Universidad de Valladolid

Escuela de Ingeniería Informática de Valladolid

Trabajo Fin de Grado

Grado en Ingeniería Informática

Mención Computación

Visión Artificial aplicada a la detección y reconocimiento de dorsales en competiciones deportivas

---

Autor:
Diego de la Puente Alonso

Tutor:
Teodoro Calonge Cano

Agradecimientos

A mi tutor, Teodoro, por su valiosa orientación y la resolución de todas las dudas que fueron surgiendo durante el proceso. Su apoyo ha sido fundamental para llevar este proyecto a buen término.

A los profesores del Grado, quienes con sus enseñanzas y dedicación han proporcionado los conocimientos y las herramientas necesarias para afrontar este trabajo.

A mis compañeros en la Universidad, quienes con su compañía, colaboración e intercambio de ideas han contribuido a hacer este camino más enriquecedor y llevadero.

A mi familia, cuyo respaldo incondicional ha sido esencial en cada etapa de esta carrera. En especial, a mi hermano Javi, por facilitarme el uso de su GPU, lo cual resultó clave para el desarrollo y entrenamiento de los modelos utilizados en este trabajo.

A todos ellos, muchas gracias.

Resumen

Los dorsales en competiciones deportivas están compuestos por un rectángulo de distintos tamaños y colores, con un número identificativo y con formato variado que se utiliza para identificar a los participantes en eventos deportivos, especialmente en carreras. Todo esto con el objetivo de saber que el deportista está inscrito en la competición y facilitar la creación de clasificaciones de los participantes.

En este trabajo, mediante Visión por Computadora, se buscará detectar y reconocer dorsales. Esto se hará explorando modelos de detección de objetos como YOLO y distintos OCR para reconocer la información alfanumérica del dorsal.

El objetivo principal de este trabajo es desarrollar una aplicación capaz de clasificar imágenes de competiciones deportivas mediante los números de los dorsales, facilitando así la localización de fotografías dentro de grandes álbumes. Así mismo, al disponer de la fecha y hora de la foto, se puede usar para confeccionar clasificaciones parciales.

Abstract

Bib numbers are rectangular boxes characterized by different size, color and formats, where the participant ID has been recorded. Nowadays, this identification is used in all races worldwide, since it is a visual registration proof, which is the key to obtain partial and final rankings.

This project employs Computer Vision techniques to detect and recognize bib numbers in images. This will involve exploring object detection models like YOLO and various OCR applications to extract the alphanumeric information from the bib.

The main goal of this project is to develop an image classification system according to bib numbers, whose inputs are extracted from a large set of photographs taken during the race. In adition, using the metadata of each image, a timetable could be obtained at every control point.

Índice general

Índice de figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Concepto de dorsal

Para comprender adecuadamente este trabajo, es fundamental entender qué se entiende por dorsal (o bib, en inglés) en el contexto de una competición deportiva. Se trata de un rectángulo que presenta variaciones en tamaño, color y formato. Contiene un número identificativo asignado a cada participante, lo que permite su reconocimiento.

Figura 1.1: Distintos dorsales con distintos formatos

En la mayoría de las competiciones deportivas se requiere el uso de un dorsal que identifique a los participantes. En este proyecto, nos enfocamos en aquellos utilizados en las carreras a pie, ya que este tipo de eventos son los más comunes y representan la mayoría de las imágenes disponibles.

Estos dorsales los debe llevar cada participante en la competición en una zona visible, para acreditar que está inscrito y acepta el reglamento de la carrera. Normalmente se lleva en el abdomen, sujetos a la camiseta con imperdibles.

Figura 1.2: Ejemplo de una foto de carrera con el dorsal en el abdomen

En la actualidad, es habitual el uso de chips en los dorsales para registrar tiempos parciales o finales, con el fin de generar las clasificaciones de la competición. La implementación de un sistema como el propuesto en este TFG, permitiría prescindir de este mecanismo, sustituyéndolo por un control basado en fotografías, en las que se pueda reconocer el dorsal del participante.

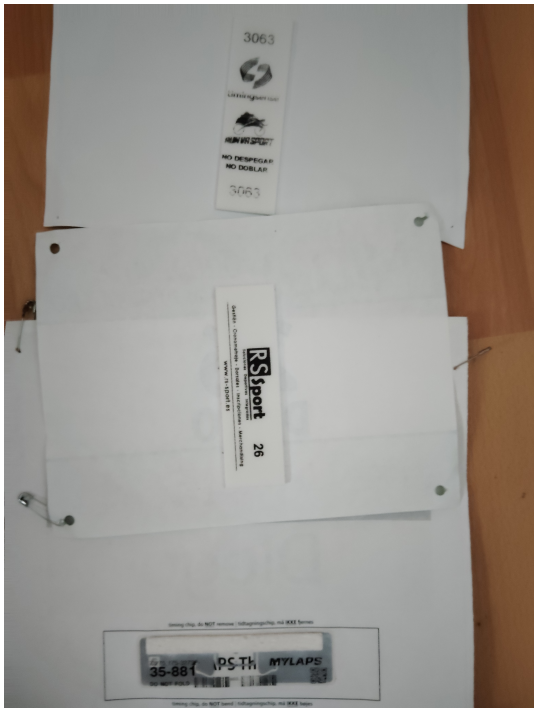

Figura 1.3: Chips de distintos dorsales

1.2. Motivación y contexto

La finalidad de este TFG es desarrollar un sistema capaz de reconocer los números de los dorsales para clasificar automáticamente las imágenes contenidas en los álbumes de fotografías. Existen muchos correspondientes a distintas carreras [1, 2], pero cuando estos contienen una gran cantidad de imágenes, resulta difícil localizar a una persona concreta. Por ello, la creación de una aplicación que clasifique automáticamente las fotos según los dorsales, podría resolver este inconveniente.

Aunque este tipo de servicio ya es ofrecido por algunas empresas especializadas [3, 4], en muchas ocasiones las carreras no disponen de este sistema. A partir de los conocimientos adquiridos durante los estudios universitarios, surge la idea de desarrollar un sistema propio que permita llevar a cabo esta tarea de forma automática.

Tras una revisión bibliográfica, se han detectado pocos proyectos de código abierto que cumplan el objetivo mencionado. Únicamente se han identificado dos iniciativas similares en la plataforma GitHub; ambas con resultados no muy buenos [5, 6]. Desde el punto de vista de la Visión por Computador, este problema se plantea como una tarea de detección de objetos, a la que se añade la necesidad de interpretar su contenido alfanumérico. Esto guarda cierta similitud con los sistemas de lectura automática de matrículas de vehículos aunque, en este caso, los dorsales presentan un formato menos estandarizado.

1.3. Objetivos

Este proyecto se centra en la detección de los dorsales en imágenes mediante distintos modelos y la identificación del número del dorsal con un OCR. Por ello, una fase importante del trabajo será un estudio profundo del problema y las soluciones que los modelos de Inteligencia Artificial puedan ofrecer. Teniendo esto en cuenta, se establecen los siguientes hitos:

- Análisis del problema y sus partes, mediante un revisión bibliográfica.
- Búsqueda y refinamiento de un conjunto de datos con imágenes adecuadas.
- Desarrollo de un sistema capaz de detectar y reconocer dorsales.
- Evaluación de las diferentes propuestas.
- Creación de una aplicación web que permita buscar imágenes según un número de dorsal.

1.4. Estructura de la memoria

Este documento se divide de la siguiente forma:

- **Capítulo 1 Introducción.** Se presenta el problema, definiendo qué es un dorsal y el contexto en el que se usa.
- **Capítulo 2 Gestión del proyecto.** En este parte se detalla la planificación, incluyendo la metodología empleada, la planificación inicial, la gestión de riesgos, el seguimiento del desarrollo y una breve evaluación de la planificación.
- **Capitulo 3 Marco teórico.** Se definen los conceptos intrínsecos relevantes de la tecnología usada para la detección y reconocimiento de los dorsales.
- **Capítulo 4 Conjuntos de datos.** Se describen los bancos de imágenes usados, así como su origen.
- **Capítulo 5 Desarrollo del modelo.** Se detalla la creación y funcionamiento del modelo.
- **Capitulo 6 Resultados.** Se exponen las diferentes tablas que resumen las prestaciones del sistema para escoger el modelo más adecuado.
- **Capítulo 7 Aplicación web.** Se describe el análisis, diseño y desarrollo de la aplicación web del TFG.
- **Capítulo 8 Conclusiones.** Se realiza una reflexión de lo aprendido y los resultados alcanzados. Se propondrán las lineas futuras que surgen del desarrollo del mismo.

Finalmente, los apéndices recogen información complementaria como parte del código usado en el desarrollo del proyecto y un manual de la aplicación web para hacer que su uso sea más sencillo.

Capítulo 2

Gestión del proyecto

Debido a que este proyecto tiene una duración elevada, es necesaria una planificación adecuada. Por esto mismo, en el presente capítulo se expondrá la metodología utilizada en la realización del proyecto, la planificación, los riesgos, los costes asociados al mismo, su seguimiento y una breve evaluación de la planificiación.

2.1. Metodologías del trabajo

En este trabajo existe una componente de investigación que lo diferencia de un proyecto de ingeniería de software convencional. Por este motivo, se opta por emplear la metodología Scrum, adaptada a un equipo formado por el estudiante con el apoyo del tutor [7]. No obstante, dado que el proyecto se enmarca en el ámbito de la Minería de Datos, también resulta imprescindible considerar otras metodologías ampliamente utilizadas, como KDD y CRISP-DM, las cuales se presentarán a continuación.

- **KDD** [8]. El proceso de Knowledge Discovery in Databases (KDD), o descubrimiento de conocimiento en bases de datos, consiste en la recogida y refinamiento de datos para extraer conocimiento útil. Se trata de un enfoque clásico dentro de la Ciencia de Datos que busca identificar información relevante a partir de grandes volúmenes de datos. El proceso de KDD se estructura en varias etapas :

  - **Selección**: identificación y recopilación de los datos relevantes a partir de diversas fuentes.
  - **Preprocesamiento**: limpieza y preparación de los datos para asegurar su calidad y fiabilidad.
  - **Transformación**: conversión de los datos en formatos adecuados para su posterior análisis.
  - **Procesamiento**: aplicación de técnicas de minería de datos para encontrar patrones o modelos para los datos.
  - **Interpretación y evaluación**: análisis de los resultados obtenidos para extraer conclusiones y generar conocimiento útil.

- **CRISP-DM** [9]. Las siglas CRISP-DM (Cross-Industry Standard Process for Data Mining) hacen referencia al proceso estándar intersectorial para la minería de datos, desarrollado con el objetivo de estandarizar las prácticas de minería de datos en las empresas. Esta metodología se estructura en seis fases principales: comprensión del negocio, comprensión de los datos, preparación de los datos, modelado, evaluación e implementación.

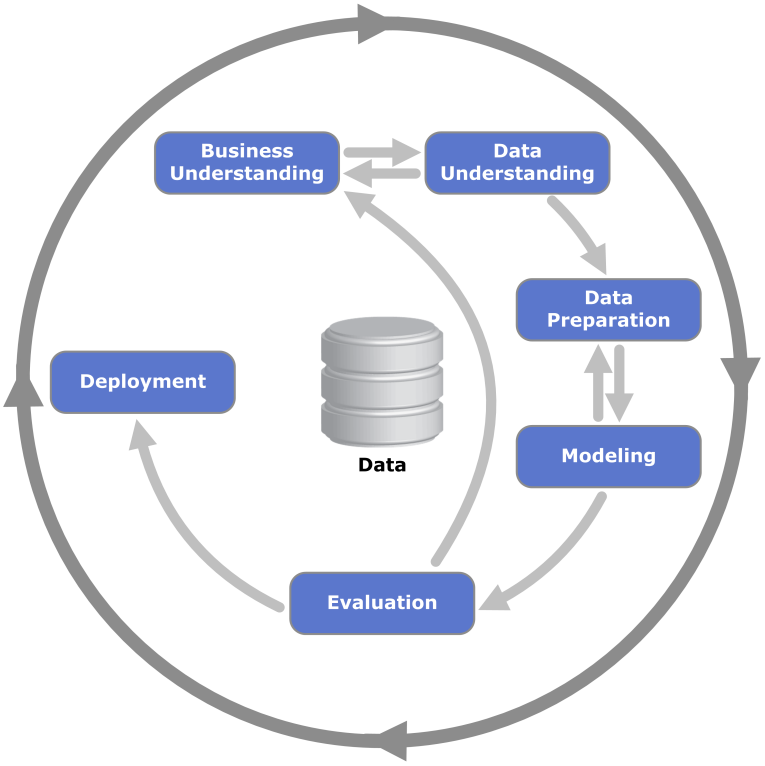

Figura 2.1: Proceso CRISP-DM[10]

Teniendo en cuenta que el objetivo del negocio es el reconocimiento de dorsales, y que la implementación final consistirá en la generación de un informe con los resultados obtenidos por el modelo, se ha decidido utilizar CRISP-DM como metodología base, dado que es la más común y flexible en el ámbito de la minería de datos. No obstante, CRISP-DM no define cómo gestionar de forma práctica la planificación y el desarrollo iterativo del proyecto. Por ello, se complementará con la metodología ágil Scrum, que permitirá estructurar el trabajo en ciclos de desarrollo cortos (sprints) y realizar una evolución progresiva del proyecto. Además, el uso de Scrum resulta especialmente adecuado dado que el proyecto presenta una fuerte componente de investigación, lo que requiere un enfoque adaptable y capaz de incorporar nuevos conocimientos de manera incremental.

2.1.1. Metodología Scrum

La metodología Scrum es un enfoque ágil que organiza el trabajo en ciclos iterativos, llamados sprints, de aproximadamente dos semanas, durante los cuales se planifican y alcanzan objetivos concretos.

Aunque Scrum puede incluir elementos adicionales, como reuniones diarias del equipo (daily meetings), la asignación de roles específicos como el Scrum Master, o el uso de tableros Kanban para el seguimiento de tareas, en este proyecto se aplicará una versión simplificada, adaptada a su escala y requerimientos. En particular, el desarrollo se estructurará en sprints desde el inicio del proyecto, comenzando con una fase de investigación inicial a cargo del alumno. Cada sprint tendrá una duración de dos semanas y finalizará con una reunión con el tutor, en la que se evaluará el progreso alcanzado y se definirán los objetivos del siguiente sprint.

2.2. Plan inicial

La planificación inicial estará compuesta por distintos sprints en los que buscarán cumplir con los objetivos iniciales del proyecto 1.3.

2.2.1. Sprint planeados

Excepcionalmente, los dos primeros sprints tendrán una duración superior a las dos semanas, debido a que, en estos, el estudiante necesita un amplio periodo de tiempo para investigar sobre posibles modelos, conseguir las imágenes y cerciorarse de que están bien etiquetadas.

Sprints Scrum	**Fechas**
Sprint 1	20/1/2025 - 13/2/2025
Sprint 2	13/2/2025 - 14/3/2025
Sprint 3	14/3/2025 - 28/3/2025
Sprint 4	28/3/2025 - 11/4/2025
Sprint 5	11/4/2025 - 25/4/2025
Sprint 6	25/4/2025 - 9/5/2025
Sprint 7	9/5/2025 - 23/5/2025
Sprint 8	23/5/2025 - 6/6/2025

Tabla 2.1: Tabla duración sprints

2.2.2. Objetivos y tareas de los sprints

Se aclararán los objetivos, tareas y el tiempo estimado en horas dedicadas en cada uno de los sprints.

Sprints 1	**20/1/2025 - 13/2/2025**
Objetivo	Investigación inicial de posibles soluciones.
Tareas	▪ Revisión bibliográfica. ▪ Búsqueda de conjuntos de datos.
Duración estimada	40 horas

Tabla 2.2: Tabla sprint 1

Sprints 2	**13/2/2025 - 14/3/2025**
Objetivo	Primera aproximación a la resolución del problema.
Tareas	▪ Primera versión del reconocedor con YOLOv4 y un OCR. ▪ Descarga de las primeras imágenes. ▪ Reetiquetado de imágenes.
Duración estimada	40 horas

Tabla 2.3: Tabla sprint 2

Sprints 3	**14/3/2025 - 28/3/2025**
Objetivo	Mejorar la solución inicial.
Tareas	▪ Cambiar YOLOv4 por YOLOv12 y YOLOv11. ▪ Añadir imágenes.
Duración estimada	30 horas

Tabla 2.4: Tabla sprint 3

Sprints 4	**28/3/2025 - 11/4/2025**
Objetivo	Mejorar los resultados del OCR.
Tareas	▪ Mejorar OCR. ▪ Revisar anotaciones y añadir imágenes.
Duración estimada	30 horas

Tabla 2.5: Tabla sprint 4

Sprints 5	**11/4/2025 - 25/4/2025**
Objetivo	Analizar datos y categorizar las imágenes.
Tareas	▪ Revisar características de las imágenes. ▪ Estratificar las imágenes para que se respete su distribución en el entrenamiento y prueba. ▪ Separar el conjunto de datos en test, entrenamiento y validación. ▪ Añadir últimas imágenes que tengan características que no se encuentren en el conjunto de datos.
Duración estimada	40 horas

Tabla 2.6: Tabla sprint 5

Sprints 6	**25/4/2025 - 9/5/2025**
Objetivo	Entrenamiento y selección del sistema de reconocimiento de dorsales y primera aproximación a la aplicación web.
Tareas	▪ Entrenar modelos con los datos particionados y 3-kfold. ▪ Analizar los resultados de los distintos modelos. ▪ Explorar soluciones para la creación de la aplicación web.
Duración estimada	40 horas

Tabla 2.7: Tabla sprint 6

Sprints 7	**9/5/2025 - 23/5/2025**
Objetivo	Creación de la aplicación web con el sistema de reconocimiento de dorsales seleccionado
Tareas	▪ Anáilisis del modelo más adecuado. ▪ Crear la aplicación web. ▪ Primera versión completa de la memoria.
Duración estimada	50 horas

Tabla 2.8: Tabla sprint 7

Sprints 8	**23/5/2025-6/6/2025**
Objetivo	Finalizar el proyecto revisando la memoría y la aplicación web
Tareas	▪ Despliegue la aplicación web. ▪ Revisión exhaustiva de la memoria.
Duración estimada	30 horas

Tabla 2.9: Tabla sprint 8

2.3. Planificación de riesgos

Antes de comenzar un proyecto, es fundamental elaborar un plan de riesgos que permita identificarlos y considerarlos durante su desarrollo, con el objetivo de establecer medidas preventivas y de mitigación.

Los riesgos se clasificarán en función de su probabilidad de ocurrencia y su impacto, utilizando las categorías baja, media y alta. Para cada riesgo identificado se definirá como mínimo un plan de mitigación y otro de contingencia. A continuación, veremos los riesgos más relevantes que se pueden dar en este trabajo.

Riesgo R01	**Mala estimación del tiempo**
Descripción	La estimación del tiempo de las tareas a realizar puede ser inadecuada por cualquier tipo de problema en su desarrollo o porque el tiempo planificado no fue el correcto.
Probabilidad	Media
Impacto	Medio
Mitigación	▪ Empezar a realizar las partes más importantes del proceso al principio y dejar detalles para el final. ▪ Conseguir una estimación del tiempo lo más realista posible.
Contingencia	▪ Realizar la planificación del proyecto teniendo en cuenta el tiempo disponible.

Tabla 2.10: Riesgo 01

Riesgo R02	**Falta de imágenes para el modelo**
Descripción	Puede darse el caso en que no se encuentren suficientes imágenes para poder entrenar el modelo.
Probabilidad	Baja
Impacto	Medio
Mitigación	▪ Buscar imágenes desde el principio del proyecto. ▪ Buscarlas en distintos lugares que contengan imágenes públicas.
Contingencia	▪ Sacar las imágenes en carreras locales.

Tabla 2.11: Riesgo 02

Riesgo R03	**Malos resultados del modelo de reconocimiento**
Descripción	Al ser un sistema de reconocimiento en el que no existe mucha literatura, puede darse el caso de no conseguir un modelo con un resultado aceptable.
Probabilidad	Media
Impacto	Alto
Mitigación	▪ Buscar distintos modelos para solucionar el problema y quedarse con el que mejores resultados dé. ▪ Buscar documentación de problemas similares, como el reconocimiento de matriculas, para encontrar modelos que puedan funcionar bien para esta problema.
Contingencia	▪ Empezar lo antes posible a buscar modelos y hacerlo de distintas fuentes.

Tabla 2.12: Riesgo 03

Riesgo R04	**Falta de conocimientos en la materia**
Descripción	Al principio del proyecto el estudiante carece de conocimientos para la resolución del problema.
Probabilidad	Media
Impacto	Alto
Mitigación	▪ Leer documentación sobre problemas similares y de los posibles modelos a usar.
Contingencia	▪ Empezar lo antes posible a buscar como funcionan los modelos de detección de objetos y reconocimiento de caracteres.

Tabla 2.13: Riesgo 04

Riesgo R05	**Enfermedades**
Descripción	El estudiante tiene síntomas de una enfermedad lo que le impide dedicarse tanto tiempo al proyecto cómo estaba planificado.
Probabilidad	Media
Impacto	Medio
Mitigación	▪ Seguir las medidas adecuadas de higiene y cuidado personal.
Contingencia	▪ Dedicar más horas al proyecto para terminar las partes incompletas debido a la enfermedad.

Tabla 2.14: Riesgo 05

Riesgo R06	**Recursos hardware**
Descripción	Se puede dar el caso de que algún componente a usar para realizar el proyecto falle.
Probabilidad	Baja
Impacto	Alto
Mitigación	▪ Realizar copias de seguridad de los ficheros clave. ▪ Tener los ficheros clave en varios sitios.
Contingencia	▪ Si fallase la GPU o el mecanismo para acceder a ella pedir acceso a una de la universidad.

Tabla 2.15: Riesgo 06

Riesgo R07	**Evaluación del sistema de reconocimiento de dorsales**
Descripción	Dificultad para elegir el modelos más adecuado, lo que puede llevar a resultados poco relevantes o mal justificados.
Probabilidad	Media
Impacto	Medio
Mitigación	▪ Estudiar y analizar las métricas más relevante que permiten evaluar los distintos sistemas. ▪ Consultar cómo se evalúan distintos modelos para problemas similares.
Contingencia	▪ Justificar de manera fundamentada las incertidumbres presentes a partir del análisis de las métricas obtenidas, y seleccionar el modelo que muestre un mejor rendimiento basadose en dichos indicadores.

Tabla 2.16: Riesgo 07

2.4. Planificación presupuesto

En esta sección, detallaremos los gastos asociados a la realización de este proyecto. A efectos de amortización, se considerará que los distintos elementos utilizados han estado en uso durante un período de cinco meses, desde finales de enero hasta el mes de junio.

2.4.1. Coste hardware

Para realizar este proyecto se utilizan los siguientes dispositivos informáticos.

- GPU NVIDIA GeForce RTX 3090, lanzada en 2020, fue adquirida por aproximadamente 1.100€. Con un uso moderado y sin un desgaste excesivo, se estima que su vida útil es de unos 10 años. Por lo tanto, el coste aproximado de utilizarla de forma no intensiva durante 5 meses, incluyendo un pequeño gasto en electricidad, se calcula en unos 60€.
- Máquina virtual que cuenta con 100 GB de almacenamiento, un procesador Intel® Core™ i5-10600K y 8 GB de RAM. Con estas especificaciones, el coste estimado de la máquina es de aproximadamente 500€. Si se utiliza de manera no intensiva durante un periodo de 5 meses, el coste aproximado sería de unos 30€.
- Portátil personal, en concreto un asus Aspire 3 que cuesta aproximadamente 400€ y cuyo coste para un de 5 meses se estima en 25€.
- Pantalla Asus de 27 pulgadas con coste aproximado de 150€ y cuyo coste para un uso de 5 meses se estima en 10€.

Por lo que el coste total estimado del hardware es de 125€.

2.4.2. Coste software

El coste asociado al software utilizado durante el desarrollo del TFG es nulo, ya que se han empleado exclusivamente herramientas y tecnologías gratuitas o de código abierto. Todas las tecnologías empleadas serán citadas y detalladas en los apartados correspondientes del documento conforme se vayan utilizando.

A continuación, mencionaremos las herramientas usadas en la elaboración de la memoria:

- draw.io [11] : para la creación de diagramas y esquemas.
- Overleaf [12] : plataforma online para la redacción de documentos en LaTeX, usada para la creacción de la memoria.
- Texmaker [13] : editor de LaTeX libre, multiplataforma e integrado, diseñado para simplificar la creación de documentos científicos y técnicos. Se emplea en este proyecto debido a que, en las etapas finales, se supera el límite de tiempo de compilación gratuito permitido por Overleaf.
- Astah [14] : herramienta usada para la creación de diagramas UML para la aplicación web.
- Gantt project [15] : utilizado para la creacción de diagramas de Gantt.

Dado que ninguna de estas herramientas implica un coste económico, no se ha incluido ningún gasto de software en el presupuesto del proyecto.

2.4.3. Coste infrastructura

Para realizar el proyecto es necesario una habitación/oficina en Valladolid durante 5 meses con Wi-Fi lo aproximamos por 625€ [16].

2.4.4. Coste recursos humanos

Para realizar el TFG se necesita el trabajo de un junior durante 300 horas. Estimando el coste por hora de un recién graduado en informática es de 19.000€ [17] al año y por tanto el coste por hora es 9,1€. Por ello, el coste por recursos humanos se estima en 2.730€.

2.4.5. Coste total

Sumando los costes anteriores, tenemos un coste total aproximado para la ejecución de este trabajo dentro de un ámbito empresarial sería de 3.480€.

2.5. Seguimiento del plan

Comenzaremos presentando los diagramas de Gantt correspondientes a los distintos sprints, organizados por etapas. Las tareas estarán diferenciadas por colores, según las fases definidas en la metodología CRISP-DM.

Primera etapa del proyecto

Esta ha sido la etapa que ha requerido más tiempo de desarrollo. En ella se llevaron a cabo las fases de comprensión del negocio (representada en rojo claro), comprensión y preparación de los datos (en amarillo), y el desarrollo inicial de los primeros modelos (en verde).

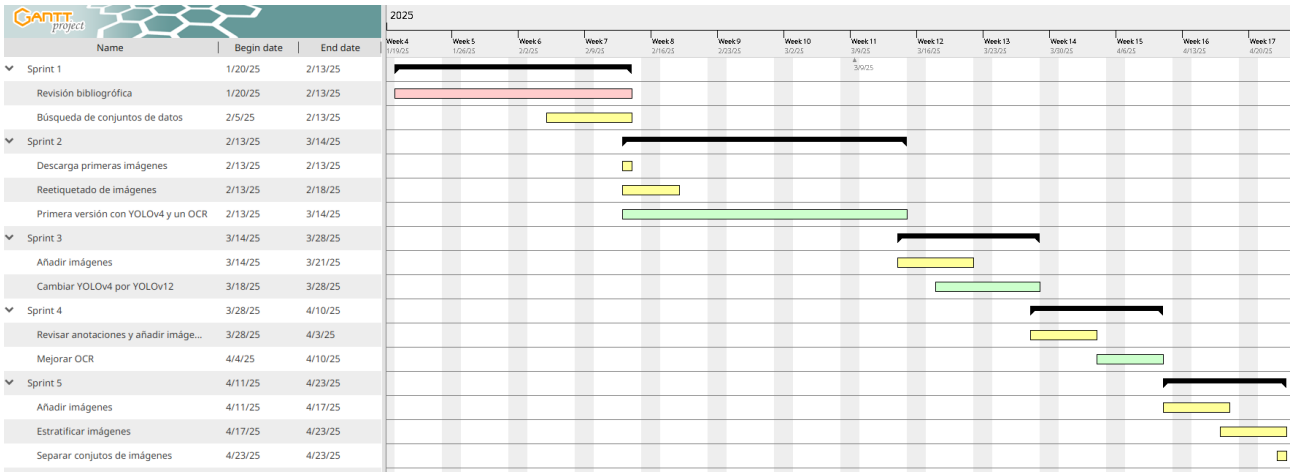

Figura 2.2: Diagrama de Gantt primera etapa del proyecto

Esta etapa se lleva a cabo entre finales de enero y mediados de abril, con una duración aproximada de 180 horas. En la siguiente fase se finalizará el proyecto, dedicando las aproximadamente 120 horas restantes previstas para el desarrollo del TFG.

Segunda etapa del proyecto

Esta etapa se desarrolla una vez obtenidos los datos y las primeras versiones del modelo de reconocimiento de dorsales. En ella se entrenan las versiones definitivas de los distintos modelos utilizando todos los datos, se realiza su evaluación (representada en azul) y se lleva a cabo la creación y el despliegue de la aplicación web (en morado).

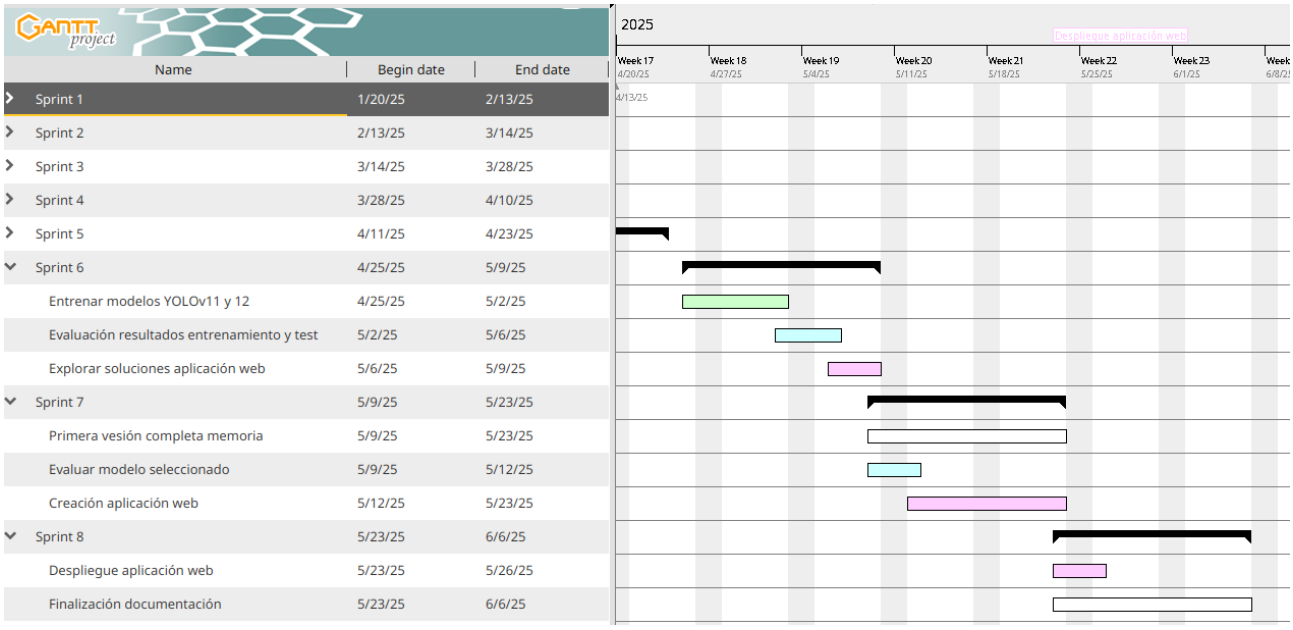

Figura 2.3: Diagrama de Gantt segunda etapa del proyecto

Tmabién, se incluyen tareas en color blanco que indican el desarrollo de la documentación. Si bien esta se elabora a lo largo de todo el proyecto, durante esta fase final su producción se intensifica considerablemente.

2.6. Evaluación de la planificación

Al realizar la planificación en cada reunión con el tutor, se ha logrado mantener el cumplimiento de los objetivos establecidos. Cabe destacar la existencia de varias iteraciones, cuyo propósito ha sido mejorar los resultados de los modelos y añadir nueva información (sprints 3, 4 y 5) que no podían preverse hasta disponer de los primeros resultados.

Asimismo, se han definido objetivos concretos y plazos específicos para cada sprint, junto con una estimación de tiempo. Estos objetivos se han ido cumpliendo y la estimación del tiempo se corresponde de una forma aproximada con el tiempo real utilizado. Esta metodología ha permitido distribuir el trabajo de manera equilibrada y evitar parones en el desarrollo del proyecto, ya que cada dos semanas debían alcanzarse ciertos objetivos.

Capítulo 3

Marco teórico

Se exponen los fundamentos teóricos necesarios para comprender el proyecto, incluyendo el funcionamiento de los modelos YOLO que se utilizarán y el reconocimiento de caracteres mediante un OCR. Para ello, empezaremos explicando los disntos tipos de redes neuronales que usan.

3.1. Redes neuronales

Son modelos de aprendizaje automático diseñados para reconocer patrones y resolver tareas complejas. En esta sección, se explicarán sus principios básicos como base teórica para distintos modelos.

3.1.1. Neurona artificial

Una red neuronal está formada por un conjunto de unidades interconectadas, conocidas como neuronas artificiales, que se inspiran en el funcionamiento de las biológicas. Se comunican entre sí mediante conexiones y procesan información inspirandose en la fisiología de nuestro sistema nervioso. Para comprender mejor su estructura y funcionamiento, compararemos las artificiales con las biológicas.

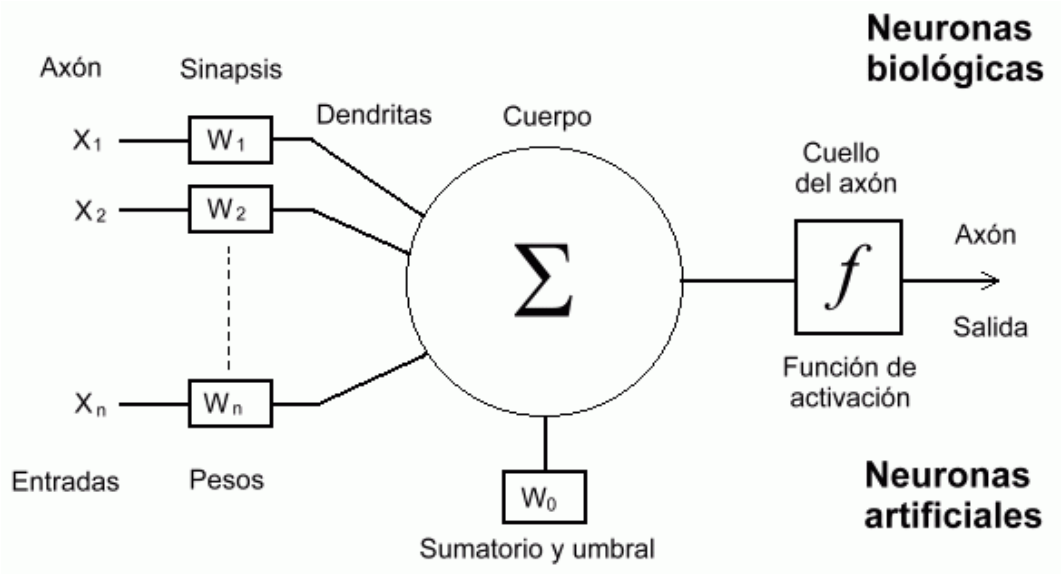

Figura 3.1: Comparación neurona artificial y neurona biológica [18]

En la figura 3.1 aparecen los siguientes términos:

1. **Axón/entradas**: en este punto las entradas son enviadas al sistema. En el modelo artifical, estas se representan como un vector $X1, ..., Xn$.
2. **Sinapsis/pesos**: ponderan la información que reciben para combinarse linealmente. En la neurona artificial, la intensidad de las entradas se debe a unos pesos $W1, ..Wn$, que modulan la influencia de cada entrada sobre el resultado final.
3. **Cuerpo/sumatorio y umbral**: en este bloque se calcula el resultado del procesamiento de las entradas.
4. **Cuello del axón/función de activación**: este es el procesamiento en el cual se transmiten los resultados anteriores, previo un filtro por una función de activación en el modelo artificial. Esta última operación es imprescindible para poder llevar a cabo un correcto procesamiento numérico posterior.
5. **Axón/salida**: aquí se materialaza las interconexiones a partir de las cuales se transmite las salidas de una neuronas a otras.

3.1.2. Funciones de activación

A partir de la ya comentado anteriormente 4, esta función se utiliza para se puedan obtener superficies de separación no lineales entre las clases. Asimismo, sirve para limitar la salida de las neuronas acotandalos numericamente con un función monótona creciente, lo que dotara una estabilidad a los algoritmos de aprendizaje. Para que los que se basen en gradientes funcionen de manera eficiente, la función de activación debe ser derivable. La elección de esta función influye directamente en el rendimiento del entrenamiento de las redes, pudiendo dar lugar a resultados significativamente distintos según la opción seleccionada. A continuación, se describen las funciones de activación que son más relevantes en este trabajo.

Sigmoide

Es muy utilizada para problemas de clasificación binaria. Además, está relacionada con la función de activación Swish que se explicará a continuación.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

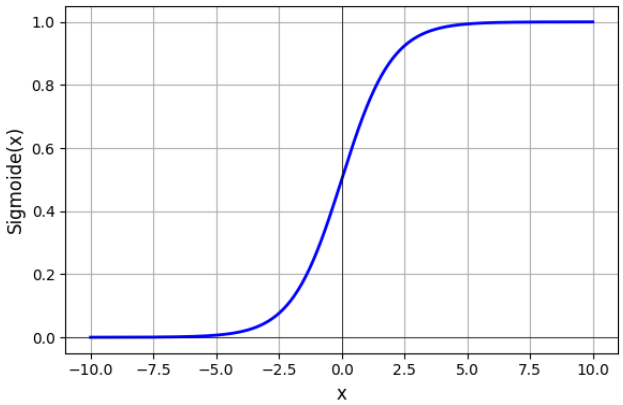

Figura 3.2: Sigmoide elaboración propia

SiLU(Sigmoid Linear Unit) o Swish

Propuesta por los científicos de Google en una publicación de 2017 [19]. Ha ganado popularidad debido a su efectividad en su desempeño en los modelos frente a la ReLU o sigmoide [20]. Esta es la función principal de los modelos de YOLO que se usarán en este proyecto.

$$SiLU(x) = x * sigmode(x) = x\left(\frac{1}{1+e^{-x}}\right)$$

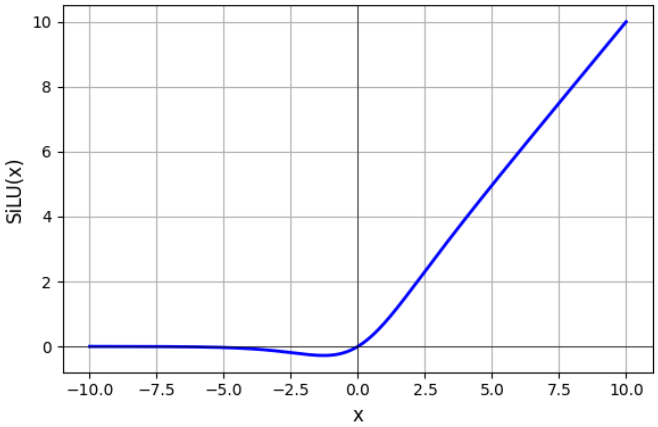

Figura 3.3: SiLU elaboración propia

Softmax

Se usa solo en la capa de salida cuando se quiere aplicar un enfoque probabilistico a la clasificaicón. En efecto, tras esta operación cada componente del vector de salida se podría interpretar como la probabilidad de pertencia a una clase especifica.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

3.1.3. Perceptrón multicapa

Las neuronas artificiales se pueden juntar de muchas maneras, entre ellas una de las más utilizadas es mediante capas. En esta estructura se basa el perceptrón multicapa o MLP por sus siglas en inglés. Cuyo esquema se observa en la siguiente figura:

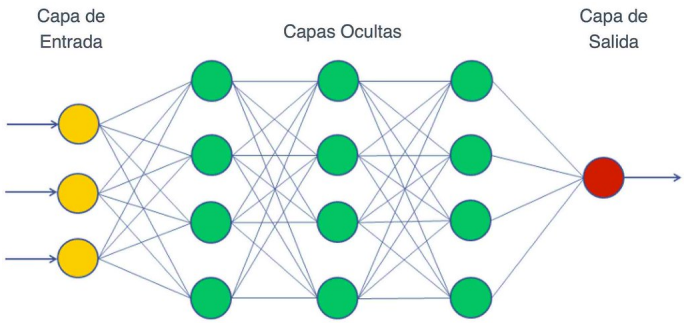

Figura 3.4: Perceptrón multicapa [21]

Hay que destacar, que el número de neuronas en la capa de salida puede ser distinto de 1.

Funcionamiento

La primera capa de neuronas recibe los datos de entrada. A continuación, un conjunto de capas ocultas procesa la información, identificando patrones o características relevantes para la tarea de clasificación. Finalmente, la capa de salida utiliza las características extraídas por la última capa oculta para realizar la clasificación de los datos de entrada.

Aprendizaje

El método más ampliamente utilizado para este proposito es la retroalimentación, o backpropagation en inglés, de la función de perdida, que mide la diferencia entre la salida del modelo y el valor real. En particular, lo que se calcula, es el gradiente de dicha función respecto a los pesos de todas las neuronas empleadas. No obstante, este proceso no siempre garantiza un entrenamiento exitoso, ya que pueden surgir diversos problemas, los cuales se abordarán en una sección posterior 3.4.

Usos

En muchos casos, dentro del ámbito de la Inteligencia Artificial, se utilizan otros tipos de redes especializadas en la extracción de características a partir de los datos. Una vez obtenidas estas representaciones, se emplea una red MLP para realizar la clasificación. En el caso de YOLO se usan estructuras similares a los MLP (redes convolucionales con un kernel de 1) para la predicción de la clase del bounding box y para los bloques de atención, como se explicará más tarde en este capítulo. En un OCR, las redes MLP se utilizan habitualmente para clasificar cada carácter detectado.

3.2. Red neuronal convolucional

En inglés CNN (convolutional neural network) es una pieza clave para los modelos YOLO y OCR. Está diseñada para el procesamiento de imágenes sin perder información espacial. Está compuesta por capas de convolución, normalmente acompañadas por capas de pooling y capas totalmente conectadas (también llamadas densas).

3.2.1. Operación de convolución

Se toman dos funciones una de las cuales se desliza una sobre la otra, multiplicando valores correspondientes en cada punto de superposición y sumando los productos obtenidos en la multiplicación [22]. En el contexto de redes neuronales se usa la versión discreta de esta operación.

Para esta operación, se utiliza una imagen de entrada con tres dimensiones que son la altura, la longitud y el número de canales o profundidad, representadas por sus siglas en inglés como (w, h, d). Sobre esta se aplica la operación de convolución con las d matrices de pesos (con dimensión (m, n)) denominadas matrices de convolución o kernel.

Siendo K cada uno de los kernel, I cada una de las matrices (w, h) de la imagen y R el resultado de la operación, definimos la convolución de la siguiente forma:

$$R(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.1}$$

Si hubiese más de un canal se aplicaría la operación anterior a cada uno por separados y luego se sumarían los resultados término a término, para obtener una sola matriz bidimensional.

Normalmente, las dimensiones (m, n) son muy inferiores a las dimensiones (w, h) con lo que se consigue un resultado con menos dimensiones tras aplicar una operación de convolución. A este resultado se le denomina mapa de características o, en inglés, feature map.

Esta operación es más fácil de entender con un ejemplo, así que en las siguientes figuras se muestra cómo se lleva a cabo con un kernel (3x3) y una imagen de un solo canal y de dimensiones (6x6).

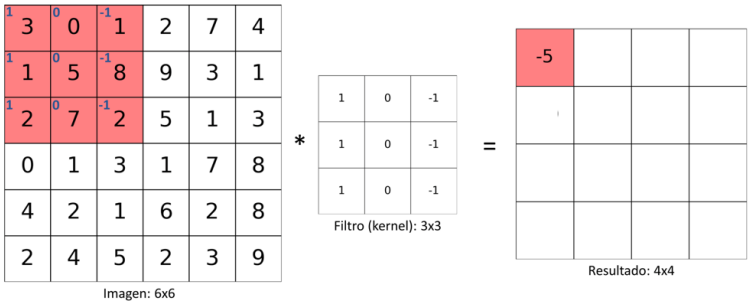

Figura 3.5: Resultado de la convolución tras la primera iteración [23]

En la figura 3.5 se ve cómo se aplica la función de convolución multiplicando término a término el primer trozo de la imagen o patch por el kernel y se suma el resultado. Tras esto, se realiza la misma operación desplazando el patch una unidad a la izquierda. Esto sucede porque el stride, que se explicará más adelante, es uno.

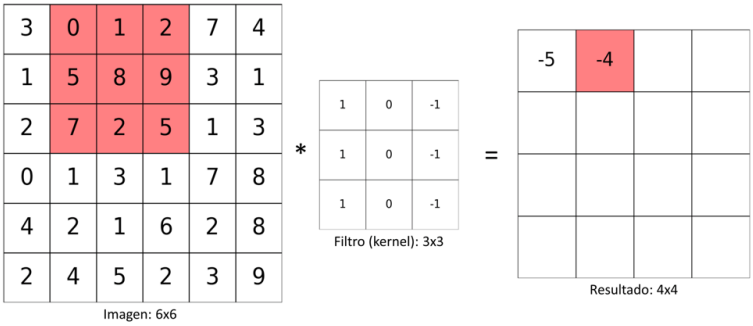

Figura 3.6: Resultado de la convolución tras la segunda iteración [23]

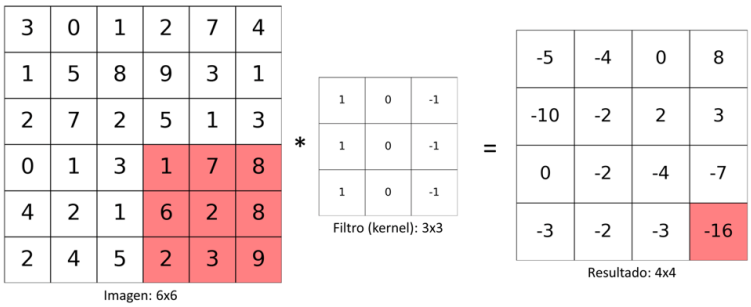

Figura 3.7: Resultado de la convolución tras la última iteración [23]

En la figura 3.7 se ilustra gráficamente cómo se aplica la operación de convolución en cada patch de la imagen. Se observa que, al llegar al borde derecho, el patch vuelve al inicio horizontal y desciende una fila en la siguiente iteración. Este proceso corresponde al descrito en la ecuación 3.1, pero representado de manera visual.

En la siguiente figura se muestra cómo se realiza esta operación cuando la entrada contiene múltiples canales.

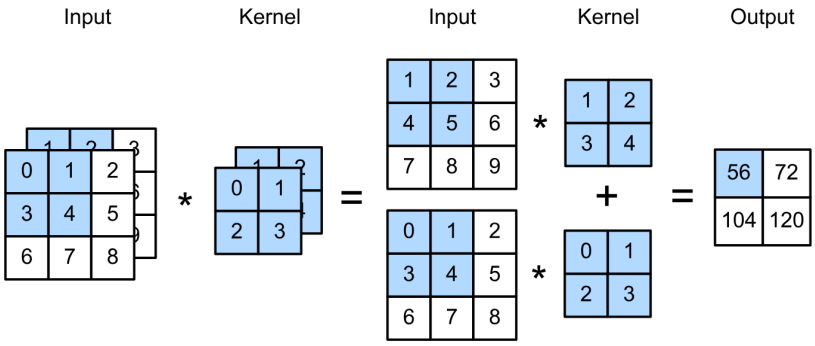

Figura 3.8: Convolución con varios canales en la imagen [24]

También, se puede hacer que una convolución devuelva varios canales en la salida. Esto se hace añadiendo una dimensión al kernel con el número de canales de salida que se quiera obtener.

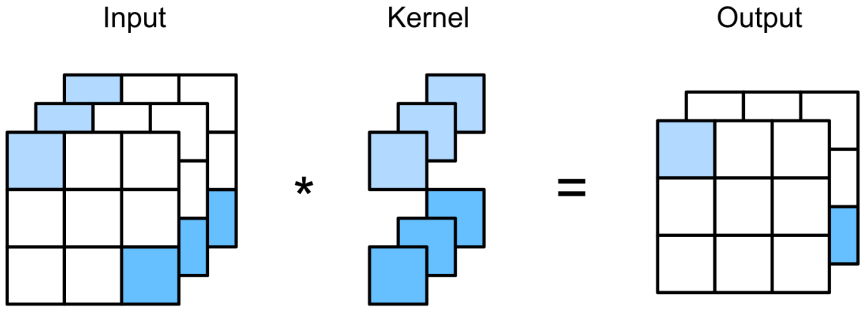

Figura 3.9: Convolución con varios canales en la salida [24]

Esto no es toda la variedad de resultados que nos da una convolución; aún nos queda hablar de dos parámetros muy usados (sobre todo el stride) que nos permiten una mayor flexibilidad.

Stride

Este parámetro define el desplazamiento del kernel al recorrer la imagen. Anteriormente se ha considerado como 1. Si este fuese mayor que 1 disminuirá la resolución de la imagen de salida a $1/stride$, si el padding es 1 y la dimensión del kernel es 1.

Padding

Determina cuántos píxeles de valor cero se rellena el contorno de la imagen de entrada antes de realizar la convolución. A este valor se le ha considerado como 0 en los ejemplos de convolución anteriores. Este parámetro permite conservar (o aumentar ligeramente) la resolución de la salida tras la convolución. Se utiliza comúnmente al trabajar con kernels cuadrados de tamaño mayor que 1, cuando se desea que la resolución de la imagen de salida sea igual o lo más cercana posible a la de la imagen de entrada.

A partir de la siguiente fórmula, se puede conocer cómo será una dimensión de una imagen cuadrada según su dimensión inicial (n), la dimensión del kernel (k), el padding (p) y el stride (s).

$$\text{Dim. salida} = \left(\frac{n + 2p - f}{s} + 1\right)$$

En modelos como YOLO, el número de canales se incrementa progresivamente a medida que se avanza a través de las capas del backbone. Esto suele ir acompañado de una reducción de la resolución espacial mediante técnicas de downsampling, de modo que la pérdida de dimensiones se compensa con una mayor riqueza de representaciones a través de más canales [24]. El objetivo de este incremento es que cada canal capture diferentes conjuntos de patrones relevantes de la imagen original. El downsampling, por su parte, consiste en disminuir las dimensiones de la imagen y puede realizarse mediante distintas estrategias, usando un stride >1, con dimensiones del kernel mayores que 1 o haciendo pooling [25] como se mostrará a continuación.

3.2.2. Pooling

La operación de pooling se lleva a cabo en la capa homónima y tiene como objetivo reducir la resolución de la imagen o del mapa de características, conservando la información más relevante. Para ello, se utiliza un filtro que se desliza sobre la entrada de la capa, aplicando una función de agregación que permite disminuir sus dimensiones en la salida.

Existen diversas funciones de agregación, aunque la más común es el max pooling, que consiste en seleccionar el valor máximo dentro de cada región definida por el filtro. A continuación, se ilustra el funcionamiento de los dos tipos de pooling más utilizados, como son el max pooling, que toma el valor máximo dentro de un patch de la imagen, y el average pooling, que calcula el valor medio de los elementos en dicho patch.

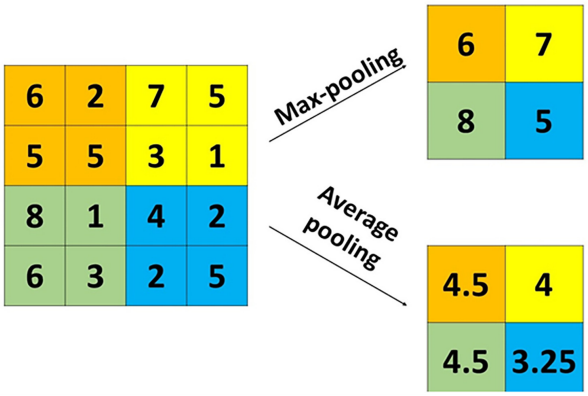

Figura 3.10: Max pooling y average pooling [26]

3.2.3. Capa densa

La capa densa, o totalmente conectada, actúa como un perceptrón multicapa que procesa la salida de las capas anteriores de la red convolucional. Su función principal es combinar las características extraídas para realizar la predicción final. A partir de las representaciones generadas por la red, la capa densa calcula una puntuación o nivel de confianza para cada clase posible, permitiendo así clasificar los datos de entrada.

3.2.4. Upsampling

El upsampling es el proceso opuesto al downsampling y consiste en aumentar las dimensiones de una imagen o mapa de características. Este proceso se utiliza para mejorar su resolución, con el objetivo de hacerlos más adecuados para su posterior procesamiento. En el caso de YOLO, el upsampling también se emplea para combinar mapas de características de baja resolución con otros de mayor resolución, lo que mejora la precisión en la detección de objetos.

Existen diversas técnicas para realizar upsampling, pero la más utilizada en las últimas versiones de YOLO es la interpolación por vecinos cercanos, que explicaremos a continuación.

Interpolación por vecinos cercanos

También conocida en inglés como nearest neighbors interpolation, este metodo de upsampling consiste en asignar al valor del nuevo punto el de su vecino más cercano. Se añade un ejemplo de cómo se hace, aunque como se puede observar, es un método bastante sencillo.

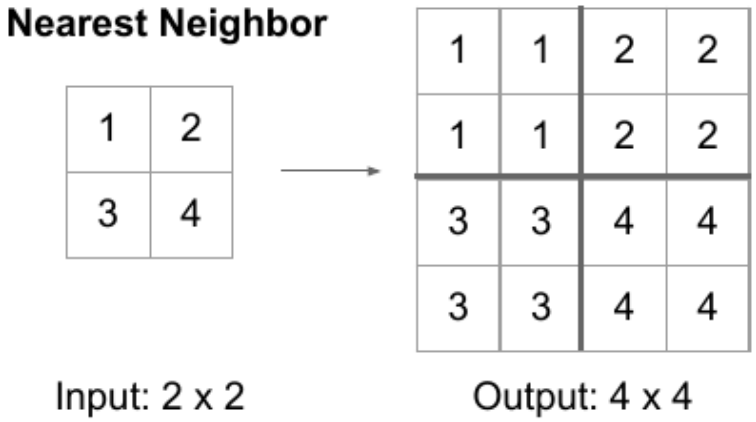

Figura 3.11: Interpolación por vecinos más cercanos [27]

3.3. Atención

El concepto de atención, introducido en los últimos años en el ámbito de la Inteligencia Artificial [28], ha supuesto una transformación en el paradigma del procesamiento de datos, especialmente en áreas como el Procesamiento del Lenguaje Natural y la Visión por Computador. Los mecanismos de atención permiten a los modelos enfocarse en las partes más relevantes de los datos de entrada, asignando pesos o puntuaciones que reflejan su importancia relativa.

En el caso de la Visión por Computador, estos elementos suelen corresponder a fragmentos de la imagen, conocidos como patches. Aquellos con puntuaciones de atención más altas reciben mayor prioridad durante el proceso de cálculo. Este enfoque dinámico permite al modelo centrarse en la información más relevante en cada etapa, lo que contribuye a mejorar tanto la precisión como la contextualización de los resultados.

Así mismo, el uso de mecanismos de atención facilita el procesamiento paralelo de la información, evitando la necesidad de recorrerla de manera secuencial, sin que ello implique una pérdida de información posicional. Debido a su eficacia, este tipo de mecanismos está siendo integrado progresivamente en las versiones más recientes de YOLO, por lo que a continuación se presentan algunas de sus variantes más relevantes.

3.3.1. Atención espacial

Atención espacial o en inglés spatial attention es una de las mejoras que se incluye en YOLOv11 mediante el bloque C2PSA (Cross Stage Partial with Spatial Attention) que se explicará en la subsección de YOLOv11. Este mecanismo permite al modelo centrarse en las partes más importantes de cada imagen.

3.3.2. Auto-atención

En la auto-atención o self-attention en inglés, la atención se usa para entender cómo cada elemento de la entrada se relaciona con los demás para entender el contexto de la imagen. Este es uno de los mecanismos de atención más relevantes y es la base de los Transformers. Se implementa en YOLOv12 mediante la atención de área.

3.3.3. Atención de área

El coste computacional de vanilla attention, es prohibitivo al ser cuadrático, por ello se introduce el area attention que usa un mecanismo de atención local que reduce el coste computacional. Esto se hace dividiendo la imagen en distintas ventanas que separan la imagen en 4 y aplicando la atención en esas ventanas [29]. A continuación, se puede ver cómo se divide la imagen en diferentes ventanas y cuál es la división que se da en la atención de área.

Figura 3.12: Distintas ventanas de atención local [29]

3.3.4. FlashAttention

Este mecanismo mejora la atención estándar en términos de velocidad, al reducir la cantidad de operaciones de lectura y escritura en memoria. Esto lo hace a partir de distintos algoritmos [30, 31]. Este mecanismo es usado en YOLOv12 mejorando la eficiencia del modelo, pero puede dar problemas si no se consigue compilar el FlashAttention en la GPU, volviéndose más lento YOLOv12 que YOLOv11 [32]. En nuestro caso, YOLOv12 tiene una velocidad de procesamiento por imagen similar que su versión anterior (a excepción del modelo más grande como se mostrará en el capítulo del desarrollo del modelo) porque compila el FlashAttention en la GPU usada para el proyecto (RTX 3090). En la documentación de YOLO se puede conocer en qué GPUs FlashAttention compila [33].

3.4. Problemas en el entrenamiento de una red

El entrenamiento de redes neuronales puede enfrentar diversos problemas que afecten la calidad del modelo resultante. A continuación, se describen algunos de los inconvenientes más comunes que pueden surgir durante esta fase, así como sus posibles causas.

3.4.1. Evanescencia del Gradiente

Durante el proceso de retropropagación, los gradientes pueden volverse extremadamente pequeños, lo que dificulta el aprendizaje del modelo. Este problema, conocido como desvanecimiento del gradiente, es especialmente común en redes profundas como las utilizadas en este trabajo. Para mitigar sus efectos, se han propuesto diversas soluciones que han sido implementadas en las últimas versiones de YOLO.

1. **Batch Normalization**: La normalización se hace restando la media y dividiendo por la varianza las salidas de las redes neuronales. De esta manera se reduce la dispersión de los datos, lo que ayuda a que este problema no se produzca.

2. **Conexiones residuales**: Estas permiten que una parte de la información de entrada se transmita directamente a la salida, facilitando el "salto"de una o más capas. Esto favorece un flujo de información más libre entre capas, lo que ayuda a mitigar problemas asociados con el desvanecimiento del gradiente y mejora la capacidad de entrenamiento de redes profundas. Han sido introducidas por un grupo de investigación de Microsoft en 2015 [34].

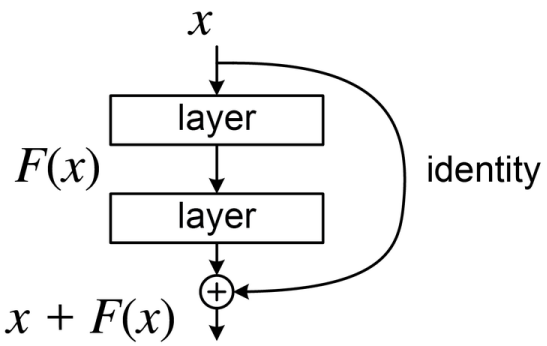

Figura 3.13: Ejemplo de una conexión residual que se salta dos capas [35]

3. **Depthwise separable convolution**:

   También conocido como depthwise convolution o separable convolution es un tipo de convolución particular que se divide en las siguientes partes:

   - **Depthwise convolution**: en la que se utiliza un kernel independiente en cada uno de los canales de la imagen, en vez de usar un kernel con más de un canal para cada canal de salida de la imagen. Cada uno de estos kernel independientes devuelven un canal de salida que se junta con el resto.
   - **Pointwise convolution**: en la que se utiliza un kernel $1x1$ junto con el número de canales de la salida del depthtwise convolution. Con varios de estos kernel se consigue que la salida tenga el número de canales deseado.

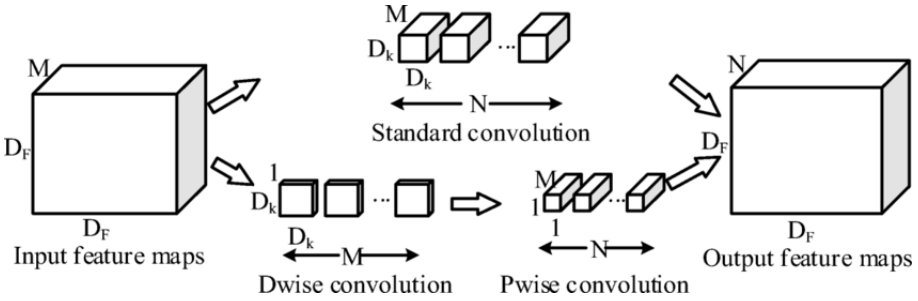

Figura 3.14: Convolution vs depthwise separable convolution [36]

En la imagen anterior, M representa el número de canales de entrada, $D_f \times D_f$ corresponde a las dimensiones del mapa de características, $D_k \times D_k$ a las dimensiones del kernel, y N al número de canales de salida.

En una convolución normal se necesitan N kernels de dimensiones $D_k \times D_k \times M$ y por tanto $M * D_k * D_k * N$ parámetros. Por otro lado, para la depthwise separable convolution sólo se necesitan M kernels de dimensiones $D_k \times D_k \times 1$ para la operación depthwise y para la pointwise convolution se necesitan N kernels de dimensiones $1 \times 1 \times M$ y por tanto se usan $D_k * D_k * M + M * N$ parámetros.

Para poner un contexto particular estos datos, si suponemos que hay 3 canales de entrada y 16 de salida, el mapa de características tiene dimensiones 10×10 y el del kernel 3×3. Para una convolución normal se utilizan 432 parámetros y para una depthwise separable convolution sólo son necesarios 75 parámetros [37].

Por tener menos parámetros el modelo que use esta operación converge más rápido y es menos propenso a la Evanescencia del gradiente y al sobreajuste. Este tipo de convolución es usado en YOLOv11 y YOLOv12.

3.4.2. Sobreajuste

El sobreajuste (overfitting en inglés) ocurre cuando una red neuronal se focaliza demasiado en los datos de entrenamiento, incluyendo información espuria que no contribuye a la solución del problema. Esto hace que el modelo solo dé buenos resultados para el conjunto de entrenamiento, pero sus resultados en los de test sean sensiblemente peores. Para solventar esta dificultad, se tomarán las siguientes medidas:

1. Asegurarse de que el conjunto de datos de entrenamiento sea diverso y con una distribución de clases representativa, lo que haremos mediante la estratificación.
2. Detener el entrenamiento del modelo cuando no se observe mejora en los datos de validación durante varias épocas consecutivas, con el fin de evitar el sobreajuste a los datos de entrenamiento.
3. La Batch Normalization, vista en el apartado anterior 3.4.1, ayuda a prevenir el sobreajuste al estabilizar el entrenamiento, reduciendo la variabilidad de la salida.

3.4.3. Subajuste

El subajuste o infrajuste ocurre cuando el modelo no es capaz de capturar la complejidad de los datos, produciendo errores altos en todos los conjuntos, no solo en el de entrenamiento. Para reducir este problema, se toman las siguientes medidas:

1. Usamos YOLO que es un modelo capaz de capturar la complejidad de los datos.
2. Se garantiza que haya una cantidad suficiente de datos para que el modelo pueda identificar y aprender correctamente los patrones presentes en ellos.

3.4.4. Datos de mala calidad

Si los datos son de mala calidad el modelo no dará buenos resultados para el problema que queremos resolver. Al final, como se dice en el campo de la ciencia de datos, *garbage in, garbage out*, si los datos son basura, los resultados del modelo devolverán basura. Para evitar esto, se han revisado manualmente las etiquetas imágenes o se han etiquetado manualmete. También, se ha buscado que los datos sean diversos y su tamaño sea suficientemente grande.

3.5. Detección de objetos con YOLO

YOLO, por sus siglas en inglés you only look once, es un software dedicado al reconocimiento de objetos en tiempo real basado en redes neuronales convolucionales. El primer modelo de YOLO fue introducido por Joseph Redmon et al [38]. Su principal innovación, que lo distingue de otros algoritmos de detección de objetos, es su capacidad para realizar todo el proceso de detección en una sola fase adelante de la red neuronal, lo que se refleja en su nombre.

A diferencia de otros enfoques que dividen la detección en múltiples etapas, YOLO divide la imagen en una malla de celdas y asigna a cada celda un conjunto de bounding boxes (rectángulos delimitadores) junto con puntuaciones que representan la confianza de que un objeto se encuentra dentro de cada región. Esta arquitectura de detección unificada permite que el modelo sea rápido y preciso, lo que ha contribuido a su evolución continua y a su consolidación como uno de los algoritmos más utilizados para la detección de objetos.

En este proyecto se emplearán las versiones más recientes de YOLO (YOLOv11 y YOLOv12), ya que han mejorado el rendimiento en tareas de detección de objetos de las versiones anteriores. Asimismo, se evaluarán diferentes configuraciones de tamaño de modelo y se compararán sus métricas de desempeño sobre nuestros datos.

3.5.1. Formato YOLO

Para entrenar un modelo YOLO, es necesario que las anotaciones de las imágenes sigan un formato específico, donde cada una de ellas debe tener los siguientes cinco elementos:

- Identificador de clase a la que pertenece.
- Posición relativa del eje x del centro de la anotación en la imagen. Esta posición, al igual que las demás, es relativa a la imagen, siendo 0 la posición más cercana al borde izquierdo y 1 la más cercana al borde derecho.
- Posición relativa del eje y del centro de la anotación en la imagen.
- Ancho relativo de la anotación en la imagen.
- Alto relativo de la anotación en la imagen.

Cada conjunto de anotaciones en una imagen debe guardarse en un archivo .txt con el mismo nombre que la imagen y debe incluir una anotación por cada objeto que se etiquetado.

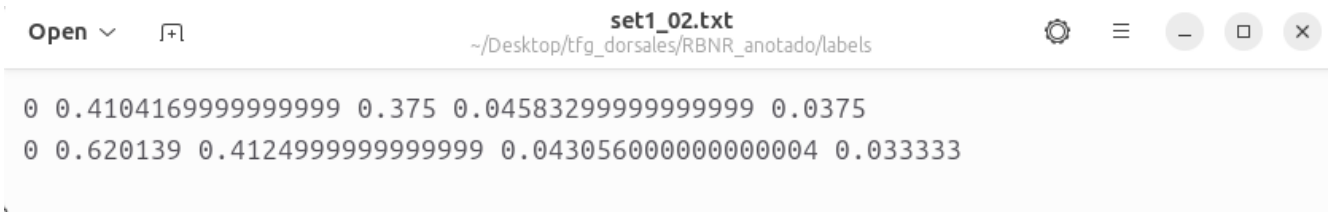

Figura 3.15: Ejemplo anotación YOLO

En la figura 3.15 se puede ver un ejemplo de las dos anotaciones de la imagen set1_02.

Además de las anotaciones y las imágenes, para entrenar los modelos más recientes de YOLO es necesario contar con un archivo data.yaml. Este debe incluir información clave, como el nombre de las clases, el número total de clases y las rutas donde se encuentran las imágenes para el entrenamiento, validación y opcionalmente test.

```
train: ./train/images
val: ./val/images
test: ./test/images

nc: 1  # Número de clases
names: ['bib']  # Nombre de las clases
```

Figura 3.16: Ejemplo data.yaml

3.5.2. Particiones entrenamiento YOLO

Antes de entrenar un modelo YOLO, se debe dividir el conjunto de datos en al menos dos subconjuntos como son el de entrenamiento y el de validación. Adicionalmente, se puede usar un conjunto de test completamente independiente del entrenamiento, como haremos en este proyecto. No existe una proporción ideal para estas particiones, pero comúnmente los porcentajes de estas particiones se encuentran en los siguientes rangos [39]:

Entrenamiento

Se utiliza aproximadamente el 70-80 % de los datos para entrenar la red neuronal.

Validación

Un 10-20 % de los datos se usa para validar los resultados durante el entrenamiento y ajustar el modelo para evitar el sobreajuste.

Test

Un 10-20 % de los datos se emplea para evaluar el rendimiento del modelo una vez entrenado.

3.5.3. Métricas YOLO

IoU

El IoU (Intersection over Union, o Intersección sobre la Unión) [40] es la métrica utilizada por YOLO para evaluar la precisión de los bounding boxes predichos en relación con los etiquetados. Se calcula como el cociente entre el área de superposición de ambos y el área total de su unión. Esta medida, junto con la correcta clasificación en el grupo correspondiente, es fundamental para la construcción de la función de pérdida (loss function) que se emplea en el entrenamiento de los modelos YOLO.

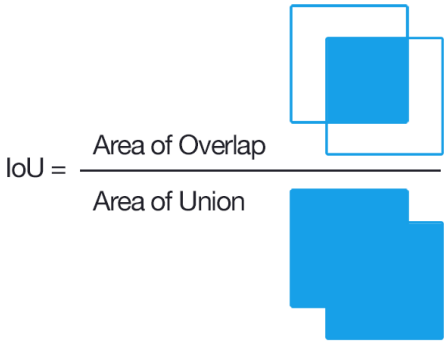

Figura 3.17: IoU[41]

Precision

Evalúa la exactitud de las predicciones realizadas por el modelo. Se define como el número de predicciones hechas correctamente dividido entre los valores positivos. En nuestro caso, esto es el número de dorsales predichos correctamente por el modelo dividido por los predichos.

$$Precision = \frac{Dorsales\ Predichos\ Correctamente}{Dorsales\ Predichos}$$

Recall

Mide la capacidad del modelo para detectar todos los dorsales, minimizando las predicciones incorrectas (falsos negativos). Se define como el número de predicciones correctas entre el número de dorsales etiquetados.

$$Recall = \frac{Dorsales\ Predichos\ Correctamente}{Dorsales\ Etiquetados}$$

F1 score

Es una métrica que une las dos medidas anteriores mediante su media armónica, su rango está entre 0 y 1. Esta media es usada para unir las medidas anteriores y tener una única para comparar varios modelos. Esta métrica muy usada para evaluar conjuntamente distintos modelos.

$$2\frac{Precision \cdot Recall}{Precision + Recall}$$

mAP

El mAP (mean Average Precision) es una métrica ampliamente utilizada en los detectores de objetos para comparar su rendimiento en un conjunto de imágenes. Esta medida se basa en la AP (Average Precision), que representa el AUC (Area Under the Curve) de la curva ROC, donde se representan la precisión y el recall. La AP se calcula para diferentes puntuaciones que reflejan la confianza de la predicción. El score más comúnmente utilizado es el IoU (Intersection over Union). Si el IoU supera un umbral determinado, la predicción se considera correcta, en otro caso se considera incorrecta. El mAP se define como el promedio de las AP para distintos valores de IoU. Normalmente se registran los valores entre 0,5 y 0,95 tomados en intervalos de 0,05 [42].

En conjuntos de datos grandes, como MS COCO, utilizan esta métrica para evaluar el rendimiento de varios algoritmos de detección. Su comparación en grandes conjuntos de datos es relevante porque, generalmente, un modelo que muestra un buen desempeño en grandes volúmenes de datos también lo hará sobre conjuntos más pequeños. En el siguiente gráfico, se puede observar el comportamiento de distintos modelos en esta métrica sobre el conjunto de datos MS COCO (también conocido como COCO) [43].

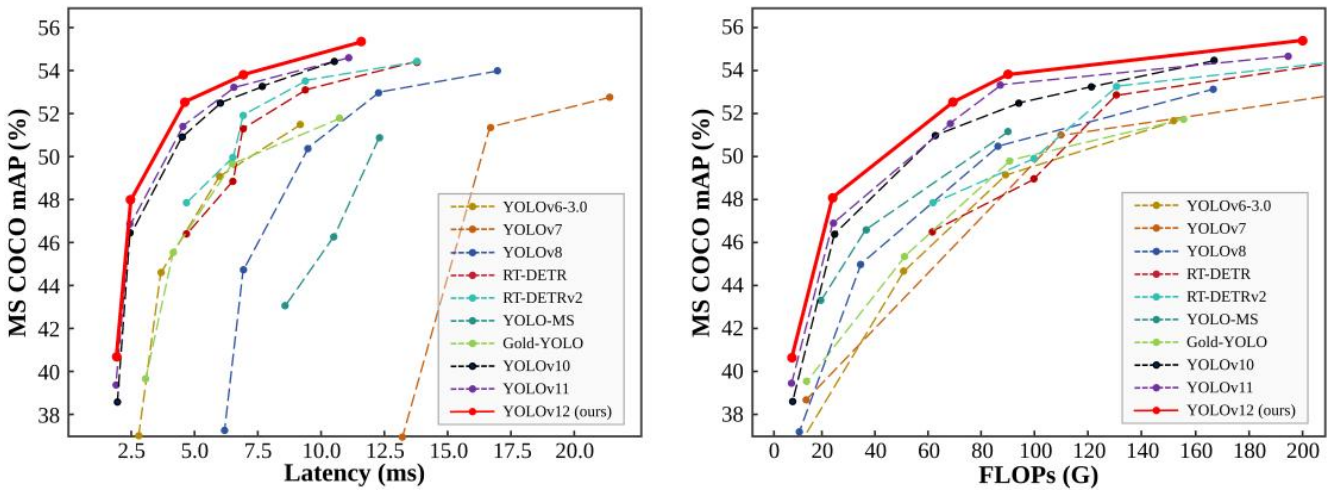

Figura 3.18: mAP y velocidad/número de operaciones en distintos modelos en una T4 GPU [44]

Latencia

Es el tiempo que transcurre entre el procesamiento de la imagen y la devolución de una predicción. Es útil para comparar la velocidad de distintos modelos sobre una misma máquina.

GFLOPS

GFLOPS (Giga Floating Point Operations Per Second) es una medida de rendimiento computacional, que corresponde con el número de operaciones de punto flotante por segundo. En el caso que nos ocupa, representa los millones de operaciones de punto flotante que se necesitan para procesar una imagen.

3.5.4. YOLOv11

Esta versión fue publicada en septiembre de 2024 [45]. A partir de esta, explicaremos el funcionamiento de un caso más concreto de YOLO y las mejoras que se introducen.

Existen distintas subversiones según el tamaño del modelo. En YOLOv11 [46, 47], estas subversiones se diferencian en tres parámetros que marcaremos en distintos colores. En la siguiente tabla se observa la variación de estos parámetros en las distintas subversiones y cuando se describa su arquitectura se podrá ver las zonas en las que aparecen estos parámetros.

Modelo	depth_multiple	width_multiple	max_channels	Millones de parámetros	GFLOPS
YOLOv11n	0,5	0,25	1.024	2,6	6,6
YOLOv11s	0,5	0,5	1.024	9,5	21,7
YOLOv11m	0,5	1,0	512	20,1	68,5
YOLOv11l	1,0	1,0	512	25,3	87,6
YOLOv11x	1,0	1,5	512	57,0	196

Tabla 3.1: Tabla modelos YOLOv11

El significado de estos parámetros es el siguiente:

- depth_multiple determina el número de cuellos de botella en el bloque C3k2, que se explicara más tarde.

- width_multiple y max_channels definen el número de canales de salida de los distintos bloques.

En general, la arquitectura de los modelos YOLO puede ser dividida en tres partes:

1. **Backbone.** Este componente se encarga de extraer las características de la imagen de entrada, generando mapas de características a diferentes resoluciones que capturan los patrones visuales presentes en la imagen. En YOLOv11, el backbone usa bloques C3k2 a diferencia de otros modelos anteriores como se vera más adelante

2. **Neck.** Actúa como puente entre el backbone y el head fusionando y refinando las características que se obtiene del backbone. YOLOv11 hace uso de los módulos SPPF y PANet para el neck.

3. **Head.** Es la parte encargada de realizar las predicciones finales a partir de la información que le pasa el neck. Incluyendo la predicción de los bounding boxes, las clases y la confianza. Esta parte está diseñada para permitir adaptaciones a la tarea según lo que se quiera hacer. En nuestro caso, usaremos la parte adaptada para la detección de objetos.

1.Backbone

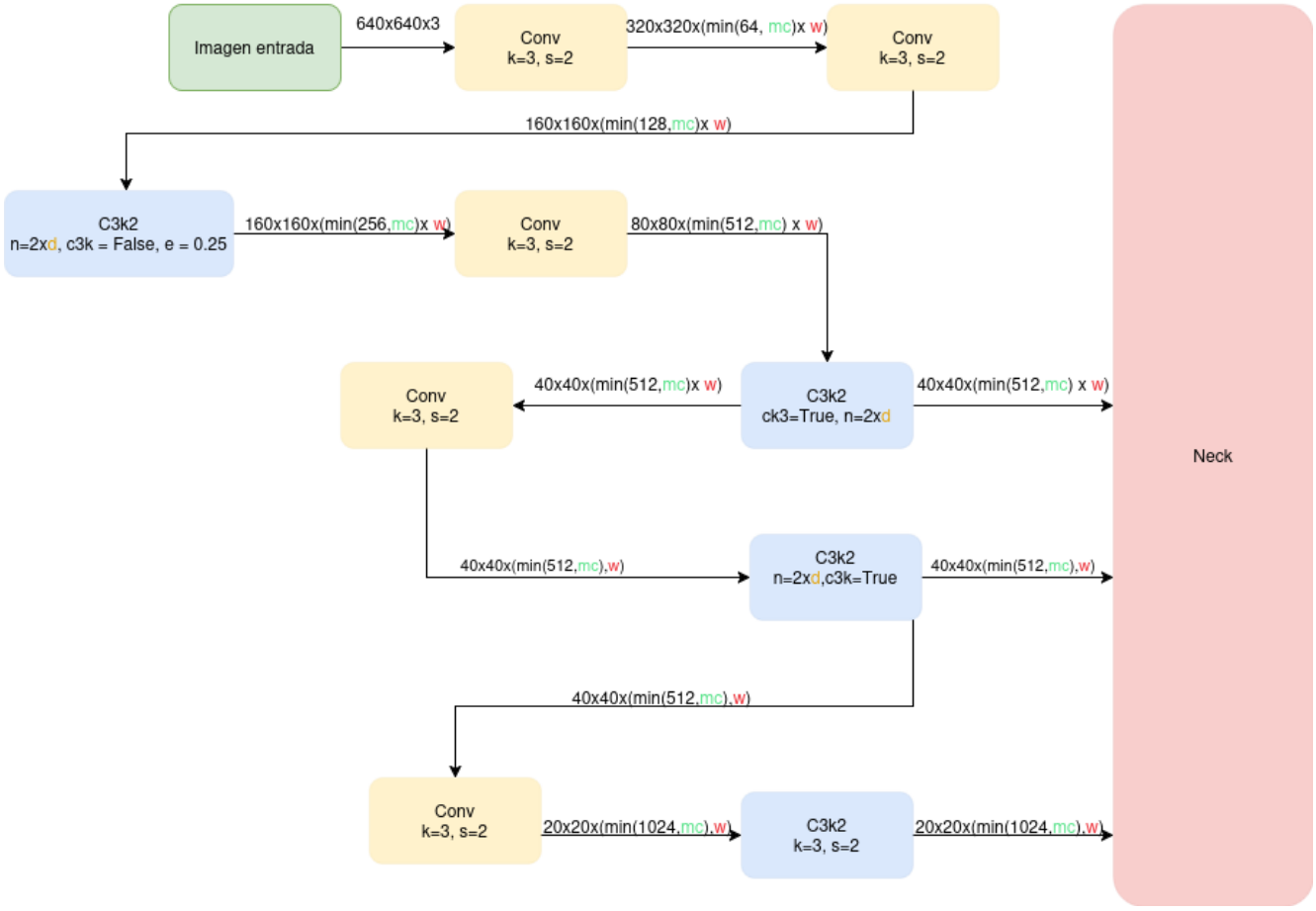

Figura 3.19: Arquitectura backbone basada en [48, 49] al igual que el resto de partes de YOLOv11

En la figura 3.19 se puede observar cómo el backbone de YOLOv11 está compuesto por varios bloques que explicaremos a continuación. Para las figuras, se considera k como la dimensión del kernel, s como el stride, p como el padding y c como el número de canales.

Conv

Son los bloques convolucionales con función de activación SiLU, kernel=3 y stride=2.

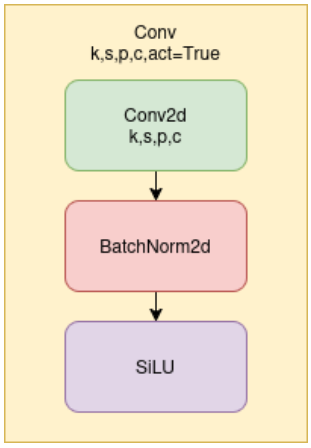

Figura 3.20: Bloque convolucional estandar en YOLO

C3k2

Es una mejora del bloque C2f de YOLOv8 y YOLOv10 introduciendo menos parámetros. Internamente, a partir del parámetro c3k se hace una bifurcación entre bolttleneck y el módulo c3k.

El bottleneck plantea dos alternativas con dos capas convolucionales, pero la diferencia entre ellas es la aparición de una conexión residual o no entre la entrada y la salida

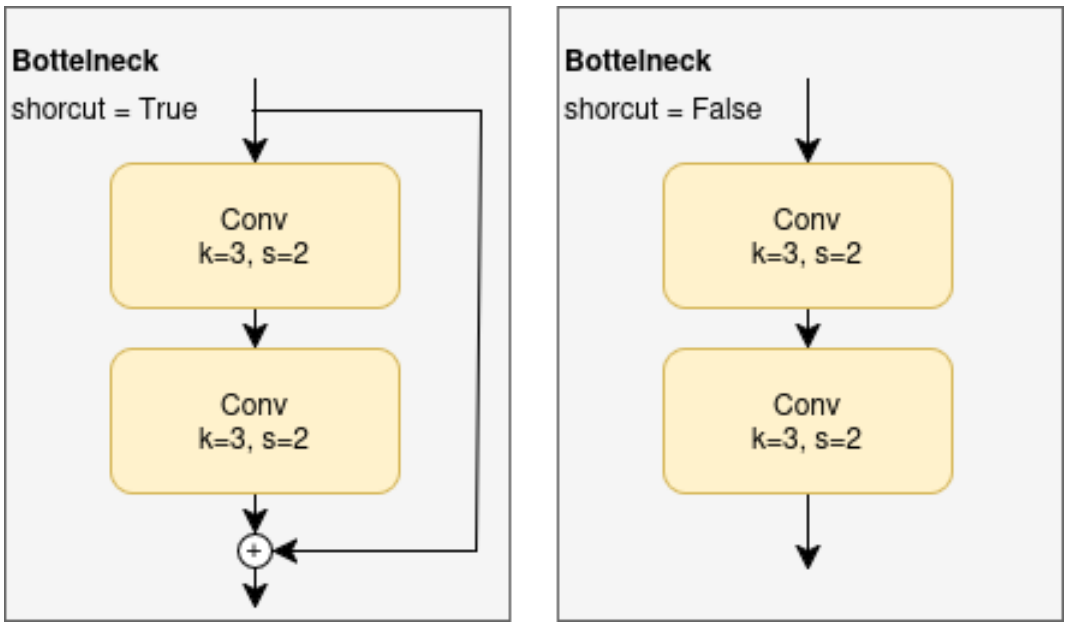

Figura 3.21: Bloque bottleneck

El módulo c3K forma parte de C3k2 si el parámetro c3k es positivo. Este está formado por tres capas convolucionales y varios bottleneck.

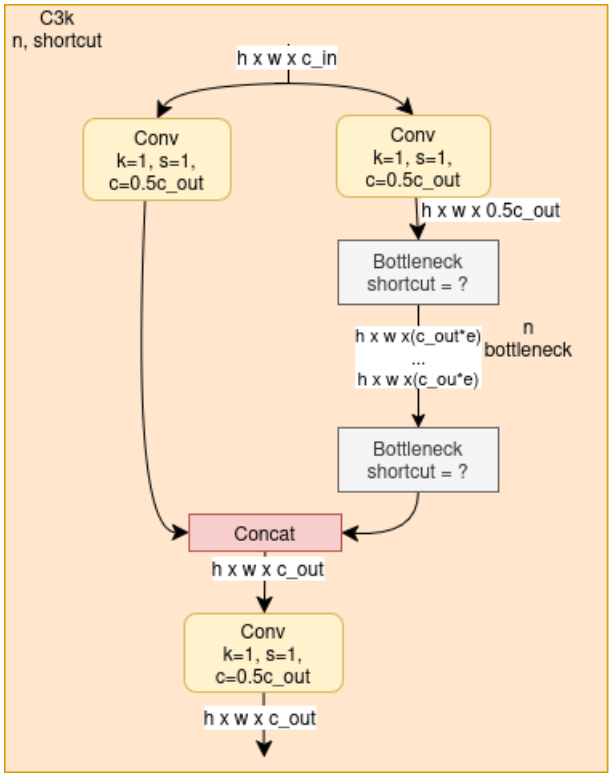

Figura 3.22: Bloque c3k

La estructura completa del C3k2 formada a partir de los módulos explicados anteriormente, se puede observar en la siguiente figura.

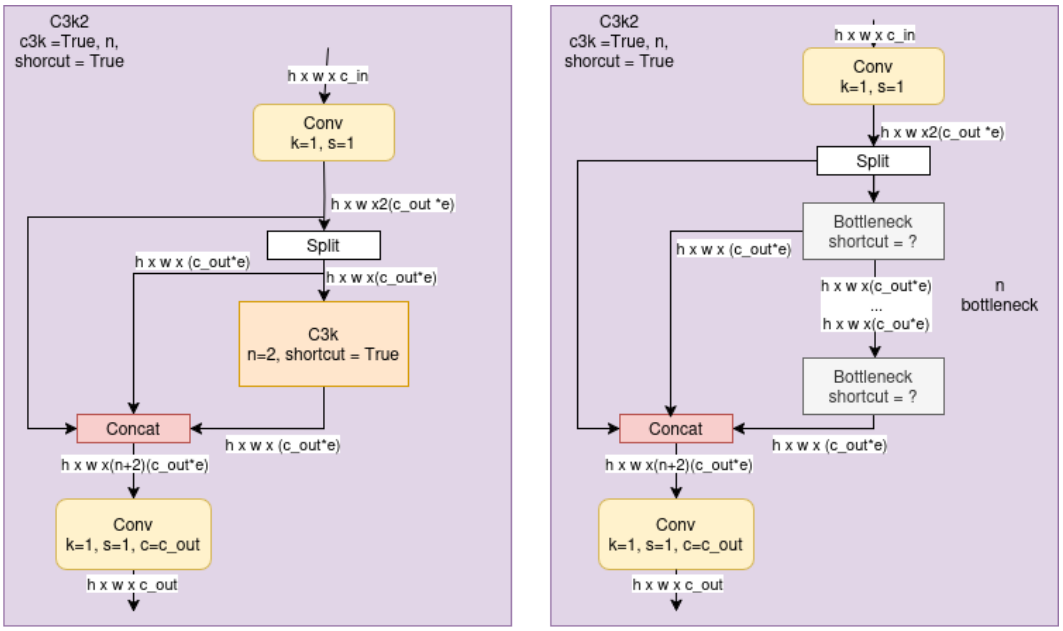

Figura 3.23: Bloque C3k2

2.Neck

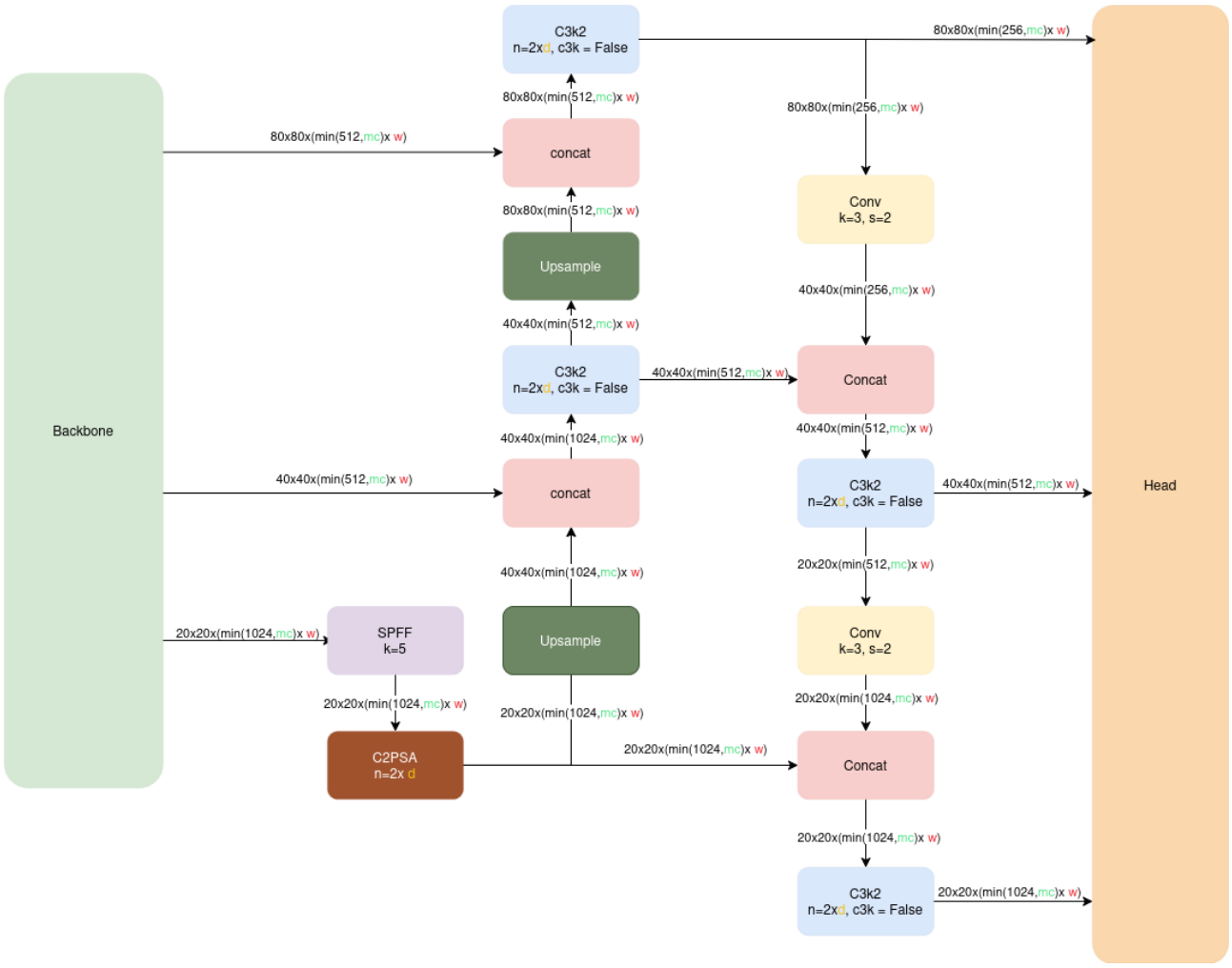

Figura 3.24: Arquitectura neck YOLOv11

Como se observa en la figura 3.24, en el neck aparecen nuevos bloques, como los siguientes:

Concat

Estos bloques permiten concatenar en una sola matriz el resultado de capas profundas con capas más superficiales.

SPPF

Este bloque viene de las siglas en inglés Spatial Pyramid Polling Fast. Se usa para representar mapas de características a diferentes escalas y su estructura es la siguiente:

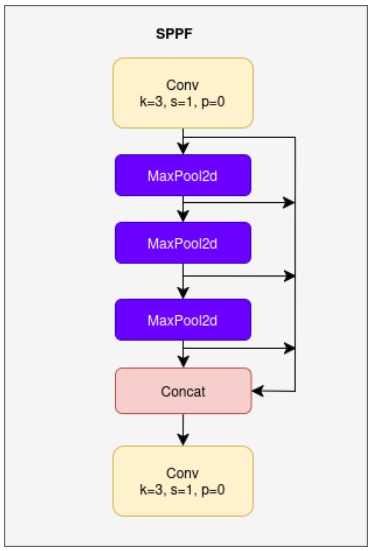

Figura 3.25: Bloque SPPF

C2PSA

Emplea autoatención permitiendo que YOLOv11 se centre de manera más efectiva en las áreas importantes de una imagen. Está compuesto por bloques PSA (Partial Self Attention) para identificar en qué partes de la imagen es necesaria la atención. A su vez, los bloques PSA contienen un bloque de atención y dos capas convolucionales. C2PSA captura características más profundas que el bloque PSA sin gasto adicional de tiempo, ya que la convolución de salida de la división inicial separa la entrada en dos vías: un conexión residual y un procesamiento profundo. A continuación se puede ver la estructura de estos bloques:

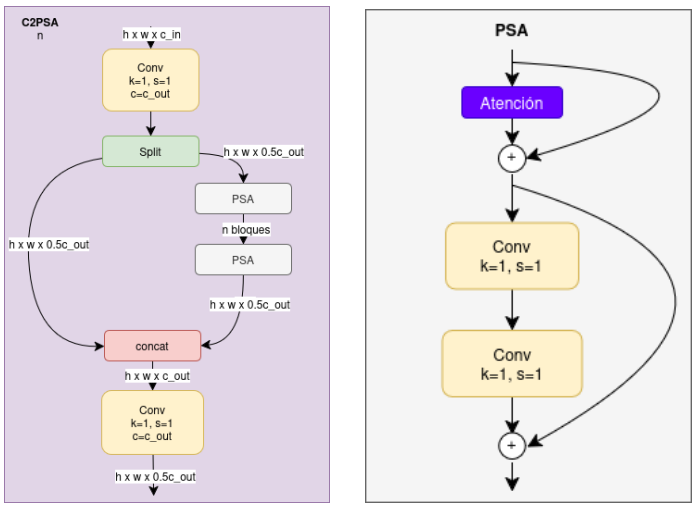

Figura 3.26: Bloques C2PSA y PSA

Upsample

Este bloque se utiliza para aumentar la resolución espacial de los mapas de características, basandose en el algoritmo nearest neighbours.

3.Head

Recoge las características del neck y es la encargada de predecir las bounding boxes. Está compuesta por varios partes encargadas de detectar bounding boxes de distinto tamaño. En este punto, se calcula la probabilidad de que cada bounding box pertenezca a una determinada clase junto con sus coordenadas.

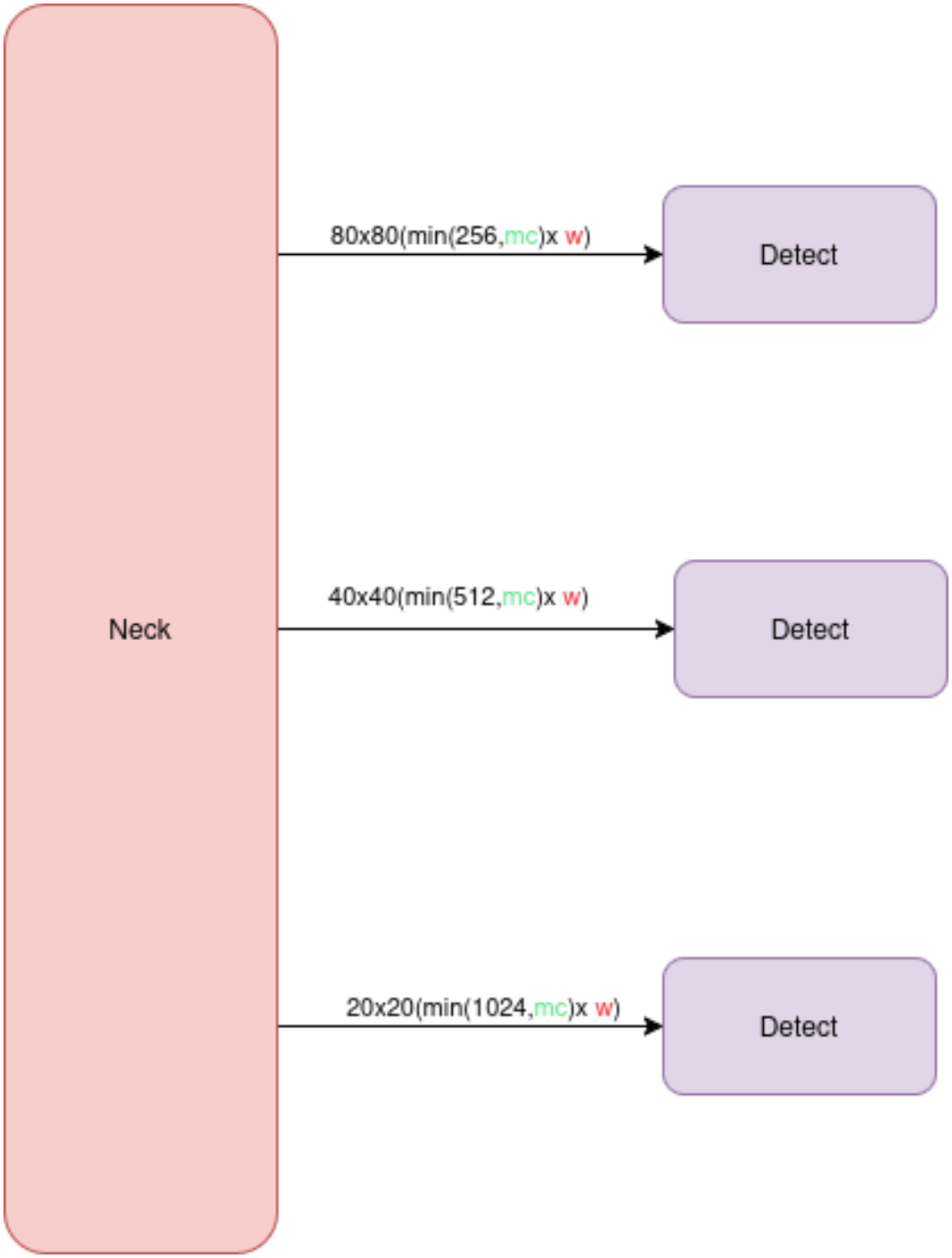

Figura 3.27: Arquitectura head YOLOv11

Como se ve en la figura 3.27, head está compuesto por tres bloques detect cuya estructura se explicará a continuación.

Detect

Está formado por dos partes que sirven para predecir el bounding box y la clase correspondiente. Estos componentes están compuestos por distintas capas convolucionales como se ve en la siguiente figura:

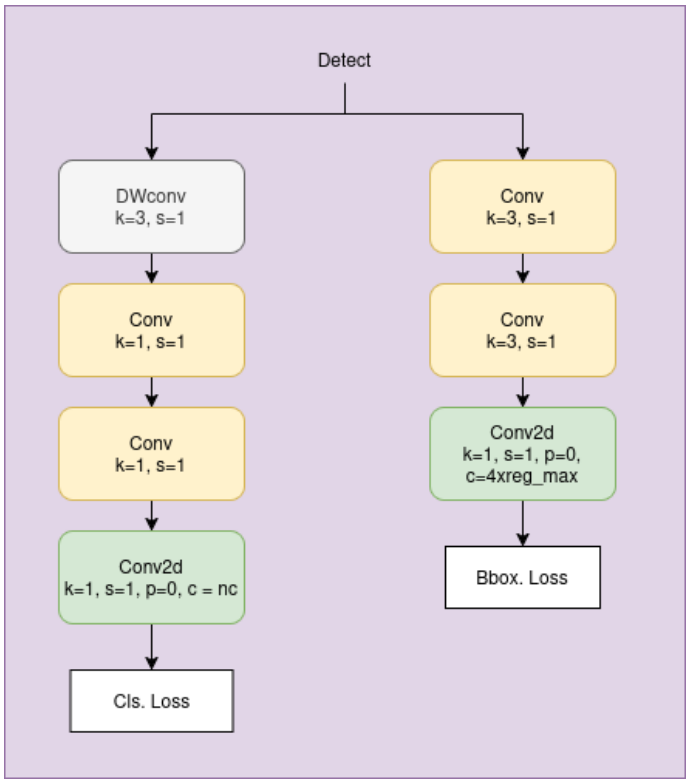

Figura 3.28: Bloque detect

El bloque que aparece como DWConv que realiza la depthwise separable convolution.

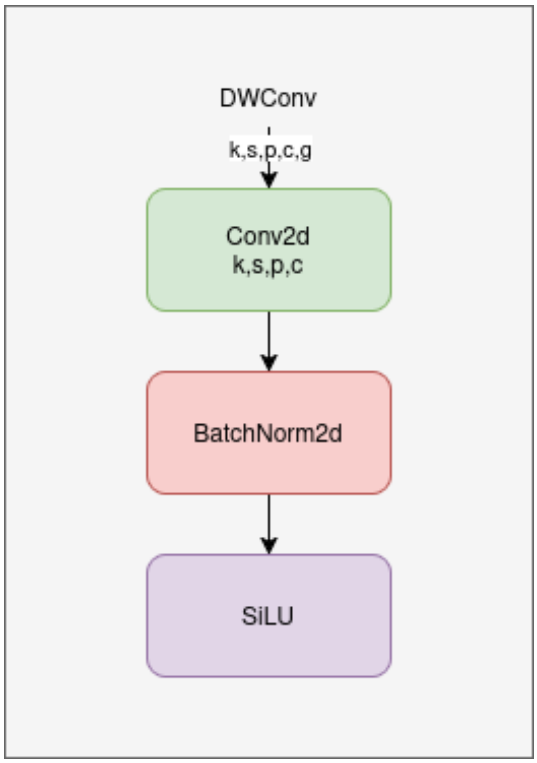

Figura 3.29: Capa convolucional depthwise

3.5.5. YOLOv12

YOLOv12 [50, 29] representa la versión más reciente de YOLO. Introduce múltiples innovaciones, destacándose por su enfoque centrado en mecanismos de atención. Mantiene la estructura modular característica de versiones anteriores con subversiones adaptadas a distintos tamaños.

Modelo	depth_multiple	width_multiple	max_channels	Millones de parámetros	GFLOPS
YOLOv12n	0,5	0,25	1.024	2,5	6,2
YOLOv12s	0,5	0,5	1.024	9,1	19,7
YOLOv12m	0,5	1,0	512	19,7	60,4
YOLOv12l	1,0	1,0	512	26,5	83,3
YOLOv12x	1,0	1,5	512	59,4	185,9

Tabla 3.2: Tabla modelos YOLOv12 [51]

Arquitectura

Se mantiene la arquitectura tradicional de YOLO con sus tres partes backbone, neck y head. Pero, se introducen varios módulos nuevos, como se puede observar en la siguiente figura:

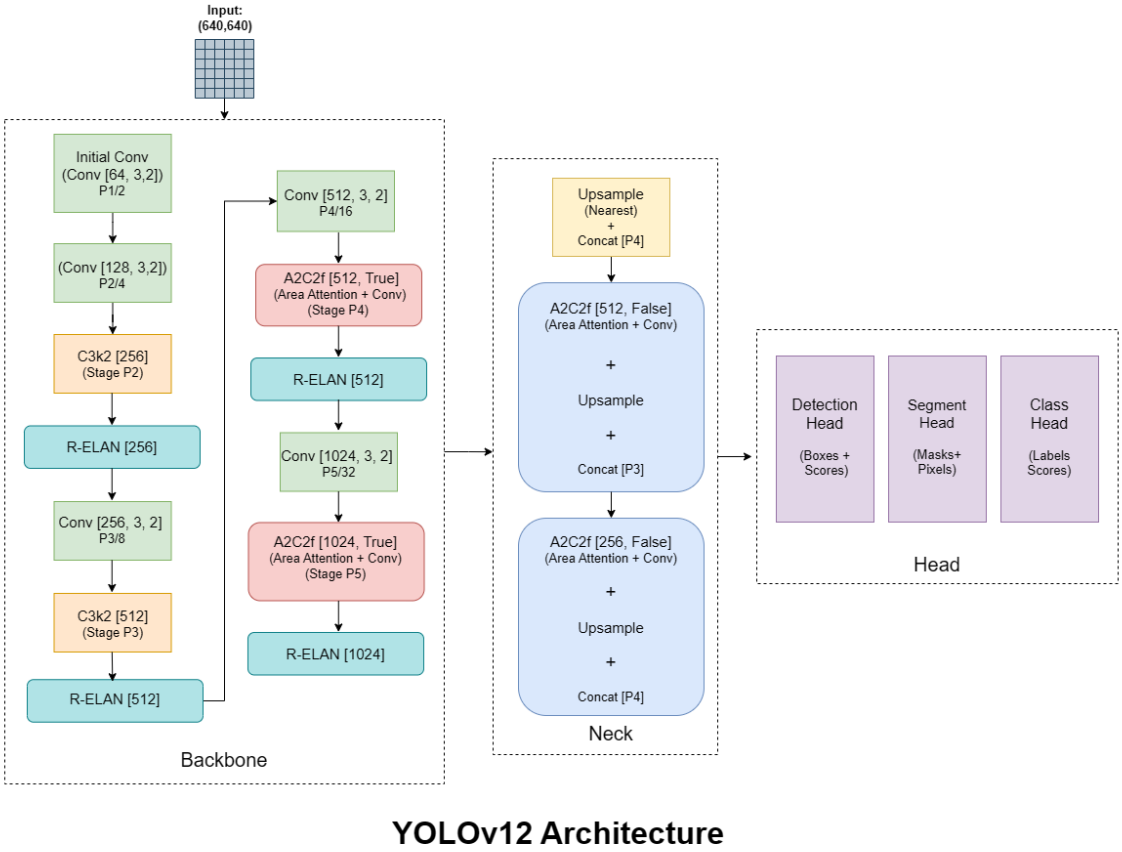

Figura 3.30: Arquitectura YOLOv12 [52]

Las novedades más relevantes en la arquitectura son la introducción de los bloques A2C2f y R-ELAN que se explicarán brevemente a continuación.

R-ELAN

El bloque R-ELAN o por sus siglas en inglés Residual Efficient Layer Aggregation Network, introduce un bloque en el para la extracción de características mediante FlashAttention con un ratio de expansión de su MLP de 1,2 y depthwise separable convolution con un kernel de dimensiones 7×7.

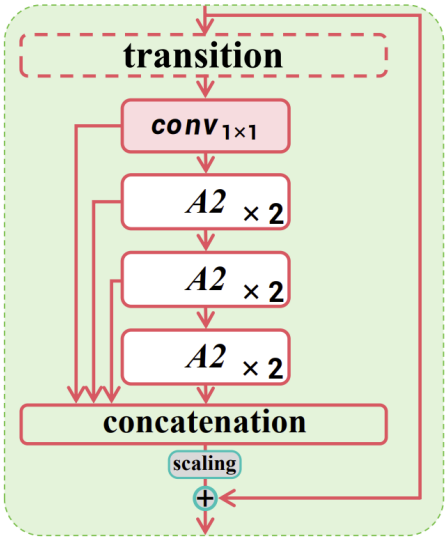

Figura 3.31: Bloque R-ELAN [53]

A2C2f

Usado en el backbone y el neck de YOLOv12, mejora la fusión de características para lograr una mejor comprensión espacial y contextual, incorporando mecanismos de atención de área.

3.6. Reconocimiento con OCR

Una vez detectada la región donde se encuentra la información numérica del dorsal, el siguiente paso es identificar correctamente el número representado. Para esta tarea se emplea un sistema de reconocimiento óptico de caracteres, conocido como OCR (por sus siglas en inglés, Optical Character Recognition).

3.6.1. Funcionamiento de un OCR

Explicaremos brevemente los pasos que sigue un OCR para identificar texto [54].

1. **Adquisición de la imagen**. Cuanto mayor sea la calidad y la resolución mejores resultados se esperan.
2. **Preprocesamiento**. Antes de extraer información, se prepara la imagen para mejorar la salida del OCR, para ello existan distintas técnicas, como las siguientes:
   - **Binarización**: se converte de la imagen a blanco y negro para resaltar el texto.
   - **Eliminación de ruido**: se aplicacan de filtros para reducir artefactos visuales, como mancahas aleatorias o líneas borrosas.
   - **Corrección de inclinación**: se ajusta la orientacion para alinear el texto horizontalmente.
   - **Normalización**: se homogeneiza el tamaño y la iluminación de la imagen.

3. **Segmentación**. Consiste en detectar los pixeles que se corresponden con texto. Esta etapa se detectan los bloques de datos y se dividen en líneas, palabras y caracteres individuales. En esta etapa y la siguiente se suelen usar redes convolucionales y otros tipos de redes neuronales. En nuestro caso, esta parte esta parcialmente hecha por YOLO, que ya detecto la zona donde se encuentra el número del dorsal que se busca reconocer.

4. **Extracción de características y clasificación**. De cada carácter segmentado se extraen características relevantes (contornos, formas, patrones de líneas) que permiten su clasificación.

5. **Reconocimiento y postprocesamiento**. Una vez obtenida la información alfanumérica, se verifica su congruencia y, en caso de detectar inconsistencias, se corrigen errores gramaticales y ortográficos.

Tras la aplicación del OCR se consigue una salida en formato string. Esta se compara con la real para comprobar si ha habido un acierto. Para este trabajo se han explorado distintos OCR en Python como pyteseract, EasyOCR y paddleOCR.

Capítulo 4

Conjunto de datos

Los datos utilizados en este proyecto incluyen imágenes de distintas carreras, acompañadas de las anotaciones correspondientes a los dorsales y sus respectivos números. Para su creación, se han recopilado fotografías de distintas competiciones deportivas, obtenidas de fotógrafos de las mismas o de fuentes públicas y accesibles. Es importante señalar que, al inscribirse en las carreras, los participantes ceden sus derechos de imagen y aceptan que se les tomen fotografías durante el desarrollo del evento. De hecho, es común que parte de estas imágenes aparezcan en la prensa o en las redes sociales de los organizadores de la carrera y de los participantes. Aún así, para no dar información personal irrelevante en este proyecto se difuminarán las caras de las fotos que pudieran permitir identificar personas. A continuación, se explicará de dónde se han sacado las fotos y se darán ejemplos de estas imágenes.

4.1. Procedencia

Las fotos se sacan de imágenes de carreras en repositorios públicos sin copyright o de una carrera con permiso del fotógrafo. Estas se clasifican en diferentes sets según la carrera a la que corresponden.

4.1.1. sets 1-3

Estas imágenes provienen de un proyecto realizado hace varios años que intentó llevar a cabo el reconocimiento de dorsales en 2011, aunque los resultados obtenidos son bastante pobres [55].

Las entradas se componen de fotografías que no coinciden con las dimensiones requeridas por los modelos YOLOv11 y YOLOv12 (640x640 píxeles). Por esta razón, antes de ser procesadas,deben someterse a una transformación que consiste en añadir padding si ambas dimensiones son menores a 640 píxeles. En el caso de que una sea mayor que 640, se coge la más grande y se reduce a 640 y la otra se modifica para mantener la relación de aspecto. Esta transformación es realizada automáticamente por YOLO.

A continuación, se presentará información sobre cada una de las carreras de estos tres primeros set, acompañada de ejemplos fotográficos correspondientes.

Set 1

Contiene 92 imágenes y 108 dorsales, siendo la mayoría de ellas tomadas en la meta. Hay dos tipos de dorsales en esta carrera: uno con los números en rojo y otro en negro. Los números ocupan casi todo el espacio del dorsal y van desde dígitos de dos hasta cuatro cifras. Se ilustran imágenes de estos dorsales:

(a) Imagen números negros (b) Imagen números rojos

Figura 4.1: Ejemplos imágenes set1

Set 2

Con 67 imágenes y 94 dorsales. Las fotografías fueron tomadas en distintos puntos de la carrera, incluyendo tanto a corredores en movimiento, como a personas detenidas. Los dorsales constan de 5 cifras, impresas sobre un fondo de distintos colores, no ocupando toda la superficie del dorsal, dejando así franjas destinadas a la publicidad. A continuación, se pueden ver algunos ejemplos de esto:

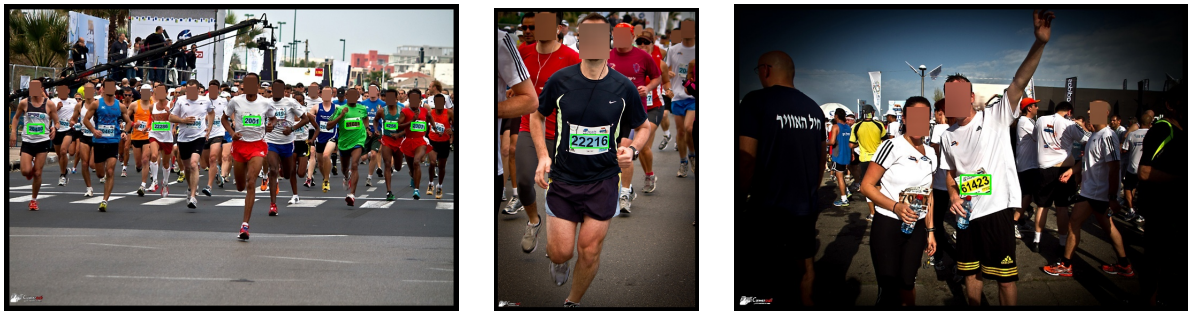

Figura 4.2: Ejemplos imágenes set2

Set 3

Se incluyen 58 imágenes y 153 dorsales. Las fotografías fueron tomadas en un único punto de la carrera y en la zona de salida, con una mayor densidad de dorsales por imagen en comparación con los conjuntos anteriores. Aunque los dorsales presentan un pequeño margen alrededor de los números (de cuatro cifras) estos son considerablemente grandes.

Figura 4.3: Ejemplos imágenes set3

4.1.2. Sets 4-6

Está sacado de roboflow [56]. Las imágenes tienen una alta resolución, pero se ha reducido a 640x640, que es el tamaño de la entrada de YOLO. Se ha descartado alguna imagen de ellas debido a que no ha sido posible reconocer ningún de los números de los dorsales.

Set4

Hay 178 imágenes y 323 dorsales. Como novedad respecto a otros conjuntos, las fotografías pueden ser nocturnas o diurnas. Hay muchas personas en ellas, que a veces presentan dorsales borrosos. El formato del dorsal tiene bastante margen y números de 5 cifras.

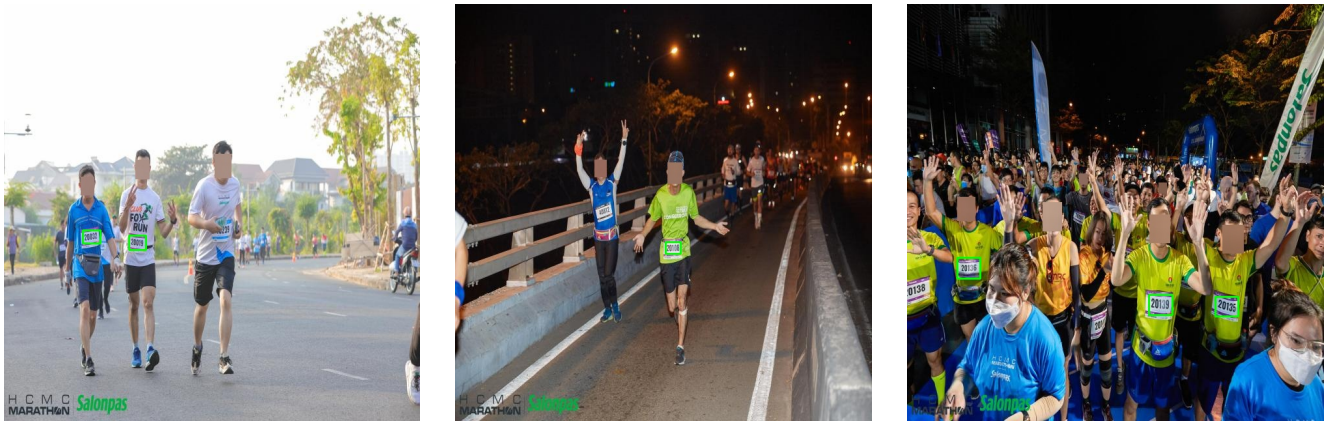

Figura 4.4: Ejemplos imágenes set4

set5

Contiene 170 imágenes y 258 dorsales. El fondo de los dorsales en la mayoría de las fotos es amarillo, aunque hay alguna excepción en la que es rojo. Cuentan con un margen amplio en la parte inferior y los números tienen 5 cifras.

Figura 4.5: Ejemplos imágenes set5

set6

Presenta 120 imágenes y 170 dorsales. Estos tienen bastante margen, el color del número es rojo en la mayoría de los casos, aunque alguno es azul y hay una letra que indica el sexo de la persona (F o M), que no nos interesa. Cada dorsal tiene 5 cifras como se puede ver a continuación:

Figura 4.6: Ejemplos imágenes set6

4.1.3. Sets 7-8

Estas fotografías provienen de un repositorio público sin derechos de autor [57], que además de capturar y publicar imágenes de carreras, también redacta artículos sobre ellas. En concreto, se utilizarán las correspondientes a las ediciones de 2025 de la Mitja de Granollers y la Mitja de Sant Cugat. Estas fotografías han sido seleccionadas porque los dorsales incluyen el nombre del corredor, una práctica cada vez más común y que enriquece nuestro datos con este tipo de ejemplos con los que no contábamos.

Set 7

Tiene 140 imágenes y 246 dorsales. Representando estos una variedad de colores, siendo de gran tamaño, contando con amplios márgenes y mostrando el nombre del corredor en la parte inferior. Las imágenes han sido extraídas de los álbumes 1, 4 y 9, publicadas por el autor de Blogmaldito [58].

Figura 4.7: Ejemplos imágenes set7

Set 8

Hay 49 imágenes y 77 dorsales. Los dorsales son parecidos a los anteriores, pero incluyen publicidad y el nombre aparece encima del número. Las imágenes han sido sacadas de los álbumes 2 y 5, su autor es el mismo que el de la carrera anterior [59].

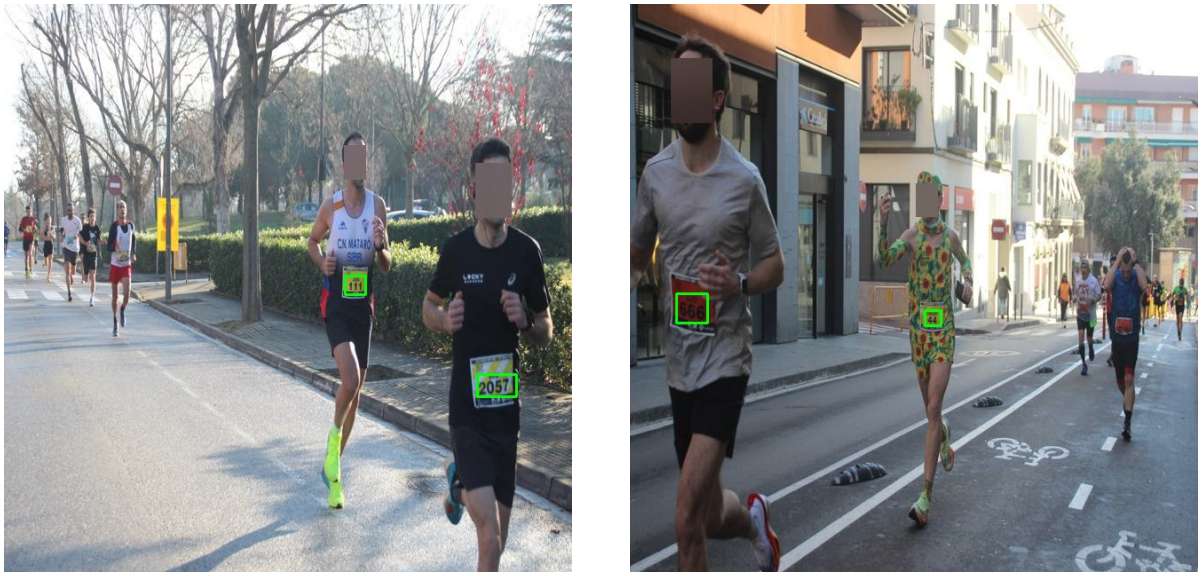

Figura 4.8: Ejemplos imágenes set8

4.1.4. Set 9

Este conjunto de imágenes proviene de la Carrera de las Familias de Valladolid 2024. Las fotografías fureon tomadas por Darío Delgado Vítores y entre estas se incluyen algunas con una alta densidad de dorsales. El set está compuesto por 30 imágenes y 89 dorsales. Estos son de gran tamaño, presentan dos bandas de publicidad en la parte superior e inferior, y los números aparecen en color morado o negro, dependiendo de si el participante completaba la distancia de 10 o 5 kilómetros, respectivamente.

Figura 4.9: Ejemplos imágenes set9

4.2. Etiquetado

Para esta parte se ha utilizado Label Studio [60], una herramienta de código abierto diseñada para la anotación de datos en diversos formatos, como texto, imágenes, audio o vídeo. A continuación, se presentará brevemente su interfaz gráfica y se explicará el proceso seguido durante la fase de etiquetado.

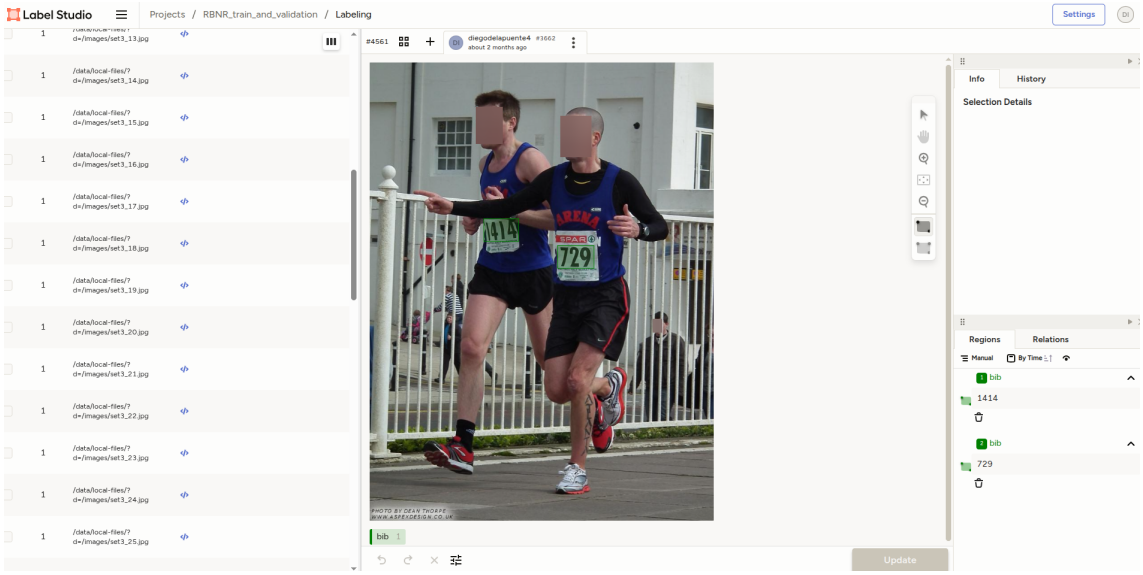

Figura 4.10: Interfaz de anotación

En la figura anterior 4.10 se muestra el mecanismo de anotación. En este proceso, se seleccionan los rectángulos correspondientes a los dorsales y se anota su número.

Antes de empezar a etiquetar, hay que configurar el *label settings* e indicar dónde se encuentran las imágenes. [61].

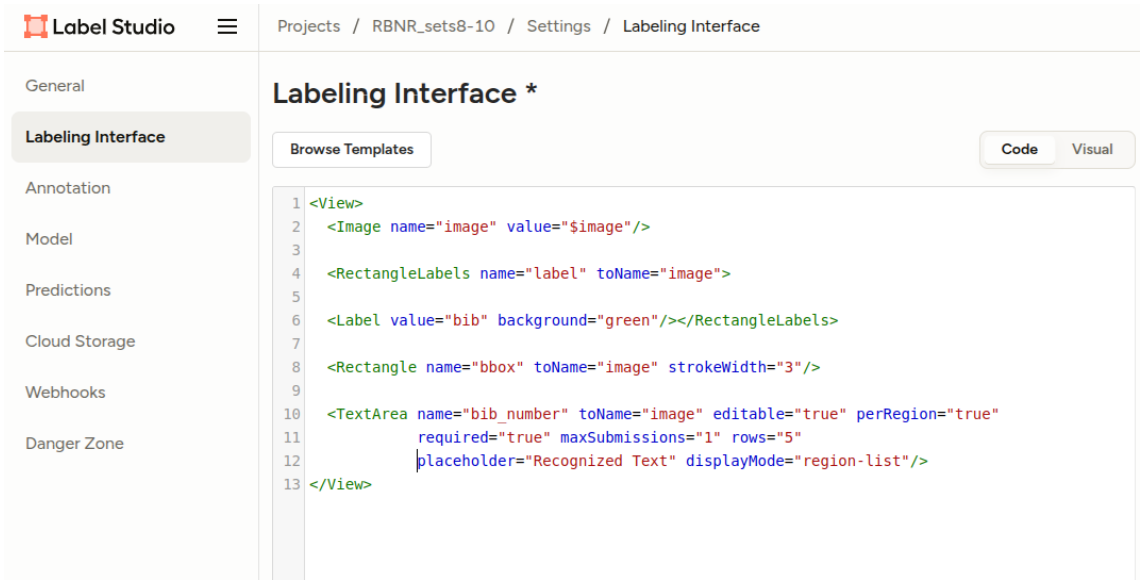

Figura 4.11: Configuración formato etiquetado

Una vez finalizado el etiquetado, Label Studio permite exportar las anotaciones en distintos formatos. En nuestro caso, optamos por el formato YOLO, el cual genera las imágenes y sus respectivas anotaciones en las carpetas images y labels.

4.3. Preprocesamiento

Tras la fase de etiquetado, se exportan las anotaciones y se crea una carpeta que relaciona las imágenes con su número de dorsal llamada bibs_numbers. Se ordenan los dorsales en las imágenes de izquierda a derecha según el eje x del centro de la imagen. Esto último se hace para facilitar las etapas posteriores de comprobación del funcionamiento de YOLO junto al OCR.

4.4. Agrupación imágenes

En total, se han recopilado 904 imágenes que contienen 1.518 dorsales. Como es de esperar, los dorsales son diferentes con respecto al tamaño, al color, al número de dígitos y otras características. Por esta razón, se propone agrupar las imágenes en función de las características de los dorsales. Con este procedimiento, se consigue que las imágenes estén debidamente representadas en los conjuntos de entrenamiento, validación y test.

Aunque sería posible realizar múltiples agrupaciones basadas, por ejemplo, en la ubicación donde se tomó la fotografía (desde la salida hasta la meta), en las condiciones de iluminación, o en otros criterios relacionados con las imágenes, en este trabajo se opta por centrarse en la agrupación según los dorsales, por ser el elemento más importante de la imagen al buscar reconocer su número. La clasificación se llevará a cabo considerando dos criterios: el número de dorsales presentes en cada imagen y el formato específico de los mismos, ya que estos son los factores que mejor representan su variabilidad.

4.4.1. Número de dorsales

Con esta clasificación se intentará evitar que en cualquiera de los conjuntos haya demasiadas imágenes con pocos o demasiados dorsales. A continuación, se puede observar un gráfico con el número de ellos que aparecen en cada imagen.

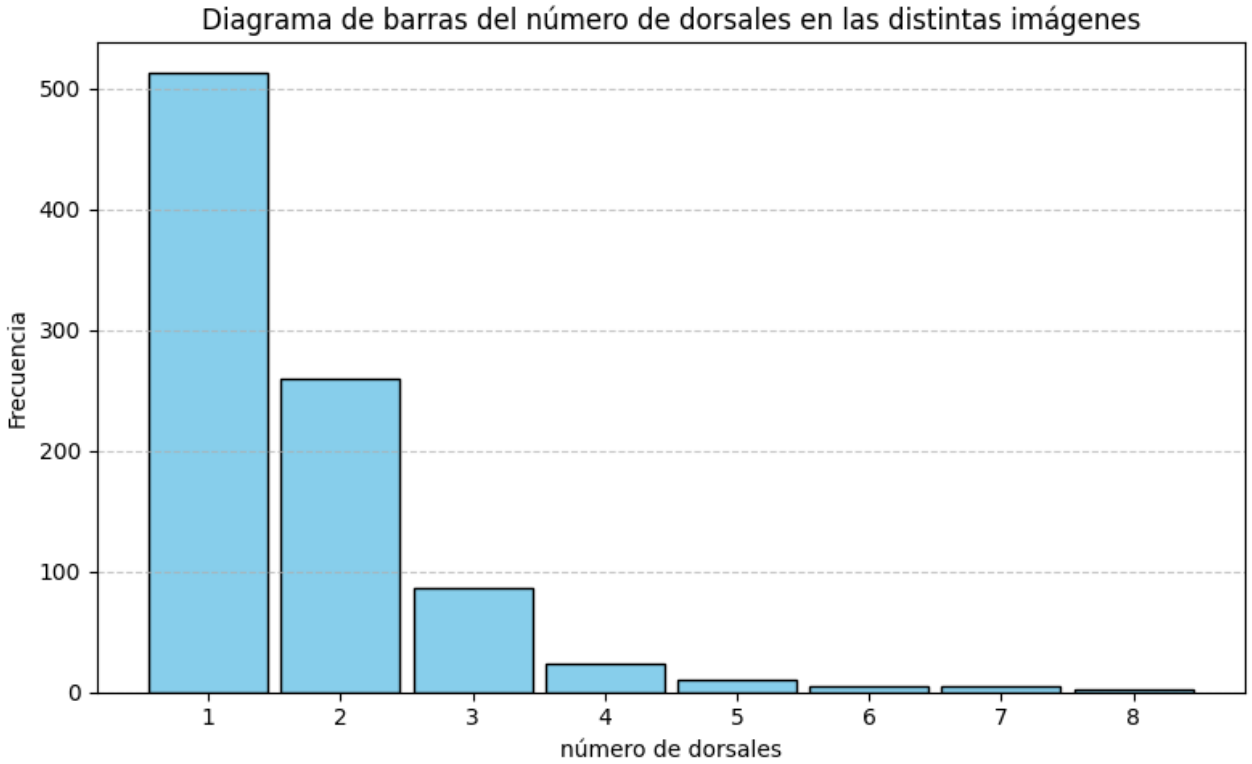

Figura 4.12: Diagrama de barras del número de dorsales por imagen

En el gráfico anterior se observa que no hay muchos de ellos por imagen. Esto sucede porque hay más dorsales de los que aquí se registran, pero en estos no son reconocibles los números. Consideramos las siguientes categorías de las imágenes según su cantidad de dorsales:

- Un dorsal.
- Dos dorsales.
- Más de dos dorsales.

4.4.2. Formato del dorsal

Como se dijo anteriormente, el formato puede variar de una carrera a otra. En este trabajo se consideran los siguientes formatos según la relación de tamaños entre los números de los dorsales y sus margenes:

Números grandes y margenes pequeños

En este grupo añadiremos las imágenes de los sets 1, 3 y 9.

Margen arriba y abajo

En este grupo estarán las imágenes de los sets 2, 4 y 5.

Números pequeños y margenes grandes

En este grupo estarán las imágenes de los sets 6, 7 y 8.

4.5. Estratificación

A partir de la agrupación de las imágenes en los diferentes grupos descritos anteriormente, realizaremos una estratificación. Esta se emplea para mantener la distribución de clases en los distintos conjuntos de entrenamiento, validación y test. El código usado para la estratificación se puede ver en el siguiente anexo A.

Para realizar esta estratificación, primero es necesario clasificar las imágenes. Los número de dorsales de cada una se extrae de las etiquetas ubicadas en la carpeta labels para cada imagen y se almacena en un archivo .csv. El formato de dorsal depende de la carrera en la que se saca la imagen.

A partir de estos dos archivos .csv, realizamos la estratificación utilizando una validación cruzada k-fold con k = 3, generando así los conjuntos de entrenamiento, validación y prueba.

Capítulo 5

Desarrollo del modelo

Para el modelo se utilizarán diferentes tamaños de YOLOv11 y YOLOv12, manteniendo las opciones de entrenamiento por defecto, salvo el tamaño de batch, que se fijará en 8 debido a limitaciones de memoria en los modelos de más grandes. Una vez detectados los dorsales con YOLO, se empleará el OCR PaddleOCR [62, 63], por ser la opción que mostró el mejor desempeño entre las evaluadas para reconocer los número de los dorsales.

5.1. PaddleOCR

Es una herramienta de código abierto desarrollada por Baidu, capaz de reconocer caracteres en distintos alfabetos según los parámetros configurados. Para la detección de texto, emplea modelos como DB (Differentiable Binarization), mientras que para el reconocimiento de caracteres utiliza modelos como CRNN (Convolutional Recurrent Neural Network), entre otros, dependiendo de la versión y configuración del sistema OCR [64].

Este sistema devuelve como resultado las coordenadas del texto detectado, el contenido reconocido y un índice de confianza (score) que indica la probabilidad de que el reconocimiento sea correcto. Además, dispone de una API en Python, lo que facilita su integración en distintos proyectos.

En este trabajo se utilizará la versión 2.10, que es la última disponible en el momento del desarrollo del TFG.

5.1.1. Uso de PaddleOCR

Antes de nada hay que instalarlo, lo cual puede hacerse de manera sencilla mediante el gestor de paquetes pip [64]. Al finalizar el TFG, se actualizó PaddleOCR a la versión 3.0. Sin embargo, en este trabajo se utilizó la versión anterior, ya que la nueva no ofrecía mejoras significativas en el reconocimiento de los números de los dorsales, como se explicara en el siguiente apartado. Una vez instalada, se debe importar el módulo correspondiente en el script, configurar los parámetros deseados para inicializar el OCR y, posteriormente, pasarle las imágenes sobre las que se desea realizar la predicción del texto.

En nuestro caso, se usarán dos OCR de paddleOCR entrenados en dos idiomas (chino e inglés) y se escogerá la salida que más confianza tenga. Esta selección de dos OCR se explicará en más detalle, pero resumidamente es por ser el método que mejor funciona reconociendo los números de los dorsales.

5.1.2. Resultados de PaddleOCR sobre dorsales etiquetados

Para obtenerlos, se proporcionaron a PaddleOCR los rectángulos correspondientes a los dorsales etiquetados, donde se encuentra el número que debe reconocerse y se compara su salida con el número real. En el anexo A se podrá ver cómo se usa este OCR para obtener los resultados sobre los dorsales etiquetados.

En este trabajo se utilizaron dos versiones de PaddleOCR: una entrenada en inglés y otra en chino, ambas con conjuntos de datos diferentes, pero ambas con la capacidad de reconocer números. De las dos predicciones generadas, se selecciona aquella con el mayor nivel de confianza. Con este método, se obtuvierón los siguientes resultados:

Datos	Número de dorsales	Porcentaje acierto
Set1	108	99,07 %
Set2	94	97,87 %
Set3	153	98,04 %
Set4	323	91,64 %
Set5	258	98,06 %
Set6	170	88,82 %
Set7	246	96,75 %
Set8	77	92,21 %
Set9	89	97,75 %
Total	1.518	95,19 %

Tabla 5.1: Resultados mayor confianza de los OCRs en inglés y chino

Las tasas de acierto usando dos OCR son significativamente mejores que usándolos individualmente. Para evaluar esta diferencia, se utilizó el test de McNemar, cuya hipótesis nula establece que no existen diferencias significativas en el rendimiento de los OCR comparados. Siendo n_{01} el número de casos en los que el primer método falla y el segundo acierta, y n_{10} el número de casos en los que ocurre lo contrario. El estadístico de este test se calcula mediante la siguiente fórmula:

$$\chi^2 \approx \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

Al usar únicamente el OCR en chino, se obtuvo un porcentaje de acierto del 91.37 % en el conjunto de dorsales. Comparado con el OCR de dos idiomas (OCR chino + inglés), se registraron 15 errores únicos del método combinado y 71 errores únicos del OCR en chino. Con estos valores, el test de McNemar arroja un p valor de $9,90 * 10^{-10}$, lo que permite rechazar la hipótesis nula. En el caso del OCR en inglés, se obtuvo un porcentaje de acierto del 92,03 %. Comparado nuevamente con los dos OCR, se observaron 17 errores únicos en este y 64 errores únicos del OCR en inglés. El test de McNemar en este caso produce un p-valor de $1,77 * 10^{-7}$, permitiendo también rechazar la hipótesis nula.

También se evaluó el reconocimiento de los números de los dorsales utilizando el mismo método basado en dos OCR, pero con la nueva versión de PaddleOCR. Esta versión alcanzó una tasa de acierto del 95,92 % para todos los dorsales, superando ligeramente a la versión anterior (95,19 %). La versión 2.10 cometió 39 errores que no se presentaban con la nueva versión, mientras que esta última tuvo 27 errores que sí fueron correctamente reconocidos por la versión anterior. Al aplicar el test de McNemar, se obtuvo un p-valor de 0,1397, por lo que no se puede rechaza con demasiada certaza que los métodos no tengan un funciomiento equivalente. Con este resultado decidimos utilizar la versión 2.10 por ser más rápida y haber sido la utilizada en el desarrollo del proyecto.

En cuanto a las tasas de acierto de otros sistemas OCR, sus resultados fueron tan deficientes que no resulta relevante compararlos en detalle. Por ejemplo, evaluados sobre el conjunto completo de dorsales, EasyOCR alcanzó una tasa de acierto del 42,10 %, mientras que pytesseract obtuvo un 32,67 %. Aunque se probaron distintos métodos para mejorar estos porcentajes, las mejoras fueron insuficientes y las tasas de acierto quedaron muy por debajo de las obtenidas con PaddleOCR.

5.1.3. Dorsales no reconocidos

Un total de 73 de los 1.518 presentaron errores de reconocimiento. Esto se debe a diversos factores, como que algunos dorsales son particularmente difíciles de interpretar y el OCR comete errores en uno/s dígito/s lo que hace que el número predicho no coincida con el número real. Además, hay unos pocos dorsales que visualmente son fáciles de reconocer, pero el OCR no lo hace correctamente por distintas razones. A continuación, se presentan ejemplos de este tipo de errores:

- **Dorsales girados en el plano.** Hay 3 de ellos que no son complicados de reconocer, pero que el OCR no lo hace correctamente. Aunque PaddleOCR dispone de parámetros que permiten corregir automáticamente la orientación del texto, se ha optado por no utilizarlos, ya que hacerlo reduce la tasa de acierto a un 93,01 % (una disminución de dos puntos porcentuales) y aumenta el tiempo de procesamiento del sistema.

Figura 5.1: Ejemplos dorsales girados no reconocidos

- **Dorsales vistos de perfil.** De este tipo hay 6 dorsales que no han sido reconocidos, a pesar de que en principio parecían fácilmente reconocibles.

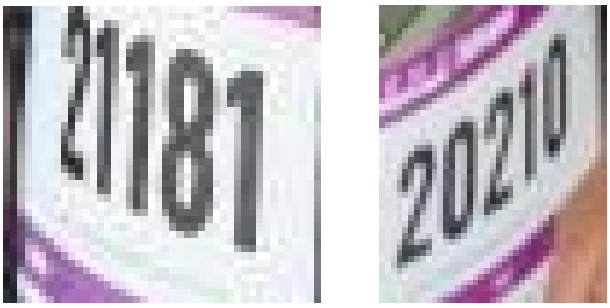

Figura 5.2: Ejemplos dorsales vistos de perfil no reconocidos

5.2. YOLO

Este es el software más importante en la consecución de objetivos de este trabajo, ya que nos permite detectar los dorsales.

5.2.1. Modelos de YOLO usados

Se emplearán versiones de distintos tamaños, con el objetivo de determinar cuál ofrece el mejor desempeño. La hipótesis inicial es que si se cuenta con una cantidad suficiente de datos y no se produce sobreajuste, los modelos de mayor tamaño deberían ofrecer mejores resultados. Para el proyecto se usarán los siguientes modelo: YOLOv11n, YOLOv11m, YOLOv11x, YOLOv12n, YOLOv12m y YOLOv12x.

5.2.2. Como usar los modelo de YOLO

Antes de utilizarlos, es necesario instalar la librería de Ultralytics, lo que se realiza con el siguiente comando: `pip install ultralytics`.

Una vez instalada, es posible emplear modelos preentrenados para detectar objetos en imágenes o videos, o bien entrenar un modelo propio con datos personalizados. Para el entrenamiento, se requieren imágenes, etiquetas en formato YOLO y un archivo de configuración data.yaml que defina las clases y las rutas a los datos. También, la librería permite validar el modelo y realizar predicciones sobre nuevos datos [65].

5.2.3. Resultados entrenamiento YOLO

Este entrenamiento se hace para las distintas carpetas y con los distintos modelos de YOLOv11 y YOLOv12. Aunque se ponga que las épocas de entrenamiento sean 3.000, siempre se para antes por no mejorar más el modelo en el conjunto de validación.

Para obtener estos resultados, se utilizaron 150 o 151 imágenes (según la carpeta), ya que un sexto de 904 no es un número entero. Estas imágenes provienen de las carpetas de validación, generadas mediante estratificación de las imágenes en los distintos grupos, según el modelo evaluado. Con este propósito, se presentarán las métricas de precisión, recall, F1-score, mAP@50, mAP@50-95 y latencia (en milisegundos), todas previamente descritas en el marco teórico.

Modelo	Precision	Recall	F1 score	mAP50	mAP50-95	latencia(ms)
YOLOv11n	0,889	0,896	0,893	0,943	0,588	3,274
YOLOv11m	0,894	0,896	0,895	0.949	0,592	7,924
YOLOv11x	0,896	0,892	0,894	0,947	0,588	13,545
YOLOv12n	0,820	0,908	0,862	0,919	0,561	3,423
YOLOv12m	0,826	0,877	0,850	0,921	0,559	8,249
YOLOv12x	0,835	0,900	0,866	0,908	0,552	18,552

Tabla 5.2: Tabla resultados validación carpeta 1

Modelo	Precision	Recall	F1 score	mAP50	mAP50-95	latencia(ms)
YOLOv11n	0,894	0,953	0,922	0,970	0,613	2,843
YOLOv11m	0,888	0,926	0,906	0,968	0,618	6,272
YOLOv11x	0,849	0,977	0,908	0,965	0,595	13,367
YOLOv12n	0,903	0,913	0,908	0,970	0,599	3,400
YOLOv12m	0,890	0,941	0,915	0,972	0,609	8,091
YOLOv12x	0,898	0,922	0,910	0,970	0,609	19,835

Tabla 5.3: Tabla resultados validación carpeta 2

Modelo	Precision	Recall	F1 score	mAP50	mAP50-95	latencia(ms)
YOLOv11n	0,899	0,898	0,898	0,961	0,601	3,003
YOLOv11m	0,905	0,909	0,907	0,959	0,603	6,432
YOLOv11x	0,842	0,947	0,892	0,960	0,604	13,312
YOLOv12n	0,876	0,918	0,897	0,961	0,601	3,412
YOLOv12m	0,891	0,934	0,912	0,955	0,594	8,529
YOLOv12x	0,870	0,870	0,870	0,944	0,593	19,478

Tabla 5.4: Tabla resultados validación carpeta 3

A partir de los resultados de la validación, podemos identificar los modelos que ofrecen un mejor desempeño y cuáles son más rápidos. En general, los valores de F1-score son similares entre los distintos modelos, por lo que, inicialmente, parece razonable optar por aquellos de menor tamaño debido a su mayor velocidad. No obstante, es importante tener en cuenta que estos resultados no son definitivos, ya que el conjunto de validación puede estar sesgado por ser el responsable de detener el entrenamiento de forma anticipada. Por ello, debemos usar los resultados del entrenamiento que se verán en el siguiente capítulo.

Capítulo 6

Resultados

Para determinar cuál de los modelos ofrece un mejor rendimiento, es necesario evaluar su comportamiento en los distintos conjuntos de prueba no solo en las tareas de detección, sino también en el reconocimiento de los dorsales. En este capítulo se verán los valores de las métricas que definirán el rendimiento de nuestro sistema de reconocimiento de dorsales. Así mismo, se analizará cómo varían estas métricas según el tipo de dorsal.

6.1. Resultados test YOLO+OCR

Para obtener estos resultados, hemos aplicado un código que será incluido y explicado en detalle en el anexo A. En este apartado, presentaremos únicamente los aspectos más relevantes del proceso.

Para aplicar nuestro sistema de detección de dorsales, comenzamos prediciendo la ubicación de los números utilizando distintos modelos YOLO. Se seleccionan aquellas bounding boxes cuya confianza supere un umbral mínimo de 0,1. Si bien este umbral bajo puede incluir predicciones incorrectas, este inconveniente se mitiga en una etapa posterior mediante el uso de OCR. En esta segunda fase, se conservan únicamente las bounding boxes en las que el OCR detecte números con una confianza superior a 0,7; de lo contrario, el dorsal es descartado. Esta estrategia permite que YOLO sea más permisivo en la detección inicial, mientras que el OCR actúa como filtro, eliminando aquellas detecciones que no presenten un contenido numérico con un nivel de certeza razonable. Estos últimos parámetros, se han escogido probando con varios valores de los mismos para mejorar los resultados.

Una bounding box predicha se considera que se corresponde con una real, cuando la distancia entre los centros de ambas es pequeña. Concretamente, se acepta la predicción como válida si la diferencia en las coordenadas X e Y es inferior a 10 píxeles.

Para los resultados, se presentarán las métricas de precisión, recall y F1-score correspondientes a la detección de dorsales mediante YOLO. Posteriormente, aplicaremos esas mismas métricas al reconocimiento de los dorsales detectados utilizando paddleOCR. Con este enfoque, se obtienen los siguientes resultados sobre los conjuntos de prueba previamente estratificados conteniendo cada uno 151/150 imágenes.

Modelo	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
YOLOv11n	0,811	0,964	0,881	0,752	0,894	0,817
YOLOv11m	0,801	0,965	0,875	0,729	0,878	0,796
YOLOv11x	0,808	0,960	0,878	0,762	0,906	0,827
YOLOv12n	0,805	0,957	0,874	0,738	0,877	0,802
YOLOv12m	0,806	0,933	0,865	0,748	0,866	0,803
YOLOv12x	0,810	0,937	0,869	0,762	0,882	0,818

Tabla 6.1: Tabla resultados test carpeta 1

Modelo	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
YOLOv11n	0,857	0,964	0,907	0,782	0,880	0,828
YOLOv11m	0,871	0,929	0,895	0,810	0,856	0,832
YOLOv11x	0,826	0,972	0,893	0,758	0,892	0,819
YOLOv12n	0,827	0,976	0,896	0,759	0,896	0,821
YOLOv12m	0,836	0,980	0,902	0,767	0,900	0,828
YOLOv12x	0,834	0,988	0,904	0,766	0,908	0,831

Tabla 6.2: Tabla resultados test carpeta 2

Modelo	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
YOLOv11n	0,856	0,971	0,910	0,820	0,931	0,872
YOLOv11m	0,881	0,967	0,922	0,818	0,898	0,856
YOLOv11x	0,850	0,971	0,907	0,800	0,914	0,853
YOLOv12n	0,851	0,980	0,911	0,784	0,902	0,839
YOLOv12m	0,862	0,967	0,912	0,803	0,902	0,850
YOLOv12x	0,839	0,959	0,895	0,789	0,902	0,842

Tabla 6.3: Tabla resultados test carpeta 3

De las métricas analizadas, cabe destacar que la precisión es relativamente bajo en muchos casos en comparación con el recall. Esto se debe a la presencia de dorsales difíciles de reconocer en las imágenes que han sido etiquetadas, pero no han sido detectados por los distintos modelos YOLO. Estos falsos positivos serán analizados en detalle utilizando el modelo que se considere mejor. Para determinar cuál es ese modelo, se compararán los valores de F1-score obtenidos por cada uno de los modelos evaluados.

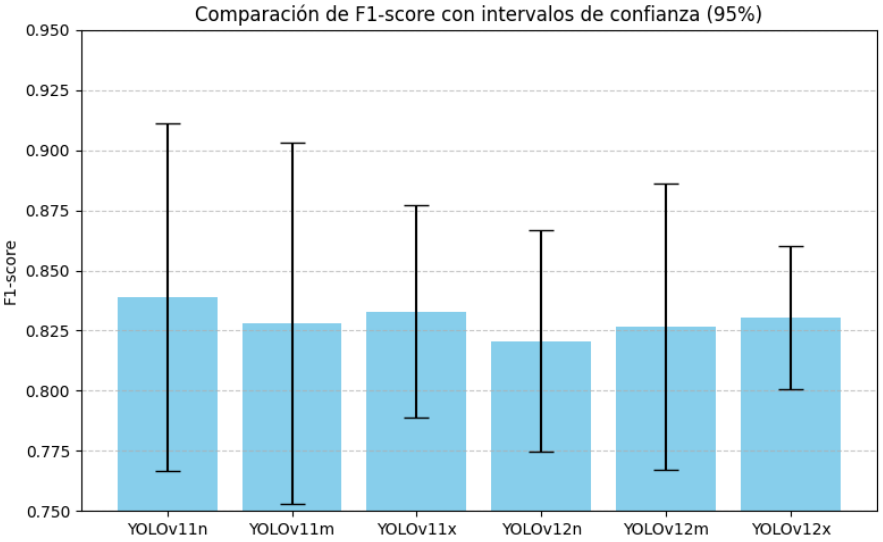

Figura 6.1: F1 score media en las carpetas con IC 95 % en los distintos modelos.

En el gráfico anterior, se observa que el F1-score de los distintos modelos es similar, aunque los modelos YOLOv11n, YOLOv12x y YOLOv11x presentan valores ligeramente superiores. Se selecciona YOLOv11n por tener uno de los F1-score más altos y tratarse de un modelo pequeño.

6.2. Comparación resultados

En esta sección, se compararán los resultados obtenidos en la validación y en la de test aplicando YOLO con y sin umbrales de confianza en el OCR. El objetivo es analizar la relevancia de utilizar un conjunto de prueba distinto al de validación, dado que este último influye indirectamente en el entrenamiento. Asimismo, se busca conocer cuál es la mejora en los modelos al añadir estos umbrales de confianza.

Para ello, se compararon los valores F1-score para la detección de los dorsales en tres escenarios:

- Sobre los conjuntos de validación.
- Realizada en los conjuntos de prueba, excluyendo los dorsales que no fueron reconocidos por el OCR con una confianza mínima de 0,7 y considerando únicamente detecciones con una confianza mayor o igual a 0,1.
- Sobre los conjuntos de prueba sin aplicar ningún umbral de confianza

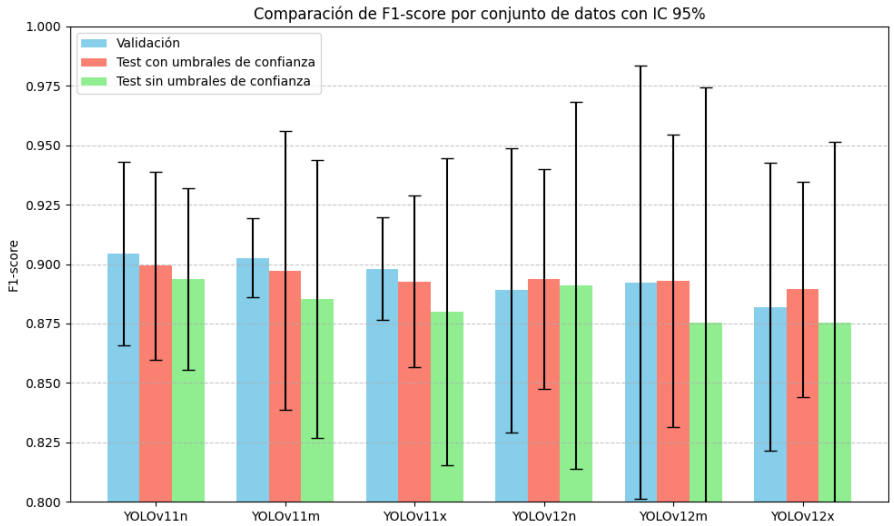

Figura 6.2: Media e IC de los F1-scores de la detección para los distintos conjuntos y carpetas

Se observa que los resultados obtenidos en el conjunto de validación, son superiores en una buena parte de los casos, lo cual es esperable dada su vinculación indirecta con el entrenamiento. Igualmente, se aprecia una mejora en el F1-score de la detección al aplicar umbrales de confianza tanto en YOLO como en el OCR. Esta mejora se debe a que, al descartar aquellos casos en los que no se ha identificado correctamente el número del dorsal, se evita realizar predicciones incorrectas, lo que contribuye a un aumento en la precisión del sistema.

Presentamos los resultados de los test sin umbrales de confianza en la detección y reconocimiento:

Modelo	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
YOLOv11n	0,816	0,945	0,876	0,751	0,870	0,807
YOLOv11m	0,786	0,957	0,863	0,712	0,866	0,781
YOLOv11x	0,779	0,995	0,854	0,731	0,854	0,801
YOLOv12n	0,830	0,902	0,864	0,757	0,823	0,789
YOLOv12m	0,790	0,874	0,830	0,733	0,811	0,770
YOLOv12x	0,783	0,909	0,841	0,732	0,850	0,787

Tabla 6.4: Tabla resultados test carpeta 1 sin umbrales de confianza

Modelo	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
YOLOv11n	0,851	0,964	0,904	0,773	0,876	0,821
YOLOv11m	0,883	0,883	0,883	0,815	0,815	0,815
YOLOv11x	0,804	0,972	0,880	0,731	0,884	0,800
YOLOv12n	0,805	0,980	0,884	0,736	0,896	0,808
YOLOv12m	0,835	0,956	0,891	0,761	0,871	0,813
YOLOv12x	0,801	0,988	0,885	0,733	0,904	0,809

Tabla 6.5: Tabla resultados test carpeta 2 sin umbrales de confianza

Modelo	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
YOLOv11n	0,857	0,951	0,901	0,816	0,906	0,859
YOLOv11m	0,875	0,947	0,910	0,804	0,869	0,835
YOLOv11x	0,856	0,967	0,906	0,799	0,906	0,849
YOLOv12n	0,873	0,984	0,925	0,801	0,902	0,848
YOLOv12m	0,845	0,976	0,905	0,784	0,906	0,841
YOLOv12x	0,851	0,955	0,900	0,800	0,898	0,846

Tabla 6.6: Tabla resultados test carpeta 3 sin umbrales de confianza

6.3. Resultados del modelo seleccionado (YOLOV11n)

Comenzaremos analizando su comportamiento sobre los diferentes grupos de imágenes, los cuales son clasificados según el número de dorsales y su formato. Denotaremos estos grupos de la siguiente manera:

- `no_mar`: si el margen alrededor del dorsal es pequeño.
- `mar`: si hay margen arriba y abajo.
- `xmar`: si el margen es considerable.

En cuanto al número de dorsales, utilizaremos las siguientes etiquetas:

- `1`: para imágenes con un solo dorsal.
- `2`: para aquellas con dos dorsales.
- `3+`: para imágenes con tres o más dorsales.

Grupos imágenes	Nºdorsales	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
1 no_mar	19	0.95	1	0.974	0.95	1	0.974
1 mar	40	0,826	0,950	0,884	0,826	0,950	0,884
1 xmar	27	0,711	1	0,831	0,658	0,926	0,769
2 no_mar	6	1	0.833	0.909	1	0.833	0.909
2 mar	40	0,850	0,756	0,800	0,725	0,644	0,682
2 xmar	40	0,745	0,950	0,835	0,608	0,775	0,681
3+ no_mar	35	0,916	0,943	0,929	0,889	0,914	0,901
3+ mar	32	0,838	0,969	0,899	0,784	0,906	0,841
3+ xmar	15	0,789	1	0,882	0,684	0,867	0,765

Tabla 6.7: Tabla resultados test YOLOV11n carpeta 1

Grupos imágenes	Nºdorsales	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
1 no_mar	19	0,947	0,947	0,947	0,947	0,947	0,947
1 mar	39	0,755	0,949	0,841	0,735	0,923	0,818
1 xmar	27	0,743	0,963	0,839	0.600	0,778	0,677
2 no_mar	8	0,889	1	0,941	0,889	1	0,941
2 mar	40	0,886	0,975	0,929	0,750	0,825	0,786
2 xmar	40	0,905	0,950	0,927	0,833	0,875	0,854
3+ no_mar	28	0,800	1	0,889	0,771	0,964	0,857
3+ mar	33	1	0,939	0,969	0,871	0,818	0,844
3+ xmar	15	0,938	1	0,968	0,875	0,933	0,903

Tabla 6.8: Tabla resultados test YOLOV11n carpeta 2

Grupos imágenes	Nºdorsales	Precision YOLO	Recall YOLO	F1 score YOLO	Precision OCR	Recall OCR	F1 score OCR
1 no_mar	19	0,950	1	0,974	0,950	1	0,974
1 mar	39	0,750	1	0,857	0,731	0,974	0,835
1 xmar	27	0,889	0,889	0,889	0,889	0,889	0,889
2 no_mar	8	0,889	1	0,941	0,889	1	0,941
2 mar	40	0,867	0,975	0,918	0,800	0,900	0,847
2 xmar	38	0,905	1	0,950	0,810	0,895	0,850
3+ no_mar	28	0,839	0,929	0,881	0,839	0,929	0,881
3+ mar	30	0,883	0,1	0,909	0,778	0,933	0,848
3+ xmar	16	0.938	0,938	0,938	0,938	0,938	0,938

Tabla 6.9: Tabla resultados test YOLOV11n carpeta 3

A partir de las tablas, se observa que si el dorsal tiene un margen pequeño con su número, los resultados son mejores. Esto tiene lógica, ya que cuanto más grande sea el número en relación al dorsal, será más fácil reconocerlo.

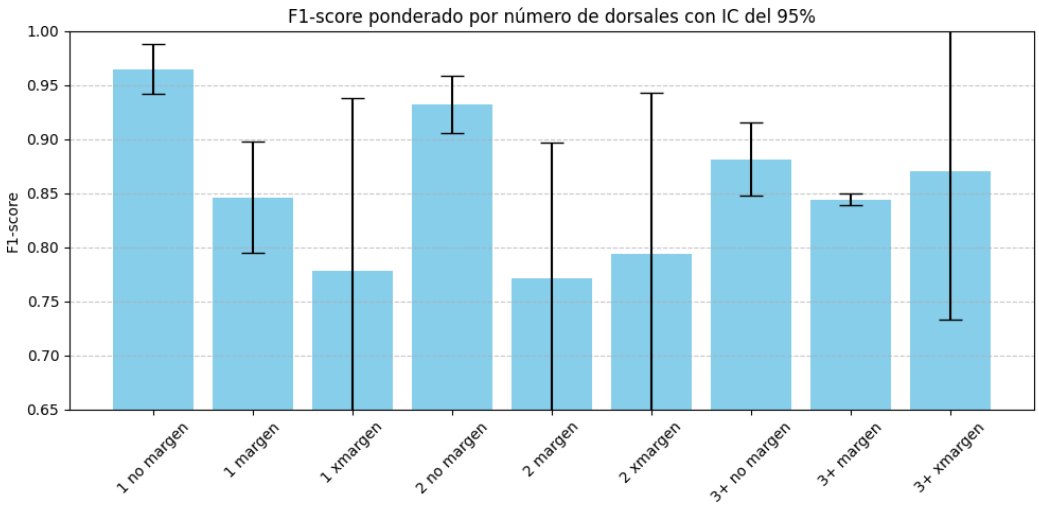

Figura 6.3: F1 score medio en las carpetas con IC en los distintos grupos de imagenes.

En el gráfico anterior, se observa que cuando el dorsal presenta un margen pequeño, los resultados son notablemente mejores. En cuanto al resto de formatos de los márgenes, no queda claro en qué grupo se comporta mejor el modelo. Se esperaría un peor rendimiento en el grupo con márgenes más amplios, pero esto no se da. Por otro lado, respecto al número de dorsales, tampoco se observa un patrón relevante.

6.3.1. Dorsales predichos erróneamente

En las tablas anteriores apartado 6.3, se observa que en casi todos los casos la precisión (proporción de dorsales predichos correctamente respecto al total de dorsales predichos) es inferior al recall (proporción de dorsales predichos correctamente respecto al total de dorsales etiquetados). Como consecuencia, el F1-score se ve influido negativamente por los valores más bajos de la precisión. Este comportamiento se debe a la detección y reconocimiento de varios dorsales que no fueron previamente etiquetados. Para visualizar las diferencias entre la salida predichos y la real, se utilizan los colores morado y verde, respectivamente. Las causas de estas discrepancias se verán a continuación:

- **Dorsales parcialmente tapados**. En algunos de estos, sólo se ve un trozo del número, razón por la cual no han sido etiquetados, aunque alguno de ellos han sido reconocidos y probalemete hallan acertado el número del dorsal. La siguiente imagen es un ejemplo de esto:

Figura 6.4: Ejemplo dorsal parcialmente tapado.

- **Dorsales difíciles no etiquetados.** Esta última operación no se ha realizado debido a la dificultad para ver el número, si bien alguno de estos, y en algún caso probablemente hallan sido reconocidos correctmente. A continuación, se muestra un ejemplo:

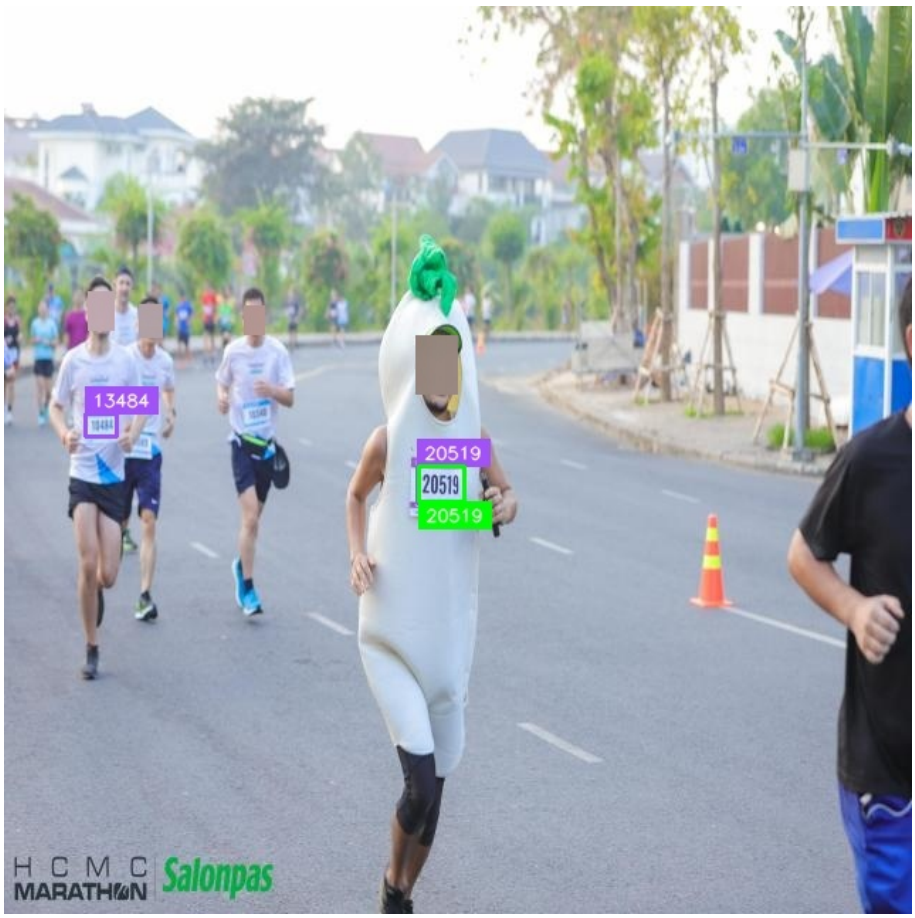

Figura 6.5: Ejemplo dorsal borroso díficilmente identificable.

No sólo se dan estos errores, sino que también, en menor medida, el OCR falla al intentar reconocer los números de YOLO detectados correctamente; a veces, YOLO no detecta los dorsales que han sido etiquetados, reduciendo en ambos casos el recall.

6.3.2. Posibles mejoras en carreras concretas

Cabe destacar que algunos errores del OCR se deben a la detección de números, que no tienen sentido dentro del contexto específico de la carrera. Por ejemplo, puede identificar dorsales de tres cifras en eventos donde todos los participantes utilizan dorsales de cuatro cifras. Aimismo, puede detectar números con una relación de aspecto que no se corresponde a los dorsales de la carrera o reconocer erróneamente alguno que no figure en la lista oficial de inscritos.

Estos fallos pueden reducirse mejorando la precisión del sistema, si se eliminasen los dorsales que no se ajustan a los criterios esperados. Para lograrlo, es fundamental contar con información específica sobre el formato de los dorsales. Incorporar este conocimiento puede mejorar el rendimiento del modelo.

Capítulo 7

Aplicación web

Este capítulo se centrará en el desarrollo de un prototipo de aplicación web intuitiva y sencilla. Su objetivo principal es proporcionar una herramienta visual que facilite la verificación del funcionamiento del reconocedor de dorsales. Para ello, la aplicación deberá permitir la incorporación de nuevas imágenes al sistema, la visualización de las anotaciones generadas sobre estas y la búsqueda de imágnes por número de dorsal.

Para obtener el modelo que se utilizará en la aplicación web, se vuelve a entrenar YOLOv11n utilizando los datos de la carpeta 1, incorporando el conjunto de prueba al de entrenamiento. Los pesos resultantes se emplean como base para el reconocimiento de dorsales en la aplicación web. Este se utilizará junto con PaddleOCR para construir el sistema de reconocimiento de dorsales, tal como se explicó, en el desarrollo del modelo.

En este capítulo, presentaremos su análisis y diseño de la misma, aplicando los conocimientos aprendidos durante las distintas asignaturas relacionadas con el desarrollo de software de la carrera de Informática en la mención de Computación.

7.1. Análisis

El objetivo principal es identificar y definir con claridad los requisitos funcionales, no funcionales y de información del sistema. Así mismo, se busca comprender en profundidad las necesidades del usuario final y el contexto en el que se utilizará la aplicación. Este análisis constituye la base sobre la cual se diseñará e implementará la solución, permitiendo minimizar errores durante las siguientes fases de la creacción de la aplicación.

7.1.1. Análisis de requisitos

Se presentarán los distintos requisitos que debe cumplir la aplicación para satisfacer las necesidades, funciones y condiciones necesarias para alcanzar su objetivo principal, que consiste en comprobar el funcionamiento del reconocedor de dorsales en distintos conjuntos de imágenes.

Requisitos funcionales

Identificador	Nombre	Descripción
RF-1	Carga de imágenes	El sistema debe permitir que se puedan cargar imágenes para que puedan ser enviadas al modelo.
RF-2	Cancelación carga de imágenes	El sistema debe consentir la cancelación del carga de las imágenes.
RF-3	Enviar imágenes	El sistema debe permitir que se puedan enviar imágenes para que el modelo reconozca los dorsales.
RF-4	Reconocimiento de dorsales	El sistema reconocerá los dorsales de la imagen mediante los modelos propuestos.
RF-5	Búsqueda de imágenes	El sistema debe permitir buscar y mostrar las imágenes que contengan un dorsal en concreto.
RF-6	Mostrar imágenes	El sistema mostrara las imágenes enviadas y predichas por el reconocedor de dorsales.
RF-7	Borrar imágenes	El sistema dejará que se borren imágenes que hayan sido enviadas.

Tabla 7.1: Requisitos funcionales

Requisitos no funcionales

Identificador	Nombre	Descripción
RNF-1	Interfaz simple	El sistema debe permitir que para realizar una tarea no halla que hacer más de tres acciones.
RNF-2	Formato de imagen	El sistema dejará que las imágenes subidas puedan tener los formatos .jpg,.jpeg y .png.
RNF-3	Tiempo de procesamiento del modelo	El sistema informará de que el proceso de procesamiento de imágenes lleva tiempo.
RNF-4	Compatibilidad con Python	El sistema deberá ser capaz de ejecutar el modelo de reconocimiento de dorsales codificado en Python.
RNF-5	Portabilidad	El sistema debe ser ejecutable en cualquier sistema que soporte Docker.

Tabla 7.2: Requisitos no funcionales

Requisitos de información

Identificador	Nombre	Descripción
RDI-1	Imágenes enviadas	El sistema guardará las imágenes enviadas.
RDI-2	Imágenes anotadas	El sistema almacenará las imágenes anotadas por el reconocedor de dorsales, generadas a partir de las imágenes enviadas.
RDI-3	Números de dorsales	El sistema guardaŕa los números de los dorsales detectados en las imágenes enviadas, junto con la imagen correspondiente a cada número de dorsal.

Tabla 7.3: Requisitos de información

7.1.2. Casos de uso

A partir de estos, expresaremos los requisitos funcionales. Estos se encuentran en el siguiente diagrama de casos de uso.

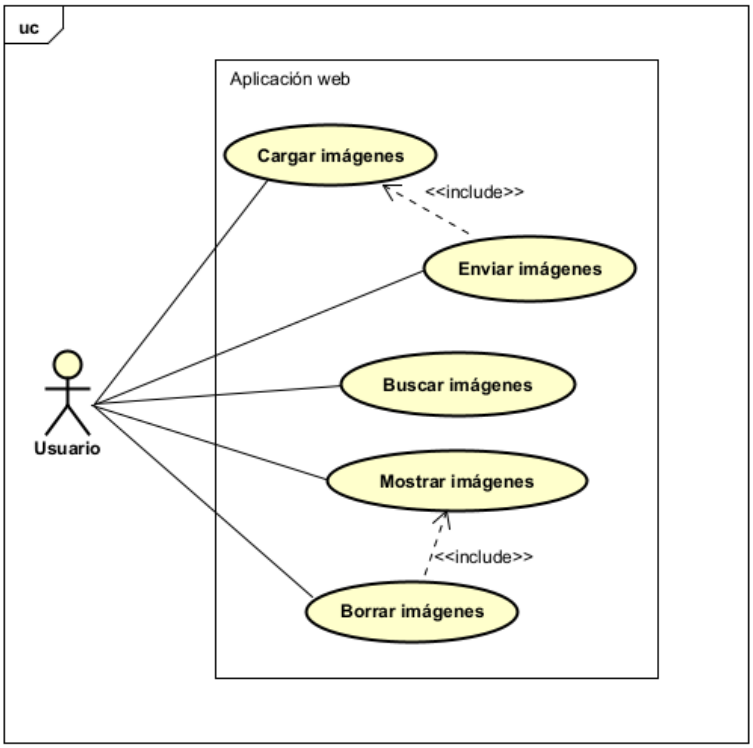

Figura 7.1: Diagrama de casos de uso

Descripciones casos de uso

Identificador	Cargar imágenes
Actor	Usuario
Descripción	Se carga una o más imágenes en el sistema y se comprueba que el formato es correcto
Precondiciones	El usuario está en la parte de subida de imágenes.
Postcondiciones	El sistema tiene las imágenes cargadas preparadas para ser enviadas al modelo.
Secuencia normal	1. El usuario sube una o más imágenes al sistema. 2. El sistema comprueba el formato de las imágenes. 3. El sistema muestra que imágenes han sido cargadas.
Secuencia alternativa	2.a El sistema detecta que el formato no es compatible lo informa al usuario y vuelve al paso 1. 3.a El usuario cancela la carga de imágenes y se vuelve al paso 1.

Tabla 7.4: Descripción del CU-1: Cargar imágenes

Identificador	Enviar imágenes
Actor	Usuario
Descripción	El usuario envia las imágenes, reconoce los dorsales y guarda tanto las imágenes como los números de los dorsales, asociando imagen con su/s número/s.
Precondiciones	Hay una o más imágenes cargadas.
Postcondiciones	Se guarda la imagen con los números de dorsales reconocidos por el modelo.
Secuencia normal	1. El usuario solicita el envió de imágenes al sistema. 2. El sistema comprueba que hay imágenes cargadas. 3. El sistema almacena las imágenes. 4. El sistema detecta los dorsales en las imágenes e informa sobre el progreso de este proceso. 5. El sistema guarda los números de los dorsales de las imágenes.
Secuencia alternativa	2.a No hay imágenes cargadas y se cancela el caso de uso.

Tabla 7.5: Descripción del CU-2: Enviar imágenes

Identificador	Buscar imágenes
Actor	Usuario
Descripción	Se buscan y muestran las imágenes que contienen un dorsal con un número concreto.
Precondiciones	El usuario ha seleccionado que número de dorsal quiere buscar.
Postcondiciones	El sistema muestra las imágenes que han sido enviadas y contienen ese número de dorsal.
Secuencia normal	1. El usuario solicita la búsqueda de un número de dorsal. 2. El sistema comprueba que se ha enviado al menos una imagen. 3. El sistema busca imágenes con ese número de dorsal. 4. El sistema muestra las imágenes con ese número de dorsal.
Secuencia alternativa	2.a El sistema informa de que no han sido enviada ninguna imagen y se vuelve al paso 1. 4.a El sistema informa que no ha encontrado imágenes con ese número de dorsal y se vuelve al paso 1.

Tabla 7.6: Descripción del CU-3: Buscar imágenes

Identificador	Mostrar imágenes
Actor	Usuario
Descripción	Se muestran las imágenes enviadas al sistema con sus prediciones.
Precondiciones	Ninguna.
Postcondiciones	El sistema muestra las imágenes que se han enviado anotadas.
Secuencia normal	1. El usuario pide ver las imágenes. 2. El sistema muestra las imágenes que han sido enviadas y ha almacenado.

Tabla 7.7: Descripción del CU-4: Mostrar imágenes

Identificador	Borrar imágenes
Actor	Usuario
Descripción	Se seleccionan que imágenes se quieren borrar y se borran del sistema.
Precondiciones	El sistema muestra las imágenes.
Postcondiciones	El sistema borra las imágenes seleccionadas.
Secuencia normal	1. El usuario selecciona las imágenes que quiere borrar y acepta que se borren. 2. El sistema elimina las imágenes y sus anotaciones.

Tabla 7.8: Descripción del CU-5: Borrar imágenes

7.1.3. Modelo de dominio

En este apartado se muestra cómo se estructura y organiza la información que se desea almacenar, de acuerdo con los requisitos definidos. Se incluyén las imágenes utilizadas, tanto las enviadas al reconocedor de dorsales como aquellas anotadas por este, con el fin de visualizar su comportamiento sobre cada una. Por ello, es fundamental que cada imagen tenga asociados los números de dorsal correspondientes, que serán utilizados en las diferentes búsquedas de imágenes.

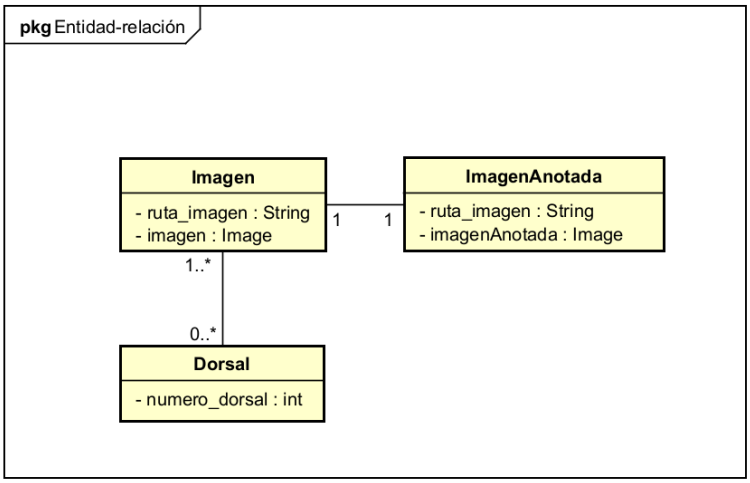

Figura 7.2: Modelo de dominio

7.2. Diseño

Para esta parte, se ha decidido utilizar el microframework Flask, debido a su integración nativa con Python, lenguaje en el que ha sido desarrollado el modelo de reconocimiento de dorsales. Esta elección se justifica por la necesidad de contar con una herramienta compatible con dicho lenguaje de programación (RNF-6), que precisamente facilite la implementación de una aplicación web capaz de satisfacer los requisitos funcionales y no funcionales del sistema.

Para el empaquetado de la aplicación, se opta por utilizar Docker (RNF-7), ya que permite ejecutarla dentro de un contenedor aislado. Esto facilita su despliegue y uso en diferentes máquinas de forma sencilla.

7.2.1. Patrones de diseño

Son soluciones reutilizables a problemas comunes en el desarrollo de software. Su aplicación permite mejorar la estructura, mantenibilidad y escalabilidad del código, al seguir principios de diseño probados y reconocidos. En el contexto de este proyecto, se han adoptado varios patrones con el fin de organizar de forma coherente la arquitectura de la aplicación, facilitar su extensión y mejorar la separación de responsabilidades entre componentes. A continuación, se describen los principales patrones utilizados durante el desarrollo.

Patrón singleton

Se basa en la creación de una única instancia compartida que se utiliza en toda la aplicación. En el caso de la base de datos (db), esta instancia se crea al iniciar la aplicación y todos los módulos acceden a la misma, garantizando un punto único de conexión y configuración.

Patrón MVT (Model-View-Template)

Este es comúnmente usado en frameworks de desarrollo web como Flask o Django. Es similar al MVC [66](Model-View-Controller), pero presenta algunas diferencias que observaremos comparando y analizando brevemente los tres componentes del MVT [67]:

- Model: en ambos, se encarga de los datos y la lógica de negocio. Este suele estructurarse en torno a una base de datos.
- View: se encarga de la lógica de la aplicación y en el MVC es el controlador.
- Template: representa la parte visual de la aplicación, generalmente mediante la renderización de un archivo HTML. En el patrón MVC, esta función corresponde a las vistas

En nuestro caso, el MVT se estructura de la siguiente forma:

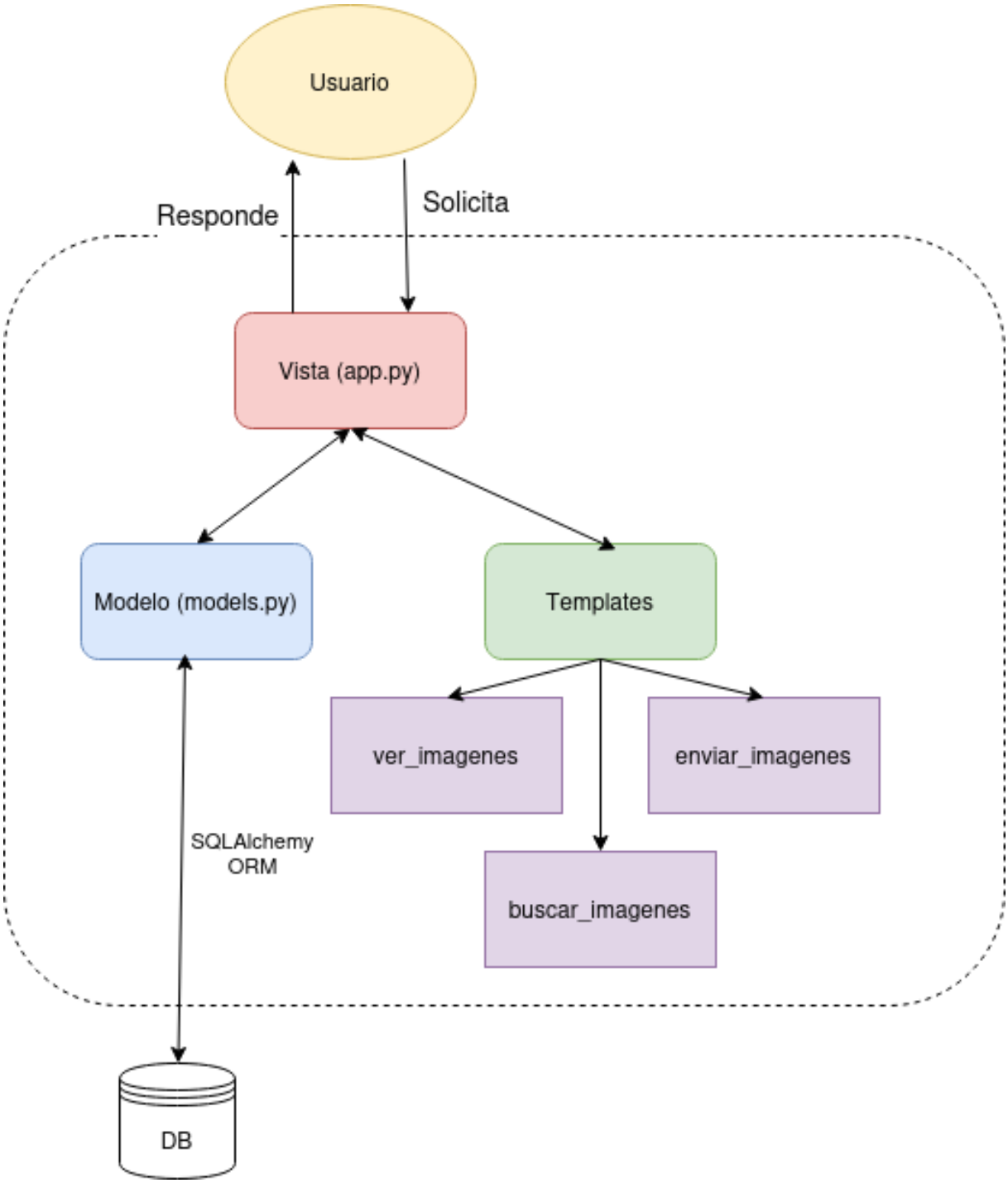

Figura 7.3: MVT

Se añadirán enlaces en los templates para permitir una navegación fluida entre ellas.

7.2.2. Arquitectura

Aquí se detallaran la estructura de nuestra app y su relación entre sus partes. Para ello, se usa el modelo *Uses Style* cuyo diagrama se muestra a continuación:

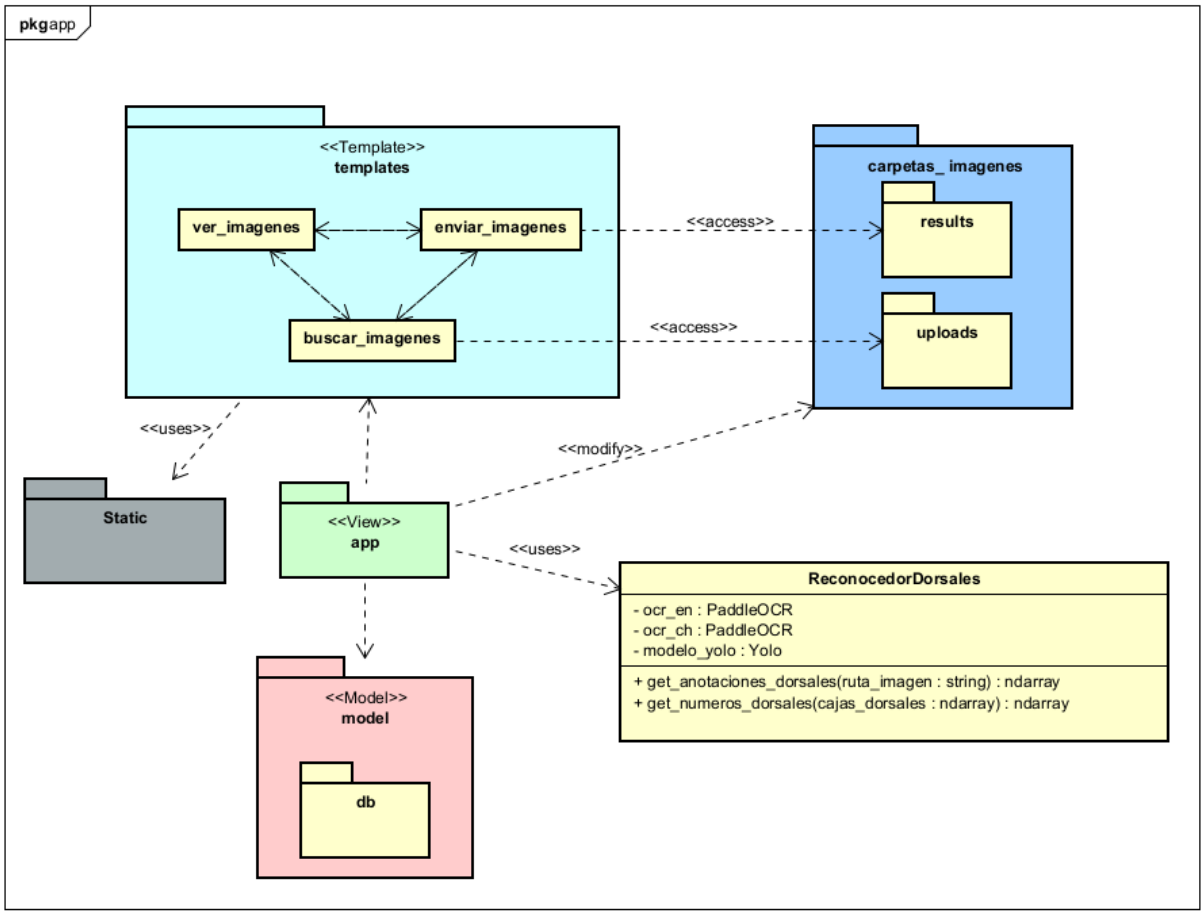

Figura 7.4: Diagrama Uses Style

La estructura de nuestra aplicación está basada en el patrón MVT, como se mostró anteriormente. En las siguientes figuras detallamos más las partes del View y Template de nuestro modelo MVT:

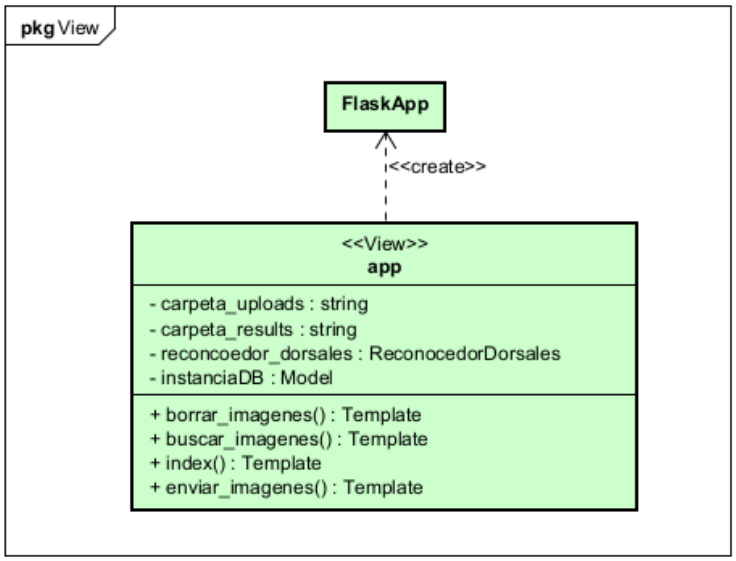

Figura 7.5: Diagrama Uses Style View

Debido a que en el diagrama primer diagrama de la arquitectura se indico las relaciones entre las carpetas de imágenes y las templates, en este se omiten para evitar rebundancias.

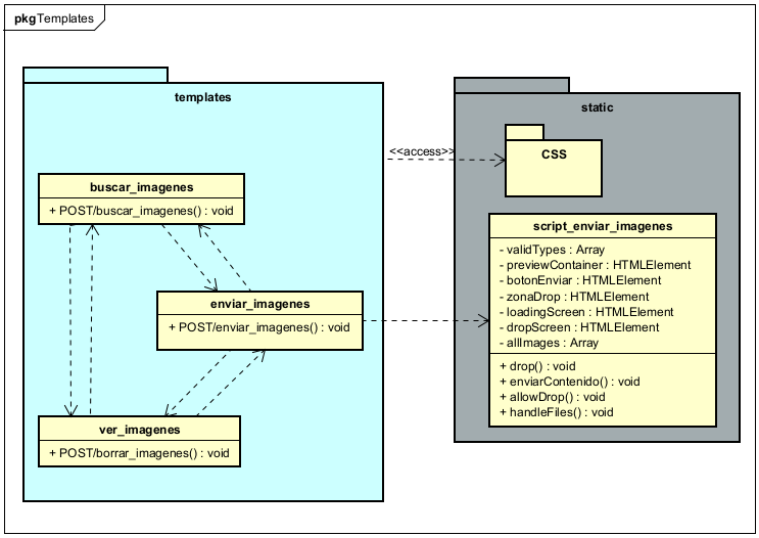

Figura 7.6: Diagrama Uses Style Template

7.2.3. Diagramas de secuencia

A partir de los casos de uso 7.1.2, se utilizarán diagramas de secuencia para modelar los flujos principales y alternativos de las funcionalidades desarrolladas, proporcionando así una visión clara y precisa del funcionamiento interno del sistema.

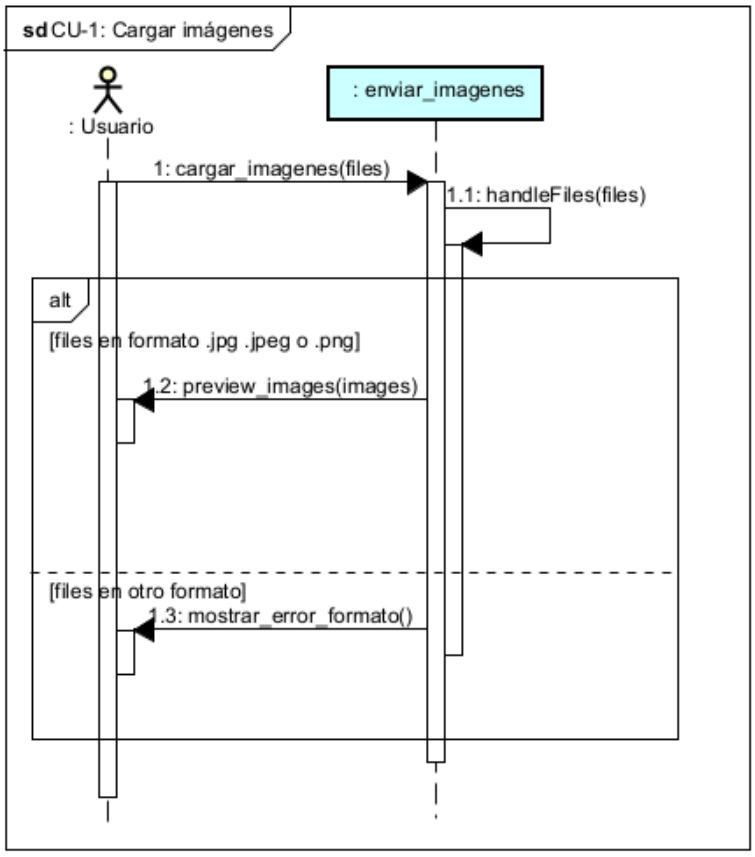

Figura 7.7: Diagrama secuencia del CU-1: Cargar imágenes

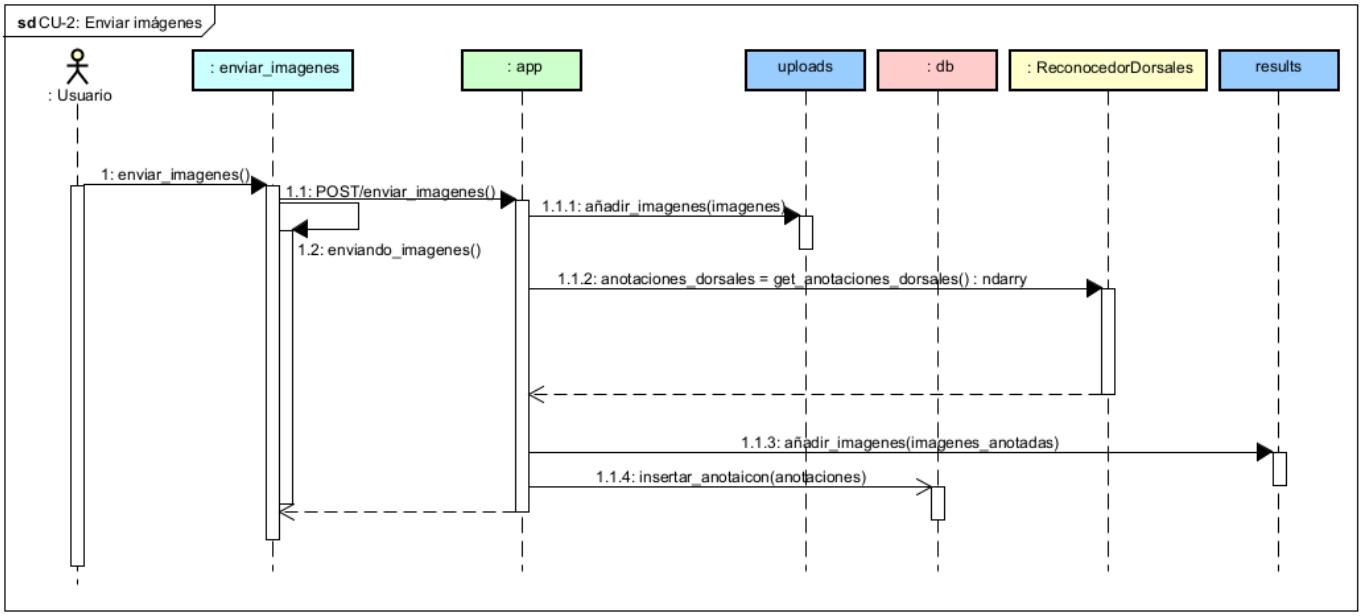

Figura 7.8: Diagrama secuencia del CU-2: Enviar imágenes

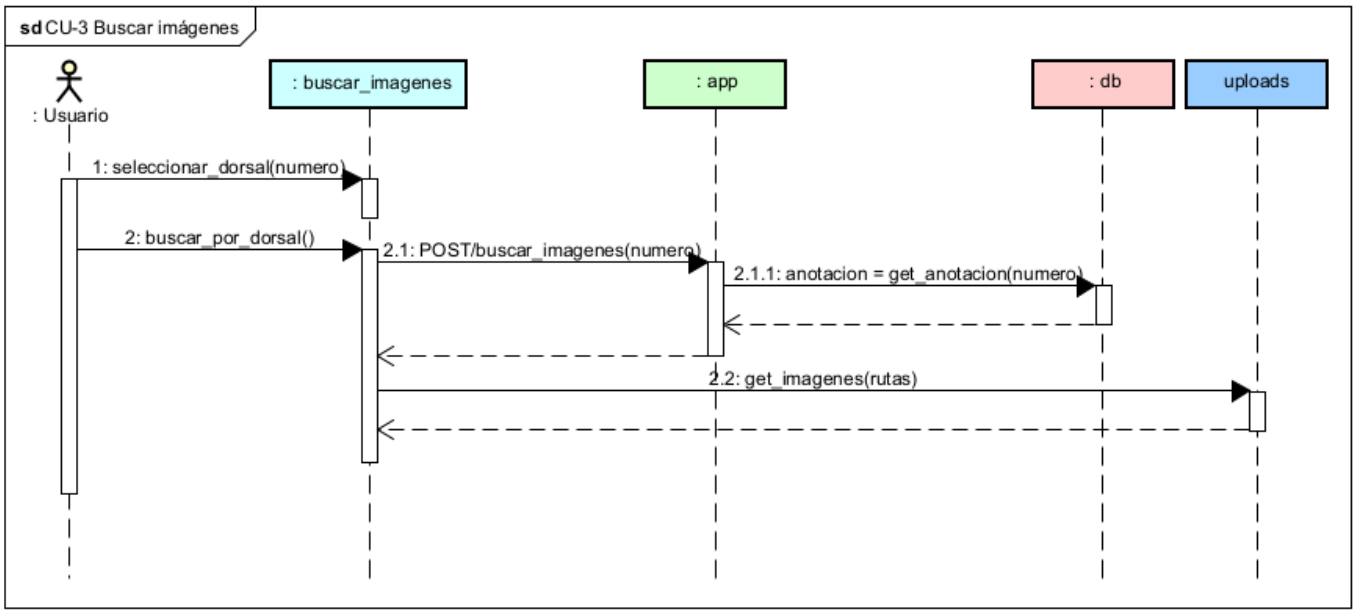

Figura 7.9: Diagrama secuencia del CU-3: Buscar imágenes

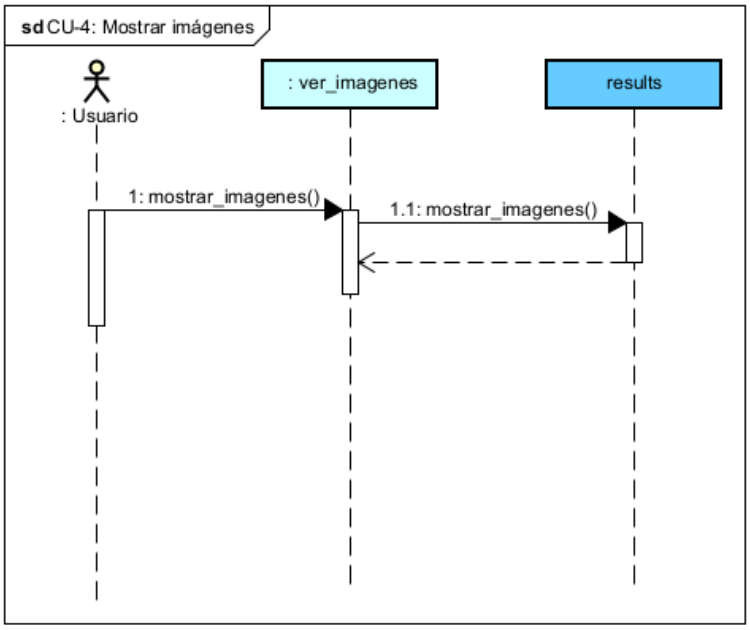

Figura 7.10: Diagrama secuencia del CU-4: Mostrar imágenes

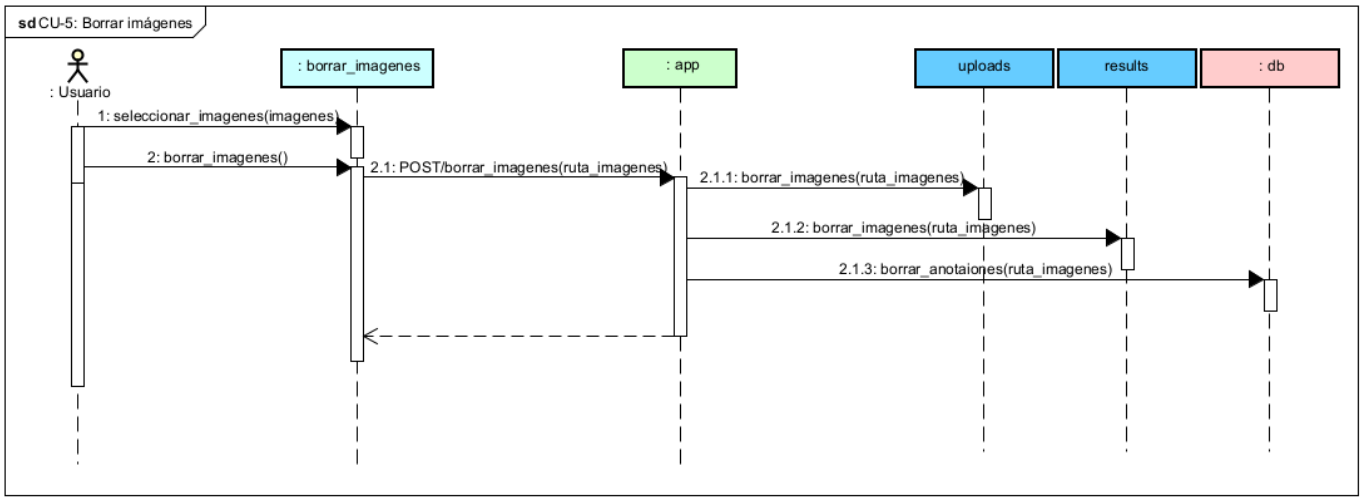

Figura 7.11: Diagrama secuencia del CU-5: Borrar imágenes

7.2.4. Pruebas

Estas son muy útiles para verificar el correcto funcionamiento del sistema. Es muy recomendable tener una batería de pruebas unitarias para comprobar que el funcionamiento base del sistema no se vea alterado por modificaciones en el mismo. Pero, por limitaciones de tiempo, y dado que no se contempla seguir avanzando en la app, ya que con esta ya se cumple sus objetivos, no se crea está bateria. Aún así, sí que es necesario probar el correcto funcionamiento del sistema como se muestra en el anexo B.

7.2.5. Seguridad

Esto es realmente importante para prevenir problemas graves. Sin embargo, dado que la aplicación no se desplegará en la web, su relevancia disminuye. Aún con todo, se implementan las siguientes medidas de seguridad: prevención de inyección SQL y verificación del formato de los archivos que se suben al sistema.

7.2.6. Acceso a la aplicación

Esto se podra hacer con Docker [68] y el repositorio de github en el que se ha subido la app [69], este proceso se explica en el anexo B. Con esto se consigue de facilitar el uso libre de la aplicación por otros usuarios.

Capítulo 8

Conclusiones

En este último capítulo, se hará una reflexión acerca del aprendizaje adquirido en este trabajo, el grado de cumplimiento de los objetivos y se presentarán posibles líneas de trabajo futuras.

8.1. Aprendizaje

Dado que el proyecto está vinculado a la Minería de Datos y la Visión Artificial, gran parte del aprendizaje se ha centrado en estas áreas. No obstante, también se han consolidado competencias en otros ámbitos igualmente relevantes, como la programación, la redacción técnica y el desarrollo de aplicaciones web.

Para estructar el aprendizaje adquirido en este proyecto, mecionaremos las asignaturas más relacionadas con el mismo y en qué partes de las mismas se tratan:

- **Fundamentos de Ingeniería de Software.** La redaccioń de requisitos y casos de uso de un sistema están inspirados en los contenidos de esta asignatura.
- **Planificación y Desarrollo de Sistemas Computacionales.** Tanto la planificación, como parte del análisis y el diseño de la aplicación, se basan en conocimientos adquiridos de esta materia que se asientan y se amplian al realizar el proyecto.
- **Diseño y Evaluación de Sistemas Interactivos.** En esta se dan las bases de la programación web en HTML, CSS y JavaScript, usadas en la creacción de la aplicación web. Asimismo, se hace uso de las guias generales empleadas para mejorar la usabilidad, a partir de los contenidos en esta asignatura y en Interación Persona Computadora.
- **Minería de Datos.** En conjunción con Técnicas de Aprendizaje Automático, dan la base que sobre la comparación de modelos, la creación y uso de métodos de aprendizaje computacional.
- **Otras.** Hay espectos relacionados en los que se ha basado este trabajo, como las relacionadas con la programación, cuya lista no se expondrá para no extenderse en demasía.

A parte de todo esto, se han adquirido conocimiento que no han sido tratados en la carrera, entre los cuáles cabe destacar:

- Estudio y aplicación de los algoritmos YOLO y OCR, así como su integración en un sistema funcional.
- Etiquetado manual de imágenes para la creación de conjuntos de datos.
- Empleo del microframework Flask para el desarrollo de aplicaciones web.
- Uso básico de Docker.

8.2. Análisis de la consecución de objetivos

En este apartado se evalúa el grado en que se han alcanzado los objetivos establecidos al inicio del proyecto. Para ello, se revisan los propuestos y se contrastan con los resultados obtenidos.

El principal objetivo consistía en la creación de un sistema capaz de reconocer dorsales en imágenes. Esto se ha logrado con éxito, siendo la mejor solución aquella basada en YOLOv11n combinado con PaddleOCR, la cual alcanza un valor de F1-score de 0,839. Se observó que este resultado podría mejorarse en carreras específicas, si se dispusiera de información previa sobre los números o el formato de los dorsales. Esta información permitiría filtrar las detecciones incorrectas, reduciendo así el número de falsos positivos y mejorando el rendimiento general del sistema.

Como paso previo a la implementación del sistema, fue necesaria una revisión bibliográfica exhaustiva, así como la construcción de un conjunto de datos amplio y diverso. En el capítulo dedicado al marco teórico, se presenta la información recopilada sobre las técnicas utilizadas. Por su parte, en el capítulo correspondiente al conjunto de datos, se detalla su estructura y se muestra su diversidad, lo que ha llevado a una bateria de un total de 904 imágenes y 1.518 dorsales.

Finalmente, con el fin de facilitar la evaluación del sistema con nuevas imágenes, se ha desarrollado una aplicación web sencilla, que permite a cualquier usuario comprobar su funcionamiento de forma práctica.

8.3. Líneas futuras

Tras la realización de este TFG, se abren diversas posibilidades de desarrollo, que permitirían mejorar y ampliar el alcance del proyecto. Entre las más destacadas, se encuentran las siguientes:

- **Mejora del modelo de reconocimiento de dorsales**. Dado que se ha desarrollado un modelo propio y se han presentado sus resultados, estos pueden utilizarse como punto de referencia (benchmark) para comparar con otras propuestas, que busquen mejorar el rendimiento. Esta tarea podría abordarse mediante la ampliación o diversificación del conjunto de datos, o bien explorando otros algoritmos que ofrezcan métricas más favorables.
- **Clasificación de fotografías en álbumes de carreras**. Una vez desarrollado el reconocedor de dorsales, se plantea su aplicación práctica en la clasificación automática de imágenes. Para ello, bastaría con desarrollar un programa específico, que podría basarse en la ya implementado en este trabajo, con lo que se aprovecharia el modelo entrenado para identificar los dorsales presentes en las imágenes.
- **Generación de clasificaciones parciales**. Como se mencionó en la introducción del proyecto, el sistema podría utilizarse para generar clasificaciones parciales, si se dispusiera de la información temporal de las fotografías, siempre que hubieran sido tomadas en el mismo punto del recorrido. Esto permitiría estimar el orden de paso de los corredores y ofrecer información útil para analizar su participación en la carrera.

Bibliografía

[1] trail tres cerros. URL: http://trailtrescerros.blogspot.com/p/fotos.html.

[2] blogmaldito. URL: https://blogmaldito.com/2025/03/16/fotos-clasificacion-marato-barcelona-marathon-results-resultados-photos/.

[3] persigueme. URL: https://persigueme.es.

[4] marahonphotos. URL: https://marathonphotos.live/Event/Sports%2FCPUK%2F2025%2FLisbon%20Half%20Marathon.

[5] ericBayless. URL: https://github.com/ericBayless/bib-detector.

[6] Lwhieldon. URL: https://github.com/Lwhieldon/BibObjectDetection?tab=readme-ov-file.

[7] "Lucidchart". URL: https://www.lucidchart.com/blog/scrum-for-one.

[8] datascience-pm. URL: https://www.datascience-pm.com/kdd-and-data-mining/.

[9] datascience-pm. URL: https://www.datascience-pm.com/crisp-dm-2/#What_are_the_6_CRISP-DM_Phases.

[10] Wikipedia. URL: https://es.wikipedia.org/wiki/Archivo:CRISP-DM_Process_Diagram.png.

[11] JGraph Ltd. *draw.io - Herramienta de diagramación en línea*. https://www.drawio.com/. 2025.

[12] Overleaf. *Overleaf - Editor de LaTeX en línea*. https://es.overleaf.com/. 2025.

[13] Pascal Brachet. *Texmaker - Editor LaTeX multiplataforma*. https://www.xm1math.net/texmaker/. 2025.

[14] Change Vision, Inc. *Astah - Software de modelado y diagramación*. https://astah.net/. 2025.

[15] BarD Software. *GanttProject - Herramienta de gestión de proyectos con diagramas de Gantt*. https://www.ganttproject.biz/. 2025.

[16] Leburó Coworking. *Precios de coworking en Valladolid*. 2025. URL: https://leburocowork.es/precio-coworking-valladolid.

[17] Glassdoor. *Sueldo de programador junior*. 2025. URL: https://www.glassdoor.es/Sueldos/programador-junior-sueldo-SRCH_KO0,18.htm.

[18] Universidad de Murcia. *VI-3.4.3 Equivalencia entre redes artificiales y biológicas*. https://www.um.es/LEQ/Atmosferas/Ch-VI-3/F63s4p3.htm.

[19] Stefan Elfwing, Eiji Uchibe y Kenji Doya. «Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning». En: *arXiv preprint arXiv:1702.03118* (2017). URL: https://arxiv.org/abs/1702.03118.

[20] Ultralytics. *SiLU (Unidad lineal sigmoidea)*. 2025. URL: https://www.ultralytics.com/es/glossary/silu-sigmoid-linear-unit.

[21] Antonio Richaud. *Redes Perceptrón Multicapa - Imagen del artículo*. Imagen del artículo en el blog de Antonio Richaud. 2020. URL: https://antonio-richaud.com/blog/imagenes/archivo/41-redes-perceptron-multicapa/redes-perceptron-multicapa-banner.jpg.

[22] MathWorks. *Convolución*. URL: https://es.mathworks.com/discovery/convolution.html.

[23] Codificando Bits. *La Convolución en las Redes Convolucionales*. URL: https://codificandobits.com/blog/convolucion-redes-convolucionales/.

[24] Aston Zhang et al. «Dive into Deep Learning». En: Sección 4. self-published, 2020. Cap. 7. URL: https://d2l.ai/chapter_convolutional-neural-networks/channels.html.

[25] Ahmed Mandour. *Downsampling and Upsampling in CNN*. 2022. URL: https://iq.opengenus.org/downsampling-and-upsampling-in-cnn/.

[26] Esraa A. Mohamed et al. *Example of calculating max-pooling and average pooling with filter of size 2X2 and stride 2*. 2022. URL: https://figshare.com/articles/figure/Example_of_calculating_max-pooling_and_average_pooling_with_filter_of_size_2X2_and_stride_2_/21381270.

[27] Ahmed Mandour. *Downsampling and Upsampling in CNN*. 2021. URL: https://iq.opengenus.org/content/images/2022/12/upsampling1.png.

[28] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[29] Yunjie Tian, Qixiang Ye y David Doermann. «YOLOv12: Attention-Centric Real-Time Object Detectors». En: *arXiv preprint arXiv:2502.12524* (2025).

[30] Tri Dao et al. «FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness». En: *arXiv preprint arXiv:2205.14135* (2022). DOI: 10.48550/arXiv.2205.14135. URL: https://arxiv.org/abs/2205.14135.

[31] Tri Dao. «FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning». En: *arXiv preprint arXiv:2307.08691* (2023). DOI: 10.48550/arXiv.2307.08691. URL: https://arxiv.org/abs/2307.08691.

[32] Zain Shariff. *YOLOv12: Actually Pretty Slow*. https://medium.com/@zainshariff6506/yolov12-actually-pretty-slow-120a9e3bfdd4. Publicado en Medium el 24 de febrero de 2025. Feb. de 2025.

[33] Ultralytics. *YOLOv12: Attention-Centric Object Detection - Requisitos*. https://docs.ultralytics.com/models/yolo12/#requirements. 2025.

[34] Kaiming He et al. «Deep Residual Learning for Image Recognition». En: *arXiv preprint arXiv:1512.03385* (2015). URL: https://arxiv.org/abs/1512.03385.

[35] LunarLullaby. *ResBlock.png*. 2015. URL: https://commons.wikimedia.org/wiki/File:ResBlock.png.

[36] Haiying Yuan et al. *Comparison between standard convolution and depthwise separable convolution*. https://www.researchgate.net/figure/Comparison-between-standard-convolution-and-depthwise-separable-convolution_fig1_360180000. 2022.

[37] Arun Meghani. *A Basic Introduction to Separable Convolutions*. https://medium.com/data-science/a-basic-introduction-to-separable-convolutions-b99ec3102728. 2020.

[38] Joseph Redmon et al. «"You Only Look Once: Unified, Real-Time Object Detection"». En: *arXiv preprint arXiv:1506.02640* (2015).

[39] Pragati Baheti. *Train Test Validation Split: How To & Best Practices [2024]*. 2021. URL: https://www.v7labs.com/blog/train-validation-test-set.

[40] ültralytics". URL: https://www.ultralytics.com/glossary/intersection-over-union-iou.

[41] "Wikipedia". URL: https://en.wikipedia.org/wiki/Jaccard_index#/media/File:Intersection_over_Union_-_visual_equation.png.

[42] Ultralytics. *Mean Average Precision (mAP) - Ultralytics Glossary*. n.d. URL: https://www.ultralytics.com/glossary/mean-average-precision-map.

[43] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». En: *European Conference on Computer Vision (ECCV)*. Springer, 2014, págs. 740-755. URL: https://cocodataset.org/.

[44] "sunsmarterjie". URL: https://github.com/sunsmarterjie/yolov12/blob/main/assets/tradeoff.svg.

[45] Agustinus Vina. *Ultralytics YOLO11 has arrived! Redefine what's possible in AI!* Accessed: 2025-04-16. 2024. URL: https://www.ultralytics.com/blog/ultralytics-yolo11-has-arrived-redefine-whats-possible-in-ai.

[46] Glenn Jocher y Jing Qiu. *Ultralytics YOLO11*. Ver. 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[47] Rahima Khanam y Muhammad Hussain. *YOLOv11: An Overview of the Key Architectural Enhancements*. 2024. arXiv: 2410.17725 [cs.CV]. URL: https://arxiv.org/abs/2410.17725.

[48] Priyanto Hidayatullah et al. *YOLOv8 to YOLO11: A Comprehensive Architecture In-depth Comparative Review*. 2025. arXiv: 2501.13400 [cs.CV]. URL: https://arxiv.org/abs/2501.13400.

[49] Glenn Jocher y Jing Qiu. *Ultralytics YOLO11*. Ver. 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[50] Yunjie Tian, Qixiang Ye y David Doermann. *YOLOv12: Attention-Centric Real-Time Object Detectors*. 2025. URL: https://github.com/sunsmarterjie/yolov12.

[51] Yunjie Tian, Qixiang Ye y David Doermann. *YOLOv12 Configuration File: yolov12.yaml*. https://github.com/sunsmarterjie/yolov12/blob/main/ultralytics/cfg/models/v12/yolov12.yaml. 2025.

[52] Nidhal Jegham et al. *YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions*. 2025. URL: https://arxiv.org/html/2411.00201v2/extracted/6229403/figs/YOLOv12_Architecture.drawio.png.

[53] Nidhal Jegham et al. *YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions*. 2025. URL: https://arxiv.org/html/2502.14740v1/extracted/6220667/RLEAN.png.

[54] Amazon Web Services. *¿Qué es el OCR? - Amazon Web Services*. n.d. URL: https://aws.amazon.com/es/what-is/ocr/.

[55] Tali Dekel. *RBNR: Realistic Binaural Noise Rendering*. https://people.csail.mit.edu/talidekel/RBNR.html.

[56] broken scrren. *bib detection big Dataset*. https://universe.roboflow.com/broken-scrren/bib-detection-big. Open Source Dataset. 2024. URL: https://universe.roboflow.com/broken-scrren/bib-detection-big.

[57] David. *Blogmaldito*. 2025. URL: https://blogmaldito.com/.

[58] David. *Fotos y clasificaciones: Mitja de Granollers*. Mar. de 2025. URL: https://blogmaldito.com/2025/03/02/fotos-clasificacion-resultados-photos-mitja-granollers-medio-maraton-half-marathon/.

[59] David. *Fotos y clasificación: Mitja de Sant Cugat*. Feb. de 2025. URL: https://blogmaldito.com/2025/02/02/fotos-y-clasificacion-mitja-sant-cugat/.

[60] Heartex. *Label Studio: Open Source Data Labeling Tool*. https://labelstud.io/. 2023.

[61] Label Studio. *Tutorial: Importing Local YOLO Pre-Annotated Images to Label Studio*. https://labelstud.io/blog/tutorial-importing-local-yolo-pre-annotated-images-to-label-studio/. 2023. URL: https://labelstud.io/blog/tutorial-importing-local-yolo-pre-annotated-images-to-label-studio/.

[62] PaddlePaddle. *PaddleOCR: Multi-language OCR Toolkit based on PaddlePaddle*. https://github.com/PaddlePaddle/PaddleOCR. 2023.

[63] Yuning Du et al. *PP-OCR: A Practical Ultra Lightweight OCR System*. 2020. arXiv: 2009.09941 [cs.CV]. URL: https://arxiv.org/abs/2009.09941.

[64] PaddlePaddle Developers. *PaddleOCR Quick Start*. URL: https://paddlepaddle.github.io/PaddleOCR/v2.10.0/.

[65] Zain Shariff. *A Simple YOLOv12 Tutorial: From Beginners to Experts*. https://medium.com/@zainshariff6506/a-simple-yolov12-tutorial-from-beginners-to-experts-e6e518c3daf4. 2025.

[66] MDN Web Docs. *MVC - MDN Web Docs Glossary*. URL: https://developer.mozilla.org/es/docs/Glossary/MVC.

[67] Jorzel. *Flask MVT - Refactor to Service Layer*. 2023. URL: https://jorzel.hashnode.dev/flask-mvt-refactor-to-service-layer.

[68] Docker Inc. *Docker*. https://www.docker.com/. 2025.

[69] dipualo. *app_reconocimiento_dorsales*. 2025. URL: https://github.com/dipualo/app_reconocimiento_dorsales.

Apéndice A

Extractos de código más relevantes

A.1. Estratificación

En esta sección se presentará, paso a paso, el código utilizado para realizar la división de los datos.

Creación de la tabla que relaciona el número de dorsales con las imágenes:

```
import pandas as pd
import numpy as np
import os

# Crea un fichero con cada imagen y el numero de dorsales que tiene
def numero_de_dorsales_en_imagenes(ruta_salida,ruta_labels, archivos_labels):
    with open(ruta_salida+"num_bibs_img.txt", "w") as num_bibs_img:

        for archivo_label in archivos_labels:

            with open(ruta_labels+archivo_label, "r") as f:
                num_dorsales = sum(1 for _ in f)

            nombre_imagen = archivo_label[:-4]
            num_bibs_img.write(nombre_imagen+","+str(num_dorsales)+"\n")

numero_de_dorsales_en_imagenes(carpeta_salida,carpeta_labels, archivos_labels)
ruta_archivo ='datos_anotados/num_bibs_img.txt'
df = pd.read_csv(ruta_archivo, sep=",", header = None)
df.columns = ['nombre_imagen', 'num_dorsales']
# Se aqrupan las imagenes según tenga uno, dos o más dorsales
df['grupo_num_dorsales'] = df['num_dorsales'].apply(lambda x: '0' if x == 1
    else ('1' if x == 2 else '2'))
# Se guarda la informacion en un csv
df.to_csv("numero_de_dorsales.csv", index=False)
```

Código A.1: Código agrupar imagenes según su número de dorsales

Estratificaicón de las imágenes en los distintos conjuntos de las distintas carpetas del 3-kfold:

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split

ruta_archivo ='datos_anotados/numero_de_dorsales.csv'
num_dorsales = pd.read_csv(ruta_archivo, sep=",")
ruta_archivo ='datos_anotados/formato_dorsal.csv'
formato_dorsales = pd.read_csv(ruta_archivo, sep=",")

# La X se define como el nombre de las imagenes
X = num_dorsales.iloc[:,0]
y_num_dorsales = num_dorsales.iloc[:,2]
y_formato_dorsales = formato_dorsales.iloc[:,1]
# Se crea una variable y para estratificar las imagenes segun su grupos
y_conjunta = y_num_dorsales.astype(str)+y_formato_dorsales.astype(str)

```

```
# Se hace el 3-kfold con una semilla concreta
skf_train = StratifiedKFold(n_splits=3, shuffle=True, random_state=3)
# Inicializamos los distintos conjuntos
train_files, test_files, val_files = [], [], []

# Se dividen las imagenes en dos grupos entrenamiento y test + validacion
for train_index, test_val_index in skf_train.split(X, y_conjunta):

    # Se separa estratificando los datos de test y validacion
    X_test, X_val, y_test, y_val = train_test_split(
        X[test_val_index], y_conjunta[test_val_index],
        test_size=0.5,random_state=42,
        stratify=y_conjunta[test_val_index]
    )
    # Se añaden las imagenes a los distintos conjunto de datos
    test_files.append(X_test)
    val_files.append(X_val)
    train_files.append(X[train_index])
```

Código A.2: Creación del 3-kfold estratificados

Creación de las distintas carpetas a partir del resultado de la estratificación.

```
import shutil, os
folds = ["fold_1", "fold_2", "fold_3"]
for i, fold in enumerate(folds):
    fold_sal = "./" + fold

    for archivo in train_files[i]:
        shutil.copy("./images/"+archivo+".jpg", fold_sal+"/train/images")
        shutil.copy("./labels/"+archivo+".txt", fold_sal+"/train/labels")

    for archivo in test_files[i]:
        shutil.copy("./images/"+archivo+".jpg", fold_sal+"/test/images")
        shutil.copy("./labels/"+archivo+".txt", fold_sal+"/test/labels")

        for number in os.listdir("./bibs_numbers/"):
            if(number.startswith("bibs_"+archivo)):
                shutil.copy("./bibs_numbers/"+number,fold_sal+"/test/
    bibs_numbers")

    for archivo in val_files[i]:
        shutil.copy("./images/"+archivo+".jpg", fold_sal+"/val/images")
        shutil.copy("./labels/"+archivo+".txt", fold_sal+"/val/labels")
```

Código A.3: Creación de las 3 carpetas

A.2. Precisión del OCR sobre dorsales anotados

En esta sección, se expone el código usado para obtener los resultados de los OCR en inglés y chino sobre los dorsales anotados.

Primero es necesario exportar varias librerías y preparar los OCRs.

```
import cv2
import numpy as np
import re
import scipy.io as sio
import os
import pandas as pd
import matplotlib.pyplot as plt

```

```
from paddleocr import PaddleOCR

ocr_en = PaddleOCR(lang='en')
ocr_ch = PaddleOCR(lang = 'ch')
```

Código A.4: Importar librerías y preparas OCRs

A continuación, expondremos las funciones que usamos para conseguir las tasas de acierto deseadas.

La siguiente función permite imprimir por pantalla un dorsal a partir de su ruta:

```
# Muestra una imagen a partir de su ruta
def imShow(path):
  %matplotlib inline

  image = cv2.imread(path)

  fig = plt.gcf()
  plt.axis("off")
  plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
  plt.show()
```

Código A.5: Mostrar dorsal

Mediante la siguiente get_cropped_bib() conseguimos los dorsales etiquetados y los guardamos en una carpeta:

```
def get_cropped_bib(image, input_path, out_path):
    """
    Escribe en la carpeta out_path los bounding boxes anotados de los dorsales

    Args
        image (str): nombre de la imagen
        input_path (str): ruta de la imagen
        out_path (str): ruta donde se guardan los bounding boxes anotados

    """

    img = cv2.imread(input_path + image)
    (h_img, w_img) = img.shape[:2]

    # ELimina images/ y añade labels/
    label_path = input_path[:-7]+"labels/"

    # Coge las bounding boxes de las labels
    with open(label_path + image[:-4]+ '.txt', "r") as file:
        boxes= file.readlines()

    # Cambia el fromato YOLO para trozear la imagen en sus bounding boxes
    x_min_l, x_max_l, y_min_l, y_max_l = [], [], [], []
    for box in boxes:
        box = box.strip().split(' ')
        x, y, w, h = map(float, box[1:])
        x *=w_img
        y *=h_img
        w *= w_img
        h *= h_img
        x_min = int(x-w/2)
        y_min = int(y-h/2)
        x_max = int(x+w/2)
        y_max = int(y+h/2)
        x_min_l.append(x_min)
        x_max_l.append(x_max)
```

```
37          y_min_l.append(y_min)
38          y_max_l.append(y_max)
39
40      # Obtiene los números anotados
41      prefix = image[:-4]
42      with open(input_path[:-7] + "bib_numbers.txt", "r") as file:
43          lines = [line.strip() for line in file if line.startswith(prefix)]
44      numbers = []
45      for line in lines:
46          number = int(line.split(",", 1)[1].strip())
47          numbers.append(number)
48
49
50      for i in range(len(boxes)):
51          # Guarda los bounding boxes
52          crop_img = img[y_min_l[i]:y_max_l[i], x_min_l[i]:x_max_l[i]]
53          crop_name = image[:-4]+'_bib_'+str(i+1)+'.jpg'
54          cv2.imwrite(out_path + crop_name, crop_img)
```

Código A.6: get_cropped_bib()

La función create_labeled_bib_image() reconoce los números de los dorsales que fueron guardados por la función anterior.

```
1  def create_labeled_bib_image(image, input_path, out_path, print_image):
2      """
3      Reconoce los numeros de los dorsales de una imagen.
4
5      Args
6          image (str): nombre de la imagen
7          input_path (str): ruta de la imagen
8          out_path (str): ruta donde se guardan los numeros predichos
9      """
10
11     img = cv2.imread(input_path + image)
12
13     rbn_pred_ch = ocr_ch.ocr(img, det=False, cls = False)
14     rbn_pred_en = ocr_en.ocr(img, det=False, cls = False)
15
16     # Me quedo con la predicción del OCR que tenga más confianza
17     if(rbn_pred_en[0][0][1]>rbn_pred_ch[0][0][1]):
18         rbn_pred = rbn_pred_en[0][0][0]
19     else:
20         rbn_pred = rbn_pred_ch[0][0][0]
21
22     # Me quedo solo con los numeros
23     rbn_pred = ''.join(re.findall(r'\d+', rbn_pred))
24
25     if(print_image):
26         plt.axis("off")
27         plt.imshow(img)
28         plt.show()
29         print("Numero predicho "+rbn_pred)
30         print(image)
31
32     rbn_pred_file = open(out_path + 'rbn_preds.txt', 'a')
33     rbn_pred_file.writelines(f"{image[:-4]},{rbn_pred}\n")
```

Código A.7: create_labeled_bib_image()

Por último, con la siguiente función comparamos los resultados predichos mediante las funciones anteriores con los anotados.

```
1  def validate_bibs(set, print_images=True):
```

```
    """
    Comprueba las distintas métricas del OCR para los conjuntos de datos anotados.

    Args
        set: rutas de las imágenes con los dorsales
        print_images: para imprimir los dorsales con sus anotaciones o no
    Returns
        all_df: tabla que contiene las rutas de los dorsales, su número anotado y su número real
        tasa_aciertos: número que se obtiene al dividir los dorsales acertados entre el total
        fallos: tabla similar a la all_df en la que se encuentran los dorsales mal predichos
    """

    output_path = './validacion_ocr/bib_sets/'

    if os.path.exists(output_path + 'bib_numbers.txt'):
        os.remove(output_path + 'bib_numbers.txt')

    images_path = './datos_anotados/images/'
    bib_numbers = './datos_anotados/bibs_numbers/'
    path_bib_numbers = './datos_anotados/bib_numbers.txt'

    images = [file for file in os.listdir(images_path) if file[-3:]=='jpg']
    images = [image for image in images if image.startswith(set)]

    for image in images:
        get_cropped_bib(image, images_path, output_path)

    images_path = './validacion_ocr/bib_sets/'
    images = [file for file in os.listdir(images_path) if file[-3:]=='jpg']
    images = [image for image in images if image.startswith(set)]

    output_path = './validacion_ocr/num_sets/'

    if os.path.exists(output_path + 'rbn_preds.txt'):
        os.remove(output_path + 'rbn_preds.txt')

    images = sorted(images)
    for image in images:
        create_labeled_bib_image(image, images_path, output_path, print_images)


    true_df = pd.read_csv(path_bib_numbers, delimiter=',', names=['image', 'rbn'])

    true_df = true_df[true_df['image'].str.startswith(set)]
    true_df = true_df.sort_values(by='image')

    pred_df = pd.read_csv('./validacion_ocr/num_sets/rbn_preds.txt', delimiter=',', names=['image', 'pred_rbn'])

    pred_df = pred_df.sort_values(by='image')

    all_df = pd.merge(true_df, pred_df, on='image', how='left')

    fallos = all_df.loc[all_df['rbn'] != all_df['pred_rbn']]
    true_positives = len(all_df.loc[all_df['rbn'] == all_df['pred_rbn']])
    total = len(true_df)

    tasa_aciertos = str(true_positives/total*100)
```

```
59
60     return(all_df, tasa_aciertos, fallos)
```

Código A.8: validate_bibs()

A partir de las funciones anteriores, es posible calcular las tasas de acierto del reconocimiento de dorsales en los distintos conjuntos de imágenes, así como identificar los errores del OCR. A continuación, se muestra cómo se realiza este proceso para el set1:

```
1
2 # Se aplican las funciones y se obtiene su tasa de aciertos y el num de
     dorsales
3 set = "set1_"
4 all_df, tasa_aciertos, fallos_set1 = validate_bibs(set, print_images = False)
5 print("Porcentaje de numeros dorsales acertados por el ocr en el "+set+ " "+
     tasa_aciertos+"%")
6 print("Numero de dorsales del " + set +" "+str(len(all_df))+"\n")
7
8 # Verifico en los dorsales no reconocidos por el OCR
9 # Verifico los dorsales en caso de que haga falta
10
11 imgs_fallo = fallos_set1['image']
12 i = 0
13 images_path = './Data/validacion_ocr/bib_sets/'
14 for img_fallo in imgs_fallo:
15
16     print(img_fallo)
17
18     img = cv2.imread(images_path + img_fallo+".jpg")
19
20     rbn_pred_en = ocr_en.ocr(img, det = False, cls = False)
21     print(rbn_pred_en)
22
23     rbn_pred_ch = ocr_ch.ocr(img, det = False, cls = False)
24     print(rbn_pred_ch)
25
26
27     print("Real ", fallos_ch_y_en.iloc[i,1])
28     imShow(images_path + img_fallo+".jpg")
29     i +=1
```

Código A.9: Importar librerías y preparas OCRs

A.3. Test

Para comprobar el funcionamiento del sistema en las distintas carpetas y con las distintas versiones de YOLO se ha usado el siguiente código, que dividiremos en las siguientes partes.

Primero importamos las librerías necesarias y cargamos los datos con la librería supervisión, el modelo de YOLO y los OCR.

```
1 from ultralytics import YOLO
2 import supervision as sv
3 from PIL import Image
4 import cv2
5 import numpy as np
6 import pandas as pd
7
8 dataset = "./datos_anotados/fold_2/"
9
10 ds = sv.DetectionDataset.from_yolo(
11     images_directory_path=f"{dataset}test/images",
```

```
    annotations_directory_path=f"{dataset}test/labels",
    data_yaml_path=f"{dataset}/data.yaml"
    )

modelo_yolo = YOLO('./runs/detect/fold2_yolov12x/weights/best.pt')

from paddleocr import PaddleOCR,draw_ocr

ocr_en = PaddleOCR(lang='en')
ocr_ch = PaddleOCR(lang = 'ch')
```

Código A.10: Preparación entrenamiento

A continuación, declaramos la ruta donde están los números de los dorsales, creamos varias métricas como son right_boxes que es el número de dorsales predichos correctamente, right_bib_numbers que simboliza el número dorsales reconozidos correctamente, predicted_boxes que es el número de dorsales predichos y true_boxes que es el número de dorsales etiquetados. Después, por cada imagen actualizaremos estás métricas y anotaremos la imagen con los dorsales etiquetados y predichos. Esto último se hace con unas funciones que mostraremos con detalle.

```
bibs_numbers_path = dataset+"test/bibs_numbers/"

metricas = {
    "right_boxes": 0,
    "right_bib_numbers": 0,
    "predicted_boxes": 0,
    "true_boxes": 0
}

from tqdm import tqdm
for i in tqdm(range(len(ds))):

    ruta_imagen, imagen, true_detections = ds[i]

    true_numeros_dorsales, true_detections = get_anotaciones_reales(imagen, ruta_imagen, true_detections)
    pred_numeros_dorsales, pred_detections, metricas = get_anotaciones_predichas(true_numeros_dorsales, true_detections, metricas)

    # Ajusta los tamaños de las anotaciones a los de la imagen
    height, width, channels = imagen.shape
    size = (height+width)/2
    text_scale = 0.4*size/640
    text_padding = 5*int(size/640)

    # Prepara los anotadores del numero de dorsal y el bounding box
    pred_box_annotator = sv.BoxAnnotator()
    pred_label_annotator = sv.LabelAnnotator(text_scale = text_scale, text_padding = text_padding)
    true_box_annotator = sv.BoxAnnotator(color = sv.Color.GREEN)
    true_label_annotator = sv.LabelAnnotator(text_padding = 5,text_scale=0.4, color = sv.Color.GREEN,text_position= sv.Position.BOTTOM_LEFT, )

    # Anota la imagen
    imagen_anotada = imagen.copy()
    imagen_anotada = pred_box_annotator.annotate(scene=imagen_anotada, detections = pred_detections)
    imagen_anotada = pred_label_annotator.annotate(scene=imagen_anotada, detections = pred_detections)
    imagen_anotada = true_box_annotator.annotate(scene=imagen_anotada, detections = true_detections)
    imagen_anotada = true_label_annotator.annotate(scene=imagen_anotada,
```

```
    detections = true_detections)
```

Código A.11: Entrenamiento YOLO

Con get_anotaciones_reales() obtenemos las anotaciones reales de los dorsales añadiendo los números como la clase de los objetos predichos para poder anotarlos en una imagen y comparar visualmente los resultados etiquetados y los predichos.

```
def get_anotaciones_reales(imagen, ruta_imagen, detections):
    """
    Dada una imagen, la ruta y las deteciones de YOLO y añade la información
    los numeros de los dorsales
    Args
        imagen, ruta_imagen, detections
    Returns
        true_numeros_dorsales, true_detections
    """
    nombre_imagen = ruta_imagen.rsplit('/', 1)[-1]

    bib_numbers_path = bibs_numbers_path + "bibs_" + nombre_imagen[:-3]+"txt"
    with open(bib_numbers_path, "r") as f:
        true_numeros_dorsales = list(numero.strip() for numero in f)

    true_detections.data = {'class_name': true_numeros_dorsales}

    return(true_numeros_dorsales, true_detections)
```

Código A.12: get_anotaciones_reales()

Mediante get_anotaciones_predichas() no sólo conseguimos las anotaciones predichas por el sistema, sino que también actualizamos las métricas.

```
def get_anotaciones_predichas(true_numeros_dorsales, true_detections, metricas):
    """
    Devuelve la imagen con los numeros de los dorsales anotados y actualiza las metricas
    Args
        true_detections, true_numeros_dorsales, metricas
    Returns
        pred_detections, pred_numeros_dorsales
    """

    results = modelo_yolo(imagen, verbose = False, conf=0.1)[0]
    pred_detections = sv.Detections.from_ultralytics(results).with_nms()
    pred_numeros_dorsales= get_numeros_dorsales(pred_detections.xyxy, imagen)

    # Se gbuscan y guardon los numeros no reconocidos de los dorsales
    dorsales_no_reconocidos = []
    for i, pred_numeros_dorsal in enumerate(pred_numeros_dorsales):
        if(pred_numeros_dorsal == "No reconocido"):
            dorsales_no_reconocidos.append(i)

    # Se eliminan de los numero predichos los no reconocidos
    pred_numeros_dorsales = np.delete(pred_numeros_dorsales, dorsales_no_reconocidos)

    # Se eliminan los bounding boxes de los numeros no reconocidos
    pred_detections = sv.Detections(
        xyxy=np.delete(pred_detections.xyxy, dorsales_no_reconocidos, axis=0),
        confidence=np.delete(pred_detections.confidence, dorsales_no_reconocidos, axis=0),
        class_id=np.delete(pred_detections.class_id, dorsales_no_reconocidos, axis=0),
```

```
28          tracker_id=np.delete(pred_detections.tracker_id,
    dorsales_no_reconocidos, axis=0) if pred_detections.tracker_id is not None
    else None,
29          data={k: np.delete(v, dorsales_no_reconocidos, axis=0) for k, v in
    pred_detections.data.items()}
30      )
31      pred_detections.data = {'class_name': pred_numeros_dorsales}
32
33      # Si el centro de la imagen detectada y predicha dista menos de 10 pixeles
    en cada eje
34      # se considera que son el mismo bounding box
35      dist_max = 10
36      for i, true_box in enumerate(true_detections.xyxy):
37          for j, pred_box in enumerate(pred_detections.xyxy):
38              if (abs(pred_box[0]-true_box[0])<dist_max and abs(pred_box[1]-
    true_box[1])<dist_max):
39                  # Se apunta que se ha acertado en un bounding box
40                  metricas['right_boxes']+=1
41                  if(true_numeros_dorsales[i] == pred_numeros_dorsales[j]):
42                      # Se apunta que se ha acertado un numero de dorsal
43                      metricas['right_bib_numbers']+=1
44
45      # Se apuntan el numero de dorsales etiquetados
46      metricas['true_boxes'] += len(true_detections.xyxy)
47
48      # Se apunta el numero de dorsales predichos
49      metricas['predicted_boxes'] += len(pred_detections.xyxy)
50
51      return(pred_numeros_dorsales, pred_detections, metricas)
```

Código A.13: get_anotaciones_predichas()

En la anterior función se llama a get_numeros_dorsales() que a partir de las bounding boxes de los dorsales devuelve el número usando los OCR.

```
1  def get_numeros_dorsales(cajas_dorsales, img):
2      """
3      Dado imagen y la posiciones de los bounding boxes
4      devuelve los numeros de los dorsales
5      Args
6          bounding boxes dorsales
7          imagen de los dorsales
8          OCRs a usar para reconocer los dorsales
9      Returns
10         Lista con los numeros de los dorsales
11     """
12     numeros_dorsales = []
13     if len(cajas_dorsales) > 0:
14         for caja_dorsal in cajas_dorsales:
15             # crop out detected bib
16             (x_min, y_min, x_max, y_max) = [round(num) for num in caja_dorsal]
17             img_dorsal = img[y_min:y_max, x_min:x_max]
18
19             rbn_pred_ch = ocr_ch.ocr(img_dorsal, det = False, cls = False)
20             rbn_pred_en = ocr_en.ocr(img_dorsal, det = False, cls = False)
21
22             if(rbn_pred_en[0][0][1]>rbn_pred_ch[0][0][1]):
23                 bib_number = rbn_pred_en[0][0][0]
24                 conf = rbn_pred_en[0][0][1]
25             else:
26                 bib_number = rbn_pred_ch[0][0][0]
27                 conf = rbn_pred_ch[0][0][1]
28
29             bib_number = "".join([c for c in bib_number if c.isdigit()])
```

```
30
31              if(len(bib_number)>0 and conf>0.7):
32                  bib_number = bib_number
33              else:
34                  bib_number = "No reconocido"
35
36              numeros_dorsales.append(bib_number)
37
38      return numeros_dorsales
```

Código A.14: get_numeros_dorsales()

Estas funciones son usadas durante otras pruebas como las pruebas en los distintos grupos de imágenes según los dorsales, para sacar las imágenes anotadas y son la base del sistema de reconocimiento de dorsales.

Por último, calculamos y mostramos las métricas relevantes (precisión, recall y F1-score) a partir de las calculadas anteriormente sobre el conjunto de test.

```
1 # Se describen las metricas dadas obtenidas en el test
2 print("Numero de dorsales etiquetados "+str(metricas['true_boxes']))
3 print("Numero de dorsales predichos "+str(metricas['predicted_boxes'])+"\n")
4
5 print("Numero de dorsales identificados correctamente "+ str(metricas['
    right_bib_numbers']))
6 print("Numero de dorsales detectados correctamente "+ str(metricas['right_boxes
    '])+"\n")
7
8 accuracy_bib_numbers = metricas['right_bib_numbers']/metricas['true_boxes']
9 accuracy_boxes = metricas['right_boxes']/metricas['true_boxes']
10
11 recall_bib_numbers = metricas['right_bib_numbers']/metricas['predicted_boxes']
12 recall_boxes = metricas['right_boxes']/metricas['predicted_boxes']
13
14
15 print("Precision detección "+str(accuracy_boxes))
16 print("Recall detección "+str(recall_boxes))
17 print("F1 score detección "+str(2*recall_boxes*accuracy_boxes/(accuracy_boxes+
    recall_boxes))+"\n")
18
19 print("Precision reconocimiento " +str(accuracy_bib_numbers))
20 print("Recall reconocimiento " +str(recall_bib_numbers))
21 print("F1 score reconocimiento " +str(2*recall_bib_numbers*accuracy_bib_numbers
    /(accuracy_bib_numbers+recall_bib_numbers)))
```

Código A.15: Métricas test

Apéndice B

Manual de la aplicación

A continuación, se explica cómo instalar la aplicación web a partir del repositorio de GitHub, utilizando Docker. Una vez completada la instalación, se podrá verificar su funcionamiento siguiendo los ejemplos de uso, basodos en los casos de uso explicados en el cápitulo de la aplicación web.

B.1. Instalación

La aplicación se instala y ejecuta a través de Docker, lo cual garantiza un entorno controlado, replicable y libre de conflictos con el sistema operativo del usuario. Para ello, es imprescindible tener instalado Docker y seguir los siguintes pasos para obtener el código del repositorio, contruir y ejecutar la imagen de Docker:

1. **Clonar el repositorio**

   Se descarga el código fuente desde el repositorio remoto de github:

   ```
   git clone https://github.com/dipualo app_reconocimiento_dorsales.git
   ```

2. **Construir la imagen del Docker**

   Se genera una imagen en el directorio de la aplicación que encapsula todo el entorno de ejecución, incluyendo las dependencias, el código y la configuración. En esta etapa, el `Dockerfile` excluye las bibliotecas específicas para GPU, asumiendo una ejecución sobre CPU, con el objetivo de reducir el tamaño de la imagen. Además, se emplea una imagen base `python:slim` para optimizar aún más el peso final.

   ```
   cd app_reconocedor dorsales
   docker build -t app_reconocedor_dorsales
   ```

3. **Ejecutar el contenedor**

   La opción -it permite la interacción con el contenedor a través de la terminal, mientras que –rm garantiza su eliminación automática al finalizar la ejecución, evitando ocupar espacio innecesario. Por otro lado, -p 5000:5000 expone la aplicación en el puerto 5000, que es el estándar utilizado por aplicaciones desarrolladas con Flask.

   ```
   docker run -it --rm -p 5000:5000 app_reconocedor_dorsales
   ```

B.2. Casos de uso

Exploraremos como funciona la aplicación comprando como se realizan los casos de uso.

B.2.1. Cargar imágenes

Al iniciar la app nos aparece la siguiente pantalla en la que podemos arrastrar las imágenes o hacer clic al recuadro para subirlas.

Figura B.1: Pantalla inicial aplicación web.

Tras subirlas se mostrarán las imágenes. Si estas se exceden del tamaño del contendor dónde han sido insertadas, aprecera un slider para poder tener una previsualización de las mismas.

Figura B.2: Aplicación web imágenes cargadas.

Tras la carga de las imágenes, se permite su cancelación volviendo a la pantalla inicial, cargar más imágenes y enviar las imágenes al sistema. También, a partir de los enlaces, se pueden ir en cualquier momento a realizar los distintos casos de uso.

B.2.2. Enviar imágenes

Tras su carga se pulsa al botón enviar y se va a una pantalla de carga hasta que se procesan.

Figura B.3: Aplicación web enviando imágenes.

Al acabar el procesamiento, se va a la pantalla de busqueda de imágenes.

B.2.3. Búsqueda de imágenes

Desde la siguiente pantalla, se permite buscar imágenes con un número de dorsal. Para facilitar este proceso, se muestra los números de dorsales asociados con las rutas de las imágenes correspondientes.

Figura B.4: Aplicación web buscando por dorsal.

B.2.4. Borrar imágenes

Desde la pantalla ver predicciones, se podrá ver los reoconocimientos de dorsales de las imágenes enviadas al modelo. También se permite eliminar imágenes del sistema.

Figura B.5: Aplicación web visualizando las predicciones.