UNIVERSIDAD
DE GRANADA

Práctica 1.
Filtrado y Detección de regiones

Lukas Häring García

October 22, 2019

Tabla de contenidos

1 Primer Apartado

En este apartado vamos a realizar unos ejercicios relacionados con las máscaras gaussianas y las laplacianas-gaussianas estudiadas en clase, con diferentes ejemplos de imágenes, bordes y sigmas.

1.1 Máscaras Gaussianas

Una máscara gaussiana es un operador de convolución que se utiliza para suavizar superficies, esta es obtenida a través del muestreo de una función gaussiana.

Vamos a convolucionar nuestra imagen con una máscara gaussiana 2D. Para ello, basta con utilizar el método **GaussianBlur** de la librería **cv2**.

Este necesita como parámetros, la imagen a aplicar; el segundo parámetro es un par de valores para el tamaño del kernel (El cual no utilizaremos ya que lo haremos dependiente del sigma); tercer y cuarto parámetro, el valor de los sigmas horizontal y vertical respectivamente y finalmente el tipo de borde que queremos aplicar, véase la referencia [1] para obtener más información sobre el tipo de bordes.

```
cv2.GaussianBlur(img,None,sigma,sigma,cv2.BORDER_CONSTANT)
```

Podemos obtener el mismo resultado a través de dos convoluciones 1D, para ellos, vamos a obtener el kernel 1D utilizando la función de cv2 **getGaussianKernel**, este nos pedirá como argumentos: El tamaño del kernel y el sigma. El tamaño del kernel es calculable a partir del sigma.

$$\#\text{Kernel} = 2 \cdot \lfloor 3\sigma \rfloor + 1$$

Esta ecuación es deducible a través del tamaño del intervalo y de la necesidad de muestrear un $\geq 95\%$ de la curva.

Para convolucionar ambos filtros 1D, utilizo el método diseñado por mí, que también es parte del **Bonus 1** y será explicado posteriormente. Este calcula eficientemente dicha convolución, recibe el nombre de **conv_1D_1D**. **conv_1D_1D** necesita como argumentos, la imagen y una matriz con las dos máscaras 1D. Además hace uso del principio de localidad, por lo que la primera convolución (horizontal) la realiza como tal y la convolución vertical, a la imagen transpuesta (y su posterior transposición). Esta función no devuelve el resultado normalizado, por lo que habrá que normalizarlo $[0, 255]$ **normalize** (Véase el pie de página).

```
gaussian_2_1d_cv2 = normalize(conv_1D_1D(img, kernel))
```

1.1.1 Resultados

A la izquierda el método **GaussianBlur** y a la derecha, mi función de convolución **conv_1D_1D**.

Fig. 1: **image** = "fish.bmp", **border** = "constant", $\boldsymbol{\sigma} = 2.0$

Fig. 2: **image** = "cat.bmp", **border** = "reflect", $\boldsymbol{\sigma} = 4.0$

Fig. 3: **image** = "bicycle.bmp", **border** = "replicate", $\boldsymbol{\sigma} = 6.0$

1.1.2 Valoración

Podemos observar una clara diferencia entre ambas imágenes, algo es extraño ya que **GaussianBlur** para "bicycle.bmp" y para "cat.bmp" parece no aplicar correctamente el filtro gaussiano y solo horizontalmente. Por otro lado, mi método consigue aplicar correctamente un difuminado homogéneo en la bicicleta. Además se observa que en la primera imagen, mi método "arrastra" el padding hacia el centro de la imagen, creando un difuminado obscuro al rededor de los bordes. Está claro que a medida que vamos subiendo $\boldsymbol{\sigma}$, este va difuminando cada vez más, por lo que los resultados, parecen ser coherentes.

1.2 Filtro Laplaciano-Gaussiano

Un filtro laplaciano-gaussiano es utilizada para marcar regiones con alto cambio de intensidad, por ello, es utilizado habitualmente para detectar bordes. Además es isotrópica, quiere decir que afecta por igual en todas direcciones.

La laplaciana-gaussiana, como su propio nombre indica, requiere de dos pasos. Aplicar un **filtro Gaussiano** de suavizado (utilizado para eliminar ruido que pueda afectar) y un **filtro Laplaciano**, que viene dado por la siguiente ecuación.

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Como podemos apreciar, la derivada lo que va a ver es el cambio en cada dirección (horizontal y vertical), de ahí su detección de bordes.

Para obtener filtros de suavizado y las segundas derivadas espaciales, vamos a utilizar el método de cv2 **getDerivKernels**.

```
d2x = cv2.getDerivKernels(2, 0, tam)
d2y = cv2.getDerivKernels(0, 2, tam)
```

Este nos devolverá un filtro de Sobel[3], es decir, una máscara 1D de suavizado direccional y otro de derivada espacial, estos los aplicaremos con mi método **conv_1D_1D** para cada dirección y luego sumaremos los resultados.

Cabe destacar que los filtros no están normalizados, por lo que deberemos multiplicar al resultado final por $\boldsymbol{\sigma}$. Además, el resultado no estará normalizado en 255, por lo que a la hora de mostrarlo, habrá que normalizar.

1.2.1 Resultados

A la izquierda el laplaciano-gaussiano sin valor absoluto. A la derecha, aplicando el valor absoluto.

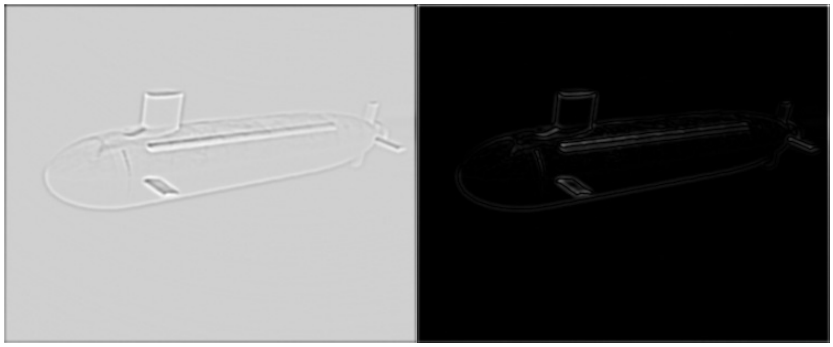

Fig. 4: **image** = "submarine.bmp", **border** = "constant", $\boldsymbol{\sigma} = 1.0$

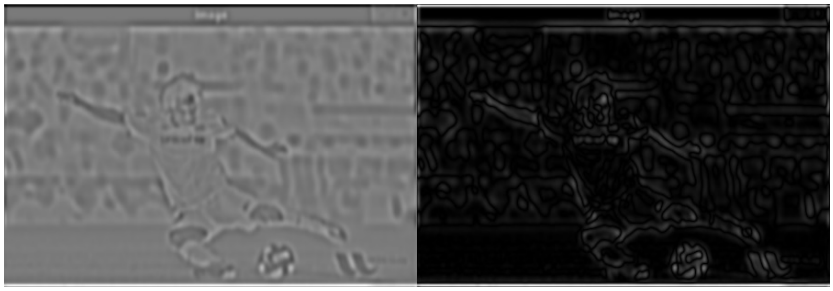

Fig. 5: **image** = "messi.jpg", **border** = "wrap", $\boldsymbol{\sigma} = 2.0$

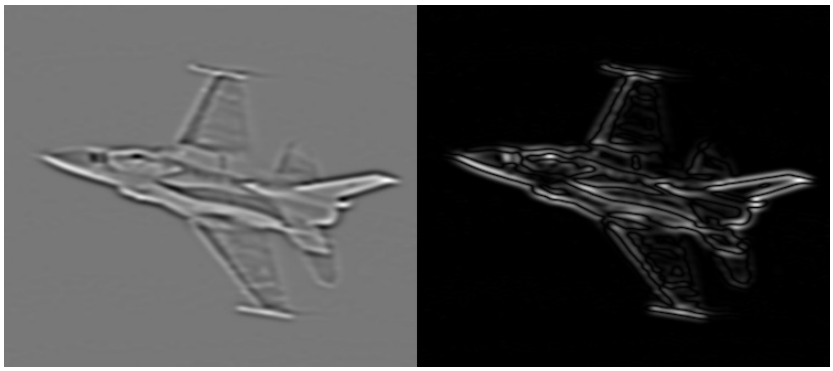

Fig. 6: **image** = "plane.bmp", **border** = "replicate", $\boldsymbol{\sigma} = 3.0$

1.2.2 Valoración

El filtro Laplaciano-Gaussiano, como ya hemos comentado, suaviza por lo que vamos a eliminar información útil o no (ruido).

Es curioso observar que al aplicar el borde "constant", este tome una intensidad baja, esto ocurre ya que al aplicar las derivadas, estos puntos se detecten como bordes ya que su exterior es cero. Podríamos decir que se obtienen "mejores" resultados con cualquier otro borde.

Además, el sigma vemos que afecta a la intensidad de los bordes así como al difuminado de estos, por lo que, podríamos afirmar que debemos utilizar un valor intermedio, para no generar bordes muy borrosos pero tampoco demasiado claros.

Algo a comentar es que el filtro gaussiano utiliza el mismo sigma que el filtro Laplaciano, algo que se podría evitar añadiendo un argumento más, pero es solo un detalle de implementación.

2 Segundo Apartado

En esta sección vamos a ver las pirámides gaussianas y laplacianas. También el llamado **blob detection**, detección de zonas interesantes.

2.1 Pirámide Gaussiana

La pirámide gaussiana es utilizada para escalar imágenes sin que pierda información vital de la imagen, ya que utilizando solo interpolación, suele perder, puntos clave, dirección de lineas, etc.

Además la pirámide gaussiana es utilizada por la GPU para generar texturas con distinto nivel de detalle, conocido como *LOD Textures*[4] o Mipmaps.

Mi método **gauss_pyramid** hace uso de cuatros parámetros: el primero, la imagen; el número de escalas (Se cuentan n + 1 escalas) como segundo argumento; El valor de sigma y por último, el tipo de borde (por defecto, borde constante). Este insertará la primera imagen como guía para las demás.

```
gauss_pyramid(img, 4, sigmam, cv2.BORDER_CONSTANT)
```

Además, hace uso de los métodos **gauss_pyramid_helper** y **GSharp**, que se explicarán a continuación. En primer lugar, el método genera una image con la misma altura que la imagen que pasamos y un ancho de 1.5 veces más.

GSharp un método que devuelve la imagen suavizada y luego re-escalada al tamaño (width, height). El suavizado es realizado a través del método **GaussianBlur** y el re-escalado a través de la función de cv2, **resize** que utiliza como tipo de interpolación **INTER_LINEAR** de **cv2**.

```
filter = GSharp(sourc, height, width, sigma);
```

gauss_pyramid_helper Es un método recursivo que requiere la imagen resultante creada por **gauss_pyramid**, la imagen a re-escalar recursivamente; el desplazamiento horizontal y vertical, utilizado para posicionar correctamente cada imagen; el nivel actual de profundidad, el máximo nivel y el sigma que se va a aplicar.

2.1.1 Resultados

Fig. 7: **image** = "motorcycle.bmp", **border** = "reflect", $\boldsymbol{\sigma} = 1.0$

Fig. 8: **image** = "messi.jpg", **border** = "constant", $\boldsymbol{\sigma} = 2.0$

2.1.2 Valoración

Para la *Fig. 7* se ha utilizado un sigma pequeño $\boldsymbol{\sigma} = 1.0$, ya que este contiene bastantes detalles que no queremos perder a la hora de suavizar mucho, vemos que para cada escala obtenemos una imagen con detalles suficientes, incluso la rueda y los cables se pueden notar suficientemente.

La segunda imagen *Fig. 8*, tenemos que el sigma utilizado es demasiado alto, ya que el jugador en cada escala parece acoplarse aún más al fondo, por lo que, en mi opinión el sigma debería ser más bajo, que a diferencia del anterior, hasta la última imagen puede diferenciarse qué objeto es.

Podemos ver que el borde no afecta al resultado, por lo que cojamos cual cojamos, no va a implicar un gran cambio entre las escalas.

2.2 Pirámide Laplaciana

La pirámide laplaciana es una técnica utilizada para comprimir imágenes junto con una buena compresión de bits de frecuencias altas, reconocer patrones, etc. Como hemos visto en las diapositivas, la imagen es escalada la mitad (1 octava), previamente se suavizada. Ahora se aplica la operación inversa, es decir, se suaviza y se escala hacia arriba. Una vez tenemos las dos imágenes, se aplica el operador de sustracción sobre la original, obteniendo así, los contornos de la imagen. Este proceso es realizado para cada imagen que hemos re-dimensionado.

De manera similar a la pirámide gaussiana, necesitamos de dos funciones, la especificada anteriormente, **GSharp** y una de ayuda, **laplacian_pyramid_helper**. Almacenamos en una variable la imagen re-escalada a la mitad, para así poder enviarla para la siguiente iteración.

```
width = np.uint(img.shape[1] * 0.5)
heihgt = np.uint(img.shape[0] * 0.5)
filtering = GSharp(img, width, height, sigma, border)
```

Ahora, obtenemos el resultado de la iteración (contornos), aplicando el proceso inverso sobre la imagen original.

```
width = img.shape[1]
heihgt = img.shape[0]
img -= GSharp(filtering, width, height, sigma, border)
img = normalize(abs(img) if absolute else img, True)
```

Como vemos, hemos normalizado la imagen de forma completa, podemos darle valor absoluto para así tener una vista más clara de los bordes.

Cabe destacar que se han omitido muchos pasos ya que son equivalentes a la pirámide gaussiana que ha sido explicada en el apartado anterior.

2.2.1 Resultados

A la izquierda, la pirámide laplaciana con valor absoluto. A la derecha, sin el valor absoluto. Ambos interpolan como está especificado en el método **GSharp**.

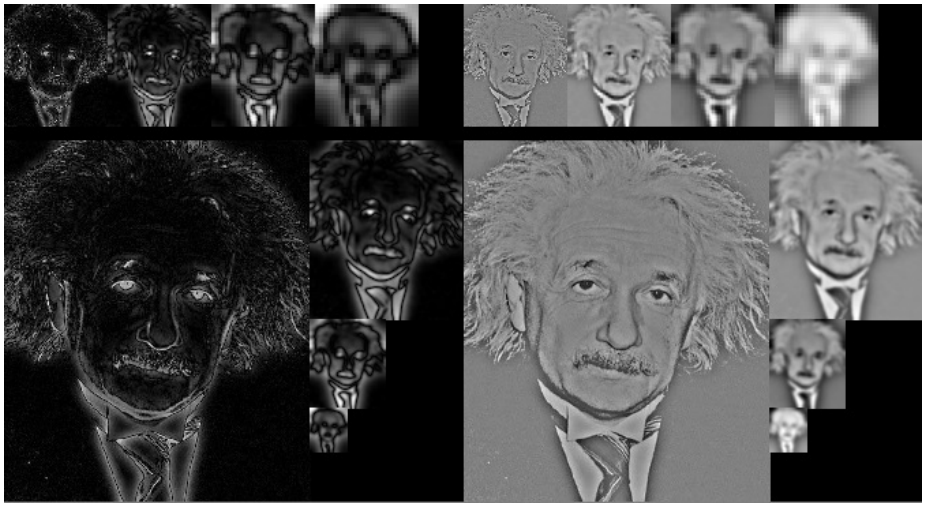

Fig. 9: **image** = "einstein.bmp", **border** = "constant", $\boldsymbol{\sigma} = 2.0$

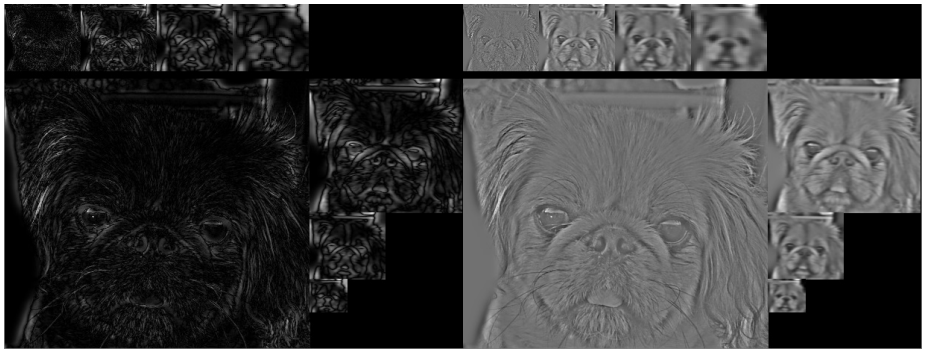

Fig. 10: **image** = "dog.bmp, **border** = "replicate", $\boldsymbol{\sigma} = 3.0$

2.2.2 Valoración

Podemos observar un resultado similar a la laplaciana-gaussiana. Como para cada escala vamos suavizando, cada vez perdemos más información (útil o no). Además podemos apreciar que se acentúan unos bordes en unas escalas y otros en otras escalas.

Para la primera imagen, observamos que con un $\boldsymbol{\sigma} = 2.0$, la primera iteración da mucho ruido al rededor del pelo, pero a medida que iteramos, ese ruido desaparece. Vemos que la corbata, los ojos y el bigote están siempre presentes, eso quiere decir que tienen una alta intensidad por lo que son los elementos que más destacan.

La imagen del perro tiene un efecto parecido a la anterior, la imagen tiene mucho ruido debido al pelo, pero de la misma forma, cada octava reduce aún más ese ruido. Además el elemento que más podemos ver que es destacado son los ojos y el hocico en más iteraciones.

Cabe destacar que el tipo de borde no parece afectar al resultado, por lo que da igual el borde que apliquemos, podría omitirse o utilizar el borde constante.

Para finalizar, el último nivel de la pirámide laplaciana (no absoluta) concuerda con el nivel último nivel de su pirámide gaussiana, esto ocurre ya que ambos tienen las frecuencias bajas, que son utilizadas en la pirámide laplaciana para formar la imagen original.

2.3 Blob detection

Como su propio nombre indica, es utilizado para obtener patrones en una imagen. En nuestro caso, vamos a detectar regiones circulares utilizando el sombrero mejicano (Función Laplaciana).

Una cualidad de la función laplaciana que nos permite detectar zonas circulares, es la importancia que le da a los valores interiores a la circunferencia y al centro de la región, junto con los extremos (positivos de la función) que "buscan" los bordes.

El algoritmo consta de tres partes, la primera parte, aplica el filtro laplaciano-gaussiano incrementando sigma por un factor para cada escala y normaliza por fronteras. En el segundo nivel, se van a aplicar la supresión de no-máximos para cada escala, se va a utilizar de la librería *numpy*, la función **np.amax** que devuelve el valor máximo de una matriz. Además, para evitar aplicar una segunda pasada, se aplica un filtro de paso alto para eliminar así frecuencias bajas, un toque artístico que puede eliminar regiones. Por último, la supresión de no-máximos por las capas vecinas, es decir, por cada máximo, mira los vecinos de la capa superior e inferior y si este no es máximo, lo elimina.

La implementación ha sido optimizada de la siguiente forma, cada vez que se han encontrado frecuencias máximas que han pasado el filtro, se introducen en un vector junto con su posición x e y. Basta con recorrer su vector de escala de máximos e ir comprobando si estos son máximos sobre los vecinos de los máximos de las escalas superiores. Además, la capa inferior comprueban de la capa de encima y la superior, la de debajo. Las capas intermedias, comprueban tanto de arriba como de debajo.

En el código, esta función recibe el nombre de **blob_detection** y requiere de los siguiente parámetros: La imagen a buscar regiones; el número de escalas; el sigma inicial; un coeficiente, que debe estar entre el intervalo $[1.2, 1.4]$ y el umbral del filtro de paso alto.

Podemos calcular cual será el factor máximo que podemos asignar antes de que *cv2* nos de una excepción debido a que no ha podido generar el kernel para realizar la función **laplacian_gauss**. Conocemos que el tamaño máximo que soporta es 31. Denotemos c como el coeficiente de incremento de sigma que desconocemos y queremos calcular para que no de una excepción, σ_0 como el valor de sigma inicial, n el número escalas y T el tamaño de la máscara.
Existe una relación entre el *sigma* y el valor del tamaño, $T = 2 \cdot \lfloor 3\sigma \rfloor + 1$.

$$T \leq 31 \Leftrightarrow 2 \cdot \lfloor 3\sigma \rfloor + 1 \leq 31 \Leftrightarrow \sigma \leq 5$$

Observar que para la última escala "n", el valor de sigma es: $\sigma_n = \sigma_0 \cdot c^n, n \geq 1$.

$$\sigma_n \leq 5 \Leftrightarrow \sigma_0 \cdot c^n \leq 5 \Leftrightarrow c^n \leq \frac{5}{\sigma_0} \Leftrightarrow c \leq \left(\frac{5}{\sigma_0}\right)^{\frac{1}{n}}$$

Vemos que el valor máximo que puede obtener el coeficiente es $c = \left(\frac{5}{\sigma_0}\right)^{\frac{1}{n}}$, con esto, podemos saber si el factor calculado entra o no en nuestro intervalo $[1.2, 1.4]$ en la última escala.

El radio de las circunferencias que dibujamos en cada escala es el valor que corta el sombrero $g(x, y)$ con el eje, es decir dónde $g(x, y) = 0$, como la parte exponencial no puede valer 0, por lo que $(x^2 + y^2 - 2\sigma^2) = 0$, vemos que es la ecuación de una circunferencia de radio $\sigma\sqrt{2}$ (Nota: $x^2 + y^2 = r^2$).

2.3.1 Resultados

A la izquierda, blob-detection con una frecuencia de corte baja f_1. A la derecha, una frecuencia de corte ajustada f_2.

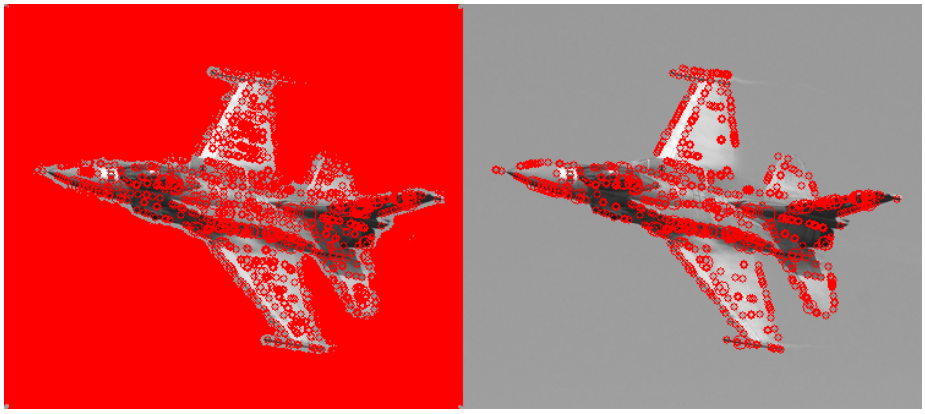

Fig. 11: **image** = "plane.bmp", 3 escalas, $c = 1.4$, $f_1 = 50$, $f_2 = 150$, $\boldsymbol{\sigma} = 2.0$

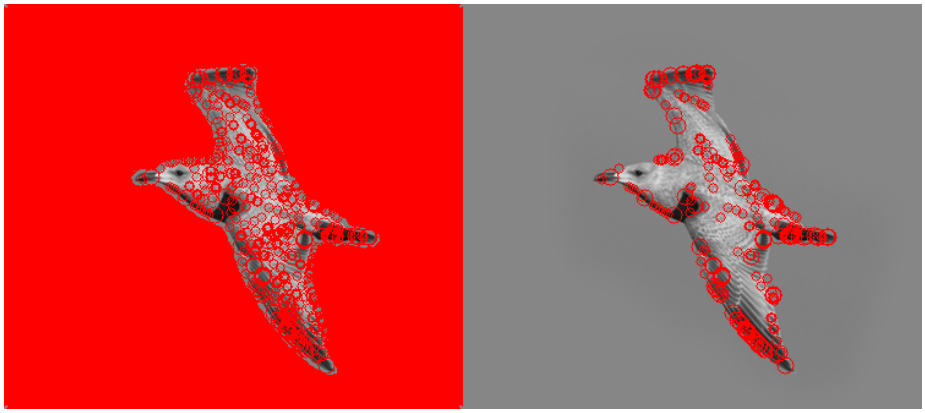

Fig. 12: **image** = "bird.bmp", 4 escalas, $c = 1.2$, $f_1 = 50$, $f_2 = 130$, $\boldsymbol{\sigma} = 3.5$

2.3.2 Valoración

Podemos observar que los resultados aparentemente parecen correctos, se detectan regiones de diferente radio, aunque el radio sea pequeño, es suficiente para observar que obtiene regiones interesantes.

Si la frecuencia de corte es demasiado baja f_1, observamos que no hemos eliminado información suficiente, en el ejemplo primero "Fig. 11", observamos que el fondo queda detectado ya que todos son máximos. Si la frecuencia de corte es apta, como podemos ver en la imagen de su derecha, conseguimos eliminar todas estas frecuencias. Vemos que en la imagen de la derecha, hemos detectado contorno, la zona de la cabina del viajero, las alas anteriores y posteriores.

En la segunda imagen, obtenemos un resultado similar que la primera, como no hemos ajustado un umbral correcto, el fondo es detectado como máximos. Ajustada a una frecuencia de corte más alta, obtenemos mejores resultados, vemos que el pico de la gaviota se detecta, así como zonas de su cuerpo con plumas, bordes de las alas y las zonas en sombra que también son detectadas.

Vemos que es muy importante ajustar correctamente los sigmas y la frecuencia de corte, estos serán decisivos a la hora de obtener regiones correctas (a nuestro parecer).

3 Tercer Apartado

En este apartado, vamos a realizar las imágenes híbridas, este tipo de imágenes tienen en sí, otras dos imágenes, perceptibles según la distancia del observador, esto es porque el ojo detecta frecuencias altas cuando está cerca y frecuencias bajas cuando está lejos.

3.1 Imágenes híbridas

Para obtener imágenes híbridas es necesario tener dos imágenes que tienen cierta similitud posicional, de tamaño y rotación.
Como se ha explicado antes, debemos seleccionar frecuencias altas de una y frecuencias bajas de otra, esto se hará según la cantidad de detalles, es por eso que si tenemos gran cantidad de detalles, es mejor obtener sus frecuencias altas, así desde cerca se podrá apreciar mejor y se ignorará la imagen que está detrás. Aquella que menos detalles tenga, se obtendrán las frecuencias bajas.

Con frecuencias bajas, nos referimos a suavizar una imagen, ya que cuando se suaviza la imagen, estamos "aplanando" las altas. Una vez obtenidas las frecuencias bajas, podemos obtener las altas de manera trivial, bastaría con coger la original y quitarle las bajas (restar).
Hay que destacar un detalle importante, los sigmas de ambas imágenes no tienen que coincidir, quizás una la alisaremos mucho y la otra poco para obtener sus frecuencias altas.

Este trabajo consiste en que el usuario sea capaz de **solo** ver desde una distancia normal, la imagen con frecuencias altas y a medida que se va alejando, observar la imagen con frecuencias bajas. Por lo que nuestro trabajo es intentar acoplar ambas imágenes lo mejor posible.

El método **low_high_hybrid** recoge cuatro parámetros, el primero, la imagen a la que se quiere obtener las frecuencias bajas, ambas del mismo tamaño; el segundo argumento, la imagen para las frecuencias altas; el tercer argumento el sigma para suavizar la primera imagen y el cuarto, el sigma que se va a utilizar para suavizar la segunda imagen y posteriormente, obtener las altas.
Este método hace uso del método **low_high**, devuelve un par, las frecuencias bajas (Aplicando **GaussianBlur** de cv2) de la primera imagen y las frecuencias altas (Imagen original - **GaussianBlur**).

Finalmente mi método construye una imagen 3 veces mas grande, la primera imagen, frecuencias bajas, la segunda, altas y la tercera, la híbrida, donde es la suma de las bajas y las altas. También hay que decir que las tres están normalizadas completamente.

3.1.1 Resultados

La primera imagen que se refiere la leyenda es aquella imagen de frecuencias altas y su sigma y la segunda imagen de la leyenda, la imagen y su sigma.

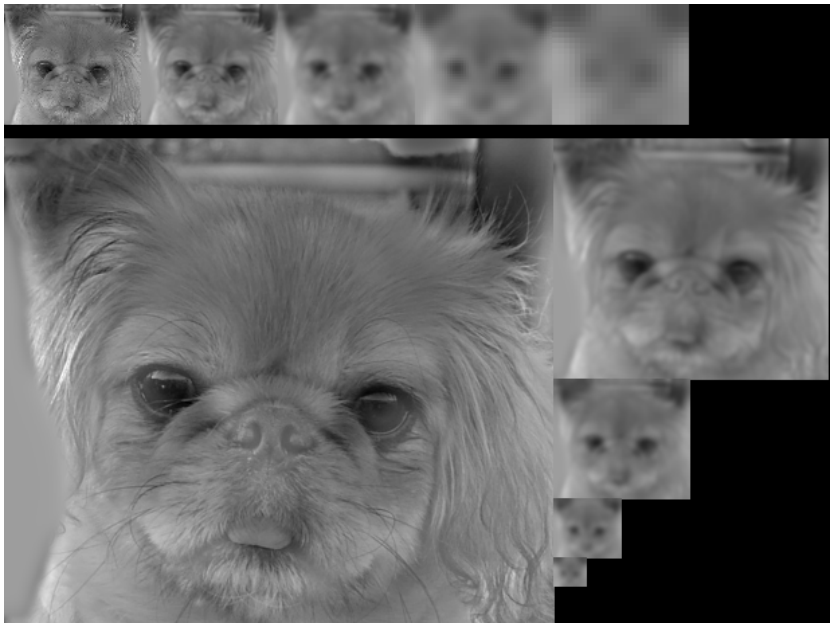

Fig. 13: **image** = "dog.bmp", $\boldsymbol{\sigma} = 7.0$; **image** = "cat.bmp", $\boldsymbol{\sigma} = 8.0$

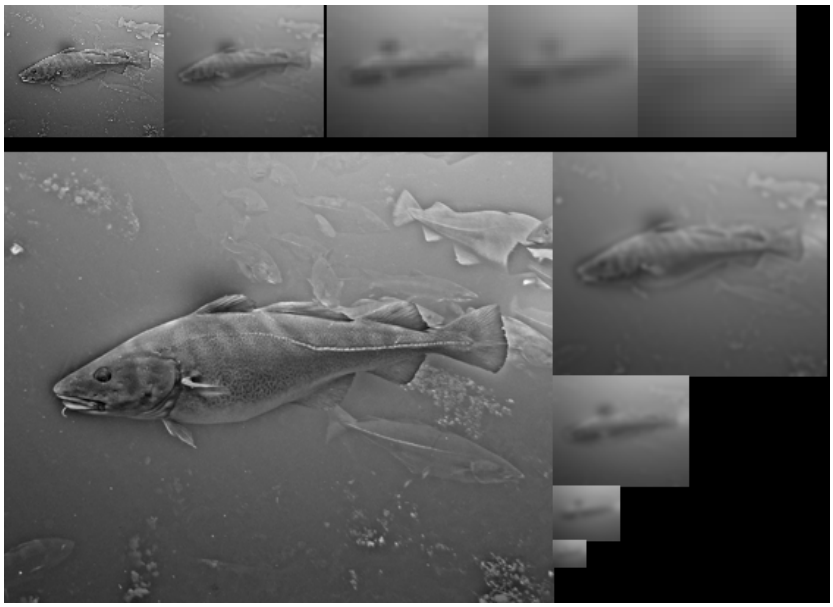

Fig. 14: **image** = "fish.bmp", $\boldsymbol{\sigma} = 4.0$; **image** = "cat.bmp", $\boldsymbol{\sigma} = 10.0$

Fig. 15: **image** = "plane.bmp", $\boldsymbol{\sigma} = 8.0$; **image** = "bird.bmp", $\boldsymbol{\sigma} = 12.0$

3.1.2 Valoración

Como vemos, se ha utilizado la pirámide gaussiana para mostrar los ejemplos, ya por cada escala, esta va eliminando frecuencias altas, por lo que el resultado es nuestra imagen de frecuencias bajas, que viene ser la imagen a la que pedimos sus frecuencias bajas (Véase las imágenes de los resultados).

Cuando dos imágenes no son del mismo tamaño, la pequeña hay que emborronarla mucho (frecuencias bajas), es decir debemos conseguir que la imagen caiga sobre la de frecuencias altas. Esto hará que de la ilusión de que la imagen con frecuencias altas tiene profundidad, algo que se había perdido al quitarle las frecuencias bajas.

La primera imagen, se ha escogido el perro por la cantidad de detalles en comparación con el gato, bordes más marcados, más pelo y se han obtenido sus frecuencias altas y el gato como frecuencias altas. Vemos que en la 3 escala ya podemos apreciar al gato. Los sigmas utilizados son, en este caso ya que ocupan más o menos la misma superficie, puestos perceptualmente.

La segunda imagen tiene un fondo con muchos detalles, los peces de alrededor y los detalles hacen que debamos situar esta imagen como la candidata a frecuencias altas y el submarino, como frecuencias altas. Como el submarino sobresale un poco del pez, hay que suavizarla mucho para que parezca parte del pez, por lo que su sigma sí es grande.

De la misma forma que el anterior, se ha escogido el avión para frecuencias altas ya que es mucho más grande que el pájaro y es imposible esconderlo detrás del pájaro. Por ello, el pájaro va a tener frecuencias bajas y además va a necesitar un gran sigma para suavizar tanto que parezca que es el color del avión.

Finalmente podemos concluir que si está bien ajustados los sigmas y las imágenes, a partir de la tercera escala, podemos observar la imagen asignada para frecuencias bajas.

4 Bonus

4.1 Bonus 1

Dada una matriz $A \in \mathbb{M}_{n\times n}(\mathbb{R})$ semejante y ortogonal a una matriz diagonal con rango 1, entonces

$$A = PDP^{-1} = PDP^t$$

Es decir, P es ortogonal ($P^t = P^{-1} = K^t$). Ahora bien, si multiplicamos para en cada lado de la matriz A por A^t:

$$AA^t = (PDK^t)(PDK^t)^t = (PDK)(KD^tP^t) = (PD(K^tK)D^tP^t) = P(DD^t)P^t$$

$$A^tA = (PDK^t)^t(PDK^t) = (KD^tP^t)(PDK) = (KD(P^tP)D^tK) = K(DD^t)K^t$$

Observamos que P son los autovectores de AA^t y que K, la matriz de autovectores de A^tA.
Además, DD^t son los valores propios de AA^t y A^tA, además como el rango es 1, solo tiene un autovalor. que coincide con la raíz cuadrada del autovalor de AA^t y A^tA.
Como $rank(A) = 1$, entonces λ_1 será el único autovalor de A, cuyo signo empíricamente es negativo (No se demostrarlo).

$$A = \begin{bmatrix} u_1 & u_4 & u_7 \\ u_2 & u_5 & u_8 \\ u_3 & u_6 & u_9 \end{bmatrix} \times \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix}$$

Observamos que P

$$A = \begin{bmatrix} u_1\lambda_1 & 0 & 0 \\ u_2\lambda_1 & 0 & 0 \\ u_3\lambda_1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix} = \begin{bmatrix} u1\lambda_1w1 & u1\lambda_1w2 & u1\lambda_1w3 \\ u2\lambda_1w1 & u2\lambda_1w2 & u2\lambda_1w3 \\ u3\lambda_1w1 & u3\lambda_1w2 & u3\lambda_1w3 \end{bmatrix}$$

$$A = \lambda_1 \begin{bmatrix} u1w1 & u1w2 & u1w3 \\ u2w1 & u2w2 & u2w3 \\ u3w1 & u3w2 & u3w3 \end{bmatrix}$$

Podemos observar que el resultado es equivalente a coger la primera columna de P, multiplicarla por columna de K transpuesta y multiplicar por el valor propio.

4.2 Bonus 2

Este bonus es una pequeña ampliación del apartado de imágenes híbridas. Como sabemos, las frecuencias altas han hecho que el color vaya desapareciendo, ya que solo se queda tonalidades de gris. Por lo que al realizar la imagen híbrida, vemos que el color que identifica a la imagen resultante es el color de las frecuencias bajas de la imagen que hemos puesto detrás.

4.2.1 Resultados

La primera imagen que se refiere la leyenda es aquella imagen de frecuencias altas y su sigma y la segunda imagen de la leyenda, la imagen y su sigma.

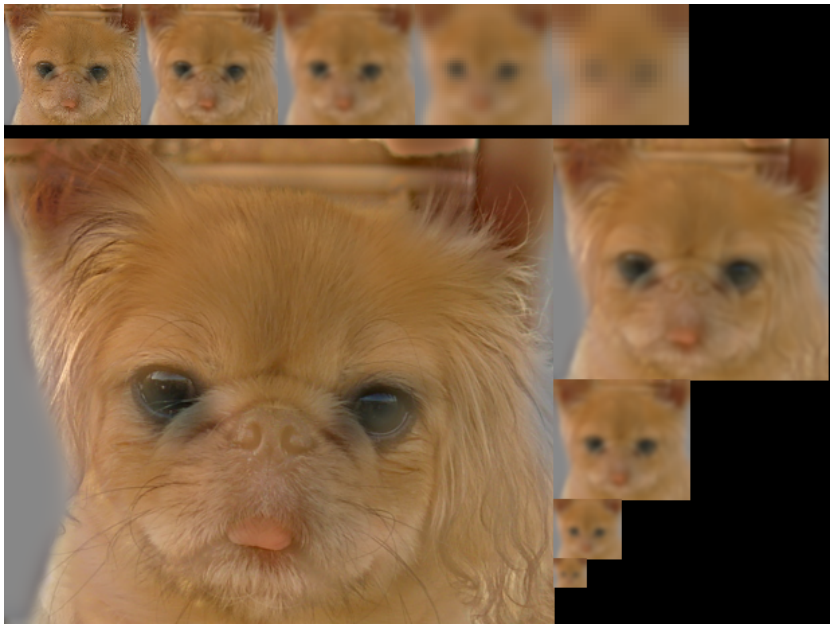

Fig. 16: **image** = "dog.bmp", $\boldsymbol{\sigma} = 7.0$; **image** = "cat.bmp", $\boldsymbol{\sigma} = 8.0$

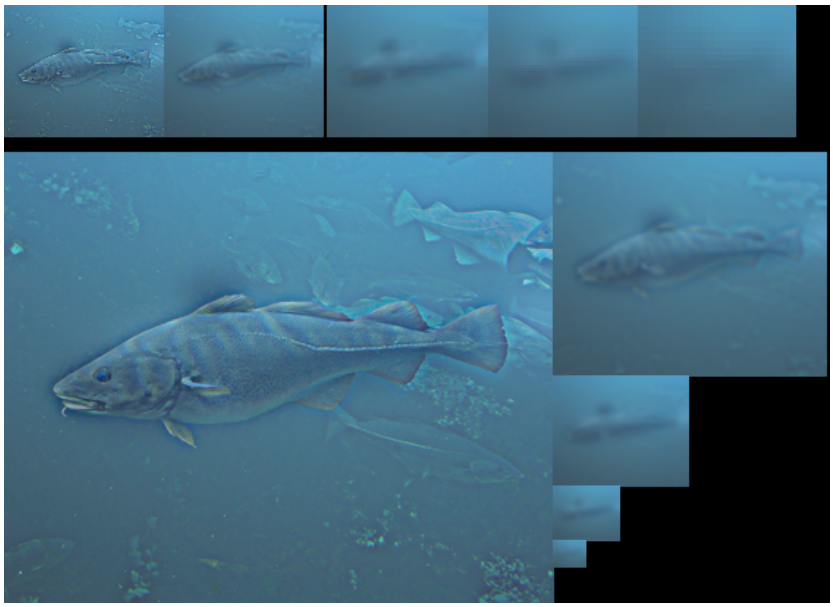

Fig. 17: **image** = "fish.bmp", $\boldsymbol{\sigma} = 4.0$; **image** = "cat.bmp", $\boldsymbol{\sigma} = 10.0$

Fig. 18: **image** = "plane.bmp", $\boldsymbol{\sigma} = 8.0$; **image** = "bird.bmp", $\boldsymbol{\sigma} = 12.0$

4.3 Bonus 3

He cogido dos imágenes que cuadran bien, con un tamaño parecido y una forma parecida. La imagen "fish-1.jpg" es la de frecuencias bajas ya que es más

Fig. 19: **image** = "fish-1.jpg", $\boldsymbol{\sigma} = 12.0$

Fig. 20: **image** = **image** = "fish-2.jpg", $\boldsymbol{\sigma} = 4.0$

pequeño que el otro pez y además lo hemos suavizado con 10 para que ocupe ese espacio. Además, el pez rojo parece tener más detalles por lo que es el que he decidido que tenga frecuencias altas.

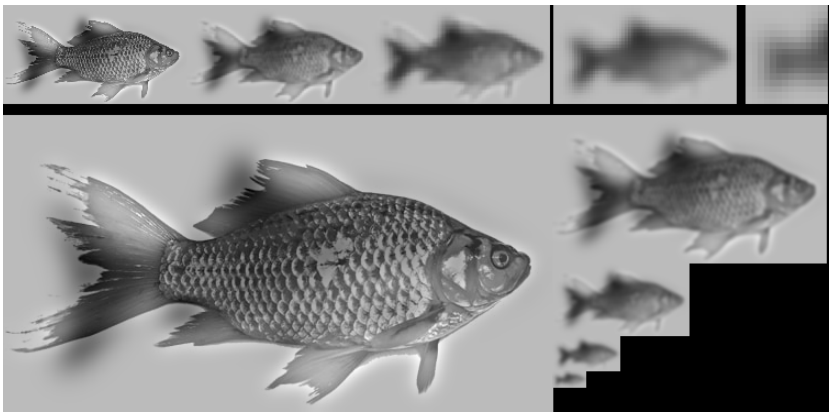

Fig. 21: Resultado

5 Pié de página

5.1 Normalización a 255

El método **normalize** recibe una matriz como argumento, esta devuelve una matriz con valores entre 0 y 255, además tiene un parámetro extra (**False** si no está definido) que identifica dos casos:

1. **Normalización completa** (Segundo argumento vale **True**).
   Esto quiere decir que, normaliza tanto valores fuera como dentro del intervalo.

2. **Normalización de fronteras** (Segundo argumento vale **False**).
   Normalizará únicamente valores exteriores al intervalo, tanto por encima como por debajo de este.

Por lo que calcula el máximo (max) y el mínimo (min) de la región (matriz) y aplica la siguiente función $f : \mathbb{R} \to \mathbb{N}$ que viene a continuación.

$$f(x) = \left\lfloor \frac{x - min}{max - min} \cdot 255 \right\rfloor$$

Este mapeo, nos devolverá una matriz cuyos valores están siempre en el intervalo $[0, 255]$ en valores enteros. Cabe destacar que también funciona para imágenes de tres canales, aplicando la misma estrategia para cada canal.

Referencias

[1] cv2 Gaussian Blur Function in Python
https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/

[2] Laplaciana Gaussiana
https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[3] Deriv Kernel
https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#getderivkernels

[4] Level of detail Textures
https://realazthat.neocities.org/demos/2016-08-17/glsl-gaussian/glsl-gaussian-suite/glsl-gaussian-suite.html

[5] Laplacian Pyramid Compression
https://www.sciencedirect.com/topics/engineering/laplacian-pyramid