Part 3: Image Processing

Image Filtering and Segmentation

Georgy Gimel'farb

COMPSCI 373 Computer Graphics and Image Processing





Why Image Filtering?




Noise removal, contrast sharpening, contour highlighting, edge detection, ...

- Other uses?

Linear and nonlinear image filters

Linear filters (also known as convolution filters):

- Can be represented with vector-matrix multiplication.

Nonlinear filters:

- Non-linear functions of signals.
- Examples: thresholding, image equalisation, or median filtering.

Basic Principles of Filtering

The output filtered image $\mathbf{g}$ is obtained from an input image $\mathbf{f}$ by:

1. Transforming the value $f(x, y)$ at a given location (x, y) by using this value and values in K other pixels:

$$f(x + \xi, y + \eta) : \ (\xi, \eta) \in \mathbf{N} = \{(\xi_1, \eta_1), \ldots, (\xi_K, \eta_K)\}$$

Each value $g(x, y)$ is a function of these $K + 1$ values from f.

2. Repeating the same transformation for each pixel $(x, y) \in \mathbf{R}$

Examples: a linear combination (or a weighted sum with the weights $w_0, \ldots, w_K$):

$$g(x, y) = w_0 f(x, y) + \sum_{k=1}^{K} w_k f(x + \xi_k, y + \eta_k);$$

an average, or mean ($w_k = \frac{1}{K+1}: \ k = 0, 1, \ldots, K$); a median; ...

Basic Principles of Filtering

The $K+1$ pixels (x, y) and $\{(x+\xi, y+\eta) : (\xi, \eta) \in \mathbf{N}\}$, used to compute the output signal $g(x, y)$, form a 2D pattern, which is frequently called a moving window.

- $\{(0, 0); \mathbf{N}\}$ – all the coordinate offsets for a moving window.
- $(0, 0)$ is the window centre:
  - The window is centred at each location (x, y).

Popular windows – rectangles, e.g., 3×3, 5×5, 3×1, etc.:

- The 3×3 square window $\Rightarrow$ the offsets

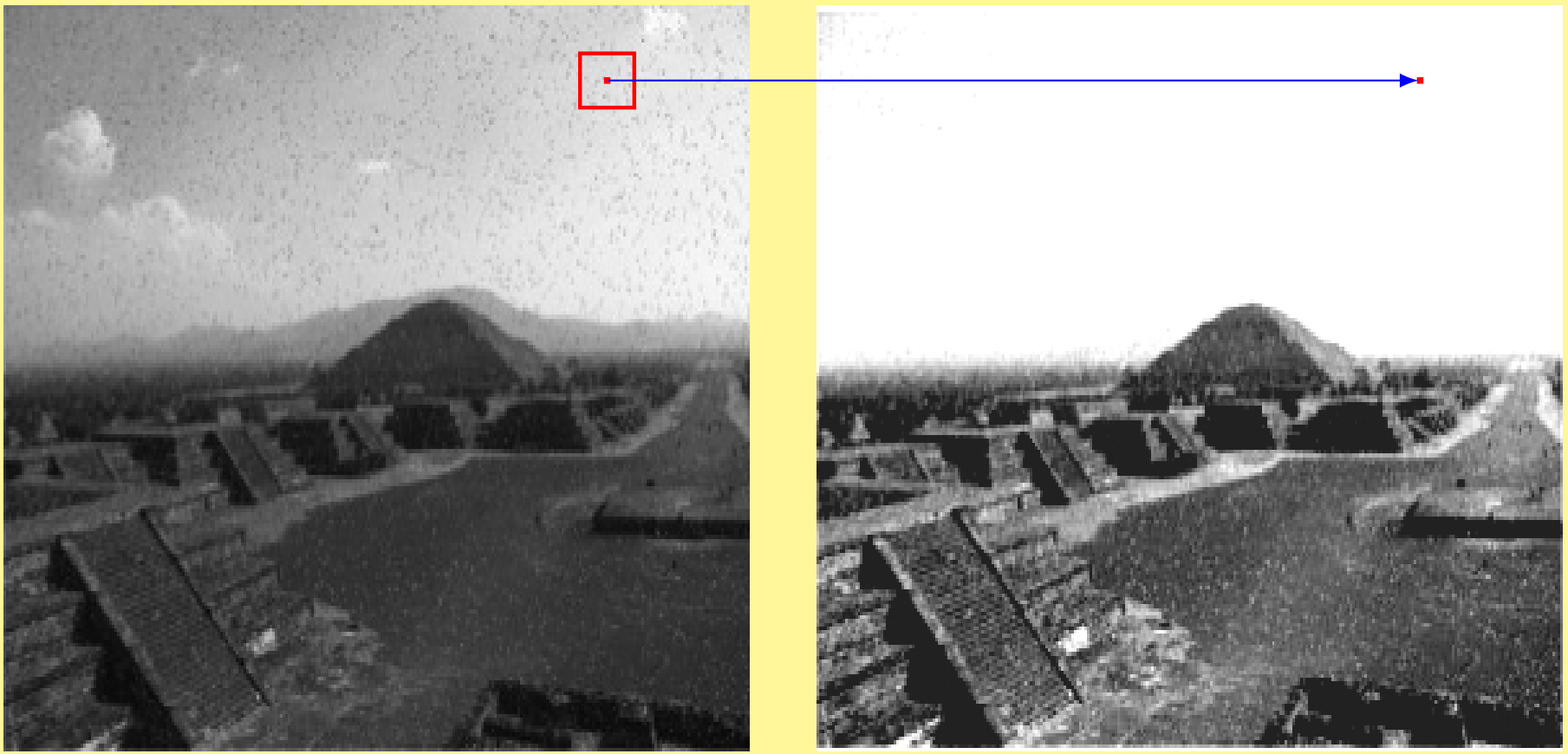
$$\{(\xi, \eta) : \xi = -1, 0, 1; \eta = -1, 0, 1\}$$

- The $(2k+1) \times (2l+1)$ rectangular window $\Rightarrow$ the offsets

$$\{(\xi, \eta) : \xi = -k, \ldots, 0, 1, \ldots, k; \ \eta = -l, \ldots, 0, 1, \ldots, l\}$$

Basic Principles of Filtering

Input image $\mathbf{f}$ Output image $\mathbf{g}$



$$g(x,y) = T\left(f(x,y), f(x+\xi_1, y+\eta_1), \ldots, f(x+\xi_K, y+\eta_K)\right)$$

Median Filtering

Effective nonlinear filtering, being used frequently to remove the "salt and pepper" image noise while preserving edges.

Replacing each $f(x, y)$ with the median of the $K+1$ neighbouring values:

$$g(x, y) = \text{median}\{f(x, y), f(x + \xi_1, y + \eta_1), \ldots, f(x + \xi_K, y + \eta_K)\}$$

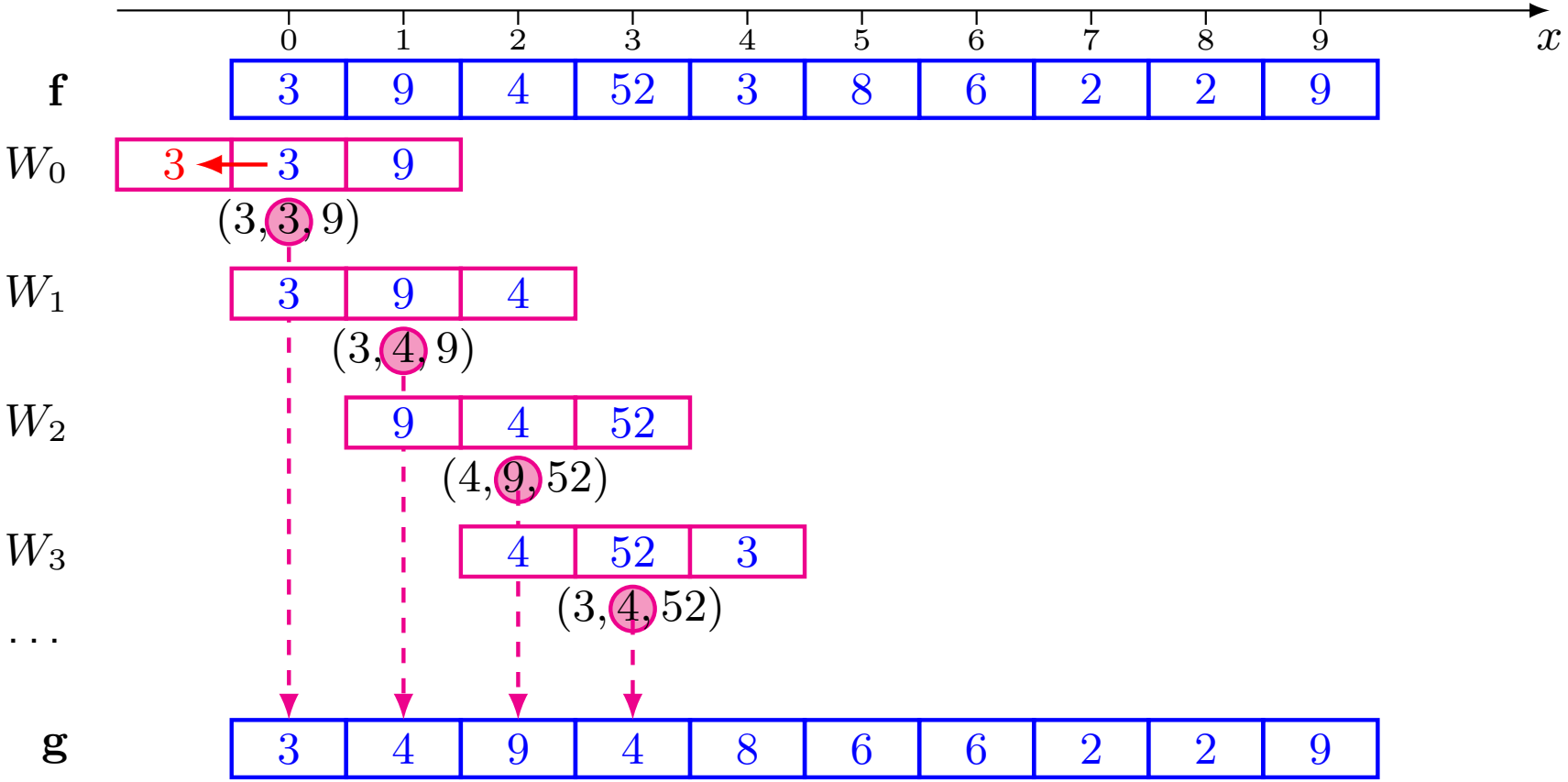
Calculating the median of the signals from the window:

1. Sorting all the $K + 1$ values from the window: $v_0 = f(x, y)$, $v_1 = f(x + \xi_1, y + \eta_1)$, $\ldots$, $v_K = f(x + \xi_K, y + \eta_K)$, into ascending numerical order[a]: $v_{[0]} \leq v_{[1]} \leq \ldots \leq v_{[K]}$.
2. Select the middle value, $g(x, y) = v_{[K/2]}$, if K is even or the average, $g(x, y) = 0.5\left(v_{[(K-1)/2]} + v_{[(K+1)/2]}\right)$, of the two middle values otherwise.

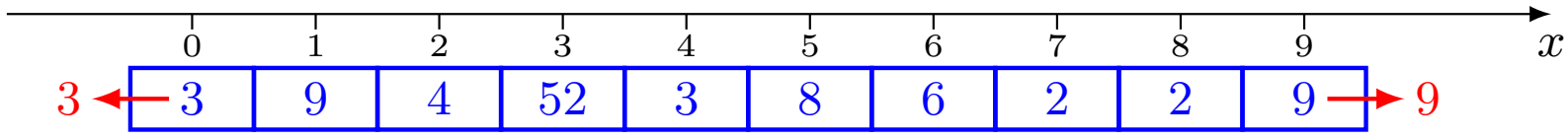
[a] Here, $[i]$ denotes the i^{th} position in the sorted array.

Median Filtering: An One-Dimensional (1D) Example

Median filtering of a simple 1D signal $\mathbf{f} = (f(x) : x = 0, 1, \ldots, 9)$ with a moving window $W_x = [x - 1, x, x + 1]$ of size 3.



Median Filtering: One-Dimensional (1D) Example

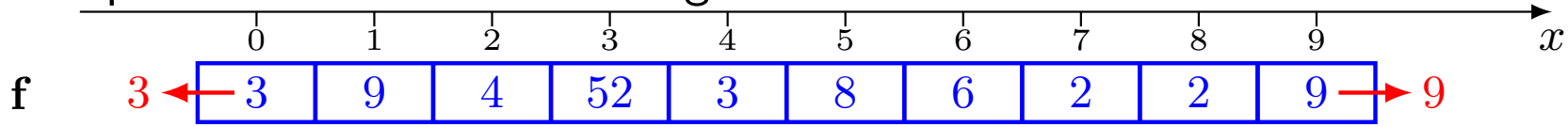


Moving window			Sorting			Selecting the median and assigning to $g(x)$		
W_0 :	$(3, 3, 9)$	$\Rightarrow$	$(3, \mathbf{3}, 9)$	$\Rightarrow$	$g(0) = \text{median}\{3, 3, 9\}$	$=$	3
W_1 :	$(3, 9, 4)$	$\Rightarrow$	$(3, \mathbf{4}, 9)$	$\Rightarrow$	$g(1) = \text{median}\{3, 9, 4\}$	$=$	4
W_2 :	$(9, 4, 52)$	$\Rightarrow$	$(4, \mathbf{9}, 52)$	$\Rightarrow$	$g(2) = \text{median}\{9, 4, 52\}$	$=$	9
W_3 :	$(4, 52, 3)$	$\Rightarrow$	$(3, \mathbf{4}, 52)$	$\Rightarrow$	$g(3) = \text{median}\{4, 52, 3\}$	$=$	4
W_4 :	$(52, 3, 8)$	$\Rightarrow$	$(3, \mathbf{8}, 52)$	$\Rightarrow$	$g(4) = \text{median}\{52, 3, 8\}$	$=$	8
W_5 :	$(3, 8, 6)$	$\Rightarrow$	$(3, \mathbf{6}, 8)$	$\Rightarrow$	$g(5) = \text{median}\{3, 8, 6\}$	$=$	6
W_6 :	$(8, 6, 2)$	$\Rightarrow$	$(2, \mathbf{6}, 8)$	$\Rightarrow$	$g(6) = \text{median}\{8, 6, 2\}$	$=$	6
W_7 :	$(6, 2, 2)$	$\Rightarrow$	$(2, \mathbf{2}, 6)$	$\Rightarrow$	$g(7) = \text{median}\{6, 2, 2\}$	$=$	2
W_8 :	$(2, 2, 9)$	$\Rightarrow$	$(2, \mathbf{2}, 9)$	$\Rightarrow$	$g(8) = \text{median}\{2, 2, 9\}$	$=$	2
W_9 :	$(2, 9, 9)$	$\Rightarrow$	$(2, \mathbf{9}, 9)$	$\Rightarrow$	$g(9) = \text{median}\{2, 9, 9\}$	$=$	9

For $g(0)$ and $g(9)$, $f(0)$ and $f(9)$, respectively, are extended outside the boundaries.

Median Filtering

The boundary (left- and right-most) values were repeated in the previous example in order to obtain enough entries to fill the window.



- What effect does this have on the boundary values?

Other approaches that might be preferred in particular circumstances:

- Avoid processing the boundaries, with or without cropping the signal or image boundary afterwards.
- Fetching entries from other places in the signal (image **padding**).
  - E.g., selecting entries from the far horizontal or vertical image boundary (padding by twisting the image plane into a torus).
- Shrinking the window near the boundaries, so that every window is full.

What effects might these approaches have on the boundary values?

Median Filtering: Boundary Effects

Large "salt-and-pepper" noise.

Median filtering result.

There is some remaining noise on the boundary of the image. Why is this?

2D Median Filtering: 3×3 Window

Keeping border values unchanged (processing no border pixels).

Sorted: (0, 0, 1, 1, **1**, 2, 2, 4, 4)

1	4	0	1	3	1
2	2	4	2	2	3
1	0	1	0	1	0
1	2	1	0	2	2
2	5	3	1	2	5
1	1	4	2	3	0

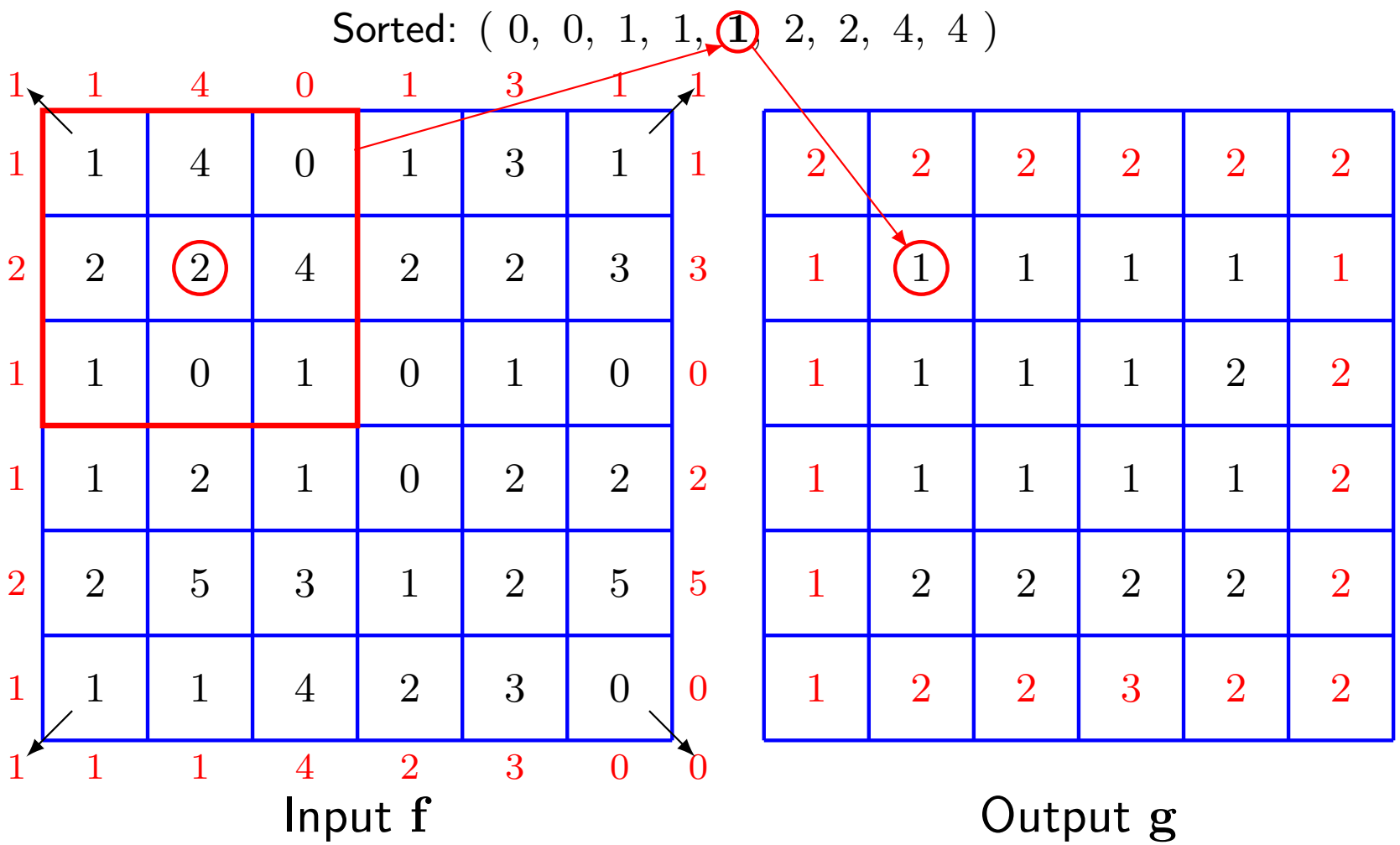
Input $\mathbf{f}$

1	4	0	1	3	1
2	1	1	1	1	3
1	1	1	1	2	0
1	1	1	1	1	2
2	2	2	2	2	5
1	1	4	2	3	0

Output $\mathbf{g}$

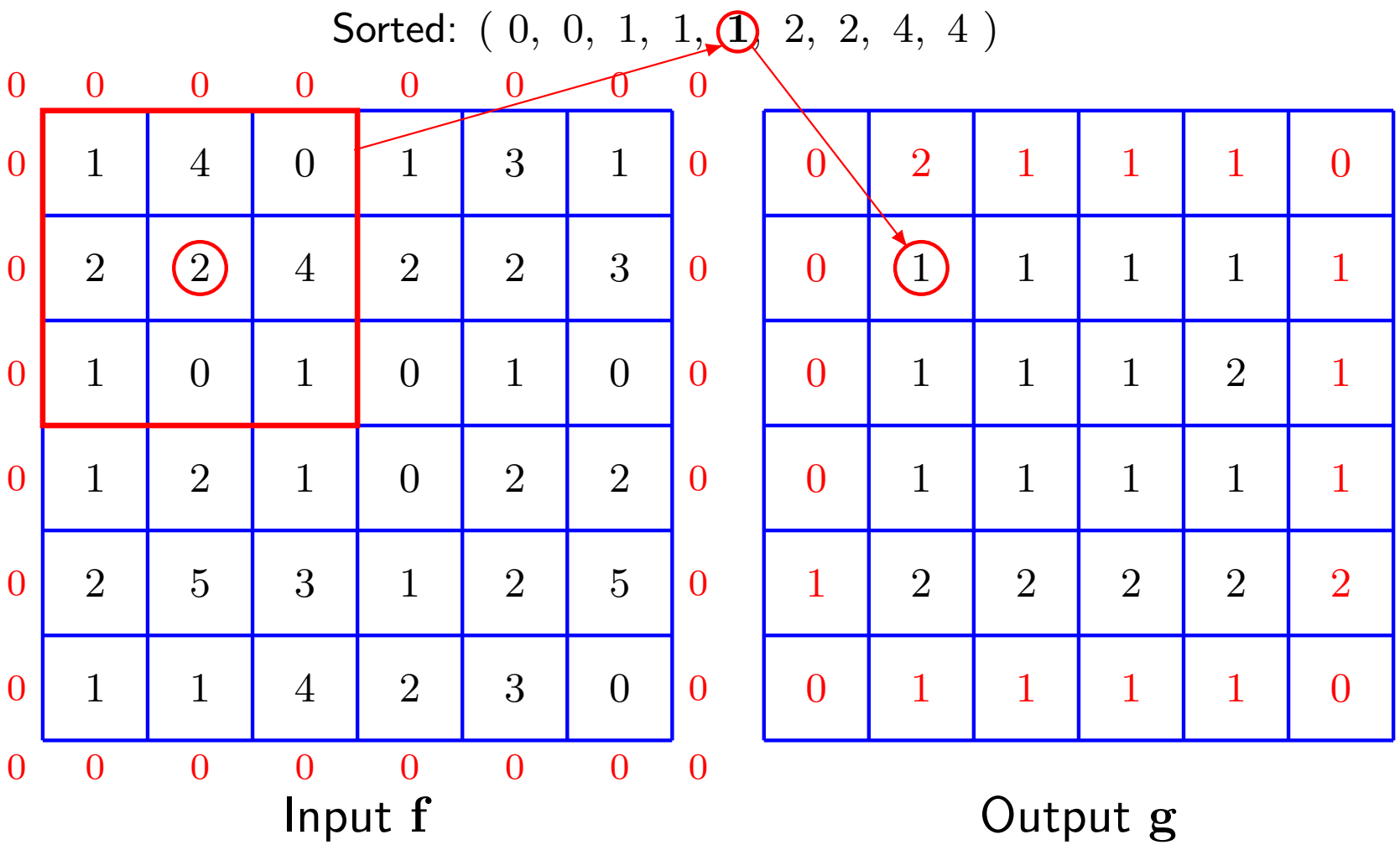
2D Median Filtering: 3×3 Window + Padding 1

1: Extending border values outside with the boundary values.



2D Median Filtering: 3×3 Window + Padding 2

2: Extending border values outside with zeros.



Mean Filtering

Mean, or average filtering:

"Smoothing" images by reducing the variation of intensities between the neighbouring pixels $\{(x, y); (x + \xi_1, y + \eta_1); \ldots, (x + \xi_K, y + \eta_K)\}$.

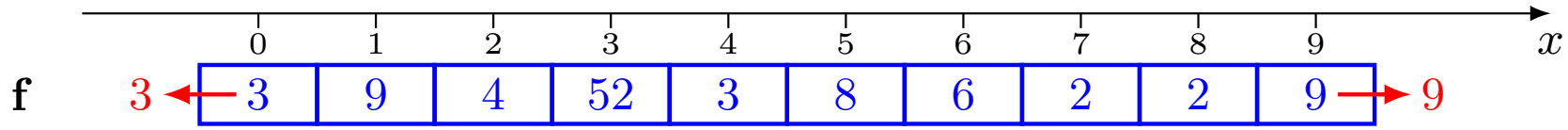
Each value is replaced with the average value of the neighbouring pixels, including itself:

$$g(x, y) = \frac{1}{K+1} \left(f(x, y) + \sum_{k=1}^{K} f(x + \xi_k, y + \eta_k) \right)$$

Potential problems:

- A single "outlier" (a very outstanding value) can significantly affect the average of all the pixel values in its neighbourhood.
- Edge blurring: when the moving window crosses an edge, the filter will interpolate new pixel values on the edge.
  - This may be a problem if sharp output edges are required.

Mean Filtering: One-Dimensional (1D) Example

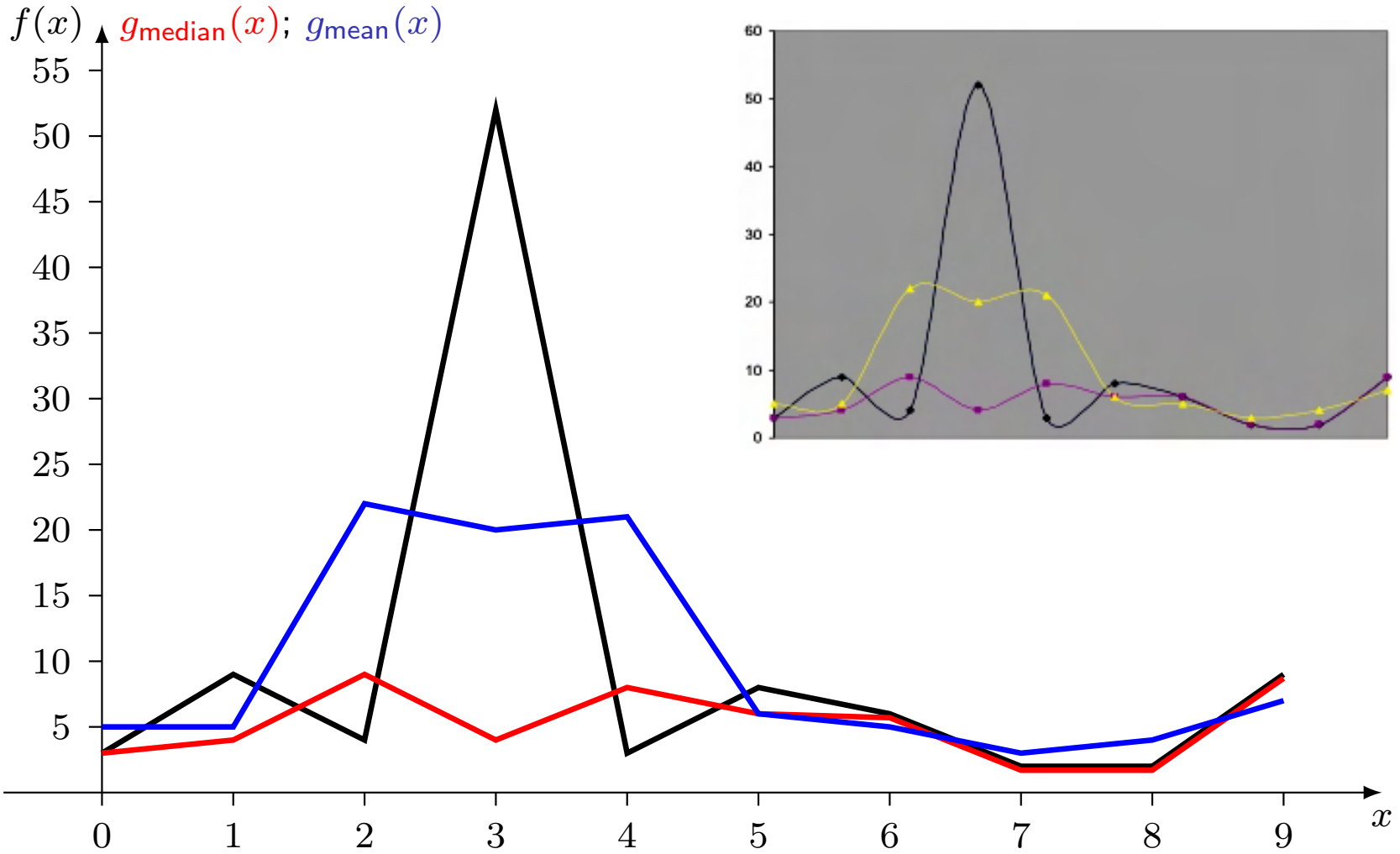


Moving window			Computing the mean		Rounding the mean and assigning to $g(x)$		
W_0 :	$(3, 3, 9)$	$\Rightarrow$	$(3+3+9)/3 = 5$	$\Rightarrow$	$g(0) = \text{round}\{5\}$	$=$	5
W_1 :	$(3, 9, 4)$	$\Rightarrow$	$(3+9+4)/3 = 5.33$	$\Rightarrow$	$g(1) = \text{round}\{5.33\}$	$=$	5
W_2 :	$(9, 4, 52)$	$\Rightarrow$	$(9+4+52)/3 = 21.67$	$\Rightarrow$	$g(2) = \text{round}\{21.67\}$	$=$	22
W_3 :	$(4, 52, 3)$	$\Rightarrow$	$(4+52+3)/3 = 19.67$	$\Rightarrow$	$g(3) = \text{round}\{19.67\}$	$=$	20
W_4 :	$(52, 3, 8)$	$\Rightarrow$	$(52+3+8)/3 = 21$	$\Rightarrow$	$g(4) = \text{round}\{21\}$	$=$	21
W_5 :	$(3, 8, 6)$	$\Rightarrow$	$(3+8+6)/3 = 5.67$	$\Rightarrow$	$g(5) = \text{round}\{5.67\}$	$=$	6
W_6 :	$(8, 6, 2)$	$\Rightarrow$	$(8+6+2)/3 = 5.33$	$\Rightarrow$	$g(6) = \text{round}\{5.33\}$	$=$	5
W_7 :	$(6, 2, 2)$	$\Rightarrow$	$(6+2+2)/3 = 3.33$	$\Rightarrow$	$g(7) = \text{round}\{3.33\}$	$=$	3
W_8 :	$(2, 2, 9)$	$\Rightarrow$	$(2+2+9)/3 = 4.33$	$\Rightarrow$	$g(8) = \text{round}\{4.33\}$	$=$	4
W_9 :	$(2, 9, 9)$	$\Rightarrow$	$(2+9+9)/3 = 6.67$	$\Rightarrow$	$g(9) = \text{round}\{6.67\}$	$=$	7

g | 5 | 5 | 22 | 20 | 21 | 6 | 5 | 3 | 4 | 7 |

For $g(0)$ and $g(9)$, $f(0)$ and $f(9)$, respectively, are extended outside the boundaries.

Comparing 1D Median and Mean Filtering



What are the differences in the way the filters have modified the original signal?

2D 3×3 Linear Mean Filter $g(x,y) = \frac{1}{9} \sum_{\xi=-1}^{1} \sum_{\eta=-1}^{1} f(x+\xi, y+\eta)$

Large "salt-and-pepper" noise.

Mean filtering result.

Mean filter $\frac{1}{9}$

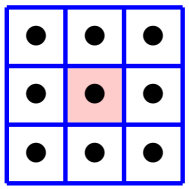
1	1	1
1	1	1
1	1	1

What are the differences in the result compared with the median filter?

2D 3×3 Nonlinear Filter $g(x,y) = \underset{\xi,\eta=-1,-1}{\overset{1,1}{\text{median}}} \{f(x+\xi, y+\eta)\}$

Large "salt-and-pepper" noise.

Median filtering result.

Median filter 

What are the differences in the result compared with the mean filter?

2D Mean Filtering: 3×3 Window

1: Keeping border values unchanged.

Averaging: $\text{round}\{(1+4+0+1+2+4+1+0+1)/9\} = 2$

1	4	0	1	3	1
2	(2)	4	2	2	3
1	0	1	0	1	0
1	2	1	0	2	2
2	5	3	1	2	5
1	1	4	2	3	0

Input $\mathbf{f}$

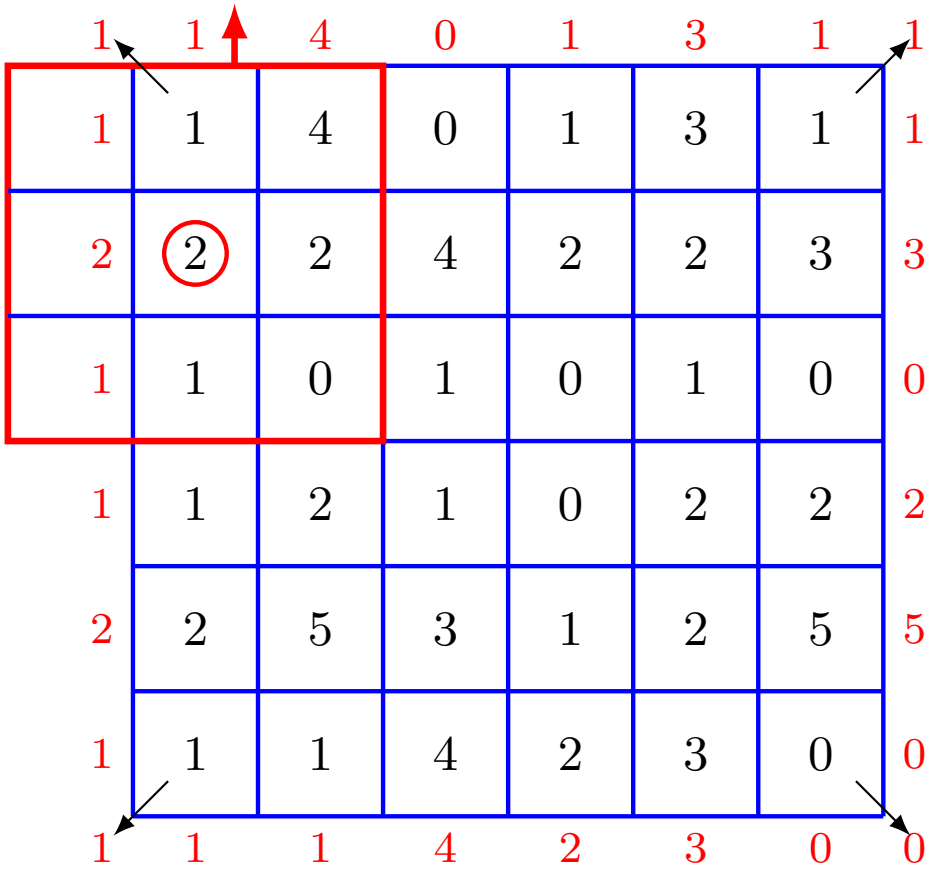
1	4	0	1	3	1
2	(2)	2	2	1	3
1	2	1	1	1	0
1	2	1	1	1	2
2	2	2	2	2	5
1	1	4	2	3	0

Output $\mathbf{g}$

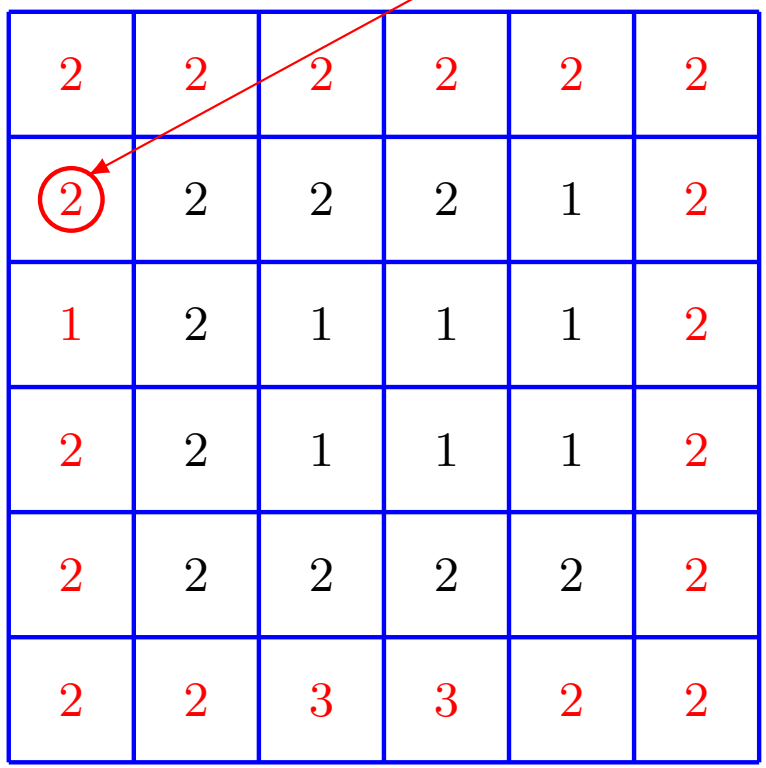
2D Mean Filtering: 3×3 Window

2: Extending border values outside with the boundary values.

Averaging: $\text{round}\{(1+1+4+2+2+2+1+1+0)/9\} = 2$



Input f

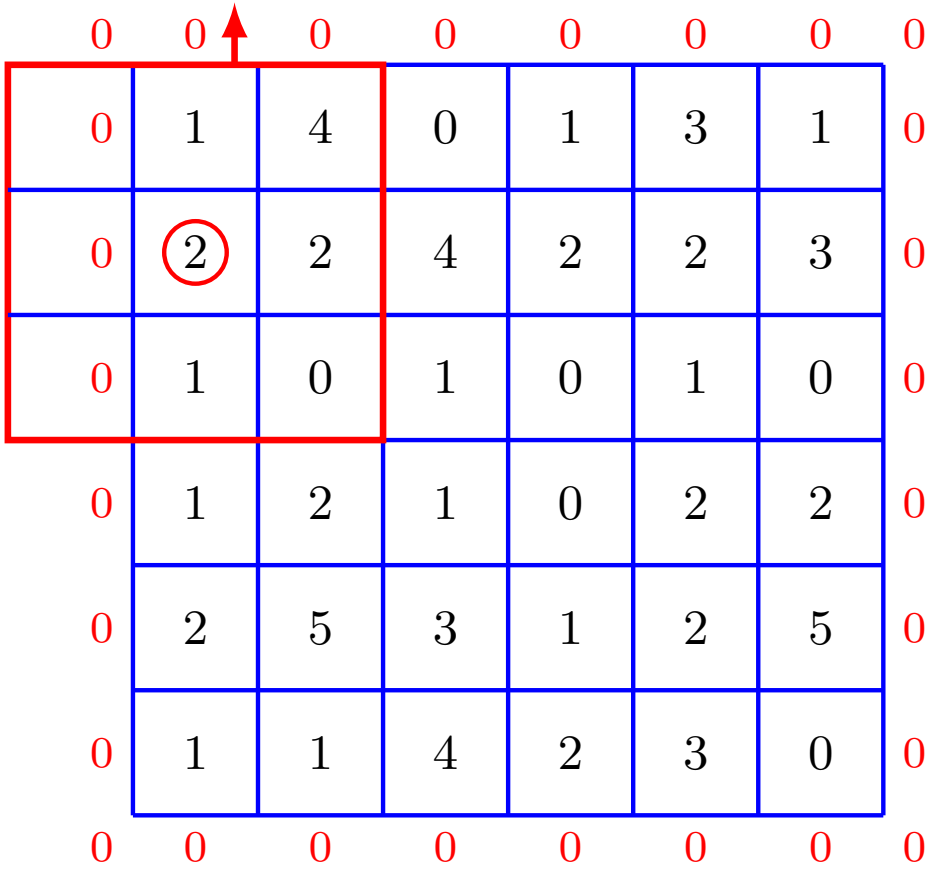


Output g

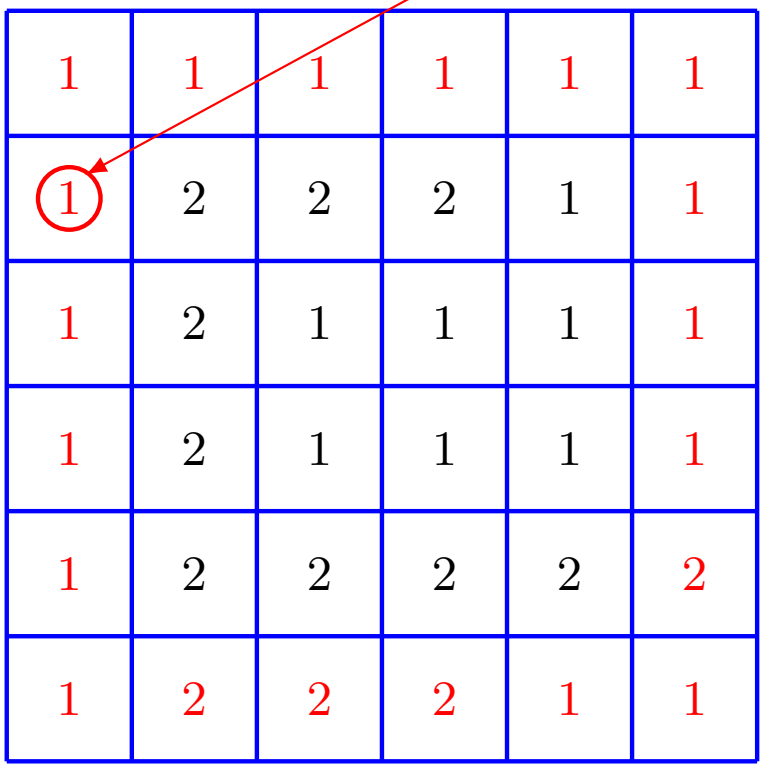
2D Mean Filtering: 3×3 Window

3: Extending border values outside with zeros (zero padding).

Averaging: $\text{round}\{(0+1+4+0+2+2+0+1+0)/9 = 1$



Input f



Output g

Median Vs. Mean Filtering: More Examples



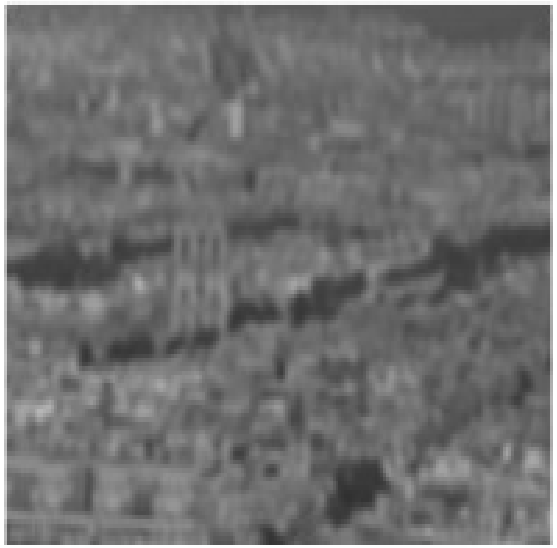
Median filter:

- Suppressing outliers (outstanding values).
- Destroying local transitions (intensity changes).

Mean filter:

- Averaging out all.
- Blurring without destroying local transitions.

Median Vs. Mean Filtering: More Examples


Median filter:

- Suppressing outliers (outstanding values).
- Destroying local transitions (intensity changes).

Mean filter:

- Averaging out all.
- Blurring without destroying local transitions.

Median Vs. Mean Filtering: More Examples

Scenario: keeping the border values unchanged.

50	50	50	50	50	50
50	50	50	50	150	50
50	150	50	150	50	50
50	50	50	50	50	50
50	150	50	150	50	50
50	50	50	50	50	50

Initial image

50	50	50	50	50	50
50	50	50	50	50	50
50	50	50	50	50	50
50	50	50	50	50	50
50	50	50	50	50	50
50	50	50	50	50	50

3×3 median filter

50	50	50	50	50	50
50	61	72	72	72	50
50	61	72	72	72	50
50	72	94	72	72	50
50	61	72	61	61	50
50	50	50	50	50	50

3×3 mean filter

50	50	50	150	50	50
50	50	50	150	50	50
50	150	150	150	50	50
50	50	50	150	50	50
50	50	50	150	50	50
50	50	50	150	50	50

50	50	50	150	50	50
50	50	150	50	50	50
50	50	150	50	50	50
50	50	150	50	50	50
50	50	50	50	50	50
50	50	50	150	50	50

50	50	50	150	50	50
50	72	106	94	83	50
50	72	106	94	83	50
50	72	106	94	83	50
50	50	83	83	83	50
50	50	50	150	50	50

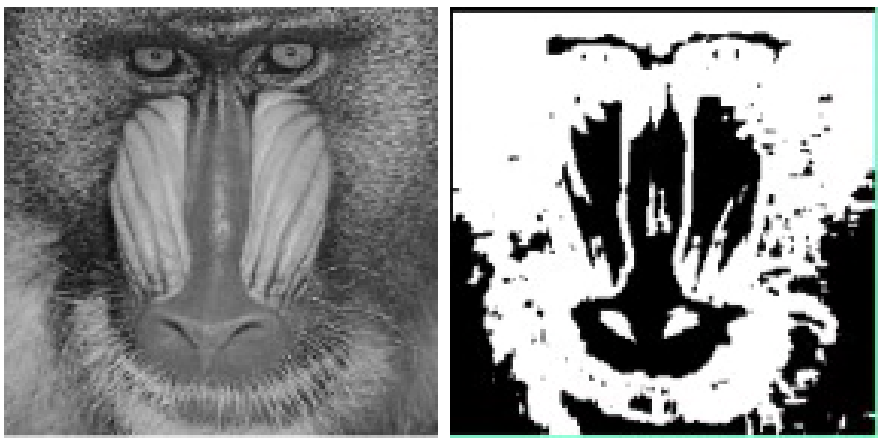
Segmentation Problem

Partitioning an image into distinct regions with similar attributes.

- Meaningful regions relate to objects or features of interest.
  - *Meaningful segmentation*: the first step from low-level image processing (IP) to high-level image analysis (IA).
    - IP: transforming an image into one or more other images.
    - IA: describing depicted scenes in terms of features and objects.
- Success of image analysis depends on reliable segmentation.
- Accurate partitioning is generally a very challenging problem!

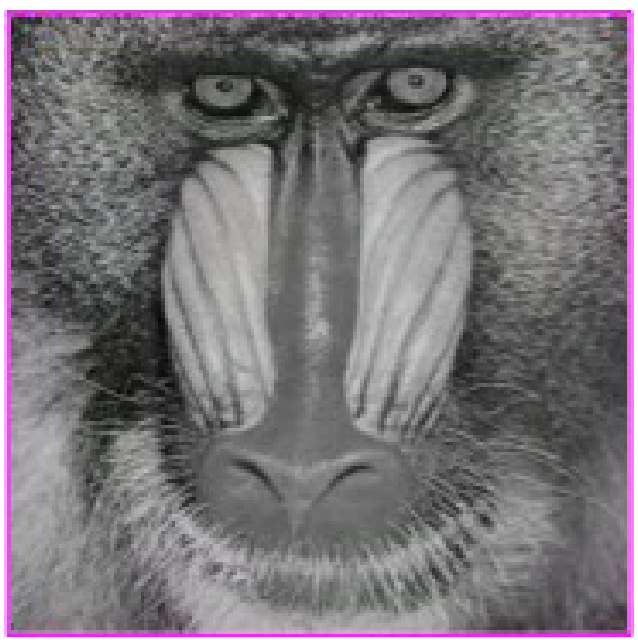
Types of segmentation:

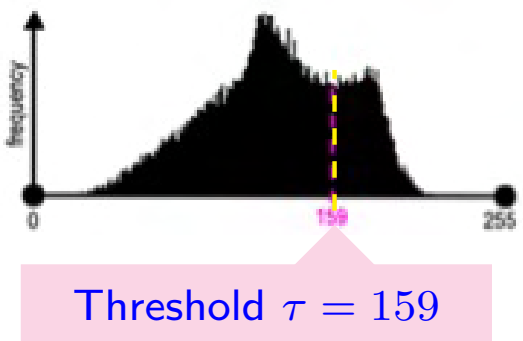
- **Non-contextual**: $\Rightarrow$ grouping pixels with similar global features.
- **Contextual**: grouping pixels with similar features and in close locations.



Non-contextual Thresholding

- The simplest non-contextual technique.
- Single threshold τ: converting a grayscale image g into a binary region map m: $m(x,y) = \begin{cases} 0 & \text{if} \quad g(x,y) < \tau \\ 1 & \text{if} \quad g(x,y) \geq \tau \end{cases}$





Threshold $\tau = 159$

Region 0 – black
Region 1 – white

g

m

Simple Thresholding

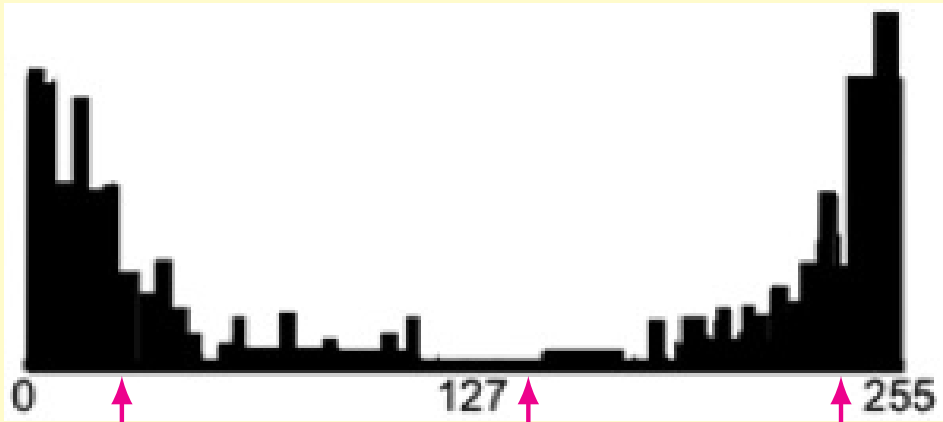
A single threshold produces the binary map with two possibly disjoint regions:

- Region 1 (label 0) with pixel values smaller than a threshold.
- Region 2 (label 1) with the values at or above the threshold.
- The regions depend on the image feature, being compared to a threshold, and on how the threshold is chosen.

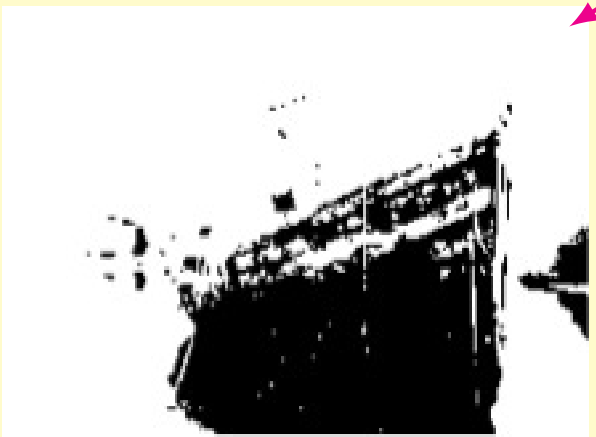
Generally, two or more thresholds can produce more than two types of regions:

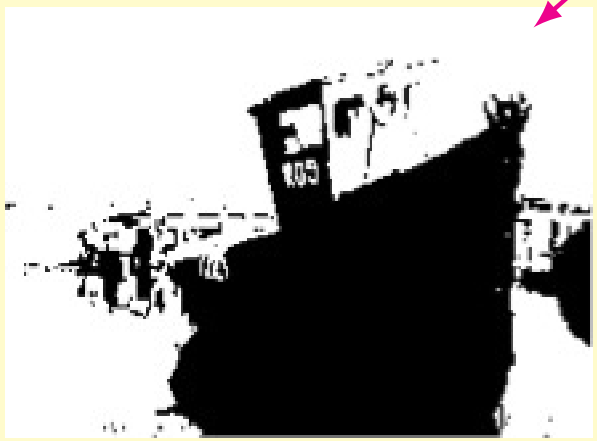
- The thresholds separate ranges of values associated with the region type.
- In principle, one region may combine several ranges of values.
  - E.g. $m(x,y) = \begin{cases} 0 & \text{if} \quad g(x,y) < \tau_1 \text{ OR } g(x,y) > \tau_2 \\ 1 & \text{if} \quad \tau_1 \leq g(x,y) \leq \tau_2 \end{cases}$

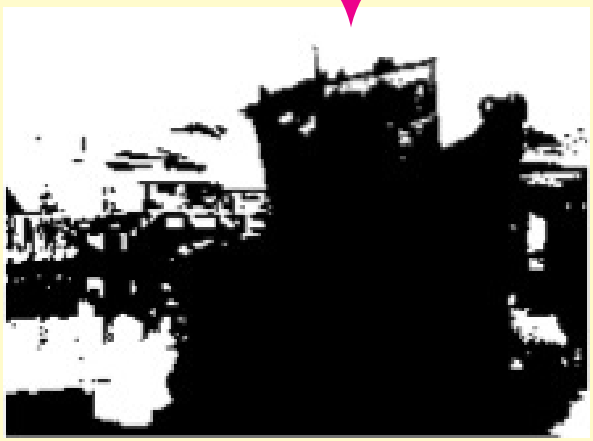
Simple Thresholding: Examples



τ=26 τ=133 τ=235







Simple Thresholding

Main problems of simple thresholding:

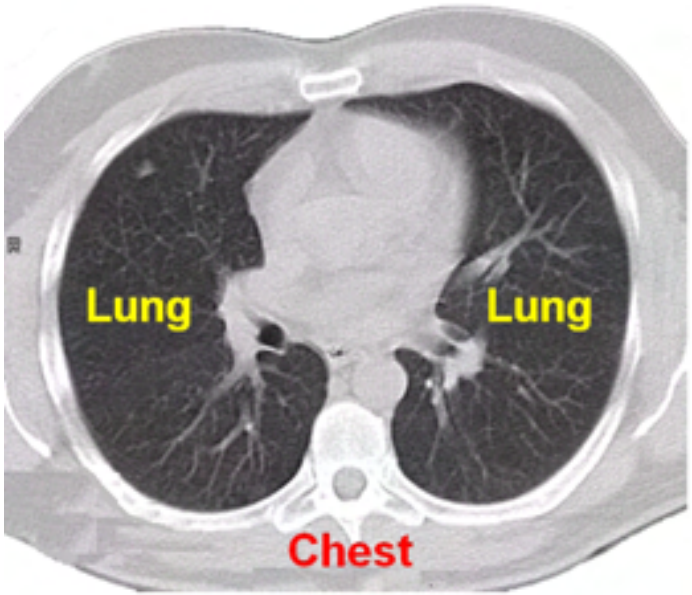
- Whether it is possible and, if yes,
- How to choose an adequate threshold or a number of thresholds to separate one or more desired objects from their background.

In many practical cases the simple thresholding is unable to segment objects of interest.

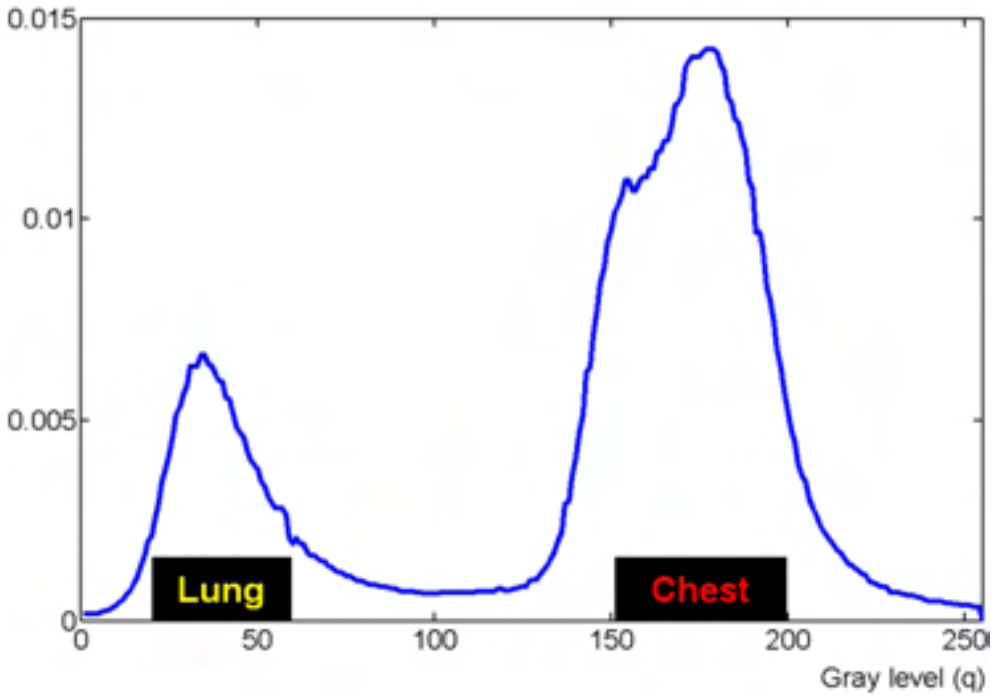
General approach – images are assumed being multimodal:

- Different objects of interest relate to distinct peaks, or modes of the 1D empirical signal histogram.
- Thresholds to optimally separate objects in spite of typical overlaps between signal ranges corresponding to individual peaks.

Multimodal Images: Medical Imaging Example

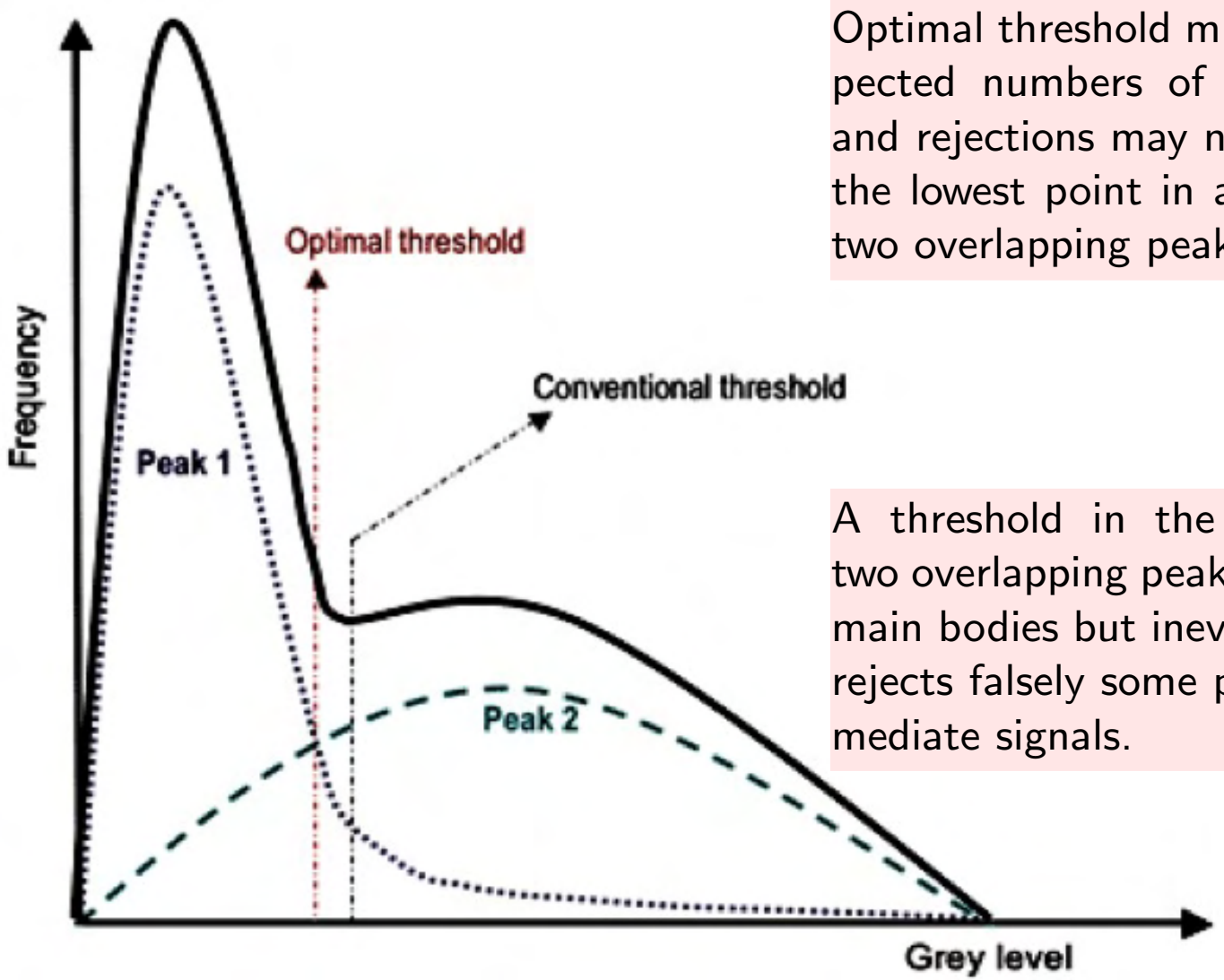


Chest slice



Histogram of occurrences of gray levels for the whole chest image

Simple Thresholding: Mixture Separation



Optimal threshold minimising the expected numbers of false detections and rejections may not coincide with the lowest point in a valley between two overlapping peaks.

A threshold in the valley between two overlapping peaks separates their main bodies but inevitably detects or rejects falsely some pixels with intermediate signals.

Adaptive Thresholding

Threshold separating a background from an object has to equalise two kinds of expected errors:

- **False alarm**: assigning a background pixel to the object;
- **Missing object**: assigning an object pixel to the background.

Adaptive non-contextual separation accounts for empirical probability distributions of object (e.g. dark) and background (bright) pixels.

- More complex adaptation: a spatially variant threshold to account for local context ("background normalisation).

Adaptive Thresholding

Simple iterative adaptation of the threshold:

- Successive refinement of the estimated peak positions.

Basic assumption – centre-symmetric distributions of object and background grey levels:

1. Each peak coincides with the mean grey level for all pixels that relate to that peak.
2. Pixel probability decreases monotonically with the growing absolute difference between the pixel grey level and the peak grey level both for an object and background.
3. Each grey level is associated with the relevant peak by thresholding.
    - The threshold is placed half-way between the peaks.

Threshold Adaptation at Each Iteration j

1. Classify each grey level $g(x, y)$ with a threshold θ_j, having been computed at the previous iteration:

$$\text{pixel } (x, y) \in \begin{cases} C_{\text{ob:}[j]} & \text{if} \quad g(x, y) \leq \theta_j \\ C_{\text{bg:}[j]} & \text{if} \quad g(x, y) > \theta_j \end{cases}$$

   where $C_{\text{ob:}[j]}$ and $C_{\text{bg:}[j]}$ are the object and background regions at iteration j, respectively, in the image g.

2. Compute mean grey values for each class:

$$\mu_{\text{ob:}[j]} = \frac{1}{|C_{\text{ob:}[j]}|} \sum_{(x,y) \in C_{\text{ob:}[j]}} g(x, y); \quad \mu_{\text{bg:}[j]} = \frac{1}{|C_{\text{bg:}[j]}|} \sum_{(x,y) \in C_{\text{bg:}[j]}} g(x, y)$$

   where $|C|$ denotes the number of pixels in the region C.

3. Compute the new threshold $\theta_{j+1} = \frac{1}{2} \left(\mu_{\text{ob:}[j]} + \mu_{\text{bg:}[j]} \right)$.

Adaptive Thresholding: Only Image Histogram is Needed

Input: an image histogram $\mathbf{H} = (H(q) : q = 0, \ldots, 255)$

Initialisation: $j = 0;\ N = \sum_{q=0}^{255} H(q);\ \theta_0 = \frac{1}{N} \sum_{q=0}^{255} qH(q)$

while $\theta_{j+1} \neq \theta_j$ **do**

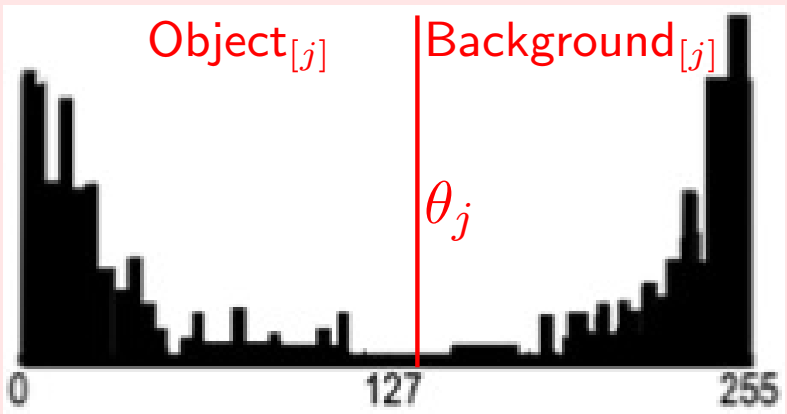
$$N_{\text{ob:}[j]} = \sum_{q=0}^{\theta_j} H(q); \quad N_{\text{bg:}[j]} = \sum_{q=\theta_j+1}^{255} H(q)$$

$$\mu_{\text{ob:}[j]} = \frac{1}{N_{\text{ob:}[j]}} \sum_{q=0}^{\theta_j} qH(q)$$

$$\mu_{\text{bg:}[j]} = \frac{1}{N_{\text{bg:}[j]}} \sum_{q=\theta_j+1}^{255} qH(q)$$

$$\theta_{j+1} = \frac{1}{2}\left(\mu_{\text{ob:}[j]} + \mu_{\text{bg:}[j]}\right)$$

end while



Colour Thresholding (the image example from http://www.matrix-vision.com/products/software)

Colour segmentation is more accurate due to more information per pixel.

RGB colour space: interdependent components.

HSI (HSV) colour space: more stable and illumination-independent segmentation.

Segmentation by partitioning of the colour space.

- Thresholding distances from pixel values ($R(x,y)$, $G(x,y)$, $B(x,y)$) to a reference colour ($R_0G_0B_0$):

$$d(x,y) = \sqrt{(R(x,y)-R_0)^2 + (G(x,y)-G_0)^2 + (B(x,y)-B_0)^2}$$

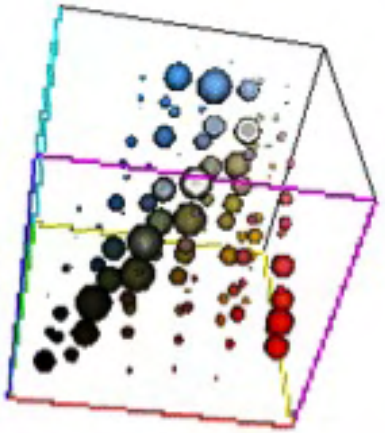
$$\Rightarrow\ m(x,y) = \begin{cases} 1 & \text{if} \quad d(x,y) \le d_{\max} \\ 0 & \text{if} \quad d(x,y) > d_{\max} \end{cases}$$

Thresholding with Colour Histogram (3D or 2D Projection)

Partitioning the colour space into bins (each bin - similar colours):

$6 \times 6 \times 6$ bins

- Dominant colours corresponding to peaks in the histogram.
- Pre-selected colours, e.g. the primary Red, Yellow, Green, Cyan, Blue, Magenta, White, and blacK.



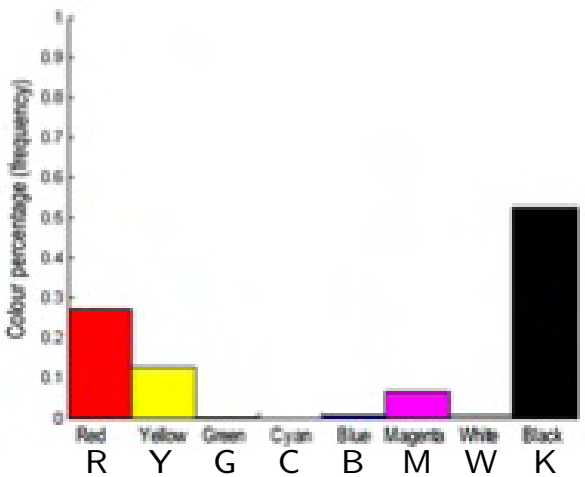


Image Segmentation: Basic Approaches

Non-contextual thresholding:

- Grouping pixels with no account of locations in the lattice.

Contextual segmentation:

- Can be more successful in separating individual objects.
- Accounts for close locations of pixels belonging to an object.
- Exploit two signal properties: **discontinuity** or **similarity**.

Discontinuity-based segmentation (finding boundaries):

- Goal: to build complete boundaries of uniform regions.
- Assumption: abrupt signal changes across each boundary.

Similarity-based segmentation (finding uniform regions):

- Goal: to group connected pixels that satisfy similarity criteria.

Both the approaches mirror each other, in the sense that a complete boundary splits one region into two.

Pixel Neighbourhood in an Arithmetic Rectangular Lattice

Normal sampling: a digital image on a finite arithmetic lattice:

$$\{(x, y) : \ x = 0, 1, \ldots, X-1; \ y = 0, 1, \ldots, Y-1\}$$

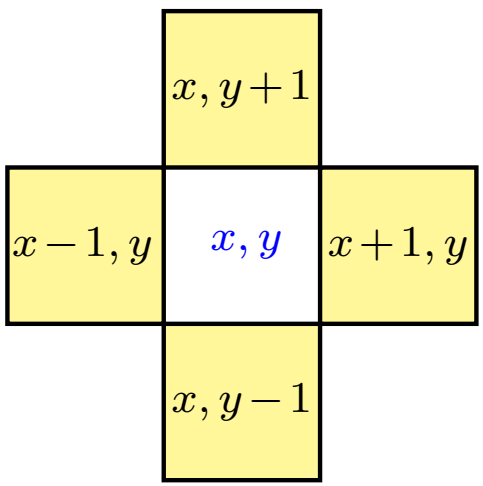
Two types of the nearest neighbourhood of a pixel (x, y):

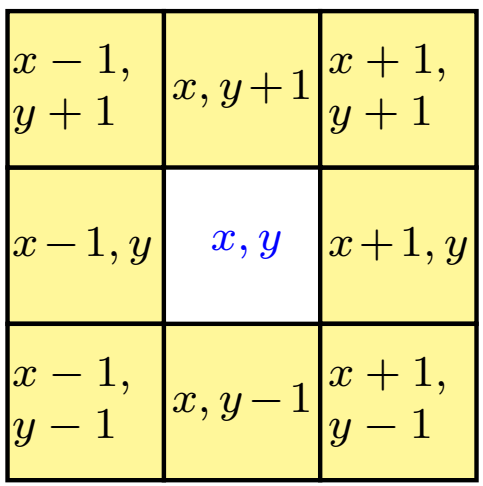
4-neigbourhood:

$$\{(x, y \pm 1), \ (x \pm 1, y)\}$$

8-neighbourhood:

$$\{(x-1, y \pm 1), \ (x, \pm 1), \ (x+1, y \pm 1), \ (x \pm 1, y)\}$$

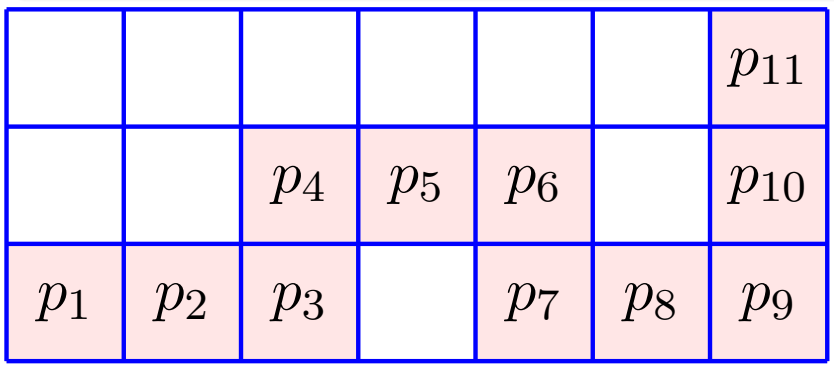




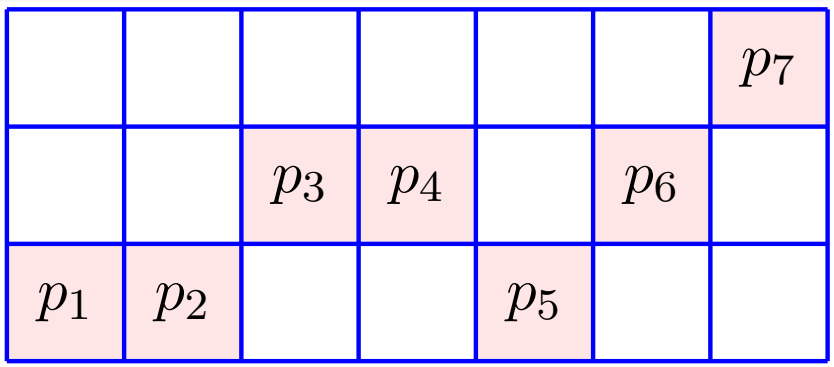
Pixel Connectivity

A 4- or 8-connected path from pixel p_1 to another pixel p_n

is a sequence of pixels $\{p_1, p_2, \ldots, p_n\}$, such that p_{i+1} is a 4- or 8-neighbour, respectively, of p_i for all $i = 1, \ldots, n-1$.



4-connected path

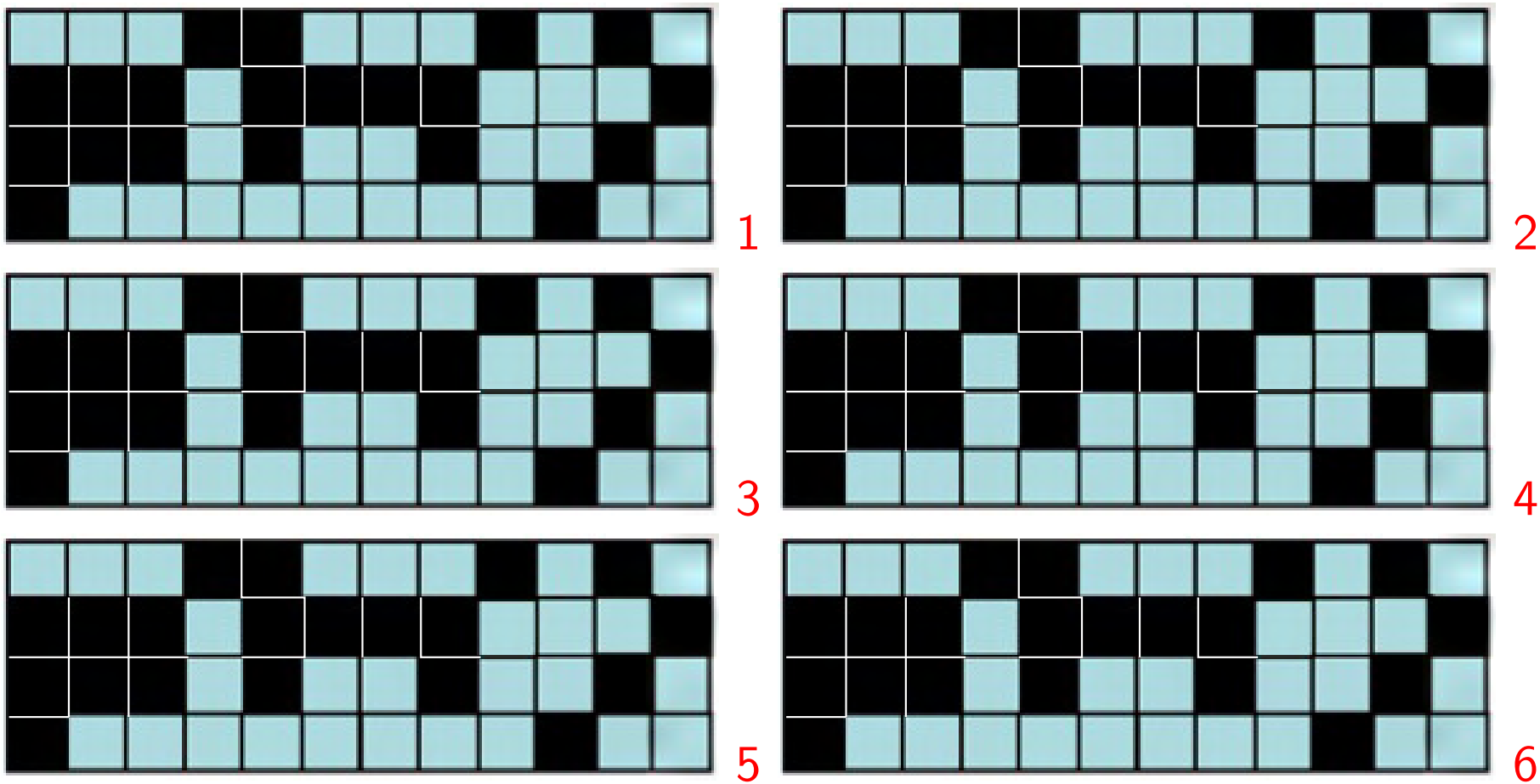


8-connected path

- A set of pixels is a **4-connected region** if there exists at least one 4-connected path between any pair of pixels from that set.
- The **8-connected region** has at least one 8-connected path between any pair of pixels from that set.

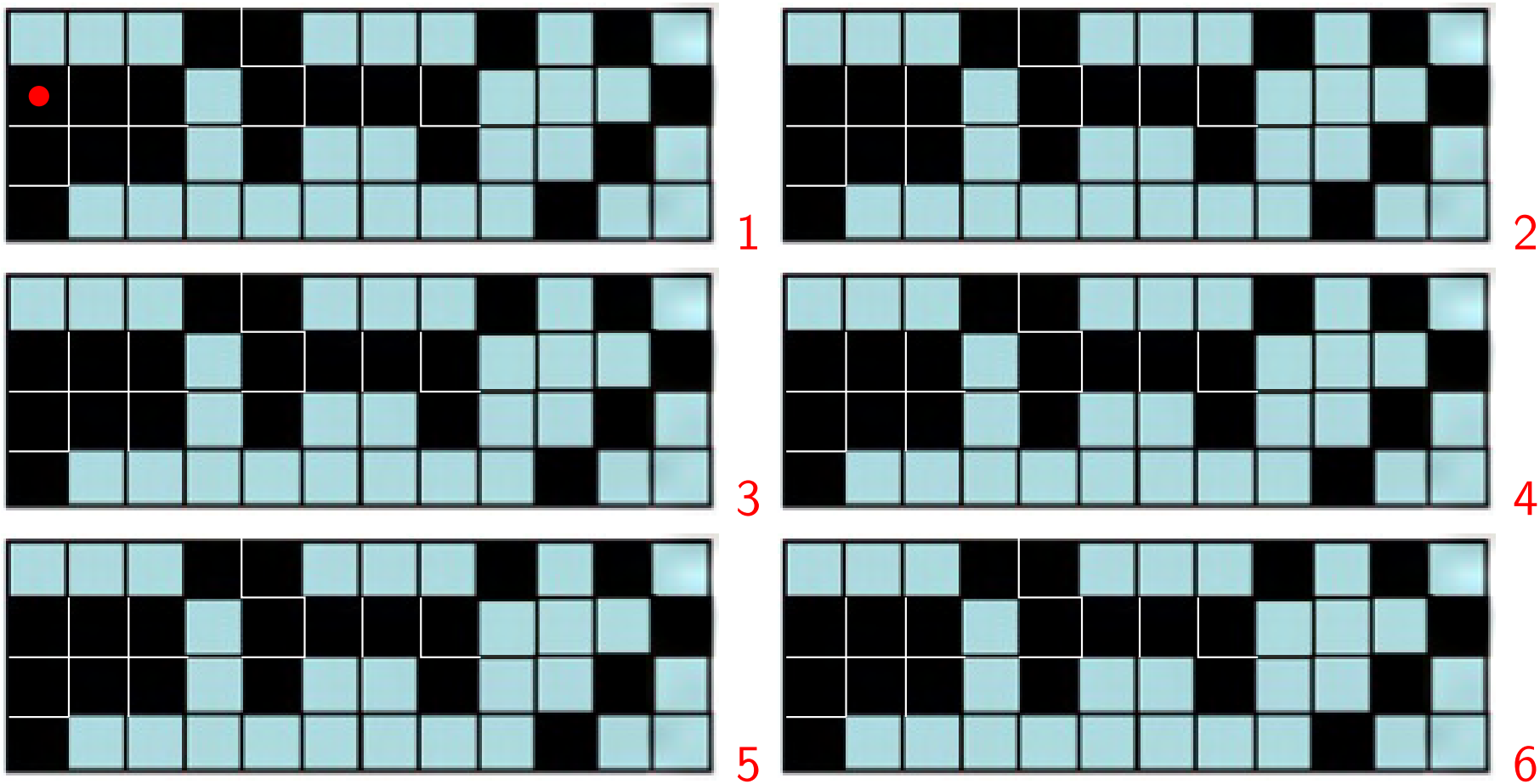
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:



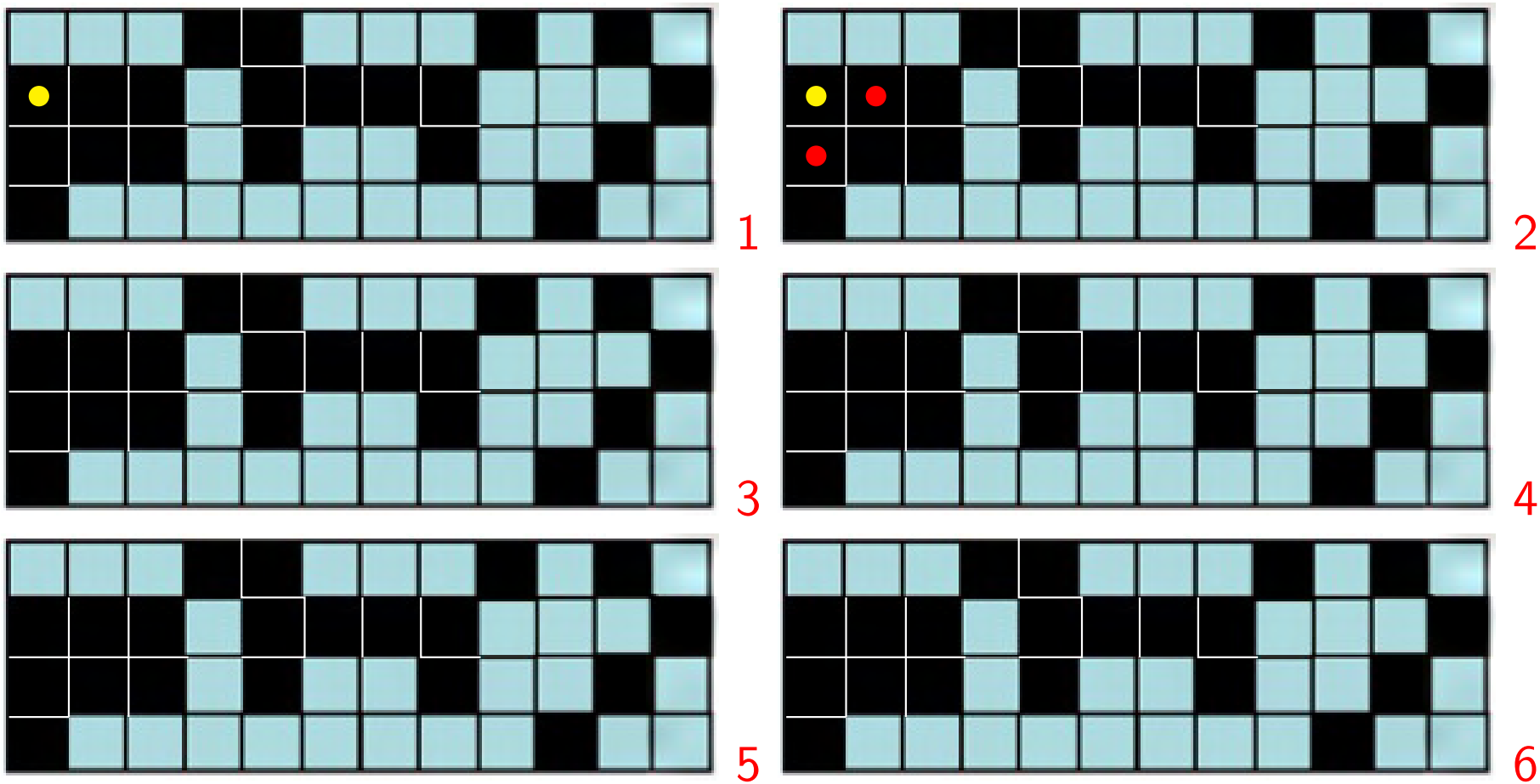
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:



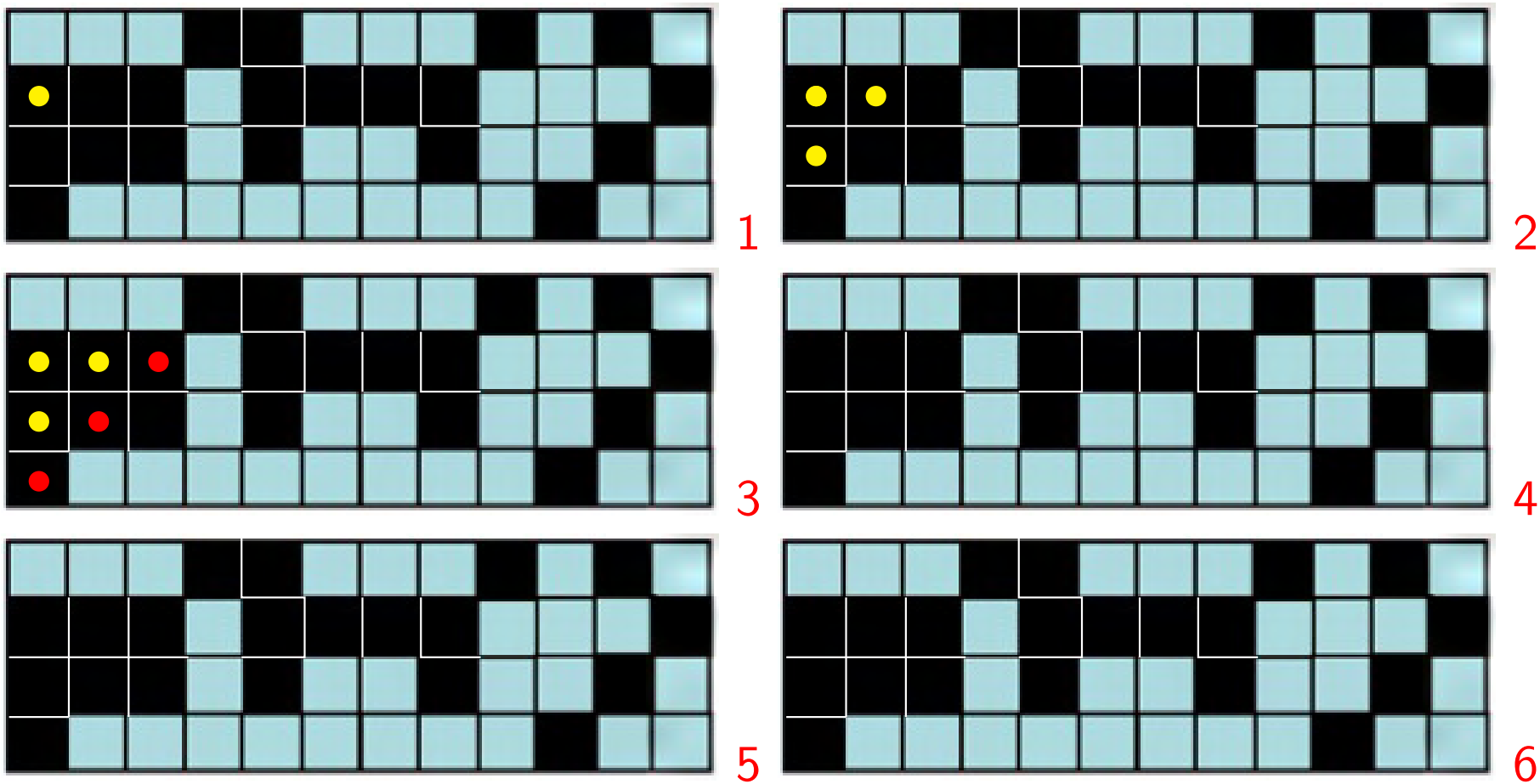
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:



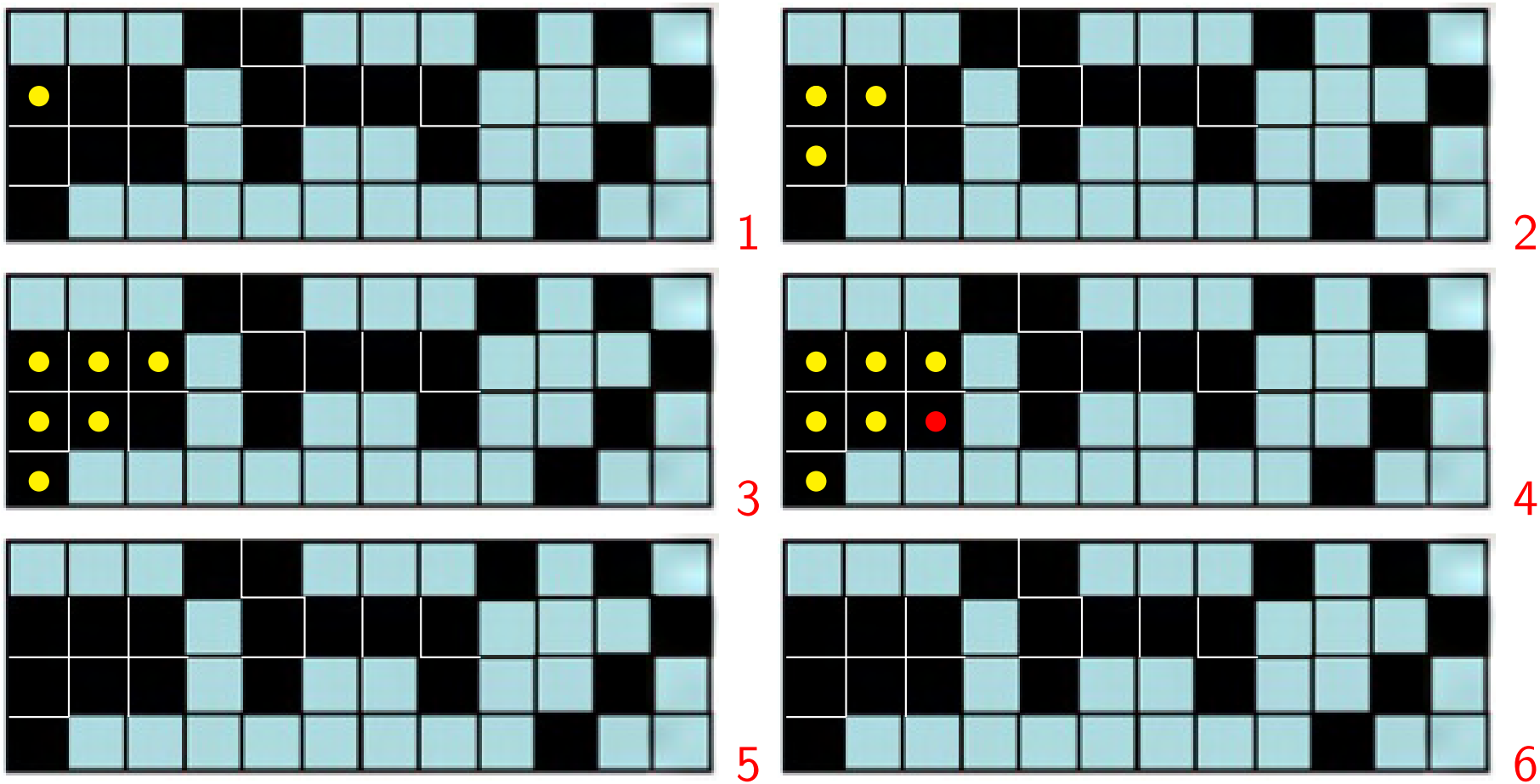
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:



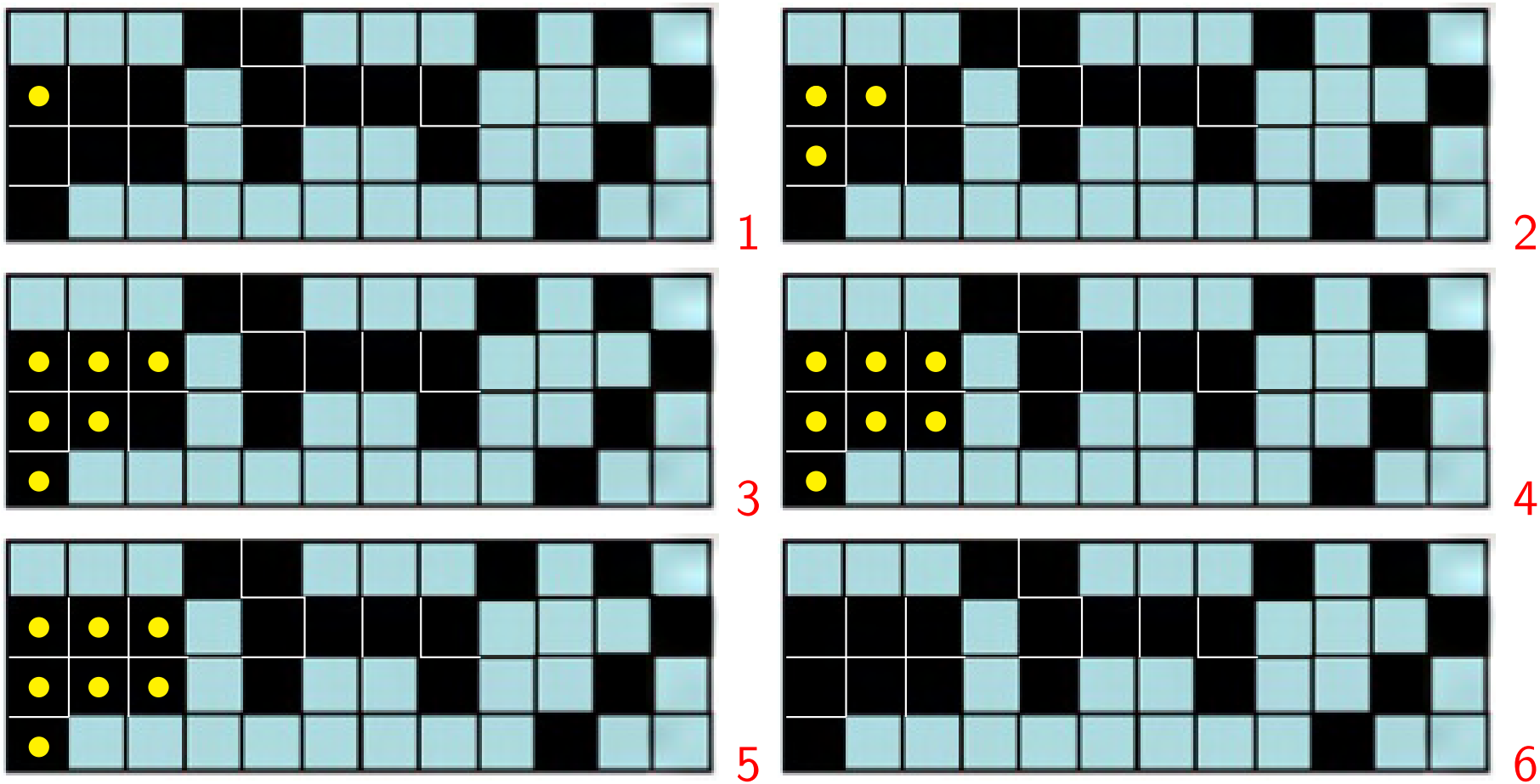
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:



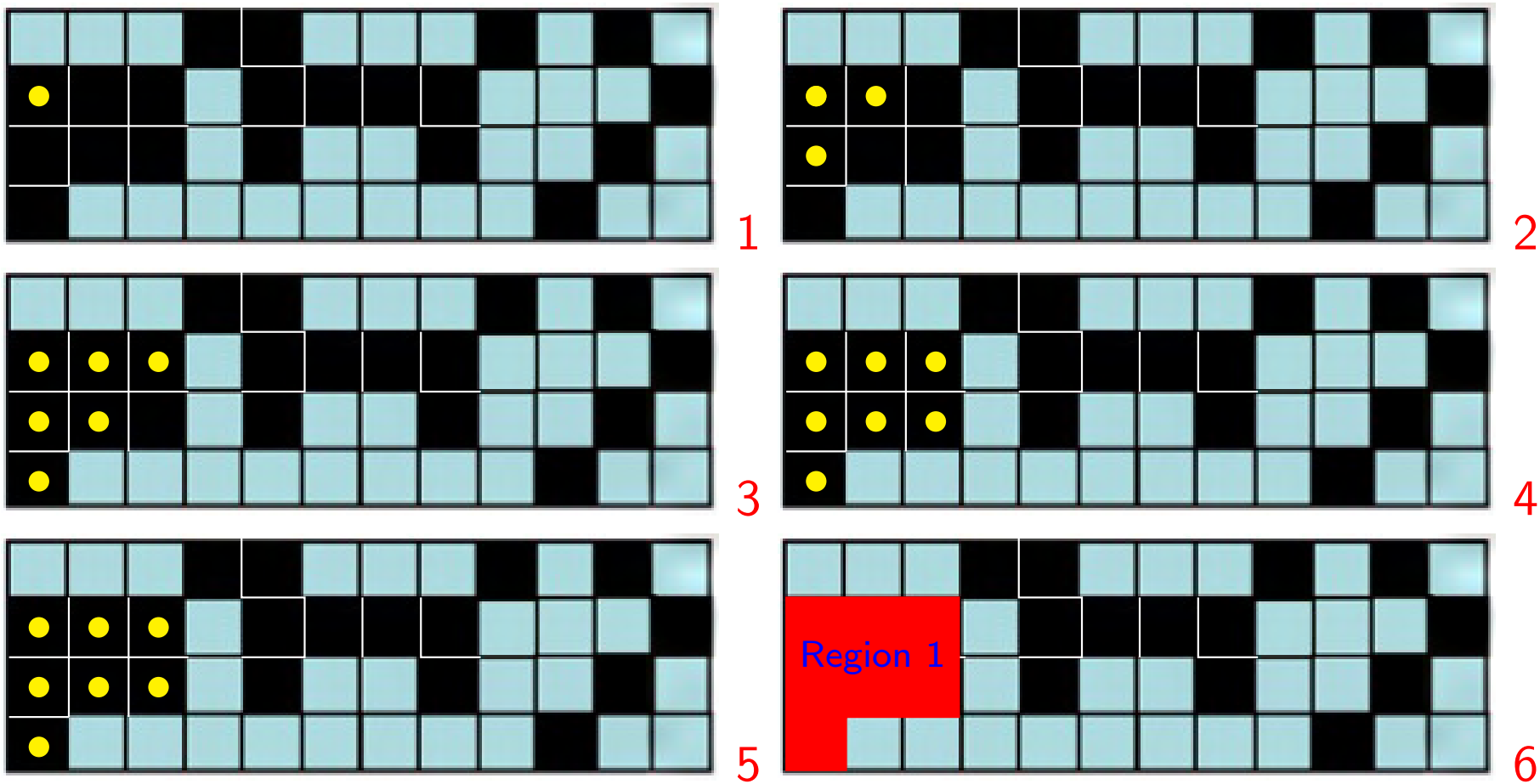
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:



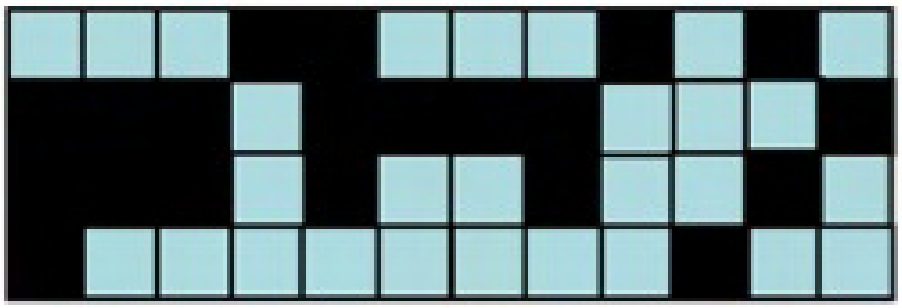
Connected Region Labelling

The simplest "grassfire (or wave) propagation" algorithm:

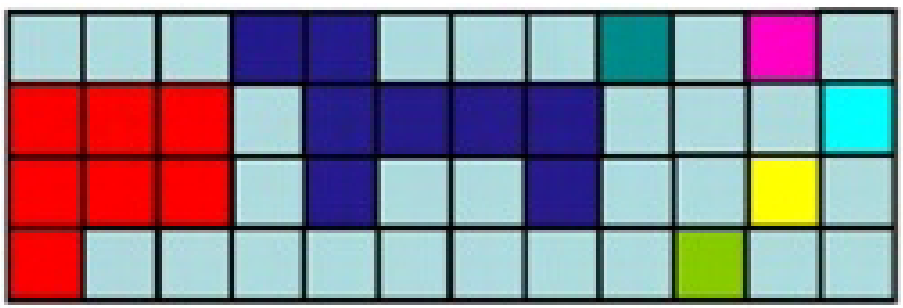


4- Vs. 8-Connected Regions

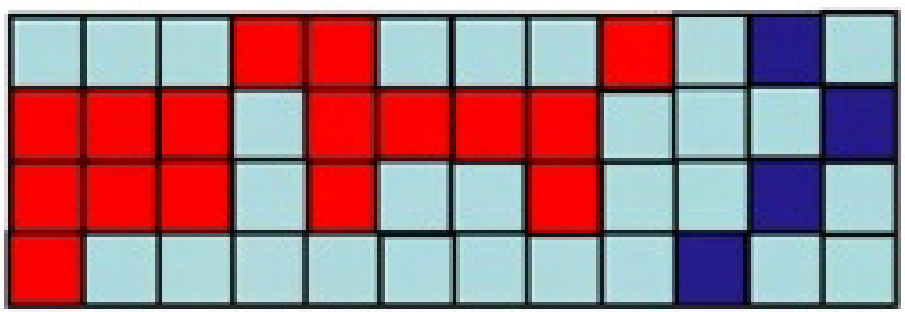
The 4- and 8-connectivity produce different segmentation maps:



Binary image:
0 (black) objects;
1 (white) background.



4-connected objects;
8-connected background.



8-connected objects;
8-connected background.

4- Vs. 8-Connected Regions

Binary image 64 x 64 (zoom $\times 5$): white objects on black background.

86 4-connected object regions

10 8-connected object regions

http://www.dca.fee.unicamp.br/projects/khoros/mmach/tutor/toolbox/basicl/labeling/front-page.html

Region Similarity

Uniformity / non-uniformity of pixels in a connected region is represented by a **uniformity predicate**.

- Logical statement, or condition being **true** if pixels in the regions are similar with respect to some property.
- Pixel properties: colour, grey level, edge strength, local texture pattern, etc.

Common simple local predicate $P(R)$

- Restricted signal variations over a pixel neighbourhood in a connected region R:

$$P(R) = \begin{cases} \mathbf{TRUE} & \text{if} \quad |g(x,y) - g(x+\xi, y+\eta)| \leq \delta \\ \mathbf{FALSE} & \text{otherwise} \end{cases}$$

where (x, y) and $(x + \xi, y + \eta$ are the lattice coordinates of the neighbouring pixels in R.

Region Similarity

The simple local predicate in Slide 45 does not restrict the variation of grey levels within an entire region.

- Small changes in the neighbouring signal values can accumulate over the region.

Intra-region signal variations can be restricted with a similar, but non-local predicate:

$$P(R) = \begin{cases} \textbf{TRUE} & \text{if} \quad |g(x,y) - \mu_R| \leq \varepsilon \\ \textbf{FALSE} & \text{otherwise} \end{cases}$$

where

- ε is a fixed signal similarity threshold;
- (x, y) are the lattice coordinates of a pixel from the region R, and
- μ_R is the mean value of signals $g(x, y)$ over the entire region R.

Region Growing Segmentation: The Bottom-up Algorithm

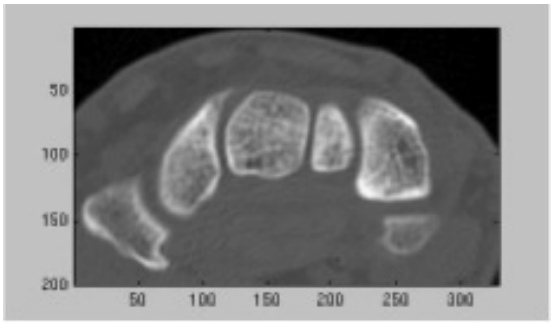
- **Initialisation**: a set of seed pixels defined by the user.
- **Region growth**: sequentially add a pixel to a region under the following conditions:
  1. The pixel has not been assigned to any other region.
  2. The pixel is a neighbour of that region.
  3. Addition of the pixel does not impact the uniformity of the growing region.

Region growing is simple but unstable:

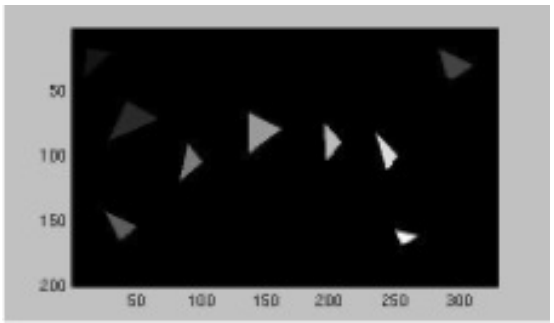
- It is very sensitive to a chosen uniformity predicate: small changes of the uniformity threshold may result in large changes of the regions found.
- Very different segmentation maps under different routes of image scanning, modes of exhausting neighbours of each growing region, seeds, and types of pixel connectivity.

Region Growing Segmentation: A Few Examples

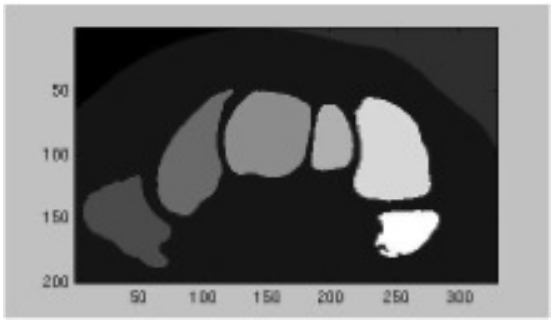
http://www.lems.brown.edu/ msj/cs292/assign5/segment.html



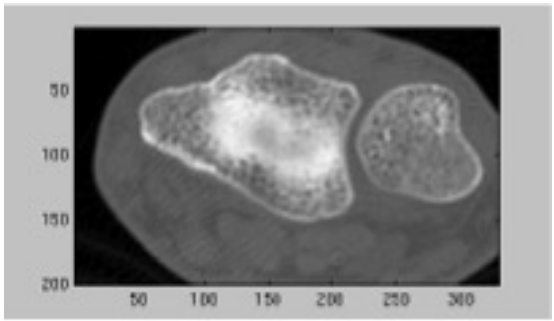
Greyscale image

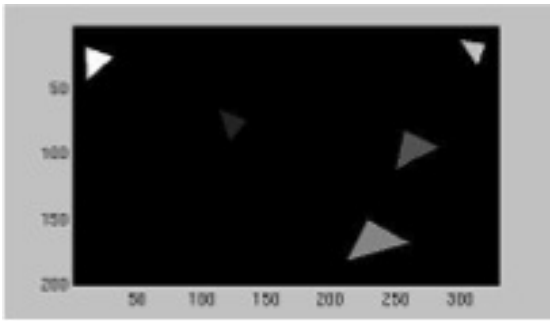


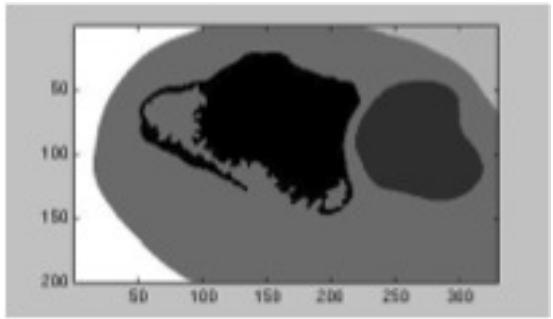
Seed regions

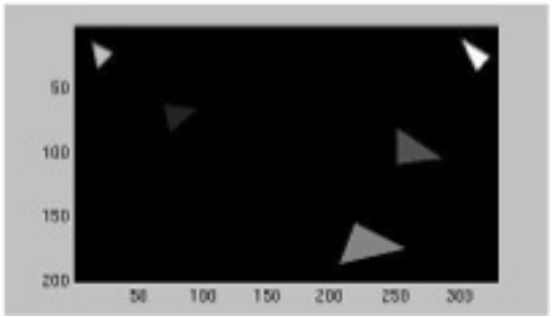


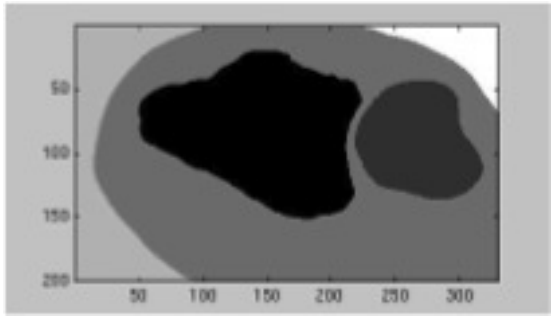
Region growing results







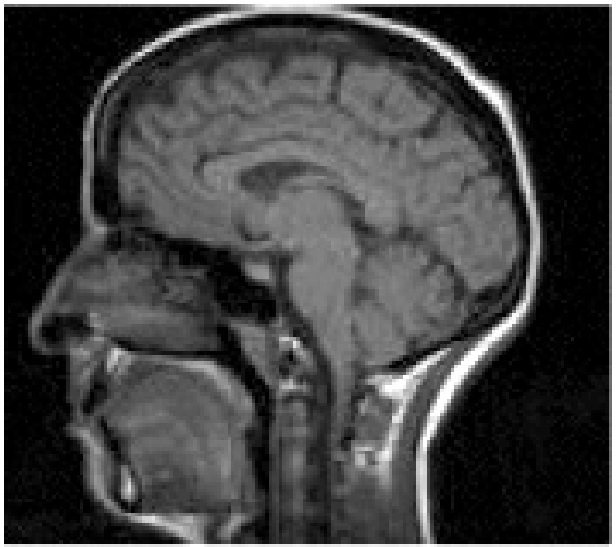




Region growing from two variants of seed regions.

Growth of the 4- and 8-Connected Regions: An Example

http://www.comp.leeds.ac.uk/ai21/examples/images/rgrow.html



Greyscale image

4-conn.

8-conn.

Basic Segmentation Criteria

1. All the pixels have to be assigned to regions.
2. Each pixel can belong to a single region only.
3. Each region is a connected set of pixels.
4. Each region is uniform w.r.t. a given uniformity predicate.
5. Merging two adjacent regions gives a non-uniform region.

- **Region growing:** Criteria 1 and 2 are not satisfied.
  - In general, the number of seeds may not be sufficient to create a region for every pixel.
- **Region growing:** Criteria 3 and 4 are satisfied.
  - Each region is connected and uniform.
- **Region growing:** Criterion 5 may not hold.
  - Regions grown from two nearby seeds within a potentially uniform part of the image are always regarded as distinct.

Split-and-Merge Segmentation: The Top-Down Algorithm

- **Initialisation:** the entire image is a single region.
- **Iterative segmentation:**
  - **Splitting stage:** Split each region into sub-regions.
  - **Merging stage:** Merge adjacent regions if the resulting larger region remains uniform.
- **Stopping rule:**
  - All regions become uniform **OR**
  - The desired number of regions have been established.

Common splitting strategy for a square image:

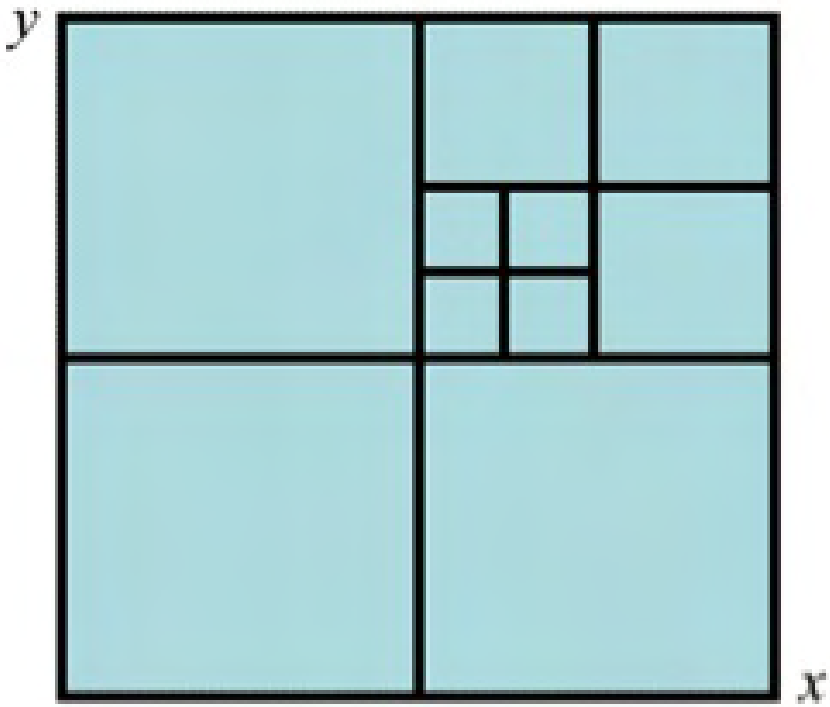
- Divide it recursively into smaller and smaller quadrants until, for any region R, the uniformity predicate $P(R)$ is **TRUE**.

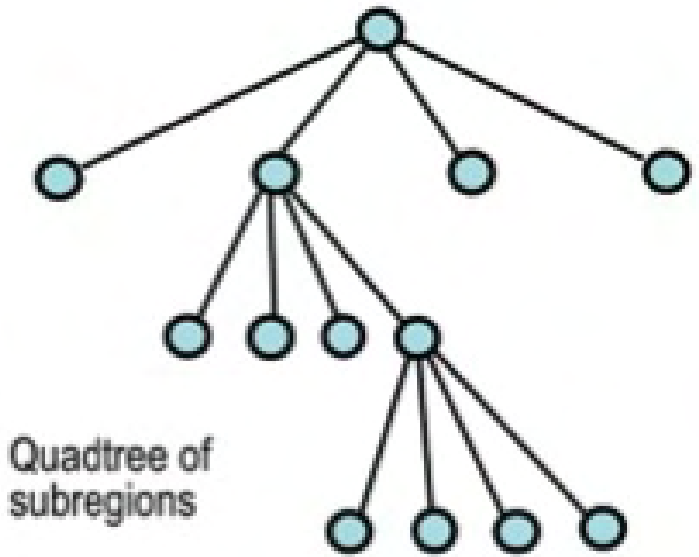
The strategy builds a top-down quadrant tree (*quadtree*):

- if $P(\text{image})$ is **FALSE**, divide the image into 4 quadrants.
- if $P(\text{quadrant})$ is **FALSE**, divide the quadrant into 4 sub-quadrants.
- Etc...

Split-and-Merge Segmentation

The splitting stage alternates with the merging stage.





The merging stage: two adjacent regions R_i and R_j are combined into a new, larger region if the uniformity predicate for the union of these two regions, $P(R_i \bigcup R_j)$, is **TRUE**.

[Optional Example] Colour Segmentation for Skin Detection

Region-of-interest in a training image – building a look-up table of skin colours:

- **Drawback:** incorrect classification of skin pixels absent in the training sample and background pixels.

Colour segmentation:

- **Homogeneity criteria:** e.g., the Euclidean distance between a current grey value or colour and its mean (average) value in the region.
- Region growing techniques:
  - **Split-and-merge**: an iterative division of an image into homogeneous regions.
  - **Seeding**: staring from a few seeds, a growth into large homogeneous regions.

Region Growing by Seeding: Colour Predicate

1. **Select a set of seed points.**
2. **Check** each neighbouring pixel (8- or 4-connected) whether it is similar to the seed highly enough to merge.
3. **Add** similar neighbours to the growing region.
4. **Iteratively test** the neighbours of the pixels at the region boundary for similarity with their neighbours.

Colour predicate from many training images by automatic segmentation of each image into skin and background regions:

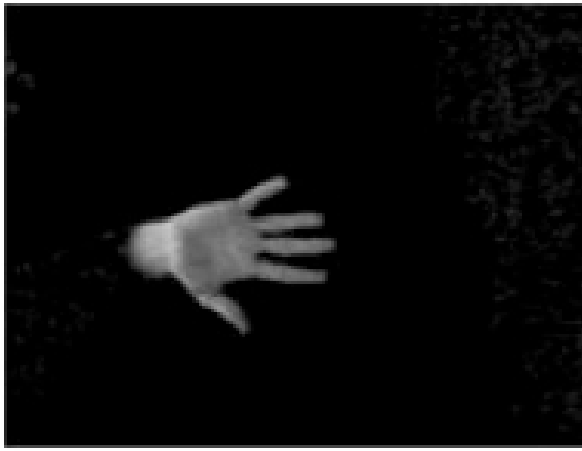
- Thresholding the hue images.
- **Logarithmic hue** values: only from red–green (R–G) colour components because red prevails in skin.
- The ratio G/R is robust to intensity changes:

$$h = \lfloor 256 \frac{\mathsf{G}}{\mathsf{R}} \rfloor \text{ if } \mathsf{G} < \mathsf{R} \text{ and } h = 255 \text{ if } \mathsf{G} \geq \mathsf{R}$$

Segmentation with Different Hue Ranges

0 - 100

0 - 150



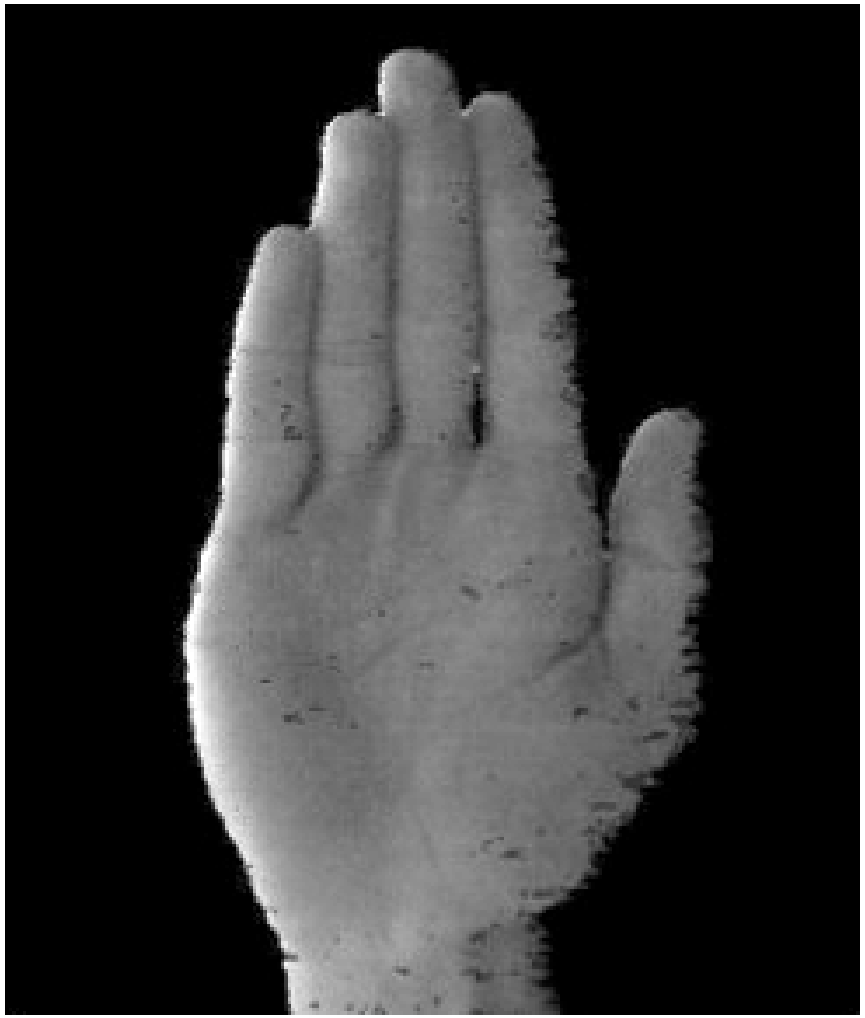
80 - 190

100 - 255

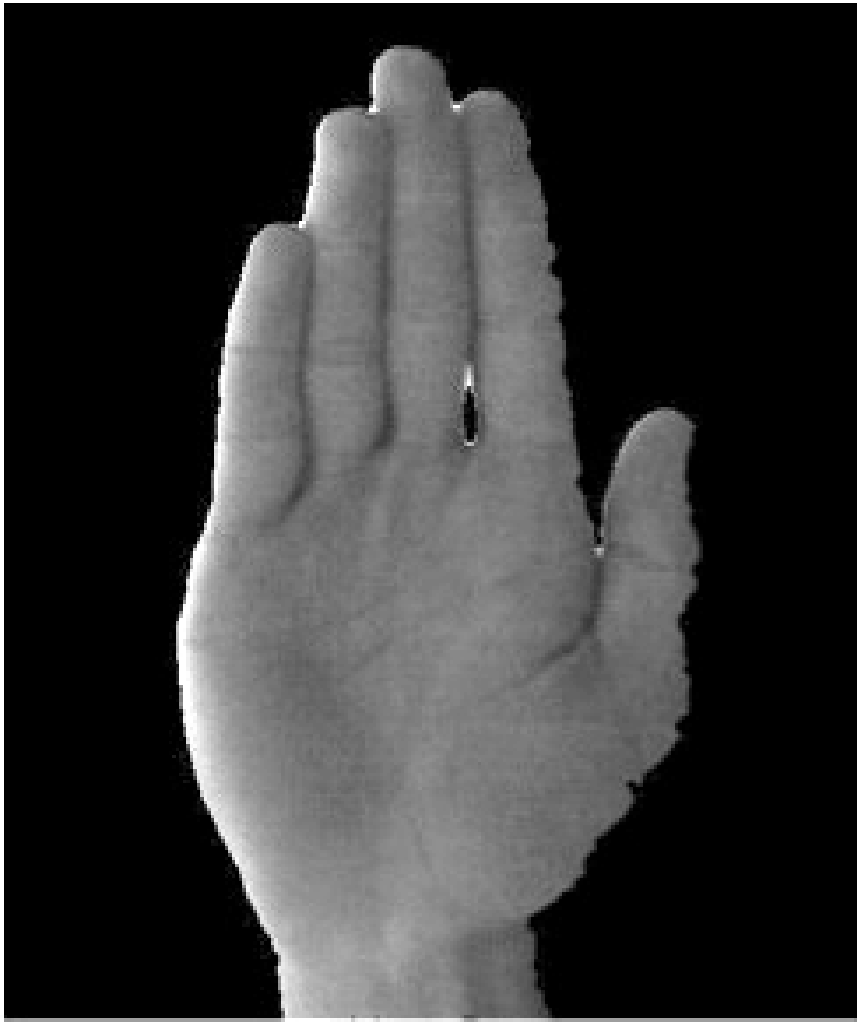
150 - 255

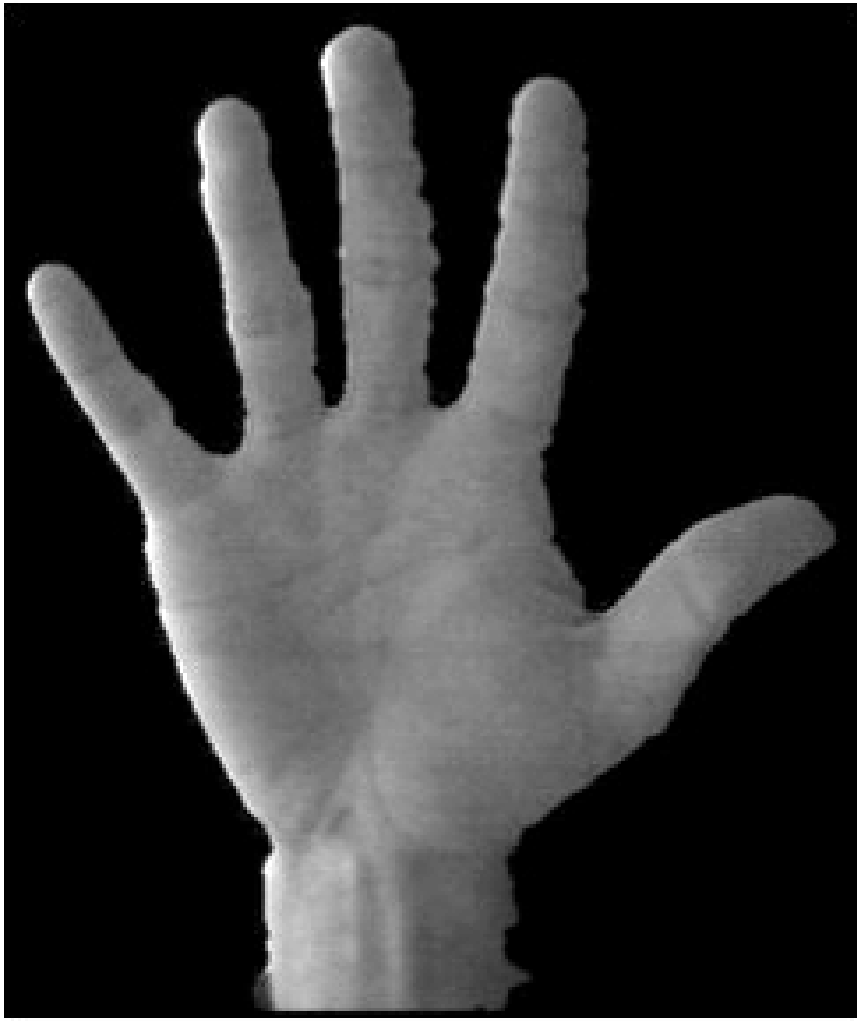
200 - 255

Hue Thresholding: Raw Segmentation



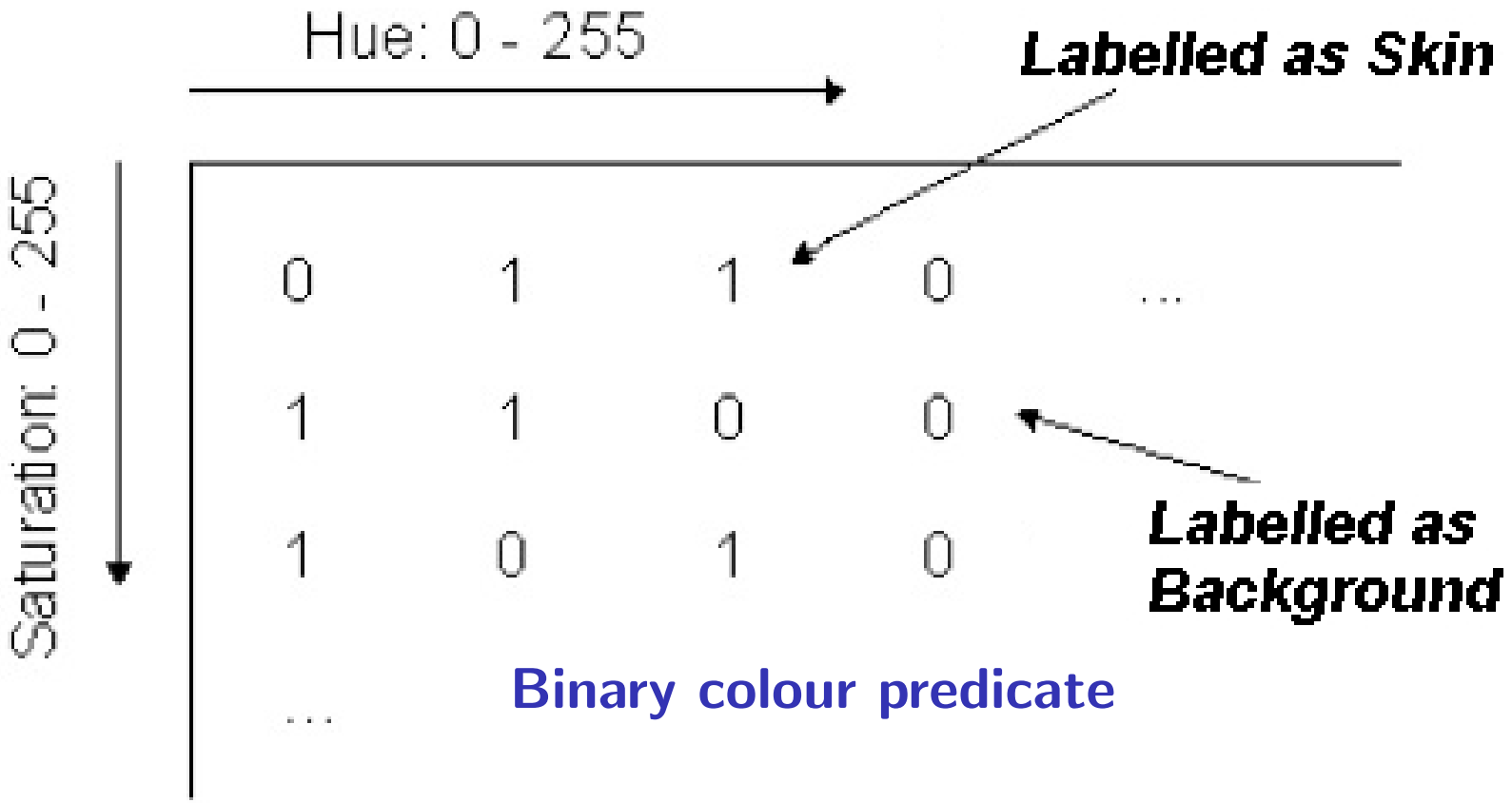
Post-Processing (by Morphological Opening + Median Filtering)





Training the Colour Predicate

Using the largest connected region of skin-coloured pixels in each training image, being found with a connected component algorithm by Haralick-Shapiro.



Training the Colour Predicate

$$\theta_1 = 5\% \times 255$$
$$\theta_2 = 95\% \times 255$$

The Use of the Colour Predicate