GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Diseño y desarrollo de un sistema de detección automática de presencia humana en imágenes de videovigilancia

Autor: Alejandro Alamán San Martín

Director: Emilio Manuel Domínguez Adán

Madrid julio 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño y desarrollo de un sistema de detección automática de presencia humana en imágenes de videovigilancia

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Firmado por ALAMAN SAN MARTIN ALEJANDRO - ***9260** el día 01/08/2025 con un certificado emitido por AC FNMT Usuarios

Fdo.: Alejandro Alamán San Martín Fecha: 31/07/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

DOMINGUEZ ADAN EMILIO MANUEL - 02620715A

Firmado digitalmente por DOMINGUEZ ADAN EMILIO MANUEL - 02620715A
Fecha: 2025.07.31 22:33:36 +02'00'

Fdo.: Emilio Manuel Domínguez Adán Fecha: 31/07/2025

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Diseño y desarrollo de un sistema de procesado de imágenes para la determinación de incidentes relacionados con presencia humana no autorizada

Autor: Alejandro Alamán San Martín

Director: Emilio Manuel Domínguez Adán

Madrid julio 2025

Agradecimientos

Gracias a mi director, Emilio, por su orientación, apoyo continuo y compromiso con el desarrollo de este proyecto.

DISEÑO Y DESARROLLO DE UN SISTEMA DE PROCESADO DE IMÁGENES PARA LA DETERMINACIÓN DE INCIDENTES RELACIONADOS CON PRESENCIA HUMANA NO AUTORIZADA

Autor: Alamán San Martín, Alejandro.
Director: Domínguez Adán, Emilio Manuel.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Se presenta el diseño y desarrollo de un sistema automatizado de detección de presencia humana en videovigilancia mediante Deep Learning, combinando Transfer Learning y Fine Tuning. Los resultados muestran alta precisión y robustez, destacando Faster R-CNN como solución óptima por su equilibrio entre fiabilidad y tiempo de inferencia.

Palabras clave: Detección de objetos, videovigilancia, Deep Learning, Faster R-CNN, YOLO, Transfer Learning, Fine Tuning.

1. Introducción

El aumento de los sistemas de videovigilancia ha creado la necesidad de automatizar la supervisión para reducir la carga humana y aumentar la fiabilidad de las alertas. Sin embargo, la detección de presencia humana en escenarios reales presenta retos significativos debido a condiciones variables de iluminación, ángulos de cámara, compresión de vídeo y escasez de incidencias reales.

El estado del arte ha avanzado notablemente con técnicas de aprendizaje profundo (Deep Learning), especialmente mediante arquitecturas como Faster R-CNN y YOLO, que ofrecen enfoques complementarios: el primero destaca por su precisión y su arquitectura en dos fases (two-stage), mientras el segundo optimiza la velocidad en un esquema de una fase (one-stage).

El objetivo de este proyecto es diseñar e implementar un sistema capaz de detectar automáticamente la presencia humana en imágenes de videovigilancia, reduciendo la carga manual de revisión y ofreciendo una herramienta fiable para personal de seguridad.

2. Metodología

El proyecto se estructuró en distintas fases, siguiendo una evolución lógica que respondió tanto a las necesidades técnicas como a los problemas concretos que fueron surgiendo durante el desarrollo.

El punto de partida fue la disponibilidad de vídeos reales de vigilancia con códec H264, que ofrecían el gran valor de reflejar situaciones auténticas y condiciones operativas reales. La primera tarea consistió en extraer tramas de estos vídeos, aplicando muestreo a intervalos configurables para controlar la frecuencia de captura y un filtro basado en diferencias de imagen para descartar tramas demasiado similares. Este diseño respondió a la necesidad de optimizar el conjunto de datos, evitando redundancias innecesarias y garantizando la diversidad de escenas.

Sin embargo, pronto se identificó un problema clave: la escasez de ejemplos con presencia humana suficiente para entrenar un detector robusto. Para superar esta limitación, se decidió incorporar otro conjunto de datos público ya etiquetado y con imágenes más variadas en contextos urbanos y exteriores ("INRIA Person Dataset"). Esta decisión permitió equilibrar la proporción de clases, enriquecer la variedad de condiciones de iluminación y ángulos, y mejorar la capacidad de generalización del modelo.

Otra etapa crítica fue el etiquetado manual de las tramas extraídas de los vídeos reales, que inicialmente carecían de anotaciones. Al principio se eligió la herramienta de etiquetado de imágenes "labelImg" por su fácil implementación, pero tras errores de cierres inesperados (posiblemente por incompatibilidades) y no poder continuar, se optó por sustituirla por "CVAT", otra herramienta potente y flexible con capacidad para gestionar proyectos colaborativos de etiquetado y la posibilidad de exportar en formato PASCAL VOC (XML), garantizando la compatibilidad y la coherencia con el ya mencionado conjunto de datos público complementario. Posteriormente, se diseñó e implementó una conversión automática al formato YOLO (necesario para uno de los modelos utilizados), adaptando la estructura de carpetas y las etiquetas para integrarlas con la librería Ultralytics de forma sencilla y reutilizable.

El entrenamiento también implicó decisiones estratégicas. Se optó por explorar dos enfoques complementarios: Faster R-CNN, con su arquitectura en dos fases (two-stage) y base ResNet50 (backbone) preentrenado en el conjunto de datos "COCO", para priorizar la precisión y reducir falsos positivos; y YOLOv8m, a través de Ultralytics, por su orientación a la detección en tiempo real y su rapidez de inferencia. Ambos modelos se entrenaron aplicando transferencia de conocimiento con refinamiento (Fine Tuning) sobre el conjunto de datos ya adaptado al problema concreto, utilizando tanto recursos locales como Google Colab Pro con GPU NVIDIA A100-SXM4-40GB, lo que permitió acelerar significativamente el proceso y experimentar con distintas configuraciones de hiperparámetros.

En conjunto, la metodología no solo se diseñó como un flujo de trabajo técnico, sino como un proceso iterativo y reflexivo, ajustándose a los desafíos del problema y aprovechando los recursos disponibles de forma eficiente para construir un sistema robusto y aplicable a la detección de presencia humana en escenarios de videovigilancia real.

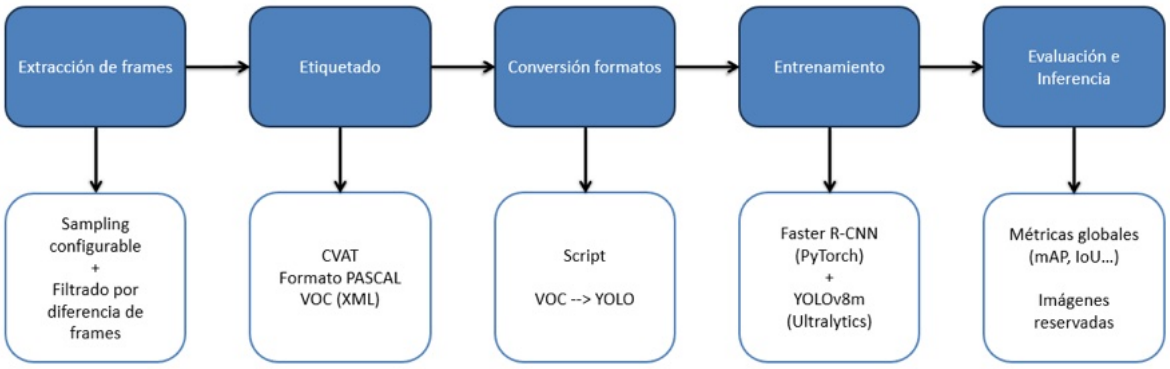

Ilustración 1. Esquema conceptual del pipeline del sistema (elaboración propia)

3. Resultados

Los resultados se evaluaron tanto cuantitativamente como cualitativamente.

Faster R-CNN refinado obtuvo un mAP / mAP50 de 0.9472, con un IoU promedio de 0.8105 y matrices de confusión con 90.1 % de TP y solo 1.8 % de FP y FN, reflejando un enfoque conservador y muy fiable. Su tiempo de inferencia medio fue de ~28.8 ms por imagen.

YOLOv8m refinado logró un mAP50 de 0.912 y mAP50-95 de 0.658, con Precision ~0.838 y Recall ~0.894. Su matriz de confusión mostró 73.0 % de TP pero un 23.3 % de FP, reflejando su perfil más agresivo. Su tiempo de inferencia fue muy bajo, ~4.6 ms por imagen, destacando para procesamiento en tiempo real.

Comparativa de métricas finales (con matriz de confusión en %)

Métrica	Faster R-CNN fine-tuned	YOLOv8m fine-tuned
mAP / mAP50	0.9472	0.912
TP (%)	90.1%	73.0%
FP (%)	1.8%	23.3%
FN (%)	1.8%	3.6%
TN (%)	6.3%	0.0%
Tiempo de inferencia (s/img)	0.0288	0.0046

Ilustración 2. Comparativa de métricas (elaboracion propia)

En la inferencia sobre el conjunto reservado de 6 imágenes (test_vigilancia) se compararon también las versiones preentrenadas y refinadas, mostrando mejoras claras en las cajas delimitadoras (bounding boxes) y reducción de falsos positivos en los modelos afinados.

Ilustración 3. Inferencia Faster R-CNN 5 (elaboración propia)

4. Conclusiones

El proyecto desarrollado demuestra de forma clara la viabilidad y efectividad de implementar un sistema de detección automatizada de presencia humana en videovigilancia utilizando técnicas avanzadas de aprendizaje profundo (Deep Learning) y transferencia de conocimiento (Transfer Learning). A través de un flujo de trabajo cuidadosamente estructurado, se han integrado datos reales procedentes de sistemas de videovigilancia, gestionando los retos propios de la calidad de compresión, la variabilidad de escenas y la escasez de incidencias etiquetadas, mediante la fusión con conjuntos de datos públicos y un proceso de etiquetado manual meticuloso.

El uso de refinamiento (Fine Tuning) sobre modelos pre-entrenados ha permitido adaptar arquitecturas del estado del arte a un dominio específico y realista, logrando una personalización que conserva la potencia de los modelos originales mientras se ajusta a las características y limitaciones de los datos reales. Esta estrategia ha resultado ser altamente eficiente en términos de desarrollo, evitando la necesidad de grandes volúmenes de datos completamente propios y reduciendo costes computacionales.

El análisis exhaustivo de los resultados cuantitativos y cualitativos respalda la elección final de Faster R-CNN refinado como la solución más adecuada para el objetivo planteado. Este modelo ha demostrado una precisión muy elevada, un IoU promedio alto y una matriz de confusión con muy bajo porcentaje de falsos positivos y negativos, características críticas para sistemas de supervisión centralizada donde el coste de una falsa alarma o de una omisión es elevado. Su capacidad para mantener un rendimiento consistente incluso con imágenes comprimidas o de calidad reducida lo convierte en una herramienta fiable para entornos reales de videovigilancia.

Por su parte, YOLOv8m refinado ha mostrado un rendimiento excepcional en velocidad, con tiempos de inferencia seis veces menores y una capacidad para procesar imágenes en tiempo real que lo hace muy atractivo para entornos distribuidos o dispositivos con capacidad de computación en el borde (edge computing). Sin embargo, su mayor porcentaje de falsos positivos indica la necesidad de estrategias adicionales de filtrado o validación para su uso en escenarios donde la fiabilidad de cada alerta sea prioritaria.

Estas conclusiones resaltan la importancia de adaptar el modelo elegido a los requisitos específicos del caso de uso. Mientras Faster R-CNN resulta óptimo para entornos centralizados con supervisión humana limitada, donde la calidad de la alerta es fundamental, YOLOv8m se presenta como una alternativa válida para sistemas que priorizan la cobertura y la rapidez.

En conjunto, el proyecto no solo valida técnicamente el enfoque propuesto, sino que ofrece un camino metodológico claro y reproducible para el diseño de sistemas similares. Al integrar datos reales, etiquetado manual, conversión de formatos, uso de plataformas consolidadas y técnicas de transferencia de conocimiento, se construyó una solución robusta, flexible y fundamentada; sentando bases sólidas para futuras extensiones, como la adaptación a otros escenarios, la inclusión de nuevas clases de objetos o la optimización para dispositivos con recursos limitados.

DESIGN AND DEVELOPMENT OF AN IMAGE PROCESSING SYSTEM FOR IDENTIFYING INCIDENTS INVOLVING UNAUTHORIZED HUMAN PRESENCE

Author: Alamán San Martín, Alejandro.
Supervisor: Domínguez Adán, Emilio Manuel.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This work presents the design and development of an automated human presence detection system for video surveillance using Deep Learning, combining Transfer Learning and Fine Tuning. The results show high precision and robustness, with Faster R-CNN standing out as the optimal solution for its balance between reliability and inference time.

Keywords: Object detection, video surveillance, Deep Learning, Faster R-CNN, YOLO, Transfer Learning, Fine Tuning.

1. Introduction

The increase in video surveillance systems has created the need to automate monitoring in order to reduce human workload and improve the reliability of alerts. However, detecting human presence in real-world scenarios poses significant challenges due to variable lighting conditions, camera angles, video compression, and the scarcity of real incidents.

The state of the art has advanced notably with Deep Learning techniques, particularly architectures such as Faster R-CNN and YOLO, which offer complementary approaches: the former stands out for its precision and two-stage architecture, while the latter optimizes speed with a one-stage design.

The objective of this project is to design and implement a system capable of automatically detecting human presence in video surveillance images, reducing manual review workload and providing a reliable tool for security personnel.

2. Metodology

The project was structured in several phases, following a logical progression that responded both to technical requirements and to the specific challenges that emerged during development.

The starting point was the availability of real surveillance videos encoded in H264, offering significant value by reflecting authentic situations and realistic operational conditions. The first task consisted of extracting frames from these videos, applying configurable sampling intervals to control capture frequency, and a filter based on image differences to discard frames that were too similar. This design aimed to optimize the dataset, avoiding unnecessary redundancy and ensuring scene diversity.

However, a key issue was soon identified: the scarcity of examples with sufficient human presence to train a robust detector. To address this limitation, the INRIA Person Dataset,

a publicly available and pre-annotated dataset with more varied images in urban and outdoor contexts, was incorporated. This decision helped balance class distribution, enrich the variety of lighting conditions and camera angles, and improve the model's capacity for generalization.

Another critical step was the manual annotation of the extracted frames, which initially lacked any labels. At first, the labelImg annotation tool was chosen for its easy setup, but after experiencing unexpected crashes (possibly due to compatibility issues) that prevented further work, it was decided to replace it with CVAT, a more powerful and flexible tool for managing collaborative annotation projects, with the ability to export in PASCAL VOC (XML) format to ensure compatibility and consistency with the complementary public dataset. Later, an automatic conversion to the YOLO format was designed and implemented, adapting the folder structure and labels to integrate them seamlessly and reusably with the Ultralytics library.

The training phase also involved strategic decisions. Two complementary approaches were selected: Faster R-CNN, with its two-stage architecture and ResNet50 backbone pre-trained on COCO, to prioritize precision and reduce false positives; and YOLOv8m via Ultralytics, for its focus on real-time detection and fast inference. Both models were trained using transfer learning with fine-tuning on the dataset already adapted to the specific problem, leveraging both local resources and Google Colab Pro with NVIDIA A100-SXM4-40GB GPU, which significantly accelerated the process and allowed experimentation with different hyperparameter configurations.

Overall, the methodology was designed not only as a technical pipeline, but as an iterative and reflective process, adapting to the challenges of the problem and efficiently leveraging available resources to build a robust system suitable for detecting human presence in real-world video surveillance scenarios.

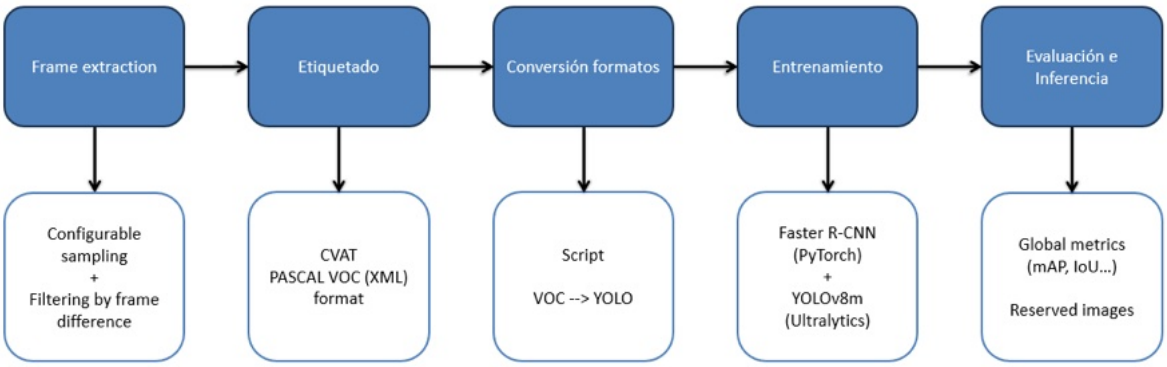

Figure 1. Conceptual diagram of the system pipeline (own elaboration)

3. Results

The results were evaluated both quantitatively and qualitatively.

Faster R-CNN fine-tuned achieved a mAP / mAP50 of 0.9472, with an average IoU of 0.8105 and confusion matrices showing 90.1 % TP and only 1.8 % FP and FN, reflecting a conservative and highly reliable approach. Its average inference time was approximately 28.8 ms per image.

Comparativa de métricas finales (con matriz de confusión en %)

Métrica	Faster R-CNN fine-tuned	YOLOv8m fine-tuned
mAP / mAP50	0.9472	0.912
TP (%)	90.1%	73.0%
FP (%)	1.8%	23.3%
FN (%)	1.8%	3.6%
TN (%)	6.3%	0.0%
Tiempo de inferencia (s/img)	0.0288	0.0046

Figure 2. Comparison of metrics (own elaboration)

YOLOv8m fine-tuned achieved a mAP50 of 0.912 and mAP50-95 of 0.658, with Precision ~0.838 and Recall ~0.894. Its confusion matrix revealed 73.0 % TP but 23.3 % FP, highlighting its more aggressive detection profile. Its inference time was very low, approximately 4.6 ms per image, making it especially suitable for real-time processing.

Inference on the reserved set of 6 images (test_vigilancia) also compared pre-trained and fine-tuned versions, showing clear improvements in bounding box accuracy and a reduction in false positives in the fine-tuned models.

Figure 3. Faster R-CNN inference 5 (own elaboration)

4. Conclusions

The developed project clearly demonstrates the feasibility and effectiveness of implementing an automated human presence detection system in video surveillance using advanced Deep Learning and Transfer Learning techniques. Through a carefully structured workflow, real data from actual surveillance systems were integrated, addressing challenges such as compression quality, scene variability, and the scarcity of labeled incidents by combining them with public datasets and a meticulous manual annotation process.

The use of fine-tuning on pre-trained models made it possible to adapt state-of-the-art architectures to a specific and realistic domain, achieving a level of customization that retains the power of the original models while tailoring them to the characteristics and limitations of real data. This strategy proved highly efficient from a development standpoint, avoiding the need for large volumes of fully proprietary data and reducing computational costs.

A thorough analysis of quantitative and qualitative results supports the final choice of the fine-tuned Faster R-CNN as the most suitable solution for the project's objectives. This model demonstrated very high precision, a high average IoU, and a confusion matrix with a very low percentage of false positives and false negatives—critical characteristics for centralized monitoring systems where the cost of a false alarm or omission is high. Its ability to maintain consistent performance even with compressed or lower-quality images makes it a reliable tool for real-world surveillance environments.

Meanwhile, the fine-tuned YOLOv8m showed exceptional speed performance, with inference times approximately six times faster and the ability to process images in real time, making it highly attractive for distributed environments or devices with edge computing capabilities. However, its higher rate of false positives indicates the need for additional filtering or validation strategies in scenarios where the reliability of each alert is a priority.

These conclusions highlight the importance of adapting the chosen model to the specific requirements of the use case. While Faster R-CNN is optimal for centralized environments with limited human supervision, where alert quality is essential, YOLOv8m emerges as a valid alternative for systems that prioritize coverage and speed.

Overall, the project not only technically validates the proposed approach but also offers a clear and reproducible methodological path for designing similar systems. By integrating real data, manual annotation, format conversion, the use of consolidated frameworks, and Transfer Learning techniques, a robust, flexible, and well-founded solution was built, laying solid groundwork for future extensions such as adaptation to other scenarios, the inclusion of new object classes, or optimization for resource-constrained devices.

Índice de la memoria

Capítulo 1. Introducción 5

1.1 Motivación del proyecto 5
1.2 Descripción del problema 6
1.3 Objetivos 7
1.4 Metodología 8

Capítulo 2. Estado del arte 10

2.1 Computer Vision: evolución y herramientas clave 10
2.2 Machine Learning & Deep Learning 12
2.3 Redes Neuronales Convolucionales (CNNs) 14
2.4 Modelos clave para detección de objetos 17
2.5 Estrategias de aprendizaje automático 20
2.6 Herramientas y entornos modernos 22
2.7 Soluciones comerciales y seguridad 24
2.8 Desafíos actuales en la detección automatizada 26

Capítulo 3. Tecnologías utilizadas 29

3.1 Framework Principal: PyTorch 29
3.2 Librerías auxiliares y procesamiento de imágenes 30
3.3 Modelos de Detección Utilizados 33
3.3.1 Faster R-CNN 34
3.3.2 YOLOv8 37
3.4 Herramientas de etiquetado de datos 40
3.5 Plataformas y recursos de computación 42
3.5.1 Recursos locales con GPU: 42
3.5.2 Google Colab: 43

Capítulo 4. Desarrollo 45

4.1 Extracción de datos 45
4.2 Problemas iniciales y soluciones 48
4.3 Etiquetado de datos 50
4.4 División del dataset 52

4.5 Conversión del formato de anotación para YOLO 53
4.6 Entrenamiento del modelo Faster R-CNN 55
4.7 Entrenamiento del modelo YOLOv8m 59

Capítulo 5. Análisis de resultados 63

5.1 Métricas y matriz de confusión 63
5.1.1 Faster R-CNN 64
5.2 YOLOv8m 66
5.3 Comparación de resultados 67
5.4 Inferencia sobre conjunto reservado 69

Capítulo 6. Conclusiones 83

6.1 Conclusiones sobre la metodología 83
6.2 Conclusiones sobre los resultados 84
6.3 Recomendaciones para futuro 86

Capítulo 7. Bibliografía 87

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS 88

ANEXO II: Librerías utilizadas 90

Índice de figuras

Ilustración 1. Esquema conceptual del pipeline del sistema (elaboración propia) 10
Ilustración 2. Comparativa de métricas (elaboracion propia) 11
Ilustración 3. Inferencia Faster R-CNN 5 (elaboración propia) 12
Ilustración 4. Machine Learning VS Deep Learning (medium.com/data-science) 13
Ilustración 5. Aplicación de un kernel en convolución (IBM) 15
Ilustración 6. Estructura típica de CNN con cabeza de clasificación (developersbreach.com) 16
Ilustración 7. One stage VS two stage object detection models (ResearchGate) 19
Ilustración 8. Transfer Learning // Fine Tuning (elaboración propia) 21
Ilustración 9. Arquitectura de Faster R-CNN (medium.com) 36
Ilustración 10. YOLO grid-based object detection framework (Abdussalam Ali Ahmed) 38
Ilustración 11. Comparación modelos y tamaños (Ultralytics) 39
Ilustración 12. Interfaz de labelImg (elaboración propia) 41
Ilustración 13. Interfaz online de CVAT (elaboración propia) 42
Ilustración 14. Ejemplo videovigilancia (elaboración propia) 45
Ilustración 15. Fragmento de salida extrayendo frames (elaboración propia) 47
Ilustración 17. Ejemplo formato de anotación YOLO (elaboración propia) 54
Ilustración 20. Train Loss VS Val Loss (elaboración propia) 59
Ilustración 21. Best yolov8m metrics (elaboración propia) 61
Ilustración 22. Matriz de confusión Faster R-CNN (elaboración propia) 65
Ilustración 23. Matriz de confusión YOLOv8m (elaboración propia) 66
Ilustración 24. Comparativa de métricas (elaboración propia) 68
Ilustración 25. Inferencia Faster R-CNN preentrenado 1 (elaboración propia) 70
Ilustración 26. Inferencia Faster R-CNN preentrenado 2 (elaboración propia) 70
Ilustración 27. Inferencia Faster R-CNN preentrenado 3 (elaboración propia) 71
Ilustración 28. Inferencia Faster R-CNN preentrenado 4 (elaboración propia) 71
Ilustración 29. Inferencia Faster R-CNN preentrenado 5 (elaboración propia) 72
Ilustración 30. Inferencia Faster R-CNN preentrenado 6 (elaboración propia) 72

Ilustración 31. Inferencia Faster R-CNN refinado 5 (elaboración propia)..... 73
Ilustración 32. Inferencia Faster R-CNN refinado 4 (elaboración propia)..... 74
Ilustración 33. Inferencia Faster R-CNN refinado 3 (elaboración propia)..... 74
Ilustración 34. Inferencia YOLOv8m preentrenado 1 (elaboración propia) 75
Ilustración 35. Inferencia YOLOv8m preentrenado 2 (elaboración propia) 76
Ilustración 36. Inferencia YOLOv8m preentrenado 3 (elaboración propia) 76
Ilustración 37. Inferencia YOLOv8m preentrenado 4 (elaboración propia) 77
Ilustración 38. Inferencia YOLOv8m preentrenado 5 (elaboración propia) 77
Ilustración 39. Inferencia YOLOv8m preentrenado 6 (elaboración propia) 78
Ilustración 40. Inferencia YOLOv8m refinado 1 (elaboración propia)..... 79
Ilustración 41. Inferencia YOLOv8m refinado 2 (elaboración propia)..... 79
Ilustración 42. Inferencia YOLOv8m refinado 3 (elaboración propia)..... 80
Ilustración 43. Inferencia YOLOv8m refinado 4 (elaboración propia)..... 80
Ilustración 44. Inferencia YOLOv8m refinado 5 (elaboración propia)..... 81
Ilustración 45. Inferencia YOLOv8m refinado 6 (elaboración propia)..... 81

Capítulo 1. INTRODUCCIÓN

1.1 MOTIVACIÓN DEL PROYECTO

La seguridad constituye un elemento esencial en el desarrollo de cualquier sociedad moderna, siendo un factor determinante para la protección de personas, activos e infraestructuras. En entornos industriales, empresariales o residenciales, la prevención de intrusiones o accesos no autorizados resulta clave para minimizar riesgos, garantizar la continuidad de las operaciones y proteger la integridad de los bienes.

En este contexto, los sistemas de videovigilancia se han consolidado como una herramienta indispensable para la supervisión continua y la detección temprana de incidentes. Sin embargo, a pesar de su amplia adopción, estos sistemas suelen presentar limitaciones importantes en su modalidad tradicional. La supervisión de imágenes o grabaciones generalmente requiere una intervención manual intensiva, obligando al personal de seguridad a revisar horas de vídeo para detectar eventos relevantes. Esta carga de trabajo elevada no solo conlleva costes económicos significativos, sino que también puede derivar en una menor eficiencia operativa y en la posibilidad de pasar por alto incidentes críticos debido al cansancio o la distracción.

La automatización de estas tareas repetitivas y sistemáticas representa, por tanto, una oportunidad relevante para mejorar la calidad y la eficacia de los sistemas de seguridad. En este marco, las tecnologías de visión artificial y aprendizaje automático surgen como alternativas especialmente prometedoras, al permitir la implementación de sistemas inteligentes capaces de analizar imágenes de forma autónoma, identificar patrones de interés y emitir alertas en tiempo real. Dentro de los sistemas de vigilancia que incorporan estas mejoras, siguen teniendo limitaciones y margen de error subóptimos. Los principales problemas engloban una gran cantidad de alarmas detectadas que, dentro de estas, muchas son falsas alarmas causadas por falsos positivos.

Este proyecto nace de la motivación por explorar, aprender y aplicar estas tecnologías de forma práctica y realista, con el objetivo de diseñar un sistema que asista al personal de seguridad reduciendo la carga de supervisión manual y mejorando la capacidad de respuesta ante eventos críticos. El interés se fundamenta tanto en el potencial de impacto social y económico que estas soluciones pueden tener, como en el reto que implica integrar de forma eficaz los distintos componentes necesarios para su funcionamiento: procesamiento de imágenes, inteligencia artificial, diseño de flujos de trabajo y evaluación rigurosa de resultados.

Además, este trabajo supone una oportunidad formativa y profesional para profundizar en campos como la visión por computador (Computer Vision) y el aprendizaje profundo (Deep Learning), consolidando y ampliando conocimientos teóricos y prácticos adquiridos a lo largo del grado. La experiencia busca no solo resolver un problema concreto, sino también desarrollar habilidades transferibles y aplicables en múltiples contextos donde la automatización y el análisis inteligente de datos sean clave para la innovación.

1.2 DESCRIPCIÓN DEL PROBLEMA

El problema que se aborda en este proyecto se enmarca en la necesidad de reforzar la seguridad en entornos controlados mediante la detección automática de la presencia humana no autorizada. En la actualidad, la videovigilancia convencional depende casi por completo del análisis visual por parte de operadores humanos, quienes deben supervisar largas secuencias de vídeo para identificar accesos indebidos o actividades sospechosas.

Esta dependencia conlleva varios inconvenientes: en primer lugar, la gran cantidad de imágenes generadas dificulta la revisión exhaustiva y sistemática, especialmente en instalaciones con múltiples cámaras o vigilancia 24/7. En segundo lugar, la intervención humana es susceptible a errores, distracciones o fatiga, reduciendo la fiabilidad del sistema y aumentando el riesgo de que eventos críticos no sean detectados a tiempo. Finalmente, la necesidad de mantener personal dedicado a esta tarea implica un coste económico sostenido que muchas organizaciones buscan reducir.

El desafío radica en reemplazar o complementar la supervisión humana con un sistema automatizado capaz de analizar imágenes de videovigilancia y detectar la presencia de personas no autorizadas con un alto grado de fiabilidad. Este sistema debe ser suficientemente preciso para minimizar los falsos positivos (alertas innecesarias) y falsos negativos (incidentes no detectados), además de ser adaptable a distintos entornos y condiciones de iluminación, perspectiva o calidad de vídeo.

La solución que se pretende diseñar y desarrollar debe abordar también limitaciones prácticas habituales, como la limitación y calidad de datos específicos para cada escenario, la necesidad de etiquetado de estos, la integración de información de distintas fuentes o formatos y la exigencia de procesar imágenes con recursos computacionales limitados. Por todo ello, el problema no solo exige una solución técnica sólida, sino también un diseño cuidadoso que considere la viabilidad de su implementación en entornos reales.

1.3 OBJETIVOS

El objetivo general de este proyecto consiste en diseñar y desarrollar un sistema de visión artificial capaz de detectar la presencia humana no autorizada en imágenes de videovigilancia, con el fin de reducir la carga de revisión manual por parte del personal de seguridad y aumentar la eficacia en la detección de incidentes.

Para alcanzar este propósito, se plantean los siguientes objetivos específicos:

- Investigar y analizar las herramientas y plataformas disponibles para el procesamiento de imágenes y la visión artificial en Python, valorando sus ventajas y limitaciones.
- Estudiar técnicas de aprendizaje automático, con especial énfasis en transferencia de conocimiento (Transfer Learning y Fine Tuning), para aprovechar modelos pre-entrenados y adaptarlos a las necesidades del proyecto.

- Experimentar con arquitecturas de detección de objetos de última generación, como Faster R-CNN y YOLOv8, comparando su rendimiento en términos de precisión, velocidad y facilidad de implementación.

- Resolver los desafíos asociados al trabajo con datos reales de videovigilancia, incluyendo la gestión de limitaciones de volumen, calidad y variedad de datos, con el fin de garantizar un entrenamiento adecuado de los modelos.

- Evaluar de forma rigurosa el rendimiento del sistema utilizando métricas adecuadas para la tarea de detección de objetos, identificando fortalezas y áreas de mejora.

- Documentar y sistematizar el proceso completo de desarrollo para garantizar la reproducibilidad del trabajo y facilitar su potencial extensión o adaptación en el futuro.

Con estos objetivos se busca no solo resolver el problema específico de la detección automatizada de presencia humana, sino también sentar las bases para el diseño de sistemas más complejos y robustos que puedan aplicarse en otros escenarios de seguridad o análisis de vídeo.

1.4 METODOLOGÍA

El enfoque metodológico del proyecto se ha diseñado para garantizar un desarrollo estructurado, riguroso y alineado con las buenas prácticas de ingeniería. Para ello, se plantea una secuencia de etapas que permiten abordar progresivamente los distintos aspectos del problema, desde la exploración teórica inicial hasta la implementación y validación de la solución final.

En una primera fase se ha llevado a cabo una investigación exhaustiva de las herramientas y librerías relevantes, como OpenCV, PyTorch, TensorFlow/Keras y Ultralytics, analizando su idoneidad para el procesamiento de imágenes y el entrenamiento de modelos de detección

de objetos (lo veremos más adelante). Esta etapa sentará las bases para la selección informada de las tecnologías a emplear.

Posteriormente se profundiza en los conceptos de inteligencia artificial aplicados al procesamiento de imágenes, como transferencia de aprendizaje y refinamiento (Transfer Learning y Fine Tuning), así como en técnicas de regularización que permitan optimizar el entrenamiento. En paralelo, se exploraron flujos de trabajo específicos mediante pequeños subproyectos o pruebas controladas que permitieran validar el enfoque y afinar la estrategia antes de abordar el desarrollo completo.

La fase central del proyecto consiste en el desarrollo del sistema de detección automática, que incluye la preparación y adaptación del conjunto de datos (extracción y filtrado de tramas, etiquetado y conversión de formatos), la personalización de modelos pre-entrenados mediante las técnicas mencionadas (Transfer Learning y Fine Tuning), y el ajuste de parámetros para optimizar su rendimiento. Esta fase central es la que deriva como mejor solución factible a desarrollar una vez son estudiadas los diferentes enfoques y herramientas actuales (se detalla en el siguiente capítulo).

Finalmente, se incorporan mecanismos de evaluación cuantitativa basados en métricas específicas de la tarea (como precisión o mAP), junto con una validación manual sobre un conjunto independiente de imágenes para verificar la robustez de la solución.

Como complemento al desarrollo técnico, se incluye un análisis crítico de los resultados obtenidos, acompañado de propuestas de mejora y distintas posibilidades de ampliación o adaptación del sistema. Este añadido busca aportar una visión más completa y rigurosa, orientada a facilitar la extensión del trabajo en futuros proyectos o su adaptación a problemáticas específicas y entornos más complejos.

Capítulo 2. ESTADO DEL ARTE

La revisión del estado del arte constituye un paso fundamental en el desarrollo de cualquier proyecto de ingeniería. Su propósito es analizar las soluciones existentes, comprender los avances tecnológicos y teóricos que se han producido en el área, e identificar tanto las oportunidades como las limitaciones presentes en el panorama actual. La evolución de los sistemas de seguridad ha ido de la mano de los avances en visión por computador, aprendizaje automático y aprendizaje profundo, disciplinas que han transformado radicalmente las capacidades de análisis de imágenes y vídeo.

Este capítulo presenta un recorrido estructurado por los principales enfoques, herramientas y técnicas utilizadas en el análisis automatizado de imágenes para tareas de seguridad. El objetivo es ofrecer un marco conceptual amplio que permita comprender las decisiones adoptadas en este proyecto, así como situar el desarrollo realizado en el contexto de la evolución tecnológica más general.

2.1 COMPUTER VISION: EVOLUCIÓN Y HERRAMIENTAS CLAVE

Visión por computador (Computer Vision) es la rama de la inteligencia artificial dedicada a dotar a las máquinas de la capacidad de interpretar imágenes y vídeos de forma similar a como lo haría un ser humano. Su desarrollo ha estado marcado por una evolución constante, desde enfoques basados en algoritmos heurísticos hasta sistemas avanzados impulsados por técnicas de aprendizaje automático y profundo.

En sus etapas iniciales, el campo se basaba principalmente en la definición manual de características y en la aplicación de transformaciones geométricas, filtros y operaciones matemáticas relativamente simples. Algoritmos como la detección de bordes, la transformada de Hough o descriptores locales como “SIFT” o “HOG” eran las herramientas más habituales para extraer información de las imágenes. Estas técnicas dependían de reglas explícitas programadas por expertos, permitiendo resolver tareas como la detección de

formas geométricas, el reconocimiento de patrones sencillos o el seguimiento de objetos en movimiento.

El avance en la capacidad computacional y la disponibilidad de grandes volúmenes de datos ha propiciado un cambio de paradigma hacia métodos basados en aprendizaje automático. En lugar de diseñar manualmente cada característica relevante, los modelos de aprendizaje automático (Machine Learning) permiten aprender patrones directamente a partir de los datos, mejorando la capacidad de generalización y reduciendo la necesidad de ingeniería manual intensiva.

Entre las herramientas más influyentes en este campo destaca OpenCV (Open Source Computer Vision Library), una biblioteca de código abierto que se ha convertido en un estándar de facto para las aplicaciones clásicas de Computer Vision. OpenCV ofrece un conjunto amplio de funciones para procesamiento de imágenes, detección de movimiento, transformación geométrica, filtrado y análisis morfológico, lo que la hace especialmente útil en etapas de preprocesamiento o en sistemas híbridos que combinan técnicas tradicionales con modelos de aprendizaje automático.

Otras librerías como PIL (Python Imaging Library) y su sucesor Pillow han facilitado la manipulación básica de imágenes, permitiendo operaciones como redimensionado, recorte o conversión de formatos, que resultan imprescindibles en flujos de trabajo más amplios. Estas herramientas siguen siendo relevantes hoy en día como complemento a enfoques más avanzados, especialmente para preparar los datos antes de su análisis mediante modelos de aprendizaje.

El salto más significativo en la evolución de Computer Vision se ha producido con la adopción generalizada del aprendizaje profundo (Deep Learning), que ha transformado radicalmente las capacidades de los sistemas de análisis de imágenes. El entrenamiento de redes neuronales profundas, especialmente redes convolucionales, ha permitido automatizar la extracción de características de forma jerárquica y aprender representaciones complejas directamente a partir de los datos, superando así las limitaciones de los enfoques basados en reglas manuales.

2.2 MACHINE LEARNING & DEEP LEARNING

El uso del aprendizaje automático (Machine Learning) en el análisis de imágenes supuso un cambio de paradigma respecto a los enfoques basados en reglas heurísticas y extracción manual de características. En lugar de programar de forma explícita qué patrones buscar en los datos, Machine Learning permite que los modelos aprendan de forma automática a partir de ejemplos etiquetados, generando funciones de decisión capaces de generalizar a datos nuevos.

En aplicaciones de imágenes, los modelos clásicos de Machine Learning empleaban aún etapas de extracción de características previamente diseñadas. Por ejemplo, se calculaban descriptores como "HOG" o "SIFT" para representar la información visual de forma más estructurada y, sobre esas características, se entrenaban clasificadores como Máquinas de Vectores de Soporte (SVM) o bosques aleatorios (Random Forest). Este enfoque ya suponía una mejora considerable respecto a las reglas puramente heurísticas, pero seguía limitado por la calidad de las características manuales y por la dificultad de capturar información compleja y jerárquica.

El gran salto se produjo con la adopción generalizada del aprendizaje profundo (Deep Learning), y en particular con las Redes Neuronales Convolucionales (CNN), que permitieron automatizar la extracción de características relevantes directamente a partir de los datos brutos. Este enfoque revolucionó la visión por computador al eliminar la necesidad de ingeniería manual intensiva en la definición de descriptores, permitiendo aprender representaciones jerárquicas que capturan desde bordes y texturas simples hasta formas y patrones complejos (*Ilustración 4*).

El entrenamiento de modelos de Deep Learning se basa en la optimización de funciones de pérdida sobre grandes conjuntos de datos, ajustando millones de parámetros internos mediante algoritmos como el Descenso de Gradiente Estocástico (SGD) y variantes más avanzadas como Adam o RMSProp. Este proceso requiere recursos computacionales

significativos, especialmente para tareas de análisis de imágenes, donde la información es de alta dimensionalidad.

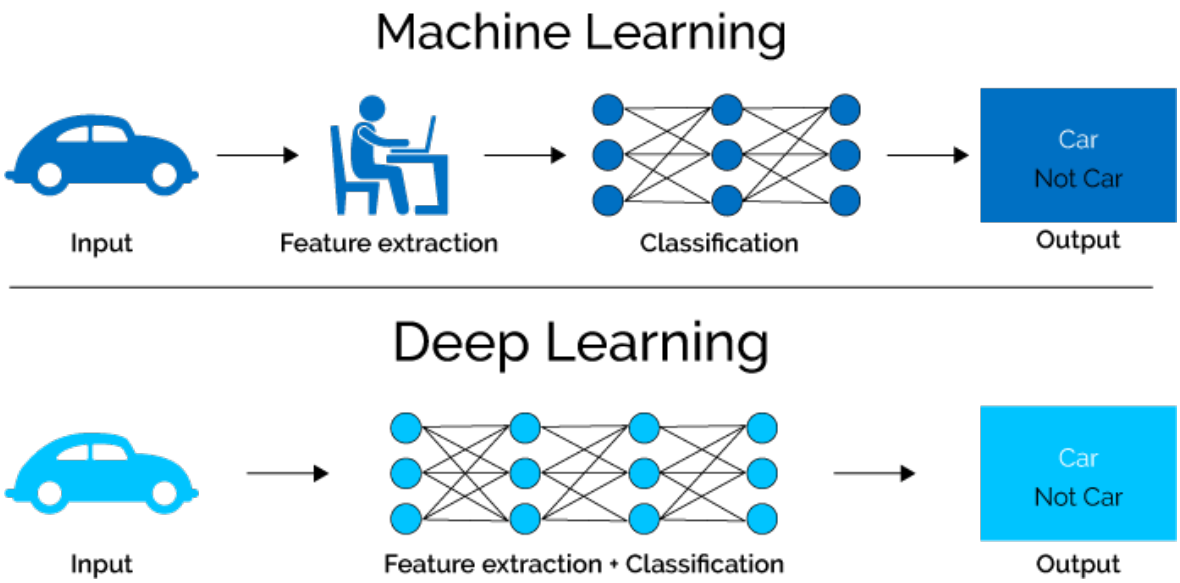

Ilustración 4. Machine Learning VS Deep Learning (medium.com/data-science)

A nivel de estructura general, un flujo de trabajo típico de Machine Learning o Deep Learning aplicado a imágenes incluye varias etapas:

- Adquisición y recopilación de datos.
- Preprocesamiento (normalización, redimensionado, limpieza de datos).
- División en conjuntos de entrenamiento, validación y test.
- Diseño o selección del modelo.
- Entrenamiento supervisado con retropropagación (Backpropagation).
- Evaluación del rendimiento mediante métricas específicas.
- Ajuste de hiperparámetros y regularización para evitar sobreajuste (overfitting).
- Validación final y despliegue o integración en sistemas reales.

El éxito de Deep Learning en Computer Vision no solo se debe a la capacidad de extraer características automáticamente, sino también a la posibilidad de aprender representaciones

generales a partir de grandes conjuntos de datos públicos y adaptarlas a casos específicos mediante técnicas como Transfer Learning y Fine Tuning. Estas estrategias permiten reducir el coste de entrenamiento, mejorar la precisión en escenarios con menos datos y facilitar la adopción de soluciones avanzadas incluso en contextos con recursos limitados.

2.3 REDES NEURONALES CONVOLUCIONALES (CNNS)

Las Redes Neuronales Convolucionales (CNNs) constituyen la piedra angular del éxito actual de Deep Learning en Computer Vision. Su arquitectura específica ha permitido resolver con gran eficacia tareas de clasificación, detección y segmentación de imágenes, gracias a su capacidad para aprender automáticamente representaciones jerárquicas y espaciales de los datos.

La motivación principal detrás de las CNNs radica en el hecho de que las imágenes contienen estructuras locales altamente correlacionadas. A diferencia de las redes totalmente conectadas, donde cada neurona se conecta con todas las anteriores, las CNNs utilizan convoluciones locales que explotan la naturaleza espacial de los datos, reduciendo drásticamente el número de parámetros y facilitando la generalización.

La arquitectura típica de una CNN se compone de varias capas diferenciadas que trabajan de manera secuencial (*Ilustración 6*):

- Capas Convolucionales: aplican filtros o kernels sobre la imagen de entrada o sobre mapas de características intermedios (*Ilustración 5*). Estos filtros van recorriendo toda la imagen o mapa intermedio convolucionalmente y aprenden a detectar patrones locales como bordes, texturas o formas más complejas conforme se avanza en profundidad.

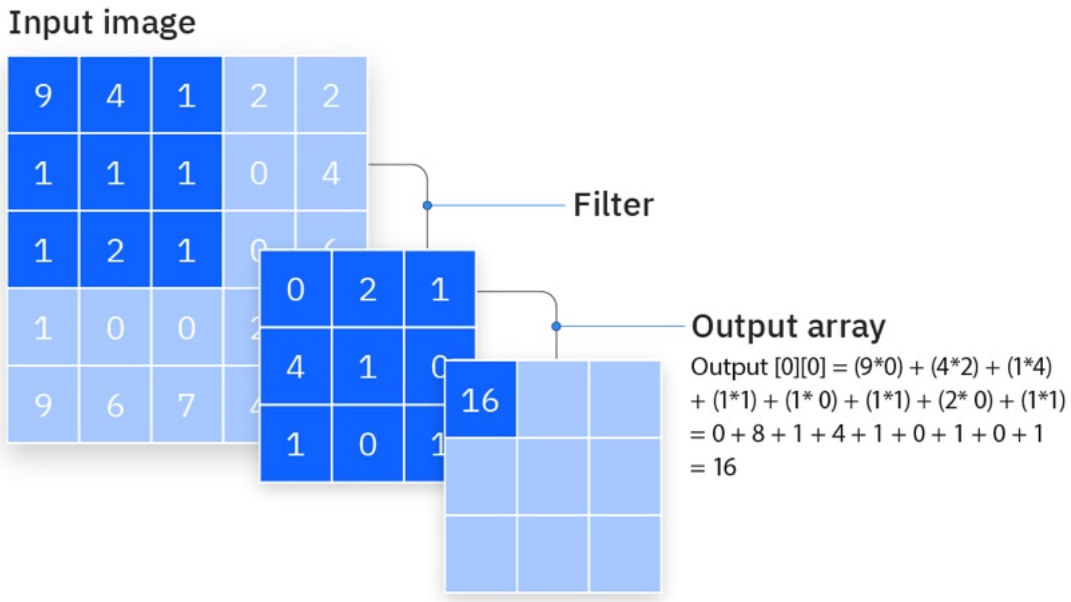

Ilustración 5. Aplicación de un kernel en convolución (IBM)

- Capas de Pooling: realizan operaciones de reducción de dimensionalidad, como Max Pooling o Average Pooling, que permiten condensar la información más relevante y aportar invariancia ante pequeñas transformaciones o desplazamientos.
- Capas de Activación: introducen no linealidades mediante funciones como ReLU (Rectified Linear Unit), lo que permite a la red aprender representaciones más complejas y no lineales de los datos.
- Capas de Normalización: conocidas como Batch Normalization, que estabilizan el entrenamiento al normalizar las salidas intermedias, acelerando la convergencia y mejorando la generalización.
- Capas de Flatten: transforman los mapas de características multidimensionales en un vector unidimensional, permitiendo conectar la salida de las capas convolucionales y de muestreo (pooling) con las capas completamente conectadas.

- Capas Completamente Conectadas (Fully Connected Layers): al final de la red, se suelen utilizar capas densas que combinan la información extraída para tomar decisiones de clasificación o regresión.

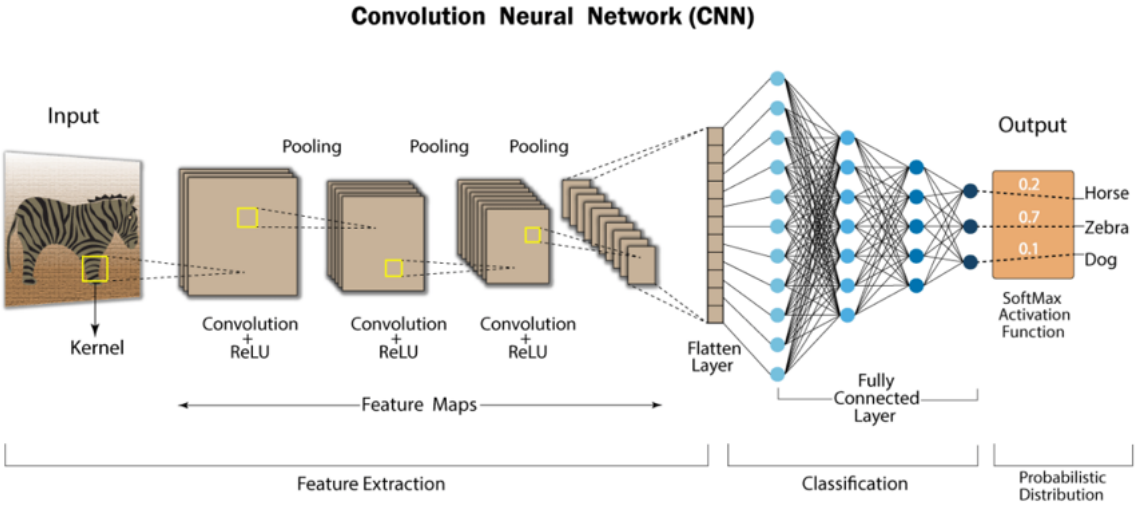

Ilustración 6. Estructura típica de CNN con cabeza de clasificación (developersbreach.com)

El aprendizaje en una CNN consiste en ajustar los parámetros de estos filtros y capas mediante "Backpropagation" (algoritmo de retropropagación que ajusta los pesos de la red neuronal en función del error obtenido), minimizando una función de pérdida definida para la tarea concreta. Durante el entrenamiento, la red aprende de forma jerárquica: las primeras capas capturan patrones simples como bordes o esquinas, mientras que las capas más profundas combinan estas características para reconocer formas complejas y estructuras semánticas de alto nivel.

Una característica destacable de las CNNs es su modularidad y adaptabilidad. Al ser capaces de aprender automáticamente representaciones relevantes, pueden aplicarse a una amplia variedad de problemas en Computer Vision, desde clasificación de imágenes hasta segmentación semántica o detección de objetos. Para estas últimas tareas, la arquitectura de la red suele extenderse mediante cabezas especializadas.

En el caso de la detección de objetos, las CNNs se utilizan para resolver dos subproblemas simultáneamente: la localización de las regiones de interés mediante cajas delimitadoras (bounding boxes) y la clasificación de los objetos contenidos en esas regiones. Las llamadas cabezas de detección añaden capas adicionales que predicen las coordenadas de las cajas y las probabilidades de pertenencia a cada clase.

La combinación de extracción automática de características y estas cabezas especializadas ha convertido a las CNNs en el estándar de facto para los sistemas modernos de detección de presencia humana en imágenes, como los que se abordarán en este proyecto.

2.4 MODELOS CLAVE PARA DETECCIÓN DE OBJETOS

La tarea de detección de objetos en imágenes requiere resolver simultáneamente dos problemas: la localización de los objetos mediante cajas delimitadoras (bounding boxes) y su clasificación en categorías específicas. A diferencia de la clasificación de imágenes convencional, donde se asigna una etiqueta única a toda la imagen, la detección implica analizar la estructura espacial y predecir la posición y clase de múltiples objetos dentro de la escena. Así, una vez se tiene proposiciones de regiones para las cajas y su clase específica asociada, todo debe quedar reflejado en los metadatos de la imagen, que suelen definirse como anotaciones (Annotations). Según el formato de estas anotaciones (VOC, YOLO…), dentro de los metadatos aparece un apartado para cada objeto con su nombre de clase y coordenadas que definen su caja (aparte de posibles atributos añadidos). Un ejemplo de esto es la siguiente anotación en formato PASCAL VOC, donde localizamos los campos descritos en `<name>…</name>` y `<bndbox>…</bndbox>` dentro de cada `<object>…</object>`:

```
<annotation>
 <folder>vigilancia</folder>
 <filename>vigilancia_000133.jpg</filename>
 <source>
  <database>Unknown</database>
  <annotation>Unknown</annotation>
  <image>Unknown</image>
 </source>
 <size>
  <width>2560</width>
  <height>1440</height>
```

```
  <depth/>
 </size>
 <segmented>0</segmented>
 <object>
  <name>person</name>
  <truncated>0</truncated>
  <occluded>0</occluded>
  <difficult>0</difficult>
  <bndbox>
   <xmin>1621.69</xmin>
   <ymin>525.43</ymin>
   <xmax>1693.59</xmax>
   <ymax>688.33</ymax>
  </bndbox>
  <attributes>
   <attribute>
    <name>rotation</name>
    <value>0.0</value>
   </attribute>
  </attributes>
 </object>
 <object>
  <name>person</name>
  <truncated>0</truncated>
  <occluded>0</occluded>
  <difficult>0</difficult>
  <bndbox>
   <xmin>1958.42</xmin>
   <ymin>524.52</ymin>
   <xmax>2022.12</xmax>
   <ymax>692.88</ymax>
  </bndbox>
  <attributes>
   <attribute>
    <name>rotation</name>
    <value>0.0</value>
   </attribute>
  </attributes>
 </object>
</annotation>
```

Para abordar este problema, se han desarrollado múltiples arquitecturas basadas en Redes Neuronales Convolucionales (CNNs) que han marcado hitos importantes en el campo de visión por computadora (Computer Vision). Entre los enfoques más influyentes se encuentran las familias de modelos de dos etapas (two-stage) y de una sola etapa (one-stage), cada una con sus características y ventajas (*Ilustración 7*).

Los modelos "two-stage", como R-CNN, Fast R-CNN y Faster R-CNN, dividen el problema en dos fases diferenciadas: primero generan propuestas de regiones de interés que podrían contener objetos y después refinan estas propuestas clasificándolas y ajustando sus coordenadas. Este enfoque suele destacar por su precisión, ya que la separación de tareas permite un análisis más detallado y controlado de cada región candidata. Faster R-CNN, en particular, introdujo mejoras significativas en velocidad y rendimiento, consolidándose como un estándar en muchos escenarios donde la precisión es prioritaria.

Por otro lado, los modelos "one-stage", como SSD, RetinaNet o la familia YOLO (You Only Look Once), abordan la detección de forma más directa. Dividen la imagen en una malla y predicen simultáneamente las clases y coordenadas de las cajas para cada celda. Este diseño integrado permite una mayor rapidez de inferencia, resultando especialmente adecuado para aplicaciones en tiempo real o con restricciones de hardware. En particular, las distintas versiones de YOLO han evolucionado hasta ofrecer un compromiso muy competitivo entre velocidad y precisión, convirtiéndose en una de las opciones más populares y versátiles en la práctica.

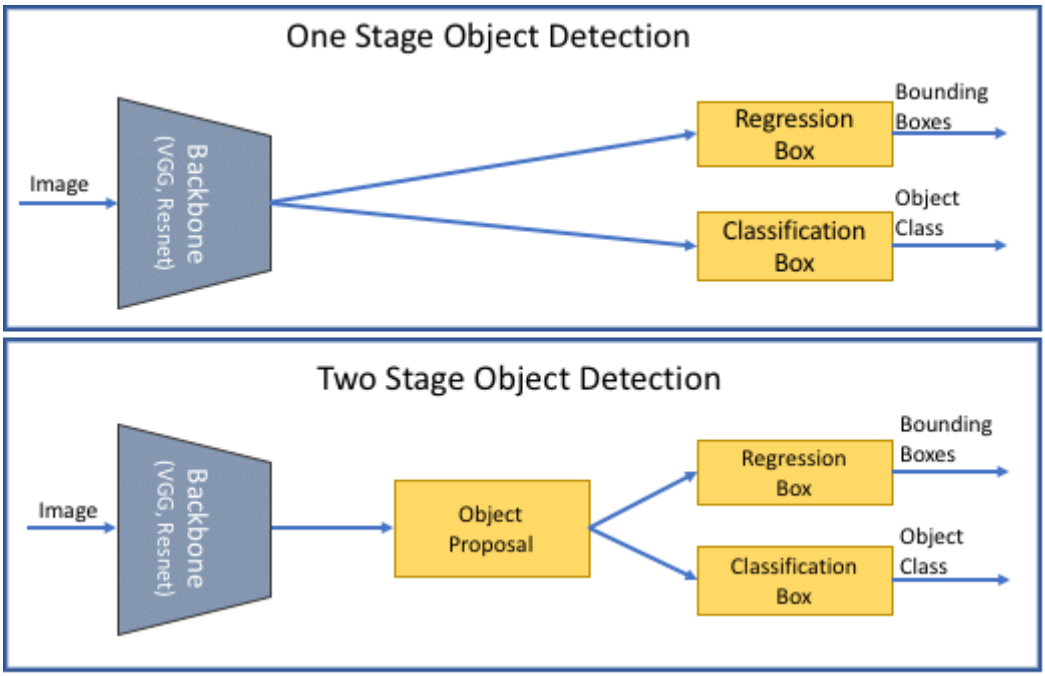

Ilustración 7. One stage VS two stage object detection models (ResearchGate)

Actualmente, la elección entre estos enfoques depende de los requisitos específicos del problema. En entornos de seguridad y videovigilancia, donde se busca detectar presencia humana no autorizada con alta fiabilidad, modelos como Faster R-CNN y las versiones más recientes de YOLO se posicionan como las mejores opciones disponibles. Ambos ofrecen la capacidad de aprovechar redes profundas preentrenadas y técnicas de transferencia de conocimiento y refinamiento (Transfer Learning y Fine Tuning) para adaptarse a distintos contextos, manteniendo un equilibrio entre precisión, velocidad y facilidad de implementación.

2.5 ESTRATEGIAS DE APRENDIZAJE AUTOMÁTICO

Transfer Learning y Fine Tuning son dos de las estrategias más relevantes y ampliamente utilizadas en el ámbito de aprendizaje profundo (Deep Learning) para visión por computador (Computer Vision). Estas técnicas permiten aprovechar el conocimiento aprendido por modelos previamente entrenados sobre grandes conjuntos de datos generales y adaptarlo de manera eficiente a tareas más específicas o dominios concretos.

En esencia, Transfer Learning consiste en reutilizar las representaciones y parámetros de una red entrenada sobre un problema diferente pero relacionado. En visión por computador, este enfoque se ha popularizado debido a la disponibilidad de modelos pre-entrenados sobre conjuntos de datos (comúnmente llamados "dataset") de gran escala como "ImageNet[1]", "COCO[2]" o "PASCAL VOC[3]", que contienen millones de imágenes y miles de categorías. Las capas iniciales de estas redes aprenden características generales de las imágenes, como bordes, texturas y formas básicas, que resultan útiles para múltiples tareas.

[1] ImageNet, [en línea], disponible en: https://www.image-net.org/ (consulta: 13/07/2025).
[2] COCO, [en línea], disponible en: https://cocodataset.org/ (consulta: 13/07/2025)
[3] Ultralytics dataset VOC, [en línea], disponible en: https://docs.ultralytics.com/es/datasets/detect/voc/ (consulta: 13/07/2025)

Fine Tuning complementa este enfoque al permitir ajustar de forma más precisa los parámetros del modelo preentrenado a la nueva tarea. Habitualmente, se congelan las primeras capas (que capturan características generales) y se reentrenan o ajustan las capas finales para que se especialicen en el conjunto de datos objetivo. Este procedimiento permite lograr resultados competitivos incluso cuando se dispone de un volumen limitado de datos específicos para la tarea concreta.

En resumen, Transfer Learning, como su propio nombre indica, es una técnica que "transfiere conocimiento" previamente aprendido, y Fine Tuning es una técnica de refinamiento del modelo preentrenado que entra dentro del Transfer Learning. Así, un modelo preentrenado puede reconocer ya mapas de características de su parte CNN, se puede cambiar su cabeza de clasificación (Fully Connected) y solo entrenar esta parte, congelando los pesos de toda la red convolucional, consiguiendo que use su conocimiento en la clasificación del caso concreto. A su vez, se pueden descongelar los pesos de capas de la parte convolucional para refinar también la extracción de características en tu entorno específico (*Ilustración 8*).

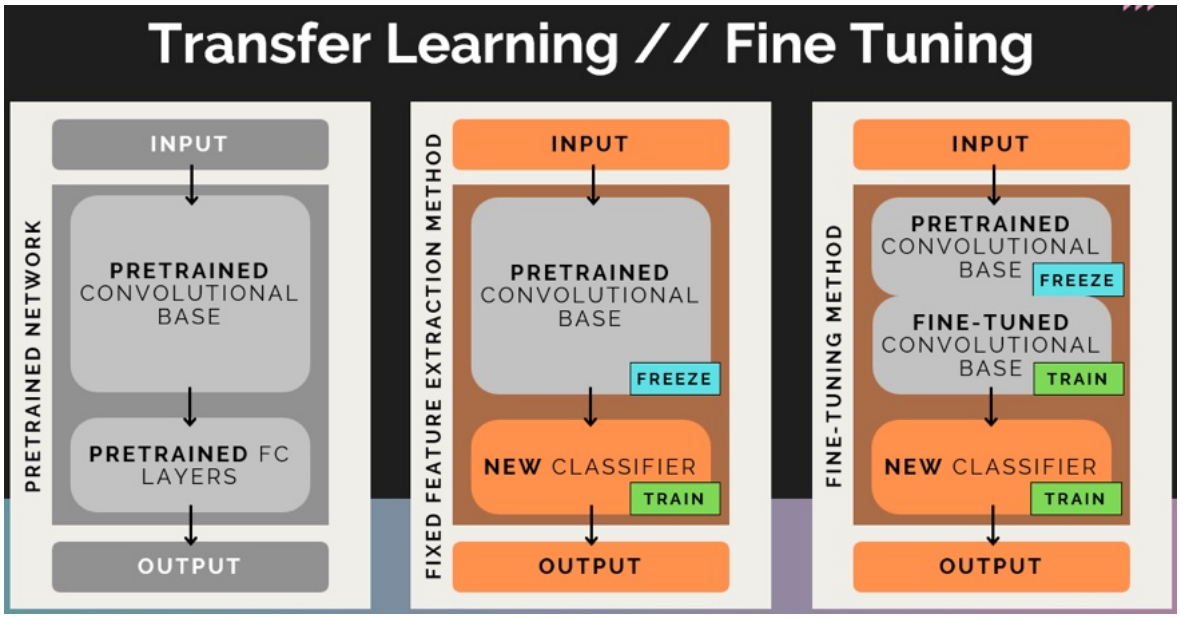

Ilustración 8. Transfer Learning // Fine Tuning (elaboración propia)

El uso de Transfer Learning y Fine Tuning resulta especialmente ventajoso en aplicaciones como la detección de objetos en imágenes de videovigilancia. El entrenamiento desde cero de modelos profundos exige grandes cantidades de datos etiquetados y recursos computacionales significativos. En cambio, aprovechar modelos pre-entrenados y adaptarlos mediante estas técnicas permite reducir drásticamente los requisitos de datos y tiempo de entrenamiento, manteniendo un alto nivel de precisión.

Además, estas estrategias son fundamentales para construir soluciones flexibles y personalizadas. Dado que existen arquitecturas y modelos pre-entrenados ampliamente validados para la detección de objetos (como Faster R-CNN o YOLO), Transfer Learning y Fine Tuning permiten adaptar estas soluciones consolidadas a casos de uso específicos, como la detección de presencia humana no autorizada en entornos de videovigilancia. Este enfoque maximiza el aprovechamiento de los avances más recientes en visión por computador y facilita el desarrollo de sistemas robustos y efectivos para escenarios del mundo real.

2.6 HERRAMIENTAS Y ENTORNOS MODERNOS

El desarrollo de soluciones avanzadas de visión por computador basadas en aprendizaje profundo ha sido posible gracias a la evolución de herramientas y entornos (frameworks) especializados que simplifican enormemente el diseño, entrenamiento y despliegue de modelos. Estas herramientas no solo ofrecen implementaciones optimizadas de las operaciones fundamentales de redes neuronales, sino que también proporcionan acceso a modelos pre-entrenados, datasets y utilidades para su personalización mediante técnicas ya mencionadas como Transfer Learning y Fine Tuning.

El origen de estos entornos se remonta a la necesidad de hacer más accesible y reproducible la investigación en el aprendizaje profundo. TensorFlow[4], lanzado en 2015 por Google

[4] TensorFlow, [en línea], disponible en: https://www.tensorflow.org/ (consulta: 13/07/2025).

Brain, nació como una evolución de herramientas internas (DistBelief[5]) para facilitar el desarrollo de redes neuronales profundas a gran escala, con énfasis en el despliegue en producción y el soporte multiplataforma. PyTorch[6], presentado por Facebook AI Research en 2016, surgió para ofrecer un enfoque más dinámico y flexible (define-by-run), especialmente orientado a la comunidad investigadora y educativa. Esta diferencia inicial en filosofía y público objetivo impulsó un desarrollo competitivo que ha enriquecido todo el ecosistema del aprendizaje profundo.

Estos entornos son los más influyentes y ampliamente utilizados. Ambos han democratizado el acceso al aprendizaje profundo, permitiendo a investigadores y profesionales desarrollar soluciones de alto rendimiento sin necesidad de implementar manualmente todos los algoritmos desde cero. TensorFlow ha destacado históricamente por su madurez en entornos de producción y despliegue, mientras que PyTorch ha ganado popularidad especialmente en la comunidad investigadora gracias a su flexibilidad, su naturaleza más dinámica y su facilidad de depuración y experimentación.

En el ámbito específico de visión por computador, estas plataformas ofrecen módulos y bibliotecas adicionales que contienen modelos pre-entrenados, arquitecturas populares y utilidades para carga y transformación de datos. Por ejemplo, "torchvision" en PyTorch incluye implementaciones optimizadas de arquitecturas como ResNet, Faster R-CNN y SSD, con pesos pre-entrenados en conjuntos de datos como COCO o ImageNet. Estas capacidades permiten a los desarrolladores aprovechar de forma sencilla los avances del estado del arte y adaptarlos a problemas concretos mediante Transfer Learning.

Además, han surgido ecosistemas y bibliotecas especializadas que simplifican aún más el entrenamiento y uso de modelos de detección de objetos. Ultralytics[7], por ejemplo, ha

[5] Dean, J. et al., "Large Scale Distributed Deep Networks", Advances in Neural Information Processing Systems 25 (NeurIPS 2012), [en línea], disponible en: https://static.googleusercontent.com/media/research.google.com/es//archive/large_deep_networks_nips2012.pdf (consulta: 13/07/2025).
[6] PyTorch, [en línea], disponible en: https://pytorch.org/ (consulta: 13/07/2025).
[7] Ultralytics, [en línea], disponible en: https://www.ultralytics.com/ (consulta: 13/07/2025)

facilitado el acceso a las versiones más recientes y eficientes de la familia YOLO (You Only Look Once), ofreciendo APIs y entornos de entrenamiento intuitivos, soporte para diferentes formatos de anotación y capacidades de exportación a diferentes plataformas.

La disponibilidad de estas herramientas modernas ha transformado radicalmente el desarrollo de soluciones personalizadas de visión por computador. En lugar de partir desde cero, es posible construir sistemas robustos y adaptados a casos de uso específicos (como la detección de presencia humana no autorizada en entornos de videovigilancia) partiendo de modelos consolidados y validando sus resultados con datos reales. Este enfoque no solo mejora la calidad y fiabilidad de las soluciones, sino que también reduce de manera significativa los costes de desarrollo y los requisitos de datos y computación, favoreciendo la adopción de la inteligencia artificial incluso en proyectos con recursos limitados y acelerando la transferencia de la investigación a aplicaciones reales.

2.7 SOLUCIONES COMERCIALES Y SEGURIDAD

El campo de la seguridad y la videovigilancia ha experimentado en los últimos años una profunda transformación gracias a la incorporación de técnicas avanzadas de Computer Vision e inteligencia artificial. Numerosas soluciones comerciales ya integran capacidades de análisis automático de imágenes y vídeo, ofreciendo funciones como detección de intrusiones, seguimiento de objetos, reconocimiento de matrículas o análisis de comportamiento.

Entre las soluciones más destacadas se encuentran los sistemas de cámaras inteligentes, que incorporan procesamiento en el borde (edge computing) para realizar detección y análisis en tiempo real sin necesidad de enviar todos los datos a servidores centrales. Estos dispositivos suelen utilizar modelos de Deep Learning optimizados para hardware limitado, lo que

permite su uso en entornos con restricciones de conectividad o latencia. Un ejemplo de estos modelos podría ser el MobileNet[8].

Por otro lado, existen plataformas integrales de videovigilancia basadas en la nube que ofrecen servicios avanzados de análisis mediante suscripciones, aprovechando la potencia de procesamiento centralizado y la capacidad de actualizar de forma continua sus algoritmos. Estas soluciones permiten a empresas y organizaciones acceder a análisis sofisticados sin tener que mantener la infraestructura necesaria localmente.

A pesar de sus ventajas, las soluciones comerciales suelen presentar limitaciones importantes desde el punto de vista de la personalización, el coste y la privacidad. Muchas de estas plataformas están diseñadas para escenarios genéricos y no permiten adaptar los modelos a las condiciones específicas de un entorno determinado, como la iluminación, el ángulo de las cámaras o el tipo de actividad esperada. Además, los costes de licencias, suscripciones y actualizaciones pueden suponer una barrera para instalaciones más pequeñas o presupuestos ajustados.

Desde el punto de vista de la privacidad, la adopción de sistemas automatizados de videovigilancia ha planteado preocupaciones regulatorias y sociales. La legislación europea, en particular el AI Act[9] en tramitación, busca garantizar un uso responsable y transparente de la inteligencia artificial, exigiendo que se evalúen los riesgos potenciales y se incorporen salvaguardas adecuadas. En este contexto, resulta especialmente relevante que el sistema propuesto en este proyecto no procese ni almacene datos personales (Personally Identifiable Information), ya que no realiza reconocimiento facial ni clasificación de identidades concretas. Su objetivo se limita a identificar la presencia genérica de personas mediante la

[8] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.", [en línea], disponible en: http://arxiv.org/abs/1704.04861 (consulta: 22/07/2025)
[9] EU Artificial Intelligence Act, [en línea], disponible en: https://artificialintelligenceact.eu/ (consulta:13/07/2025)

detección de siluetas y cajas delimitadoras, cumpliendo así con los requisitos de protección de datos personales y privacidad.

Estas limitaciones de las soluciones comerciales justifican la necesidad de investigar y desarrollar alternativas más flexibles y adaptables. Un sistema propio permite ajustar los modelos al escenario real, controlar el flujo de datos y garantizar el cumplimiento de la legislación vigente, maximizando tanto la efectividad como la responsabilidad en su uso.

2.8 DESAFÍOS ACTUALES EN LA DETECCIÓN AUTOMATIZADA

A pesar de los notables avances logrados en el ámbito de la detección automática de objetos mediante Deep Learning, existen aún múltiples desafíos que condicionan la adopción y el rendimiento de estas soluciones en entornos reales. Comprender estas limitaciones resulta esencial para diseñar sistemas más robustos y adaptados a casos concretos como la videovigilancia para la detección de presencia humana no autorizada.

Uno de los retos principales es la variabilidad de las condiciones ambientales. Las diferencias en iluminación (como escenas nocturnas o con sombras pronunciadas), las condiciones meteorológicas adversas, el movimiento de las cámaras o los cambios de ángulo, así como el envío de las imágenes si el procesamiento es centralizado, pueden afectar significativamente la calidad de las imágenes y la capacidad del modelo para detectar objetos con fiabilidad. Este problema exige entrenar modelos con datos que representen adecuadamente la diversidad de escenarios esperados o emplear técnicas de aumento de datos y adaptación de dominio.

Otro desafío relevante es la gestión de falsos positivos y falsos negativos. Un sistema de seguridad debe minimizar ambos tipos de error para resultar útil en la práctica. Los falsos positivos pueden generar alertas innecesarias que sobrecargan al personal y reducen la confianza en el sistema, mientras que los falsos negativos implican no detectar intrusiones reales, comprometiendo la seguridad del entorno. El diseño de la arquitectura, la calidad de

los datos y la definición de umbrales adecuados son factores críticos para equilibrar esta problemática.

La diversidad de escenarios y objetivos también plantea retos importantes. Un modelo entrenado en un entorno puede no generalizar bien a otros con características distintas. Por ejemplo, el mismo sistema puede enfrentarse a variaciones en el tipo de vestimenta, posturas, distancias a la cámara o patrones de movimiento. Este desafío subraya la importancia de contar con conjuntos de datos variados y de emplear técnicas como Transfer Learning y Fine Tuning para adaptar modelos pre-entrenados a contextos específicos.

Desde el punto de vista computacional, la complejidad de los modelos de aprendizaje profundo (Deep Learning) puede implicar costes elevados tanto en entrenamiento como en despliegue. En aplicaciones de videovigilancia en tiempo real, la necesidad de procesar múltiples flujos de vídeo de manera simultánea plantea retos específicos que requieren soluciones adaptadas.

Por un lado, el uso de unidades de procesamiento gráfico (GPU, Graphics Processing Units) permite acelerar de forma considerable el entrenamiento y la inferencia de modelos complejos gracias a su capacidad para realizar cálculos en paralelo. Su principal ventaja es el alto rendimiento para tareas de gran volumen o gran precisión, pero su inconveniente reside en el coste elevado del hardware y en el consumo energético, lo que limita su aplicación en dispositivos con recursos reducidos o en instalaciones distribuidas.

Por otro lado, el procesamiento en el borde (edge computing) consiste en ejecutar modelos directamente en dispositivos cercanos a la fuente de datos, como cámaras inteligentes o nodos locales. Su principal ventaja es la reducción de latencia y el menor consumo de ancho de banda al evitar transmitir vídeo en tiempo real a servidores centrales. Sin embargo, su limitación radica en la menor capacidad de cómputo disponible en estos dispositivos, lo que obliga a utilizar modelos más ligeros o simplificados y puede afectar a la precisión o el alcance de las detecciones.

Finalmente, es esencial considerar los aspectos relacionados con la protección de la privacidad y la legislación vigente. La implementación de sistemas automatizados de videovigilancia debe cumplir con marcos legales como el Reglamento General de Protección de Datos[10] (RGPD) y el futuro AI Act europeo[11]. En este sentido, el sistema propuesto en este proyecto se diseña para preservar la privacidad de los usuarios al no realizar reconocimiento facial ni almacenar datos personales (PII), sino limitándose a detectar la presencia genérica de personas mediante siluetas y cajas delimitadoras.

[10] Reglamento General de Protección de Datos, [en línea], disponible en: https://www.boe.es/doue/2016/119/L00001-00088.pdf (consulta: 13/07/2025)
[11] Ver nota 9.

Capítulo 3. TECNOLOGÍAS UTILIZADAS

Este capítulo describe en detalle las principales tecnologías empleadas en el proyecto, justificando su elección en función de sus características técnicas, su relevancia en el estado del arte y su idoneidad para los objetivos planteados. Se presentarán tanto los frameworks de Deep Learning utilizados para la construcción y entrenamiento de modelos, como las librerías de procesamiento de imágenes, las herramientas de anotación y etiquetado de datos, y las plataformas de computación empleadas para facilitar el desarrollo y validación del sistema.

3.1 FRAMEWORK PRINCIPAL: PYTORCH

El desarrollo de modelos avanzados de aprendizaje profundo para tareas de detección de objetos requiere un lenguaje de programación versátil y ampliamente respaldado. En este proyecto se ha elegido Python debido a su popularidad en el ámbito de la inteligencia artificial, su sintaxis clara y su extenso ecosistema de librerías especializadas. Su uso generalizado en la comunidad investigadora y profesional facilita la integración de múltiples herramientas y acelera el proceso de desarrollo y experimentación.

Como plataforma principal para la construcción, entrenamiento y evaluación de modelos de detección de objetos, se ha seleccionado PyTorch. Como ya se ha comentado, esta plataforma de código abierto fue desarrollada inicialmente por Facebook AI Research (FAIR) y presentada en 2016 como respuesta a la necesidad de herramientas más flexibles y accesibles para la investigación en aprendizaje profundo (Deep Learning). Desde entonces, PyTorch se ha consolidado como una de las herramientas más utilizadas en el ámbito académico y científico gracias a su diseño dinámico y su flexibilidad.

A diferencia de otros marcos de trabajo más orientados a la producción, que utilizan grafos de computación estáticos, PyTorch adopta un enfoque define-by-run (definición en tiempo de ejecución) que permite construir y depurar modelos de forma más intuitiva, facilitando la

experimentación, la personalización de arquitecturas y el refinamiento (Fine Tuning) de modelos pre-entrenados. PyTorch cuenta con un ecosistema sólido que incluye librerías como torchvision, la cual ofrece implementaciones optimizadas de modelos pre-entrenados para tareas de visión por computador. Esta biblioteca incluye arquitecturas de referencia como ResNet, Faster R-CNN o SSD, entrenadas en datasets de gran escala como COCO o ImageNet. Estas capacidades hacen posible aplicar técnicas de Transfer Learning y Fine Tuning de manera eficiente, adaptando modelos consolidados a dominios específicos con menor cantidad de datos etiquetados.

Además, PyTorch ofrece compatibilidad con aceleradores de hardware como GPUs mediante la librería CUDA, lo que permite reducir de forma significativa los tiempos de entrenamiento. Su amplia comunidad de usuarios y desarrolladores garantiza soporte continuo, abundante documentación y acceso a recursos educativos y modelos de última generación.

La elección de PyTorch para este proyecto responde, por tanto, a su idoneidad para el desarrollo de soluciones personalizadas de Computer Vision, su flexibilidad para la investigación y su capacidad para integrar fácilmente modelos pre-entrenados y técnicas avanzadas de entrenamiento.

3.2 LIBRERÍAS AUXILIARES Y PROCESAMIENTO DE IMÁGENES

Entre las herramientas fundamentales utilizadas destaca OpenCV (Open Source Computer Vision Library), una de las bibliotecas más consolidadas y versátiles para el procesamiento de imágenes en Python. OpenCV permite realizar operaciones esenciales como la lectura y escritura de imágenes y vídeos, la extracción de tramas a intervalos definidos y la aplicación de filtros y transformaciones geométricas. En este proyecto se ha empleado especialmente para la extracción de tramas (frames) a partir de secuencias de vídeo, permitiendo definir tasas de muestreo personalizadas y aplicar filtros para descartar imágenes redundantes o poco informativas mediante comparación de diferencias entre tramas consecutivas.

Otra librería utilizada ha sido PIL/Pillow, que facilita la manipulación básica de imágenes como redimensionado, recorte o conversión de formatos. Estas operaciones son fundamentales para normalizar los datos de entrada y adaptarlos a los requisitos de los modelos de aprendizaje profundo seleccionados, garantizando la coherencia en tamaño y formato de los conjuntos de entrenamiento, validación y prueba.

Para la visualización de resultados y análisis exploratorio de datos, se ha empleado Matplotlib, una librería estándar en el ecosistema Python. Matplotlib permite generar gráficos y representaciones gráficas de las imágenes con sus correspondientes cajas delimitadoras (bounding boxes), facilitando la verificación del etiquetado y el análisis cualitativo de las predicciones generadas por los modelos. Además, su capacidad para personalizar gráficos y exportarlos en formatos de alta calidad la hace especialmente útil para la elaboración de figuras y resultados incluidos en la memoria del proyecto.

También se ha empleado tqdm, una librería ligera pero especialmente útil para el seguimiento de la evolución de bucles largos y procesos por lotes. Su integración permite generar barras de progreso en terminales o notebooks, ofreciendo una estimación visual del avance de tareas como la extracción y procesamiento masivo de imágenes o el entrenamiento de modelos. Gracias a esta funcionalidad, fue posible confirmar que los procesos se ejecutaban correctamente y detectar posibles bloqueos o comportamientos inesperados sin tener que esperar a su finalización completa, mejorando así la supervisión y el control durante el desarrollo.

NumPy ha sido una herramienta indispensable para el manejo eficiente de datos numéricos, ofreciendo estructuras de matrices (arrays) y operaciones vectorizadas que facilitan transformaciones y cálculos intermedios durante el preprocesamiento y análisis de imágenes.

Pandas se ha utilizado para la gestión y análisis de datos tabulares, permitiendo estructurar resultados, generar tablas comparativas de métricas y facilitar la interpretación de resultados de validación y prueba.

Scikit-learn ha proporcionado funciones esenciales como "train_test_split" para dividir los datos en conjuntos de entrenamiento y validación de forma reproducible y aleatoria, además de utilidades para el cálculo y visualización de matrices de confusión que permiten evaluar el rendimiento de los modelos en términos de verdaderos positivos, falsos positivos y falsos negativos.

xml.etree.ElementTree, del módulo estándar de Python, se ha empleado para el procesamiento de anotaciones en formato PASCAL VOC (XML), extrayendo la información de las cajas delimitadoras (bounding boxes) para su posterior conversión al formato YOLO requerido por algunos de los modelos utilizados.

os, glob, shutil y time son librerías del núcleo de Python empleadas para la gestión de archivos y carpetas, la búsqueda automatizada de rutas y nombres de archivos, la organización y copia de directorios y la medición de tiempos de ejecución, respectivamente. Su uso ha permitido estructurar el flujo de trabajo de forma ordenada, automatizar tareas repetitivas y controlar el rendimiento de los procesos.

gc (garbage collector) se ha utilizado para la gestión explícita de la memoria durante la ejecución de scripts largos o en entornos con recursos limitados, asegurando la liberación de memoria no utilizada para evitar cuellos de botella o errores por falta de espacio.

IPython.display ha permitido enriquecer la presentación de resultados en notebooks, mostrando imágenes procesadas o resultados de inferencia de forma interactiva e inmediata, facilitando la evaluación cualitativa del comportamiento de los modelos durante el desarrollo.

google.colab.files se ha empleado para la carga y descarga de archivos en Google Colab, optimizando la integración con Google Drive y permitiendo gestionar fácilmente datasets y resultados desde la nube.

Ultralytics ha sido fundamental para el entrenamiento e inferencia con modelos YOLOv8, proporcionando una API intuitiva que gestiona la carga de datos, la configuración de hiper-

parámetros, el seguimiento de métricas y la exportación de modelos entrenados de forma sencilla y eficiente.

La combinación de todas estas librerías auxiliares ha resultado esencial para garantizar un flujo de trabajo ordenado y eficiente, cubriendo todas las etapas del procesamiento de datos: desde la extracción y preparación de imágenes hasta la inspección, validación y documentación de los resultados obtenidos tras el entrenamiento de los modelos. En el Anexo II se muestra una tabla con las librerías utilizadas, una breve descripción de su uso y sus referencias.

3.3 MODELOS DE DETECCIÓN UTILIZADOS

La elección del modelo de detección de objetos constituye uno de los elementos más críticos en el diseño de un sistema capaz de identificar la presencia humana no autorizada en imágenes de videovigilancia. Para este proyecto, se buscaba un equilibrio entre precisión, capacidad de adaptación al problema concreto y viabilidad de implementación en un flujo de trabajo realista.

En este contexto, se seleccionaron Faster R-CNN y YOLOv8 como arquitecturas principales por su complementariedad y relevancia en el estado del arte. Faster R-CNN, con su enfoque two-stage, ofrece una alta precisión en la localización de objetos y una tasa reducida de falsos positivos, cualidades especialmente valiosas en sistemas donde las falsas alarmas tienen un coste operativo elevado. Además, su arquitectura modular y el soporte de bibliotecas como torchvision facilitan su entrenamiento personalizado mediante técnicas de Transfer Learning y Fine Tuning.

Por otro lado, YOLOv8 representa la evolución de una familia de modelos “one-stage” orientados a la detección en tiempo real, capaz de procesar imágenes con latencia mínima y alta velocidad de inferencia. Esta característica resulta fundamental en escenarios donde se requiere analizar múltiples flujos de vídeo simultáneamente o se contemplan implementaciones en dispositivos de procesamiento local (edge computing). Aunque existen

versiones más recientes de la familia YOLO, como YOLOv11, la elección de YOLOv8 en este proyecto responde a un equilibrio entre madurez, disponibilidad de documentación y compatibilidad con herramientas consolidadas. YOLOv8 cuenta con una implementación estable y muy bien integrada en la librería Ultralytics, que facilita de forma significativa su entrenamiento y validación con datos personalizados gracias a su API intuitiva.

Ambas arquitecturas cuentan con implementaciones maduras y optimizadas, modelos pre-entrenados en conjuntos de datos de referencia (como COCO) y comunidades de desarrolladores activas, lo que reduce la barrera de entrada técnica y asegura la sostenibilidad del desarrollo. Esta combinación de precisión, adaptabilidad y soporte práctico ha motivado la selección de Faster R-CNN y YOLOv8 como opciones idóneas para abordar el problema planteado en el proyecto.

3.3.1 FASTER R-CNN

Faster R-CNN es uno de los modelos más influyentes y utilizados en la detección de objetos, perteneciente a la familia de arquitecturas "two-stage" (dos etapas). Su diseño modular y su enfoque jerárquico permiten separar la generación de regiones candidatas de la clasificación y regresión de las cajas, logrando así una mayor precisión en la localización y etiquetado de objetos.

La arquitectura de Faster R-CNN puede desglosarse en tres componentes principales:

1. Backbone o extractor de características:

El primer bloque consiste en una Red Neuronal Convolucional profunda utilizada para extraer mapas de características de la imagen de entrada. Habitualmente se utilizan arquitecturas pre-entrenadas sobre grandes conjuntos de datos (como ImageNet) para facilitar el Transfer Learning y mejorar la capacidad de generalización. Un ejemplo común es ResNet50, que ofrece un equilibrio adecuado entre profundidad y eficiencia computacional, es el elegido en este proyecto. Alternativamente, se pueden emplear variantes más ligeras como ResNet18 para escenarios con limitaciones de hardware, sacrificando algo de precisión a cambio de mayor velocidad.

El backbone convierte la imagen en un mapa de características rico y compacto que contiene información espacial y semántica clave para las siguientes etapas.

2. Region Proposal Network (RPN):

La innovación central de Faster R-CNN frente a sus predecesores (R-CNN y Fast R-CNN) es la integración de esta red que genera propuestas de regiones (Region Proposals) de forma eficiente y completamente diferenciable. La RPN desglosa el mapa de características en pequeñas ventanas y para cada una predice:

- El desplazamiento (offset) necesario para ajustar una caja ancla (anchor box) a la forma real del objeto.
- La probabilidad de que esa región contenga un objeto (objetividad).

El entrenamiento conjunto de la RPN con el backbone permite que el sistema aprenda a sugerir regiones relevantes para la tarea de detección, reduciendo el número de propuestas redundantes o irrelevantes. A diferencia de métodos anteriores que dependían de algoritmos externos como Selective Search, la RPN está completamente integrada en el flujo (pipeline) y es altamente eficiente.

3. Cabeza de detección (Detection Head):

Las regiones propuestas por la RPN se utilizan para extraer, mediante técnicas como ROI Pooling o ROI Align, regiones fijas y normalizadas del mapa de características. Estas regiones son posteriormente procesadas por una red de clasificación y regresión de cajas que predice:

- La clase del objeto contenido en la región.
- Las coordenadas refinadas de la caja delimitadora (bounding box).

Este diseño en dos etapas permite especializar cada componente para su tarea concreta: la RPN para localizar regiones de interés de forma general, y la cabeza de detección para clasificar con precisión y ajustar las cajas (*Ilustración 9*).

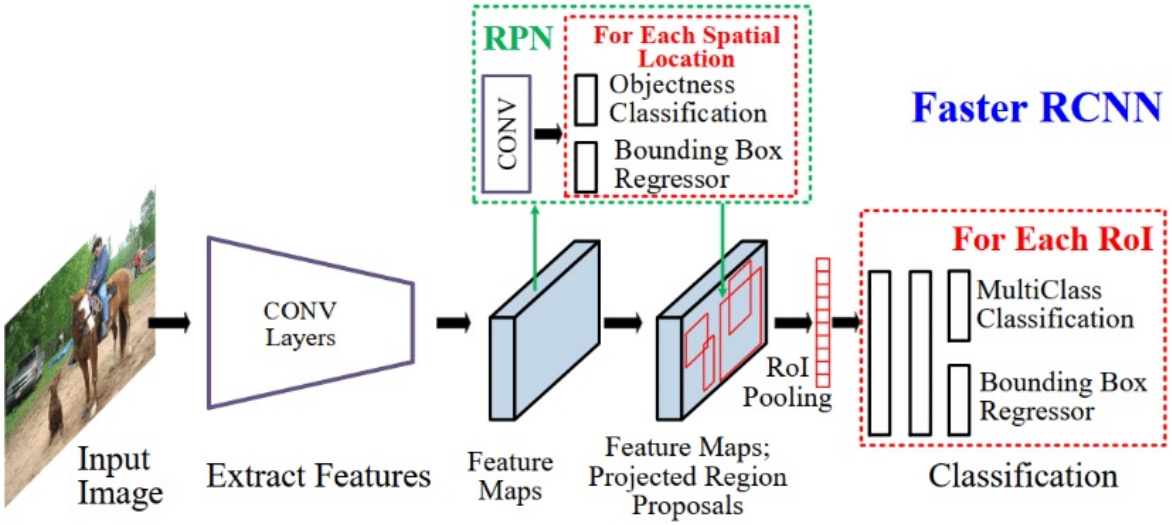

Ilustración 9. Arquitectura de Faster R-CNN (medium.com)

Faster R-CNN es la evolución natural de sus predecesores:

- R-CNN (2014): usaba Selective Search para generar ~2000 regiones por imagen, procesadas individualmente por una CNN. Muy preciso pero lento y costoso.
- Fast R-CNN (2015): optimizó el proceso al aplicar la CNN sobre la imagen completa antes de extraer regiones, acelerando la inferencia.
- Faster R-CNN (2015): integró la RPN en el entrenamiento y la inferencia, eliminando la dependencia de algoritmos externos para propuestas.

Desde su introducción, Faster R-CNN ha servido de base para múltiples mejoras y variantes, incluyendo:

- Backbones más profundos o ligeros (ResNet50, ResNet101, MobileNet).
- ROI Align (más preciso que ROI Pooling).
- Feature Pyramid Networks (FPN) para mejorar detección a múltiples escalas.

Faster R-CNN destaca por su alta precisión y robustez, especialmente en escenarios donde los falsos positivos y negativos son críticos. Su arquitectura modular permite ajustar el balance entre complejidad y rendimiento, eligiendo la base (backbone) más adecuada a las

necesidades y recursos disponibles. Para el problema de detección de presencia humana no autorizada en videovigilancia, ofrece la capacidad de identificar personas con gran fiabilidad incluso en escenas complejas o con múltiples objetos presentes.

3.3.2 YOLOv8

YOLOv8 representa la evolución más reciente de la familia de modelos You Only Look Once (YOLO), una serie de arquitecturas "one-stage" ampliamente utilizadas en la detección de objetos por su capacidad de procesar imágenes de forma rápida y eficiente. A diferencia de los modelos "two-stage", como Faster R-CNN, que separan la generación de propuestas y la clasificación en pasos distintos, YOLO adopta un enfoque integral donde ambas tareas se resuelven de manera conjunta y en una sola pasada por la red.

La idea central detrás de YOLO es dividir la imagen de entrada en una rejilla (grid) y asignar a cada celda la responsabilidad de predecir tanto las coordenadas de las cajas delimitadoras como las probabilidades de clase para los objetos presentes. Este diseño permite realizar predicciones completas en un solo paso, reduciendo de manera significativa el tiempo de inferencia y haciéndolo especialmente adecuado para aplicaciones en tiempo real (*Ilustración 10*).

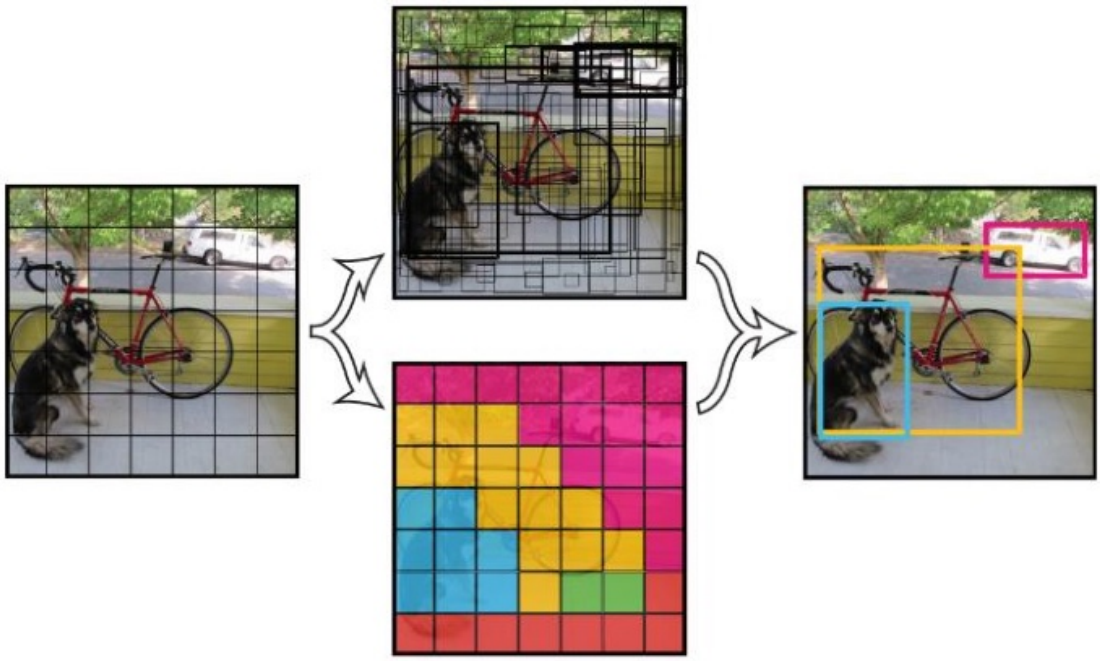

Ilustración 10. YOLO grid-based object detection framework (Abdussalam Ali Ahmed)

En YOLOv8, la arquitectura ha experimentado varias mejoras respecto a sus versiones anteriores, consolidándose como uno de los modelos más equilibrados en términos de velocidad y precisión. Entre sus innovaciones destacan:

- Arquitectura modular optimizada: Uso de bloques convolucionales eficientes y diseño más ligero sin sacrificar capacidad expresiva.
- Mejor manejo de objetos pequeños: Integración de Feature Pyramid Networks (FPN) para fusionar información multiescala.
- Head avanzado para predicciones: Enfoques anchor-free o adaptativos para predicción de cajas, eliminando la necesidad de definir manualmente anchor boxes.
- Estrategias de entrenamiento mejoradas: Augmentación de datos más sofisticada, funciones de pérdida optimizadas y uso de técnicas modernas de regularización.

YOLOv8 ofrece diferentes tamaños de modelo (nano, small, medium, large, extra-large), cada uno diseñado para balancear precisión y coste computacional (*Ilustración 11*). En este

proyecto se ha elegido YOLOv8m (medium) por su equilibrio óptimo entre precisión y eficiencia, especialmente adecuado para un caso de videovigilancia que, aunque realista, presenta condiciones relativamente controladas y no excesivamente complejas. Además, se ha considerado que elegir modelos más grandes no garantiza siempre mejores resultados, ya que conlleva un mayor riesgo de sobreentrenamiento (overfitting), especialmente cuando se dispone de un conjunto de datos con limitaciones en volumen o diversidad.

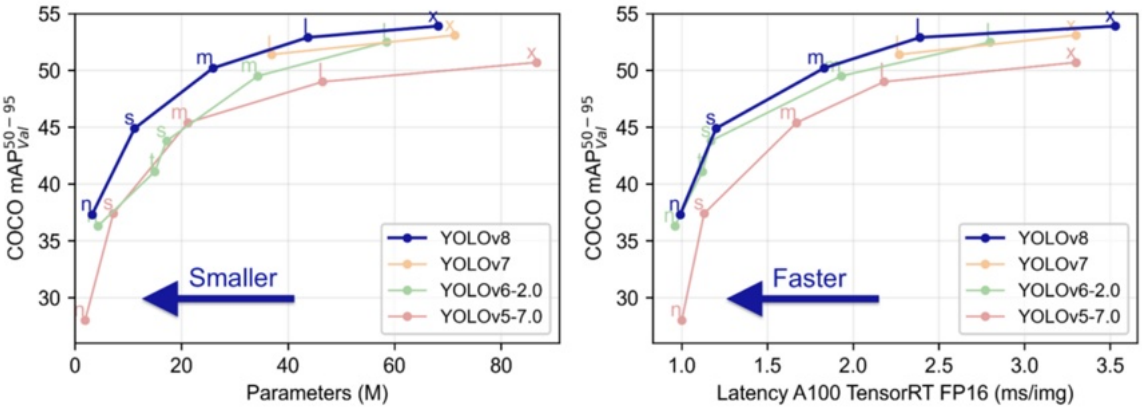

Ilustración 11. Comparación modelos y tamaños (Ultralytics)

Desde el punto de vista práctico, YOLOv8 destaca por la sencillez y accesibilidad de su implementación a través de la librería Ultralytics, que ofrece interfaces claras tanto en línea de comandos como en código Python. Sin embargo, esta accesibilidad exige cumplir con ciertos requisitos estructurales en la preparación de los datos. Ultralytics requiere una estructura de carpetas muy definida, normalmente siguiendo el esquema:

- *images/train* y *images/val*: directorios con las imágenes del conjunto de entrenamiento y validación, respectivamente.
- *labels/train* y *labels/val*: directorios con los archivos de etiquetas correspondientes, en formato YOLO (archivos .txt que contienen las coordenadas normalizadas de las cajas delimitadoras y la clase asociada).

Cada archivo de etiqueta debe compartir nombre con su imagen correspondiente y detallar cada objeto presente en la imagen en líneas separadas. Este formato compacto y estandarizado permite a YOLOv8 procesar de manera eficiente los datos durante el entrenamiento.

Por ello, en el proyecto ha sido necesario preparar cuidadosamente los datos para cumplir con esta estructura, incluyendo la conversión de formatos de anotación (por ejemplo, de PASCAL VOC a YOLO) y la organización en conjuntos de entrenamiento y validación adecuados. Estas etapas son fundamentales para garantizar la compatibilidad con la API de Ultralytics y permitir un entrenamiento sin errores, asegurando además la reproducibilidad del proceso.

3.4 HERRAMIENTAS DE ETIQUETADO DE DATOS

El proceso de desarrollo de modelos de detección de objetos supervisado depende de manera crítica de la calidad y precisión de los datos de entrenamiento. Una parte esencial de este pipeline es el etiquetado de datos (ya que no se parte de datos etiquetados), que consiste en delimitar manualmente las regiones de interés (bounding boxes) en las imágenes y asignarles las clases correspondientes.

Para este proyecto se han utilizado dos herramientas principales: “LabelImg” y “CVAT” (Computer Vision Annotation Tool), seleccionadas en función de sus características, facilidad de uso y compatibilidad con los formatos de anotación requeridos.

LabelImg es una herramienta de etiquetado ligera y de código abierto, muy extendida en la comunidad de Computer Vision. Permite la creación rápida de anotaciones en el formato PASCAL VOC (XML), ofreciendo una interfaz sencilla para dibujar cajas delimitadoras y asignar clases (*Ilustración 12*). Su instalación local y su bajo consumo de recursos la hacen ideal para tareas de etiquetado inicial y proyectos de pequeño o mediano tamaño. Sin embargo, durante el desarrollo se detectaron limitaciones de estabilidad, especialmente al

procesar lotes grandes de imágenes, experimentando cierres inesperados y dificultades para manejar proyectos extensos de forma eficiente.

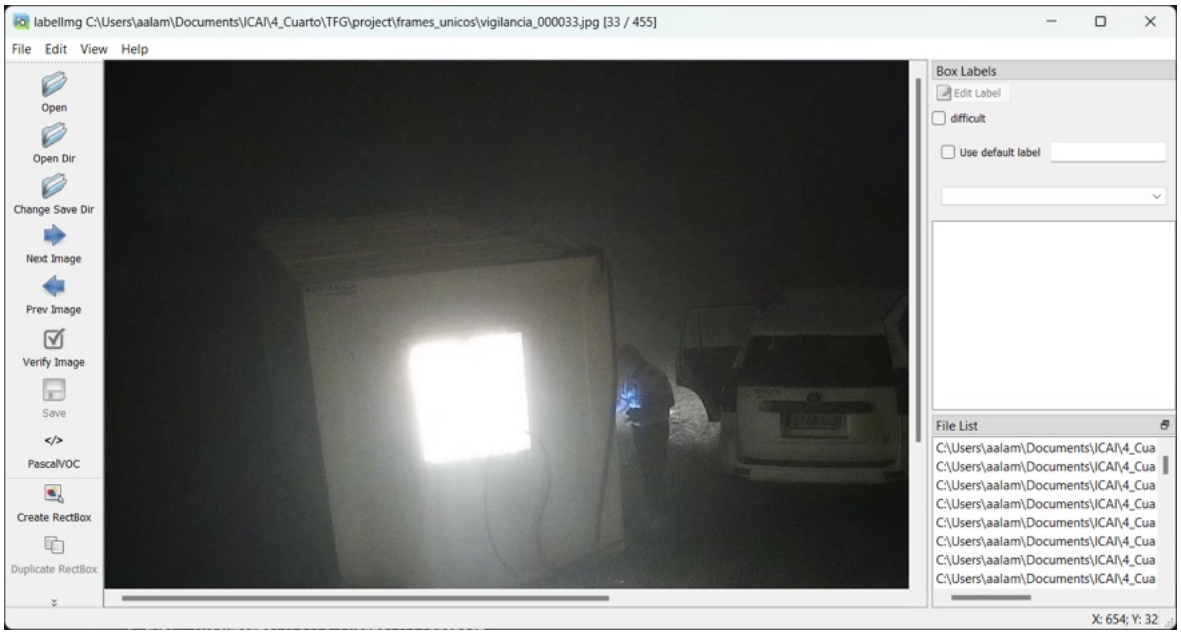

Ilustración 12. Interfaz de labelImg (elaboración propia)

Por este motivo, el flujo de trabajo se complementó con CVAT, una herramienta de anotación más avanzada y robusta, diseñada para entornos colaborativos y proyectos de mayor escala. CVAT ofrece una interfaz web accesible desde distintos dispositivos y navegadores, soporte para múltiples usuarios y una gestión estructurada de proyectos y tareas. Su flexibilidad permite exportar anotaciones en diversos formatos estándar, como PASCAL VOC y YOLO, lo que resulta especialmente útil para proyectos que requieren compatibilidad con distintos entornos de entrenamiento (*Ilustración 13*).

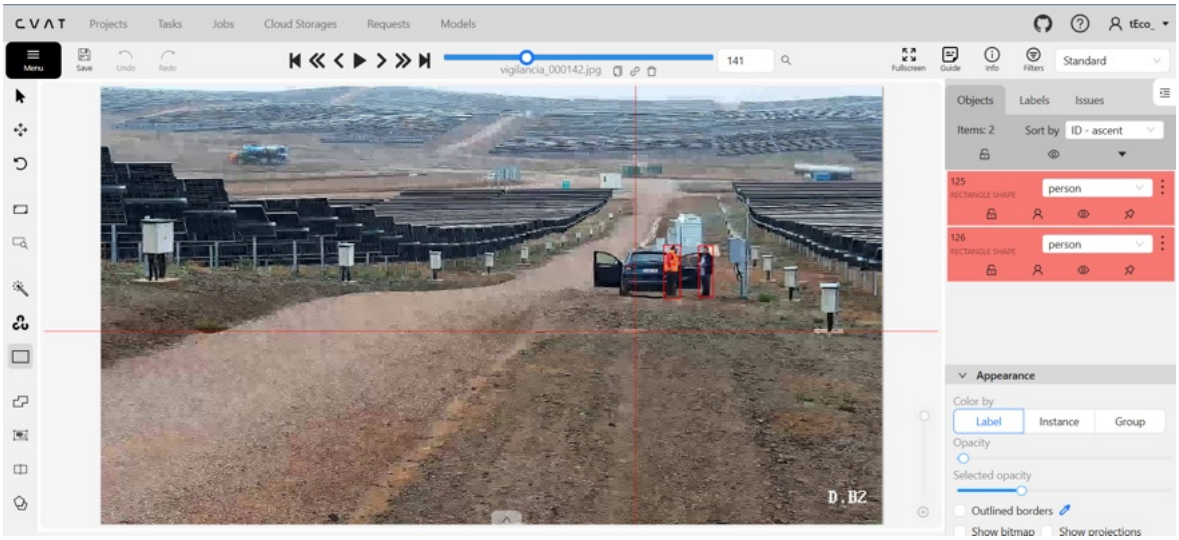

Ilustración 13. Interfaz online de CVAT (elaboración propia)

3.5 PLATAFORMAS Y RECURSOS DE COMPUTACIÓN

El entrenamiento de modelos de Deep Learning, especialmente aquellos aplicados a tareas de Computer Vision como la detección de objetos, requiere recursos computacionales adecuados para manejar grandes volúmenes de datos, procesar redes neuronales profundas y acelerar las iteraciones de prueba y ajuste. En este proyecto se han empleado distintos recursos locales y en la nube para cubrir las necesidades específicas de cada fase del desarrollo.

3.5.1 RECURSOS LOCALES CON GPU:

Para tareas de procesamiento previo, como la extracción y filtrado de tramas a partir de secuencias de vídeo, se han utilizado equipos personales con GPUs compatibles con CUDA. Estas tarjetas gráficas permitieron acelerar operaciones de procesamiento de imágenes realizadas con OpenCV y otras librerías, reduciendo significativamente los tiempos necesarios para generar conjuntos de datos de entrenamiento.

El desarrollo del código y la organización de scripts se realizaron utilizando Visual Studio Code, un entorno de desarrollo ligero y altamente configurable. Se creó un entorno virtual (.venv) específico para el proyecto, aislando las librerías utilizadas y sus versiones. Esta práctica permitió:

- Evitar conflictos de dependencias con otros proyectos o con el sistema base.
- Controlar con precisión las versiones de cada paquete, garantizando la reproducibilidad del entorno.
- Facilitar la reproducibilidad, al poder documentar y compartir de forma sencilla el conjunto de dependencias requeridas mediante archivos como requirements.txt.
- Mejorar la organización del código, con soporte para extensiones, depuración interactiva y gestión de notebooks.

Disponer de este entorno local configurado de forma estructurada y aislada resultó fundamental para la preparación de datos, la validación de scripts y el análisis exploratorio previo al entrenamiento intensivo de modelos.

3.5.2 GOOGLE COLAB:

Para el entrenamiento de los modelos de detección de objetos (Faster R-CNN y YOLOv8), se empleó Google Colab, una plataforma en la nube que ofrece entornos de desarrollo basados en Jupyter Notebook con acceso a GPUs. Google Colab dispone de recursos gratuitos accesibles para todos los usuarios, pero en este proyecto se utilizó una suscripción de pago (Colab Pro) para disponer de prestaciones superiores y mayor capacidad de cómputo.

Gracias a la suscripción, se tuvo acceso a hardware de alto rendimiento, concretamente:

- GPU: NVIDIA A100-SXM4-40GB
- VRAM disponible: 42.47 GB

El uso de esta GPU permitió acelerar de forma significativa los tiempos de entrenamiento y realizar experimentos más ambiciosos, con mayor tamaño de batch y modelos más pesados, que no serían viables en recursos gratuitos o en hardware local más limitado.

Google Colab resultó especialmente adecuado para este proyecto por diversas razones:

- Acceso sencillo y económico a GPUs de última generación sin necesidad de adquirir hardware propio.
- Entorno basado en Jupyter Notebook que facilita la documentación del proceso, la visualización de resultados y la organización del flujo de trabajo.
- Compatibilidad con Google Drive, lo que permitió almacenar conjuntos de datos, modelos entrenados y resultados de forma centralizada y accesible desde distintos dispositivos.
- Entorno preconfigurado con librerías de Machine Learning y Deep Learning como PyTorch, torchvision ya instaladas y optimizadas, reduciendo la necesidad de configuraciones manuales. Aun así, sigue siendo posible instalar y modificar lo que se necesite (en este caso se instaló Ultralytics).

La elección de Google Colab Pro fue clave para superar las limitaciones de hardware local durante las fases más intensivas del entrenamiento, garantizando un desarrollo fluido, eficiente y adaptado a las necesidades del proyecto.

Capítulo 4. DESARROLLO

4.1 EXTRACCIÓN DE DATOS.

El punto de partida para el desarrollo del sistema fue la disponibilidad de vídeos reales de videovigilancia con códec H264, obtenidos gracias a la colaboración y supervisión del director del proyecto. Estas grabaciones procedían de entornos de seguridad reales, aportando un valor significativo al garantizar que los datos representaban situaciones auténticas y condiciones operativas plausibles (*Ilustración 14*).

Ilustración 14. Ejemplo videovigilancia (elaboración propia)

La elección del códec H264 (MPEG-4 AVC) se justifica por ser uno de los estándares más extendidos y compatibles en aplicaciones de videovigilancia y transmisión en red. Su uso está generalizado en cámaras IP, grabadores digitales y sistemas de gestión de vídeo, gracias a su equilibrio entre compresión eficiente y calidad de imagen, que permite almacenar y transmitir grandes volúmenes de vídeo sin requerir ancho de banda o espacio excesivos.

Además, emplear H264 evitó problemas de compatibilidad que podrían surgir con otros códecs menos estandarizados o más recientes, cuya decodificación puede requerir software o hardware especializado no siempre disponible en todos los entornos de trabajo. Al elegir un formato ampliamente soportado, se aseguró la reproducibilidad y portabilidad del flujo de procesamiento, facilitando tanto la extracción de tramas como la integración de la solución en infraestructuras reales de videovigilancia.

Sin embargo, trabajar directamente con vídeos completos resultaba ineficiente y poco práctico para el entrenamiento de modelos de detección de objetos, que requieren imágenes estáticas con anotaciones precisas de las regiones de interés. Por ello, se diseñó un proceso de extracción de tramas que transformara los vídeos en un conjunto de imágenes útiles y representativas para su posterior etiquetado y análisis.

Para esta tarea se empleó la librería OpenCV. Se implementaron scripts personalizados para recorrer los vídeos almacenados en distintas carpetas y extraer tramas a intervalos configurables. Este intervalo se definía en milisegundos (por ejemplo, cada 200–500 ms) para lograr un equilibrio entre:

- Capturar suficiente variabilidad temporal.
- Evitar la extracción de tramas consecutivas demasiado similares, que aportarían poco valor y ocuparían espacio innecesario.

Una vez extraídas las tramas iniciales, se introdujo un mecanismo de filtrado adicional basado en la diferencia media entre tramas consecutivas. El objetivo de este filtrado era descartar imágenes excesivamente parecidas (por ejemplo, en escenas estáticas sin cambios significativos) para optimizar el conjunto de datos y reducir la redundancia. El algoritmo

implementado calculaba la media de las diferencias absolutas de píxeles entre la trama actual y la anterior. Si esta diferencia media era inferior a un umbral configurable, la trama se descartaba automáticamente. De este modo, se retuvieron solo aquellas imágenes que mostraban cambios sustanciales, mejorando la diversidad del conjunto final y haciendo más eficiente el etiquetado posterior (*Ilustración 15*).

```
Procesando vídeo: 00000002740000000.mp4
   -> FPS: 25.00
   -> Saltando cada 7 frames aproximadamente.
Progreso global:   5%|          | 317721/6342760 [07:37<18:12, 5513.03frame/s]
Frames guardados de 00000002740000000.mp4: 48

Procesando vídeo: 00010002417000000.mp4
   -> FPS: 25.00
   -> Saltando cada 7 frames aproximadamente.
Progreso global:  10%|          | 646061/6342760 [22:36<5:39:37, 279.55frame/s]
Frames guardados de 00010002417000000.mp4: 140
```

Ilustración 15. Fragmento de salida extrayendo frames (elaboración propia)

La gestión de carpetas y nombres de archivos se diseñó con especial cuidado para evitar solapamientos y pérdidas de información. Cada trama extraída se almacenaba con un nombre único que incluía identificadores del vídeo de origen y del instante temporal, garantizando la trazabilidad y la organización clara de los datos. Además, los scripts generaban automáticamente la estructura de carpetas necesaria para mantener separados los datos por origen o categoría, facilitando su manejo posterior.

Para supervisar y monitorizar el progreso en la extracción de grandes volúmenes de datos, se integró la librería tqdm[12], que permite mostrar barras de progreso en terminales o notebooks (puede observarse también en *Ilustración 12*). Esta herramienta ofrecía información en tiempo real sobre el avance del proceso, el tiempo estimado restante y la tasa

[12] tqdm, [en línea], disponible en: https://tqdm.github.io/ (consulta: 13/07/2025).

de procesamiento, resultando especialmente útil para identificar posibles bloqueos o errores en ejecuciones prolongadas.

En conjunto, se implementaron scripts personalizados en Python que automatizaban todo este flujo de trabajo, desde la carga de vídeos hasta la generación del conjunto de imágenes filtradas y estructuradas. Esta automatización aportó varias ventajas clave:

- Reducción drástica del tiempo y esfuerzo manual necesario para procesar los datos.
- Mayor consistencia y reproducibilidad del pipeline de extracción.
- Flexibilidad para ajustar parámetros como el intervalo de muestreo (sampling) o el umbral de diferencia según las necesidades del proyecto.

Gracias a este proceso sistemático y cuidadosamente diseñado, se obtuvo un conjunto inicial de imágenes estáticas preparado para la siguiente etapa del pipeline: el etiquetado de las regiones de interés para su uso en modelos de detección de objetos.

4.2 PROBLEMAS INICIALES Y SOLUCIONES

Tras completar la extracción inicial de tramas a partir de los vídeos de videovigilancia, se realizó un análisis preliminar del conjunto de datos generado para evaluar su idoneidad como base para el entrenamiento de modelos de detección de objetos. Este análisis permitió identificar varios problemas significativos que condicionaban la calidad y utilidad del conjunto de datos en su estado original.

En primer lugar, se observó una cantidad insuficiente de datos en general. Aunque la extracción desde vídeos generó un volumen considerable de imágenes, la gran mayoría correspondía a escenas estáticas, pasillos vacíos o áreas sin actividad relevante. Esto ocurre aun con el filtrado implementado en la extracción de tramas ya que solo tenía en cuenta la diferencia entre tramas consecutivas, pero ante cambios momentáneos en la imagen (de calidad, brillo, etc.) seguían apareciendo imágenes muy parecidas, aunque no fuesen inmediatamente una detrás de la otra. Se aplicó nuevamente un filtrado manual en este caso.

El número de imágenes potencialmente útiles para entrenar un detector de presencia humana era muy limitado.

Además, se identificó un desbalanceo severo de clases. La proporción de imágenes con presencia humana frente a aquellas sin personas era extremadamente desigual. Este desbalance planteaba un riesgo alto de sesgo durante el entrenamiento: el modelo podría aprender a "acertar" en la mayoría de los casos prediciendo siempre la ausencia de personas, con un desempeño muy pobre para detectar las situaciones realmente relevantes.

Se evaluaron diversas soluciones para superar estas limitaciones. Una opción inicial considerada fue aplicar técnicas de aumento de datos (Data Augmentation, como rotaciones, flips, cambios de iluminación, etc.) sobre las pocas imágenes con personas disponibles. Sin embargo, aunque estas técnicas pueden ampliar artificialmente el conjunto de datos, no generan ejemplos nuevos genuinos ni solucionan el problema de la falta de variedad.

Otra posibilidad teórica habría sido recoger más datos reales, es decir, grabar o conseguir más vídeos de videovigilancia con incidencias de intrusión humana. Sin embargo, esta opción resultaba impracticable por la propia naturaleza del problema: afortunadamente para la seguridad real, las incidencias con intrusos son poco frecuentes. Esta escasez, aunque deseable en términos reales, complicaba la recopilación de datos con suficientes ejemplos positivos.

Ante estas limitaciones, se decidió complementar el conjunto de datos propio con un conjunto de datos público ya disponible y etiquetado: el "INRIA Person Dataset[13]". Este conjunto de datos contiene imágenes con personas en diferentes entornos y condiciones, con anotaciones en formato PASCAL VOC que describen las cajas delimitadoras correspondientes. Su uso aportó varias ventajas clave:

- Incrementó el número total de ejemplos positivos (imágenes con personas).

[13] INRIA Person Dataset, [en línea], disponible en: https://universe.roboflow.com/pascal-to-yolo-8yygq/inria-person-detection-dataset (consulta: 13/07/2025)

- Aumentó la variedad de contextos y escenarios, mejorando la capacidad del modelo para generalizar.
- Evitó la necesidad de recolectar más datos reales o de forzar técnicas de augmentación poco realistas, así como su etiquetado manual.

La solución adoptada consistió en fusionar controladamente los datos propios extraídos de vídeos reales con las imágenes del conjunto de datos público, generando así un conjunto más equilibrado y rico en términos de ejemplos positivos y negativos. Pasamos de 455 imágenes de tramas a un total de 1363 imágenes combinadas. Esta decisión permitió superar los problemas iniciales y sentó las bases para construir un sistema de detección más fiable, robusto y adaptado a escenarios realistas de videovigilancia.

4.3 ETIQUETADO DE DATOS

Una vez extraído y filtrado el conjunto inicial de imágenes, el siguiente paso esencial en el pipeline de desarrollo fue el etiquetado de datos, consistente en definir manualmente las cajas delimitadoras (bounding boxes) que delimitan la presencia humana en las imágenes. Este paso es imprescindible para entrenar modelos supervisados de detección de objetos con la precisión requerida.

Para abordar esta tarea se consideró inicialmente el uso de LabelImg[14]. Sin embargo, como ya se adelantó, durante las primeras pruebas se detectaron problemas de estabilidad importantes: la aplicación experimentaba cierres inesperados y errores críticos al intentar dibujar las cajas delimitadoras en lotes grandes de imágenes. Estas limitaciones hacían impracticable su uso para el volumen de datos planificado en el proyecto.

Ante estas dificultades se decidió sustituir LabelImg por CVAT[15] (Computer Vision Annotation Tool), una herramienta de anotación más robusta, avanzada y diseñada para

[14] LabelImg, [en línea], disponible en: https://github.com/HumanSignal/labelImg (consulta: 13/07/2025)
[15] CVAT, [en línea], disponible en: https://app.cvat.ai/ (consulta: 13/07/2025)

proyectos de mayor escala. Su estabilidad y conjunto de funciones facilitaron enormemente el proceso de etiquetado, permitiendo trabajar de manera sistemática y reducir la probabilidad de errores (*Ilustración 10*).

Una decisión clave en esta fase fue exportar todas las anotaciones en formato PASCAL VOC (XML), a pesar de que se sabía que para el entrenamiento con YOLOv8 sería necesario el formato YOLO (TXT) más adelante. Esta estrategia respondió a la necesidad de mantener la coherencia con el conjunto de datos complementario utilizado, el "INRIA Person Dataset", que ya estaba etiquetado en formato VOC.

Optar por un formato maestro común antes de la conversión permitió unificar todos los datos (propios y públicos) en una única estructura estandarizada y compatible. Esto facilitó:

- La revisión y validación de las anotaciones.
- La integración controlada de datos de distintas fuentes.
- La preparación de un conjunto de datos consistente y homogéneo para el entrenamiento con modelos como Faster R-CNN, que acepta directamente el formato VOC.

Aunque CVAT ofrecía la opción de exportar directamente en formato YOLO, se decidió posponer la conversión a este formato para una etapa posterior. Esta decisión se basó en el hecho de que las imágenes que etiquetamos en CVAT eran solo una parte de nuestros datos, los de videovigilancia que no estaban etiquetados inicialmente. Como las imágenes de INRIA estaban ya etiquetadas en formato VOC, se decidió que la conversión se haría conjuntamente desarrollando un pequeño script de código, así tratábamos ya los datos como un único conjunto de datos lo antes posible.

Esto permitía mantener el control del proceso y asegurar que todas las transformaciones se hicieran de manera uniforme y consistente sobre un conjunto de datos previamente validado y estructurado en VOC, reduciendo la posibilidad de errores y garantizando la reproducibilidad del pipeline.

4.4 DIVISIÓN DEL DATASET

Antes de nada, en esta parte se pasó ya a utilizar Google Colab, por lo que se utilizó la librería drive de google.colab para acceder a drive donde subí los datos, además de activar la aceleración con CUDA en el dispositivo. Naturalmente, se dividió el conjunto de datos en subconjuntos de entrenamiento y validación, empleando para ello la función train_test_split de scikit-learn con una proporción de 80/20.

Otra parte que hubo que llevar a cabo es el filtrado de imágenes para entrenamiento sin cajas delimitadoras, es decir, descartamos en entrenamiento todas las imágenes que no contenían personas. Esto se hizo así por ver que los modelos daban errores puesto que esperan que en entrenamiento todas las imágenes tengan alguna caja delimitadora.

Durante este proceso se generaron archivos o listas de índices que identificaban de forma clara qué imágenes pertenecían a cada conjunto. Esto permitió reproducir la división en experimentos posteriores y facilitó la organización de carpetas para cada fase del pipeline.

Además, se tomó la decisión de reservar un pequeño conjunto de imágenes aparte para inferencia manual al final del proyecto. Estas imágenes no se incluyeron ni en entrenamiento ni en validación, con el objetivo de contar con un subconjunto independiente para realizar evaluaciones cualitativas finales. Este enfoque permitió:

- Verificar de forma realista la capacidad del modelo para generalizar.
- Realizar análisis visuales de predicciones en condiciones no vistas durante el entrenamiento.
- Generar resultados ilustrativos para la documentación.

Con todo esto, el número final de imágenes en entrenamiento era de 1004 y de 250 en validación.

4.5 CONVERSIÓN DEL FORMATO DE ANOTACIÓN PARA YOLO

Tras unificar y etiquetar todas las imágenes en formato PASCAL VOC (XML), se abordó la necesidad de convertir las anotaciones al formato YOLO (TXT) para poder entrenar el modelo YOLOv8m.

El formato PASCAL VOC representa las anotaciones mediante archivos XML con el formato siguiente:

```
<annotation>
 <folder>VOC2007</folder>
 <filename>crop_000005.png</filename>
 <source>
 ...
 </source>
 <size>
 ...
 </size>
 <segmented>0</segmented>
 <object>
  <name>person</name>
  <pose>Unspecified</pose>
  <truncated>0</truncated>
  <difficult>0</difficult>
  <bndbox>
   <xmin>171</xmin>
   <ymin>72</ymin>
   <xmax>260</xmax>
   <ymax>321</ymax>
  </bndbox>
 </object>
 <object>
 ...
 </object>
</annotation>
```

Este formato es ampliamente utilizado y resulta especialmente adecuado para modelos como Faster R-CNN, que pueden procesar directamente los datos en VOC gracias al soporte de librerías como torchvision.

Por otro lado, YOLOv8 requiere un formato diferente, más compacto y normalizado, en el que las coordenadas de las cajas delimitadoras se almacenan en formato relativo y

normalizado respecto al tamaño de la imagen. Cada archivo de anotación en YOLO consiste en un archivo de texto (".txt") asociado a cada imagen, donde cada línea representa un objeto con los siguientes campos (*Ilustración 17)*:

- ID de clase.
- Coordenadas normalizadas: centro x, centro y, ancho, alto (todos en el rango [0,1]).

```
0 0.647516 0.421444 0.028086 0.113125
0 0.777449 0.422708 0.024883 0.116917
```

Ilustración 16. Ejemplo formato de anotación YOLO (elaboración propia)

Para realizar la conversión, se desarrolló un script propio en Python. Este script leía los archivos XML del formato VOC, extraía la información de cada caja delimitadora y convertía las coordenadas absolutas a coordenadas relativas normalizadas en el rango [0,1]. Cada archivo de imagen generaba su archivo de texto correspondiente en el formato YOLO exigido por Ultralytics.

Este proceso incluyó validaciones para asegurar que todas las cajas delimitadoras se convertían correctamente y que no se perdía ninguna anotación en la transformación. Además, se diseñó para manejar automáticamente toda la estructura de carpetas.

Antes de realizar la conversión, se creó la estructura de carpetas exigida por Ultralytics YOLOv8, separando de antemano las imágenes y dejando preparadas las carpetas para las anotaciones en entrenamiento y validación:

- *images/train* y *images/val* para las imágenes del conjunto de entrenamiento y validación.
- *labels/train* y *labels/val* para los archivos de etiquetas correspondientes en formato YOLO.

Durante la conversión, el script guardaba directamente los archivos de texto generados en su carpeta de destino correspondiente (labels/train o labels/val), asegurando así que el conjunto de datos final quedara completamente preparado y organizado para el entrenamiento.

4.6 ENTRENAMIENTO DEL MODELO FASTER R-CNN

El entrenamiento del modelo Faster R-CNN constituyó una de las fases clave del proyecto, se diseñó cuidadosamente para aprovechar las capacidades del modelo preentrenado y adaptarlas al dominio específico de la videovigilancia.

El conjunto de datos se preparó en formato PASCAL VOC (XML), que se había adoptado como formato maestro para mantener la coherencia entre los datos propios y el conjunto de datos público complementario. Este formato era compatible de forma nativa con torchvision.datasets.VOCDetection, facilitando su integración en el flujo (pipeline) de entrenamiento en PyTorch.

Se utilizó la arquitectura torchvision.models.detection.fasterrcnn_resnet50_fpn, cargada con pesos pre-entrenados en el conjunto de datos COCO. Esta elección permitió Transfer Learning, reutilizando las características generales aprendidas en un conjunto de datos amplio para mejorar la detección de personas en escenarios de videovigilancia.

El entrenamiento se estructuró como un proceso de Fine Tuning:

- Las capas iniciales de la base ResNet50 (backbone) se congelaron para conservar su conocimiento general.
- La capa convolucional final y la cabeza de detección se reentrenaron para especializarlas en el conjunto de datos objetivo.

Se configuraron y ajustaron experimentalmente los siguientes hiperparámetros clave:

- Número de épocas: 100. En la práctica infinito, no es importante, lo limitaremos con Early Stopping.

- Paciencia: 3. Este parámetro aplica Early Stopping (parar entrenamiento antes de terminar las epochs) si el modelo no mejora en 3 epochs consecutivas.
- Learning rate inicial (lr0): 0.0001, valor bajo para fine-tuning cuidadoso sobre pesos pre-entrenados.
- Batch size: 8, relativamente alto y rápido gracias a los recursos de Colab Pro.
- Optimizer: Adam, elegido por su capacidad de adaptación y estabilidad durante el entrenamiento.

Estos valores se ajustaron tras varias pruebas piloto, buscando un compromiso entre rapidez de convergencia y estabilidad en el entrenamiento. Además, se implementaron impresiones automáticas por pantalla de monitoreo del entrenamiento cada 10 batches para asegurar pronto que todo va bien y no perdemos tiempo con errores. También se guardaba automáticamente el último mejor modelo en drive para no perderlo si caducaba la sesión o había algún error y el entorno se reiniciaba. El código se muestra a continuación:

```
best_val_loss = float('inf')
patience = 3
counter = 0
epochs = 100

train_losses = []
val_losses = []

for epoch in range(epochs):
    print(f"\nEpoch {epoch+1}/{epochs} iniciando...")
    epoch_start = time.time()

    # TRAIN
    model_frcnn.train()
    train_loss_epoch = 0
    num_batches = len(train_loader_frcnn)

    for batch_idx, (images, targets) in enumerate(train_loader_frcnn):
        batch_start = time.time()

        images = [img.to(device) for img in images]
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        try:
            loss_dict = model_frcnn(images, targets)
            loss = sum(loss for loss in loss_dict.values())

            optimizer.zero_grad()
```

```
            loss.backward()
            optimizer.step()

            batch_time = time.time() - batch_start
            train_loss_epoch += loss.item()

            if batch_idx % 10 == 0:
                print(f"Batch {batch_idx}/{num_batches} - Loss: {loss.item():.4f}
- {batch_time:.2f}s")

        except Exception as e:
            print(f"Error en batch {batch_idx}: {e}")
            continue

    avg_train_loss = train_loss_epoch / num_batches
    train_losses.append(avg_train_loss)

    # VALIDATION
    model_frcnn.train()
    val_loss_epoch = 0
    num_val_batches = len(val_loader_frcnn)

    with torch.no_grad():
        for batch_idx, (images, targets) in enumerate(val_loader_frcnn):
            batch_start = time.time()

            images = [img.to(device) for img in images]
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

            try:
                loss_dict = model_frcnn(images, targets)
                loss = sum(loss for loss in loss_dict.values())

                batch_time = time.time() - batch_start
                val_loss_epoch += loss.item()

                if batch_idx % 10 == 0:
                    print(f"Val Batch {batch_idx}/{num_val_batches} - Loss:
{loss.item():.4f} - {batch_time:.2f}s")

            except Exception as e:
                print(f"Error en batch val {batch_idx}: {e}")
                continue

    avg_val_loss = val_loss_epoch / num_val_batches
    val_losses.append(avg_val_loss)

    epoch_time = time.time() - epoch_start
    print(f"\nEpoch {epoch+1} completada en {epoch_time:.2f}s")
    print(f"Train Loss: {avg_train_loss:.4f} | Val Loss: {avg_val_loss:.4f}")

    # Early Stopping
    if avg_val_loss < best_val_loss:
```

```
        best_val_loss = avg_val_loss
        counter = 0
        torch.save(model_frcnn.state_dict(), "best_fasterrcnn.pth")
        print("Mejor modelo guardado")
    else:
        counter += 1
        print(f"No mejora. EarlyStopping contador: {counter}/{patience}")
        if counter >= patience:
            print(f"Early stopping activado en epoch {epoch+1}")
            break

    # Limpieza de memoria
    torch.cuda.empty_cache()
    gc.collect()

# Guardar histórico
history_df = pd.DataFrame({
    'epoch': range(1, len(train_losses)+1),
    'train_loss': train_losses,
    'val_loss': val_losses
})
history_df.to_csv("loss_history.csv", index=False)
print("Historial de pérdidas guardado en loss_history.csv")
```

Durante el entrenamiento, el Early Stopping se activó en la epoch 8, por lo que el mejor modelo guardado correspondía al de la epoch 5. En la *Ilustración 20* podemos ver la evolución del Loss (métrica a minimizar) en entrenamiento frente a validación. Es normal que el entrenamiento baje más (si la diferencia es mucha puede indicar overfitting), pero buscamos mínima Loss en validación (epoch 5). Este fue el último entrenamiento puesto que se ve bastante bien, la Loss en validación no sube y la diferencia con el entrenamiento no es muy grande.

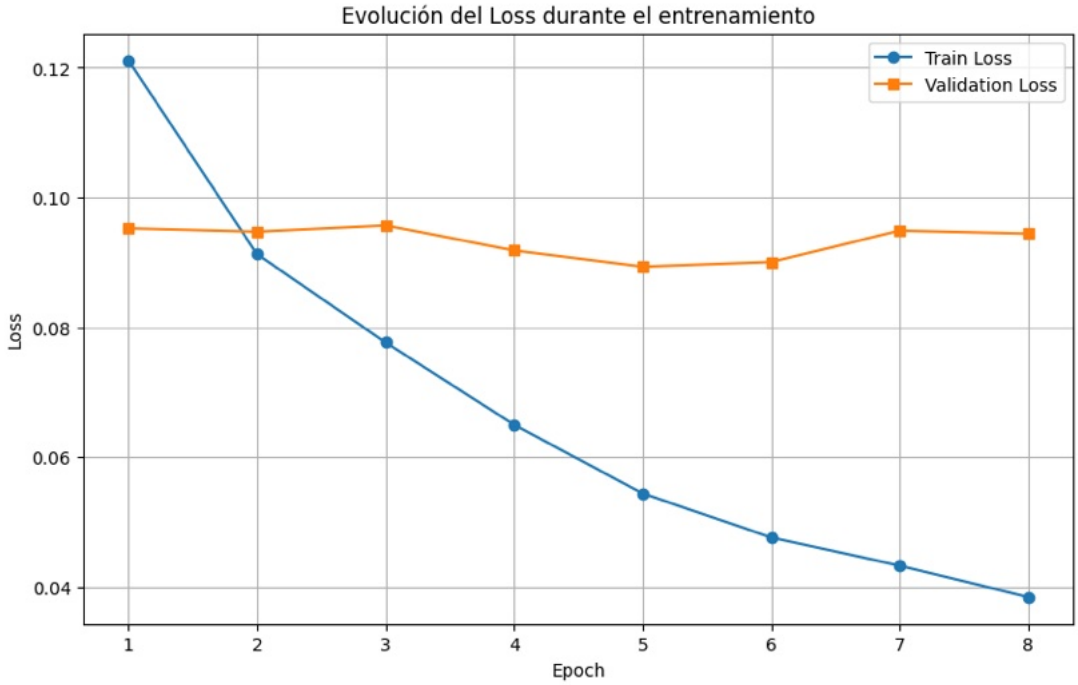

Ilustración 17. Train Loss VS Val Loss (elaboración propia)

4.7 ENTRENAMIENTO DEL MODELO YOLOV8M

El entrenamiento del modelo YOLOv8m constituyó la segunda rama principal del pipeline de detección de objetos en el proyecto.

Con la estructura de carpetas preparada previamente no hacía falta más que ejecutar un código muy simple y directo con pocos parámetros para entrenar el modelo ya que se hace de forma más directa con Ultralytics. Solo hubo que crear el archivo data.yaml que definía la ruta de estas carpetas y las clases a detectar, asegurando la compatibilidad con la API de Ultralytics. En este caso, el Fine Tuning también es automático, no hay pesos congelados, por lo que solo faltaba correr el siguiente código con los hiperparámetros seleccionados y optimizados experimentalmente:

```
# Entrenar (fine-tuning)
yolov8m.train(
    data='/content/drive/MyDrive/TFG/yolo_data/data.yaml',
    epochs=100,
    imgsz=800,
```

```
    batch=4,
    lr0=0.0001,
    optimizer='Adam',
    patience=3,
    workers=2,            # Menos workers para evitar RAM del sistema
    project='/content/drive/MyDrive/TFG/yolo_data',
    name='train_yolov8m',
    device=0
)
```

- Número de épocas: 100
- Imagen de entrada (imgsz): 800 píxeles (tamaño considerable para mayor resolución en detección).
- Batch size: 4, ajustado para balancear entre aprovechamiento de la GPU y evitar errores de memoria.
- Learning rate inicial (lr0): 0.0001, valor bajo para fine-tuning cuidadoso sobre pesos pre-entrenados.
- Optimizer: Adam, elegido por su capacidad de adaptación y estabilidad durante el entrenamiento.
- Patience: 3 (criterio de early stopping para prevenir overfitting si la validación no mejoraba).
- Workers: 2 (reducido para controlar el consumo de RAM en el entorno de entrenamiento).
- Device: 0 (uso de GPU disponible).

El entrenamiento se realizó en Google Colab Pro, aprovechando el acceso a GPUs de alto rendimiento como la NVIDIA A100-SXM4-40GB. Este recurso permitió reducir los tiempos de entrenamiento por epoch y procesar imágenes de alta resolución (800x800) manteniendo batch size y workers adecuados.

El framework de Ultralytics ofrece visualizaciones de métricas de entrenamiento y validación automáticamente, incluyendo las curvas de pérdida, precisión, recall, mAP (mean Average Precision), y otros indicadores (*Ilustración 21*).

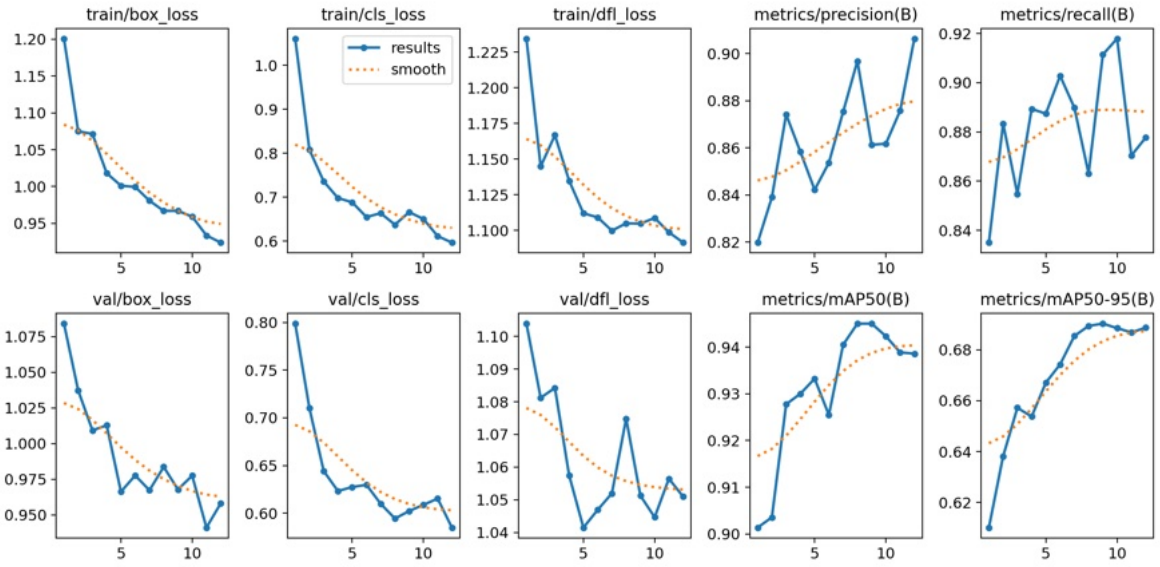

Ilustración 18. Best yolov8m metrics (elaboración propia)

Se observa un descenso sostenido de la pérdida de entrenamiento y validación, aunque en val/box_loss la mejor epoch debería ser la 9 pero parece ser la 11 según la gráfica. Esto puede deberse a que estas gráficas se hacen al final aproximadas, no con los mismos datos de cada epoch. Si esto no es así, entonces no se entiende, ya que el Early Stopping sí que es automático dentro del entrenamiento y coge y guarda siempre los valores del mejor modelo en su epoch correspondiente. Lo que está claro según las gráficas es que no hay overfitting ya que ambas funciones de pérdida (entrenamiento y validación) se reducen de la misma forma, sin aumentar significativamente en validación.

Se generaron también checkpoints automáticos, facilitando la recuperación del mejor modelo en validación.

El tamaño de imagen elevado (800 px) mejoró la capacidad del modelo para detectar detalles finos, especialmente en objetos pequeños o parcialmente ocluidos. Sin embargo, supuso también una mayor carga computacional, justificando el uso de recursos avanzados en la nube.

Al finalizar, el modelo entrenado se guardó con la estructura de proyecto definida (*yolo_data/train_yolov8m2*, el 2 es porque ya había otra carpeta de otro entrenamiento fallido por memoria RAM, Out Of Memory, por lo que se cambió el número de workers y batch_size), asegurando la organización y trazabilidad de resultados. El modelo resultante quedó listo para su uso en inferencia, permitiendo evaluar su capacidad para detectar presencia humana no autorizada en distintos escenarios de videovigilancia.

Capítulo 5. ANÁLISIS DE RESULTADOS

5.1 MÉTRICAS Y MATRIZ DE CONFUSIÓN

Para evaluar de forma objetiva el rendimiento de los modelos refinados desarrollados en este proyecto, se emplearon métricas cuantitativas obtenidas sobre el conjunto de validación y prueba.

En tareas de detección de objetos, estas métricas se derivan principalmente de la matriz de confusión, que clasifica las predicciones en cuatro categorías:

- True Positives (TP): casos en que el modelo detecta correctamente la presencia humana.
- False Positives (FP): predicciones erróneas donde el modelo indica presencia humana donde no la hay.
- False Negatives (FN): omisiones donde el modelo no detecta una presencia humana real.
- True Negatives (TN): predicciones correctas de ausencia de personas.

A partir de estos valores se calculan métricas clave:

- Precision: proporción de predicciones positivas correctas, calculada como TP / (TP + FP). Indica la exactitud del modelo al detectar personas.
- Recall: proporción de casos reales detectados, calculada como TP / (TP + FN). Mide la cobertura o sensibilidad del modelo.
- IoU (Intersection over Union): mide la superposición entre la caja delimitadora predicha y la real. Se calcula como área de intersección dividida por área de unión entre ambas cajas, con valores entre 0 y 1.

- mean Average Precision (mAP): media de la precisión obtenida en todos los niveles de recall. Para mAP50 se considera un umbral de IoU ≥ 0.50, mientras mAP50-95 promedia sobre umbrales más estrictos (de 0.50 a 0.95 en pasos de 0.05).

Estas métricas ofrecen una visión integral del rendimiento del modelo, abarcando tanto la capacidad de detectar correctamente objetos como la precisión en su localización.

5.1.1 FASTER R-CNN

El modelo Faster R-CNN refinado fue entrenado aplicando Fine Tuning sobre una base ResNet50 preentrenada, adaptándose al problema específico del proyecto.

Principales métricas obtenidas:

- IoU promedio: 0.8105.
- AP aproximado (mAP / mAP50): 0.9472
- Threshold óptimo encontrado: 0.10
- Tiempo promedio de inferencia por imagen: 0.0288 segundos (batch_size=1, GPU NVIDIA A100-SXM4-40GB en Google Colab Pro).

Matriz de confusión (*Ilustración 22*):

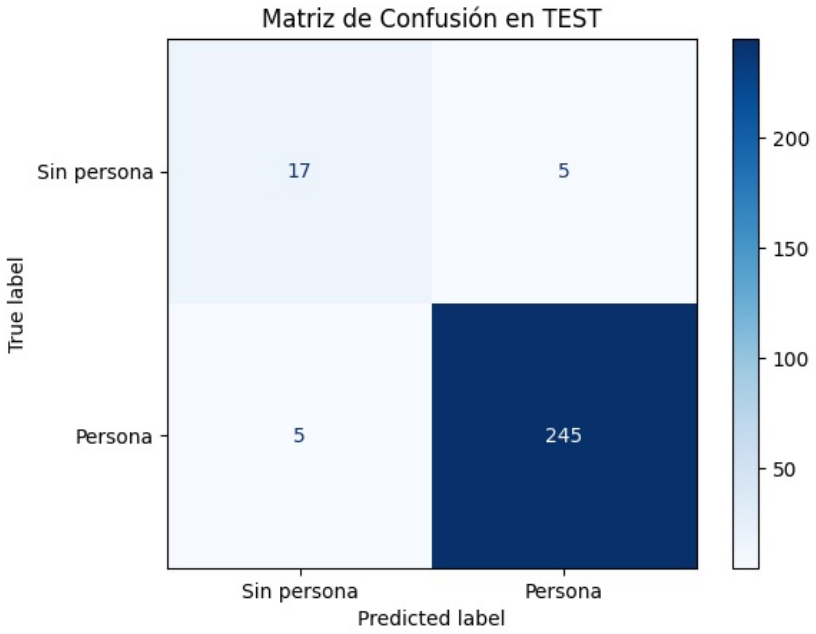

Ilustración 19. Matriz de confusión Faster R-CNN (elaboración propia)

El modelo mostró un rendimiento sobresaliente, con un mAP cercano a 0.95 y un IoU promedio superior a 0.8, lo que evidencia alta precisión en la localización de las personas en las imágenes. La matriz de confusión presenta valores muy bajos de FP y FN, destacando su capacidad para minimizar tanto alarmas falsas como omisiones de detección.

El umbral (threshold) óptimo bajo (0.10) permitió maximizar la recuperación de detecciones sin penalizar significativamente la precisión; aun así, este valor puede aumentarse si queremos reducir los falsos positivos (FP) aún más. Además, su tiempo de inferencia por imagen sigue siendo adecuado para escenarios cercanos a tiempo real, especialmente donde la prioridad es la fiabilidad de la detección.

5.2 YOLOv8m

El modelo YOLOv8m refinado fue entrenado con la biblioteca Ultralytics, aplicando Fine Tuning a partir de pesos pre-entrenados. Su diseño "one-stage" está optimizado para rapidez y procesamiento en tiempo real.

Principales métricas obtenidas:

- mAP50: 0.912
- mAP50-95: 0.658
- Precision: ~0.838
- Recall: ~0.894
- Tiempo de inferencia por imagen: ~0.0046 segundos (GPU NVIDIA A100-SXM4-40GB en Google Colab Pro).

Matriz de confusión (*Ilustración 23*):

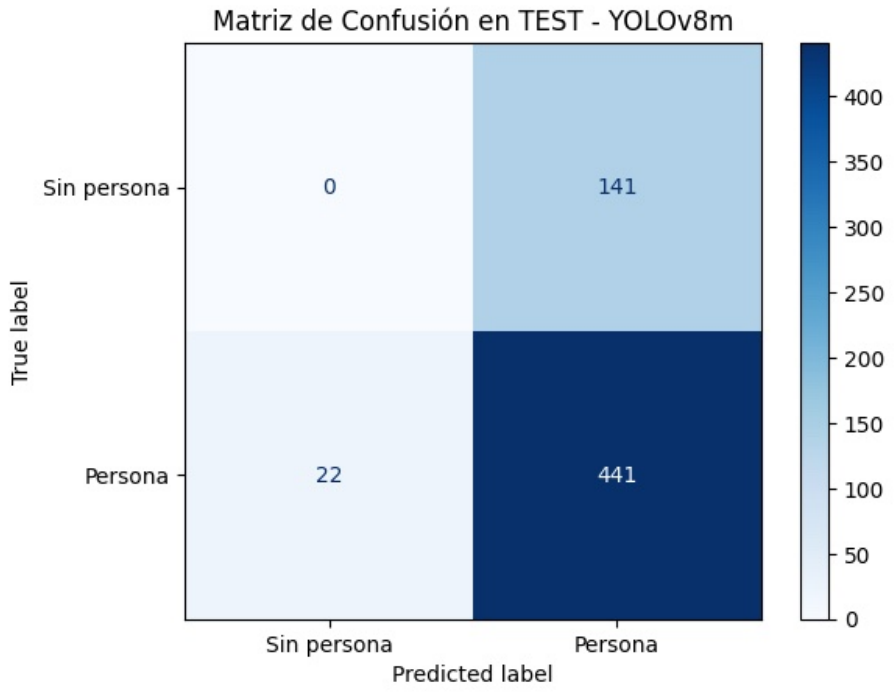

Ilustración 20. Matriz de confusión YOLOv8m (elaboración propia)

YOLOv8m mostró muy buena capacidad de detección global, con precision y recall elevados y un mAP50 superior a 0.91. Sin embargo, la matriz de confusión revela un número significativamente más alto de falsos positivos (141), reflejando su tendencia a ser más "agresivo" en las detecciones para no perder casos positivos.

El mAP50-95 más bajo (~0.658) indica que su precisión de localización fina disminuye cuando se exigen IoU más estrictos. Por otro lado, su tiempo de inferencia extremadamente bajo (menos de 5 ms por imagen) lo hace especialmente atractivo para aplicaciones en tiempo real con recursos limitados.

5.3 COMPARACIÓN DE RESULTADOS

A continuación, se comparan los valores más relevantes de las métricas obtenidas en cada modelo. Los datos de la matriz de confusión se normalizan en porcentaje para facilitar la interpretación relativa de aciertos y errores entre modelos con distintos números totales de predicciones. Esto es especialmente importante ya que el número total de predicciones evaluadas en cada modelo difiere. La razón de esto se debe a que cada arquitectura procesa y filtra resultados de manera distinta.

Al principio hicimos una división en train y test para Faster R-CNN que, al entrenar, se dividieron los datos de train en train y val (validación) para el entrenamiento completo. Tras esto, se calcularon las métricas sobre el conjunto de test reservado inicialmente. Como YOLO es más directo y automático, cogió directamente la división de sus carpetas en train y val, y las métricas finales son sobre el conjunto de validación sin más divisiones. Por esto, el número total evaluado es mayor en YOLO que en Faster R-CNN, pero los dos han sido entrenados sobre los mismos datos y pueden compararse sin mayor problema normalizando los datos de sus matrices de confusión (*Ilustración 24*).

Comparativa de métricas finales (con matriz de confusión en %)

Métrica	Faster R-CNN fine-tuned	YOLOv8m fine-tuned
mAP / mAP50	0.9472	0.912
TP (%)	90.1%	73.0%
FP (%)	1.8%	23.3%
FN (%)	1.8%	3.6%
TN (%)	6.3%	0.0%
Tiempo de inferencia (s/img)	0.0288	0.0046

Ilustración 21. Comparativa de métricas (elaboración propia)

Faster R-CNN alcanzó un mAP / mAP50 de 0.9472, superior al 0.912 de YOLOv8m. Aunque ambos valores son muy altos (indicando una excelente capacidad para predecir cajas delimitadoras con IoU ≥ 0.50), Faster R-CNN mostró una ventaja de aproximadamente 3.5 puntos porcentuales. Su IoU promedio, superior a 0.8, confirma esta capacidad de ajustar las cajas delimitadoras con más exactitud. En la matriz de confusión normalizada, Faster R-CNN mostró un 90.1 % de verdaderos positivos y apenas un 1.8 % de falsos positivos y negativos, reflejando un enfoque conservador y muy fiable para minimizar alertas erróneas.

En contraste, YOLOv8m evidenció un perfil más agresivo: obtuvo valores elevados de precision (~0.838) y recall (~0.894), pero con un 23.3 % de falsos positivos. Su diseño "one-stage" prioriza la rapidez, con tiempos de inferencia por imagen seis veces menores (~4.6 ms frente a ~28.8 ms de Faster R-CNN), haciéndolo mucho más adecuado para flujos de vídeo en tiempo real o entornos con hardware limitado.

Estas diferencias muestran dos estrategias complementarias: Faster R-CNN es más adecuado para sistemas donde la fiabilidad y la reducción de falsas alarmas son críticas, mientras YOLOv8m resulta más flexible para aplicaciones que requieren velocidad extrema y pueden

tolerar un mayor volumen de detecciones a filtrar o revisar. La elección entre ambos modelos depende por tanto de las prioridades del sistema de videovigilancia en el que se integren.

5.4 INFERENCIA SOBRE CONJUNTO RESERVADO

Para complementar el análisis cuantitativo de métricas globales y matrices de confusión, se realizó una evaluación cualitativa de los modelos sobre un conjunto de imágenes reservado, denominado test_vigilancia. Este conjunto incluía seis imágenes seleccionadas específicamente para no haber sido empleadas ni en entrenamiento ni en validación, garantizando así una prueba independiente y más realista de la capacidad de generalización de los modelos.

El objetivo de esta inferencia fue analizar y comparar el comportamiento real de los modelos en condiciones similares a las de aplicación, observando tanto sus aciertos como sus errores en situaciones nuevas.

Antes de mostrar las imágenes inferidas por el modelo Faster R-CNN refinado, veremos el resultado de inferir directamente con el modelo preentrenado sin aplicar ningún tipo de reentrenamiento (sin Transfer Learning ni Fine Tuning):

Ilustración 22. Inferencia Faster R-CNN preentrenado 1 (elaboración propia)

Ilustración 23. Inferencia Faster R-CNN preentrenado 2 (elaboración propia)

Ilustración 24. Inferencia Faster R-CNN preentrenado 3 (elaboración propia)

Ilustración 25. Inferencia Faster R-CNN preentrenado 4 (elaboración propia)

Ilustración 26. Inferencia Faster R-CNN preentrenado 5 (elaboración propia)

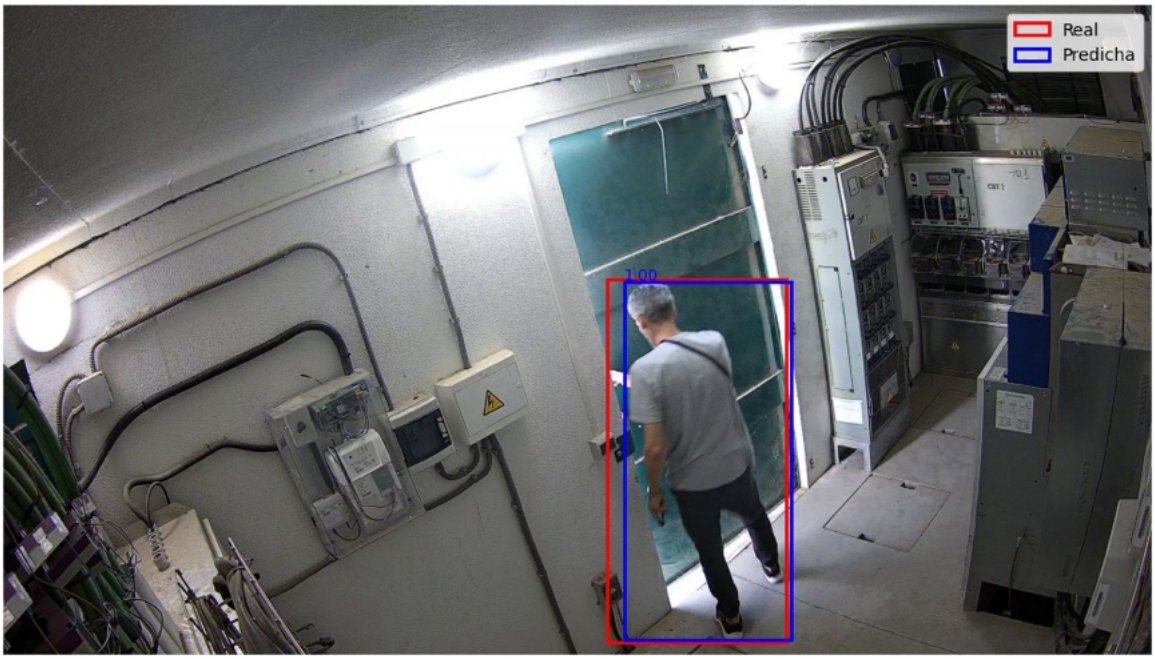

Ilustración 27. Inferencia Faster R-CNN preentrenado 6 (elaboración propia)

Como podemos ver, hay algunas imágenes con las que ya obtiene un muy buen resultado como en *Ilustración 25*, *Ilustración 26*, *Ilustración 30*. Al fin y al cabo, es un modelo muy potente entrenado sobre un conjunto de datos extenso sobre el que ya sabe lo que es una persona en general. Aun así, en la *Ilustración 27* aparece una tercera predicción de persona con una probabilidad del 0.80 que no debería estar ahí. La *Ilustración 28* quizás era muy optimista esperar que predijera una persona tan lejana con poca calidad, cosa que no consigue. Y en la *Ilustración 29* consigue identificar una persona, pero no la segunda.

A continuación, veremos si el mismo modelo refinado mejora estos resultados. Por ahorrar espacio, solo se mostrarán las imágenes con mejoras potenciales sustanciales, aquellas donde el modelo preentrenado no llegaba a tener el rendimiento esperado:

Ilustración 28. Inferencia Faster R-CNN refinado 5 (elaboración propia)

Ilustración 29. Inferencia Faster R-CNN refinado 4 (elaboración propia)

Ilustración 30. Inferencia Faster R-CNN refinado 3 (elaboración propia)

Ahora sí, los resultados son muy prometedores (ya lo adelantaban las métricas). En la *Ilustración 31* consigue identificar a las 2 personas cuando antes le faltaba una. En la *Ilustración 32* logra predecir una persona que está muy lejana en condiciones de baja calidad con una seguridad del 0.99, cosa que podía no esperarse de primeras por considerarse muy optimista, pero puede ser una prueba más de que la inteligencia artificial puede llegar a ver mejor que nosotros. Por último, la *Ilustración 33* es curiosa: identifica correctamente las dos personas presentes, pero sigue apareciendo una tercera caja, medio solapada con otra, que no debería estar ahí. Aun así, si nos fijamos bien, se puede apreciar que la probabilidad que le da es del 0.10 (nuestro umbral límite), valor que el modelo preentrenado daba como 0.80. Con esta información vemos que el modelo funciona por encima de las expectativas, el valor umbral puede aumentarse para reducir los falsos positivos como en este último caso, que lo eliminaría solo con un 0.15 o 0.2.

Veamos ahora qué tal infiere el modelo YOLOv8m preentrenado (sin refinar) sobre estas mismas imágenes:

Ilustración 31. Inferencia YOLOv8m preentrenado 1 (elaboración propia)

Ilustración 32. Inferencia YOLOv8m preentrenado 2 (elaboración propia)

Ilustración 33. Inferencia YOLOv8m preentrenado 3 (elaboración propia)

Ilustración 34. Inferencia YOLOv8m preentrenado 4 (elaboración propia)

Ilustración 35. Inferencia YOLOv8m preentrenado 5 (elaboración propia)

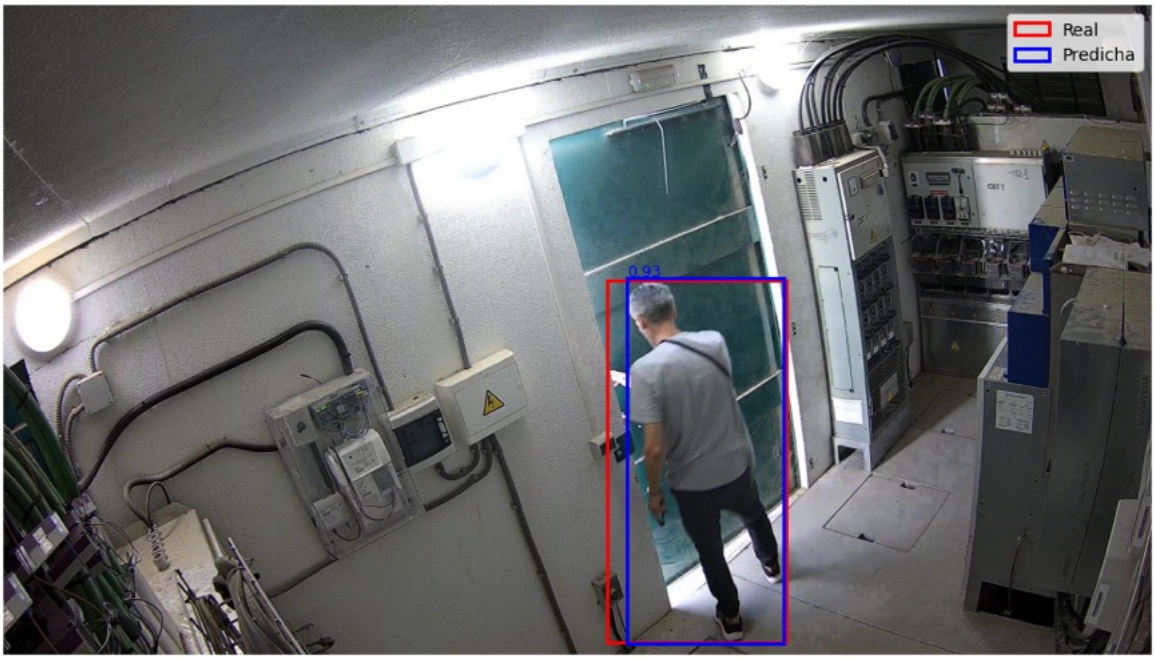

Ilustración 36. Inferencia YOLOv8m preentrenado 6 (elaboración propia)

Podemos ver que el desempeño es muy similar que el modelo Faster R-CNN preentrenado, al fin y al cabo, son dos modelos que están a la orden del día en detección de objetos, y además fueron entrenados sobre el mismo conjunto de datos (COCO). Aun así, podemos apreciar su menor precisión ya que los valores de probabilidad de predicción son menores. Esto implica que, si se sube el umbral detectará menos personas aún, y es importante tenerlo en cuenta ya que vimos que en la matriz de confusión que había un alto porcentaje de falsos positivos (23.3%). Dicha matriz era sobre el modelo refinado, pero tenemos una idea inicial de que puede ser interesante subir el umbral para reducir los falsos positivos, y no hay que olvidarse de que esto afectaría también reduciendo los TP y aumentando los falsos negativos.

Veamos el desempeño del modelo refinado YOLOv8m. En este caso, se mostrará la inferencia sobre las 6 imágenes para fijarnos también en cómo cambian los valores que discutíamos:

Ilustración 37. Inferencia YOLOv8m refinado 1 (elaboración propia)

Ilustración 38. Inferencia YOLOv8m refinado 2 (elaboración propia)

Ilustración 39. Inferencia YOLOv8m refinado 3 (elaboración propia)

Ilustración 40. Inferencia YOLOv8m refinado 4 (elaboración propia)

Ilustración 41. Inferencia YOLOv8m refinado 5 (elaboración propia)

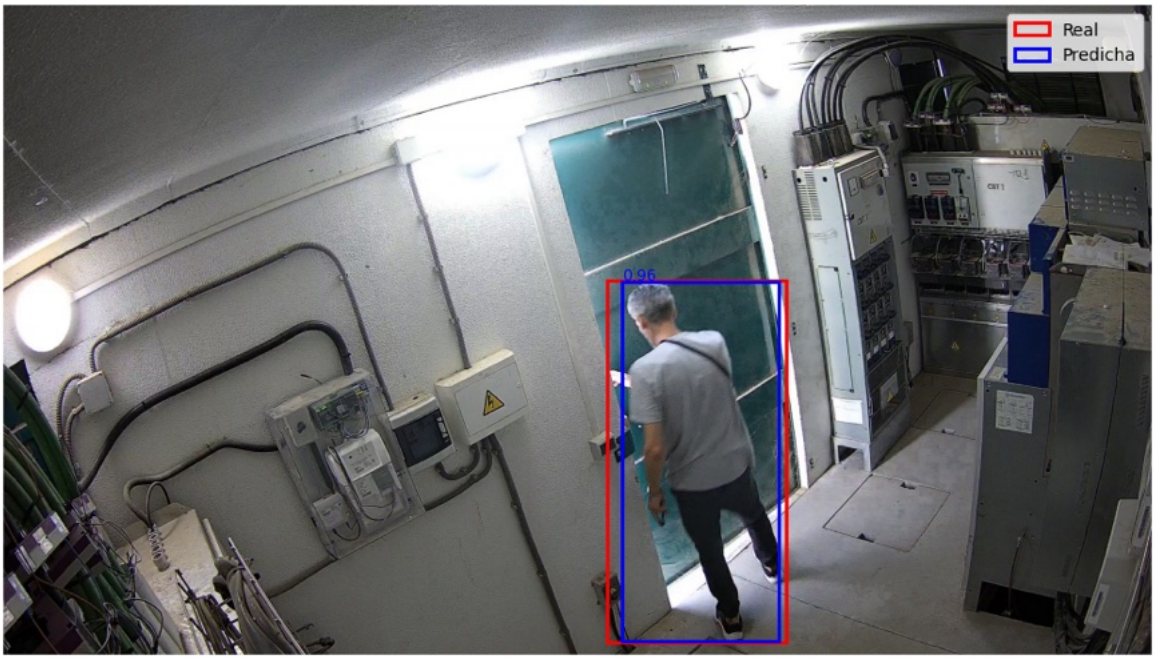

Ilustración 42. Inferencia YOLOv8m refinado 6 (elaboración propia)

De primeras podemos ver cómo todos los valores de predicción han aumentado considerablemente, el modelo está más seguro de las predicciones como era de esperar, ya que ha sido reentrenado para identificar personas en las condiciones específicas de estas cámaras de videovigilancia. También cabe destacar que este modelo no está mostrando predicciones con probabilidades menores a 0.50 (es el umbral en este caso), por lo que no vemos cajas que no deberíamos como en el caso anterior, lo cual no quiere decir que este sea mejor solo por eso (en el otro caso de umbral muy pequeño se explicó que se puede filtrar esas cajas con un umbral mayor). Por último, podemos concluir que tiene un muy buen desempeño a la vez de rápido; aun así, observamos cómo en casos complicados no llega a predecir bien las personas, como en la *Ilustración 43*.

Capítulo 6. CONCLUSIONES

6.1 CONCLUSIONES SOBRE LA METODOLOGÍA

El desarrollo del proyecto ha permitido diseñar y aplicar una metodología estructurada y rigurosa para construir un sistema de detección de presencia humana no autorizada en imágenes de videovigilancia.

Desde el inicio, se planteó un enfoque modular y planificado, que permitiera abordar de forma ordenada cada fase del flujo de trabajo y facilitar la identificación y resolución de problemas a medida que surgieran. Se comenzó con la extracción sistemática de datos a partir de vídeos reales, aplicando técnicas de muestreo configurable y filtros basados en diferencias de imagen para evitar redundancias y garantizar la diversidad de escenas.

La incorporación controlada de un conjunto de datos público complementario ("INRIA Person Dataset") resolvió la limitación inicial de datos positivos, permitiendo equilibrar el conjunto de entrenamiento e incrementar la variedad de contextos y condiciones, con el objetivo de mejorar la capacidad de generalización del modelo en escenarios reales.

La fase de etiquetado de datos se abordó con la herramienta CVAT, consolidando un formato común (PASCAL VOC) para unificar datos propios y públicos. Posteriormente, se implementó un proceso de conversión al formato YOLO, adaptando la estructura de carpetas y anotaciones al esquema requerido por Ultralytics, asegurando la compatibilidad con distintos modelos y la reutilización eficiente de los datos.

El pipeline se completó con la aplicación de Transfer Learning y Fine Tuning sobre modelos pre-entrenados (Faster R-CNN y YOLOv8m), entrenados con datos específicos del problema. Para ello se combinaron recursos locales y en la nube (Google Colab Pro con GPU), optimizando tiempos de procesamiento y permitiendo experimentar con configuraciones avanzadas.

Esta metodología no solo permitió construir un flujo de trabajo reproducible y bien documentado, sino también dejar sentadas las bases para futuras ampliaciones o adaptaciones. Gracias a su diseño modular, el sistema puede evolucionar incorporando nuevos conjuntos de datos, arquitecturas más ligeras para "edge computing" o detección multiclase, asegurando su aplicabilidad a problemas reales de videovigilancia en contextos variados.

6.2 CONCLUSIONES SOBRE LOS RESULTADOS

El análisis cuantitativo y cualitativo de los modelos entrenados ha puesto de manifiesto la eficacia del enfoque de refinamiento (Fine Tuning) para especializar arquitecturas del estado del arte en un dominio concreto como la videovigilancia. Partiendo de pesos pre-entrenados en conjuntos de datos generales, se ha logrado adaptar los modelos a las características propias de las imágenes de seguridad, que presentan condiciones particulares de iluminación, ángulos de cámara y compresión.

Faster R-CNN refinado ha mostrado resultados especialmente sólidos, logrando métricas de mAP / mAP50 y IoU superiores, junto con una matriz de confusión que refleja un porcentaje muy bajo de falsos positivos y negativos. Su arquitectura en dos fases ("two-stage"), orientada a la generación y refinamiento de propuestas, ofrece un enfoque conservador que prioriza la calidad de la detección y la localización precisa de las personas en la escena. Esto lo hace especialmente adecuado para aplicaciones donde la fiabilidad es crítica y el coste de las falsas alarmas resulta elevado, como en centros de control o sistemas de análisis centralizado. Además, su rendimiento se ha mantenido estable incluso en escenarios con iluminación adversa o pérdidas de calidad por compresión, condiciones realistas en sistemas de videovigilancia reales.

Por su parte, YOLOv8m refinado ha destacado por su velocidad de inferencia muy superior, con tiempos por imagen aproximadamente seis veces menores, lo que le permite analizar flujos de vídeo en tiempo real o incluso soportar implementaciones en dispositivos de computación en el borde ("edge computing"). Su diseño en una fase ("one-stage") está

optimizado para maximizar la cobertura y detectar el mayor número posible de objetos con latencia mínima. Sin embargo, esta agresividad en la detección se traduce en un mayor porcentaje de falsos positivos, lo que implica que, en un sistema práctico, sus resultados podrían requerir validación o filtrado adicional antes de generar alertas al operador humano.

En la evaluación sobre el conjunto reservado de imágenes se han confirmado claramente estas diferencias. Faster R-CNN mostró un comportamiento más equilibrado y consistente, con pocas alertas falsas y localizaciones ajustadas, adaptándose bien al escenario de procesamiento centralizado donde las imágenes pueden presentar compresión o pérdidas de calidad. En contraste, YOLOv8m demostró su fortaleza en rapidez y detección temprana, pero mantuvo cierta tendencia a generar falsas alarmas en casos más ambiguos o complejos.

En conjunto, se considera que Faster R-CNN refinado representa la solución óptima para el objetivo del proyecto, equilibrando alta precisión, bajo porcentaje de falsos positivos y un tiempo de inferencia suficientemente bajo para aplicaciones reales de videovigilancia. Su capacidad para reducir falsos positivos, falsos negativos y mantener la calidad de las detecciones incluso en condiciones no ideales lo convierte en la opción más adecuada para entornos de supervisión centralizada, donde la prioridad es minimizar las falsas alarmas y garantizar la fiabilidad del sistema.

Al mismo tiempo, los resultados de YOLOv8m destacan la posibilidad de adaptar la solución a otros escenarios, como la detección en tiempo real en el propio dispositivo o sistemas distribuidos, donde la latencia y el consumo de recursos son factores críticos. Esta complementariedad sugiere que ambas familias de modelos tienen su lugar en una solución más amplia y flexible, que pueda ajustarse a las necesidades y limitaciones concretas de cada instalación o sistema de videovigilancia.

Estas conclusiones no solo validan el enfoque metodológico adoptado en el proyecto, sino que también evidencian la importancia de seleccionar y ajustar cuidadosamente el modelo de detección en función de los requisitos operativos, el hardware disponible y las condiciones del entorno real de aplicación.

6.3 RECOMENDACIONES PARA FUTURO

El presente proyecto ha demostrado la viabilidad de un sistema automatizado de detección de presencia humana no autorizada, pero también abre la puerta a distintas líneas de mejora y adaptación para su despliegue en casos de uso variados.

Para sistemas distribuidos o procesamiento directamente en cada cámara (edge-computing), se recomienda evaluar arquitecturas más ligeras y optimizadas como YOLOv8n, MobileNet o EfficientDet, capaces de mantener un nivel aceptable de precisión con requisitos computacionales mucho menores. Este enfoque permitiría procesar imágenes localmente en tiempo real, reduciendo la necesidad de transmisión constante al centro de control.

Otra posible mejora consiste en entrenar los modelos con imágenes recopiladas directamente de cada cámara o ubicación específica en la que se pretenda desplegar el sistema. Esta personalización permitiría adaptar los detectores a las características particulares de cada escena (ángulos, iluminación, entorno), aumentando la precisión y reduciendo errores derivados de condiciones no vistas durante el entrenamiento general.

Finalmente, el sistema podría extenderse a más clases de objetos relevantes, como vehículos (coches, furgonetas) que también pueden ser objeto de intrusiones o accesos no autorizados en recintos vigilados. Esto requeriría un proceso adicional de anotación y entrenamiento multiclase, pero permitiría construir un sistema más completo y versátil para la protección de instalaciones sensibles.

Estas recomendaciones definen un camino claro para futuras ampliaciones y mejoras, orientadas a optimizar el sistema para distintas arquitecturas, escenarios de despliegue y requisitos operativos reales.

Capítulo 7. BIBLIOGRAFÍA

[1] Wani, M. Arif; Bhat, Farooq Ahmad; Afzal, Saduf. "Advances in Deep Learning: Applications and Challenges". Springer, 2020.

[2] Valkov, Venelin. "Hackers Guide to Machine Learning with Python". Leanpub, 2018.

[3] Ren, S.; He, K.; Girshick, R.; Sun, J. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". Advances in Neural Information Processing Systems 28 (NIPS 2015). December, 2015. https://arxiv.org/abs/1506.01497

[4] Bochkovskiy, A.; Wang, C. Y.; Liao, H. Y. M. "YOLOv4: Optimal Speed and Accuracy of Object Detection". arXiv preprint, April 2020. https://arxiv.org/abs/2004.10934

[5] Ultralytics. "YOLOv8 Documentation". Ultralytics, 2024. https://docs.ultralytics.com/

[6] INRIA. "INRIA Person Dataset". INRIA Research. 2008. http://pascal.inrialpes.fr/data/human/

[7] Alamán San Martín, Alejandro. "detector-presencia-humana". GitHub. 2025. https://github.com/alexx-asm/detector-presencia-humana

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

El organismo de las Naciones Unidas establece una serie de Objetivos de Desarrollo Sostenible (ODS) con el fin de conseguir un futuro sostenible para todos incorporando desafíos globales a los que nos enfrentamos día a día. Lo ideal sería cumplir con cada uno de ellos para 2030 de un total de 17 objetivos.

El presente proyecto tiene cierta relevancia según varios de estos objetivos:

ODS 8 – Trabajo decente y crecimiento económico: Una de las metas de este objetivo es lograr niveles más elevados de productividad económica mediante la diversificación, la modernización tecnológica y la innovación. Por tanto, poder automatizar el trabajo de revisión de la videovigilancia reduciendo el tiempo empleado conseguiría formar parte de un aumento de productividad.

ODS 9 – Industria, innovación e infraestructura: El sistema a desarrollar contribuiría con infraestructuras más seguras y eficientes en la industria.

ODS 11 – Ciudades y comunidades sostenibles: Así como se ha mencionado que contribuye a infraestructuras seguras en la industria, también se podría aplicar a ciudades y, por tanto, a su comunidad. El bienestar de la comunidad se reforzaría por medio de su seguridad, aspecto esencial para tener entornos urbanos sostenibles.

ODS 12 – Producción y consumo responsables: El sistema puede promover un uso más responsable de los recursos y disminuir los costes asociados no solo ahorrando tiempo en la visualización de imágenes de seguridad, sino que también puede llegar a reducir pérdidas y daños relacionados con ciertas intrusiones no autorizadas.

ODS 13 – Acción por el clima: Mejorando la eficiencia y seguridad de los sistemas de vigilancia es posible cesar o reducir la necesidad de otros procedimientos más contaminantes o energéticamente ineficientes. Ejemplos de esto pueden ser la iluminación, climatización, ventilación u otros servicios constantes posiblemente requeridos por la vigilancia física ininterrumpida.

ODS 16 – Paz, justicia e instituciones sólidas: Claro está que el sistema ayudaría a mantener entornos seguros contribuyendo a instituciones sólidas promoviendo la paz y la justicia.

ODS 17 – Alianzas para lograr los objetivos: Acorde al último objetivo de los ODS, el proyecto precisa de alianzas tanto técnicas como éticas. Para su realización es preciso la cooperación entre profesionales que desarrollen la tecnología, expertos en seguridad y las entidades que correspondan para su implementación. Aparte, para el desarrollo sostenible es necesario acompañarlo de unos principios y valores que garanticen el equilibrio entre seguridad y privacidad en este caso.

ANEXO II: LIBRERÍAS UTILIZADAS

Librería	**Uso principal en el proyecto**	**Referencia**
torch	Desarrollo, entrenamiento e inferencia de modelos de Deep Learning en PyTorch.	PyTorch, [en línea], disponible en: https://pytorch.org/ (consulta: 13/07/2025).
torchvision	Acceso a modelos preentrenados, transformaciones y utilidades para Computer Vision.	PyTorch, [en línea], disponible en: https://pytorch.org/ (consulta: 13/07/2025).
tqdm	Barra de progreso para bucles largos y tareas masivas.	tqdm, [en línea], disponible en: https://tqdm.github.io/ (consulta: 13/07/2025).
PIL / Pillow	Manipulación básica de imágenes (carga, redimensionado, formato).	Pillow, [en línea], disponible en: https://python-pillow.org/ (consulta: 13/07/2025).
matplotlib	Visualización de imágenes, gráficos y resultados con bounding boxes.	Matplotlib, [en línea], disponible en: https://matplotlib.org/ (consulta: 13/07/2025).
cv2 (OpenCV)	Procesamiento y extracción de frames desde vídeos, filtrado de imágenes.	OpenCV, [en línea], disponible en: https://opencv.org/ (consulta: 13/07/2025).

sklearn.model_selection	División de datos.	Scikit-learn, [en línea], disponible en: https://scikit-learn.org/ (consulta: 13/07/2025).
sklearn.metrics	Cálculo y visualización de matrices de confusión.	Scikit-learn, [en línea], disponible en: https://scikit-learn.org/ (consulta: 13/07/2025).
xml.etree.ElementTree	Procesamiento de anotaciones en formato PASCAL VOC (XML).	Python Standard Library, [en línea], disponible en: https://docs.python.org/3/library/xml.etree.elementtree.html (consulta: 13/07/2025).
torch.utils.data	Creación de datasets personalizados, DataLoader, división aleatoria.	PyTorch Docs, [en línea], disponible en: https://pytorch.org/docs/stable/data.html (consulta: 13/07/2025).
glob	Búsqueda y gestión de rutas de archivos.	Python Standard Library, [en línea], disponible en: https://docs.python.org/3/library/glob.html (consulta: 13/07/2025).
os	Gestión de rutas, carpetas y archivos del sistema.	Python Standard Library, [en línea], disponible en: https://docs.python.org/3/library/os.html (consulta: 13/07/2025).
time	Medición de tiempos de ejecución.	Python Standard Library, [en línea], disponible en: https://docs.python.org/3/library/time.html (consulta: 13/07/2025).

gc	Gestión del recolector de basura en Python.	Python Standard Library, [en línea], disponible en: https://docs.python.org/3/library/gc.html (consulta: 13/07/2025).
pandas	Estructuración y análisis de datos tabulares.	Pandas, [en línea], disponible en: https://pandas.pydata.org/ (consulta: 13/07/2025).
numpy	Operaciones numéricas de bajo nivel, arrays.	NumPy, [en línea], disponible en: https://numpy.org/ (consulta: 13/07/2025).
google.colab.files	Interacción con Google Colab para subida/descarga de archivos.	Google Colab, [en línea], disponible en: https://colab.research.google.com/ (consulta: 13/07/2025).
ultralytics	Entrenamiento e inferencia con modelos YOLOv8.	Ultralytics YOLO Docs, [en línea], disponible en: https://docs.ultralytics.com/ (consulta: 13/07/2025).
shutil	Operaciones avanzadas de copia, movimiento y eliminación de archivos y carpetas.	Python Standard Library, [en línea], disponible en: https://docs.python.org/3/library/shutil.html (consulta: 13/07/2025).
IPython.display	Visualización enriquecida de imágenes en notebooks.	IPython Documentation, [en línea], disponible en: https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html (consulta: 13/07/2025).