***Descripción formato YOLOv8***

Para empezar, las imágenes que se van a usar como entrada para la red neuronal deben estar anotadas. Para ello se puede usar una herramienta online de anotación de visión artificial como [ésta](https://app.cvat.ai/).

En esta herramienta, se define un proyecto y una tarea de anotación para anotar las imágenes que se suben a la herramienta desde local o cualquier otro medio. Es importante guardar ocasionalmente para no perder las anotaciones realizadas hasta ese momento. Una vez se anotan todas las imágenes, se debe exportar la tarea de anotación en formato YOLO, ya que YOLO sigue también una estructura de carpetas para realizar el entrenamiento del modelo:

**dataset-|**

**|-images-| (obligatorio)**

**|                      |- train (NO obligatorio)**

**| |- imagen\_0001.jpg**

**|                      |- test (NO obligatorio)**

**|                      |- val (NO obligatorio)**

**|-labels-| (obligatorio)**

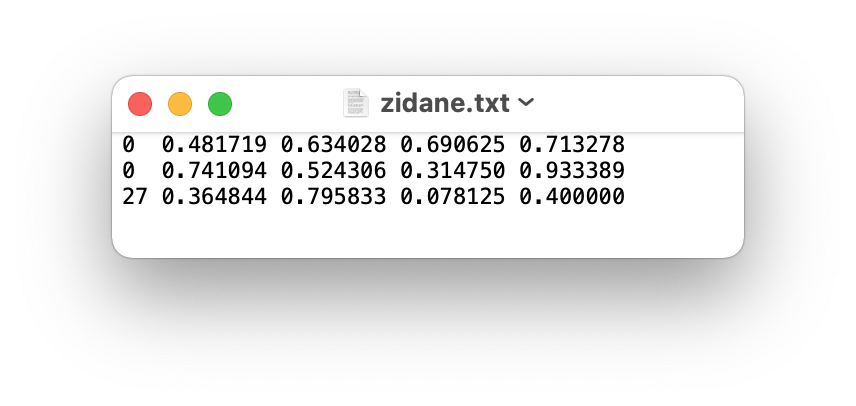
**|- train (NO obligatorio)**

**|- imagen\_0001.txt**

**|- test (NO obligatorio)**

**|- val (NO obligatorio)**

En cada archivo de texto habrá una fila por anotación realizada en esa imagen, tal como se puede ver en la imagen (en este caso, hay, por tanto, tres anotaciones en la imagen):



Y en cada fila aparecen los siguientes datos:

* Clase de la anotación
* Posición en el eje X del centro de la anotación (normalizada al tamaño de la imagen)
* Posición en el eje Y del centro de la anotación (normalizada al tamaño de la imagen)
* Anchura de la anotación (normalizada al tamaño de la imagen)
* Altura de la anotación (normalizada al tamaño de la imagen)

Respecto a las coordenadas, anchuras y alturas, si éstas están en píxeles, para normalizarlas hay que dividir la coordenada en el eje X y la anchura por el ancho total de la imagen; y la coordenada en el eje Y y la altura por la altura total de la imagen.

El origen de las imágenes se encuentra en la esquina superior izquierda de la misma:

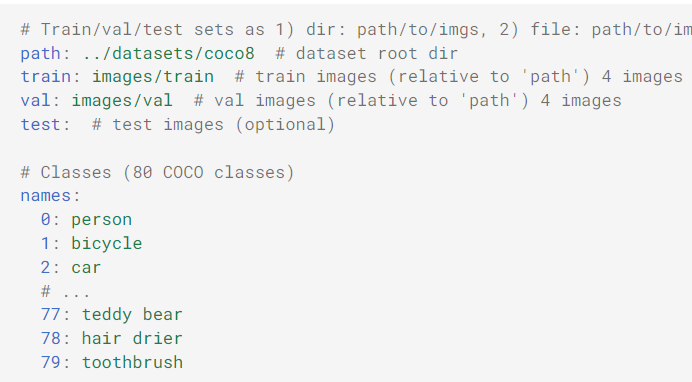


**IMPORTANTE**: Tiene que existir un ‘**.txt’** por cada imagen, aunque la imagen no contenga ninguna anotación.

Una vez los datos estén en el formato especificado en este apartado, ya se podría entrenar el modelo.

***Entrenamiento detección de objetos YOLOv8***

Para realizar el entrenamiento caso hay que definir un archivo de configuración **YAML** como el siguiente, donde se especifica:



* La ruta absoluta del directorio raíz donde se encuentra el dataset (**path**), es decir, la ruta hacia el directorio donde se encuentran las carpetas **images** y **labels** comentadas en el apartado anterior sobre la estructura del dataset.
* Las rutas relativas al directorio raíz donde se encuentran las imágenes de entrenamiento (**train**), validación (**val**) o test (**test**), en caso de que existan estas subcarpetas.
* El diccionario **names**, que especifica el número y el nombre de cada una de las clases a detectar durante el proceso de entrenamiento (alternativamente se puede usar **nc** para especificar el número de clases y **names** como una lista para nombrar todas).

Una vez se tiene este archivo **YAML**, se puede entrenar ya el modelo. Esto se puede realizar mediante dos métodos

* Mediante un simple código en Python: Una vez el archivo de configuración YAML este configurado, se puede realizar el entrenamiento con varias simples líneas de código:

# Importa librería

from ultralytics import YOLO

# Carga la estructura del modelo YOLO (en este caso la versión ‘nano’)

model = YOLO("yolov8n.yaml") # build a new model from scratch

# model = YOLO('yolov8n.pt') # load pretrained model (recommended for training)

# Entrenamiento especificando en el parámetro ‘data’ la ruta del archivo

# de configuración YAML

model.train(data="config.yaml", epochs=3)

Cuando termina el entrenamiento, se guardan los resultados en un directorio especificado en los logs de ese entrenamiento y que se crea solo. En este directorio se ven, por ejemplo, las predicciones en el set de validación y las métricas en imagen.

* Mediante una interfaz de línea de comandos (CLI): En este caso para entrenar bastaría un comando que sigue las siguientes directrices:



En este caso, para detección de objetos y entrenamiento, el comando sería el siguiente:

yolo detect train data=config.yaml model=yolov8n.pt epochs=2

La lista completa de parámetros para realizar el entrenamiento en Python (o de argumentos para el modo CLI) se encuentra [aquí](https://docs.ultralytics.com/es/modes/train/#train-settings). Las opciones de **data augmentation** [aquí](https://docs.ultralytics.com/es/modes/train/#augmentation-settings-and-hyperparameters).

**IMPORTANTE**: si se entrena en Colab, hay que copiar los resultados al Drive de forma manual o ejecutar este comando (y así no perderlos tras desconectar el entorno).

¡scp -r /content/runs ‘DirectorioColabDondeEstaDataset’

***Inferencia detección de objetos YOLOv8***

Una vez terminado el entrenamiento, en la carpeta de resultados creada se crean una serie de de archivos (imagen con la matriz de confusión, diferentes curvas para métricas y *loss,* predicciones sobre las imágenes de validación. Toda la información sobre predicción [aquí](https://docs.ultralytics.com/es/modes/predict/).

Una vez el modelo ha sido entrenado, se puede realizar la inferencia a imágenes o a vídeos.

Código de ejemplo para cargar vídeos:

import os

from ultralytics import YOLO

import cv2

VIDEOS\_DIR = os.path.join('.', 'videos')

video\_path = os.path.join(VIDEOS\_DIR, 'alpaca1.mp4')

video\_path\_out = '{}\_out.mp4'.format(video\_path)

cap = cv2.VideoCapture(video\_path)

ret, frame = cap.read()

H, W, \_ = frame.shape

out = cv2.VideoWriter(video\_path\_out, cv2.VideoWriter\_fourcc(\*'MP4V'), int(cap.get(cv2.CAP\_PROP\_FPS)), (W, H))

# Load a model

model\_path = os.path.join('.', 'runs', 'detect', 'train', 'weights', 'last.pt')

model = YOLO(model\_path) # load a custom model

threshold = 0.5

while ret:

results = model(frame)[0]

for result in results.boxes.data.tolist():

x1, y1, x2, y2, score, class\_id = result

if score > threshold:

cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 4)

cv2.putText(frame, results.names[int(class\_id)].upper(), (int(x1), int(y1 - 10)), cv2.FONT\_HERSHEY\_SIMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE\_AA)

out.write(frame)

ret, frame = cap.read()

cap.release()

out.release()

cv2.destroyAllWindows()