Prácticas de Visión por Computador
Grupo 2

Trabajo 1 – Más comentarios sobre: Convolución y Derivadas

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

UNIVERSIDAD DE GRANADA

Float vs Int

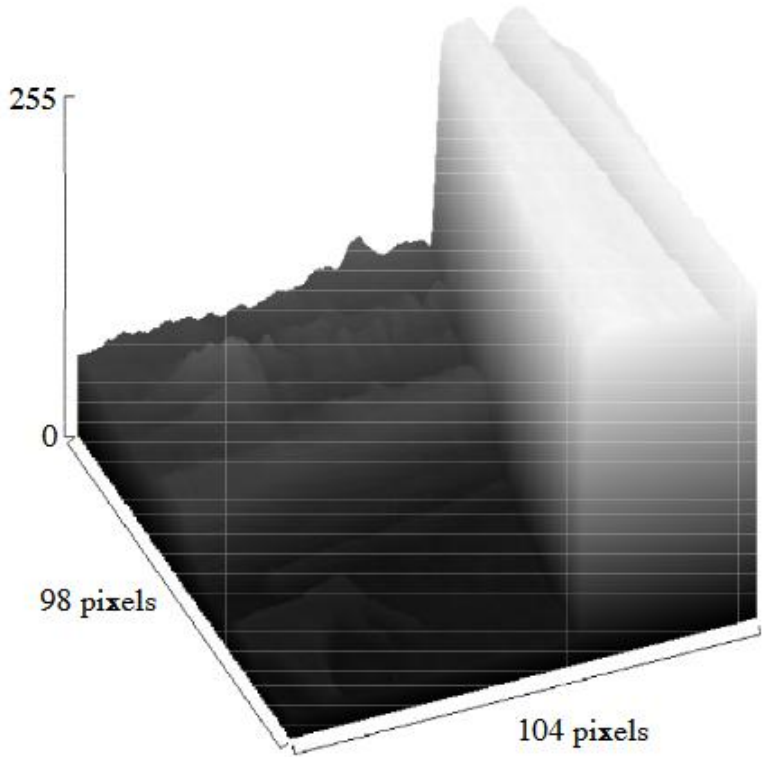
```
filename = 'data/motorcycle.bmp'
im = read_image(filename,'gray',0).astype(np.float64)
GaussianKernel = kernelGauss1D(1,None,0)
image = insert_padding(im,GaussianKernel,2)

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)
```

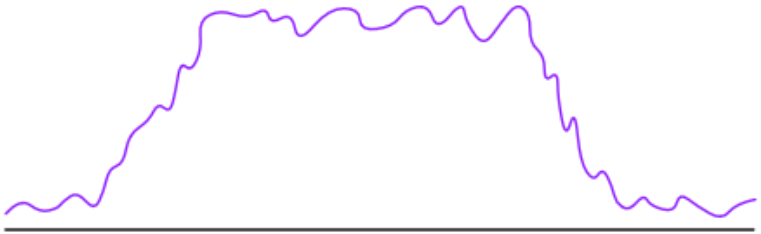
Números de píxeles diferentes en ambas imágenes (con una diferencia superior a 1 nivel de gris): **8.117%** (11400 de 140454)

Visualización 3D de un borde

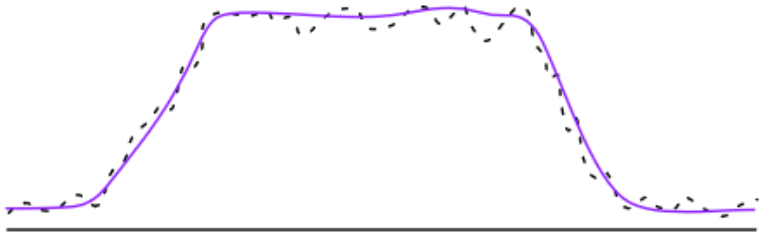


Ejemplo extraído de
https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/pdfs/D02-Images.pdf

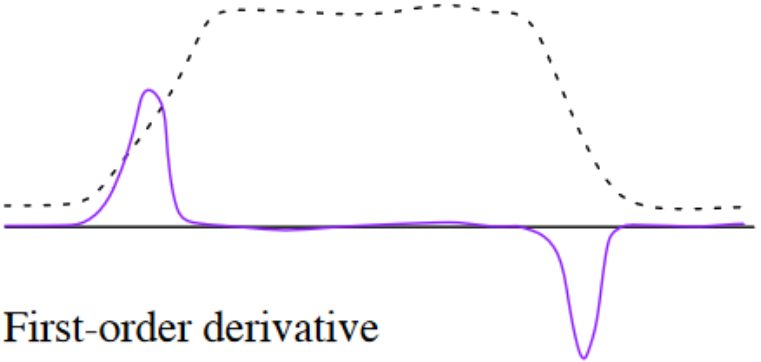
Más sobre bordes…



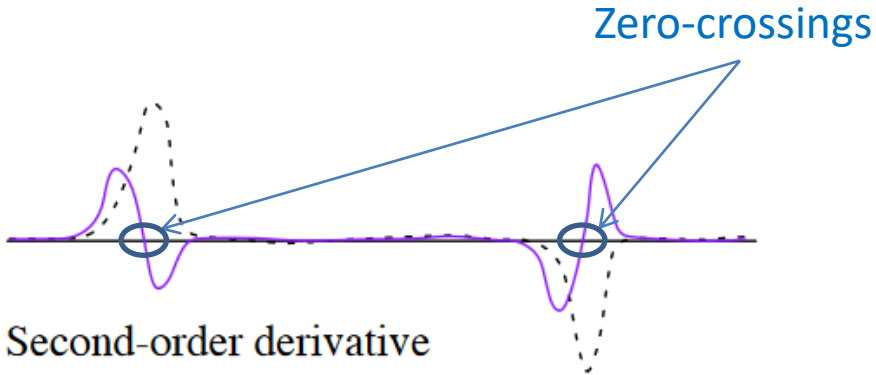
Intensity profile of an input image



After noise removal



First-order derivative



Second-order derivative

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

Bordes y Gradientes

Derivada horizontal

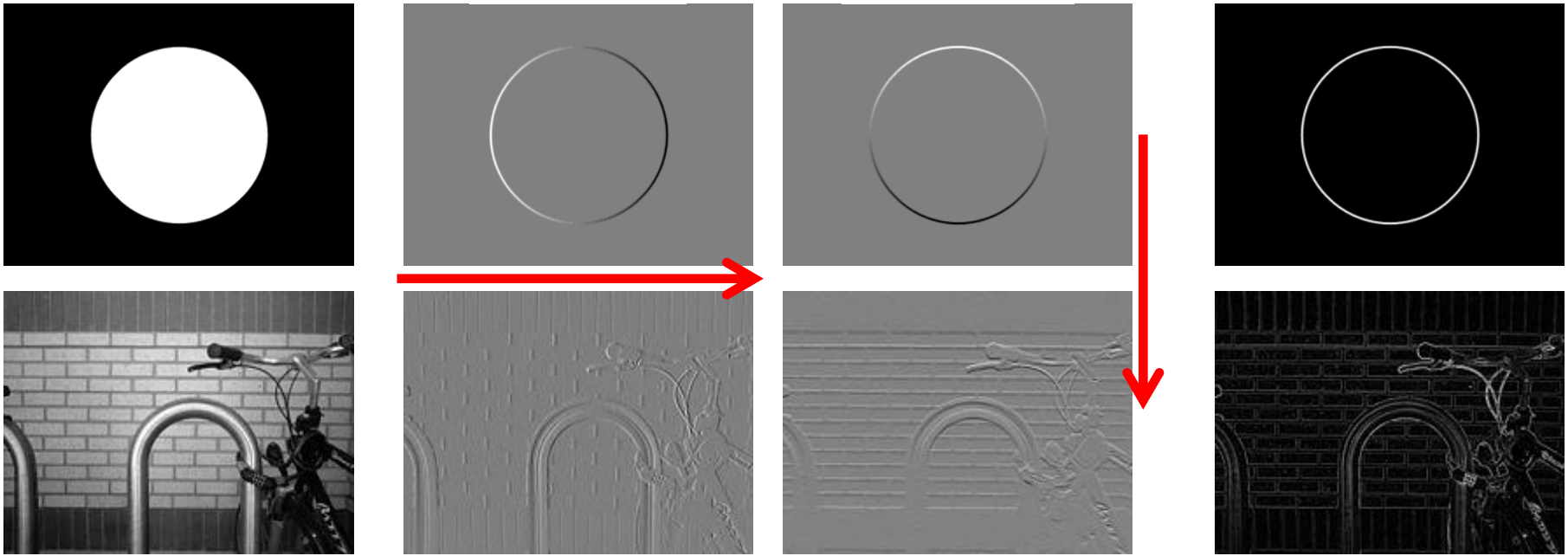
$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

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

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*

Outer Product

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$

```
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)
```

EJERCICIO 1.C

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 y hacer, primero, la convolución por filas y, luego, sobre el resultado, aplicar la convolución del kernel 1D por columnas**.

EJERCICIO 1.C

Pero… ¿cómo **implementar la convolución de una imagen con una máscara 2D de forma eficiente**?

EJERCICIO 1.C

Veamos algunas posibilidades… Evidentemente, hay otras…

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]
```

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

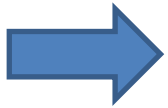
EJERCICIO 1.C

OPCIÓN 1:

```
[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]
```

Podemos hacerlo mucho mejor!!!

No es necesario recorrer siempre todas las filas y columnas!!!

EJERCICIO 1.C

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]
```

OPCIÓN 2:

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

```
[  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]
```

T

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

T

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

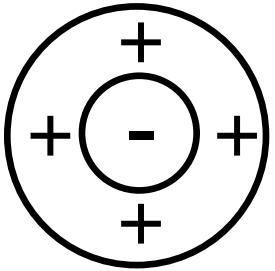
```
[  0,   0,   0,   0,   0]
[  0, 210, 360, 330,   0]
[  0, 520, 800, 680,   0]
[  0, 570, 840, 690,   0]
[  0,   0,   0,   0,   0]
```

Mismo resultado de antes! Y solo recorriendo, en cada convolución 1D, una vez las filas!!!

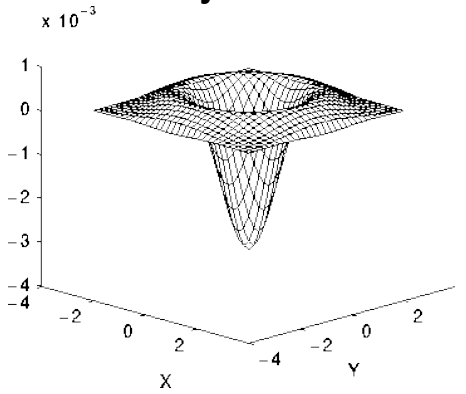
EJERCICIO 1.D

- 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.D

- ¿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:

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

→ Son vectores linealmente independientes!!

EJERCICIO 1.D

```
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(kernelGauss1D(1, None, 0), kernelGauss1D(1, None, 2)))
1
```

Si el rango no es 1 → no es 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.D

- 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.D

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.D

¿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.D

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

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.D

- ¿Qué tenemos que hacer en el ejercicio entonces?
  - Mostrar los kernels 1D que usáis en las convoluciones 1D de la LoG
  - Mostrar el kernel 2D equivalente
  - Mostrar ejemplos visuales del uso de la LoG

EJERCICIO 2

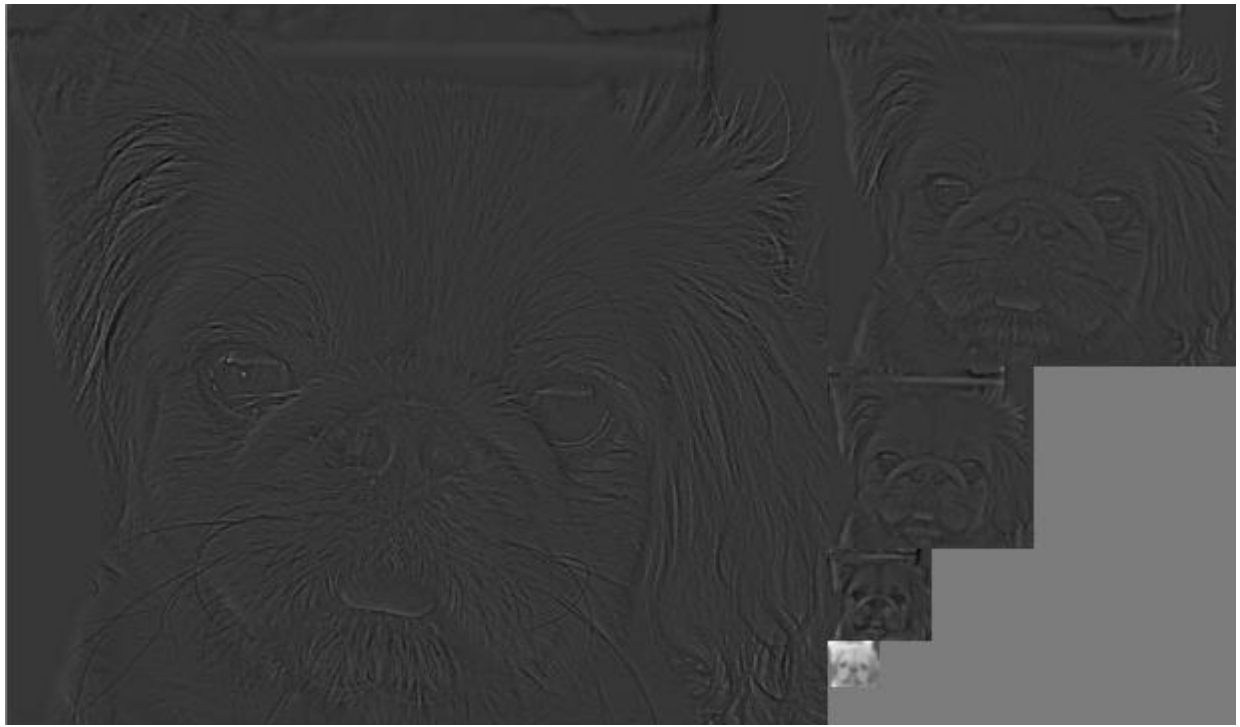
- ¿Es posible reconstruir perfectamente la imagen original a partir de una pirámide Laplaciana? **¡Sí!**
- ¿De qué depende dicha reconstrucción perfecta: del sigma, de la interpolación,…? **¡De que lo programéis bien!** ☺

EJERCICIO 2

Pirámide Gaussiana con sigma=1 y 4 niveles

EJERCICIO 2

Pirámide Laplaciana de 4 niveles



EJERCICIO 2

Imagen Original VS Imagen Reconstruida

Idénticas! Todos los píxeles son iguales!!

EJERCICIO 2

- Posibles errores:
  - No usar el **mismo tipo de interpolación para calcular la pirámide Laplaciana y para reconstruir la imagen**
  - Dependiendo del tipo de borde que se use para el *padding* la reconstrucción puede no ser exacta al 100%
  - 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

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

Pirámide Gaussiana (Sigma=10)

Pirámide Laplaciana (Sigma=10)

EJERCICIO 2

- 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

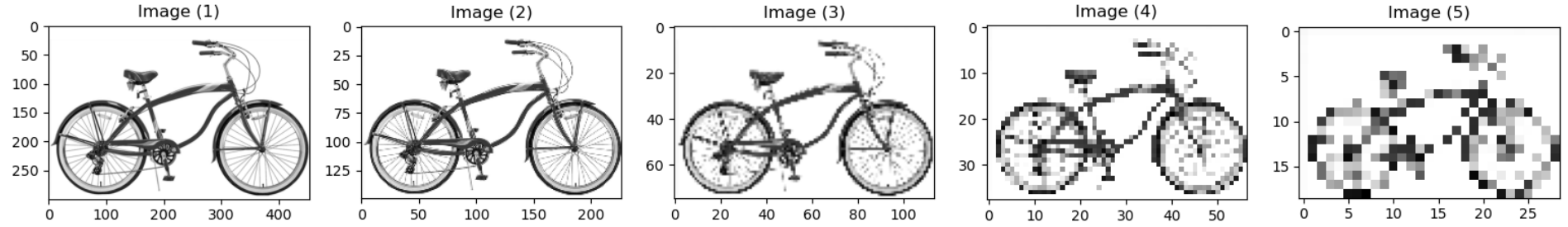
S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

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.

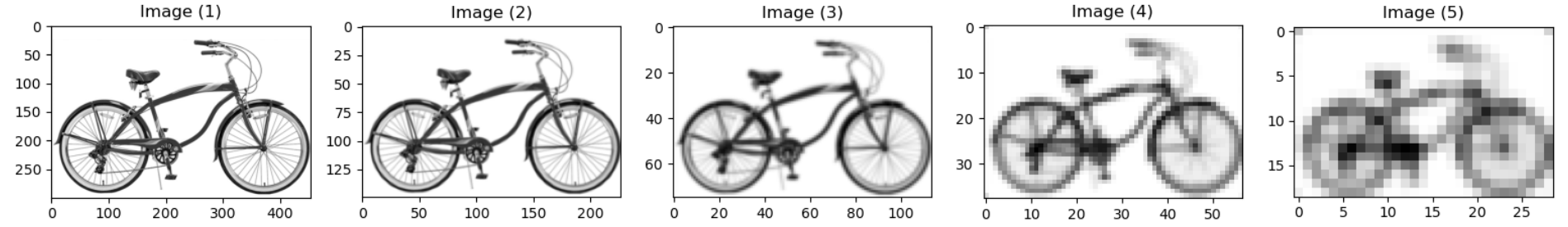
EJERCICIO 2

- Efecto del suavizado a la hora de crear la pirámide

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

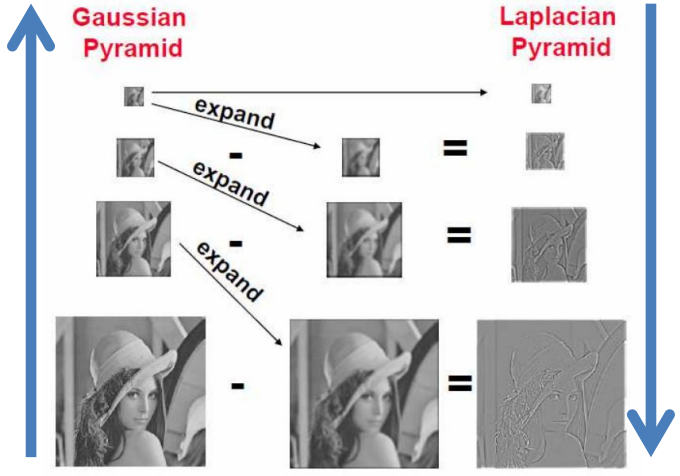


Incluyendo suavizado Gaussiano:



EJERCICIO 2

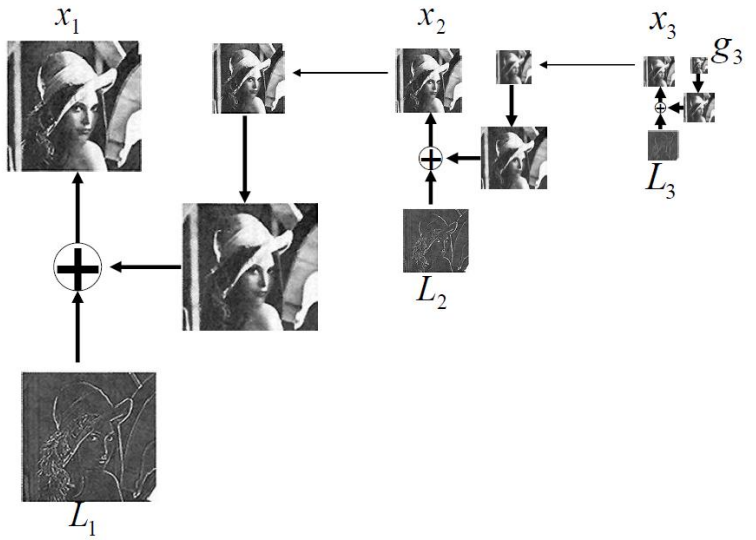
```
piramide_gaussiana(img, sigma, niveles):
    # Cálculo de la máscara Gaussiana

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

    return np.array(piramide)
```




```
piramide_laplaciana(img, sigma, niveles):
    # Calcular pirámide Gaussiana de la imagen

    for i in range(niveles):
        # 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 np.array(piramide_laplaciana)
```

EJERCICIO 2

```
reconstruir_imagen(piramide_laplaciana):
    # 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 interpolación bilineal
          # Calcular piramide_laplaciana[i] + imagen_expandida)

    return imagen_reconstruida
```



Prácticas de Visión por Computador
Grupo 2

Trabajo 1 – Más comentarios sobre: Convolución y Derivadas

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

UNIVERSIDAD DE GRANADA