

Práctica 1: Filtros de Máscaras

Alejandro Alonso Membrilla

Contents

Introducción	3
Ejercicio 1	3
Apartado A	3
Apartado B	4
Apartado C	5
Apartado D	6
Ejercicio 2	8
Apartado A	8
Apartado B	10
Ejercicio 3	12
Apartado 1	13
Apartado 2	13
Apartado 3	13
Apartado 4	17
Bonus 1	18
Bonus 2	20
Referencias	21

Introducción

Esta práctica consistirá en una serie de ejercicios relacionados con filtros de imágenes, en particular aquellos obtenidos del uso de la función gaussiana y sus derivadas. Construiremos el cálculo de dichas máscaras y programaremos el proceso de convolución desde 0, y usaremos OpenCV como apoyo y referencia sobre la bondad de nuestros resultados.

Como aplicaciones interesantes del uso de estos filtros, estudiaremos las pirámides gaussiana y laplaciana (útiles en compresión y reconstrucción de imágenes), además de las imágenes híbridas, con las que haremos un uso combinado de filtros de paso alto y bajo.

La figura 1 muestra el hardware donde se ha probado el código entregado de forma local.



Figure 1: Especificaciones del hardware usado

Ejercicio 1

Este ejercicio es sobre el cálculo de los filtros básicos que usaremos en los ejercicios 2 y 3.

Apartado A

Calcularemos la máscara de la gaussiana y de sus dos primeras derivadas mediante una discretización de sus expresiones analíticas, la cual consiste en evaluarlas en un conjunto de puntos y, en el caso de la gaussiana, normalizar su valor de forma que la máscara quede balanceada (la suma de todos los pesos de la máscara debe valer 1). El hecho de que la función gaussiana n-dimensional sea separable (igual al producto de la gaussiana unidimensional en cada una de sus variables) hace que solo sea necesario calcular las máscaras 1D de cada uno de estos filtros.

La función gaussiana ha sido vista en teoría. La expresión de sus derivadas es la siguiente:

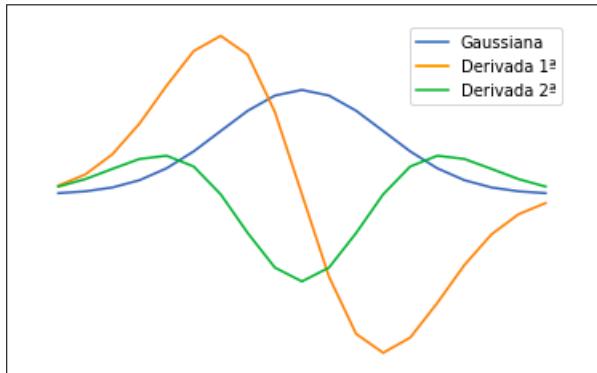
$$G'(x) = \frac{-x}{\sqrt{2\pi}\sigma^3} e^{\frac{-x^2}{2\sigma^2}} \quad G''(x) = \frac{1}{\sqrt{2\pi}\sigma^5} (-\sigma^2 + x^2) e^{\frac{-x^2}{2\sigma^2}}$$

Como debemos normalizar la máscara de la gaussiana, no tiene sentido considerar el valor de las constantes que la multiplican. Pordemos eliminar, por tanto, estas constantes de la expresión de la misma y de sus derivadas, resultando finalmente:

$$(G_{norm})'(x) = \frac{-x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}} \quad (G_{norm})''(x) = \frac{-\sigma^2 + x^2}{\sigma^4} e^{\frac{-x^2}{2\sigma^2}}$$

Para realizar la discretización, debemos elegir previamente el número de elementos de los que constará. Como la gaussiana tiene de dominio toda la recta real, hemos limitado este dominio a $[-[3\sigma], [3\sigma]]$, donde está contenido más del 95% de la masa de probabilidad de la distribución normal. Finalmente, hemos evaluado estas funciones en los números enteros dentro de este intervalo.

Hemos calculado y mostrado en una gráfica las máscaras calculadas. En este caso, el valor de sigma solamente influye en el tamaño del dominio. Un valor de σ de 3 es suficiente para exemplificar la forma de estas máscaras y comprobar que, efectivamente, corresponden a la función gaussiana y sus derivadas.



Apartado B

Este ejercicio consiste en la implementación de una función que realice la convolución de máscaras 2D separables, y el uso de esta función para implementar un filtro gaussiano bidimensional.

Sin entrar en detalles vistos en clase de teoría, la convolución consiste en sustituir cada elemento de una matriz por una combinación lineal de los elementos que la rodean. Que un filtro 2D sea separable implica que es posible calcular su convolución sobre una imagen como el resultado de aplicar una convolución 1D sobre dicha imagen con cada uno de sus factores. Esta propiedad tan interesante puede encontrarse en la función gaussiana y en sus derivadas (y, por tanto, en sus discretizaciones).

Para aplicar una convolución sin que la máscara se salga de los bordes de la imagen, debemos extender dichos bordes. Esta extensión se denomina *padding* y ha sido implementada para este ejercicio de tres formas distintas: borde fijo (negro), extendido y en bucle. Vemos a continuación un ejemplo de cada uno:



Para implementar la convolución se ha programado antes la correlación, conceptualmente más sencilla, pero que no es más que una convolución sin invertir los índices de la máscara. La convolución se ha escrito como una correlación con los índices invertidos.

Un detalle a mencionar sobre la implementación, realizada como toda la práctica usando `numpy` y su capacidad para vectorizar operaciones paralelizables, es que la máscara se ha pasado por todos los elementos de cada columna al mismo tiempo.

Una vez contábamos con una forma de aplicar la convolución entre una máscara y una imagen, podemos "alisar" una imagen dada convolucionándola con una máscara gaussiana bidimensional (o, equivalentemente, con dos máscaras gaussianas 1D). Un ejemplo de uso de esta función de suavizado, comparado con la función GaussianBlur de OpenCV, puede verse a continuación:



El método de padding usado en este ejercicio ha sido el de bordes en ciclo, puesto que este es el método usado por OpenCV e imitar esta característica facilita la comparación entre ambos.

Apartado C

En este ejercicio compararemos nuestras máscaras con las de OpenCV, obtenidas mediante la función [getDerivKernels\(\)](#). Una visualización superficial de lo que devuelve esta función nos muestra lo siguiente:

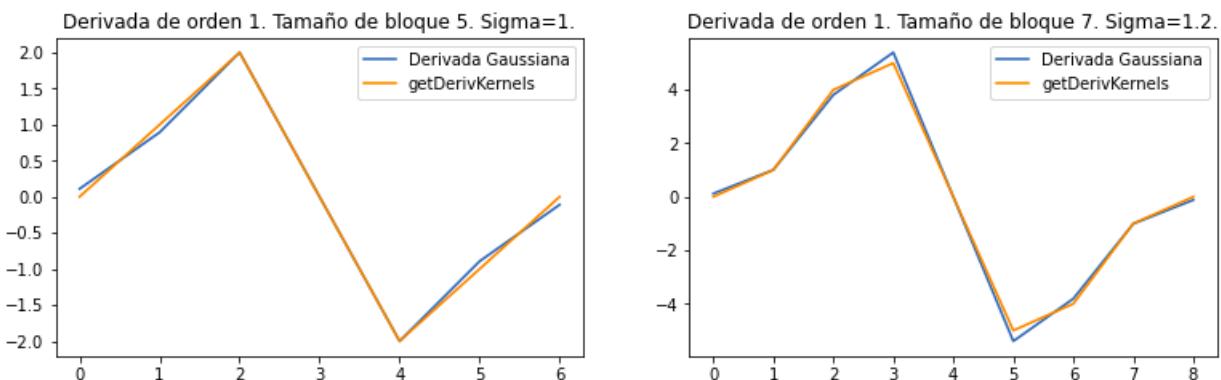
$$DerivKernel_1 = [-1 \ -2 \ 0 \ 2 \ 1]$$

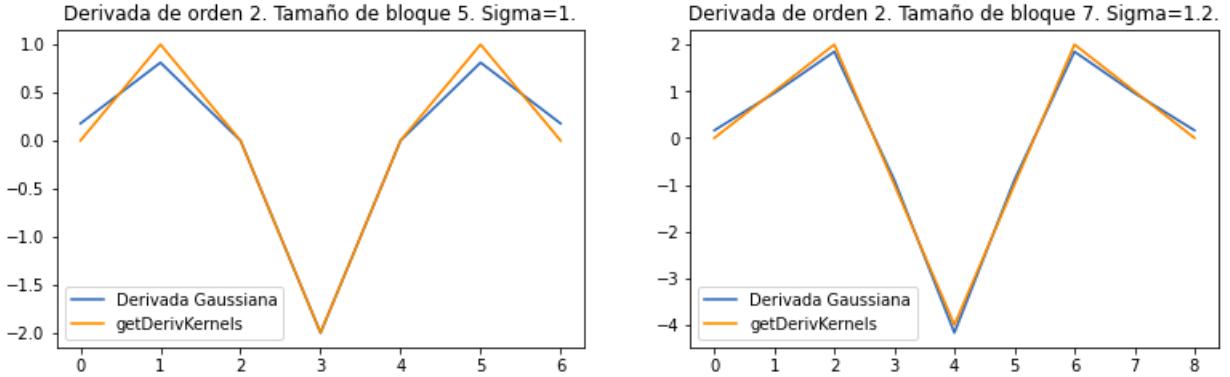
$$DerivKernel_2 = [1 \ 0 \ -2 \ 0 \ 1]$$

Observamos que las formas de nuestros resultados y los de OpenCV son similares, pero en el caso de la derivada primera de la gaussiana el vector aparece invertido. No es de extrañar teniendo en cuenta que la función `getDerivKernels` está pensada para ser pasada como parámetro de `sepFilters2D` (documentación de `getDerivKernels`).

Teniendo esto en cuenta, hemos probado a mostrar gráficamente las máscaras comparadas con las generadas por OpenCV, para lo que ha sido necesario centrar unas máscaras con respecto a otras y colocarlas de forma que puedan ser comparables, puesto que para distintos tamaños de sigma nuestras máscaras cambian de tamaño, al contrario que los kernels de OpenCV que toman su tamaño como parámetro. La figura 2 muestra los resultados de dicha comparación.

Nuestras máscaras y las de OpenCV parecen tener una forma parecida para los sigmas utilizados, salvo por un factor escalar (aparentemente distinto en cada caso). Encontrar el valor de sigma y el escalar exactos para los que nuestra implementación es más cercana a la de OpenCV para cada tamaño de bloque supondría una labor de optimización considerable que queda fuera del alcance de esta práctica. Para algunos de los casos anteriores se ha observado que los siguientes escalados son suficientemente similares a los kernels de OpenCV:





La implementación de OpenCV no calcula analíticamente la derivada de la función (en este caso la Gaussiana) y luego la discretiza, como hemos hecho nosotros, sino que convoluciona la imagen con un operador de Sobel, que combina diferenciación con alisamiento gaussiano. Los operadores de Sobel para calcular la derivada con respecto de x y con respecto de y son, respectivamente:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Teniendo en cuenta que este método es mucho más eficiente que el seguido por nosotros, puesto que emplea operaciones con enteros mucho más rápidas para una computadora, los kernels de OpenCV son muy parecidos a los obtenidos siguiendo un método analítico.

Apartado D

Este apartado está dedicado a la laplaciana de la gaussiana. Su cálculo consiste en la suma de las segundas derivadas parciales de la gaussiana aplicada sobre una imagen dada. Sin entrar en detalles de cálculo, sabemos que:

$$\frac{\partial G_{2D}}{\partial x^2}(x, y) = (G_{1D})''(x)G_{1D}(y) \quad \frac{\partial G_{2D}}{\partial y^2}(x, y) = G_{1D}(x)(G_{1D})''(y)$$

Luego cada derivada parcial es computable mediante una convolución 1D con la gaussiana y otra con su segunda derivada. Los resultados obtenidos y las máscaras utilizadas para calcularlos pueden verse a continuación:

$\sigma = 1$, modo **negro**

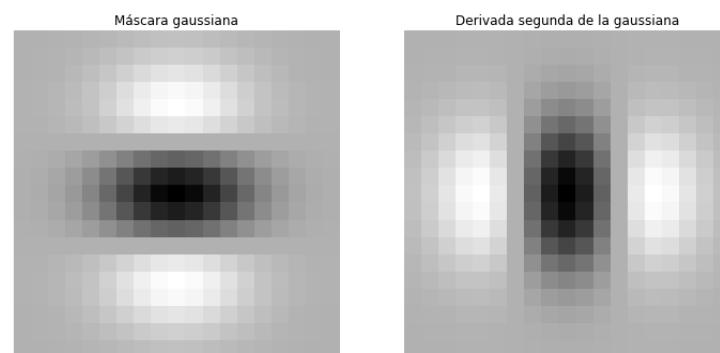




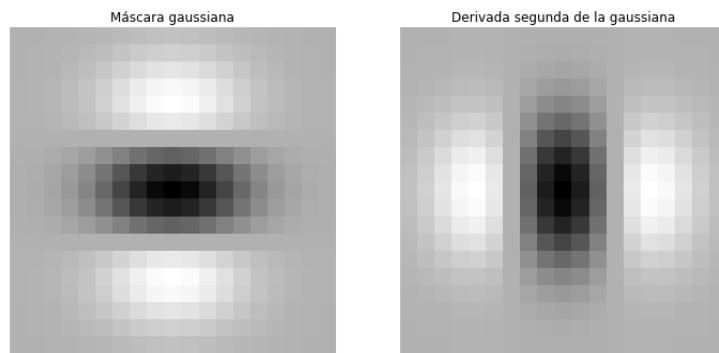
$\sigma = 1$, modo **replicado**



$\sigma = 3$, modo **negro**



$\sigma = 3$, modo **replicado**



Comparando derivadas parciales para un mismo σ vemos que la derivada con respecto a x tiende a destacar los detalles verticales, mientras que la derivada con respecto a y encuentra los horizontales. Los filtros con un σ pequeño, como viene siendo habitual, destacan detalles más finos, mientras que los que tienen un mayor σ deforman ligeramente la imagen y amplifican detalles más gruesos.

Ejercicio 2

Apartado A

Vamos a escribir una función para construir una pirámide gaussiana. Una pirámide gaussiana se forma iteradamente aplicando un suavizado gaussiano y luego eliminando la mitad de los elementos de la imagen.

Estos son los resultados de eliminar la mitad de las filas y columnas de una imagen sucesivamente sin suavizar:



Observamos una pérdida de calidad considerable. Veamos ahora como evoluciona la imagen para $\sigma = 0.9, 1.0, 1.1, 1.2$:



Para variaciones pequeñas de σ en torno a 1 no vemos cambios significativos entre pirámides. Probemos para desviaciones típicas más distanciadas:





Para la imagen usada como ejemplo, un valor de σ en torno a 1 parece obtener resultados adecuados. Si $\sigma \leq 0.5$ casi no se nota el suavizado a partir del nivel 3. Si $\sigma \geq 2$, a partir de ese mismo nivel observamos una pérdida de detalle considerable en los bordes y en los ojos.

Apartado B

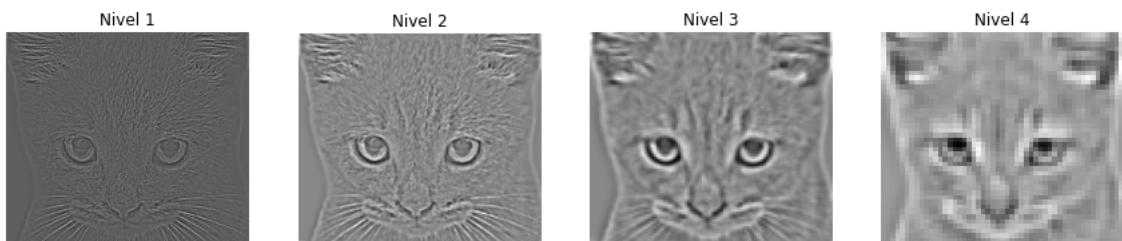
En este caso, la pirámide a construir es la laplaciana. Esta pirámide necesita de una función que aumente el tamaño de una imagen y la suavice (en OpenCV, `pyrUp()`). Hemos intentado implementar dicha función de forma similar a `cv.pyrUp`, insertando filas y columnas negras y suavizando con una máscara no balanceada (de suma 4, para equilibrar la pérdida de color perdido). El resultado no ha sido el deseado, y hemos escrito otra versión que aumenta la imagen con interpolación bilineal y le aplica un alisamiento gaussiano estándar. Los resultados obtenidos con cada versión son los siguientes:

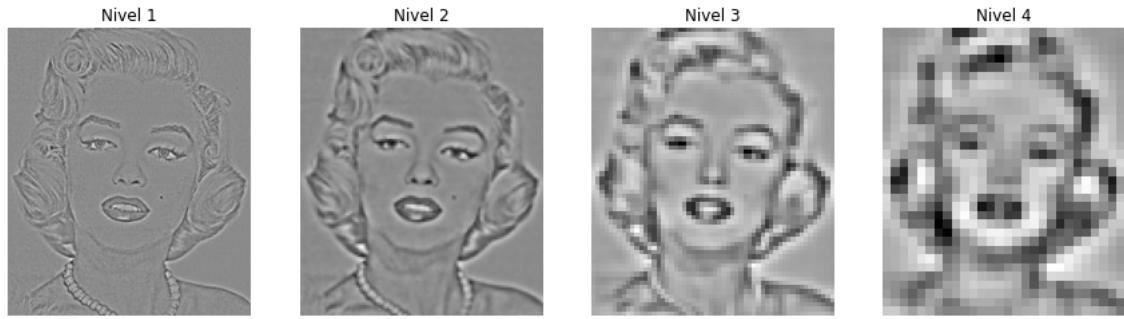


También es necesario calcular la diferencia entre dos imágenes (el escalón anterior de la pirámide gaussiana, y el escalón actual aumentado mediante `pyrUp`). Sin embargo, los métodos implementados hasta ahora para reducir y aumentar las imágenes asumen que estas tienen número de filas y de columnas que es potencia de 2, pero este no es el caso para nuestras imágenes de prueba, y no es el caso en muchos problemas del mundo real. Para solucionar este inconveniente hemos vuelto a aplicar el reescalado con interpolación bilineal de OpenCV para asegurarnos de que ambas imágenes a restar tienen el mismo tamaño.

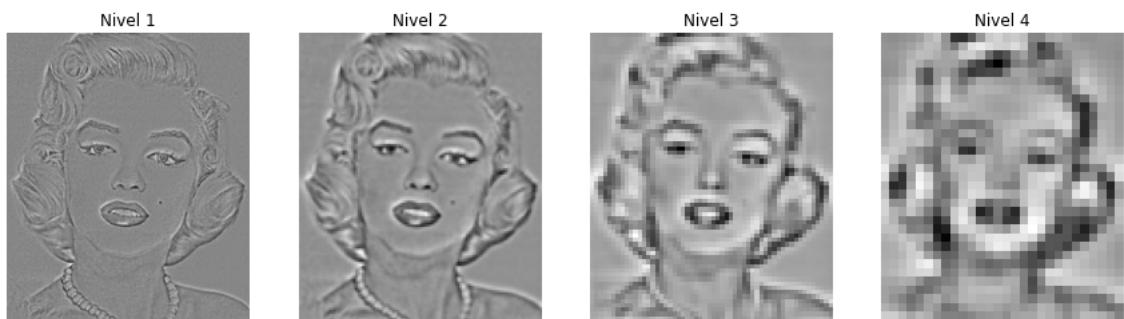
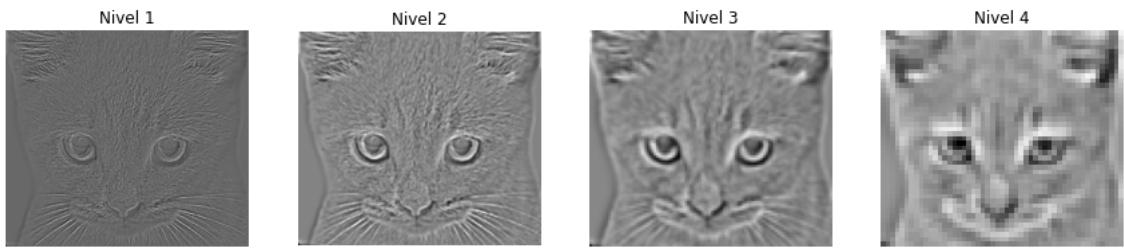
Procedemos a mostrar una pirámide laplaciana calculada mediante nuestra `pyrUp` y mediante `cv.pyrUp`. A partir del análisis realizado en el apartado anterior, concluimos que $\sigma = 1$ es adecuado para la construcción de la gaussiana y su respectiva laplaciana. Los resultados son los que siguen:

pyrUp():





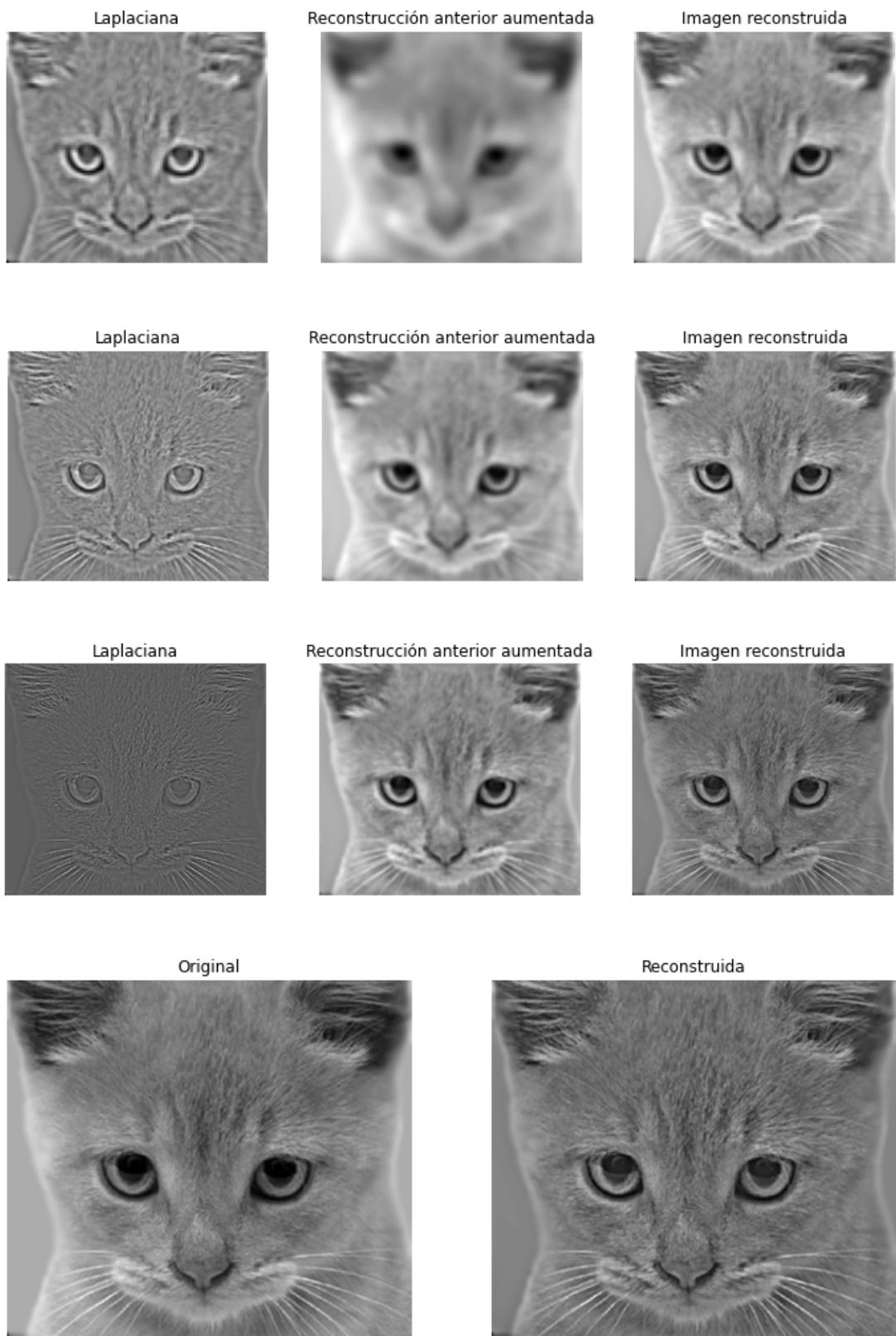
`cv.pyrUp()`:



Los resultados nos son exactamente iguales a los obtenidos usando `cv.pyrUp`, pero son suficientemente similares como para concluir que nuestra función para crear la pirámide laplaciana de una imagen funciona adecuadamente.

Sabemos que la pirámide laplaciana puede usarse para reconstruir imágenes a partir de un elemento de su pirámide gaussiana. Las imágenes siguientes muestran la reconstrucción completa de una imagen usando este método con bordes replicados en el suavizado:





Ejercicio 3

El objetivo del ejercicio 3 será programar una función para crear y visualizar hibridaciones de imágenes de diferentes formas, junto con la búsqueda de parámetros adecuados para generar el mejor efecto posible entre las dos imágenes.

Apartado 1

Para encontrar valores de σ que favorezcan el efecto de las imágenes híbridas, se ha tenido en cuenta las recomendaciones de Oliva, Torralba y Schyns en su artículo *Hybrid Images (2006)*. Ahí se sugiere tomar valores de sigma para cada imagen tales que el valor de la frecuencia se corresponde con la mitad de la ganancia de la imagen (es decir, la mitad de la información de la imagen está contenida en frecuencias más bajas/más altas que la utilizada). Sin embargo, no hemos sido capaces de hallar un método para encontrar dichas frecuencias a partir de una imagen dada y, por tanto, no hemos podido seguir esta sugerencia.

El método utilizado ha sido el de comparar, para cada par de imágenes, distintos valores de σ , de forma que acabamos seleccionando aquellos para los que se consigue mejor el efecto buscado. En cualquier lugar, cabe destacar que este efecto depende enormemente del tamaño de la imagen (puesto que este influye en la frecuencia) y, por tanto, los parámetros establecidos solamente son válidos para los tamaños que se presentan.

La búsqueda de parámetros adecuados se ha realizado en conjunción con el apartado 3, a la vez que se representa cada pareja y su imagen híbrida.

Apartado 2

La implementación de este apartado ha sido sencilla usando una función de la Práctica 0. Aquí se muestra un ejemplo de uso:



Apartado 3

Hemos formado cinco parejas de imágenes, y dentro de cada pareja hemos elegido una imagen para quedarse con las frecuencias altas y la otra para las frecuencias bajas. En general, la forma de proceder ha sido elegir para frecuencias altas aquellas que muestran un mayor número de detalles. Las imágenes híbridas generadas y sus respectivos valores de σ se muestran a continuación: $\sigma_{bajo} = 5.0$, $\sigma_{alto} = 1.0$



$$\sigma_{bajo} = 8.0, \sigma_{alto} = 1.5$$



$$\sigma_{bajo} = 11.0, \sigma_{alto} = 2.0$$



$$\sigma_{bajo} = 7.0, \sigma_{alto} = 0.6$$



$$\sigma_{bajo} = 9.0, \sigma_{alto} = 0.8$$



$$\sigma_{bajo} = 11.0, \sigma_{alto} = 1.4$$



$$\sigma_{bajo} = 10.0, \sigma_{alto} = 2.0$$



$$\sigma_{bajo} = 12.0, \sigma_{alto} = 2.8$$



$$\sigma_{bajo} = 14.0, \sigma_{alto} = 3.6$$



$$\sigma_{bajo} = 5.0, \sigma_{alto} = 0.4$$



$$\sigma_{bajo} = 7.0, \sigma_{alto} = 0.7$$



$$\sigma_{bajo} = 9.0, \sigma_{alto} = 1.2$$



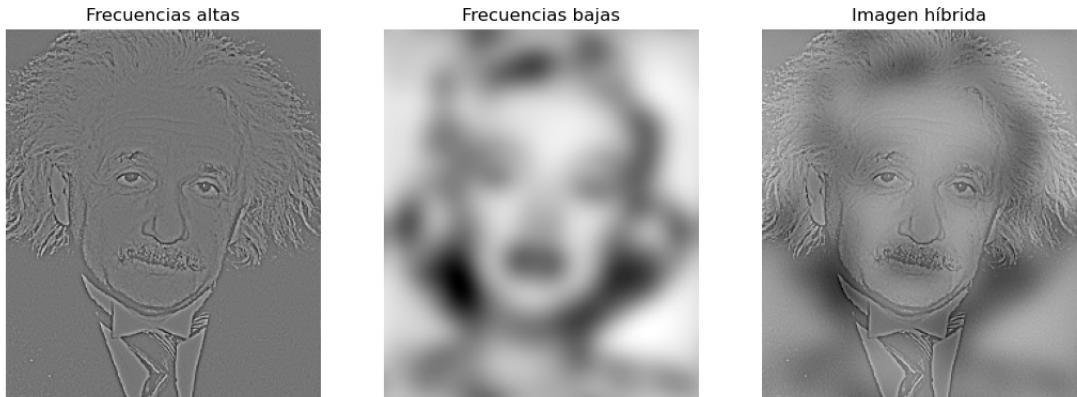
$$\sigma_{bajo} = 5.0, \sigma_{alto} = 0.4$$



$$\sigma_{bajo} = 7.0, \sigma_{alto} = 0.8$$



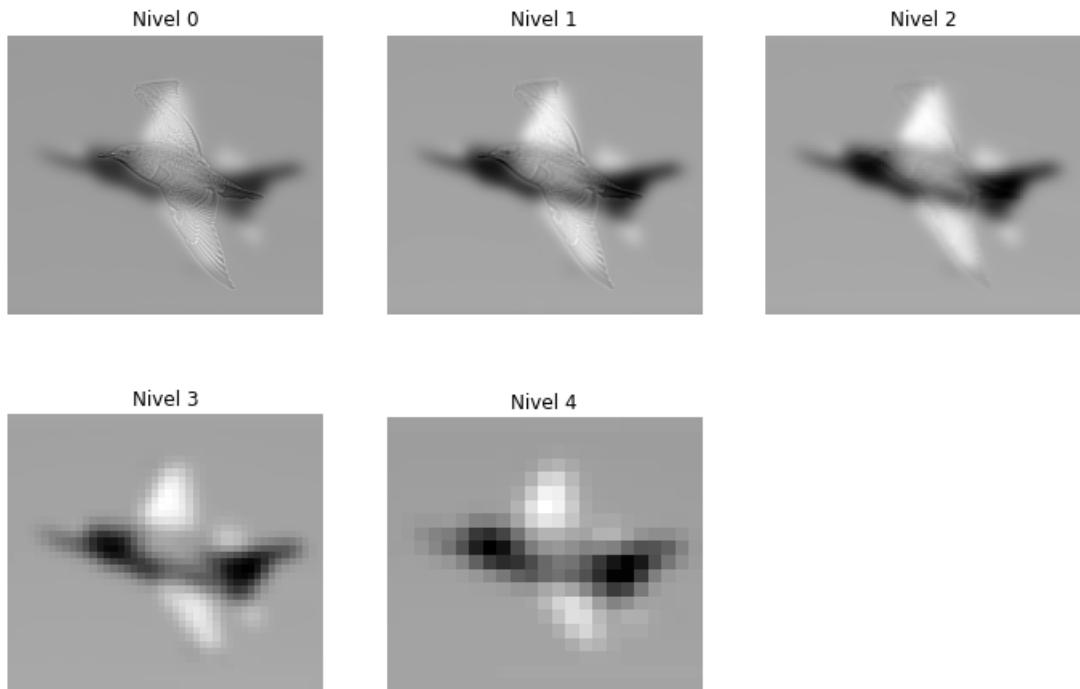
$$\sigma_{bajo} = 9.0, \sigma_{alto} = 1.3$$

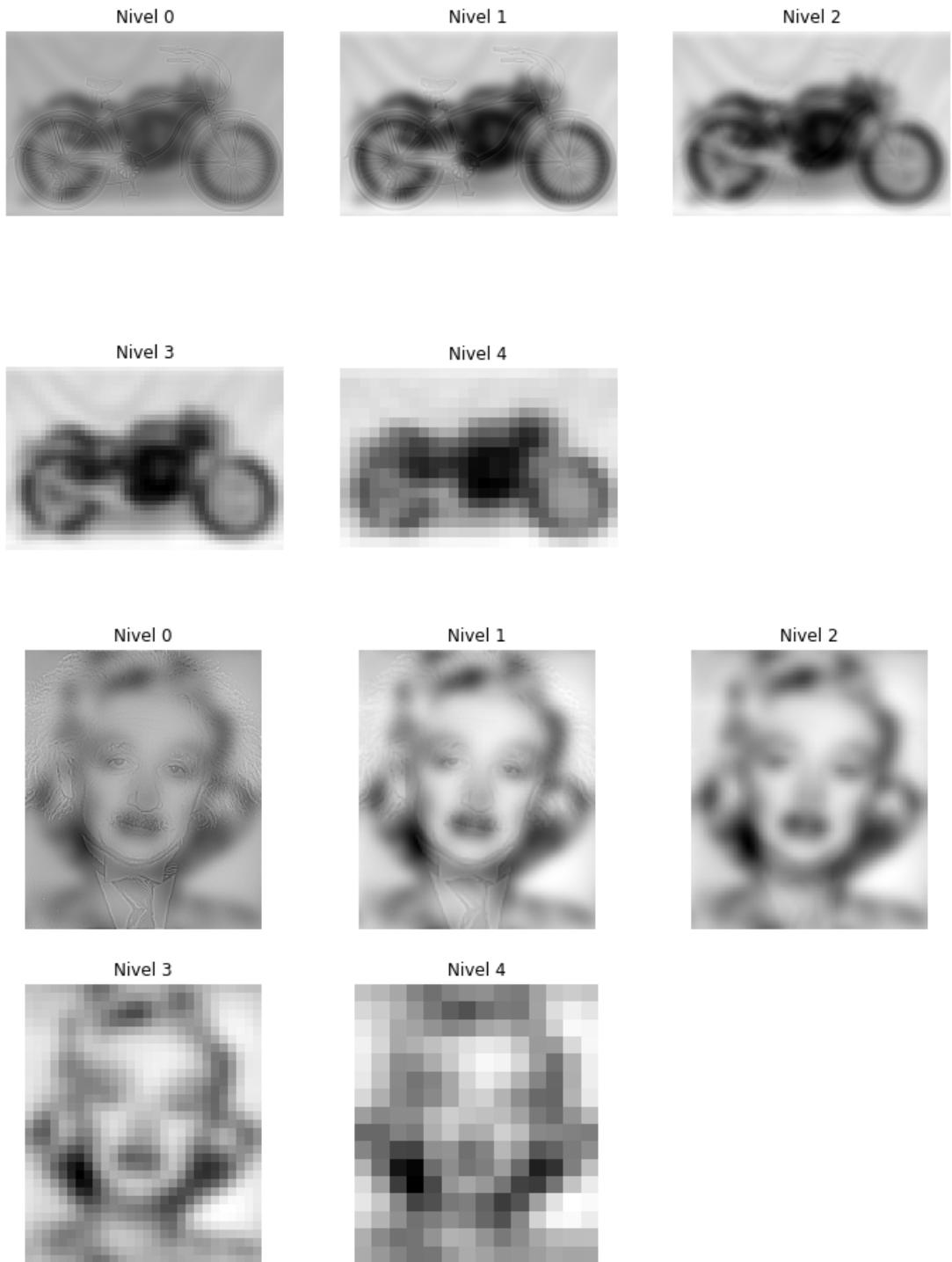


En lo que respecta a la selección de los σ , vemos que los valores adecuados cambian de una imagen a otra (de nuevo, estos muestran dependencia con las frecuencias presentes en cada imagen). Para el avión y el pájaro, unos valores de sigma de 8 y 1.5 parecen los más adecuados. El resto de los probados dan la impresión de que, o bien no dejan ver al pájaro, o son tan altos que este se sigue viendo desde la distancia. Por criterios similares, valores como 9 y 0.8 para la moto y la bici, o 7 y 0.8 para Einstein y Marilyn parecen los adecuados. En el caso del pez y el submarino, estos parecen estar en $7 \leq \sigma_{bajo} \leq 9$ y $0.7 \leq \sigma_{alto} \leq 1.2$. Para el perro y el gato, ningún valor parece mostrar claramente al perro desde cerca y al gato desde lejos. Esto puede deberse a que ambas imágenes no estén tan diferenciadas en cuanto a detalle como las demás.

Apartado 4

El último apartado del ejercicio 3 consiste en realizar pirámides laplacianas con imágenes híbridas y observar los resultados.



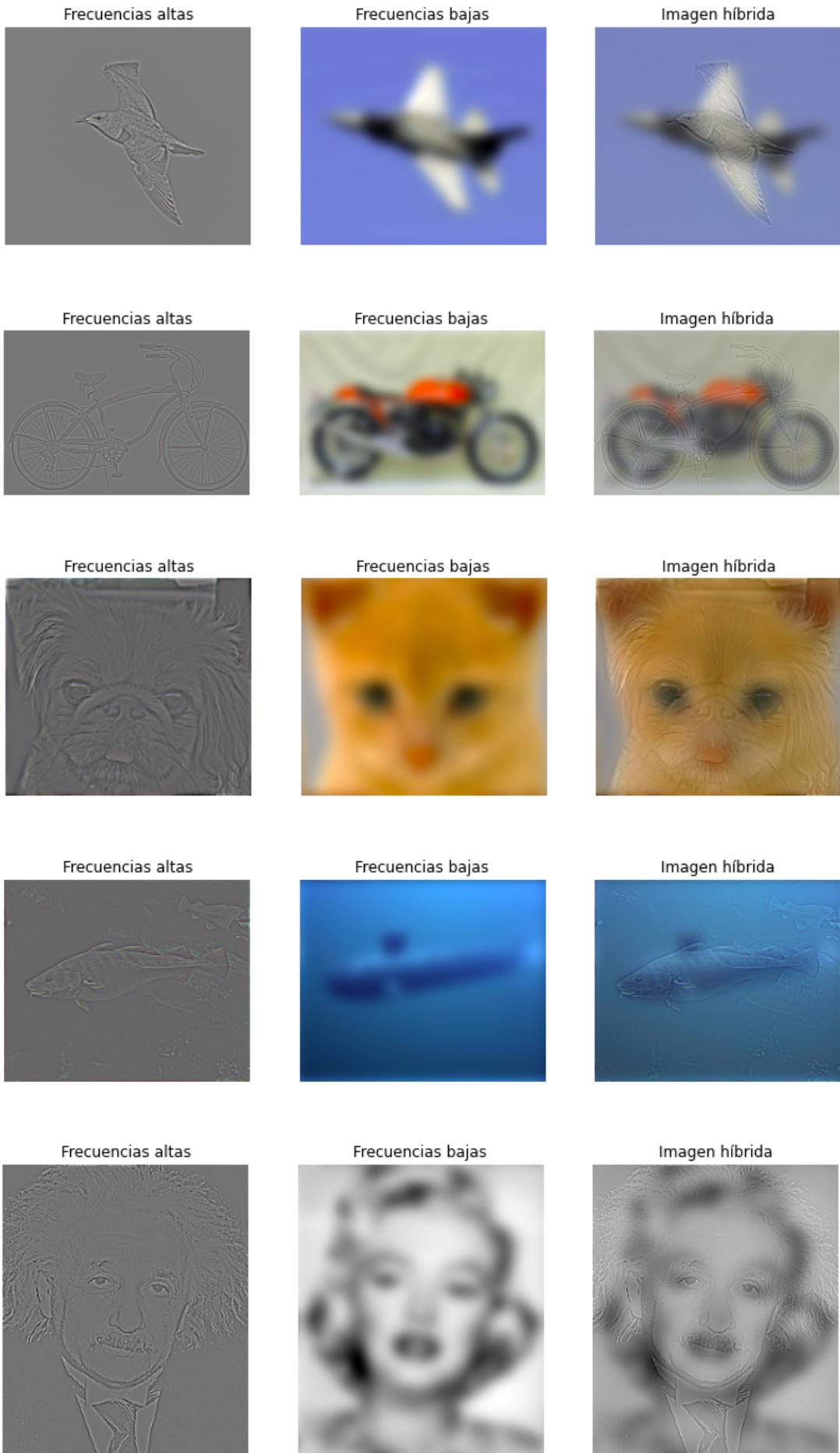


En cada uno de los ejemplos observamos el mismo patrón. Al subir en la pirámide, la imagen con frecuencias altas va desapareciendo y aquella con frecuencias bajas puede percibirse con mayor facilidad. Esto se debe a que, al formar la pirámide gaussiana, aplicamos un filtro de suavizado en cada nivel que elimina las frecuencias más altas, luego aquella imagen formada únicamente por frecuencias altas desaparece.

Bonus 1

En este bonus se ha realizado una ligera modificación de los parámetros usados en cada pareja para conservar el efecto de la hibridación. Además, tanto el filtro de paso bajo como el de paso alto usado al crear la imagen híbrida utilizan un suavizado gaussiano, que había sido implementado para operar con matrices monobanda y ha tenido que ser extendido al caso de las imágenes tribanda.

Las parejas y sus combinaciones a color obtenidas son:



Bonus 2

Para el Bonus 2 hemos hecho una imagen híbrida nueva a partir de una foto de un erizo, de la que hemos tomado las frecuencias altas, y la de una bola de bolos, de la que hemos tomado las bajas. El motivo es que, como se verá, las púas del erizo muestran una gran cantidad de detalles, mientras que la bola tiene colores más planos. Ambos aparecen en la imagen con forma redondeada, lo que nos ha permitido superponer uno sobre el otro.

El primer paso ha sido descargar estas imágenes. Ambas se han conseguido desde Google Imágenes (webs de origen en las referencias). A continuación, hemos recortado cada imagen para que el erizo y la bola queden encuadrados en la misma zona. Este es el resultado:



Quedan dos problemas por resolver. En primer lugar, las manos que sujetan ambas cosas aparecen en lados opuestos. Esto puede resolverse reflejando una de las imágenes. Por otro lado, la bola muestra un brillo intenso en el centro que afectaría a la imagen final. La solución propuesta es pintar este brillo con una mezcla de colores presentes en la bola. Como esta será suavizada en gran medida al hibridarla con el erizo, la falta de habilidad y de detalles en el resultado no supondrá un problema grave. Esta es la imagen obtenida de aplicar ambas correcciones:



Nótese que el hecho de haber recortado cada imagen no implica que ambas tengan ahora el mismo tamaño. Antes de hibridarlas, hemos usado cv.resize para escalar una al tamaño de la otra. Finalmente, se muestra la imagen híbrida:



Referencias

- Documentación de OpenCV para filtrado de imágenes.
- Definición de algunas funciones de OpenCV, como getDerivKernels.
- *Oliva, Torralba, Schyns: Hybrid Images (2006)*
- Web de la imagen del erizo.
- Web de la imagen de la bola de bolos.

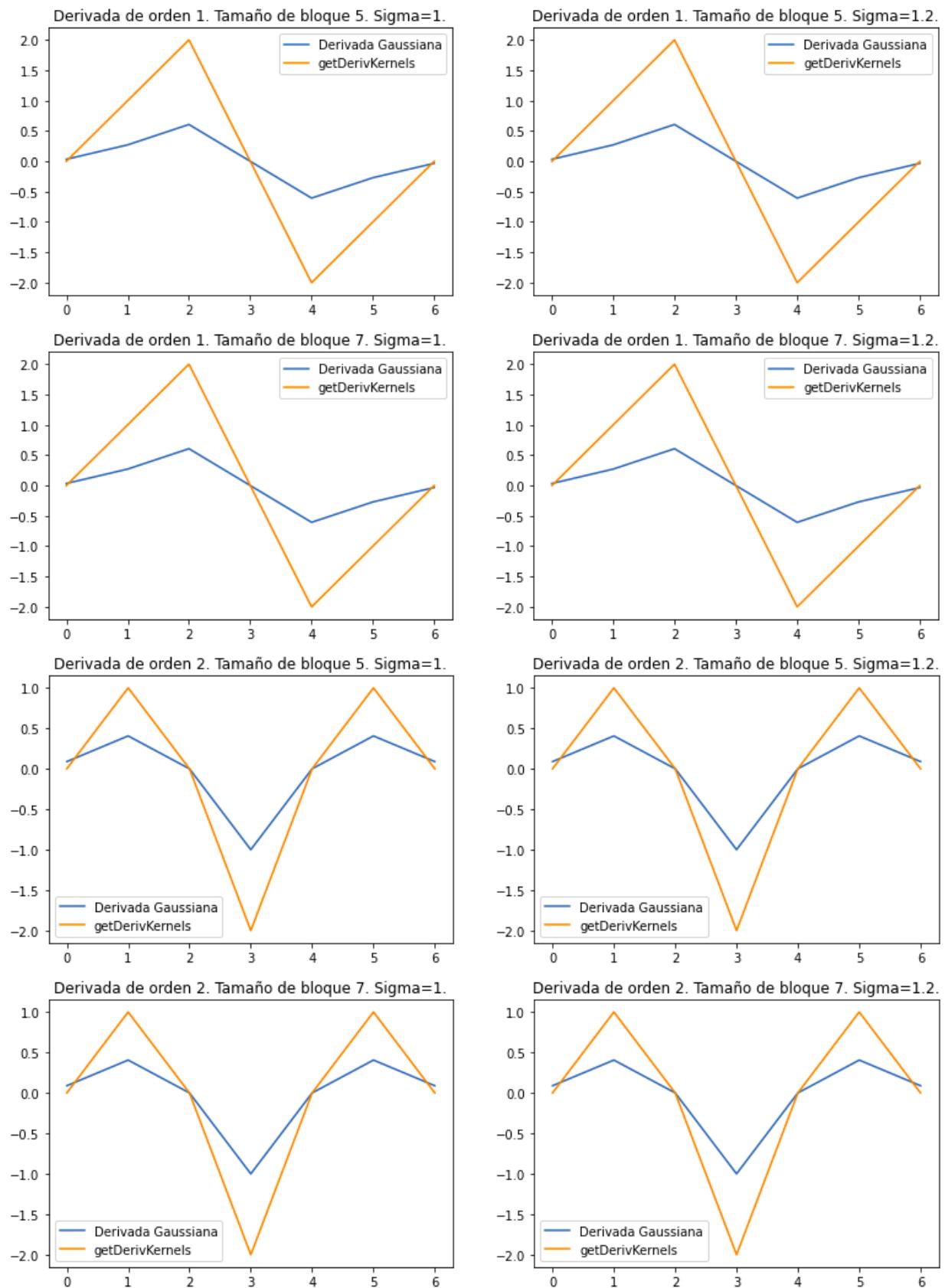


Figure 2: Comparación entre derivadas gaussianas discretizadas y kernels de OpenCV