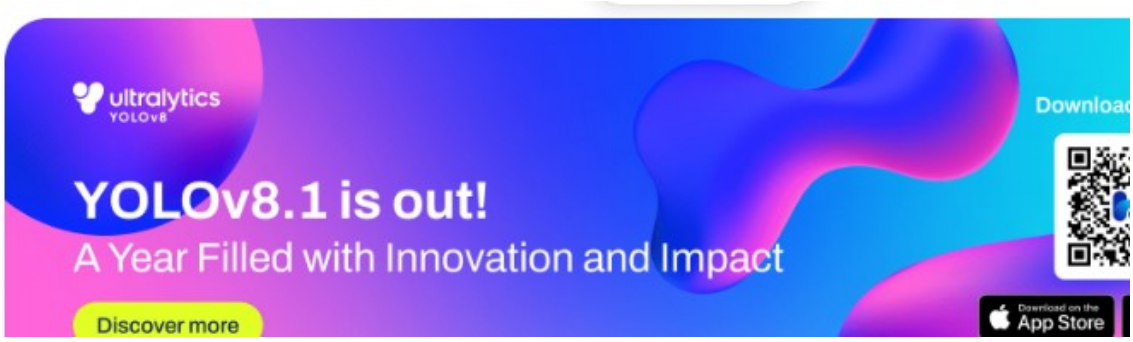
Capítulo 2: Conociendo YOLO (You Only Look Once)



Machine Learning _YOLO

Toda la documentación está disponible en :

https://docs.ultralytics.com/es/

1. Preparando el Dataset

Vamos a preparar nuestro modelo para entrenar, para ello lo primero es saber ¿¿¿Qué entrenamos???? , en este caso como ejemplo hemos elegido generar un modelo personalizado que reconozca naranjas

1º Recopilamos a modo de comienzo de 20 fotos (60 sería ideal).

Foto 1: Fotos del modelo e entrenar

Una vez definidos las fotos, se procede a reunir una variedad de imágenes que contengan dichos objetos o clases. Estas imágenes deben ser de alta calidad y tener una resolución adecuada para asegurar un rendimiento óptimo del modelo de reconocimiento de máquina. Es importante incluir imágenes que cubran diferentes

variaciones, como ángulos, iluminaciones y fondos, para garantizar la robustez del modelo ante diversas condiciones.

CVAT (Computer Vision Annotation Tool) es una herramienta poderosa que facilita el proceso de etiquetado y parametrización de las imágenes. Después de instalar y configurar CVAT, se crea un nuevo proyecto en la plataforma y se importan las imágenes recopiladas. En este punto, se definen las clases de objetos que se van a etiquetar en las imágenes.

El siguiente paso implica etiquetar manualmente cada imagen mediante la creación de cajas de delimitación alrededor de los objetos de interés y asignándoles la clase correspondiente. Es fundamental etiquetar todas las instancias de los objetos en cada imagen para garantizar que el modelo pueda reconocerlos con precisión.

Tras el etiquetado, se realiza una revisión exhaustiva para verificar la precisión y la completitud de las etiquetas. Se corrigen los errores de etiquetado y se añaden etiquetas faltantes si es necesario. Una vez finalizada la revisión, se exportan los datos etiquetados en el formato deseado, como XML, JSON, YOLO, VOC, entre otros, para su uso en el entrenamiento de modelos de reconocimiento de máquina.

Las fotos y los archivos de texto generados están estrechamente relacionados en el contexto del reconocimiento de máquina. Cuando se recopilan fotos para entrenar un modelo de inteligencia artificial, es fundamental etiquetar estas imágenes para indicar la presencia y la ubicación de los objetos de interés en ellas. Estas etiquetas se almacenan típicamente en archivos de texto, donde cada archivo está asociado con una imagen y contiene la información de las coordenadas de los objetos etiquetados.

La relación entre las fotos y los archivos de texto generados radica en que los archivos de texto proporcionan la información necesaria para entrenar y evaluar modelos de reconocimiento de máquina utilizando las imágenes recopiladas. Por ejemplo, si se está entrenando un modelo para reconocer vehículos en imágenes de tráfico, los archivos de texto contendrían las coordenadas de las cajas delimitadoras que rodean cada vehículo en las imágenes correspondientes. Estas coordenadas indican al modelo dónde se encuentran los vehículos en la imagen y qué forma tienen.

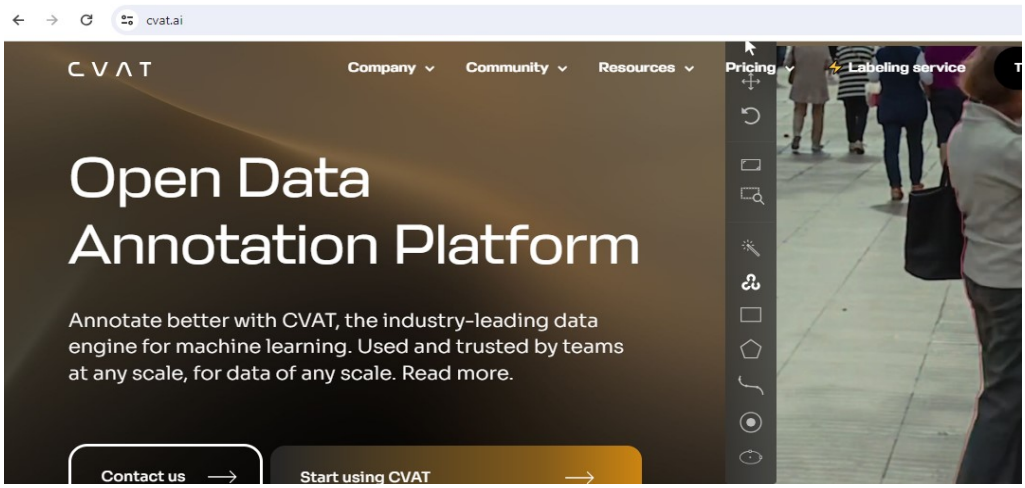
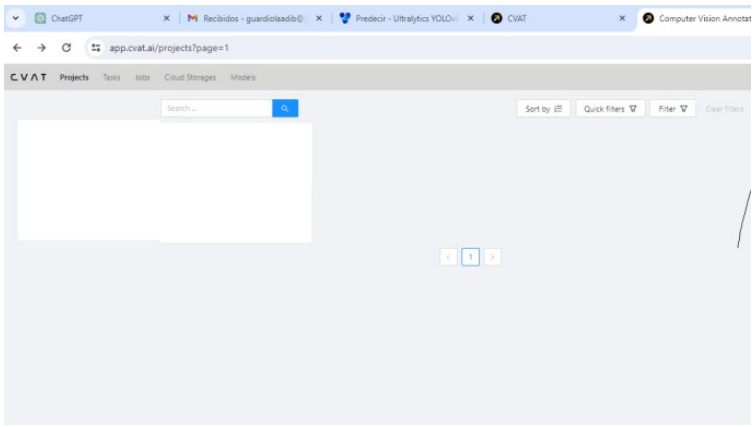
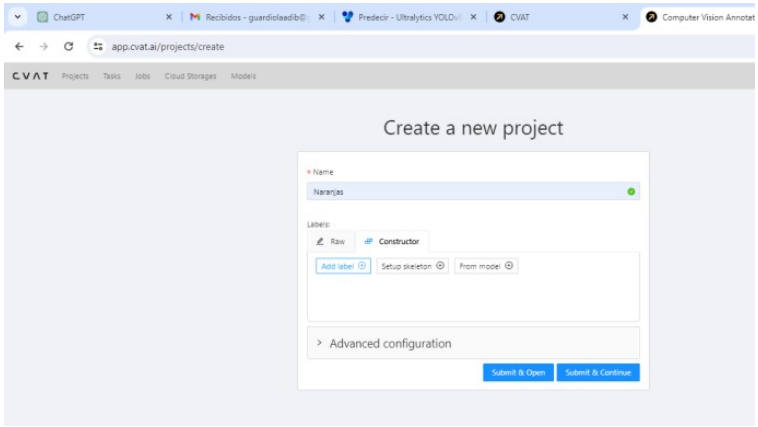


Foto2 :CVAT software para etiquetar los marcadores de referencia

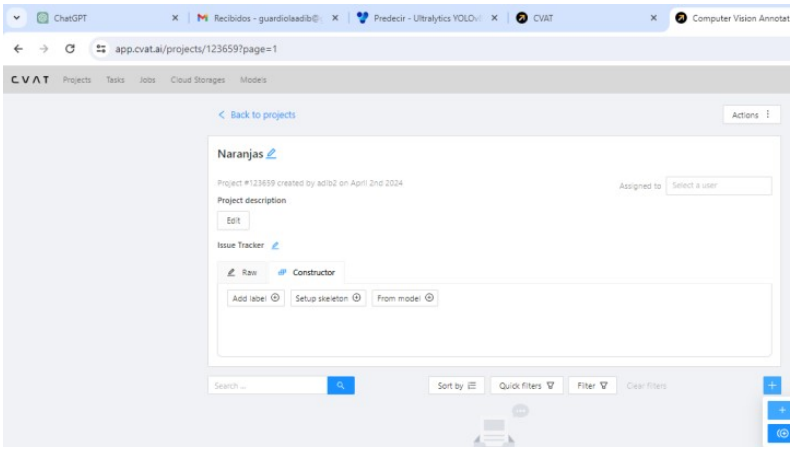
1.1 Preparando los archivos de imagen .txt



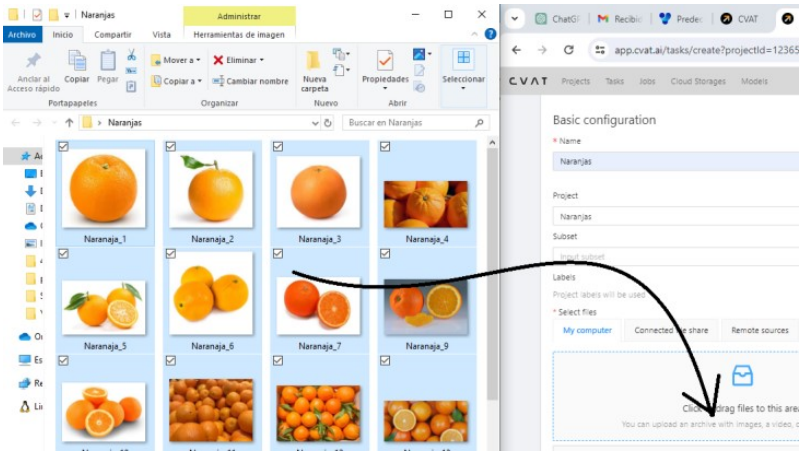
Creamos nuevo proyecto



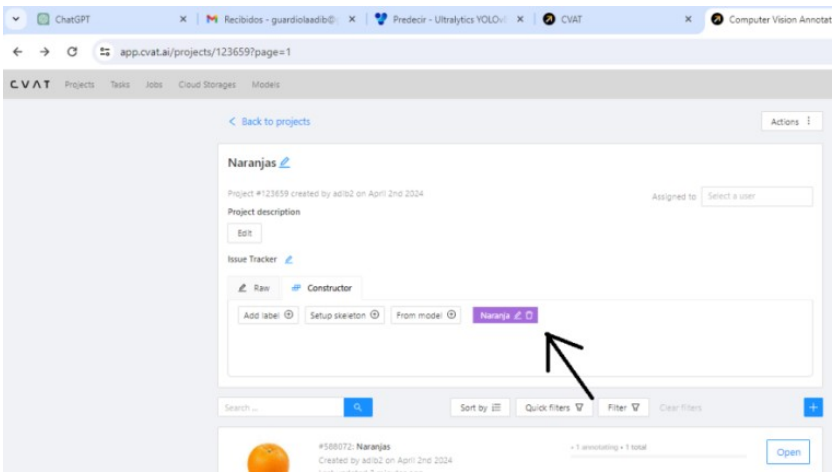
Lo nombramos y le damos etiqueta



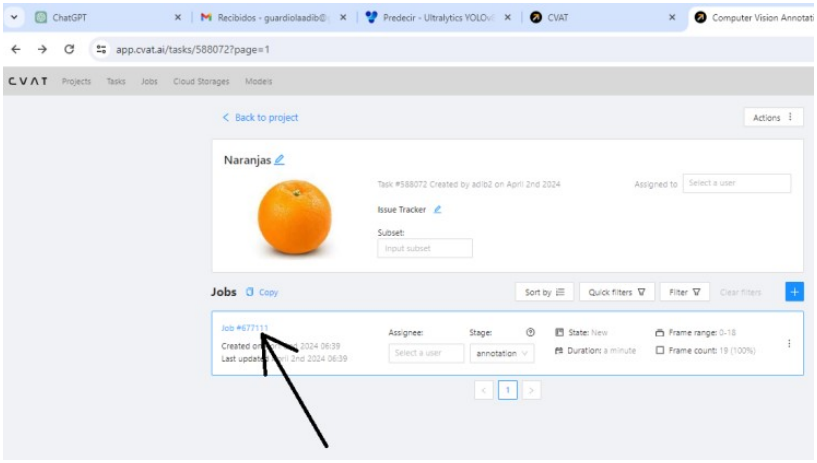
Entramos en el proyecto y le damos a new task



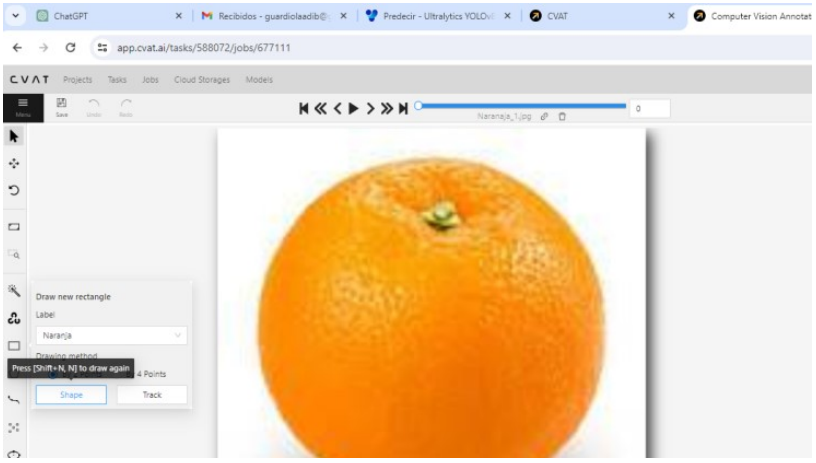
En configuración básica arrastramos las fotos a la zona de análisis de imágenes.



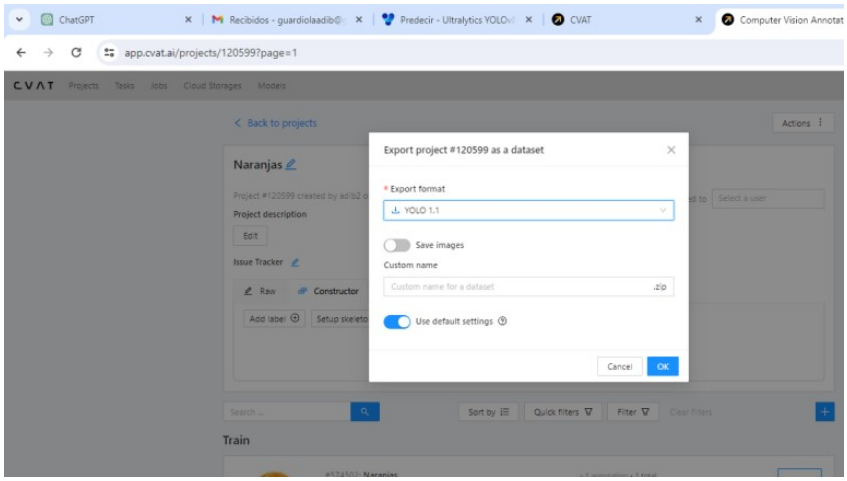
En la creación de nuestro proyecto es importante darle una etiqueta, piensa que por cada foto puedes dar varias etiquetas con nombres diferentes.



Una vez con el proyecto creado nos vamos a job asignado.



Etiquetamos todas las imágenes



Una vez que tengamos todas las etiquetas marcadas ,procedemos a exportar nuestro Dataset.

2. Entrenando el modelo

De nuestra aplica CVAT hemos obtenido unos archivo .txt uno por cada foto de forma que la estructura de los datos para entrenar un modelo YOLO generalmente consta de imágenes en formato .jpg junto con archivos de texto .txt que contienen las etiquetas de los objetos en esas imágenes. Aquí hay una descripción de la estructura típica:

Imágenes (*.jpg): Estas son las imágenes de entrada que se utilizarán para entrenar el modelo. Deben contener los objetos que deseas que el modelo detecte.

Etiquetas (*.txt): Para cada imagen, hay un archivo de texto asociado con la misma base (por ejemplo, si la imagen se llama imagen.jpg, el archivo de texto correspondiente se llamará imagen.txt). Cada archivo de texto contiene información sobre los objetos presentes en la imagen en el formato siguiente:

Cada línea del archivo de texto corresponde a un objeto detectado en la imagen.

Cada línea contiene cinco valores separados por comas: class_id, x_center, y_center, width, height.

class_id es el índice de la clase del objeto (empezando desde 0).

x_center y y_center son las coordenadas normalizadas del centro del objeto (en relación con el ancho y alto de la imagen).

width y height son el ancho y alto normalizados del objeto (en relación con el ancho y alto de la imagen).

Por ejemplo, un archivo de texto podría tener el siguiente contenido:

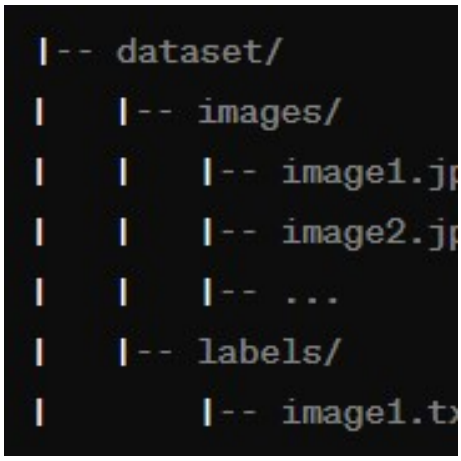
0 0.5 0.5 0.3 0.4 1 0.8 0.6 0.2 0.3

Esto indicaría que hay dos objetos en la imagen:

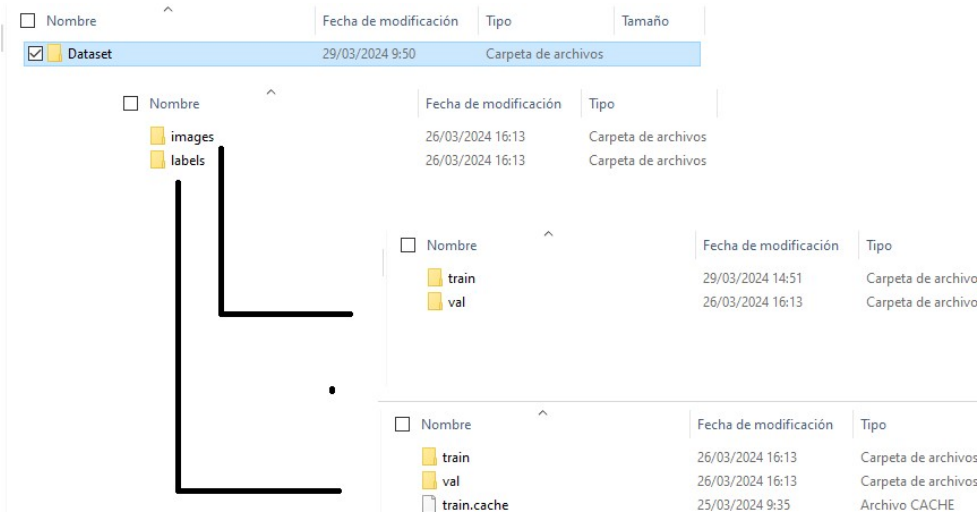
El primer objeto tiene class_id = 0 (la primera clase), con su centro en (0.5, 0.5) de la imagen y dimensiones normalizadas de 0.3x0.4.

El segundo objeto tiene class_id = 1 (la segunda clase), con su centro en (0.8, 0.6) de la imagen y dimensiones normalizadas de 0.2x0.3.

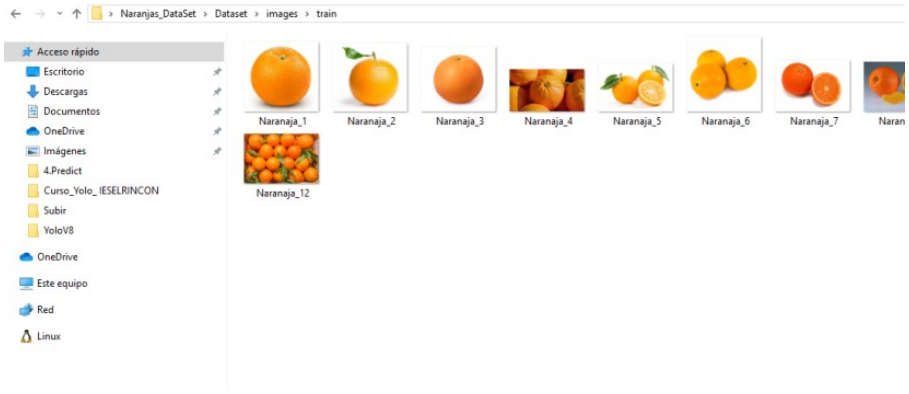
Es importante tener en cuenta que las coordenadas y dimensiones están normalizadas en el rango [0, 1], donde (0,0) es la esquina superior izquierda de la imagen y (1,1) es la esquina inferior derecha.



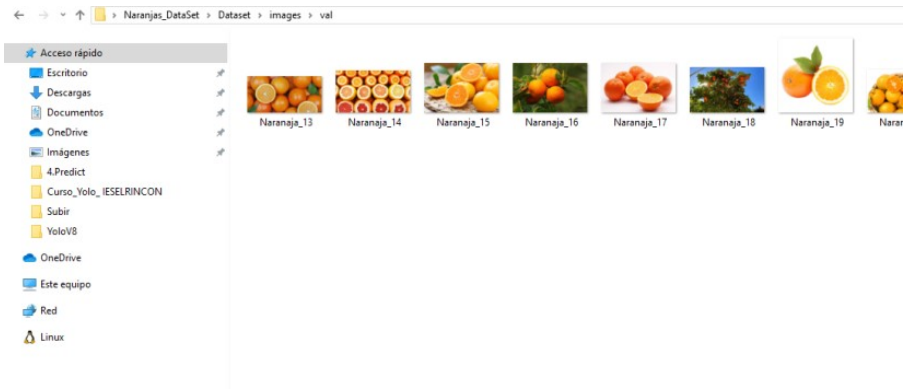
Estructura de las carpetas de nuestro DataSET



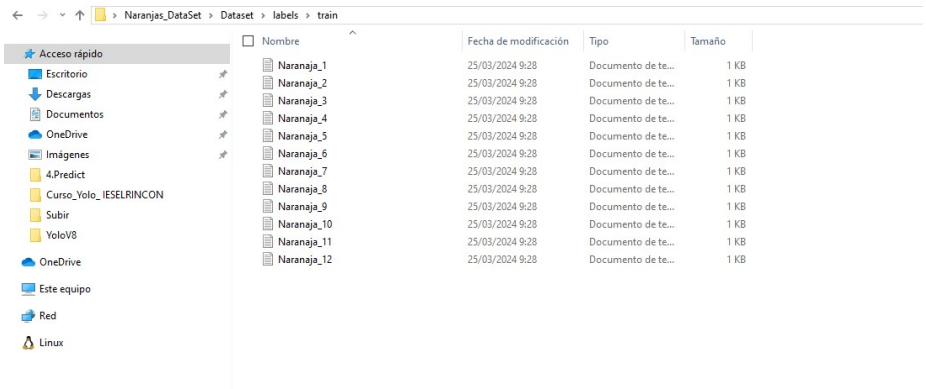
Creamos una carpeta principal llamada Dataser donde habrán dos carpetas ,una llamada images y otra labels. Dentro de cada unas de las carpetas habrán dos directorios llamados train y val.



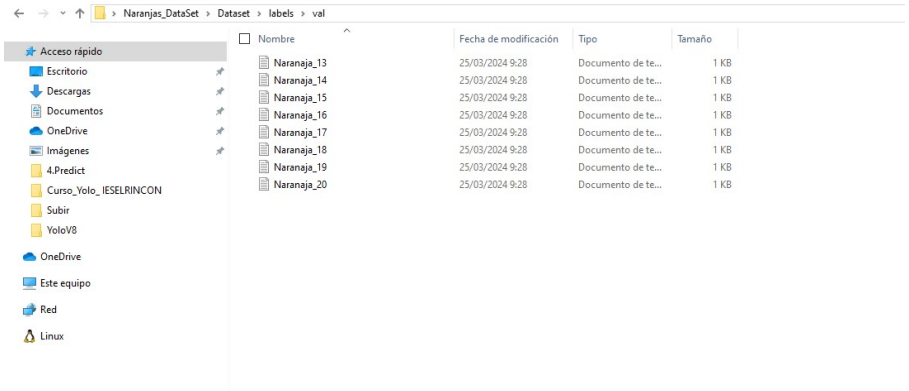
Archivo **images/train**



Archivo **images/val**



Archivo **labels/train**



Archivo **labels/val**

El entrenamiento de modelos de aprendizaje automático en Google Colab se ha vuelto cada vez más popular debido a su acceso gratuito a GPUs y TPUs, lo que acelera significativamente el proceso de entrenamiento. Al utilizar Colab, los usuarios pueden aprovechar recursos computacionales poderosos sin la necesidad de una GPU local, lo que es especialmente beneficioso para tareas que requieren un procesamiento intensivo, como el entrenamiento de modelos de visión por computadora.

El proceso de entrenamiento en Colab generalmente implica cargar y preprocesar los datos, definir y compilar el modelo de aprendizaje automático, y finalmente entrenar el modelo utilizando el conjunto de datos proporcionado. El código se ejecuta en el entorno de Colab y los resultados se muestran en tiempo real, lo que facilita la experimentación y la depuración.

El fragmento de código proporcionado se utilizaría para entrenar un modelo de aprendizaje automático utilizando el conjunto de datos especificado en 'config.yaml' durante 80 épocas. Este modelo utiliza el modelo YOLOv8n preentrenado y se entrena con un tamaño de imagen de 600x600 píxeles.

El código comienza montando Google Drive para acceder al archivo de configuración y otros recursos necesarios. Luego, se instala la biblioteca Ultralytics, que proporciona funcionalidades para entrenar modelos YOLO en Colab. El modelo se entrena utilizando el método train de Ultralytics, donde se especifica el archivo de configuración, el número de épocas y el tamaño de la imagen, entre otros parámetros.

El uso de Google Colab para entrenar modelos de aprendizaje automático ofrece varias ventajas, como acceso gratuito a recursos de GPU/TPU, facilidad de uso, y la posibilidad de colaborar y compartir proyectos con otros usuarios. En resumen, Google Colab se ha convertido en una herramienta invaluable para la comunidad de aprendizaje automático al proporcionar un entorno de desarrollo potente y accesible.

Pegamos en Google Colab:

```
!nvidia-smi
```

```
import os
HOME = os.getcwd()
print(HOME)
```

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
import tensorflow as tf
import timeit
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  print(
      '\n\nThis error most likely means that this notebook is not '
      'configured to use a GPU.  Change this in Notebook Settings via the '
      'command palette (cmd/ctrl-shift-P) or the Edit menu.\n\n')
  raise SystemError('GPU device not found')
def cpu():
  with tf.device('/cpu:0'):
    random_image_cpu = tf.random.normal((100, 100, 100, 3))
    net_cpu = tf.keras.layers.Conv2D(32, 7)(random_image_cpu)
    return tf.math.reduce_sum(net_cpu)
def gpu():
  with tf.device('/device:GPU:0'):
    random_image_gpu = tf.random.normal((100, 100, 100, 3))
    net_gpu = tf.keras.layers.Conv2D(32, 7)(random_image_gpu)
    return tf.math.reduce_sum(net_gpu)
# We run each op once to warm up; see: https://stackoverflow.com/a/45067900
cpu()
gpu()
# Run the op several times.
print('Time (s) to convolve 32x7x7x3 filter over random 100x100x100x3 images '
      '(batch x height x width x channel). Sum of ten runs.')
print('CPU (s):')
cpu_time = timeit.timeit('cpu()', number=10, setup="from __main__ import cpu")
print(cpu_time)
print('GPU (s):')
gpu_time = timeit.timeit('gpu()', number=10, setup="from __main__ import gpu")
print(gpu_time)
print('GPU speedup over CPU: {}x'.format(int(cpu_time/gpu_time)))
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
!pip install ultralytics
from ultralytics import YOLO
from IPython.display import display, Image
```

```
from ultralytics import YOLO

# Carga un modelo YOLOv8n preentrenado en COCO
model = YOLO('yolov8n.pt')

# Entrena el modelo utilizando el conjunto de datos especificado en config.yaml durante 80 épocas
model.train(data='/content/drive/MyDrive/config.yaml', epochs=80, imgsz=600)
```

Resumen del código

Este código ejecuta una serie de acciones en un entorno de Google Colab. Aquí hay un resumen de lo que hace cada parte:

- **Verificación de la GPU**: Comprueba si una GPU está disponible en el entorno de Colab utilizando `!nvidia-smi`. Luego, utiliza TensorFlow para detectar la GPU disponible y asegurarse de que la GPU esté disponible y funcione correctamente. Si no se encuentra una GPU, se muestra un error.

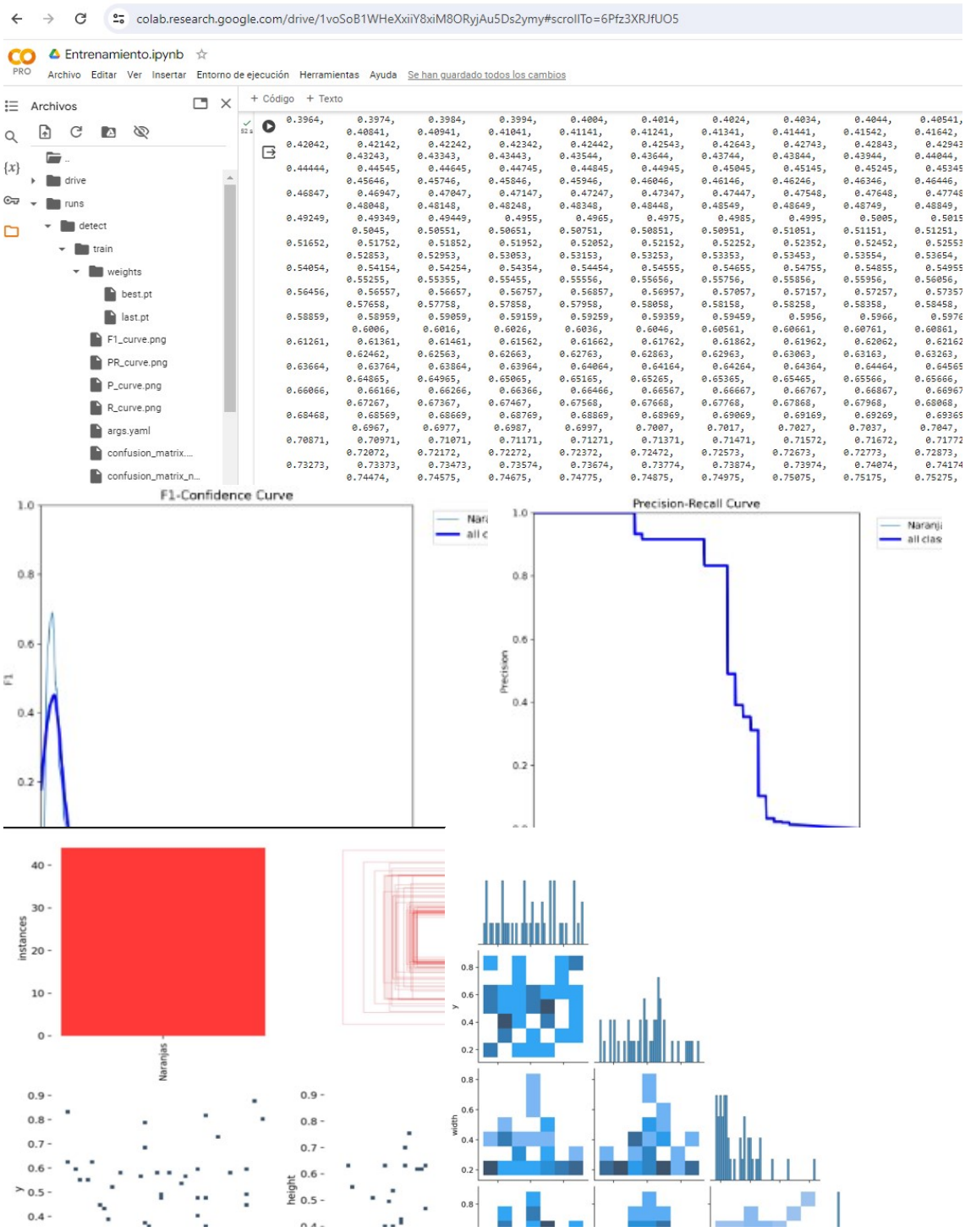
- **Pruebas de rendimiento**: Define dos funciones `cpu()` y `gpu()` que realizan operaciones de convolución en imágenes aleatorias utilizando la CPU y la GPU, respectivamente. Luego, ejecuta estas funciones para medir el tiempo de ejecución en la CPU y la GPU y calcula la aceleración de la GPU sobre la CPU.

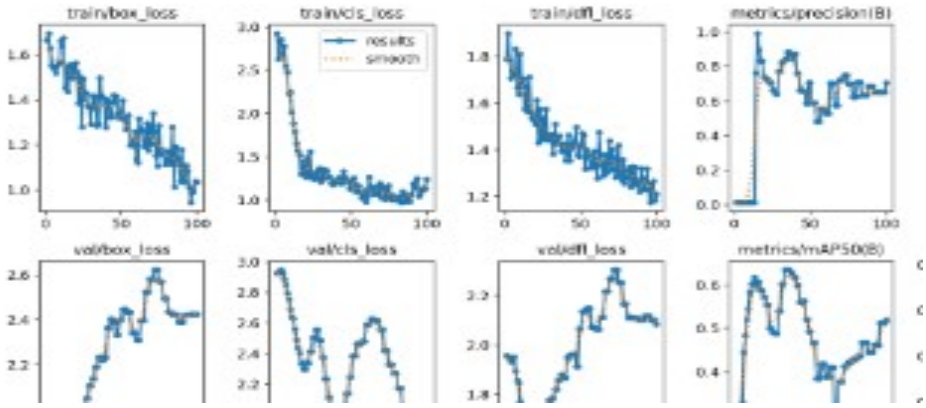
- **Montaje de Google Drive**: Monta Google Drive en el entorno de Colab para acceder a los archivos almacenados en Google Drive. Esto permite acceder a los archivos de configuración y datos necesarios para el entrenamiento del modelo.

- **Instalación de bibliotecas**: Instala la biblioteca Ultralytics, que proporciona herramientas para entrenar modelos YOLO en Colab.

- **Carga del modelo YOLO**: Carga un modelo YOLOv8n preentrenado en COCO utilizando la biblioteca Ultralytics.

- **Entrenamiento del modelo**: Inicia el entrenamiento del modelo YOLO utilizando el conjunto de datos especificado en el archivo de configuración durante 80 épocas y con un tamaño de imagen de 600x600 píxeles.





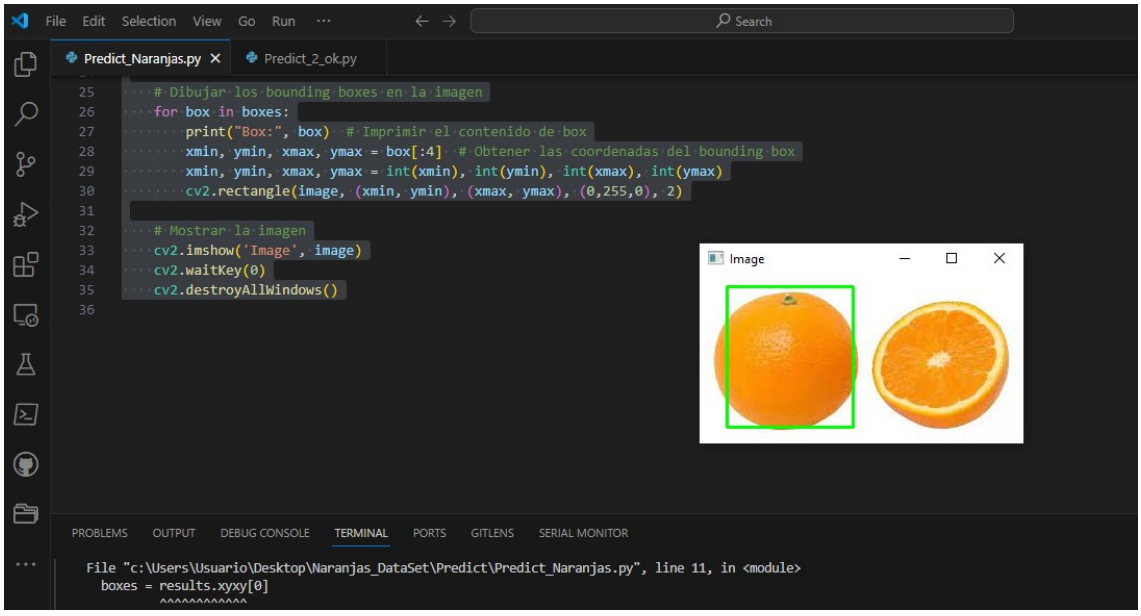
3.Comprobando los resultados de predicción

Una vez terminado el modelo probamos la predicción con vuestro modelo en con el siguiente script:

```
import cv2
from ultralytics import YOLO
import numpy as np

# Cargar el modelo YOLO preentrenado
model = YOLO("C:/Users/Usuario/Desktop/YoloV8/4.Predict/last_Naranjas.pt")

# Realizar la predicción en una imagen
results = model("C:/Users/Usuario/Desktop/YoloV8/4.Predict/imagen3.jpg", conf=0.3, iou=0.5)
```

```
# Iterar sobre cada resultado en la lista
for result in results:
    # Obtener las coordenadas de los bounding boxes
    boxes = result.boxes[0].xyxy

    # Obtener el diccionario de nombres de clases
    names_dict = result.names

    # Cargar la imagen
    image = cv2.imread("C:/Users/Usuario/Desktop/YoloV8/4.Predict/imagen3.jpg")

    # Dibujar los bounding boxes en la imagen
    for box in boxes:
        print("Box:", box)  # Imprimir el contenido de box
        xmin, ymin, xmax, ymax = box[:4]  # Obtener las coordenadas del bounding box
        xmin, ymin, xmax, ymax = int(xmin), int(ymin), int(xmax), int(ymax)
        cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (0,255,0), 2)

    # Mostrar la imagen
    cv2.imshow('Image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

En la misma carpeta debemos tener el este script la imagen que vamos a realizar la predicción ,así como nuestro modelo ya sea best.pt o last.pt



Reto del aprendizaje:

https://www.youtube.com/watch?v=m9fH9OWn8YM&ab_channel=Computervisionengineer

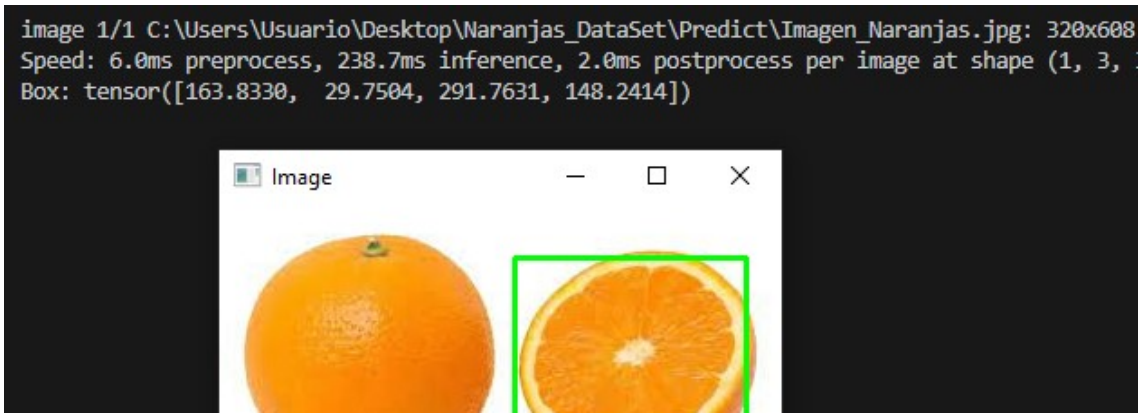
https://www.youtube.com/watch?v=ZeLg5rxLGLg&list=RDCMUCLALGa2fSkx0MI88eALnILw&index=2&ab_channel=Computervisionengineer

Enunciado:

El script de predicción que tienes, está incompleto ,detecta a los objetos para lo cual ha sido entrenado pero no los marca con los bounding box (sólo algunos) y además le falta las etiquetas.

Objetivo :

Hacer un script similar al propuesto ,mostrando todos los bounding box y sus etiquetas.



Fíate que marca dos naranjas y los hay un bounding box ,habría que añadir las etiquetas, tanto de nombre como valor de confianza.