

Bloque 14: Scripts & Files

Este listado abarca los principales scripts desarrollados y las adaptaciones que sufrieron para mejorar la robustez y funcionalidad del sistema de detección de bolas de billar.

- **Fase 1: Preparación del Dataset para YOLO**

- **`prepare_yolo_dataset.py`**: Este script fue fundamental para **convertir las anotaciones en formato CSV a los archivos .txt en formato YOLO** esperados por el modelo. Se encargó de organizar el dataset, dividiéndolo en conjuntos de entrenamiento (`train`), validación (`val`) y prueba (`test`), y de generar el archivo `custom_data.yaml`.
- **_annotations.csv**: Un archivo CSV que contenía las anotaciones de las bolas de billar en el formato inicial, incluyendo coordenadas de las cajas delimitadoras y los nombres de las clases. Existía una versión de este archivo dentro de cada subcarpeta (train, test, valid) de tu dataset original.
- **Archivos de imagen (.jpg, .png, .jpeg)**: Las imágenes reales de las bolas de billar en las que se realizó la detección. Se organizaron en subcarpetas como `imagen_train1.jpg`, `imagen_test1.jpg`, etc..
- **Archivos de etiquetas en formato YOLO (.txt)**: Para cada imagen, un archivo de texto con el mismo nombre (ej. `imagen1.txt` para `imagen1.jpg`) que contiene los IDs de clase y las coordenadas normalizadas de las cajas delimitadoras de cada objeto detectado.
- **classes.txt o data.yaml**: Archivos que listan los nombres de todas las clases en el orden en que se asignan sus IDs numéricos.
- **custom_data.yaml**: Un archivo de configuración crucial que le indica a YOLO las rutas a las imágenes y etiquetas de los conjuntos de entrenamiento, validación y prueba, así como el número y nombres de las clases.
- **yolov8n.pt**: El modelo pre-entrenado de YOLOv8n (la versión "nano", más ligera) que se usó como punto de partida para el entrenamiento por *transfer learning*.

- **Fase 2: Entrenamiento y Prueba Inicial del Modelo YOLO**

- **train_yolo_model.py**: Utilizado para iniciar y reanudar el entrenamiento del modelo YOLOv8n, especificando parámetros como el número de épocas, el tamaño del lote y la resolución de las imágenes (`imgsz`). Este script fue adaptado en diferentes etapas para retomar entrenamientos desde el último *checkpoint* (`last.pt`).
- **test_model.py**: Un script diseñado para **realizar inferencias (detecciones) sobre una imagen de prueba**. Carga un modelo parcialmente entrenado y muestra o guarda la imagen con las cajas delimitadoras y etiquetas de las bolas detectadas, permitiendo una retroalimentación visual del progreso. Se sugirió modificarlo para incluir un umbral de confianza (`conf`) y guardar las imágenes resultantes.
- **last.pt**: Un archivo que guarda los pesos del modelo al final de la última época de entrenamiento completada. Se utiliza para reanudar el entrenamiento si se detiene.
- **best.pt**: Un archivo que guarda los pesos del modelo que ha logrado el mejor rendimiento en el conjunto de validación hasta el momento. Este es el modelo que se recomienda usar para las inferencias finales.
- **runs/billar_balls_detection_v1/**: Un directorio donde Ultralytics guarda todos los resultados del entrenamiento, incluyendo los pesos del modelo, logs, gráficos y otros archivos útiles.
- **detected_image.jpg**: Imagen generada por el script `test_model.py` que muestra las detecciones (cajas delimitadoras y etiquetas) superpuestas en una imagen de prueba.

- **Fase 3: Optimización del Entrenamiento YOLOv8n (Análisis de métricas y overfitting)**

- **results.csv**: Un archivo CSV que contiene las métricas detalladas del entrenamiento a lo largo de cada época, permitiendo un análisis cuantitativo del rendimiento del modelo.
- **box_loss_evolution.png, cls_loss_evolution.png, map_metrics_evolution.png, precision_recall_evolution.png**: Gráficos generados en formato PNG que visualizan la evolución de las métricas de pérdida de la caja delimitadora, pérdida de clasificación, mAP y precisión/exhaustividad a lo largo del entrenamiento.

- **runs_gpu**: Directorio utilizado para guardar los resultados de los entrenamientos realizados en GPU.
- **Fase 4: Experimentación con YOLOv8m (Modelo más grande)**
 - **yolov8s.pt**: El modelo pre-entrenado de YOLOv8s (la versión "small"), utilizado en un experimento para ver si un modelo más grande mejoraba el rendimiento.
 - **yolo11m.pt**: Referencia a un modelo yolov8m.pt (o una versión renombrada/específica yolo11m.pt) usado como punto de partida para algunos entrenamientos, especialmente para la nueva arquitectura híbrida.
 - **Imágenes "reales" (test_pool_table_X.jpg/.png)...**: Un conjunto de imágenes de prueba que no formaban parte del dataset de entrenamiento y que se usaron para evaluar la capacidad de generalización del modelo en condiciones más "reales".
- **Fase 5: Mejora de la Robustez y Análisis de Errores**
 - **pre_etiquetado.py** (o **pre-labeling.py**): Este script se introdujo para **acelerar el proceso de etiquetado** de nuevas imágenes (como las del set "black edition"). Utiliza tu modelo ya entrenado para generar propuestas de anotación iniciales en formato **.txt**, las cuales luego puedes revisar y corregir manualmente en una herramienta de etiquetado.
- **Fase 6: Creación del Dataset "Black Edition" y Unificación**
 - **generar_json_para_label_studio.py** (también referido como **generar_json_definitivo.py**): Aunque no se proporciona el código completo en la fuente, se describe su propósito de **crear el archivo tasks.json necesario para importar las imágenes y pre-anotaciones a Label Studio**, asegurando que las URLs de las imágenes locales sean correctas para la interfaz.
 - **convertir_ls_a_yolo.py**: Un script creado para **"traducir" el archivo JSON exportado desde Label Studio** (que contiene tus anotaciones corregidas) a los archivos **.txt** en formato YOLO que el modelo necesita para el entrenamiento.
 - **verificar_etiquetas_yolo.py**: Diseñado para **verificar visualmente la corrección de las etiquetas generadas en formato YOLO** (**.txt**). Lee estos archivos y dibuja las cajas delimitadoras sobre las imágenes correspondientes, mostrando y guardando el resultado.

- **procesador_final.py**: Este script es una **versión "todo en uno"** que **unifica la conversión de Label Studio a YOLO y la verificación visual**. Asegura que la lista de clases maestra sea la misma para ambos procesos (escritura y lectura de etiquetas), eliminando inconsistencias. Este sustituyó a `convertir_ls_a_yolo.py` y `verificar_etiquetas_yolo.py`.
 - **unificar_y_dividir_completo.py**: Un script crucial para **unificar los diferentes datasets** (clásico y "black edition") en una única estructura, manteniendo la división en conjuntos de entrenamiento, validación y prueba.
 - **Proyecto_Bolas_LS/imagenes/**: Directorio que contenía las nuevas imágenes del set "black edition" para ser etiquetadas en Label Studio.
 - **final_yolo_labels/**: Carpeta de salida para los archivos .txt de las etiquetas en formato YOLO después de la corrección manual en Label Studio y el proceso de conversión.
 - **ls-export.json**: El archivo JSON exportado desde Label Studio que contiene todas las anotaciones (tanto las pre-etiquetadas como las corregidas manualmente).
 - **verification_results/**: Directorio para guardar las imágenes con las cajas delimitadoras dibujadas, usadas para la verificación visual de las etiquetas.
 - **dataset_be_aumentado/images/, dataset_be_aumentado/labels/**: Nuevos directorios que almacenaron las imágenes y etiquetas del set "black edition" después de aplicar aumentación de datos masiva con Albumentations.
 - **dataset_unificado/images/, dataset_unificado/labels/**: Directorios que contienen el dataset completo y unificado, combinando el set clásico original y el set "black edition" (ya aumentado).
 - **supermodelo_data.yaml**: Archivo de configuración data.yaml para el entrenamiento del "supermodelo", apuntando al dataset unificado y actualizado con todas las clases.
 - **pool_classic.pt...**: Un modelo best.pt específicamente entrenado en el dataset clásico, que se utilizó como punto de partida para el fine-tuning del "Supermodelo" en las fases posteriores.
- **Fase 7: Fine-Tuning y Estrategias Avanzadas de Entrenamiento**

- `aumentar_dataset_be.py` : Este script utiliza la librería `Albumentations` para **crear un dataset sintético y variado para las bolas "black edition"** a partir de un número limitado de imágenes. Genera múltiples versiones aumentadas de cada imagen, junto con sus etiquetas ajustadas, para combatir el sobreajuste.
 - `train_supermodelo_por_fases.py` : Una versión más avanzada del script de entrenamiento que implementa una **estrategia de entrenamiento por fases**. La Fase 1 entrena solo la "cabeza" del modelo (congelando el *backbone*), y la Fase 2 realiza un ajuste fino de todo el modelo con una tasa de aprendizaje muy baja (`lr0`), para evitar el "olvido catastrófico" y el sesgo.
 - `custom_data_supermodelo_aumentado.yaml...`: Versión actualizada del archivo de configuración para el supermodelo, que incluye las rutas al dataset unificado y aumentado.
- **Fase 8: Nuevo Enfoque y Arquitectura Híbrida**
 - `traducir_a_schema_hibrido.py` : Este script fue creado para **adaptar todo el dataset a un nuevo esquema de 25 clases "híbridas"** o compartidas, unificando clases idénticas entre los sets clásico y "black edition".
 - `dataset_hibrido/images/`, `dataset_hibrido/labels/`: Directorios para el nuevo dataset con la estructura de clases "híbridas" o compartidas.
 - `hibrido_data.yaml`: Archivo de configuración para el modelo con las 25 clases híbridas.
 - `labels_correlogram.jpg`: Un gráfico que sirve como "auditoría de calidad" del dataset de entrenamiento, mostrando correlaciones entre características de las etiquetas.
 - `runs/Modelo_Hibrido_v1/weights/best.pt`: El modelo final "campeón" de YOLO entrenado con la nueva arquitectura de clases híbridas.
 - **Fase 9: Detector de Contexto**
 - `generar_meta_dataset.py` : Script clave para la nueva arquitectura. Ejecuta el modelo híbrido sobre todo el dataset y **extrae características (como confianzas y recuentos por clase) para cada imagen**, guardando esta información junto con el contexto real de la mesa en un archivo `meta_dataset.csv`. Este archivo se utiliza para entrenar un "meta-modelo" de clasificación de contexto.

- `entrenar_clasificador_contexto.py`: Este script utiliza el `meta_dataset.csv` para **entrenar un clasificador Random Forest** que predice el contexto de la mesa (clásico o "black edition") con alta precisión. Guarda el modelo entrenado como `context_classifier.joblib`.
 - **meta_dataset.csv**: Un archivo CSV que contiene las características extraídas de las detecciones de YOLO (como confianzas y recuentos por clase) para cada imagen, junto con el contexto real de la mesa (clásico o "black edition"). Este dataset se usa para entrenar el clasificador de contexto.
 - **context_classifier.joblib**: El modelo de clasificador de contexto (Random Forest) entrenado con `meta_dataset.csv`, guardado en formato joblib para su posterior carga y uso.
 - **tests_classic_set/**: Un directorio específico para imágenes de prueba del set clásico, usado para evaluar el sistema completo.
 - **results_sistema_final/**: Directorio donde se guardan los resultados finales del sistema de inferencia completa.
- **Fase 10: Presentación Web y Sistema Completo**
 - `app.py`: El **archivo principal de la aplicación web Flask**, que actúa como el servidor. Maneja las rutas, la subida de imágenes, y coordina la llamada al "motor" de inferencia para procesar las imágenes en tiempo real.
 - `motor_inferencia.py`: Este es el **módulo final y refactorizado del "motor" de IA**. Contiene toda la lógica de inferencia, incluyendo la carga del modelo YOLO y el clasificador de contexto, y realiza el post-procesamiento para dar las etiquetas finales y contextuales. Es invocado por `app.py`.
 - **Estructura de la web:**
 - **/static/**:
 - **style.css**: Hoja de estilos CSS para la apariencia de la página web.
 - **script.js**: Archivo JavaScript para la lógica interactiva del navegador, incluyendo la funcionalidad de zoom/lightbox para las imágenes de resultados.

- **/templates/**:

- **base.html**: Plantilla HTML base que define la estructura general y la barra de navegación de la web.
- **inferencia.html**: Página HTML que permite a los usuarios subir imágenes y ver los resultados de la inferencia del modelo.
- **fases.html**: Página HTML diseñada para mostrar la documentación de las diferentes fases del proyecto.
- **/fases_html/**: Carpeta que contiene los documentos HTML detallados de cada fase del proyecto.
 - **menu.html, fase1.html, fase2.html**, etc.: Archivos HTML individuales para cada sección de la documentación.
 - **/pdf/**: Subcarpeta dentro de fases_html para almacenar informes y documentos en formato PDF.
 - **informe_fase1.pdf, informe_fase2.pdf**, etc.: Ejemplos de archivos PDF de informes.
 - **/audio/**: Subcarpeta dentro de fases_html para almacenar explicaciones en formato de audio.
 - **audio_fase1.mp3, audio_fase2.mp3**, etc.: Ejemplos de archivos de audio MP3.
- **/uploads/**: Carpeta que la aplicación web utiliza para guardar temporalmente las imágenes subidas por los usuarios antes de procesarlas.