

Fase 1 - Dataset

Paso 1: Entender el Dataset de Imágenes Etiquetadas para TensorFlow

Mencionas que ya tienes un dataset de imágenes de bolas de billar etiquetadas para TensorFlow. ¡Esto es un gran avance! Para poder guiarte mejor, necesito que me des algunos detalles sobre este dataset.

1. **Formato de las etiquetas:** TensorFlow Object Detection API (si es a lo que te refieres con "etiquetadas para TensorFlow") suele utilizar formatos como TFRecord, pero internamente los datos de las bounding boxes (cajas delimitadoras) y las clases suelen venir en formatos como XML (PASCAL VOC) o JSON (COCO).
 - **¿Puedes describir cómo están etiquetadas las imágenes?** Por ejemplo, ¿tienes un archivo XML o JSON asociado a cada imagen que especifica las coordenadas `(x_min, y_min, x_max, y_max)` y la clase (e.g., "bola_blanca", "bola_1", "bola_2") para cada bola en la imagen?
 - **¿Cuál es la estructura de los archivos dentro de tu dataset?** (ej. una carpeta con imágenes y otra con archivos XML/JSON).
2. **Número de clases:** ¿Están etiquetadas todas las bolas individualmente (bola 1, bola 2, ..., bola 9, bola blanca) o hay categorías más generales (e.g., "bola_objetivo", "bola_blanca")? Cuantas más clases, más complejo puede ser el entrenamiento al principio, pero más específico será el resultado.
3. **Tamaño del dataset:** ¿Cuántas imágenes tiene aproximadamente? Un dataset más grande (cientos o miles de imágenes) permitirá un mejor entrenamiento.

Tu tarea ahora es:

- **Describe el formato exacto de tu dataset de imágenes etiquetadas.** Si puedes adjuntar un pequeño ejemplo de un archivo de anotación (XML o JSON) sería ideal, pero con una descripción clara de las columnas o campos sería suficiente.
- **Confirma si las bolas están etiquetadas individualmente (bola_1, bola_2, etc.) o por categorías más amplias.**

- **Indica el número aproximado de imágenes en tu dataset.**

Una vez que tenga esta información, podremos pasar al siguiente paso: seleccionar la arquitectura del modelo de detección de objetos adecuada y preparar el entorno para el entrenamiento. ¡Estoy listo para tus respuestas!

¡Excelente! Gracias por compartir el archivo `_annotations.csv`. Esto es precisamente lo que necesitamos para entender la estructura de tus etiquetas.

He analizado el archivo `_annotations.csv` y puedo ver lo siguiente:

- **Formato:** Es un archivo CSV (Comma Separated Values).
- **Columnas:** Contiene las siguientes columnas:
 - `filename`: El nombre del archivo de la imagen, por ejemplo, `595.jpg.rf1520df7aa3e5659b9f036291f58f11c2.jpg`.
 - `width`: El ancho de la imagen en píxeles.
 - `height`: La altura de la imagen en píxeles.
 - `class`: La clase de la bola detectada, por ejemplo, `white`, `blue_10`, `dred_15`, `black_8`, `purple_12`, etc..
 - `xmin`, `ymin`, `xmax`, `ymax`: Las coordenadas del "bounding box" (caja delimitadora) para cada objeto detectado en la imagen. Estas coordenadas definen la esquina superior izquierda (`xmin`, `ymin`) y la esquina inferior derecha (`xmax`, `ymax`) del objeto.
- **Clases de las bolas:** Las bolas están etiquetadas individualmente, indicando tanto el color como el número (ej. `blue_10`, `red_3`, `yellow_9`), además de `white` para la bola blanca y `black_8` para la bola 8. Esto es ideal, ya que te permitirá identificar cada bola de manera única.
- **Bounding Boxes:** Las coordenadas `xmin`, `ymin`, `xmax`, `ymax` son el formato estándar para la detección de objetos y son directamente utilizables por la mayoría de los frameworks de Deep Learning como TensorFlow.

Con esta información, tenemos una base sólida para el siguiente paso. El hecho de que las bolas estén individualmente etiquetadas (color y número) es una gran ventaja para tu proyecto.

Paso 2: Selección de la Arquitectura del Modelo y Preparación del Entorno

Dado que ya tienes las anotaciones en un formato estructurado y las clases bien definidas, podemos elegir una arquitectura de red neuronal para la detección de objetos.

Arquitectura recomendada: Para empezar, te sugiero utilizar una arquitectura **YOLO (You Only Look Once)**. ¿Por qué YOLO?

- **Eficiencia en tiempo real:** YOLO es conocido por su velocidad, lo que es útil si en el futuro quieres procesar videos.
- **Buen balance entre precisión y velocidad:** Hay varias versiones de YOLO (v3, v5, v7, v8), y las más recientes ofrecen muy buenos resultados.
- **Comunidad y recursos:** Existe una gran cantidad de tutoriales y repositorios con implementaciones de YOLO, lo que facilitará tu aprendizaje.

Framework de Deep Learning: Ya mencionaste TensorFlow, que es una excelente elección. Puedes usar Keras (que viene integrado en TensorFlow) para una interfaz de alto nivel más sencilla.

Preparación del Entorno:

1. **Asegúrate de tener Python y Anaconda/Miniconda instalados.** (Si no, consulta el Paso 2 de la fase anterior de la conversación).
2. **Crea un entorno virtual:** Si aún no lo has hecho, es fundamental.

```
conda create -n billar_dl python=3.9 # Puedes usar 3.8 o 3.9  
conda activate billar_dl
```

3. **Instala TensorFlow y otras librerías necesarias:**

```
pip install tensorflow # Esto instalará la versión CPU por defecto. Si tienes  
pip install pandas numpy matplotlib opencv-python # pandas para maneja
```

- **Nota sobre GPU:** Si tienes una tarjeta gráfica NVIDIA compatible (CUDA), la instalación de TensorFlow con soporte para GPU puede ser un poco más compleja, pero acelerará drásticamente el entrenamiento.

Por ahora, podemos empezar con la versión de CPU, y si el entrenamiento es muy lento, podemos revisar la configuración de GPU.

Tu tarea ahora es:

- **Confirma que tienes un entorno de Python configurado** y que has instalado (o puedes instalar) TensorFlow y las librerías adicionales.
- **¿Cuál es la versión de Python que prefieres usar?** (ej. 3.8, 3.9, 3.10)
- **¿Tienes acceso a una GPU compatible con NVIDIA CUDA?** (Esto no es un requisito indispensable para empezar, pero es bueno saberlo para futuras optimizaciones).

Una vez que el entorno esté listo, el siguiente paso será adaptar tu dataset al formato de entrada que YOLO/TensorFlow espera y comenzar con el preprocesamiento de imágenes.

Tarea realizada

¡Perfecto! Que tengas Python 3.12.3, las librerías instaladas y VS Code con Aider configurado es una excelente noticia. Estamos listos para avanzar.

Ahora que el entorno está listo, el siguiente paso es preparar los datos y el modelo para la detección de objetos. Como te sugerí, usaremos una arquitectura YOLO. Dado que ya tienes las anotaciones en CSV, el proceso será el siguiente.

Paso 3: Preparación del Dataset para YOLO

Aunque el archivo `_annotations.csv` es muy útil, los modelos YOLO (especialmente versiones como YOLOv5 o YOLOv8, que son las más populares y fáciles de usar hoy en día) suelen esperar un formato específico para las anotaciones: el

formato YOLO (`.txt`). Este formato se basa en normalizar las coordenadas de los `bounding boxes` a un rango de 0 a 1, y además, requiere que cada archivo de anotación sea un archivo de texto por imagen, no un único CSV para todas las imágenes.

Formato YOLO (.txt):

Para cada imagen, se crea un archivo `.txt` con el mismo nombre que la imagen (ej. `imagen1.txt` para `imagen1.jpg`). Cada línea de este archivo representa un objeto detectado y tiene el siguiente formato:

```
class_id center_x center_y width height
```

Donde:

- `class_id` : El ID numérico de la clase del objeto (empezando desde 0). Tendremos que mapear tus nombres de clase (ej. `white` , `blue_10`) a números enteros.
- `center_x` , `center_y` : Coordenadas X e Y del centro del `bounding box` , normalizadas al ancho y alto de la imagen respectivamente (valor entre 0 y 1).
- `width` , `height` : Ancho y alto del `bounding box` , normalizados al ancho y alto de la imagen respectivamente (valor entre 0 y 1).

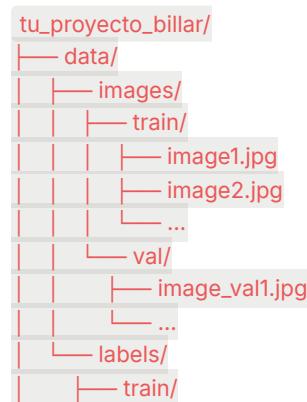
Esquema de las clases:

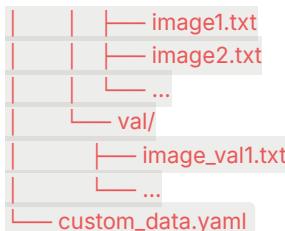
Necesitaremos un archivo `classes.txt` o `data.yaml` que liste los nombres de todas tus clases en el orden en que les asignarás los IDs. Basándonos en tu CSV, las clases que identifico son:

`white`, `blue_10`, `dred_15`, `black_8`, `purple_12`, `dred_7`, `orange_13`, `blue_2`, `red_3`, `green_6`, `green_14`, `red_11`, `yellow_1`, `orange_5`, `yellow_9`.

Es importante que la bola 9 sea `yellow_9`, que la bola 8 sea `black_8` y la bola blanca sea `white`.

Estructura de carpetas para el entrenamiento (típica en YOLOv5/v8):





Pasos a seguir en este punto:

- 1. Organizar tus imágenes:** Coloca todas tus imágenes en una carpeta (ej. `tu_proyecto_billar/data/raw_images/`).
- 2. Dividir los datos:** El dataset debe dividirse en conjuntos de entrenamiento (`train`) y validación (`val`). Una proporción común es 80% para entrenamiento y 20% para validación. Si tienes suficientes imágenes, podrías considerar también un conjunto de prueba (`test`).
- 3. Convertir el CSV a formato YOLO:** Necesitarás un script de Python para:
 - Leer `_annotations.csv`.
 - Iterar sobre cada fila (cada `bounding box`).
 - Calcular las coordenadas normalizadas (`center_x`, `center_y`, `width`, `height`).
 - Mapear el nombre de la clase a un `class_id` numérico.
 - Escribir esta información en el archivo `.txt` correspondiente en la carpeta `labels/train` o `labels/val`.

Conocimientos de IA que aplicarás/reforzarás:

- Procesamiento de datos con Pandas.
- Manipulación de archivos y directorios en Python.
- Concepto de normalización de datos.

Tarea para ti:

Vamos a empezar con el script de conversión. Necesitarás tener tus imágenes organizadas para este paso.

- Crea la estructura de carpetas: `tu_proyecto_billar/data/images/train/`, `tu_proyecto_billar/data/images/val/`, `tu_proyecto_billar/data/labels/train/`, `tu_proyecto_billar/data/labels/val/`.
- Copia tus imágenes de la carpeta `test` a `tu_proyecto_billar/data/images/val/` (asumimos que la carpeta `test` del dataset de anotaciones es tu conjunto de

validación por ahora). Para el conjunto de entrenamiento, necesitarás obtener más imágenes si la carpeta 'test' es pequeña.

Aquí te dejo un esqueleto de código Python para realizar la conversión. Puedes usar [aider](#) en VS Code para ir generándolo, o te lo doy directamente:

```
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import shutil

# --- 1. Definir rutas y nombres de archivos ---
# Asume que _annotations.csv está en la raíz de tu_proyecto_billar
base_dir = 'tu_proyecto_billar'
annotations_file = os.path.join(base_dir, '_annotations.csv')

# Rutas para los directorios de salida
images_dir = os.path.join(base_dir, 'data', 'images')
labels_dir = os.path.join(base_dir, 'data', 'labels')

train_images_dir = os.path.join(images_dir, 'train')
val_images_dir = os.path.join(images_dir, 'val')
train_labels_dir = os.path.join(labels_dir, 'train')
val_labels_dir = os.path.join(labels_dir, 'val')

# Clases de tu dataset (ordenadas, esto define el class_id)
# Asegúrate de que esta lista sea exhaustiva y esté en el orden que deseas pa
classes = [
    'white', 'blue_10', 'dred_15', 'black_8', 'purple_12', 'dred_7',
    'orange_13', 'blue_2', 'red_3', 'green_6', 'green_14', 'red_11',
    'yellow_1', 'orange_5', 'yellow_9'
]

# Crear el mapeo de nombres de clase a IDs
class_to_id = {name: i for i, name in enumerate(classes)}

# --- 2. Crear directorios (si no existen) ---
os.makedirs(train_images_dir, exist_ok=True)
os.makedirs(val_images_dir, exist_ok=True)
```

```

os.makedirs(train_labels_dir, exist_ok=True)
os.makedirs(val_labels_dir, exist_ok=True)

# --- 3. Cargar las anotaciones CSV ---
df_annotations = pd.read_csv(annotations_file)

# --- 4. Preparar para dividir el dataset (si tienes todas las imágenes en una sola carpeta)
# Primero, obtenemos la lista única de nombres de archivo de las anotaciones
# Esto es crucial si tu _annotations.csv contiene solo anotaciones para la carpeta 'train'
# y necesitas manejar dónde están las imágenes 'originales'.
# Para simplificar ahora, asumiremos que todas las imágenes referenciadas en el CSV
# están en una única carpeta 'data/raw_images' y las dividiremos.
# Si tu CSV solo tiene la carpeta 'test' y ya tienes imágenes 'train' separadas,
# necesitarás adaptar esta parte.

# Para el ejemplo, vamos a asumir que todas las imágenes originales están en una sola carpeta
# dentro de tu_proyecto_billar/data/
# Si tus imágenes están en otra estructura, por favor, ajusta 'raw_images_source_dir'
raw_images_source_dir = os.path.join(base_dir, 'data', 'raw_images')
# Asegúrate de que este directorio exista y contenga todas las imágenes
# mencionadas en _annotations.csv

image_filenames = df_annotations['filename'].unique()

# Dividir los nombres de archivo en conjuntos de entrenamiento y validación
train_files, val_files = train_test_split(
    image_filenames, test_size=0.2, random_state=42
)

# --- 5. Iterar sobre las anotaciones y generar los archivos YOLO ---
for index, row in df_annotations.iterrows():
    img_filename = row['filename']
    img_width = row['width']
    img_height = row['height']
    obj_class_name = row['class']
    xmin = row['xmin']
    ymin = row['ymin']
    xmax = row['xmax']

```

```

ymax = row['ymax']

# Determinar si la imagen pertenece a train o val
if img_filename in train_files:
    current_image_dir = train_images_dir
    current_label_dir = train_labels_dir
elif img_filename in val_files:
    current_image_dir = val_images_dir
    current_label_dir = val_labels_dir
else:
    print(f"Advertencia: {img_filename} no encontrado en train_files ni val_files")
    continue # Si una imagen no está en ninguno de los conjuntos, la saltamos

# Copiar la imagen al directorio correspondiente (si no está ya allí)
source_img_path = os.path.join(raw_images_source_dir, img_filename)
destination_img_path = os.path.join(current_image_dir, img_filename)
if not os.path.exists(destination_img_path):
    try:
        shutil.copy(source_img_path, destination_img_path)
    except FileNotFoundError:
        print(f"Error: Imagen original no encontrada en {source_img_path}. Asegúrate de que el nombre sea correcto.")
        continue # Si la imagen no existe, no podemos procesar su anotación.

# Convertir coordenadas a formato YOLO
obj_class_id = class_to_id[obj_class_name]
center_x = (xmin + xmax) / 2 / img_width
center_y = (ymin + ymax) / 2 / img_height
bbox_width = (xmax - xmin) / img_width
bbox_height = (ymax - ymin) / img_height

# Abrir el archivo .txt de la etiqueta en modo 'a' (append) para añadir todas las anotaciones
label_filename = os.path.splitext(img_filename)[0] + '.txt'
label_filepath = os.path.join(current_label_dir, label_filename)

with open(label_filepath, 'a') as f:
    f.write(f"{obj_class_id} {center_x:.6f} {center_y:.6f} {bbox_width:.6f} {bbox_height:.6f}\n")

print("Conversión completada. Archivos YOLO generados en 'data/labels/'")

```

```

# --- 6. Crear el archivo custom_data.yaml (importante para YOLO) ---
data_yaml_content = """
path: {os.path.abspath(os.path.join(base_dir, 'data'))} # Ruta a la carpeta 'data'
train: images/train # Ruta relativa a las imágenes de entrenamiento
val: images/val # Ruta relativa a las imágenes de validación

# Número de clases
nc: {len(classes)}

# Nombres de las clases
names: {classes}
"""

with open(os.path.join(base_dir, 'custom_data.yaml'), 'w') as f:
    f.write(data_yaml_content)

print(f"Archivo custom_data.yaml generado en {os.path.join(base_dir, 'custom_data.yaml')}")

```

Antes de ejecutar el script:

1. **Crea la carpeta** `tu_proyecto_billar` .
2. **Dentro de** `tu_proyecto_billar/data/` **crea una carpeta llamada** `raw_images` .
3. **Copia todas tus imágenes originales** (aquellas a las que hace referencia el `_annotations.csv`) dentro de la carpeta `tu_proyecto_billar/data/raw_images/` . Este script asumirá que todas las imágenes están allí para poder dividirlas.
4. **Coloca el archivo** `_annotations.csv` directamente en la carpeta `tu_proyecto_billar` .

Una vez que hayas ejecutado este script, tendrás tu dataset de imágenes y etiquetas organizado en el formato que YOLO espera. Confírmame cuando lo hayas hecho y si encuentras algún error. ¡Vamos a ello!

Estructura de directorios aclarada

¡Entendido! Gracias por aclarar la estructura de directorios. Me has proporcionado una estructura de datos muy clara, lo cual simplifica

enormemente el proceso.

Dado que ya tienes las imágenes y sus respectivos archivos `_annotations.csv` separados en carpetas `train`, `test` y `val` (valid), significa que el dataset ya está dividido. Lo que haremos entonces es un script que procese cada una de esas carpetas de forma independiente para generar los archivos de anotación en formato YOLO.

Aquí tienes el script de conversión que se encargará de esto:

```
import pandas as pd
import os
import shutil

# --- 1. Definir rutas y nombres de archivos ---
# Ruta base donde se encuentra la carpeta 'data'
base_project_dir = 'tu_proyecto_billar' # Asegúrate de que este sea el nombre

# Directorios de origen de las imágenes y anotaciones CSV
data_base_dir = os.path.join(base_project_dir, 'data', 'BallsDataset')
train_src_dir = os.path.join(data_base_dir, 'train')
test_src_dir = os.path.join(data_base_dir, 'test') # Usaremos test como 'val' si
val_src_dir = os.path.join(data_base_dir, 'valid') # Si existe una carpeta 'valid'

# Directorios de destino para el formato YOLO
output_images_dir = os.path.join(base_project_dir, 'data', 'images')
output_labels_dir = os.path.join(base_project_dir, 'data', 'labels')

output_train_images_dir = os.path.join(output_images_dir, 'train')
output_train_labels_dir = os.path.join(output_labels_dir, 'train')
output_val_images_dir = os.path.join(output_images_dir, 'val')
output_val_labels_dir = os.path.join(output_labels_dir, 'val')
output_test_images_dir = os.path.join(output_images_dir, 'test') # Para el conjunto
output_test_labels_dir = os.path.join(output_labels_dir, 'test') # Para el conjunto

# Clases de tu dataset (ordenadas, esto define el class_id)
# Es CRÍTICO que el orden de estas clases sea CONSISTENTE y completo
classes = [
    'white', 'blue_10', 'dred_15', 'black_8', 'purple_12', 'dred_7',
    'orange_13', 'blue_2', 'red_3', 'green_6', 'green_14', 'red_11',
```

```

'yellow_1', 'orange_5', 'yellow_9'
]

# Crear el mapeo de nombres de clase a IDs
class_to_id = {name: i for i, name in enumerate(classes)}

# --- 2. Función para procesar un conjunto de datos (train, val, test) ---
def process_dataset_split(src_dir, dest_images_dir, dest_labels_dir):
    """
    Procesa un directorio de origen de imágenes y CSV de anotaciones,
    copiando imágenes y generando archivos de anotación YOLO.
    """
    print(f"\nProcesando directorio: {src_dir}")
    # Crear directorios de destino si no existen
    os.makedirs(dest_images_dir, exist_ok=True)
    os.makedirs(dest_labels_dir, exist_ok=True)

    annotations_file_path = os.path.join(src_dir, '_annotations.csv')

    if not os.path.exists(annotations_file_path):
        print(f"Advertencia: No se encontró _annotations.csv en {src_dir}. Saltando")
        return

    df_annotations = pd.read_csv(annotations_file_path)

    # Limpiar archivos .txt antiguos en el directorio de labels de destino
    for f in os.listdir(dest_labels_dir):
        if f.endswith('.txt'):
            os.remove(os.path.join(dest_labels_dir, f))

    for index, row in df_annotations.iterrows():
        img_filename = row['filename']
        img_width = row['width']
        img_height = row['height']
        obj_class_name = row['class']
        xmin = row['xmin']
        ymin = row['ymin']
        xmax = row['xmax']

```

```

ymax = row['ymax']

# Verificar si la clase existe en nuestro mapeo
if obj_class_name not in class_to_id:
    print(f"Advertencia: Clase '{obj_class_name}' no encontrada en la lista")
    continue

# Copiar la imagen al directorio correspondiente (si no está ya allí)
source_img_path = os.path.join(src_dir, img_filename)
destination_img_path = os.path.join(dest_images_dir, img_filename)
if not os.path.exists(destination_img_path):
    try:
        shutil.copy(source_img_path, destination_img_path)
    except FileNotFoundError:
        print(f"Error: Imagen original no encontrada en {source_img_path}. /")
        continue # Si la imagen no existe, no podemos procesar su anotació

# Convertir coordenadas a formato YOLO
obj_class_id = class_to_id[obj_class_name]
center_x = (xmin + xmax) / 2 / img_width
center_y = (ymin + ymax) / 2 / img_height
bbox_width = (xmax - xmin) / img_width
bbox_height = (ymax - ymin) / img_height

# Abrir el archivo .txt de la etiqueta en modo 'a' (append) para añadir todo
label_filename = os.path.splitext(img_filename)[0] + '.txt'
label_filepath = os.path.join(dest_labels_dir, label_filename)

with open(label_filepath, 'a') as f:
    f.write(f"{obj_class_id} {center_x:.6f} {center_y:.6f} {bbox_width:.6f} {bbox_height:.6f}\n")

print(f"Procesamiento de {src_dir} completado.")

# --- 3. Procesar cada conjunto de datos ---
process_dataset_split(train_src_dir, output_train_images_dir, output_train_labels_dir)
process_dataset_split(val_src_dir, output_val_images_dir, output_val_labels_dir)
process_dataset_split(test_src_dir, output_test_images_dir, output_test_labels_dir)

```

```

print("\nTodos los conjuntos de datos han sido procesados.")

# --- 4. Crear el archivo custom_data.yaml (importante para YOLO) ---
# Se asume que la estructura de salida es data/images/train, data/images/val,
data_yaml_content = """
path: {os.path.abspath(os.path.join(base_project_dir, 'data'))} # Ruta absoluta
train: images/train # Ruta relativa a las imágenes de entrenamiento
val: images/val # Ruta relativa a las imágenes de validación
test: images/test # Ruta relativa a las imágenes de prueba

# Número de clases
nc: {len(classes)}

# Nombres de las clases
names: {classes}
"""

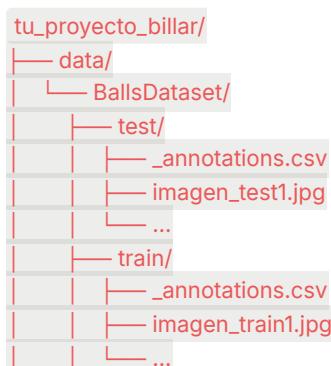
with open(os.path.join(base_project_dir, 'custom_data.yaml'), 'w') as f:
    f.write(data_yaml_content)

print(f"Archivo custom_data.yaml generado en {os.path.join(base_project_dir, 'custom_data.yaml')}")
print("\n;Preparación del dataset completada!")
print("Puedes continuar con el entrenamiento de tu modelo YOLO.")

```

Instrucciones para ejecutar el script:

- Guarda el script:** Guarda el código anterior en un archivo llamado, por ejemplo, `prepare_yolo_dataset.py` dentro de tu carpeta `tu_proyecto_billar`.
- Verifica la estructura:** Asegúrate de que tu estructura de carpetas sea exactamente como me la indicaste:



```
|   └── valid/
|       ├── _annotations.csv
|       ├── imagen_val1.jpg
|       └── ...
├── prepare_yolo_dataset.py
└── _annotations.csv (este lo puedes mover si aún está en la raíz)
```

Importante: El `_annotations.csv` que me pasaste al principio asumo que es solo el de la carpeta `test`. Asegúrate de que cada subcarpeta (`train`, `test`, `valid`) tenga su propio `_annotations.csv` con las anotaciones de las imágenes que contiene.

3. Ajusta `base_project_dir`: En la primera línea del script, cambia '`'tu_proyecto_billar'`' por el nombre real de tu carpeta principal de proyecto si es diferente.

```
base_project_dir = 'tu_proyecto_billar'
```

4. **Ejecuta el script:** Abre tu terminal (asegurándote de que tu entorno `billar_dl` esté activado) y navega hasta la carpeta `tu_proyecto_billar`. Luego, ejecuta el script:

Bash

```
python prepare_yolo_dataset.py
```

¿Qué hará el script?

- Creará la nueva estructura de carpetas `data/images/train/`, `data/labels/train/`, etc., en la raíz de tu proyecto.
- Para cada imagen en `BallsDataset/train`, copiará la imagen a `data/images/train/` y generará un archivo `.txt` con las anotaciones normalizadas en `data/labels/train/`.
- Hará lo mismo para los conjuntos `valid` (que será tu `val`) y `test`.
- Finalmente, creará un archivo `custom_data.yaml` que es necesario para configurar el entrenamiento en YOLO.

Este script es un paso crucial y el más "tedioso" en la preparación de datos para la detección de objetos. Una vez que funcione correctamente, tendremos el dataset listo para ser utilizado con un modelo YOLO.

Avísame si tienes alguna duda o si encuentras algún error durante la ejecución.