Práctica 3. Segmentación de imágenes

Gonzalo Mesas Aranda

17 de noviembre de 2023

1. Introducción

La siguiente práctica ha sido realizada en Python por decisión personal. Se detallará tanto el desarrollo de cada ejercicio como su solución. La **imagen** a estudiar es la número **8**.

2. Pasos previos

Antes de proceder a la resolución de la práctica, es necesario llevar a cabo unos sencillos pasos previos, de igual manera que en prácticas anteriores. Entre ellos se encuentran la importación de librerías y la apertura de la imagen correspondiente.

```
import numpy as np
import pydicom as pdc
import matplotlib.pyplot as plt
import cv2

# Abrir la imagen
def abrir_imagen(num_imagen):
    return pdc.dcmread(fr"C:\Users\Gonzalo_MA\Desktop\Universidad\Tercero
    \Imagenes biomedicas\Practica 1/im{num_imagen}.dcm")

imagen = abrir_imagen(8)

# Guardar la imagen en una matriz de datos indicando los bits por pixel
def imagen_bits(imagen, bpp = 12):
    if bpp == 12:
        sol = np.array(imagen.pixel_array)
    elif bpp == 8:
        sol = (np.array(imagen.pixel_array) * (255/4095)).astype(np.uint8)
    return sol, bpp

# Creaamos la matriz de la imagen con 12 bpp
imagen_pixeles, bpp = imagen_bits(imagen, 12)

# Visualizamos la imagen
plt.imshow(imagen_pixeles, cmap = 'gray')
plt.axis('off')
plt.title("Imagen original")
plt.show()
```

Listing 1: Librerías y pasos previos.

3. Ejercicio 1

3.1. Enunciado

Implementar un algoritmo de umbralización para obtener la estructura anatómica de interés. Para ello, se propone la realización de las siguientes tareas:

1. Realizar una **umbralización manual** con los umbrales de visualización utilizados en la práctica 1 para la visualización óptima de la región de interés.

2. Realizar una **umbralización automática e iterativa**, utilizando algún método que permita el cálculo automático de los umbrales a partir de la información obtenida con el histograma. (Por ejemplo, puede usar la idea de la transparencia 16 de la clase 3.1 Segmentación, el algoritmo de la transparencia 20 u otro que se adapte mejor a su problema.

3. Realizar la **umbralización automática e iterativa**, con un algoritmo similar al del apartado anterior, pero **después de filtrar previamente la imagen original** con un filtro gaussiano de 5x5, con sigma=1.

3.2. Umbralización manual

De acuerdo con la práctica primera, la umbralización manual para la visualización óptima de la región de interés se corresponde con la ventana lineal para el tejido encefálico. Ya que ha sido explicada en las prácticas anteriores, únicamente se mostrará el resultado y se valorará brevemente.

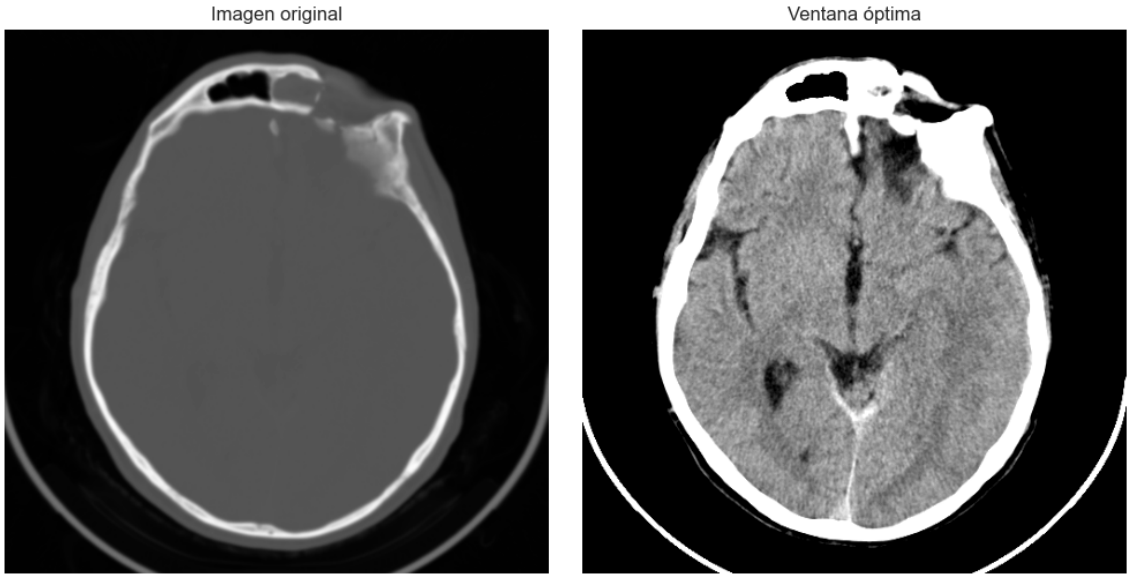

Figura 1: Imagen original y ventana de tejido cerebral.

3.3. Conclusión de la umbralización manual

Con el uso de la ventana conseguimos diferenciar estructuras que no son distinguibles en la imagen original, correspondientes al encéfalo a la altura de los senos paranasales frontales. Por otro lado se pierde información osea en el cráneo.

3.4. Umbralización automática

Para la realización de este apartado, se ha programado el algoritmo propuesto en las diapositivas de clase descrito a continuación:

1. Seleccionar un **umbral inicial** T (media de los valores de los píxeles).
2. Aplicar la umbralización con T para obtener dos grupos de píxeles G_1 y G_2.
3. Calcular el valor medio (m_1 y m_2) de los grupos G_1 y G_2.
4. Calcular el nuevo umbral promediando m_1 y m_2.
5. Volver a al paso 2 y repetir hasta que la diferencia entre dos T sucesivos sea menor que un valor dT.

La propuesta para el siguiente algoritmo (Ver Listing 3) toma como argumentos la imagen y una matriz booleana del mismo tamaño que la imagen.

Esta matriz hace que los valores del histograma sobre los que se aplica el algoritmo sean únicamente los que se correspondan con un valor $True$, es decir, actúa como una máscara que permite decidir que valores se tiene en cuenta para la umbralización. Su implementación será útil en el siguiente apartado, por lo que por ahora se tomará este argumento como una matriz de valores $True$.

Además se hace uso de una función auxiliar para calcular la media de los grupos. El diseño es el siguiente:

```
def media(values, ammount):
    suma = 0
    for x in range(len(values)):
        suma += values[x] * ammount[x]
    return suma / sum(ammount)
```

Listing 2: Código de la función auxiliar para caluclar media.

```python
def umb_aut(imagen, condicion):
    # Seleccionamos solo los pixeles que están en la máscara
    imagen_plana = imagen.flatten()
    condicion = condicion.flatten()
    imagen_modificada = imagen_plana[condicion]

    # Inicializamos la media
    val, cuen = np.unique(imagen_modificada, return_counts= True)
    val = val.tolist()
    cuen = cuen.tolist()

    old_mean = -5
    mean = media(val, cuen)
    iteraciones = 0

    # Iteramos hasta que la diferencia entre dos medias sucesivas
    # sea menor que 1e-5 o supere un max de iteraciones
    while abs(old_mean - mean) > 1e-5 and iteraciones < 50:
        val1 = []
        cuen1 = []
        val2 = []
        cuen2 = []
        for i in range(len(val)):
            if val[i] <= mean:
                val1 += [val[i]]
                cuen1 += [cuen[i]]
            else:
                val2 += [val[i]]
                cuen2 += [cuen[i]]

        # Calculamos las medias de los dos grupos
        mean1 = media(val1, cuen1)
        mean2 = media(val2, cuen2)

        # Actualizamos los valores
        old_mean = mean
        mean = (mean1 + mean2) / 2
        iteraciones += 1

    return (imagen > mean), mean
```

Listing 3: Código de la umbralización automática iterativa.

Después de aplicar el algoritmo tanto a la imagen original como a la ventana, se obtiene el siguiente resultado.

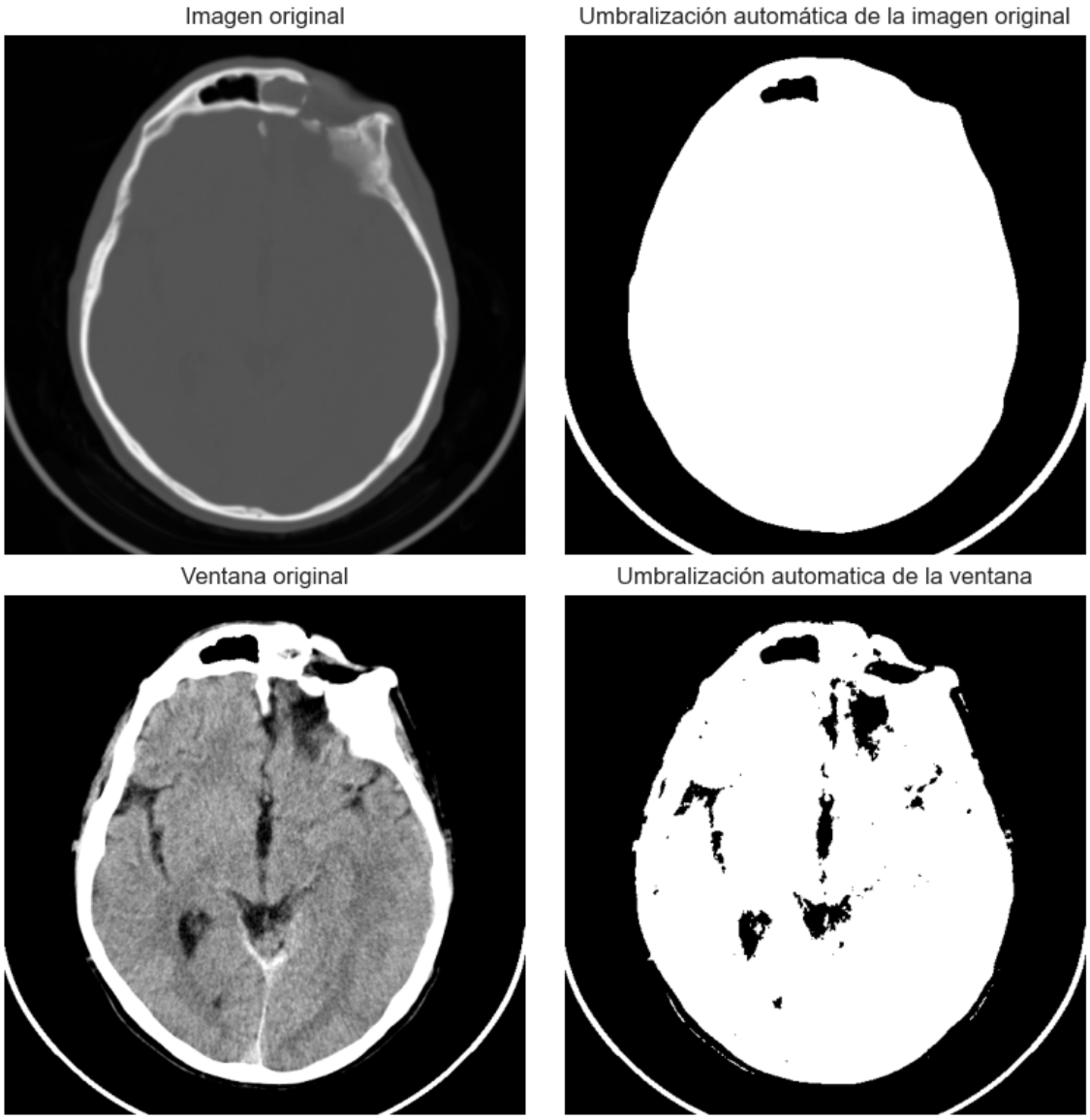

Figura 2: Imagen original y ventana umbralizadas (1 aplicación).

Otra información relevante sobre la figura anterior son los histogramas con los valores finales de la media.

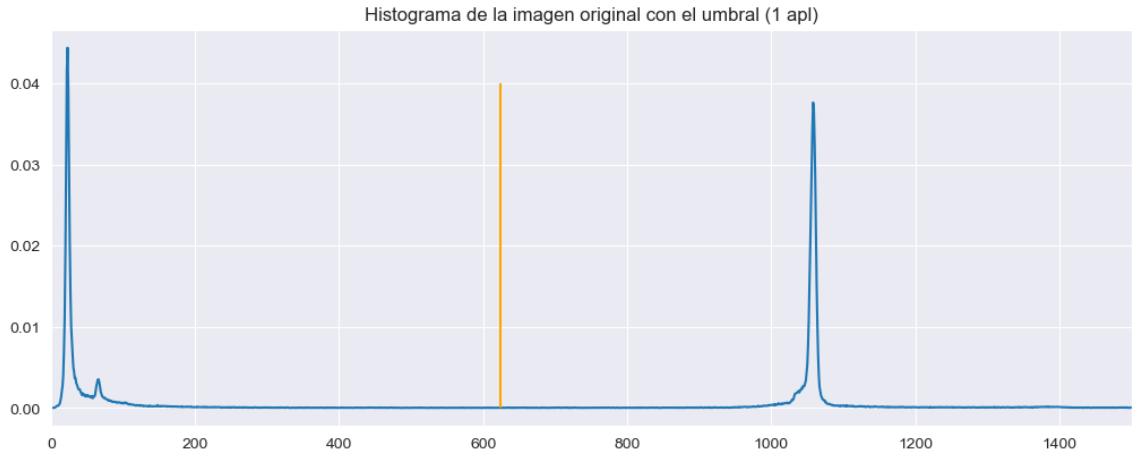

Figura 3: Histograma de la imagen original (1 aplicación).

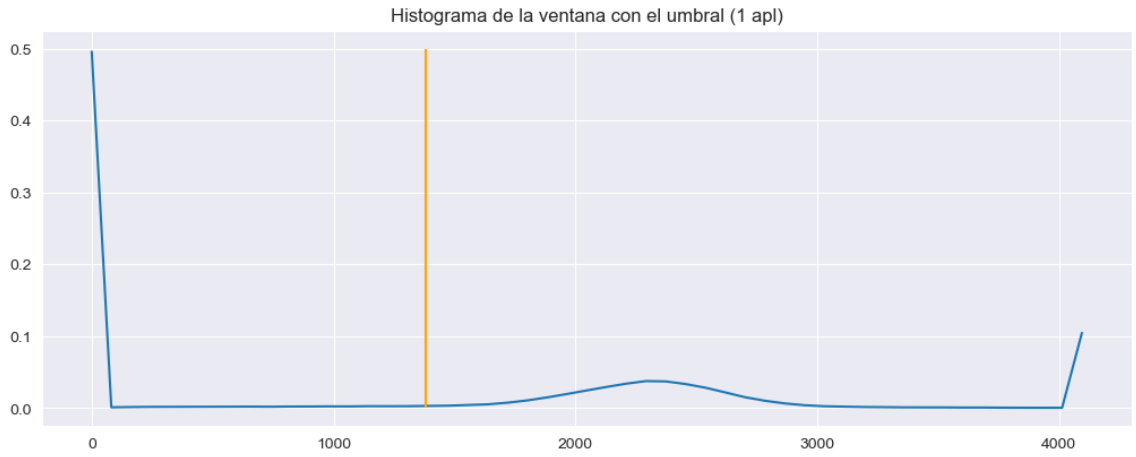

Figura 4: Histograma de la ventana (1 aplicación).

Umbral final de las imágenes	
Imagen original	0624.64
Ventana	1379.59

Cuadro 1: Umbral resaltado en los histogramas.

El resultado de la imagen original no resalta ninguna estructura de interés, por lo que se propone aplicar el algoritmo un número N de veces para cumplimentar el objetivo.

Para ello se define una función que repite la función vista anteriormente (Ver Listing 3) iterativamente sobre una imagen dada. Es en esta función donde se hará uso del argumento *condicion*. La idea es aplicar el algoritmo un número dado de veces a la imagen original, pero en cada iteración, solo se tienen en cuenta los píxeles que encajan en la máscara resultante de la binarización en la iteración previa.

```
def n_umb_aut(imagen, n):
    # Inicializamos la primera iteración con una máscara de valores True
    imagen_binaria, media_im = umb_aut(imagen, np.ones(
    (imagen.shape[0], imagen.shape[1]), dtype=bool))

    # Iteramos
    for i in range(n-1):
        imagen_binaria, media_im = umb_aut(imagen, imagen_binaria)
    return imagen_binaria, media_im
```

Listing 4: Código de la N-umbralización.

Después de probar con diferentes números de iteraciones, se ha determinado que 2 aplicaciones en la imagen original es el número ideal. Por otro lado, la ventana ya da un buen resultado tras una aplicación, aunque a efectos de contraste se han realizado también dos aplicaciones. Por tanto, usando esta nueva función y después de iterar 2 veces para cada imagen, el resultado es el siguiente.

Umbral final de las imágenes (2 aplic.)	
Imagen original	1757.62
Ventana	3162.77

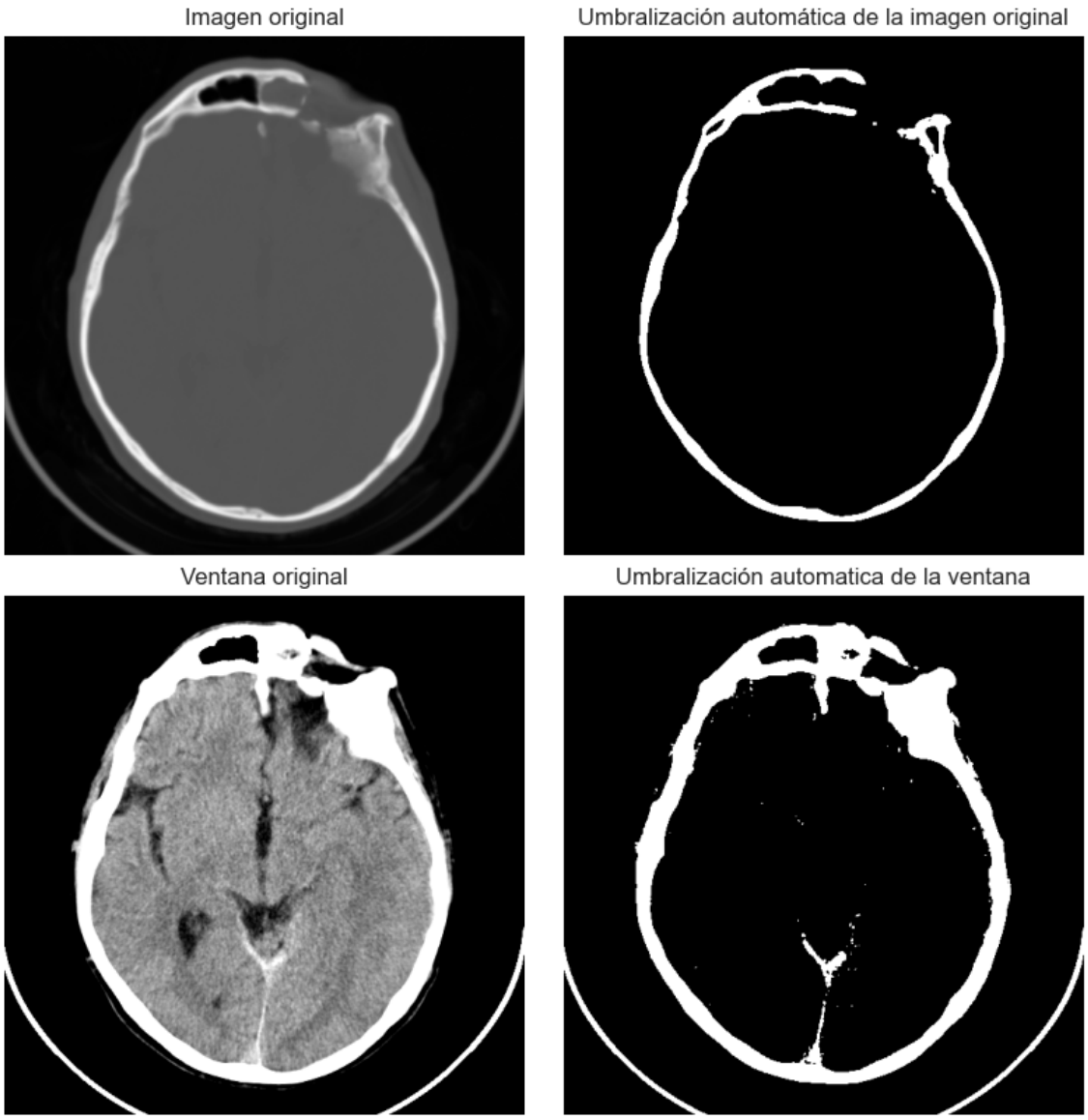

Figura 5: Dos aplicaciones de la umbralización.

3.5. Conclusión de la umbralización automática

El resultado después de una única aplicación del algoritmo sobre ambas imágenes, da el resultado esperado sobre la ventana, donde se han diferenciado las zonas más oscuras del área intraencefálica. Sin embargo, la umbralización de la imagen original solo ha resultado en la silueta de la cabeza, por lo que a priori no es un resultado muy útil al no resaltar ninguna región de interés.

No obstante, después de aplicar el algoritmo dos veces para la imagen original, se obtiene una buena segmentación del cráneo, a diferencia de la ventana, la cual da como resultado una basta segmentación al no ser esta la zona resaltada al aplicar la misma. Además, se pierde la información útil obtenida en la primera aplicación.

Resumiendo, el mejor resultado de la imagen original se obtiene después de una iteración y el de la ventana después de dos iteraciones.

3.6. Umbralización automática (prefiltrado)

En este apartado se realizarán los mismos pasos que en el anterior para obtener el mejor el resultado, con la diferencia que las imágenes sobre las que se aplica el algoritmo han sido filtradas con un filtro gaussiano 5x5 y desviación estándar 1 .

Se ha utilizado la función diseñada en la práctica anterior para aplicar el filtro gaussiano. Una vez aplicado y umbralizado cada imagen con su número correspondiente de aplicaciones, se obtiene: (Ver Figura 6)

3.7. Conclusión de la umbralización automática (prefiltrado)

A la vista de la Figura 7, se puede concluir que para la imagen original no hay diferencia aparente, por lo que se podría omitir este prefiltrado para aumentar la eficiencia computacional. Sin embargo, en la umbralización de la ventana, se ha reducido un pequeño ruido residual visible en la binarización en distintas partes de la imagen, así que ha sido de utilidad el filtrado en este caso.

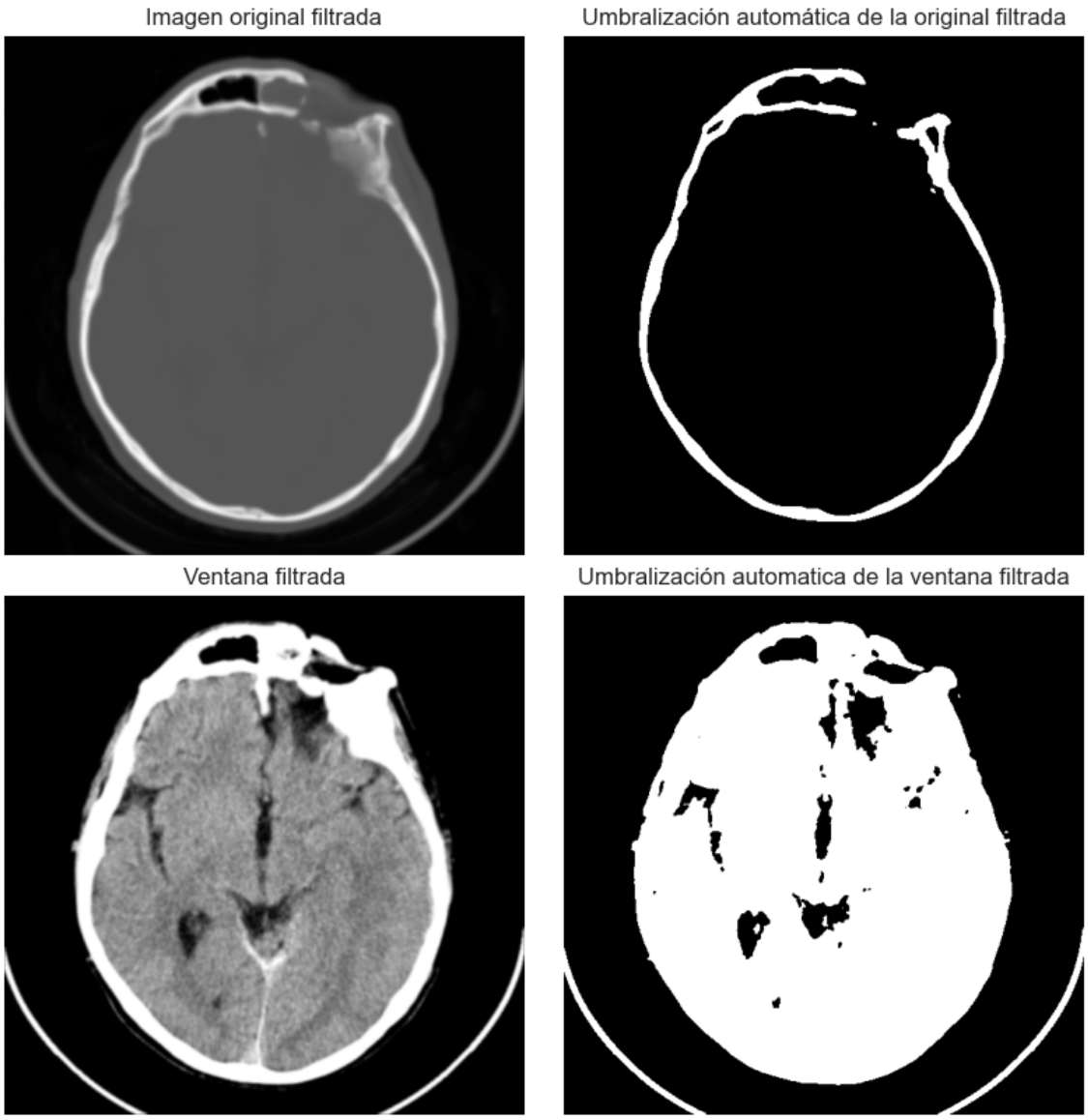

Figura 6: Umbralización de las imágenes filtradas.

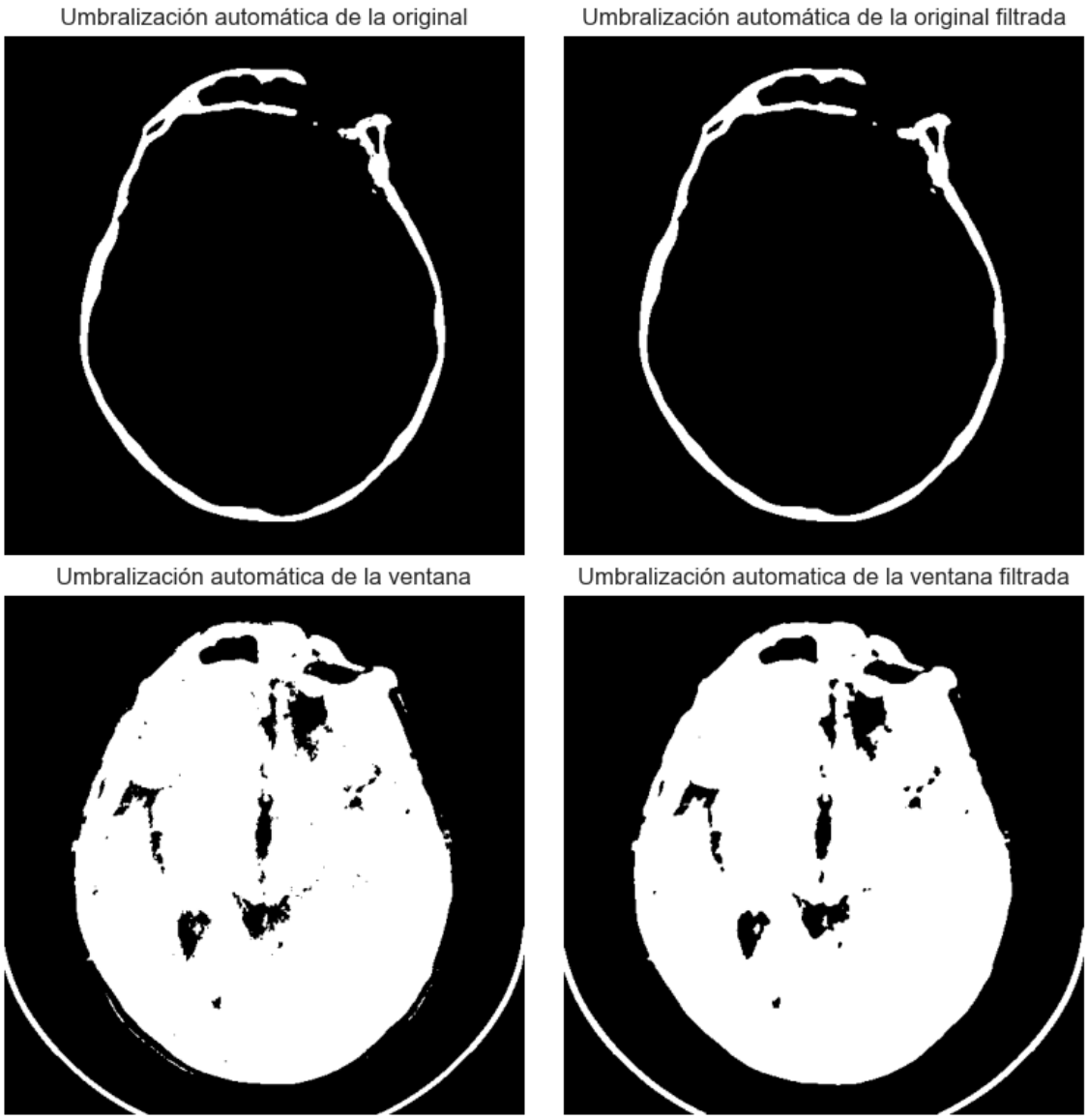

Figura 7: Comparación imágenes prefiltradas vs sin filtrar.

4. Ejercicio 2

4.1. Enunciado

Utilizar el algoritmo **SLIC (simple linear iterative clustering)** para generar superpíxeles sobre la imagen utilizada y evaluar el efecto que su número tiene sobre el resultado. Puede utilizar la función de Matlab llamada superpixels(im,N). Esta función da un buen resultado cuando los superpíxeles delimitan con nitidez la zona de interés, ya que puede aplicarse posteriormente un análisis cuantitativo de cada región delimitada. El objetivo es encontrar un valor de N adecuado para ello.

4.2. SLIC (Simple linear iterative clustering)

El algoritmo SLIC se diseñó para ser eficiente computacionalmente y se usa a menudo como paso previo a otros algoritmos. Este crea una malla de puntos sobre los que aplica el algortimo de k-means, teniendo en cuenta el color de los píxeles y su posición. Al final, genera unos superpíxeles, cuya forma podemos ajustar mediante el parámetro de compacidad, y cuyo significado es una región homogénea de píxeles, por lo que tienden a seguir los bordes de la imagen.

```
# Inicializamos las imagenes en 8bpp
imagen_8bpp, _ = imagen_bits(imagen, 8)

# Configuramos los parámetros del algoritmo SLIC
tam_superpixeles_o = 10
compacidad_o = 5

# Creamos el objeto SLIC
slic_o = cv2.ximgproc.createSuperpixelSLIC(imagen_8bpp,
region_size=tam_superpixeles_o, ruler=compacidad_o)

# Realizamos la segmentación
slic_o.iterate(100)

# Obtenemos máscara de contornos de superpíxeles
mascara_contornos_o = slic_o.getLabelContourMask()

# Pintamos los contornos de los superpíxeles en la imagen original
resultado_o = cv2.cvtColor(imagen_8bpp, cv2.COLOR_BGR2RGB)
resultado_o[mascara_contornos_o > 0] = [0, 255, 255]
```

Listing 5: Código del SLIC.

Para su implementación se ha usado la librería OpenCv. La explicación del código se puede ver en el mismo código previamente mostrado. Únicamente queda reflejado el usado para la imagen original, ya que de igual forma se hace con la ventana.

Los parámetros a ajustar son:

1. **Tamaño de los superpíxeles**: distancia entre los centroides con los que se inicializa el algoritmo de k-means, formando la malla.
2. **Compacidad**: un mayor valor dará como resultado superpíxeles con un contorno más parecido al de una circunferencia.
3. **Numero de iteraciones**: veces que se itera el algoritmo de k-means.

Después de ajustar manualmente los parámetros, se obtienen los siguientes valores para las imágenes.

Imagen	**Tamaño sp.**	**Compacidad**	**Iteraciones**
Original	10	5	100
Ventana	12	1	20

Cuadro 2: Parámetros del SLIC.

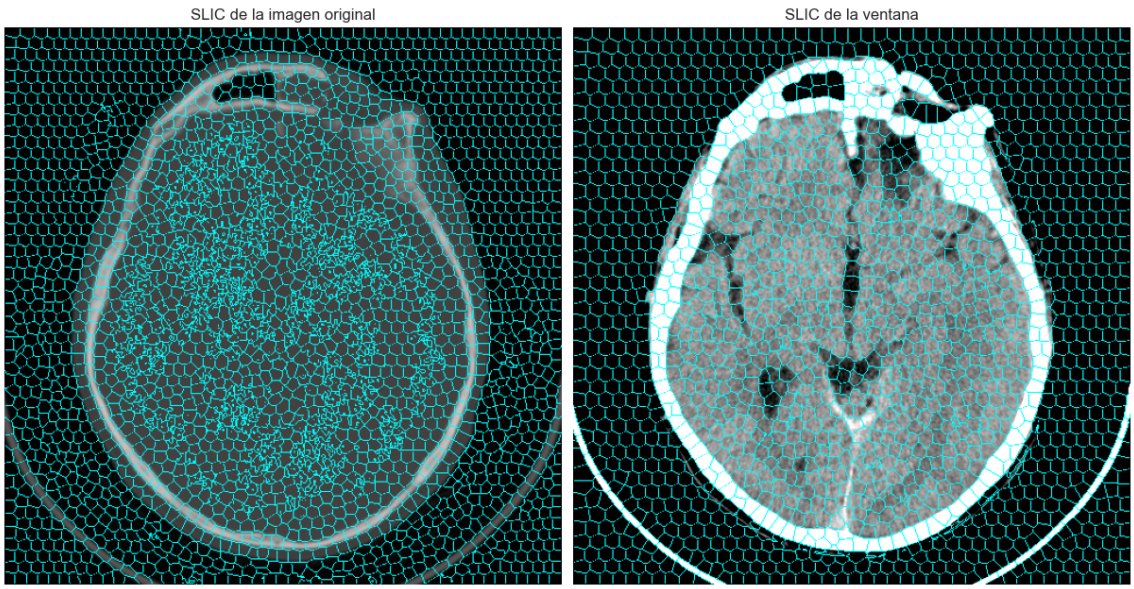

Figura 8: SLIC de la original y la ventana.

4.3. Conclusión SLIC

De acuerdo con la imagen original, el algoritmo se adapta bien a los bordes de la imagen, aunque se puede ver que en el tejido blando no se ha conseguido un

buen resultado. Probablemente, el resultado puede seguir siendo usado para algún procesado posterior, aunque no es óptimo.

Por otro lado, ha dado un muy buen resultado en la ventana, donde se ha adaptado casi perfectamente a todos los bordes. En caso de usar una de las segmentaciones, la de la ventana sería la más idónea.

Por último, cabe destacar la rápida ejecución del algoritmo, el cual en el caso de la ventana ha tardado solamente un tercio de segundo.

5. Licencia y repositorio.

El código completo se encuentra en mi repositorio de Github.