UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería del Software

Inteligencia artificial aplicada a detección de objetivos pequeños por luz infrarroja

Artificial intelligence applied to small target detection using infrared light

Realizado por
Antonio Navajas Ortega

Tutorizado por
Miguel Ángel Molina Cabello

Paula Ariadna Jiménez Partinen

Departamento
Lenguaje y Ciencias de la Computación

MÁLAGA, junio de 2025

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

Inteligencia artificial aplicada a detección de objetivos pequeños por luz infrarroja

Artificial intelligence applied to small target detection using infrared light

Realizado por

Antonio Navajas Ortega

Tutorizado por

Miguel Ángel Molina Cabello

Paula Ariadna Jiménez Partinen

Departamento

Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2025

Fecha de defensa: Julio de 2025

Resumen

La identificación de pequeños objetivos en imágenes infrarrojas es esencial en ámbitos como la vigilancia, la industria aeroespacial y la medicina. Estas imágenes capturan la radiación térmica emitida por los objetos, lo que las hace valiosas en condiciones de baja visibilidad o en entornos con fondos complejos. Sin embargo, la detección de estos objetivos es complicada debido al bajo contraste, la elevada interferencia y el tamaño reducido de los elementos de interés respecto al total de la imagen.

Este proyecto analiza el rendimiento de modelos de aprendizaje profundo diseñados para detectar pequeños objetivos en imágenes infrarrojas. Se utilizaron redes neuronales convolucionales (CNN) por su capacidad para aprender patrones espaciales y contextuales complejos. Se evaluaron distintas arquitecturas y se ajustaron para lograr un equilibrio entre precisión y eficiencia computacional. Además, se realizó un análisis detallado de las métricas de rendimiento y la robustez de los modelos evaluados.

También se exploraron técnicas de preprocesamiento para mejorar los resultados, como la división del conjunto de datos según el tipo de imagen y la segmentación en mosaicos. Estas estrategias permiten adaptar el procesamiento a las particularidades de cada grupo y mejorar la detección en regiones pequeñas, incrementando así la precisión general.

Este trabajo busca ofrecer soluciones prácticas y escalables mediante tecnología avanzada, abordando los desafíos de la detección de pequeños objetivos en imágenes infrarrojas. La combinación de modelos optimizados, preprocesamiento adaptado y evaluación rigurosa permite avanzar en aplicaciones donde la precisión en condiciones adversas es crítica.

Palabras clave: imagen infrarroja, detección de objetivos pequeños, aprendizaje profundo, redes neuronales convolucionales, visión por computador.

Abstract

The identification of small targets in infrared images is essential in fields such as surveillance, aerospace, and medicine. These images capture the thermal radiation emitted by objects, making them valuable in low-visibility conditions or environments with complex backgrounds. However, detecting these targets is challenging due to low contrast, high interference, and the small size of the elements of interest relative to the entire image.

This project analyzes the performance of deep learning models designed to detect small targets in infrared images. Convolutional Neural Networks (CNNs) were used due to their ability to learn complex spatial and contextual patterns. Various architectures were evaluated and fine-tuned to achieve a balance between accuracy and computational efficiency. Additionally, a detailed analysis of performance metrics and the robustness of the evaluated models was conducted.

Preprocessing techniques were also explored to improve results, such as dataset division by image type and using tiled segmentation. These strategies allow the processing to be tailored to the characteristics of each group and enhance detection in small regions, thereby increasing overall accuracy.

This work aims to provide practical and scalable solutions through advanced technology, addressing the challenges of detecting small targets in infrared images. The combination of optimized models, adapted preprocessing, and rigorous evaluation enables progress in applications where precision under adverse conditions is critical.

Keywords: infrared image, small target detection, deep learning, convolutional neural networks, computer vision.

Índice

1 Introducción

1.1. Contexto

1.1.1. Acerca de las imágenes infrarrojas

En la actualidad, el análisis de imágenes infrarrojas [1] constituye un área clave en diversas aplicaciones, desde la seguridad y defensa hasta la monitorización ambiental y el diagnóstico médico. Las imágenes infrarrojas capturan la radiación térmica emitida por los objetos en el rango no visible del espectro electromagnético, lo que permite detectar patrones y características que no son perceptibles con luz visible.

El análisis de imágenes infrarrojas es particularmente valioso para la detección de objetivos pequeños en entornos complejos, donde factores como el bajo contraste, el ruido y la presencia de estructuras similares al objetivo presentan desafíos significativos. Estas limitaciones requieren el desarrollo de algoritmos robustos que permitan identificar y localizar objetivos de manera precisa y eficiente.

Según estudios recientes, la importancia del análisis de imágenes infrarrojas se refleja en las siguientes cifras y aplicaciones:

- El uso de imágenes infrarrojas para la vigilancia y la defensa ha crecido exponencialmente en la última década, se espera un crecimiento similar en la próxima década [2].
- La tecnología infrarroja permite detectar objetivos en condiciones de poca luz o visibilidad nula, lo que ha revolucionado áreas como la búsqueda y rescate, y la monitorización de fronteras.
- Los avances en redes neuronales convolucionales (CNN) han mejorado significativamente la precisión en la detección de objetivos pequeños en imágenes infrarrojas.
- La investigación en sistemas de detección ha llevado a la creación de nuevos datasets y desafíos, como el ICPR Resource-Limited Infrared Small Target Detection Challenge,

que fomenta la innovación en esta área.

Dentro del análisis de imágenes infrarrojas, la detección de objetivos pequeños es un problema crítico. Los objetivos pequeños, como vehículos o drones, a menudo se pierden en el ruido térmico del fondo o carecen de suficiente contraste para ser detectados mediante métodos tradicionales.

Tal como se muestra en la Figura 1, se puede observar un ejemplo de predicción realizada con YOLOv8.

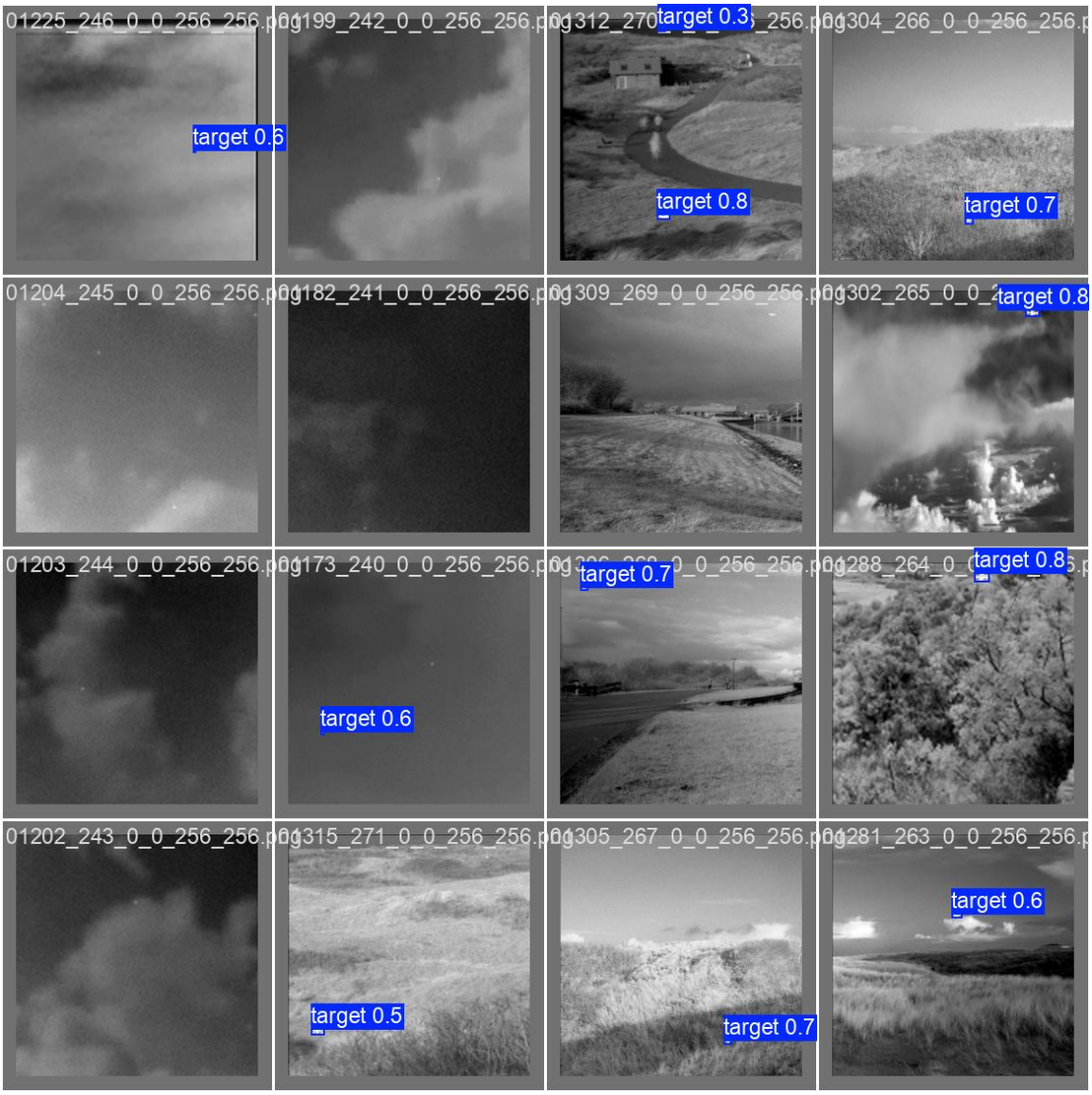


Figura 1: Ejemplo de predicción YOLOv8.

Para abordar estos desafíos, se han desarrollado técnicas avanzadas basadas en redes neuronales profundas, especialmente arquitecturas como YOLO (You Only Look Once) [3], que permiten detectar y localizar objetivos en tiempo real.

1.1.2. Redes neuronales aplicadas a imágenes infrarrojas

El uso de redes neuronales convolucionales (CNN, *Convolutional Neural Networks*) ha revolucionado el campo de la visión artificial aplicada a imágenes infrarrojas [4]. Estas redes permiten identificar patrones complejos y han mostrado un rendimiento superior en comparación con los métodos tradicionales de procesamiento de imágenes.

- Las arquitecturas preentrenadas, como YOLO, han sido adaptadas y optimizadas para la detección de objetivos pequeños en imágenes infrarrojas.
- Estas arquitecturas suelen requerir técnicas de preprocesamiento previas, como la normalización del histograma y la supresión de ruido, que se aplican externamente para mejorar la calidad de las imágenes y aumentar la precisión de detección.
- Los modelos avanzados son capaces de procesar grandes volúmenes de datos en entornos con recursos limitados, lo que permite su uso incluso en dispositivos de bajo coste.

Tal como se muestra en la Figura 2, se presenta un ejemplo de detección automática de objetivos pequeños utilizando YOLO.

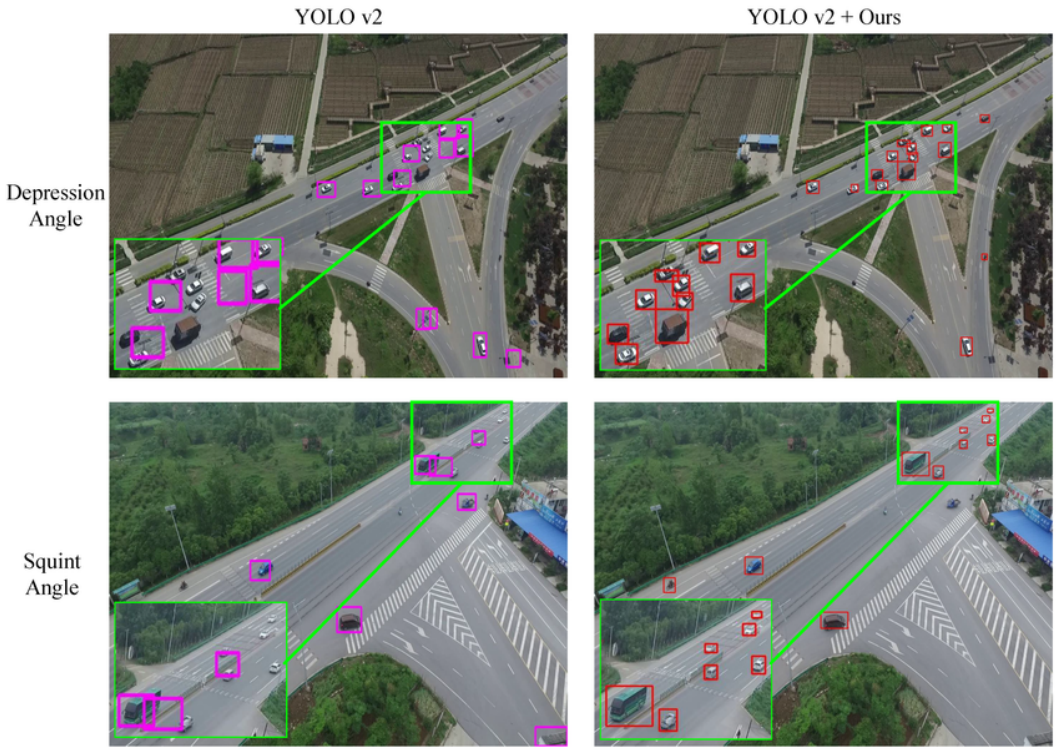


Figura 2: Ejemplo de detección automática de objetivos pequeños con YOLO[5].

Los resultados obtenidos en diversos estudios y desafíos internacionales subrayan la eficacia de las redes neuronales para abordar problemas complejos de detección en imágenes infrarrojas, contribuyendo a la evolución de aplicaciones prácticas en múltiples dominios.

1.2. Motivación

Los avances recientes en Inteligencia Artificial han facilitado el análisis de imágenes infrarrojas [6], lo que permite una detección más precisa de objetos pequeños, incluso en condiciones adversas como bajo contraste y ruido. Esta capacidad es crucial para mejorar la identificación y clasificación de estos objetos, con aplicaciones en áreas como la vigilancia, la navegación autónoma y el monitoreo ambiental.
Estos objetivos pequeños pueden representar anomalías dentro del campo de visión, lo que los hace críticos en aplicaciones como defensa o vigilancia.

Los modelos propuestos no buscan reemplazar la intervención humana, sino más bien complementar el trabajo de los expertos, proporcionándoles herramientas automáticas para verificar sus observaciones y tomar decisiones más informadas y consistentes.

1.3. Objetivos

El objetivo principal de este proyecto es investigar la viabilidad de un sistema basado en redes neuronales convolucionales (CNN) para la detección de objetivos pequeños en imágenes infrarrojas en tiempo real.

Se busca explorar el uso de modelos preentrenados, concretamente de la familia YOLOv8, ajustándolos mediante técnicas de preprocesamiento y configuración adaptada.

Además, se pretende analizar el comportamiento del sistema en diferentes condiciones del conjunto de datos, considerando aspectos como el origen de captura, el espectro infrarrojo y la resolución de las imágenes.

Finalmente, se busca aplicar estrategias complementarias como la segmentación en mosaicos para favorecer la robustez y capacidad de generalización del sistema.

1.4. Áreas del Conocimiento

Los dominios de conocimiento involucrados en el desarrollo de este proyecto son:

1. **Visión por Computador (*Computer Vision*)**
    - Procesamiento de imágenes.
    - Detección de objetos.
2. **Aprendizaje Profundo (*Deep Learning*)**
    - Redes neuronales convolucionales (CNNs).
    - Ajuste Fino (*Finetuning*).
3. **Ingeniería de Datos (*Data Science*)**
    - Anotación de datos.
    - Limpieza y gestión de conjuntos de datos.
    - Aumento de datos (*Data Augmentation*).
    - Validación cruzada (*Cross-validation*).
4. **Evaluación de Modelos**
    - Métricas: mAP, Precision, Recall, IoU .
    - Reducción de falsos positivos/negativos.
5. **Conocimientos de Dominio Específico**
    - Imágenes infrarrojas.
    - Conocimientos intermedios en programación y básicos en python.
    - Características particulares de los objetivos pequeños.
    - Nociones básicas de matemáticas y estadística.

1.5. Estructura del documento

El trabajo presentado sigue la siguiente estructura:

- En la introducción, Sección 1, se expone el contexto del problema, la motivación detrás del trabajo y los objetivos que se persiguen.
- En la Sección 2 se presentan los fundamentos teóricos necesarios, incluyendo conceptos de Inteligencia Artificial, Redes Neuronales Convolucionales y el uso de imágenes infrarrojas, junto con las tecnologías y métricas empleadas.
- La Sección 3 describe el conjunto de datos utilizado, detallando sus características.
- La metodología empleada para el desarrollo del trabajo es explicada, así como el enfoque en la Sección 4.
- Se realizan análisis de requisitos y diseño de la arquitectura en la Sección 5.
- En la Sección 6 se encuentra la explicación de la configuración necesaria previa al entrenamiento.
- Se presentan diferentes fases de experimentación, donde se comentan los resultados y se analiza el rendimiento a lo largo de la Secciones 7, 8 y 9.
- En la Sección 10 se discuten impresiones de los experimentos, así como patrones detectados.
- Las conclusiones extraídas del trabajo se presentan en la Sección 11 junto con propuestas para futuras líneas de investigación y mejora.
- Finalmente, se incluyen apéndices con información adicional como manual de instalación, de ejecución e información adicional de la arquitectura YOLO, además de un glosario con términos clave empleados en el documento.
- Las referencias bibliográficas se encuentran al final del documento.

2

Estado del arte

2.1. Inteligencia Artificial

Recientemente, una nueva revolución industrial se está abriendo camino, cambiando la forma de actuar de negocios e instituciones.

Se le conoce como "Industria 4.0" [7], una revolución tecnológica, que da lugar a recientes campos como *Big Data*, Inteligencia Artificial, Ciberseguridad o Computación en la Nube.

Como se muestra en la Figura 3, la Industria 4.0 representa una nueva era en la que los sistemas cibernéticos y la automatización juegan un papel clave en la evolución de los procesos industriales.

Figura 3: Elementos de la Industria 4.0 [7].

Dado lo reciente que es el ámbito de la Inteligencia Artificial, no existe una definición única

o completamente precisa sobre este término. Sin embargo, se pueden proporcionar algunas aproximaciones que intentan describirlo [8]:

- Consiste en aplicar tecnologías que permitan a máquinas o programas simular comportamientos que normalmente asociamos con la inteligencia humana.

- Se trata de una rama de la informática enfocada en crear sistemas capaces de replicar habilidades humanas como el aprendizaje, la toma de decisiones y la comprensión.

En sus comienzos, la Inteligencia Artificial se basaba en seguir un conjunto de reglas fijas y previamente definidas para llevar a cabo ciertas tareas. No obstante, a medida que los problemas se tornaban más complejos y los entornos presentaban mayor incertidumbre, este enfoque dejó de ser eficiente y se consideró obsoleto.

De esta necesidad surgió el Aprendizaje Automático (*Machine Learning*), una subárea de la Inteligencia Artificial que permite a las máquinas aprender de los datos suministrados, en lugar de depender exclusivamente de reglas predefinidas.

En el Aprendizaje Automático podemos identificar tres enfoques principales:

- Aprendizaje Supervisado: Este método utiliza datos de entrada junto con las correspondientes salidas esperadas, como por ejemplo, la clasificación en categorías específicas.

- Aprendizaje No Supervisado: En este caso, se trabaja únicamente con los datos de entrada, sin información sobre las salidas, buscando identificar patrones o grupos dentro del conjunto de datos.

- Aprendizaje por Refuerzo: Se basa en que la máquina aprenda observando las consecuencias de sus acciones a través de la interacción con el entorno.

Dentro de este campo, destaca el Aprendizaje Profundo (*Deep Learning*), una disciplina que emplea redes neuronales artificiales para identificar patrones complejos y extraer información oculta en los datos de entrada, lo que resulta esencial para abordar tareas de aprendizaje avanzadas.

2.2. Red Neuronal

2.2.1. Red Neuronal Artificial

Las redes neuronales artificiales [9] se inspiran en la forma de transmitir la información de las neuronas biológicas. En el cerebro humano, las neuronas están compuestas por dendritas, el soma y el axón. El proceso funciona de la siguiente manera: las dendritas reciben impulsos nerviosos de otras neuronas, que luego son procesados en el soma. Después de este procesamiento, la información se transmite a través del axón hacia las neuronas adyacentes.

Tal como se muestra en la Figura 4, una representación común de una neurona artificial incluye las entradas, los pesos, la función de activación y la salida.

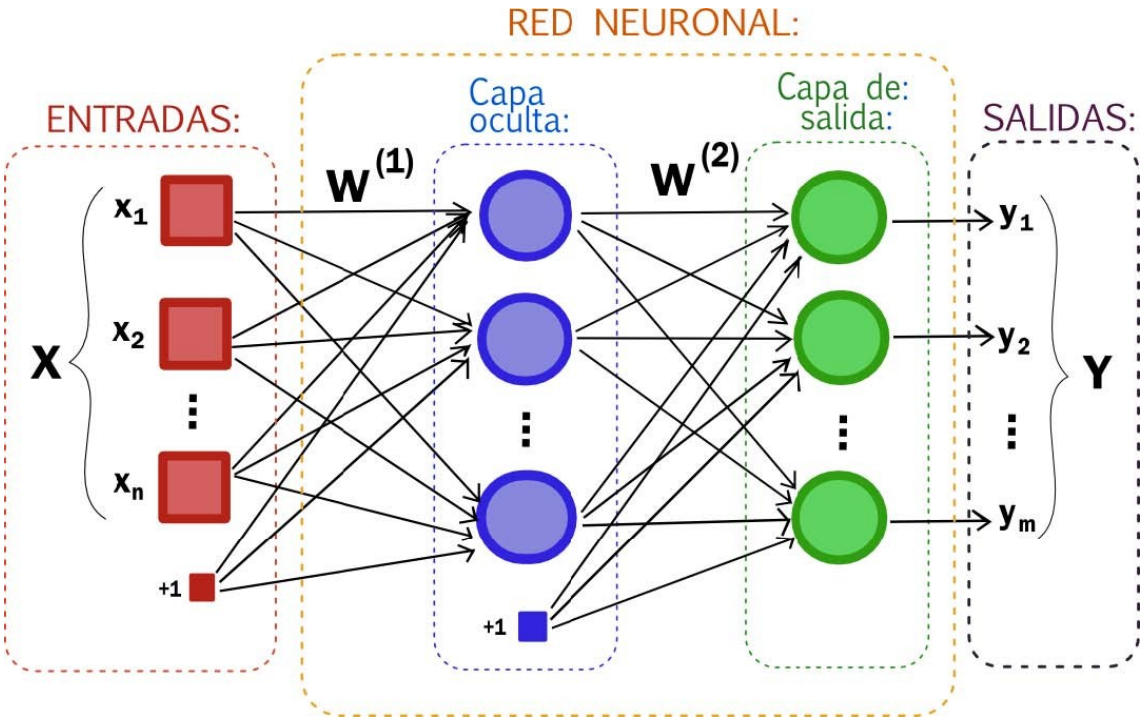


Figura 4: Representación de una red neuronal artificial [10].

En el caso de las neuronas artificiales, el impulso nervioso recibido por la neurona es determinado por la suma de las entradas multiplicadas por sus respectivos pesos. Este valor resultante se pasa luego por una función de activación, que genera la salida de la neurona.

Una red neuronal artificial está formada por neuronas artificiales conectadas entre sí y organizadas en capas. Cada capa consiste en un grupo de neuronas cuyas entradas provienen de la capa anterior, o de los datos de entrada si es la primera capa, y cuyas salidas sirven como entradas para la capa siguiente.

La capa inicial de la red neuronal artificial se denomina capa de entrada, y la última capa es la de salida. Las capas que se encuentran entre la de entrada y la de salida son llamadas capas ocultas, ya que tanto las entradas como las salidas de estas capas son desconocidas.

El concepto de Aprendizaje Profundo (*Deep Learning*) se deriva del uso de múltiples capas ocultas en las redes neuronales.

El proceso de entrenamiento de una red neuronal artificial implica ajustar los pesos de las entradas en todas las neuronas para que las respuestas de la capa de salida se acerquen lo más posible a los datos conocidos.

2.2.2. Red Neuronal Convolucional

Las redes neuronales convolucionales [11] son un tipo de red neuronal, comúnmente utilizada en el campo del Aprendizaje Profundo para tareas de visión computacional, como la clasificación de imágenes o la detección y localización de objetos.

Como se observa en la Figura 5, se presenta la estructura general de una red neuronal convolucional.

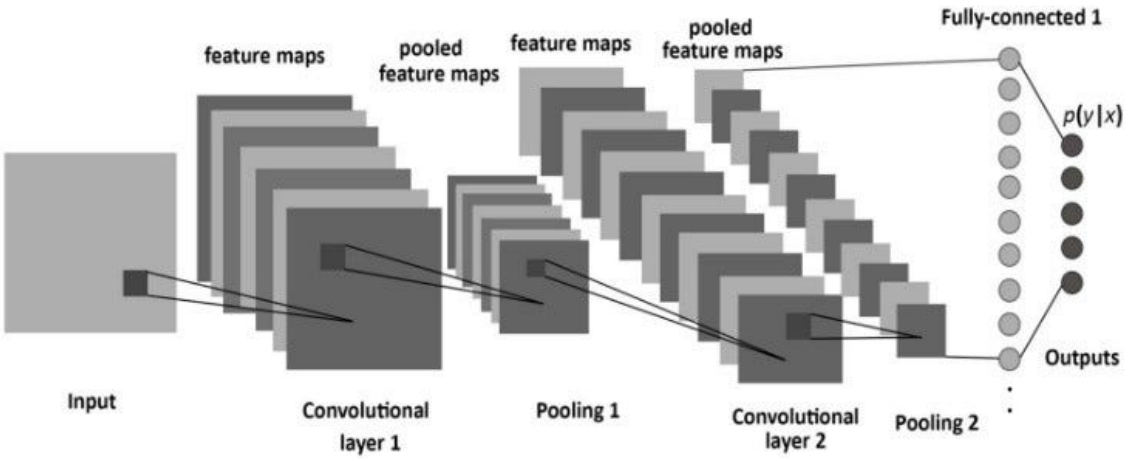


Figura 5: Imagen red neuronal convolucional [12].

- Se comienza con una imagen de entrada, sobre la que se realizan varias operaciones de convolución y *pooling*, seguidas por una serie de capas *fully connected* (completamente conectadas). La imagen de entrada se transforma en un tensor multidimensional que representa la información de píxeles y se procesa a través de distintas capas convolucionales.

- Las capas convolucionales tienen como objetivo extraer características de la imagen de entrada. Cada capa convolucional genera un mapa de características, que representa las regiones de la imagen donde se han detectado ciertos patrones como bordes o texturas. La operación de convolución se lleva a cabo mediante un kernel o filtro, que es una matriz con dimensiones más pequeñas que la imagen de entrada.

- Si la tarea es una clasificación multiclase, la salida pasa por una función de activación *softmax* [13].

En el contexto de las CNNs, la función *softmax* [13] se emplea habitualmente en la capa de salida para realizar tareas de clasificación multiclase. Esta función permite interpretar las salidas del modelo como probabilidades asociadas a cada clase.

Supongamos que una CNN ha procesado una imagen a través de varias capas convolucionales, de activación y de agrupamiento, y finalmente entrega un vector de características $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ en la capa final. La función *softmax* transforma este vector en una distribución de probabilidad mediante la expresión:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad para\ i = 1, 2, \ldots, n \tag{1}$$

Cada valor z_i representa la evidencia a favor de la clase i antes de aplicar la normalización. La función *softmax* asigna a cada clase una probabilidad relativa, amplificando las diferencias entre los valores más altos y reduciendo los demás.
Esto permite que el modelo seleccione la clase con mayor probabilidad como predicción final, lo cual es especialmente útil en tareas como clasificación de imágenes, reconocimiento de objetos y detección de anomalías visuales.

2.3. Tecnologías empleadas

2.3.1. Lenguaje de programación

Python[14] es el lenguaje elegido ya que ofrece una programación versátil que soporta múltiples paradigmas y destaca por la claridad y sencillez de su sintaxis, lo que facilita su uso y lo hace altamente accesible para los desarrolladores. Su diseño intuitivo permite un desarrollo ágil y eficiente, logrando una gran abstracción. Debido a librerías como PyTorch o TensorFlow es el lenguaje predilecto para tareas de inteligencia artificial. Versión de Python utilizada: “3.11.11” .

2.3.2. Entorno de Ejecución

Las ejecuciones de los experimentos se han realizado en dos entornos:

- Google Colaboratory. Es una plataforma gratuita basada en la nube que permite ejecutar código Python en un entorno de cuadernos Jupyter, ofreciendo acceso a GPU y TPU para desarrollar y entrenar modelos de aprendizaje automático de manera eficiente. Además de integrarse con Google Drive para tener acceso a archivos y en este caso nuestro dataset. Dispone de una unidad de procesamiento gráfico de las siguientes características: Tesla T4 de 15GB de VRAM.
- Servidor Grupo de Investigación ICAI (UMA) : Servidor dedicado de la universidad que dispone de los recursos computacionales para realizar experimentos costosos y pesados computacionalmente. La gran mayoría de entrenamientos se realizarán aquí por conveniencia debido a las restricciones del plan gratuito de Google Colaboratory. Dispone de una unidad de procesamiento gráfico de las siguientes características: NVIDIA RTX 3080 de 10GB de VRAM.
- Máquina Local del Alumno : Durante un breve periodo y para ínfimas pruebas, se ha utilizado una máquina de uso personal con la siguiente unidad de procesamiento gráfico: NVIDIA GTX 1050 de 3GB de VRAM.

2.3.3. Bibliotecas y Herramientas

Principales bibliotecas y herramientas utilizadas :

- **Ultralytics**[15]. Versión 8.3.59. Es una herramienta avanzada de inteligencia artificial enfocada en la visión por computadora. Destaca por su facilidad de uso, eficiencia y capacidad para implementar modelos de detección de objetos de alto rendimiento, como la popular arquitectura YOLO (*You Only Look Once*) [3]. Ultralytics es ampliamente utilizado en aplicaciones como la vigilancia, el análisis de imágenes médicas y la automatización industrial. Utiliza varias librerías como PyTorch[4], Numpy, TensorBoard, Torchvision y Pandas.

- **OpenCV**[16]. Facilita el procesamiento de imágenes y videos, permitiendo tareas como la preprocesamiento de datos y la visualización de resultados. Usado para crear

archivos de bounding boxes y preprocesamiento del conjunto de datos.

- **Sklearn**[17]. Proporciona herramientas simples y eficientes para análisis de datos, modelado y evaluación de modelos. Usado en creación de la validación cruzada.

- **Matplotlib** [18]. Matplotlib es una biblioteca para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays. Usado en creación de algunos histogramas.

- **SAHI**[19]: Es una herramienta utilizada en la visión por computadora para realizar detección de objetos en imágenes grandes, dividiéndolas en "porciones" más pequeñas y facilitando la predicción precisa en imágenes de gran tamaño o de alta resolución. Usado para fragmentar el conjunto de datos en porciones más manejables.

- **Anaconda**[20]: Es una distribución de Python que facilita la instalación de paquetes y la gestión de entornos virtuales, especialmente útil para el análisis de datos, machine learning y ciencia de datos. Usado en el servidor para no interferir con el trabajo del resto de usuarios.

- **FileZilla**[21]: Es un cliente de FTP (*File Transfer Protocol*) de código abierto que se utiliza para transferir archivos entre un ordenador local y un servidor de manera rápida y segura. Usado para importar y exportar ficheros entre la máquina local y el servidor.

- **X2Go**[22]: Es una herramienta de acceso remoto (SSH) que permite conectarse a escritorios de sistemas Linux de forma eficiente, ofreciendo una experiencia de usuario fluida incluso con conexiones de baja velocidad. Usado por comodidad y legibilidad que ofrece al trabajar con servidores.

2.4. Arquitectura YOLO

YOLO (*You Only Look Once*) es una arquitectura de red neuronal convolucional diseñada para la detección de objetos en imágenes en tiempo real. A diferencia de otros enfoques tradicionales que dividen el proceso en etapas (como la generación de propuestas de regiones y luego su clasificación), YOLO lo realiza todo en una sola pasada, lo que la hace extremadamente rápida y eficiente.

YOLO divide la imagen de entrada en una cuadrícula (por ejemplo, 13×13 en YOLOv1). Cada celda de la cuadrícula predice un número fijo de ***bounding boxes***, junto con la probabilidad de que cada una contenga un objeto y la clase del objeto si lo hay. Esta predicción directa convierte la detección de objetos en un problema de regresión, evitando procesos más lentos como el escaneo por regiones.

La arquitectura clásica de YOLO incluye:

- ***Backbone***: es la red convolucional encargada de extraer características. En versiones antiguas se utilizaba ***Darknet***, mientras que en versiones modernas se puede usar ***CSP-Darknet*** o incluso **EfficientNet**.
- ***Neck***: conecta el **backbone** con la cabeza de detección. Utiliza estructuras como **FPN** (*Feature Pyramid Network*) o **PANet** para combinar características de diferentes escalas.
- ***Head*** (cabeza de detección): realiza las predicciones finales (clases, coordenadas y confianza) a diferentes escalas para detectar objetos grandes y pequeños.

Tal como se muestra en la Figura 6, la arquitectura típica de los modelos de detección de objetos es la siguiente:

Desde su primera versión, YOLO ha tenido múltiples mejoras:

- **YOLOv1–v3**: mejoras en precisión y uso de anclas para predecir ***bounding boxes***.
- **YOLOv4**: optimización del rendimiento en hardware real sin sacrificar precisión.
- **YOLOv5**: desarrollado por Ultralytics, incorpora técnicas modernas como ***auto-learning bounding box anchors***, ***augmentation***, y mejoras en la facilidad de uso.
- **YOLOv6, v7, v8**: versiones más recientes con mejoras tanto en velocidad como en precisión, adaptadas para tareas más amplias como segmentación y clasificación multietiqueta.

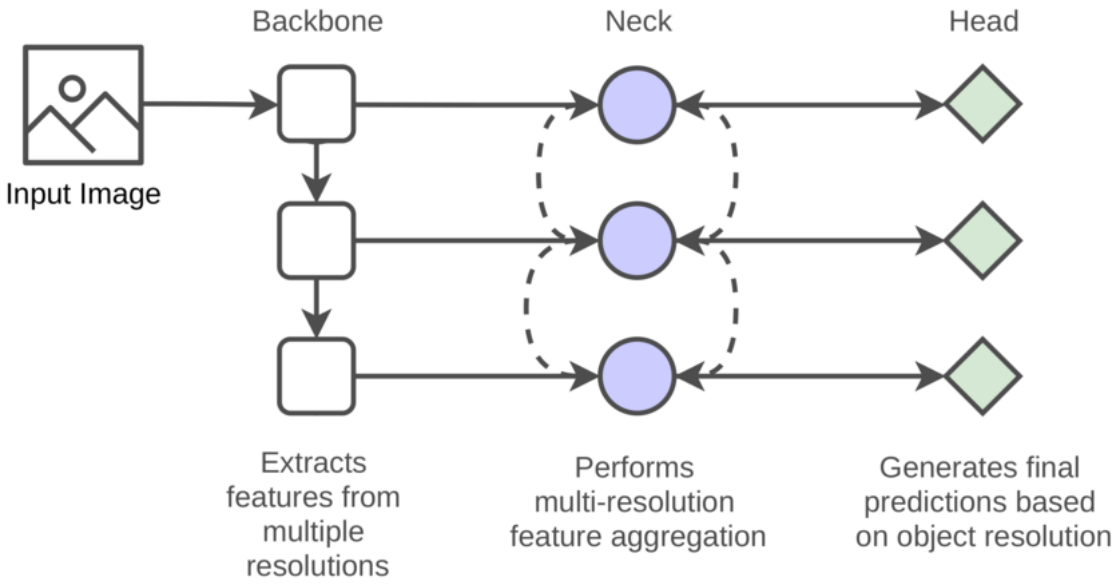


Figura 6: Esquema de la arquitectura tronco-cuello-cabeza [23].

Entre sus principales ventajas destacan la alta velocidad de inferencia, su eficiencia en dispositivos con recursos limitados y un buen equilibrio entre precisión y rendimiento.

YOLO se utiliza ampliamente en diversos campos, como la vigilancia en tiempo real, la conducción autónoma, la agricultura inteligente o la medicina, por ejemplo en la detección de tumores y anomalías.

El grupo YOLOv8 es el elegido para el entrenamiento ya que es la versión más estable y documentada. Existen 5 tipos de subgrupos:

- **YOLOv8n (*nano*):**
  - Muy ligero y rápido.
  - Aprox. 3 millones de parámetros.
  - Ideal para dispositivos con recursos limitados o aplicaciones en tiempo real con baja latencia.
- **YOLOv8s (*small*):**
  - Balance entre velocidad y precisión.
  - Aprox. 11 millones de parámetros.
  - Adecuado para tareas en dispositivos móviles o edge computing.
- **YOLOv8m (*medium*):**

- Aprox. 25 millones de parámetros.
- Mejor precisión que los modelos nano y small.
- Ideal para entornos con recursos moderados (GPU de gama media).

- **YOLOv8l (*large*):**
  - Aprox. 43 millones de parámetros.
  - Alta precisión.
  - Ideal para servidores o GPUs potentes.

- **YOLOv8x (*extra large*):**
  - Aprox. 68 millones de parámetros.
  - Máxima precisión entre los modelos de YOLOv8.
  - Ideal para tareas donde la precisión es prioritaria y no hay restricciones de recursos.

Tal como se observa en la Figura 7, se presenta una comparación entre distintos métodos de detección y sus respectivas prestaciones, destacando el compromiso entre precisión y velocidad de inferencia.

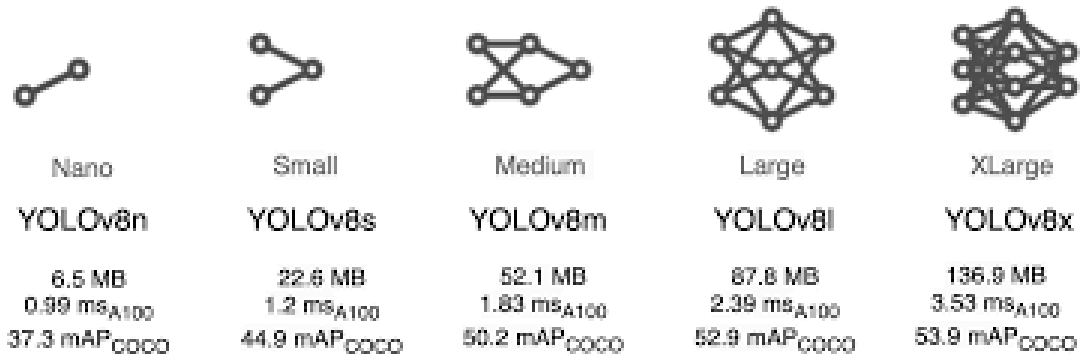


Figura 7: Comparación de métodos y prestaciones [24].

Los métodos YOLO de Ultralytics están preentrenados usando datasets públicos y grandes como COCO (*Common Objects in Context*), por tanto la totalidad de entrenamientos de este documento se consideran *Finetuning* total (sin congelar capas) y no se consideran desde cero.

2.5. Función de pérdida

La función de pérdida [25] es una parte fundamental para optimizar los modelos de aprendizaje automático. En particular, YOLOv8 utiliza una fórmula compuesta por tres componentes principales, cuyo funcionamiento puede consultarse en su código fuente [26]:

- ***Box Loss*:** mide la precisión en la predicción de las cajas delimitadoras mediante **CIoU loss**, una variante de IoU que incorpora distancia, aspecto y penalizaciones geométricas [27].
- ***CLS Loss*:** evalúa la exactitud en la clasificación de las clases mediante ***Varifocal Loss***, que pondera las predicciones según su confiabilidad [28].
- ***DFL Loss*:** mejora la regresión de los bordes de las cajas mediante ***Distribution Focal Loss***, que discretiza los bordes para mayor precisión [29].

Fórmula general de la pérdida en YOLOv8:

$$\mathcal{L}_{\text{total}} = B \cdot (\lambda_{\text{box}} \cdot \mathcal{L}_{\text{box}} + \lambda_{\text{cls}} \cdot \mathcal{L}_{\text{cls}} + \lambda_{\text{dfl}} \cdot \mathcal{L}_{\text{dfl}}) \tag{2}$$

Donde:

- B: tamaño del lote
- $\lambda_{\text{box}}, \lambda_{\text{cls}}, \lambda_{\text{dfl}}$: hiperparámetros que ponderan cada componente de la pérdida.
- $\mathcal{L}_{\text{box}}$: pérdida de localización.
- $\mathcal{L}_{\text{cls}}$: pérdida de clasificación.
- $\mathcal{L}_{\text{dfl}}$: pérdida de regresión basada en distribución.

Para comprender las dificultades que plantean los objetos pequeños en la detección, es importante analizar en detalle ***Loss Box*** (también llamada por Ultralytics $loss\_iou$):

$$loss\_iou = \frac{\sum_i (1 - \text{CIoU}_i) \cdot w_i}{\sum_i w_i} \tag{3}$$

Donde:

$$\begin{cases} \text{CIoU}_i & \text{es el Complete Intersection over Union para el objeto } i \\ w_i & \text{es el peso asignado al objeto } i, \text{derivado de su confianza} \\ i & \text{índice que recorre los objetos positivos (foreground)} \end{cases} \tag{4}$$

El CIoU es una variante mejorada del IoU con penalizaciones.

$$\text{CIoU}(B, B^{gt}) = \text{IoU}(B, B^{gt}) - \frac{\rho^2(\mathbf{B}, \mathbf{B}^{gt})}{c^2} - \alpha v \tag{5}$$

Donde:

- $\rho(\mathbf{B}, \mathbf{B}^{gt})$ es la distancia euclidiana entre los centros de ambas cajas.
- c es la diagonal del cuadro que encierra ambas cajas.
- v es el término que penaliza la diferencia en la relación de aspecto.
- α es un coeficiente de balance para v.

El IoU se calcula como:

$$\text{IoU}(B, B^{gt}) = \frac{|B \cap B_{GT}|}{|B \cup B_{GT}|} \tag{6}$$

Aplicando el principio de inclusión-exclusión:

$$\text{IoU}(B, B^{gt}) = \frac{B_{\text{intersección}}}{B + B_{GT} - B_{\text{intersección}}} \tag{7}$$

Desglosando cada término en función del producto de sus dimensiones coordenadas $(x_1, y_1), (x_2, y_2)$:

$$\text{IoU}(B, B^{gt}) = \frac{w_{\text{int}} \times h_{\text{int}}}{w \times h + w^{gt} \times h^{gt} - w_{\text{int}} \times h_{\text{int}}} \tag{8}$$

Donde:

$$\begin{cases} w = x_2 - x_1 & \text{ancho de la caja predicha } B \\ h = y_2 - y_1 & \text{alto de la caja predicha } B \\ w^{gt} = x_2^{gt} - x_1^{gt} & \text{ancho de la ground truth } B_{gt} \\ h^{gt} = y_2^{gt} - y_1^{gt} & \text{alto de la ground truth } B_{gt} \\ w_{\text{int}} = \text{máx}(0, \text{mín}(x_2, x_2^{gt}) - \text{máx}(x_1, x_1^{gt})) & \text{ancho de la intersección} \\ h_{\text{int}} = \text{máx}(0, \text{mín}(y_2, y_2^{gt}) - \text{máx}(y_1, y_1^{gt})) & \text{alto de la intersección} \end{cases} \tag{9}$$

Cuando los objetos son pequeños, es decir, cuando w^{gt} y h^{gt} son valores reducidos, la caja de ground truth cubre un área pequeña. En este caso, pequeñas variaciones en el ancho, altura o posición de la caja predicha afectan significativamente a la IoU.

Por ejemplo, para un objeto pequeño con $w^{gt} = 10$ y $h^{gt} = 10$, una predicción centrada con $w = 9$ y $h = 9$ puede obtener una IoU alta, alrededor de 0.81. Sin embargo, un ligero desplazamiento o un cambio a $w = 11$, $h = 11$ reduce considerablemente la intersección y puede hacer que la IoU baje a cerca de 0.5 o menos.

En cambio, para objetos grandes, como con $w^{gt} = 100$ y $h^{gt} = 100$, una variación pequeña de uno o dos píxeles representa una proporción menor respecto al área total, por lo que la IoU apenas se ve afectada. Esto implica que el modelo es menos sensible a pequeños errores para objetos grandes.

Conclusión: Para objetos pequeños, errores mínimos en tamaño o posición provocan una gran reducción en la IoU, lo que genera una penalización mayor durante el entrenamiento y dificulta que el modelo aprenda a detectarlos con precisión.

2.6. Métricas

Las métricas que más atención van a recibir son Precisión, Recuperación (*Recall* en inglés) y *mean Average Precision* 0.5 (mAP@0.5), sin embargo para entender mejor estos conceptos es beneficioso entender la matriz de confusión.

Aunque la matriz de confusión se usa típicamente en clasificación, también puede aplicarse a la detección de objetos al analizar la precisión con la que el modelo detecta y clasifica objetos en imágenes. En este contexto, se basa en comparaciones entre los objetos detectados por el modelo y los objetos reales (ground truth).

Elementos clave de la matriz de confusión:

- *True Positives* (TP): El modelo detecta un objeto de forma correcta (clase correcta y buena ubicación).
- *False Positives* (FP): El modelo detecta algo que no debería (detección incorrecta o sin

objeto real correspondiente).

- *False Negatives* (FN): El modelo no detecta un objeto que sí estaba presente.
- *True Negatives* (TN): No suele aplicarse directamente en detección de objetos, ya que sería "no detectar" donde no hay nada, lo cual es demasiado común en imágenes.

La siguiente Tabla 1 se puede observar como luciría una matriz de confusión para detección de una sola clase.

Predicción / Realidad	**Positivo (Objeto)**	**Negativo (Sin objeto)**
Positivo (Predicción de objeto)	TP	FP
Negativo (Predicción de no objeto)	FN	N/A

Tabla 1: Matriz de Confusión para un problema binario.

Métricas derivadas de la matriz de confusión. Puedes explicar las métricas principales derivadas de **TP**, **FP** y **FN** como sigue:

- Precision (Precisión): Qué proporción de las detecciones del modelo fueron correctas.

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

- Recall (Sensibilidad): Qué proporción de los objetos reales fueron detectados.

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

- *F1 Score*: Media armónica entre precisión y recall, útil cuando hay desequilibrio.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{12}$$

- mAP (*mean Average Precision*): Es la métrica principal en detección de objetos. Se calcula promediando la precisión en distintos niveles de IoU para cada clase y luego promediando entre clases:
  - **mAP@0.5**: Usa IoU > 0.5 como umbral.
  - **mAP@0.5:0.95**: Media de la precisión desde IoU 0.5 hasta 0.95, con pasos de 0.05.

3

Conjunto de datos

En este trabajo se ha utilizado el conjunto de datos **ICPR 2024** [30], correspondiente al desafío "Resource-Limited Infrared Small Target Detection Challenge". Este conjunto contiene 9000 imágenes infrarrojas en blanco y negro, diseñadas específicamente para el desarrollo de algoritmos de detección de pequeños objetivos en condiciones de recursos limitados. En consecuencia, los modelos deben minimizar el número de parámetros y el uso de memoria.

El conjunto de datos está compuesto por siete datasets públicos —**SIRST-V2**, **IRSTD-1K**, **IRDST**, **NUDT-SIRST**, **NUDT-SIRST-Sea**, **NUDT-MIRSDT**, **Anti-UAV**—, además de un dataset adicional desarrollado por la National University of Defense Technology (NUDT).

La Figura 8 ofrece una visión general de los diferentes tipos de imágenes que lo conforman:

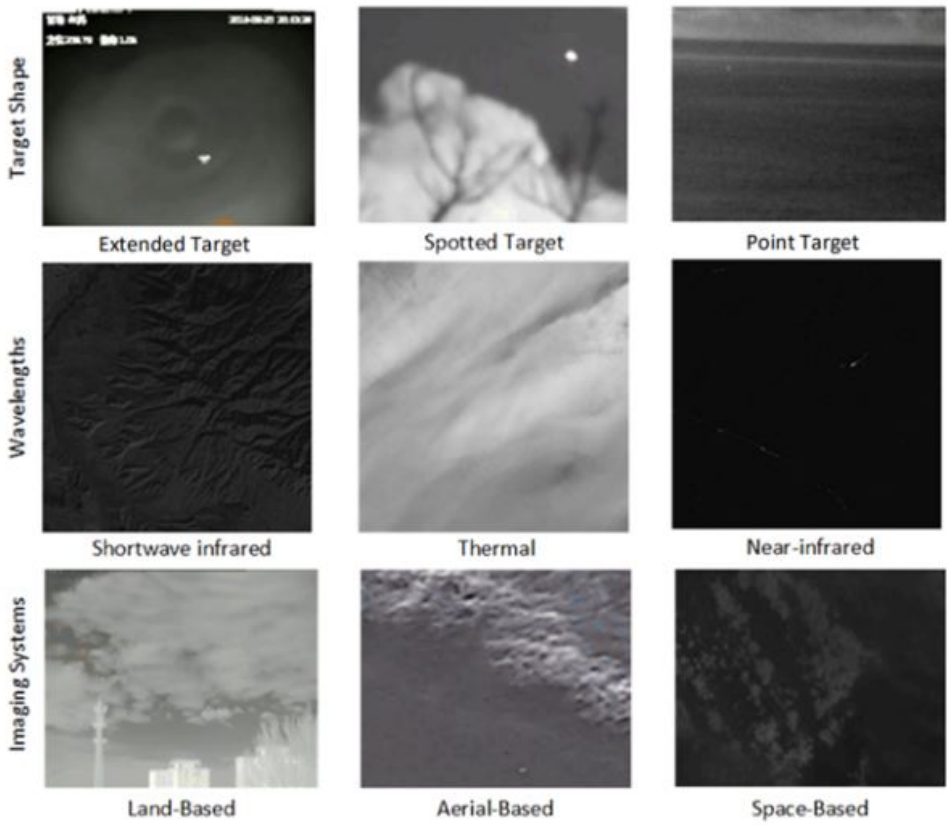


Figura 8: Resumen visual del dataset proporcionado por ICPR 2024 [30].

Las imágenes pueden clasificarse según tres dimensiones principales: el origen de captura, el espectro de luz utilizado y el tipo de objetivo.

Además, el conjunto de datos incluye un archivo llamado `statistics.txt`, donde se especifican metadatos relevantes de cada imagen, tales como:

- **ID de imagen** (00001–08851)
- **Origen de captura:** AIR, LAND, SPACE
- **Espectro infrarrojo:** LWIR, SWIR, NIR
- **Resolución:** ancho y alto en píxeles
- **Número total de píxeles objetivo (correspondientes a la máscara en blanco)**

Este archivo permite realizar una partición automatizada de los datos para análisis específicos. Asimismo, se proporcionan máscaras que segmentan los objetivos en cada imagen, lo que facilita la generación de ***bounding boxes*** para el entrenamiento y la evaluación de modelos.

3.1. Origen de captura

Según el lugar desde el cual se tomaron las imágenes, se distinguen tres categorías:

- **Land:** capturadas desde la superficie terrestre.
- **Air:** capturadas desde drones, globos o aeronaves.
- **Space:** capturadas desde satélites ubicados en órbita.

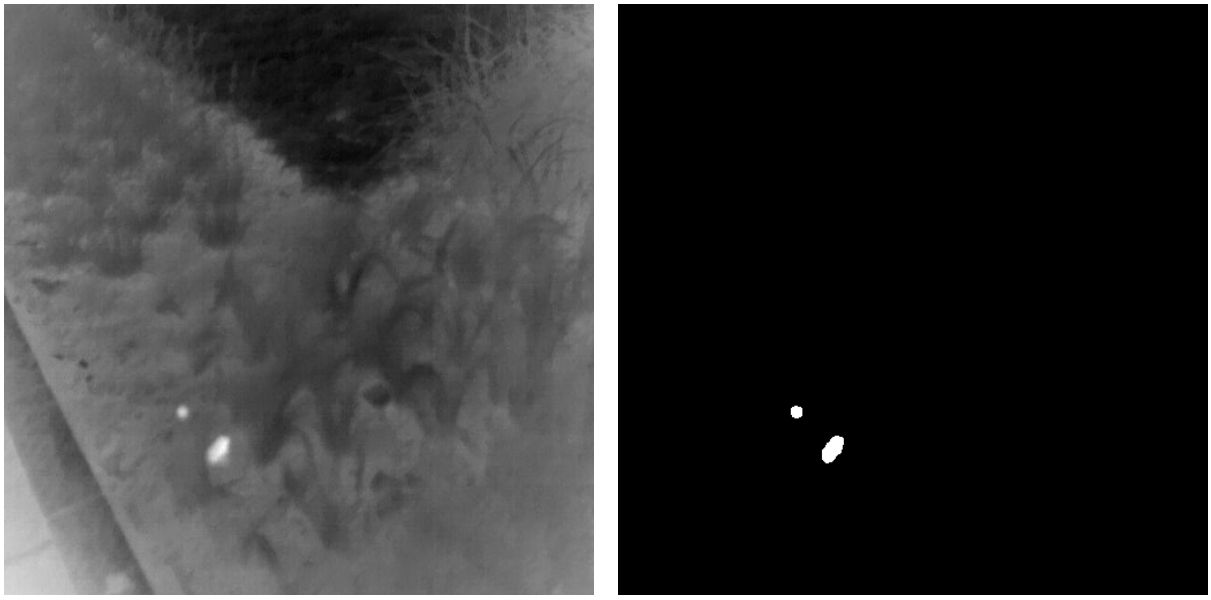
La Tabla 2 muestra la distribución de imágenes por tipo de entorno:

Entorno	Número de imágenes
Espacio (Space)	5808
Tierra (Land)	1599
Aire (Air)	1444

Tabla 2: Distribución de imágenes según el entorno de captura.

A continuación se muestran ejemplos representativos de cada entorno junto con sus respectivas máscaras.

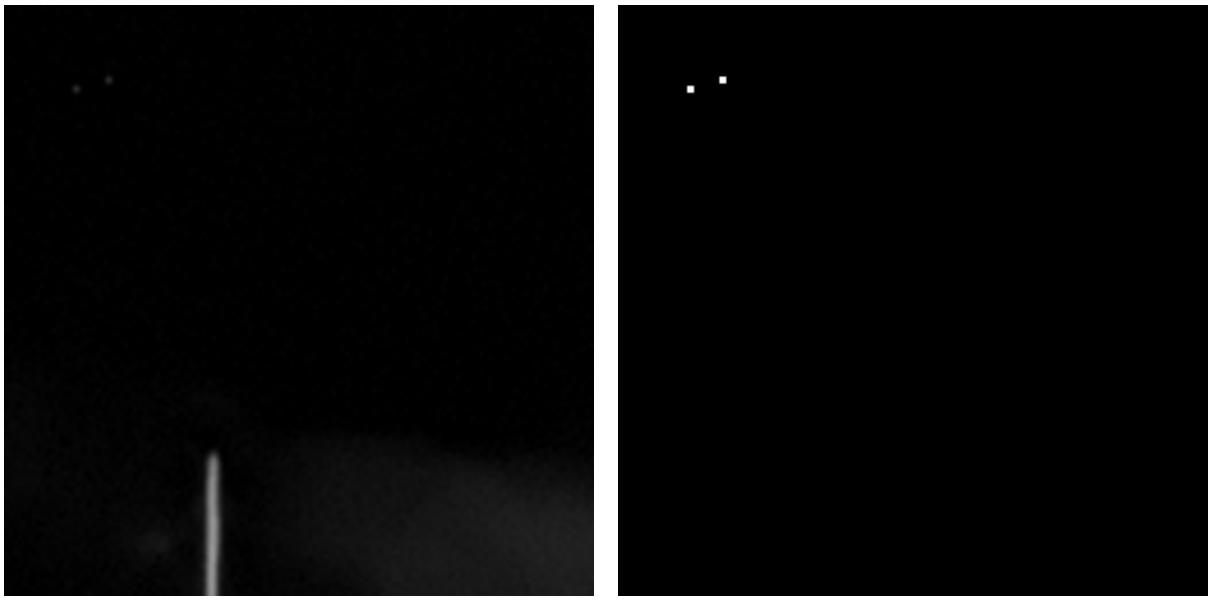
La Figura 9 muestra un ejemplo de imagen LAND y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 9: Categoría LAND.

La Figura 10 muestra un ejemplo de imagen AIR y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 10: Categoría AIR.

La Figura 11 muestra un ejemplo de imagen SPACE y su correspondiente *Ground Truth*:

(a) Imagen (b) *Ground Truth*

Figura 11: Categoría SPACE.

3.2. Tipo de objetivo

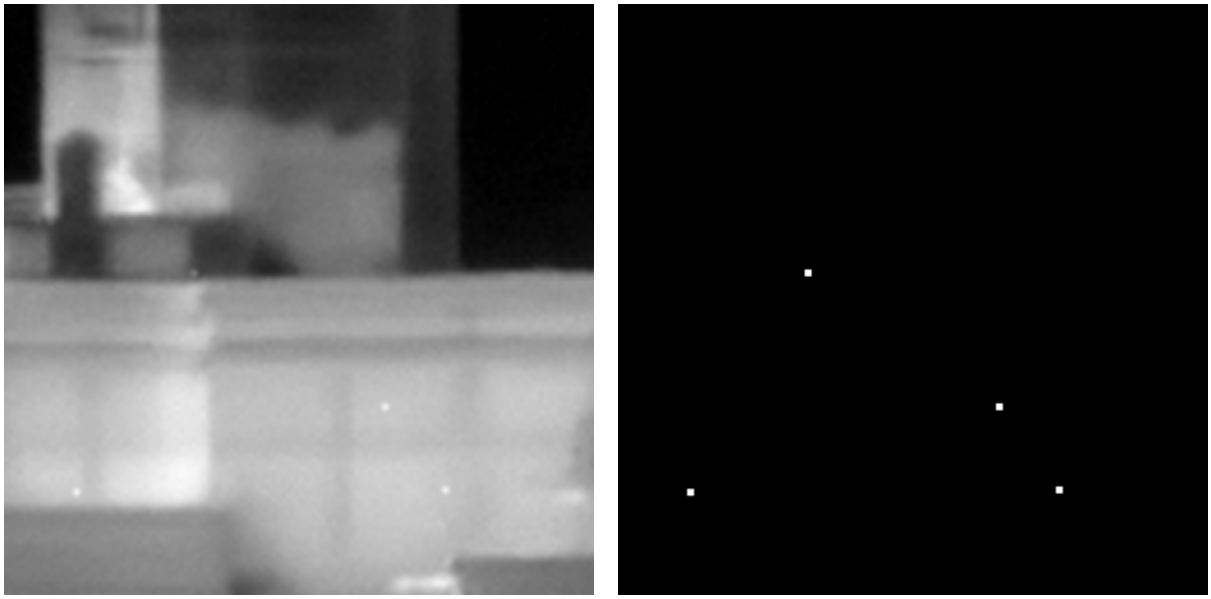
Según el tamaño y forma del objetivo en la imagen, se definen tres clases:

- **Point Target:** objetivo minúsculo, visible como uno o pocos píxeles brillantes. Suele confundirse con el ruido.
- **Spotted Target:** ligeramente más grande, compuesto por un grupo compacto de píxeles.
- **Extended Target:** ocupa una zona más amplia, con contornos más evidentes.

El dataset no incluye una etiqueta que especifique esta categoría por imagen, por lo que no es posible clasificarlas automáticamente según este criterio.

A continuación se muestran ejemplos de cada tipo de objetivo:

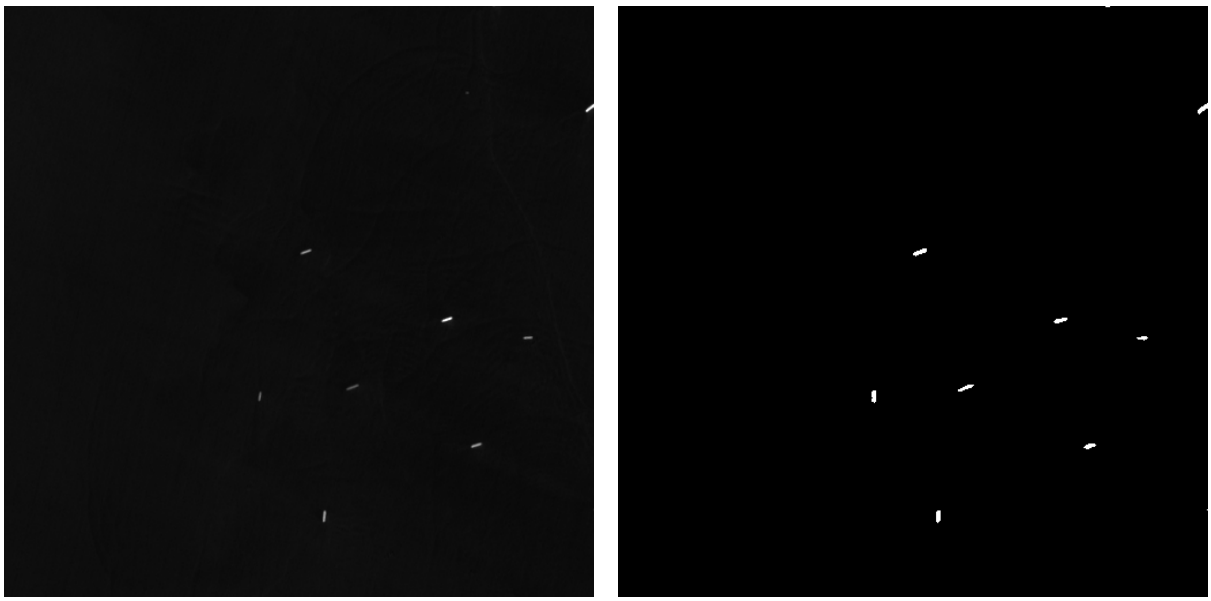
La Figura 12 muestra un ejemplo de imagen con Point Target y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 12: Categoría Point Target.

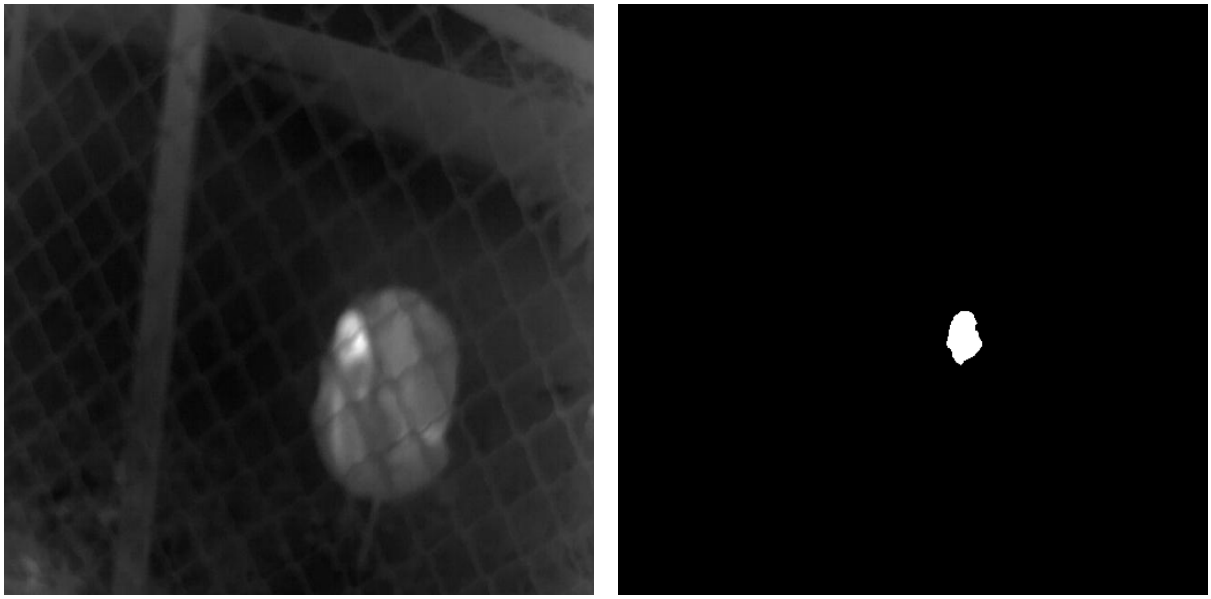
La Figura 13 muestra un ejemplo de imagen con Spotted Target y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 13: Categoría Spotted Target.

La Figura 14 muestra un ejemplo de imagen con Extended Target y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 14: Categoría Extended Target.

3.3. Espectro infrarrojo

El espectro infrarrojo abarca diversas longitudes de onda, las cuales afectan directamente la calidad y naturaleza de la información capturada por los sensores. La Figura 15 muestra la clasificación típica del espectro infrarrojo:

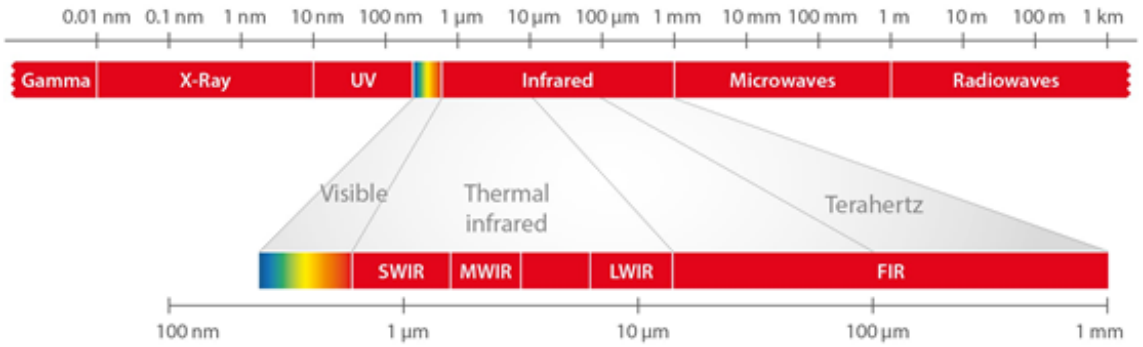


Figura 15: Clasificación del espectro infrarrojo [31].

Los tres rangos utilizados en el dataset son:

- **LWIR (*Long-Wave Infrared*):** 8–15 µm

- **SWIR (*Short-Wave Infrared*):** 0.9–1.7 µm
- **NIR (*Near Infrared*):** 0.75–1.4 µm

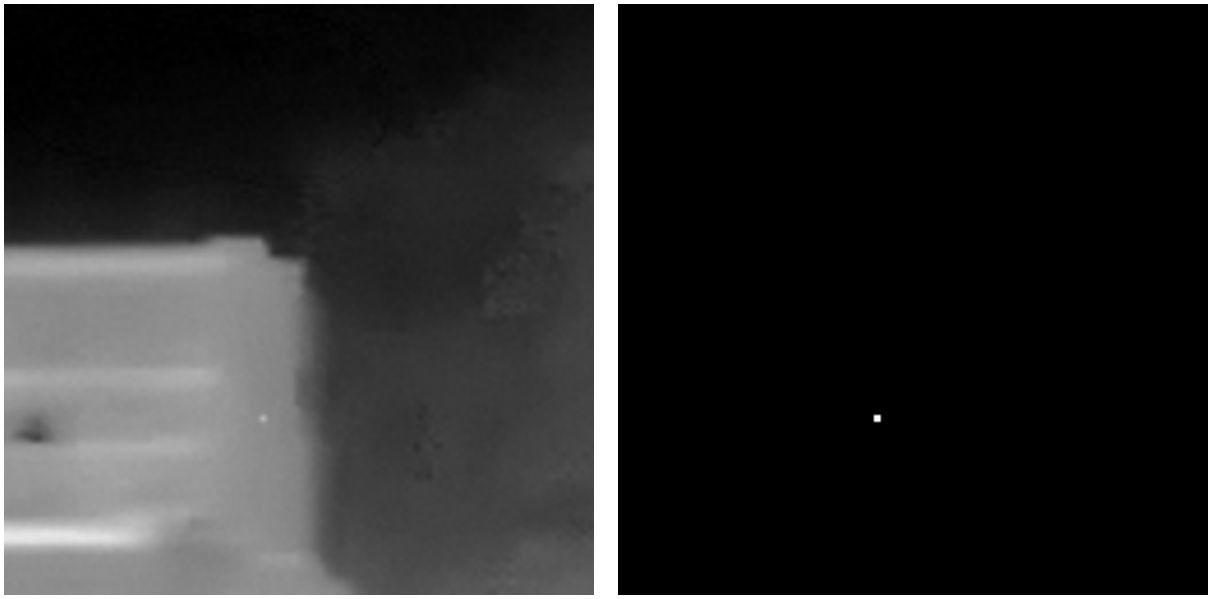
Cabe destacar que las imágenes tipo LWIR provienen exclusivamente de entornos **Air** y **Land**, mientras que las NIR proceden exclusivamente del entorno **Space**.

Tal como se muestra en la Tabla 3, la mayoría de las imágenes corresponden a los tipos LWIR y NIR, mientras que la cantidad de imágenes SWIR es muy reducida.

Tipo de espectro	Número de imágenes
LWIR	3043
SWIR	21
NIR	5787

Tabla 3: Distribución de imágenes por tipo de espectro.

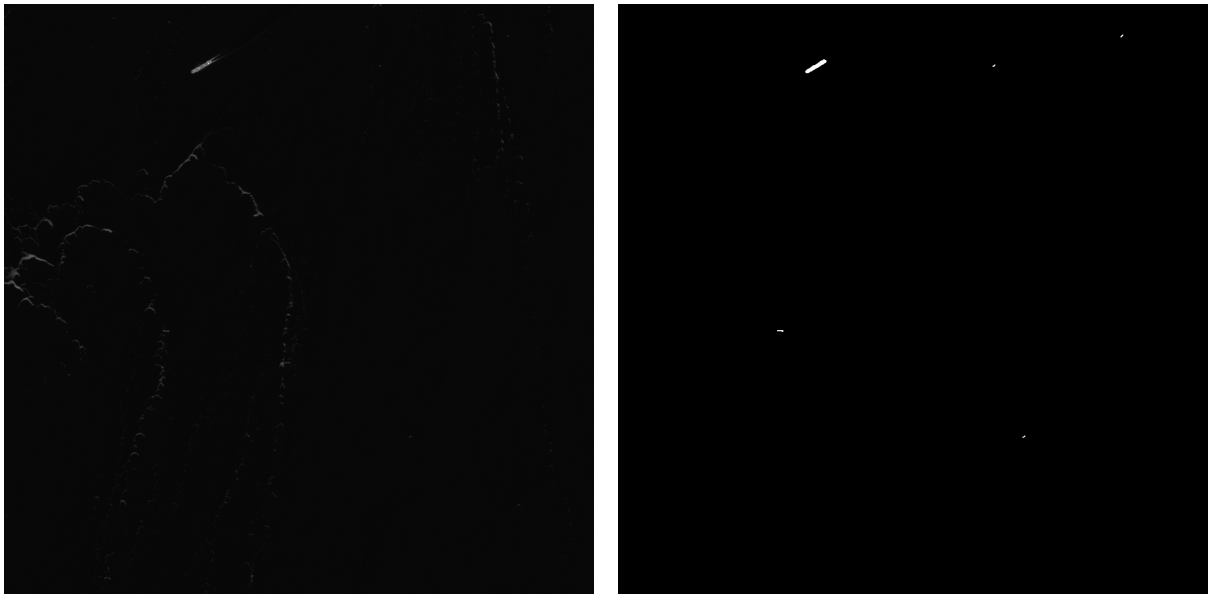
La Figura 16 muestra un ejemplo de imagen LWIR y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 16: Categoría LWIR.

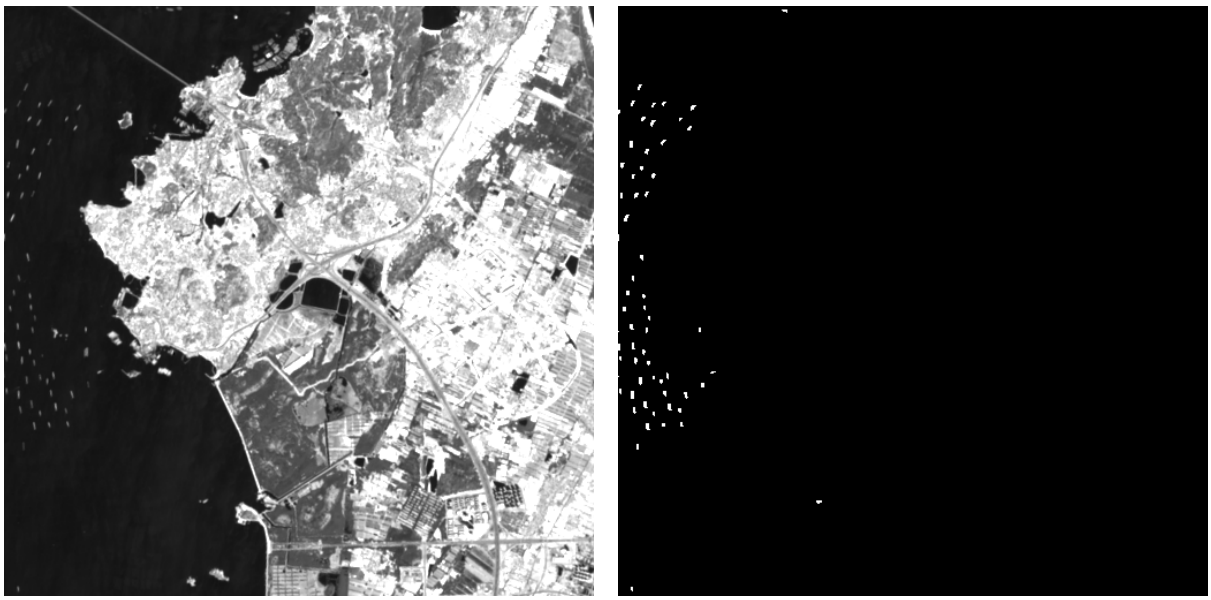
La Figura 17 muestra un ejemplo de imagen NIR y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 17: Categoría NIR.

La Figura 18 muestra un ejemplo de imagen SWIR y su correspondiente *Ground Truth*:



(a) Imagen (b) *Ground Truth*

Figura 18: Categoría SWIR.

4

Metodología de trabajo

En la realización de este trabajo se han seguido los siguientes principios metodológicos:

4.1. Método científico

Este trabajo se fundamenta en un enfoque metodológico basado en las etapas clásicas del método científico, adaptado al contexto experimental del desarrollo de modelos de detección. A lo largo del proyecto se siguió una estructura lógica que garantiza la replicabilidad de los experimentos y el análisis empírico de los resultados. Las etapas clave fueron:

1. **Identificación del problema:** Se definió como desafío principal la detección de objetivos pequeños en imágenes infrarrojas en condiciones adversas, con restricciones computacionales.

2. **Investigación del problema:** Se llevó a cabo una revisión del estado del arte en técnicas de visión por computador, redes neuronales convolucionales y arquitecturas como YOLO.

3. **Diseño experimental:** Se propusieron distintas configuraciones y técnicas (segmentación, aumento de datos, variantes de modelos) como hipótesis de mejora sobre el rendimiento.

4. **Validación:** Se realizaron experimentos controlados con validación cruzada para contrastar el comportamiento de las distintas estrategias, midiendo métricas como precisión, recall y mAP.

5. **Comunicación científica:** Los resultados obtenidos se documentaron, analizaron y discutieron, ofreciendo conclusiones, de forma estructurada y verificable.

4.2. Metodología ágil

Para este trabajo se ha adoptado un enfoque ágil que permite iterar e incrementar de manera progresiva fases de análisis del conjunto de datos y literatura científica, diseño de experimentos, implementación del código y evaluación de resultados, facilitando escalabilidad y reutilización. Este enfoque facilita la construcción de una base funcional inicial que se mejora paulatinamente, asegurando la estabilidad de los componentes previos e integrando cambios o requisitos nuevos con facilidad. Bajo esta premisa se diferencian distintas fases sobre las que trabajar.

- Modelo general: Inicialmente se compararán distintas sub-arquitecturas YOLOv8 que difieren en tamaño y prestaciones. Sobre esta comparativa inicial eligiremos el modelo más eficiente para mejorar en las futuras iteraciones.

- Modelo específico: Esta vez se comparará el rendimiento según el origen y espectro de la imagen para poder evaluar mejor que imágenes son más complejas, además posteriormente serán segmentadas con el objetivo de mejorar los resultados ya que generalmente es la solución más común ante el desafío de detección de objetivos pequeños.

- Análisis por resolución y tamaño de objetivo: Se investigará según la resolución de la imágen y la cantidad de píxeles objetivo el rendimiento de los modelos, haciendo la división de objetivos pequeños, medianos y grandes.

El código se estructuró de forma modular, y se documentaron los pasos necesarios para replicar la ejecución, lo que facilita su extensión a otros datasets o contextos.

4.3. Consideraciones

La detección de objetivos pequeños suele ser bastante problemática. Sirva como ejemplo las siguientes figuras donde se muestran comparativas de rendimiento de distintos métodos en otros artículos científicos. En particular, en la Figura 19, se observa el desempeño de YOLOv5 en contextos de vehículos autónomos.

Por otro lado, la Figura 20 muestra el rendimiento de una versión mejorada de YOLOv8s aplicada a imágenes capturadas por UAVs.

	mAP .5			mAP .5 small			inference (ms)		
Scales	YOLOv5	YOLO-Z	difference	YOLOv5	YOLO-Z	difference	YOLOv5	YOLO-Z	difference
S	0.926	**0.955**	3.13%	0.869	**0.925**	6.44%	**8**	8.9	0.9
M	0.932	**0.9605**	3.06%	0.8795	**0.9425**	7.16%	**11.6**	14.3	2.7
L	0.935	**0.964**	3.10%	0.886	**0.9545**	7.73%	**16.6**	19.6	3
X	0.9385	**0.9605**	2.34%	0.8975	**0.9465**	5.46%	**26.9**	30.6	3.7

Table 2: Comparing performance and inference time of YOLOv5 and YOLO-Z (optimal values for each scale in bold).

Figura 19: Comparativa de rendimiento en detección de objetivos pequeños usando YOLOv5 para vehículos autónomos [32].

Table 2. Detection performance (%) of different methods on Visdrone2019 validation dataset.

Models	mAP_{50}	mAP	Parameters (M)	FLOPs (G)
SSD [51]	23.9	13.1	24.5	87.9
RetinaNet [52]	27.3	15.5	19.8	93.7
ATSS [56]	31.7	18.6	10.3	57.0
Faster-RCNN [19]	33.2	17.0	41.2	207.1
GCGE-YOLO [55]	34.1	19.2	4.5	10.8
YOLOv5s [37]	35.4	20.5	9.1	23.8
Swin Transformer [29]	35.6	20.6	34.2	44.5
YOLOv8s [53]	36.4	21.6	11.1	28.5
C3TB-YOLOv5 [49]	38.3	22.0	8.0	19.7
TPH-YOLO [48]	39.3	23.6	51.5	138.1
DTSSNet [58]	39.9	24.2	10.1	50.4
YOLOv9 [54]	43.4	26.5	51.0	239.0
LV-YOLOv5 [57]	41.7	25.6	36.6	38.8
Ours	47.1	28.7	10.2	64.9

Figura 20: Rendimiento de una versión mejorada de YOLOv8s en escenarios con imágenes de UAV [33].

5
Análisis y Diseño

5.1. Análisis de Requisitos

En circunstancias normales la toma de requisitos se realizaría con el cliente (o Product Owner), pero adaptándonos al contexto de este proyecto, se pueden destacar ciertos requisitos que debe cumplir el sistema:

5.1.1. Requisitos Funcionales

Se detallan los requisitos funcionales que debe cumplir el sistema:

- **RF1.** Generar etiquetas (*bounding boxes*) a partir de máscaras binarias.
- **RF2.** Dividir automáticamente el conjunto de datos en pliegues para validación cruzada.
- **RF3.** Capacidad de entrenamiento de modelos YOLOv8 a partir de imágenes etiquetadas.
- **RF4.** El sistema debe devolver métricas como mAP, *Precision* y *Recall* para poder evaluar el rendimiento del modelo.
- **RF5.** El sistema debe poder filtrar y seleccionar imágenes según diferentes criterios.
- **RF6.** Preprocesamiento de imágenes y anotaciones por segmentación para mejorar la detección.

5.1.2. Requisitos No Funcionales

Se detallan los requisitos no funcionales que debe cumplir el sistema:

- **RNF1.** Eficiencia computacional: El sistema debe ser ejecutable en entornos con recursos limitados, por tanto los modelos pesaran menos de 100 MB.

- **RNF2.** Modularidad: Cada funcionalidad debe poder ejecutarse por separado.

- **RNF3.** Escalabilidad: El sistema debe poder adaptarse a conjuntos de datos más grandes o diferentes.

- **RNF4.** Reproducibilidad: Los experimentos deben poder replicarse con los mismos resultados, por lo tanto misma semilla para todos los entrenos.

- **RNF5.** Tiempos de entrenamiento razonables: Los modelos seleccionados deben equilibrar rendimiento y tiempo de ejecución, se evitarán entrenos de varios días.

5.1.3. Casos de uso

La siguiente Figura 21 muestra un diagrama de casos de uso con dos actores: Usuario y Modelo YOLO de Ultralytics. Se hace esta distinción ya que pese a que el entrenamiento esta contenido en el sistema local, no es una entidad intrínseca del sistema sino algo externo.

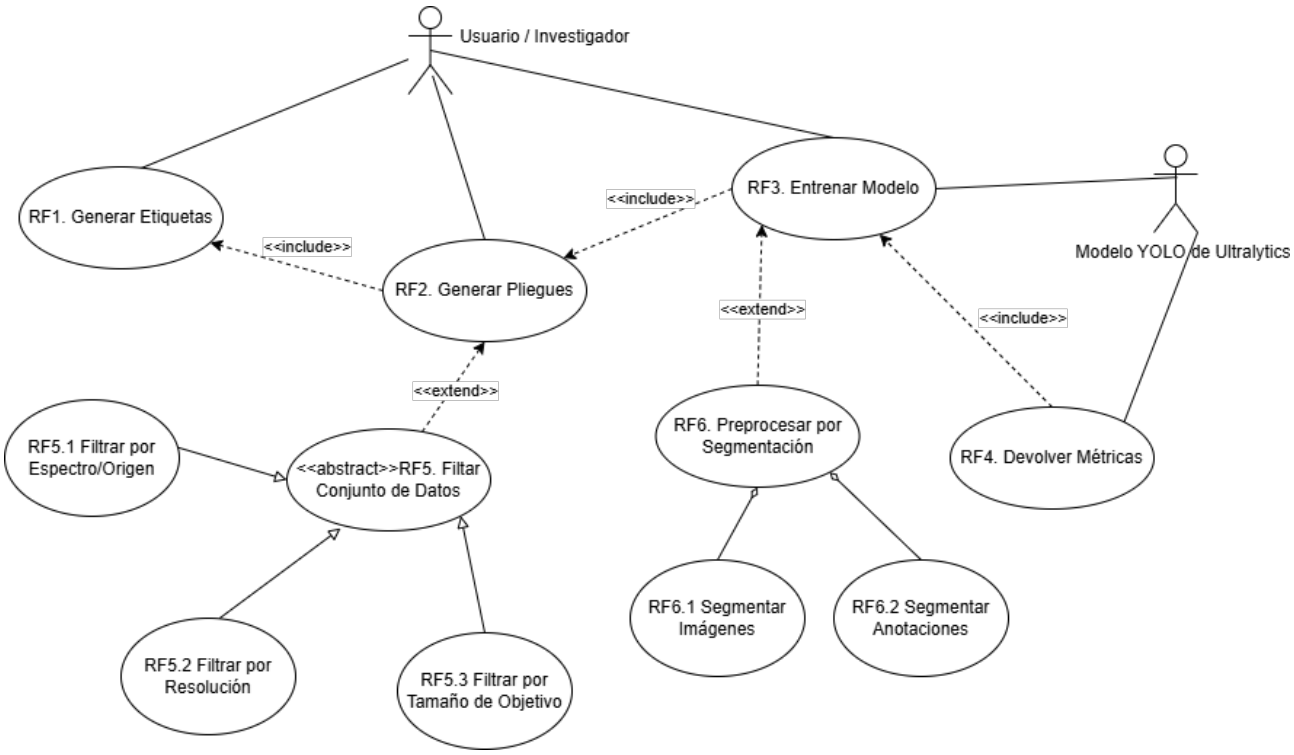


Figura 21: Diagrama casos de uso.

Las siguientes Tablas 4, 5, 6, 7, 8 y 9 muestran un análisis breve de cada caso de uso:

RF1.	Generar Etiquetas.
Descripción	El sistema permite al usuario generar etiquetas para los datos, necesarias para el entrenamiento del modelo.
Precondición	Suministrar carpeta con máscaras binarias y directorio de salida.
Escenario principal	▪ Ejecutar script. ▪ El sistema aplica el script de etiquetado y produce las etiquetas correspondientes.
Postcondición	Se generan en el directorio de salida los archivos en formato YOLO con la *ground truth*.

Tabla 4: Análisis caso de uso de RF1.

RF2.	Generar Pliegues.
Descripción	El sistema divide el conjunto de datos en pliegues para entrenamiento y validación.
Precondición	Deben existir ya las etiquetas («include» RF1). Suministrar número de pligues, carpeta con las imágenes y directorio de salida.
Escenario principal	▪ Ejecutar el script eligiendo el número de pliegues. ▪ El sistema genera los pliegues correspondientes.
Postcondición	Se generan subconjuntos de datos para cada etapa del ciclo de entrenamiento.

Tabla 5: Análisis caso de uso de RF2.

RF3.	Entrenar Modelo.
Descripción	El sistema entrena un modelo de detección usando los pliegues creados.
Precondición	Los pliegues ya están generados («include» RF2). Se disponen de recursos computacionales y existen archivos YAML pertinentes.
Escenario principal	▪ El usuario prepara un entorno con GPU disponible. ▪ Ejecuta el entrenamiento con hiperparámetros deseados. ▪ El sistema entrena el modelo.
Postcondición	Se guarda un modelo entrenado para futuras predicciones.

Tabla 6: Análisis caso de uso de RF3.

RF4.	Devolver Métricas.
Descripción	El sistema calcula y muestra métricas de rendimiento del modelo (por ejemplo, precisión, recall, F1).
Precondición	Debe haber existido un entrenamiento completo («include» RF3).
Escenario principal	▪ El sistema termina el entrenamiento. ▪ Automáticamente genera métricas de rendimiento en el directorio de salida.
Postcondición	Las métricas se almacenan para análisis posterior.

Tabla 7: Análisis caso de uso de RF4.

RF5.	Filtrar Conjunto de Datos.
Descripción	Permite aplicar filtros al conjunto de datos según criterios específicos.
Precondición	Deben existir las imágenes y el filtro a usar.
Escenario principal	▪ El usuario ejecuta el script de filtrado. ▪ Según el argumento proporcionado se preveen 3 escenarios: • **RF5.1 Filtrar por Espectro/Origen:** El sistema filtra imágenes por espectro (LWIR/NIR). • **RF5.2 Filtrar por Resolución:** El sistema filtra imágenes por resolución (baja, media, alta). • **RF5.3 Filtrar por Tamaño de Objetivo:** El sistema filtra por tamaño de objetivo en la imagen (pequeño, mediano, grande). ▪ El sistema devuelve un .txt con la lista de imágenes filtradas.
Postcondición	Archivo .txt con la lista de imágenes filtradas.

Tabla 8: Análisis caso de uso de RF5.

RF6.	Preprocesar por Segmentación.
Descripción	Realiza un preprocesamiento específico para segmentación en mosaicos de imágenes y sus etiquetas.
Precondición	Suministrar directorio pliegues, directorio de salida y tamaño de la porción y overlap.
Escenario principal	▪ Ejecutar script eligiendo tamaño de porción y overlap. ▪ **RF6.1 Segmentar Imágenes:** El sistema realiza recortes de imágenes, de dimensiones seleccionadas. ▪ **RF6.2 Segmentar Anotaciones:** El sistema transforma las etiquetas de formato YOLO a COCO, realiza los cortes y vuelve a transformarlas a formato YOLO. ▪ En el directorio de salida se generan los pliegues segmentados.
Postcondición	Los pliegues quedan segmentados.

Tabla 9: Análisis caso de uso de RF6.

5.2. Diseño

Una arquitectura en *pipeline* (tubería) en informática consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. En el contexto de Ingeniería de Datos y Aprendizaje Automático un *pipeline* [34] se puede considerar como una serie ordenada de pasos que van desde la entrada (datos) hasta la salida (modelo entrenado, métricas...) sin importar si es automático o no. Puede incluir tanto scripts automatizados como pasos manuales.

La siguiente Figura 22 muestra un ejemplo de arquitectura en *pipeline* en Aprendizaje Automático:

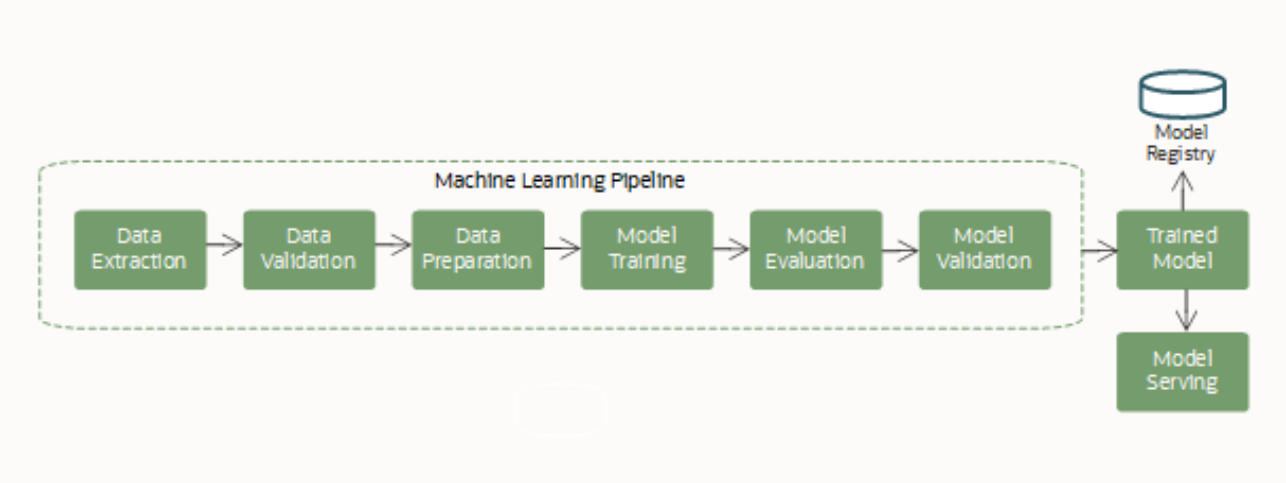


Figura 22: Ejemplo de *pipeline* en Aprendizaje Automático [34].

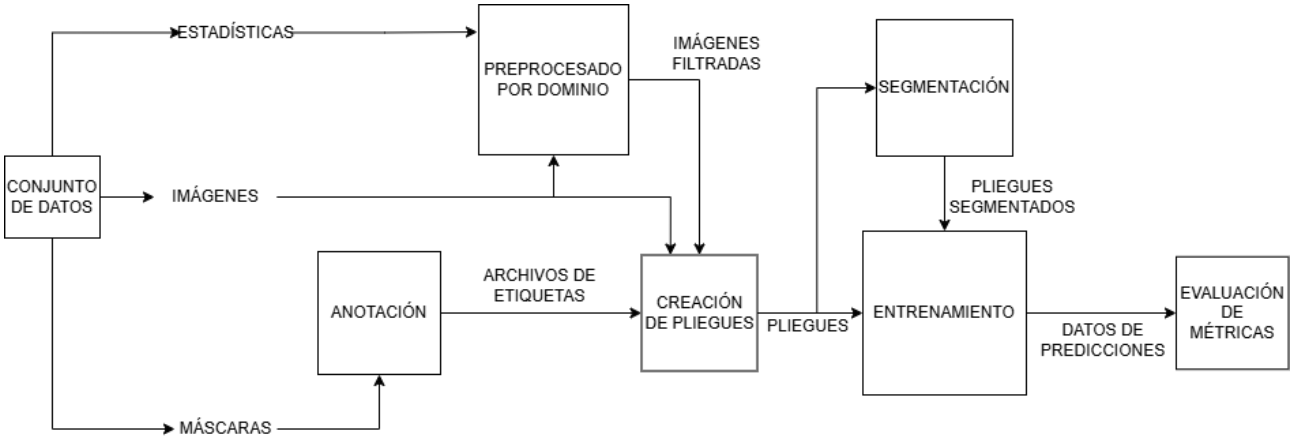


Figura 23: Resumen del *pipeline* mediante una infografía.

La Figura 23 superior muestra un resumen del conjunto de procesos desarrollados en este trabajo.

Se pueden destacar los siguientes elementos:

- ***Dataset*:** El conjunto de datos proporciona los siguientes archivos:
  - ***Images*:** Contenido principal para el entrenamiento.
  - ***Statistics*:** Características de las imágenes.
  - ***ground truth*:** Máscara de cada imagen.
- ***Bounding Box Files*:** Anotaciones en formato YOLO para poder comparar las predicciones del modelo con la verdad fundamental. Se generan con la *ground truth*.

- ***Filtered Images*:** En ciertas configuraciones se filtraran las imágenes para poder hacer entrenamientos específicos. Aparte de las imágenes es necesario también el archivo `statistics.txt`.

- ***Folds*:** Los pliegues para poder realizar la validación cruzada. Se generan con las imágenes con o sin filtrar y con las anotaciones generadas.

- ***Sliced Folds*:** En una configuración se segmentarán las imágenes. La segmentación se hace directamente desde un pliegue, ya que las anotaciones deben ser segmentadas también.

- ***Prediction Data*:** Las predicciones del modelo serán revisadas y se generarán gráficos y tablas para poder evaluar las métricas.

Si bien los procesos de: *Labeling, Domain Preprocessing, Cross-Validation, Tiling* y *Training* son automáticos, el arranque de cada uno de ellos es manual.

5.2.1. Diagrama de arquitectura

La siguiente Figura 24 muestra un diagrama de la arquitectura para representar visualmente los distintos componentes del sistema y sus relaciones e interacciones:

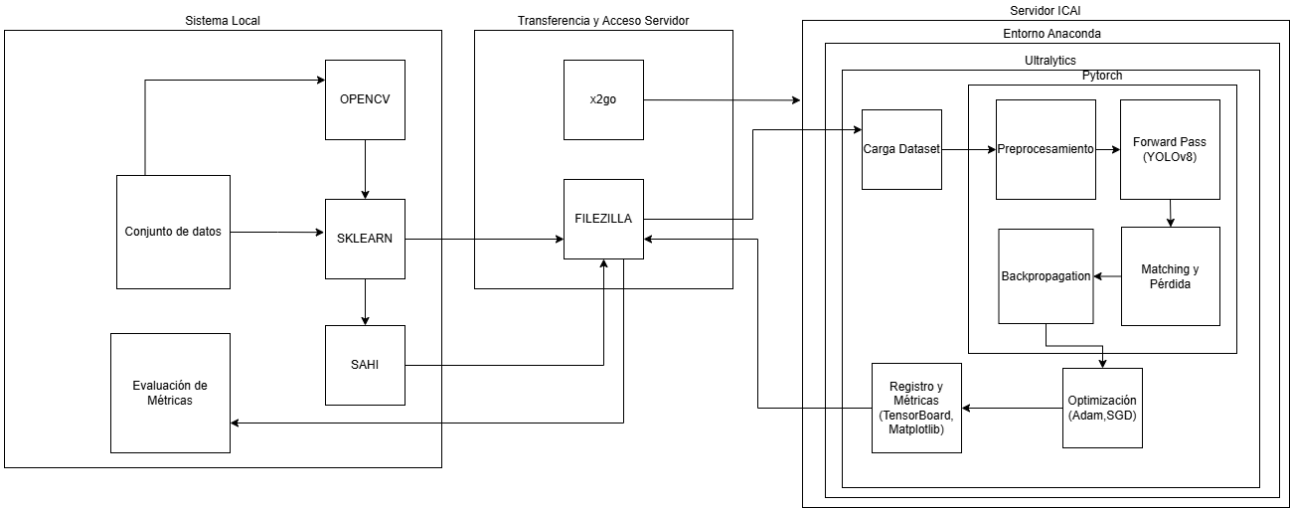


Figura 24: Diagrama de arquitectura del sistema.

La gran parte de tecnologías ya han sido descritas anteriormente pero se pueden destacar como se integran en la arquitectura:

- **OpenCV**: Creación de anotaciones formato YOLO.

- **Sklearn**: Con las anotaciones y las imágenes se crean los pliegues.
- **SAHI**: Segmentación de los pliegues.
- **FileZilla**: Transferir pliegues a servidor.
- **x2go**: Acceder a servidor para manipulación y ejecución de archivos.
- **Anaconda**: Entorno para gestionar los paquetes en el servidor.
- **Ultralytics**: Núcleo del entrenamiento.
- **DataLoader**: Utilidad en PyYAML para cargar datos en batch y con paralelismo durante el entrenamiento.
- **PyTorch**: *Framework* de *Deep Learning* que incluye:
  - *Preprocesamiento*: Re-dimensión, normalización y transformación de datos a tensores para el modelo.
  - *Forward pass YOLOv8*: Cálculo de predicciones a partir de las entradas.
  - *Matching y pérdida*: Asociación de predicciones con etiquetas y cálculo del error.
  - *Backpropagation*: Cálculo de gradientes [35].
- **Optimización Adam[36], SGD[37]**: Ajuste de pesos.
- **TensorBoard y Matplotlib**: Se registran las métricas para análisis visual.
- **Evaluación métricas** : Finalmente se transfieren los resultados a la máquina local para poder evaluarlos.

6
Configuración

6.1. Bounding Boxes

Un *bounding box* [38] (cuadro delimitador) es un rectángulo que se utiliza en tareas de visión por computadora para delimitar o encerrar un objeto en una imagen. Es una de las formas más comunes de representar y localizar objetos en aplicaciones como detección de objetos, segmentación y seguimiento.

Los archivos de *bounding box* en los *labels* (etiquetas) de YOLO son necesarios para comparar las predicciones del modelo con las anotaciones reales durante el proceso de entrenamiento y evaluación. Esto es crucial para medir qué tan bien ha aprendido el modelo a identificar y localizar objetos.

La Figura 25 muestra un ejemplo de imagen con cuadros delimitadores definidos para coches y motocicletas.

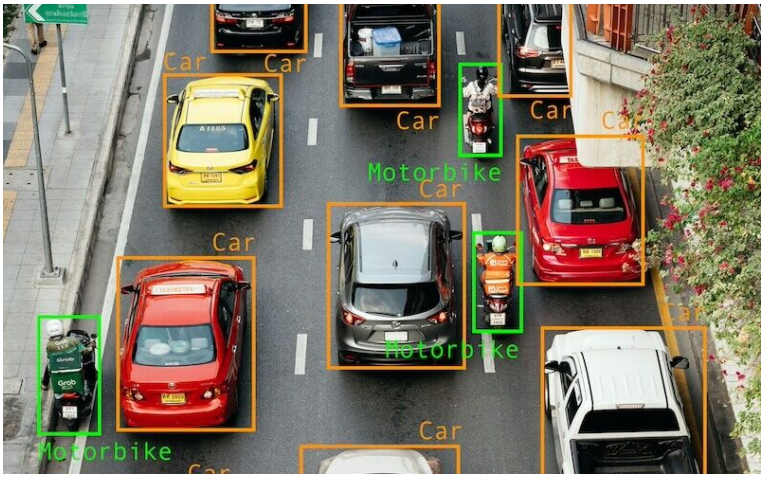


Figura 25: Imagen con *bounding boxes* definidos para coches y motocicletas [38].

Es preciso crear un archivo de texto para cada imagen con la información correspondiente a la ubicación de los objetivos a detectar. A continuación se muestra un ejemplo de su contenido en formato YOLO:

```
0 0.232421875 0.787109375 0.01171875 0.01953125
0 0.439453125 0.501953125 0.01171875 0.01953125
```

Cada línea indica un nuevo objetivo:

- Primera cifra: Es el identificador de clase. Ya que solo identificamos un único tipo de objeto, solo existirá un label.
- Segunda y tercera cifra: Coordenadas del centro del *bounding box*, normalizadas respecto al ancho y alto de la imagen.
- Cuarta y quinta cifra: Ancho y alto del *bounding box*, también normalizados.

Para crear estos archivos, se debe iterar sobre el contorno de cada imagen máscara, en la cual el contraste de píxeles representa las áreas de interés. En la Figura 26 se muestra un ejemplo de imagen de máscara binaria, estas se pueden crear de manera manual, automática o ambas (automática con corrección manual).

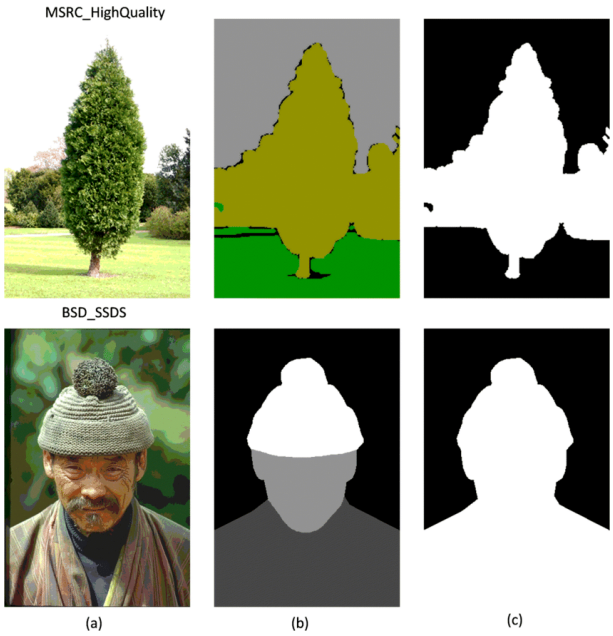


Figura 26: Dos máscaras binarias de referencia generadas manualmente. A la izquierda, la imagen original. En el centro, segmentación semántica. A la derecha, dos máscaras binarias. [39]

En el apéndice B.1 se puede encontrar más información sobre el proceso de creación de las *bounding boxes*.

6.2. Validación Cruzada

La validación cruzada [40] es una técnica en el Aprendizaje Automático utilizada para evaluar la capacidad de generalización de un modelo, es decir, qué tan bien se desempeñará con datos nuevos no vistos durante el entrenamiento. La idea principal es dividir los datos disponibles en partes o subconjuntos para asegurarse de que el modelo no esté sobreajustando los datos de entrenamiento y funcione bien en datos reales.

El proceso general se resume en los siguientes pasos:

- **División de los datos:** El conjunto de datos se divide en varias partes o pliegues (folds) de igual tamaño.
- **Entrenamiento y validación:** En cada iteración, se entrena el modelo utilizando todos los pliegues excepto uno. El pliegue restante se usa para validar el modelo.
- **Repetición:** Este proceso se repite tantas veces como pliegues se hayan creado, cambiando el pliegue de validación en cada iteración.
- **Promediar resultados:** Finalmente, se calculan los resultados promedio de todas las iteraciones para obtener una métrica de desempeño más confiable.

La Figura 27 ilustra este proceso con un ejemplo visual.

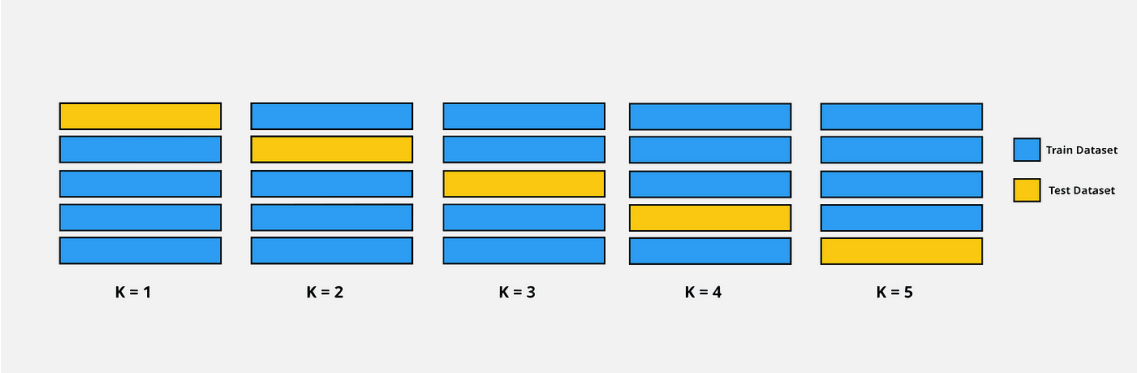


Figura 27: Ejemplo de validación cruzada de 5 pliegues. [41]

En esta ocasión se ha usado validación cruzada con 5 pliegues, ya que ofrece un buen equilibrio entre precisión y tiempo de ejecución. Para cada iteración, el 80 % de los datos se usan para entrenamiento y el 20 % para test. Además, se incluye la información de los *labels* de *bounding box* creados previamente.

En el apéndice B.2 se puede encontrar más información sobre el proceso de creación de pliegues.

6.3. Entrenamiento

Antes de proceder con el entrenamiento, es necesario crear para cada pliegue un archivo con la información de rutas donde se encuentran las imágenes y clases a detectar. Este archivo YAML tiene la siguiente estructura:

- **path:** `../kfold_data/fold_1`
  Ruta raíz donde se encuentran los conjuntos de datos (entrenamiento y validación). El resto de rutas se interpretan relativas a esta.
- **train:** `train`
  Carpeta dentro de `path` que contiene las imágenes y etiquetas de entrenamiento, en las subcarpetas `images/` y `labels/`.
- **val:** `test`
  Carpeta dentro de `path` utilizada para la validación.
- **nc: 1**
  Número de clases a detectar. En este caso, solo hay un tipo de objeto.
- **names:** `["target"]`
  Lista de nombres de clases, donde el índice `0` representa la clase `"target"`.

7

Modelo General

7.1. Entrenamiento

Este primer entrenamiento servirá de referencia para futuras mejoras y para comparar el rendimiento entre los métodos `nano`, `small`, `medium` y `large`. El método `extra-large` queda fuera tanto por necesidades como por limitaciones de recursos.

En la Tabla 10 se resumen los tiempos de entrenamiento aproximados por método para 100 épocas.

Modelo	Tiempo por época	Tiempo por pliegue (100 épocas)	Tiempo total (5 pliegues)
n	50 segundos	1 hora 23 minutos	6 horas 56 minutos
s	1 minuto	1 hora 40 minutos	8 horas 20 minutos
m	1 minuto 30 segundos	2 horas 30 minutos	12 horas 30 minutos
l	2 minutos 40 segundos	4 horas 26 minutos	22 horas 13 minutos

Tabla 10: Tiempos de entrenamiento por modelo expresados en horas, minutos y segundos.

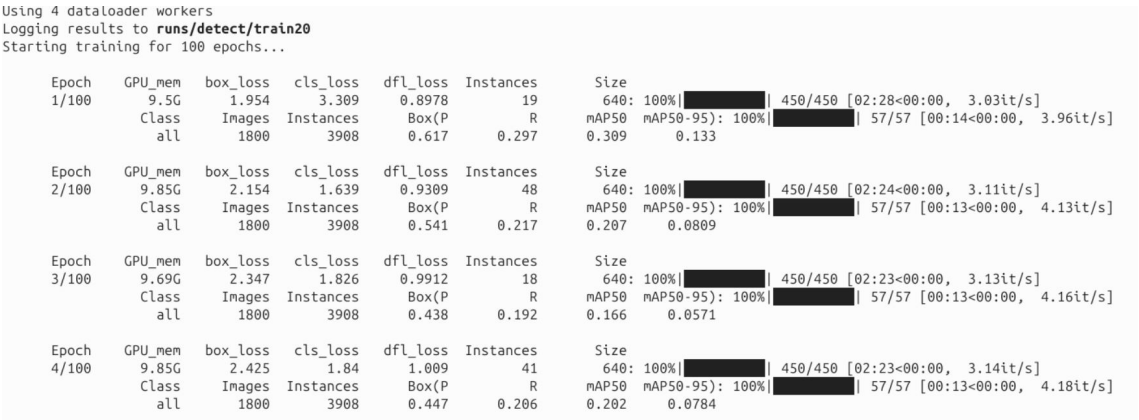


Figura 28: Información mostrada en la terminal durante el entrenamiento con Ultralytics.

En la Figura 28 se puede observar un ejemplo de la salida mostrada en la terminal durante el proceso de entrenamiento.

7.2. Resultados

Por defecto, Ultralytics realiza una validación usando el conjunto de test después del entrenamiento de cada época. Por tanto, no es necesario ejecutar pasos adicionales para obtener métricas.

Las métricas elegidas para valorar el rendimiento en validación son la Precisión, el Recall y el mAP@50. La siguiente Tabla (11) muestra los resultados obtenidos en cada uno de los cinco pliegues usados para la validación cruzada:

Métrica	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**
N-mAP	0.44055	0.43182	0.41859	0.40061	0.4579
N-Precision	0.75528	0.75225	0.73374	0.69089	0.74269
N-Recall	0.38383	0.36946	0.35322	0.34844	0.38951
S-mAP	0.46453	0.44241	0.42778	0.40679	0.46395
S-Precision	0.74693	0.75583	0.71242	0.71704	0.71477
S-Recall	0.40027	0.37088	0.36216	0.33867	0.39524
M-mAP	0.47154	0.44784	0.43115	0.40523	0.48039
M-Precision	0.7508	0.76782	0.71956	0.68462	0.74296
M-Recall	0.4002	0.36969	0.35275	0.35114	0.4089
L-mAP	0.45503	0.44339	0.44635	0.42896	0.46431
L-Precision	0.74145	0.75087	0.72404	0.71802	0.74106
L-Recall	0.38332	0.36946	0.37114	0.36424	0.38555

Tabla 11: Resultados por pliegue.

Métrica	**N**	**S**	**M**	**L**
Precision	0.7350 ± 0.0233	0.7294 ± 0.0182	0.7332 ± 0.0288	0.7351 ± 0.0122
Recall	0.3689 ± 0.0162	0.3734 ± 0.0225	0.3765 ± 0.0239	0.3747 ± 0.0083
mAP@50	0.4299 ± 0.0194	0.4411 ± 0.0220	0.4472 ± 0.0273	0.4476 ± 0.0118

Tabla 12: Resultados promedio ($\mu \pm \sigma$) por subarquitectura.

Para facilitar la comparación y mejorar la visibilidad, la Tabla 15 resume los promedios por subarquitectura.

A partir de los datos de la Tabla 15, se puede concluir que los modelos muestran un comportamiento conservador: aunque las predicciones alcanzan una precisión razonable, se observa un recall bajo, lo que indica que muchos objetivos no son detectados.

Además, no se aprecia una mejora significativa al emplear modelos de mayor tamaño, a pesar del incremento en el coste computacional asociado a estos. Esto sugiere que, en este contexto específico, utilizar arquitecturas más complejas no aporta beneficios proporcionales en términos de rendimiento.

Por otro lado, la salida de cada pliegue del entrenamiento incluye una curva de evaluación F1 que permite analizar cómo varía esta métrica en función del umbral de confianza. La Figura 29 muestra un ejemplo representativo de este tipo de gráfica.

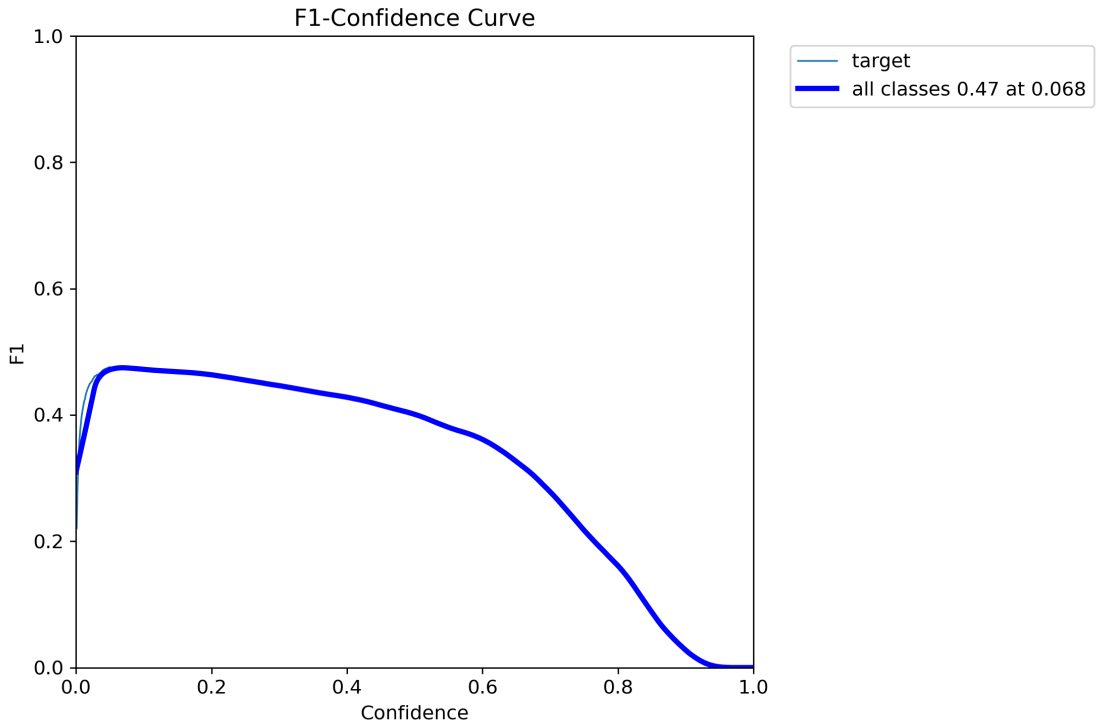


Figura 29: Curva F1 para Medium-fold3.

Como se aprecia en la Figura 29, aunque YOLOv8 emplea por defecto un umbral de confianza de 0.25, la puntuación F1 máxima se alcanza reduciendo dicho umbral. Esto implica favorecer un mayor recall a costa de una ligera disminución en la precisión. Aunque este ajuste no mejora directamente los resultados durante la fase de entrenamiento, puede ser útil en el momento de realizar predicciones en condiciones reales.

En principio, los modelos más grandes deberían tardar más en converger y beneficiarse de un mayor número de épocas. Sin embargo, al comparar la evolución del entrenamiento entre modelos pequeños y grandes, no se aprecian diferencias particularmente destacadas. Para ilustrar este comportamiento, se presentan a continuación dos figuras que muestran la evolución de las métricas durante el entrenamiento para un modelo small y otro large.

Como puede observarse en las Figuras 30 y 31, el comportamiento de las métricas es similar, lo cual refuerza la idea de que el aumento en la complejidad del modelo no se traduce en una mejora proporcional en el aprendizaje dentro del número de épocas utilizado.

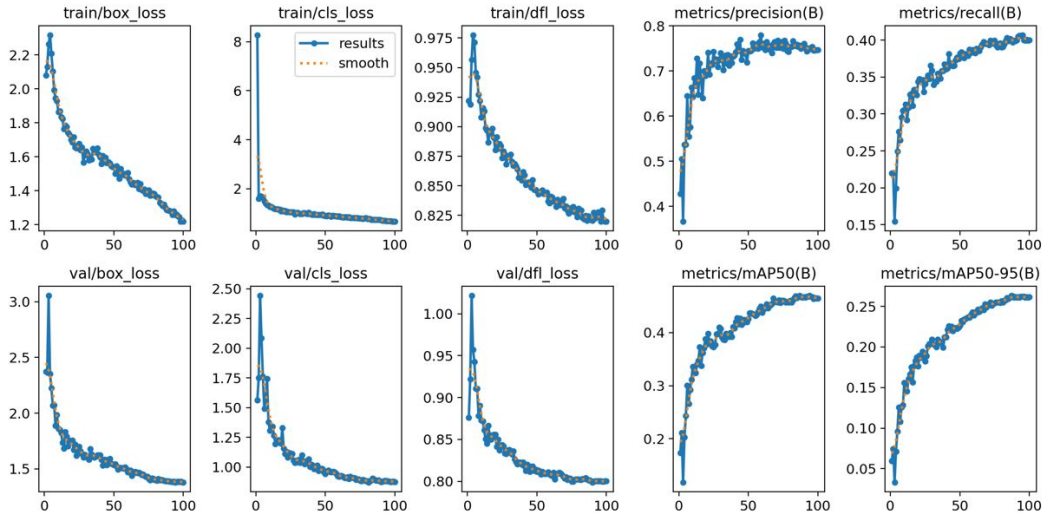


Figura 30: Evolución de las métricas durante el entrenamiento para small-fold1.

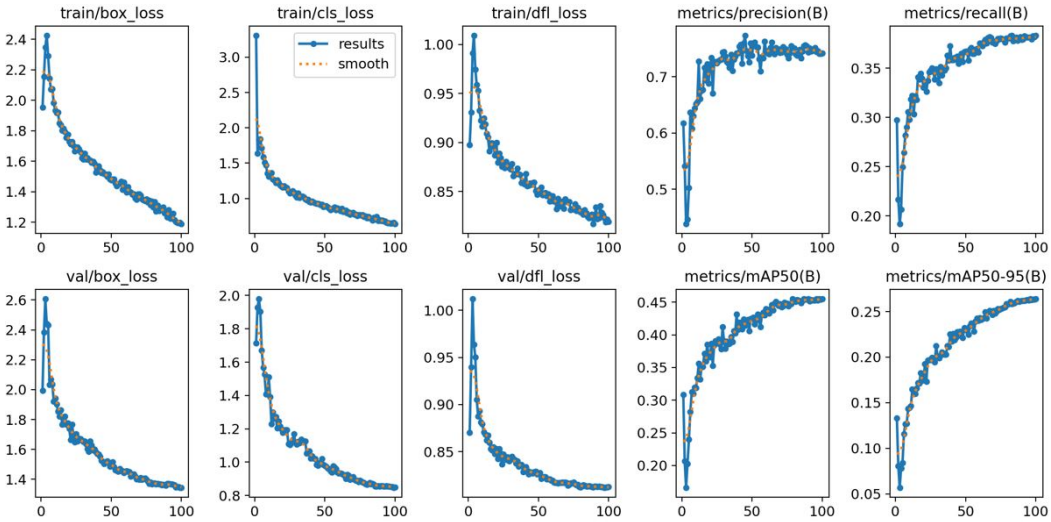


Figura 31: Evolución de las métricas durante el entrenamiento para large-fold1.

8

Modelo Específico

8.1. División por espectro y origen

Para poder discernir mejor qué tipo de imágenes funcionan mejor y para que el modelo pueda aprender de una manera más específica, se pretende dividir las imágenes en distintos grupos para hacer entrenamientos específicos.
Debido a que las imágenes tipo NIR son exclusivamente de tipo Space, mientras que las de tipo LWIR pertenecen a los tipos Air y Land, y dado que no hay suficientes imágenes tipo SWIR para poder realizar un entrenamiento óptimo, se ha realizado la siguiente escisión en dos dominios diferentes, como se muestra en la Tabla 13.

Dominio	**Características (**$Ancho \times Alto$**)**
LWIR - Air/Land	3043 Imágenes de entre 250×256 y 512×512 píxeles
NIR - Space	5787 Imágenes principalmente de 1024×1024 píxeles

Tabla 13: División del conjunto de datos por espectro y origen.

La división en NIR y LWIR ha sido posible por el contenido del archivo `statistics.txt` .

8.2. Resultados

El entrenamiento fue realizado con el subgrupo YOLOv8n, ya que las arquitecturas superiores no demostraron mayor rendimiento. Esto último se alinea con la literatura científica actual [32, 33], donde no se observa una correlación lo suficientemente grande entre el tamaño del modelo y el rendimiento como para justificar el uso de más recursos computacionales.

En problemas como la detección de objetivos pequeños, el uso de modelos excesivamente grandes puede resultar contraproducente. Según el bias-variance tradeoff [42], al aumentar la capacidad del modelo (como ocurre con arquitecturas profundas o complejas), se reduce el sesgo, pero a costa de incrementar la varianza, cayendo en sobreajuste. En estos casos, el

modelo detecta patrones irrelevantes o ruido, lo cual es especialmente perjudicial cuando los objetivos a identificar son pequeños y escasos. Por ello, no siempre un modelo más grande implica un mejor rendimiento; en muchos contextos.

Los resultados del entrenamiento con YOLOv8n se detallan en la Tabla 14, donde se incluyen las métricas obtenidas por pliegue durante la validación cruzada.

Métrica	**Pliegue1**	**Pliegue2**	**Pliegue3**	**Pliegue4**	**Pliegue5**
LWIR-mAP	0,87462	0,87880	0,87650	0,87806	0,85613
LWIR-Precision	0,8592	0,89721	0,87497	0,88575	0,84339
LWIR-Recall	0,83098	0,78671	0,82527	0,81896	0,81671
NIR-mAP	0,33707	0,33915	0,30223	0,33980	0,30242
NIR-Precision	0,66565	0,64020	0,61135	0,62732	0,61801
NIR-Recall	0,27904	0,28234	0,25188	0,27999	0,25873

Tabla 14: Resultados por pliegue para los dominios LWIR y NIR.

El tiempo de ejecución por pliegue en LWIR es de 25 minutos y en NIR de 1 hora y 5 minutos.

Para facilitar la comparación entre dominios y obtener una visión más clara, la Tabla 15 resume los resultados promedios para cada división.

Métrica	**LWIR**	**NIR**
Precision	$0{,}87250 \pm 0{,}02029$	$0{,}63251 \pm 0{,}02000$
Recall	$0{,}81533 \pm 0{,}01618$	$0{,}27040 \pm 0{,}01273$
mAP@50	$0{,}87282 \pm 0{,}00849$	$0{,}32453 \pm 0{,}02058$

Tabla 15: Resultados promedio ($\mu \pm \sigma$) para las divisiones LWIR y NIR.

El rendimiento en LWIR ha mejorado de forma notable, mientras que en NIR ha experimentado una disminución, especialmente en el valor de Recall, lo que sugiere que el sistema presenta más dificultades para localizar objetos que para evitar falsas detecciones.

Tras una inspección manual, se pueden destacar los siguientes problemas:

- **Resolución final usada en el entrenamiento:** Las imágenes NIR tienen una mayor resolución que las LWIR por lo tanto al entrenar en un imgsz de 640 las imágenes en

NIR se podrían estrechar, perdiendo información. Una mayor proporción entre las dimensiones del objetivo frente al tamaño de la imagen sugiere un mayor rendimiento.

- **Origen y calidad de los datos:** Las imágenes NIR fueron tomadas desde el espacio, lo que introduce posibles variaciones atmosféricas, mayor ruido, y cambios de escala más drásticos, así como mucha más información por imagen.

- **Cantidad y distribución de objetos en las imágenes:** Las imágenes NIR presentaban una gran variabilidad en el número de objetos, incluyendo muchas sin ningún objeto o con múltiples objetivos, lo cual dificulta la tarea de detección. En contraste, las imágenes LWIR contienen consistentemente uno o dos objetos, lo que facilita la generalización del modelo.

Las Figuras 32 y 33 inferiores demuestran que el umbral de confianza para maximizar recall y precisión difiere, tendiendo a un modelo menos conservador en NIR y uno algo más conservador en LWIR:

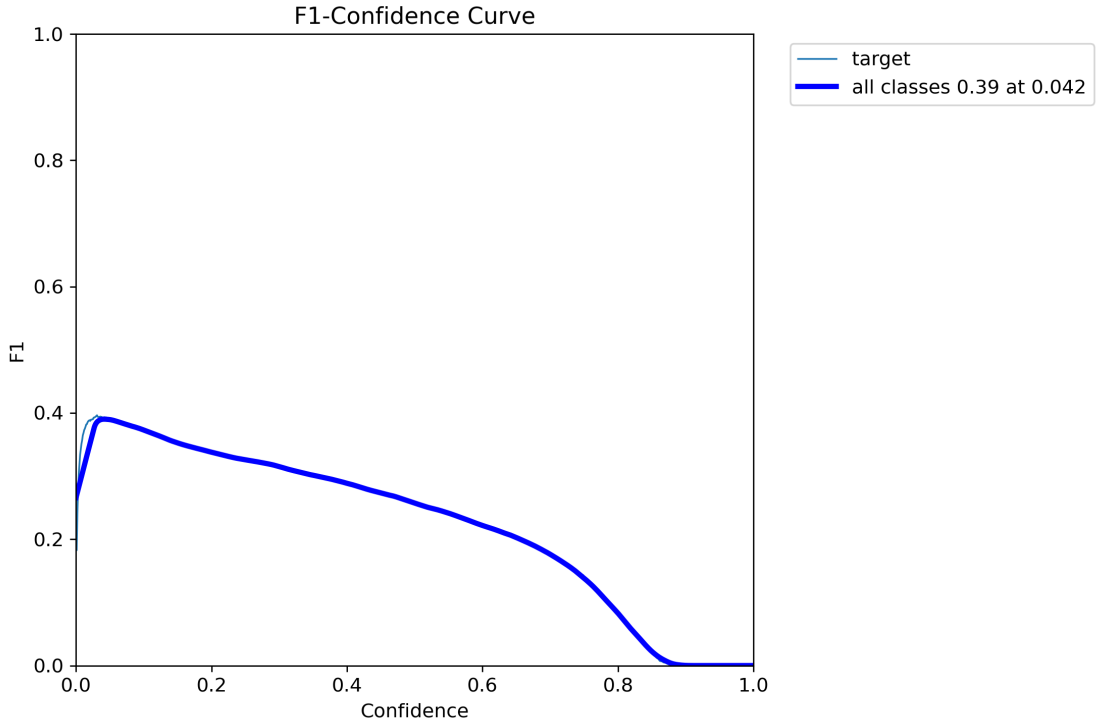


Figura 32: Curva F1 NIR-Fold1.

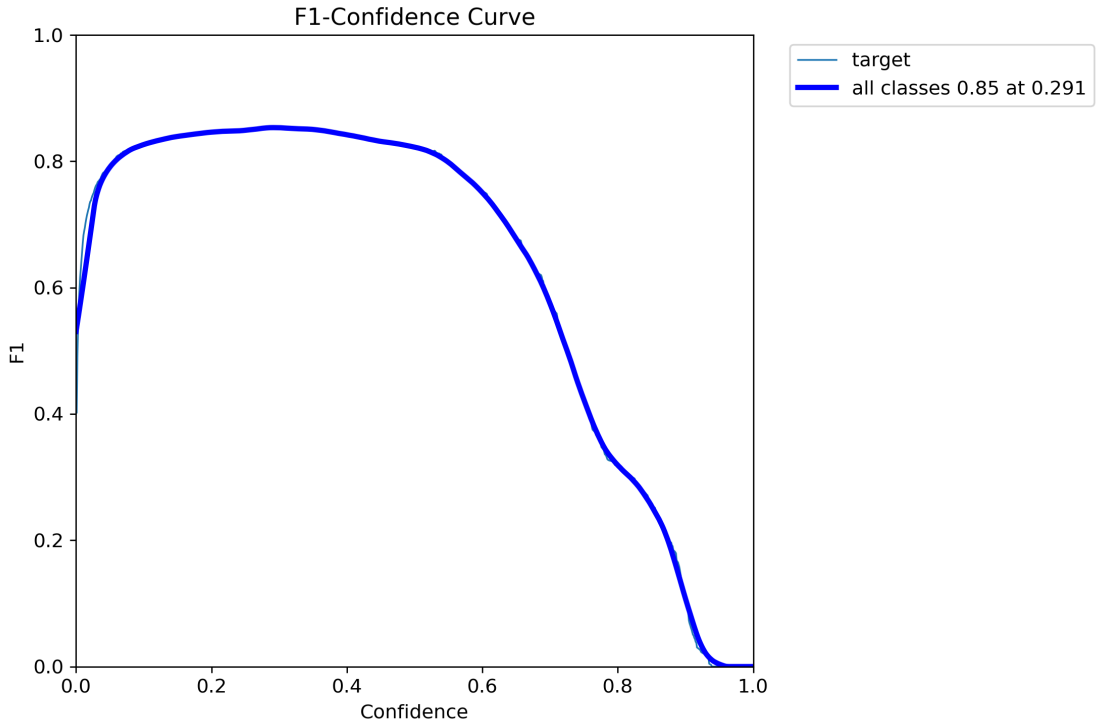


Figura 33: Curva F1 LWIR-Fold1.

Las Figuras 34 y 35 inferiores demuestran que no existen diferencias en la convergencia y que 100 épocas siguen siendo suficientes:

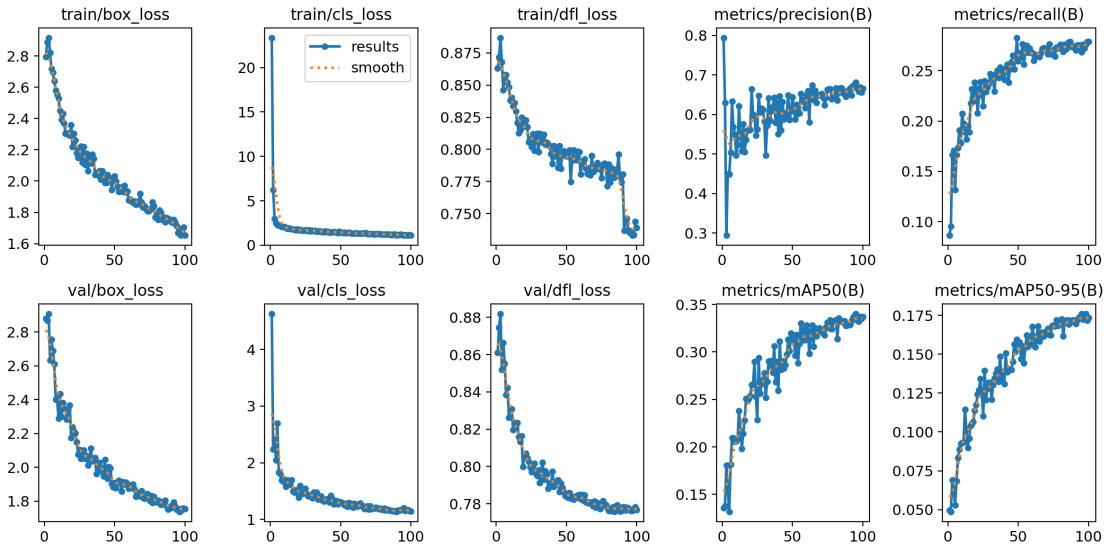


Figura 34: Gráficas resultados NWIR-Fold1.

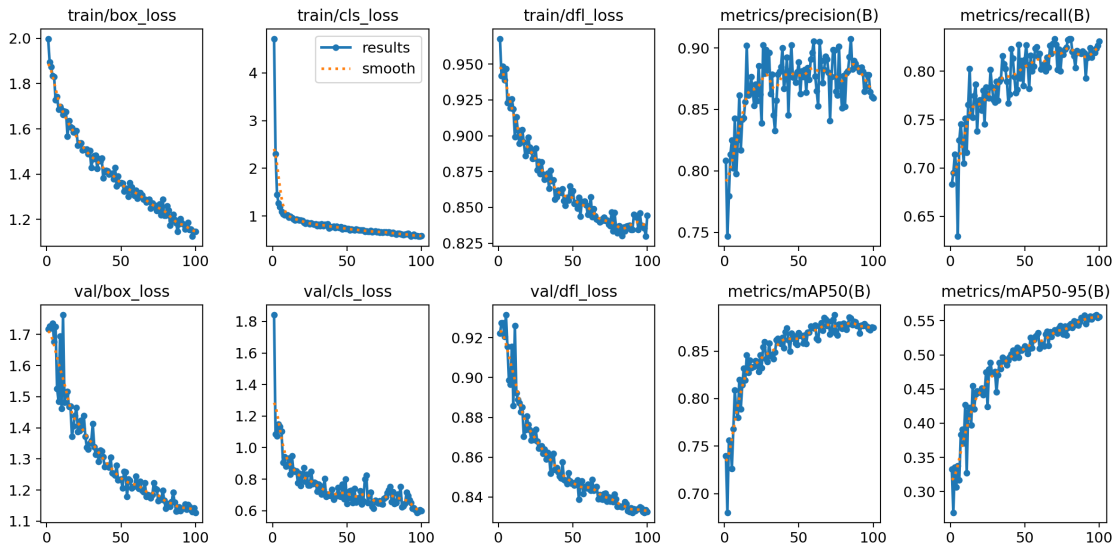


Figura 35: Gráficas resultados LWIR-Fold1.

8.3. Segmentación en Mosaicos

8.3.1. SAHI

La solución más común ante el problema de detección de objetivos, que constituyen un porcentaje muy pequeño del total de la imagen, es fragmentarla en porciones de tamaño reducido (tiling)[43]. Debido a que las imágenes LWIR - Air/Land presentan menor resolución que las imágenes NIR - Space, se usa un tamaño de fragmento de 256 píxeles para las primeras y de 512 píxeles para las segundas.

Además, se emplea un 20 % de superposición (*overlap*) ya que los objetos podrían perderse si el corte se realiza justo donde se encuentran.

La Figura 36 ilustra visualmente el proceso realizado mediante la herramienta SAHI (*Slicing aided hyper inference*) [44].

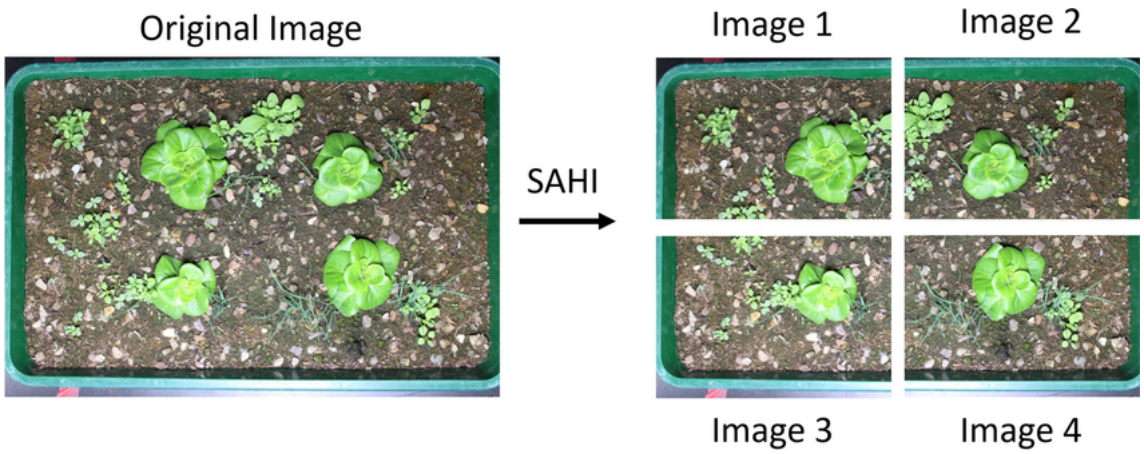


Figura 36: Ejemplo del proceso de segmentación utilizando SAHI. [45]

SAHI fue utilizado como herramienta de preprocesamiento para realizar la segmentación inicial de las imágenes. No se empleó en tiempo de inferencia, ya que los fragmentos generados fueron tratados como entradas independientes durante el entrenamiento.

Para un tamaño de mosaico de 256 píxeles, se han multiplicado por 4 las imágenes totales para el entrenamiento de LWIR, mientras que para un tamaño de mosaico de 512 píxeles, se han multiplicado por 8.

Las etiquetas con la información de la clase deben ser adaptadas para las nuevas imágenes segmentadas. Sin embargo, la biblioteca SAHI trabaja con el formato COCO mientras que Ultralytics utiliza el formato YOLO, por tanto, es necesario convertir las etiquetas a COCO, posteriormente realizar la división, lo cual generará archivos en formato COCO, y finalmente volver a convertirlos al formato YOLO.

A continuación se muestra un ejemplo de etiqueta en formato COCO:

```
images{"height":256,"width":256,"id":42,"file_name":"00051_41_0_0_256_256.png"}
annotations{"iscrowd":0,"image_id":1,"bbox":[111.0,178.0,3.0,3.0],"segmentation":
[[111,178,111,181,114,181,114,178]],"category_id":0,"id":1,"area":9}
```

Y su equivalente en formato YOLO: `0 0.912109 0.728516 0.011719 0.011719`

En el apéndice B.3 se puede encontrar más información sobre el proceso de segmentación en mosaicos.

8.3.2. Resultados

La Tabla16 muestra los resultados por pliegue para ambas divisiones:

Métrica	Pliegue1	Pliegue2	Pliegue3	Pliegue4	Pliegue5
LWIR-mAP	0,70928	0,69218	0,72161	0,69472	0,66150
LWIR-Precision	0,80229	0,81812	0,81461	0,78220	0,79409
LWIR-Recall	0,62286	0,60354	0,63819	0,62788	0,59214
NIR-mAP	0,54227	0,30780	0,53112	0,56088	0,51343
NIR-Precision	0,68872	0,56478	0,69494	0,71610	0,70030
NIR-Recall	0,47699	0,29088	0,46299	0,48421	0,43562

Tabla 16: Resultados por pliegue para LWIR y NIR.

Se puede observar que en el pliegue 2 de NIR el rendimiento es muy inferior, lo que puede indicar dos cosas:

- **1. Mala distribución de datos en el pliegue 2.** Es posible que durante la validación cruzada, el pliegue 2 haya recibido ejemplos menos representativos del conjunto general, o incluso desbalanceados. También puede contener casos atípicos o imágenes más ruidosas/difíciles de detectar correctamente.

- **2. Variabilidad en la calidad de imágenes NIR.** Las imágenes NIR pueden ser más sensibles a condiciones de iluminación, clima o reflectancia de los materiales. Si las imágenes de ese pliegue tienen más ruido, menor contraste o condiciones adversas, el modelo podría no generalizar bien.

Para facilitar la comparación y mejorar la visualización, la Tabla 17 muestra los promedios de cada división calculados como una media aritmética:

	LWIR	NIR
Precision	$0{,}80226 \pm 0{,}01322$	$0{,}67257 \pm 0{,}05485$
Recall	$0{,}61612 \pm 0{,}01674$	$0{,}43054 \pm 0{,}07158$
mAP@50	$0{,}69546 \pm 0{,}02019$	$0{,}49190 \pm 0{,}09294$

Tabla 17: Resultados promedio ($\mu \pm \sigma$) para LWIR y NIR.

Existe una discrepancia entre la tendencia de los resultados para NIR y LWIR. El recall ha aumentado en NIR, posiblemente por tener una imagen total más pequeña en la que encontrar

detecciones, pero esto podría haber provocado una pérdida de información en LWIR, disminuyendo el recall.

Sigue predominando la precisión, lo cual indica que la arquitectura continúa teniendo dificultades para detectar todos los objetivos, aunque sus predicciones son medianamente válidas.

Cada pliegue de LWIR tardó 30 minutos, mientras que en NIR necesitó 5 horas debido al tamaño y cantidad de imágenes.

Las siguientes figuras, 37 y 38, muestran las métricas de entrenamiento para el primer pliegue de cada conjunto, evidenciando que 100 épocas siguen siendo suficientes para la convergencia.

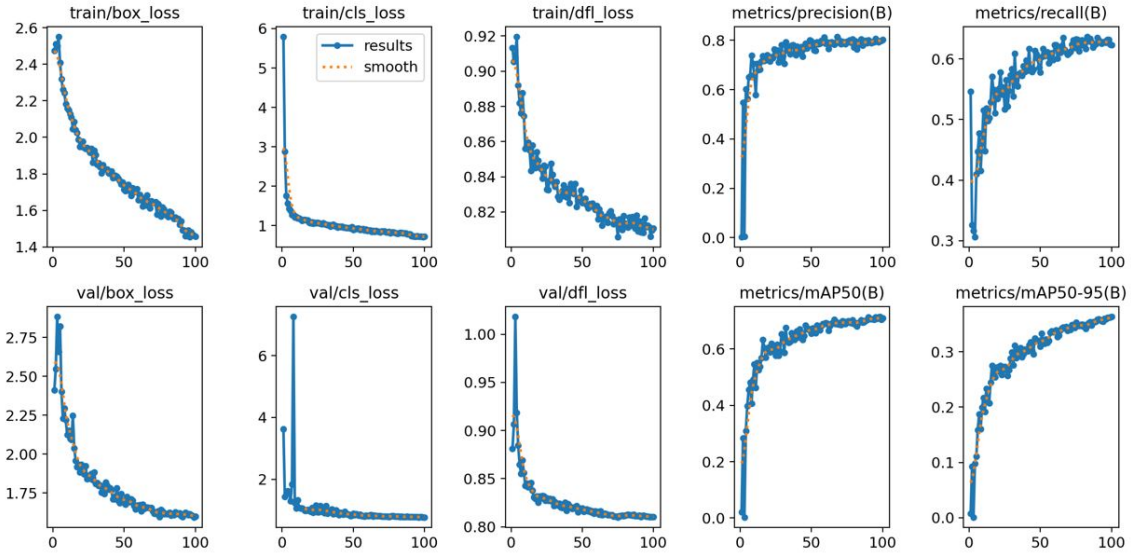


Figura 37: Métricas del entrenamiento para LWIR (pliegue 1).

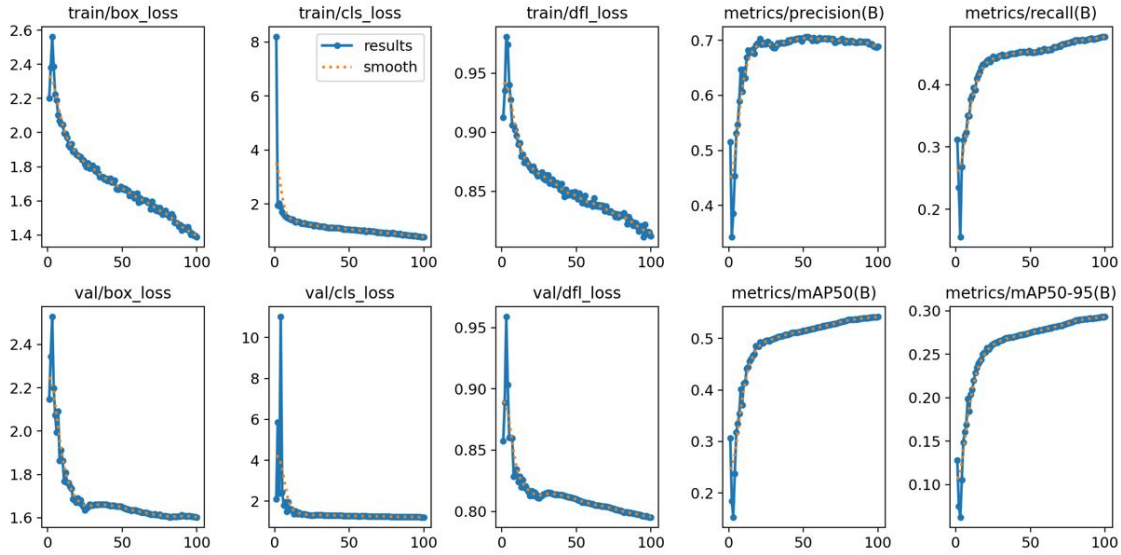


Figura 38: Métricas del entrenamiento para NIR (pliegue 1).

9

Análisis por resolución y tamaño de objetivo

Con el fin de entender mejor qué imágenes son las más complejas, se ha realizado un estudio para comparar el rendimiento según resolución y tamaño de objetivo. La información del tamaño de píxel y resolución han sido extraídas del archivo `statistics.txt` el cuál no contiene información de las 9000 imágenes del dataset sino de 8851, siendo las 149 últimas imágenes aquellas sin información asociada en el archivo de texto mencionado.

9.1. Resolución

Existen 411 resoluciones distintas, aunque la mayoría se concentra en las que se pueden visualizar en la Tabla 18 inferior:

Resolución (px)	**Número de imágenes**
1024×1024	4848
256×256	1367
512×512	1082
740×1024	457
1024×740	457

Tabla 18: Distribución de imágenes por resolución.

Las 800 imágenes restantes presentan resoluciones variadas, generalmente menores a 512×512 píxeles. Esta heterogeneidad complica el entrenamiento óptimo de los modelos, que se benefician de entradas homogéneas.

Se ha dividido el conjunto de datos de la siguiente forma:

- **Low-Res:** Imágenes con una resolución inferior a 500×500 píxeles.
- **Mid-Res:** Imágenes con una resolución entre 500×500 y 1000×1000 píxeles.
- **High-Res:** Imágenes con una resolución superior a 1000×1000 píxeles.

Tiempos de ejecución por pliegue:

- **Low-Res:** 15 minutos.
- **Mid-Res:** 30 minutos.
- **High-Res:** 60 minutos.

La Tabla 19 inferior muestra los resultados del entreno:

Métrica	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**
Low_res-mAP	0,960	0,950	0,959	0,964	0,963
Low_res-Precision	0,929	0,935	0,916	0,914	0,932
Low_res-Recall	0,896	0,894	0,910	0,926	0,893
Mid_res-mAP	0,466	0,587	0,561	0,570	0,498
Mid_res-Precision	0,712	0,806	0,761	0,789	0,756
Mid_res-Recall	0,398	0,512	0,481	0,484	0,432
High_res-mAP	0,350	0,313	0,328	0,328	0,354
High_res-Precision	0,639	0,592	0,652	0,659	0,733
High_res-Recall	0,298	0,275	0,262	0,260	0,219

Tabla 19: Resultados por resolución y métricas de cada fold.

Métrica	**Low_res**	**Mid_res**	**High_res**
Precision	$0{,}92520 \pm 0{,}00843$	$0{,}76480 \pm 0{,}03281$	$0{,}65500 \pm 0{,}05027$
Recall	$0{,}90380 \pm 0{,}01326$	$0{,}46140 \pm 0{,}04583$	$0{,}26280 \pm 0{,}02701$
mAP@50	$0{,}96040 \pm 0{,}00500$	$0{,}53640 \pm 0{,}04592$	$0{,}33460 \pm 0{,}01547$

Tabla 20: Resultados promedio ($\mu \pm \sigma$) para cada resolución.

Por legibilidad la Tabla 20 superior muestra la media por pliegues y desviación típica. Resultados drásticamente distintos según la resolución, si bien la precisión disminuye, la disminución en recall es inmensa. Debido a la descomunal diferencia en el rendimiento de estos modelos, se plantea la siguiente hipótesis:

Hipótesis 1 : ¿Es el factor más determinante en el rendimiento la resolución ?

9.2. Tamaño de objetivo

La siguiente Figura 39 muestra un histograma de los distintos tipos de tamaño de objetivos:

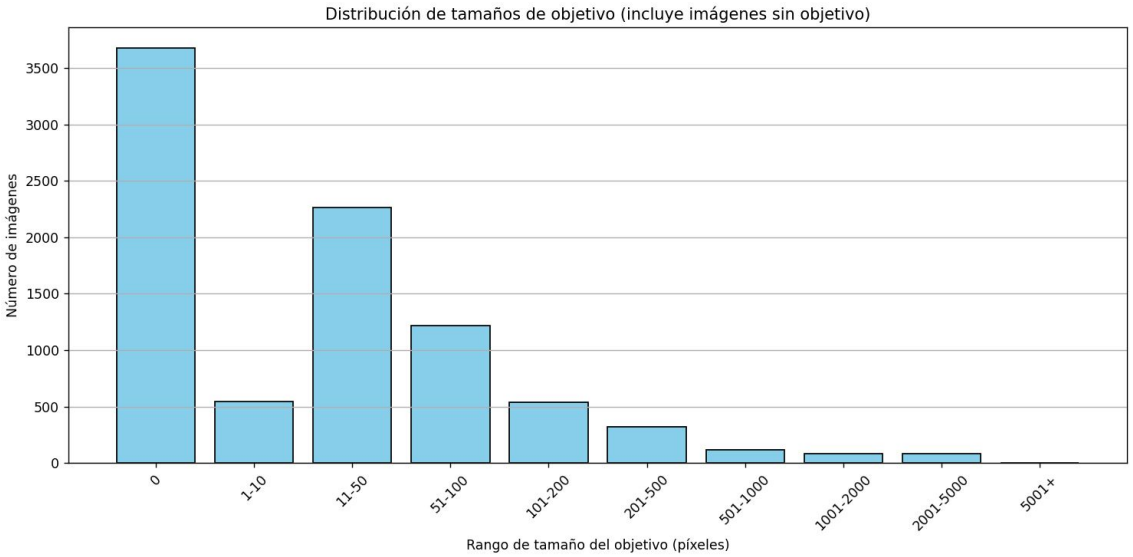


Figura 39: Histograma con 10 bins según tamaño de objetivo.

Los datos son bastante heterogéneos y desbalanceados, lo cuál puede dificultar la división. Para la división se han considerado los siguientes clases:

- **No-target:** 0 píxeles objetivo.
- **Small-target:** Entre 1 y 50 píxeles objetivo.
- **Medium-target:** Entre 51 y 200 píxeles objetivo.
- **Large-target:** Más de 201 píxeles objetivo.

El grupo de imágenes sin objetivos no puede ser estudiado ya que no hay ***ground truth*** con lo que comparar las predicciones realizadas por el modelo.
Las Figuras 40 y 41 muestran histogramas con los umbrales seleccionados: la primera restringe

los valores a 4 bins (intervalos) distintos, mientras que la segunda presenta un histograma hipotético con conjuntos de datos más balanceados (omitiendo aquellos que no contienen objetivos).

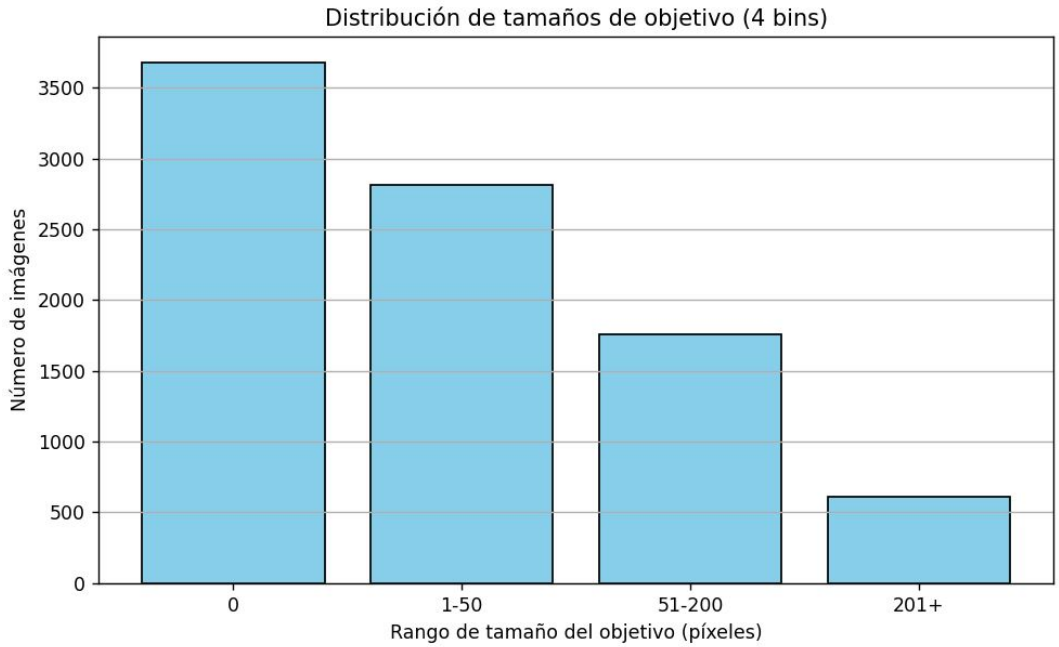


Figura 40: Histograma con 4 bins con imágenes No-target.

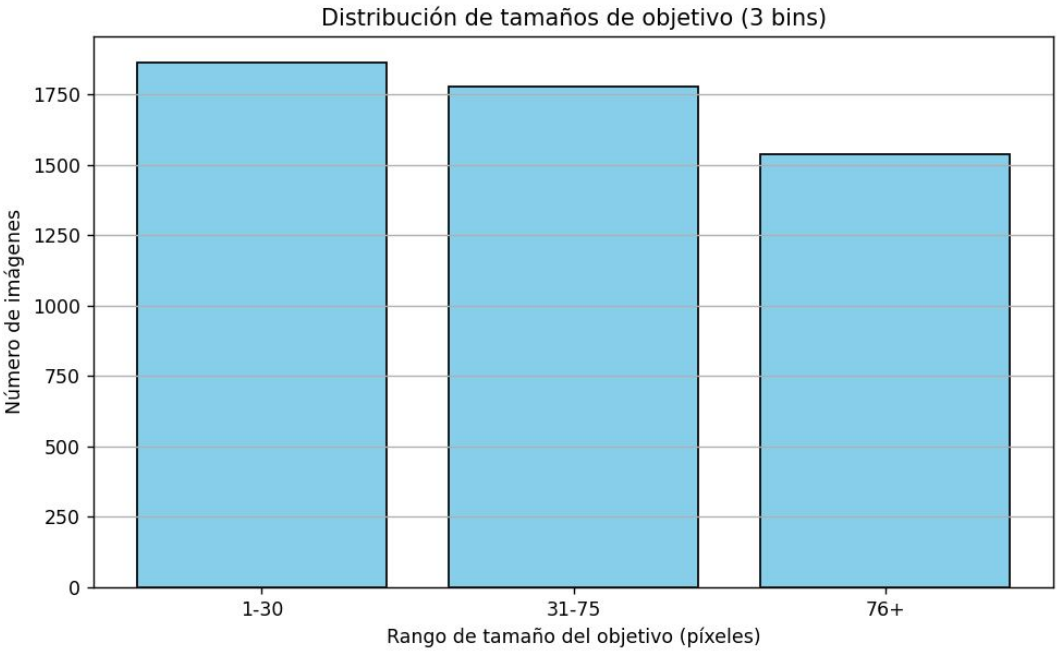


Figura 41: Histograma con 3 bins sin imágenes No-target.

Se ha priorizado una división que mantenga las características de cada grupo, si bien esto

implica conjuntos de datos desbalanceados. El problema de la distribución de la Figura 41 y por la que se ha descartado es que los dos primeros grupos de imágenes son muy similares, dando lugar a dos conjuntos de imágenes relativamente pequeñas y otro con imágenes que varían desde medianas a muy grandes.

Tiempos de ejecución por pliegue:

- Small-target: 30 minutos.
- Medium-target: 20 minutos.
- Large-target: 10 minutos.

La Tabla 21 inferior muestra los resultados del entreno:

Métrica	**Fold1**	**Fold2**	**Fold3**	**Fold4**	**Fold5**
Small_target-mAP	0,597	0,605	0,583	0,591	0,567
Small_target-Precision	0,779	0,799	0,831	0,859	0,776
Small_target-Recall	0,546	0,529	0,500	0,493	0,498
Medium_target-mAP	0,502	0,468	0,473	0,508	0,472
Medium_target-Precision	0,812	0,817	0,784	0,810	0,768
Medium_target-Recall	0,416	0,374	0,381	0,434	0,392
Large_target-mAP	0,394	0,408	0,442	0,378	0,375
Large_target-Precision	0,669	0,668	0,735	0,699	0,675
Large_target-Recall	0,326	0,347	0,363	0,305	0,308

Tabla 21: Resultados por tamaño de objetivo y métricas de cada fold.

Por legibilidad la siguiente Tabla 22 muestra la media por pliegues y desviación típica:

Métrica	**Small_target**	**Medium_target**	**Large_target**
Precision	$0{,}80880 \pm 0{,}03171$	$0{,}79820 \pm 0{,}01966$	$0{,}68920 \pm 0{,}02655$
Recall	$0{,}51320 \pm 0{,}02077$	$0{,}39940 \pm 0{,}02164$	$0{,}32980 \pm 0{,}02157$
mAP@50	$0{,}58860 \pm 0{,}01338$	$0{,}48460 \pm 0{,}01533$	$0{,}39940 \pm 0{,}02569$

Tabla 22: Resultados promedio ($\mu \pm \sigma$) por tamaño de objetivo.

Hay una clara tendencia a mejor rendimiento a menor tamaño de objetivo, disminuyendo el recall cuanto mayor sean las dimensiones de los píxeles a detectar.

Los resultados son menos dispares que aquellos que compararon resolución, por lo que se plantea la siguiente hipótesis:

Hipótesis 2 : ¿Es la bajada de rendimiento al comparar por tamaño de objetivo debida realmente a que mayor tamaño de objetivos implica mayor resolución?

9.3. Estudio

Para poder responder a las hipótesis previas se ha realizado una comparativa de donde se podrán observar las combinaciones que existen entre resolución y tamaño de píxel objetivo.

La Figura 42 muestra un gráfico donde el eje x es la resolución en el orden de millones de píxeles totales de la foto y el eje y es la cantidad de píxeles objetivos. Además muestra `Coeficiente de correlación Pearson (r)` y `Coeficiente de determinación (R²)` así como la regresión lineal:

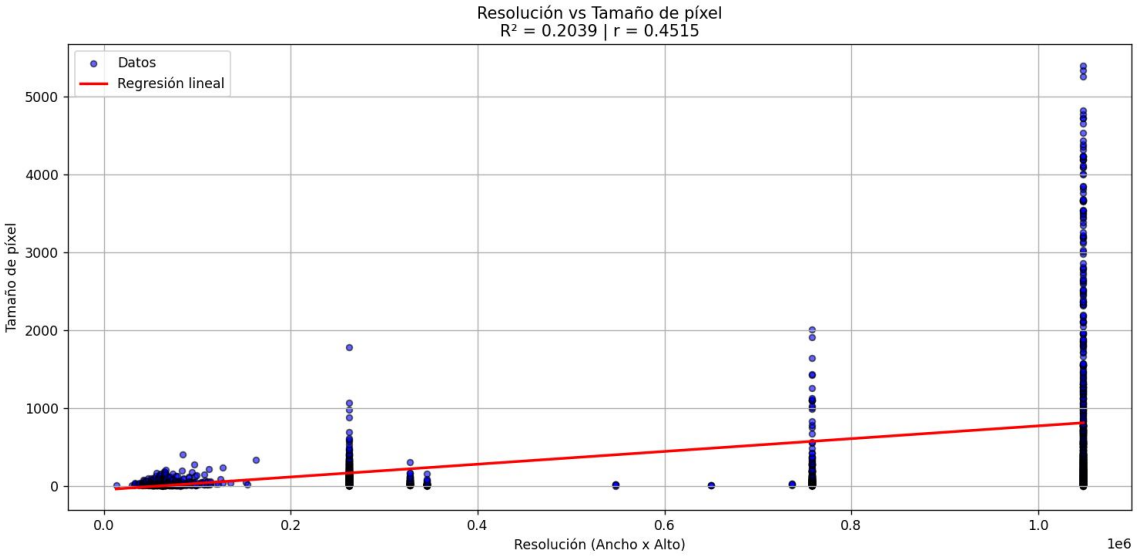


Figura 42: Regresión lineal entre resolución y cantidad de píxeles objetivo.

- **Coeficiente de correlación (r):** 0.4515

  Indica una correlación lineal moderada y positiva entre la resolución y el tamaño de píxel.

- **Coeficiente de determinación (R^2):** 0.2039

  Aproximadamente el 20 % de la variabilidad del tamaño de píxel se explica por la resolución.

- **P-valor:** $2{,}3491 \times 10^{-86}$

  Valor extremadamente pequeño que indica que la relación es estadísticamente significativa.

- **Error estándar de la pendiente:** $3{,}9287 \times 10^{-5}$

  Señala que la estimación de la pendiente es muy precisa.

La Figura 43 muestra un gráfico similar pero por legibilidad se representa como gráfico de burbujas ***(bubble plot)***. Los círculos con mayor radio representan una mayor presencia de imágenes del conjunto de datos con esa combinación específica.

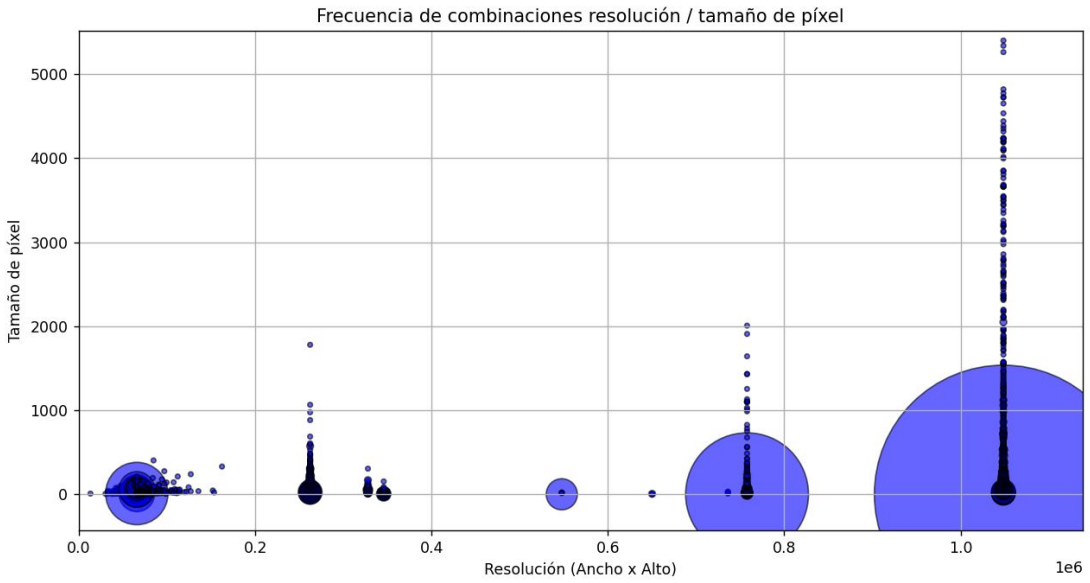


Figura 43: Relación entre resolución y tamaño de objetivo como gráfico de burbujas.

Una gran cantidad de imágenes sin objetivos se concentran resoluciones más altas, a simple vista esto podría explicar el rendimiento bajo en resoluciones de 1024x1024 píxeles o superior, sin embargo esto afectaría negativamente a la precisión, la cual no es la que más disminuye, sino el ***recall***.

Las imágenes con menor resolución presentan un abanico de tamaño de píxel menor, dicho abanico se va ampliando a medida que sube la resolución. Diferencias muy grandes en el tamaño de píxel en un mismo conjunto de datos podría dificultar un óptimo entrenamiento, al analizar imágenes con objetivos muy distintos.

Veredicto :

- **Hipótesis 1 :** Si bien no es posible esclarecer si el bajo rendimiento en resoluciones altas es parcialmente causado por el tamaño de objetivos amplio y alto, o las características de dichas imágenes (En general son NIR-SPACE), sí que se puede afirmar que la resolución es inversamente proporcional al rendimiento, por tanto se acepta parcialmente la hipótesis.

- **Hipótesis 2:** No se puede aceptar la hipótesis, ya que la correlación entre la resolución y el tamaño de los objetivos no es lo suficientemente fuerte como para atribuir la disminución del rendimiento principalmente al aumento de resolución. Esta caída en el rendimiento podría deberse, en cambio, a factores intrínsecos del tamaño del objetivo, como una mayor diversidad de valores en los ***medium*** y ***large targets***, un mayor número de imágenes sin objetivos, o la dificultad inherente de detectar múltiples objetivos o de mayor tamaño.

10 Discusión

A continuación se discutirán los resultados y se extraerán conclusiones y patrones detectados:

Análisis General y Patrones Observados

- El modelo **YOLOv8n** es el más eficiente en este contexto, ofreciendo un equilibrio óptimo entre rendimiento y coste computacional. Modelos más grandes no siempre mejoran el desempeño, confirmando el *bias-variance tradeoff*. "A medida que la complejidad de un modelo aumenta, su sesgo disminuye, pero su varianza aumenta. Se dice que un modelo con alta varianza está sobreajustando los datos, ya que es demasiado complejo y ha aprendido a ajustarse tanto al ruido en los datos de entrenamiento como a los patrones subyacentes." ([42], traducción propia)

- Las imágenes **LWIR** (baja resolución, Air/Land) presentan menor ruido, variabilidad y complejidad, lo que facilita una detección más precisa y estable. En cambio, las **NIR** (alta resolución, Space) son más problemáticas por su mayor ruido, bajo contraste y baja proporción objeto/píxeles, lo que reduce significativamente el *recall*.

- Se observa que el **recall** es el punto débil del sistema, mientras que la precisión se mantiene aceptable. Por ello, el *F1-score* maximiza con umbrales de confianza bajos, indicando un comportamiento conservador del modelo que prioriza evitar falsos positivos a costa de aumentar falsos negativos.

- El **umbral de confianza óptimo** no coincide con el predeterminado (0.25). Ajustarlo dinámicamente según el tipo de imagen puede mejorar el equilibrio entre *precision* y *recall*.

- En cuanto al tamaño del objeto, los resultados son **buenos para objetivos pequeños**, **aceptables para medianos** y **pobres para grandes**, aunque estos últimos pueden beneficiarse del uso de segmentación en mosaicos.

La Tabla 23 muestra los rendimientos promedios de cada configuración descrita en la memoria, ofreciendo una perspectiva complementaria:

Configuración	**Media de mAP@50**
General Tamaños	$0{,}4415 \pm 0{,}0221$
Específico Sin Segmentación	$0{,}5987 \pm 0{,}2853$
Específico Con Segmentación	$0{,}5937 \pm 0{,}0725$
Por Resolución	$0{,}6105 \pm 0{,}2541$
Por Tamaño De Objetivo	$0{,}4909 \pm 0{,}0771$

Tabla 23: mAP medio de cada configuración.

Se observan los siguientes resultados relevantes:

- La configuración más efectiva es **Por Resolución**, con la media más alta (0,6105), aunque con alta desviación estándar (0,2541), lo que indica variabilidad significativa entre casos.
- Las configuraciones **Específico Sin Segmentación** y **Específico Con Segmentación** presentan resultados similares. La segmentación no aporta una mejora clara, aunque reduce la variabilidad, sugiriendo mayor estabilidad.
- **General Tamaños** es la configuración con peor desempeño, con baja variabilidad pero rendimiento promedio claramente inferior, indicando que ni enfoques generales ni modelos grandes mejoran el rendimiento.
- **Por Tamaño De Objetivo** muestra un rendimiento intermedio, algo mejor que General Tamaños, pero por debajo de configuraciones específicas.

Con base en estos resultados, se pueden argumentar las siguientes conclusiones:

- La especialización del modelo para un conjunto de datos específico tiende a mejorar su rendimiento, al adaptarse mejor a las características intrínsecas del dominio y permitir un ajuste más eficaz de los pesos durante el entrenamiento, mejorando la generalización.
- La **resolución de la imagen** es un factor más determinante que el tamaño absoluto del objetivo, debido a varios motivos:

    - Un aumento en la resolución incrementa la cantidad de información espacial disponible, añadiendo una gran cantidad de píxeles de los que extraer características y ruido.
    - Si un objeto mantiene su tamaño absoluto en píxeles al aumentar la resolución sin un escalado proporcional, el área relativa respecto a la imagen disminuye, dificultando la detección.
    - Los modelos preentrenados, como los de la familia YOLO, están preentrenados con resoluciones específicas (por ejemplo, 640x640 píxeles). La discrepancia entre la resolución de preentrenamiento y la resolución de las imágenes objetivo puede afectar negativamente al aprendizaje.
    - Cuando la resolución de entrada supera la esperada, la redimensión puede conllevar pérdida de información visual crítica, afectando la precisión y sensibilidad.

- La segmentación por mosaicos no ha sido tan eficaz como se anticipaba. Aunque mostró una leve mejora en imágenes NIR, los beneficios fueron marginales, probablemente debido a la pérdida de contexto global al dividir la imagen, lo que afecta la detección de objetos parcialmente visibles o en zonas limítrofes. Aun así, este método mejora el *recall*, especialmente en imágenes NIR, aunque con una ligera disminución en *precision*, siendo más beneficioso en escenarios complejos con objetos pequeños o mal definidos.

11

Conclusiones y Líneas Futuras

11.1. Conclusiones

A continuación se expone la sección final, donde se presentan las principales conclusiones del proyecto, identificando las dificultades encontradas, valorando los resultados obtenidos y proponiendo posibles líneas de desarrollo futuro para continuar mejorando el rendimiento de los modelos en este tipo de escenarios complejos.

11.1.1. Dificultades

Las principales dificultades se han encontrado al inicio del proyecto debido tanto al tema del proyecto como el contenido del mismo.

La inteligencia artificial es un sector de vanguardia y requiere una revisión de la literatura exhaustiva para poder entender el problema a tratar, máxime considerando que se ha tratado con detección, que en general es más difícil que tareas de clasificación.

Los conceptos tratados en este trabajo suponen algo novedoso frente a los contenidos del grado por lo que el alumno ha tenido que aprender campos desconocidos. Algunas de las librerías y herramientas tampoco presentan familiaridad al alumno.

Además las imágenes infrarrojas presentan dificultades inherentes discutidas previamente a lo largo del documento.

Si bien la tecnología Ultralytics está bien documentada y posee un recorrido de una década desde su fundación, sólo comenzó a ser ampliamente conocida en el último lustro, por tanto no es tan popular como tecnologías con décadas de uso debido a una adopción masiva.

Los proyectos de inteligencia artificial son costosos computacionalmente y conllevan decenas de horas de ejecución, si bien esto fue subsanado con el uso del Servidor ICAI, dicho servidor ha de ser compartido por varios usuarios por lo que pueden ocurrir contratiempos.

11.1.2. Valoración Final

En este trabajo se ha realizado un estudio de modelos basado en redes neuronales convolucionales, utilizando la arquitectura YOLO, con el objetivo de detectar objetivos pequeños en imágenes infrarrojas en blanco y negro. Se ha profundizado e indagado en las características que dificultan o directamente impiden un rendimiento adecuado.
Se desarrollaron las 3 siguientes fases experimentales :

- Modelo General: Se entrenaron y evaluaron distintas variantes del grupo YOLOv8 (n, s, m, l) con el objetivo de obtener una visión general del rendimiento base sobre el conjunto de datos. Se analizaron métricas clave como *mAP*, *precision* y *recall*, con el fin de establecer una línea base sólida para comparaciones posteriores. Esta primera fase permitió identificar qué modelos presentaban mejor relación entre rendimiento y coste computacional, eligiendo finalmente el modelo más pequeño.

- Modelo Específico: Se realizó una partición del conjunto de datos según el tipo de espectro infrarrojo (LWIR frente a NIR), con el objetivo de estudiar si los modelos respondían de forma diferente dependiendo del tipo de imagen. Además, se llevó a cabo un experimento con segmentación en mosaicos *(tile cropping)*, para mejorar la detección de objetivos pequeños en imágenes de alta resolución, aumentando la proporción de píxeles relevantes por objetivo. Este último experimento, si bien tuvo utilidad para imágenes NIR, en general no brindó la mejora que se esperaba.

- Análisis por resolución y tamaño de objetivo: Dado que el *dataset* presenta una gran variabilidad tanto en resolución como en el tamaño de los objetos a detectar, se investigó específicamente cómo influían estos factores en el rendimiento del modelo. Se agruparon las imágenes en categorías según su resolución y se analizaron métricas por tamaño de objetivo (pequeño, mediano, grande), observando que los modelos presentan mayores dificultades a medida que la resolución y el tamaño de objetivo aumenta.

11.2. Líneas Futuras

Las líneas futuras que podrían plantearse para el desarrollo posterior de este trabajo son muy diversas:

- Experimentar con otras arquitecturas de redes neuronales convolucionales que puedan complementar o superar a YOLO en este contexto, tales como EfficientNet[46], ConvNeXt[47], o variantes más recientes de YOLO (YOLOv9 o superior). Estas arquitecturas podrían ofrecer mejoras en eficiencia computacional o capacidad de generalización.

- Usar una mayor cantidad de aumento de datos o experimentar con el preprocesamiento de imágenes, así como una mayor cantidad de segmentación en imágenes grandes. El uso de técnicas como *histogram equalization*, desenfoque adaptativo o generación sintética de objetos podría enriquecer la variedad de ejemplos.

- Usar un conjunto de datos mayor y más homogéneo tanto en resoluciones, como en tipo de imagen, como en tamaño de objetivos. Esto reduciría la varianza entre ejemplos y mejoraría la robustez del modelo. Además, facilitaría la obtención de métricas más representativas y comparables entre diferentes configuraciones.

- Experimentar con hiperparámetros para hallar una combinación óptima que pueda mejorar las métricas como ajustes en la tasa de aprendizaje *(learning rate)*, impulso *(momentum)* o congelar capas iniciales. Utilizar herramientas como Optuna[48] o *grid search* podría automatizar este proceso de forma eficiente.

Apéndice A
Manual de Instalación

Instrucciones:

- 1. Abrir terminal.
- 2. Crear entorno anaconda: **-conda create –name nombreEntorno python=3.11.12**
- 3. Activar entorno: **-source activate nombreEntorno**
- 4. Instalar dependencias: **-conda install –file requirements.txt**
- 5. Descargar conjunto de datos: **-python setup.py**

Apéndice B
Manual de Ejecución

B.1. Archivos BoundingBox

Con la librería OpenCV se pueden encontrar las fronteras de las áreas de interés con un cambio de intensidad (blanco contra negro) e iterar sobre cada una para crear una *bounding box* y posteriormente obtener las medidas normalizadas. Comando Ejecución:

-python boundingBoxes.py maskFolder outputFolder

Lógica interna explicada:

- 1. Crear una carpeta de salida si no existe.

```
os.makedirs(outputFolder, exist_ok=True)
```

- 2. Para cada imagen `.png` en la carpeta **maskFolder**:

```
for mask_filename in os.listdir(maskFolder):
```

  - 2.1. Leer la imagen y convierte su máscara en contornos.

```
gray = cv2.cvtColor(mask_image, cv2.COLOR_BGR2GRAY)
contours, _ = cv2.findContours(gray, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

  - 2.2. Calcular cajas delimitadoras normalizadas (formato YOLO).

```
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    x_center = (x + w / 2) / width
    y_center = (y + h / 2) / height
    norm_width = w / width
    norm_height = h / height
        bounding_boxes.append([0, x_center, y_center,
        ↪  norm_width, norm_height])
```

- 2.3. Guardar estas cajas en un archivo **.txt** con el mismo nombre.

```
txt_filename = os.path.splitext(mask_filename)[0] +
↪  ".txt"
txt_path = os.path.join(outputFolder, txt_filename)
with open(txt_path, "w") as f:
for bbox in bboxes:
   f.write(" ".join(map(str, bbox)) + "\n")
```

B.2. Pliegues

La librería Sklearn facilita la creación de los *folds* dividiendo los datos aleatoriamente, posteriormente al iterar cada pliegue se crean las carpetas y se distribuyen las imágenes en ellas. Comando Ejecución:

-python foldSetup.py imgFolder labelFolder outputFolder foldsNumber
Lógica interna explicada:

- 1. Crear la carpeta base de salida si no existe.

```
os.makedirs(outputFolder, exist_ok=True)
```

- 2. Obtener las imágenes **.png** de la carpeta de entrada y empareja cada una con su archivo de etiquetas correspondiente.

```
images = sorted([f for f in os.listdir(imgFolder) if
↪  f.endswith(".png")])
data = [(img, img.replace(".png", ".txt")) for img in images
if os.path.exists(os.path.join(labelFolder,
↪  img.replace(".png", ".txt")))]
```

- 3. Aplicar validación cruzada (**KFold**) con n divisiones.

```
kf = KFold(n_splits=foldsNumber, shuffle=True,
↪  random_state=42)
```

- 4. Para cada pliegue:

```
for train_index, test_index in kf.split(data):
```

- 4.1 Crear las carpetas necesarias para entrenamiento y validación (imágenes y etiquetas).

```
os.makedirs(train_img_folder, exist_ok=True)
os.makedirs(train_label_folder, exist_ok=True)
os.makedirs(test_img_folder, exist_ok=True)
os.makedirs(test_label_folder, exist_ok=True)
```

- 4.2 Copiar las imágenes y etiquetas del conjunto de entrenamiento a las carpetas correspondientes.

```
for idx in train_index:
    img, label = data[idx]
    shutil.copy(os.path.join(imgFolder, img),
    ↪ train_img_folder)
    shutil.copy(os.path.join(labelFolder, label),
    ↪ train_label_folder)
```

- 4.3 Copiar las imágenes y etiquetas del conjunto de validación a las carpetas correspondientes.

```
for idx in test_index:
    img, label = data[idx]
    shutil.copy(os.path.join(imgFolder, img),
    ↪ test_img_folder)
    shutil.copy(os.path.join(labelFolder, label),
    ↪ test_label_folder)
```

B.3. Slicing

La herramienta SAHI permite realizar el *slicing* de imágenes y anotaciones en formato COCO de forma eficiente. Para ello, primero se realiza la conversión de anotaciones desde el formato YOLO al formato COCO, luego se aplica el corte con superposición y finalmente se convierte el resultado de nuevo a YOLO, reorganizando las imágenes y etiquetas. Comando Ejecución:

-python slicing.py baseFolder outputFolder sliceSize overlap

Lógica interna explicada:

- 1. Para cada subconjunto de cada pliegue:

```
FOLDS = ["fold_1", "fold_2", "fold_3", "fold_4", "fold_5"]
for fold in FOLDS:
    for split in ["train", "test"]:
        process_fold_split(fold, split)
```

- 2. Convertir las anotaciones de YOLO a COCO para cada pliegue y subconjunto (entrenamiento o validación).

```
convert_yolo_to_coco(image_dir, label_dir, json_path, categories)
```

- 3. Aplica slicing sobre las imágenes y anotaciones COCO con SAHI.

```
slice_coco(
    coco_annotation_file_path=json_path,
    output_coco_annotation_file_name="outputFolder.json",
    image_dir=image_dir,
    output_dir=slice_base_dir,
    slice_height=sliceSize,
    slice_width=sliceSize,
    overlap_height_ratio=overlap,
    overlap_width_ratio=overlap,
    ignore_negative_samples=False
)
```

- 4. Convertir las nuevas anotaciones generadas por SAHI de COCO a YOLO y reorganiza los archivos.

```
coco_to_yolo(new_json_path, label_output_dir, image_output_dir)
```

B.4. Entrenamiento

Instrucciones:

- 1. Abrir terminal.

- 2. Activar entorno :

  -source activate nombreEntorno

- 3. Acceder a directorio con los archivos necesarios para el entrenamiento :

  -cd data La siguiente Figura 44 muestra el directorio usado en el servidor:

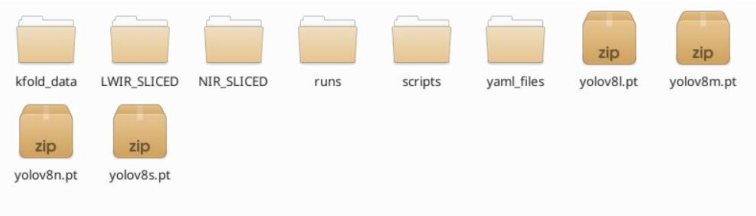


Figura 44: Ejemplo de directorio de trabajo.

- 4. Ejecutar *script* de entrenamiento:

  -python train.py imgSize batchSize yamlFolder model

  Lógica interna explicada:

  - 1. Obtener todos los archivos **.yaml** dentro de la carpeta especificada.

    ```
    archivos_yaml = sorted(yamlFolder.glob("*.yaml"))
    ```

  - 2. Recorrer cada archivo **.yaml** y entrenar un modelo YOLO por separado.

    ```
    for archivo_yaml in archivos_yaml:
        nombre_experimento = archivo_yaml.stem
        modelo = YOLO(model)
        modelo.train(
            data=str(archivo_yaml),
            epochs=100,
            imgsz=imgSize,
            batch=batchSize,
            name=nombre_experimento,
            project="runs/train",
            device=0,
            workers=4
        )
    ```

Apéndice C
Manual de Ultralytics

Información recogida de la página oficial de Ultralytics [49].

C.1. Configuración Hiperparámetros

Ultralytics ofrece decenas de hiperparámetros cuya modificación afecta al rendimiento. La Tabla 24 muestra algunos de los más importantes:

Parámetro	**Por defecto**	**Descripción**
`model`	None, str	Ruta al modelo preentrenado o config .yaml
`data`	None, str	Ruta al archivo .yaml con info del dataset
`epochs`	100, int	Número total de épocas
`batch`	16, int	Tamaño del batch o -1 para automático
`imgsz`	640, int o list	Tamaño de imagen de entrada
`device`	None, str/int	GPU, CPU o lista de dispositivos
`patience`	100, int	Early stopping: épocas sin mejora
`lr0`	0.01, float	Tasa de aprendizaje inicial
`weight_decay`	0.0005, float	Regularización L2
`warmup_epochs`	3.0, float	Épocas de calentamiento
`pretrained`	True, bool/str	Usar pesos preentrenados
`val`	True, bool	Realizar validación por época
`amp`	True, bool	Usa precisión mixta (más rápido)
`seed`	0, int	Semilla para reproducibilidad
`save_conf`	False, bool	Guarda la confianza de las predicciones
`save_json`	False, bool	Guarda las predicciones en formato COCO
`save_txt`	False, bool	Guarda las predicciones en formato YOLO

Tabla 24: Parámetros de configuración más comunes

C.2. Data Augmentation

Por defecto Ultralytics realiza una serie de modificaciones a las imágenes de entrenamiento para obtener más imágenes de forma artificial, se han omitido fotométricas ya que no aplican o tienen efecto nulo en imágenes en blanco y negro.
La Tabla 25 muestra algunos de los parámetros principales de data augmentation:

Parámetro	**Tipo**	**Rango / Valor**	**Descripción**
`degrees`	float	0.0 – 180	Rotación aleatoria de la imagen.
`translate`	float	0.0 – 1.0	Desplazamiento horizontal y vertical.
`scale`	float	$\geq$ 0.0	Escala la imagen, simulando objetos a diferentes distancias.
`shear`	float	-180 – 180	Aplica distorsión angular.
`perspective`	float	0.0 – 0.001	Transformación en perspectiva.
`flipud`	float	0.0 – 1.0	Voltea la imagen verticalmente con cierta probabilidad.
`fliplr`	float	0.0 – 1.0	Voltea la imagen horizontalmente, aumenta la diversidad.
`mosaic`	float	0.0 – 1.0	Combina 4 imágenes en una.
`mixup`	float	0.0 – 1.0	Mezcla 2 imágenes, introduciendo ruido en las etiquetas.
`cutmix`	float	0.0 – 1.0	Mezcla regiones de dos imágenes.

Tabla 25: Parámetros de Data Augmentation

Apéndice D
Glosario

A continuación se expone la terminología presente en la memoria en orden alfabético:

- **Algoritmo de detección:** Método computacional utilizado para identificar y localizar objetivos en una imagen.
- **Anomalía:** Elemento o característica que se desvía del patrón típico de una imagen, indicando la posible presencia de un objetivo pequeño o ruido.
- **Anotación:** Proceso de asignar etiquetas o cajas delimitadoras a los objetos de interés en un conjunto de datos para entrenar modelos supervisados.
- **Batch (tamaño del lote):** Conjunto de muestras procesadas simultáneamente durante una iteración del entrenamiento de una red neuronal. Permite aprovechar la paralelización en GPU.
- **Bajo contraste:** Condición en la que las diferencias entre los valores de intensidad de los píxeles son mínimas, dificultando la detección de objetivos.
- **Bin:** Intervalo dentro de un histograma que agrupa un rango de valores de una variable, acumulando la frecuencia de datos que caen dentro de su rango, permitiendo representar distribuciones de manera discreta.
- **Bubble plot:** Tipo de gráfico de dispersión en el que cada punto se representa con un círculo cuyo tamaño es proporcional a una tercera variable (frecuencia, magnitud, etc.).
- **Capas Fully Connected:** Capas en una red neuronal en las que cada neurona está conectada a todas las neuronas de la capa anterior, utilizadas para combinar y transformar las características extraídas previamente.
- **Coeficiente de correlación (r):** Medida estadística que indica la fuerza y dirección de la relación lineal entre dos variables.
- **Coeficiente de determinación (R^2):** Proporción de la varianza en la variable dependiente que es explicada por la variable independiente en un modelo de regresión lineal.

- **Congelación de capas:** Técnica de entrenamiento en la que se mantienen fijos los pesos de ciertas capas de una red neuronal, para preservar el conocimiento aprendido previamente y reducir el coste computacional durante el Finetuning.

- **Desenfoque adaptativo:** Técnica que suaviza una imagen variando la intensidad del desenfoque según el contenido local, preservando bordes importantes mientras reduce el ruido en otras zonas.

- **Desviación Típica (σ):** Medida que indica cuánto varían o se dispersan los valores de un conjunto de datos con respecto a su media. Se calcula como la raíz cuadrada de la varianza.

  item **Distancia euclídea:** Medida de la distancia directa entre dos puntos en un espacio n-dimensional, calculada como la raíz cuadrada de la suma de las diferencias cuadradas entre coordenadas correspondientes.

- **Error estándar de la pendiente:** Medida de la precisión con la que se estima la pendiente en un modelo de regresión lineal; indica la variabilidad esperada de la pendiente si se repitiera el muestreo.

- **FLOPs (Floating Point Operations):** Métrica que representa la cantidad de operaciones en coma flotante necesarias para que un modelo realice una inferencia.

- **Fine-tuning:** Técnica de aprendizaje profundo que consiste en partir de un modelo preentrenado y ajustar sus pesos mediante un entrenamiento adicional con un nuevo conjunto de datos específico.

- **Grid Search:** Método de optimización que consiste en probar todas las combinaciones posibles de un conjunto predefinido de hiperparámetros para encontrar la mejor configuración del modelo.

- **Ground Truth:** Conjunto de datos de referencia que contiene las anotaciones verdaderas y verificadas sobre los objetos presentes en una imagen.

- **Hiperparámetro:** Valor configurado manualmente antes del entrenamiento de un modelo que controla su comportamiento, como la tasa de aprendizaje, el número de épocas o el tamaño del lote.

- **Histogram Equalization:** Técnica de procesamiento de imagen que ajusta el contraste distribuyendo uniformemente los niveles de intensidad.
- **Imagen infrarroja:** Imagen capturada utilizando sensores que detectan radiación en el espectro infrarrojo, comúnmente empleada en aplicaciones de vigilancia y monitoreo.
- **Imágenes sintéticas:** Imágenes generadas artificialmente por un sistema, en lugar de ser capturadas por un sensor real.
- **Impulso (ρ):** Técnica utilizada en el entrenamiento de redes neuronales que acumula un historial de gradientes anteriores para acelerar la convergencia y evitar quedarse atrapado en mínimos locales.
- **Intensidad del píxel:** Valor numérico asociado a cada píxel de una imagen que representa su brillo o nivel de energía en el espectro capturado.
- **Mapa de características:** Representación en múltiples capas de una imagen, generada por redes neuronales, que resalta patrones relevantes para la detección de objetivos.
- **Máscara:** Representación binaria que indica las áreas de interés en una imagen, generalmente utilizada para segmentación de objetos.
- **Media (μ):** Valor promedio de un conjunto de datos, calculado sumando todos los valores y dividiendo entre el número total de observaciones. Representa una medida de tendencia central.
- **Nivel de ruido:** Interferencia o distorsión presente en una imagen que dificulta el proceso de detección.
- **Objetivo pequeño:** Elemento de dimensiones reducidas en una imagen que debe ser identificado y clasificado por el sistema.
- **Operaciones de Convolución:** Operaciones fundamentales en redes neuronales convolucionales (CNN) que aplican filtros (kernels) deslizantes sobre la entrada para extraer características espaciales y patrones locales relevantes en imágenes, como bordes, texturas o formas.
- **Operaciones de Pooling:** Procesos que reducen la dimensionalidad espacial de las representaciones intermedias en una red convolucional, resumiendo regiones mediante

funciones como máximo (max pooling) o promedio (average pooling), con el fin de disminuir el coste computacional.

- **Penalización por diferencia en la relación de aspecto:** Estrategia utilizada en tareas de detección de objetos para castigar discrepancias entre la relación de aspecto (alto/ancho) de las cajas predichas y las reales, mejorando así la precisión de las predicciones.

- **Principio de inclusión-exclusión:** Regla de la teoría de conjuntos que permite calcular el tamaño de la unión de conjuntos considerando sus intersecciones.

- **P-valor:** Probabilidad de obtener resultados al menos tan extremos como los observados, bajo la hipótesis nula; un p-valor pequeño indica evidencia suficiente para rechazar la hipótesis nula (suposición estándar que se define como la predicción de que no hay interacción entre las variables).

- **Regresión lineal:** Técnica estadística que modela la relación entre una variable dependiente y una o más variables independientes mediante una ecuación lineal.

- **Robustez:** Capacidad del modelo para mantener un buen desempeño frente a variaciones o perturbaciones en los datos de entrada, como ruido, cambios en la distribución o datos no vistos durante el entrenamiento.

- **Segmentación en mosaicos:** Proceso de dividir una imagen en regiones más pequeñas para facilitar la identificación de características específicas.

- **Sesgo y varianza:** Componentes del error de un modelo. El sesgo se refiere a los errores introducidos por suposiciones simplificadas en el modelo, mientras que la varianza representa la sensibilidad del modelo a las fluctuaciones en los datos de entrenamiento.

- **Sobreajuste:** Situación en la que un modelo aprende demasiado bien los datos de entrenamiento, incluyendo ruido o patrones irrelevantes, lo que deteriora su capacidad para generalizar a nuevos datos.

- **SSH:** Protocolo de red que permite el acceso remoto seguro a otros equipos a través de una red insegura.

- **Tasa de aprendizaje (η):** Parámetro que controla cuánto se ajustan los pesos del modelo en cada iteración durante el entrenamiento.
- **Tensor:** Estructura de datos multidimensional utilizada en computación y aprendizaje automático para almacenar y manipular datos, como imágenes, vectores o matrices.
- **Umbral de confianza:** Es el valor mínimo de probabilidad que una predicción debe superar para ser considerada válida o aceptada como positiva.
- **Umbral de detección:** Valor predeterminado que define los límites para clasificar un píxel o región como parte de un objetivo.
- **Visión por Computador:** Campo de la informática que desarrolla sistemas para interpretar información visual de imágenes y videos.

Referencias

[1] Nimbus. *INTERPRETACION Y USO DE LAS IMAGENES INFRARROJAS (IR) TOMADAS DESDE SATELITES METEOROLOGICOS*. 2001. URL: https://www.webnorte.com/parapente/meteo/articulos/Interpretacion-%20infrarrojas.htm (visitado 17-10-2024).

[2] Global Marketing Insights. *Infrared Imaging Market Trends ← Forecast Report, 2032*. n.d. URL: https://www.gminsights.com/es/industry-analysis/infrared-imaging-market (visitado 10-12-2024).

[3] Joseph Redmon et al. "You only look once: Unified, real-time object detection". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.

[4] Luca Antiga, Eli Stevens y Thomas Viehmann. *Deep Learning with PyTorch*. New York: Manning Publications Co. LLC, 2020.

[5] Jing Li et al. "Visual Detail Augmented Mapping for Small Aerial Target Detection". En: *Remote Sensing* (2018). URL: https://www.researchgate.net/figure/Some-small-object-detection-results-with-You-Only-Look-Once-Version-2-YOLO-v2-deep_fig1_329854566 (visitado 15-11-2024).

[6] U. Snekhalatha, K. Palani Thanaraj y Kurt Ammer. *Artificial Intelligence-Based Infrared Thermal Image Processing and Its Applications*. CRC Press, 2022.

[7] Elion. *La cuarta revolución industrial*. 2025. URL: https://www.elion.es/tecnologias/industry40/ (visitado 10-12-2024).

[8] ABAMobile. *Aplicaciones de la inteligencia artificial en las empresas*. n.d. URL: https://abamobile.com/web/aplicaciones-de-la-inteligencia-artificial-en-las-empresas/ (visitado 10-12-2024).

[9] Oscar García-Olalla Olivera. *Redes Neuronales Artificiales: Qué son y cómo se entrenan*. 2019. URL: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i (visitado 10-12-2024).

[10] Walther Curo De La Cruz. *Crea tu primera Red Neuronal Artificial*. 2024. URL: https://blog.walthercuro.com/crear-tu-primera-red-neuronal-artificial/ (visitado 04-05-2025).

[11] Egor Howell. *Convolution Explained – Introduction to Convolutional Neural Networks*. 2023. URL: https://towardsdatascience.com/convolution-explained-introduction-to-convolutional-neural-networks-5babc47fbcaa/ (visitado 04-05-2024).

[12] Marcos Esparza Arizpe. *Clasificación de imágenes con redes convolucionales*. 2022. URL: https://medium.com/@marcosesparzaarizpe/clasificaci%C3%B3n-de-im%C3%A1genes-con-redes-convolucionales-ee98a7dd7697 (visitado 04-05-2025).

[13] Jacar - javi. *Función SoftMax: Activación para la clasificación*. 2023. URL: https://jacar.es/funcion-softmax-activacion-para-la-clasificacion/ (visitado 04-05-2025).

[14] Gustavo Cimas Cuadrado. *Qué es Python: Características, evolución y futuro*. 2019. URL: https://openwebinars.net/blog/que-es-python/ (visitado 10-12-2024).

[15] Ultralytics. *Ultralytics: Computer Vision AI for the Future*. 2025. URL: https://www.ultralytics.com/es (visitado 10-12-2024).

[16] Candela García Fernández. *OpenCV: Introducción y su rol en la visión por computadora*. 2024. URL: https://openwebinars.net/blog/opencv-introduccion-y-su-rol-en-la-vision-por-computadora/ (visitado 10-12-2024).

[17] Ander Fernández Jauregui. *Tutorial Sklearn Python*. n.d. URL: https://anderfernandez.com/blog/tutorial-sklearn-machine-learning-python/ (visitado 10-12-2024).

[18] Alfredo Sánchez Alberca. *La librería Matplotlib*. 2020. URL: https://aprendeconalf.es/docencia/python/manual/matplotlib/ (visitado 20-05-2025).

[19] Fatih Cagatay Akyon et al. *SAHI: A lightweight vision library for performing large scale object detection and instance segmentation*. Nov. de 2021. DOI: 10.5281/zenodo.5718950. URL: https://doi.org/10.5281/zenodo.5718950.

[20] Izary Rondón. *¿Qué es Anaconda?* 2022. URL: https://www.anaconda.com/ (visitado 15-01-2025).

[21] Carol Ramos. *¿Qué es FileZilla y cómo se usa?* 2024. URL: https://www.lucushost.com/blog/filezilla/ (visitado 15-01-2025).

[22] Emilio Torres Manzanera. *Acceso remoto a un servidor vía el entorno gráfico X2Go*. 2020. URL: https://torres.epv.uniovi.es/centon/servidor-x2go-acceso.html (visitado 15-01-2025).

[23] Faris A. Kateb et al. "Researchgate: Arquitectura tronco cuello cabeza." En: *Agronomy* (2021). URL: https://www.researchgate.net/figure/A-detection-model-contains-a-backbone-neck-head-module-The-backbone-module-exploits_fig1_356638292 (visitado 04-05-2025).

[24] Aditya Sharma. *Pyimagesearch: Yolov8 detección de objetos*. 2023. URL: https://pyimagesearch.com/2023/05/01/training-the-yolov8-object-detector-for-oak-d/ (visitado 04-05-2025).

[25] Ultralytics. *Función de pérdida*. 2025. URL: https://www.ultralytics.com/es/glossary/loss-function (visitado 04-02-2025).

[26] Ultralytics. *Código Fuente Función de Pérdida*. 2025. URL: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/utils/loss.py (visitado 04-02-2025).

[27] Prakash Chandra. *IoU Loss Functions for Faster & More Accurate Object Detection*. 2023. URL: https://learnopencv.com/iou-loss-functions-object-detection/ (visitado 04-02-2025).

[28] soumyadip. *YOLO Loss Function Part 2: GFL and VFL Loss*. 2024. URL: https://learnopencv.com/yolo-loss-function-gfl-vfl-loss/ (visitado 04-02-2025).

[29] Elven Kim. *Distribution Focal Loss (DFL) in YOLO colab*. 2024. URL: https://medium.com/@elvenkim1/dual-focal-loss-dfl-in-yolo-colab-0c9ac722f917 (visitado 04-02-2025).

[30] ICPR Challenge Team. *ICPR Challenge: Infrared Small Target Detection*. 2024. URL: https://limitirstd.github.io (visitado 20-10-2024).

[31] ViewSheen. *What is NIR/SWIR/MWIR/LWIR/FIR Spectral Range?* 2022. URL: https://www.viewsheen.com/blog/what-is-nir-swir-mwir-lwir-fir-spectral-range-/ (visitado 04-05-2024).

[32] Aduen Benjumea et al. “YOLO-Z: Improving small object detection in YOLOv5 for autonomous vehicles”. En: *arXiv preprint arXiv:2112.11798* (2023). URL: https://arxiv.org/pdf/2112.11798 (visitado 12-05-2025).

[33] Jianjun Ni et al. “A Small-Object Detection Model Based on Improved YOLOv8s for UAV Image Scenarios”. En: *Remote Sensing* (2024). URL: https://www.mdpi.com/2072-4292/16/13/2465 (visitado 12-05-2025).

[34] Oracle. *Pipelines*. 2025. URL: https://docs.oracle.com/en-us/iaas/Content/data-science/using/pipelines-about.htm (visitado 31-05-2025).

[35] Ultralytics. *Descenso Gradiente*. 2025. URL: https://www.ultralytics.com/es/glossary/gradient-descent (visitado 04-02-2025).

[36] Ultralytics. *Optimizador Adam*. 2025. URL: https://www.ultralytics.com/es/glossary/adam-optimizer (visitado 04-02-2025).

[37] Ultralytics. *Descenso Gradiente Estocástico (SGD)*. 2025. URL: https://www.ultralytics.com/es/glossary/stochastic-gradient-descent-sgd (visitado 04-02-2025).

[38] People For AI. *Bounding Box Definition*. 2025. URL: https://www.peopleforai.com/glossary/bounding-box/ (visitado 17-10-2024).

[39] Hu Lu. “Click-cut: a framework for interactive object selection”. En: *Multimedia Tools and Applications* (2021). URL: https://www.researchgate.net/figure/Two-manually-generated-binary-mask-ground-truths-Left-the-original-image-Middle_fig3_350791722 (visitado 16-03-2025).

[40] Wikipedia. *Validación cruzada*. 2020. URL: https://es.wikipedia.org/w/index.php?title=Validaci%C3%B3n_cruzada&oldid=124047241 (visitado 10-12-2024).

[41] Talking with data. *Aprendizaje automático de validación cruzada: K-Fold*. 2023. URL: https://talkingwithdata.medium.com/asegura-el-%C3%A9xito-de-tus-modelos-de-machine-learning-el-poder-de-la-validaci%C3%B3n-cruzada-701e7635e299 (visitado 04-05-2024).

[42] Editorial Team. *Mastering the Bias-Variance Dilemma: A Guide for Machine Learning Practitioners*. 2023. URL: https://towardsai.net/p/l/mastering-the-bias-variance-dilemma-a-guide-for-machine-learning-practitioners (visitado 24-05-2025).

[43] Jacob Solawetz Linas Kondrackis. *Roboflow: Detección de objetos pequeños*. 2024. URL: https://blog.roboflow.com/detect-small-objects/ (visitado 10-12-2024).

[44] Fatih Cagatay Akyon, Sinan Onur Altinuc y Alptekin Temizel. "Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection". En: *2022 IEEE International Conference on Image Processing (ICIP)* (2022), págs. 966-970. DOI: 10.1109/ICIP46576.2022.9897990.

[45] Harry Rogers et al. "Advancing precision agriculture: domain-specific augmentations and robustness testing for convolutional neural networks in precision spraying evaluation". En: *Neural Computing and Applications* (2024). URL: https://www.researchgate.net/figure/Slicing-aided-hyper-inference-SAHI-process-applied-on-the-collected-data_fig6_383059121 (visitado 16-05-2025).

[46] Moreluz-ia-Juan. *Efficientnet*. 2023. URL: https://moreluz-ia.com/ia/deep-learning/neuronal-networking/efficientnet/ (visitado 23-05-2025).

[47] Akira Sakamoto. *ConvNeXt: El futuro de las redes convolucionales*. 2023. URL: https://docs.kanaries.net/es/topics/ChatGPT/convnext (visitado 23-05-2025).

[48] Optuna. *¿Qué es Optuna? Hiperparámetros, enfoque y características*. n.d. URL: https://ciberseguridad.com/guias/nuevas-tecnologias/machine-learning/optuna/ (visitado 24-05-2025).

[49] Ultralytics Team. *Configuración*. 2023. URL: https://docs.ultralytics.com/usage/cfg/ (visitado 10-12-2024).

UNIVERSIDAD DE MÁLAGA | uma.es

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

E.T.S. DE INGENIERÍA INFORMÁTICA