).
- For scikit-image, the utility function measure.regionprops can be used to compute several properties of the labeled regions.

Example



	centroid-0	centroid-1	orientation	major_axis_length	minor_axis_length
0	100.000000	100.000000	0.000000	119.807049	99.823995
1	286.914167	348.412995	-1.308966	440.015503	199.918850