



UNIVERSIDAD DE MÁLAGA

Trabajo fin de grado

Sistema de tracking y reidentificación de jugadores en fútbol amateur

Realizado por
Soriano Muñoz Juan Ignacio

Profesor encargado:
Luque Baena Rafael Marcos
Jerez Aragónés Jose Manuel

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, DICIEMBRE de 2024



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
ESTUDIANTES DE INGENIERÍA BIOINFORMÁTICA

Sistema de tracking y reidentificación de jugadores en fútbol amateur

Trabajo fin de grado

Realizado por
Soriano Muñoz Juan Ignacio

Profesor encargado:
Luque Baena Rafael Marcos
Jerez Aragonés Jose Manuel

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, DICIEMBRE DE 2024

Contents

1	Introducción	4
2	Diario de avances	4
2.1	Semana 3-16 de marzo)	4
2.2	Semana 17-31 de marzo [3]	4
2.3	Semana 31-6 de abril)	6
2.4	Semana 7-13 de abril)	7
2.5	Semana 7-13 de abril)	8
3	Motivación y Contexto	8
3.1	¿Por qué es importante el análisis en fútbol amateur?	8
3.2	¿Qué problema hay actualmente?	8
4	Objetivos	8
5	Estado del sector	9
5.1	Soluciones en el Ámbito Profesional	9
5.2	Herramientas Disponibles para el Fútbol Amateur	10
5.3	Limitaciones	11
6	Tecnologías empleadas	11
6.1	Programación	11
6.2	Gestión de datos y flujo de trabajo	12
6.3	Modelos utilizados	12
7	Módulo de detección (YOLO)	12
7.1	Arquitectura YOLO	12
7.2	¿Cómo funciona la detección de objetos con YOLO?	13
8	Módulo de tracking + Gestión de IDs	16
8.1	Módulo de tracking	16
8.2	Gestión de IDs	16
9	Propuesta	17
10	Arquitectura general del sistema	18
10.1	Preparar el dataset (Roboflow)	18
10.2	Entrenamiento del modelo de YOLO	19
10.3	Preparar el dataset de entrenamiento del modelo de ReID	20
10.3.1	annotate_review.py	20
10.3.2	annotate_review_precise.py	24
10.3.3	dicMaker_idPlayer.py	28
10.3.4	dicMakerForTrainingReID.py	30
10.4	Entrenar el modelo de ReID	32
10.5	Preparar el dataset de testeo del modelo de ReID	37
10.6	Aplicar modelo de detección de objetos (YOLOX)	38
10.6.1	Configuración del Modelo YOLOX en yolox_x_ch_sportsmot.py	38
10.6.2	Implementación en demo.py	39

10.7 Aplicar tracker (DeepEIoU) y modelo de reID	40
10.7.1 Deep-EIoU en DeepEIoU.py	40
10.7.2 Modelo de Reidentificación (ReID)	41
10.8 Aplicar GTA-LINK	42
10.8.1 generate_tracklets.py	42
10.8.2 refine_tracklets.py	43
11 Resultados Preliminares	44
11.1 Definiciones de las métricas usadas	44
11.1.1 Rank-k Accuracy (Rank-1, Rank-5, Rank-10, etc.)	44
11.1.2 mAP – mean Average Precision	45
11.2 Comparación de protocolos de re-identificación	45
11.2.1 Ejemplo de evaluación de Re-Identification	46
11.3 Evaluación de la re-identificación	47
11.3.1 YOLOv10	48
11.3.2 Osnet Sportsmot	49
11.3.3 Osnet Soccernet	50
11.3.4 ResNet Soccernet	51
11.3.5 Osnet mi dataset estiquetado	52
11.4 Resultados de MOT	52
12 Dificultades encontradas	53
13 Conclusiones	53

1 Introducción

2 Diario de avances

2.1 Semana 3-16 de marzo)

Por ahora lo que llevamos es un dataset hecho en roboflow con un partido de España contra Suiza. Realizamos capturas y dividimos el conjunto en training, validation y test.

Entrené el modelo de YOLO con este dataset revisado y el modelo no supo detectar bien el balón debido a la poca cantidad de imágenes donde se pueda ver bien la bola. El árbitro y los jugadores fueron bien detectados.

Se replanteó el objetivo del TFG. Se focalizará en la reidentificación de jugadores cuando salen fuera de plano y en el desarrollo de una aplicación que permita al usuario decidir si cuando se produce un cambio de identificador, mantenerlo o cambiarlo, creando un dataset revisado.

2.2 Semana 17-31 de marzo [3]

A la hora de medir resultados como estas gráficas:

Results on Datasets

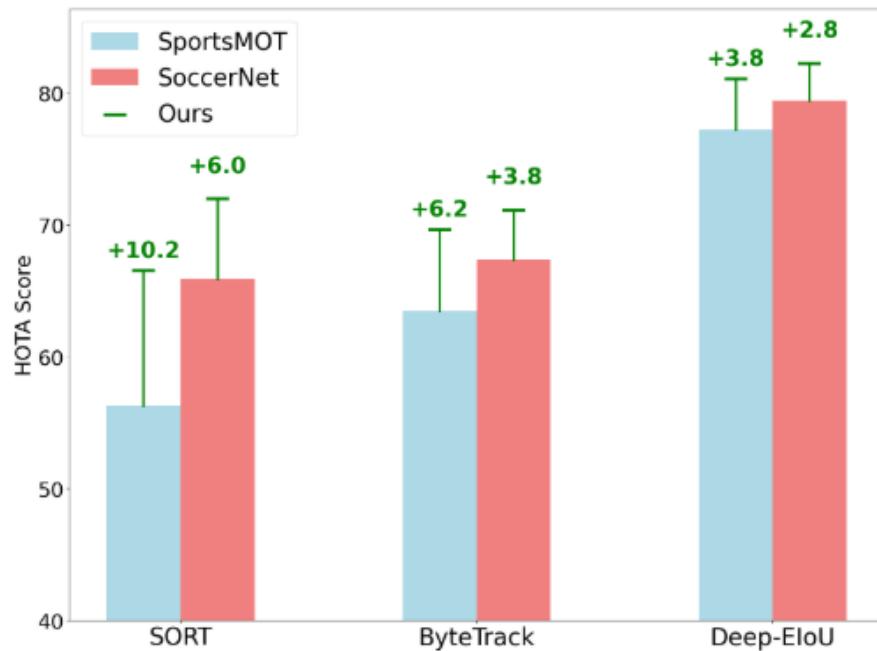


Figure 1: Métricas gta_link

Nos centraremos en la medida de los ID's intentando minimizarlos, lo máximo posible, ya que la métrica significa número de ids generados.

Me he puesto a mirar lo que hace exactamente el código del repositorio de gta-link. Y en resumidas cuentas necesitamos un dataset con una pred, ya realizada,

Table 1: Tracking performance on SportsMOT before and after applying our Global Tracklet Association (GTA) method.

Method	HOTA↑	AssA↑	IDF1↑	DetA↑	MOTA↑	IDs↓
SORT [26]	56.28	42.67	58.83	74.30	85.11	5180
SORT + GTA	66.52 (+10.24)	59.59 (+16.92)	77.37 (+18.54)	74.29	85.27	3547 (-1633)
ByteTrack [31]	63.46	51.81	70.76	77.81	94.91	3147
ByteTrack + GTA	69.74 (+6.28)	62.61 (+10.80)	83.16 (+12.40)	77.72	95.01	2107 (-1040)
Deep-EIoU [19]	77.21	67.63	79.81	88.22	96.30	2909
Deep-EIoU + GTA	81.04 (+3.83)	74.51 (+6.88)	86.51 (+6.70)	88.21	96.32	2737 (-172)

Figure 2: Objetivo

como es el caso de SoccerNet. Podemos descargar el dataset con el tracking realizado en los archivos .txt que se encuentran dentro de cada clip.

He utilizado las siguientes líneas de comando para realizar la descarga del dataset.

```
from SoccerNetDownloader import SoccerNetDownloader
mySoccerNetDownloader = SoccerNetDownloader(LocalDirectory="path/to/SoccerNet")
mySoccerNetDownloader.downloadDataTask(task="tracking",
    split=["train", "test", "challenge"])
```

Posteriormente hice unos ajustes en el código para que cogiera los archivos `gt.txt` para que se tomara como referencia `-pred_dir tracking results directory`, ya que estos son archivos **MOT**, que guardan información sobre los **bounding box** de cada objeto de cada frame, conteniendo información sobre **frame**, **id**, **bb_left**, **bb_top**, **bb_width**, **bb_height**, **conf**, **x**, **y**, **z**, en este orden.

Estos son esenciales para que se pueda ejecutar el archivo `generate_tracklets.py` para generar los **tracklets**.

Un **tracklet** se genera al seguir a un objeto desde su detección en un fotograma hasta el siguiente. En este proceso, el código asocia las detecciones de objetos de cada fotograma con un identificador único (**ID**) para cada objeto, y los agrupa en una **"trayectoria"** que sigue ese objeto a lo largo del tiempo.

En el código:

- **Extracción de características:** El código utiliza un modelo de reidentificación de personas (`FeatureExtractor`) para extraer características visuales de cada objeto detectado en los fotogramas. Esto es útil para seguir objetos entre fotogramas, incluso cuando se producen cambios de apariencia debido a variaciones en la vista o el movimiento.
- **Cálculo de tracklets:** Para cada fotograma, el código agrupa las detecciones de objetos utilizando el identificador único (`track_id`). Si un objeto se detecta en un fotograma y luego aparece en otro, se agrega a un tracklet, que es una colección de todas las detecciones del mismo objeto a lo largo de varios fotogramas. Además, se guarda información como el puntaje de la detección y las características extraídas del modelo de reidentificación.
- **Salvado de tracklets:** Al final de cada secuencia de video o serie de fotografías, los tracklets generados se guardan en un archivo `pickle` para su posterior uso, permitiendo analizar y trabajar con las trayectorias de los objetos en el futuro.

El programa `generate_tracklets.py`, lo que hará será generar unos ficheros `.pkl` que guardan los datos de las trayectorias de los objetos detectados a lo largo del tiempo. Esto incluye, por ejemplo, las **detecciones de objetos en cada fotograma**, las **características extraídas por el modelo de reidentificación**, y los **identificadores únicos (ID)** asignados a cada objeto (en formato binario).

Una vez que se tenga esos ficheros se realiza una **refinación de los tracklets** para evitar que se cambien los **ids con frecuencia**. Mejora los **tracklets** (trayectorias de objetos) generados por un **tracker en tareas de seguimiento de múltiples objetos (MOT)**. Utiliza dos componentes principales: el **Tracklet Splitter**, que divide **tracklets impuros** (con múltiples identidades) en subtracklets más precisos mediante **clustering (DBSCAN)**, y el **Tracklet Connector**, que fusiona **tracklets fragmentados** que pertenecen al mismo objeto basándose en **similitudes visuales y restricciones espaciales**. Los resultados refinados se guardan en archivos `.txt` en formato **MOT**, listos para su evaluación, visualización o análisis posterior. Este proceso optimiza la **precisión del seguimiento**, corrigiendo errores como **cambios de identidad y fragmentaciones**.

Ahora que tenemos un dataset etiquetado, tendremos que:

- **Encontrar una combinación de hiperparámetros óptima:** Con el objetivo de que el cambio de id sea el mínimo posible.
- **Desarrollar una aplicación:** Con el objetivo de que avise al usuario sobre los cambios de id en los frames y confirme si está bien cambiado o debería conservarse el anterior.

2.3 Semana 31-6 de abril)

En estas semanas hemos estado invirtiendo tiempo en las siguientes cosas. En un principio lo que teníamos era un dataset preparado con los ficheros ground truth fruto de un tracker usado (como DeepEIoU). Pero dado un vídeo, no podíamos aplicar los programas de gta-link, por lo que teníamos que conseguir los ficheros MOT de alguna manera.

Había dos opciones. La primera aplicar un tracker y la segunda aplicar un modelo de YOLO.

Para la primera opción busqué en varios repositorios, que trataran con trackers, principalmente DeepEIoU, que es el tracker con el mejor rendimiento según las estadísticas de paperscode [2].

Rank	Model	HOTA↑	IDF1	AssA	MOTA	DetA	Extra Training Data	Paper	Code	Result	Year	Tags
1	DeepEIoU + GTA	81.0	86.5	74.5	96.3	88.2	✓	GTA: Global Tracklet Association for Multi-Object Tracking in Sports	🔗	🔗	2024	
2	Deep HM-SORT	80.1	85.2	72.7	96.6	88.3	✓	Deep HM-SORT: Enhancing Multi-Object Tracking in Sports with Deep Features, Harmonic Mean, and Expansion IOU	🔗	🔗	2024	
3	AED	79.1	81.8	70.1	97.1	89.4	✓	Associate Everything Detected: Facilitating Tracking-by-Detection to the Unknown	🔗	🔗	2024	

Figure 3: Métricas DeepEIoU

Tras encontrar un repositorio que trataba en específico con este [1]. Tuve que modificar código para que utilizara cpu, ya que estaba configurado para gpu de nvidia. Los paquetes no se me instalaban correctamente. Los inputs que utilizaba para los programas eran ficheros tipo .npy.

Dado que me estaba dando muchos problemas, después de dos días decidí que la mejor solución era utilizar un modelo de YOLO. Empecé utilizando un modelo de YOLO estandar capaz de detectar personas. El inconveniente de esto es que el video, estaba grabado desde un ángulo muy bajo, por lo que el modelo detectara las personas de las gradas y esto, a la larga, es un problema.

Entonces para resolver esto había dos opciones. La primera era realizar una segmentación del campo y la segunda entrenar un modelo de YOLO específico para jugadores. La segmentación del campo es demasiado costosa, porque no hay nada automatizado y sería segmentar infinidad de frames. Además de que es específica para un partido de fútbol en un ángulo en específico. Por lo tanto, la mejor opción para mi gusto es entrenar un modelo de YOLO con un dataset, por ahora, específico del partido de balonmano que me proporcionó mi tutor Jose Manuel Jerez. El dataset lo hice creando un programa utilizando las librerías de opencv (cv2) que extraía frames cada 20 segundos que transcurría de vídeo. Los 20 segundos previenen levemente más overfitting del que ocurre por tener frames del mismo partido.

Esta opción me permite etiquetar poco, ya que Roboflow tiene una herramienta de etiquetado automático una vez se han etiquetado unas cuantas imágenes, por lo que simplemente tuve que corregir aquellas imágenes con un mal etiquetado.

Tras tener el dataset etiquetado le añadí algunas augmentations.

A la hora de entrenar el modelo lo hice con 25 epochs. Tardó hora y media en entrenarse.

Apliqué los programas del repositorio de gta-link y los resultados del trackeo son muy buenos, pero la ReID no tiene nada que ver con los resultados tan consistentes del dataset de SoccerNet debido a que la grabación no es profesional.

Aun así tras 23 intentos observe que los parámetros que daban mejores resultados eran los siguientes: `-use_split -min_len 100 -eps 0.8 -min_samples 10 -max_k 4 -use_connect -spatial_factor 1.0 -merge_dist_thres 0.7`

Recientemente le he añadido frames del partido de fútbol 7 al dataset de roboflow. Y he creado un nuevo modelo de YOLO entrenado exclusivamente con ese dataset.

Los objetivos para la próxima semana:

- **Buscar como aplicar un tracker:** Después de aplicar un trackeo con YOLO, sería necesario aplicar DeepEIoU. Investigar si aplicar un doble gta-link puede funcionar para refinar los tracklets todavía más.
- **Seguir desarrollando la aplicación:** Con el objetivo de que avise al usuario sobre los cambios de id en los frames y confirme si está bien cambiado o debería conservarse el anterior. Ya tengo un programa prueba_revision_ReID.py, que funciona como prototipo, pero falta desarrollarlo mucho.

2.4 Semana 7-13 de abril)

Al final el proceso de usar el tracker es obligatorio, ya que los objetos son detectados por YOLO, pero luego los ids son colocados por el tracker, y de ahí saldrán ficheros tipo MOT. Entonces tras aplicar YOLO para tener ficheros con la info de

los bounding box sobre los jugadores intentaremos aplicarle un tracker y de ahí un modelo de ReID, para posteriormente utilizar gta.

Secuencia de trabajo: YOLO + DeepEIoU + GTA

2.5 Semana 7-13 de abril)

Planteé una estructura oficial de lo que llevo hecho para que sea más formal y dar contexto e información sobre todo lo que rodea las prácticas y el proyecto de TFG.

3 Motivación y Contexto

3.1 ¿Por qué es importante el análisis en fútbol amateur?

El análisis de rendimiento, la estadística avanzada y el seguimiento de jugadores han revolucionado el fútbol profesional. Sin embargo, en el fútbol amateur estas tecnologías todavía son muy poco accesibles. Poder aplicar técnicas de tracking y análisis en estos niveles permitiría:

- Mejorar el entrenamiento de jugadores jóvenes.
- Facilitar la toma de decisiones a entrenadores y clubes pequeños.
- Promover el desarrollo del talento de forma más justa y basada en datos.

3.2 ¿Qué problema hay actualmente?

La causa del primer problema que encontramos es la deficiencia en las infraestructuras tecnológicas para la práctica del fútbol amateur; el hecho de no disponer de sistemas avanzados que implementan las cámaras múltiples o los sensores del fútbol en el sector profesional, por el alto coste que implicaría el uso de las soluciones comerciales de tracking (GPS, cámaras VICON, sistemas ópticos, etc.) y la escasez de datos de movimiento y rendimiento que sufren las categorías base y amateur.

4 Objetivos

El presente proyecto tiene como principal objetivo colaborar en el análisis de los deportes a nivel de fútbol amateur implementando un sistema de detección automática, de seguimiento y de reidentificación de los jugadores. A día de hoy, existe una diferencia muy considerable entre la tecnología que existe para el análisis táctico y de rendimiento del fútbol profesional y la del ámbito amateur, lo cual significa que clubes, entrenadores y jugadores amateurs no tienen las herramientas disponible para realizar un análisis automatizado y estructurado de sus partidos. Este trabajo pretende eliminar esta desigualdad creando un flujo de trabajo modular y eficiente que permita aplicar tecnología avanzada de visión por computador en el ámbito amateur. Para ello se combinan modelos de detección de objetos (YOLOX) con algoritmos de seguimiento multiobjeto (DeepEIoU) y técnicas de reidentificación de personas (ReID) con el modelo OSNet, las cuales son reforzadas con un refinamiento posterior (GTA-Link) que mejora la calidad de las trayectorias generadas.

Los objetivos específicos del proyecto son los siguientes:

- Desarrollar un sistema capaz de identificar y seguir automáticamente a los jugadores durante el transcurso de un partido de fútbol grabado en vídeo.
- Implementar un modelo de reidentificación que permita mantener la coherencia de los IDs incluso ante desapariciones temporales del jugador en la escena.
- Aplicar técnicas de post-procesamiento para corregir errores comunes en los tracklets generados, como fragmentación o mezcla de identidades.
- Facilitar la reutilización y escalabilidad del flujo de trabajo en otros entornos deportivos, fomentando así su adopción por parte de entidades del fútbol base o amateur.
- Probar y validar el sistema sobre secuencias reales de fútbol amateur, evaluando su rendimiento en condiciones no controladas.

5 Estado del sector

El análisis de rendimiento en el fútbol ha experimentado una transformación significativa en las últimas décadas, impulsada por avances tecnológicos que permiten una recopilación y procesamiento de datos cada vez más precisos y en tiempo real. Sin embargo, existe una marcada diferencia entre las herramientas utilizadas en el ámbito profesional y las accesibles para equipos amateurs.

5.1 Soluciones en el Ámbito Profesional

En el fútbol profesional, sistemas como TRACAB se han consolidado como referentes en el seguimiento y análisis de jugadores. TRACAB es un sistema de seguimiento óptico que utiliza cámaras y algoritmos de visión por computadora para rastrear en tiempo real la posición de todos los jugadores y el balón en el campo. Este sistema ha sido implementado en ligas de élite como la Premier League, La Liga y la Bundesliga, y está certificado por la FIFA para programas de seguimiento electrónico de rendimiento (EPTS). https://tracab.com/products/tracab-technologies/?utm_source=chatgpt.com

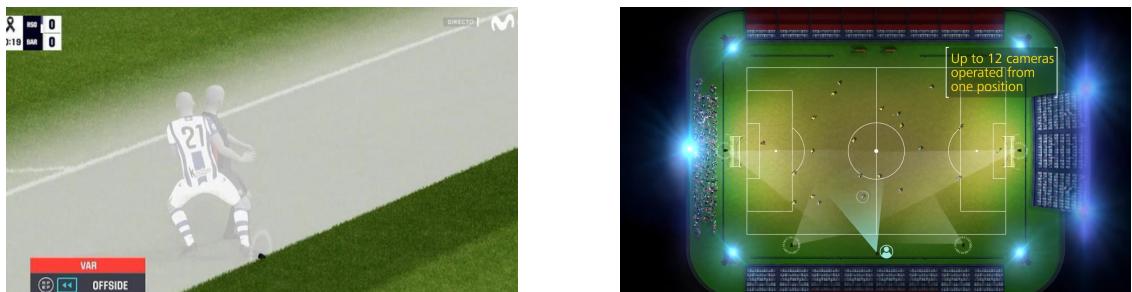


Figure 4: Tracab

5.2 Herramientas Disponibles para el Fútbol Amateur

En contraste, el fútbol amateur ha comenzado a adoptar herramientas más asequibles y adaptadas a sus necesidades y recursos. Una de las más destacadas es LongoMatch, un software de análisis de vídeo que permite a entrenadores y analistas etiquetar eventos, crear estadísticas y generar informes tácticos a partir de grabaciones de partidos.

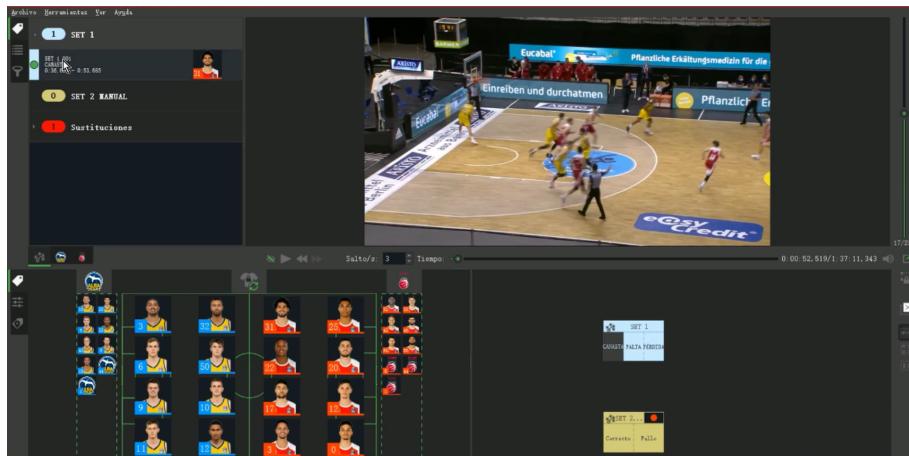


Figure 5: Interfaz de Longomatch

LongoMatch ofrece "<https://longomatch.com/en/>":

- Análisis en tiempo real y post-partido con soporte para múltiples cámaras.
- Personalización de paneles de análisis según las necesidades del equipo.
- Exportación de datos y vídeos para compartir con jugadores y cuerpo técnico.
- Compatibilidad con diversas plataformas y dispositivos móviles.

Además, herramientas como PIX4TEAM 2 han emergido para automatizar la grabación de partidos sin necesidad de un operador de cámara, facilitando la recopilación de material para análisis en equipos con recursos limitados. "<https://shop.movensee.com/es/collections/pix4team-camara-automatica-para-los-deportes-colectivos>"

5.3 Limitaciones

Limitaciones del sector amateur	Aportaciones del presente proyecto
Los sistemas comerciales como Second Spectrum o STATS SportVU requieren múltiples cámaras y hardware costoso.	Se utiliza vídeo monocámara accesible y fácil de capturar con dispositivos comunes.
Los trackers convencionales pierden frecuentemente la identidad del jugador en escenas con occlusiones o cambios de plano.	Se implementa un módulo de reidentificación (ReID) basado en TorchReID y OSNet para mantener la identidad a lo largo del partido.
No existen herramientas que generen anotaciones automáticas de forma sencilla a partir del vídeo.	Se desarrolla un flujo de trabajo que genera archivos de anotaciones (formato MOT) automáticamente a partir del vídeo de entrada.
Los modelos existentes están entrenados en entornos profesionales, con condiciones muy diferentes al fútbol amateur.	Los modelos utilizados en este proyecto han sido entrenados y validados específicamente sobre partidos de fútbol amateur.
Las soluciones actuales no están diseñadas para clubes pequeños: requieren licencias, suscripciones o personal técnico.	Se busca una solución reproducible, ligera y gratuita basada en herramientas <i>open-source</i> , accesible a cualquier usuario técnico.

6 Tecnologías empleadas

Este proyecto ha hecho uso de diversas tecnologías de software y frameworks de inteligencia artificial que permiten implementar un sistema de tracking con detección, seguimiento y reidentificación de jugadores en partidos de fútbol amateur. A continuación, se detallan las principales herramientas y recursos utilizados.

6.1 Programación

Para el desarrollo del sistema se ha utilizado el lenguaje de programación Python, por su gran ecosistema de librerías para visión por computadora y deep learning. El flujo de trabajo se ha dividido en diferentes etapas, desarrolladas principalmente en entornos Jupyter Notebook y scripts de Python:

- **Jupyter Notebook:** Se utilizó para entrenar el modelo de detección de objetos YOLO, aprovechando la facilidad para experimentar de forma interactiva con parámetros y visualizar resultados.
- **Python (scripts):** Fue empleado para el entrenamiento del modelo de reidentificación (ReID), así como para aplicar todos los modelos en el flujo final de inferencia y generación de vídeos etiquetados con las predicciones.

6.2 Gestión de datos y flujo de trabajo

Roboflow ha sido una herramienta fundamental para gestionar el dataset de entrenamiento del detector YOLO. Roboflow permite subir imágenes, etiquetarlas, realizar aumentos de datos automáticamente y exportar el dataset en el formato deseado (en nuestro caso, YOLOv8).

GitHub ha sido el sistema elegido para el control de versiones y colaboración. Se ha utilizado para organizar y almacenar los diferentes módulos del proyecto, así como mantener un historial ordenado de los avances y pruebas realizadas durante el desarrollo del TFG.

6.3 Modelos utilizados

YOLO (You Only Look Once) ha sido el modelo principal utilizado para la detección de jugadores. Su arquitectura permite realizar detección en tiempo real, con una alta precisión y bajo coste computacional. Se entrenó una versión ligera del modelo con imágenes de fútbol amateur obtenidas del dataset gestionado en Roboflow.

DeepEIoU fue el algoritmo de tracking utilizado para asociar detecciones entre frames y mantener la identidad de cada jugador. DeepEIoU combina estrategias geométricas (IoU) y de apariencia (cuando se usa ReID), lo que mejora la robustez frente a occlusiones y cambios de plano.

Este modelo ha sido clave para generar archivos de anotación con los ID temporales de los jugadores a lo largo del vídeo, y servir como base para el post-procesamiento posterior mediante GTA-Link.

TorchReID se ha usado como framework para entrenar el modelo de re-identificación. Se utilizó el modelo *osnet_x1_0*, que fue ajustado sobre imágenes recortadas de jugadores, organizadas en carpetas según su identidad. Este modelo se utilizó posteriormente para mejorar el seguimiento, reduciendo los cambios de identidad.

7 Módulo de detección (YOLO)

<https://arxiv.org/abs/1506.02640>

7.1 Arquitectura YOLO

YOLO (You Only Look Once) es una arquitectura de detección de objetos en imágenes y vídeos que ha destacado por su velocidad y precisión. La arquitectura original de YOLO, según la propuesta en el artículo de Redmon et al. (2016)¹, se inspira parcialmente en GoogleNet y está compuesta por 24 capas convolucionales, cuatro capas de agrupamiento máximo (max pooling) y dos capas totalmente conectadas al final.

Antes de procesarse, la imagen de entrada es redimensionada a un tamaño fijo de 448x448 píxeles. El flujo interno de la red emplea convoluciones de tipo 1x1 que tienen como función principal reducir el número de canales de activación intermedios sin afectar a la información espacial. Posteriormente, se aplican convoluciones de

¹Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. arXiv preprint arXiv:1506.02640.

3×3 para capturar patrones locales más complejos. Cada una de estas capas es seguida por una función de activación ReLU, que introduce no linealidad al modelo, excepto en la capa de salida, donde se emplea una función de activación lineal.

Además, para mejorar la estabilidad del entrenamiento y evitar el sobreajuste, se aplican técnicas como la normalización por lotes (Batch Normalization) y el abandono (Dropout). Estas estrategias permiten que el modelo generalice mejor ante nuevos datos y se mantenga robusto frente a variaciones del entorno.

Esta arquitectura permite que YOLO realice detección de múltiples objetos en una sola pasada sobre la imagen (de ahí su nombre), lo que lo convierte en un modelo extremadamente eficiente para tareas en tiempo real.

7.2 ¿Cómo funciona la detección de objetos con YOLO?

Una vez entendida la arquitectura interna de YOLO, es importante comprender cómo se lleva a cabo el proceso de detección de objetos. Para ello, se puede imaginar el caso práctico de una aplicación que detecta jugadores y balones de fútbol a partir de una imagen de vídeo.

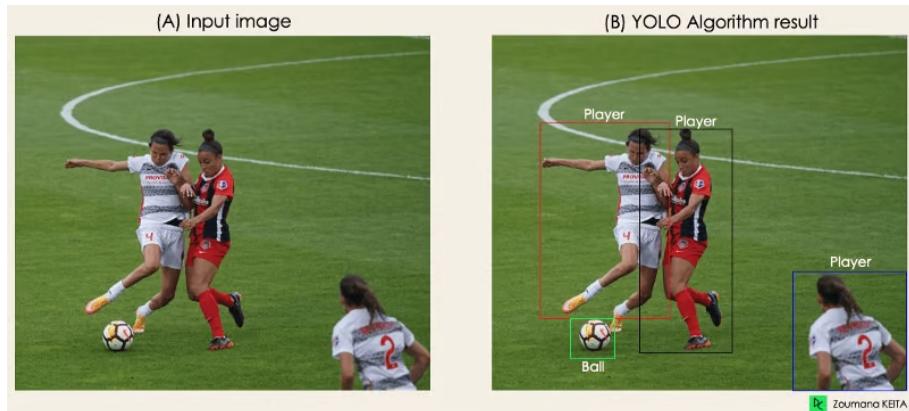


Figure 6: **YOLO Detection. First Step**

El algoritmo de detección en YOLO se basa en cuatro pilares fundamentales: la división en cuadrículas, la regresión de cajas delimitadoras (bounding boxes), el cálculo del coeficiente de intersección sobre unión (IoU), y la supresión no máxima (NMS).

1. **División en cuadrículas (bloques residuales):** La imagen de entrada se divide en una cuadrícula de tamaño $N \times N$, donde cada celda es responsable de detectar los objetos que contiene. Cada celda predice una clase y una probabilidad asociada, además de parámetros que definen la caja delimitadora.

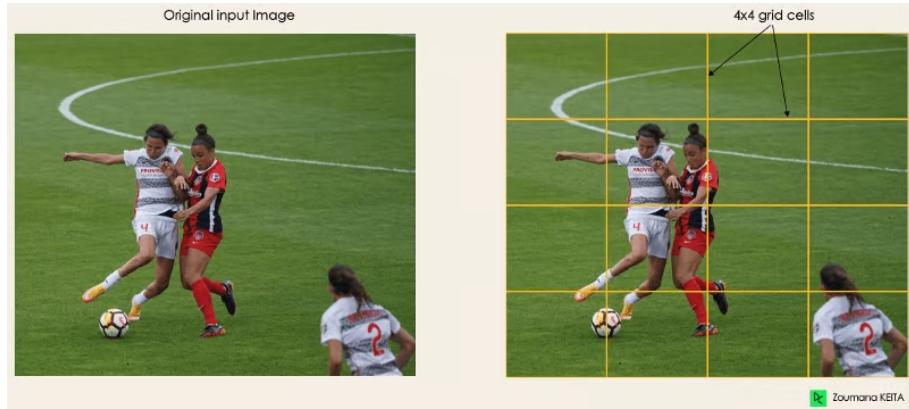


Figure 7: **YOLO Detection. Second Step**

2. **Regresión de cajas delimitadoras:** Cada celda predice varias cajas en el formato $Y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_n]$, donde:

- p_c es la probabilidad de que la celda contenga un objeto.

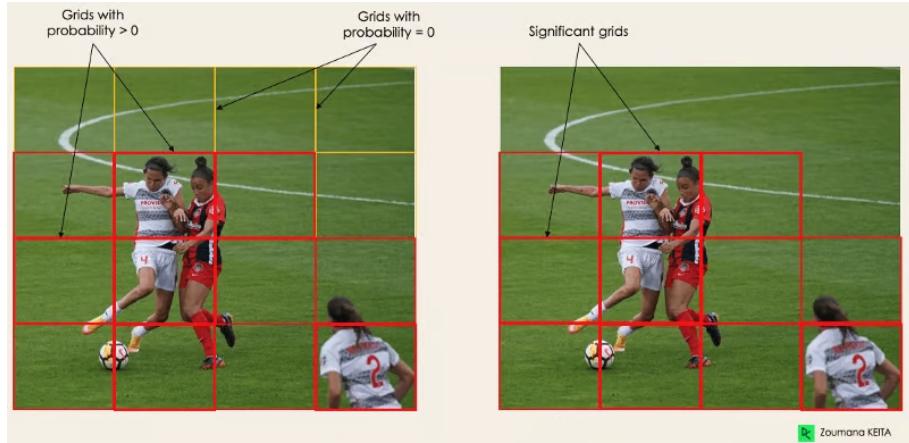


Figure 8: **YOLO Detection. Third Step**

- b_x, b_y son las coordenadas del centro de la caja respecto a la celda.
- b_h, b_w son la altura y la anchura de la caja.
- c_1, c_2, \dots son las probabilidades de pertenencia a cada clase (jugador, balón, etc.).

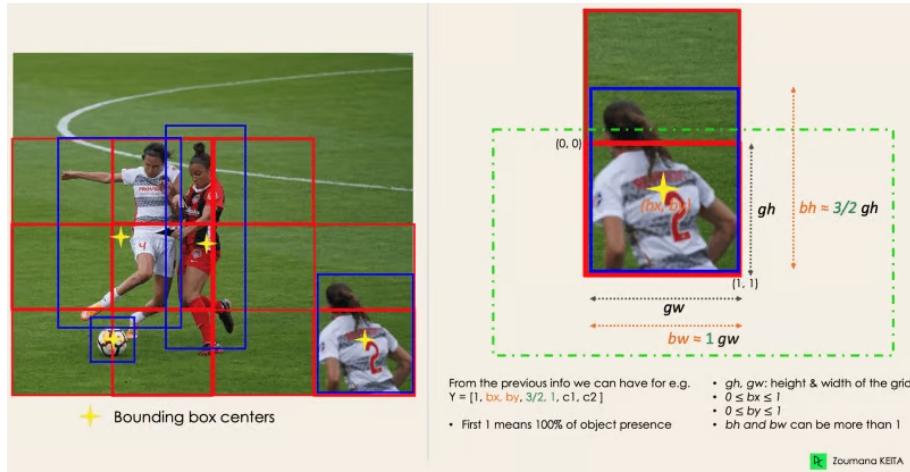


Figure 9: **YOLO Detection. Fourth Step**

3. **Intersección sobre Unión (IoU):** Para cada predicción, YOLO calcula el solapamiento entre la caja predicha y las cajas reales (ground truth). El valor del IoU (entre 0 y 1) permite descartar predicciones de baja calidad. Sólo aquellas con un IoU superior a un umbral son tenidas en cuenta.

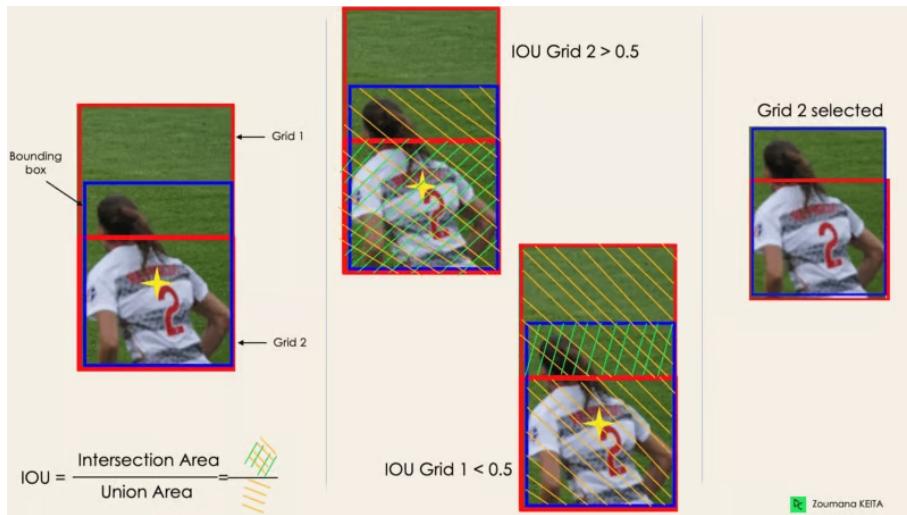


Figure 10: **YOLO Detection. Fifth Step**

4. **Supresión No Máxima (NMS):** Cuando varias cajas se solapan con valores altos de IoU, el algoritmo aplica NMS para conservar sólo la de mayor probabilidad, eliminando duplicados y reduciendo el ruido en las predicciones.

Este proceso completo permite que, en un único paso y de forma muy eficiente, YOLO identifique múltiples objetos en una imagen, indicando su clase y ubicación precisa. Esta capacidad lo convierte en una herramienta especialmente útil en el análisis de vídeo deportivo, donde se requiere detectar varios jugadores en escenas rápidas y dinámicas. "<https://www.datacamp.com/blog/yolo-object-detection-explained>"

8 Módulo de tracking + Gestión de IDs

8.1 Módulo de tracking

En el contexto de la detección de objetos en vídeo, como la realizada por modelos como YOLO, cada fotograma es procesado de forma independiente. Esto significa que, aunque el detector puede identificar y localizar objetos en cada imagen, no mantiene información sobre la identidad de estos objetos a lo largo del tiempo. Por ejemplo, no puede determinar si el jugador detectado en el fotograma 1 es el mismo que en el fotograma 2.

Para abordar esta limitación, se emplean algoritmos de seguimiento de objetos (trackers) que asocian las detecciones entre fotogramas consecutivos, permitiendo así mantener la identidad de cada objeto a lo largo del tiempo. Estos trackers asignan un identificador único a cada objeto y actualizan su posición en cada nuevo fotograma, incluso en presencia de oclusiones o cambios de apariencia.

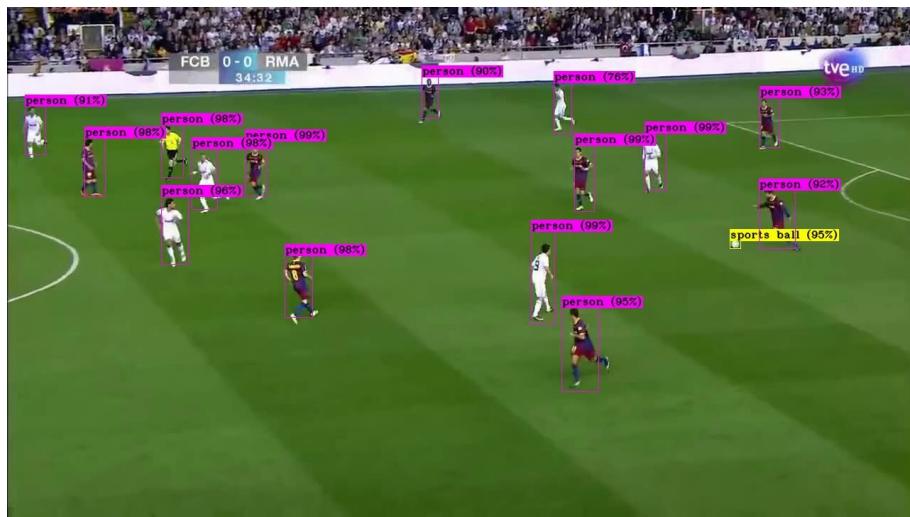


Figure 11: Tracking

Un ejemplo destacado de este enfoque es el algoritmo Deep SORT (Simple Online and Realtime Tracking with a Deep Association Metric), que combina información de movimiento y apariencia para asociar detecciones entre fotogramas. Deep SORT utiliza un filtro de Kalman para predecir la posición futura de los objetos y un descriptor de apariencia basado en redes neuronales profundas para comparar similitudes visuales entre detecciones. La asociación entre detecciones y trayectorias se realiza mediante el algoritmo húngaro, que minimiza el coste total de asociación considerando tanto la distancia de movimiento como la similitud de apariencia.
"<https://doi.org/10.1109/ICIP.2017.8296962>"

8.2 Gestión de IDs

"<https://paperswithcode.com/task/person-re-identification>"

La Re-identificación de Personas es una tarea de visión por computadora cuyo objetivo es asociar la identidad de una persona a través de diferentes cámaras o ubicaciones en una secuencia de vídeo o imágenes. Implica detectar y seguir a una persona y luego utilizar características como apariencia, forma del cuerpo y

ropa para asociar su identidad en diferentes fotogramas. El objetivo es asociar a la misma persona en múltiples vistas de cámara no superpuestas de manera robusta y eficiente.



Figure 12: **Person Re-Identification**

9 Propuesta

El flujo de trabajo propuesto tiene como objetivo optimizar el análisis de videos deportivos mediante un proceso de detección, seguimiento y reidentificación de jugadores, utilizando técnicas avanzadas de visión por computadora. El propósito es automatizar y mejorar la precisión en la identificación de jugadores a lo largo del video, proporcionando un sistema que permita analizar el comportamiento de cada jugador durante el transcurso del partido.

El resultado esperado de este flujo de trabajo es un archivo etiquetado que permita identificar a cada jugador con un seguimiento consistente a lo largo de las secuencias del video que utilizamos como input, lo que se traduce en un análisis más eficaz y exhaustivo. Este sistema facilitaría la recopilación de datos clave sobre el rendimiento individual de los jugadores, como su posición, movimientos y comportamientos durante el partido, ayudando a cerrar la brecha entre el análisis del fútbol profesional y el amateur.

Al final del proceso, se espera obtener una salida en formato de video con los jugadores anotados en el mismo y otra en formato MOT que contenga las trayectorias y las identificaciones de los jugadores, lo cual puede ser utilizado para estudios adicionales, análisis tácticos y optimización del rendimiento. Este flujo de trabajo proporciona la base para un sistema más completo y adaptable, capaz de integrarse con otras herramientas y métodos de análisis deportivos.

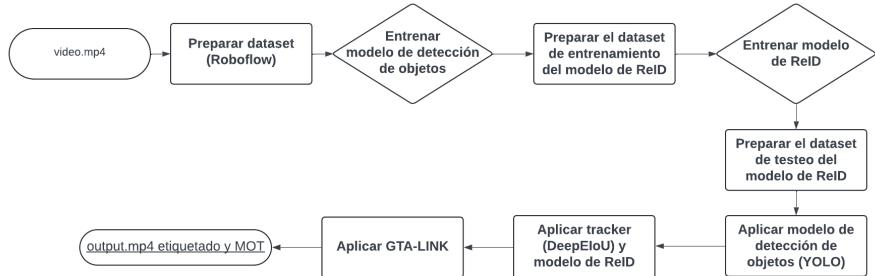


Figure 13: Propuesta

10 Arquitectura general del sistema

A continuación, se explica paso a paso el flujo completo, desde la entrada de vídeo original hasta la salida final con el vídeo anotado y anotaciones generadas. Cada módulo del sistema se presenta de forma independiente, especificando su función dentro del flujo, la tecnología empleada y los beneficios que aporta al conjunto. 13

10.1 Preparar el dataset (Roboflow)

<https://roboflow.com/>

Para entrenar el modelo de detección de jugadores, utilizamos la plataforma Roboflow, una herramienta online ampliamente utilizada para gestionar datasets de visión por computador. Roboflow facilita la carga, etiquetado, preprocesado y exportación de conjuntos de datos en formatos compatibles con múltiples frameworks de deep learning, incluyendo YOLO.

En nuestro caso, el proceso comenzó con la recopilación de fotogramas extraídos de vídeos de partidos de fútbol amateur. Estos frames fueron subidos a Roboflow, donde se realizó el etiquetado manual de los jugadores. Cada jugador fue anotado mediante la creación de una *bounding box* alrededor de su figura, y se le asignó la clase **jugador**.

Aunque en este proyecto únicamente se ha utilizado una clase (**jugador**), Roboflow permite una gestión flexible de múltiples clases. En el caso de que se hubiera querido incluir también la detección de otros roles como **árbitro** o **balón**, bastaría con asignar un nuevo nombre de clase a las anotaciones correspondientes durante el etiquetado. Cada vez que se etiqueta un objeto en un fotograma, se puede seleccionar (o crear) la clase deseada desde el menú desplegable de etiquetas.

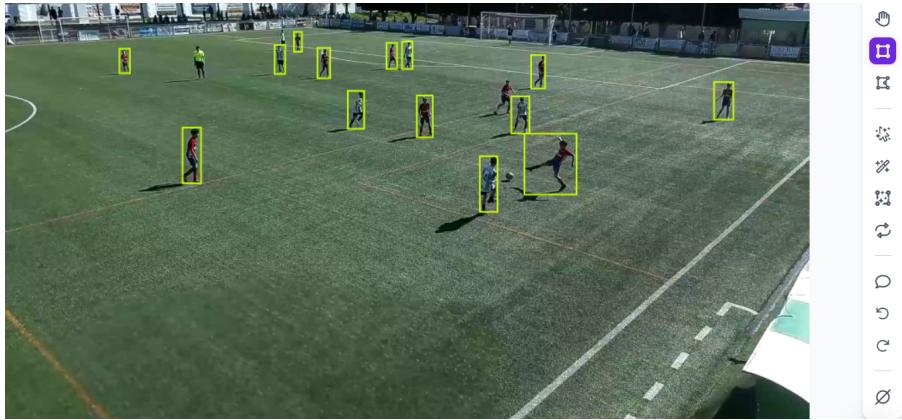


Figure 14: Diferenciación de clases

De este modo, el dataset resultante puede incluir múltiples clases, y el modelo YOLO aprenderá a diferenciarlas durante el entrenamiento, basándose en sus apariencias y posiciones relativas. Esta capacidad de distinguir entre diferentes tipos de objetos resulta esencial en contextos deportivos donde intervienen múltiples agentes.

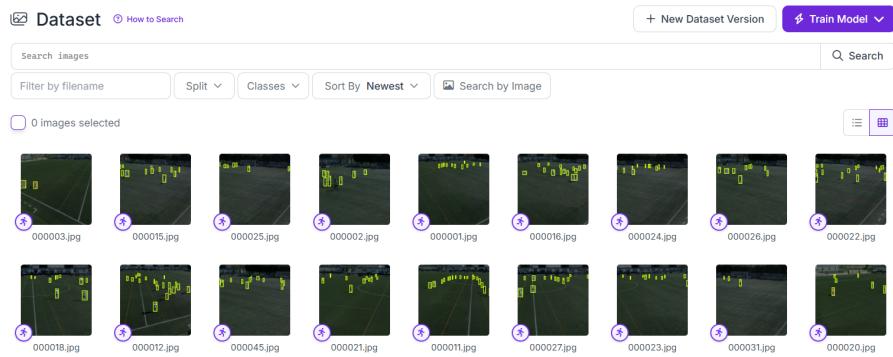


Figure 15: Entrenamiento YOLO

Una vez completado el etiquetado, el dataset fue exportado desde Roboflow en el formato YOLOv5/YOLOv8, incluyendo las anotaciones en archivos de texto por imagen, y un archivo `data.yaml` que especifica la configuración general del conjunto de datos, como las rutas, el número de clases y sus nombres.

10.2 Entrenamiento del modelo de YOLO

Una vez preparado el conjunto de datos en Roboflow y exportado en formato compatible con YOLOv8, se procedió al entrenamiento del modelo utilizando el framework **Ultralytics**, que proporciona una implementación optimizada y fácil de usar de los modelos YOLO.

El entrenamiento se realizó en Python, comenzando por la descarga del dataset directamente desde la API de Roboflow. Para ello, se utilizó el siguiente fragmento de código:

```
from roboflow import Roboflow
rf = Roboflow(api_key="s0NwH1D0Zcp6DurtT8Nw")
project = rf.workspace("pruebabalonmano").project("handball-dataset-def-2")
```

```

version = project.version(1)
dataset = version.download("yolov8")

```

Este código descarga automáticamente las imágenes y las anotaciones en formato YOLOv8, incluyendo el archivo `data.yaml` con la configuración del dataset.

A continuación, se cargó un modelo base preentrenado `yolov8n.pt` (versión ligera del modelo), y se inició el entrenamiento personalizado con los parámetros definidos:

```

from ultralytics import YOLO

model = YOLO("yolov8n.pt")  # Modelo base preentrenado
results = model.train(
    data=f"{dataset.location}/data.yaml",
    epochs=25,
    imgsz=640,
    batch=16,
    name="yolov8_handball"
)

```

En este caso, se entrenó durante 25 épocas, con imágenes de tamaño 640×640 píxeles y un tamaño de lote (`batch size`) de 16 imágenes. El modelo aprende a identificar la clase `jugador` a partir de los ejemplos anotados.

Una vez finalizado el entrenamiento, se genera automáticamente el archivo `best.pt`, que contiene los pesos del modelo con mejor rendimiento en validación. Este modelo final puede utilizarse directamente para la inferencia:

```
best_model = YOLO("runs/detect/yolov8_handball/weights/best.pt")
```

Este flujo de trabajo permite adaptar el modelo YOLOv8 a un dominio concreto, como el fútbol amateur, utilizando anotaciones específicas y obteniendo así un detector más ajustado al contexto real de aplicación.

10.3 Preparar el dataset de entrenamiento del modelo de ReID

El manejo de errores relacionados con la asignación incorrecta de IDs es un paso crucial en el proceso de reidentificación (ReID). Para resolver este problema, se utilizó un enfoque interactivo en el que se corrigen manualmente los errores de asignación de IDs en las anotaciones generadas por el modelo de ReID previamente entrenado con el dataset de SportMOT, que contiene imágenes de fútbol profesional.

10.3.1 annotate_review.py

En primer lugar, las anotaciones de un partido de fútbol amateur fueron generadas utilizando el modelo de ReID entrenado con el dataset de SportMOT. Posteriormente, estas anotaciones fueron procesadas para corregir posibles errores de asignación de IDs. El código utilizado para este proceso interactivo es el siguiente:

```

import os
import cv2
import argparse
import tkinter as tk
from tkinter import simpledialog
from collections import Counter

# Interactive MOT correction with flexible ID modification and distinct duplicate

def parse_mot_file(mot_path):
    entries = []
    with open(mot_path, 'r') as f:
        for line in f:
            if not line.strip(): continue
            parts = line.strip().split(',')
            entries.append({
                'frame': int(float(parts[0])),
                'id': int(float(parts[1])),
                'bbox': tuple(map(float, parts[2:6])),
                'raw': line.strip()
            })
    return entries

def draw_annotations(img, detections, global_highlight=None, raw_color_map=None):
    out = img.copy()
    global_h = set(global_highlight or [])
    for det in detections:
        x, y, w, h = map(int, det['bbox'])
        tid = det['id']
        raw = det['raw']
        if raw_color_map and raw in raw_color_map:
            color = raw_color_map[raw]
        elif tid in global_h:
            color = (0, 0, 255)
        else:
            color = (0, 255, 0)
        cv2.rectangle(out, (x, y), (x + w, y + h), color, 2)
        cv2.putText(out, str(tid), (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    return out

```

Este código permite corregir las asignaciones incorrectas de IDs mediante una interfaz gráfica que permite al usuario modificar o eliminar detecciones, asignar nuevos IDs a instancias específicas, y aprobar o eliminar duplicados.

Interacción del usuario: La interacción con el programa es completamente visual e intuitiva. Al ejecutar el código, el usuario puede visualizar las detecciones en cada fotograma del video y recibir la opción de:

- Navegar entre los fotogramas (siguiente y anterior):

```

if key == ord('n') and i < len(frames) - 1:
    i += 1
elif key == ord('p') and i > 0:
    i -= 1

```

- Modificar o eliminar instancias específicas de detecciones con IDs incorrectos:

```

if key in (ord('m'), ord('d')):
    root = tk.Tk(); root.withdraw()
    prompt = "Introduzca 'i'<indice> para instancia o  

              ID para global:"
    sel = simpledialog.askstring("Seleccionar deteccion  
",
                                 prompt)
    root.destroy()
    if not sel: continue
    if sel.startswith('i'):
        try:
            idx = int(sel[1:]) - 1
            chosen = duplicate_entries[idx]
        except:
            print("Seleccion invalida")
            continue
        if key == ord('m'):
            new_id = simpledialog.askinteger("Nuevo ID", "Nuevo  
ID para la instancia seleccionada:")
            if new_id is not None:
                raw_map[chosen['raw']] = new_id
                print(f"Instancia bbox {chosen['bbox']} mapeada a {  
                  new_id}")
            else:
                removed.add(chosen['raw'])
                print(f"Instancia bbox {chosen['bbox']} eliminada")
        else:
            try:
                orig = int(sel)
            except:
                print("ID invalido.")
                continue
            if key == ord('m'):
                new = simpledialog.askinteger("Nuevo ID", f"Nuevo  
ID para todas detecciones {orig}:")
                if new is not None:
                    id_map[orig] = new
                    print(f"Mapeado global ID {orig} -> {new}")
                else:
                    for d in by_frame[orig_frame]:
                        cur = raw_map.get(d['raw'], id_map.get(d['id'], d['  
id'])))
                        if cur == orig:
                            removed.add(d['raw'])
                            print(f"Todas detecciones ID {orig} eliminadas en")

```

```
frame {orig_frame})
```

- Corregir duplicados, donde se le presenta un mensaje con las IDs duplicadas detectadas en el fotograma:

```
if duplicate_entries:  
    dup_msgs = []  
    dup_ids = []  
    for idx_det, det in enumerate(duplicate_entries, 1)  
        :  
        det_id = raw_map.get(det['raw'], id_map.get(det['id'],  
            , det['id']))  
        name = color_names[(idx_det - 1) % len(color_names)]  
        ]  
        dup_msgs.append(f"{idx_det} ({name}) id {det_id}")  
        dup_ids.append(det_id)  
    print("Duplicados: " + ", ".join(dup_msgs) + f" en  
        frame {orig_frame}")
```

- Aceptar los cambios realizados y continuar con la corrección de otras detecciones:

```
elif key == ord('c'):  
    if duplicate_entries:  
        approved_dup_ids.update(dup_ids)  
    break
```

El programa recorre automáticamente los fotogramas del video y, al detectar un nuevo ID, muestra una serie de opciones al usuario. Estos nuevos IDs se resaltan en rojo, lo que facilita su identificación como detecciones recién encontradas. Una vez que el usuario aprueba el ID, este se mantiene en verde, indicando que ha sido aprobado y no se volverá a revisar en el futuro. 16

Este programa permite arreglar problemas graves de asignación de ids. Por ejemplo, que un jugador tenga un nuevo id durante el resto de la secuencia.



Figure 16: Programa para el entrenamiento

10.3.2 annotate_review_precise.py

El programa `annotate_review_precise.py` es una versión complementaria del programa anterior `annotate_review.py`.

El código utilizado en este programa es el siguiente:

```
import os
import cv2
import argparse
import tkinter as tk
from tkinter import simpledialog
from collections import Counter

# Interactive MOT correction with manual frame control,
# per-frame ID modifications, approval colors, and undo

def parse_mot_file(mot_path):
    entries = []
    with open(mot_path, 'r') as f:
        for line in f:
            if not line.strip(): continue
            parts = line.strip().split(',')
            entries.append({
                'frame': int(float(parts[0])), # frame number
                'id': int(float(parts[1])), # original ID
                'bbox': tuple(map(float, parts[2:6])), # bbox
                'raw': line.strip() # raw line
            })
    return entries

def draw_annotations(img, detections,
    new_ids, approved_new_ids,
    duplicate_entries, approved_dup_ids,
    raw_color_map=None, raw_index_map=None):
    out = img.copy()
    for det in detections:
        x, y, w, h = map(int, det['bbox'])
        tid = det['id']
        raw = det['raw']
        # determine color
        if raw_color_map and raw in raw_color_map:
            color = raw_color_map[raw]
        elif tid in approved_new_ids or tid in approved_dup_ids:
            color = (0, 255, 0) # green approved
        elif tid in new_ids or any(d['id'] == tid for d in
            duplicate_entries):
            color = (0, 0, 255) # red pending
        else:
            color = (0, 255, 0)
        # draw rectangle
```

```

cv2.rectangle(out, (x, y), (x + w, y + h), color, 2)
# prepare label
label = str(tid)
if raw_index_map and raw in raw_index_map:
    label += f"(i{raw_index_map[raw]} + 1)"
# put text
cv2.putText(out, label, (x, y - 5), cv2.
    FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
return out

def get_mapped_id(frame, det,
    id_map, raw_map,
    frame_id_map, frame_raw_map):
    # per-frame raw map
    if frame in frame_raw_map and det['raw'] in frame_raw_map[frame]:
        return frame_raw_map[frame][det['raw']]
    # per-frame id map
    if frame in frame_id_map and det['id'] in frame_id_map[frame]:
        return frame_id_map[frame][det['id']]
    # fallback global maps
    return raw_map.get(det['raw'], id_map.get(det['id'], det['id']))


def pause_loop(by_frame, frames,
    id_map, raw_map,
    frame_id_map, frame_raw_map,
    removed, frames_dir,
    new_ids, duplicate_entries,
    approved_new_ids, approved_dup_ids,
    start_i):
    i = start_i
    orig_frame = frames[i]
    # color palette for duplicates
    palette = [(255, 0, 0), (255, 255, 0), (255, 0, 255), (0,
        255, 255), (128, 0, 255)]
    raw_color_map = {d['raw']: palette[idx % len(palette)]
        for idx, d in enumerate(duplicate_entries)}
    raw_index_map = {d['raw']: idx for idx, d in enumerate(
        duplicate_entries)}

    # display duplicate info with indices
    if duplicate_entries:
        dup_info = ', '.join(f"i{idx+1}:{d['id']}" for idx, d in
            enumerate(duplicate_entries))
        print(f"Duplicados en frame {orig_frame}: {dup_info}")
    elif new_ids:
        print(f"Nuevos IDs en frame {orig_frame}: {new_ids}")

```

```

print("Navegacion: [n]/[c]=next , [p]=prev , [u]=undo , [m]=
      modificar , [d]=delete , [q]=quit")

while True:
    f2 = frames[i]
    dets2 = [d for d in by_frame[f2] if d['raw'] not in
             removed]
    mapped2 = []
    for d in dets2:
        mid = get_mapped_id(f2, d, id_map, raw_map, frame_id_map,
                             frame_raw_map)
        mapped2.append({'bbox': d['bbox'], 'id': mid, 'raw': d['
            raw']})
    img2 = cv2.imread(os.path.join(frames_dir, f"{f2:06d}.jpg
        "))
    cv2.imshow('Annotations', draw_annotations(
        img2, mapped2,
        new_ids, approved_new_ids,
        duplicate_entries, approved_dup_ids,
        raw_color_map if f2 == orig_frame else None,
        raw_index_map if f2 == orig_frame else None
    ))

    key = cv2.waitKey(0) & 0xFF
    if key in (ord('n'), ord('c')):
        approved_new_ids.update(new_ids)
        approved_dup_ids.update(d['id'] for d in
            duplicate_entries)
    return min(i + 1, len(frames) - 1), False
    elif key == ord('p'):
        return max(i - 1, 0), False
    elif key == ord('u'):
        return max(i - 1, 0), True
    elif key in (ord('m'), ord('d')):
        root = tk.Tk(); root.withdraw()
        sel = simpledialog.askstring("Seleccionar deteccion", "i
            <numero> o ID:")
        root.destroy()
        if not sel: continue
        # instance-level
        if sel.startswith('i'):
            try:
                idx = int(sel[1:]) - 1
                chosen = duplicate_entries[idx]
            except:
                print("Seleccion invalida"); continue
            if key == ord('m'):
                new_id = simpledialog.askinteger("Nuevo ID", "ID para
                    esta instancia:")
                if new_id is not None:
                    frame_raw_map.setdefault(orig_frame, {})[chosen['raw']] =

```

```

        new_id
print(f"Instancia raw={chosen['raw']} -> {new_id} en
      frame {orig_frame}")
else:
removed.add(chosen['raw'])
print("Instancia eliminada")
# global-level
else:
try:
orig = int(sel)
except:
print("ID invalido"); continue
if key == ord('m'):
new = simpledialog.askinteger("Nuevo ID", f"ID {orig} ->
    ? en frame {orig_frame}:")
if new is not None:
frame_id_map.setdefault(orig_frame, {})[orig] = new
print(f"ID {orig} -> {new} en frame {orig_frame}")
else:
for d in by_frame[orig_frame]:
cur = get_mapped_id(orig_frame, d, id_map, raw_map,
    frame_id_map, frame_raw_map)
if cur == orig:
removed.add(d['raw'])
print(f"Instancias ID {orig} eliminadas en frame {
    orig_frame}")
continue
elif key == ord('q'):
return len(frames), False
else:
continue

```

Este programa introduce varias mejoras y características adicionales respecto a la versión anterior:

- **Deshacer cambios:** La opción de deshacer ([u]) permite revertir los cambios realizados en el fotograma actual, añadiendo flexibilidad al proceso de corrección.
- **Visualización de duplicados mejorada:** Los duplicados se visualizan con colores específicos y se puede asignar un nuevo ID o eliminarlos en función de la decisión del usuario.
- **Control manual:** Se cambia la manera de recorrer los frames automáticamente a manualmente. Cada vez que se apruebe un frame se pasa al siguiente.

Este enfoque permite una corrección más precisa y segura, evitando acarrear ids erróneos.

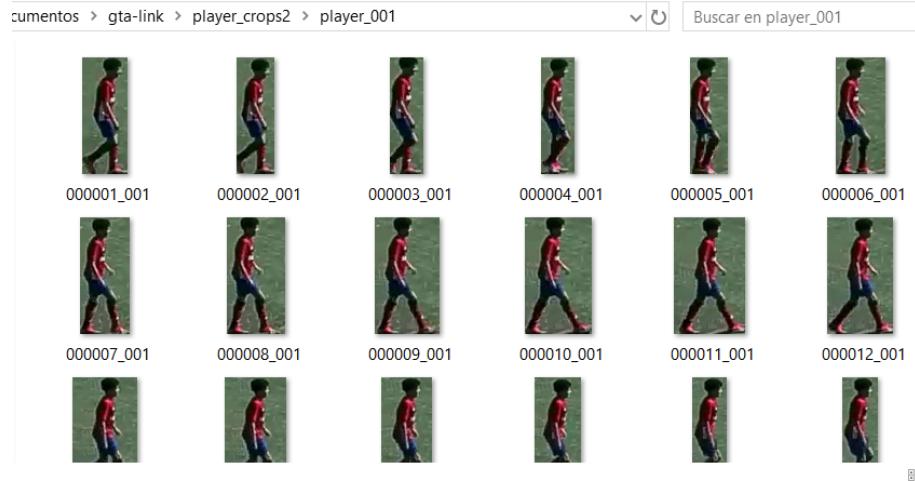


Figure 17: Estructura del dataset

10.3.3 dicMaker_idPlayer.py

El programa `dicMaker_idPlayer.py` se utiliza para procesar un archivo MOT y generar recortes de imágenes de los jugadores en cada fotograma, organizados por ID de jugador. El archivo MOT contiene las anotaciones de las detecciones para cada fotograma, y el objetivo de este programa es generar recortes individuales de cada jugador basado en esas anotaciones.

El proceso se realiza en dos etapas principales:

1. **Lectura y análisis del archivo MOT:** El programa lee el archivo MOT que contiene las anotaciones de cada detección (frame, ID de jugador, coordenadas de la caja delimitadora) y organiza las detecciones por fotograma.

2. **Recorte y guardado de las imágenes de los jugadores:** Para cada fotograma y cada jugador identificado en las detecciones, el programa recorta la región de la imagen correspondiente a las coordenadas de la caja delimitadora y guarda cada recorte en una carpeta organizada por ID de jugador.

El código utilizado en este proceso es el siguiente:

```

import os
import cv2
import argparse
from tqdm import tqdm

def parse_mot_file(mot_file_path):
    detections = []
    with open(mot_file_path, 'r') as f:
        for line in f:
            parts = line.strip().split(',')
            if len(parts) < 6:
                continue

            frame_id = int(float(parts[0]))
            track_id = int(float(parts[1]))
            x, y, w, h = map(float, parts[2:6])
    
```

```

    if frame_id not in detections:
        detections[frame_id] = []
        detections[frame_id].append((track_id, int(x), int(y),
                                      int(w), int(h)))

    return detections

def crop_and_save_detections(image_folder, detections,
                             output_folder):
    os.makedirs(output_folder, exist_ok=True)

    for frame_id in tqdm(sorted(detections.keys()), desc="Procesando frames"):
        img_path = os.path.join(image_folder, f"{frame_id:06d}.jpg")

        if not os.path.exists(img_path):
            continue

        img = cv2.imread(img_path)
        if img is None:
            continue

        for track_id, x, y, w, h in detections[frame_id]:
            track_folder = os.path.join(output_folder, f"player_{track_id:03d}")
            os.makedirs(track_folder, exist_ok=True)

            x1 = max(0, x)
            y1 = max(0, y)
            x2 = min(img.shape[1], x + w)
            y2 = min(img.shape[0], y + h)

            if x2 <= x1 or y2 <= y1:
                continue

            crop = img[y1:y2, x1:x2]

            # Cambiado el nombre del archivo al formato deseado:
            # frameId_trackId.jpg
            output_filename = f"{frame_id:06d}_{track_id:03d}.jpg"
            output_path = os.path.join(track_folder, output_filename)
            cv2.imwrite(output_path, crop)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--mot_file', required=True)
    parser.add_argument('--image_folder', required=True)
    parser.add_argument('--output_folder', required=True)

    args = parser.parse_args()

```

```

print("Leyendo archivo MOT...")
detections = parse_mot_file(args mot_file)

print("\nRecortando detecciones...")
crop_and_save_detections(args.image_folder, detections,
    args.output_folder)

print(f"\nProceso completado. Recortes en: {args.
    output_folder}")

if __name__ == "__main__":
    main()

```

El programa realiza las siguientes funciones:

- **Parseo del archivo MOT:** El archivo MOT se procesa para organizar las detecciones por fotograma y por jugador. Cada detección incluye las coordenadas de la caja delimitadora y el ID de jugador.
- **Recorte de las imágenes:** Para cada fotograma y cada jugador detectado, se recorta la imagen según las coordenadas de la caja delimitadora. Los recortes se guardan en una estructura de carpetas donde cada jugador tiene su propia carpeta identificada por su `track_id`.
- **Uso de la barra de progreso:** Se utiliza la librería `tqdm` para mostrar una barra de progreso durante el procesamiento de los fotogramas, facilitando el seguimiento del proceso.

10.3.4 dicMakerForTrainingReID.py

El programa `dicMakerForTrainingReID.py` se usa para reorganizar un conjunto de imágenes de jugadores en una estructura compatible con el formato Torchreid, utilizado para entrenar sistemas de Re-Identificación (ReID). El script divide las imágenes de cada jugador en tres conjuntos distintos: entrenamiento (train), consulta (query) y galería (gallery).

El proceso se realiza en dos etapas principales:

Preparación de la estructura de directorios: Crea las carpetas necesarias para los tres conjuntos de datos en el directorio de destino.

Distribución de imágenes: Para cada jugador, divide sus imágenes disponibles en los tres conjuntos según las proporciones especificadas, asegurando un mínimo de imágenes por jugador.

El código utilizado en este proceso es el siguiente:

```

import os
import shutil
import random
from pathlib import Path

# Parámetros

```

```

source_dir = Path(r"C:\Users\jismbs\Documents\gta-link\
    player_crops2")
target_dir = Path(r"C:\Users\jismbs\Documents\gta-link\
    player_crops2_reid")

train_ratio = 0.7
query_ratio = 0.15
gallery_ratio = 0.15

# suman 1.0
assert abs(train_ratio + query_ratio + gallery_ratio -
    1.0) < 1e-6

splits = ['train', 'query', 'gallery']

for split in splits:
    (target_dir / split).mkdir(parents=True, exist_ok=True)

# Para cada jugador (identidad)
for player_folder in source_dir.iterdir():
    if not player_folder.is_dir():
        continue

    images = list(player_folder.glob("*.jpg"))
    if len(images) < 3:
        print(f"Saltando {player_folder.name}: no hay suficientes
            imágenes (minimo 3)")
        continue

    random.shuffle(images)
    n = len(images)
    n_train = int(n * train_ratio)
    n_query = int(n * query_ratio)

    split_counts = {
        'train': images[:n_train],
        'query': images[n_train:n_train + n_query],
        'gallery': images[n_train + n_query:]
    }

    for split, split_images in split_counts.items():
        split_target_dir = target_dir / split / player_folder.
            name
        split_target_dir.mkdir(parents=True, exist_ok=True)
        for img_path in split_images:
            shutil.copy(img_path, split_target_dir / img_path.name)

    print("Dataset reorganizado correctamente en formato
        Torchreid.")

```

El programa realiza las siguientes funciones:

- **División proporcional:** Distribuye automáticamente las imágenes de cada jugador en los conjuntos train (70)
- **Filtrado de jugadores:** Omite aquellos jugadores con menos de 3 imágenes, asegurando que cada identidad tenga suficientes muestras para ser útil en ReID.
- **Estructura compatible con Torchreid:** Crea una organización de directorios estándar que puede ser directamente utilizada por frameworks como Torchreid.
- **Barrido recursivo:** Detecta automáticamente todas las carpetas de jugadores en el directorio fuente y las procesa individualmente.

Consideraciones importantes:

- El script mantiene la estructura original de nombres de archivos y carpetas.
- La distribución de imágenes es aleatoria pero determinista (depende de la semilla aleatoria de Python).
- Se recomienda tener al menos 3 imágenes por jugador para que la división sea efectiva.

10.4 Entrenar el modelo de ReID

Para el entrenamiento del modelo de re-identificación (ReID), se utilizó el repositorio **Sportsreid**, que está basado en **SoccerNet Re-Identification** y **Torchreid**. Este repositorio permite reidentificar al mismo jugador en diferentes fotogramas de un video de un partido de fútbol. Sportsreid ha alcanzado el segundo puesto en el leaderboard de la división de prueba del reto SoccerNet 2022 de Re-Identification.

El entrenamiento se lleva a cabo utilizando un archivo de configuración que define los parámetros del modelo, el optimizador y el scheduler, así como el conjunto de datos a utilizar. A continuación se describe el proceso de entrenamiento utilizando el archivo `main.py` proporcionado en el repositorio.

El código de entrenamiento es el siguiente:

```

import sys
import time
import os.path as osp
import argparse
import torch
import torch.nn as nn

import torchreid
from torchreid.reid.utils import (
    Logger, check_isfile, set_random_seed, collect_env_info,
    resume_from_checkpoint, load_pretrained_weights,
    compute_model_complexity
)
from default_config import (

```

```

        imagedata_kwargs, optimizer_kwargs, videodata_kwargs,
        engine_run_kwargs,
        get_default_config, lr_scheduler_kwargs
    )

def build_datamanager(cfg):
    # ImageDataManager con batch sizes de train/test y
    # relabel de IDs
    return torchreid.data.ImageDataManager(
        root=cfg.data.root,
        sources=tuple(cfg.data.sources),
        targets=(),                                # sin target set
        separado
        height=cfg.data.height,
        width=cfg.data.width,
        batch_size_train=cfg.train.batch_size,
        batch_size_test=cfg.test.batch_size,
        transform_train=cfg.data.transforms,
        transform_test=cfg.data.transforms,
        combineall=cfg.data.combineall,
        relabel=True,                               # remapea IDs a [0..
        N-1]
        num_workers=cfg.data.workers,
        verbose=True,
        use_gpu=cfg.use_gpu
    )

def build_engine(cfg, datamanager, model, optimizer,
                 scheduler):
    if cfg.data.type == 'image':
        if cfg.loss.name == 'softmax':
            engine = torchreid.engine.ImageSoftmaxEngine(
                datamanager,
                model,
                optimizer=optimizer,
                scheduler=scheduler,
                use_gpu=cfg.use_gpu,
                label_smooth=cfg.loss.softmax.label_smooth
            )
        else:
            engine = torchreid.engine.ImageTripletEngine(
                datamanager,
                model,
                optimizer=optimizer,
                margin=cfg.loss.triplet.margin,
                weight_t=cfg.loss.triplet.weight_t,
                weight_x=cfg.loss.triplet.weight_x,
                weight_tc=cfg.loss.triplet.weight_tc,
                weight_cc=cfg.loss.triplet.weight_cc,
                scheduler=scheduler,
                use_gpu=cfg.use_gpu,

```

```

label_smooth=cfg.loss.softmax.label_smooth,
topk=cfg.loss.triplet.topk,
bottomk=cfg.loss.triplet.bottomk,
warmup_lr=cfg.train.warmup_lr,
warmup_steps=cfg.train.warmup_steps,
lr=cfg.train.lr
)
else:
if cfg.loss.name == 'softmax':
engine = torchreid.engine.VideoSoftmaxEngine(
datamanager,
model,
optimizer=optimizer,
scheduler=scheduler,
use_gpu=cfg.use_gpu,
label_smooth=cfg.loss.softmax.label_smooth,
pooling_method=cfg.video.pooling_method
)
else:
engine = torchreid.engine.VideoTripletEngine(
datamanager,
model,
optimizer=optimizer,
margin=cfg.loss.triplet.margin,
weight_t=cfg.loss.triplet.weight_t,
weight_x=cfg.loss.triplet.weight_x,
scheduler=scheduler,
use_gpu=cfg.use_gpu,
label_smooth=cfg.loss.softmax.label_smooth
)
return engine

def reset_config(cfg, args):
# Ruta por defecto a tu carpeta GTA-Link
cfg.data.root = args.root or r'C:\Users\jismbs\Documents\
    gta-link'
# Dataset de entrada por defecto
cfg.data.sources = args.sources or ['player_crops2']
# Transforms si se especifican
if args.transforms:
cfg.data.transforms = args.transforms

def check_cfg(cfg):
if cfg.loss.name == 'triplet' and cfg.loss.triplet.
    weight_x == 0:
assert cfg.train.fixbase_epoch == 0, \
'The output of classifier is not included in the
computational graph'

def main():
parser = argparse.ArgumentParser(

```

```

formatter_class=argparse.ArgumentDefaultsHelpFormatter
)
parser.add_argument(
    '--config-file', type=str, default='', help='path to
        config file'
)
parser.add_argument(
    '-s', '--sources', type=str, nargs='+',
    help='source datasets (delimited by space)'
)
parser.add_argument(
    '-t', '--targets', type=str, nargs='+',
    help='target datasets (delimited by space)'
)
parser.add_argument(
    '--transforms', type=str, nargs='+', help='data
        augmentation'
)
parser.add_argument(
    '--root', type=str, default='', help='path to data root'
)
parser.add_argument(
    'opts', default=None, nargs=argparse.REMAINDER,
    help='Modify config options using the command-line'
)
args = parser.parse_args()

cfg = get_default_config()
cfg.use_gpu = torch.cuda.is_available()
if args.config_file:
    cfg.merge_from_file(args.config_file)
    reset_config(cfg, args)
    cfg.merge_from_list(args.opts)
    set_random_seed(cfg.train.seed)
    check_cfg(cfg)

log_name = 'test.log' if cfg.test.evaluate else 'train.
    log'
log_name += time.strftime('-%Y-%m-%d-%H-%M-%S')
sys.stdout = Logger(osp.join(cfg.data.save_dir, log_name
    ))

print('Show configuration\n{}\n'.format(cfg))
print('Collecting env info ...')
print('* System info **\n{}\n'.format(collect_env_info()
    ))

if cfg.use_gpu:
    torch.backends.cudnn.benchmark = False # 
        reproducibilidad

```

```

datamanager = build_datamanager(cfg)

print('Building model: {}'.format(cfg.model.name))
model = torchreid.models.build_model(
    name=cfg.model.name,
    num_classes=datamanager.num_train_pids,
    loss=cfg.loss.name,
    pretrained=cfg.model.pretrained,
    use_gpu=cfg.use_gpu,
    img_size=(cfg.data.height, cfg.data.width),
)
num_params, flops = compute_model_complexity(
    model, (1, 3, cfg.data.height, cfg.data.width)
)
print('Model complexity: params={:,} flops={:,}'.format(
    num_params, flops))

if cfg.model.load_weights and check_isfile(cfg.model.
    load_weights):
    load_pretrained_weights(model, cfg.model.load_weights)

if cfg.use_gpu:
    model = nn.DataParallel(model).cuda()

optimizer = torchreid.optim.build_optimizer(model, **
    optimizer_kwargs(cfg))
scheduler = torchreid.optim.build_lr_scheduler(
    optimizer, **lr_scheduler_kwargs(cfg)
)

if cfg.model.resume and check_isfile(cfg.model.resume):
    cfg.train.start_epoch = resume_from_checkpoint(
        cfg.model.resume, model, optimizer=optimizer, scheduler=
        scheduler
)

print('Building {}-engine for {}-reid'.format(cfg.loss.
    name, cfg.data.type))
engine = build_engine(cfg, datamanager, model, optimizer,
    scheduler)
engine.run(**engine_run_kwargs(cfg))

if __name__ == '__main__':
    main()

```

Este código realiza las siguientes funciones clave:

- **Configuración del entorno de entrenamiento:** El código configura los parámetros del modelo, el optimizador y el scheduler para el entrenamiento del modelo de ReID utilizando la configuración especificada en el archivo YAML.
- **Gestión de datos:** Utiliza torchreid.data.ImageDataManager para ges-

tionar el conjunto de datos de entrada y salida, aplicando las transformaciones necesarias y ajustando los batch sizes para el entrenamiento y la evaluación.

- **Entrenamiento del modelo:** El modelo es entrenado utilizando diferentes arquitecturas de red, como ResNet50, OSNet, y ViT, con la opción de utilizar diferentes funciones de pérdida como `softmax` o `triplet`, dependiendo de la configuración elegida.
- **Gestión del proceso de entrenamiento:** El código proporciona un sistema de logging para registrar el progreso del entrenamiento y las métricas obtenidas, como la precisión media (mAP) y el rendimiento en el ranking-1.

Este proceso de entrenamiento es fundamental para ajustar el modelo de ReID a las características del conjunto de datos de fútbol amateur, permitiendo una mejor identificación de los jugadores en los distintos fotogramas del video.

10.5 Preparar el dataset de testeo del modelo de ReID

En este punto del flujo, lo que llevaremos a cabo es realizar las mismas anotaciones que para preparar el ‘dataset’ de entrenamiento del modelo ReID, pero esta vez aplicadas a otra secuencia de vídeo. El objetivo sería comprobar si el modelo ha aprendido correctamente las características de cada uno de los jugadores como para re-identificar al jugador en distintos fotogramas del nuevo vídeo.

El proceso sigue los siguientes pasos:

- **Selección de la secuencia de testeo:** Para establecer el conjunto de datos de prueba se elige una secuencia de vídeo que sea diferente a la del conjunto de datos de entrenamiento. Este conjunto de datos de prueba deberá ser representativo de las condiciones del entorno de aplicación real, de tal forma que las características visuales de la nueva secuencia de vídeo proporcionen un reto suficiente al modelo.
- **Anotación de las detecciones:** Dado que el procedimiento es análogo al anterior de la fase de entrenamiento, se realiza la anotación de las detecciones de los jugadores desde cada fotograma de la secuencia de test. Este procedimiento se puede llevar a cabo de la misma forma que el sistema de anotaciones del archivo MOT, indicando las coordenadas de las cajas delimitadoras y los IDs de los jugadores en el correspondiente fotograma.
- **Uso de un modelo preentrenado:** El modelo que se había entrenado previamente con el dataset de entrenamiento es el que se utiliza ahora para procesar la secuencia de test, el que debe ser capaz de re-identificar correctamente a los jugadores en cada fotograma de acuerdo con las características aprendidas en el entrenamiento.
- **Evaluación de la re-identificación:** Una vez que el modelo halla procesado la secuencia de testeo, se evalúa su capacidad de re-identificación observando a qué jugadores va asignando el mismo ID a lo largo del vídeo. El modelo se evalúa utilizando métricas estándar de re-identificación, tales como la precisión a top-1 y el mAP (Mean Average Precision).

Este proceso de preparación del dataset de testeo es fundamental para verificar si el modelo de ReID ha generalizado correctamente las características aprendidas durante el entrenamiento y si puede realizar una re-identificación exitosa en nuevas secuencias de vídeo, lo que es esencial para su despliegue en entornos reales.

10.6 Aplicar modelo de detección de objetos (YOLOX)

Para aplicar el modelo de detección de objetos YOLOX, se utilizó el código del repositorio DeepEIoU, específicamente el archivo `demo.py`, junto con una configuración personalizada definida en el archivo `yolox_x_ch_sportsmot.py`. El objetivo es detectar jugadores en los fotogramas de un vídeo utilizando el modelo YOLOX.

10.6.1 Configuración del Modelo YOLOX en `yolox_x_ch_sportsmot.py`

El archivo `yolox_x_ch_sportsmot.py` define los parámetros clave del modelo YOLOX, que son cruciales para la detección de objetos en los fotogramas. Los puntos clave de esta configuración son:

- **Número de Clases y Tamaño de Entrada:** El número de clases se establece en 1, ya que solo se está detectando la clase "jugador". El tamaño de la entrada de la imagen se define como (800, 1440), lo que permite trabajar con imágenes de alta resolución.

```
self.num_classes = 1
self.input_size = (800, 1440)
self.test_size = (800, 1440)
```

- **Parámetros de Evaluación:** Se especifican los umbrales de confianza (`test_conf`) y de NMS (`nmsthre`), que son cruciales para filtrar las detecciones de baja calidad y evitar duplicados en la misma imagen.

```
self.test_conf = 0.1
self.nmsthre = 0.7
```

- **Configuración de Datos y Transformaciones:** Se utiliza la clase `MOTDataset` para cargar el conjunto de datos de detección de objetos, con las transformaciones definidas en `TrainTransform` y `ValTransform` para preprocesar las imágenes de entrada.

```
dataset = MOTDataset(
    data_dir='/work/hsiangwei/dataset/',
    json_file=self.train_ann,
    name='',
    img_size=self.input_size,
    preproc=TrainTransform(
        rgb_means=(0.485, 0.456, 0.406),
        std=(0.229, 0.224, 0.225),
```

```

    max_labels=500,
),
)

```

10.6.2 Implementación en demo.py

El archivo `demo.py` aplica el modelo YOLOX para realizar la detección de objetos en una secuencia de vídeo. Los aspectos clave de su funcionamiento son:

- **Cargar el Modelo:** El modelo se carga utilizando la configuración definida en el archivo de experimentos, que especifica la arquitectura y los parámetros de entrenamiento. Se utiliza la clase `Predictor` para gestionar la inferencia.

```

model = exp.get_model().to(args.device)
model.eval()

```

- **Inferencia en Fotogramas de Vídeo:** En cada fotograma del vídeo, se realiza la detección de objetos utilizando el modelo YOLOX. El preprocesamiento de las imágenes se realiza con la función `preproc`, que ajusta el tamaño de las imágenes y normaliza los valores RGB.

```

img, ratio = preproc(img, self.test_size, self.rgb_means, self.std)
img = torch.from_numpy(img).unsqueeze(0).float().to(self.device)

```

- **Postprocesamiento de Detecciones:** Después de realizar la inferencia, se aplica el postprocesamiento para filtrar las detecciones utilizando los umbrales de confianza y NMS. Esto garantiza que solo se conserven las detecciones más confiables.

```

outputs = postprocess(
outputs, self.num_classes, self.confthre, self.nmsthre
)

```

- **Visualización de Detecciones:** Finalmente, las detecciones de objetos se visualizan en el fotograma utilizando la función `plot_tracking`. Esta función dibuja las cajas delimitadoras alrededor de los jugadores detectados, y puede mostrar el ID del jugador junto con la caja de la detección.

```

online_im = plot_tracking(
img_info['raw_img'], online_tlwhs,
online_ids, frame_id=frame_id + 1, fps=1. timer.average_time
)

```

10.7 Aplicar tracker (DeepEIoU) y modelo de reID

En esta sección, se aplican el tracker Deep-EIoU y el modelo de reidentificación (ReID) para hacer el seguimiento y asociar las detecciones de jugadores a lo largo de los fotogramas de un vídeo.

10.7.1 Deep-EIoU en DeepEIoU.py

El archivo DeepEIoU.py contiene la implementación del tracker Deep-EIoU, que utiliza el algoritmo de seguimiento y Kalman filter para asociar las detecciones de objetos a lo largo de los fotogramas. A continuación se describen los puntos clave de su implementación:

- **Clase STrack:** La clase STrack representa un objeto de seguimiento y contiene las características principales, como la predicción de la posición del objeto, la actualización de las características y la gestión de su estado utilizando un filtro de Kalman.

```
class STrack(BaseTrack):
    shared_kalman = KalmanFilter()
    def __init__(self, tlwh, score, feat=None,
                 feat_history=30):
        ...

```

La clase gestiona el estado del objeto (`tracked`, `lost`), la actualización de sus características visuales (con el método `update_features`) y la predicción de la siguiente posición usando el filtro de Kalman.

- **Predicción y Actualización:** La predicción de la siguiente posición de un objeto se realiza usando el filtro de Kalman para predecir su estado y su covarianza. La función `predict()` actualiza la posición del objeto.

```
def predict(self):
    self.mean, self.covariance = self.kalman_filter.
        predict(self.mean, self.covariance)
```

- **Activación y Re-activación:** Los objetos se activan al ser detectados por primera vez y se reactivan si han sido perdidos temporalmente. El método `activate()` inicia un nuevo seguimiento, mientras que `re_activate()` se utiliza para volver a activar objetos previamente perdidos.

```
def activate(self, kalman_filter, frame_id):
    ...
def re_activate(self, new_track, frame_id, new_id=False):
    ...

```

- **Estado del Objeto:** La propiedad `tlwh` convierte la posición de los objetos en el formato de caja delimitadora y la propiedad `xywh` convierte la caja en un formato de coordenadas (centro x, centro y, ancho, altura).

```
@property
def tlwh(self):
    ...
@property
```

```

def xywh(self):
    ...

```

- **Tracker Deep_EIoU:** El tracker Deep_EIoU gestiona los objetos detectados y su seguimiento a lo largo de los fotogramas. El método `update()` actualiza los objetos seguidos, gestiona las asociaciones entre objetos y detecciones, y controla la activación y reactivación de los objetos.

```

class Deep_EIoU(object):
    def __init__(self, args, frame_rate=30):
        ...
    def update(self, output_results, embedding):
        ...

```

El método `update()` recibe las detecciones y las características de cada objeto, realiza asociaciones entre los objetos seguidos y las nuevas detecciones, y actualiza el estado de cada objeto.

- **Asociación de Detecciones:** La asociación de objetos se realiza utilizando una combinación de la distancia IoU (Intersection over Union) y la distancia en el espacio de características (`embedding`) de ReID. Los objetos se asocian primero con detecciones de alta puntuación y luego con detecciones de puntuación baja.

```

ious_dists = matching.eiou_distance(strack_pool,
                                     detections, cur_expand_scale)
matches, u_track, u_detection = matching.
    linear_assignment(dists, thresh=self.args.
                      match_thresh)

```

- **Gestión de la Pérdida de Objetos:** Los objetos perdidos se gestionan mediante el parámetro `max_time_lost`, y se marcan como eliminados si no se han rastreado durante el número máximo de fotogramas permitidos.

```

for track in self.lost_stracks:
    if self.frame_id - track.end_frame > self.
        max_time_lost:
        track.mark_removed()

```

10.7.2 Modelo de Reidentificación (ReID)

El modelo de ReID se utiliza para asociar objetos detectados en diferentes fotogramas a través de sus características visuales. A continuación, se describe cómo se integran las características de ReID con el tracker Deep-EIoU:

- **Extracción de Características de ReID:** Se extraen características visuales de las detecciones utilizando un extractor de características preentrenado, como el modelo `osnet_x1_0`.

```

extractor = FeatureExtractor(
    model_name='osnet_x1_0',
    model_path='checkpoints/sports_model.pth.tar-60',
    device='cpu'
)

```

- **Asignación de Características a Detecciones:** Las características extraídas se asignan a las detecciones de los jugadores, permitiendo al tracker realizar la reidentificación de los mismos a lo largo de los fotogramas.

```

cropped_imgs = [frame[max(0, int(y1)):min(height, int(y2)), max(0, int(x1)):min(width, int(x2))] for x1,
y1, x2, y2, _, _, _ in det]
embs = extractor(cropped_imgs)
embs = embs.cpu().detach().numpy()

```

10.8 Aplicar GTA-LINK

El proceso de seguimiento de objetos en un vídeo puede verse afectado por problemas como cambios de identidad o interrupciones temporales en el seguimiento de jugadores. Para abordar estos desafíos, se aplica el método **GTA: Global Tracklet Association**. Este enfoque se utiliza para refinar y mejorar la asociación de tracklets generados durante el seguimiento. A través de la aplicación de GTA, es posible corregir problemas como cambios de identidad, fusionar tracklets incorrectos y garantizar una asociación más precisa de los jugadores a lo largo del tiempo.

GTA-LINK permite mejorar el rendimiento del seguimiento de objetos al trabajar sobre los resultados de cualquier modelo de seguimiento, como DeepEIoU. Este proceso de post-procesamiento refina las asociaciones de tracklets mediante dos componentes principales: el **tracklet splitter** y el **tracklet connector**. El primero se encarga de dividir tracklets que contienen múltiples identidades utilizando un algoritmo de clustering no supervisado basado en densidad, mientras que el segundo conecta los tracklets correctos mediante el cálculo de distancias promedio entre las características de los tracklets.

La importancia de GTA-LINK radica en su capacidad para corregir y mejorar los resultados de cualquier modelo de seguimiento sin necesidad de modificaciones en el modelo de seguimiento original. En el flujo de trabajo, se emplea este método después de obtener las predicciones iniciales de seguimiento y antes de la reidentificación para garantizar que las asociaciones de tracklets sean lo más precisas posibles.

En este apartado del flujo, se utilizan dos scripts principales para generar y refinar los tracklets: **generate_tracklets.py** y **refine_tracklets.py**. A continuación, se explica cómo funciona cada uno de estos scripts y qué aportan al flujo general de trabajo.

10.8.1 generate_tracklets.py

El script **generate_tracklets.py** tiene como objetivo generar tracklets a partir de los resultados de seguimiento y las secuencias originales de fotogramas en formato RGB. Este proceso se realiza de la siguiente manera:

- Primero, se carga un extractor de características utilizando el modelo de reidentificación **osnet_x1_0**. Este modelo es responsable de generar las características que se usarán para asociar las detecciones de los jugadores en cada fotograma.
- Se procesan las secuencias de vídeo para obtener las predicciones de seguimiento en cada fotograma. Para cada fotograma, se extraen las características de las detecciones y se actualizan los tracklets existentes.

- El script organiza los resultados de las detecciones de cada secuencia en archivos .pkl, que contienen los tracklets de la secuencia, con las características extraídas asociadas a cada uno.

El código esencial del script se muestra a continuación:

```
def main(model_path, data_path, pred_dir, tracker):
    # Cargar extractor de características
    val_transforms = T.Compose([T.Resize([256, 128]), T.
        ToTensor(), T.Normalize(mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])])
    extractor = FeatureExtractor(model_name='osnet_x1_0',
        model_path=model_path, device=device)
    # Procesar secuencias y generar tracklets
    for s_id, seq in tqdm(enumerate(seqs, 1), total=len(seqs),
        desc='Processing Seqs'):
        # Iterar sobre los frames y actualizar los tracklets con
        # las detecciones y características
        for frame_id in range(1, last_frame+1):
            # Actualización de tracklets con las características
            # extraídas
            features = extractor(input_batch)
            # Guardar tracklets en archivos .pkl
            with open(track_output_path, 'wb') as f:
                pickle.dump(seq_tracks, f)
```

Este script genera los tracklets a partir de las predicciones del seguimiento, asociando las características extraídas a cada tracklet y guardando los resultados en formato **pickle**.

10.8.2 refine_tracklets.py

El script **refine_tracklets.py** es un proceso de post-procesamiento que se aplica a los tracklets generados para mejorar su precisión y corregir errores en la asociación de identidades. Este script realiza las siguientes tareas:

- **Tracklet splitter:** Utiliza el algoritmo de clustering DBSCAN para dividir los tracklets impuros que contienen múltiples identidades. Esto se logra mediante el análisis de las características de los jugadores en los diferentes fotogramas.
- **Tracklet connector:** Después de dividir los tracklets, se conecta los tracklets correctos basándose en la similitud de sus características. Se usa una métrica de distancia para evaluar las características y decidir si dos tracklets deben fusionarse.
- **Visualización de distancias:** El script también genera mapas de calor para visualizar las distancias entre los tracklets, lo que permite evaluar cómo las conexiones y divisiones afectan la precisión del seguimiento.

El código esencial de este script es el siguiente:

```
def merge_tracklets(tracklets, seq2Dist, Dist, seq_name=None,
    max_x_range=None, max_y_range=None,
    merge_dist_thres=None):
```

```

# Realiza la fusión de los tracklets
while (np.any(Dist[non_diagonal_mask] < merge_dist_thres)):
    min_index = np.argmin(Dist[non_diagonal_mask])
    track1 = tracklets[idx2tid[track1_idx]]
    track2 = tracklets[idx2tid[track2_idx]]
    inSpatialRange = check_spatial_constraints(track1, track2,
                                                max_x_range, max_y_range)
    if inSpatialRange:
        track1.features += track2.features
        track1.times += track2.times
        track1.bboxes += track2.bboxes
        tracklets[idx2tid[track1_idx]] = track1
        tracklets.pop(idx2tid[track2_idx])

```

Este script se encarga de mejorar los tracklets mediante la fusión y la división de tracklets en función de sus características y la similitud espacial. Esto mejora la precisión del seguimiento de jugadores en el vídeo.

11 Resultados Preliminares

11.1 Definiciones de las métricas usadas

En esta subsección definiré el significado de las métricas usadas y su importancia en la evaluación.

11.1.1 Rank-k Accuracy (Rank-1, Rank-5, Rank-10, etc.)

La Rank-k accuracy es una métrica que permite calcular con qué frecuencia la identidad correcta aparece dentro de las k predicciones más próximas (medidas mediante distancia de embeddings) generadas por el modelo considerando k: la Rank-1 accuracy mide si la identidad correcta ocupa la primera posición (top-1) cuando el modelo devuleve una predicción; ejemplos para Rank-1: para una determinada imagen de consulta (query) del jugador "player_005", el modelo genera un embedding para la query, que confronta con todos los embeddings de la galería, con los que genera ranking ordenando los resultados (según similitud o distancia Euclídea/coseno) obteniendo el ranking. La imagen que ocupa el primer lugar (top-1) en la galería será un acierto en Rank-1, si corresponde a la misma identidad que el jugador de la query.

$$\text{Rank-1} = \left(\frac{\# \text{ queries con identidad correcta en top-1}}{\# \text{ total de queries}} \right) \times 100\% \quad (1)$$

Nota importante: Un embedding es transformar una imagen (por ejemplo, de un jugador) en un vector numérico, de dimensión fija (por ejemplo, 512 dimensiones), que resume su contenido visual de forma discriminativa (permite distinguir entre diferentes identidades de forma efectiva).

11.1.2 mAP – mean Average Precision

La mAP no solo toma en cuenta que la predicción correcta esté entre las top-k, sino que también se interesa por el orden de las predicciones y por cómo se distribuyen las identidades que son correctas a lo largo del ranking.

Es una métrica más exigente y también más representativa de la calidad de recuperación en general.

Para cada query:

Se calcula el Average Precision (AP), que evalúa la precisión acumulada mientras se van encontrando imágenes correctas en el ranking.

Finalmente, se hace la media de estos valores sobre todas las queries, que es el mean Average Precision (mAP).

Fórmula del AP para una query con varias coincidencias relevantes:

$$AP = \sum_{k=1}^N P(k) \cdot \Delta r(k) \quad (2)$$

Donde:

- $P(k)$ es la precisión en la posición k .
- $\Delta r(k)$ es el cambio en recall en esa posición.

Si solo hay una coincidencia relevante (una imagen en la galería con la misma identidad), el AP será igual a la precisión en la posición en que aparece esa coincidencia.

$$mAP = \frac{1}{Q} \sum_{i=1}^Q AP_i \quad (3)$$

Donde:

- Q es el número de queries.
- AP_i es la *average precision* para la query i .

11.2 Comparación de protocolos de re-identificación

[4]

En ambos casos, tanto en el protocolo de Market-1501 como en la evaluación actual, la idea central es muy parecida, pero difieren en qué se pone en la galería y cómo se definen las consultas:

1. Market-1501 (Zheng et al., CVPR 2015)

- **Dataset.** Cada persona del set de test aporta varias imágenes (por ejemplo, desde varias cámaras).
- **Split query vs. gallery.** Para cada persona, se elige una imagen como “consulta” y el resto pasa a la “galería”. Así reproduce un escenario realista: buscas a alguien (consulta) entre muchas posibles (galería), pero sin que la misma tome aparezca en ambos lados.

- **Distancias y ranking.** Mides la distancia (euclídea o coseno) entre cada consulta y todas las imágenes de la galería, y ordenas los candidatos de menor a mayor distancia.
- **CMC (Rank- k).** Calculas el porcentaje de consultas en que la persona correcta está entre los k primeros resultados (Rank-1, Rank-5, Rank-10).
- **mAP.** Para cada consulta construyes su curva Precisión–Recall sobre todo el ranking y obtienes su *Average Precision*; luego promedias sobre todas las consultas.
- **Ventajas.**
 - Evalúa “one-shot” re-identificación en un benchmark controlado.
 - Galería relativamente pequeña (decenas–centenas de imágenes).

2. Tu protocolo actual (detección vs. detección por frame)

- **Extracción.** A partir de un vídeo, generas miles de recortes (detecciones) y extraes un embedding por cada uno.
- **Split query vs. gallery.** Para cada jugador (`track_id`), tomas una detección como consulta y todas las demás detecciones de ese mismo jugador pasan a la galería, que puede ser muy grande (p. ej. 25 000 detecciones).
- **Distancias y ranking.** Igual que en Market-1501, mides distancias consulta→galería.
- **CMC (Rank- k) y mAP.** Cálculo idéntico: porcentaje de veces que tu embedding de consulta aparece en top- k de esas 25 000 detecciones, y AP sobre el ranking completo.
- **Diferencias clave.**
 - **Escala de la galería:** Market-1501 suele tener menos de 1 000 imágenes; aquí tienes decenas de miles.
 - **Naturaleza de la galería:** Market-1501 agrupa vistas distintas (cámaras), mientras que tú repites la misma cámara+jugador cientos de veces (frames contiguos).
 - **Objetivo:** Market-1501 mide robustez a cambios de punto de vista e iluminación entre cámaras; tu versión mide robustez a pequeñas variaciones de pose y ruido de detección dentro de un mismo vídeo.

11.2.1 Ejemplo de evaluación de Re-Identification

Supongamos que ya tienes dos ficheros de entrada:

`feats_all.npy` con 6 embeddings (cada uno es un vector, pero aquí sólo indicamos su índice):

idx:	0	1	2	3	4	5
feats:	e_0	e_1	e_2	e_3	e_4	e_5

`ids_all.npy` con los IDs correspondientes:

idx:	0	1	2	3	4	5
ids:	1	2	1	2	3	3

1. Split consulta vs. galería.

Para cada ID que aparece al menos dos veces:

- ID 1 en índices 0 y 2: consulta $q_0 = 0$, galería $g_0 = 2$.
- ID 2 en índices 1 y 3: consulta $q_1 = 1$, galería $g_1 = 3$.
- ID 3 en índices 4 y 5: consulta $q_2 = 4$, galería $g_2 = 5$.

Quedan consultas $[0, 1, 4]$ y galería $[2, 3, 5]$.

2. Matriz de distancias (Query \times Galería).

Supongamos estas distancias euclídeas:

	$g = 2$	$g = 3$	$g = 5$
$q = 0$	0.5	1.2	2.0
$q = 1$	1.0	0.4	1.5
$q = 4$	2.5	2.0	0.3

3. Rank-1 (mini-ejemplo).

Para cada consulta, el candidato más cercano:

- $q = 0$ (ID 1): mejor es $g = 2$ (ID 1) \rightarrow acierto.
- $q = 1$ (ID 2): mejor es $g = 3$ (ID 2) \rightarrow acierto.
- $q = 4$ (ID 3): mejor es $g = 5$ (ID 3) \rightarrow acierto.

Rank-1 = 3/3 = 100%.

4. mAP (Average Precision).

Para cada consulta marcamos hits en el ranking completo:

Consulta	AP
$q = 0$	1.0
$q = 1$	1.0
$q = 4$	1.0

$$\text{mAP} = \frac{1.0 + 1.0 + 1.0}{3} = 100\%.$$

En un caso real con miles de embeddings y detecciones, el procedimiento es idéntico: se calculan Rank-1, Rank-5, Rank-10 y mAP sobre las listas ordenadas de distancias.

11.3 Evaluación de la re-identificación

Los resultados recopilados pertenecen a los modelos de Reid aplicados al conjunto de datos etiquetado, considerando solo las detecciones de cada jugador.

11.3.1 YOLOv10

Para YOLOv10 he tenido que hacer una evaluación más creativa para calcular las métricas.

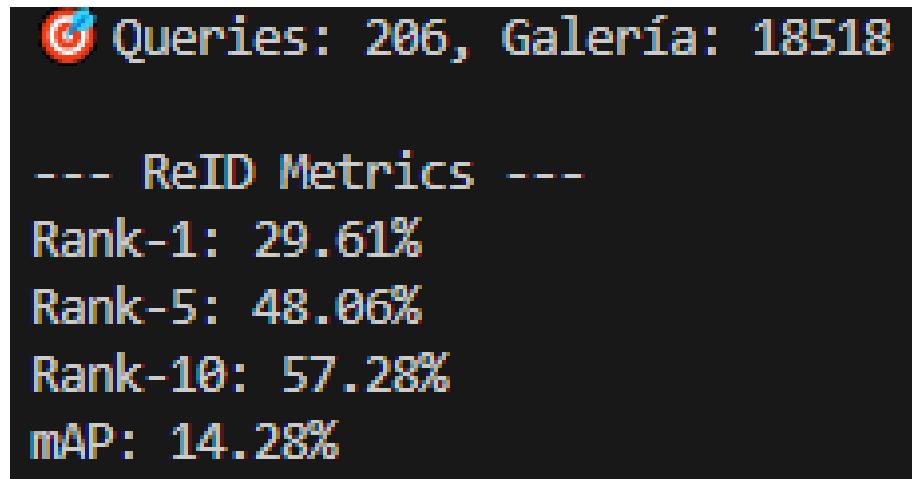


Figure 18: Resultado ReID Yolov10

Este experimento ilustra una forma poco habitual de aproximarse a utilizar un detector como el YOLOv10 para tareas de reidentificación y el resultado lo deja claramente manifiesto para tal propósito. El YOLOv10 está especializado para tareas de detección, no para reconocer si las clases son (muy) similares visualmente. Su arquitectura es capaz de localizar objetos, no de mapear imágenes en el espacio donde las distancias reflejan similitud de identidad, además de que sus capas de salida están específicamente diseñadas para boxes delimitadores, no para embeddings. El resultado lo pone de manifiesto: un Rank-1 que solamente alcanza el 29.6

11.3.2 Osnet Sportsmot

```
Computing CMC and mAP ....
** Results **
mAP: 28.6%
CMC curve
Rank-1 : 91.3%
Rank-5 : 98.0%
Rank-10 : 99.1%
Rank-20 : 99.5%
```

Figure 19: Resultado ReID con Osnet entrenado con el dataset de Sportsmot

Este modelo, que se basa en OSNet, fue entrenado con el dataset SportsMOT, un dataset de fútbol profesional que contiene secuencias de jugadores en entornos controlados y de alta calidad visual. La arquitectura OSNet permite extraer de manera efectiva tanto la información visual global como local, lo que se traduce en su alta Rank-1 del 91.3

A pesar de que es un modelo OSNet entrenado específicamente en el dominio futbolístico, las diferencias que existen del dominio en un sentido profesional y amateur limitan la capacidad de generalizar, lo que afecta la calidad general de la escala y penaliza el mAP.

11.3.3 Osnet Soccernet

```
Computing CMC and mAP ....
** Results **
mAP: 30.3%
CMC curve
Rank-1 : 90.4%
Rank-5 : 97.7%
Rank-10 : 98.6%
Rank-20 : 99.3%
```

Figure 20: Resultado ReID con Osnet entrenado con el dataset de Soccer-net

El presente modelo también se basa en OSNet, aunque, a diferencia del anterior, no se encuentra ajustado al dominio, sino que cuenta con preentrenamiento realizado sobre SoccerNet, es decir, un dataset profesional de fútbol.

El hecho de que el atributo Rank-1 sea relativamente alto (90.4%) un modelo tiene la capacidad de identificar patrones generales (forma del cuerpo, color de la camiseta) pero no está ajustado para poder lidiar con las especificidades visuales del fútbol amateur, lo que se traduce en un deterioro muy importante en las predicciones que son más difíciles.

El hecho de que OSNet sea una arquitectura potente sólo nos puede llevar a la conclusión que el desajuste de dominio impide la precisión global del modelo.

11.3.4 ResNet Soccernet

```
Computing CMC and mAP ...
** Results **
mAP: 28.8%
CMC curve
Rank-1 : 88.1%
Rank-5 : 96.7%
Rank-10 : 98.5%
Rank-20 : 99.4%
```

Figure 21: Resultado ReID con Resnet entrenado con el dataset de Soccernet

A pesar de las ventajas que proporciona su arquitectura, que es clásica y robusta, ResNet-50 no es una arquitectura adecuada para ReID por sí sola, ya que se enfoca en la extracción de características muy globales, abstractas, sin un mantenimiento local de la información espiritual. Mientras que OSNet es una arquitectura desenvolvente que extrae características de varias escalas y es capaz de mantener la información al nivel local, ResNet-50 tiende a extraer características más globales (aunque también reciben información de las escalas restantes). Esto perjudica el rendimiento en tareas como la reidentificación de las personas, donde los detalles visuales estereotipados marcan la gran diferencia. Al final, el resultado es similar al modelo OSNet preentrenado, pero un poco inferior, en especial en Rank-1.

A pesar de ser robusta, ResNet-50 pierde poder de discriminación frente a las arquitecturas entrenadas de manera específica a la tarea (OSNet) y su rendimiento hace evidentes estas capacidades.

11.3.5 Osnet mi dataset estiquetado

```
Computing CMC and mAP . . .
** Results **
mAP: 89.7%
CMC curve
Rank-1 : 97.9%
Rank-5 : 99.3%
Rank-10 : 99.5%
Rank-20 : 99.7%
```

Figure 22: Resultado ReID con Osnet entrenado con mi dataset etiquetado

El máximo rendimiento de toda la batería de pruebas corresponde a este modelo, porque es el modelo mejor adaptado a la problemática de clasificar el fútbol amateur, dado que está entrenado directamente sobre las imágenes de tu propio conjunto de datos. OSNet está diseñado para aprender características discriminativas a la vez que captura información a nivel local y global gracias a su diseño multi-escala y a sus bloques convolucionales eficientes. En fotografía de fútbol amateur (donde las condiciones de luz, resolución y calidad son muy variables), esta arquitectura es particularmente adecuada en cuanto a que puede sacarle el mejor provecho a elementos visuales muy finos (textura de las camisetas, siluetas, posturas, etc.) para distinguir a los jugadores.

Una mAP alta indica que el modelo es capaz de mantener la consistencia incluso en los casos más difíciles, y un Rank-1 cercano al 98

11.4 Resultados de MOT

	MOTA	MOTP	IDF1	idp	idr	num_switches
TRACK	76.0%	0.065	71.0%	81.0%	63.1%	334

Figure 23: Resultado MOT Yolov10

	MOTA	MOTP	IDF1	idp	idr	num_switches
TRACK	92.9%	0.598	87.3%	84.7%	90.0%	193

Figure 24: Resultado MOT DeepEIoU con Osnet entrenado con el dataset de Sportmot

	MOTA	MOTP	IDF1	idp	idr	num_switches	
TRACK	92.7%	0.471	88.6%	86.0%	91.5%		241

Figure 25: Resultado MOT DeepEIoU con Osnet entrenado con el dataset de Soccernet

resnet50_fc512

12 Dificultades encontradas

13 Conclusiones

References

- [1] Hsiang-Wei Huang et al. “Iterative Scale-Up ExpansionIoU and Deep Features Association for Multi-Object Tracking in Sports”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2024, pp. 163–172.
- [2] Papers With Code. *Multiple Object Tracking on SportsMOT*. Accessed: 2025-04-06. 2025. URL: <https://paperswithcode.com/sota/multiple-object-tracking-on-sportsmot>.
- [3] Jiacheng Sun et al. “GTA: Global Tracklet Association for Multi-Object Tracking in Sports”. In: *Proceedings of the Asian Conference on Computer Vision*. Springer, 2024, pp. 421–434.
- [4] Liang Zheng et al. “Scalable Person Re-identification: A Benchmark”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1116–1124. DOI: 10.1109/ICCV.2015.133.



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga