Prácticas de Visión por Computador

Práctica 1:
Filtrado de Imágenes

Pablo Mesejo y Víctor Vargas
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA

Índice

- Normas de entrega
- Breve repaso de convoluciones y filtros
- Descripción y análisis de la Práctica 1

Normas de la Entrega de Prácticas

- Se entrega solamente un fichero .ipynb integrando directamente código, resultados, análisis y discusión.

- Disponéis de una plantilla en PRADO a partir de la cual trabajar.

Normas de la Entrega de Prácticas

- Subid a PRADO solamente el fichero .ipynb. ¡Nada de subir imágenes a PRADO!
- Lectura de imágenes o cualquier fichero de entrada: "`/content/drive/MyDrive/images`"
- No escribáis nada en disco (es decir, no grabéis nada en Drive).
- La estructura de la plantilla debe ser respetada.

Entrega

- Fecha límite: 22 de Octubre
- Puntuación máxima: 10 puntos
- Lugar de entrega: PRADO

- **Se valorará mucho la explicación/discusión que acompañe a código y resultados.**

Objetivo del trabajo

- Aprender a implementar **filtros de convolución** y, particularmente, el cálculo de las derivadas de una imagen.
- Mostrar cómo usando técnicas de filtrado lineal es posible extraer información relevante de una imagen que permita su **interpretación**.

- Se trata de una práctica muy orientada hacia **procesado de imagen**.

¿Qué es una convolución?

- **Operación lineal a nivel local** con una máscara.
  - Los coeficientes/valores de dicha máscara/filtro determinan la operación realizada.

Si f y g son imágenes, a y b escalares, y L un operador lineal:

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

¿Qué es una convolución?

- Técnicamente, una convolución es una correlación cruzada (*cross-correlation*) en donde el filtro/máscara se rota 180 grados.
  - A diferencia de la correlación, la convolución verifica la propiedad asociativa, lo que nos permite construir filtros complejos a partir de filtros simples.
    - Si tenemos una imagen f, y queremos convolucionarla con g y luego con h. Al saber que $f * g * h = f * (g * h)$, Podemos convolucionar g y h, crear un único filtro, y luego convolucionar f con él.
  - Si el kernel es simétrico:
    - Convolución = Cross-correlation
  - Si el kernel es antisimétrico (p.ej. [-1 0 1]), solo cambia el signo

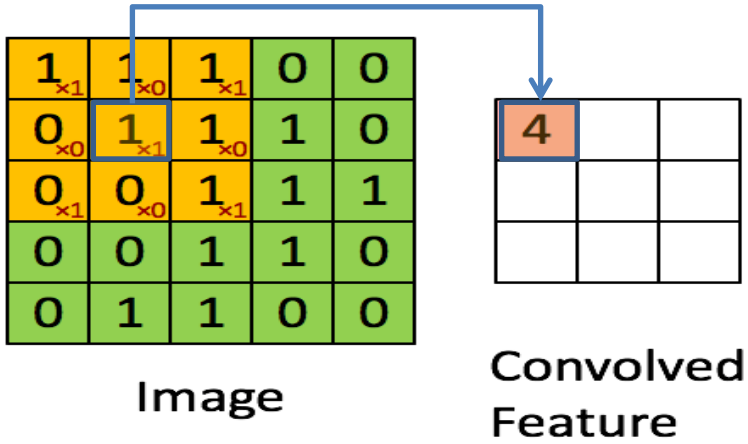
Nota: funciones como `cv2.filter2D`, realmente, aplican correlación y no convolución.

¿Qué es una convolución?

- **Operación lineal a nivel local** con una máscara.

Los números rojos representan los valores/coeficientes del filtro/máscara.

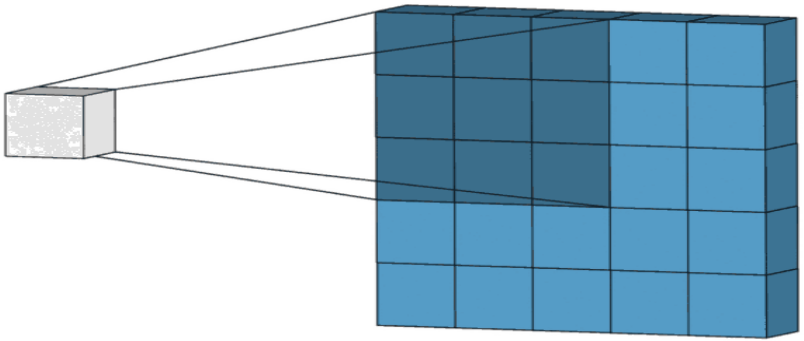
Se multiplica elemento-a-elemento el filtro con la imagen, se suman los productos, y se sustituye la posición central del filtro en la imagen.

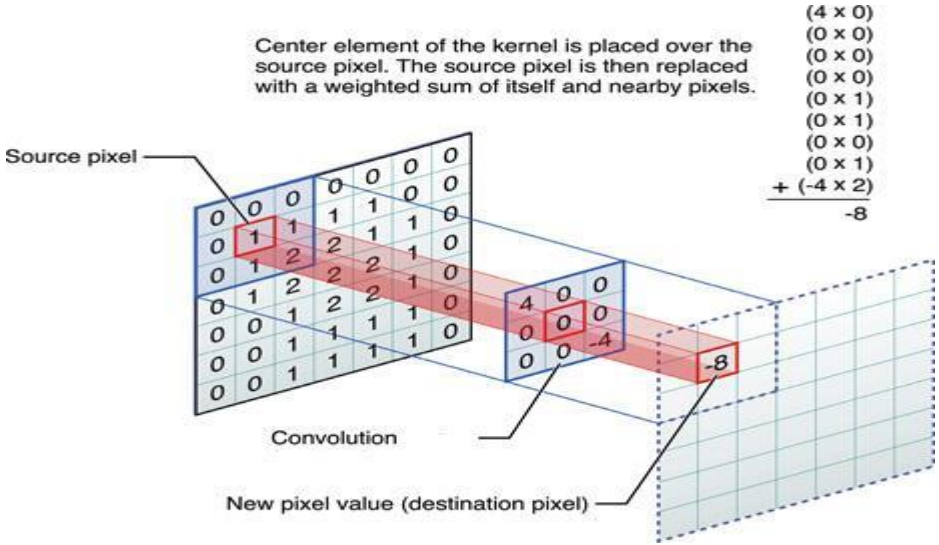


Imágenes extraídas de https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/

¿Qué es una convolución?

- Por si así lo veis más claro…





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

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

¿Qué es una convolución?

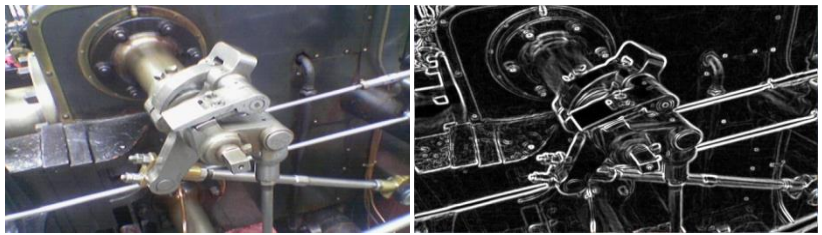
- **Operación lineal a nivel local** con una máscara.
  - Los valores de la máscara determinan el resultado (características que se extraen)

Filtro Gaussiano (elimina frecuencias altas → suaviza la imagen)

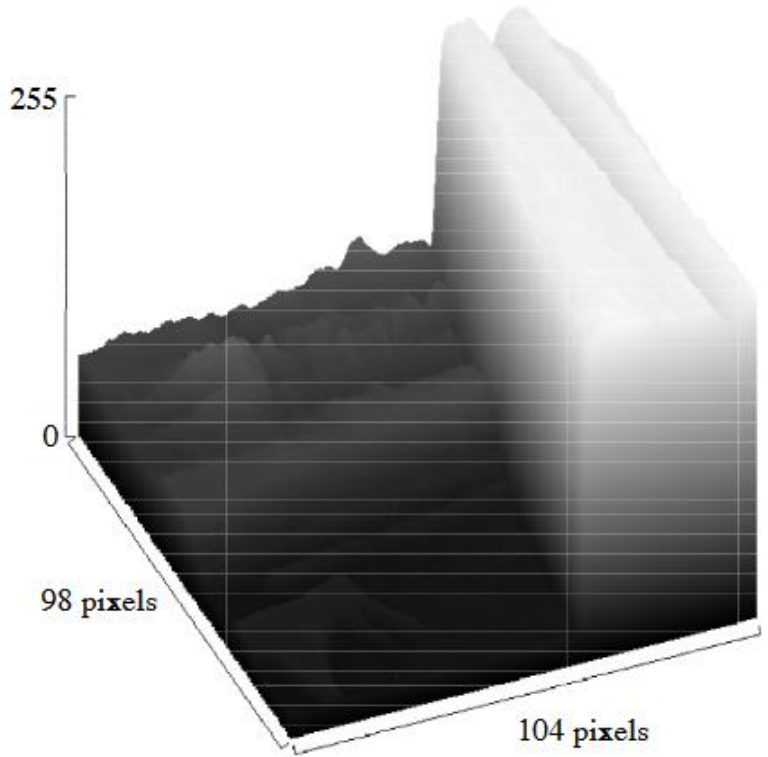
$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Filtro de Sobel (elimina bajas frecuencias → realza bordes)

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



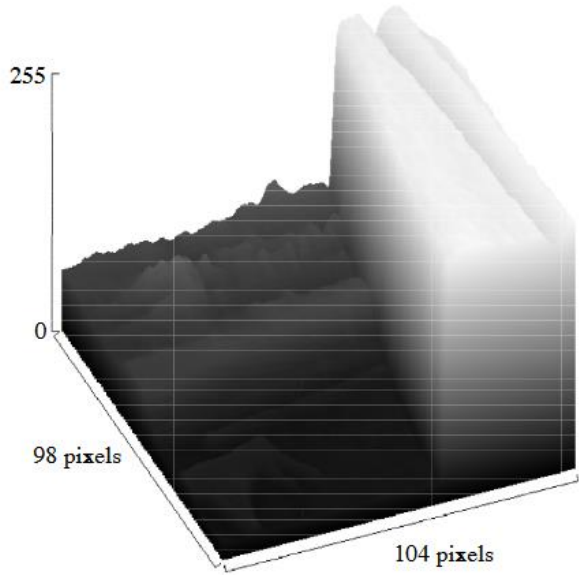
Visualización 3D de un borde



Ejemplo extraído de
https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/pdfs/D02-Images.pdf

Bordes y Altas Frecuencias

- Frecuencia en imágenes 2D:
  - tasa de cambio de niveles de gris con respecto al espacio
    - Si “implica” muchos píxeles el realizar un cambio → baja frecuencia
    - Si “implica” pocos píxeles el realizar un cambio (es decir, el cambio es brusco) → alta frecuencia



- Véase también *Spatial Frequency* o *Fourier Transform*

A la hora de calcular convoluciones…

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

Aunque esto a vosotros os da un poco igual, porque en la práctica usaréis `sepFilter2D()`

- La convolución no se puede hacer “in place”
- Imagen **de salida y entrada deben estar totalmente separadas**
  - Es decir, los nuevos valores calculados al desplazar la ventana no se pueden emplear en cálculos posteriores
- De lo contrario los cálculos son erróneos

En esta práctica

Suavizar/emborronar
(para eliminar ruido)

Diferenciación
(para remarcar detalles)

Dos tipos de kernels

Suavizado

$$\sum g_i = 1$$

(suavizar una función constante no debería cambiarla)

Ejemplo: (1/4) * [1 2 1]

Filtro paso bajo (Gaussian)

De hecho, ¿qué pasa si no dividís por la suma de los coeficientes?

🤔

Derivada/diferenciación

$$\sum g_i = 0$$

(derivar una función constante debería devolver 0)

Ejemplo: [-1 0 1]

Filtro paso alto (derivative of Gaussian)

Adelantando ideas…

En ConvNets, ¡estos valores se aprenden! No vienen prefijados por un experto humano. Son parámetros libres de la red y se entrenan como cualquier otro peso.

1989: LeCun et al. utilizaron **back-propagation para aprender directamente los coeficientes de los filtros convolucionales** a partir de imágenes de números manuscritos.

1998: LeCun et al. presentaron **LeNet-5**, ConvNet con 7 capas que permitía reconocer automáticamente números manuscritos en cheques, y demostraron que **las ConvNets superan a todos los otros modelos en esta tarea**.

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.

Ejercicio 1

- Ideas clave:
  - En un principio, solo podéis usar funciones básicas de OpenCV
    - Debéis conocer cómo ciertas operaciones se realizan a bajo nivel.
    - A no ser que os digamos explícitamente lo contrario, no podéis usar, por ejemplo, *GaussianBlur* o *pyrDown*. Tenéis que hacerlo a mano.

Ejercicio 1

- Ideas clave:
  - “implementar de modo eficiente”
    - Nos referimos a utilizar solamente convoluciones 1D.
      - Es decir, queremos hacer uso de la “separabilidad”, según la cual una convolución 2D puede ser reducida a dos convoluciones 1D.
      - Función de OpenCV `sepFilter2D()`

Ejercicio 1.A

- Calcular las máscaras discretas 1D de la función **Gaussiana**, la **derivada de la Gaussiana** y la **segunda derivada de la Gaussiana**.

EJERCICIO 1.A

Queremos discretizar una Gaussiana.

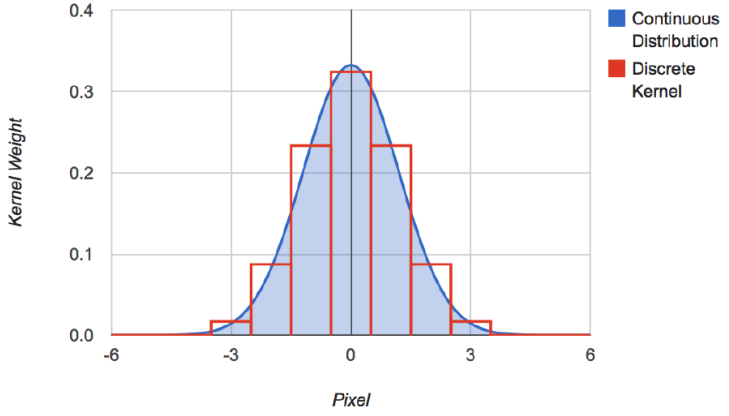
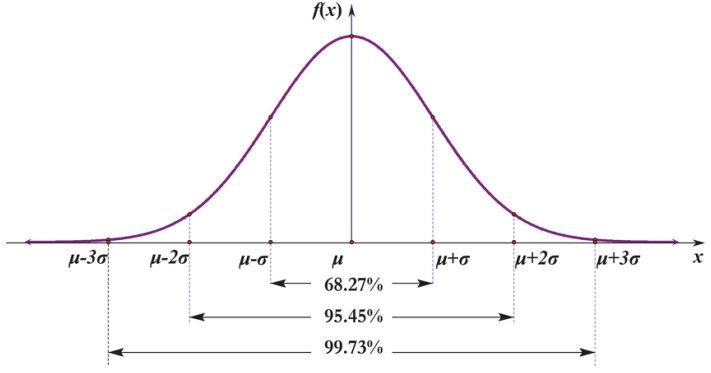


Imagen extraída de
http://dev.theomader.com/gaussian-kernel-calculator/

Sabemos que casi todos los valores se encuentran dentro de tres desviaciones estándar de la media → Tiene sentido usar, por defecto, 3σ.



EJERCICIO 1.A

Gaussian Mask 1D

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

Ignoramos la constante *c*!

- $mask: [f(-k), f(-k+1), \cdots, f(0), f(k-1), f(k)],\ k\ an\ integer$

- What k to choose ?
  - According to the Gaussian properties the k-value that verifies $\min(k) \geq 3\sigma$
  - In addition,

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

para σ= 1 el tamaño de máscara es 7 (es decir, K = 3).

La máscara debe sumar 1 → recordad que debéis dividir vuestra máscara por la suma de sus coeficientes

EJERCICIO 1.A

Debéis crear una función que:

- dado un σ, calcula el tamaño de máscara (T), y proporciona una máscara Gaussiana 1D.
- dado un tamaño de máscara (T), se ajusta automáticamente el σ, y devuelve también la máscara Gaussiana 1D.

Siendo T el tamaño de la máscara, $T = 2 \cdot k + 1 \rightarrow k = (T-1)/2 \rightarrow (T-1)/2 = 3 \cdot \sigma \rightarrow \mathbf{2 \cdot [3 \cdot \sigma] + 1 = T}$

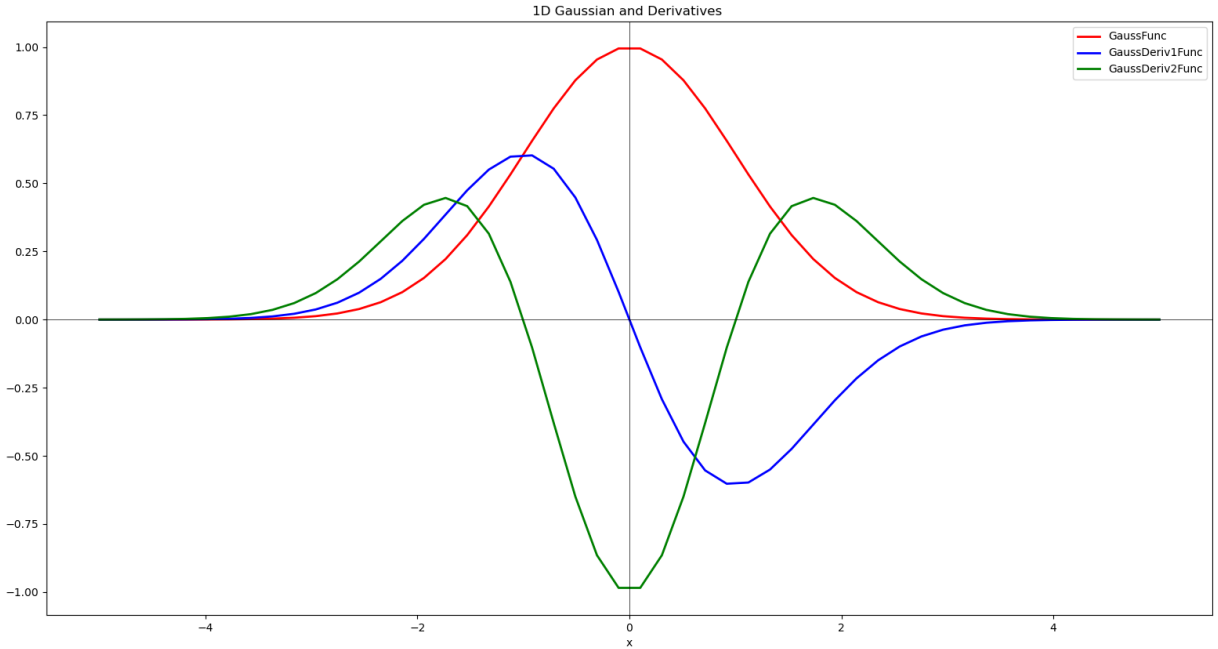
Si se proporciona, por ejemplo, T=5:

• podemos rellenar los valores de la máscara haciendo [f(-2), f(-1), f(0), f(1), f(2)] y sustituyendo en la función Gaussiana un sigma de 0.67 (que es lo que se obtiene al despejar la anterior fórmula).

Una vez tenemos σ y K ya podemos discretizar la máscara aplicando la función Gaussiana (o sus derivadas)

EJERCICIO 1.A

- Todo el proceso anteriormente descrito aplica a las derivadas de la función Gaussiana



Nota: si se incluyese el factor de normalización *c*, la forma sería igual y solo cambiaría la escala.

EJERCICIO 1.A

- Un momento… hasta ahora casi todos los filtros/kernels/máscaras que vimos eran matrices (2D), ¿esto no son vectores? 🤔
  - Sí, pero recordad que hacemos uso de la **separabilidad de los filtros**: primero, convolucionamos en una dirección (1D) y, luego, en la otra (1D). Volveremos a esto más adelante…

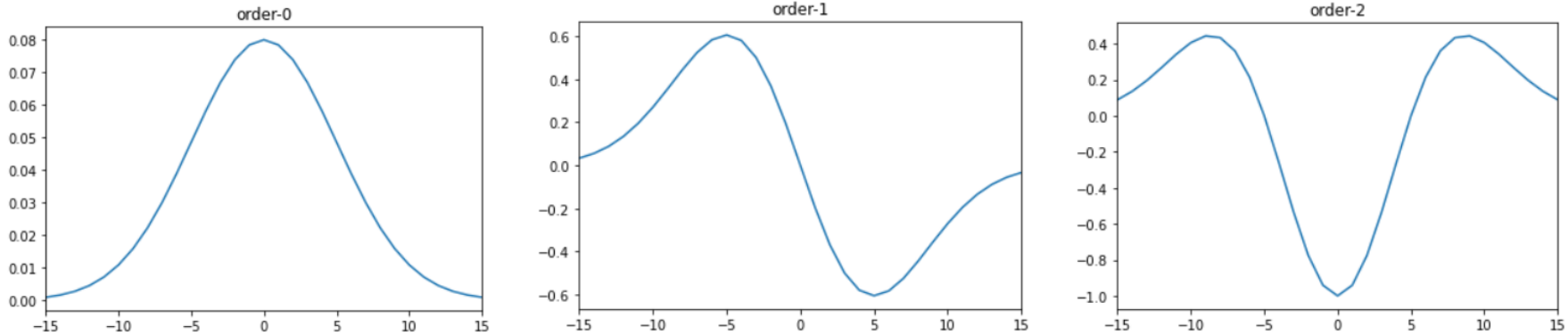
EJERCICIO 1.A

- En el caso de las derivadas:
  - Lo único que cambia es la función empleada para obtener los valores de la máscara.
    - Debéis calcular las derivadas y aplicar las funciones resultantes.
  - Recordad que **los valores de una máscara de derivadas suman cero** (son los mismos valores repetidos, pero con signo distinto).
    - Esto os puede resultar de utilidad para verificar que habéis creado correctamente la máscara.
    - Así como para ser conscientes de que, **en el caso de máscaras de derivadas, no debéis dividirlas por la suma**
      - que sería 0, o un número muy pequeño, por lo que los valores del kernel os saldrían enormes.

EJERCICIO 1.A

- **Represente el perfil (es decir, la silueta de las máscaras como funciones 1D) para verificar que las máscaras creadas son correctas**

Ejemplo de comparación visual de máscaras 1D



Nota: por un lado tenemos las máscaras Gaussianas, con sus derivadas primera y segunda, y por otro tenemos las máscaras de *getDerivKernels* que proporcionan filtros de Scharr o Sobel. Ambas sirven para detectar bordes, pero los valores concretos son diferentes

EJERCICIO 1.A

- Cómo usar cv.getDerivKernels, en caso de que os interese
  - Revisar documentación: https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga6d6c23f7bd3f5836c31cfae994fc4aea
  - cv.getDerivKernels(dx, dy, ksize)

Orden de derivada con respecto a x

Orden de derivada con respecto a y

Tamaño del kernel

```
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

EJERCICIO 1.B

- **Realizar convoluciones 2D empleando las máscaras 1D creadas en el apartado anterior**

- Debéis usar *sepFilter2D()* para aplicar sobre la imagen los filtros separables.

EJERCICIO 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 src .
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 $(-1, -1)$ 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**. **BORDER_WRAP** is not supported.

See also

filter2D, **Sobel**, **GaussianBlur**, **boxFilter**, **blur**

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

```
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.

Se refiere al tipo empleado. **Si ponéis -1 empleará el tipo de la imagen de entrada. Si es uint8, saturará a cero y no aparecerán valores negativos**. Para solucionar esto, se puede emplear cv2.CV_64F

EJERCICIO 1.B

- Se trata de crear una función que, **dada una imagen**, **un sigma**, **y** una lista con dos elementos (que representan **el orden de derivada aplicado por filas y columnas**, respectivamente) **proporciona una imagen convolucionada**.

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

EJERCICIO 1.B

La **derivada 1ª de una Gaussiana 2D** es separable:

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

Derivada de kernel Gaussiano 1D horizontal (por filas)

Kernel Gaussiano 1D vertical (por columnas)

Suaviza en una dirección, diferencia en la otra

Esto nos permite saber **qué kernels 1D aplicar, y en qué orden, para calcular las derivadas de una imagen**. En el ejemplo anterior, la derivada primera en X.

EJERCICIO 1.B

Y las **derivadas 2as de una Gaussiana** también:

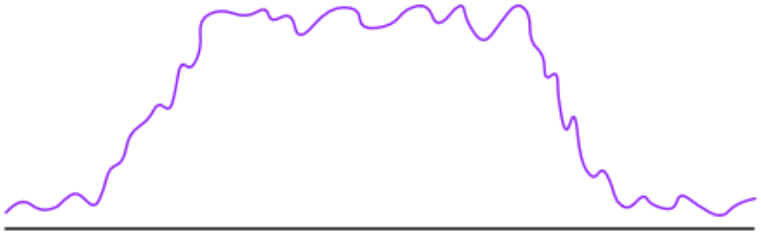
$$\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = G''_h(x) \cdot G_v(y) + G_h(x) \cdot G''_v(y)$$

convolución en una dirección con la derivada 2ª de la Gaussiana y, luego, en la otra, convolución con la Gaussiana.

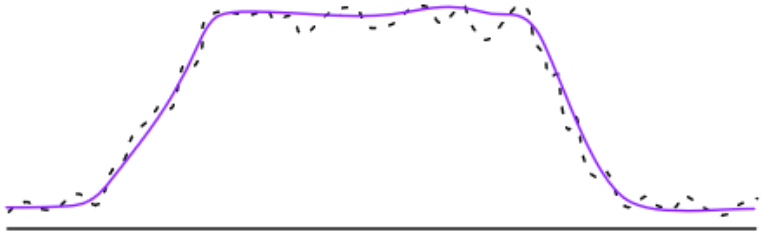
convolución en una dirección con la Gaussiana y, luego, en la otra, convolución con la derivada 2ª de la Gaussiana.

Nota: lo que veis en esta diapositiva, de hecho, es la **Laplaciana de la Gaussiana**

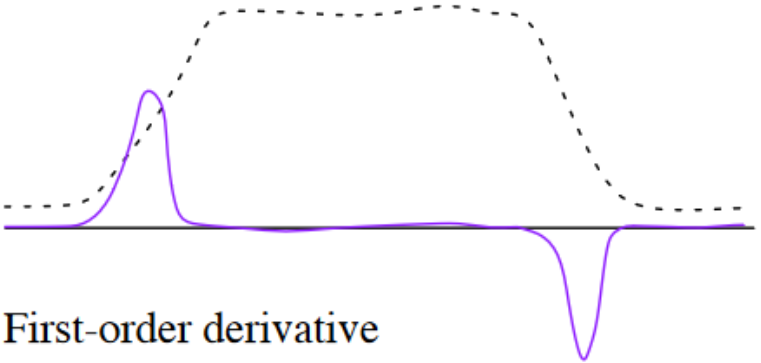
Derivadas y bordes



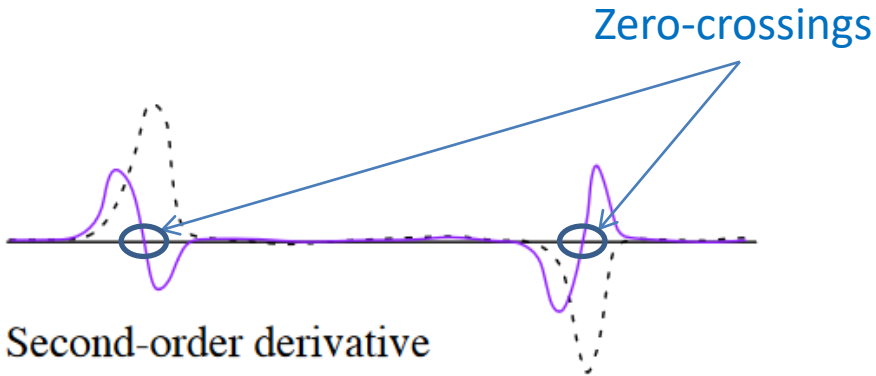
Intensity profile of an input image



After noise removal



First-order derivative



Second-order derivative

¿Ventaja de las segundas derivadas? ¡Permiten localizar con mayor precisión el borde!

Ejemplo extraído de https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/pdfs/D06-LocalOp1.pdf

EJERCICIO 1.B

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 ← 0 to height − 1 do
3        for x ← w̃ to width − 1 − w̃ do
4              val ← 0
5              for i ← 0 to w − 1 do
6                    val ← val + g_h[i] ∗ I(x + w̃ − i, y)
7              tmp(x, y) ← val
8  ▷ convolve vertical
9  for y ← w̃ to height − 1 − w̃ do
10       for x ← 0 to width − 1 do
11             val ← 0
12             for i ← 0 to w − 1 do
13                   val ← val + g_v[i] ∗ tmp(x, y + w̃ − i)
14             out(x, y) ← val
15 return out
```

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

En esencia, consiste en **recorrer toda la imagen, píxel a píxel, haciendo, primero, la convolución por filas y, luego, sobre el resultado, aplicar la convolución del kernel 1D por columnas**.

EJERCICIO 1.B

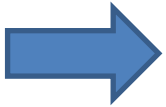
Combinamos la información de distintas filas
(kernel Y - Coefficients for filtering each column)

Combinamos la información de distintas columnas
(kernel X - Coefficients for filtering each row)

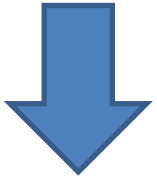
```
[1]
[2]
[1]
```

```
[ 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]
```

```
[1, 2, 1]
```



```
[  0,   0,   0,   0,   0]
[  0,  60,  90, 120,   0]
[  0, 160, 200, 240,   0]
[  0, 180, 210, 240,   0]
[  0,   0,   0,   0,   0]
```



```
[  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]
```

Imagen de entrada

```
[ 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

```
[1, 2, 1]
```

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

Separable Kernel

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

EJERCICIO 1.B

- Notas importantes:
  - Todo se hace en escala de grises: *cv.imread(filename,0)*
    - trabajar en color RGB es solo hacer por triplicado lo que se hace en monobanda
  - Operad en flotante para evitar problemas de cálculo y truncamiento de valores (véase slide siguiente)

- Enlace interesante a nivel práctico: http://www.songho.ca/dsp/convolution/convolution2d_separable.html
  - Sirve para comprender mejor la convolución 2D a partir de kernels 1D.

Nota técnica: 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(G
    aussianKernel)),1,1)

compare_images(imageConvolved, imGaussianBlur)
```

Números de píxeles diferentes en ambas imágenes (con una diferencia superior a 1 nivel de gris): **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(G
    aussianKernel)),1,1)

compare_images(imageConvolved, imGaussianBlur)
```

Números de píxeles diferentes en ambas imágenes (con una diferencia superior a 1 nivel de gris): **8.117%** (11400 de 140454)

EJERCICIO 1.B

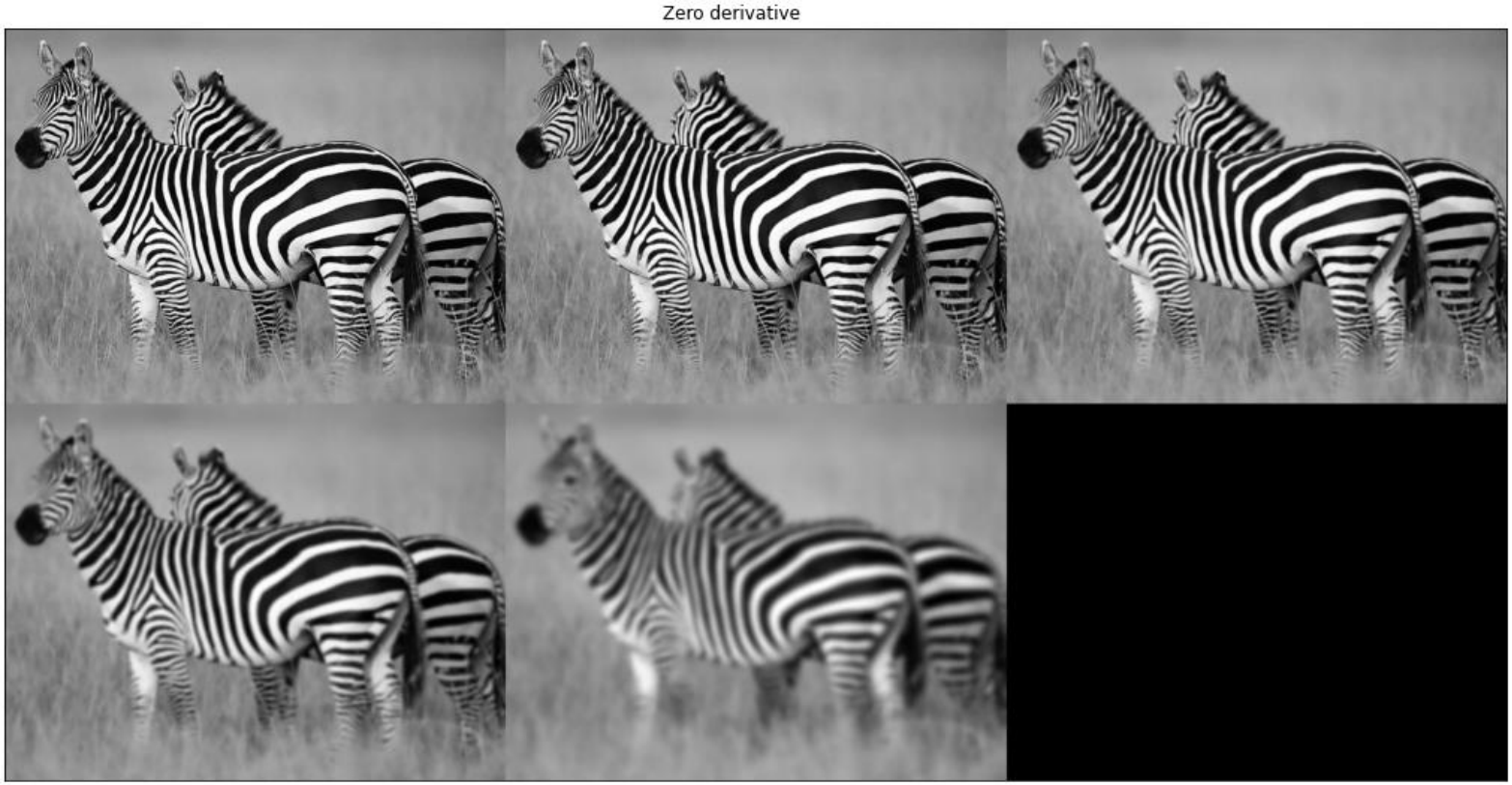
- En nuestro caso, todo es Gaussiano y, por tanto, directamente descomponible.

  → en toda la práctica, no es necesario calcular la descomposición en valores singulares. Véase https://bartwronski.com/2020/02/03/separate-your-filters-svd-and-low-rank-approximation-of-image-filters/

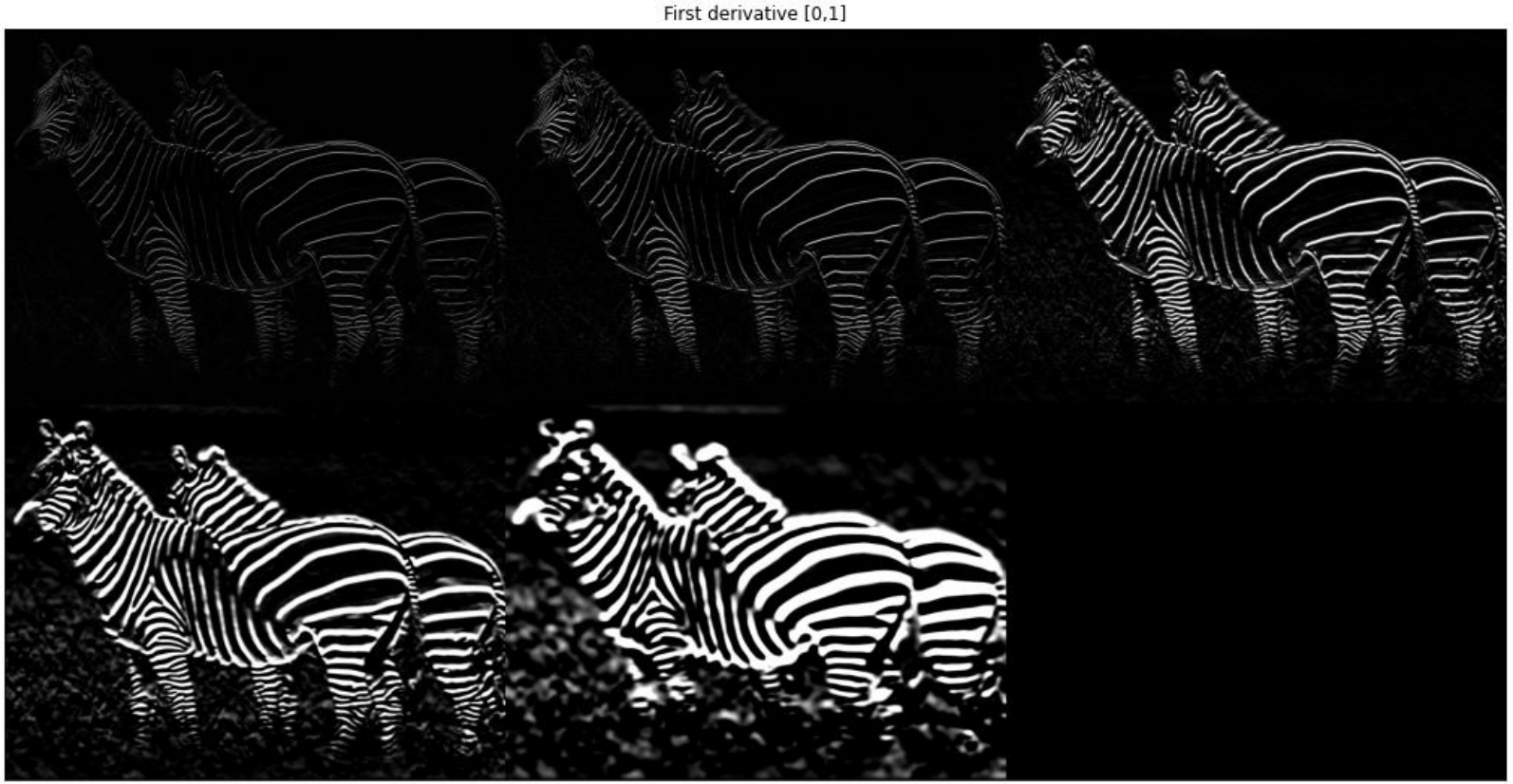
EJERCICIO 1.B

- **Podéis comparar vuestros resultados con *cv.GaussianBlur***
  - Si a esta función le pasáis un tamaño de kernel y sigma=-1, la propia función os estima el sigma adecuado.
  - Si el tamaño de kernel es 0 → lo estima a partir de sigma
- Nota importante:
  - **OpenCV prima eficiencia ante precisión**.
  - Si implementais a mano la convolución:
    - que no os extrañe si vuestro resultado de la convolución no sea exactamente igual al proporcionado por *cv2.GaussianBlur*.
    - seguramente vuestro código será “más correcto”, pero más lento.

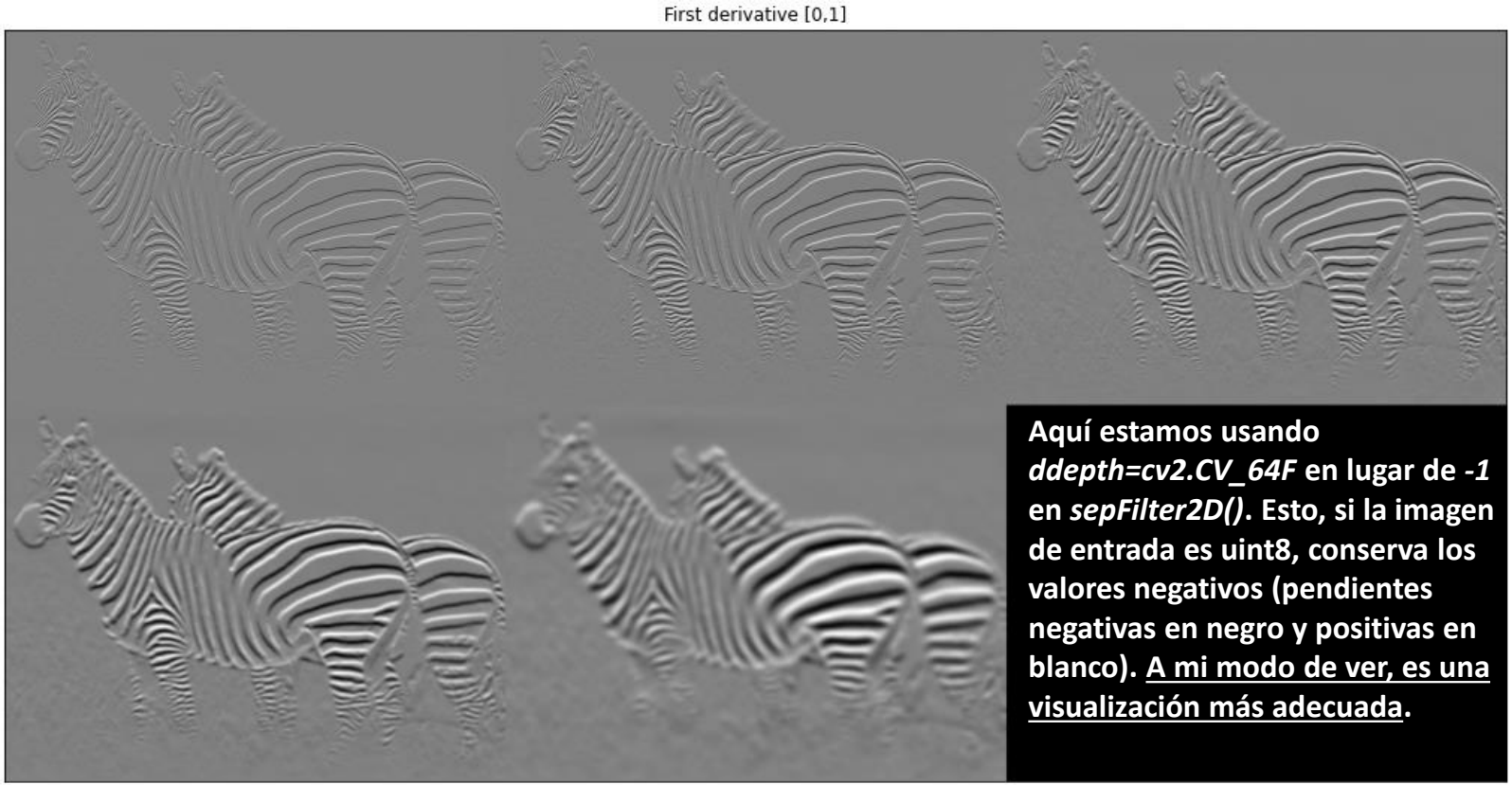
EJERCICIO 1.B



EJERCICIO 1.B



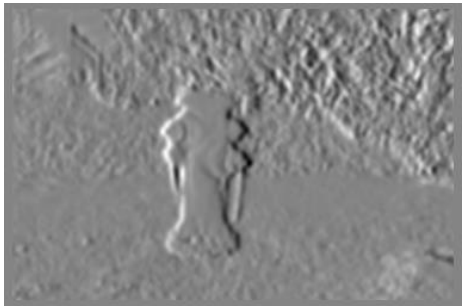
EJERCICIO 1.B: cuestión de visualización



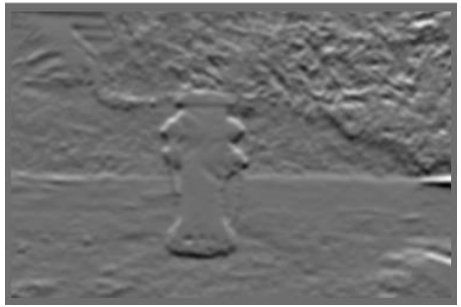
Aquí estamos usando *ddepth=cv2.CV_64F* en lugar de *-1* en *sepFilter2D()*. Esto, si la imagen de entrada es uint8, conserva los valores negativos (pendientes negativas en negro y positivas en blanco). A mi modo de ver, es una visualización más adecuada.

EJERCICIO 1.C

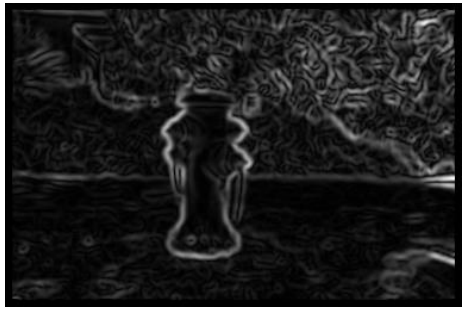
- **Usando las máscaras del ejercicio 1.A, calcular el gradiente y la Laplaciana de una imagen dada. Comenzamos por el gradiente:**



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



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



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

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

Derivadas y Bordes

Derivada horizontal

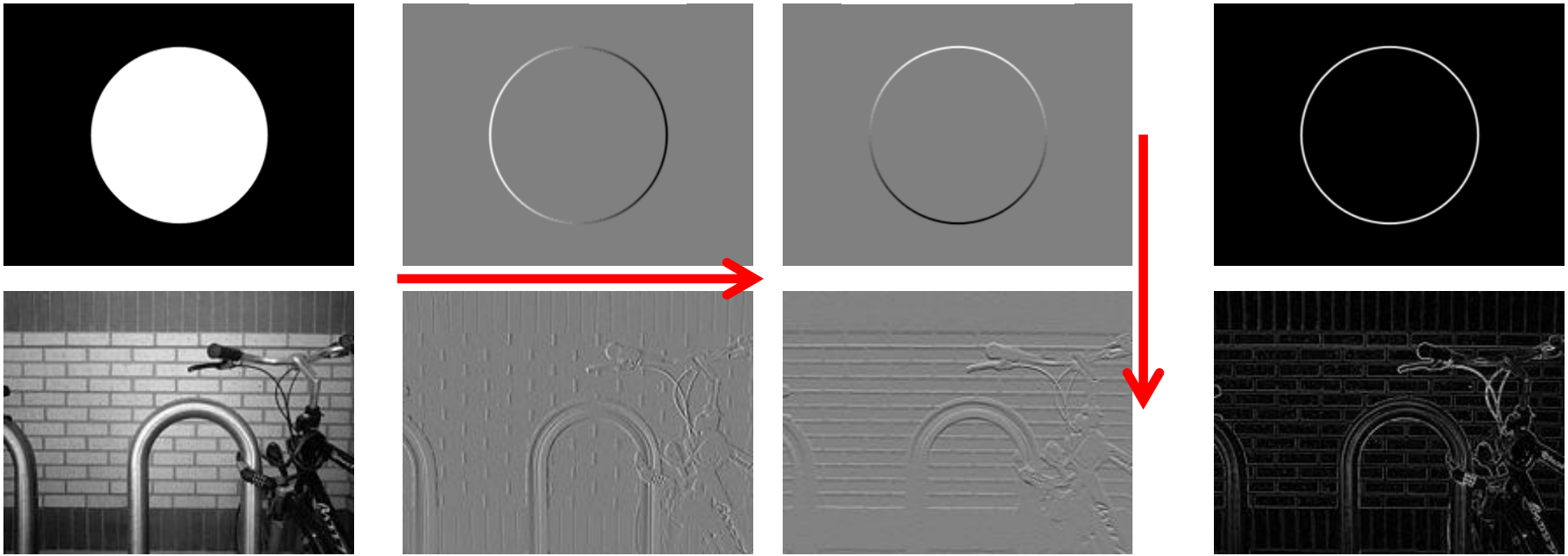
$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

Derivada vertical

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

Magnitud del gradiente

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



Ejemplos visuales extraídos de https://stackoverflow.com/questions/19815732/what-is-the-gradient-orientation-and-gradient-magnitude y https://en.wikipedia.org/wiki/Sobel_operator

Derivadas y Bordes

Derivada horizontal

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

Derivada vertical

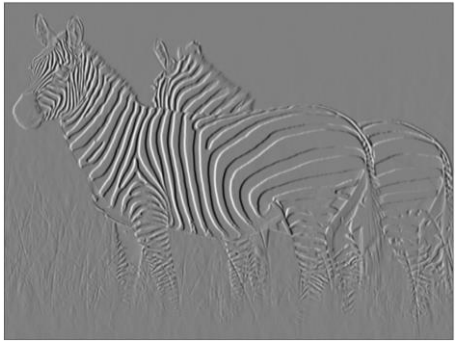
$$\mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

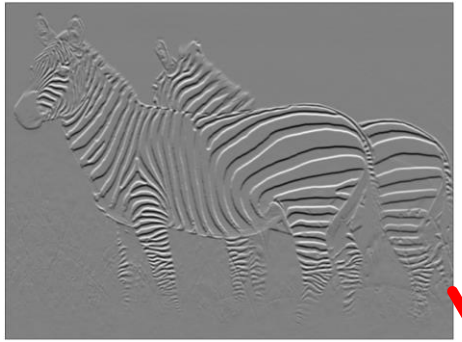
Magnitud del gradiente

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

Orientación del gradiente

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

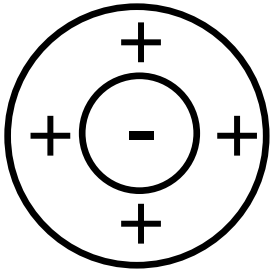




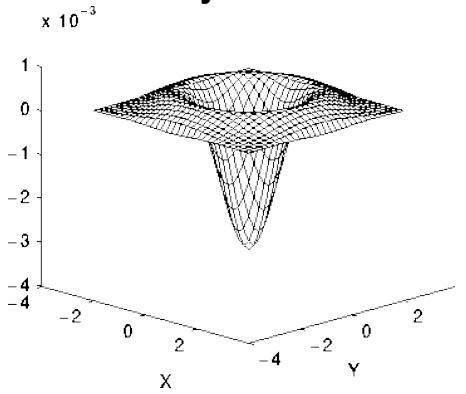
EJERCICIO 1.C

- Ejemplo de kernel Laplaciano 2D 3x3:

```
 0  1  0
 1 -4  1
 0  1  0
```



“sombrero Mejicano invertido”



- ¿Es separable?
  - Primero, ¿qué es necesario para que un kernel sea separable?

EJERCICIO 1.C

- ¿Qué es necesario para que un kernel sea separable?

Un kernel 2D es *separable* si y solo si todas sus filas/columnas son linealmente dependientes

- Este filtro $\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$, por tanto, NO lo es:

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

→ Son vectores linealmente independientes!!

EJERCICIO 1.C

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


np.linalg.matrix_rank(G)
2

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

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
```

Si el rango no es 1 → no es directamente separable!
Recordad que el rango de una matriz es el número máximo de columnas/filas linealmente independientes.

En el caso de la LoG, tenemos dos matrices con rango=1
→ **Tenemos dos filtros 2D separables!!!!**

EJERCICIO 1.C

- Pero… podemos calcular LoG como la suma de convoluciones 2D que sí son separables
  - Porque la Gaussiana y sus derivadas sí lo son!!!

    → en la práctica, necesitamos 4 convoluciones 1D
    (y eso sigue siendo más "económico" computacionalmente que hacer una única convolución 2D; a no ser que el kernel sea muy pequeño)

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

Interesantes referencias sobre la LoG: https://stackoverflow.com/questions/53544983/how-is-laplacian-filter-calculated/53545480#53545480 y https://www.crisluengo.net/archives/1099/

EJERCICIO 1.C

El **operador Laplaciano** es la divergencia del gradiente, y viene dado por la suma de las derivadas de segundo orden (i.e. la traza de la matriz Hessiana):

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

Laplaciano de la **Gaussiana**

Propiedad asociativa

Propiedad distributiva

$$\frac{\partial^2 (I \circledast G)}{\partial x^2} + \frac{\partial^2 (I \circledast G)}{\partial y^2} = I \circledast \left(\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right) = I \circledast \frac{\partial^2 G}{\partial x^2} + I \circledast \frac{\partial^2 G}{\partial y^2}$$

Laplaciana de una imagen suavizada con un kernel Gaussiano

$=$

Imagen convolucionada con la Laplaciana de una Gaussiana

EJERCICIO 1.C

¿Cómo calculamos $I \circledast \frac{\partial^2 G}{\partial x^2} + I \circledast \frac{\partial^2 G}{\partial y^2}$?

Sabemos que la **derivada 1ª de una Gaussiana 2D** es separable:

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

Derivada de kernel Gaussiano 1D horizontal

Kernel Gaussiano 1D vertical

Suaviza en una dirección, diferencia en la otra

EJERCICIO 1.C

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

convolución en una dirección con la derivada 2ª de la Gaussiana y, luego, en la otra, convolución con la Gaussiana.

convolución en una dirección con la Gaussiana y, luego, en la otra, convolución con la derivada 2ª de la Gaussiana.

$$\begin{aligned}
\frac{d^2 G(x)}{dx^2} &= \frac{d^2}{dx^2}\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-x^2}{2\sigma^2}\right)\right] \\
&= \frac{d}{dx}\left[\frac{dG(x)}{dx}\right] \\
&= \frac{d}{dx}\left[-\frac{x}{\sigma^2}G(x)\right] \\
&= -\frac{G(x)}{\sigma^2} - \frac{x}{\sigma^2}\frac{dG(x)}{dx} \\
&= -\frac{G(x)}{\sigma^2} - \frac{x}{\sigma^2}\left[-\frac{x}{\sigma^2}G(x)\right] \\
&= \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right] G(x)
\end{aligned}$$

EJERCICIO 1.C

- **Laplaciana de una Gaussiana**

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

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

1) dxx: la convolución por filas con la derivada segunda de la Gaussiana y, luego, por columnas, la convolución con la Gaussiana.

2) dyy: la convolución por filas con la Gaussiana y, luego, por columnas, la convolución con la derivada segunda de la Gaussiana.

3) sumamos dxx y dyy

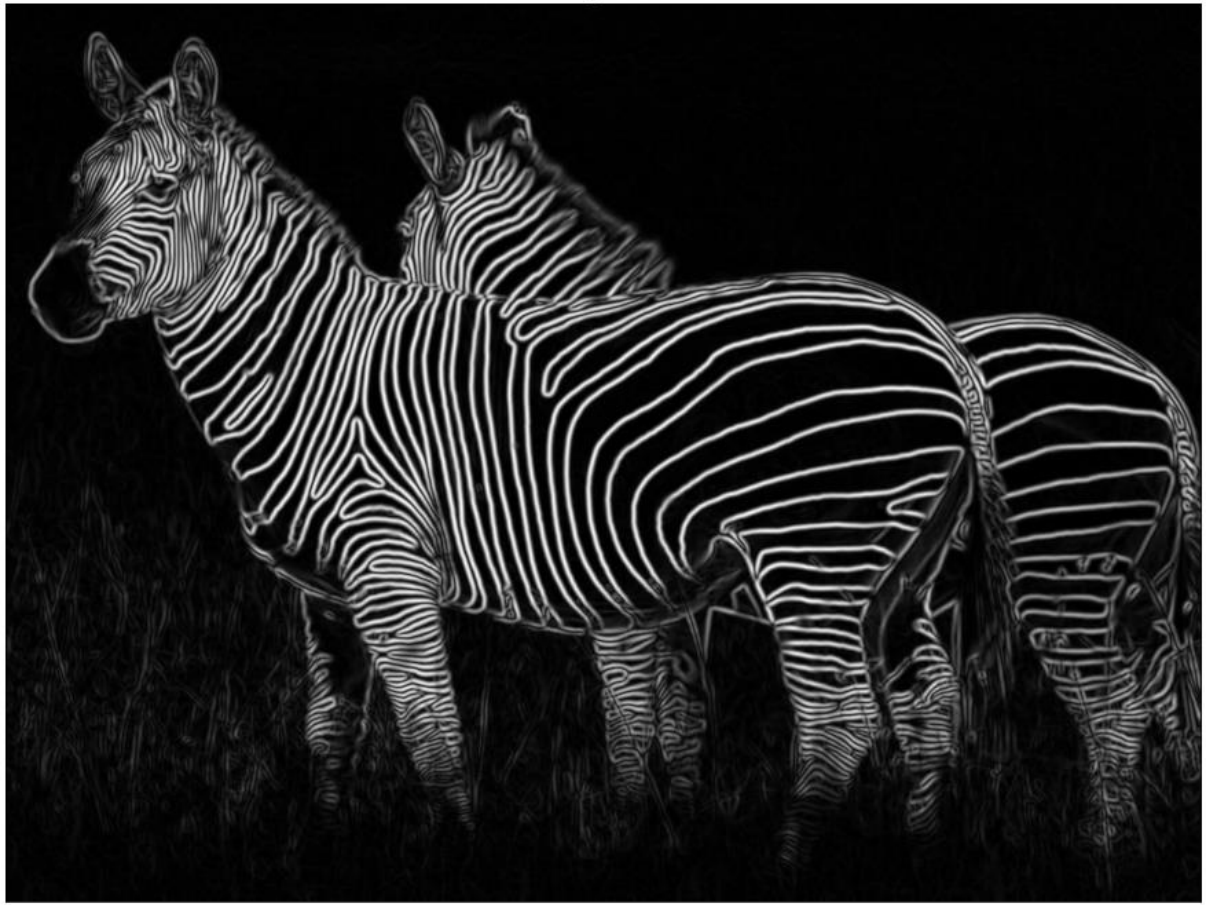
4) el resultado de la suma lo multiplicamos por σ^2

Realmente, debéis escalar/normalizar las máscaras obtenidas en 1.A, **multiplicándolas por σ (1ª derivada) o σ² (2ª derivada)**

Referencia útil: https://www.crisluengo.net/archives/1099/

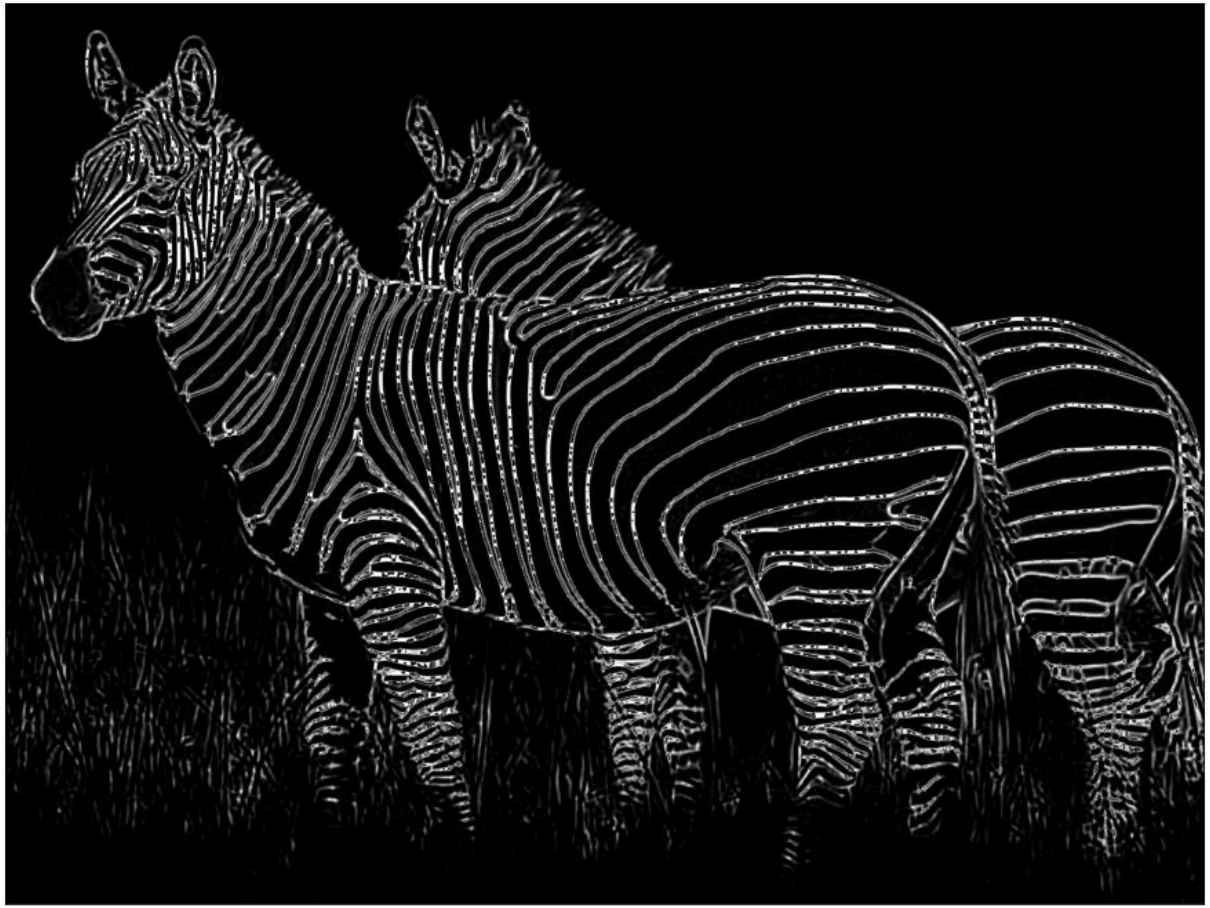
EJERCICIO 1.C

Magnitud del gradiente



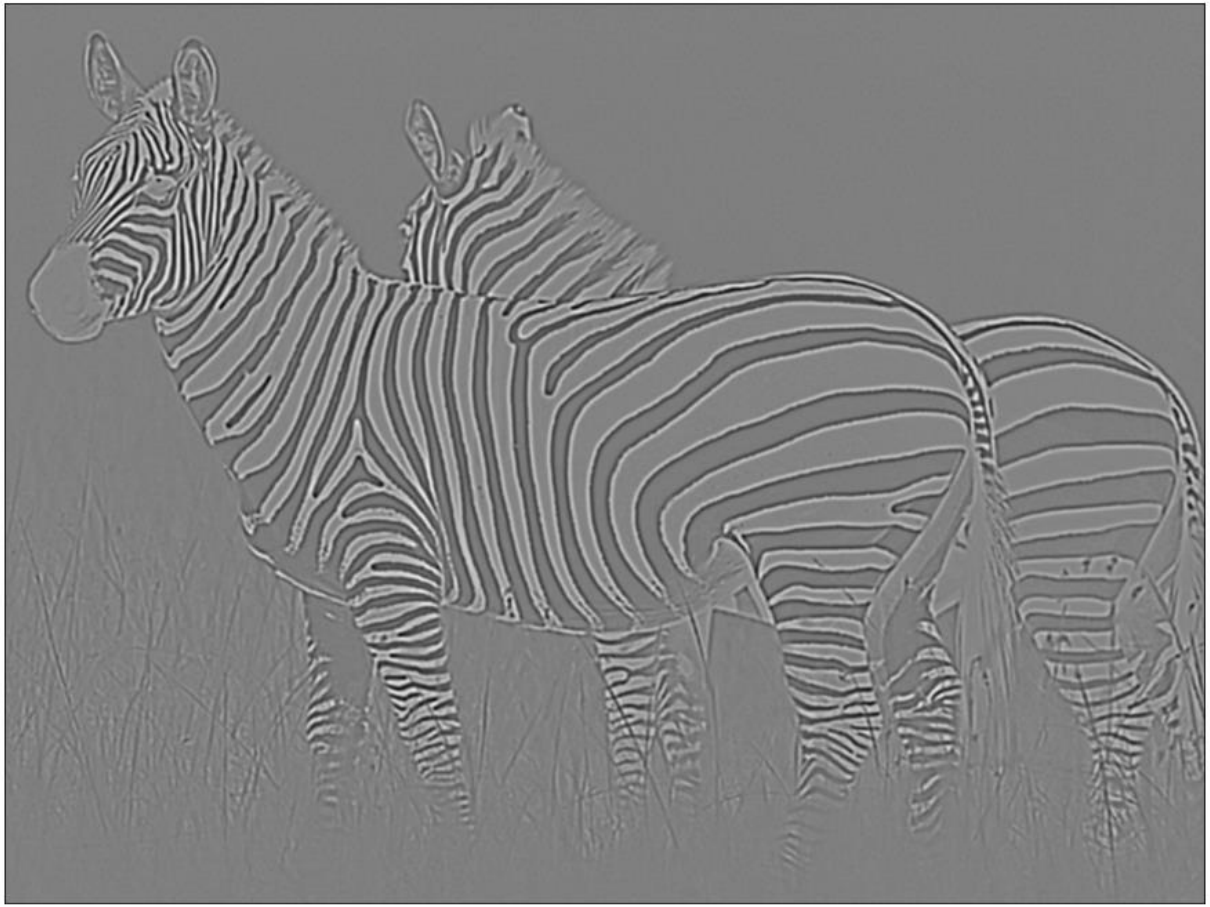
EJERCICIO 1.C

Laplacian of Gaussian



EJERCICIO 1.C

Laplacian of Gaussian
(usando ddepth=cv2.CV_64F)



EJERCICIO 1

Discretización de máscaras + convoluciones 2D por medio de máscaras 1D (separabilidad)

Suavizado de imágenes por medio de filtrado Gaussiano

Detección/realce de bordes por medio de derivadas de la Gaussiana, y Laplacian of Gaussian

EJERCICIO 2

- **2.A Generar una representación en pirámide Gaussiana de 4 niveles**
- **2.B Generar una representación en pirámide Laplaciana de 4 niveles**
- **2.C Emplear la pirámide Laplaciana para recuperar la imagen original**

EJERCICIO 2

- Importancia de las **pirámides de imágenes**:
  - Estamos habituados a trabajar con imágenes de tamaño fijo.
  - Pero, en ocasiones, podemos necesitar **trabajar con una imagen a diferentes resoluciones**.
    - Por ejemplo, si buscamos algo concreto, como una cara, y no sabemos *a priori* el tamaño del objeto buscado.
    - O si necesitamos acceder a una imagen con distintos niveles de difuminación/suavizado (*blur*).
    - O si necesitamos comprimir una imagen.

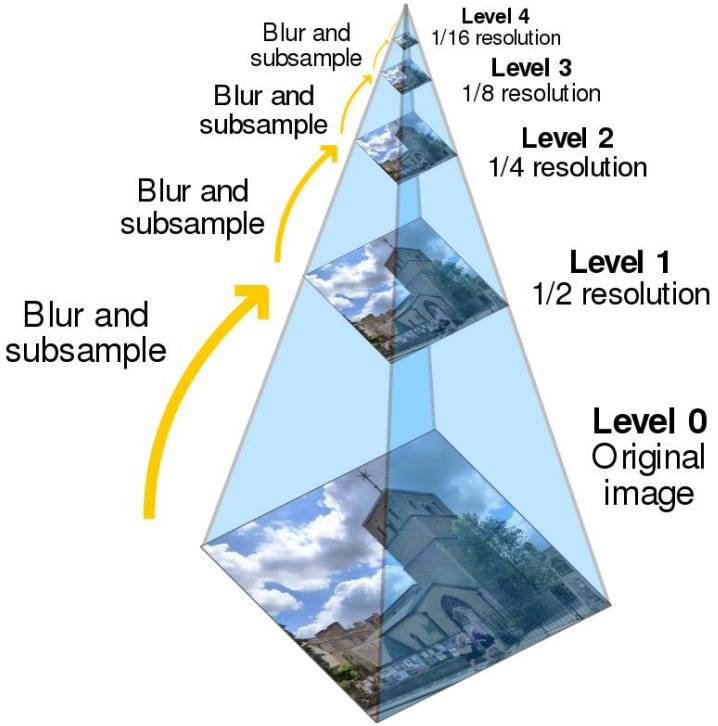
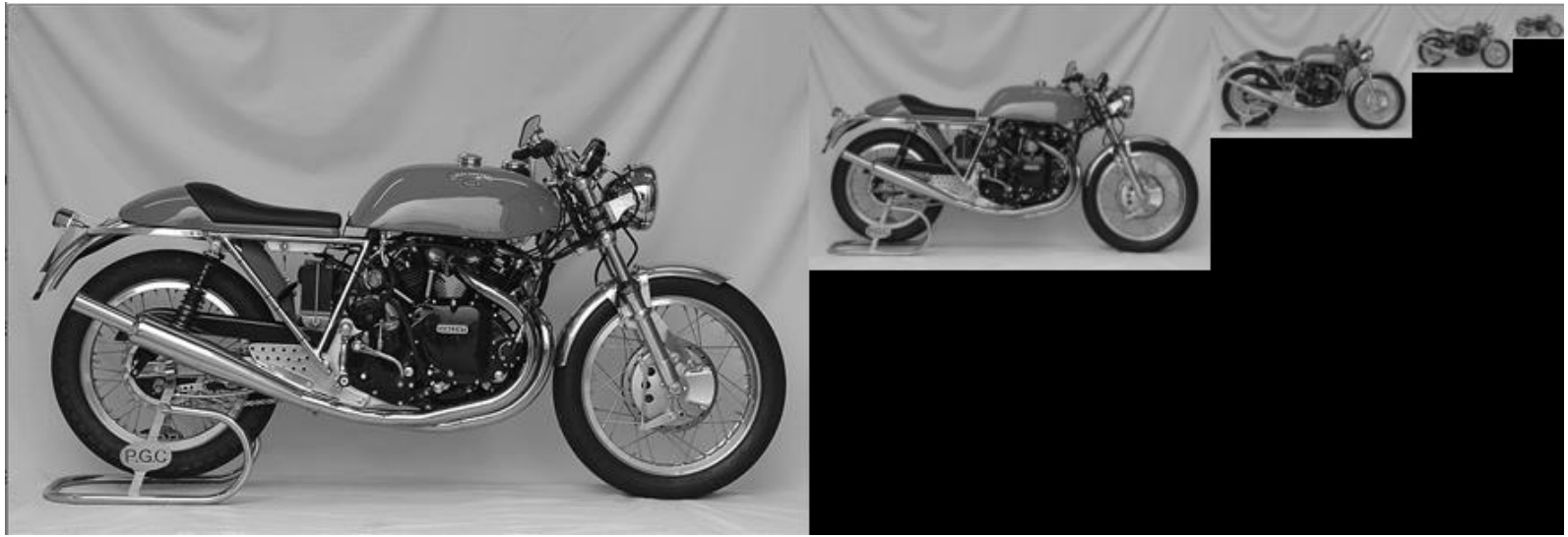


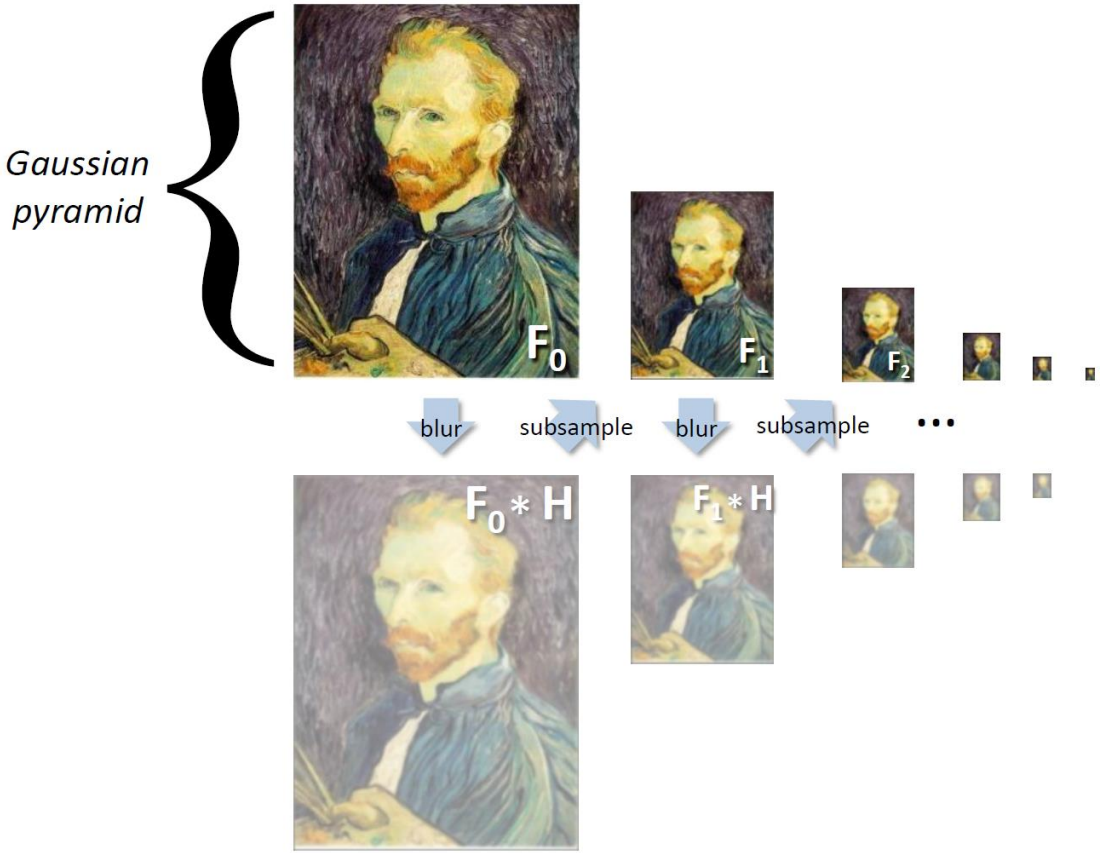
Imagen extraída de Wikimedia

EJERCICIO 2.A

- Ejemplo de pirámide Gaussiana de 4 niveles:



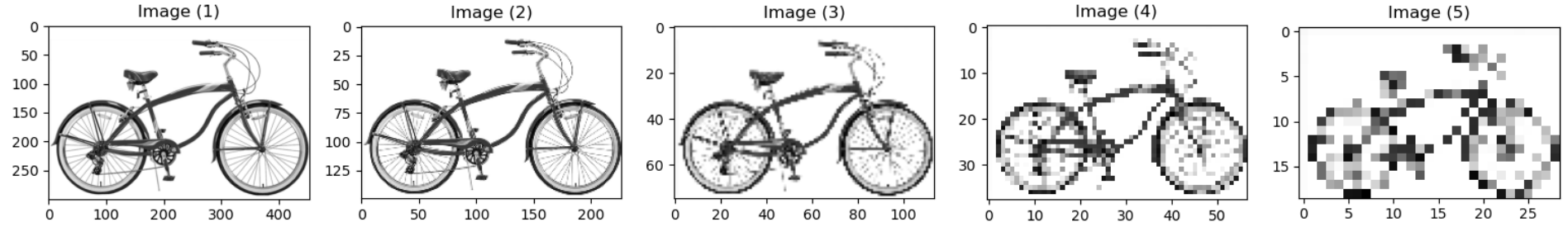
EJERCICIO 2.A



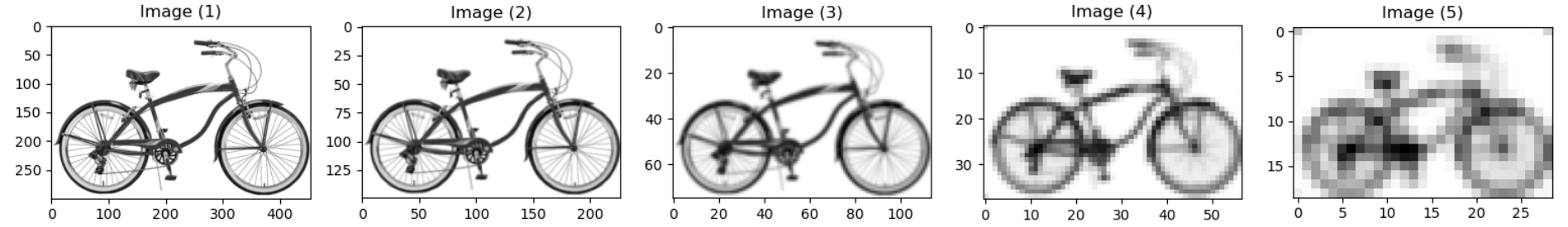
EJERCICIO 2.A

- Efecto del suavizado a la hora de crear la pirámide

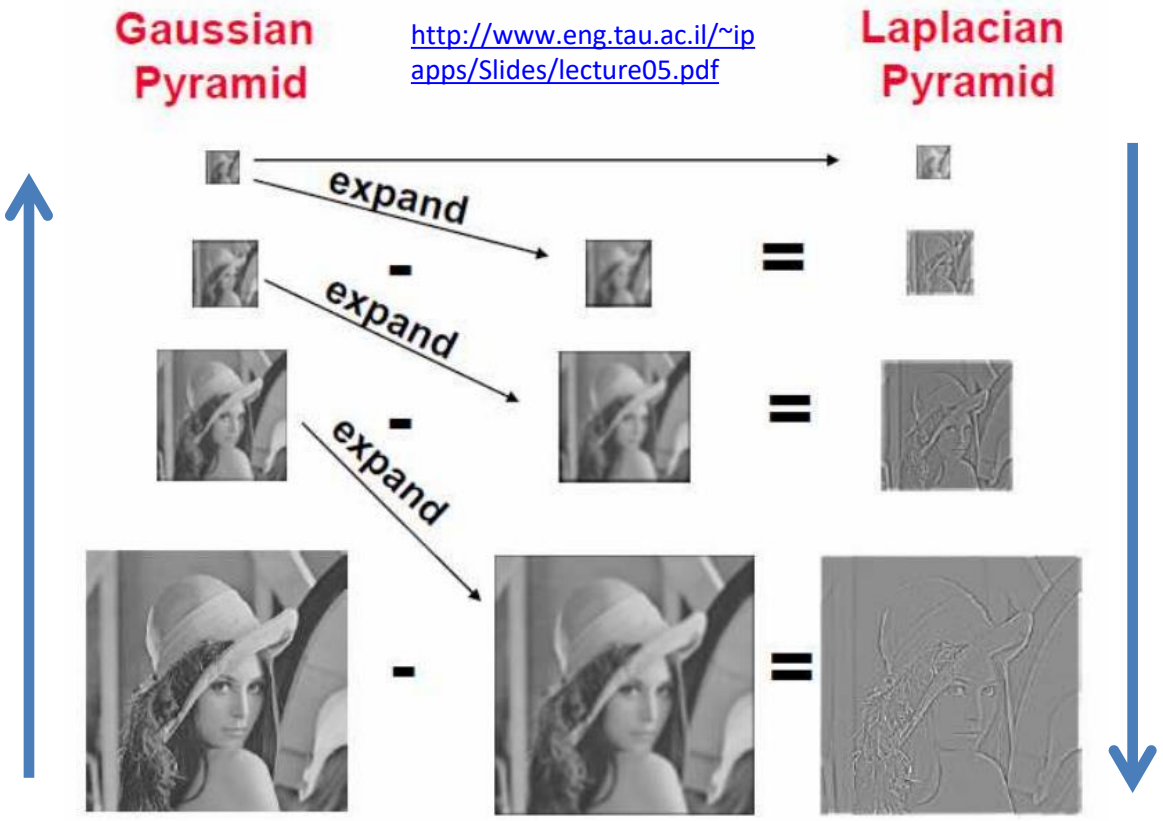
Sin suavizado Gaussiano. Solo submuestreando, quedándonos con las filas/columnas pares:



Incluyendo suavizado Gaussiano:



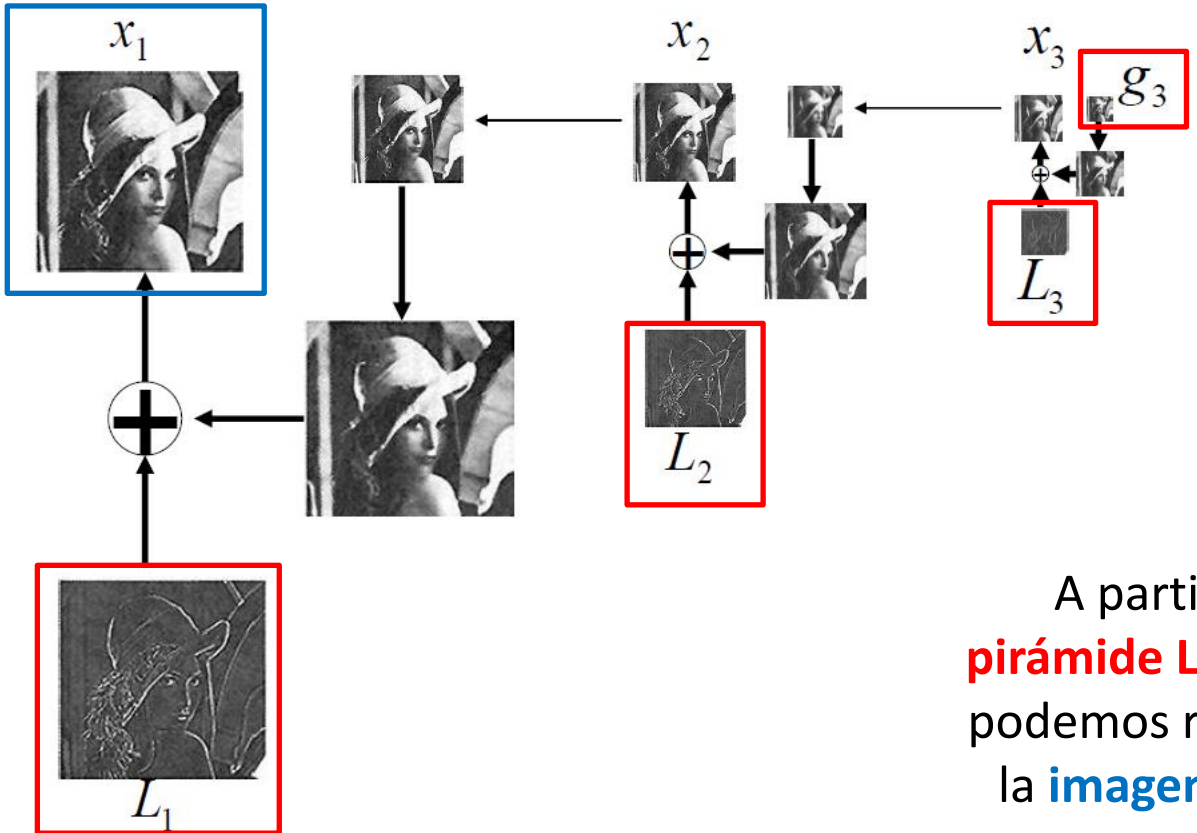
EJERCICIO 2.B



1) se parte de la imagen más pequeña producida por la Gaussian Pyramid.
2) se expande, se calcula la diferencia, y tenemos el nivel L_i.
3) Pasamos al siguiente nivel de la pirámide Gaussiana, expandimos y calculamos la diferencia. Y ya tenemos el nivel L_{i-1} de la Laplacian Pyramid.
4) Y así sucesivamente.

EJERCICIO 2.C

- Reconstruimos x_1 a partir de L_1, L_2, L_3 y g_3



A partir de la **pirámide Laplaciana**, podemos reconstruir la **imagen original**

EJERCICIO 2.C

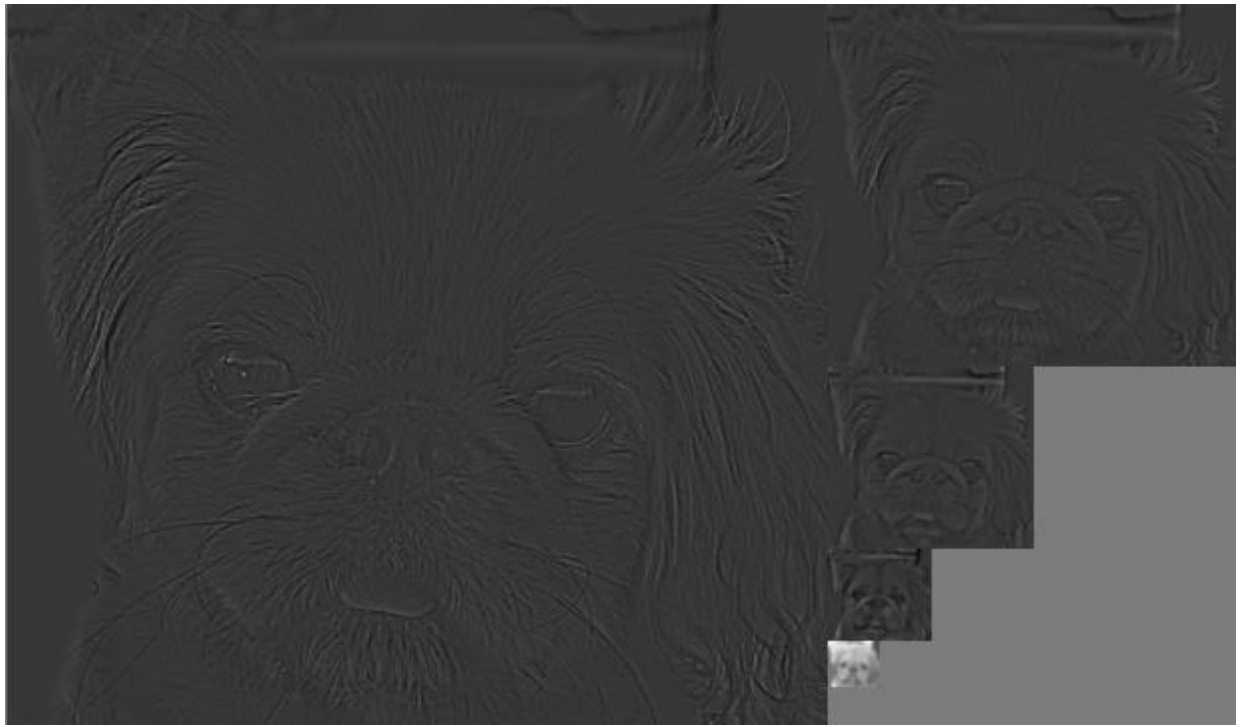
- ¿Es posible reconstruir perfectamente la imagen original a partir de una pirámide Laplaciana?
  ¡Sí!
- ¿De qué depende dicha reconstrucción perfecta: del sigma, de la interpolación,…?
  ¡Principalmente, de que lo programéis bien! ☺

EJERCICIO 2.C

Pirámide Gaussiana con sigma=1 y 4 niveles

EJERCICIO 2.C

Pirámide Laplaciana de 4 niveles



EJERCICIO 2.C

Imagen Original VS Imagen Reconstruida

Idénticas! Todos los píxeles son iguales!!

EJERCICIO 2.C

- Posibles errores:
  - No usar el **mismo tipo de interpolación para calcular la pirámide Laplaciana y para reconstruir la imagen.**
  - Si el *subsampling* no es adecuado también podría dar problemas en la reconstrucción.
    - Por ejemplo, si reducís y expandís con distintos tamaños.
    - Por defecto, quedaos siempre con las filas/columnas pares.

EJERCICIO 2.C

- Nota:
  - El sigma no influye en la reconstrucción!

Pirámide Gaussiana (Sigma=10)

Pirámide Laplaciana (Sigma=10)

EJERCICIO 2.C

- Pero… ¿esto cómo es posible? ¿Brujería?

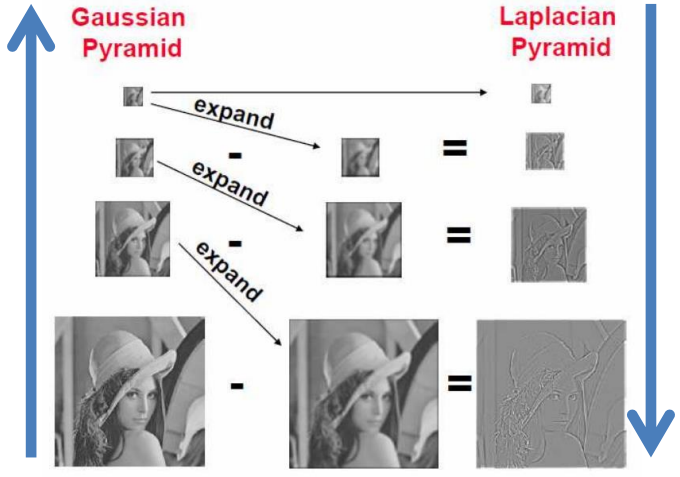
Teorema de muestreo de Nyquist-Shannon: tras el suavizado, hay píxeles redundantes. Por tanto, podemos descartarlos (por medio de submuestreo) sin perder información

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

Disponemos de una versión reducida de la imagen original, y todos los detalles (altas frecuencias) a distintas escalas.
De modo que tenemos toda la información necesaria.

EJERCICIO 2

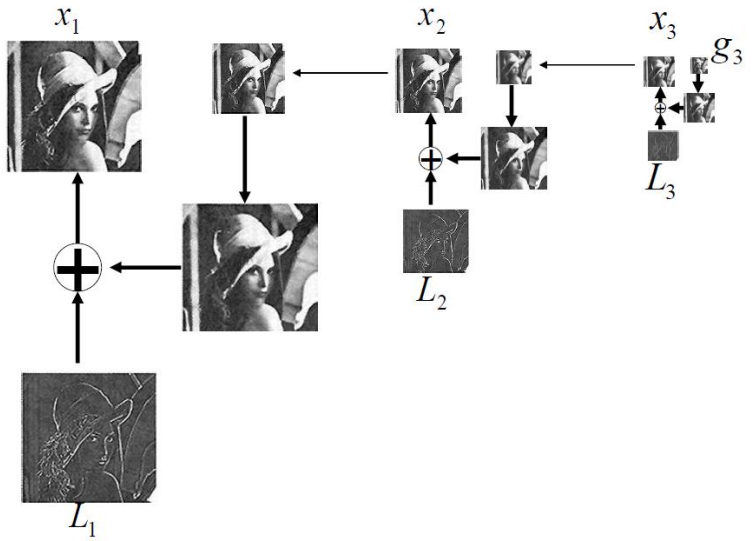
```
pyramidGauss(im, sizeMask, nlevel):
    # Cálculo de la máscara Gaussiana

    piramide = []
    # Guardamos la imagen original como base de la pirámide
    piramide.append(im)
    for i in range(nlevel):
        # Creamos un nuevo nivel alisando el anterior
        # Y quedándonos con la mitad de las filas y de las columnas

    return vim #lista de imágenes
```




```
pyramidLap(im, sizeMask, nlevel):
    # Calcular pirámide Gaussiana de la imagen

    for i in range(nlevel):
        # Poner el nivel i+1 de la pirámide Gaussiana en el tamaño de i (cv2.resize) usando interpolación bilineal
        # Calcular piramide_gauss[i] - imagen_expandida

    # Guardamos el último nivel de la piramide Gaussiana para poder reconstruir la imagen original

    return vimL #lista de imágenes
```

EJERCICIO 2

```
reconstructLap(pyL, flagInterp):
    # Partimos del último nivel de la pirámide Laplaciana de la imagen

    # Recorremos la pirámide desde el nivel más alto aplicando el algoritmo:
          # Expandir el nivel al tamaño del nivel anterior (cv2.resize) usando la interpolación
          determinada por flagInterp (bilineal)
          # Calcular piramide_laplaciana[i] + imagen_expandida

    return im #imagen_reconstruida
```



EJERCICIO 3

- Trabajaremos con un paper:
  - A. Oliva, A. Torralba, P.G. Schyns (2006). Hybrid Images. ACM Transactions on Graphics.
  - http://olivalab.mit.edu/hybridimage.htm

- **Mezclando adecuadamente una parte de las frecuencias altas de una imagen con una parte de las frecuencias bajas de otra imagen, obtenemos una imagen híbrida que admite distintas interpretaciones a distintas distancias**.

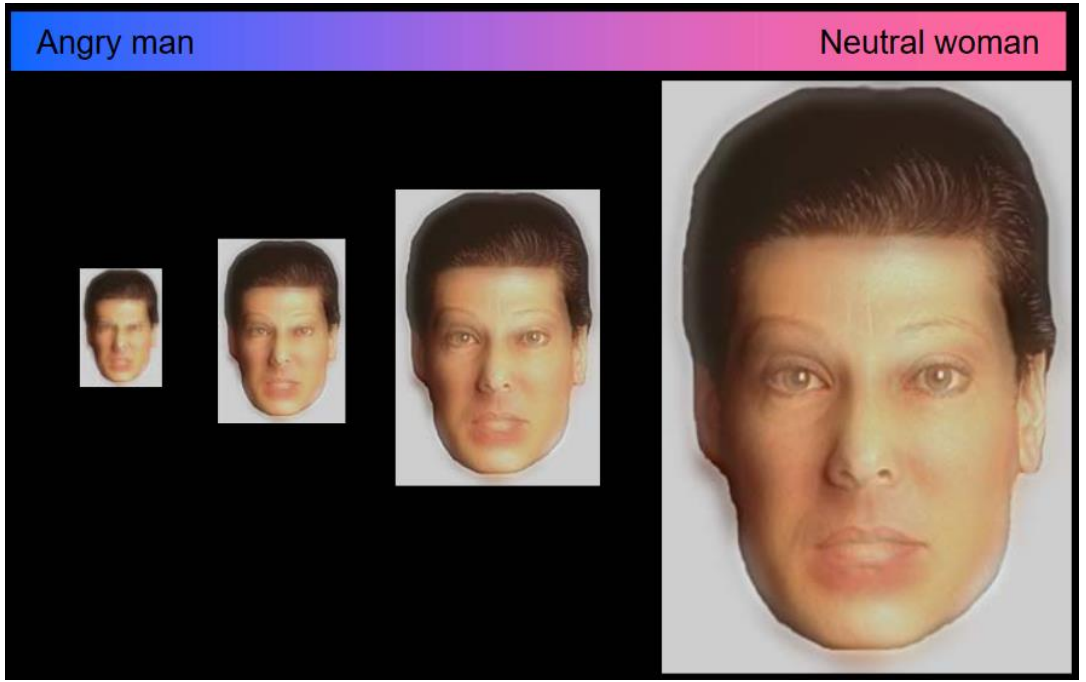


Imagen extraída de
http://olivalab.mit.edu/publications/Talk_Hybrid_Siggraph06.pdf

EJERCICIO 3

- Para seleccionar la parte de frecuencias altas y bajas usaremos el parámetro sigma del kernel Gaussiano.
  - A mayor valor de sigma, mayor eliminación de altas frecuencias en la imagen convolucionada.
  - A veces es necesario elegir dicho valor de forma separada para cada una de las dos imágenes.

EJERCICIO 3

- **Se pide implementar una función que genere las imágenes de baja y alta frecuencia a partir de las parejas de imágenes.**

Las frecuencias bajas predominan a largas distancias, mientras que las altas predominan a cortas

Imagen de entrada y alta frecuencia

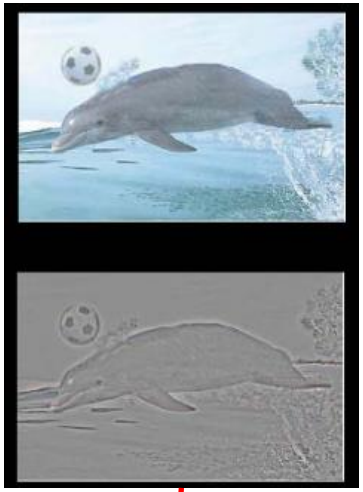
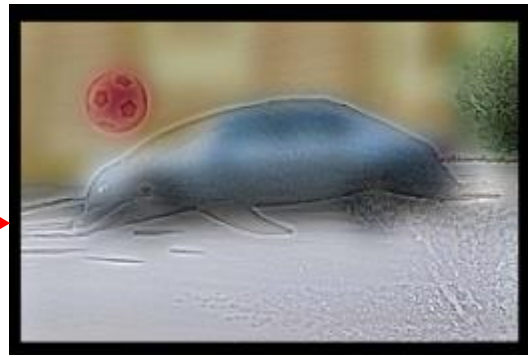


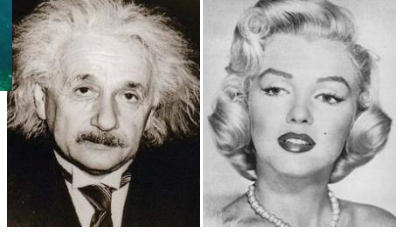
Imagen híbrida

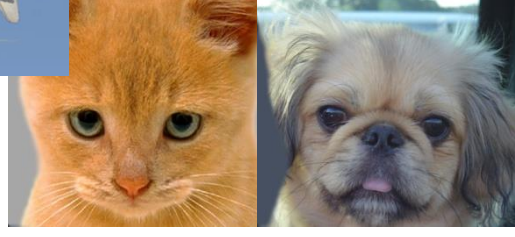
Imagen de entrada y baja frecuencia



EJERCICIO 3

- **Si el efecto está conseguido se observa en la pirámide Gaussiana**
  - No es necesario que os alejéis del ordenador para verlo!
- Si el efecto no está conseguido se considera que el ejercicio no se ha resuelto correctamente
  - P.ej., si una figura predomina claramente siempre sobre la otra
- Elección de imágenes de altas y bajas frecuencias:
  - Alta: aquella que presenta bordes más acentuados
  - Baja: aquella que presenta un aspecto más suavizado



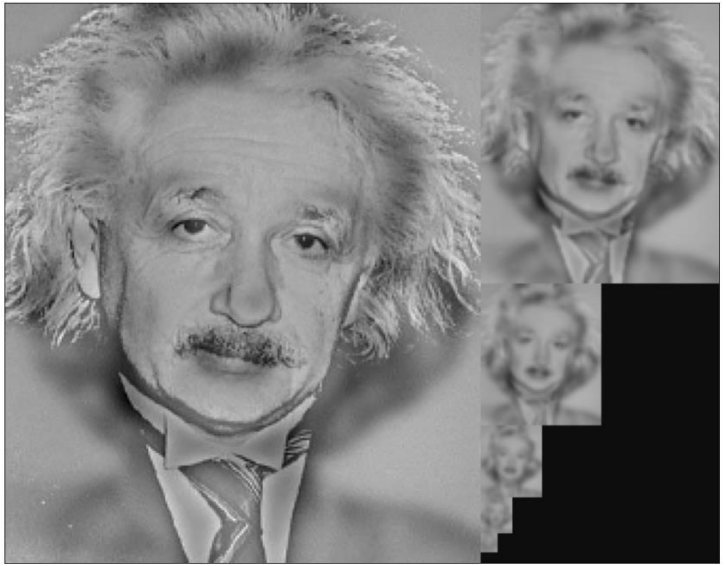


EJERCICIO 3

- Posibilidad sobre cómo proceder:

1) Frecuencias bajas: alisarla fuertemente (hasta que solo quede casi una mancha sin detalles).

2) Frecuencias altas: calcular la diferencia entre la original y su versión alisada.



EJERCICIO 4

- Pyramid Blending

vs

EJERCICIO 4

- Pyramid Blending

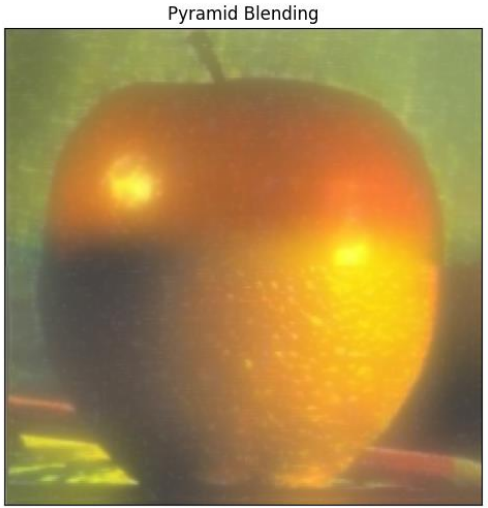
1) Leer dos imágenes a color (*imagen1* e *imagen2*)

2) Generar la pirámide Laplaciana de imagen1 (*lp_imagen1*)

3) Generar la pirámide Laplaciana de imagen2 (*lp_imagen2*)

4) Mezclar ambas laplacianas (directamente, tomar la mitad de las filas/columnas de *lp_imagen1* y la otra mitad de *lp_imagen2*)

5) Partir de esta nueva pirámide mezclada/fusionada, y realizar la reconstrucción de la imagen "original"



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

Nota: si queréis, en este ejercicio sí podéis emplear *pyrUp* y *pyrDown*

Útil referencia: https://becominghuman.ai/image-blending-using-laplacian-pyramids-2f8e9982077f

Notas finales

- Acordaos de
  - consultar la ayuda:
    - https://docs.opencv.org/4.6.0/d4/d86/group__imgproc__filter.html
  - trabajar la discusión de resultados y explicación de las decisiones tomadas, para que todo lo que hacéis quede claro y bien justificado.

Prácticas de Visión por Computador

Práctica 1:
Filtrado de Imágenes

Pablo Mesejo y Víctor Vargas
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA