UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Detección de personas perdidas mediante vehículos aéreos no tripulados

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Botea, Sabin Costin

Tutor/a: Tavares de Araujo Cesariny Calafate, Carlos Miguel

Cotutor/a: Wubben, Jamie

CURSO ACADÉMICO: 2024/2025

Resumen

Los drones no tripulados son una tecnología actualmente en auge. En los últimos años, han experimentado avances significativos en autonomía y capacidad de cómputo, lo que permite integrarlos con técnicas de inteligencia artificial (IA) en aplicaciones cada vez más complejas. En este contexto, el presente proyecto desarrolla una herramienta de inteligencia artificial orientada a la detección de personas mediante un dron equipado con una cámara, utilizando redes neuronales convolucionales (CNN).

El bajo coste, la portabilidad y la capacidad de operar en tiempo real hacen a esta combinación de tecnologías especialmente útiles en situaciones de emergencia o en entornos sin infraestructura tecnológica, facilitando una respuesta rápida y autónoma ante eventos críticos.

La red se entrena con un gran conjunto de imágenes, capturadas en entornos variados con distintas condiciones de iluminación y paisajes, lo que mejora la robustez y capacidad de generalización del modelo. Para la detección, se emplea una versión actualizada del algoritmo YOLO (You Only Look Once).

Se llevan a cabo pruebas en escenarios reales para evaluar el rendimiento del modelo en diferentes condiciones de iluminación, altitud de vuelo y nitidez.

Este proyecto se enmarca en el creciente interés por soluciones basadas en Deep Learning aplicadas a sistemas autónomos, y pone de manifiesto el potencial de la inteligencia artificial en tareas como la vigilancia, la búsqueda y el rescate.

Palabras clave: Inteligencia artificial, redes neuronales convolucionales, YOLO, Deep Learning

Resum

Els drons no tripulats són una tecnologia actualment en auge. En els últims anys, han experimentat avanços significatius en autonomia i capacitat de càlcul, fet que permet integrar-los amb tècniques d'intel·ligència artificial (IA) en aplicacions cada vegada més complexes. En este context, el present projecte desenvolupa una eina d'intel·ligència artificial orientada a la detecció de persones mitjançant un dron equipat amb una càmera, utilitzant xarxes neuronals convolucionals (CNN).

El baix cost, la portabilitat i la capacitat d'operar en temps real fan que esta combinació de tecnologies siga especialment útil en situacions d'emergència o en entorns sense infraestructura tecnològica, facilitant una resposta ràpida i autònoma davant esdeveniments crítics.

La xarxa s'entrena amb un conjunt gran d'imatges capturades en entorns variats, amb diferents condicions d'il·luminació i paisatges, cosa que millora la robustesa i la capacitat de generalització del model. Per a la detecció, s'utilitza una versió actualitzada de l'algorisme YOLO (You Only Look Once).

Es duen a terme proves en escenaris reals per a avaluar el rendiment del model en diferents condicions d'il·luminació, altitud de vol i nitidesa.

Este projecte s'emmarca en l'interés creixent per les solucions basades en el Deep Learning aplicades a sistemes autònoms, i posa de manifest el potencial de la intel·ligència artificial en tasques com ara la vigilància, la busca i el rescat.

Paraules clau: Intel·ligència artificial, xarxes neuronals convolucionals, YOLO, Deep Learning

Abstract

Unmanned drones are a rapidly growing technology. In recent years, they have experienced significant advances in autonomy and computational power, enabling their integration with artificial intelligence (AI) techniques in increasingly complex applications. In this context, the present project develops an AI-based tool aimed at person detection using a drone equipped with a camera and employing convolutional neural networks (CNNs).

The low cost, portability, and real-time operating capabilities of this technology make it especially useful in emergency situations or in environments lacking technological infrastructure, allowing for a rapid and autonomous response to critical events.

The neural network is trained on a large dataset of images captured in diverse environments with varying lighting conditions and landscapes, improving the model's robustness and generalization ability. For detection, an updated version of the YOLO (You Only Look Once) algorithm is used.

Real-world tests are conducted to evaluate the model's performance under different lighting conditions, flight altitude, and image clarity.

This project aligns with the growing interest in Deep Learning-based solutions applied to autonomous systems and highlights the potential of AI in tasks such as surveillance, search, and rescue.

Key words: Artificial intelligence, convolutional neural networks, YOLO, Deep Learning

Índice general

Apéndices

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

En los últimos años la inteligencia artificial (IA) ha experimentado un crecimiento exponencial, transformando numerosos sectores gracias a su capacidad para analizar grandes volúmenes de datos, aprender patrones complejos, y tomar decisiones de manera autónoma. Uno de los ámbitos donde la IA ha comenzado a tener un impacto significativo es, tal y como se afirma en [1], en las operaciones de búsqueda y rescate, donde la rapidez en la localización de personas puede marcar la diferencia entre la vida y la muerte.

Eventos recientes, como terremotos, incendios forestales, inundaciones y otras catástrofes naturales, han puesto de manifiesto la necesidad urgente de tecnologías que apoyen a los equipos de emergencia en tareas críticas. Las noticias sobre el uso de drones y algoritmos de visión por computador en estos contextos son cada vez más frecuentes, mostrando cómo los sistemas automatizados pueden cubrir grandes áreas en tiempos reducidos, así como acceder a zonas de difícil alcance para el ser humano. Por ello, la detección automática de personas a través de cámaras aéreas se convierte en una herramienta clave, ya que permite identificar con rapidez la presencia de individuos atrapados o en peligro.

La combinación de vehículos aéreos no tripulados (drones) con algoritmos avanzados de inteligencia artificial, como las redes neuronales convolucionales (CNN) y modelos de detección como YOLO (You Only Look Once), representa una solución prometedora para mejorar la eficacia de los operativos de rescate. Estas tecnologías permiten no solo la captación de imágenes en tiempo real, sino también el análisis y procesamiento automático de la información para tomar decisiones de forma inmediata.

Aunque se han logrado avances notables, aún existen desafíos técnicos importantes relacionados con la precisión del reconocimiento en condiciones ambientales adversas, la limitación de recursos computacionales en dispositivos embarcados, y la necesidad de adaptar los modelos a escenarios no estructurados. Estos retos abren un espacio claro para la investigación y desarrollo de soluciones personalizadas que optimicen el uso de IA en situaciones reales.

El presente proyecto surge como respuesta a esta necesidad, con la intención de contribuir al desarrollo de herramientas inteligentes aplicadas a la detección de personas desde drones. Aprovechando las capacidades actuales de las redes neuronales profundas y el poder de procesamiento de dispositivos al borde de la red (*edge*), se busca construir un sistema eficiente y funcional que pueda ser aplicado en contextos reales, sirviendo como base para futuras investigaciones y aplicaciones en el campo humanitario.

1.2 Objetivos

Entrenar un sistema de detección de personas presenta una serie de desafíos importantes. En primer lugar, la variabilidad de los escenarios en los que puede encontrarse una persona (diferente ropa, posturas, tamaños, obstáculos, etc.) obliga a construir un conjunto de datos lo suficientemente diverso para que el modelo aprenda a generalizar correctamente. Además, la precisión del sistema no solo depende de la calidad del entrenamiento, sino también de su capacidad para adaptarse a imágenes captadas desde ángulos y alturas poco convencionales, como las que toma un dron en movimiento.

Por otro lado, la implementación del sistema en un dispositivo *edge* implica limitaciones importantes en términos de potencia de cálculo y consumo energético. A diferencia de los ordenadores tradicionales o servidores con GPU, estos dispositivos requieren modelos compactos y eficientes, capaces de ejecutarse en tiempo real sin agotar los recursos disponibles. Esto obliga a aplicar técnicas de optimización, como la selección de versiones ligeras de redes neuronales y el uso de *frameworks* adaptados a *hardware* embebido.

El objetivo principal de este proyecto es desarrollar un sistema con la capacidad de reconocer personas en entornos reales desde una perspectiva zenital mediante técnicas de visión por computador basadas en inteligencia artificial. Esta herramienta está pensada para ser utilizada en situaciones donde la intervención humana directa es difícil, peligrosa o poco eficiente, como en catástrofes naturales, zonas de difícil acceso, o búsquedas en espacios abiertos. A través de un modelo de red neuronal se busca mejorar la eficiencia de las operaciones de búsqueda y rescate, ofreciendo una solución ágil, autónoma y de bajo coste.

Para alcanzar este objetivo general, se han establecido los siguientes objetivos específicos:

- Diseñar y entrenar un modelo de detección de personas basado en redes neuronales convolucionales, que sea capaz de identificar con precisión la presencia humana en imágenes captadas desde el aire, incluso en condiciones variables de iluminación, perspectiva y fondo.

- Validar el sistema mediante pruebas en entornos reales, comprobando su capacidad para detectar personas en diferentes tipos de terreno, distancias y condiciones ambientales, y evaluando métricas de rendimiento como precisión y tasa de falsos positivos.

- Crear un conjunto de datos personalizado, compuesto por multitud de imágenes etiquetadas adecuadamente, que refleje escenarios realistas y diversos, como campos abiertos, bosques, caminos y algunas zonas urbanas.

- Ofrecer opciones de despliegue del modelo para su ejecución en dispositivos con recursos limitados, lo que puede implica reducir el tamaño del modelo, ajustando su arquitectura y balanceando el rendimiento entre precisión y velocidad de inferencia.

1.3 Estructura de la memoria

En este apartado se pretende desglosar los contenidos del presente documento y explicar de forma breve cada capítulo.

- **Capítulo 1:** Introducción al proyecto, exponiendo los motivos que justifican su realización y los objetivos que se pretenden alcanzar
- **Capítulo 2:** Estado del arte de las principales tecnologías de interés que envuelven al proyecto. Se analiza la evolución de los drones, de las redes neuronales, y de las técnicas de procesamiento de imágenes.
- **Capítulo 3:** Se analizan los problemas a los que nos enfrentamos, desde buscar un *dataset* completo que refleje condiciones reales, a la elección del modelo de inteligencia artificial y el *hardware* utilizado para su entrenamiento.
- **Capítulo 4:** En este capítulo se desarrolla la solución al problema en base al análisis anterior.
- **Capítulo 5:** Se pone a prueba los modelos entrenados sobre un conjunto de datos captados por un dron y se ofrecen opciones de despliegue.
- **Capítulo 6:** Corresponde al capítulo final, donde se discuten los objetivos alcanzados, se relaciona el proyecto con las asignaturas cursadas y se plantea el posible trabajo a futuro.

CAPÍTULO 2
Estado del arte

2.1 Sobre los drones

2.1.1. Los primeros drones

Un vehículo aéreo no tripulado (VANT), UAV (del inglés unmanned aerial vehicle)[2], comúnmente conocido como dron, hace referencia a una aeronave que vuela sin tripulación y ejerce su función de forma remota.

La evolución de los drones ha sido una historia de innovación constante, marcada por avances tecnológicos significativos que han transformado su uso desde aplicaciones militares hasta una amplia gama de sectores civiles.

El primer uso de un UAV, en el sentido más amplio, data del año 1849 [3] con el uso de globos incendiarios por parte del ejercito austriaco en el asedio a Venecia. Estos dispositivos volaban gracias a la fuerza del viento y no contaban con ningún sistema de guiado. De 200 globos lanzados, solamente uno logró impactar en el objetivo.

A principios del siglo XX, con la invención del primer aeroplano y el estallido de la primera guerra mundial, se profundizó en las tecnologías de aviación y radio. En 1916 el ingeniero británico Archibald Low desarrolló el *Aerial Target* (ver figura 2.1), considerado el primer dron propulsado, utilizando sistemas de control por radio. Aunque no se desplegó en combate, sentó las bases para futuros desarrollos en vehículos aéreos no tripulados

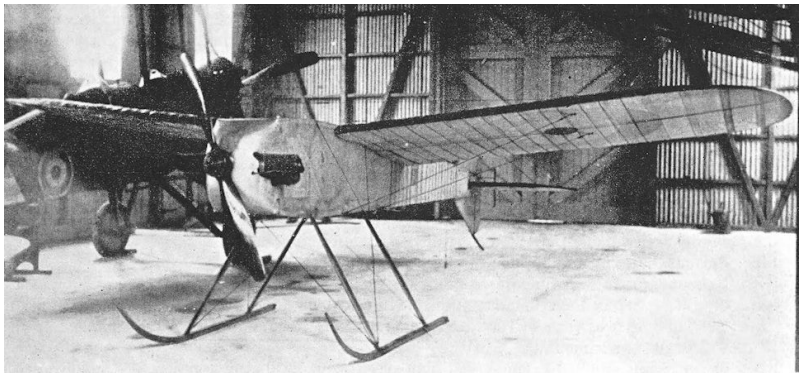


Figura 2.1: Aerial Target. Fuente: flyingmachines.ru

En los años 30, el Reino Unido introdujo el "Queen Bee", un avión teledirigido utilizado como blanco para prácticas de tiro, dando origen al término "drone". Simultánea-

mente, en Estados Unidos se desarrolló el Radioplane OQ-2", el primer UAV producido en masa, con más de 15,000 unidades fabricadas durante la Segunda Guerra Mundial.

La siguiente gran revolución tecnológica tuvo lugar durante la Guerra Fría, donde se impulsó el desarrollo de drones con capacidades avanzadas. El Ryan Firebee", introducido en 1951, fue uno de los primeros drones a reacción, utilizado principalmente como blanco aéreo y para misiones de reconocimiento.

En las décadas de 1970 y 1980, la miniaturización de componentes electrónicos y la incorporación de sistemas GPS permitieron la creación de drones más autónomos y precisos. El "Scout", desarrollado por Israel Aerospace Industries, fue uno de los primeros UAVs en utilizar navegación GPS y sistemas de comunicación digital.

La introducción de microprocesadores potentes y sensores avanzados en los años 90 permitió el desarrollo de drones como el "MQ-1 Predator", utilizado por las fuerzas armadas de EE.UU. para misiones de vigilancia y ataques dirigidos. En el ámbito civil, empresas como DJI y Parrot popularizaron los drones de consumo, equipados con cámaras de alta resolución y sistemas de navegación avanzados, ampliando su uso a la fotografía aérea, agricultura de precisión y monitoreo ambiental.

2.1.2. Drones en la actualidad

Aunque el origen y desarrollo inicial de los drones estuvo estrechamente ligado a fines militares, en la actualidad su uso se ha extendido ampliamente al ámbito civil. Estas aeronaves no tripuladas han ganado una gran popularidad en todo el mundo gracias a su versatilidad, facilidad de manejo y eficacia para realizar tareas en una amplia variedad de sectores.

En España, desde la aprobación del primer Real Decreto[4] que regulaba el uso de drones en 2014, el número de pilotos y operadores registrados en la Agencia Estatal de Seguridad Aérea (AESA) no ha dejado de aumentar. Esta creciente demanda ha impulsado la homologación de cursos y másteres universitarios que permiten obtener licencias de vuelo, aunque con las limitaciones impuestas por la legislación vigente, como la prohibición de volar sobre áreas urbanas, aglomeraciones de personas o en horario nocturno. Se prevé que futuras normativas flexibilicen algunas de estas restricciones.

La utilidad práctica de los vehículos aéreos no tripulados está siendo explorada en diversos campos. Uno de los usos emergentes es el reparto automatizado de paquetes. En España, CORREOS[5] ya ha realizado pruebas piloto en este sentido (ver figura 2.2). En Estados Unidos, compañías como Amazon experimentan con entregas de comida y medicamentos mediante drones, mientras que en Ruanda[6] se estudia la creación de bases logísticas para distribuir productos médicos. Francia, Suiza y Reino Unido también están probando estos sistemas, aunque aún existen desafíos técnicos y normativos, especialmente relacionados con la gestión del tráfico aéreo.

Figura 2.2: Dron de Correos. Fuente: correos.com

La capacidad de los drones para sobrevolar terrenos irregulares o de difícil acceso, esquivando obstáculos y proporcionando imágenes aéreas y datos de sensores, los hace ideales para múltiples tareas.

En el ámbito agrícola [7] se emplean para cartografiar terrenos, detectar incendios, vigilar cultivos y ganado, realizar fumigaciones aéreas, y localizar plagas o malas hierbas, contribuyendo así a una agricultura más precisa y eficiente.

Otras aplicaciones incluyen la colaboración en investigaciones arqueológicas y geológicas, como la toma de muestras en volcanes, y en meteorología, mediante la recopilación de datos sobre temperatura y humedad. En el sector sanitario, los drones también están adquiriendo un papel relevante: desde la recolección de polen para el estudio de alergias hasta la entrega de suministros en situaciones de emergencia médica, ayudando a reducir considerablemente los tiempos de respuesta.

2.1.3. Tipos de drones

Dada la diversidad de aplicaciones —- tanto civiles como militares —- y la descentralización en el desarrollo tecnológico de los drones, no existe actualmente una clasificación única y universalmente aceptada. Por ello, muchos especialistas optan por organizar los vehículos aéreos no tripulados en función de sus características operativas o técnicas. A continuación, se presentan algunas posibles clasificaciones inspiradas por algunos autores [8][9][3] .

Clasificación por el origen de la misión del dron

- **Drones civiles**
  - Utilizados en sectores como:
    - Agricultura de precisión.
    - Inspección de infraestructuras.
    - Cartografía y topografía.
    - Entrega de paquetes y logística.
    - Fotografía y vídeo aéreo.
    - Vigilancia medioambiental.
    - Apoyo en operaciones de emergencia y rescate.
- **Drones militares**
  - Empleados en operaciones como:
    - Inteligencia, vigilancia y reconocimiento (ISR).

    - Ataques de precisión y apoyo en combate.
    - Logística y transporte de suministros.
  - Incorporan sistemas avanzados de navegación, comunicaciones seguras y, en algunos casos, armamento.

Clasificación según la tecnología de los motores utilizados

- Drones con motor alternativo**:** Utilizan motores de combustión interna de tipo alternativo (pistón), similares a los de automóviles o avionetas ligeras.
- Drones con turbina: Emplean motores a reacción o turbinas, principalmente en modelos militares o de altas prestaciones, capaces de alcanzar mayores velocidades y altitudes.
- Drones eléctricos: Funcionan con motores eléctricos alimentados por baterías. Son los más comunes actualmente debido a su eficiencia, menor ruido, y mantenimiento reducido.

Clasificación según la altura de vuelo

- Alta o muy alta cota: Operan por encima de los 9.000 metros. Suelen ser utilizados para misiones estratégicas de vigilancia, reconocimiento y comunicaciones. En esta categoría se encuentran algunos drones militares de gran autonomía.
- Media cota: Vuelan entre los 1.000 y 9.000 metros de altitud. Se emplean tanto en operaciones militares como civiles, incluyendo observación aérea, seguimiento de fenómenos meteorológicos o misiones de búsqueda y rescate.
- Baja cota : Vuelan por debajo de los 1.000 metros. Son los más comunes en aplicaciones comerciales y recreativas, como fotografía aérea, agricultura de precisión, inspección de infraestructuras, o entregas de última milla.

Clasificación según el tipo de control

- Autónomo y adaptativo: El vehículo aéreo está completamente gestionado por sus propios sistemas embarcados. Posee la capacidad de modificar su ruta o comportamiento en función de cambios en el entorno, gracias a la información captada por sus sensores.
- Autónomo monitorizado: Aunque el UAV opera de forma autónoma, un operador supervisa su actividad desde tierra. Este no controla directamente el vuelo, pero puede intervenir en la toma de decisiones si es necesario.
- Supervisado: El dron ejecuta algunas tareas automáticamente, pero la mayor parte del control recae en el operador humano.
- Autónomo no adaptativo: El UAV sigue una ruta o misión preprogramada sin capacidad para modificarla, incluso si cambian las condiciones externas.
- Controlado por mando directo (R/C): El operador en tierra tiene el control total del UAV mediante control remoto, actuando directamente sobre sus movimientos en tiempo real.

Clasificación según la estructura del dron

- Ala fija: Similares a pequeños aviones, aprovechan la aerodinámica para elevarse. Son ideales para vuelos de largo alcance y uso en agricultura, fotogrametría o vigilancia.
- Ala rotatoria (multirrotores): Obtienen sustentación mediante hélices. Versátiles y estables, se usan para grabaciones, seguridad y tareas en espacios reducidos. Se subdividen en:
  - Tricóptero (3 hélices)
  - Cuadricóptero (4 hélices)
  - Hexacóptero (6 hélices)
  - Octocóptero (8 hélices)
  - Coaxial (múltiples hélices en el mismo eje)

2.2 Sobre las redes neuronales

2.2.1. ¿Qué es una red neuronal?

Las redes neuronales[10], también conocidas como redes neuronales artificiales, son modelos computacionales inspirados en el funcionamiento del cerebro humano. Se utilizan para identificar patrones y modelar relaciones complejas entre datos de entrada y salida. De forma más precisa, una red neuronal es una herramienta estadística no lineal capaz de aprender a partir de ejemplos y generalizar su conocimiento a nuevos datos.

El componente básico de una red neuronal es la neurona artificial, también llamada unidad de procesamiento. Estas neuronas están organizadas en capas, y están conectadas entre sí por medio de enlaces que poseen un valor escalar llamado peso, el cual indica la importancia de dicha conexión. Adicionalmente, cada neurona puede tener un sesgo (bias), que permite ajustar la salida de la neurona incluso cuando todas sus entradas son cero.

El funcionamiento básico consiste en que cada neurona recibe un conjunto de entradas ($X = (x_1, x_2, \ldots, x_n)$), donde cada x_i representa una característica o valor de entrada, calcula una combinación ponderada de ellas mediante el producto de sus pesos ($W = (w_1, w_2, \ldots, w_n)$), le añade el sesgo ($b$), y aplica una función de activación (ϕ) que determina la salida a de la neurona. Estas salidas se propagan a las neuronas de la siguiente capa, y así sucesivamente, hasta llegar a la capa final.

$$z = X \cdot W^t + b = (x_1, x_2, \ldots, x_n) \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} + b = \sum_{i=1}^{n} x_i \cdot w_i + b$$

A continuación, se aplica una función de activación:

$$a = \phi(z)$$

El proceso de aprendizaje en una red neuronal consiste en ajustar los pesos y los sesgos mediante algoritmos de optimización, como el descenso del gradiente, en base a un conjunto de datos de entrenamiento. Esto permite que el modelo aprenda a realizar tareas como clasificación, regresión o reconocimiento de patrones.

Las redes neuronales suelen organizarse en tres tipos de capas:

- Capa de entrada: recibe los datos iniciales que se introducen al modelo.
- Capas ocultas: una o varias capas intermedias que transforman progresivamente los datos; son responsables de aprender las representaciones internas.
- Capa de salida: produce el resultado final, como una categoría o un valor numérico.

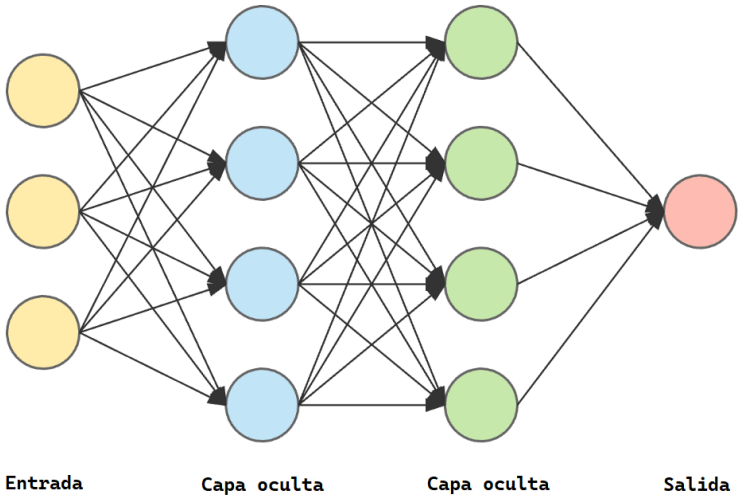


Figura 2.3: Ejemplo de red neuronal. Fuente: openwebinars.net

Las redes neuronales forman parte de un campo más amplio conocido como inteligencia artificial (IA), cuyo objetivo es desarrollar sistemas capaces de realizar tareas que normalmente requieren inteligencia humana.

Dentro de la IA, el aprendizaje automático (machine learning, ML) es un subcampo que se enfoca en que las máquinas aprendan patrones a partir de los datos sin ser explícitamente programadas. Las redes neuronales son uno de los algoritmos utilizados en ML.

Cuando se utilizan redes neuronales con múltiples capas ocultas, se habla de aprendizaje profundo (deep learning, DL). Este enfoque ha sido responsable de muchos avances recientes en tareas complejas como el reconocimiento de voz, la visión por computador, y la traducción automática.

2.2.2. Funciones de activación

Las funciones de activación [11] son componentes fundamentales en las redes neuronales artificiales, cuya principal función es introducir *no linealidad* en el modelo. Esta propiedad es esencial para que la red neuronal pueda aprender y representar relaciones complejas entre los datos de entrada y salida.

En términos generales, una función de activación transforma la señal que recibe una neurona z (conocida como *potencial de activación*) en una salida a que será propagada a las neuronas de la siguiente capa.

$$a = \phi(z)$$

Desde una perspectiva biológica, el término "potencial de activación"hace referencia al estímulo eléctrico que debe superar cierto umbral para activar una neurona en el cerebro. De forma análoga, en las redes neuronales artificiales la función de activación permite decidir si una neurona debe activarse (es decir, emitir una señal) o no, dependiendo del valor del potencial z.

Sin el uso de funciones de activación, la red neuronal se reduciría a una combinación puramente lineal de las entradas. Esto limitaría severamente su capacidad, haciendo que no pudiera resolver tareas complejas como clasificación no lineal, visión por computadora o procesamiento del lenguaje natural. Al introducir funciones no lineales, la red adquiere la capacidad de modelar funciones complejas, aprender representaciones abstractas y tomar decisiones basadas en relaciones sofisticadas entre los datos.

Existen diversas funciones de activación[11], cada una con características particulares según el tipo de tarea y arquitectura. Entre las más comunes se encuentran:

- Sigmoide: La función sigmoide, también conocida como función logística, transforma cualquier valor real de entrada en un valor en el rango (0, 1). Su expresión matemática es:

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

- Tangente hiperbólica (tanh): La función tangente hiperbólica es similar a la sigmoide, pero su rango de salida es (-1, 1), lo que la hace centrada en cero. Se define como:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

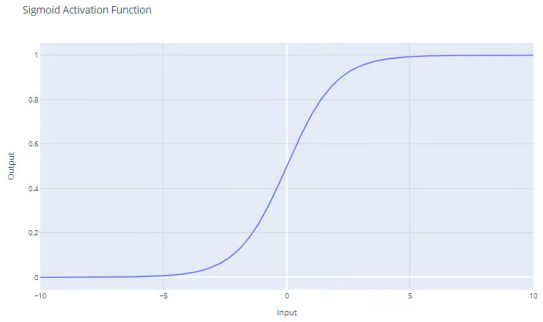
- ReLU (Rectified Linear Unit): La función ReLU es ampliamente utilizada en redes neuronales profundas debido a su simplicidad y eficiencia computacional. Se define como:

$$\text{ReLU}(z) = \text{máx}(0, z)$$

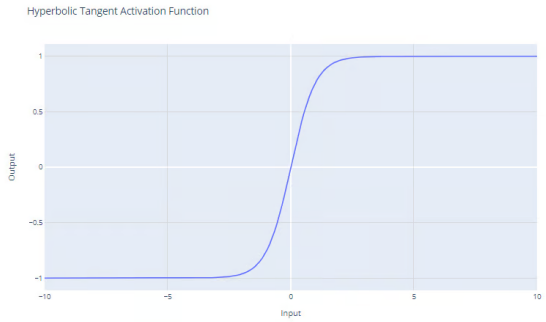
- Softmax: La función softmax se utiliza principalmente en la capa de salida de redes neuronales para problemas de clasificación multiclase. Convierte un vector de valores reales en un vector de probabilidades, donde cada valor está en el rango (0, 1) y la suma total es 1. Se define como:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

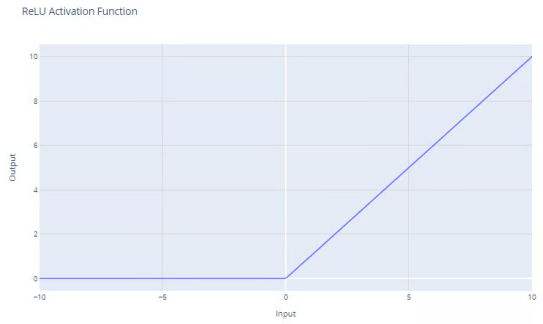
  donde $z = (z_1, z_2, \ldots, z_K)$ es el vector de entradas y K es el número de clases.



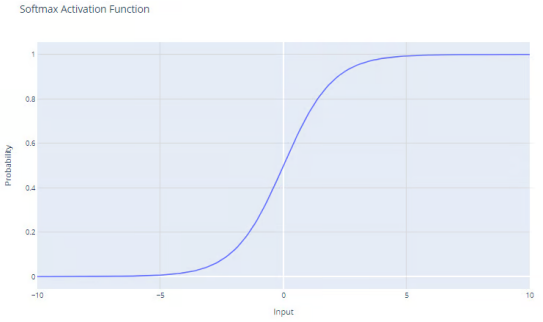
(a) Sigmoide



(b) Tangente hiperbólica



(c) Rectified Linear Unit



(d) Softmax

Figura 2.4: Funciones de activación. Fuente: www.datacamp.com

La elección adecuada de la función de activación, al igual que se menciona en [12], es un aspecto crítico en el diseño de redes neuronales, y puede influir significativamente en el rendimiento del modelo y su velocidad de entrenamiento.

2.2.3. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés *Convolutional Neural Networks*) representan uno de los pilares fundamentales del aprendizaje profundo. Se caracterizan por su capacidad para procesar datos estructurados en forma de mallas, como imágenes, vídeo, o incluso secuencias de audio.

Estructura general de una CNN

Una CNN[13] está compuesta por varias capas especializadas que actúan de manera secuencial:

- Capas convolucionales, responsables de la extracción de características.
- Capas de pooling o agrupación, encargadas de la reducción de dimensionalidad.
- Capas totalmente conectadas, que integran la información y permiten la clasificación final.

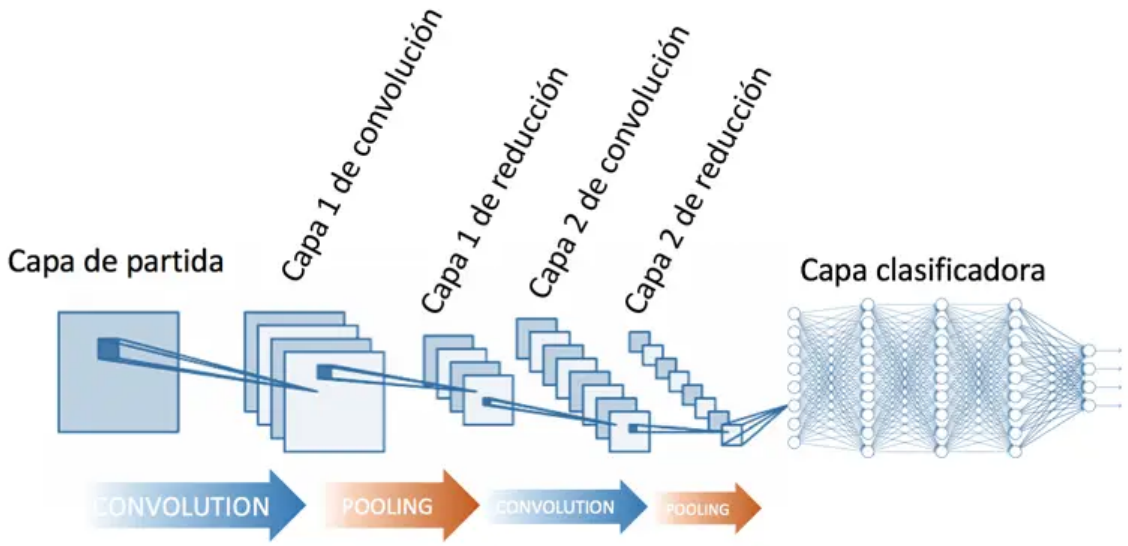


Figura 2.5: Estructura de una CNN. Fuente diegocalvo.es

Capa convolucional

La entrada a estas capas es un volumen tridimensional que representa la información espacial y de color de la imagen (altura, anchura y número de canales, como RGB). Sobre este volumen se aplican filtros o *kernels*, que son pequeñas matrices de pesos que se desplazan (operación conocida como convolución) [14] a lo largo de la imagen.

En cada posición, el filtro calcula un producto escalar entre sus valores y la región local de la entrada, generando un valor en la matriz de salida o mapa de características. De este modo, un mismo filtro detecta patrones específicos en toda la imagen, como bordes horizontales, verticales o esquinas. Al entrenar la red, los pesos de estos filtros se ajustan automáticamente para que las primeras capas respondan a patrones básicos, mientras que capas más profundas detectan combinaciones complejas como extremidades, siluetas humanas o incluso la figura completa de una persona.

Algunos hiperparámetros clave que controlan el funcionamiento de esta capa son:

- *Número de filtros:* define cuántos patrones distintos se aprenden en una capa. En aplicaciones de detección de personas, un mayor número de filtros suele ser beneficioso para capturar variaciones de postura, iluminación o perspectiva.
- *Stride:* determina el paso con el que el filtro se desplaza sobre la entrada. Un stride mayor reduce la resolución espacial de los mapas de activación, acelerando el procesamiento pero sacrificando detalle.
- *Padding:* consiste en añadir ceros en los bordes de la imagen para controlar el tamaño de la salida. Existen variantes como *valid* (sin padding), *same* (mantiene las dimensiones originales) o *full* (expande la salida).

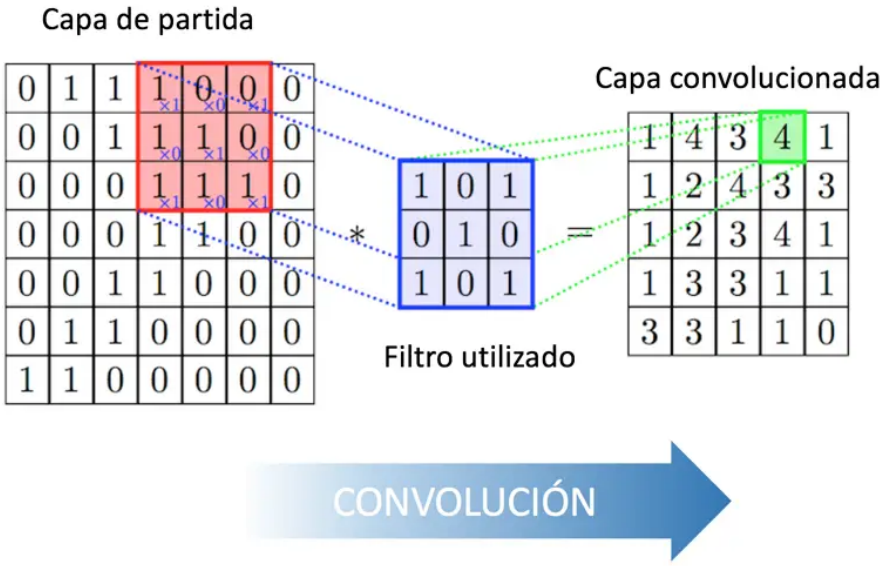


Figura 2.6: Convolución. Fuente diegocalvo.es

Tras la convolución, se aplica una función de activación no lineal, comúnmente ReLU (*Rectified Linear Unit*), que introduce no linealidad y evita que la red se limite a combinaciones lineales de las entradas.

Capas convolucionales jerárquicas

Mientras las primeras se centran en rasgos locales simples, las capas intermedias y profundas integran la información en estructuras más abstractas. Por poner un ejemplo, en el caso de la detección de personas, las primeras capas pueden detectar bordes de brazos o piernas. Las Capas intermedias reconocen formas de extremidades o torsos. Por último, las capas finales pueden identificar siluetas, incluso en condiciones difíciles como baja iluminación o variaciones en el ángulo de visión.

Capa de pooling

Después de cada bloque convolucional suele emplearse una capa de *pooling*, cuyo objetivo es reducir la dimensionalidad espacial del mapa de activación. Esto aporta tres beneficios principales:

1. Disminuye la carga computacional.

2. Aporta invariancia a pequeñas traslaciones.

3. Evita el sobreajuste, al forzar a la red a centrarse en las características más relevantes.

Los dos tipos más habituales son:

- Max pooling**:** selecciona el valor máximo de cada ventana, preservando las características más relevantes.

- Average pooling: calcula la media de los valores de cada región, suavizando la representación.

Capa clasificadora

En la parte final de la red, los mapas de características resultantes se aplanan y pasan a través de capas totalmente conectadas. Cada neurona de estas capas se conecta con todas las activaciones anteriores, permitiendo que la red integre toda la información en una representación compacta y tome decisiones de clasificación. En el contexto de este trabajo, estas capas son las responsables de decidir si la entrada corresponde a una persona o no.

2.3 La visión artificial

2.3.1. Definición y avances

La visión artificial, también denominada visión por computadora, visión de máquina o visión computacional, es una rama de la inteligencia artificial que busca dotar a las máquinas de la capacidad de percibir, interpretar y comprender el mundo visual a partir de imágenes y vídeos. Su objetivo fundamental es extraer información útil de datos visuales y transformarla en representaciones que puedan ser procesadas por un sistema informático, imitando en cierta medida el funcionamiento de la visión humana, pero con ventajas como la precisión en la medición de magnitudes físicas y la capacidad de operar de forma continua en tareas repetitivas.

El origen de esta disciplina[15] se remonta a finales de la década de 1950. En 1959, un grupo de neurofisiólogos experimentó mostrando diferentes imágenes a un gato para estudiar cómo su cerebro procesaba la información visual. Descubrieron que las primeras respuestas se producían ante bordes y líneas rectas, sentando las bases para entender que el procesamiento visual comienza con formas simples. Paralelamente, se desarrollaron las primeras técnicas de escaneo artificial de imágenes, que permitieron digitalizar y adquirir imágenes mediante ordenador. Un avance clave llegó en 1963, cuando las computadoras fueron capaces de transformar imágenes bidimensionales en representaciones tridimensionales.

En las décadas siguientes, la visión artificial experimentó avances notables. En 1974 apareció la tecnología de reconocimiento óptico de caracteres (OCR), capaz de leer texto impreso en diversas tipografías, y posteriormente el reconocimiento inteligente de caracteres (ICR) que, mediante redes neuronales, podía interpretar escritura a mano. Estas tecnologías se extendieron rápidamente a la gestión documental, la lectura automática de matrículas, y la verificación de pagos. Más tarde, en 1982, el seudocientífico David Marr propuso un modelo jerárquico del proceso visual, y desarrolló algoritmos para detectar bordes, esquinas y curvas.

Con el cambio de siglo, el interés se centró en el reconocimiento automático de objetos. Hacia 2001 surgieron las primeras aplicaciones de reconocimiento facial en tiempo real. La década de 2000 estuvo marcada por la estandarización de métodos para etiquetar y anotar grandes volúmenes de datos visuales, sentando las bases para el *Deep Learning*. En 2010, la aparición del conjunto de datos ImageNet, con millones de imágenes etiquetadas en mil categorías, impulsó el desarrollo de modelos de redes neuronales convolucionales a gran escala. En 2012, la arquitectura AlexNet demostró un salto cualitativo al reducir significativamente los errores en la clasificación de imágenes, marcando el inicio de la era moderna del aprendizaje profundo aplicado a la visión artificial.

En la actualidad, la visión artificial integra técnicas avanzadas de *Deep Learning* y arquitecturas como YOLO (You Only Look Once) o SSD (Single Shot MultiBox Detector), capaces de detectar y reconocer objetos en tiempo real con alta precisión, incluso en entornos dinámicos. El desarrollo de *hardware* especializado, como GPU y TPU, ha permitido entrenar modelos más complejos y rápidos, mientras que la optimización para dispositivos móviles y sistemas embebidos ha ampliado su alcance a campos como la robótica autónoma, la seguridad, la medicina, la realidad aumentada y la realidad virtual.

2.3.2. Etapas de un proceso de visión artificial

A continuación se describen los procedimientos comunes en el procesado de una imagen digital con visión artificial[16].

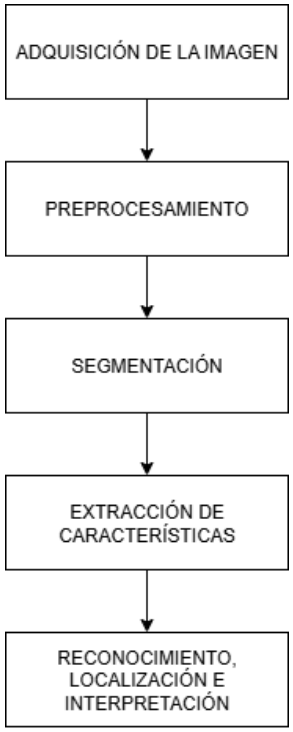


Figura 2.7: Etapas de un proceso de visión artificial. Fuente: ijcrt.org

1. *Adquisición de la imagen*. En esta etapa simplemente se toma o se prepara una imagen para su procesamiento.

2. *Preprocesamiento*. El preprocesamiento comprende un conjunto de operaciones destinadas a acondicionar la imagen antes de su análisis, empleando técnicas como la mejora, la restauración o la descompresión.

3. *Segmentación*. La segmentación consiste en identificar y delimitar las diferentes regiones que conforman una imagen, asignando a cada una de ellas una etiqueta específica. Cada región se compone de píxeles con propiedades homogéneas —por ejemplo, niveles de gris similares— y que se encuentran espacialmente conectados.

4. *Extracción de características*. Este procedimiento consiste en transformar la información de los píxeles en datos más compactos y significativos que describan aspectos como la forma, el tamaño, el color, la textura o la posición. Las características pueden provenir del contorno (por ejemplo, longitud, curvatura o perímetro) o del interior de la región (densidad, uniformidad, patrones internos, etc.).

5. *Reconocimiento, localización e interpretación*. El proceso de reconocimiento, localización e interpretación consiste en identificar qué objetos aparecen en una imagen, determinar su posición y extraer información relevante para la toma de decisiones.

   Actualmente las redes neuronales convolucionales (CNN) permiten realizar estas tareas de forma conjunta: arquitecturas como YOLO clasifican los objetos y generan coordenadas precisas de sus ubicaciones (cajas delimitadoras), mientras que modelos de nivel superior interpretan el contexto visual para evaluar estados, detectar interacciones o comprender la escena. De esta manera, los datos visuales se transforman en información procesable.

2.3.3. Técnicas comunes

A continuación se exponen algunas de las principales técnicas y métodos de preprocesamiento [17]. Estas forman parte de los primeros pasos en un proceso de visión por computador, descritos anteriormente.

En primer lugar, como técnicas de preprocesado encontramos la conversión a escala de grises, los filtros de suavizado (media y gaussiano), y las operaciones de erosión y dilatación.

Dado que una imagen a color cuenta con 3 canales de color R(Red), G(Green) y B(Blue), la conversión a escala de grises *combina* dichos canales para trabajar con una tercera parte de los datos totales de la imagen. Esta técnica es muy útil cuando el color no es relevante en la tarea de visión artificial. De esta manera se logra mejorar significativamente los tiempos de cómputo y la eficiencia de almacenamiento.

El suavizado más básico se logra mediante el filtro de media, que calcula el valor de un píxel como la media aritmética de los píxeles contenidos en una ventana definida alrededor de él. A mayor tamaño de la ventana, más pronunciado será el efecto de desenfoque. Aunque es sencillo de implementar, presenta como inconvenientes su marcada sensibilidad a variaciones locales, y la posible aparición de niveles de intensidad que no existían en la imagen original.

El filtro gaussiano funciona de forma similar al de promedio, pero utiliza una ventana con ponderaciones diferentes: los píxeles más próximos al píxel central tienen mayor influencia que los más lejanos. Dichos pesos se determinan siguiendo una distribución gaussiana, cuya varianza controla el grado de suavizado aplicado. La inclusión del parámetro de varianza ofrece un control más flexible sobre el suavizado, independientemente del tamaño de la ventana empleada.

En el procesamiento morfológico de imágenes binarias – aquellas en las que cada píxel toma el valor 1 para representar el objeto, y 0 para el fondo –, la erosión y la dilatación son operaciones complementarias que se suelen aplicar de forma combinada, principalmente para eliminar ruido y mejorar la calidad de la imagen antes de la extracción de

características. La erosión de una imagen A con un elemento estructural B (ambos imágenes binarias con fondo blanco) se define como:

$$A \ominus B = \{x \mid B_x \subseteq A\}$$

y consiste en poner a 0 todos los píxeles del objeto que estén en contacto con el fondo, reduciendo así el tamaño de las regiones y eliminando detalles pequeños. Por el contrario, la dilatación se define como:

$$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \emptyset\}$$

y realiza el proceso inverso: pone a 1 todos los píxeles del fondo que sean vecinos de píxeles del objeto, expandiendo las regiones y rellenando huecos. Aplicar erosión seguida de dilatación permite suprimir ruido manteniendo la forma global de los objetos, produciendo imágenes más adecuadas para el análisis posterior.

Figura 2.8: Comparación métodos de preprocesado.

CAPÍTULO 3
Análisis

3.1 Descripción del problema

Antes de entrar en detalle sobre los datos y el modelo usado en este proyecto, es importante conocer bien el problema al que nos enfrentamos. Al igual que se describe en el capítulo anterior, el objetivo principal es la detección de personas en diferentes lugares. El propósito del sistema a desarrollar es el de dotar a un dron de la capacidad de distinguir personas en ambientes de posible peligro, o que simplemente se hayan podido perder. Por eso, trabajaremos principalmente con datos en zonas rurales y, sobre todo, en parajes naturales como bosques o planicies.

Además de la geografía específica del terreno, debemos tener en cuenta la meteorología. El clima valenciano es de tipo mediterráneo con tránsito al clima desértico, seco y soleado, definido básicamente por precipitaciones escasas e irregulares, acentuada sequía estival, débil nubosidad, elevado número de días despejados, inviernos muy suaves, verano caluroso, fuerte insolación, intensa evaporación, y acusado déficit hídrico. Por tanto, el entrenamiento del modelo de red neuronal deberá estar adaptado a parajes ligeramente áridos.

Por otro lado, uno de los desafíos a la hora de entrenar una red neuronal es el *hardware* utilizado. El proceso de entrenamiento suele ser costoso tanto en términos de tiempo como de recursos computacionales, ya que requiere realizar millones de operaciones matemáticas sobre grandes volúmenes de datos. Este tipo de cargas de trabajo aprovecha de forma intensiva las unidades de procesamiento gráfico (GPU) o incluso clusters de cómputo distribuido, lo que implica la necesidad de disponer de equipos especializados que no siempre están al alcance de todos los usuarios. Además, a mayor complejidad del modelo y tamaño del conjunto de datos, más elevado será el consumo energético y económico asociado al entrenamiento.

3.2 Dataset

Los datos usados en este proyecto se han obtenido de Roboflow (https://roboflow.com/, una plataforma que facilita la creación, gestión y despliegue de modelos de visión artificial. El dataset esta formado por un conjunto de 1787 imágenes. La única clase presente es "human", y no todas las imágenes contienen personas (imágenes *background*).

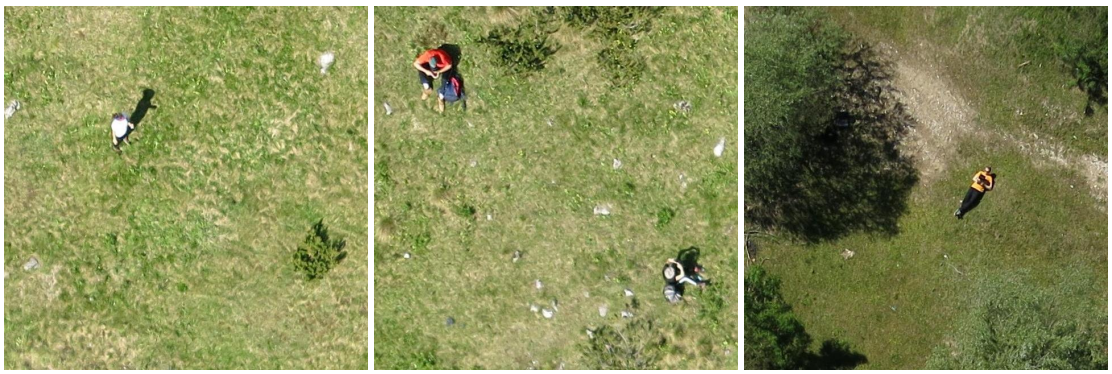
Todas las imágenes están redimensionadas a 640x640 para facilitar el entrenamiento del modelo y evitar hacer ajustes durante dicho proceso. Este tamaño representa un compromiso entre velocidad y precisión debido a que es lo suficientemente grande para captar gran cantidad de detalles, y permite realizar inferencias en un tiempo adecuado.

A continuación, veamos algunas de las imágenes que conforman el conjunto seleccionado.

Figura 3.1: Subconjunto del dataset.

Observamos que el conjunto de datos contiene bastante variabilidad de paisajes. Se mezclan gran cantidad de imágenes en climas tanto húmedos como secos, y cuenta también con ejemplos en zonas más urbanizadas, como pueden ser caminos y lugares apartados del núcleo urbano.

En cuanto a las personas, estas pueden estar presentes en las imágenes adoptando diferentes posturas. Observemos brevemente algunos ejemplos.



(a) De pie **(b)** Sentado **(c)** Tumbado

Figura 3.2: Ejemplos de posiciones de personas.

Aumentar el número de imágenes de entrenamiento puede ser bastante útil en tecnologías basadas en *Deep Learning*, véase [18]. *Data Augmentation* es una técnica sencilla que nos permite conseguir un *dataset* más variado a partir de transformaciones sencillas. En este caso, buscaremos modificar los valores de saturación, brillo y exposición para simular diferentes condiciones de luminosidad. También será interesante variar el tamaño de las imágenes para producir un efecto de vuelo a diferentes alturas.

Por último, estos datos estarán distribuidos en diferentes conjuntos, denominados *train* y *validation*, cada uno con un propósito específico dentro del proceso de entrena-

miento de un modelo. Apoyándonos en autores como [19] seguiremos la siguiente distribución.

El conjunto de *train* contendrá aproximadamente el 80 % de los datos, y se utiliza para que el modelo aprenda, ajustando sus parámetros internos a partir de los ejemplos proporcionados.

El conjunto de *validation*, que representará el 20 % de los datos, se emplea para evaluar de manera periódica el rendimiento del modelo durante el entrenamiento, ayudando a detectar problemas como el sobreajuste (*overfitting*) y a ajustar hiperparámetros.

3.3 Elección del modelo

Para realizar este proyecto, se requiere un modelo de red optimizado para la detección de objetos, capaz de identificar formas y patrones en las imágenes con el fin de localizar personas. Actualmente, existe una gran variedad de modelos de visión artificial desarrollados para esta tarea. Entre los más destacados se encuentran *Faster R-CNN*, *SSD*, *RetinaNet* y *YOLO*.

Es importante introducir los conceptos[20] de *backbone*, *neck* y *head*. Por lo general, los modelos de detección de objetos se dividen en tres componentes principales: el *backbone*, encargado de realizar las operaciones de convolución y extraer las características relevantes de la imagen; el *neck*, que toma los mapas de características generados por el *backbone* y los combina o refina con el objetivo de enriquecer la información espacial y contextual; y el *head*, que se encarga de realizar las predicciones finales, como las coordenadas de las cajas delimitadoras y las clases detectadas. Además, una red también puede clasificarse en etapas (una etapa o dos etapas), dependiendo de si emplea o no una red adicional de propuestas para la detección.

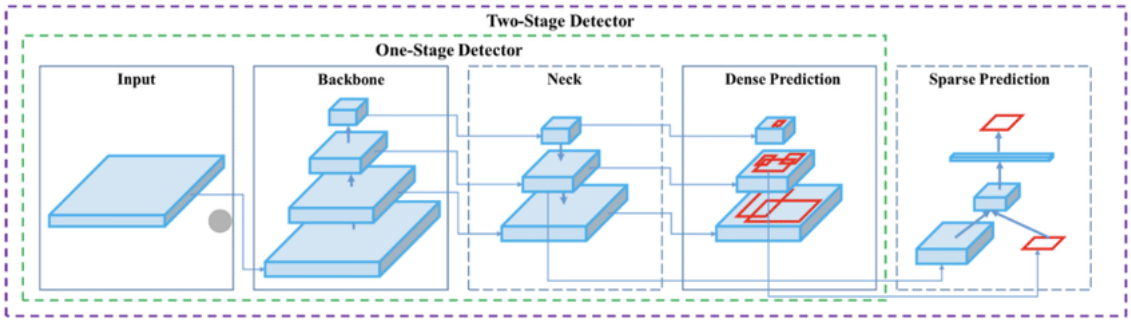


Figura 3.3: Esquema de distribución por etapas. Fuente:

En primer lugar, *Faster R-CNN* es un modelo pionero de dos etapas que alcanza gran precisión, aunque con mayor complejidad y coste computacional. *RetinaNet* introduce la *Focal Loss* para equilibrar precisión y velocidad frente a los negativos abundantes, con buenos resultados, pero con un ecosistema hoy menos dinámico. *SSD* ofrece detección en una sola etapa con alta velocidad, pero su rendimiento puede degradarse en objetos pequeños. Por su parte, *YOLO* (una etapa) en sus variantes recientes proporciona un compromiso muy competitivo entre precisión y rapidez, pipelines sencillos de entrenamiento e inferencia, amplia disponibilidad de pesos preentrenados, y herramientas de despliegue. Por estas razones – modernidad, comunidad activa y facilidad de uso – **se elige *YOLO* en su versión 11 como base de este proyecto**.

A continuación se exponen brevemente los modelos previamente mencionados. En el caso de *YOLO*, se profundizará más en su estructura.

3.3.1. Faster R-CNN

La mayoría de los modelos actuales de detección de objetos se apoyan en los fundamentos introducidos por *Faster R-CNN*[21]. Este modelo identifica objetos en una imagen, dibuja *bounding boxes* alrededor de ellos, y los clasifica en sus categorías correspondientes. Se trata de un detector en dos etapas.

La primera etapa corresponde a la *Region Proposal Network* (*RPN*), que utiliza como entrada un mapa de características generado por una red convolucional base, como *VGG16* o *ResNet*, empleada como *backbone*. A partir de este mapa se generan múltiples *anchor boxes* (regiones en la imagen) de distintos tamaños y proporciones, los cuales son clasificados como fondo o posibles objetos, y refinados mediante funciones de pérdida de clasificación y regresión.

La segunda etapa utiliza las regiones candidatas producidas por el *RPN* para realizar la clasificación final de los objetos, y un ajuste adicional de las cajas para normalizar el tamaño de las propuestas, optimizando de manera conjunta la clasificación y la regresión de las cajas.

Durante la inferencia el modelo filtra las propuestas de menor confianza, generando finalmente las predicciones más precisas. El entrenamiento de *Faster R-CNN* puede realizarse de forma secuencial, entrenando primero el *RPN* y después el clasificador, o bien de manera conjunta.

3.3.2. RetinaNet

RetinaNet[22] es un modelo de detección de objetos de una sola etapa, diseñado para equilibrar rapidez con precisión. Su principal innovación radica en la introducción de la *Focal Loss*, una función de pérdida que resuelve el problema del desequilibrio entre clases durante el entrenamiento. Un problema común a la hora de entrenar un modelo de detección de objetos es la posible presencia de gran cantidad de ejemplos que corresponden a *background* (regiones sin objeto), lo que puede dificultar el aprendizaje. La función *Focal Loss* atenúa la contribución de los ejemplos fáciles (fondos bien clasificados) y otorga mayor peso a los ejemplos difíciles, con objetos clasificables, permitiendo al modelo concentrarse en aquellos casos más relevantes.

La arquitectura de *RetinaNet* se organiza en tres componentes principales. En primer lugar, emplea un *backbone*, típicamente una red convolucional profunda como *ResNet*, encargada de extraer un mapa de características representativo de la imagen de entrada. A continuación, se utiliza un *Feature Pyramid Network* (*FPN*) como *neck*, el cual genera mapas de características en múltiples escalas, facilitando la detección de objetos de distintos tamaños. Finalmente, en el *head*, el modelo incorpora dos subredes paralelas: una dedicada a la clasificación de las regiones candidatas, y otra a la regresión de las cajas delimitadoras.

Durante la inferencia, *RetinaNet* produce un gran número de predicciones a partir de los *anchors* definidos sobre los mapas de características multiescala. Posteriormente, aplica un umbral de confianza para descartar predicciones de baja probabilidad y utiliza la técnica de *Non-Max Suppression* (*NMS*) para eliminar cajas solapadas, conservando las detecciones más relevantes.

3.3.3. Single Shot Multibox Detector

SSD[23] es un modelo de detección de objetos de una sola etapa que busca ofrecer un equilibrio óptimo entre rapidez y precisión. A diferencia de los detectores en dos etapas, este enfoque descarta por completo la fase de generación de propuestas (*RPN*), lo que permite encapsular todo el proceso de detección en una única red neuronal y agilizar significativamente el rendimiento.

En cuanto a su arquitectura, *SSD* emplea como *backbone* una red convolucional pre-entrenada (como *VGG-16*), para extraer un mapa de características representativo de la imagen. A este *backbone* le siguen capas adicionales de convolución que generan mapas de características a distintas resoluciones, construyendo una estructura similar a un *neck*, aunque más simple, orientada a detectar objetos de diferentes escalas.

Para cada uno de esos mapas de características, SSD define un conjunto fijo de cajas predefinidas —llamadas *default boxes* o *anchors*— con distintas proporciones y escalas, lo cual permite cubrir la variedad de formas que pueden presentar los objetos. El *head* del modelo está formado por pequeñas convoluciones 3×3 por cada posición del mapa, que simultáneamente predicen la clase del objeto y refinan las coordenadas de las cajas con respecto a las *default boxes*.

Durante la inferencia, todas estas predicciones se generan en un solo pase de la red. Posteriormente, se descartan aquellas con baja confianza, y se aplica *Non-Max Suppression* (NMS) para eliminar detecciones redundantes o solapadas, reteniendo únicamente las más precisas y representativas.

3.3.4. YOLOv11

YOLO (You Only Look Once)[20] es una familia de modelos de detección de objetos en tiempo real basada en redes neuronales convolucionales. Fue introducido en 2015 por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi. A diferencia de los métodos tradicionales de detección, que aplicaban la red neuronal sobre múltiples regiones de la imagen, *YOLO* realiza una única pasada (*single step*) por la red para predecir todas las detecciones, lo que le confiere gran velocidad y eficiencia.

El modelo divide la imagen en una cuadrícula y, para cada celda, predice un conjunto de *bounding boxes*, junto con las probabilidades de que cada caja contenga un objeto de determinada clase.

En cuanto a su estructura, *YOLOv11* utiliza un *backbone* personalizado basado en *CSP-Darknet*. A lo largo de la arquitectura se integran distintos bloques diseñados para mejorar la representación de la imagen y facilitar la detección de objetos en múltiples escalas. Los más relevantes son *C3K2*, *SPPF* y *C2PSA*.

El bloque *C3K2* está compuesto por varias capas convolucionales junto con conexiones residuales. Las convoluciones extraen patrones visuales para diferentes niveles de detalle (bordes, texturas, formas más complejas), mientras que las conexiones residuales permiten que parte de la información original fluya directamente, evitando la pérdida de características útiles y reduciendo problemas de degradación durante el entrenamiento.

El bloque *SPPF* (*Spatial Pyramid Pooling Fast*) tiene como objetivo aumentar el campo receptivo del modelo, es decir, la cantidad de contexto que cada neurona puede "ver" en la imagen. Para ello aplica operaciones de *pooling* a diferentes escalas en paralelo, lo que permite capturar información de objetos grandes y pequeños simultáneamente. A diferencia de versiones anteriores de *SPP*, la variante fast optimiza el cálculo para que la red mantenga una alta velocidad de inferencia sin sacrificar precisión.

Por último, el bloque *C2PSA* (*Cross Stage Partial Spatial Attention*) introduce un mecanismo de atención espacial. Este mecanismo asigna más peso a las regiones de la imagen que contienen información relevante para la detección, mientras que atenúa aquellas zonas que no aportan valor (fondo, ruido, etc.). De esta manera, la red concentra sus recursos en áreas clave, mejorando la precisión y reduciendo falsas detecciones.

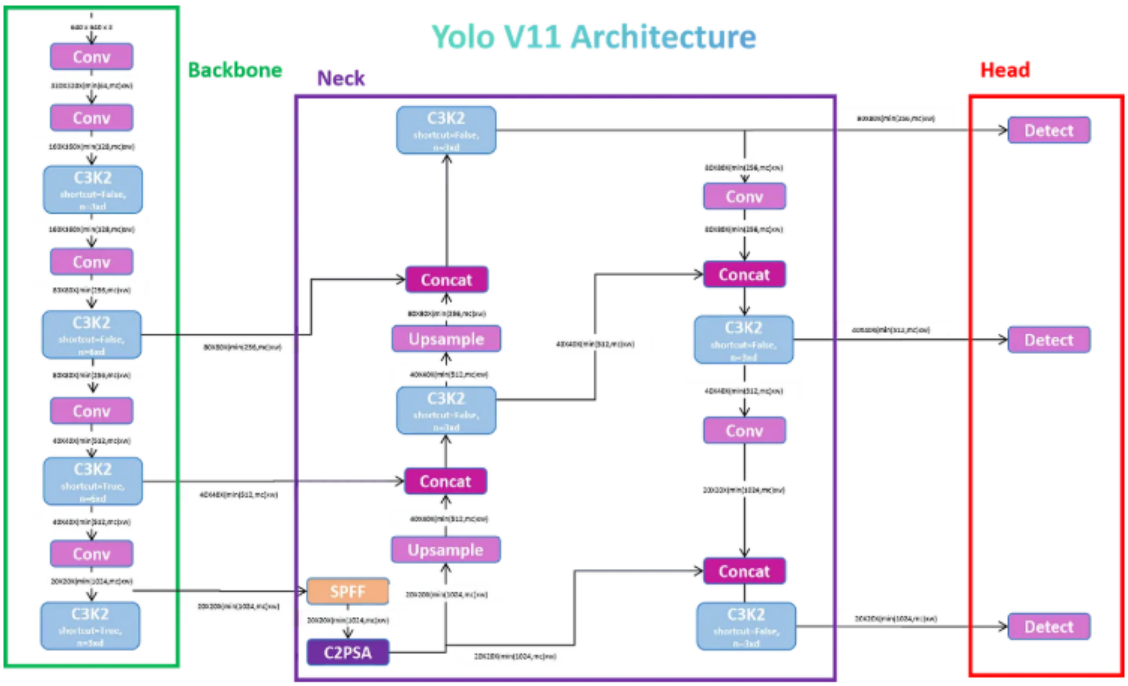


Figura 3.4: Arquitectura de YOLOv11. Fuente: medium.com

El proceso comienza con la entrada de imágenes de tamaño 640x640x3 (RGB) al *backbone*. En la primera capa convolucional, la imagen se transforma en un mapa de características de 320x320x64, reduciendo su resolución a la mitad, y aumentando la profundidad de los canales. Posteriormente, este mapa pasa por sucesivos bloques de convolución y módulos residuales tipo *C3k2*, que siguen extrayendo características cada vez más abstractas. En esta etapa la resolución espacial se reduce progresivamente: primero a 160x160x128, luego a 80x80x256, y más adelante a 40x40x512. Finalmente, el backbone genera un mapa de 20x20x1024, que concentra las características más ricas y semánticamente profundas de la imagen.

Como salida del *backbone* se producen tres *feature maps* de distintas escalas: 20x20x1024, 40x40x512 y 80x80x256. Estos mapas representan distintos niveles de detalle: los de mayor resolución capturan mejor objetos pequeños, mientras que los de menor resolución son más adecuados para detectar objetos grandes. Para enriquecer estas representaciones se incorpora un bloque *SPPF* y un módulo *C2PSA*.

En el neck, los mapas de características pasan por operaciones de *upsampling* (aumentar resolución) y concatenación, lo que permite fusionar la información de distintas escalas. Por ejemplo, el mapa de 20x20x1024 se combina con el de 40x40x512 tras un proceso de *upsampling* y, de forma análoga, este último se integra con el de 80x80x256. Esta estructura jerárquica facilita que el modelo preserve tanto información semántica profunda como detalles espaciales finos.

Finalmente, en el *head* del modelo se encuentran tres ramas de detección, que corresponden directamente a los tres niveles de escalas procesados. Cada rama toma un mapa

de características de diferente resolución (80x80, 40x40, 20x20) y genera predicciones finales en forma de *bounding boxes*, confianza de objeto y clasificación de clase.

3.4 Hardware para el entrenamiento

En este proyecto se hace uso del clúster Sirius de la UPV [24] , que es un clúster de cálculo científico que el ASIC (Área de Sistemas de Información y Comunicaciones) de la Universitat Politècnica de València pone a disposición de la comunidad investigadora, diseñado para abordar tanto trabajos de proceso distribuido como de memoria compartida. Este sistema se basa en un supercomputador exaescala BullSequana XH3000 desarrollado por Atos, y representa un salto cualitativo respecto a su predecesor (Rigel).

El clúster está compuesto por 33 nodos de cómputo equipados con procesadores Intel Xeon 8480 de 56 núcleos, y por 2 nodos acelerados con 4 GPUs NVIDIA HGX H100 de 80 GB cada una, alcanzando una capacidad total de 259 Tflops en propósito general, y más de 535 Tflops en supercomputación y aplicaciones de Inteligencia Artificial. Para ello, los nodos se interconectan mediante tecnología Infiniband NDR a 400 Gb, con baja latencia, y con un ancho de banda agregado de 1600 Gb, además de contar con almacenamiento de alto rendimiento gestionado por un sistema Lustre SFA200NVX que ofrece hasta 240 TB efectivos en discos NVMe de cuarta generación.

El entorno está gestionado mediante *Bull Smart Management Center* sobre Linux Red Hat 8 (edición HPC), lo que facilita el control centralizado de *hardware*, *firmware* y monitorización de recursos. Desde la perspectiva del usuario, Sirius integra el sistema de gestión de colas Slurm, así como librerías y herramientas optimizadas para computación científica y paralela, incluyendo OpenMPI, CUDA para GPUs NVIDIA, Singularity para contenedores, y el paquete Intel OneAPI con compiladores y librerías de alto rendimiento.

Finalmente, cabe destacar que Sirius se encuentra entre los sistemas de supercomputación más eficientes del país, con un PUE (*Power Usage Effectiveness*) de 1.12 gracias a técnicas de refrigeración directa al chip (DLC) y *free-cooling*, situándose entre los clústeres energéticamente más sostenibles a nivel internacional según el *ranking Green500*.

CAPÍTULO 4
Desarrollo de la solución

4.1 Preparación del dataset

4.1.1. El formato de datos YOLO

El *framework* YOLO (You Only Look Once) utiliza un formato de datos sencillo y altamente optimizado para el entrenamiento de modelos de detección de objetos.

Habitualmente un *dataset* en YOLO se organiza en carpetas que contienen las imágenes (/images) y sus anotaciones (/labels) correspondientes, separadas en subconjuntos: entrenamiento (*train*), validación (*val*) y, de forma opcional, prueba (*test*).

Cada imagen, normalmente en formato *.jpg* o *.png*, tiene asociado un archivo de texto con el mismo nombre en la carpeta */labels*. Estos archivos contienen las anotaciones de los objetos presentes en la imagen en forma de cajas delimitadoras (*bounding boxes*).

Cada línea de un archivo de etiquetas representa un objeto y está formada por cinco valores:

1. El identificador de la clase (un entero que indica la categoría del objeto).
2. Las coordenadas normalizadas del rectángulo que lo contiene: x_centro, y_centro, ancho y alto.

Todas estas coordenadas se expresan como valores relativos entre 0 y 1 respecto al tamaño de la imagen, lo cual garantiza que el formato sea independiente de la resolución original, favoreciendo así la portabilidad y la generalización del modelo.

Figura 4.1: Ejemplo visual de etiquetado en YOLO. Fuente: roboflow.com

Por último, la relación entre las carpetas de imágenes y etiquetas, junto con la definición de las clases, se especifica en un archivo de configuración llamado `data.yaml`, donde se indican las rutas de los subconjuntos y la lista de nombres de las categorías. A continuación se muestra el archivo de nuestro *dataset*:

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

nc: 1
names: ['human']

roboflow:
  workspace: main-1js2p
  project: drone-icerc
  version: 2
  license: CC BY 4.0
  url: https://universe.roboflow.com/main-1js2p/drone-icerc/dataset/2
```

Explicación de parámetros

- *train*, *valid*, *test*: rutas relativas a las carpetas que contienen las imágenes de entrenamiento, validación y prueba (no usaremos este último). Estas referencias permiten a YOLO cargar correctamente cada subconjunto del dataset.
- *nc*: número total de clases presentes en el dataset. En este caso, el valor es 1 porque solo se detecta una categoría de objeto.
- *names*: lista con los nombres de las clases que debe aprender el modelo. Aquí aparece únicamente un nombre, correspondiente a la clase única del dataset.
- *roboflow*: bloque adicional generado automáticamente al exportar el dataset desde la plataforma Roboflow. Incluye:
  - *workspace*: identificador del espacio de trabajo en Roboflow.
  - *project*: nombre del proyecto al que pertenece el dataset.
  - *version*: número de versión del dataset, útil para mantener control de cambios.
  - *license*: tipo de licencia bajo la que se distribuye el dataset.
  - *url*: enlace directo al dataset en Roboflow Universe.

Este archivo es esencial, ya que actúa como puente entre el modelo y los datos, asegurando que YOLO pueda localizar tanto las imágenes como las etiquetas, y reconocer las clases que debe detectar durante el entrenamiento.

4.1.2. Data Augmentation

En este apartado vamos a utilizar técnicas de *data augmentation* para conseguir mayor variabilidad y mejorar el entrenamiento [19]. Para ello usaremos la propia plataforma de Roboflow, que cuenta con la capacidad de crear nuevos sets de datos modificados a partir de uno inicial.

En la plataforma de Roboflow, dentro del apartado "Dataset", seleccionamos "New Dataset Versionz nos redirigirá a una nueva pestaña donde podremos ajustar la configuración del nuevo set. En primer lugar encontramos parámetros sencillos, como la proporción de imágenes de *train* y *testing*, que no vamos a modificar. También se pueden incluir

opciones de preprocesado, donde seleccionaremos la orientación automática y el reescalado a 640x640 píxeles. A continuación encontramos las opciones de "data augmentation", donde nos interesa, sobre todo, los efectos de *crop*, *saturation*, *brightness* y *exposure*.

- *Crop*: Agrega variabilidad al posicionamiento y al tamaño para ayudar al modelo a ser más resistente a las traslaciones del sujeto y a la posición de la cámara. Esta característica es especialmente útil en el contexto de este proyecto, puesto que el dron puede volar a diferentes alturas.

- *Saturation*: Ajusta aleatoriamente la saturación de los colores en las imágenes.

- *Brightness* y *exposure*: Agrega variabilidad al brillo de la imagen para ayudar al modelo a ser más resistente a los cambios de iluminación y configuración de la cámara.

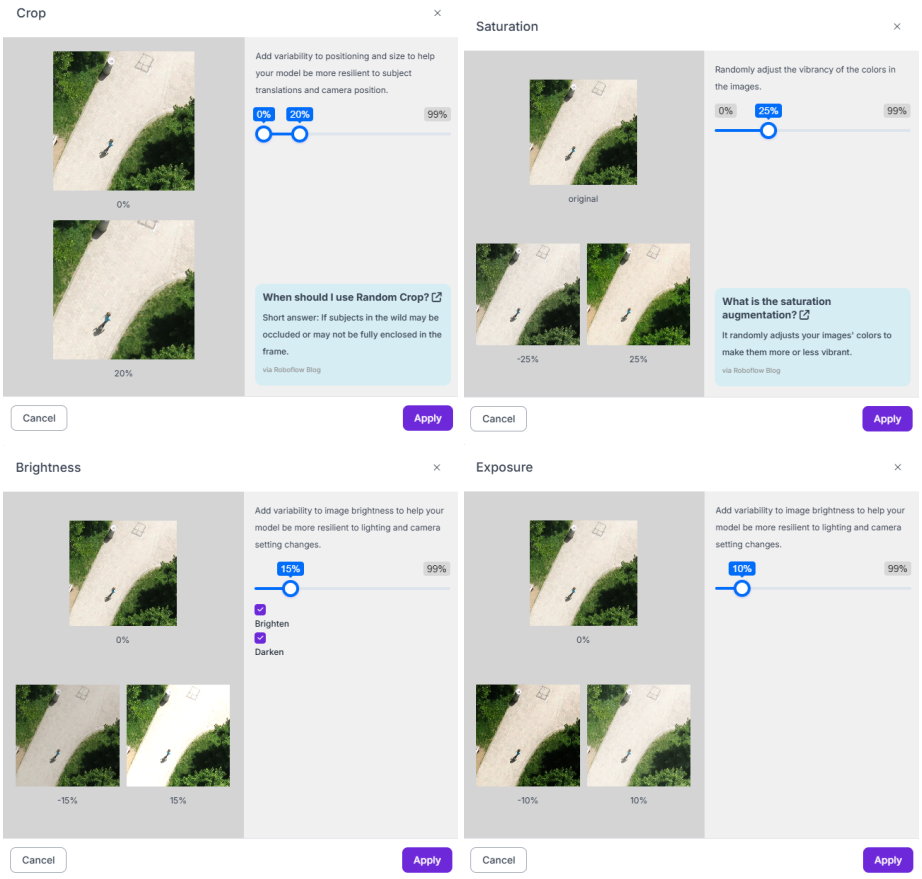


Figura 4.2: Configuración de los efectos de data augmentation.

Ahora seleccionamos la opción de multiplicar por 3 (realmente se generan menos) el número de imágenes del set de datos. De esta forma conseguimos una nueva versión con 4289 elementos. Sin embargo, nos damos cuenta que las proporciones de *train* y *validation* están desequilibradas.

Para resolver el problema, dado que Roboflow no ofrece una solución sencilla, usaremos un *script* de Python que redistribuya todas las imágenes en los sets de *train*, y *validation*, hasta conseguir la proporción planteada en el capítulo anterior de 80 % y 20 % respectivamente.

4.1.3. Configuración de Github

Uno de los problemas que surgieron en el desarrollo de este proyecto fue la subida del *dataset* al clúster remoto. Finalmente, se optó por crear un repositorio en Github para gestionar todas las versiones del *dataset*, y poder recoger fácilmente los resultados del entrenamiento.

El procedimiento para la creación del mismo es bastante simple. Después de registrarse y configurar el cliente de consola en el ordenador personal, se crea el repositorio, en este caso privado, y se clona en una carpeta local. Como el repositorio es privado (al final del proyecto se convertirá en público), se debe crear también un *token* personal para clonar el repositorio en la máquina remota. Después, basta con mover el set de datos a dicha carpeta y, mediante las instrucciones de *add*, *commit* y *push*, se pueden subir fácilmente los cambios a la rama principal.

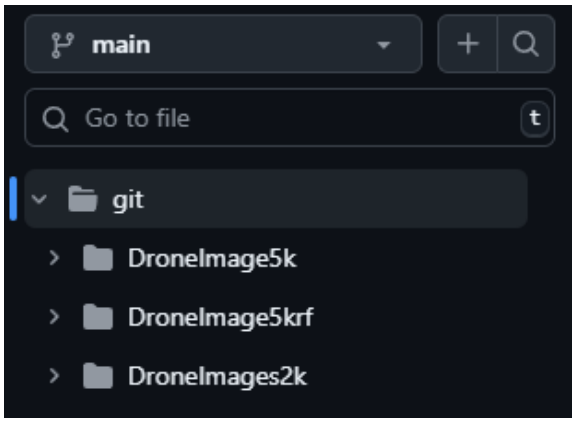


Figura 4.3: Captura de la estructura del repositorio en Github.

La figura anterior muestra la estructura del repositorio tras añadir 3 *datasets*. Cada versión está nombrada de acuerdo con el contenido aproximado de imágenes que contienen, y si han sufrido alguna modificación:

- DroneImages2k: El dataset original sin cambios, conformado por 1787 elementos.
- DroneImage5k: Dataset al que se le ha aplicado *Data Augmentation*, dando como resultado un set de 4289 imágenes.
- DroneImage5krf: Dataset final, ampliado con *Data Augmentation*, y redistribuido en los sets de train, validation y test, siguiendo las proporciones 80 % y 20 %, respectivamente.

En este proyecto sólo se ha hecho uso del último dataset, el resto han sido útiles para realizar pruebas.

4.2 Preparación del entrenamiento

4.2.1. Conexión al clúster

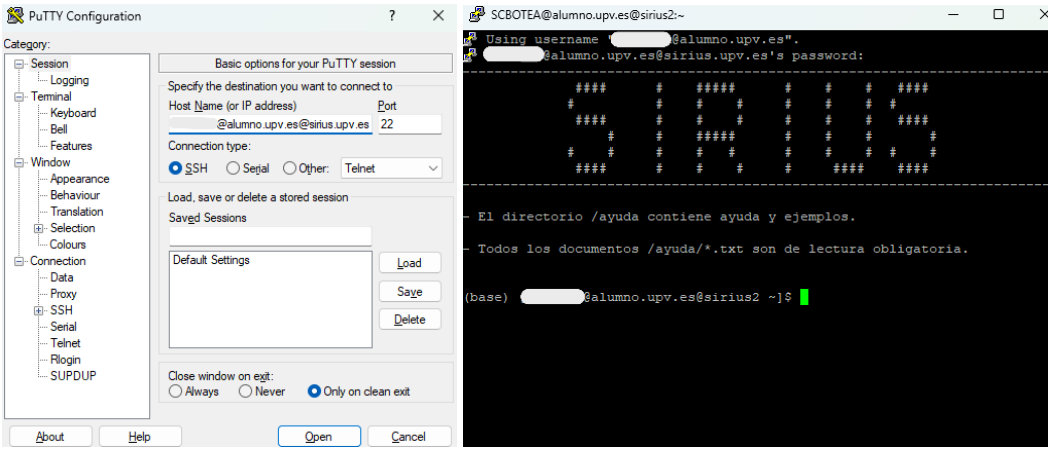
Al igual que se ha descrito anteriormente, el clúster Sirius cuenta con 35 nodos de cómputo, de los cuales dos, los que disponen de aceleración por GPU, son los que se

usarán en este proyecto. Antes de poder usarlos, debemos comprender la estructura de acceso al clúster.

Tal como explica la guía[24], los usuarios no pueden acceder de modo interactivo a los nodos de cálculo, sólo trabajan en los nodos de cabecera, a los que pueden acceder por *ssh*. Este frontend de usuario es un clúster en alta disponibilidad de dos sistemas, y su dirección es sirius.upv.es

Para la conexión por *ssh* usaremos el cliente de escritorio PuTTY (https://www.putty.org/). Se trata de un cliente fácil de usar que ofrece varias herramientas para conexiones *SSH*, *Telnet*, *rlogin*, y *TCP raw*, y que además cuenta con licencia libre. Destacar que estamos usando un equipo local con Windows 11.

Tras seleccionar el tipo de conexión que necesitamos (*ssh*), introducimos la dirección del *frontend* y las credenciales. Seguidamente el clúster nos da la bienvenida con un mensaje donde nos proporciona algunas instrucciones básicas.



4.2.2. Configuración del entorno

Una vez dentro del entorno, encontramos nuestro propio directorio */HOME* vacío. Además de este, tenemos asignado otro directorio de propósito general con gran cantidad de almacenamiento, y en el que podemos trabajar con cargas de datos pesadas, concretamente en la dirección */lustre/scratch/'usuario'/*. Destacar que todos los archivos de esta dirección se borran a los 30 días sin cambios.

Siguiendo las prácticas recomendadas, y para mantener todo ordenado, lo primero que vamos a hacer es instalar Conda (https://anaconda.org/anaconda/conda) en nuestro directorio *HOME*. Se trata de un sistema de gestión de paquetes y entornos de código abierto que permite instalar múltiples versiones de paquetes de software y sus dependencias, y alternar fácilmente entre ellas. Funciona en Linux, Mac y Windows, y fue creado para programas Python.

A continuación se muestra la sucesión de comandos necesarios para realizar la instalación:

Instalación del shell *script*

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Instalación de conda y adición al *PATH*

```
chmod +x Miniconda3-latest-Linux-x86_64.sh
```

```
./Miniconda3-latest-Linux-x86_64.sh
export PATH=~/miniconda3/bin:$PATH
source ~/.bashrc
```

Iniciación, actualización y creación del entorno

```
conda update -n base -c defaults conda
conda init
conda create --name yolotrain python=3.10
conda activate yolotrain
```

Como resultado, obtenemos una carpeta */miniconda* en nuestro */home* que contiene todas las utilidades y dependencias del programa.

Dentro del entorno *yolotrain* recién creado instalaremos también *ultralytics*, *pytorch* y *torchvision*, junto con los paquetes de *cuda* para poder usar las GPU's de NVIDIA. La siguiente instrucción realiza todo el proceso:

```
conda install -c pytorch -c nvidia -c conda-forge pytorch torchvision
pytorch-cuda=11.8 ultralytics
```

A continuación pasaremos a configurar el directorio de Github. El frontend del clúster ya dispone del cliente de consola de git, por tanto, no hace falta instalarlo. Debido a que en este repositorio albergará un espacio de almacenamiento considerable, lo vamos a mantener en */lustre/scratch/'usuario'/*.

Primero creamos una carpeta */github*, y dentro de ella clonamos el repositorio. Sin embargo, al ser un repositorio privado debemos modificar ligeramente la *url* con un *token* de acceso personal previamente generado. El resultado del comando es el siguiente:

```
git clone https://username:<token>
github.com/sabinzw/Datasets.git
```

Esto nos generará la estructura de directorios del repositorio y descargará los *datasets* que necesitamos para el entrenamiento. Además, usaremos esta dirección para subir los entrenamientos exitosos generados por Ultralytics. De esta forma se facilita bastante el flujo de información entre el equipo personal y el clúster remoto.

También necesitaremos un archivo con extensión *.sh* para mandar los trabajos a los nodos de cómputo desde el frontend. Estos nodos de Sirius operan con un gestor de colas denominado *SLURM*; en el siguiente enlace pueden encontrarse varios manuales y ejemplos de uso que vamos a necesitar (https://slurm.schedmd.com/documentation.html).

SLURM es un sistema de gestión de clústeres y programación de tareas de código abierto, tolerante a fallos, y altamente escalable para clústeres Linux grandes y pequeños. Como gestor de cargas de trabajo de clúster, *SLURM* tiene tres funciones clave. En primer lugar, asigna a los usuarios acceso exclusivo o no exclusivo a los recursos (nodos de cómputo) durante un tiempo determinado para que puedan realizar su trabajo. En segundo lugar, proporciona un marco para iniciar, ejecutar y supervisar el trabajo (normalmente un trabajo paralelo) en el conjunto de nodos asignados. Por último, arbitra la contención de recursos mediante la gestión de una cola de trabajo pendiente.

Nombraremos al script de lanzamiento como *run.sh*, y lo alojaremos en una nueva carpeta */scripts* con dirección: */lustre/scratch/usuario'/scripts*. El contenido del archivo que usaremos inicialmente para lanzar el entrenamiento de los modelos *YOLOv11* se muestra a continuación.

```
#!/bin/bash
#SBATCH --job-name=yolotrain
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=160G
#SBATCH --gres=gpu:2
#SBATCH --time=72:00:00
#SBATCH --output=yolo11.out
#SBATCH --error=yolo11.err
#SBATCH --partition=gpu

eval "$(conda shell.bash hook)"
conda activate yolotrain

yolo detect train data=/lustre/scratch/'usuario'/github/Datasets/git/
    DroneImages5krf/data.yaml model=yolov11n.pt epochs=100 imgsz=640 device
    =[0,1] pretrained=True
```

Los parámetros de SLURM son:

- `#SBATCH --job-name=yolotrain`
  Nombre del trabajo en la cola.
- `#SBATCH --nodes=1`
  Número de nodos solicitados. En este caso usaremos uno de los dos disponibles para no saturar el sistema.
- `#SBATCH --ntasks-per-node=1`
  Número de tareas por nodo. Usaremos una sola tarea principal.
- `#SBATCH --cpus-per-task=48`
  Número de núcleos de CPU asignados. Seguiremos la recomendación del equipo de Sirius de 48 hilos.
- `#SBATCH --mem=160G`
  Memoria RAM disponible para el trabajo. Cada GPU NVIDIA HGX H100 cuenta con 80G, por tanto, usaremos 160 GB.
- `#SBATCH --gres=gpu:2`
  Número de GPUs reservadas. Cada nodo cuenta con dos GPU's NVIDIA HGX H100, usaremos ambas para mayor velocidad de entrenamiento.
- `#SBATCH --time=72:00:00`
  Tiempo máximo de ejecución. Usaremos el tiempo límite de 72 horas o 3 días.
- `#SBATCH --output=yolo11.out`
  Archivo de salida estándar, donde se mostrará la ejecución normal de las órdenes (*stdout*).
- `#SBATCH --error=yolo11.err`
  Archivo de salida de errores (*stderr*).
- `#SBATCH --partition=gpu`
  Cola o partición del *clúster* utilizada. Estaremos usando aquella que contiene GPU's.

Luego, se activa el entorno de trabajo con conda:

```
eval "$(conda shell.bash hook)"
conda activate yolotrain
```

Finalmente, se ejecuta el entrenamiento con YOLOv11:

```
yolo detect train data=/lustre/scratch/github/Datasets/git/DroneImages2k/data.yaml \
    model=yolov11n.pt epochs=100 imgsz=640 device=[0,1] pretrained=True
```

- `data`: ruta al archivo `data.yaml` con la configuración del dataset. Usaremos datos del repositorio de github que hemos clonado anteriormente.
- `model`: modelo base utilizado (`yolov11n.pt`).
- `epochs`: número de épocas de entrenamiento. Como vamos a usar un modelo pre-entrenado, no necesitamos muchas.
- `imgsz`: tamaño de las imágenes de entrada (640x640 píxeles).
- `device`: GPUs empleadas durante el entrenamiento (en este caso, la 0 y la 1).
- `pretrained`: opción para cargar los pesos de un modelo ya entrenado en el *dataset* COCO (*transfer learning*).

Por último, crearemos una carpeta */results* que usaremos para recoger los resultados de entrenamiento. La ruta a este directorio es */lustre/scratch/'usuario'/results*.

4.3 Entrenamiento de modelos

4.3.1. Elección de modelos YOLOv11

En Ultralytics (https://docs.ultralytics.com/es/models/yolo11/) se encuentran disponibles cinco modelos YOLOv11 de diferentes tamaños. La siguiente tabla presenta una comparativa de los modelos entrenados sobre la base de datos COCO (*Common Objects in Context*), uno de los *datasets* más utilizados en visión por computador. COCO contiene más de 200.000 imágenes y 80 clases diferentes de objetos anotados con *bounding boxes*, segmentaciones y *keypoints*, lo que lo convierte en un estándar de referencia para evaluar arquitecturas de detección de objetos.

Tabla 4.1: Comparativa de los modelos YOLOv11 entrenados en la base de datos COCO.

Modelo	Tamaño (px)	mAP_{val} 50-95	Vel. CPU (ms)	Vel. T4 (ms)	Parám. (M)	FLOPs (B)
YOLOv11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLOv11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLOv11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLOv11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLOv11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

A continuación se explica cada una de las columnas presentadas en la tabla 4.1:

- Modelo: Identificador del modelo YOLOv11, donde la letra final indica el tamaño de la arquitectura (*n* = nano, *s* = small, *m* = medium, *l* = large, *x* = extra-large).

- Tamaño (px): Resolución de entrada de las imágenes utilizada durante el entrenamiento y la inferencia. En este caso, todos los modelos han sido evaluados con imágenes de 640×640 píxeles.
- mAP_{val} 50-95: Precisión media promedio (*mean Average Precision*) en el conjunto de validación de COCO, calculada en un rango de umbrales de IoU (*Intersection over Union*) entre 0.5 y 0.95. Esta métrica mide la precisión al momento de predecir las *bounding boxes* y tiene en cuenta el área de intersección entre las cajas correctas y las de la predicción del modelo. Es un indicador del rendimiento general.
- Vel. CPU (ms): Tiempo promedio de inferencia medido en milisegundos utilizando un procesador. Refleja la velocidad en entornos sin GPU.
- Vel. T4 (ms): Tiempo promedio de inferencia en milisegundos utilizando una GPU NVIDIA T4 con TensorRT. Permite medir el rendimiento en aceleradores de *hardware* optimizados.
- Parám. (M): Número de parámetros entrenables de la red neuronal, expresados en millones. Modelos más grandes suelen tener más capacidad, pero requieren más memoria y potencia de cómputo.
- FLOPs (B): Número estimado de operaciones de coma flotante necesarias para procesar una imagen, expresado en miles de millones (*billions*). Este valor indica la complejidad computacional del modelo.

A partir de la misma tabla se puede observar cómo los distintos modelos de la familia YOLOv11 tienen requisitos distintos de memoria y coste computacional. Pasemos ahora a estimar los requerimientos de *hardware* de cada uno.

- YOLOv11n (nano): Es el modelo más ligero, con únicamente 2.6 millones de parámetros y un coste de 6.5 GFLOPs. Está diseñado para dispositivos con recursos muy limitados, sacrificando precisión en favor de velocidad y eficiencia.
- YOLOv11s (small): Incrementa el número de parámetros hasta 9.4 millones, y eleva el coste computacional a 21.5 GFLOPs. Supone una mejora significativa en precisión respecto al modelo nano, manteniendo aún un tamaño reducido.
- YOLOv11m (medium): Con 20.1 millones de parámetros y 68 GFLOPs, se ubica como una opción intermedia, ofreciendo un equilibrio entre precisión y coste de cómputo. Requiere ya de *hardware* más potente para un entrenamiento y despliegue eficientes.
- YOLOv11l (large): Aumenta a 25.3 millones de parámetros y 86.9 GFLOPs. Está orientado a escenarios en los que la precisión es más prioritaria que la velocidad de inferencia o el consumo de recursos.
- YOLOv11x (extra large): Es el modelo más grande de la familia, con 56.9 millones de parámetros y un coste de 194.9 GFLOPs. Requiere de *hardware* de alto rendimiento (como GPUs dedicadas) y no resulta viable para dispositivos embebidos de baja potencia.

En conclusión, los modelos más pequeños (*nano* y *small*) están orientados a sistemas embebidos y aplicaciones en tiempo real, mientras que los modelos más grandes (*large* y *extra large*) priorizan la precisión, pero con un coste computacional que los hace inadecuados para *hardware* limitado.

Como nuestro objetivo es preparar un modelo que se ajuste a las especificaciones de un sistema limitado, entrenaremos un modelos de tamaño *nano* y *small*.

4.3.2. Entrenamiento

Comenzamos el entrenamiento de los dos modelos. Lo primero es entrar al clúster Sirius y desplazarnos a la carpeta */results*, creada previamente. Desde aquí lanzaremos los entrenamientos. Empezaremos entrenando un modelo YOLOv11n. En este caso, el script ya está preparado y no necesitamos modificarlo. Con la siguiente orden lanzamos el entrenamiento:

```
sbatch ../scripts/run.sh
```

Esto nos genera tres archivos: *yolo11.out*, *yolo11.err* y *yolo11n.pt*. Además, se crea una carpeta *runs* donde se van guardando los resultados de cada entrenamiento de forma numerada, es decir, los distintos entrenamientos sucesivos que lanzamos se guardan en una nueva carpeta de este directorio siguiendo la nomenclatura *train + número*.

Lanzaremos el comando anterior dos veces, modificando el *script* de lanzamiento, concretamente el parámetro *model* de la función de entrenamiento de Ultralytics.

A continuación, accederemos a la carpeta de train y, mediante las siguientes instrucciones, cambiaremos el nombre del directorio generado tras el entrenamiento:

```
mv train1 train_n
mv train_n /lustre/scratch/'usuario'/github/Datasets/
```

Repetiremos este proceso para cada resultado de entrenamiento. Finalmente, actualizamos el repositorio de Github mediante:

```
cd /lustre/scratch/'usuario'/github/Datasets/
git add .
git commit -m "resultados de entrenamiento"
git push
```

Una vez obtenidos los resultados de este entrenamiento, nos dimos cuenta de que los resultados no eran del todo satisfactorios. Cabe aclarar que este entrenamiento consta de sólo 100 épocas, lo cual puede no ser suficiente. Por eso, siguiendo el método de early-stoping inspirado en [25], volvimos a lanzar los entrenamientos, pero esta vez aumentaremos el número de épocas de entrenamiento, e introduciremos el parámetro *patience* que nos permitirá acabar prematuramente el entrenamiento después de varias épocas sin hay mejora en la función de pérdida.

Resumidamente, lanzaremos para cada modelo:

- Un entrenamiento de 100 épocas fijas
- Un entrenamiento de 800 épocas con patience = 20
- Un entrenamiento de 800 épocas con patience = 50

4.3.3. Resultados de entrenamiento

Durante el entrenamiento del modelo YOLO se registran diversas métricas que permiten evaluar tanto el proceso de aprendizaje como el rendimiento en la tarea de detección de objetos.

Las pérdidas (*losses*) indican el error cometido por el modelo, y se minimizan a lo largo del entrenamiento:

- train/box_loss y val/box_loss: error en la regresión de las *bounding boxes*, es decir, en el ajuste entre las cajas predichas y las cajas reales.
- train/cls_loss y val/cls_loss: error de clasificación de los objetos dentro de cada caja, penalizando cuando se confunden clases.
- train/dfl_loss y val/dfl_loss: pérdida de *Distribution Focal Loss (DFL)*, utilizada para refinar la predicción de los bordes de las cajas y mejorar la localización.

Las pérdidas train/ corresponden al conjunto de entrenamiento, y las val/ al conjunto de validación, indicando la capacidad de generalización del modelo.

Las figuras 4.5 y 4.6 muestran los resultados del entrenamiento con 100 épocas.

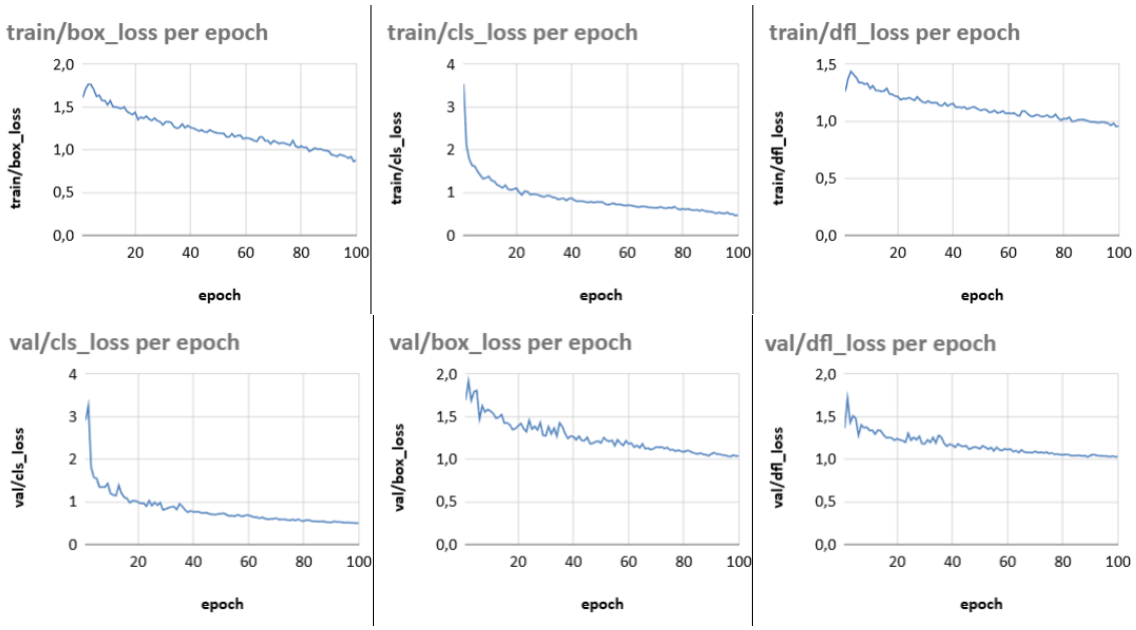


Figura 4.5: Gráficos de entrenamiento y validación de YOLOv11n en 100 épocas.

Observamos que el entrenamiento ha avanzado de forma progresiva y estable. Podemos destacar también que *cls_loss* es la función que más rápido se reduce debido a que en el *dataset* solo existe una clase.

Los resultados sobre el conjunto de validación progresan adecuadamente en ambos modelos. Observamos cierta similitud con los resultados de entrenamiento, aunque estos presentan algo más de inestabilidad.

En general, se sospecha *overfitting* cuando el rendimiento del modelo es excelente en el conjunto de entrenamiento, pero bajo en el conjunto de validación. Los gráficos muestran que la pérdida de entrenamiento y validación sigue disminuyendo. Por tanto, el modelo no parece estar sobreajustado. No obstante, los valores de las funciones de pérdida siguen siendo bastante altos.

Ahora, pasamos a analizar los resultados de entrenamiento en 800 épocas con *patience* = 20 (ver figuras 4.7 y 4.8).

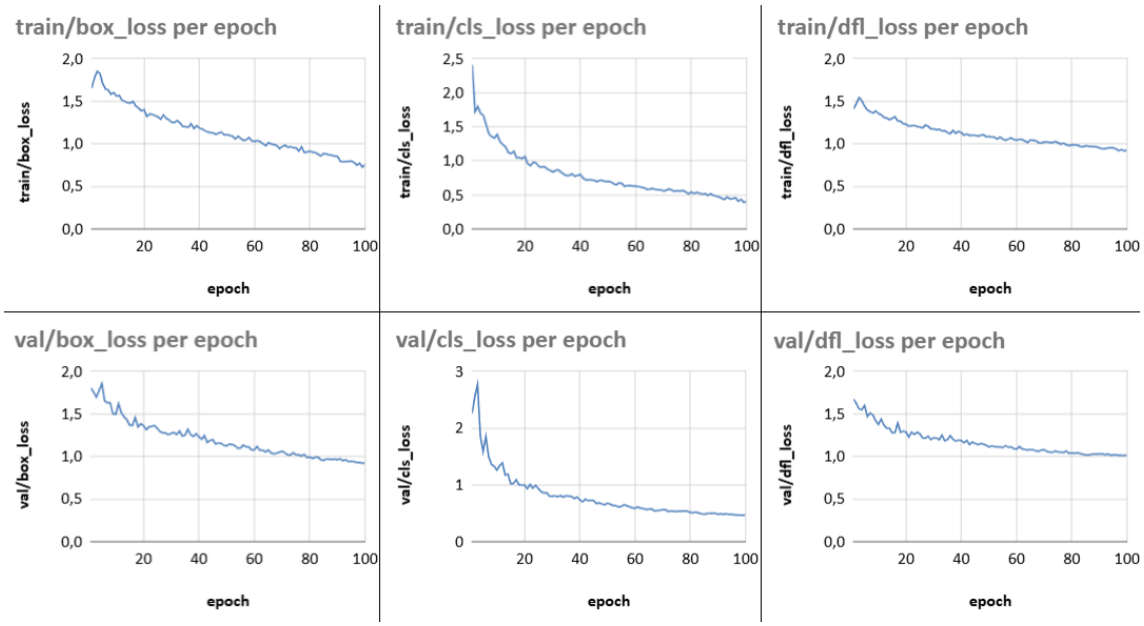


Figura 4.6: Gráficos de entrenamiento y validación de YOLOv11s en 100 épocas.

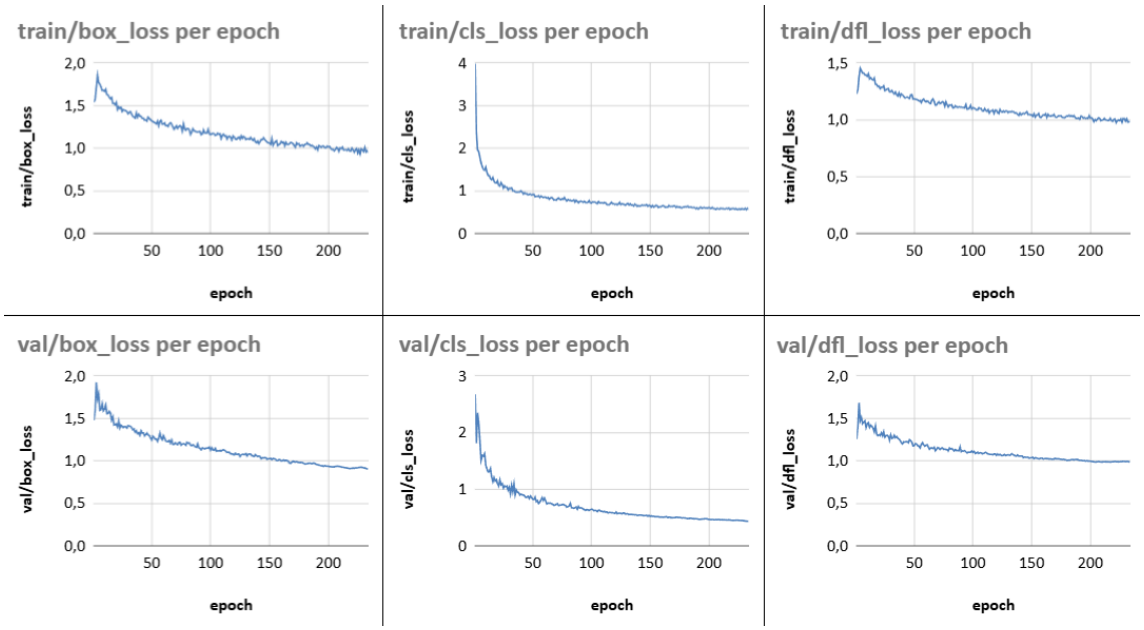


Figura 4.7: Gráficos de entrenamiento y validación de YOLOv11n en 800 épocas con *patience* = 20.

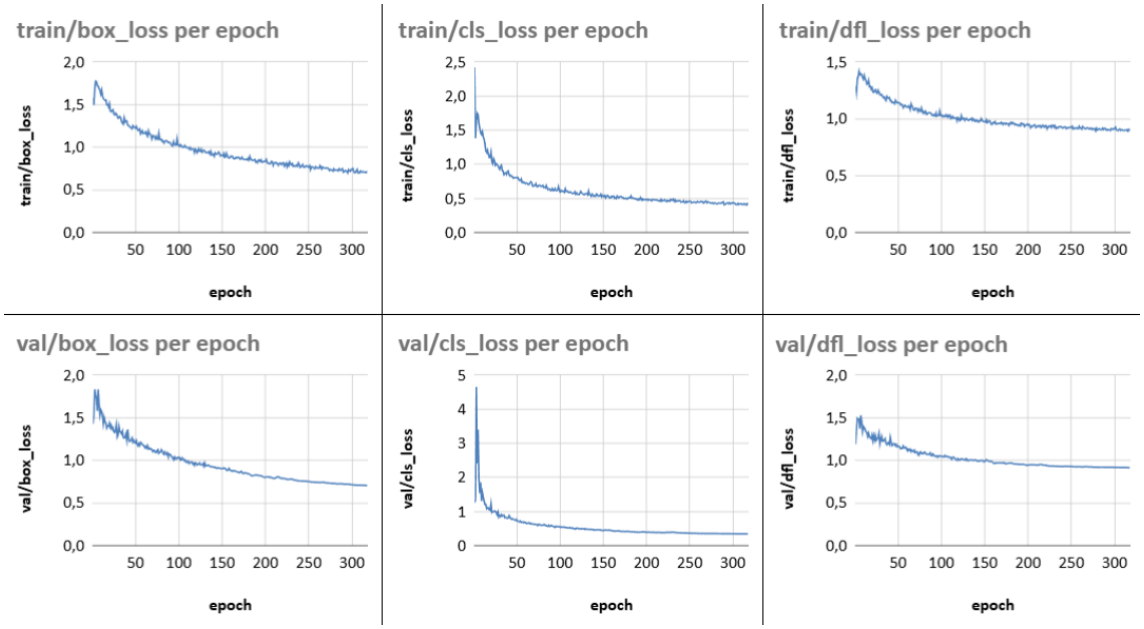


Figura 4.8: Gráficos de entrenamiento y validación de YOLOv11s en 800 épocas con *patience* = 20.

Cabe destacar que, para este par de entrenamientos, el modelo *nano* ha acabado prematuramente en la época 234, mientras que el *small* lo hace en la 318. Es decir, alcanzan los mejores resultados en las épocas 214 y 298, respectivamente.

En este caso notamos que las pérdidas de validación, sobretodo en el modelo *small*, comienzan a estancarse, lo que podría indicar *overfitting*. No obstante, se alcanzan valores de pérdida menores que en el entrenamiento anterior.

Por último, revisaremos el entrenamiento de 800 épocas con patience = 50 (ver figuras 4.9 y 4.10).

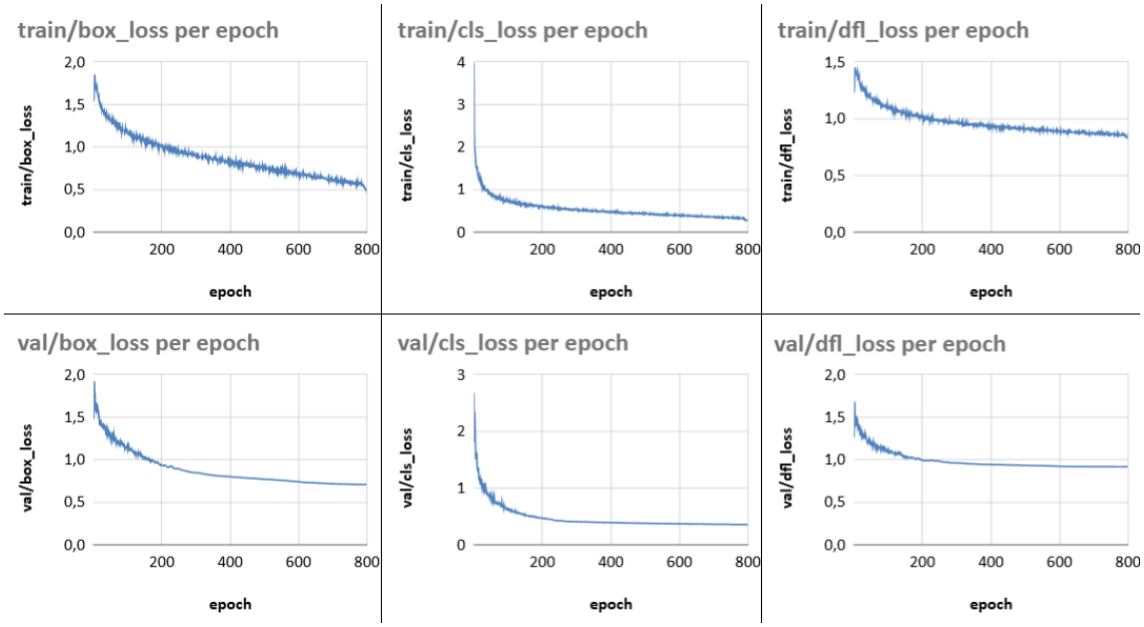


Figura 4.9: Gráficos de entrenamiento y validación de YOLOv11n en 800 épocas con *patience* = 50

Observamos que, con patience = 50, en 800 épocas, los modelos continúan mejorando las funciones de pérdida de entrenamiento y, por tanto, no se activa el *early stopping* en ninguno de los casos. Sin embargo, las funciones de pérdida de validación presentan,

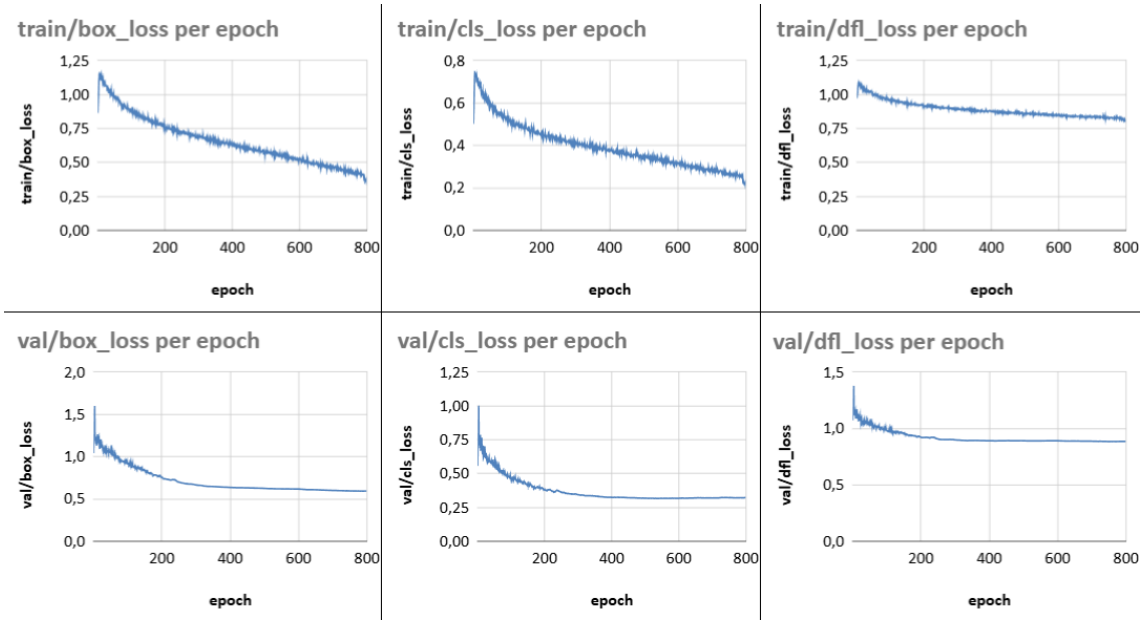


Figura 4.10: Gráficos de entrenamiento y validación de YOLOv11s en 800 épocas con *patience* = 50

especialmente en el modelo *small*, una pendiente cercana a 0. Este comportamiento puede interpretarse como un indicio de entrenamiento excesivo, ya que el modelo no logra seguir reduciendo de manera significativa el error en los datos de validación.

Si se continuara con el proceso de entrenamiento, es probable que las pendientes de las funciones de validación adquirieran valores positivos, lo que señalaría de forma clara la presencia de *overfitting* respecto a los datos de entrenamiento.

Para finalizar, y a modo de resumen, se muestra una tabla con los resultados de todos los entrenamientos para las configuraciones que minimizan las funciones de pérdida.

Tabla 4.2: Comparación de métricas de mejores resultados (YOLOn y YOLOs)

	100 épocas		800 épocas con patience = 20		800 épocas con patience = 50	
Métrica	YOLOn	YOLOs	YOLOn	YOLOs	YOLOn	YOLOs
train/box_loss	0.86312	0.72628	0,97943	0,71778	0.48933	0,35799
train/cls_loss	0.46130	0.38862	0,57961	0,42809	0.27184	0,21441
train/dfl_loss	0.95277	0.91707	0,99657	0,90592	0.82767	0,80534
val/box_loss	1.03673	0.92031	0,92022	0,71641	0.70953	0,59342
val/cls_loss	0.50275	0.46411	0,45794	0,34805	0.35761	0,32114
val/dfl_loss	1.02952	1.01183	0,98782	0,91763	0.91674	0,88552

En base a los resultados, **elegiremos los modelos del tercer entrenamiento** para realizar las pruebas.

CAPÍTULO 5
Pruebas

Para medir el rendimiento de los modelos entrenados anteriormente vamos a realizar inferencias sobre un video capturado por un dron en un terreno de la Comunidad Valenciana.

Las siguientes métricas miden la calidad de las predicciones :

- Precisión (*metrics/precision*): mide la proporción de predicciones correctas sobre todas las predicciones realizadas. Una alta precisión indica baja tasa de falsos positivos.
$$\text{Precisión} = \frac{TP}{TP + FP}$$
- *Recall* (*metrics/recall*): mide la proporción de objetos detectados correctamente sobre todos los que realmente existen. Un alto *recall* indica baja tasa de falsos negativos.
$$\text{Recall} = \frac{TP}{TP + FN}$$
- mAP@50 (*metrics*/mAP50): *Mean Average Precision* con umbral de IoU $\geq$ 0,5. Evalúa el rendimiento en detecciones consideradas correctas si la intersección entre la caja predicha y la real supera el 50 %.
- mAP@50–95 (*metrics*/mAP50-95): métrica más estricta, calculada promediando el mAP en varios umbrales de IoU desde 0.5 hasta 0.95 con incrementos de 0.05. Es la métrica de referencia en el conjunto COCO, ya que combina tanto precisión como localización precisa.

5.1 Procesamiento de vídeos

En total, se han grabado dos videos en el aeródromo de La Llosa, Castellón, usando un *Dron DJI Air 3S*. Estas son algunas de las especificaciones técnicas más relevantes de dicho dron obtenidas desde la página oficial de DJI:

- Peso al despegue: 724 g
- Dimensiones (sin hélices):
  - Plegado: 214,19 × 100,63 × 89,17 mm
  - Desplegado: 266,11 × 325,47 × 106,00 mm

- Velocidades máximas:
  - Ascenso: 10 m/s
  - Descenso: 10 m/s
  - Horizontal 21 m/s
- Alcance máximo: 32 km
- Sistemas de posicionamiento: GPS + Galileo + BeiDou
- Memoria interna: 42 GB
- Cámara principal (gran angular):
  - Sensor: CMOS de 1", 50 MP
  - Campo de visión: 84°
  - Equivalente focal: 24 mm
  - Apertura: f/1.8
- Vídeo:
  - Hasta 4K/60 fps con HDR y hasta 4K/120 fps (cámara lenta)
  - Registro en 10 bits (D-Log M y HLG), 14 pasos de rango dinámico
  - Formatos: H.264 / H.265 (MP4)

El primer video (ver figura 5.1) se ha tomado en en un paisaje más húmedo, con presencia de hierbas altas y bajas, cerca de la pista del aeródromo. en cambio, el segundo vídeo (ver figura 5.2) se ha tomado en una zona más seca.

Figura 5.1: Fotogramas del Vídeo 1.

Realizaremos pruebas de los dos modelos sobre ambos vídeos, obteniendo como resultado un conjunto de métricas que nos permitirán medir el rendimiento de dichos modelos sobre un entorno que no ha sido usado anteriormente para su entrenamiento. De

Figura 5.2: Fotogramas del Vídeo 2.

esta forma, podremos valorar el desempeño general de la herramienta, y su aplicación en las zonas rurales de la Comunidad Valenciana.

En primer lugar, es necesario dividir el vídeo en fotogramas. Debido a ciertos problemas de compatibilidad con el códec *HEVC*, se opta por utilizar un *script* en Python para generar los fotogramas manualmente. Para ello se hace uso de las bibliotecas MoviePy y Pillow. La primera permite realizar tareas básicas de edición de vídeo, mientras que la segunda está especializada en el procesamiento de imágenes.

Con MoviePy se carga el archivo de vídeo, y mediante la función `subclip()`, se recortan las secciones correspondientes al *despegue* y *aterrizaje* del dron. A partir de estos segmentos, se extraen los fotogramas, separados por intervalos de un segundo. Posteriormente, se procesan utilizando Pillow para recortarlos. Se prioriza recortar el centro de cada imagen, ajustándola al tamaño de entrada del modelo, es decir, 640x640 píxeles, lo que permite evitar pasos adicionales de preprocesado. Se seleccionan un total de 600 fotogramas, distribuidos equitativamente entre el vídeo 1 y el vídeo 2.

A continuación, se utiliza nuevamente la plataforma *Roboflow* para etiquetar los fotogramas extraídos. Primero, se crea un nuevo proyecto donde se suben todas las imágenes y se define la clase `human`. Una vez cargadas, se procede a su etiquetado manual mediante la herramienta de edición integrada (ver figura 5.3), que permite dibujar los cuadros delimitadores (*bounding boxes*) alrededor de las personas detectadas. Destacar que etiquetaremos un objeto en la imagen sólo si está presente al menos la mitad del mismo.

Después de un tiempo de trabajo, se logran etiquetar correctamente todas las imágenes que contienen al menos una persona. Aquellas que no presentan el objeto de la clase se marcan como *null*. Finalmente, se confirma el etiquetado y se trasladan todas las imágenes al conjunto de test, quedando así listas para su uso en la evaluación del modelo.

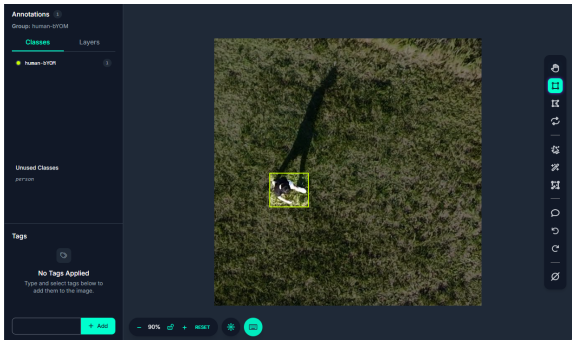


Figura 5.3: Proceso manual de etiquetado.

5.2 Ejecución de pruebas y análisis de resultados

En primer lugar, descargamos el conjunto de imágenes, lo almacenamos de forma local, y creamos un nuevo *script* de Python que se encargará de cargar el modelo y localizar dichas imágenes. Dicho script puede encontrarse en el repositorio público de Github. Las carpetas se nombran como *valid_v1, para el* Video 1, *y valid_v2* para el Video 2. Se configura un *threshold* o umbral de confianza de 0.5.

Recordar que estamos usando los modelos del tercer entrenamiento, es decir, aquellos que han sido entrenados durante 800 épocas.

A continuación, la tabla 5.1 muestra los resultados de la prueba de ambos modelos en relación a los videos 1 y 2.

Tabla 5.1: Tabla comparativa de desempeño bruto de los modelos

	Video 1		Video 2	
Métrica	YOLOn	YOLOs	YOLOn	YOLOs
Precision	0.736	0.806	0.713	0.752
Recall	0.732	0.760	0.724	0.711
mAP50	0.763	0.784	0.731	0.739
mAP50-95	0.292	0.307	0.290	0.298

En base a los resultados anteriores, observamos dos cosas: en primer lugar, el modelo *small* funciona, en términos generales, mejor que el modelo *nano*, lo cual es un resultado esperable. En segundo lugar, los resultados son mejores sobre el Video 1.

Debe tenerse en cuenta que el Video 2 está compuesto, en su mayoría, por imágenes de *background*, es decir, imágenes que no presentan ningún objeto. Además, en varias de las imágenes, se encuentran obstáculos como árboles altos y objetos que pueden confundir al modelo.

Otro detalle importante es la baja capacidad del modelo para predecir exactamente los *bounding boxes*, probablemente debido al hecho de que el conjunto de entrenamiento (y la base de datos COCO) contienen multitud de imágenes etiquetadas por diferentes personas. Sin embargo, el índice mAP50 es relativamente alto.

A continuación revisaremos algunos *batchs* de validación generados por Ultralytics después de las pruebas. Estos *batchs* son bastante grandes, por eso, se han elegido aquellos ejemplos que muestran errores en las predicciones.

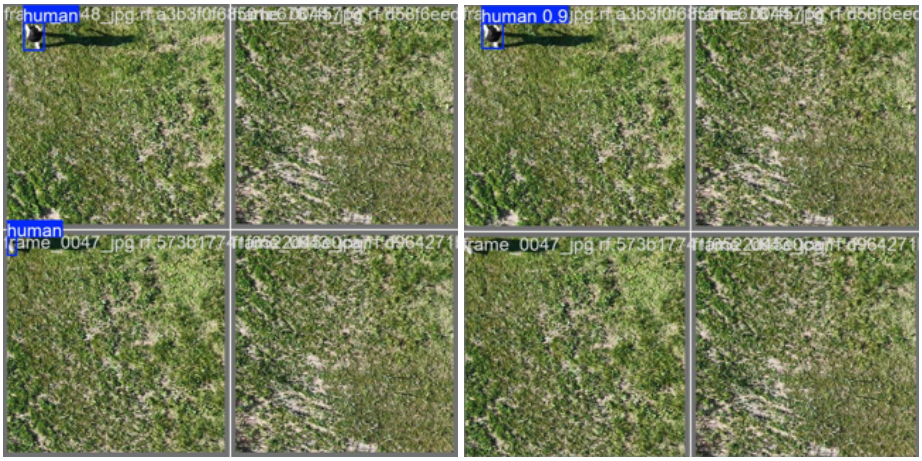


Figura 5.4: Predicciones de YOLOn en Video 1. A la izquierda, los objetos localizados correctamente y, a la derecha, las predicciones del modelo.

En la figura 5.4 observamos que el modelo *nano* acierta en tres de los cuatro casos expuestos. Durante la revisión, se ha observado que el modelo falla principalmente en detectar objetos en los bordes de la imagen.

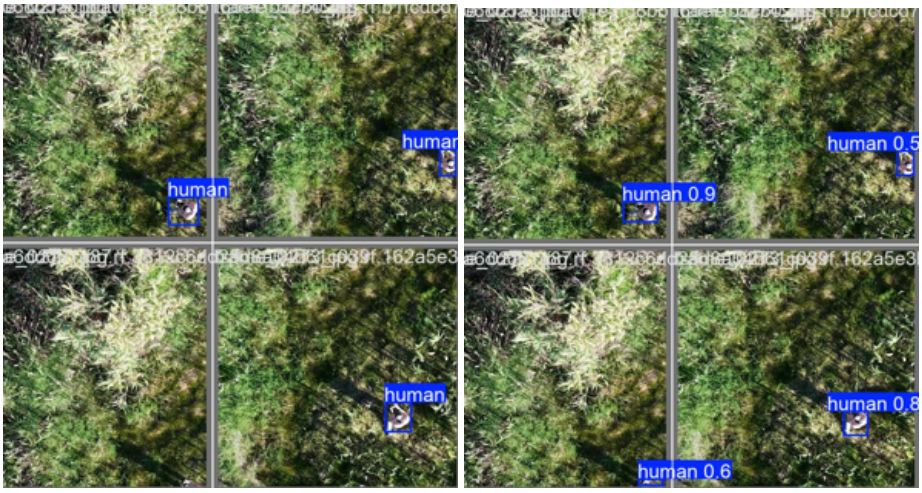


Figura 5.5: Predicciones de YOLOs en Video 1. A la izquierda, los objetos localizados correctamente y, a la derecha, las predicciones del modelo.

En el caso del modelo small (ver figura 5.5), observamos también el mismo problema. Hay veces que el modelo sí detecta los objetos al borde de la imagen, sobre todo si reconoce un brazo.

En la figura 5.6 podemos observar que todas las imágenes de la izquierda de la figura son de background (sin objetos) . El modelo *nano*, en ocasiones, confunde pequeños objetos, en este caso una especie de pequeña piedra azulada. En general, ambos modelos sufren este problema en el segundo vídeo, lo que perjudica sus métricas de precisión y recall.

No obstante, el modelo *small* tiene mejores puntuaciones y detecta mejor a las personas, como podemos ver en la figura 5.7. En este ejemplo, el modelo small consigue detectar a la persona con una confianza superior al 90 %.

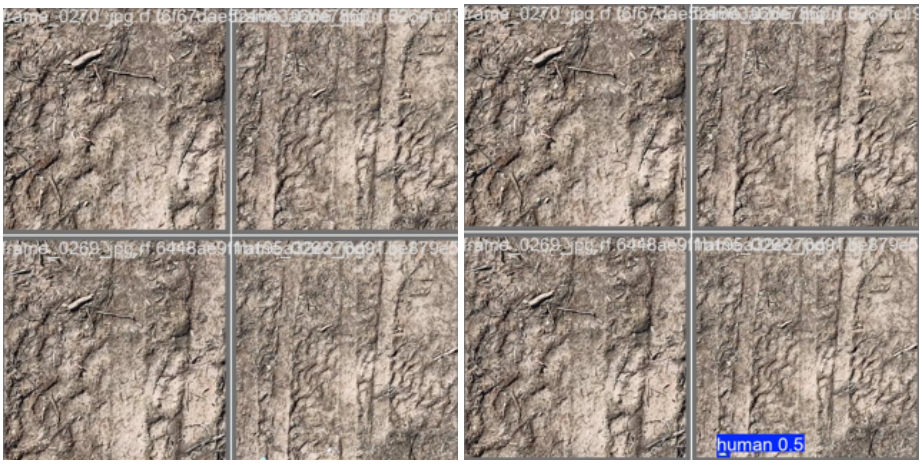


Figura 5.6: Predicciones de YOLOn en Video 2. A la izquierda, los objetos localizados correctamente y, a la derecha, las predicciones del modelo.

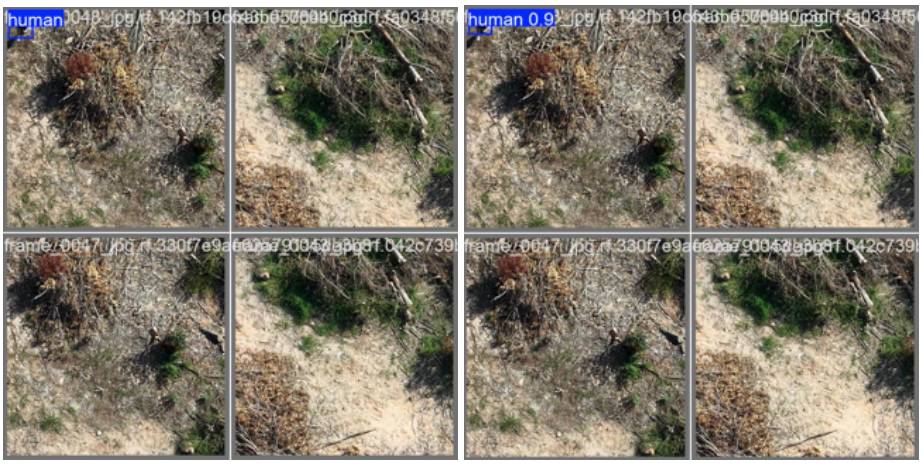


Figura 5.7: Predicciones de YOLOs en Video 2. A la izquierda, los objetos localizados correctamente y, a la derecha, las predicciones del modelo

5.3 Opciones de despliegue

Como último apartado de este capítulo se ofrecen algunas opciones para el despliegue de los modelos probados. Toda esta información puede consultarse en https://docs.ultralytics.com/es/modes/export/.

Ultralytics YOLOv11 ofrece múltiples opciones de exportación, lo que permite adaptar un mismo modelo a diferentes entornos. Entre estas opciones se encuentran exportaciones hacia formatos como TorchScript, que facilita el despliegue en entornos basados en PyTorch; ONNX, que asegura compatibilidad con diversos motores de inferencia y frameworks de producción; TensorRT, diseñado para optimizar la velocidad en tarjetas gráficas NVIDIA; y TFLite o Edge TPU, que habilitan su uso en dispositivos móviles o microcontroladores de bajo consumo. Asimismo, se contemplan exportaciones a CoreML para el ecosistema Apple, TF.js para aplicaciones en navegadores, OpenVINO para *hardware* Intel, y otros formatos especializados como NCNN, RKNN o IMX500, enfocados en sistemas embebidos e industriales.

Una vez definido el formato de exportación, se dispone de argumentos específicos que permiten ajustar el modelo según las necesidades del entorno. Por ejemplo, *imgsz* controla el tamaño de las imágenes de entrada, lo que influye directamente en la veloci-

dad y precisión de las predicciones. El argumento *half* habilita el uso de precisión reducida (FP16), lo que disminuye el tamaño del modelo y acelera la inferencia en *hardware* compatible, mientras que *int8* aplica una cuantización más agresiva que favorece el despliegue en dispositivos de borde con recursos limitados. Además, el parámetro *dynamic* permite trabajar con imágenes de distintos tamaños, lo cual otorga mayor flexibilidad en aplicaciones reales.

Finalmente, otros ajustes relevantes incluyen *simplify*, que elimina operaciones redundantes para optimizar el modelo, y *nms*, que incorpora un paso automático de supresión de detecciones repetidas dentro del propio modelo exportado. El argumento *batch* define cuántas imágenes pueden procesarse simultáneamente, lo que resulta útil en escenarios donde se prioriza la velocidad de procesamiento en lotes. Por último, parámetros como *device* (que selecciona CPU, GPU u otro *hardware* disponible) y *workspace* (que ajusta la memoria reservada en optimizaciones con TensorRT) garantizan que el proceso de exportación pueda adaptarse tanto a dispositivos de alta capacidad como a sistemas con limitaciones de recursos.

CAPÍTULO 6
Conclusiones

En esta sección final discutiremos la consecución de los objetivos planteados en el primer capítulo, la relación con las materias cursadas, así como posibles líneas de trabajo que se pueden plantear a futuro.

6.1 Objetivos cumplidos

En primer lugar, respecto al objetivo de diseñar y entrenar un modelo de detección de personas basado en redes neuronales convolucionales, podemos decir que el objetivo ha sido superado con éxito. Como resultado del trabajo se han obtenido obtenido dos modelos capaces de funcionar correctamente en diferentes condiciones de clima, altitud de vuelo y luminosidad.

En segundo lugar, dichos modelos han sido probados exitosamente sobre conjuntos de datos no incluidos en el *set* de entrenamiento, capturados manualmente desde un dron.

En tercer lugar, se ha creado una base de datos con una gran cantidad de imágenes que pueden servir de base para futuros proyectos. Todas las imágenes aumentadas, y todas aquellas nuevas, capturadas y etiquetadas manualmente, se encuentran disponibles en el repositorio público de *Github* creado para este proyecto, a la disposición de cualquier persona.

Por último, en relación al último objetivo, en el capítulo anterior se han ofrecido instrucciones para poder desplegar dichos modelos en dispositivos con recursos limitados.

6.2 Relación con las asignaturas cursadas

El trabajo realizado se relaciona en buena medida con las asignaturas cursadas en la carrera, especialmente aquellas de la rama de ingeniería de computadores.

La inteligencia artificial y las redes neuronales son temas estudiados en profundidad en la asignatura de *Deep Learning para la Industria*. Además, en esta asignatura se enseña a entrenar modelos y cuenta con prácticas en entornos reales. Por tanto, considero que ha sido la materia que más me ha ayudado en el desarrollo del proyecto.

Por otro lado, todas las asignaturas de los primeros dos cursos académicos me han dado una buena base para desenvolverme en entornos de consola. Otras asignaturas específicas de la rama, como la asignatura de *Tecnologías avanzadas para redes corporativas*

y Datacenters, han sido indispensables para comprender, acceder y hacer uso del clúster Sirius.

Finalmente, aunque en menor medida, asignaturas generales de tercer año orientadas al software, como *Ingeniería de Software*, me han ayudado a desarrollar todo el código *Python* necesario.

6.3 Trabajo futuro

Como trabajo futuro, aprovechando los resultados de este proyecto, se podrían desplegar los modelos obtenidos sobre un dron, y hacer pruebas de su rendimiento en *hardware* real. Sería interesante también usar técnicas de *cuantización* y *pruning* para mejorar dicho rendimiento.

Por otro lado, sería interesante aumentar el set de datos, obteniendo más imágenes en situaciones diversas, y mejorando así la robustez de los modelos.

Bibliografía

[1] Christian Wankmüller, Maximilian Kunovjanek, and Sebastian Mayrgündter. Drones in emergency response–evidence from cross-border, multi-disciplinary usability tests. *International Journal of Disaster Risk Reduction*, 65:102567, 2021.

[2] Piotr Kardasz, Jacek Doskocz, Mateusz Hejduk, Paweł Wiejkut, and Hubert Zarzycki. Drones and possibilities of their using. *J. Civ. Environ. Eng*, 6(3):1–7, 2016.

[3] Márton Bálint. History, types, application and control of drones. 2022.

[4] Ley 18/2014, de 15 de octubre, de aprobación de medidas urgentes para el crecimiento, la competitividad y la eficiencia. «BOE» núm. 252, de 17/10/2014., 2014. Revisado el 5 de septiembre de 2025. Disponible en: https://www.boe.es/eli/es/l/2014/10/15/18/con.

[5] Correos exhibe los drones desarrollados en el marco del proyecto delorean, 2022. Revisado el 5 de septiembre de 2025 https://www.correos.com/sala-prensa/correos-exhibe-los-drones-desarrollados-en-el-marco-del-proyecto-delorean/.

[6] Pablo G. Bejerano. Ruanda tendrá la primera base de drones para enviar medicamentos, 2015. Revisado el 5 de septiembre de 2025. https://blogthinkbig.com/ruanda-tendra-la-primera-base-drones-enviar-medicamentos.

[7] Abderahman Rejeb, Alireza Abdollahi, Karim Rejeb, and Horst Treiblmaier. Drones in agriculture: A review and bibliometric analysis. *Computers and electronics in agriculture*, 198:107017, 2022.

[8] Gastón A Addati and Gabriel Pérez Lance. Introducción a los UAV's, Drones o VANTs de uso civil. Technical report, Serie Documentos de Trabajo, 2014.

[9] Eleftheria Mitka and Spiridon G Mouroutsos. Classification of drones. *Am. J. Eng. Res*, 6(7):36–41, 2017.

[10] Raquel Flórez López and José Miguel Fernández Fernández. *Las redes neuronales artificiales*. Netbiblo, 2008.

[11] Mohit Goyal, Rajan Goyal, P Venkatappa Reddy, and Brejesh Lall. Activation functions. In *Deep learning: Algorithms and applications*, pages 4–10. Springer, 2019.

[12] Ameya D Jagtap and George Em Karniadakis. How important are activation functions in regression and classification? a survey, performance comparison, and future directions. *Journal of Machine Learning for Modeling and Computing*, 4(1), 2023.

[13] Shadman Sakib, Nazib Ahmed, Ahmed Jawad Kabir, and Hridon Ahmed. An overview of convolutional neural network: Its architecture and applications. 2019.

[14] Nikhil Ketkar and Jojo Moolayil. Convolutional neural networks. In *Deep learning with Python: learn best practices of deep learning models with PyTorch*, pages 197–242. Springer, 2021.

[15] ¿Qué es la visión artificial? *IBM*, 2021. Revisado el 5 de septiembre de 2025. https://www.ibm.com/es-es/think/topics/computer-vision.

[16] Iván S., Víctor S., and Tierra Infinita. La visión artificial y los campos de aplicación artificial. *Tierra Infinita*, 1:98–108, 04 2015.

[17] Eloi García. Visión artificial. *FUOC Fundación para la Universitat Oberta de Catalunya*, 2012.

[18] Solmaz Arezoomandan, John Klohoker, and David K Han. Data augmentation pipeline for enhanced uav surveillance. In *International Conference on Pattern Recognition*, pages 366–380. Springer, 2025.

[19] Houda Bichri, Adil Chergui, and Hain Mustapha. Investigating the impact of train / test split ratio on the performance of pre-trained models with custom datasets. *International Journal of Advanced Computer Science and Applications*, 15, 01 2024.

[20] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*, 2024.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. *SSD: Single Shot MultiBox Detector*, page 21–37. Springer International Publishing, 2016.

[24] ASIC. Nuevo clúster hpc de cálculo científico: Sirius. *Área de Sistemas de Información y Comunicaciones*. Revisado el 5 de septiembre de 2025. https://www.upv.es/entidades/asic/cluster-hpc-sirius/.

[25] Bihi Sabiri, Bouchra El Asri, and Maryem Rhanoui. Mechanism of overfitting avoidance techniques for training deep neural networks. In *ICEIS (1)*, pages 418–427, 2022.

Anexo: Objetivos de Desarrollo Sostenible

Tabla 1: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible.

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**		X		
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**		X		
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**		X		
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Este Trabajo Fin de Grado, titulado *"Detección de personas perdidas mediante vehículos aéreos no tripulados"*, es un proyecto que abarca puntos clave como el uso de drones y de sistemas de redes neuronales basados en Deep Learning. Por tanto, se puede relacionar de varias formas con los objetivos de desarrollo sostenibles.

En primer lugar, el trabajo desarrollado puede suponer una mejora en la calidad de vida de las personas. Los drones son máquinas que pueden funcionar de manera autónoma y son capaces de registrar grandes áreas de terreno en busca de personas desaparecidas, mejorando así los servicios de asistencia y rescate. Además, debido a su coste, pueden ser adquiridos y usados por las personas menos pudientes. Por estas razones, el proyecto se relaciona en buena medida con los objetivos de salud y bienestar (3) y reducción de las desigualdades (10).

El uso de drones con inteligencia artificial para tareas cada vez más complejas permite mejorar la eficiencia en numerosos escenarios (industria, agricultura, servicios, etc.). Por tanto, la investigación en este campo contribuye al trabajo decente y crecimiento económico (8) y a la industria, innovación e infraestructuras (9).

Por otro lado, los drones inteligentes, al usar energía eléctrica, no liberan gases perjudiciales para el medio ambiente (13) y pueden beneficiarse de la energía generada de forma limpia por las fuentes de energía renovables (7). Generalmente, no hacen ruido, por lo que tampoco molestan a la vida salvaje (15). Además, en su diseño y entrenamiento se busca ser lo más eficiente posible, y con simplemente un cambio de *software* (cambiar modelo de red neuronal) se puede reutilizar un dron para diferentes tareas (12).