Image similarities and clustering

Anna Monreale and Francesca Naretto
Computer Science Department

Introduction to Data Mining, 2nd Edition
Chapter 1 & Data Exploration (Additional Resources)

Clustering analysis

Objective: finding groups of objects such that in a group the objects are similar, while they are different w.r.t. the objects in another group.

Hence, to have a clustering we need to understand what we mean for similarity among images.

Clustering characteristics

They exploit a proximity or density measure.

How to evaluate the proximity/similarity among images?

Also in this case, the image characteristics may affect proximity measures, such as strange distributions of the data, different light conditions, different position of the object in the image.

How to evaluate similarity in images?

We want a number that tells us how similar two images are:

- Feature based approaches
- Frequency based approaches
- Structural similarity
- Histogram based similarity
- Deep learning based approaches

How to evaluate similarity in images?

Simplest way: Euclidean Distance.

As for tabular data, when we are working with numbers we can compare them using Euclidean Distance.

In particular, with images, we can compare them by calculating the distance pixel by pixel.

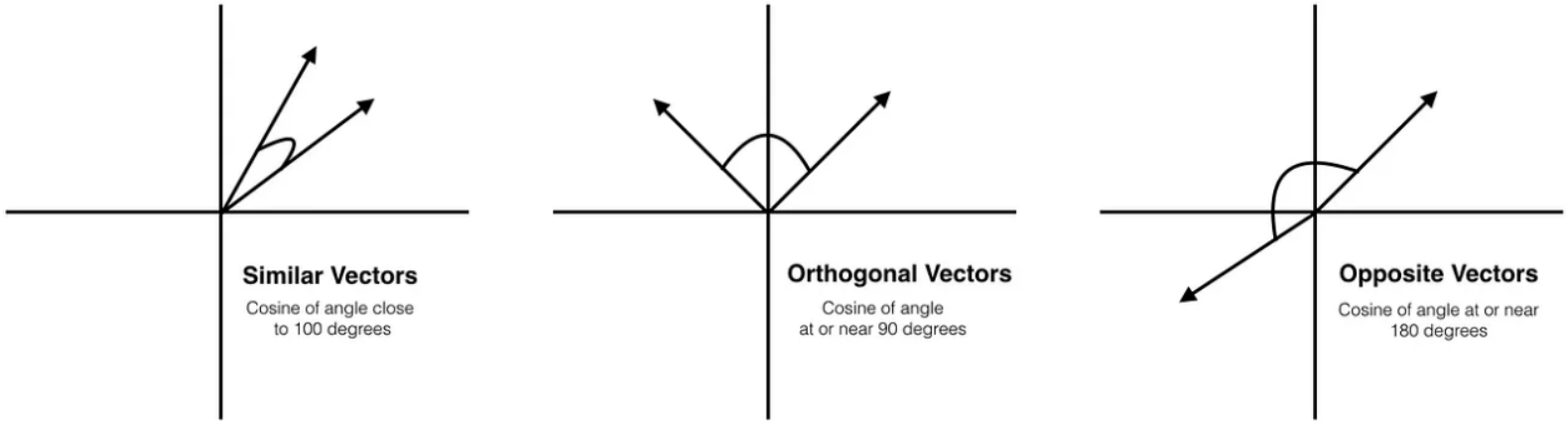
Drawbacks:

Affected by changing of lights, positions etc.

Cosine Similarity

This metric operates on the image vectors. Given two image vectors, representing the pixel values of images.

To compare the two vectors, we measure the cosine of the angle between the vectors.



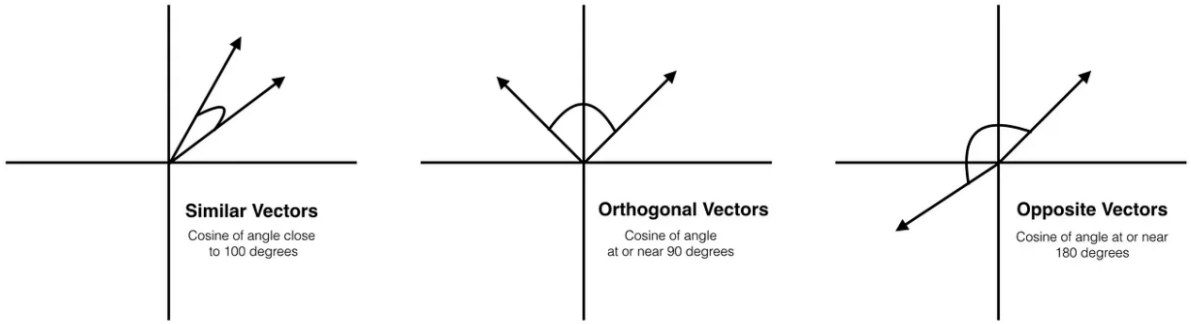
Cosine Similarity

The formula for evaluating the cosine similarity is:

$$\frac{\sum_{i=1}^{n} A_i . B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

The dot product of the 2 vectors, while the denominator is the dot product of the size of the 2 vectors

Close to 1



Close to -1

Mean Squared Error (MSE)

This metric measures the average squared difference between the pixel values of two images.

Simple, widely used for its simplicity, easy to apply, reasonably good results and fast.

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

How to evaluate similarity in images?

Feature descriptors

score[0]	score[1]	score[2]	score[3]	score[4]	score[5]	score[6]
1	2	3	4	5	6	7
1000	1002	1004	1006	1008	1010	1012

Histogram based approaches

These methods capture the distribution of pixels in an image.

By comparing histograms, you can measure similarity.

PROS: it is not affected by images of different dimensions.

Histogram based approaches

These methods capture the distribution of pixels in an image.

By comparing histograms, you can measure similarity.

PROS: it is not affected by images of different dimensions.

To compare the similarity, we can then use:

- Histogram Intersection (the larger the value, greater is the similarity)
- Histogram correlation (greater correlation means greater similarity)

Histogram of oriented gradients (HOG)

It computes the histogram of gradients which are used as the features of an image. Hence, we obtain a feature descriptor in the end.

How to calculate the gradients for images?

You can find the full description on this paper:
https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Histogram of oriented gradients (HOG)

1. Pre-process the data
    1. We need images with the width to height ratio to 1:2. (Later on we split the image)

    At this point, we can calculate the gradients on the x and y directions. Let's assume we calculate it only for a portion of the image at each time.

You can find the full description on this paper:
https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Histogram of oriented gradients (HOG)

At this point, we can calculate the gradients on the x and y directions. Let's assume we calculate it only for a portion of the image at each time.

Our image:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

-1	0	1

-1
0
1

Histogram of oriented gradients (HOG)

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

-1	0	1

-1
0
1

For the pixel 85:
Gradient in x direction (change in x direction): G_x = 89-78 = 11
Gradient in y direction (change in y direction): G_y = 68-56 = 8

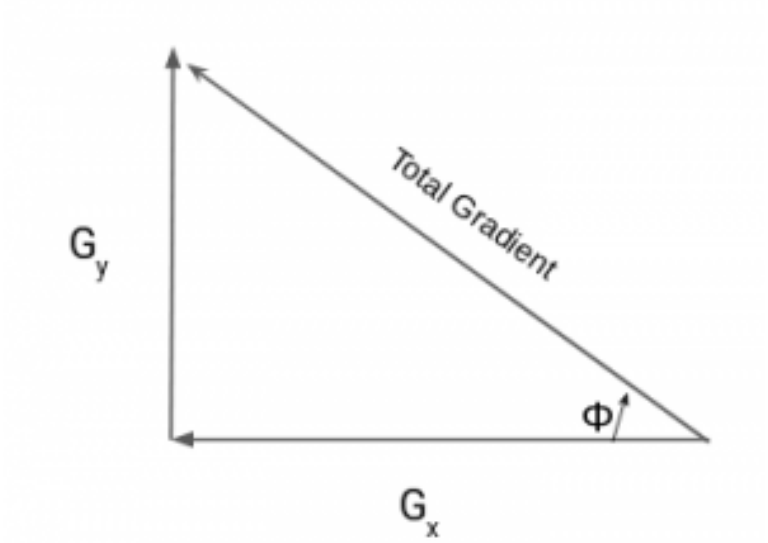
Histogram of oriented gradients (HOG)

- We need to repeat the computation considering all the pixels (kernel again!)
- In the end we are going to have 2 matrices, one with the gradients in the x direction and the other with the gradients in y direction
- General idea: the magnitude will be higher when there is a sharp change in the intensity of the pixel values

You can find the full description on this paper:
https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

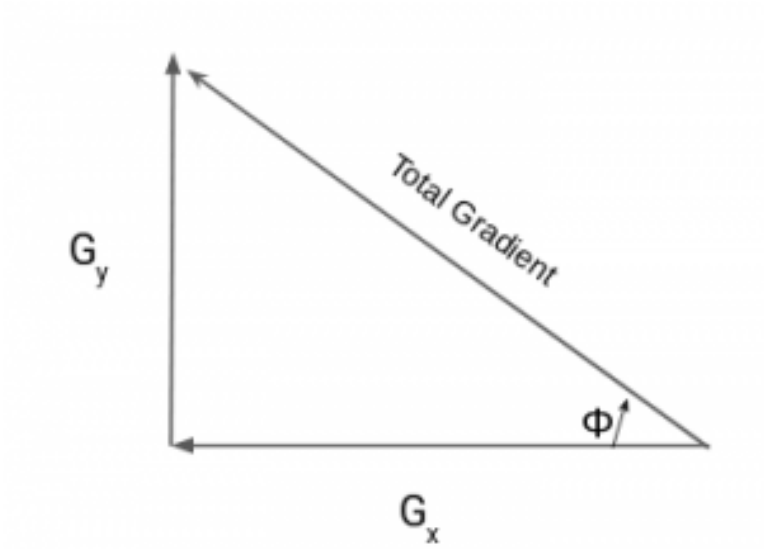
Histogram of oriented gradients (HOG)

Now we have computed the gradients.
How to calculate the magnitude and orientation of the gradients?



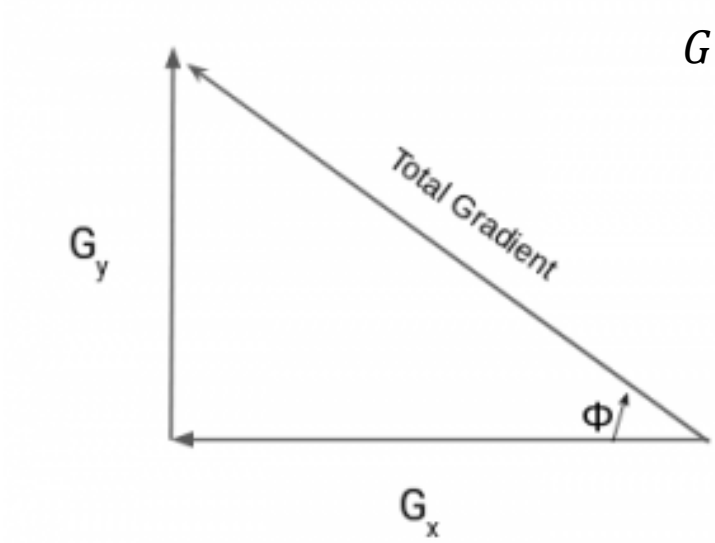
Histogram of oriented gradients (HOG)

Pitagora's theorem!



Histogram of oriented gradients (HOG)

Pitagora's theorem!



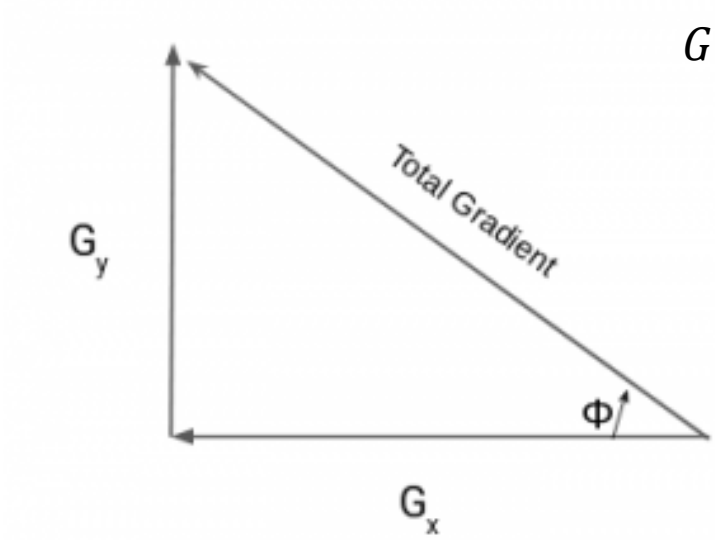
$$G = \sqrt{(G_x^2 + G_y^2)}$$

$$\tan(\phi) = \frac{G_y}{G_x}$$

$$\phi = \text{atan}(\frac{G_y}{G_x})$$

Histogram of oriented gradients (HOG)

For each pixel we calculate these two values to obtain magnitude and orientation.



$$G = \sqrt{(G_x^2 + G_y^2)}$$

$$\tan(\phi) = \frac{G_y}{G_x}$$

$$\phi = \operatorname{atan}(\frac{G_y}{G_x})$$

Histogram of oriented gradients (HOG)

Now that we have computed the gradients, their orientations and magnitudes, we can create the histogram.

Simple method:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Frequency						1									
Angle	1	2	3	4 ...	35	36	37	38	39....	175	176	177	178	179	180

Histogram of oriented gradients (HOG)

Now that we have computed the gradients, their orientations and magnitudes, we can create the histogram.
'Maybe better' method:

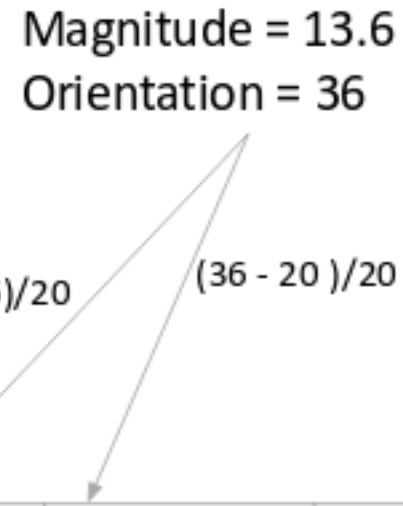
Magnitude = 13.6
Orientation = 36

Magnitude		13.6							
Bin	0	20	40	60	80	100	120	140	160

Histogram of oriented gradients (HOG)

Now that we have computed the gradients, their orientations and magnitudes, we can create the histogram.

Best** method:



Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

Histogram of oriented gradients (HOG)

- This method is usually fast and accurate, without exploiting any machine learning based approaches.
- It exploits sliding window to compute the histogram for sub-spaces of the image.
- It works well both for gray scale images and color ones
- It does not need a lot of pre-processing since it can handle changes in lights and small noise

You can find the full description on this paper:
https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Histogram of oriented gradients (HOG)

1. It takes as input the original image
2. Divides it in block of equal size
3. Each block is divided again into cells of equal size
4. At this point, for each cell a histogram is generated
5. Then, a block normalization (like L2 norm) is applied to overcome small problems like changing of lights

Structural similarity Index (SSIM)

This method was first introduced to compare two versions of the same image with the aim of evaluating the quality of the image after compression or pre-processing techniques. Hence you can compare the original image vs. its modified version.

It can be used also for evaluating the similarity between 2 different images.

It considers luminance, contrasts and structure of the images.

Structural similarity Index (SSIM)

1. It is a sliding window technique that calculates mean, variance and covariance for each window.
2. It then computes a single value, which represent the Structural Similarity between the 2 images
3. -1 means the two images are different, 1 they are identical.

NB: it is affected by different dimensions of the images. Pre-processing, like background subtraction or noise removing is fundamental for it to work as a similarity between images.

Scale Invariant Feature Transform (SIFT)

It is a feature based approach that identifies key points in the image and then compares them to detect similarities. The output is a feature descriptor again.

1. Constructing a Scale Space
2. Keypoint Localization
3. Orientation Assignment
4. Keypoint descriptor

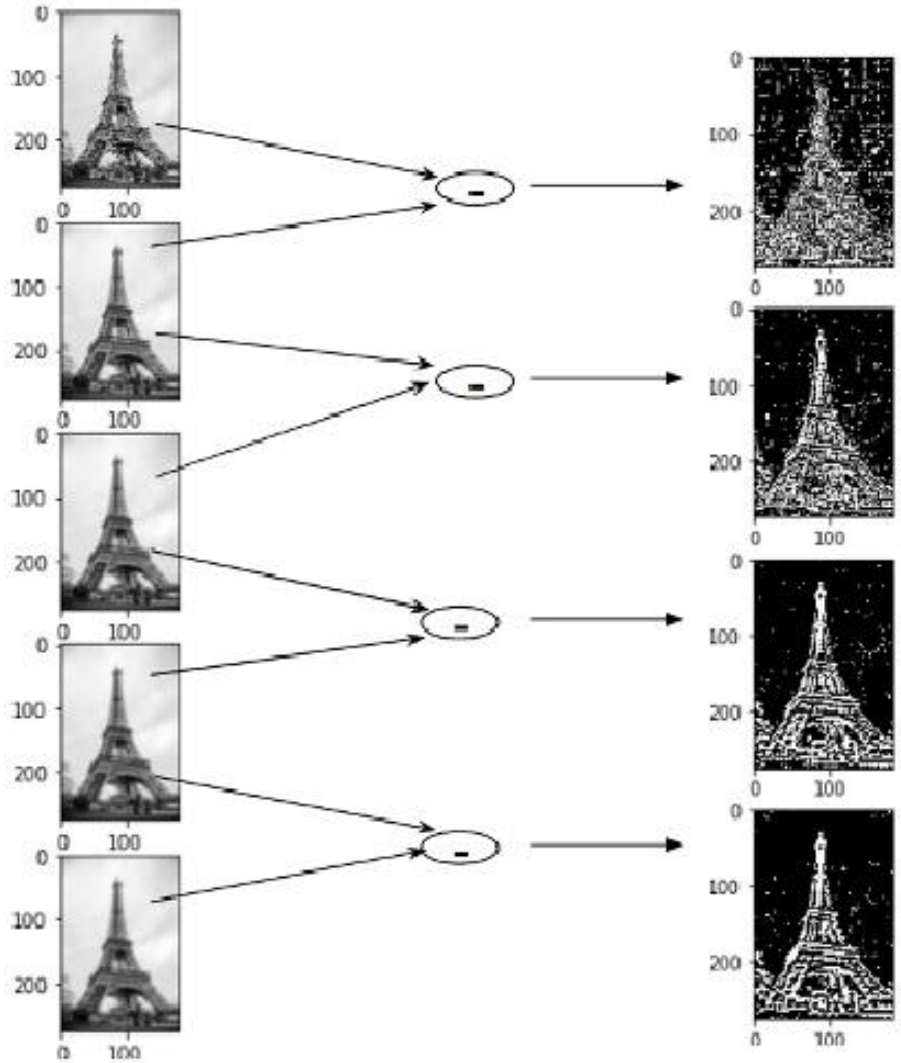
Scale Invariant Feature Transform (SIFT)

It is a feature based approach that identifies key points in the image and then compares them to detect similarities.

1. Constructing a Scale Space: Objective: make sure that the keypoint are not scale-dependent.
    1. Create a collection of images from the original one, with different scales
    2. Apply Gaussian Blur to all of them by different amounts
    3. Difference of gaussian kernel: calculate the difference of Gaussian Blurs of an image with different scales

Scale Invariant Feature Transform (SIFT)

Difference of Gaussian kernels



Scale Invariant Feature Transform (SIFT)

It is a feature based approach that identifies key points in the image and then compares them to detect similarities.

1. Constructing a Scale Space
2. Keypoint Localization
    1. Find local minima and maxima by comparing each pixel with its neighbors and the ones in the same space but in the close images on different scales.
    2. Drop low contrast points or edge points

Scale Invariant Feature Transform (SIFT)

It is a feature based approach that identifies key points in the image and then compares them to detect similarities.

1. Constructing a Scale Space
2. Keypoint Localization
3. Orientation Assignment: to make the keypoint invariant to rotation
    1. Calculate magnitude and orientation
    2. Create an histogram and keep only the points above 80% in the peaks.

Scale Invariant Feature Transform (SIFT)

It is a feature based approach that identifies key points in the image and then compares them to detect similarities.

1. Constructing a Scale Space
2. Keypoint Localization
3. Orientation Assignment
4. Keypoint descriptor: generate a fingerprint for each keypoint.

Scale Invariant Feature Transform (SIFT)

1. It is a feature based approach.
2. It is not affected by the size or the orientation of the image
3. It is scale invariant, rotation invariant and robust
4. It looks at the edges, key points, corners and so on. After having identified the most salient points of the image, they are compared to evaluate the similarity.

Template Matching

Given the template of the object we are interested in, we can compare the template with the image under analysis via sliding.

We slide the template over the image and we calculate how similar the sub-image and the template are.

We can use various metrics to evaluate the similarity, such as correlation or sum of square differences.

Fourier Transformations

A transformation is a mapping between domains.

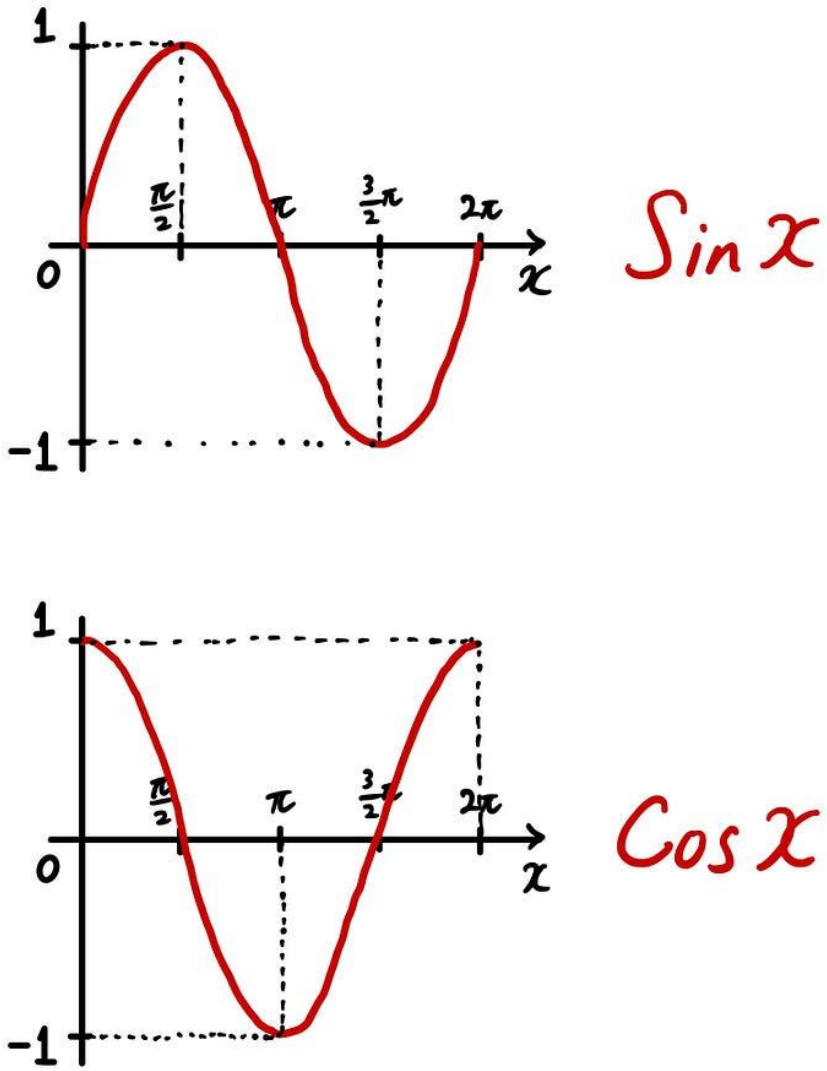
We represent the same information in 2 different domains.

In our case, we want to transform our information into something that is easier to work with.

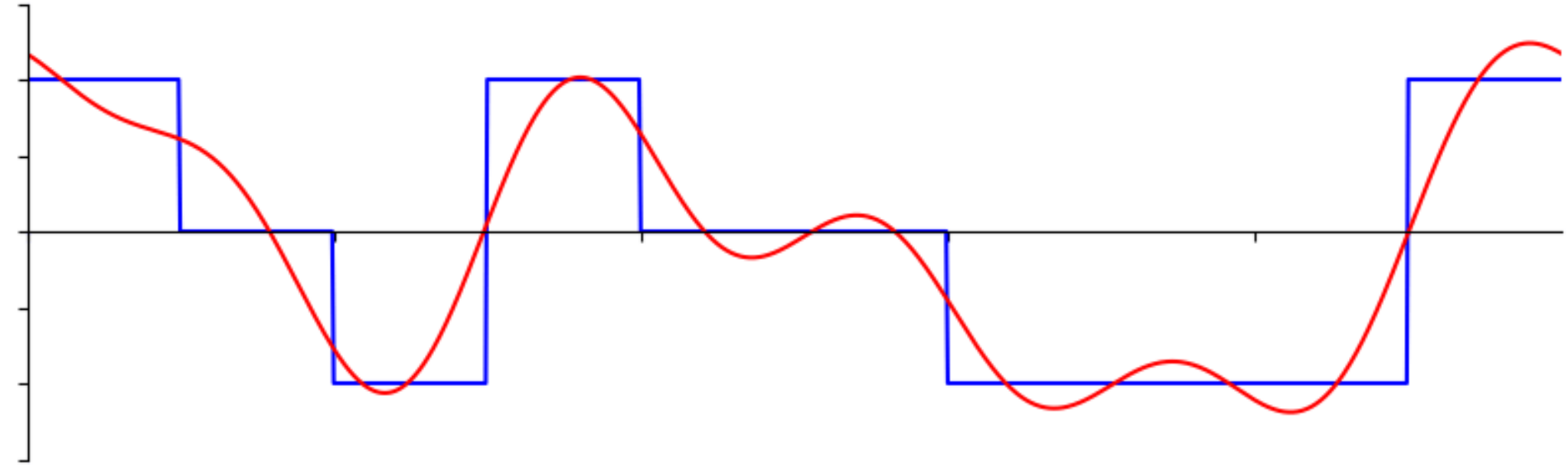
In this case, it is a mapping between a signal in the time domain ($f(t)$) and the frequency domain F(v), where f is the function we are looking at, t is the time and F is the frequency.

Fourier Transformations

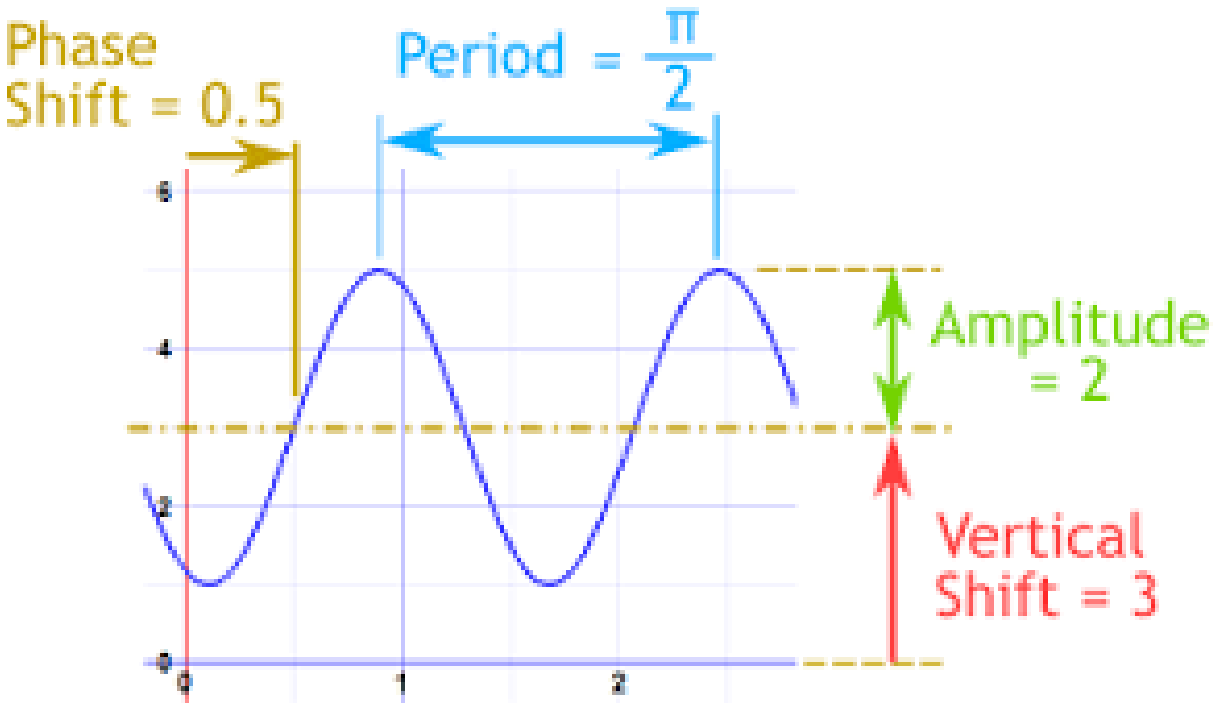
Fourier Transform approximates the signal using different sine and cosine waves, combined linearly.



Fourier Transformations



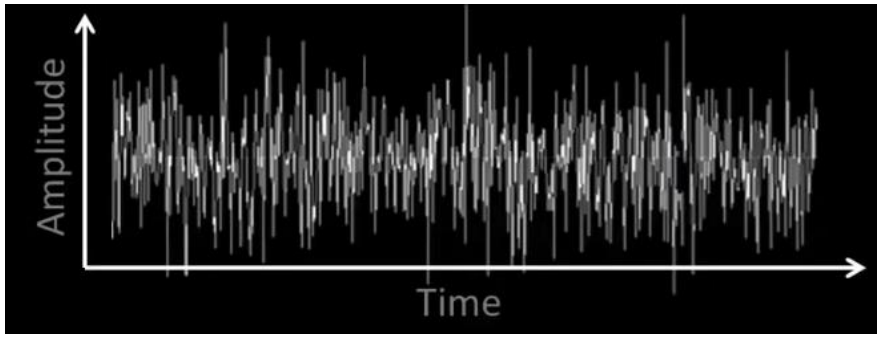
Fourier Transformations



Fourier Transform

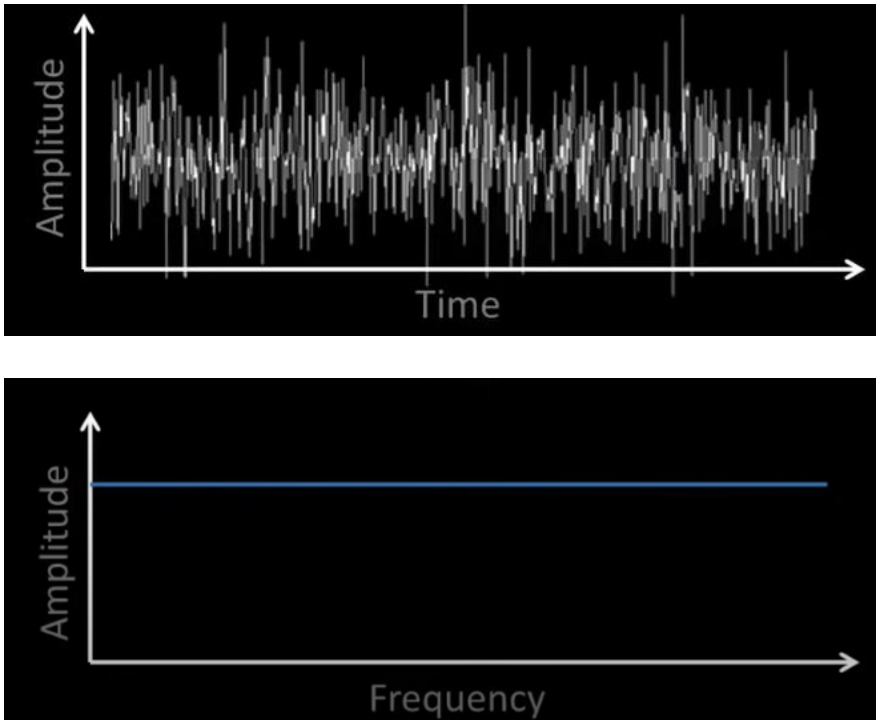
$$f(t) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty}(a_k cos2\pi kt + b_k sin2\pi kt)$$

Given any periodic function in time, we can decompose it into a sum of a constant and a series of sines and cosines, with their individual frequencies. Once you have the signal represented in the frequency domain, you can go back to your original signal without loss!



Fourier Transform

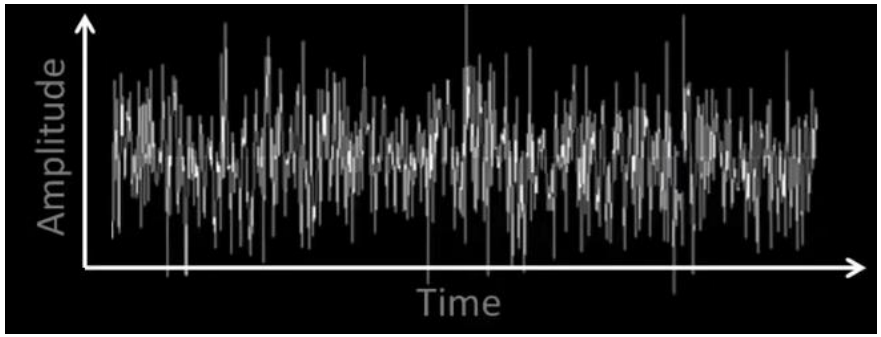
$$f(t) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty}(a_k cos2\pi kt + b_k sin2\pi kt)$$



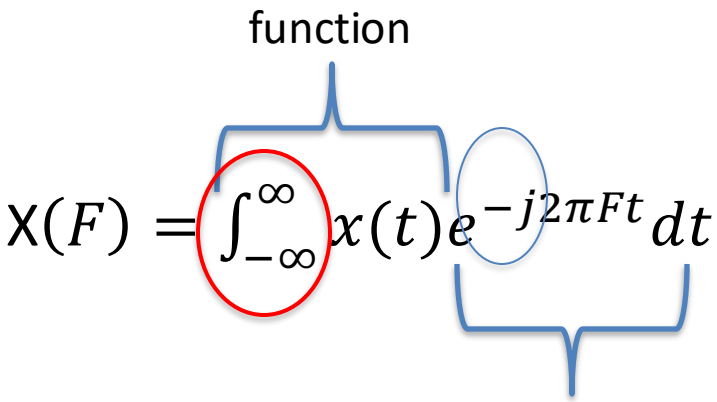
Fourier Transform

$$f(t) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty}(a_k cos2\pi kt + b_k sin2\pi kt)$$

How to calculate these coefficients at each particular frequency?



Fourier Transform



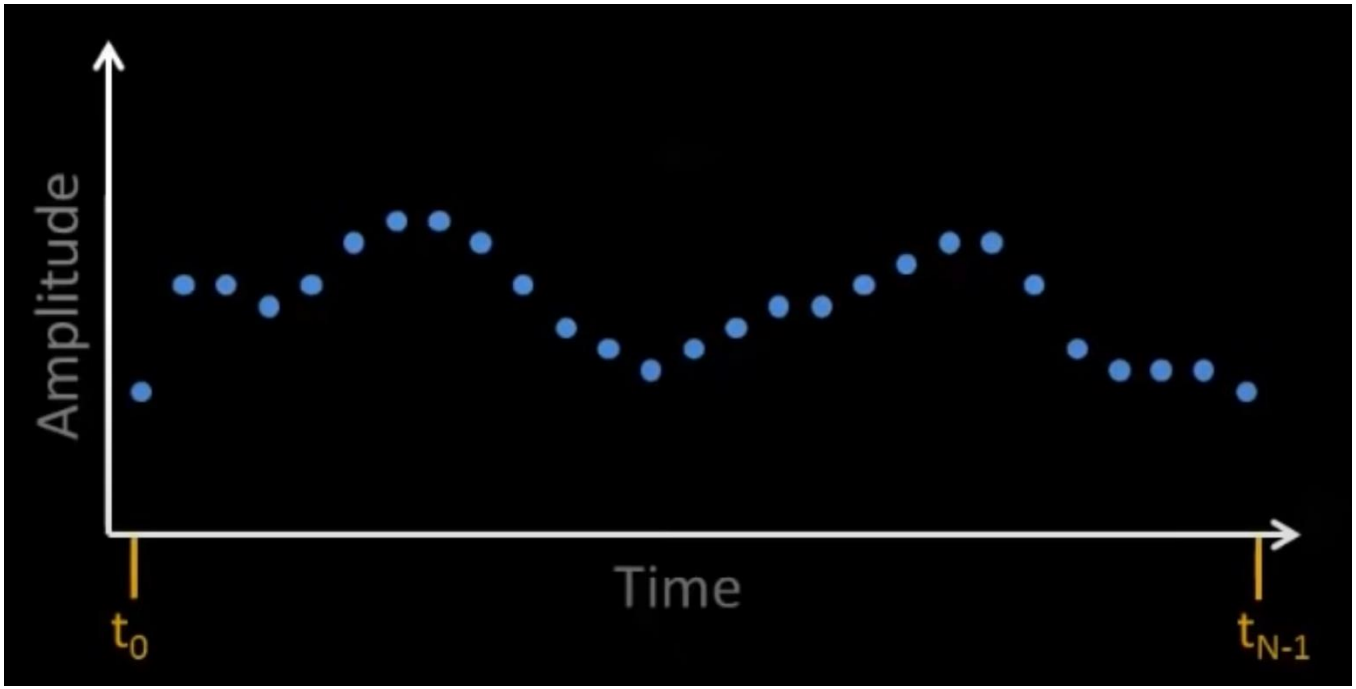
General idea: It is a complex number cause it holds all the information about the different sinusoids, such as phase, amplitude, etc.
But we have one complex coefficient per frequency!

Analysing function: sinusoids

$$Xa(F) = \int_{-\infty}^{\infty} x(t) cos 2\pi f t dt \qquad Xb(F) = \int_{-\infty}^{\infty} x(t) sin 2\pi f t \, dt$$

In this way we have 2 real coefficients per frequency

Fourier Transform



In discrete domain:

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi kn}{N}}$$

Fourier Transform: expansion

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi kn}{N}} = b_n$$

$$X_k = x_0 e^{-b_0 j} + x_1 e^{-b_1 j} + x_2 e^{-b_2 j} + \cdots + x_n e^{-b_n j}$$

K-th frequency bin

n-th sample

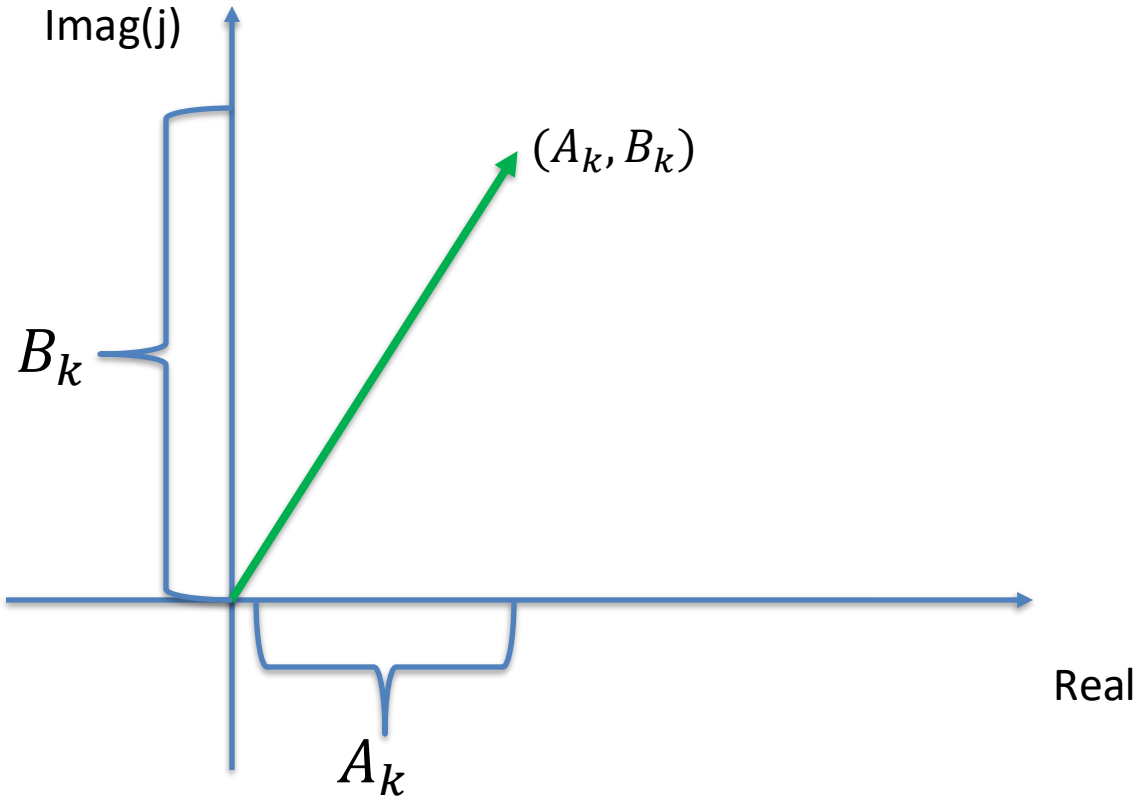
Euler's formula: $e^{jx} = \cos x + \mathrm{j} \sin x$

$$X_k = x_0[cos(-b_0) + jsin(-b_0)] + \cdots + x_n[cos(-b_n) + jsin(-b_n)]$$

$$X_k = A_k + B_k j$$

Fourier Transform

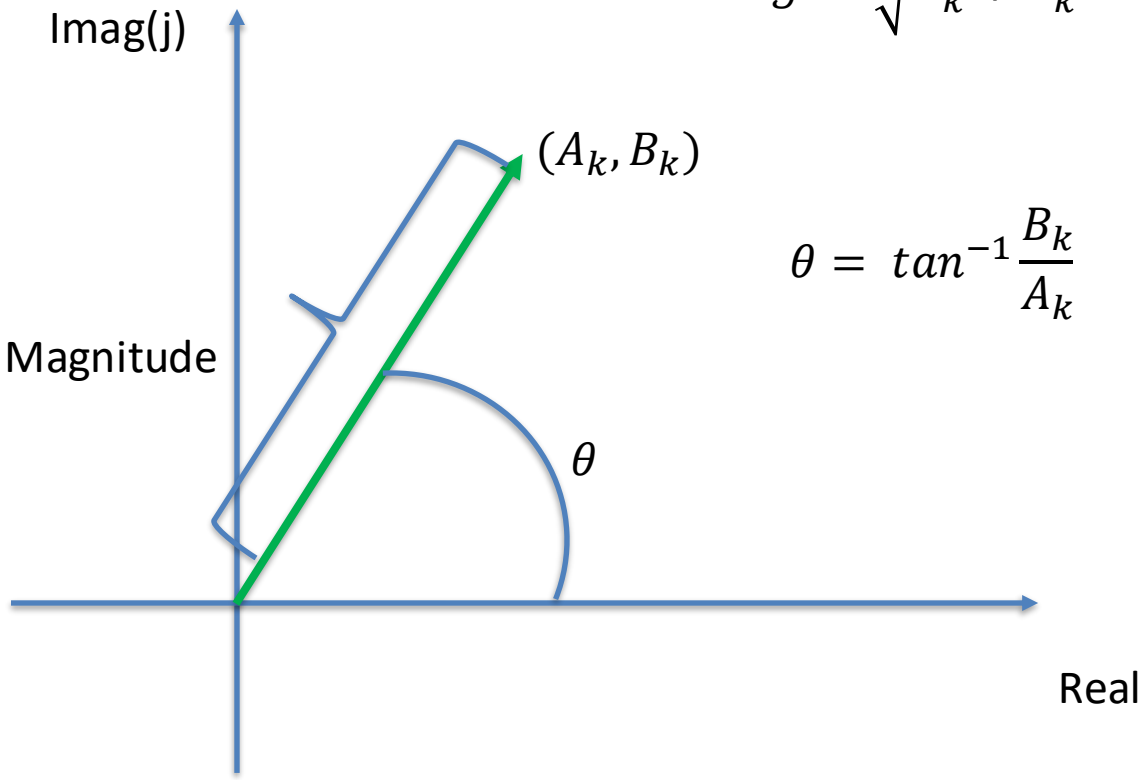
$$X_k = A_k + B_k j$$



Fourier Transform

$$X_k = A_k + B_k j$$

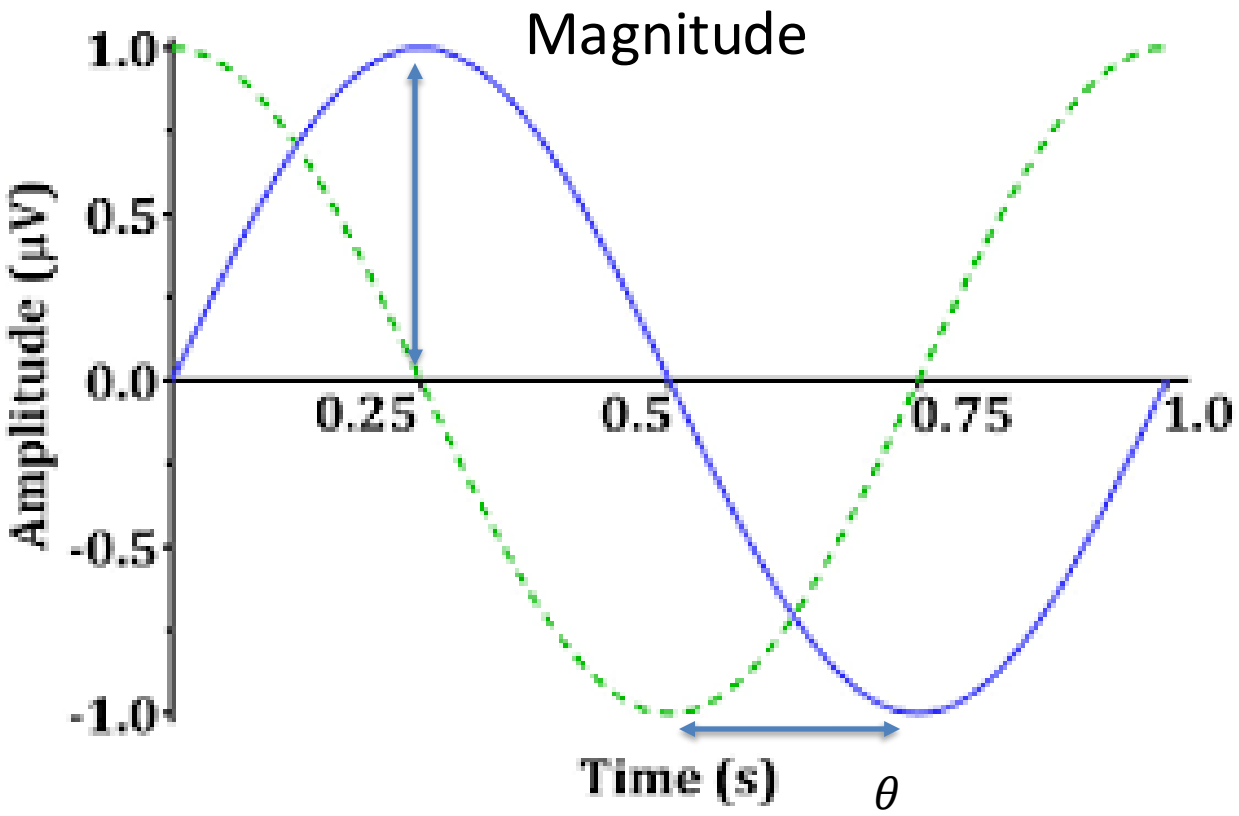
$$Mag = \sqrt{A_k^2 + B_k^2}$$



$$\theta = tan^{-1}\frac{B_k}{A_k}$$

Fourier Transform

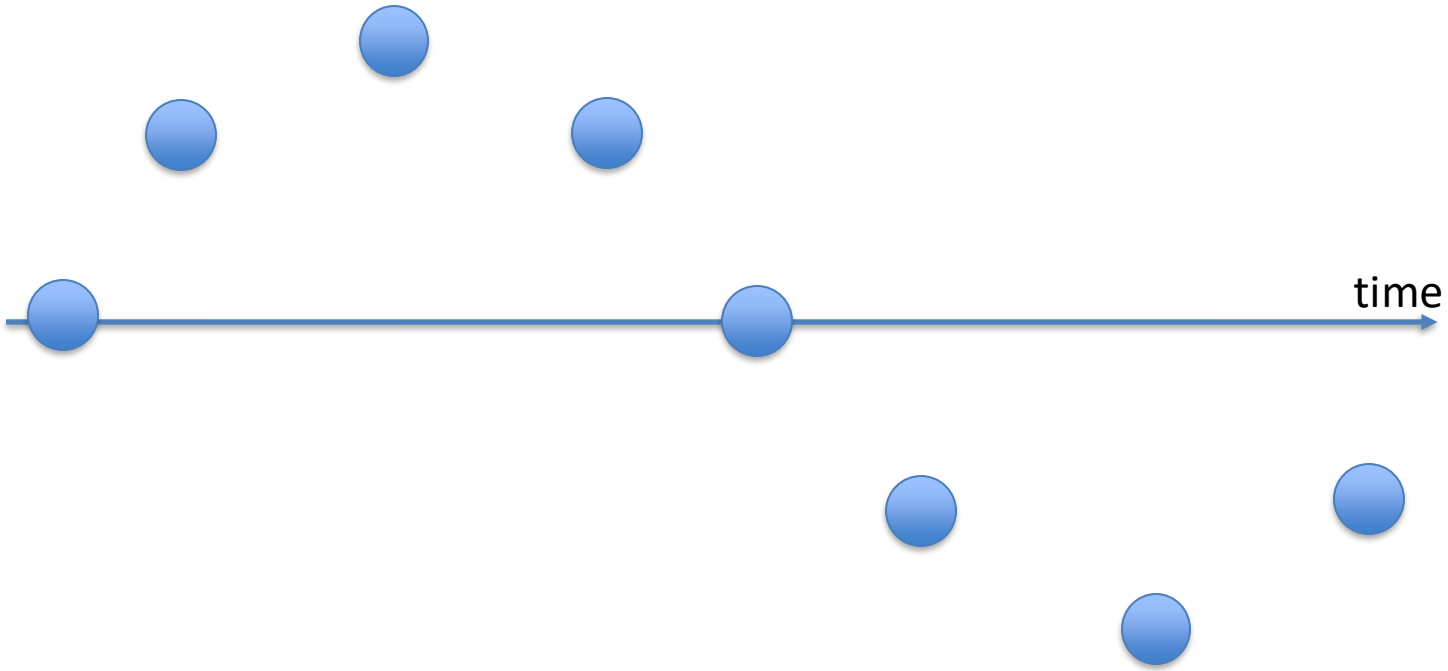
$$X_k = A_k + B_k j$$



Fourier Transform: example

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi kn}{N}}$$

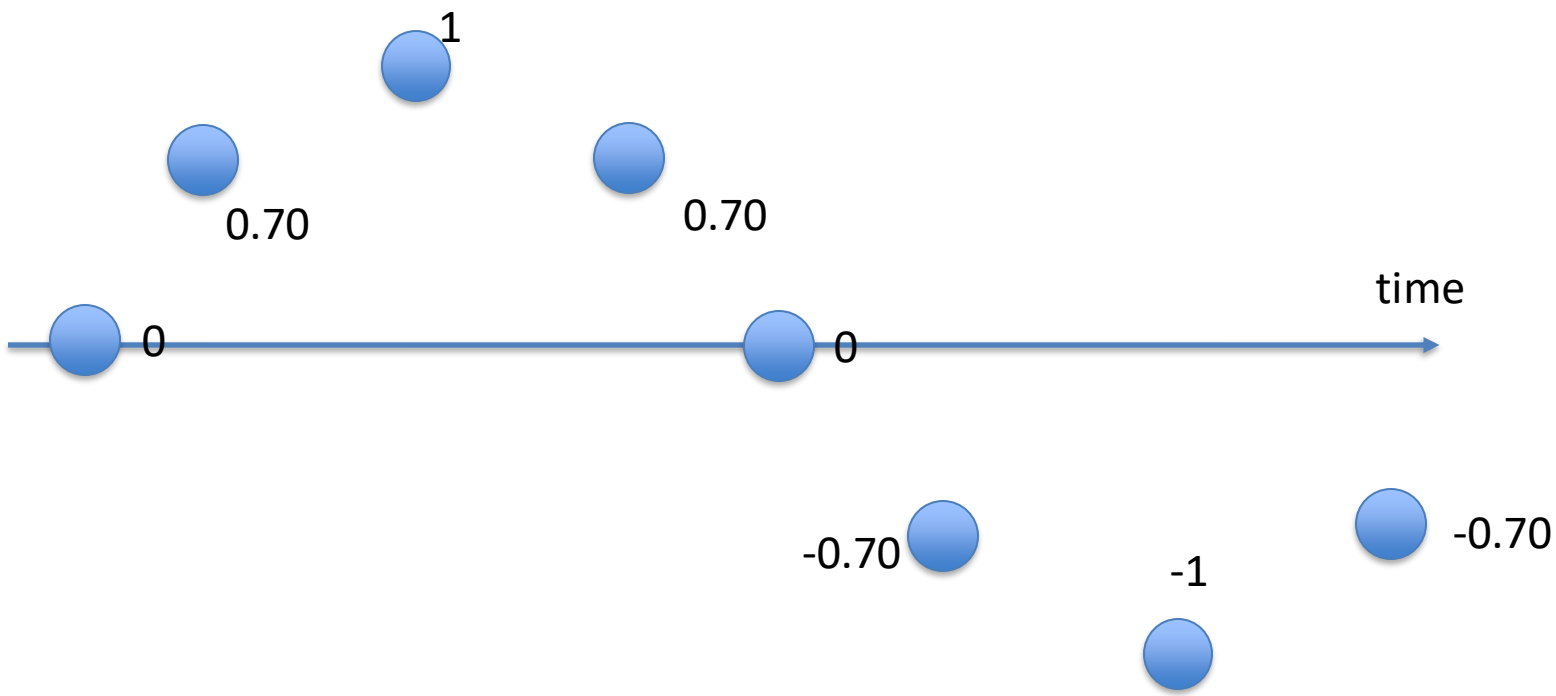
Sine wave of 1Hz
Amplitude of 1
Sampling frequency 8 Hz
8 samples



Fourier Transform example

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi kn}{N}}$$

Sine wave of 1Hz
Amplitude of 1
Sampling frequency 8 Hz
8 samples



Fourier Transform example

Sine wave of 1Hz
Amplitude of 1
Sampling frequency 8 Hz
8 samples

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi kn}{N}}$$

$x_0 = 0$

$x_1 = 0.70$

$x_2 = 1$

$x_3 = 0.70$

$x_4 = 0$

$x_5 = -0.70$

$x_6 = -1$

$x_7 = -0.70$

$$X_0 = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi 0n}{N}} = 0+0.70+\ldots-0.70 = 0$$

$$X_1 = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi 1n}{N}} = 0 * e^{-\frac{j2\pi(1)(0)}{8}} +$$

$$0.70 * e^{-\frac{j2\pi(1)(1)}{8}} + 1 * e^{-\frac{j2\pi 1(1)(2)}{8}} \pm \cdots$$

$$X_1 = 0 + 0.70\left[\cos\left(-\frac{\pi}{4}\right)\right] + jsin(-\frac{\pi}{4})] + 1[\cos\left(-\frac{\pi}{2}\right) + jsin(-\frac{\pi}{2})] + \cdots.$$

Fourier Transform example

Sine wave of 1Hz
Amplitude of 1
Sampling frequency 8 Hz
8 samples

$X_0 = 0$

$X_1 = 0 - 4\text{j}$

$X_2 = 0$

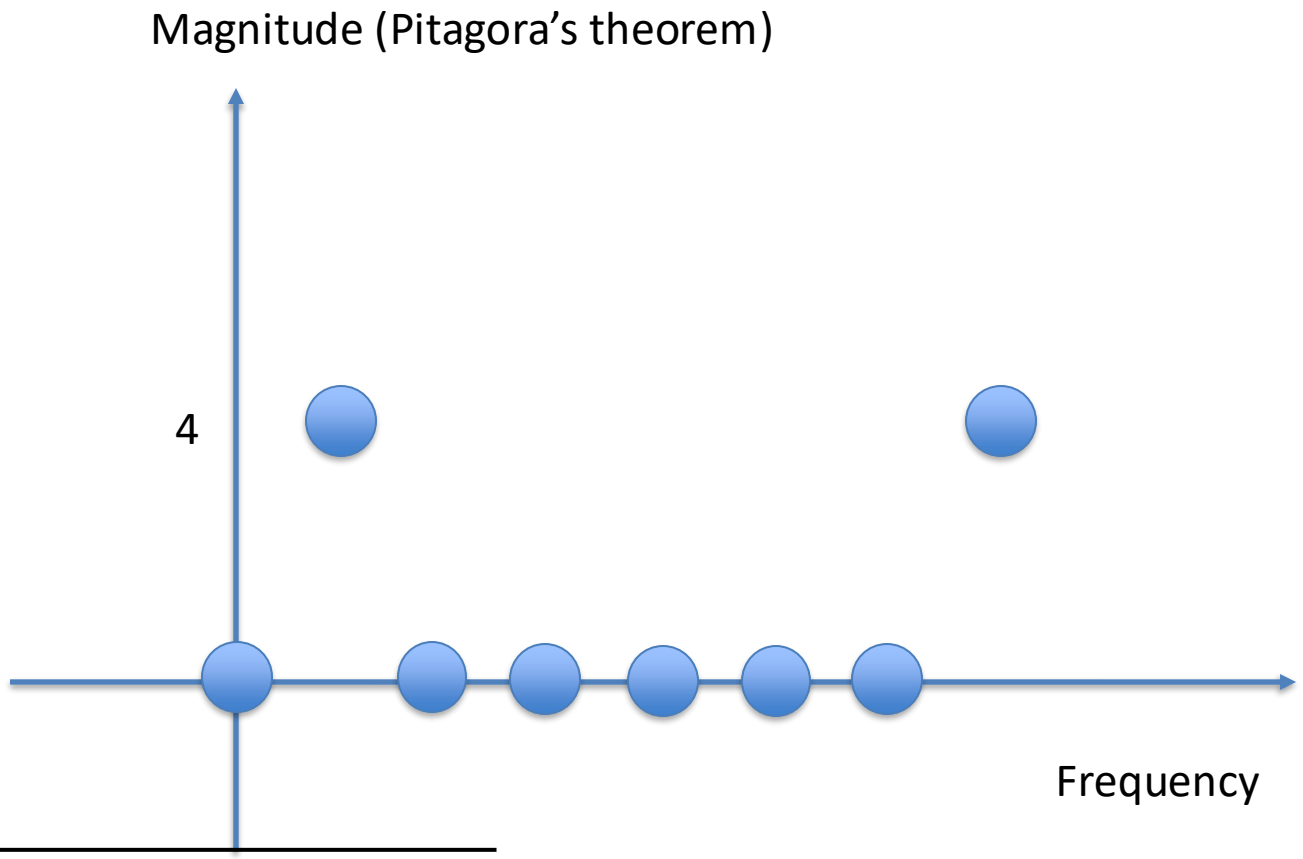
$X_3 = 0$

$X_4 = 0$

$X_5 = 0$

$X_6 = 0$

$X_7 = 0 + 4\text{j}$



$$Mag = \sqrt{A_k^2 + B_k^2} = \sqrt{0^2 + (-4)^2} = 4$$

Fourier Transform and the Nyquist limit

Imagine you are trying to capture the details of a wave.

You need to take a lot of snapshots of your wave, otherwise you are going to loose important details of your wave, ending up with an inaccurate description of your wave.

If you now consider your signal, if you do not have enough information, you may end up having a distorted version of your signal, which is called aliasing.

The Nyquist limit ensures you are taking enough information of your signal so that you do not end up having an aliasing.

The rule is: sample at least twice as fast as the highest frequency in the signal.

Fourier Transform example

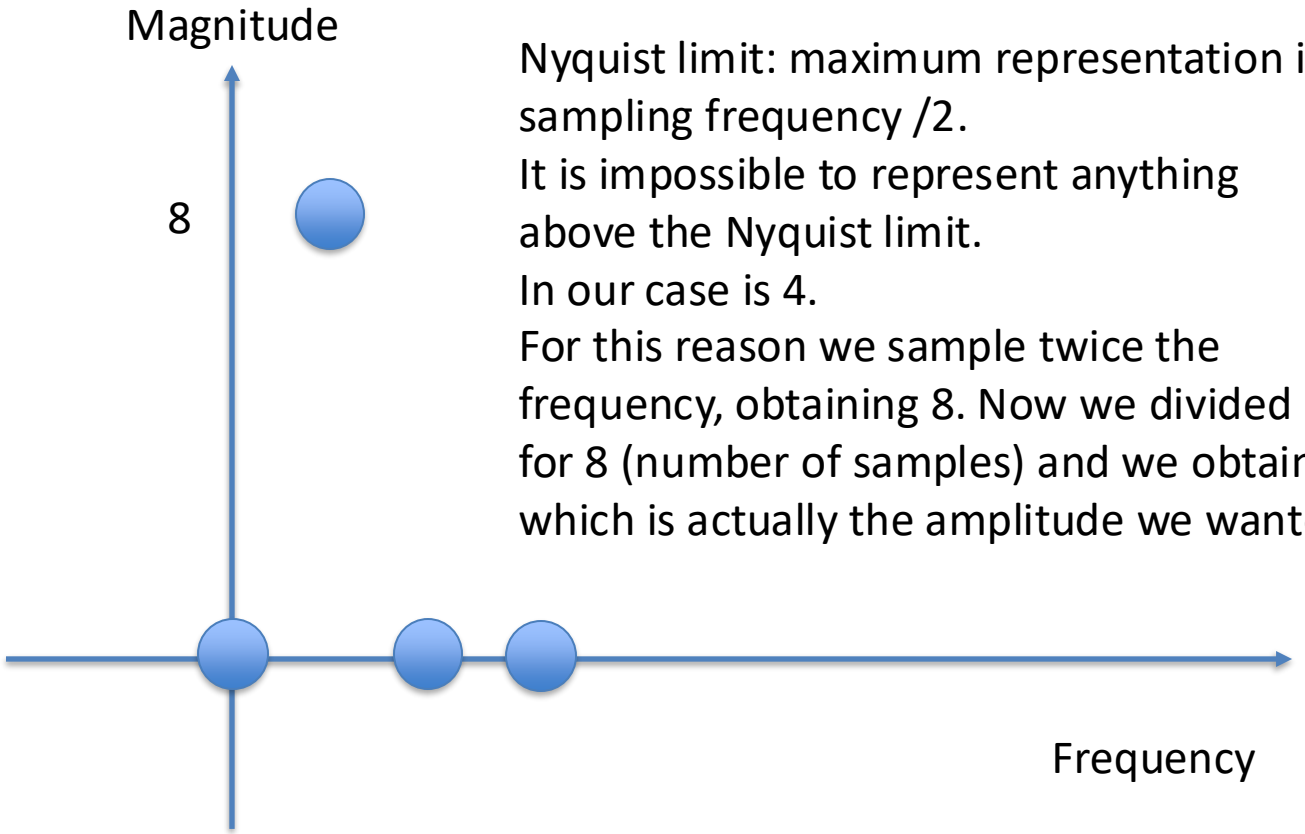
Sine wave of 1Hz
Amplitude of 1
Sampling frequency 8 Hz
8 samples

$X_0 = 0$

$X_1 = 0 - 4j$

$X_2 = 0$

$X_3 = 0$



Nyquist limit: maximum representation is sampling frequency /2.
It is impossible to represent anything above the Nyquist limit.
In our case is 4.
For this reason we sample twice the frequency, obtaining 8. Now we divided it for 8 (number of samples) and we obtain 1, which is actually the amplitude we wanted

Fourier Transform

Properties of Fourier Transform

Property	Spatial Domain	Frequency Domain
Linearity	$\alpha f_1(x) + \beta f_2(x)$	$\alpha F_1(u) + \beta F_2(u)$
Scaling	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{u}{a}\right)$
Shifting	$f(x-a)$	$e^{-i2\pi ua} F(u)$

Fourier Transform and convolution

Convolution of 2 functions f(x) and h(x) is:

$$g(x) = f(x) * h(x) = \int_{-\infty}^{\infty} f(\tau)h(x-\tau)d\tau$$

Fourier transform of g(x):

$$G(u) = \int_{-\infty}^{\infty} g(x)e^{-i2\pi ux}dx$$

$$G(u) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau)\, h(x-\tau)e^{-i2\pi ux}d\tau dx$$

$$G(u) = \int_{-\infty}^{\infty} \underbrace{f(\tau)\, e^{-i2\pi ux}}_{F(u)} d\tau + \int_{-\infty}^{\infty} \underbrace{h(x-\tau)e^{-i2\pi ux}}_{H(u)}\, dx$$

Fourier Transform and convolution

The convolution in the spatial domain corresponds to the multiplication in the frequency domain.

At the same time, if you take the product of 2 functions in the spatial domain, it is equivalent to the convolution of the 2 functions in the frequency domain.

$$g(x) = f(x) * h(x) \rightarrow G(u) = F(u)H(u)$$
$$g(x) = f(x)h(x) \rightarrow G(u) = F(u) * H(u)$$

Convolution using the Fourier Transform

There are very fast algorithms to find the Fourier transform (and its inverse) than the convolution in the spatial domain. Hence, if you have huge images, or huge kernels, it is faster!

You can use this approach to compare by similarity the image in the frequency domain, but also for the filtering we saw in image pre-processing.

$$g(x) = f(x) * h(x)$$

$$\downarrow \qquad \uparrow \qquad \uparrow$$

$$G(u) = F(u)\ x\ H(u)$$

Clustering

Now that we have seen a little bit of distance/similarity metrics, we can exploit them for the clustering.

Clustering: k-means

We can apply k-means also to images, with various settings:

1. Images and Euclidean distance
2. A feature descriptor of the images and Euclidean distance (based on SSIM, SIFT…)

Clustering: hierarchical

Also in this case, we can apply the hierarchical approach looking for image clusters, but again we need to pay attention to the notion of distance and the selection of the image representation.

Clustering: DB-scan

We can use DB-scan also for images.

Which distance metric can we use?

1. Euclidean
2. Cosine

Also in this case, you can compare the images and check their distance, or you can compare the feature descriptors.

Clustering: drawbacks

In theory we can apply all of the techniques we saw so far also to images.

But, these algorithms may be quite slow when working with huge dimensions, like the ones of the images.

For this reason is extremely important to select carefully the feature descriptors for our images:

1. A poor one will not represent well our image
2. A big representation will make the computation impossible
3. Simply using the entire image requires high computational costs, as well as difficulties in handling similarities