Librerías Utilizadas y Elementos Más Relevantes

Librería `os` (`import os`)

Permite trabajar con rutas y directorios del sistema de manera segura y portable entre distintos sistemas operativos.

- `os.path.join(dir_1, dir_2, ..., dir_n)`: Une correctamente múltiples directorios evitando errores por concatenación manual de strings.
- `os.makedirs(dir, exist_ok=True)`: Crea una carpeta en la ruta especificada. Si ya existe y `exist_ok=True`, no genera error.
- `os.listdir(dir)`: Retorna una lista con los nombres de los elementos contenidos en el directorio.
- `os.path.dirname(dir)`: Retorna la ruta eliminando el último elemento del directorio.
- `os.path.abspath(__file__)`: Convierte la ruta del script actual en una ruta absoluta.
- `__file__`: Variable reservada que almacena la ruta del archivo en ejecución.
- `os.path.splitext(nombre)`: Retorna una tupla del tipo (nombre, extensión).

  Ejemplo:

  - `imagen.png` → (`imagen`, `.png`)
  - `archivo.tar.gz` → (`archivo.tar`, `.gz`)

Ruta Relativa: Depende del directorio desde donde se ejecuta el script.
Ruta Absoluta: Indica la ubicación completa del archivo desde la raíz del sistema.

Analogía:
Ruta Absoluta → "Vivo en país X, ciudad Y, calle Z número N".
Ruta Relativa → "Desde donde estás, camina dos cuadras a la derecha".

El uso de rutas absolutas permite independizar la ejecución del script respecto al directorio actual.

—

Librería `NumPy` (`import numpy as np`)

Permite trabajar con vectores, matrices y tensores, fundamentales para representar imágenes como arreglos numéricos.

- `np.full((dimensiones), elemento, dtype)`: Crea un arreglo relleno completamente con un valor específico.

- `np.array(obj)`: Crea o convierte un objeto a un arreglo de NumPy.
- `np.zeros((dimensiones))`: Crea un arreglo compuesto únicamente por ceros.
- `np.logical_and(a,b)`: Operación lógica AND elemento a elemento.
- `np.logical_or(a,b)`: Operación lógica OR elemento a elemento.

En imágenes RGB, el tercer parámetro representa las tres componentes de color.

—

Librería `OpenCV` (`import cv2`)

Permite manipulación avanzada de imágenes con alto rendimiento.

- `cv2.imread(filename, flag)`: Carga una imagen como arreglo NumPy. Las imágenes a color se leen en formato BGR.
- `cv2.fillPoly(image, points, color)`: Dibuja polígonos sobre una imagen, utilizado para construir máscaras desde anotaciones.

—

Librería `Pillow` (`from PIL import Image`)

Permite manipulación básica de imágenes.

- `Image.fromarray(array).save(dir)`: Guarda un arreglo NumPy como imagen.
- `Image.open(imagen)`: Abre una imagen como objeto PIL.
- `convert("RGB")`: Convierte la imagen a formato RGB.

Observación: Para guardar imágenes correctamente, el arreglo debe ser tipo `uint8` con valores en el rango [0,255].

—

Librería `json` (`import json`)

Permite trabajar con archivos en formato JSON.

- `json.load(file)`: Convierte el contenido del archivo JSON a un objeto de Python.

Es importante que el archivo JSON esté correctamente estructurado.

—

Librería `subprocess`

Permite ejecutar comandos del sistema desde Python.

- `subprocess.run()`: Ejecuta un comando en la terminal. En este proyecto se utilizó para abrir automáticamente LabelMe con rutas predefinidas de inicializacion de imágenes y guardado de los json.

—

Librería `sys` (`import sys`)

Permite interactuar con el entorno del intérprete.

- `sys.path.append(dir)`: Añade un directorio a la lista de rutas donde Python busca módulos.

—

Librería `Torch` (`import torch`)

Constituye el núcleo del modelo de segmentación.

- `DataLoader (torch.utils.data)`: Permite crear iteradores eficientes para cargar datos durante el entrenamiento.
- `torchvision.transforms`: Permite aplicar transformaciones como normalización, conversión a tensor y redimensionamiento.
- `deeplabv3_resnet50 (torchvision.models.segmentation)`: Arquitectura de segmentación semántica basada en DeepLabV3+ con backbone ResNet-50.