Proyecto 2 - Limpieza de imágenes

- Sebastian Aristondo 20880
- Diego Franco 20240
- Manuel Archila 161250
- Juan Diego Avila 20090
- Daniel Gonzalez Carrillo 20293


```
import pandas as pd
import numpy as np
import pydicom
from pydicom.pixel_data_handlers.util import apply_voi_lut
import os
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from scipy import ndimage
import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.models as models
from torch.utils.data import DataLoader, TensorDataset
import torch.optim as optim
import cv2
from PIL import Image
import zipfile
import io
```

```
data = pd.read_csv('meta_train_with_vertebrae.csv', encoding='utf-8')
```

```
data.head()
```

	StudyInstanceUID	Slice	ImageHeight	ImageWidth	SliceThickness	ImagePositionPatient_x
0	1.2.826.0.1.3680043.10001	1	512	512	0.625	-52.308
1	1.2.826.0.1.3680043.10001	2	512	512	0.625	-52.308
2	1.2.826.0.1.3680043.10001	3	512	512	0.625	-52.308
3	1.2.826.0.1.3680043.10001	4	512	512	0.625	-52.308
4	1.2.826.0.1.3680043.10001	5	512	512	0.625	-52.308

Creación de volúmenes para 3DCNN

```
def extract_number(filename):
    # Extraer el número de la cadena (por ejemplo, "10" de "10.dcm")
    return int(filename.split('.')[0])
```

```
def read_images(path):
    # Obtener la lista de nombres de archivos de imágenes
    image_files = os.listdir(path)
    image_files.sort(key=extract_number)  # Asegura un orden adecuado

    # Inicializar una lista para almacenar los volúmenes 3D
    volumes = []

    # Leer y apilar las imágenes en un volumen 3D
    for image_file in image_files:
        image_path = os.path.join(path, image_file)
        dicom_data = pydicom.dcmread(image_path)
        image = apply_voi_lut(dicom_data.pixel_array, dicom_data)
        volumes.append(image)

    # Convertir la lista de volúmenes a un arreglo NumPy 3D
    volumes_array = np.stack(volumes, axis=-1)

    # Normalizar los valores de píxeles en el rango [0, 1]
    # Esto puede variar dependiendo de la información específica de las imágenes DICOM
    volumes_array = (volumes_array - np.min(volumes_array)) / (np.max(volumes_array) - np.min(volu
    return volumes_array
```

```
def normalize(volume):
    """Normalize the volume"""
    min = -1000
    max = 400
    volume[volume < min] = min
    volume[volume > max] = max
    volume = (volume - min) / (max - min)
    volume = volume.astype("float32")
    return volume
```

```
def resize_volume(img):
    """Resize across z-axis"""
    # Set the desired depth
    desired_depth = 300
    desired_width = 256
    desired_height = 256
    # Get current depth
```

```
    current_depth = img.shape[-1]
    current_width = img.shape[0]
    current_height = img.shape[1]
    # Compute depth factor
    depth = current_depth / desired_depth
    width = current_width / desired_width
    height = current_height / desired_height
    depth_factor = 1 / depth
    width_factor = 1 / width
    height_factor = 1 / height
    # Rotate
    img = ndimage.rotate(img, 90, reshape=False)
    # Resize across z-axis
    img = ndimage.zoom(img, (width_factor, height_factor, depth_factor), order=1)
    return img
```

```
def process_scan(path):
    """Read and resize volume"""
    # Read scan
    volume = read_images(path)
    # Normalize
    volume = normalize(volume)
    # Resize width, height and depth
    volume = resize_volume(volume)
    return volume
```

```
root = "train_images"
save_dir = "volumes"
patients = os.listdir(root)
count = 0
for patient in patients:
    patient_dir = os.path.join(root, patient)
    volumes = process_scan(patient_dir)
    output_file = "volumes/" + patient + ".npy"
    np.save(output_file, volumes)
    #if count >= 10:
    #    break
    #count += 1
```

```
C:\Users\Daniel\AppData\Local\Temp\ipykernel_29716\1781923139.py:21: RuntimeWarning: invalid
value encountered in divide
  volumes_array = (volumes_array - np.min(volumes_array)) / (np.max(volumes_array) -
np.min(volumes_array))

KeyboardInterrupt:
```

Se realizó una lectura de todas las imágenes en formato dicom que se pudieron descargar del dataset original. Luego, se leyeron y se convirtió en un volumen de numpy para cada paciente. Estos volúmenes se guardaron en un archivo .npy para su posterior uso en la red neuronal.

Dicom a JPG

```
import os
import pydicom
import cv2
import numpy as np
from PIL import Image
import shutil

# Ruta de la carpeta con archivos DICOM
carpeta_dicom_base = "train_images"
carpeta_jpg_base = "imagenes_train"

# Obtener una lista de carpetas en la carpeta base
carpetas = [f for f in os.listdir(carpeta_dicom_base) if os.path.isdir(os.path.join(carpeta_dicom_

# Iterar sobre cada carpeta y procesar los archivos DICOM
for carpeta in carpetas:
    carpeta_dicom = os.path.join(carpeta_dicom_base, carpeta)
    carpeta_jpg = os.path.join(carpeta_jpg_base, carpeta)

    # Obtener lista de archivos DICOM en la carpeta
    archivos_dicom = [f for f in os.listdir(carpeta_dicom) if f.endswith('.dcm')]

    # Crear un objeto en memoria para almacenar el archivo ZIP
    zip_memory = io.BytesIO()
    with zipfile.ZipFile(zip_memory, "a", zipfile.ZIP_DEFLATED, False) as zipf:
        # Iterar sobre los archivos DICOM y convertirlos a JPG y agregarlos al archivo ZIP
        for archivo_dicom in archivos_dicom:
            ruta_archivo_dicom = os.path.join(carpeta_dicom, archivo_dicom)
            imagen_dicom = pydicom.dcmread(ruta_archivo_dicom)

            gray_image = imagen_dicom.pixel_array

            # Normalizar la imagen en escala de grises
            normalized_gray_image = gray_image / np.max(gray_image)

            # Crear una imagen RGB a partir de la imagen en escala de grises
            # Asignar el valor de intensidad de grises a los canales R, G y B
            rgb_image = cv2.cvtColor((normalized_gray_image * 255).astype(np.uint8), cv2.COLOR_GR

            # Convertir la imagen en formato JPG en memoria
            img_memory = io.BytesIO()
            Image.fromarray(rgb_image).save(img_memory, format='JPEG')
            img_memory.seek(0)

            # Agregar la imagen al archivo ZIP
            nombre_archivo_jpg = os.path.join(carpeta, archivo_dicom.replace('.dcm', '.jpg'))
            zipf.writestr(nombre_archivo_jpg, img_memory.read())
```

```
    # Guardar el archivo ZIP en disco
    with open(f"{carpeta_jpg}.zip", "wb") as zip_file:
        zip_file.write(zip_memory.getvalue())
```

```
ValueError: The length of the pixel data in the dataset (392484 bytes) doesn't match the expected length (524288 bytes). The dataset may be corrupted or there may be an issue with the pixel data handler.
```

Se convirtieron todas las imágenes de formato dicom a jpg, para poder utilizarlas en la red neuronal que fue un híbrido de EfficientNet y GRU.

Filtro de imagenes

```
path = 'imagenes_train'
image_files = os.listdir(path)
image_files = [x.split('.zip')[0] for x in image_files]

df = pd.read_csv('train.csv', encoding='utf-8')

condicion = df["StudyInstanceUID"].isin(image_files)

df_filta = df[condicion]

df_filta.to_csv('train_filtrado_images.csv', index=False)
```

Lo que se realizó acá fue un filtrado del dataset original de train, para contener solo los pacientes existentes. Esto se realizó tanto para los volúmenes, como para las imágenes en jpg.