## Código Fuente del Modelo YOLO (versión 8, 10 y 11)

**Clonar el repositorio de YOLO e instalar las librerías Init y load de las librerías**

!git clone https://github.com/ultralytics/ultralytics.git  
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118  
!pip install ultralytics --upgrade

import torch  
print("GPU disponible :", torch.cuda.is\_available())  
if torch.cuda.is\_available():  
 print("Nom du GPU :", torch.cuda.get\_device\_name(0))

GPU disponible : True  
Nom du GPU : NVIDIA L4

# Función para crear el archivo YAML

Si ya existe un archivo dataset.yaml en tu configuración, no necesita ejecutar esta función.

import os  
from google.colab import drive  
  
def create\_yaml(ruta\_dataset, clases):  
 """  
 Crea el archivo YAML necesario para entrenar el modelo YOLO.  
 Define las rutas de entrenamiento y validación, el número de clases y sus nombres.  
 """  
 contenido\_yaml = f"""  
 train: {os.path.join(ruta\_dataset, 'train')}  
 val: {os.path.join(ruta\_dataset, 'valid')}  
  
 nc: {len(clases)} # Número de clases  
 names: {clases} # Lista de clases  
 """  
 ruta\_yaml = os.path.join(ruta\_dataset, 'dataset.yaml')  
 with open(ruta\_yaml, 'w') as archivo\_yaml:  
 archivo\_yaml.write(contenido\_yaml)  
 print(f"✅ Archivo dataset.yaml creado en: {ruta\_yaml}")  
 return ruta\_yaml  
# Número de clases  
drive.mount('/content/drive')  
  
# Ruta al directorio que contiene el dataset (padre de train, valid, test)  
dataset\_path = '/content/drive/My Drive/TFM DATASET YOLO'  
  
# Verifica si el directorio existe, si no, lo crea  
if not os.path.exists(dataset\_path):  
 os.makedirs(dataset\_path)  
  
# Definir las clases  
clases = ['Casco', 'Chaleco', 'No-Casco', 'No-Chaleco', 'Persona']  
  
# Ruta al archivo YAML  
ruta\_yaml = os.path.join(dataset\_path, 'dataset.yaml')  
  
# Verificar si el archivo YAML existe, si no, crearlo  
if not os.path.exists(ruta\_yaml):  
 print(f"Archivo {ruta\_yaml} no encontrado. Creando el archivo...")  
 ruta\_yaml = create\_yaml(dataset\_path, clases)  
else:  
 print(f"El archivo {ruta\_yaml} ya existe. No es necesario crearlo nuevamente.")

# Este codigo verifica la existencia de las carpetas 'train' y 'valid'

train\_path = os.path.join(dataset\_path, 'train')  
valid\_path = os.path.join(dataset\_path, 'valid')  
  
if not os.path.exists(train\_path):  
 print(f"La carpeta 'train' no existe: {train\_path}")  
else:  
 print(f"La carpeta 'train' contiene: {os.listdir(train\_path)}")  
  
if not os.path.exists(valid\_path):  
 print(f"La carpeta 'valid' no existe: {valid\_path}")  
else:  
 print(f"La carpeta 'valid' contiene: {os.listdir(valid\_path)}")

**Script de validación del dataset YOLO** Esta función permite que el script valide el dataset de imágenes y anotaciones, asegurándose de que las carpetas necesarias existan y que haya coherencia entre las imágenes y sus correspondientes anotaciones.

import os  
  
def validate\_dataset(dataset\_path):  
  
 # Lista para almacenar los problemas detectados  
 issues = []  
  
 # Validar las subcarpetas 'train' y 'valid'  
 for split in ['train', 'valid']:  
 # Rutas a las carpetas de imágenes y anotaciones  
 images\_dir = os.path.join(dataset\_path, split, 'images')  
 labels\_dir = os.path.join(dataset\_path, split, 'labels')  
  
 # Verificar si las carpetas existen  
 if not os.path.exists(images\_dir):  
 issues.append(f"La carpeta de imágenes no existe: {images\_dir}")  
 continue # Continuar con la siguiente carpeta  
 if not os.path.exists(labels\_dir):  
 issues.append(f"La carpeta de anotaciones no existe: {labels\_dir}")  
 continue  
  
 # Listar los archivos en las carpetas (solo extensiones válidas)  
 images = {os.path.splitext(f)[0] for f in os.listdir(images\_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))}  
 labels = {os.path.splitext(f)[0] for f in os.listdir(labels\_dir) if f.lower().endswith('.txt')}  
  
 # Detectar imágenes sin anotaciones y anotaciones sin imágenes  
 missing\_labels = images - labels # Imágenes sin archivo .txt  
 missing\_images = labels - images # Archivos .txt sin imagen  
  
 # Agregar problemas encontrados a la lista de errores  
 if missing\_labels:  
 issues.append(f"En '{split}': Faltan anotaciones para las imágenes: {', '.join(missing\_labels)}")  
 if missing\_images:  
 issues.append(f"En '{split}': Faltan imágenes para las anotaciones: {', '.join(missing\_images)}")  
  
 # Si se detectaron problemas, lanzar una excepción con detalles  
 if issues:  
 raise ValueError("\n".join(issues))  
  
 # Si todo está correcto, mostrar mensaje de éxito  
 print("Validación del dataset completada con éxito. No se detectaron problemas.")  
  
# Ruta al dataset principal  
dataset\_path = "/content/drive/My Drive/TFM DATASET YOLO"  
  
# Validar el dataset  
try:  
 validate\_dataset(dataset\_path)  
except ValueError as e:  
 print(f"Errores detectados durante la validación:\n{e}")

# Entrenamiento de un modelo YOLO

Este código entrena un modelo **YOLO** utilizando un archivo de configuración **YAML** y un dataset predefinido. Una vez completado el entrenamiento, verifica la presencia de los archivos de peso “best.pt” y “last.pt” generados durante el proceso. Permite configurar parámetros clave como el número de épocas y el tamaño de las imágenes para ajustar el entrenamiento según sea necesario.

Se asegura de que el archivo **YAML** exista antes de comenzar y notifica sobre cualquier archivo de peso que falte al finalizar el entrenamiento.

import os  
from ultralytics import YOLO  
from google.colab import files  
  
# Funcion para entrenar un modelo YOLO  
def entrenar\_yolo(version, variante, ruta\_yaml, epocas=30, tamano\_img=640, tamano\_batch=16):  
 versiones\_validas = ["8", "10", "11"]  
 variantes\_validas = ["n", "s", "m"]  
  
 # Validar la version de YOLO en input  
 if version not in versiones\_validas:  
 raise ValueError(f"Versión de YOLO no válida: {version}. Versiones soportadas: {versiones\_validas}.")  
  
 # Validar la variante de YOLO en input  
 if variante not in variantes\_validas:  
 raise ValueError(f"Variante de YOLO no válida: {variante}. Variantes soportadas: {variantes\_validas}.")  
  
 # Definir el nombre del proyecto y de la ejecución  
 nombre\_proyecto = f"YOLOv{version}" # Ejemplo: YOLOv8 o YOLOv10  
 nombre\_ejecucion = f"ejecucion\_{variante}"  
  
 archivo\_pesos = f"yolov{version}{variante}.pt" # Construir el nombre del archivo de pesos  
  
 print(f"\n Cargando el modelo YOLOv{version}{variante} desde {archivo\_pesos}")  
 modelo = YOLO(archivo\_pesos)  
  
 # Inicio del entrenamiento  
 print("\n=== Inicio del entrenamiento de YOLO ===")  
 modelo.train(  
 data=ruta\_yaml,  
 epochs=epocas,  
 imgsz=tamano\_img,  
 batch=tamano\_batch,  
 lr0=0.005, # Learning rate mas baja  
 weight\_decay=0.0005, # Regularizacion  
 momentum=0.9, # Inercia  
 optimizer='SGD', # Probar Adam o SGD  
 project=nombre\_proyecto,  
 name=nombre\_ejecucion,  
 save=True  
 )  
  
 # Capturar dinamicamente la ruta del directorio de resultados  
 directorio\_resultados = os.path.join(modelo.trainer.save\_dir, "weights")  
 print(f"\n Entrenamiento completado. Los resultados se han guardado en {directorio\_resultados}.")  
  
 # Descargar los archivos generados  
 descargar\_archivos\_especificos(directorio\_resultados)  
  
# Función para descargar archivos específicos  
def descargar\_archivos\_especificos(ruta\_directorio):  
 archivos\_especificos = ["last.pt", "best.pt"]  
 if os.path.exists(ruta\_directorio):  
 print(f" Directorio encontrado: {ruta\_directorio}")  
 for nombre\_archivo in archivos\_especificos:  
 ruta\_archivo = os.path.join(ruta\_directorio, nombre\_archivo)  
 if os.path.exists(ruta\_archivo):  
 print(f"Descargando el archivo: {nombre\_archivo}")  
 files.download(ruta\_archivo) # Descargar cada archivo  
 else:  
 print(f" Archivo no encontrado: {ruta\_archivo}")  
 else:  
 print(f" Directorio no encontrado: {ruta\_directorio}")  
  
# Boucle para configurar y entrenar el modelo  
while True:  
 try:  
 print("\n=== Configuración del Modelo YOLO ===")  
 version\_input = input("Ingrese la versión de YOLO (8, 10, 11): ").strip()  
 variante\_input = input("Ingrese la variante (n para Nano, s para Small): ").strip().lower()  
 # Se cambia la ruta para coresponder a su configuracion donde esta el fichero  
 ruta\_yaml = "/content/drive/My Drive/TFM DATASET YOLO/dataset.yaml"  
 if not os.path.exists(ruta\_yaml):  
 raise FileNotFoundError(f"Archivo YAML no encontrado: {ruta\_yaml}")  
  
 # Se entrena el modelo con los parametros proporcionados  
 entrenar\_yolo(version=version\_input, variante=variante\_input, ruta\_yaml=ruta\_yaml)  
 break # Salir del bucle si todo es válido  
  
 except ValueError as ve:  
 print(ve)  
 print("Por favor, inténtelo de nuevo con entradas válidas.")  
 except FileNotFoundError as fe:  
 print(fe)  
 break # Salir del bucle si no se encuentra un archivo esencial  
 except Exception as e:  
 print(f"Ocurrió un error inesperado: {e}")

# Validación del Modelo YOLO

Este script permite la gestión de un flujo de trabajo relacionado con el entrenamiento y validación de modelos YOLO. Incluye funciones para subir archivos de pesos desde el sistema local a un directorio en Colab, buscar el archivo más relevante (best.pt) y cargar el modelo **YOLO** para realizar la validación utilizando un archivo **YAML** del conjunto de datos. También verifica métricas clave como precisión, **recall** y **mAP** para evaluar el rendimiento del modelo

# Part 1: Gestión de imports y funciones comunes  
# ==============================================  
  
# Importamos las bibliotecas necesarias  
import os  
from ultralytics import YOLO # Para cargar y validar el modelo YOLO  
from google.colab import files # Para subir archivos desde el sistema local  
  
# Función para crear un directorio si no existe  
def create\_directory(directory\_path):  
 os.makedirs(directory\_path, exist\_ok=True)  
 print(f"Directorio verificado/creado: {directory\_path}")  
  
# Función para subir archivos de pesos  
def upload\_weights(uploaded\_weights\_dir='./uploaded\_weights'):  
 """  
 Permite al usuario subir archivos desde su computadora y los guarda en un directorio específico.  
 """  
 create\_directory(uploaded\_weights\_dir)  
 print("\n=== Subir archivos de pesos ===")  
 uploaded\_files = files.upload() # Subimos los archivos desde el sistema local  
 for file\_name in uploaded\_files.keys():  
 file\_path = os.path.join(uploaded\_weights\_dir, file\_name)  
 with open(file\_path, 'wb') as f: # Guardamos los archivos en el directorio destino  
 f.write(uploaded\_files[file\_name])  
 print(f"Archivo {file\_name} guardado en: {file\_path}")  
 return uploaded\_weights\_dir  
  
# Función para seleccionar un archivo en la lista  
def select\_file\_from\_list(uploaded\_weights\_dir='./uploaded\_weights'):  
 """  
 Permite al usuario seleccionar un archivo en la lista de archivos disponibles.  
 """  
 files = os.listdir(uploaded\_weights\_dir)  
 if not files:  
 raise FileNotFoundError(f"No hay archivos en el directorio: {uploaded\_weights\_dir}")  
  
 print("\nArchivos disponibles:")  
 for idx, file\_name in enumerate(files, 1):  
 print(f"{idx}. {file\_name}")  
  
 while True:  
 try:  
 choice = int(input("\nSeleccione un archivo ingresando su número: ").strip())  
 if 1 <= choice <= len(files):  
 selected\_file = files[choice - 1]  
 print(f"Archivo seleccionado: {selected\_file}")  
 return selected\_file  
 else:  
 print("Número inválido. Intente nuevamente.")  
 except ValueError:  
 print("Entrada inválida. Ingrese un número válido.")  
  
# Función para cargar un modelo YOLO  
def load\_model(weight\_path):  
 """  
 Carga un modelo YOLO desde un archivo de pesos.  
 """  
 if not os.path.exists(weight\_path):  
 raise FileNotFoundError(f"❌ Archivo no encontrado: {weight\_path}")  
 print(f"Cargando modelo desde: {weight\_path}")  
 return YOLO(weight\_path)  
  
# Función para imprimir detalles del modelo  
def print\_model\_details(model):  
 """  
 Imprime la estructura del modelo y el número total de parámetros.  
 """  
 print("\n=== Detalles del modelo ===")  
 print(model) # Imprime la estructura del modelo  
 total\_params = sum(p.numel() for p in model.parameters())  
 print(f"Número total de parámetros: {total\_params}")  
  
# Función para calcular FPS  
def calculate\_fps(results):  
 """  
 Calcula los FPS (Frames per Second) basado en el tiempo de inferencia promedio por imagen.  
 """  
 avg\_inference\_time = results.speed['inference'] # Tiempo promedio de inferencia (ms)  
 fps = 1000 / avg\_inference\_time # Convertir de ms a FPS  
 print("\n=== Métricas de rendimiento ===")  
 print(f"Tiempo de inferencia promedio por imagen: {avg\_inference\_time:.2f} ms")  
 print(f"FPS estimados: {fps:.2f} imágenes/segundo")  
 return fps  
  
# Part 2: Ejecución del flujo principal  
# ================================================  
  
# Directorio para subir archivos de pesos  
uploaded\_weights\_dir = upload\_weights()  
  
# Verificar los archivos en el directorio  
print("Contenido del directorio de archivos subidos:")  
print(os.listdir(uploaded\_weights\_dir))  
  
try:  
 # Seleccionar un archivo de pesos de la lista  
 selected\_weight\_name = select\_file\_from\_list(uploaded\_weights\_dir)  
  
 # Cargar el modelo desde el archivo seleccionado  
 best\_weight\_path = os.path.join(uploaded\_weights\_dir, selected\_weight\_name)  
 model = load\_model(best\_weight\_path)  
  
 # Imprimir los detalles del modelo  
 print\_model\_details(model)  
  
 # Ruta del archivo YAML del conjunto de datos  
 yaml\_file\_path = '/content/drive/My Drive/TFM DATASET YOLO/dataset.yaml'  
 if not os.path.exists(yaml\_file\_path):  
 raise FileNotFoundError(f"❌ Archivo YAML no encontrado: {yaml\_file\_path}")  
 print(f"Archivo YAML encontrado: {yaml\_file\_path}")  
  
 # Validar el modelo  
 print("\n=== Iniciando la validación del modelo ===")  
 results = model.val(  
 data=yaml\_file\_path,  
 imgsz=640,  
 save=True  
 )  
  
 # Mostrar métricas de validación  
 print("\n=== Resultados de la validación ===")  
 print(f"Precisión (P): {results.box.mp:.2f}")  
 print(f"Recall (R): {results.box.mr:.2f}")  
 print(f"mAP@50: {results.box.map50:.2f}")  
 print(f"mAP@50-95: {results.box.map:.2f}")  
  
 # Calcular y mostrar FPS  
 calculate\_fps(results)  
  
except FileNotFoundError as e:  
 print(f"Error: {e}")  
except Exception as e:  
 print(f"Error inesperado: {e}")

# Análisis Detallado de Métricas YOLO por Clase

Este bloque analiza las métricas de validación **YOLO** por clase, incluyendo precisión, recall, [mAP@50](mailto:mAP@50) y [mAP@50-95](mailto:mAP@50-95). Evalúa el rendimiento de cada clase y proporciona una interpretación basada en los valores de las métricas, identificando si el modelo necesita ajustes o si tiene un buen desempeño. Es una herramienta útil para identificar fortalezas y debilidades en el rendimiento del modelo por categoría.

Precisión (P): Indica qué tan exacto es el modelo al detectar objetos, mostrando la proporción de detecciones correctas sobre el total de predicciones realizadas.

Recall (R): Mide la capacidad del modelo para encontrar todos los objetos relevantes en las imágenes.

[mAP@50](mailto:mAP@50): Representa la precisión promedio del modelo usando un umbral de coincidencia (IoU) del 50%.

[mAP@50-95](mailto:mAP@50-95): Evalúa la precisión promedio en un rango de umbrales (50% a 95%), proporcionando una evaluación más completa del rendimiento.

# Bloque para analizar las métricas YOLO por clase  
def analyze\_class\_metrics(results):  
 """  
 Analiza las métricas de validación YOLO para cada clase y muestra el estado por clase.  
 """  
 print("\n=== Análisis de métricas por clase ===\n")  
  
 # Acceder a los nombres de las clases y métricas  
 for i, class\_name in results.names.items():  
 # Obtener las métricas por clase  
 class\_metrics = results.box.class\_result(i) # Devuelve [precision, recall, ap50, ap50-95]  
 precision = class\_metrics[0]  
 recall = class\_metrics[1]  
 map50 = class\_metrics[2]  
 map50\_95 = class\_metrics[3]  
  
 # Mostrar las métricas por clase  
 print(f"=== Clase: {class\_name} ===")  
 print(f"Precisión (P): {precision:.2f}")  
 print(f"Recall (R): {recall:.2f}")  
 print(f"mAP@50: {map50:.2f}")  
 print(f"mAP@50-95: {map50\_95:.2f}")  
  
 # Estado basado en las métricas  
 print("\n=== Estado de la clase ===")  
 if precision > 0.8 and recall > 0.6:  
 print("Buen rendimiento: Alta precisión y recall aceptable.")  
 elif precision > 0.8:  
 print("Alta precisión, pero bajo recall.")  
 elif recall > 0.6:  
 print("Recall aceptable, pero precisión baja.")  
 else:  
 print("Rendimiento insuficiente: Se necesitan ajustes.")  
 print("\n" + "="\*30 + "\n")  
  
  
def calculate\_fps(results):  
 """  
 Calcula los FPS (Frames per Second) basado en el tiempo de inferencia promedio por imagen.  
 """  
 avg\_inference\_time = results.speed['inference'] # Tiempo promedio de inferencia (ms)  
 fps = 1000 / avg\_inference\_time # Convertir de ms a FPS  
 print("\n=== Métricas de rendimiento ===")  
 print(f"Tiempo de inferencia promedio por imagen: {avg\_inference\_time:.2f} ms")  
 print(f"FPS estimados: {fps:.2f} imágenes/segundo")  
 return fps  
  
  
# Ejemplo de uso con los resultados de un modelo YOLO  
try:  
 # Validación del modelo (ejemplo)  
 results = model.val(  
 data=yaml\_file\_path, # Ruta al archivo YAML del dataset  
 imgsz=640, # Tamaño de las imágenes usadas  
 save=True # Guardar los resultados  
 )  
  
 # Analizar las métricas por clase  
 analyze\_class\_metrics(results)  
  
 # Calcular y mostrar los FPS  
 calculate\_fps(results)  
  
except FileNotFoundError as e:  
 print(f"Error: {e}")  
except AttributeError as e:  
 print(f"Error de atributo: {e}")  
except Exception as e:  
 print(f"Error inesperado: {e}")

Error inesperado: name 'model' is not defined

# Visualización de predicciones YOLO

Este script carga un modelo YOLO previamente entrenado y genera predicciones sobre un conjunto de imágenes de test. Las imágenes con los resultados se guardan en un directorio especificado y se muestran directamente en el entorno. Además, se verifica que las imágenes de prueba existan y que el modelo cargado contenga un archivo de pesos con el prefijo "best".

import os  
import glob  
import matplotlib.pyplot as plt  
from PIL import Image  
from ultralytics import YOLO  
from google.colab import files  
  
  
import locale  
# Correction locale UTF-8  
locale.getpreferredencoding = lambda: "UTF-8"  
  
drive.mount('/content/drive')  
!ls "/content/drive/My Drive/TFM DATASET YOLO/"  
print(os.listdir('/content/drive/My Drive/TFM DATASET YOLO/test1'))  
  
# Directorio para cargar el archivo de pesos  
uploaded\_weights\_dir = './uploaded\_weights'  
os.makedirs(uploaded\_weights\_dir, exist\_ok=True)  
  
# Subir el archivo de pesos  
print("\n=== Cargar el modelo YOLO ===")  
uploaded\_files = files.upload()  
  
# Detectar el archivo de pesos subido  
best\_weight\_path = None  
for file\_name in uploaded\_files.keys():  
 file\_path = os.path.join(uploaded\_weights\_dir, file\_name)  
 with open(file\_path, 'wb') as f:  
 f.write(uploaded\_files[file\_name])  
 if "best" in file\_name:  
 best\_weight\_path = file\_path  
 print(f"Archivo {file\_name} guardado en: {file\_path}")  
  
if not best\_weight\_path:  
 raise FileNotFoundError("❌ No se encontró ningún archivo con 'best' en el nombre entre los archivos subidos.")  
  
# Configurar directorios  
test\_dir = '/content/drive/My Drive/TFM DATASET YOLO/test1' # Directorio con imágenes de prueba  
predictions\_dir = '/content/drive/My Drive/TFM DATASET YOLO/predictions/' # Directorio para guardar las predicciones  
  
# Crear el directorio de predicciones si no existe  
os.makedirs(predictions\_dir, exist\_ok=True)  
  
  
# Verificar si el directorio de imágenes de prueba contiene imágenes  
image\_paths = glob.glob(os.path.join(test\_dir, '\*.[jp][pn]g')) + glob.glob(os.path.join(test\_dir, '\*.png')) # Procesar solo imágenes .jpg  
if not image\_paths:  
 print(f"No se encontraron imágenes en el directorio: {test\_dir}")  
 raise FileNotFoundError(f"No se encontraron imágenes en el directorio: {test\_dir}")  
else:  
 print(f"{len(image\_paths)} imágenes encontradas. Generando predicciones...")  
  
# Cargar el modelo YOLO  
print(f"Cargando el modelo desde: {best\_weight\_path}")  
model = YOLO(best\_weight\_path)  
  
# Generar predicciones y mostrar resultados  
for img\_path in image\_paths:  
 # Realizar predicción  
 results = model.predict(  
 source=img\_path, # Imagen de entrada  
 save=True, # Guardar resultados  
 save\_txt=True, # Guardar resultados en formato TXT (opcional)  
 conf=0.25, # Umbral de confianza para las predicciones  
 project=predictions\_dir, # Directorio para guardar las predicciones  
 name='predictions', # Subdirectorio donde se organizan las predicciones  
 exist\_ok=True # No sobrescribir directorios existentes  
 )  
  
 # Mostrar la imagen predicha  
 predicted\_image\_path = os.path.join(predictions\_dir, 'predictions', os.path.basename(img\_path))  
 if os.path.exists(predicted\_image\_path):  
 predicted\_image = Image.open(predicted\_image\_path)  
 plt.figure(figsize=(8, 8))  
 plt.imshow(predicted\_image)  
 plt.axis('off') # Ocultar ejes  
 plt.title(f"Predicción para: {os.path.basename(img\_path)}")  
 plt.show()  
  
print(f"Las predicciones se han guardado en: {predictions\_dir}")



