Manejo Básico De Imágenes

Julio Pérez
Estudiante, Facultad de Ingeniería
Ciudad Universitaria, Ciudad de México, México

Mauricio Bautista
Estudiante, Facultad de Ingeniería
Ciudad Universitaria Ciudad de México, México

Alisson Veloz
Estudiante, Facultad de Ingeniería
Ciudad Universitaria Ciudad de México, México

***Resumen*—En este documento se habla sobre los más de 100 formatos de imagen que hay, como JPEG, PNG, DICOM, NIfTI, TIFF, ics, ims, entre otros. Cada uno tiene usos específicos, como fotografía, imagenología médica o imágenes de microscopio. Existen módulos y paquetes de Python que pueden leerlos y extraer datos útiles, como tamaño, tipo y modo. Algunos formatos comunes son: JPEG, TIFF y DICOM.**

Palabras clave*—*JPEG, TIFF, DICOM.

I. Introducción

En esta practica observaremos las maneras básicas para desplegar y manipular imágenes en el lenguaje de programación Python para entender cómo se pueden manipular.

Principalmente se harán manipulaciones básicas de las imágenes como lo seria abrir las imágenes con diferentes bibliotecas, cambiar su espacio de color, representar las imágenes por un solo canal, recortar las imágenes y poder reproducir imágenes como videos.

II. Ejercicio 1

El ejercicio uno pide desplegar una imagen con las bibliotecas Matplotlib, OpenCV, Scikit-Image y PIL y para ello primero importamos cada una.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import time
import cv2
from skimage import io
from PIL import Image
```

Fig.1 Bibliotecas usadas en ejercicio 1.

Después desplegamos cada imagen en el orden en el que nos dan las bibliotecas empezando con Matplotlib, para ello usamos la variable imagen con el nombre de la imagen que usamos y con la función imread la guardamos en una variable que posteriormente mostraremos con la función *show* con el nombre de su biblioteca que después de pasar el flujo hará que se cierre la imagen.

```
img = mpimg.imread(Imagen)
plt.imshow(img)
plt.title('Matplotlib')
plt.show()
time.sleep(2)
plt.close()
```

Fig.2 Código de la biblioteca Matplotlib para desplegar imagen.

Para Opencv usaremos su función imread para interpretar la imagen que usamos y la desplegaremos con su función imshow que nos despliega la imagen y con el nombre de la ventana con el nombre de la biblioteca que desplegaremos 2 segundos antes de cerrarlas.

```
img = cv2.imread(Imagen)
cv2.imshow('OpenCV -
Imagen', img)
cv2.waitKey(2000)  # 2000
milisegundos = 2 segundos
cv2.destroyAllWindows()
```

Fig.3 Código de la biblioteca Opencv para desplegar imagen.

Con Sckit-Image haremos un procedimiento similar a la biblioteca Matplotlib al tener las mismas funciones para desplegar imágenes.

```
img = io.imread(Imagen)
io.imshow(img)
plt.title('Scikit-Image')
io.show()
time.sleep(2)
io.show()
```

Fig.4 Código de la biblioteca Sckit-Image para desplegar imagen.

Por último, tenemos la biblioteca Pil que al igual que las otras usara una función en este caso open para guardar la información de la imagen y la desplegara con la función *show* con el nombre de la biblioteca y la desplegaremos 2 segundos.

```
img = Image.open(Imagen)
img.show()
img.title = 'PIL (Pillow)'
time.sleep(2)
img.close()
```

Fig.5 Código de la biblioteca PIL para desplegar imagen.

Resultado obtenido

Fig.6 Imagen .jpg desplegada con Matplotlib.

III. EJERCICIO 2

El ejercicio 2 nos pide desplegar la información de la imagen que es su extensión, tamaño y su tipo de dato, para esto nos apoyaremos de las bibliotecas Opencv, numpy tkinter, os, pydicom, pylibjpeg y matplotlib para poder ver cualquier formato de la imagen según lo que se necesite.

```
import cv2
import numpy as np
from tkinter import Tk,
filedialog
import os
import pydicom
import pylibjpeg
import matplotlib.pyplot as
plt
```

Fig.7 Bibliotecas usadas en el ejercicio 2.

Empezaremos con las funciones importadas de tkinter para poder seleccionar cualquier archivo del explorador y anexaremos las extensiones de imágenes que usaremos además de agregar nuestra condicional para ver lo que tenemos no es una imagen.

```
root = Tk()
root.withdraw()
Imagen =
filedialog.askopenfilename(title='Seleccion
ar archivo de imagen', filetypes=
[('Imagen', ['*.jpg', '*.raw', '*.tif',
'*.dicom', '*.dcm'])])
root.destroy()

if not Imagen:
    print("No se seleccionó ningún
archivo.")
```

Fig.8 Selector de archivos.

Una vez que se a detectado que el archivo es una imagen con la biblioteca OS obtendremos su extensión y su tamaño para imprimirlos en pantalla.

```
else:
    # Obtener información sobre el archivo
    _, ext = os.path.splitext(Imagen)
    file_size = os.path.getsize(Imagen)

    # Mostrar información
    print(f"Nombre del archivo:
{os.path.basename(Imagen)}")
    print(f"Extensión del archivo: {ext}")
    print(f"Tamaño del archivo: {file_size}
bytes")o se seleccionó ningún archivo.")
```

Fig.9 Código para obtener los datos base del archivo.

Después tendremos una cadena de if else para obtener la información de los archivos según el formato siendo uno para dicom en el que nos apoyaremos de pydicom para obtener su resolución, después el otro será para raw donde nos apoyaremos me numpy y matpltlib para obtener sus datos y en caso de no ser ninguno de los dos obtendremos la resolución con Opencv.

```
if ext.lower() in ['.dcm']:
        # Leer el archivo DICOM
        ds = pydicom.dcmread(Imagen)

        # Obtener información de resolución
        if 'PixelSpacing' in ds:
            print(f"Pixel Spacing:
{ds.PixelSpacing}")

        if 'Rows' in ds and 'Columns' in
ds:
            print(f"Dimensiones de la
imagen DICOM: {ds.Rows} x {ds.Columns}")

        # Otros atributos relacionados con
la resolución pueden imprimirse según sea
necesario

        # Mostrar la imagen (si es
relevante para tu caso)
        if 'PixelData' in ds:
            imagen = ds.pixel_array
```

Fig.10 If para interpretar datos de archivos DICOM.

```
elif ext.lower() in ['.raw']:
        # Intentar cargar la imagen RAW con
numpy y Matplotlib
        try:
            # Ajustar las dimensiones según
tus necesidades (800 x 600)
            ancho, alto = 800, 600
            dtype = np.uint8  # o np.uint16
si la imagen es de 16 bits

            with open(Imagen, 'rb') as f:
                img_raw = np.fromfile(f,
dtype=dtype)
                img =
img_raw.reshape((ancho, alto))  # o (ancho,
alto) dependiendo de la orientación

                # Mostrar la imagen en
escala de grises con Matplotlib
                plt.imshow(img,
cmap='gray')  # cmap='jet' para una
representación en color falsa
                print("Resolucion: ",ancho,
alto)
                plt.show()

        except Exception as e:
            print(f"Error al cargar y
mostrar la imagen RAW: {e}")
```

Fig.11 If para interpretar datos de archivos RAW.

```
    else:
        # Mostrar imagen con OpenCV y
obtener la resolución
        img = cv2.imread(Imagen)
        height, width, _ = img.shape
        print(f"Resolución de la imagen:
{width} x {height}")

        cv2.imshow('Imagen', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

Fig.12 If para obtener y desplegar imágenes en un formato diferente a raw y DICOM.

Resultado obtenido

Fig.13 Imagen de Lena con sus datos correspondientes.

IV. EJERCICIO 3

El ejercicio pide que de las imágenes “lena_color_512.tif”, “peppers_color.tif” cambiar sus espacios de color usando primero Opencv y luego Sckit-image, nosotros nos apoyamos de otras bibliotecas para poder elegir las imágenes con el mismo código de la figura número 7.

```
import cv2
import numpy as np
from tkinter import Tk, filedialog
import matplotlib.pyplot as plt
from skimage import io, color
```

Fig.14 Bibliotecas usadas en el ejercicio 3.

Ya con la imagen elegida con el código de la figura 7 lo leeremos con la función de leer de Opencv y en 3 variable diferentes guardaremos la figura en los diferentes espacios de color aplicándoles la función cvtcolor que de parámetros tienen la imagen y el espacio de color al que se le cambia.

```
image = cv2.imread(nombre_archivo)

image_gray = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)

image_yuv = cv2.cvtColor(image,
cv2.COLOR_BGR2YUV)

image_hsv = cv2.cvtColor(image,
cv2.COLOR_BGR2HSV)

cv2.imshow('Escala de Grises cv2',
image_gray)

cv2.imshow('YUV cv2', image_yuv)

cv2.imshow('HSV cv2', image_hsv)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Fig.15 Código para cambiar el espacio de color para Opencv.

Después de cerrar las imágenes de opencv leerá la imagen con sckit-image y para cambiar los espacios de color analizará las dimensiones de la imagen para ver si es necesario convertirla a escalas de grises o si no ignorar su canal alfa para convertirlo con la función rgb2gray debido a que nos dio errores al usar la función sola y a los demás espacios usaremos *rgb2yuv* para yuv y rgb2hsv para hsv ignorando el canal alfa para convertirlas.

```python
image = io.imread(nombre_archivo)

# Verificar la forma de la imagen para asegurarse de que sea compatible con rgb2gray
if image.ndim == 2:  # La imagen ya está en escala de grises
    image_gray = image
elif image.ndim == 3 and image.shape[2] in [3, 4]:  # La imagen es RGB o RGBA
    # Convertir a escala de grises
    image_gray = color.rgb2gray(image[..., :3])  # Ignorar el canal alfa si existe
else:
    raise ValueError(f"La imagen tiene una forma inesperada: {image.shape}")

# Convertir de RGB a YUV
image_yuv = color.rgb2yuv(image[..., :3])

# Convertir de RGB a HSV
image_hsv = color.rgb2hsv(image[..., :3])
```

Fig.16 Código para cambiar el espacio para Sckit-Image.

Finalmente, no apoyaremos de Matplotlib para desplegar las 3 imágenes en diferentes espacios juntas con sus respectivas etiquetas de cada una.

```python
# Crear una figura con tres subgráficos
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

# Mostrar la imagen en escala de grises en el primer subgráfico
axs[0].imshow(image_gray, cmap='gray')
axs[0].set_title('Escala de Grises')

# Mostrar la imagen en espacio de color YUV en el segundo subgráfico
axs[1].imshow(image_yuv)
axs[1].set_title('YUV')

# Mostrar la imagen en espacio de color HSV en el tercer subgráfico
axs[2].imshow(image_hsv)
axs[2].set_title('HSV')

# Ajustar el diseño y mostrar la figura
plt.tight_layout()
plt.show()
```

Fig.17 Código para cambiar imprimir los resultados de Sckit-Image con Matplotlib.

Resultado obtenido

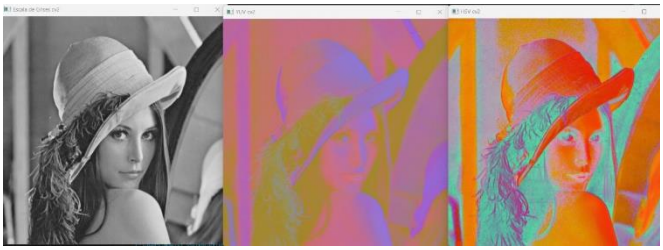

Fig.18 Imagen de Lena en formatos CV2, YUV y HSV.

V. EJERCICIO 4

El ejercicio 4 nos pide desplegar las paletas de colores de una imagen RGB, por lo que usamos las bibliotecas de sckit-image y matplitlib para obtener la información de la imagen y numpy y tkinter para poder elegir el archivo a voluntad.

```python
import matplotlib.pyplot as plt
import numpy as np
from skimage import io, color
from tkinter import Tk, filedialog
```

Fig.19 Bibliotecas usadas en el ejercicio 4.

Para la lectura del archivo nos apoyamos del código de la figura 7, una vez recolectada la imagen la leemos con la biblioteca sckit-image y analizamos si la imagen es RGB, en caso contrario informa que no es compatible

```python
# Cargar la imagen
    imagen = io.imread(nombre_archivo)

    # Verificar si la imagen es en color (3 canales) o en escala de grises (1 canal)
    if imagen.ndim == 2:  # Imagen en escala de grises
        print("La imagen es en escala de grises.")
        paleta_rgb = None  # No hay paleta para escala de grises
    elif imagen.ndim == 3 and imagen.shape[2] == 3:  # Imagen en color (RGB)
        print("La imagen es en color (RGB).")
        paleta_rgb = imagen
    else:
        raise ValueError(f"Formato de imagen no compatible: {imagen.shape}")
```

Fig.20 Verificar si la imagen es RGB con sckit-image

Ya confirmado que el archivo es RGB definimos los subgrafos para cada color con matplotlib pasando la imagen en un for para que en cada espacio solo se vea un color del RGB con su titulo y mostrarla con la función show.

```python
if paleta_rgb is not None:
        canales_rgb = ['Rojo', 'Verde', 'Azul']

        fig, axs = plt.subplots(1, 4, figsize=(15, 4))

        # Visualizar la imagen completa
        axs[0].imshow(paleta_rgb)
        axs[0].set_title('Imagen completa')
        axs[0].axis('off')

        # Visualizar cada canal RGB por separado con mapa de colores 'gray'
```

Fig.21 definimos con Matplotlib el espacio de cada color.

```
for i in range(3):
            canal = paleta_rgb[..., i]
            axs[i+1].imshow(canal, cmap='gray')  #
Utilizar 'gray' para canales individuales
            axs[i+1].set_title(f'Canal
{canales_rgb[i]}')
            axs[i+1].axis('off')

        # Mostrar la barra de colores a la derecha
        axs[-1].axis('off')
        cbar =
plt.colorbar(axs[1].imshow(paleta_rgb[..., 0],
cmap='gray'), ax=axs[-1])
        cbar.set_label('Valor')

        # Ajustar el diseño y mostrar la figura
        plt.tight_layout()
        plt.show()
```

Fig.22 Usamos el for para poner en cada espacio la imagen con cada color y lo despliega

Resultado obtenido

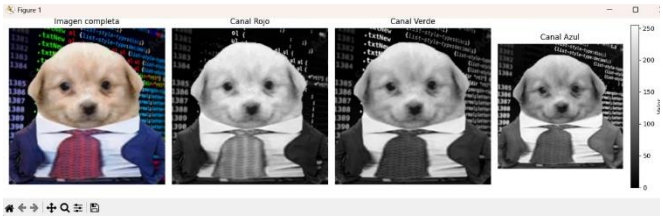

Fig.23 Imagen “perrito” presentado en diferentes canales.

VI. EJERCICIO 5

El ejercicio pide que a una imagen cualquiera sea convertida a escala de grises y luego se le haga una declinación, haciendo que este a la mitad de su tamaño y en grupos de 4 pixeles, por lo que usaremos las bibliotecas PIL y tkinter para acceder a las imágenes y la biblioteca numpy para manipular la imagen y con las funciones matplotlib y sckit-image aunque no se usan.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color
from tkinter import Tk, filedialog
```

Fig.24 Bibliotecas ejercicio 5

Usaremos tkinter para poder seleccionar los archivos con libertad que respeten el formato de imagen para podamos pasar el producto entre nuestras dos funciones llamadas cargar y preparar imagen y decimar imagen.

```
#Ruta de la imagen a procesar
ruta_imagen = filedialog.askopenfilename(
    title='Seleccionar archivo de imagen',
    filetypes=[
        ('Archivos de imagen',
'*.tif;*.png;*.jpg;*.raw'),
        ('Todos los archivos', '*.*')
    ]
)
```

Fig.25 Código para leer la imagen

```
def cargar_y_preparar_imagen(ruta_imagen):
    # Cargar la imagen
    imagen = Image.open(ruta_imagen)
    # Convertir a escala de grises
    imagen_gris = imagen.convert("L")
    # Asegurarse que es cuadrada recortando el lado
más largo
    lado_corto = min(imagen_gris.size)
    imagen_cuadrada = imagen_gris.crop((0, 0,
lado_corto, lado_corto))
    return imagen_cuadrada
```

Fig.26 Función para interpretar y convertir la imagen en escalas de grises

```
def decimar_imagen(imagen):
    # Convertir la imagen a una matriz numpy
    arr = np.array(imagen)
    # Reducir a la mitad del tamaño mediante el
promedio de grupos de 4 pixeles
    arr_decimada = arr.reshape((arr.shape[0]//2, 2,
arr.shape[1]//2, 2)).mean(axis=(1, 3))
    # Convertir de nuevo a una imagen PIL
    imagen_decimada =
Image.fromarray(arr_decimada.astype(np.uint8))
    return imagen_decimada
```

Fig.27 Función para escalar y promediar la imagen

Ya con el archivo leído lo cargamos como imagen con la biblioteca pil y la convertimos a escala de grises con la misma biblioteca y busca el lado corto para cortar la imagen en caso de que no sea cuadrada.

En la siguiente función con la función numpy convertimos la imagen ya condicionada a un arreglo para poder estructurarla a grupos de 4 pixeles y con la función mean reducir a la mitad el arreglo con el promedio de los pixeles, ya reorganizado el arreglo lo convertimos al formato de 8 bits por pixel y con fromarray le devolvemos el formato de imagen.

Ya con la imagen tratada imprimimos la resolución de ambas para ver que esta a la mitad y las mostramos con la función *show* además de guardarla.

```python
#Cargar, preparar y decimar la imagen
imagen_preparada =
cargar_y_preparar_imagen(ruta_imagen)
imagen_decimada = decimar_imagen(imagen_preparada)

#Imprimir las resoluciones
print(f"Resolución de la imagen preparada:
{imagen_preparada.size} (ancho x alto)")
print(f"Resolución de la imagen decimada:
{imagen_decimada.size} (ancho x alto)")

#Mostrar la imagen original y la imagen decimada
imagen_preparada.show()
imagen_decimada.show()

#Guardar la imagen decimada si lo deseas
imagen_decimada.save('imagen_decimada.jpg')
```

Fig.28 Código para reinterpretar la imagen y mostrar sus datos

Resultado obtenido

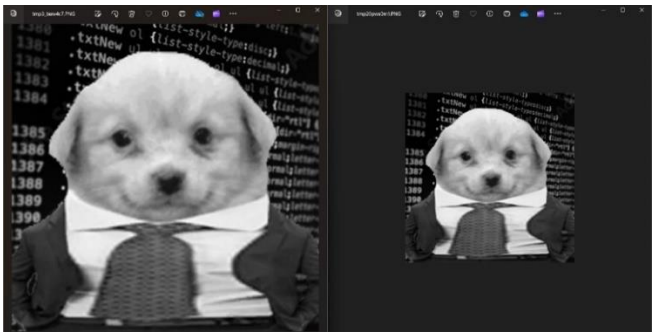

Fig.29 Imagen original vs imagen reducida a la mitad.

VII. EJERCICIO 6

El ejercicio 6 pide que se convierta la imagen peppers_color.tif a escala de grises, además de elegir un pimiento verde y cortar la imagen para que solo se vea ese pimiento para posteriormente guardarlo con el formato JPG.

Para este ejercicio usamos las bibliotecas sckit-image y para leer la imagen y pil para reinterpretarla cuando la cortemos no usamos opencv ni numpy al no tener que desplegarla.

```python
from skimage import io
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
```

Fig.30 Bibliotecas usadas en el ejercicio 6.

Primero definimos una variable con la ruta, en este caso su nombre y lo leemos con la biblioteca sckit-image, para que en otra variable obtengamos solo la escala de grises y apoyados por una variable con las coordenadas y Photoshop para ver exactamente donde recortar.

```python
image_path = 'peppers_color.tif'

image_skimage = io.imread(image_path)

gray_image_from_first_channel = image_skimage[:, :,
0]

crop_coordinates = (180, 220, 420, 470)
```

Fig. 31 Código para leer la ruta y ponerla a escala de grises

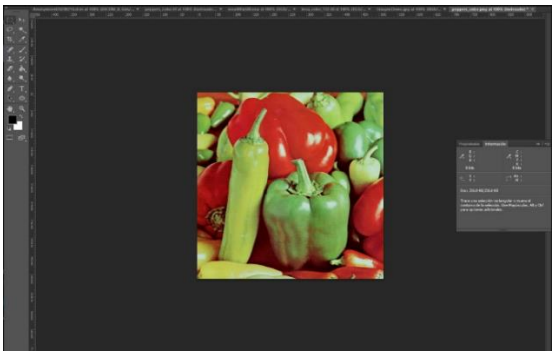

Fig.32 Uso de Photoshop para definir el corte (mediante coordenadas).

Ya con las coordenadas las seleccionamos de la imagen en grises y la guardamos en otra, para que lo cortado lo reagrupamos con la biblioteca pil, definimos su nombre y lo guardamos como JPG junto con una confirmación de que se guardó.

```python
cropped_image =
gray_image_from_first_channel[crop_coordinates[1]:cro
p_coordinates[3],
crop_coordinates[0]:crop_coordinates[2]]


cropped_image_pil = Image.fromarray(cropped_image)


cropped_image_path = 'cropped_pepper.jpg'


cropped_image_pil.save(cropped_image_path, 'JPEG')

print(f'Imagen recortada guardada como
{cropped_image_path}')
```

Fig.33 Código para convertir de regreso la imagen cortada por medio de pil y su guardado en el formato JPG

Resultado obtenido

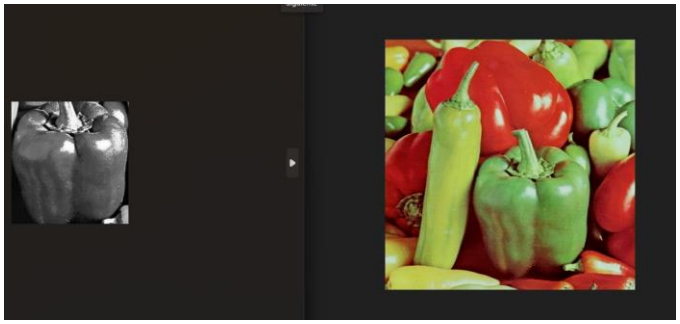

Fig.34 Obtención de imagen en escala de grises mediante coordenadas.

VIII. EJERCICIO 7

El ejercicio pide obtener un formato crudo (RAW), es decir, que no contenga ningún tipo de codificación, ni modificación. Se busca conservar los archivos tal cual fueron capturados por la cámara. Este formato es importante porque contiene datos como exposición, balance de blancos entre otros, lo que permite un control mayor en el procesamiento de la imagen.

Tomando como referencia, lo previamente explicado en el ejercicio 2, buscamos resaltar la carga de la imagen, y del cómo según lo que deseemos, visualizaremos las dimensiones deseadas, así como mostrar la imagen en su escala de grises.

```
elif ext.lower() in ['.raw']:
    # Intentar cargar la imagen RAW con numpy y Matplotlib
    try:
        # Ajustar las dimensiones según tus necesidades (800 x 600)
        ancho, alto = 800, 600
        dtype = np.uint8  # o np.uint16 si la imagen es de 16 bits

        with open(Imagen, 'rb') as f:
            img_raw = np.fromfile(f, dtype=dtype)
            img = img_raw.reshape((ancho, alto))  # o (ancho, alto) dependiendo de la orientación

            # Mostrar la imagen en escala de grises con Matplotlib
            plt.imshow(img, cmap='gray')  # cmap='jet' para una representación en color falsa
            print("Resolucion: ",ancho, alto)
            plt.show()
```

Fig.35 Modificación de la imagen (ancho, altura) y escala de grises.

Resultado obtenido

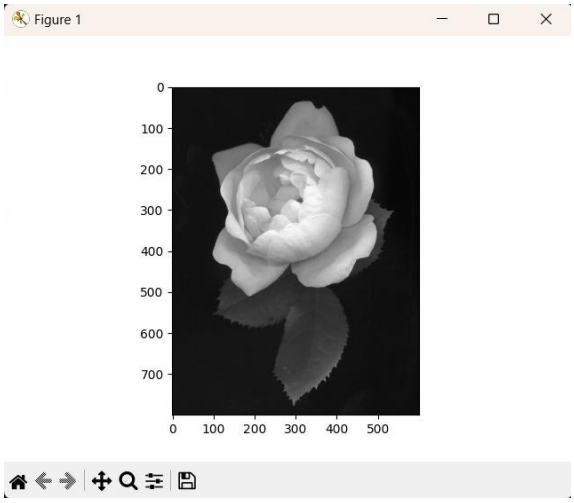

Fig.36 Imagen RAW desplegada.

IX. EJERCICIO 8

El formato .avi es un contenedor de datos de audio y video comprimidos, el cual es muy útil ya que no tiene pérdida de calidad, haciéndolo ideal para el procesamiento de imágenes y videos.

Para la lectura del video .avi utilizamos la biblioteca de OpenCV y verificamos si se pudo abrir el archivo correctamente mediante un if. Una vez realizada la comprobación, lee y muestra cada frame, además permitir ajustar el tiempo de pausa para una mejor visualización.

```
# Leer y mostrar cada frame
while True:
    ret, frame = cap.read()

# Mostrar el frame
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.pause(0.03)  # Ajusta el tiempo de pausa según tu preferencia
    plt.draw()
```

Fig.37 Lectura de frame por frame del video .avi.

Resultado obtenido

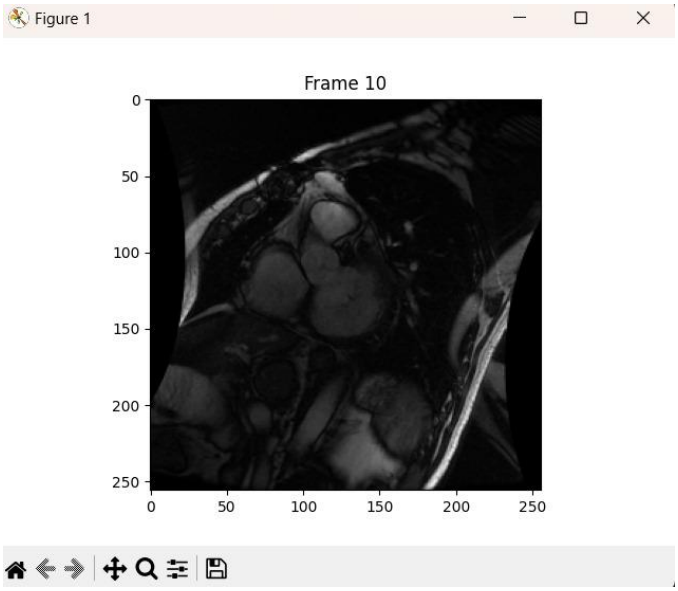

Fig.38 Video .avi presentado frame por frame.

X. EJERCICIO 9

Imágenes DICOM

DICOM significa “Imagen y Comunicaciones Digitales en Medicina” el cual es un estándar para la transmisión, almacenamiento, impresión y visualización de imágenes médicas y datos relacionados. Se suele usar para visualizar Tomografías Computarizadas (CT) e Imágenes por Resonancia Magnética (MRI).

Para la lectura de imágenes de la carpeta DICOM, se utilizaron diferentes funciones y bibliotecas de Python, las cuales nos permitieron trabajar específicamente con estos archivos.

```
import os       #manipulacion de rutas de archivos y directorios
import pydicom    #Biblioteca de python para trabajar con archivo DICOM
import numpy as np
import imageio   #Biblioteca para leer y escribir multiples formatos de
imagenes
from tkinter import Tk, filedialog  #Biblioteca para crear interfaces graficas
import matplotlib.pyplot as plt
```

Fig.39 Declaración de las librerías/bibliotecas usadas.

Posteriormente, se crea una ventana Tkinter, se comprueba la ruta de la carpeta y comienza la lectura de los archivos, es necesario crear un arreglo para ordenarlos.

```
# Ordenar los archivos por nombre
    dcm_files.sort()

    # Crear una lista para almacenar las imágenes DICOM
    dicom_images = []

    # Leer cada archivo DICOM y agregarlo a la lista
    for dcm_file in dcm_files:
        dcm_path = os.path.join(folder_path, dcm_file)
        ds = pydicom.dcmread(dcm_path)
        dicom_images.append(ds.pixel_array)
```

Fig.40 Creación del array de imágenes DICOM.

Finalmente, se normalizan los pixeles y se guarda como GIF, un formato que permite tener 'movimiento' a la imagen; Este formato se muestra usando librerías de Matplotlib.

```
Guardar el gif
  gif_path = os.path.join(folder_path, 'output.gif')
  imageio.mimsave(gif_path, dicom_array.transpose(2, 0, 1), duration=0.1)

  # Mostrar el gif utilizando matplotlib
  with imageio.get_reader(gif_path) as reader:
      for i, frame in enumerate(reader):
          plt.imshow(frame, cmap='gray')
          plt.title(f'Frame {i + 1}')
          plt.pause(0.001)  # Tiempo de pausa más corto para una reproducción
          plt.draw()

  plt.show()
```

Fig.41 Reproducción de la imagen GIF.

Resultado obtenido

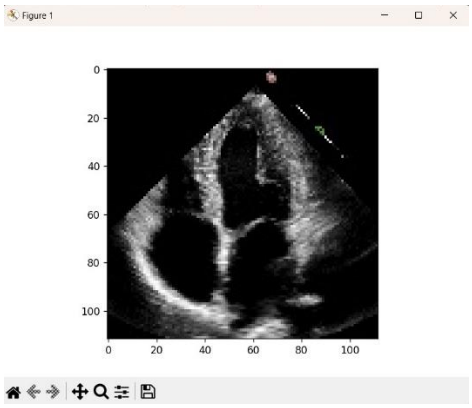

Fig.42 Reproducción de la imagen GIF.

XI. Conclusión

En dicha práctica pudimos aprender las distintas formas de manipulación básica de imágenes en diferentes formatos. Se aprendió el uso de las librerías necesarias para la apertura, obtención de información, manipulación de estructura y reproducción de archivos DICOM.

Sin duda, este conocimiento nos ayudará en futuras prácticas para poder trabajar con un mayor número de muestras e implementar según el formato, las herramientas necesarias para la clasificación de imágenes y la limpieza de estas.

XII. Referencias

[1] DICOM. (s/f). *About DICOM- overview*. DICOM. Recuperado el 19 de febrero de 2024, de https://www.dicomstandard.org/about-home

[2] Fernandez, R. (2019, mayo 27). *10 Librerías para manipular imágenes en Python*. Cursos de Programación de Python. https://unipython.com/10-librerias-para-manipular-imagenes-en-python/

[3] *¿Qué es el formato DICOM? Las claves del estándar en imágenes médicas*. (2021, julio 22). Clinic Cloud. https://clinic-cloud.com/blog/formato-dicom-que-es-estandar-imagenes-medicas/

[4] Siemens. (s/f). *Imágenes DICOM*. Recuperado el 19 de febrero de 2024, de https://www.siemens-healthineers.com/mx/services/it-standards/dicom