Práctica 4. Segmentación de imágenes II

Gonzalo Mesas Aranda

24 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

Utilizar uno o varios (en cuyo caso la calificación será mejor) de los algoritmos de segmentación de imágenes (detección de bordes, crecimiento de regiones, contornos activos, etc.) para obtener uno o varios de los objetos de interés contenidos en la imagen utilizada. El profesor de la asignatura sugerirá opciones durante la sesión de práctica y proporcionará consejo sobre como conseguir los resultados, pero el trabajo y la elección debe ser individual. El resultado de estos algoritmos debe ser una imagen binaria.

3.2. Crecimiento de regiones

Con el propósito de obtener una máscara de la zona de interés de la imagen, se ha elegido el algoritmo de crecimiento de regiones. En el caso de la imagen 8, imagen estudiada en esta práctica, la zona de interés (ROI) es el tejido blando. En la imagen original, esta región carece de contraste, es por eso que se ha tomado como imagen base para la realización de la práctica la ventana.

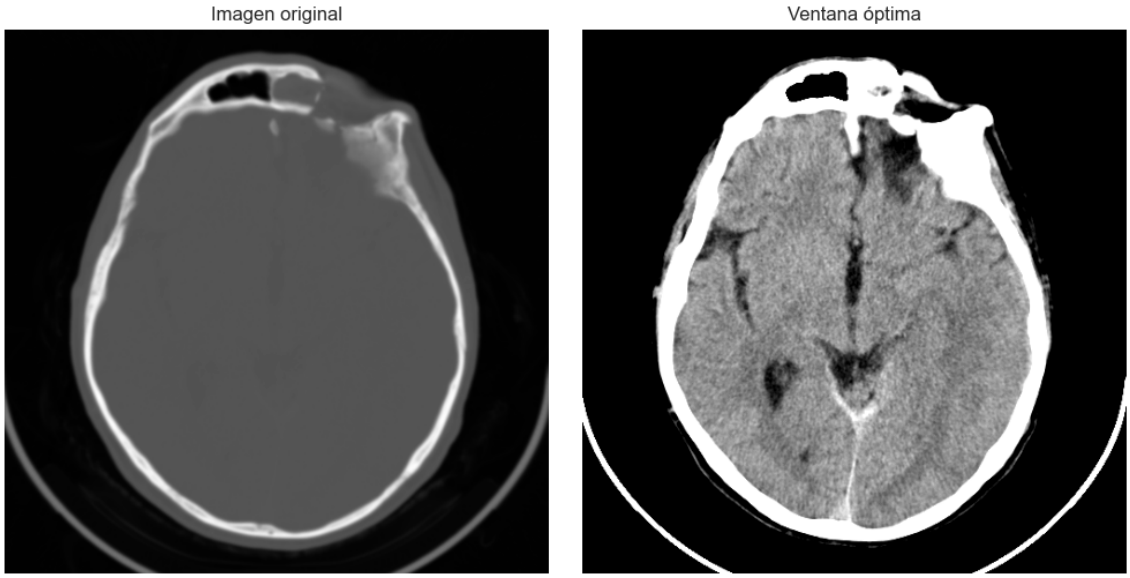

Figura 1: Imagen original y ventana de tejido cerebral.

El algoritmo tiene como finalidad definir una región en la imagen cuyos píxeles cumplan un criterio de homogeneidad. Funciona de la siguiente manera:

- Se inicializa el algoritmo con un píxel (semilla), colocado sobre la región que se quiere segmentar.

- Se añaden los píxeles de alrededor a una lista de píxeles a comprobar (posible crecimiento).
- Para cada uno de estos píxeles, se comprueba si cumple el criterio de homogeneidad. Si lo cumple, se añade a la región, se actualizan los parámetros de homogeneidad y se añaden los píxeles de alrededor que no hayan sido comprobados con anterioridad a la lista.
- El proceso continua hasta que no haya más píxeles por comprobar. El resultado es una imagen binaria con la región de interés.

Como criterio para determinar si un píxel pertenece o no a la región, se ha considerado uno de los propuestos en las diapositivas de clase. El único parámetro que se debe ajustar, es el denominado umbral, el cual será menor cuanto más .estricto"se quiera que sea el algoritmo a la hora de decidir.

```
def crecimiento_region(imagen, semilla, umbral):
    # Tamaño de la imagen
    rows, columns = imagen.shape

    # Pixeles que ya se han comprobado / mascara final
    pixeles_visitados = np.zeros_like(imagen, dtype=np.bool_)
    region = np.zeros_like(imagen, dtype=np.bool_)

    # Lista con los valores a comprobar
    lista = [semilla]

    # Inicializacion de la media y desviacion
    numero_pixeles = 1
    media = imagen[semilla[0], semilla[1]]
    desviacion = 0

    # Comprobamos los valores de la lista
    while len(lista) > 0:
        x, y = lista.pop()
        pixeles_visitados[x, y] = True

        # Comprobamos condicion de crecimiento
        if abs(imagen[x,y] - media) <= (1-(desviacion/media))*umbral:
            # Añadimos a la mascara final
            region[x,y] = True
```

Listing 2: Crecimiento de regiones parte primera.

```
            # Actualizamos los criterios
            media = ((numero_pixeles*media)+imagen[x,y])/(numero_pixeles+1)
            numero_pixeles += 1
            desviacion = np.std(imagen, where=region==True)

            # Añadimos los pixeles con conectividad 8 a la cola
            for i in range(max(0, x - 1), min(x + 2, rows)):
                for j in range(max(0, y - 1), min(y + 2, columns)):
                    if pixeles_visitados[i,j] == False:
                        lista.append([i,j])
    return region
```

Listing 3: Crecimiento de regiones parte segunda.

A continuación se determina la semilla, para la cual se ha elegido la posición $(210, 220)$.

Figura 2: Semilla usada para el crecimiento de regiones.

El algoritmo ha sido aplicado a la imagen filtrada (gauss) y sin filtrar, con umbrales de 2 y 22 respectivamente. El resultado es el siguiente:

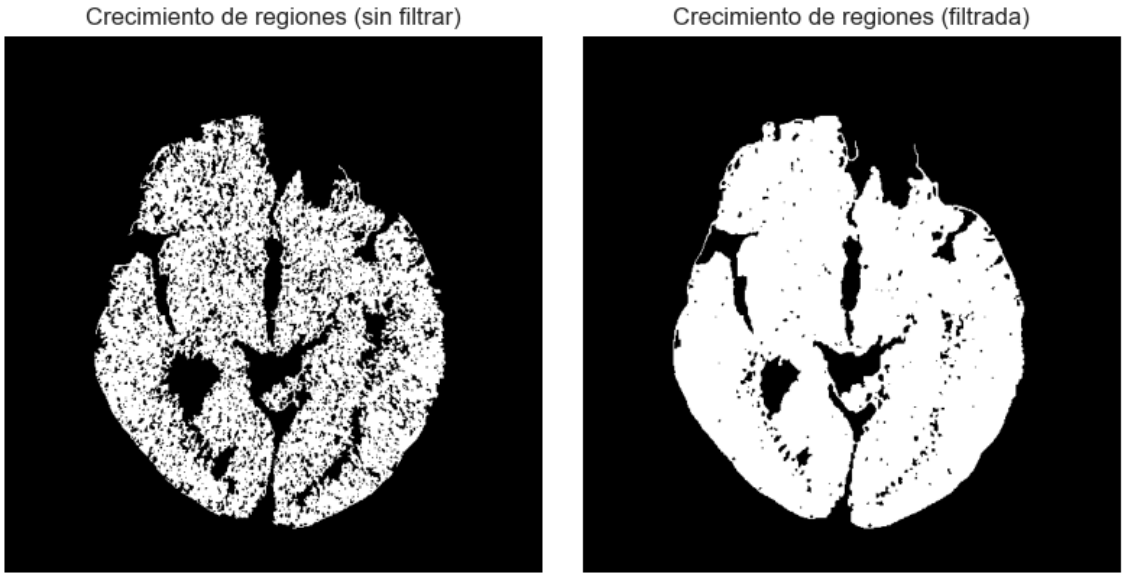

Figura 3: Crecimiento de regiones sobre la imagen filtrada y sin filtrar.

3.3. Conclusión del crecimiento de regiones

El resultado en ambos casos es bueno, se adapta bastante bien a la zona de interés. Sin embargo, se puede considerar mejor el de la imagen filtrada, ya que no contiene tanto ruido como la imagen original.

Esto se debe a que, al suavizarse la imagen previamente, hay menos valores que se salgan anormalmente del rango de interés, aún estando en la ROI. Además, esto explica el bajo valor del umbral utilizado (2), respecto al umbral que se usa con la original (22).

4. Ejercicio 2

4.1. Enunciado

Con la imagen (o imágenes, en caso de emplear varios algoritmos) obtenida en el apartado anterior, realizar un filtrado morfológico que permita obtener una mejora (eliminar pequeños objetos indeseados, rellenar objetos, suavizar los bordes del objeto, ...) del resultado obtenido por el algoritmo de segmentación utilizado. Puede utilizar la función de Matlab llamada bwmorph(...).

4.2. Cierre

Como se puede ver en el ejercicio anterior, el defecto de la segmentación es la presencia de ruido en forma de huecos en la máscara. Para intentar solucionar este problema, se va a aplicar un cierre, el cual tiene la función de rellenar estos huecos, entre otras.

El cierre se define como una dilatación seguida de una erosión. A su vez, se puede implementar el funcionamiento de la erosión mediante la dilatación del complementario de la máscara, es decir, el fondo y los huecos. La dilatación se puede realizar con un elemento estructurante, el cual incluye en la máscara el píxel actual cuando se produce un hit. Se propone el siguiente algoritmo para la dilatación y erosión:

```
def dilatacion(imagen, elem_estruc):
    t = elem_estruc.shape[0]//2
    rows, columns = imagen.shape
    imagen_dilatada = np.zeros_like(imagen, dtype=np.bool_)

    for x in range(t, rows-t-1):
        for y in range(t, columns-t-1):
            convolucion = np.logical_and(
            imagen[x-t:x+t+1, y-t:y+t+1], elem_estruc)
            if True in convolucion:
                imagen_dilatada[x,y] = True
    return imagen_dilatada
```

Listing 4: Código para la dilatación.

```
def erosion(imagen, elem_estruc):
    imagen_inversa = np.logical_not(imagen)
    erosion = np.logical_not(dilatacion(imagen_inversa, elem_estruc))

    # Correción de los bordes
    erosion[0,:] = False
    erosion[-1,:] = False
    erosion[:,0] = False
    erosion[:,-1] = False
    return erosion
```

Listing 5: Código para la erosión.

El elemento estructurante elegido es una matriz 3x3, el cual va a permitir realizar el cierre sin que algunas estructuras que están separadas pero próximas se unan, y por tanto perder más información de la que se gana.

El resultado de aplicar el cierre a las dos máscaras obtenidas en el apartado anterior es:

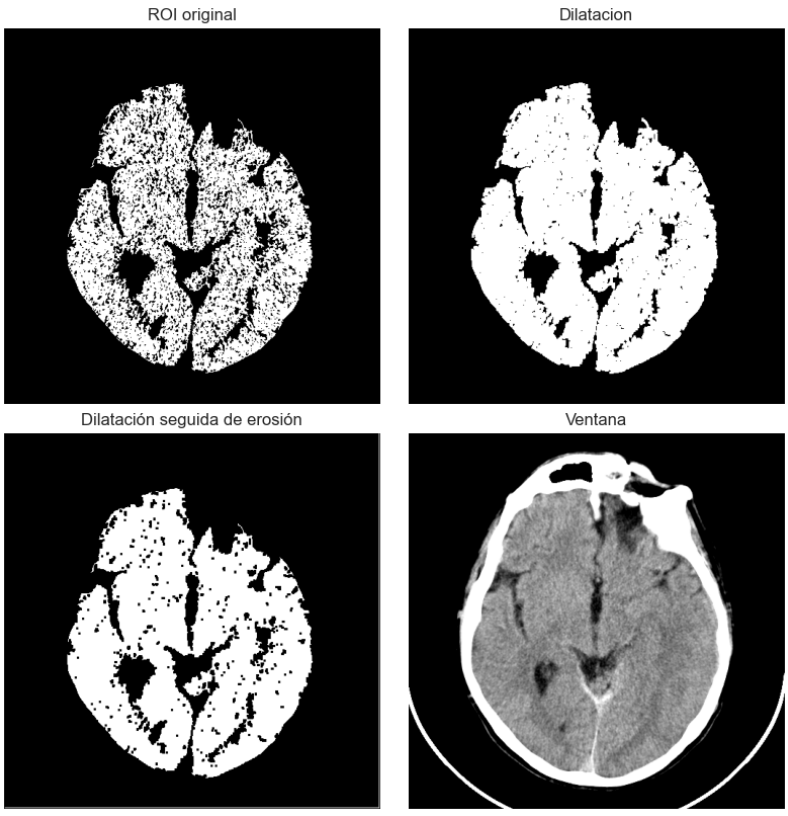

Figura 4: Cierre sobre la máscara sin filtrar.

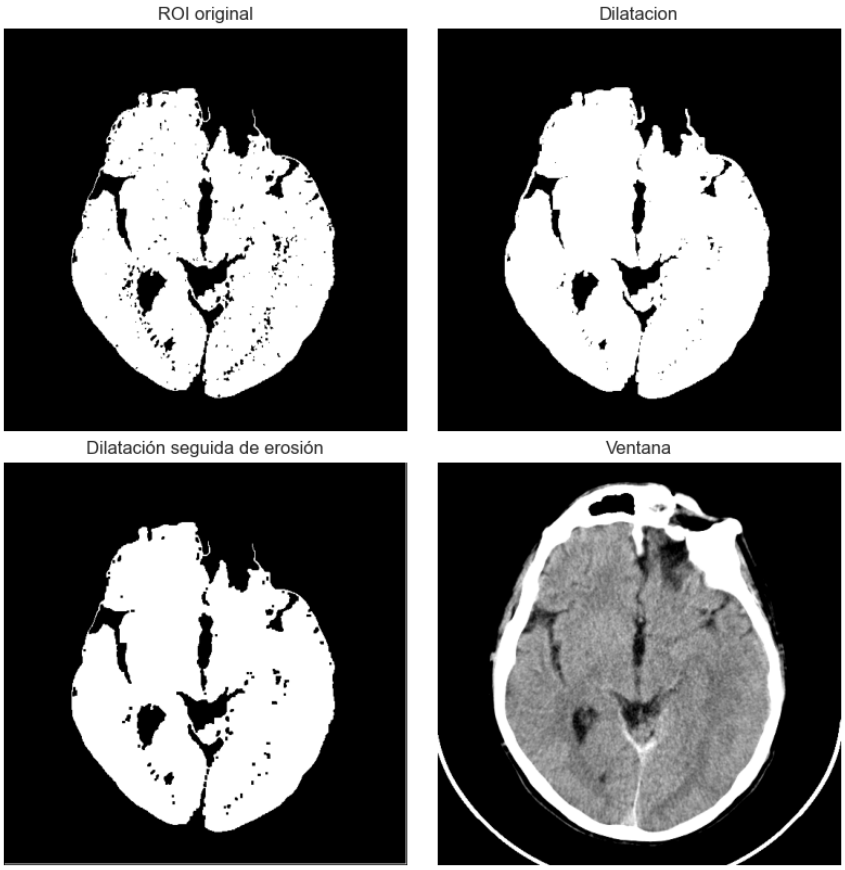

Figura 5: Cierre sobre la máscara filtrada.

4.3. Conclusión del cierre

El cierre ha sido efectivo en ambas situaciones, sin embargo, como la máscara de la imagen filtrada tenía menos huecos y más pequeños, el resultado es mejor. Usar un elemento estructurante más grande no se ha considerado ya que, al aplicarlo se cierran estructuras que se consideran importantes que estén separadas, como se comentó anteriormente.

Para concluir, el cierre de la máscara filtrada es el mejor resultado hasta el momento.

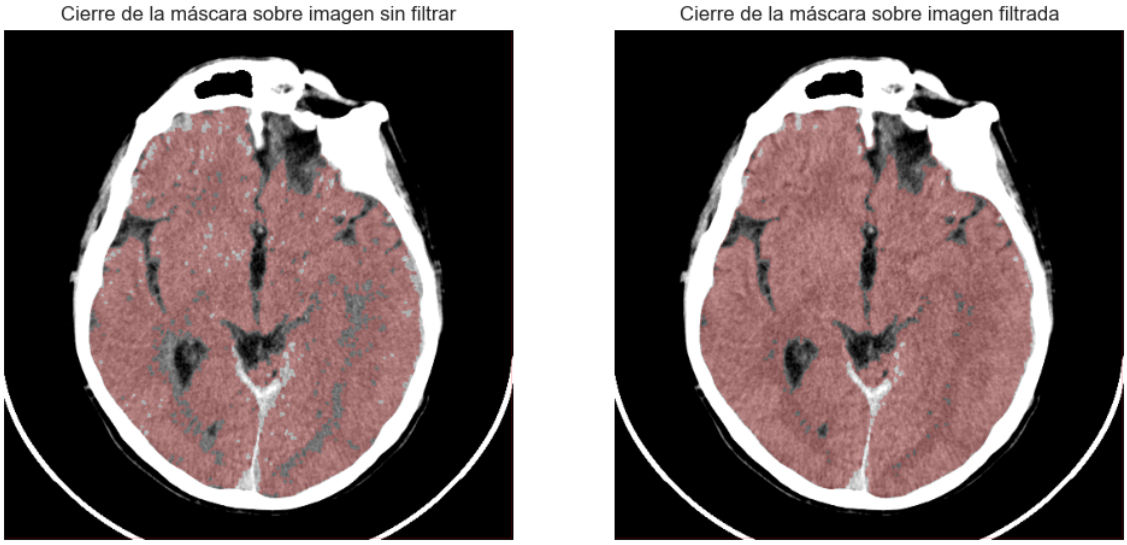

Figura 6: Máscaras sobre la imagen original.

5. Ejercicio 3

5.1. Enunciado

Obtener descriptores básicos de cada uno de los objetos de interés, entre ellos el área, perímetro y compacidad. Como parámetro opcional, se propone la obtención de la firma del contorno [1-3] y de algunos momentos básicos (media, desviación típica, skewness, ...). Puede utilizar la función de Matlab llamada regionprops(...).

5.2. Descriptores básicos

Para la obtención de algunos de los descriptores se ha utilizado la librería OpenCv. Para obtener el área, el perímetro y la compacidad es necesario disponer del contorno de nuestra zona de interés. Es de eso de lo que se encarga la función $cv2.findContours$ junto con el argumento $cv2.RETR\_EXTERNAL$, el cual selecciona el contorno más exterior y no los interiores que no son de interés.

```
# Contorno externo
contornos, _ = cv2.findContours(ROI_cierre_f.astype(np.uint8),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
primer_contorno = contornos[0]

# Calculamos el perímetro, área y compacidad
perimetro = cv2.arcLength(primer_contorno, closed=True)
area = cv2.contourArea(primer_contorno)
compacidad = (perimetro**2) / area

print("Perímetro:", round(perimetro, 1))
print("Área:", round(area, 1))
print("Compacidad:", round(compacidad, 1))
print("Media:", round(np.mean(v, where=ROI_cierre_f==True), 1))
print("Desviación estándar:", round(
np.std(v, where=ROI_cierre_f==True), 1))

valores, cuenta = np.unique(v, return_counts= True)
cuenta = cuenta[1:-1]
print("Moda:", valores[np.argmax(cuenta)-1])
```

Listing 6: Descriptores básicos.

Los descriptores de la zona de interés de la imagen se muestran a continuación. Cabe destacar que el perímetro y el área tiene como unidad píxeles y píxeles^2 respectivamente. La compacidad y circularidad son adimensionales.

Parámetro	**Valor**
Perímetro	1625.4
Área	91880.0
Compacidad	28.8
Circularidad	0.4
Media	141.6
Desviación estándar	23.2
Moda	132

Cuadro 1: Descriptores básicos

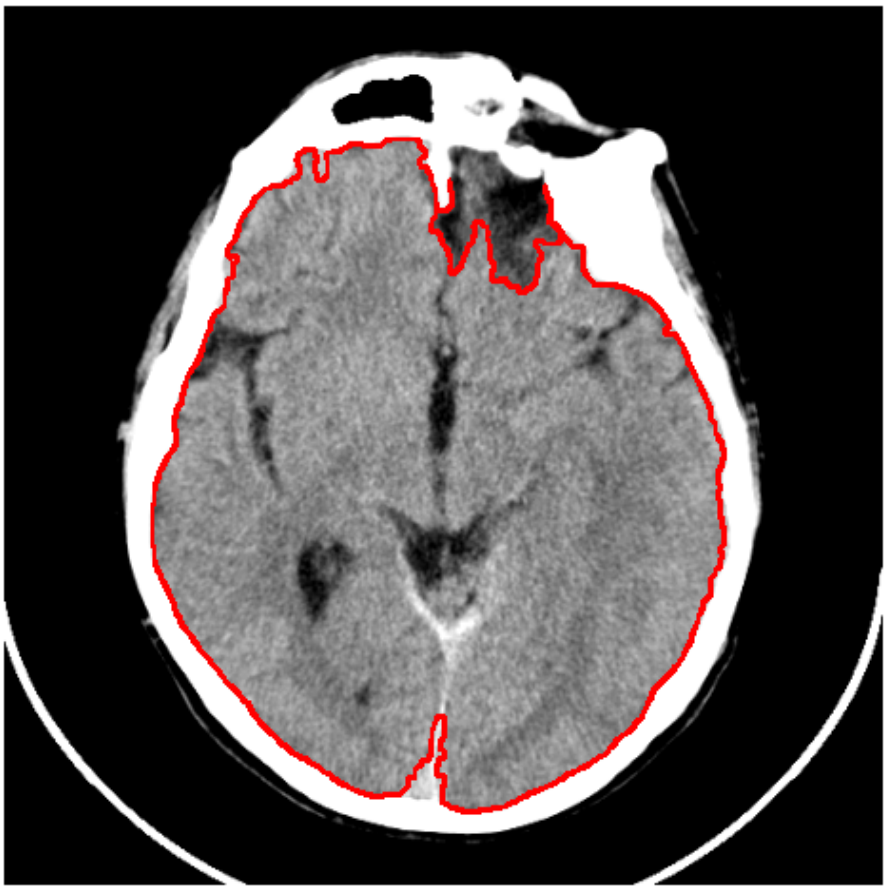

Figura 7: Contorno usado para los descriptores.

5.3. Conclusión

El contorno obtenido para el calculo de algunos parámetros es bastante acertado, por lo que las medidas obtenidas se pueden considerar precisas.

En cuanto a los descriptores, ofrecen información adicional sobre la zona de interés, la cual en algunos casos es crucial para el diagnóstico diferencial, aunque no sea el caso de esta imagen. Sin embargo, se puede observar que a partir de una buena segmentación, es sencillo obtener este tipo de datos.

6. Licencia y repositorio.

El código completo se encuentra en mi repositorio de Github.

Licencia: MIT License