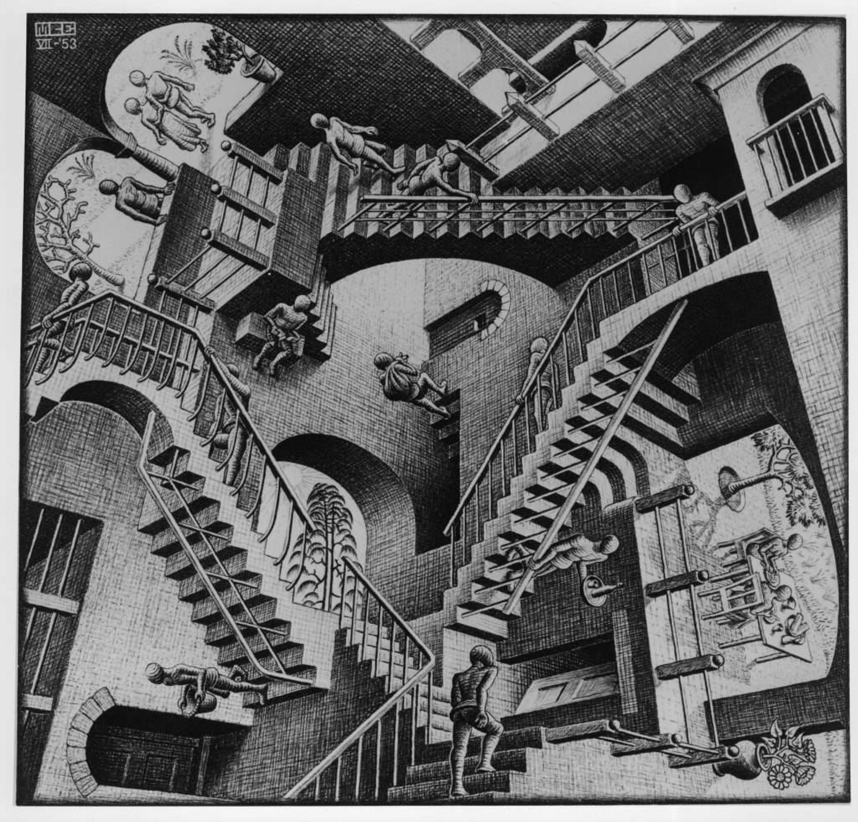
ME 536

Week 9-10: Images as Tensors

Pixels: oh pixels

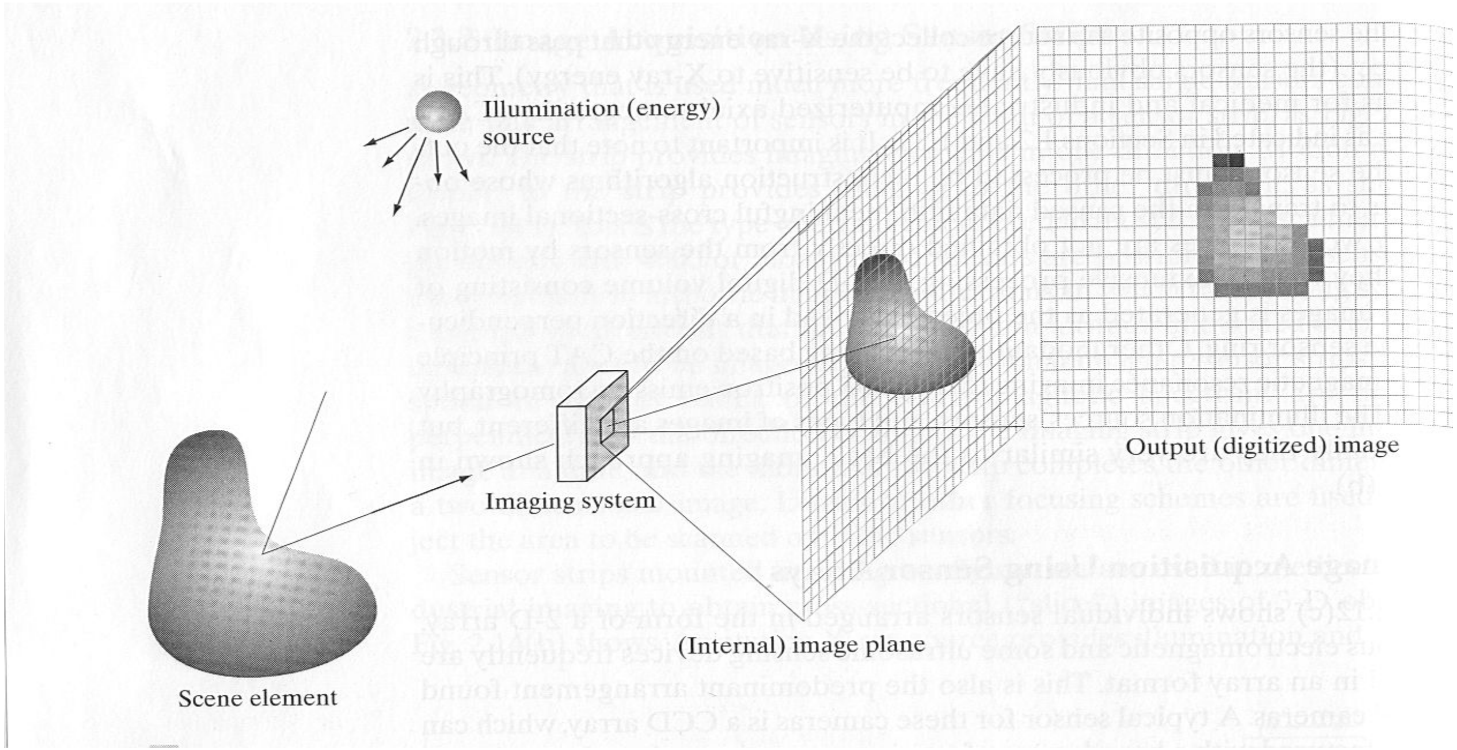


Neighbor pixels are mostly related

Far away pixels are sometimes / somehow related

- Physically related: a *path*
- Contextually related: common *meaning*

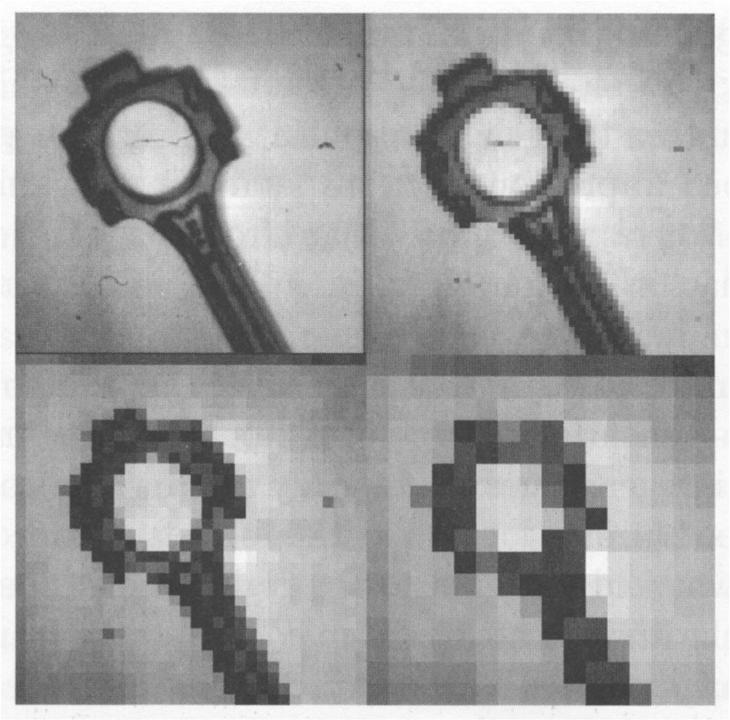
Digital Image Acquisition Process



Quantization → Gray-Level Resolution

Sampling → Spatial Resolution

Example: Quantization vs Sampling



Quantization → from 32 to 4 gray levels all with 256x256 sampling

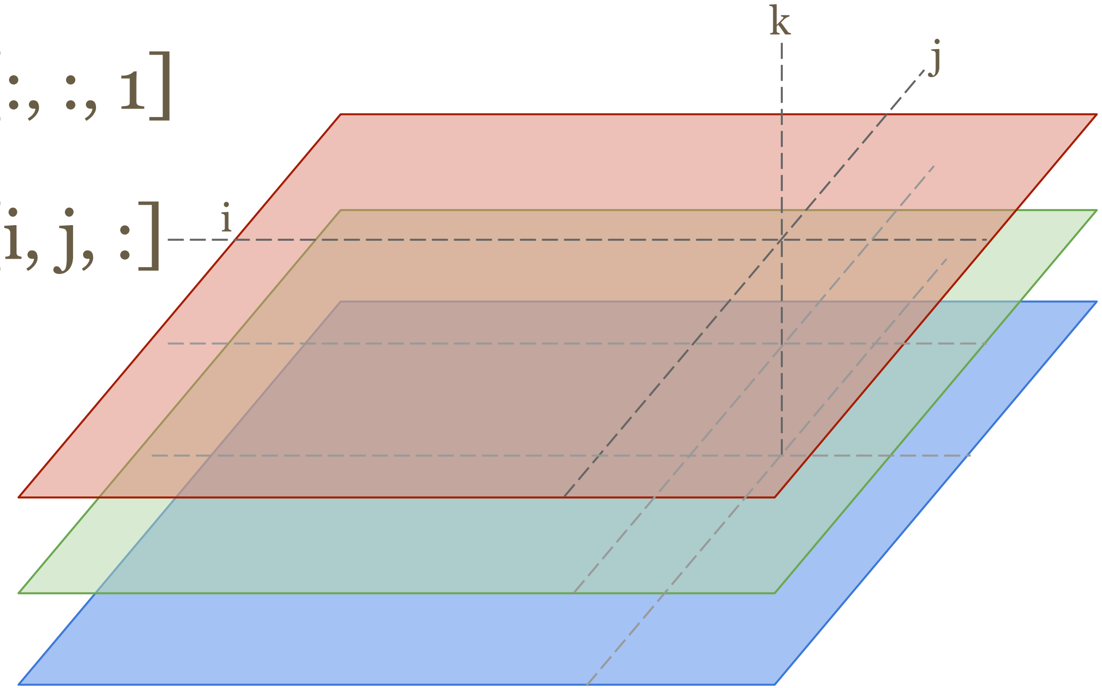
Sampling → 256x256 to 16x16 all with 128 gray levels

Pixels in Images: RGB implied...

Pixels in tensor I: I [i, j, k]

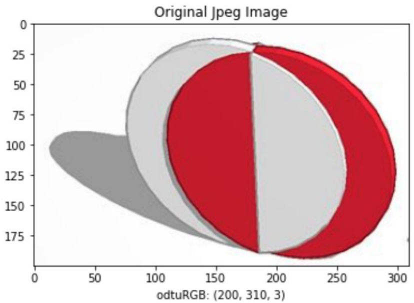
All pixels on Green plane: I [:, :, 1]

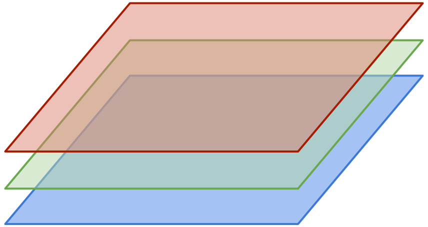
RGV values of pixel (i,j): I [i, j, :]



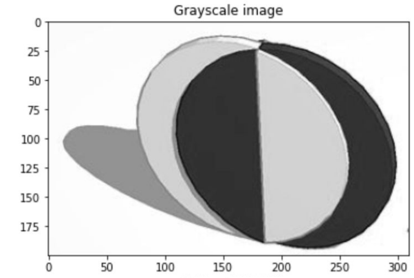
Any plane -> grayscale

Color Depth in Images: Color, Gray, Binary

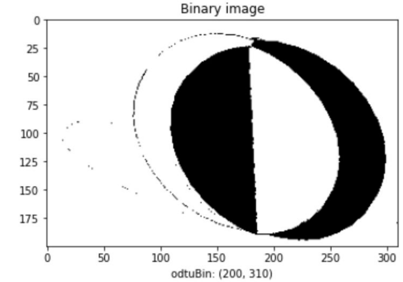


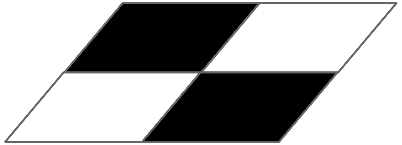


- Color images:
  - has layers such as RGB
  - Typical values: [0-255] or [0.0 - 1.0]



- Gray-scale images:
  - has 1 layer
  - Typical values: [0-255] or [0.0 - 1.0]



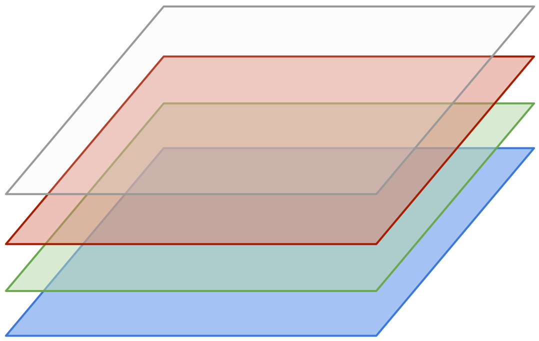


- Binary images
  - just like Gray scale has 1 layer
  - Values are 0 or 1

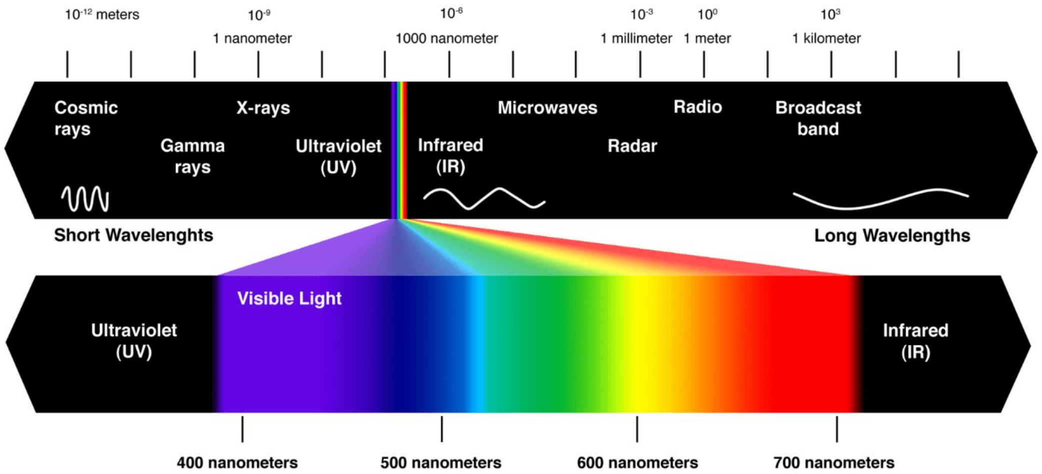
https://www.tinkercad.com/things/htir6tWlgGA-odtu-metu-logo

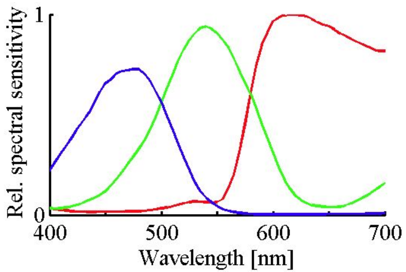
Color Depth in Images: Transparency, Hyperspectral

- Color images with transparency:
  - Typically has an additional layer



RGB response: of some/typical daylight camera*

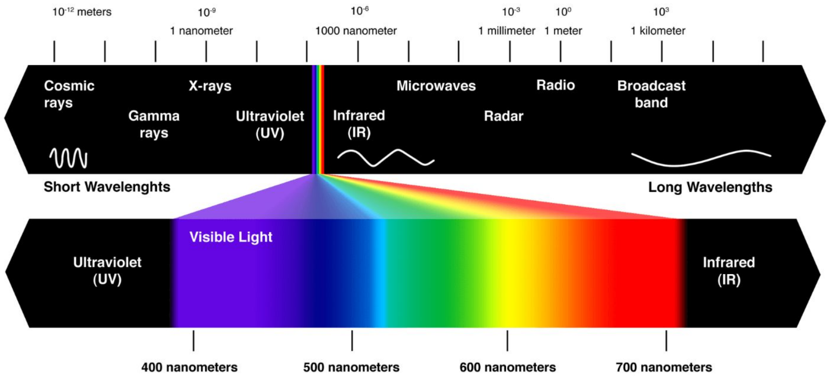


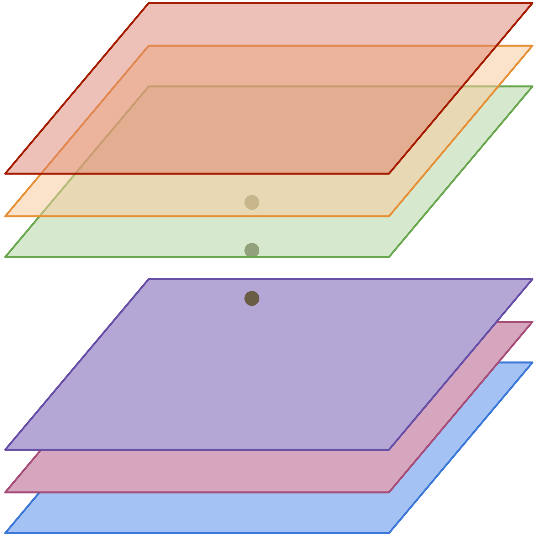


*Dissanayake, Dissanayake & Herath, Vijitha & Godaliyadda, G M Roshan & Ekanayake, Mervyn Parakrama & Bandara, Chaminda & Prabhath, Gode Withanage. (2019). Design of a Multispectral Imaging System for Industrial Applications. 10.13140/RG.2.2.36834.68800

Color Depth in Images: Transparency, Hyperspectral

- Multispectral / Hyperspectral images:
  - has several layer corresponding to different wavelengths
    - Near IR (~700nm - 900nm)
    - Short-Wave IR (SWIR: ~1.7um-2.5um)
    - Mid-Wave IR (MWIR: ~2.7um-5.3um)
    - Long-Wave IR (LWIR: ~ 8um-12um)





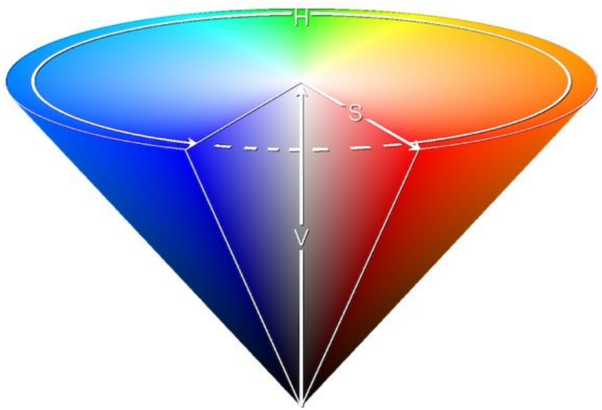
Color Spaces: RGB → Most famous but ?

- a.k.a. Color Formats
- RGB is one of many: CMYK, CIE, HSV, ...
- Different uses, different formats
- HSV / (a.k.a. *HSB*) is more ?
- How about you come up with one? Other do so*

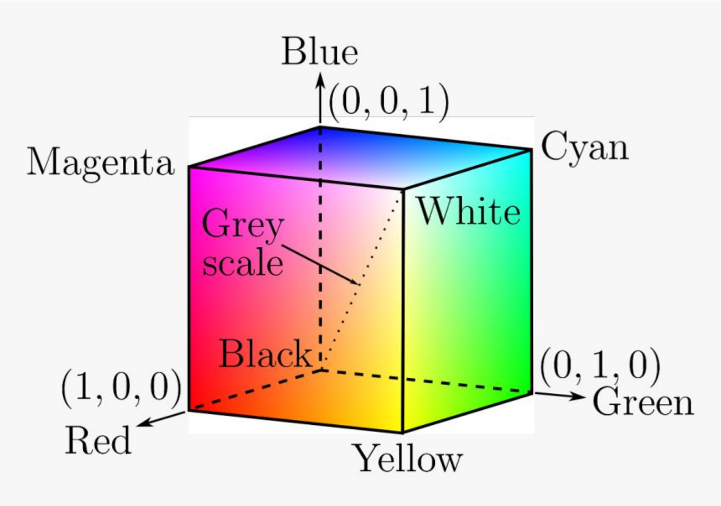
*https://www.cs.harvard.edu/~sjg/papers/cspace.pdf

Color Spaces: RGB vs HSV

When lighting conditions change less play on HSV



HSB Cone. Image courtesy of Wikimedia.



https://www.pngitem.com/middle/ThRhwmb_rgb-color-model-cube-hd-png-download/

Python libraries available

Scikit-Image

OpenCV

PIL

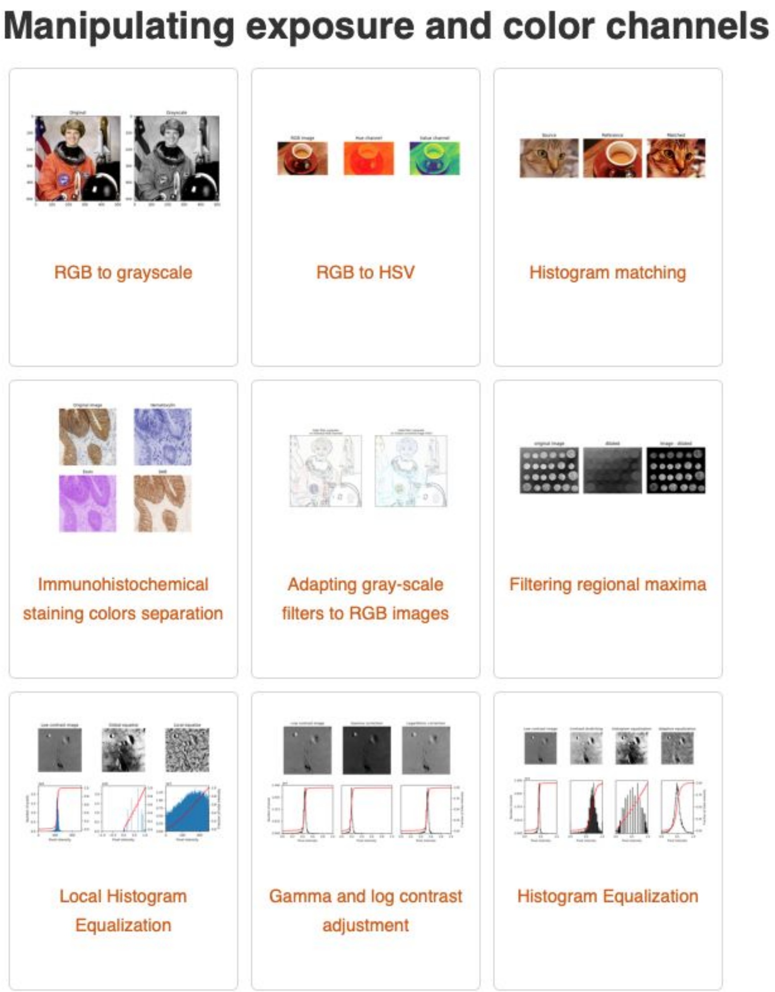
...

Scikit-Image

Check out:

[Examples page]

for more...



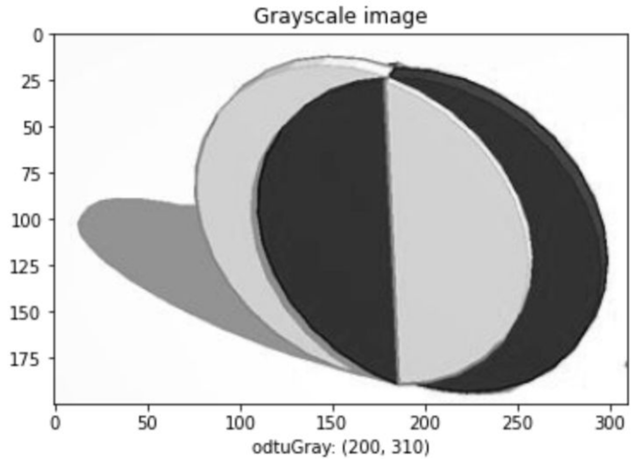
Format change:

- RGB → HSV
  - Linear transformation
  - not unique different transformation in literature
- RGB → Gray
  - Generally a linear transformation
  - not unique different transformation in literature
  - GrayValue = 0.2125 R + 0.7154 G + 0.0721 B [1]
    - Used by CRT phosphors as they better represent human perception of red, green and blue than equal weights
- Gray → Binary
  - Tricky
  - Depends on the objective
  - Manual inspection is useful
  - Automatic thresholding methods exist such as OTSU[2] and many more[3]

1- https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_rgb_to_gray.html#id2
2- https://en.wikipedia.org/wiki/Otsu%27s_method
3- https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.threshold_isodata

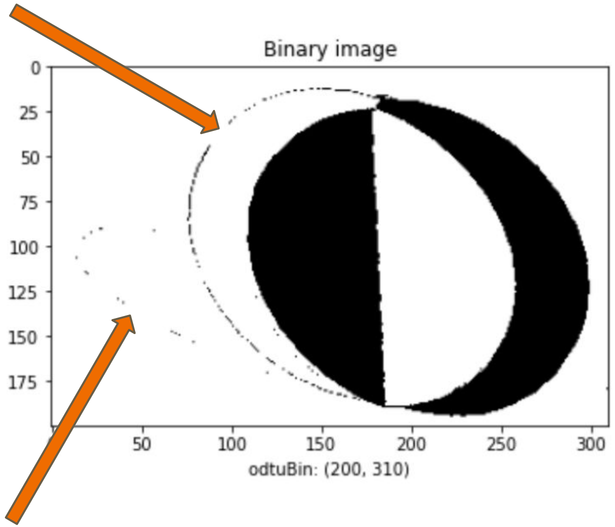
Binary images: Thresholding

- **Proper thresholding** improves detection performance



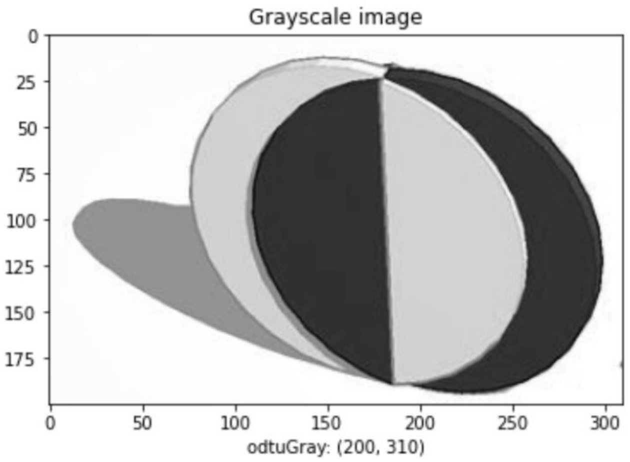
?

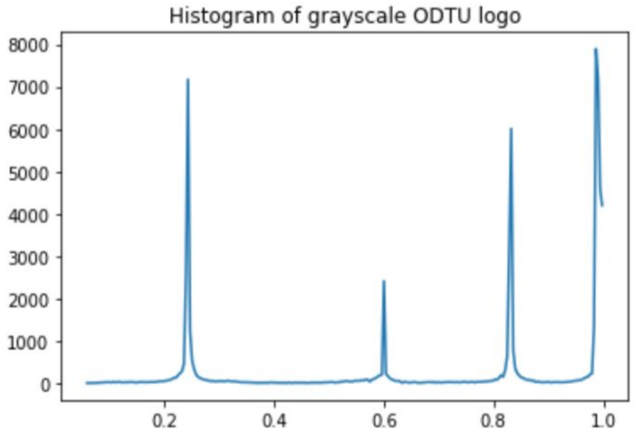
→



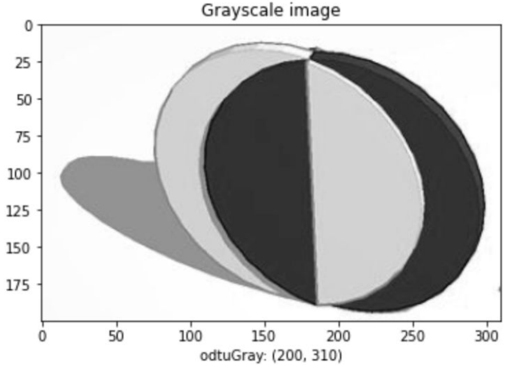
Histograms:

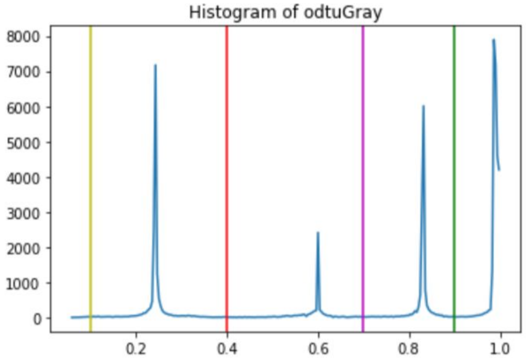
A way of analyzing distribution of data

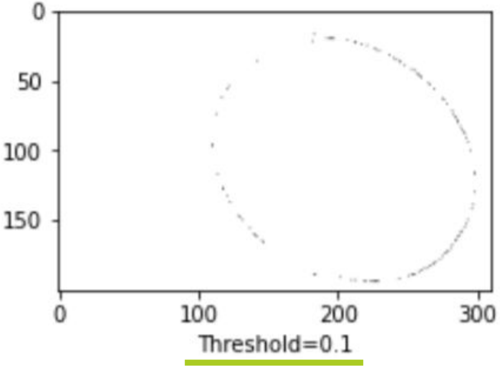


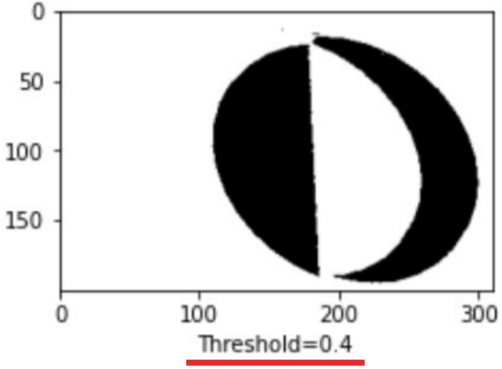


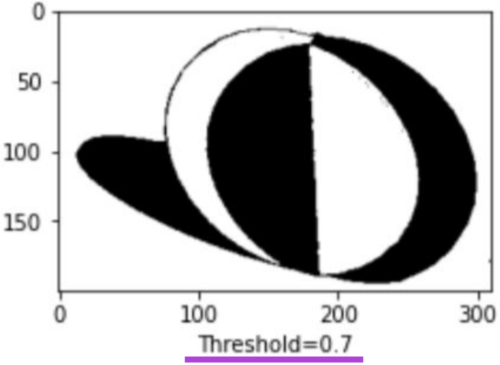
Histograms: Manual Thresholding a delicate balance

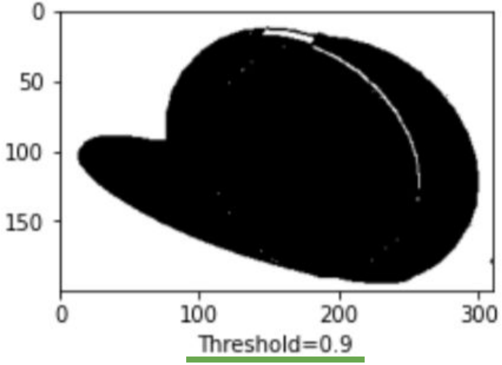




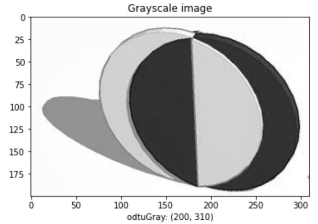


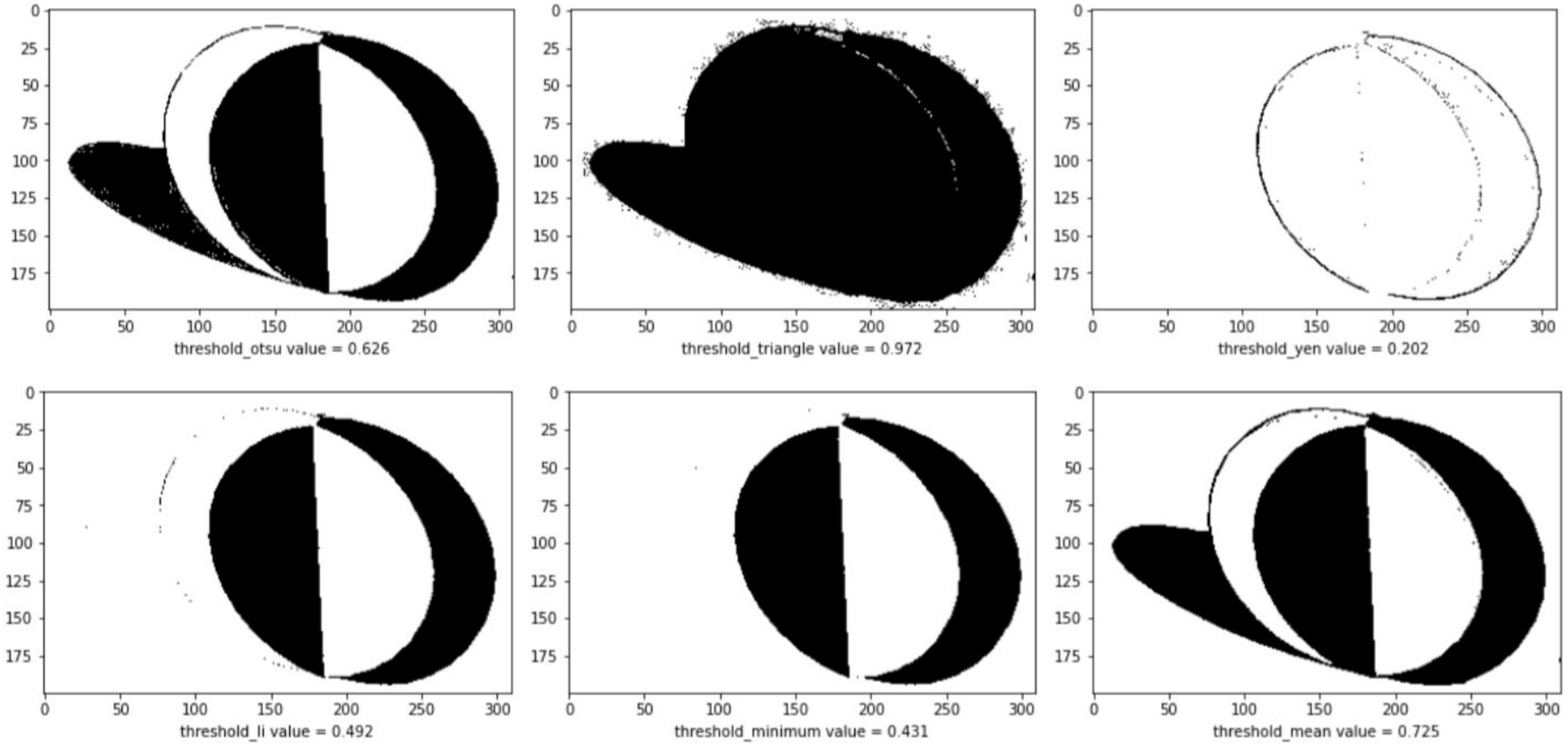






Histograms: Automatic Thresholding





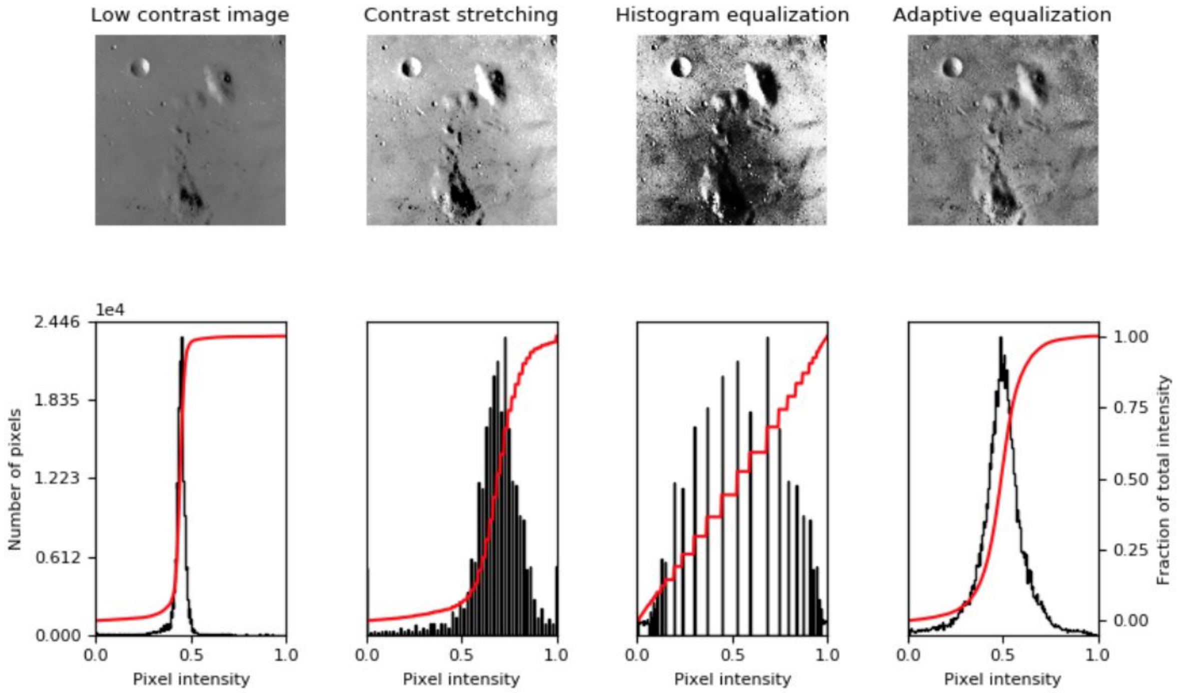
Histograms: Proper Choice

Doesn't it sound like a clustering problem in a way? Where there are 2 clusters...

😱 How about a **thresholding method** that **understands what's in the image**?

Histogram: Beyond Thresholding

Histogram enhancement

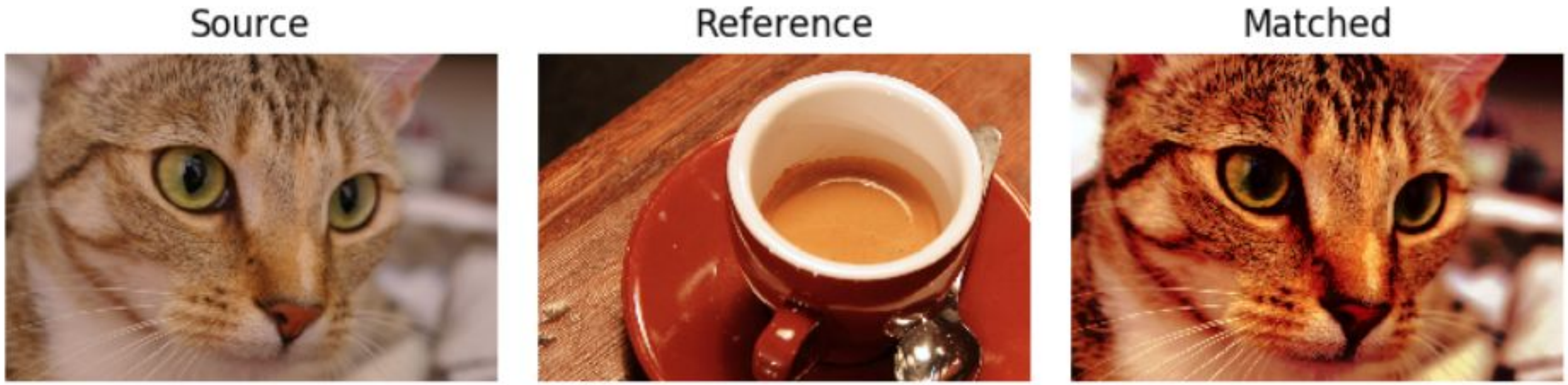


https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_equalize.html#sphx-glr-auto-examples-color-exposure-plot-equalize-py

Histogram: Beyond Thresholding

Histogram matching:

- when groups of images are analyzed
- extends to histogram equalization



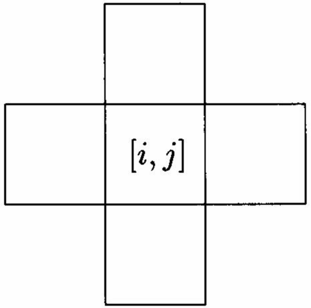
https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_histogram_matching.html#sphx-glr-auto-examples-color-exposure-plot-histogram-matching-py

Basic Definitions: used in dealing with images

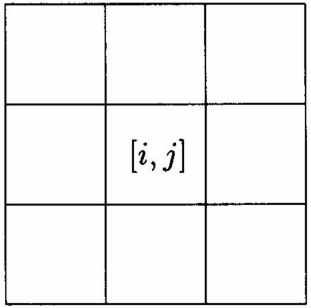
- Neighbors
- Path
- Foreground, Background
- Connectivity
- Connected Components
- Boundary, Interior, Surrounds

Neighbors: 4 or 8

4-neighbors $[i+1, j]$, $[i-1, j]$, $[i, j+1]$, $[i, j-1]$



8-neighbors $[i+1, j+1]$, $[i+1, j-1]$, $[i-1, j+1]$, $[i-1, j-1]$ plus all of the 4-neighbors



Path:

- A *path* from the pixel P_o to pixel P_n is a sequence of pixel indices (i_o,j_o), (i_1,j_1), (i_2,j_2), ..., (i_n,j_n) such that the pixel at P_k is a neighbor of the pixel at P_{k+1} for all k with $0 \leq k \leq n-1$

- If a 4-connection is used the path is a *4-path*, and it is an *8-path* in case 8-connection is used

Foreground vs Background:

- Set of all 1 pixels in an image is called the foreground and denoted by S.

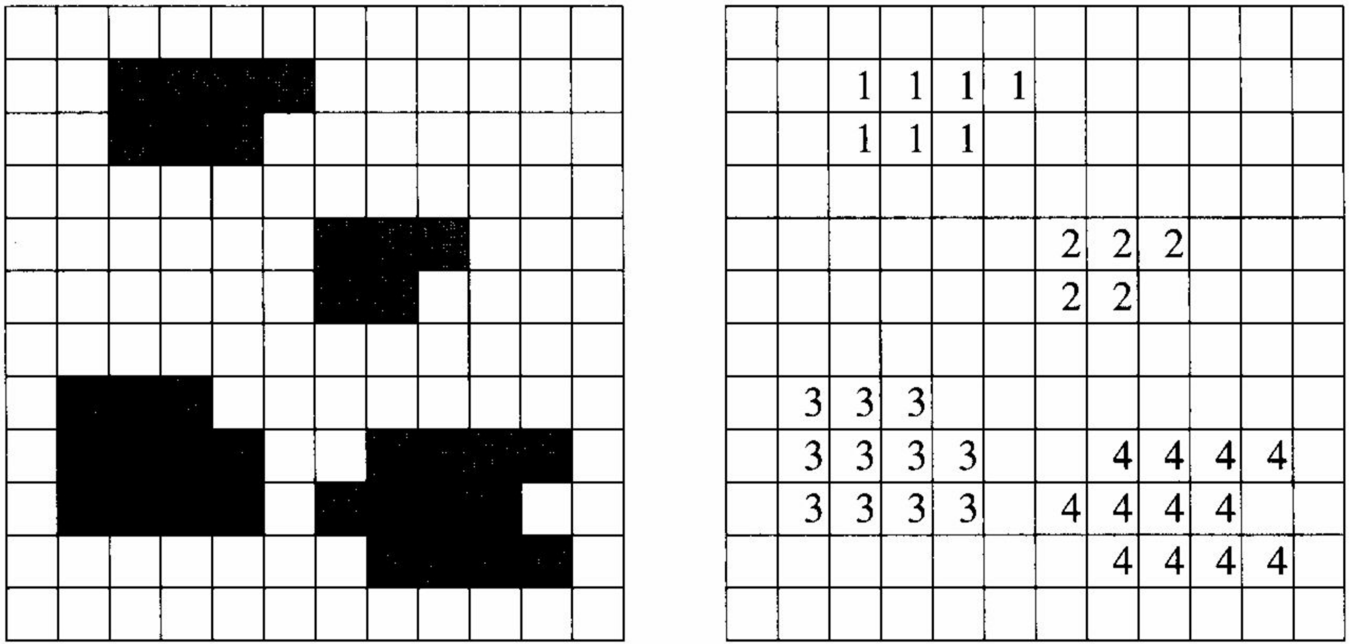
- Set of all 0 pixels in an image is called the *background* and denoted by $\overline{S}$.

Connectivity:

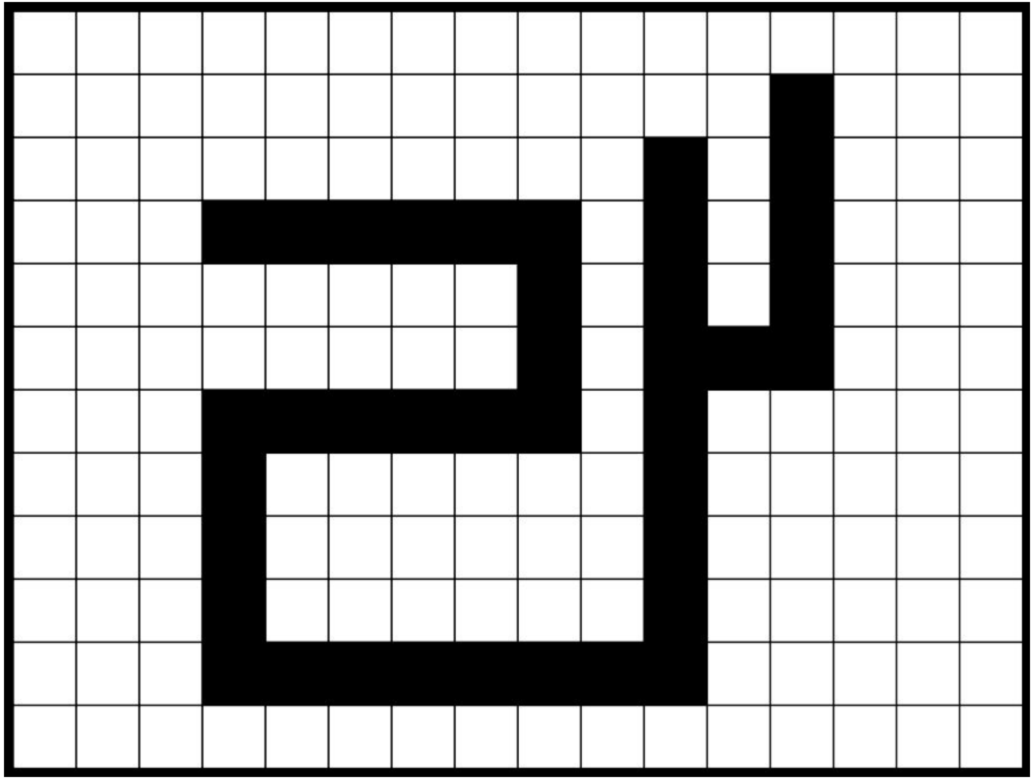
- A pixel *p* in S is said to be *connected* to pixel q in S if there is a path from p to q.

- Connectivity is:
  - Reflexive: p is connected to p
  - Commutative: if p is connected to q, then q is connected to p
  - Transitive: If p is connected to q and q is connected to r, then p is connected to r

Connected Component Labeling

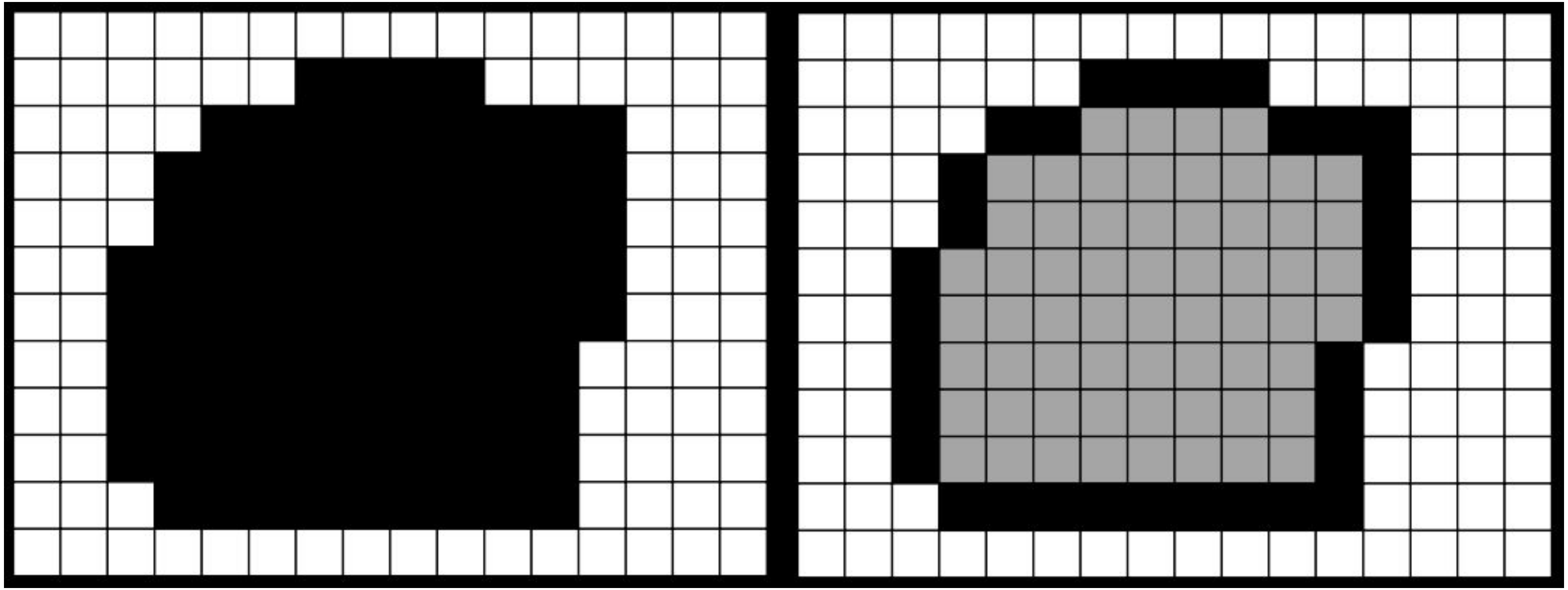


Connected Component Labeling



Boundary, Interior, Surrounds

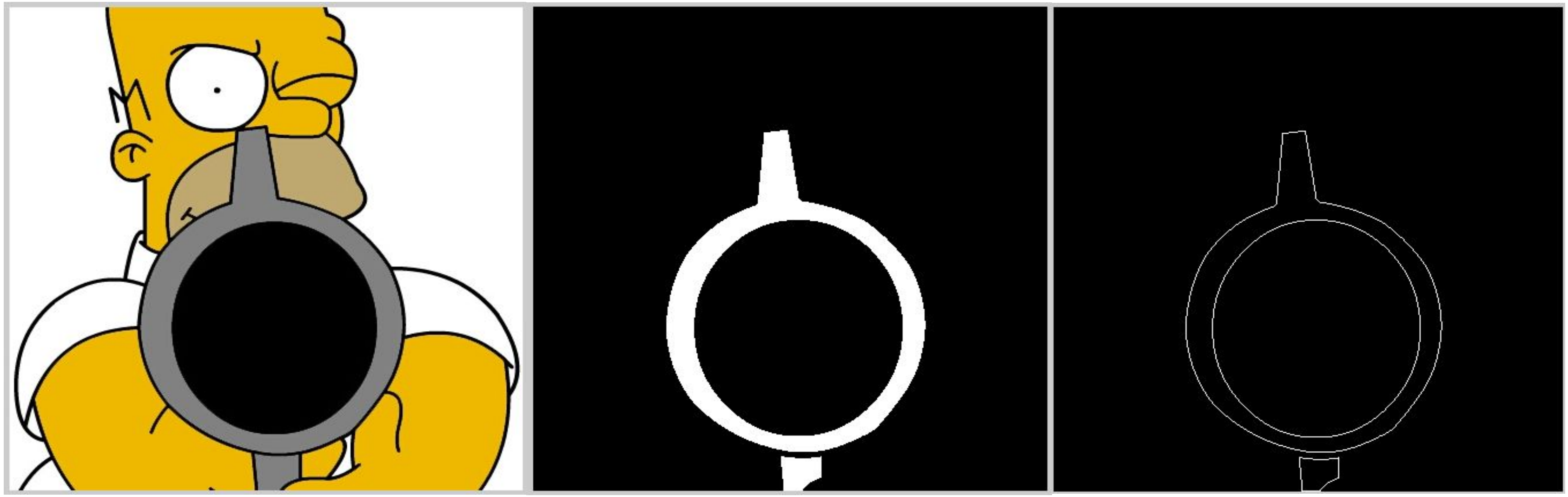
Neighborhood?



Properties

- Size $\rightarrow$ Area, perimeter
- Position
- Orientation
- Compactness
- Euler number
- Distance measures
- Medial Axis

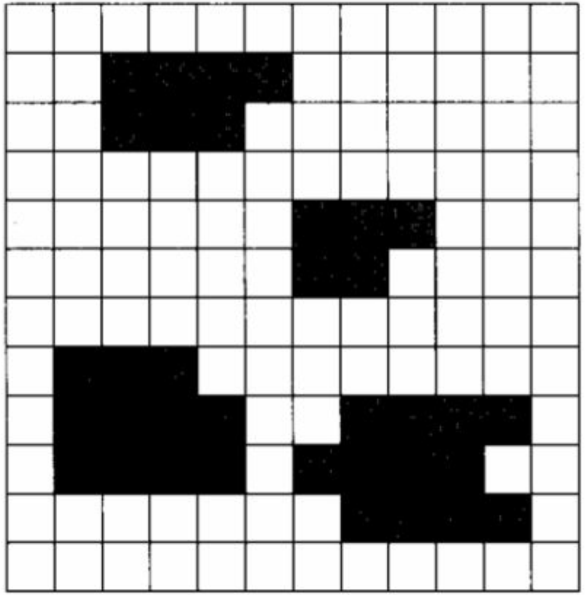
Size: Area, Perimeter, …



Area = 15132 Perimeter = 1414

Position: Center of mass of each component

Orientation: Axis that minimized second moment

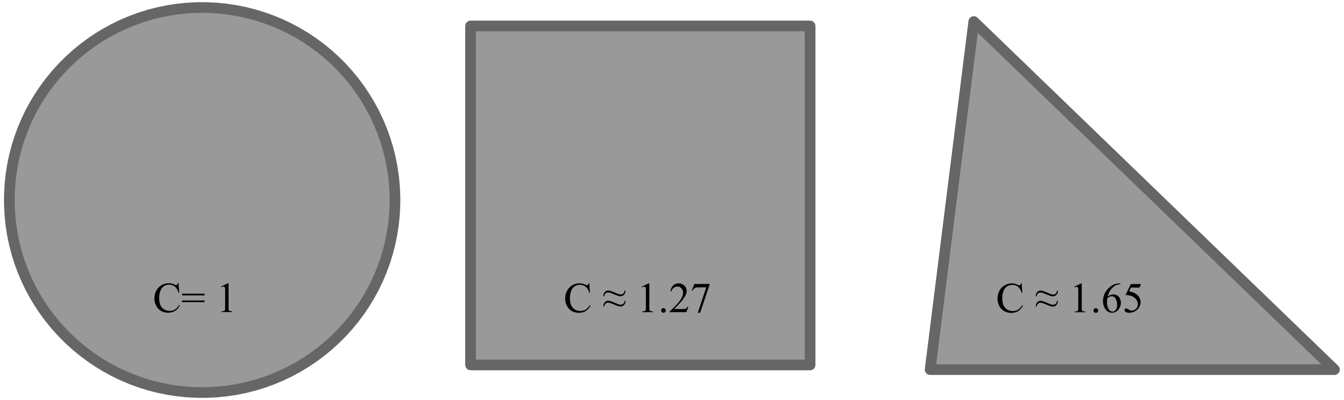


Compactness:

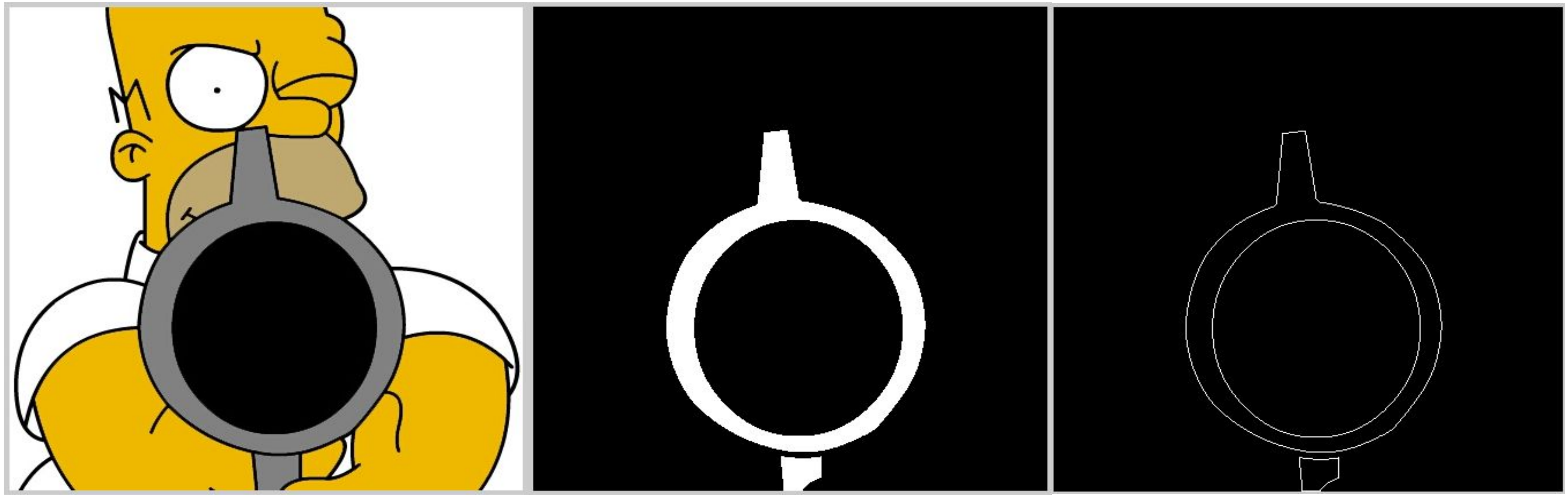
- Compactness of a continuous geometric figure is measured by the isoperimetric inequality:

$$C = (P^2/A)\,/4\pi \geq 1$$

where P is the perimeter and A is the area of the object



Compactness: C ≈ 132



Area = 15132   Perimeter = 1414

Euler Number:

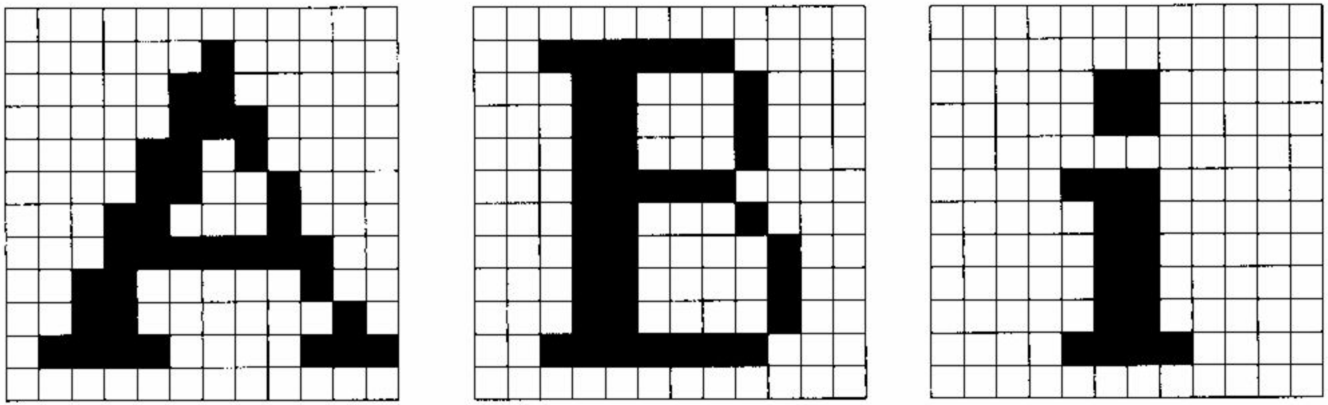
- Euler number is the number of components minus the number of holes on the image:

$$E = C - H$$

- Neighborhood definition is important for both the back and the foreground
- Invariant to translation, rotation, and scaling!

Euler Number: $E = C - H$

Background N4, foreground N8



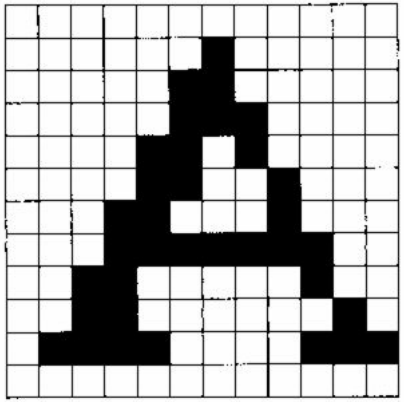
C =?
H =?

C =?
H =?

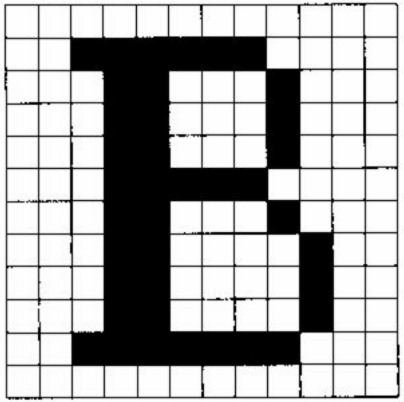
C =?
H =?

Euler Number: $E = C - H$

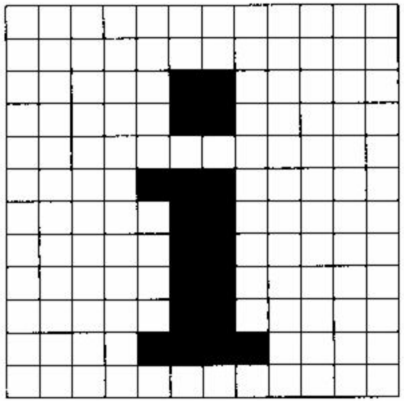
Background N4, foreground N8



C =1
H =1



C =1
H =2



C =2
H =0

Distance:

Euclidean

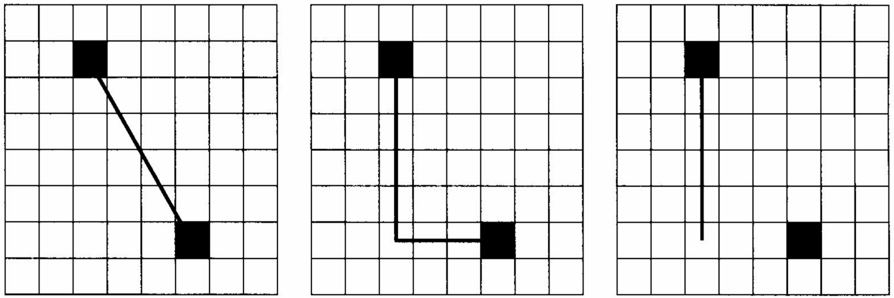
$$d_{\text{Euclidean}}([i_1, j_1], [i_2, j_2]) = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2}$$

City-block

$$d_{\text{city}} = |i_1 - i_2| + |j_1 - j_2|$$

Chessboard

$$d_{\text{chess}} = \max(|i_1 - i_2|, |j_1 - j_2|)$$



Euclidean distance:

			3			
	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$	
	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$	
3	2	1	0	1	2	3
	$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$	
	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$	
			3			

City-block distance:

			3			
		3	2	3		
	3	2	1	2	3	
3	2	1	0	1	2	3
	3	2	1	2	3	
		3	2	3		
			3			

Chessboard distance:

3	3	3	3	3	3	3
3	2	2	2	2	2	3
3	2	1	1	1	2	3
3	2	1	0	1	2	3
3	2	1	1	1	2	3
3	2	2	2	2	2	3
3	3	3	3	3	3	3

Medial Axis

- If the distance $d((i,j), \overline{S})$ for the pixel (i,j) in S to $\overline{S}$ is locally maximum i.e.

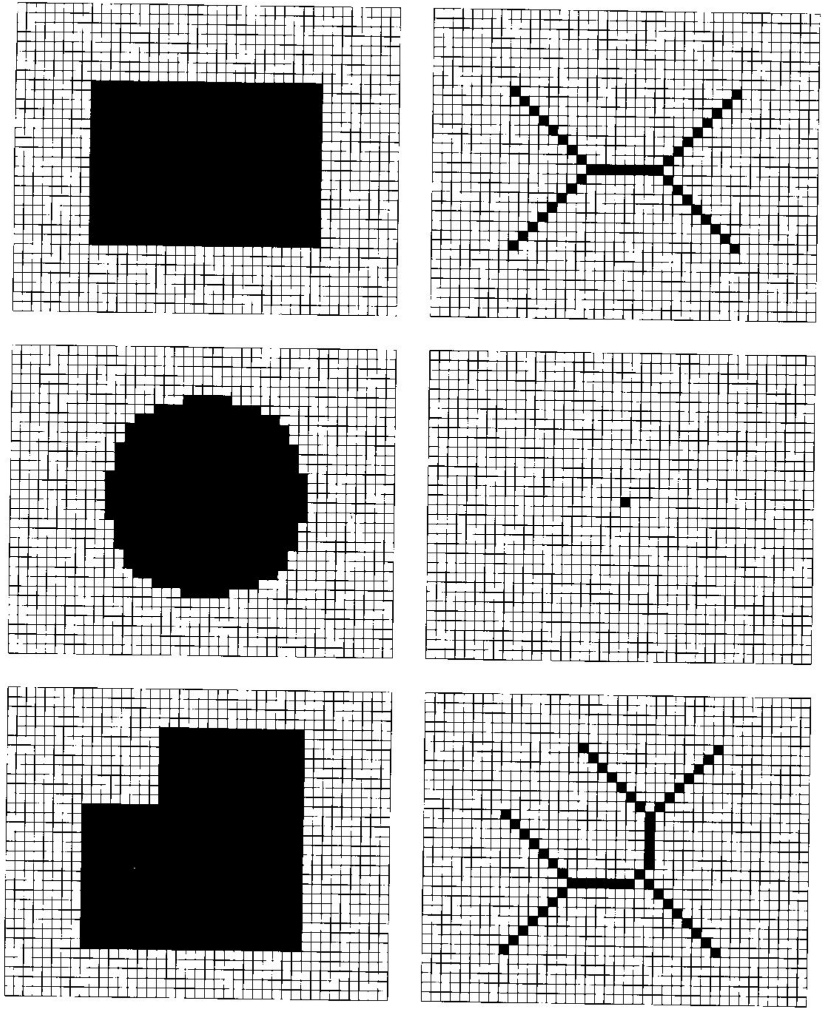
  $$d(\,(i,j), \overline{S}) \geq d(\,(u,v), \overline{S})$$

for all pixels (u,v) in the neighborhood of (i,j), then pixel (i,j) is on the medial axis.

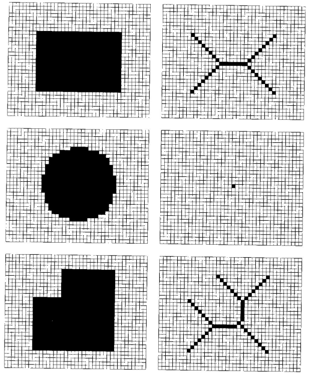
(Recall that: $\overline{S}$ is the background and S is the foreground)

- Medial axis has been used for compact representation of objects.
- Also check out **scikit-image** : ***skeletonize***

Medial Axis: Examples



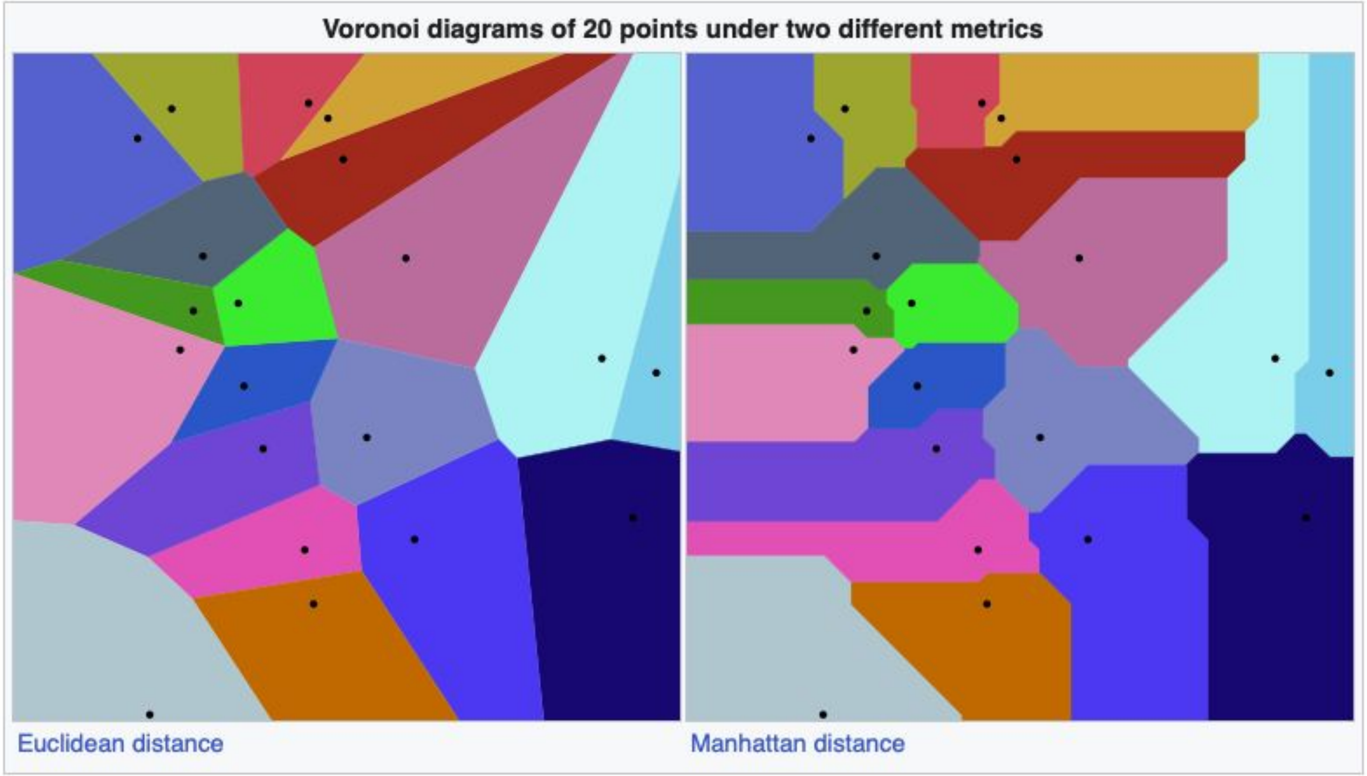
Recalls or extends to:
Voronoi graphs



Observe that:

Voronoi graphs has a **k-means** flavor

Voronoi: Distance Metrics



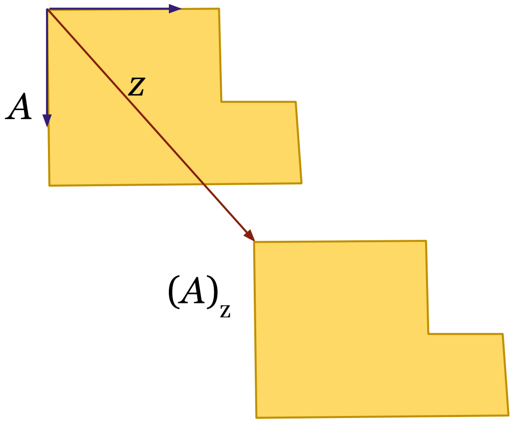
https://en.wikipedia.org/wiki/Voronoi_diagram

Dilation & Erosion

- Fundamental to morphological processing of binary images
- Many other algorithms are based on these two primitives

Dilation & Erosion: Translation and Reflection

On image $\mathbf{I}$, A & B are foreground objects:

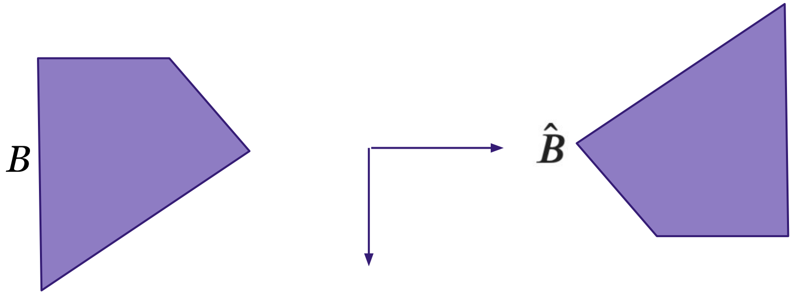


- Translation

  $(A)_z = \{c \mid c = a + z, \text{ for } a \in A\}$

- Reflection

  $\hat{B} = \{w \mid w = -b, \text{ for } b \in B\}$



Erosion:

erosion noun

Save Word

ero·sion | \ i-ˈrō-zhən \

Definition of *erosion*

1 a : the action or process of eroding

b : the state of being eroded

2 : an instance or product of erosion

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$

A: foreground on the image to be eroded

B: structuring element (i.e. `selem` in `skimage`)

Erosion → contains all points where the translated `selem` is fully contained by the image foreground (i.e. $(B)_z$ is completely inside A)

Analogy → using an eraser going over the perimeter of the foreground components

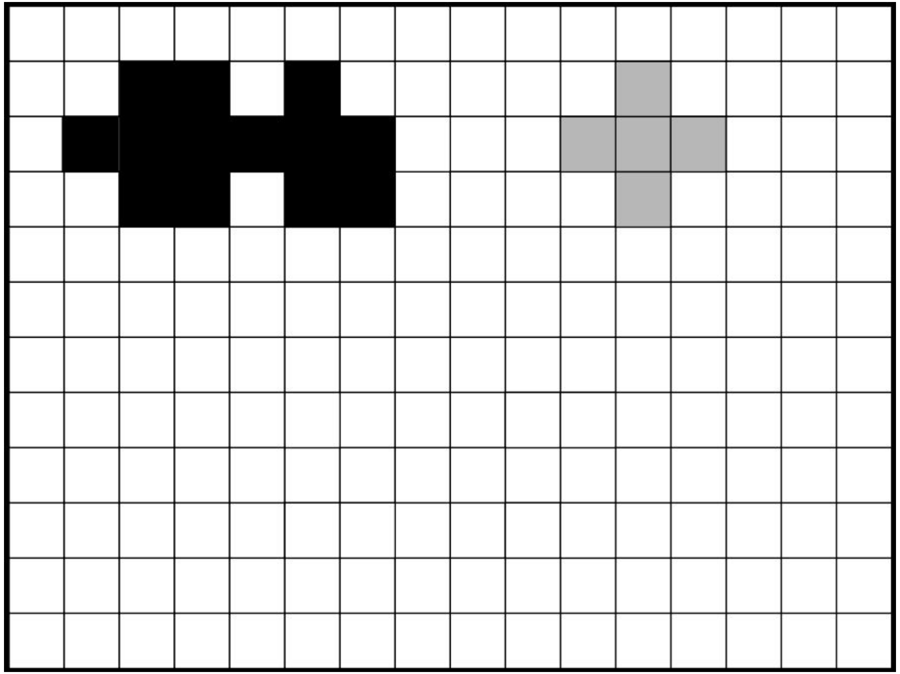
Erosion: the process in practice

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$

1. Hold `selem` at its center
2. Put it on the top left of the image
3. If `selem` is *fully* contained by the foreground (1s of `selem` match the 1s that of the image), mark the pixel that coincides with the center of the `selem` to belong to the eroded image
4. Systematically move `selem` pixel by pixel all over the *foreground*

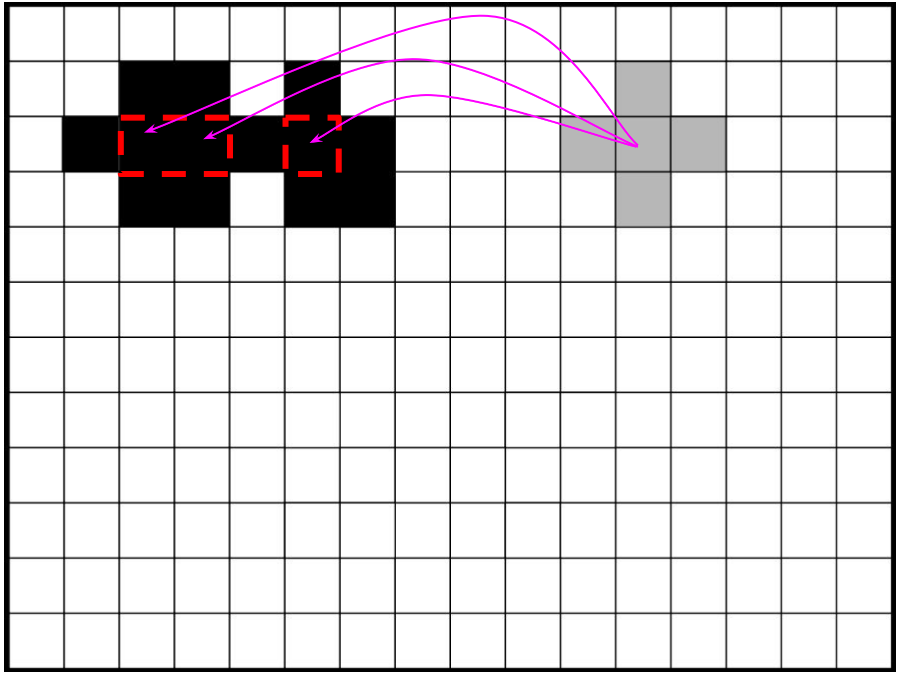
Erosion: the process in practice

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$



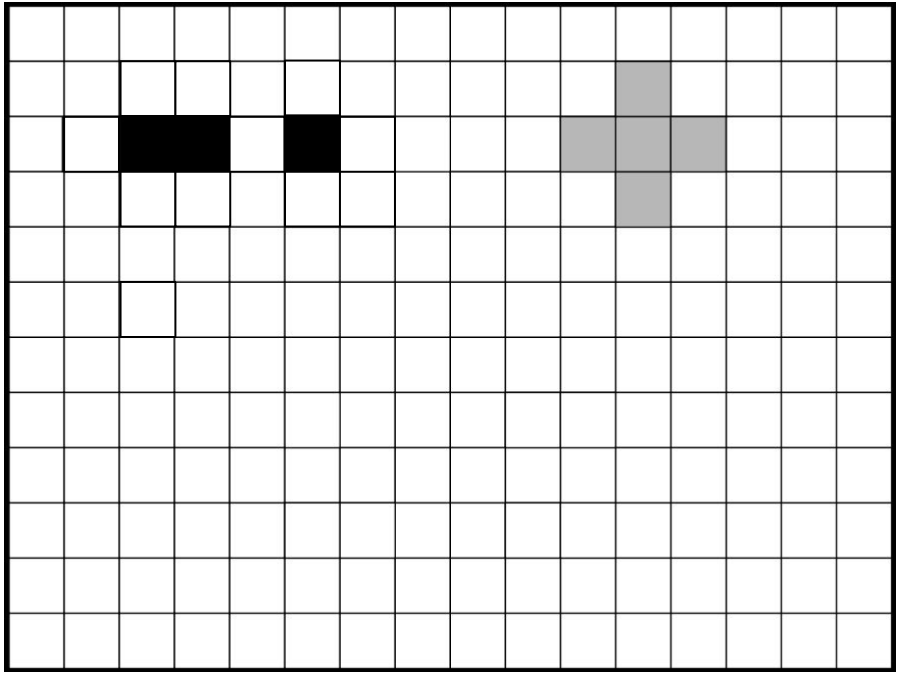
Erosion: the process in practice

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$



Erosion: the process in practice

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$



Dilation:

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$

A: foreground on the image to be dilated

B: structuring element

dilation noun

Save Word

di·la·tion | \ dī-ˈlā-shən \

Definition of *dilation*

1 : the act or action of dilating : the state of being dilated : EXPANSION, DILATATION

2 : the action of stretching or enlarging an organ or part of the body

Dilation → contains all points where the reflection of the translated `selem` has any intersection (even 1 pixel) with the image foreground

Analogy → painting over the perimeter of the foreground components with a Brush (i.e. the structuring element B)

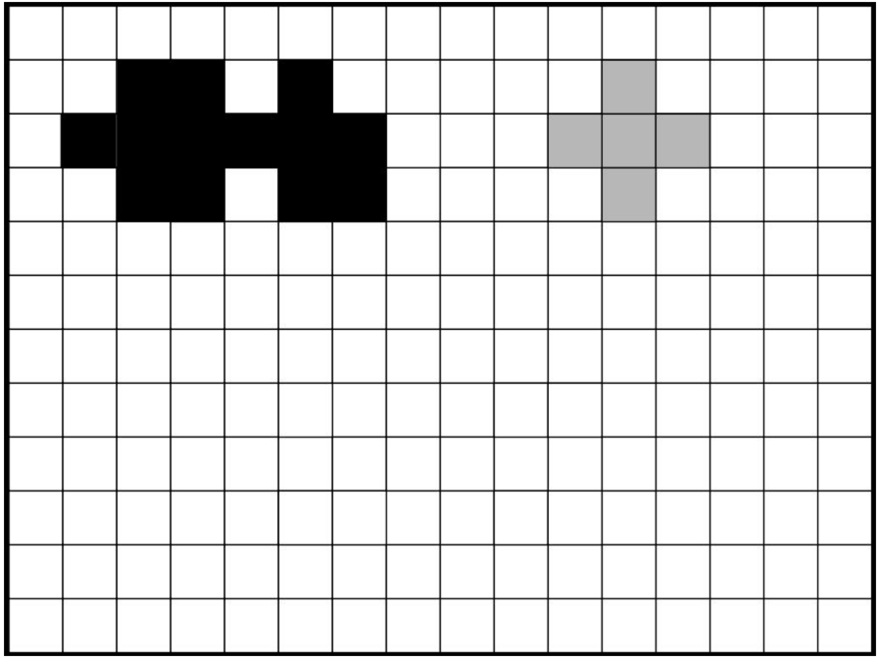
Dilation: the process in practice

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$

1. Hold the reflection of `selem` at its center
2. Put it on the top left of the image
3. If `selem` has any overlap with the foreground, mark the pixel that coincides with the center of the `selem` to belong to the dilated image
4. Systematically move `selem` pixel by pixel all over the image

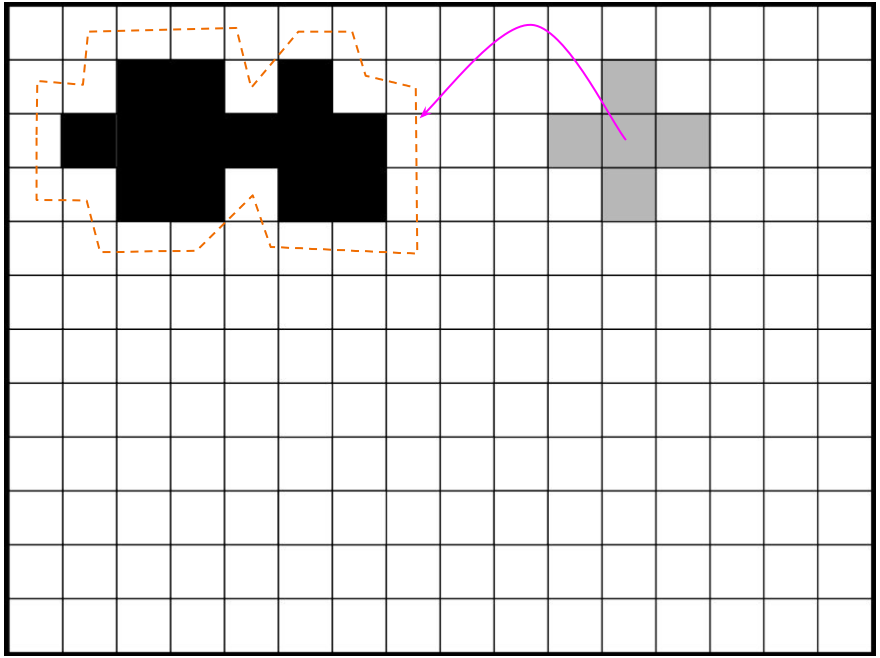
Dilation: the process in practice

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$



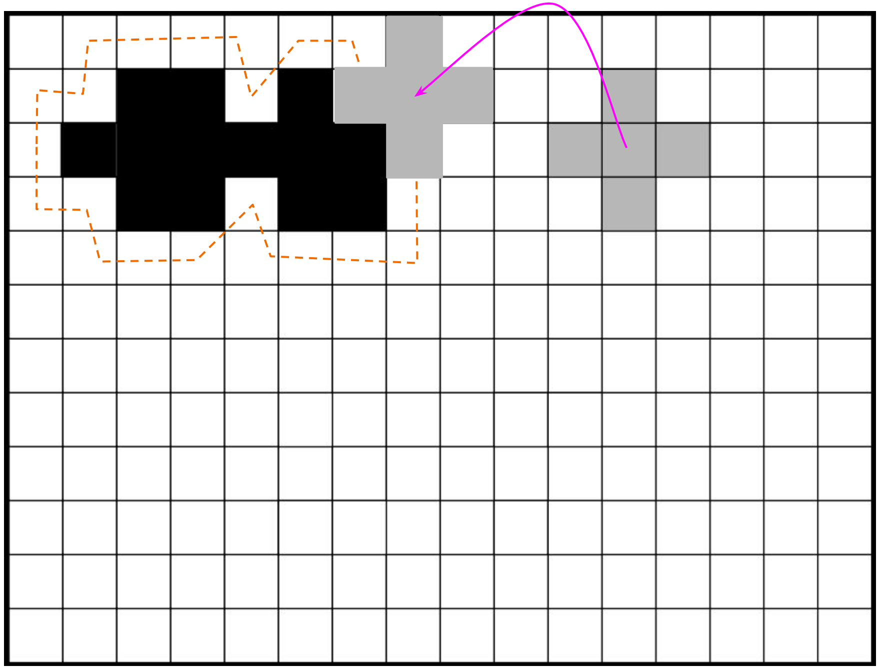
Dilation: the process in practice

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$



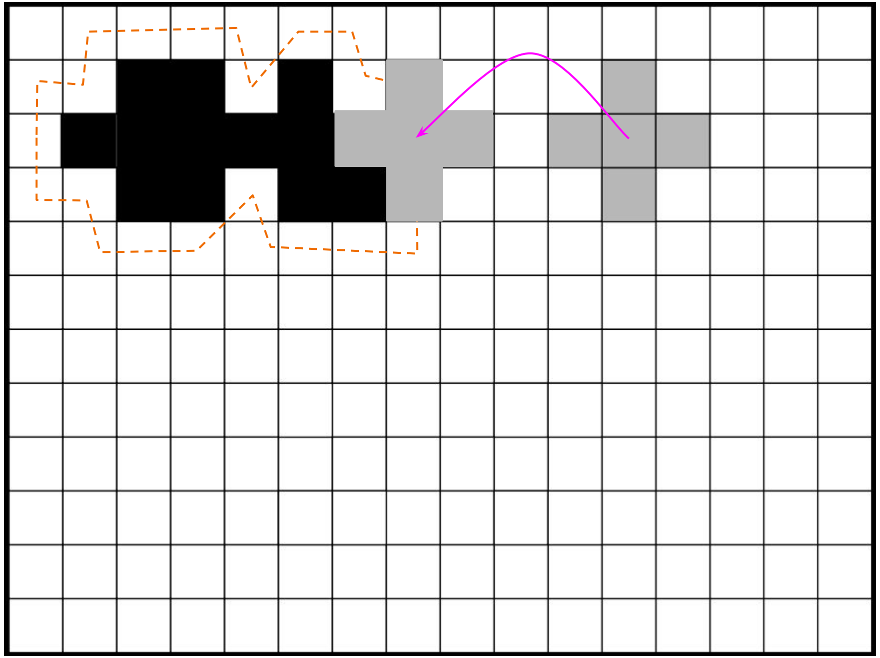
Dilation: the process in practice

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$



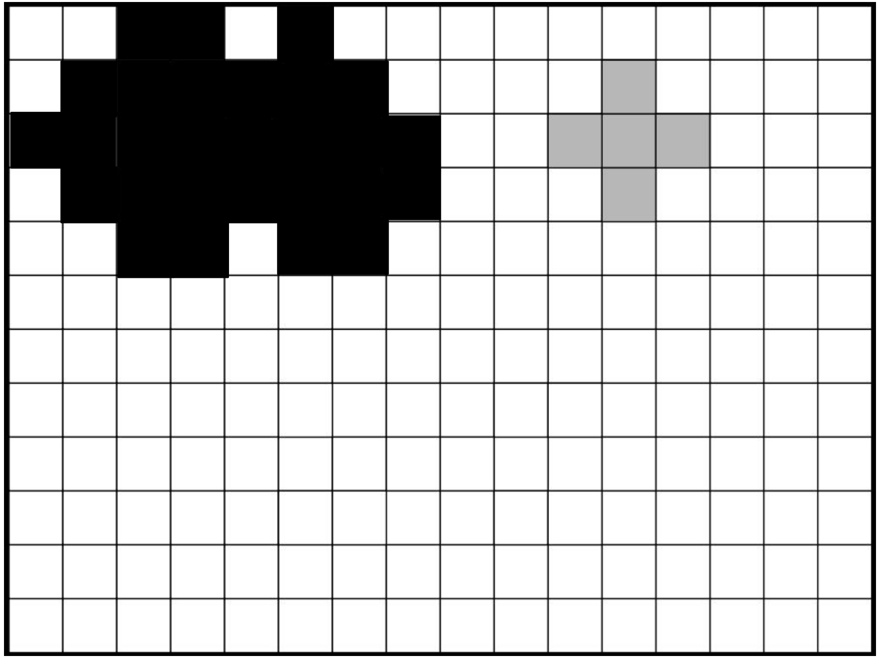
Dilation: the process in practice

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$



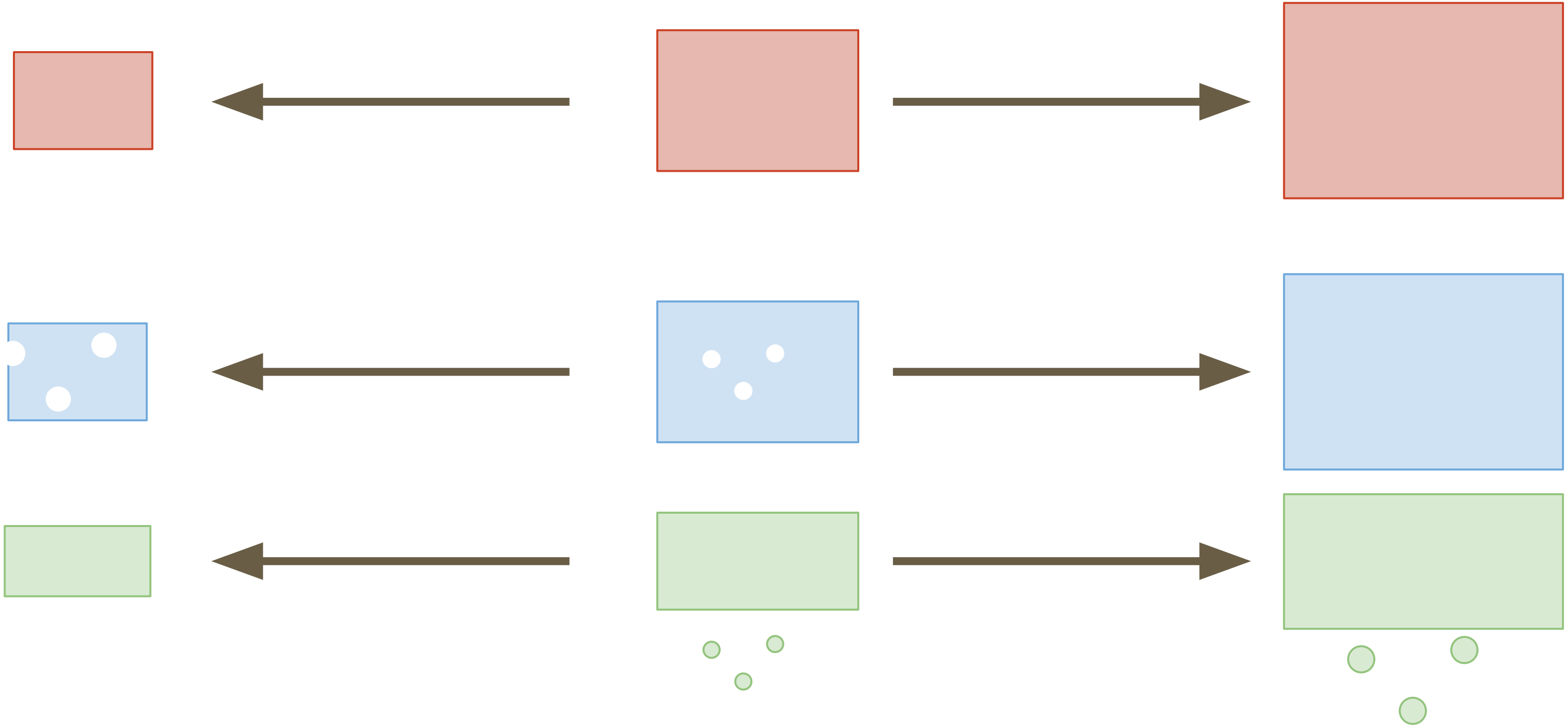
Dilation: the process in practice

$$A \oplus B = \{z \mid (\hat{B})_z \bigcap A \neq \emptyset\}$$



Erosion & Dilation

Erosion Original Image Dilation



Opening & Closing an Image

Opening A: $A \circ B = (A \ominus B) \oplus B$

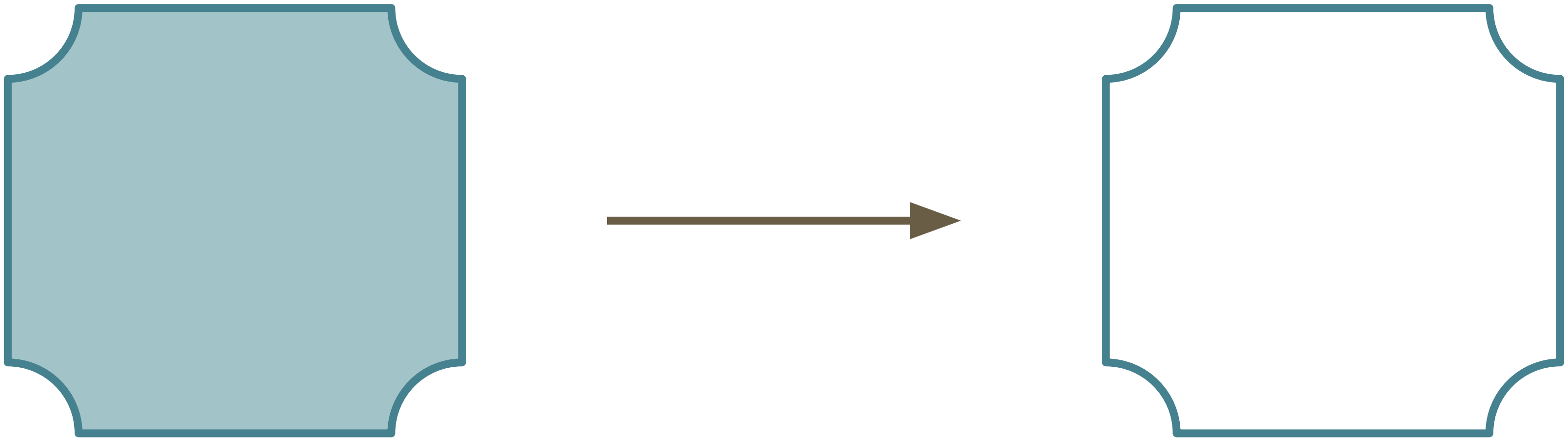
- Erode first, then dilate with B
- Get rid of tiny foreground components
- Separate weakly connected components

Closing A: $A \bullet B = (A \oplus B) \ominus B$

- Dilate first, then erode with B
- Get rid of holes and small irregularities on the perimeter
- close gaps between components

Boundary Extraction:

How to extract the boundary?



Images multiplied: say what?

Hadamard (a.k.a Schur) product: $\mathbf{A} \circ \mathbf{B}$

Yes even the simplest things/ ideas have names as usual

Note that this is the default multiplication for numpy, pytorch etc for matrices of equal size

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\, b_{11} & a_{12}\, b_{12} & a_{13}\, b_{13} \\ a_{21}\, b_{21} & a_{22}\, b_{22} & a_{23}\, b_{23} \\ a_{31}\, b_{31} & a_{32}\, b_{32} & a_{33}\, b_{33} \end{bmatrix}$$

Images multiplied: come again **Hadamard product**

$\mathbf{A}$	$\mathbf{B}$	$\mathbf{A} \circ \mathbf{B}$	$\sum(\mathbf{A} \circ \mathbf{B})$
$\begin{bmatrix} 1 & 5 & 10 \\ 1 & 2 & 0 \\ 3 & 0 & 5 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 5 & 10 \\ 1 & 2 & 0 \\ 3 & 0 & 5 \end{bmatrix}$	$\boxed{\mathbf{27}}$

What is the meaning of $\sum(\mathbf{A} \circ \mathbf{B})$?

Images multiplied: sum of Hadamard product

A

1	5	10
1	$\boxed{2}$	0
3	0	5

B

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$\mathbf{A \circ B}$

1/9	5/9	10/9
1/9	2/9	0/9
3/9	0/9	5/9

$\mathbf{\sum(A \circ B)}$

3

What is the **meaning of** $\mathbf{\sum(A \circ B)}$?

Why are all neighbors equally important?

Images multiplied: come again Hadamard product

A | **B** | **A ∘ B** | **∑(A ∘ B)**

1	5	10
1	2	0
3	0	5

1/9	5/9	1/9
5/9	10/9	5/9
1/9	5/9	1/9

1/9	25/9	10/9
5/9	20/9	0/9
3/9	0/9	5/9

7.66

What is the meaning of $\sum(\mathbf{A} \circ \mathbf{B})$?

Note that $\rightarrow \sum \mathbf{B} = 34 / 9$

Images multiplied: come again Hadamard product

$\mathbf{A}$		
1	5	10
1	2	0
3	0	5

$\mathbf{B}$		
1/34	5/34	1/34
5/34	10/34	5/34
1/34	5/34	1/34

$\mathbf{A} \circ \mathbf{B}$		
1/34	25/34	10/34
5/34	20/34	0/34
3/34	0/34	5/34

$\sum(\mathbf{A} \circ \mathbf{B})$

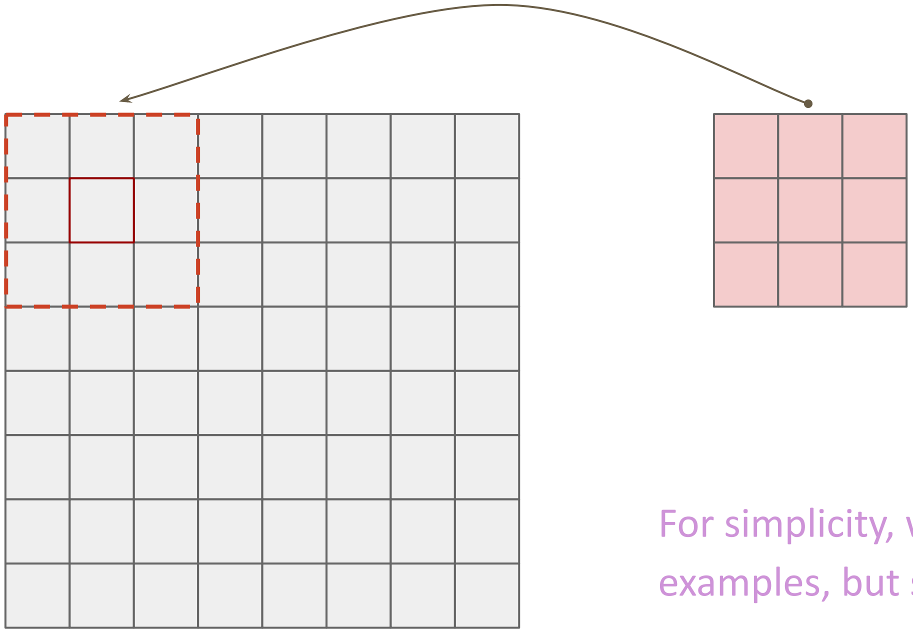
2.03

What is the meaning of $\sum(\mathbf{A} \circ \mathbf{B})$?

$$\sum \mathbf{B} = 1$$

Convolution: very similar to cross correlation

Recall: reflection from morphological operators $\hat{B} = \{w \mid w = -b, \text{for } b \in B\}$



I : image

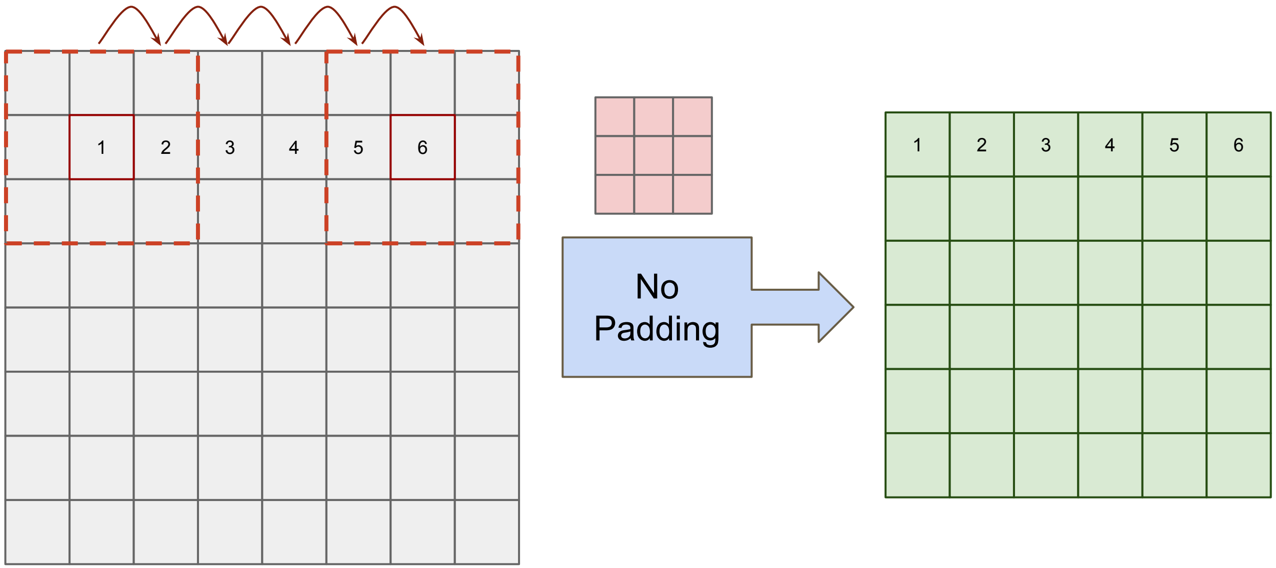
B : kernel

Convolution: Reflection of the kernel B travels around the image I

For simplicity, we will **not** incorporate **reflection** in the following examples, but sample codes demonstrate the actual usage

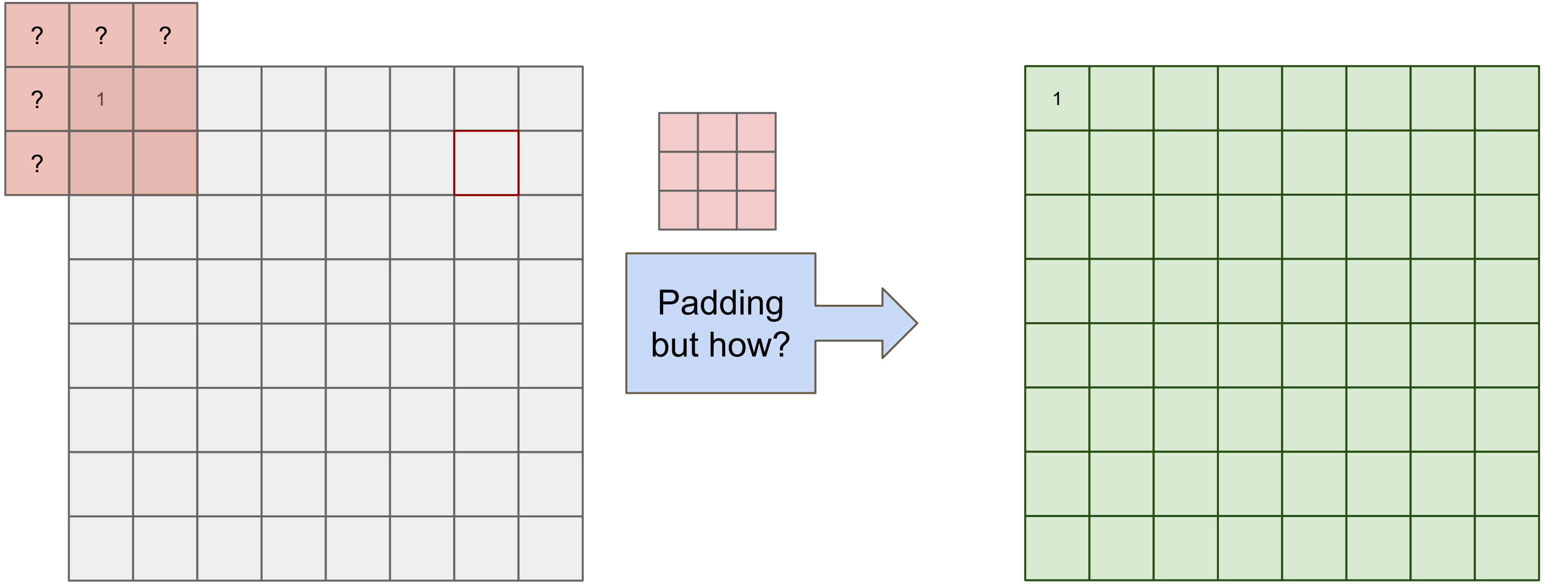
Convolution dilemma: pad or shrink

What happens when center pixel of B is on the edge of the I ?



Convolution dilemma: pad or shrink

What happens when center pixel of B is on the edge of the I ?

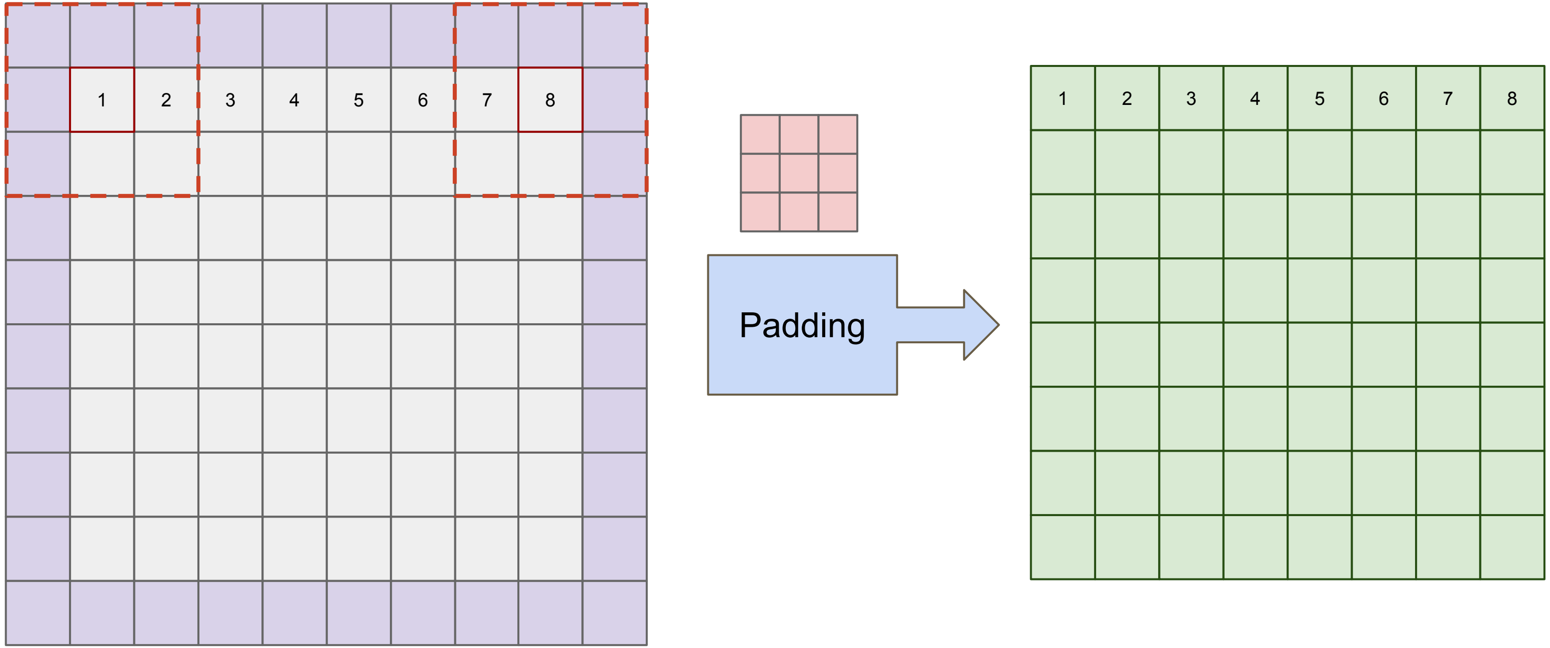


Convolution dilemma: pad not to shrink

Note that padding depends on the size of the kernel

Kernel size: in general a center pixel (i.e. kernel fully contained by the image)

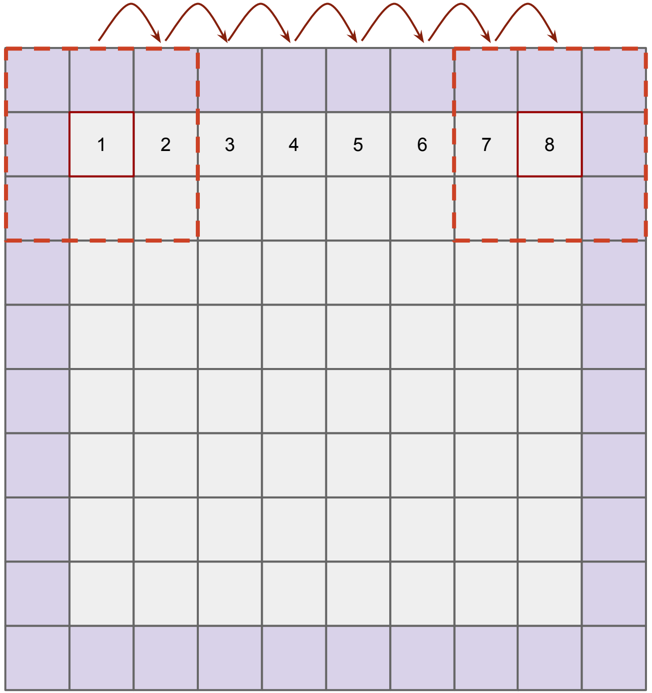
can be found **without ambiguity**



Convolution: pad but how?

Note that **number** of padded pixels **depends** on the **size of the kernel**

Kernel size: in general a center pixel can be found without ambiguity



`scipy.ndimage.convolve()`

Constant: zero, one padding commonly used

Reflect: d c b a | a b c d | d c b a

Nearest: a a a a | a b c d | d d d d

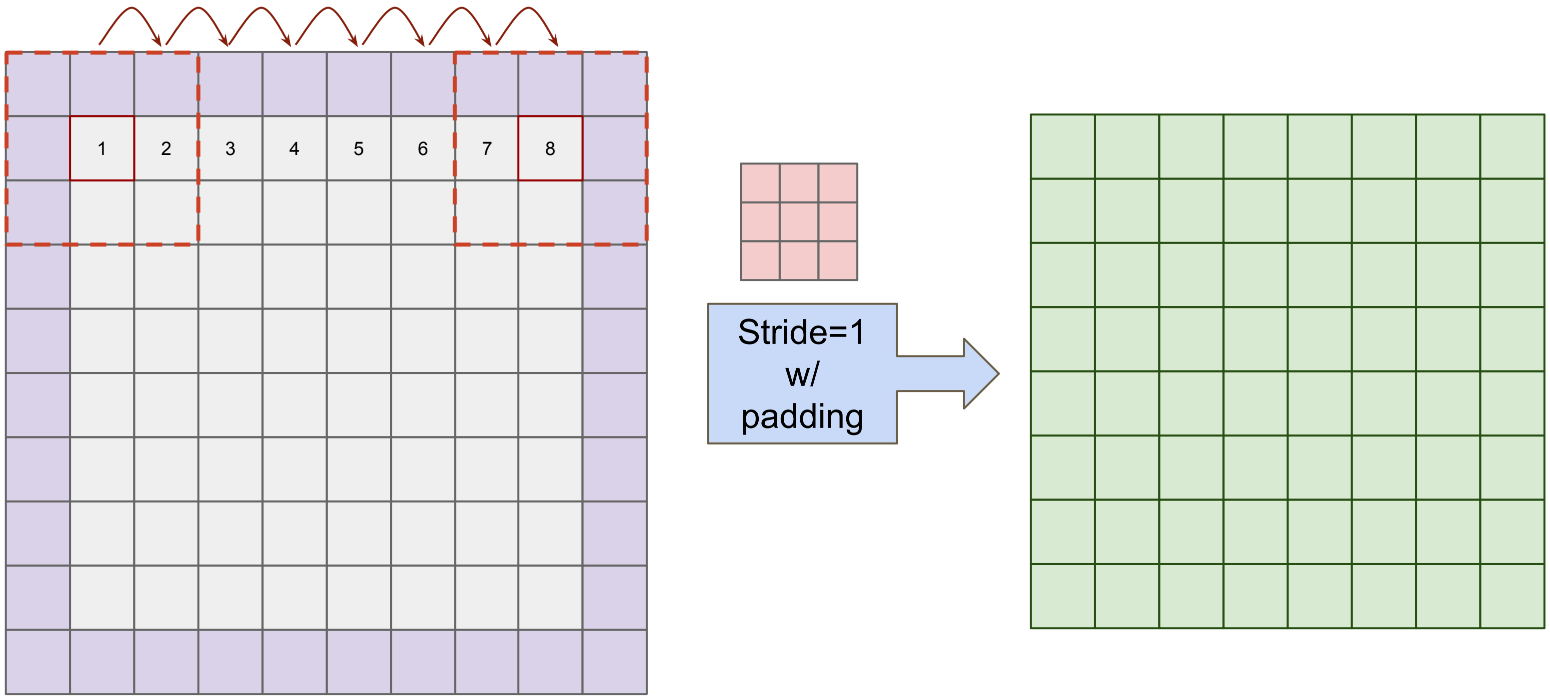
Mirror: d c b | a b c d | c b a

Wrap: a b c d | a b c d | a b c d

and more…

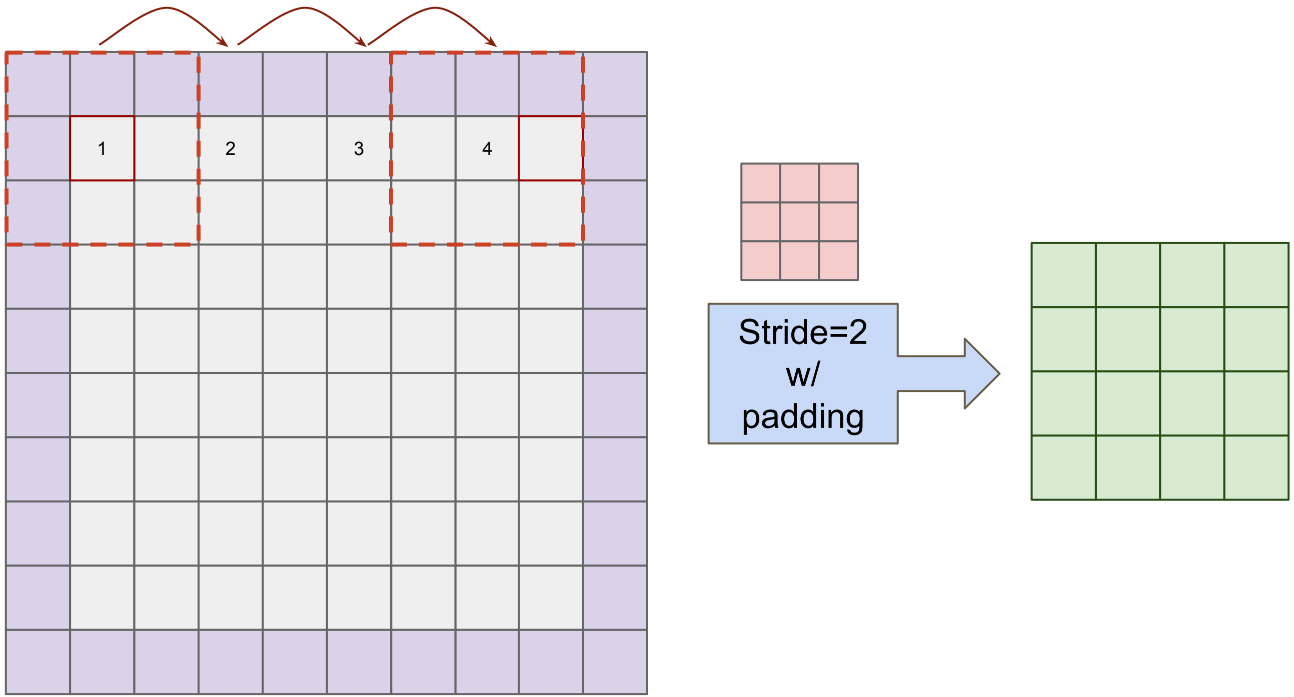
Convolution: stride - 1

Does the kernel have to move 1 step at a time?



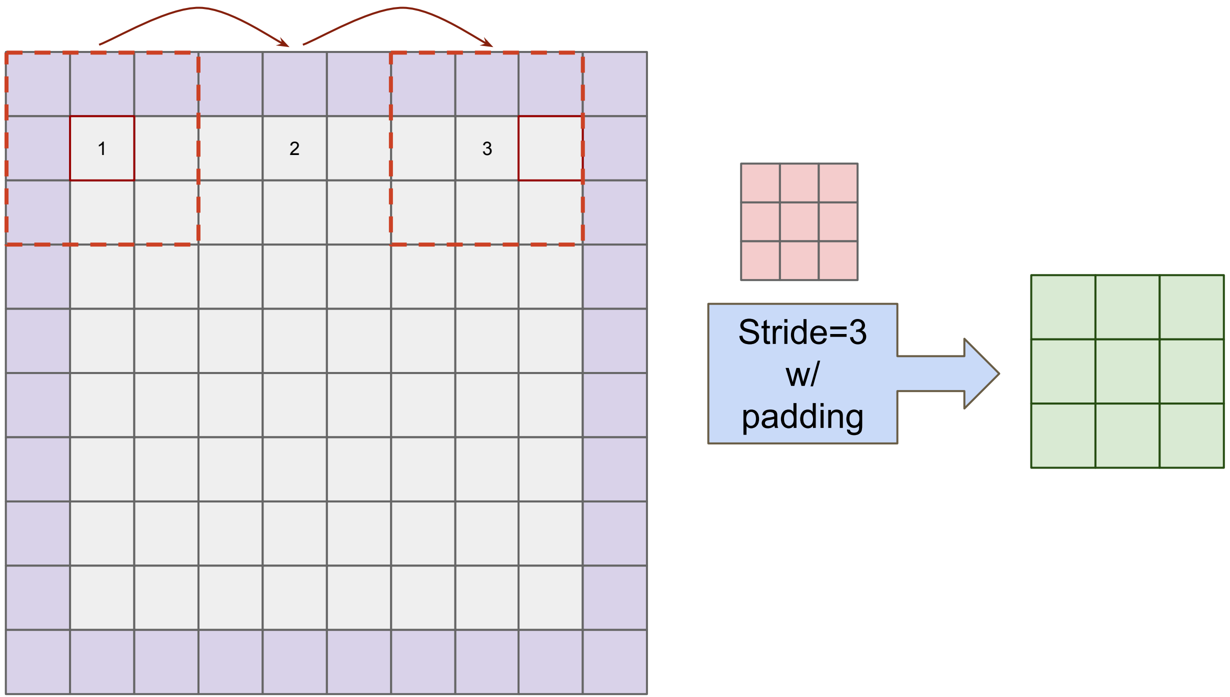
Convolution: stride - 2

Does the kernel have to move 1 step at a time?



Convolution: stride - 3

With a proper kernel, result is like a summary of the bigger picture



Convolution: popular kernels

Edge Detection:

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Sharpen:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Blur:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian Blur 3x3:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur 5x5:

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Different Kernels in action

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Operation	Kernel ω	Image result g(x,y)
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Kernels: how to choose the best?

Choice depends on the need

Tailored based on *experience* or *trial & error*

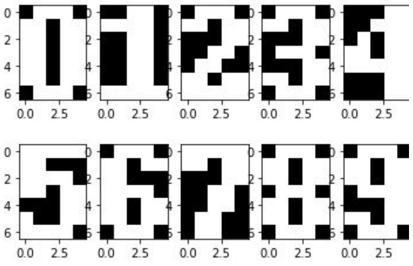
trial & error $\rightarrow$ *neural network training ?*

Using a set of specialized kernels to detect objects

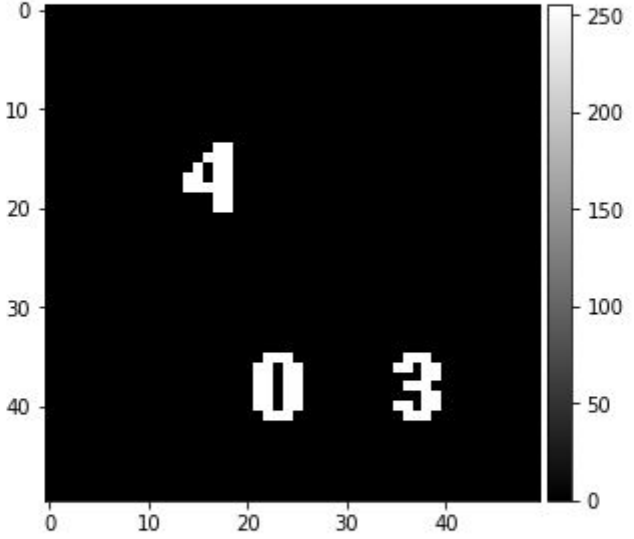
Hands on with [this notebook]

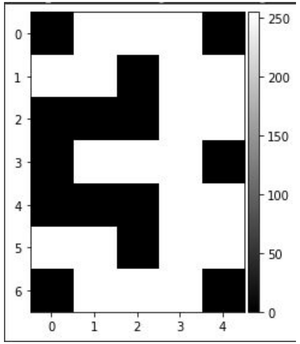
This is where convolutional networks shine!!!

Talking about detection:



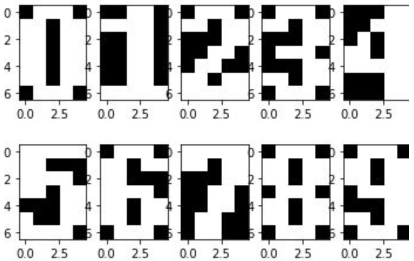
Check this page out for matching patterns using the simple convolution idea



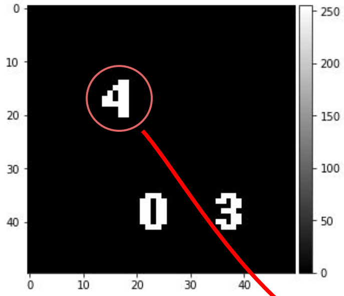


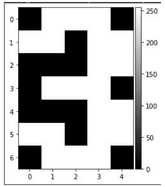
???

Talking about detection: Method 1



Match same foreground pixels in the mask (kernel, template etc)





```
[[0 1 1 1 0]
 [1 1 0 1 1]
 [0 0 0 1 1]
 [0 1 1 1 0]
 [0 0 0 1 1]
 [1 1 0 1 1]
 [0 1 1 1 0]]
```

M

```
[[0 0 0 1 1]
 [0 0 1 1 1]
 [0 1 0 1 1]
 [1 1 0 1 1]
 [1 1 1 1 1]
 [0 0 0 1 1]
 [0 0 0 1 1]]
```

C

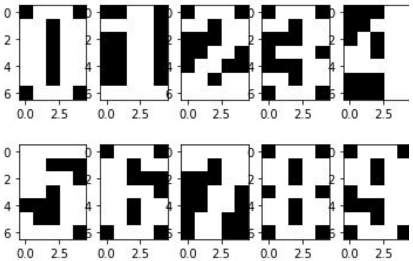
np.sum(M*C)

Result: number of of matching 1s, i.e. foreground pixels

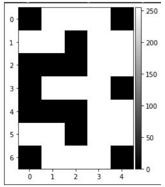
A perfect match → 21

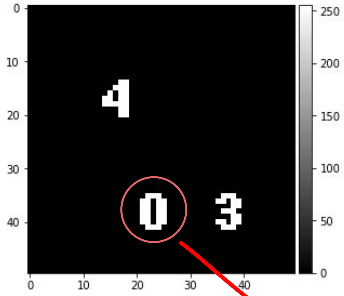
Current Score → 12

Talking about detection: Method 1



Match same foreground pixels in the mask (kernel, template etc)





```
[[0 1 1 1 0]
 [1 1 0 1 1]
 [0 0 0 1 1]
 [0 1 1 1 0]
 [0 0 0 1 1]
 [1 1 0 1 1]
 [0 1 1 1 0]]
```

M

```
[[0 1 1 1 0]
 [1 1 0 1 1]
 [1 1 0 1 1]
 [1 1 0 1 1]
 [1 1 0 1 1]
 [1 1 0 1 1]
 [0 1 1 1 0]]
```

C

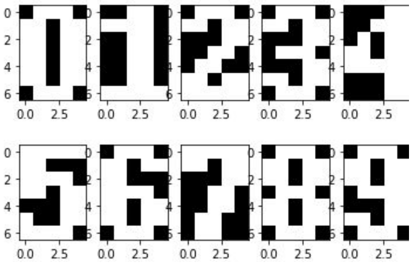
np.sum(M*C)

Result: number of of matching 1s, i.e. foreground pixels

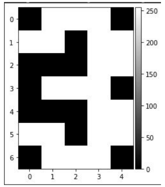
A perfect match → 21

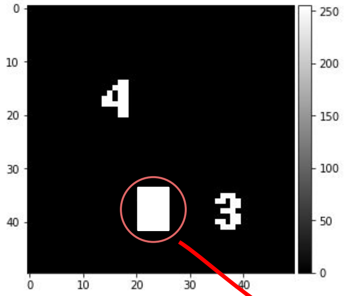
Current Score → 20

Talking about detection: Method 1



Match same foreground pixels in the mask (kernel, template etc)





```
[[0 1 1 1 0]
 [1 1 0 1 1]
 [0 0 0 1 1]
 [0 1 1 1 0]
 [0 0 0 1 1]
 [1 1 0 1 1]
 [0 1 1 1 0]]
```

M

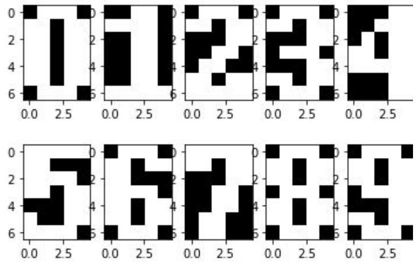
np.sum(M*C)

Oooops

Note that a perfectly white region will result in perfect match!!!

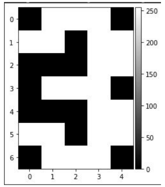
```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

C



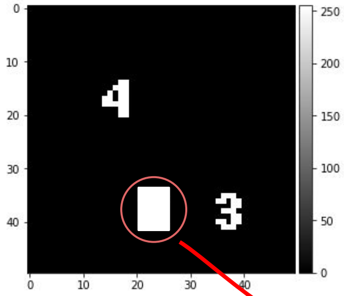
Talking about detection: Method 1

Match same foreground pixels in the mask (kernel, template etc)



```
[[0 1 1 1 0]
 [1 1 0 1 1]
 [0 0 0 1 1]
 [0 1 1 1 0]
 [0 0 0 1 1]
 [1 1 0 1 1]
 [0 1 1 1 0]]
```

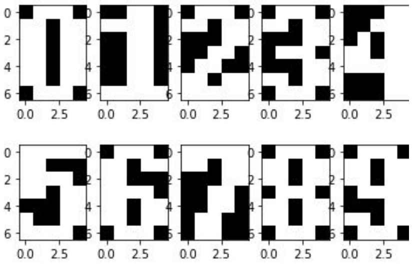
M



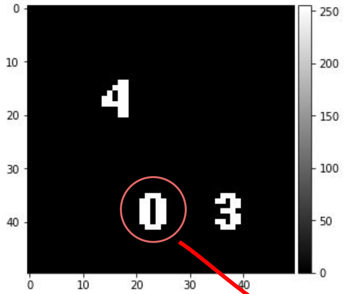
```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

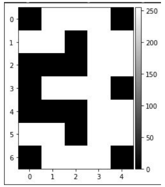
C

Talking about detection: Method 2



Match same foreground and background pixels and punish for mismatches





```
[[-1  1  1  1 -1]
 [ 1  1 -1  1  1]
 [-1 -1 -1  1  1]
 [-1  1  1  1 -1]
 [-1 -1 -1  1  1]
 [ 1  1 -1  1  1]
 [-1  1  1  1 -1]]
```

M

np.sum(M*C)

```
[[-1  1  1  1 -1]
 [ 1  1 -1  1  1]
 [ 1  1 -1  1  1]
 [ 1  1 -1  1  1]
 [ 1  1 -1  1  1]
 [ 1  1 -1  1  1]
 [-1  1  1  1 -1]]
```

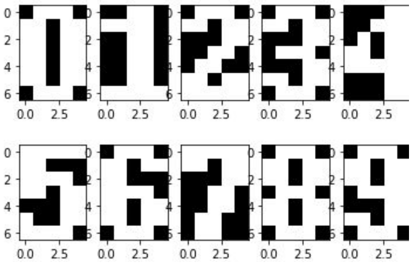
C

Result: number of all matching pixels minus all non-matching ones

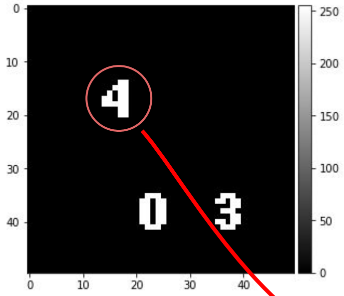
A perfect match → 35

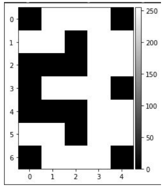
Current Score → 21

Talking about detection: Method 2



Match same foreground and background pixels and punish for mismatches






```
[[-1  1  1  1 -1]
 [ 1  1 -1  1  1]
 [-1 -1 -1  1  1]
 [-1  1  1  1 -1]
 [-1 -1 -1  1  1]
 [ 1  1 -1  1  1]
 [-1  1  1  1 -1]]
```

M

np.sum(M*C)

```
[[-1 -1 -1  1  1]
 [-1 -1  1  1  1]
 [-1  1 -1  1  1]
 [ 1  1 -1  1  1]
 [ 1  1  1  1  1]
 [-1 -1 -1  1  1]
 [-1 -1 -1  1  1]]
```

C

Result: number of all matching pixels minus all non-matching ones

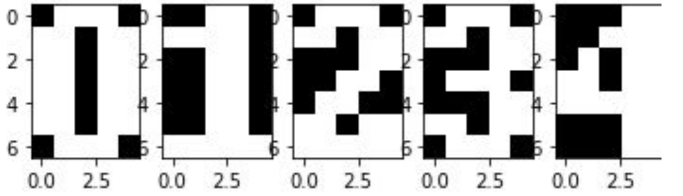
A perfect match → 35

Current Score → -1

Matching results for all the digits

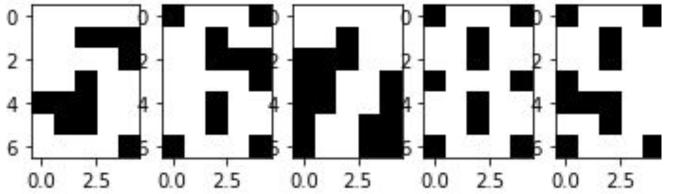
Method 1

26.00	12.00	18.00	20.00	17.00	20.00	23.00	14.00	24.00	23.00
12.00	19.00	15.00	13.00	10.00	13.00	12.00	13.00	13.00	13.00
18.00	15.00	22.00	18.00	12.00	14.00	17.00	15.00	19.00	18.00
20.00	13.00	18.00	21.00	12.00	16.00	19.00	15.00	21.00	21.00
17.00	10.00	12.00	12.00	21.00	13.00	14.00	9.00	15.00	14.00
20.00	13.00	14.00	16.00	13.00	24.00	18.00	12.00	18.00	19.00
23.00	12.00	17.00	19.00	14.00	18.00	24.00	13.00	23.00	21.00
14.00	13.00	15.00	15.00	9.00	12.00	13.00	19.00	15.00	15.00
24.00	13.00	19.00	21.00	15.00	18.00	23.00	15.00	25.00	23.00
23.00	13.00	18.00	21.00	14.00	19.00	21.00	15.00	23.00	24.00

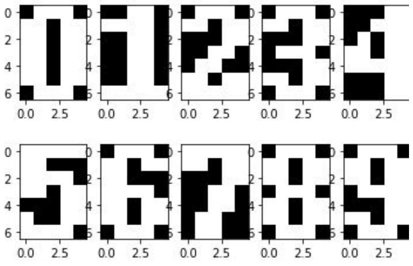


Method 2

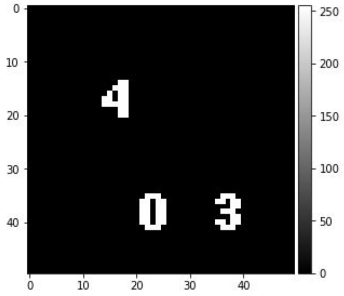
35.00	-7.00	11.00	21.00	9.00	15.00	27.00	1.00	29.00	27.00
-7.00	35.00	13.00	7.00	-5.00	1.00	-3.00	11.00	-1.00	1.00
11.00	13.00	35.00	21.00	-3.00	-1.00	11.00	13.00	17.00	15.00
21.00	7.00	21.00	35.00	-1.00	9.00	21.00	15.00	27.00	29.00
9.00	-5.00	-3.00	-1.00	35.00	-3.00	1.00	-9.00	3.00	1.00
15.00	1.00	-1.00	9.00	-3.00	35.00	11.00	-3.00	9.00	15.00
27.00	-3.00	11.00	21.00	1.00	11.00	35.00	1.00	29.00	23.00
1.00	11.00	13.00	15.00	-9.00	-3.00	1.00	35.00	7.00	9.00
29.00	-1.00	17.00	27.00	3.00	9.00	29.00	7.00	35.00	29.00
27.00	1.00	15.00	29.00	1.00	15.00	23.00	9.00	29.00	35.00



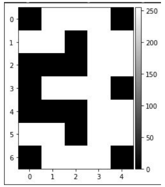
My way or highway:



You can come up with your own method...



???



What if:

- Digits are scaled?
- Digits are rotated?
- There is noise in the image?