PRÁCTICA 1: FILTROS DE MÁSCARAS

Antonio José Blánquez Pérez

Visión por Computador

Índice

1. (1A) Cálculo de máscaras discretas de las funciones

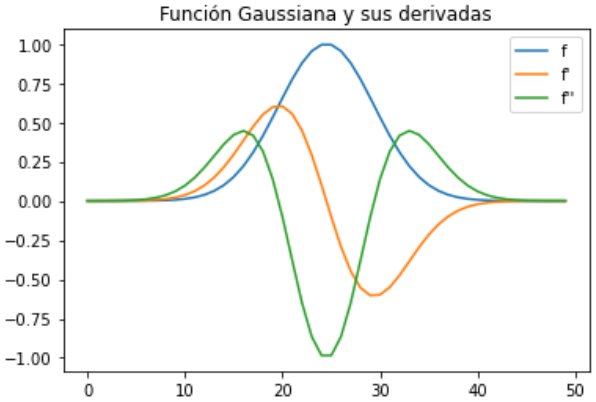
En este primer apartado se abordará el cálculo de las máscaras discretas, en concreto las de la Gaussiana y su primera y segunda derivadas, 1D, ya se hablará en el punto siguiente por qué sólo es necesario que sean 1D. Como ya se ha hablado en clase, usaremos la función Gaussiana ignorando la constante c,

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

esto es porque esta constante resultará luego al normalizar, ya que será aquella constante que haga que todas las componentes de la máscara sea igual a 1. El cálculo de sus primera y segunda derivadas es sencillo:

$$f'(x) = \frac{\partial f(x)}{\partial x} = -\frac{xe^{-\frac{x^2}{2\sigma^2}}}{\sigma^2}$$
$$f''(x) = \frac{\partial f'(x)}{\partial x} = \frac{e^{-\frac{x^2}{2\sigma^2}}(x^2-\sigma^2)}{\sigma^4}$$

Podemos comprobar que las funciones calculadas son correctas representándolas gráficamente, teniendo en cuenta que la derivada corresponde con la pendiente de la función original(σ=1).



Calculadas las derivadas e implementadas las funciones que devuelvan los valores de estas hay que ver como calcular las máscaras discretas. Para ello necesitaremos dos parámetros fundamentales: el valor σ para la Gaussiana, el cual establece la desviación típica, que se relaciona de manera directamente proporcional con el suavizado($> \sigma, >$ suavizado), y el tamaño T de la máscara, aunque es importante remarcar que uno depende del otro, por lo que en nuestra función no permitiremos establecer ambos si no pasar uno y calcular el otro a partir de él. En nuestro caso tomaremos la equivalencia $T = 2k + 1$, siendo k el valor tal que la máscara discreta de f se forme de la siguiente forma:

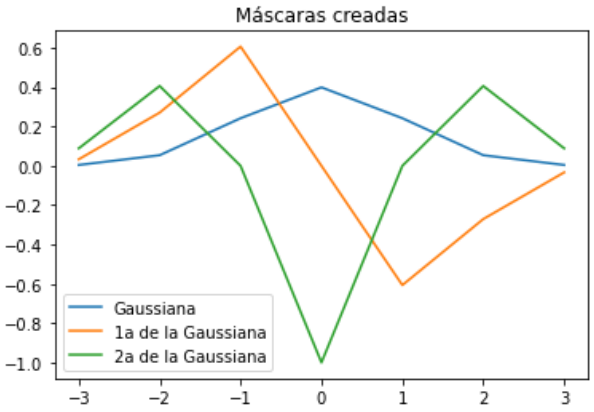
$$mask = [f(-k), f(-k+1), ..., f(k-1), f(k)]$$

y el valor 0 coincida con el píxel al que le vamos a aplicar la máscara en cada momento y con el centro de la Gaussiana. En nuestro caso se asignará a k

el mínimo valor entero que cumpla $k > 3\sigma$, debido a que a partir de las propiedades de la Gaussiana sabemos que en el intervalo $[-3\sigma, 3\sigma]$ se encuentra más del 95 % de la masa de probabilidad de esta, por lo que podemos discretizarla con estas dimensiones con garantías.

A partir de estos valores es sencillo generar la máscara discretizada, recordemos la definición arriba expuesta, basta con calcular $f(i) \forall i \in [-k, k]$. Esto es sencillo de entender ya que esta máscara la aplicaremos a los píxeles de una imagen, y por definición los píxeles están a una distancia de 1, por lo que la información que vamos a usar, aunque la Gaussiana sea continua, es tan solo la de los valores enteros en $[-k, k]$, que con las decisiones que hemos tomado y como ya hemos comentado nos proporciona más de un 95 % de la masa de probabilidad de la Gaussiana.

Realizando este proceso y haciendo uso de las funciones comentadas al principio del apartado podemos calcular máscaras discretizadas para la Gaussiana y sus derivadas, siendo necesario, eso sí, normalizar las máscaras, dado que los valores de la máscara Gaussiana deben sumar 1(dividiendo todos los valores por su sumatoria) debido a la naturaleza de este tipo de máscaras, y las derivadas con el objetivo de que el área sea constante y el filtro responda correctamente antes distintos valores de σ. Esto último se hace a través de multiplicar la primera derivada por σ, dado que cada lóbulos tienen un área de $\frac{1}{\sigma\sqrt{2\pi}}$, si multiplicamos por σ el área dejará de depender de ella; y, por tanto, multiplicando la segunda derivada por σ^2. Podemos ver que el resultado del cálculo muestra que las máscaras normalizadas respetan la forma comentada al principio del apartado(igual σ).



Todo lo expuesto en este apartado conforma nuestra función *gaussian_kernel*, que a partir de un σ o T dado nos devolverá una máscara discretizada 1D normalizada de la Gaussiana, su primera, o su segunda derivada.

2. (1B) Convolución de una imagen con una máscara Gaussiana cuadrada 2D y comparación con *cv2.GaussianBlur*

La pregunta principal en este apartado es cómo podemos convolucionar una imagen con una máscara Gaussiana cuadrada 2D a partir de una máscara Gaussiana 1D. Esto es fácil de entender si descomponemos la función Gaussiana:

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} = (\frac{1}{2\pi\sigma}e^{-\frac{x^2}{2\sigma^2}})(\frac{1}{2\pi\sigma}e^{-\frac{y^2}{2\sigma^2}})$$

Primeramente, al ser separable en dos funciones, una dependiente de x y otra de y, podrán usarse 2 máscaras 1D y no una 2D dado que aplicar esta última será equivalente a pasar una función por filas y la otra por columnas, lo cual es más eficiente a la hora de aplicarla a una imagen. Incluso, además de ser separables, podemos comprobar que ambas funciones son la misma si cambiamos x por y, es decir, podemos usar la misma máscara tanto por filas como por columnas(con la orientación correspondiente). Y esta función única es simplemente la usada en el apartado anterior, por lo que a través de la función implementada en dicho apartado podemos realizar la convolución de la imagen con máscara Gaussiana.

Para este apartado también se ha implementado una función, *convolucion*, la cual se encargará de convolucionar una imagen a través de dos máscaras que pasarán como parámetros(además de la imagen), y devolverá el resultado. Como hemos comentado, en realidad solo nos hará falta una máscara, pero dado que esta función se usará más adelante y para hacerla algo más genérica y versátil, estará adaptada a usar dos máscaras diferentes, una por filas y otra por columnas.

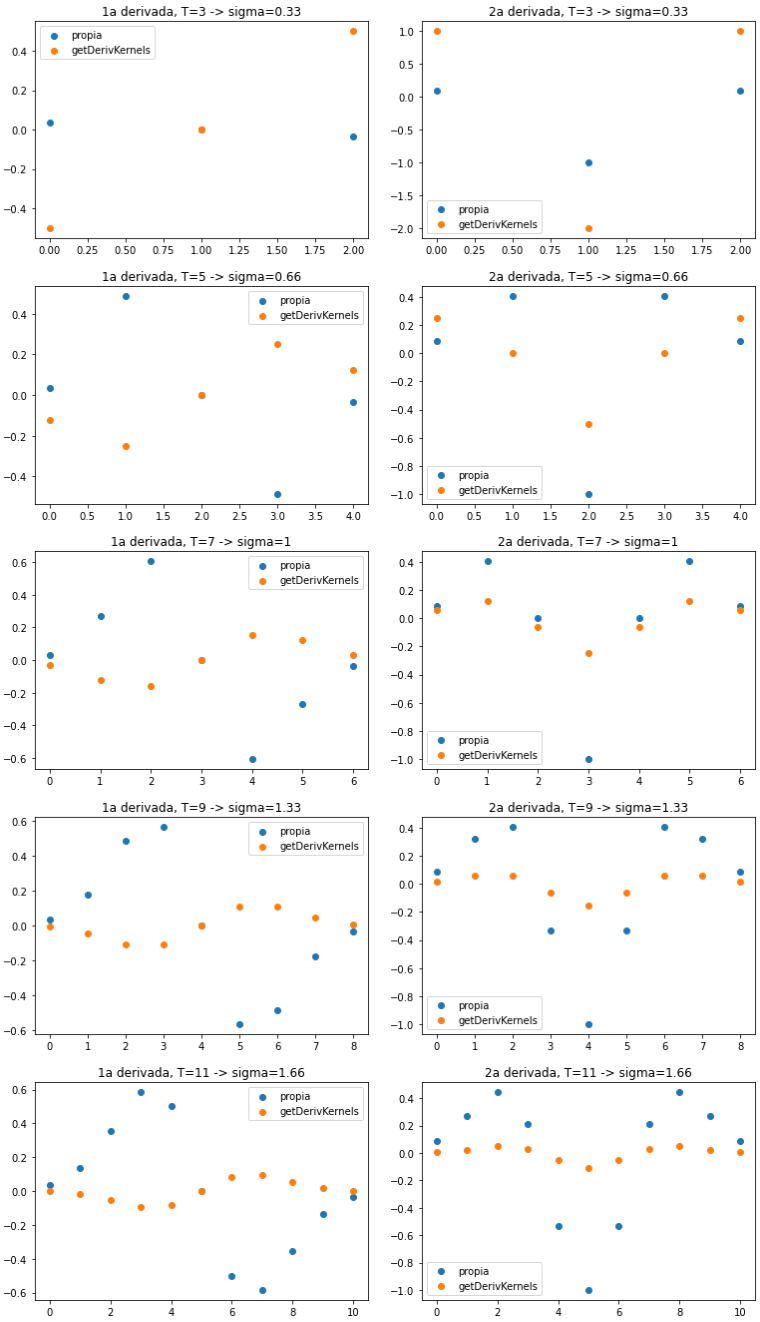
Antes de convolucionar es necesario añadir *padding*(bordes), al menos del tamaño k de la máscara, para que el resultado sea del mismo tamaño que la imagen original, dado que el proceso requiere de acceder a los k vecinos de cada lado y esto es imposible en los bordes de la imagen, por lo que debemos añadir nuevos bordes. A lo largo de esta práctica se hará uso de dos tipos de padding, bordes negros y bordes replicados, el primero, como su nombre indica, añadirá un marco negro a la imagen, mientras que el segundo replicará los bordes a modo de espejo, es decir que si los bordes de la imagen original son |abcd|, los bordes replicados quedarán tal que así dcb|abcd|cba. La implementación usada no tiene mucho interés, la función *addPadding* tan solo se crea una matriz de 0s del tamaño de la imagen sumado a dos veces un valor k que pasará como parámetro(el tamaño del borde) y se sobrescribirá la imagen centrada en ella, obteniendo la imagen con los bordes negros. Sólo es necesario, si se pide replicar, copiar el subvector correspondiente de cada fila y columna, invertido, en el borde. Además se ha implementado la función *removePadding* que elimina el padding, que tan solo extrae la submatriz correspondiente quitando unos bordes de tamaño k dado. Podemos ver gráficamente cómo queda el padding de ambos tipos, aunque posteriormente se añadirán y eliminarán de manera transparente en la ejecución, ya que se añadirá antes y se eliminará después de convoluciones(fuera de la función *convolucion*, por decisiones de implementación).

Lo primero que debemos hacer para convolucionar es invertir las máscaras, darles la vuelta, ya que vamos a convolucionar y no a correlacionar, aunque en este caso obtenemos el mismo resultado ya que la Gaussiana es simétrica, pero como hemos dicho, vamos a hacer la función general, no sólo para máscaras Gaussianas. Previamente se debe haber añadido un padding igual al valor k de la máscara(un k para las filas y otro para las columnas, dado que las máscaras pueden tener distinto tamaño), por lo que obtenemos dichos valores de las máscaras y comenzamos aplicando la máscara por filas, desde la posición k_{filas} hasta $tam_{img} - k_{filas}$ para cada fila. Cada píxel de la imagen resultante será el resultado de la sumatoria del vector resultante de multiplicar componente a componente la máscara con el subvector cuyo centro sea el propio píxel original. Se seguirá el mismo procedimiento aplicando la segunda máscara por columnas, desde $k_{columnas}$ hasta $tam_{img} - k_{columnas}$ al resultado del proceso anterior, obteniendo entonces el resultado final.

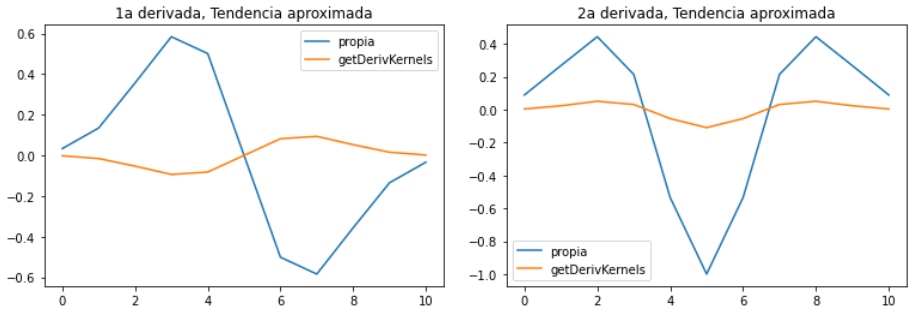
En cuanto a temas de implementación, es posible aplicar la segunda máscara por filas en vez de por columnas si transponemos la imagen, haciendo más sencillo el código, sin embargo no es demasiado complicado hacerlo por columnas y ahorramos dos transposiciones, por lo que se ha hecho de esta manera. Importante tener en cuenta que por simplicidad, en lugar de crear la nueva imagen con las medidas corregidas se aplica la convolución solo a los píxeles correspondientes y luego, fuera de *convolucion* se elimina el padding mediante *removePadding*. Por último se normaliza la imagen a los valores 0-255 y de tipo entero(en adelante se sobreentenderá lo mencionado cuando se use el término normalización para una imagen). Podemos ver el resultado final de aplicar lo comentado en este apartado a continuación, comparándolo con el resultado devuelto con $cv.GaussianBlur$(a igual σ, 2), teniendo en cuenta que es conveniente pasarle a este último ambos parámetros, ya que el cálculo de σ no es el mismo que el que hemos aplicado nosotros.

3. (1C) Comparación entre las máscaras 1D calculadas y las dadas por $cv2.getDerivKernels$

En este apartado vamos a comprobar las diferencias entre las máscaras de las dos derivadas que calculamos con las funciones comentadas en apartados anteriores con las que te devuelve $getDerivKernels$. Para ello vamos a analizar distintas gráficas comparando distintos tamaños de máscara.



Podemos ver, como es lógico, que al aumentar el tamaño de la máscara los puntos representan mejor la función constante. Podemos ver diferencias entre nuestras máscaras y las de *getDerivKernels*. Podemos ver una aproximación si juntamos los puntos de la última gráfica. Aunque es importante también ver que la máscara de la segunda derivada para *getDerivKernels* se va achatando a medida que aumentamos el tamaño.



En realidad *getDerivKernels* no devuelve filtros Gaussianos, si no que devuelve filtros Sobel o Scharr, en nuestro caso Sobel. Estos son también filtros de suavizado, siendo en realidad aproximaciones más livianas, también separables, que al contrario que los Gaussianos son simétricos tan solo en un eje(si lo visualizamos en 2D). Sin embargo, las principales diferencias no son relevantes debido a varios motivos. Primero, la diferencia de forma es debido a que *getDerivKernels* devuelve las máscaras ya invertidas para hacer la convolución, ahorrándose luego el paso durante esta; entonces, teniendo en cuenta esto, las dos funciones tendrían la misma forma, de hecho la forma de la primera y segunda derivada de la Gaussiana, siendo ahora la principal diferencia la escala. Aun así, la escala en definitiva no es demasiado determinante, ya que es posible alterarla multiplicando por una constante. De hecho, si a *getDerivKernels* le pedimos la máscara sin normalizar, los papeles se invierten ya que esta pasa a tener una escala mucho mayor a la de las máscaras propias. Por tanto podemos concluir que, aunque las máscaras son diferentes, estas son esencialmente equivalentes.

4. (1D) Cálculo de máscara normalizada de la Laplaciana de la Gaussiana

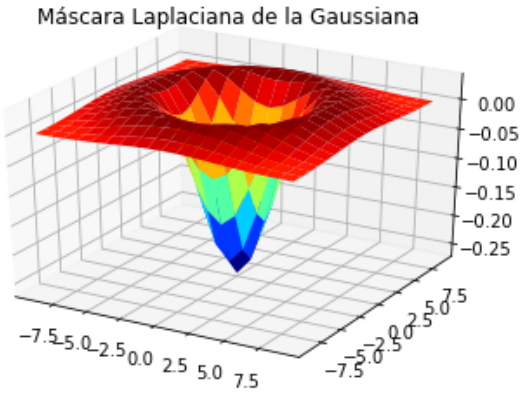
La parte principal de este apartado se centra en calcular la Laplaciana de la Gaussiana, empecemos a desarrollar a partir de la definición de la Laplaciana(ignoraremos las constantes ya se normaliza al calcular las máscaras Gaussianas, por lo tanto ya queda ese operador implícito a la hora de implementar).

$$L = (G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma))$$
$$G_{xx}(x,y,\sigma) = \frac{\partial G}{\partial x^2} = \frac{\partial}{\partial x^2}(e^{\frac{x^2+y^2}{2\sigma^2}}) = e^{\frac{y^2}{2\sigma^2}}\frac{\partial}{\partial x^2}e^{\frac{x^2}{2\sigma^2}}$$
$$G_{yy}(x,y,\sigma) = \frac{\partial G}{\partial y^2} = \frac{\partial}{\partial y^2}(e^{\frac{x^2+y^2}{2\sigma^2}}) = e^{\frac{x^2}{2\sigma^2}}\frac{\partial}{\partial y^2}e^{\frac{y^2}{2\sigma^2}}$$

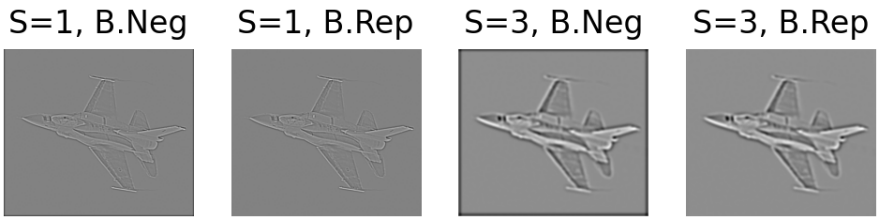
Si nos damos cuenta tenemos varias equivalencias con los cálculos hechos en el apartado 1A.

$$e^{\frac{x^2}{2\sigma^2}} = f(x) = G_\sigma(x) \Rightarrow$$
$$\frac{\partial}{\partial x^2} e^{\frac{x^2}{2\sigma^2}} = f''(x) = G''_\sigma(x)$$

Es decir, que podemos calcular la Gaussiana a partir de las máscaras 1D que hemos calculado en el apartado 1A. De la misma manera que hemos explicado en el apartado 1B, G_{xx} se puede calcular convolucionando la máscara segunda derivada de la Gaussiana por filas y y la Gaussiana por columnas. De igual manera, si nos damos cuenta, G_{yy} es igual a G_{xx} pero cambiando x por y, es decir, que se calcula haciendo el mismo proceso que con G_{xx} pero con la imagen traspuesta, o lo que es lo mismo, si aplicamos la Gaussiana por filas y la segunda derivada por columnas. Podemos apreciar la máscara que se forma al sumar ambas componentes formando la Laplaciana de la Gaussiana, con su característica forma de "sombrero mexicano"



Y esto es lo que se ha implementado en este apartado, se han calculado G_{xx} y G_{yy} mediante el procedimiento antes comentado, se suman ambos y se normaliza la imagen. Se ha realizado varias ejecuciones para dos valores de σ, 1 y 3, y para los dos tipos de bordes que se pueden usar al aplicar padding en la convolucion, negros y replicados.



Se puede apreciar que, a mayor σ, más clara es la detección de bordes, lógico porque da más peso a los valores vecinos, y que si se usan bordes negros se genera un marco negro dentro de la propia imagen. Esto último no es porque de alguna manera se trasladen los bordes dentro de la imagen, si no que la Laplaciana muestra el borde formado entre la imagen y el padding.

5. (2A) Representación en pirámide Gaussiana de 4 niveles de una imagen

El mayor interés en este apartado es el proceso de subsampling, que es bastante simple: para cada nivel se convoluciona la imagen con un sigma previamente establecido(en nuestro caso usando bordes replicados para el padding), y se reduce la imagen resultado a la mitad eliminando las filas y columnas pares o impares(en nuestro caso las pares, por una simple decisión de implementación, es irrelevante). Basta con repetir este proceso tantas veces como niveles quieras en la pirámide. Implementado este procedimiento se ha obtenido una pirámide de 4 niveles usando un σ de 1 para las convoluciones.

El valor de σ se ha obtenido experimentalmente, a prueba y error, hasta que se ha obtenido el efecto deseado, que en este caso no es un suavizado excesivo si no el justo para disimular el aliasing que aparece en la imagen al reducirla.

6. (2B) Representación en pirámide Laplaciana de 4 niveles de una imagen

En este caso no es una pirámide Gaussiana sino Laplaciana, sin embargo se va a usar el mismo procedimiento del apartado anterior con algunas adiciones. Esto es porque esta vez vamos a aproximar las Laplacianas mediante la diferencia de Gaussianas.

$$L \approx DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

Es decir, este apartado se va a implementar realizando primero una pirámide Gaussiana con el procedimiento ya comentado(la diferencia de σ aparecerá al aplicar varias veces el filtro), esta vez con un nivel más, ya veremos por qué, esta

vez teniendo cuidado de que no se pierda información al reducir las imágenes forzando unas dimensiones múltiplo de 16, dado que luego las Laplacianas sirven para reconstruir la imagen. Hecha la pirámida Gaussiana debemos sobreescribir los niveles de manera que cada nivel sea igual a sí mismo restando el siguiente nivel, es decir, level[i] = level[i] - level[i+1]. Tras la posterior normalización de cada nivel, podemos mostrar la pirámide Laplaciana de 4 niveles, abajo podemos ver una ejecución con $\sigma = 1$ establecido también por prueba y error al valor que nos proporcione mejor resultado.

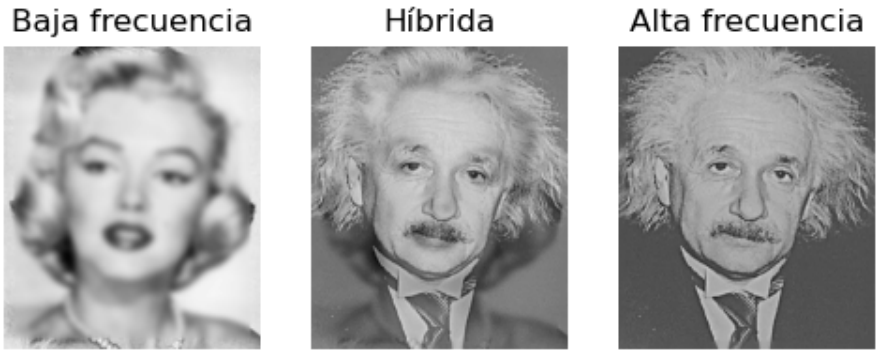
7. (3) Imágenes híbridas

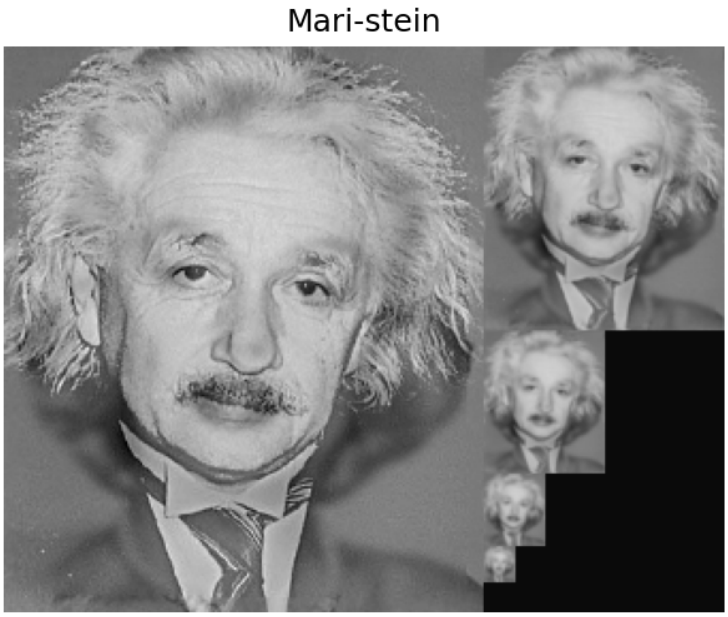
En este último apartado se hará uso de algunos de los recursos que ya hemos implementado en apartados anteriores para conseguir imágenes híbridas, es decir a partir de aplicar un filtro de paso alto a una imagen y uno de paso bajo a otra, llegar a una imagen que admite distintas interpretaciones a distintas distancias. El procedimiento que se ha aplicado en la práctica es sencillo, se aplica un filtro Gaussiano(paso bajo) a una de las imágenes y se calcula el mismo filtro para la segunda, pero esté se le restará a ella misma(multiplicada por un cierto valor$k$ que se podrá establecer para cada pareja), de manera que conseguimos un filtro de paso alto. Con ambos resultados basta hacer blending al 50 % de cada imagen(usando $cv.addWeighted$) y obtenemos la imagen híbrida, que podemos visualizar a través de una pirámide Gaussiana ya comentada para poder apreciar el resultado.

Para hacer las tres ejecuciones se han seleccionado las parejas Marilyn-Einstein, Bicicleta-Moto y Pez-Submarino. En general los valores se han establecido a prueba y error, ya que depende bastante de las imágenes, hasta obtener una pirámide en la que en el nivel 0 se escuentren bien delimitados los bordes de la imagen de altas frecuencias y en el nivel 4 la de bajas frecuencias. Como criterios generales, el valor de k depende mucho de las imágenes, pero los

valores de σ sí que cumplen el criterio de que el de la imagen de bajas frecuencias debe ser mayor al de altas frecuencias.

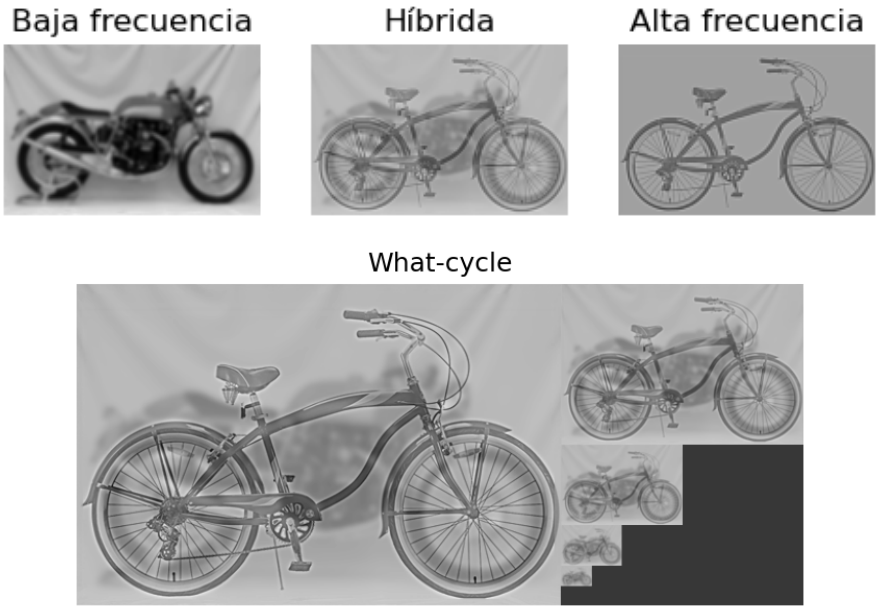
Para la primera pareja, Maristein(Marilyn-Einstein), parece que la imagen que funciona mejor a bajas frecuencias es la de Marilyn, por lo que la de Einstein quedará de alta frecuencia. En este caso el valor de k seleccionado para multiplicar a Einstein ha sido de 1.9, y los valores de σ son 2.7 y 2.2.





Como podemos ver, en el nivel 0 se ve a Marilyn de fondo y a Einstein bien definido, mientras que el nivel 4 se intuye a Marilyn, sobre todo los ojos. Este ha sido el mejor resultado que se ha podido conseguir, no está perfecto, pero está bastante conseguido.

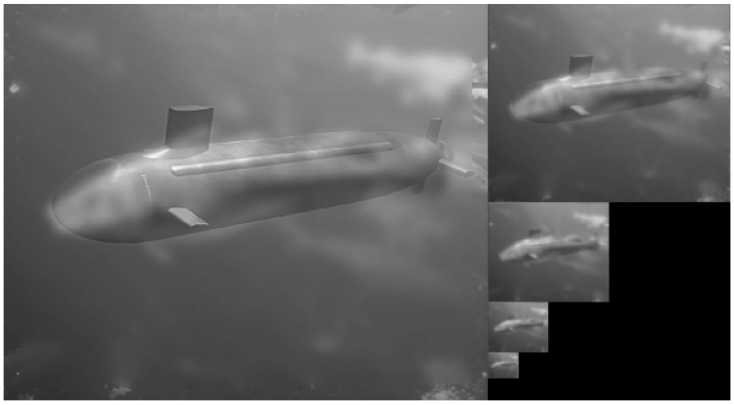
La segunda pareja, Whatcycle(Bicicleta-Moto), se ha establecido tal que la moto sea la imagen de baja frecuencia, ya que a la inversa no ha habido manera de conseguir que ambas imágenes sean visible en la híbrida. El valor de k se ha establecido a 1.5 y los σ de 3 y 2 respectivamente a moto y bicicleta.



Podemos apreciar perfectamente la bicicleta en el nivel 0 y como va apareciendo la moto hasta el nivel 4, donde la solo se aprecia la sombra de la bicicleta. Cuesta bastante que la bicicleta desaparezca correctamente, pero creo que el resultado ha quedado bastante bien.

La tercera y última pareja es Fishmarine(Pez-Submarino), que probablemente es la más difícil de conseguir de las tres, ya que es muy difícil de conseguir que el submarino aparezca por delante del pez, es por ello que en este caso se ha llegado a valores algo más exagerados: $k = 2$ y σ de 4 y 1 respectivamente para pez y submarino, siendo el pez la imagen de bajas frecuencias.

Fish-marine



Como vemos, se ha conseguido que, aunque comparado con otras imágenes tenga poca presencia, en el nivel 0 el submarino tenga los bordes bien definidos y el pez se vea como una sombra de fondo, de manera que en el otro extremo, el nivel 4, el submarino no tenga los bordes demasiado establecidos y el pez sí esté bien definido.