Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática (10II)

Trabajo de Fin de Grado

Localización de pelotas de pádel en secuencias de vídeo multi-vista

Autor: Adrián García-Pozuelo Fornieles
Tutor: Roberto Valle Fernández

Madrid, Junio - 2025

Resumen

La detección automática de objetos que se desplazan a gran velocidad representa un reto considerable dentro del área de visión por computador. Dentro del ámbito deportivo, y en especial para la práctica del pádel, la localización de la pelota en juego es especialmente interesante para extraer métricas durante los partidos.

Las soluciones basadas en aprendizaje automático tradicional presentan limitaciones a la hora de detectar la pelota en condiciones adversas. Localizar la pelota con precisión en escenarios sin restricciones es una tarea difícil debido a diversos factores: la iluminación puede provocar confusión entre la pelota y el fondo, la pelota puede estar ocluida por la raqueta o por el propio jugador según la perspectiva de la cámara, y el tamaño de la pelota en cada imagen varía según su distancia a la cámara.

A su vez, es imprescindible el uso de cámaras especializadas con una alta tasa de fotogramas por segundo (FPS) para capturar la posición y el contorno de la pelota con nitidez. Como resultado, para procesar la ingente cantidad de fotogramas que contiene cada vídeo, es imprescindible minimizar el tiempo de cómputo que requiera el modelo para procesar cada imagen.

Por ello, el principal desafío de este trabajo consiste en mejorar los resultados existentes para que la detección de la pelota sea más robusta frente a estas situaciones, sin incrementar demasiado el tiempo de cómputo requerido. La idea es entrenar y evaluar distintas redes de neuronas convolucionales (CNNs) diseñadas para detectar objetos de manera eficiente.

Para entrenar y evaluar el rendimiento de cada modelo, se ha llevado a cabo una anotación semiautomática de la posición de la pelota en 25 vídeos grabados en exteriores. En los fotogramas en los que la pelota está completamente ocluida, se ha desarrollado otro modelo que estima la posición de la pelota durante los momentos en que esta desaparece, utilizando como referencia sus posiciones anteriores y posteriores. Asimismo, se ha creado una aplicación que permite a cualquier usuario tanto visualizar los vídeos y anotaciones, como evaluar cada modelo de forma accesible.

Como resultado, se ha logrado un sistema capaz de realizar detecciones de forma precisa y eficiente mediante el detector de objetos YOLO. Los experimentos revelan que el modelo YOLO detecta la pelota correctamente en el 98.4 % de los fotogramas de los vídeos etiquetados, procesando cada fotograma a 29.52 FPS en CPU y 119.81 FPS en GPU.

Abstract

The automatic detection of fast moving objects presents a significant challenge in the field of computer vision. In the sports domain, particularly in padel, the localization of the ball during gameplay is especially valuable for extracting metrics during matches.

Traditional machine learning based solutions face limitations when it comes to detecting the ball under adverse conditions. Accurately locating the ball in unconstrained environments is difficult due to several factors: lighting variations may cause the ball to blend with the background, the ball may be occluded by the racket or the player depending on the camera's perspective, and the ball's size in each frame varies according to its distance from the camera.

Additionally, the use of specialized cameras with a high frame rate (FPS) is essential to capture the ball's position and contour clearly. As a result, in order to process the massive number of frames contained in each video, it is crucial to minimize the model's computation time per image.

Therefore, the main challenge of this work is to improve existing results to make ball detection more robust under these conditions, without significantly increasing the computational cost. The idea is to train and evaluate various convolutional neural networks (CNNs) designed to detect objects efficiently.

To train and assess the performance of each model, a semi automatic annotation of the ball's position was carried out on 25 outdoor recorded videos. For frames in which the ball is fully occluded, another model was developed to estimate the ball's position during the moments it disappears, using its previous and subsequent positions as reference. Furthermore, an application has been developed that allows any user to visualize the videos and annotations, as well as evaluate each model in an accessible way.

As a result, a system capable of performing accurate and efficient detections using the YOLO object detector has been achieved. Experiments show that the YOLO model correctly detects the ball in 98.4 % of the frames in the annotated videos, processing each frame at 29.52 FPS in CPU and 119.81 FPS in GPU.

Índice general

Capítulo 1

Introducción

En este primer capítulo se presentan, el contexto bajo el que se enmarca el TFG (sección 1.1), los motivos que han impulsado la realización de este proyecto (sección 1.2), así como los objetivos que se buscan alcanzar (sección 1.3). Finalmente, se describe la estructura que organiza el desarrollo del presente documento (sección 1.4).

1.1. Contexto

Este trabajo forma parte de una iniciativa centrada en el análisis avanzado del juego en deportes de raqueta, concretamente el pádel, que lleva a cabo la empresa SPORTS RTD HUB S.L.

Actualmente dicha empresa dispone de las bases para el análisis del comportamiento de la pelota de pádel mediante técnicas tradicionales de aprendizaje automático, abordando desafíos fundamentales como la detección de la pelota y la pala, así como la obtención de métricas asociadas a ambas [1].

Partiendo de ese enfoque inicial, el presente proyecto debe avanzar un paso en el diseño de soluciones inteligentes aplicadas al pádel, poniendo especial atención en la mejora de la detección de la pelota frente a condiciones adversas. Debido a que la pelota se graba usando cámaras de alta velocidad de captura de imágenes, en este caso una “Chronos 1.4 High Speed Camera” [2], colocada en diversas posiciones de la pista, la perspectiva de la escena en cada vídeo es distinta. A su vez, la pelota puede experimentar oclusiones parciales o totales, destellos por la iluminación y aparecer a diferentes escalas según la distancia con la cámara, condiciones que afectan a la precisión de los modelos encargados de la detección de la pelota.

1.2. Motivación

La actividad física y deportiva es fundamental para mantener el bienestar físico y mental, y la incorporación de tecnologías avanzadas puede fomentar significativamente la participación en diversas disciplinas deportivas. En este marco, la integración tecnológica en deportes como el pádel ofrece oportunidades para mejorar tanto la experiencia de los jugadores como la del público en general. Actualmente existe una demanda creciente de soluciones técnicas capaces de proporcionar análisis automáticos precisos del juego. Una parte fundamental de esta iniciativa consiste

en recopilar información relevante que permita optimizar tanto las técnicas como el rendimiento de los jugadores, mediante entrenamientos personalizados que contribuyan a mejorar su desempeño deportivo. En concreto, para el deporte del pádel, es especialmente importante detectar la pelota y a cada jugador a la hora de extraer métricas.

La relevancia técnica del proyecto radica en enfrentar desafíos específicos que surgen en entornos reales de captura de vídeo. Entre estos retos destacan las variaciones repentinas en las condiciones de iluminación, reflejos inesperados generados por superficies brillantes o cambiantes, y obstrucciones temporales o permanentes del objeto de interés. Estos retos requieren la implementación de métodos avanzados de visión por computador y técnicas robustas de aprendizaje automático, que aprenden mediante el uso de conjuntos de datos etiquetados.

La Figura 1.1 muestra el tipo de detección que se espera obtener por parte de los modelos. En ella se observa la pelota en el momento de ser golpeada por una pala, cuya superficie está recubierta con patrones de color blanco y negro conocidos como Arucos. Estos marcadores permiten al modelo "`arckpadel_detection`" [1] localizar con precisión la posición de la pala. Además de su utilidad como puntos de referencia, los Arucos proporcionan un alto contraste visual con respecto a la pelota, lo que facilita una detección en condiciones óptimas, sin interferencias visuales significativas. En la Figura 1.2, debido al fondo de color verde y a la dirección con la que la luz del sol incide sobre la pelota, el color de esta se satura volviéndose similar al del fondo y, por ende, haciendo más complicada su detección. En estas figuras se observan unos recuadros de color verde, conocidos como "bounding boxes". Estos recuadros representan las zonas donde el modelo ha detectado la pelota. Cada uno de ellos está definido por un punto (x, y), correspondiente a su esquina superior izquierda, junto con su altura y anchura. Esta información permite dibujar los recuadros sobre la imagen, proporcionando una representación visual de las detecciones realizadas por el modelo.

En la Figura 1.3 y Figura 1.4, se observa como el modelo "`arckpadel_detection`" [1], falla cuando trata de detectar la pelota en imágenes donde la pelota se confunde con el fondo y cuando hay oclusiones parciales.

Figura 1.1: Anotación manual de la pelota justo en el instante del impacto con la raqueta.

Figura 1.2: Anotación manual de la pelota cuando se encuentra sobre un fondo de color similar.

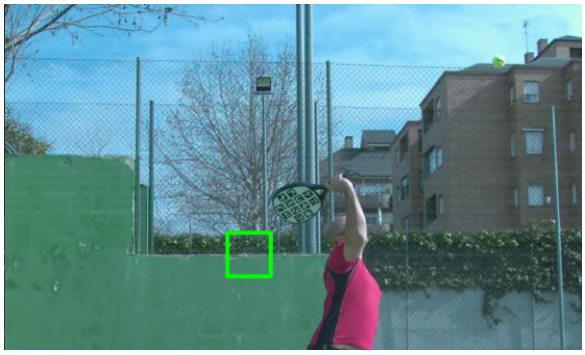


Figura 1.3: "arckpadel_detection" falla en la detección de la pelota cuando esta se encuentra sobre un fondo de color similar.

Figura 1.4: "arckpadel_detection" falla en la detección de la pelota en caso de oclusión parcial.

1.3. Objetivos

La meta principal del TFG es implementar un algoritmo de detección preciso y fiable, que funcione bien bajo múltiples condiciones de captura, incluyendo iluminación extrema, cambios de perspectiva y escala, u oclusiones parciales. El trabajo está enfocado exclusivamente en la detección de la pelota sin considerar métricas extraídas de la pala.

En este apartado se definen los objetivos que orientarán el desarrollo y la implementación del proyecto.

1. Etiquetado automático de una base de datos de 25 vídeos, con 1069 fotogramas por segundo (FPS), exportados a 60 FPS con una resolución entre 1920x1080 y 1280×720 píxeles. Dicha base de datos debe ser refinada manualmente para corregir posibles errores. Sirve tanto para entrenar como para evaluar el modelo de detección.
2. Detección automática de casos de oclusión total de la pelota, donde el etiquetador automático falla, y creación de un modelo para predecir la posición de la pelota en dichos casos.
3. Uso de redes de neuronas profundas para aprender la tarea de detección de la pelota en condiciones de cambios de luz, perspectiva y oclusiones.
4. Evaluación de los modelos y comparación con otros métodos.

1.4. Estructura del TFG

En el capítulo 1, se presentan las motivaciones, contexto y objetivos principales del trabajo, además de una visión general de la estructura del documento. En el capítulo 2, se explica el estado del arte sobre la detección de objetos en visión por computador usando redes de neuronas profundas. En el capítulo 3, se analiza en detalle el trabajo desarrollado desde la creación del base de datos a los modelos empleados para las detecciones y comparaciones entre estos. Finalmente, en el capítulo 4, se

ofrece una síntesis de todo lo realizado durante el proyecto y se proponen mejoras futuras.

Capítulo 2

Estado del arte

Este capítulo está dedicado a comprender los aspectos fundamentales y el estado del arte actual tanto en la detección de objetos en imágenes como métodos para hacer más robustas las predicciones de los modelos a condiciones de captura adversas.

2.1. Detección de objetos

La detección de objetos es una técnica dentro de la visión por computador cuyo objetivo es la identificación y localización automática de elementos de interés tanto en imágenes como en secuencias de vídeo. En la visión por computador existen varias técnicas que analizan imágenes, como puedan ser, la detección de objetos mediante bounding boxes, la clasificación del objeto capturado en estos delimitadores, o el seguimiento de objetos (tracking) a lo largo del tiempo en un vídeo. Cuando se trata de objetos que se mueven a gran velocidad, es necesario utilizar cámaras especiales [2] que cuentan con la capacidad de capturar una gran cantidad de imágenes por segundo (FPS). Además, debido a que las secuencias capturadas por estas cámaras contienen una gran cantidad de fotogramas por segundo, un modelo que se pueda usar en la detección de objetos a tiempo real debe ser suficientemente rápido como para procesarlas a esta velocidad mientras mantiene la precisión de detección.

Un ejemplo representativo de aplicación es la detección de una pelota en partidos de pádel: es un objeto pequeño, de movimiento veloz, y típico de entornos exteriores con iluminación variable. Lograr detectar consistentemente esta pelota requiere de algoritmos de visión artificial robustos tanto en precisión como en velocidad de inferencia y de equipamiento de vídeo especializado como son las cámaras que pueden capturar imágenes a alta tasa de fotogramas.

En los últimos años, los avances en redes neuronales convolucionales (CNN) y estrategias donde todo el proceso de detección de objetos se aprende y realiza en una única arquitectura de red, han permitido mejorar significativamente el rendimiento de la detección de objetos en tiempo real [3].

La necesidad de detectar objetos en condiciones de alta velocidad ha impulsado desarrollos tanto en algoritmos como en hardware. Esto se debe a que, cuando los objetos se mueven a elevada velocidad, es imprescindible capturar un número elevado de fotogramas por segundo (FPS) para no perder información crítica del movimiento. Tradicionalmente, los detectores de dos etapas (como Faster R-CNN [4]) lograban alta

precisión a costa de un elevado tiempo de cómputo, mientras que detectores de una sola etapa [5],[6] priorizaban la rapidez. Los avances recientes buscan combinar alta precisión con inferencia rápida, habilitando detección en tiempo real.

Modelos modernos alcanzan decenas de FPS en GPUs comunes, e incluso cientos de FPS en hardware especializado, sin sacrificar exactitud [7].

En particular, la familia de modelos YOLO (You Only Look Once) [8] ha dominado el campo de la detección en tiempo real gracias a su excelente balance entre precisión y velocidad. A continuación, se presenta un estado del arte de la detección, con énfasis en los modelos basados en YOLO más relevantes, así como los desafíos de robustez (iluminación, perspectiva, oclusiones) y las técnicas de entrenamiento empleadas para afrontarlos.

YOLO es una familia de modelos de código abierto desarrollados por Joseph Redmon y Ali Farhadi en la Universidad de Washington [9] que ha evolucionado rápidamente, dando lugar a múltiples versiones y variantes enfocadas en maximizar la velocidad sin perder exactitud [10]. A continuación se explican las principales versiones de "`YOLOv3`" a "`YOLOv11`" y variantes destacadas, indicando sus mejoras y desempeños:

El punto de inflexión para el desarrollo exponencial de la familia YOLO se produce en 2018 cuando la empresa Ultralytics, partiendo desde los modelos "`YOLOv1`" y "`YOLOv2`", desarrollan el modelo YOLOv3 [11], que podía lograr precisión competitiva de $\approx$20 FPS [12]. Desde entonces, cada nueva generación de modelos YOLO y derivados ha introducido mejoras de arquitectura y entrenamiento para empujar la frontera de velocidad/precisión.

Los modelos YOLO entrenados por Ultralytics utilizan el conjunto de datos COCO (Common Objects in Context), una base de datos a gran escala diseñada para tareas de detección, segmentación y subtitulación de objetos. COCO incluye aproximadamente 330.000 imágenes, de las cuales 200.000 están anotadas específicamente para estas tareas. El dataset abarca 80 categorías de objetos, desde elementos comunes como coches, bicicletas y animales, hasta clases más específicas como paraguas, bolsos o equipamiento deportivo. Además, COCO proporciona métricas de evaluación estandarizadas, como la precisión media media (mAP) (Sección 3.2) para la evaluación del rendimiento en detección de objetos[13].

2.1.1. YOLOv3

Tercera iteración del modelo YOLO, originalmente desarrollado por Joseph Redmon, que introdujo en 2018 una arquitectura más profunda (Darknet-53) con conexiones residuales y predicciones multi-escala para mejorar la detección de objetos de diferentes tamaños [10].

"`YOLOv3`" fue diseñado específicamente para tareas de detección de objetos. Actualmente, Ultralytics ofrece tres variantes de este modelo: "`YOLOv3u`", "`yolov3-tinyu`" y "`yolov3-sppu`". La "`u`" en sus nombres indica que estas versiones utilizan una cabeza de predicción "anchor-free" inspirada en la arquitectura de YOLOv8, a diferencia del diseño original de "`YOLOv3`" que empleaba un enfoque basado en "anchors" [14].

Estas variantes logran un buen equilibrio entre precisión y velocidad. Además, cada una de ellas presenta características y optimizaciones particulares que las hacen adecuadas para diferentes tipos de aplicaciones, permitiendo adaptar el modelo según

los requisitos de complejidad computacional o precisión deseada.

2.1.2. YOLOv4

YOLO, versión 4 es un modelo de detección de objetos en tiempo real desarrollado con el objetivo de superar las limitaciones de versiones anteriores, así como de otros detectores basados en redes neuronales convolucionales (CNN). A diferencia de otros modelos tradicionales, "YOLOv4" no solo se limita a tareas visuales, sino que también es eficaz en sistemas, gestión de procesos autónomos y reducción de intervención humana. Una de sus principales ventajas es que puede operar en GPUs de gama media, permitiendo un uso masivo con un coste asequible, estando además diseñado para funcionar a tiempo real utilizando una sola GPU tanto para entrenamiento como para inferencia [12][15].

Desarrollado por Bochkovskiy, "YOLOv4" introdujo múltiples avances tanto a nivel de entrenamiento (bag of freebies) como de arquitectura (bag of specials). Entre sus principales innovaciones se encuentran las conexiones residuales ponderadas (WRC), las conexiones parciales entre etapas (CSP), la normalización cruzada entre minibatches (CmBN), el entrenamiento auto-adversario (SAT), la función de activación Mish, la técnica de aumento de datos Mosaic, que combina múltiples imágenes en una sola, la regularización mediante DropBlock y la función de pérdida CIoU [10].

En cuanto a su arquitectura, "YOLOv4" sigue una estructura común en detectores de objetos, compuesta por: entrada, "backbone", "neck" y "head".

El "backbone", que puede basarse en modelos como VGG, ResNet o DenseNet [16], está preentrenado en ImageNet y es responsable de extraer las características principales de la imagen [17].

La cabeza de detección realiza las predicciones finales, tanto de las clases como de las coordenadas de las cajas de detección (bounding boxes).

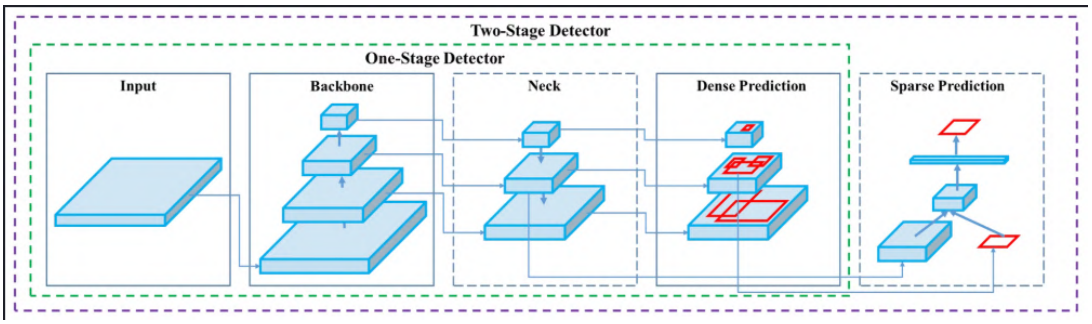


Figura 2.1: Arquitectura del modelo "YOLOv4". Fuente original de la imagen [17]

"YOLOv4" utiliza como "backbone" por defecto CSPDarknet53 [18], una arquitectura que mejora el flujo de gradientes y la reutilización de características. Gracias a estas mejoras, "YOLOv4" se convirtió rápidamente en un modelo de referencia, ya que demostró que era posible obtener alta precisión (mAP >40 % en COCO (Sección 3.2)) manteniendo una velocidad de procesamiento adecuada para aplicaciones en tiempo real.

2.1.3. YOLOv5

Lanzado en 2020 por Ultralytics, se caracteriza por su rendimiento, facilidad de uso en PyTorch y capacidad de adaptación a diferentes tareas relacionadas con la visión por ordenador.

"YOLOv5" incorpora muchos de los avances de "YOLOv4", como "Mosaic augmentation", "Cross-Stage Partial connections" (CSP) y técnicas de entrenamiento robusto. Además, introduce funcionalidades propias como "AutoAnchor", que permite calcular automáticamente las anclas óptimas. Cuenta con una serie de versiones adaptadas a diferentes recursos computacionales: "nano (n)", "small (s)", "medium (m)", "large (l)" y "extra-large (x)". Por ejemplo, la versión "YOLOv5x" logra un rendimiento de ≈50.7 % mAP (Sección 3.2) en COCO (input 640 px) con inferencia de hasta 200 FPS en batch en una GPU Tesla V100 [12]. Este modelo enfatiza la flexibilidad como eje central: facilidad para entrenar, desplegar y exportar modelos incluso en entornos móviles, lo que lo convierte en la base de numerosas soluciones industriales en tiempo real [19].

"YOLOv5u" presenta varias mejoras clave que lo hacen especialmente adecuado para aplicaciones en tiempo real y entornos exigentes. Su cabeza de detección Ultralytics "anchor-free split" elimina la dependencia de anclas, permitiendo una detección más flexible y adaptativa en distintos escenarios. Además, logra un equilibrio entre precisión y velocidad, lo que lo hace ideal para sistemas que requieren respuestas rápidas, como vehículos autónomos, robótica o análisis de vídeo en directo. Por último, ofrece una amplia gama de modelos preentrenados optimizados para diferentes fines como inferencia, validación o entrenamiento [20].

2.1.4. YOLOv8

Esta versión está diseñada tomando como base la arquitectura de "YOLOv5", pero integrando avances como una estructura "anchor-free", una cabeza de predicción "decouple" y el uso de funciones de pérdida avanzadas como CIoU y Distribution Focal Loss, que mejoran significativamente la localización precisa de objetos pequeños. Además, se mantiene el uso de un "backbone" CSP modificado (con capas C2f) y un bloque SPPF (Spatial Pyramid Pooling - Fast) para aumentar la eficiencia del procesamiento sin sacrificar precisión. [21][22]

Una de las características de "YOLOv8" es su arquitectura modular, que permite ejecutar distintas tareas, como segmentación de instancias y estimación de poses humanas. Ampliando su aplicabilidad en diversos contextos. [21]

"YOLOv8" incorpora múltiples mejoras que lo posicionan como una de las arquitecturas más avanzadas en detección de objetos. Su "backbone" y "neck" han sido rediseñados con arquitecturas de última generación para mejorar la extracción de características y el rendimiento general del modelo (Figura 2.2). La cabeza de detección "anchor-free" desarrollada por Ultralytics elimina la necesidad de utilizar cajas ancla, lo que mejora la eficiencia del entrenamiento y aumenta la precisión, especialmente en la detección de objetos pequeños y complejos. "YOLOv8" también se ha optimizado para alcanzar un equilibrio sobresaliente entre precisión y velocidad, lo que lo hace especialmente adecuado para aplicaciones en tiempo real. Además, ofrece una amplia variedad de modelos preentrenados adaptados a diferentes entornos, desde dispositivos móviles hasta servidores de alto rendimiento, evitando la necesidad de

realizar un entrenamiento desde cero. En términos de rendimiento, la versión más potente, "YOLOv8x", logró un mAP (Sección 3.2) de aproximadamente 53.9 % en el conjunto COCO y una velocidad de inferencia de hasta 280 FPS en una GPU NVIDIA A100 utilizando TensorRT, superando a su predecesor "YOLOv5" tanto en precisión como en velocidad gracias a una mejor optimización para hardware moderno. Finalmente, Ultralytics ha puesto un especial énfasis en la facilidad de uso y extensión de "YOLOv8", proporcionando una base de código unificada que facilita las tareas de entrenamiento, validación, exportación e inferencia, favoreciendo su integración en flujos de trabajo de investigación y producción [21],[22],[23].

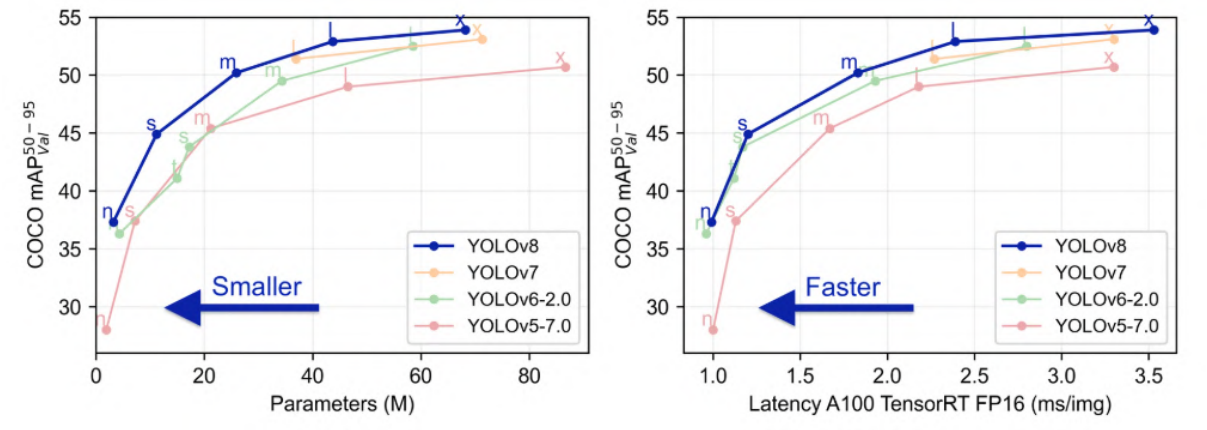


Figura 2.2: Métricas de rendimiento modelo "YOLOv8". Fuente original de la imagen [21]

2.1.5. YOLOv11

"YOLOv11" es la última iteración de la serie de detectores de objetos en tiempo real desarrollada por Ultralytics, y representa un nuevo estándar en cuanto a precisión, velocidad y eficiencia. Basándose en los avances alcanzados por versiones anteriores como "YOLOv8", esta versión introduce mejoras sustanciales tanto en la arquitectura como en los métodos de entrenamiento, convirtiéndose en una herramienta altamente versátil para una amplia gama de tareas de visión por computador. [24]

Uno de los objetivos principales de "YOLOv11" ha sido lograr una mayor eficiencia computacional sin comprometer la precisión, algo que se ha materializado en su versión intermedia "YOLOv11m", la cual obtiene un mAP superior al de "YOLOv8m" en el conjunto de datos COCO, utilizando un 22 % menos de parámetros. Esto supone una mejora significativa en el rendimiento, especialmente para su implementación en entornos con recursos limitados [25][26].

En resumen, los avances recientes han permitido que la detección de objetos rápidos sea factible en tiempo real, combinando innovaciones en modelos (especialmente la saga YOLO) con técnicas de optimización y paralelización.

Los modelos elegidos para el desarrollo de este trabajo son "YOLOv8n" y "YOLOv11n". Esta elección se fundamenta en que son los modelos más recientes, sin embargo aunque "YOLOv8n" sea el predecesor de "YOLOv11n", también se ha incluido pues se quiere comprobar diferencia entre la eficiencia computacional que se ha logrado en el

desarrollo del modelo "YOLOv11" y su precisión con las del modelo "YOLOv8n". Además la elección de la versión "n" se debe a la importancia de la velocidad de detección de los modelos, como se está tratando vídeos de alta tasa de fotogramas, se necesita que el procesamiento de cada fotograma individual sea lo más rápido posible. Se espera que la diferencia de precisión que hay entre los modelos "n" con modelos más pesados como el "m","l" o "x" se compense con el refinamiento de los mismos.

2.2. Condiciones de captura adversas

Además de la velocidad, un sistema de detección debe ser robusto a variaciones en el entorno y la apariencia del objeto. En escenarios reales (por ejemplo un partido al aire libre) la iluminación puede cambiar, la cámara puede observar desde distintos ángulos, y el objeto de interés puede ocultarse parcial o totalmente detrás de otros. A continuación, se analizan estos desafíos y cómo los aborda el estado del arte:

2.2.1. Invarianza a la iluminación

La iluminación en exteriores varía con la hora del día, condiciones meteorológicas (sol, nublado) y la presencia de luces artificiales. Un objeto puede verse muy diferente en luz solar intensa que en sombra. Estos cambios pueden afectar las predicciones de un detector entrenado en condiciones fijas. Para lograr robustez a la iluminación, en cuanto al estado del arte, la estrategia principal es introducir variaciones de brillo y color durante el entrenamiento (data augmentation) [27]. Por ejemplo, los modelos YOLO modernos aplican ajustes aleatorios de HSV (tono, saturación, valor) a las imágenes de entrenamiento [28]. Esto expone a la red a versiones más claras, oscuras o con diferente saturación de los objetos. En Ultralytics YOLO, el hiperparámetro hsv_v controla la variación de brillo aleatorio [29] permitiendo que durante el entrenamiento se oscurezcan o aclaren imágenes, simulando condiciones de luz distintas. Esta técnica logra que, por ejemplo, una pelota amarilla pueda ser detectada tanto bajo sol directo como en sombra. Adicionalmente, se emplean a veces filtros de aleatorización de contraste y gamma, e incluso convertir imágenes a escala de grises, para asegurar que el detector no dependa de un color específico [30].

De esta forma, las imágenes con las que se entrenan los modelos tendrían las variaciones representadas en las figuras 2.3, 2.4, 2.5 y 2.6:

Figura 2.3: Imagen original sin cambios.

Figura 2.4: Imagen original en escala de grises.

Figura 2.5: Imagen con aumento de brillo del 40 %.

Figura 2.6: Imagen con reducción de brillo del 40 %.

2.2.2. Invarianza al punto de vista (viewpoint)

En aplicaciones reales, la cámara puede capturar al objeto desde distintos ángulos: por ejemplo, una pelota vista de frente, desde un lateral, desde una cámara elevada o a nivel del suelo. Estas variaciones de perspectiva alteran la forma proyectada y el tamaño aparente del objeto en la imagen (figuras 2.7y 2.8). Un detector robusto debe reconocer el objeto pese a estas diferencias. Actualmente el estado del arte consiste en diversificar los datos de entrenamiento. Se emplean transformaciones geométricas aleatorias como rotaciones, escalados, traslaciones e incluso transformaciones de perspectiva sobre las imágenes de entrenamiento. Por ejemplo, "`YOLOv5`" implementa el parámetro "degrees" que permite realizar rotaciones aleatorias comprendidas entre los grados especificados y "perspective" para aplicar inclinaciones de la imagen [31]. El efecto es exponer al modelo a objetos orientados de maneras distintas y mediante transformaciones como rotaciones o "shear" que ayudan a que el modelo generalice las variaciones según el ángulo de visión, reconociendo objetos inclinados u oblicuos. De forma similar, las traslaciones aleatorias (desplazar recortes de la imagen) enseñan al modelo a detectar objetos aunque solo se vea una parte de ellos en el fotograma, lo cual también aporta robustez frente a encuadres distintos. En algunos casos, disponer de datos reales multi-vista es ideal, pero no siempre es posible, por lo que el incremento sintético de imágenes (data augmentation) suple muchas de estas variaciones.

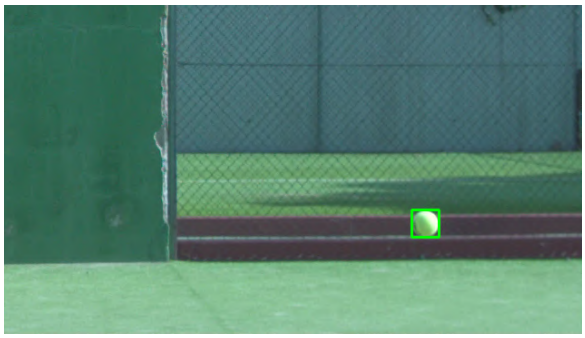


Figura 2.7: Punto de vista perpendicular al movimiento de la pelota.

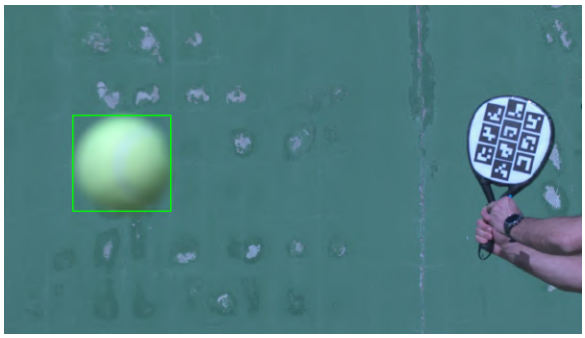


Figura 2.8: Punto de vista paralelo al movimiento de la pelota.

2.2.3. Invarianza a oclusiones

La oclusión ocurre cuando el objeto de interés está parcial (Figura 2.9) o completamente tapado por otro objeto (Figura 2.10) o por el límite de la imagen. En el caso de una pelota de pádel, puede quedar oculta momentáneamente detrás de un jugador, la red o desaparecer de la vista al salirse del cuadro. Las oclusiones representan un reto mayor, ya que el detector dispone de información incompleta. Para las oclusiones parciales, es deseable que el modelo pueda inferir la presencia del objeto a partir de sus partes visibles. El entrenamiento es crítico: se pueden simular oclusiones mediante técnicas de data augmentation dedicadas. Dos ejemplos populares son CutOut y CutMix. CutOut consiste en enmascarar regiones aleatorias de la imagen mientras que CutMix mezcla partes de dos imágenes diferentes (utilizando trozos de otra imagen sobre la original). Ambas obligan a la red a aprender a reconocer objetos aunque les falte un fragmento visual [32],[33]. "YOLOv4", por ejemplo, utiliza CutMix y Mosaic intensivamente en su entrenamiento [17]. Otro enfoque introducido en "YOLOv4" fue el Self-Adversarial Training (SAT), donde el modelo primero produce un adversarial attack sobre la imagen (modifica ligeramente la imagen para hacer más difícil la detección) y luego entrena con esa imagen modificada. Este proceso puede entenderse como forzar al detector a aprender a sobreponerse a perturbaciones que en efecto actúan como "oclusiones" o camuflaje del objeto, aumentando su robustez. Para las oclusiones totales, debido a que ningún detector puede "ver" un objeto que no está en la imagen, se depende de técnicas de seguimiento temporal para no perder el movimiento del objeto hasta que reaparezca. Por ejemplo, en un sistema de seguimiento de pelota, si la pelota se oculta tras un jugador por unos fotogramas, un algoritmo de tracking (como SORT [34] o Kalman filter [35]) puede predecir su posición y re-asociarla cuando vuelva a ser visible. Algunos enfoques combinan detección YOLO con tracking para manejar oclusiones prolongadas.

Figura 2.9: Oclusión parcial: la pelota está parcialmente fuera del recuadro.

Figura 2.10: Oclusión total: la pelota no es visible debido al tipo de golpe.

Capítulo 3

Resultados obtenidos

En esta sección se explica, en orden, los elementos y fases que han intervenido en el desarrollo de los modelos refinados "`YOLOv8n`" y "`YOLOv11n`" y sus comparaciones con el modelo "`arckpadel_detection`" [1]. Además se mostrará la aplicación de escritorio desarrollada que permite procesar los vídeos de forma intuitiva.

En primer lugar, se describirá la base de datos utilizada, incluyendo, sus características, el proceso de construcción y las decisiones de diseño adoptadas. A continuación, se analizarán los modelos empleados.

Posteriormente, se compararán y extraerán conclusiones a partir de los resultados obtenidos por ambos modelos. Finalmente se explicará la aplicación de escritorio desarrollada y el funcionamiento del código.

3.1. Construcción de la base de datos

La base de datos constituye la base de cualquier modelo de inteligencia artificial. En este proyecto, el modelo desarrollado sigue un enfoque de aprendizaje supervisado, lo que implica que los datos de entrada están etiquetados con sus correspondientes soluciones. Concretamente, se han anotado manualmente 25 vídeos grabados a 1069 fotogramas por segundo (FPS) [1].

En estos vídeos se capturan distintos golpeos a una pelota de pádel desde múltiples perspectivas, lo cual resulta un factor clave para desarrollar una base de datos variada. Esta diversidad es esencial para representar el mayor número de casuísticas posibles, tanto en los tipos de golpeo como los ángulos de visión que puedan tener las cámaras.

Contar con una base de datos amplia, realista y representativa no solo es imprescindible para entrenar un modelo de detección eficaz, sino que también es fundamental para poder evaluar rigurosamente su rendimiento.

3.1.1. Metodología para la anotación de la base de datos

El hecho de usar 25 vídeos como base de datos implica un reto técnico para etiquetar manualmente todos los fotogramas de estos vídeos (55.325 fotogramas). Este etiquetado tradicionalmente se realiza de forma manual con herramientas como labelImg

[36] o roboFlow [37], sin embargo, etiquetar esta cantidad de fotogramas puede resultar en una tarea altamente demandante para una sola persona, por ello se decidió usar un modelo de YOLO preentrenado por Ultralytics [22] (concretamente "`YOLOv8l`") para realizar una primera detección de los vídeos, permitiendo así que el etiquetado manual se tranformara en una revisión de las detecciones del modelo YOLO empleado, modificando, añadiendo y eliminando detecciones cuando hiciera falta.

El proceso de esta predetección cuenta con las siguientes fases:

1. Obtención de detecciones en la predetección: Las detecciones del modelo se guardan en un "`.csv`" con una estructura que permite ver de forma rápida si hay algún fotograma que no se ha detectado correctamente.

```
Frame,Deteccion Pelota,Esquina Superior Izquierda,Esquina Superior Derecha,Esquina Inferior Izquierda,Esquina Inferior Derecha,Area Cuadrado
580,False,"(0,0)","(0,0)","(0,0)","(0,0)",0
581,False,"(0,0)","(0,0)","(0,0)","(0,0)",0
582,True,"(1880, 649)","(2112, 649)","(1880, 847)","(2112, 847)",45936
583,True,"(1872, 660)","(2114, 660)","(1872, 868)","(2114, 868)",50336
```

Figura 3.1: Ejemplo de la estructura de los "`.csv`" empleados.

2. Refinamiento de las detecciones: Se ha elaborado un programa que permite visualizar y refinar las predetecciones realizadas (Figura 3.2). Con este programa, el usuario puede elegir tanto el vídeo como el fotograma por el que quiere comenzar a refinar. El usuario puede dibujar el contorno que corresponde a la pelota en cada fotograma, sobreescribiendo automáticamente las detecciones en el "`.csv`" correspondiente.

Figura 3.2: Aplicación desarrollada que permite al usuario refinar las detecciones.

Una vez refinadas las detecciones, se exportan los fotogramas, dibujando su detección, para poder comprobar que ninguna se ha guardado en el fotograma incorrecto y que todas describen correctamente la posición de la pelota.

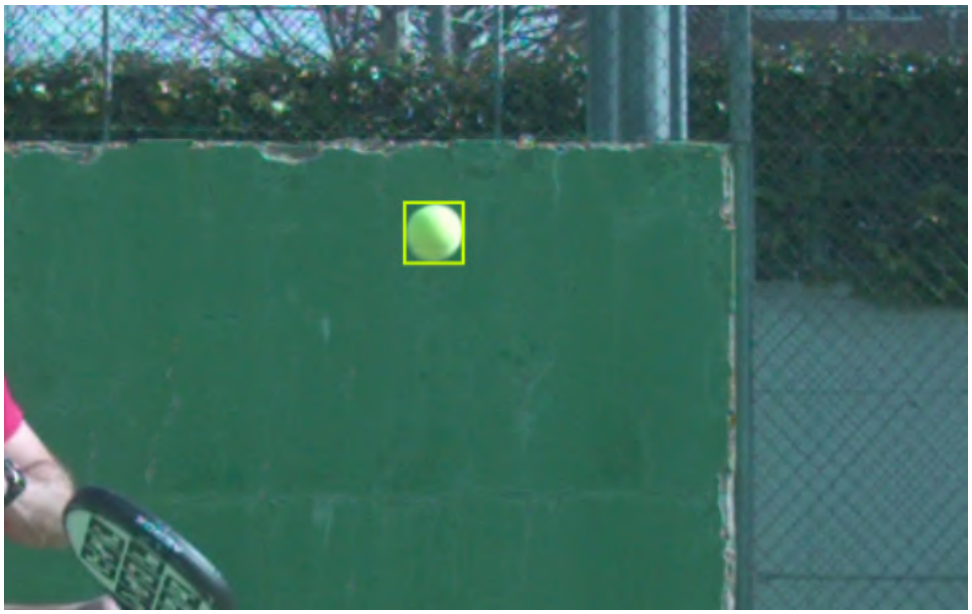


Figura 3.3: Ejemplo de cómo se ven las comprobaciones visuales de los fotogramas.

La herramienta utilizada para obtener automáticamente una base de datos con un formato que los modelos entienden para su refinamiento es roboFlow [38]. Se ha hecho uso de esta herramienta pues contiene opciones de data augmentation integradas (usadas posteriormente en la Sección 3.1.2), un sistema de control de versiones de la base de datos automático, redimensionado automático de las imágenes de forma que toda la base de datos mantenga el mismo tamaño y una interfaz web intuitiva.

Para poder trasladar la base de datos con su etiquetación correspondiente de un entorno local a la herramienta de refinado roboFlow, es necesario transformar las detecciones a un formato ampliamente utilizado como es el utilizado por la base de datos COCO (Common Objects in Context) (Sección 2.1) [39]. Un conjunto de anotaciones COCO consta de cinco secciones de información que aportan información para todo el conjunto de datos. El formato de un fichero "`.json`" que representa un conjunto de datos de detección de objetos COCO viene documentado de la siguiente manera [39]:.

- info: información general sobre el conjunto de datos.
- licenses: información de las licencias de imágenes del conjunto de datos.
- images: lista de imágenes del conjunto de datos.
- annotations: lista de anotaciones (incluidos los cuadros delimitadores) que están presentes en todas las imágenes del conjunto de datos.
- categories: lista de categorías de etiquetas.

Para ello se ha realizado un analizador que utiliza automáticamente los datos que se necesitan de cada fotograma para crear su sección dentro del formato COCO,

fotograma que posteriormente copiará en una carpeta dedicada que se utilizará en roboFlow.

Debido al elevado número de imágenes que se desea refinar, la herramienta roboFlow requiere que tanto las imágenes como sus detecciones, recogidas en un archivo "`.json`" con formato COCO, se suban a través de su API. En el archivo "`.json`" se especifica un identificador único para cada fotograma y Roboflow realiza un proceso automático de asociación entre cada imagen y su correspondiente detección.

El preprocesamiento de las imágenes sigue el esquema de la Figura 3.4

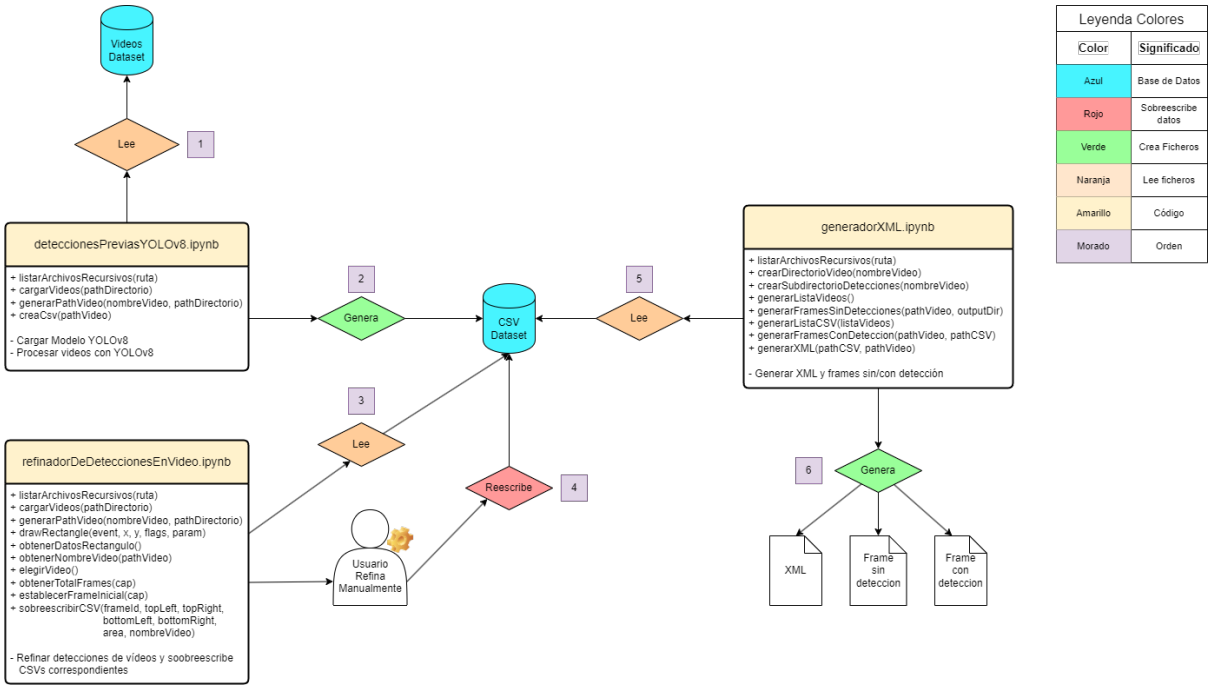


Figura 3.4: Esquema del flujo de trabajo de la creación del dataset.

1. Desde el fichero "`deteccionesPreviasYOLOv8`", se leen los vídeos a procesar, y a través de una llamada al modelo "`YOLOv8m`", se preprocesan, guardando las detecciones en múltiples "`.csv`".
2. Con el fichero "`refinadorDeteccionesEnVideo`", se utilizan las detecciones guardadas en los "`.csv`" para que el usuario pueda refinar manualmente los vídeos.
3. Finalmente, con el fichero "`generadorXML`", se leerán los "`.csv`" refinados y se generarán los ficheros "`.xml`" junto a sus fotogramas, necesarios para obtener las métricas del modelo "`arckpadel_detection`" más adelante, así como se guardarán los fotogramas con las detecciones dibujadas para que el usuario pueda hacer una revisión visual de las detecciones.

3.1.2. Data augmentation en la base de datos de imágenes

Para abordar el problema de los cambios de luz (Subsección 2.2.1) y oclusiones parciales (Subsección 2.2.3), se han creado imágenes nuevas a partir de las existentes,

que a través de diversas transformaciones, simulan los posibles cambios de luz que se pueden experimentar en los vídeos. Estas transformaciones son:

- Ajustes de brillo: transformaciones aleatorias de los valores de brillo de entre un -15 % y 15 %.
- Desenfoque: transformaciones aleatorias del desenfoque de la imagen de entre 0 y 1.5 píxeles. Permite simular posibles desenfoques de la cámara.
- Ruido salt and peper: se aplica al 0.1 % de los píxeles. Permite simular posibles suciedades que se encuentren en la lente de la cámara o hacer robusto al modelo frente a la lluvia.

Figura 3.5: Ejemplo de brillo al -12 %, ruido salt and peper y desenfoque de 1.3 píxeles.

Tanto los cambios de perspectivas como las oclusiones parciales se tratan de manera indirecta durante el data augmentation. Este tipo de oclusión se produce en dos situaciones: la pelota está siendo cubierta por la pala o la persona, o la pelota está entrando o saliendo de la imagen. Cuando se realiza el data augmentation para reforzar el modelo ante las variaciones de luz, también se están transformando las imágenes en las que la pelota se encuentra parcialmente ocluida y las imágenes con diferentes perspectivas.

3.1.3. División de la base de datos

Con el objetivo de refinar los modelos, se han separado los datos de la base de datos en entrenamiento, validación y test. A esta división hay que añadir que solo se usan para el refinado, aquellas imágenes donde la pelota es visible parcial o totalmente, pasando de tener una base de datos con 55.325 elementos, a una base de datos de 9.831 elementos, que después de realizar el data augmentation, acaba con un total de 23.545 imágenes divididas de la siguiente manera:

- Entrenamiento: 20.571 imágenes pertenecientes a 17 vídeos (87 %).
- Validación: 1.488 imágenes pertenecientes a 4 vídeos (6 %).
- Test: 1.486 imágenes pertenecientes a 4 vídeos (6 %).

3.1.4. Etiquetado en caso de oclusión

Si bien las oclusiones parciales están contempladas y tratadas en la base de datos de los modelos de detección de objetos en vídeo (Sección 3.1.1), las oclusiones totales no pueden resolverse mediante detección directa, ya que el detector utilizado para anotar los vídeos falla inevitablemente cuando la pelota queda completamente oculta.

Para abordar este problema, se ha adoptado un enfoque complementario basado en el análisis de la trayectoria de la pelota antes y después del momento de oclusión. A partir de las detecciones generadas por el modelo en los instantes en los que la pelota es visible, se extraen métricas que describen su movimiento. Estas métricas se emplean para predecir su posición en los fotogramas donde la pelota no aparece en la imagen. De este modo, se consigue una estimación de su posición cuando se producen oclusiones totales.

El modelo consta de las siguientes partes:

- Encoder: Captura la dinámica de movimiento de la pelota (posición y tamaño) justo antes y justo después del tramo oculto. El encoder se encarga de comprender el contexto, extrayendo características relevantes de la trayectoria de la pelota.
- Decoder: Genera la secuencia de posiciones y tamaños de la pelota durante los fotogramas ocluidos, auto-regresivamente. El decoder se enfoca en generar la trayectoria faltante, autónomamente, a partir del contexto resumido por el encoder.

De esta forma el encoder está implementado mediante un "`LSTM`" unidireccional [40],[41] que trabaja sobre una secuencia de longitud fija. Al terminar, extrae el vector final que condensa el contexto completo.

Finalmente el decoder está basado en un "`LSTMCell`" [42], el cual se inicializa con el estado final dejado por el encoder. Produce como salida un tensor con todas las predicciones de posición y tamaño.

De esta forma primero se codifica la secuencia de entrada a un estado interno, y luego se descodifica ese estado para producir la secuencia de salida.

Durante el entrenamiento se pueden observar los resultados de la Figura 3.6.

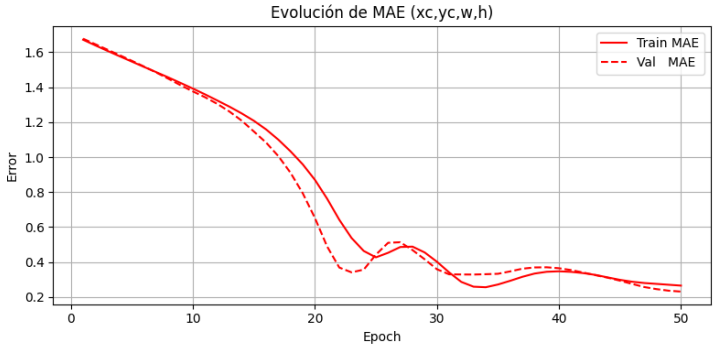


Figura 3.6: Métricas obtenidas durante el entrenamiento — evolución del MAE.

Al analizar las métricas asociadas al MAE (Mean Absolute Error) (Figura 3.6), se

observa que el modelo alcanza un mínimo constante en torno a 0.27. Este error representa la diferencia absoluta media entre las predicciones del modelo y los valores reales de las etiquetas, considerando las coordenadas de la esquina inferior izquierda de la caja de detección (x, y), así como su ancho y alto. Esto implica que, aunque el modelo localiza correctamente la región de la imagen donde se encuentra la pelota, las cajas de detección pueden presentar ligeros desplazamientos o desviaciones en tamaño respecto a las anotaciones reales, como se puede ver en la Figura 3.7 la diferencia entre las predicciones del modelo "pred" y los resultados correctos "gt".

```
Frame 1/3: pred=(0.618,0.959,0.051,0.058) gt=(0.701,0.937,0.033,0.055)
Frame 2/3: pred=(0.666,1.042,0.031,0.076) gt=(0.701,0.938,0.033,0.055)
Frame 3/3: pred=(0.699,1.064,0.024,0.081) gt=(0.701,0.938,0.033,0.055)
```

Figura 3.7: Resultados obtenidos durante la detección de una oclusión.

Para llamar al modelo predictor de posiciones en caso de oclusión, se sigue el esquema de la Figura 3.8.

1. Desde el fichero "`predictorOclusiones`" se hace que un modelo refinado haga el procesamiento de un vídeo. Las detecciones encontradas se guardan en un "`.csv`" que se usará para entrenar el modelo.
2. Al entrenar el modelo con el "`.csv`" de las detecciones realizadas, se guarda tanto el modelo entrenado, como las métricas de rendimiento durante el entrenamiento.

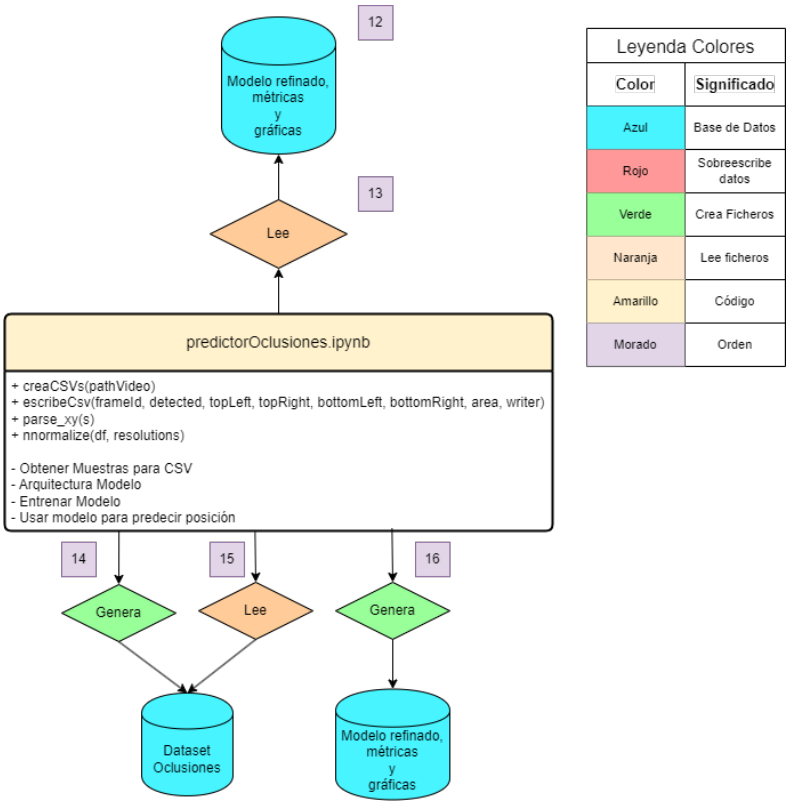


Figura 3.8: Esquema del workflow para entrenar y llamar al modelo predictor de posiciones en caso de oclusión.

3.1.5. Mapa de calor

Uno de los gráficos que nos permite encontrar posibles sesgos en la base de datos es el mapa de calor de las detecciones, que representa la distribución espacial de la pelota dentro de las imágenes en las que aparece.

Dado que la base de datos cuenta con vídeos grabados con diferentes resoluciones (1920x1080, 1280x1024 y 1280x720), el mapa de calor (Figura 3.9), que se ha generado a partir de las anotaciones del conjunto de entrenamiento acumulando la frecuencia de aparición de la pelota, se ha normalizado sobre el eje horizontal (x) y vertical (y), asegurando una representación independiente a la resolución original del vídeo. Este tipo de visualización permite identificar zonas de la imagen donde la pelota aparece con mayor frecuencia, lo cual puede influir en el aprendizaje del modelo.

En este caso se observa cómo la mayor concentración de pelotas en la base de datos se encuentran en el área compuesta por los sectores (x=0.3, y=0.5) y (x=0.8, y=0.9), lo que indica que la mayoría de las apariciones de las pelotas de la base de datos se encuentran en la parte superior central de la imagen. Esta agrupación se explica por la naturaleza de los vídeos incluidos en la base de datos, cuya perspectiva es generalmente perpendicular o paralela al movimiento de la pelota, favoreciendo su aparición en esa zona concreta del encuadre.

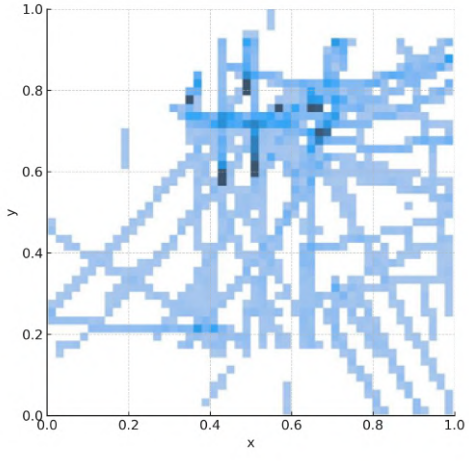


Figura 3.9: Mapa de Calor normalizado para representar la posición de la pelota detectada independientemente de la resolución del vídeo original.

Figura 3.10: Vista paralela al movimiento de la pelota.

Figura 3.11: Vista perpendicular al movimiento de la pelota.

3.2. Métricas de detección de objetos

Para poder analizar y comparar los modelos, se han empleado métricas ampliamente utilizadas en la visión por computador.

Estas son:

- Precision: Proporción de verdaderas detecciones positivas (True Positives, TP) entre todas las predicciones positivas (TP + False Positives, FP).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

- Recall: Proporción de verdaderas detecciones positivas respecto al total de objetos realmente presentes (TP + False Negatives, FN).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

- Average Precision (AP): Evalúa la precisión del modelo a lo largo de todos los niveles de recall.

$$\text{AP} = \int_0^1 \text{Precision}(r), dr$$

- mAP50: Promedio de precisión (AP) para cada clase, cuando la predicción se considera correcta si tiene un IoU ≥ 0.5. El IoU (Intersection over Union) mide la superposición entre la predicción y la ground truth.

$$\text{IoU} = \frac{\text{Área común}}{\text{Área total combinada}} \tag{3.3}$$

- mAP50-95: Promedia el AP para múltiples umbrales de IoU entre 0.50 y 0.95 (con paso 0.05) [43].

Estas métricas nos permiten realizar, entre otros, los siguientes análisis:

1. Si se obtienen valores bajos en el mAP50-95 y altos en el mAP50, el modelo detecta correctamente la pelota, pero las cajas de detección son poco precisas.
2. Se puede utilizar el mAP50 como una métrica resumida del rendimiento general del modelo, considerendo que es suficiente un 50 % de solapamiento para que una predicción se considere correcta.

3.3. Detalles de implementación

El código se ha desarrollado completamente en python, concretamente en cuadernos Jupyter Notebook, pues al contar este formato con celdas separadas que se pueden ejecutar independientemente sin tener que ejecutar las anteriores, pero compartiendo las variables y funciones definidas en otras celdas, permite tener ficheros autocontenidos donde está definido todo lo necesario para el correcto funcionamiento del código sin necesidad de llamar a otros módulos, característica que ayuda a mantener un orden con un número menor de archivos.

La versión de python utiliada es la 3.10.11 por motivos de compatibilidad con las librerías usadas, donde las descargadas son:

- torch versión 2.1.0: Núcleo de Pytorch, Framework de Facebook para deep learning [44].
- numpy, versión 1.26.4: Librería fundamental para cálculos numéricos y operaciones con arrays en Python [45].
- matplotlib, versión 3.10.3: Librería para generar gráficos y visualizaciones en 2D [46].
- cv2, versión 4.11.0.86: Módulo principal de OpenCV; se usa para procesamiento de imágenes y visión por computador [47].
- pandas, versión 2.2.3: Herramienta para análisis y manipulación de datos en forma de tablas (DataFrames) [48].
- PIL, versión 11.2.1: Biblioteca para abrir, editar y guardar imágenes [49].
- ultralytics, versión8.3.133: Implementación oficial de los modelos YOLO desde la versión 3 hasta las 11, para detección de objetos [50].
- roboflow, versión 1.1.63: Biblioteca que facilita es uso de la api de la herramienta roboFlow que simplifica la gestión de datasets y la integración con modelos de visión artificial [38].
- flet, versión 0.28.3: Framework para construir interfaces gráficas (GUIs) en Python de forma reactiva y multiplataforma [51].

Alguna de estas librerías como torch o ultralytics, permiten la ejecución del código en GPU gracias al uso de la tecnología Compute Unified Device Architecture (CUDA) Figura 3.12, que es una tecnología de NVIDIA que permite ejecutar cálculos masivamente en paralelo desde la GPU para acelerar aplicaciones como deep learning, procesamiento de imágenes, simulaciones científicas, entre otras [52].

Para su uso y compatibilidad con las librerías es necesaria la instalación de CUDA versión 11.8 y CUDA Deep Neural Network library (cuDNN) versión 8.6, que es una librería de aceleración desarrollada por NVIDIA para optimizar el rendimiento de redes neuronales profundas cuando se ejecutan en GPUs con CUDA [53].

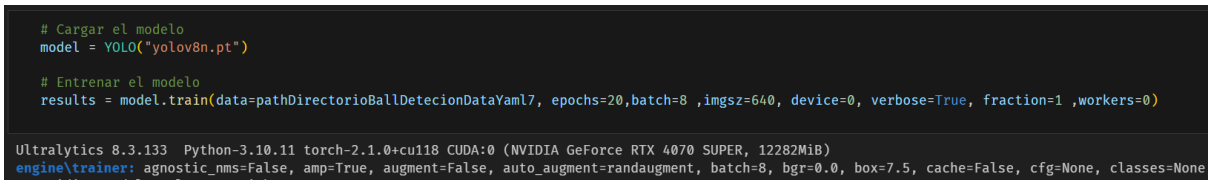


Figura 3.12: Ejemplo de la llamada para refinar el modelo preentrenado "YOLOv8" con una base de datos anotada en formato COCO (Sección 2.1). En la parte inferior de la imagen se puede observar cómo, al disponer de las versiones correctas de CUDA y cuDNN previamente instaladas y configuradas, la librería ultralytics detecta automáticamente la GPU disponible en el sistema y la emplea para llevar a cabo el proceso de entrenamiento.

3.4. Experimentos usando YOLO

Tanto para el refinamiento del modelo "`YOLOv8n`" como del modelo "`YOLOv11n`", se parte de los respectivos modelos preentrenados con la base de datos COCO, proporcionados por Ultralytics (Sección 3.2).

En el refinamiento de los modelos YOLO, se sigue el esquema de la Figura 3.13.

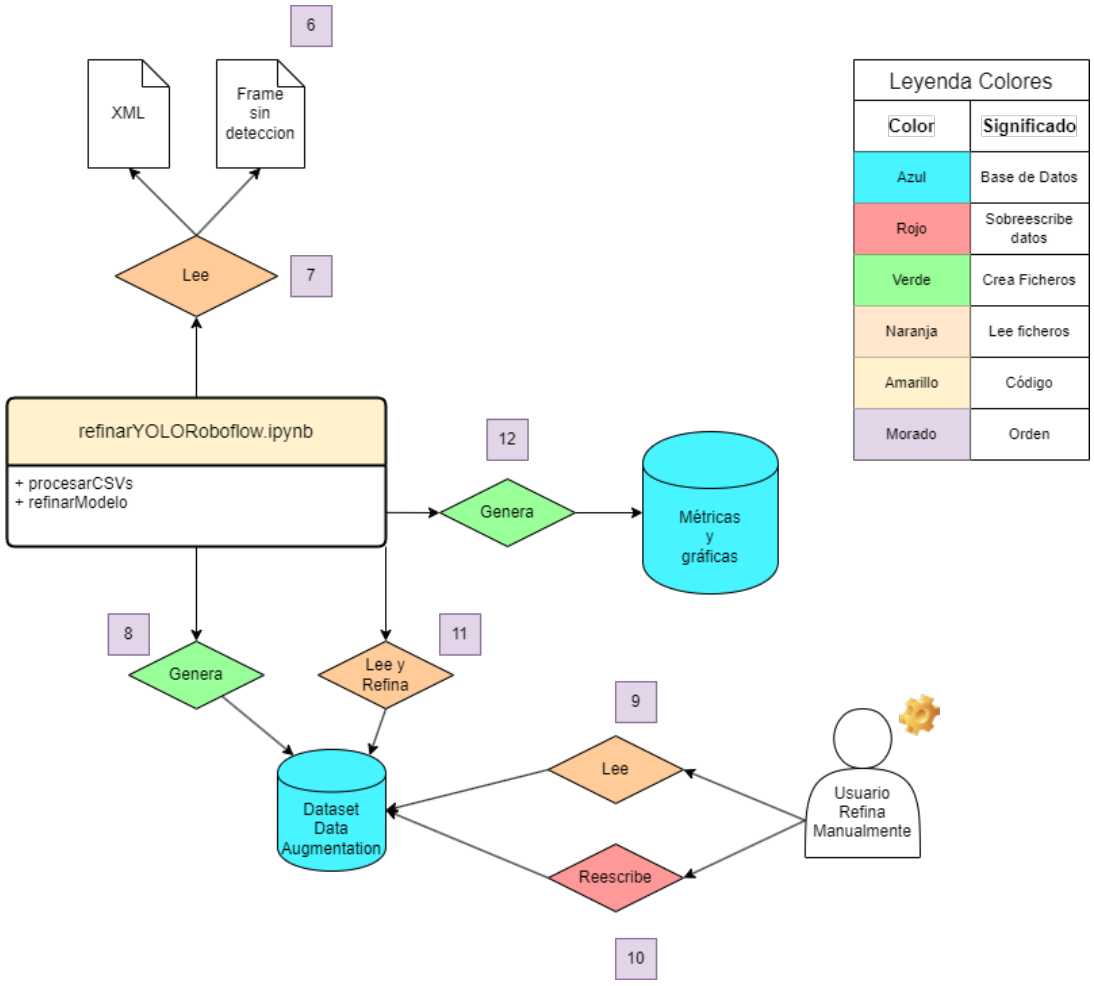


Figura 3.13: Esquema del flujo de trabajo del refinamiento de los modelos YOLO

1. Desde el fichero "`refinarYOLORoboflow`", se leerán tanto las detecciones guardadas por fotogramas contenidas en los "`.xml`" como los fotogramas a los que hacen referencia. Estos se utilizarán para crear un archivo "`.json`" con formato COCO (Common Objects in Context), el cual la herramienta RoboFlow es capaz de interpretar.
2. Una vez subida las imágenes y el fichero "`.json`", el usuario deberá revisar y reescribir las imágenes cuyas detecciones se hayan podido corromper durante la subida.
3. En el fichero "`refinarYOLORoboflow`", se utilizarán las detecciones revisadas para entrenar el modelo YOLO de nuestra elección.
4. Durante el entrenamiento del modelo, se generarán las métricas y las gráficas

correspondientes al entrenamiento.

3.4.1. YOLOv8n

Al finalizar el entrenamiento, en la Figura 3.14 se pueden observar las siguientes métricas de rendimiento del modelo "`YOLOv8n`"

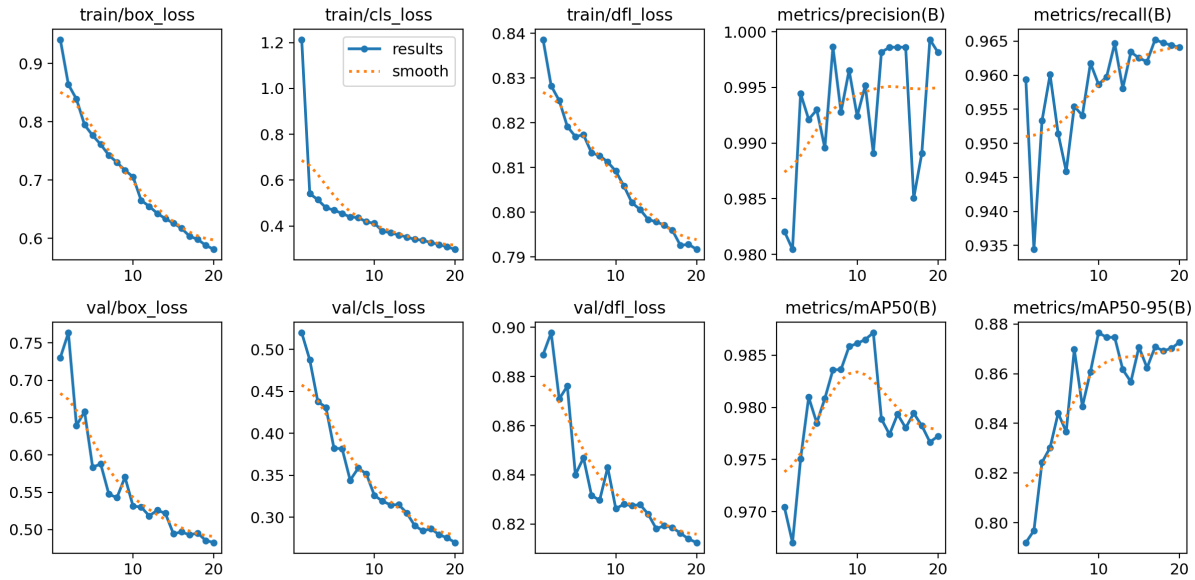


Figura 3.14: Métricas obtenidas durante el entrenamiento del modelo "`YOLOv8n`" refinado.

A partir de las métricas de la Figura 3.14, se puede deducir lo siguiente:

1. mAP50 de 0.986: Implica que el 98,6 % de las veces que haya una pelota en la imagen el modelo la va a detectar. Entre otras causas, el porcentaje faltante para una precisió perfecta, se puede justificar debido a la existencia de imágenes en la base de datos donde la pelota está entrando o saliendo de la misma y solo aparece en pocos píxeles, con los que el modelo no es capaz de detectar la pelota (Figura 3.15, Figura 3.16).

Figura 3.15: Imagen en la que la pelota está saliendo de la imagen.

Figura 3.16: Imagen en la que la pelota está entrando en la imagen.

2. mAP50-95 de 0.877: Implica que una vez detectadas las pelotas las cajas con las que se marca su posición tiene un porcentaje de precisión del 87,7 %, valor que se puede deber a que, al tener la base de datos, un considerable número de imágenes etiquetadas manualmente, estas pueden tener pequeñas desviaciones por error humano que el modelo no comete, pero al tomar como referencia los valores humanos, se cuenta como desviación la diferencia del etiquetado del modelo respecto a las humanas.

3.4.2. YOLOv11

Al finalizar el entrenamiento, en la Figura 3.17 se pueden observar las siguientes métricas de entrenamiento del modelo "`YOLOv11n`".

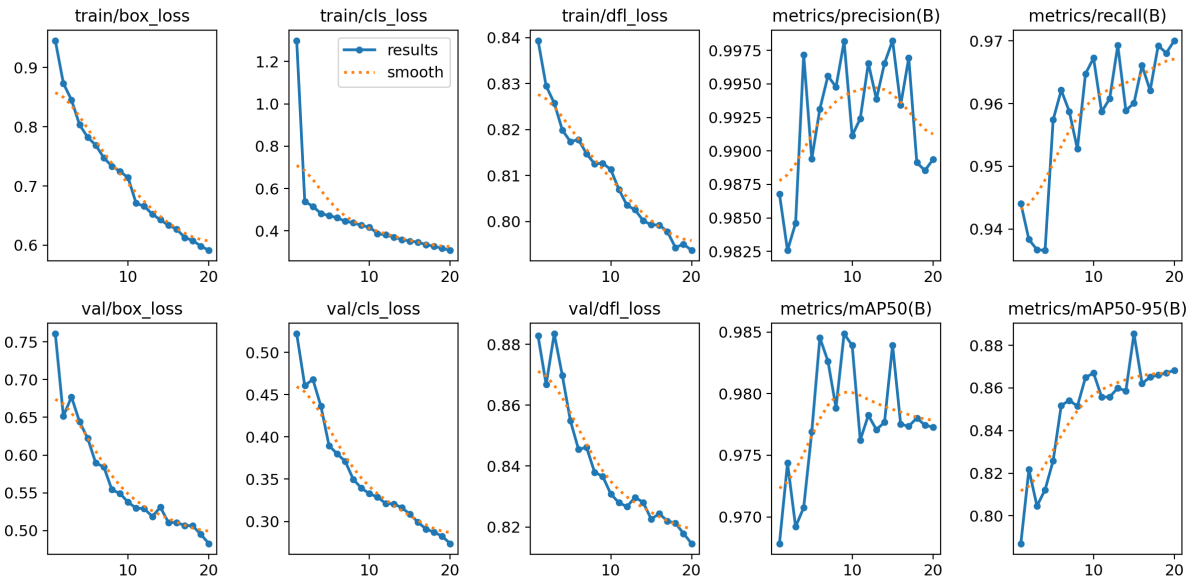


Figura 3.17: Métricas obtenidas durante el entrenamiento del modelo "`YOLOv11n`" refinado.

A partir de las métricas que se encuentran en la Figura 3.17 se puede deducir lo siguiente:

1. **mAP50 de 0.984:** implica que el 98,4 % de las veces que haya una pelota en la imagen el modelo la va a detectar. Al igual que en el modelo "`YOLOv8n`", este valor se puede justificar por la existencia de imágenes en la base de datos donde la pelota está entrando o saliendo de la escena y solo aparece en pocos píxeles.
2. **mAP50-95 de 0.885:** implica que, una vez detectadas las pelotas, las cajas con las que se marca su posición tienen un porcentaje de precisión del dibujado que supera en un 2 % al modelo "`YOLOv8n`".

3.5. Experimentos usando arckpadel_detection [1]

El modelo "`arckpadel_detection`" es un modelo de visión por ordenador cedido por la empresa SPORTS RTD HUB S.L., desarrollado en el trabajo de fin de master

"Detección y estimación de métricas sobre una pelota de pádel usando cámaras de alta velocidad" [1].

Para comparar este modelo con los previamente explicados, se han utilizado los mismos vídeos empleados en la evaluación de los modelos refinados, los cuales no fueron utilizados durante su entrenamiento.

Para obtener las métricas del modelo "`arckpadel_detection`", se sigue el esquema de la Figura 3.18.

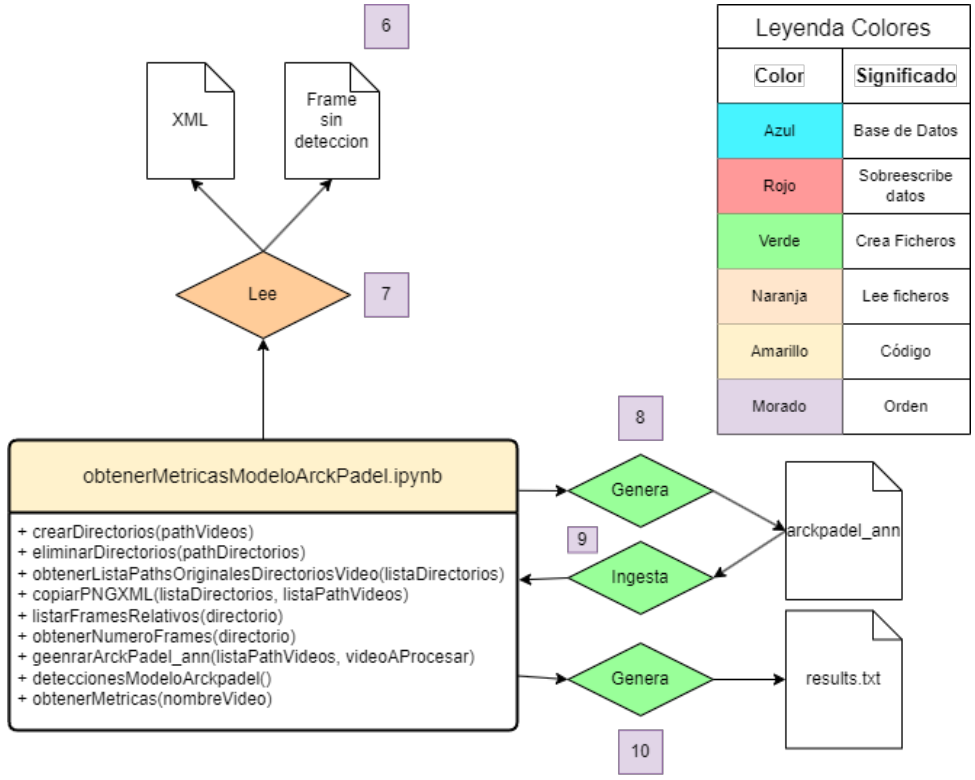


Figura 3.18: Esquema del workflow para obtener las métricas del modelo "`arckpadel_detection`".

1. El proceso comienza con la ejecución del fichero "`obtenerMetricasModeloArckPadel`", el cual se encarga de leer y procesar los archivos "`.xml`". Estos archivos contienen las anotaciones estructuradas que indican en qué en qué posiciones ha sido detectada la pelota en un frame. Al mismo tiempo, se extraen los fotogramas correspondientes a dichas detecciones, lo que permite asociar las anotaciones con las imágenes originales del vídeo. A continuación se genera por cada vídeo un fichero de anotaciones que utiliza el modelo "`arckpadel_detection`" para realizar las detecciones de la pelota.

2. Una vez que el modelo ha ejecutado su proceso de detección sobre los distintos vídeos y ha generado los resultados correspondientes, se procede al cálculo de las diferentes métricas de evaluación como el mAP50 y la precisión (Sección 3.2). Estas métricas se guardan en ficheros "`results.txt`".

3.6. Comparativa entre detectores

3.6.1. Precisión

La precisión de los modelos se muestra en la Figura 3.19, Figura 3.20 y Figura 3.21, donde se puede observar que en todos los umbrales de IoU, los modelos YOLO tienen una precisión superior al modelo "`arckpadel_detection`", aunque la diferencia de precisión se empieza a hacer más notable a partir del límite 0.8, límite a partir del cual la precisión del modelo "`arckpadel_detection`", comienza a decaer rápidamente mientras que la precisión de los modelos YOLO se mantiene más alta en los límites más avanzados.

IoU threshold	Recall	Precision	AP	mAP
0.5	95.372 %	95.110 %	93.149 %	93.149 %
0.55	94.821 %	94.560 %	92.396 %	92.396 %
0.6	94.545 %	94.286 %	91.966 %	91.966 %
0.65	93.939 %	93.681 %	91.032 %	91.032 %
0.7	93.444 %	93.187 %	90.351 %	90.351 %
0.75	92.948 %	92.692 %	89.546 %	89.546 %
0.8	91.965 %	91.703 %	88.142 %	88.142 %
0.85	84.518 %	84.286 %	75.314 %	75.314 %
0.9	48.815 %	48.681 %	24.052 %	24.052 %
0.95	10.193 %	10.165 %	1.062 %	1.062 %

Figura 3.19: Métricas de rendimiento por umbral de IoU "`arckpadel_detection`". [1]

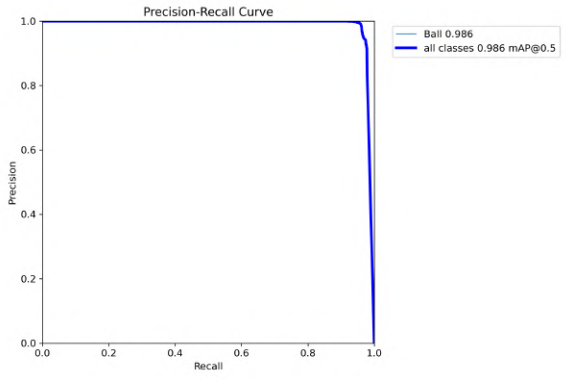


Figura 3.20: Métricas Precisión - Recall "`YOLOv8n`".

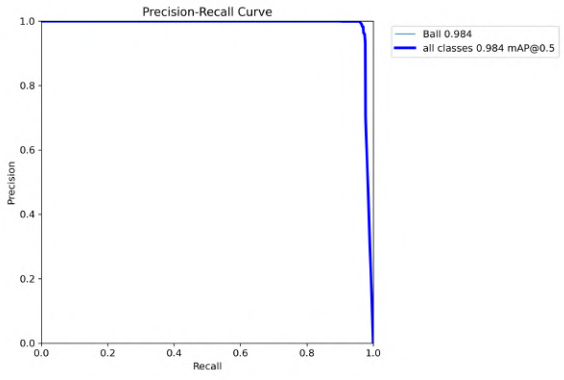


Figura 3.21: Métricas Precisión - Recall "`YOLOv11n`".

A partir de la Figura 3.20 y Figura 3.21, se puede concretar la Tabla 3.1, que muestra la diferencia de precisión entre el modelo "`YOLOv8n`" y "`YOLOv11n`" .

A partir de la tabla, se puede observar que, aunque casi idénticos, el modelo "`YOLOv8n`" es levemente superior en cuanto a la precisión de detección. Esta diferencia en el rendimiento puede deberse a que, durante la fase de predetección de imágenes empleada para la creación de la base de datos utilizada en el refinamiento (Sección 3.1.1), se utilizó el modelo "`YOLOv8l`". Como consecuencia, las predicciones del modelo `YOLOv8n`" tienden a estar más alineadas con dichas anotaciones que las del modelo "`YOLOv11n`".

Métrica / Modelo	YOLOv8n	YOLOv11n
mAP0.5 (all classes)	0.986	0.984
Umbral para Precisión = 1.0	0.842	0.839
Recall máximo (conf = 0)	0.97	0.97

Cuadro 3.1: Comparativa de métricas entre modelos "YOLOv8n" y "YOLOv11n" refinados.

3.6.2. Tiempo de cómputo

Cuando se procesa un vídeo para la detección de objetos no solo es importante la precisión, sino también la velocidad de detección, y sobre todo, si se trata de proyectos como este, donde los modelos deben procesar vídeos capturados por cámaras especiales de alto frame rate. La decisión de utilizar el modelo más ligero y por tanto más rápido tanto de "YOLOv8n" como "YOLOv11n", está contemplada por la importancia de la velocidad de detección. Cuando se han procesado los diferentes vídeos, se ha guardado el tiempo utilizado por fotograma. Además, dado que el modelo "arckpadel_detection" solo se puede ejecutar en CPU (Tabla 3.2), los modelos YOLO se han evaluado tanto en GPU (tablas 3.3 y 3.5) como en CPU (tablas 3.4 y 3.6). Esto permite, no solo realizar una comparación directa con arckpadel_detection", sino también analizar la ganancia en términos de tiempo de procesamiento que se obtiene al utilizar aceleración por GPU frente a la ejecución en CPU.

- **Modelo arckpadel_detection:**

Vídeo	Total de Fotogramas	Tiempo CPU	Tiempo medio por fotograma
00001	1689	0:019919 (min:seg)	0.00071 s
00002	2126	0:0155198 (min:seg)	0.00073 s
00003	1843	0:0132696 (min:seg)	0.00072 s

Cuadro 3.2: Tiempo de procesamiento "arckpadel_detection" ejecutado en CPU.

- **Modelo YOLOv8n:**

Vídeo	Total de Fotogramas	Tiempo GPU	Tiempo medio por fotograma
00001	1689	0:17 (min:seg)	0.0103 s
00002	2126	0:21 (min:seg)	0.0101 s
00003	1843	0:18 (min:seg)	0.0101 s

Cuadro 3.3: Tiempo de procesamiento del modelo "YOLOv8n" ejecutado en GPU.

Vídeo	Total de Fotogramas	Tiempo CPU	Tiempo medio por fotograma
00001	1689	1:00 (min:seg)	0.0358 s
00002	2126	1:28 (min:seg)	0.0363 s
00003	1843	1:23 (min:seg)	0.0401 s

Cuadro 3.4: Tiempo de procesamiento del modelo "YOLOv8n" ejecutado en CPU.

- **Modelo YOLOv11n:**

Vídeo	Total de Fotogramas	Tiempo CPU	Tiempo medio por fotograma
00001	1689	0:17 (min:seg)	0.0103 s
00002	2126	0:18 (min:seg)	0.0087 s
00003	1843	0:15 (min:seg)	0.0084 s

Cuadro 3.5: Tiempo de procesamiento "`YOLOv11n`" ejecutado en GPU.

Vídeo	Total de Fotogramas	Tiempo CPU	Tiempo medio por fotograma
00001	1689	0:53 (min:seg)	0.0317 s
00002	2126	1:21 (min:seg)	0.0343 s
00003	1843	1:05 (min:seg)	0.0357 s

Cuadro 3.6: Tiempo de procesamiento del modelo "`YOLOv11n`" ejecutado en CPU.

Utilizando el promedio de tiempo por iteración de todos los modelos ejecutados en CPU (tabla 3.2, tabla 3.4 y tabla 3.6) y sus precisiones (Figura 3.19, Figura 3.20 y Figura 3.21) , se obtiene la tabla 3.7.

Modelo	mAP50	Tiempo medio por fotograma
arckpadel_detection	0.47686	0.00072 s
YOLOv8n	0.986	0.0383 s
YOLOv11n	0.984	0.0339 s/it

Cuadro 3.7: Comparación del mAP y el tiempo promedio por fotograma de los modelos evaluados.

Al comparar en la tablas los tiempos, se determina que el modelo más rápido es el "`arckpadel_detection`", con tiempos por fotograma del orden de 10^{-4} segundos, en contraste con los 10^{-3} segundos registrados por los modelos basados en YOLO ejecutados en GPU, lo que representa una diferencia de una órden de magnitud. A continuación se sitúa el modelo `YOLOv11n`", que presenta un rendimiento ligeramente superior en velocidad respecto a "`YOLOv8n`". En comparación con el modelo "`arckpadel_detection`", en los modelos YOLO se aprecia que el tiempo por iteración aumenta junto a la precisión obtenida.

Cuando se compara los resultados de los tiempos según el uso de CPU o GPU en los modelos "`YOLOv8n`" y "`YOLOv11n`", se puede determinar que el uso de GPU permite, en estos modelos, mejorar los tiempos de ejecución en un $\approx 70\,\%$.

3.6.3. Matrices de confusión

Las matrices de confusión de la fase de validación (Sección 3.1.3), están formadas por dos clases, 'ball' y 'background', esto se debe a que el modelo llama 'ball' a la clase que representa a la pelota y 'background' a todo lo que no sea la pelota.

De esta forma, respecto al modelo "`YOLOv8n`", en la Figura 3.22 se observa que de un total de 1524 detecciones realizadas por el modelo, en 1447 ocasiones se detecta correctamente la pelota, representando un 0.96 % (Figura 3.23) de éxito respecto al

número total de instancias en las que la pelota debía ser detectada. De la misma manera en 21 ocasiones se detecta la pelota cuando realmente esta no se encuentra en la imagen y finalmente, en 56 ocasiones no se detecta la pelota en la imagen.

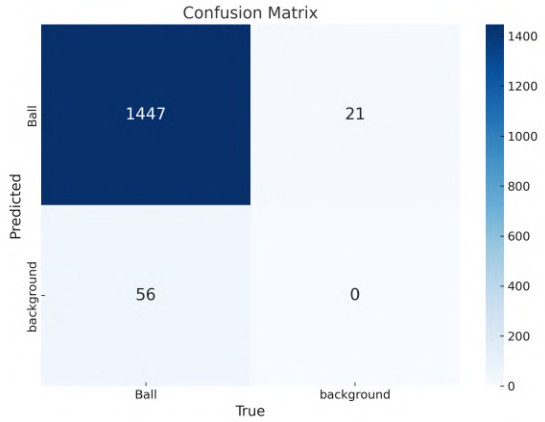


Figura 3.22: Matriz de confusión del modelo "YOLOv8n" refinado.

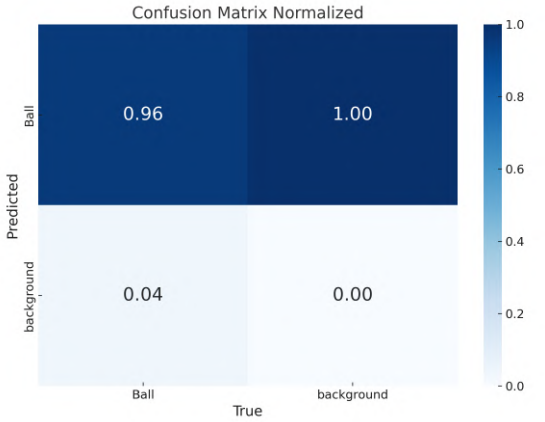


Figura 3.23: Matriz de confusión normalizada del modelo "YOLOv8n" refinado.

En cuanto al modelo "YOLOv11n", sus matrices de confusión de la fase de validación mostradas en la Figura 3.24 y Figura 3.25 muestran que de un total de 1524 detecciones realizadas por el modelo, en 1453 ocasiones se detecta correctamente la pelota, representando un 0.97 % de éxito respecto al número total de instancias en las que la pelota debía ser detectada. De la misma manera en 21 ocasiones se detecta la pelota cuando realmente esta no se encuentra en la imagen y finalmente, en 50 ocasiones no se detecta la pelota en la imagen cuando realmente sí se encuentra en la misma.

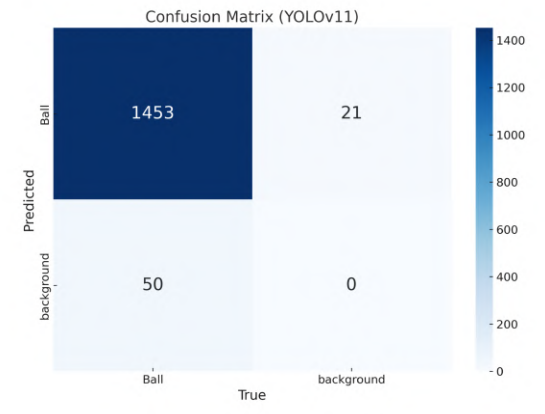


Figura 3.24: Matriz de confusión del modelo "YOLOv11n".

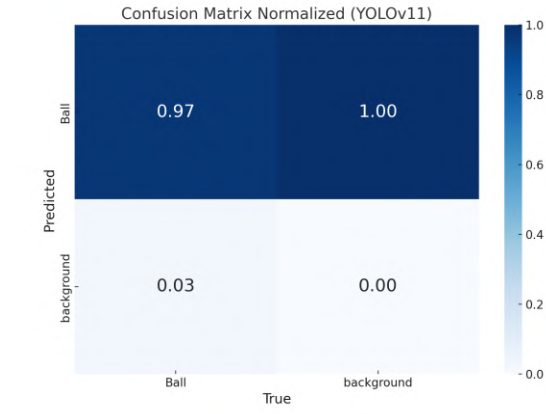


Figura 3.25: Matriz de confusión normalizada del modelo "YOLOv11n".

Respecto al modelo "arckpadel_detection", sus matrices de confusión se pueden observar en la Figura 3.26 y en la Figura 3.27.

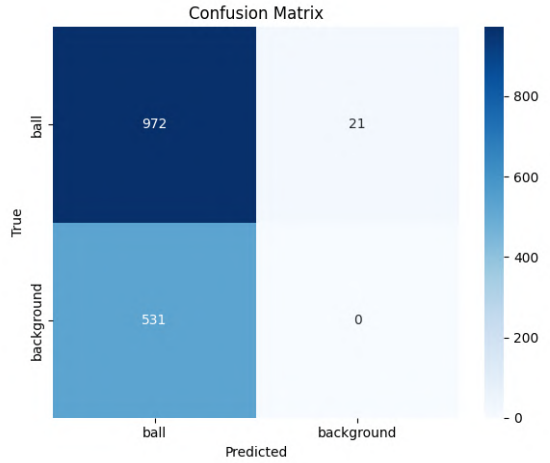


Figura 3.26: Matriz de confusión del modelo "`arckpadel_detection`".

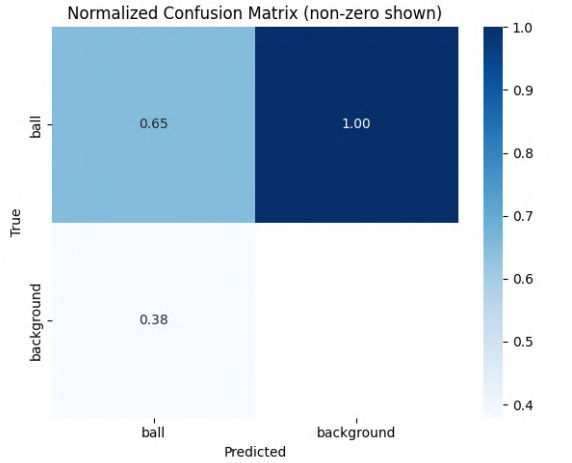


Figura 3.27: Matriz de confusión normalizada del modelo "`arckpadel_detection`".

Los resultados de detección correcta más bajos que en los modelos anteiores se deben a que el modelo "`arckpadel_detection`" no es tan robusto frente a cambios de luces y oclusiones como los modelos refinados de "`YOLOv8n`" y "`YOLOv11n`", y al estar la base de datos formada por una elevada cantidad de fotogramas modificados para forzar estas condiciones adversas, la precisión del modelo "`arckpadel_detection`" disminuye notablemente.

3.7. Aplicación de usuario

Para facilitar el uso del modelo de detección de objetos mediante YOLO, se ha desarrollado una interfaz gráfica a modo de aplicación que automatiza todos los pasos necesarios para ejecutar cada modelo en cualquier vídeo de entrada.

La Figura 3.28 muestra lo que se ve cuando se ejecuta la aplicación por primera vez.

En la imagen se pueden observar lo siguiente:

1. Desplegable (arriba izquierda): Se trata de un desplegable que cuenta con todos los vídeos que se pueden procesar. Cuando se selecciona un vídeo, el nombre del mismo se muestra justo debajo del desplegable.
2. Vídeo superior: Es el vídeo seleccionado en el desplegable. Se puede reprocucir y aumentar de tamaño por si se quiere ver con más detalle.
3. Vídeo inferior: Aquí se reproducirá el vídeo procesado con las detecciones de la pelota una vez esté disponible.
4. Deplegable (arriba derecha): Permite elegir si se quiere procesar el vídeo con el modelo "`YOLOv8n`" o "`YOLOv11n`".
5. Botón "Procesar Vídeo": Al pulsar este botón, el vídeo seleccionado será procesado por el modelo seleccionado en el desplegable que se encuentra debajo y

generará un vídeo con las detecciones dibujadas, el cual, se representará en la posición del vídeo inferior.

6. Botón “Procesar Oclusiones”: Cuando se pulsa, se buscan oclusiones automáticamente en el vídeo. Si no hay oclusiones se indicará, pero si hay oclusiones aparecerán los fotogramas correspondientes a la oclusión con la predicción.

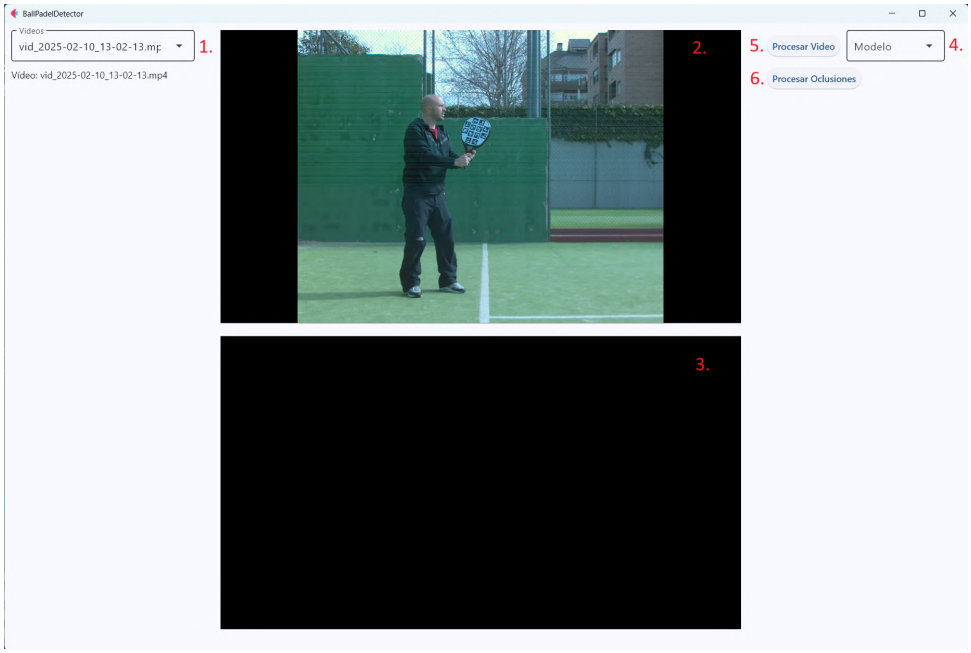


Figura 3.28: Interfaz de la aplicación.

De esta forma si seguimos los pasos anteriores se pueden observer los resultados de las figuras 3.29, 3.30, 3.31y 3.32:

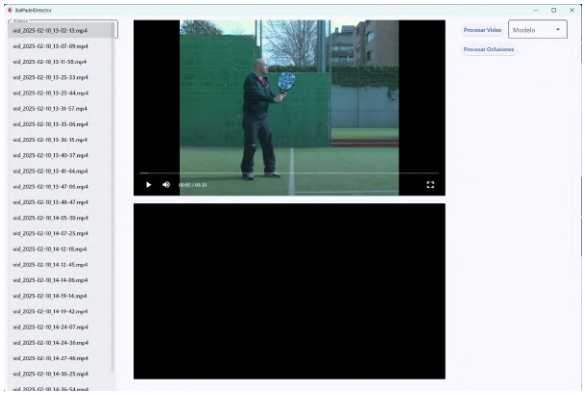


Figura 3.29: Ejemplo de los vídeos seleccionables en el desplegable.

Figura 3.30: Vídeo procesado con las detecciones que se muestra cuando se pulsa el botón “Procesar Video”.

Figura 3.31: Ejemplo de la indicación de que no hay oclusiones en un vídeo.

Figura 3.32: Ejemplo de cómo se ven las imágenes que corresponden a las oclusiones con sus detecciones dibujadas.

El funcionamiento de la aplicación de escritorio sigue el esquema de la Figura 3.33

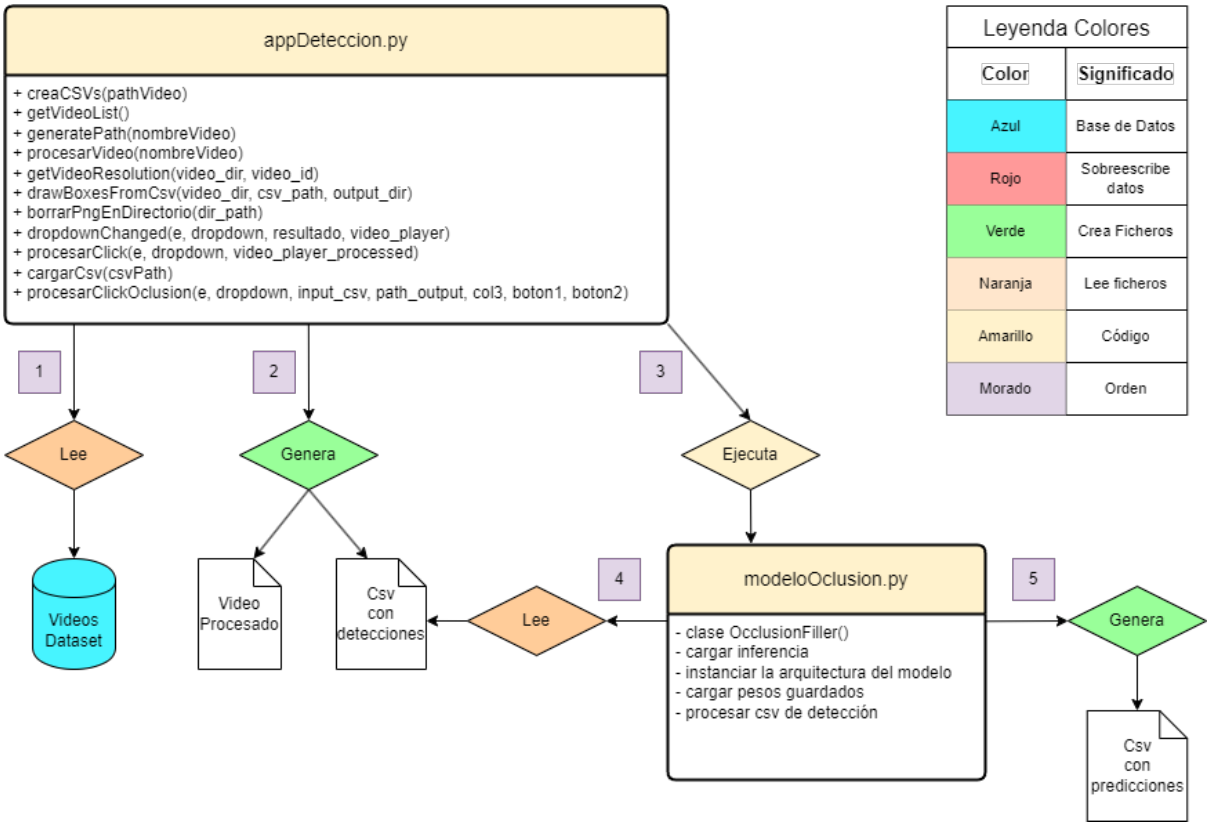


Figura 3.33: Esquema del flujo de trabajo del funcionamiento de la aplicación.

1. El fichero "`appDeteccion.py`", cuando se selecciona en el desplegable un vídeo, este se lee.
2. Una vez pulsado el botón "Procesar Video", se genera el vídeo con las detecciones que se muestra en la aplicación y un fichero "`.csv`" con las detecciones

realizadas.

3. Cuando se pulsa el botón “Procesar Oclusiones”, se inicia una subrutina, donde se manda a ejecutar el fichero “`modeloOclusion.py`”.

4. El fichero “`modeloOclusion.py`” leerá el fichero “`.csv`” generado por “`appDeteccion.py`” y lo usará para predecir las oclusiones, generando un último “`.csv`” con las predicciones hechas.

Capítulo 4

Conclusiones

Este trabajo demuestra que el uso del detector de objetos YOLO para localizar la pelota de pádel en secuencias de video, mejora la aproximación base que usa aprendizaje automático tradicional ("`arckpadel_detection`") [1]. Para realizar los experimentos correspondientes, se han etiquetado 25 videos, de los cuales han sido utilizados 17 para entrenar el detector y 8 para evaluar su rendimiento. Tras realizar este costoso proceso de anotación, se ha entrenado un modelo "`YOLOv11n`" que obtiene un mAP de 0.986 frente al 0.47686 que obtiene `arckpadel_detection`. No obstante, sin hacer uso de ninguna GPU, el modelo "`YOLOv11n`" procesa cada imagen a 29.52 FPS (hasta 119.81 FPS usando la GPU modelo NVIDIA 4070 Super), mientras que el modelo `arckpadel_detection` lo hace mucho más rápido a 138.889 FPS. Dado que el objetivo del TFG es localizar la pelota en condiciones de captura adversas como en la Figura 2.9 o Figura 1.3, es preferible hacer uso de YOLOv11.

4.1. Trabajo futuro

Gracias a las métricas que se pueden obtener con los modelos refinados, se propone abordar la detección de la raqueta y de las personas en juego siguiendo el flujo de trabajo definido en este proyecto. De esta forma se podría obtener un sistema de agentes que obtuviera mediciones de todos los elementos que participan en un partido, permitiendo así, que los deportistas tengan un nuevo medio de aprendizaje. Se propone además a realizar una representación en 3D de la posición de la pelota, permitiendo tener una simulación virtual de lo que ocurre en pista.

4.2. Impacto ético

El impacto ético de este proyecto basado en la detección de la pelota de pádel mediante visión por computador, es relativamente bajo comparado con otros sistemas de inteligencia artificial, pero merece ser analizado en varios ámbitos:

- Transparencia y explicabilidad:
  El algoritmo desarrollado se basa en modelos de código abierto (YOLO). El funcionamiento del mismo está completamente explicado y está disponible de forma pública, lo que facilita su entendimiento y uso.

- Privacidad:
  La base de datos utilizada para el entrenamiento contiene rostros identificables, sin embargo para acceder al mismo se necesita ser un desarrollador acreditado por la empresa SPORTS RTD HUB S.L.. Además como el código desarrollado está pensado para poder aplicarse a cualquier video donde puedan aparecer otras personas, pues el sistema no guarda métricas que no sean propias de la pelota.
- Responsabilidad y control humano:
  Dado que las predicciones de los modelos pueden influir en consejos de entrenamiento o decisiones comerciales, conviene clarificar, que la herramienta ofrece soporte al analista o entrenador encargado, y no un veredicto definitivo.
- Riesgo GPIA:
  Según la clasificación del Reglamento de IA de la UE, el sistema no califica como "alto riesgo" (GPIA), pues, no impacta decisiones críticas de seguridad, salud, empleo, crédito ni identificaciones biométricas en tiempo real. Además su uso se circunscribe al deporte y al análisis de vídeo, ámbitos de bajo impacto social [54].

Bibliografía

[1] Jaime Sánchez Cotta. «Detección y estimación de métricas sobre una pelota de pádel usando cámaras de alta velocidad». Tesis de mtría. Universidad Politécnica de Madrid, ETSI Informáticos, 2024.

[2] Kron Technologies. *Chronos 1.4 high speed camera*. [Online]. 2024. URL: https://www.krontech.ca/product/chronos-1-4-high-speed-camera/?gad_source=1&gclid=CjwKCAjwjqWzBhAqEiwAQmtgT0m001s0N27QQlkglw_HD2yGBdi3I_PwzojTCsjVpZuBGCXDRZBxchoCvbcQAvD_BwE&v=04c19fa1e772.

[3] Victor Molla. *Redes Neuronales Convolucionales: Funcionamiento y Aplicaciones*. [Online]. URL: https://www.victormolla.com/redes-neuronales-convolucionales (visitado 14-04-2025).

[4] Ross Girshick. «Fast R-CNN». En: *The Computer Vision Fundation* (2015).

[5] Andrés Jiménez Láinez y María Dolores Pérez Godoy. «Experimentación con modelos de Deep Learning para la detección de objetos». En: *Departamento de Informática. Universidad de Jaén* (2022).

[6] José Luis Magaña Vázquez. «MODELO DE APRENDIZAJE PROFUNDO PARA LA DETECCIÓN DE OBJETOS DE INTERÉS EN EL BOSQUE DE LA PRIMAVERA». En: *Instituto Tecnológico y de Estudios Superiores de Occidente* (2022).

[7] Jonathan Hui. «YOLOv4». En: *Medium* (2020).

[8] Ultralytics. *Evento YOLOVision - Presentación general de modelos YOLO (es)*. [Online]. URL: https://www.ultralytics.com/es/events/yolovision (visitado 18-02-2025).

[9] Joseph Redmon y Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». En: *University of Washington* (2015). [Online].

[10] Ultralytics. *Ultralytics Documentation*. [Online]. URL: https://docs.ultralytics.com/ (visitado 01-02-2025).

[11] Ultralytics. *Guía de inicio para usar modelos YOLO - Documentación Ultralytics (es)*. [Online]. URL: https://docs.ultralytics.com/es/#where-to-start (visitado 03-04-2025).

[12] Julio-Alejandro Romero-González Juan Terven Diana-Margarita Córdoba Esperanza. «A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS». En: *mdpi* (2023).

[13] Ultralytics. *Base de datos COCO*. [Online]. URL: https://docs.ultralytics.com/es/datasets/detect/coco/#key-features (visitado 11-04-2025).

[14] Ultralytics. *YOLOv3 - Documentación del modelo (es)*. [Online]. URL: https://docs.ultralytics.com/es/models/yolov3/ (visitado 08-04-2025).

[15] Ultralytics. *YOLOv4 - Documentación del modelo (es)*. [Online]. URL: https://docs.ultralytics.com/es/models/yolov4/ (visitado 08-04-2025).

[16] A. *Resumen de arquitecturas CNN: VGG, ResNet, DenseNet, MobileNet, EfficientNet y YOLO*. [Online]. 2024. URL: https://medium.com/@jaguuai/cnn-vgg-resnet-densenet-mobilenet-effecientnet-and-yolo-2329a9fa2d0f.

[17] Ultralytics. *YOLOv4 Model Overview*. [Online]. URL: https://docs.ultralytics.com/ (visitado 04-02-2025).

[18] Irma Amelia Dewi Marsa Mahasin. «Comparison of CSPDarkNet53, CSPResNeXt-50, and EfficientNet-B0 Backbones on YOLO V4 as Object». En: *International Journal of Engineering, Science InformationTechnology (IJESTY)* (2022).

[19] Then Yung, NigelandDale and Wong, W. K., Juwono, Filbert H. y Sim, Zee Ang. «Safety Helmet Detection Using Deep Learning: Implementation and Comparative Study Using YOLOv5, YOLOv6, and YOLOv7». En: *International Conference on Green Energy, Computing and Sustainable Technology (GECOST)* (2022).

[20] Ultralytics. *YOLOv5 - Documentación del modelo (es)*. [Online]. URL: https://docs.ultralytics.com/es/models/yolov5/ (visitado 11-04-2025).

[21] Ultralytics. *YOLOv8 Model Documentation*. [Online]. URL: https://docs.ultralytics.com/ (visitado 23-02-2025).

[22] Ultralytics. *YOLOv8 Performance Metrics*. [Online]. URL: https://docs.ultralytics.com/ (visitado 23-02-2025).

[23] Aarón Schcolnik-Elias et al. «Detección de armas tipo pistola mediante el uso de redes convolucionales con una arquitectura tipo YOLO y estereoscopía». En: *Instituto Tecnológico de La Paz* (2023).

[24] Ultralytics. *YOLOv11 Model in Spanish*. [Online]. URL: https://docs.ultralytics.com/ (visitado 05-03-2025).

[25] Ultralytics. *YOLOv11 Performance Metrics*. [Online]. URL: https://docs.ultralytics.com/ (visitado 02-03-2025).

[26] Ultralytics. *YOLOv11 Model Documentation*. [Online]. URL: https://docs.ultralytics.com/ (visitado 05-03-2025).

[27] Eun Kyeong Kim et al. «Data Augmentation Method by Applying Color Perturbation of Inverse PSNR and Geometric Transformations for Object Recognition Based on Deep Learning». En: *Department of Electrical and Computer Engineering, Pusan National University* (2020). URL: https://www.mdpi.com/2076-3417/10/11/3755.

[28] Ultralytics. *Ajuste de hiperparámetros en modelos YOLO - Guía Ultralytics (es)*. [Online]. URL: https://docs.ultralytics.com/es/guides/hyperparameter-tuning/ (visitado 11-04-2025).

[29] Ultralytics. *Guía de ajuste de hiperparámetros*. [Online]. URL: https://docs.ultralytics.com/es/guides/hyperparameter-tuning/#default-search-space-description (visitado 11-04-2025).

[30] Imran Khan Mirani, Chen Tianhua, Malak Abid Ali Khan, Syed Muhammad Aamir y Waseef Menhaj. «Object Recognition in Different Lighting Conditions at Various Anglesby Deep Learning Method». En: *Department of Electronic Information Engineering, Anhui Normal University, Wuhu, 241000, China* (2022).

[31] Ultralytics. *YOLOv5 Supported Tasks and Modes*. [Online]. URL: https://docs.ultralytics.com/ (visitado 04-02-2025).

[32] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe y Youngjoon Yoo. «CutMix: Regularization Strategy to Train Strong Classifiers-with Localizable Features». En: *Yonsei University* (2019).

[33] Junsoo Oh, Chulhee Yun. «Provable Benefit of Cutout and CutMix for Feature Learning». En: *38th Conference on Neural Information Processing Systems (NeurIPS)*. (2024).

[34] Diego Camilo Celada Lozada, Julián René Chaux y Karol Johana Zambrano Cruz. «Detección y control de aforo en ambientes del Centro de la industria, la Empresa y los Servicios, implementando YOLO V5, Deep SORT». En: *Revista EDIA* (2022).

[35] José Alano Peres de Abreu, Roberto Célio Limão de Oliveira y João Viana da Fonseca Neto. «Rocket tracking impact point prediction using -, standard Kalman, extended, Kalman, and unscented Kalman filters». En: *Dialnet* (2020).

[36] Isell. *LabelImg: Herramienta de etiquetado gráfico para anotaciones en imágenes*. [Online]. URL: https://github.com/HumanSignal/labelImg (visitado 20-04-2025).

[37] Roboflow. *Roboflow: Plataforma para gestión y entrenamiento de datasets de visión por computador*. [Online]. URL: https://roboflow.com/ (visitado 20-04-2025).

[38] Roboflow. *Uso de la REST API de Roboflow*. [Online]. URL: https://docs.roboflow.com/developer/rest-api/using-the-rest-api (visitado 25-04-2025).

[39] AWS. *Visión general del formato COCO en Amazon Rekognition*. [Online]. URL: https://docs.aws.amazon.com/es_es/rekognition/latest/customlabels-dg/md-coco-overview.html (visitado 08-04-2025).

[40] Peña Vargas y Tomás Carlos. «Modelando la dirección del precio y retornos de las acciones chilenas usando redes neuronales recurrentes - LSTM». En: *Repositorio academécido de la Universidad de Chile* (2022).

[41] Bauz et al. «Análisis de redes neuronales recurrentes LSTM y modelos de vectores autorregresivos VAR en la generación de pronósticos de series de tiempo aplicado a las ventas de una empresa dedicada a la comercialización de productos de consumo masivo». En: *Repositorio academécido de la Universidad de Chile* (2023). URL: http://www.dspace.espol.edu.ec/handle/123456789/58549.

[42] Data Camp. *Tutorial de LSTM en Python para predicción en bolsa*. [Online]. URL: https://www.datacamp.com/es/tutorial/lstm-python-stock-market (visitado 01-05-2025).

[43] E. Berezin, E. Pikalov y K. Palaguta. «Features of Preparing a Data Set for Training a Neural Network of an Object Tracking System in a Stream». En: *IEEE* (2023).

[44] PyTorch. *Guía de instalación de PyTorch*. [Online]. URL: https://pytorch.org/get-started/locally/ (visitado 01-05-2025).

[45] NumPy. *Documentación oficial de NumPy 2.2*. [Online]. URL: https://numpy.org/doc/2.2/ (visitado 03-05-2025).

[46] Matplotlib. *Documentación oficial de Matplotlib*. [Online]. URL: https://matplotlib.org/stable/index.html (visitado 03-05-2025).

[47] OpenCV. *Curso gratuito de OpenCV University*. [Online]. URL: https://opencv.org/university/free-opencv-course/?utm_source=opcv&utm_medium=menu&utm_campaign=obc (visitado 05-05-2025).

[48] Pandas. *Documentación oficial de Pandas*. [Online]. URL: https://pandas.pydata.org/ (visitado 05-05-2025).

[49] Pillow. *Documentación de Pillow (PIL Fork)*. [Online]. URL: https://pillow.readthedocs.io/en/stable/ (visitado 25-04-2025).

[50] Ultralytics. *API de inferencia de Ultralytics Hub*. [Online]. URL: https://docs.ultralytics.com/hub/inference-api/#dedicated-inference-api (visitado 02-05-2025).

[51] Flet. *Flet: Framework para apps con interfaz de usuario en Python*. [Online]. URL: https://flet.dev/ (visitado 01-05-2025).

[52] NVIDIA. *NVIDIA CUDA Toolkit: Herramientas para computación paralela*. [Online]. URL: https://developer.nvidia.com/cuda-toolkit (visitado 03-05-2025).

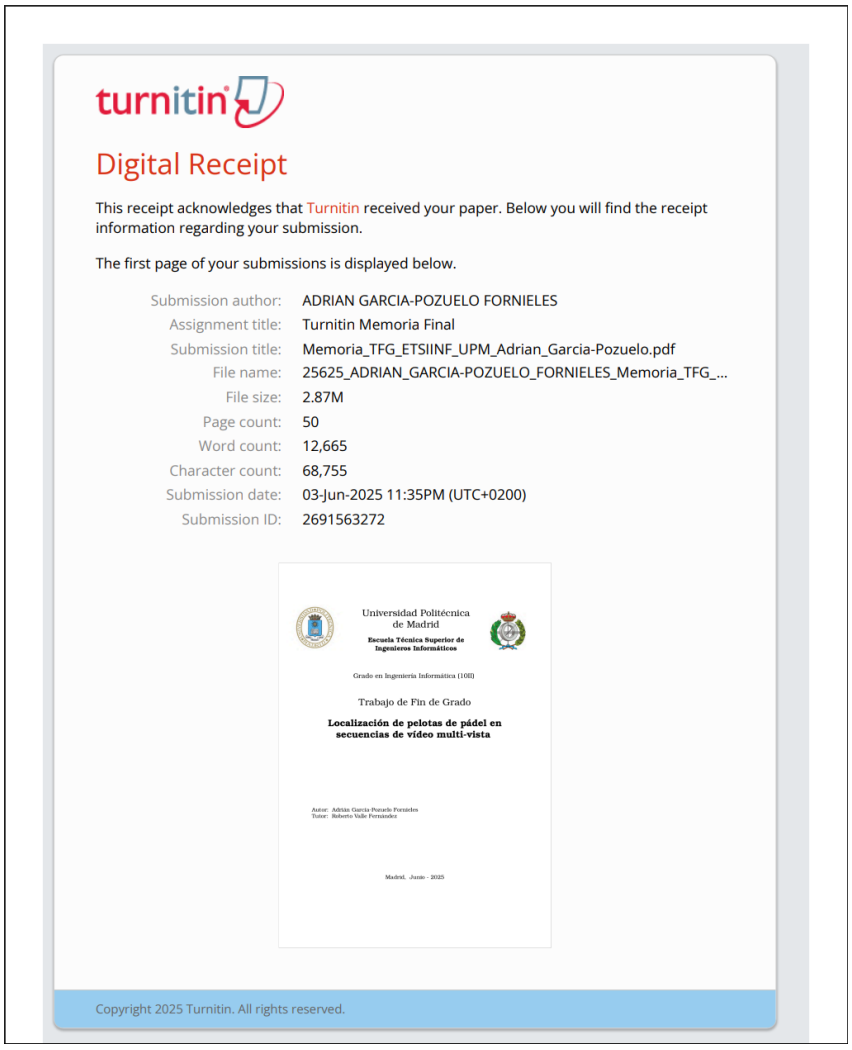
[53] NVIDIA. *cuDNN: Biblioteca de aceleración de redes neuronales profundas*. [Online]. URL: https://developer.nvidia.com/cudnn (visitado 03-05-2025).

[54] Nathalie Smuha. «Ethics guidelines for trustworthy AI». En: *European Commission* (2024).

Capítulo 5

Anexo

Anexo A: Informe de Originalidad



turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	ADRIAN GARCIA-POZUELO FORNIELES
Assignment title:	Turnitin Memoria Final
Submission title:	Memoria_TFG_ETSIINF_UPM_Adrian_Garcia-Pozuelo.pdf
File name:	25625_ADRIAN_GARCIA-POZUELO_FORNIELES_Memoria_TFG_...
File size:	2.87M
Page count:	50
Word count:	12,665
Character count:	68,755
Submission date:	03-Jun-2025 11:35PM (UTC+0200)
Submission ID:	2691563272

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática (10II)

Trabajo de Fin de Grado

Localización de pelotas de pádel en secuencias de vídeo multi-vista

Autor: Adrián García-Pozuelo Fornieles
Tutor: Roberto Valle Fernández

Madrid, Junio - 2025

Copyright 2025 Turnitin. All rights reserved.

ADRIAN GARCIA-POZUELO FORNIELES

Memoria_TFG_ETSIINF_UPM_Adrian_Garcia-Pozuelo.pdf

Turnitin Memoria Final
TFG ETSIINF (Moodle PP)
Universidad Politecnica de Madrid

Document Details

Submission ID
trn:oid:::1:3267983670

Submission Date
Jun 3, 2025, 11:35 PM GMT+2

Download Date
Jun 3, 2025, 11:38 PM GMT+2

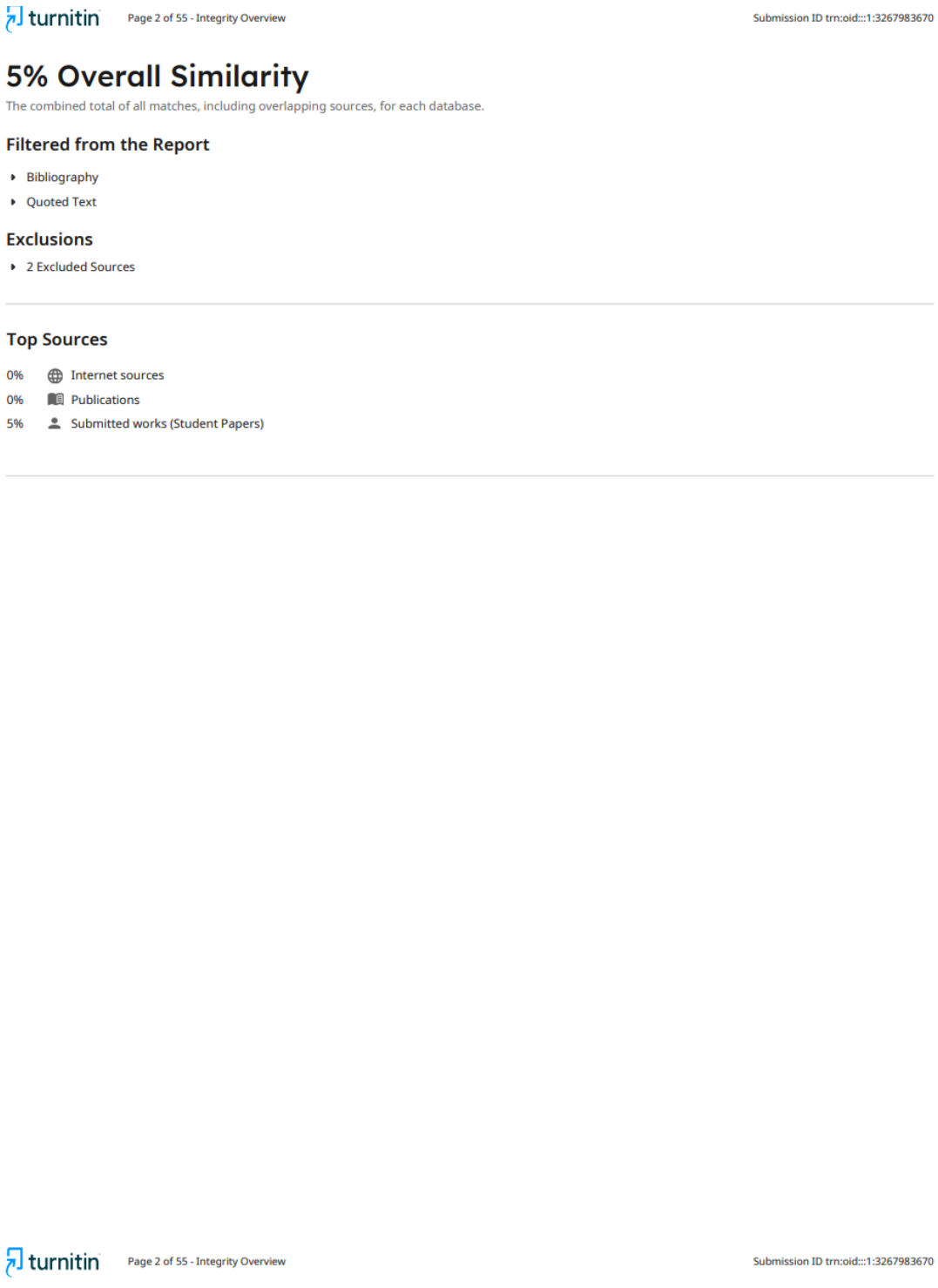
File Name
25625_ADRIAN_GARCIA-POZUELO_FORNIELES_Memoria_TFG_ETSIINF_UPM_Adrian_Garcia-Pozuel....pdf

File Size
2.9 MB

50 Pages
12,665 Words
68,755 Characters

turnitin Page 1 of 55 - Cover Page Submission ID trn:oid:::1:3267983670

Anexo


turnitin
Page 2 of 55 - Integrity Overview
Submission ID trn:oid:::1:3267983670
5% Overall Similarity
The combined total of all matches, including overlapping sources, for each database.
Filtered from the Report
Bibliography
Quoted Text
Exclusions
2 Excluded Sources
Top Sources
0% Internet sources
0% Publications
5% Submitted works (Student Papers)
turnitin
Page 2 of 55 - Integrity Overview
Submission ID trn:oid:::1:3267983670

Top Sources

0% Internet sources
0% Publications
5% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	Corporación Universitaria Iberoamericana <1%
2	Student papers	Universidad Carlos III de Madrid - EUR <1%
3	Student papers	Universidad de León <1%
4	Student papers	Amrita Vishwa Vidyapeetham <1%
5	Student papers	Universitat Politècnica de València <1%
6	Student papers	Universidad Rey Juan Carlos <1%
7	Student papers	Monash University <1%
8	Student papers	Universidad Carlos III de Madrid <1%
9	Student papers	University of Hong Kong <1%
10	Student papers	The University of Manchester <1%
11	Student papers	Universidad Tecnica De Ambato- Direccion de Investigacion y Desarrollo , DIDE <1%

turnitin Page 3 of 55 - Integrity Overview Submission ID trn:oid:::1:3267983670

Anexo

12	Student papers	
Universidad Católica Boliviana "San Pablo"		<1%
13	Student papers	
Universidad Politécnica de Madrid		<1%
14	Student papers	
Arab Academy for Science, Technology & Maritime Transport CAIRO		<1%
15	Student papers	
Universidad Loyola Andalucia		<1%
16	Student papers	
University of Sheffield		<1%
17	Student papers	
University of Technology, Sydney		<1%
18	Student papers	
University of the Andes		<1%
19	Student papers	
ipn		<1%
20	Student papers	
Manipal University		<1%
21	Student papers	
84752		<1%
22	Student papers	
Universidad Internacional de la Rioja		<1%
23	Student papers	
Universidad de Alcalá		<1%
24	Student papers	
University of Western Australia		<1%
25	Student papers	
Swinburne University of Technology		<1%

26 Student papers
Universidad Francisco de Vitoria <1%

27 Student papers
Systems Link <1%

28 Student papers
University of Manouba <1%

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 11:13:10 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)