Fase 6: Custom Datase Black-Edition

Paso 45: Dataset Fine-tunning

¡Hola! ¡Qué buena iniciativa! Buscar un dataset de bolas de billar "black edition" (como los sets modernos y estilizados tipo Aramith Black) es una idea excelente para poner a prueba la robustez de tu modelo y adaptarlo a diferentes estéticas.

He realizado una búsqueda exhaustiva en las principales plataformas de datasets (Roboflow Universe, Kaggle, Hugging Face, etc.) y te confirmo lo que probablemente ya sospechabas: **no parece existir un dataset público ya etiquetado específicamente para bolas de billar "black edition"**.

¿Por qué no existe y por qué es una buena noticia para ti?

- **Nicho de Mercado:** Estos sets de bolas son un producto de nicho y relativamente moderno. La mayoría de los datasets de visión por computadora se crean para los objetos más comunes y estandarizados, en este caso, las bolas de billar tradicionales de colores vivos, ya que ofrecen un buen contraste y son universalmente reconocibles.
- **Oportunidad para tu Proyecto:** ¡Esto es una oportunidad fantástica! Crear tu propio dataset especializado es una de las tareas más importantes y formativas en un proyecto de Machine Learning. Demuestra una comprensión profunda del ciclo de vida de un proyecto de IA, desde la recolección de datos hasta el modelo final.

Plan de Acción: Creando tu Propio Dataset "Black Edition"

Dado que no podemos descargar uno ya hecho, te guiaré paso a paso para que construyas el tuyo. Este proceso te dará un control total sobre la calidad y variedad de tus datos.

Paso 1: La Recolección de Datos (Data Gathering)

Tu primera misión es conseguir imágenes y/o vídeos de estas bolas. Tienes varias opciones:

1. **Fotos y Vídeos Propios (La Mejor Opción):**
    - Si tienes acceso a uno de estos sets, esta es la mejor vía.
    - **Consejos para las fotos:**
        - **Variedad de Ángulos:** Toma fotos desde arriba, desde los lados, en perspectiva.

    - **Diferentes Condiciones de Luz:** Haz fotos con luz brillante, con sombras, con reflejos sobre las bolas. Esto es CRUCIAL para que tu modelo sea robusto.
    - **Distintos Estados de la Mesa:** Saca fotos con las bolas en el triángulo inicial, justo después del saque, a mitad de la partida, y con pocas bolas en la mesa.
    - **Calidad:** Usa la mayor resolución que puedas, aunque luego las redimensionemos para entrenar.

2. **Búsqueda de Imágenes y Vídeos Online:**
    - Es la alternativa más accesible. Utiliza buscadores de imágenes y plataformas de vídeo.
    - **Términos de Búsqueda Clave:**
        - `"Aramith Black pool set"`
        - `"Modern billiard balls"`
        - `"Custom pool balls"`
        - `"Black edition pool table"`
    - **Fuentes:**
        - **YouTube:** Busca vídeos de reseñas (`review`) o partidas (`gameplay`) con estos sets. Puedes extraer fotogramas de los vídeos.
        - **Instagram/Pinterest:** Busca con hashtags como `#custompooltable` o `#aramithblack`.
        - **Sitios de Fabricantes y Tiendas:** Webs como la de Aramith o distribuidores suelen tener galerías de imágenes de alta calidad.
    - ⚠️ **Aviso sobre Copyright:** Para un proyecto académico, usar estas imágenes suele considerarse "uso justo" (fair use). Sin embargo, si el proyecto fuera a ser comercial, necesitarías permiso o usar imágenes con licencias libres.

Paso 2: La Preparación y Etiquetado (Labeling)

Una vez que tengas una buena colección de imágenes (intenta conseguir al menos 100-200 imágenes variadas para empezar), es hora de etiquetarlas.

1. **Herramienta Recomendada:** Te sugiero **Roboflow**. Es una plataforma online que te facilita enormemente todo el proceso:
    - Puedes subir tus imágenes.
    - Tiene una interfaz muy intuitiva para dibujar las cajas delimitadoras (bounding boxes) sobre cada bola.

- Te permite gestionar las clases y exportar las anotaciones directamente en el **formato YOLO** que ya conoces, ahorrándote el script de conversión que hicimos.

2. **Proceso de Etiquetado:**
   - Sube tus imágenes a un nuevo proyecto en Roboflow.
   - Define tus clases. Aquí tendrás que tomar una decisión: ¿las llamarás `ball_black`, `ball_grey` o usarás los números si son visibles? Sé consistente. Por ejemplo, `black_edition_1`, `black_edition_2`, etc.
   - Dibuja una caja lo más ajustada posible alrededor de cada bola en cada imagen y asígnale su clase.
3. **Exportación:** Cuando termines de etiquetar, Roboflow te permitirá generar una versión de tu dataset, dividiéndolo automáticamente en `train`, `valid` y `test`, y te dará un archivo `.zip` con las imágenes y las etiquetas en formato YOLO, además del archivo `data.yaml` ya configurado.

Paso 3: El Entrenamiento (Training)

Ahora viene la parte más gratificante. Tienes un nuevo dataset de alta calidad, listo para usar.

1. **Transfer Learning:** ¡No empieces desde cero! Coge tu mejor modelo entrenado hasta ahora con las bolas de colores (`best.pt` de tu último entrenamiento). Este modelo ya sabe qué es una "bola", su forma esférica, cómo le afecta la luz, etc. Usarlo como punto de partida para entrenar con el nuevo dataset (esto se llama *fine-tuning*) será mucho más rápido y efectivo.
2. **Configura un Nuevo Entrenamiento:**
   - En tu script de entrenamiento, apunta al `data.yaml` de tu nuevo dataset "black edition".
   - Carga los pesos de tu mejor modelo anterior: `model = YOLO('./ruta/a/tu/anterior/best.pt')`
   - Dale un nuevo nombre a la ejecución para no sobreescribir tus resultados: `name='detect_balls_black_edition_v1'`

Resumen del Plan de Acción

1. **Recolecta** un mínimo de 100-200 imágenes de sets de billar "black edition".
2. **Regístrate** en Roboflow, crea un proyecto y sube tus imágenes.
3. **Etiqueta** las bolas pacientemente, definiendo clases consistentes.
4. **Exporta** tu dataset en formato YOLO.

5. **Entrena** un nuevo modelo partiendo de tu `best.pt` anterior, pero con este nuevo dataset.

Este es un paso excelente y muy avanzado para tu proyecto. Demostrará tu capacidad para manejar todo el flujo de trabajo de la IA y te dará un modelo final mucho más versátil y profesional.

¡Adelante y a por esas imágenes! Estoy aquí para cuando las tengas listas y quieras pasar al siguiente paso.

Paso 46: Pre-etiquetado

¡Absolutamente! Es una idea excelente y, de hecho, es una técnica muy profesional para acelerar el proceso de etiquetado. No podemos crear un script que etiquete desde cero con una precisión perfecta, pero lo que sí podemos hacer es un **script de "pre-etiquetado" o etiquetado asistido**.

La idea es brillante y se basa en algo que ya tienes: **un modelo que, aunque entrenado para bolas de colores, ya sabe lo que es una "bola"**.

El Concepto: Usar tu IA Actual como un "Asistente Junior"

Imagina que tu modelo entrenado (`best.pt`) es un asistente al que le pides una primera pasada sobre las nuevas imágenes de las bolas "black edition".

1. **El Asistente Hace el Trabajo Pesado:** El modelo analizará las nuevas imágenes y, basándose en la forma, los reflejos y las sombras, intentará detectar todo lo que parezca una bola.
2. **Puede que se Equivoque:** Como no conoce estos colores, es muy probable que cometa errores:
   - **Clasificación Incorrecta:** Podría ver una bola gris y etiquetarla como `white` o `black_8`.
   - **Detecciones Fallidas:** Puede que no detecte algunas bolas que no se parezcan a nada que haya visto antes.
   - **Detecciones Correctas:** ¡Pero también es muy probable que detecte muchas bolas correctamente! Su conocimiento de la forma esférica es muy potente.
3. **Tú Haces el Control de Calidad:** Tu trabajo ya no será dibujar cada caja desde cero. Será mucho más rápido:
   - **Corregir etiquetas:** Cambiar una clase incorrecta (ej. de `white` a `grey_ball`).
   - **Ajustar cajas:** Mover o redimensionar una caja que no esté perfecta.
   - **Eliminar detecciones erróneas.**

- **Añadir las pocas cajas que el modelo no detectó.**

Este método puede ahorrarte entre un 60% y un 80% del tiempo de etiquetado.

Script de Pre-Etiquetado en Python

Aquí tienes un script en Python que puedes usar. Este script cargará tu mejor modelo actual, procesará una carpeta con tus nuevas imágenes "black edition" y generará los archivos de etiquetas `.txt` en formato YOLO como "borrador".

```
# pre_etiquetado.py
import os
from ultralytics import YOLO
import cv2 # Se usa para obtener las dimensiones de la imagen

# --- 1. CONFIGURACIÓN ---

# Ruta a tu mejor modelo entrenado hasta ahora
MODELO_PATH = './detect_balls/runs/detect_balls_v012/weights/best.pt'

# Carpeta donde tienes TODAS tus nuevas imágenes "black edition" sin etiquetar
IMAGENES_NUEVAS_DIR = './detect_balls/datasets/black_edition_raw/images/'

# Carpeta donde se guardarán las etiquetas generadas por la IA
ETIQUETAS_GENERADAS_DIR = './detect_balls/datasets/black_edition_raw/labels/'

# Umbral de confianza: Solo guardaremos las detecciones que superen esta confianza
# Empieza con un valor bajo (ej. 0.25) para capturar más propuestas, aunque algunas s
UMBRAL_CONFIANZA = 0.25

# --- FIN DE LA CONFIGURACIÓN ---

# Crear el directorio de salida si no existe
os.makedirs(ETIQUETAS_GENERADAS_DIR, exist_ok=True)

# Cargar el modelo YOLO entrenado
try:
    model = YOLO(MODELO_PATH)
    print(f"Modelo cargado exitosamente desde: {MODELO_PATH}")
except Exception as e:
    print(f"Error al cargar el modelo: {e}")
    exit()

# Listar todas las imágenes en el directorio de entrada
```

```
try:
    lista_imagenes = [f for f in os.listdir(IMAGENES_NUEVAS_DIR) if f.lower().endswith(('
    if not lista_imagenes:
        print(f"No se encontraron imágenes en: {IMAGENES_NUEVAS_DIR}")
        exit()
    print(f"Se encontraron {len(lista_imagenes)} imágenes para pre-etiquetar.")
except FileNotFoundError:
    print(f"Error: El directorio de imágenes no existe: {IMAGENES_NUEVAS_DIR}")
    exit()


# Procesar cada imagen
for nombre_imagen in lista_imagenes:
    ruta_imagen = os.path.join(IMAGENES_NUEVAS_DIR, nombre_imagen)
    print(f"\nProcesando imagen: {nombre_imagen}...")

    # Realizar la predicción
    # verbose=False para una salida más limpia en la consola
    results = model.predict(ruta_imagen, conf=UMBRAL_CONFIANZA, verbose=False)

    # El resultado es una lista, tomamos el primer elemento para una sola imagen
    result = results[0]

    # Crear el nombre del archivo de etiqueta
    nombre_base = os.path.splitext(nombre_imagen)[0]
    ruta_etiqueta = os.path.join(ETIQUETAS_GENERADAS_DIR, f"{nombre_base}.txt")

    # Abrir el archivo de etiqueta en modo escritura (sobrescribe si existe)
    with open(ruta_etiqueta, 'w') as f_etiqueta:
        if result.boxes:
            print(f"  > Se detectaron {len(result.boxes)} objetos.")
            # Iterar sobre cada detección
            for box in result.boxes:
                # Obtener coordenadas normalizadas en formato (x_center, y_center, width, h
                coords_xywhn = box.xywhn[0]

                # Obtener el ID de la clase
                # ¡OJO! El modelo predecirá clases del 0 al 15 (de tu entrenamiento anterior)
                # Probablemente necesites corregir esto manualmente.
                id_clase = int(box.cls[0])

                # Formatear la línea para el archivo YOLO
                # Formato: <class_id> <x_center> <y_center> <width> <height>
```

```
            linea = f"{id_clase} {coords_xywhn[0]:.6f} {coords_xywhn[1]:.6f} {coords_xyw

            # Escribir la línea en el archivo
            f_etiqueta.write(linea)
        else:
            print("  > No se detectó ningún objeto con la confianza suficiente.")

print("\n¡Proceso de pre-etiquetado completado!")
print(f"Puedes encontrar las etiquetas generadas en: {ETIQUETAS_GENERADAS_DIR}")
```

Plan de Acción Detallado

1. **Organiza tus Archivos:**
   - Crea una carpeta, por ejemplo, `detect_balls/datasets/black_edition_raw/images/`.
   - Copia **todas** tus nuevas imágenes de bolas "black edition" en esa carpeta.
2. **Adapta y Ejecuta el Script:**
   - Guarda el código anterior en un archivo llamado `pre_etiquetado.py` en la raíz de tu proyecto.
   - Asegúrate de que las rutas (`MODELO_PATH`, `IMAGENES_NUEVAS_DIR`, `ETIQUETAS_GENERADAS_DIR`) en el script sean correctas para tu estructura de carpetas.
   - Ejecuta el script desde tu terminal con el entorno activado: `python ./detect_balls/pre_etiquetado.py`.
3. **Revisa y Corrige (El Paso Crucial):**
   - Ahora tendrás la carpeta `detect_balls/datasets/black_edition_raw/labels/` llena de archivos `.txt`.
   - **Usa una Herramienta de Etiquetado:** Este es el momento perfecto para usar una herramienta como **Roboflow**, **CVAT** o **LabelImg**.
   - **Flujo de trabajo en Roboflow:**
     - Crea un nuevo proyecto en Roboflow.
     - Sube la carpeta que contiene tanto las **imágenes** (`images`) como las **etiquetas** (`labels`) que acabas de generar.
     - ¡Magia! Roboflow leerá los `.txt` y te mostrará las imágenes con las cajas de detección ya dibujadas.
     - Ahora solo tienes que recorrer las imágenes y hacer las correcciones: ajustar cajas, cambiar clases (muy importante) y añadir las que falten.
4. **Exporta y Entrena:**

- Una vez que hayas corregido todas las etiquetas, usa la función de Roboflow para generar y exportar el dataset en formato YOLO, ya dividido en `train/valid/test`.
- ¡Ya tienes un dataset de alta calidad listo para entrenar tu modelo "black edition"!

¡Estás a punto de acelerar tu trabajo de forma masiva! Este enfoque es una demostración de cómo usar la IA para ayudarse a sí misma a mejorar. ¡Adelante!

Resultados Pre-etiquetado

(.venv) oscar@Pepon:~/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src$ /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/bin/python /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/detect_balls/pre-labeling.py
Modelo cargado exitosamente desde: ./detect_balls/runs/detect_balls_v12/weights/best.pt
Se encontraron 69 imágenes para pre-etiquetar.

Procesando imagen: s-l400.jpg...

> Se detectaron 17 objetos:
> - Propuesta: green_6 (ID: 6, Conf: 0.89)
> - Propuesta: yellow_9 (ID: 15, Conf: 0.89)
> - Propuesta: black_8 (ID: 0, Conf: 0.87)
> - Propuesta: blue_10 (ID: 1, Conf: 0.87)
> - Propuesta: green_14 (ID: 5, Conf: 0.84)
> - Propuesta: blue_10 (ID: 1, Conf: 0.83)
> - Propuesta: red_11 (ID: 11, Conf: 0.78)
> - Propuesta: blue_10 (ID: 1, Conf: 0.71)
> - Propuesta: black_8 (ID: 0, Conf: 0.69)
> - Propuesta: orange_13 (ID: 7, Conf: 0.67)
> - Propuesta: yellow_1 (ID: 14, Conf: 0.66)
> - Propuesta: dred_7 (ID: 4, Conf: 0.66)
> - Propuesta: purple_4 (ID: 10, Conf: 0.55)
> - Propuesta: black_8 (ID: 0, Conf: 0.53)

> - Propuesta: blue_10 (ID: 1, Conf: 0.48)
> - Propuesta: blue_10 (ID: 1, Conf: 0.42)
> - Propuesta: orange_13 (ID: 7, Conf: 0.38)

Procesando imagen: 2025-06-15 20-27-04.png...

> Se detectaron 4 objetos:
> - Propuesta: red_3 (ID: 12, Conf: 0.63)
> - Propuesta: black_8 (ID: 0, Conf: 0.48)
> - Propuesta: black_8 (ID: 0, Conf: 0.34)
> - Propuesta: black_8 (ID: 0, Conf: 0.27)

... resto de imagenes ...

Procesando imagen: 2025-06-15 20-03-49.png...

> Se detectaron 3 objetos:
> - Propuesta: black_8 (ID: 0, Conf: 0.65)
> - Propuesta: white (ID: 13, Conf: 0.64)
> - Propuesta: dred_7 (ID: 4, Conf: 0.57)

Procesando imagen: 2025-06-15 20-24-46.png...

> Se detectaron 2 objetos:
> - Propuesta: black_8 (ID: 0, Conf: 0.57)
> - Propuesta: orange_5 (ID: 8, Conf: 0.41)

¡Proceso de pre-etiquetado completado!
Las etiquetas generadas (con los IDs correctos) están en:
./detect_balls/datasets/black_edition_raw/labels/

Paso 47: Instalar label-studio para Etiquetado

```
# Install the package
# into python virtual environment
```

```
pip install -U label-studio

# Launch it!
label-studio
```

```
pip install -U label-studio
```

```
Collecting label-studio
Downloading label_studio-1.19.0-py3-none-any.whl.metadata (14 kB)
Collecting Django<5.2.0,>=5.1.8 (from label-studio)
Downloading django-5.1.11-py3-none-any.whl.metadata (4.1 kB)
Collecting appdirs>=1.4.3 (from label-studio)
Downloading appdirs-1.4.4-py2.py3-none-any.whl.metadata (9.0 kB)
Collecting attr==0.3.1 (from label-studio)
Downloading attr-0.3.1.tar.gz (1.7 kB)
Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>=19.2.0 in ./.venv/lib/python3.12/site-packages (fr
Collecting azure-storage-blob>=12.6.0 (from label-studio)
Downloading azure_storage_blob-12.25.1-py3-none-any.whl.metadata (26 kB)
Collecting bleach<5.1.0,>=5.0.0 (from label-studio)
Downloading bleach-5.0.1-py3-none-any.whl.metadata (27 kB)
Collecting boto<3.0.0,>=2.49.0 (from label-studio)
Downloading boto-2.49.0-py2.py3-none-any.whl.metadata (7.3 kB)
Collecting boto3<2.0.0,>=1.28.58 (from label-studio)
Downloading boto3-1.38.38-py3-none-any.whl.metadata (6.6 kB)
Collecting botocore<2.0.0,>=1.31.58 (from label-studio)
Downloading botocore-1.38.38-py3-none-any.whl.metadata (5.7 kB)
Collecting colorama>=0.4.4 (from label-studio)
Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting cryptography>=44.0.1 (from label-studio)
Downloading cryptography-45.0.4-cp311-abi3-manylinux_2_34_x86_64.whl.metadata (
Collecting defusedxml>=0.7.1 (from label-studio)
Downloading defusedxml-0.7.1-py2.py3-none-any.whl.metadata (32 kB)
Collecting django-annoying==0.10.6 (from label-studio)
Downloading django_annoying-0.10.6-py2.py3-none-any.whl.metadata (2.3 kB)
Collecting django-cors-headers==4.7.0 (from label-studio)
Downloading django_cors_headers-4.7.0-py3-none-any.whl.metadata (16 kB)
Collecting django-csp==3.7 (from label-studio)
Downloading django_csp-3.7-py2.py3-none-any.whl.metadata (2.7 kB)
Collecting django-debug-toolbar==3.2.1 (from label-studio)
```

```
Downloading django_debug_toolbar-3.2.1-py3-none-any.whl.metadata (3.2 kB)
Collecting django-environ==0.10.0 (from label-studio)
Downloading django_environ-0.10.0-py2.py3-none-any.whl.metadata (13 kB)
Collecting django-extensions==3.2.3 (from label-studio)
Downloading django_extensions-3.2.3-py3-none-any.whl.metadata (6.3 kB)
Collecting django-filter==24.3 (from label-studio)
Downloading django_filter-24.3-py3-none-any.whl.metadata (5.2 kB)
Collecting django-migration-linter<6.0.0,>=5.1.0 (from label-studio)
Downloading django_migration_linter-5.2.0-py3-none-any.whl.metadata (6.4 kB)
Collecting django-model-utils==4.1.1 (from label-studio)
Downloading django_model_utils-4.1.1-py3-none-any.whl.metadata (17 kB)
Collecting django-ranged-fileresponse>=0.1.2 (from label-studio)
Downloading django-ranged-fileresponse-0.1.2.tar.gz (2.7 kB)
Preparing metadata (setup.py) ... done
Collecting django-rq<3.0.0,>=2.10.2 (from label-studio)
Downloading django_rq-2.10.3-py2.py3-none-any.whl.metadata (18 kB)
Collecting django-storages==1.12.3 (from label-studio)
Downloading django_storages-1.12.3-py3-none-any.whl.metadata (54 kB)
Collecting django-user-agents==0.4.0 (from label-studio)
Downloading django_user_agents-0.4.0-py3-none-any.whl.metadata (6.2 kB)
Collecting djangorestframework==3.15.2 (from label-studio)
Downloading djangorestframework-3.15.2-py3-none-any.whl.metadata (10 kB)
Collecting djangorestframework-simplejwt<6.0.0,>=5.4.0 (from djangorestframework-s
Downloading djangorestframework_simplejwt-5.5.0-py3-none-any.whl.metadata (4.6 k
Collecting drf-dynamic-fields==0.3.0 (from label-studio)
Downloading drf_dynamic_fields-0.3.0-py2.py3-none-any.whl.metadata (4.8 kB)
Collecting drf-flex-fields==0.9.5 (from label-studio)
Downloading drf-flex-fields-0.9.5.tar.gz (27 kB)
Preparing metadata (setup.py) ... done
Collecting drf-generators==0.3.0 (from label-studio)
Downloading drf-generators-0.3.0.tar.gz (9.4 kB)
Preparing metadata (setup.py) ... done
Collecting google-cloud-logging<4.0.0,>=3.10.0 (from label-studio)
Downloading google_cloud_logging-3.12.1-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting google-cloud-storage<3.0.0,>=2.13.0 (from label-studio)
Downloading google_cloud_storage-2.19.0-py2.py3-none-any.whl.metadata (9.1 kB)
Collecting humansignal-drf-yasg>=1.21.10.post1 (from label-studio)
Downloading humansignal_drf_yasg-1.21.10.post1-py3-none-any.whl.metadata (16 kB)
Collecting label-studio-sdk==1.0.16 (from label-studio)
Downloading label_studio_sdk-1.0.16-py3-none-any.whl.metadata (6.0 kB)
Collecting launchdarkly-server-sdk==8.2.1 (from label-studio)
Downloading launchdarkly_server_sdk-8.2.1-py3-none-any.whl.metadata (1.3 kB)
Collecting lockfile>=0.12.0 (from label-studio)
```

```
Downloading lockfile-0.12.2-py2.py3-none-any.whl.metadata (2.4 kB)
Collecting lxml>=4.9.4 (from lxml[html-clean]>=4.9.4→label-studio)
Downloading lxml-5.4.0-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (3.5 kB)
Requirement already satisfied: numpy<2.0.0,>=1.26.4 in ./.venv/lib/python3.12/site-pac
Requirement already satisfied: openai<2.0.0,>=1.10.0 in ./.venv/lib/python3.12/site-pack
Collecting ordered-set==4.0.2 (from label-studio)
Downloading ordered-set-4.0.2.tar.gz (10 kB)
Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas>=2.2.3 in ./.venv/lib/python3.12/site-packages (
Collecting psycopg2-binary==2.9.10 (from label-studio)
Downloading psycopg2_binary-2.9.10-cp312-cp312-manylinux_2_17_x86_64.manylinux2
Collecting pyboxen>=1.3.0 (from label-studio)
Downloading pyboxen-1.3.0-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: pydantic>=2.9.2 in ./.venv/lib/python3.12/site-packages
Requirement already satisfied: python-dateutil>=2.8.1 in ./.venv/lib/python3.12/site-pac
Collecting python-json-logger==2.0.4 (from label-studio)
Downloading python_json_logger-2.0.4-py3-none-any.whl.metadata (6.5 kB)
Collecting pytz<2023.0,>=2022.1 (from label-studio)
Downloading pytz-2022.7.1-py2.py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: pyyaml>=6.0.0 in ./.venv/lib/python3.12/site-packages (
Collecting redis<5.3.0,>=5.2.1 (from label-studio)
Downloading redis-5.2.1-py3-none-any.whl.metadata (9.1 kB)
Requirement already satisfied: requests<2.33.0,>=2.32.3 in ./.venv/lib/python3.12/site-
Collecting rq<2.0.0,>=1.16.2 (from label-studio)
Downloading rq-1.16.2-py3-none-any.whl.metadata (5.7 kB)
Collecting rules==3.4 (from label-studio)
Downloading rules-3.4-py2.py3-none-any.whl.metadata (38 kB)
Collecting sentry-sdk>=2.16.0 (from label-studio)
Downloading sentry_sdk-2.30.0-py2.py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: setuptools>=75.4.0 in ./.venv/lib/python3.12/site-packag
Collecting tldextract>=5.1.3 (from label-studio)
Downloading tldextract-5.3.0-py3-none-any.whl.metadata (11 kB)
Collecting ujson>=3.0.0 (from label-studio)
Downloading ujson-5.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64
Collecting urllib3<2.0.0,>=1.26.18 (from label-studio)
Downloading urllib3-1.26.20-py2.py3-none-any.whl.metadata (50 kB)
Collecting wheel<=0.40.0,>=0.38.1 (from label-studio)
Downloading wheel-0.40.0-py3-none-any.whl.metadata (2.1 kB)
Collecting xmljson==0.2.1 (from label-studio)
Downloading xmljson-0.2.1-py2.py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: six in ./.venv/lib/python3.12/site-packages (from django
Collecting asgiref>=3.6 (from django-cors-headers==4.7.0→label-studio)
Downloading asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
```

```
Collecting sqlparse>=0.2.0 (from django-debug-toolbar==3.2.1→label-studio)
Downloading sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Collecting user-agents (from django-user-agents==0.4.0→label-studio)
Downloading user_agents-2.2.0-py3-none-any.whl.metadata (7.9 kB)
Requirement already satisfied: Pillow>=10.0.1 in ./.venv/lib/python3.12/site-packages (fr
Collecting datamodel-code-generator==0.26.1 (from label-studio-sdk==1.0.16→label-st
Downloading datamodel_code_generator-0.26.1-py3-none-any.whl.metadata (24 kB)
Requirement already satisfied: httpx>=0.21.2 in ./.venv/lib/python3.12/site-packages (fr
Collecting ijson>=3.2.3 (from label-studio-sdk==1.0.16→label-studio)
Downloading ijson-3.4.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.
Collecting jsf<0.12.0,>=0.11.2 (from label-studio-sdk==1.0.16→label-studio)
Downloading jsf-0.11.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: jsonschema>=4.23.0 in ./.venv/lib/python3.12/site-pack
Collecting nltk<4.0.0,>=3.9.1 (from label-studio-sdk==1.0.16→label-studio)
Downloading nltk-3.9.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: opencv-python<5.0.0,>=4.9.0 in ./.venv/lib/python3.12/s
Requirement already satisfied: pydantic-core<3.0.0,>=2.18.2 in ./.venv/lib/python3.12/s
Collecting pyjwt<3.0.0,>=2.10.1 (from label-studio-sdk==1.0.16→label-studio)
Downloading PyJWT-2.10.1-py3-none-any.whl.metadata (4.0 kB)
Collecting requests-mock==1.12.1 (from label-studio-sdk==1.0.16→label-studio)
Downloading requests_mock-1.12.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: typing_extensions>=4.0.0 in ./.venv/lib/python3.12/site-p
Collecting argcomplete<4.0,>=1.10 (from datamodel-code-generator==0.26.1→label-st
Downloading argcomplete-3.6.2-py3-none-any.whl.metadata (16 kB)
Collecting black>=19.10b0 (from datamodel-code-generator==0.26.1→label-studio-sdk
Downloading black-25.1.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64
Collecting genson<2.0,>=1.2.1 (from datamodel-code-generator==0.26.1→label-studio
Downloading genson-1.3.0-py3-none-any.whl.metadata (28 kB)
Collecting inflect<6.0,>=4.1.0 (from datamodel-code-generator==0.26.1→label-studio-
Downloading inflect-5.6.2-py3-none-any.whl.metadata (21 kB)
Collecting isort<6.0,>=4.3.21 (from datamodel-code-generator==0.26.1→label-studio-s
Downloading isort-5.13.2-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: jinja2<4.0,>=2.10.1 in ./.venv/lib/python3.12/site-package
Requirement already satisfied: packaging in ./.venv/lib/python3.12/site-packages (from
Requirement already satisfied: certifi>=2018.4.16 in ./.venv/lib/python3.12/site-package
Collecting expiringdict>=1.1.4 (from launchdarkly-server-sdk==8.2.1→label-studio)
Downloading expiringdict-1.2.2-py3-none-any.whl.metadata (3.7 kB)
Collecting pyRFC3339>=1.0 (from launchdarkly-server-sdk==8.2.1→label-studio)
Downloading pyRFC3339-2.0.1-py3-none-any.whl.metadata (2.1 kB)
Collecting semver>=2.10.2 (from launchdarkly-server-sdk==8.2.1→label-studio)
Downloading semver-3.0.4-py3-none-any.whl.metadata (6.8 kB)
Collecting webencodings (from bleach<5.1.0,>=5.0.0→label-studio)
Downloading webencodings-0.5.1-py2.py3-none-any.whl.metadata (2.1 kB)
```

```
Collecting jmespath<2.0.0,>=0.7.1 (from boto3<2.0.0,>=1.28.58→label-studio)
Downloading jmespath-1.0.1-py3-none-any.whl.metadata (7.6 kB)
Collecting s3transfer<0.14.0,>=0.13.0 (from boto3<2.0.0,>=1.28.58→label-studio)
Downloading s3transfer-0.13.0-py3-none-any.whl.metadata (1.7 kB)
Collecting toml>=0.10.2 (from django-migration-linter<6.0.0,>=5.1.0→label-studio)
Downloading toml-0.10.2-py2.py3-none-any.whl.metadata (7.1 kB)
INFO: pip is looking at multiple versions of djangorestframework-simplejwt to determin
Collecting djangorestframework-simplejwt<6.0.0,>=5.4.0 (from djangorestframework-s
Downloading djangorestframework_simplejwt-5.4.0-py3-none-any.whl.metadata (4.3 k
INFO: pip is looking at multiple versions of djangorestframework-simplejwt[crypto] to d
Requirement already satisfied: google-api-core!=2.0.,!=2.1.,!=2.10.,!=2.2.,!=2.3.,!=2.4.,!
Requirement already satisfied: google-auth!=2.24.0,!=2.25.0,<3.0.0,>=2.14.1 in ./.venv/l
Collecting google-cloud-appengine-logging<2.0.0,>=0.1.3 (from google-cloud-logging
Downloading google_cloud_appengine_logging-1.6.2-py3-none-any.whl.metadata (9.9
Collecting google-cloud-audit-log<1.0.0,>=0.3.1 (from google-cloud-logging<4.0.0,>=3
Downloading google_cloud_audit_log-0.3.2-py3-none-any.whl.metadata (9.0 kB)
Collecting google-cloud-core<3.0.0,>=2.0.0 (from google-cloud-logging<4.0.0,>=3.10
Downloading google_cloud_core-2.4.3-py2.py3-none-any.whl.metadata (2.7 kB)
Collecting grpc-google-iam-v1<1.0.0,>=0.12.4 (from google-cloud-logging<4.0.0,>=3.1
Downloading grpc_google_iam_v1-0.14.2-py3-none-any.whl.metadata (9.1 kB)
Collecting opentelemetry-api>=1.9.0 (from google-cloud-logging<4.0.0,>=3.10.0→labe
Downloading opentelemetry_api-1.34.1-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: proto-plus<2.0.0,>=1.22.0 in ./.venv/lib/python3.12/site-
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.2
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in ./.venv/lib
Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in ./.venv/lib/python3.12/site-pa
Requirement already satisfied: grpcio-status<2.0.dev0,>=1.33.2 in ./.venv/lib/python3.1
Requirement already satisfied: cachetools<6.0,>=2.0.0 in ./.venv/lib/python3.12/site-pa
Requirement already satisfied: pyasn1-modules>=0.2.1 in ./.venv/lib/python3.12/site-pa
Requirement already satisfied: rsa<5,>=3.1.4 in ./.venv/lib/python3.12/site-packages (fr
Collecting google-resumable-media>=2.7.2 (from google-cloud-storage<3.0.0,>=2.13.
Downloading google_resumable_media-2.7.2-py2.py3-none-any.whl.metadata (2.2 kB)
Collecting google-crc32c<2.0dev,>=1.0 (from google-cloud-storage<3.0.0,>=2.13.0→l
Downloading google_crc32c-1.7.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014
Collecting protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<7.0.0,>=3.20.2
Downloading protobuf-5.29.5-cp38-abi3-manylinux2014_x86_64.whl.metadata (592 by
Requirement already satisfied: MarkupSafe>=2.0 in ./.venv/lib/python3.12/site-package
Collecting faker>=15.3.4 (from jsf<0.12.0,>=0.11.2→label-studio-sdk==1.0.16→label-stu
Downloading faker-37.4.0-py3-none-any.whl.metadata (15 kB)
Collecting rstr>=3.2.0 (from jsf<0.12.0,>=0.11.2→label-studio-sdk==1.0.16→label-studio
Downloading rstr-3.2.2-py3-none-any.whl.metadata (7.1 kB)
Collecting smart-open>=6.3.0 (from smart-open[http]>=6.3.0→jsf<0.12.0,>=0.11.2→lab
Downloading smart_open-7.1.0-py3-none-any.whl.metadata (24 kB)
```

```
Requirement already satisfied: click in ./.venv/lib/python3.12/site-packages (from nltk<
Collecting joblib (from nltk<4.0.0,>=3.9.1→label-studio-sdk==1.0.16→label-studio)
Using cached joblib-1.5.1-py3-none-any.whl.metadata (5.6 kB)
Requirement already satisfied: regex>=2021.8.3 in ./.venv/lib/python3.12/site-packages
Requirement already satisfied: tqdm in ./.venv/lib/python3.12/site-packages (from nltk<
Requirement already satisfied: anyio<5,>=3.5.0 in ./.venv/lib/python3.12/site-packages
Requirement already satisfied: distro<2,>=1.7.0 in ./.venv/lib/python3.12/site-packages
Requirement already satisfied: jiter<1,>=0.4.0 in ./.venv/lib/python3.12/site-packages (f
Requirement already satisfied: sniffio in ./.venv/lib/python3.12/site-packages (from ope
Requirement already satisfied: idna>=2.8 in ./.venv/lib/python3.12/site-packages (from
Requirement already satisfied: httpcore==1.* in ./.venv/lib/python3.12/site-packages (fr
Requirement already satisfied: h11>=0.16 in ./.venv/lib/python3.12/site-packages (from
Requirement already satisfied: annotated-types>=0.6.0 in ./.venv/lib/python3.12/site-pa
Requirement already satisfied: typing-inspection>=0.4.0 in ./.venv/lib/python3.12/site-p
Collecting email-validator>=2.0.0 (from pydantic[email]!=2.0.0,!=2.0.1,!=2.4.0,<3.0,>=1
Downloading email_validator-2.2.0-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in ./.venv/lib/python3.12/site-
Requirement already satisfied: pyasn1>=0.1.3 in ./.venv/lib/python3.12/site-packages (fr
Collecting azure-core>=1.30.0 (from azure-storage-blob>=12.6.0→label-studio)
Downloading azure_core-1.34.0-py3-none-any.whl.metadata (42 kB)
Collecting isodate>=0.6.1 (from azure-storage-blob>=12.6.0→label-studio)
Downloading isodate-0.7.2-py3-none-any.whl.metadata (11 kB)
Collecting mypy-extensions>=0.4.3 (from black>=19.10b0→datamodel-code-generato
Downloading mypy_extensions-1.1.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: pathspec>=0.9.0 in ./.venv/lib/python3.12/site-packages
Collecting platformdirs>=2 (from black>=19.10b0→datamodel-code-generator==0.26.1
Downloading platformdirs-4.3.8-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: cffi>=1.14 in ./.venv/lib/python3.12/site-packages (from
Requirement already satisfied: pycparser in ./.venv/lib/python3.12/site-packages (from
Collecting dnspython>=2.0.0 (from email-validator>=2.0.0→pydantic[email]!=2.0.0,!=2
Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: tzdata in ./.venv/lib/python3.12/site-packages (from fake
Collecting inflection>=0.3.1 (from humansignal-drf-yasg>=1.21.10.post1→label-studio)
Downloading inflection-0.5.1-py2.py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: uritemplate>=3.0.0 in ./.venv/lib/python3.12/site-packag
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in ./.venv/lib/py
Requirement already satisfied: referencing>=0.28.4 in ./.venv/lib/python3.12/site-packa
Requirement already satisfied: rpds-py>=0.7.1 in ./.venv/lib/python3.12/site-packages (
Collecting lxml_html_clean (from lxml[html-clean]>=4.9.4→label-studio)
Downloading lxml_html_clean-0.4.2-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: importlib-metadata<8.8.0,>=6.0 in ./.venv/lib/python3.12
Requirement already satisfied: zipp>=0.5 in ./.venv/lib/python3.12/site-packages (from
Requirement already satisfied: rich>=12.5.1 in ./.venv/lib/python3.12/site-packages (fro
```

```
Requirement already satisfied: markdown-it-py>=2.2.0 in ./.venv/lib/python3.12/site-pa
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in ./.venv/lib/python3.12/site-p
Requirement already satisfied: mdurl~=0.1 in ./.venv/lib/python3.12/site-packages (from
Requirement already satisfied: wrapt in ./.venv/lib/python3.12/site-packages (from sma
Collecting requests-file>=1.4 (from tldextract>=5.1.3→label-studio)
Downloading requests_file-2.1.0-py2.py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: filelock>=3.0.8 in ./.venv/lib/python3.12/site-packages (
Collecting ua-parser>=0.10.0 (from user-agents→django-user-agents==0.4.0→label-s
Downloading ua_parser-1.0.1-py3-none-any.whl.metadata (5.6 kB)
Collecting ua-parser-builtins (from ua-parser>=0.10.0→user-agents→django-user-age
Downloading ua_parser_builtins-0.18.0.post1-py3-none-any.whl.metadata (1.4 kB)
Downloading label_studio-1.19.0-py3-none-any.whl (65.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.4/65.4 MB 77.7 MB/s eta 0:00:0
Downloading django_annoying-0.10.6-py2.py3-none-any.whl (9.5 kB)
Downloading django_cors_headers-4.7.0-py3-none-any.whl (12 kB)
Downloading django_csp-3.7-py2.py3-none-any.whl (17 kB)
Downloading django_debug_toolbar-3.2.1-py3-none-any.whl (199 kB)
Downloading django_environ-0.10.0-py2.py3-none-any.whl (19 kB)
Downloading django_extensions-3.2.3-py3-none-any.whl (229 kB)
Downloading django_filter-24.3-py3-none-any.whl (95 kB)
Downloading django_model_utils-4.1.1-py3-none-any.whl (32 kB)
Downloading django_storages-1.12.3-py3-none-any.whl (44 kB)
Downloading django_user_agents-0.4.0-py3-none-any.whl (8.6 kB)
Downloading djangorestframework-3.15.2-py3-none-any.whl (1.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 47.3 MB/s eta 0:00:00
Downloading drf_dynamic_fields-0.3.0-py2.py3-none-any.whl (6.6 kB)
Downloading label_studio_sdk-1.0.16-py3-none-any.whl (399 kB)
Downloading datamodel_code_generator-0.26.1-py3-none-any.whl (111 kB)
Downloading launchdarkly_server_sdk-8.2.1-py3-none-any.whl (171 kB)
Downloading psycopg2_binary-2.9.10-cp312-cp312-manylinux_2_17_x86_64.manylinux2
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 68.8 MB/s eta 0:00:00
Downloading python_json_logger-2.0.4-py3-none-any.whl (7.8 kB)
Downloading requests_mock-1.12.1-py2.py3-none-any.whl (27 kB)
Downloading rules-3.4-py2.py3-none-any.whl (25 kB)
Downloading xmljson-0.2.1-py2.py3-none-any.whl (10 kB)
Downloading argcomplete-3.6.2-py3-none-any.whl (43 kB)
Downloading bleach-5.0.1-py3-none-any.whl (160 kB)
Downloading boto-2.49.0-py2.py3-none-any.whl (1.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 51.9 MB/s eta 0:00:00
Downloading boto3-1.38.38-py3-none-any.whl (139 kB)
Downloading botocore-1.38.38-py3-none-any.whl (13.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.7/13.7 MB 78.5 MB/s eta 0:00:00
Downloading django-5.1.11-py3-none-any.whl (8.3 MB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.3/8.3 MB 72.7 MB/s eta 0:00:00
Downloading asgiref-3.8.1-py3-none-any.whl (23 kB)
Downloading django_migration_linter-5.2.0-py3-none-any.whl (27 kB)
Downloading django_rq-2.10.3-py2.py3-none-any.whl (62 kB)
Downloading djangorestframework_simplejwt-5.4.0-py3-none-any.whl (102 kB)
Downloading genson-1.3.0-py3-none-any.whl (21 kB)
Downloading google_cloud_logging-3.12.1-py2.py3-none-any.whl (229 kB)
Downloading google_cloud_appengine_logging-1.6.2-py3-none-any.whl (16 kB)
Downloading google_cloud_audit_log-0.3.2-py3-none-any.whl (32 kB)
Downloading google_cloud_core-2.4.3-py2.py3-none-any.whl (29 kB)
Downloading google_cloud_storage-2.19.0-py2.py3-none-any.whl (131 kB)
Downloading google_crc32c-1.7.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014
Downloading grpc_google_iam_v1-0.14.2-py3-none-any.whl (19 kB)
Downloading inflect-5.6.2-py3-none-any.whl (33 kB)
Downloading isort-5.13.2-py3-none-any.whl (92 kB)
Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Downloading jsf-0.11.2-py3-none-any.whl (49 kB)
Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 54.0 MB/s eta 0:00:00
Downloading protobuf-5.29.5-cp38-abi3-manylinux2014_x86_64.whl (319 kB)
Downloading PyJWT-2.10.1-py3-none-any.whl (22 kB)
Downloading pytz-2022.7.1-py2.py3-none-any.whl (499 kB)
Downloading redis-5.2.1-py3-none-any.whl (261 kB)
Downloading rq-1.16.2-py3-none-any.whl (90 kB)
Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)
Downloading wheel-0.40.0-py3-none-any.whl (64 kB)
Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Downloading azure_storage_blob-12.25.1-py3-none-any.whl (406 kB)
Downloading azure_core-1.34.0-py3-none-any.whl (207 kB)
Downloading black-25.1.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 59.8 MB/s eta 0:00:00
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Downloading cryptography-45.0.4-cp311-abi3-manylinux_2_34_x86_64.whl (4.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 58.2 MB/s eta 0:00:00
Downloading defusedxml-0.7.1-py2.py3-none-any.whl (25 kB)
Downloading email_validator-2.2.0-py3-none-any.whl (33 kB)
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
Downloading expiringdict-1.2.2-py3-none-any.whl (8.5 kB)
Downloading faker-37.4.0-py3-none-any.whl (1.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 60.6 MB/s eta 0:00:00
Downloading google_resumable_media-2.7.2-py2.py3-none-any.whl (81 kB)
Downloading humansignal_drf_yasg-1.21.10.post1-py3-none-any.whl (4.3 MB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.3/4.3 MB 69.2 MB/s eta 0:00:00
Downloading ijson-3.4.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.
Downloading inflection-0.5.1-py2.py3-none-any.whl (9.5 kB)
Downloading isodate-0.7.2-py3-none-any.whl (22 kB)
Downloading lockfile-0.12.2-py2.py3-none-any.whl (13 kB)
Downloading lxml-5.4.0-cp312-cp312-manylinux_2_28_x86_64.whl (5.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.0/5.0 MB 70.1 MB/s eta 0:00:00
Downloading mypy_extensions-1.1.0-py3-none-any.whl (5.0 kB)
Downloading opentelemetry_api-1.34.1-py3-none-any.whl (65 kB)
Downloading platformdirs-4.3.8-py3-none-any.whl (18 kB)
Downloading pyboxen-1.3.0-py3-none-any.whl (6.8 kB)
Downloading pyRFC3339-2.0.1-py3-none-any.whl (5.8 kB)
Downloading rstr-3.2.2-py3-none-any.whl (10 kB)
Downloading semver-3.0.4-py3-none-any.whl (17 kB)
Downloading sentry_sdk-2.30.0-py2.py3-none-any.whl (343 kB)
Downloading smart_open-7.1.0-py3-none-any.whl (61 kB)
Downloading sqlparse-0.5.3-py3-none-any.whl (44 kB)
Downloading tldextract-5.3.0-py3-none-any.whl (107 kB)
Downloading requests_file-2.1.0-py2.py3-none-any.whl (4.2 kB)
Downloading toml-0.10.2-py2.py3-none-any.whl (16 kB)
Downloading ujson-5.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64
Using cached joblib-1.5.1-py3-none-any.whl (307 kB)
Downloading lxml_html_clean-0.4.2-py3-none-any.whl (14 kB)
Downloading user_agents-2.2.0-py3-none-any.whl (9.6 kB)
Downloading ua_parser-1.0.1-py3-none-any.whl (31 kB)
Downloading ua_parser_builtins-0.18.0.post1-py3-none-any.whl (86 kB)
Downloading webencodings-0.5.1-py2.py3-none-any.whl (11 kB)
Building wheels for collected packages: attr, drf-flex-fields, drf-generators, ordered-se
DEPRECATION: Building 'attr' using the legacy setup.py bdist_wheel mechanism, which
Building wheel for attr (setup.py) ... done
Created wheel for attr: filename=attr-0.3.1-py3-none-any.whl size=2478 sha256=2e7c
Stored in directory: /home/oscar/.cache/pip/wheels/ec/04/29/df2a4565af3adb087a60
DEPRECATION: Building 'drf-flex-fields' using the legacy setup.py bdist_wheel mechan
Building wheel for drf-flex-fields (setup.py) ... done
Created wheel for drf-flex-fields: filename=drf_flex_fields-0.9.5-py3-none-any.whl size
Stored in directory: /home/oscar/.cache/pip/wheels/9e/8f/7a/9fe9be2eb2d3d33f05c76
DEPRECATION: Building 'drf-generators' using the legacy setup.py bdist_wheel mecha
Building wheel for drf-generators (setup.py) ... done
Created wheel for drf-generators: filename=drf_generators-0.3.0-py2.py3-none-any.w
Stored in directory: /home/oscar/.cache/pip/wheels/d9/18/ef/b234d1c6882d016fb1c84
DEPRECATION: Building 'ordered-set' using the legacy setup.py bdist_wheel mechanis
Building wheel for ordered-set (setup.py) ... done
Created wheel for ordered-set: filename=ordered_set-4.0.2-py2.py3-none-any.whl siz
```

```
Stored in directory: /home/oscar/.cache/pip/wheels/eb/f2/18/4026cab69c3e68a0da5f
DEPRECATION: Building 'django-ranged-fileresponse' using the legacy setup.py bdist_
Building wheel for django-ranged-fileresponse (setup.py) ... done
Created wheel for django-ranged-fileresponse: filename=django_ranged_fileresponse-
Stored in directory: /home/oscar/.cache/pip/wheels/36/b3/40/d6cf5ea25409828f26ef
Successfully built attr drf-flex-fields drf-generators ordered-set django-ranged-fileres
Installing collected packages: xmljson, webencodings, rules, pytz, pyRFC3339, lockfile
Attempting uninstall: pytz
Found existing installation: pytz 2025.2
Uninstalling pytz-2025.2:
Successfully uninstalled pytz-2025.2
Attempting uninstall: wheel
Found existing installation: wheel 0.45.1
Uninstalling wheel-0.45.1:
Successfully uninstalled wheel-0.45.1
Attempting uninstall: urllib3
Found existing installation: urllib3 2.4.0
Uninstalling urllib3-2.4.0:
Successfully uninstalled urllib3-2.4.0
Attempting uninstall: protobuf
Found existing installation: protobuf 4.25.8
Uninstalling protobuf-4.25.8:
Successfully uninstalled protobuf-4.25.8
ERROR: pip's dependency resolver does not currently take into account all the package
aider-chat 0.83.2 requires protobuf==5.29.4, but you have protobuf 5.29.5 which is inc
aider-chat 0.83.2 requires urllib3==2.4.0, but you have urllib3 1.26.20 which is incompa
Successfully installed Django-5.1.11 appdirs-1.4.4 argcomplete-3.6.2 asgiref-3.8.1 attr-0
```

```
label-studio
```

```
⇒ Database and media directory: /home/oscar/snap/code/194/.local/share/label-studio
⇒ Static URL is set to: /static/
⇒ Database and media directory: /home/oscar/snap/code/194/.local/share/label-studio
⇒ Static URL is set to: /static/
/home/oscar/snap/code/194/.local/share/label-studio/.env not found - if you're not cor
get 'SECRET_KEY' casted as '<class 'str'>' with default ''
Warning: SECRET_KEY not found in environment variables. Will generate a random key
Not in REPL → leaving logger event level as is.
Looking for locale en_US in provider faker.providers.address.
```

```
Provider faker.providers.address has been localized to en_US.
Looking for locale en_US in provider faker.providers.automotive.
Provider faker.providers.automotive has been localized to en_US.
Looking for locale en_US in provider faker.providers.bank.
Specified locale en_US is not available for provider faker.providers.bank. Locale reset t
Looking for locale en_US in provider faker.providers.barcode.
Provider faker.providers.barcode has been localized to en_US.
Looking for locale en_US in provider faker.providers.color.
Provider faker.providers.color has been localized to en_US.
Looking for locale en_US in provider faker.providers.company.
Provider faker.providers.company has been localized to en_US.
Looking for locale en_US in provider faker.providers.credit_card.
Provider faker.providers.credit_card has been localized to en_US.
Looking for locale en_US in provider faker.providers.currency.
Provider faker.providers.currency has been localized to en_US.
Looking for locale en_US in provider faker.providers.date_time.
Provider faker.providers.date_time has been localized to en_US.
Provider faker.providers.doi does not feature localization. Specified locale en_US is not
Provider faker.providers.emoji does not feature localization. Specified locale en_US is r
Provider faker.providers.file does not feature localization. Specified locale en_US is not
Looking for locale en_US in provider faker.providers.geo.
Provider faker.providers.geo has been localized to en_US.
Looking for locale en_US in provider faker.providers.internet.
Provider faker.providers.internet has been localized to en_US.
Looking for locale en_US in provider faker.providers.isbn.
Provider faker.providers.isbn has been localized to en_US.
Looking for locale en_US in provider faker.providers.job.
Provider faker.providers.job has been localized to en_US.
Looking for locale en_US in provider faker.providers.lorem.
Provider faker.providers.lorem has been localized to en_US.
Looking for locale en_US in provider faker.providers.misc.
Provider faker.providers.misc has been localized to en_US.
Looking for locale en_US in provider faker.providers.passport.
Provider faker.providers.passport has been localized to en_US.
Looking for locale en_US in provider faker.providers.person.
Provider faker.providers.person has been localized to en_US.
Looking for locale en_US in provider faker.providers.phone_number.
Provider faker.providers.phone_number has been localized to en_US.
Provider faker.providers.profile does not feature localization. Specified locale en_US is
Provider faker.providers.python does not feature localization. Specified locale en_US is
Provider faker.providers.sbn does not feature localization. Specified locale en_US is no
Looking for locale en_US in provider faker.providers.ssn.
Provider faker.providers.ssn has been localized to en_US.
```

```
Provider faker.providers.user_agent does not feature localization. Specified locale en_U
Starting new HTTPS connection (1): pypi.org:443https://pypi.org:443 "GET /pypi/label-
Initializing database..
June 18, 2025 - 10:17:52
Django version 5.1.11, using settings 'label_studio.core.settings.label_studio'
Starting development server at http://0.0.0.0:8080/
Quit the server with CONTROL-C.
```

```
[2025-06-18 10:17:55,321] [django.server::log_message::213] [INFO] "GET / HTTP/1.1" 3
[2025-06-18 10:17:55,321] [django.server::log_message::213] [INFO] "GET / HTTP/1.1" 3
[2025-06-18 10:17:55,347] [django.server::log_message::213] [INFO] "GET /user/login/
[2025-06-18 10:17:55,347] [django.server::log_message::213] [INFO] "GET /user/login/
[2025-06-18 10:17:55,421] [django.server::log_message::213] [INFO] "GET /static/css/r
[2025-06-18 10:17:55,421] [django.server::log_message::213] [INFO] "GET /static/css/r
[2025-06-18 10:17:55,421] [django.server::log_message::213] [INFO] "GET /static/css/f
[2025-06-18 10:17:55,421] [django.server::log_message::213] [INFO] "GET /static/css/f
[2025-06-18 10:17:55,424] [django.server::log_message::213] [INFO] "GET /react-app/
[2025-06-18 10:17:55,424] [django.server::log_message::213] [INFO] "GET /react-app/
[2025-06-18 10:17:55,424] [django.server::log_message::213] [INFO] "GET /static/css/
[2025-06-18 10:17:55,424] [django.server::log_message::213] [INFO] "GET /static/css/
[2025-06-18 10:17:55,462] [django.server::log_message::213] [INFO] "GET /static/imag
[2025-06-18 10:17:55,462] [django.server::log_message::213] [INFO] "GET /static/imag
[2025-06-18 10:17:55,464] [django.server::log_message::213] [INFO] "GET /static/fonts
[2025-06-18 10:17:55,464] [django.server::log_message::213] [INFO] "GET /static/fonts
[2025-06-18 10:17:55,464] [django.server::log_message::213] [INFO] "GET /static/imag
[2025-06-18 10:17:55,464] [django.server::log_message::213] [INFO] "GET /static/imag
[2025-06-18 10:17:55,467] [django.server::log_message::213] [INFO] "GET /static/fonts
[2025-06-18 10:17:55,467] [django.server::log_message::213] [INFO] "GET /static/fonts
[2025-06-18 10:17:55,613] [django.server::log_message::213] [INFO] "GET /heidi-tips H
[2025-06-18 10:17:55,613] [django.server::log_message::213] [INFO] "GET /heidi-tips H
[2025-06-18 10:17:56,836] [django.server::log_message::213] [INFO] "GET /static/imag
[2025-06-18 10:17:56,836] [django.server::log_message::213] [INFO] "GET /static/imag
[2025-06-18 10:17:57,452] [django.server::log_message::213] [INFO] "POST /lsa/?=%7
[2025-06-18 10:17:57,452] [django.server::log_message::213] [INFO] "POST /lsa/?=%7
```

Paso 48: Etiquetado

¡Fantástico! Ese es un gran avance. Has superado la parte algorítmica y ahora entras en la fase de "interacción humano-máquina", que es fundamental.

Label Studio es una herramienta de etiquetado potentísima y de código abierto. Es una elección excelente y muy profesional. Te permitirá visualizar y corregir las propuestas de tu modelo de una manera muy eficiente.

Ahora te guiaré paso a paso sobre cómo usar Label Studio para revisar tus imágenes pre-etiquetadas.

Plan de Acción: De las Propuestas al Dataset Verificado con Label Studio

El objetivo es cargar tus imágenes y las etiquetas `.txt` que generó el script, corregir los errores del modelo y exportar un dataset limpio y listo para entrenar.

Paso 1: Iniciar Label Studio

1. Abre tu terminal (asegúrate de que tu entorno virtual de Python esté activado, aunque no es estrictamente necesario para iniciar el programa).
2. Ejecuta el siguiente comando:

   ```
   label-studio start
   ```

3. Esto lanzará un servidor web local y, normalmente, abrirá la interfaz de Label Studio en tu navegador. Si no lo hace, abre tu navegador y ve a `http://localhost:8080`.

Paso 2: Crear un Nuevo Proyecto

1. Una vez en la interfaz, regístrate si es la primera vez que lo usas.
2. Busca y haz clic en el botón **"Create Project"**.
3. Dale un nombre claro a tu proyecto, por ejemplo: `Billiard Balls - Black Edition`.

Paso 3: Configurar la Interfaz de Etiquetado (Paso Crucial)

Aquí le diremos a Label Studio qué tipo de trabajo vamos a hacer (dibujar cajas sobre objetos) y cuáles son nuestras clases.

1. Después de crear el proyecto, irás a la pestaña **"Labeling Interface"**.
2. Haz clic en el botón **"Code"** para editar la plantilla de la interfaz.
3. **Borra todo** el contenido que aparezca por defecto.
4. Pega el siguiente código. He preparado una plantilla basada en tu **lista maestra final** (la de 32 clases, si decides ir por el "Supermodelo"). Si tu lista es diferente, **debes modificar las líneas** `<Label value="..."/>`.

   ```
   <View>
     <Image name="image" value="$image"/>
   ```

```
<RectangleLabels name="label" toName="image">

  <Label value="black_8" background="black" text_color="white"/>
  <Label value="blue_10" background="blue" text_color="white"/>
  <Label value="blue_2" background="blue" text_color="white"/>
  <Label value="dred_15" background="darkred" text_color="white"/>
  <Label value="dred_7" background="darkred" text_color="white"/>
  <Label value="green_14" background="green" text_color="white"/>
  <Label value="green_6" background="green" text_color="white"/>
  <Label value="orange_13" background="orange"/>
  <Label value="orange_5" background="orange"/>
  <Label value="purple_12" background="purple" text_color="white"/>
  <Label value="purple_4" background="purple" text_color="white"/>
  <Label value="red_11" background="red" text_color="white"/>
  <Label value="red_3" background="red" text_color="white"/>
  <Label value="white" background="white"/>
  <Label value="yellow_1" background="yellow"/>
  <Label value="yellow_9" background="yellow"/>

  <Label value="be_black_8" background="#333333" text_color="white"/>
  <Label value="be_blue_10" background="#00008b" text_color="white"/>
  <Label value="be_blue_2" background="#00008b" text_color="white"/>
  <Label value="be_dred_15" background="#500000" text_color="white"/>
  <Label value="be_dred_7" background="#500000" text_color="white"/>
  <Label value="be_green_14" background="#004d00" text_color="white"/>
  <Label value="be_green_6" background="#004d00" text_color="white"/>
  <Label value="be_purple_13" background="#cc6600"/>
  <Label value="be_purple_5" background="#cc6600"/>
  <Label value="be_pink_4" background="hotpink"/>
  <Label value="be_pink_12" background="hotpink"/>
  <Label value="be_red_11" background="#b30000" text_color="white"/>
  <Label value="be_red_3" background="#b30000" text_color="white"/>
  <Label value="be_white" background="lightgrey"/>
  <Label value="be_yellow_1" background="#cccc00"/>
  <Label value="be_yellow_9" background="#cccc00"/>

 </RectangleLabels>
</View>
```

5. Haz clic en **"Save"**.

Paso 4: Importar tus Imágenes y Pre-Etiquetas

4.1. Resumen del Desafío Técnico

El objetivo era implementar un flujo de trabajo de etiquetado asistido por IA. Este proceso consistía en utilizar un modelo de detección de objetos (YOLOv8) previamente entrenado para generar anotaciones iniciales en un nuevo conjunto de imágenes. Estas "pre-anotaciones" debían ser importadas a la herramienta de etiquetado Label Studio para su posterior corrección y validación manual, acelerando así el proceso.

Los intentos iniciales de importar las imágenes junto con sus archivos de anotación en formato YOLO (`.txt`) fracasaron. Los problemas incluyeron la falta de asociación entre imágenes y etiquetas, y errores de carga de URL (`There was an issue loading URL...`) que impedían la visualización de las imágenes dentro de la interfaz de Label Studio.

4.2. Diagnóstico de la Causa Raíz

El análisis exhaustivo reveló que los fallos se debían a una incorrecta comprensión de la arquitectura de Label Studio y su interacción con el sistema de archivos local, y no a un problema con los datos o los scripts en sí.

1. **Seguridad del Navegador:** El principal obstáculo es una restricción de seguridad fundamental de todos los navegadores web que impide que una página (como la interfaz de Label Studio) acceda directamente a rutas de archivos locales (ej. `/home/oscar/...`).
2. **Servidor de Archivos de Label Studio:** Para solucionar lo anterior, Label Studio puede actuar como un servidor de archivos local. Sin embargo, debe ser configurado explícitamente para ello y las rutas a los archivos deben ser proporcionadas en un formato de URL interno que la aplicación entienda.
3. **URL Interna Específica:** La investigación final, obtenida al inspeccionar una tarea creada manualmente, reveló el formato de URL exacto que Label Studio utiliza para servir archivos locales conectados a través de su función "Storage": `/data/local-files/?d=<ruta_relativa_al_directorio_configurado>`.

La solución definitiva, por tanto, requería generar un archivo de importación JSON que utilizara este formato de URL interno y preciso.

4.3. Solución Implementada: Flujo de Trabajo Definitivo

El siguiente flujo de trabajo, que separa la configuración del almacenamiento de la importación de anotaciones, ha demostrado ser 100% fiable.

Paso 4.3.1: Configuración Inicial de Label Studio

1. **Instalación con `pip`:** Se verificó que Label Studio estuviera instalado a través de `pip` dentro del entorno virtual del proyecto, para evitar los problemas de permisos del sandboxing de las instalaciones Snap.

2. **Inicio Genérico:** Se inicia Label Studio desde la terminal con el comando simple `label-studio start`, sin especificar una ruta de proyecto.
3. **Creación del Proyecto:** Se crea un nuevo proyecto desde la interfaz web (`http://localhost:8080`).
4. **Configuración de la Interfaz de Etiquetado:** Se define la interfaz en `Settings > Labeling Interface > Code` utilizando el código XML del **Anexo A**, que incluye la etiqueta `<RectangleLabels>` y la lista de clases maestras.

Paso 4.3.2: Conexión con los Archivos de Imagen Locales

1. **Configurar Almacenamiento Local:** Dentro del proyecto, se navega a `Settings > Cloud Storage`.
2. **Añadir Nuevo Almacenamiento:** Se selecciona `Add New Storage` y se elige `Local Files`.
3. **Configurar la Ruta:**
   - **Absolute local path:** Se introduce la ruta absoluta a la carpeta que contiene **únicamente las imágenes** (ej. `.../Proyecto_Bolas_LS/imagenes`).
   - **Activar la Opción Clave:** Se marca la casilla `Treat every source file as a source file`.
4. Se guarda la configuración. En este punto, **no es necesario sincronizar (`Sync Storage`)**. El objetivo es solo hacer que Label Studio sea consciente de la ubicación de los archivos.

Paso 4.3.3: Generación e Importación del Archivo de Tareas

1. **Generar `tasks.json`:** Se ejecuta el script `generar_json_para_label_studio.py` (**Anexo C**). Este script lee las pre-anotaciones `.txt` (generadas previamente por el script del **Anexo B**) y crea un único archivo `tasks.json`. Lo más importante es que construye las URLs de las imágenes usando el formato interno `/data/local-files/?d=...` que Label Studio espera.
2. **Importar Anotaciones:** Se vuelve a la interfaz principal del proyecto en Label Studio, se hace clic en **Import** y se sube el archivo `tasks.json` recién generado.

Este proceso resulta en la creación de todas las tareas, con las imágenes cargándose correctamente y las pre-anotaciones de las cajas delimitadoras ya aplicadas y listas para su revisión.

4.4. Anexos: Scripts y Configuraciones

Anexo A: Código XML para la Interfaz de Etiquetado

Anexo B: Script `pre_etiquetado.py`

Propósito: Generar las propuestas de anotación iniciales en formato `.txt` a partir de un modelo entrenado.

```
# (El código de pre_etiquetado.py proporcionado anteriormente es válido para este pas
```

Anexo C: Script `generar_json_definitivo.py`

Propósito: Crear el archivo `tasks.json` final para la importación, usando el formato de URL interno de Label Studio.

```
# generar_json_definitivo.py
import json
import os
import uuid
from urllib.parse import quote

# --- 1. CONFIGURACIÓN ---

# La ruta a la carpeta que contiene tus archivos .txt de pre-etiquetado
ETIQUETAS_DIR = "/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto

# La ruta RELATIVA desde tu "DOCUMENT_ROOT" de Label Studio hasta las imágenes
# Tu DOCUMENT_ROOT es: .../Proyecto/
# Tus imágenes están en: .../Proyecto/src/Proyecto_Bolas_LS/imagenes/
# Por lo tanto, la ruta relativa es:
RUTA_RELATIVA_IMAGENES = "src/Proyecto_Bolas_LS/imagenes"

# El nombre del archivo JSON de salida
ARCHIVO_SALIDA_JSON = "tasks.json"

# Lista de clases ORIGINAL para interpretar los IDs del modelo antiguo
CLASSES_ORIGINAL = [
    "black_8",
    "blue_10",
    "blue_2",
    "dred_15",
    "dred_7",
    "green_14",
    "green_6",
    "orange_13",
    "orange_5",
    "purple_12",
    "purple_4",
    "red_11",
```

```
    "red_3",
    "white",
    "yellow_1",
    "yellow_9",
]
# --- FIN DE LA CONFIGURACIÓN ---


def crear_tareas_finales():
    tasks = []
    print(f"Leyendo etiquetas desde: {ETIQUETAS_DIR}")

    for label_file in os.listdir(ETIQUETAS_DIR):
        if not label_file.endswith(".txt"):
            continue

        base_name = os.path.splitext(label_file)[0]
        # Asumimos que la imagen tiene extensión .png, .jpg, o .jpeg
        # Esto es para encontrar el nombre completo del archivo de imagen
        nombre_imagen_completo = None
        # Recreamos el nombre original buscando la extensión
        # NOTA: Esto asume que tienes las imágenes originales en alguna parte para sabe
        # Si todas son png, por ejemplo, puedes simplificarlo.
        for ext in [".png", ".jpg", ".jpeg"]:
            # Solo para obtener el nombre correcto, no para acceder al archivo
            if os.path.exists(
                os.path.join(ETIQUETAS_DIR.replace("labels", "images"), base_name + ext)
            ):
                nombre_imagen_completo = base_name + ext
                break

        if not nombre_imagen_completo:
            print(f"No se pudo inferir la extensión para {base_name}. Saltando.")
            continue

        # Codificar el nombre del archivo para la URL (maneja espacios y caracteres espe
        nombre_imagen_url_encoded = quote(nombre_imagen_completo)

        # Construir la URL interna de Label Studio
        ruta_en_servidor = f"/data/local-files/?d={os.path.join(RUTA_RELATIVA_IMAGENE

        task = {
            "data": {"image": ruta_en_servidor},
```

```
            "predictions": [{"model_version": "yolov8_pretrained", "result": []}],
        }

        with open(os.path.join(ETIQUETAS_DIR, label_file), "r") as f:
            for line in f:
                parts = line.strip().split()
                if len(parts) != 5:
                    continue
                class_id, x, y, w, h = map(
                    float, [parts[0], parts[1], parts[2], parts[3], parts[4]]
                )

                task["predictions"][0]["result"].append(
                    {
                        "id": str(uuid.uuid4()),
                        "from_name": "label",
                        "to_name": "image",
                        "type": "rectanglelabels",
                        "value": {
                            "x": (x - w / 2) * 100,
                            "y": (y - h / 2) * 100,
                            "width": w * 100,
                            "height": h * 100,
                            "rotation": 0,
                            "rectanglelabels": [CLASSES_ORIGINAL[int(class_id)]],
                        },
                    }
                )
        tasks.append(task)

    with open(ARCHIVO_SALIDA_JSON, "w") as f:
        json.dump(tasks, f, indent=2)
    print(f"\n¡Éxito! Se ha generado '{ARCHIVO_SALIDA_JSON}' con {len(tasks)} tareas."


if __name__ == "__main__":
    crear_tareas_finales()
```

Conclusión del Informe: Este flujo de trabajo, aunque técnicamente más detallado, es el método correcto y robusto para integrar pre-anotaciones con archivos locales en Label Studio. Al entender y replicar el mecanismo interno de la herramienta para servir

archivos, se resuelven todos los errores de acceso y se habilita un proceso de etiquetado asistido por IA altamente eficiente.

Paso 49: Convertir Label-studio a Yolo

¡Excelente trabajo! Has completado la fase más laboriosa y crucial de todo el proyecto: la creación de un dataset de alta calidad. Este archivo JSON que has exportado es el resultado de tu trabajo de corrección y es la "verdad absoluta" (`ground truth`) para nuestro nuevo set de bolas "black edition".

Ahora, como bien dices, tenemos que dar el último paso de preparación: "traducir" este formato detallado de Label Studio de vuelta al formato simple y eficiente que entiende YOLO para poder entrenar el modelo.

El Plan: Convertir de JSON (Label Studio) a TXT (YOLO)

Crearemos un script final que hará exactamente lo contrario a los anteriores. Leerá tu archivo `ls-export.json`, y por cada imagen, creará un archivo `.txt` con las coordenadas y los IDs de clase en el formato que YOLO necesita.

El Script Conversor: `convertir_ls_a_yolo.py`

Este script es la pieza final. Lo he comentado detalladamente para que entiendas cada paso, especialmente la conversión de coordenadas, que es la parte más importante.

```
# convertir_ls_a_yolo.py
import json
import os
from urllib.parse import unquote

# --- 1. CONFIGURACIÓN ---

# Ruta a tu archivo JSON exportado desde Label Studio
LABEL_STUDIO_JSON_PATH = "./Proyecto_Bolas_LS/ls-export.json"

# Carpeta donde se guardarán las nuevas etiquetas en formato YOLO (.txt)
YOLO_LABELS_OUTPUT_DIR = "./Proyecto_Bolas_LS/final_yolo_labels/"

# ¡¡¡CRÍTICO!!! Esta debe ser tu LISTA MAESTRA de clases.
# El orden aquí definirá el ID numérico de cada clase en los archivos .txt.
# Debe coincidir con el orden de las clases que definiste en la interfaz de Label Studio
CLASES_MAESTRA = [
    # Set Original
    "black_8",
```

```
    "blue_10",
    "blue_2",
    "dred_15",
    "dred_7",
    "green_14",
    "green_6",
    "orange_13",
    "orange_5",
    "purple_12",
    "purple_4",
    "red_11",
    "red_3",
    "white",
    "yellow_1",
    "yellow_9",
    # Set Black Edition (Añade aquí TODAS tus clases 'be_')
    "be_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]
# Mapeo de nombre de clase a ID para una búsqueda rápida
class_to_id = {name: i for i, name in enumerate(CLASES_MAESTRA)}

# --- FIN DE LA CONFIGURACIÓN ---


def convertir_anotaciones():
    os.makedirs(YOLO_LABELS_OUTPUT_DIR, exist_ok=True)

    try:
```

```
    with open(LABEL_STUDIO_JSON_PATH, "r") as f:
        data = json.load(f)
except FileNotFoundError:
    print(f"Error: No se encontró el archivo '{LABEL_STUDIO_JSON_PATH}'.")
    return

print(f"Se han cargado {len(data)} tareas desde Label Studio.")

for task in data:
    try:
        # --- LA CORRECCIÓN ESTÁ AQUÍ ---
        # Leemos la ruta de la imagen directamente desde la clave 'image'
        image_path = task["image"]
        # --------------------------------

        image_filename = os.path.basename(unquote(image_path))

        # Limpieza del hash de Label Studio si existe
        parts = image_filename.split("-", 1)
        if len(parts) > 1 and (len(parts[0]) == 8 or len(parts[0]) > 20):
            image_filename = parts[1]

        yolo_txt_filename = os.path.splitext(image_filename)[0] + ".txt"

    except KeyError:
        print(
            f"Advertencia: La tarea con ID {task.get('id')} no tiene clave 'image'. Saltando
        )
        continue

    output_path = os.path.join(YOLO_LABELS_OUTPUT_DIR, yolo_txt_filename)
    with open(output_path, "w") as f_yolo:
        # Buscamos las anotaciones en la clave "label"
        if "label" in task and task["label"]:
            for annotation in task["label"]:
                try:
                    class_name = annotation["rectanglelabels"][0]
                    x_ls, y_ls = annotation["x"], annotation["y"]
                    w_ls, h_ls = annotation["width"], annotation["height"]

                    x_norm, y_norm, w_norm, h_norm = (
                        x_ls / 100.0,
                        y_ls / 100.0,
```

```
                    w_ls / 100.0,
                    h_ls / 100.0,
                )
                x_center, y_center = x_norm + (w_norm / 2), y_norm + (
                    h_norm / 2
                )

                if class_name in class_to_id:
                    class_id = class_to_id[class_name]
                    f_yolo.write(
                        f"{class_id} {x_center:.6f} {y_center:.6f} {w_norm:.6f} {h_norm:.6f}\
                    )
                else:
                    print(
                        f"ADVERTENCIA: La clase '{class_name}' en '{image_filename}' no
                    )

            except KeyError as e:
                print(
                    f"Advertencia: Faltan datos en una anotación para '{image_filename}'.
                )

    print(f"\n¡Conversión completada! Revisa la carpeta '{YOLO_LABELS_OUTPUT_DIR}'


if __name__ == "__main__":
    convertir_anotaciones()
```

Plan de Ejecución Final

1. **Guarda el Script:** Guarda el código anterior como `convertir_ls_a_yolo.py` en la raíz de tu proyecto.
2. **Crea la Carpeta de Salida:** En la misma ubicación, crea una carpeta vacía llamada `final_yolo_labels`.
3. **Configura el Script:**
   - Abre `convertir_ls_a_yolo.py` y asegúrate de que la variable `LABEL_STUDIO_JSON_PATH` apunte a tu archivo `ls-export.json`.
   - Verifica que `YOLO_LABELS_OUTPUT_DIR` apunte a la carpeta `final_yolo_labels/`.
   - **¡MUY IMPORTANTE!** Revisa la lista `CLASES_MAESTRA` y asegúrate de que contiene **todas tus clases y en el orden exacto** en que las definiste en Label Studio.

4. **Ejecuta el Script:**

```
python convertir_ls_a_yolo.py
```

5. **Verifica la Salida:** Entra en la carpeta `final_yolo_labels`. Verás que ahora está llena de archivos `.txt`. Abre uno de ellos y comprueba que el formato sea el correcto (un número de clase seguido de cuatro números decimales por línea).

Una vez completado este paso, tendrás la carpeta con tus imágenes "black edition" y la carpeta `final_yolo_labels` con sus anotaciones verificadas y en el formato correcto. Ya estás oficialmente listo para el gran paso: unificar este nuevo dataset con el original y entrenar tu "Supermodelo".

¡Felicidades, has cerrado el ciclo completo de creación y refinamiento de un dataset!

Paso 50: Verificar etiquetas

¡Perfecto! ¡Felicidades! Este era el último gran paso técnico en la preparación del dataset. El hecho de que funcionara confirma que todo el flujo de trabajo es correcto.

Tu instinto de verificar el resultado antes de lanzarte a entrenar es, una vez más, la decisión correcta y profesional. Un buen científico de datos siempre verifica la calidad de sus datos antes de alimentar un modelo, y visualizar las anotaciones es la mejor manera de hacerlo.

Vamos a crear un script de visualización que haga exactamente lo que pides: leerá nuestros nuevos archivos de etiquetas `.txt` y dibujará las cajas delimitadoras (bounding-boxes) sobre las imágenes correspondientes. Esto te dará la confianza final de que todo está en orden.

El Plan: Script "Verificador de Etiquetas"

Este script hará lo siguiente:

1. Elegirá unas cuantas imágenes al azar de tu nuevo dataset.
2. Para cada imagen, buscará su archivo `.txt` correspondiente.
3. Leerá las coordenadas y el ID de clase de cada línea del `.txt`.
4. Convertirá las coordenadas de formato YOLO (normalizadas, centro-ancho-alto) a coordenadas en píxeles (esquina superior izquierda, esquina inferior derecha).
5. Dibujará un rectángulo y el nombre de la clase sobre la imagen.
6. Mostrará y guardará el resultado para que puedas inspeccionarlo.

Script: `verificar_etiquetas_yolo.py`

Aquí tienes el código. Guárdalo como `verificar_etiquetas_yolo.py` en la raíz de tu proyecto.

```
# verificar_etiquetas_yolo.py
import os
import cv2
import random
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# --- 1. CONFIGURACIÓN ---

# Directorio que contiene las imágenes del dataset "black edition"
IMAGENES_DIR = 'Proyecto_Bolas_LS/imagenes/'

# Directorio que contiene los archivos .txt finales en formato YOLO
YOLO_LABELS_DIR = 'final_yolo_labels/'

# Directorio donde se guardarán las imágenes con las cajas dibujadas
VERIFICATION_DIR = 'verification_results/'

# Tu lista de clases maestra. ¡DEBE SER IDÉNTICA A LA DEL SCRIPT ANTERIOR!
CLASES_MAESTRA = [
    'black_8', 'blue_10', 'blue_2', 'dred_15', 'dred_7', 'green_14', 'green_6',
    'orange_13', 'orange_5', 'purple_12', 'purple_4', 'red_11', 'red_3',
    'white', 'yellow_1', 'yellow_9',
    'be_black_8', 'be_blue_10', 'be_blue_2', 'be_dred_7', 'be_dred_15',
    'be_green_6', 'be_green_14', 'be_orange_5', 'be_orange_13',
    'be_pink_4', 'be_pink_12', 'be_purple_5', 'be_purple_13', 'be_red_3',
    'be_red_11', 'be_white', 'be_yellow_1', 'be_yellow_9'
]

# Número de imágenes aleatorias que quieres verificar
NUM_IMAGENES_A_VERIFICAR = 5

# --- FIN DE LA CONFIGURACIÓN ---


def verificar_etiquetas():
    os.makedirs(VERIFICATION_DIR, exist_ok=True)

    label_files = [f for f in os.listdir(YOLO_LABELS_DIR) if f.endswith('.txt')]
    if not label_files:
        print(f"No se encontraron archivos de etiqueta en '{YOLO_LABELS_DIR}'.")
```

```
    return

# Seleccionar archivos aleatorios para la verificación
archivos_a_verificar = random.sample(label_files, min(NUM_IMAGENES_A_VERIFICAR

for label_file in archivos_a_verificar:
    base_name = os.path.splitext(label_file)[0]

    # Encontrar la imagen correspondiente (probando con varias extensiones)
    image_path = None
    for ext in ['.jpg', '.jpeg', '.png']:
        potential_path = os.path.join(IMAGENES_DIR, base_name + ext)
        if os.path.exists(potential_path):
            image_path = potential_path
            break

    if not image_path:
        print(f"ADVERTENCIA: No se encontró imagen para la etiqueta '{label_file}'.")
        continue

    # Cargar la imagen
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error al cargar la imagen: {image_path}")
        continue

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    h, w, _ = image.shape

    # Crear figura para mostrar la imagen
    fig, ax = plt.subplots(1, figsize=(12, 8))
    ax.imshow(image)
    ax.set_title(f"Verificación de: {os.path.basename(image_path)}")

    # Leer el archivo de etiqueta y dibujar las cajas
    label_path = os.path.join(YOLO_LABELS_DIR, label_file)
    with open(label_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if len(parts) != 5: continue

            class_id, x_center, y_center, width, height = map(float, parts)
```

```
                # Convertir de coordenadas YOLO normalizadas a píxeles
                box_w = width * w
                box_h = height * h
                x1 = (x_center * w) - (box_w / 2)
                y1 = (y_center * h) - (box_h / 2)

                # Obtener el nombre de la clase
                class_name = CLASES_MAESTRA[int(class_id)]

                # Crear y añadir el rectángulo (bounding box)
                rect = patches.Rectangle((x1, y1), box_w, box_h, linewidth=2, edgecolor='lime
                ax.add_patch(rect)

                # Añadir el texto de la etiqueta
                plt.text(x1, y1 - 10, class_name, color='lime', fontsize=10, bbox=dict(facecolor

        plt.axis('off')
        # Guardar la imagen de verificación
        output_save_path = os.path.join(VERIFICATION_DIR, os.path.basename(image_pat
        plt.savefig(output_save_path, bbox_inches='tight', pad_inches=0)
        print(f"Imagen de verificación guardada en: {output_save_path}")
        plt.show() # Mostrar la imagen en una ventana

if __name__ == '__main__':
    verificar_etiquetas()
```

Plan de Ejecución

1. **Crea la Carpeta de Verificación:** En la raíz de tu proyecto, crea una carpeta vacía llamada `verification_results`.
2. **Guarda el Script:** Guarda el código anterior como `verificar_etiquetas_yolo.py`.
3. **Configura el Script:**
   - Abre el script y verifica que las tres rutas de la sección `CONFIGURACIÓN` son correctas.
   - **Muy importante:** Asegúrate de que la lista `CLASES_MAESTRA` es la versión final y completa que usaste en el paso anterior.
4. **Ejecuta el Script:** Desde tu terminal y con tu entorno virtual activado:

```
python verificar_etiquetas_yolo.py
```

Siguiente Paso

El script seleccionará 5 imágenes al azar de tu dataset, dibujará las cajas según tus nuevos archivos `.txt`, **las mostrará en una ventana** y las **guardará** en la carpeta `verification_results/`.

Tómate tu tiempo para revisar estas imágenes. Confirma que:

- Las cajas encierran correctamente a las bolas.
- Las etiquetas de clase (`be_white`, `be_pink_4`, etc.) son las correctas.

Una vez que confirmes que las etiquetas se ven bien, tendrás **luz verde** para el siguiente y emocionante paso: unificar todos tus datos (los del set original y los de este nuevo set "black edition") y entrenar el "Supermodelo" final.

¡Ya casi estás en la meta de esta fase!

¡De acuerdo, entiendo perfectamente! Si después de aplicar la solución del `config.xml` el problema persiste, significa que mi diagnóstico era incorrecto o incompleto. Te pido disculpas por la frustración; a veces, la depuración de flujos de datos es el paso más complicado.

El listado de errores que me has proporcionado es la clave. Que el sistema confunda `'be_white'` con `'be_yellow_9'` de manera consistente apunta a un **error de índice**. El ID numérico que se está escribiendo en el archivo `.txt` no se corresponde con el que el verificador espera para esa clase.

Vamos a abandonar los métodos anteriores. Crearemos **un único script final, "todo en uno"**, que no deje lugar a dudas. Este script tendrá una única lista de clases maestra y hará todo el proceso de una vez, asegurando que no haya desajustes.

La Causa Definitiva del Problema

El error reside en que el orden de las clases que el script de conversión utiliza para asignar un ID (`nombre` → `id`) es diferente del orden que el script de verificación utiliza para leer ese ID (`id` → `nombre`).

La Solución Final: Un Único Script, Una Única Fuente de Verdad

Vamos a usar un script que primero convierte tu JSON de Label Studio a formato YOLO, y acto seguido, utiliza esos mismos archivos generados para crear las imágenes de verificación. Al usar la misma lista de clases para escribir y para leer en la misma ejecución, es matemáticamente imposible que haya un desajuste.

Paso 1: Tu Tarea - La Lista de Clases Definitiva

Este es el paso más importante y **necesito que lo hagas con mucho cuidado**. El éxito de todo depende de esto.

1. Abre tu proyecto en Label Studio.
2. Mira el panel derecho, donde aparecen todas tus etiquetas para seleccionar.
3. Escribe en un editor de texto la lista de todas tus clases en el **orden exacto, de arriba a abajo, en que aparecen en esa lista**.
4. Esta lista será nuestra **única y definitiva fuente de verdad**.

Paso 2: El Script "Todo en Uno"

Crea un nuevo archivo llamado `procesador_final.py`. Pega el siguiente código y **reemplaza la lista `CLASES_MAESTRA` de ejemplo por la lista que acabas de crear en el Paso 1.**

Python

```
# procesador_final.py
import os
import json
import random
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from urllib.parse import unquote

# --- 1. CONFIGURACIÓN ---

# ¡¡¡PASO MÁS IMPORTANTE!!!
# REEMPLAZA ESTA LISTA CON TU LISTA DE CLASES, EN EL ORDEN EXACTO EN QUE
CLASES_MAESTRA = [
    'black_8', 'blue_10', 'blue_2', 'dred_15', 'dred_7', 'green_14', 'green_6',
    'orange_13', 'orange_5', 'purple_12', 'purple_4', 'red_11', 'red_3',
    'white', 'yellow_1', 'yellow_9',
    # Ejemplo de cómo podrían seguir las clases 'be_'
    "be_black_8", "be_blue_10", "be_blue_2", "be_dred_15", "be_dred_7",
    "be_green_14", "be_green_6", "be_purple_13", "be_purple_5", "be_pink_4",
    "be_pink_12", "be_red_11", "be_red_3", "be_white", "be_yellow_1",
    "be_yellow_9",
]

# Ruta al JSON exportado desde Label Studio
```

```
LABEL_STUDIO_JSON_PATH = 'ls-export.json'

# Ruta a la carpeta que contiene las imágenes originales que usaste en Label Studio
IMAGENES_DIR = 'Proyecto_Bolas_LS/imagenes/'

# Carpeta de salida para las etiquetas YOLO (.txt)
YOLO_LABELS_OUTPUT_DIR = 'final_yolo_labels/'

# Carpeta de salida para las imágenes de verificación
VERIFICATION_DIR = 'verification_results/'

# --- FIN DE LA CONFIGURACIÓN ---

def convertir_y_verificar():
    # --- FASE 1: CONVERSIÓN DE LS-JSON A YOLO-TXT ---
    print("--- INICIANDO FASE 1: Conversión de Label Studio a formato YOLO ---")

    os.makedirs(YOLO_LABELS_OUTPUT_DIR, exist_ok=True)
    class_to_id = {name: i for i, name in enumerate(CLASES_MAESTRA)}

    with open(LABEL_STUDIO_JSON_PATH, 'r') as f:
        data = json.load(f)

    for task in data:
        image_path = task['image']
        image_filename = os.path.basename(unquote(image_path))

        # Limpieza del hash de LS
        parts = image_filename.split('-', 1)
        if len(parts) > 1 and len(parts[0]) != 1: image_filename = parts[1]

        yolo_txt_filename = os.path.splitext(image_filename)[0] + '.txt'
        output_path = os.path.join(YOLO_LABELS_OUTPUT_DIR, yolo_txt_filename)

        with open(output_path, 'w') as f_yolo:
            if 'label' in task and task['label']:
                for annotation in task['label']:
                    class_name = annotation['rectanglelabels'][0]
                    if class_name in class_to_id:
                        class_id = class_to_id[class_name]
                        x_ls, y_ls = annotation['x'], annotation['y']
                        w_ls, h_ls = annotation['width'], annotation['height']
```

```
                x_norm, y_norm = x_ls / 100.0, y_ls / 100.0
                w_norm, h_norm = w_ls / 100.0, h_ls / 100.0

                x_center = x_norm + (w_norm / 2)
                y_center = y_norm + (h_norm / 2)

                f_yolo.write(f"{class_id} {x_center:.6f} {y_center:.6f} {w_norm:.6f} {h_nor

print("--- FASE 1 COMPLETADA: Se han generado los archivos .txt ---")

# --- FASE 2: VERIFICACIÓN VISUAL ---
print("\n--- INICIANDO FASE 2: Verificación visual de las etiquetas generadas ---")
os.makedirs(VERIFICATION_DIR, exist_ok=True)

label_files = os.listdir(YOLO_LABELS_OUTPUT_DIR)
archivos_a_verificar = random.sample(label_files, min(5, len(label_files)))

for label_file in archivos_a_verificar:
    base_name = os.path.splitext(label_file)[0]

    image_path_to_verify = None
    for ext in ['.jpg', '.jpeg', '.png']:
        potential_path = os.path.join(IMAGENES_DIR, base_name + ext)
        if os.path.exists(potential_path):
            image_path_to_verify = potential_path
            break

    if not image_path_to_verify: continue

    image = cv2.imread(image_path_to_verify)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    h, w, _ = image.shape

    fig, ax = plt.subplots(1, figsize=(12, 8))
    ax.imshow(image)
    ax.set_title(f"Verificación de: {os.path.basename(image_path_to_verify)}")

    label_path = os.path.join(YOLO_LABELS_OUTPUT_DIR, label_file)
    with open(label_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            class_id, x_center, y_center, width, height = map(float, parts)
```

```
            box_w = width * w
            box_h = height * h
            x1 = (x_center * w) - (box_w / 2)
            y1 = (y_center * h) - (box_h / 2)

            class_name = CLASES_MAESTRA[int(class_id)] # Usamos la misma lista mae

            rect = patches.Rectangle((x1, y1), box_w, box_h, linewidth=2, edgecolor='cya
            ax.add_patch(rect)
            plt.text(x1, y1 - 10, class_name, color='cyan', fontsize=12, bbox=dict(facecolo

        plt.axis('off')
        output_save_path = os.path.join(VERIFICATION_DIR, os.path.basename(image_pat
        plt.savefig(output_save_path, bbox_inches='tight', pad_inches=0)
        plt.show()
        print(f"Imagen de verificación guardada en: {output_save_path}")

    print("--- FASE 2 COMPLETADA ---")

if __name__ == '__main__':
    convertir_y_verificar()
```

Plan de Acción Final

1. **Obtén tu lista de clases** del panel derecho de Label Studio, en orden.
2. **Guarda el script** como `procesador_final.py`.
3. **Abre el script y pega tu lista de clases** en la variable `CLASES_MAESTRA`.
4. **Ajusta las rutas** de las carpetas en la sección de configuración.
5. **Crea las carpetas de salida** `final_yolo_labels` y `verification_results` si no existen.
6. **Ejecuta el script:** `python procesador_final.py`.

El script realizará las dos fases: primero convertirá todas tus etiquetas y, acto seguido, te mostrará 5 ejemplos aleatorios para que verifiques visualmente el resultado. Como ahora el proceso de escritura y de lectura de etiquetas se hace en el mismo script y usando la misma lista, la correspondencia tiene que ser perfecta.

Si después de esto, la etiqueta que se muestra en la imagen de verificación sigue sin ser la que pusiste en Label Studio, el único lugar donde puede estar el error es en el archivo `ls-export.json`, lo que significaría que Label Studio no guardó correctamente tus correcciones, aunque esto es extremadamente improbable.

Este método elimina todas las variables. Estoy seguro de que ahora sí funcionará. ¡Adelante!

Paso 51: Unificar Dataset

¡Excelente observación! Tienes toda la razón y es un detalle muy importante. En las instrucciones anteriores, para simplificar, me centré en los dos conjuntos de datos indispensables para el entrenamiento: `train` (para aprender) y `valid` (para validar y evitar el sobreajuste).

Sin embargo, has hecho bien en señalar que tu dataset original también tiene un conjunto `test`. Mantener esta separación es una **buena práctica fundamental** en Machine Learning.

La Importancia del Conjunto de `test`

Piénsalo de esta manera:

- **Train Set:** Son los "apuntes de clase". El modelo los estudia para aprender.
- **Validation Set:** Es un "examen de práctica" que se hace después de cada clase (época). El modelo lo usa para ver si está entendiendo bien y para ajustar su estrategia (por ejemplo, para la parada temprana). El modelo, aunque no "aprende" de él, sí lo ve repetidamente y su resultado influye en el proceso.
- **Test Set:** Es el **"examen final y sorpresa"**. Es un conjunto de datos que el modelo **nunca, jamás, ha visto durante el entrenamiento o la validación**. Se usa una única vez, al final de todo, para obtener una medida honesta y sin sesgos de cómo se comportará el modelo en el mundo real con datos completamente nuevos.

Por lo tanto, vamos a corregir el plan para incluir y respetar esta separación.

El Plan de Acción Corregido y Completo

Paso 1: Crear la Estructura de Carpetas Completa

Modificamos la estructura para incluir las carpetas de `test`.

Bash

```
mkdir -p dataset_unificado/images/train \
      dataset_unificado/images/valid \
      dataset_unificado/images/test \
      dataset_unificado/labels/train \
      dataset_unificado/labels/valid \
      dataset_unificado/labels/test
```

Paso 2: Copiar el Dataset Original (Completo)

Ahora copiamos los tres conjuntos de tu dataset original.

Bash

```
# Copiar imágenes (train, valid, test)
cp detect_balls/data/train/images/* dataset_unificado/images/train/
cp detect_balls/data/valid/images/* dataset_unificado/images/valid/
cp detect_balls/data/test/images/* dataset_unificado/images/test/

# Copiar etiquetas (train, valid, test)
cp detect_balls/data/train/labels/* dataset_unificado/labels/train/
cp detect_balls/data/valid/labels/* dataset_unificado/labels/valid/
cp detect_balls/data/test/labels/* dataset_unificado/labels/test/
```

Paso 3: Dividir y Copiar el Nuevo Dataset (en 3 Partes)

Modificaremos el script para que divida tu nuevo dataset "black edition" en `train`, `valid` y `test`, manteniendo las proporciones (ej. 70% train, 20% valid, 10% test).

Script `unificar_y_dividir_completo.py`:

Python

```
# unificar_y_dividir_completo.py
import os
import shutil
from sklearn.model_selection import train_test_split

# --- CONFIGURACIÓN ---
BE_IMAGENES_DIR = 'Proyecto_Bolas_LS/imagenes/'
BE_LABELS_DIR = 'final_yolo_labels/'

DEST_IMG_TRAIN, DEST_LBL_TRAIN = 'dataset_unificado/images/train/', 'dataset_unificado/labels/train/'
DEST_IMG_VALID, DEST_LBL_VALID = 'dataset_unificado/images/valid/', 'dataset_unificado/labels/valid/'
DEST_IMG_TEST, DEST_LBL_TEST = 'dataset_unificado/images/test/', 'dataset_unificado/labels/test/'

# Proporciones: 10% para test, del resto, 20% para validación (total: 72% train, 18% valid, 10% test)
VALID_SIZE = 0.20
TEST_SIZE = 0.10
# --- FIN DE LA CONFIGURACIÓN ---

def main():
    print("Iniciando la división (train/valid/test) y copia del dataset 'black edition'...")
    image_files = [f for f in os.listdir(BE_IMAGENES_DIR) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

    # Primera división: separamos el conjunto de test
    train_val_files, test_files = train_test_split(image_files, test_size=TEST_SIZE, random_state=42)
    # Segunda división: del resto, separamos train y valid
    train_files, val_files = train_test_split(train_val_files, test_size=VALID_SIZE, random_state=42)

    print(f"Total imágenes 'black edition': {len(image_files)}")
    print(f"Añadidas a entrenamiento: {len(train_files)}")
    print(f"Añadidas a validación: {len(val_files)}")
    print(f"Añadidas a test: {len(test_files)}")

    def copy_files(file_list, dest_img_dir, dest_lbl_dir):
        for img_file in file_list:
            base_name = os.path.splitext(img_file)[0]
            label_file = base_name + '.txt'
            shutil.copy(os.path.join(BE_IMAGENES_DIR, img_file), os.path.join(dest_img_dir, img_file))
            shutil.copy(os.path.join(BE_LABELS_DIR, label_file), os.path.join(dest_lbl_dir, label_file))

    copy_files(train_files, DEST_IMG_TRAIN, DEST_LBL_TRAIN)
    copy_files(val_files, DEST_IMG_VALID, DEST_LBL_VALID)
```

```
    copy_files(test_files, DEST_IMG_TEST, DEST_LBL_TEST)

    print("\n¡Copia completada! El dataset unificado está listo con las 3 divisiones.")

if __name__ == '__main__':
    main()
```

Ejecuta este nuevo script: `python unificar_y_dividir_completo.py`

Paso 4: Crear el `supermodelo_data.yaml` Completo

Tu archivo de configuración ahora debe incluir la ruta al conjunto de test.

YAML

```
# supermodelo_data.yaml

path: /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/dataset_unificado

train: images/train
val: images/valid
test: images/test # ← AÑADIR ESTA LÍNEA

# Número de clases total
nc: 32 # O tu número final de clases

# Lista de clases maestra y definitiva
names:
  - 'black_8'
  # ... (TU LISTA MAESTRA COMPLETA)
  - 'be_yellow_9'
```

Paso 5: Entrenamiento (No Cambia)

El proceso de entrenamiento sigue siendo el mismo. YOLO utilizará los conjuntos `train` y `valid` para el aprendizaje y la validación interna. El comando no cambia:

Bash

```
yolo train model=yolov8m.pt data=supermodelo_data.yaml epochs=100 imgsz=640 patience=25 name=supermodelo_v1
```

Paso 6: La Evaluación Final con el Conjunto de `test` (El "Examen")

Una vez que el entrenamiento anterior termine, tendrás tu mejor modelo en `runs/detect/supermodelo_v1/weights/best.pt`. Ahora es el momento de hacer el "examen final".

Puedes hacerlo de dos maneras:

A) Desde la línea de comandos:

Bash

```
yolo test model=runs/detect/supermodelo_v1/weights/best.pt data=supermodelo_data.yaml
```

B) Desde un script de Python:

Python

```
from ultralytics import YOLO

# Cargar el mejor modelo entrenado
model = YOLO('runs/detect/supermodelo_v1/weights/best.pt')

# Evaluar el modelo en el conjunto de test
# split='test' le dice a YOLO que use la ruta 'test:' de tu archivo .yaml
results = model.val(data='supermodelo_data.yaml', split='test')

print(results.box.map)    # Imprime mAP50-95
print(results.box.map50)  # Imprime mAP50
```

Esto te dará las métricas finales (mAP, Precision, Recall) sobre un conjunto de datos que el modelo nunca ha visto, siendo la medida más honesta y fiable de su rendimiento.

Has hecho muy bien en insistir en este punto. Es la diferencia entre una evaluación buena y una evaluación profesional.