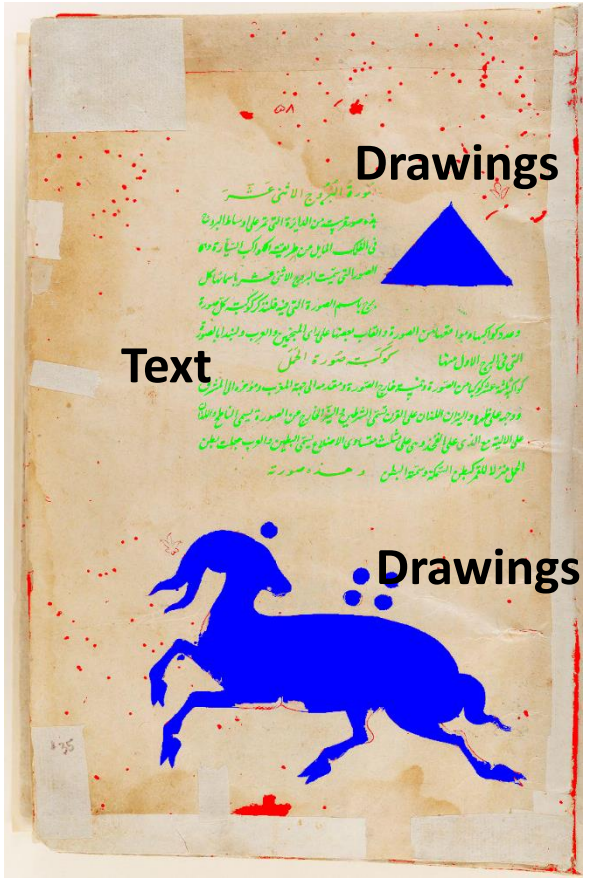
Segmentation

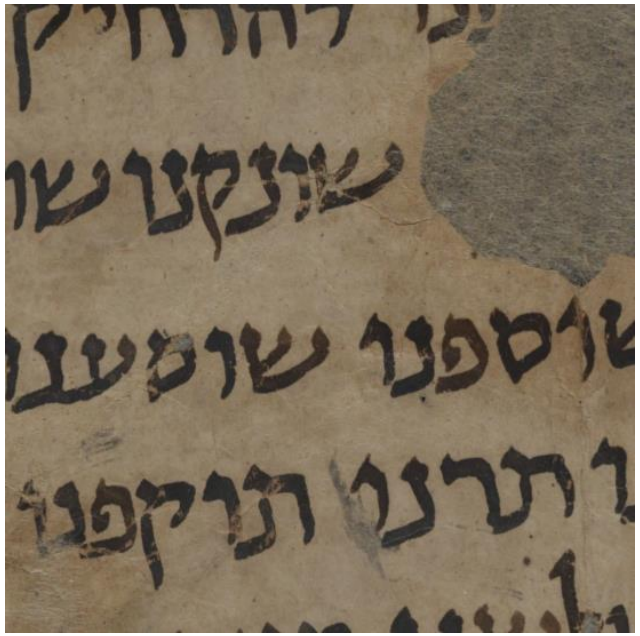
Segmentation

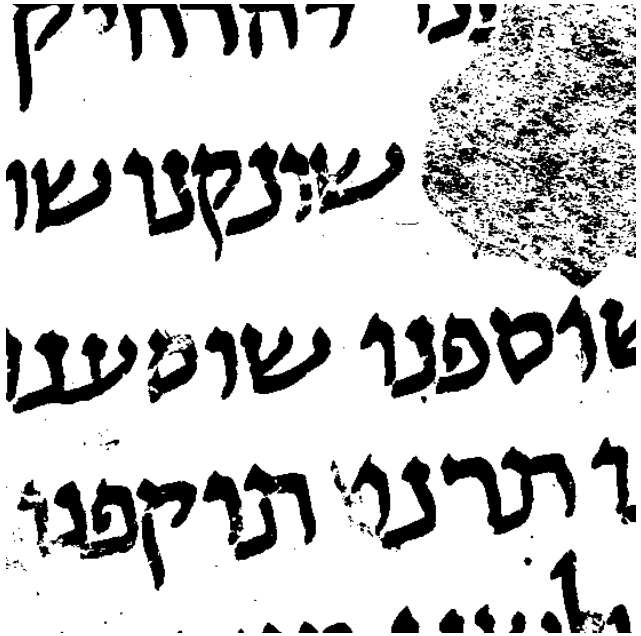
- Segmentation is the process of separating an image into multiple logical regions.
- There are many different methods of segmentation
- In this course we will study thresholding/binarization, which is a simplest kind of segmentation



Thresholding / Binarization

- Thresholding is converting the greyscale image into a binary image, where the pixels are either 0 or 255.





Simple thresholding

Pick the threshold value T

if $pv \geq T$ *threshold then*

segpv = 255

else

segpv = 0

where *pv* is the pixel value in the input image, and *segpv* is the pixel value in the segmented image.

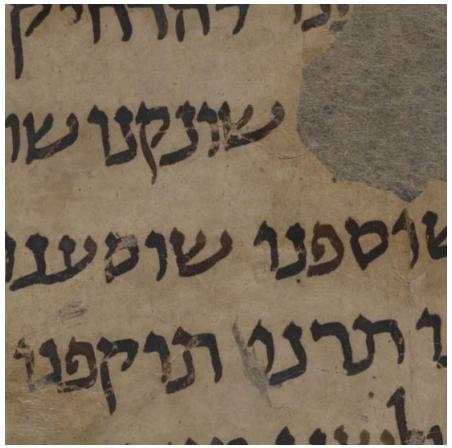
Simple thresholding – OpenCV implementation

retval, dst=cv2.threshold(src, thresh, maxval, type[, dst])

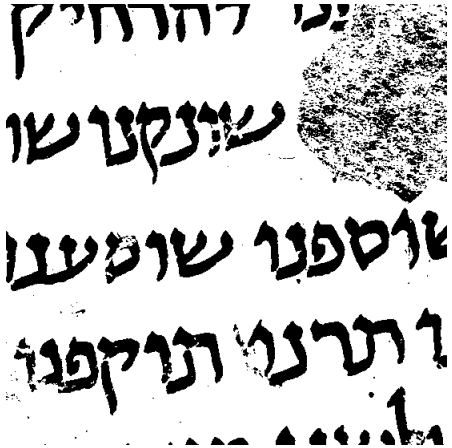
src	input array (multiple-channel, 8-bit or 32-bit floating point).
dst	output array of the same size and type and the same number of channels as src.
thresh	threshold value.
maxval	maximum value to use with the **THRESH_BINARY** and **THRESH_BINARY_INV** thresholding types.
type	thresholding type (see **ThresholdTypes**).

Simple thresholding – OpenCV implementation

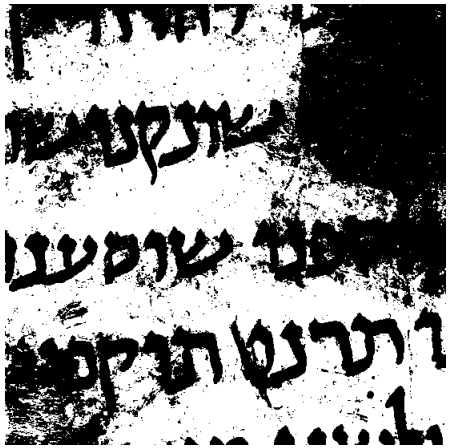
- img = cv2.imread(path)
  imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  (T, thresh) = cv2.threshold(imgGray, 95, 255, cv2.THRESH_BINARY)



Original



Thresh = 95



Thresh =120

Simple thresholding – OpenCV implementation

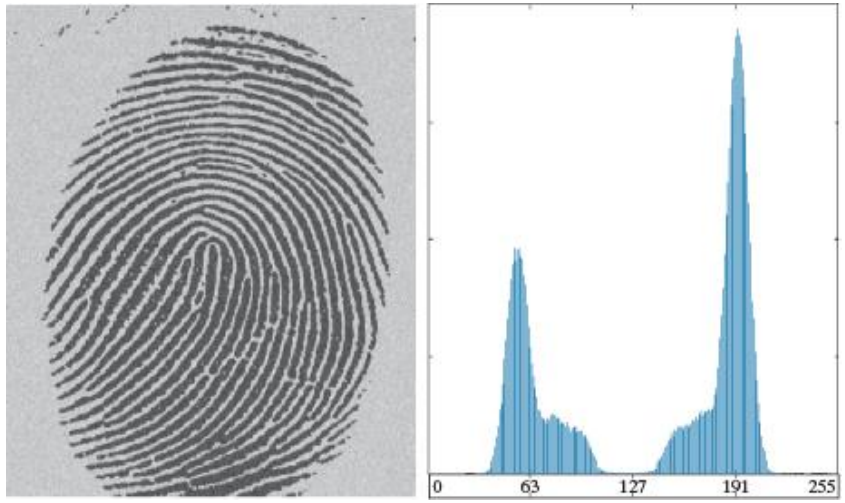
- In most cases we want the segmented objects to appear as *white* on a *black* background.
  - use cv2.THRESH_BINARY_INV
  - (T, thresh) = cv2.threshold(imgGray, 95, 255, cv2.THRESH_BINARY_INV)

Simple thresholding – the problem

- The problem – we must manually specify the threshold value.
  - tedious
  - problematic if we want the application to be *dynamic* and work under various lighting conditions.

Otsu’s binarization

- Assumes that we have two classes of pixels: background and foreground
- Assumes that the intensity distribution of objects and background pixels are sufficiently distinct, e.g. assumes that the pixel intensities of the input image is bi-modal



- The main idea:
  - Maximize the between class variance (variance between the foreground and background pixel values)

Otsu's binarization

- Let p_i be the probability of each intensity value
- For a threshold value t
  - Calculate the probability of each class $P_1(t) = \sum_{i=0}^{t} p_i\,, \quad P_2(t) = \sum_{i=t+1}^{L-1} p_i$
  - Calculate the expected value for each class

$$\mu_1 = \sum_{i=0}^{t} ip_i\,, \qquad \mu_2 = \sum_{i=t+1}^{L-1} ip_i$$

- The variance between two classes is given by

$$\sigma_{between}^2 = P_1 P_2 (\mu_1 - \mu_2)^2$$

Otsu's algorithm

- Let L be the number of intensities in the image
- For each t from 0 to $L-1$
  - Calculate $\mathrm{P}_1, \mathrm{P}_2, \mu_1, \mu_2$
  - Calculate $\sigma^2_{between}$
- The final threshold is the value of t for which $\sigma^2_{between}$ has its maximum value

Otsu’s binarization – OpenCV implementation

```
otsu_threshold, image_result = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY_INV |

cv2.THRESH_OTSU)

print("Obtained threshold: ", otsu_threshold)

cv2.imwrite(os.path.join(folder, "binary_otsu.png"), image_result)
```

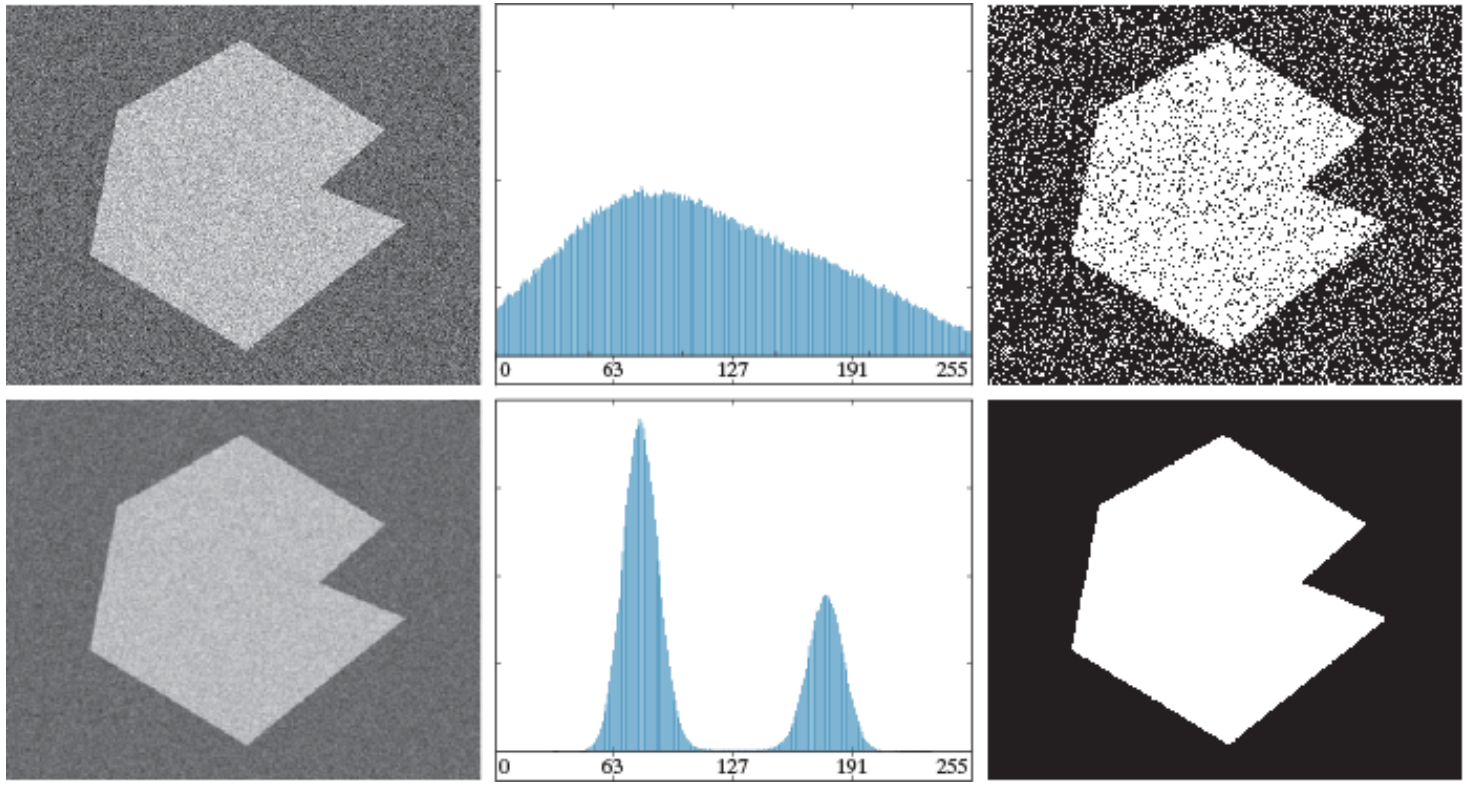
- There are also implementations of Otsu’s method in scikit and mahotas libraries

Otsu’s binarization – OpenCV implementation

Simple thresholding
Thresh = 95

Otsu

Using image smoothing to improve thresholding



a b c
d e f

FIGURE 10.37 (a) Noisy image from Fig. 10.33(c) and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging kernel and (e) its histogram. (f) Result of thresholding using Otsu's method.

Using image smoothing to improve thresholding

```
blurred = cv2.GaussianBlur(imgGray, (7, 7), 0)
threshold, res = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)
```

Otsu

Smoothing + Otsu

Adaptive Thresholding

- Otsu's method is a global thresholding
  - A global threshold might not provide accurate segmentation
- Adaptive thresholding can help
  - The image is first divided into many sub-images
  - The threshold value for each sub-image is computed and is used to segment the sub-image
  - Most common method to compute threshold for each sub-image is to use mean or median or Gaussian
    - Mean ➔ the mean of the sub-image is used to calculate a threshold
    - Gaussian ➔ pixel values farther away from the center of the sub-image contribute less to the overall calculation of T

Adaptive Thresholding-OpenCV

- **dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)**

- **src** – Source 8-bit single-channel image.
  dst – Destination image of the same size and the same type as src .
  maxValue – Non-zero value assigned to the pixels for which the condition is satisfied. See the details below.
  adaptiveMethod – Adaptive thresholding algorithm to use, ADAPTIVE_THRESH_MEAN_C or ADAPTIVE_THRESH_GAUSSIAN_C

  - **cv2.ADAPTIVE_THRESH_MEAN_C**: The threshold value is the mean of the neighbourhood area minus the constant **C**.
  - **cv2.ADAPTIVE_THRESH_GAUSSIAN_C**: The threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant **C**.

  thresholdType – Thresholding type that must be either THRESH_BINARY or THRESH_BINARY_INV
  blockSize – Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.
  C – Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may
  be zero or negative as well.

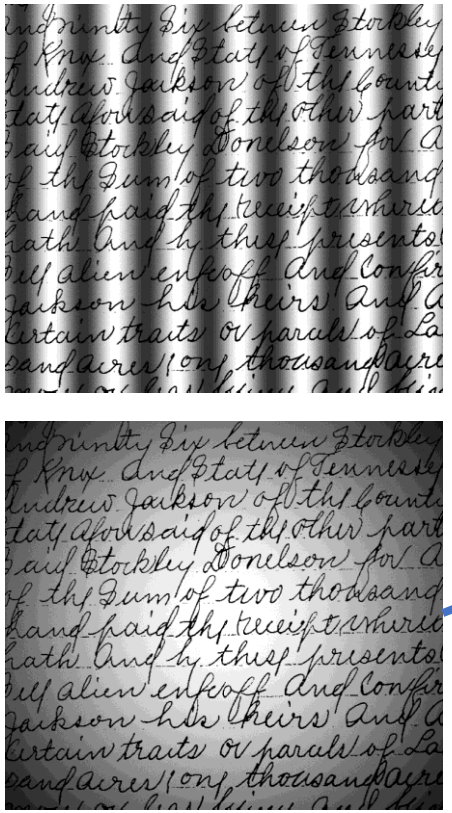
Adaptive Thresholding-OpenCV

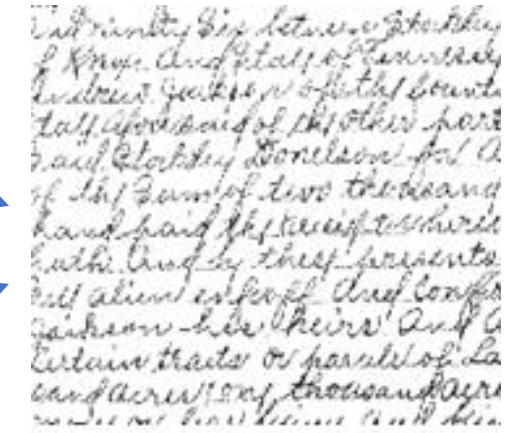
Summary

- In this lesson we've learned about
  - Simple thresholding
  - Otsu's thresholding
  - Adaptive thresholding

Practice

1. Download the following images and try to reproduce the following results





Practice

2. Segment the polymer cells from the following image