

Computer Vision

Assignment 1: Image Processing

Pablo Mesejo and David Criado

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA



Index

- Submission rules
- Brief review of convolutions and filters
- Description and analysis of Assignment 1

Submission Rules

- Only one .ipynb file is submitted, directly **integrating code, results, analysis and discussion**.
- The template/statement is available in PRADO.
- You also have supplementary materials!

<https://pradogrado2425.ugr.es/mod/folder/view.php?id=91458>



Materiales de Apoyo (Python + consejos sobre cómo elaborar un informe académico-científico)



- 1. Introduccion a Python.pdf
- 2. Introducción a Numpy Matplotlib y sklearn.pdf
- 3. Ejercicios y ejemplos de repaso.pdf
- 4. Fundamentos de Python para manipulacion de imagenes.pdf
- Algunas ideas sobre cómo elaborar un informe académico-científico.pdf
- bank-full.csv
- Guia_Python.ipynb
- mat.txt
- YearPredictionMSD.txt

Submission Rules

- Upload only the .ipynb file to PRADO. Do not upload images to PRADO!
- Reading images or any input file:
“/content/drive/MyDrive/images”
- Do not write anything into disk (i.e, do not write anything in Google Drive).
- The structure of the template must be respected.

Submission

- Deadline: 03 November
 - Maximum score: 12 points
 - Submission site: PRADO
-
- **Explanation/discussion/analysis accompanying code and results is essential.**

Goals

- To learn how to implement **convolution filters** and, in particular, the calculation of the derivatives of an image.
- To show how, using linear filtering techniques, it is possible to extract relevant information from an image to allow its **interpretation**.
- This is an assignment very oriented towards **image processing**.

What is a convolution?

- **Local linear operation** with a mask.
 - The coefficients/values of this mask/filter/kernel determine the operation performed.

If f and g are images, a and b scalar, and L a linear operator:

$$L(af + bg) = aLf + bLg$$

What is a convolution?

- Technically, a convolution is a *cross-correlation* where the filter/mask is rotated 180 degrees.
 - Unlike correlation, convolution verifies the **associative property**, which **allows us to build complex filters from simple ones**.
 - If we have an image f , and we want to convolve it with g and then with h . Knowing that $f * g * h = f * (g * h)$, we can convolve g and h , create a single filter, and then convolve it with f .
 - If the kernel is symmetric:
 - Convolution = Cross-correlation
 - If the kernel is antisymmetric (e.g. $[-1 \ 0 \ 1]$), only the sign changes

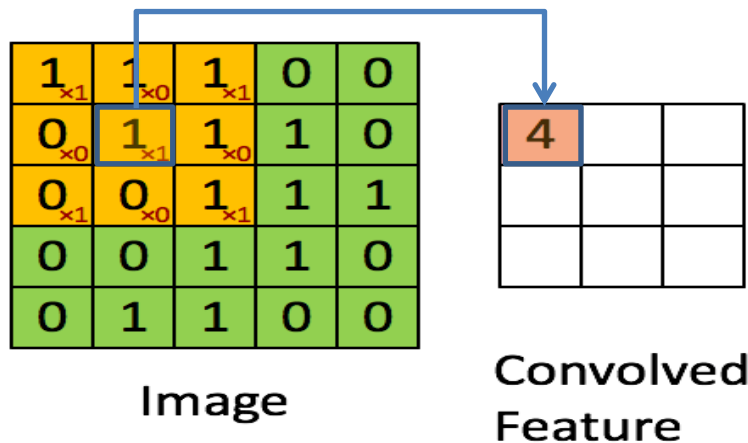
Note: functions like `cv2.filter2D`, actually apply correlation and not convolution.

What is a convolution?

- **Local linear operation** with a mask.

The **red numbers** represent the values/coefficients of the filter/mask.

The filter is multiplied element-by-element with the image, the products are added, and the central position of the filter in the image is substituted.



1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

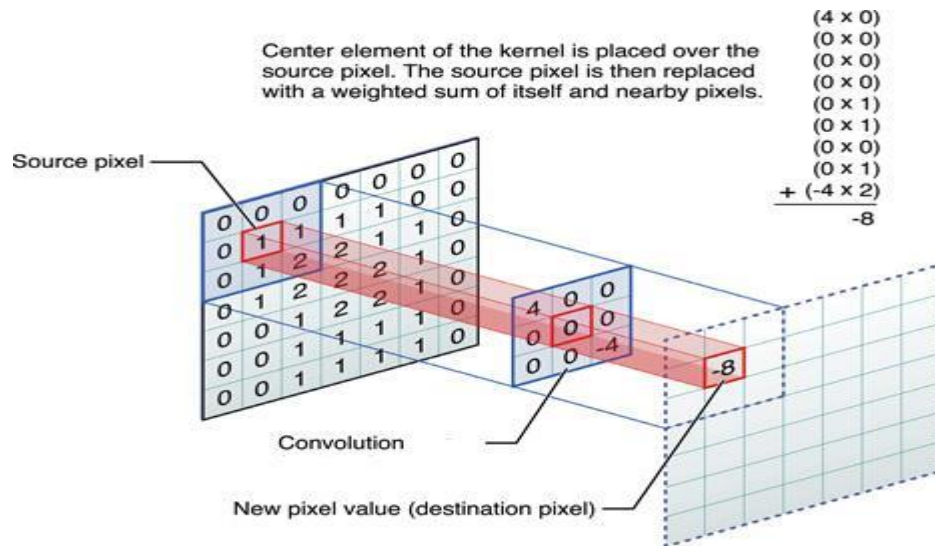
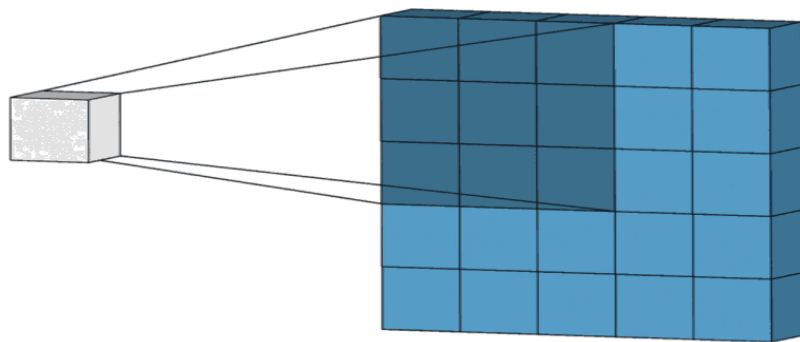
Image

4		

Convolved Feature

What is a convolution?

- In case this makes it clearer...



<https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>

<https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411>

What is a convolution?

- **Local linear operation** with a mask.
 - The values of the mask determine the result (features to be extracted)
- Gaussian filter (removes high frequencies → smooths the image)

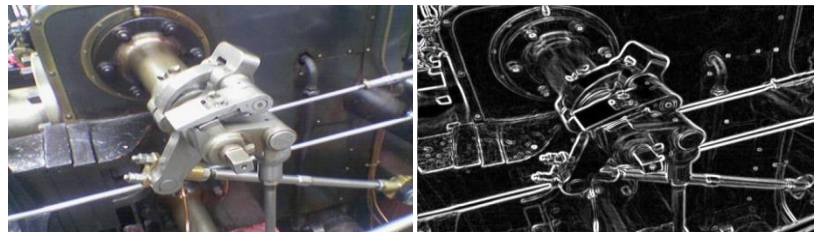
$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

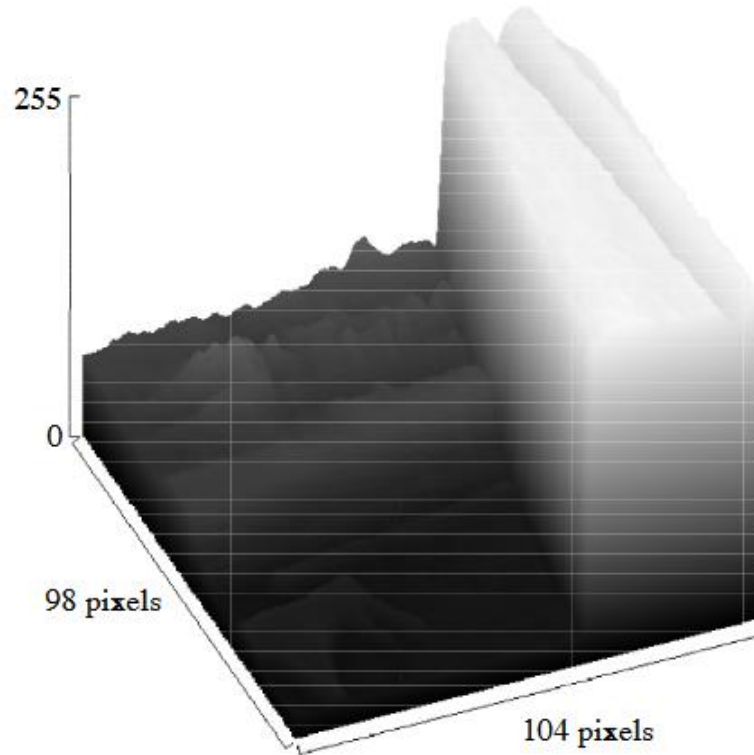


Sobel filter (removes low frequencies → enhances edges)

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



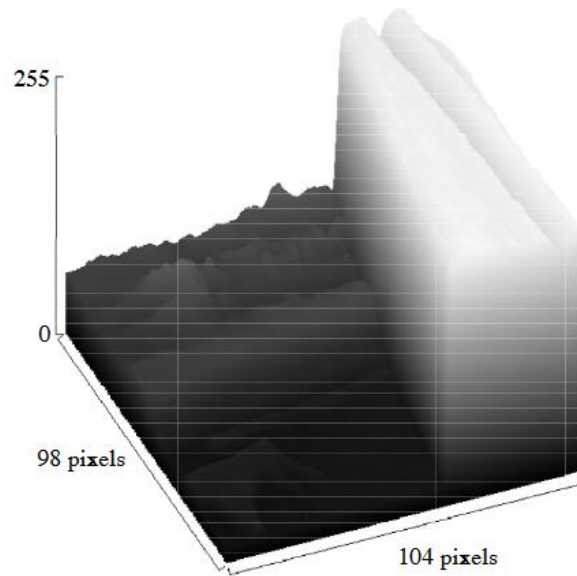
3D visualization of an edge



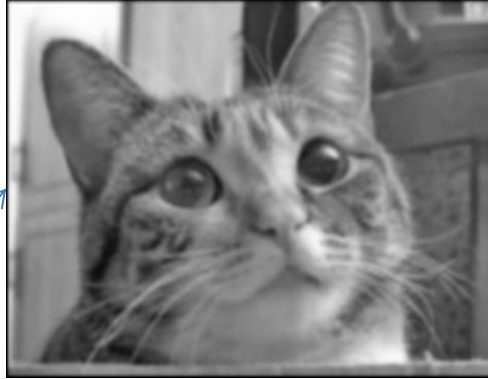
Example extracted from
<https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/pdfs/D02-Images.pdf>

Edges and High Frequencies

- Frequency in 2D images:
 - Rate of change of grey levels with respect to space
 - If it “involves” many pixels to make a change → low frequency
 - If it “involves” few pixels to make a change (i.e., the change is abrupt) → high frequency



In this assignment



Smoothing/blurring
(to remove noise)



Differentiation
(to highlight details)

Two types of kernels

Smoothing

$$\sum g_i = 1$$

(smoothing a constant function should not change it)

Example: $(1/4) * [1 \ 2 \ 1]$

Low pass filter (Gaussian)

In fact, what happens if you do not divide by the sum of the coefficients?



Derivative/differentiation

$$\sum g_i = 0$$

(differentiating a constant function should return 0)

Example: $[-1 \ 0 \ 1]$

High pass filter (derivative of Gaussian)

Advancing ideas...

In ConvNets, these values are learned! They are not defined by a human expert. They are free network parameters and are trained like any other weight.



1989: LeCun et al. used **back-propagation to directly learn the coefficients of convolutional filters** from images of handwritten digits.

1998: LeCun et al. presented **LeNet-5**, a 5-layer ConvNet that could automatically recognise handwritten digits in checks, and proved that **ConvNets outperform all other models in this task**.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 541-551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

Exercise 1

- Key ideas:
 - Initially, you can only use basic OpenCV functions
 - You must know how certain operations are performed at a low level.
 - Unless I explicitly tell you otherwise, you cannot use, for example, `GaussianBlur()` or `pyrDown()`. You have to do it by hand.

Exercise 1

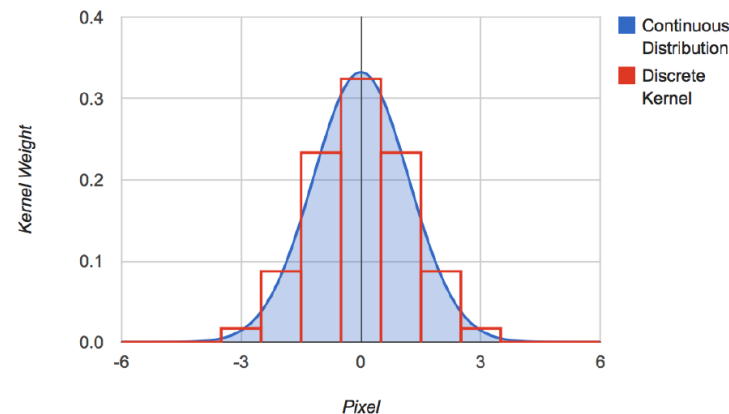
- Key ideas:
 - “implement efficiently”
 - Use only 1D convolutions.
 - I want you to make use of “separability”, according to which a 2D convolution can be reduced to two 1D convolutions.
 - OpenCV function `sepFilter2D()`

Exercise 1.A

- Calculate the discrete 1D masks of the Gaussian function, the derivative of the Gaussian and the second derivative of the Gaussian.

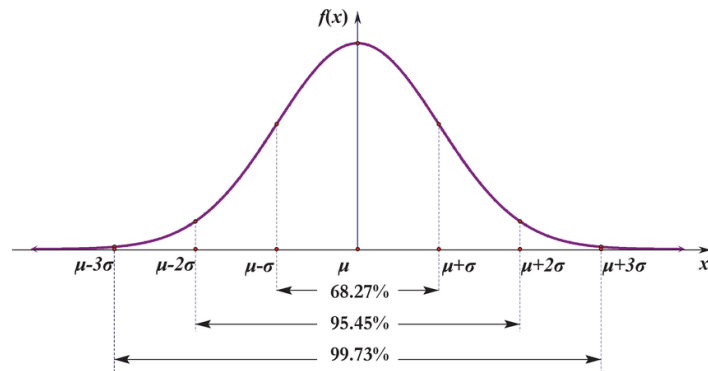
EXERCISE 1.A

We want to discretize a Gaussian.



<http://dev.theomader.com/gaussian-kernel-calculator/>

We know that almost all values are within three standard deviations of the mean → It makes sense to use, by default, 3σ .



EXERCISE 1.A

Gaussian Mask 1D

- $f(x) = c \cdot e^{-\frac{x^2}{2\sigma^2}}$

We ignore this normalizing constant ($1/\sigma\sqrt{2\pi}$)! The shape of the filter is the same! We later normalize the kernel, making the sum of the coefficients equal to 1

- $mask: [f(-k), f(-k+1), \dots, f(0), f(k-1), f(k)], k \text{ an integer}$

- What k to choose ?

- According to the Gaussian properties the k-value that verifies $\min(k) \geq 3\sigma$

- In addition,

$$\sum_{i=-k}^k f(i) = 1$$

for $\sigma = 1$ the mask size is 7 (i.e. $k = 3$).

The masks must add up to 1
→ remember to divide your mask by the sum of its coefficients

EXERCISE 1.A

You must create a function that:

- given a σ , calculates the mask size (T), and provides a 1D Gaussian mask.
- given a mask size (T), automatically adjusts the σ , and returns the 1D Gaussian mask.

$$T \text{ being the size of the mask, } T = 2 \cdot k + 1 \rightarrow k = (T-1)/2 \rightarrow (T-1)/2 = 3 \cdot \sigma \rightarrow 2 \cdot [3 \cdot \sigma] + 1 = T$$

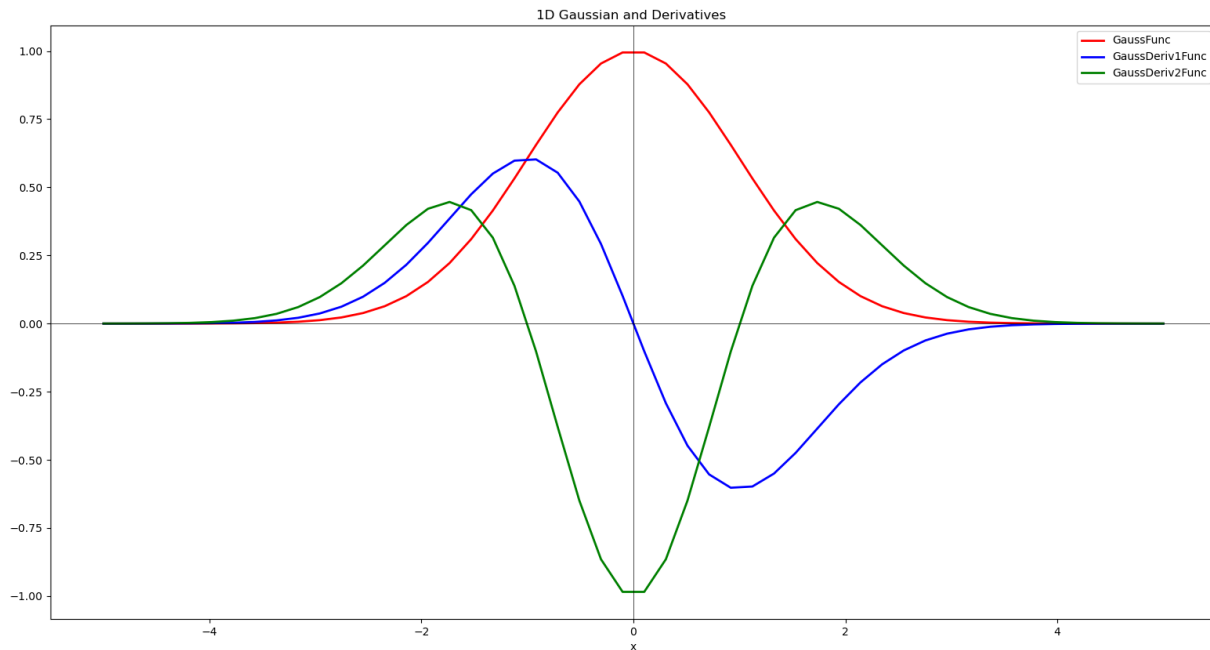
If given, for example, $T=5$:

- we can fill in the mask values by making $[f(-2), f(-1), f(0), f(1), f(2)]$ and substituting in the Gaussian function a sigma of 0.67 (obtained from the previous formula).


Once we have σ and K we can discretize the mask by applying the Gaussian function (or its derivatives)

EXERCISE 1.A

- The whole process described above applies to the derivatives of the Gaussian function



EXERCISE 1.A

- But... so far almost all the filters/kernels/masks we have seen were matrices (2D)... are not these vectors? 
 - Yes, but remember that we make use of the **separability of filters**: first, we convolve in one direction (1D) and then in the other (1D). We will come back to this later...

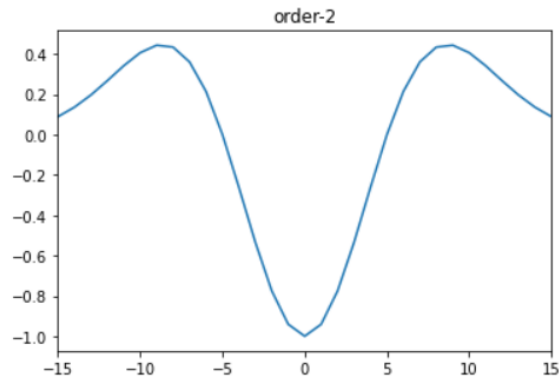
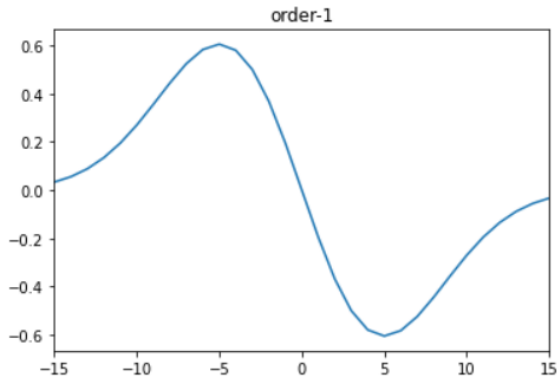
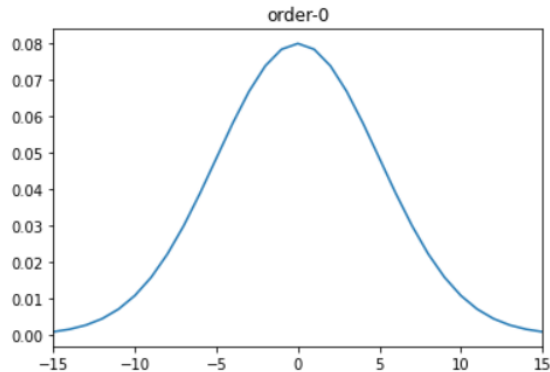
EXERCISE 1.A

- In the case of derivatives:
 - The only thing that changes is the function used to obtain the mask values.
 - You must calculate the derivatives and apply the resulting functions.
 - Remember that **the values of a derivative mask add up to approximately zero** (they are the same values repeated, but with a different sign).
 - **In the case of derivative masks, you should not divide them by the sum of the coefficients**
 - that would be 0, or a very small number, so that the kernel values would be huge.

EXERCISE 1.A

- Represent the profile (i.e. the silhouette of the mask as 1D functions) to verify that the masks created are correct

Example of visual comparison of 1D masks



If you add the coefficients of the derivative kernels, which addition do you think will be closer to zero: the 1st derivative kernel or the 2nd derivative kernel?



How to present this exercise?

- The following tips apply to all exercises/sections.
 - And they are the logical consequence of what is indicated in https://pradogrado2425.ugr.es/pluginfile.php/159983/mod_folder/content/0/Algunas%20ideas%20sobre%20cómo%20elaborar%20un%20informe%20académico-científico.pdf
- Kernels should be visible and easy to analyze.
 - Adequate size to perceive nuances.
 - There is little point in putting them all in a single column, where the reader has to continuously scroll up/down (it makes it difficult to compare them).
 - If they are displayed in a grid of 3 columns and 7 rows it's easier to observe them together.
 - Figures with kernels must have a title to be easily identified.

How to present this exercise?

- The text accompanying the exercise must demonstrate your knowledge.
 - When only one or two short sentences are included as a comment (even if they are accurate) → impression of little effort.
- But... I don't know what to explain... 😞
 - Use your common sense! In Exercise 1.A, what does it seem logical to explain/show?
 - The analytical expressions of the calculated derivatives.
 - The relationship between σ and the mask size.
 - The importance of scale normalization: what is it and what is it used for.
 - Why derivative masks add up to zero and smoothing masks add up to one.
 - Analyze shape of kernels and why they have that particular shape.
 - ...

EXERCISE 1.B

- Perform 2D convolutions using the 1D masks created in the previous section
- You must use *cv2.sepFilter2D()* to apply the separable filters on the image.

EXERCISE 1.B

- The idea is to create a function that, **given an image, a sigma, and a list with two elements (representing the order of derivative applied by rows and columns, respectively)** provides a new image as result of the convolution operation.

```
my2DConv(im, sigma, orders)
```

EXERCISE 1.B

◆ sepFilter2D()

```
void cv::sepFilter2D ( InputArray  src,
                      OutputArray dst,
                      int         ddepth,
                      InputArray  kernelX,
                      InputArray  kernelY,
                      Point       anchor = Point(-1,-1) ,
                      double      delta = 0 ,
                      int         borderType = BORDER_DEFAULT
                    )
```

Python:

```
cv.sepFilter2D( src, ddepth, kernelX, kernelY[, dst[, anchor[, delta[, borderType]]]] ) -> dst
```

```
#include <opencv2/imgproc.hpp>
```

Applies a separable linear filter to an image.

The function applies a separable linear filter to the image. That is, first, every row of `src` is filtered with the 1D kernel `kernelX`. Then, every column of the result is filtered with the 1D kernel `kernelY`. The final result shifted by `delta` is stored in `dst`.

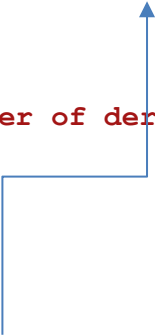
Parameters

src	Source image.
dst	Destination image of the same size and the same number of channels as <code>src</code> .
ddepth	Destination image depth, see combinations
kernelX	Coefficients for filtering each row.
kernelY	Coefficients for filtering each column.
anchor	Anchor position within the kernel. The default value <code>(-1, -1)</code> means that the anchor is at the kernel center.
delta	Value added to the filtered results before storing them.
borderType	Pixel extrapolation method, see BorderTypes . <code>BORDER_WRAP</code> is not supported.

See also

[filter2D](#), [Sobel](#), [GaussianBlur](#), [boxFilter](#), [blur](#)

```
if orders==[0,0]:
    return cv2.sepFilter2D(im,-1,maskG,maskG)
elif
    ...
else:
    print('error in order of derivative')
```



Note

when `ddepth=-1`, the output image will have the same depth as the source.

This refers to the type used. If you set `-1` it will use the type of the input image. If `uint8`, it will saturate to zero and no negative values will appear. To solve this, you can use `cv2.CV_64F`

https://docs.opencv.org/4.6.0/d4/d86/group_imgproc_filter.html#ga910e29ff7d7b105057d1625a4bf6318d

EXERCISE 1.B

The **1st derivative of a 2D Gaussian** is separable:

$$\begin{aligned} G(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(\frac{-(x^2 + y^2)}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= G_h(x) \cdot G_v(y) \\ \frac{\partial G(x, y)}{\partial x} &= -\frac{x}{\sigma^2} \cdot \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= G'_h(x) \cdot G_v(y) \end{aligned}$$

**Derivative of horizontal 1D
Gaussian kernel (row-
wise)**

**1D vertical
Gaussian kernel
(column-wise)**

*Smooths in one
direction, differs in
the other*

This allows us to know **which 1D kernels to apply, and in which order, to calculate the derivatives of an image**. In the example above, the first derivative in X.

EXERCISE 1.B

- These 1D kernels are returned by `cv2.getDerivKernels()`
 - Check documentation:
https://docs.opencv.org/master/d4/d86/group_imgproc_filter.html#ga6d6c23f7bd3f5836c31cfae994fc4aea
 - `cv.getDerivKernels(dx, dy, ksize)`

Order of derivative with respect to X

Order of derivative with respect to Y

Kernel size

```
sobel3x3 = cv.getDerivKernels(1,0,3)
(array([[ -1.],
        [  0.],
        [  1.]], dtype=float32),
 array([[1.],
        [2.],
        [1.]], dtype=float32))
```

```
np.outer(sobel3x3[0],sobel3x3[1])
array([[ -1.,  -2.,  -1.],
       [  0.,   0.,   0.],
       [  1.,   2.,   1.]], dtype=float32)
```

G_y

```
np.outer(sobel3x3[1],sobel3x3[0])
array([[ -1.,   0.,   1.],
       [-2.,   0.,   2.],
       [-1.,   0.,   1.]], dtype=float32)
```

G_x

EXERCISE 1.B

And the **2nd derivatives of Gaussian** as well:

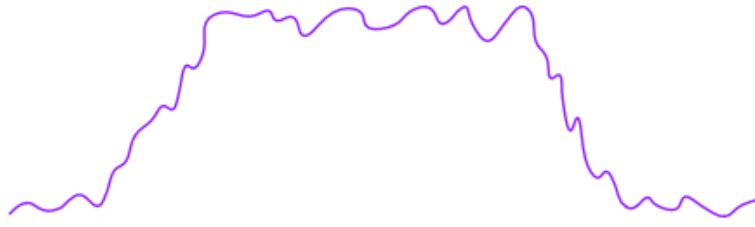
$$\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = \overset{\text{2nd derivative in X}}{\boxed{G_h''(x) \cdot G_v(y)}} + \overset{\text{2nd derivative in Y}}{\boxed{G_h(x) \cdot G_v''(y)}}$$

convolution in one direction with the 2nd derivative of Gaussian and then, in the other direction, convolution with Gaussian.

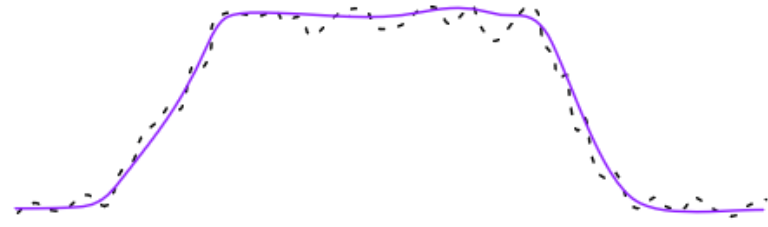
convolution in one direction with Gaussian and then, in the other direction, convolution with the 2nd derivative of Gaussian.

Note: what you see in this slide ($\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$) is, in fact, the **Laplacian of Gaussian**

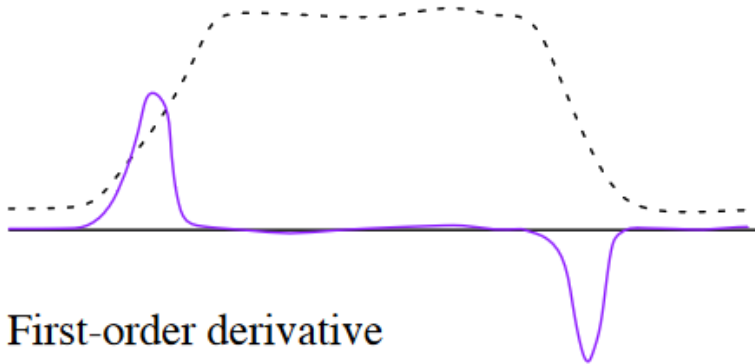
Derivatives and edges



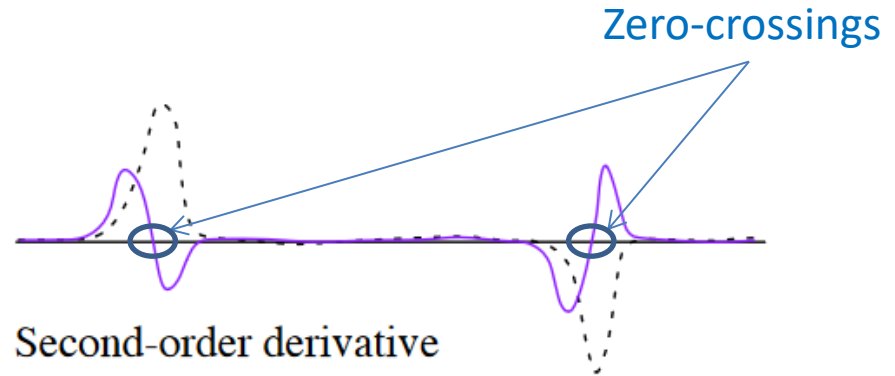
Intensity profile of an input image



After noise removal



First-order derivative



Second-order derivative

Zero-crossings

The advantage of second derivatives? They make it possible to locate an edge more precisely!

EXERCISE 1.B

- Important notes:
 - Everything is done in greyscale:
cv.imread(filename,0)
 - Working in RGB colour is just doing in triplicate what is done in a single band.
 - Operate in float to avoid precision problems and truncation of integer values (see next slide)

Technical note: Float vs Int

```
filename = 'data/motorcycle.bmp'  
im = read_image(filename, 'gray', 0).astype(np.float64)  
GaussianKernel = kernelGauss1D(1, None, 0)  
image = insert_padding(im, GaussianKernel, 2)
```

```
imageConvolved = convolve2D(image, GaussianKernel,  
                             GaussianKernel)  
imGaussianBlur =  
    cv.GaussianBlur(image, (len(GaussianKernel), len(GaussianKernel)), 1, 1)
```

```
compare_images(imageConvolved, imGaussianBlur)
```

Number of different pixels in both images (with a difference of more than 1 grey level): **1.081%** (1518 de 140454)

```
filename = 'data/motorcycle.bmp'  
im = read_image(filename, 'gray', 0)  
GaussianKernel = kernelGauss1D(1, None, 0)  
image = insert_padding(im, GaussianKernel, 2)
```

```
imageConvolved = convolve2D(image, GaussianKernel,  
                             GaussianKernel)  
imGaussianBlur =  
    cv.GaussianBlur(image, (len(GaussianKernel), len(GaussianKernel)), 1, 1)
```

```
compare_images(imageConvolved, imGaussianBlur)
```

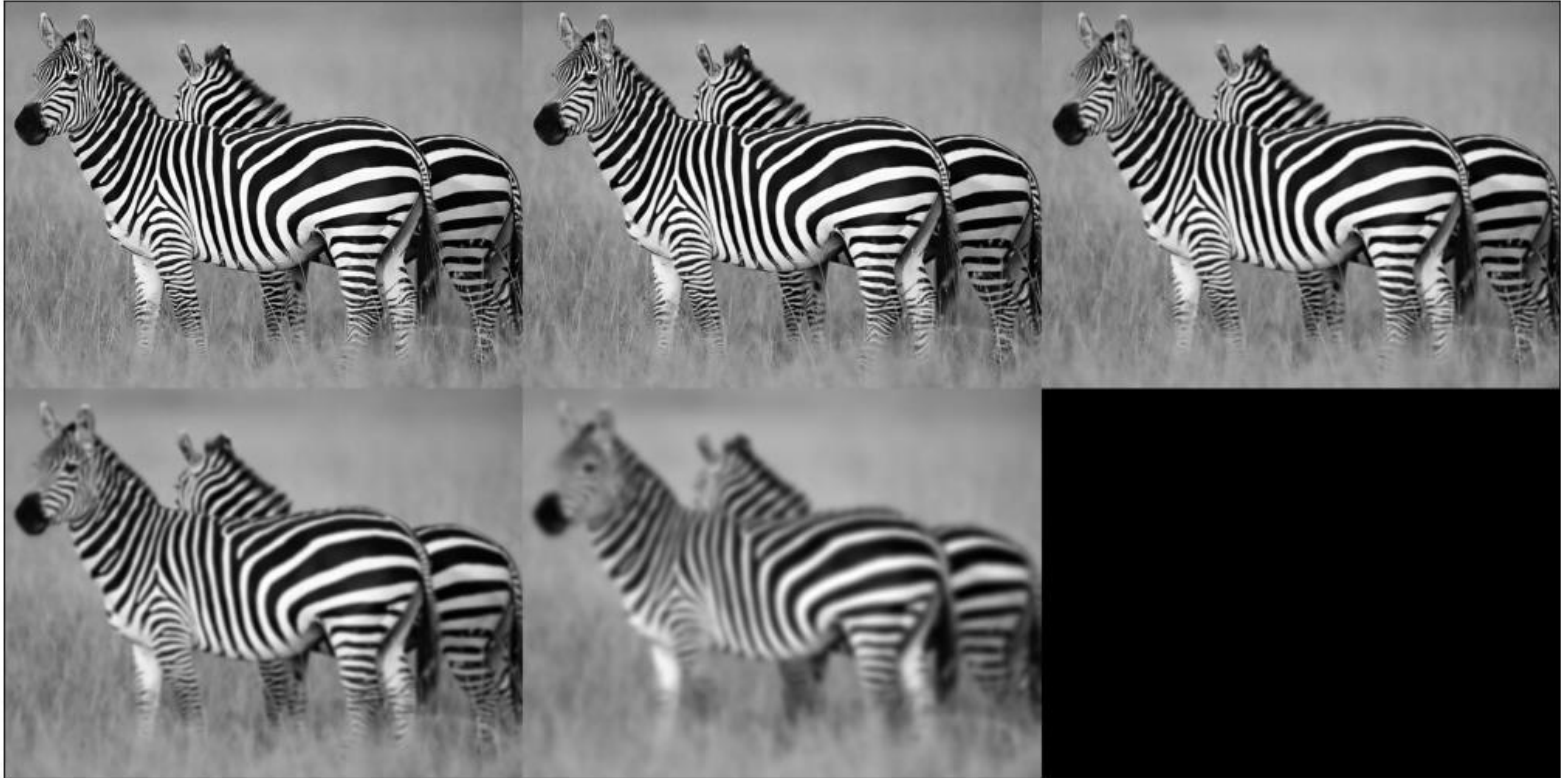
Number of different pixels in both images (with a difference of more than 1 grey level): **8.117%** (11400 de 140454)

EXERCISE 1.B

- In our case, everything is Gaussian and, therefore, directly decomposable.
 - In all the assignment, it is not necessary to calculate the decomposition into singular values. See <https://bartwronski.com/2020/02/03/separate-your-filters-svd-and-low-rank-approximation-of-image-filters/>

EXERCISE 1.B

Zero derivative



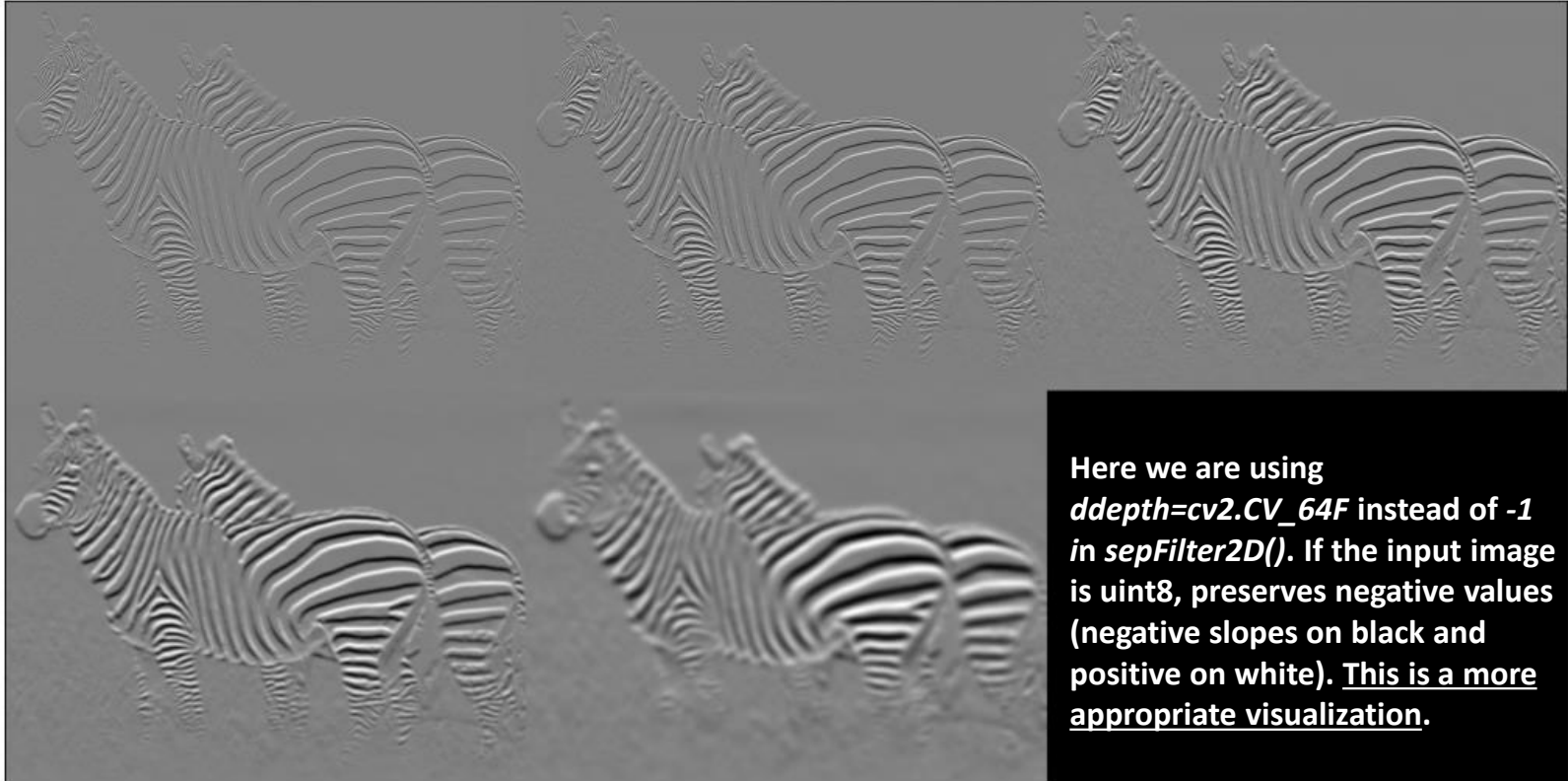
EXERCISE 1.B

First derivative [0,1]



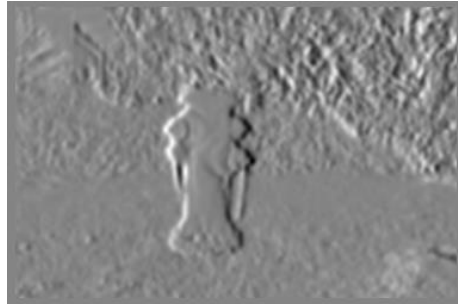
EXERCISE 1.B: about visualization

First derivative [0,1]

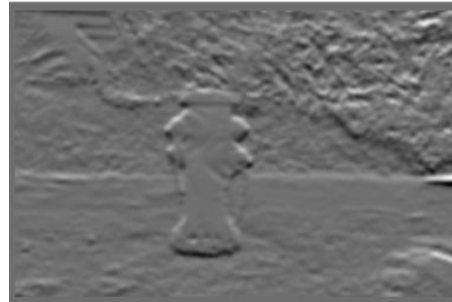


EXERCISE 1.C

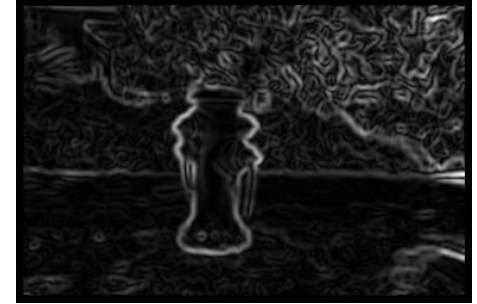
- Using the masks of exercise 1.A, calculate the gradient and Laplacian of a given image. We start with the gradient:



$$g_x = I * G_y * G'_x$$



$$g_y = I * G_x * G'_y$$



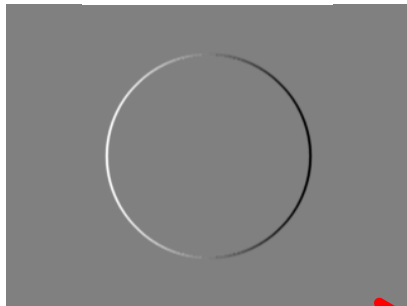
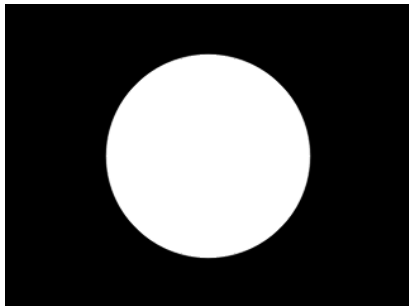
$$|\nabla I| = \sqrt{g_x^2 + g_y^2}$$

Extracted from "Image
Filtering and Edge
Detection" de Stan
Birchfield, Clemson
University

Derivatives and Edges

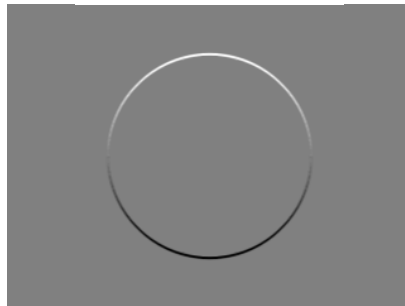
Horizontal derivative

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$



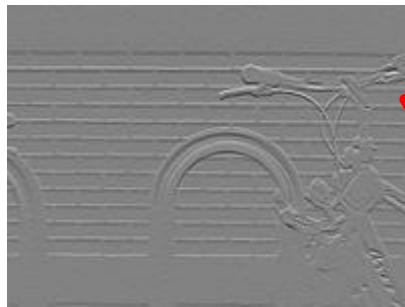
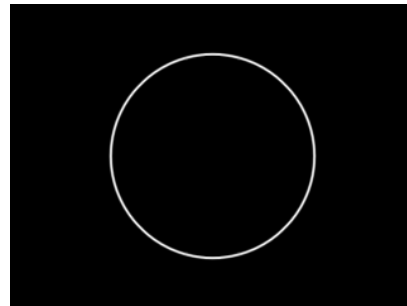
Vertical derivative

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$



Gradient Magnitude

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



Visual examples taken from <https://stackoverflow.com/questions/19815732/what-is-the-gradient-orientation-and-gradient-magnitude> and https://en.wikipedia.org/wiki/Sobel_operator

Derivatives and Edges

Horizontal derivative

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

Vertical derivative

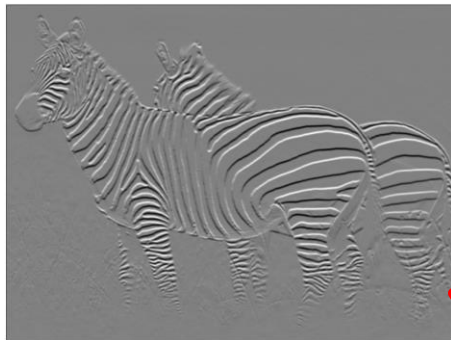
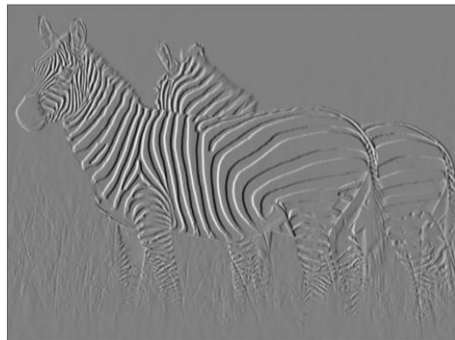
$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Gradient magnitude

$$G = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

Gradient orientation

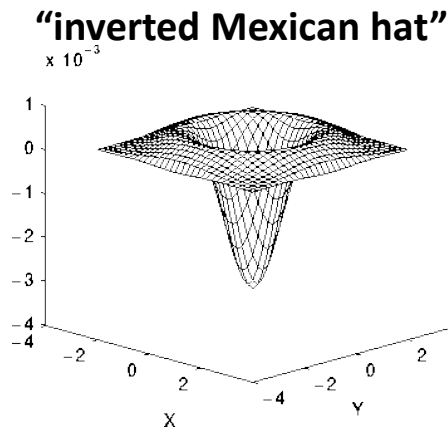
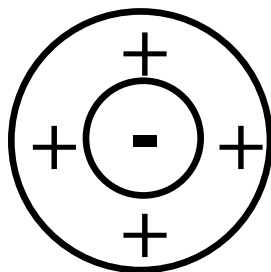
$$\Theta = \text{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$



EXERCISE 1.C

- Example of a 2D 3x3 Laplacian kernel:

0	1	0
1	-4	1
0	1	0



- Is it separable?
 - First, what is necessary for a kernel to be separable?

EXERCISE 1.C

- What is necessary for a kernel to be separable?

A 2D kernel is *separable* if and only if all its rows/columns are linearly dependent

- This filter

0	1	0
1	-4	1
0	1	0

, therefore, is NOT:

$$\alpha \cdot [0, 1, 0] + \beta \cdot [1, -4, 1] = [0, 0, 0] \iff \alpha = 0 \wedge \beta = 0$$

→ They are linearly independent vectors!!

EXERCISE 1.C

```
G = np.array([[0,1,0],[1,-4,1],[0,1,0]])  
array([[ 0,  1,  0],  
       [ 1, -4,  1],  
       [ 0,  1,  0]])
```

If the rank is not 1 → it is not directly separable!
Remember that **the rank of a matrix is the maximum number of linearly independent columns/rows.**

```
np.linalg.matrix_rank(G)  
2
```

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

if $\alpha = 2$ and $\beta = -1 \rightarrow \alpha \cdot [1, 2, 1] + \beta \cdot [2, 4, 2] = [0, 0, 0]$

```
G = np.outer([1,2,1],[1,2,1])  
np.linalg.matrix_rank(G)  
1
```

$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

if $\alpha = -2$ and $\beta = 1 \rightarrow \alpha \cdot [-1, 0, 1] + \beta \cdot [-2, 0, 2] = [0, 0, 0]$

```
G = np.outer([1,2,1],[-1,0,1])  
np.linalg.matrix_rank(G)  
1
```

```
np.linalg.matrix_rank(np.outer(gaussianMask1D(1, None, 0), gaussianMask1D(1, None, 2)))  
1
```

**In the case of the LoG, we have two matrices with rank=1
→ We have two separable 2D filters!!!!**

EXERCISE 1.C

- We can calculate LoG as the sum of 2D convolutions that are separable
 - Because the Gaussian and its derivatives are!!!
 - we need 4 1D convolutions (and that is still less computationally expensive than doing a single 2D convolution; unless the kernel is very small)

if the kernel is 7×7 we need 49 multiplications and additions per pixel for the 2D kernel, or $4 \cdot 7 = 28$ multiplications and additions per pixel for the four 1D kernels; this difference grows as the kernel gets larger

Interesting references on the LoG: <https://stackoverflow.com/questions/53544983/how-is-laplacian-filter-calculated/53545480#53545480> and <https://www.crisluengo.net/archives/1099/>

EXERCISE 1.C

The **Laplacian operator** is the divergence of the gradient, and is given by the sum of the second order derivatives (i.e. the trace of the Hessian matrix):

$$\boxed{\nabla^2} I = \nabla \cdot \nabla I = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix}^T = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Laplacian of the **Gaussian**

Associative property

Distributive property

$$\underbrace{\frac{\partial^2 (I * G)}{\partial x^2} + \frac{\partial^2 (I * G)}{\partial y^2}}_{\text{Laplacian of an image convolved/smoothed with a Gaussian kernel}} \overset{\text{Associative property}}{=} \underbrace{I * \left(\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right)}_{\text{Image convolved with the Laplacian of a Gaussian}} \overset{\text{Distributive property}}{=} I * \frac{\partial^2 G}{\partial x^2} + I * \frac{\partial^2 G}{\partial y^2}$$

EXERCISE 1.C

How do we calculate $I * \frac{\partial^2 G}{\partial x^2} + I * \frac{\partial^2 G}{\partial y^2}$?

We know that the **1st derivative of a 2D Gaussian** is separable:


$$\begin{aligned} G(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= G_h(x) \cdot G_v(y) \\ \frac{\partial G(x, y)}{\partial x} &= -\frac{x}{\sigma^2} \cdot \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= G'_h(x) \cdot G_v(y) \end{aligned}$$

**1D horizontal Gaussian
kernel derivative**

**1D vertical
Gaussian kernel**

*Smooth in one
direction, differentiate
in the other*

EXERCISE 1.C

$$\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = \boxed{G_h''(x) \cdot G_v(y)} + \boxed{G_h(x) \cdot G_v''(y)}$$


convolution in one direction with the 2nd derivative of Gaussian and, then, in the other, convolution with Gaussian.

convolution in one direction with Gaussian and, then, in the other, convolution with the 2nd derivative of Gaussian.

EXERCISE 1.C

- Laplacian of Gaussian

$$L = \sigma^2 (G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma)) \quad \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = G''_h(x) \cdot G_v(y) + G_h(x) \cdot G''_v(y)$$

(Laplacian)

1) dxx: the convolution by rows with the second derivative of the Gaussian and, then, by columns, the convolution with the Gaussian.

2) dyy: the convolution by rows with the Gaussian and, then, by columns, the convolution with the second derivative of the Gaussian.

3) We add dxx and dyy

4) We multiply the result of the sum by σ^2

Actually, you have to scale/normalize the masks obtained in 1.A, **multiplying them by σ (1st derivative) or σ^2 (2nd derivative)**

Useful reference: <https://www.crisluengo.net/archives/1099/>

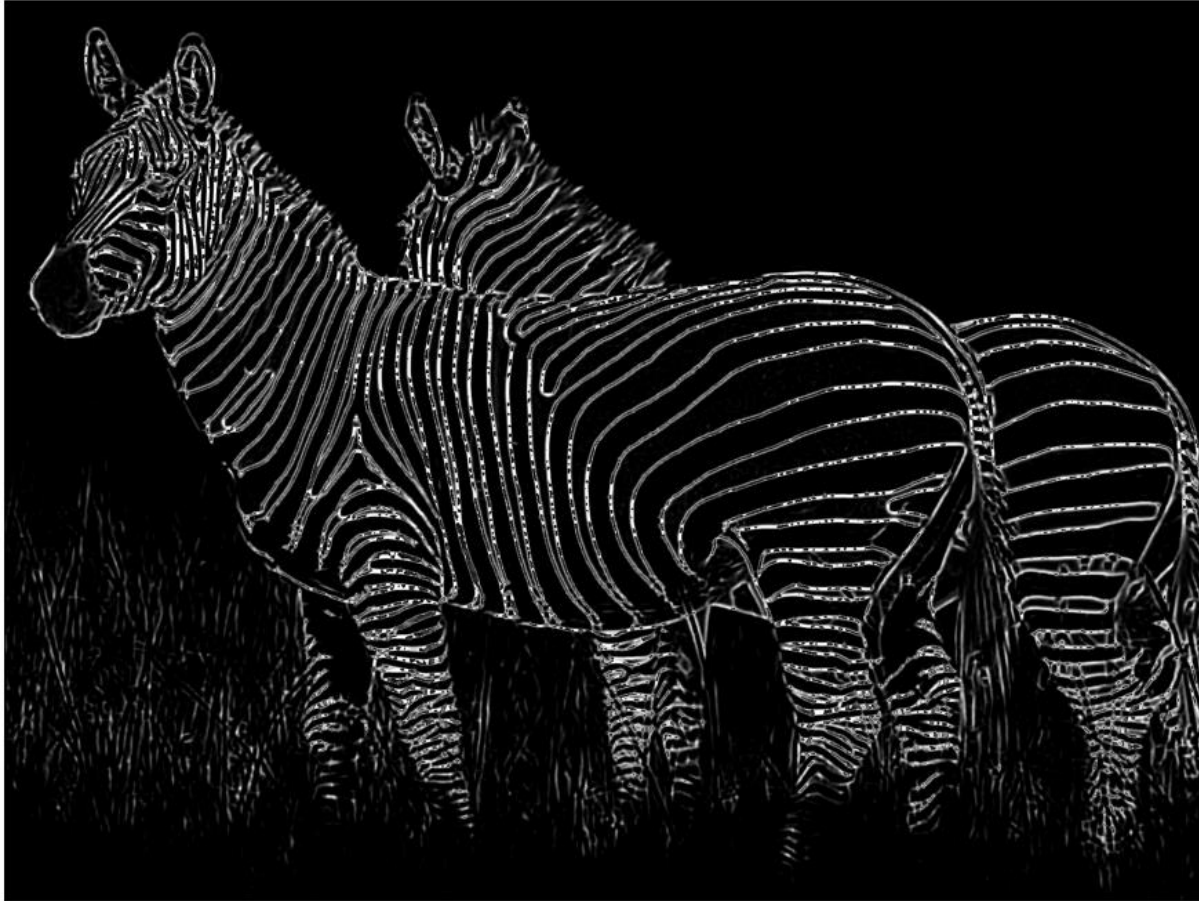
EXERCISE 1.C

Gradient
magnitude



EXERCISE 1.C

Laplacian of
Gaussian



EXERCISE 1.C

Laplacian of
Gaussian
(using
`ddepth=cv2.CV_64F`;
much better
representation!!)



EXERCISE 1.D

- You have to implement your own separable convolution function (`def convolve1D(image, kernel)`) and check the differences/similarities of the results obtained with `cv2.GaussianBlur()`

EXERCISE 1.D

CONVOLVE2D(f, g)

Input: 2D image $f_{\{width \times height\}}$, 2D kernel $g_{\{w \times h\}}$

Output: the 2D convolution of f and g

```
1   $\tilde{h} \leftarrow \lfloor (h - 1) / 2 \rfloor$ 
2   $\tilde{w} \leftarrow \lfloor (w - 1) / 2 \rfloor$ 
3  for  $y \leftarrow 0$  to  $height - 1$  do
4      for  $x \leftarrow 0$  to  $width - 1$  do
5           $val \leftarrow 0$ 
6          for  $j \leftarrow 0$  to  $h - 1$  do
7              for  $i \leftarrow 0$  to  $w - 1$  do
8                   $val \leftarrow val + g(i, j) * f(x + \tilde{w} - i, y + \tilde{h} - j)$ 
9               $h(x, y) \leftarrow val$ 
10 return  $h$ 
```

- Convolution cannot be done “in place”
- **Output and input image must be completely separated**
 - The new values calculated by sliding the window/kernel cannot be used in subsequent calculations.
- Otherwise the calculations are wrong

EXERCISE 1.D

CONVOLVESEPARABLE(I, g_h, g_v)

Input: 2D image $I_{\{width \times height\}}$, 1D kernels g_h and g_v of length w

Output: the 2D convolution of I and $g_v \circledast g_h$

```
1  ▷ convolve horizontal
2  for  $y \leftarrow 0$  to  $height - 1$  do
3      for  $x \leftarrow \tilde{w}$  to  $width - 1 - \tilde{w}$  do
4           $val \leftarrow 0$ 
5          for  $i \leftarrow 0$  to  $w - 1$  do
6               $val \leftarrow val + g_h[i] * I(x + \tilde{w} - i, y)$ 
7               $tmp(x, y) \leftarrow val$ 
8  ▷ convolve vertical
9  for  $y \leftarrow \tilde{w}$  to  $height - 1 - \tilde{w}$  do
10     for  $x \leftarrow 0$  to  $width - 1$  do
11          $val \leftarrow 0$ 
12         for  $i \leftarrow 0$  to  $w - 1$  do
13              $val \leftarrow val + g_v[i] * tmp(x, y + \tilde{w} - i)$ 
14          $out(x, y) \leftarrow val$ 
15  return  $out$ 
```

Extracted from "Image
Filtering and Edge
Detection" by Stan
Birchfield, Clemson
University

Interesting reference:

http://www.songho.ca/dsp/convolution/convolution2d_separable.html

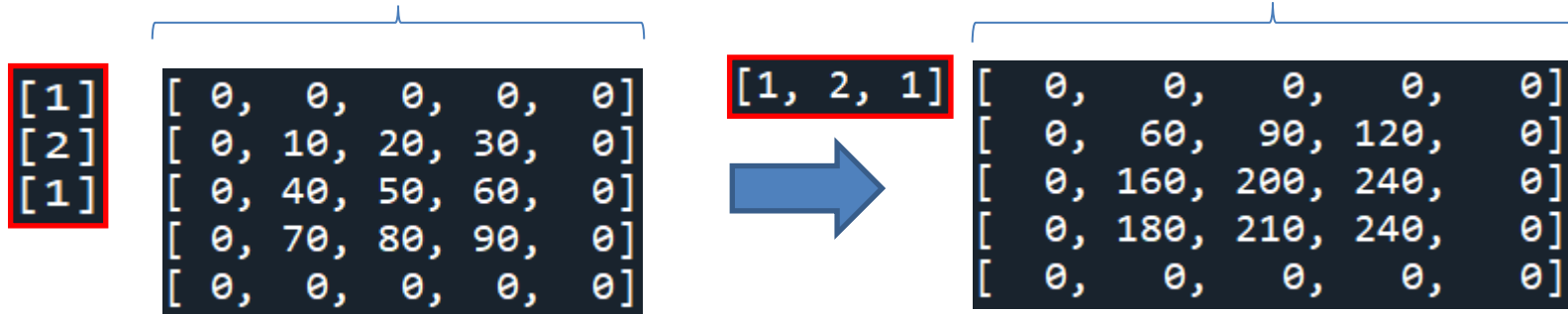
It helps to better understand 2D
convolution from 1D kernels.

In essence, it consists of **going through the whole image, pixel by pixel, first doing the convolution by rows and then, on the result, applying the convolution of the 1D kernel by columns.**

EXERCISE 1.D

We combine the information from different rows
(kernel Y - Coefficients for filtering each column)

We combine the information from different columns
(kernel X - Coefficients for filtering each row)



Input image

[0, 0, 0, 0, 0]
[0, 10, 20, 30, 0]
[0, 40, 50, 60, 0]
[0, 70, 80, 90, 0]
[0, 0, 0, 0, 0]

Kernel

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Separable Kernel

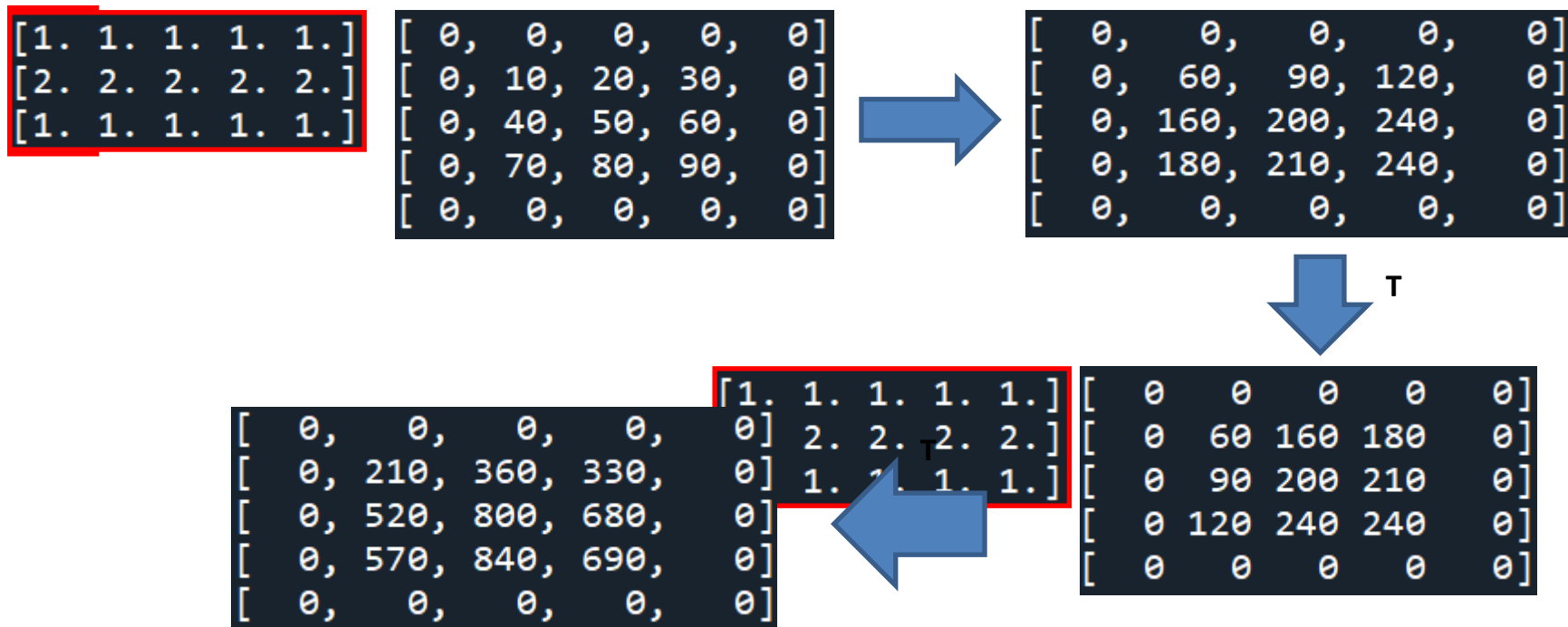
http://www.songho.ca/dsp/convolution/convolution2d_separable.html

[0, 0, 0, 0, 0]
[0, 210, 360, 330, 0]
[0, 520, 800, 680, 0]
[0, 570, 840, 690, 0]
[0, 0, 0, 0, 0]

EXERCISE 1.D

It is not necessary to always go through all the rows and columns (4 for-loops)!!!

For each 1D convolution, we only need to traverse the rows once!!!



EXERCISE 1.D

- **You have to compare your results with *cv.GaussianBlur***
 - If you pass to this function a kernel size and $\sigma=-1$, the function itself estimates the appropriate σ .
 - If the kernel size is 0 \rightarrow it estimates it from σ
- Important note:
 - **OpenCV looks for efficiency (and not precision).**
 - If you implement convolution by hand:
 - don't be surprised if your convolution result is not exactly the same as the one provided by *cv2.GaussianBlur*.
 - your code will probably be “more correct”, but slower.

EXERCISE 1.E

- Critical use of generative AI tools (ChatGPT):
 - "What is scale normalization and what is it used for? Why, in this scale normalization, we multiply our masks by σ^n , where n is the order of the derivative?"

EXERCISE 1

Masks discretization +
Separable convolutions

Gaussian smoothing

Edge detection/enhancement by
means of Gaussian derivatives, and
Laplacian of Gaussian

EXERCISE 2

- 2.A Generate a 4-level Gaussian pyramid representation
- 2.B Generate a 4-level Laplacian pyramid representation
- 2.C What if we don't smooth before subsampling? What is happening to the Laplacian pyramid?
- 2.D Use Laplacian pyramid to recover the original image

EXERCISE 2

- Importance of **image pyramids**:
 - We are used to work with fixed size images.
 - But many times we need **to work with an image at different resolutions**.
 - For example, if we are looking for something specific, such as a face, and we do not know *a priori* the size of the object we are looking for.
 - Or if we need to access an image with different levels of blurring/smoothed (*blur*).
 - Or if we need to compress an image.

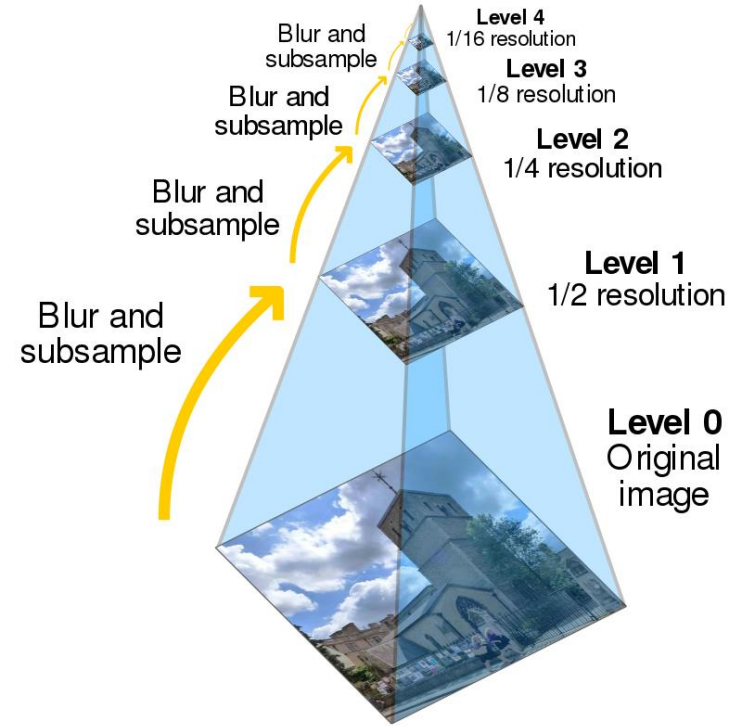
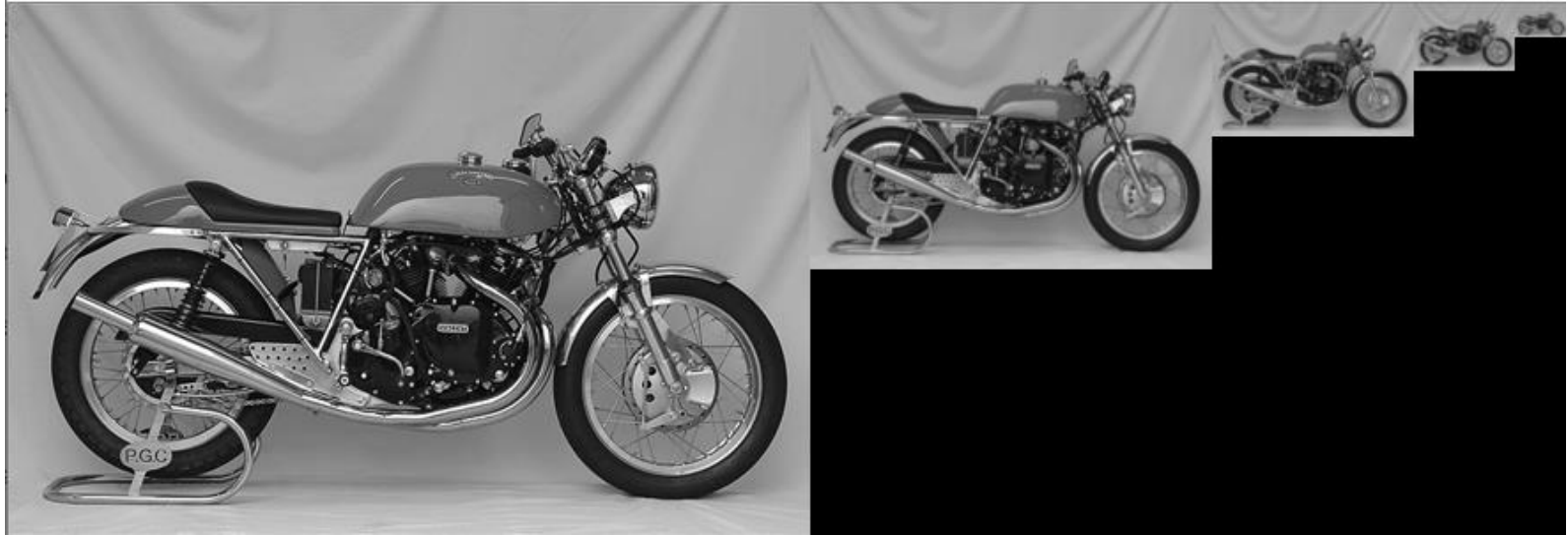


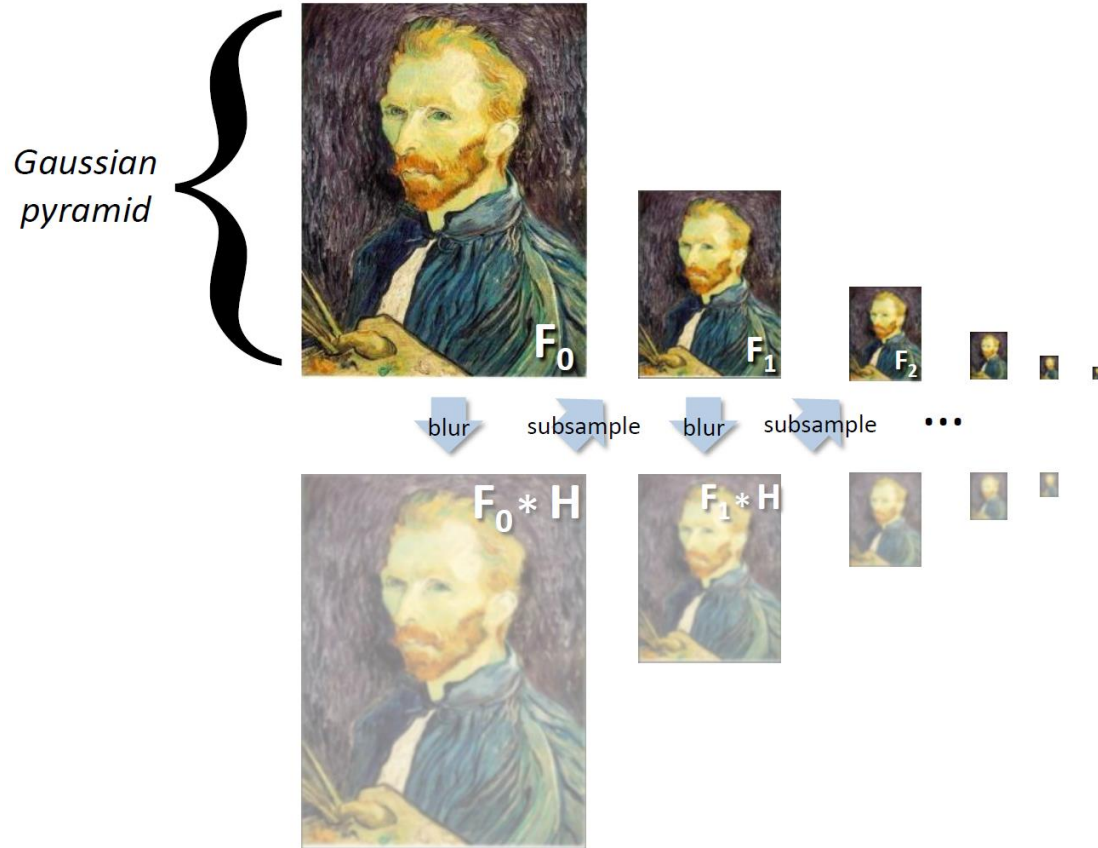
Image extracted from [Wikimedia](#)

EXERCISE 2.A

- Example of a 4-level Gaussian pyramid:



EXERCISE 2.A

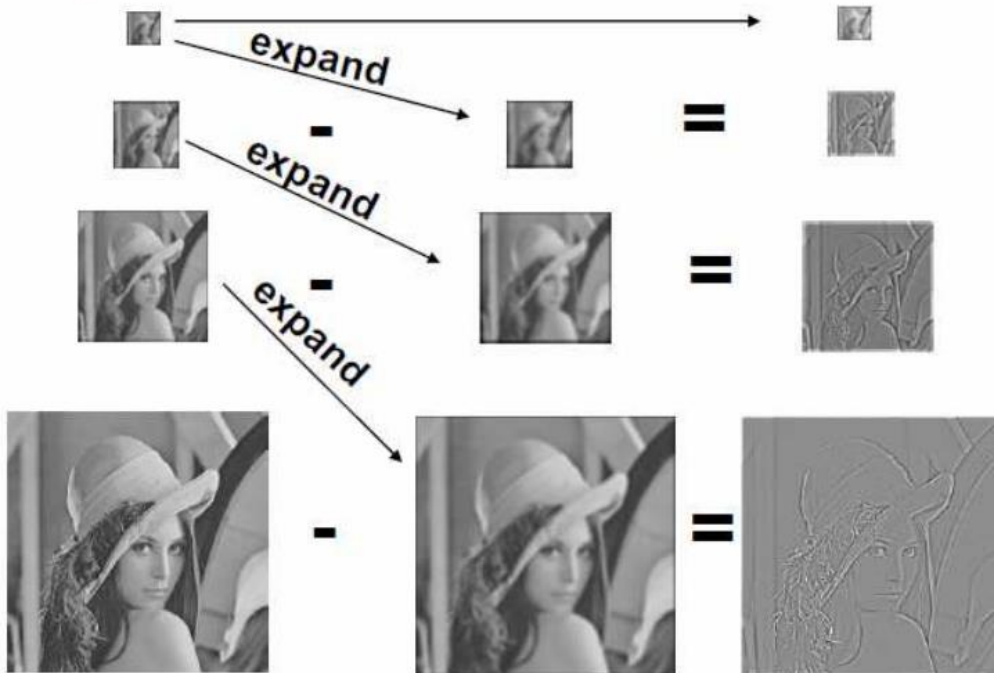


EXERCISE 2.B

Gaussian Pyramid

<http://www.eng.tau.ac.il/~ip/apps/Slides/lecture05.pdf>

Laplacian Pyramid

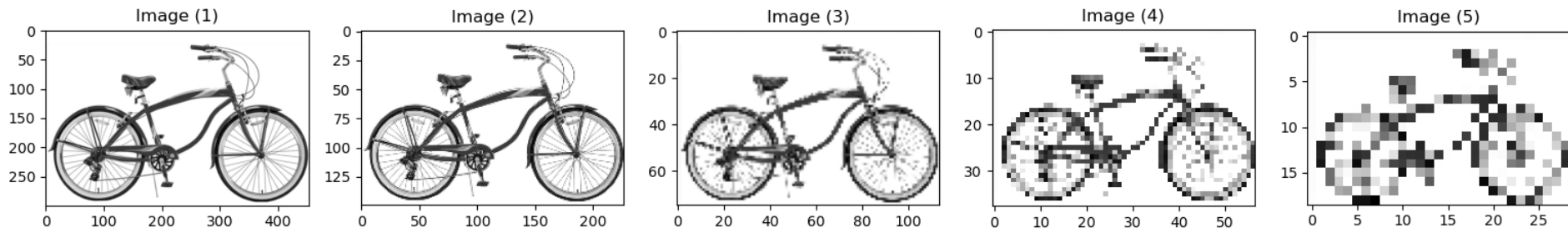


- 1) the smallest image produced by the Gaussian Pyramid is taken as the starting point.
- 2) we expand, calculate the difference, and we have the L_i level.
- 3) We move to the next level of the Gaussian pyramid, expand and calculate the difference. And we have the L_{i-1} level of the Laplacian Pyramid.
- 4) And so on...

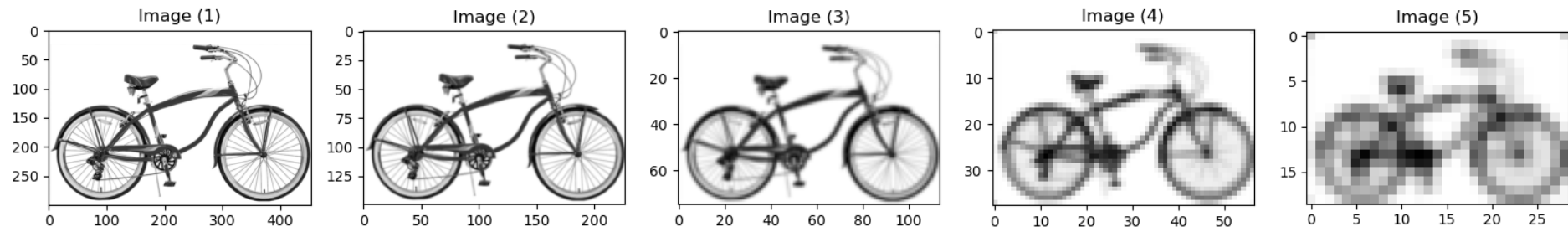
EXERCISE 2.C

- Effect of smoothing → how does this affect in terms of high frequencies?

No Gaussian smoothing. Only sub-sampling, keeping the even rows/columns (aliasing):

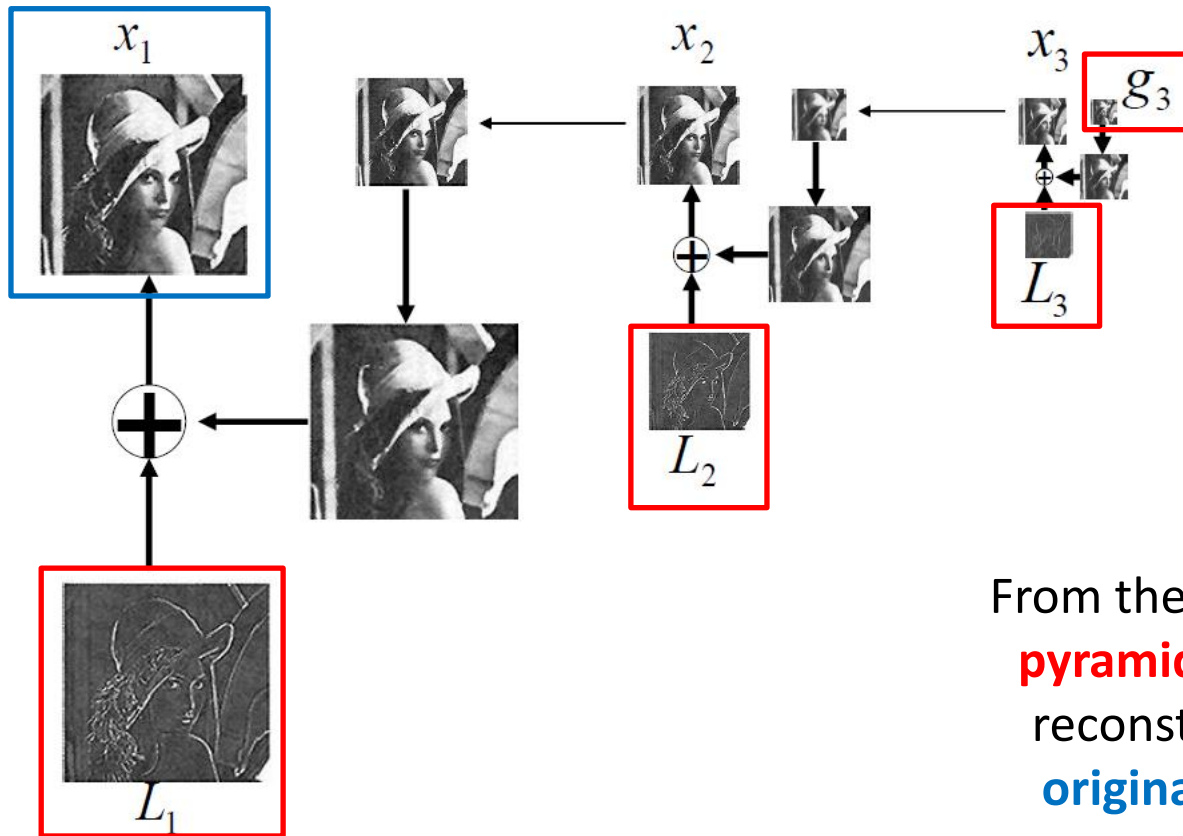


Including Gaussian smoothing:



EXERCISE 2.D

- We reconstruct x_1 from L_1, L_2, L_3 y g_3



From the **Laplacian pyramid**, we can reconstruct the **original image**

EXERCISE 2.D

- Is it possible to perfectly reconstruct the original image from a Laplacian pyramid? **Yes!**
- What does this perfect reconstruction depend on: the particular sigma employed, the interpolation used,...? **Mainly on how well you code it! 😊**

EXERCISE 2.D

Gaussian pyramid with sizeMask=7 and 4 levels

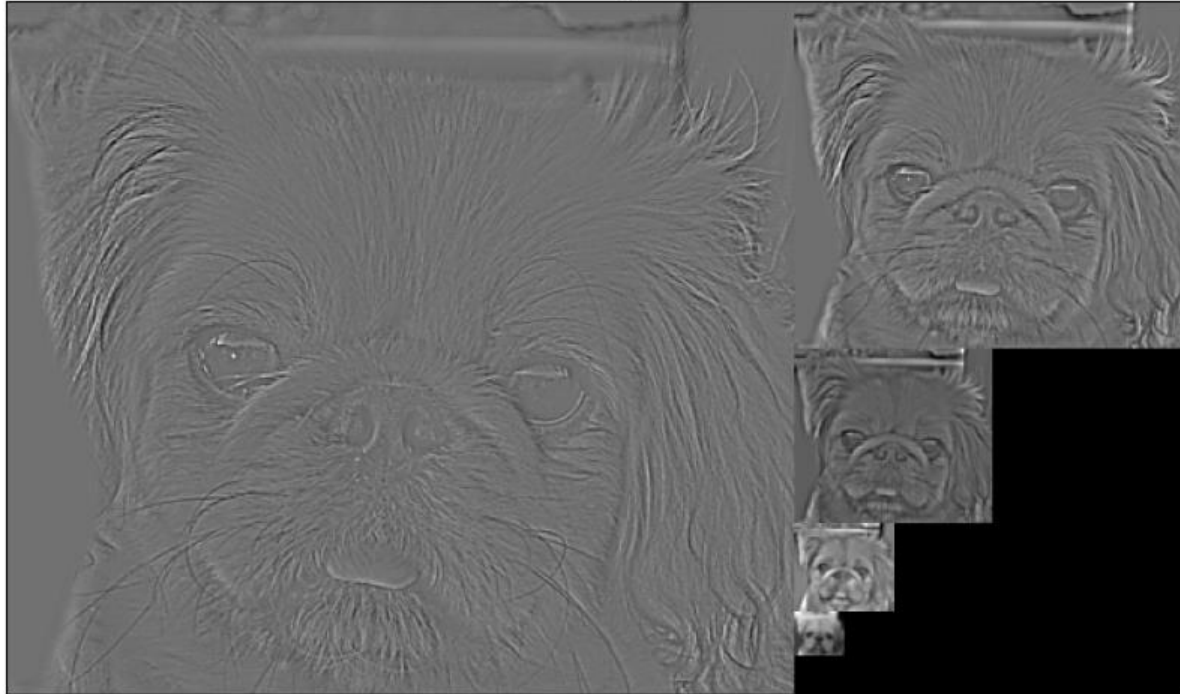
Gaussian Pyramid



EXERCISE 2.D

4-level Laplacian pyramid

Laplacian Pyramid



EXERCISE 2.D

Original Image VS Reconstructed image



Identical! All pixels are the same!!

EXERCISE 2.D

- Note:
 - Sigma does not influence the reconstruction!

Gaussian pyramid (kSize=31)

Gaussian Pyramid



Laplacian pyramid (ksize=31)

Laplacian Pyramid



EXERCISE 2.D

- But... how is this possible? Witchcraft?

Nyquist-Shannon sampling theorem: after smoothing, there are redundant pixels.
Therefore, we can discard them (by subsampling)
without losing information

S. Birchfield, Clemson Univ., ECE 847, <http://www.ces.clemson.edu/~stb/ece847>

We have a reduced version of the original image, and all the details (high frequencies) at different scales.
So, in fact, we have all the information we need.

EXERCISE 2

```
pyramidGauss(im, sizeMask, nlevel):
```

```
# We compute the Gaussian mask
```

```
pyramid = []
```

```
# We keep the original image as pyramid base (level-0)
```

```
pyramid.append(im)
```

```
for i in range(nlevel):
```

```
# We create a new level by smoothing the previous one
```

```
# and keeping half of rows and columns
```

```
return vim #list of images
```

```
pyramidLap(im, sizeMask, nlevel):
```

```
# Compute Gaussian pyramid from input image (im)
```

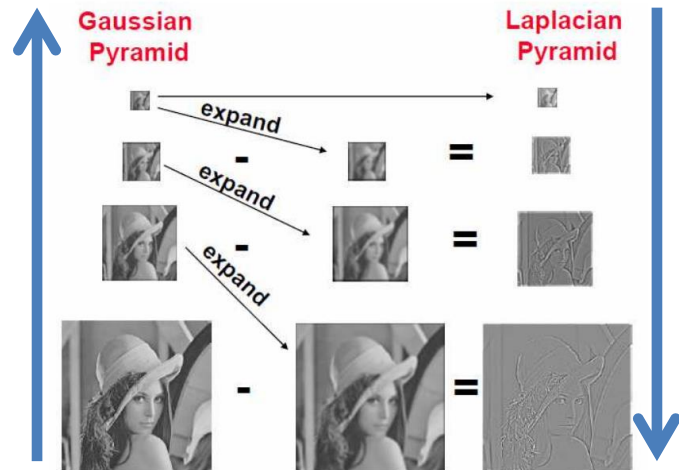
```
for i in range(nlevel):
```

```
# Resize i+1 level from Gaussian pyramid to the size of i level (cv2.resize) using bilinear interpolation
```

```
# Compute gaussian_pyramid[i] - expanded_image
```

```
# Keep the last/smallest level of the Gaussian pyramid to be able to reconstruct the original image
```

```
return vimL #list of images
```



EXERCISE 2

```
reconstructLap(pyL, flagInterp):
```

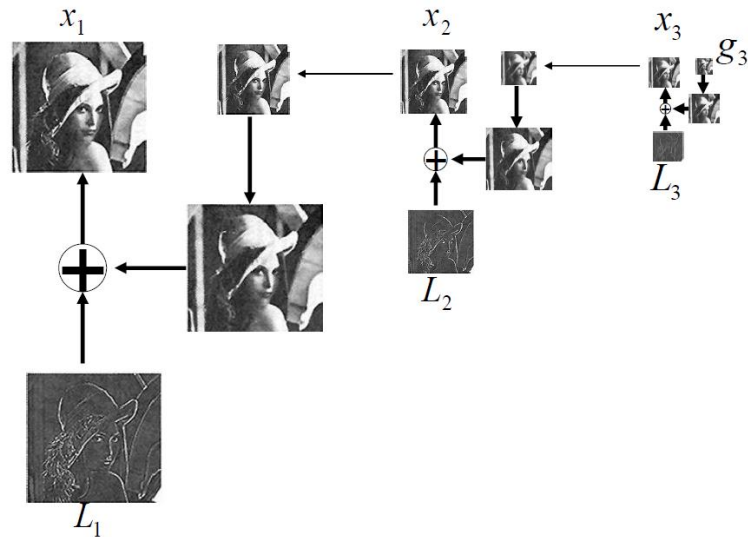
```
# Start from the last level of the Laplacian pyramid
```

```
# Go through the Laplacian pyramid, from the top, applying the algorithm:
```

```
# Expand the level to the size of the immediately lower level (cv2.resize) using the  
interpolation determined in flagInterp (bilinear)
```

```
# Compute laplacian_pyramid[i] + expanded_image
```

```
return im #return reconstructed image
```



EXERCISE 3

- A. Oliva, A. Torralba, P.G. Schyns (2006). Hybrid Images. ACM Transactions on Graphics.
 - <http://olivalab.mit.edu/hybridimage.htm>
- By appropriately **mixing part of the high frequencies of one image with part of the low frequencies of another image**, we obtain a **hybrid image** that allows for **different interpretations at different distances**.

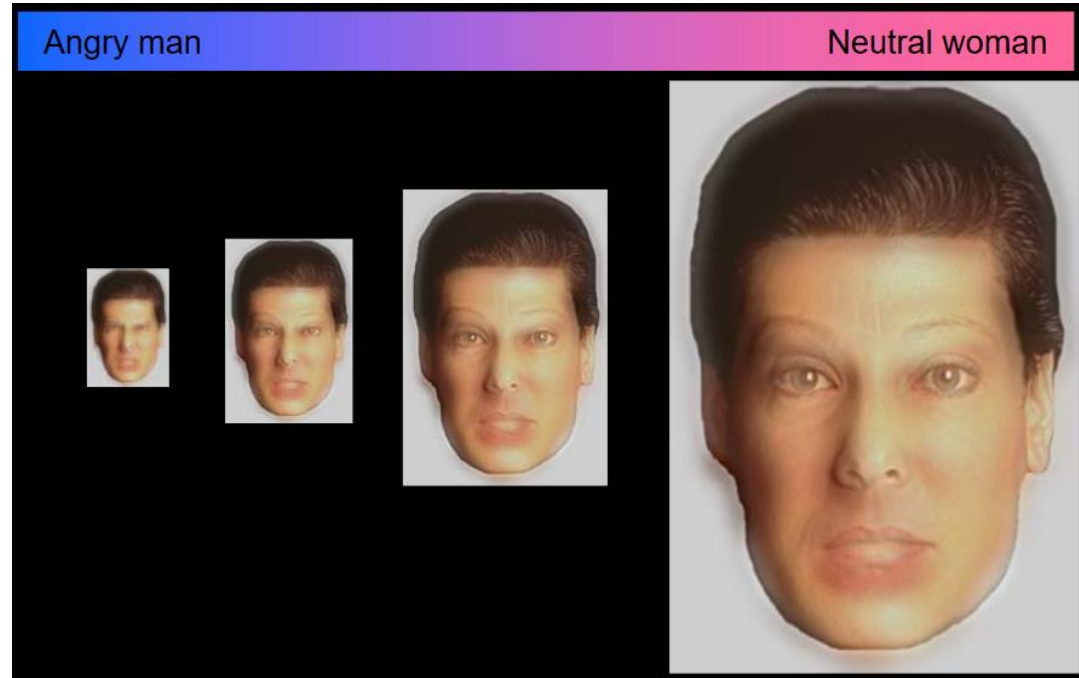


Image taken from

http://olivalab.mit.edu/publications/Talk_Hybrid_Siggraph06.pdf

EXERCISE 3

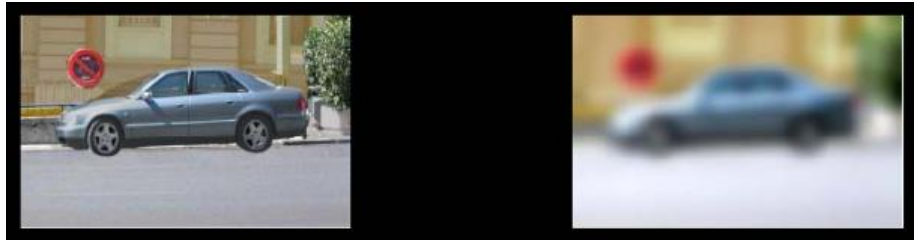
- To select the high and low frequencies we will use the sigma parameter of the Gaussian filter
 - The higher the sigma value, the greater the removal of high frequencies in the convolved image.
 - Sometimes it's necessary to choose this value separately for each of the two images.

EXERCISE 3

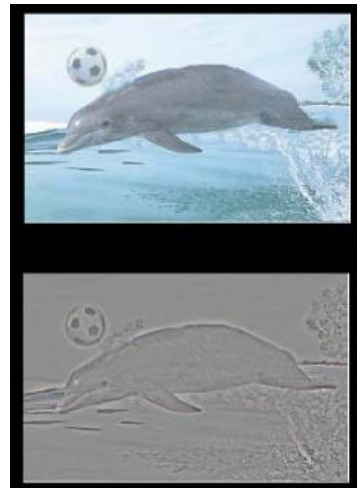
- You have to implement a function that generates the low and high frequency images from the image pairs.

Low frequencies predominate at long distances, while high frequencies predominate at short distances

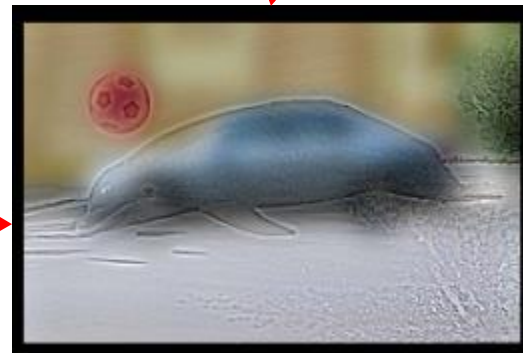
Low frequency input image



High frequency input image

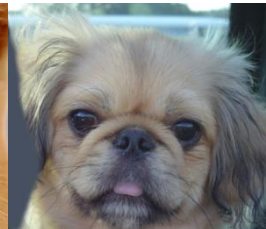
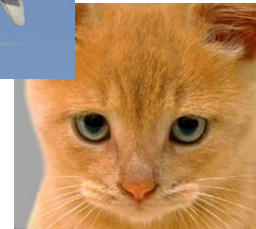
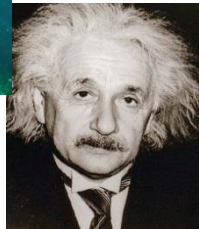


Hybrid image



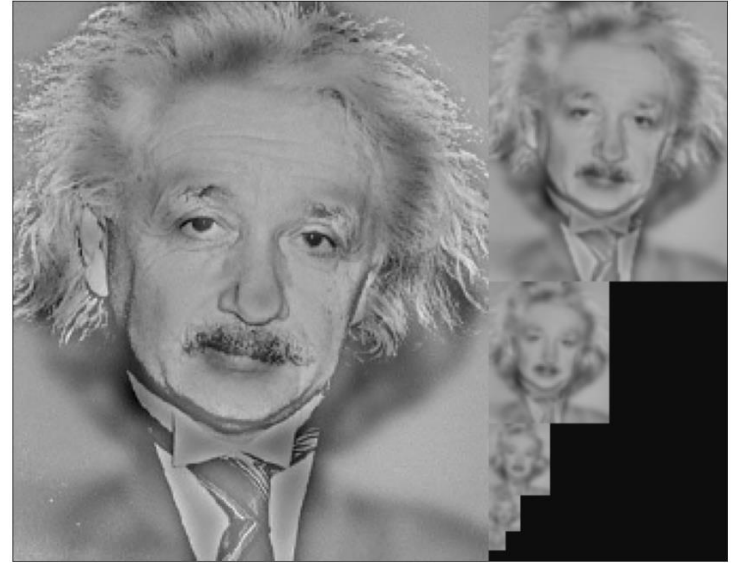
EXERCISE 3

- **If the effect is achieved, it can be seen in the Gaussian pyramid**
 - You do not need to move away from the computer to see it!
- If the effect is not achieved, the exercise is not correct!
 - E.g. if one figure always clearly predominates over the other
- Choice of high and low frequencies images:
 - High: the one with the most accentuated edges
 - Low: the one with a softer appearance



EXERCISE 3

- Possibility on how to proceed:
 - 1) Low frequencies: smooth it heavily (until only almost a spot without details remains).
 - 2) High frequencies: calculate the difference between the original and its smoothed version.



EXERCISE 4

- Pyramid Blending



VS



Burt and Adelson, "A multiresolution spline with application to image mosaics", ACM Transactions on Graphics, 1983, Vol.2, pp.217-236.

EXERCISE 4

- Pyramid Blending

1) Read two color images (A and B)

2) Generate the Laplacian pyramids for A (L_A) and B (L_B). To do so, you need to compute their Gaussian Pyramids as well.

3) Build a Gaussian pyramid (G_R) from the desired region/mask (R). To get a possibly better final result, this initial input mask (R) can also be smoothed.

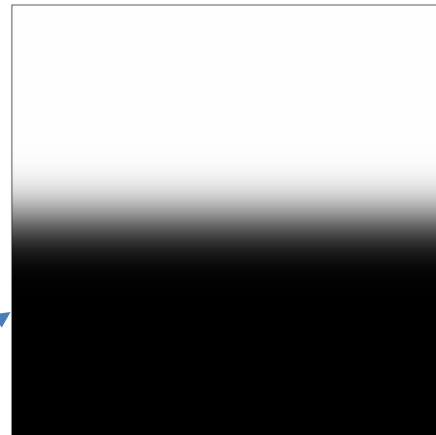
4) Mix both laplacians using G_R as a weighted mask

$$L(i, j) = G_R(i, j)L_A(i, j) + (1 - G_R(i, j))L_B(i, j)$$

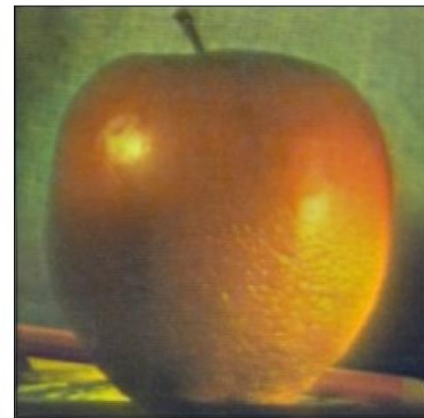
5) Start from this new blended/merged pyramid (L), and reconstruct/collapse the “original” image

Note: you can use *pyrUp* and *pyrDown* in this exercise, if you want

Useful reference: <https://becominghuman.ai/image-blending-using-laplacian-pyramids-2f8e9982077f>



Final Reconstruction



Final Remarks

- Remember to
 - Check the documentation:
 - https://docs.opencv.org/4.10.0/d4/d86/group_imgproc_filter.html
 - **Work on the discussion of results and explanation of the decisions taken, so that everything you do is clear and well justified.**
 - It's about making it easier for the reader to understand the work (and highlighting your efforts and knowledge)
→ **I cannot evaluate things that are not in the report!**

Final Remarks

- Figures must be easy to understand:
 - Appropriate resolution and size
 - Suitable title
 - They should be commented in a text cell (and, if they are not explicitly demanded in the exercise statement, their existence should be justified in some way)

Final Remarks

- About the discussion/explanation/analysis:
 - It goes in text cells, not in code cells (as code comments do)
 - One should not
 - deviate from the point
 - be excessively brief
 - Do not hesitate to
 - link it to theory in the best way you can
 - incorporate additional experiments that you consider appropriate to demonstrate your hypotheses or to delve into each exercise
 - This is not necessary but, obviously, all the students' hard work and effort will be valued positively!
 - Avoid systematic spelling mistakes → they spoil the final result!

Computer Vision

Assignment 1: Image Processing

Pablo Mesejo and David Criado

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA

