TAREA 6 (24-25) DETECCIÓN DE OBJETOS CON YOLO

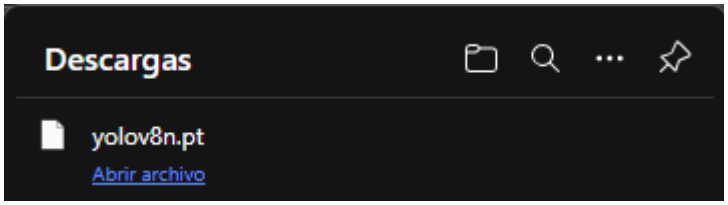
Link al repositorio de github: Repositorio de github

Índice

Script de iniciación

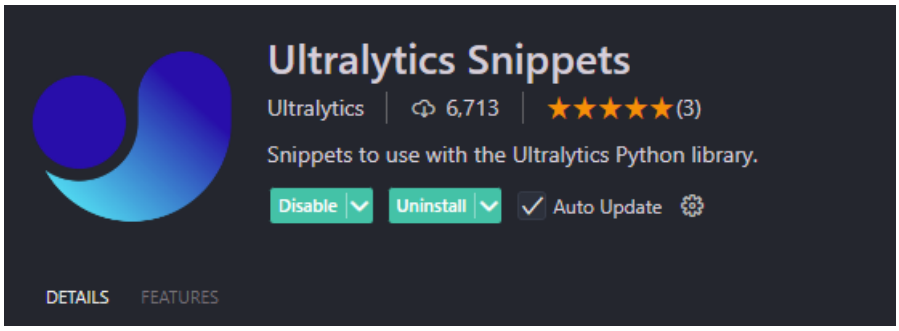
Descargar el archivo Yolov8n

Descargamos el archivo con el link: Yolov8n



Abrir visual Code Studio e instalar Ultralytics

Instalamos la extensión en nuestro Visual Studio Code



Copiamos el código y lo ejecutamos

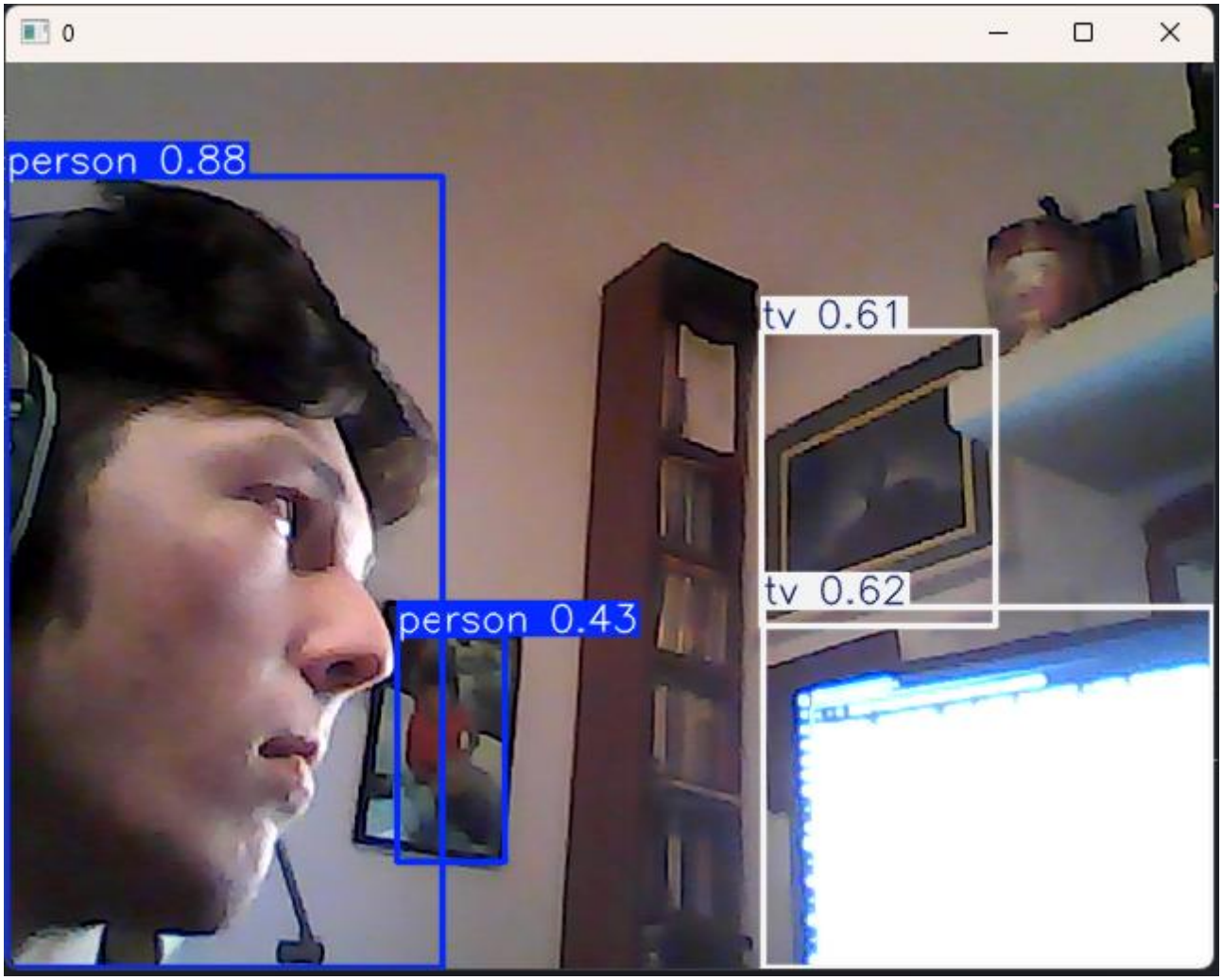
Antes de ejecutar el código, he tenido que instalar la librería “ultralytics” con este comando:

```
pip install ultralytics
```

Utilizo una versión del código proporcionado en el pdf:

```python
from ultralytics import YOLO
import cv2

# Instancia del modelo YOLO con los pesos especificados
pesos_yolo = "yolov8n.pt" # Usamos la ruta relativa
model = YOLO(pesos_yolo)

# Verificar si la cámara está disponible
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("No se pudo acceder a la cámara.")
else:
    try:
        # Realiza la detección de objetos en el flujo de video de la cámara del computador
        results = model(source=0, show=True, conf=0.3, save=True)
    except Exception as e:
        print("Ocurrió un error al realizar la detección de objetos:", e)
    finally:
        cap.release()
```

Aquí el resultado:

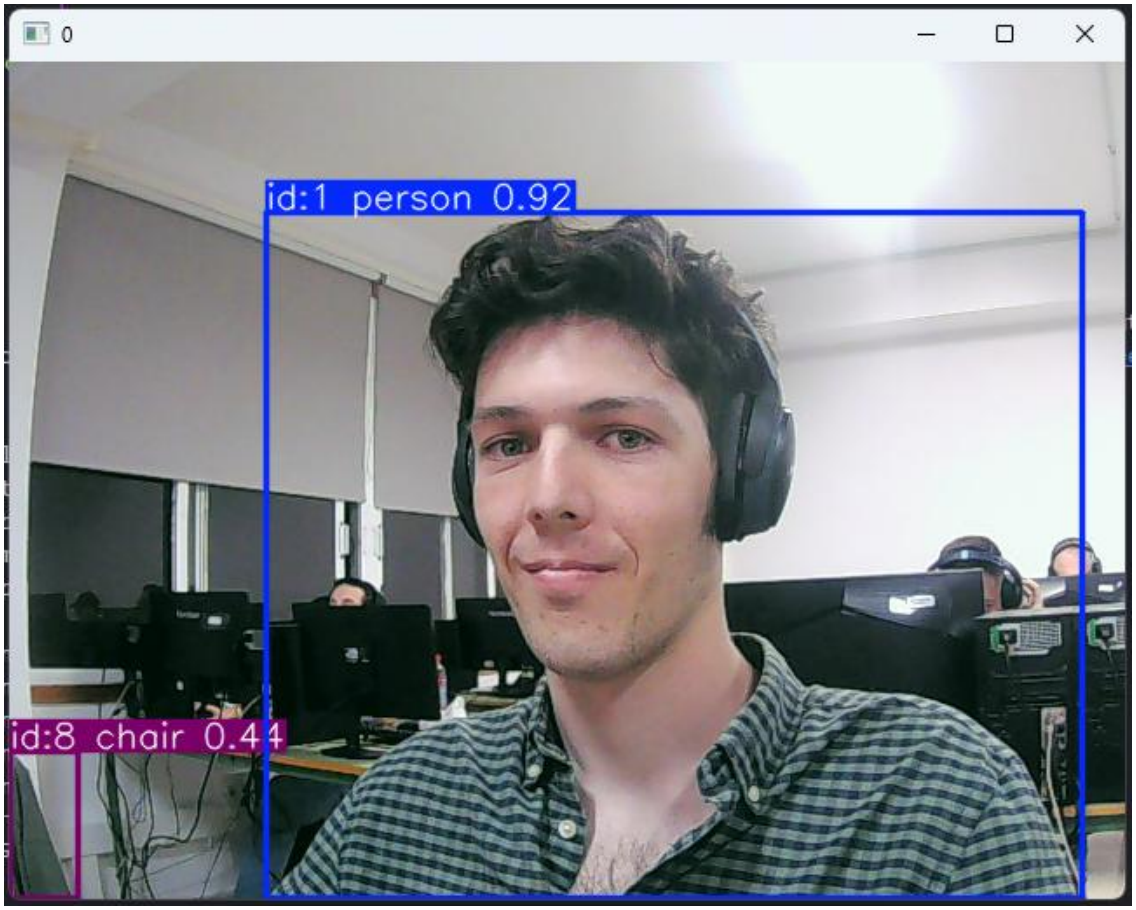


Script de tracking

Utilizamos el siguiente script para hacer un tracking de lo que capture la cámara.

```
# Tracking
from ultralytics import YOLO
# Configure the tracking parameters and run the tracker
model = YOLO('yolov8n.pt')
results = model.track(source="0", conf=0.3, iou=0.5, show=True)
```

Podemos ver que asigna un ID a cada objeto que encuentra:

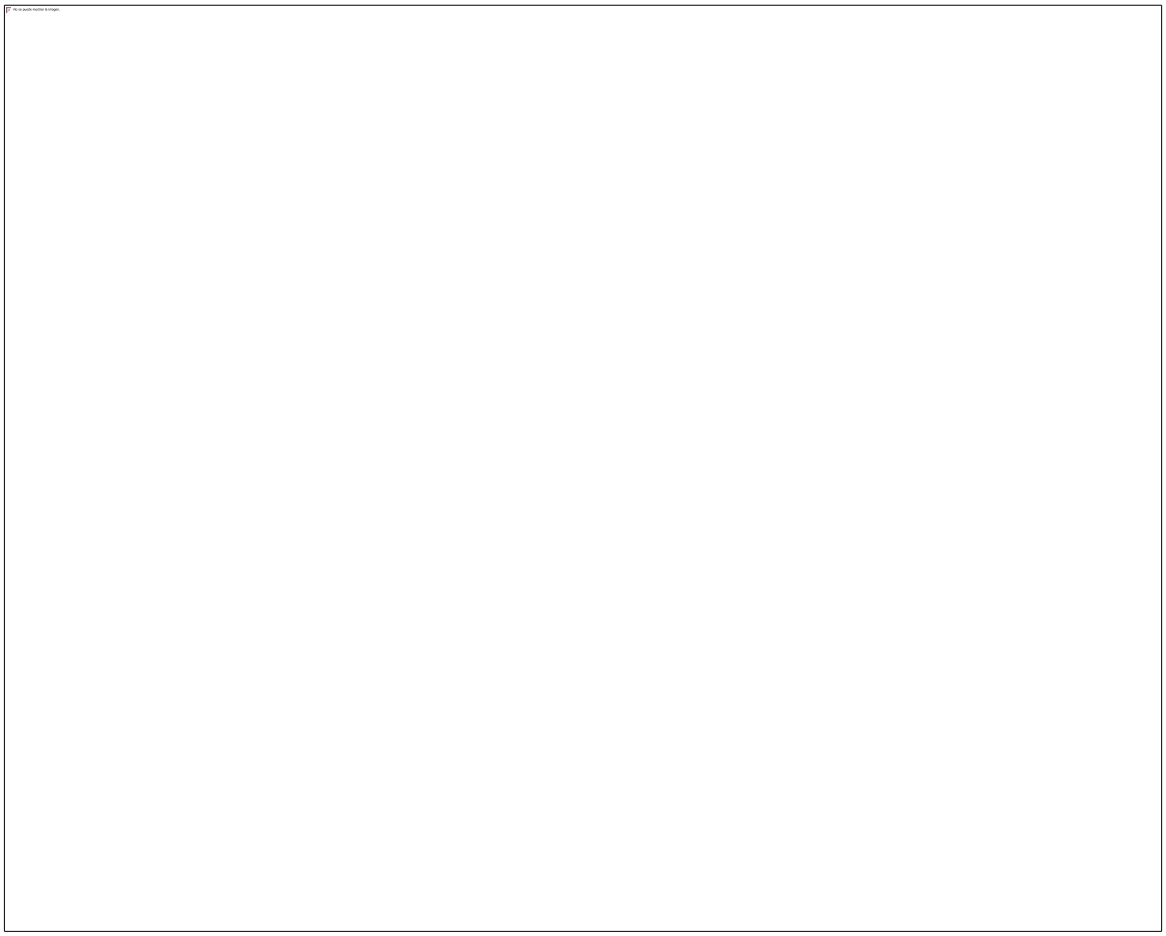


Modifica el script para que rastrease otro objeto de dicha lista

Utilizamos este código para trackear teclados:

```
# Tracking
from ultralytics import YOLO
# Configure the tracking parameters and run the tracker
model = YOLO('yolov8n.pt')
# Seleccionamos la clase 66 (keyboard)
results = model.track(source="0", conf=0.3, iou=0.5, show=True, classes=[66])
```

Aquí tenemos los resultados:



Script entrenado el modelo

Preparando el Dataset

El dataset lo he descargado de https://universe.roboflow.com/dice-cm3wy/dice-faces-2/dataset/1
He modificado el config.yaml y la estructura de las carpetas, quedando así:

La estructura del dataset es la siguiente:

```
datasets
├──images
│   ├──train
│   └──val
└──labels
  ├──train
  └──val
  └── config.yaml
```

Este es el contenido de config.yaml:

```
train: ../images/train # carpeta de imágenes de entrenamiento
val: ../images/val # carpeta de imágenes de validación

# Clases
names:
  0: "1" # Sólo la clase de nuestra etiqueta
  1: "2"
  2: "3"
  3: "4"
  4: "5"
  5: "6"
```

Entrenado el modelo en Google Colab

Con este fragmento del código, descomprimimos mi dataset:

```python
from sklearn.model_selection import train_test_split
import shutil
import os
import zipfile

dataset_ZIP_path = "datasets.zip"

dataset_path = dataset_ZIP_path[:-4]
if not os.path.exists(dataset_path):
    # Create a directory to store the dataset
    os.makedirs(dataset_path)
    # Unzip the dataset in the created directory

    with zipfile.ZipFile(dataset_ZIP_path, "r") as zip_ref:
        zip_ref.extractall(".")
```

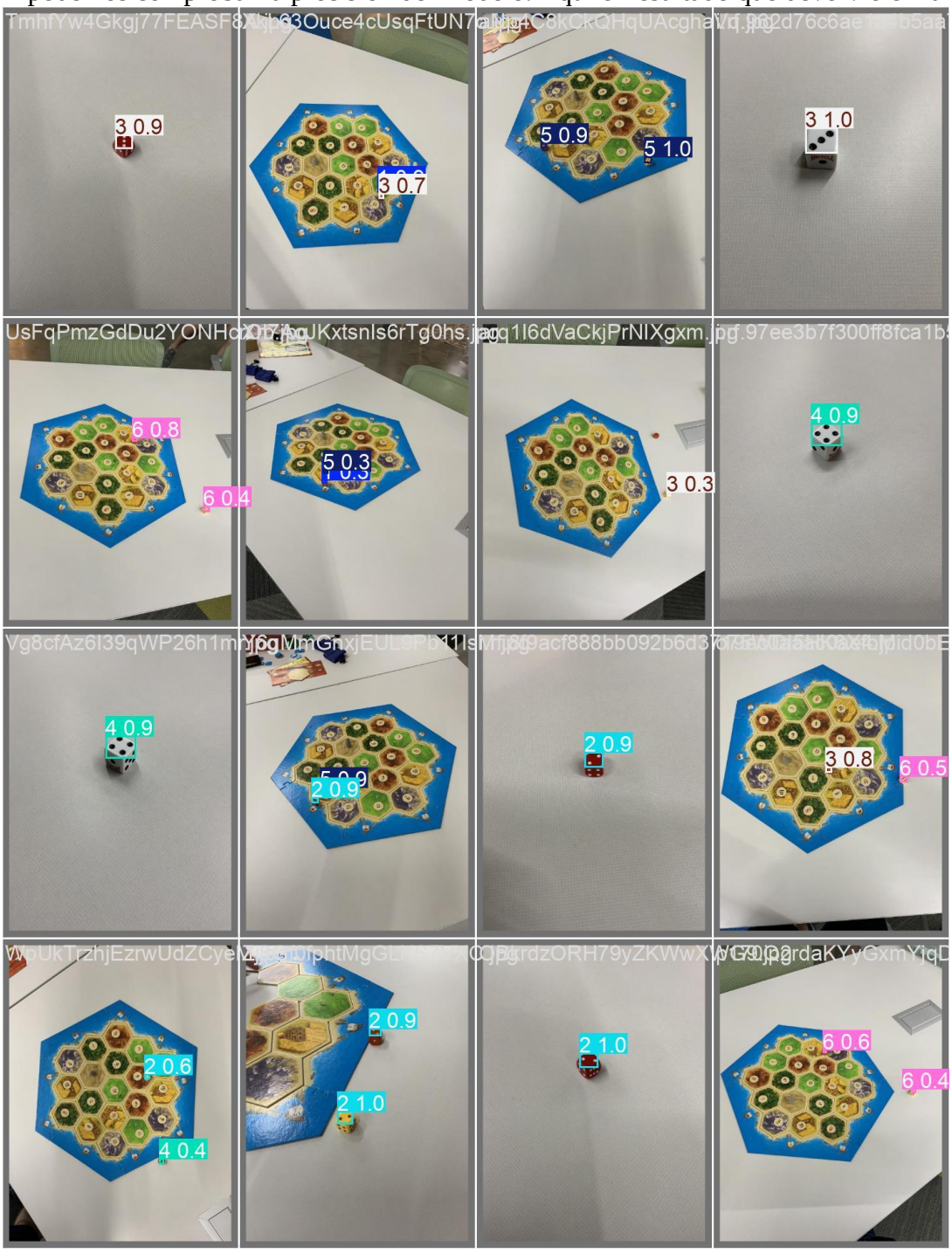
Y con este otro fragmento, utilizamos el dataset para entrenar al modelo:

```python
import zipfile
import os
from ultralytics import YOLO
# import torch
# Load a model
model = YOLO("yolov8n.pt")


# Train the model
train_results = model.train(
    data=dataset_ZIP_path[:-4] + "/config.yaml",  # path to dataset YAML
    epochs=50,  # number of training epochs
    imgsz=640,  # training image size
)

# Evaluate model performance on the validation set
metrics = model.val()
```

Aquí tenemos el resumen del entrenamiento:

```
Model summary (fused): 72 layers, 3,006,818 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/datasets/labels/val.cache... 180 images, 0 backgrounds, 0 corrupt: 100%|██████████| 180/180 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:03<00:00,  3.35it/s]
                   all        180        432      0.903      0.844      0.947      0.722
                     1         50         67      0.972      0.776      0.943      0.773
                     2         47         63      0.988       0.81      0.962       0.71
                     3         38         75      0.874       0.88      0.952      0.738
                     4         60         79      0.918      0.848      0.947      0.718
                     5         42         48      0.828        0.8      0.908      0.679
                     6         43        100      0.838       0.95      0.968      0.715
Speed: 3.9ms preprocess, 4.7ms inference, 0.0ms loss, 1.5ms postprocess per image
Results saved to runs/detect/train2
```

(Los resultados se guardaron en runs/detect/train2)
Y podemos comprobar la precisión del modelo. Aquí el resultado que devolvió en la fase de val_2:



Y aquí, lo que debería haber devuelto según las labels: