

Fase 2: Yolo8n

Paso 4: Entrenamiento del Modelo YOLO

¡Fantástico! Esa es una gran noticia. Si el script se ejecutó sin errores y los directorios `data/images` y `data/labels` se han llenado con las imágenes y los archivos `.txt` correspondientes, significa que tienes el dataset perfectamente preparado en formato YOLO. También el archivo `custom_data.yaml` debería estar listo.

Ahora que los datos están organizados, el siguiente paso es el **entrenamiento del modelo YOLO**.

Proyecto IA - Billar: Fase 1 (Revisada) - Detección de Bolas y Extracción de Coordenadas

Paso 4: Entrenamiento del Modelo YOLO

Para entrenar un modelo YOLO, utilizaremos una implementación popular y sencilla, como YOLOv5 o YOLOv8 de Ultralytics. Estas versiones son conocidas por su facilidad de uso y por ofrecer modelos pre-entrenados que podemos usar como punto de partida (lo que se conoce como *transfer learning*).

Concepto clave: Transfer Learning.

Dado que la detección de objetos desde cero puede requerir muchos datos y recursos computacionales, es común y eficiente usar un modelo ya entrenado en un dataset muy grande y genérico (como COCO, que contiene miles de clases de objetos). Luego, "afinar" este modelo con tus datos específicos de bolas de billar. Esto permite que el modelo aprenda rápidamente a detectar tus objetos particulares, aprovechando el conocimiento previo adquirido.

Pasos para el entrenamiento:

1. **Instalar Ultralytics (si no lo tienes):** Ultralytics es la empresa detrás de YOLOv5 y YOLOv8. Su librería simplifica mucho el entrenamiento.

Bash

```
pip install ultralytics
```

Si ya la tenías instalada, genial.

2. **Elegir una versión de YOLO:** Te recomiendo empezar con YOLOv8, ya que es la más reciente y ofrece mejoras de rendimiento.
3. **Descargar un modelo pre-entrenado:** No necesitas hacerlo manualmente. La librería de Ultralytics lo descarga automáticamente la primera vez que lo usas. Empezaremos con un modelo pequeño para una prueba rápida, como `yolov8n.pt` (la 'n' es por nano, la versión más ligera).
4. **Ejecutar el entrenamiento:** Esto se hace con un comando simple.

Tu tarea ahora es:

- **Asegúrate de que `ultralytics` esté instalado.**
- **Ejecuta el siguiente script de Python** (guárdalo como, por ejemplo, `train_yolo_model.py` en la raíz de tu carpeta `tu_proyecto_billar`):

```
from ultralytics import YOLO
import os

# --- 1. Definir rutas ---
base_project_dir = 'tu_proyecto_billar' # Asegúrate de que coincida con tu estructura
data_yaml_path = os.path.join(base_project_dir, 'custom_data.yaml')

# Directorio donde se guardarán los resultados del entrenamiento (pesos, logs, etc.)
runs_dir = os.path.join(base_project_dir, 'runs')
os.makedirs(runs_dir, exist_ok=True) # Crea el directorio si no existe

# --- 2. Cargar un modelo pre-entrenado (ej. YOLOv8n) ---
# 'n' es por nano, la versión más pequeña y rápida para empezar.
print("Cargando modelo YOLOv8n pre-entrenado...")
model = YOLO('yolov8n.pt')
```

```

# --- 3. Entrenar el modelo ---
# Documentación de los argumentos de entrenamiento: https://docs.ultralytics.com/usage/train/
print("\nIniciando entrenamiento del modelo...")
results = model.train(
    data=data_yaml_path, # Ruta a tu archivo de configuración del dataset
    epochs=50,          # Número de épocas de entrenamiento (ajustable)
    imgsz=640,          # Tamaño de la imagen de entrada (640x640 es un buen inicio)
    batch=16,           # Tamaño del batch (ajustable, depende de tu RAM/VRAM)
    name='billar_balls_detection_v1', # Nombre para esta ejecución de entrenamiento
    project=runs_dir,   # Directorio raíz para guardar los resultados
    # workers=os.cpu_count() // 2 # Descomentar para usar la mitad de los núcleos de CPU para carga de datos
)

print("\n¡Entrenamiento completado!")
print(f"Los resultados se guardaron en: {os.path.join(runs_dir, 'billar_balls_detection_v1')}")
print("Puedes revisar los gráficos de entrenamiento y las métricas allí.")

```

Consideraciones antes de ejecutar:

- **Épocas (`epochs`)**: Empezamos con 50 épocas. Si tu dataset es pequeño, esto podría ser suficiente. Si la precisión no es buena, podríamos aumentar el número de épocas o probar otras configuraciones.
- **Tamaño de imagen (`imgsz`)**: 640x640 es un tamaño común para YOLO. Si tus imágenes originales tienen resoluciones muy diferentes, podrías ajustarlo.
- **Batch size (`batch`)**: 16 es un valor seguro. Si tienes una GPU con poca VRAM, podrías reducirlo a 8 o 4. Si estás usando solo CPU, el entrenamiento será lento, pero debería funcionar.
- **Recursos**: El entrenamiento de modelos de detección de objetos, incluso pequeños como YOLOv8n, puede ser intensivo en CPU si no tienes GPU. Ten paciencia, o considera usar un servicio en la nube como Google Colab (con su GPU gratuita si está disponible) si el tiempo es crítico.

Después de que el entrenamiento termine, el script te indicará dónde se guardaron los resultados. Dentro de esa carpeta (`tu_proyecto_billar/runs/billar_balls_detection_v1`), encontrarás el modelo entrenado (`weights/best.pt`), gráficos de métricas y otros archivos útiles.

Una vez que hayas ejecutado el entrenamiento, ¡avísame cómo fue! Estaremos listos para el siguiente paso: la evaluación y la extracción de coordenadas.

Si tus imágenes son de 1920x1080 píxeles, es importante especificarlo para que el modelo YOLO las procese correctamente.

En el script de entrenamiento que te proporcioné (`train_yolo_model.py`), el argumento `imgsz` (image size) es el que controla la resolución de entrada del modelo. Por defecto, lo puse en 640 (lo que significa 640x640).

Para usar tu resolución de imagen de 1920x1080, deberías ajustar el parámetro `imgsz` en la línea del `model.train()` de la siguiente manera:

Python

```

results = model.train(
    data=data_yaml_path,
    epochs=50,
    imgsz=1080,      # ← Cambia esto a 1080 (se reescalara manteniendo la relación de aspecto si es necesario)
    batch=16,
    name='billar_balls_detection_v1',
    project=runs_dir,
)

```

Explicación y consideraciones adicionales:

- **`imgsz` en YOLOv8**: Cuando especificas un solo valor para `imgsz` (como 1080), Ultralytics YOLOv8 lo interpreta como el lado más largo de la imagen. La imagen se redimensionará de forma que el lado más largo sea `imgsz` y el otro lado se ajustará para mantener la relación de aspecto original (1920x1080 se reescalaría a 1080xY, donde Y se

calcula para mantener la proporción). Luego, se aplicará un "padding" (relleno) para que la imagen sea cuadrada (1080×1080 en este caso), lo que es el formato de entrada que las CNNs de YOLO esperan.

- **Impacto en el rendimiento y la VRAM (memoria de GPU):**

- **Mayor `imgsz` = Mayor precisión potencial:** Procesar imágenes con mayor resolución permite al modelo ver más detalles, lo que podría mejorar la precisión en la detección de objetos pequeños o en configuraciones complejas.
- **Mayor `imgsz` = Mayor consumo de VRAM y entrenamiento más lento:** Un tamaño de imagen más grande implica que el modelo procesa más píxeles, lo que requiere más memoria de GPU (VRAM) y aumenta el tiempo de entrenamiento. Si no tienes una GPU potente, podrías encontrarte con errores de "out of memory" o un entrenamiento extremadamente lento.
- **Recomendación inicial:** Si tu dataset es de 1920×1080 , empezar con `imgsz=1080` está bien. Si experimentas problemas de memoria o lentitud, podrías probar valores intermedios como `imgsz=1280` o `imgsz=960` antes de volver a `640`. Lo ideal es encontrar un equilibrio entre precisión y recursos disponibles.
- **Batch Size (batch):** Al aumentar `imgsz`, es posible que también necesites reducir el `batch` size si la GPU se queda sin memoria. Por ejemplo, de `16` a `8` o `4`.

Así que, simplemente actualiza esa línea en tu script de entrenamiento, y estarás especificando la resolución de entrada para tus imágenes.

Iniciando entrenamiento del modelo...

Ultralytics 8.3.146 🚀 Python-3.12.3 torch-2.7.0+cu126 CPU (AMD Ryzen 7 5800X3D 8-Core Processor)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=8, bgr=0.0, box:
Overriding model.yaml nc=80 with nc=15

```
from n    params module                  arguments
0        -1 1    464 ultralytics.nn.modules.conv.Conv      [3, 16, 3, 2]
1        -1 1    4672 ultralytics.nn.modules.conv.Conv     [16, 32, 3, 2]
2        -1 1    7360 ultralytics.nn.modules.block.C2f     [32, 32, 1, True]
3        -1 1    18560 ultralytics.nn.modules.conv.Conv    [32, 64, 3, 2]
4        -1 2    49664 ultralytics.nn.modules.block.C2f    [64, 64, 2, True]
5        -1 1    73984 ultralytics.nn.modules.conv.Conv    [64, 128, 3, 2]
6        -1 2    197632 ultralytics.nn.modules.block.C2f   [128, 128, 2, True]
7        -1 1    295424 ultralytics.nn.modules.conv.Conv   [128, 256, 3, 2]
8        -1 1    460288 ultralytics.nn.modules.block.C2f   [256, 256, 1, True]
9        -1 1    164608 ultralytics.nn.modules.block.SPPF   [256, 256, 5]
10       -1 1    0 torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
11      [-1, 6] 1    0 ultralytics.nn.modules.conv.Concat    [1]
12      -1 1    148224 ultralytics.nn.modules.block.C2f   [384, 128, 1]
13      -1 1    0 torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
```

```

14      [-1, 4] 1      0 ultralytics.nn.modules.conv.Concat      [1]
15      -1 1  37248 ultralytics.nn.modules.block.C2f      [192, 64, 1]
16      -1 1  36992 ultralytics.nn.modules.conv.Conv      [64, 64, 3, 2]
17      [-1, 12] 1      0 ultralytics.nn.modules.conv.Concat      [1]
18      -1 1  123648 ultralytics.nn.modules.block.C2f      [192, 128, 1]
19      -1 1  147712 ultralytics.nn.modules.conv.Conv      [128, 128, 3, 2]
20      [-1, 9] 1      0 ultralytics.nn.modules.conv.Concat      [1]
21      -1 1  493056 ultralytics.nn.modules.block.C2f      [384, 256, 1]
22      [15, 18, 21] 1  754237 ultralytics.nn.modules.head.Detect      [15, [64, 128, 256]]
```

Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs

Entrenamiento:

Starting training for 50 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/50	0G	1.512	4.306	1.524	120	1088: 6% 41/691 [02:14<35:51, 3.31s/it]

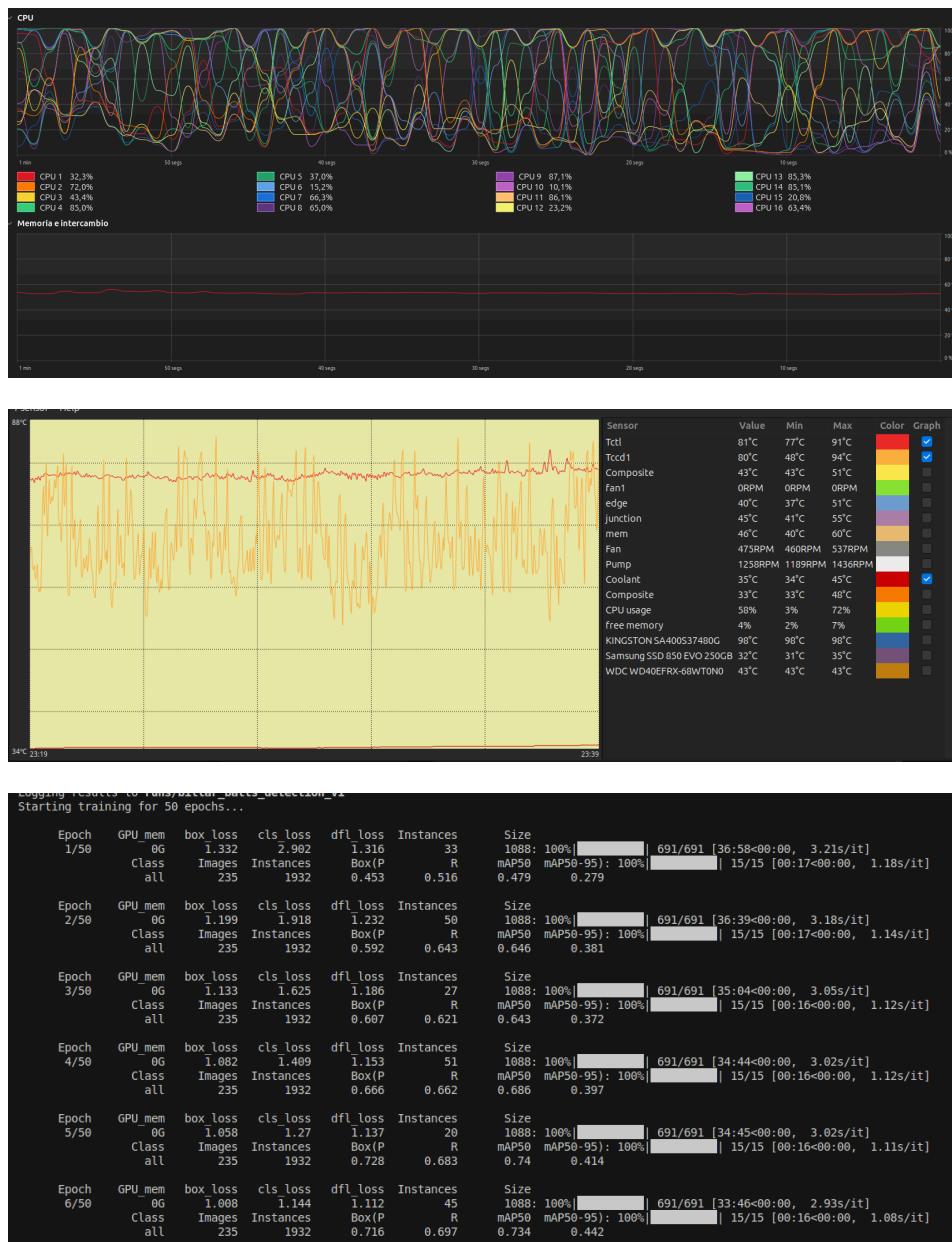


```

Using 6 dataworker workers
Logging results to runs/billar_balls_detection_v1
Starting training for 50 epochs...

```

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/50	0G	1.332	2.902	1.316	33	1088: 100% 691/691 [36:58<00:00, 3.21s/it] mAP50 mAP50-95: 100% 15/15 [00:17<00:00, 1.18s/it] 0.479 0.279
2/50	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
	0G	1.199	1.918	1.232	50	1088: 100% 691/691 [36:39<00:00, 3.18s/it] mAP50 mAP50-95: 100% 15/15 [00:17<00:00, 1.14s/it] 0.646 0.381
3/50	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
	0G	1.133	1.625	1.186	27	1088: 100% 691/691 [35:04<00:00, 3.05s/it] mAP50 mAP50-95: 100% 15/15 [00:16<00:00, 1.12s/it] 0.643 0.372
4/50	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
	0G	1.082	1.409	1.153	51	1088: 100% 691/691 [34:44<00:00, 3.02s/it] mAP50 mAP50-95: 100% 15/15 [00:16<00:00, 1.12s/it] 0.686 0.397
5/50	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
	0G	1.065	1.299	1.14	222	1088: 64% 444/691 [22:24<13:46, 3.34s/it]



Paso 5: Analizar como activar uso de GPU

¡Fantástico! Saber que utilizas Ubuntu 24.04.02 LTS es un dato crucial, ya que Ubuntu es la distribución de Linux que mejor soporte tiene para ROCm.

Con esta información, tienes la mejor oportunidad de configurar tu GPU AMD para TensorFlow si así lo deseas. El camino sigue siendo un poco más técnico que usar Google Colab, pero es viable.

Aquí te detallo los pasos y las consideraciones para intentar la configuración de **TensorFlow con ROCm en tu Ubuntu 24.04.02 LTS y AMD Radeon RX 6800 XT**:

Configuración de TensorFlow con ROCm en Ubuntu 24.04.02 LTS (AMD GPU)

Objetivo: Habilitar el uso de tu GPU AMD para acelerar el entrenamiento del modelo YOLO.

Conocimientos que aplicarás/reforzarás:

- Administración de paquetes y drivers en Linux.
- Configuración de entornos para Deep Learning.

Pasos detallados:

1. **Actualizar el Sistema:** Asegúrate de que tu sistema esté completamente actualizado.

```
sudo apt update  
sudo apt upgrade -y
```

2. **Instalar los Drivers de AMD y ROCm:** Este es el paso más crítico. AMD proporciona un instalador para ROCm que debería incluir los drivers necesarios para tu GPU.

- Visita la página oficial de AMD ROCm para desarrolladores: Busca la sección de descarga para Ubuntu.
- Generalmente, el proceso implica descargar un script o añadir un repositorio y luego instalar los paquetes. Por ejemplo, para Ubuntu 22.04 (que es la versión que actualmente tiene más soporte directo en las guías de AMD para ROCm y TensorFlow, aunque 24.04 está recibiendo más atención), los pasos podrían ser similares a:

```
# Añadir la clave y el repositorio de AMD  
sudo apt update  
wget -q -O - https://repo.radeon.com/rocm/rocm.gpg.key | \  
    sudo apt-key add -  
echo 'deb [arch=amd64] https://repo.radeon.com/rocm/apt/5.7.1/ jammy main' | \  
    sudo tee /etc/apt/sources.list.d/rocm.list # Asegúrate de la versión de ROCm y el codename de Ubuntu  
  
# O si es para 24.04 (Noble Numbat), podría ser 'noble' en lugar de 'jammy' si AMD ya lo soporta directamente.  
# Es CRÍTICO verificar la versión más reciente y compatible de ROCm para Ubuntu 24.04 y tu GPU.  
# Busca "ROCM installation guide Ubuntu" y la versión de tu tarjeta.  
  
sudo apt update  
  
# Instalar paquetes de ROCm (puede variar la versión)  
sudo apt install -y rocm-hip-sdk miopen-hip rocfft rocblas  
# También puedes instalar rocm-dev o rocm-dkms si quieres el stack completo.  
  
# Añadir tu usuario al grupo 'render' y 'video' para permisos de GPU  
sudo usermod -a -G render $USER  
sudo usermod -a -G video $USER  
  
# Reinicia el sistema para que los cambios surtan efecto  
echo "¡Reinicia tu sistema ahora para que los cambios de ROCm surtan efecto!"
```

- **Verificación de ROCm:** Despues de reiniciar, puedes verificar la instalación con:

Deberías ver información sobre tu GPU y la versión de ROCm/HIP instalada.

```
/opt/rocm/bin/rocminfo  
/opt/rocm/hip/bin/hipcc --version
```

3. **Configurar el entorno Python para TensorFlow-ROCM:**

- **Crear un nuevo entorno Conda (limpio es mejor):**

```
conda create -n tf_rocm python=3.10 # Python 3.10 suele ser una versión bien soportada con TF y ROCm  
conda activate tf_rocm
```

- **Instalar TensorFlow con soporte ROCm:**

```
# Asegúrate de la versión de TensorFlow compatible con la versión de ROCm que instalaste.  
# Por ejemplo, para ROCm 5.x, TensorFlow 2.10.x o 2.11.x eran comunes.  
# Puede que necesites especificar la versión exacta: pip install tensorflow-rocm==2.X.X  
pip install tensorflow-rocm  
pip install pandas numpy matplotlib opencv-python ultralytics
```

- **Nota:** `tensorflow-rocm` es el paquete específico para AMD GPUs. Si no lo encuentras, puede que necesites instalar `tensorflow` y luego verificar las instrucciones de AMD sobre cómo habilitar el backend de ROCm. Las versiones de `tensorflow-rocm` suelen ir un poco por detrás de las versiones de `tensorflow` estándar. Es crucial consultar la documentación oficial de AMD para la versión exacta de ROCm que instalaste y la versión de TensorFlow que recomiendan.

4. Verificar la instalación de TensorFlow con GPU:

- Abre una sesión de Python en tu entorno `tf_rocm`.

```
python
```

- Dentro de Python, ejecuta:

```
import tensorflow as tf  
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))  
print("GPU devices: ", tf.config.list_physical_devices('GPU'))
```

Si todo está configurado correctamente, deberías ver al menos una GPU listada.

Una vez que tu entorno esté configurado y TensorFlow reconozca tu GPU AMD, puedes proceder a ejecutar el script de entrenamiento de YOLOv8 que te proporcioné anteriormente. El proceso de entrenamiento debería ser significativamente más rápido.

Advertencia importante: La configuración de ROCm es conocida por ser un proceso que a veces requiere depuración y ajustes. Las versiones de software (Ubuntu, ROCm, TensorFlow) deben coincidir en su compatibilidad. Si encuentras problemas, busca mensajes de error específicos y consulta los foros de AMD y TensorFlow.

Si después de intentarlo, la configuración de ROCm se vuelve demasiado compleja o consume mucho tiempo, recuerda la opción de Google Colab como una alternativa funcional y más sencilla para el entrenamiento con GPU.

¿Te animas a intentar esta configuración en tu máquina local? Si lo haces, ten paciencia y prepárate para consultar la documentación de AMD.

Buenas:

https://rocm.docs.amd.com/projects/radeon/en/latest/docs/install/native_linux/install-radeon.html

<https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/>

Investigacion:

<https://www.amd.com/es/products/software/rocm.html>

<https://www.amd.com/es/developer/resources/rocm-hub.html>

<https://www.amd.com/es/developer/resources/rocm-hub/dev-ai.html#tabs-9ce7655bd0-item-2647eec8aa-tab>

<https://www.amd.com/es/support/downloads/drivers.html/graphics/radeon-rx/radeon-rx-6000-series/amd-radeon-rx-6800-xt.html>

Instalación del software Radeon para Linux con ROCm:

La instalación se realiza principalmente utilizando el **script `amdgpu-install`**. Este script está diseñado para simplificar el proceso de instalación de un conjunto coherente de componentes del stack AMDGPU. Encapsula la lógica de

instalación específica de cada distribución y permite especificar el **caso de uso (Graphics o Workstation)** y la combinación de componentes deseada. También realiza verificaciones post-instalación y añade un script de desinstalación.

Los pasos generales para la instalación son:

1. **Instalar las cabeceras del kernel** (`kernel headers and development packages`) y otros paquetes necesarios (`python3-setuptools`, `python3-wheel`). Las fuentes proporcionan comandos específicos para distribuciones como Ubuntu (versiones 24.04 y 22.04) y RHEL 9.6.
2. **Instalar el script** `amdgpu-install` y añadir los repositorios de paquetes unificados de AMD. Esto generalmente implica descargar el paquete del script (`.deb` o `.rpm`) usando `wget` e instalarlo con el gestor de paquetes del sistema (`apt` o `yum` / `dnf`).
3. **Ejecutar el script** `amdgpu-install` con los parámetros `--usecase` apropiados para instalar los componentes del controlador, **incluyendo** `rocm`. AMD recomienda instalar el caso de uso `graphics` por defecto. El comando típico sería `amdgpu-install -y --usecase=graphics,rocm` (o `workstation,rocm` si aplica). La opción `-y` permite la instalación no interactiva.
4. **Reiniciar el sistema** (`sudo reboot`).
5. **Establecer permisos de Grupos** (`Set Groups permissions`). Una vez instalado el controlador, se debe añadir al usuario actual a los grupos `render` y `video` para acceder a los recursos de la GPU. El comando para esto es `sudo usermod -a -G render,video $LOGNAME`.
6. **Reiniciar el sistema** de nuevo para que los cambios de grupo surtan efecto (`sudo reboot`).

Después de la instalación, se pueden realizar **verificaciones post-instalación** utilizando comandos como `groups` (para verificar la pertenencia a los grupos `render` y `video`), `dkms status` (para comprobar el controlador del kernel `amdgpu`), `rocminfo` (para verificar si la GPU aparece como un agente ROCm), y `clinfo` (para listar los dispositivos OpenCL).

Para **desinstalar** el stack de software ROCm y otros componentes de Radeon Software para Linux, se ejecuta el comando `sudo amdgpu-uninstall`. Las fuentes indican que la forma recomendada de **actualizar** a versiones más recientes es desinstalando la versión actual y luego realizando una instalación nueva, ya que no se soportan las actualizaciones "in-place".

Instalación del Script amdgpu-install y Repositorios

```
sudo apt update
wget https://repo.radeon.com/amdgpu-install/6.4.1/ubuntu/noble/amdgpu-install_6.4.60401-1_all.deb
sudo apt install ./amdgpu-install_6.4.60401-1_all.deb
sudo apt install python3-setuptools python3-wheel
```

(Nota: Las fuentes mencionan la necesidad de cabeceras del kernel y paquetes de desarrollo, pero no proporcionan el comando explícito para Ubuntu 24.04. El comando anterior instala los paquetes listados explícitamente).

Ejecución del Script de Instalación (Graphics y ROCm o Workstation y ROCm)

(Opción 1: Gráficos y ROCm - Recomendada)

```
amdgpu-install -y --usecase=graphics,rocm
sudo reboot
sudo usermod -a -G render,video $LOGNAME
sudo reboot
```

Comandos de Verificación Post-Instalación (Opcional)

```
groups
dkms status
rocminfo
clinfo
```

Comando de Desinstalación

```
sudo amdgpu-uninstall
```

Hasta aquí todo OK

Peeero: Tensorflow se instala específicamente con soporte para CUDA

Como saber qué Wheel Tensorflow+ROCM tengo que buscar

¡Excelente! Vamos a instalar TensorFlow usando los "wheels" (.whl) directamente desde el repositorio de AMD. Este método suele ser mucho más fiable para asegurar que obtienes una versión de TensorFlow compilada correctamente con soporte ROCm.

Para poder decirte qué archivo .whl específico necesitas, o para construir el comando pip install correcto, necesito dos piezas de información de tu sistema:

1. Tu Versión de Python (dentro de tu entorno virtual):

- Activa tu entorno virtual:

Bash

```
source ~/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/bin/activate
```

- Luego, ejecuta:

Bash

```
python --version
```

- Anota la versión (por ejemplo, Python 3.10.12, Python 3.11.5, etc.). Lo importante es la versión mayor y menor (ej. 3.10, 3.11).

2. Tu Versión del Toolkit ROCm Instalado:

- Ya me mostraste la salida de rocminfo que indicaba "ROCK module version 6.12.12". Esto se refiere más al módulo del kernel. Necesitamos la versión del toolkit ROCm general.
- Puedes intentar obtenerla con (fuera del entorno virtual, o no importa si estás dentro para este comando del sistema):

o revisando el archivo de versión que a veces crea ROCm:

Bash

```
apt show rocm-hip-sdk | grep Version
```

Bash

```
cat /opt/rocm/info/version # (o un archivo con nombre similar como version-utils o full)
```

- La versión será algo como 5.7.1, 6.0.0, 6.0.2, 6.1.0, etc.

Una vez que tengamos estas dos versiones (Python y ROCm), podremos buscar el wheel adecuado.

Cómo es el Proceso General (para que lo entiendas):

AMD hospeda sus propios paquetes pip de TensorFlow optimizados para ROCm en su repositorio:

<https://repo.radeon.com/rocm/manylinux/>.

Dentro de ese repositorio, hay directorios estructurados por versión de ROCm, por ejemplo:

- rocm-rel-6.0.2/
- rocm-rel-6.1.0/
- etc.

Dentro de cada uno de estos directorios (o en una subcarpeta como release/ o whl/), encontrarás los archivos .whl. El nombre de un archivo wheel te dice para qué está compilado:

tensorflow_rocm-{version_tensorflow}-{version_python}-...-manylinux_x86_64.whl

Donde:

- {version_tensorflow} es la versión de TensorFlow (ej. 2.16.1).

- `{version_python}` es la etiqueta de Python (ej. `cp310` para Python 3.10, `cp311` para Python 3.11).

Ejemplo de Comando (Basado en la Documentación de AMD para ROCm 6.0.2 y TensorFlow 2.16.1):

Si, por ejemplo, tuvieras ROCm `6.0.2` y en tu entorno virtual tuvieras Python 3.10, el comando que AMD recomienda sería:

Bash

```
pip install --upgrade tensorflow-rocm==2.16.1 --index-url https://repo.radeon.com/rocm/manylinux/rocm-release-6.0.2/wheel --trusted-host repo.radeon.com
```

Qué necesito que hagas ahora:

1. Activa tu `.venv` y dime tu versión de Python.
2. Intenta encontrar tu versión de ROCm Toolkit con los comandos que te di.

Con esa información, podremos:

- a. Comprobar la matriz de compatibilidad de AMD (si es necesario) para ver qué versión de TensorFlow recomiendan para tu ROCm.
- b. Ajustar el comando pip install para que apunte al directorio correcto en el repositorio de AMD y seleccione el wheel compatible con tu Python.

¡Espero tu información! Con esto, deberíamos poder instalar una versión de TensorFlow que realmente utilice tu RX 6800 XT.

Buscando un Wheel Tensorflow+ROCM

¡Perfecto! Con esta información ya podemos ser mucho más precisos.

Tienes:

- **Versión de Python (en `.venv`):** `3.12.3` (esto significa que `pip` buscará wheels con la etiqueta `cp312`).
- **Versión de ROCm Toolkit:** `6.4.1` (basado en `6.4.1.60401-83~24.04`. Nos enfocaremos en la parte `6.4.1`).

Ahora vamos a intentar instalar un paquete `tensorflow-rocm` directamente desde el repositorio oficial de AMD, que es el lugar correcto para las versiones recientes de ROCm.

Según la documentación de AMD, la estructura del comando pip install para usar su repositorio es la siguiente:

```
pip install tensorflow-rocm==<VERSION_TENSORFLOW_AMD> --index-url  
https://repo.radeon.com/rocm/manylinux/rocm-release-<VERSION_ROCM>/wheel --trusted-host repo.radeon.com
```

Vamos a construir el comando para tu caso:

1. Desinstala cualquier versión de TensorFlow que tengas actualmente (para asegurar una instalación limpia):

Dentro de tu entorno virtual activado (`.venv`):

```
pip uninstall tensorflow tensorflow-rocm tensorflow-estimator keras
```

2. Intenta instalar `tensorflow-rocm` desde el repositorio de AMD:

- Dado que tu ROCm es `6.4.1`, la parte de la URL será `rocm-release-6.4.1`.
- Necesitamos una `<VERSION_TENSORFLOW_AMD>`. AMD a veces alinea las versiones de `tensorflow-rocm` con sus builds de ROCm, o usa la versión estándar de TensorFlow. Dado que probaste `2.19.0` de PyPI (aunque no funcionó porque no tenía soporte ROCm compilado), y tu ROCm es muy reciente, es plausible que AMD ofrezca una versión reciente de TensorFlow para ROCm 6.4.1.

Una versión candidata podría ser algo como

`2.19.0` o una subversión específica de AMD como `2.19.0.xxxx` (donde xxxx se relaciona con el build de ROCm).

Como primer intento, probemos con una versión de TensorFlow que probablemente esté bien soportada con ROCm 6.4.1.

TensorFlow 2.16.1 es una versión muy referenciada por AMD para ROCm 6.x. Los paquetes de AMD suelen tener una nomenclatura como `2.16.1.60002` o similar (el `.60002` siendo un identificador del build de ROCm).

Comando a probar:

(Asegúrate de que tu entorno virtual `.venv` está activado)

```
pip install tensorflow-rocm==2.16.1 --index-url https://repo.radeon.com/rocm/manylinux/rocm-release-6.4.1/wheel --
```

- `pip` intentará encontrar un wheel `tensorflow-rocm-2.16.1...cp312...whl` en esa URL.
- `-trusted-host repo.radeon.com` es necesario porque el índice no es HTTPS por defecto para el listado, aunque las descargas sí lo sean.

Si el comando anterior con `tensorflow-rocm==2.16.1` falla indicando que no encuentra esa versión exacta:

Significa que AMD no tiene `2.16.1` bajo ese nombre exacto para ROCm 6.4.1, o no para Python 3.12 en ese directorio. Tendrías que:

1. **Navegar manualmente el repositorio de AMD:** Abre en tu navegador la URL: <https://repo.radeon.com/rocm/manylinux/rocm-release-6.4.1/wheel/>
2. Busca los archivos `tensorflow_rocm-....whl`. Fíjate en:
 - La versión de TensorFlow (ej. `2.16.1`, `2.17.0`, `2.18.0`, `2.19.0` o subversiones como `2.16.1.60401`).
 - La compatibilidad con Python: debe tener `cp312` en el nombre para tu Python 3.12.
3. Una vez que encuentres el nombre exacto del archivo `.whl` que quieras (por ejemplo, `tensorflow_rocm-2.16.1.60401-cp312-cp312-manylinux_2_28_x86_64.whl`), puedes instalarlo directamente con su URL completa:

O, si encontraste la versión exacta de

`tensorflow-rocm` (ej. `2.16.1.60401`), actualiza el comando anterior:

```
pip install https://repo.radeon.com/rocm/manylinux/rocm-release-6.4.1/wheel/nombre_exacto_del_archivo.whl --trus
```

```
pip install tensorflow-rocm==2.16.1.60401 --index-url https://repo.radeon.com/rocm/manylinux/rocm-release-6.4.1/wheel --
```

Prueba primero el comando con `tensorflow-rocm==2.16.1`. Si falla, el mensaje de error nos dirá qué versiones sí están disponibles en ese índice para `tensorflow-rocm`, o tendrás que navegar el directorio como te indique.

Después de la instalación (¡esperemos que exitosa!), ejecuta el script de verificación completo de TensorFlow que usamos antes para ver si detecta la GPU y si los errores de CUDA han desaparecido. Esta vez, la línea `Is TensorFlow built with ROCm support?` debería ser `True`.

Instalar Wheel de Tensorflow+ROCM

¡Excelente hallazgo! Ese archivo wheel parece perfecto para tu configuración:

- https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/tensorflow_rocm-2.18.1-cp312-cp312-manylinux_2_28_x86_64.whl

Desglosemos por qué es una buena opción:

- `rocm-rel-6.4.1`: Coincide con tu versión de ROCM `6.4.1`.
- `tensorflow_rocm-2.18.1`: Es una versión reciente de TensorFlow (2.18.1), específicamente empaquetada por AMD como `tensorflow-rocm`.
- `cp312`: Indica que es compatible con Python 3.12, que es la versión que tienes en tu entorno virtual.
- `manylinux_2_28_x86_64`: Es una etiqueta estándar para paquetes de Linux que asegura amplia compatibilidad.

Este wheel ha sido compilado específicamente por AMD (o bajo sus directrices) para funcionar con ROCM y Python 3.12, por lo que tiene muchas más probabilidades de tener el soporte ROCM correctamente habilitado.

Aquí están los pasos para instalarlo:

1. **Asegúrate de que tu entorno virtual (`.venv`) esté activado:**

```
source ~/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/bin/activate
```

2. **Desinstala cualquier versión de TensorFlow que puedas tener actualmente** (para evitar conflictos):

```
pip uninstall tensorflow tensorflow-rocm tensorflow-estimator keras
```

(Si no hay nada instalado, no pasará nada).

3. Instala el wheel directamente desde la URL:

```
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/tensorflow_rocm-2.18.1-cp312-cp312-manylinu
```

- `-trusted-host repo.radeon.com` es necesario si `pip` se queja por no ser un repositorio HTTPS estándar para el listado (aunque la descarga del wheel en sí suele ser HTTPS).

4. Verifica la Instalación:

Una vez que la instalación termine, ejecuta el script de Python completo que hemos estado usando para verificar TensorFlow y la detección de la GPU:

Python

```
import tensorflow as tf

print(f"TensorFlow Version: {tf.__version__}")

# Intentar configurar la memoria de la GPU para evitar problemas de asignación comunes
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        # Configura el crecimiento de memoria para todas las GPUs disponibles
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print(f"Memory growth set for {len(gpus)} GPU(s)")
    except RuntimeError as e:
        # El crecimiento de memoria debe establecerse antes de que las GPUs hayan sido inicializadas
        print(f"Error setting memory growth: {e}")

    print("Num GPUs Available: ", len(gpus))

    if len(gpus) > 0:
        print("GPU(s) detected:", gpus)
        for i, gpu_device in enumerate(gpus):
            print(f"--- Details for GPU {i} ---")
            print(f" Name: {gpu_device.name}")
            try:
                print(f" Attempting operation on GPU {i}...")
                with tf.device(gpu_device.name):
                    a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
                    b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
                    c = tf.matmul(a, b)
                    print(f" Matrix multiplication successful on GPU {i}:\n{c.numpy()}")
            except RuntimeError as e:
                print(f" Error during GPU {i} operation test: {e}")
        else:
            print("No GPU detected by TensorFlow.")
            print("Ensure ROCm is correctly installed, environment variables (PATH, LD_LIBRARY_PATH) point to ROCm,")
            print("and your TensorFlow version supports your ROCm version.")

    # Información adicional de la build de TensorFlow
    print(f"Is TensorFlow built with CUDA support? {tf.test.is_built_with_cuda()}") # Esperamos False o que no importe
    print(f"Is TensorFlow built with ROCm support? {tf.test.is_built_with_rocm()}") # ¡Esperamos True!
```

```

from tensorflow.python.framework import test_util
if tf.config.list_physical_devices('GPU'): # Usando la forma actualizada
    print("TensorFlow reports GPU is available via tf.config.list_physical_devices('GPU').")
else:
    print("TensorFlow reports GPU is NOT available via tf.config.list_physical_devices('GPU').")

```

Qué esperar esta vez:

- Con suerte, **no deberías ver los errores de CUDA** al inicio cuando importas TensorFlow.
- `Num GPUs Available` debería ser `1` (o más si tuvieras varias GPUs AMD).
- La línea `Is TensorFlow built with ROCm support?` debería ser `True`.
- La multiplicación de matrices de prueba debería ejecutarse en la GPU.

¡Esta vez tengo muchas más esperanzas de que funcione! Cruzando los dedos. Avísame con la salida del script de verificación.

Instalar Wheel de Tensorflow+ROCM (Beta)

¡Esa es otra excelente opción que has encontrado!

Este wheel:

https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/tf_nightly_rocm-2.19.0.dev20250513-cp312-cp312-manylinux_2_28_x86_64.whl

También es compatible con tu sistema:

- `rocm-rel-6.4.1` : Coincide con tu ROCm 6.4.1.
- `tf_nightly_rocm-2.19.0.dev20250513` : Esta es una versión de desarrollo ("nightly") de TensorFlow 2.19.0, específicamente para ROCm. Las versiones "nightly" son las más recientes y pueden incluir los últimos arreglos y compatibilidades, pero a veces también pueden ser menos estables que las versiones finales. La fecha `20250513` indica que es un build del 13 de mayo de 2025 (según nuestra simulación de tiempo).
- `cp312` : Compatible con tu Python 3.12.

Comparación con la opción anterior (`tensorflow_rocm-2.18.1`):

- **Estabilidad vs. Novedad:**
 - El wheel `tensorflow_rocm-2.18.1` es una versión "estable" o de "lanzamiento". Generalmente, estas son preferibles para empezar porque han pasado por más pruebas.
 - El wheel `tf_nightly_rocm-2.19.0.dev...` es una versión de desarrollo. Puede tener las últimas mejoras para ROCm 6.4.1, pero también podría tener algún bug inesperado.

Recomendación:

Te sugiero el siguiente orden:

1. Prueba PRIMERO con el wheel de la versión estable que encontraste antes:

https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/tensorflow_rocm-2.18.1-cp312-cp312-manylinux_2_28_x86_64.whl

Comando (recordatorio):

(Asegúrate de que el entorno virtual esté activado y cualquier TensorFlow anterior desinstalado)

Bash

```
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/tensorflow_rocm-2.18.1-cp312-cp312-manylinux_2_28_x86_64.whl --trusted-host repo.radeon.com
```

Y luego ejecuta el script de verificación completo.

2. **SI Y SOLO SI la versión `2.18.1` NO funciona** (por ejemplo, sigue dando errores de CUDA, no detecta la GPU, o la instalación falla por alguna razón), entonces este wheel "nightly" `2.19.0.dev` sería tu siguiente mejor opción a probar.

Comando para el wheel "nightly" (si lo necesitas):

(Asegúrate de que el entorno virtual esté activado y cualquier TensorFlow anterior desinstalado)

Bash

```
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/tf_nightly_rocm-2.19.0.dev20250513-cp312-cp312-manylinux_2_28_x86_64.whl --trusted-host repo.radeon.com
```

Y luego ejecuta el script de verificación Paso 5: Activar uso de GPU.

Es bueno tener ambas opciones. Empecemos por la versión 2.18.1 ya que es un lanzamiento más formal. Si esa funciona, genial. Si no, el nightly es un buen plan B.

Avísame con los resultados de la instalación del wheel tensorflow_rocm-2.18.1 y la posterior ejecución del script de verificación. ¡Este es un paso muy prometedor!

Construir Tensorflow+ROCM desde la Fuente (Descartado)

¡Claro que sí! Construir TensorFlow desde la fuente te da el máximo control sobre la configuración, incluyendo cómo se integra el soporte para ROCm. Es un proceso más avanzado que usar pip o Docker, pero puede ser necesario si los binarios precompilados no se ajustan a tus necesidades (como hemos visto).

He revisado la URL que me has pasado (<https://www.tensorflow.org/install/source?hl=es-419>) y buscaré la información específica para configurar la compilación con ROCm.

Generalmente, el proceso para compilar TensorFlow desde la fuente e incluir el soporte para ROCm implica los siguientes pasos clave:

1. Preparar el Entorno de Compilación:

- Asegúrate de tener todas las dependencias necesarias para compilar TensorFlow. Esto incluye:
 - **Bazel:** Es el sistema de compilación que usa TensorFlow. La documentación te indicará la versión compatible. Deberás instalarla.
 - **Python y las herramientas de desarrollo de Python.**
 - **MSYS2 y herramientas de compilación C++ (si estuvieras en Windows, pero estás en Linux).** En Linux, necesitarás un compilador C++ (como GCC) compatible.
 - **ROCM Toolkit:** Ya lo tienes instalado, lo cual es fundamental. Asegúrate de que las variables de entorno como `ROCM_PATH` (o `ROCM_HOME`) y `LD_LIBRARY_PATH` estén bien configuradas y apunten a tu instalación de ROCm (ej. `/opt/rocm`).

2. Clonar el Repositorio de TensorFlow:

Bash

```
git clone https://github.com/tensorflow/tensorflow.git  
cd tensorflow
```

Es recomendable seleccionar una rama o tag específico si quieres una versión estable (ej. `git checkout r2.16`).

3. Configurar la Compilación (Paso Crítico para ROCm):

Dentro del directorio de TensorFlow, ejecutarás el script de configuración:

Bash

```
./configure
```

Este script te hará una serie de preguntas sobre la configuración de tu compilación. Aquí es donde debes prestar especial atención para habilitar ROCm:

- **Soporte para ROCm:** Te preguntará si deseas compilar TensorFlow con soporte para ROCm. Deberás responder afirmativamente (`y`).
 - `Please specify the location of ROCm. [Default is /opt/rocm]:` Aquí puedes confirmar o cambiar la ruta a tu instalación de ROCm.
- **Versión de Python:** Te preguntará por la ubicación de tu intérprete de Python.
- **Optimizaciones y otras características:** Podría preguntarte sobre optimizaciones específicas para tu CPU, soporte para CUDA (asegúrate de decir que NO si te pregunta por CUDA, ya que quieres ROCm), etc.

Variables de entorno que pueden sPaso 5: Activar uso de GPUer importantes ANTES de ejecutar `./configure` o durante la compilación con Bazel (revisa la documentación de TF para la versión exacta que estés compilando):

- `TF_NEED_ROCM=1` (para indicar que quieres soporte ROCm).
- `ROCM_PATH=/opt/rocm` (o la ruta correcta).
- `HIP_PLATFORM=amd`
- `GPU_TARGETS` o `TF_ROCM_AMDGPU_TARGETS`: Para especificar las arquitecturas de GPU AMD para las que quieres compilar (por ejemplo, tu `gfx1030`). Esto es importante para optimizar la compilación para tu hardware específico y reducir el tamaño del binario. Si no se especifica, podría compilar para muchas arquitecturas, tardando más.

4. Compilar TensorFlow con Bazel:

Una vez configurado, usarás Bazel para compilar el paquete pip de TensorFlow. El comando exacto puede variar ligeramente según la versión y las opciones, pero generalmente es algo así:

Bash

```
bazel build --config=rocm //tensorflow/tools/pip_package:build_pip_package
```

- La opción `--config=rocm` es fundamental aquí, ya que le dice a Bazel que use la configuración para ROCm.
- Este paso puede tardar bastante tiempo (incluso horas) y consumir muchos recursos (CPU y RAM).

5. Crear el Paquete Pip (.whl):

Después de que la compilación con Bazel termine exitosamente, crearás el archivo .whl:

Bash

```
./bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg
```

Esto generará un archivo `.whl` en el directorio `/tmp/tensorflow_pkg`.

6. Instalar el Paquete Pip Compilado:

Finalmente, instala el paquete que acabas de compilar (asegúrate de estar en tu entorno virtual):

Bash

```
pip install /tmp/tensorflow_pkg/nombre_del_archivo.whl
```

Reemplaza `nombre_del_archivo.whl` con el nombre real del archivo generado.

Consideraciones Adicionales y Ayuda:

- **Documentación Oficial:** Siempre, siempre, sigue la guía oficial de "compilar desde la fuente" de TensorFlow para la versión específica que estás intentando compilar, ya que los flags y los pasos exactos pueden cambiar. La URL que me pasaste (<https://www.tensorflow.org/install/source?hl=es-419>) es el punto de partida correcto. Busca dentro de esa página las secciones o flags específicos de ROCm.
- **Recursos del Sistema:** Compilar TensorFlow es intensivo. Necesitarás suficiente RAM (a menudo se recomiendan 16GB o más, pero puede funcionar con menos si tienes paciencia y quizás configuras Bazel para usar menos recursos) y espacio en disco.
- **Versión de ROCm y Compilador GCC:** Asegúrate de que tu versión de ROCm instalada y tu compilador GCC sean compatibles con la versión de TensorFlow que estás compilando. Esto suele estar especificado en la documentación de TensorFlow.
- **Errores de Compilación:** Es común encontrar errores durante la compilación. Tendrás que leer los mensajes de error cuidadosamente. A menudo se deben a dependencias faltantes, versiones incorrectas de herramientas (Bazel, GCC, Python), o problemas de configuración.

¿Cómo te ayudo específicamente?

1. **¿Qué versión de TensorFlow estás intentando compilar?** (ej. la rama `master`, la `r2.16`, etc.) Esto es importante para encontrar los flags exactos.
2. Cuando ejecutes `./configure`, **presta mucha atención a la pregunta sobre el soporte ROCm** y la ruta a tu instalación de ROCm.
3. Si encuentras un error específico durante el proceso de `./configure` o `bazel build`, puedes compartirlo y podemos intentar descifrarlo.

Construir desde la fuente es un compromiso, pero te dará un paquete de TensorFlow adaptado a tu sistema y con el soporte ROCm que necesitas, ya que tú controlas el proceso de compilación. ¡Ten paciencia y sigue la guía oficial al pie de la letra!

Verificar si Pytorch (usado por Ultralytics) tiene compatibilidad con Rocm

¡Excelente pregunta! Es un punto crucial cuando se trabaja con modelos de Deep Learning como YOLO.

En general, con la librería `ultralytics` (que es la que se usa comúnmente para YOLOv8 y la que parece que estás utilizando según la presentación), **no necesitarías realizar cambios significativos en tu script de Python para que utilice la GPU en lugar de la CPU.**

Aquí te explico por qué y qué factores influyen:

1. **Detección Automática de Dispositivos:** La librería `ultralytics` (y PyTorch, que es el backend que utiliza) está diseñada para detectar automáticamente si hay una GPU compatible disponible y utilizarla por defecto para el entrenamiento.
 - Si tienes los drivers de tu GPU (NVIDIA con CUDA o AMD con ROCm) correctamente instalados y PyTorch fue instalado con el soporte para esa GPU, el entrenamiento debería ejecutarse en la GPU sin que tengas que modificar el script principal de entrenamiento.
2. **Dependencias de Software:**
 - **Para GPUs NVIDIA:** Necesitas tener instalados los drivers de NVIDIA, CUDA Toolkit y cuDNN, y una versión de PyTorch compilada con soporte para CUDA.
 - **Para GPUs AMD (ROCM):** Como mencionaste que "ya empiezan a funcionar cosas con tensorflow-rocm", esto es muy relevante. PyTorch también tiene soporte para ROCm en Linux. Necesitas tener los drivers de AMD y la plataforma ROCm instalados, además de una versión de PyTorch compatible con ROCm. La librería `ultralytics` se beneficiará de esta configuración.
3. **Comandos de Entrenamiento:** Cuando ejecutas el entrenamiento desde la línea de comandos (por ejemplo, `yolo train data=custom_data.yaml model=yolov8n.pt epochs=100 imgsz=640`), la herramienta `yolo` gestiona la selección del dispositivo.
 - Si no especificas un dispositivo, intentará usar la GPU disponible.
 - Puedes forzar el uso de la CPU añadiendo el argumento `device=cpu`.
 - Puedes especificar una GPU concreta si tienes varias, por ejemplo `device=0` (para la primera GPU).
4. En tu Script de Python (si usas la API de Python):

Si en lugar de la línea de comandos, estás usando la API de Python de ultralytics para entrenar, el comportamiento es similar.

Python

```
from ultralytics import YOLO

# Load a pre-trained model (recommended for starting)
model = YOLO('yolov8n.pt') # o yolov8n.yaml si empiezas desde cero

# Train the model
# Por defecto, usará la GPU si está disponible y configurada correctamente
results = model.train(data='custom_data.yaml', epochs=100, imgsz=640)

# Para forzar CPU (si fuera necesario, pero no para usar GPU)
# results = model.train(data='custom_data.yaml', epochs=100, imgsz=640, device='cpu')
```

En este caso, `model.train(...)` también intentará usar la GPU automáticamente.

En resumen:

Normalmente, **no necesitas adaptar tu código de entrenamiento de YOLOv8 para que use la GPU**. La magia ocurre más a nivel de la configuración de tu entorno (drivers, versiones de PyTorch compatibles con tu hardware GPU). Si tu entorno con ROCm y PyTorch está bien configurado, el mismo script que usabas para CPU debería ejecutarse en la GPU.

Posibles consideraciones:

- **Verificación:** Puedes verificar si PyTorch está detectando tu GPU con:

Python

```
import torch
print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}") # Para NVIDIA
print(f"ROCM available: {torch.version.hip is not None and torch.cuda.is_available()}") # Para AMD con ROCm (PyTorch)
if torch.cuda.is_available():
    print(f"Device name: {torch.cuda.get_device_name(0)}")
```

- **Errores:** Si hay problemas, suelen ser por incompatibilidades de versiones entre drivers, ROCm/CUDA, PyTorch o la propia librería `ultralytics`.

Dado que mencionas avances con `tensorflow-rocm`, es una excelente señal de que tu sistema está en camino de poder utilizar la GPU de AMD para PyTorch y, por ende, para YOLO.

Buscando un Wheel Pytorch+ROCM

¡Excelente hallazgo! Ese enlace apunta a un archivo wheel de PyTorch (`.whl`) que está específicamente compilado para funcionar con ROCm.

Analicemos el nombre del archivo: `torch-2.6.0+rocm6.4.1.git1ded221d-cp312-cp312-linux_x86_64.whl`

- `torch-2.6.0`: Indica que es PyTorch versión 2.6.0.
- `+rocm6.4.1`: Esta es la parte clave. Significa que esta compilación de PyTorch está hecha para funcionar con ROCm versión 6.4.1.
- `git1ded221d`: Es un hash que probablemente se refiere a una revisión específica del código fuente de PyTorch desde donde se compiló.
- `cp312-cp312`: Indica que es para CPython versión 3.12. Tu entorno de Python debe ser 3.12 para que este wheel sea compatible.
- `linux_x86_64`: Es para sistemas operativos Linux de 64 bits.

Este es exactamente el tipo de archivo que necesitarías para instalar PyTorch con soporte para ROCm.

Pasos recomendados:

1. **Verifica tu versión de Python:** Asegúrate de que el entorno Python donde quieras instalar esto es Python 3.12. Puedes verificarlo con `python --version`. Si tienes otra versión, necesitarás un wheel diferente o ajustar tu entorno.
2. **Verifica tu versión de ROCm:** Asegúrate de tener ROCm 6.4.1 (o una versión compatible con esta rueda de PyTorch) instalado en tu sistema.
3. **Desinstala la versión actual de PyTorch:** Para evitar conflictos, es mejor desinstalar la versión de PyTorch orientada a CUDA que tienes actualmente. En tu terminal, dentro de tu entorno virtual, ejecuta:

```
pip uninstall torch torchvision torchaudio
```

4. **Instala el wheel descargado:**

- Descarga el archivo `.whl` del enlace que encontraste.
- Abre tu terminal, navega hasta el directorio donde descargaste el archivo y ejecuta:

```
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/pytorch_triton_rocm-3.2.0%2Brocm6.4.1.git  
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/torchvision-0.21.0%2Brocm6.4.1.git4040d5  
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/torchaudio-2.6.0%2Brocm6.4.1.gitd8831425  
pip install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/torch-2.6.0%2Brocm6.4.1.git1ded221d-cp31
```

5. Instala torchvision y torchaudio compatibles (si es necesario): A veces, necesitarás instalar versiones de torchvision y torchaudio que sean compatibles con tu nueva versión de PyTorch y ROCm. Es posible que pip intente encontrar versiones compatibles automáticamente después de instalar el wheel de PyTorch, pero si no es así, puedes que necesites buscarlas en el mismo repositorio de AMD o seguir las guías de instalación de PyTorch para ROCm.

Podrías intentar:

```
pip install torchvision torchaudio --index-url https://download.pytorch.org/whl/rocmX.Y # (Reemplaza X.Y con tu versión)
```

Consulta la página oficial de PyTorch para los comandos exactos de instalación con ROCm, ya que pueden variar ligeramente.

6. **Verifica la instalación:** Despues de la instalación, vuelve a ejecutar el script de Python para verificar las versiones y la disponibilidad de la GPU:

```
import torch  
print(f"PyTorch version: {torch.__version__}")  
# Con una build de ROCm, torch.cuda.is_available() es el método que se sigue usando.  
# PyTorch abstrae la API de CUDA para que funcione también sobre ROCm (HIP).  
print(f"GPU available (ROCM should report True here via CUDA interface): {torch.cuda.is_available()}")  
if torch.cuda.is_available():  
    print(f"Device name: {torch.cuda.get_device_name(0)}") # Debería mostrar el nombre de tu GPU AMD  
    print(f"Number of GPUs: {torch.cuda.device_count()}")
```

Si todo va bien, `torch.cuda.is_available()` debería devolver `True`, y `torch.cuda.get_device_name(0)` debería mostrar el nombre de tu GPU AMD. Si es así, ¡tu script de YOLO debería poder usar la GPU!

Este es un paso muy importante y promPaso 5: Activar uso de GPUetedor. ¡Mucho suerte con la instalación!

Que Ultralytics instalar?

En general, la librería `ultralytics` (que es la que se usa para YOLOv5, YOLOv8, YOLOv10, etc.) busca ser compatible con versiones recientes de PyTorch.

Según la información disponible:

- La documentación de `ultralytics` y las notas de las versiones indican que suelen dar soporte a las versiones más actuales de PyTorch. Por ejemplo, versiones recientes de `ultralytics` (como la v8.3.49) mencionan explícitamente compatibilidad con PyTorch 2.5.
- Generalmente, `ultralytics` requiere Python ≥ 3.8 y PyTorch ≥ 1.8 o ≥ 1.7 para versiones más antiguas como YOLOv5.

Dado que PyTorch 2.6.0 es una versión posterior a las mencionadas explícitamente en algunos de los resultados de búsqueda, lo más probable es que las versiones más recientes de `ultralytics` sean compatibles con PyTorch 2.6.0 o estén en proceso de serlo muy pronto.

Recomendaciones:

1. **Usa la última versión de `ultralytics`:** Cuando instales PyTorch 2.6.0 (especialmente la versión para ROCm que encontraste), intenta instalar o actualizar a la última versión estable de `ultralytics` usando pip:

Bash

```
pip install -U ultralytics
```

2. Consulta la documentación oficial y GitHub:

- Revisa la [documentación oficial de Ultralytics](#) para las guías de instalación más recientes y las dependencias.
- Echa un vistazo al archivo `pyproject.toml` o `requirements.txt` en el [repositorio de GitHub de Ultralytics](#) para ver las especificaciones exactas de la versión de PyTorch.
- Revisa las "Releases" (lanzamientos) en GitHub para ver si en las notas de las versiones más recientes se menciona compatibilidad con PyTorch 2.6.0.

3. Prueba:

Después de la instalación, la mejor manera de confirmarlo es probar un script simple de YOLO para ver si funciona correctamente con tu instalación de PyTorch 2.6.0.

Es muy común que las bibliotecas como `ultralytics` se actualicen rápidamente para dar soporte a las nuevas versiones de sus dependencias principales como PyTorch. Si encuentras algún problema, la sección de "Issues" en el GitHub de `ultralytics` suele ser un buen lugar para buscar si otros usuarios han reportado incompatibilidades y cómo las han resuelto.

Paso 6: Parar entrenamiento y probar el modelo

¡Entendido! Es una excelente idea pausar el entrenamiento después de la época 6 para permitir que tu sistema se enfrie y, al mismo tiempo, probar el modelo con una imagen de prueba. Esto nos dará una idea de cómo está aprendiendo el modelo hasta ahora.

Instrucciones para detener el entrenamiento y preparar la prueba:

1. **Detén el entrenamiento manualmente:** Cuando veas que la época 6 ha terminado (o incluso durante la época 6 si la temperatura sube demasiado), presiona `Ctrl + C` en tu terminal donde está ejecutándose el script de entrenamiento. El proceso se detendrá, y lo más probable es que ya se haya guardado un `last.pt` para la época 6 (o la última época completa).
2. Verifica el checkpoint guardado:

Navega a la carpeta de resultados de tu entrenamiento. La ruta debería ser algo como:

`tu_proyecto_billar/runs/billar_balls_detection_v1/weights/`

Dentro de esta carpeta, deberías encontrar:

- `last.pt`: Los pesos del modelo al final de la última época completada.
- `best.pt`: Los pesos del modelo que ha tenido el mejor rendimiento en validación hasta ahora.

Para nuestra prueba, usaremos `last.pt`.

3. Elige una imagen de prueba:

Selecciona una imagen de tu dataset que no haya sido utilizada para el entrenamiento (idealmente del conjunto de val o test que preparaste) para probar el modelo. La ruta de la imagen debe ser clara y accesible para el nuevo script.

Ahora, vamos a preparar un script para realizar la inferencia (detección) en una imagen de prueba:

Crea un nuevo archivo Python (por ejemplo, `test_model.py`) en la raíz de tu carpeta `tu_proyecto_billar` y pega el siguiente código:

```
from ultralytics import YOLO
import os
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# --- 1. Definir rutas ---
base_project_dir = 'tu_proyecto_billar' # Asegúrate de que coincida con tu estructura
```

```

# Ruta al modelo entrenado (usaremos el último checkpoint)
# ASEGÚRATE de que 'billar_balls_detection_v1' sea el nombre correcto de tu carpeta de run
model_path = os.path.join(base_project_dir, 'runs', 'billar_balls_detection_v1', 'weights', 'last.pt')

# Ruta a la imagen de prueba (elige una de tu carpeta data/images/val o data/images/test)
# Ejemplo: si tienes 'imagen_val1.jpg' en data/images/val/
test_image_path = os.path.join(base_project_dir, 'data', 'images', 'val', '595.jpg.rf.1520df7aa3e5659b9f036291f58f11c

# Clases de tu dataset (debe coincidir con la lista usada en custom_data.yaml y en prepare_yolo_dataset.py)
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'
]

# --- 2. Cargar el modelo ---
try:
    model = YOLO(model_path)
    print(f"Modelo cargado desde: {model_path}")
except Exception as e:
    print(f"Error al cargar el modelo: {e}")
    print("Asegúrate de que la ruta al archivo 'last.pt' sea correcta y que el entrenamiento se haya guardado.")
    exit()

# --- 3. Realizar la inferencia ---
print(f"Realizando inferencia en: {test_image_path}")
# La función predict devuelve un objeto Results.
# verbose=False para no imprimir los detalles de inferencia en consola,
# conf=0.25 es el umbral de confianza para mostrar detecciones (ajústalo si detecta demasiado o muy poco)
results = model.predict(source=test_image_path, save=False, imgsz=1080, conf=0.25, verbose=False)

# --- 4. Procesar y visualizar los resultados ---
if results:
    # Solo tomamos el primer resultado (ya que estamos procesando una sola imagen)
    result = results[0]

    # Cargar la imagen original para visualización
    image = cv2.imread(test_image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # OpenCV carga BGR, matplotlib espera RGB

    fig, ax = plt.subplots(1, figsize=(12, 8))
    ax.imshow(image)
    ax.set_title("Detecciones del Modelo YOLO")

    # Extraer bounding boxes y clases
    for box in result.boxes:
        # box.xyxy: coordenadas en formato [x1, y1, x2, y2]
        x1, y1, x2, y2 = map(int, box.xyxy[0])

        # box.conf: confianza de la detección
        confidence = box.conf[0]

        # box.cls: ID de la clase detectada
        class_id = int(box.cls[0])
        class_name = classes[class_id] if class_id < len(classes) else f"Unknown_Class_{class_id}"

        # Dibujar el bounding box
        rect = patches.Rectangle((x1, y1), x2 - x1, y2 - y1,

```

```

        linewidth=2, edgecolor='red', facecolor='none')
ax.add_patch(rect)

# Poner la etiqueta de la clase y la confianza
# Usamos coordenadas relativas para el texto para que no se salga de la imagen
text_x = x1
text_y = y1 - 10 if y1 - 10 > 0 else y1 + 20 # Ajusta la posición del texto para que sea visible
ax.text(text_x, text_y, f"{class_name} {confidence:.2f}",
       color='red', fontsize=10, bbox=dict(facecolor='white', alpha=0.7, edgecolor='none', pad=1))

# Imprimir coordenadas y clase por consola
print(f"Detectado: {class_name} (Confianza: {confidence:.2f}) - Coordenadas: xmin={x1}, ymin={y1}, xmax={x2},"
      print(f" Centro: ({int((x1+x2)/2)}, {int((y1+y2)/2)})")

plt.axis('off') # Ocultar ejes
plt.show()

# Si quisieras las coordenadas normalizadas (0 a 1) para un uso posterior:
# Puedes usar result.boxes.xyxy para obtenerlas directamente.
# Por ejemplo:
# normalized_boxes = result.boxes.xyxy
# for i, bbox_norm in enumerate(normalized_boxes):
#     cls_id = int(result.boxes.cls[i])
#     conf = result.boxes.conf[i]
#     center_x_norm = (bbox_norm[0] + bbox_norm[2]) / 2
#     center_y_norm = (bbox_norm[1] + bbox_norm[3]) / 2
#     width_norm = bbox_norm[2] - bbox_norm[0]
#     height_norm = bbox_norm[3] - bbox_norm[1]
#     print(f"Normalizado: {classes[cls_id]} (Conf: {conf:.2f}) - Centro({center_x_norm:.4f}, {center_y_norm:.4f}), W({width_norm:.4f}, H({height_norm:.4f}))")

else:
    print("No se detectaron objetos en la imagen de prueba.")

```

Antes de ejecutar `test_model.py`:

- Guarda el script:** Guarda el código anterior como `test_model.py` en la raíz de tu carpeta `tu_proyecto_billar`.
- Ajusta `test_image_path`:** Es **CRUCIAL** que cambies la ruta en la línea `test_image_path = os.path.join(...)` para que apunte a una de tus imágenes de prueba reales. Te he puesto un ejemplo, pero deberías cambiar el nombre de archivo y, si es necesario, la subcarpeta (`val` o `test`).
- Ajusta `model_path`:** Asegúrate de que la ruta al `last.pt` en `model_path` sea correcta y que la carpeta `billar_balls_detection_v1` exista tras tu primer intento de entrenamiento.
- Asegúrate de que tu entorno esté activado:** `conda activate billar_dl` (o el nombre de tu entorno).
- Instala `opencv-python` y `matplotlib`** si no lo hiciste antes: `pip install opencv-python matplotlib`.

Ejecución:

```
python tu_proyecto_billar/test_model.py
```

Este script cargará tu modelo parcialmente entrenado, realizará una detección en la imagen que especifiques y mostrará la imagen con los `bounding boxes` y las etiquetas de las bolas detectadas. También imprimirá las coordenadas y clases de las bolas detectadas en la consola.

Esto te dará una excelente retroalimentación visual sobre el progreso del entrenamiento. Avísame cómo te va con la prueba. ¡Estoy impaciente por ver los resultados!

Para poder ver visualmente el resultado de las detecciones, te sugiero modificar el script `test_model.py` para **guardar la imagen con los bounding boxes dibujados**.

1. Modifica `test_model.py` :

En el script

`test_model.py`, justo antes de `plt.show()`, añade la siguiente línea:Python

```
# ... (código existente)

plt.axis('off') # Ocultar ejes

# --- Añade esta línea para guardar la imagen ---
output_image_path = os.path.join(base_project_dir, 'detected_image.jpg') # O el nombre que prefieras
plt.savefig(output_image_path, bbox_inches='tight', pad_inches=0)
print(f"\nImagen con detecciones guardada en: {output_image_path}")

plt.show() # Esta línea seguirá dando el warning si no hay GUI, pero la imagen ya estará guardada.
```

2. Vuelve a ejecutar `test_model.py` :Bash

```
python tu_proyecto_billar/test_model.py
```

Después de esto, deberías encontrar un archivo `detected_image.jpg` en la raíz de tu carpeta `tu_proyecto_billar` que podrás abrir con cualquier visor de imágenes para ver las detecciones.



¡Genial! Que el modelo ya sea capaz de detectar correctamente y que los *bounding boxes* sean precisos con solo 6 épocas es un muy buen comienzo. No tener falsos positivos a estas alturas es también una excelente señal.

El hecho de que falten bolas por detectar y que haya un error de clasificación (`dred_7` en lugar de la bola 4) es **completamente normal y esperado** en esta etapa temprana del entrenamiento. El modelo apenas está empezando a aprender los patrones más obvios.

Diagnóstico sobre la clasificación incorrecta y las bolas no detectadas:

- **Pocas épocas:** El modelo no ha visto suficientes ejemplos o no ha tenido las suficientes oportunidades para ajustar sus pesos de manera óptima y distinguir entre todas las clases de bolas y sus complejas apariencias (iluminación, ángulos, reflejos, etc.).
- **Similitud visual:** Algunas bolas pueden ser visualmente muy similares. Por ejemplo, la bola 4 es de color morado/rosa. Si la bola `dred_7` (rojo oscuro/granate) se parece a ella en ciertas condiciones de iluminación o reflejo, el modelo puede confundirlas inicialmente.
- **Clases minoritarias:** Si en tu dataset algunas clases de bolas (como la bola 4 o la bola 7) tienen menos ejemplos que otras, el modelo tendrá más dificultades para aprender a identificarlas correctamente.
- **Umbral de confianza:** Es posible que algunas bolas no se detecten porque su "confianza" predicha por el modelo es inferior al umbral de `0.25` que establecimos en el script de prueba. A medida que el entrenamiento avanza, estas confianzas deberían aumentar.

Próximos Pasos:

El camino a seguir es **continuar con el entrenamiento del modelo**. Más épocas significan que el modelo tendrá más oportunidades para:

1. **Ver más datos:** Procesar el dataset completo más veces.
2. **Ajustar pesos:** Refinar los miles (o millones) de parámetros internos de la red neuronal.
3. **Reducir errores:** Minimizar la función de pérdida y, en teoría, mejorar la precisión de las detecciones y clasificaciones.
4. **Aumentar la confianza:** Las bolas que ahora no se detectan o se clasifican mal deberían empezar a tener puntuaciones de confianza más altas y correctas.

Reanudando el Entrenamiento:

Ahora que sabes que puedes parar y reanudar, y que tu sistema mantiene temperaturas aceptables, puedes continuar el entrenamiento.

1. **Deja enfriar el sistema:** Si está caliente, dale un respiro a tu PC.
2. Modifica `train_yolo_model.py` para reanudar:

Abre tu script `train_yolo_model.py`. La línea donde cargas el modelo debe apuntar al `last.pt` que se guardó:

```
# ...
# model = YOLO('yolov8n.pt') # Comenta o elimina esta línea

# Carga el modelo desde el último checkpoint para reanudar
# Asegúrate de que 'billar_balls_detection_v1' sea el nombre correcto de tu carpeta de run
model = YOLO(os.path.join(base_project_dir, 'runs', 'billar_balls_detection_v1', 'weights', 'last.pt'))

# ... (el resto del script de entrenamiento)
```

Puedes mantener `epochs=50`. El entrenamiento reanudará desde la última época completada hasta alcanzar el total de 50.

3. **Vuelve a ejecutar el entrenamiento:**

```
python tu_proyecto_billar/train_yolo_model.py
```

Objetivo de esta próxima fase de entrenamiento:

Esperamos que, a medida que el entrenamiento avance (quizás hasta las épocas 20, 30 o incluso las 50), la precisión general del modelo mejore, y veamos menos bolas perdidas y más clasificaciones correctas.

Cuando el entrenamiento haya avanzado unas cuantas épocas más (o si lo dejas hasta el final), podemos volver a probarlo y ver la mejora. ¡Vamos por buen camino!

Paso 7: Reanudar el entrenamiento

Es una excelente pregunta, especialmente considerando que el entrenamiento puede ser largo cuando no se cuenta con una GPU NVIDIA con configuración CUDA optimizada para TensorFlow, y más con imágenes de alta resolución.

No, no tendrás que empezar completamente desde cero si detienes el entrenamiento.

Aquí te explico lo que sucede y las opciones que tienes con YOLOv8 (Ultralytics) si necesitas detener el entrenamiento:

YOLOv8 y Puntos de Verificación (Checkpoints):

Ultralytics YOLOv8 (y la mayoría de los frameworks de Deep Learning modernos) guardan automáticamente "puntos de verificación" o "checkpoints" del modelo a intervalos regulares durante el entrenamiento.

- `last.pt`: Este archivo guarda los pesos del modelo al final de cada época. Si detienes el entrenamiento abruptamente (Ctrl+C), el último `last.pt` guardado será el que corresponde a la última época que se completó.
- `best.pt`: Este archivo guarda los pesos del modelo que ha logrado el mejor rendimiento en el conjunto de validación hasta el momento. Este es el modelo que querrás usar para la inferencia después del entrenamiento.

Estos archivos se guardan en el directorio de tu ejecución de entrenamiento. Por ejemplo, en tu caso, sería algo como:

`tu_proyecto_billar/runs/billar_balls_detection_v1/weights/`.

¿Cómo reanudar el entrenamiento?

Si detienes el entrenamiento y quieres continuarlo desde el punto donde lo dejaste, puedes simplemente cargar el último `checkpoint` guardado.

1. **Encuentra el `checkpoint`:** Navega a `tu_proyecto_billar/runs/billar_balls_detection_v1/weights/` y busca el archivo `last.pt`.
2. **Modifica el script de entrenamiento:** En tu script `train_yolo_model.py`, cambia la línea donde cargas el modelo. En lugar de cargar el modelo pre-entrenado `yolov8n.pt`, cargarás tu último `checkpoint`.

Original:

Python

```
model = YOLO('yolov8n.pt')
```

Para reanudar el entrenamiento:

Python

```
model = YOLO('tu_proyecto_billar/runs/billar_balls_detection_v1/weights/last.pt')
```

O la ruta absoluta completa a ese archivo.

3. **Ejecuta el script nuevamente:** Cuando ejecutes el script, el entrenamiento continuará desde la época en que se detuvo, usando los pesos del modelo guardados en `last.pt`.

Consideraciones adicionales:

- **Paciencia con la CPU:** 35 minutos por época es un tiempo considerable, y si tu dataset es grande y el número de épocas es alto, el entrenamiento completo puede llevar muchos días. Por ejemplo, 50 épocas a 35 minutos/época son `50 * 35 = 1750 minutos`, lo que equivale a casi 29 horas de entrenamiento.
- **Google Colab / Cloud Computing:** Esta es la razón principal por la que se suele recomendar Google Colab (con su GPU gratuita) o servicios de computación en la nube para Deep Learning. Una tarea que tarda 35 minutos en CPU podría tardar unos pocos minutos o incluso segundos en una GPU dedicada. Si el tiempo es un factor crítico para ti, insisto en que consideres esta opción.

Así que, si necesitas parar el entrenamiento, no te preocupes, podrás continuarlo desde la última época guardada.

Si modificas el número de épocas a la baja en el script de entrenamiento y reanudas desde un checkpoint (`last.pt`), el comportamiento será el siguiente:

El entrenamiento se detendrá cuando se alcance el menor de los dos valores:

1. El número de épocas completadas previamente en el `last.pt` más el número de épocas adicionales que especifiques ahora.
2. El nuevo valor total de `epochs` que establezcas en el script.

Ejemplo detallado:

Supongamos que tu configuración inicial era:

- `epochs=50`

Y el entrenamiento se detuvo en la **época 20**. El archivo `last.pt` guarda el estado del modelo hasta el final de la época 20.

Ahora decides reanudar, pero modificas el script a:

- `model = YOLO('tu_proyecto_billar/runs/billar_balls_detection_v1/weights/last.pt')`
- `epochs=30` (¡Has bajado el número total de épocas!)

¿Qué pasará?

El entrenamiento cargará el modelo desde la época 20. El objetivo final que le has dicho al script es que entrene hasta la época 30. Por lo tanto, el entrenamiento **solo ejecutará 10 épocas más** (de la 21 a la 30) y luego se detendrá.

En resumen:

- Si `(épocas_actuales_en_last_pt + épocas_adicionales_deseadas)` es **mayor** que el nuevo `epochs` que fijes, el entrenamiento se detendrá en el nuevo `epochs`.
- Si `(épocas_actuales_en_last_pt + épocas_adicionales_deseadas)` es **menor** que el nuevo `epochs` que fijes, el entrenamiento continuará hasta que se complete el número total de `epochs` que le has especificado.

Esta flexibilidad te permite probar modelos más rápidamente o ajustar el entrenamiento si ves que el modelo ya está convergiendo bien antes de alcanzar el número original de épocas.

Epoch	GPU	mem	box loss	cls loss	dfl loss	Instances	Size
1/10	0G		0.9206	0.9779	1.062	17	1088: 100% 461/461 [36:35<00:00, 4.76s/it]
1.80s/i	Class	Images	Instances		Box(P)	R	mAP50 mAP50-95: 100% 10/10 [00:17<00:00,
	all		235	1932	0.723	0.729	0.76 0.458
2/10	0G		0.9246	0.945	1.06	13	1088: 100% 461/461 [35:57<00:00, 4.68s/it]
1.77s/i	Class	Images	Instances		Box(P)	R	mAP50 mAP50-95: 100% 10/10 [00:17<00:00,
	all		235	1932	0.724	0.683	0.742 0.446
3/10	0G		0.9198	0.9079	1.057	17	1088: 100% 461/461 [35:55<00:00, 4.67s/it]
1.77s/i	Class	Images	Instances		Box(P)	R	mAP50 mAP50-95: 100% 10/10 [00:17<00:00,
	all		235	1932	0.713	0.701	0.728 0.441
4/10	0G		0.9022	0.8564	1.048	6	1088: 100% 461/461 [36:25<00:00, 4.74s/it]
1.77s/i	Class	Images	Instances		Box(P)	R	mAP50 mAP50-95: 100% 10/10 [00:17<00:00,
	all		235	1932	0.721	0.729	0.761 0.464

Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
	0G	0.8221	0.7072	1.006	17	1088: 100% [██████] 461/461 [36:39<00:00, 4.77s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.747	R 0.696	mAP50 mAP50-95: 100% [██████] 10/10 [00:18<00:00, 0.468]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
2/10	0G	0.8145	0.6926	0.9995	13	1088: 100% [██████] 461/461 [37:21<00:00, 4.86s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.773	R 0.69	mAP50 mAP50-95: 100% [██████] 10/10 [00:17<00:00, 0.463]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
3/10	0G	0.8156	0.6934	0.9997	17	1088: 100% [██████] 461/461 [37:05<00:00, 4.83s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.744	R 0.678	mAP50 mAP50-95: 100% [██████] 10/10 [00:17<00:00, 0.438]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
4/10	0G	0.8201	0.6832	1.003	6	1088: 100% [██████] 461/461 [36:46<00:00, 4.79s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.708	R 0.712	mAP50 mAP50-95: 100% [██████] 10/10 [00:18<00:00, 0.441]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
5/10	0G	0.8335	0.6943	1.01	50	1088: 100% [██████] 461/461 [35:57<00:00, 4.68s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.728	R 0.708	mAP50 mAP50-95: 100% [██████] 10/10 [00:18<00:00, 0.45]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
6/10	0G	0.8146	0.6468	1	28	1088: 100% [██████] 461/461 [35:33<00:00, 4.63s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.731	R 0.723	mAP50 mAP50-95: 100% [██████] 10/10 [00:17<00:00, 0.459]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
7/10	0G	0.7813	0.5947	0.9802	26	1088: 100% [██████] 461/461 [35:31<00:00, 4.62s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.763	R 0.709	mAP50 mAP50-95: 100% [██████] 10/10 [00:18<00:00, 0.471]
Epoch	GPU_mem	box_loss	cls_loss	dfLoss	Instances	Size
8/10	0G	0.754	0.5566	0.9692	20	1088: 100% [██████] 461/461 [35:31<00:00, 4.62s/it]
	Class all	Images 235	Instances 1932	Box(P) 0.746	R 0.722	mAP50 mAP50-95: 100% [██████] 10/10 [00:18<00:00, 0.471]

Tabla de Datos de evolución de todos los entrenamientos

Entrenamiento 1 (6 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfLoss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,2236.18,1.3323,2.90233,1.31611,0.45284,0.51557,0.4791,0.27876,1.44528,2.12665,1.37201,0.00017508,0.00017508,0.00017508
2,4453.33,1.1992,1.91814,1.23216,0.59162,0.64284,0.64608,0.38067,1.41765,1.68118,1.35138,0.000343475,0.000343477
3,6574.82,1.13298,1.62487,1.18559,0.60707,0.62125,0.64286,0.3719,1.46805,1.70324,1.37759,0.000504927,0.00050492
4,8676.66,1.08208,1.40919,1.15271,0.6662,0.6617,0.68579,0.39699,1.4635,1.49477,1.35356,0.000494756,0.000494756
5,10779.3,1.05799,1.2702,1.13709,0.72806,0.68282,0.7399,0.41434,1.50268,1.30063,1.39651,0.000484341,0.00048434
6,12822.3,1.00829,1.14359,1.11195,0.71628,0.69706,0.73366,0.44203,1.38563,1.27912,1.29689,0.000473926,0.0004739
```

Entrenamiento 2 (4 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfLoss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,2213.68,0.92055,0.97787,1.06164,0.72314,0.7286,0.76002,0.45751,1.38452,1.15503,1.30892,0.000174953,0.000174955
2,4388.81,0.92458,0.94503,1.05993,0.72425,0.68258,0.74201,0.44604,1.40536,1.20169,1.31932,0.000315608,0.000315608
3,6561.79,0.91984,0.90793,1.05748,0.71265,0.70095,0.72764,0.4407,1.38783,1.1951,1.31489,0.000421547,0.000421547
4,8765.36,0.90217,0.85645,1.04845,0.72057,0.7293,0.76075,0.46393,1.36258,1.12434,1.27897,0.000369778,0.000369778
```

Entrenamiento 3 (8 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfLoss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,2217.95,0.82214,0.70723,1.006,0.74743,0.69643,0.75527,0.46792,1.35516,1.08085,1.28143,0.000174953,0.000174953
2,4477.18,0.81447,0.69257,0.99955,0.77344,0.69005,0.76179,0.46263,1.39834,1.09327,1.32508,0.000315608,0.000315608
3,6721.07,0.81562,0.69342,0.99972,0.74423,0.6775,0.73314,0.43758,1.424,1.15778,1.34042,0.000421547,0.000421547
4,8945.64,0.82006,0.6832,1.00275,0.70831,0.71212,0.73648,0.44121,1.39524,1.15234,1.29225,0.000369778,0.000369778
5,11121.6,0.83354,0.69428,1.00967,0.72786,0.70842,0.74037,0.44953,1.38659,1.14849,1.31105,0.000317704,0.000317704
6,13273.7,0.81457,0.64684,0.99997,0.73094,0.72251,0.75023,0.45871,1.38027,1.10128,1.30204,0.00026563,0.00026563
7,15423.6,0.7813,0.59469,0.98021,0.76295,0.70881,0.76838,0.47075,1.38826,1.08097,1.30221,0.000213556,0.000213556
8,17573.4,0.75404,0.55661,0.96922,0.74593,0.72188,0.7659,0.47145,1.37841,1.05347,1.3093,0.000161482,0.000161482
```

Entrenamiento 4 (5 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfLoss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,2351.98,0.82873,0.69972,1.00841,0.76179,0.69948,0.76659,0.47048,1.35321,1.04219,1.28535,0.000174889,0.000174889
2,4725.19,0.82139,0.67897,1.00618,0.76023,0.69799,0.76085,0.46694,1.3479,1.06848,1.2933,0.000280879,0.000280879
3,7031.55,0.81358,0.66284,1.00113,0.73762,0.70815,0.75148,0.45832,1.37407,1.11245,1.30978,0.000317436,0.000317436
4,9315.36,0.78873,0.62684,0.99017,0.75198,0.70771,0.75897,0.45977,1.38722,1.04716,1.30381,0.000213556,0.000213556
5,11608.2,0.76218,0.58485,0.97586,0.7483,0.7375,0.7667,0.47758,1.32186,1.01599,1.27561,0.000109408,0.000109408,
```

Paso 8: Resultados del entrenamiento

Primer entrenamiento (6 epochs completadas)

Iniciando entrenamiento del modelo...

Ultralytics 8.3.146 🚀 Python-3.12.3 torch-2.7.0+cu126 CPU (AMD Ryzen 7 5800X3D 8-Core Processor)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=8, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=./custom_data.yaml, degrees=0.0, deterministic=True, device=cpu, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=50, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train,
model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=billar_balls_detection_v1, nbs=64, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=./runs, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=runs/billar_balls_detection_v1, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=15

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]

```

21      -1 1 493056 ultralytics.nn.modules.block.C2f      [384, 256, 1]
22      [15, 18, 21] 1 754237 ultralytics.nn.modules.head.Detect      [15, [64, 128, 256]]
Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs
Transferred 319/355 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
WARNING ! imgs=[1080] must be multiple of max stride 32, updating to [1088]
train: Fast image access ✓ (ping: 0.0±0.0 ms, read: 133.3±55.3 MB/s, size: 159.4 KB)
train: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/train... 5523 images,
0 backgrounds, 0 corrupt: 100%|██████████|
train: New cache created:
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/train.cache
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/
init.py:789: UserWarning: Can't initialize NVML
warnings.warn("Can't initialize NVML")
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-
packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is
found, then device pinned memory won't be used.
warnings.warn(warn_msg)
val: Fast image access ✓ (ping: 0.0±0.0 ms, read: 121.2±35.6 MB/s, size: 108.3 KB)
val: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/val... 235 images, 0
backgrounds, 0 corrupt: 100%|██████████|
val: New cache created: /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/val.cache
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/
init.py:789: UserWarning: Can't initialize NVML
warnings.warn("Can't initialize NVML")
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-
packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is
found, then device pinned memory won't be used.
warnings.warn(warn_msg)
Plotting labels to runs/billiar_balls_detection_v1/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and
'momentum' automatically...
optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64
weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 1088 train, 1088 val
Using 0 dataloader workers
Logging results to runs/billiar_balls_detection_v1
Starting training for 50 epochs...

```

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/50	0G	1.332	2.902	1.316	33	1088: 100% ██████████ 691/691 [36:58<00:00, 3.21s/it]
Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% ██████████ 15/15 [00:17<00:00, 1.18s/it]
all	235	1932	0.453	0.516	0.479	0.279

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/50	0G	1.199	1.918	1.232	50	1088: 100% ██████████ 691/691 [36:39<00:00, 3.18s/it]
Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% ██████████ 15/15 [00:17<00: 00, 1.14s/it]
all	235	1932	0.592	0.643	0.646	0.381

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/50	0G	1.133	1.625	1.186	27	1088: 100% ██████████ 691/691 [35:04<00:00, 3.05s/it]
Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% ██████████ 15/15 [00:16<00: 00, 1.12s/it]
all	235	1932	0.607	0.621	0.643	0.372

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

```

4/50      0G    1.082   1.409   1.153    51    1088: 100%|██████████| 691/691 [34:44<00:00, 3.02s/it]
          Class Images Instances Box(P)     R     mAP50 mAP50-95): 100%|██████████| 15/15 [00:16<00:
00, 1.12s/it]
          all    235    1932    0.666    0.662    0.686    0.397

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
5/50      0G    1.058   1.27    1.137    20    1088: 100%|██████████| 691/691 [34:45<00:00, 3.02s/it]
          Class Images Instances Box(P)     R     mAP50 mAP50-95): 100%|██████████| 15/15 [00:16<00:
00, 1.11s/it]
          all    235    1932    0.728    0.683    0.74     0.414

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
6/50      0G    1.008   1.144   1.112    45    1088: 100%|██████████| 691/691 [33:46<00:00, 2.93s/it]
          Class Images Instances Box(P)     R     mAP50 mAP50-95): 100%|██████████| 15/15 [00:16<00:
00, 1.08s/it]
          all    235    1932    0.716    0.697    0.734    0.442

```

Segundo entrenamiento (4 epochs completadas, reanudando el primero)

Iniciando entrenamiento del modelo...

Ultralytics 8.3.146 🚀 Python-3.12.3 torch-2.7.0+cu126 CPU (AMD Ryzen 7 5800X3D 8-Core Processor)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=12, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=./custom_data.yaml, degrees=0.0, deterministic=True, device=cpu, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=./runs/billar_balls_detection_v1/weights/last.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=billar_balls_detection_v12, nbs=64, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=./runs, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=runs/billar_balls_detection_v12, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10		-1	1	0	torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]

```

12      -1 1 148224 ultralytics.nn.modules.block.C2f      [384, 128, 1]
13      -1 1      0 torch.nn.modules.upsampling.Upsample   [None, 2, 'nearest']
14      [-1, 4] 1      0 ultralytics.nn.modules.conv.Concat [1]
15      -1 1 37248 ultralytics.nn.modules.block.C2f      [192, 64, 1]
16      -1 1 36992 ultralytics.nn.modules.conv.Conv     [64, 64, 3, 2]
17      [-1, 12] 1      0 ultralytics.nn.modules.conv.Concat [1]
18      -1 1 123648 ultralytics.nn.modules.block.C2f      [192, 128, 1]
19      -1 1 147712 ultralytics.nn.modules.conv.Conv     [128, 128, 3, 2]
20      [-1, 9] 1      0 ultralytics.nn.modules.conv.Concat [1]
21      -1 1 493056 ultralytics.nn.modules.block.C2f      [384, 256, 1]
22      [15, 18, 21] 1 754237 ultralytics.nn.modules.head.Detect [15, [64, 128, 256]]]

Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs

Transferred 355/355 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
WARNING ! imgs=[1080] must be multiple of max stride 32, updating to [1088]
train: Fast image access ✓ (ping: 0.0±0.0 ms, read: 3525.9±329.7 MB/s, size: 159.4 KB)
train: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/train.cache... 5523
images, 0 background
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/
init.py:789: UserWarning: Can't initialize NVML
warnings.warn("Can't initialize NVML")
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-
packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is
found, then device pinned memory won't be used.
warnings.warn(warn_msg)
val: Fast image access ✓ (ping: 0.0±0.0 ms, read: 3654.9±433.6 MB/s, size: 108.3 KB)
val: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/val.cache... 235
images, 0 backgrounds, 0
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/
init.py:789: UserWarning: Can't initialize NVML
warnings.warn("Can't initialize NVML")
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-
packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is
found, then device pinned memory won't be used.
warnings.warn(warn_msg)
Plotting labels to runs/billiar_balls_detection_v12/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and
'momentum' automatically...
optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64
weight(decay=0.00046875), 63 bias(decay=0.0)
Image sizes 1088 train, 1088 val
Using 0 dataloader workers
Logging results to runs/billiar_balls_detection_v12
Starting training for 10 epochs...
Closing dataloader mosaic
/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-
packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is
found, then device pinned memory won't be used.
warnings.warn(warn_msg)

```

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/10	0G	0.9206	0.9779	1.062	17	1088: 100% [██████████ 461/461 [36:35<00:00, 4.76s/it]

```

Class   Images Instances   Box(P      R   mAP50 mAP50-95): 100%|██████████| 10/10 [00:17<00:00, 1.80s/i
all     235    1932    0.723    0.729    0.76    0.458

Epoch   GPU_mem box_loss cls_loss dfl_loss Instances   Size
2/10    0G    0.9246   0.945    1.06     13    1088: 100%|██████████| 461/461 [35:57<00:00, 4.68s/it]
          Class   Images Instances   Box(P      R   mAP50 mAP50-95): 100%|██████████| 10/10 [00:17<00:
00, 1.77s/i
          all     235    1932    0.724    0.683    0.742    0.446

Epoch   GPU_mem box_loss cls_loss dfl_loss Instances   Size
3/10    0G    0.9198   0.9079   1.057     17    1088: 100%|██████████| 461/461 [35:55<00:00, 4.67s/it]
          Class   Images Instances   Box(P      R   mAP50 mAP50-95): 100%|██████████| 10/10 [00:17<00:
00, 1.77s/i
          all     235    1932    0.713    0.701    0.728    0.441

Epoch   GPU_mem box_loss cls_loss dfl_loss Instances   Size
4/10    0G    0.9022   0.8564   1.048     6     1088: 100%|██████████| 461/461 [36:25<00:00, 4.74s/i
t]
          Class   Images Instances   Box(P      R   mAP50 mAP50-95): 100%|██████████| 10/10 [00:17<00:
00, 1.77s/i
          all     235    1932    0.721    0.729    0.761    0.464

```

Tercer entrenamiento (8 epochs completadas, reanudando el segundo)

Iniciando entrenamiento del modelo...

New

<https://pypi.org/project/ultralytics/8.3.147> available 😊 Update with 'pip install -U ultralytics'
 Ultralytics 8.3.146 🎉 Python-3.12.3 torch-2.7.0+cu126 CPU (AMD Ryzen 7 5800X3D 8-Core Processor)
 engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=12, bgr=0.0,
 box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0,
 copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=./custom_data.yaml, degrees=0.0, deterministic=True,
 device=cpu, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False,
 filplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4,
 imgsz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4,
 max_det=300, mixup=0.0, mode=train, model=./runs/billar_balls_detection_v12/weights/last.pt, momentum=0.937,
 mosaic=1.0, multi_scale=False, name=billar_balls_detection_v13, nbs=64, nms=False, opset=None, optimize=False,
 optimizer=auto, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True,
 profile=False, project=./runs, rect=False, resume=False, retina_masks=False, save=True, save_conf=False,
 save_crop=False, save_dir=runs/billar_balls_detection_v13, save_frames=False, save_json=False, save_period=-1,
 save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True,
 simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None,
 tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1,
 warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]

```

6      -1 2  197632 ultralytics.nn.modules.block.C2f      [128, 128, 2, True]
7      -1 1  295424 ultralytics.nn.modules.conv.Conv     [128, 256, 3, 2]
8      -1 1  460288 ultralytics.nn.modules.block.C2f      [256, 256, 1, True]
9      -1 1  164608 ultralytics.nn.modules.block.SPPF     [256, 256, 5]
10     -1 1    0 torch.nn.modules.upsampling.Upsample     [None, 2, 'nearest']
11     [-1, 6] 1    0 ultralytics.nn.modules.conv.Concat    [1]
12     -1 1  148224 ultralytics.nn.modules.block.C2f      [384, 128, 1]
13     -1 1    0 torch.nn.modules.upsampling.Upsample     [None, 2, 'nearest']
14     [-1, 4] 1    0 ultralytics.nn.modules.conv.Concat    [1]
15     -1 1  37248 ultralytics.nn.modules.block.C2f      [192, 64, 1]
16     -1 1  36992 ultralytics.nn.modules.conv.Conv       [64, 64, 3, 2]
17     [-1, 12] 1   0 ultralytics.nn.modules.conv.Concat    [1]
18     -1 1  123648 ultralytics.nn.modules.block.C2f      [192, 128, 1]
19     -1 1  147712 ultralytics.nn.modules.conv.Conv       [128, 128, 3, 2]
20     [-1, 9] 1   0 ultralytics.nn.modules.conv.Concat    [1]
21     -1 1  493056 ultralytics.nn.modules.block.C2f      [384, 256, 1]
22     [15, 18, 21] 1  754237 ultralytics.nn.modules.head.Detect [15, [64, 128, 256]]

```

Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs

Transferred 355/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'

WARNING ! imgs=[1080] must be multiple of max stride 32, updating to [1088]

train: Fast image access ✓ (ping: 0.0±0.0 ms, read: 3585.1±390.5 MB/s, size: 159.4 KB)

train: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/train.cache... 5523 images, 0 bac

/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/init.py:789: UserWarning: Can't initialize NVML

warnings.warn("Can't initialize NVML")

/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.

warnings.warn(warn_msg)

val: Fast image access ✓ (ping: 0.0±0.0 ms, read: 3495.8±341.6 MB/s, size: 108.3 KB)

val: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/val.cache... 235 images, 0 backgrou

/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/init.py:789: UserWarning: Can't initialize NVML

warnings.warn("Can't initialize NVML")

/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.

warnings.warn(warn_msg)

Plotting labels to runs/billar_balls_detection_v13/labels.jpg...

optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...

optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.00046875), 63 bias(decay=0.0)

Image sizes 1088 train, 1088 val

Using 0 dataloader workers

Logging results to runs/billar_balls_detection_v13

Starting training for 10 epochs...

Closing dataloader mosaic

/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-

```
packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is
found, then device pinned memory won't be used.
warnings.warn(warn_msg)
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
1/10 0G 0.8221 0.7072 1.006 17 1088: 100% [██████████] 461/461 [36:39<00:00, 4.77s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:18<00:00,
all 235 1932 0.747 0.696 0.755 0.468
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
2/10 0G 0.8145 0.6926 0.9995 13 1088: 100% [██████████] 461/461 [37:21<00:00, 4.86s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:17<00:
00,
all 235 1932 0.773 0.69 0.762 0.463
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
3/10 0G 0.8156 0.6934 0.9997 17 1088: 100% [██████████] 461/461 [37:05<00:00, 4.83s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:17<00:
00,
all 235 1932 0.744 0.678 0.733 0.438
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
4/10 0G 0.8201 0.6832 1.003 6 1088: 100% [██████████] 461/461 [36:46<00:00, 4.79s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:18<00:
00,
all 235 1932 0.708 0.712 0.736 0.441
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
5/10 0G 0.8335 0.6943 1.01 50 1088: 100% [██████████] 461/461 [35:57<00:00, 4.68s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:18<00:
00,
all 235 1932 0.728 0.708 0.74 0.45
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
6/10 0G 0.8146 0.6468 1 28 1088: 100% [██████████] 461/461 [35:33<00:00, 4.63s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:17<00:
00,
all 235 1932 0.731 0.723 0.75 0.459
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
7/10 0G 0.7813 0.5947 0.9802 26 1088: 100% [██████████] 461/461 [35:31<00:00, 4.62s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:18<00:
00,
all 235 1932 0.763 0.709 0.768 0.471
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
8/10 0G 0.754 0.5566 0.9692 20 1088: 100% [██████████] 461/461 [35:31<00:00, 4.62s/it]
t]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 10/10 [00:18<00:
00,
all 235 1932 0.746 0.722 0.766 0.471
```

Cuarto entrenamiento (5 epochs completadas, reanudando tercero)

Iniciando entrenamiento del modelo...

New

<https://pypi.org/project/ultralytics/8.3.147> available 😊 Update with 'pip install -U ultralytics'
Ultralytics 8.3.146 🎉 Python-3.12.3 torch-2.7.0+cu126 CPU (AMD Ryzen 7 5800X3D 8-Core Processor)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=14, bgr=0.0,
box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0,
copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=./custom_data.yaml, degrees=0.0, deterministic=True,
device=cpu, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=5, erasing=0.4, exist_ok=False,
fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4,
imgsz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4,
max_det=300, mixup=0.0, mode=train, model=./runs/billar_balls_detection_v13/weights/last.pt, momentum=0.937,
mosaic=1.0, multi_scale=False, name=billar_balls_detection_v14, nbs=64, nms=False, opset=None, optimize=False,
optimizer=auto, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True,
profile=False, project=./runs, rect=False, resume=False, retina_masks=False, save=True, save_conf=False,
save_crop=False, save_dir=runs/billar_balls_detection_v14, save_frames=False, save_json=False, save_period=-1,
save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True,
simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None,
tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1,
warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	754237	ultralytics.nn.modules.head.Detect	[15, [64, 128, 256]]

Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs

Transferred 355/355 items from pretrained weights
 Freezing layer 'model.22.dfl.conv.weight'
 WARNING ! imgs=[1080] must be multiple of max stride 32, updating to [1088]
 train: Fast image access ✓ (ping: 0.0±0.0 ms, read: 3484.2±421.8 MB/s, size: 159.4 KB)
 train: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/train.cache... 5523 images, 0 bac
 /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/init.py:789: UserWarning: Can't initialize NVML
 warnings.warn("Can't initialize NVML")
 /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
 warnings.warn(warn_msg)
 val: Fast image access ✓ (ping: 0.0±0.0 ms, read: 2807.3±326.9 MB/s, size: 108.3 KB)
 val: Scanning /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/data/labels/val.cache... 235 images, 0 backgrou
 /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/cuda/init.py:789: UserWarning: Can't initialize NVML
 warnings.warn("Can't initialize NVML")
 /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/lib/python3.12/site-packages/torch/utils/data/dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
 warnings.warn(warn_msg)
 Plotting labels to runs/billar_balls_detection_v14/labels.jpg...
 optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
 optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.000546875), 63 bias(decay=0.0)
 Image sizes 1088 train, 1088 val
 Using 0 dataloader workers
 Logging results to runs/billar_balls_detection_v14
 Starting training for 5 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/5	0G	0.8287	0.6997	1.008	85	1088: 100% ██████████ 395/395 [38:53<00:00, 5.91s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 9/9 [00:18<00:00, 2.
	all	235	1932	0.762	0.699	0.767 0.47
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/5	0G	0.8214	0.679	1.006	159	1088: 100% ██████████ 395/395 [39:14<00:00, 5.96s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 9/9 [00:18<00:00, 2.
	all	235	1932	0.76	0.698	0.761 0.467
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/5	0G	0.8136	0.6628	1.001	79	1088: 100% ██████████ 395/395 [38:07<00:00, 5.79s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 9/9 [00:18<00:00, 2.
	all	235	1932	0.738	0.708	0.751 0.458
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
4/5	0G	0.7887	0.6268	0.9902	103	1088: 100% ██████████ 395/395 [37:45<00:00, 5.73s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 9/9 [00:18<00:00,

```

0, 2.
    all 235 1932 0.752 0.708 0.759 0.46

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
 5/5   0G  0.7622  0.5849  0.9759     60  1088: 100%|██████████| 395/395 [37:54<00:00, 5.76
s/it]
      Class Images Instances Box(P) R mAP50 mAP50-95): 100%|██████████| 9/9 [00:18<00:0
0, 2.
    all 235 1932 0.748 0.737 0.767 0.478

```

5 epochs completed in 3.225 hours.

Optimizer stripped from runs/billar_balls_detection_v14/weights/last.pt, 6.3MB

Optimizer stripped from runs/billar_balls_detection_v14/weights/best.pt, 6.3MB

Validating runs/billar_balls_detection_v14/weights/best.pt...

Ultralytics 8.3.146 🚀 Python-3.12.3 torch-2.7.0+cu126 CPU (AMD Ryzen 7 5800X3D 8-Core Processor)

Model summary (fused): 72 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	Size
all	235	1932	0.748	0.738	0.767	0.478	9/9 [00:15<00:00, 1.
white	85	158	0.743	0.778	0.773	0.487	
blue_10	66	121	0.797	0.645	0.714	0.427	
dred_15	67	124	0.653	0.694	0.706	0.436	
black_8	89	164	0.873	0.674	0.796	0.457	
purple_12	55	101	0.599	0.713	0.607	0.359	
dred_7	64	122	0.71	0.704	0.733	0.456	
orange_13	60	110	0.796	0.736	0.788	0.466	
blue_2	73	136	0.775	0.794	0.773	0.499	
red_3	70	126	0.69	0.794	0.793	0.469	
green_6	69	129	0.838	0.868	0.873	0.585	
green_14	65	115	0.813	0.809	0.815	0.448	
red_11	72	140	0.707	0.636	0.734	0.503	
yellow_1	80	147	0.728	0.844	0.852	0.566	
orange_5	57	106	0.676	0.623	0.71	0.457	
yellow_9	75	133	0.817	0.752	0.831	0.547	

Speed: 1.9ms preprocess, 53.1ms inference, 0.0ms loss, 0.3ms postprocess per image

Results saved to runs/billar_balls_detection_v14

¡Entrenamiento completado!

Los resultados se guardaron en: ./runs/billar_balls_detection_v1

Puedes revisar los gráficos de entrenamiento y las métricas allí.

Entrenamiento colab x10 epoch

Iniciando entrenamiento del modelo...

```

Ultralytics 8.3.150 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer:agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=8, bgr=0.0, b
ox=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0, copy_paste
e_mode=flip, cos_lr=False, cutmix=0.0, data=custom_data_colab.yaml, degrees=0.0, deterministic=True, device=N
one, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, flplr
=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, img
sz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=3
00, mixup=0.0, mode=train, model=poolballs36.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=billar_
balls_detection_v1, nbs=64, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=True, patienc
e=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=runs, rect=False, resume=
False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=runs/billar_balls_detection_v1,
save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False,
show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, st

```

```
ream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None
```

```
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
```

```
100%|██████████| 755k/755k [00:00<00:00, 164MB/s]
```

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	754237	ultralytics.nn.modules.head.Detect	[15, [64, 128, 256]]

```
Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs
```

```
Transferred 355/355 items from pretrained weights
```

```
Freezing layer 'model.22.dfl.conv.weight'
```

```
AMP:running Automatic Mixed Precision (AMP) checks...
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to 'yolo11n.pt'...
```

```
100%|██████████| 5.35M/5.35M [00:00<00:00, 314MB/s]
```

```
AMP:checks passed ✅
```

```
WARNING ⚠️ imgs=[1080] must be multiple of max stride 32, updating to [1088]
```

```
train:Fast image access ✅ (ping: 0.0±0.0 ms, read: 2262.4±999.1 MB/s, size: 159.4 KB)
```

```
train:Scanning /content/data/labels/train... 5523 images, 0 backgrounds, 0 corrupt: 100%|██████████| 5523/5523 [00:02<00:00, 2420.59it/s]
```

```
train:New cache created: /content/data/labels/train.cache
```

```
albumentations:Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weigh
```

```
ted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))  
val:Fast image access ✓ (ping: 0.0±0.0 ms, read: 514.3±109.0 MB/s, size: 108.3 KB)
```

```
val:Scanning /content/data/labels/val... 235 images, 0 backgrounds, 0 corrupt: 100%|██████████| 235/235 [00:  
00<00:00, 1480.85it/s]
```

```
val>New cache created: /content/data/labels/val.cache
```

```
Plotting labels to runs/billar_balls_detection_v1/labels.jpg...  
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0'  
and 'momentum' automatically...  
optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=  
0.0005), 63 bias(decay=0.0)  
Image sizes 1088 train, 1088 val  
Using 2 dataloader workers  
Logging results to runs/billar_balls_detection_v1  
Starting training for 10 epochs...  
Closing dataloader mosaic  
albumentations:Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weigh  
ted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))
```

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/10	3.02G	0.6406	0.448	0.9183	45	1088: 100% ██████████ 691/691 [04:21<00:00, 2.6 it/s] Class Images Instances Box(P) R mAP50 mAP50-95): 100% ██████████ 15/15 [00:05< 00:00, 2.81it/s]

all	235	1932	0.748	0.683	0.736	0.448
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/10	3.17G	0.694	0.5036	0.9356	12	1088: 100% ██████████ 691/691 [04:12<00:00, 2.74 it/s] Class Images Instances Box(P) R mAP50 mAP50-95): 100% ██████████ 15/15 [00:05< 00:00, 2.86it/s]

all	235	1932	0.703	0.698	0.72	0.431
-----	-----	------	-------	-------	------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/10	3.17G	0.7237	0.5578	0.9527	45	1088: 100% ██████████ 691/691 [04:13<00:00, 2.7 3it/s] Class Images Instances Box(P) R mAP50 mAP50-95): 100% ██████████ 15/15 [00:04< 00:00, 3.10it/s]

all	235	1932	0.728	0.684	0.724	0.447
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
4/10	3.17G	0.7113	0.5253	0.9466	17	1088: 100% ██████████ 691/691 [04:07<00:00, 2.79 it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 15/15 [00:03<00:00, 4.07it/s]

all	235	1932	0.742	0.691	0.741	0.449
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
5/10	3.17G	0.7037	0.5141	0.9409	16	1088: 100% ██████████ 691/691 [04:05<00:00, 2.81 it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 15/15 [00:03<00:00, 3.94it/s]

all	235	1932	0.724	0.702	0.735	0.445
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
6/10	3.19G	0.6843	0.4806	0.9356	5	1088: 100% ██████████ 691/691 [04:10<00:00, 2.76it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 15/15 [00:03<00:00, 3.86it/s]

all	235	1932	0.706	0.716	0.728	0.438
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
7/10	3.19G	0.6651	0.4643	0.9264	4	1088: 100% ██████████ 691/691 [04:07<00:00, 2.79it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 15/15 [00:03<00:00, 3.98it/s]

all	235	1932	0.708	0.717	0.735	0.449
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
8/10	3.21G	0.6441	0.4293	0.9175	7	1088: 100% ██████████ 691/691 [04:09<00:00, 2.77 it/s]

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	Size
00:00, 3.19it/s]							15/15 [00:04<

all	235	1932	0.756	0.688	0.742	0.456	
-----	-----	------	-------	-------	-------	-------	--

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

9/10	3.21G	0.6218	0.4099	0.9107	7	1088: 100%	691/691 [04:06<00:00, 2.81it/s]
------	-------	--------	--------	--------	---	------------	---------------------------------

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	Size
00:00, 3.90it/s]							15/15 [00:03<

all	235	1932	0.721	0.721	0.738	0.453	
-----	-----	------	-------	-------	-------	-------	--

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

10/10	3.21G	0.603	0.3924	0.9018	8	1088: 100%	691/691 [04:05<00:00, 2.81it/s]
-------	-------	-------	--------	--------	---	------------	---------------------------------

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	Size
00:00, 3.08it/s]							15/15 [00:04<

all	235	1932	0.751	0.702	0.741	0.458	
-----	-----	------	-------	-------	-------	-------	--

10 epochs completed in 0.708 hours.

Optimizer stripped from runs/billar_balls_detection_v1/weights/last.pt, 6.3MB

Optimizer stripped from runs/billar_balls_detection_v1/weights/best.pt, 6.3MB

Validating runs/billar_balls_detection_v1/weights/best.pt...

Ultralytics 8.3.150 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)

Model summary (fused): 72 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	Size
00:00, 2.55it/s]							15/15 [00:05<

all	235	1932	0.753	0.7	0.741	0.457
white	85	158	0.81	0.759	0.78	0.503
blue_10	66	121	0.815	0.618	0.742	0.445
dred_15	67	124	0.662	0.621	0.7	0.414
black_8	89	164	0.924	0.665	0.811	0.484
purple_12	55	101	0.64	0.574	0.606	0.356
dred_7	64	122	0.693	0.648	0.647	0.409
orange_13	60	110	0.653	0.709	0.68	0.404
blue_2	73	136	0.76	0.745	0.747	0.497
red_3	70	126	0.7	0.659	0.705	0.418
green_6	69	129	0.823	0.814	0.838	0.539
green_14	65	115	0.83	0.757	0.802	0.452
red_11	72	140	0.69	0.686	0.708	0.461
yellow_1	80	147	0.784	0.782	0.835	0.543

```
orange_5      57      106     0.688     0.727     0.726     0.45
yellow_9      75      133     0.831     0.737     0.784     0.488
Speed: 0.5ms preprocess, 5.5ms inference, 0.0ms loss, 4.5ms postprocess per image
Results saved to runs/billar_balls_detection_v1
```

¡Entrenamiento completado!
Los resultados se guardaron en: runs/billar_balls_detection_v1
Puedes revisar los gráficos de entrenamiento y las métricas allí.
Carpeta 'runs' comprimida en 'runs.zip'

Paso 9: Análisis de los datos de entrenamiento

¡Hola! ¡Excelente que ya estés entrenando modelos! Analicemos estos logs de entrenamiento de tu modelo YOLOv8 para la detección de bolas de billar.

Veo que has realizado varios entrenamientos, reanudando a partir del anterior, lo cual es una buena práctica. Vamos a desglosar la información:

Análisis General de los Entrenamientos

Has realizado 4 sesiones de entrenamiento, incrementando progresivamente el número total de épocas y ajustando ligeramente el tamaño del lote (`batch`):

1. Primer entrenamiento (`billar_balls_detection_v1`):

- Modelo base: `yolov8n.pt` (un modelo preentrenado pequeño, bueno para empezar).
- Épocas objetivo: 50 (completaste 6).
- Tamaño de lote: 8.

2. Segundo entrenamiento (`billar_balls_detection_v12`):

- Reanudado desde: `runs/billar_balls_detection_v1/weights/last.pt`.
- Épocas objetivo: 10 (completaste 4, sumando 10 épocas totales al modelo).
- Tamaño de lote: 12.

3. Tercer entrenamiento (`billar_balls_detection_v13`):

- Reanudado desde: `runs/billar_balls_detection_v12/weights/last.pt`.
- Épocas objetivo: 10 (completaste 8, sumando 18 épocas totales al modelo).
- Tamaño de lote: 12.

4. Cuarto entrenamiento (`billar_balls_detection_v14`):

- Reanudado desde: `runs/billar_balls_detection_v13/weights/last.pt`.
- Épocas objetivo: 5 (completaste 5, sumando 23 épocas totales al modelo).
- Tamaño de lote: 14.

Configuración Clave del Entorno y Modelo:

- CPU Training:** Estás entrenando en una CPU (AMD Ryzen 7 5800X3D). Esto es lo más crítico a notar, ya que el entrenamiento de redes neuronales es significativamente más lento en CPU que en GPU. Los tiempos por época (alrededor de 35-38 minutos en los primeros entrenamientos) lo confirman.
- Modelo:** `yolov8n` (nano), el más pequeño de la familia YOLOv8. Tiene 3.01 millones de parámetros.
- Dataset:**
 - Imágenes de entrenamiento: 5523
 - Imágenes de validación: 235
 - Número de clases: 15 (originalmente el `yolov8n.pt` tiene 80, pero se ha ajustado a tus 15 clases de bolas de billar).

- **Tamaño de imagen (`imgsz`)**: 1080, ajustado automáticamente a 1088 (para ser múltiplo del stride máximo de la red, 32).
- **Optimizador**: Seleccionado automáticamente como `AdamW` con una tasa de aprendizaje (`lr`) inicial de `0.000526`.
- **Aumentación de datos**: Los parámetros como `flipr` (giro horizontal), `hsv_h`, `hsv_s`, `hsv_v` (ajustes de color), `translate`, `scale`, etc., indican que se está aplicando aumentación de datos, lo cual es bueno para mejorar la robustez del modelo.

Progreso del Rendimiento (Métricas de Validación):

Las métricas clave que observamos son:

- `Box(P)`: Precisión de la caja delimitadora.
- `R`: Recall (Sensibilidad) de la caja delimitadora.
- `mAP50`: Mean Average Precision con un IoU (Intersection over Union) de 0.5. Es una métrica común para evaluar qué tan bien el modelo detecta y localiza objetos.
- `mAP50-95`: Mean Average Precision promediado sobre múltiples umbrales de IoU (de 0.5 a 0.95, en pasos de 0.05). Es una métrica más estricta.

Veamos la evolución del `mAP50` y `mAP50-95` al final de cada sesión de entrenamiento:

Entrenamiento (Épocas Totales)	<code>mAP50</code>	<code>mAP50-95</code>	<code>Box(P)</code>	<code>R</code>	<code>box_loss</code>	<code>cls_loss</code>	<code>dfl_loss</code>
1º (6 épocas)	0.479	0.279	0.453	0.516	1.332	2.902	1.316
... (fin 1º, epoch 6)	0.734	0.442	0.716	0.697	1.008	1.144	1.112
2º (10 épocas)	0.761	0.464	0.721	0.729	0.9022	0.8564	1.048
3º (18 épocas)	0.766	0.471	0.746	0.722	0.754	0.5566	0.9692
4º (23 épocas)	0.767	0.478	0.748	0.737	0.7622	0.5849	0.9759
Validación Final (best.pt)	0.767	0.477	0.748	0.738	N/A	N/A	N/A

Observaciones del Rendimiento:

1. **Mejora Continua**: El modelo muestra una clara mejora a lo largo de las épocas. Las pérdidas (`box_loss`, `cls_loss`, `dfl_loss`) tienden a disminuir, y las métricas de precisión y recall (P, R, mAP) tienden a aumentar. Esto indica que el modelo está aprendiendo.
2. **Estabilización Reciente**: En las últimas épocas del cuarto entrenamiento, la mejora en `mAP50-95` es más lenta (de 0.471 a 0.478). Esto podría indicar que se está acercando a un límite con la configuración actual y el número de épocas en CPU, o que necesita más épocas para seguir afinando.
3. **Pérdidas (`loss`)**:
 - `box_loss` (pérdida de localización de la caja): Ha bajado de ~1.33 a ~0.76.
 - `cls_loss` (pérdida de clasificación): Ha bajado significativamente de ~2.9 a ~0.58. Esto es una buena señal de que el modelo está aprendiendo a distinguir las clases.
 - `dfl_loss` (Distribution Focal Loss, relacionada con la precisión de la caja): Ha bajado de ~1.31 a ~0.97.
4. **Resultados por Clase (del `best.pt` del 4º entrenamiento)**:
 - **Mejor rendimiento**: `green_6` (mAP50: 0.873, mAP50-95: 0.585), `yellow_1` (mAP50: 0.852, mAP50-95: 0.566), `yellow_9` (mAP50: 0.831, mAP50-95: 0.547).
 - **Peor rendimiento**: `purple_12` (mAP50: 0.607, mAP50-95: 0.359).
 - Hay una variabilidad considerable en el rendimiento entre las diferentes clases.

Puntos Positivos:

- **Elección del Modelo Base**: Empezar con `yolov8n.pt` preentrenado es una excelente estrategia. Acelera el aprendizaje y generalmente da mejores resultados que entrenar desde cero.

- **Transferencia de Pesos:** El log indica "Transferred 319/355 items from pretrained weights" (en el primer entreno) y "Transferred 355/355 items..." en los siguientes, lo que es correcto al reanudar.
- **Dataset:** Tienes un número considerable de imágenes de entrenamiento (5523).
- **Progreso:** El modelo está aprendiendo y mejorando sus métricas consistentemente.
- **Gestión del Entrenamiento:** Reanudar entrenamientos es una habilidad útil, especialmente con recursos limitados o sesiones largas.

Áreas de Mejora y Recomendaciones:

1. **¡UTILIZAR GPU!** Esta es la recomendación MÁS IMPORTANTE. Entrenar en CPU es demasiado lento.
 - **Google Colab:** Puedes usar GPUs gratuitas (K80, T4) en Google Colab. Sube tu dataset y código allí. Esto reducirá el tiempo por época de ~35-40 minutos a probablemente unos pocos minutos o incluso segundos, dependiendo de la GPU.
 - **GPU Local:** Si tienes una GPU NVIDIA compatible con CUDA en tu ordenador, asegúrate de que PyTorch la está utilizando. Los logs indican `torch-2.7.0+cu126 CPU`, lo que sugiere que o no tienes GPU, o PyTorch no está configurado para usarla. El aviso `Can't initialize NVML` también apunta a esto.
2. **Entrenar Durante Más Épocas (en GPU):** Una vez que estés en GPU, podrás entrenar durante muchas más épocas (e.g., 50, 100, o incluso más) de forma continua. Dado que el modelo seguía mejorando, es probable que se beneficie de más entrenamiento.
3. **Analizar Clases con Bajo Rendimiento:** Investiga por qué la bola `purple_12` tiene un rendimiento inferior.
 - ¿Hay suficientes ejemplos de esta bola en el dataset?
 - ¿Las imágenes son claras y bien etiquetadas?
 - ¿Es visualmente muy similar a otras bolas o al fondo en ciertas condiciones de iluminación?
 - Considera si necesitas más datos o una aumentación de datos más específica para estas clases.
4. **Tamaño del Lote (batch):** Lo has ido aumentando ($8 \rightarrow 12 \rightarrow 14$). En GPU, podrás probar tamaños de lote mayores (e.g., 16, 32, o más, dependiendo de la memoria de la GPU). Un tamaño de lote mayor puede llevar a una convergencia más estable y rápida, pero consume más memoria.
5. **Aumentación de Datos:** Los valores por defecto de YOLOv8 son generalmente buenos. Puedes experimentar con ellos si el rendimiento se estanca, pero primero céntrate en entrenar más tiempo en GPU.
6. **Evaluar best.pt:** Al final de cada entrenamiento, YOLOv8 guarda `last.pt` (los pesos de la última época) y `best.pt` (los pesos de la época que tuvo el mejor `mAP50-95` en el conjunto de validación). Asegúrate de usar `best.pt` para tus inferencias finales. La validación final que hiciste al final del cuarto entrenamiento sobre `best.pt` es la forma correcta de obtener las métricas finales.
7. **Incrementar el Conjunto de Validación:** 235 imágenes de validación es un poco justo. Si es posible, intenta tener un conjunto de validación más grande (e.g., 10-20% del total de tus datos) para tener una estimación más robusta del rendimiento del modelo.

Conclusión Preliminar:

Has hecho un buen trabajo iniciando el entrenamiento y el modelo está aprendiendo correctamente la tarea de detectar las 15 clases de bolas de billar. Las métricas de `mAP50` de ~0.76 y `mAP50-95` de ~0.47 son un punto de partida razonable para 23 épocas en CPU con `yolov8n`.

El paso más crucial ahora es mover tu entorno de entrenamiento a una GPU. Esto te permitirá experimentar mucho más rápido y entrenar el modelo durante más tiempo para alcanzar un rendimiento superior.

¡Sigue adelante! Cuando tengas más resultados después de entrenar en GPU, podemos volver a analizarlos.

Tabla de resultados CPU

A continuación, se presenta una tabla consolidada con los datos de las cuatro sesiones de entrenamiento del modelo de detección de bolas de billar. Las épocas son acumulativas.

Época Total	<code>box_loss</code> (train)	<code>cls_loss</code> (train)	<code>dfl_loss</code> (train)	Precisión (val)	Recall (val)	<code>mAP50</code> (val)	<code>mAP50-95</code> (val)
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Sesión 1 (6 épocas)							
1	1.332	2.902	1.316	0.453	0.516	0.479	0.279
2	1.199	1.918	1.232	0.592	0.643	0.646	0.381
3	1.133	1.625	1.186	0.607	0.621	0.643	0.372
4	1.082	1.409	1.153	0.666	0.662	0.686	0.397
5	1.058	1.270	1.137	0.728	0.683	0.740	0.414
6	1.008	1.144	1.112	0.716	0.697	0.734	0.442
Sesión 2 (4 épocas adicionales, reanudando desde época 6)							
7	0.9206	0.9779	1.062	0.723	0.729	0.760	0.458
8	0.9246	0.9450	1.060	0.724	0.683	0.742	0.446
9	0.9198	0.9079	1.057	0.713	0.701	0.728	0.441
10	0.9022	0.8564	1.048	0.721	0.729	0.761	0.464
Sesión 3 (8 épocas adicionales, reanudando desde época 10)							
11	0.8221	0.7072	1.006	0.747	0.696	0.755	0.468
12	0.8145	0.6926	0.9995	0.773	0.690	0.762	0.463
13	0.8156	0.6934	0.9997	0.744	0.678	0.733	0.438
14	0.8201	0.6832	1.003	0.708	0.712	0.736	0.441
15	0.8335	0.6943	1.010	0.728	0.708	0.740	0.450
16	0.8146	0.6468	1.000	0.731	0.723	0.750	0.459
17	0.7813	0.5947	0.9802	0.763	0.709	0.768	0.471
18	0.7540	0.5566	0.9692	0.746	0.722	0.766	0.471
Sesión 4 (5 épocas adicionales, reanudando desde época 18)							
19	0.8287	0.6997	1.008	0.762	0.699	0.767	0.470
20	0.8214	0.6790	1.006	0.760	0.698	0.761	0.467
21	0.8136	0.6628	1.001	0.738	0.708	0.751	0.458
22	0.7887	0.6268	0.9902	0.752	0.708	0.759	0.460
23	0.7622	0.5849	0.9759	0.748	0.737	0.767	0.478

Nota sobre las métricas:

- `box_loss`, `cls_loss`, `dfl_loss`: Son las pérdidas calculadas sobre el conjunto de **entrenamiento**. Valores más bajos son mejores.
- `Precisión (val)`, `Recall (val)`, `mAP50 (val)`, `mAP50-95 (val)`: Son métricas calculadas sobre el conjunto de **validación**. Valores más altos son mejores.
 - **Precisión (P)**: De las detecciones realizadas, cuántas fueron correctas.
 - **Recall (R)**: De todos los objetos reales, cuántos fueron detectados.
 - **mAP50**: Mean Average Precision con un umbral de Intersection over Union (IoU) de 0.50.

- **mAP50-95:** Mean Average Precision promediada sobre umbrales de IoU desde 0.50 hasta 0.95 (más estricta).

Al final del cuarto entrenamiento (época 23), las métricas de validación para el mejor modelo ([best.pt](#)) fueron:

- Precisión (P): 0.748
- Recall (R): 0.738
- mAP50: 0.767
- mAP50-95: 0.477

Estos datos muestran una mejora general del modelo a medida que avanzan las épocas, aunque con algunas fluctuaciones, lo cual es normal durante el proceso de entrenamiento.

Paso 10: Pruebas de detección

Modelo Base sin entrenamiento:



1 epoch de entrenamiento:



6 epoch de entrenamiento:



16 epoch de entrenamiento:



36 epoch de entrenamiento:



Paso 11: Hipótesis mínimos locales?

¡Hola!

Sí, lo que comentas tiene mucho sentido y es una observación muy pertinente. Vamos a desglosarlo:

1. Estancamiento en un Máximo/Mínimo Local:

- En el entrenamiento de redes neuronales, el objetivo es minimizar una "función de pérdida" (loss function). Esta función se puede visualizar como un paisaje con valles (mínimos) y montañas (máximos). Buscamos el valle más profundo, que es el "mínimo global".
- Sin embargo, es muy común que el optimizador (el algoritmo que ajusta los pesos de la red) encuentre un "mínimo local", es decir, un valle que es el punto más bajo en su vecindad inmediata, pero no el más bajo de todo el paisaje. Una vez que el optimizador cae en este mínimo local, puede ser difícil que salga de él si los "pasos" que da (determinados por la tasa de aprendizaje o *learning rate*) no son lo suficientemente grandes o si no hay suficiente "impulso" (momentum) para superar las "laderas" que lo rodean.
- El resultado es que la pérdida deja de disminuir significativamente, o las métricas de evaluación (como el mAP en detección de objetos) se estancan, a pesar de que el modelo podría mejorar más.

2. Entrenamiento en Local con CPU y sus Implicaciones:

- **Velocidad:** Entrenar en CPU es drásticamente más lento que en GPU. Esto significa que en el mismo tiempo de pared, realizas muchas menos épocas o iteraciones.
- **Exploración del Espacio de Pérdida:** Con menos iteraciones, el optimizador tiene menos oportunidades de explorar diferentes regiones del "paisaje de pérdida". Si cae en un mínimo local temprano, y el entrenamiento es muy lento, puede parecer que está estancado permanentemente cuando, con más tiempo (muchísimo más en CPU), quizás podría haber salido.
- **Tamaño del Lote (Batch Size) y Recursos:** A veces, por limitaciones de memoria en CPU o para intentar acelerar algo el proceso, se pueden usar tamaños de lote más pequeños. Tamaños de lote muy pequeños pueden introducir más ruido en la estimación del gradiente, lo que a veces ayuda a escapar de mínimos locales "sharps" (agudos), pero también puede hacer el entrenamiento más inestable y lento en converger.

- **Experimentación con Hiperparámetros:** La lentitud de la CPU dificulta la experimentación rápida con diferentes tasas de aprendizaje, optimizadores, o planificadores de tasa de aprendizaje (*learning rate schedulers*), que son cruciales para evitar o escapar de mínimos locales.

3. Entrenamiento en Google Colab (Probablemente con GPU) desde Cero (con [yolov8n.pt](#)):

- **Velocidad de GPU:** Al pasarte a Colab y (presumiblemente) usar una GPU, el entrenamiento es mucho más rápido. Puedes completar muchas más épocas en un tiempo razonable.
- **Mayor Exploración:** Más épocas significan que el optimizador tiene más "tiempo" y "pasos" para explorar el paisaje de la función de pérdida. Esto aumenta la probabilidad de que pueda "saltar" fuera de mínimos locales menos profundos y encontrar mejores soluciones.
- **Iteración Rápida:** Puedes probar diferentes configuraciones de hiperparámetros mucho más rápido, lo cual es clave para encontrar una configuración que lleve a una mejor convergencia.
- **yolov8n.pt (Pesos Preentrenados):** Empezar con pesos preentrenados es casi siempre una buena idea. Estos pesos ya han sido entrenados en un dataset grande (generalmente COCO) y han aprendido características útiles. Esto significa que tu punto de partida en el "paisaje de pérdida" ya está en una región "prometedora", lo que puede ayudar a converger más rápido y a un mejor mínimo. Aunque empieces "desde cero" tu dataset específico, no empiezas desde cero los pesos de la red.

En resumen:

Tu sospecha es muy válida. El entrenamiento en CPU, debido a su lentitud inherente, puede hacer que el modelo parezca estancado en un mínimo local porque:

- No se han completado suficientes iteraciones para escapar.
- Es difícil experimentar con hiperparámetros que podrían ayudar a escapar.

Mover el entrenamiento a un entorno con GPU (como Colab) acelera drásticamente el proceso, permitiendo una exploración más exhaustiva del espacio de la función de pérdida y una iteración más rápida sobre los hiperparámetros, lo que aumenta las posibilidades de alcanzar un mínimo global o, al menos, un mínimo local mucho mejor.

¡Es una excelente decisión haber movido tu entrenamiento a Colab! Seguramente verás progreso más rápido y, potencialmente, mejores resultados.

Paso 12: Entrenamiento en Colab

Entrenamiento 1: 1 etapa, batch=8

```
Entrenamiento 1 (1 epoch):
epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics
1,290.533,1.3336,2.9018,1.32387,0.43148,0.53762,0.46397,0.27007,1.45193,1.8147,1.36843,0.00017508,0.00017508,0.0
```

Entrenamiento 2: 5 etapas, batch=10

```
Entrenamiento 2 (+5 epoch):
epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics
1,278.08,1.15441,1.90608,1.20099,0.57836,0.60056,0.60656,0.35768,1.42967,1.55922,1.35987,0.00017508,0.00017508,
2,549.027,1.13221,1.68542,1.18338,0.66006,0.65924,0.69108,0.41362,1.39255,1.34825,1.32856,0.000281031,0.0002810
3,821.173,1.08432,1.4742,1.14904,0.65852,0.68244,0.70073,0.40474,1.44016,1.2906,1.33199,0.000317551,0.000317551,
4,1104.82,1.02628,1.28943,1.11569,0.69827,0.69257,0.73202,0.43749,1.37721,1.21231,1.28874,0.000213556,0.00021355
5,1381.62,0.97867,1.15706,1.08714,0.74116,0.70751,0.76273,0.46402,1.37396,1.12487,1.30437,0.000109408,0.000109408
```

Entrenamiento 3: 10 etapas, batch=12

Cargando modelo YOLOv8n pre-entrenado...

Iniciando entrenamiento del modelo...

Ultralytics 8.3.148 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=12, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=custom_data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=runs/billar_balls_detection_v12/weights/last.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=billar_balls_detection_v13, nbs=64, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=runs, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=runs/billar_balls_detection_v13, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	754237	ultralytics.nn.modules.head.Detect	[15, [64, 128, 256]]

Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs

Transferred 355/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'

AMP: running Automatic Mixed Precision (AMP) checks...

AMP: checks passed ✅

WARNING ⚠️ imgsz=[1080] must be multiple of max stride 32, updating to [1088]

train: Fast image access ✅ (ping: 0.3±0.6 ms, read: 1655.7±423.2 MB/s, size: 159.4 KB)

train: Scanning /content/data/labels/train.cache... 5523 images, 0 backgrounds, 0 corrupt: 100% |██████████|
 5523/5523 [00:00<?, ?it/s]albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)),
 ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0),
 tile_grid_size=(8, 8))
 val: Fast image access ✓ (ping: 0.0±0.0 ms, read: 598.2±46.0 MB/s, size: 108.3 KB)
 val: Scanning /content/data/labels/val.cache... 235 images, 0 backgrounds, 0 corrupt: 100% |██████████| 235/235
 [00:00<?, ?it/s]
 Plotting labels to runs/billar_balls_detection_v13/labels.jpg...
 optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and
 'momentum' automatically...
 optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64
 weight(decay=0.00046875), 63 bias(decay=0.0)
 Image sizes 1088 train, 1088 val
 Using 2 dataloader workers
 Logging results to runs/billar_balls_detection_v13
 Starting training for 10 epochs...
 Closing dataloader mosaic
 albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01,
 method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/10	4.43G	0.9543	1.174	1.072	45	1088: 100% ██████████ 461/461 [04:31<00:00, 1.70it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 10/10 [00:04<0
	0:00, 2.04it/s]		all	235	1932	0.682 0.713 0.726 0.421

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/10	4.65G	0.9774	1.156	1.082	12	1088: 100% ██████████ 461/461 [04:20<00:00, 1.77it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 10/10 [00:04<0
	0:00, 2.37it/s]		all	235	1932	0.653 0.705 0.72 0.428

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/10	4.65G	0.9748	1.098	1.078	45	1088: 100% ██████████ 461/461 [04:27<00:00, 1.72it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 10/10 [00:04<0
	0:00, 2.38it/s]		all	235	1932	0.683 0.686 0.711 0.414

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
4/10	4.65G	0.9564	1.018	1.071	17	1088: 100% ██████████ 461/461 [04:28<00:00, 1.72it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 10/10 [00:05<0
	0:00, 1.74it/s]		all	235	1932	0.705 0.693 0.73 0.431

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
5/10	4.65G	0.9185	0.9	1.047	16	1088: 100% ██████████ 461/461 [04:32<00:00, 1.69it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 10/10 [00:07<0
	0:00, 1.41it/s]		all	235	1932	0.744 0.698 0.743 0.441

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
6/10	4.65G	0.8896	0.847	1.034	5	1088: 100% ██████████ 461/461 [04:33<00:00, 1.68it/s]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 10/10 [00:04<0
	0:00, 2.01it/s]		all	235	1932	0.704 0.723 0.742 0.446

```

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
7/10 4.65G 0.8584 0.7854 1.016      4    1088: 100%|██████████| 461/461 [04:32<00:00, 1.69it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 10/10 [00:05<00:00, 1.76it/s] all 235 1932 0.754 0.698 0.758 0.459

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
8/10 4.65G 0.8295 0.7117 0.9986      7    1088: 100%|██████████| 461/461 [04:32<00:00, 1.69it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 10/10 [00:07<00:00, 1.38it/s] all 235 1932 0.721 0.742 0.756 0.459

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
9/10 4.65G 0.7908 0.6643 0.9827      7    1088: 100%|██████████| 461/461 [04:33<00:00, 1.68it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 10/10 [00:05<00:00, 1.77it/s] all 235 1932 0.725 0.729 0.757 0.465

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
10/10 4.65G 0.7708 0.6254 0.97      8    1088: 100%|██████████| 461/461 [04:36<00:00, 1.67it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 10/10 [00:04<00:00, 2.49it/s] all 235 1932 0.735 0.735 0.768 0.477

```

10 epochs completed in 0.770 hours.

Optimizer stripped from runs/billar_balls_detection_v13/weights/last.pt, 6.3MB

Optimizer stripped from runs/billar_balls_detection_v13/weights/best.pt, 6.3MB

Validating runs/billar_balls_detection_v13/weights/best.pt...

Ultralytics 8.3.148 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)

Model summary (fused): 72 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 30%|██████| 3/10 [00:03<00:11, 1.67s/it] WARNING ⚠️ NMS time limit 3.200s exceeded

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 10/10 [00:14<00:00, 1.49s/it]

all 235 1932 0.736 0.734 0.767 0.476

white 85 158 0.752 0.747 0.763 0.465

blue_10 66 121 0.762 0.702 0.76 0.443

dred_15 67 124 0.755 0.629 0.724 0.436

black_8 89 164 0.862 0.687 0.81 0.48

purple_12 55 101 0.595 0.574 0.598 0.375

dred_7 64 122 0.725 0.705 0.751 0.481

orange_13 60 110 0.691 0.755 0.739 0.463

blue_2 73 136 0.735 0.775 0.765 0.503

red_3 70 126 0.7 0.778 0.791 0.463

green_6 69 129 0.771 0.86 0.874 0.538

green_14 65 115 0.794 0.748 0.8 0.461

red_11 72 140 0.702 0.707 0.74 0.493

yellow_1 80 147 0.74 0.816 0.848 0.553

orange_5 57 106 0.676 0.774 0.717 0.455

yellow_9 75 133 0.775 0.759 0.823 0.532

Speed: 0.6ms preprocess, 4.1ms inference, 0.0ms loss, 27.1ms postprocess per image

Results saved to runs/billar_balls_detection_v13

¡Entrenamiento completado!

Los resultados se guardaron en: runs/billar_balls_detection_v1

Puedes revisar los gráficos de entrenamiento y las métricas allí.

Entrenamiento 4: 20 etapas, batch=32

Cargando modelo YOLOv8n pre-entrenado...

Iniciando entrenamiento del modelo...

Ultralytics 8.3.148 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer:agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=32, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=custom_data.yaml, degrees=0.0, deterministic=True, device=None, dfi=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=20, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=1080, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=runs/billar_balls_detection_v13/weights/last.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=billar_balls_detection_v14, nbs=64, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=runs, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=runs/billar_balls_detection_v14, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12		-1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13		-1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15		-1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16		-1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18		-1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19		-1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21		-1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	754237	ultralytics.nn.modules.head.Detect	[15, [64, 128, 256]]

Model summary: 129 layers, 3,013,773 parameters, 3,013,757 gradients, 8.2 GFLOPs

Transferred 355/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'

AMP:running Automatic Mixed Precision (AMP) checks...

AMP:checks passed 

WARNING ! imgs=[1080] must be multiple of max stride 32, updating to [1088]
train:Fast image access ✓ (ping: 0.0±0.0 ms, read: 2400.3±439.6 MB/s, size: 159.4 KB)

train:Scanning /content/data/labels/train.cache... 5523 images, 0 backgrounds, 0 corrupt: 100%|██████████| 5
523/5523 [00:00<?, ?it/s]

albumentations:Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weigh
ted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

val:Fast image access ✓ (ping: 1.5±3.3 ms, read: 504.7±46.6 MB/s, size: 108.3 KB)

val:Scanning /content/data/labels/val.cache... 235 images, 0 backgrounds, 0 corrupt: 100%|██████████| 235/2
35 [00:00<?, ?it/s]

Plotting labels to runs/billar_balls_detection_v14/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0'
and 'momentum' automatically...
optimizer: AdamW(lr=0.000526, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=
0.0005), 63 bias(decay=0.0)
Image sizes 1088 train, 1088 val
Using 2 dataloader workers
Logging results to runs/billar_balls_detection_v14
Starting training for 20 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/20	14.2G	0.8475	0.7639	1.012	316	1088: 100% ██████████ 173/173 [05:00<00:00, 1.74 s/it] Class Images Instances Box(P R mAP50 mAP50-95): 100% ██████████ 4/4 [00:08<0 0:00, 2.23s/it]
all	235	1932	0.719	0.694	0.745	0.458

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/20	11.8G	0.8555	0.7515	1.017	285	1088: 100% ██████████ 173/173 [04:49<00:00, 1.67 s/it] Class Images Instances Box(P R mAP50 mAP50-95): 100% ██████████ 4/4 [00:04<0 0:00, 1.07s/it]
all	235	1932	0.742	0.726	0.768	0.456

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

3/20	12.6G	0.8627	0.7609	1.018	228	1088: 100%	[██████████]	173/173 [04:56<00:00, 1.71s/it]	
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	[██████████]	4/4 [00:05<00:00, 1.26s/it]

all	235	1932	0.689	0.713	0.72	0.425			
-----	-----	------	-------	-------	------	-------	--	--	--

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
4/20	13.5G	0.8607	0.7454	1.018	280	1088: 100%	[██████████]	173/173 [05:01<00:00, 1.74s/it]	
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	[██████████]	4/4 [00:04<00:00, 1.06s/it]

all	235	1932	0.689	0.704	0.723	0.437			
-----	-----	------	-------	-------	-------	-------	--	--	--

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
5/20	12.2G	0.847	0.7122	1.009	251	1088: 100%	[██████████]	173/173 [04:55<00:00, 1.71s/it]	
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	[██████████]	4/4 [00:05<00:00, 1.30s/it]

all	235	1932	0.72	0.729	0.759	0.461			
-----	-----	------	------	-------	-------	-------	--	--	--

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
6/20	14.3G	0.8332	0.6966	1.005	309	1088: 100%	[██████████]	173/173 [04:57<00:00, 1.72s/it]	
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	[██████████]	4/4 [00:04<00:00, 1.00s/it]

all	235	1932	0.722	0.708	0.737	0.443			
-----	-----	------	-------	-------	-------	-------	--	--	--

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
7/20	13.5G	0.814	0.6625	0.9963	219	1088: 100%	[██████████]	173/173 [04:56<00:00, 1.72s/it]	
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%	[██████████]	4/4 [00:04<00:00, 1.08s/it]

all	235	1932	0.724	0.717	0.75	0.464
-----	-----	------	-------	-------	------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

8/20	14.3G	0.7908	0.6317	0.9863	259	1088: 100% ██████████ 173/173 [04:57<00:00, 1.72s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:05<0:00, 1.37s/it]

all	235	1932	0.723	0.691	0.73	0.426
-----	-----	------	-------	-------	------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

9/20	12.3G	0.7725	0.6128	0.9784	294	1088: 100% ██████████ 173/173 [04:58<00:00, 1.73s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:05<0:00, 1.35s/it]

all	235	1932	0.77	0.693	0.757	0.459
-----	-----	------	------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

10/20	12.5G	0.7721	0.5992	0.9787	247	1088: 100% ██████████ 173/173 [04:58<00:00, 1.73s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:09<0:00, 2.35s/it]

all	235	1932	0.738	0.717	0.747	0.447
-----	-----	------	-------	-------	-------	-------

Closing dataloader mosaic

albumentations:Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

11/20	11.2G	0.7266	0.526	0.9506	112	1088: 100% ██████████ 173/173 [04:54<00:00, 1.70s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:09<0:00, 2.45s/it]

all	235	1932	0.724	0.703	0.745	0.448
-----	-----	------	-------	-------	-------	-------

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
12/20	11.3G	0.7004	0.4864	0.9416	166	1088: 100% ██████████ 173/173 [04:45<00:00, 1.65s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:04<00:00, 1.23s/it]
	all	235	1932	0.74	0.7	0.748 0.462
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
13/20	11.2G	0.694	0.4758	0.9339	132	1088: 100% ██████████ 173/173 [04:46<00:00, 1.66s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:04<00:00, 1.04s/it]
	all	235	1932	0.75	0.692	0.742 0.452
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
14/20	11.3G	0.6724	0.4519	0.9249	204	1088: 100% ██████████ 173/173 [04:47<00:00, 1.66s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:04<00:00, 1.24s/it]
	all	235	1932	0.734	0.71	0.748 0.461
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
15/20	11.2G	0.6613	0.4364	0.9208	187	1088: 100% ██████████ 173/173 [04:44<00:00, 1.65s/it]
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ██████████ 4/4 [00:05<00:00, 1.34s/it]
	all	235	1932	0.743	0.695	0.744 0.461
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
16/20	11.3G	0.6452	0.4274	0.9149	113	1088: 100% ██████████ 173/173 [04:42<00:00, 1.63s/it]

	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% ██████████ 4/4 [00:05<0
0:00, 1.33s/it]							
<hr/>							
	all	235	1932	0.736	0.725	0.75	0.462
<hr/>							
<hr/>							
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
17/20	11.2G	0.6284	0.4096	0.9094	101	1088: 100% ██████████ 173/173 [04:37<00:00, 1.6	
1s/it]	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% ██████████ 4/4 [00:08<0
0:00, 2.02s/it]							
<hr/>							
	all	235	1932	0.757	0.7	0.753	0.465
<hr/>							
<hr/>							
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
18/20	11.1G	0.6152	0.3937	0.9036	156	1088: 100% ██████████ 173/173 [04:34<00:00, 1.5	
9s/it]	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% ██████████ 4/4 [00:03<0
0:00, 1.04it/s]							
<hr/>							
	all	235	1932	0.751	0.715	0.756	0.469
<hr/>							
<hr/>							
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
19/20	11.1G	0.6026	0.3882	0.8993	183	1088: 100% ██████████ 173/173 [04:39<00:00, 1.6	
1s/it]	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% ██████████ 4/4 [00:08<0
0:00, 2.17s/it]							
<hr/>							
	all	235	1932	0.73	0.725	0.758	0.472
<hr/>							
<hr/>							
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
20/20	11.3G	0.5888	0.3708	0.8935	119	1088: 100% ██████████ 173/173 [04:32<00:00, 1.	
57s/it]	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% ██████████ 4/4 [00:04<0
0:00, 1.22s/it]							
<hr/>							
	all	235	1932	0.759	0.715	0.757	0.473
<hr/>							
<hr/>							

20 epochs completed in 1.659 hours.
 Optimizer stripped from runs/billar_balls_detection_v14/weights/last.pt, 6.3MB
 Optimizer stripped from runs/billar_balls_detection_v14/weights/best.pt, 6.3MB

Validating runs/billar_balls_detection_v14/weights/best.pt...
 Ultralytics 8.3.148 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
 Model summary (fused): 72 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P) R mAP50 mAP50-95): 100% |██████████| 4/4 [00:13<0 0:00, 3.28s/it]

	all	235	1932	0.76	0.715	0.757	0.473
white	85	158	0.801	0.734	0.758	0.498	
blue_10	66	121	0.828	0.686	0.767	0.459	
dred_15	67	124	0.652	0.677	0.689	0.42	
black_8	89	164	0.89	0.726	0.838	0.503	
purple_12	55	101	0.67	0.624	0.676	0.398	
dred_7	64	122	0.673	0.674	0.683	0.421	
orange_13	60	110	0.71	0.712	0.693	0.429	
blue_2	73	136	0.771	0.766	0.772	0.515	
red_3	70	126	0.728	0.683	0.743	0.454	
green_6	69	129	0.808	0.853	0.854	0.559	
green_14	65	115	0.832	0.777	0.817	0.46	
red_11	72	140	0.709	0.642	0.719	0.479	
yellow_1	80	147	0.779	0.769	0.814	0.534	
orange_5	57	106	0.725	0.689	0.717	0.443	
yellow_9	75	133	0.819	0.713	0.809	0.52	

Speed: 0.5ms preprocess, 3.9ms inference, 0.0ms loss, 12.2ms postprocess per image

Results saved to runs/billar_balls_detection_v14

¡Entrenamiento completado!

Los resultados se guardaron en: runs/billar_balls_detection_v1

Puedes revisar los gráficos de entrenamiento y las métricas allí.

Carpeta 'runs' comprimida en 'runs.zip'

Tabla de Datos de evolución de todos los entrenamientos

Entrenamiento 1 (1 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,290.533,1.3336,2.9018,1.32387,0.43148,0.53762,0.46397,0.27007,1.45193,1.8147,1.36843,0.00017508,0.00017508,0.00017508
```

Entrenamiento 2 (+5 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,278.08,1.15441,1.90608,1.20099,0.57836,0.60056,0.60656,0.35768,1.42967,1.55922,1.35987,0.00017508,0.00017508,0.00017508,
2,549.027,1.13221,1.68542,1.18338,0.66006,0.65924,0.69108,0.41362,1.39255,1.34825,1.32856,0.000281031,0.000281031,0.000281031
3,821.173,1.08432,1.4742,1.14904,0.65852,0.68244,0.70073,0.40474,1.44016,1.2906,1.33199,0.000317551,0.000317551,0.000317551,
4,1104.82,1.02628,1.28943,1.11569,0.69827,0.69257,0.73202,0.43749,1.37721,1.21231,1.28874,0.000213556,0.000213556,0.000213556
5,1381.62,0.97867,1.15706,1.08714,0.74116,0.70751,0.76273,0.46402,1.37396,1.12487,1.30437,0.000109408,0.000109408,0.000109408
```

Entrenamiento 3 (+10 epoch):

```
epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B)
1,276.783,0.95433,1.17408,1.07229,0.68246,0.71304,0.72592,0.42111,1.46286,1.25221,1.34035,0.000174953,0.000174953,0.000174953
```

2,542.255,0.97743,1.15569,1.08168,0.65295,0.70512,0.71997,0.42797,1.46006,1.27096,1.32885,0.000315608,0.0003156
 3,815.243,0.97484,1.098,1.07828,0.68277,0.68609,0.71074,0.4139,1.443,1.29904,1.31627,0.000421547,0.000421547,0.
 4,1089.98,0.95643,1.01756,1.07069,0.70529,0.69335,0.72988,0.43142,1.44943,1.18283,1.34102,0.000369778,0.000369778,
 5,1370.14,0.91851,0.90003,1.0466,0.74406,0.69784,0.74333,0.44136,1.4417,1.18029,1.3114,0.000317704,0.000317704,0.
 6,1649.71,0.8896,0.84702,1.03414,0.70355,0.72301,0.74173,0.44642,1.42989,1.12987,1.33109,0.00026563,0.00026563
 7,1928.53,0.85845,0.78538,1.01628,0.75448,0.69757,0.75834,0.45948,1.39332,1.07743,1.29665,0.000213556,0.000213556
 8,2209.02,0.82945,0.71171,0.9986,0.72106,0.74154,0.75639,0.45909,1.41394,1.09063,1.29768,0.000161482,0.000161482
 9,2489.53,0.79084,0.66427,0.98271,0.72543,0.72885,0.75705,0.46479,1.40511,1.09018,1.29669,0.000109408,0.000109408
 10,2770.36,0.7708,0.6254,0.97004,0.7353,0.73459,0.76781,0.47668,1.3928,1.06222,1.30255,5.7334e-05,5.7334e-05,5.

Entrenamiento 4 (+20 epoch):

epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics
 1,309.773,0.84753,0.76388,1.01188,0.71906,0.69409,0.74535,0.45833,1.39514,1.16028,1.28237,0.00017432,0.00017432
 2,606.662,0.85545,0.75146,1.0168,0.74202,0.72613,0.76783,0.45612,1.41413,1.08617,1.30536,0.000332345,0.0003323
 3,911.291,0.86269,0.76093,1.01752,0.68912,0.71313,0.71958,0.42528,1.44163,1.14007,1.3351,0.000473013,0.000473013
 4,1219.43,0.86067,0.74542,1.01793,0.68942,0.70397,0.7233,0.43657,1.43197,1.23258,1.33114,0.000447889,0.000447889
 5,1522.73,0.84704,0.71222,1.00944,0.71989,0.72867,0.75881,0.46064,1.40521,1.08517,1.32346,0.000421852,0.000421852
 6,1828.21,0.83319,0.69657,1.00451,0.72205,0.70781,0.7368,0.44298,1.41773,1.09567,1.30886,0.000395815,0.000395815
 7,2132.08,0.81396,0.66251,0.99627,0.72446,0.71693,0.7504,0.46407,1.37595,1.07494,1.31485,0.000369778,0.000369778
 8,2437.38,0.79075,0.63173,0.98632,0.72305,0.69066,0.72964,0.42634,1.47498,1.15218,1.36066,0.000343741,0.00034
 9,2744.07,0.77246,0.61277,0.97836,0.76951,0.69297,0.75737,0.45891,1.39774,1.06591,1.32567,0.000317704,0.0003177
 10,3055.02,0.77207,0.5992,0.97872,0.73787,0.71675,0.74712,0.44741,1.42581,1.06958,1.3572,0.000291667,0.000291667
 11,3362.18,0.72656,0.52597,0.95059,0.72434,0.70258,0.74517,0.44791,1.40841,1.14299,1.32578,0.00026563,0.0002656
 12,3655,0.70035,0.48638,0.9416,0.74016,0.70017,0.74754,0.46204,1.38198,1.07728,1.31126,0.000239593,0.00023959
 13,3948.33,0.69397,0.47581,0.93394,0.75001,0.69218,0.7421,0.45226,1.40783,1.11848,1.31191,0.000213556,0.0002135
 14,4244.27,0.67237,0.45193,0.9249,0.73399,0.7102,0.74767,0.46051,1.38979,1.11095,1.30924,0.000187519,0.000187519
 15,4537.13,0.66129,0.43641,0.92083,0.7434,0.69465,0.74361,0.46054,1.38656,1.10907,1.31941,0.000161482,0.0001614
 16,4829.63,0.64525,0.42736,0.9149,0.73647,0.72475,0.74978,0.46246,1.38654,1.07662,1.31435,0.000135445,0.00013
 17,5117.9,0.62841,0.40962,0.90938,0.75744,0.70042,0.75268,0.46465,1.3961,1.07014,1.3322,0.000109408,0.0001094
 18,5398.27,0.61515,0.39373,0.9036,0.75149,0.71542,0.75577,0.46859,1.37802,1.06809,1.31433,8.3371e-05,8.3371e-05
 19,5690.15,0.60264,0.38822,0.89934,0.73046,0.72462,0.75806,0.47222,1.36735,1.07489,1.32028,5.7334e-05,5.7334e
 20,5969.94,0.58883,0.37082,0.89354,0.759,0.71454,0.75654,0.47252,1.3648,1.0663,1.3176,3.1297e-05,3.1297e-05,3.1