Visión por Computador

Práctica 1:
Filtrado y Detección de Regiones

Curso 2019-2020
Cuarto Curso del Grado en Ingeniería Informática

Laura Rabadán Ortega 79088745W
laurarabadan@correo.ugr.es

Contenido

1. USANDO LAS FUNCIONES DE OPENCV

A) El cálculo de la convolución de una imagen con una máscara 2D. Mostrar ejemplos con distintos tamaños de máscara, valores de sigma y condiciones de contorno. Valorar los resultados.

Para aplicar la convolución de una imagen con máscaras 1D se ha creado una función que, con ayuda de la función *filter2D*, de *OpenCV*, aplica la máscara pasada como argumento por filas, con una primera llamada a la función, y por columnas, con una segunda llamada.

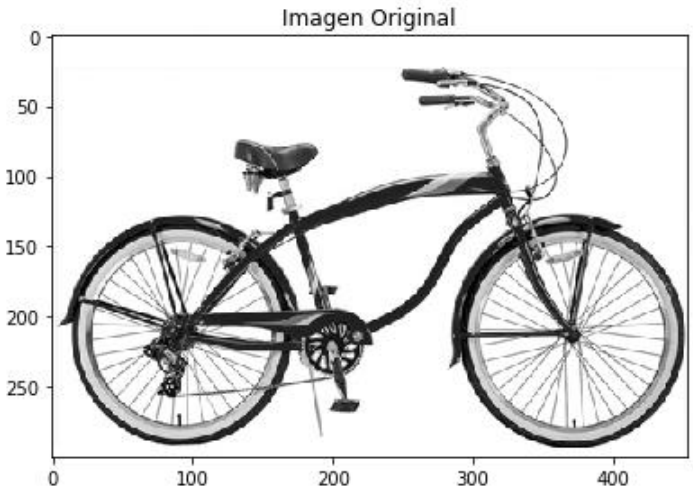
La función utilizada, *filter2D*, realiza la correlación de la imagen y la máscara pasadas como argumento. Por este motivo, para realizar la convolución debería modificarse la máscara. Este procedimiento no es necesario en este caso, ni en ninguno de la práctica, porque se están utilizando máscaras Gaussianas, las cuales son simétricas. Por tanto, el resultado de la correlación y de la convolución será el mismo.

Como la máscara Gaussiana es simétrica y separable, la máscara 1D que se pasea por las filas es la misma que la que se usa para las columnas, por lo que, al probar esta función en el apartado 1A, solo se necesita crear una máscara, que se usará para las filas, y trasponerla para usarla para las columnas.

Para probar la función, se utiliza una imagen a color y otra en escala de grises. Como parámetros, se prueban combinaciones de los siguientes valores:

- Tamaño: {3, 31}
- Sigma: {0.5, 5.17}
- Bordes: {Borde reflejado, Borde constante}
    - El borde reflejado copia la imagen en forma de espejo sin repetir el ultimo píxel, es decir, si se tiene [a b c d] y se necesita un borde de tamaño 2, se conseguiría [c b a d].
    - El borde constante añade tantos 0 como margen se necesite.

En primer lugar estudiamos la imagen en escala de grises. La imagen original es la siguiente:



Una vez se realiza la convolución con los parámetros anteriores se consigue:

- Tamaño de máscara 3:

- Tamaño de máscara 31:

Los cambios significativos se aprecian cuanto mayor sea tanto la máscara, su tamaño, como la sigma de la Gaussiana, ya que los detalles que se eliminan son mayores. Además, en la última imagen a la que se le ha aplicado una máscara de tamaño 31 es dónde mejor se aprecia como el borde puede influir en el resultado de la imagen, ya que la imagen se ve modificada por ese borde que se le añade para poder realizar la convolución.

En el caso de las imágenes a color ocurre lo mismo que se ha contemplado en la imagen en escala de grises. Partiendo de la imagen original mostrada, se aplican las mismas máscaras que en el caso anterior, consiguiendo un resultado similar pero con color.

- Tamaño de máscara de 3:

- Tamaño de máscara de 31:

Como en este caso el fondo de la imagen no es blanco, la influencia del borde constante de 0 es más sutil, pero aun así se nota el efecto en la imagen. Es por esto por lo que debe decidirse el borde en función de lo que se quiera hacer, ya que puede influir en futuros resultados.

Por último, para comprobar que la función que se ha implementado muestra resultados correctos se muestra una comparación entre el resultado mostrado por la función implementada (izquierda) y el resultado conseguido con *GaussianBlur* (derecha).

B) Usar la función Laplacian para el cálculo de la convolución 2D con una máscara normalizada de Laplaciana-de-Gaussiana de tamaño variable. Mostrar ejemplos de funcionamiento usando dos tipos de bordes y dos valores de sigma: 1 y 3.

Para este apartado se crea la función *Laplacian* utilizando la función del apartado anterior como base. Su funcionamiento es muy sencillo, ya que, en función del tamaño de máscara que se especifique que se quiere se consiguen dos máscaras distintas de la función Gaussiana: la primera es la segunda derivada de la función en el eje x y la segunda es la segunda derivada de la función en el eje y.

Por tanto, se llama 2 veces a la función *Convolution1D*, una con la máscara de la derivada en x y otra con la máscara de la derivada en y. Estos dos resultados obtenidos se suman y se normalizan multiplicando la suma por el valor de sigma utilizado. Para sacar sigma se utiliza la siguiente función de relación del tamaño de la máscara y sigma:

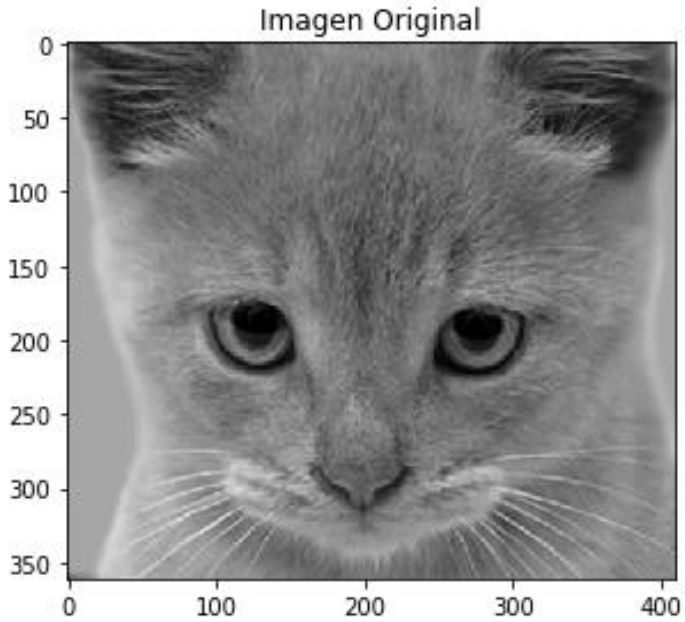
$$tam = 6 * sigma + 1$$

Por tanto, teniendo el tamaño de la máscara o la sigma se puede conseguir la sigma o tamaño utilizado.

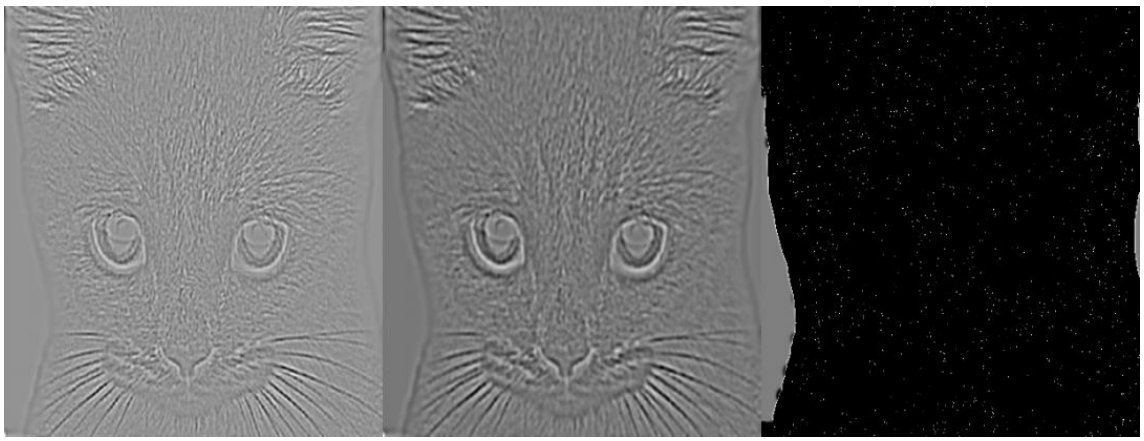
Al igual que en el apartado anterior, se utiliza una combinación de distintos tamaños para probar la función implementada. Estos parámetros son:

- Tamaños de máscara: {7, 13, 19}
- Bordes de la imagen: {Borde reflejado, Borde constante}

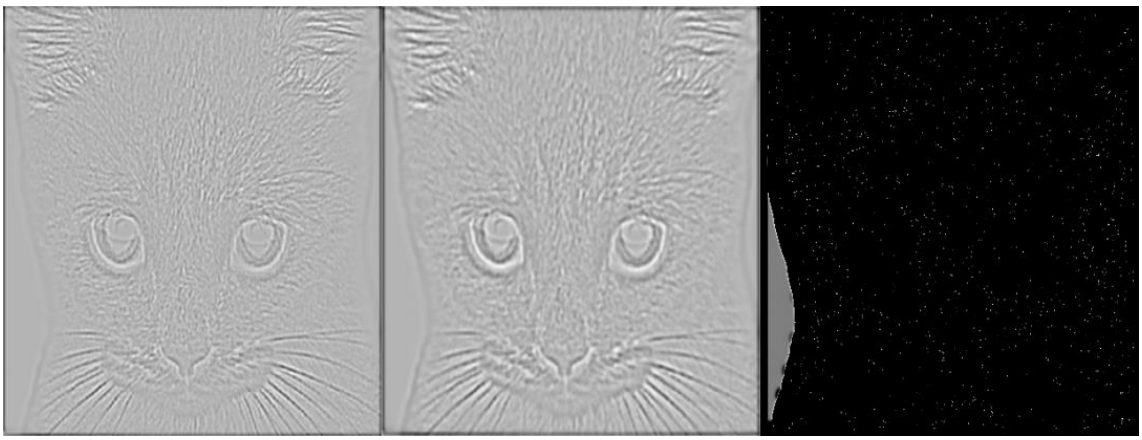
Los resultados conseguidos, en escala de grises y partiendo de la imagen que se muestra a continuación son los siguientes:



- Resultados usando el borde reflejado:



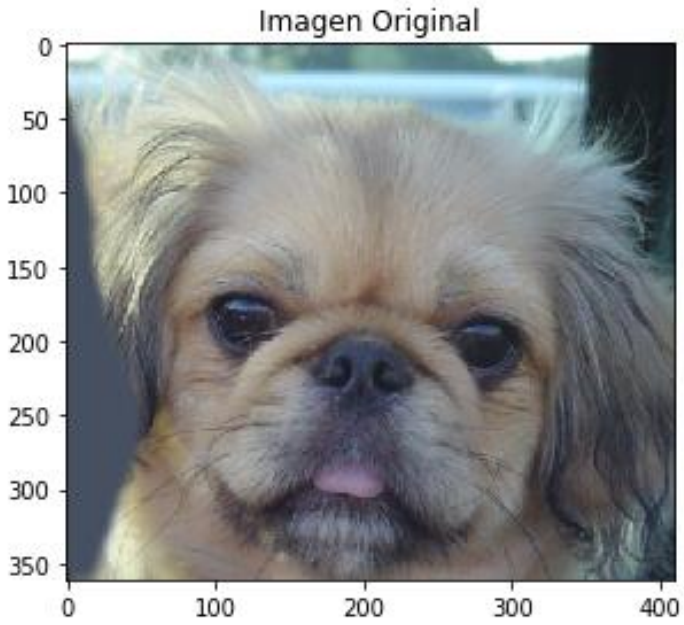
- Resultados usando el borde constante:



En primer lugar, al igual que pasaba en el apartado anterior el borde negro influye en la imagen, ya que se le añade algo que no estaba en la imagen original, pero su uso depende del propósito que se le quiera dar.

En segundo lugar, la última imagen en ambos casos parece ser un resultado incorrecto, ya que con una máscara de tamaño 31 se deberían resaltar los detalles más significativos. Lo que pasa es que al ser una máscara de tamaño tan grande solo se destacan pequeñas zonas en la que detecta cambio y por eso se ve con pequeños puntitos blancos. Si se aprecia bien, se pueden distinguir que tanto los ojos como el hocico están prácticamente negros, y el pelo es lo que realmente destaca con esa máscara. Esto no ocurre en imágenes a color, como se comprueba a continuación.

Si se realiza el mismo experimento con una imagen en color, se obtienen los siguientes resultados, partiendo de la imagen siguiente:

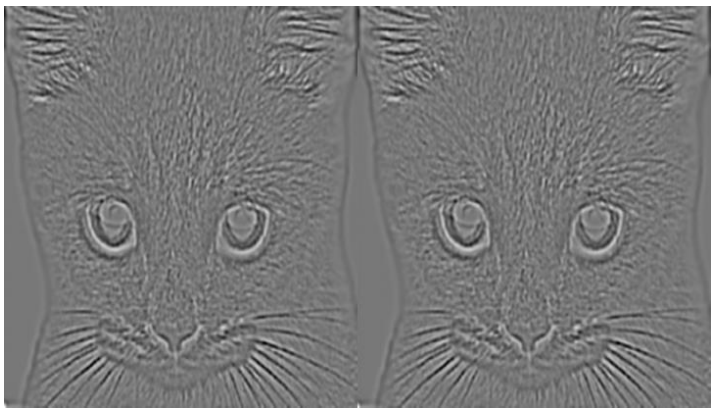


- Resultados usando el borde reflejado:

- Resultados usando el borde constante:

En este caso ocurre lo mismo con los bordes y, además, se aprecia mejor que con el borde reflejado el resultado es más intenso, en cambio, con el borde constante, parece un resultado suavizado. Al aplicar la máscara Laplaciana a la imagen pierde el color ya que lo que se destaca son detalles, cambios de valores.

Como se hizo en el apartado anterior, se utiliza la función de la biblioteca de *OpenCV*, *Laplacian*, para comprobar que los resultados que se obtienen son correctos. A la derecha se muestran los resultados de la función de *OpenCV* y a la izquierda los conseguido con la función propia.



2. IMPLEMENTAR

A) Una función que genere una representación en pirámide Gaussiana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes y justificar la elección de los parámetros.

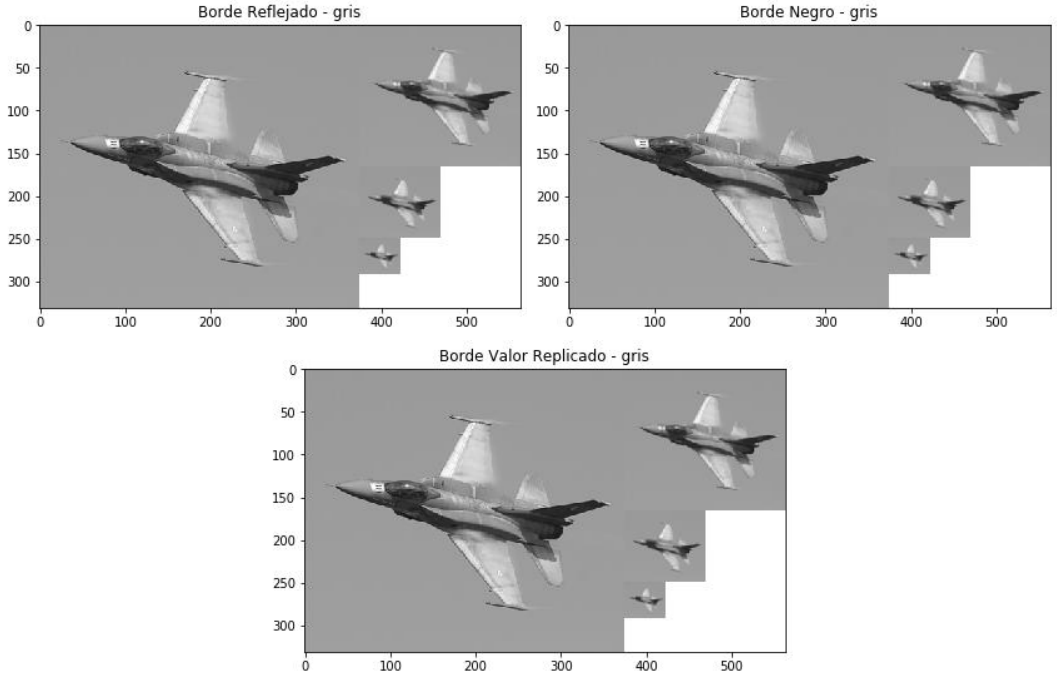
Este ejercicio consiste en construir la pirámide Gaussiana de una imagen. Para ello, se implementa una función, *PiramideGaussiana*, que calcula todos los niveles de la pirámide. Para cada nivel, se calcula las máscaras 1D con la sigma correspondiente, sigma aumenta en k cada escala, y se calcula la convolución de la imagen con esa máscara.

Antes de calcular la convolución, se tiene que interpolar la imagen para reducirla una escala (a la mitad). Para esto se utiliza la función *pyrDown* de *OpenCV*, que al dejarle los parámetros por defecto reduce la imagen una escala. Las imágenes que se tienen al convolucionar la imagen se guardan para poder montar la composición que se utiliza para mostrar la pirámide.

Para pintar la pirámide, tanto la Gaussiana como más adelante la Laplaciana, se copian las imágenes conseguidas en una nueva matriz. Para esto se define una nueva función, *Piramide*, que crea una matriz vacía con tantas filas como filas tenga la imagen original y tantas columnas como el resultado de la suma de las columnas de la primera y segunda imagen de la pirámide. De esta manera se consiguen las imágenes que se muestran más adelante.

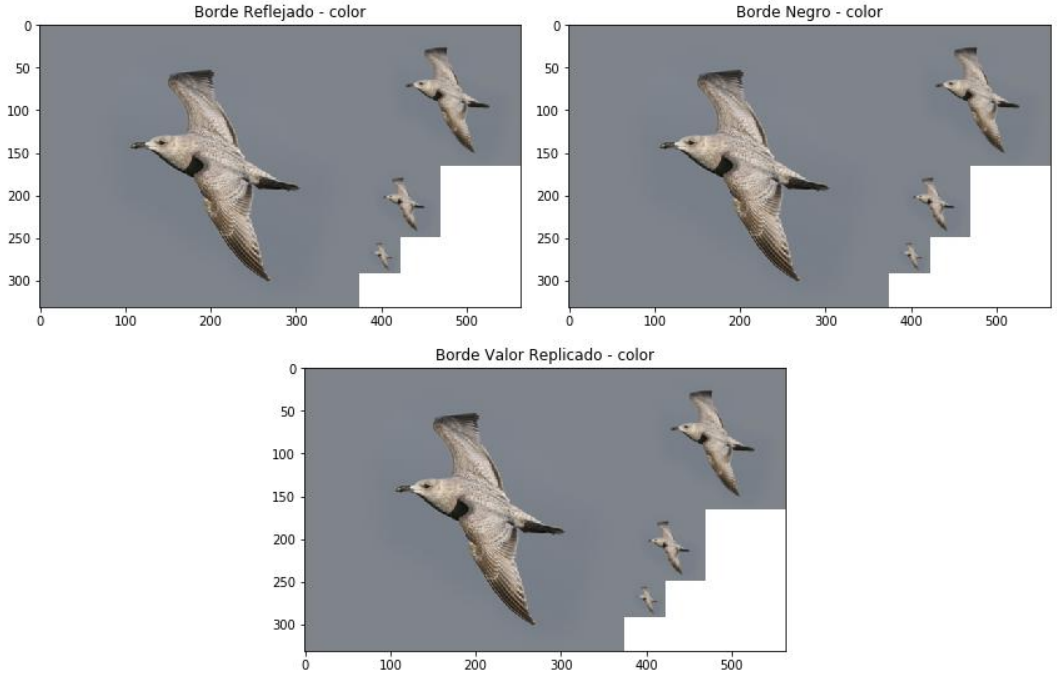
Para comprobar el funcionamiento de las funciones se utiliza un sigma e incremento constante, de 0.1 y 1.2 respectivamente, y se varía el tipo de borde, utilizando un borde reflejado, negro y replicado ([a b c d] → [a a a b c d d d]). Al probar los parámetros se consigue:

- Imágenes en escala de grises:



Como se puede observar, en este caso, y con los valores de sigma asignado, el borde que se le aplica a la imagen no es relevante en el resultado, por lo que en todos los casos se consiguen pirámides equivalentes. Se puede saber que la pirámide está bien montada porque, aunque con menos detalles, la última imagen es *similar*, dentro de su escala, a la imagen original. En cada nivel se pierden más detalles pero se sigue diferenciando el resultado de la imagen.

- Imágenes a color:



En este caso el resultado es similar al caso anterior, pero con imágenes a color. Al igual que pasa con las imágenes en escala de grises, ninguno de los bordes elegidos, por la sigma utilizado, afecta de manera considerable al resultado de la imagen. Además, en este caso es más fácil comprobar que la última imagen, que apena conserva detalles, es similar a la imagen original: la raya negra del cuello, las puntas de las alas y la cola son detalles que siguen pudiéndose contemplar en la última imagen.

B) Una función que genere una representación en pirámide Laplaciana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes.

Este apartado es similar al anterior pero construyendo la pirámide Laplaciana en vez de la pirámide Gaussiana. Para ello, se hace uso de las dos funciones anteriores.

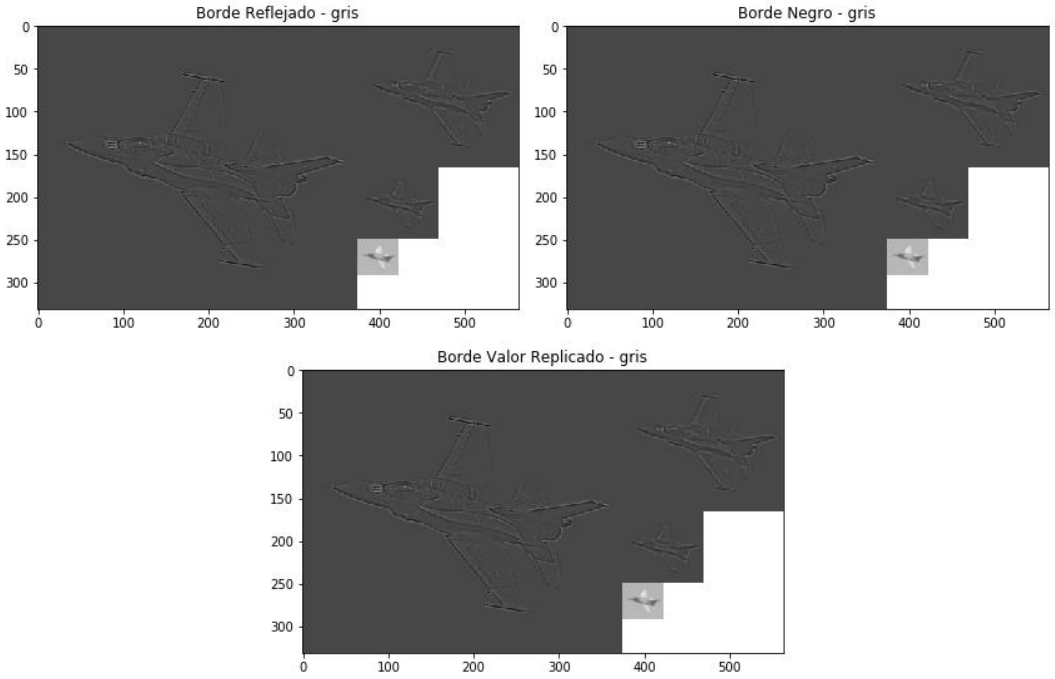
En primer lugar, se crea la pirámide Gaussiana de la imagen, con tantos niveles como niveles se quieren en la pirámide Laplaciana. A partir de ahora, para cada nivel de la pirámide Gaussiana se va a calcular la diferencia con el siguiente nivel, previamente extrapolado. Cada resultado de cada resta de imágenes será un nivel de la pirámide Laplaciana, añadiendo siempre como último nivel de la pirámide las frecuencias bajas de la imagen, que se corresponden con el último nivel de la pirámide Gaussiana.

La pirámide se pinta de la misma manera que se pinta la pirámide Gaussiana. Para hacer la diferencia entre las imágenes de distintos niveles, como al extrapolar salían problemas de dimensiones (aparecía un fila o columna más de lo que debía aparecer porque las imágenes tienes dimensiones impares) se crea la función *DiferencuaGaussiana*. Esta función calcula en cada casilla el valor que le corresponde, ajustando la imagen resultado a las dimensiones más

pequeñas. De esta manera, se evitan problemas al extrapolar, entendiendo que la diferencia no es de más de una fila o columna.

Para probar estas funciones se utilizan los mismos parámetros que en el apartado anterior: sigma inicial de 0.1, incremento de 1.2 en cada nivel, 4 niveles y borde reflejado, negro y replicando el valor. Los resultados conseguidos son los siguientes:

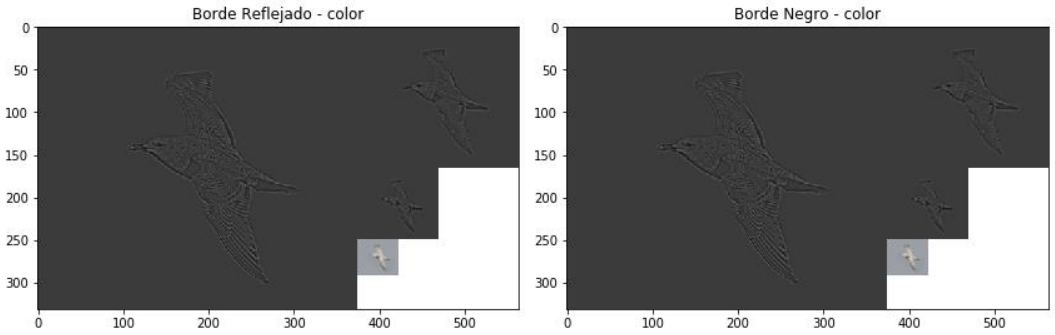
- Imágenes en escala de grises:

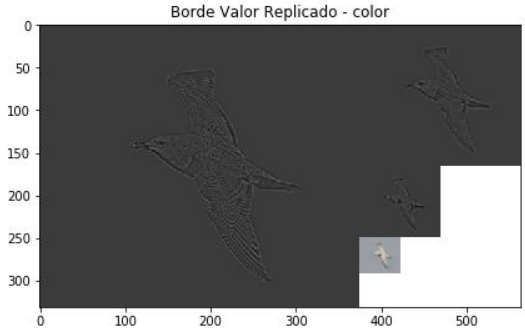


El resultado en este caso es contrario al del apartado anterior. Al hacer la pirámide Gaussiana, en cada nivel se perdían detalles, es decir, se eliminaban las frecuencias altas de la imagen. Con la pirámide Laplaciana, en cambio, se consigue el efecto inverso; las frecuencias altas, los detalles, se mantienen entre los niveles, y lo que se elimina son las frecuencias bajas.

Igual que pasaba en el apartado anterior, aquí tampoco tiene gran influencia el borde que se utilice, a que la sigma es muy pequeña.

- Imágenes a color:





Este caso es idéntico al anterior. En cada nivel, se mantienen solo las frecuencias altas, eliminando las bajas. No hay que olvidarse de añadir como último nivel de la pirámide las frecuencias bajas de la imagen. Si se añaden la imagen original podría reconstruirse, ya que la suma de las frecuencias altas y bajas da la imagen *normal*.

Esa deducción es la que se utiliza a la hora de construir la pirámide Laplaciana. Cada nivel se crea restándole a la imagen sus frecuencias bajas, consiguiendo así una imagen solo con las frecuencias altas, lo que se corresponde con cada nivel de la pirámide Laplaciana.

C) Construir un espacio de escalas Laplaciano para implementar la búsqueda de regiones usando el algoritmo dado.

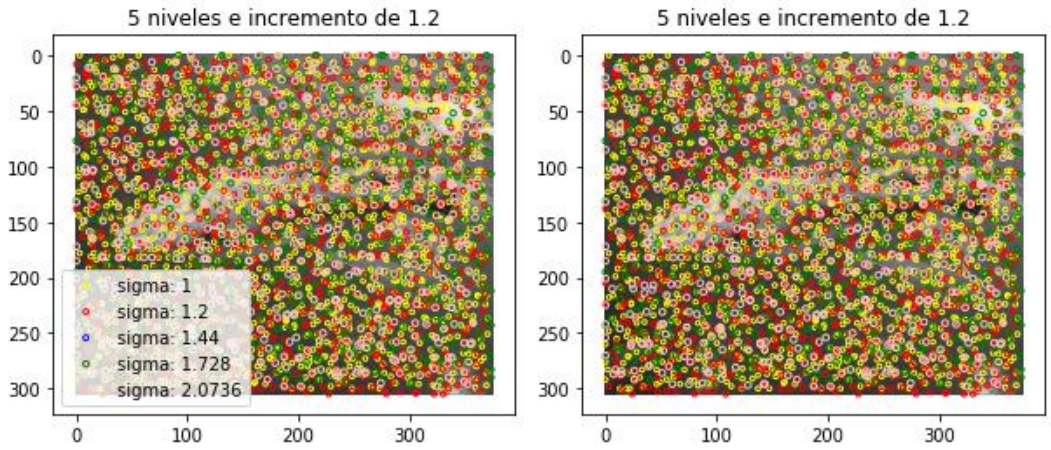
Este ejercicio, grosso modo, consiste en implementar el algoritmo que se da en el enunciado. Para ello, se creó la función *BusquedaRegiones* que se encarga de ejecutarlo con los parámetros pasados como argumento. En cada escala, se calcula la convolución de la imagen con una máscara Laplaciana-de-Gaussiana normalizada, usando la función del ejercicio 1B. A continuación, se estudia la imagen por secciones, en función del tamaño de sigma, y se busca el punto máximo, si hay, del área. Cuando se termina, se incrementa sigma para la siguiente escala.

Cuando se tienen todas las zonas de cada sigma seleccionadas, se pasa a la función *ReducciónNoMaximos*, la cual se encarga de seleccionar, para cada píxel, el valor mayor en todas las escalas, conservando solo ese valor y eliminando el resto de los valores (poniéndolos a 0).

Una vez se tienen los resultados filtrados, se le pasa a la función *PintaAreas* que se encarga de pintar la imagen y representar las áreas seleccionadas en la búsqueda de regiones con una circunferencia, variando el color y tamaño en función de la sigma. Esas áreas que se seleccionan son las que, de primeras, son interesantes para el estudio. Esas áreas deben aún procesarse para seleccionar solo los valores que sean superiores a un umbral determinado.

Para probar esta función, se establece una sigma inicial de 1 y 4 escalas para estudiar. Se van a comparar los resultados conseguidos con un incremento de 1.2 y con incremento de 1.4.

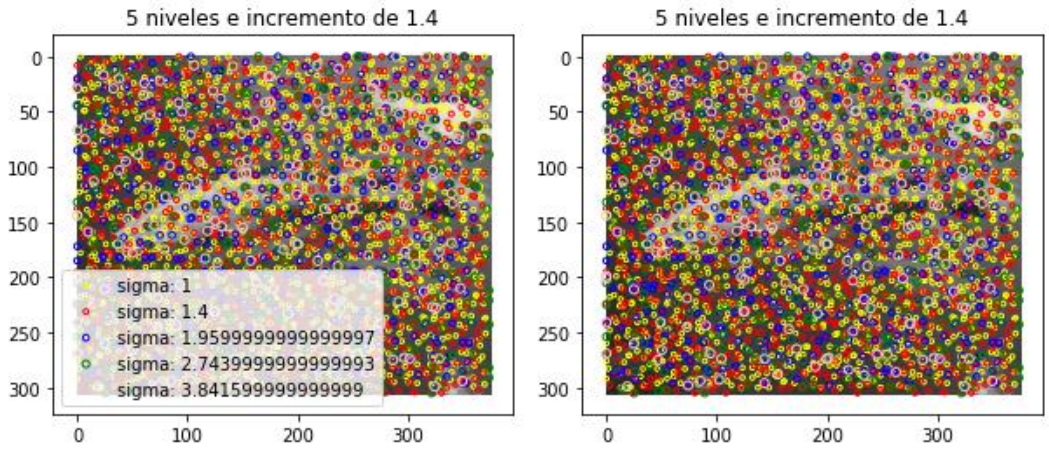
- Incremento de 1.2:



La imagen de la derecha es la misma que la de la izquierda, pero sin leyenda para que pueda verse la imagen al completo.

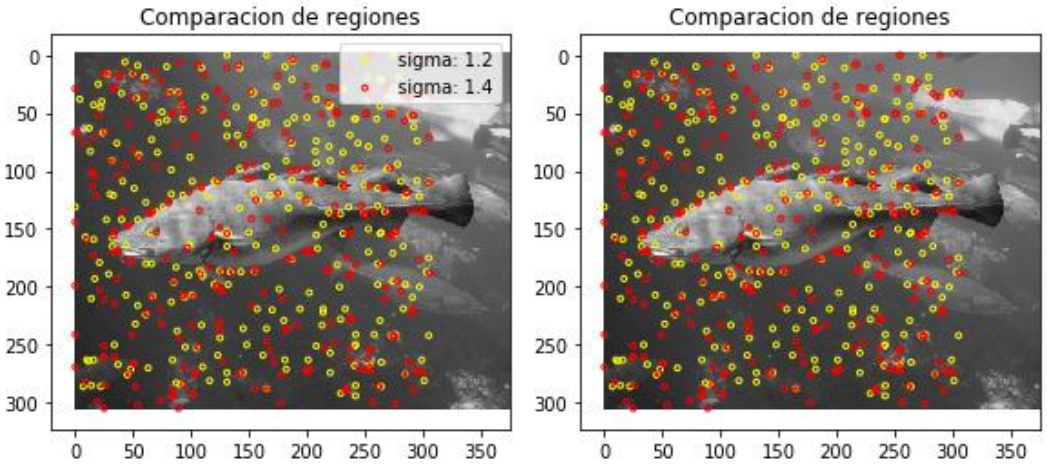
Con este incremento, se consiguen solo 4 niveles, ya que los tamaños de máscara para sigma 1.2 y 1.44 son iguales (por eso no hay circunferencias azules). Se puede observar que donde menos áreas seleccionadas hay es en la superficie del pez, que es más lisa. En cambio, hay muchos elementos en el contorno del pez.

- Incremento de 1.4:



En este caso, si se consiguen 5 niveles distintos, por lo que hay más áreas seleccionadas en la imagen, ya que las sigmas son muy distintas. Estas áreas deben aún que filtrarse usando el umbral, consiguiendo así eliminar muchas de las áreas que de primeras el algoritmo considera importantes. En este caso, el lomo del pez presenta más áreas significativas al usar áreas mayores.

Para comprobar la diferencia entre las áreas conseguidas con cada incremento se crea la función *ComparacionRegiones* que recibe dos listas de regiones y muestra las regiones que no aparecen en las dos. Al ejecutarlo se consigue el siguiente resultado:



Las regiones amarillas son las regiones que se consiguieron con el incremento de 1.2 pero no con el de 1.4, y las rojas al contrario. Quitando regiones aisladas, la mayoría de las áreas están muy próximas y al refinar el algoritmo en prácticas posteriores se solucionaría.

También se observa que donde más peces aparecían, el margen de la derecha, con los dos incrementos se consiguen las mismas áreas. Cuanta menos texturas haya, más diferencia a la hora de escoger regiones se consigue, según los resultados conseguidos y observando el margen izquierdo.

3. IMÁGENES HIBRIDAS

Para llevar a cabo la hibridación de dos imágenes se utiliza una función definida como *Hibridas*. Esta función necesita las imágenes que se van a hibridar y las sigmas que se van a utilizar para las máscaras de cada convolución.

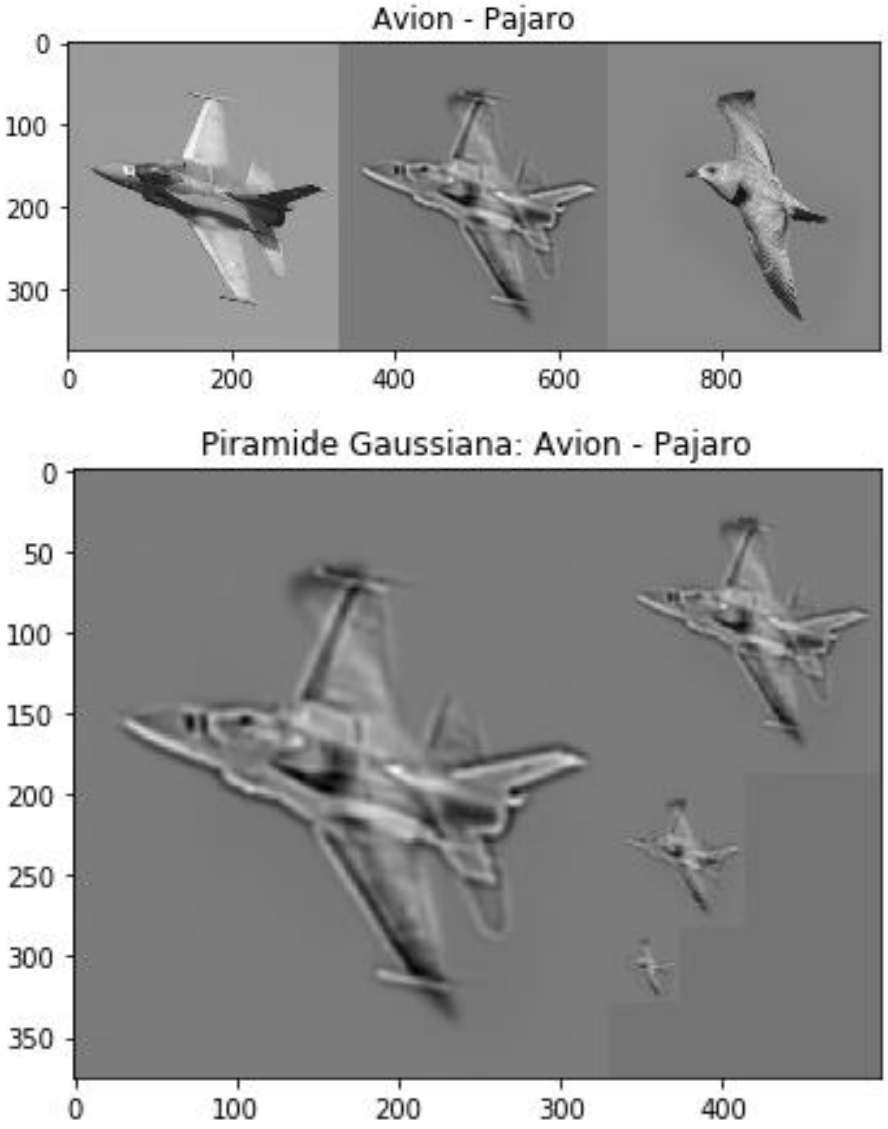
Primero se ajustan las dos imágenes para que tengan el mismo tamaño con la función *resize* de *OpenCV*, indicándole en el parámetro *interpolation* que interpole según el área de cada píxel. Cuando se tienen las imágenes del mismo tamaño, se convoluciona cada imagen con su máscara correspondiente: a la imagen de la que se quiere conservar las frecuencias bajas se le pasa una máscara Gaussiana 1D y a la imagen que conservará las frecuencias altas una máscara Laplaciana-de-Gaussiana. Después de cada convolución se normaliza el resultado conseguido para conseguir una buena hibridación.

Una vez se tiene una imagen que solo conserva las frecuencias bajas (Gaussiana) y otra que solo conserva las frecuencias altas (Laplaciana-de-Gaussiana), se suman píxel a píxel para conseguir la imagen hibridada.

Para comprobar que los resultados son correctos, se utiliza la pirámide Gaussiana. De esta manera, en la imagen más grande debe verse la imagen que se ha decidido que conserve los detalles, la imagen con las frecuencias altas, y en la última imagen la que mantenía las frecuencias bajas.

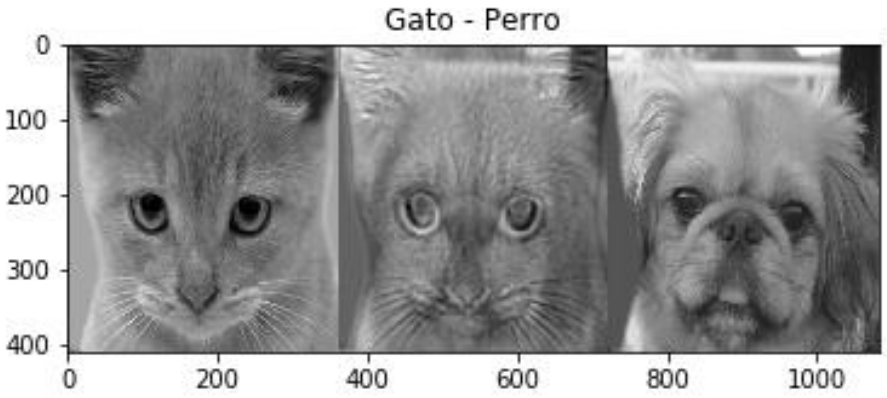
Para probar su funcionamiento se le han pasado 3 conjuntos de imágenes para hibridarlos. Los resultados son los siguientes:

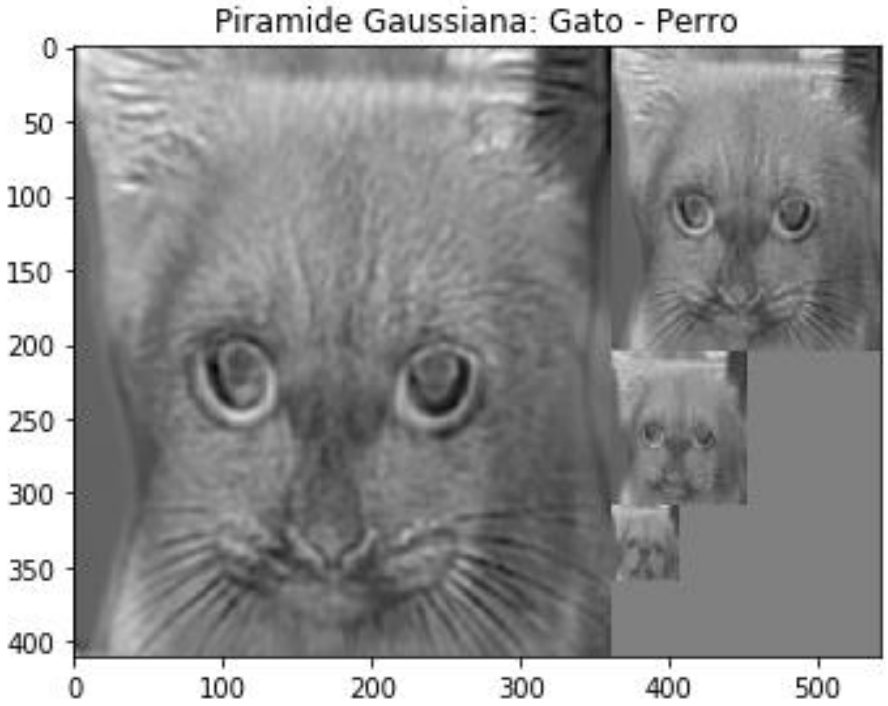
- Conjunto avión y pájaro:



En esta pareja, el avión tiene mucho más detalle que el pájaro, los motores, ventanas, etc. Por eso, se decide que el avión será el que conserva las frecuencias altas y el pájaro las frecuencias bajas. Al montar la pirámide Gaussiana se observa un avión en la primera imagen y un pájaro en la última.

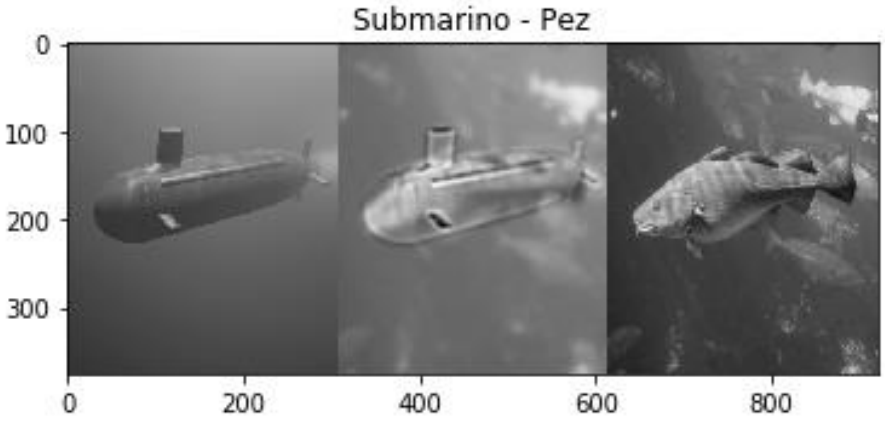
- Conjunto gato y perro:

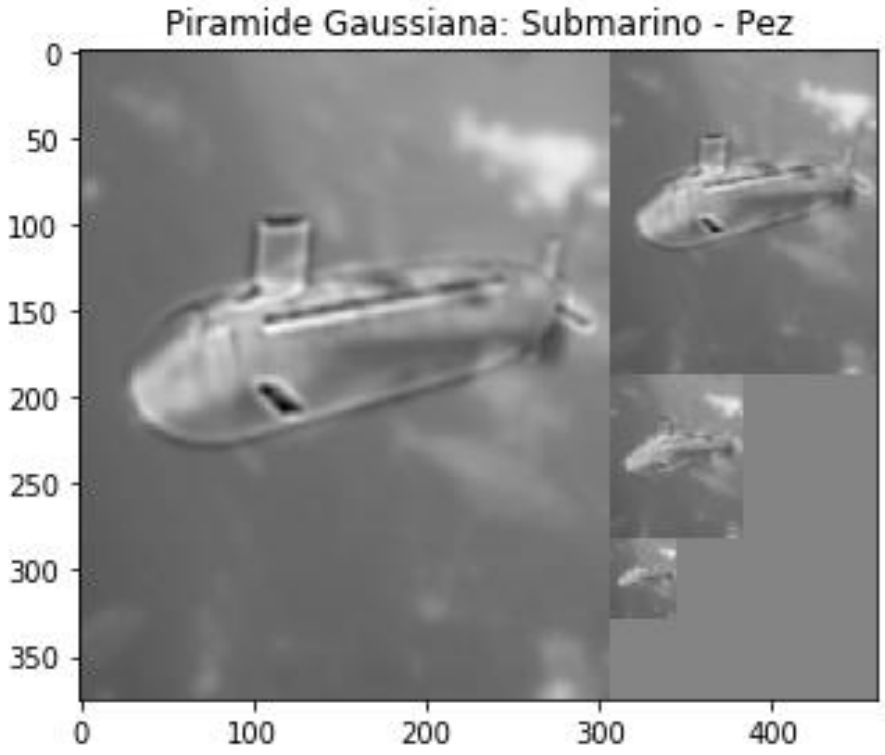




En este caso, el gato tiene más texturas que el perro (el pelo más corto, por lo que parece más, los bigotes o las orejas), por lo que se decide que el gato mantenga las frecuencias altas y el perro las bajas. Al montar la pirámide Gaussiana se observa cómo, efectivamente, la imagen principal es un gato y la última un perro.

- Conjunto submarino y pez:





En esta última pareja, el submarino mantiene las frecuencias altas ya que el pez es mucho más liso. Al repetir el proceso de las otras y montar la pirámide Gaussiana se distingue un submarino en la primera imagen y un pez en la fotografía más pequeña.

BONUS:

2) Realizar todas las parejas de imágenes híbridas en su formato a color.

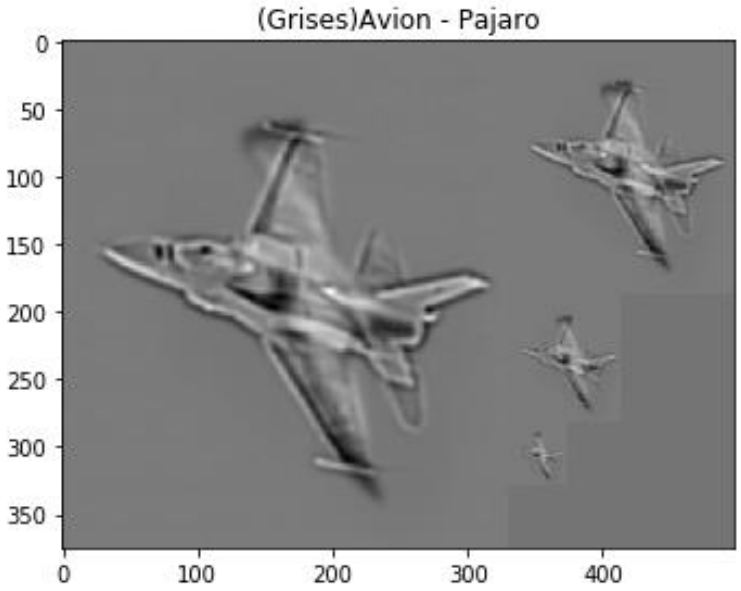
Para realizar este ejercicio no hace falta modificar la función *Hibrida*, ya que está implementada de manera que permita recibir tanto imágenes monobanda (escala de grises) como imágenes tribanda (RGB, a color). Cuando se suman los píxeles de dos imágenes tribanda se suma cada color por separado, es decir:

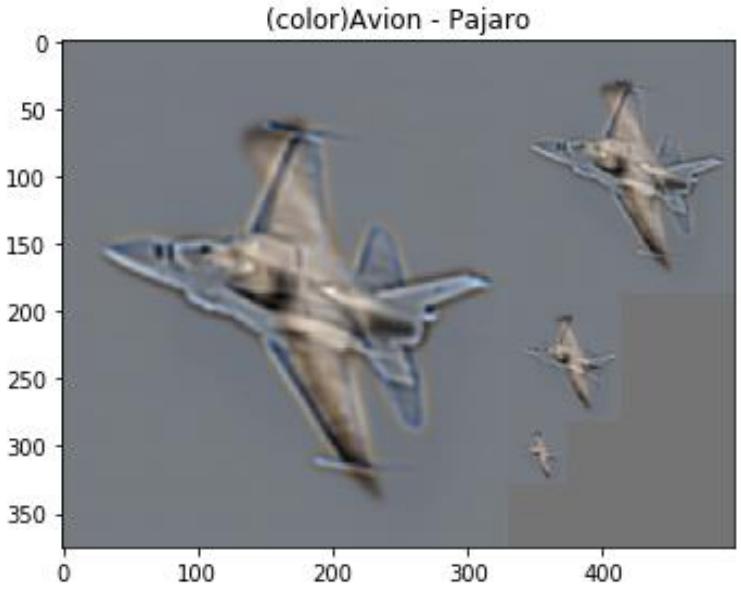
$$[R1\ G1\ B1] + [R2\ G2\ B2] = [R1 + R2\ G1 + G2\ B1 + B2]$$

Por tanto, lo que debe comprobarse es si el resultado que se obtiene es el esperado. Para todas las imágenes, tanto mono como tribanda, se utiliza un tamaño de máscara de 31, para las imágenes que se quiere que conserven las frecuencias altas, y de 21 para las otras. La pirámide Gaussiana, que es el resultado que se muestra, es de 4 niveles y utiliza una sigma inicial de 0.5 con incremento de 0.5.

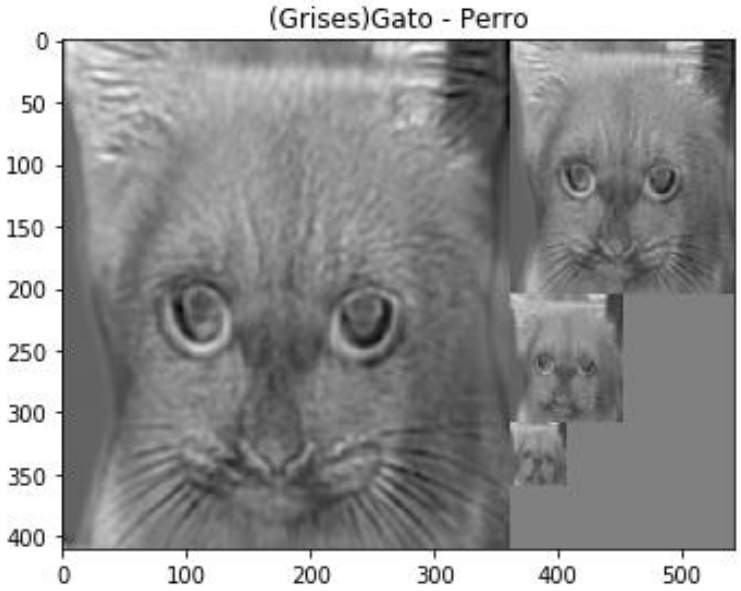
Cada pareja de imágenes se muestra en escala de grises y color a la vez para comprobar resultados. Las imágenes deben mostrarse en grande para poder apreciar de manera correcta las frecuencias altas.

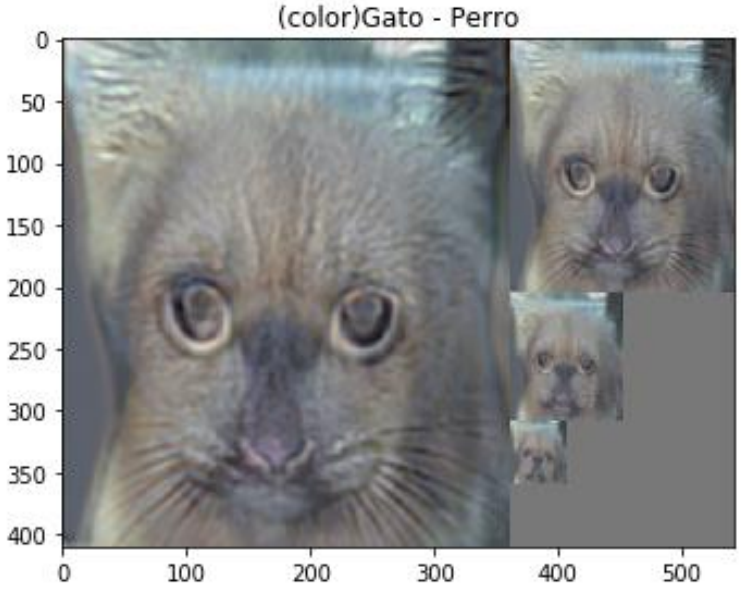
En esta primera pareja queda mejor la imagen en escala de grises porque el pájaro no coincide completamente con el avión (como el caso del morro del avión), cosa que se disimula al pintar en grises. Aun así, en ambos casos se ve en la primera imagen un avión y en la última un pájaro.



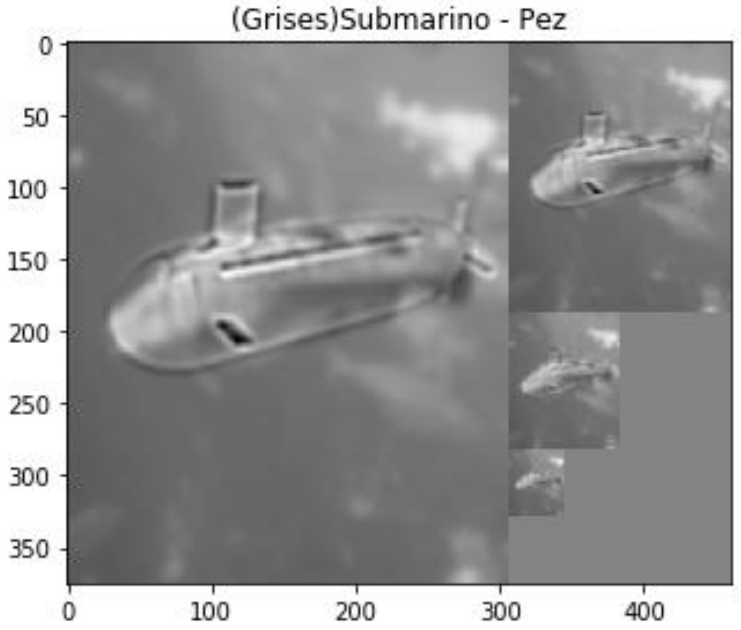


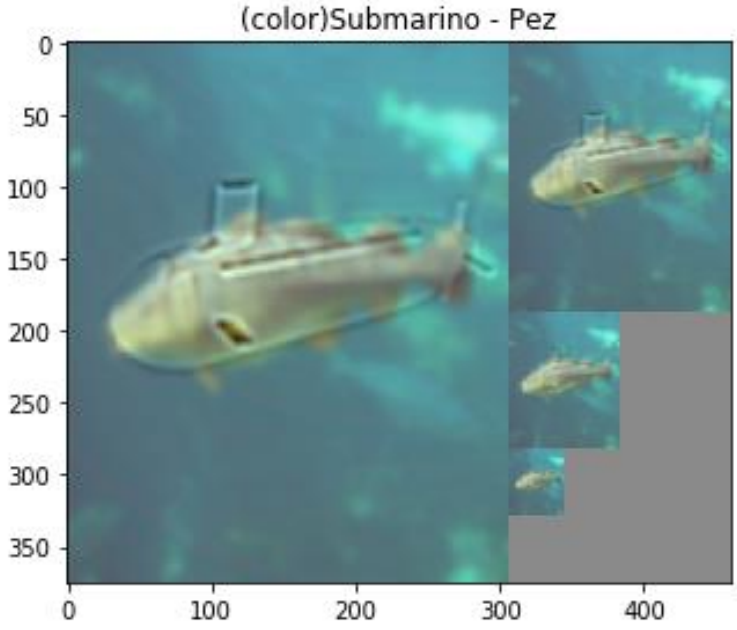
Esta es una de las mejores hibridaciones que se consiguen. Con el color se distingue más el hocico del perro en la primera imagen, pero aun así, en la primera imagen se ve al gato y en la última al perro.





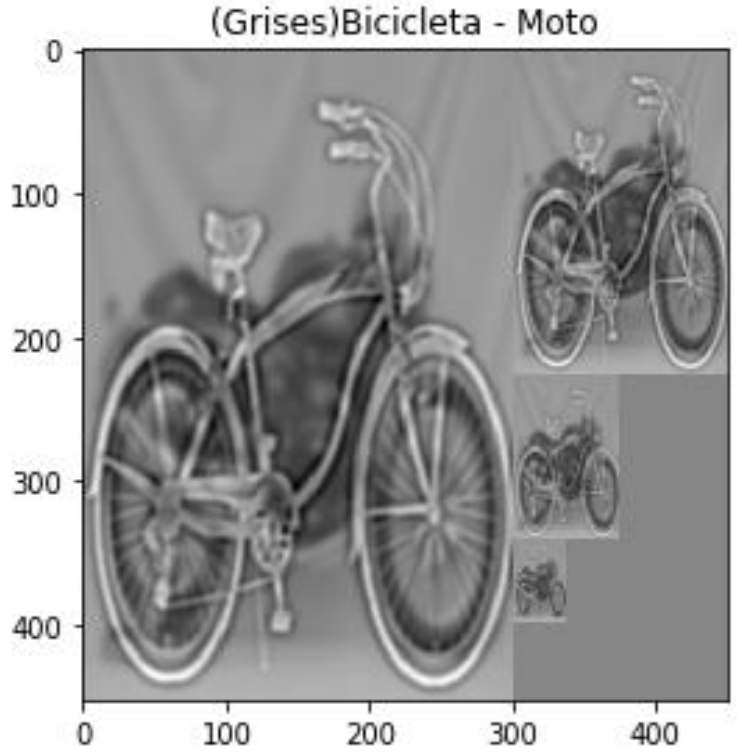
Junto a la hibridación anterior (perro y gato) esta es otra de las mejores combinaciones. El pez coincide bastante bien con el submarino, por lo que no distorsiona mucho la imagen. Además, el fondo de la imagen acompaña a la imagen. Al igual que en las otras, en la primera se distingue mejor el submarino y en la última el pez.

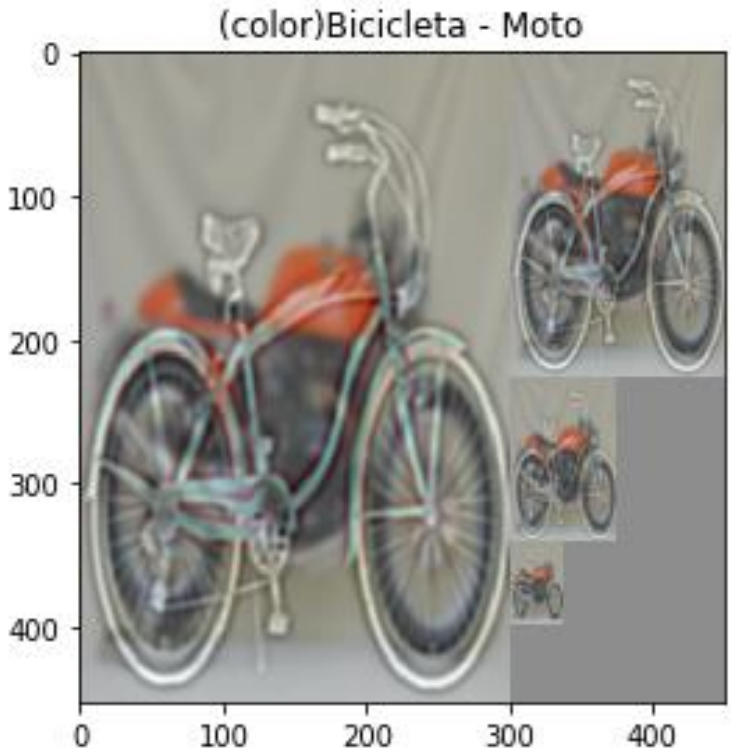




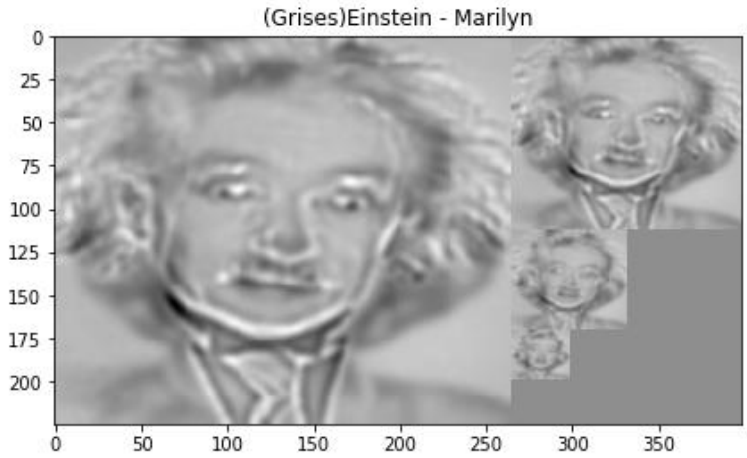
Hibridar la moto y la bicicleta es de las imágenes más complicadas, ya que son bastante distintas. En la primera imagen se distingue la bicicleta, pero también la moto, aunque en la última si que solo se ve la moto (se han eliminado las frecuencias altas). Al hibridarlas en color el resultado es peor, ya que la moto tiene mucha superficie que no se puede ocultar detrás de la bicicleta.

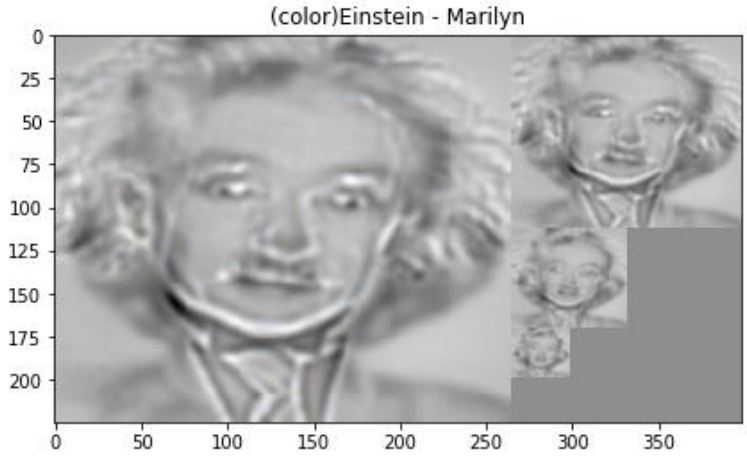
Se probó a invertir la hibridación, es decir, utilizar la moto para las frecuencias altas y la bicicleta para las frecuencias bajas, pero el resultado era mucho peor a causa de la intensidad de los elementos de la bicicleta.





Por último, la hibridación de Einstein y Marilyn es similar en escala de grises y en color porque la imagen original es en escala de grises. Aun así, en la imagen a color la primera imagen está más suavizada que en el caso de la escala de grises. En la primera imagen puede verse a Einstein y en la última, en ambos casos, a Marilyn.




(color)Einstein - Marilyn
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350

REFERENCIAS

Basic Operations on Images. (21 de Octubre de 2019). Obtenido de OpenCV: https://docs.opencv.org/master/d3/df2/tutorial_py_basic_ops.html

DataCamp. (s.f.). *Python For Data Science Cheat Sheet Numpy*. Obtenido de DataCamp: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

image Filtering. (21 de Octubre de 2019). Obtenido de OpenCV: https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6

opencv dev team. (16 de Octubre de 2019). *Image Filtering*. Obtenido de OpenCV: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=pyrup#int%20borderInterpolate(int%20p,%20int%20len,%20int%20borderType)

opencv dev team. (16 de Octubre de 2019). *Operations on Arrays*. Obtenido de OpenCV: https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#void%20flip(InputArray%20src,%20OutputArray%20dst,%20int%20flipCode)

OpenCV Maintainers. (s.f.). *enum cv::InterpolationFlags*. Obtenido de https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/enum_cv_InterpolationFlags.html#doxid-da-d54-group-imgproc-transform-1gga5bb5a1fea74ea38e1a5445ca803ff121ac97d8e4880d8b5d509e96825c7522deb

OpenCV Python Image Smoothing – Gaussian Blur. (2018). Obtenido de TutorialKart: https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/

Python cv2.filter2D() Examples. (s.f.). Obtenido de ProgramCreek: https://www.programcreek.com/python/example/89373/cv2.filter2D

Python cv2.getGaussianKernel() Examples. (s.f.). Obtenido de ProgramCreek: https://www.programcreek.com/python/example/89346/cv2.getGaussianKernel

Python cv2.Laplacian() Examples. (s.f.). Obtenido de ProgramCreek: https://www.programcreek.com/python/example/89339/cv2.Laplacian

Turmero, P. (s.f.). *Transformaciones geométricas en OpenCV*. Obtenido de monografias.com: https://www.monografias.com/trabajos108/transformaciones-geometricas-opencv/transformaciones-geometricas-opencv.shtml