Bloque 11: Informe Final

FASE 1

1. Entendimiento y Caracterización del Dataset de Imágenes Etiquetadas

El proyecto parte de la base de un dataset de imágenes de bolas de billar ya etiquetadas. La comprensión de este dataset es fundamental para los pasos siguientes.

Formato de Anotaciones: Las anotaciones se encuentran en un archivo CSV (*annotations.csv*), lo cual es una ventaja significativa. Este archivo contiene las siguientes columnas clave:

- filename: Nombre del archivo de la imagen.
- width, height: Dimensiones de la imagen.
- class: La clase o tipo de la bola detectada.
- xmin, ymin, xmax, ymax: Coordenadas del "bounding box" (caja delimitadora) para cada objeto. Estas coordenadas definen la esquina superior izquierda (xmin, ymin) y la esquina inferior derecha (xmax, ymax).
- "Las coordenadas xmin, ymin, xmax, ymax son el formato estándar para la detección de objetos y son directamente utilizables por la mayoría de los frameworks de Deep Learning como TensorFlow."

Clases de las Bolas: Un aspecto crucial es que las bolas están etiquetadas individualmente, incluyendo tanto el color como el número (ej., blue_10, red_3, yellow_9), además de white para la bola blanca y black_8 para la bola 8.

- "Esto es ideal, ya que te permitirá identificar cada bola de manera única." Las clases identificadas en el CSV son: white, blue_10, dred_15, black_8, purple_12, dred_7, orange_13, blue_2, red_3, green_6, green_14, red_11, yellow_1, orange_5, yellow_9. Es fundamental mantener un orden consistente para la asignación de IDs.

Tamaño del Dataset: Aunque no se especifica un número exacto de imágenes, se reconoce que "un dataset más grande (cientos o miles de imágenes) permitirá un mejor entrenamiento". La estructura actual del dataset indica que ya está pre-dividido en conjuntos de train, test y valid.

2. Selección de la Arquitectura del Modelo y Preparación del Entorno

La elección de la arquitectura del modelo es el siguiente paso lógico, seguida de la configuración del entorno de desarrollo.

Arquitectura del Modelo (YOLO): Se recomienda el uso de una arquitectura **YOLO (You Only Look Once)** para la detección de objetos. Las razones para esta elección incluyen:

- **Eficiencia en tiempo real:** "YOLO es conocido por su velocidad, lo que es útil si en el futuro quieres procesar videos."
- **Buen balance entre precisión y velocidad:** Versiones más recientes (YOLOv5, v7, v8) ofrecen muy buenos resultados.
- **Comunidad y recursos:** Amplio soporte y tutoriales disponibles.

Framework de Deep Learning: Se utilizará **TensorFlow**, integrando Keras para una interfaz de alto nivel que facilita el trabajo.

Preparación del Entorno de Desarrollo: Se requiere la siguiente configuración:

- **Python y Anaconda/Miniconda:** Confirmada la instalación de Python 3.12.3.
- **Entorno virtual:** Creación de un entorno virtual (billar_dl) para gestionar las dependencias.
- Comando sugerido: conda create -n billar_dl python=3.9 y conda activate billar_dl.
- **Instalación de Librerías:**pip install tensorflow (por defecto, versión CPU).
- pip install pandas numpy matplotlib opencv-python.
- **GPU (Opcional pero Recomendado):** Se menciona la posibilidad de usar una tarjeta gráfica NVIDIA compatible con CUDA para "acelerar drásticamente el entrenamiento", aunque se puede comenzar con la versión de CPU.

3. Preparación del Dataset para YOLO (Conversión de Anotaciones)

Aunque el CSV es útil, los modelos YOLO (especialmente YOLOv5 o YOLOv8) requieren un formato de anotación específico y una estructura de carpetas particular.

Formato YOLO (.txt): Para cada imagen, se debe crear un archivo .txt con el mismo nombre que la imagen (ej. imagen1.txt para imagen1.jpg). Cada línea en este archivo representa un objeto detectado y sigue el formato: class_id center_x center_y width height

- class_id: ID numérico de la clase (comenzando desde 0), que se mapeará desde los nombres de clase (ej. white → 0).
- center_x, center_y: Coordenadas X e Y del centro del bounding box, normalizadas (valor entre 0 y 1).
- width, height: Ancho y alto del bounding box, normalizados (valor entre 0 y 1).
- "Este formato se basa en normalizar las coordenadas de los bounding boxes a un rango de 0 a 1, y además, requiere que cada archivo de anotación sea un archivo de texto por imagen, no un único CSV para todas las imágenes."

Esquema de Clases: Es necesario un archivo classes.txt o data.yaml que liste los nombres de las clases en el orden en que se les asignarán los IDs numéricos.

Estructura de Carpetas para Entrenamiento (Típica en YOLOv5/v8):

```
tu_proyecto_billar/
├── data/
│   ├── images/
│   │   ├── train/
│   │   ├── val/
│   │   └── test/
│   └── labels/
│       ├── train/
│       ├── val/
│       └── test/
└── custom_data.yaml
```

Proceso de Conversión: Se ha proporcionado un script de Python (prepare_yolo_dataset.py) para automatizar esta conversión. El script realizará las siguientes acciones:

1. **Organizar imágenes:** Las imágenes ya están divididas en carpetas train, test y valid dentro de data/BallsDataset/. El script copiará estas imágenes a la nueva estructura data/images/.
2. **Dividir datos:** No se requiere una división explícita ya que el dataset original ya está segmentado en train, test y valid (que se usará como val).

- **Convertir CSV a formato YOLO:**Leerá el _annotations.csv de cada subdirectorio (train, test, valid).
- Iterará sobre cada anotación.
- Calculará las coordenadas normalizadas (center_x, center_y, width, height).
- Mapeará el nombre de la clase a un class_id numérico.
- Escribirá esta información en el archivo .txt correspondiente en las carpetas data/labels/train/, data/labels/val/, data/labels/test/.
- "Este script es un paso crucial y el más 'tedioso' en la preparación de datos para la detección de objetos."

Archivo custom_data.yaml: El script también generará automáticamente el archivo custom_data.yaml, que es esencial para la configuración de YOLO. Este archivo contendrá la ruta a la carpeta de datos, las rutas relativas a las imágenes de entrenamiento, validación y prueba, el número total de clases (nc), y los nombres de las clases (names).

FASE 2

1. Introducción y Contexto del Proyecto

El objetivo principal de este proyecto es desarrollar un modelo de detección de objetos, utilizando la arquitectura YOLOv8n, capaz de identificar y clasificar las 15 diferentes bolas de billar en imágenes. El entrenamiento se está llevando a cabo en un entorno local, inicialmente utilizando CPU, con planes para optimizar el rendimiento mediante la habilitación de la GPU (AMD ROCm). El proyecto utiliza la librería ultralytics para el entrenamiento y la inferencia del modelo.

2. Configuración y Parámetros del Modelo YOLOv8n

El modelo base elegido es **YOLOv8n** (nano), la versión más ligera y rápida de la familia YOLOv8, adecuada para empezar. Tiene un total de **3,013,773 parámetros** y **129 capas**.

Los parámetros iniciales y ajustados para el entrenamiento son:

- **Modelo Base:** yolov8n.pt (un modelo pre-entrenado, que se adapta a las 15 clases de bolas de billar, en lugar de las 80 originales de COCO).
- **Dataset:**Imágenes de entrenamiento: **5523**
- Imágenes de validación: **235**
- Número de clases detectadas: **15**
- **Épocas (epochs):** Inicialmente 50, ajustado en sesiones subsiguientes.
- **Tamaño de imagen (imgsz):** Originalmente 640×640, pero se ajustó a **1080×1080** para el dataset de alta resolución, y automáticamente a **1088×1088** (para ser múltiplo del stride máximo de la red, 32). "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." Sin embargo, "Mayor imgsz = Mayor consumo de VRAM y entrenamiento más lento."
- **Tamaño del Batch (batch size):** Inicialmente 16, luego ajustado a 8 y posteriormente aumentado progresivamente a 12 y 14 en las sesiones de entrenamiento con CPU. En GPU, se sugiere probar tamaños de lote mayores (ej. 16, 32 o más).
- **Optimizador:** Seleccionado automáticamente como **AdamW** con una tasa de aprendizaje (lr) inicial de 0.000526.
- **Aumentación de datos:** Se aplican técnicas de aumentación de datos (como fliplr, hsv_h, hsv_s, hsv_v, translate, scale) para mejorar la robustez del modelo.

3. Proceso de Entrenamiento y Reanudación

El entrenamiento de modelos de detección de objetos como YOLOv8n puede ser intensivo en recursos.

- **Entrenamiento en CPU:** Se observó una duración de aproximadamente **35 minutos por época** cuando se utilizaba únicamente la CPU. Esto implica que 50 épocas equivalen a casi 29 horas de entrenamiento. La lentitud en CPU dificulta la experimentación con hiperparámetros.
- **Reanudación del Entrenamiento:** El modelo Ultralytics YOLOv8 guarda periódicamente **checkpoints** (last.pt y best.pt) en el directorio de la ejecución (tu_proyecto_billar/runs/billar_balls_detection_v1/weights/). Esto permite detener y reanudar el entrenamiento desde el último punto guardado, modificando el script para cargar last.pt en lugar del modelo pre-entrenado yolov8n.pt. La cantidad total de épocas a entrenar se calculará a partir de la época del checkpoint cargado y el nuevo epochs especificado en el script.

4. Habilitación de GPU (AMD ROCm) para Aceleración

Para acelerar el entrenamiento, se ha priorizado la configuración de la GPU AMD (RX 6800 XT) con ROCm.

- **Requisitos:**Instalación de **drivers AMD** y la plataforma **ROCm**. Esto incluye la instalación de cabeceras del kernel y paquetes de desarrollo, y la ejecución del script amdgpu-install con el parámetro --usecase=graphics,rocm.
- Añadir el usuario a los grupos render y video para permisos de GPU.
- Instalación de **TensorFlow con soporte para ROCm**. Esto implica identificar la versión de Python y la versión del Toolkit ROCm (apt show rocm-hip-sdk | grep Version o cat /opt/rocm/.info/version) para descargar el archivo .whl compatible desde el repositorio de AMD (https://repo.radeon.com/rocm/manylinux/).
- **PyTorch con soporte para ROCm:** Similar a TensorFlow, PyTorch también tiene soporte para ROCm en Linux. Se requiere desinstalar versiones de PyTorch orientadas a CUDA y luego instalar los wheel de PyTorch, torchvision y torchaudio desde el repositorio de AMD ROCm (ej., https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4.1/).

- **Verificación:** Se utiliza un script de Python para verificar que TensorFlow o PyTorch detecten y utilicen la GPU. Se espera que len(tf.config.list_physical_devices('GPU')) o torch.cuda.is_available() devuelvan True y muestren la GPU AMD. Los logs iniciales de entrenamiento (torch-2.7.0+cu126 CPU y Can't initialize NVML) indican que la GPU no estaba siendo utilizada, lo cual se solucionaría con la configuración de ROCm.

5. Análisis del Rendimiento del Entrenamiento

Se realizaron cuatro sesiones de entrenamiento, reanudando desde el último checkpoint, acumulando un total de 23 épocas en CPU y luego probando con una GPU:

Entrenamiento (Épocas Totales)mAP50mAP50-95Box(P)Rbox_loss (train)cls_loss (train)dfl_loss (train)**1º (6 épocas - CPU)**0.7340.4420.7160.6971.0081.1441.112**2º (10 épocas - CPU)**0.7610.4640.7210.7290.90220.85641.048**3º (18 épocas - CPU)**0.7660.4710.7460.7220.7540.55660.9692**4º (23 épocas - CPU)**0.7670.4780.7480.7370.76220.58490.9759**Validación Final (best.pt)**0.7670.4770.7480.738N/AN/AN/A**Observaciones Clave:**

- **Mejora Continua:** El modelo mostró una mejora constante en las métricas de precisión (mAP50, mAP50-95, Box(P), R) y una disminución en las pérdidas (box_loss, cls_loss, dfl_loss) a lo largo de las épocas, lo que indica un aprendizaje efectivo.
- **Estancamiento en CPU:** En las últimas épocas del entrenamiento en CPU, la mejora en mAP50-95 se volvió más lenta, sugiriendo un posible estancamiento en un "mínimo local" debido a la lentitud del proceso en CPU.
- **Problemas de Clasificación Iniciales:** Tras 6 épocas, se observó que el modelo detectaba correctamente las bounding boxes pero tenía errores de clasificación (ej., "dred_7" en lugar de "bola 4") y no detectaba todas las bolas. Esto es "completamente normal y esperado en esta etapa temprana del entrenamiento" y se atribuye a pocas épocas y similitud visual entre algunas bolas.
- **Impacto de la GPU:** Tras la activación de la GPU, se observa un cambio significativo en los tiempos por época. Mientras que en CPU el tiempo por época era de aproximadamente 35-37 minutos, con la GPU (Tesla T4 en Colab) el tiempo por época se reduce drásticamente a **4-5 minutos**, con un consumo de GPU_mem de ~3-4GB. "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."

6. Desafíos y Próximos Pasos

- **Optimización del rendimiento en GPU:** Asegurar que PyTorch y Ultralytics estén utilizando correctamente la GPU AMD local para aprovechar la aceleración. Las advertencias "Can't initialize NVML" y "no accelerator is found" deben ser resueltas.
- **Más Épocas de Entrenamiento:** Una vez en GPU, entrenar durante muchas más épocas (ej., 50, 100 o más) para permitir que el modelo refine aún más sus pesos y escape de posibles mínimos locales.
- **Análisis de Clases con Bajo Rendimiento:** La clase purple_12 muestra un rendimiento inferior. Se recomienda investigar si hay suficientes ejemplos de esta bola en el dataset, la calidad de las etiquetas o si es visualmente muy similar a otras bolas.
- **Tamaño del Lote (Batch Size):** Experimentar con tamaños de lote mayores en GPU para lograr una convergencia más estable y rápida, considerando la VRAM disponible.
- **Evaluación del best.pt:** Utilizar siempre el modelo best.pt para la inferencia final, ya que guarda los pesos de la época con el mejor mAP50-95 en el conjunto de validación.
- **Incrementar Conjunto de Validación:** Si es posible, aumentar el tamaño del conjunto de validación (idealmente 10-20% del total de datos) para obtener una estimación más robusta del rendimiento del modelo.

7. Conclusión Preliminar

El progreso en el entrenamiento del modelo YOLOv8n es prometedor. El modelo está aprendiendo eficazmente la tarea de detección de bolas de billar, mostrando una mejora constante en las métricas. La transición exitosa a un entorno con GPU es crucial para acelerar el proceso y explorar configuraciones de entrenamiento más avanzadas, permitiendo al modelo alcanzar un rendimiento superior y superar las limitaciones observadas en el entrenamiento exclusivo con CPU. Las pruebas de detección visuales confirman la mejora en la precisión y recall de las detecciones a medida que aumenta el número de épocas.

FASE 3

1. Modelo y Entorno de Entrenamiento

El sistema de detección utiliza **Ultralytics 8.3.146 (o 8.3.152 en fases posteriores) con Python 3.12.3 y PyTorch 2.6.0 + ROCm 6.4.1**, ejecutándose en una **AMD Radeon RX 6800 XT con 16368MiB de VRAM**. El modelo empleado es **YOLOv11n**, que consta de **100 capas, 2,585,077 (o 2,585,272 en v15 en adelante) parámetros y 6.3 GFLOPs**. La configuración de entrenamiento incluye:

- **Épocas:** Variable, desde 10 hasta 200 en las pruebas de "entrenamiento mejorado".
- **Tamaño de imagen (imgsz):** Originalmente 640×640 píxeles, aunque se observa una advertencia que lo actualiza a 1088×1088 en las pruebas de inferencia (WARNING ⚠ imgsz=[1080] must be multiple of max stride 32, updating to [1088]).
- **Tamaño de lote (batch):** 24 o 32.
- **HSV Augmentation:** Parámetros hsv_h, hsv_s (por defecto 0.7), hsv_v son utilizados para ajustar aleatoriamente el tono, saturación y brillo de las imágenes, respectivamente. Esto es crucial para la robustez del modelo, ya que "el modelo no aprende a detectar 'bolas muy saturadas', sino que aprende a detectar 'bolas' sin importar si su color es muy vivo, está un poco desteñido por un reflejo, o más oscuro por una sombra. El modelo se vuelve robusto."

2. Métricas de Rendimiento (mAP50 y mAP50-95)

Las validaciones del modelo se realizan utilizando métricas estándar de detección de objetos:

- **mAP50:** Mean Average Precision a un umbral de IoU (Intersection over Union) de 0.50.
- **mAP50-95:** Mean Average Precision promediada sobre IoU del 0.50 al 0.95.

A continuación, se presentan los resultados de validación de varias versiones del modelo, destacando la métrica mAP50 global y el rendimiento para clases específicas:

2.1. Validaciones Iniciales (runs/billar_balls_detection_vx)

Versiónall (mAP50)whiteblack_8green_6yellow_1blue_2v160.7340.7680.7760.8550.8060.79v170.7300.768 Validaciones en GPU (runs_gpu/billar_balls_detection_vx)

Estas versiones muestran un rendimiento generalmente superior, lo que sugiere una mejora en el entrenamiento o en la configuración.

Versiónall (mAP50)whiteblack_8green_6yellow_1blue_2v10.7790.7700.7920.8480.8590.776v120.7680.7560.8080.8620.8460.766v1 **(Epoch 66)0.7410.7020.7960.8780.8370.792***Nota:* El mejor modelo (detect_balls_v012/weights/best.pt) se logró en la época 66 con un mAP50 global de 0.741.

3. Análisis del Modelo HSV para Detección de Color

La fuente enfatiza la importancia del modelo de color HSV (Hue, Saturation, Value) para una detección de color robusta, especialmente en condiciones de iluminación variables:

- **H (Hue/Tono o Matiz):** "Es el color 'puro' (rojo, verde, amarillo, etc.). Se representa como un círculo de 0 a 360 grados, aunque en OpenCV se mapea a un rango de 0 a 179 para que quepa en un byte. Este es el componente clave para identificar qué color es."
- **S (Saturation/Saturación):** "Es la 'pureza' o 'intensidad' del color. Un valor bajo de saturación significa que el color es más grisáceo y 'desteñido'. Un valor alto significa que es un color muy vivo y puro. El rango es de 0 a 255."
- **V (Value/Valor o Brillo):** "Es el brillo del color. Un valor bajo significa que el color es más oscuro (cercano al negro). Un valor alto significa que es más brillante (cercano al blanco). El rango es de 0 a 255."

La "gran ventaja" del modelo HSV es que "el Tono (H) de una bola de billar será relativamente constante sin importar si está en una zona con mucha luz o en una sombra. Lo que cambiará será su Valor (V) y quizás su Saturación (S). Al aislar el Tono, podemos crear un detector de color muy fiable."

4. Pruebas de Inferencia y Análisis de Errores

El "modelo campeón" (detect_balls/runs/detect_balls_v012/weights/best.pt de la época 66) fue utilizado para la inferencia en nuevas imágenes. Los resultados de la inferencia detallan las detecciones con su clase, confianza y coordenadas.

4.1. Observaciones Clave de las Pruebas:

1. **Detecciones de baja confianza:** Se observan detecciones con "confianzas de 0.48, 0.44 e incluso 0.39". Esto indica que el modelo no está completamente seguro de su predicción para algunas instancias.
2. **Detecciones múltiples/duplicadas:** "El modelo detecta dos veces la bola white, dos veces la red_3 y dos veces la purple_4." Esto ocurre cuando el modelo genera múltiples "bounding boxes" para el mismo objeto.

4.2. Análisis Detallado y Acciones Recomendadas:

Estos dos problemas están interrelacionados y pueden abordarse mediante la optimización de hiperparámetros.

- **Umbral de Confianza (Confidence Threshold):** Se recomienda ajustar este parámetro. "Si tu sistema muestra un alto número de 'falsos positivos' (detecta cosas que no son bolas) o 'duplicados' (detecta la misma bola varias veces con cajas ligeramente diferentes), la primera y más sencilla medida es **subir el umbral de confianza**." Un umbral por defecto suele ser 0.25, pero subirlo a 0.50 o incluso 0.70 podría filtrar las detecciones dudosas.
- **Non-Maximum Suppression (NMS):** Aunque YOLO aplica NMS automáticamente, se explica su función:

1. Identifica cajas delimitadoras con alto solapamiento.
2. Mantiene solo la caja con la puntuación de confianza más alta.
3. Suprime las demás cajas solapadas. La fuente sugiere que el ajuste del umbral de confianza probablemente resolverá las detecciones duplicadas, ya que las detecciones de baja confianza que sobreviven a NMS se eliminarán con un umbral más estricto.

5. Configuración del Entrenamiento y Hiperparámetros

La sección sobre "La Configuración del Entrenamiento (Hiperparámetros)" aclara otros ajustes importantes:

- epochs: Número de ciclos completos de entrenamiento sobre el dataset.
- batch: Tamaño del lote de imágenes procesadas simultáneamente.
- imgsz: Tamaño de la imagen a la que se redimensionan las entradas del modelo.
- hsv_s: "Ganancia de aumentación de Saturación (Saturation)", por defecto 0.7, lo que indica que la saturación de las imágenes ya se altera aleatoriamente para mejorar la robustez.

FASE 4

1. Introducción: Transición del Modelo y Metodología

El proyecto ha avanzado a una fase crítica: **la mejora del rendimiento del modelo YOLO mediante la adopción de una arquitectura más potente**. Inicialmente, se trabajó con un modelo "nano" (YOLOv8n), y la estrategia actual implica la migración a un modelo "medium" (YOLOv8m) para abordar limitaciones previas, como el bajo *Recall* (bolas no detectadas) y la baja *Precisión* (detecciones incorrectas o confusiones).

La transición se realizó de manera metódica, manteniendo los hiperparámetros del entrenamiento anterior (augment=True, mixup=0.1, hsv_s=0.9, patience=20) para asegurar una comparación justa. El único cambio significativo fue la arquitectura del modelo, pasando de yolov8n.pt a yolov8m.pt.

2. Justificación del Cambio a un Modelo Más Grande (YOLOv8m)

La decisión de migrar a un modelo más grande se basa en la expectativa de una mayor capacidad de aprendizaje. Se utiliza la analogía de "estudiantes con distinta capacidad":

- **YOLOv8n (Nano):** "Es un estudiante brillante y súper rápido. Aprende los conceptos generales a una velocidad increíble. Sin embargo, su 'memoria' y 'capacidad de análisis' (el número de parámetros) son limitadas. Puede que le cueste diferenciar matices muy sutiles o recordar todos los casos excepcionales."

- **YOLOv8m (Medium):** "Son estudiantes con más 'capacidad cerebral'. Tardan más en estudiar (entrenar) y en responder a una pregunta (inferencia), pero pueden comprender y retener detalles mucho más complejos y sutiles."

Ventajas esperadas de un modelo más grande:

- **Mayor Precisión Potencial:** El objetivo principal es que el mAP (mean Average Precision) sea superior.
- **Mejor Recall:** El modelo debería detectar más objetos en condiciones difíciles (reflejos, ocluidas). Por ejemplo, "Una bola que el modelo nano detectaba con conf=0.4 (y que tú filtrabas), el modelo small podría detectarla con conf=0.7 (y ahora sí la verías)."
- **Mayor Confianza:** Las detecciones correctas tendrán una confianza más alta, haciendo que el umbral de confianza sea más efectivo.
- **Capacidad de diferenciar matices sutiles:** Crucial para problemas como "confundir, por ejemplo, la bola azul con la morada".

Desventajas a considerar:

- **Más Costoso de Entrenar:** Requiere más tiempo por época y mayor VRAM de la GPU. Sin embargo, se confirma que la RX 6800 XT de 16GB del usuario es suficiente para YOLOv8s o YOLOv8m.
- **Inferencia más Lenta:** Si el objetivo es una aplicación en tiempo real, esto es clave. Un modelo nano puede tardar 15ms por imagen, mientras que un small podría tardar 25ms. El modelo medium utilizado (yolo11m.pt) tiene un **costo computacional de 68.3 GFLOPs y 20.1 millones de parámetros**, casi 8 veces más parámetros y 10 veces más capacidad de cálculo que el modelo nano (2.6 millones de parámetros, 6.5 GFLOPs).

3. Resultados del Entrenamiento del Modelo YOLO11m

El entrenamiento del modelo YOLO11m fue un éxito rotundo, confirmando las hipótesis iniciales sobre la mejora de rendimiento:

A. Métricas de Pérdida (Loss)

- **box_loss (error de localización):** Disminuyó de 1.313 a 1.116 en la primera época.
- **cls_loss (error de clasificación):** Cayó drásticamente de 2.109 a 1.146 en la primera época.
- **dfl_loss (pérdida técnica):** Bajó de 1.317 a 1.131 en la primera época.

Conclusión: "Todas las métricas de error están disminuyendo, especialmente la de clasificación. Esto significa que el modelo está aprendiendo muy rápido a saber 'dónde' está una bola y 'qué' bola es."

B. Métricas de Rendimiento (mAP)

- **mAP50 (métrica principal):** Subió de 0.728 a 0.751 en la primera época.
- **mAP50-95 (métrica más estricta):** Subió de 0.418 a 0.449 en la primera época.

Conclusión: "Un incremento de más de 2 puntos en el mAP50 en una sola época es un salto de calidad enorme. Empezar ya con un mAP tan alto es una señal fantástica."

C. Comparativa con el Modelo Nano Anterior

- El modelo YOLO11m alcanzó un mAP50 máximo de **0.799 en la época 13**.
- El modelo YOLO11m alcanzó un mAP50-95 máximo de **0.496 en la época 26**.
- El modelo nano anterior alcanzó un mAP50 máximo de aproximadamente 0.75.

Conclusión: El modelo medium es "objetivamente superior en precisión (mAP) al modelo nano. El salto de ≈0.70 a ≈0.78 es muy significativo."

D. Comportamiento del Entrenamiento (Paciencia y Sobreajuste)

- Las pérdidas de entrenamiento (train/box_loss, train/cls_loss) muestran una "trayectoria descendente perfecta", mientras que las pérdidas de validación (val/box_loss, val/cls_loss) se aplanaron y tendieron a subir ligeramente después de una bajada inicial.
- "La brecha que se abre entre ambas líneas es la firma visual del sobreajuste."

- El entrenamiento se detuvo automáticamente después de 46 épocas, ya que el modelo no mostró mejora en el mAP50-95 durante 20 épocas consecutivas (desde la época 26 hasta la 46), activando el parámetro patience=20. Esto demuestra un "mecanismo de parada temprana para obtener el mejor modelo posible sin malgastar recursos."

4. Análisis Detallado por Clase y Hipótesis de Confusión

Un análisis detallado del mAP50 por clase revela las fortalezas y debilidades del modelo:

BolamAP50 (Rendimiento)Notagreen_6 (verde lisa)0.877🏆 Campeonagreen_14 (verde rayada)0.842Excelenteblack_8 (negra)0.831Excelente.....purple_4 (morada lisa)0.736Regularorange_5 (naranja lisa)0.726Regularpurple_12 (morada rayada)0.618Área de Mejora**Hipótesis Clave:** "Mi hipótesis es que la principal fuente de error del modelo ahora mismo es la confusión entre las versiones lisas y rayadas de un mismo color." Esto se evidencia en el bajo rendimiento de purple_12, orange_5 y purple_4. Se sugiere que el modelo confunde purple_4 (lisa) con purple_12 (rayada) y viceversa, lo que degrada el rendimiento de ambas clases.

5. Evaluación de Robustez: El "Domain Gap"

La comparación entre last.pt (el modelo de la última época entrenada) y best.pt (el modelo campeón guardado en la época 26) en imágenes de prueba, incluyendo "imágenes reales" (no vistas durante el entrenamiento), reveló un problema crítico: el "Domain Gap" o brecha de generalización.

A. Rendimiento de last.pt

- **Imágenes del Dataset (Imágenes 6, 8, 9):** "el rendimiento es espectacular. El modelo detecta una cantidad masiva de bolas y lo hace con una confianza altísima (muchas por encima de 0.90)." Esto se debe a que provienen del mismo "universo" que los datos de entrenamiento.
- **Imágenes "Reales" (Imágenes 1-5):** "el rendimiento es muy pobre. El modelo detecta poquísimas bolas, e incluso falla por completo en la Imagen 5." Esto ocurre porque estas imágenes provienen de un "dominio" diferente (iluminación, tapete, ángulo de cámara, resolución/compresión distintos), y el modelo sobreajustado last.pt no generaliza bien a este nuevo entorno.

B. Rendimiento de best.pt

La evaluación posterior con best.pt confirmó su superioridad en generalización:

- **test_pool_table_1.png:** De 0 a 3 bolas detectadas ("✅ Mejora Sustancial").
- **test_pool_table_2.png:** De 0 a 4 bolas detectadas ("✅ Mejora Sustancial").
- **test_pool_table_3.png:** De 2 a 3 bolas detectadas ("✅ Mejora Ligera").
- **test_pool_table_4.jpg:** De 7 a 7 bolas detectadas ("↔ Estable").
- **test_pool_table_5.png:** De 0 a 0 bolas detectadas ("↔ Fallo Persistente").

Conclusión: "El modelo best.pt (de la época 26), al estar menos sobreajustado, tiene una capacidad para generalizar a imágenes nuevas muy superior. Ha sido capaz de encontrar muchas más bolas que el modelo last.pt."

6. Próximos Pasos Estratégicos

El proyecto ha completado con éxito la fase de optimización del modelo, pero el siguiente gran salto de calidad debe centrarse en los datos para mejorar la robustez y cerrar el "Domain Gap".

A. Acciones Inmediatas y de Análisis:

- **Usar el Nuevo Campeón:** Para todas las pruebas y aplicaciones futuras, se debe utilizar detect_balls/runs/detect_balls_v12/weights/best.pt.
- **Confirmar la Hipótesis de Confusión:** Realizar un análisis visual detallado de los errores en las imágenes de prueba con best.pt. Buscar activamente casos de confusión entre bolas lisas y rayadas para confirmar la hipótesis.
- **Enfocarse en la Imagen 5:** La Imagen 5 es un "punto ciego" y debe ser la prioridad para entender qué la hace tan diferente (iluminación, reflejos, oscuridad).

B. Campaña de Datos 2.0 (Mejora Dirigida):

- **Enriquecer el Dataset:** Añadir imágenes al conjunto de entrenamiento que se asemejen a las "imágenes reales" donde el modelo falla.

- **Datos Específicos para Confusión:** Incluir más imágenes de entrenamiento que muestren claramente las bolas lisas y rayadas del mismo color, idealmente juntas, en diferentes ángulos y condiciones de luz. El objetivo es que el modelo aprenda a diferenciar la franja blanca.

C. Re-entrenamiento (Fine-tuning):

- Una vez que el dataset esté enriquecido, se puede volver a entrenar el modelo. No es necesario empezar de cero; se puede usar el best.pt actual como punto de partida para que el modelo se ajuste con los nuevos datos.

7. Conclusión General

El proyecto ha demostrado un "ciclo de investigación y desarrollo de IA de principio a fin", desde la identificación del problema y la formulación de una hipótesis, hasta la experimentación metódica y el análisis de resultados. Se ha confirmado que el modelo medium es superior en precisión al nano, aunque a costa de un mayor costo computacional. Sin embargo, se ha identificado que el **límite principal para una precisión aún mayor ahora reside en la calidad y variedad del dataset**, especialmente para resolver casos de confusión entre bolas lisas y rayadas y para mejorar la generalización a "imágenes reales". La habilidad para identificar y solucionar un "Domain Gap" es una lección fundamental en el Machine Learning práctico.

FASE 5

1. Metodología de Análisis Avanzado de Errores: Diferenciando "Ceguera" de "Inseguridad" (Paso 41)

El análisis crítico del modelo se basó en una técnica avanzada: la reducción temporal del umbral de confianza (conf=0.25) durante la fase de análisis. Esto permitió una diferenciación crucial entre dos tipos de "Falsos Negativos", cada uno con una solución distinta:

- **Tipo 1: El modelo está "Ciego" (Falsos Negativos Absolutos)**
- **Descripción:** Bolas que el modelo no detecta en absoluto, incluso con un umbral de confianza muy bajo (0.25). No propone ninguna caja delimitadora para ellas.
- **Diagnóstico:** "El modelo tiene un problema fundamental con estos casos. Las características de esa bola en esa situación (iluminación, oclusión, ángulo) son tan extrañas para él que ni siquiera las reconoce como 'posiblemente una bola'."
- **Solución Requerida:** Introducir una mayor diversidad en el dataset, enfocándose en "añadir imágenes que representen estas situaciones tan difíciles que actualmente lo dejan 'ciego'."
- **Tipo 2: El modelo está "Inseguro" (Falsos Negativos por Umbral)**
- **Descripción:** Bolas que no aparecían con el umbral estándar (conf=0.5), pero sí lo hacen al bajarlo a conf=0.25, aunque con una confianza baja (ej. 0.35).
- **Diagnóstico:** "¡Estas son las mejores noticias que podemos tener! Significa que el modelo sí está viendo la bola, está en el camino correcto, pero simplemente no está lo suficientemente 'seguro' de su predicción como para superar un umbral de confianza estándar."
- **Solución Requerida:** Estos son los "frutos maduros" o "low-hanging fruit". Se corrigen más fácilmente: "añadir más ejemplos de entrenamiento de esa bola en esa misma situación es muy probable que le dé el 'empujón' necesario para que en el siguiente entrenamiento su confianza suba a 0.6 o 0.7."

Este enfoque permite transformar un simple "fallo" en "información mucho más rica y accionable", posibilitando un diagnóstico preciso y la aplicación del "tratamiento correcto y más eficiente" para mejorar el modelo.

2. Resultados de las Pruebas con conf=0.25 (Paso 42)

Se realizaron pruebas con el modelo best.pt sobre un conjunto de cinco imágenes de prueba "reales", aplicando un umbral de confianza de 0.25. A continuación, se presenta un resumen de las detecciones para cada imagen:

- **Imagen 1 (real):** Se detectaron 5 bolas: 2 red_3 (0.77, 0.70), 1 green_6 (0.60), 1 purple_4 (0.49), 1 white (0.35).
- **Imagen 2 (real):** Se detectaron 5 bolas: 2 red_3 (0.68, 0.49), 1 green_6 (0.33), 1 white (0.28), 1 black_8 (0.26).
- **Imagen 3 (real):** Se detectaron 3 bolas: 1 purple_4 (0.81), 1 black_8 (0.81), 1 yellow_9 (0.42).

- **Imagen 4 (real):** Se detectaron 11 bolas: 1 white (0.82), 1 yellow_9 (0.81), 1 black_8 (0.81), 1 red_3 (0.81), 1 blue_2 (0.81), 1 orange_5 (0.78), 1 dred_7 (0.74), 1 purple_4 (0.68), 1 yellow_1 (0.65), 1 green_6 (0.42), 1 yellow_1 (0.37).
- **Imagen 5 (real):** Se detectaron 11 bolas: 1 black_8 (0.82), 1 dred_7 (0.81), 1 yellow_1 (0.81), 1 green_6 (0.77), 1 yellow_9 (0.72), 1 orange_5 (0.71), 1 red_3 (0.71), 1 purple_4 (0.57), 1 yellow_1 (0.42), 1 white (0.40), 1 blue_2 (0.27).

3. Diagnóstico Definitivo: No es "Ceguera", es "Inseguridad" (Paso 43)

El análisis de los resultados con el umbral reducido reveló una conclusión fundamental: "el problema principal de tu modelo en las imágenes 'reales' no es que esté 'ciego' (que no vea las bolas), sino que está extremadamente 'inseguro' en esos entornos."

El caso más ilustrativo es la **Imagen 5**:

- Con conf >= 0.5: 0 detecciones.
- Con conf >= 0.25: ¡De repente detecta 11 bolas!

Esto demuestra que el modelo "SÍ ve las bolas, sabe que están ahí, pero las características de esa imagen (iluminación, ángulo, calidad) le generan tantas dudas que asigna puntuaciones de confianza muy bajas."

Informe de Errores Detallado (Bolas "Inseguras"):

Imagen RealNuevas Detecciones "Inseguras" (conf entre 0.25 y 0.5)Conclusión ClaveImagen 1purple_4 (0.49), white (0.35)El modelo duda con la morada y la blanca.Imagen 2red_3 (0.49), green_6 (0.33), white (0.28), black_8 (0.26)Un gran número de bolas son vistas pero con muy baja confianza.Imagen 3yellow_9 (0.42)Confirma que la amarilla era una detección insegura.Imagen 4green_6 (0.42), yellow_1 (0.37)La verde y una amarilla son dudosas en esta foto de móvil.Imagen 5¡Casi todas! (white (0.40), blue_2 (0.27), etc.)¡Esta imagen es una mina de oro! Contiene todas las características que hacen dudar al modelo. Esto incluye purple_4 (0.57), yellow_1 (0.42) y blue_2 (0.27).4. Análisis Visual y Conclusiones Detalladas (Paso 44)

El análisis visual pormenorizado de las imágenes y los resultados de detección reveló **tres causas raíz específicas** de los errores del modelo:

4.1. La Brecha de Dominio ("Domain Gap"): Las "TV Balls"

- **Observación:** En las Imágenes 1 y 2, se detectan red_3 donde la bola real es pink_4 (0.77, 0.49) y purple_4 donde la bola real es purple_5 (0.49). El analista observa: "Es normal ya que este set de bolas tiene otros colores (son las TV balls)".
- **Análisis:** El modelo no ha sido entrenado con estos colores específicos de bolas (rosa, morado oscuro). Por lo tanto, las clasifica como la clase más similar conocida, lo que "no son fallos de 'lógica' del modelo, sino una clara limitación de su conocimiento (sus datos de entrenamiento)".

4.2. Los Casos Límite: Oclusiones y Troneras

- **Observación:**Imagen 1: Una yellow_1 en la tronera es detectada como white con confianza baja (0.35).
- Imagen 2: Una blue_2 en la tronera no es detectada ("Falso negativo, La bola está dentro de la tronera y casi no se ve").
- **Análisis:** "Esto revela una debilidad clásica. Cuando una bola está parcialmente oculta (oclusión), sus características visuales cambian drásticamente." El modelo, entrenado con "círculos de colores completos", falla al ver una "media luna" o una bola oscurecida por una tronera.

4.3. La Falta de "Confianza" y Falsos Negativos Simples

- **Observación:** Existencia de falsos negativos inexplicables ("No deberia haber fallado") y muchas detecciones correctas con confianza media-baja (ej., green_6 con 0.33, white con 0.40).
- **Análisis:** "Las condiciones de tus imágenes 'reales' (cámara del móvil, iluminación de la sala) son distintas a las del dataset original." El modelo se vuelve "inseguro" y necesita más ejemplos en este "nuevo dominio" para ganar confianza en sus predicciones correctas.

5. Plan de Acción Definitivo: La Estrategia de Datos 3.0

El diagnóstico preciso permite un plan de acción "quirúrgico" para el dataset:

5.1. Decisión Estratégica sobre el "Dominio" del Proyecto

Antes de la recolección de datos, se debe decidir el alcance del proyecto:

- **Opción A (Enfoque Específico):** Si el objetivo es solo bolas de billar estándar, los errores con las "TV balls" son irrelevantes y esas imágenes no se usan para juzgar el rendimiento en ese aspecto.
- **Opción B (Enfoque Robusto):** Si el objetivo es un modelo que funcione con cualquier tipo de set de bolas, "debes ampliar tu dataset para incluir y etiquetar estas nuevas bolas. Tendrías que crear nuevas clases (pink_4, purple_5, etc.) y recolectar imágenes de ellas."

5.2. Campaña de Datos para Casos Límite y Confianza (Recomendación Principal)

Independientemente de la decisión anterior, esta es la "lista de la compra" para la mejora crítica:

- **Imágenes de Oclusión:**
- Recolectar 20-30 fotos nuevas con bolas parcialmente tapadas por otras.
- Recolectar 15-20 fotos con bolas entrando o parcialmente visibles dentro de las troneras.
- **Imágenes para Cerrar el "Domain Gap" de las Fotos Reales:**
- Añadir las 5 imágenes de prueba actuales (o 20-30 fotos nuevas tomadas con móvil en condiciones variadas de luz y ángulos) al conjunto de entrenamiento. "Esta es la forma más directa y efectiva de enseñarle al modelo cómo son las imágenes en el 'mundo real' donde lo vas a usar."
- **Imágenes para Falsos Negativos Simples:**
- Revisar casos de bolas no detectadas sin razón aparente (ej. black_8 en Imagen 1) y asegurar suficientes ejemplos "normales" de esa bola en diferentes posiciones en el dataset.

5.3. El Re-entrenamiento Final (Fine-Tuning)

Una vez enriquecido el dataset, se realizará un re-entrenamiento:

- **Acción:** Usar el best.pt actual como punto de partida. "Así, el modelo no empieza de cero, sino que 're-aprende' y se ajusta para manejar estos nuevos casos difíciles."

Este proceso de depuración y mejora del dataset, basado en un análisis profundo y estructurado, representa un "ciclo de depuración de nivel profesional" que llevará el modelo al siguiente y definitivo salto de calidad.

FASE 6

1. Introducción y Contexto del Proyecto

El objetivo principal de este proyecto es desarrollar un modelo de detección de objetos, utilizando la arquitectura YOLO, capaz de identificar bolas de billar, con un enfoque particular en la adaptación a sets "Black Edition" (ediciones modernas y estilizadas como Aramith Black). La dificultad radica en la inexistencia de un dataset público pre-etiquetado para este tipo de bolas, lo que convierte la creación y refinamiento del dataset en una fase crítica y formativa del ciclo de vida del Machine Learning.

2. Creación del Dataset "Black Edition": Desafíos y Estrategias

2.1. Inexistencia de Dataset Público y Oportunidad

Se ha confirmado la ausencia de un dataset público específicamente etiquetado para bolas de billar "black edition" en plataformas como Roboflow Universe, Kaggle, y Hugging Face. Esto se debe a que "estos sets de bolas son un producto de nicho y relativamente moderno. La mayoría de los datasets de visión por computadora se crean para los objetos más comunes y estandarizados". Esta situación, sin embargo, representa "una oportunidad fantástica" para el proyecto, ya que "Crear tu propio dataset especializado es una de las tareas más importantes y formativas en un proyecto de Machine Learning".

2.2. Plan de Acción para la Recolección de Datos (Paso 45)

Para construir el dataset, se proponen varias estrategias de recolección de datos, priorizando la generación propia de imágenes y videos si se tiene acceso a un set físico. Los consejos clave para la recolección incluyen:

- **Variedad de Ángulos**: Tomar fotos desde diversas perspectivas (arriba, lados).
- **Diferentes Condiciones de Luz**: Incluir imágenes con luz brillante, sombras y reflejos, lo cual es "CRUCIAL para que tu modelo sea robusto".
- **Distintos Estados de la Mesa**: Capturar escenas con las bolas en diferentes configuraciones de juego (triángulo inicial, a mitad de partida, pocas bolas).
- **Calidad**: Usar la mayor resolución posible, aunque se redimensionen posteriormente.
- **Cantidad**: Recolectar un "mínimo de 100-200 imágenes" de sets "black edition".

3. Pre-Etiquetado Asistido por IA (Paso 46)

Para acelerar el proceso de etiquetado manual, se implementa una técnica de "pre-etiquetado" o etiquetado asistido.

3.1. Concepto y Beneficios

La idea central es "usar tu IA Actual como un 'Asistente Junior'". El modelo YOLO pre-entrenado con bolas de colores tradicionales (best.pt) se utiliza para realizar una primera pasada sobre las nuevas imágenes "black edition". Aunque puede cometer errores de "Clasificación Incorrecta" o "Detecciones Fallidas" debido a los colores desconocidos, su "conocimiento de la forma esférica es muy potente", lo que permite detectar muchas bolas correctamente. Este método puede "ahorrarte entre un 60% y un 80% del tiempo de etiquetado".

3.2. Script de Pre-Etiquetado (pre_etiquetado.py)

Se proporciona un script Python que carga el modelo YOLO existente, procesa una carpeta de nuevas imágenes y genera archivos de etiquetas .txt en formato YOLO como "borrador". Un UMBRAL_CONFIANZA configurable permite ajustar la sensibilidad de las detecciones. Los resultados del pre-etiquetado muestran detecciones con IDs de clase del modelo antiguo (ej., "green_6", "yellow_9", "black_8"), que requerirán corrección manual posterior.

4. Etiquetado y Verificación con Herramientas Profesionales

4.1. Instalación y Uso de Label Studio (Paso 47 y 48)

Label Studio se identifica como una "herramienta de etiquetado potentísima y de código abierto" y una "elección excelente y muy profesional" para revisar y corregir las pre-anotaciones.

4.1.1. Problemas Iniciales y Diagnóstico

Los intentos iniciales de importar imágenes y anotaciones YOLO directamente a Label Studio fracasaron debido a:

- **Seguridad del Navegador**: Restricciones que impiden el acceso directo a rutas de archivos locales.
- **Servidor de Archivos de Label Studio**: Necesidad de configuración explícita para que Label Studio actúe como servidor local.
- **URL Interna Específica**: El formato de URL esperado por Label Studio para archivos locales es /data/local-files/?d=<ruta_relativa_al_directorio_configurado>.

4.1.2. Solución Implementada: Flujo de Trabajo Robusto

Se estableció un flujo de trabajo de tres pasos para la importación exitosa:

1. **Configuración Inicial de Label Studio**: Instalación vía pip, inicio genérico, creación de proyecto y configuración de la interfaz de etiquetado con un XML que incluye las clases maestras (ej., RectangleLabels).
2. **Conexión con Archivos de Imagen Locales**: Configurar "Cloud Storage" en Label Studio para "Local Files", especificando la ruta absoluta a la carpeta de imágenes y marcando "Treat every source file as a source file".
3. **Generación e Importación del Archivo de Tareas (generar_json_definitivo.py)**: Este script Python lee las pre-anotaciones .txt y crea un archivo tasks.json que utiliza el formato de URL interna de Label Studio para las imágenes, permitiendo la carga correcta de imágenes y pre-anotaciones para su revisión.

4.2. Conversión de Label Studio a YOLO (Paso 49)

Una vez completado el etiquetado y las correcciones en Label Studio, el dataset debe convertirse de su formato JSON al formato .txt de YOLO.

4.2.1. Script Conversor (convertir_ls_a_yolo.py)

Este script lee el archivo ls-export.json generado por Label Studio y crea archivos .txt para cada imagen con las coordenadas y los IDs de clase en el formato YOLO. Un aspecto "¡¡¡CRÍTICO!!!" es que la CLASES_MAESTRA definida en el script debe ser "IDÉNTICA A LA DEL SCRIPT ANTERIOR" y coincidir "con el orden de las clases que definiste en la interfaz de Label Studio" para asegurar una asignación de ID numérica correcta.

4.3. Verificación de Etiquetas (verificar_etiquetas_yolo.py y procesador_final.py) (Paso 50)

La verificación visual de las etiquetas generadas es un "paso correcto y profesional" para asegurar la calidad de los datos.

4.3.1. Corrección de Desajustes de IDs

Inicialmente, se detectaron errores de índice donde el sistema "confund[ía] 'be_white' con 'be_yellow_9' de manera consistente", lo que indicaba que "el orden de las clases que el script de conversión utiliza para asignar un ID... es diferente del orden que el script de verificación utiliza para leer ese ID".

4.3.2. Solución "Todo en Uno" (procesador_final.py)

Para eliminar cualquier desajuste, se propone un script unificado. Este script realiza tanto la conversión de Label Studio a YOLO como la verificación visual, utilizando "una única lista de clases maestra" para ambos procesos. Esto garantiza que "la correspondencia tiene que ser perfecta".

5. Unificación y Preparación del Dataset Final (Paso 51)

Para el entrenamiento del "Supermodelo" final, es fundamental unificar el dataset original con el nuevo dataset "black edition" y mantener una separación adecuada en conjuntos de train, valid y test.

5.1. Importancia del Conjunto de Test

El conjunto de test es el "examen final y sorpresa" y es "un conjunto de datos que el modelo nunca, jamás, ha visto durante el entrenamiento o la validación". Su uso, una única vez al final, proporciona "una medida honesta y sin sesgos de cómo se comportará el modelo en el mundo real con datos completamente nuevos".

5.2. Plan de Acción Corregido y Completo

1. **Estructura de Carpetas**: Se crea una estructura de directorios completa para dataset_unificado, incluyendo subcarpetas images/train, images/valid, images/test y labels/train, labels/valid, labels/test.
2. **Copia del Dataset Original**: Las imágenes y etiquetas de los conjuntos train, valid y test del dataset original se copian a sus respectivas ubicaciones en dataset_unificado.
3. **División y Copia del Nuevo Dataset (unificar_y_dividir_completo.py)**: Un script Python divide el dataset "black edition" (imágenes y etiquetas) en proporciones específicas (ej., 72% train, 18% valid, 10% test) y copia los archivos a las carpetas correspondientes en dataset_unificado.
4. **supermodelo_data.yaml Completo**: El archivo de configuración para YOLO se actualiza para incluir la ruta al conjunto de test y la lista names completa con todas las clases.
5. **Entrenamiento**: El proceso de entrenamiento con YOLO no cambia, utilizando los conjuntos train y valid.
6. **Evaluación Final con Conjunto de Test**: Una vez finalizado el entrenamiento, el best.pt del modelo se evalúa en el conjunto de test para obtener métricas de rendimiento imparciales (mAP, Precision, Recall).

6. Conclusión de la Fase

Esta fase ha abarcado la creación, curación y preparación exhaustiva de un dataset de alta calidad, culminando en un conjunto de datos unificado y verificado, listo para el entrenamiento del "Supermodelo" final. La iteración a través de los desafíos del etiquetado y la implementación de soluciones robustas demuestra una comprensión profunda del flujo de trabajo de IA. "Has cerrado el ciclo completo de creación y refinamiento de un dataset".

FASE 7

1. Problema Identificado: Sobreajuste (Overfitting) en la Clasificación de "white ball"

El problema central que se busca abordar es el sobreajuste de un modelo de clasificación, específicamente en la distinción entre una "white ball" clásica y una "black edition" (be_).

- **Descripción del Problema**:
- Un "Dataset Clásico" con miles de imágenes ha permitido al modelo aprender un "concepto muy amplio y general de lo que es una white ball."
- Por contraste, el "Dataset Black Edition", con solo 60 imágenes, ha provocado que el modelo "no ha podido generalizar. En su lugar, ha memorizado las características específicas y únicas de esas 60 imágenes (un tipo de reflejo particular, el logo exacto, el acabado del material, etc.)."
- **Consecuencia**: "Cuando el modelo analiza una bola clásica, si por casualidad una sombra o un reflejo se parece a una de esas características 'memorizadas' del set be_, el modelo prioriza esa característica única y clasifica la bola incorrectamente como be_..., a pesar de haber visto miles de ejemplos de la versión clásica." Esto es una "definición de libro del sobreajuste (overfitting)."

2. Estrategia de Solución: Aumento de Datos y Entrenamiento por Fases

Para mitigar el sobreajuste y mejorar la capacidad de generalización del modelo, se propone un plan de acción que incluye el aumento de datos y un entrenamiento por fases.

2.1. Aumento de Datos con Albumentations

El plan de acción inicia con la creación de un "dataset 'black edition' rico y variado a partir de tus 60 imágenes" utilizando la librería Albumentations.

- **Ventaja clave de Albumentations**: "Su gran ventaja es que cuando rota, cambia el brillo o deforma una imagen, también calcula y ajusta automáticamente las coordenadas de las cajas delimitadoras (bounding boxes) para que sigan encajando perfectamente con los objetos."
- **Implementación**: Se requiere instalar Albumentations y activar un entorno virtual específico (source /home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/.venv/bin/activate).
- **Augmentaciones aplicadas (ejemplos del log)**: 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))

2.2. Entrenamiento por Fases para el Modelo YOLO11m

Se implementa una estrategia de entrenamiento en dos fases para optimizar el rendimiento del modelo, identificado como YOLO11m. Este modelo tiene un "summary: 231 layers, 20,077,680 parameters, 20,077,664 gradients, 68.3 GFLOPs".

- **Advertencia de Ultralytics**: Inicialmente, el framework de Ultralytics ignoró los parámetros de tasa de aprendizaje (lr0) debido a que el optimizer estaba en modo auto. El mensaje "optimizer: 'optimizer=auto' found, ignoring 'lr0=0.001' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically..." lo confirma. Esto llevó a un estancamiento en el rendimiento del modelo anterior.
- **Plan de Ejecución del Entrenamiento por Fases**: El script de Python se divide en dos partes:
- **Fase 1: Entrenamiento de la "cabeza" del modelo (50 épocas)**
- **Estrategia**: Congelar la mayoría de las capas del modelo pre-entrenado y entrenar solo las capas de salida (la "cabeza") para que el modelo se adapte a las nuevas clases sin modificar el conocimiento general.
- **Congelamiento de Capas**: Numerosas capas del modelo (ej., model.0.conv.weight, model.1.bn.weight, model.2.cv1.conv.weight, etc.) fueron congeladas, transfiriendo 643 de 649 ítems de los pesos pre-entrenados.
- **Resultados de la Fase 1 (Supermodelo_Fase1_Head2)**:
- **Duración**: 50 épocas completadas en aproximadamente 1.94 horas (6991.55 segundos).
- **Métricas Finales**:
- all (general): Box(P=0.88164, R=0.8308, mAP50=0.87576, mAP50-95=0.59076)

- **Rendimiento en clases be_ (ejemplos)**:
- be_black_8: P=0.943, R=0.944, mAP50=0.964, mAP50-95=0.672
- be_blue_10: P=0.973, R=1, mAP50=0.995, mAP50-95=0.657
- be_green_6: P=0.977, R=0.921, mAP50=0.959, mAP50-95=0.716
- Se observa que las clases be_ tienen un mAP50 significativamente más alto que las clases clásicas en esta fase.
- **Uso de GPU**: Se mantuvo alrededor de 6.41G - 6.46G.
- **Fase 2: Ajuste Fino de todo el modelo (150 épocas)**
- **Estrategia**: Descongelar todas las capas del modelo y continuar el entrenamiento con una tasa de aprendizaje muy baja (lr0=0.0005) para realizar "ajustes mínimos y de alta precisión sin olvidar lo que ya sabían". Esto fuerza el uso del optimizador AdamW y la tasa de aprendizaje correcta.
- **Resultados de la Fase 2 (Supermodelo_FineTuned_LR_Bajo_v1 / Supermodelo_Fase2_Final)**:
- **Duración**: 150 épocas completadas en aproximadamente 10.805 horas.
- **Métricas Finales (Supermodelo_FineTuned_LR_Bajo_v1)**:
- all (general): Box(P=0.885, R=0.833, mAP50=0.873, mAP50-95=0.69)
- **Rendimiento en clases be_ (ejemplos)**:
- be_black_8: P=0.984, R=0.979, mAP50=0.994, mAP50-95=0.877
- be_blue_10: P=0.984, R=1, mAP50=0.995, mAP50-95=0.908
- be_green_6: P=0.98, R=0.985, mAP50=0.994, mAP50-95=0.879
- **Métricas Finales (Supermodelo_Fase2_Final)**:
- all (general): Box(P=0.888, R=0.824, mAP50=0.869, mAP50-95=0.689)
- **Rendimiento en clases be_ (ejemplos)**:
- be_black_8: P=0.979, R=0.986, mAP50=0.994, mAP50-95=0.864
- be_blue_10: P=0.986, R=1, mAP50=0.995, mAP50-95=0.905
- be_green_6: P=0.993, R=0.982, mAP50=0.995, mAP50-95=0.887
- **Uso de GPU**: Se mantuvo alrededor de 11.7G - 12.3G.

3. Resultados y Observaciones Clave

- **Mejora en el mAP50-95 para clases be_**: La estrategia de entrenamiento por fases, combinada con el aumento de datos, ha resultado en una mejora sustancial en el rendimiento de las clases "black edition". El mAP50-95 para estas clases es consistentemente alto (ej., be_black_8 de 0.672 en Fase 1 a 0.877/0.864 en Fase 2; be_blue_10 de 0.657 a 0.908/0.905).
- **Rendimiento general (mAP50-95)**: Aunque el mAP50-95 general (all) muestra una ligera fluctuación entre las versiones finales (0.694 en la primera corrida completa de 50 épocas, 0.684 después de 50 épocas, 0.690 en la corrida de 150 épocas con LR bajo, y 0.689 en la final), los resultados en las clases problemáticas (be_) son notablemente superiores.
- **Confirmación de sobreajuste previo**: El informe menciona que en un entrenamiento anterior (no detallado en este documento, pero inferido por la sección "Análisis del Log que has Enviado"), a partir de la época ~20, el modelo se "estancó" con val/cls_loss entre 0.76 y 0.86, sin mejora significativa, lo que es la "definición de libro del sobreajuste (overfitting)". Esto valida la necesidad del entrenamiento por fases.
- **Parámetros del modelo YOLO11m**: Se detalla la arquitectura del modelo YOLO11m, incluyendo el número de capas (231), parámetros (20,077,680), y GFLOPs (68.3), y su versión fusionada (125 capas, 20,054,704 parámetros, 67.8 GFLOPs).

4. Conclusión

La implementación de un plan de acción que incluye el aumento de datos con Albumentations y un entrenamiento por fases (primero la "cabeza" con capas congeladas, luego un ajuste fino de todo el modelo con una tasa de aprendizaje baja) ha sido efectiva para abordar el problema de sobreajuste. Los resultados muestran una mejora significativa en la

capacidad del modelo para clasificar correctamente las "bolas black edition", superando el estancamiento observado en intentos anteriores donde los parámetros de optimización no se aplicaron correctamente. Esto sugiere que el enfoque "controlado y profesional" de entrenamiento por fases es crucial para este tipo de escenarios de clasificación con datasets desequilibrados o complejos.

FASE 8

1. Resumen Ejecutivo

El proyecto ha logrado un avance significativo en la detección y clasificación de bolas de billar mediante la implementación de una **arquitectura híbrida**. Esta arquitectura combina un modelo YOLO "semi-específico" para la detección visual inicial con un módulo de post-procesamiento basado en lógica y contexto ("el cerebro del juego") para la desambiguación y refinamiento final de las etiquetas. Este enfoque modular ha demostrado ser "sólido, inteligente y factible", y "superior a la de usar colores completamente genéricos", culminando en un modelo de alta calidad y un sistema robusto.

2. Problema Inicial y Soluciones Propuestas

Inicialmente, el modelo YOLO se enfrentaba a una tarea compleja: clasificar 32 clases muy específicas, incluyendo duplicados visuales entre los sets "Clásico" y "Black Edition" (por ejemplo, 'red_3' y 'be_red_3'). Esto creaba ambigüedad y dificultaba el entrenamiento.

Se exploraron dos arquitecturas principales:

- **Detector de Color Genérico (El "Ojo" de YOLO):** Propuesta inicial para entrenar YOLO a reconocer solo colores base (8-9 clases genéricas como 'white', 'yellow', 'blue'). El módulo de lógica posterior determinaría el set y la especificidad. Aunque "sólida", se consideró una alternativa más refinada.
- **Modelo YOLO "Semi-Específico" con Lógica y Contexto (Arquitectura Híbrida Final):** Esta es la arquitectura adoptada y desarrollada.
- **Etapa 1: Modelo YOLO "Semi-Específico":** Entrenado con un número reducido de **25 "clases compartidas"**. Para las bolas idénticas en ambos sets (ej. 'white', 'red_3'), el modelo las reconoce bajo una única etiqueta genérica. Para las bolas únicas de cada set (ej. 'blue_10' para clásico, 'be_pink_4' para Black Edition), mantienen su especificidad. El objetivo del modelo es "identificar la bola por su apariencia principal".
- **Etapa 2: Módulo de Lógica y Contexto (Post-Procesamiento):** Un script de Python que recibe las detecciones "semi-específicas" de YOLO. Su tarea es "detectar el contexto" del set (Clásico o Black Edition) y luego "desambiguar etiquetas" aplicando reglas para refinar las predicciones. Este módulo añade una capa de "razonamiento" sobre la "percepción" de la IA.

3. Proceso de Implementación y Entrenamiento

El desarrollo de la arquitectura híbrida siguió estos pasos clave:

- **Re-etiquetado del Dataset:** El dataset original de 32 clases fue re-etiquetado a las 25 clases compartidas. Un script (traducir_a_schema_hibrido.py) fue desarrollado para automatizar este proceso, mapeando las clases originales a las nuevas etiquetas híbridas (ej., 'black_8' y 'be_black_8' se unifican en 'black_8').
- **Entrenamiento del Nuevo Modelo YOLO:** Se entrenó un nuevo modelo YOLO (yolov8m) desde cero (sin sesgos previos sobre bolas de billar) utilizando el dataset re-etiquetado. Se utilizó el comando yolo train model=yolov8m.pt data=hibrido_data.yaml epochs=150 patience=30 name=Modelo_Hibrido_v1.
- **Desarrollo del Módulo de Post-Procesamiento:** Se creó una función en Python (post_procesar_detecciones) para aplicar la lógica contextual.

4. Análisis del Dataset (labels_correlogram.jpg)

El análisis del gráfico labels_correlogram.jpg confirmó la "calidad" del dataset:

- **Distribución de Centros (X vs Y):** Los centros de las bolas están bien distribuidos en el gráfico, no concentrados en el centro. Esto es "excelente" e indica que el dataset contiene imágenes con bolas en "todas las partes de la mesa", evitando sesgos de posición.

- **Relación Ancho vs Alto (Width vs Height):** Se observa una concentración diagonal, "perfectamente lógica" para objetos redondos. La dispersión indica una "buena variedad de tamaños de bolas" (cercanas y lejanas), lo cual también es "excelente".
- **Veredicto Final:** El dataset está "bien construido", es "variado, no tiene sesgos de posición o tamaño evidentes y es una base sólida para entrenar un buen modelo".

5. Resultados del Entrenamiento del Modelo Híbrido

El entrenamiento del modelo Modelo_Hibrido_v1 mostró un rendimiento "excelente y muy saludable" a lo largo de las épocas:

- **Métricas mAP:**metrics/mAP50-95(B): Creció de 0.359 (época 1) a un pico de **0.696** (época 148, aunque el informe final sobre el test set arroja un **0.716**). Un valor cercano o superior a 0.7 es un "resultado fantástico" para 25 clases visualmente complejas, indicando una "precisión espacial muy alta" en la localización.
- metrics/mAP50(B): Alcanzó un impresionante **0.867** en el test set, confirmando una alta precisión con un criterio de acierto estándar.
- **Generalización Exitosa:** Las métricas de validación (mAP) mejoraron de forma "clara y consistente", y la val/cls_loss (pérdida en validación) bajó en sintonía con train/cls_loss, lo que es "crucial" e indica que el modelo no está sobreajustando (overfitting) y está "aprendiendo las reglas y características visuales" para aplicar en imágenes nuevas.
- **Paciencia (patience):** El parámetro patience=30 (o 40) aseguró que el entrenamiento se detuviera automáticamente en el "momento óptimo", cuando el modelo alcanzara su "máximo potencial". El best.pt guardado corresponde al modelo de mejor rendimiento.
- **Comportamiento de Meseta (Plateau):** A partir de la época 35-40, las mejoras en mAP50-95(B) se volvieron marginales (ej., de 0.617 a 0.633 en 15 épocas), indicando que el modelo estaba llegando a su "meseta de rendimiento", habiendo extraído casi toda la información útil del dataset.

6. Debilidades Identificadas y Refinamiento del Post-Procesamiento

Aunque el modelo YOLO era "muy bueno", no era "infalible" y presentaba confusiones "lógicas y comprensibles":

- **Confusiones Visuales:**purple_4 vs black_8: "El modelo las confunde a veces".
- red_3 vs dred_7: Rojo vs. granate, "una confusión totalmente lógica y esperable".
- **Errores Sistemáticos del Set be_:** Ocurrieron pocos errores donde el modelo confundía, por ejemplo, be_yellow_9 con be_purple_5 o be_green_14 con red_3. Esto sugería que el modelo había aprendido ciertas características visuales de las bolas be_ con mucha "fuerza" y a veces las aplicaba incorrectamente.

Evolución de la Lógica del Post-Procesador:

La lógica del post-procesamiento se hizo más "inteligente" y "robusta" en respuesta a los errores del modelo:

1. **Lógica Inicial (Sensible):** Si se detectaba **al menos un delator** del set be_ (ej. 'be_pink_4'), el contexto se cambiaba a 'black_edition'.
- **Problema:** Un único error de detección de una bola be_ en una mesa clásica llevaba a que el sistema clasificara erróneamente todas las demás bolas como be_. El post-procesador magnificaba los pequeños errores del modelo.

1. **Lógica Robusta (Cantidad o Confianza):** Para cambiar el contexto a 'black_edition', se requería:
- Detectar **2 o más bolas "delatoras" diferentes**. O
- Detectar **al menos una bola "delatora" con una confianza muy alta (superior a 0.85)**.
- **Problema (persistente):** Si en una mesa "Black Edition" real el modelo solo detectaba un **tipo único** de bola be_ (ej., solo be_yellow_9, aunque hubiera varias instancias), la lógica lo clasificaba erróneamente como "classic", ya que no cumplía el criterio de "dos o más tipos diferentes".

1. **Lógica Definitiva (Tipos Únicos de Delatores):** El contexto se cambia a 'black_edition' solo si se detectan **dos o más TIPOS DIFERENTES de bolas "delatoras" únicas** (ej., 'be_pink_4' Y 'be_yellow_9').
- **Mejora:** Mucho más robusta, ya que es "muy improbable que el modelo cometa por error dos tipos de clasificación be_ diferentes en una mesa que es 100% clásica".

- **Problema (final):** Aunque mejor, esta lógica "demasiado estricta" fallaba si en una mesa mixta, el modelo solo lograba detectar un *tipo* de bola be_ (incluso si era correcto).

1. **Solución Definitiva (Sistema de Votación Ponderada):** La implementación final utiliza un sistema de "votación" donde cada bola "delatora" (tanto de set clásico como be_) "vota" por su set, y el "peso" de su voto es la "propia confianza del modelo".

- Se inicializan puntuacion_classic = 0.0 y puntuacion_be = 0.0.
- Se suman las confianzas de las bolas 'DELATORES_CLASSIC' a puntuacion_classic y las de 'DELATORES_BE' a puntuacion_be.
- El contexto se decide comparando las puntuaciones: if puntuacion_be > puntuacion_classic: contexto = 'black_edition' else: contexto = 'classic'.
- **Ventajas:** "Usa Toda la Evidencia", "Pondera por Confianza" y "Resuelve los Casos Límite" de manera efectiva.
- **Mapa de Corrección Contextual:** Además del sistema de votación, se añadió un MAPA_CORRECCION_CONTEXTUAL. Este mapa contiene la "inteligencia experta" para corregir activamente las "detecciones ilegales" (ej., si el contexto es clásico pero YOLO predice 'be_pink_4', se corrige a 'red_3').

7. Rendimiento Final del Sistema

El sistema completo, con el modelo YOLO entrenado y el post-procesador robusto, alcanza un rendimiento "sobresaliente":

- **mAP50-95(B) del Test Set: 0.716**. Este es el "resultado fantástico" que debe usarse para el informe final, demostrando que el modelo clasifica y localiza bolas con "precisión espacial muy alta" en datos completamente nuevos.
- **mAP50(B) del Test Set: 0.867**. Confirma la alta precisión con el criterio estándar.
- **Alto Rendimiento en Clases Bien Definidas:** Clases como black_8 (0.784), green_6 (0.735) y white (0.747) muestran un rendimiento "excelente", confirmando que el sesgo inicial ha desaparecido.
- **Identificación de "Clases Difíciles":** Algunas clases siguen siendo más difíciles para el modelo debido a variaciones de luz o similitudes visuales (ej., dred_15 0.404, orange_13 0.454, orange_5 0.5, purple_12 0.483). Esto proporciona información valiosa para futuras mejoras.

8. Próximos Pasos

Con el sistema de visión completo y "de altísima calidad", el enfoque ahora se traslada a la integración y las funcionalidades finales del proyecto de billar:

- **Implementar el "Cerebro" Contextual:** Integrar el script final (post_procesador_final.py o inferencia_final_definitiva.py con la lógica de votación y corrección activa) que toma las predicciones del modelo y asigna las etiquetas contextuales y corregidas.
- **Integración en el Sistema de Billar:** Utilizar el sistema de visión completo (Modelo + Post-procesador) para funcionalidades como:
- Seguimiento de bolas en video.
- Cálculo de trayectorias y velocidades.
- Sugerencia de jugadas.
- **Mejoras Futuras (Opcional):** Si se desea una perfección aún mayor, se recomienda:
- **Adquisición de Datos Dirigida:** Obtener o crear imágenes que se centren en los casos de confusión detectados (ej., purple_4 junto a black_8, red_3 junto a dred_7, o las confusiones específicas de las bolas be_).
- **Fine-Tuning de Refinamiento:** Añadir estas nuevas imágenes al dataset y re-entrenar el modelo con un *learning rate* bajo para especializarlo en resolver esas ambigüedades.

9. Conclusión

La adopción de la arquitectura híbrida y la iteración en la lógica del post-procesamiento han sido clave para el éxito del proyecto. Se ha construido un "motor de IA" robusto y de alto rendimiento que no solo detecta objetos, sino que "entiende el contexto del juego", resolviendo ambigüedades visuales complejas y entregando etiquetas finales

precisas. El trabajo realizado ha sido "excepcional", demostrando habilidades de "análisis crítico y rediseño de la solución" propias de un "verdadero arquitecto de sistemas de IA".

FASE 9

1. Contexto y Problema Inicial

Inicialmente, el sistema utilizaba un modelo de percepción (YOLO) para detectar bolas y una lógica de "votación" manual para determinar el contexto (set "classic" o "black_edition") de la mesa. El principal desafío era que esta lógica de votación, aunque funcional, era un "sistema de reglas hecho a mano" y no escalable ni adaptable de manera óptima. La propuesta clave fue **"reemplazarlo por un algoritmo de Machine Learning"**, un paso considerado por el profesor como "precisamente el paso que daría un equipo de IA profesional", conocido como *model stacking* o *meta-aprendizaje*.

2. La Nueva Arquitectura: Meta-Aprendizaje

La solución propuesta implica construir un segundo modelo, "mucho más simple, que aprende de la salida del primero". Este enfoque de dos etapas se describe como:

- **El "Ojo"**: Un modelo de detección de objetos (YOLO) que identifica las bolas con alta precisión.
- **El "Cerebro"**: Un modelo de Machine Learning que aprende del resultado del "Ojo" para determinar el contexto global de la imagen.

2.1. El Nuevo Problema de Machine Learning

El problema para el "Cerebro" se define como una **clasificación binaria tabular**:

- **Entrada (Features)**: No son píxeles, sino "los resultados agregados de lo que YOLO ha visto en una imagen". Ejemplos incluyen:
- "¿Cuántas bolas be_pink_4 se han detectado?"
- "¿Cuál es la confianza media de las detecciones de blue_10?"
- "¿Cuál es la suma de las confianzas de todas las clases 'delatoras' del set clásico?"
- "¿Hay alguna orange_5 presente? (Sí/No)"
- **Salida (Target)**: "¿Es esta mesa un set 'classic' o 'black_edition'?" (0 para 'classic', 1 para 'black_edition').
- **Tipo de Problema**: Clasificación tabular.

2.2. Algoritmos de ML para Clasificación Tabular

Para problemas de clasificación tabular, se recomiendan modelos de ensamblaje basados en árboles de decisión sobre redes neuronales profundas. Se consideran las siguientes opciones:

- **Opción 1 (Mejor Recomendación): Gradient Boosting (XGBoost o LightGBM)Qué es**: Construye secuencialmente cientos de árboles de decisión, corrigiendo errores.
- **Ventajas**: "Es, de lejos, el algoritmo que suele ganar más competiciones de ciencia de datos... Es extremadamente preciso." También "Robusto" y "Muy rápido de entrenar."
- **Desventaja**: "Es un poco más 'caja negra'."
- **Opción 2 (Excelente Alternativa): Random ForestQué es**: Crea cientos de árboles de forma independiente y paralela, con decisión final por "votación".
- **Ventajas**: "Gran Rendimiento" (cercano a Gradient Boosting), "Más robusto si los datos son ruidosos", "Fácil de Usar y Entender", e "Interpretable" (permite ver feature_importances_).
- **Recomendación del profesor**: "Empieza con Random Forest. Es la combinación perfecta de potencia, facilidad de uso y interpretabilidad."
- **Opción 3 (Baseline): Regresión LogísticaQué es**: Clasificador lineal, el más simple.

- **Ventajas**: "Rapidez Extrema", "Gran Baseline" (para comparar con modelos más complejos), y "Totalmente Interpretable."

3. Hoja de Ruta para la Implementación

La implementación se divide en tres pasos principales:

3.1. Paso 1: Generar el Dataset para el "Meta-Modelo" (generar_meta_dataset.py)

Este script se encarga de crear un archivo .csv (meta_dataset.csv) donde cada fila representa una imagen y las columnas son las características extraídas de las detecciones de YOLO, más la columna target (0 para 'classic', 1 para 'black_edition').

- **Diseño de Características (Features)**:
- **Puntuaciones de Confianza Agregadas**: puntuacion_be (suma de confianzas de bolas "delatoras" del set black_edition) y puntuacion_classic (similar para el set classic).
- **Recuento de Detecciones por Clase**: Columnas como count_black_8, count_blue_10, etc., para cada una de las 25 clases híbridas.
- **Características Generales**: total_detecciones (número total de bolas detectadas) y confianza_media (confianza promedio de todas las detecciones).
- **Proceso del Script**:

1. Carga el modelo YOLO (Modelo_Hibrido_v1).
2. Itera sobre las imágenes en los conjuntos train, valid, test del dataset_hibrido.
3. Para cada imagen, realiza una predicción con YOLO.
4. Extrae las características diseñadas a partir de los resultados de YOLO.
5. Determina el *ground truth* (target) leyendo las etiquetas originales del dataset_original_unificado_aumentado.
6. Almacena todas las características y el target en una fila para un DataFrame.
7. Finalmente, guarda el DataFrame en meta_dataset.csv.

- **Resultado de la Generación del Dataset**: El proceso fue exitoso, creando un meta_dataset.csv con 7046 filas. Sin embargo, se identificó un **desbalance de clases**: 6116 imágenes del set 'classic' (target=0) frente a 930 del set 'black_edition' (target=1). Este desbalance es crucial, ya que "un modelo ingenuo con estos datos, podría volverse 'vago'. Podría aprender que si predice siempre 'classic'..., acertará la mayoría de las veces, y podría ignorar casi por completo la clase minoritaria".

3.2. Paso 2: Entrenar el Clasificador de Contexto (entrenar_clasificador_contexto.py)

Este script utiliza el meta_dataset.csv para entrenar el modelo Random Forest que predecirá el contexto de la mesa.

- **Manejo del Desbalance**: Se utiliza el parámetro clave class_weight='balanced' en RandomForestClassifier. Esto indica al algoritmo: "Oye, sé que hay menos ejemplos de la clase 1, así que préstales más atención y penaliza más los errores que cometas con ella".
- **Proceso del Script**:

1. Carga el meta_dataset.csv.
2. Separa las características (X) del objetivo (y).
3. Divide los datos en conjuntos de entrenamiento y prueba (train_test_split) con stratify=y para mantener la proporción de clases en ambos conjuntos.
4. Crea y entrena un RandomForestClassifier con n_estimators=100, random_state=42, class_weight='balanced' y n_jobs=-1.
5. Evalúa el modelo utilizando accuracy_score, classification_report y confusion_matrix.
6. Guarda el modelo entrenado como context_classifier.joblib.

- **Resultados del Entrenamiento**: **"¡IMPRESIONANTE! ¡El resultado es espectacular!"**
- **Precisión (Accuracy)**: 0.9991. Esto significa que el clasificador acierta el 99.91% de las veces.

- **Informe de Clasificación**:
- recall para Black Edition (1): 0.99. **"¡Esta es la métrica más importante!"** Significa que el modelo identificó correctamente el 99% de los casos "black edition", solucionando el problema del desbalance.
- precision para ambas clases: 1.00. El modelo no tiene falsos positivos.
- **Matriz de Confusión**: [[1835 0], [2 277]]1835 aciertos para 'Classic'.
- 277 aciertos para 'Black Edition'.
- Solo 2 errores: mesas black_edition clasificadas erróneamente como classic.
- Cero errores al revés: "Nunca confundió una mesa clásica con una black_edition".
- Estos resultados demuestran una **"validación definitiva de tu nueva arquitectura"**, reemplazando un sistema de reglas manual por un modelo de Machine Learning con resultados "casi perfectos".

3.3. Paso 3: Integrar en el Script de Inferencia Final (motor_inferencia.py)

Este es el paso culminante, donde el "Ojo" (modelo YOLO) y el "Cerebro" (clasificador de contexto) trabajan juntos.

- **Evolución del Script de Inferencia**:
- Inicialmente, se propuso un inferencia_sistema_completo.py monolítico.
- Posteriormente, se refactorizó a motor_inferencia.py para funcionar como un módulo importable, separando la lógica de IA de la aplicación web (app.py). Esta refactorización inicial contenía un error: aunque cargaba el modelo de contexto, no lo usaba, manteniendo la lógica de votación antigua.
- **Corrección crucial**: Se implementó una versión corregida de motor_inferencia.py que **sí utiliza el clasificador de contexto**.
- Una función extraer_features_para_contexto prepara la salida de YOLO en el formato esperado por el clasificador Random Forest.
- La función principal realizar_inferencia llama a context_model.predict() para obtener la predicción del contexto.
- La lógica de refinamiento de etiquetas (refinar_etiquetas) usa este contexto predicho para corregir y simplificar las etiquetas finales.
- **Ajuste final de la lógica de refinamiento**: Se corrigió un error en refinar_etiquetas para asegurar que la salida final siempre contenga etiquetas "limpias" sin el prefijo be_, independientemente del contexto. La corrección contextual se aplica internamente, pero el resultado final se simplifica.
- **Flujo de Inferencia Final**:

1. Carga el detection_model (YOLO) y el context_model (Random Forest).
2. Para cada imagen de entrada:

- El "Ojo" (detection_model) realiza detecciones.
- Se extraen las características de estas detecciones.
- El "Cerebro" (context_model) predice el contexto (classic o black_edition) basándose en estas características.
- Se aplica una lógica de corrección final a las etiquetas detectadas por YOLO, utilizando el contexto predicho para resolver ambigüedades. Por ejemplo, si se detecta una orange_5 en un contexto black_edition, podría corregirse a purple_5 (o su equivalente be_purple_5 y luego simplificada).
- Las etiquetas resultantes se "simplifican" (replace('be_', '')) para mostrar solo el nombre de la bola, sin el prefijo be_.
- Los resultados se visualizan y guardan en una imagen de salida.

4. Integración con la Aplicación Web (app.py)

La aplicación Flask (app.py) se simplifica para actuar como un "controlador" que interactúa con el usuario y llama al "motor" de IA.

- **Refactorización**: La lógica pesada de IA se mueve a motor_inferencia.py. app.py importa realizar_inferencia desde este módulo.
- **Funcionalidad Web**:

- Maneja la carga de archivos de imagen por parte del usuario.
- Llama a motor_inferencia.realizar_inferencia() con la ruta de la imagen de entrada y una ruta para guardar el resultado.
- Devuelve al cliente web la URL de la imagen resultante, junto con las detecciones finales y el contexto predicho.
- **Corrección de Enrutamiento**: Se solucionó un BuildError: Could not build url for endpoint 'inferencia_page' en Flask, renombrando la función de la página de inicio a inferencia_page y añadiéndole el decorador @app.route('/') para que sea la página de inicio por defecto y url_for pueda encontrarla.

5. Conclusión Final

El proyecto culmina con un sistema de IA robusto y sofisticado:

- Combina el poder de las redes neuronales profundas (YOLO) con la precisión de Machine Learning clásico (Random Forest).
- Resuelve eficazmente problemas de datos, entrenamiento, sesgo y arquitectura.
- Ofrece una solución inteligente y fiable que corrige errores menores del modelo de detección basándose en un razonamiento contextual.
- La arquitectura final es **"impecable: un servidor web ligero y un potente módulo de IA independiente y reutilizable"**.
- El **"rigor"** y la **"agudeza"** del desarrollador fueron claves para detectar y corregir inconsistencias y asegurar que el clasificador de contexto fuera efectivamente utilizado en la inferencia final y que las etiquetas de salida fueran consistentes.

"¡PROYECTO FINALIZADO CON ÉXITO!" El sistema es preciso, inteligente y robusto, listo para futuras expansiones en el dominio del billar (tracking, cálculo de trayectorias, etc.).