Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Detección en tiempo real de comportamiento anómalo en animales utilizando Deep Learning y visión por computadora

---

Alumno: Carlos Escribano Merino

Tutor/es: Francisco Hernando Gallego
Diego Martín de Andrés

Fecha: 16 de septiembre de 2025

Detección en tiempo real de comportamiento anómalo en animales utilizando Deep Learning y visión por computadora

Carlos Escribano Merino

16 de septiembre de 2025

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

En primer lugar me gustaría agradecer a mis tutotres Diego Martín de Andrés y Francisco Hernando Gallego por darme la oportunidad de desarrollar y trabajar en este TFG y por toda la ayuda y las sugerencias aportadas.

También agradecer a toda mi familia por apoyarme y soportarme durante todos estos años de indecisiones.

Respecto a los datos, agradezco a los integrantes del estudio Edinburgh Pig Behavior Video Dataset por permitir el libre uso de los vídeos en proyectos de investigación.

Por último, y no menos importante, agradecer a Animal Data Analytics por cerderme vídeos para darles uso, que finalmente no se utilizaron, pero se tuvieron de respaldo.

Resumen

La visión por computadora es un campo fundamental dentro de la inteligencia artificial teniendo aplicaciones en diversas áreas, incluyendo la monitorización del comportamiento animal. Los modelos basados en deep learning han demostrado ser altamente eficaces en la detección y análisis de patrones complejos en imágenes y vídeos. Sin embargo, la implementación en entornos reales presenta desafíos, como la necesidad de modelos eficientes capaces de operar en tiempo real.

Este trabajo se centra en el desarrollo de un sistema basado en deep learning para la detección y seguimiento de cerdos en sus respectivas granjas, con el objetivo de identificar comportamientos anómalos que puedan indicar problemas de salud o bienestar. Castilla y León es una de las regiones con mayor producción porcina en España, lo que hace especialmente relevante la aplicación de estas tecnologías en su sector ganadero. Para ello, se emplearán modelos avanzados de detección de objetos y algoritmos de seguimiento optimizados, permitiendo un análisis automatizado en tiempo real. Esta propuesta busca contribuir a la modernización del sector, proporcionando herramientas que faciliten la monitorización eficiente del estado de los animales y mejoren la gestión de las explotaciones porcinas.

Palabras claves: visión por computadora, aprendizaje profundo, seguimiento, comportamiento anómalo animal, ganadería de precisión.

Abstract

Computer vision is a fundamental field within artificial intelligence, with applications in multiple domains, including animal behavior monitoring. Deep learning-based models have proven to be highly effective in detecting and analyzing complex patterns in images and videos. However, real-world deployment presents challenges, such as the need for efficient models capable of operating in real-time.

This work focuses on developing a deep learning-based system for the detection and tracking of pigs on their respective farms, aiming to identify anomalous behaviors that may indicate health or health issues. Castilla y León is one of the leading pig production regions in Spain, making the application of these technologies particularly relevant to its livestock sector. To achieve this, state-of-the-art object detection models and optimized tracking algorithms will be employed, enabling automated real-time analysis. This proposal seeks to contribute to the modernization of the sector by providing tools that facilitate efficient animal monitoring and enhance the management of pig farming operations.

Keywords: computer vision, deep learning, tracking, anomalous animal behavior, precision livestock farming.

Parte I

Memoria del Proyecto

Capítulo 1

Introducción

En 2023, España fue el país de la Unión Europea que produjo mayor cantidad de carne de cerdo (Figura 1.1), siendo Castilla y León una de las comunidades autónomas con mayor número de explotaciones de ganado porcino por sistema productivo [1]. En adición, en 2024 el país se afianzó como referente europeo respecto a la producción de carne porcina y tercer mayor productor mundial (por detrás de China y Estados Unidos) [2], por lo que es importante mantener un foco en este aspecto.

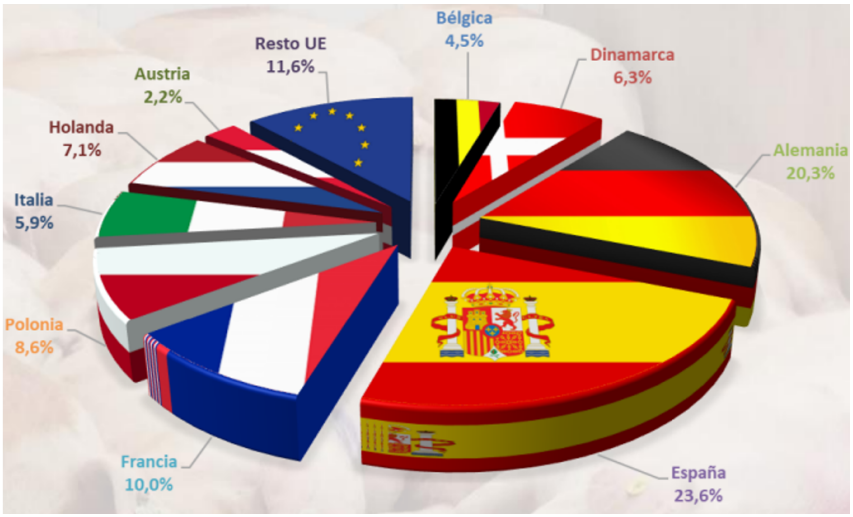


Figura 1.1: Producción de carne de cerdo en la unión europea durante el año 2023.

El bienestar animal y la detección temprana de problemas de salud son aspectos críticos en la producción porcina. La identificación de comportamientos anómalos en los cerdos es esencial para prevenir enfermedades, reducir el estrés y mejorar la calidad de la producción ganadera [3].

Tradicionalmente, la monitorización del comportamiento animal en explotaciones porcinas se ha basado en la observación manual por parte de los trabajadores. Este enfoque presenta limitaciones significativas, como la subjetividad, la posibilidad de errores humanos y la ineficiencia en granjas de gran escala. [4] Además, la supervisión manual no permite una vigilancia continua, lo que dificulta la detección temprana de problemas como enfermedades, lesiones o comportamientos agresivos [5].

En este contexto, la Inteligencia Artificial (IA) ha emergido como una herramienta clave en la ganadería de precisión. La visión por computadora y el *Deep Learning* han demostrado ser efectivos en la monitorización automática del comportamiento animal en tiempo real. Estas tecnologías permiten analizar imágenes y vídeos para detectar patrones de movimiento y comportamientos atípicos sin intervención humana directa, mejorando la precisión y rapidez en la identificación de anomalías [6].

En los últimos años, el uso de modelos avanzados de detección de objetos como YOLO (You Only Look Once) [7] ha ganado protagonismo en la visión por computadora por su capacidad para identificar eventos relevantes en tiempo real con alta precisión. Estas arquitecturas permiten no solo detectar individuos dentro del campo de visión, sino también analizar sus interacciones y movimientos para inferir posibles anomalías. Su aplicación en explotaciones ganaderas representa una oportunidad para automatizar el seguimiento del comportamiento animal con un nivel de detalle que antes era impensable.

Este trabajo tiene como objetivo desarrollar un sistema basado en *Deep Learning* y visión por computadora para la detección en tiempo real de comportamientos anómalos en cerdos. La implementación de este sistema en granjas industriales buscaría mejorar el bienestar animal, reducir la carga de trabajo de los operarios y optimizar la gestión de la producción porcina en Castilla y León.

1.1. Planteamiento del problema

El monitoreo del comportamiento de los cerdos es esencial para detectar signos tempranos de enfermedad, estrés o agresividad que pueden afectar tanto la salud de los animales como la rentabilidad de la producción. Sin embargo, el método tradicional basado en la observación manual presenta varias limitaciones:

- **Subjetividad y variabilidad:** La interpretación del comportamiento animal puede variar entre los trabajadores, lo que genera inconsistencias en la detección de problemas [4].
- **Falta de monitoreo continuo:** La supervisión manual es intermitente y no permite un control permanente del estado de los animales [5].
- **Dificultad para analizar grandes volúmenes de datos:** En explotaciones con miles de cerdos, la observación individualizada es inviable sin apoyo tecnológico [6].

Estas limitaciones pueden resultar en la detección tardía de enfermedades como la cojera, infecciones respiratorias o problemas derivados del estrés, lo que provoca pérdidas económicas y compromete el bienestar animal [3].

El uso de visión por computadora y *Deep Learning* ofrece una solución a estos desafíos mediante la monitorización automática del comportamiento porcino. Gracias a algoritmos de detección de objetos y seguimiento en vídeo, es posible identificar de manera precisa y en tiempo real signos de comportamiento anómalo, reduciendo la dependencia de la observación manual y mejorando la eficacia en la detección de problemas de salud [6].

1.2. Objetivos del trabajo

Los principales objetivos abarcados por este proyecto son:

- **OBJ-01**. Desarrollar un sistema basado en Deep Learning para la detección en tiempo real de comportamientos anómalos en cerdos mediante visión por computadora.
- **OBJ-02**. Integrar algoritmos de seguimiento de objetos para mantener la identificación de cada cerdo en el tiempo.
- **OBJ-03**. Integrar algoritmos de estimación de peso mediante visión por computadora.
- **OBJ-04**. Garantizar que el modelo sea capaz de identificar patrones anómalos en los cerdos y genere alertas en función de ello.

1.2.1. Restricciones

Las restricciones encontradas son:

- **RES-01**. Disponer de suficientes datos para entrenar el modelo y posteriormente probarlo [8].
- **RES-02**. Disponer de un hardware suficientemente potente para entrenar el modelo.
- **RES-03**. Distinción entre tamaños de animales para la estimación de pesos.

1.3. Estructura de la memoria

Este documento se organiza en varios capítulos que abordan el desarrollo del sistema de detección de comportamiento anómalo en cerdos y su aplicación en la ganadería de precisión. En la Parte I se encuentra el siguiente contenido:

- **Capítulo 1.** Aborda la introducción, el planteamiento del problema y los objetivos deseados con el proyecto, así como la estructura de la memoria.
- **Capítulo 2.** Describe la metodología empleada en la realización del proyecto, incluyendo la planificación temporal y el presupuesto del mismo.
- **Capítulo 3.** Se presenta el contexto de la tecnología en aplicaciones similares a este proyecto, abarcando varias opciones tecnológicas actuales.

La Parte II contiene:

- **Capítulo 4.** Este capítulo se centra en todo el desarrollo práctico del proyecto, incluyendo los datos utilizados, los casos de estudio, las tecnologías empleadas y la muestra de los cuadernos.
- **Capítulo 5.** Abarca toda la explicación sobre la preparación y carga de los datos empleados.
- **Capítulo 6.** Contiene el proceso seguido en el entrenamiento, así como otras métricas, hiperparámetros y resultados.
- **Capítulo 7.** Este capítulo se centra en comentar los resultados obtenidos a lo largo del proyecto, desde un punto de vista del trabajo y otro punto de vista personal, incluyendo posibles mejoras o implementaciones a futuro.
- **Apéndice A.** Breve manual de instalación del entorno utilizado para poder ejecutar los scripts.
- **Apéndice B.** Contenido adjunto a la memoria.
- **Webgrafía.**

Capítulo 2

Planificación

En este capítulo se presenta la metodología seguida a lo largo del desarrollo de este TFG, así como las herramientas empleadas, la planificación temporal, los presupuestos y el balance temporal y económico. En este proyecto se ha seguido una metodología agile basada en Kanban descrita en la Sección 2.1, que permite tener un enfoque más flexible y adaptable permitiendo gestionar las tareas de una forma visual y continua.

Kanban es un método visual de gestión del trabajo que permite organizar tareas en diferentes etapas de un flujo, desde que se planifican hasta que se completan.

Su funcionamiento se basa en visualizar el trabajo, limitar la cantidad de tareas que se hacen simultáneamente y mejorar progresivamente la forma en que se trabaja. Kanban no impone roles ni ciclos fijos, como otros marcos Agile, lo que lo hace altamente flexible. Su filosofía es hacer el trabajo visible y enfocar el esfuerzo en una cosa a la vez para mantener un flujo continuo.

2.1. Metodología de trabajo

Como he mencionado anteriormente, este Trabajo Final de Grado se estructura y gestiona mediante Kanban Personal [9] (basado en Kanban) debido a la fácil adaptación al trabajo individual, algo que otras metodologías ágiles no permiten tan fácilmente. A continuación, se explica en qué consiste Kanban Personal.

2.1.1. Método de gestión Kanban Personal

Kanban personal es un sistema de organización del trabajo individual que permite planificar, visualizar y gestionar de manera eficaz las tareas personales o profesionales. Su base está en los principios del método Kanban tradicional, pero su enfoque está adaptado a las necesidades de una sola persona, no de equipos ni proyectos complejos.

Este enfoque parte de la idea de que muchas veces nuestras tareas se acumulan y nos cuesta mantener el control, priorizar o simplemente avanzar con claridad. El Kanban personal ofrece una estructura ligera pero potente que ayuda a entender qué hay que hacer, en qué se está trabajando en este momento y qué se ha completado, todo de un vistazo.

- **Estructura del sistema.**
  La herramienta central de Kanban personal es el tablero, donde se representan las tareas en forma de tarjetas que se mueven entre columnas que reflejan su estado. Aunque el esquema más habitual usa tres columnas básicas —Por hacer, En proceso y Hecho—, en mi caso lo distribuí en cinco columnas —Largo Plazo, Corto Plazo, En proceso, Revisar y Completo/terminado— empleando la aplicación de Trello, como explicaré y mostraré más adelante.

- **Principios fundamentales.**
  Aunque es un método flexible, hay dos pilares fundamentales en Kanban personal:
  Visualizar el trabajo: sacar de la cabeza todas las tareas pendientes y mostrarlas en un tablero (ya sea físico o digital). Esto no solo reduce el estrés mental, sino que permite entender el volumen de trabajo y tomar decisiones más claras.
  Limitar el trabajo en curso. Consiste en establecer un número máximo de tareas que se pueden tener activas al mismo tiempo. Esto fomenta el enfoque, evita la dispersión y mejora la eficiencia.

- **Ventajas prácticas.**
  Utilizar Kanban personal permite ganar claridad sobre las propias prioridades, detectar cuellos de botella, evitar la multitarea improductiva y desarrollar hábitos de trabajo más sostenibles. Además, facilita revisar el progreso y aprender de forma continua, ya que el tablero actúa también como un registro visual del trabajo realizado.

- **Adaptabilidad.**
  Otra de sus grandes virtudes es su adaptabilidad. Puede usarse para gestionar tanto obligaciones laborales como tareas del hogar, estudios o proyectos personales, como podría ser este Trabajo de Fin de Grado.

2.1.2. Entorno de trabajo

- **Espacio de trabajo compartido:** este espacio se trata de un repositorio github en el que fui subiendo los avances y cambios realizados para que el tutor me fuese indicando correcciones.

- **Tablero del proyecto:** es un tablero Kanban, como se ve en la Figura 2.1, en el que distribuyo todas las tareas del proyecto entre las cinco columnas: en las dos primeras columnas coloco las tareas que aún no se han comenzado. Se dividen en dos columnas debido a la prioridad de las tareas, siendo la columna de "corto plazo" más prioritaria; en la columna "en proceso" se encuentran las tareas en las que estoy trabajando actualmente; y para terminar las columnas "revisar" y "completo/terminado" contienen las tareas completadas, con la diferencia que en la segunda el trabajo ya está revisado. Para trabajar con un tablero así empleé la herramienta Trello [10], que permite una gran personalización.

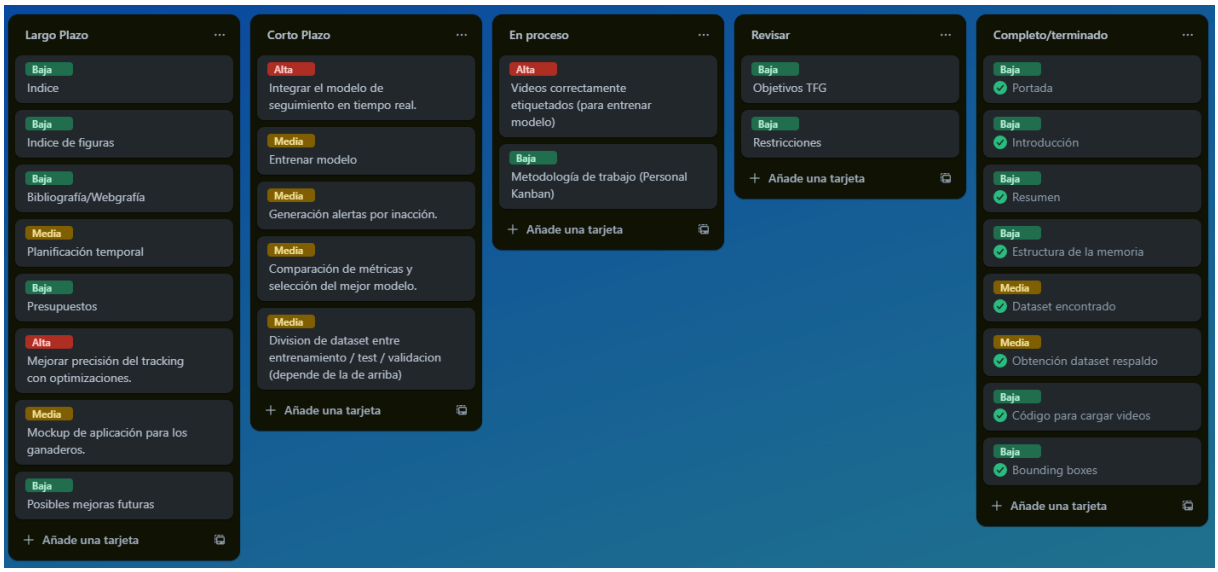


Figura 2.1: Tablero del proyecto en Trello.

2.2. Planificación temporal

A diferencia de metodologías que organizan el trabajo en bloques fijos de tiempo (como los sprints en Scrum), la estrategia adoptada se fundamenta en el Kanban Personal, un sistema que favorece la flexibilidad y la adaptabilidad al ritmo individual de trabajo. Este modelo prescinde de una planificación temporal rígida, sustentándose, en cambio, en la gestión visual del flujo de tareas en tiempo real.

En lugar de fijar fechas concretas para cada tarea, se ha priorizado mantener una visión clara del estado de cada actividad, lo que ha permitido adaptar el avance del proyecto a la disponibilidad real de tiempo y energía en cada momento. Esta forma de organización es especialmente útil en contextos donde los horarios son irregulares o variables.

Para asegurar un progreso consistente a lo largo del tiempo, se emplearon indicadores cualitativos de carga de trabajo (ver apartado de estimación de esfuerzo), lo que permitió alternar tareas más exigentes con otras más ligeras y evitar bloqueos por saturación.

En resumen, más que una planificación temporal cerrada, se ha optado por una planificación dinámica y contextual, centrada en la visualización del trabajo y la toma de decisiones constante basada en prioridades.

2.2.1. Estimación de esfuerzo (KPIs)

Para facilitar la gestión del flujo de trabajo, establecí un sistema de estimación de esfuerzo cualitativo, basado en tres indicadores combinados:

- Carga de trabajo separada.
- Dificultad técnica.
- Tiempo estimado a dedicar a cada tarea.

Cada tarjeta fue etiquetada con un color según el nivel de esfuerzo previsto, como se observa de la Figura 2.2 hasta la Figura 2.4:

- **Alta (color rojo):** tareas que requieren mayor concentración, más tiempo o conocimientos específicos.

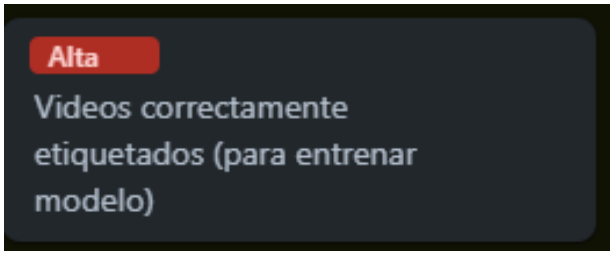


Figura 2.2: Tarea con etiqueta Alta.

- **Media (color naranja):** tareas que requieren también concentración, tiempo o conocimientos, pero no tantos como para las etiquetas rojas.

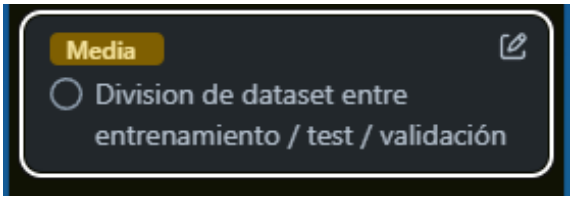


Figura 2.3: Tarea con etiqueta Media.

- **Baja (color verde):** tareas que son más livianas.

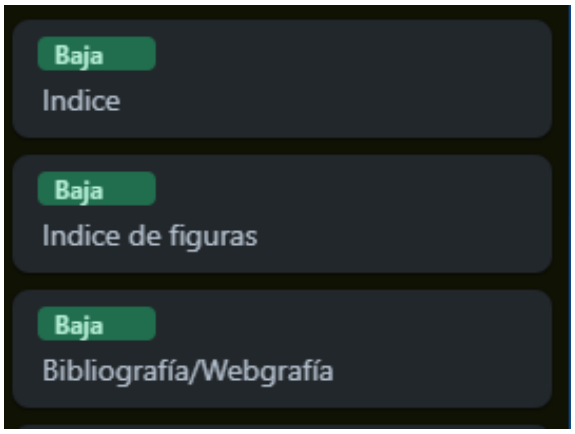


Figura 2.4: Tarea con etiqueta Baja.

2.3. Presupuestos

En este aparatado se encuentra un análisis detallado de los presupuestos necesarios para llevar a cabo el trabajo, indicando los recursos utilizados. Estos se dividen en tres, en función de que sea un recurso hardware, software o humano.

2.3.1. Hardware

El proyecto se desarrollará en el ordenador portátil con las siguientes especificaciones: procesador Intel Core i5-12450H, 16GB RAM, disco SSD de 512GB y tarjeta gráfica de portátil RTX 3050. Este portátil de gama media-alta que costó 650€ (actualmente está por 800€ [10]) tiene 15 meses de vida útil. Más adelante calcularemos su coste de amortización asociado al período de trabajo de este proyecto, alrededor de 8 meses.

El etiquetado se realizará en el ordenador de torre con: procesador Intel Xeon CPU E5-2680 V4, 32GB RAM, disco SSD de 520GB y otro de 2TB y tarjeta gráfica NVIDIA GeForce GTX 1060 6GB, que ronda alrededor de los 1200€ (entre unas piezas y otras) y tiene 9 años de vida útil. Como con el anterior, calcularemos después su coste de amortización. En el coste ya van incluidos los cambios de componentes por mal funcionamiento o rendimiento.

También necesitaré almacenamiento suficiente para guardar los datos, para ello utilizaremos el disco HDD externo de 5TB, que tiene espacio de almacenamiento de sobra para ello. Tiene una vida útil de 7 años.

En adición, se necesita conexión a Internet para buscar la información necesaria para poder llevar a cabo este trabajo, para poder usar algunas de las herramientas que he utilizado para la realización del mismo, y finalmente, para poder descargar los datasets que se emplearán posteriormente. La tarifa elegida oferta más cosas añadidas a la conexión a Internet, pero el coste único de éste es de 25€/mes, separando el resto de aditivos.

Los anteriores elementos se representan en la Tabla 2.1.

Recurso	Coste	% Uso	Total
Ordenador portátil	650€	53.33 %	346.67€
Ordenador de torre	1500€	7.41 %	111.15€
HDD externo 5 TB	128€	9.52 %	12.19€
Internet	25€/mes	100 %	200€
			670.01€

Tabla 2.1: Costes hardware

2.3.2. Software

Los recursos software empleados son gratuitos, bien porque son de libre acceso; porque son de licencia libre; porque tienen versiones inferiores gratuitas; o porque la licencia (en el caso del sistema operativo) venía con la compra del ordenador. Los elementos software utilizados fueron los que aparecen en la Tabla 2.2. Overleaf y Microsft Office fueron empleados para la redacción de la memoria, Trello para llevar el seguimiento y la organización del trabajo. Para el etiquetado, se utilizó CVAT. Y para el desarrollo del código Jupyter Notebook. Todo ello implementado en sistemas operativos Windows 10 y 11, del ordenador de torre y portátil respectivamente.

Recurso	Coste	Total
Overleaf	0€	0€
Microsoft Office	0€	0€
Trello	0€	0€
CVAT	0€	0€
Jupyter Notebook	0€	0€
Windows 10	0€	0€
Windows 11	0€	0€
		0€

Tabla 2.2: Costes software

2.3.3. Recursos humanos

Para calcular el coste en recursos humanos para este trabajo hay que tener en cuenta la duración de éste y los salarios de los distintos roles implicados. El proyecto tiene una duración

de 300 horas, que se ha dividido como aparece en la tabla (ver Tabla 2.3). Los roles implicados en este proyecto han sido: técnico de datos, fue necesario para preparar, limpiar y organizar los datos, con sueldo base promedio de 24mil/año [11]; científico de datos o data scientist analiza los datos y extrae conclusiones útiles de ellos, con sueldo base promedio de 38mil/año [12]; y el último rol, desarrollador Python, necesario para implementar todo el código necesario para llevar a cabo el proyecto, con un sueldo base promedio de 30mil/año [13]. Para calcular el salario por hora, tendré en cuenta una jornada de 40 horas/semana y que se trabajan 52 semanas al año, son 2080 horas anuales, el resultado es el de la tabla (ver Tabla 2.3). Las empresas pagan aproximadamente un 30 % del salario bruto a la Seguridad Social [14], por lo que así calcularemos el total.

2.4. Balance temporal y ecónomico

En esta sección se va a describir el balance temporal y económico del proyecto.

Puesto	Salario por hora	Horas totales	Total sin SS	Coste SS	Total
Técnico de datos	11.54€/hora	70	807.8€	242.34€	1050.14€
Científico de datos	18.27€/hora	130	2375.1€	712.53€	3087.63€
Desarrollador Python	14.42€/hora	100	1442€	432.6€	1874.6€
					6012.37€

Tabla 2.3: Costes de recursos humanos

Con los resultados mostrados en las tablas (ver Tabla 2.1, Tabla 2.2 y Tabla 2.3), queda como presupuesto total del proyecto el mostrado en la Tabla 2.4.

Tipo presupuesto	Coste
Recursos hardware	670.01€
Recursos software	0€
Recursos humanos	6012.37€
	6682.38€

Tabla 2.4: Presupuesto total del proyecto

Capítulo 3

Estado del arte

3.1. Entorno de negocio

Como se mencionó en la introducción, España ocupa un lugar destacado en la producción de carne de cerdo tanto en la Unión Europea como a nivel mundial, con regiones como Castilla y León liderando en número de explotaciones porcinas. Este posicionamiento estratégico hace que la mejora continua en la gestión y el bienestar animal sea una prioridad, no solo para mantener la competitividad del sector sino también para garantizar la calidad y sostenibilidad de la producción. Tradicionalmente, la monitorización del comportamiento animal ha dependido de la observación manual, la cual presenta limitaciones importantes en términos de subjetividad, continuidad y capacidad de análisis en grandes explotaciones. Por ello, la adopción de tecnologías basadas en *Machine Learning* y visión por computadora se presenta como una solución clave para superar estos retos y optimizar el control y cuidado de los animales en tiempo real.

3.1.1. Problemática actual en la industria porcina

A pesar del avance tecnológico y del peso económico del sector porcino, aún existen diversas problemáticas relacionadas con el bienestar animal, la sostenibilidad ambiental y el control normativo, muchas de las cuales fueron abordadas en el Real Decreto 159/2023, de 7 de marzo [15] [16].

En primer lugar, una de las preocupaciones más destacadas es la necesidad de garantizar comportamientos naturales en los animales. La normativa establece la obligatoriedad de proporcionar material manipulable adecuado, como paja, cuerdas o bloques de madera, que permita a los cerdos expresar conductas exploratorias. La carencia de estos estímulos puede derivar en estrés, frustración y comportamientos agresivos, lo que repercute negativamente tanto en el bienestar animal como en la calidad de la producción.

Asimismo, el decreto hace hincapié en la obligación de disponer de zonas de aislamiento para animales enfermos o heridos, a fin de evitar contagios y facilitar su recuperación. Sin embargo, la dificultad para detectar precozmente signos de enfermedad o comportamiento anó-

malo limita la eficacia de estas medidas. En granjas de gran escala, la identificación manual de estos casos resulta poco práctica, lo que incrementa el riesgo de propagación de enfermedades y de pérdidas económicas.

Otra barrera importante es el alto coste y el tiempo asociado a la supervisión tradicional, basada en la observación directa por parte del personal. Este método no solo es intensivo en recursos humanos, sino que también presenta limitaciones de cobertura temporal y espacial, dificultando el seguimiento continuo de los animales.

Además, muchas explotaciones carecen de herramientas tecnológicas que permitan llevar un control estructurado de las condiciones de los animales y del entorno. Esto conlleva una falta de sistematización en la recogida de datos y la ausencia de registros históricos fiables, lo cual limita la capacidad para tomar decisiones informadas y aplicar estrategias preventivas o correctivas basadas en evidencia.

El cumplimiento de los controles oficiales en materia de bienestar animal exigidos por la normativa europea también plantea desafíos importantes. Las explotaciones deben remitir datos verificables sobre el estado de los animales, sus condiciones de alojamiento y la gestión de situaciones de riesgo. En ausencia de tecnologías de monitorización automatizada, estos procesos pueden volverse ineficientes y poco precisos.

Por último, el sector se ve afectado por nuevas exigencias ambientales, que buscan reducir el impacto ecológico de la actividad ganadera, en especial en lo relativo a la emisión de gases contaminante. La presión normativa en este sentido ha motivado la adopción de prácticas más sostenibles, como la mejora de la ventilación, la gestión eficiente de residuos y el uso de sensores ambientales [17].

Frente a todas estas problemáticas, surge una oportunidad clara: la integración de tecnologías basadas en inteligencia artificial y visión por computador. Estos sistemas permiten monitorizar de forma continua y objetiva el comportamiento de los animales, facilitando la detección temprana de anomalías, la reducción de costes operativos, la generación automática de informes y la optimización de la toma de decisiones a través del análisis de datos históricos.

3.1.2. Oportunidades de mercado y tendencias tecnológicas

En los últimos años, el sector ganadero ha comenzado una transformación significativa impulsada por la digitalización y la adopción de tecnologías avanzadas. Este fenómeno, conocido como ganadería 4.0, representa una oportunidad única para mejorar la eficiencia productiva, la sostenibilidad y el bienestar animal, al mismo tiempo que se reducen los costes operativos y se optimiza la toma de decisiones. A continuación, se exponen las principales tendencias tecnológicas y su proyección en el mercado.

- **Digitalización y sensorización de procesos productivos** [18]

Una de las líneas más prometedoras en esta transformación es la incorporación del Internet de las Cosas (IoT) en explotaciones agrícolas y ganaderas. Mediante sensores distribuidos en animales, infraestructuras o maquinaria, es posible monitorizar variables como la salud, temperatura corporal, posición o nivel de actividad de cada ejemplar. Esta monitorización en tiempo real no solo permite actuar de forma precoz ante enfermedades o comportamientos anómalos, sino que también facilita la automatización de tareas que anteriormente requerían intervención humana constante, como la alimentación.

También destacan los sistemas inteligentes de alimentación. Gracias a la recopilación y análisis de datos individuales, se pueden ajustar las dietas según las necesidades nutricionales específicas de cada animal, reduciendo el desperdicio.

En España, la ganadería 4.0 también está avanzando, aunque con un nivel de adopción desigual entre grandes explotaciones y pequeños productores.

- **Aplicaciones de IA, *Big Data* y análisis predictivo**

A su vez, la Inteligencia Artificial (IA) se está consolidando como una herramienta clave para el análisis de datos biométricos, la predicción de enfermedades, el seguimiento genético o incluso la selección reproductiva. Algoritmos de aprendizaje automático entrenados con datos históricos pueden detectar patrones que escapan a la observación humana, contribuyendo así a una ganadería de precisión más eficiente y basada en evidencia.

Asimismo, la integración de sistemas basados en *Big Data* facilita la toma de decisiones estratégicas mediante la visualización de informes y alertas personalizadas.

En España se están desplegando soluciones como collares con GPS, sensores fisiológicos y cercas virtuales para ganado, así como sistemas de visión por computadora para identificar y predecir estrés o enfermedades [19].

- **Retos actuales y consideraciones finales**

Pese al prometedor avance tecnológico, existen obstáculos que ralentizan su adopción, como el coste de los sistemas de monitorización y robótica, la falta de formación especializada entre los ganaderos o las limitaciones de conectividad en entornos rurales. No obstante, estas barreras pueden superarse con políticas de apoyo, formación técnica y desarrollo de soluciones adaptadas a las condiciones locales.

Aquí entran estrategias promovidas y financiadas por la Unión Europea que buscan que, mediante el uso de estas tecnologías, como llevo comentando a lo largo del proyecto, hacer la

ganadería más sostenible, eficiente y respetuosa con el bienestar animal[20] [21].

En definitiva, la Ganadería 4.0 no es solo una tendencia, sino una realidad emergente que ofrece oportunidades concretas de innovación y mejora en todos los niveles del proceso productivo. La integración de tecnologías como la inteligencia artificial, el IoT, el *Big Data*, la automatización y la visión por computadora marcará el rumbo de un sector que, aunque tradicional, se encuentra en plena evolución hacia una gestión más eficiente, sostenible y basada en datos [22] [23] [24].

3.2. Contexto científico-técnico

3.2.1. *Machine Learning*

El *Machine Learning* o Aprendizaje Automático es una rama de la Inteligencia Artificial que busca que los ordenadores aprendan de los datos y mejoren su desempeño en la realización de alguna tarea mediante la experiencia, y lo hagan de forma autónoma. Los ordenadores, mediante los algoritmos utilizados en esta disciplina de la Inteligencia Artificial, son capaces de detectar patrones en una cantidad muy extensa de datos, así como de realizar predicciones [25] [26].

Las tres principales áreas en las que se dividen los distintos algoritmos de *Machine Learning* son:

- **Aprendizaje supervisado.**
  Los algoritmos pertenecientes a este tipo de aprendizaje utilizan conjuntos de datos previamente etiquetados, es decir, para cada entrada se conoce cuál debería ser su salida. Durante el entrenamiento, el algoritmo analiza los pares entrada-salida e identifica patrones o relaciones entre ellos. El objetivo es que, una vez entrenado, sea capaz de generalizar su conocimiento y realizar predicciones precisas con datos nuevos. Algunos de los algoritmos más utilizados para este tipo de aprendizaje son Naive Bayes, regresión lineal, K-vecino más cercano, redes neuronales y bosque aleatorio [27].
  Los algoritmos anteriores tratan de resolver dos tipos de problemas:

  - **Clasificación:** trata de asignar una etiqueta a cada entrada de datos. El modelo aprende, a partir del conjunto de entrenamiento con datos ya etiquetados, los patrones que permiten distinguir entre clases.
  - **Regresión:** su objetivo es predecir un valor numérico continuo a partir de los datos de entrada. Busca modelar la relación entre una o varias variables independientes y una variable dependiente.

- **Aprendizaje no supervisado.**
  Los algoritmos de este tipo de aprendizaje trabajan con conjuntos de datos no etiquetados, es decir, no se dispone de información previa sobre su correspondiente salida. Durante

el entrenamiento, el algoritmo analiza los datos en busca de patrones, similitudes o estructuras internas con el objetivo de organizar o transformar los datos de forma útil [28]. Algunos de los algoritmos más comunes k-medias o la incrustación estocástica de vecinos distribuida [29].
Este tipo de aprendizaje se suele utilizar en tareas de:

- ***Clustering*:** consiste en dividir los datos en grupos o clústers según su similitud interna, sin que se conozca previamente a qué grupo pertenece cada elemento.
- **Reducción de la dimensionalidad:** busca simplificar la información manteniendo sus características esenciales, lo que mejora el rendimiento y visualización de los datos.

- **Aprendizaje por refuerzo.**
  El algoritmo o agente aprende a tomar decisiones mediante la interacción con un entorno. No dispone de datos etiquetados ni conoce de antemano la respuesta correcta, sino que descubre qué acciones les conducen a mejores resultados a través de prueba y error. Tras cada acción, el agente recibe una recompensa o penalización, lo que permite ajustar su comportamiento progresivamente [30].

3.2.2. Redes neuronales

Una red neuronal artificial (ANN) es un modelo computacional inspirado en la estructura del cerebro humano. Uno de sus principales objetivos es reconocer patrones y generalizar a partir de ejemplos, de forma similar a como lo hace la mente humana. Este enfoque resulta útil en tareas complejas como la clasificación de imágenes, el procesamiento del lenguaje natural o la predicción de resultados a partir de grandes volúmenes de datos. Las redes neuronales artificiales forman parte del campo del *Machine Learning*, habiendo cobrado especial relevancia con la aparición del *Deep Learning*.

Para comprender mejor cómo funcionan las redes neuronales artificiales, antes hay que entender cómo funciona una neurona, podemos revisar brevemente su contraparte biológica. Aunque se trata de un modelo simplificado, toma como referencia la estructura y el funcionamiento de las neuronas en el cerebro humano, lo que permite establecer paralelismos conceptuales entre ambos sistemas.

Como se puede observar en la Figura 3.1, las neuronas biológicas presentan 3 partes diferenciadas: dendritas, soma o cuerpo y axón. Las dendritas reciben señales de otras neuronas, el soma procesa la información y el axón transmite señales procesadas [31].

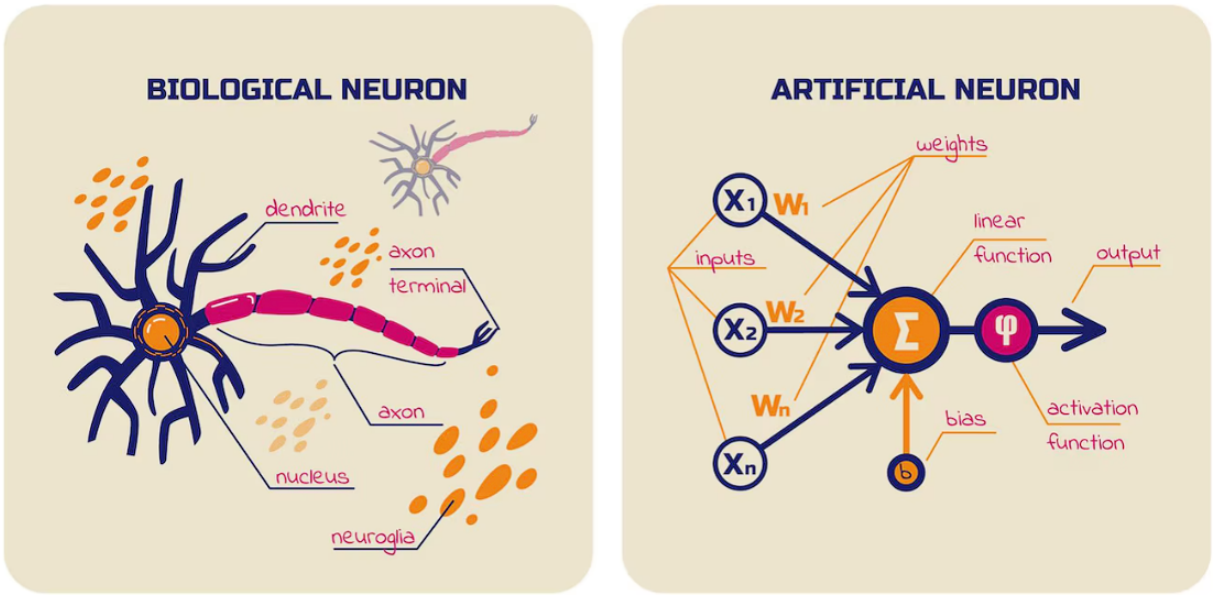


Figura 3.1: Estructura de una neurona biológica y artificial.

Del mismo modo, una neurona artificial está formada por un conjunto de entradas, cada una con un peso asociado, una función de propagación que combina dichas entradas, un sesgo (bias) que ajusta el resultado, y una función de activación que determina la salida final del modelo. En la Figura 3.1, se representa gráficamente esta comparación entre una neurona biológica y una artificial.

En este esquema, las entradas son comparables a las dendritas, ya que son las responsables de recibir la información del entorno o de otras neuronas. Cada entrada se multiplica por un peso, lo que permite ajustar la importancia relativa de cada una. Posteriormente, se realiza una suma ponderada de todas las entradas, a la que se añade un sesgo (b), y este resultado se introduce en una función de activación, que simula el comportamiento del axón, generando la salida de la neurona artificial [32].

Se pueden utilizar distintas funciones de activación, algunas de ellas son [33]:

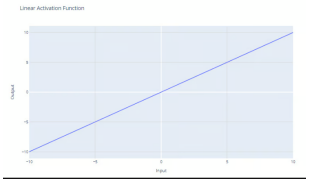
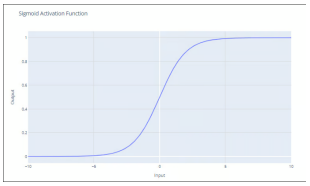
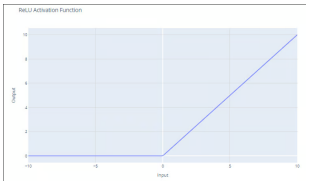
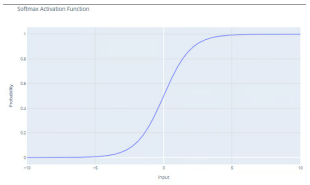
Tipo	Función	Gráfica
Lineal	$f(x) = x$	
Sigmoide	$f(x) = \frac{1}{1+e^{-x}}$	
ReLU	$f(x) = \text{máx}(0, x)$	
Softmax	$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$	

Tabla 3.1: Funciones de activación más representativas

Una vez explicado cómo funcionan las neuronas de forma independiente, podemos definir la red neuronal como un conjunto de capas de neuronas artificiales conectadas entre sí. No todas las neuronas van a realizar la misma labor, por ello se pueden distinguir 3 capas distintas, como se observa en la Figura 3.2 [34].

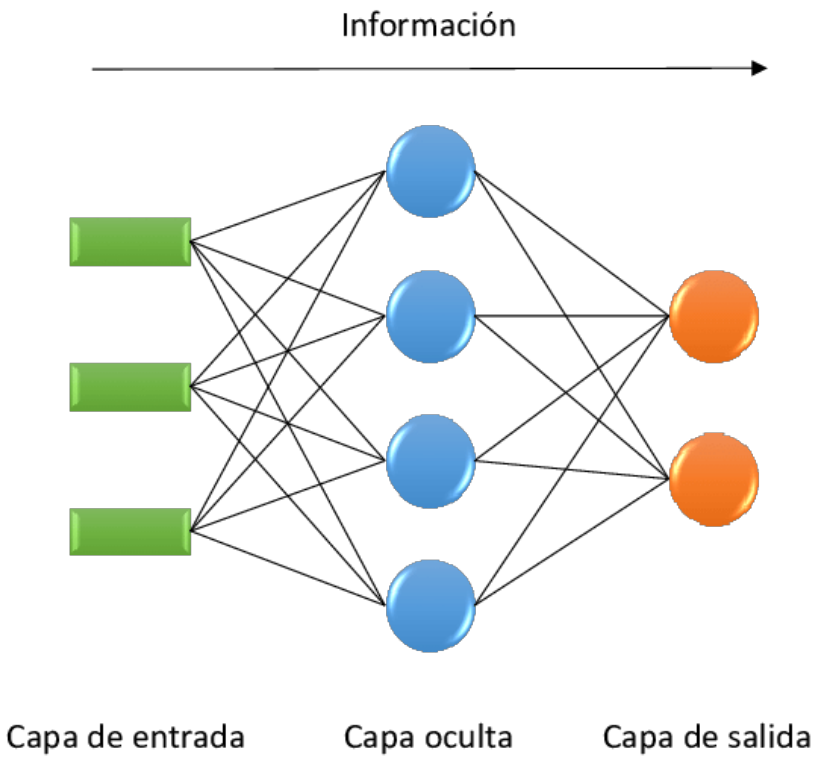


Figura 3.2: Red neuronal por capas.

En primer lugar, la capa de entrada es la encargada de recibir los datos del entorno o del conjunto de entrenamiento. Cada neurona de esta capa representa una característica o variable del problema que se quiere resolver, por lo que su tamaño depende directamente del número de atributos de entrada.

A continuación, la información se transmite a una o varias capas ocultas hacia adelante (proceso conocido como *forward propagation*), que son responsables del procesamiento interno. En ellas, cada neurona realiza operaciones sobre los datos recibidos aplicando los pesos, el sesgo y la función de activación correspondiente.

Finalmente, la capa de salida genera el resultado final del modelo. La cantidad de neuronas en esta capa depende del tipo de problema a resolver.

En cada una de estas capas, las neuronas realizan una operación sobre las entradas que reciben: se aplica una combinación lineal de los valores de entrada ponderados por sus respectivos pesos, se añade un sesgo y, finalmente, el resultado pasa por una función de activación.

Este conjunto de operaciones transforma progresivamente los datos, permitiendo a la red modelar relaciones complejas. Las salidas generadas por la red son entonces comparadas con las salidas reales del conjunto de entrenamiento. Esta comparación permite calcular un error.

El aprendizaje se realiza ajustando los pesos y sesgos internos de la red para reducir este error. Para ello, se utiliza un algoritmo de optimización, como el descenso del gradiente [35], que actualiza los parámetros de la red en la dirección en la que el error disminuye. Este ajuste se realiza gracias al algoritmo de retropropagación [36], que permite distribuir el error desde la capa de salida hacia las capas anteriores, asignando una contribución específica del error a cada conexión de la red.

Este proceso de entrenamiento se repite durante múltiples épocas [37]], utilizando una gran cantidad de ejemplos. El objetivo es que la red no solo aprenda a producir salidas correctas con los datos conocidos, sino que también sea capaz de generalizar su conocimiento para obtener buenos resultados con datos nuevos. Para ello, se necesita un conjunto de entrenamiento representativo y diverso, de forma que el modelo aprenda las características relevantes del problema sin limitarse a memorizar los ejemplos concretos, evitando así el sobreajuste [38] [39].

3.2.3. *Deep Learning*

El *Deep Learning* [40] o aprendizaje profundo es un subcampo del aprendizaje automático (*Machine Learning*) que permite el aprendizaje de patrones complejos en conjuntos de datos utilizando redes neuronales con múltiples capas. A diferencia del *Machine Learning* tradicional, donde muchas veces es necesario diseñar manualmente las características relevantes, el *Deep Learning* automatiza este proceso, extrayendo representaciones útiles a través de varias capas de procesamiento. Esto permite crear modelos que, con suficiente capacidad computacional y datos, pueden superar el rendimiento de otros enfoques en tareas complejas como visión por computador, reconocimiento del habla o procesamiento del lenguaje natural.

En las redes neuronales profundas, la arquitectura se basa en una jerarquía de capas. Las primeras capas aprenden características básicas del conjunto de datos (por ejemplo, bordes o líneas en imágenes), mientras que las capas más profundas combinan esa información para formar representaciones cada vez más abstractas. Esta estructura jerárquica es la que permite al modelo detectar patrones complejos y tomar decisiones más precisas. El conocimiento se va refinando progresivamente a medida que los datos se propagan por la red.

El *Deep Learning* tiene un impacto destacado en numerosos sectores. En visión computacional se utiliza para tareas como clasificación de imágenes o segmentación de objetos. También tiene aplicaciones importantes en el reconocimiento de voz, el procesamiento del lenguaje natural, la medicina (por ejemplo, en diagnóstico automático), la seguridad o el comercio electrónico. Su capacidad de adaptación y su alto rendimiento lo convierten en una herramienta clave en el desarrollo de soluciones inteligentes [41].

Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal artificial especialmente eficaz para tratar datos con una estructura espacial, como las imágenes. Se inspiran en la forma en que la corteza visual del cerebro humano procesa la información visual. Estas redes utilizan filtros que extraen características locales de las imágenes, permitiendo detectar desde patrones simples hasta estructuras más complejas como siluetas u objetos completos.

Una CNN suele estar compuesta por las siguientes capas:

- **Convolución:** se aplican filtros (o *kernels*) sobre la imagen para generar mapas de características. Estos filtros pueden configurarse en tamaño, cantidad y paso (stride).
- **Activación (ReLU):** introduce no linealidad eliminando los valores negativos y conservando los positivos, como vimos con su función en la Tabla 3.1.
- ***Pooling*:** reduce la dimensionalidad del mapa de características. El más común es el *Max Pooling*, que selecciona el valor máximo dentro de una ventana.
- ***Fully Connected*:** transforma las matrices obtenidas en un vector, conectando todas las neuronas como en una red densa.
- ***Softmax*:** convierte el vector de salida en una distribución de probabilidad, útil para clasificación.

Como se observa en la Figura 3.3, estas capas se organizan de forma secuencial y jerárquica, permitiendo que la red extraiga progresivamente características cada vez más complejas de la imagen original.

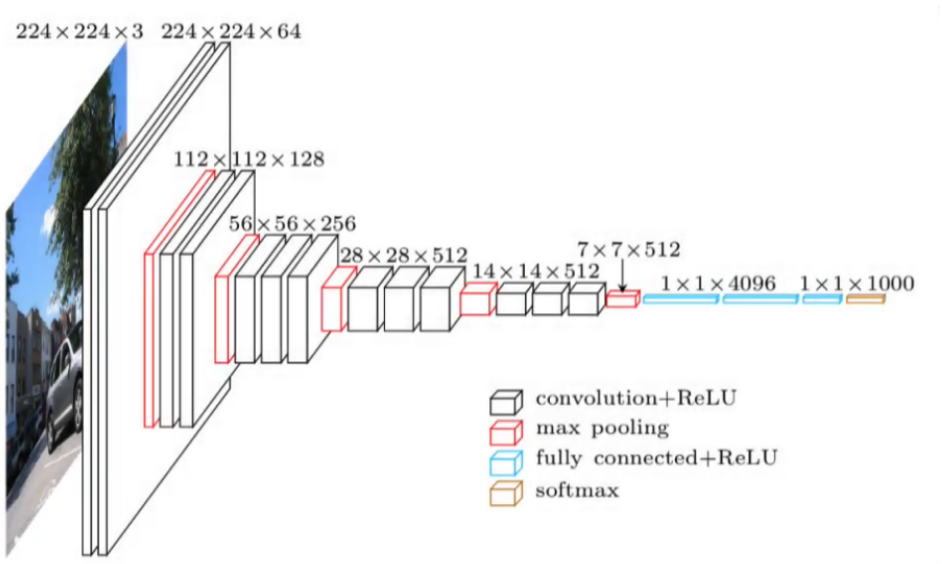


Figura 3.3: Ejemplo estructura red neuronal convolucional

Las CNN operan de forma jerárquica: las primeras capas detectan elementos básicos (bordes, líneas), y conforme se avanza en la red, se combinan esas características para identificar formas más complejas como contornos, objetos o patrones completos. Esta estructura hace que las CNN sean especialmente potentes en visión por computador.

Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes (RNN) son especialmente útiles para el tratamiento de datos secuenciales, como texto, audio o vídeo. A diferencia de las redes neuronales tradicionales (como las CNN), que asumen que las entradas y salidas son independientes entre sí, las RNN utilizan recurrencia para tener en cuenta la dependencia temporal entre elementos de una secuencia. Esto significa que la salida de una neurona en una iteración puede influir en la entrada de esa misma neurona en la siguiente (ver Figura 3.4).

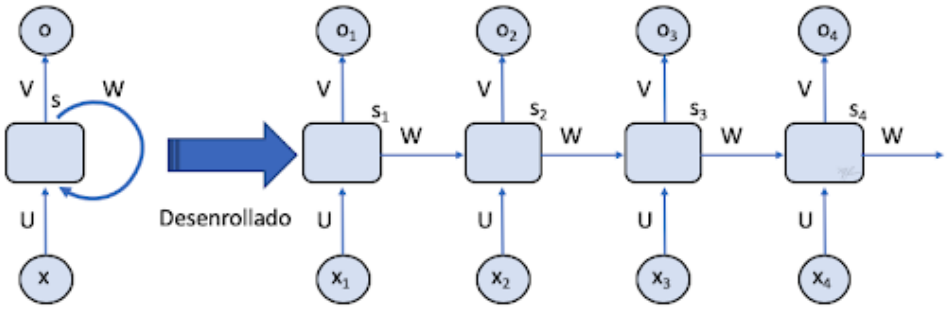


Figura 3.4: Ejemplo estructura red neuronal recurrente.

Las RNN permiten abordar tareas donde el orden y la dependencia entre los datos son esenciales. Por ejemplo, para comprender una frase no basta con analizar palabra por palabra, sino que es necesario considerar el contexto global. Del mismo modo, en vídeo, la información relevante puede estar repartida en distintos fotogramas, por lo que se requiere un modelo que capture estas correlaciones temporales. Por eso, se suelen combinar CNN (para extraer información de cada imagen) con RNN (para analizar la secuencia de imágenes).

Las RNN permiten una gran flexibilidad en las formas de entrada y salida, adaptándose a diferentes tipos de tareas. Existen cuatro configuraciones principales:

- **Uno a uno:** modelo tradicional de red neuronal con una sola entrada y una sola salida. Ejemplo: clasificación de imágenes simples.
- **Uno a muchos:** una entrada y múltiples salidas. Ejemplo: generación de texto o pies de foto a partir de una imagen.
- **Muchos a uno:** varias entradas y una sola salida. Ejemplo: análisis de sentimientos, donde una secuencia de texto produce una única etiqueta.

- **Muchos a muchos:** tanto entradas como salidas son secuencias. Ejemplo: traducción automática de texto.

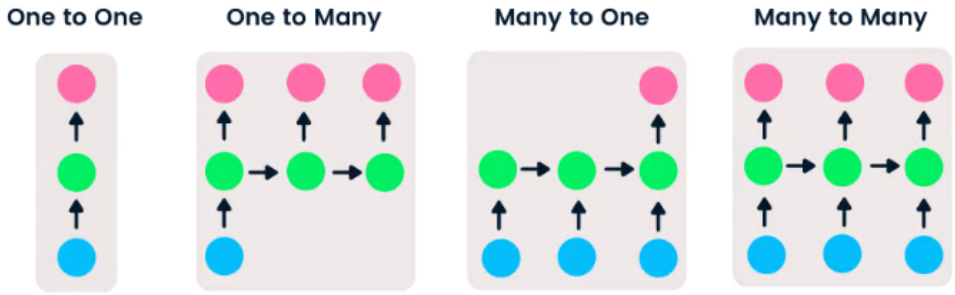


Figura 3.5: Configuraciones de entradas y salidas en redes neuronales recurrentes.

Como se puede observar en la Figura 3.5, estas configuraciones permiten adaptar las RNN a un amplio rango de aplicaciones secuenciales.

Las RNN tradicionales sufren de dos problemas principales durante el entrenamiento:

- **Desvanecimiento del gradiente:** el gradiente se vuelve tan pequeño que la red deja de aprender.
- **Explosión del gradiente:** los gradientes se hacen tan grandes que los pesos se vuelven inestables, lo que dificulta la convergencia del modelo.

Estos problemas se presentan especialmente cuando se intenta modelar dependencias a largo plazo en secuencias largas.

Para superar esas limitaciones, se desarrollaron variantes como las *Long Short-Term Memory* (LSTM) y las *Gated Recurrent Units* (GRU). Estas redes incorporan mecanismos de compuertas que controlan el flujo de información y ayudan a mantener memoria a largo plazo [42].

La tabla 3.2 muestra una comparativa simplificada entre redes neuronales convolucionales y redes neuronales recurrentes.

Característica	CNN	RNN
Tipo de datos	Imágenes	Temporales/secuenciales
Arquitectura	Prealimentada	Recurrente
Entradas/salidas	Fijas	Flexibles
Retropropagación	Estándar	Backpropagation Through Time
Aplicaciones	Visión por computadora	Audio, texto, vídeo

Tabla 3.2: Comparativa entre redes neuronales convolucionales y recurrentes.

En tareas de análisis de vídeo, el uso de redes convolucionales (CNN) permite extraer características relevantes de cada fotograma de manera individual. No obstante, cuando el objetivo es detectar patrones temporales o secuencias de comportamiento que se desarrollan a lo largo del tiempo (como ocurre en aplicaciones de vigilancia, seguimiento o análisis en tiempo real), es habitual combinar estas arquitecturas con modelos que capturen dependencias temporales. En este contexto, las redes neuronales recurrentes (RNN), especialmente sus variantes como LSTM o GRU, resultan especialmente útiles al permitir modelar relaciones entre elementos sucesivos de una secuencia.

3.3. Análisis técnico

En este apartado se presenta una revisión del estado actual de las técnicas, herramientas y aplicaciones relacionadas con el análisis del comportamiento animal mediante *Deep Learning* y visión por computadora. Se abordan los métodos más empleados para la detección y seguimiento de objetos, con especial atención a su aplicación en el estudio del comportamiento animal, tanto en vídeo como en escenarios en tiempo real.

Asimismo, se incluyen referencias a herramientas de etiquetado de datos, algoritmos de seguimiento, *frameworks* y bibliotecas utilizadas en el desarrollo de estos sistemas, así como una visión general de los avances recientes y las líneas de investigación abiertas. Esta revisión permite contextualizar la solución propuesta en el presente trabajo dentro de las tendencias actuales del ámbito científico-tecnológico.

3.3.1. Herramientas de etiquetado de datos

Empezaremos explicando la importancia de las herramientas de etiquetado [43] [44] y algunas de las más utilizadas.

Uno de los pasos más fundamentales en el desarrollo de sistemas de aprendizaje automático supervisado es la preparación de datos anotados o etiquetados. Este proceso consiste en asignar manual o automáticamente información contextual o categórica a datos brutos, como imágenes o vídeos, con el fin de que los modelos puedan aprender a partir de ellos. En el caso del aprendizaje supervisado, las etiquetas actúan como guía directa durante la fase de entrenamiento, permitiendo que el modelo relacione una entrada determinada con una salida esperada.

El etiquetado de datos es especialmente relevante en tareas de visión por computadora, como la detección de objetos o el seguimiento de individuos a lo largo de una secuencia. Etiquetar correctamente qué objeto aparece, su ubicación y, en algunos casos, su identidad o comportamiento asociado, es clave para lograr modelos precisos y funcionales. Sin una base de datos anotada adecuadamente, incluso el modelo más sofisticado perdería su capacidad predic-

tiva o cometería errores sistemáticos. La calidad de las etiquetas, por tanto, incide directamente en la robustez y fiabilidad del sistema resultante.

No obstante, este proceso también implica ciertos retos: por un lado, el etiquetado manual puede requerir un esfuerzo considerable, sobre todo cuando se trabaja con miles de imágenes o secuencias de vídeo. Por otro, la subjetividad o falta de consistencia entre anotadores puede introducir sesgos o errores en los datos. Para paliar estas dificultades, han surgido herramientas especializadas que permiten facilitar, acelerar y estandarizar este proceso.

Entre las herramientas más utilizadas se encuentra CVAT (*Computer Vision Annotation Tool*), una plataforma de código abierto desarrollada por Intel que permite la anotación de imágenes y vídeos para tareas de segmentación, detección y clasificación. CVAT ofrece una interfaz interactiva, anotación fotograma a fotograma, acciones masivas y compatibilidad con múltiples formatos, lo que la convierte en una opción muy extendida tanto en proyectos académicos como industriales.

Otra herramienta popular es LabelImg, una aplicación ligera diseñada para crear fácilmente archivos de anotación en formato XML o YOLO. Se centra principalmente en tareas de detección de objetos en imágenes individuales, y su sencillez la hace ideal para proyectos de pequeña o mediana escala.

Estas herramientas, entre otras, han facilitado enormemente la generación de conjuntos de datos etiquetados de calidad, acelerando así el desarrollo de sistemas de visión artificial en diferentes campos, incluido el análisis del comportamiento animal.

3.3.2. Modelos de detección de objetos

La detección de objetos [45] [46] [47] ha sido uno de los campos más desarrollados dentro de la visión por computadora, con múltiples aproximaciones propuestas a lo largo del tiempo. Consiste en localizar y clasificar instancias específicas dentro de una imagen o fotograma mediante cajas delimitadoras (bounding boxes). La capacidad para detectar objetos de forma precisa y eficiente es crucial para desarrollar sistemas automatizados que interactúen de manera efectiva con su entorno visual.

La detección de objetos puede seguir dos paradigmas principales: modelos en dos etapas (basados en propuestas de regiones) y modelos en una etapa (detección directa sobre la imagen completa). Ambos enfoques se benefician del uso de métricas como Intersection over Union (IoU) y Mean Average Precision (mAP) para evaluar la calidad de las predicciones.

Entre las primeras arquitecturas CNN destacadas se encuentran los modelos R-CNN (Region-based CNN). Este enfoque realiza propuestas de regiones, deformando cada región a un tamaño

uniforme y pasándola por redes separadas para extraer características y clasificarlas. Pese a su precisión, su elevado coste computacional dio paso a variantes más rápidas:

- **Fast R-CNN:** optimiza R-CNN extrayendo características de toda la imagen con una sola pasada de la CNN, y luego aplica *Region of Interest Pooling* para clasificar cada región propuesta. Esto reduce el tiempo de inferencia, aunque aún depende de un algoritmo externo para la generación de regiones.
- **Faster R-CNN:** reemplaza la búsqueda selectiva con una red entrenable que predice regiones directamente desde el mapa de características. Esta arquitectura es un estándar en detección de alta precisión.
- **Mask R-CNN:** extiende Faster R-CNN añadiendo una tercera rama para realizar segmentación, permitiendo predecir la máscara de cada objeto a nivel de píxel, no solo su ubicación aproximada.

Posteriormente surgieron los modelos de detección de una sola etapa, como YOLO (*You Only Look Once*) y SSD (*Single Shot Detector*). Estos condensan las tareas de localización y clasificación en una única red, lo que permite predicciones mucho más rápidas, haciendo viable su uso en contextos de tiempo real.

- **YOLO [48]:** la imagen se divide en una cuadrícula fija, y cada celda predice coordenadas, clase y confianza para varios bounding boxes. Para hacerlo se apoya de CNNs. Este enfoque elimina la necesidad de generar propuestas de regiones, lo que permite realizar detección en tiempo real.
  A lo largo del tiempo han ido saliendo versiones que van mejorándolo en todos los aspectos. Las versiones YOLOv1 a YOLOv3 introdujeron mejoras en arquitectura y resolución multiescala. YOLOv4 y YOLOv5 supusieron avances significativos en términos de precisión y facilidad de implementación. Más recientemente, versiones como YOLOv8, YOLOv10 y YOLOv11 han incorporado nuevas optimizaciones estructurales, mayor eficiencia en dispositivos de baja capacidad y adaptación a tareas más complejas como segmentación o detección por instancia. Así como existen diferentes versiones de YOLO, también hay distintas variantes por tamaño y capacidad de carga computacional, diseñadas para adaptarse a distintos escenarios de uso. Estas variantes se identifican habitualmente con sufijos como n (*nano*), s (*small*), m (*medium*), l (*large*) y x (*extra large*). Cada una de ellas ofrece un equilibrio distinto entre velocidad de inferencia y precisión: las versiones más pequeñas (n y s) priorizan la rapidez y el bajo consumo de recursos, lo que las hace idóneas para dispositivos con capacidades limitadas o aplicaciones en tiempo real; mientras que las más grandes (l y x) proporcionan mayor precisión a costa de un mayor tiempo de procesamiento y uso de memoria.
- **SSD:** utiliza múltiples capas de una CNN para detectar objetos a diferentes escalas. Introduce anclas (prior boxes) con múltiples proporciones y escalas sobre cada celda del

mapa de características. Es especialmente eficaz para detectar objetos grandes y pequeños en una sola pasada, logrando un balance entre precisión y velocidad.

3.3.3. Algoritmos de seguimiento de objetos

La detección de objetos [49] [50] por fotograma es solo el primer paso para analizar el comportamiento a lo largo del tiempo. Para ello, se requiere una fase de seguimiento que permita asociar las detecciones individuales entre fotogramas consecutivos, identificando cada instancia en el transcurso del tiempo.

Los algoritmos de seguimiento pueden clasificarse en dos grandes grupos: los métodos tradicionales basados en visión artificial y los enfoques modernos basados en aprendizaje profundo.

Métodos tradicionales

Estos algoritmos suelen operar directamente sobre las características básicas de la imagen. Son rápidos y eficientes, aunque menos robustos ante oclusiones, deformaciones o cambios de apariencia del objeto. Algunos de ellos son:

- **KCF (*Kernelized Correlation Filters*)**: Utiliza filtros de correlación en el dominio de la frecuencia. Es rápido y eficiente en escenas estables, pero menos robusto ante cambios bruscos de apariencia.
- **MOSSE (*Minimum Output Sum of Squared Error*)**: Extremadamente rápido y apto para tiempo real. A cambio, su precisión es limitada en escenarios complejos.
- **CSRT (*Discriminative Correlation Filter with Channel and Spatial Reliability*)**: Mejora a KCF incorporando confiabilidad espacial y de canales. Más preciso, pero computacionalmente más costoso.
- **TLD (*Tracking-Learning-Detection*)**: Combina detección, aprendizaje y seguimiento. Puede recuperar el objeto tras una pérdida, pero es más lento y complejo.

Filtros probabilísticos

Estos métodos predicen la ubicación futura del objeto y corrigen la predicción usando observaciones:

- **Filtro de Kalman**: Ideal para movimientos lineales o ligeramente ruidosos. Se combina frecuentemente con detectores modernos (como en SORT o Deep SORT).
- **Filtro de partículas**: Variante más flexible del anterior, capaz de manejar trayectorias no lineales, aunque a mayor coste computacional.

Seguimiento basado en aprendizaje profundo [51]

Aprovechan redes neuronales para extraer características visuales de alto nivel. Son más precisos y robustos frente a oclusiones, deformaciones o variaciones de apariencia, aunque requieren mayor potencia de cálculo. Algunos algoritmos son:

- **GOTURN**: Uno de los primeros rastreadores basados en CNN. Rápido, pero dependiente de la GPU y sin actualización online del modelo.
- **MDNet**: Utiliza redes convolucionales entrenadas en múltiples dominios, adaptando la representación a cada nuevo objeto en seguimiento. Alta precisión, pero lento.
- **DeepSORT**: Extiende el algoritmo SORT al añadir una red de características de apariencia, mejorando la identificación de objetos similares o tras oclusiones. Muy usado junto a detectores como YOLO o Faster R-CNN.
- **ByteTrack**: Mejora la asociación al conservar detecciones de baja confianza. Supera a Deep SORT en precisión sin perder velocidad.
- **BoT-SORT**: evolución del algoritmo ByteTrack orientada a mejorar la precisión del seguimiento en entornos dinámicos y complejos.
- **SMILEtrack**: rastreador basado en ByteTrack que mejora la asociación entre objetos similares y mantiene la coherencia de identidad en presencia de oclusiones, reapariciones y trayectorias complejas.

3.4. Librerías más utilizadas

En los sistemas modernos de visión artificial, la detección y el seguimiento de objetos constituyen dos etapas complementarias que, en muchos casos, se integran en un único flujo de trabajo. Para facilitar el desarrollo, entrenamiento y despliegue de estos sistemas, existe un conjunto de librerías especializadas ampliamente utilizadas tanto en investigación como en aplicaciones industriales. A continuación, se describen las más relevantes:

- **OpenCV:** es una biblioteca robusta para visión por computador que permite capturar y procesar imágenes y vídeos en tiempo real, aplicar transformaciones como cambio de tamaño, rotación o conversión de color, y realizar preprocesamientos como corrección de iluminación y eliminación de ruido mediante diversos filtros. Además, permite trabajar con modelos de *Deep Learning* integrando *frameworks* como TensorFlow o PyTorch, lo que posibilita tareas avanzadas como clasificación, detección y segmentación semántica de objetos. OpenCV incluye herramientas para preprocesar los datos de entrada, realizar inferencias optimizadas con aceleración por hardware, aplicar post-procesamientos, y visualizar resultados con anotaciones gráficas [52].

- **Norfair:** es una biblioteca liviana y flexible para el seguimiento de objetos en video que se adapta fácilmente a cualquier sistema de detección. Funciona prediciendo el movimiento de objetos y asociando nuevas detecciones con trayectorias activas mediante algoritmos personalizables como la distancia euclidiana o funciones propias definidas por el usuario. Su diseño modular permite integrarse con cualquier detector moderno y seguir objetos en 2D o 3D, incluso con entradas ruidosas o detecciones irregulares. Norfair no requiere redes neuronales para operar, por lo que es rápido y fácil de usar, pero puede complementarse con modelos complejos si se desea. Es ideal para proyectos en tiempo real, análisis de video en deportes, drones, robótica y sistemas de vigilancia, permitiendo dibujar trayectorias visuales y manejar múltiples objetos con un código mínimo [53].
- **Detectron2:** es una poderosa biblioteca de visión por computador desarrollada por Meta AI que permite entrenar, personalizar e implementar modelos de detección de objetos y segmentación de imágenes con precisión de última generación. Está basada en PyTorch y ofrece soporte para tareas como detección de instancias, segmentación semántica, segmentación de instancias y keypoint detection, todo con arquitecturas avanzadas como Faster R-CNN, Mask R-CNN, RetinaNet y más. Detectron2 está diseñado para ser altamente modular, lo que facilita ajustar hiperparámetros, integrar datasets personalizados, hacer fine-tuning de modelos preentrenados y realizar inferencia en imágenes o video. Además, permite trabajar con aceleración por GPU y escalar experimentos desde prototipos hasta producción, siendo ampliamente utilizado en investigación, robótica, visión industrial y sistemas autónomos [54].
- **Ultralytics YOLO:** es una suite completa de modelos en tiempo real para detección, segmentación, clasificación, estimación de poses, enmarcado orientado y seguimiento diseñada para ofrecer un equilibrio óptimo entre velocidad, precisión y eficiencia de tamaño. Permite entrenar desde cero o continuar el entrenamiento sobre modelos preentrenados. Soporta múltiples fuentes (imágenes, videos, transmisiones) con procesamiento por lotes y en streaming, optimizado para alcanzar alta velocidad sin sacrificar precisión [55].
- **PyTorch:** es un *framework* de aprendizaje profundo de código abierto que destaca por su flexibilidad y facilidad para la experimentación gracias a su soporte para grafos computacionales dinámicos. Su diseño integrado con Python facilita la creación y entrenamiento de modelos complejos de manera ágil, desde la fase de prototipado hasta el despliegue en entornos productivos. PyTorch soporta aplicaciones, como visión por computadora, procesamiento de lenguaje natural y aprendizaje por refuerzo [56].

3.5. Aplicaciones en el mercado y en desarrollo

La visión por computadora, en combinación con el aprendizaje profundo, ha dejado de ser una tecnología experimental para convertirse en una herramienta clave en sectores productivos y sociales de alto impacto. Más allá del desarrollo técnico, algunos de los usos más relevantes y consolidados en el mercado se encuentran ya operativos, integrados en sistemas reales que funcionan y evolucionan de forma continua en entornos exigentes y diversos.

Este apartado recoge cuatro ejemplos concretos de aplicaciones reales, donde las técnicas de detección, seguimiento y análisis visual están resolviendo problemas específicos en ámbitos tan variados como la movilidad autónoma, el comercio inteligente, la monitorización de especies, o la conservación ambiental. Se trata de proyectos representativos por su adopción práctica, impacto social o ambiental, y su consolidación en escenarios reales de uso.

1. **Tesla Autopilot** [57] [58] [59]
   Tesla Autopilot es un sistema avanzado de asistencia a la conducción desarrollado por Tesla, Inc., que utiliza inteligencia artificial para interpretar el entorno del vehículo y ejecutar maniobras de forma semiautónoma. Introducido inicialmente en 2014, Autopilot ha ido evolucionando, como la navegación automática en carretera, el cambio de carril asistido, el aparcamiento autónomo y el reconocimiento de señales de tráfico, entre otras funciones.

   El sistema utiliza ocho cámaras exteriores para ofrecer una cobertura visual completa de 360°, eliminando progresivamente sensores como el radar (desde 2021) y los ultrasonidos (desde 2022), conforme a su transición hacia un enfoque Visual. Las imágenes captadas son procesadas mediante redes neuronales profundas, principalmente CNNs y, más recientemente, modelos *end-to-end* que combinan técnicas basadas en *transformers* y redes recurrentes para percepción, predicción y planificación del movimiento.

   Los modelos se entrenan con datos reales provenientes de una gran parte de la flota de vehículos Tesla, lo que permite captar millones de situaciones de conducción distintas en entornos urbanos, rurales y autopistas. Este proceso se ejecuta utilizando la infraestructura propia de computación de Tesla, entre la que destaca su supercomputadora Dojo, optimizada para entrenar redes neuronales a gran escala.

   A partir de esta base, Tesla ha desarrollado la opción *Full Self-Driving* (FSD), que amplía las funciones del Autopilot al entorno urbano. Aunque el sistema todavía requiere supervisión activa del conductor, representa uno de los enfoques más avanzados hacia la conducción autónoma total basada en aprendizaje profundo.

2. **Amazon Go** [60] [61] [62] [63]
   Amazon Go es una iniciativa comercial desarrollada por Amazon que representa uno de los avances más notorios en la automatización del comercio minorista mediante el uso combinado de visión por computadora, aprendizaje profundo y sensores integrados. Inaugurada al público en enero de 2018 en Seattle, esta tienda introduce el concepto de compra sin necesidad de pasar por caja, gracias a su sistema de cobro automático denominado *Just Walk Out*.

   El funcionamiento de Amazon Go se basa en un entorno altamente sensorizado, donde múltiples cámaras, sensores de movimiento y algoritmos de inteligencia artificial trabajan de forma conjunta para detectar qué productos toma o devuelve cada cliente. Este sistema, apoyado por redes neuronales convolucionales (CNNs) entrenadas para reconocimiento de objetos y seguimiento de personas en entornos dinámicos, permite registrar automáticamente los artículos seleccionados y cargar su coste directamente a la cuenta del cliente tras abandonar la tienda.

La tecnología de Amazon Go integra además procesamiento de datos en tiempo real, aprendizaje profundo para la reidentificación de usuarios y fusión de datos multifuente. Todo ello permite minimizar errores de seguimiento incluso cuando varios usuarios interactúan simultáneamente con estanterías compartidas. No obstante, esta complejidad tecnológica conlleva una elevada inversión en infraestructura, así como desafíos logísticos vinculados al control de inventario y reabastecimiento automático.

En la actualidad, aunque Amazon ha ralentizado la expansión de tiendas Go propias por cuestiones de escalabilidad y eficiencia operativa, la tecnología Just Walk Out se encuentra implementada en contextos como aeropuertos (por ejemplo, en Newark y Heathrow), estadios, campus universitarios, hospitales y tiendas corporativas, donde el flujo de clientes y la variedad limitada de productos favorecen su viabilidad.

Asimismo, Amazon ha comenzado a sustituir progresivamente *Just Walk Out* por soluciones más ligeras, como los carros inteligentes *Dash Cart*, en tiendas Amazon Fresh. Esta transición busca optimizar el coste operativo sin abandonar la experiencia sin caja que caracteriza su propuesta.

3. **Smart Parks** [64]
Smart Parks es una plataforma tecnológica avanzada destinada a la conservación ambiental y al monitoreo de la biodiversidad, que integra múltiples tecnologías de sensores, Internet de las cosas (IoT), y análisis de datos para proporcionar un sistema de gestión inteligente de áreas protegidas y parques naturales.

El sistema combina dispositivos físicos como cámaras inteligentes, collares GPS para fauna, sensores ambientales, y estaciones base con conectividad inalámbrica para recopilar datos en tiempo real sobre la ubicación, comportamiento y estado de los animales, así como las condiciones ambientales del entorno. Esta infraestructura habilita la vigilancia continua y no invasiva de la fauna silvestre, facilitando la detección temprana de amenazas como la caza furtiva o cambios críticos en los ecosistemas.

Para el procesamiento y análisis de la gran cantidad de datos generados, Smart Parks emplea algoritmos avanzados de aprendizaje automático y visión por computadora, incluyendo técnicas de *Deep Learning* para la detección automática de especies y el seguimiento individual de animales en imágenes y videos. Estas técnicas permiten una clasificación precisa, así como la evaluación del comportamiento animal en su hábitat natural, sin necesidad de intervención humana constante.

La plataforma transmite datos desde ubicaciones remotas a servidores centrales, donde se analizan. Esto posibilita la generación de informes en tiempo real, la gestión de alertas y la toma de decisiones informadas para la conservación y el manejo sostenible de los recursos naturales.

Smart Parks representa un avance significativo en la aplicación de tecnologías emergentes para la protección de la biodiversidad, al ofrecer un sistema integrado y escalable que optimiza el monitoreo ecológico, reduce costos operativos y mejora la efectividad en la gestión ambiental.

4. **Seguimiento y monitorización de insectos** [65]
   Este sistema de monitoreo y seguimiento de insectos en tiempo real es una solución innovadora que integra hardware accesible en el mercado con algoritmos de aprendizaje profundo para la detección, clasificación y rastreo individualizado de insectos vivos. Desarrollado para abordar la necesidad crítica de datos automatizados sobre la dinámica poblacional de insectos polinizadores, este método automatiza el proceso tradicionalmente manual y laborioso de identificación y conteo de insectos.

   El sistema emplea una cámara web para capturar imágenes de manera continua a baja frecuencia de fotogramas. Estas imágenes son procesadas en tiempo real mediante una red neuronal convolucional basada en el *framework* YOLOv3, optimizada para la detección rápida y precisa de insectos en escenarios naturales. El modelo fue entrenado con un conjunto de datos etiquetados que abarca múltiples especies y condiciones ambientales, utilizando técnicas de aumento de datos para mejorar la generalización.

   Para garantizar la integridad del conteo, el sistema incorpora un algoritmo de seguimiento múltiple que asigna detecciones consecutivas a individuos únicos, evitando así la sobreestimación de abundancia causada por múltiples apariciones del mismo insecto.

   El procesamiento de datos se divide en dos fases: una local, que incluye la captura y detección inicial en el dispositivo, y otra remota, donde se realiza el filtrado de falsos positivos y el seguimiento detallado de los individuos mediante análisis secuencial de las detecciones.

   Este enfoque permite realizar monitoreos continuos durante toda la temporada de crecimiento, proporcionando datos precisos sobre la abundancia, diversidad y comportamiento temporal de los insectos. Su implementación con componentes comerciales de bajo costo y código abierto facilita la replicabilidad y escalabilidad del sistema, posicionándolo como una herramienta prometedora para la investigación ecológica y la conservación basada en tecnologías de visión por computadora y *Deep Learning*.

5. **Aportaciones y diferenciación del sistema propuesto**

   Si bien los ejemplos anteriores muestran la versatilidad y consolidación de la visión por computadora en sectores como la movilidad, el comercio minorista, la conservación de la fauna silvestre o la monitorización de insectos, el sistema desarrollado en este trabajo se diferencia en varios aspectos clave:

   - **Enfoque específico en bienestar animal de granja:** mientras que la mayoría de soluciones actuales se orientan a entornos urbanos, comerciales o de conservación de fauna salvaje, la propuesta de este trabajo aborda un problema aún poco explorado: la monitorización automática del comportamiento de cerdos en instalaciones de producción. Este campo tiene un gran potencial de impacto tanto económico como ético, al facilitar la detección temprana de indicadores de bienestar animal.
   - **Simplicidad y bajo coste de infraestructura:** a diferencia de sistemas como Amazon Go o Smart Parks, que requieren entornos sensorizados complejos y una infra-

estructura costosa, el sistema propuesto funciona únicamente con cámaras de vídeo convencionales y un modelo de detección y seguimiento basado en aprendizaje profundo (YOLOv10n + ByteTrack). Esto facilita su adopción en granjas con recursos limitados, sin necesidad de grandes inversiones.

- **Orientación a métricas de comportamiento individual:** al combinar detección, seguimiento y registro de alertas, el sistema no solo identifica animales y posturas, sino que genera métricas longitudinales por individuo. Este enfoque se acerca a lo que hacen los sistemas de Smart Parks o de monitorización de insectos, pero adaptado a un contexto cerrado, donde los individuos tienen interacciones sociales constantes y su comportamiento es un indicador crítico de salud y productividad.
- **Escalabilidad y aplicabilidad inmediata:** la propuesta constituye una solución modular, que puede ampliarse con facilidad a otras clases de comportamiento o adaptarse a otras especies animales. Esto lo posiciona como una herramienta flexible, que podría evolucionar hacia un sistema integral de análisis de bienestar animal en entornos de producción intensiva.

En conjunto, el sistema propuesto complementa las aplicaciones ya existentes en el mercado, ocupando un nicho todavía desatendido: la automatización de la monitorización de bienestar animal en granja mediante visión por computadora. Su principal aportación radica en demostrar que, con una infraestructura sencilla y modelos optimizados, es posible trasladar técnicas avanzadas de *Deep Learning* a un entorno de gran impacto social y económico.

Parte II

Desarrollo de propuesta y resultados

Capítulo 4

Cuadernos de aprendizaje

En esta sección se presenta un análisis del caso de estudio trabajado mediante cuadernos interactivos, conocidos comúnmente como *notebooks*, así como los objetivos específicos abordados en cada uno de ellos. Un *notebook* es una aplicación que permite integrar código ejecutable, visualizaciones, resultados y texto explicativo en un mismo documento. Esta funcionalidad facilita el desarrollo de experimentos reproducibles, la documentación del flujo de trabajo y la presentación clara del razonamiento detrás del código.

Existen diversas plataformas que permiten el uso de *notebooks*, siendo Jupyter Notebook una de las más utilizadas en entornos locales. Jupyter destaca por su flexibilidad, compatibilidad con múltiples lenguajes de programación y por ofrecer una experiencia interactiva que resulta especialmente útil para tareas de análisis de datos, prototipado de modelos y visualización.

Por otro lado, Google Colab es una alternativa basada en la nube que mantiene la estructura de Jupyter pero incorpora ventajas adicionales. Entre ellas se encuentra la disponibilidad de recursos computacionales remotos, como el uso gratuito de unidades GPU y TPU, así como un entorno preconfigurado con muchas de las librerías científicas y de *Machine Learning* más comunes. Esta característica lo convierte en una opción accesible y potente para el desarrollo de proyectos sin necesidad de realizar configuraciones locales.

4.1. Caso de estudio elegido

Como se mencionó en la introducción, el análisis automatizado del comportamiento animal ha cobrado una creciente relevancia en ámbitos como la producción ganadera, el bienestar animal o la vigilancia sanitaria. En este contexto, la visión por computador se consolida como una herramienta clave, ya que permite extraer información útil a partir de secuencias visuales sin necesidad de sensores adicionales. Las cámaras instaladas en granjas o centros de investigación proporcionan datos continuos que pueden ser analizados para detectar patrones de movimiento, interacciones sociales, signos de estrés o posibles comportamientos anómalos en los animales.

Este tipo de análisis ofrece múltiples ventajas: no solo permite realizar un seguimiento

no invasivo del estado de los animales, sino que, además, puede integrarse en sistemas automáticos de alerta que contribuyan a la mejora del control sanitario o a la optimización de recursos en entornos agrícolas. Frente a otros sistemas de monitorización tradicionales, el uso de visión por computador reduce costes, minimiza la intervención humana y permite analizar múltiples variables de forma simultánea.

En este trabajo se ha desarrollado un caso de estudio orientado al análisis de grabaciones de cerdos en un entorno controlado. Mediante técnicas de aprendizaje profundo y herramientas de anotación, se han abordado tareas de detección y clasificación de comportamiento con el objetivo de explorar la viabilidad de aplicar modelos de visión artificial al monitoreo animal. Aunque la solución no se ha implementado en tiempo real, se sientan las bases para una futura integración en sistemas de vigilancia continua.

4.2. Dataset seleccionado

4.2.1. *Edinburgh Pig Behavior Video Dataset*

El conjunto de datos utilizado [8] en este estudio corresponde a grabaciones de vídeo registradas durante el día en un corral con ocho cerdos, capturadas mediante una cámara RGB-D colocada en vista cenital (ver Figura 4.1). Las grabaciones fueron adquiridas con una frecuencia de 6 fotogramas por segundo y una resolución de 1280×720 píxeles. Cada clip tiene una duración de un minuto, lo que equivale a un total de 1800 imágenes por clip. El *dataset* completo original está organizado en múltiples clips agrupados por día de grabación, y cada uno de ellos contiene:

- **color.mp4:** vídeo RGB con 1800 frames.
- **depth.mp4:** vídeo de profundidad sincronizado.
- **background.png y background_depth.png:** imágenes estáticas de referencia.
- **depth_scale.npy:** archivo con la escala de profundidad en centímetros.
- **inverse_intrinsic.npy:** matriz inversa de parámetros intrínsecos de la cámara.
- **rot.npy:** parámetros de rotación y traslación (no utilizados).
- **times.txt:** archivo con las marcas de tiempo correspondientes a cada fotograma.

Figura 4.1: Imagen de ejemplo de la visión de la cámara.

El *dataset* original incluye anotaciones de comportamiento con etiquetas aplicadas a cada cerdo en cada imagen, empleando cajas delimitadoras y un identificador de seguimiento. Estas etiquetas cubren 17 comportamientos distintos, aplicados sobre una muestra de clips seleccionados. Las anotaciones están estructuradas en formato JSON y se encuentran asociadas directamente a cada clip etiquetado.

4.3. Objetivos de aprendizaje

El desarrollo técnico de este trabajo se ha basado principalmente en un cuaderno de trabajo (*notebook*) que concentra las fases centrales del sistema: el entrenamiento del modelo, la generación de inferencias y el control de alertas. Las tareas previas, como la preparación del conjunto de datos, el etiquetado o la organización de los ficheros de entrada, se han implementado de forma modular mediante scripts externos que permiten separar responsabilidades y facilitar la reutilización del código.

Este enfoque permite una mayor claridad en el flujo de trabajo y ofrece una estructura flexible, donde el *notebook* actúa como núcleo funcional del sistema mientras los scripts aportan soporte en fases específicas.

Los objetivos de aprendizaje asociados a este desarrollo técnico son los siguientes:

- Familiarizarse con las principales librerías utilizadas en visión por computador y aprendizaje profundo, como OpenCV, Ultralytics, y otras herramientas auxiliares.

- Aplicar técnicas de preparación de datos mediante scripts personalizados, incluyendo organización, transformación y validación de conjuntos de imágenes etiquetadas.
- Utilizar modelos de detección de objetos preentrenados, ajustándolos a cerdos mediante fine-tuning [66].
- Entrenar un sistema de detección de comportamientos anómalos en animales a partir de datos reales, integrando lógica personalizada para el control de alertas según variables configurables.
- Desarrollar un *notebook* funcional que integre entrenamiento, validación, inferencia y sistema de alertas de forma modular y reproducible.
- Evaluar el rendimiento del modelo mediante métricas adecuadas y pruebas sobre vídeos de entrada, considerando su posible aplicación futura en un entorno en tiempo real.
- Diseñar un flujo de trabajo reproducible y adaptable, que pueda servir de base para ampliaciones futuras o para su despliegue en entornos productivos con streaming de vídeo.

4.4. Tecnologías usadas

A lo largo del desarrollo del presente trabajo se han empleado diversas herramientas y librerías de software enfocadas en la visión por computador, el aprendizaje profundo y el seguimiento de objetos. A continuación, se describen brevemente las principales tecnologías utilizadas:

4.4.1. Herramienta de etiquetado: CVAT

Para la anotación manual del conjunto de datos se ha utilizado CVAT (*Computer Vision Annotation Tool*), una herramienta open source desarrollada para la creación de anotaciones en imágenes y vídeos. CVAT permite generar etiquetas en formatos compatibles con los principales frameworks de visión por computador, lo que facilita la integración directa con herramientas como YOLO. Su interfaz web permite definir con precisión las regiones de interés en cada imagen, asignar clases y exportar las anotaciones en formatos como YOLO.

4.4.2. Ultralytics YOLO

Se ha utilizado la implementación oficial de YOLOv10, proporcionada por la librería ultralytics, como base para la detección de objetos. Esta herramienta permite entrenar modelos con configuraciones personalizadas de manera sencilla, aprovechar modelos preentrenados y realizar inferencias de forma rápida y eficiente. Ofrece compatibilidad con múltiples arquitecturas, así

como integración directa con PyTorch.

4.4.3. Bytetrack

Para la parte de seguimiento de objetos se empleó ByteTrack, un algoritmo de *multi-object tracking* (MOT) de alto rendimiento, que se integra con YOLO y permite enlazar detecciones consecutivas en el tiempo, generando trayectorias coherentes. Su implementación se ha utilizado para aplicar seguimiento post-detección a partir de las predicciones realizadas por el modelo.

4.4.4. OpenCV

La librería OpenCV (*Open Source Computer Vision Library*) se ha empleado para tareas relacionadas con la lectura, procesamiento y visualización de imágenes y vídeos. Ha sido una herramienta clave para manipular los flujos de datos visuales tanto en la etapa de entrenamiento como en la de inferencia.

4.4.5. Numpy

Se ha utilizado NumPy para el manejo eficiente de estructuras de datos en forma de arrays multidimensionales. Esta librería es fundamental para la manipulación numérica de imágenes y matrices durante las transformaciones previas y posteriores al entrenamiento.

4.4.6. PyTorch

Como backend principal para el entrenamiento del modelo, se ha utilizado PyTorch, una de las bibliotecas más populares para *Deep Learning*. Facilita el diseño y la ejecución de redes neuronales mediante una estructura flexible y dinámica.

4.4.7. Matplotlib

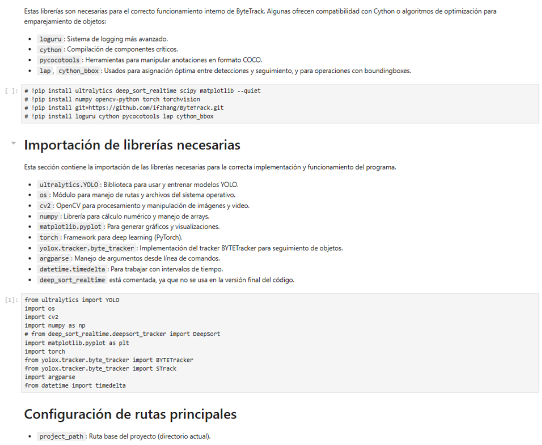
La librería Matplotlib se ha usado para la visualización de resultados, como la representación de métricas de entrenamiento, predicciones del modelo o imágenes con anotaciones superpuestas.

4.4.8. Otras librerías

Adicionalmente, se han utilizado módulos estándar como os, para la interacción con el sistema de archivos; argparse, para la gestión de argumentos desde terminal; datetime, para cálculos temporales; y scipy, cython_bbox o pycocotools, que facilitan cálculos de métricas y soporte para anotaciones en formato COCO.

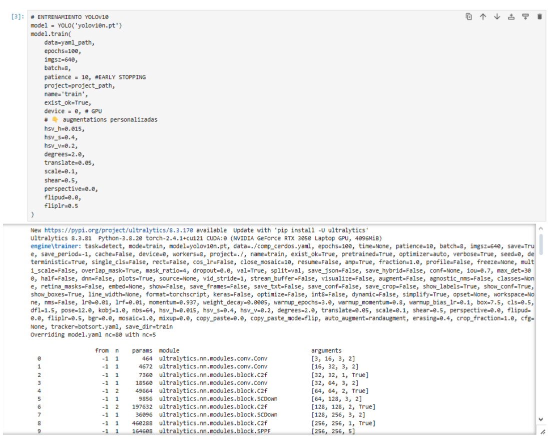
4.5. Muestra del cuaderno

A continuación, en la Figura 4.2, se muestran unas capturas del *notebook* de entrenamiento. En ellas se puede observar comentarios para cada celda de código, código ejecutable y resultados de la ejecución. En el Apéndice A, se encuentra un breve manual sobre la instalación de Anaconda por si el usuario quisiese ejecutarlos en local.

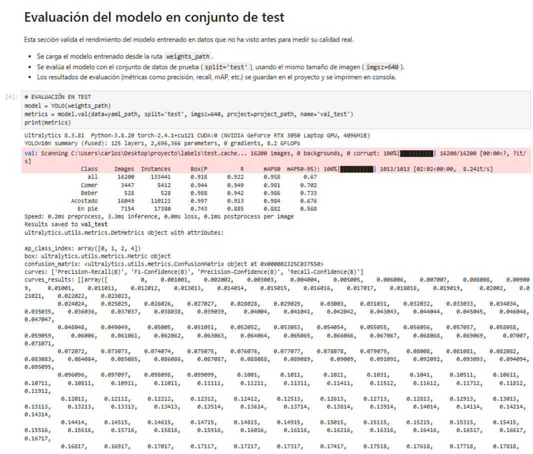


(a) Captura 1

(b) Captura 2



(c) Captura 3



(d) Captura 4

Figura 4.2: Capturas de pantalla del cuaderno de entrenamiento.

Capítulo 5

Caso de estudio

5.1. Descripción del conjunto de datos

En este apartado se describirá el dataset utilizado para entrenar el modelo de detección y seguimiento.

En una fase inicial, la idea era utilizar el *Edinburgh Pig Behavior Video Dataset* completo, debido a su variedad de etiquetas y cantidad de vídeos disponibles. Sin embargo, al realizar las primeras pruebas de entrenamiento, se detectaron errores significativos en el etiquetado original: al observar el conjunto de imágenes (*batch*), se veía que las *bounding boxes* no delimitaban correctamente a los cerdos, sino que aparecían sobre zonas vacías o sin presencia animal, lo que comprometía la calidad del entrenamiento.

Como solución a estas inconsistencias, se optó por realizar un proceso de reetiquetado mediante la herramienta CVAT. Este consistió en seleccionar y anotar nuevamente un subconjunto del dataset original, compuesto por 60 vídeos de 1800 imágenes cada uno, utilizando un nuevo esquema de etiquetas simplificado. En lugar de las diecisiete anteriores, las nuevas clases de comportamiento consideradas fueron cinco: *en pie, acostado, sentado, comer y beber*. Este etiquetado se llevó a cabo manualmente, asegurando una correcta alineación entre las cajas delimitadoras y la posición real de cada animal en la escena (ver Figura 5.1).

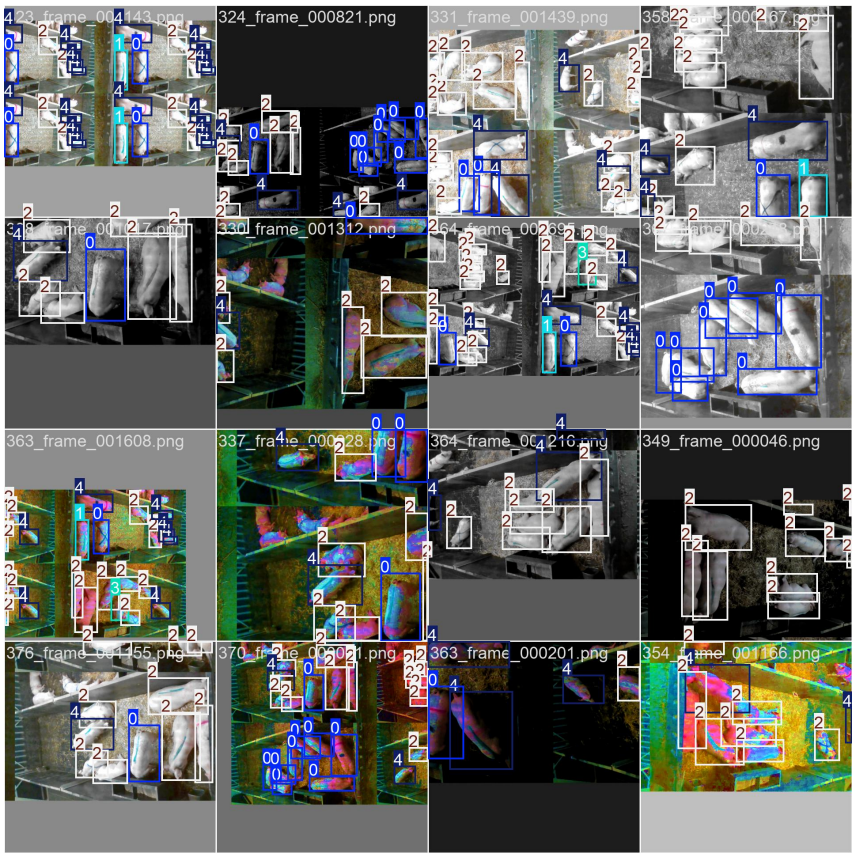


Figura 5.1: Conjunto imágenes dataset reetiquetado.

Una vez completado el proceso de etiquetado y descargado el conjunto de datos en el formato requerido, se procedió a unificar su estructura, organizando todas las imágenes y sus anotaciones bajo un mismo esquema común. Para asegurar una identificación precisa de cada fotograma y evitar conflictos por nombres repetidos, los archivos fueron renombrados siguiendo un patrón homogéneo que reflejaba tanto el vídeo de origen como el número de frame dentro del clip original. Esto permitió preservar la secuencia temporal de cada grabación y evitar ambigüedades entre imágenes con nombres iniciales idénticos.

Posteriormente, el *dataset* fue dividido en tres particiones: *entrenamiento* (43 vídeos), *validación* (8 vídeos) y *test* (9 vídeos), respetando una distribución equilibrada en número de clips y asegurando que no existiera solapamiento entre ellas. Esta estructuración final facilitó su uso directo durante las fases de entrenamiento y evaluación del modelo.

El resultado de todo este proceso constituye el dataset final utilizado en este trabajo, el cual se encuentra en el formato YOLO, compuesto por:

- Las imágenes individuales (fotogramas) extraídas de cada clip etiquetado.

- Los archivos de anotación correspondientes a cada imagen, en formato YOLO, con una línea por cada cerdo detectado incluyendo su clase y coordenadas normalizadas de la *bounding box*.

- Un archivo de configuración que describe las rutas relativas a las carpetas de entrenamiento, validación y test, así como la lista de clases del *dataset*.

5.2. Carga y preparación del conjuntos de datos

El planteamiento inicial consistía en emplear una red preentrenada YOLOv8n de Ultralytics para adaptarla a la detección de comportamientos en cerdos mediante fine-tuning utilizando el dataset original *Edinburgh Pig Behavior Video Dataset*. Sin embargo, como se mencionó anteriormente, durante las primeras pruebas de entrenamiento, se observaron errores de etiquetado en las anotaciones, por lo que se optó por el reetiquetado manual empleando la herramienta CVAT.

Previo a la carga en CVAT, se empleó el script *dividir_dataset.py* para organizar el material de partida. Este script automatiza dos tareas fundamentales:

1. **Renombrado y centralización de vídeos:** localiza todos los archivos color.mp4 del dataset original, los renombra de forma secuencial con ceros a la izquierda (p. ej., 0000001.mp4) y los mueve a un único directorio de trabajo.

2. **División inicial en subconjuntos:** distribuye los vídeos aleatoriamente en tres particiones (*train, val y test*) respetando una proporción aproximada del 70 %, 10 % y 20 %, respectivamente.

Con los vídeos organizados, se creó en CVAT un nuevo proyecto, definiendo un esquema de etiquetas simplificado de cinco clases: *en pie, acostado, sentado, comer y beber*. Los vídeos fueron asignados como tareas independientes, permitiendo su etiquetado de manera segmentada (ver Figura 5.2).

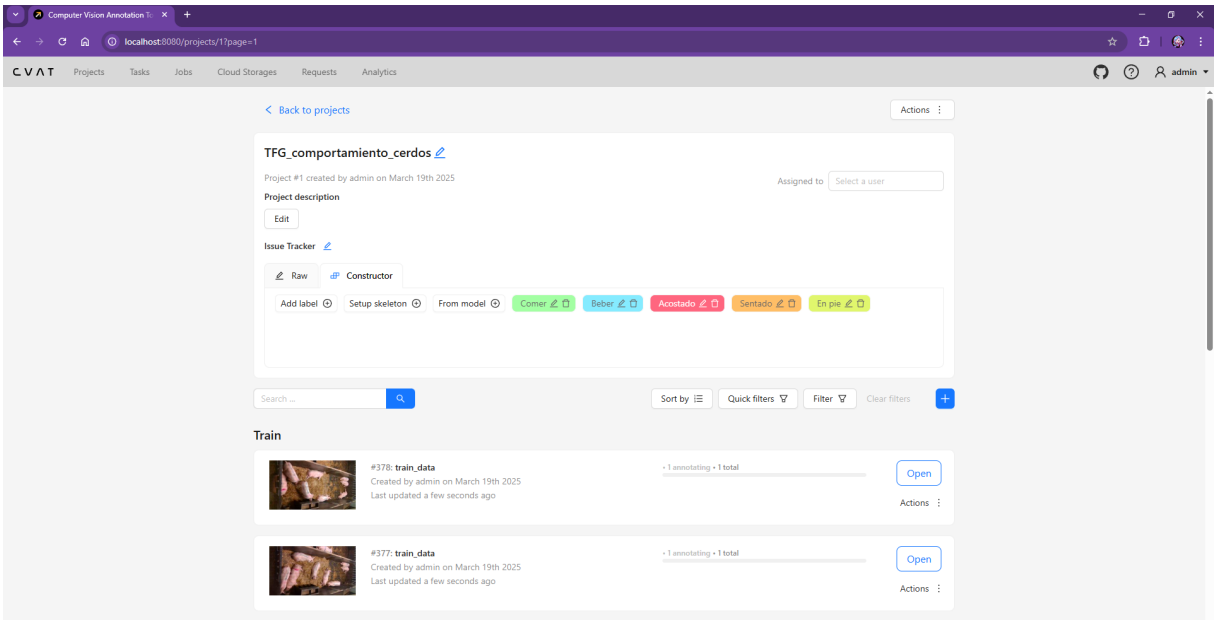


Figura 5.2: Pantalla del proyecto CVAT con las etiquetas y vídeos añadidos.

Durante la fase de anotación, se empleó la herramienta de caja rectangular (*bounding box tool*) por su compatibilidad con el formato YOLO. El etiquetado se realizó revisando manualmente los fotogramas relevantes, avanzando de 10 en 10 frames cuando era posible, corrigiendo errores y asegurando la alineación precisa de cada caja con la posición real del animal. En casos de oclusión parcial, las cajas se ajustaron intuyendo la posición mediante el contorno visible y el movimiento previo del animal.

Observando la Figura 5.3 se puede apreciar cómo es el proceso completo para un vídeo, destacando elementos clave como:

- Delimitar un nuevo rectángulo: esta opción permite seleccionar la clase o etiqueta, el tipo de dibujo 2 o 4 puntos/clics y la opción de etiquetar manualmente frame a frame (*shape*) o que realice el seguimiento (*track*), ajustando la posición de la caja cuando no sea correcta.
- Barra de progreso: en la parte superior se observa una barra de progreso para poder ir avanzando en los frames del vídeo. Permite ir hacia delante y atrás, ya sea 1 o 10 frames, o incluso hasta el frame inicial o final del vídeo.
- Objetos: en la parte derecha se listan los objetos etiquetados (en este caso, cerdos) con el color asignado a cada etiqueta. Las opciones permiten ocultar o cambiar etiquetas según sea necesario, así como marcar puntos clave o frames específicos donde se haya modificado la etiqueta, facilitando cambiarla directamente desde ese punto, para, por ejemplo, un cambio de comportamiento.

El programa ofrece muchas más opciones y funcionalidades, pero las exploradas principalmente aquí son las mostradas.

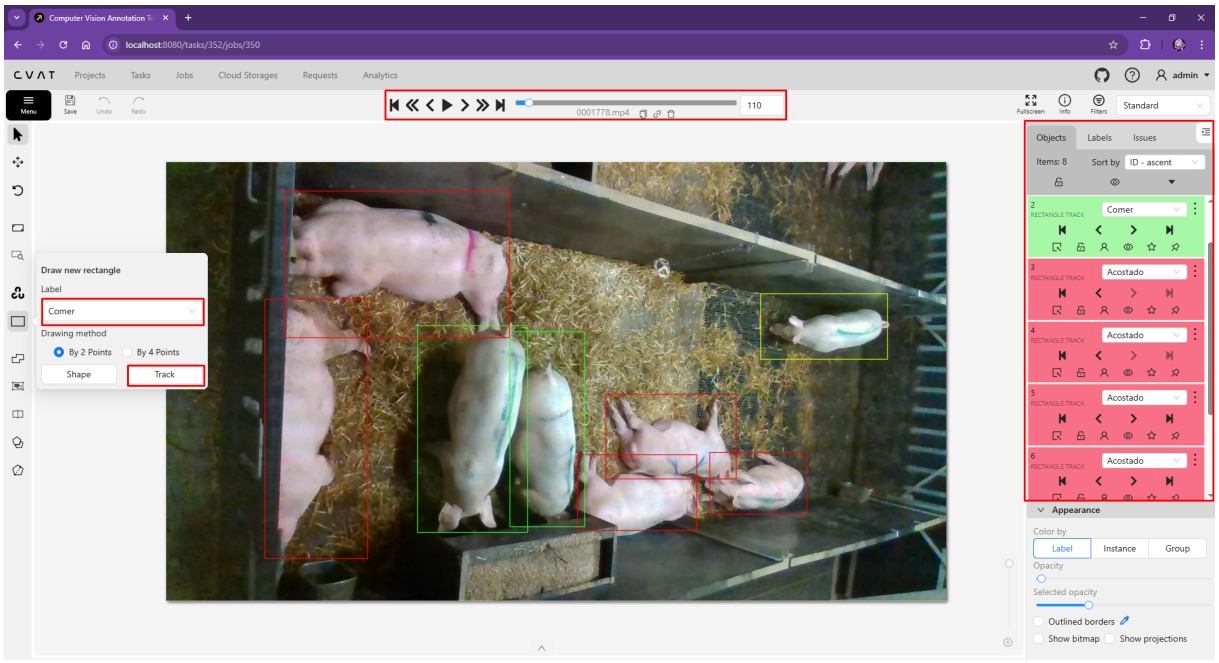


Figura 5.3: Etiquetado manual de un vídeo.

Una vez finalizado el etiquetado, los datos fueron exportados en formato YOLO (ver Figura 5.4), generando para cada imagen un archivo .txt con una línea por cada cerdo detectado, incluyendo la clase y las coordenadas normalizadas de la caja delimitadora, además de otros archivos de configuración que no resultaron relevantes debido a la versión de YOLO utilizada.

A continuación, un ejemplo real del formato de etiquetas YOLO utilizado, perteneciente a 357_frame_000977.txt.

```
1 0.195297 0.693646 0.131141 0.527569
2 0.251188 0.686257 0.208156 0.551486
2 0.521637 0.302424 0.196664 0.269431
2 0.848707 0.402667 0.105023 0.189222
2 0.555324 0.714653 0.134477 0.274722
2 0.706180 0.726910 0.139187 0.192847
2 0.820516 0.649667 0.103609 0.141361
2 0.624105 0.593264 0.131773 0.336167
```

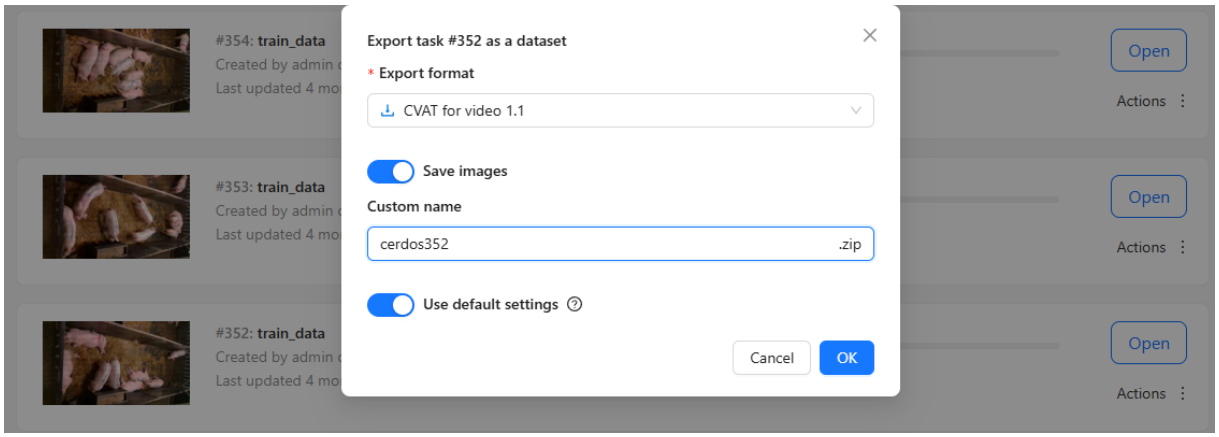


Figura 5.4: Exportación del dataset.

Al exportar, se constató que para cada vídeo las anotaciones e imágenes compartían nombre y ubicación, por lo que el siguiente paso fue procesar y normalizar la estructura de datos mediante scripts desarrollados específicamente para este proyecto:

- *descomprimir_organizar.py*: extrae los archivos exportados por CVAT y organiza su contenido en subcarpetas images y labels por vídeo.
- *unificar_eti_img.py*: integra las imágenes y anotaciones de cada partición en carpetas globales (*train, val, test*), añadiendo un prefijo con el identificador del vídeo a cada archivo para evitar conflictos de nombres.
- Particionado final: el dataset quedó distribuido en 43 vídeos para entrenamiento (71,7 %), 8 para validación (13,3 %) y 9 para test (15 %), sin solapamiento entre particiones.
- Archivo de configuración *.yaml*: se definieron en él las rutas relativas de cada conjunto y la lista de clases utilizadas.

Finalmente, se reservaron algunos vídeos sin etiquetar para su uso posterior en la evaluación cualitativa del modelo, permitiendo validar su rendimiento sobre datos completamente nuevos.

La organización final del conjunto de datos quedó estructurada de la siguiente manera:

```
proyecto/
- images/
    - train/
    - val/
    - test/
- labels/
    - train/
```

```
    - val/
    - test/
- com_cerdos.yaml
```

Este flujo de trabajo garantizó que el conjunto de datos final estuviese completamente limpio, estructurado y listo para el entrenamiento del modelo YOLOv8n, y más adelante, del modelo YOLOv10n, optimizando la calidad de las muestras y minimizando el riesgo de sesgos o errores derivados de las anotaciones originales.

Capítulo 6

Experimentación y evaluación

El objetivo de este capítulo es presentar el proceso experimental seguido para entrenar, evaluar e implementar un sistema de detección y seguimiento de cerdos en vídeo. Antes de detallar los hiperparámetros específicos y los resultados obtenidos, se ofrece una visión global del flujo de trabajo y del contexto visual de los datos empleados..

A lo largo del proceso, algunos de estos hiperparámetros se mantuvieron constantes mientras que otros se modificaron para estudiar su influencia en el rendimiento del modelo.

En primer lugar, se parte de la captura de vídeos de los animales en su entorno, que constituyen el conjunto de datos inicial. La Figura 6.1 ilustra el estado inicial de los datos, donde los cerdos aparecen sin anotaciones ni detecciones, y el estado final tras el procesamiento, mostrando las detecciones con cajas delimitadoras y alertas activas en pantalla. Cabe destacar que, aunque en las imágenes se observan más de ocho cerdos en las cuadras aledañas, el experimento se centra únicamente en la cuadra a la que apunta la cámara, es decir, sobre los ocho animales de interés. Esta ilustración permite al lector comprender la finalidad práctica del sistema y el tipo de información que se genera durante la fase de inferencia.

Aunque en las imágenes 6.1a se observan más de ocho cerdos en la cuadra contigua, el

(a) Estado inicial del dataset

(b) Estado final con objetivos cumplidos

Figura 6.1: Evolución visual de los datos desde su captura hasta la inferencia final.

experimento se centra únicamente en la cuadra a la que apunta la cámara, es decir, sobre los ocho animales de interés.

Visión general del proceso experimental

La Figura 6.2 presenta un esquema de alto nivel del flujo experimental seguido en este trabajo. Este pipeline refleja la secuencia completa del proceso: desde la preparación y anotación del conjunto de datos, pasando por la fase de entrenamiento de los modelos, hasta la evaluación de su desempeño y la posterior fase de inferencia en vídeo, que incluye seguimiento de animales y generación de alertas, así como el almacenamiento de métricas individuales.

El esquema destaca la naturaleza iterativa del proceso: tras la fase de inferencia, los hallazgos pueden retroalimentar ajustes en el entrenamiento para explorar nuevas configuraciones o estrategias. De esta manera, el pipeline no solo describe los pasos principales del experimento, sino que también enfatiza la lógica iterativa que permite optimizar progresivamente el sistema, facilitando al lector una visión clara de cómo se integran las distintas etapas en el desarrollo del modelo final.

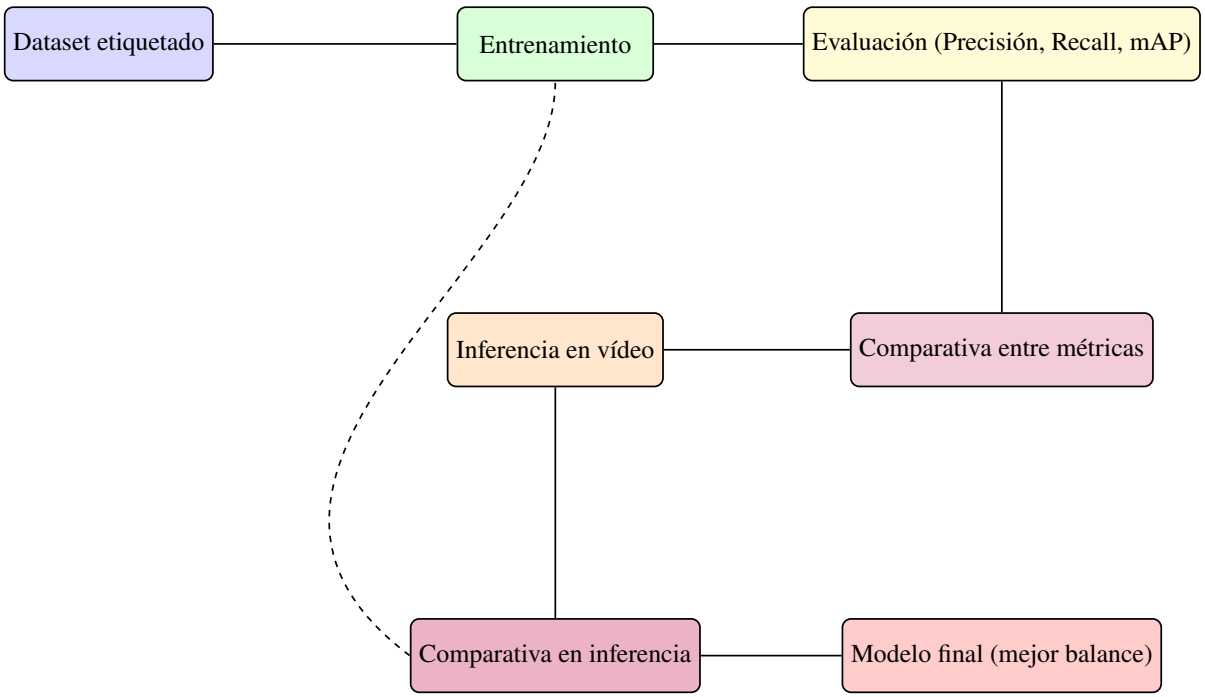


Figura 6.2: Pipeline experimental iterativo.

6.1. Hiperparámetros

En el entrenamiento de redes neuronales, los hiperparámetros son valores configurables que controlan el comportamiento del algoritmo de aprendizaje. A diferencia de los parámetros internos del modelo (pesos y sesgos), que se aprenden automáticamente, los hiperparámetros deben definirse manualmente antes del entrenamiento. Su elección influye directamente en la calidad de los resultados, el tiempo de entrenamiento y la capacidad de generalización del modelo.

En este proyecto, algunos hiperparámetros se mantuvieron en sus valores por defecto establecidos por la librería Ultralytics, mientras que otros fueron modificados para adaptarse mejor a las características del conjunto de datos, a las limitaciones de hardware y a los objetivos del sistema. A continuación, se describen los más relevantes:

- **Tamaño de imagen (*imgsz*):** se fijó en 640 píxeles para mantener un equilibrio entre la calidad de detección y el coste computacional. Un tamaño mayor podría mejorar la precisión en objetos pequeños, pero aumentaría considerablemente el tiempo de entrenamiento y la memoria necesaria.

- **Tamaño de batch (*batch*):** inicialmente se trabajó con 16 imágenes por *batch*, valor habitual en YOLO. Sin embargo, debido a limitaciones de memoria en la GPU, se redujo a 8. Esto permite entrenar con menos riesgo de interrupciones, aunque puede aumentar ligeramente el tiempo de entrenamiento por época.

- **Número máximo de épocas (*epochs*):** se estableció en 100 como límite superior. Este valor no implica que siempre se alcancen las 100 épocas, ya que el entrenamiento puede detenerse antes gracias al uso de *early stopping*.

- ***Early stopping* – Paciencia (*patience*):** se configuró con un valor de 10. Esto significa que el entrenamiento se detiene automáticamente si no se observa mejora en las métricas de validación durante 10 épocas consecutivas, evitando así un sobreentrenamiento innecesario y reduciendo el tiempo total.

- **Aumentaciones de datos:** se aplicaron modificaciones controladas en parámetros como tono, saturación y brillo, junto con pequeñas rotaciones, traslaciones, escalados y volteos horizontales. El objetivo de estas técnicas es aumentar la diversidad de las imágenes de entrenamiento, mejorando la capacidad del modelo para generalizar y adaptarse a variaciones en el entorno real.

- **Dispositivo de entrenamiento (*device*):** el entrenamiento comenzó utilizando CPU debido a la configuración inicial, pero posteriormente se migró a GPU para acelerar significativamente el procesamiento y permitir un mayor volumen de datos por época.

6.2. Métricas

En el proceso de entrenamiento y evaluación de modelos de detección de objetos, la elección de las métricas resulta fundamental para cuantificar el rendimiento y garantizar que el sistema cumple con los objetivos planteados. En este proyecto se han empleado las métricas estándar que ofrece la librería Ultralytics YOLO, prestando especial atención a aquellas que han guiado el *early stopping* y la selección del mejor modelo.

Las métricas principales utilizadas han sido:

- **Precisión (Precision):** mide la proporción de predicciones positivas que han sido correctas. En el contexto de detección de objetos, representa la fracción de objetos detectados que realmente pertenecen a la clase predicha. Se calcula como:

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos + Falsos Positivos}}$$

  Su valor oscila entre 0 y 1 (o 0 % y 100 %). Valores cercanos a 1.0 indican que el modelo comete pocos falsos positivos, por lo que valores por encima de 0.90 se consideran excelentes en la práctica.

- **Exhaustividad (*Recall*):** evalúa la proporción de objetos reales que el modelo ha detectado correctamente. Se define como:

$$\text{Recall} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos + Falsos Negativos}}$$

  También varía entre 0 y 1. Un *recall* alto ($\geq$ 0,90) significa que el modelo deja pocos objetos reales sin detectar. En la práctica, valores entre 0.80 y 0.90 se consideran buenos, y superiores a 0.90, muy buenos.

- **Media de la Precisión Promedio a IoU 0.5 (mAP@0.5):** es la métrica principal utilizada para medir la mejora del modelo durante el entrenamiento. El mAP combina precisión y *recall* a diferentes umbrales de confianza, calculando el área bajo la curva *Precision-Recall*. El valor *mAP@0.5* corresponde a un umbral de *Intersection over Union* (IoU) de 0.5, lo que significa que una predicción se considera correcta si el solapamiento entre el cuadro predicho y el real es de al menos el 50 %.

- **Media de la Precisión Promedio entre IoU 0.5 y 0.95 (*mAP@0.5:0.95*):** es una versión más estricta del *mAP*, ya que promedia los valores de AP obtenidos para diferentes umbrales de IoU (desde 0.5 hasta 0.95 en pasos de 0.05). Esta métrica ofrece una visión más exigente del rendimiento, penalizando errores en la localización y detección de objetos

pequeños o parcialmente visibles. Este valor tiende a ser notablemente más bajo que el *mAP@0.5* debido a la exigencia en la localización. Valores por encima de 0.50 se consideran buenos, y superiores a 0.65 suelen indicar un modelo de alto rendimiento. Esta métrica fue la empleada por defecto en YOLO para el criterio de *early stopping*, deteniendo el entrenamiento cuando no se observó mejora significativa durante el número de épocas indicado.
Sus valores están entre 0 y 1, siendo 1.0 el rendimiento perfecto. En visión por computadora, valores superiores a 0.80 suelen considerarse excelentes, y por encima de 0.90, de nivel avanzado. En este proyecto, esta métrica fue la empleada por defecto en YOLO para el criterio de *early stopping*.

Aunque durante el entrenamiento también se monitorizaron las métricas de pérdida (*box_loss, cls_loss, dfl_loss*), estas no fueron utilizadas como criterio principal para la selección del mejor modelo debido a la elevada cantidad de datos y a que el objetivo se centraba en maximizar la calidad de detección medida por el *mAP*.

En conjunto, el seguimiento simultáneo de precisión, *recall, mAP@0.5* y *mAP@0.5:0.95* ha permitido evaluar de forma integral la evolución del modelo, equilibrando la capacidad de detectar el mayor número posible de objetos con la minimización de falsas detecciones, y garantizando que el sistema mantuviera un alto nivel de rendimiento en todas las clases del conjunto de datos.

6.3. Proceso de entrenamiento

Una vez preparado el conjunto de datos en formato YOLO y definida su estructura en tres particiones (train, val y test), junto con el archivo de configuración *comp_cerdos.yaml* que especifica las rutas y las clases, se procedió a la fase de entrenamiento del modelo.

Para optimizar el tiempo de convergencia y aprovechar conocimiento previo, se empleó la estrategia de *transfer learning*, partiendo de los pesos preentrenados *yolov10n.pt*. Este enfoque permite reutilizar características visuales aprendidas en grandes bases de datos, lo que mejora el rendimiento inicial y reduce la cantidad de datos necesarios para obtener resultados satisfactorios.

Los hiperparámetros se establecieron en función de un equilibrio entre precisión, coste computacional y limitaciones de hardware:

- **Tamaño de imagen (*imgsz*):** se fijó en 640 píxeles por recomendación del propio modelo YOLO, ya que ofrece un buen compromiso entre detección de objetos pequeños y velocidad de inferencia. Un valor mayor podría aumentar la precisión, pero también el tiempo de entrenamiento y el consumo de memoria.

- **Tamaño de batch (*batch*):** inicialmente se empleó un valor de 16, habitual en entrenamientos con YOLO, pero debido a la limitación de memoria de la GPU se redujo a 8, lo que permitió mantener la estabilidad sin provocar errores por falta de memoria.
- **Número máximo de épocas (*epochs*):** se definió un límite de 100 para asegurar suficiente tiempo de aprendizaje, aunque el uso de *early stopping* permitió detener el proceso antes si no se observaban mejoras.
- **Paciencia (*patience*):** se estableció en 10 épocas, de modo que si durante ese intervalo no mejoraban las métricas de validación, el entrenamiento se interrumpía para evitar sobreajuste y desperdicio de recursos.
- **Aumentaciones de datos:** se incluyeron transformaciones de color (tono, saturación y brillo) y geométricas (rotaciones, escalados, traslaciones y volteos horizontales). El objetivo fue mejorar la capacidad de generalización del modelo frente a variaciones en las condiciones reales de captura.
- **Dispositivo (*device*):** el entrenamiento comenzó en CPU, pero se migró a GPU (device=0) para multiplicar la velocidad de procesamiento y permitir manejar de forma eficiente el tamaño de lote establecido.

El entrenamiento se desarrolló en ciclos (épocas), en los que el modelo procesó de forma iterativa los lotes de datos de la partición train. En cada iteración, las imágenes se sometieron a una propagación hacia adelante (*forward pass*), se calcularon las pérdidas asociadas a la detección (localización, clasificación y ajuste de caja), y se realizó la retropropagación (*backpropagation*) para ajustar los pesos mediante el optimizador por defecto de la librería Ultralytics.

Al final de cada época, el modelo fue evaluado sobre el conjunto val, registrando métricas clave para monitorizar el aprendizaje y determinar si se cumplía el criterio de parada anticipada. Una vez alcanzado este criterio, los pesos correspondientes al mejor rendimiento se almacenaron y se utilizaron posteriormente para la fase de evaluación sobre el conjunto test.

6.4. Experimentación y resultados

A continuación, se describen las diferentes pruebas llevadas a cabo, así como los ajustes realizados y la justificación de las decisiones tomadas. Cada experimento se diseñó para modificar de forma controlada un parámetro o conjunto de parámetros, evaluando su impacto sobre el rendimiento del modelo.

Como se mencionó anteriormente, el conjunto de datos fue preparado en formato YOLO, dividido en particiones train, val y test, con su correspondiente archivo de configuración comp_cerdos.yaml.

Las métricas empleadas para la evaluación fueron: *mAP@50, mAP@50-95*, precisión y

recall, explicadas en el apartado anterior, así como el seguimiento de las curvas de pérdida de entrenamiento. El criterio de *early stopping* se estableció en 10 épocas sin mejora del *mAP@50-95* en validación, siguiendo el criterio predeterminado del framework YOLO.

6.4.1. Prueba inicial

En la primera prueba se entrenó el modelo YOLOv8n sin modificaciones sobre el dataset ni las clases, realizando únicamente ajustes básicos de configuración. Se estableció un máximo de 100 épocas, aplicando el *early stopping* (10 épocas sin mejora). El entrenamiento se ejecutó en GPU, aprovechando núcleos CUDA, lo que redujo significativamente los tiempos de cómputo.

Durante esta primera ejecución, el tamaño de batch inicial se fijó en 16. Este valor define cuántas imágenes se procesan simultáneamente antes de actualizar los pesos del modelo.

En esta configuración inicial se empleó el método de seguimiento DeepSORT (ver detalles en el apartado de Inferencia).

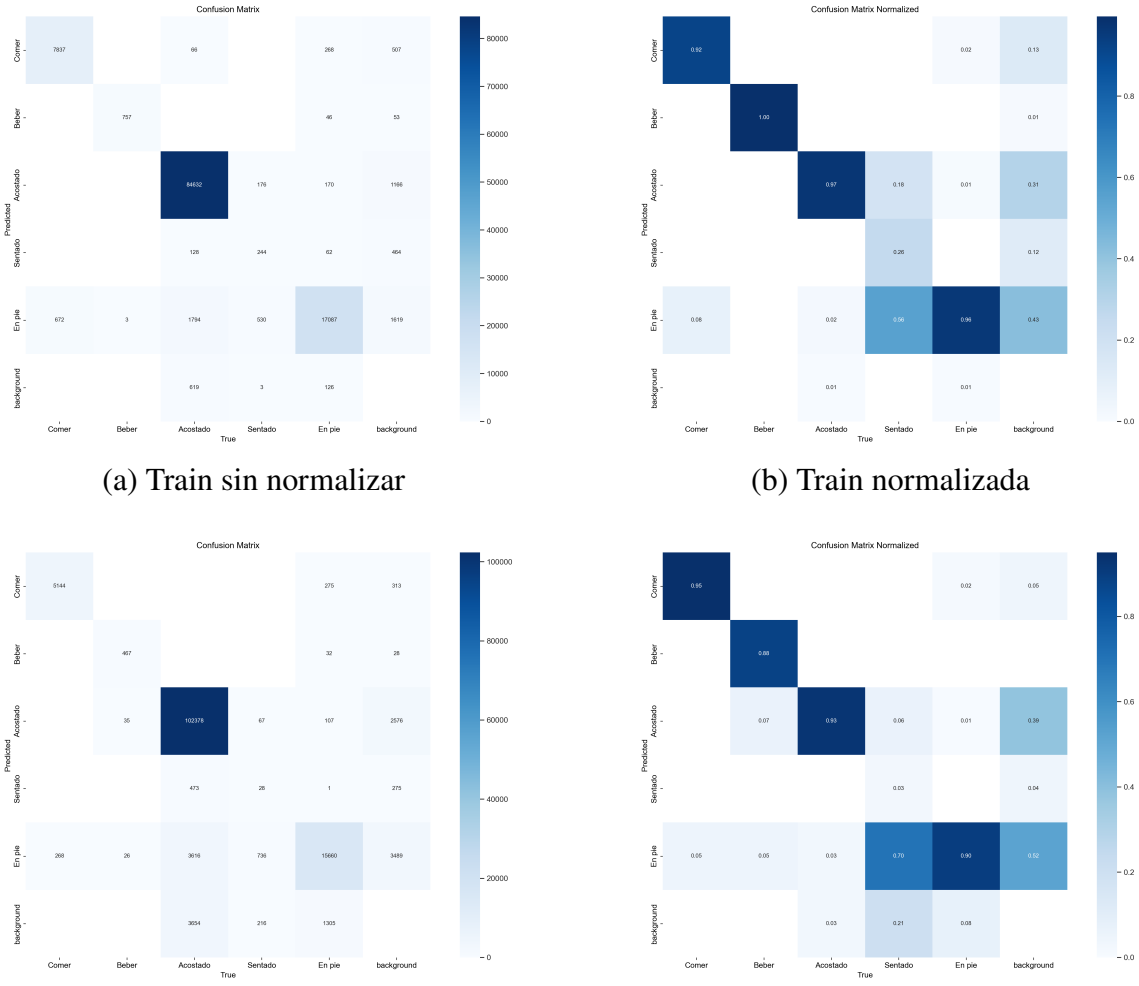
Los resultados obtenidos fueron aceptables para una prueba inicial (ver Tabla 6.1):

Precisión	Recall	mAP@50	mAP@50-95
0.749	0.733	0.763	0.528

Tabla 6.1: Métricas globales para la prueba inicial en los datos de test

Las matrices de confusión de la Figura 6.3 evidencian un desbalance acusado: la clase *acostado* domina el conjunto, mientras que la clase *sentado* presenta una escasez crítica de ejemplos, lo que se traduce en frecuentes errores de clasificación.

La curva precisión–recall (Figura 6.4) confirma que la clase *sentado* genera falsos positivos recurrentes, confundida con *en pie* o *acostado*. Este comportamiento refleja su carácter intermedio y su baja representación.



(a) Train sin normalizar

(b) Train normalizada

(c) Test sin normalizar

(d) Test normalizada

Figura 6.3: Matrices de confusión de la prueba inicial.

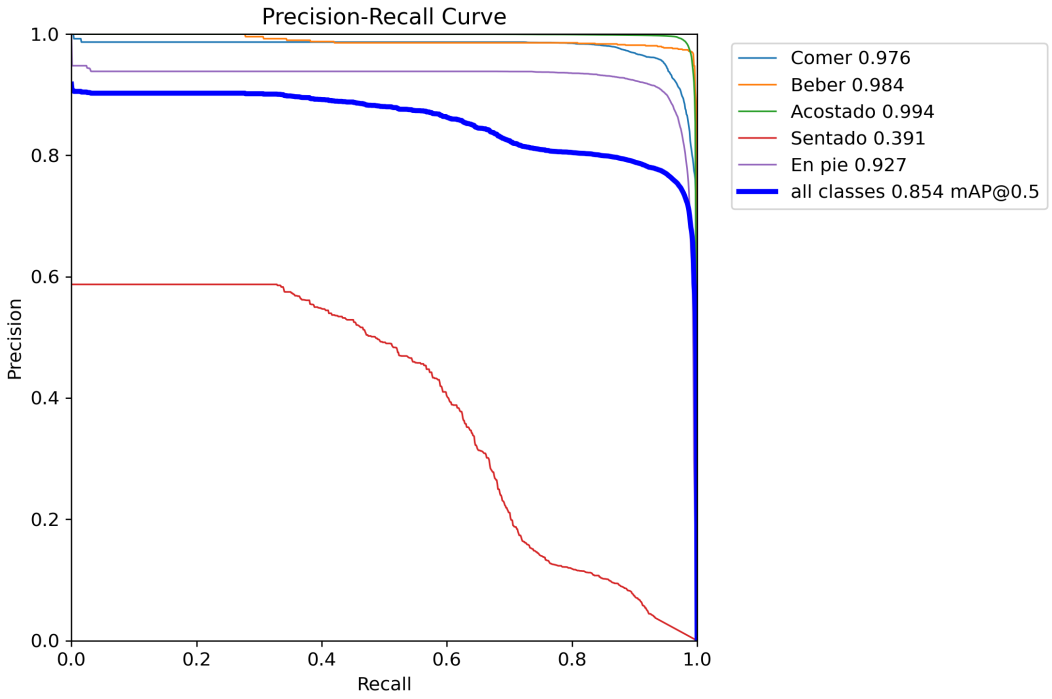


Figura 6.4: Curva precisión-recall para datos de entrenamiento de la prueba inicial.

En la evolución de pérdidas (Figura 6.5), se observa un descenso estable en entrenamiento, pero un ascenso temprano en validación (especialmente en *val/box_loss* y *val/dfl_loss*), indicando sobreajuste prematuro. Esto, junto con la inestabilidad de las métricas en validación, muestra que el modelo aprendió patrones específicos del entrenamiento sin generalizarlos de forma consistente.

En este contexto, el sobreajuste se debe principalmente a la limitada cantidad de datos disponibles y al fuerte desbalance entre clases. A pesar de aplicar técnicas de regularización y ajustes de entrenamiento, la escasez de ejemplos hace que el modelo tienda a memorizar los datos de entrenamiento en lugar de generalizar.

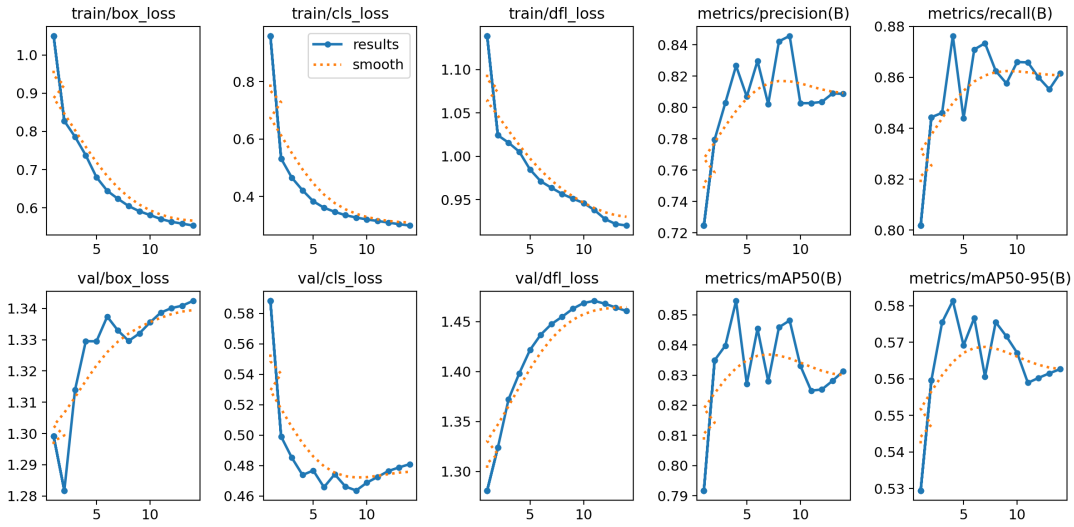


Figura 6.5: Métricas de entrenamiento y validación en la prueba inicial.

En conclusión, esta prueba confirmó la limitación estructural del dataset: el desbalance de clases y la escasez de ejemplos condicionan fuertemente el rendimiento.

6.4.2. Prueba 2 - Data Augmentation

En la segunda prueba se introdujeron técnicas de data augmentation con el objetivo de aumentar la variabilidad del conjunto de datos y mejorar la capacidad de generalización del modelo. Para poder aplicar estas transformaciones, fue necesario reducir el tamaño del *batch* de 16 a 8, ya que el aumento de la complejidad computacional derivado de las operaciones de augmentación provocaba problemas de memoria en la GPU.

Las transformaciones incluyeron ajustes de color (*hsv_h, hsv_s, hsv_v*), rotación (*degrees*), desplazamiento (*translate*), escalado (*scale*), cizalladura (*shear*), perspectiva y volteo horizontal (*fliplr*).

Es importante señalar que en YOLO estas técnicas se aplican de forma global, sin diferenciación por clase, lo que no soluciona el desbalance estructural del dataset.

Los resultados de esta prueba se recogen en la Tabla 6.2:

Precisión	Recall	mAP@50	mAP@50-95
0.752	0.792	0.789	0.555

Tabla 6.2: Métricas globales en test prueba 2 - data augmentation

Datos	Precisión	Recall	mAP@50	mAP@50-95
Train	0.953	0.958	0.974	0.681
Test	0.947	0.928	0.967	0.672

Tabla 6.3: Métricas globales para la prueba 3 – etiqueta “sentado” modificada

La mejora fue ligera pero consistente, indicando que la *data augmentation* aportó robustez, aunque sin resolver el problema fundamental de la clase *sentado*.

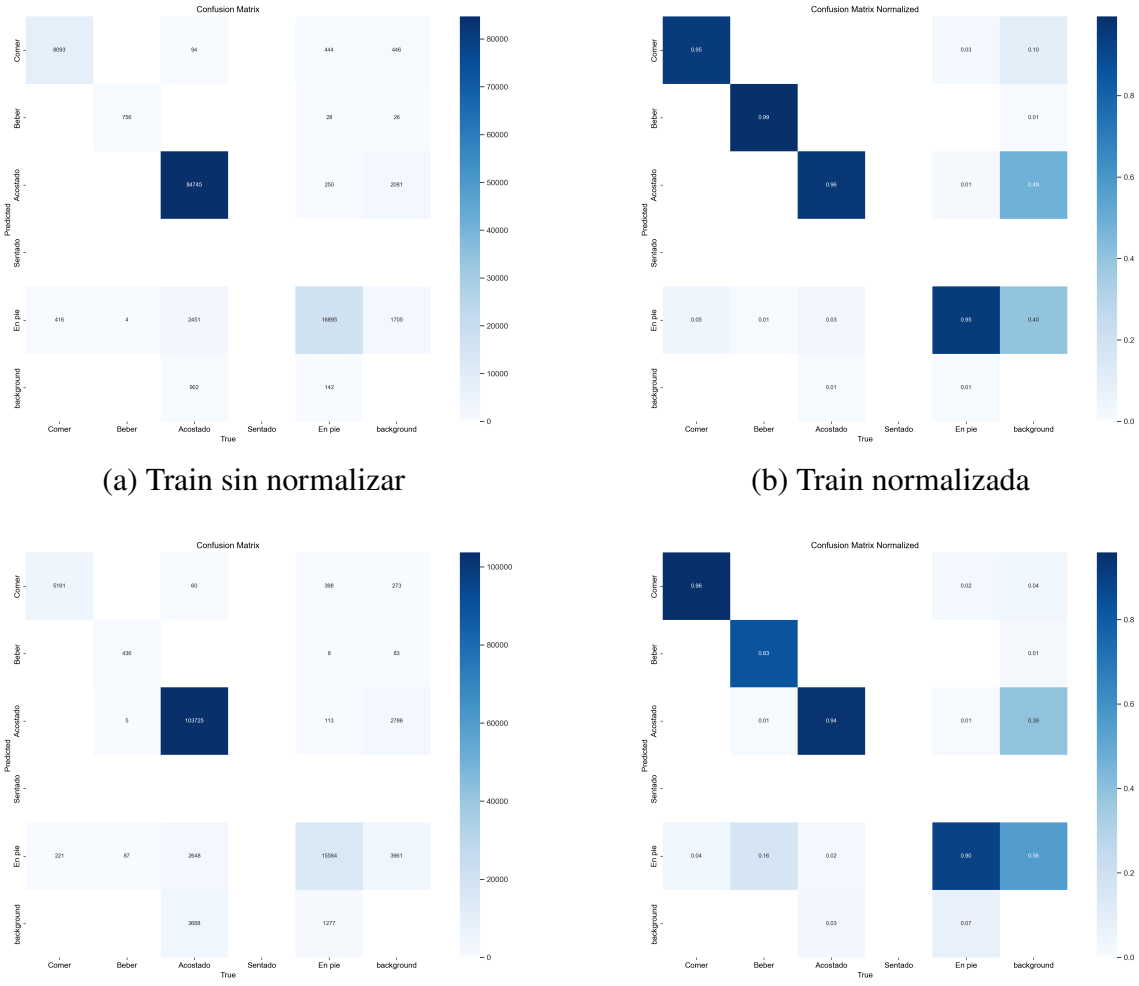
6.4.3. Prueba 3 - Etiqueta *Sentado* modificada

Con el objetivo de mitigar este problema, se plantearon dos estrategias. La primera de ellas consistió en modificar las etiquetas originales: tras generar una copia de seguridad de los ficheros de anotaciones, mediante un script (*sustituir_sentado.py*) que recorría todas las etiquetas y sustituía la clase *sentado* por la clase *en pie*, modificando el identificador de cada anotación correspondiente.

Los resultados mejoraron notablemente (ver Tabla 6.3), tanto en precisión como en recall y mAP.

Esta mejora también se refleja en las matrices de confusión de la Figura 6.6, donde se aprecia una disminución de errores en comparación con las pruebas iniciales.

De forma consistente, la curva precision-recall de la Figura 6.7 presenta un rendimiento superior al no incluir la clase sentado, lo que confirma que esta categoría introducía confusión en el entrenamiento.



(a) Train sin normalizar

(b) Train normalizada

(c) Test sin normalizar

(d) Test normalizada

Figura 6.6: Matrices de confusión para la prueba 3.

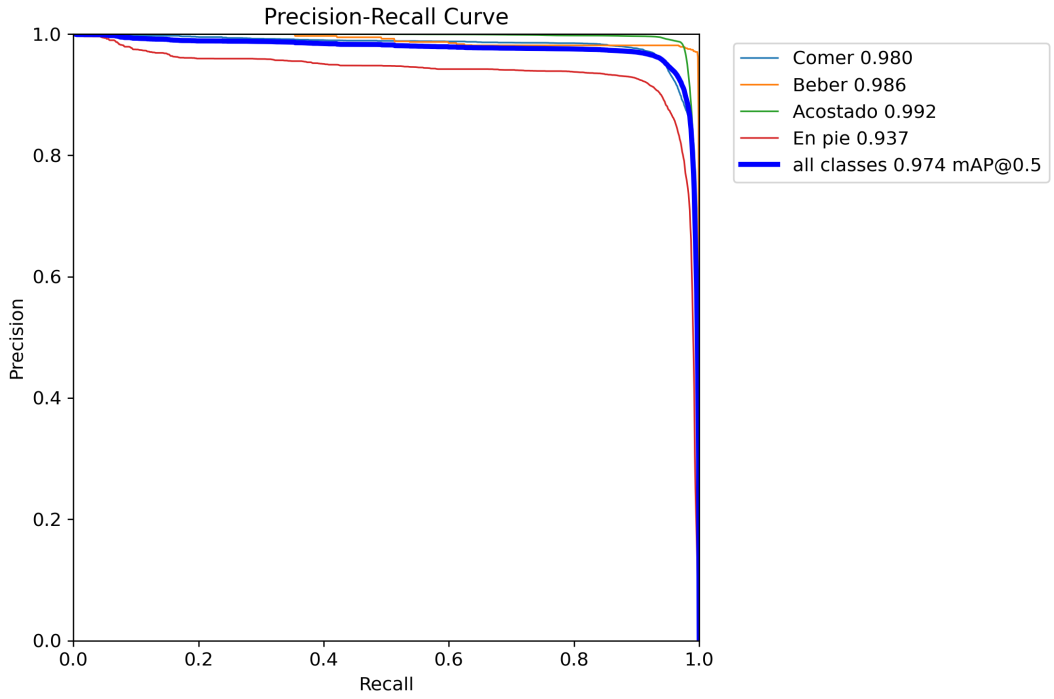


Figura 6.7: Curva precisión-recall para datos de entrenamiento de la prueba 3.

No obstante, la eliminación de la etiqueta no estuvo exenta de efectos secundarios. Al revisar los vídeos etiquetados automáticamente por el modelo, se observaron inconsistencias en la clasificación de determinados comportamientos: en situaciones sin apenas movimiento, el sistema alternaba entre *en pie* y *acostado* de forma poco coherente, sin justificación aparente. Este comportamiento revela que, si bien el cambio de etiquetas permitió estabilizar los resultados numéricos, también introdujo nuevas fuentes de error que comprometían la robustez de la detección.

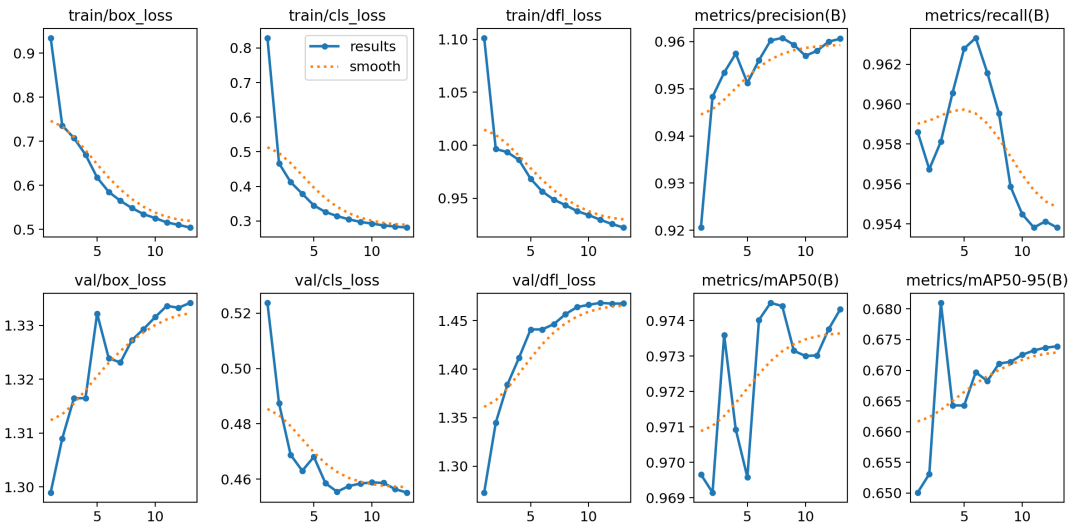


Figura 6.8: Métricas de entrenamiento y validación en la prueba 3.

La evolución de las pérdidas y métricas durante el entrenamiento y validación de la Figura 6.8 muestran curvas similares a las anteriores, lo que sigue indicando un pronto sobreajuste. Este efecto se explica, una vez más, en gran medida por la limitada cantidad de datos disponibles y el desbalance de clases, factores que condicionaron todos los experimentos realizados y que no fue posible resolver por completo en ninguna de las pruebas.

En conjunto, esta prueba demuestra que la clase *sentado* constituía una fuente de ambigüedad relevante dentro del conjunto de datos. Su sustitución permitió mejorar los indicadores cuantitativos, pero a costa de perder coherencia en determinadas situaciones, lo que llevó a plantear una segunda estrategia en la que se optó por eliminar la clase de forma definitiva (etiqueta *sentado* eliminada), evaluada en el siguiente apartado.

6.4.4. Prueba 4 - Etiqueta *Sentado* eliminada

La segunda estrategia consistió en eliminar las etiquetas con identificador *sentado* originales. Nuevamente se hizo otra copia de las etiquetas originales y se procedió a ejecutar un script (*borrar_sentado.py*) que recorría todas las etiquetas y eliminaba la línea completa para la clase *sentado*.

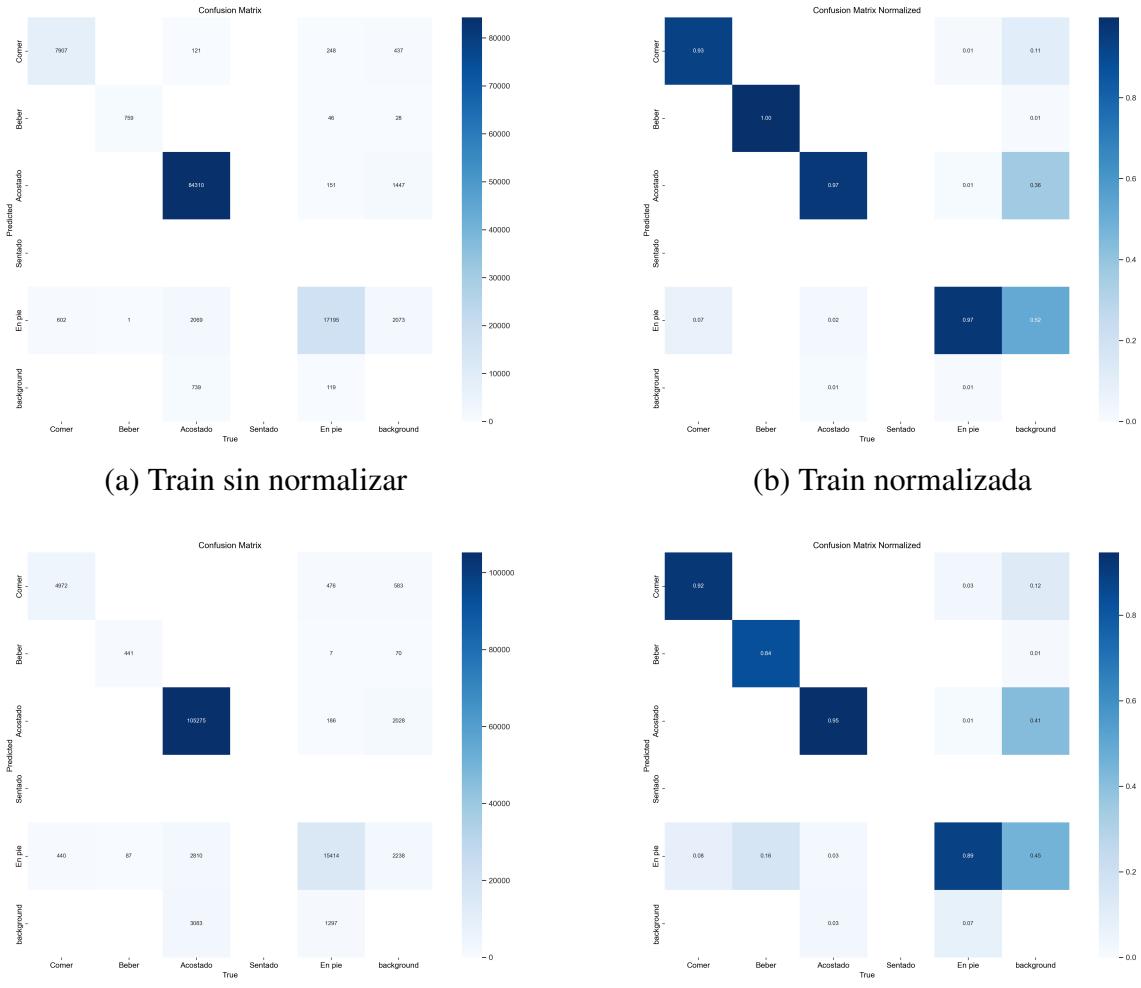
Los resultados obtenidos al ejecutar el entrenamiento con esas etiquetas empeoraron muy levemente las métricas globales de precisión, recall y mAP respecto a la primera estrategia, como se muestra en la Tabla 6.4.

Datos	Precisión	Recall	mAP@50	mAP@50-95
Train	0.954	0.957	0.975	0.675
Test	0.946	0.923	0.965	0.671

Tabla 6.4: Métricas globales para la prueba 4 – etiqueta “sentado” eliminada

En las matrices de confusión obtenidas (ver Figura 6.9) se aprecia un comportamiento similar al de la estrategia anterior, con una reducción clara de los errores asociados a la clase *sentado*, ya que esta ha sido eliminada. Sin embargo, persisten ciertos casos de confusión con la categoría *background*. Este término hace referencia a detecciones erróneas de regiones sin objeto (falsos positivos) o a instancias no detectadas (falsos negativos). Se observa que las clases más frecuentes del conjunto, como *acostado* y *en pie*, son las que presentan un mayor nivel de confusión con el fondo.

No obstante, cabe destacar que este nivel de confusión con el fondo no se traduce necesariamente en un mal desempeño durante la inferencia. Al evaluar el modelo sobre vídeos del dominio de entrenamiento, las detecciones resultaron coherentes y las métricas globales se mantuvieron estables. Esto indica que el problema observado en las matrices responde principalmente a la



(a) Train sin normalizar

(b) Train normalizada

(c) Test sin normalizar

(d) Test normalizada

Figura 6.9: Matrices de confusión para la prueba 4.

cantidad limitada y al desbalance de datos disponibles, más que a una deficiencia intrínseca del modelo. Con un conjunto de datos más amplio y equilibrado, es previsible que la confusión con el fondo disminuya, reforzando aún más la capacidad de generalización del sistema.

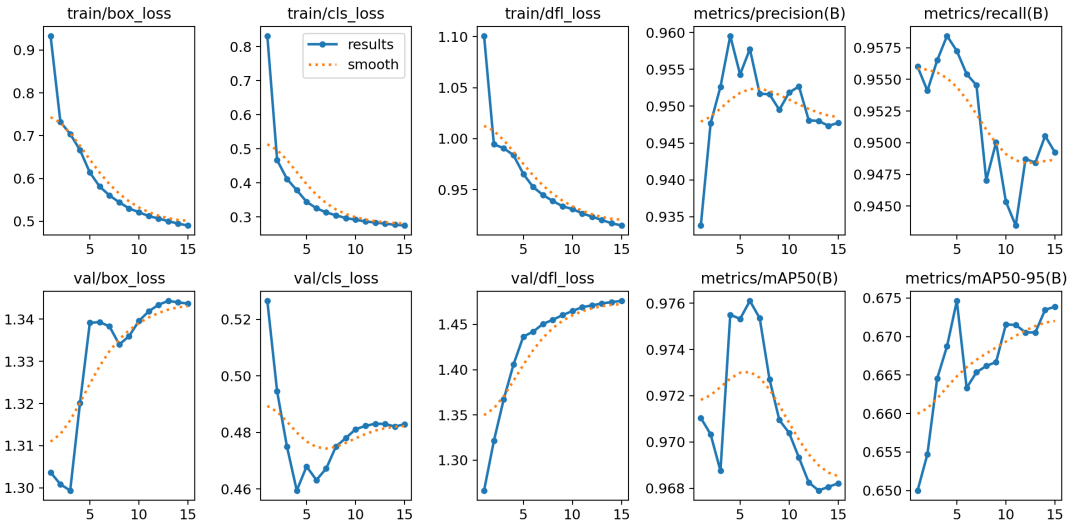


Figura 6.10: Métricas de entrenamiento y validación en la prueba 4.

La Figura 6.10 muestra la evolución de las pérdidas y métricas durante el entrenamiento y validación de la prueba 4. Como se puede observar en las pérdidas, se valora un descenso progresivo y estable en *train/box_loss*, *train/cls_loss* y *train/dfl_loss*, mientras que, en los datos de validación, las curvas de *val/box_loss* y *val/dfl_loss* son ascendentes, una vez más, indicando un sobreajuste.

En las métricas de precisión, recall y mAP, se observa una rápida mejora inicial hasta la época 5, seguida de un descenso en todas las métricas a partir de ahí con un criterio de parada de 10 épocas para la métrica mAP@50-95, el entrenamiento termina deteniéndose en la época 15.

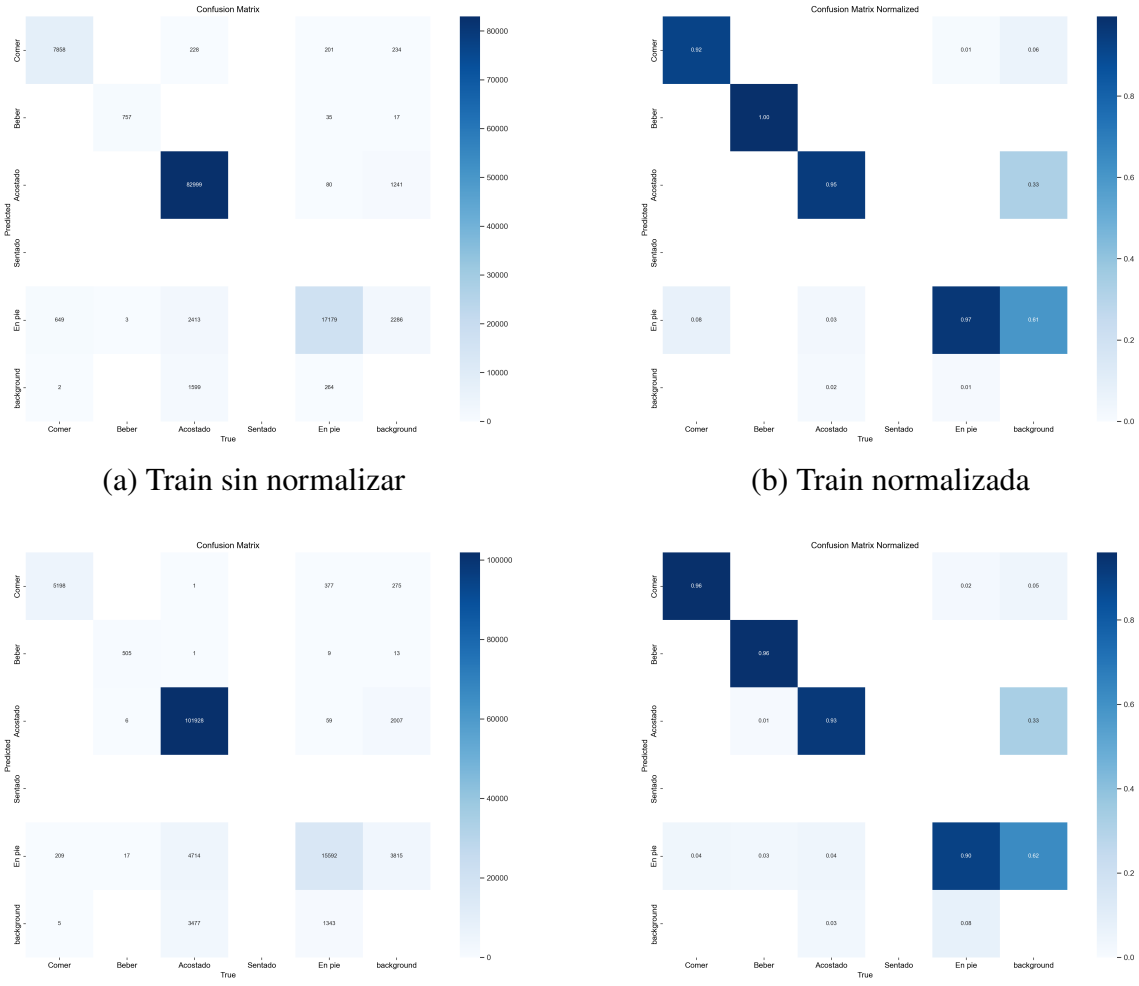
6.4.5. Prueba 5 - YOLOv10n

Finalmente, se realizó una prueba con un modelo superior para intentar mejorar los resultados anteriores. Pese a teóricamente tener mejor rendimiento y es más rápido que YOLOv8, las métricas globales de precisión, recall y mAP obtenidas no lo demuestran para mi dataset (ver Tabla 6.5).

Datos	Precisión	Recall	mAP@50	mAP@50-95
Train	0.935	0.954	0.974	0.682
Test	0.918	0.922	0.958	0.670

Tabla 6.5: Métricas globales para la prueba 5 – YOLOv10n

En esta prueba, las matrices de confusión obtenidas (Figura 6.11 muestran una detección de la clase *en pie* más confusa (columna *background*).



(a) Train sin normalizar

(b) Train normalizada

(c) Test sin normalizar

(d) Test normalizada

Figura 6.11: Matrices de confusión para la prueba 5.

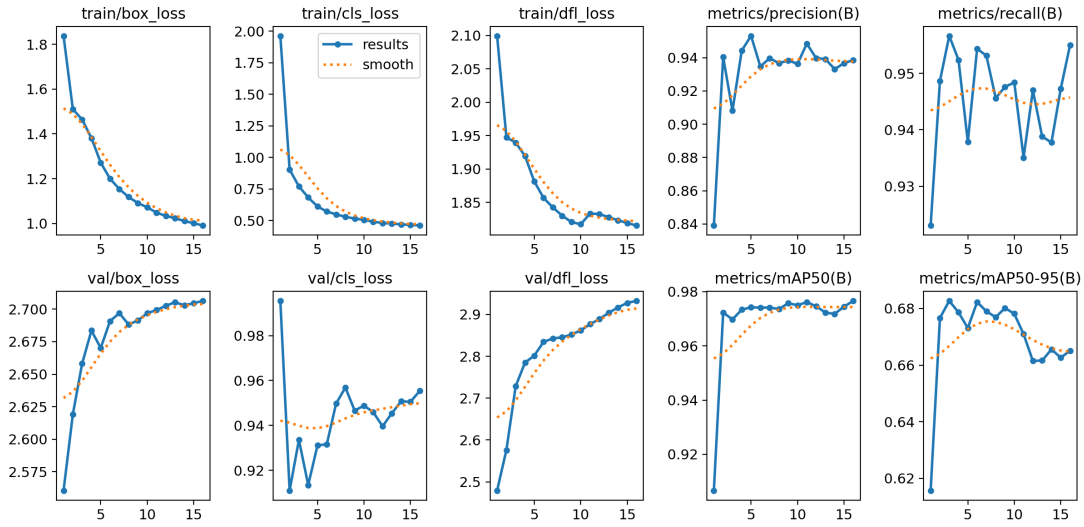


Figura 6.12: Métricas de entrenamiento y validación en la prueba 5.

En cuanto a los verdaderos positivos, hay cambios mínimos.

En la Figura 6.12 se observa que los valores de las pérdidas, tanto en entrenamiento como en test, son muy similares a las pruebas anteriores, siendo en las métricas donde hay mayores cambios, ya que, para esta prueba, se mantienen más estables con el paso de las épocas, teniendo la métrica mAP@50-95 (criterio de parada) un descenso significativo en la época 10, habiendo tenido su mejor valor en la época 6.

6.4.6. Comparativa global

Los resultados de la Tabla 6.6 permiten comparar las cinco estrategias evaluadas.

Prueba	Estrategia	Precisión	Recall	mAP@50	mAP@50-95
1	YOLOv8n (baseline)	0.749	0.733	0.763	0.528
2	YOLOv8n + augmentation	0.752	0.792	0.789	0.555
3	YOLOv8n (sentado modificado)	0.947	0.928	0.967	0.672
4	YOLOv8n (sentado eliminado)	0.946	0.923	0.965	0.671
5	YOLOv10n	0.918	0.922	0.958	0.670

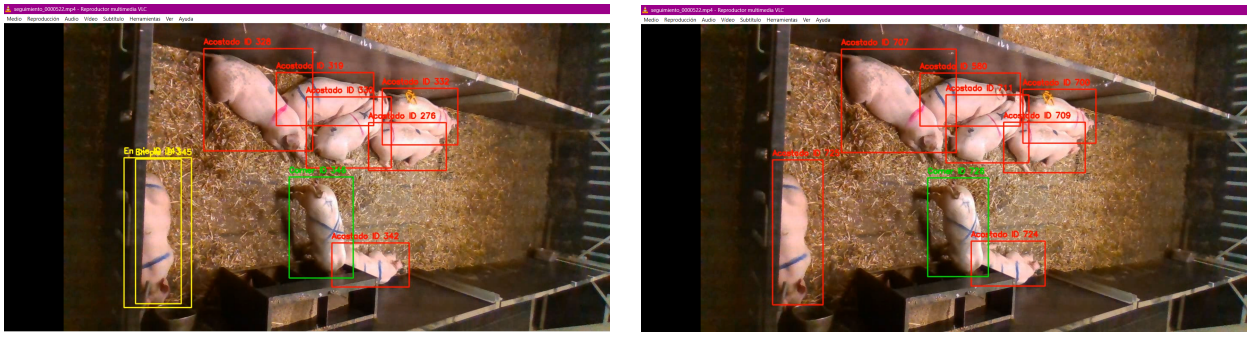
Tabla 6.6: Comparativa global de resultados en test

El primer modelo probado (YOLOv8n baseline) mostró limitaciones debido al desbalance en los datos, con un rendimiento bajo. Al añadir data augmentation (Prueba 2), hubo una mejora pequeña, lo que confirma que aumentar la variedad de datos ayuda, pero al ser aumentación automática, crecen todas las clases.

Las Pruebas 3 y 4, que modificaron la clase "sentado", tuvieron un efecto más notable:

- Prueba 3 (cambiar *sentado* por *en pie*) mejoró el rendimiento sobre la Prueba 2.
- Prueba 4 (eliminar *sentado* por completo) dio un resultado similar.

Esto sugiere que el problema principal era la ambigüedad de la clase *sentado*, que se parece tanto a *en pie* como a *acostado* y a la falta de más datos.



(a) YOLOv8n (b) YOLOv10n

Figura 6.13: Imagen comparativa del mismo vídeo etiquetado por el modelo.

Finalmente, aunque YOLOv10n (Prueba 5) no superó a YOLOv8n en las métricas, en la evolución práctica se observaron ventajas: menos errores en el seguimiento y detecciones más estables (ver Figura 6.13). Esto refuerza la idea de que, además de las métricas, hay que evaluar cómo funciona el modelo en situaciones reales.

En este sentido, se comprobó que YOLOv8n tendía a generar errores como asignar múltiples cajas a un mismo animal, mientras que YOLOv10n mostró una mayor estabilidad en la detección y seguimiento. Así, aunque con un dataset más amplio y balanceado es razonable pensar que YOLOv10n alcanzaría también mejores métricas cuantitativas, en este trabajo la elección final recae en YOLOv10n por su mayor fiabilidad en inferencia y por la calidad visual de los resultados en los vídeos procesados.

6.5. Inferencia

Una vez finalizada la fase de entrenamiento, se abordó la inferencia y el seguimiento de los individuos detectados en vídeo. El objetivo fue evaluar el comportamiento del sistema en condiciones más próximas a su aplicación real, comprobando la capacidad del modelo para mantener la coherencia en la identificación y el seguimiento de los animales a lo largo del tiempo.

6.5.1. Algoritmos de seguimiento evaluados

Inicialmente se implementó DeepSORT, pero este presentó problemas frecuentes de coherencia espacial y temporal: generación de cajas fantasma en regiones vacías, asignación errática de IDs y pérdida de seguimiento en movimientos simples (Figura 6.14a).

(a) Mediante DeepSORT (b) Mediante Bytetrack

Figura 6.14: Imagen comparativa de algoritmos de seguimiento para el mismo vídeo.

Ante estas limitaciones, se probó ByteTrack (Figura 6.14b), que mostró un rendimiento notablemente más estable. Para su configuración se exploraron distintos valores hasta fijar los siguientes parámetros finales:

```
tracker_args = argparse.Namespace(
    track_thresh=0.5,
    match_thresh=0.8,
    track_buffer=60,
    frame_rate=30,
    mot20=False
)
tracker = BYTETracker(tracker_args, frame_rate=30)
```

Con esta configuración, ByteTrack logró mantener la consistencia de los IDs a lo largo del tiempo (salvo en algunos casos de oclusiones o movimientos rápidos) y redujo la aparición de cajas superfluas o duplicadas.

6.5.2. Visualización y generación de alertas

Para facilitar la interpretación de los resultados, cada clase fue representada mediante un color específico en sus cajas delimitadoras. Además, se incorporó un sistema de generación de alertas superpuestas en el vídeo, indicando cuándo se producía un comportamiento de interés. Estas alertas se almacenaron simultáneamente en un log para su posterior análisis, permitiendo llevar un registro estructurado de los eventos detectados.

Se implementó un sistema de alertas, en el que cada cerdo en estado acostado durante al menos 3 segundos generaba un aviso en pantalla y en un fichero de registro. Este mecanismo, aunque simple, permitió validar la capacidad del sistema para detectar y registrar comportamientos de interés en tiempo real.

Así como se almacenan las alertas, también se almacenan las métricas de comportamiento para cada cerdo.

6.5.3. Elección final del modelo para inferencia

Aunque en términos numéricos YOLOv8n parecía ofrecer métricas superiores, las pruebas visuales en vídeo mostraron que YOLOv10n, combinado con ByteTrack, proporcionaba un resultado más fiable y coherente en la práctica. Esta elección se basó en:

- Mayor estabilidad en el seguimiento.
- Reducción de errores visuales como cajas duplicadas.
- Inferencia más consistente al analizar vídeos completos.

Por tanto, el sistema final se implementó con YOLOv10n + ByteTrack, usando el *dataset* con la clase *sentado* eliminada.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

En este capítulo se presenta una valoración global del Trabajo Final de Grado, analizando hasta qué punto se han alcanzado los objetivos definidos inicialmente, las dificultades encontradas a lo largo del desarrollo y los aprendizajes derivados de la experiencia.

7.1.1. Perspectiva del proyecto

El trabajo se planteó con una serie de metas que, en términos generales, se han logrado cumplir:

- **Objetivo 1.** Se construyó un sistema de detección de comportamientos en cerdos mediante técnicas de visión por computadora basadas en *Deep Learning*. Tras comparar distintos modelos de la familia YOLO, se seleccionó YOLOv10n como la mejor alternativa para la fase de inferencia, debido a su mayor estabilidad práctica en secuencias de vídeo.
- **Objetivo 2.** Se evaluaron e implementaron métodos de seguimiento de individuos a lo largo del tiempo. Aunque inicialmente se empleó DeepSORT, los problemas de coherencia en la asignación de identificadores llevaron a sustituirlo por ByteTrack, que mostró un rendimiento mucho más sólido y fiable con la configuración ajustada.
- **Objetivo 3.** Se integró un sistema de alertas capaz de señalar comportamientos relevantes en tiempo real. Estas notificaciones se mostraban sobre el vídeo y, al mismo tiempo, quedaban registradas en un archivo para su análisis posterior, junto con métricas asociadas a cada animal.

La estrategia de trabajo adoptada permitió avanzar de manera progresiva en cada etapa (entrenamiento, validación, inferencia y visualización), aunque no estuvo exenta de obstáculos:

- La escasez y desbalance de datos condicionó el aprendizaje del modelo, afectando especialmente a las clases con menor representación.
- El hardware limitado impuso restricciones en la configuración de los entrenamientos, aumentando los tiempos de cómputo y reduciendo las posibilidades de experimentación.
- La falta de conjuntos de datos públicos específicos obligó a trabajar con recursos reducidos y a realizar modificaciones para adaptarlos al objetivo del proyecto.

Pese a estas limitaciones, se alcanzó el propósito central: demostrar la viabilidad de un sistema que combina modelos de detección, algoritmos de seguimiento y un mecanismo de alertas para monitorizar comportamientos animales.

7.1.2. Perspectiva personal

Desde un punto de vista personal, este proyecto ha supuesto una experiencia de gran valor formativo. Algunos de los aprendizajes más significativos han sido:

- **Gestión de limitaciones reales:** abordar simultáneamente la escasez y el desbalance de datos y las restricciones de hardware, traduciéndolas en decisiones técnicas.
- Profundizar en el uso de modelos de *Deep Learning* para visión por computadora, así como en la integración de algoritmos de seguimiento y herramientas de visualización.
- Desarrollar competencias transversales relacionadas con la organización del trabajo, la documentación de resultados y la redacción de un informe académico con un enfoque claro y riguroso.

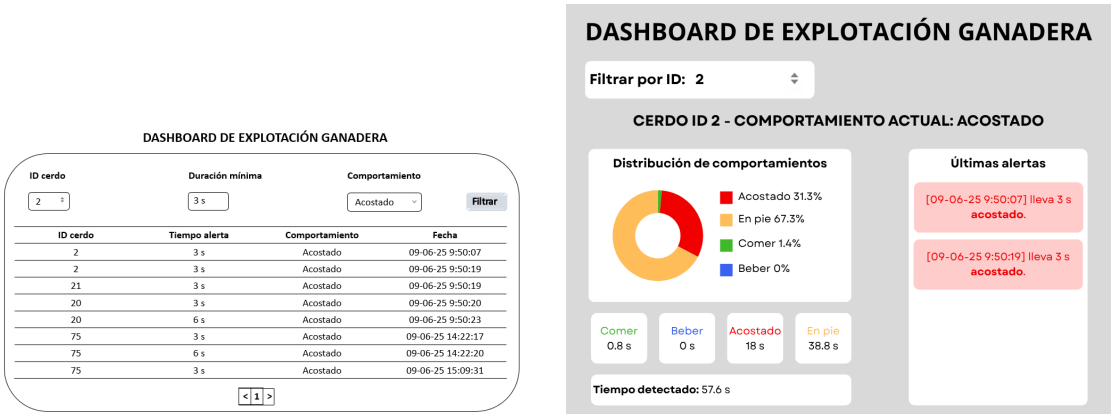
En suma, este TFG no solo ha permitido alcanzar los objetivos planteados, sino también adquirir una visión más realista de los retos que implica aplicar la inteligencia artificial a problemas prácticos.

7.2. Trabajo futuro

El proyecto realizado abre la puerta a diversas mejoras que podrían explorarse en el futuro:

- **Ampliación del conjunto de datos:** recopilar un mayor número de muestras y equilibrar la representación de cada clase, lo que reforzaría la capacidad del modelo para generalizar en distintos escenarios.
- **Procesamiento en tiempo real:** adaptar el sistema para operar directamente sobre streaming de vídeo, acercando la solución a un despliegue práctico.

- **Desarrollo de una interfaz gráfica para usuarios:** se propone una herramienta interactiva que facilite la visualización de resultados y métricas a los ganaderos. A modo de referencia, se han creado mockups preliminares que muestran cómo podría organizarse esta interfaz, respaldada por una base de datos para almacenar información histórica de cada animal (ver Figura 7.1).



(a) (b)

Figura 7.1: *Mockups* de posibles interfaces para mostrar métricas al ganadero.

- **Validación en entornos diversos:** evaluar el sistema en condiciones reales de uso, con variaciones de iluminación, instalaciones y razas, con el fin de comprobar su robustez en situaciones heterogéneas.
- **Complemento con sensores adicionales:** explorar la integración de datos de sensores de movimiento, temperatura o peso, lo que permitiría enriquecer la información disponible y aumentar la fiabilidad en la detección de anomalías.

Aunque muchas de estas propuestas exceden el alcance de este trabajo académico, representan una evolución natural de la línea de investigación, orientada hacia un sistema más completo y aplicable en la práctica.

Parte III

Apéndices

Apéndice A

Manuales de instalación

En este apartado se describen los procedimientos necesarios para la creación y configuración del entorno de trabajo empleado en el desarrollo del presente proyecto. El objetivo es garantizar la replicabilidad de los experimentos realizados, proporcionando al lector una guía clara y estructurada.

El entorno se preparó inicialmente en un ordenador portátil con Windows 11, y posteriormente fue replicado en un equipo de sobremesa con Windows 10. Esta comprobación permitió validar que la instalación y ejecución de los notebooks no dependían de un único entorno, sino que podían ser reproducidos en diferentes máquinas.

Cabe señalar que, aunque en el portátil se utilizó Anaconda 2.6.4 y en el sobremesa Anaconda 2.5.2, ambas versiones resultaron completamente funcionales para la ejecución de este trabajo.

Un aspecto importante a tener en cuenta es la instalación de la librería PyTorch, ya que depende de la compatibilidad con la tarjeta gráfica y la versión de CUDA disponible en el equipo. En caso de no disponer de GPU compatible, el entrenamiento puede ejecutarse alternativamente sobre CPU, aunque con un incremento significativo en los tiempos de cómputo.

A.1. Instalación de Anaconda

1. Descargar el instalador oficial desde la página de Anaconda [67].

2. Ejecutar el instalador seleccionando las opciones recomendadas:

   - Ruta de instalación personalizada.
   - Creación de acceso directo en el menú de inicio.

3. Abrir Anaconda Prompt para crear el entorno.

A.2. Creación y activación del entorno

Se creó un entorno denominado TFG con la versión de Python 3.8.20:

```
conda create -n TFG python=3.8.20 -y
conda activate TFG
```

A.3. Instalación de librerías necesarias

La instalación de librerías se realizó principalmente mediante pip. A continuación, se detallan las más relevantes, junto con su función en el proyecto y la versión utilizada:

- **Ultralytics (v8.3.81):** utilizada para la ejecución de YOLO en la detección de objetos.
- **Torch, Torchvision y Torchaudio:** constituyen la base para la construcción y entrenamiento de redes neuronales profundas. Se instalaron en la variante compatible con CUDA 12.1, adaptada a la GPU del equipo [68]:

  ```
  pip install torch torchvision torchaudio
  --index-url https://download.pytorch.org/whl/cu121
  ```

- **OpenCV-Python (v4.11.0.86):** para la lectura, manipulación y preprocesamiento de imágenes y vídeos.
- **SciPy (v1.10.1):** soporte para operaciones matemáticas avanzadas y funciones de optimización, necesarias para el algoritmo DeepSORT.
- **Matplotlib (v3.7.5) y Pillow (v10.4.0):** utilizadas en la visualización de resultados y en la manipulación básica de imágenes.
- **NumPy (v1.24.4):** librería fundamental para operaciones numéricas.
- **Deep-SORT-Realtime (v1.3.2):** implementación de DeepSORT para el seguimiento de objetos.
- **FilterPy (v1.4.5):** utilizada para la implementación del filtro de Kalman.

Previo a la instalación de Bytetrack, para evitar problemas con algunas de las librerías, se requiere la instalación predeterminada de Visual Studio Build Tools, siendo importante seleccionar los componentes “Desarrollo de escritorio con C++” y “Windows 10 SDK” o “Windows 11 SDK” dependiendo del sistema operativo.

Asimismo, fue necesaria la instalación de ByteTrack y sus dependencias adicionales:

```
pip install git+https://github.com/ifzhang/ByteTrack.git
pip install loguru cython pycocotools lap cython_bbox
```

Las versiones para las librerías de apoyo a Bytetrack anteriores son:

- loguru: 0.7.3.
- cython: 3.1.1.
- pycocotools: 2.0.7.
- lap: 0.5.12.
- cython_bbox: 0.1.5.

Finalmente, para asegurar la compatibilidad del entorno, fue necesario desinstalar las librerías onnx y onnxruntime:

```
pip uninstall onnx onnxruntime -y
```

A.4. Instalación de CVAT

Para la anotación de datos, se empleó CVAT, instalado mediante Docker y Git en Windows:

1. Descargar e instalar Git manteniendo la configuración por defecto [69].
2. Instalar Docker Desktop y reiniciar el equipo tras la instalación [70].
3. Clonar el repositorio oficial de CVAT desde GitHub:

   ```
   git clone https://github.com/cvat-ai/cvat.git
   cd cvat
   docker compose up -d
   ```

4. Verificar la correcta ejecución de los contenedores con:

   ```
   docker ps
   ```

5. Crear un superusuario para acceder a la interfaz:

   ```
   docker exec -it cvat_server bash
   python manage.py createsuperuser
   ```

6. Acceder a CVAT desde el navegador en http://localhost:8080.

Los pasos seguidos para la instalación de CVAT se basaron en la documentación oficial [71] y en material de apoyo audiovisual [72].

Apéndice B

Contenido adjunto

Con el objetivo de complementar la memoria del proyecto, se adjuntan los siguientes elementos:

- ***Notebook* final** con el código implementado para el entrenamiento y validación, así como la inferencia, la generación de alertas y métricas. El *dataset* no se entrega debido a su alto peso (alrededor de 150 GB):
  - seguimiento_comportamiento_cerdos.ipynb
- Scripts auxiliares desarrollados:
  - borrar_sentado.py
  - sustituir_sentado.py
  - descomprimir_organizar.py
  - dividir_dataset.py
  - unificar_eti_img.py

  Para evitar redundancia, únicamente se mencionan los nombres de cada script, ya que la explicación de cada uno se dio con anterioridad. Todos ellos se encuentran en el directorio /scripts.
- **Modelos entrenados**: en este aparatado se entregan los modelos de las 5 pruebas principales realizadas. Se entrega el fichero .pt de cada uno, ya que se comentaron los resultados de cada una en su correspondiente sección. Tienen nombres descriptivos cambiados a mano para poder diferenciar cada prueba.
  Todos ellos se encuentran en la carpeta /modelos, exceptuando el definitivo, que se encuentra en /train.
  - prueba_inical.pt.
  - prueba_data_augmentation.pt.

  - prueba_sentado_moficado.pt.
  - prueba_sentado_eliminado.pt.
  - best.pt.

- **Instaladores de programas** utilizados:

  - Docker Desktop Installer.exe.
  - Git-2.48.1-64-bit.exe.
  - Python-3.8.2-amd64.exe.
  - Visual Studio Build Tools.exe.
  - Anaconda3-2024.02-1-Windows-x86_64.

No se incluye el fichero .yaml de exportación del entorno de Anaconda, debido a que su importación resulta excesivamente lenta y no aporta ventajas frente a la instalación manual descrita en el Apéndice A.
Respecto a las pruebas realizadas, se conserva únicamente el notebook con la implementación final, ya que los experimentos intermedios no fueron archivados. Aunque lo ideal hubiera sido conservar todas las versiones, la información proporcionada resulta suficiente para asegurar la trazabilidad de los resultados.

Nota: Todos los enlaces de la webgrafía han sido comprobados una última vez el día 21/08/2025.

Webgrafía

[1] Ministerio de Agricultura, Pesca y Alimentación. *El sector de la carne de cerdo en cifras*. Accedido: 21/08/25. 2023. URL: https://www.mapa.gob.es/en/ganaderia/estadisticas/indicadoressectorporcino2023_tcm38-564427.pdf.

[2] Rotecna. *El sector porcino español, líder europeo*. Accedido: 21/08/25. 2024. URL: https://www.rotecna.com/blog/el-sector-porcino-espanol-lider-europeo-en-2024/.

[3] J. M. Herrero Medrano. *Tecnología al servicio del bienestar animal*. Accedido: 21/08/25. 2024. URL: https://porciforum.info/wp-content/uploads/2023/12/Juan-Manuel-Herrero-porciFORUM24.pdf.

[4] M. Vierto-Vega et al. *Machine Learning en la detección y predicción de enfermedades del ganado: una visión general*. Accedido: 21/08/25. 2024. URL: https://www.researchgate.net/publication/387028760_Machine_Learning_en_la_deteccion_y_prediccion_de_enfermedades_del_ganado_una_vision_general.

[5] D. Cartes et al. *Porcicultura de Precisión: una Tecnología al Servicio del Bienestar Animal*. Accedido: 21/08/25. 2023. URL: https://porcinews.com/porcicultura-de-precision-una-tecnologia-al-servicio-del-bienestar-animal/.

[6] X. Zhang et al. *Detección de objetos y análisis de estado de cerdos por aprendizaje profundo en cría de cerdos*. Accedido: 21/08/25. 2024. URL: https://doi.org/10.56294/sctconf2024.1211.

[7] Ultralytics. *Página de presentación de YOLO*. Accedido: 21/08/25. 2025. URL: https://docs.ultralytics.com/es/.

[8] L. Bergamini et al. "Extracting Accurate Long-Term Behavior Changes from a Large Pig Dataset". En: *Proc. Int. Conf. on Computer Vision Theory and Applications (VISAPP)*. Accedido: 21/08/25. Online, 8–10 Feb, 2021. URL: https://homepages.inf.ed.ac.uk/rbf/PIGDATA.

[9] Kanban Tool. *¿Qué es el Kanban personal?* Accedido: 21/08/25. URL: https://kanbantool.com/es/guia-kanban/el-personal-kanban.

[10] Pccomponentes. *Portátil utilizado para el proyecto*. Accedido: 21/08/25. URL: https://www.pccomponentes.com/hp-victus-15-fa0059ns-intel-core-i5-12450h-16gb-512gb-ssd-rtx-3050-156.

[11] Glassdoor. *Sueldos de Técnico de Datos*. Accedido: 21/08/25. URL: https://www.glassdoor.es/Sueldos/tecnico-de-datos-sueldo-SRCH_KO0,16.htm.

[12] Glassdoor. *Sueldos de Data Scientist*. Accedido: 21/08/25. URL: https://www.glassdoor.es/Sueldos/data-scientist-sueldo-SRCH_KO0,14.htm.

[13] Glassdoor. *Sueldos de Python Developer*. Accedido: 21/08/25. URL: https://www.glassdoor.es/Sueldos/python-developer-sueldo-SRCH_KO0,16.htm.

[14] S. V. Ranz. *¿Cuál es el coste de un trabajador para la empresa en 2025?* Accedido: 21/08/25. 2024. URL: https://www.grupocastilla.es/coste-trabajador/.

[15] Gobierno de España. *Real Decreto 159/2023, de 7 de marzo, en materia de Bienestar Animal*. Accedido: 21/08/25. 2023. URL: https://www.boe.es/buscar/doc.php?id=BOE-A-2023-6083.

[16] Rotecna. *El sector porcino español, líder europeo en 2024*. Accedido: 21/08/25. 2025. URL: https://www.rotecna.com/blog/el-sector-porcino-espanol-lider-europeo-en-2024/.

[17] E. B. Lagua et al. *A Comprehensive Monitoring System for Health Status and Behavior of Pigs Under Different Environmental Conditions Using Vision and Audio Technologies*. Accedido: 21/08/25. 2023. URL: https://www.researchgate.net/publication/373843674_A_Comprehensive_Monitoring_System_for_Health_Status_and_Behavior_of_Pigs_Under_Different_Environmental_Conditions_Using_Vision_and_Audio_Technologies.

[18] ITI. *Vacas conectadas. La ganadería 4.0 llega a granjas españolas*. Accedido: 21/08/25. 2019. URL: https://www.iti.es/noticias/vacas-conectadas-la-ganaderia-4-0-llega-a-granjas-espanolas/.

[19] Overstand. *Ganadería y producción animal inteligente en 2025*. Accedido: 21/08/25. 2025. URL: https://overstand.es/blog/post/ganaderia-inteligente-espana-2025.

[20] Comisión Europea. *The European Green Deal*. Accedido: 21/08/25. URL: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en.

[21] Comisión Europea. *Horizon Europe*. Accedido: 21/08/25. URL: https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe_en.

[22] E. Canorea. *Ganadería 4.0 y Smart Agriculture: tecnologías*. Accedido: 21/08/25. URL: https://www.plainconcepts.com/es/ganaderia-agricultura-tecnologias/.

[23] Club Ganaderías. *Industria ganadera 4.0: perspectivas y desafíos*. Accedido: 21/08/25. URL: https://www.clubganadero.com/industria-ganadera/.

[24] Nutrinews. *Tecnología 4.0 en ganadería: Impulso a la eficiencia nutricional del ganado*. Accedido: 21/08/25. 2025. URL: https://nutrinews.com/tecnologia-4-0-ganaderia-innovacion-eficiencia-nutricional/.

[25] SAP. *¿Qué es Machine Learning?* Accedido: 21/08/25. URL: https://www.sap.com/spain/products/artificial-intelligence/what-is-machine-learning.html.

[26] E. Alpaydin. *Introduction to Machine Learning*. 4th. Accedido: 21/08/25. Cambridge, MA: MIT Press, 2020. URL: https://books.google.es/books?hl=es&lr=&id=uZnSDwAAQBAJ.

[27] I. Belcic y C. Stryker. *¿Qué es el aprendizaje supervisado?* Accedido: 21/08/25. 2024. URL: https://www.ibm.com/es-es/think/topics/supervised-learning.

[28] IBM. *¿Qué es el aprendizaje no supervisado?* Accedido: 21/08/25. 2021. URL: https://www.ibm.com/es-es/think/topics/unsupervised-learning.

[29] Ultralytics. *Aprendizaje no supervisado*. Accedido: 21/08/25. URL: https://www.ultralytics.com/es/glossary/unsupervised-learning.

[30] J. Murel y E. Kavlakoglu. *¿Qué es el aprendizaje de refuerzo?* Accedido: 21/08/25. 2024. URL: https://www.ibm.com/es-es/think/topics/reinforcement-learning.

[31] D. G. Rameshkumar y S. Samundeswari. "Neural network, artificial neural network (ANN) and biological neural network (BNN) in soft computing". En: *Volume* 30.3 (2014). Accedido: 21/08/25, pág. 3. URL: https://d1wqtxts1xzle7.cloudfront.net/33346172/11-libre.pdf.

[32] Wikipedia. *Neurona artificial*. Accedido: 21/08/25. URL: https://es.wikipedia.org/wiki/Neurona_artificial.

[33] M. All. *Introducción a las funciones de activación en las redes neuronales*. Accedido: 21/08/25. 2024. URL: https://www.datacamp.com/es/tutorial/introduction-to-activation-functions-in-neural-networks.

[34] UNIE Universidad. *¿Qué son las redes neuronales y cómo se aplican a la Inteligencia Artificial?* Accedido: 21/08/25. 2024. URL: https://www.universidadunie.com/blog/que-son-redes-neuronales.

[35] IBM. *¿Qué es el descenso del gradiente?* Accedido: 21/08/25. URL: https://www.ibm.com/es-es/think/topics/gradient-descent.

[36] GAMCO. *¿Qué es Retropropagación?* Accedido: 21/08/25. URL: https://gamco.es/glosario/retropropagacion.

[37] J. Brownlee. *What is the difference between a batch and an epoch in a neural network*. Accedido: 21/08/25. 2018. URL: https://deeplearning.lipingyang.org/wp-content/uploads/2018/07/What-is-the-Difference-Between-a-Batch-and-an-Epoch-in-a-Neural-Network_.pdf.

[38] IBM. *¿Qué son las redes neuronales?* Accedido: 21/08/25. URL: https://www.ibm.com/es-es/think/topics/neural-networks.

[39] Xeridia. *Redes Neuronales Artificiales: Qué son y cómo se entrenan*. Accedido: 21/08/25. 2019. URL: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i.

[40] I. P. Borrero y M. E. G. Arias. *Deep learning*. Vol. 19. Accedido: 21/08/25. Huelva, España: Servicio de Publicaciones de la Universidad de Huelva, 2021. URL: https://books.google.es/books?hl=es&lr=&id=kzsvEAAAQBAJ&oi=fnd&pg=PA1.

[41] J. Holdsworth y M. Scapicchio. *¿Qué es el deep learning?* Accedido: 21/08/25. 2024. URL: https://www.ibm.com/es-es/think/topics/deep-learning.

[42] A. A. Awan. *Tutorial de redes neuronales recurrentes (RNN)*. Accedido: 21/08/25. 2024. URL: https://www.datacamp.com/es/tutorial/tutorial-for-recurrent-neural-network.

[43] A. A. Awan. *¿Qué son los datos etiquetados?* Accedido: 21/08/25. 2024. URL: https://www.datacamp.com/es/blog/what-is-labeled-data.

[44] Amazon Web Services (AWS). *¿Qué es el etiquetado de datos?* Accedido: 21/08/25. URL: https://aws.amazon.com/es/what-is/data-labeling.

[45] J. I. Bagnato. *Modelos de Detección de Objetos*. Accedido: 21/08/25. 2020. URL: https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos.

[46] Visionplatform. *La guía completa para la detección de objetos: Una introducción a la detección en 2024*. Accedido: 21/08/25. 2024. URL: https://visionplatform.ai/es/la-guia-completa-para-la-deteccion-de-objetos-una-introduccion-a-la-deteccion-en-2024.

[47] IBM. *¿Qué es la detección de objetos?* Accedido: 21/08/25. 2024. URL: https://www.ibm.com/es-es/think/topics/object-detection.

[48] Ultralytics. *Documentación de Ultralytics YOLO*. Accedido: 21/08/25. 2024. URL: https://docs.ultralytics.com/es/#where-to-start.

[49] Innovatiana. *Seguimiento de objetos: una tecnología en el corazón de la visión automatizada*. Accedido: 21/08/25. 2024. URL: https://www.innovatiana.com/es/post/object-tracking-in-ai.

[50] FlyPix AI. *Seguimiento de objetos mediante aprendizaje profundo: una guía completa*. Accedido: 21/08/25. 2025. URL: https://flypix.ai/es/deep-learning-object-tracking.

[51] M. Méndez. *Multi-Object Tracking by Detection: A Comprehensive Guide*. Accedido: 21/08/25. 2023. URL: https://miguel-mendez-ai.com/2023/11/08/tracking-by-detection-overview.

[52] TRBL Services. *OpenCV y Deep Learning – Visión por computador en redes neuronales*. Accedido: 21/08/25. 2023. URL: https://trbl-services.eu/blog-opencv-y-deep-learning-vision-por-computador-en-redes-neuronales.

[53] Tryolabs. *Norfair*. Accedido: 21/08/25. URL: https://tryolabs.github.io/norfair/2.2.

[54] Meta AI. *Detectron2*. Accedido: 21/08/25. 2021. URL: https://github.com/facebookresearch/detectron2.

[55] Ultralytics. *Instalar Ultralytics*. Accedido: 21/08/25. 2024. URL: https://docs.ultralytics.com/es/quickstart.

[56] PyTorch. *PyTorch*. Accedido: 21/08/25. URL: https://pytorch.org/projects/pytorch.

[57] Tesla. *Piloto automático y Capacidad de conducción autónoma total*. Accedido: 21/08/25. URL: https://www.tesla.com/es_es/support/autopilot.

[58] J. Ramey. *Tesla Bets on AI in Latest FSD Update*. Accedido: 21/08/25. 2024. URL: https://www.autoweek.com/news/a46535912/tesla-fsd-ai-neural-networks-update.

[59] Wikipedia. *Tesla Autopilot*. Accedido: 21/08/25. URL: https://es.wikipedia.org/wiki/Tesla_Autopilot.

[60] Amazon Web Services (AWS). *Amazon Just Walk Out*. Accedido: 21/08/25. URL: https://aws.amazon.com/es/just-walk-out.

[61] P. Llorente. *Amazon Go, ¿De qué se trata el ambicioso proyecto de Amazon?* Accedido: 21/08/25. 2020. URL: https://www.esic.edu/rethink/comercial-y-ventas/que-es-amazon-go-%20y-como-funciona.

[62] S. Tobar. *Amazon Go y sus copias no son el futuro del 'súper'...* Accedido: 21/08/25. 2025. URL: https://www.elespanol.com/invertia/empresas/distribucion/20250427/amazon-go-copias-no-futuro-super-tiendas-sin-colas-cajeros-iban-cambiar-retail/1003743729397_0.html.

[63] Amazon. *Amazon Go es un nuevo tipo de tienda de barrio*. Accedido: 21/08/25. URL: https://www.amazon.com/b?node=16008589011.

[64] Smart Parks. *What is a Smart Park?* Accedido: 21/08/25. URL: https://www.smartparks.org.

[65] K. Bjerge, H. M. R. Mann y T. T. Høye. *Real-time insect tracking and monitoring with computer vision and deep learning*. Accedido: 21/08/25. 2021. URL: https://zslpublications.onlinelibrary.wiley.com/doi/10.1002/rse2.245.

[66] S. Friederich. *Fine-tuning*. Accedido: 21/08/25. 2017. URL: https://pure.rug.nl/ws/portalfiles/portal/49767863/Fine_tuning.pdf.

[67] Anaconda. *Download*. Accedido: 21/08/25. URL: https://www.anaconda.com/download.

[68] PyTorch. *Get Started Locally*. Accedido: 21/08/25. URL: https://pytorch.org/get-started/locally.

[69] Git. *Downloads for Windows*. Accedido: 21/08/25. URL: https://git-scm.com/downloads/win.

[70] Docker. *Docker Desktop*. Accedido: 21/08/25. URL: https://www.docker.com/products/docker-desktop.

[71] CVAT. *Installation basics*. Accedido: 21/08/25. URL: https://docs.cvat.ai/docs/administration/basics/installation.

[72] CVAT. *How to install CVAT on Windows using Docker*. Accedido: 21/08/25. 2023. URL: https://www.youtube.com/watch?v=hyvNFuc06qQ.

[73] J. Everitt. *La guía completa de Kanban personal*. Accedido: 21/08/25. 2021. URL: https://www.wrike.com/es/blog/la-guia-completa-de-kanban-personal/.

[74] M. Arias. *Redes neuronales artificiales: fundamentos y aplicaciones*. Accedido: 21/08/25. 2025. URL: https://evidenciasenpediatria.es/articulo/8518/redes-neuronales-artificiales-fundamentos-y-aplicaciones.

[75] F. Izaurieta y C. Saavedra. *Redes neuronales artificiales*. Accedido: 21/08/25. Concepción, Chile: Departamento de Física, Universidad de Concepción, 2000. URL: https://d1wqtxts1xzle7.cloudfront.net/36957207/Redes_neuronales-libre.pdf.