Fusión de imágenes con pirámides

Hergueta Ximénez, Pablo – Higuera Viso, Tomás

– – – – – – – – – – ◆ – – – – – – – – – –

1 INTRODUCCIÓN

En esta práctica se han implementado las funciones necesarias para fusionar dos imágenes mediante pirámides. Se ha desarrollado a lo largo de cinco tareas y de manera secuencial, es decir, comenzando por las funciones más básicas y avanzando hacia las más complejas hasta finalizar con la operación principal que permite fusionar dos imágenes RGB.

2 TAREAS

2.1 Tarea 1 - Implementación de funciones básicas

- *reduce(imagen)*: para implementar esta función hemos hecho uso de *generar_kernel_suavizado(a)* para poder convolucionar nuestra imagen. Una vez convolucionada, hemos reducido indexando la matriz resultante con saltos de 2 de posiciones, tanto en el eje de ordenadas como en el de absisas, de manera que solo se mantienen la mitad de los píxeles.
- *expand(imagen)*: para implementar esta función hemos creado una matriz de ceros cuya dimensión es el doble de la imagen haciendo uso de *numpy*. A continuación, hemos guardado los píxeles de la imagen intercalándolos con los ceros de la matriz (indexando con saltos como en *reduce()*). Finalmente, se suaviza la imagen generando un kernel y convolucionndola con este.

Ambas funciones solo pueden ser utilizadas con imágenes en escala de grises, bidimensionales, ya que hacen uso de la función *convolve2d()* que trabaja solo sobre un canal.

2.2 Tarea 2 - Implementación de funciones para construir pirámides

- *gaus_piramide(imagen, niveles)*: para implementar esta función que permite crear una pirámide Gaussiana, hemos utilizado un bucle que aplicara la operación *reduce()*, desarrollada en la tarea anterior, tantas veces como niveles haya. Cada resultado, incluyendo la imagen original al comienzo del proceso, se va añadiendo a una lista (*gaus_pyr*) que será la variable de salida. De esta manera, la operación se aplicará sobre el último elemento que se hubiera añadido a esta lista en la iteración anterior, comenzando con la imagen original.
- *lapl_piramide*(*gaus_pyr)*: para implementar esta función que, a partir de una pirámide Gaussiana, calcula la pirámide Laplaciana, utilizamos la operación *expand()*, desarrollada en la Tarea 1 también. Sabiendo que la pirámide resultante ha de tener el mismo número de niveles que la Gaussiana y que, para calcular un nivel (k) de esta, hay que restar al mismo nivel (k) de la Gaussiana la expansión del siguiente nivel (k + 1), nos dimos cuenta de que el bucle debía tener un número de iteraciones equivalente al número de niveles de la pirámide Gaussiana – 1. De esta manera, no se accede a un valor no existente en el último paso. Finalmente, el último paso consiste en añadir el elemento final a la pirámide Laplaciana, que será el último que haya en la Gaussiana.

  Para solucionar el problema de que el resultado de la operación *expand()* no coincida con la dimensión del nivel k, impidiendo, por tanto, el cálculo de la diferencia de ambas matrices, hemos decidido "recortar" la expansión indexándola con el tamaño del nivel k (atributo *shape* de esta matriz), eliminando así las filas o columnas que "sobran", para que coincidan las dimensiones y poder operar ambas matrices.

2.3 Tarea 3 – Implementación de fusión y reconstrucción a partir de pirámides de Laplace

- *fusionar_lapl_pyr(lapl_pyr_imgA, lapl_pyr_imgB, gaus_pyr_mask)*: esta función fusiona dos imágenes mediante sus lapacianas y la gausiana de la mascara. Para realizar la fusión es necesario que todas las pirámides tengan los mismos niveles, así que antes de realizar la fusión comprobamos los tamaños de las mismas y si no son iguales lanzamos una excepción de tipo *TypeError*.Una vez comprobado que los parámetros son correctos pasamos a realizar la fusión que no es mas que una suma ponderada de ambas imágenes y la mascara nivel a nivel.
- *Reconstruir_lapl_pyr(lapl_pyr)*: esta función devuelve una imagen dada una laplaciana. Para obtener la imagen es necesario expandir la imagen original guardada en el ultimo nivel de la pirámide y sumarle el valor de la laplaciana en el nivel anterior. Iremos realizando esta operación de manera recursiva. Para abordar el problema de los tamaños diferentes entre el nivel de la laplaciana y la imagen expandida hemos capturado la excepción *ValueError* que se lanza cuando se trata

de realizar una suma de matrices de diferentes dimensiones. Una vez capturada la excepción hemos reducido la imagen expandida al tamaño correcto mediante indexación y el *shape* de la laplaciana en el nivel correspondiente.

2.4 Tarea 4 - Implementación de la fusión de dos imágenes en escala de grises

Esta tarea consiste en conseguir, finalmente, la fusión de dos imágenes utilizando toda la funcionalidad implementada hasta el momento. Para ello, se realizará la fusión y la reconstrucción (con las funciones desarrolladas en la Tarea 3), calculando las pirámides Laplacianas de ambas imágenes y la pirámide Gaussiana de la máscara que se vaya a utilizar previamente, además de comprobar que ambas imágenes son bidimensionales (ya que, de momento, la fusión solo servirá para imágenes en escala de grises) y de nomalizar y pasar a tipo *float* tanto las imágenes como la máscara. Para esto último se ha hecho uso de la función *img_as_float32()*, que realiza ambas operaciones.

Una vez realizada la fusión es posible que los valores resultantes se salgan de los rangos 0, 1, por lo que para formalizarlo hemos utilizado la función de *numpy clip*.

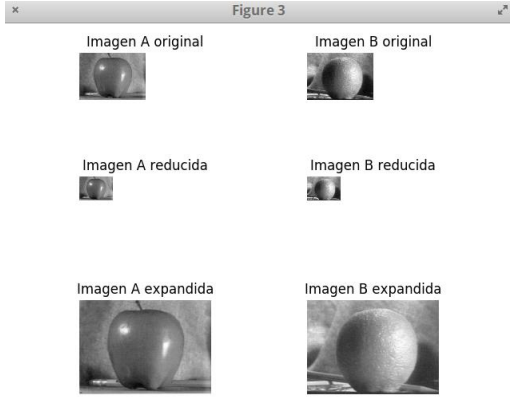
2.5 Tarea 5 - Implementación de la fusión de dos imágenes RGB

Para realizar la fusión en RGB, seguimos la misma metodología que en la fusión en un canal, la única diferencia es que esta vez realizaremos tres fusiones, una por cada canal. Para ello dividiremos la imagen por canales, así com la mascara, he iremos realizando las operaciones necesarias para fusionar (normalización y float, gaussianas, laplacianas y fusión). Una vez tengamos las tres fusiones realizadas las juntaremos haciendo uso de la función de *opencv merge*.

3 TAREAS OPCIONALES

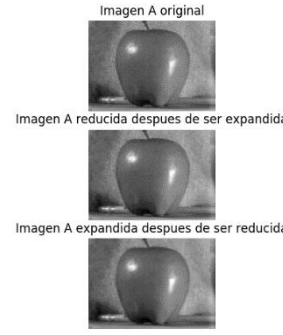
3.1 Tarea opcional 1

Para probar el correcto funcionamiento de las funciones *expand* y *reduce*, las hemos probado haciendo uso de las imágenes proporcionadas.



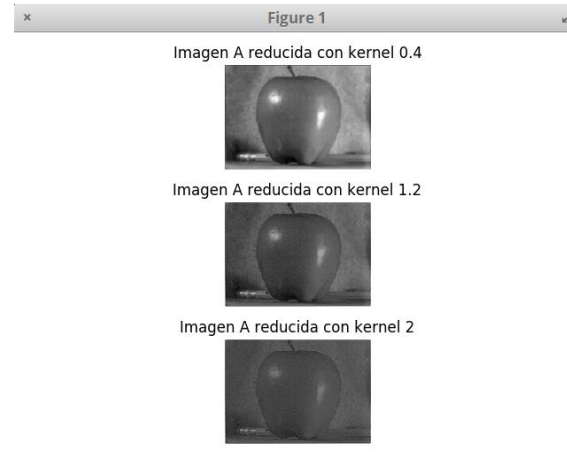
1. Imagen reducida y expandida.

Además de esto hemos probado a expandir una imagen después de haberla reducido para ver la calidad de esta, tras la perdida de pixeles, así como la acción contraria (reducir después de expandir), que por lo contrario no pierde calidad. Asimismo, hemos hallado el valor de la energía de la imagen resultante (grado de dispersión de grises entre dos imágenes)

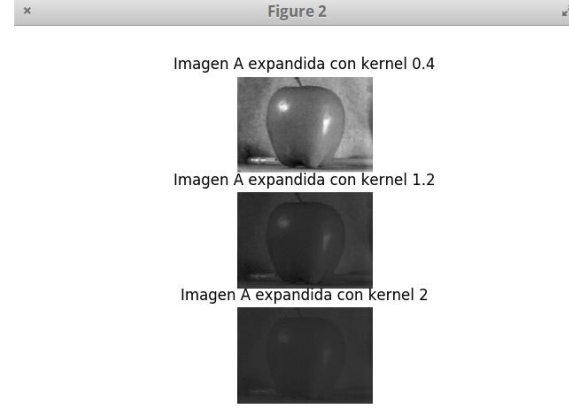


2. Pruebas de acciones consecutivas de reducción y expansión.

La ultima prueba que hemos realizado con estas funciones ha sido la de realizar expansiones y reducciones con diferentes kernel.



3. Reducciones con kernel variado.

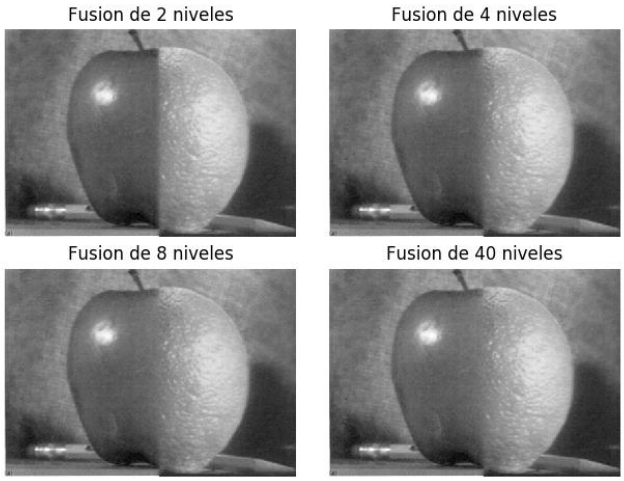


4.Expansiones con kernel variado.

Como se puede observar en las imágenes, a medida que aumentamos el parámetro a del kernel se va perdiendo calidad en la imagen. Esto es porque al realizar las funciones de expand y reduce, no solo se cambian las dimensiones de la imagen, sino que se realiza una convolución de la imagen con esl kernel, por lo que, si aumentamos el parámetro a, la operación de convolución afectara a la calidad de la imagen.

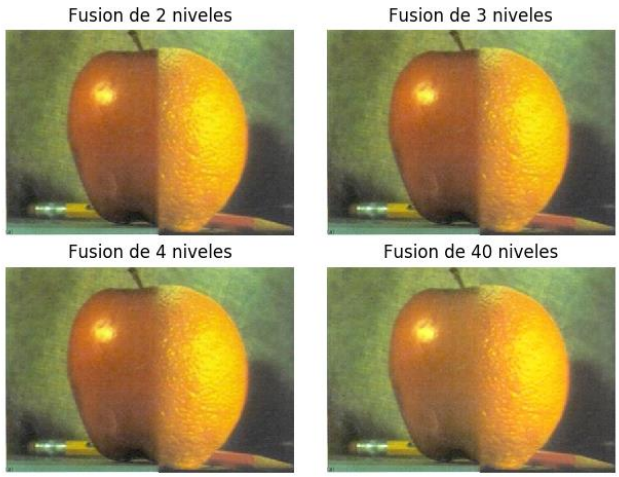
3.2 Tarea opcional 2

Una vez realizada la operación de fusión entre imágenes hemos probado a cambiar los niveles de las pirámides en la fusión para ver que efecto se producía.



5. Fusion con diferentes niveles.

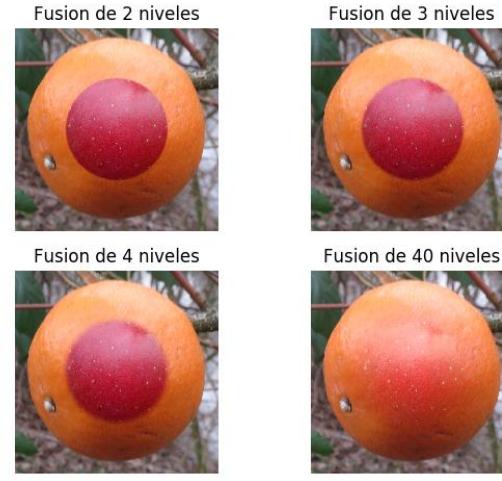
Al principio realizamos las fusiones con las imágenes en escala de grises, pero prácticamente no se podía apreciar la diferencia entre las distintas fusiones, por lo que pasamos a realizar pruebas con las imágenes en RGB.



6. Fusion con diferentes niveles en RGB.

Aun haciendo la fusión en RGB no se podía apreciar en gran medida la diferencia, tan solo que a medida que aumentamos los niveles de las pirámides en la fusión parece que la unión entre ambas imágenes se difumina un poco.

La ultima prueba que nos quedaba realizar era hacer la fusión con otras dos imágenes y esta vez si pudimos apreciar las diferencias.

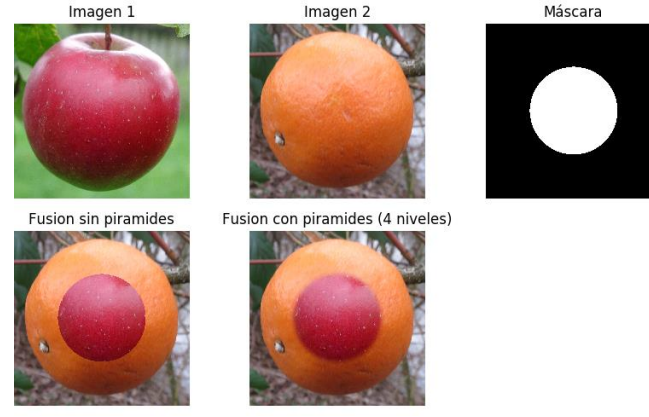


7. Fusion con diferentes niveles en RGB.

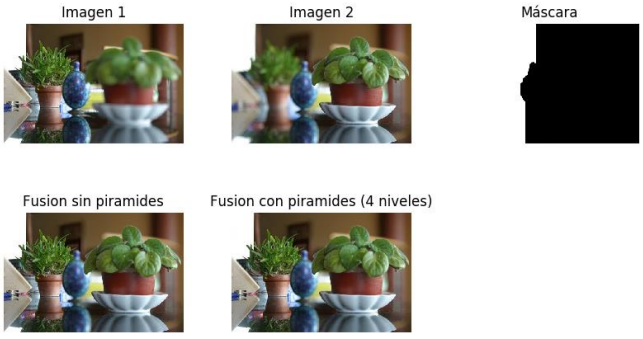
Como se puede apreciar cuando aumentamos mucho los niveles de las pirámides el color rojo de la manzana con la que hemos fusionado la naranja tiene un tono mas bajo, al igual que el tono naranja es mas alto, esto se debe porque al realizar una fusión con pirámides de tantos niveles, se realiza una gran cantidad de ponderaciones que acaban igualando los colores de ambas imágenes.

3.3 Tarea opcional 3

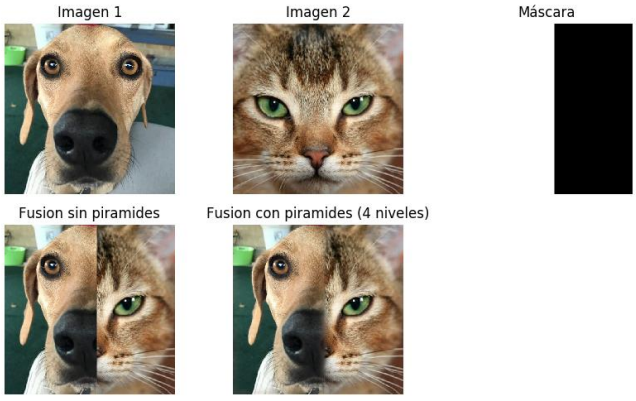
Como ejercicio final hemos realizado fusiones con diferentes imágenes y los resultados obtenidos son los siguientes.



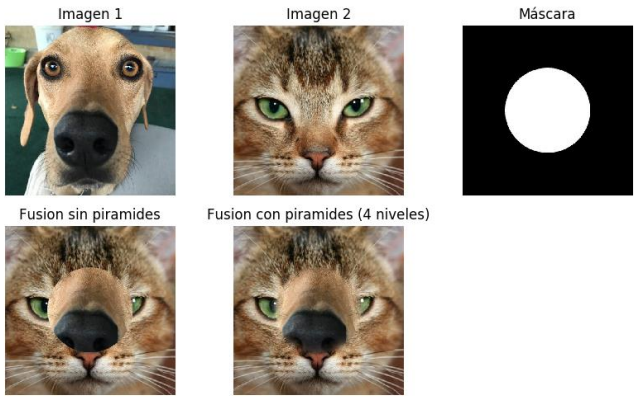
8. Fusion de manzana y naranja.



9. Fusion entre imágenes con diferente enfoque.



10. Fusion entre gato y perro 1



11. Fusion entre gato y perro 2.

12. Fusion entre gato y perro 3.

4 Conclusiones

Tras realizar esta practica hemos aprendido a realizar la fusión de dos imágenes en función de una mascara. Esta fusión no es tan solo cortar y pegar, sino que requiere la realización de pirámides laplacianas y gaussianas que nos permiten realizar una fusión de una manera mas pura en la que se aprecia en menor medida la zona en la que se unen las dos imágenes.

Además de esto hemos investigado acerca de la cantidad de pixeles que se pierden al reducir una imagen, así como lo que ocurre al aumentar y disminuir los niveles de las pirámides para realizar una fusión.

5 Carga de trabajo

Para finalizar las diferentes tareas que se nos solocitaban en la practica, tan solo hemos requerido las sesiones de laboratorio, es decir, 2 horas por cada sesión, 6 horas en total. A esto hay que sumarle el tiempo que hemos utilizado para realizar la memoria y las tareas opcionales que en total han sido 5 horas. Por lo que el tiempo total que hemos dedicado a esta practica han sido 11 horas, 6 horas presenciales y 5 horas no presenciales.

Referencias

[1] https://docs.scipy.org/doc/

[2] http://www.elai.upm.es/webantigua/spain/Asignaturas/MIP_VisionArtificial/ApuntesVA/cap4Procesadov1.pdf

[3] https://stackoverflow.com/