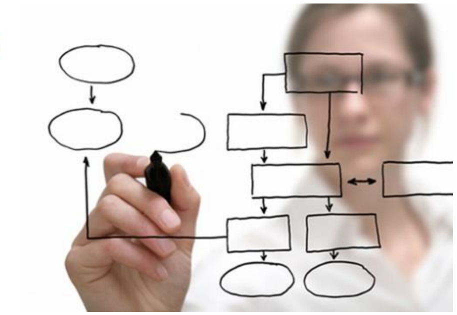
CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT



Machine Vision

Lecture 5: Image Segmentation & Binary Images

Segmentation

- The goal of image segmentation is to partition the image into two (or more) **image objects**
- We typically consider one of the segments the **background** and the rest the foreground objects

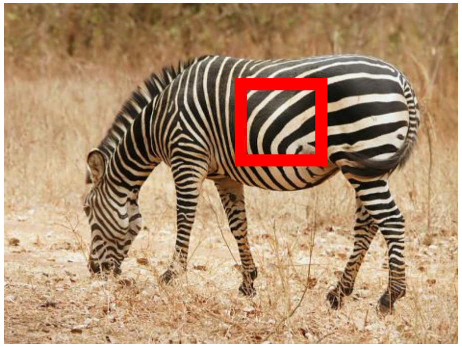
Segmentation

- We have already seen how thresholding can be used to segment into foreground and background
- This approach decided pixel by pixel if it belongs to the foreground or the background
- The drawback of this is, that it cannot take into account coherence across larger areas nor is it able to deal with textures

Texture descriptors

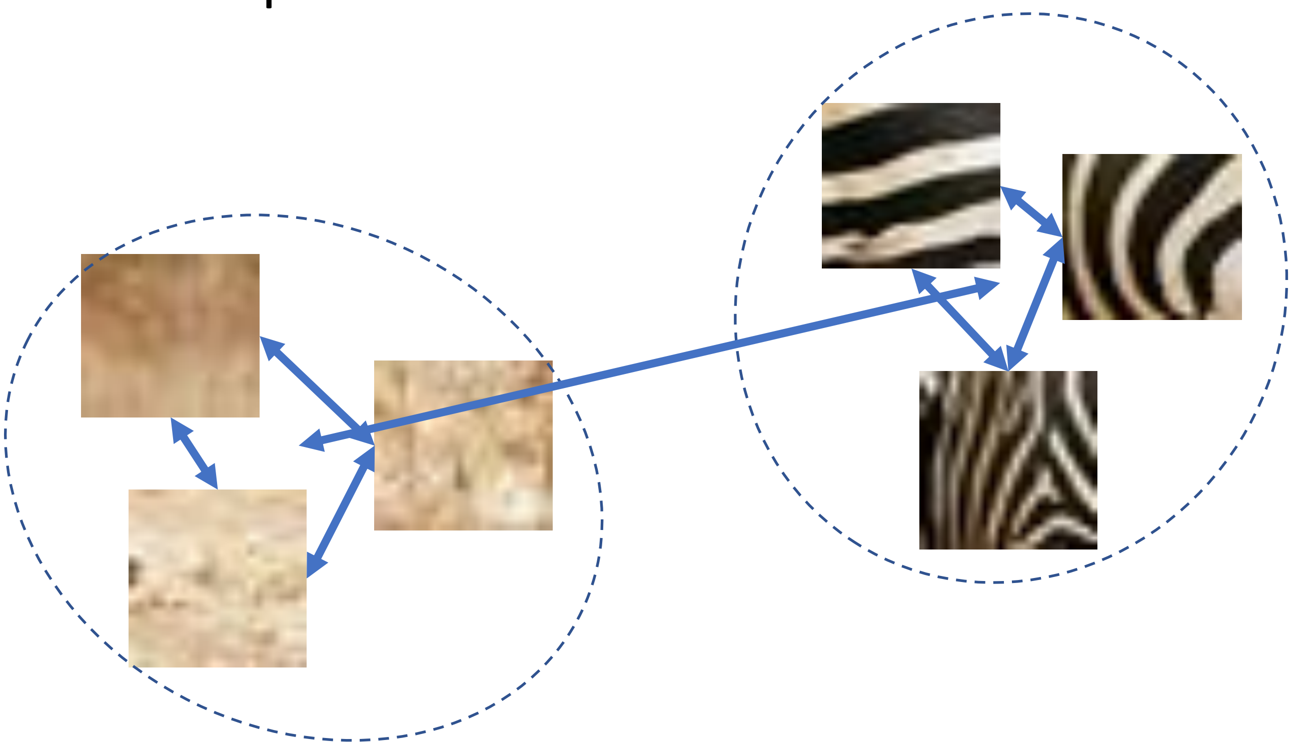
- Instead of using single pixels we can use texture descriptors, such as gradient histograms or windowed Fourier transformation

$$= \begin{pmatrix} \phi_1[x,y] \\ \vdots \\ \phi_n[x,y] \end{pmatrix}$$



K-means clustering

- The image is then considered a set of features $\{\phi[x, y]\}$ for which unsupervised learning techniques, such as k-means clustering, can be applied to identify compact clusters in the descriptor data



Smooth contours

- Typically we want objects to be delineated by smooth and closed curves, which neither thresholding nor k-means clustering is taking into consideration
- A contour can be described as a function $\Gamma: [0,1] \rightarrow \mathbb{R}^2$
- We can now state the problem of segmentation as finding an optimal contour that best fits the image while being smooth, or as minimising the energy function

$$E[\Gamma] = \int_0^1 \alpha E_{int}\left[\Gamma[c]\right] + \beta E_{img}\left[I\left[\Gamma[c]\right]\right] dc$$

- Where

$$E_{img} = -|\nabla I|$$

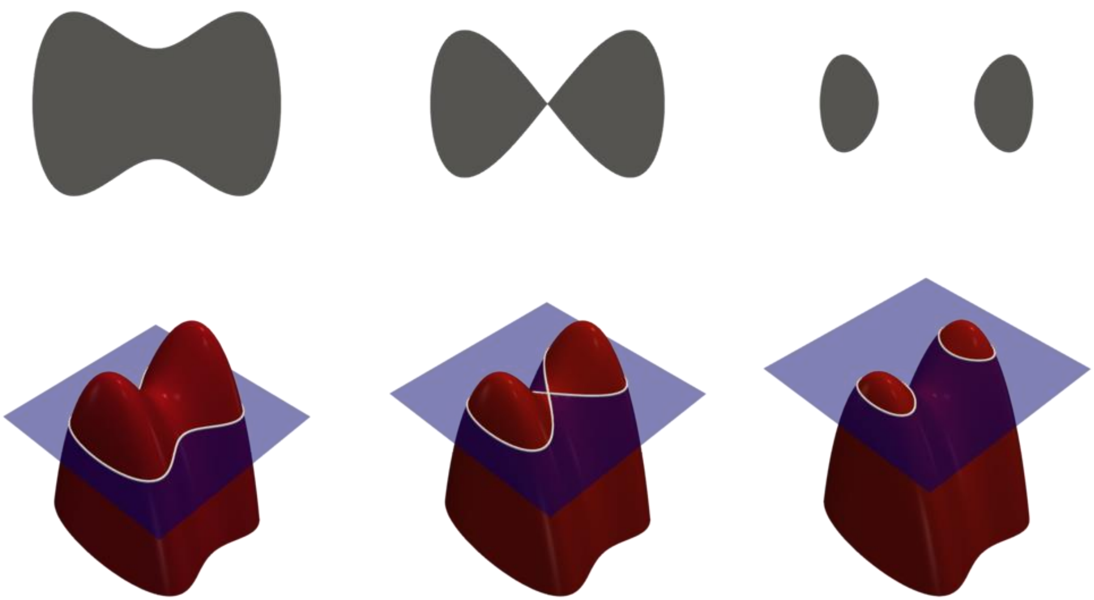
- And

$$E_{int} = w_1|\Gamma'| + w_2|\Gamma''|$$

Level sets

- A closed curve Γ can be described by a level-set of an implicit function

$$\psi(x,y) = \begin{cases} 0 & (x,y) \in \Gamma \\ -\epsilon & (x,y) \in \Gamma_{in} \\ +\epsilon & (x,y) \in \Gamma_{out} \end{cases}$$



Segmentation cost functions

- The segmentation problem can therefore be stated as a cost optimisation on the implicit function ψ

$$E[\psi] = \sum_{x,y} \alpha\, E_{int}[\psi] + \beta E_{img}[\psi]$$

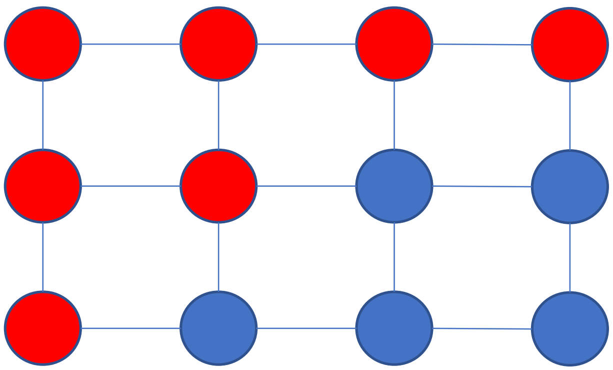
- Where E_{int} enforces smoothness and E_{img} enforces consistency with the image data
- The benefit over curve representations is, that the function $\psi[x, y]$ is defined over the image domain, so the definition of E_{img} is more straightforward

Markov-Random-Fields

- A discretised view on the segmentation problem is to consider it as a Markov-Random-Field
- The image is considered a graph of pixels, with each pixel's segment assignment depending only on its neighbouring pixels
- For each node a cost is incurred for the dissimilarity of the image descriptors as well as for the dissimilarity of the assignments

$$E[x,y] = \sum_{(x',y')\in N[x,y]} \alpha e^{-\lambda(\psi[x,y]=\psi'[x',y'])(\phi[x,y]-\phi[x',y'])^2} + \beta e^{-\lambda(\psi[x,y]\neq\psi'[x',y'])}$$

- The total cost $E = \sum_{x,y} E[x,y]$ can then be minimised using Markov-Chain-Monte-Carlo optimisation



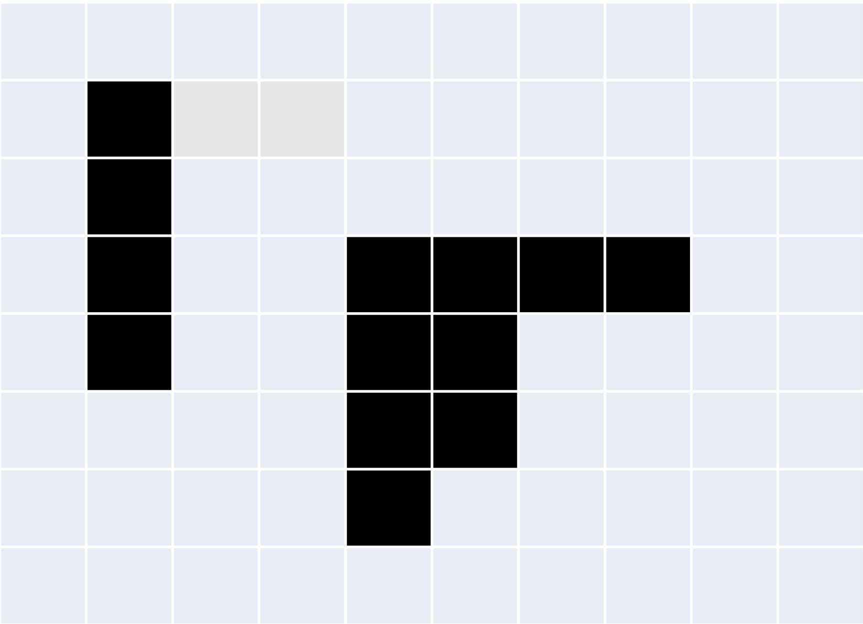
Binary images

- The result of segmentation is typically a binary image
- These binary images have only values 0 or 1, typically 1 signifies some relevant object while 0 is the background

- We will now look into image processing algorithms explicitly for binary images, which can be used to further process the segmentation results

Connected regions

- Often we are interested in connected regions in a binary image, as these are usually individual “objects”



Connected regions

- Often we are interested in connected regions in a binary image, as these usually individual “objects”
- For example in Optical Character Recognition (OCR) we are interested in isolating individual letters for further processing

Neighbourhoods

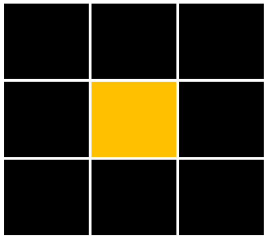
- Connectedness of regions requires a definition of neighbourhood
- Typically, we work with two different types of neighbourhoods on a raster

- The 4-neighbourhood of a pixel (x, y) is the set

$$N_4(x, y) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$$

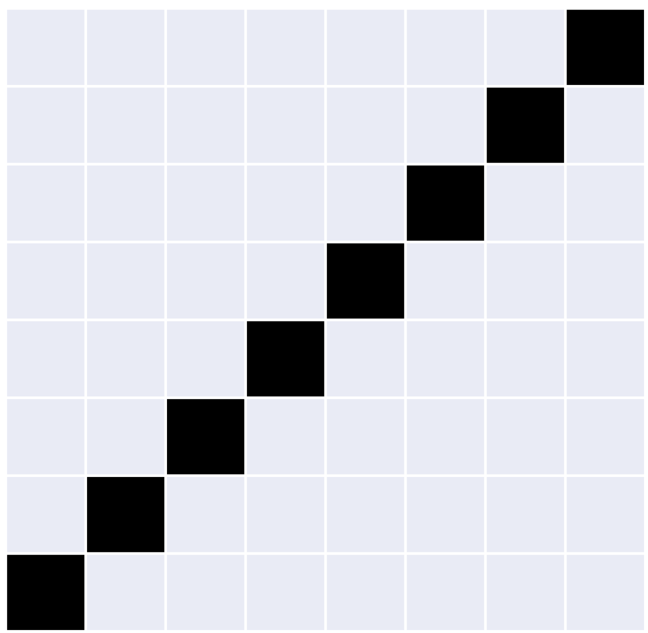
- The 8-neighbourhood of a pixel (x, y) is the set

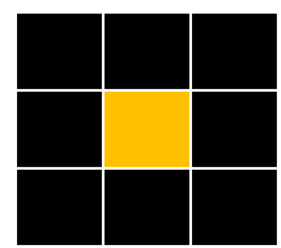
$$N_8(x, y) = N_4 \cup \{(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}$$



Neighbourhoods

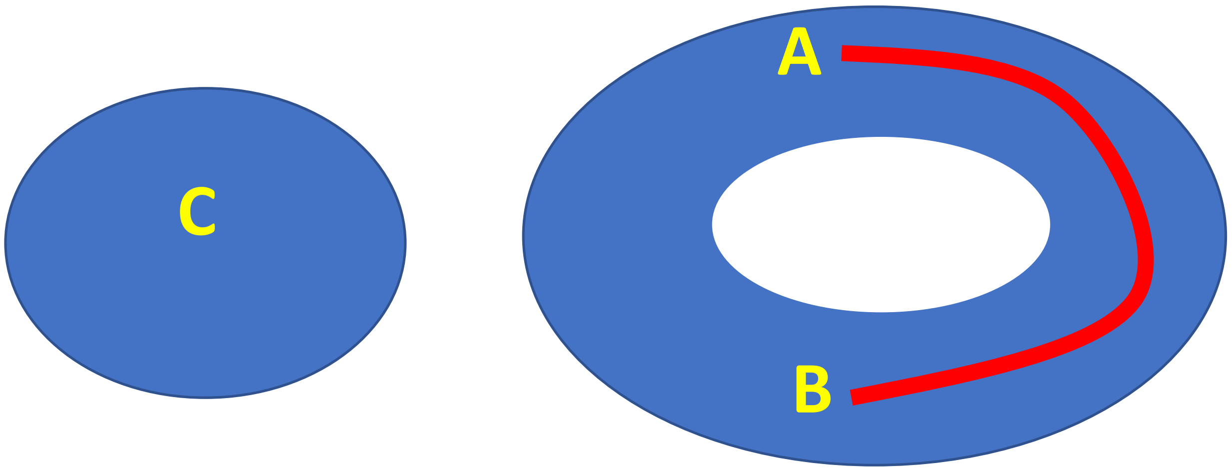
- For example, a diagonal line is connected using a N_8 definition of neighbourhood, but not connected using a N_4 definition of neighbourhood
- Therefore, we need to be careful to understand what neighbourhood to use





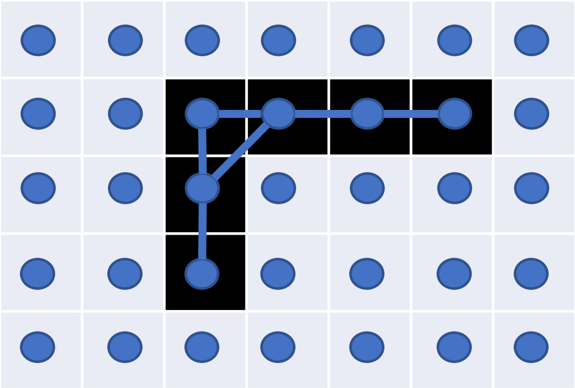
Connected components

- A connected component in a binary image is a region in which every pair of points can be connected by a line passing solely through the region
- We also can define holes to be the connected components of the inverse image



Connected components

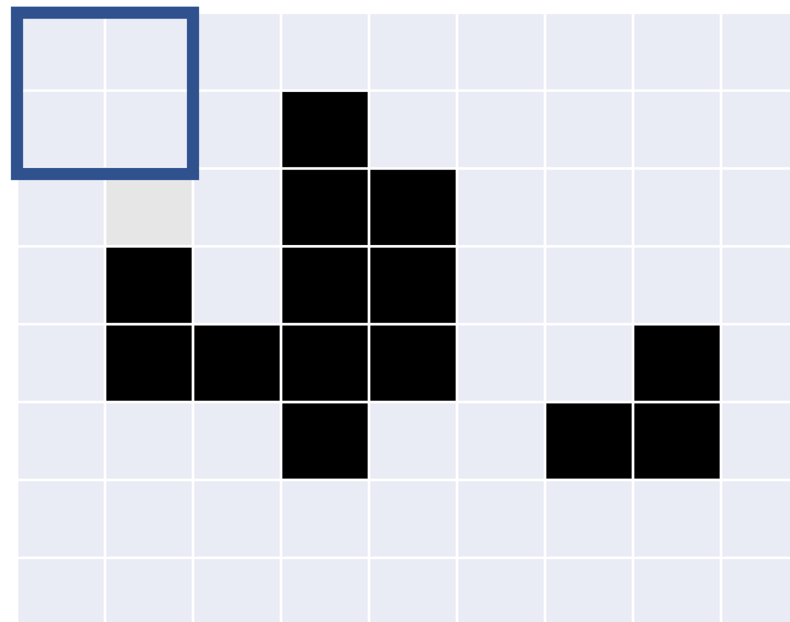
- We can consider the binary image as a graph
$$G = (V, E)$$
- Where every pixel is a vertex $\mathrm{V} = \{(x, y)\}$ and the neighbourhood relation defines the edges
$$E = \{((x_1, y_1), (x_2, y_2)) \mid (x_2, y_2) \in N_8(x_1, x_2)\}$$
- We can now apply standard graph algorithms to this data structure
- However, this is not very efficient



Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

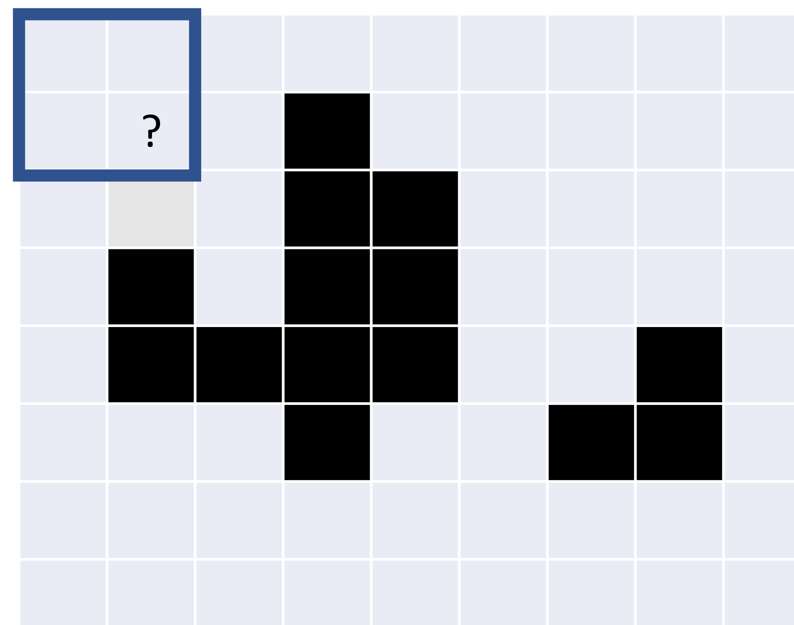


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- *If A=0, then A=0*

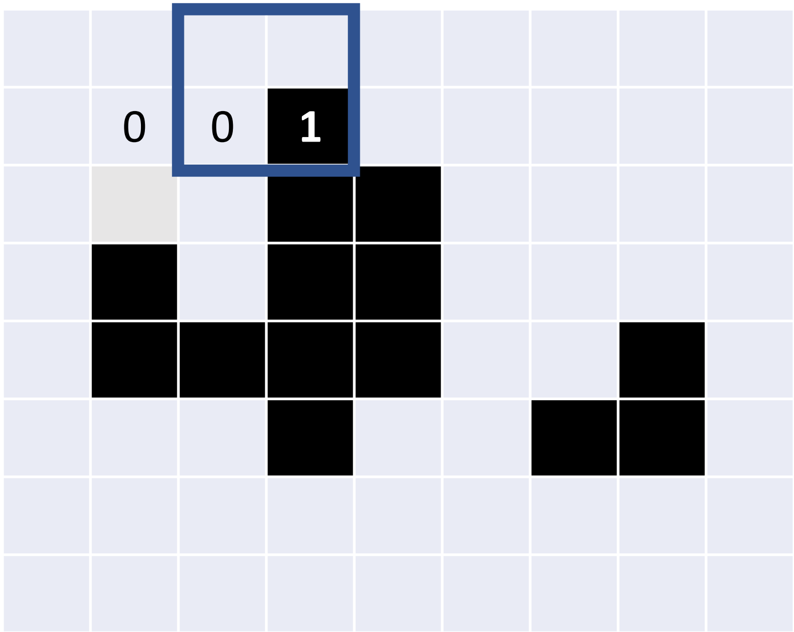


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- *If A>0 and B=C=D=0, then assign a new label to A*

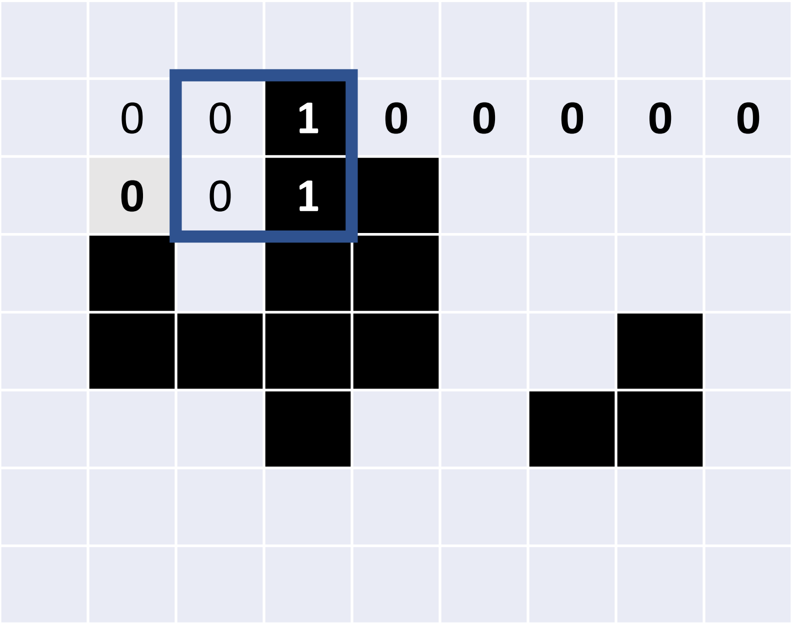


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- If A>0 and B=C=D=0, then assign a new label to A
- *If A>0 and C>0, then A=C*

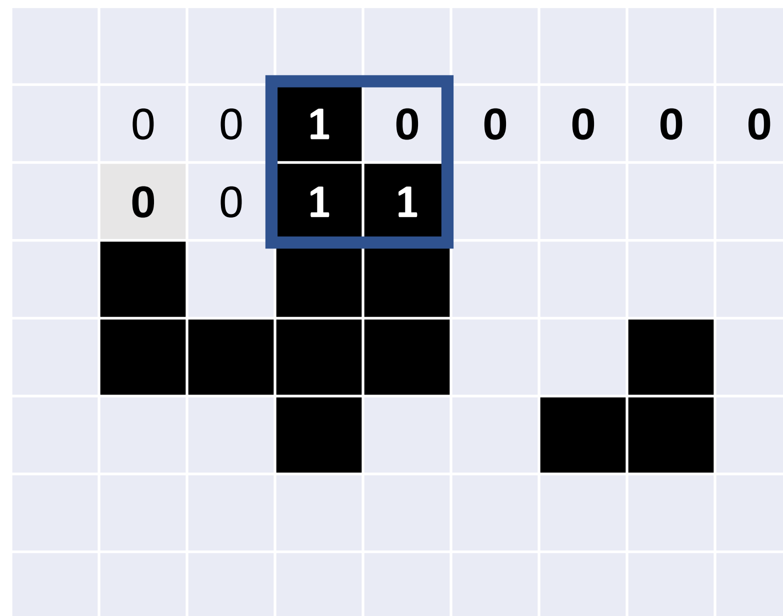


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- If A>0 and B=C=D=0, then assign a new label to A
- If A>0 and C>0, then A=C
- *If A>0 and B,C,D consistent >0, assign the consistent label to A*

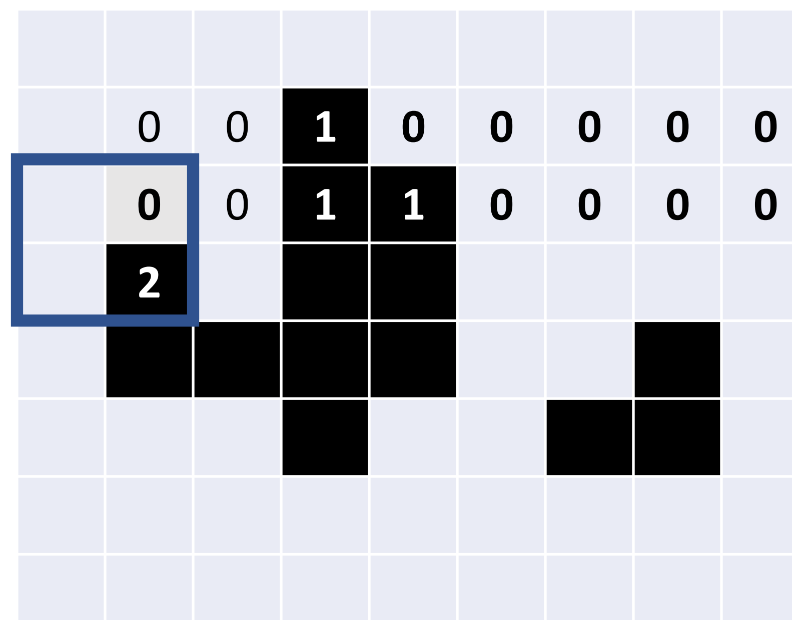


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- *If A>0 and B=C=D=0, then assign a new label to A*
- If A>0 and C>0, then A=C
- If A>0 and B,C,D consistent >0, assign the consistent label to A

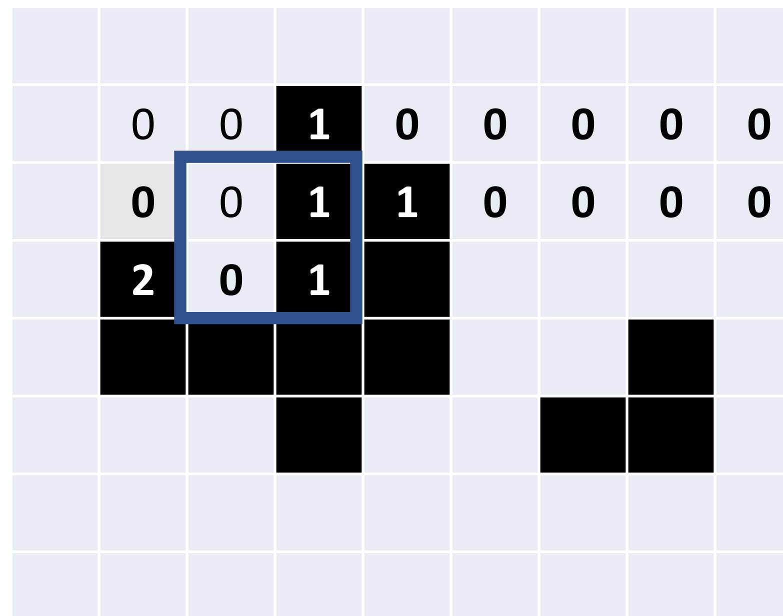


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- If A>0 and B=C=D=0, then assign a new label to A
- *If A>0 and C>0, then A=C*
- If A>0 and B,C,D consistent >0, assign the consistent label to A

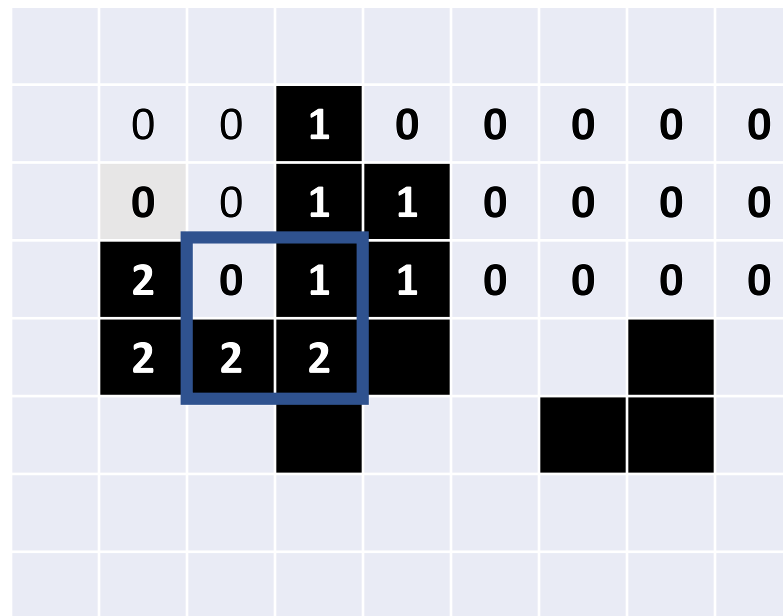


Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- If A>0 and B=C=D=0, then assign a new label to A
- If A>0 and C>0, then A=C
- If A>0 and B,C,D consistent >0, assign the consistent label to A
- *If A>0 and B,C,D inconsistent >0, assign any of B,C,D and take note of equality of labels*



$$\{1 = 2\}$$

Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- *If A>0 and B=C=D=0, then assign a new label to A*
- If A>0 and C>0, then A=C
- If A>0 and B,C,D consistent >0, assign the consistent label to A
- If A>0 and B,C,D inconsistent >0, assign any of B,C,D and take note of equality of labels

	0	0	**1**	**0**	**0**	**0**	**0**	**0**
	0	0	**1**	**1**	**0**	**0**	**0**	**0**
	2	**0**	**1**	**1**	**0**	**0**	**0**	**0**
	2	**2**	**2**	**2**	**0**	**0**	**3**	**0**
	0	**0**	**2**	**0**	**0**	**4**		

$\{1 = 2\}$

Connected components

- The graph approach does not exploit the specific neighbourhood structure imposed by the image grid
- It is more efficient to move a 2×2 grid over the image and assign labels sequentially for as follows:

D	**C**
B	**A**

- If A=0, then A=0
- If A>0 and B=C=D=0, then assign a new label to A
- If A>0 and C>0, then A=C
- If A>0 and B,C,D consistent >0, assign the consistent label to A
- *If A>0 and B,C,D inconsistent >0, assign any of B,C,D and take note of equality of labels*

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

$\{1 = 2, 3 = 4\}$

Connected components

- Using an efficient implementation of a Union-Set data structure the equality relations can be easily maintained and resolved
- The result is an image in which every pixel value indicates the connected component it belongs to
- We can now easily separate individual components

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

$$\{1 = 2, 3 = 4\}$$

Connected components

```
count,labels = cv2.connectedComponents(binary_image)
```

The output is the number of connected components and a label image

A connected components algorithm is implemented in OpenCV

Morphological operations

- Morphological image processing uses the topology (i.e. the neighbourhood relations) to define operations

- Two operations are of special interest

  - **Erosion**, which can be used to <u>shrink</u> the foreground and reduce the size of regions

  - **Dilation**, which can be used to <u>expand</u> the foreground and increase the size of regions

Structuring element

- First we define the neighbourhood of a pixel more generically using a **structuring element**

- A structuring element is a binary mask that describes what surrounding pixels constitute a neighbourhood
- We have already seen two such structuring elements N_4 and N_8, and these are the most useful, but every other shape is possible

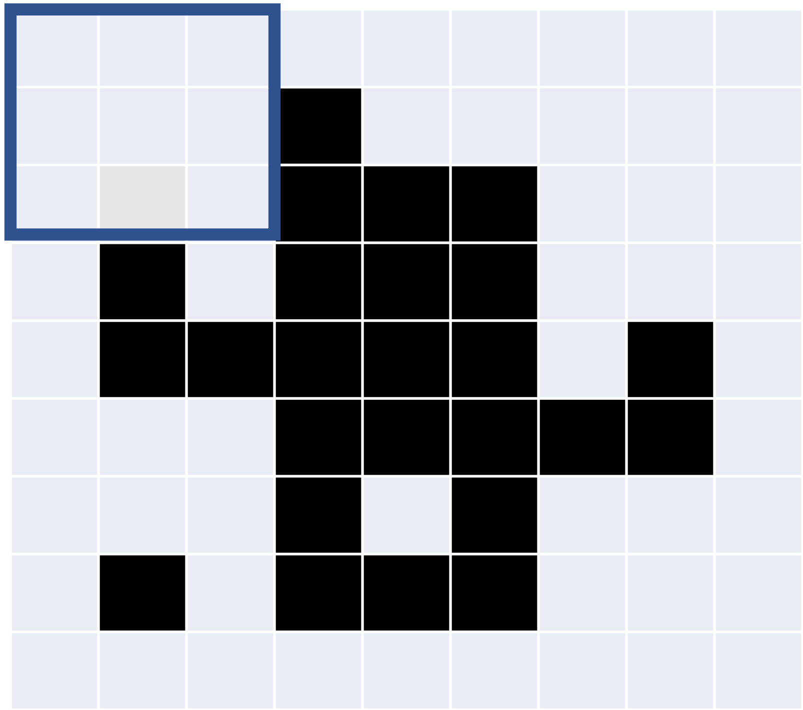
0	1	0
1	1	1
0	1	0

1	1	1
1	1	1
1	1	1

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

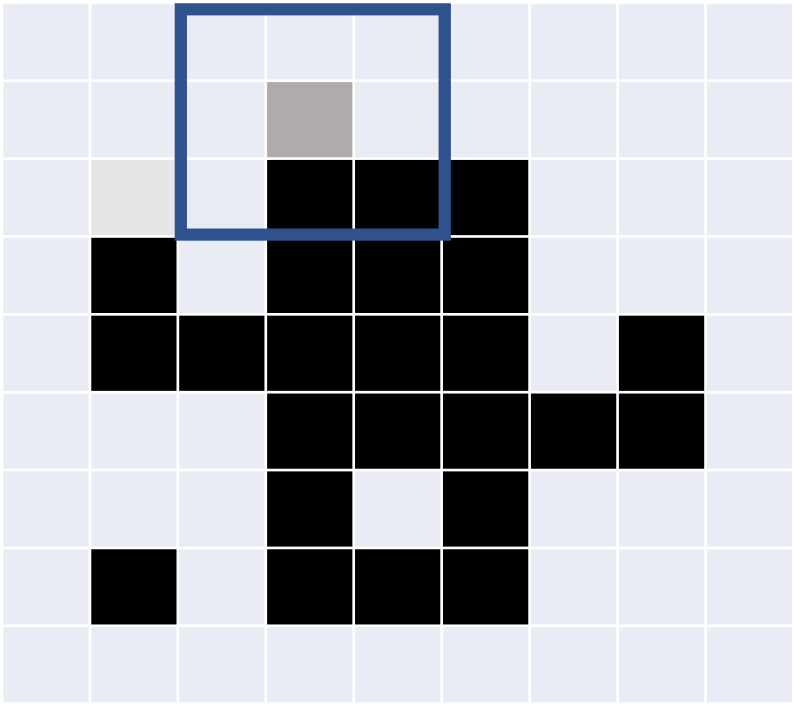
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set



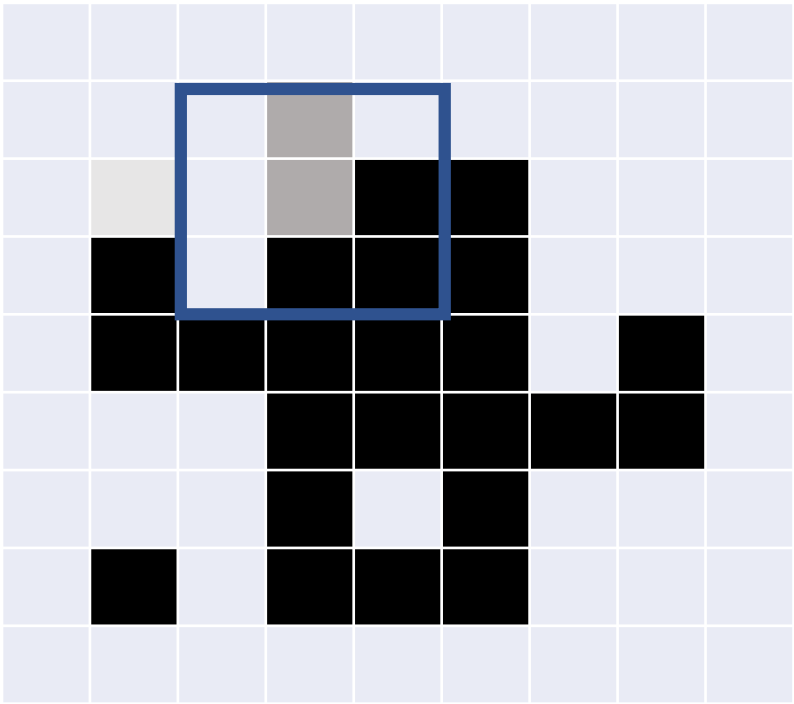
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set



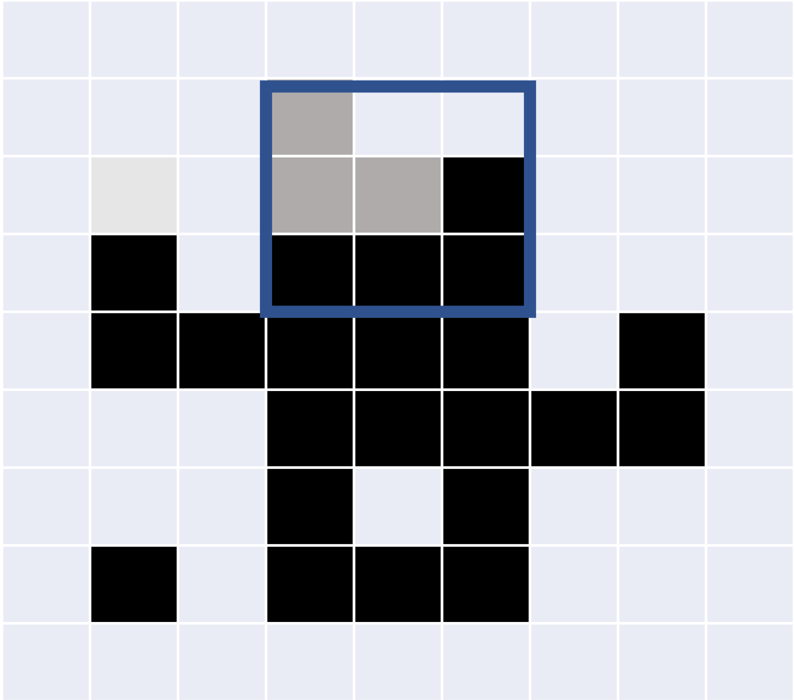
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set



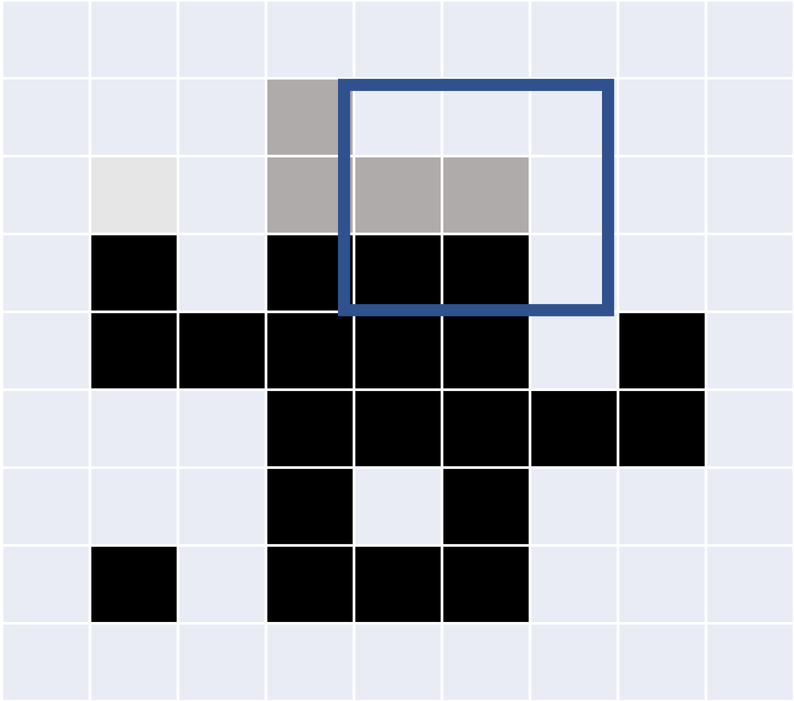
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set



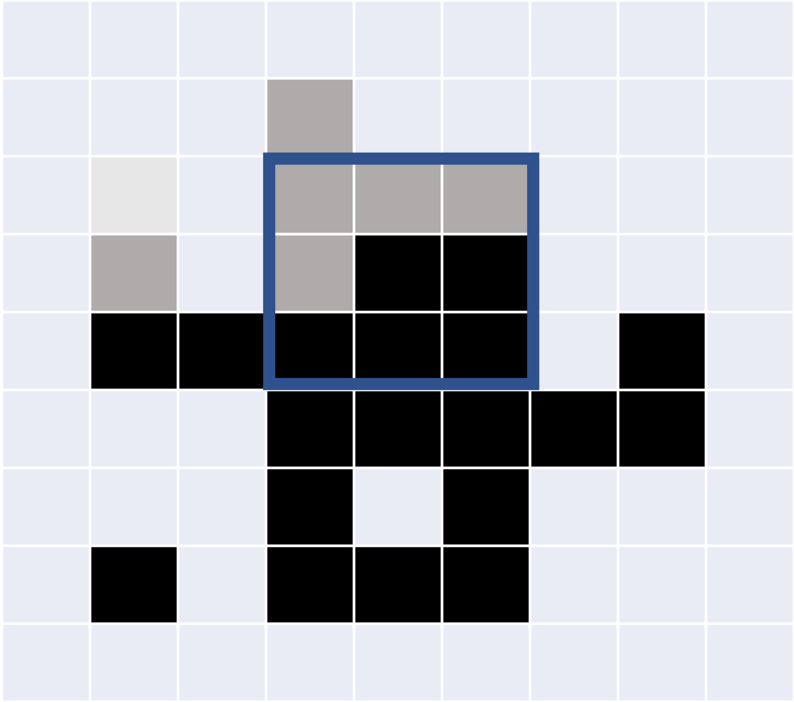
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set



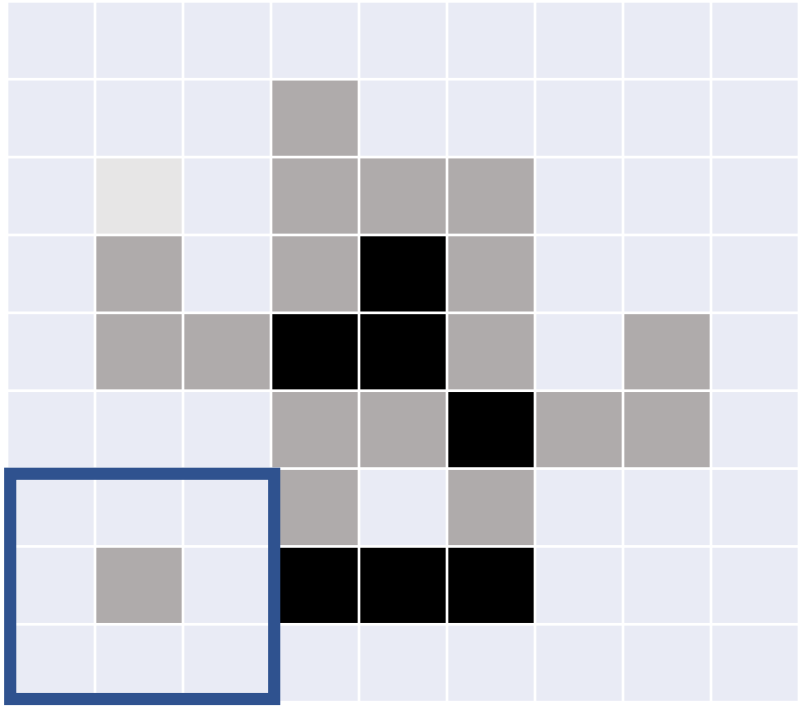
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set
- This means, only inside pixels are retained, all pixels on the boundary of the objects are removed



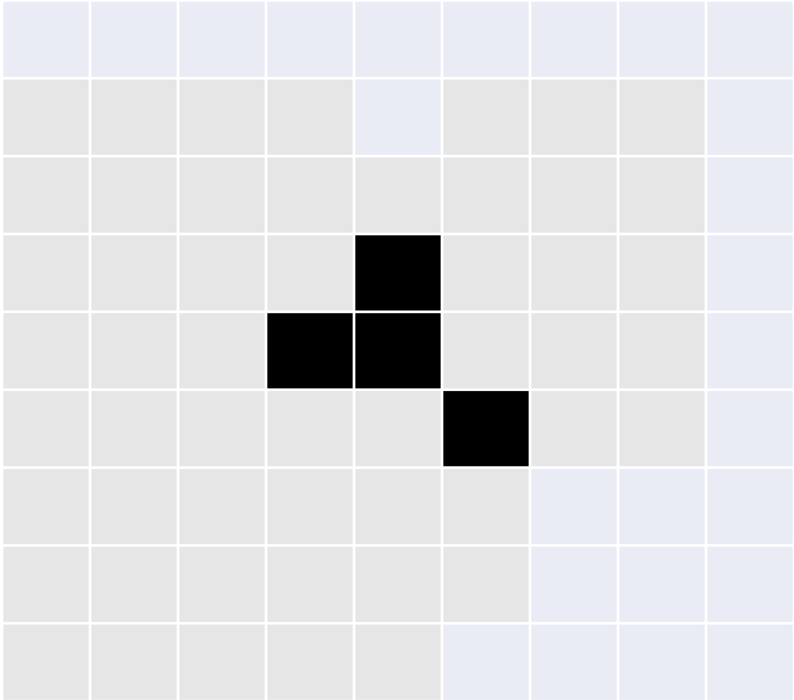
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set
- This means, only inside pixels are retained, all pixels on the boundary of the objects are removed
- Small structures are removed



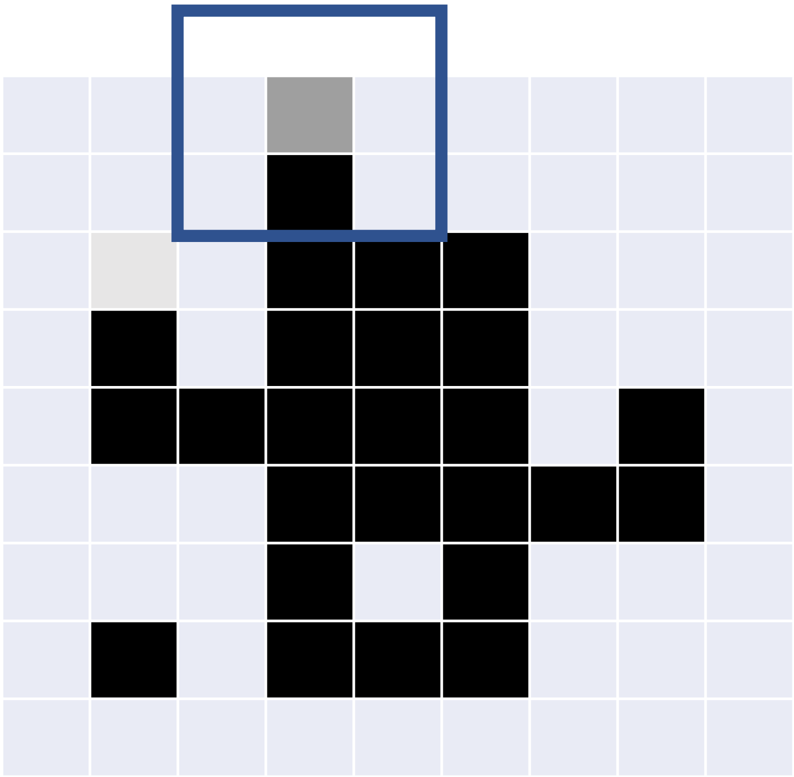
Erosion

- The **erosion** operation moves the structuring element over the image and **removes** all pixels where **not all neighbours** are set
- This means, only inside pixels are retained, all pixels on the boundary of the objects are removed
- Small structures are removed
- The boundary is shaved off, so that only larger objects are retained



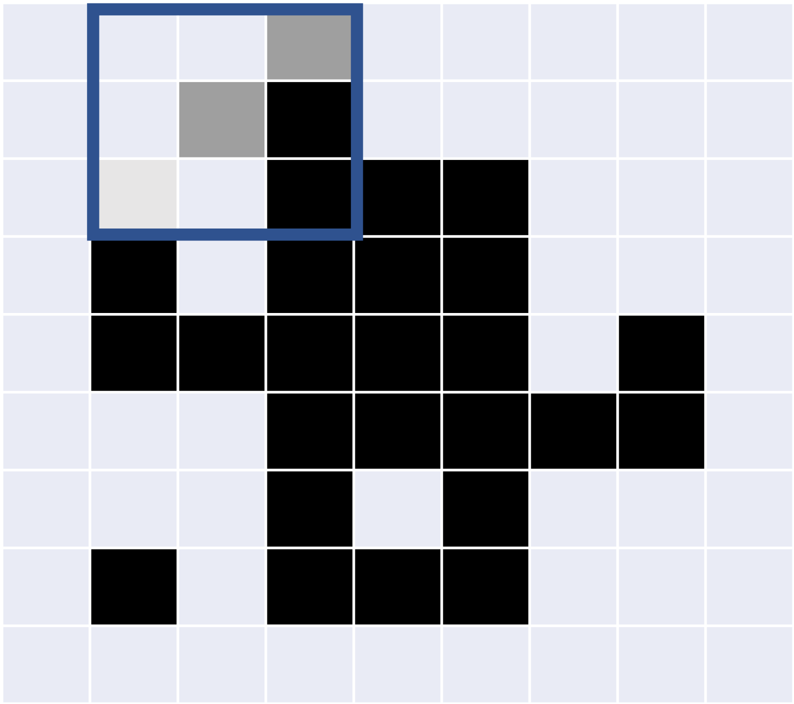
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set



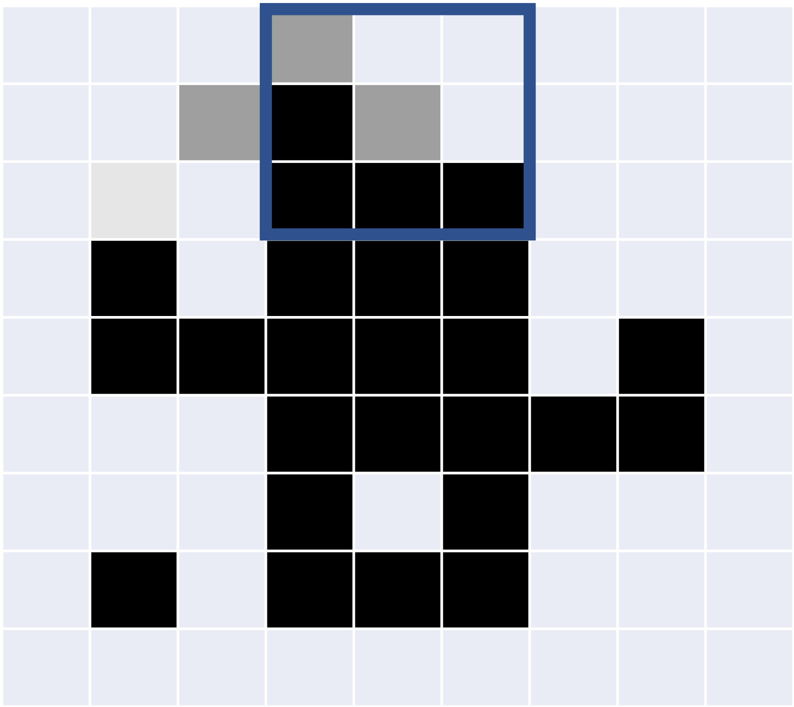
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set



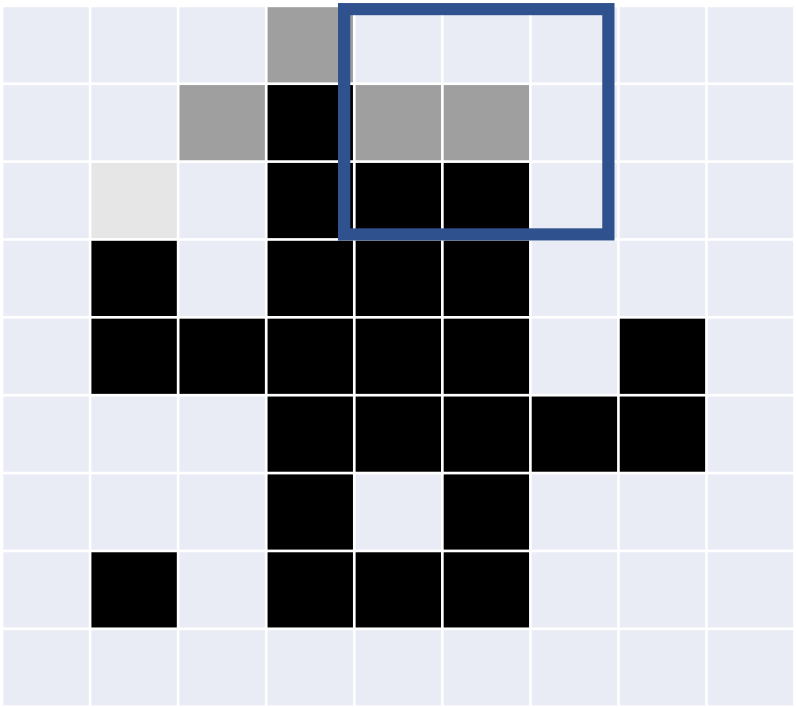
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set



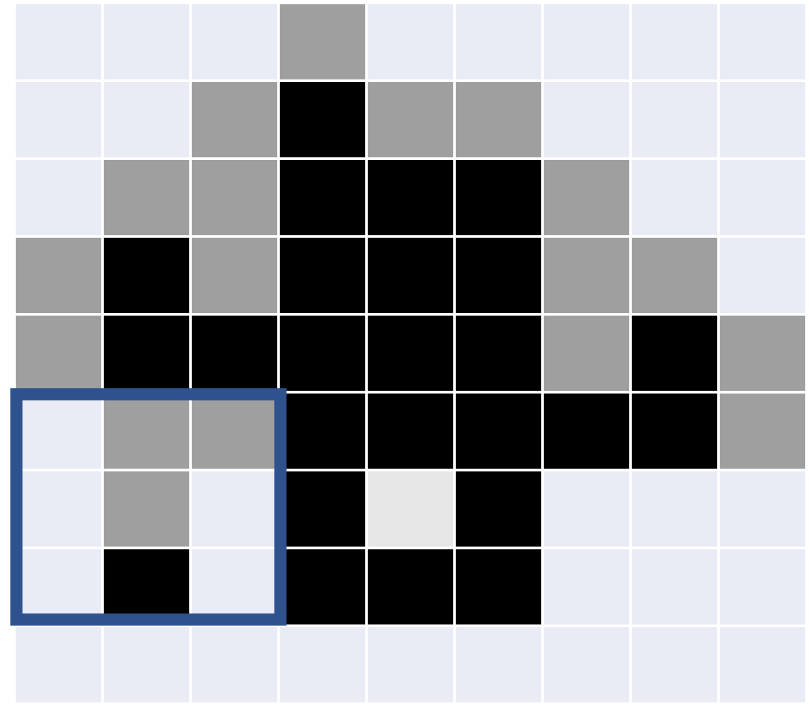
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set
- A layer is added to the outside of every object



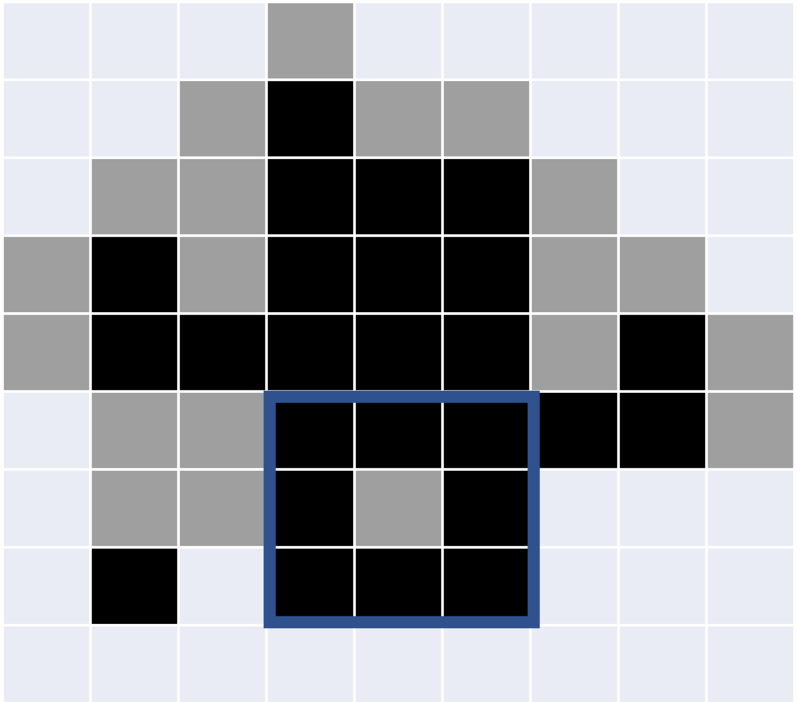
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set
- A layer is added to the outside of every object
- This can lead to structures being connected together



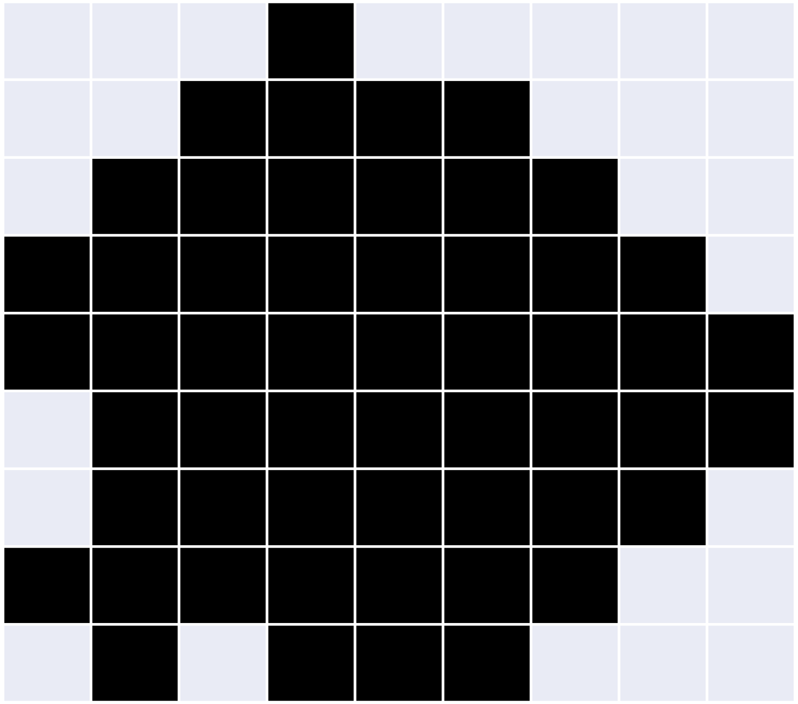
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set
- A layer is added to the outside of every object
- This can lead to structures being connected together
- Holes are filled



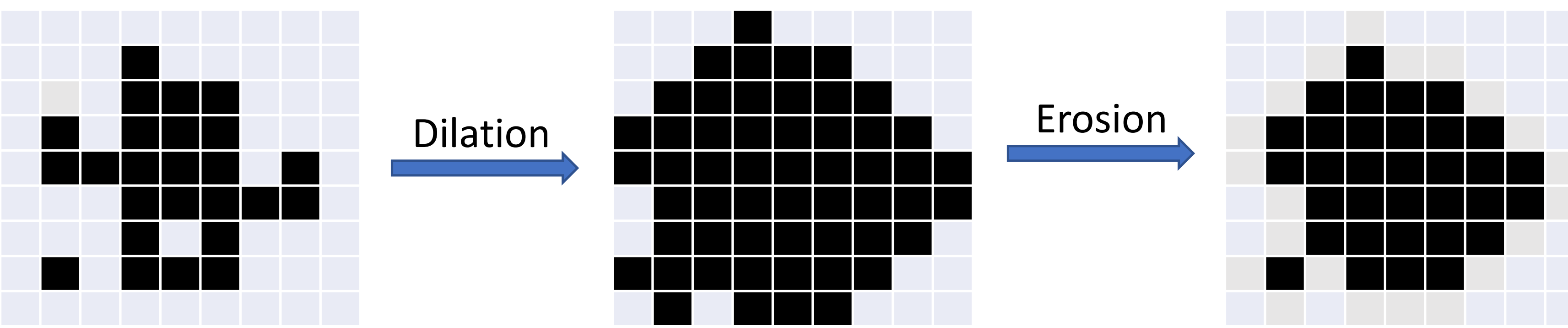
Dilation

- The **dilation** operation moves the structuring element over the image and **adds** pixels where **any neighbour** is set
- A layer is added to the outside of every object
- This can lead to structures being connected together
- Holes are filled
- Objects get larger and finer structures are removed



Closing

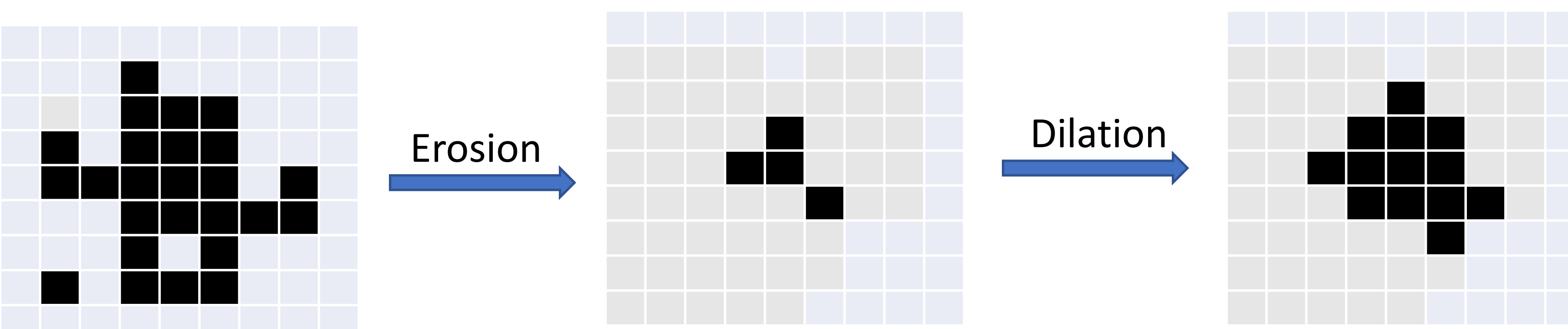
- A useful property of dilation is that is closes small holes and removes smaller details that could be caused by noise
- The problem is, that the foreground grows with each iteration
- The **closing** operation tries to solve this by first dilating the image to close holes and then eroding the image again to remove the extra boundary



Opening

- Similar to closing, we can first erode and the dilate
- This operation is called **opening**

- It is used to remove smaller objects and structures while retaining the larger objects



Morphological operations

The structuring element is defined as NumPy array

```
structuring_element = np.ones((3,3),np.uint8)
erosion = cv2.erode(binary_img,structuring_element, iterations=1)
```

Erosion

```
dilation = cv2.dilate(binary_img,structuring_element, iterations=1)
```

Dilation

All morphological operations can be iterated, resulting in more layers being added/removed

Morphological operations

```
opening = cv2.morphologyEx(binary,
                           cv2.MORPH_OPEN,
                           structuring_element,
                           iterations = 1)
```

Other morphological operations are called like this

The type of operation is passed as parameter

```
closing = cv2.morphologyEx(binary,
                           cv2.MORPH_CLOSE,
                           structuring_element,
                           iterations = 1)
```

Features of binary images

- There are some feature descriptors that are specific to binary images
- They can the categorised in two groups
  - **Topological feature descriptors** that characterise the neighbourhood structure of the object (e.g. the number of connected components)
  - **Geometric feature descriptors** that characterise the shape of the object (e.g. the area or the perimeter)

Topological descriptors

- The **Euler characteristic** is the difference between the number of components and the number of holes

$$E = \#components\ - \#holes$$

- For example: $E = 2 - 3 = -1$

- Topological feature descriptors are invariant to geometric distortions

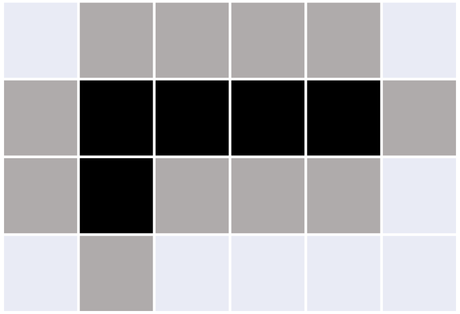
Area and perimeter

- The **area** of a binary image is simply the number of foreground pixels

$$A = \sum_{x,y} I[x,y]$$

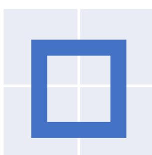
- The **perimeter** is the number of background pixels with a foreground pixel in their neighbourhood

$$P = |\{(x,y) \mid \exists (x',y') \in N_4(x,y): I[x',y'] = 1\}|$$

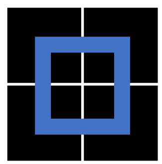


Area and perimeter

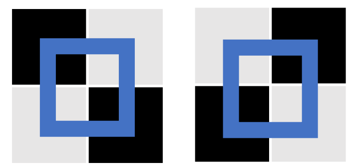
- To calculate area and perimeter, we scan over all $2x2$ patches and consider a pixel sized rectangle between the four pixels
- Depending on the configuration of points we accumulate



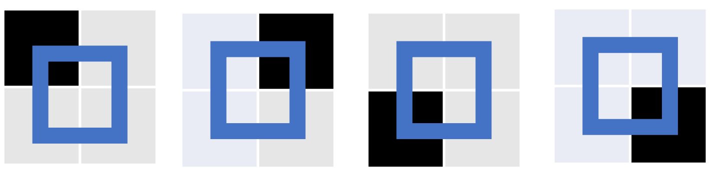
$A += 0$
$P += 0$



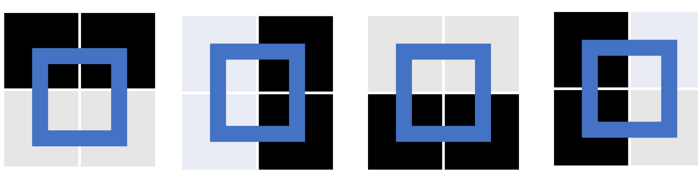
$A += 1$
$P += 0$



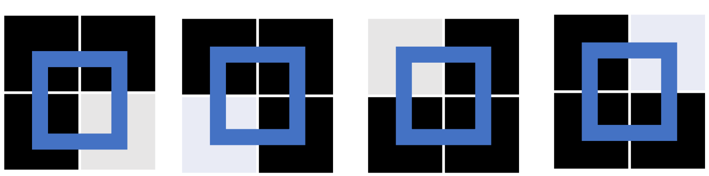
$A += 1/2$
$P += 1$



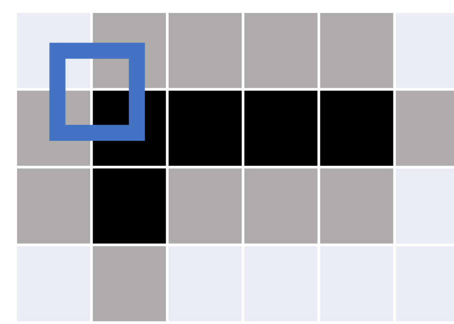
$A += 1/4$
$P += 1/2$



$A += 1/2$
$P += 1/2$



$A += 3/4$
$P += 1/2$



Form factor

- Area and perimeter are invariant to translation, but not to scale changes

- The **form factor** is the normalised ratio between squared perimeter and area

$$k = \frac{P^2}{4\pi A}$$

- It is invariant to scale changes
- A circle has a form factor of $k = 1$

Moments

- The **raw moment** of a binary image is defined as

$$M_{ij} = \sum_{x,y} x^i y^j I[x, y]$$

- The area is m_{00}, the centroid is $\left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right)$
- Subtracting the centroid yields the **central moments** defined as

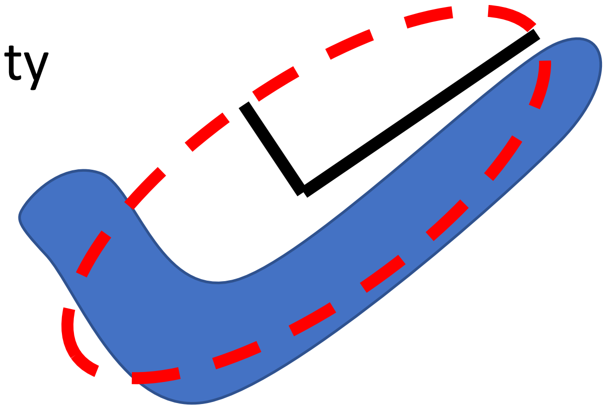
$$\mu_{ij} = \sum_{x,y} \left(x - \frac{M_{10}}{M_{00}}\right)^i \left(y - \frac{M_{10}}{M_{00}}\right)^j I[x, y]$$

Moments

- The eigenvalues and eigenvectors of the second moment matrix (or covariance matrix)

$$\Sigma = \begin{pmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{pmatrix}$$

- Tell us the orientation and shape of the object
- From this we can derive quantities like eccentricity



$$e = \sqrt{1 - \frac{\lambda_2}{\lambda_1}}$$

- or the direction of the major axis

$$\theta = \operatorname{atan} \frac{2\mu_{11}}{\mu_{20} - \mu_{02}}$$

Thank you for your attention!