


浙江大学爱丁堡大学联合学院

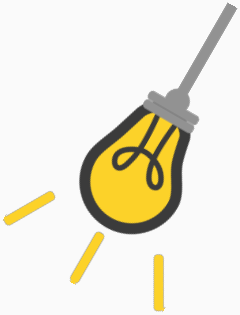
ZJU-UoE Institute

Lecture 6 - Image features

Nicola Romanò - nicola.romano@ed.ac.uk

Learning objectives

- Define and give examples of image features
- Describe and apply methods to extract geometric features in an image
- Describe and apply methods for texture analysis in images
- Discuss advantages and issues with these methods



What is a feature?

Features

Feature, n.: a distinctive or characteristic part of a thing; some part which arrests the attention by its conspicuousness or prominence.
(*Source: Oxford English Dictionary*)

Features

Feature, n.: a distinctive or characteristic part of a thing; some part which arrests the attention by its conspicuousness or prominence.
(*Source: Oxford English Dictionary*)

In image analysis, a **feature** is a characteristic of the image, or of a part of it, that defines some of its properties.

Examples of features include edges, corners, ridges, texture, colour, shape etc.

Features

Feature, n.: a distinctive or characteristic part of a thing; some part which arrests the attention by its conspicuousness or prominence.
(*Source: Oxford English Dictionary*)

In image analysis, a **feature** is a characteristic of the image, or of a part of it, that defines some of its properties.

Examples of features include edges, corners, ridges, texture, colour, shape etc.

Detecting features is a fundamental step to extract information from images.

Features in images

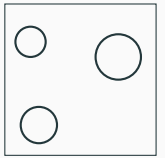
View in the Gran Sasso mountain range, Italy - Photo: Nicola Romanò

Which are the prominent features of this image?

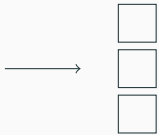
Why detect features?

Examples of tasks for which we may use features

- To detect whether specific objects are present in an image and where



Image



Features

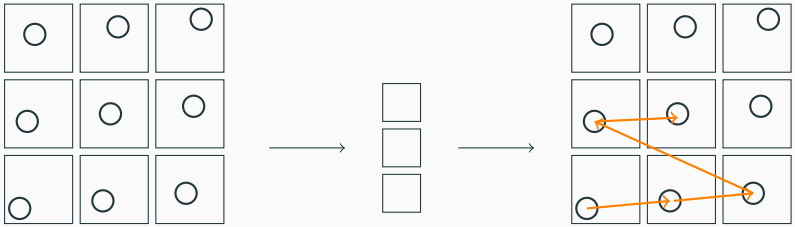
X	Y	r
3	10	20
15	13	30
45	40	24

Output

Why detect features?

Examples of tasks for which we may use features

- To detect whether specific objects are present in an image and where
- To track the position of objects in an video (by using the position of specific features)



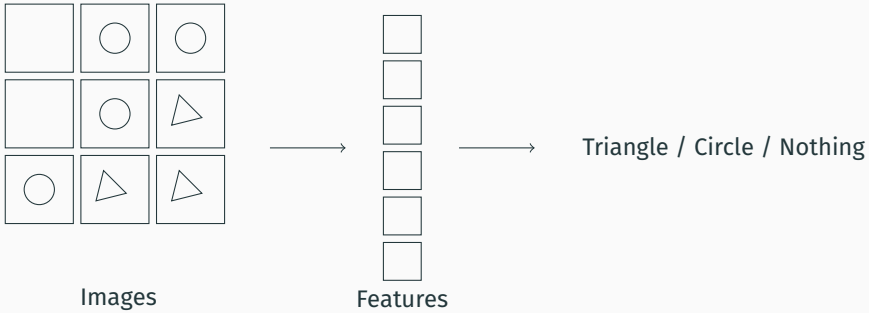
Image

Features

Why detect features?

Examples of tasks for which we may use features

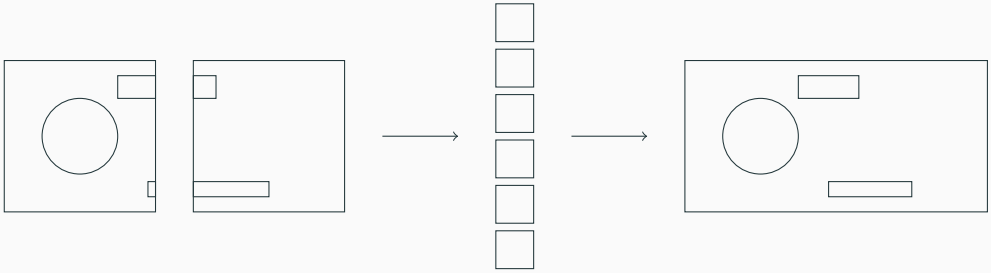
- To detect whether specific objects are present in an image and where
- To track the position of objects in an video (by using the position of specific features)
- To classify images (by using features as descriptors)



Why detect features?

Examples of tasks for which we may use features

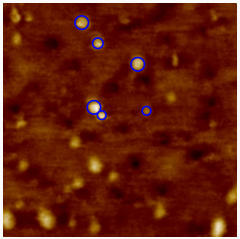
- To detect whether specific objects are present in an image and where
- To track the position of objects in an video (by using the position of specific features)
- To classify images (by using features as descriptors)
- To stitch images together



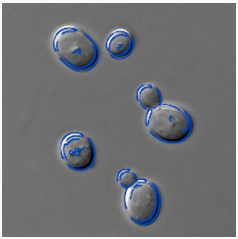
Geometric features

Geometric features

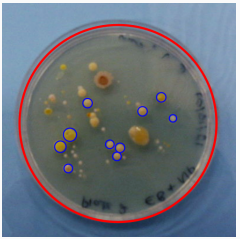
Examples of **geometric features** include:



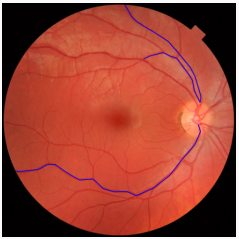
(a) Blobs - Marsh et al., Sci Rep 2018



(b) Corner and edges (see lecture 5) - Credits: Masur, Wikipedia, CC-0



(c) Lines, circles, etc. - Credits: Vivien Rolfe, Flickr, CC-BY-SA-2.0



(d) Ridges - Credits: Medical gallery of Mikael Häggström., CC-0

Geometric features

Blobs

Blob detection

Blob detection is the task of detecting blobs in an image.

Blobs are defined as regions of pixels with uniform properties that are clearly distinguishable from the background.

There are many methods for detecting blobs in an image; today we will look at using the Laplacian of the Gaussian (**LoG**) method, and its approximation by the difference of Gaussians (**DoG**) method.

The Laplacian operator

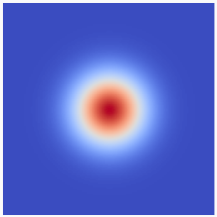
In the last lecture we talked about using second order derivatives to detect edges in an image. The Laplacian of Gaussian operator is defined as the sum of the second order partial derivatives of the image.

$$\text{Laplacian(I)} = \nabla^2(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

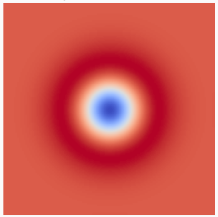
The Laplacian of the Gaussian

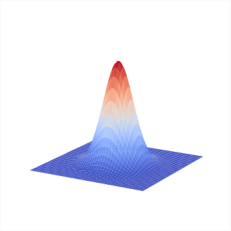
What does the Laplacian of a Gaussian function look like?

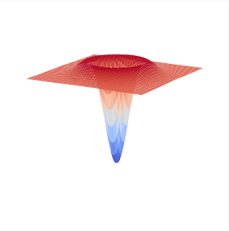
Gaussian (G)



Laplacian ($\nabla^2(G)$)







So... how do we use this to detect blobs?

The LoG method for blob detection

The idea is to:

1. Apply a LoG filter $LoG(\sigma)$ to the image, as discussed in Lecture 5
2. Dark blobs on light background will appear as bright spots in the Laplacian of the Gaussian-filtered image; bright blobs on dark background will appear as dark spots.

The LoG method for blob detection

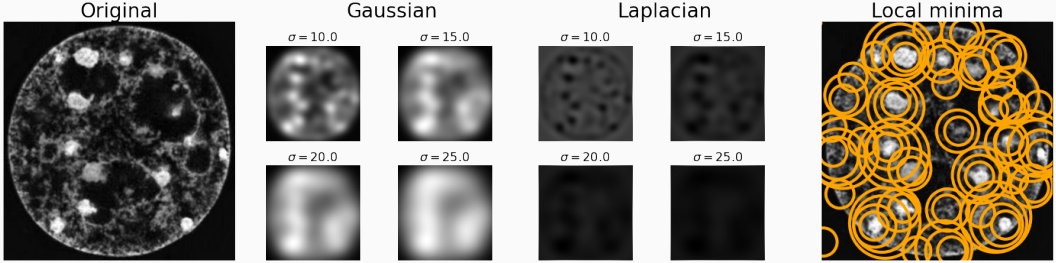
The idea is to:

1. Apply a LoG filter $LoG(\sigma)$ to the image, as discussed in Lecture 5
2. Dark blobs on light background will appear as bright spots in the Laplacian of the Gaussian-filtered image; bright blobs on dark background will appear as dark spots.
3. We find local maxima (or minima) in the Laplacian of the Gaussian-filtered image to determine the position of the blobs
4. The blob size is approximately $\sqrt{2}\sigma$ (or $\sqrt{3}\sigma$ for 3D)

The LoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

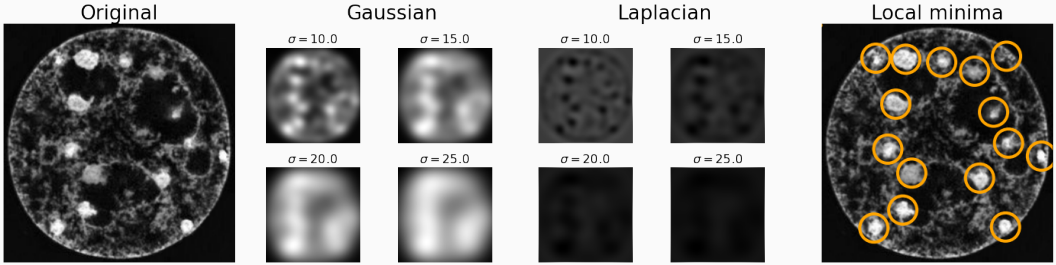
We can apply a threshold to the LoG image to remove spurious blobs and can also remove blobs overlapping over a certain %



The LoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

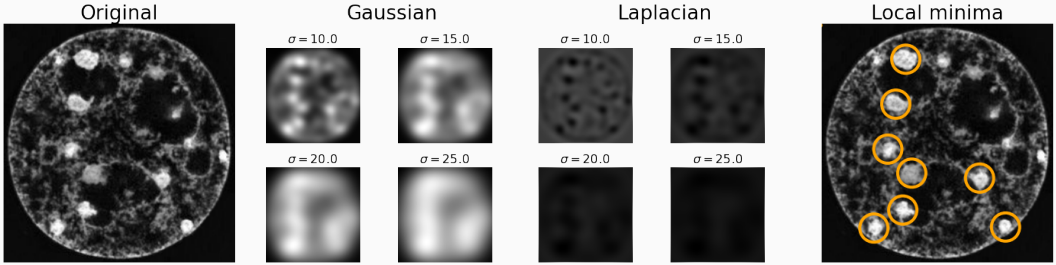
We can apply a threshold to the LoG image to remove spurious blobs and can also remove blobs overlapping over a certain %



The LoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

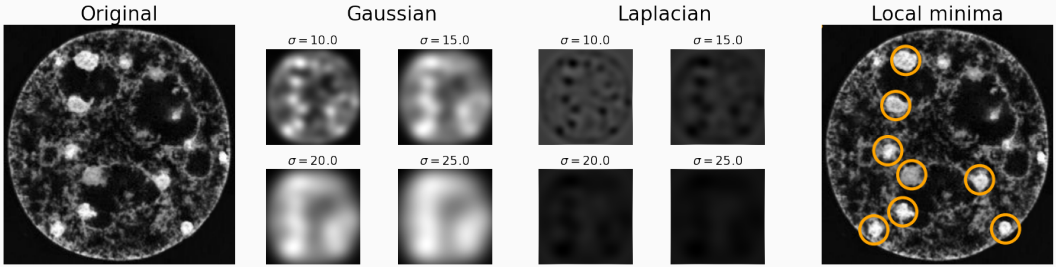
We can apply a threshold to the LoG image to remove spurious blobs and can also remove blobs overlapping over a certain %



The LoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

We can apply a threshold to the LoG image to remove spurious blobs and can also remove blobs overlapping over a certain %



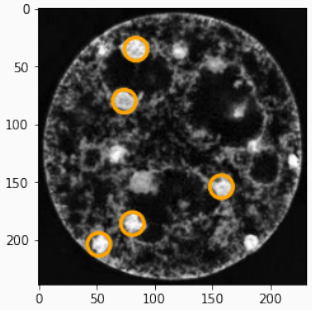
Scikit image implements these steps in the `skimage.feature.blob_log` function.

Example of blob detection

```
from skimage.feature import blob_log
from skimage.io import imread

nucleus = imread("nucleus_blob.png")
blobs = blob_log(nucleus, min_sigma=10, max_sigma=15,
                 num_sigma=5, threshold=0.3, overlap=.4)
```

Example of blob detection

```
from skimage.feature import blob_log
from skimage.io import imread

nucleus = imread("nucleus_blob.png")
blobs = blob_log(nucleus, min_sigma=10, max_sigma=15,
                 num_sigma=5, threshold=0.3, overlap=.4)
plt.imshow(nucleus, cmap="gray")

for b in blobs:
   y, x, radius = b
   # Create a circle at the blob center
   c = plt.Circle((x, y), radius, color='orange',
                  linewidth=3, fill=False)
   # Draw the circle (gca = "get current axis")
   plt.gca().add_artist(c)
plt.show()
```



Other methods for blob detection

Scikit image impements two other methods for blob detection: the DoG (Difference of Gaussians) and the DoH (Determinant of the Hessian) methods.

Other methods for blob detection

Scikit image impements two other methods for blob detection: the DoG (Difference of Gaussians) and the DoH (Determinant of the Hessian) methods.

The **DoG method** is a fast approximation of the LoG method. It works by calculating the difference between two version of the image blurred by Gaussians with different σ.

The **DoH method** is the fastest method. It works by calculating the determinant of the Hessian of the image (the Hessian is a matrix of second partial derivatives). Contrary to DoG and LoG, the detection speed is independent of the size of the blobs. However, DoH is less accurate at detecting small blobs.

Other methods for blob detection

Scikit image impements two other methods for blob detection: the DoG (Difference of Gaussians) and the DoH (Determinant of the Hessian) methods.

The **DoG method** is a fast approximation of the LoG method. It works by calculating the difference between two version of the image blurred by Gaussians with different σ.

The **DoH method** is the fastest method. It works by calculating the determinant of the Hessian of the image (the Hessian is a matrix of second partial derivatives). Contrary to DoG and LoG, the detection speed is independent of the size of the blobs. However, DoH is less accurate at detecting small blobs.

These are implemented in `skimage.feature.blob_dog` and `skimage.feature.blob_doh` respectively.

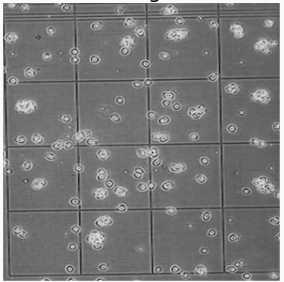
Geometric features

Lines and circles (the Hough transform)

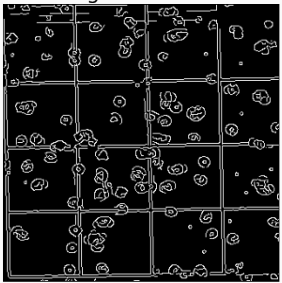
The Hough transform

The Hough transform is a technique originally developed by Paul Hough in 1962 for finding straight lines in an image. Extensions of this technique allow to find circles and quadrilaterals.

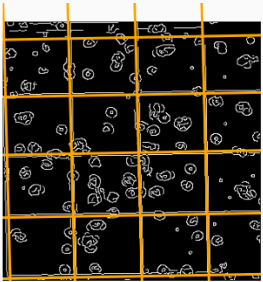
Image



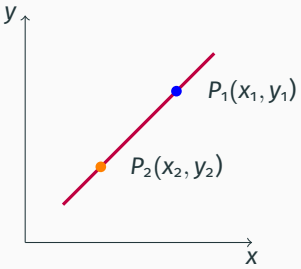
Edge detection



Detected lines

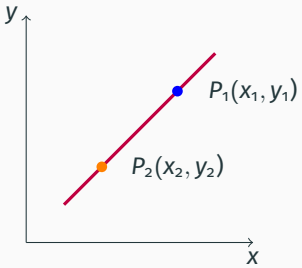


Representing a line

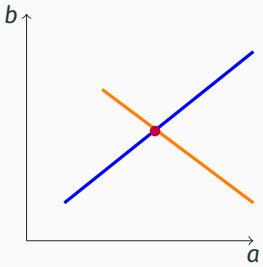


Consider a line with equation: $y = a \cdot x + b$ and two points P_1 and P_2 on it.

Representing a line

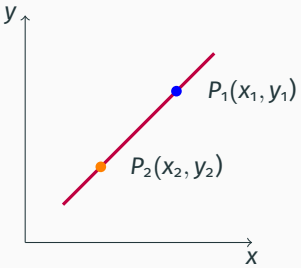


Consider a line with equation: $y = a \cdot x + b$ and two points P_1 and P_2 on it.

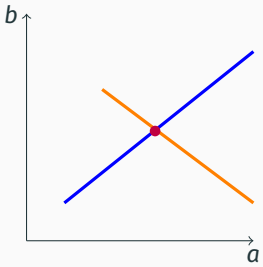


The families of lines passing through P_1 (or P_2) have equation: $b = -x_1 \cdot a + y_1$ (or $b = -x_2 \cdot a + y_2$).

Representing a line



Consider a line with equation: $y = a \cdot x + b$ and two points P_1 and P_2 on it.



The families of lines passing through P_1 (or P_2) have equation: $b = -x_1 \cdot a + y_1$ (or $b = -x_2 \cdot a + y_2$).

The intersection is the slope and intercept of the original line! Any point on the red line on the left will correspond to a line passing through the red point in the a/b space.

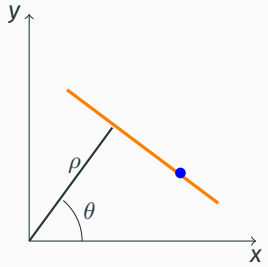
This is the principle of the Hough transform.

Representing a line - another way

We can also represent a line in terms of an angle and a distance from the origin. This is a better alternative, because we can also use it to represent a vertical line.

$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = \rho$$

Where θ is the angle of the line and *rho* is the distance from the origin.

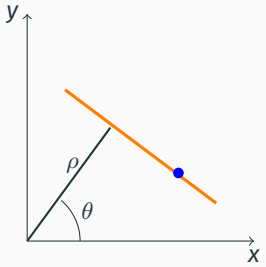


Representing a line - another way

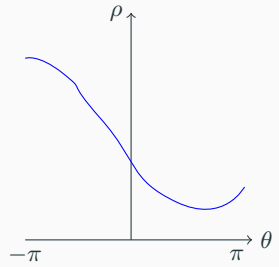
We can also represent a line in terms of an angle and a distance from the origin. This is a better alternative, because we can also use it to represent a vertical line.

$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = \rho$$

Where θ is the angle of the line and *rho* is the distance from the origin.



We can map each point in the θ/ρ space (called the Hough space) as we did before.



The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.

The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.

The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.
3. Iterate through all the pixels of the image and if they are an edge map they get to "cast a vote" onto the Hough space.

The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.
3. Iterate through all the pixels of the image and if they are an edge map they get to "cast a vote" onto the Hough space.
4. Each value in the Hough space matrix is use as an "accumulator". We add 1 to the value of the Hough space at the corresponding θ and ρ coordinates for each possible line passing through the edge point.

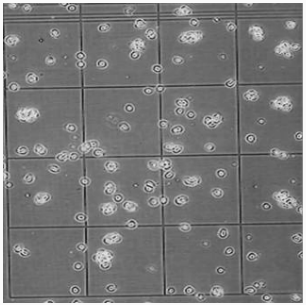
The Hough transform

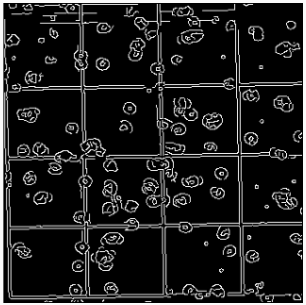
There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.
3. Iterate through all the pixels of the image and if they are an edge map they get to "cast a vote" onto the Hough space.
4. Each value in the Hough space matrix is use as an "accumulator". We add 1 to the value of the Hough space at the corresponding θ and ρ coordinates for each possible line passing through the edge point.
5. The Hough space matrix is then thresholded and non-maximum-suppression applied to find the strongest lines.

Hough transform - an example

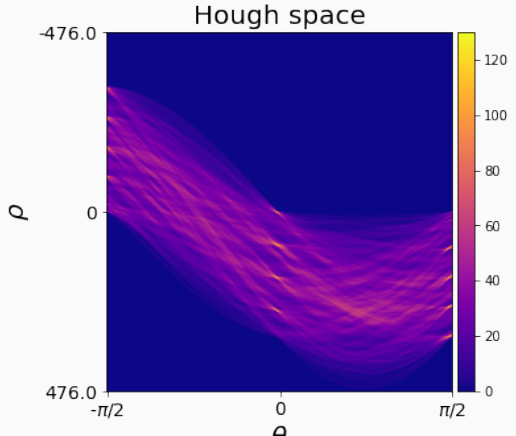
```
from skimage.feature import canny
from skimage.transform import hough_line, hough_line_peaks

chamber = imread("cell_counting_chamber.jpg")
chamber_canny = canny(chamber, sigma=1.5)
```





Hough transform - an example

```
hough_space, angles, d = hough_line(chamber_canny)
accum, theta, rho = hough_line_peaks(hough_space, angles, d)
for t, r in zip(theta, rho):
   print(f"np.rad2deg(t):0.2f, r:0.2f")
```




```
88.99, 253.77
88.99, 97.60
-1.51, 258.77
-1.51, 171.68
88.99, 332.85
-1.51, 84.59
88.99, 172.68
-1.51, 0.50
88.99, 23.52
```

The circle Hough transform

It is easy to extend the Hough transform to circles.
A circle with radius r and centre $(a; b)$ has equation

$$(x - a)^2 + (y - b)^2 = r^2$$

The circle Hough transform

It is easy to extend the Hough transform to circles.
A circle with radius r and centre $(a; b)$ has equation

$$(x - a)^2 + (y - b)^2 = r^2$$

We can create a 3D Hough space (a, b, r) and proceed as before, testing a number of different radii.

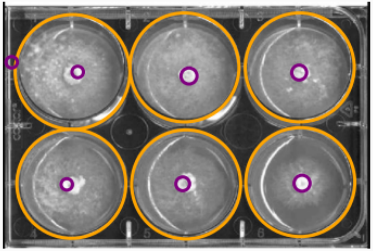
The circle Hough transform

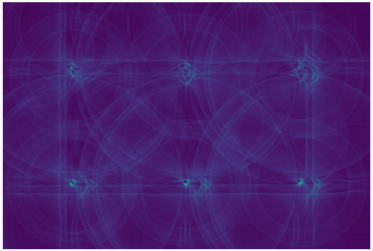
It is easy to extend the Hough transform to circles.
A circle with radius r and centre $(a; b)$ has equation

$$(x - a)^2 + (y - b)^2 = r^2$$

We can create a 3D Hough space (a, b, r) and proceed as before, testing a number of different radii.
The circle Hough transform is implemented in the `hough_circle` and `hough_circle_peaks` functions.

On the right: example of the circle Hough transform (top detected circles, bottom the Hough space for the larger radius). Note the spurious detection of a small circle at the top left of the image. - Source: Kwak et al, 2009





Texture features

Texture features

Texture features describe the local appearance of an image.
The texture of an image is determined by the spatial distribution of intensity levels in a neighborhood.

Given the neighborhood of a pixel, we can define basic texture features such as:

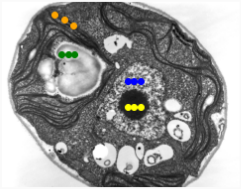
1. **Range**: max intensity - min intensity
2. **Variance**: variance of the intensity values in the neighborhood

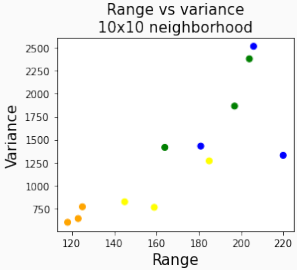
Texture features

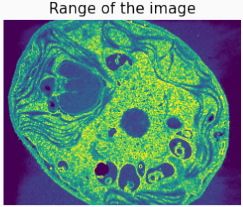
Texture features describe the local appearance of an image.
The texture of an image is determined by the spatial distribution of intensity levels in a neighborhood.

Given the neighborhood of a pixel, we can define basic texture features such as:

1. **Range**: max intensity - min intensity
2. **Variance**: variance of the intensity values in the neighborhood







TEM image of a Chlamydomonas green algae - CCo, Dartmouth College

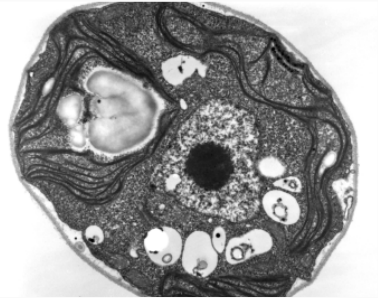
Entropy

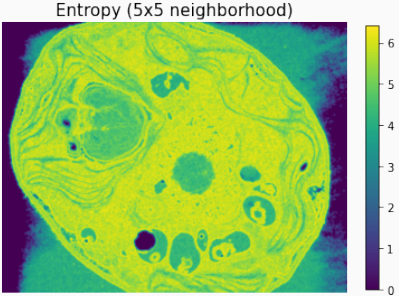
In information theory, **entropy** is a quantity related to the complexity of information in a system.

For an image, entropy is the minimum number of bits needed to encode the local greylevel distribution.

Entropy can be calculated using the `skimage.filters.rank.entropy` function.

```
from skimage.filters.rank import entropy
im_entr = entropy(image, selem=np.ones((5, 5))
```





GLCM features

A more quantitative approach to texture features is the **grey-level co-occurrence matrix (GLCM)**.

A GLCM is definted as a $n \times n$ matrix of values counting the pixels with a specific intensity at a certain distance and angle in the image. n is the number of grey levels in the image.

Example GLCM

Image

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

GLCM (distance=1, angle=0)

	0	***1***	***2***	***3***
0	0	0	1	0
1	1	3	3	3
2	0	5	1	0
3	0	1	1	1

For example, the GLCM above shows that there are five pixels with intensity 2 and 1 at distance 1 and angle 0 (so to the right).

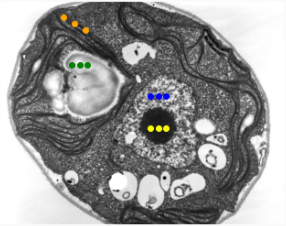
What to do with a GLCM?

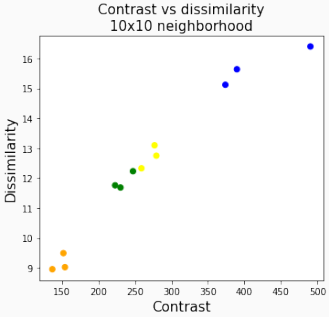
The GLCM itself is not very useful for our purposes, but we can calculate numeric features from it. After normalising the GLCM to a sum of 1, we can calculate:

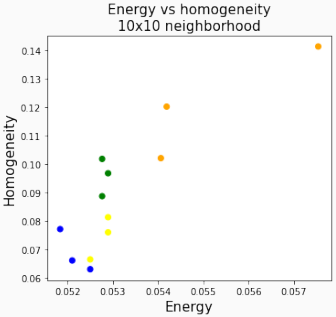
- **Contrast** - a measure of the contrast between each pixel and its neighbor. $\sum_{i,j=0}^{levels-1} P_{i,j}(i-j)^2$
- **Dissimilarity** - a measure of distance between each pixel and its neighbor. $\sum_{i,j=0}^{levels-1} P_{i,j}|i-j|$
- **Homogeneity** - a measure of the closeness of the distribution of elements in the GLCM - $\sum_{i,j=0}^{levels-1} \frac{P_{i,j}}{1+(i-j)^2}$
- **ASM** (angular second moment) - - $\sum_{i,j=0}^{levels-1} P_{i,j}^2$
- **Energy** - $\sqrt{ASM}$
- **Correlation** - how correlated neighbour pixels are in an image - $\sum_{i,j=0}^{levels-1} P_{i,j} \frac{(i-\mu_i)(j-\mu_j)}{\sqrt{\sigma_i^2 \sigma_j^2}}$

Again, these are for reference only... don't worry about remembering the formulas!

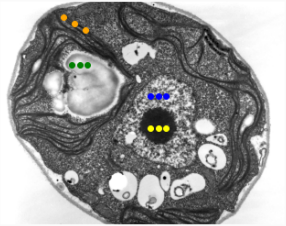
GLCM features - example

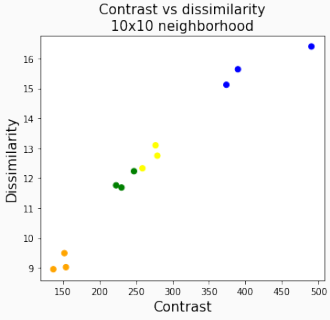


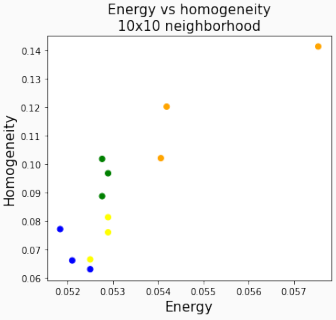




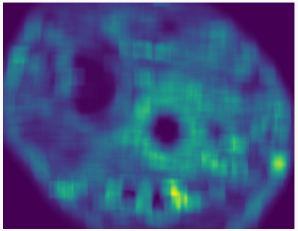
GLCM features - example



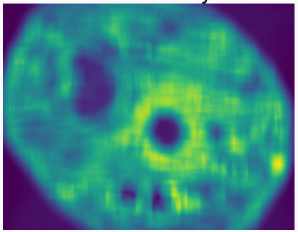




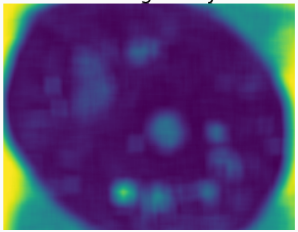
Contrast



Dissimilarity



Homogeneity



HOG features

Histogram of oriented gradients (HOG) features are a set of features that describe the local gradient orientation of an image. They are commonly used for object detection.

Overview of the algorithm

1. **Global image normalisation** - an optional step to reduce the influence of local changes in illumination. Usually done by computing the log or square root of the image.

HOG features

Histogram of oriented gradients (HOG) features are a set of features that describe the local gradient orientation of an image. They are commonly used for object detection.

Overview of the algorithm

1. **Global image normalisation** - an optional step to reduce the influence of local changes in illumination. Usually done by computing the log or square root of the image.
2. **Compute** ∇I - some variant methods calculate second derivatives instead.

HOG features

Histogram of oriented gradients (HOG) features are a set of features that describe the local gradient orientation of an image. They are commonly used for object detection.

Overview of the algorithm

1. **Global image normalisation** - an optional step to reduce the influence of local changes in illumination. Usually done by computing the log or square root of the image.
2. **Compute** ∇I - some variant methods calculate second derivatives instead.
3. **Compute gradient histograms** - the image is divided in cells and the histogram of the gradient orientation in each cell is computed.

HOG features

Histogram of oriented gradients (HOG) features are a set of features that describe the local gradient orientation of an image. They are commonly used for object detection.

Overview of the algorithm

1. **Global image normalisation** - an optional step to reduce the influence of local changes in illumination. Usually done by computing the log or square root of the image.
2. **Compute** ∇I - some variant methods calculate second derivatives instead.
3. **Compute gradient histograms** - the image is divided in cells and the histogram of the gradient orientation in each cell is computed.
4. **Normalise across blocks** - we define a block as a group of cells, and normalise the histograms of the gradient orientations in each block.

HOG features

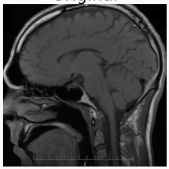
Histogram of oriented gradients (HOG) features are a set of features that describe the local gradient orientation of an image. They are commonly used for object detection.

Overview of the algorithm

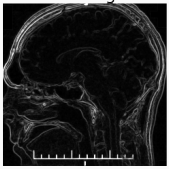
1. **Global image normalisation** - an optional step to reduce the influence of local changes in illumination. Usually done by computing the log or square root of the image.
2. **Compute** ∇I - some variant methods calculate second derivatives instead.
3. **Compute gradient histograms** - the image is divided in cells and the histogram of the gradient orientation in each cell is computed.
4. **Normalise across blocks** - we define a block as a group of cells, and normalise the histograms of the gradient orientations in each block.
5. **Flatten** results into a feature vector for further use.

Example of HOG features

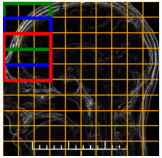
Original



Gradient magnitude

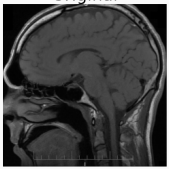


Cells and blocks

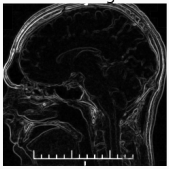


Example of HOG features

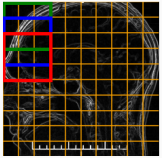
Original



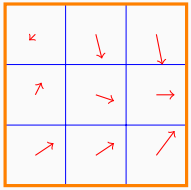
Gradient magnitude



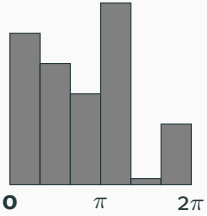
Cells and blocks



Gradients in a cell

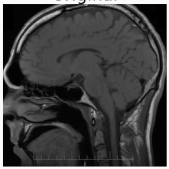


Histogram of gradients

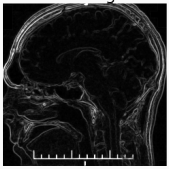


Example of HOG features

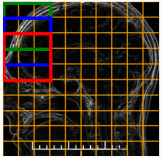
Original



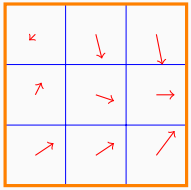
Gradient magnitude



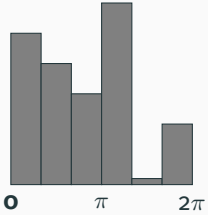
Cells and blocks

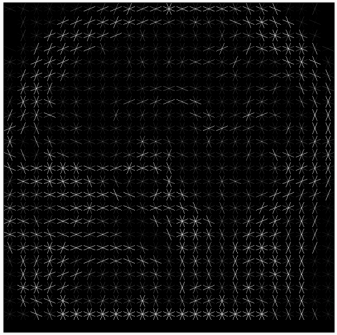


Gradients in a cell



Histogram of gradients





HOG features in Scikit-Image

```
from skimage.feature import hog
from skimage.filters.edges import sobel
from skimage.io import imread

img = imread('MRI.jpg')
img_sobel = sobel(img)
# Because visualize is True, hog returns a tuple of (feature vector, HOG image)
fd, hog_image = hog(img_sobel, orientations=4,
pixels_per_cell=(32, 32), cells_per_block=(5, 5),
visualize=True)
```

Summary

1. Feature extraction is a very important step in image analysis.
2. Features can be fed to machine learning models, or used to detect specific objects in an image.
3. Many other feature detectors exist, and many are implemented in `skimage.feature`