Práctica 1. Algoritmos básicos de tratamiento digital de imágenes biomédicas

Gonzalo Mesas Aranda

31 de octubre 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. Ejercicio 1

2.1. Enunciado

Abrir la imagen o imágenes que le correspondan, creando una **matriz de** datos donde almacenar los **valores de los píxeles**. Una vez abierta, debe comprobar que los **metadatos** son correctos y determinar los **parámetros básicos de la imagen**, como el número de filas, columnas o número de bits por píxel. **Describir el tipo de imagen** obtenida (Radiología, TC, etc.) y su contenido anatómico (básico).

2.2. Acceso a metadatos y apertura de la imagen

Para empezar, se importan las librerías que se van a usar a lo largo de la práctica.

```
import numpy as np
import pydicom as pdc
import matplotlib.pyplot as plt
%matplotlib inline
```

Listing 1: Librerías usadas para el proyecto.

- Numpy: guardar la imagen como una matriz y realizar operaciones sobre ella.
- Pydicom: permite abrir y tratar ficheros en formato DICOM.
- Matplotlib: se usará para todo lo relacionado con gráficas y visualización de las imágenes.

Se procede a la apertura de la imagen y se accede a los metadatos. Hay un gran número de estos, por lo que se ha dado un vista de los mismos en todo su formato (Ver cuadro 1) y al final una selección de los más relevantes (Ver cuadro 2).

Para la apertura (Ver Listing 2) hemos usado la función *dcmread()* de la librería pydicom. Esta imagen contiene tanto los metadatos como la imagen en sí.

```python
# Abrir la imagen
def abrir_imagen(num_imagen):
    return pdc.dcmread(f"./im{num_imagen}.dcm")

imagen = abrir_imagen(8)

# Metadatos de la imagen
print(imagen)

# Metadatos importantes
print("Bits por pixel:", imagen.BitsStored)
print("Modalidad:", imagen.Modality)
print("Descripción del estudio:",imagen.StudyDescription)
print("Numero de filas:", imagen.Rows)
print("Numero de columnas:", imagen.Columns)
```

Listing 2: Apertura de la imagen y acceso a metadatos.

Dataset.file_meta	
(0002, 0010) Transfer Syntax UID	UI: Explicit VR Little Endian
(0002, 0012) Implementation Class UID	UI: 1.3.6.1.4.1.5962.99.2
(0002, 0013) Implementation Version Name	SH: 'PIXELMEDJAVA001'
(0002, 0016) Source Application Entity Title	AE: 'LASKER_11112'
(0008, 0016) SOP Class UID	UI: CT Image Storage
(0018, 1140) Rotation Direction	CS: 'CW'
(0018, 1150) Exposure Time	IS: '1500'
(0028, 0002) Samples per Pixel	US: 1
(0028, 0004) Photometric Interpretation	CS: 'MONOCHROME2'

Cuadro 1: Vista de metadatos en todo su formato.

Metadato	**Valor**
Bits por pixel	12
Modalidad	CT
Descripción del estudio	Cabezaˆ HeadSeq (Adulto)
Número de filas	512
Número de columnas	512

Cuadro 2: Metadatos relevantes.

Se definen dos funciones útiles (Ver Listing 3). La primera carga la imagen como una matriz con los valores de los píxeles y la devuelve junto al valor de bits por píxel elegido, normalmente 8 o 12 (por defecto). En segundo lugar, una función que guarda una cualquier imagen, con 8 bpp. Esta función será usada a lo largo del documento.

Se crea la matriz de la imagen con 12 bpp, ya que es la totalidad de la información que tenemos por píxel, es decir la mayor calidad posible. A lo largo del documento, esto se debe tener en cuenta por ejemplo para el histograma.

```
# 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

# Guarda una imagen con 8 bpp y con un nombre
def guardar_8b(imagen, titulo):
    im = (np.array((imagen) * (255/4095))).astype(np.uint8)
    plt.plot(im)
    plt.show(False)
    plt.savefig(titulo)

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

Listing 3: Funciones de creación de imagen y guardado.

Se visualiza la imagen usando Matplotlib (Ver Listing 4). En la función *imshow()* se ha usado 'gray' como color de mapa, que toma los valores de la imagen con un valor en escala de grises.

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

Listing 4: Visualización de la imagen.

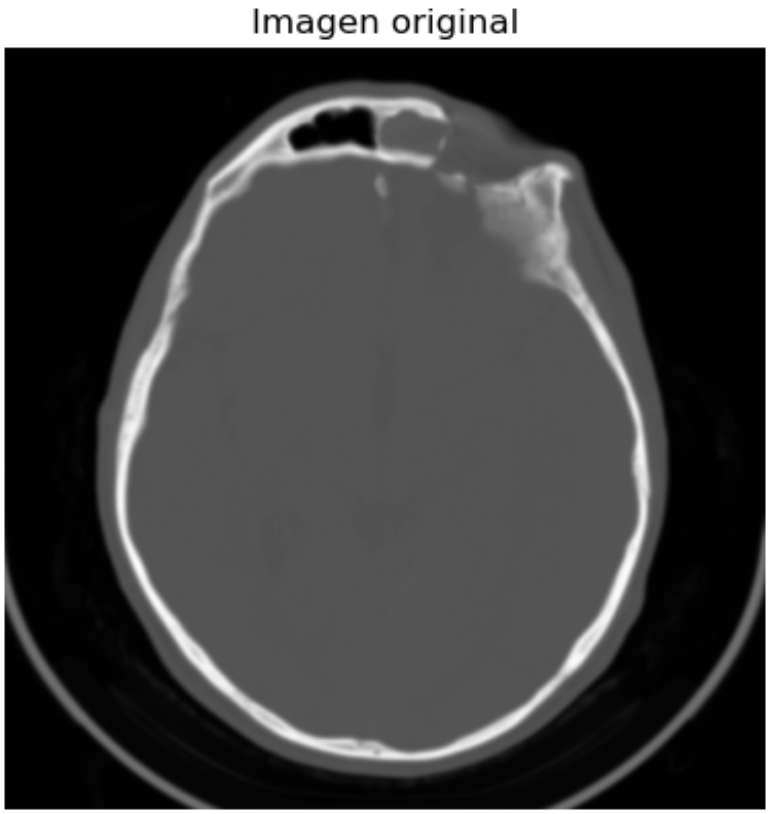

Figura 1: Imagen original.

2.3. Contenido anatómico

Esta imagen médica se ha obtenido mediante tomografía computarizada (TC). Se puede deducir ya que por un lado, es uno de los valores que nos indican los metadatos (Ver Cuadro 2). Además, se trata de un plano transversal en el cual el contenido de los tejidos blandos (tejido encefálico en este caso) no se puede distinguir adecuadamente.

El corte se encuentra en la parte superior a las cavidades oculares, ya que lo que se puede ver en la parte de arriba de la imagen parecen ser los senos paranasales frontales. Así mismo, en la zona del seno frontal izquierdo se puede observar una anomalía, la cual podría tratarse de una fractura.

3. Ejercicio 2

3.1. Enunciado

Realizar el **cálculo del histograma** de dos formas independientes y visualizarlo en un gráfico de forma óptima (de forma que se vean con claridad los picos que corresponden a las zonas de interés de la imagen, que podrán ser identificadas, para lo que deberá definir los valores de los rangos adecuados en los ejes x e y):

- Utilizando la rutina imhist(...) de Matlab (con los parámetros adecuados).
- Mediante un algoritmo que deberá programar, que obtenga un resultado similar al de la rutina de Matlab (aquí intentamos aprender a programar nosotros mismos lo mismo que hace Matlab).

Nota: bins es el nombre que se da en estadística al número de intervalos (variable aleatoria discreta) en que se divide el rango de la variable (en este caso, el nivel de gris) para calcular el histograma.

3.2. Primera forma

Debido a que no se está usando Matlab para el desarrollo de la práctica, se ha decidido usar un método lo más automático posible para que se asemeje al concepto original. Para ello se ha usado la función *np.histogram()*, la cual lo permite hacer de manera automática indicándole los argumentos necesarios (Ver Listing 5).

```
def histograma_1(imagen, zoom_min = 0, zoom_max = (2**bpp)-1):
    # Creamos el histograma
    histogram, bins = np.histogram(imagen.flatten(), bins = 2**bpp,
    range = (0, 2**bpp), density = True)
    # Visualizamos las imágenes
    _, ax = plt.subplots(1, 2, figsize = (12, 4), layout = "constrained")
    ax[0].imshow(imagen_pixeles, cmap = "gray")
    ax[0].axis("off")
    ax[0].set_title("Imagen")
    ax[1].plot(bins[:-1], histogram)
    ax[1].set_title("Histograma")
    # Zoom
    if zoom_min != 0 or zoom_max != (2 ** bpp)-1:
        ax[1].set(xlim=(zoom_min, zoom_max))

histograma_1(imagen_pixeles)
```

Listing 5: Código de la primera forma del histograma.

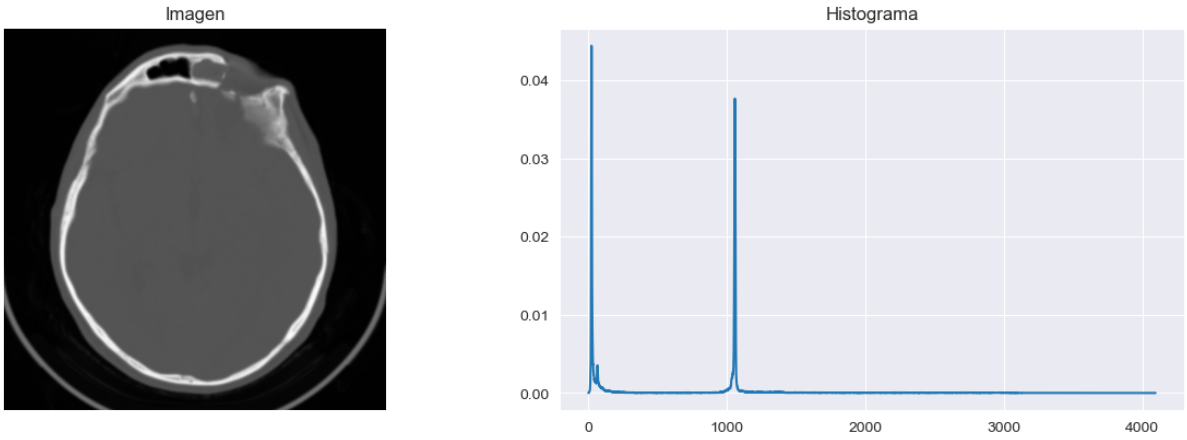

Figura 2: Primera forma del histograma, original.

Para poder ver el histograma con más detalle se ha implementado un zoom que permite poner el foco en las zonas de interés (Ver Listing 5).

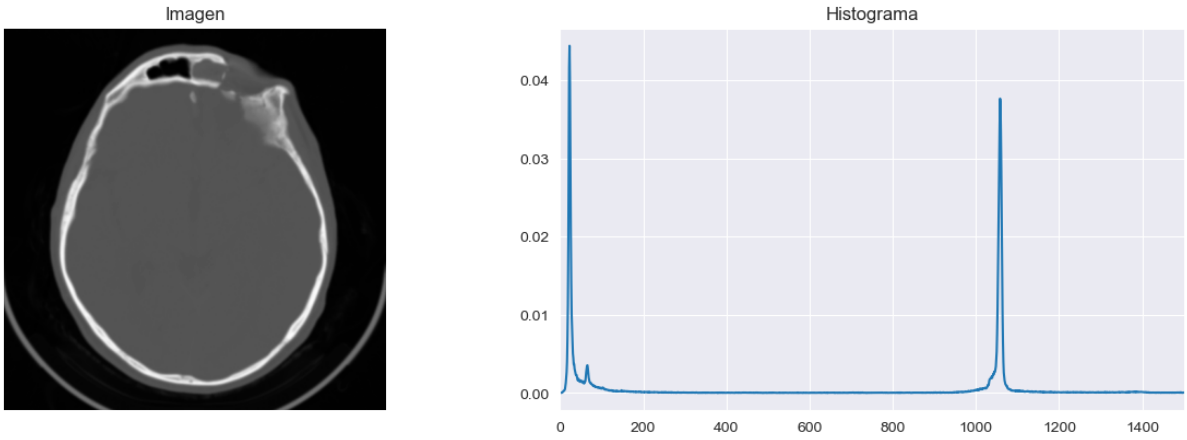

Figura 3: Primera forma del histograma, con zoom.

Se pueden ver dos picos fundamentalmente:

- Pico más cercano a cero: se corresponde con los valores más oscuros, que corresponden con el fondo de la imagen.
- Pico en 1050: se corresponde con valores de gris intermedios, que corresponden al contenido encefálico.

3.3. Segunda forma

Para esta segunda forma se ha decidido usar la función *np.unique()*, que devuelve los valores únicos de nuestra matriz, además de cuantas veces aparece cada uno. A continuación se procede al cambio de frecuencia a frecuencia relativa mediante la división de la lista con la cuenta de los valores por el número total de píxeles de la imagen. Por último solo queda visualizar la imagen de la misma manera que en la forma primera.

```
def histograma_2(imagen, zoom_min = 0, zoom_max = (2**bpp)-1):
    # Calculamos las veces que aparece cada valor
    valores, cuenta = np.unique(imagen.flatten(), return_counts= True)
    # Pasamos a frecuencias relativas
    cuenta = cuenta/(512*512)
    # Visualizamos las imágenes
    _, ax = plt.subplots(1, 2, figsize = (12, 4), layout = "constrained")
    ax[0].imshow(imagen_pixeles, cmap = "gray")
    ax[0].axis("off")
    ax[0].set_title("Imagen")
    ax[1].plot(valores, cuenta)
    ax[1].set_title("Histograma")
    # Zoom
    if zoom_min != 0 or zoom_max != (2 ** bpp)-1:
        ax.set(xlim=(zoom_min, zoom_max))

histograma_2(imagen_pixeles)
```

Listing 6: Código de la segunda forma del histograma.

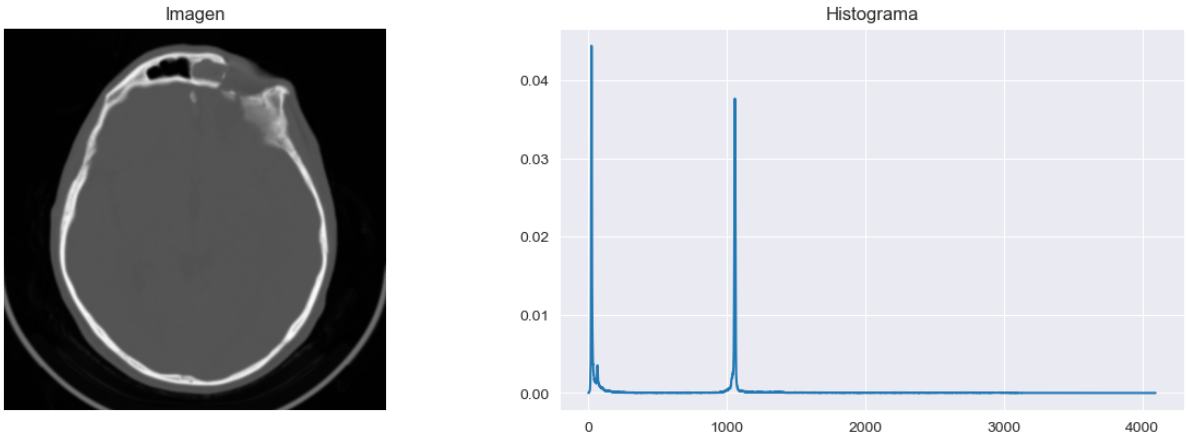

Figura 4: Segunda forma del histograma, original.

El resultado es trivialmente idéntico. La única diferencia entre los métodos es la forma de realizar los cálculos para el histograma, pero la representación se hace de la misma manera.

3.4. Conclusión

Las dos formas llevan al mismo resultado y computacionalmente son prácticamente idénticas, estando ambas en torno a los 500 ms de tiempo de ejecución.

Para complementar a la herramienta de zoom, también se ha programado una rudimentaria pero útil función que nos representa dos lineas verticales (Ver Listing 7). Esta se ha desarrollado con el objetivo de .[en]marcar"más precisamente los picos en un rango de valores que posteriormente usaremos para las ventanas.

```
def linea(arg):
    plt.vlines(x = arg, ymin = 0, ymax = 0.04, colors = "orange")
```

Listing 7: Lineas verticales en el histograma.

Con esto podemos generar un último y definitivo histograma en el que se vean adecuadamente todos los valores de interés.

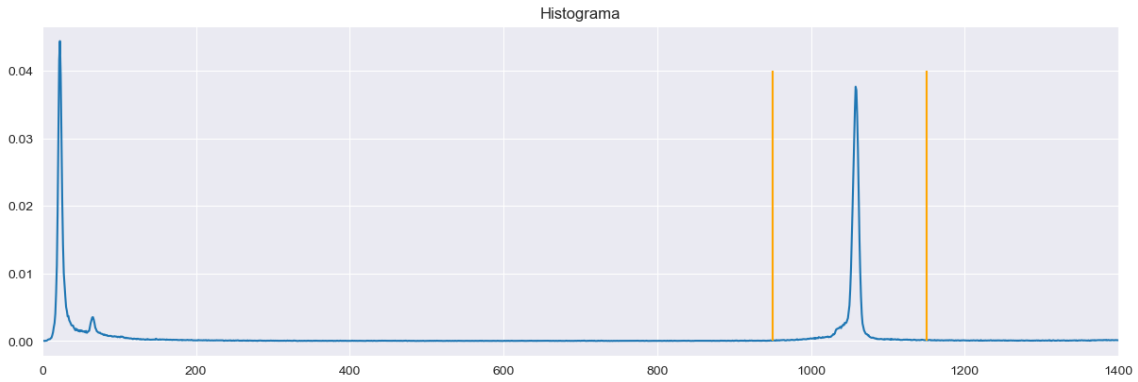

Figura 5: Histograma con zoom y valores de la ventana de tejido blando.

4. Ejercicio 3

4.1. Enunciado

Visualizar la parte más significativa de la información contenida en la imagen que te corresponda, mediante la **selección de la ventana idónea** en cada caso. Para cada imagen (según el número de la imagen que le ha sido asignada), la región de interés es:

1. Estructura ósea
2. Hígado
3. Hueso
4. Tejido graso/microcalcificaciones vasculares
5. Tumor y masa cerebral
6. Estructura ósea
7. Parénquima pulmonar
8. **Tejido cerebral**
9. Tejido graso
10. Órganos

Se recomienda usar el comando imshow(...). Tenga en cuenta que la mayoría de las imágenes médicas tienen un tipo de datos uint16 que permite más de 256 niveles de gris. La visualización es adecuada si la región de interés se muestra con un contraste adecuado y puede apreciarse con nitidez el órgano o estructura de interés.

4.2. Código y visualización

En la imagen sobre la cual se está realizando esta práctica el tejido de interés es el tejido cerebral. Además, como el paciente presenta una anomalía osea, también se dará una ventana para la observación del hueso.

La función desarrollada (Ver Listing 8)implementa la funcionalidad de una ventana cualquiera, dados los valores mínimo y máximo de la misma. Para ello todos los píxeles cuyo valor se encuentra por debajo del mínimo o por encima del máximo son asignados el valor más pequeño (0) y el más grande, de acuerdo con el tipo de dato, respectivamente.

A continuación, estos valores son reescalados para que ocupen de nuevo todo el histograma, de manera que las diferencias de contraste entre los píxeles dentro de la ventana se ven resaltadas ya que ahora hay más tonos de gris entre ellas. Por otro lado, perdemos toda la información que no se halle en la ventana.

```
def ventana(imagen, vmin, vmax):
    # Valores por debajo del min puestos a 0
    # Valores por encima del max puestos puestos
    # al valor max del tipo de datos
    clipeada = np.clip(imagen, a_min=vmin, a_max=vmax)
    # Reescalamiento para que la ventana ocupe todos los valores
    # de píxeles
    clipeada = ((clipeada - vmin)/(vmax - vmin))*((2**bpp)-1)
    # Visualización
    _, ax = plt.subplots()
    ax.imshow(clipeada, cmap = "gray")
    ax.axis("off")
    ax.set_title("Ventana")
```

Listing 8: Código de una ventana genérica.

Los valores óptimos para las ventanas mencionadas son los siguientes:

- Para el tejido cerebral los valores óptimos son 950 / 1150 (bpp = 12)
- Para el tejido cerebral los valores óptimos son 57 / 75 (bpp = 8)
- Para la ventana osea los valores óptimos son 1100 / 4000 (bpp = 12)

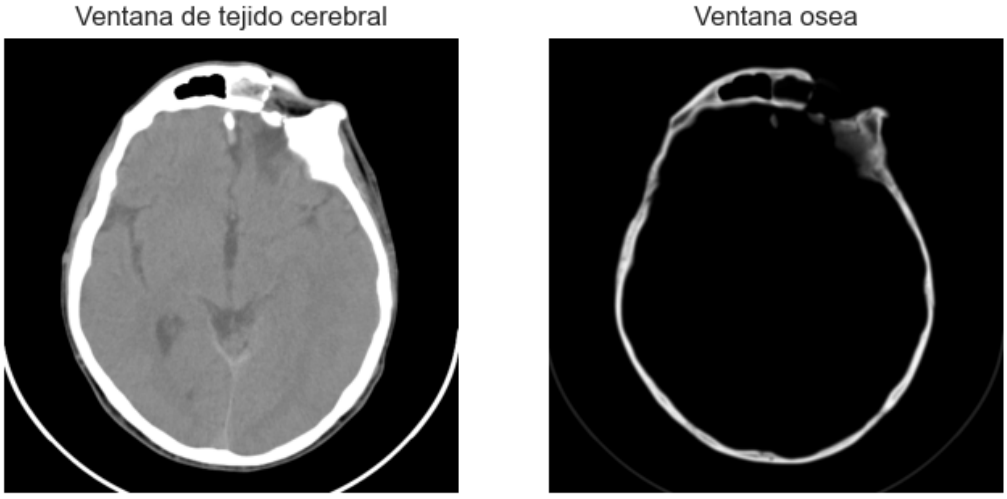

Figura 6: Ventanas ósea y de tejido cerebral.

4.3. Conclusión

Las ventanas son una manera muy efectiva para visualizar una zona de interés de una imagen. Para el tejido cerebral, estructuras no diferenciables en la imagen original, ahora son visibles. Complementariamente, la ventana ósea permite distinguir el hueso con una mayor calidad, incluso se pueden observar las zonas donde el hueso es más denso y las zonas donde menos.

Para finalizar una última comparación con la imagen original.

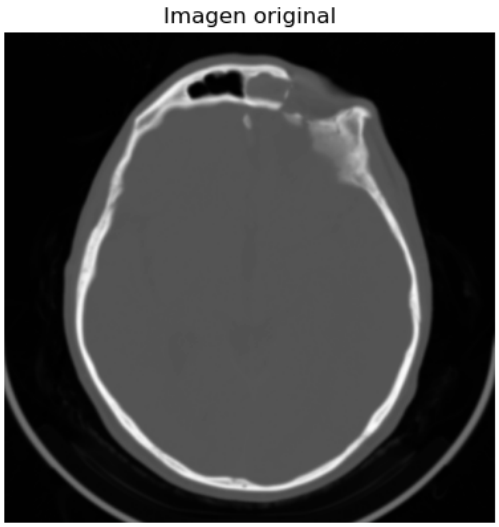

Figura 7: Imagen original comparada.

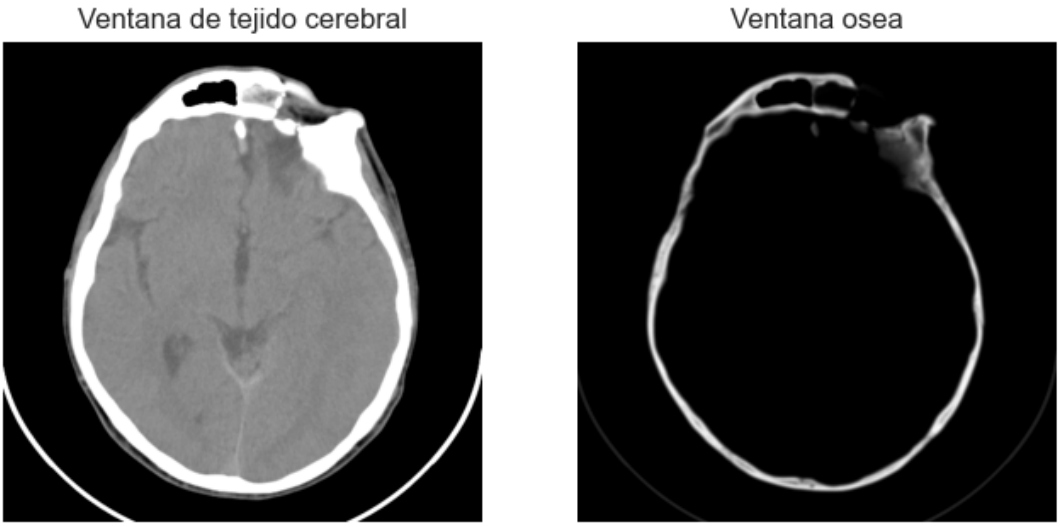

Figura 8: Ventanas ósea y de tejido cerebral comparadas.

5. Ejercicio 4

5.1. Enunciado

Transformar la imagen, de forma que se mantenga la visualización de la parte más significativa de la información de interés (ver apartado anterior) contenida en la imagen. Este tipo de algoritmos modifican el contraste de la imagen, eligiendo una ventana del rango de niveles de gris original y convirtiendo esos niveles de gris en el rango completo de niveles de gris, para mejorar su visualización subjetiva. Para la modificación del contraste, deberá utilizar:

1. La función imadjust(...) de Matlab. Los algoritmos a emplear son:
   - *a*) Negativo de la imagen.
   - *b*) Corrección gamma=2.
   - *c*) Ajuste y=sqrt(x).
   - *d*) Ajuste lineal a una ventana óptima (se entiende que eligiendo el rango de niveles de gris de interés para la visualización del órgano o estructura de interés). **Este apartado ya se ha resuelto en el ejercicio anterior y no se volverá a realizar (Ver Listing 8).**
   - *e*) Ajuste lineal a un rango de entrada comprendido entre los percentiles 5 % y el 95 % de los niveles de gris. (El rango inferior es el nivel de gris para el cual un máximo del 5 % de los píxeles no superan dicho nivel de gris, lo mismo con el rango superior en el 95 %). Para calcular los valores de nivel de gris que corresponden a los percentiles 5 % y 95 %, puede usar el histograma calculado en el apartado 2.
2. De los cinco algoritmos anteriores, indique cuál es que ofrece una mejor visualización, según su criterio subjetivo.
3. Deberá programar un algoritmo propio que obtenga un resultado similar al de la rutina de Matlab imadjust (sólo la opción, de entre las cinco, con la que se obtiene la mejor visualización).

5.2. Negativo

Para obtener el negativo de la imagen, simplemente debemos restar el valor de cada uno de los píxeles al valor máximo que puedan alcanzar. El negativo de una imagen es la imagen asociada a la ïmagen especular"del histograma de la imagen original.

```
def negativo(imagen):
    neg = ((2**bpp)-1) - imagen
    _, ax = plt.subplots()
    ax.imshow(neg, cmap = "gray")
    ax.axis("off")
    ax.set_title("Negativo")
negativo(imagen_pixeles)
```

Listing 9: Código del negativo de una imagen.

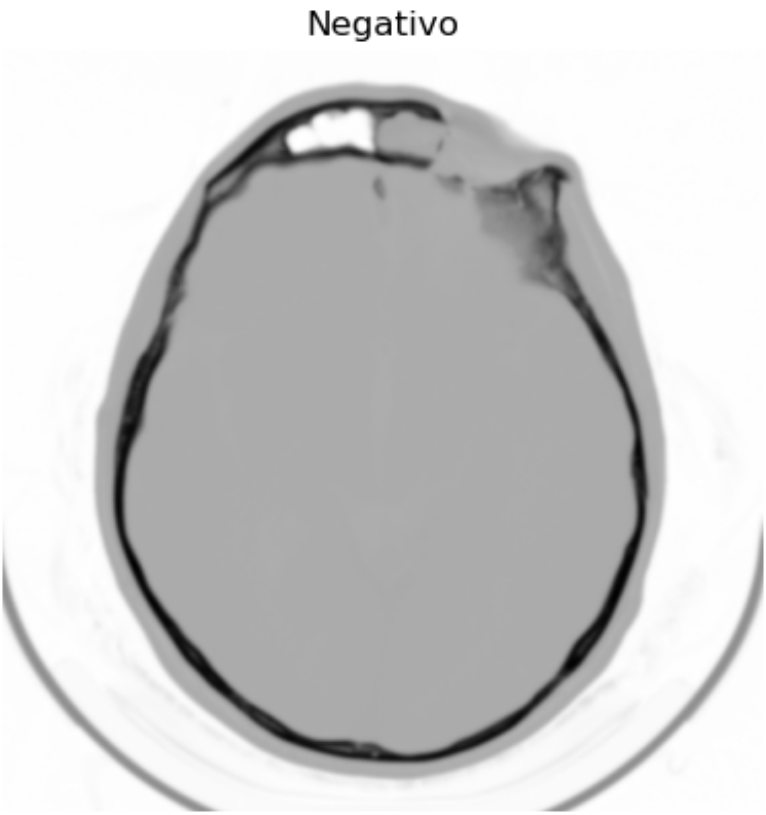

Figura 9: Negativo de la imagen original.

5.3. Conclusión negativo

El negativo de la imagen simplemente nos invierte los valores de la imagen, por lo que no da información extra y en este caso no ofrece una ayuda significativa.

5.4. Corrección gamma

La corrección gamma es otra manera de modificar el contraste de la imagen. La idea principal es elevar el valor de cada píxel al valor gamma elegido. La imagen final, por supuesto no debe cambiar el tipo de datos, por lo que se reescala.

Para este ejercicio, se nos ha encomendado usar gamma con un valor de 2, es decir elevar cada píxel al cuadrado. Esto nos deja con una imagen la cual gana contraste entre los píxeles más blancos, pero oscurece la imagen. La siguiente gráfica ilustra los valores nuevos de la imagen después de aplicar distintos valores de gamma.

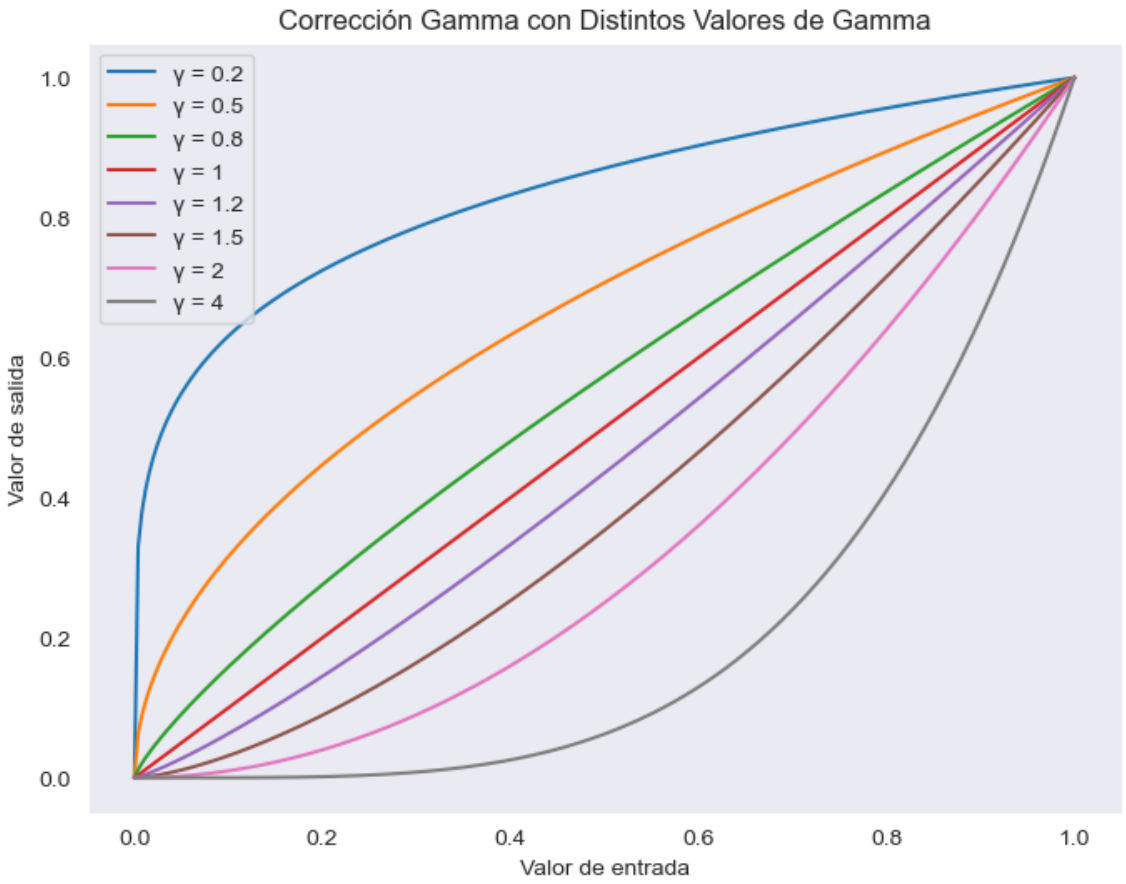

Figura 10: Corrección con distintos valores gamma.

Hay dos formas principales de programar una corrección gamma. A continuación se indica como se hacen ambas y su correspondiente implementación.

- Forma directa: se pasan los valores de los píxeles al intervalo [0, 1], se eleva cada valor al gamma elegido y se vuelve a reescalar a todo el rango de valores (Ver Listing 10).

- LUT: "lookup table."[es] un método más eficiente, ya que se calculan todas las potencias de los posibles valores y se guardan en una lista. A continuación simplemente se busca el valor correspondiente en la tabla y se sustituye en la imagen, lo cual ahorra repetir una misma operación un gran número innecesario de veces (Ver Listing 11).

```
def gamma_1(imagen, nivel):
    _, ax = plt.subplots()
    ax.imshow(((imagen/((2**bpp)-1))**nivel)*((2**bpp)-1), cmap = "gray")
    ax.axis("off")
    ax.set_title(f"Gamma = {nivel}")
```

Listing 10: Código de la forma directa para corrección gamma.

```
def gamma_2(imagen, nivel):
    potencias = (np.arange((2**bpp)-1)/((2**bpp)-1))**nivel
    def buscar(valor):
        return potencias[valor]
    func = np.vectorize(buscar)
    _, ax = plt.subplots()
    ax.imshow(func(imagen), cmap = "gray")
    ax.axis("off")
    ax.set_title(f"Gamma = {nivel}")
```

Listing 11: Código del LUT para corrección gamma.

5.5. Conclusión

La imagen final (Ver Figura 13), como se ha mencionado con anterioridad, nos da más información en los tonos claros de la imagen, en este caso el hueso. En su totalidad, la imagen se ha visto oscurecida y no ha ayudado a visualizar más adecuadamente el tejido encefálico, zona de interés.

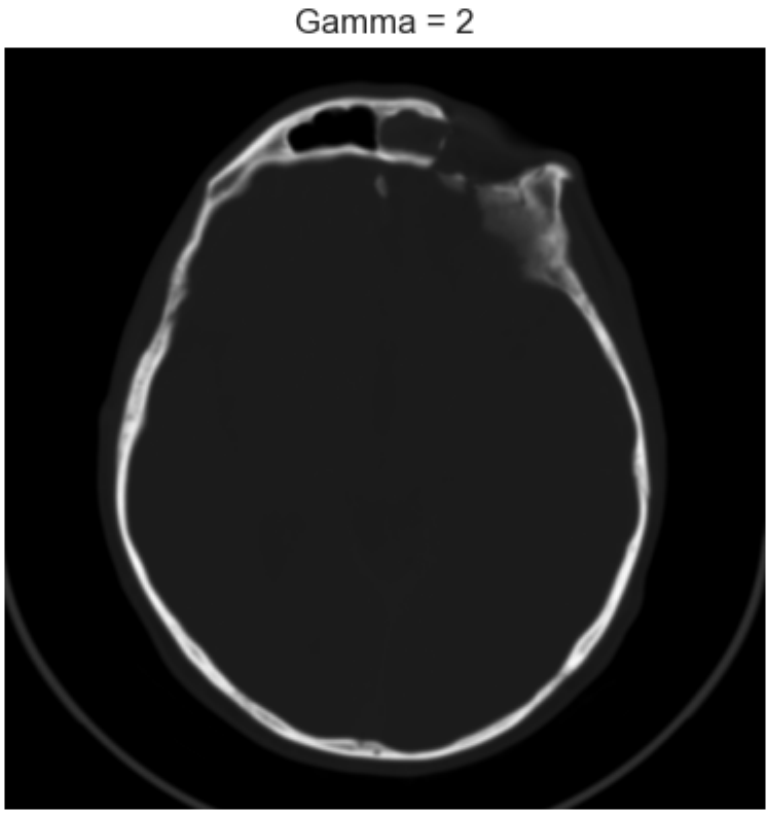

Figura 11: Corrección con gamma 2.

5.6. Ajuste Sqrt

El ajuste Sqrt es un caso particular de corrección gamma, donde el valor de ajuste es 0.5. Elevar cada píxel a un medio es el equivalente a hacer la raíz cuadrada. Se proporciona un código específico para esta corrección y su resultado.

```
def ajuste_sqrt(imagen):
    _, ax = plt.subplots()
    ax.imshow(np.sqrt(imagen), cmap = "gray")
    ax.axis("off")
    ax.set_title("Sqrt")
```

Listing 12: Código de la corrección Sqrt.

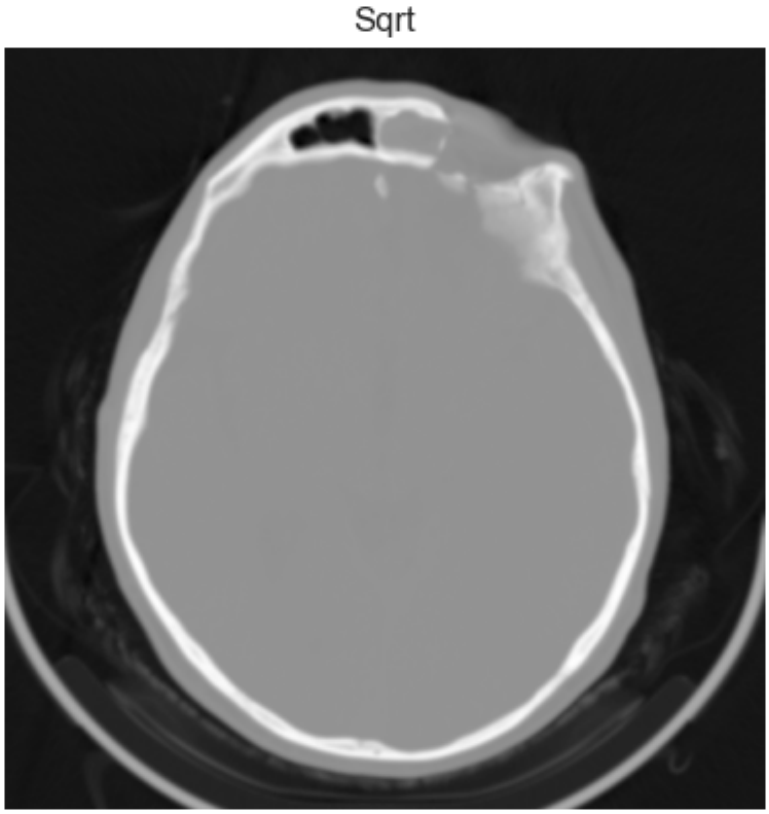

Figura 12: Corrección Sqrt.

5.7. Conclusión Sqrt

En este caso la imagen ha ganado información en las tonalidades más oscuras y en su totalidad se ha aclarecido. Sin embargo, aún no es posible distinguir estructuras de interés con claridad y no resulta muy útil.

5.8. Corrección por percentiles

El ajuste lineal por percentiles consiste en aplicar la ventana lineal resuelta anteriormente, en la que los valores entre los cuales aplica son los percentiles 5 y 95 de la imagen. Para el computo de los percentiles se ha usado la función *np.percentile()*, la cual devuelve el valor de ambos percentiles a partir de la imagen y una lista de los percentiles deseados.

```
def percentil(imagen, down_perce = 5, up_percen = 95):
    down, up = np.percentile(imagen, [down_perce, up_percen])
    ventana(imagen, vmin= down, vmax=up)
    plt.title("Percentiles")
```

Listing 13: Código de la corrección por percentiles.

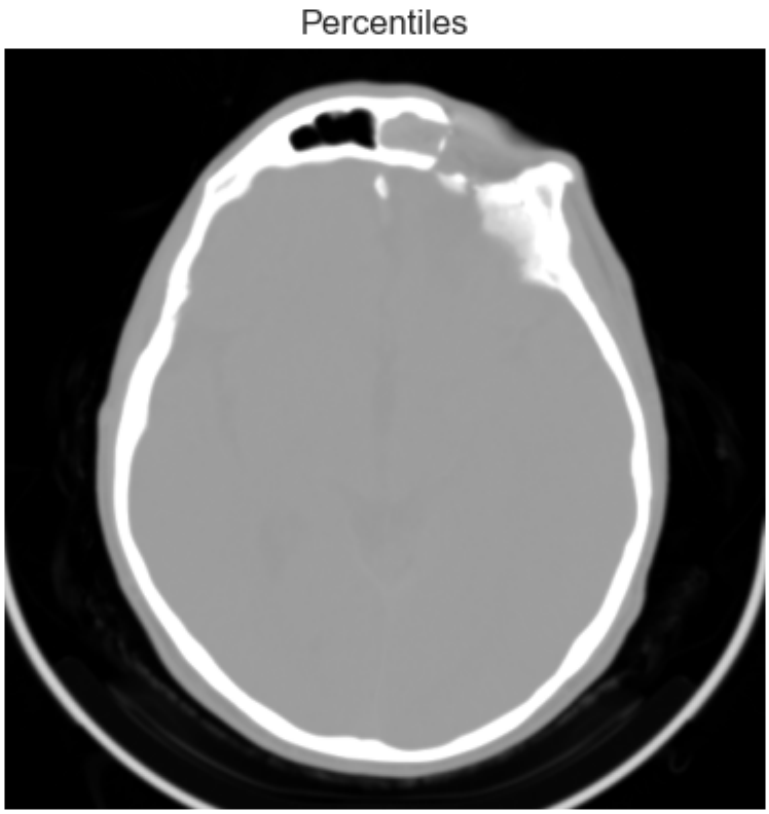

Figura 13: Corrección por percentiles.

5.9. Conclusión percentiles

Al igual que en el ajuste Sqrt, se empiezan a distinguir estructuras de interés en el encéfalo, aunque la ganancia de información es demasiado baja para que resulte útil en este caso concreto.

5.10. Conclusión

Como se puede ver en la figura resumen (Ver Figura 14), el mejor ajuste es la ventana lineal para tejido cerebral. Es la única que nos permite distinguir adecuadamente estructuras en la zona de interés. Por otro lado la corrección gamma 2 es excelente para el estudio de la zona ósea de la imagen.

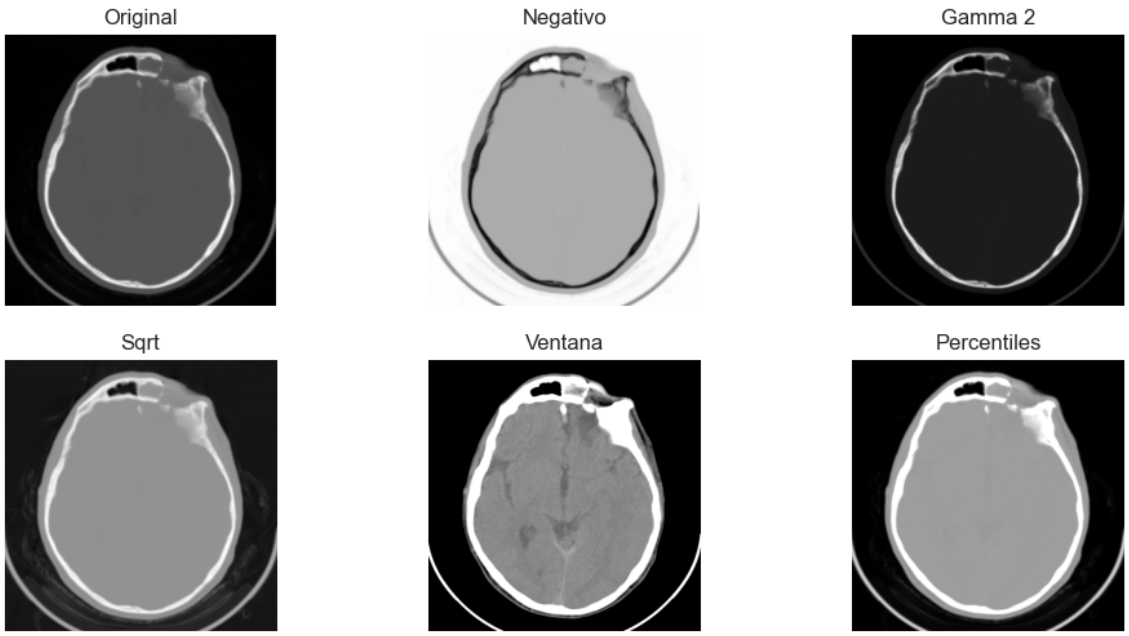

Figura 14: Figura resumen.

Para concluir con el ejercicio, la ventana lineal no ha vuelto a ser programada ya que se resolvió con anterioridad en el ejercicio 3.

6. Ejercicio 5

6.1. Enunciado

Guardar todos los resultados de imágenes finales obtenidas para la memoria de prácticas. Use la función imwrite(...).

1. En formato png, con el mismo tipo de datos que la imagen original.
2. En formato jpg, tras reescalar la imagen al tipo uint8 (256 niveles de gris).
3. Adjuntar un fichero comprimido a la memoria con los ficheros de imagen obtenidos.

6.2. Guardado PNG y JPG

La imagen original contiene 12 bits por píxel. Todas las imágenes del documento son también de 12 bpp, ya que todos los algoritmos se adaptan al tipo de dato con el que se abrió la imagen originalmente.

En en el entorno usado para programar los algoritmos (Data Spell), para guardar una imagen en formato png solo hace falta hacer clic derecho y guardar la imagen.

Sin embargo, para guardar las imágenes en formato jpg con 8 bpp, es necesario abrir la imagen original con ese tipo de dato y aplicar todos los algoritmos a esta nueva imagen. Para finalizar, han sido guardadas usando la función *plt.savefig()* e indicando jpg como formato. Estas funciones ya fueron descritas al principio del documento (Ver Listing 3).

7. Licencia y repositorio.

El código completo se encuentra en :mi repositorio de Github.
Licencia: MIT License