TRATAMIENTO DIGITAL DE LA IMAGEN

PROYECTO 1 - SEGMENTACIÓN DE IMÁGENES DERMATOSCÓPICAS

Alberto López Valero NIA: 100346384
Raúl Linio Alonso NIA: 100346329

MEMORIA

Para realizar la memoria de nuestro proceso de segmentación de melanomas, hemos seleccionado un conjunto de imágenes que contienen las características que nosotros hemos identificado como más importantes de abordar para realizar una buena segmentación (pelos, viñeteado negro, zonas claras dentro de las lesiones...)

El proceso de segmentación se divide en tres fases: pre-procesado, segmentación y post-procesado.

PRE-PROCESADO

En esta etapa, lo primero que haremos es decidir qué estrategia vamos a seguir, y de qué formas queremos jugar con la información de las imágenes.

Tras valorar si hacerlo mediante las componentes RGB, HSV, … decidimos utilizar los valores de luminancia, ya que en las imágenes que hemos podido ver, denotamos un gran contraste entre la luminancia de las lesiones y el resto de la piel, o entre la piel y los diferentes viñeteados. De esta forma creemos que podemos sacar la información necesaria para segmentar de forma precisa.

Una vez elegido esto, procedemos a convertir las imágenes al espacio de luminancia con la función ***color.rgb2gray()*** de la librería *skimage.* Tras esto, encontramos los dos primeros problemas para algunas imágenes: el vello corporal y el viñeteado.

- Eliminación del vello corporal:

  Como la luminancia de las lesiones que buscamos y la del vello puede ser similar en algunos casos, a la hora de posteriormente realizar una segmentación por umbral, seguramente cometamos errores. Por ello daremos el paso de eliminarlo, y lo haremos primero detectando este vello con la función ***morphology.black_tophat()*** de la librería *skimage*, con un elemento estructurante *np.ones(9,9),* y después sumándoselo a la imagen.

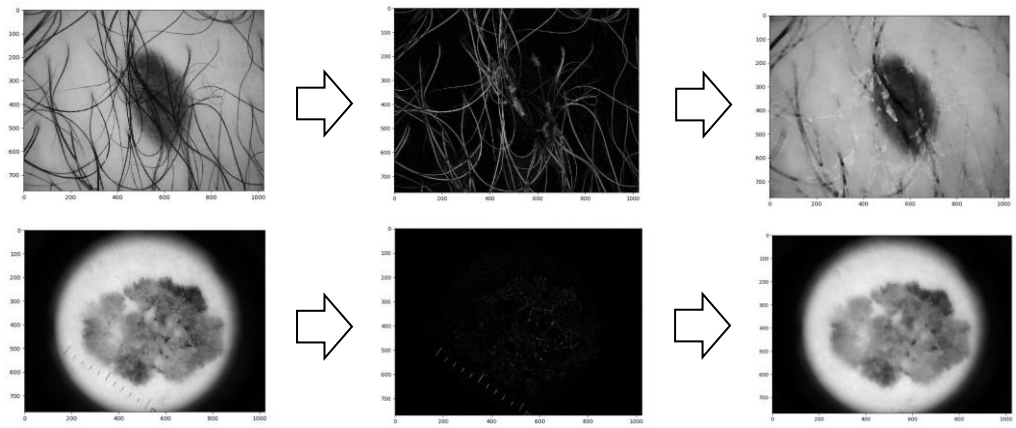

Figura XX: Ejemplo de eliminación de vello corporal de las imágenes

  Con esta función también eliminamos ciertos residuos indeseados que encontramos en algunas imágenes (líneas de medida, poros...), y conseguimos una imagen limpia.

  Tras este proceso, aplicamos a la imagen un filtro de mediana a través de la función ***dimage.median_filter()*** de la librería *scipy,* para suavizar los bordes y aclarar ciertas zonas.

- Eliminación de viñeteado:

  En algunas imágenes encontramos un viñeteado de baja luminancia que rodea las imágenes. Esto puede ser un problema ya que en algunos casos, llega a tocar la lesión o difumina sus bordes, por lo que al segmentar puede llegar a dar confusiones. Para ello hemos decidido crear otra imagen que se adapte a las características de la viñeta que podamos tener.

Para ello, imponiendo condiciones, tratamos de detectar este viñeteado buscando valores de luminancia 0 en todas las esquinas de la imagen o cerca de ellos. Si los detectamos, creamos un array del mismo tamaño de la imagen original, con un valor de luminancia máximo (255), y en su interior situamos un círculo de valor 0, con la función ***draw.circle()***, de la librería skimage, el cual tendrá un radio variable dependiendo del grosor del viñeteado que detectemos.

Al sumar esta máscara a la imagen original, conseguimos dejar exactamente igual la zona del círculo central (zona de la lesión), y dejar en máxima luminancia la zona antes viñeteada. Puesto que algunos píxeles de esta zona pueden sobreexponerse, recorremos los pixeles sobresaturados y le damos el valor máximo de luminancia.

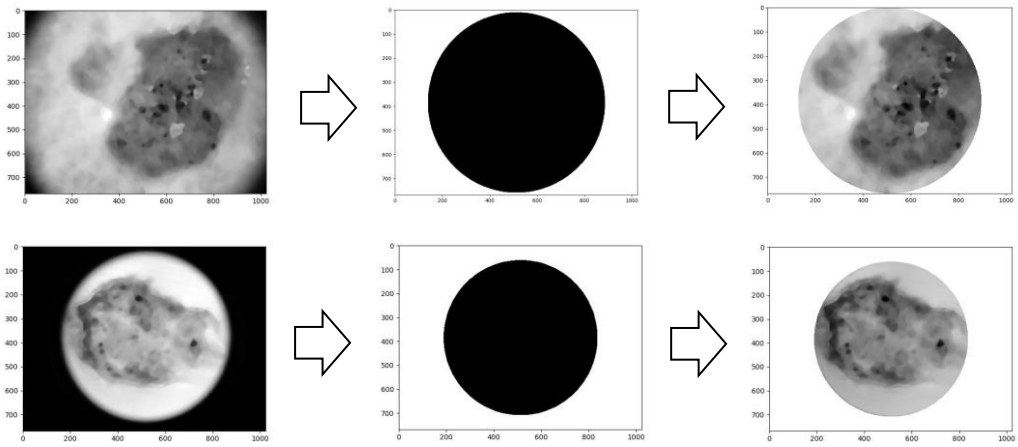

Figura 2: Ejemplo de eliminación de viñeteado de las imágenes

Por supuesto, cometemos algún error, ya que en algunas de las imágenes en las que detectamos viñeteado, la lesión no se encuentra exactamente en el centro y podemos llegar a cortar alguna parte de ella. Pero por lo general, acertamos.

Al final de este proceso, a la imagen obtenida le restamos la misma máscara pero con la zona exterior del circulo en gris oscuro (50). Lo que conseguimos con esto es que no obtengamos una gran densidad de pixeles blancos, lo que posteriormente puede causarnos errores al utilizar segmentación por umbral.

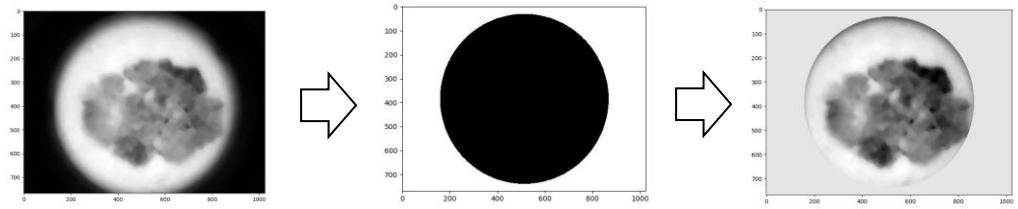

Figura 3: Ejemplo de eliminación de viñeteado de las imágenes + viñeta gris

En esta última parte hemos tenido problemas, ya que de manera inexplicable, para ciertas imágenes no conseguimos realizar esta última resta, pero con el resto conseguimos un buen resultado,

La última fase de nuestro pre-procesado, es un reescalado de intensidad. Para ello usamos la función ***exposure.rescale_intesity()***, de la biblioteca *skimage,* que devuelve la imagen después de estirar o reducir sus niveles de intensidad. Para calcular el rango de valores de intensidad al que queremos reescalar, utilizamos la función ***np.percentil()*** de la biblioteca *numpy*. Con esta función, se ordenan de menor a mayor el valor de los pixeles, y podemos hallar el valor bajo el que se encuentra el porcentaje que decidamos. En nuestro caso queremos el valor bajo el 19% y el 89%, y con estos valores, reescalar la intensidad de las imágenes.

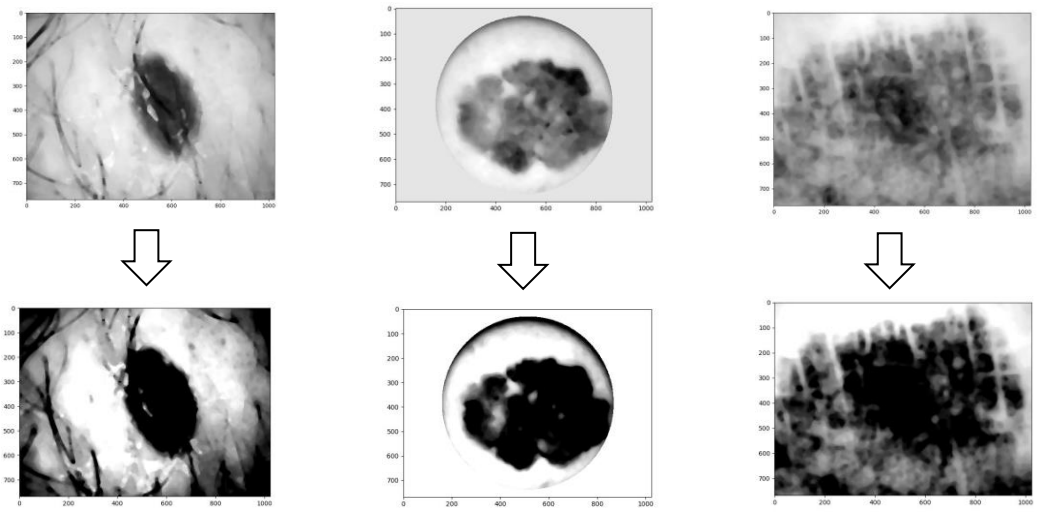

Figura 4: Ejemplo de reescalado de intensidad de las imágenes

SEGMENTACIÓN

En cuanto a la segmentación, tras varias pruebas con *K-means*, nos dimos cuenta de que este método resultaba ser computacionalmente muy pesado, y como realmente el pre-procesado lo estamos orientando hacia una segmentación por umbral, decidimos implementar el *Método Otsu*.

Este método se basa en el cálculo de un umbral óptimo que nos haga diferenciar la lesión del resto de la imagen. Para calcular este umbral, la función lleva a cabo varios pasos. Primero calcula el histograma normalizado y la varianza global de la imagen. El umbral va a dividir el histograma en dos y calcula la media de cada parte. A continuación calcula la varianza interclase, y con esta y la varianza global calculada previamente, lleva a cabo el último paso que es la elección del umbral óptimo.

Para calcular el *Umbral de Otsu* usamos la función ***filters.threshold_otsu()*** de la biblioteca *skimage.* Una vez tenemos ese valor de umbral, damos valor mínimo (0) a aquellos valores que superen el umbral, y máximo (255) a los que sean inferiores.

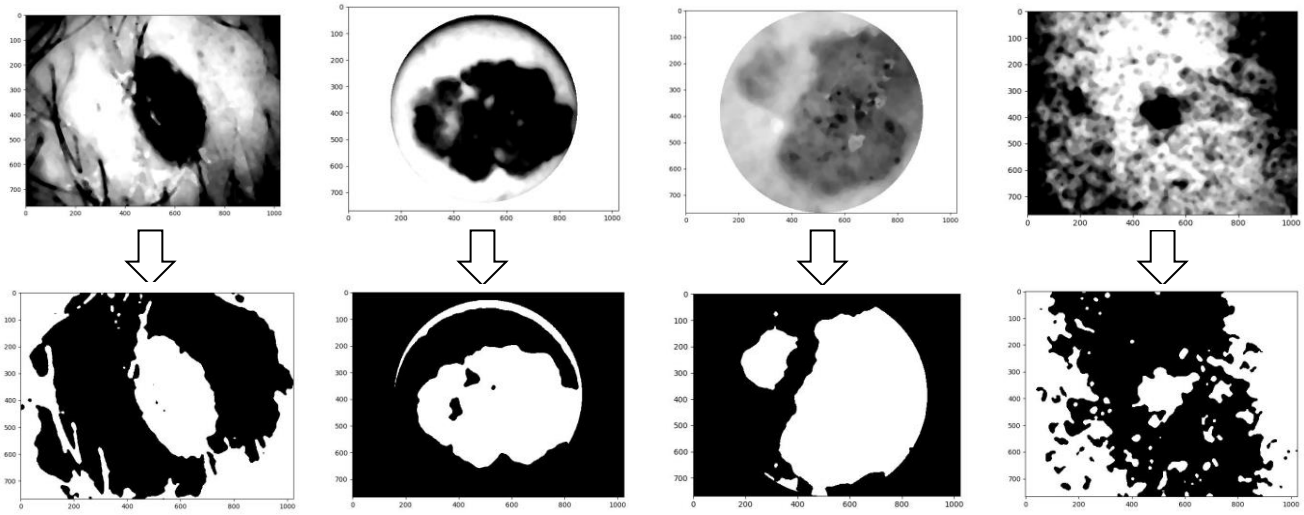

Figura 5: Ejemplo de segmentación por Método de Otsu

Con este método detectamos bastante bien cuál es el área de la lesión la mayoría de las ocasiones, aunque como se puede ver en los ejemplos anteriores, no siempre cubrimos el área completa. En algunas imágenes se nos escapan ciertas zonas de la lesión más claras o difuminadas, dando lugar a errores en dicha segmentación.

POST-PROCESADO

Para empezar con el post-procesado lo primero que hacemos es aplicar dos operaciones de morfología matemática.

En primer lugar, aplicamos una apertura con el objetivo de eliminar los istmos y protuberancias delgados. Para ello hacemos uso de la función ***morphology.opening()***, de la librería *skimage*.

En segundo lugar, aplicamos un cierre para eliminar todos los agujeros y grietas que nos aparecían. Para ello usamos la función ***morphology.closing()***, obteniendo el resultado de los siguientes ejemplos:

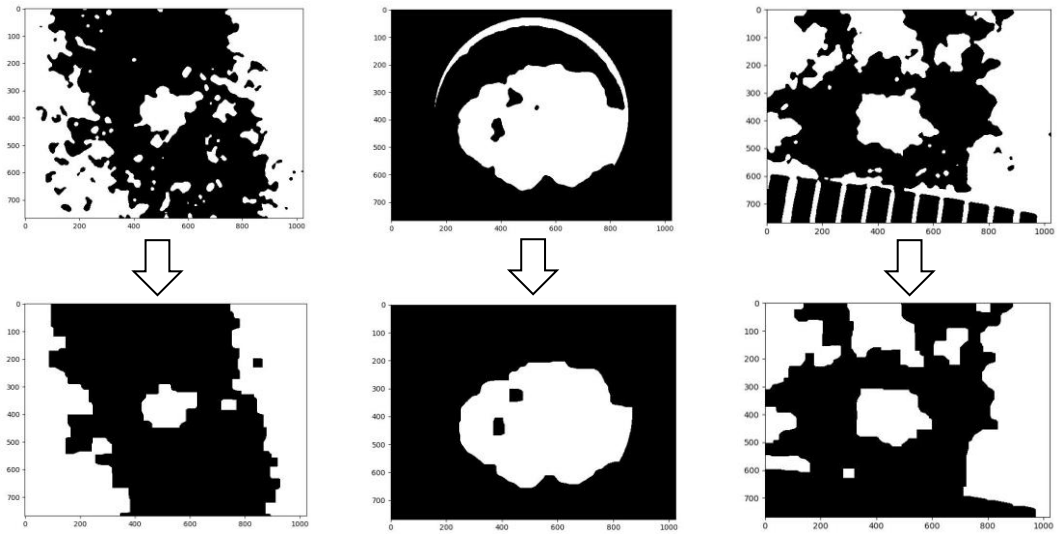

Figura 6: Ejemplo de apertura y cierre morfológico de las imágenes

A continuación, como sabemos que la mayor parte de las veces, la lesión se encontrará en el centro de la imagen, la buscaremos y solo nos quedaremos con ella.

Para eso, utilizaremos la función ***morphology.label()***, con la cual vamos a etiquetar regiones conectadas dentro de la imagen, es decir, la función va comparando pixeles y estableciendo cuáles de sus píxeles vecinos están conectados comprobando si tienen su mismo valor.

Uno de los parámetros de la función es “connectivity” que define el número de saltos ortogonales para considerar un pixel como vecino, y tras las pruebas pertinentes, nosotros hemos optado por connectivity = 2, que es la siguiente matriz:

```
2-connectivity

[ ]  [ ]  [ ]
   \  |  /
[ ]--[x]--[ ]
   /  |  \
[ ]  [ ]  [ ]
```

La segunda función que utilizamos para la extracción de las propiedades es ***measure.regionprops()*** que hace precisamente eso, medir las propiedades de las regiones etiquetadas anteriormente. De estas propiedades, las que más nos interesan son el área de cada región, y las coordenadas bidimensionales de sus centroides.

Con esta información, podemos recorrer la lista de medidas tratando de encontrar aquellas regiones de la imagen que cumplan un parámetro de área mínima, y que las coordenadas de sus centroides se encuentren en la zona central de la imagen. Si esto no se cumple y no detectamos ninguna región, ya que hay algunas lesiones que prácticamente ocupan toda la imagen, volvemos a recorrer la lista imponiendo condiciones menos restrictivas para así poder almacenar estas regiones más grandes.

Como podemos ver en los siguientes ejemplos, conseguimos detectar la región que queremos prácticamente siempre. El problema le podemos encontrar con aquellas imágenes en las que la segmentación por el Método de Otsu ha dividido nuestra lesión en porciones, las cuales serán detectadas por la función ***morphology.label()*** como regiones distintas.

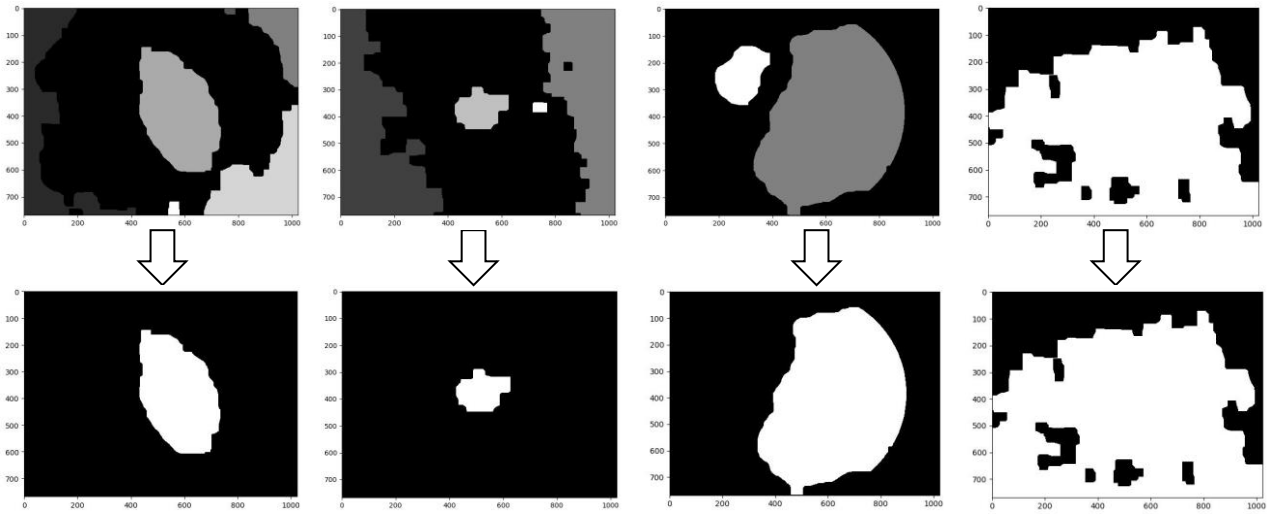

Figura 7: Ejemplo de selección de región correcta

Por último, utilizamos dos funciones de morfología matemática más: ***morphology.dilation(),*** para dilatar el área de nuestra máscara y ajustarla mejor al área de la lesión, y ***morphology.convex_hull_image()***. Esta última función transforma nuestros pixeles blancos a un polígono convexo de área mínima capaz de rodear todos ellos. Esto nos ayuda a la hora de homogeneizar la segmentación de áreas grandes, ya que normalmente se nos quedan huecos grandes o pequeñas zonas sin detectar. Igualmente, esta última función nos penaliza en otras segmentaciones, ya que podemos llegar a perder detalles en el borde de estas máscaras.

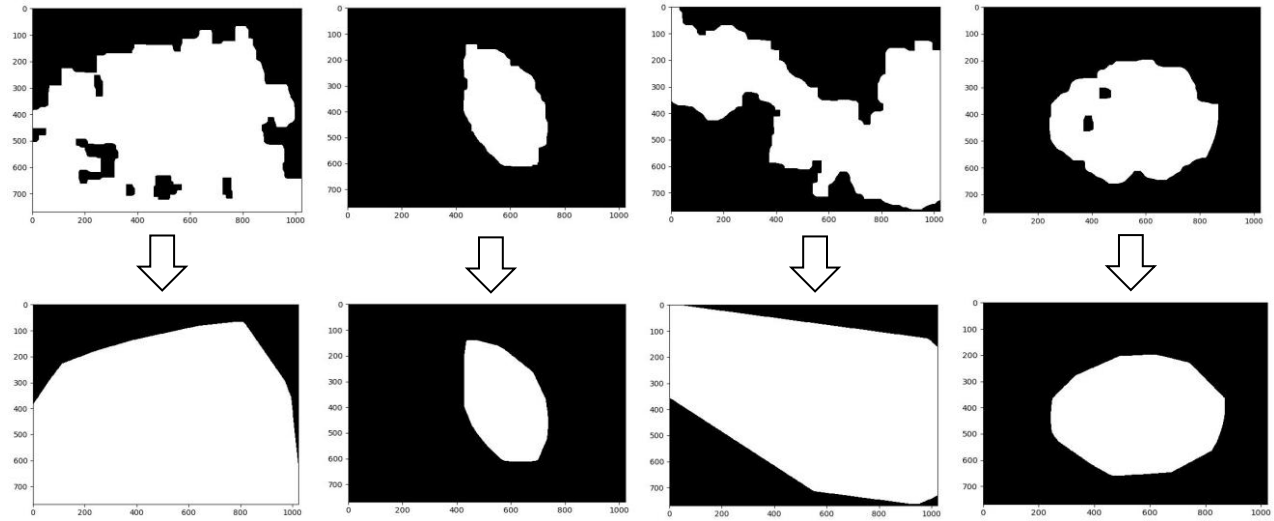

Figura 7: Ejemplo segmentación final

Aquí finaliza nuestro proceso de segmentación, en el que obtenemos máscaras que en la mayoría de los casos se ajustan bien a los bordes de la lesión cutánea.

RESULTADOS

Viendo las máscaras obtenidas y comparándolas con las imágenes a segmentar, somos conscientes de los errores que hemos cometido. Por ejemplo, seguramente hay alguna manera más óptima de eliminar el efecto viñeta, o de realizar la segmentación en una zona ignorando el resto.

Quizá con algo más de tiempo podríamos haber llegado a otras soluciones incluso más simples para abordar estos problemas, por ejemplo, haber podido seleccionar a qué imágenes aplicarles la función *morphology.convex_hull_image(),* e incluso haber diseñado nosotros una función que pudiese aplicar a la imagen un polígono cóncavo, en vez de convexo como la función anterior.

Aun así, con el proceso de segmentación que hemos realizado, en *Spyder* conseguimos un **Jaccard Score** sobre el conjunto de imágenes de **0.680033973** (nota obtenida antes de tener las máscaras solución de aula global), y en el leaderboard de **Kaggle** una marca de **0.52500**, quedando 6º en la competición.

Nuestra valoración de los resultados es buena. Hemos aprendido mucho con esta práctica, y hemos invertido bastante tiempo en su realización por nuestro interés en este tema desde que ambos lo descubrimos hace unos años. Aunque finalmente hayamos utilizado las funciones anteriormente mencionadas en la memoria, podríamos decir que hemos estudiado y probado casi todas la funciones estudiadas en la asignatura hasta el momento.

BIBLIOGRAFÍA

Gouillart, Emmanuelle. Scipy-Lectures:
https://scipy-lectures.org/packages/scikit-image/index.html

Gouillart, Emmanuelle y Varoquaux, Gaël. Scipy-Lectures:
https://scipy-lectures.org/advanced/image_processing/

Scikit-image:
https://scikit-image.org/docs/dev/user_guide/transforming_image_data.html

Gouillart, Emmanuelle y Varoquaux, Gaël. Github:
https://claudiovz.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html

Scikit-image:

- https://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.convex_hull_image
- https://scikit-image.org/docs/dev/api/skimage.morphology.html
- https://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.dilation
- https://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.label
- https://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.opening
- https://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.watershed

Diapositivas de teoría Aula Global.

Prácticas realizadas en la asignatura.