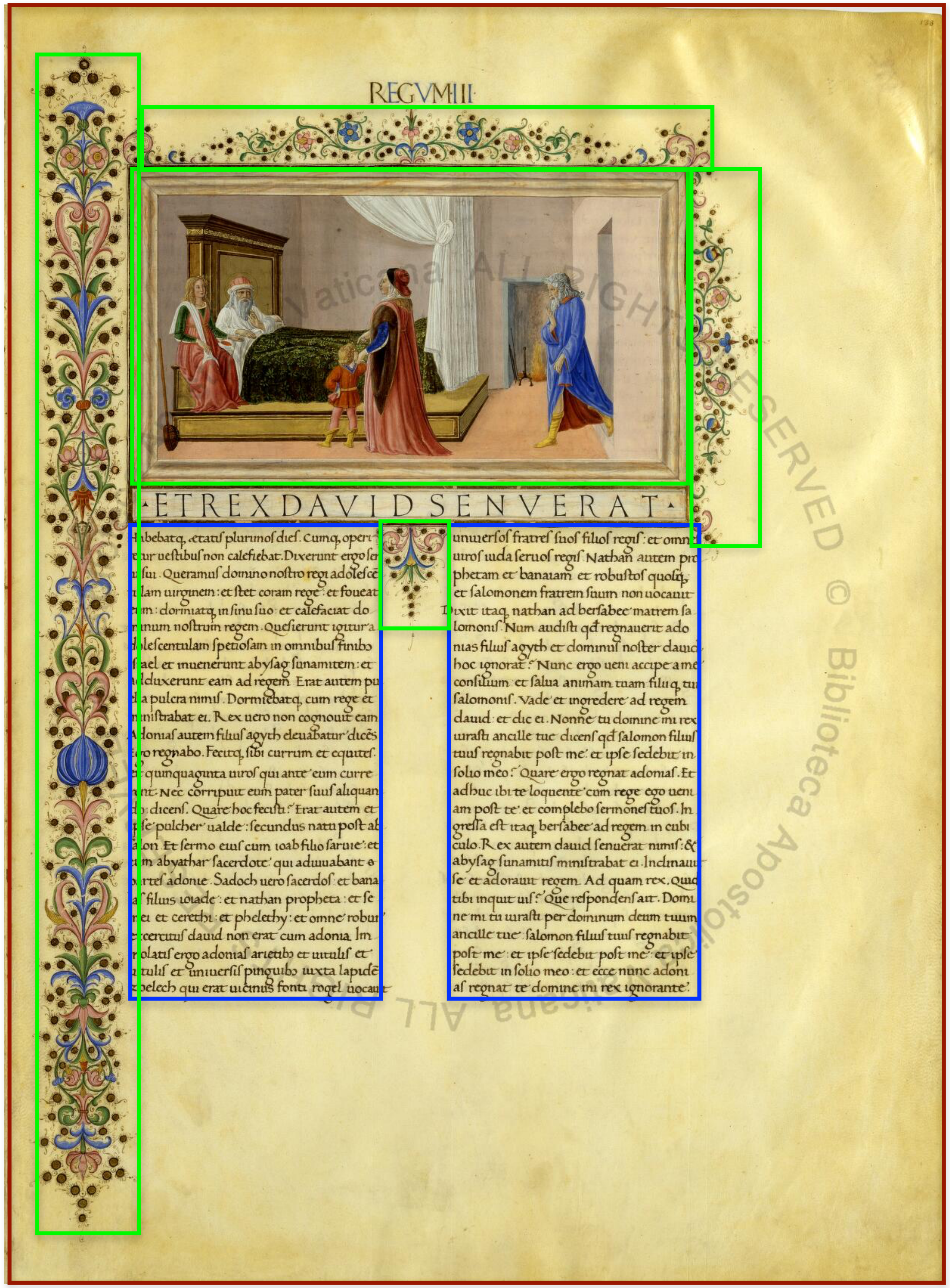
Image Segmentation



Giuliano Giuffrida
Coordination of IT Services, Vatican Apostolic Library

g.giuffrida@vatlib.it

Image Segmentation

Image segmentation is the operation of partitioning an image into a collection of sets of pixels.

1. into regions, which usually cover the entire nimage

2. into linear structures, such as
   - line segments
   - curve segments

3. into 2D shapes, such as
   - circles
   - ellipses
   - ribbons (long, symmetric regions)

a. $\bigcup_{i=1}^{n} R_i = R$.

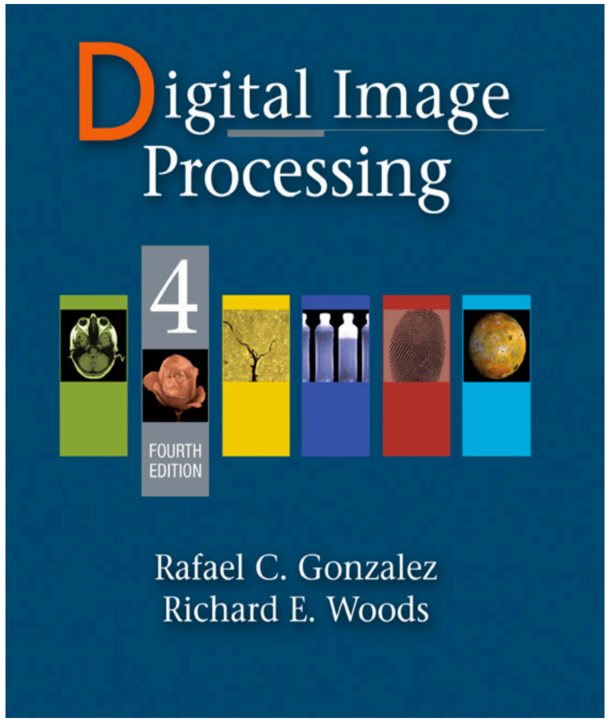
b. R_i is a connected set, for $i = 0, 1, 2, \ldots, n$.

c. $R_i \cap R_j = \emptyset$ for all *i* and *j*, $i \neq j$.

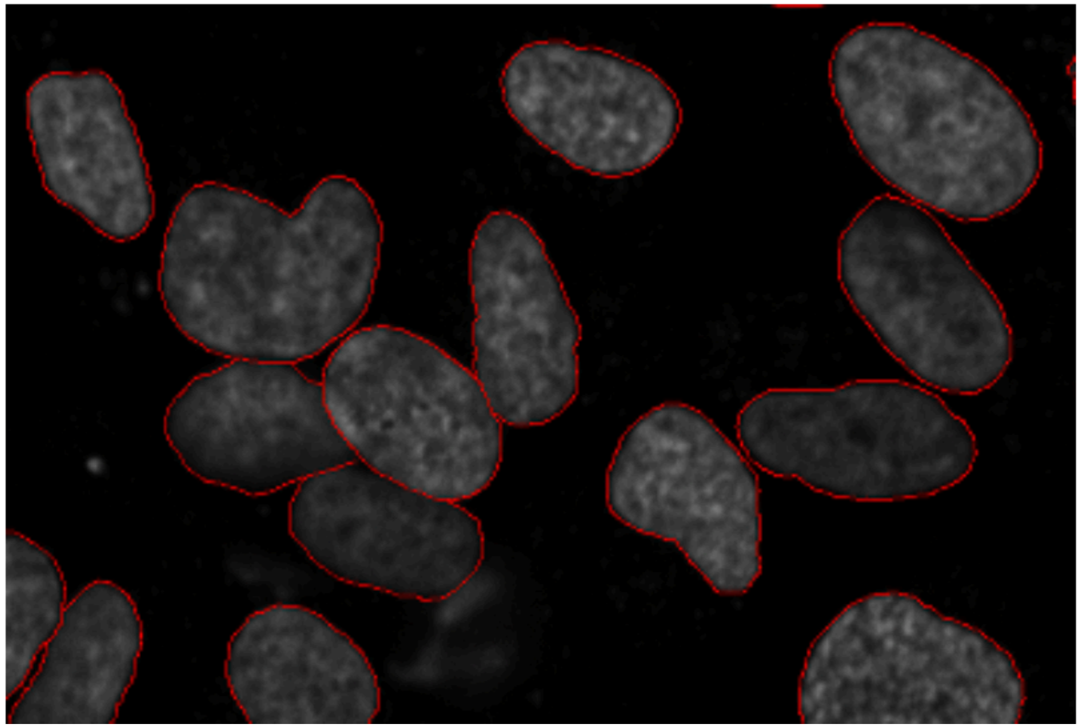
d. $Q(R_i) = \text{TRUE}$ for $i = 0, 1, 2, \ldots, n$.

e. $Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions R_i and R_j

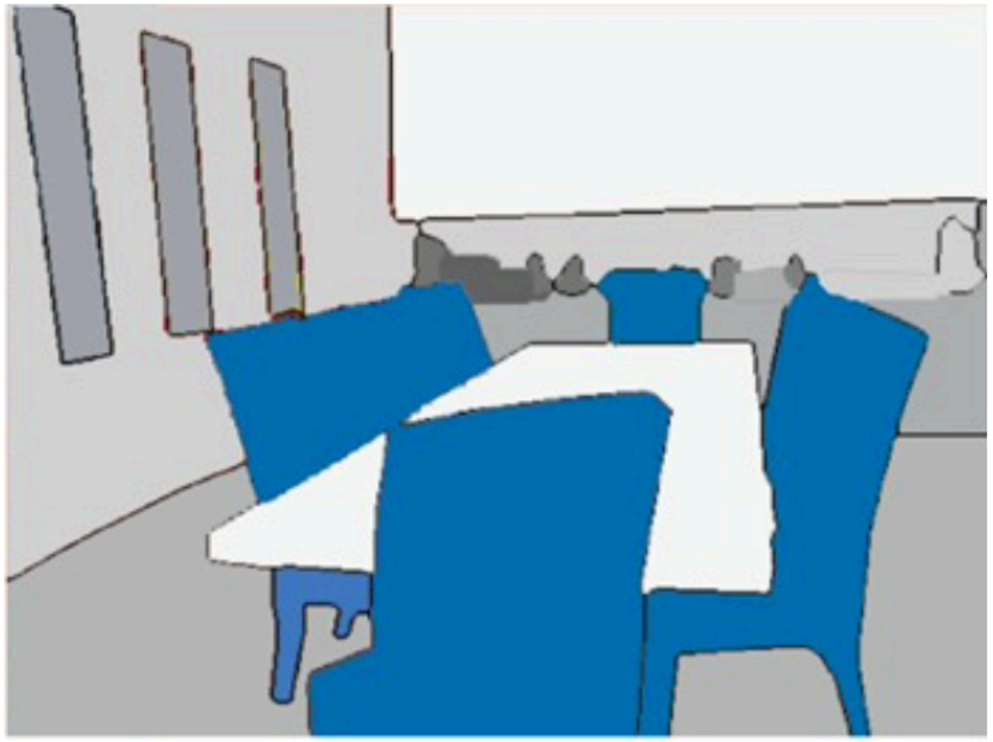
Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region for example if all pixels in have the same intensity.



Separation foreground-background



Semantic segmentation



Instance segmentation

https://gmanco.github.io/courses/computervision/

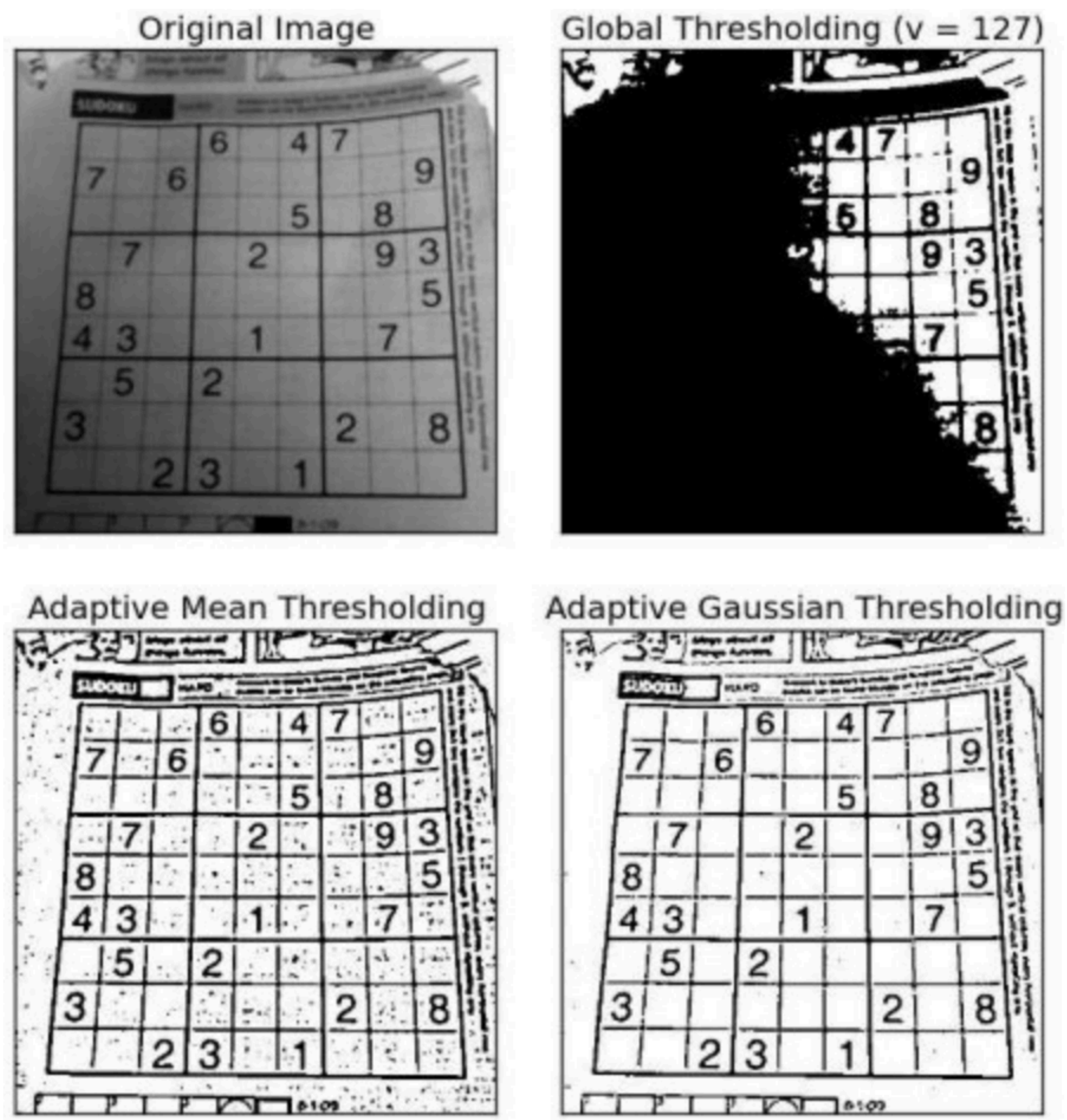
Instance segmentation is an enhanced type of object detection that generates a segmentation map for each detected instance of an object

Semantic segmentation is the task of clustering parts of an image together which belong to the same object class

Thresholding and Binarization

the threshold value $T(x,y)$ is a mean of the $\texttt{blockSize} \times \texttt{blockSize}$ neighborhood of (x,y) minus C

Global and Local Thresholding



$$\theta(I,t) = \begin{cases} 1 & I > t \\ 0 & \end{cases}$$

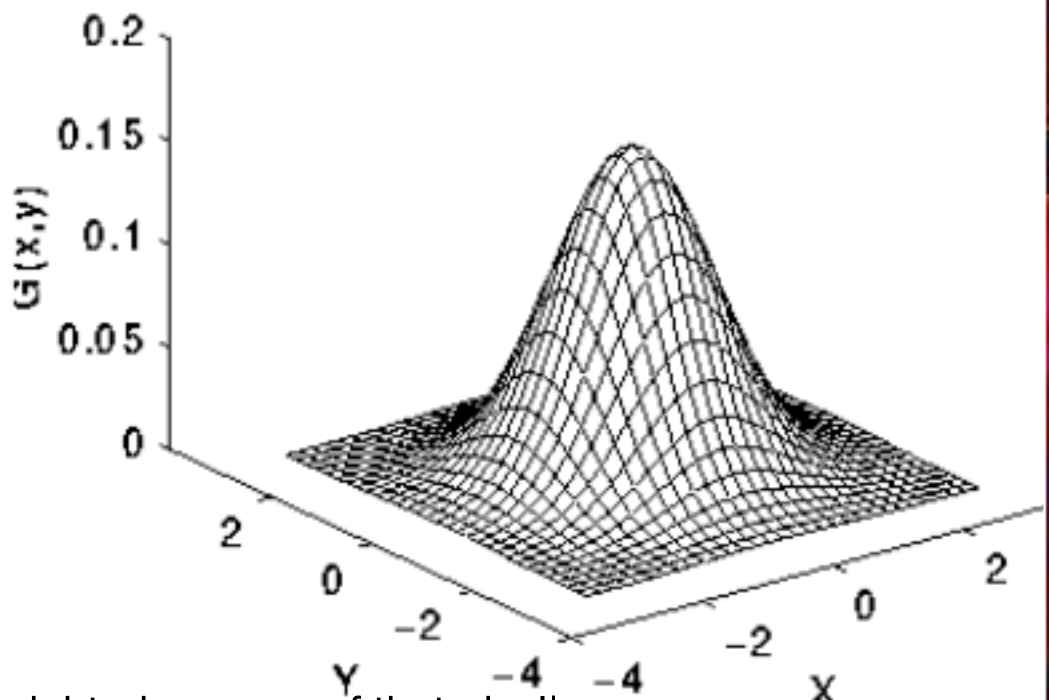
the threshold value $T(x,y)$ is a weighted sum (cross-correlation with a Gaussian window) of the $\texttt{blockSize} \times \texttt{blockSize}$ neighborhood of (x,y) minus C . The default sigma (standard deviation) is used for the specified blockSize

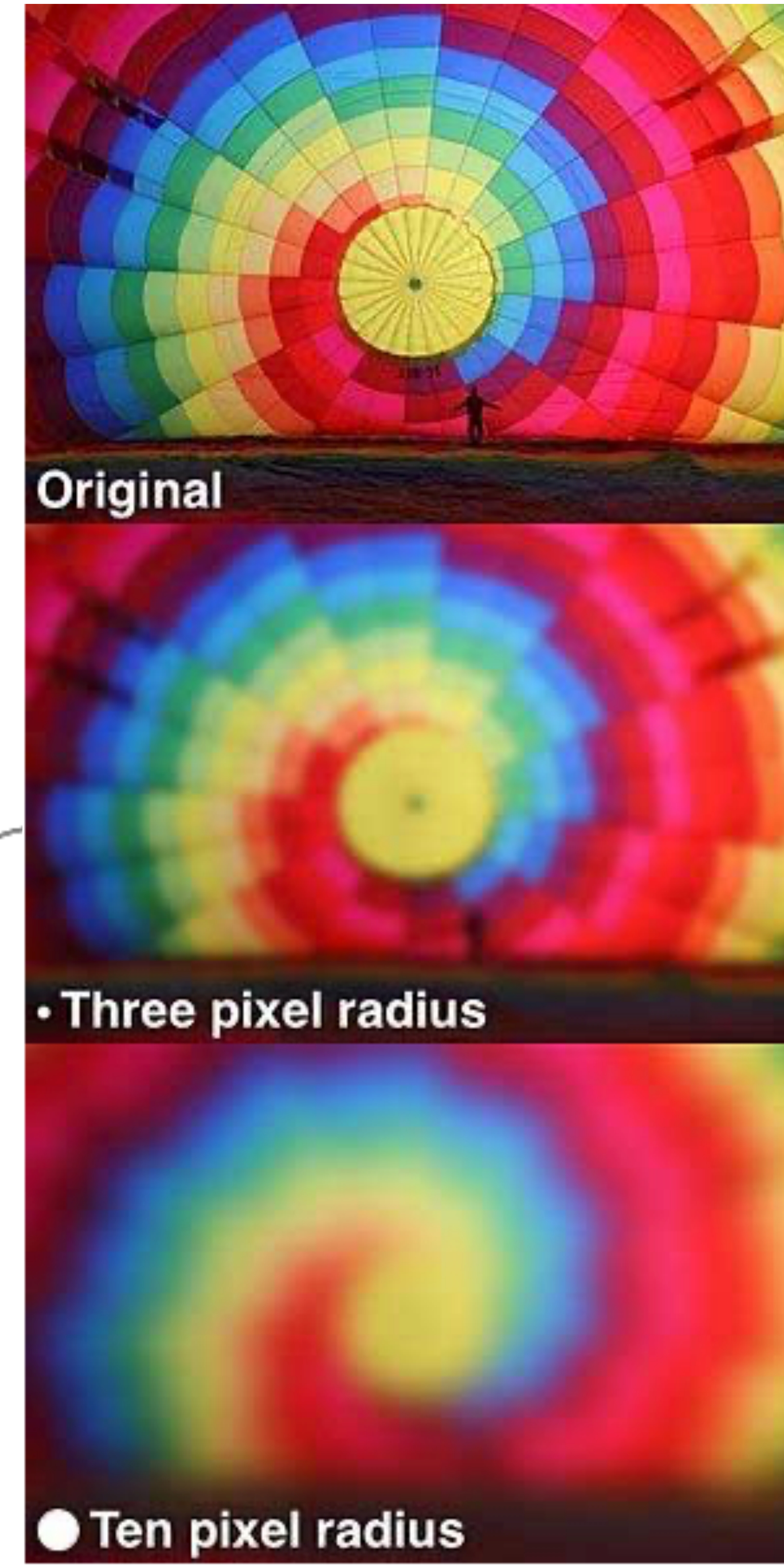
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Gaussian filter

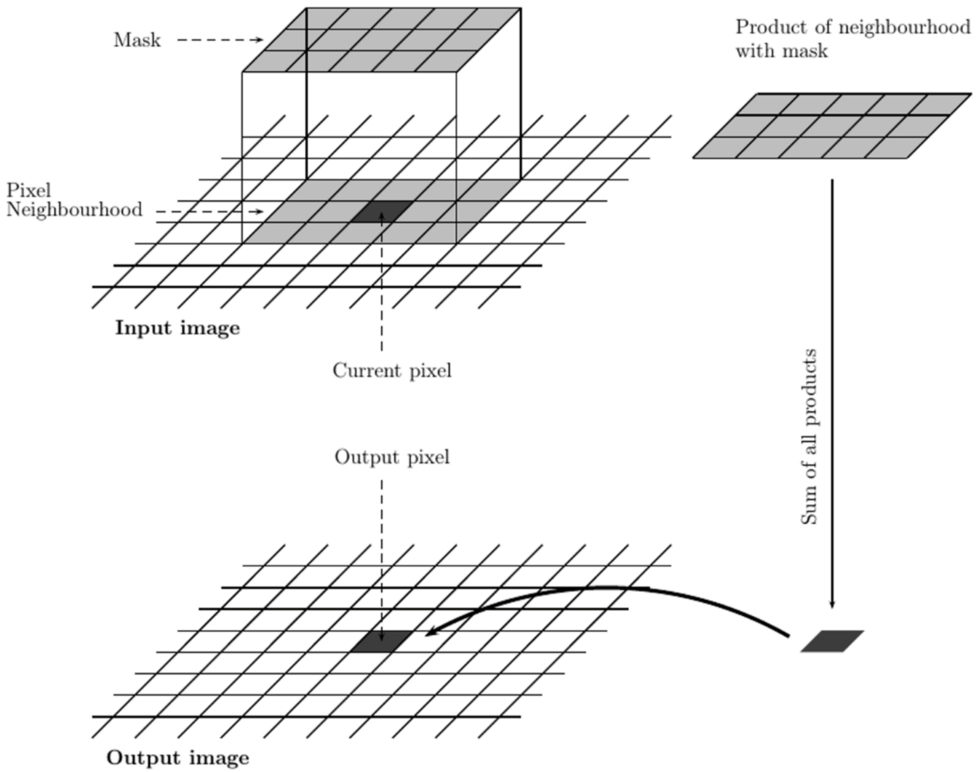
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$





- Each pixel's new value is set to a weighted average of that pixel's neighborhood.
- The original pixel's value receives the heaviest weight and neighboring pixels receive smaller weights as their distance to the original pixel increases.
- This results in a blur that preserves boundaries and edges better than other, more uniform blurring filters
- 2 parameters: sigma and kernel radius



$$\mathrm{h}[i,j] = \sum_{k=-a}^{a} \sum_{l=-b}^{b} f[k+a, l+b] \times I[i+k, j+l]$$

f = coefficient matrix, also known as kernel, filter, mask, …

1/16

1	2	1
2	4	2
1	2	1

1/273

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

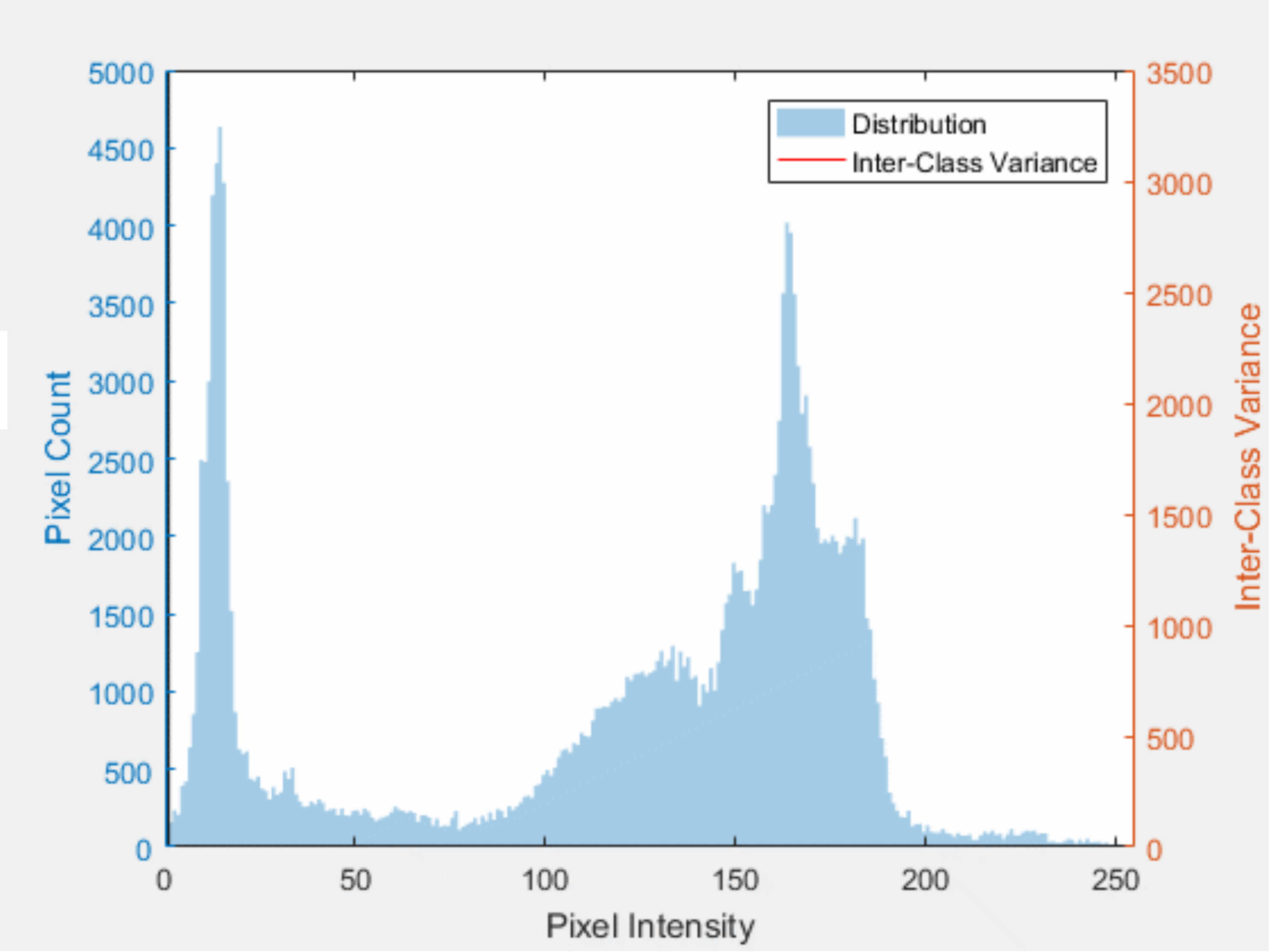
1/1003

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

Shipitko, Oleg & Grigoryev, Anton. (2018). Gaussian filtering for FPGA based image processing with High-Level Synthesis tools.

Thresholding and Binarization

Otsu Thresholding



$$\sigma^2_w(t) = \omega_0(t)\sigma^2_0(t) + \omega_1(t)\sigma^2_1(t)$$

$$\sigma^2_b(t) = \sigma^2 - \sigma^2_w(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

Soglia migliore = quella che minimizza $\sigma^2_w(t)$

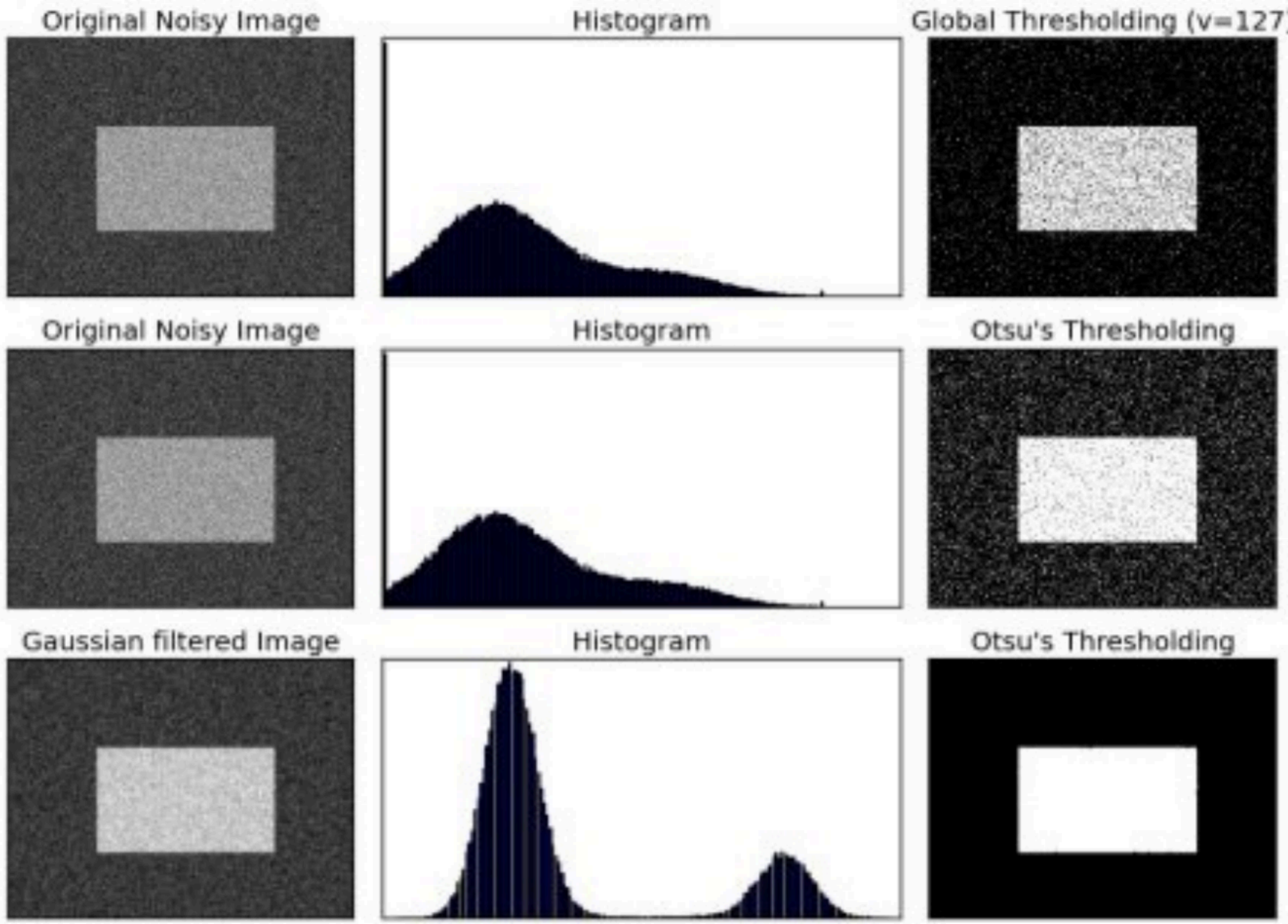
Equivalente a massimizzare $\sigma^2_b(t)$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

Sigma = varianza dei punti sopra soglia e sotto soglia

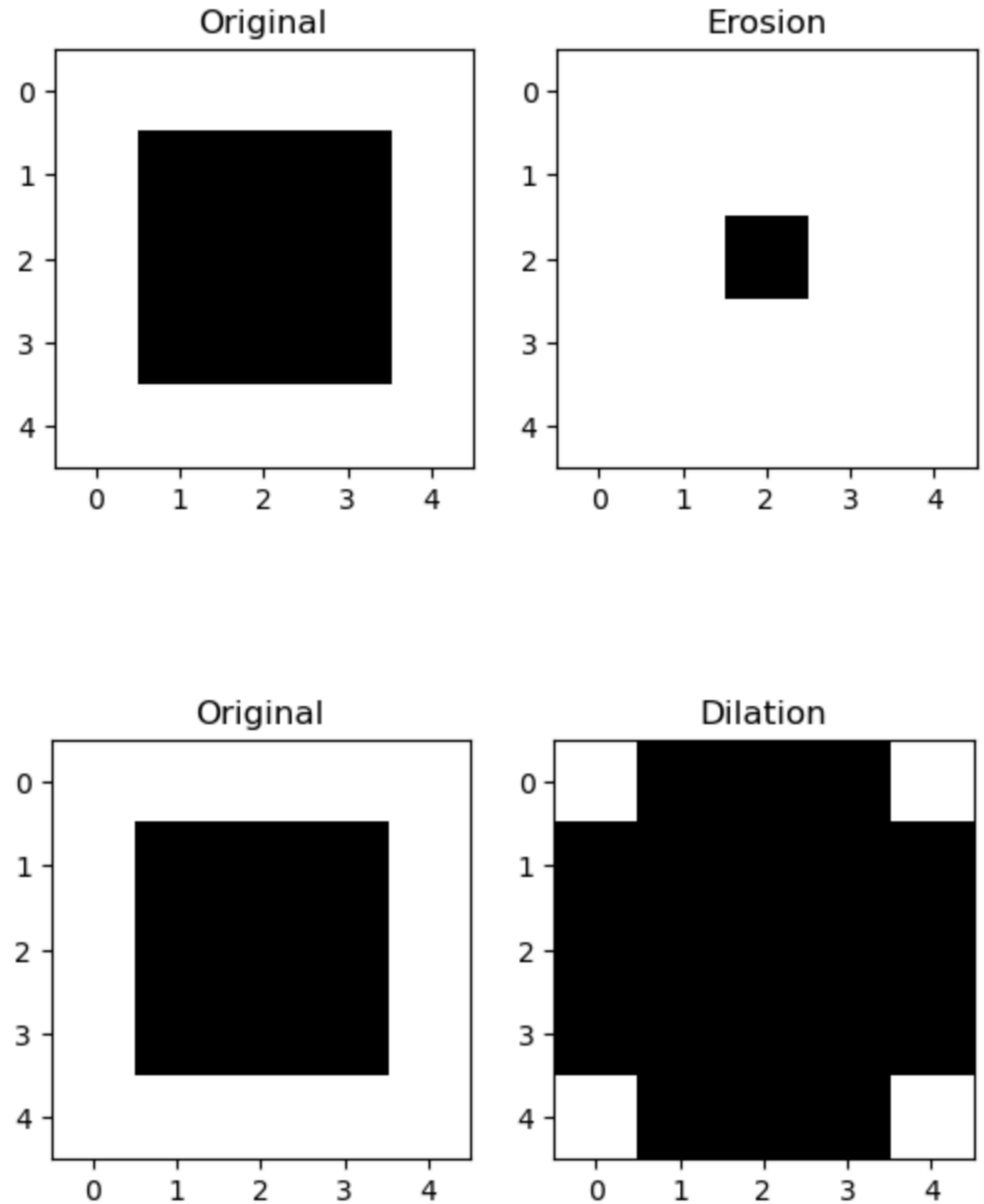
$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$ t=soglia

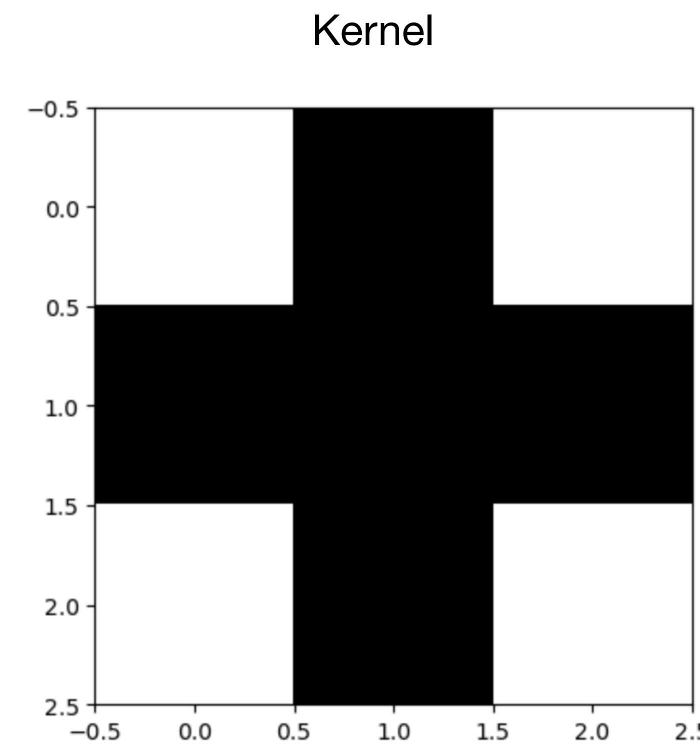
```python
import numpy as np

def compute_otsu_criteria(im, th):
    # create the thresholded image
    thresholded_im = np.zeros(im.shape)
    thresholded_im[im >= th] = 1

    # compute weights
    nb_pixels = im.size
    nb_pixels1 = np.count_nonzero(thresholded_im)
    weight1 = nb_pixels1 / nb_pixels
    weight0 = 1 - weight1

    # if one the classes is empty, eg all pixels are below or above the threshold, that threshold will not be considered
    # in the search for the best threshold
    if weight1 == 0 or weight0 == 0:
        return np.inf

    # find all pixels belonging to each class
    val_pixels1 = im[thresholded_im == 1]
    val_pixels0 = im[thresholded_im == 0]

    # compute variance of these classes
    var0 = np.var(val_pixels0) if len(val_pixels0) > 0 else 0
    var1 = np.var(val_pixels1) if len(val_pixels1) > 0 else 0

    return weight0 * var0 + weight1 * var1

im = # load your image as a numpy array.
# For testing purposes, one can use for example im = np.random.randint(0,255, size = (50,50))

# testing all thresholds from 0 to the maximum of the image
threshold_range = range(np.max(im)+1)
criterias = [compute_otsu_criteria(im, th) for th in threshold_range]

# best threshold is the one minimizing the Otsu criteria
best_threshold = threshold_range[np.argmin(criterias)]
```


Original Noisy Image
Histogram
Global Thresholding (v=127)
Original Noisy Image
Histogram
Otsu's Thresholding
Gaussian filtered Image
Histogram
Otsu's Thresholding

Demo dei vari thresholding su jupyter
th_demo.ipynb

Erosion and dilation





Erosion=minimum
Dilation = maximum

Opening and closing

Opening = Erosion+ Dilation

Closing = Dilation +Erosion

First segmentation

Thresholding + closing + binarization + contour

Contour = curve joining all the continuous points with same intensity

Edge detection

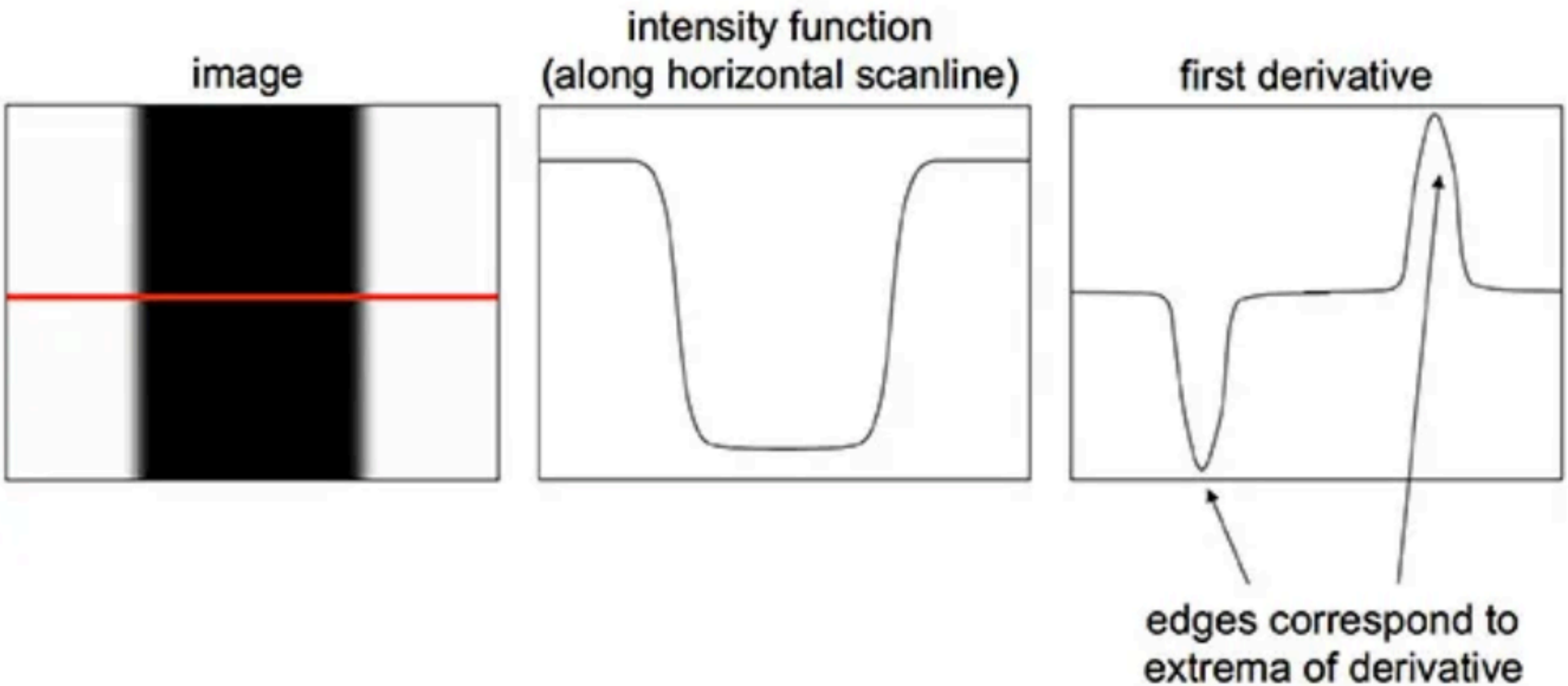
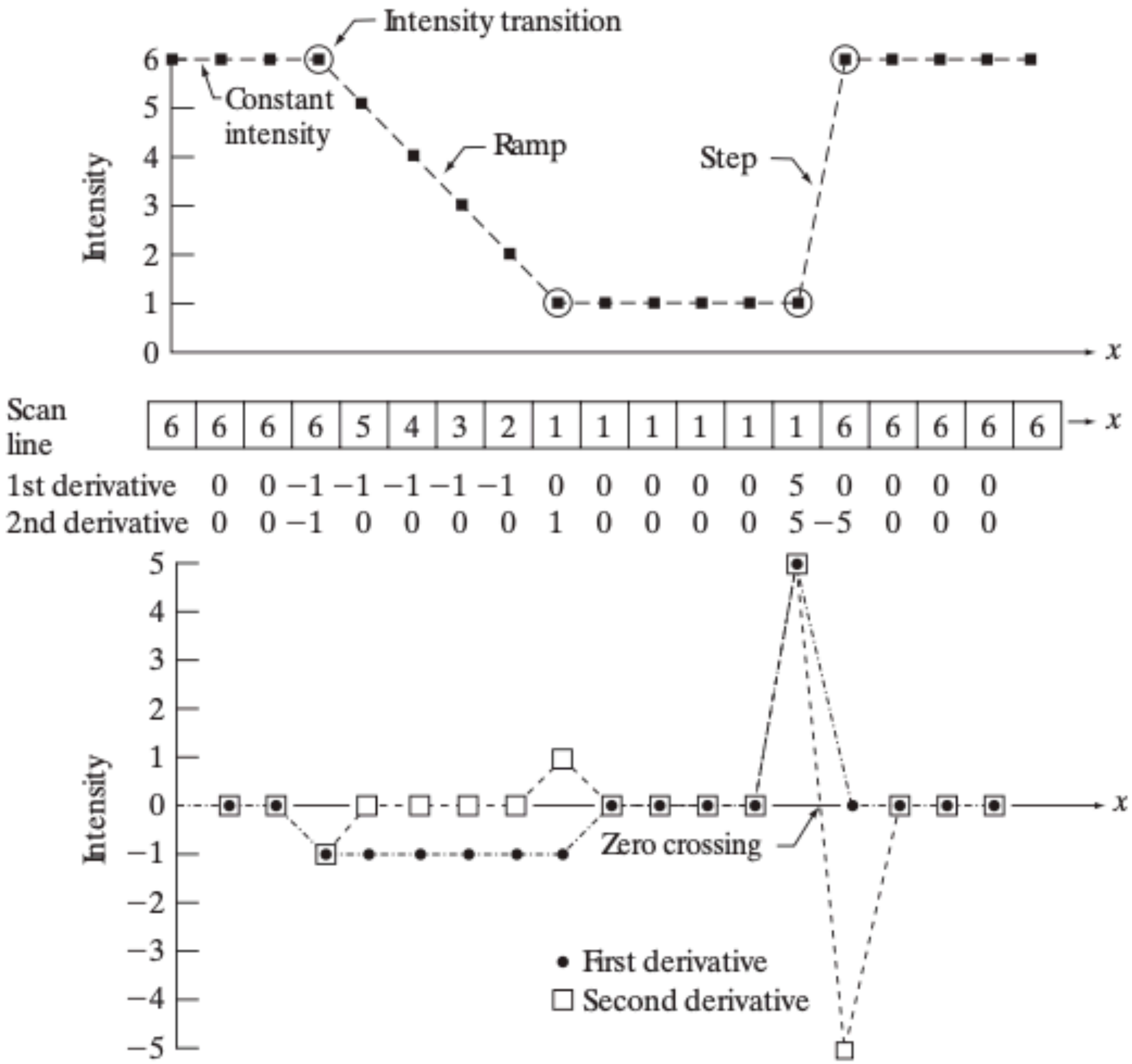


Image Credit: http://stanford.edu/



$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

Source: https://gmanco.github.io/courses/computervision/

Sobel filter

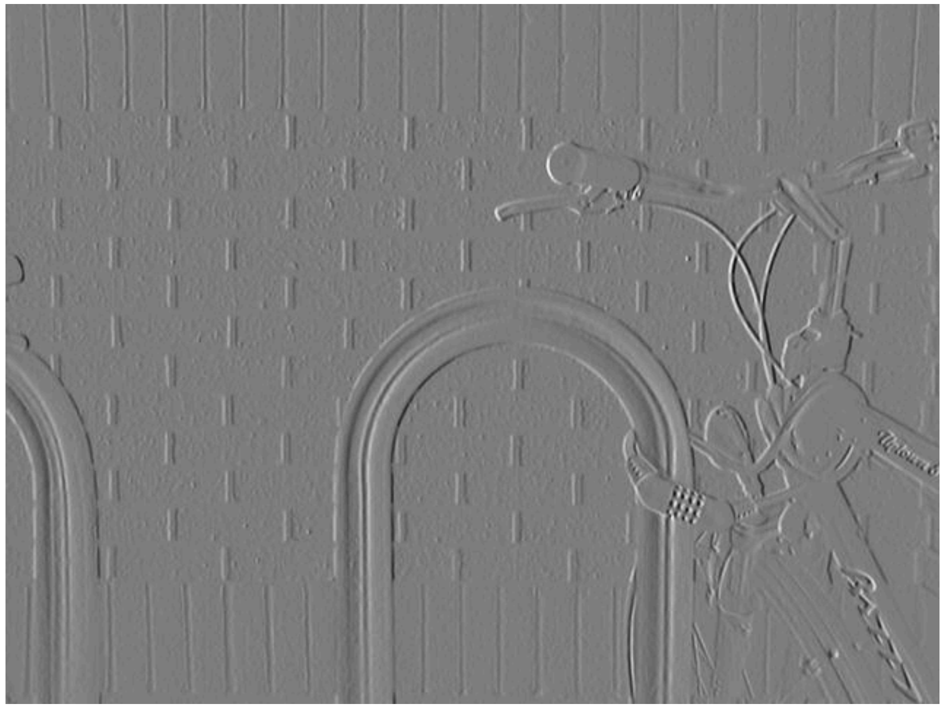
Orizzontale

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

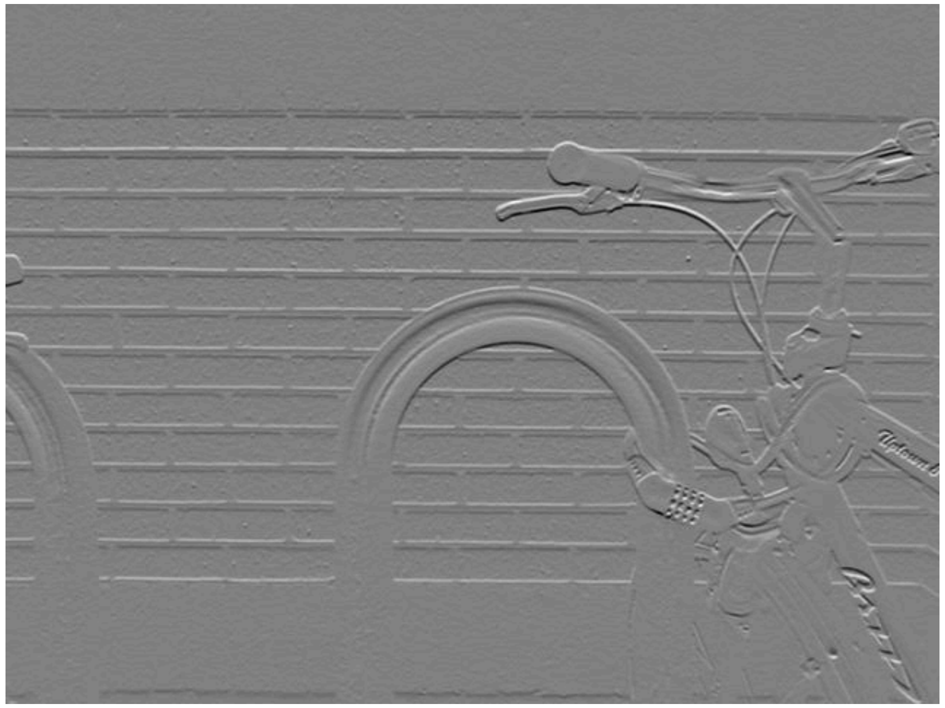
Verticale

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Orizzontale



Verticale



Source: wikipedia

Gradient-based filtering

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \qquad \boldsymbol{S}_y = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

2. Convolvi con l'immagine

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x * \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y * \boldsymbol{f}$$

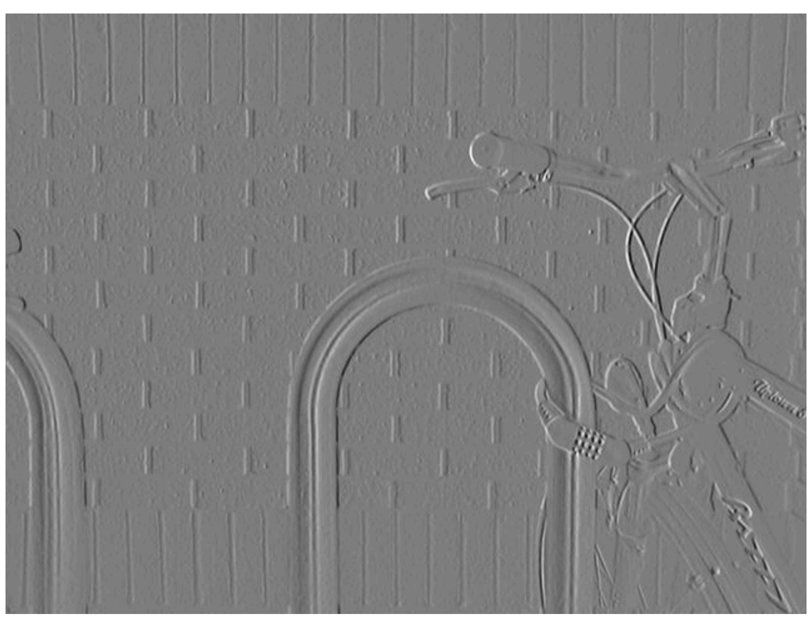
3. Calcola direzione e ampiezza del gradiente.

$$\nabla \boldsymbol{f} = \left[\frac{\partial \boldsymbol{f}}{\partial x}, \frac{\partial \boldsymbol{f}}{\partial y}\right] \qquad \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right) \qquad ||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

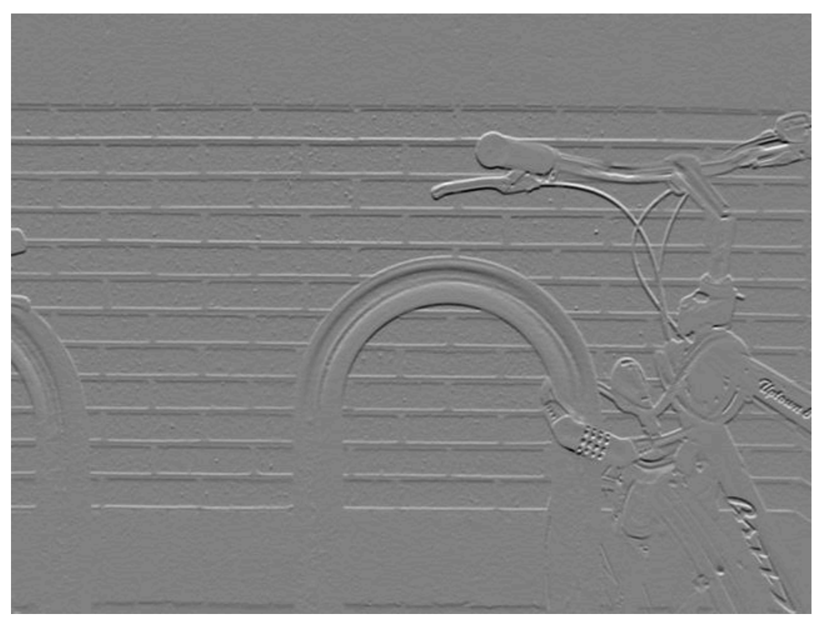
gradient direction amplitude

Source: https://gmanco.github.io/courses/computervision/

Orizzontale



Verticale



Gradient

Demo sobel

Edge detection - Canny

- Edges with gradient could be good, but typically some edges thick that others
- Ideally all edges should be thin and with the same intensity (255)

Edge detection - Canny

5 Steps:

1. Noise reduction
2. Gradient calculation
3. Non-maximum suppression
4. Double threshold
5. Edge Tracking by Hysteresis

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

1. Noise reduction

Gaussian blur

Original image (left) — Blurred image with a Gaussian filter (sigma=1.4 and kernel size of 5×5)

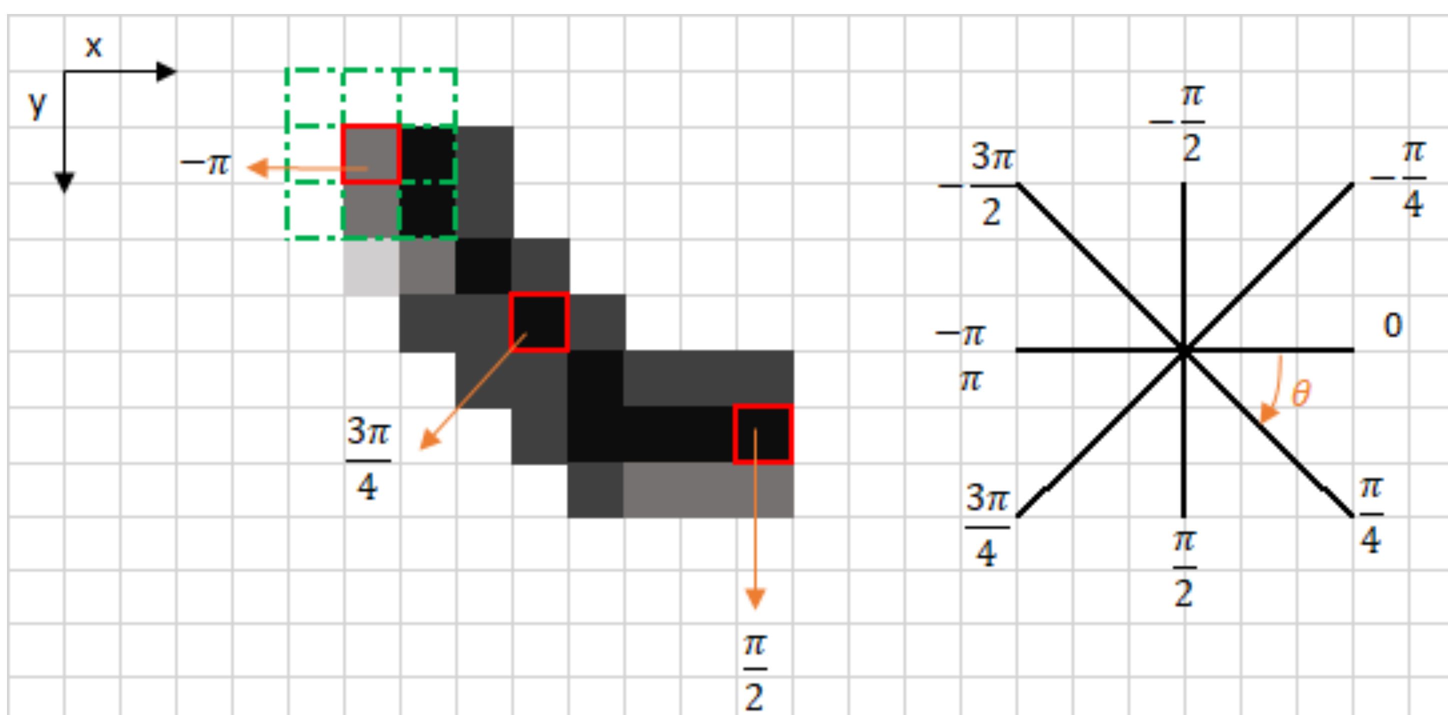
2. Gradient calculation

Blurred image (left) — Gradient intensity (right)

3. Non-maximum suppression

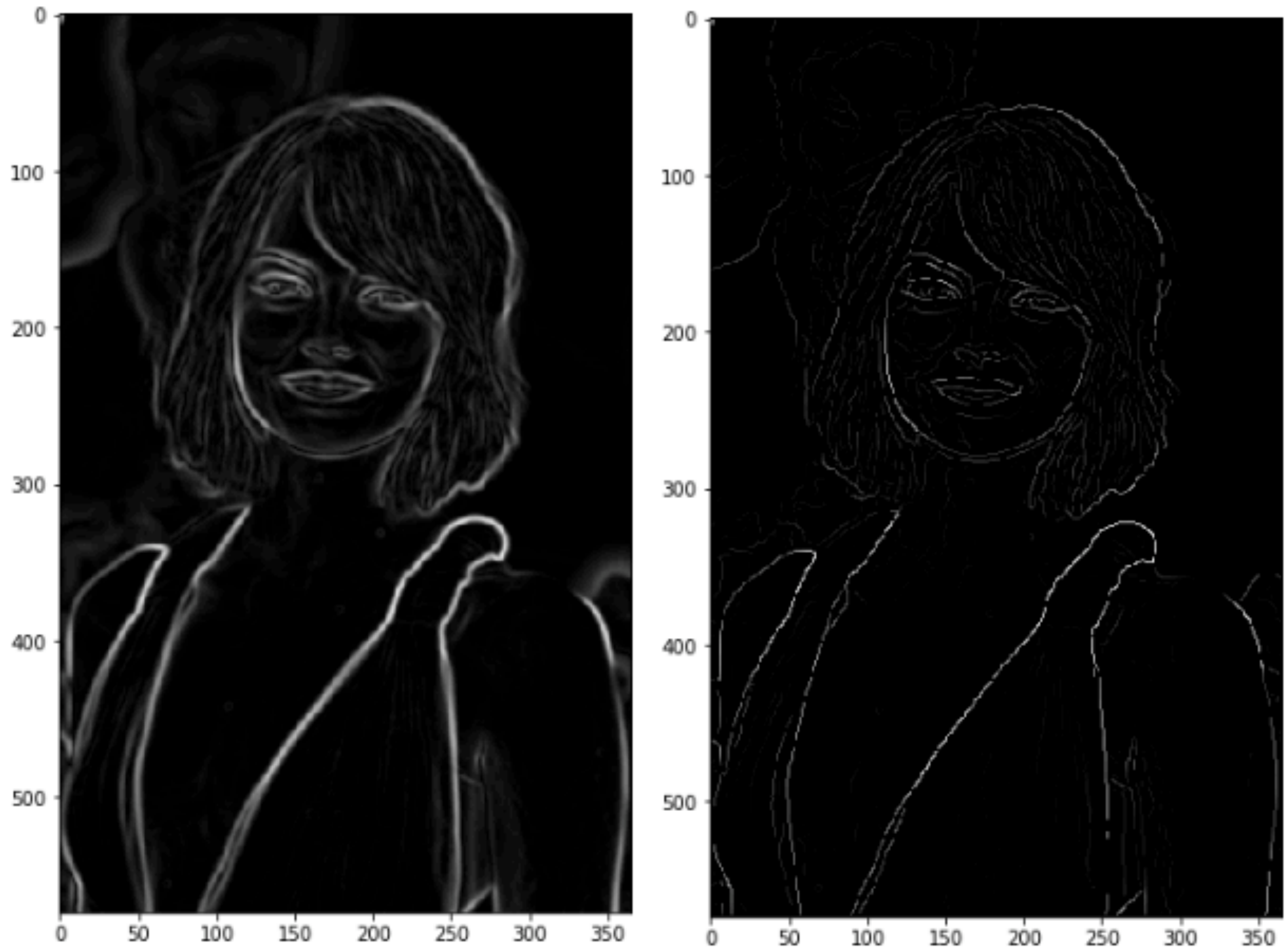
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

direction



Pixels with a first neighbor in theta direction with bigger values are set to zero

3. Non-maximum suppression



Result of the non-max suppression.

Now thinner but with different intensities

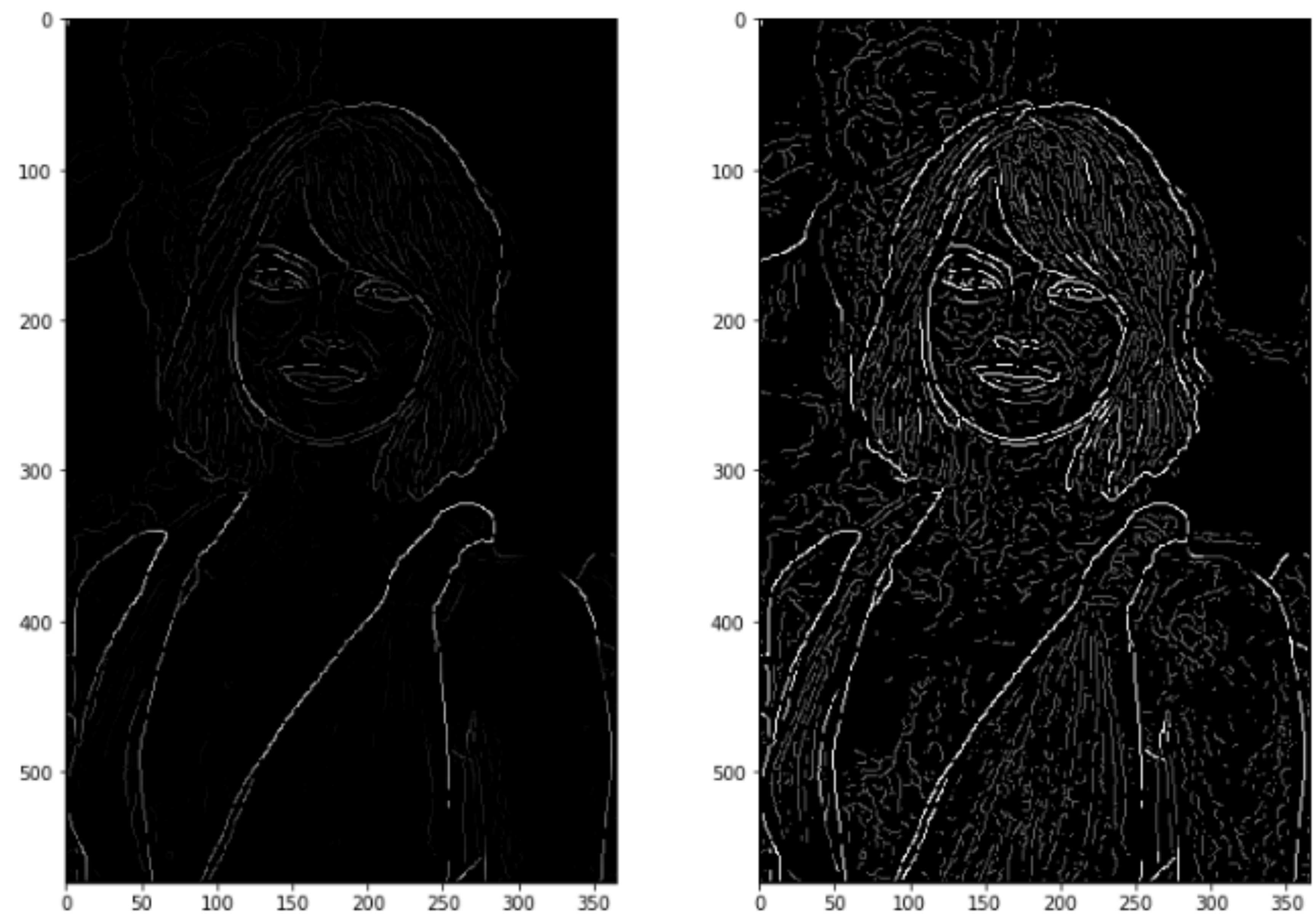
4. Double threshold

Two thresholds: high and low (es. 70,10)

Strong pixel: img(x,y)>=high —> img(x,y)=255

Weak pixel: img(x,y)<high and img(x,y)>=low —> img(x,y)=25

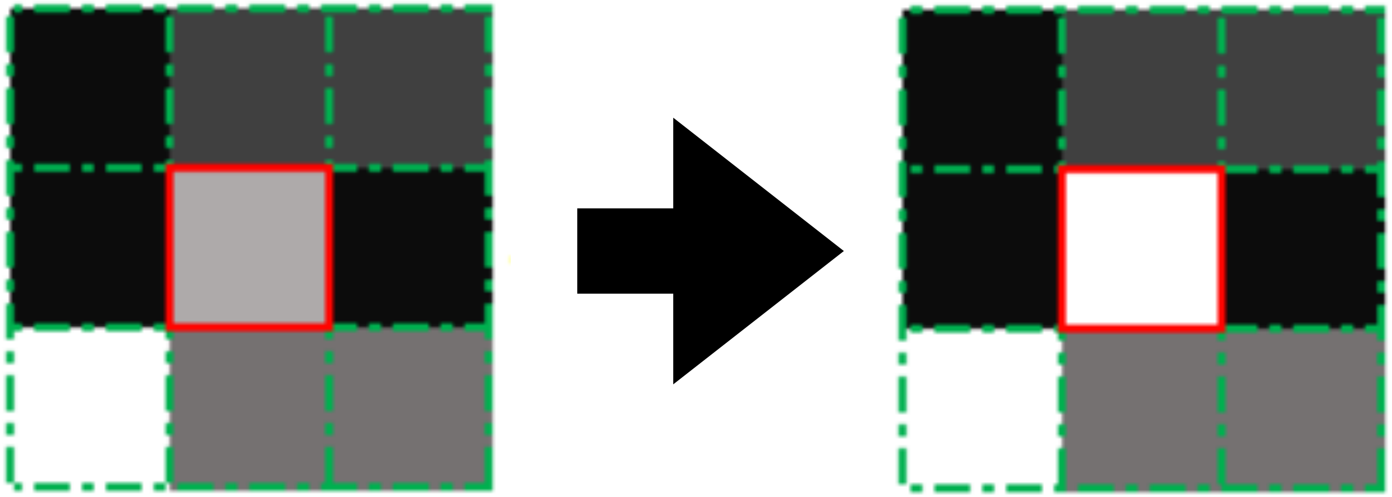
Zero pixel: img(x,y)<low —> img(x,y)=0



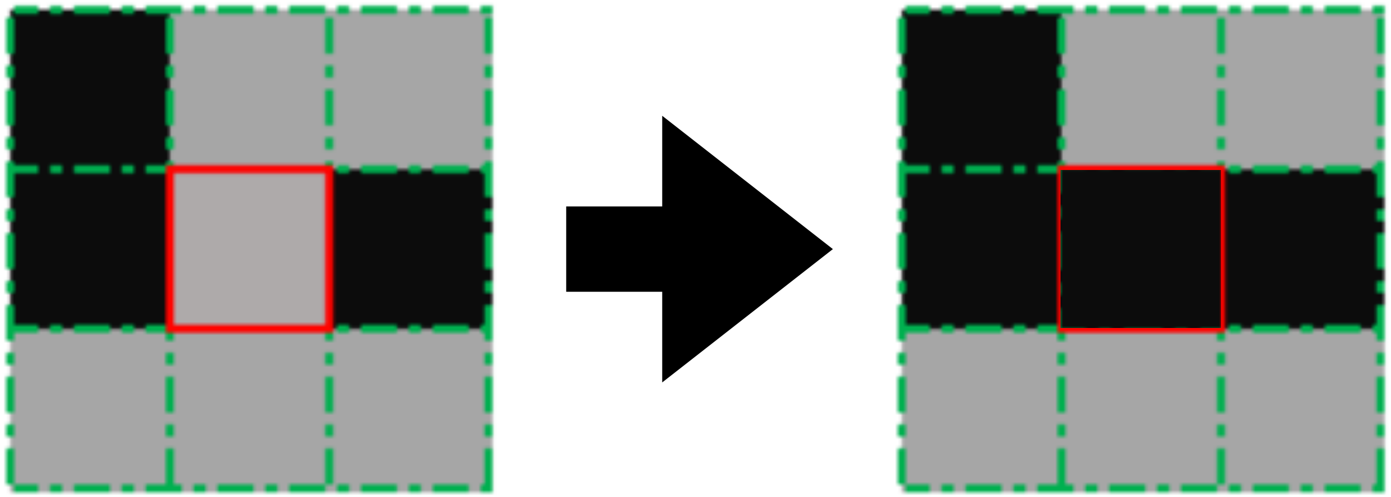
Non-Max Suppression image (left) — Threshold result (right): weak pixels in gray and strong ones in white.

5. Edge Tracking by Hysteresis

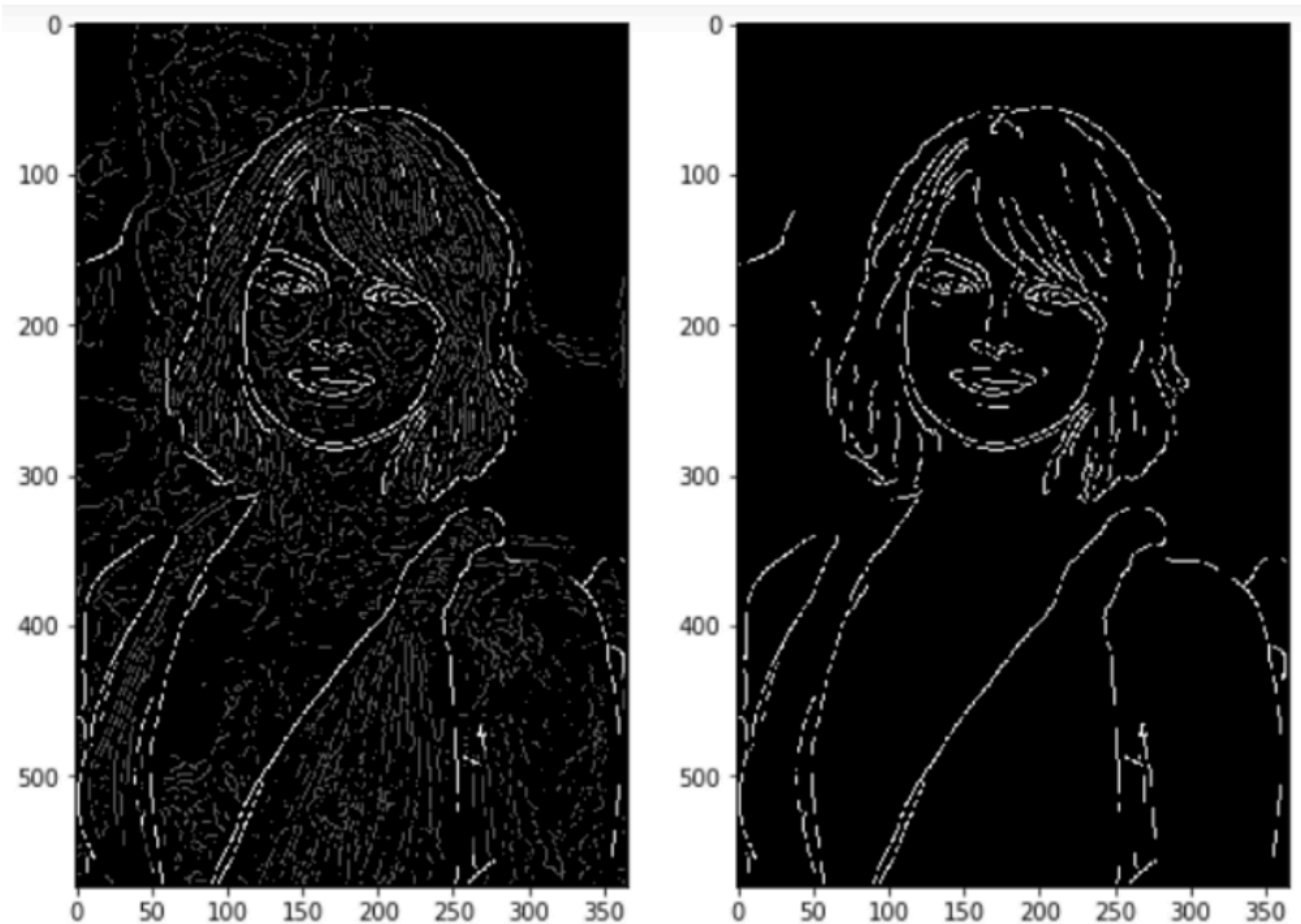
Strong pixel around



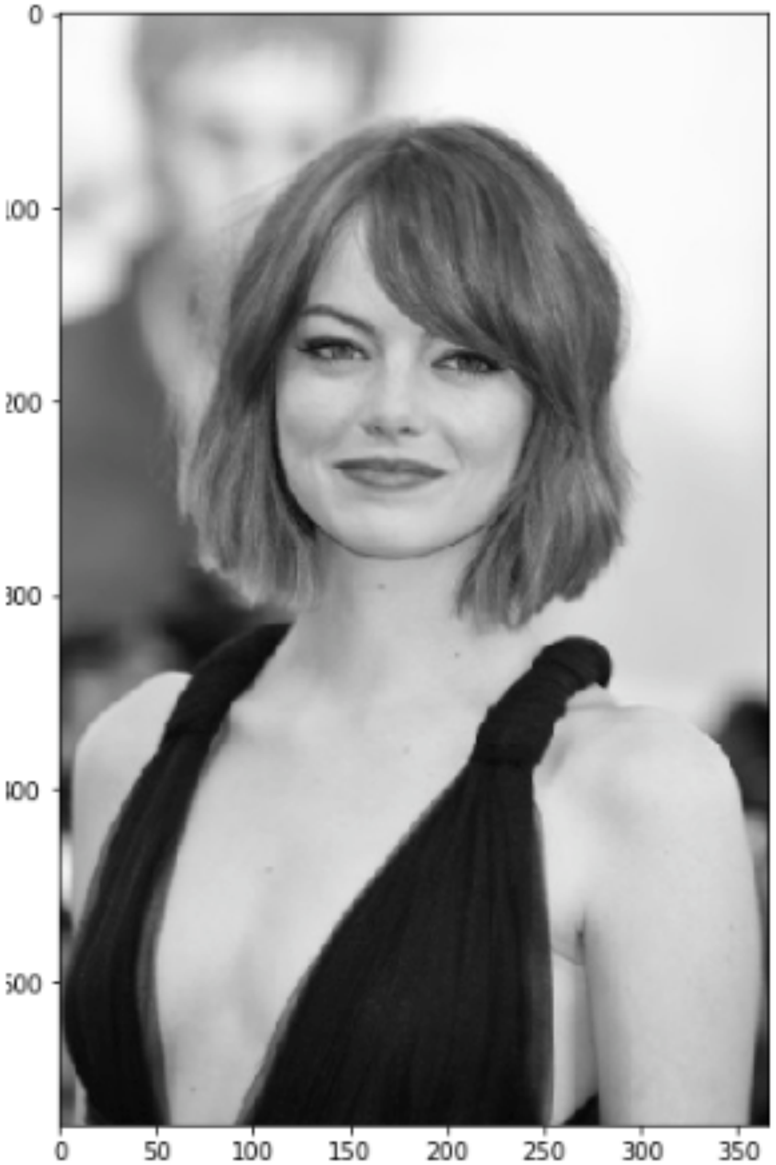
NO Strong pixel around



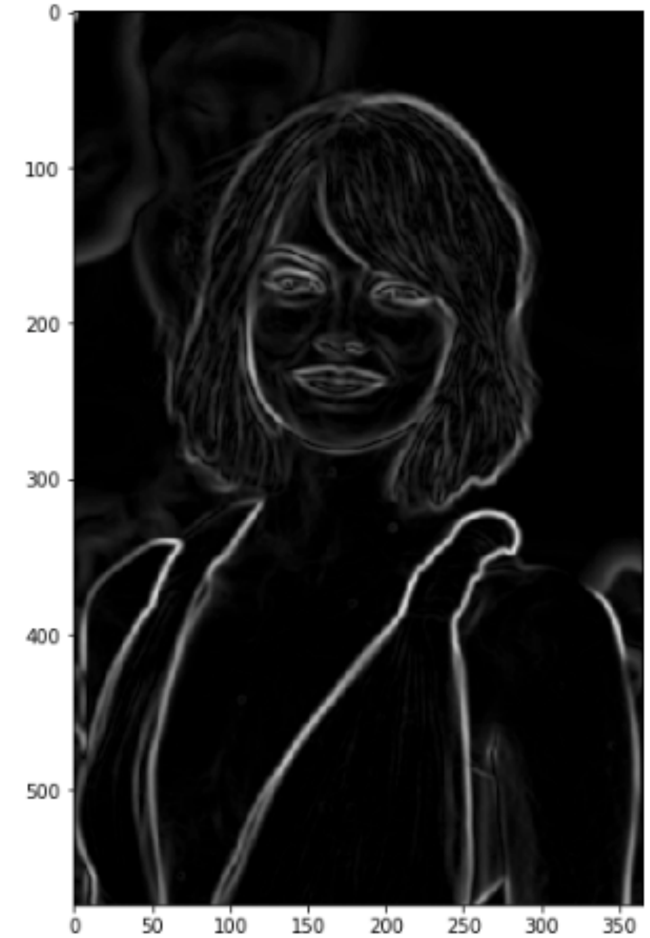
5. Edge Tracking by Hysteresis



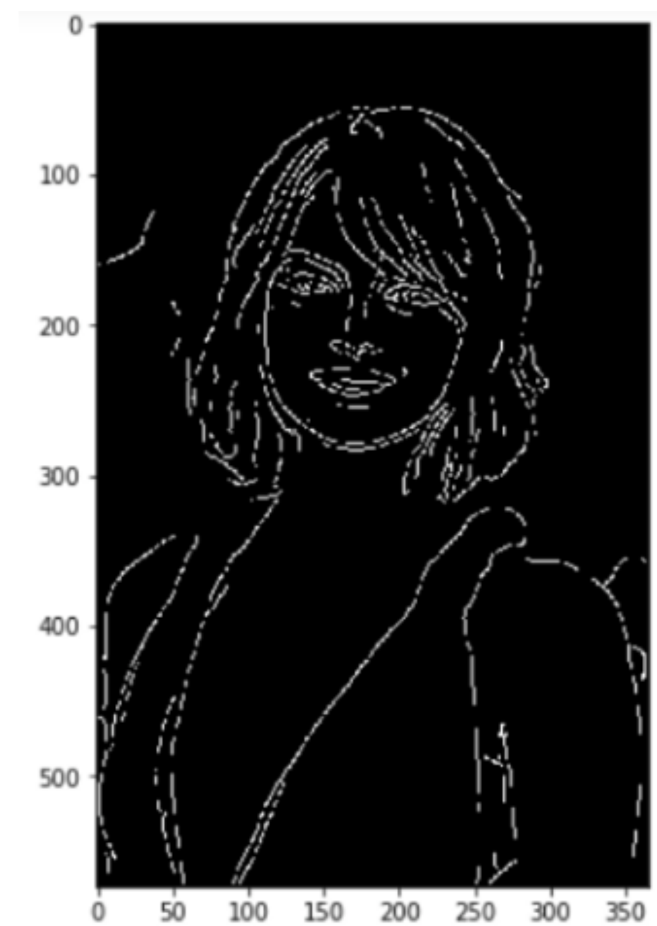
Results of hysteresis process



Gradient (Sobel)



Canny



Demo canny

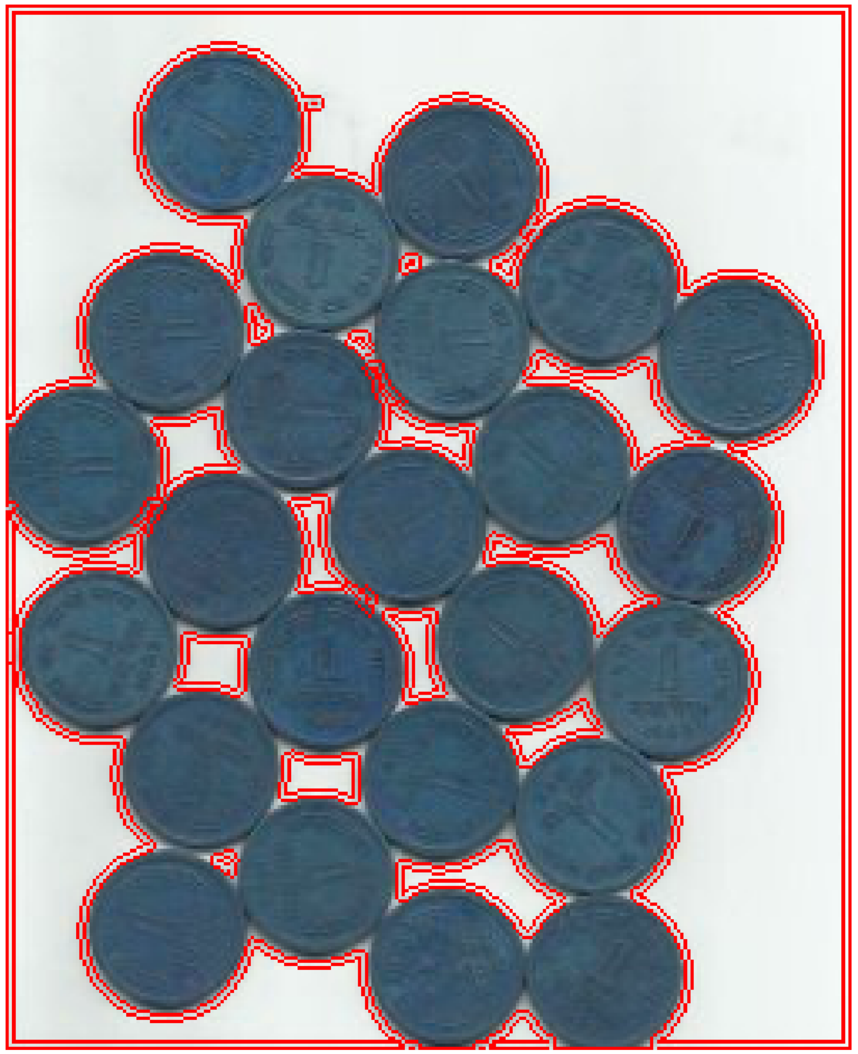
Second segmentation

Edge detection + contour

Watershed segmentation

Threshold segmentation

Edge segmentation

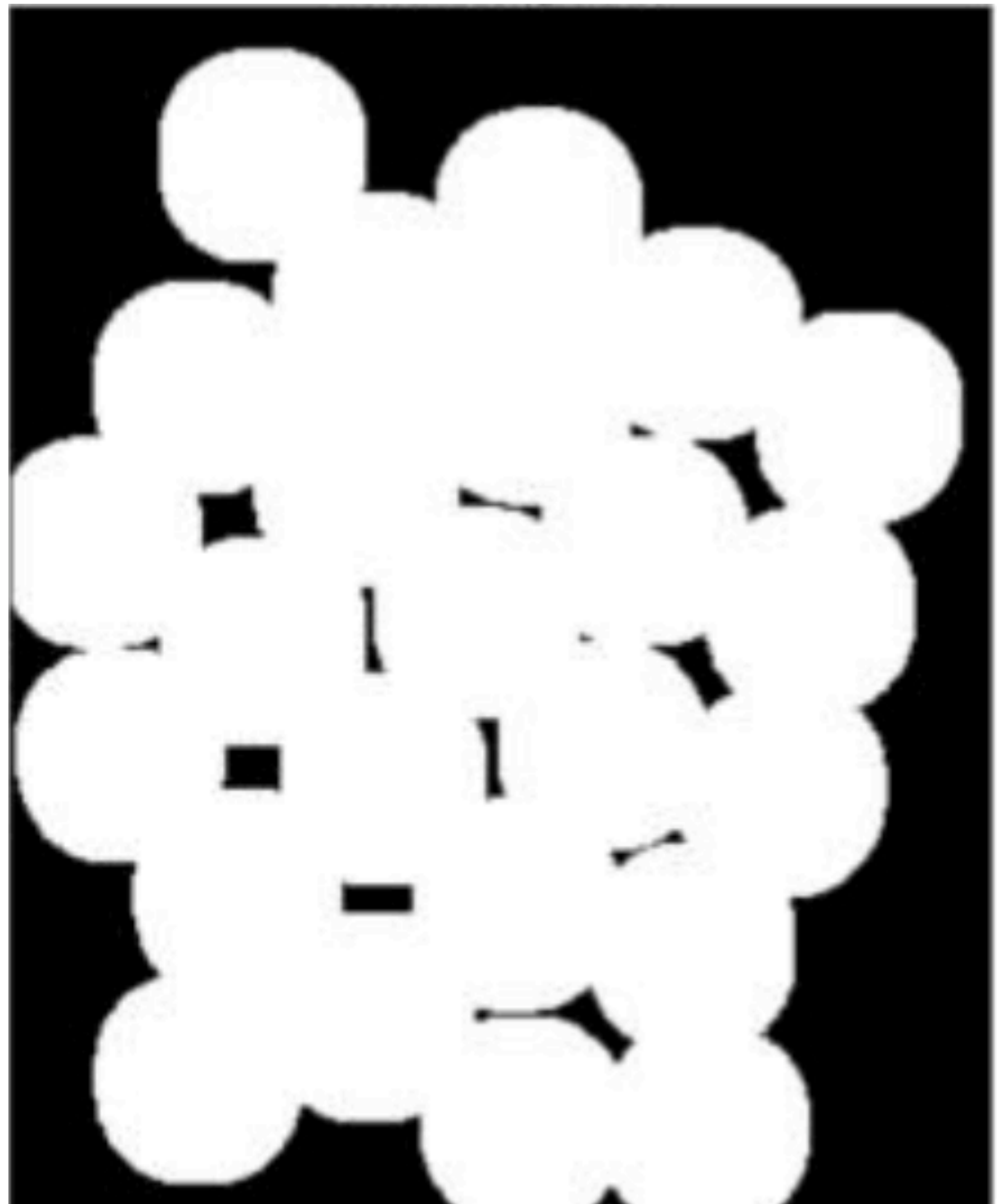


Contours are touching each other, how to separate the coins contours ?

https://docs.opencv.org/4.x/d3/db4/tutorial_py_watershed.html

Watershed segmentation

Otsu thresholding + opening + dilate

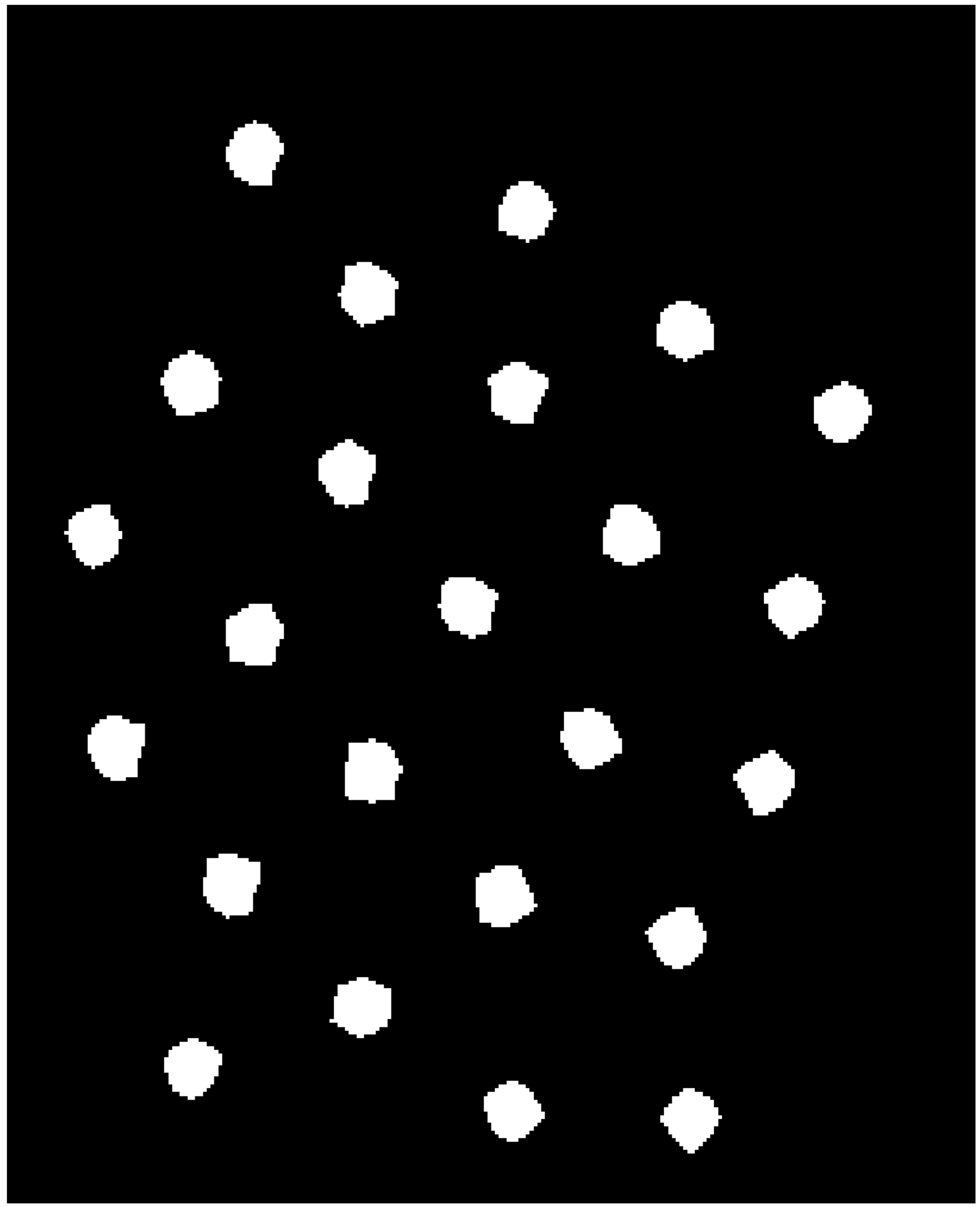


All black pixel are definitely background pixel (i.e. not coins) —> sure_bg

Watershed segmentation

Otsu thresholding + opening + distance transform + thresholding

Distance transform = derived representation of a binary image, where the value of each pixel is replaced by its distance to the nearest background pixel



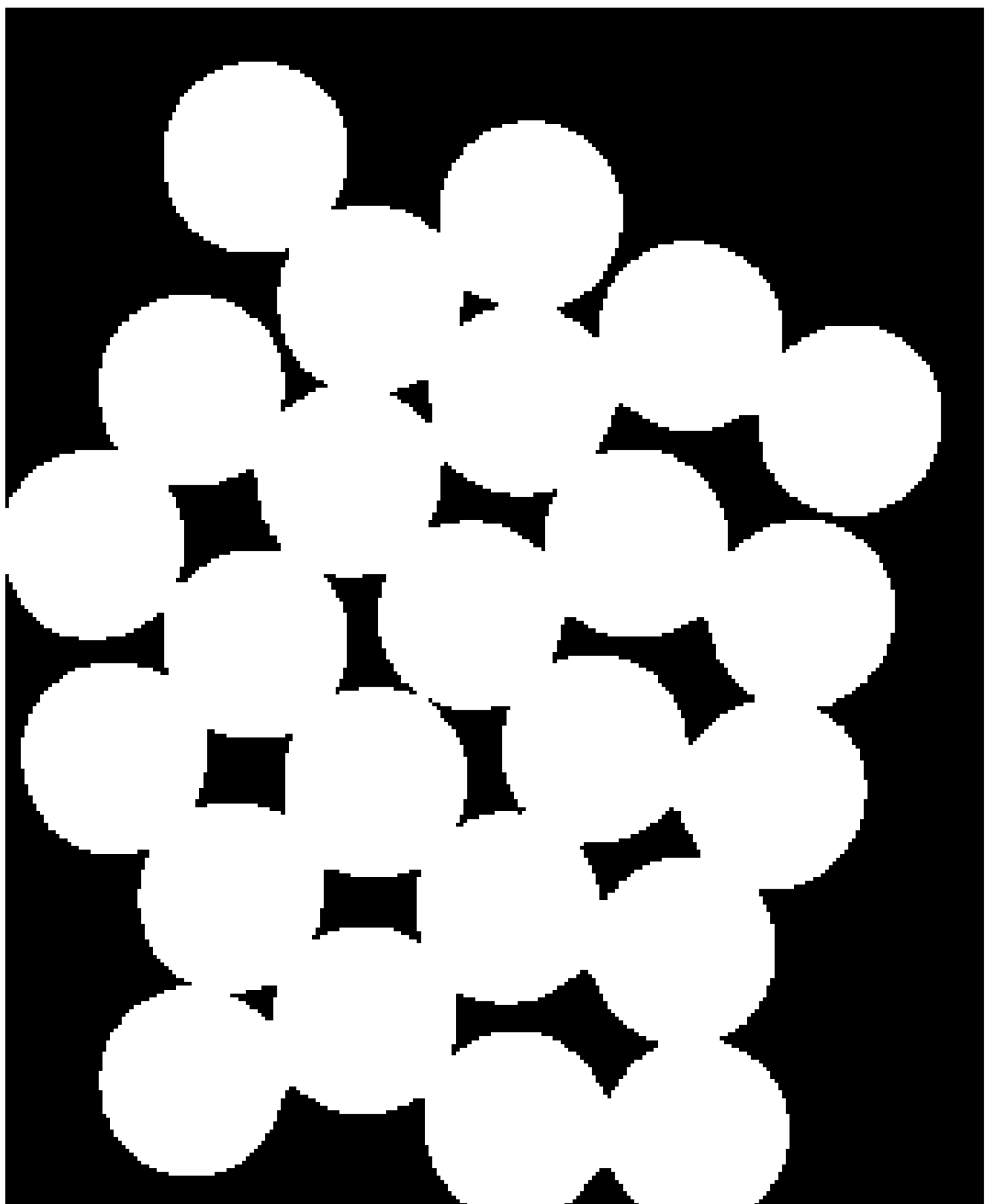
All white pixel are definitely foreground pixel (i.e. coins) —> sure_fg

Watershed segmentation

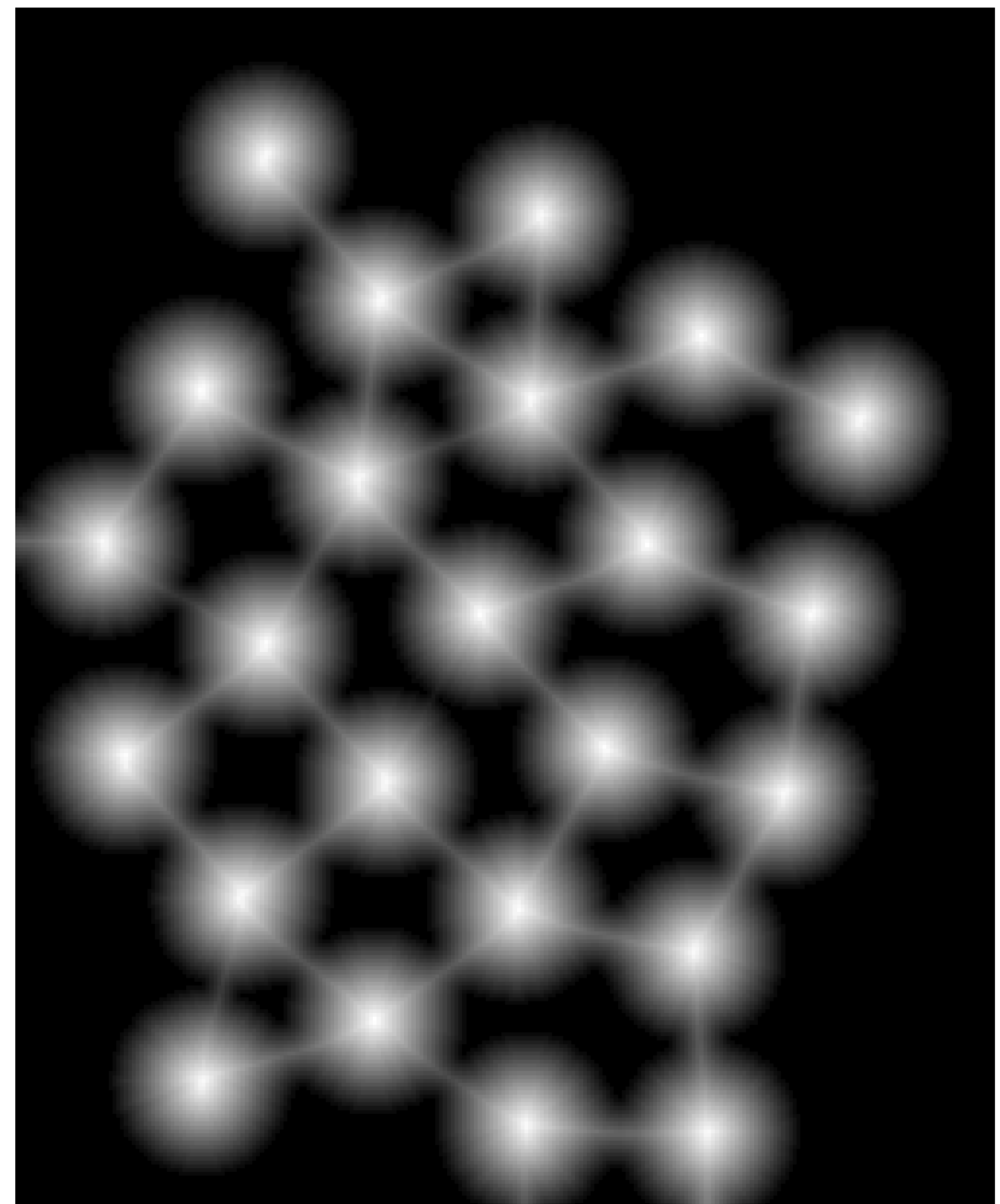
Distance transform

Distance transform = derived representation of a binary image, where the value of each pixel is replaced by its distance to the nearest background pixel

Opening

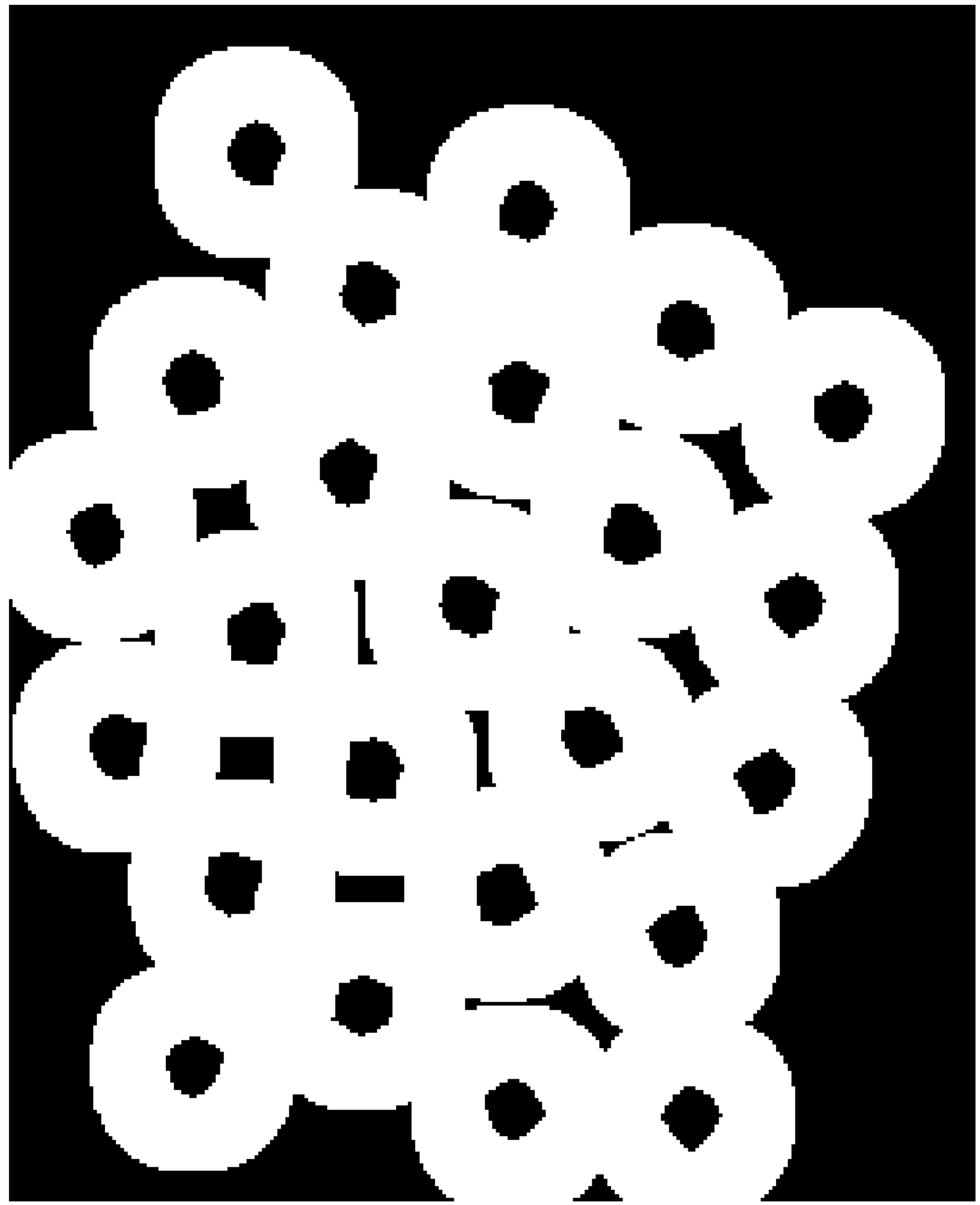


Distance tranform



Watershed segmentation

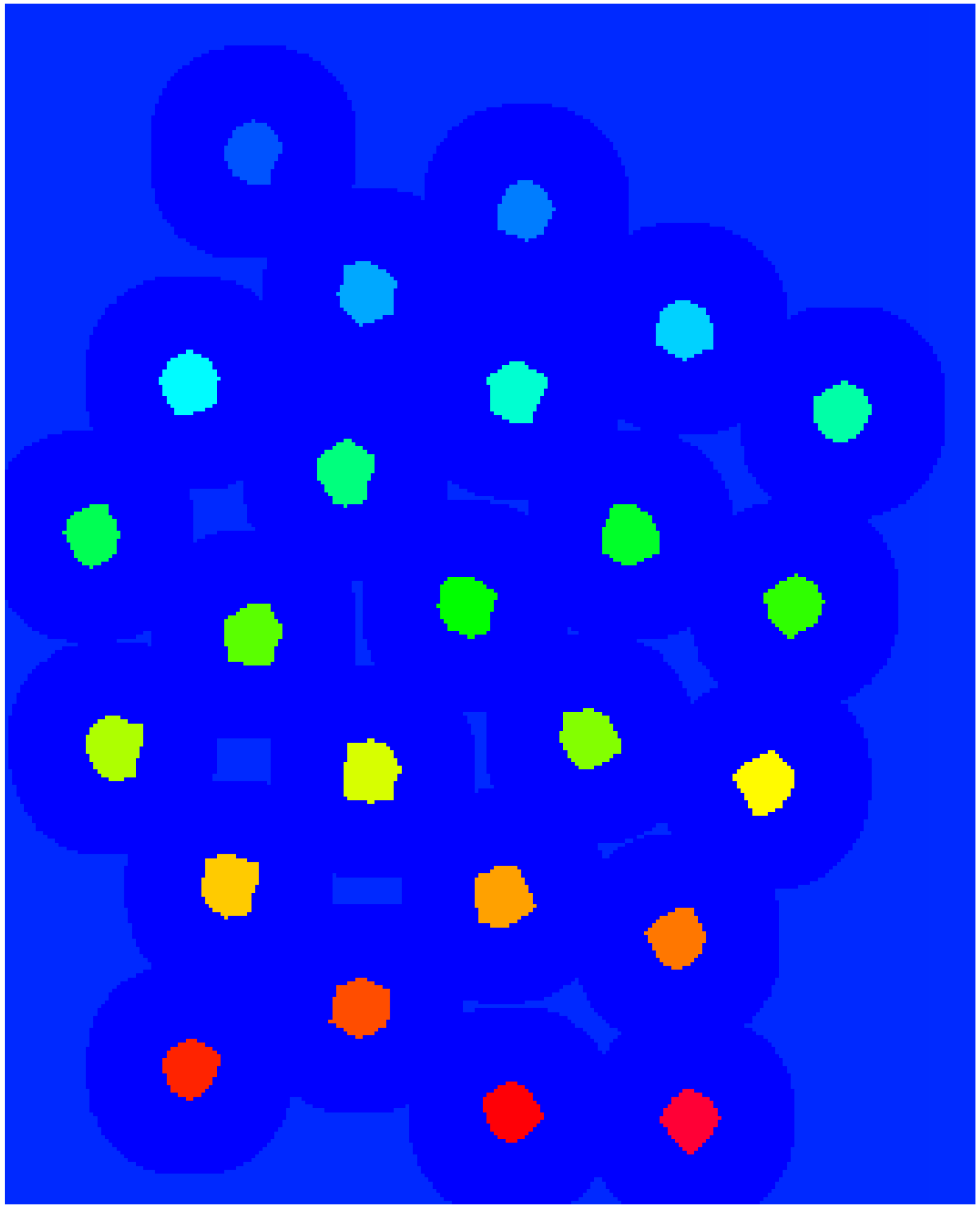
Unknown = sure_bg-sure_fg



All white pixel are unknown, we must decide for each pixel

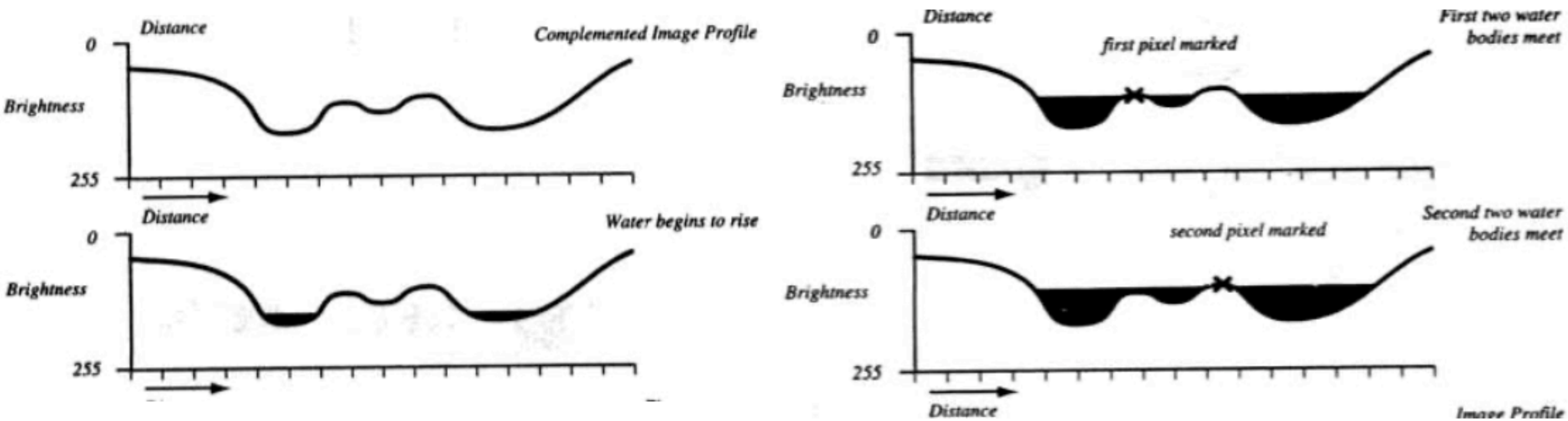
Watershed segmentation

Markers: one color for each connected region



Watershed segmentation

Final step: flooding each region to find ridge line



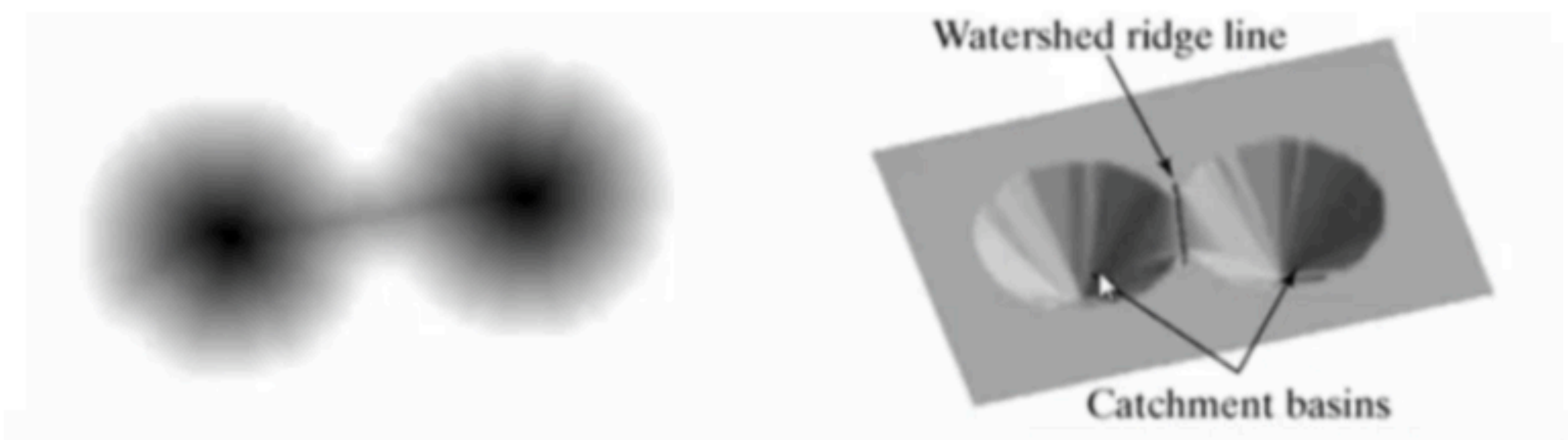


Figure 2. Visualising the watershed: the image on the left can be topographically represented as the image on the right. Source: (Agarwal, 2015)

Demo watershed

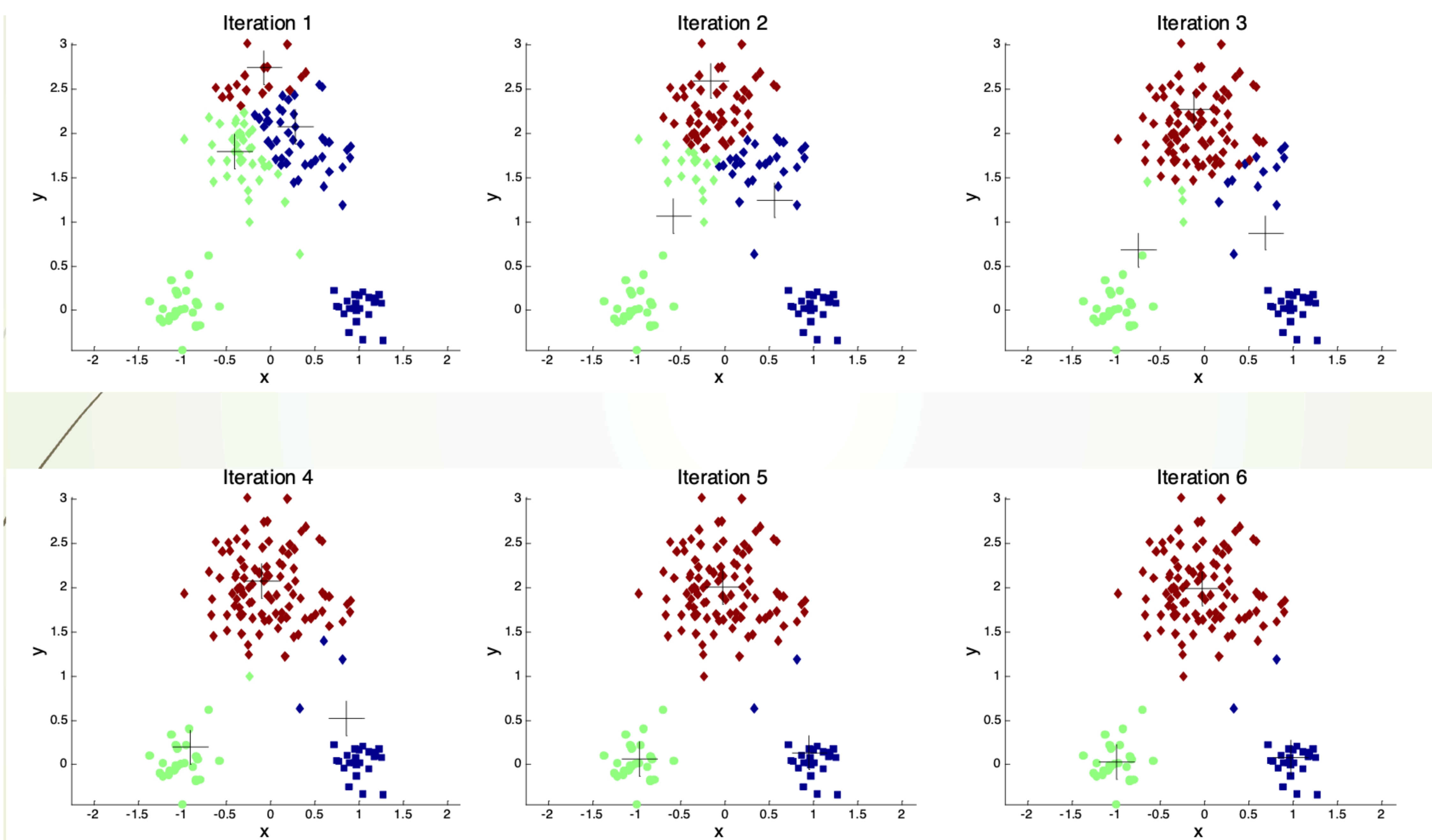
Third segmentation

Watershed

Clustering segmentation

What is clustering (K-means) ?

```
Select K points as the initial centroids.
repeat
    Form K clusters by assigning all points to the closest centroid.
    Recompute the centroid of each cluster.
until The centroids don't change
```

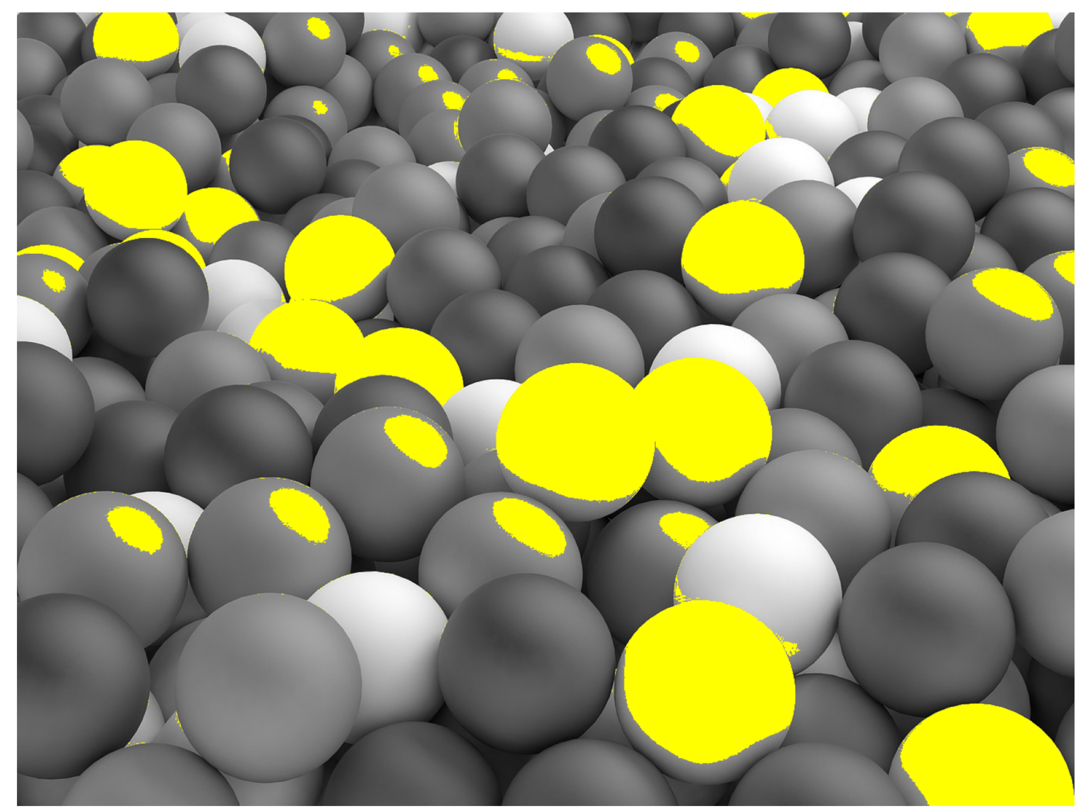


https://cs.csub.edu/~clei/teaching/files/DataScience/Kmeans/Kmeans.pptx

Clustering segmentation

K-means on color image —> color quantization, i.e. reduce number of colors

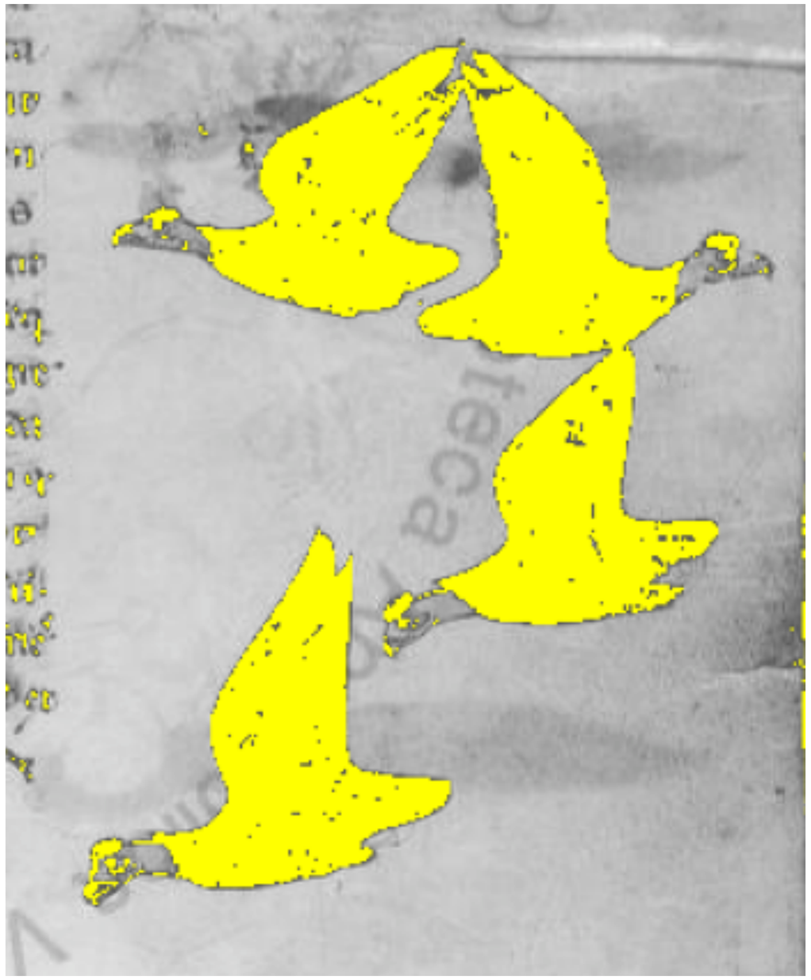
Example: K=7, display only the class that corresponds to yellow

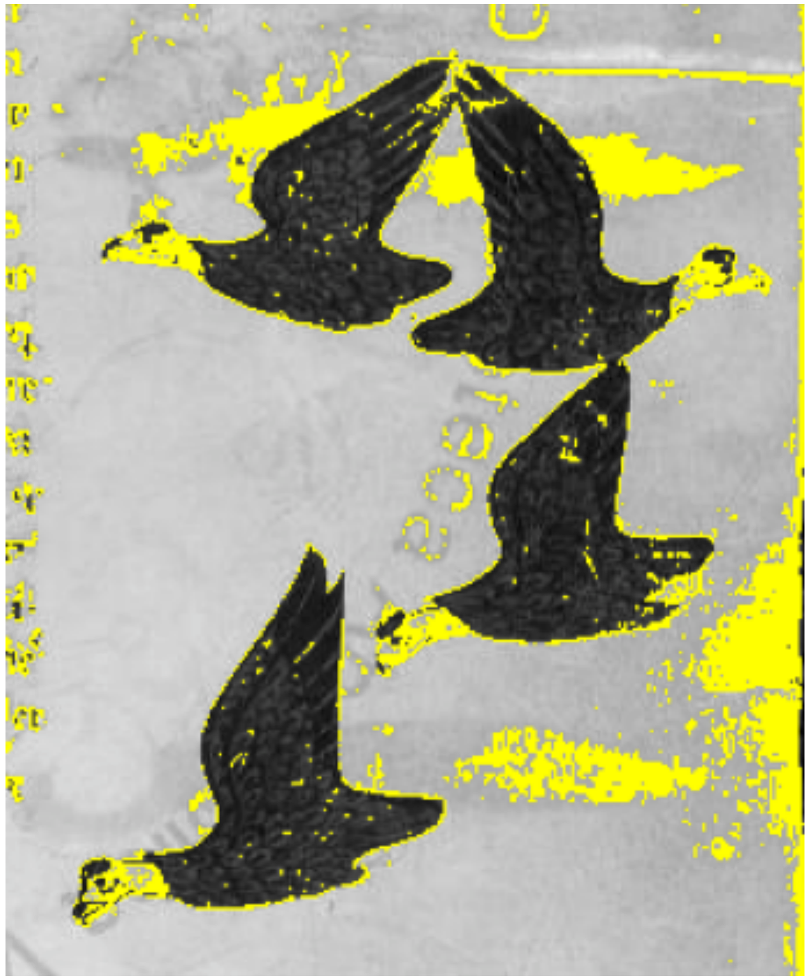


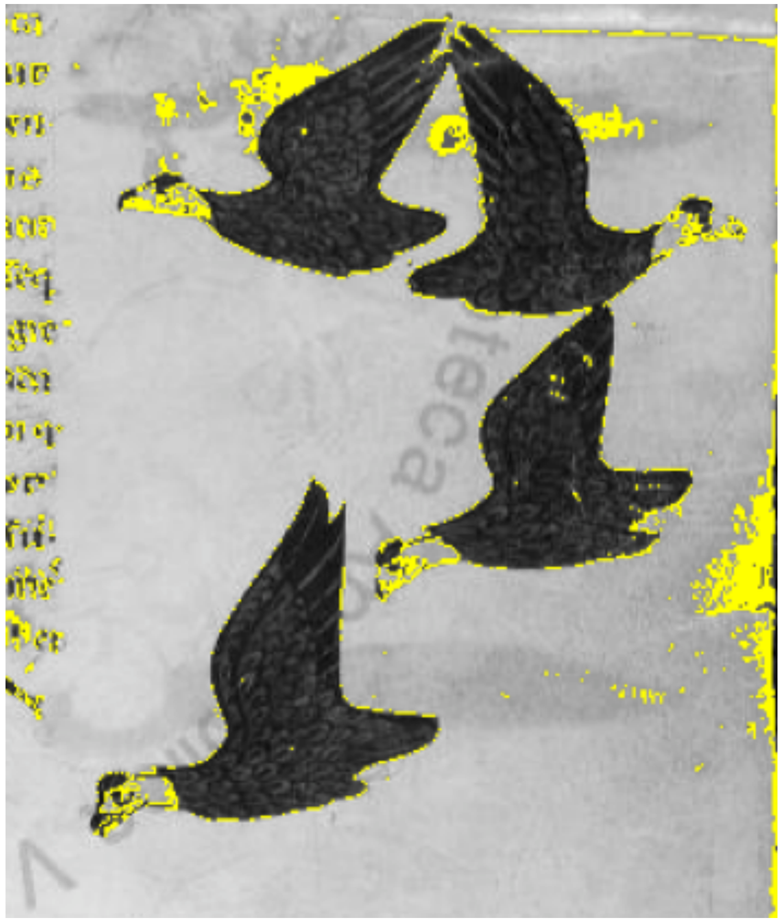
https://cs.csub.edu/~clei/teaching/files/DataScience/Kmeans/Kmeans.pptx

Fourth segmentation

Clustering (K=3)

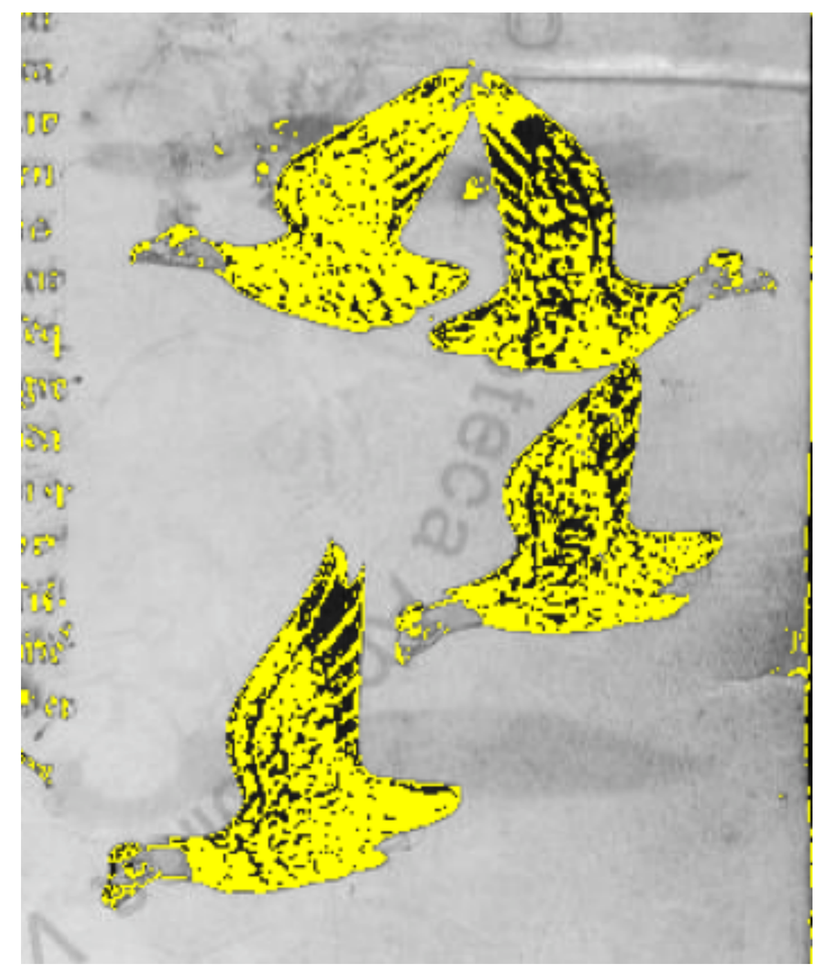






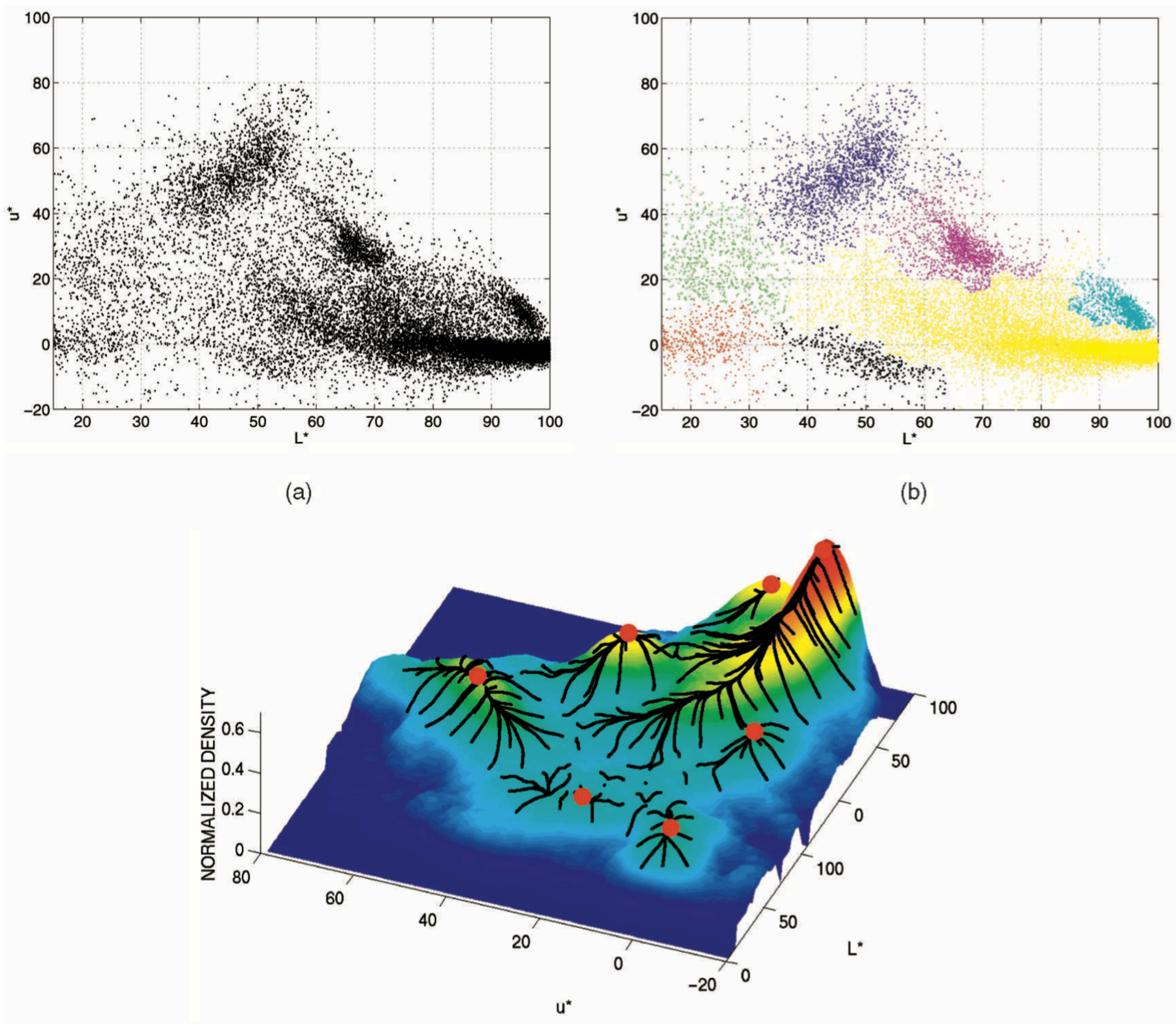
Fourth segmentation

Clustering (K=5)



Mean Shift segmentation

Variante non-parametrica al clustering

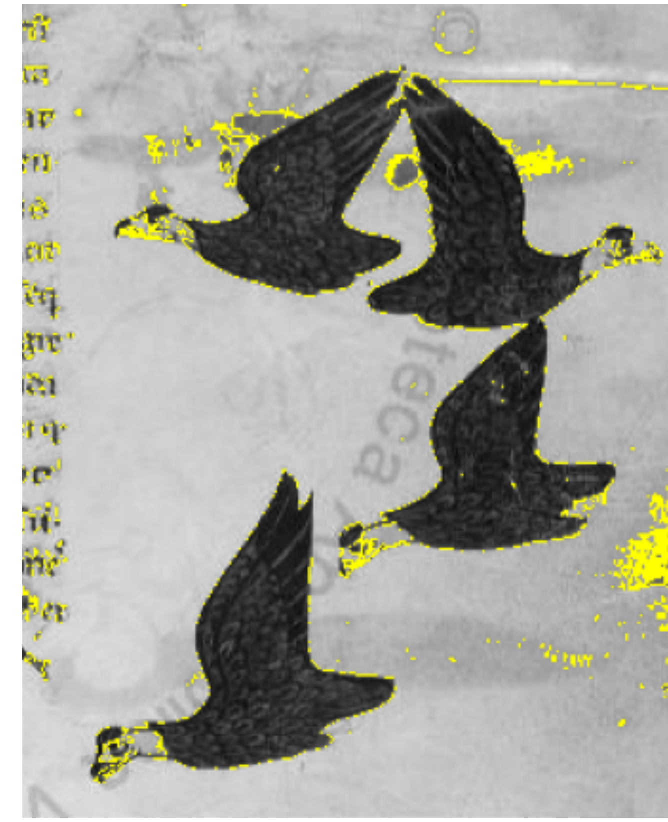


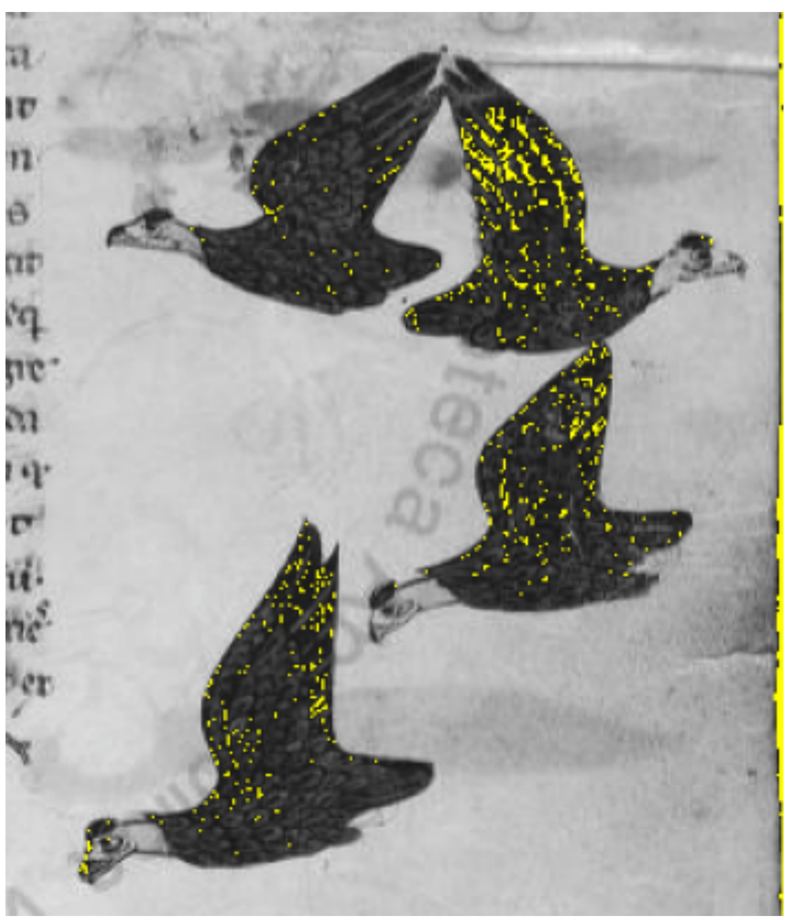
No need to pre-define the number of clusters

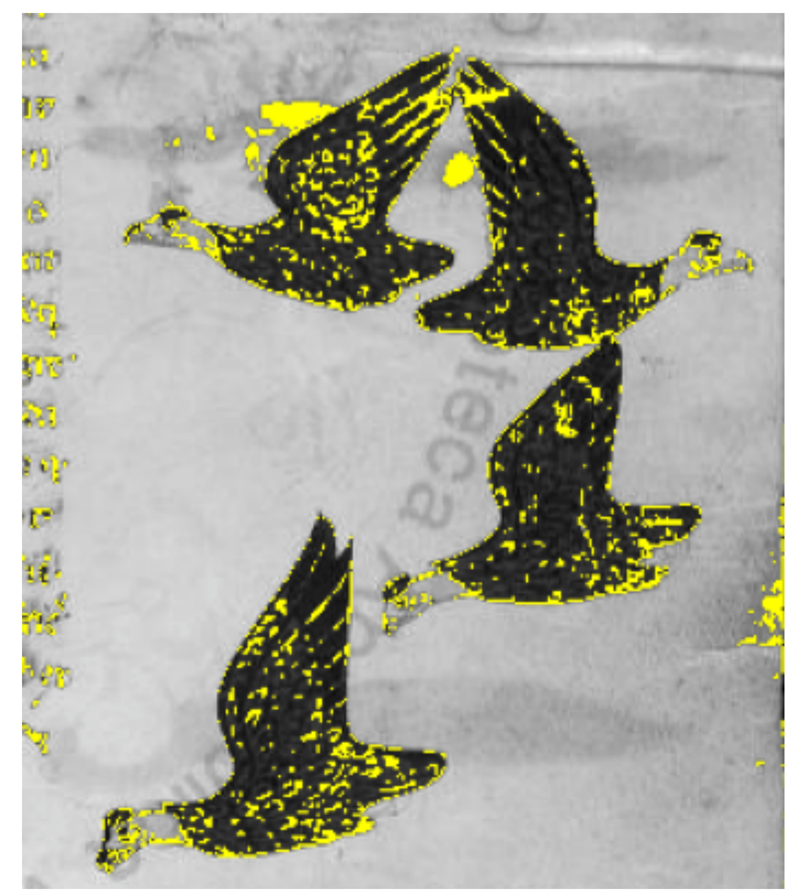
“Mean Shift: A robust approach toward feature space analysis”. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002

Fifth segmentation

Mean Shift (number of clusters=9)







Demo clustering+mean shift

Bibliografia

Libri

Rafael C. Gonzalez, Richard E. Woods, Digital image processing. 4th edition. Pearson, 2018

Richard Szeliski, Computer Vision: Algorithms and Applications. Springer, 2022

Articoli

Comaniciu, D.; Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. , 24(5), 0–619. doi:10.1109/34.1000236

Thiago Teixeira Santos, SciPy and OpenCV as an interactive computing environment for computer vision 2015
DOI:10.22456/2175-2745.49491

Shipitko, Oleg & Grigoryev, Anton. (2018). Gaussian filtering for FPGA based image processing with High-Level Synthesis tools.

Tutorial e slides web

https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html

https://pyimagesearch.com/2021/02/22/opencv-connected-component-labeling-and-analysis/

https://gmanco.github.io/courses/computervision/

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

https://docs.opencv.org/4.x/d3/db4/tutorial_py_watershed.html

https://www.youtube.com/watch?v=K5P5rjDiZzk

https://cs.csub.edu/~clei/teaching/files/DataScience/Kmeans/Kmeans.pptx