Prácticas de Visión por Computador
Grupo 2

Trabajo 1:
Convolución y Derivadas

Pablo Mesejo
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 de la Práctica 1

Normas de la Entrega de Prácticas

- Dos opciones:
  - Se entrega memoria (PDF) y código (.py o .ipynb) → ZIP/RAR
  - Se entrega solamente un fichero .ipynb integrando directamente el análisis y discusión

Normas de la Entrega de Prácticas

- Solo se entrega memoria y código fuente → no imágenes!
  - Excepto en el Bonus! Para el bonus podéis usar dos imágenes que os gusten y las incluís en el ZIP/RAR.
- Lectura de imágenes o cualquier fichero de entrada: “imagenes/nombre_fichero”
- Todos los resultados numéricos serán mostrados por pantalla. No escribir nada en el disco.
- La práctica deberá poder ser ejecutada de principio a fin sin necesidad de ninguna selección de opciones.
- Puntos de parada para mostrar imágenes, o datos por terminal.

Entrega

- Fecha límite: 30 de Octubre
- Valoración: 8 puntos (+4)
- Lugar de entrega: PRADO

- **Se valorará mucho la memoria/discusión.**

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.

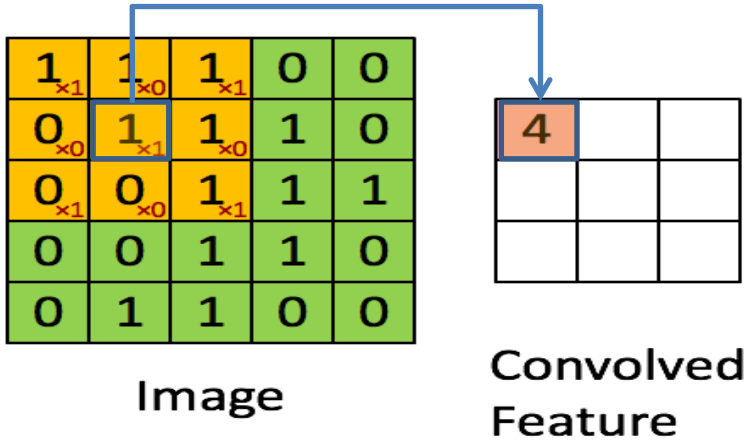
- Técnicamente, una convolución es una correlación cruzada (*cross-correlation*) en donde el filtro/máscara se rota 180 grados.
  - A diferencia de la correlación, la convolución verifica la propiedad asociativa, lo que nos permite construir filtros complejos a partir de filtros simples.
  - Para imágenes reales y kernels simétricos
    - Convolución = Cross-correlation
  - Si el kernel es antisimétrico (p.ej. [-1 0 1]), solo cambia el signo

¿Qué es una convolución?

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

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

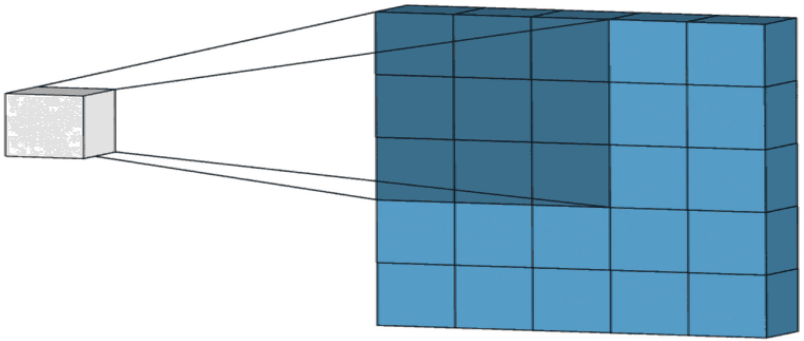
Se multiplica elemento-a-elemento el filtro con la imagen, se suman los productos, y se sustituye la posición central del filtro en la imagen.



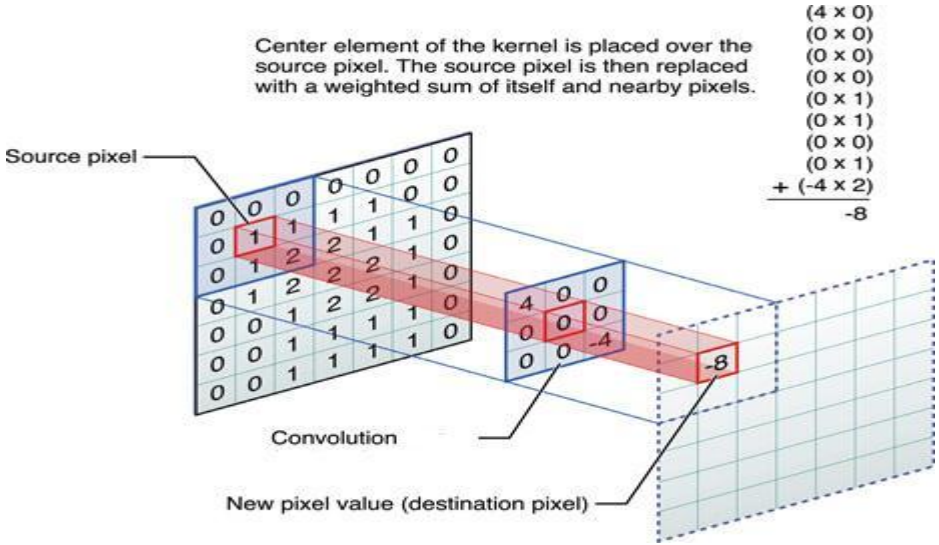
Imágenes extraídas de https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/

¿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?

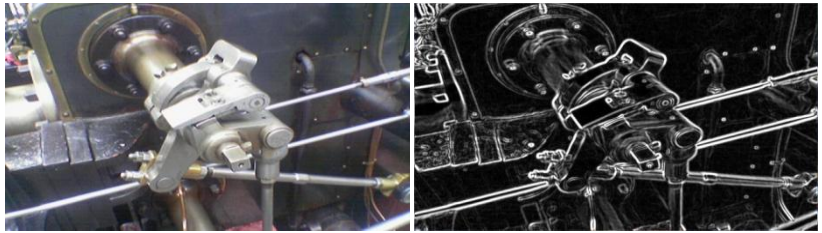
- **Operación lineal a nivel local** con una máscara.
  - Los valores de la máscara determinan el resultado (características que se extraen)

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)

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



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  h̃ ← ⌊(h − 1)/2⌋
2  w̃ ← ⌊(w − 1)/2⌋
3  for y ← 0 to height −1 do
4        for x ← 0 to width −1 do
5              val ← 0
6              for j ← 0 to h − 1 do
7                    for i ← 0 to w − 1 do
8                          val ← val +g(i, j) ∗ f(x + w̃ − i, y + h̃ − j)
9              h(x, y) ← val
10 return h
```

- 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: (1/3) * [1 1 1]

Filtro paso bajo (Gaussian)

Derivada/diferenciación

$$\sum g_i = 0$$

(derivar una función constante debería devolver 0)

Ejemplo: (1/2) * [-1 0 1]

Filtro paso alto (derivative of Gaussian)

… filtros paso banda (Laplacian 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:
  - “USANDO SOLO 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*, *sepFilter2D* o *filter2D*. 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.

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.

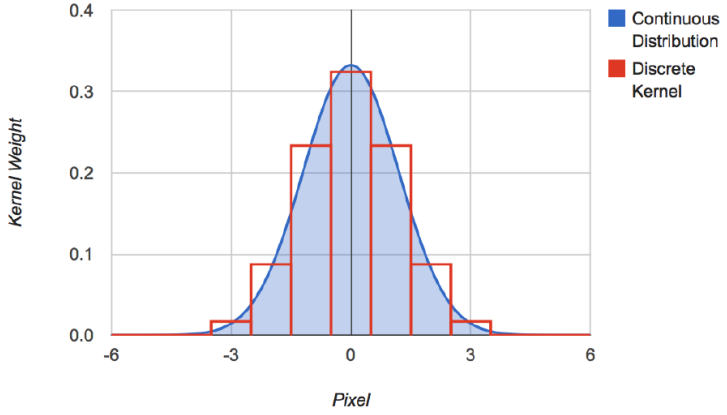
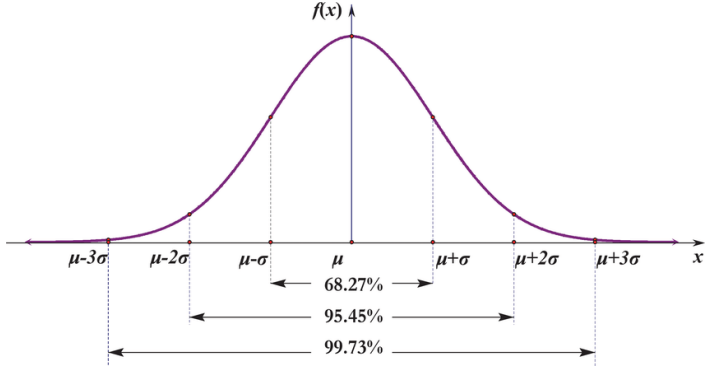


Imagen extraída de
http://dev.theomader.com/gaussian-kernel-calculator/

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 los valores

EJERCICIO 1.A

Debéis crear una función que:

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

2·[3·σ] + 1 = T, siendo T el tamaño de la máscara (T = 2k+1)

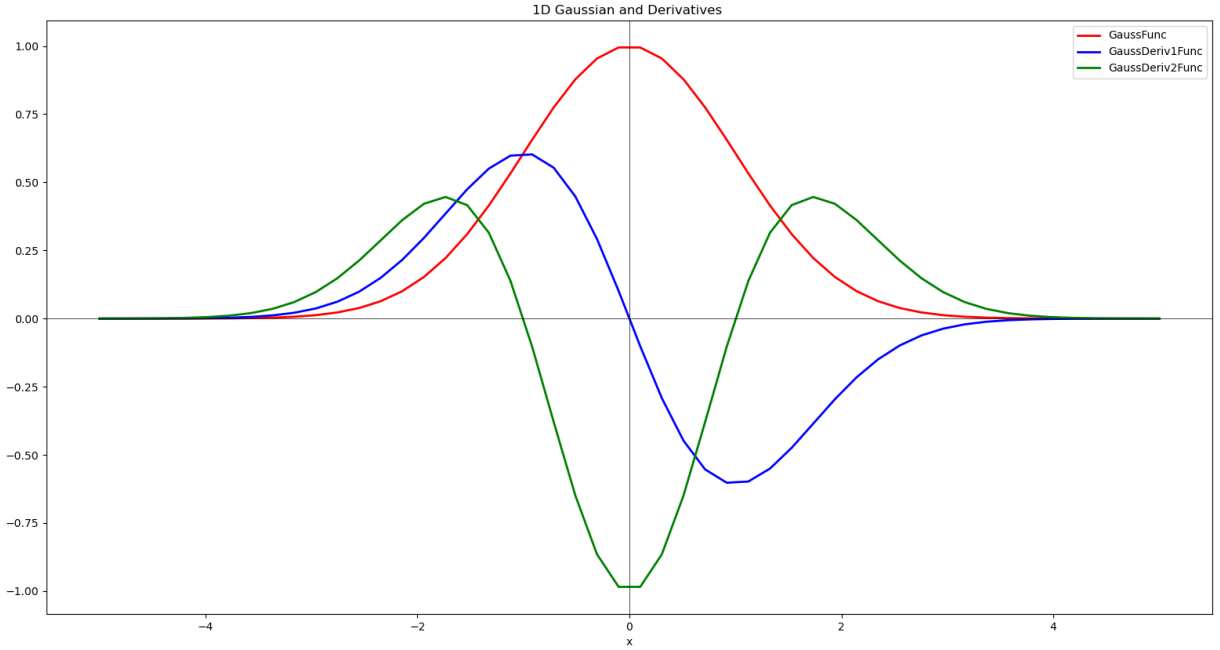
Si se proporciona, por ejemplo, T=5:

• 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



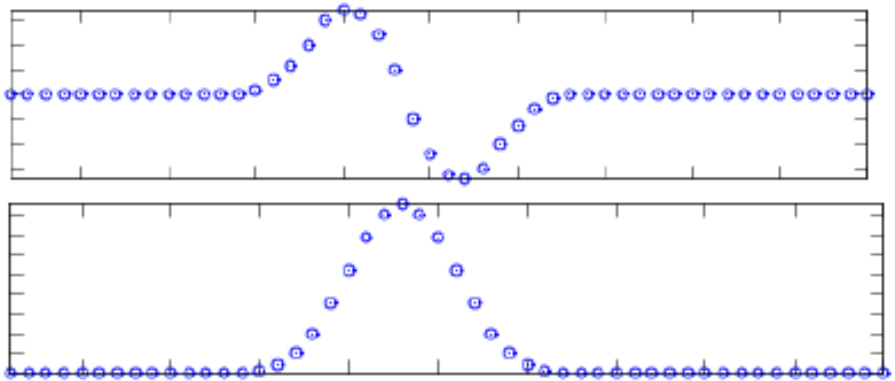
EJERCICIO 1.A

- 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 la máscara de la primera derivada suman cero porque son los mismos valores repetidos, pero con signo distinto.
    - Os puede resultar de utilidad para verificar que habéis creado correctamente la máscara.
    - Así como para ser conscientes de que, en el caso de esta máscara, no debéis dividirla 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 en ejes cartesianos las máscaras como funciones 1D y compare sus formas con las máscaras dadas por la función de OpenCV *getDerivKernels***

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 *getDerivKernel* que proporcionan filtros de Scharr o Sobel. Ambas sirven para detectar bordes, pero los valores son diferentes

EJERCICIO 1.A

- ¿Cómo usar cv.getDerivKernels?
  - Revisar documentación: https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga6d6c23f7bd3f5836c31cfae994fc4aea
  - cv.getDerivKernels(dx, dy, ksize)

Derivative order in respect of x

Derivative order in respect of 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

Calcule las máscaras discretas 1D de longitud 5 y 7 tanto de alisamiento como de derivada de primer orden generadas a partir de la aproximación binomial de la Gaussiana y la máscara de derivada de longitud 3.

a	$w = 2a + 1$	binomial kernel	$\sigma^2 = \frac{a}{2}$	σ	$\alpha = \frac{w}{2\sigma}$	area
1	3	$\frac{1}{4}[1 \quad 2 \quad 1]$	0.5	0.71	2.12	96.60%
2	5	$\frac{1}{16}[1 \quad 4 \quad 6 \quad 4 \quad 1]$	1.0	1.0	2.50	98.76%
3	7	$\frac{1}{64}[1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1]$	1.5	1.22	2.86	99.58%
4	9	$\frac{1}{256}[1 \quad 8 \quad 28 \quad 56 \quad 70 \quad 56 \quad 28 \quad 8 \quad 1]$	2.0	1.41	3.18	99.85%

EJERCICIO 1.B

Ejecute la función *cv2.getDerivKernels(0,1,9)*. Observe los vectores de salida y compárelos con los previamente calculados. ¿Qué relación hay? ¿Qué conclusiones extrae sobre la aproximación de OpenCV al cálculo de las máscaras de derivadas de primer orden?

- *cv2.getDerivKernels(0,1,9)* proporciona un kernel de alisamiento y uno de derivada.
- La aproximación binomial de tamaño 9 es, tal cual, el primer vector que proporciona *cv2.getDerivKernels(0,1,9)* $[1 \quad 8 \quad 28 \quad 56 \quad 70 \quad 56 \quad 28 \quad 8 \quad 1]$
- ¿Cómo podemos obtenerlo a partir de los kernels anteriormente calculados?

EJERCICIO 1.C

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

- No podéis usar *filter2D* ni *numpy.convolve*. Debéis implementar vosotros la convolución 2D a partir de filtros 1D.

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**.

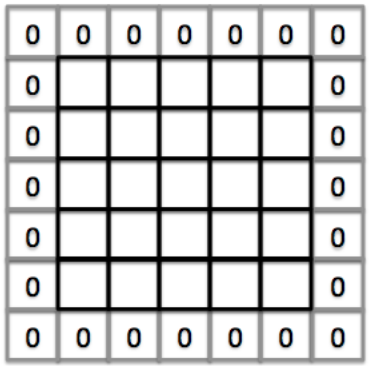
EJERCICIO 1.C

- Se trata de crear una función que, dada una señal de entrada (en nuestro caso, una imagen) y un kernel, proporciona una imagen convolucionada.
  - El tema de rellenar bordes, conocido también como *padding*, lo consideramos un tema de preprocesado.
    - Es decir, el tipo de *padding* no es algo que le interese a nuestra función, que va a recorrer la imagen entera ajustándose perfectamente a los bordes de la misma.

EJERCICIO 1.C

- *Padding*:
  - Types (OpenCV):
    - * *BORDER_REPLICATE: aaaaaa | abcdefgh | hhhhhhh*
    - * *BORDER_REFLECT: fedcba | abcdefgh | hgfedcb*
    - * *BORDER_REFLECT_101: gfedcb | abcdefgh| gfedcba*
    - * *BORDER_WRAP: cdefgh | abcdefgh | abcdefg*
    - * *BORDER_CONSTANT: iiiiii | abcdefgh | iiiiiii*

Image



EJERCICIO 1.C

- Notas importantes:
  - Menos el bonus, 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
  - Convertid la imagen de entrada a flotante en primer lugar, para evitar problemas de cálculo y truncamiento de valores

- Enlace interesante a nivel práctico: http://www.songho.ca/dsp/convolution/convolution2d_separable.html
  - Sirve, tanto para comprender mejor la convolución 2D a partir de kernels 1D, como para tener un ejemplo sencillo con el que depurar vuestro código, si es necesario.

EJERCICIO 1.C

- La idea es que reutilicéis, en la medida de lo posible, las funciones que ya creasteis para la P0
  - Por ejemplo, se pueden emplear una serie de funciones auxiliares para
    - leer imágenes,
    - visualizar imágenes,
    - reescalar intensidades (al modo en que lo hacíamos en el ejercicio 2 de la P0),
    - comparar imágenes (y ver si son idénticas o no, y si son diferentes cuánto se diferencian),
    - funciones para incluir padding o eliminarlo,
    - Etc.

EJERCICIO 1.C

- Nota importante:
  - en toda la práctica, no es necesario calcular la descomposición en valores singulares
  - en nuestro caso, todo es Gaussiano y, por tanto, directamente descomponible.
  - Lo comento para que lo tengáis claro y no sobre-compliquéis las cosas.

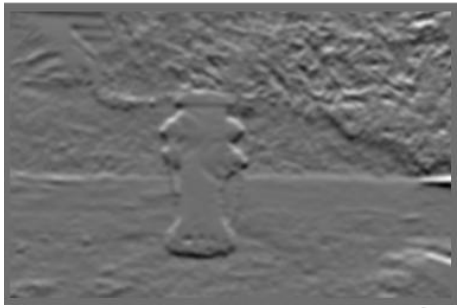
EJERCICIO 1.C.a

- **Comparar 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
  - que no os extrañe si vuestro resultado de la convolución no es exactamente igual al proporcionado por *cv2.GaussianBlur*.
  - Seguramente vuestro código será “más correcto”, pero más lento.

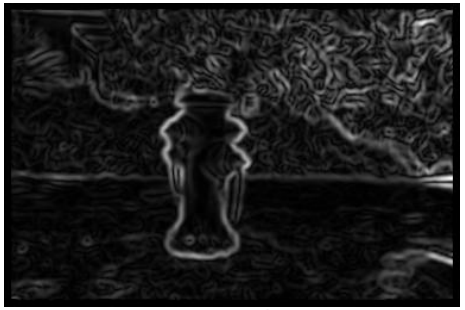
EJERCICIO 1.C.b

- **Usando las máscaras del ejercicio 1.A, calcular la derivada de una imagen dada**

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



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



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

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

EJERCICIO 1.C

- Ideas:
  - Podéis implementar a mano las convoluciones 2D y 1D, y ver cuánto ahorráis en tiempo
  - Medid explícitamente la diferencia a nivel de píxel.
    - Por ejemplo, diferencia mediana a nivel de valores de gris entre *cv.GaussianBlur* y vuestra implementación

EJERCICIO 1.D

- **Calcular máscaras normalizadas de la Laplaciana de una Gaussiana**
  - La Laplaciana de la Gaussiana es la suma de las dos segundas derivadas.

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

(Laplacian)

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

EJERCICIO 1.D

- **Calcular máscaras normalizadas de la Laplaciana de una Gaussiana**

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

(Laplacian)

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 sigma^2

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, filtros Sobel/Scharr, 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**

NOTA: se permiten utilizar funciones como *cv2.pyrUp()* o *cv2.pyrDown()*, pero quien resuelva el ejercicio con sus propias funciones de bajo nivel será recompensado a la hora de la evaluación.

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.

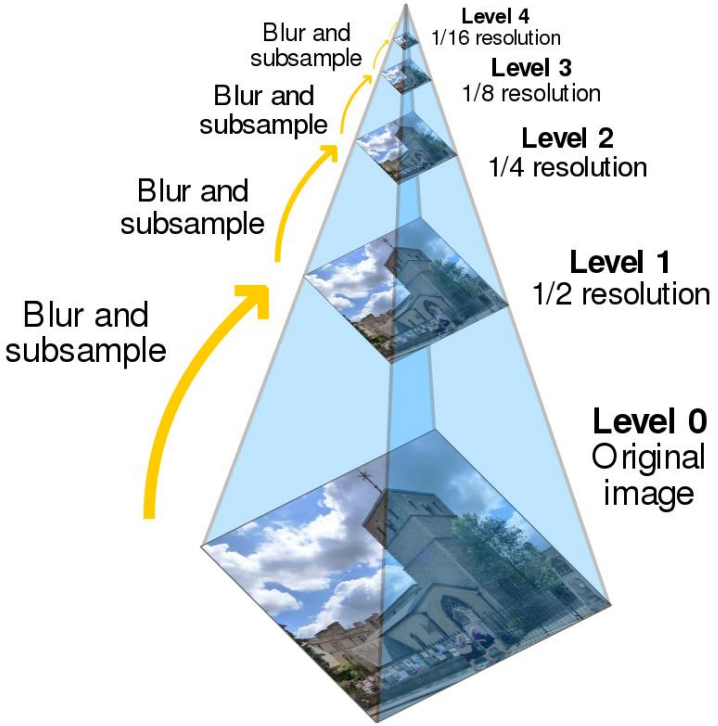
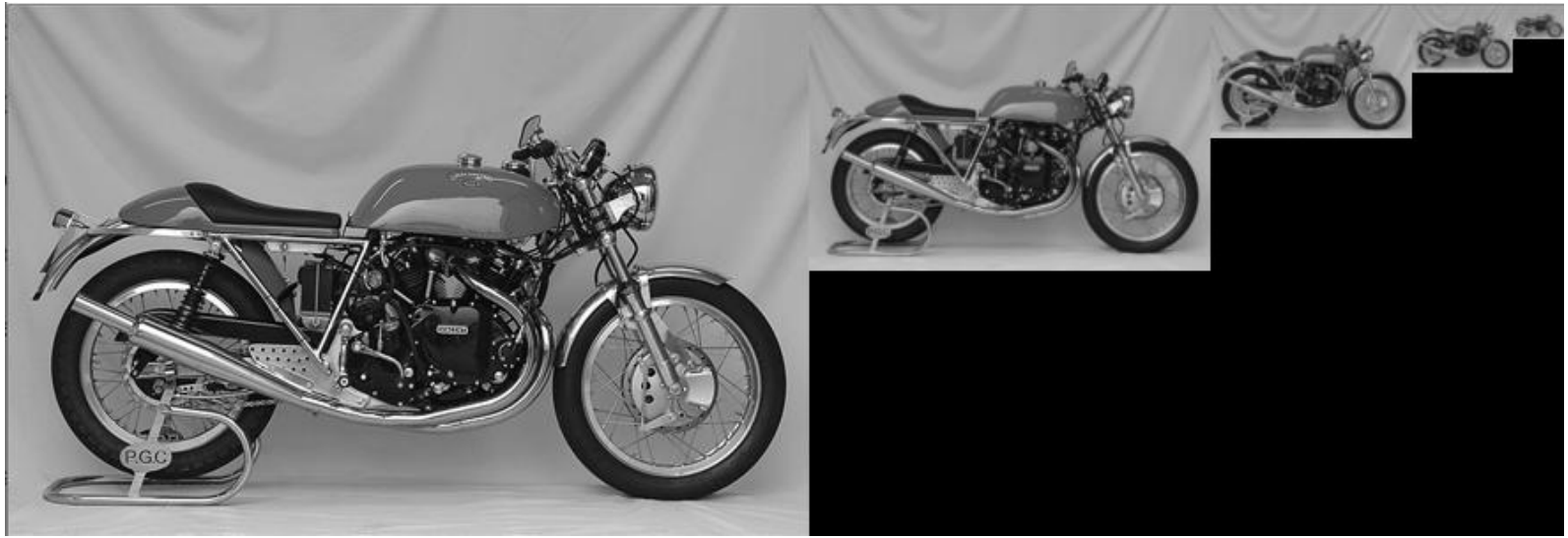


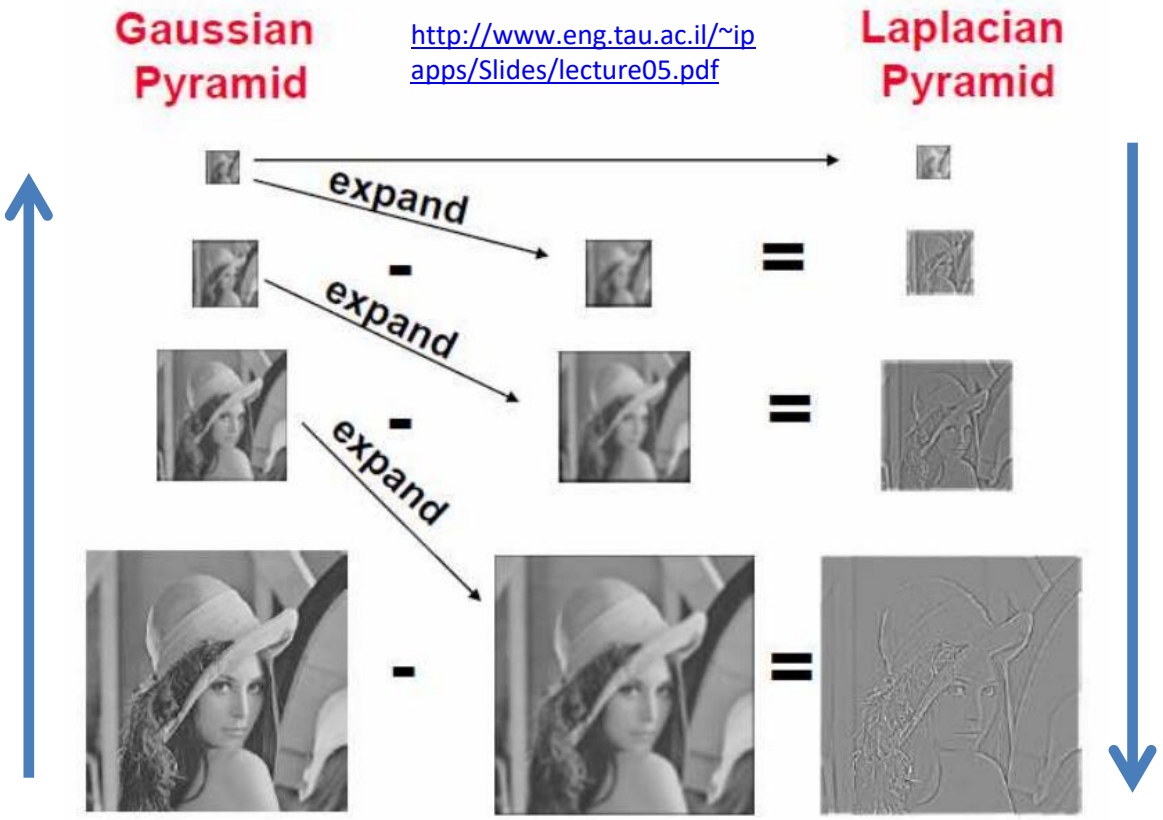
Imagen extraída de Wikimedia

EJERCICIO 2.A

- Ejemplo de pirámide Gaussiana de 4 niveles:



EJERCICIO 2.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.

BONUS

- 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**.

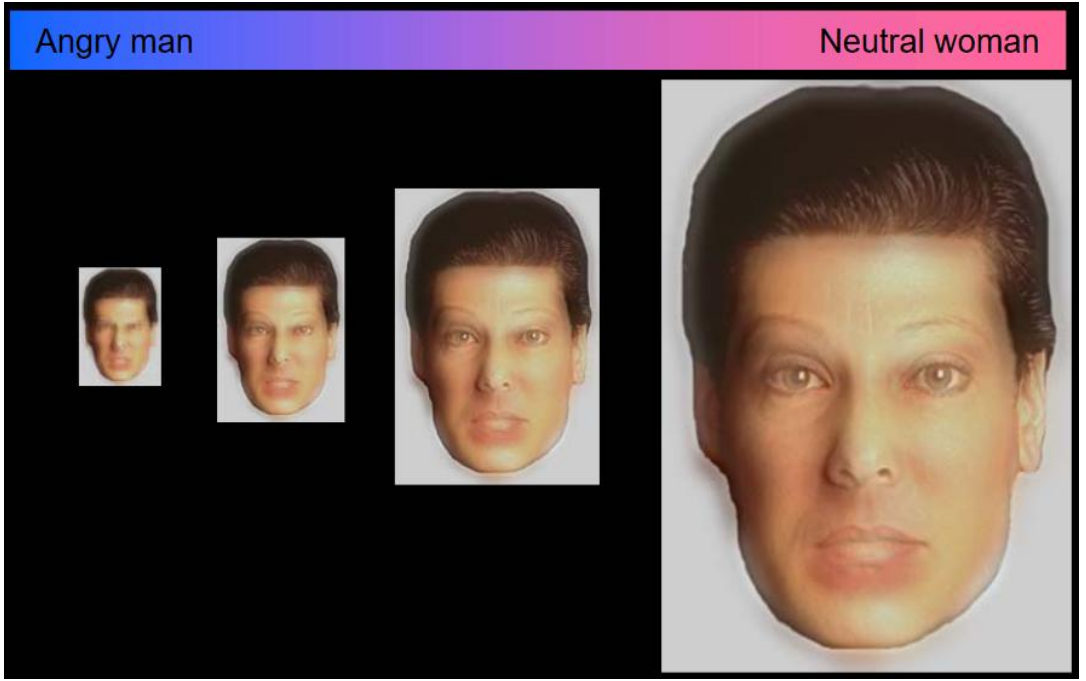


Imagen extraída de
http://olivalab.mit.edu/publications/Talk_Hybrid_Siggraph06.pdf

BONUS

- 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.

BONUS

- **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

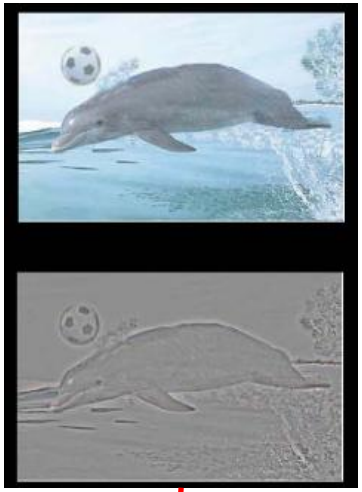
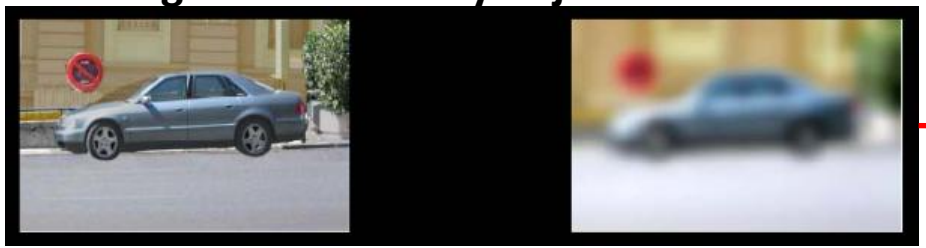
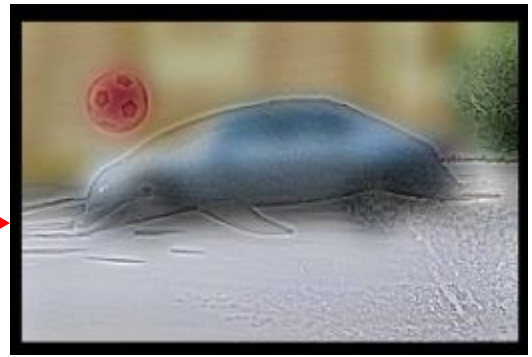


Imagen híbrida

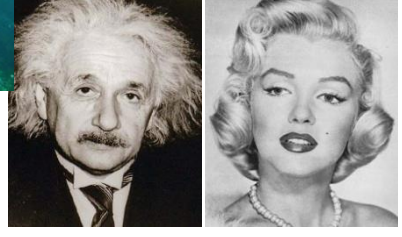
Imagen de entrada y baja frecuencia

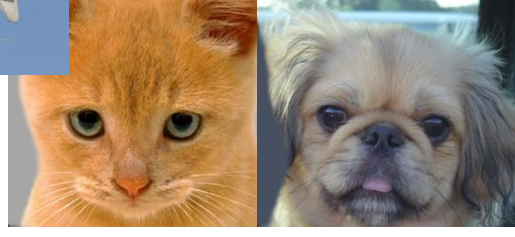




BONUS

- 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



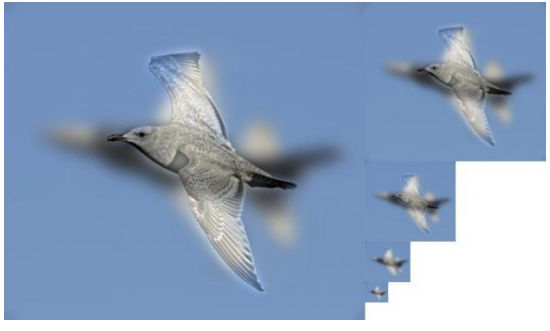


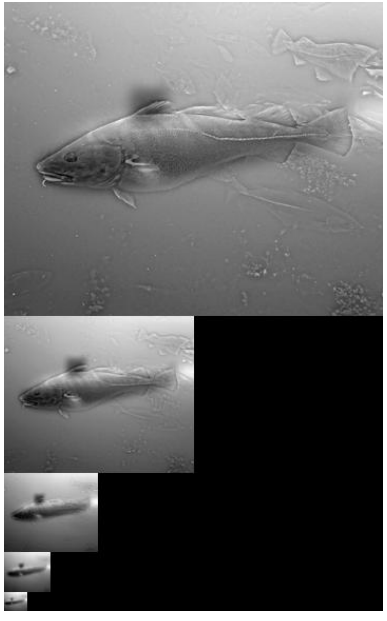
BONUS

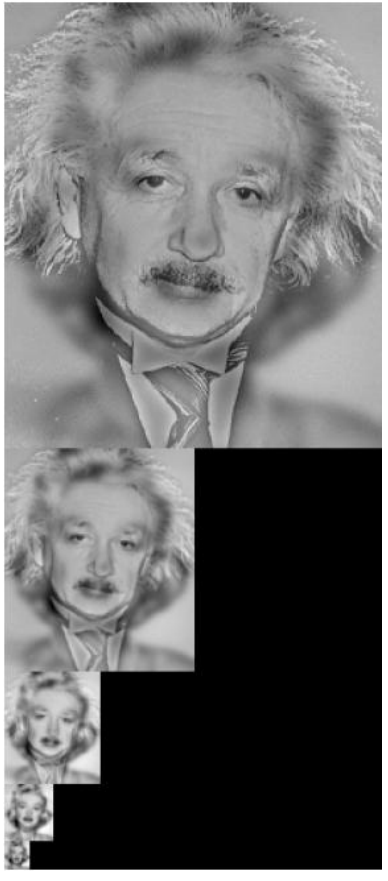
- Posibilidad sobre cómo proceder:

1) Alisáis fuertemente la imagen de frecuencias bajas hasta que solo quede casi una mancha sin detalles.

2) Para la de frecuencias altas calculáis la diferencia entre un realce de la original y el resultado de un alisamiento de la original.







Notas finales

- Acordaos de consultar la ayuda:
  - https://docs.opencv.org/4.1.1/d4/d86/group__imgproc__filter.html
- Acordaos de trabajar la memoria, para que todo lo que hacéis quede claro y bien justificado.
- Para esta práctica podéis hacer todo en Spyder (+ PDF con la memoria) o directamente en Colab.

Prácticas de Visión por Computador
Grupo 2

Trabajo 1:
Convolución y Derivadas

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

UNIVERSIDAD
DE GRANADA