MORPHOLOGICAL OPERATIONS

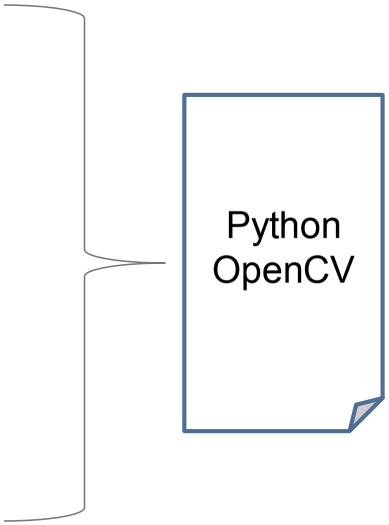
(DIGITAL IMAGE PROCESSING)

Faculty of Information Technology
Ton Duc Thang University

August 2023

Morphological Operations

- Image thresholding (binarization)
  - Example: Foreground extraction
- Connected-components
  - Pixel neighborhoods
  - Region labeling
  - Region properties
  - Application: Blob-based motion detection
- Morphological operations
  - Erosion/dilation/opening/closing
  - Applications



Binary image processing

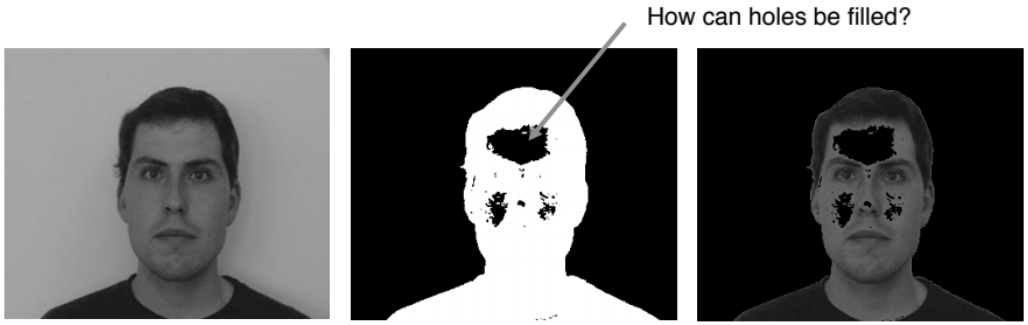
- Binary images are common
  - Intermediate abstraction in a gray-scale/color image analysis system
    - Thresholding/segmentation
    - Presence/absence of some image property
  - Text and line graphics, document image processing
- Representation of individual pixels as 0 or 1, convention:
  - foreground, object = 1 (white)
  - background = 0 (black)
- Processing by logical functions is fast and simple
- Shift-invariant logical operations on binary images: “morphological” image processing

What is Thresholding?

- The simplest binarization method
- Application example:
  - Separate out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels.
- To differentiate the pixels we are interested in from the rest (which will eventually be rejected),
  - we perform a comparison of each pixel intensity value with respect to a threshold (determined according to the problem to solve).
- Once we have separated properly the important pixels,
  - we can set them with a determined value to identify them (i.e. we can assign them a value of (black), (white) or any value that suits your needs).

Thresholding

Gray-level thresholding

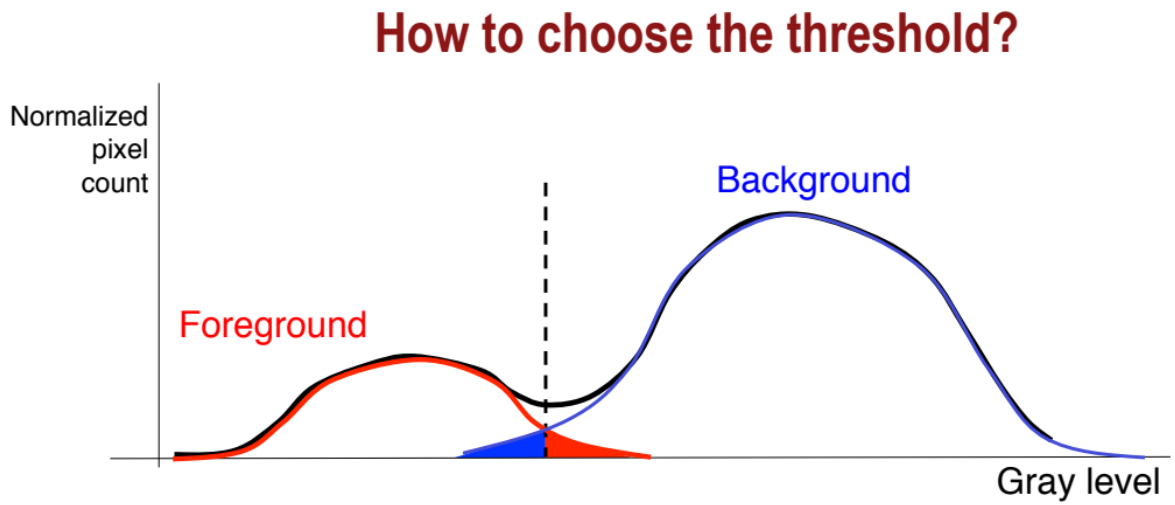


Original image
Peter $f[x,y]$

Thresholded
Peter $m[x,y]$

$f[x,y] \cdot m[x,y]$

Thresholding

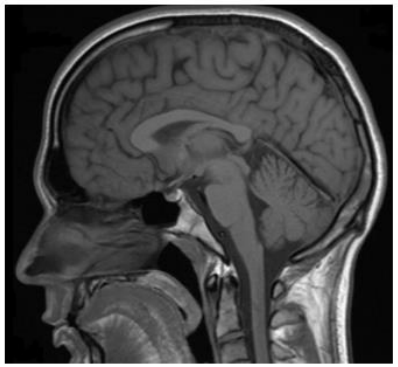


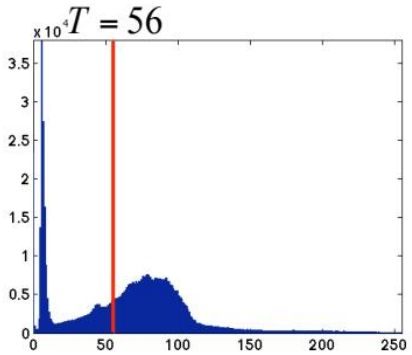
- Image segmentation based on a simple threshold:

$$g[n,m] = \begin{cases} 255, & f[n,m] > 100 \\ 0, & \text{otherwise.} \end{cases}$$

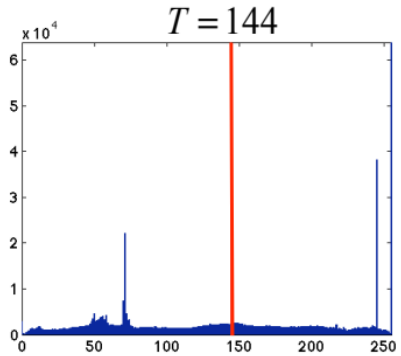
Fei-Fei Li Lecture 4- 22 6-Oct-16

Thresholding

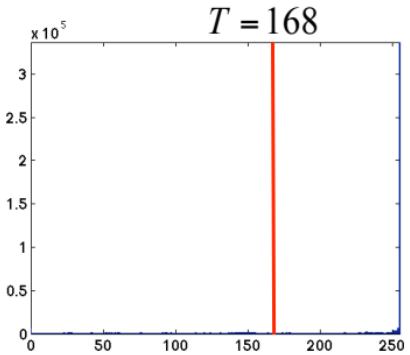




Thresholding



Thresholding



Thresholding

- Sometimes, a global threshold does not work

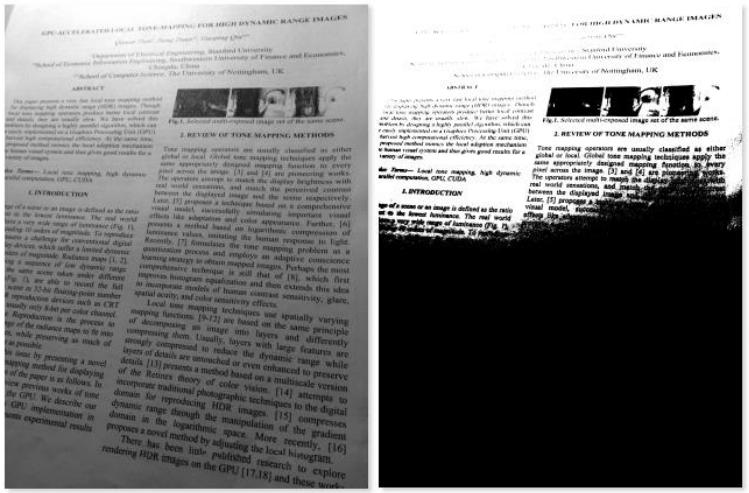


Image thresholding

- Binary Thresholding
  - If pixel intensity is greater than the threshold, value set to maxval, else set to 0 (black)
- Inverted Binary Thresholding
  - Inverted case of Binary Thresholding
- Truncated Thresholding
  - If pixel intensity value is greater than threshold, it is truncated to the threshold. All other values remain the same.

Binary

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Inverted Binary

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

Truncated

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

To Zero

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

To Zero Inverted

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Image thresholding

- To-zero Thresholding
  - Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.

- Inverted To-zero Thresholding
  - Inverted case of To-zero Thresholding.

Binary

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Inverted Binary

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

Truncated

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

To Zero

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

To Zero Inverted

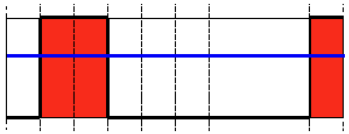
$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Threshold Binary

- This thresholding operation can be expressed as:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- So, if the intensity of the pixel $\texttt{src}(x,y)$ is higher than $\texttt{thresh}$, then the new pixel intensity is set to a $\texttt{MaxVal}$. Otherwise, the pixels are set to 0.

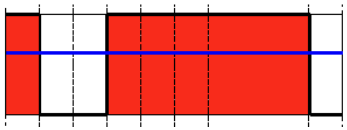


Threshold Binary, Inverted

- This thresholding operation can be expressed as:

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

- If the intensity of the pixel $\texttt{src}(x,y)$ is higher than $\texttt{thresh}$, then the new pixel intensity is set to a 0. Otherwise, it is set to $\texttt{MaxVal}$.

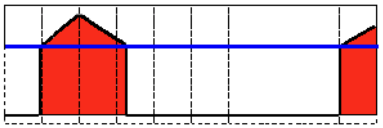


Threshold to Zero ¶

- This operation can be expressed as:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- If $\texttt{src}(x,y)$ is lower than $\texttt{thresh}$, the new pixel value will be set to 0.

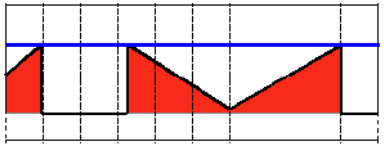


Threshold to Zero, Inverted

- This operation can be expressed as:

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- If $\texttt{src}(x,y)$ is greater than $\texttt{thresh}$, the new pixel value will be set to 0.

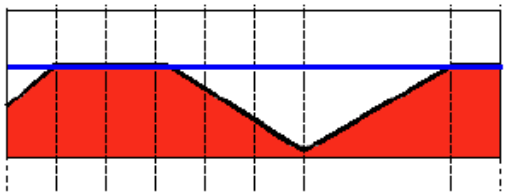


Truncate

- This thresholding operation can be expressed as:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- The maximum intensity value for the pixels is $thresh$, if $src(x,y)$ is greater, then its value is *truncated*. See figure below:



https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html

Python:

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

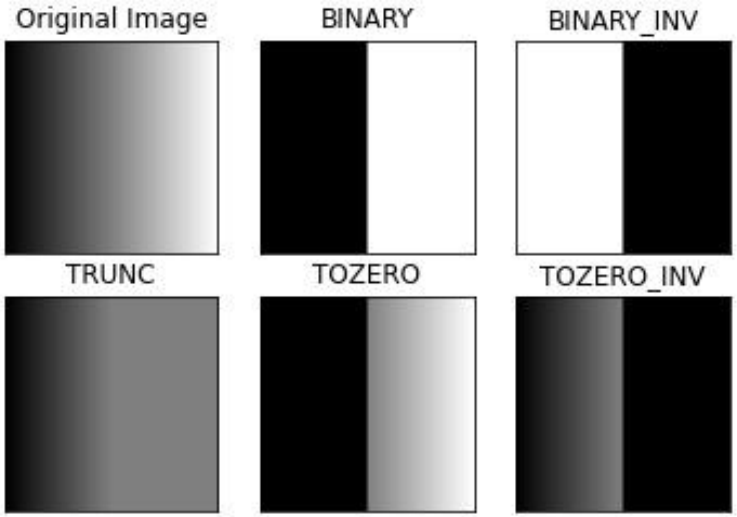
```
#include <opencv2/imgproc.hpp>
```

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_TOZERO_INV

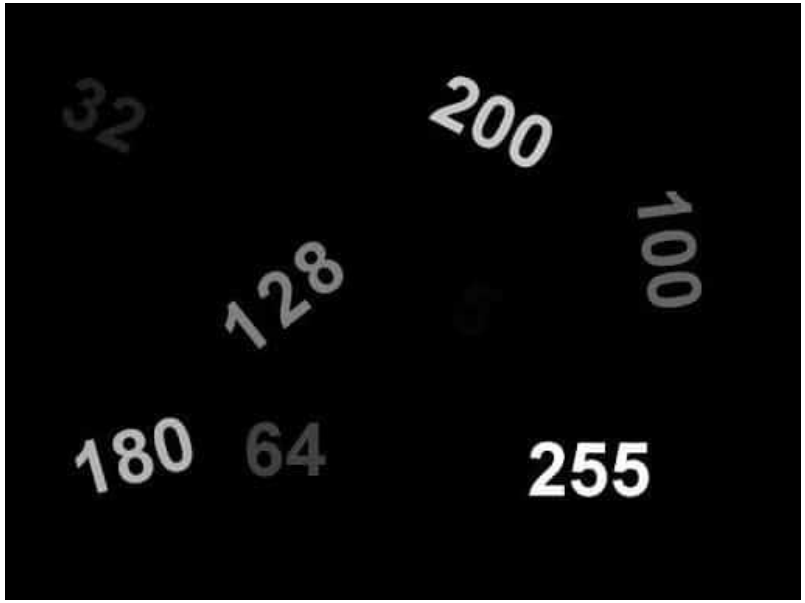
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Thresholding examples

- threshold = 127
- maxVal = 255



Input 1

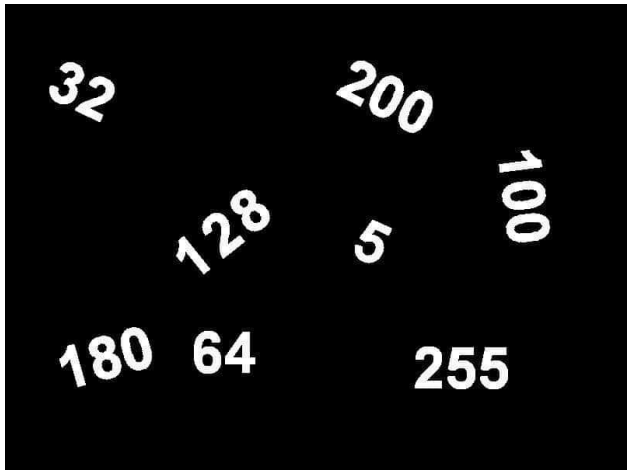


- The input image contains numbers written with intensity equal to the number itself. For ex., the pixel intensity of the number ‘200’ is 200, and the intensity of the number ‘32’ is 32. No wonder ‘32’ appears much darker than ‘200’.

Ex .1a

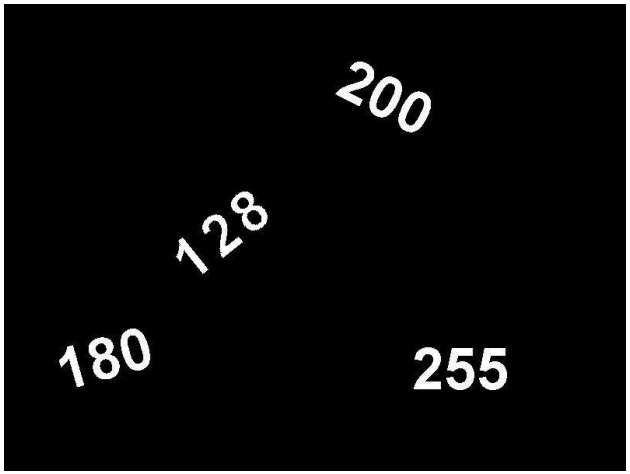
- For the “Input 1” image, apply thresholding methods to:
  - change all numbers to WHITE color (intensity = 255).

  ⇒Binary thresholding with **thresh** = 0 and **maxVal** = 255



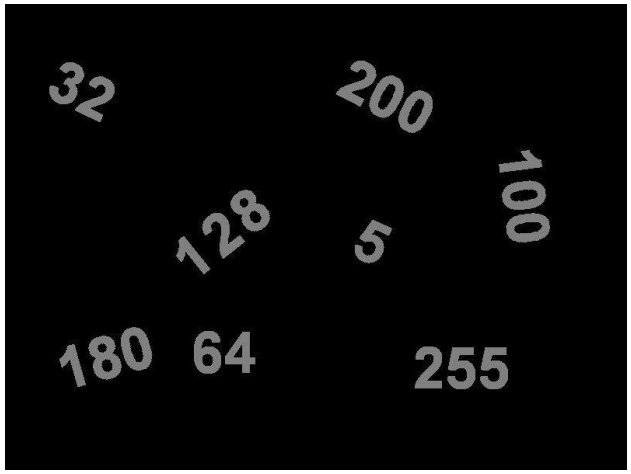
Ex .1b

- change all numbers whose values are greater than or equal to 128 into WHITE color



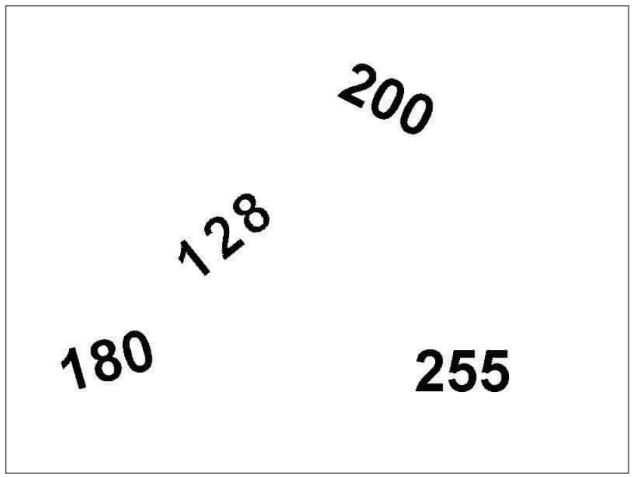
Ex .1c

- change all numbers into the Gray color (intensity = 128)



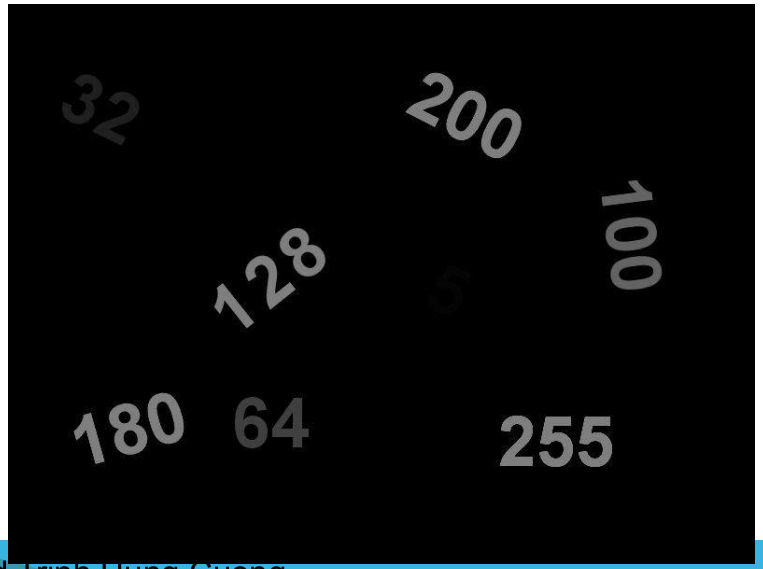
Ex .1d

- change all numbers whose values are greater than or equal to 128 into BLACK color, and also change the background color to WHITE.



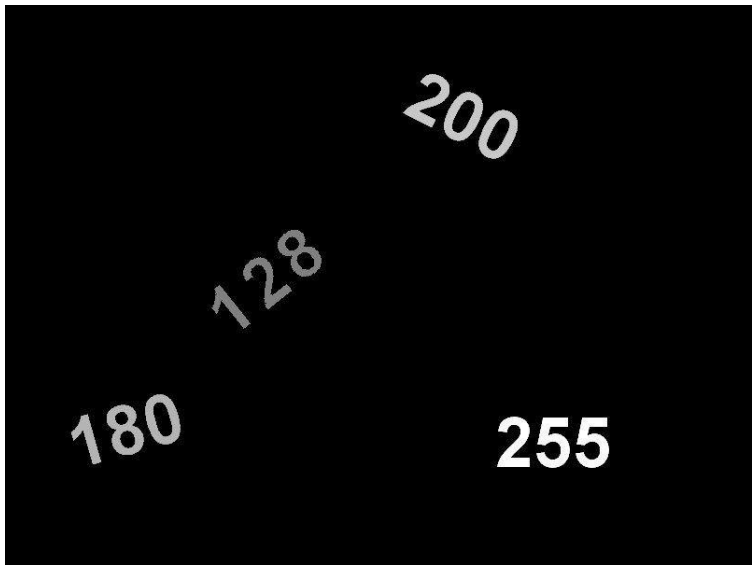
Ex .1e

- change all numbers whose values are greater than 127 into the Gray color (intensity = 127), and other numbers are unchanged.



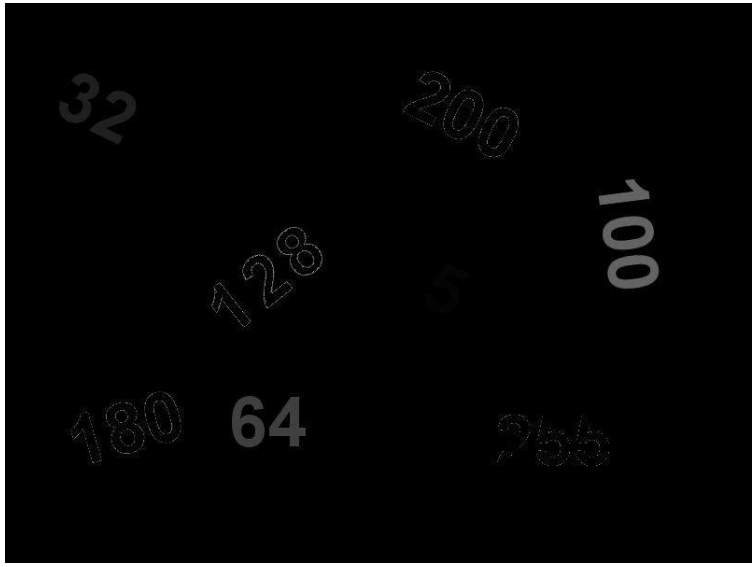
Ex .1f

- Remove all numbers whose values are less than 128, and other numbers are remained.



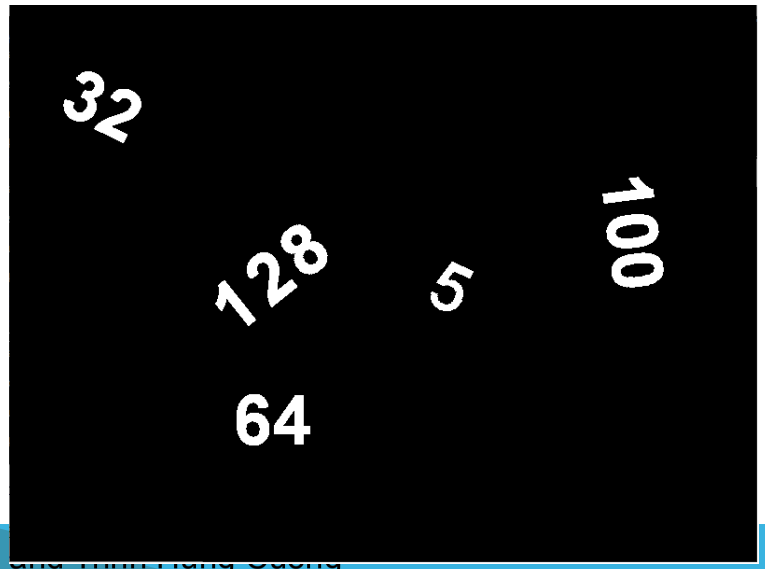
Ex .1g

- Remove all numbers whose values are greater than 127, and other numbers are remained.



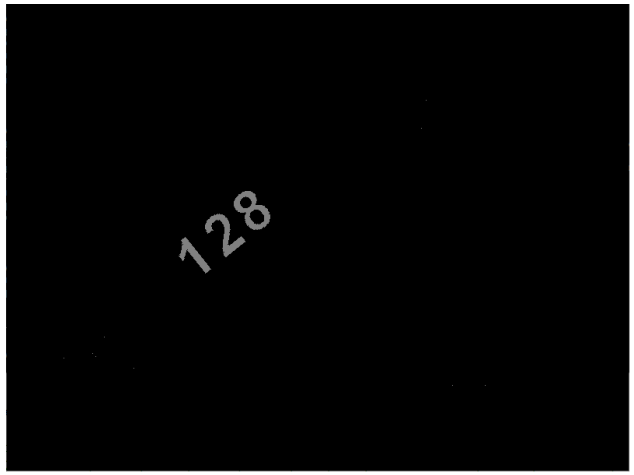
Ex .1h

- Change numbers whose values are less than 180 into WHITE color.
  - Hints: Using different thresholding techniques.



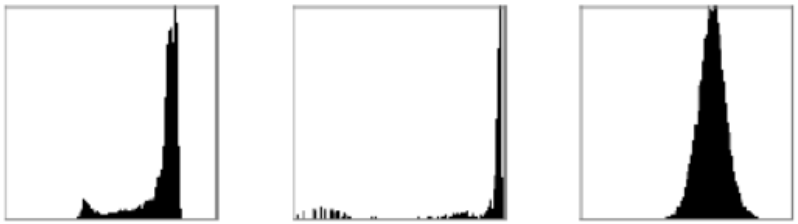
Ex .1i

- Extract each number in the input image as in separated images.

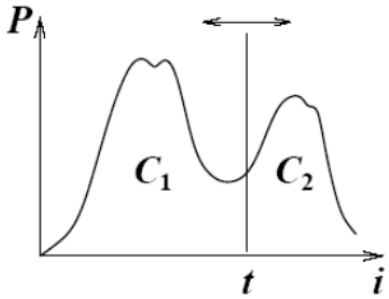


Otsu's Thresholding

- determines an optimal global threshold value from the image histogram
  - produces the appropriate results for bimodal images (histogram contains two clearly expressed peaks, which represent different ranges of intensity values)



Otsu's Thresholding (ct)



https://www.scribd.com/document/407105529/ME5286-Lecture9-pdf

- Proposed by N.Otsu (Japan), 1978
- Consider a **candidate threshold** *t*
  - *t* defines two classes of grayvalues
- Define measure of **separation of classes**
  - distance between classes as function of *t*
- Find optimal threshold t_{opt} that **maximises separation**

Otsu's Thresholding (ct)

Mean and variance of **total** normalised histogram $P(i)$:

$$\mu = \sum_{i=0}^{G_{max}} iP(i) \qquad \sigma^2 = \sum_{i=0}^{G_{max}} (i-\mu)^2 P(i)$$

https://www.scribd.com/document/407105529/ME5286-Lecture9-pdf

Threshold t splits $P(i)$ into **two classes** C_1, C_2 with

$$\mu_1(t) = \frac{1}{q_1(t)} \sum_{i=0}^{t} iP(i) \qquad \sigma_1^2(t) = \frac{1}{q_1(t)} \sum_{i=0}^{t} [i-\mu_1(t)]^2 P(i)$$

$$\mu_2(t) = \frac{1}{q_2(t)} \sum_{i=t+1}^{G_{max}} iP(i) \qquad \sigma_2^2(t) = \frac{1}{q_2(t)} \sum_{i=t+1}^{G_{max}} [i-\mu_2(t)]^2 P(i)$$

$$q_1(t) = \sum_{i=0}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{G_{max}} P(i) \qquad q_1(t) + q_2(t) = 1$$

Otsu's Thresholding (ct)

- Total variance σ^2 has two components
  - **within-class variance** for given t
  - $\Rightarrow$ weighted sum of two class variances
  - **between-class variance** for given t
  - $\Rightarrow$ distance between classes
- Within-class variance is

$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

  - $\Rightarrow$ note that $\mu = q_1(t)\mu_1(t) + q_2(t)\mu_2(t)$
- Between-class variance is the rest of σ^2

$$\sigma_B^2(t) = \sigma^2 - \sigma_W^2(t)$$

https://www.scribd.com/document/407105529/ME5286-Lecture9-pdf

Otsu's Thresholding (ct)

- It is easy to show that

$$\sigma_B^2(t) = q_1(t)q_2(t)\left[\mu_1(t) - \mu_2(t)\right]^2$$
$$= q_1(t)\left[1 - q_1(t)\right]\left[\mu_1(t) - \mu_2(t)\right]^2$$

- Optimal threshold t_{opt} best separates the two classes
- $\sigma_W^2(t) + \sigma_B^2(t)$ is constant $\longrightarrow$ two equivalent options
  - *minimise* $\sigma_W^2(t)$ as *overlap of classes*
  - *maximise* $\sigma_B^2(t)$ as *distance between classes*

$\Rightarrow$ Use second option

https://www.scribd.com/document/407105529/ME5286-Lecture9-pdf

Otsu's Thresholding (ct)

$$q_1(t+1) = q_1(t) + P(t+1) \quad \text{with} \quad q_1(0) = P(0)$$

$$\mu_1(t+1) = \frac{q_1(t)\mu_1(t) + (t+1)P(t+1)}{q_1(t+1)} \quad \text{with} \quad \mu_1(0) = 0 \quad (2)$$

$$\mu_2(t+1) = \frac{\mu - q_1(t+1)\mu_1(t+1)}{1 - q_1(t+1)}$$

https://www.scribd.com/document/407105529/ME5286-Lecture9-pdf

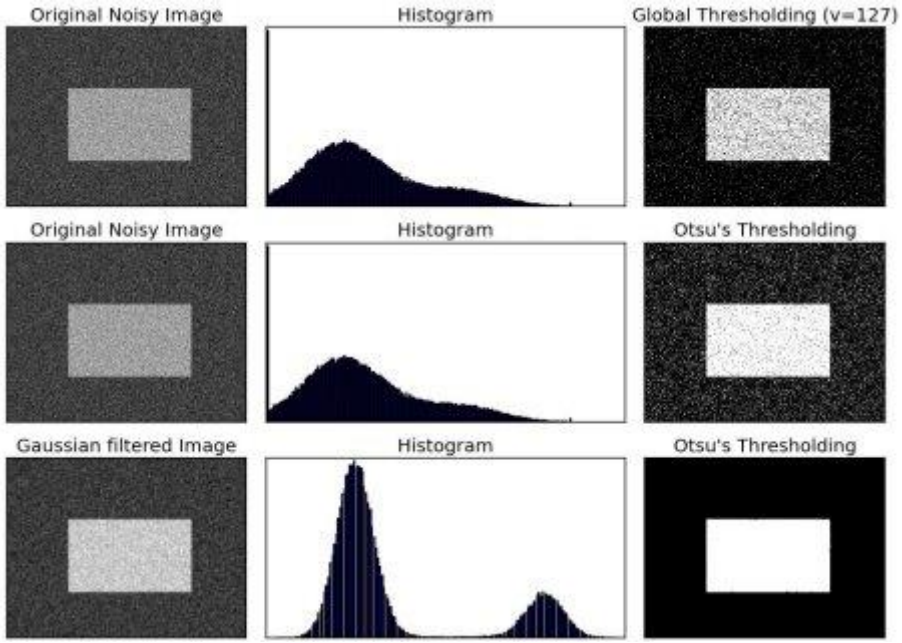
Algorithm 1: Otsu threshold selection

1. Compute image histogram $P(i)$, calculate μ and σ
2. For each $0 < t < G_{max}$
   - recursively compute $q_1(t)$, $\mu_1(t)$ and $\mu_2(t)$ by eq.(2)
   - calculate $\sigma_B^2(t)$ by eq.(1)
3. Select threshold as $t_{opt} = \arg\max_t \sigma_B^2(t)$

Otsu's Thresholding (ct)

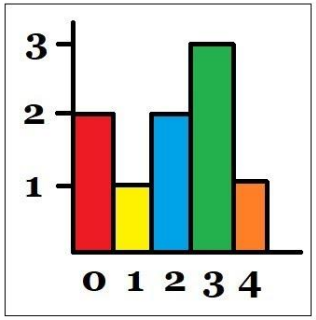
https://learnopencv.com/otsu-thresholding-with-opencv/

Otsu's Thresholding (ct)

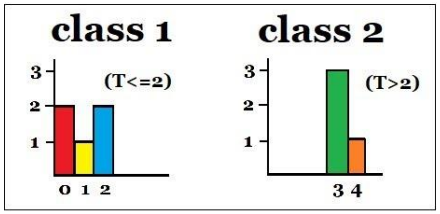


Otsu thresholding

The aim is to find the threshold "T"
& classify pixels into 2 classes (class 1 & class 2)
so that the V_w is minimum (V_b is maximum)



if T = 2 →



	T=0	T=1	T=2	T=3	T=4
V_b	1.142857	1.38888	1.25	0.5	0
V_w	0.63492	0.38888	0.52777	1.277777	1.777777

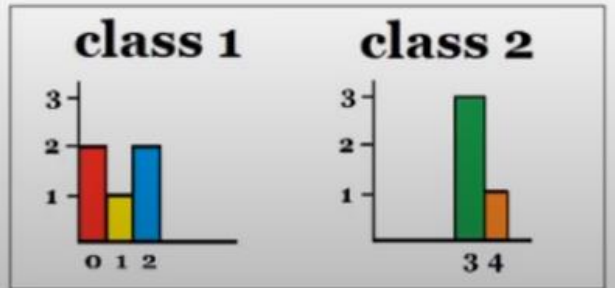
Ex. 2: Reproduce the resulted table.

https://www.youtube.com/watch?v=Ofi1Fn18YLc

Between class variance

if pixels are classified into N classes (categories),
then the between class variance (V_b) = V_T - V_w,
where V_T is the total variance

if pixels are classified into 2 classes,
then the between class variance (V_b) = $W_1W_2(\mu_1-\mu_2)^2$



$W_1 = 5/9$ $W_2 = 4/9$

$\mu_1 = 1$ $\mu_2 = 13/4$

$$V_b = \frac{5}{9} * \frac{4}{9} * (1-\frac{13}{4})^2 = 1.25$$

Ex. 3

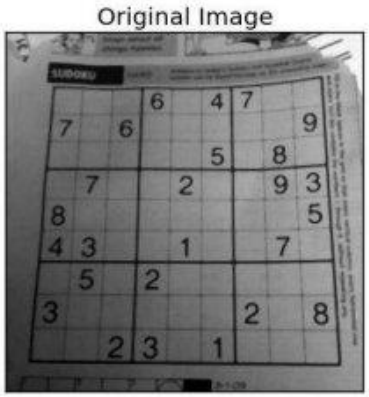
- Let The Following Image Be Given:

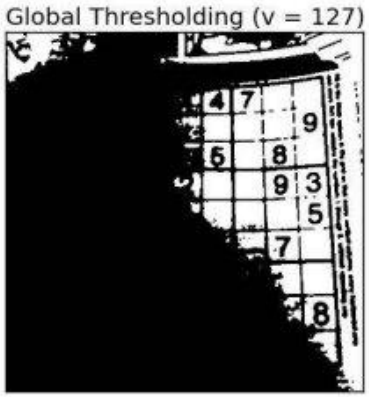
3	2	8	2
5	1	7	1
9	1	6	1
8	0	5	2

- Determine The Threshold For Image Binarization Using The Otsu Method.

Adaptive Thresholding

- A global threshold might not be good in all cases, e.g. if an image has different lighting conditions in different areas
- Adaptive thresholding determines the threshold for a pixel based on a small region around it
  - get different thresholds for different regions of the same image which gives better results for images with varying illumination





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

Adaptive Thresholding (ct)

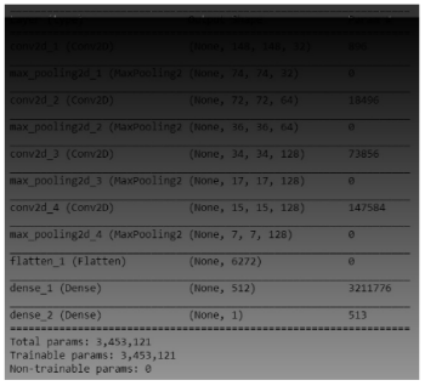
- Parameters:
  - **Threshold type**: Must be either “Binary thresholding” or “inverted Binary thresholding”. This tells us what value to assign to pixels greater/less than the threshold.
  - **maxValue**: This is the value assigned to the pixels after thresholding. This depends on the thresholding type. If the type is “Binary thresholding”, all the pixels greater than the threshold are assigned this maxValue.
  - **adaptive Method**: This tells us how the threshold is calculated from the pixel neighborhood
    - **MEAN_C** : The threshold value is the mean of the neighbourhood area minus the constant **C**
    - **GAUSSIAN_C** : The threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant **C** (read more in [1], [2], [3])
  - **blockSize** determines the size of the neighbourhood area.
  - **C** : Constant subtracted from the mean or weighted mean.

Adaptive Thresholding (ct)

- Neighborhood size has to be large enough to cover sufficient foreground and background pixels
- Choosing regions which are too large can violate the assumption of approximately uniform illumination
- **C** constant is used to properly tune the threshold value
  - There may be situations where the mean value alone is not discriminating enough between the background and foreground — thus by adding or subtracting some value C, we can improve the results of our threshold.

- For each pixel in the image:
  - Calculate the statistics (such as mean, median, etc.) from its neighbourhood. Its statistics minus the constant **C** will be the threshold value for that pixel.
  - Change the pixel value using the threshold

Adaptive Thr



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dense_2 (Dense)	(None, 1)	513

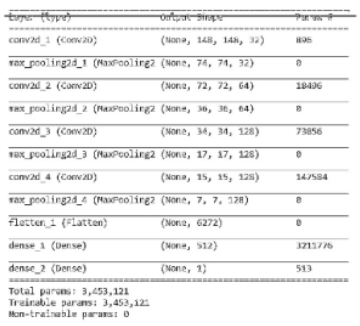
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

Original image

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dense_2 (Dense)	(None, 1)	513

Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

Otsu method



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dense_2 (Dense)	(None, 1)	513

Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

Adaptive thresholding

Ex. 4

- Apply the adaptive thresholding ($Threshold\ type\ =\ "Binary\ thresholding"$, $maxValue\ =\ 255,\ method = MEAN\_C,\ C\ =\ 2,\ blockSize\ = 3$) into the following image:

Input image 6×9

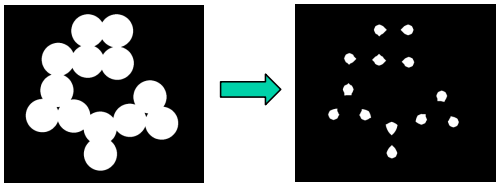
```
[[  0 255 255 255 255 255  55  55  55]
[  0 190 190 190 190 190  85  85  85]
[  0 127 127 127 127 127 255 255 255]
[  0   0   0   0   0   0   0   0   0]
[  0 190 190 190 190 190  85  85  85]
[  0 255 255 255 255 255  55  55  55]]
```

Output image 6×9

0	255	?	?	?	?	?	0	0
0	255	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
0	255	?	?	?	?	?	?	?

Morphological Image Processing

- Binary dilation and erosion
- Set-theoretic interpretation
- Opening, closing, morphological edge detectors
- Hit-miss filter
- Morphological filters for gray-level images
- Cascading dilations and erosions
- Rank filters, median filters, majority filters

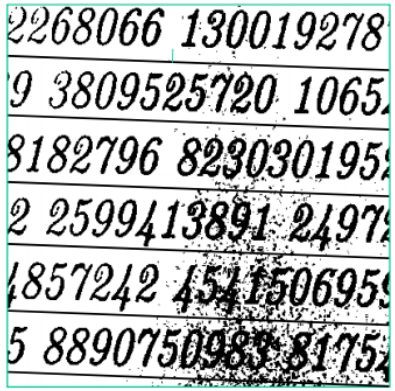


INTEREST-POINT DETECTION

Feature extraction typically starts by finding the salient interest points in the image. For robust image matching, we desire interest points to be repeatable under perspective transformations (or, at least, scale changes, rotation, and translation) and real-world lighting variations. An example of feature extraction is illustrated in Figure 3. To achieve scale invariance, interest points are typically computed at multiple scales using an image pyramid [15]. To achieve rotation invariance, the patch around each interest point is canonically oriented in the direction of the dominant gradient. Illumination changes are compensated by normalizing the mean and standard deviation of the pixels of the gray values within each patch [16].

Example

- Text segmentation and recognition.
- Binary morphological operations useful after segmentation to get better segmentation of the objects.



Some symbols have been fragmented.

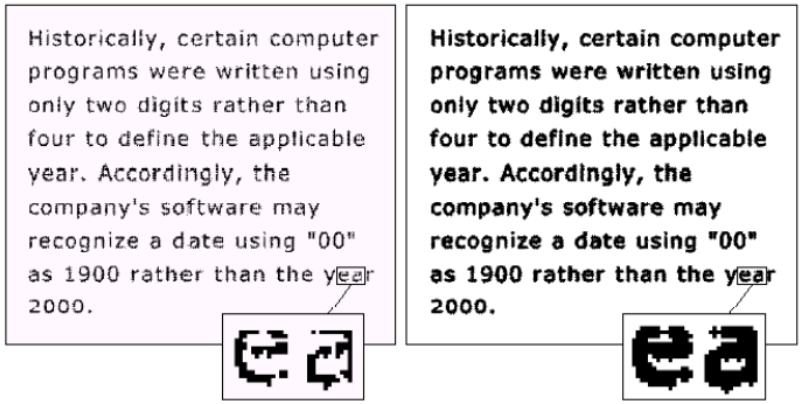
Some symbols are connected with background noise.

Symbols can be connected with neighboring symbols.

Find and remove lines or frames.

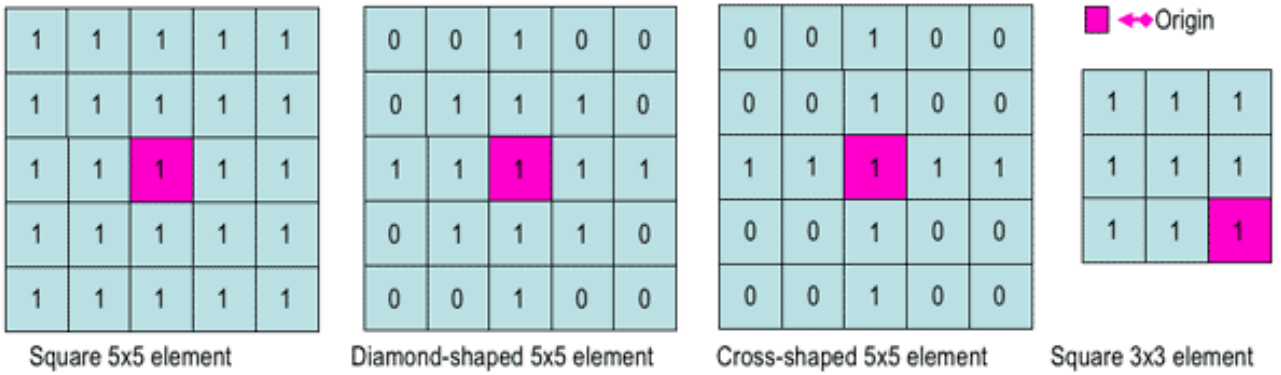
INF 4300

Morphological operations

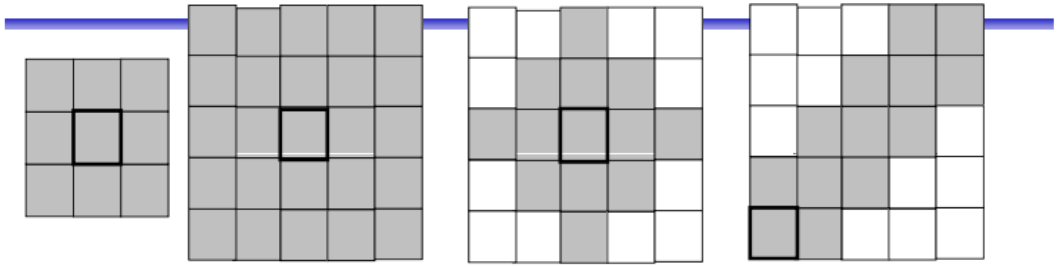


0	1	0
1	1	1
0	1	0

Structure elements



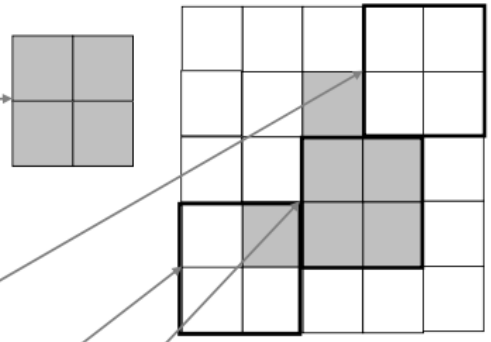
Repetition – structure elements



- Can have different sizes and shapes
- A structuring element has an origo
  - Origo is a pixel
  - Origo *can* be outside the element.
  - Origo is often marked on the structuring element using 
  - Origo can be flat or non-flat (have different values)
    - We will only work with a flat structure element

Repetition - Fit vs. Hit

- A structure element for a binary image is a small matrix of pixels
- Let the structure element overlay the binary image containing an objects at different pixel positions. The following cases arise:
  - Positions where the element does not overlap with the object.
  - Positions where the element parly overlaps the object – where the element hits the object
  - Positions where the element fits inside the object – where the element fits the object



HIT vs FIT

https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

The structuring element is said to **fit** the image if, for each of its pixels set to 1, the corresponding image pixel is also 1.

Similarly, a structuring element is said to **hit**, or intersect, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.

Erosion example

Repetition -Erosion of a binary image
Simplified notation

- To compute the erosion of pixel (x,y) in image f with the structure element S: place the structure elements such that its origo is at (x,y). Compute

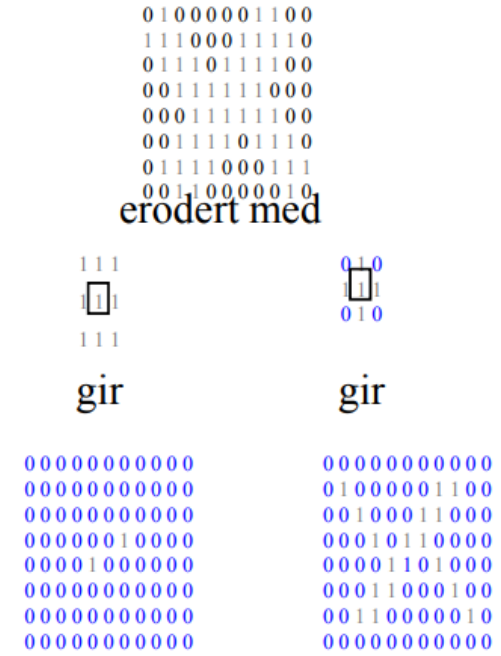
$$g(x,y)=\begin{cases}1 \text{ if } S \text{ fits } f\\ 0 \text{ otherwise}\end{cases}$$

- Erosion of the image f with structure element S is denoted

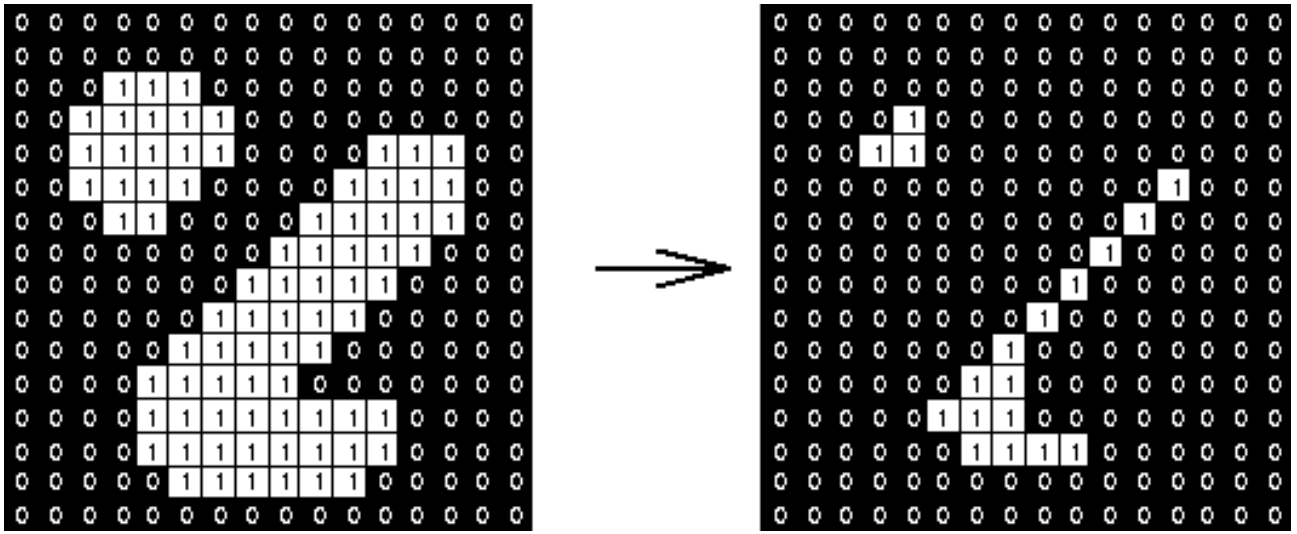
  $\varepsilon(f|S) = f \ominus S$

- Erosion of a set A with the stucture element B is defined as the position of all pixels x in A such that B is included in A when origo of B is at x.

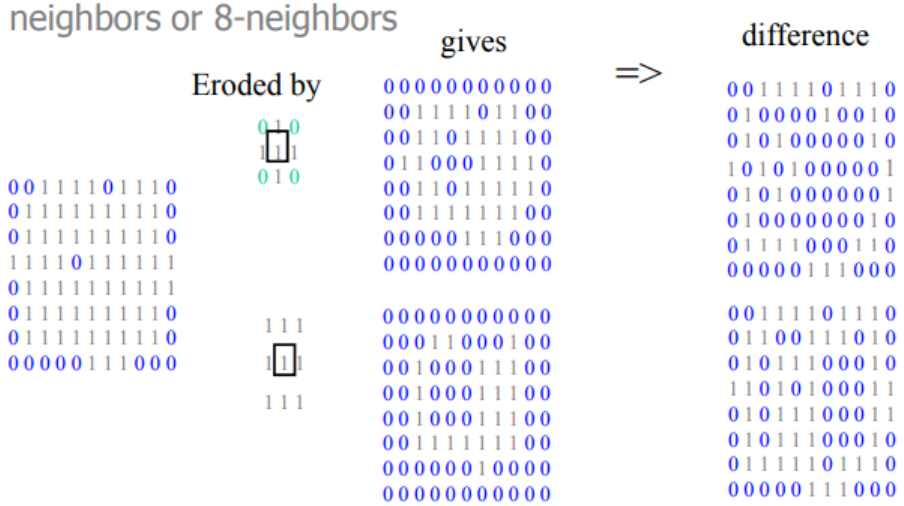
$$A \theta B = \{x | B_x \subseteq A\}$$



Erosion



- Erosion removes pixels on the border of an object.
- We can find the border by subtracting an eroded image from the original image: $g = f - (f \ominus s)$
- The structure element decides if the edge pixels will be 4-neighbors or 8-neighbors



Edge detection

$$f - (f \ominus S)$$

Example use: find border pixels in a region

Binary erosion with square structuring element

Original
(701x781)

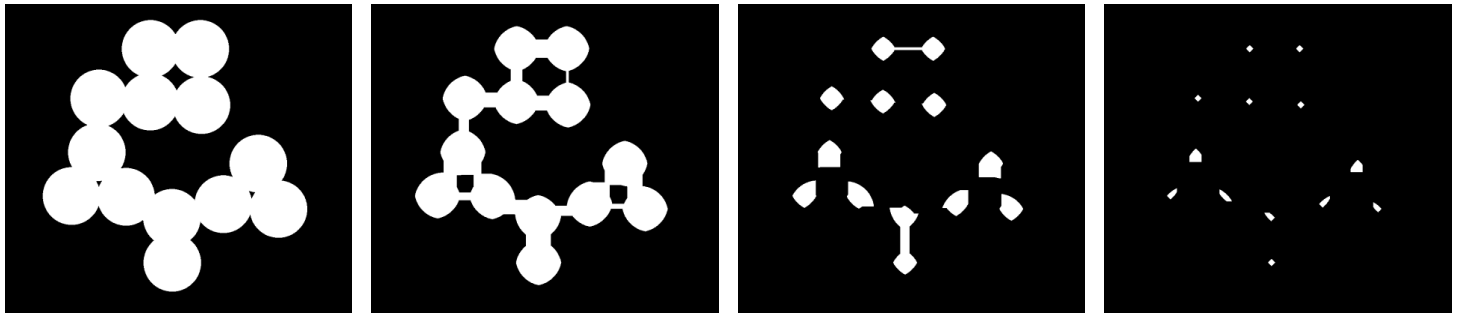
erosion with
3x3 structuring element

erosion with
7x7 structuring element

- Shrinks the size of 1-valued objects
- Smoothens object boundaries
- Removes peninsulas, fingers, and small objects

$$g[x,y] = AND\left[\hat{W}\{f[x,y]\}\right] := erode(f,W)$$

Example: blob separation/detection by erosion



Original binary image

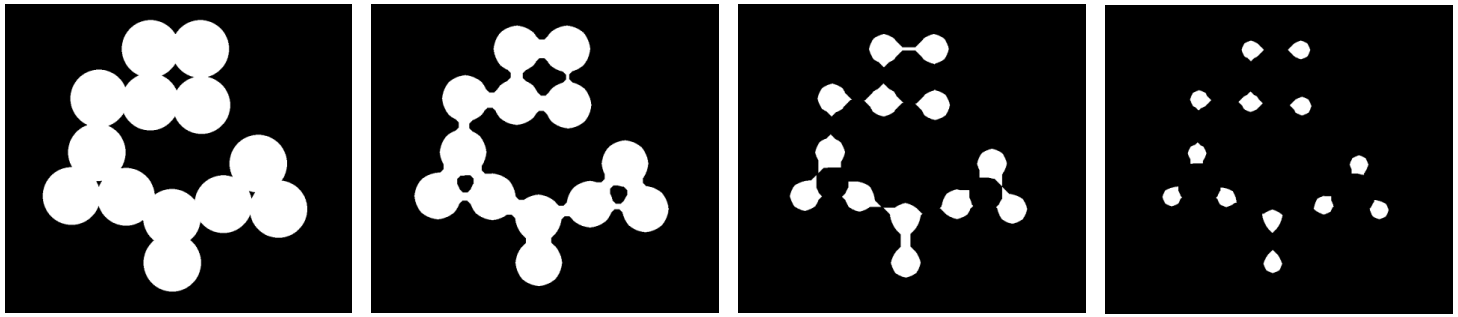
Circles (792x892)

Erosion by 30x30 structuring element

Erosion by 70x70 structuring element

Erosion by 96x96 structuring element

Example: blob separation/detection by erosion



Original binary image

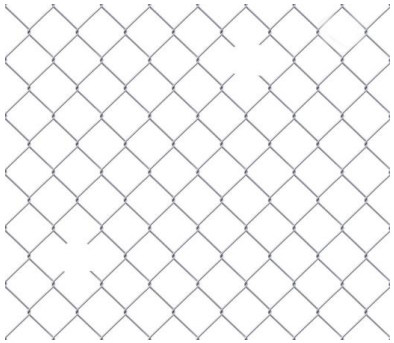
Circles (792x892)

Erosion by disk-shaped structuring element Diameter=15

Erosion by disk-shaped structuring element Diameter=35

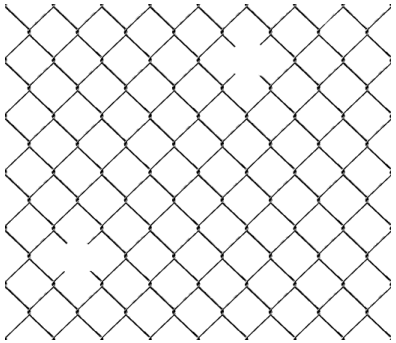
Erosion by disk-shaped structuring element Diameter=48

Example: chain link fence hole detection

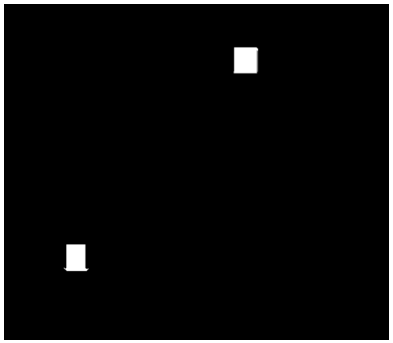


Original grayscale image

Fence (1023 x 1173)



Fence thresholded using Otsu's method



Erosion with 151x151 "cross" structuring element

Ex. 5

- Apply the erosion into the following image:

Input image 6×9

0	1	1	1	0	0	1	1	1
0	1	1	1	0	0	1	1	0
1	1	1	0	0	0	1	1	1
1	1	1	0	0	1	1	1	0
1	1	1	0	0	1	1	1	0
0	1	1	1	1	1	1	1	0

Output image 6×9

0	0	1	0	?	?	?	?	?
0	0	1	0	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?

Kernel (S) = $\begin{matrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{matrix}$

Dilation example

Dilation of a binary image

- Place S such that origo lies in pixel (x,y) and use the rule

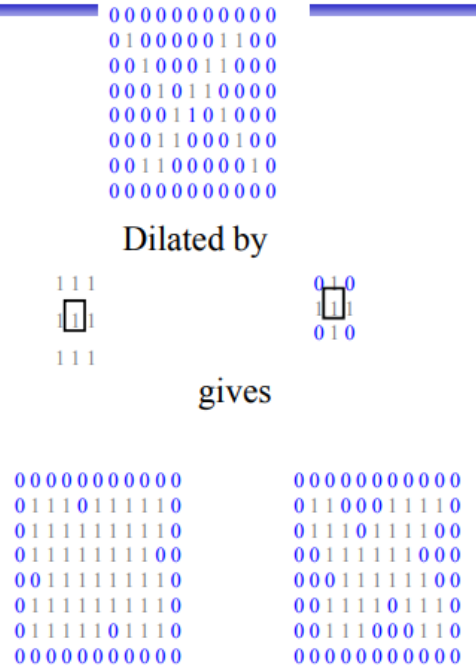
$$g(x,y)=\begin{cases}1 \text{ if } S \text{ hits } f\\ 0 \text{ otherwise}\end{cases}$$

- The image f dilated by the structure element S is denoted:

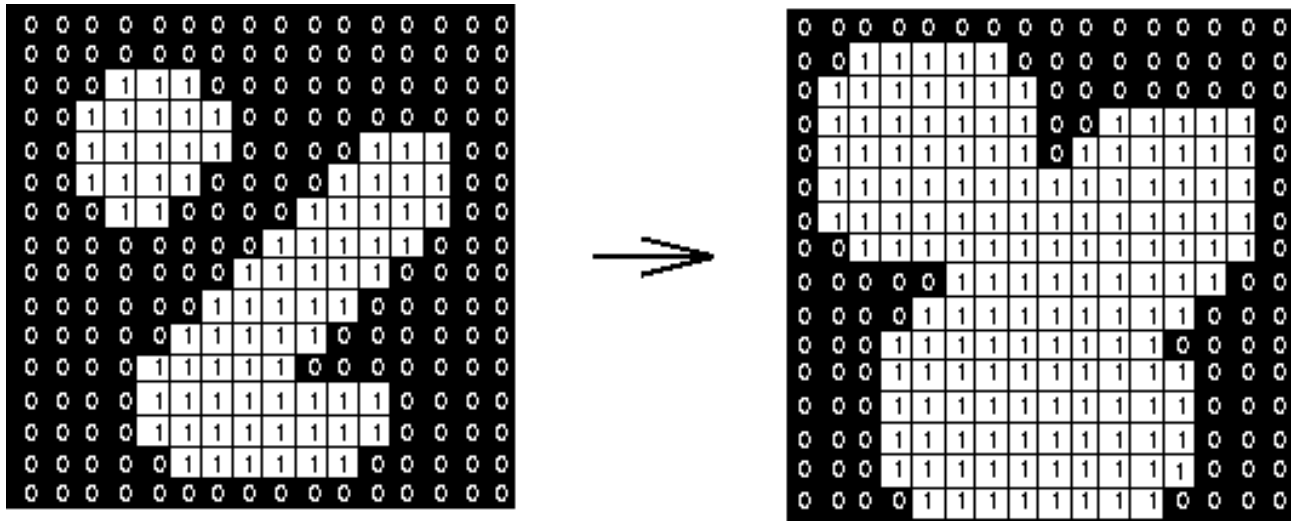
$$f \oplus S$$

- Dilation of a set A with a structure element B is defined as the position of all pixels x such that B overlaps with at least one pixel in A when origo is placed at x.

$$A \oplus B = \{x \mid B_x \cap A \neq \emptyset\}$$



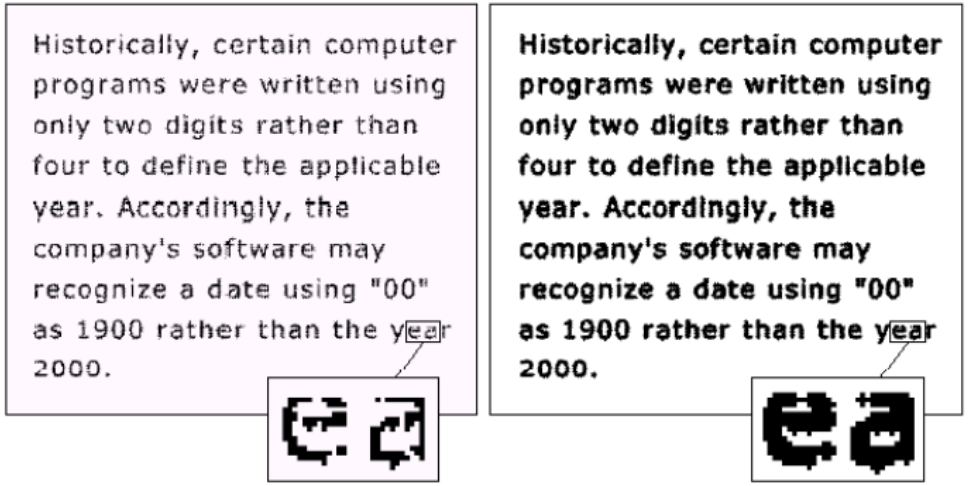
Dilation



Effect of dilation

- Expand the object
- Both inside and outside borders of the object
- Dilation fills holes in the object
- Dilation smooths out the object contour
- Depends on the structure element
- Bigger structure element gives greater effect

Example of use of dilation – fill gaps

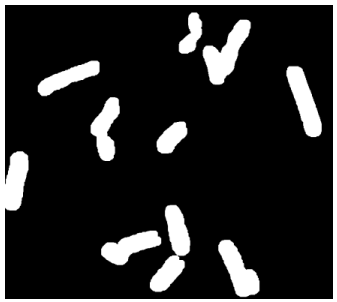


0	1	0
1	1	1
0	1	0

Binary dilation with square structuring element

Original
(701x781)

dilation with
3x3 structuring element



dilation with
7x7 structuring element

- Expands the size of 1-valued objects
- Smoothes object boundaries
- Closes holes and gaps

$$g[x,y] = OR[W\{f[x,y]\}] := dilate(f,W)$$

Ex. 6

- Apply the dilation into the following image:

Input image 6×9

0	0	1	1	0	0	1	1	1
0	0	0	1	0	0	1	1	0
1	0	1	0	0	0	1	0	1
0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	0	0

Output image 6×9

0	1	1	1	?	?	?	?	?
1	0	1	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?

Kernel (S) = $\begin{matrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{matrix}$

Relationship between dilation and erosion

- Duality: erosion is dilation of the background

$$dilate\left(f,W\right)=NOT\left[erode\left(NOT\left[f\right],\hat{W}\right)\right]$$
$$erode\left(f,W\right)=NOT\left[dilate\left(NOT\left[f\right],\hat{W}\right)\right]$$

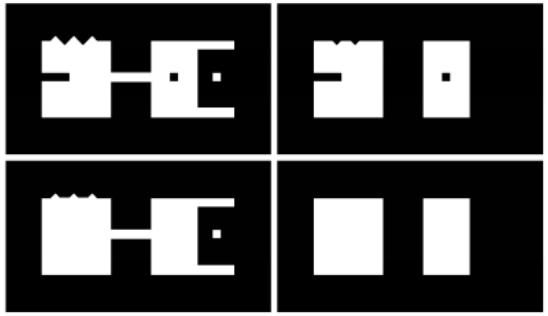
- But: erosion is not the inverse of dilation

$$f\left[x,y\right]\neq erode\left(dilate\left(f,W\right),W\right)$$
$$\neq dilate\left(erode\left(f,W\right),W\right)$$

Opening and closing

- Goal: smoothing without size change
- Open filter: $open(f, W) = dilate(erode(f, W), W)$
- Close filter: $close(f, W) = erode(dilate(f, W), W)$
- Open filter and close filter are biased
  - Open filter removes small 1-regions
  - Close filter removes small 0-regions
- Unbiased size-preserving smoothers
  - $close - open(f, W) = close(open(f, W), W)$
  - $open - close(f, W) = open(close(f, W), W)$
  - *close-open* and *open-close* are duals, but not inverses of each other.

Example of opening and closing



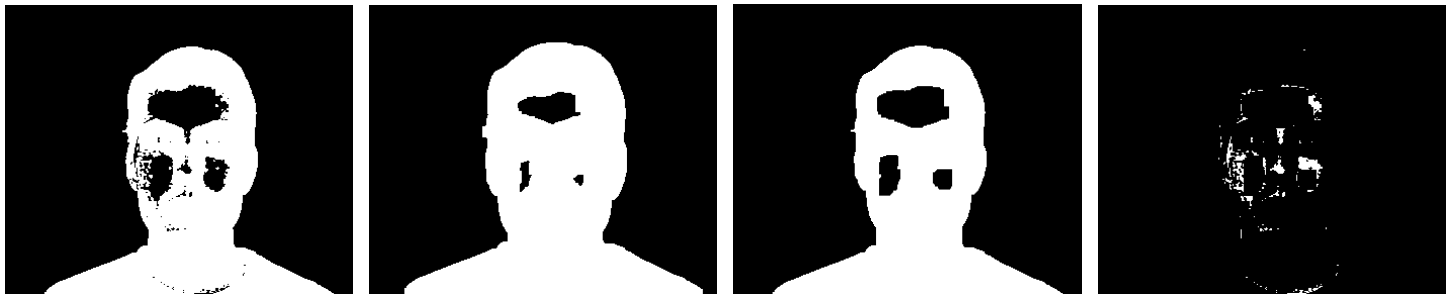
a b
c d

a) **Original image I**
b) **Opening of I**
c) **Closing of I**
d) **Closing of b)**

Closing is a dual operation to opening

$$f \circ S \subseteq f \subseteq f \bullet S$$

Small hole removal by closing



Original binary mask

Dilation 10x10

Closing 10x10

Difference to original mask

Ex. 7

- Apply the opening into the following image:

Input image 9×9

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0

Kernel (S):

0	1	0
1	**1**	1
0	1	0

Output image 9×9

?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?

Ex. 8

- Apply the closing into the following image:

Input image 9×9

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0

Kernel (S):

0	1	0
1	**1**	1
0	1	0

Output image 9×9

?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?

Ex. 9

- Apply the opening and then closing into the following image:

Input image 9×9

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0

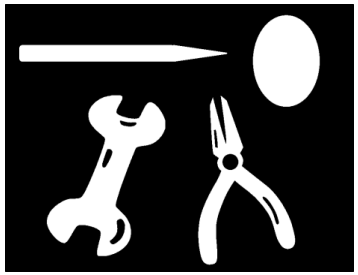
Kernel (S):

0	1	0
1	**1**	1
0	1	0

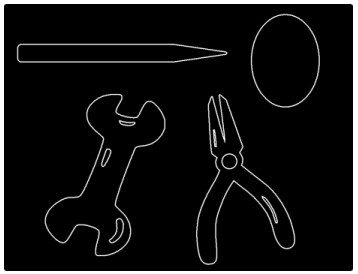
Output image 9×9

?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?

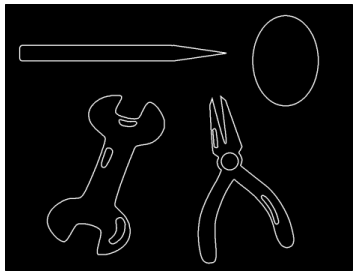
Morphological edge detectors



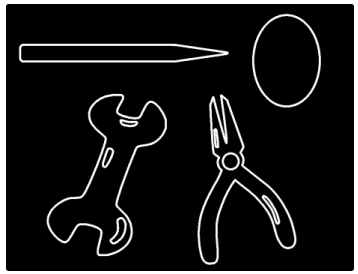
$f[x,y]$



$dilate(f,W) \neq f$



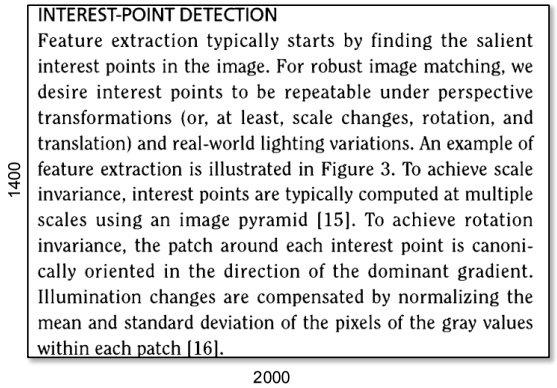
$erode(f,W) \neq f$



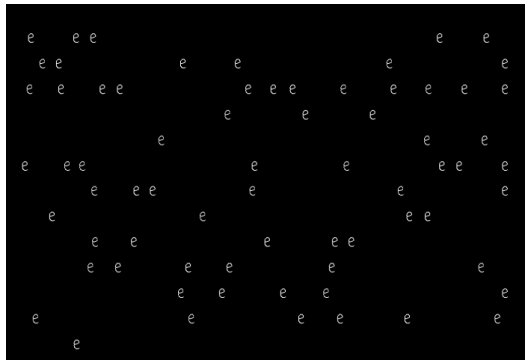
$dilate(f,W) \neq erode(f,W)$

Recognition by erosion

Binary image f



$dilate\left(erode\left(NOT\left[f\right],W\right),W\right)$



Structuring element W

Recognition by erosion

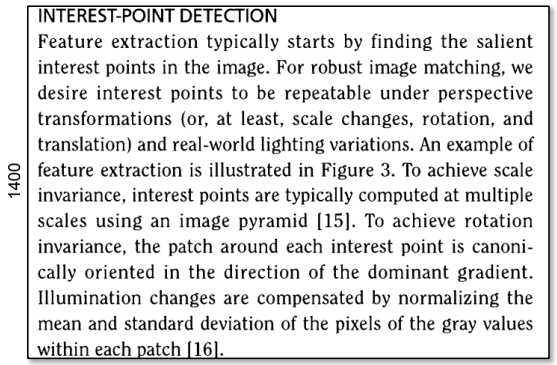
INTEREST-POINT DETECTION

Feature extraction typically starts by finding the salient interest points in the image. For robust image matching, we desire interest points to be repeatable under perspective transformations (or, at least, scale changes, rotation, and translation) and real-world lighting variations. An example of feature extraction is illustrated in Figure 3. To achieve scale invariance, interest points are typically computed at multiple scales using an image pyramid [15]. To achieve rotation invariance, the patch around each interest point is canonically oriented in the direction of the dominant gradient. Illumination changes are compensated by normalizing the mean and standard deviation of the pixels of the gray values within each patch [16].

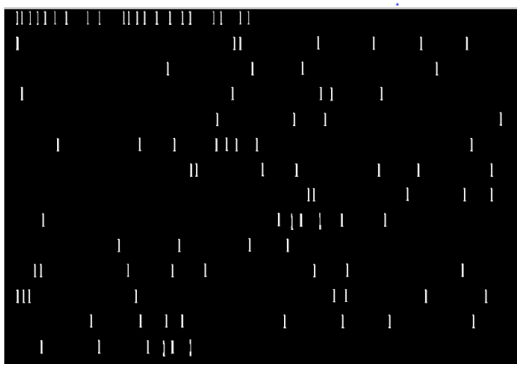
Structuring element W

Recognition by erosion

Binary image f



$dilate\left(erode\left(NOT\left[f\right],W\right),W\right)$



Structuring element W

Recognition by erosion

INTEREST-POINT DETECTION

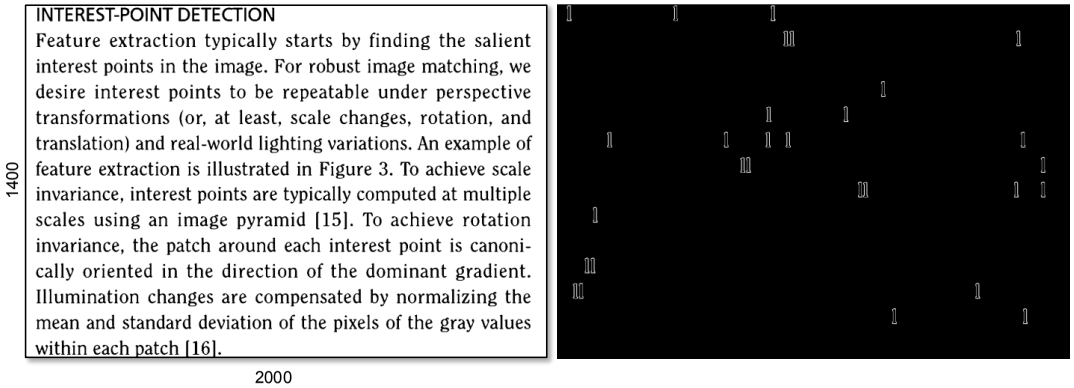
Feature extraction typically starts by finding the salient interest points in the image. For robust image matching, we desire interest points to be repeatable under perspective transformations (or, at least, scale changes, rotation, and translation) and real-world lighting variations. An example of feature extraction is illustrated in Figure 3. To achieve scale invariance, interest points are typically computed at multiple scales using an image pyramid [15]. To achieve rotation invariance, the patch around each interest point is canonically oriented in the direction of the dominant gradient. Illumination changes are compensated by normalizing the mean and standard deviation of the pixels of the gray values within each patch [16].

Structuring element W

Hit-miss filter

$$dilate\Big(erode\big(NOT\big[f\big],V\big)\,\&\,erode\big(f,W\big),W\Big)$$

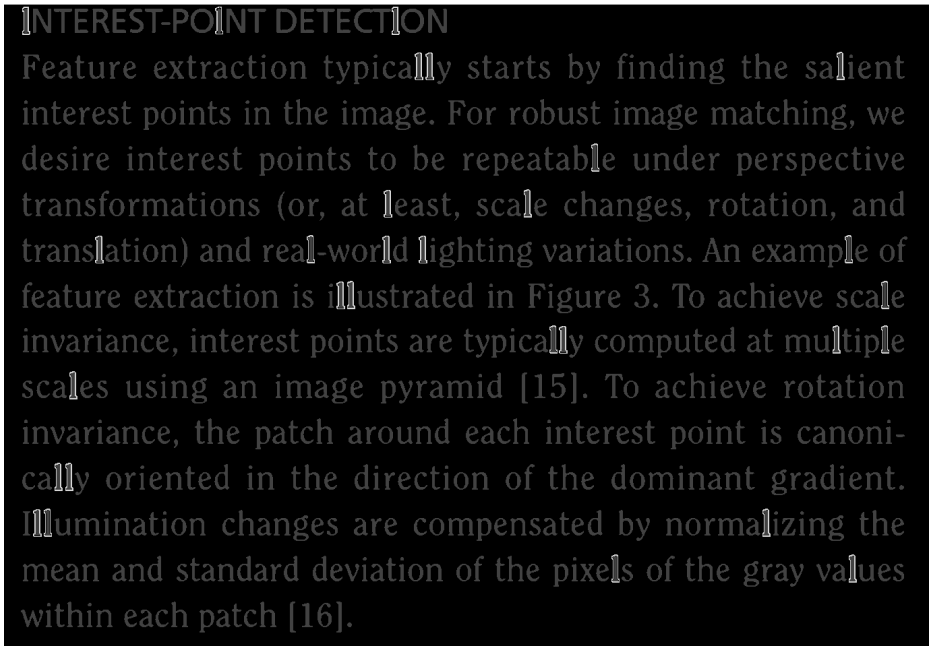
Binary image f



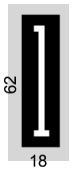
Structuring element V

Structuring element W

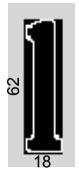
Hit-miss filter



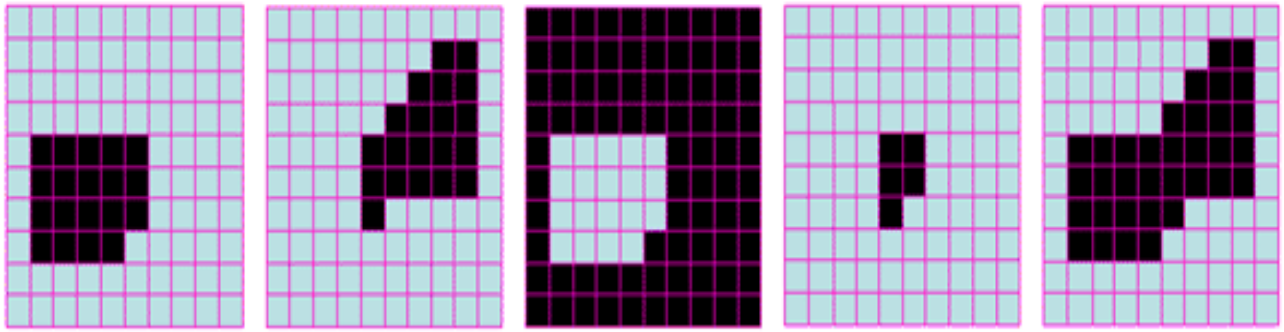
Structuring element V



Structuring element W



- Set operations on binary images: (from left to right)
  (1) a binary image f,
  (2) a binary image g,
  (3) the complement f^c of f,
  (4) the intersection f ∩ g, and
  (5) the union f ∪ g



Dilation/erosion for gray-level images

- Explicit decomposition into threshold sets not required in practice
  - Flat dilation operator: local maximum over window W

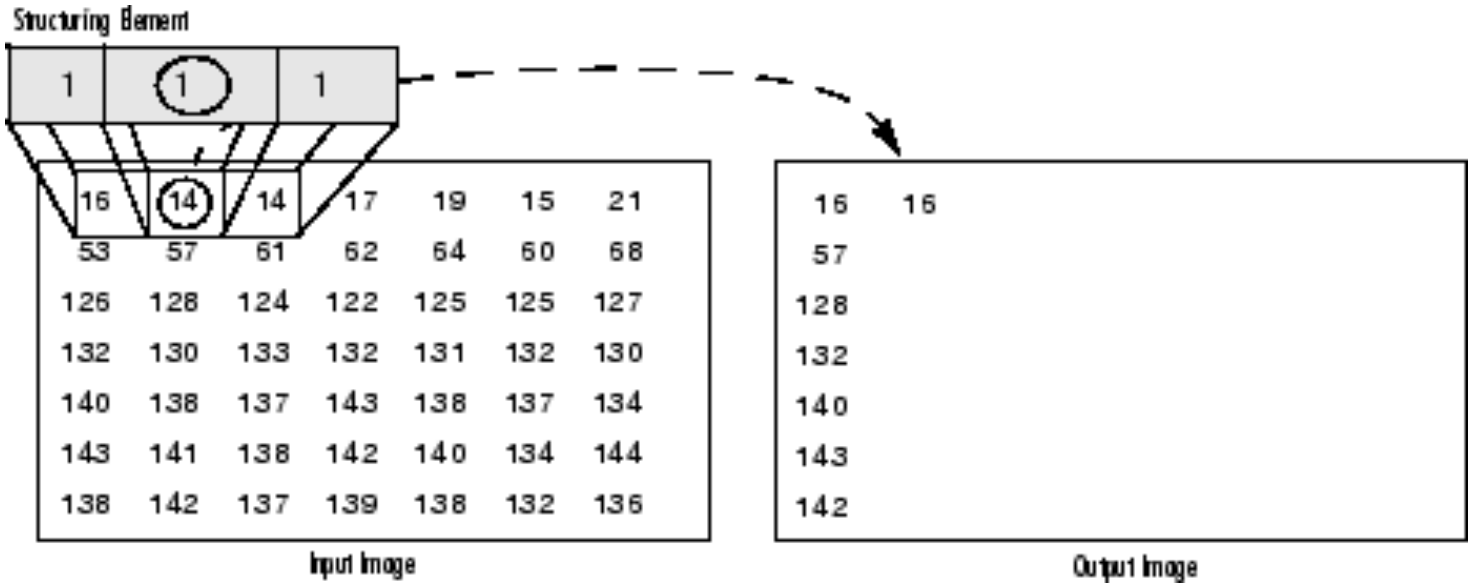
$$g[x,y] = \max\left\{W\left\{f[x,y]\right\}\right\} := dilate(f,W)$$

  - Flat erosion operator: local minimum over window W

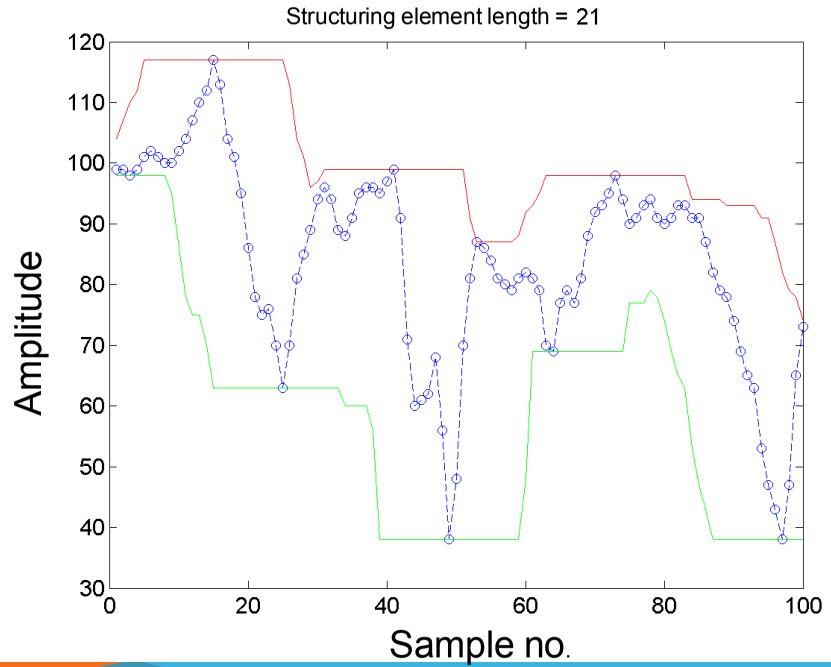
$$g[x,y] = \min\left\{\hat{W}\left\{f[x,y]\right\}\right\} := erode(f,W)$$

  - Binary dilation/erosion operators contained as special case

Dilation in Grascale image – Ex. 10



1-d illustration of erosion and dilation



——Dilation
——Erosion
——Original

Structuring element: horizontal line

Image example

Original 394 x 305


Dilation 10x10 square


Erosion 10x10 square

Flat dilation with different structuring elements

Original


Diamond

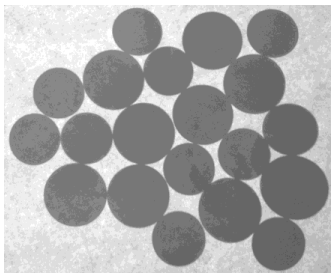

Disk


20 degree line

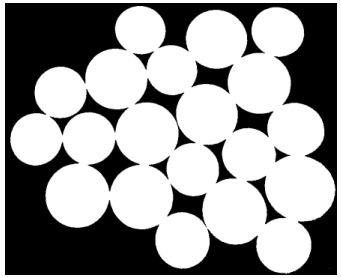

9 points


2 horizontal lines

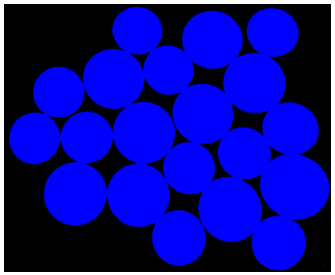
Example: counting coins



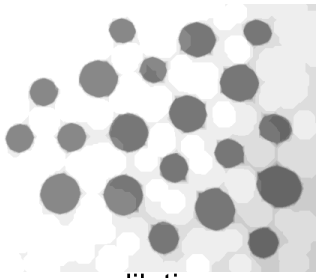
Original



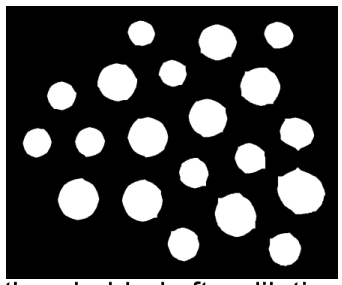
thresholded



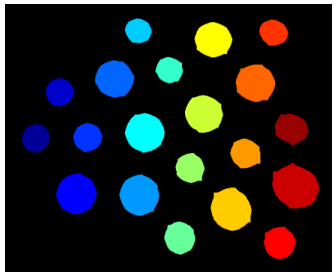
1 connected component



dilation

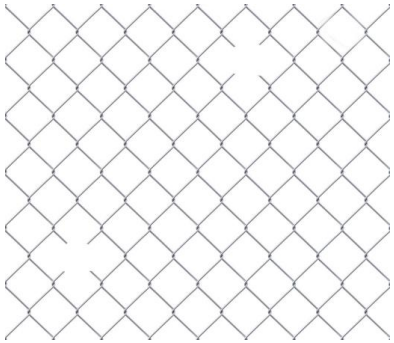


thresholded after dilation

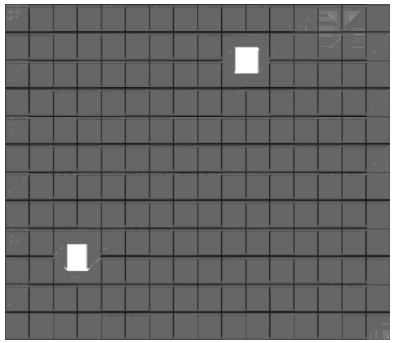


20 connected components

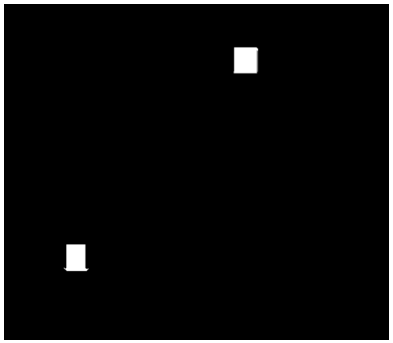
Example: chain link fence hole detection



Original grayscale image
Fence (1023 x 1173*)*

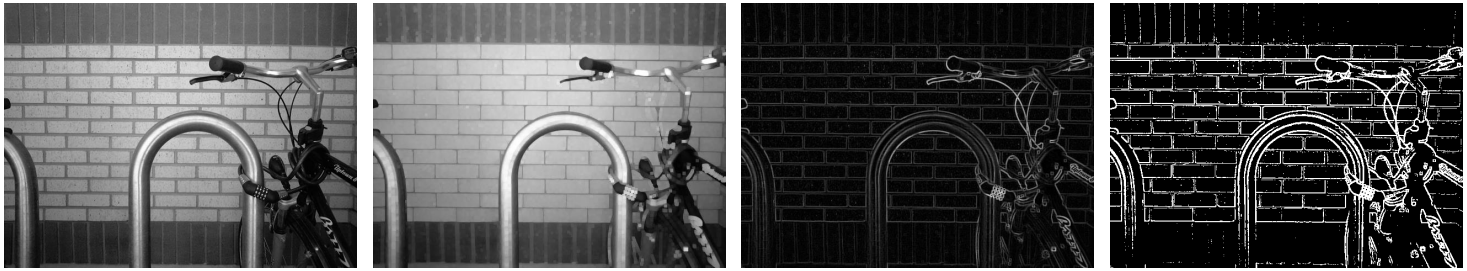


Flat erosion with 151x151 “cross” structuring element



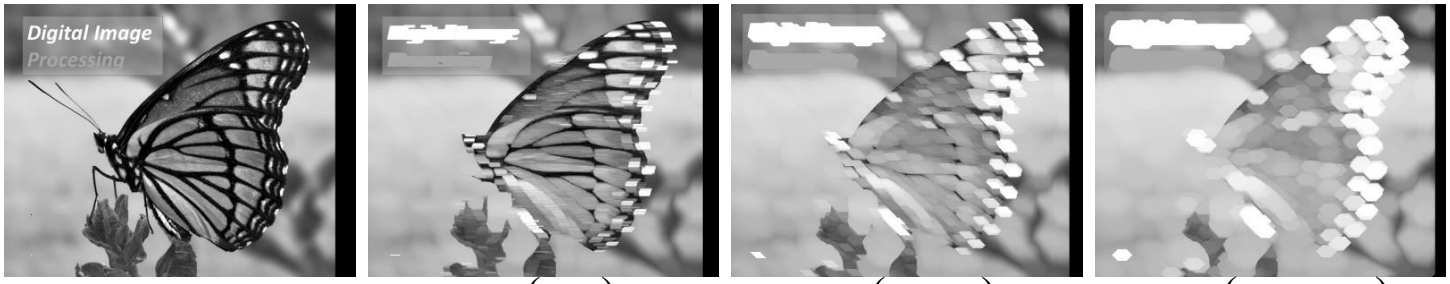
Binarized by Thresholding

Morphological edge detector



original f | dilation g | $g - f$ | $g - f$ thresholded

Cascaded dilations



f $\quad dilate(f, w_1)$ $\quad dilate(f, w_1, w_2)$ $\quad dilate(f, w_1, w_2, w_3)$

$$dilate\left[dilate(f, w_1), w_2\right] = dilate(f, w)$$

$$\text{where } w = dilate(w_1, w_2)$$

Cascaded erosions

- Cascaded erosions can be lumped into single erosion

$$
\begin{aligned}
erode\left[erode\left(f, w_1\right), w_2\right] &= erode\left[-dilate\left(-f, \hat{w}_1\right), w_2\right] \\
&= -dilate\left[dilate\left(-f, \hat{w}_1\right), \hat{w}_2\right] \\
&= -dilate\left(-f, \hat{w}\right) \\
&= erode\left(f, w\right) \\
\text{where} \quad w &= dilate\left(w_1, w_2\right)
\end{aligned}
$$

  - New structuring element (SE) is not the erosion of one SE by the other, but dilation.

References

- https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gaa42a3e6ef26247da787bf34030ed772c
- https://www.geeksforgeeks.org/apply-a-gauss-filter-to-an-image-with-python/
- https://theailearner.com/tag/adaptive-thresholding/
- https://dsp.stackexchange.com/questions/54375/how-to-approximate-gaussian-kernel-for-image-blur
- https://docs.opencv.org/