CITY
UNIVERSITY OF LONDON
EST 1894

Computer Vision
INM460 / IN3060

Dr Sepehr Jalali

Lecture 4: Image segmentation

Recap from last time

Image filtering

- Convolution
- Blurring, sharpening, edge detectors

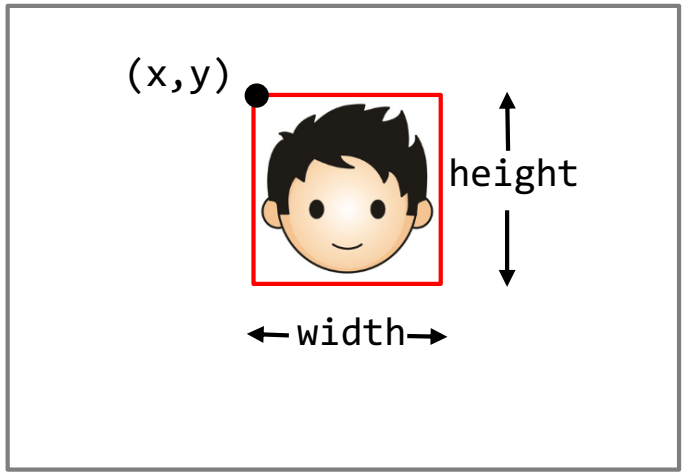
Image features

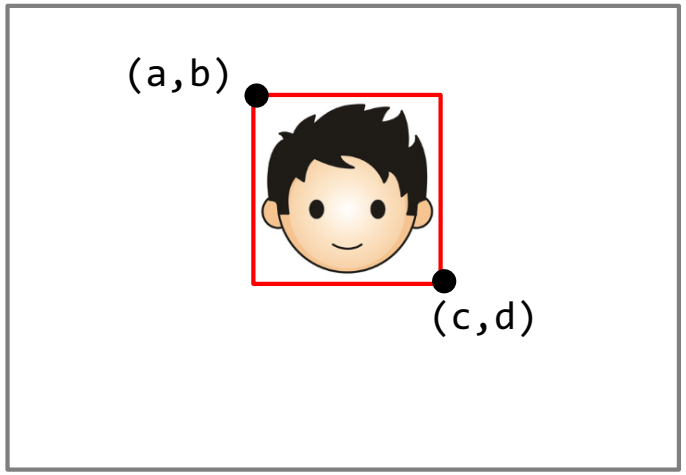
- Intensity (Haar-like features)
- Edges
- Corner points
- Texture
- Colour

Face detector bounding boxes

When you run the face detector, it produces a matrix bbox. Each row of this matrix is a bounding box of a detected face in the form

```
[x, y, width, height]
```





for the *i*th box

```
a = bbox(i, 1);
b = bbox(i, 2);
c = a+bbox(i, 3);
d = b+bbox(i, 4);
```

Lab 3, Task 1: Solution 1 (blur faces)

Approach 1

- Run the face detector
- Loop over each bounding box
- Extract a face from the image
- Blur the face
- Put the blurred face back in the image

```
I = imread('Class.jpg');
FaceDetector = vision.CascadeObjectDetector();
bbox = step(FaceDetector, I);
N = size(bbox, 1);
K = fspecial('disk', 20);
for i=1:N
    % Extract ith face
    a = bbox(i, 1);
    b = bbox(i, 2);
    c = a+bbox(i, 3);
    d = b+bbox(i, 4);
    F = I(b:d, a:c, :);
    %K = fspecial('disk', 20);
    K = fspecial('disk', bbox(i, 3)/5);
    % Blur face F with kernel K to produce R
    R = imfilter(F,K,'replicate');
    % Overwrite I
    I(b:d, a:c, :) = R;
end
imshow(I);
```

Lab 3, Task 1: Solution 1

Approach 2

- Blur the entire image
- Run the face detector
- Loop over each bounding box
- Extract a face from the blurred image
- Put the blurred face back in the image

```
I = imread('Class.jpg');
K = fspecial('disk', 20);
R = imfilter(I,K,'replicate');
FaceDetector = vision.CascadeObjectDetector();
bbox = step(FaceDetector, I);
N = size(bbox, 1);
for i=1:N
    % Extract ith face
    a = bbox(i, 1);
    b = bbox(i, 2);
    c = a+bbox(i, 3);
    d = b+bbox(i, 4);
    % Overwrite I
    I(b:d, a:c, :) = R(b:d, a:c, :);
end
imshow(I);
```

Optical Character Recognition

- https://uk.mathworks.com/help/vision/examples/recognize-text-using-optical-character-recognition-ocr.html
- https://uk.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html

Optical Character Recognition

```
I = imread('/users/sepehr/desktop/OCRTest.png');
%I = imresize(I,.4);
IG = rgb2gray(I);
IGB = IG>210;

blobAnalyzer = vision.BlobAnalysis('MaximumCount', 500);

% Run the blob analyser to find connected components
[area, centroids, roi] = step(blobAnalyzer, IGB); % ( OR CAN USE OCR FUNCTION TOO)
fori = 2 : size(area,1)
   %roi = results.Words{i}

   wordBBox = roi(i,:);
   % Show the location of the word in the original image
   ifwordBBox(3)>150
       figure;
       hold
       Iname = insertObjectAnnotation(I,'rectangle', wordBBox,i);
       imshow(Iname);
       wordBBoxInterest = wordBBox;
   end
end
box =
I(wordBBoxInterest(2):wordBBoxInterest(2)+wordBBoxInterest(4),wordBBoxInterest(1):wordBBoxInterest(1)+w
ordBBoxInterest(3));
[sizex, sizey] = size(box);
box = box(floor(sizex/4) : floor(sizex*3/4), floor(sizey/4):floor(sizey*3/4));
box = imresize(box,0.5);
resultsNew = ocr(box,'TextLayout','Block');
text = deblank( {resultsNew.Text(1:2)} );
img = insertObjectAnnotation(I,'rectangle', wordBBoxInterest, text);
figure;
imshowpair(I, img,'montage');
```

CITY
UNIVERSITY OF LONDON
EST 1894

Overview of today's lecture

Image segmentation

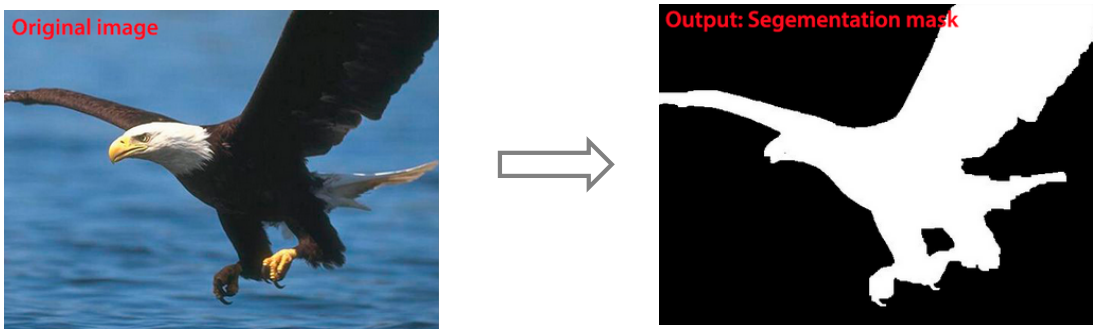
- Thresholding
- Clustering
- Watershed
- Active contours
- Graph cuts

Quantifying results

How to tell if we have a good result?

Segmentation

- Image segmentation partitions an image into regions (aka *segments*).



- You can think of image segmentation as assigning a label to each pixel in the image. In the case above, there are two labels: *bird* (foreground, white) and *non-bird* (background, black).

Segmentation

- Essentially, segmentation is a grouping problem. Depending on the image, this can be difficult! Humans are adept at visual grouping, as explored in Lecture 1.

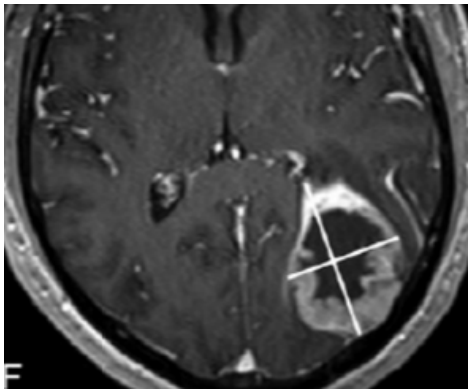
Segmentation

- Segmentation has a variety of practical uses

cut out

recolouring

compositing

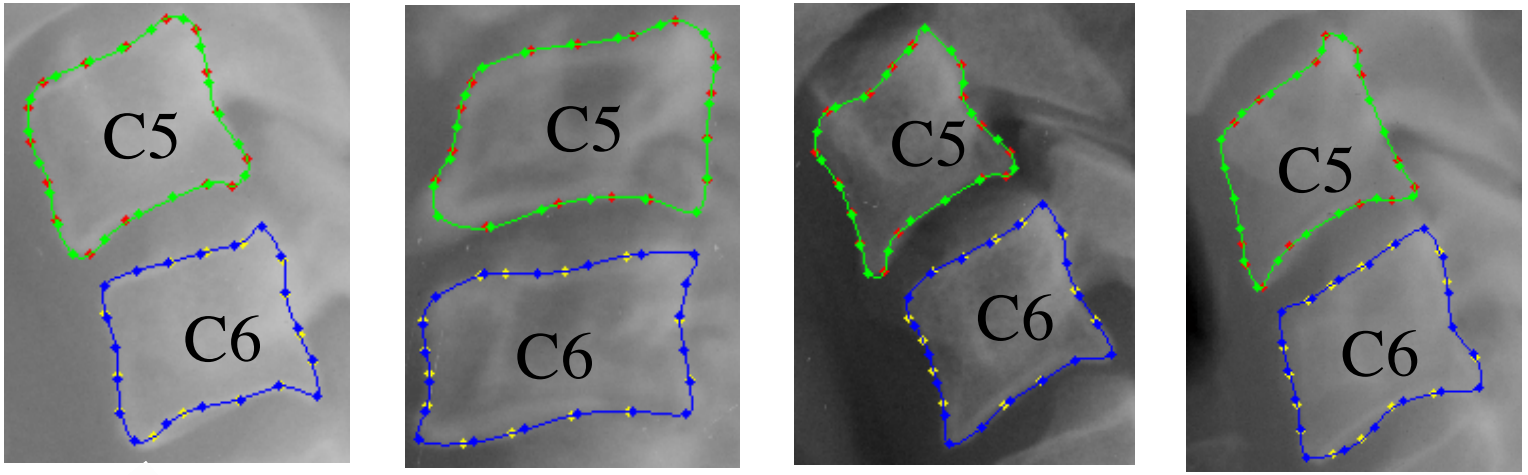


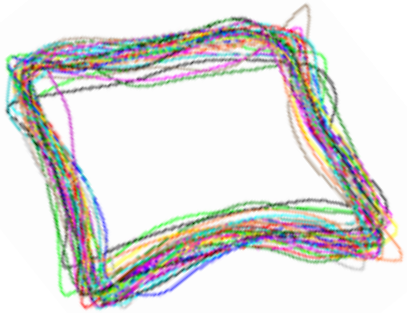
measurement

https://www.youtube.com/watch?v=U1sZbeO41Wc

Segmentation

- The boundary of a segment provides a *shape*.



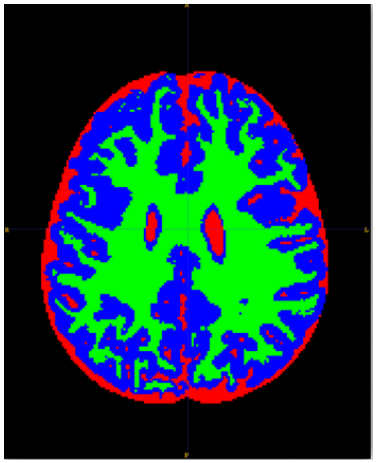


A collection of aligned C4 vertebra shapes. From this, we can do all sorts of interesting things

- Population statistics
- Mean, variation (principal component analysis)
- (more)

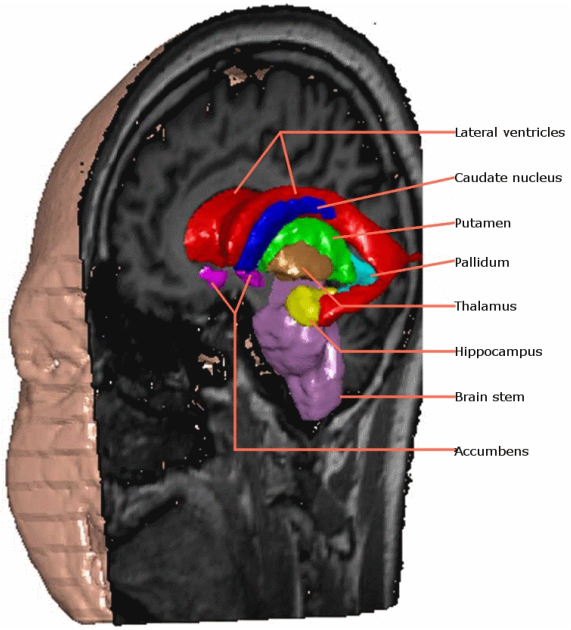
Segmentation

- There can be any number of labels



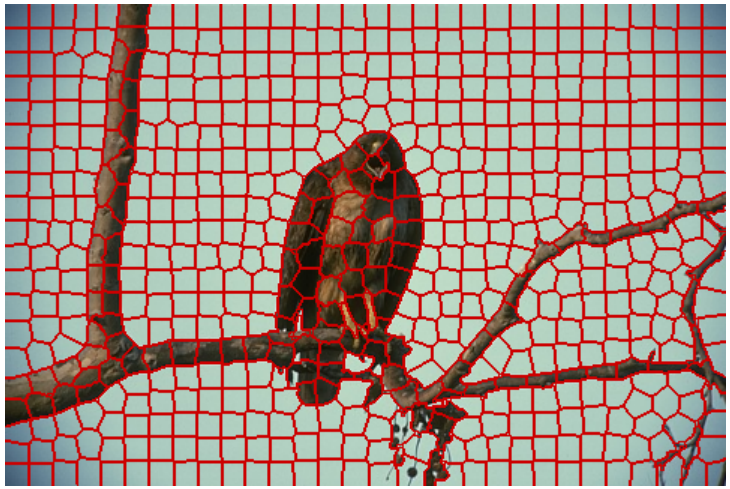
Four labels:
1. Non-brain
2. Grey matter
3. White matter
4. CSF (Cerebrospinal fluid)

- Different types of data; more than two dimensions



Under and over segmentation

- An image that has been broken into too many segments is considered to be *over-segmented*



- Not enough segments is considered *under-segmented*

Pink flowers merged together

Bird counting

- How would you determine the number of birds in this image?

http://350sav.fotomaps.ru/flock-of-birds.php

- One way is to count the dark blobs. First we need to *find* the dark blobs.
- Let's load the image and convert to grayscale

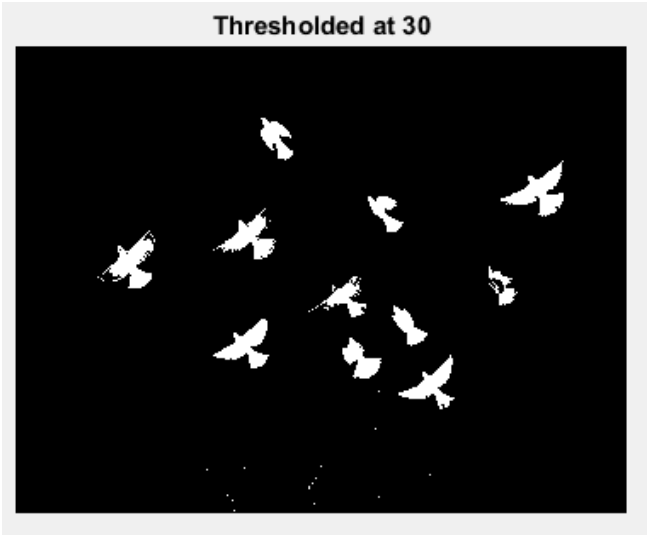
```
I = rgb2gray(imread('birds.png'));
```

Thresholding

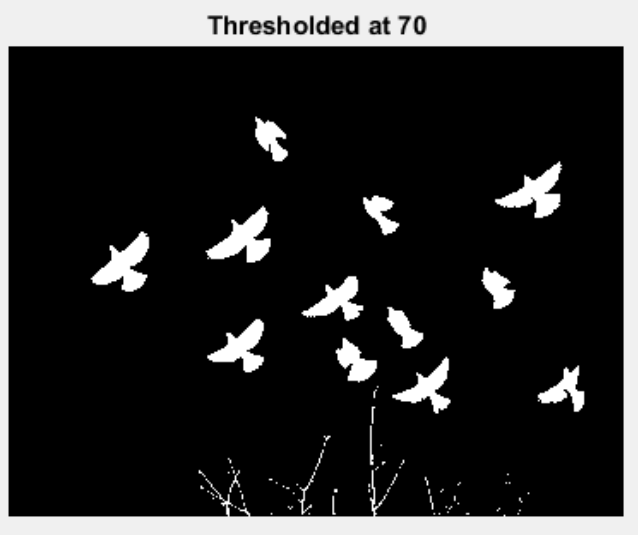
- The simplest form of segmentation is to *threshold* an image. This assigns a value of 1 to any pixel that satisfies the threshold, otherwise, 0. A *binary* image is produced.

$$B(x,y) = \begin{cases} 1, & I(x,y) < T \\ 0, & \text{otherwise} \end{cases}$$

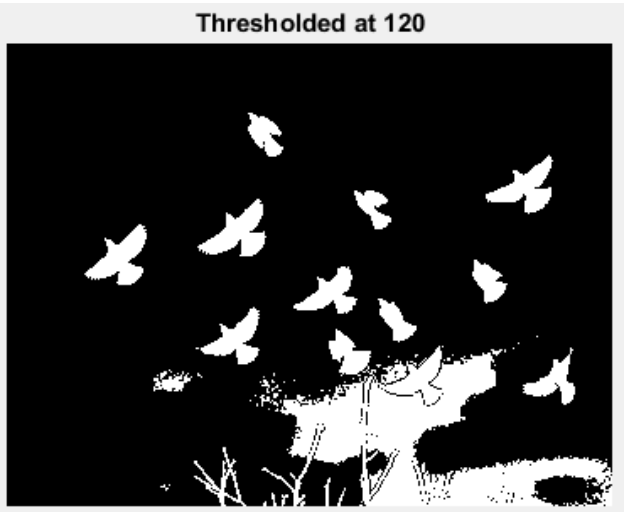
>T if looking for bright regions




```
B = I < 30;
figure; imshow(B);
title('Thresholded at 30');
```

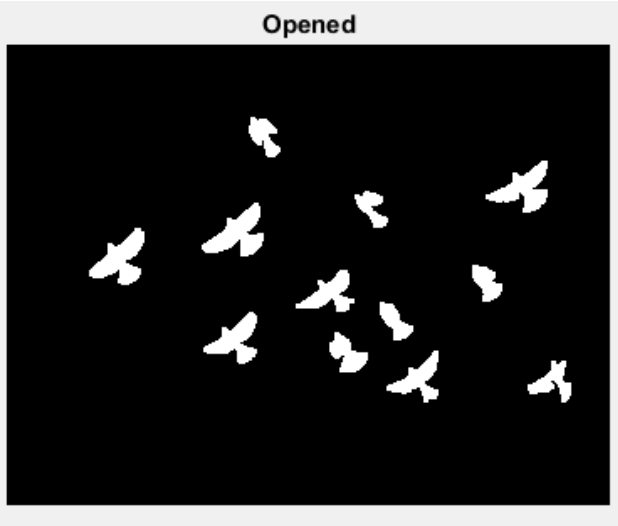


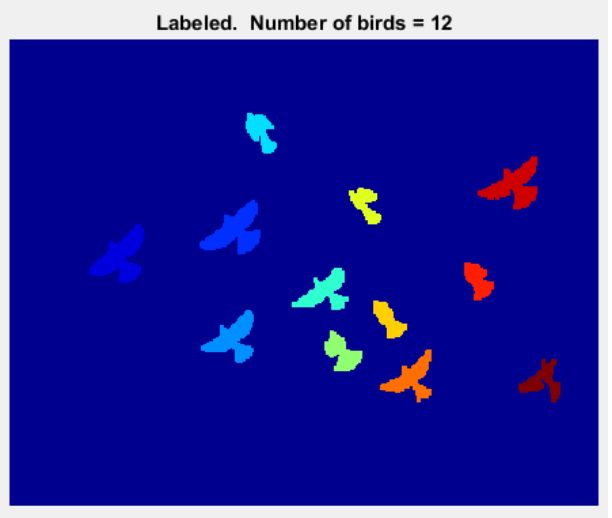

```
B = I < 70;
figure; imshow(B);
title('Thresholded at 70');
```




```
B = I < 120;
figure; imshow(B);
title('Thresholded at 120');
```

Bird counting






```
I = rgb2gray(imread('birds.png'));
B = I < 70;
figure; imshow(B); title('Thresholded at 70');
E = imopen(B, ones(3,3));
figure; imshow(E); title('Opened');
L = bwlabel(E);
figure; imagesc(L); colormap('jet');
title(['Labeled.  Number of birds = ' num2str(max(L(:)))]);
```

`bwlabel` (I,n): Labels n-connected components in 2-D binary image I.

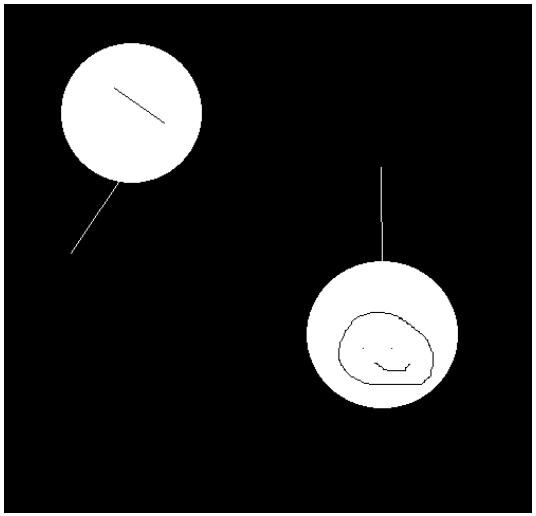
`Imopen:` The morphological open operation is an erosion followed by a dilation.

Mathematical morphology

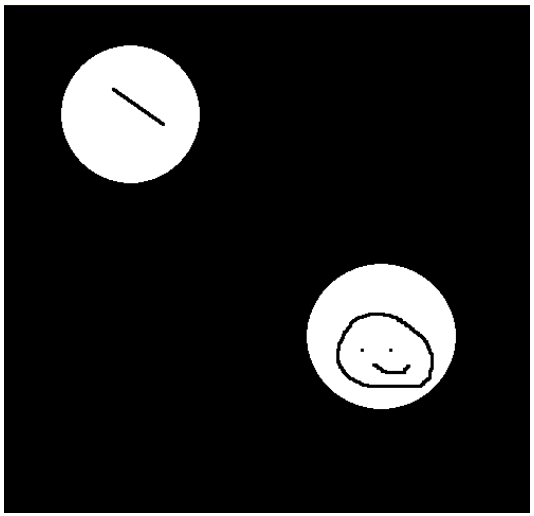
- A related class of nonlinear image processing techniques is known as mathematical morphology.
- These techniques also use kernels, which are called *structuring elements* in image processing.
- There are two basic operations, *erosion* and *dilation*.
- **Erosion is like a min filter** (over the structuring element extent on the image), and
- **Dilation is like a max filter**.
- These operations are often used on binary images, but can be used on grayscale images too.
- A ***closing*** is *dilation*, followed by *erosion*. It fills holes smaller than the structuring element.
- An ***opening*** is *erosion*, followed by *dilation*. It removes objects smaller than the structuring element.
- Mathematical morphology may be useful in “cleaning up” segmentations.

- https://uk.mathworks.com/help/images/morphological-dilation-and-erosion.html

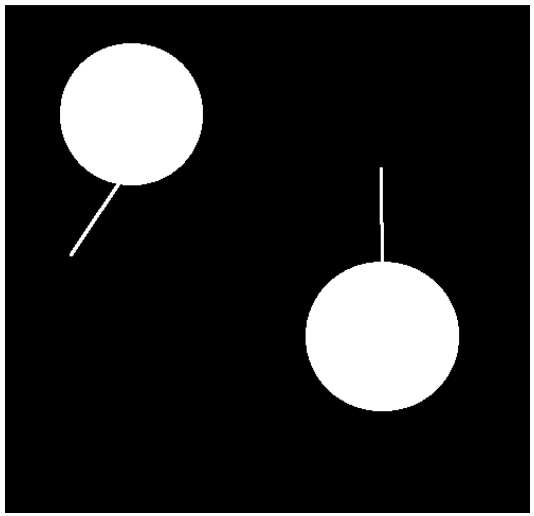
Erosion and dilation



Original image

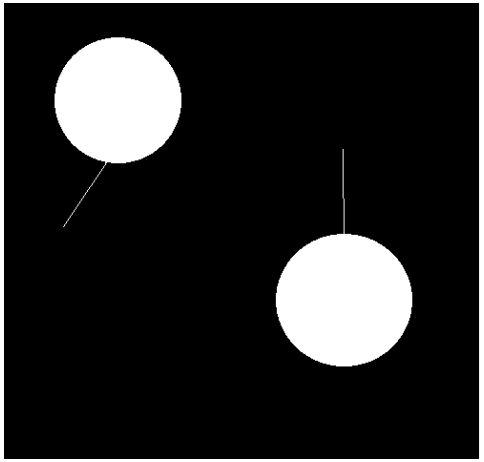


```
J = imerode(I, ones(3,3));
figure; imshow(J);
title('erosion');
```

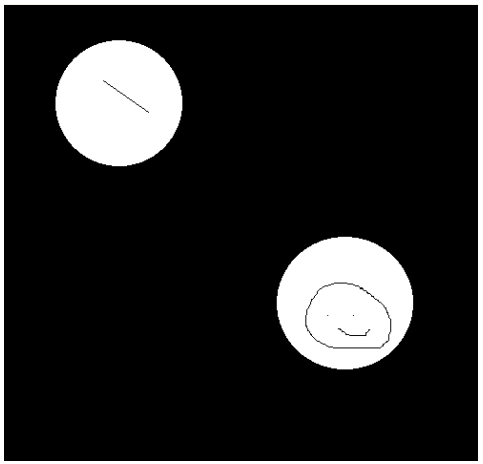


```
J = imdilate(I, ones(3,3));
figure; imshow(J);
title('dilation');
```

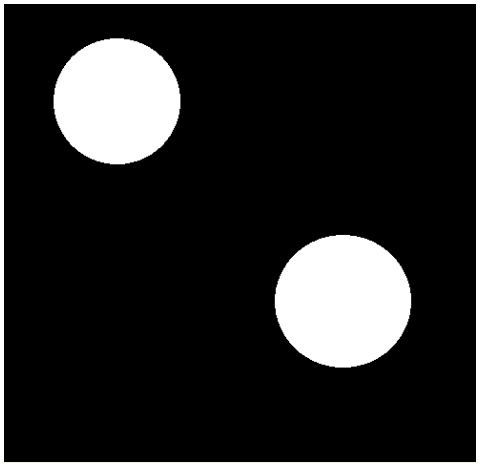
Closing and opening



```
J = imclose(I, ones(3,3));
figure; imshow(J);
```



```
J = imopen(I, ones(3,3));
figure; imshow(J);
```

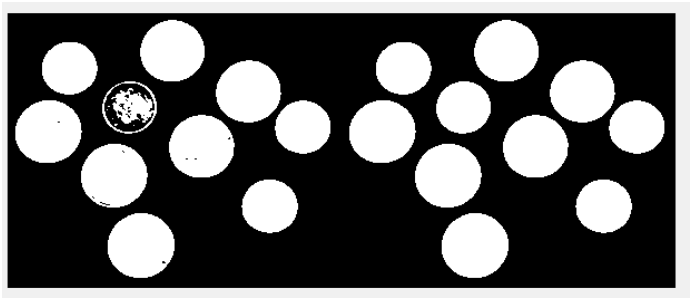


```
K = ones(3,3);
J1 = imopen(I, K);
J2 = imclose(J1, K);
figure; imshow(J2);
```

Some other handy morphological functions

- imfill can be used to fill holes in each binary image region

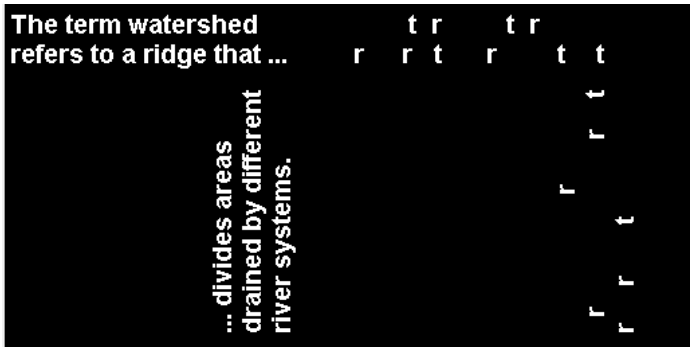
```
I = imread('coins.png');
T = graythresh(I); % otsu threshold
J = I > 255*T;
K = imfill(J,'holes');
imshowpair(J,K,'montage');
```



J K

- bwareafilt can be used to filter regions in a binary image based on size (in pixels)

```
I = imread('text.png');
J = bwareafilt(I,[40 50]);
imshowpair(I,J,'montage');
```

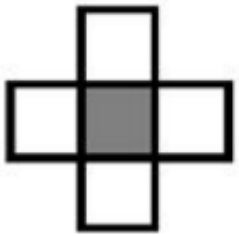


I J

- You can also use regionprops to get the properties of each region, like its area, perimeter, etc. Based on this, you can filter regions to keep those within prescribed limits (Lab 4).

Going beyond colour

- When segmenting a pixel **p**, we might consider pixels in **p**'s neighbourhood.
- This requires a definition what a neighbouring pixel is.

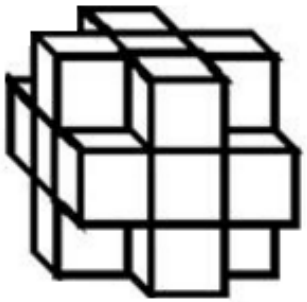


4-neighbourhood

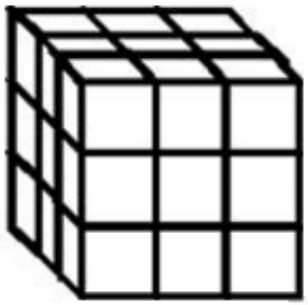
8-neighbourhood

Common 2D neighbourhoods

6-neighbourhood



18-neighbourhood



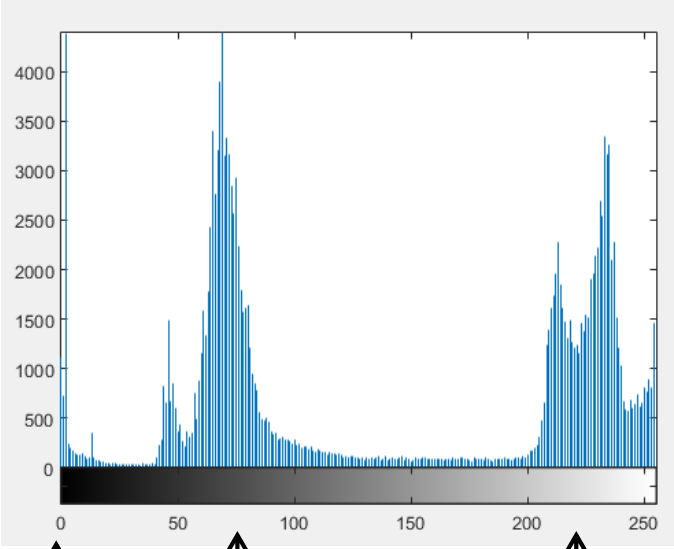
26-neighbourhood

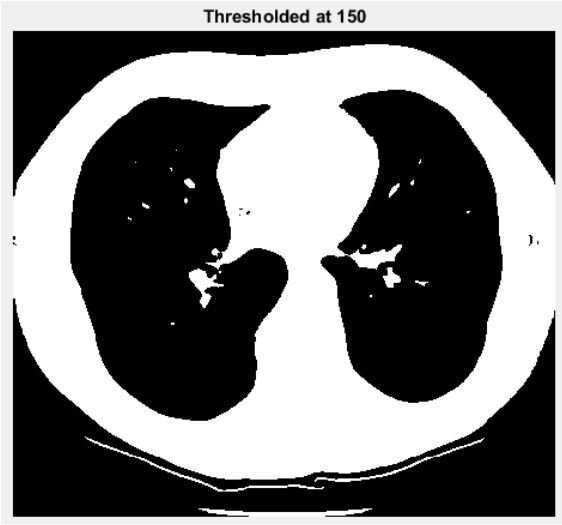
Common 3D neighbourhoods

https://www.linkedin.com/pulse/20140506202304-14648565-3d-is-not-1-5-times-2d

Finding a good threshold

- The success of thresholding often hinges on finding a good threshold *T*.
- Recall the histogram h(I) of an image. It counts how many pixels of a particular intensity are in the image.






```
I = rgb2gray(imread('lung.png'));
imhist(I, 256);
J = I > 150;
figure; imshow(J);
title('Thresholded at 150');
```

Skin segmentation

- Segmentation of skin is a notoriously difficult problem, given the variation observed in practice. One can perform simple skin detectors using *colour thresholding*.
- The Y′UV model defines a color space in terms of one luma (Y′) and two chrominance (UV) components.

[Chai et al.]

Convert to YUV
Skin = (77 <= U <= 127) and
(133 <= V <= 173)

[Al-Tairi et al.]

Convert to YUV
Skin = (80 < U < 130) and
(136 < V < 200) and
(V > U) & (R > 80) and
(G > 30) & (B > 15) and
|R-G| > 15

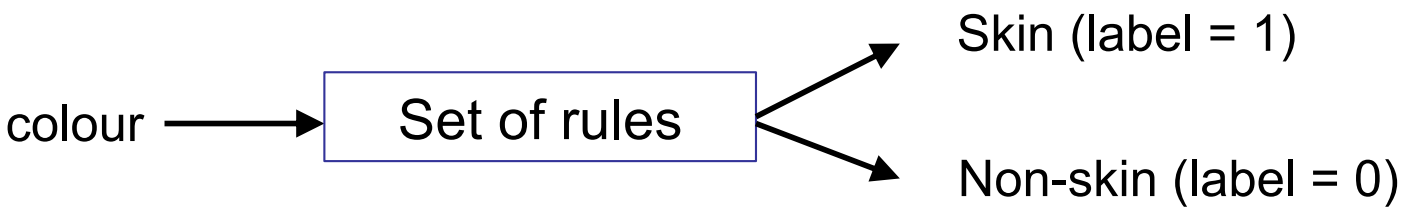
https://en.wikipedia.org/wiki/YUV

```
I = imread('faces.png');
YUV = rgb2ycbcr(I);
U = YUV(:, :, 2); V = YUV(:, :, 3);
R = I(:, :, 1); G = I(:, :, 2); B = I(:, :, 3);
[rows, cols, planes] = size(I);

% Chai et al.
skin = zeros(rows, cols);
ind = find(77 <= U & U <= 127 & ...
      133 <= V & V <= 173);
skin(ind) = 1;
figure;  subplot(3, 1, 1); imshow(I);
subplot(3, 1, 2); imshow(skin); title('Chai');

% Al-Tairi et al.
skin = zeros(rows, cols);
ind = find(80 < U & U < 130 & 136 < V & ...
      V <= 200 & V > U & R > 80 & G > 30 & ...
      B > 15 & abs(R-G) > 15);
skin(ind) = 1;
subplot(3, 1, 3); imshow(skin); title('Al-Tairi');
```

Example

Does this look familiar?

- We have a set of rules that label a pixel as skin or non-skin.
- More abstractly, we can think of this as

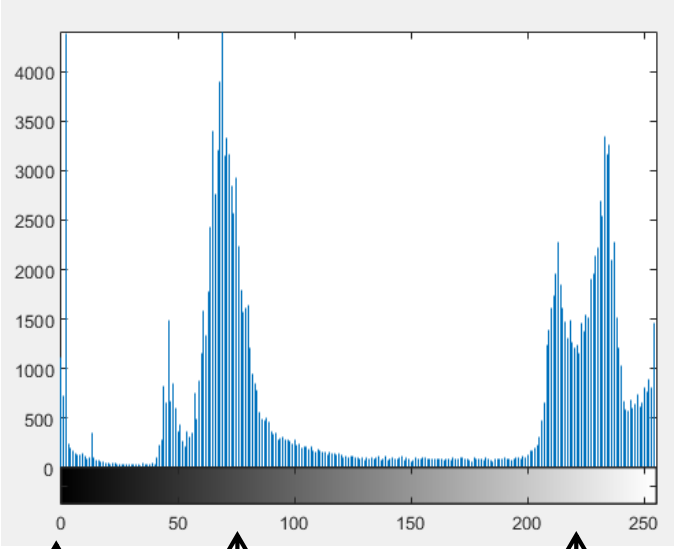


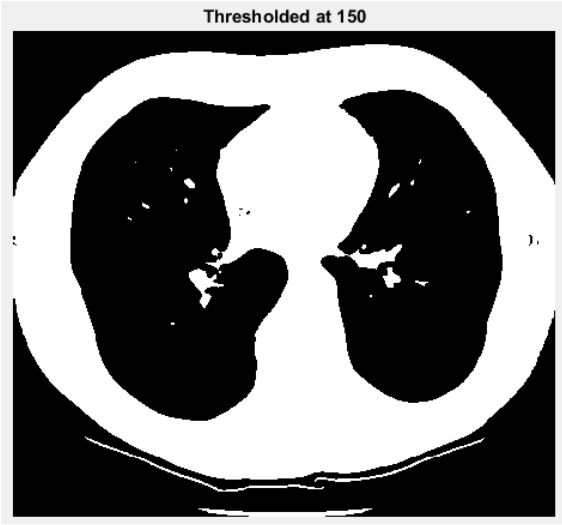
⇨ This is a rule-based classifier

In this case, the rules have been generated by a human by looking at many datasets. An alternative is to use *machine learning* to develop a supervised approach to skin detection. More on supervised learning in upcoming lectures.

Finding a good threshold

- The success of thresholding often hinges on finding a good threshold *T*.
- Recall the histogram h(I) of an image. It counts how many pixels of a particular intensity are in the image.

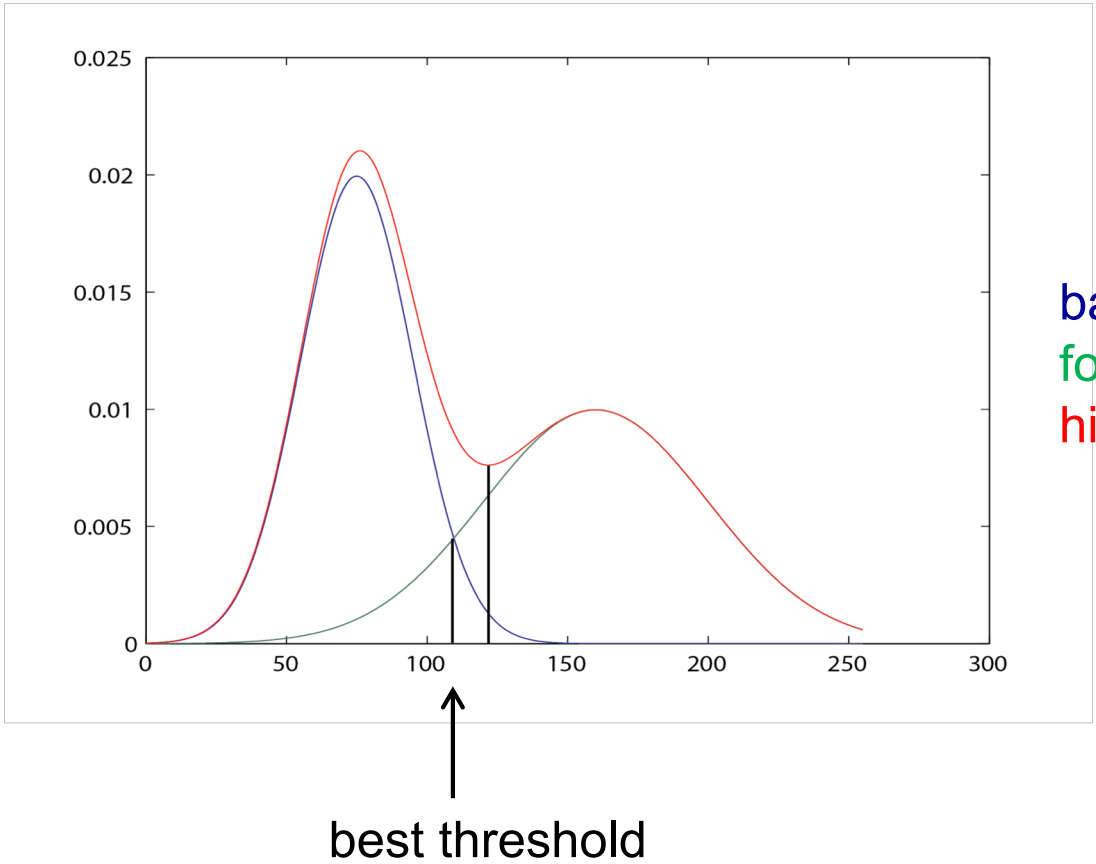





```
I = rgb2gray(imread('lung.png'));
imhist(I, 256);
J = I > 150;
figure; imshow(J);
title('Thresholded at 150');
```

Automatic thresholding methods

- Many automatic methods to determine the threshold assume a *bimodal* histogram, meaning there are two peaks (one for foreground, the other for background)



Optimal thresholding (Otsu)

- An approach was proposed by Otsu.
- Otsu's method finds the threshold *T* that minimises the intra-class variance, defined as

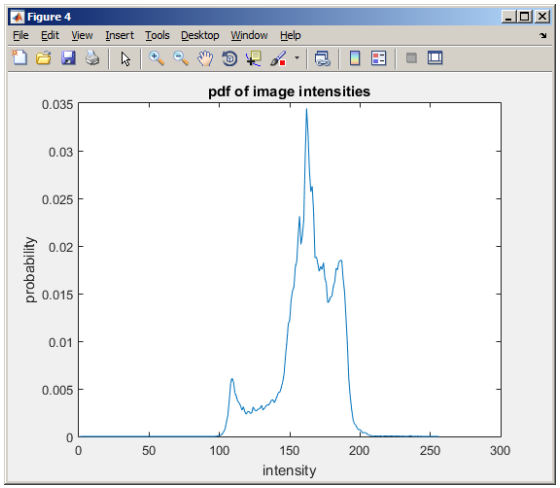
$$\sigma^2 = w_f \sigma_f^2 + w_b \sigma_b^2$$

where σ_f^2 and σ_b^2 are the variances of the foreground and background, and

$$w_b = \sum_{I=0}^{T} p(I) \qquad w_f = \sum_{I=T+1}^{255} p(I)$$

are weights formed by summing the histogram pdf over the background and foreground intensities, respectively.

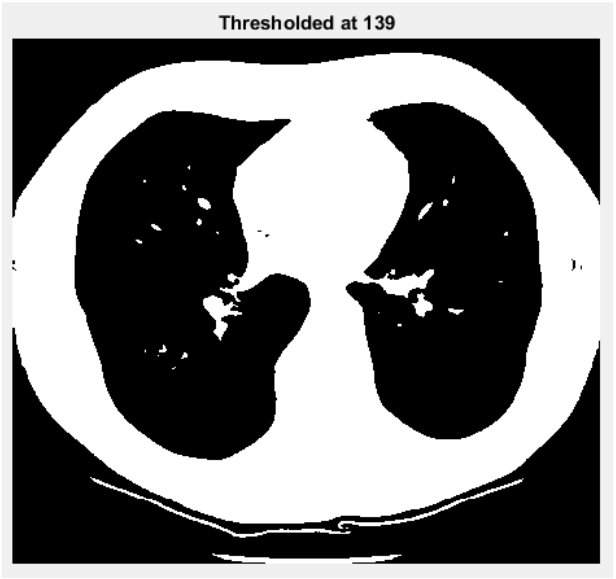
A histogram can be normalised and interpreted as a probability density function (pdf), representing the probability of a pixel having a particular brightness.



Optimal thresholding

1. Set T to some initial value, and determine foreground and background pixels.
2. Compute m_b, the mean of the background, and m_f, the mean of the foreground, based on the current value of T.
3. Set $T = 0.5 *(m_b + m_f)$
4. Go to step 2 until convergence (when T no longer changes).

```
I = rgb2gray(imread('lung.png'));
T = 50;
lastT = 0;
while (abs(T - lastT) > 1)
    lastT = T;
    mf = mean(I(I>T));
    mb = mean(I(I<=T));
    T = 0.5*(mb+mf);
end
```



In this example, the loop executes 5 times, with T starting at 50, and going to 81.9, 126.7, 138.2, 139.3, 139.4

- In Matlab, Otsu thresholding is implemented with the function `grayThresh`, which returns a threshold in the range of 0 to 1.
- For the previous example, `level = 255*graythresh(I)` returns a value of 139.

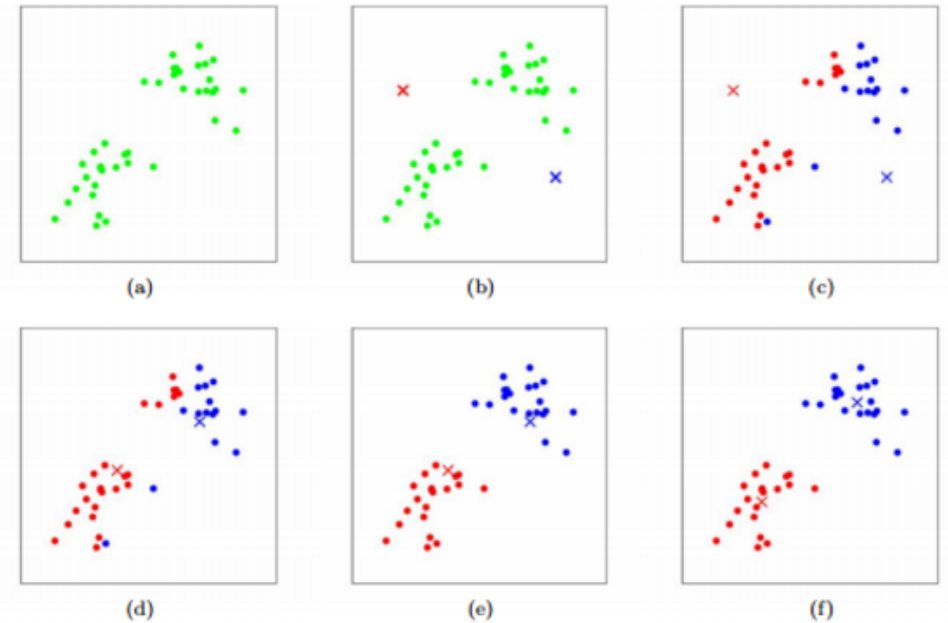
K-means

- The optimal thresholding method described on the previous slide is an example of the K-means algorithm, where K = 2.
- K-means is a clustering technique, in this case grouping pixels of similar intensity into two groups, or *clusters*.
- However, K-means can be used to cluster data into any number (K) of classes. The data can be based on intensity, colour, or any attribute associated with a pixel (e.g., texture, depth, etc.)
- It requires K to be provided in advance.

K-Means is a method of Unsupervised Machine Learning (will be revisited in next lectures)…

K-means

- Algorithm:
  1. Start with a set of N points (green points below)
  2. Set K means (x's below), for example, randomly
  3. Assignment: Assign the N points to their closest mean
  4. Update: Calculate new means from the assigned points
  5. Go to 3 until convergence (means / assignments no longer change)

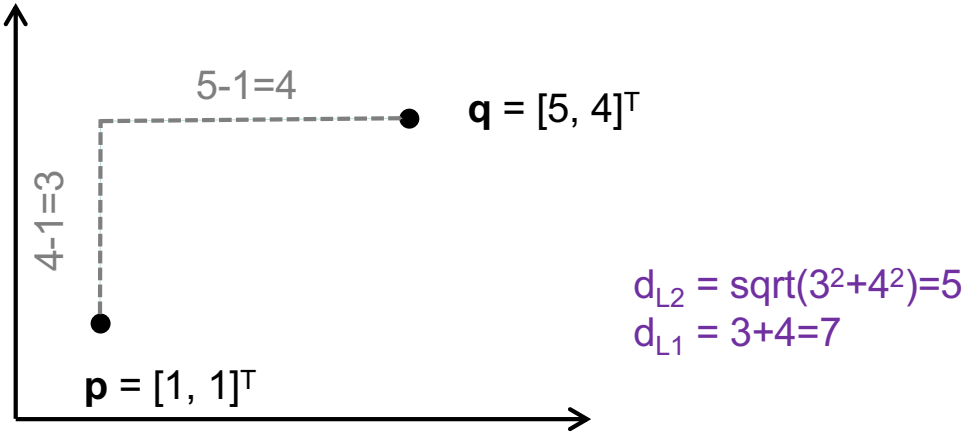


Issues with K means

- What is K?
  - In some applications, there is an obvious K. In others, it may be unclear.
- How to initialise the means? Potentially different results each time the algorithm is run
  - One can rerun the algorithm multiple times and merge the results
- Dealing with high dimensional data
  - Finding nearest neighbours can be slow (kd tree can help) https://en.wikipedia.org/wiki/K-d_tree
- How do you measure distance?
  - Normally the L2 norm (Euclidean distance) is selected, but there are other options, for example, the L1 distance (aka CityBlock distance).

$$d_{\mathrm{L2}}(\mathbf{p},\mathbf{q}) = \sqrt{\sum_i (p_i - q_i)^2}$$

$$d_{\mathrm{L1}}(\mathbf{p},\mathbf{q}) = \sum_i |(p_i - q_i)|$$



K-means

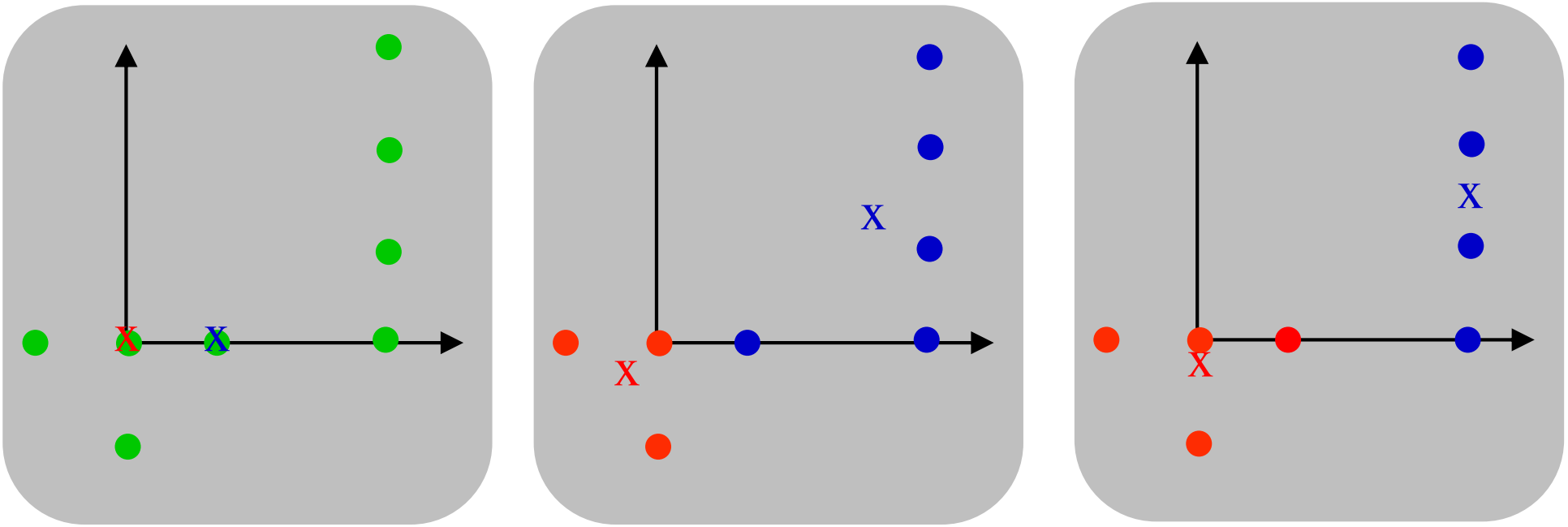
- Matlab has support for K means using the kmeans function. It requires an NxM matrix. Here, N is the number of pixels, and M are the features (colours) used in the grouping. This can be achieved using the reshape function.

```
I = imresize(imread('road_sky.jpg'), 0.25);
imshow(I);
[rows, cols, planes] = size(I);

% Run k means on the RGB data, k = 4
R = double(reshape(I, rows*cols, 3));
[clusterID, clusterCentre] = kmeans(R, 4);

% Reshape back to an image and display using a colormap
clusterID = reshape(clusterID, rows, cols);
figure; imshow(clusterID,[]), title('image labelled by cluster index');
```

`colormap(gca, clusterCentre/255);` `%` **gca** returns the current axes or chart for the current figure, which is typically the last one created or clicked with the mouse.

Exercise

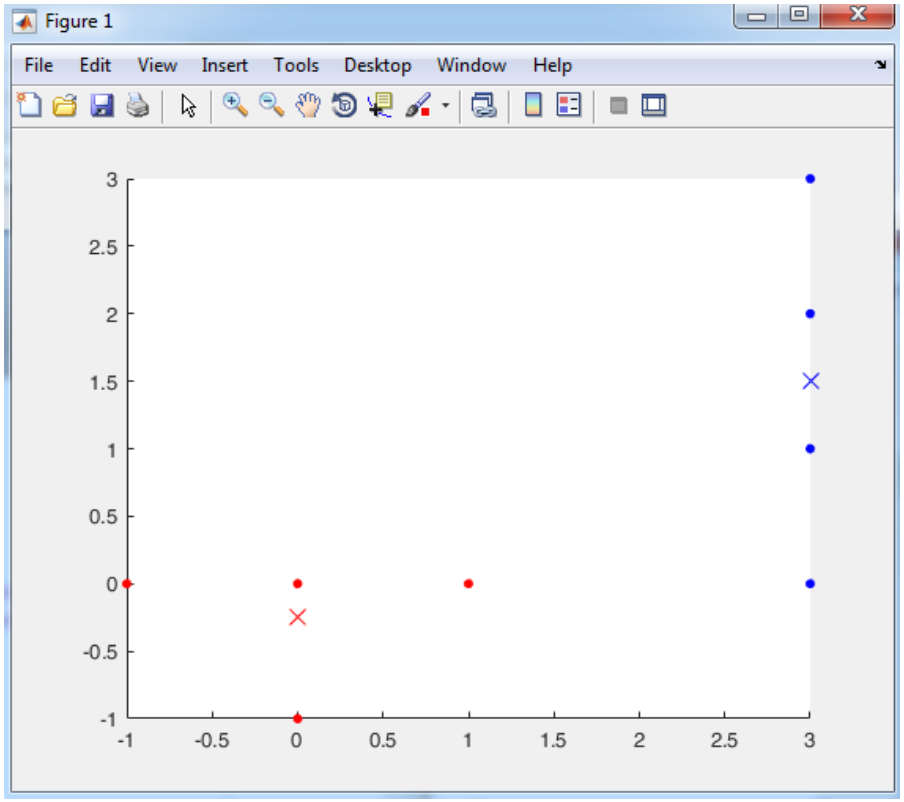
- Implement the k-means algorithm for k = 2 for the following initialisation and data consisting of eight points, using Euclidean distance



1. *Start with a set of N points (green points)*
2. *Set K means (x's)*
3. *Assignment: Assign the N points to their closest mean*
4. *Update: Calculate new means from the assigned points*
5. *Go to 3 until convergence (means / assignments no longer change)*

In code

```
f = [-1, 0; 0, 0; 0, -1; 1, 0; 3, 0; 3, 1; 3, 2; 3, 3];
[clusterID, clusterCentre] = kmeans(f, [], 'Start', [0, 0; 1, 0]);
scatter(f(:,1),f(:,2), 20, clusterID, 'filled');
colormap(gca, [1, 0, 0; 0, 0, 1])
hold on;
plot(clusterCentre(1,1),clusterCentre(1,2),'rx','MarkerSize',10);
plot(clusterCentre(2,1),clusterCentre(2,2),'bx','MarkerSize',10);
```



https://uk.mathworks.com/help/matlab/ref/scatter.html

K means

- Different results for different K

K = 2

K = 3

K = 4

K = 5

K means

- Different results for different *runs* due to random initialisation.

K = 4

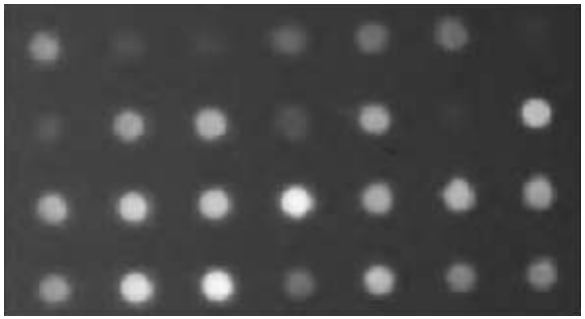
K = 4

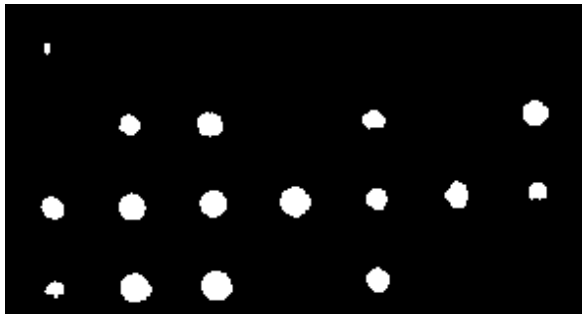
- Replicates (running the algorithm multiple times) can produce more consistent results

```
[clusterID, clusterCentre] = kmeans(R, 4, 'Replicates', 3);
```

Hysteresis thresholding

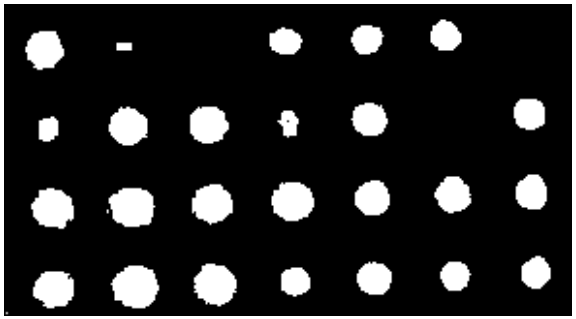
- Hysteresis thresholding applies *two thresholds* to achieve a segmentation.
- A region will be labelled as foreground if
  - It contains connections to a pixel with intensity greater than T_{high} *and*
  - All its pixels have an intensity greater than T_{low},
- Goal: Perform a segmentation of the brighter blobs





$I>T_{high}$

Finds brighter blobs, but shapes are not correct.



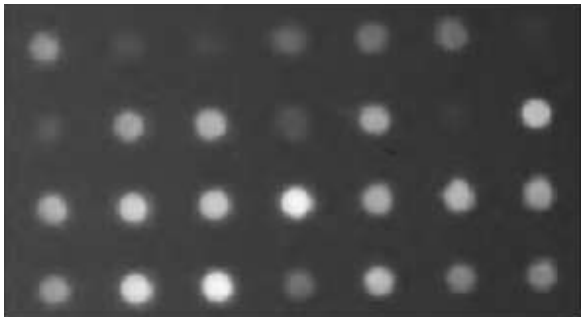
$I>T_{low}$

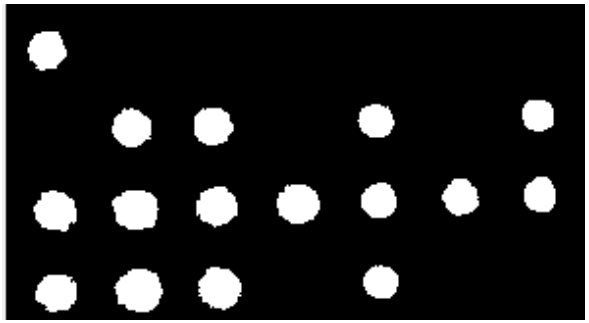
Shapes are (mostly) correct but darker blobs included.

- Hysteresis thresholding will select brighter blobs using T_{high}, and for those blobs, use the shape from T_{low}.

Hysteresis thresholding

```
I = imread('dots.png');
I = double(rgb2gray(I));
figure; imshow(I/255);

Thigh = 150;
Tlow = 80;
H = I > Thigh;
figure; imshow(H);
title('High threshold');

L = I > Tlow;
figure; imshow(L);
title('Low threshold');

labels = bwlabel(L); %label connected components
validLabels = unique(labels(H));
J = ismember(labels, validLabels);
figure; imshow(J);
title('Hysteresis tresholding result');
```



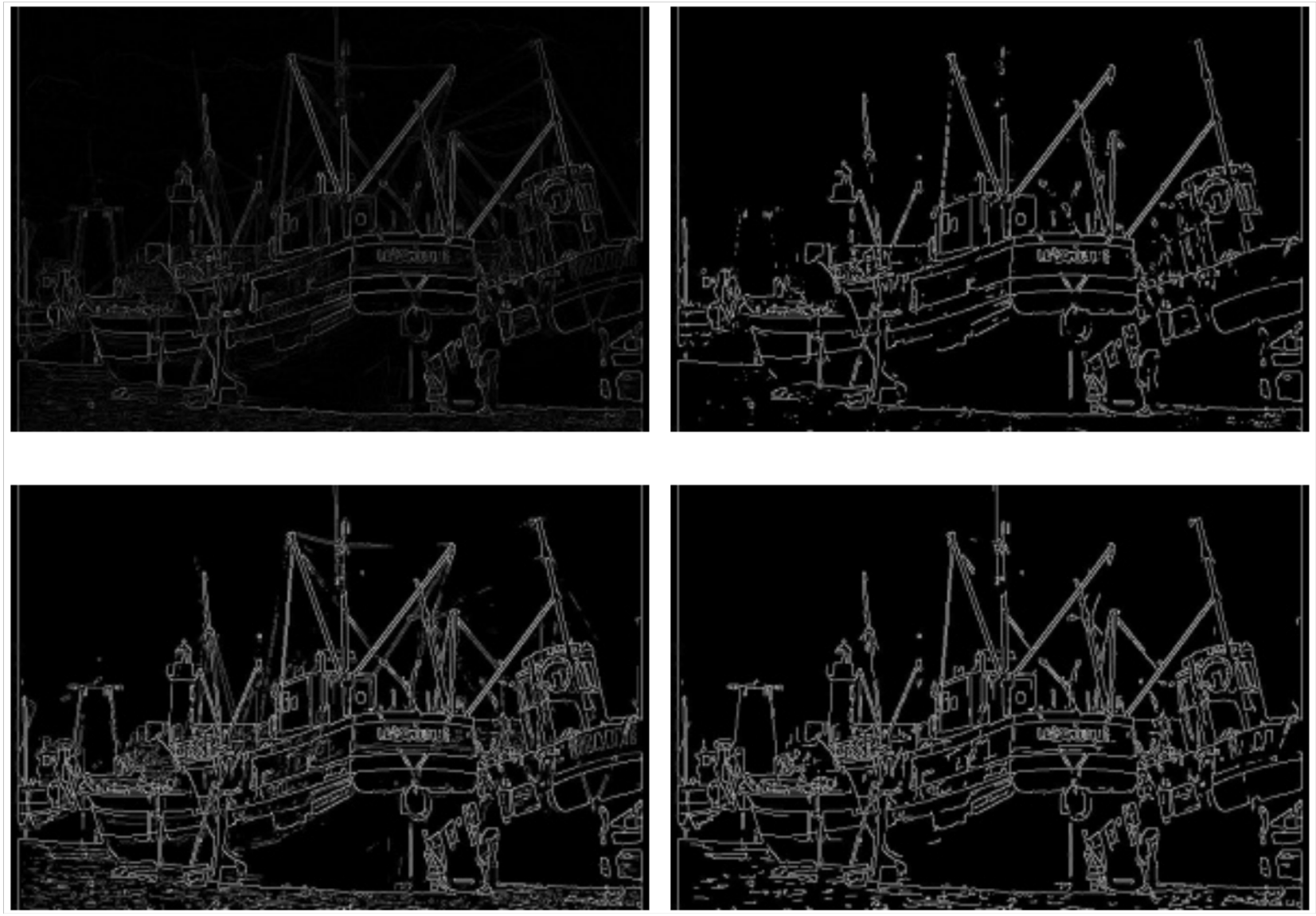


This code labels the regions detected using the low thresholded image (L), and uses the high thresholded image (H) to select valid regions. **The pixels from L that are part of a valid region (are connected to H) are retained.**

https://uk.mathworks.com/help/matlab/ref/unique.html

https://uk.mathworks.com/help/matlab/ref/ismember.html

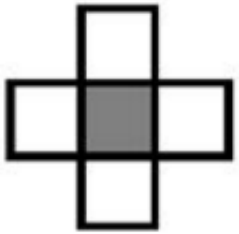
Hysteresis Thresholding in Edge Detection



Low thresholded edges which are connected to high thresholded edges are retained. Low thresholded edges which are non connected to high thresholded edges are removed

Going beyond colour

- Colour (or intensity) is a powerful feature and typically used in image segmentation. However, typically there is additional contextual information we can use, often encoded *spatially*.
- When segmenting a pixel **p**, we might consider pixels in **p**'s neighbourhood.
- This requires a definition what a neighbouring pixel is.

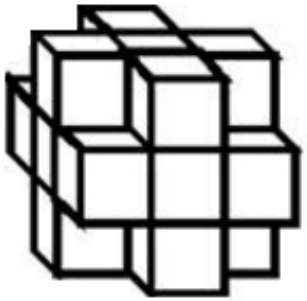


4-neighbourhood

8-neighbourhood

Common 2D neighbourhoods

6-neighbourhood



18-neighbourhood

26-neighbourhood

Common 3D neighbourhoods

https://www.linkedin.com/pulse/20140506202304-14648565-3d-is-not-1-5-times-2d

Distance transform

- In Matlab, a *distance transform* computes the distance from closest marked pixels in a binary image. It is implemented with the bwdist function.
- For example, consider the code below:

```
B = zeros(5,5);
B(2,2) = 1;
B(4,4) = 1;
[D, index] = bwdist(B);
```

```
B =
     0     0     0     0     0
     0     1     0     0     0
     0     0     0     0     0
     0     0     0     1     0
     0     0     0     0     0
```

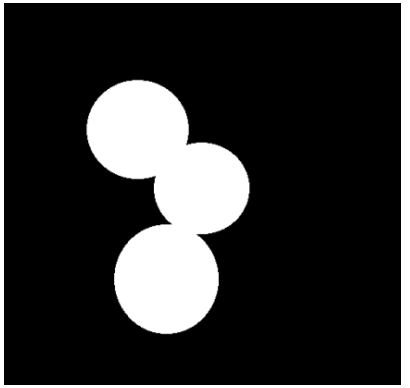
```
D =
    1.4142    1.0000    1.4142    2.2361    3.1623
    1.0000         0    1.0000    2.0000    2.2361
    1.4142    1.0000    1.4142    1.0000    1.4142
    2.2361    2.0000    1.0000         0    1.0000
    3.1623    2.2361    1.4142    1.0000    1.4142
```

```
index =
     7     7     7     7     7
     7     7     7     7    19
     7     7     7    19    19
     7     7    19    19    19
     7    19    19    19    19
```

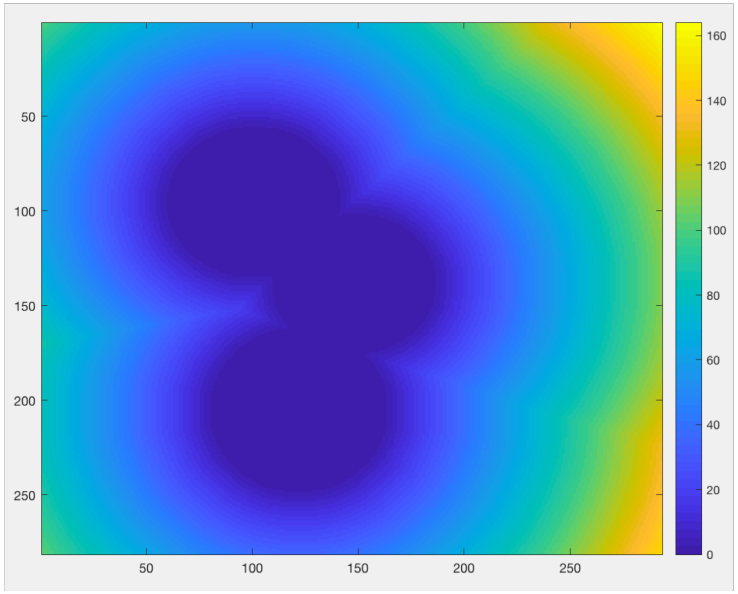
- For each value in B, `bwdist` returns the distance to the closest 1. It also returns in the index into B where the closest 1 is located.

Distance transform

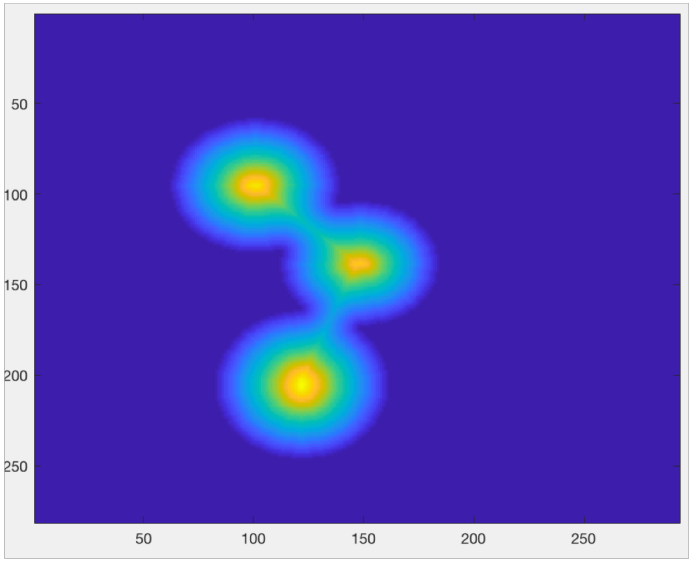
- A visual example



```
B = logical(rgb2gray(imread('threeCircles.png')));
imshow(B);
```




```
D1 = bwdist(B);
imagesc(D1);
colorbar;
```




```
D2 = bwdist(~B);
imagesc(D2);
```

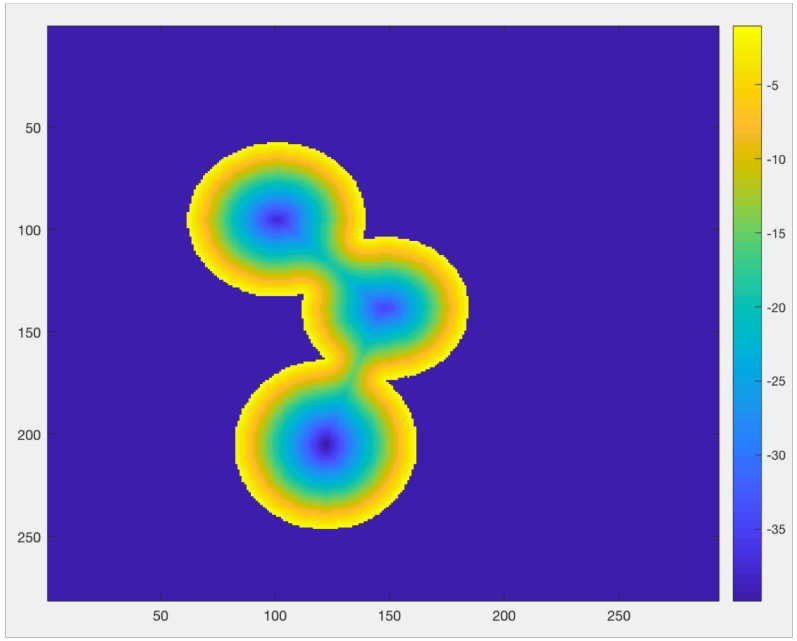
imagesc: Display image with scaled colours

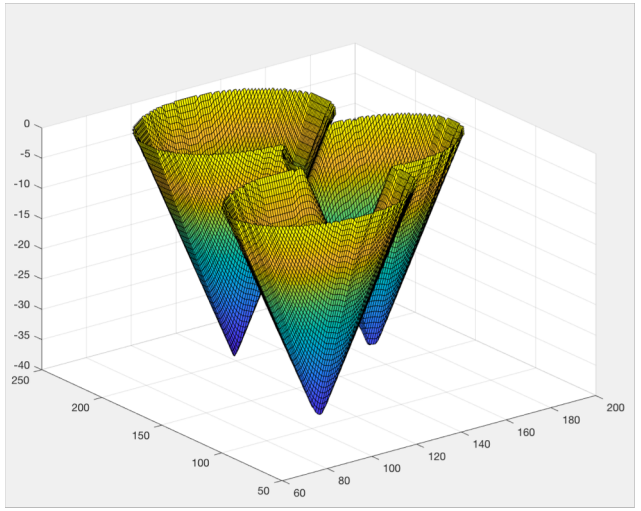
https://uk.mathworks.com/help/matlab/ref/imagesc.html

Watershed segmentation

- Consider an image D as a topographical surface, where the intensity of the image is mapped to a height.

```
D = -D2;
D(~B) = -Inf;
imagesc(D);
colorbar;
```



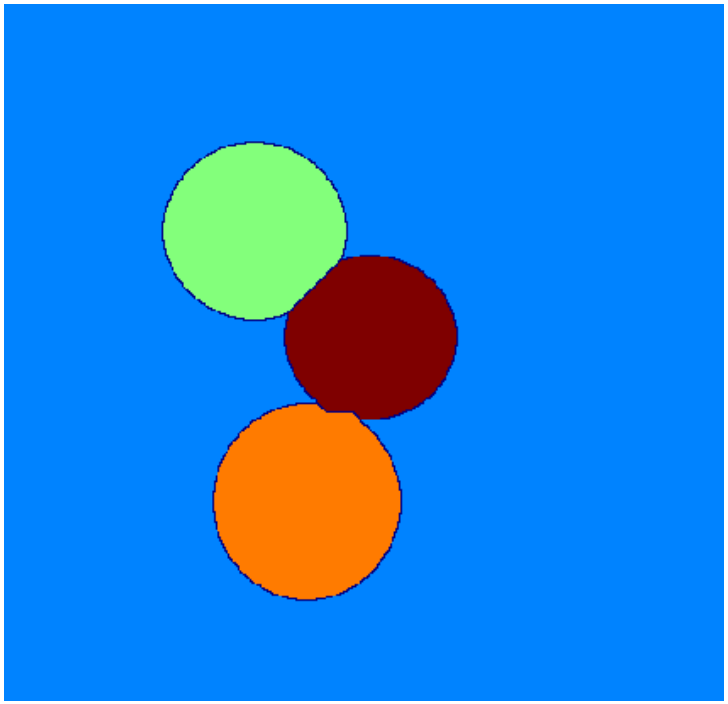



```
surf(D);
```

- Imagine it starts raining. Excluding the background, water will start to accumulate in the basins (lowest points). We can think of each basin as a region. As water is continually added, eventually regions will collide at *ridge lines*.

Watershed segmentation

- The `watershed` function in Matlab computes these ridge lines, and returns a matrix L with labels grouping pixels into regions based on the catchment basin.

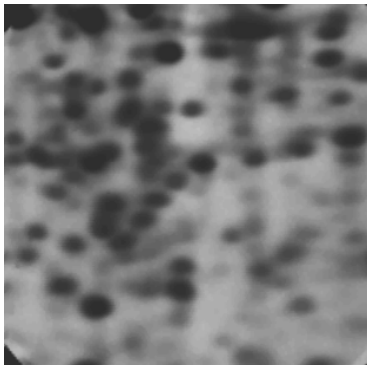


```
L = watershed(D);
imshow(L, []);
colormap('jet');
```

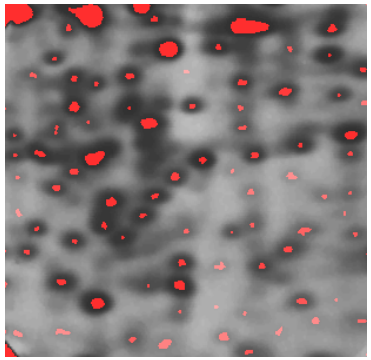
- Here, multiple regions are detected, even though circles are connected.

Marker-controlled watershed

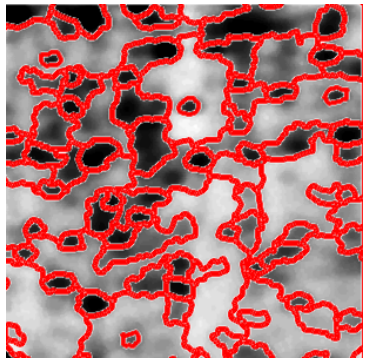
- On a grayscale image, one typically applies watershed to the distance transform of an edge map. However, this will result in *over-segmentation*.
- Instead, one can provide markers (also known as seeds). Each seed defines a catchment basin (region in the final segmentation).
- Nearby regions grow until they collide with other regions.



image



markers (on image)



segmentation

```
I = imread('cells2.png');
I = double(rgb2gray(I));
imshow(I/255);
K = 0.5*[-1 0 1];
Ix = imfilter(I, K);
K = 0.5*[-1 0 1]';
Iy = imfilter(I, K);
E = sqrt(Ix.^2 + Iy.^2);
```

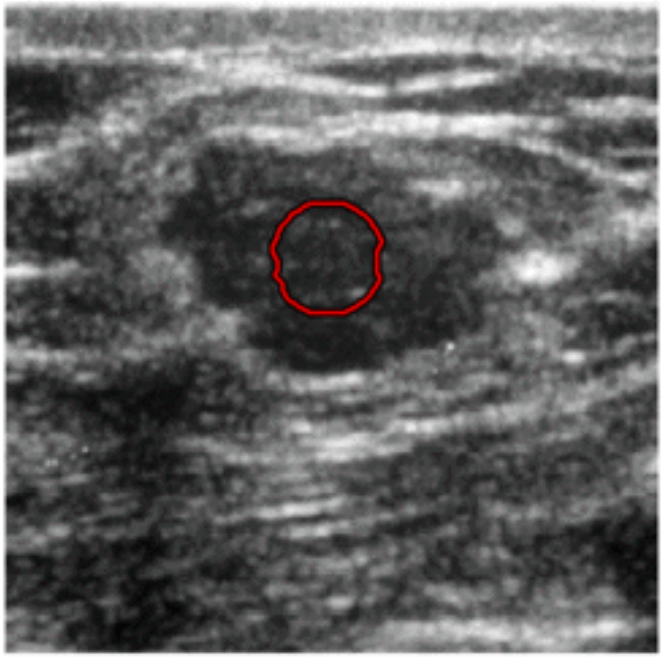
```
B = imregionalmin(I);
```

Regional minima are connected components of pixels with a constant intensity value, and whose external boundary pixels all have a higher value.

```
Enew = imimposemin(E, B);
L = watershed(Enew);
figure;
imshow(I, []);
hold on;
visboundaries(L);
```

Active contour

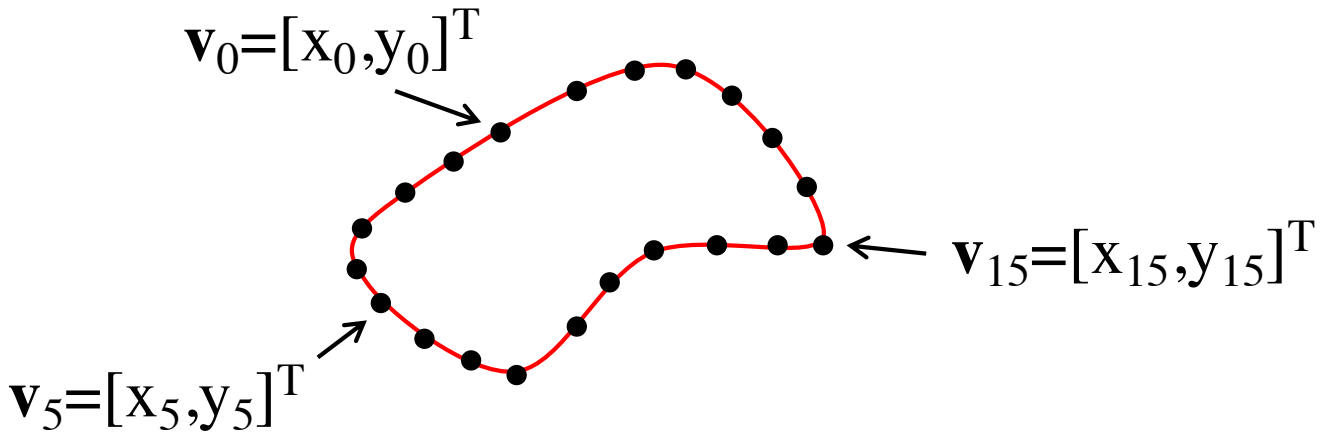
- A contour defines a shape; the region inside the contour can be considered the segmentation.
- An *active contour* is a contour that deforms through time to achieve a segmentation.



https://github.com/pmneila/morphsnakes

Active contour

- Active contour methods are based on energy minimisation using partial differential equations: the contour continually updates until it reaches some minimum of an energy. How that works depends on the contour representation (explicit or implicit) and the energy itself.
- **Explicit** contour representation (aka **snake**): models the contour as a collection of connected points, for example, as a contour **C** which is a collection of connected points, $\mathbf{v}_i$.
- The last point is connected to the first (0^{th}) point for a *closed* contour.



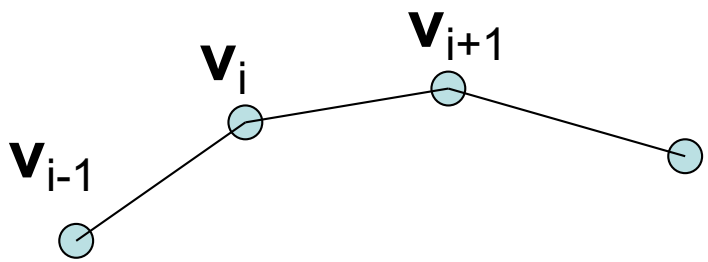
Energy (internal)

- We define an energy based on *internal* and *external* terms.
  - The internal terms seek to keep the curve regular and smooth.
  - The external terms seek to match the contour's shape to the image data.
  - $E = E_{int} + E_{ext}$
- A typical internal term penalises stretching and bending, so the curve wants to have equally separated points and be smooth.

$$E_{int}(\mathbf{v}(s)) = \alpha \left| \frac{\partial \mathbf{v}}{\partial s} \right|^2 + \beta \left| \frac{\partial^2 \mathbf{v}}{\partial^2 s} \right|^2$$

stretching bending

s interpolates the points to form the contour



$$\frac{\partial \mathbf{v}}{\partial s} \approx \mathbf{v}_{i+1} - \mathbf{v}_i$$

$$\frac{\partial^2 \mathbf{v}}{\partial^2 s} \approx -\mathbf{v}_{i-1} + 2\mathbf{v}_i - \mathbf{v}_{i+1}$$

Energy (external)

- A common external energy is an edge map. This will attract the contour to nearby edges.

$$E_{ext}(\mathbf{v}(s)) = -||\nabla I(\mathbf{v})||^2 = -I_x^2 - I_y^2$$

*This is just the negative of the image gradient squared evaluated at **v**.*

- Goal: Iteratively update the active contour points **v** until energy $E = E_{int} + E_{ext}$ is minimised (when the contour converges).
- Since the contour reaches a minimum of the energy, we can say we have an *optimal* segmentation.

$$\begin{aligned} E &= E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s)) \\ &= \alpha \left| \frac{\partial \mathbf{v}}{\partial s} \right|^2 + \beta \left| \frac{\partial^2 \mathbf{v}}{\partial^2 s} \right|^2 - I_x^2 - I_y^2 \end{aligned}$$

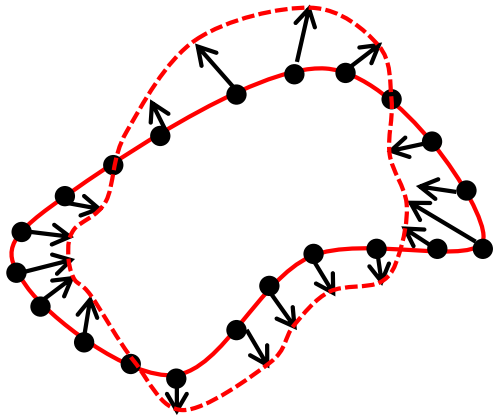
Gradient descent

- A common optimisation approach is gradient descent.
  - Start with an initial contour **C** as a collection of points $\mathbf{v}_i$
  - For each point $\mathbf{v}_i$, compute the gradient of the energy at $\mathbf{v}_i$
  - Update the point in the negative gradient direction

Gradient, or nabla E

$$\nabla E = \left[\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}\right]^T$$

$$\mathbf{v}_i' = \mathbf{v}_i - \Delta t \nabla E$$



Derivation

$$
\begin{aligned}
E(\mathbf{v}) &= E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s)) \\
&= \alpha \left| \frac{\partial \mathbf{v}}{\partial s} \right|^2 + \beta \left| \frac{\partial^2 \mathbf{v}}{\partial^2 s} \right|^2 - I_x^2 - I_y^2 \\
&= \alpha \left| \mathbf{v}_{i+1} - \mathbf{v}_i \right|^2 + \beta \left| -\mathbf{v}_{i-1} + 2\mathbf{v}_i - \mathbf{v}_{i+1} \right|^2 - I_x^2 - I_y^2 \\
&= \alpha (x_{i+1} - x_i)^2 + \alpha (y_{i+1} - y_i)^2 + \beta (-x_{i-1} + 2x_i - x_{i+1})^2 \\
&\quad + \beta (-y_{i-1} + 2y_i - y_{i+1})^2 - I_x^2 - I_y^2
\end{aligned}
$$

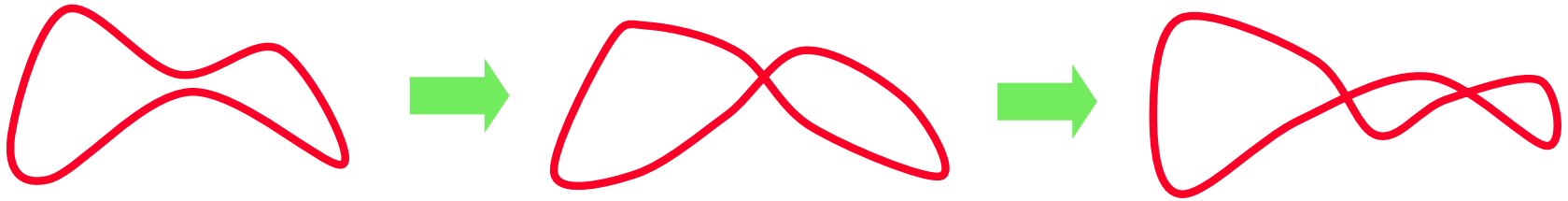
$$
\begin{aligned}
\frac{\partial E(\mathbf{v}(s))}{\partial x} &= -2\alpha (x_{i+1} - x_i) + 2\alpha (x_i - x_{i-1}) \\
&\quad 4\beta (-x_{i-1} + 2x_i - x_{i+1}) - 2\beta (-x_i + 2x_{i+1} - x_{i+2}) - 2\beta (-x_{i-2} + 2x_{i-1} - x_i) \\
&\quad -2I_x I_{xx} - 2I_y I_{xy} \\
&= -2\alpha (x_{i+1} - x_i) + 2\alpha (x_i - x_{i-1}) \\
&\quad +2\beta (x_{i-2} - 4x_{i-1} + 6x_i - 4x_{i+1} + x_{i+2}) - 2I_x I_{xx} - 2I_y I_{xy}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E(\mathbf{v}(s))}{\partial y} &= -2\alpha (y_{i+1} - y_i) + 2\alpha (y_i - y_{i-1}) \\
&\quad 4\beta (-y_{i-1} + 2y_i - y_{i+1}) - 2\beta (-y_i + 2y_{i+1} - y_{i+2}) - 2\beta (-y_{i-2} + 2y_{i-1} - y_i) \\
&\quad -2I_x I_{xy} - 2I_y I_{yy} \\
&= -2\alpha (y_{i+1} - y_i) + 2\alpha (y_i - y_{i-1}) \\
&\quad +2\beta (y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2}) - 2I_x I_{xy} - 2I_y I_{yy}
\end{aligned}
$$

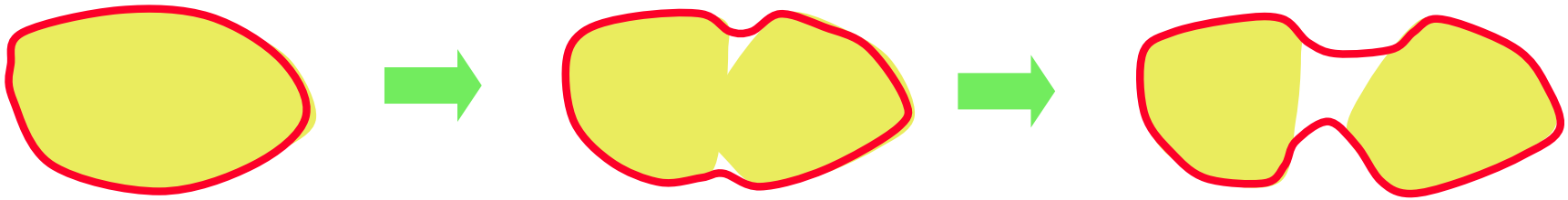
Questions about this derivation will not appear on the exam.

Issues

- Snake may not be aware of edges if too far away. Gradient descent only produces a local optimum of the energy. So, the result is dependent on the initialisation: one should start close to the correct segmentation.
- Depending on parameters, the snake may over-smooth the boundary, or self-intersect.

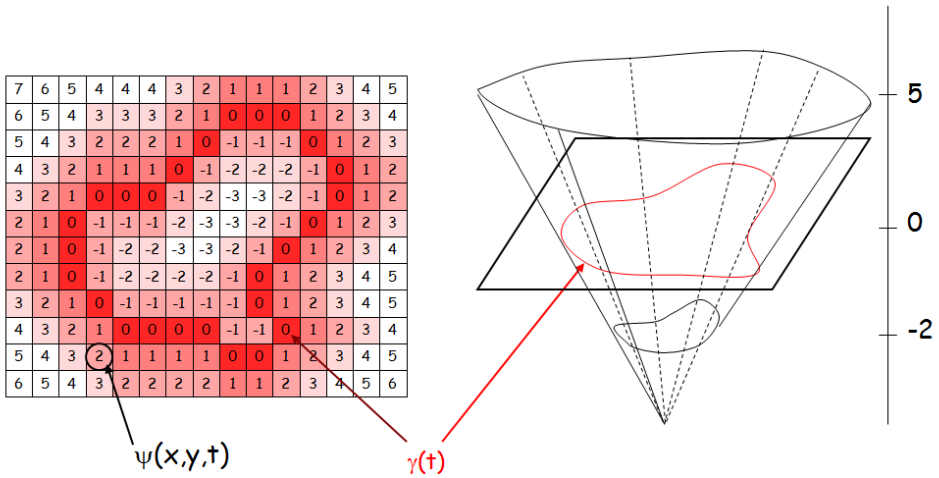


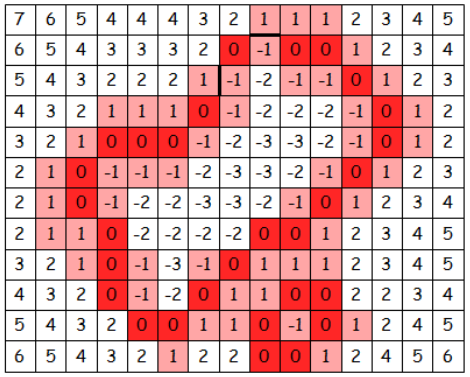
- Topological changes are difficult to model.



Level set methods

- In a level set method, the active contour is represented **implicitly**, as the zero level set of a distance function ψ, which is negative inside the contour, and positive outside.
- Anywhere that is zero is “on the curve”. Curve can have sub-pixel accuracy.
- This representation naturally allows for topological changes (curve can split or merge as needed).





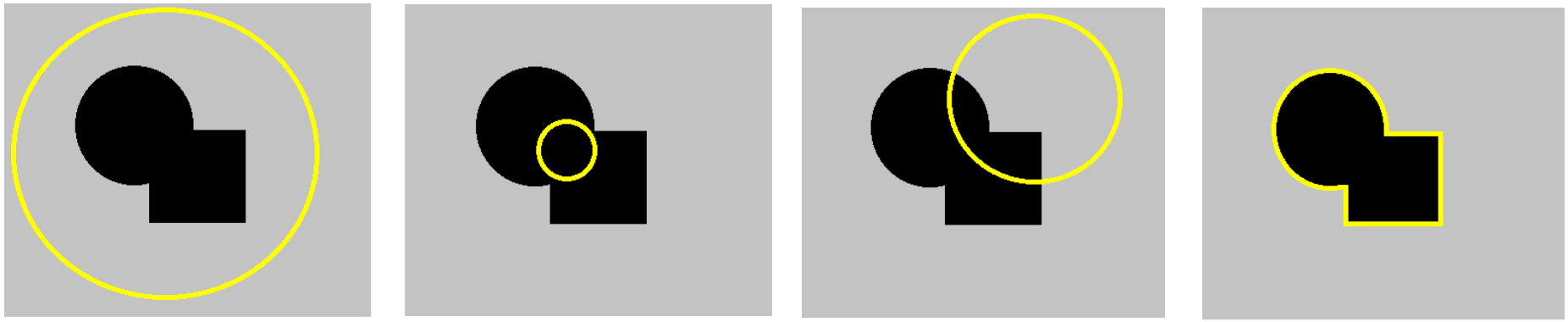
Chan-Vese

- A popular level set method is based on the Chan-Vese model, which deforms the contour to minimise the variance both *inside* and *outside* the contour.

$$E(C) = \int_{R_i} |I - \mu_i|^2 dxdy + \int_{R_o} |I - \mu_o|^2 dxdy$$

here μ_i and μ_o are the mean intensities inside and outside the contour, and R_i and R_o are the regions inside and outside the contour, respectively.

- Note: this method does not depend on edges.



E>0 E>0 E>0 E~0

Chan-Vese

- In Matlab, this can be implemented using the `activeContour` function, as

```
B = activecontour(I, mask, N, 'Chan-Vese');
```

where *I* is the image, mask is a 2D binary image representing the initial contour,

N is the number of iterations of energy minimisation

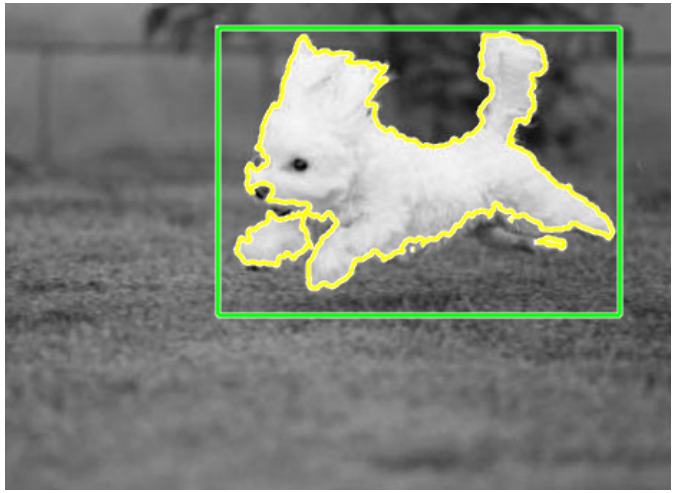
B is the resulting segmentation in the form of a binary image.

- Note: if you have a binary image B, you can visualise it as a contour by calling

```
visboundaries(B,'Color','y');
```

- Example:

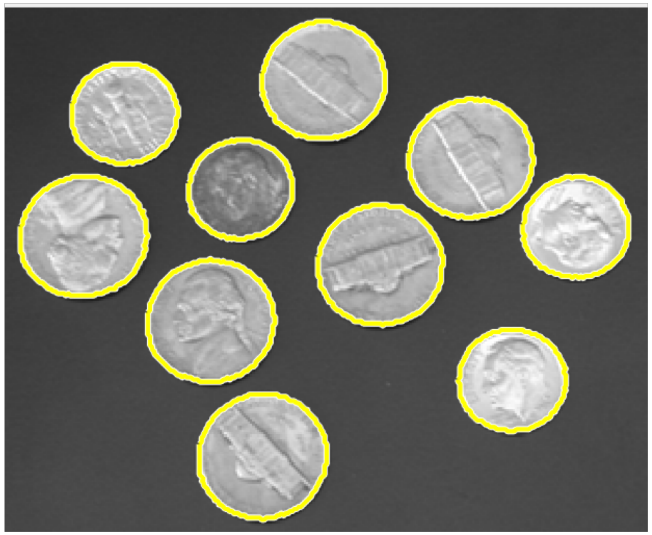
```
I = rgb2gray(imread('whitedog1.jpg'));
imshow(I);
hold on;
mask = false(size(I));
mask(20:220, 150:430) = true;
visboundaries(mask,'Color','g');
B = activecontour(I, mask, 250, 'Chan-Vese');
visboundaries(B,'Color','y');
```



https://stuffmydoghates.wordpress.com/2008/10/30/cute-little-white-dogs/

Example

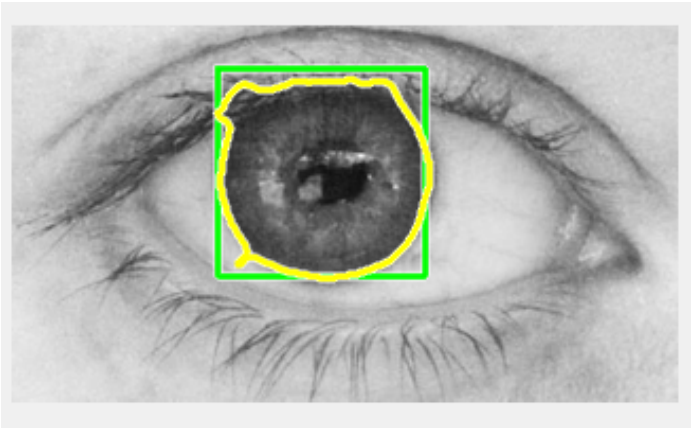
```
close all;
I = imread('coins.png');
B = true(size(I));
for i = 1:7
    figure;
    imshow(I);
    hold on;
    B = activecontour(I, B, 50, 'Chan-Vese');
    visboundaries(B,'Color','y');
    pause;
end
```



Geodesic active contour

- The activecontour function in Matlab also supports an edge-based method that implements the geodesic active contour.
- This method by default tries to shrink the active contour, until it “locks” onto edges that are detected.
- You can call it by passing the argument `'edge'` as the fourth parameter.

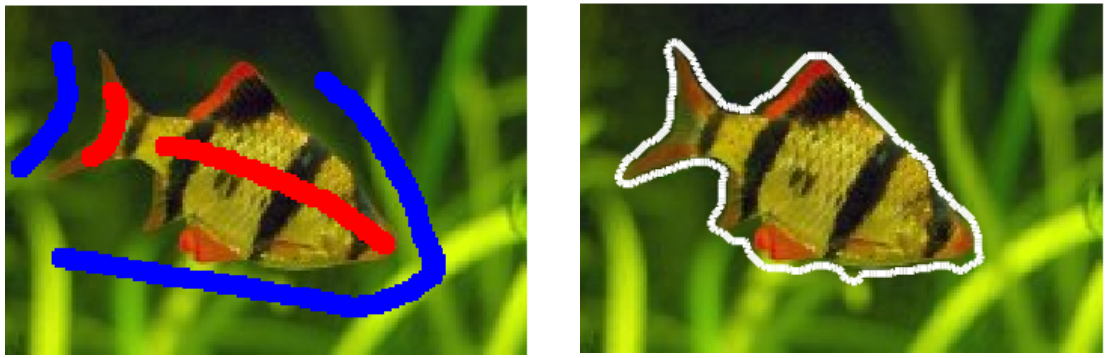
```
I = imresize(rgb2gray(imread('eye.jpg')), 0.25);
imshow(I);
mask = false(size(I));
mask(21:119, 98:196) = true;

B = activecontour(I, mask, 250, 'edge');
hold on;
visboundaries(mask,'Color','g');
visboundaries(B,'Color','y');
```



- Active contours can be applied in a higher number of dimensions, like 3D images (CT, MR). *Note however, the implementation built-in to Matlab only supports 2D images.*

Interactive segmentation

- Recently a number of interactive segmentation methods have been presented in the literature. These methods often ask the user to provide some strokes providing seeds.



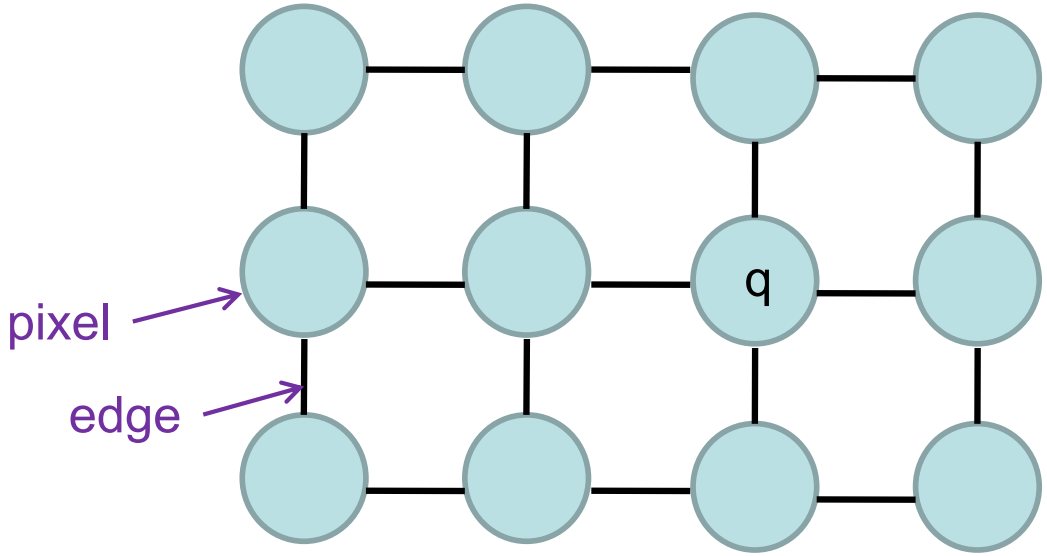
http://jgmalcolm.com/pubs/research.html

- The strokes:
  - Are used to form a colour model of different regions.
  - Also provide spatial context.

- https://www.youtube.com/watch?v=KXQuR-lnVFQ

Markov Random Field

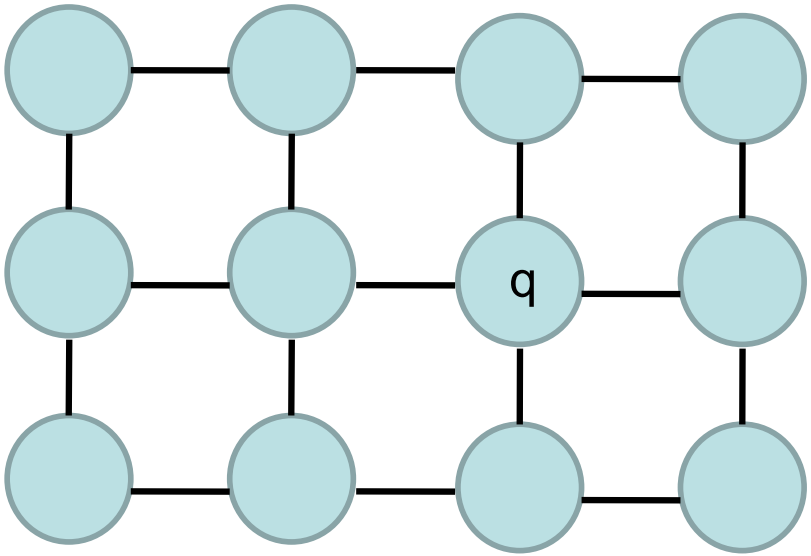
- Often, one employs a Markov Random Field to solve this problem.
- In a Markov Random Field (MRF), each pixel in the image is considered a random variable. It is connected to its adjacent pixels through edges.
- The segmentation label for a pixel depends on two terms:
  - A *unary* term, the cost of assigning a label to a pixel, independent of the neighbours.
  - A *pairwise* term, the cost of assigning a label to a pixel based on a neighbour in the neighbourhood.



The cost of assigning a label ℓ to pixel q depends on the image and the values of labels of the connected pixels as well.

Markov Random Field

- Example

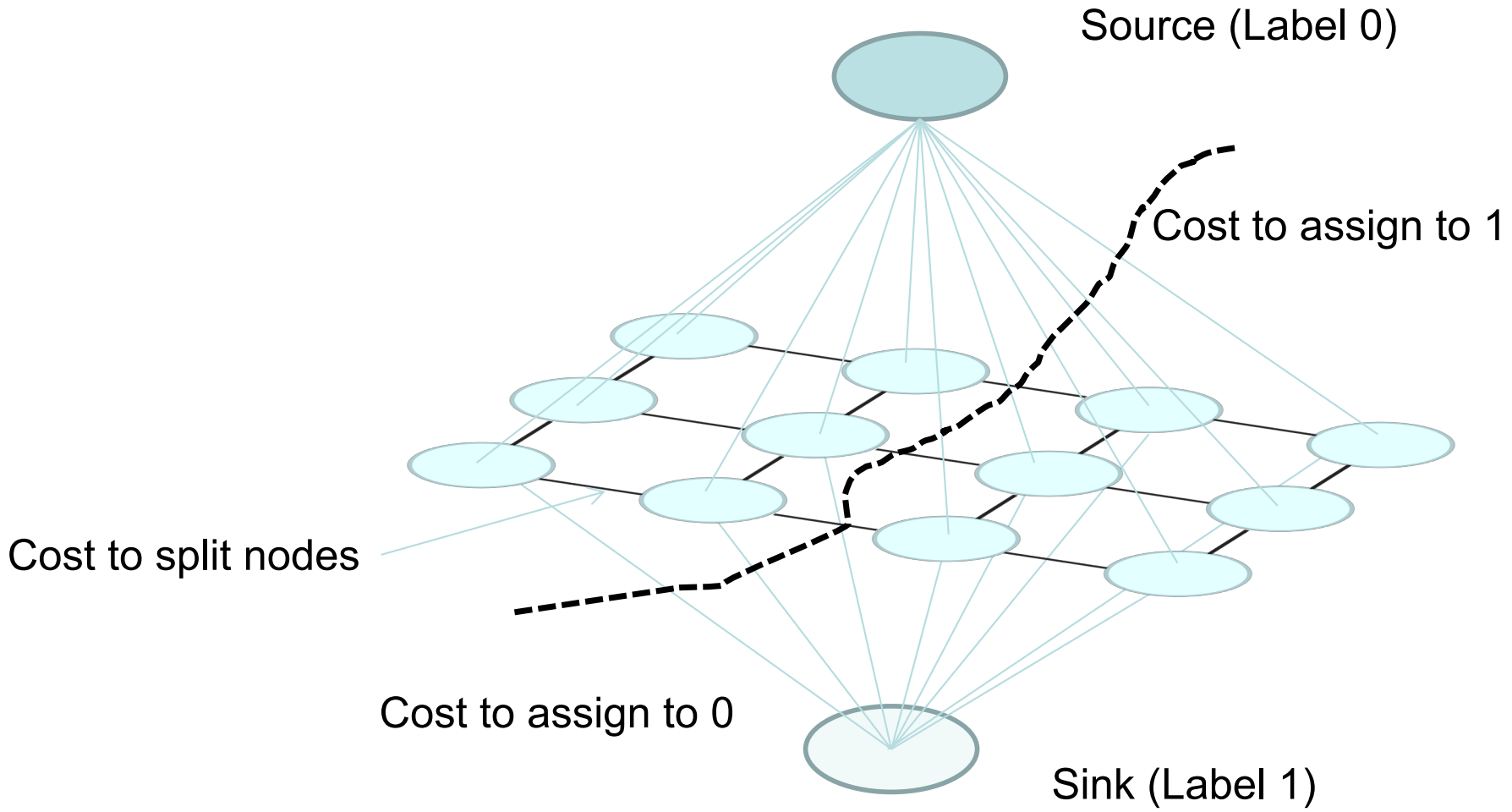


- Unary
  - Label 0: -log p(ℓ(q) = 0| image)
  - Label 1: -log p(ℓ(q) = 1| image)
- Pairwise

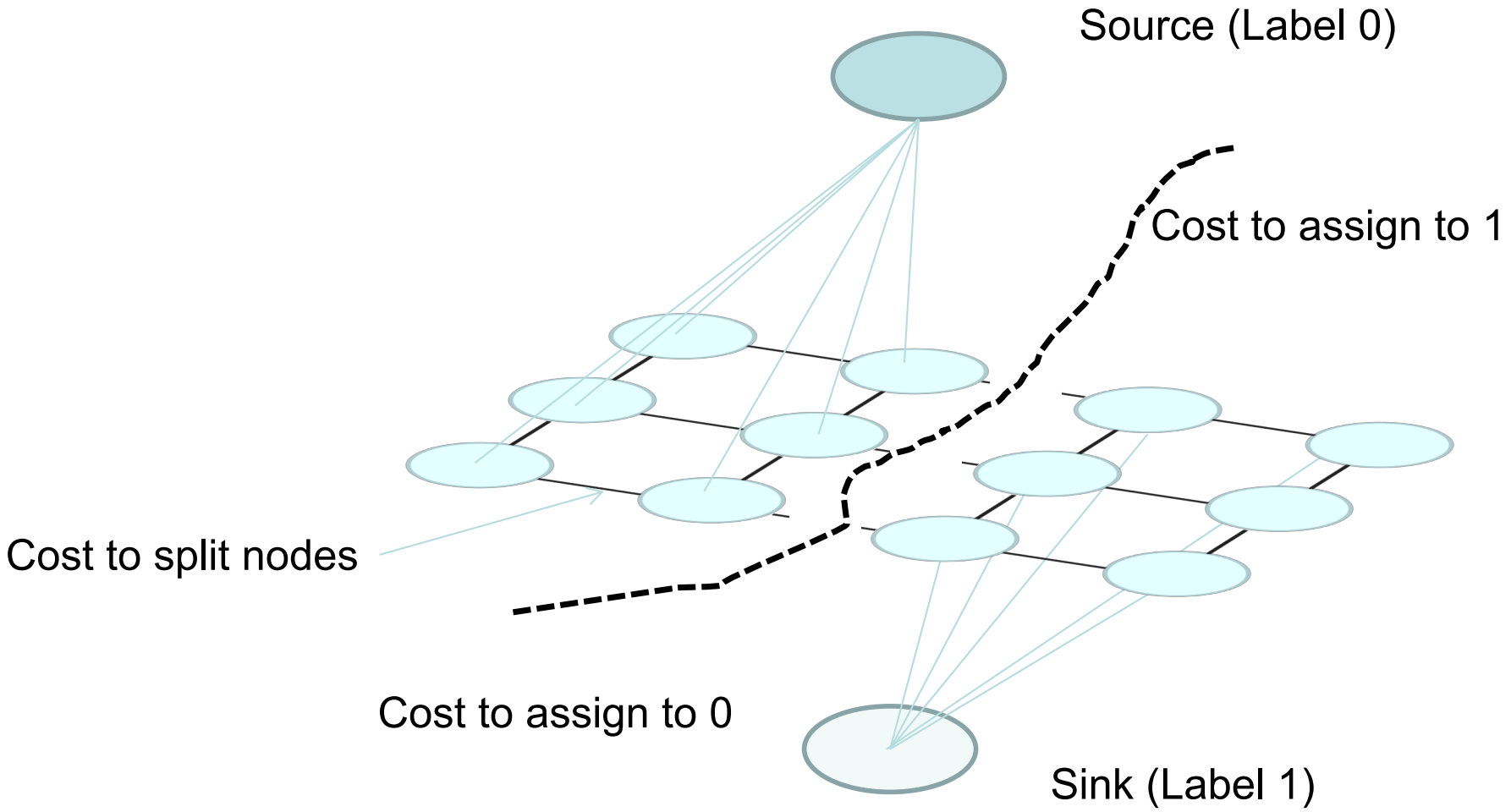
	0	1
0	0	K
1	K	0

K>0

Graph cuts


Source (Label 0)
Cost to assign to 1
Cost to split nodes
Cost to assign to 0
Sink (Label 1)

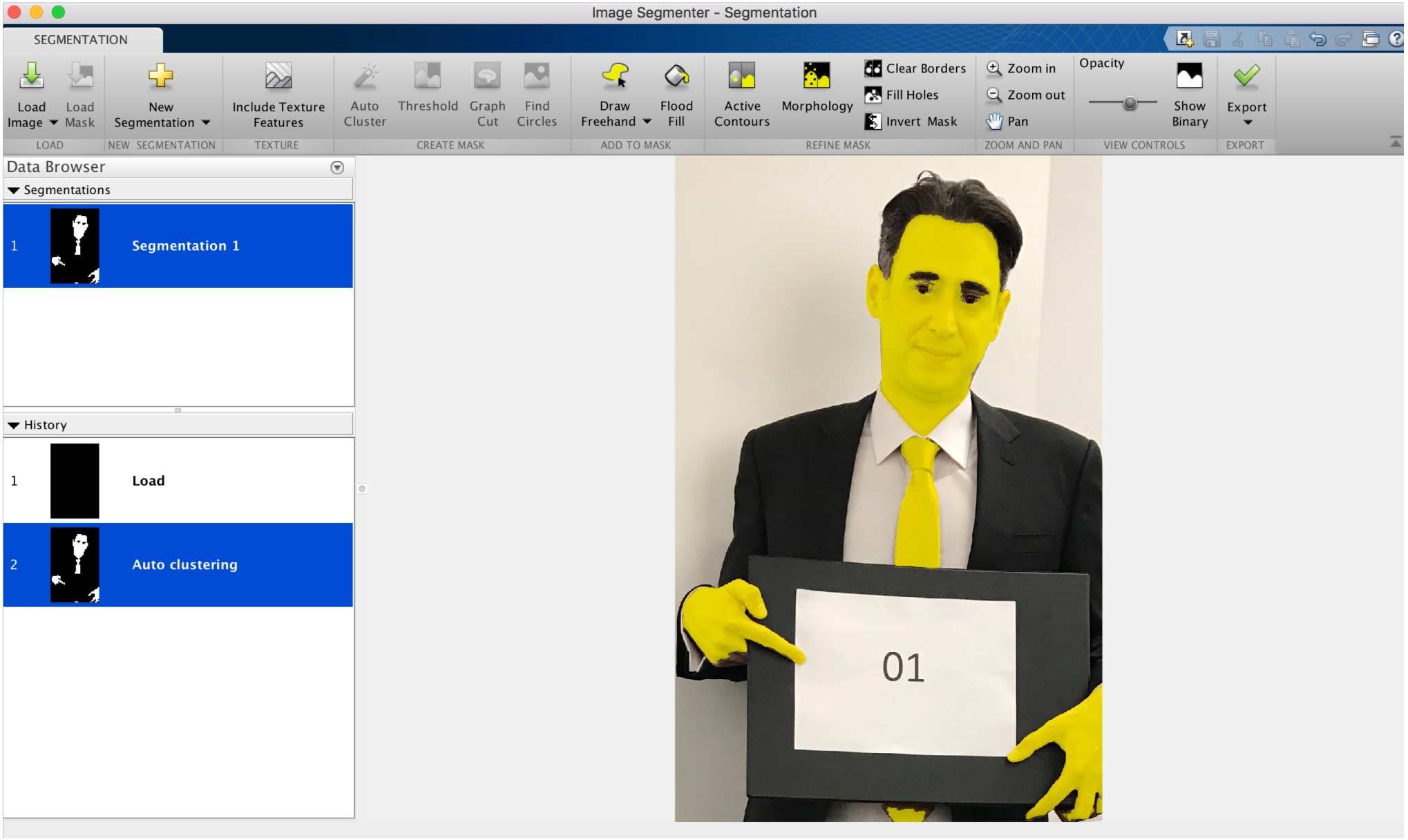
Graph cuts



https://www.youtube.com/watch?v=HMGX8HXskKk

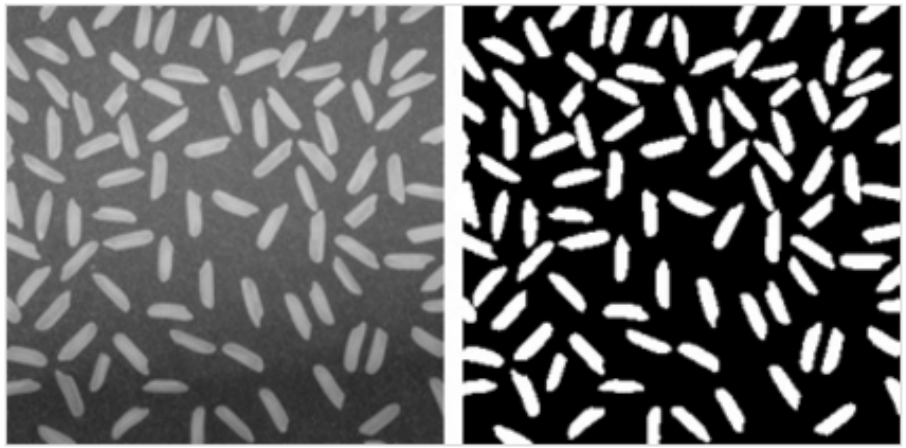
Graph cuts

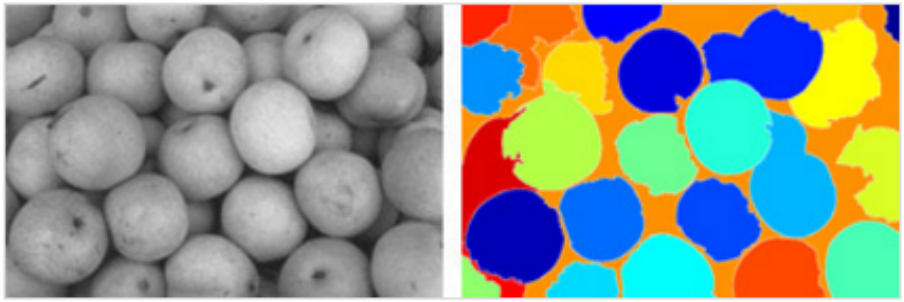
- https://uk.mathworks.com/help/images/segment-image-using-graph-cut.html

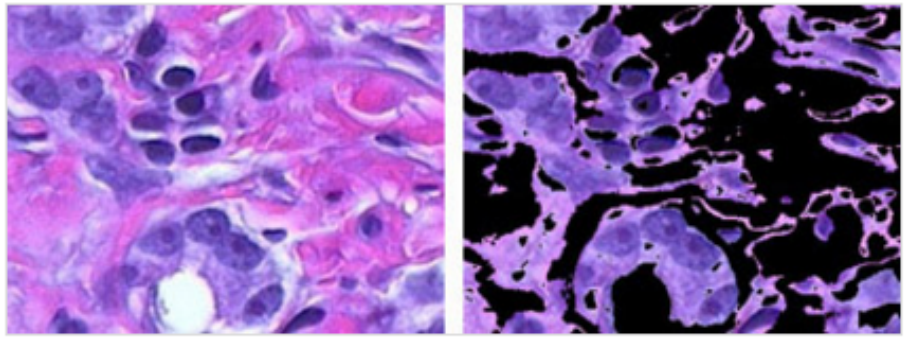


Matlab documentation and examples

- https://uk.mathworks.com/discovery/image-segmentation.html







Superpixels

- A popular approach currently in image segmentation is to first group pixels together into superpixels to form an oversegmentation of the image.
- A superpixel consists of a connected group of pixels that have similar properties, like colour.
- Then, one can group superpixels together to form the final segmentation. This can be more efficient than processing each pixel in the image.

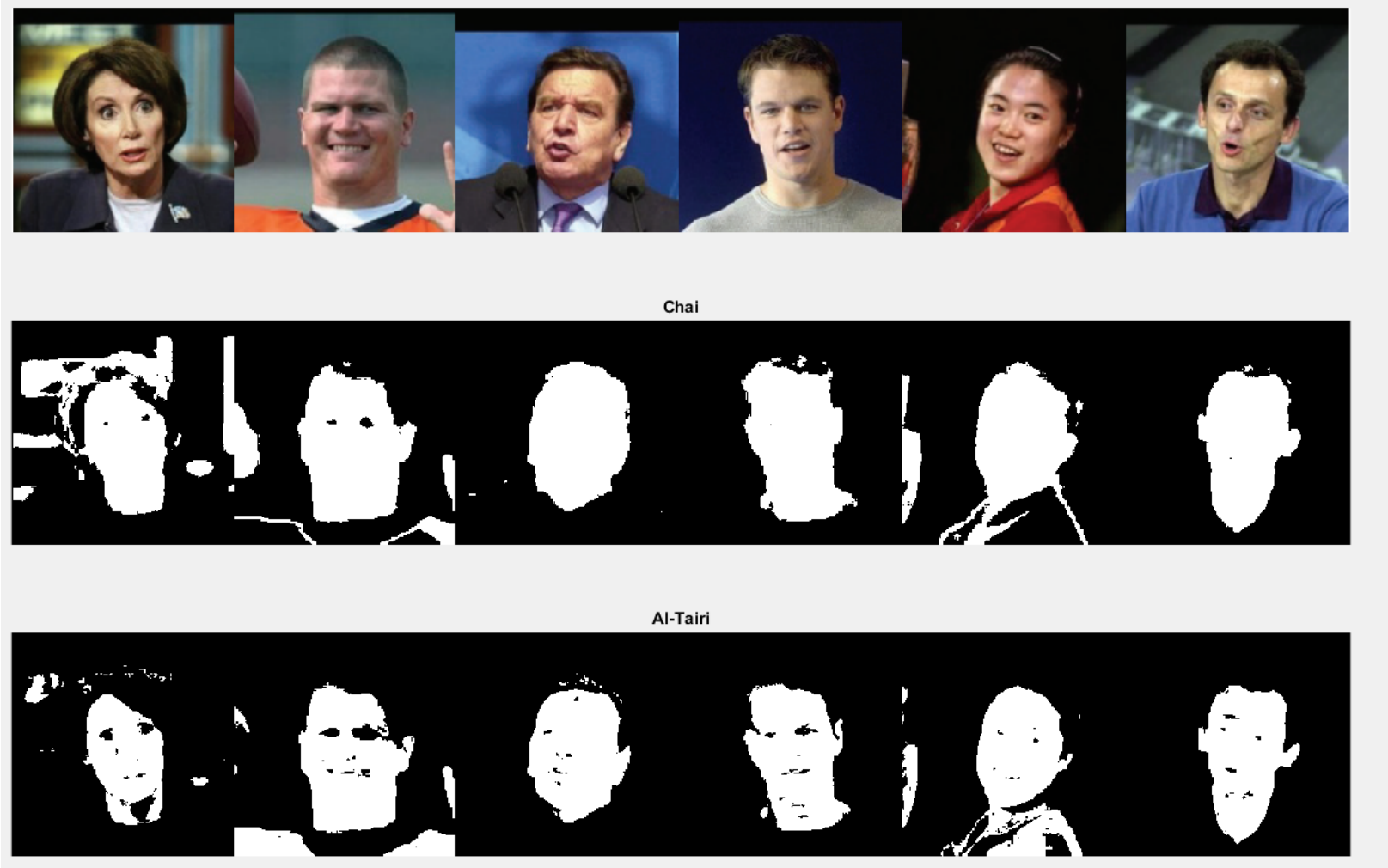
SLIC

- SLIC (Simple Linear Iterative Clustering) is a popular superpixel segmentation technique, published by [Achanta et al. 2012]
- It performs a local grouping of pixels in 5D space defined by the LAB colours and x, y positions of the pixels [LABXY]. A distance measure in this 5D space combines distance measured in colour and the image itself
- It defines K regularly spaced cluster centres, and moves them to locations where the edge response is smallest, and assigns pixels to clusters based on distance.

- https://uk.mathworks.com/help/images/ref/superpixels.html

Evaluating segmentations

- What makes for a “good” segmentation?



Evaluating segmentations

- A common approach is to establish some ground truth. This can specify, for each pixel in the image if it should be part of the segmentation (or not).
- Example data for skin detection: http://web.fsktm.um.edu.my/~cschan/downloads_skin_dataset.html

Evaluating segmentations

- One can perform the segmentation, and using the ground truth, categorise pixels as:
  - True positives (TP): correctly detected skin pixels
  - True negatives (TN): correctly detected non-skin (background) pixels
  - False positives (FP): non-skin pixels incorrectly detected as skin pixels
  - False negatives (FN): skin pixels missed by the skin detector

Accuracy, sensitivity, specificity

- One can then produce quantitative measures
  - Accuracy

    $$ACC = (TP + TN) / (TP + FP + FN + TN)$$

    This measures the number of correctly labelled pixels in the image

  - Sensitivity, or True Positive Rate

    $$SENS = TP / P = TP / (TP + FN)$$

    This measures the percentage of positives correctly labelled

  - Specificity, or True Negative Rate

    $$SPEC = TN / N = (TN + FP)$$

    This measures the percentage of negatives correctly labelled

- Where TP and TN are the number of true positives and true negatives, and FP and FN are the number of false positives and false negatives.
- A perfect result would have 100% accuracy, sensitivity, and specificity.

Example

```
I = imread('face.jpg');
T = rgb2gray(imread('groundTruth.png')) / 255;
YUV = rgb2ycbcr(I);
U = YUV(:, :, 2); V = YUV(:, :, 3);
R = I(:, :, 1); G = I(:, :, 2); B = I(:, :, 3);
[rows, cols, planes] = size(I);

% Al-Tairi et al.
skin = zeros(rows, cols);
ind = find(80 < U & U < 130 & 136 < V & V <= 200 & V > U & R > 80 & G > 30 & ...
      B > 15 & abs(R-G) > 15);
skin(ind) = 1;
figure;
subplot(1, 3, 1); imshow(I); title('Image');
subplot(1, 3, 2); imshow(T*255); title('Ground Truth');
subplot(1, 3, 3); imshow(skin); title('Al-Tairi');

tpInd = find (skin == 1 & T == 1);
tnInd = find (skin == 0 & T == 0);
fpInd = find (skin == 1 & T == 0);
fnInd = find (skin == 0 & T == 1);
```

Example, part 2

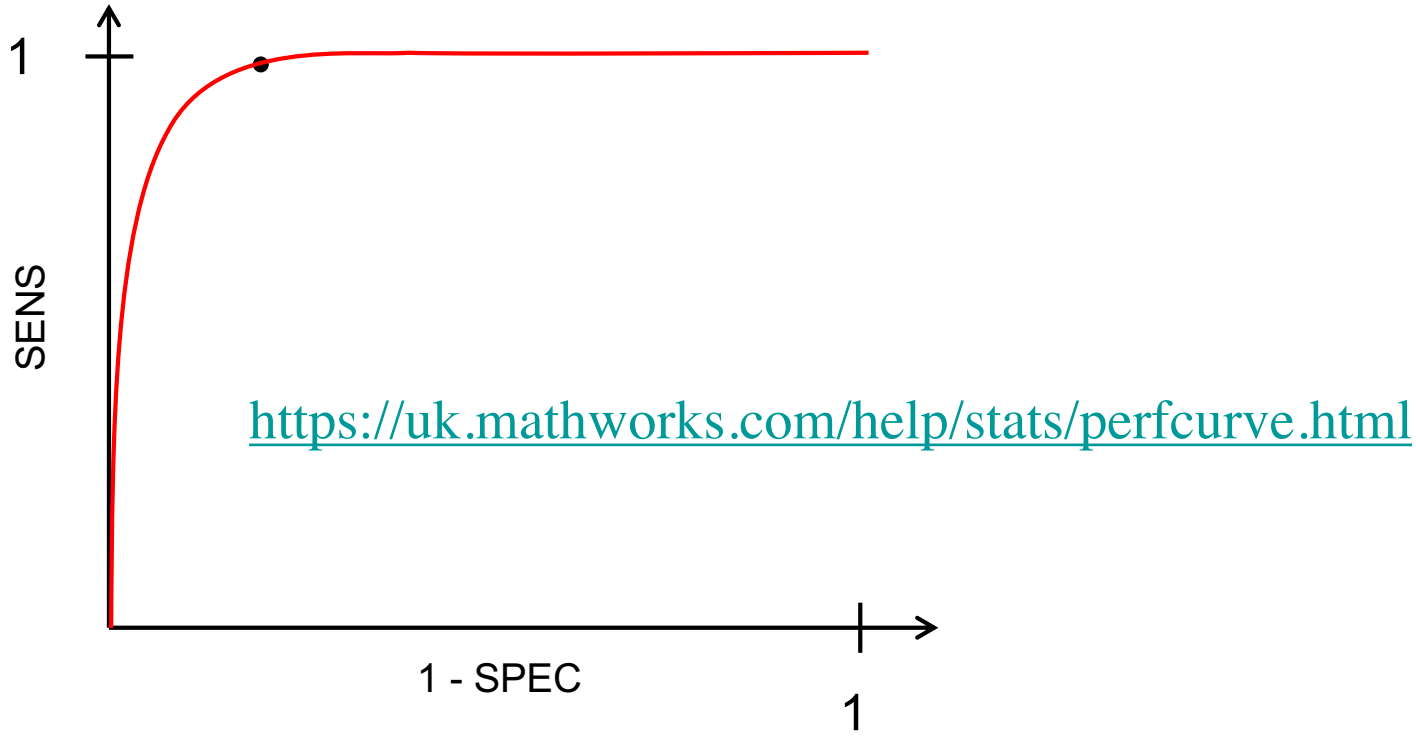
```
tpImage = zeros(rows, cols); tpImage(tpInd) = 1;
tnImage = zeros(rows, cols); tnImage(tnInd) = 1;
fpImage = zeros(rows, cols); fpImage(fpInd) = 1;
fnImage = zeros(rows, cols); fnImage(fnInd) = 1;

figure;
subplot(1, 4, 1); imshow(tpImage); title('true positives');
subplot(1, 4, 2); imshow(tnImage); title('true negatives');
subplot(1, 4, 3); imshow(fpImage); title('false positives');
subplot(1, 4, 4); imshow(fnImage); title('false negatives');

tp = length(tpInd); tn = length(tnInd);
fp = length(fpInd); fn = length(fnInd);

% Compute measures
accuracy = (tp + tn) / (tp + tn + fp + fn);
sens = tp / (tp + fn);
spec = tn / (tn + fp);
disp(['Accuracy = ' num2str(accuracy) ', sensitivity = ' num2str(sens) ', ...
specificity = ' num2str(spec)]);
```

>> Accuracy = 0.8979 , sensitivity = 0.98949, specificity = 0.82425

Interpretation: Sensitivity is high so many skin pixels correctly identified. However, specificity is lower, due primarily to the false positives. Overall, roughly 90% of the pixels are correctly labelled.

ROC curve

- A Receiver Operating Characteristic curve plots the sensitivity as a function of 1 – specificity.
- In our previous example, SENS = 0.99, 1 – SPEC = 0.28
- One can sweep a parameter that trades off sensitivity vs specificity to generate a curve. The area under the curve gives another way to characterise the quality of a classification method.



https://uk.mathworks.com/help/stats/perfcurve.html

Confusion matrix

- The confusion matrix describes the number of predicted labels vs the ground truth. In this example, we have skin and non-skin pixels.

		Predicted	
		Skin	Non-skin
Actual	Skin	TP	FN
	Non-skin	FP	TN

- A perfect result would be a diagonal matrix (no FPs or FNs)

- The confusion matrix generalises to multiple labels

		Predicted		
		Label 1	Label 2	Label 3
Actual	Label 1			
	Label 2			
	Label 3			

Overview of next lecture (5)

- Introduction to image classification
- Feature extraction:
  - HOG (Histogram of Oriented Gradients)
  - Blob
  - Gabor
  - Bag of Features
- Introduction to machine learning
- Supervised Learning:
  - Probabilistic Models
  - Neural Networks
  - Support Vector Machines
  - Cascading classifiers
- **NEXT SESSION (Lecture 6):**
  - Unsupervised Learning:
    - K-Means
      - Mean-Shift
    - Self Organising Maps
  - Other feature extraction methods:
    - SIFT (Scale Invariant Feature Transform)
    - SURF (Speeded Up Robust Features)