Prácticas de Visión por Computador
Grupo 2

Presentación de la Práctica 1: filtros de máscaras

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

UNIVERSIDAD DE GRANADA

Normas de la Entrega de Prácticas

- Un único fichero Python irá llamando de forma secuencial a distintas funciones, una por cada apartado de la práctica.
- El código debe estar comentado.
- Se entrega memoria (PDF) y código (Python) → ZIP/RAR

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. Hay que fijar los parámetros que se consideren óptimos.
- Puntos de parada para mostrar imágenes, o datos por terminal.

Entrega

- Fecha límite: 28 de Octubre
- Valoración: hasta 8 puntos
- Lugar de entrega: PRADO

  https://pradogrado2021.ugr.es/course/view.php?id=14596#section-3

- **Se valorará mucho la memoria.**

Organización Prácticas

- Podéis ir al grupo que queráis, pero os corregirá vuestro ejercicio el profesor del grupo al que estéis asignados.
- Enviar dudas o solicitar tutorías online a pmesejo@decsai.ugr.es o pablomesejo@gmail.com.
  - Preferentemente Martes y Miércoles de 10:00 a 12:00 y Viernes de 12:00 a 14:00.

EJERCICIO 1

1.- USANDO SOLO FUNCIONES BASICAS DE OPENCV : escribir funciones Python que implementen de forma eficiente el cálculo de los siguientes puntos: (2.5 puntos)

A) Máscaras discretas de la funciones: Gaussiana, derivada de la Gaussiana y segunda derivada de la Gaussiana, todas ellas en el caso 1D.

B) El cálculo de la convolución de una imagen con una máscara Gaussiana cuadrada 2D de dimensiones inferiores a las de la imagen. Comparar su funcionamiento con la salida de la función de OpenCV GaussianBlur para el mismo tamaño de máscara.

C) Comparar las máscaras 1D calculadas en los puntos anteriores para el cálculo de las derivadas de una imagen usando alisamiento Gaussiano y las máscaras 1D dadas por la función getDerivKernels. Mostrar ejemplos con distintos tamaños de máscara (dibujar las máscaras), valores de sigma. Leer la implementación de OpenCV y valorar los resultados.

D) Usar las implementaciones de los puntos A y B para calcular las máscaras normalizadas de la Laplaciana de una Gaussiana para un sigma dado. Visualizar las máscaras encontradas y mostrar ejemplos de funcionamiento usando dos tipos de bordes (cero y replicados) y dos valores de sigma: 1 y 3.

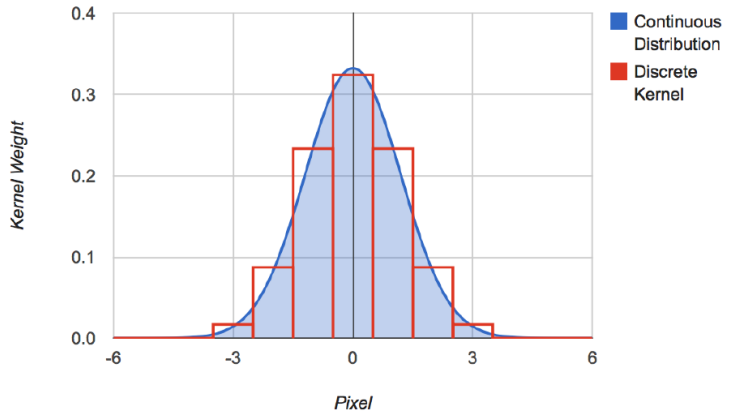
EJERCICIO 1.A

Gaussian Mask 1D

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

Ignoramos la constante *c*!

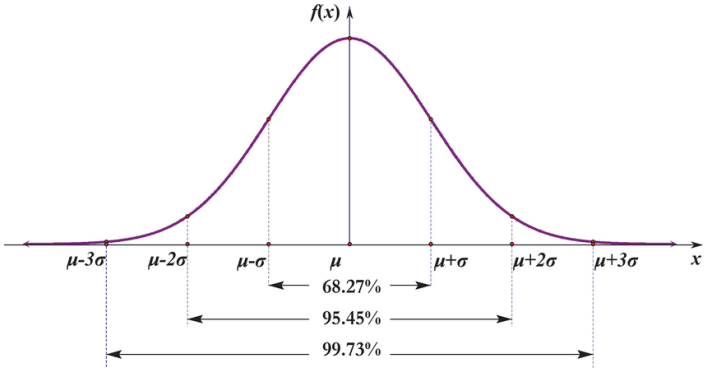
- $mask: [f(-k), f(-k+1), \cdots, f(0), f(k-1), f(k)],\ k\ an\ integer$



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

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

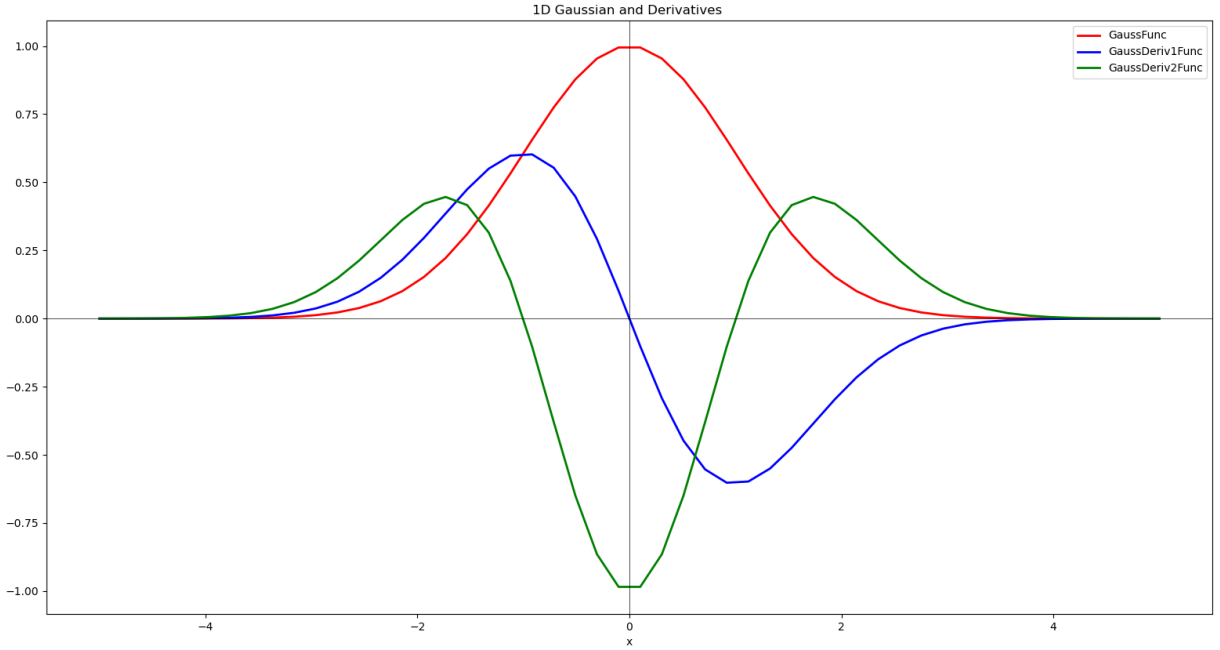
Imagen extraída de
http://dev.theomader.com/gaussian-kernel-calculator/



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

EJERCICIO 1.A

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



EJERCICIO 1.B

1.- USANDO SOLO FUNCIONES BASICAS DE OPENCV : escribir funciones Python que implementen de forma eficiente el cálculo de los siguientes puntos: (2.5 puntos)

A) **Máscaras discretas de la funciones: Gaussiana, derivada de la Gaussiana y segunda derivada de la Gaussiana, todas ellas en el caso 1D.**
B) **El cálculo de la convolución de una imagen con una máscara Gaussiana cuadrada 2D de dimensiones inferiores a las de la imagen. Comparar su funcionamiento con la salida de la función de OpenCV GaussianBlur para el mismo tamaño de máscara.**
C) **Comparar las máscaras 1D calculadas en los puntos anteriores para el cálculo de las derivadas de una imagen usando alisamiento Gaussiano y las máscaras 1D dadas por la función getDerivKernels. Mostrar ejemplos con distintos tamaños de máscara (dibujar las máscaras), valores de sigma. Leer la implementación de OpenCV y valorar los resultados.**
D) **Usar las implementaciones de los puntos A y B para calcular las máscaras normalizadas de la Laplaciana de una Gaussiana para un sigma dado. Visualizar las máscaras encontradas y mostrar ejemplos de funcionamiento usando dos tipos de bordes (cero y replicados) y dos valores de sigma: 1 y 3.**

Reutilizad en lo posible las funciones creadas en la P0: leerImagen, mostrarImagen, reescalarImagen,…

Asumimos que si hay *padding* es un tema de preprocesado. Cuando creéis una función de convolución, que tome como entrada la señal (es decir, la imagen) y el kernel 1D. Y el kernel recorre la imagen ajustándose perfectamente a los bordes.

OpenCV prima eficiencia a la precisión: que no os extrañe si vuestro resultado de la convolución no es exactamente igual al proporcionado por GaussianBlur. Seguramente vuestro código será “más correcto”, pero más lento.

EJERCICIO 1.B

1.- USANDO SOLO FUNCIONES BASICAS DE OPENCV : escribir funciones Python que implementen de forma eficiente el cálculo de los siguientes puntos: (2.5 puntos)

A) **Máscaras discretas de la funciones: Gaussiana, derivada de la Gaussiana y segunda derivada de la Gaussiana, todas ellas en el caso 1D.**
B) **El cálculo de la convolución de una imagen con una máscara Gaussiana cuadrada 2D de dimensiones inferiores a las de la imagen. Comparar su funcionamiento con la salida de la función de OpenCV GaussianBlur para el mismo tamaño de máscara.**
C) **Comparar las máscaras 1D calculadas en los puntos anteriores para el cálculo de las derivadas de una imagen usando alisamiento Gaussiano y las máscaras 1D dadas por la función getDerivKernels. Mostrar ejemplos con distintos tamaños de máscara (dibujar las máscaras), valores de sigma. Leer la implementación de OpenCV y valorar los resultados.**
D) **Usar las implementaciones de los puntos A y B para calcular las máscaras normalizadas de la Laplaciana de una Gaussiana para un sigma dado. Visualizar las máscaras encontradas y mostrar ejemplos de funcionamiento usando dos tipos de bordes (cero y replicados) y dos valores de sigma: 1 y 3.**

NOTA IMPORTANTE:
Menos el bonus, todo se hace en escala de grises:
`cv.imread(filename,0)`

Enlace interesante a nivel práctico:
http://www.songho.ca/dsp/convolution/convolution2d_separable.html

Sirve, tanto para visualizar la convolución 2D a partir de kernels 1D, como para tener un ejemplo sencilllo con el que depurar vuestro código, si es necesario.

EJERCICIO 1.C

1.- USANDO SOLO FUNCIONES BASICAS DE OPENCV : escribir funciones Python que implementen de forma eficiente el cálculo de los siguientes puntos: (2.5 puntos)

A) **Máscaras discretas de la funciones: Gaussiana, derivada de la Gaussiana y segunda derivada de la Gaussiana, todas ellas en el caso 1D.**

B) **El cálculo de la convolución de una imagen con una máscara Gaussiana cuadrada 2D de dimensiones inferiores a las de la imagen. Comparar su funcionamiento con la salida de la función de OpenCV GaussianBlur para el mismo tamaño de máscara.**

C) **Comparar las máscaras 1D calculadas en los puntos anteriores para el cálculo de las derivadas de una imagen usando alisamiento Gaussiano y las máscaras 1D dadas por la función getDerivKernels. Mostrar ejemplos con distintos tamaños de máscara (dibujar las máscaras), valores de sigma. Leer la implementación de OpenCV y valorar los resultados.**

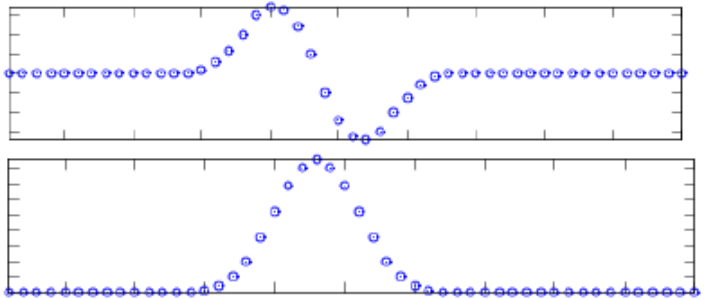
D) **Usar las implementaciones de los puntos A y B para calcular las máscaras normalizadas de la Laplaciana de una Gaussiana para un sigma dado. Visualizar las máscaras encontradas y mostrar ejemplos de funcionamiento usando dos tipos de bordes (cero y replicados) y dos valores de sigma: 1 y 3.**

EJERCICIO 1.D

1.- USANDO SOLO FUNCIONES BASICAS DE OPENCV : escribir funciones Python que implementen de forma eficiente el cálculo de los siguientes puntos: (2.5 puntos)

A) **Máscaras discretas de la funciones: Gaussiana, derivada de la Gaussiana y segunda derivada de la Gaussiana, todas ellas en el caso 1D.**
B) **El cálculo de la convolución de una imagen con una máscara Gaussiana cuadrada 2D de dimensiones inferiores a las de la imagen. Comparar su funcionamiento con la salida de la función de OpenCV GaussianBlur para el mismo tamaño de máscara.**
C) **Comparar las máscaras 1D calculadas en los puntos anteriores para el cálculo de las derivadas de una imagen usando alisamiento Gaussiano y las máscaras 1D dadas por la función getDerivKernels. Mostrar ejemplos con distintos tamaños de máscara (dibujar las máscaras), valores de sigma. Leer la implementación de OpenCV y valorar los resultados.**
D) **Usar las implementaciones de los puntos A y B para calcular las máscaras normalizadas de la Laplaciana de una Gaussiana para un sigma dado. Visualizar las máscaras encontradas y mostrar ejemplos de funcionamiento usando dos tipos de bordes (cero y replicados) y dos valores de sigma: 1 y 3.**

Ejemplo de comparación visual de máscaras 1D



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

(Laplacian)

EJERCICIO 1

1.- USANDO SOLO FUNCIONES BASICAS DE OPENCV : escribir funciones Python que implementen de forma eficiente el cálculo de los siguientes puntos: (2.5 puntos)

A) Máscaras discretas de la funciones: Gaussiana, derivada de la Gaussiana y segunda derivada de la Gaussiana, todas ellas en el caso 1D.

B) El cálculo de la convolución de una imagen con una máscara Gaussiana cuadrada 2D de dimensiones inferiores a las de la imagen. Comparar su funcionamiento con la salida de la función de OpenCV GaussianBlur para el mismo tamaño de máscara.

C) Comparar las máscaras 1D calculadas en los puntos anteriores para el cálculo de las derivadas de una imagen usando alisamiento Gaussiano y las máscaras 1D dadas por la función getDerivKernels. Mostrar ejemplos con distintos tamaños de máscara (dibujar las máscaras), valores de sigma. Leer la implementación de OpenCV y valorar los resultados.

D) Usar las implementaciones de los puntos A y B para calcular las máscaras normalizadas de la Laplaciana de una Gaussiana para un sigma dado. Visualizar las máscaras encontradas y mostrar ejemplos de funcionamiento usando dos tipos de bordes (cero y replicados) y dos valores de sigma: 1 y 3.

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.- IMPLEMENTAR funciones para las siguiente tareas (2.5 puntos)

Usar las funciones implementadas en el punto.1

A. **(1) Una función que genere una representación en pirámide Gaussiana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes replicados y justificar la elección de los parámetros**
B. **(1.5) Una función que genere una representación en pirámide Laplaciana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes replicados.**

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.

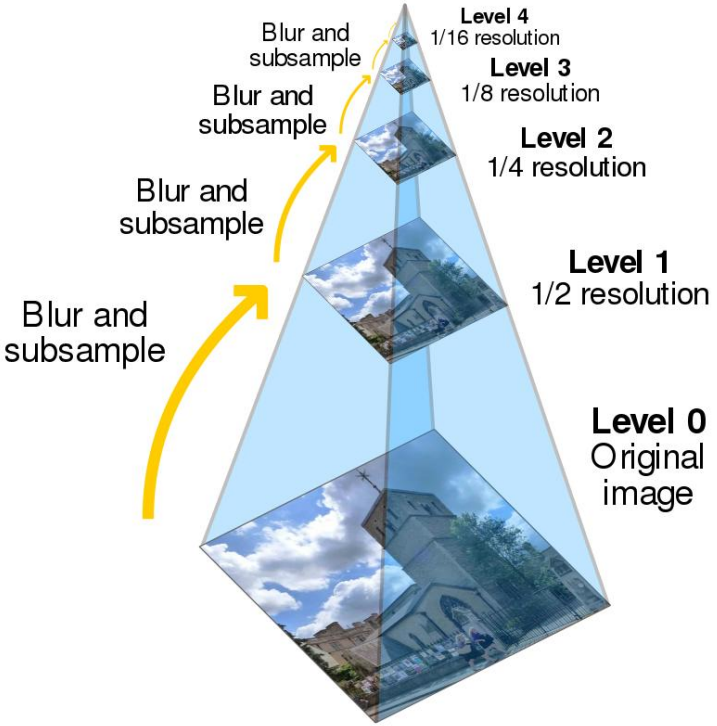


Imagen extraída de Wikimedia

EJERCICIO 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.

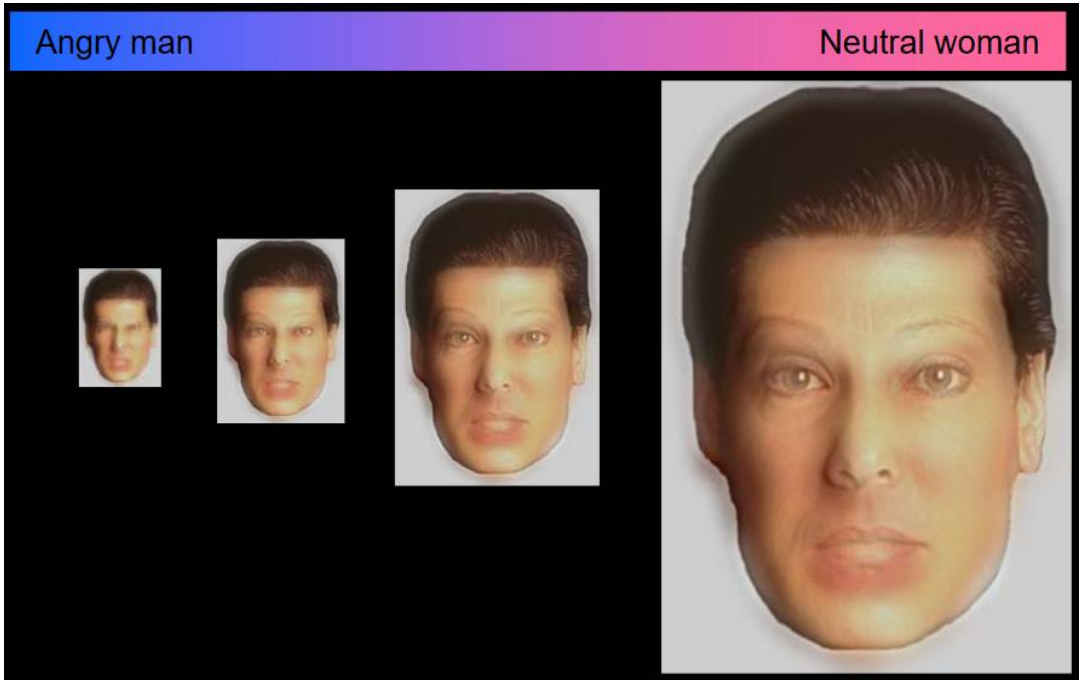


Imagen extraída de
http://olivalab.mit.edu/publications/Talk_Hybrid_Siggraph06.pdf

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

Implementar una función que genere las imágenes de baja y alta frecuencia a partir de las parejas de imágenes (solo en la versión de imágenes de gris).

1. **Buscar el valor/es de sigma más adecuado/s para cada pareja**
2. **Escribir una función que normalice los valores de los píxeles al intervalo [0,1] y muestre las tres imágenes (alta, baja e híbrida) en una misma ventana. (Recordar que las imágenes después de una convolución contienen número flotantes que pueden ser positivos y negativos)**
3. **Realizar la composición con al menos 3 de las parejas de imágenes**
4. **Construir pirámides gaussianas de al menos 4 níveles con las imágenes híbridas encontradas. Explicar el efecto que se observa a lo largo de la pirámide.**

EJERCICIO 3

Implementar una función que genere las imágenes de baja y alta frecuencia a partir de las parejas de imágenes (solo en la versión de imágenes de gris).

1. Buscar el valor/es de sigma más adecuado/s para cada pareja
2. Escribir una función que normalice los valores de los píxeles al intervalo [0,1] y muestre las tres imágenes (alta, baja e híbrida) en una misma ventana. (Recordar que las imágenes después de una convolución contienen número flotantes que pueden ser positivos y negativos)
3. Realizar la composición con al menos 3 de las parejas de imágenes
4. Construir pirámides gaussianas de al menos 4 níveles con las imágenes híbridas encontradas. Explicar el efecto que se observa a lo largo de la pirámide.

Imagen de entrada y alta frecuencia

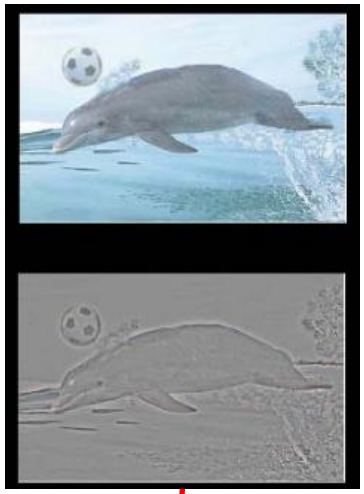


Imagen híbrida

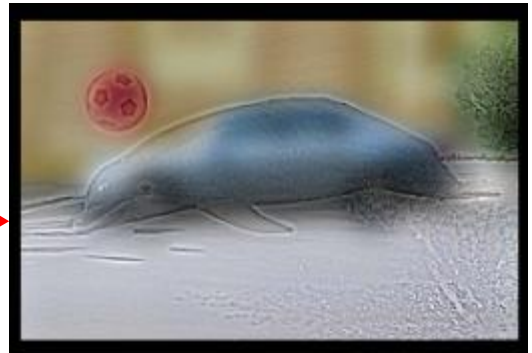
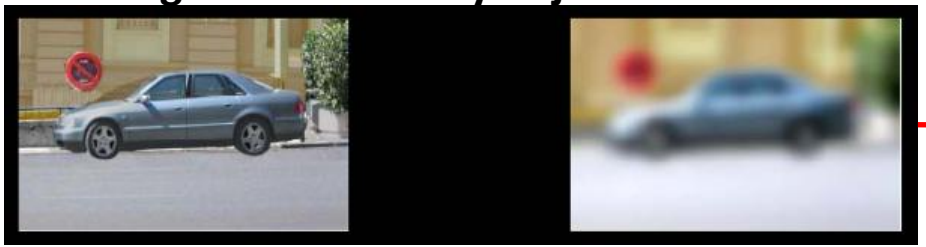


Imagen de entrada y baja frecuencia



EJERCICIO 3

Implementar una función que genere las imágenes de baja y alta frecuencia a partir de las parejas de imágenes (solo en la versión de imágenes de gris).

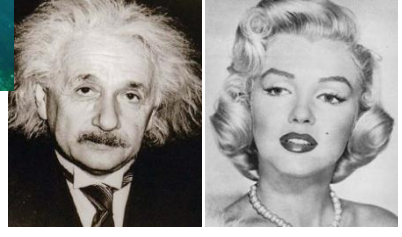
1. **Buscar el valor/es de sigma más adecuado/s para cada pareja**
2. **Escribir una función que normalice los valores de los píxeles al intervalo [0,1] y muestre las tres imágenes (alta, baja e híbrida) en una misma ventana. (Recordar que las imágenes después de una convolución contienen número flotantes que pueden ser positivos y negativos)**
3. **Realizar la composición con al menos 3 de las parejas de imágenes**
4. **Construir pirámides gaussianas de al menos 4 níveles con las imágenes híbridas encontradas. Explicar el efecto que se observa a lo largo de la pirámide.**

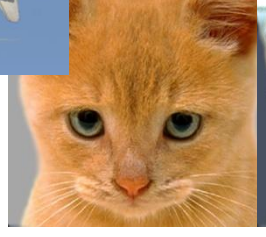
BONUS

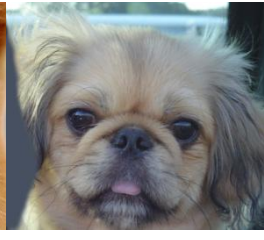
BONUS : Solo se tendrán en cuenta los bonus si se ha logrado al menos el 75% de los puntos en la parte obligatoria.

1.- Realizar todas las parejas de imágenes híbridas en su formato a color (1 punto) (solo se tendrá en cuenta si la versión de gris es correcta)

2.- Realizar una imagen híbrida con al menos una pareja de imágenes de su elección que hayan sido extraídas de imágenes más grandes. Justifique la elección y todos los pasos que realiza (1 punto)







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 no tenéis que verificar que vuestros códigos funcionan en Colab, como en la P0. Podéis hacer todo directamente en Spyder.

Prácticas de Visión por Computador
Grupo 2

Presentación de la Práctica 1: filtros de máscaras

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

UNIVERSIDAD DE GRANADA