UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Universidad de Málaga

Escuela de Ingenierías Industriales

TRABAJO FIN DE GRADO

Incremental learning for object detection on video
Aprendizaje incremental para la detección de objetos en vídeo

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Mención en Robótica y Automatización

Autora: María Terán Vázquez

Tutor: Francisco Manuel Castro Payán
Departamento: Arquitectura de Computadores
Área de conocimiento: Arquitectura y Tecnología de Computadores

Cotutora: Paula Ruiz Barroso
Departamento: Arquitectura de Computadores
Área de conocimiento: Arquitectura y Tecnología de Computadores

Málaga, 22 de mayo de 2025

A mis padres, por no dudar de mí y darme la oportunidad de seguir este camino.

A Gabri, por ser mi compañero de vida y hacer de ella algo muy bonito.

A Andrea y Montse, por estar y ser hogar.

A toda mi familia, por su cariño incondicional y por acompañarme siempre.

A mi abuelo Paco, por enseñarme la pasión por aprender.

Y, en especial, a Pablo, por ser un verdadero ejemplo de lucha.

Resumen

El avance tecnológico en campos como la Inteligencia Artificial y la computación se ha visto reflejado en un incremento del interés por los sistemas de identificación de objetos en vídeo. Se trata de un área de investigación dentro de las aplicaciones de la visión por computador con multitud de posibilidades, como pueden ser el seguimiento de objetos, la vigilancia o la conducción autónoma. En los últimos años, ha tenido lugar una transformación significativa en la forma en la que se detectan objetos, abandonando métodos convencionales para adoptar otros más complejos, como el *Deep Learning*.

El uso del *Deep Learning* para la detección de objetos es un estándar, ya que obtiene resultados sin precedentes con una precisión superior a la de los humanos. Sin embargo, en entornos realistas, este rendimiento baja drásticamente, ya que el dominio puede cambiar considerablemente y su entrenamiento se ha realizado utilizando bases de datos académicas, con poca información cercana a la realidad. Esto hace necesaria una adaptación del dominio del modelo para que pueda manejar situaciones nuevas con diferentes puntos de vista de objetos ya conocidos o detectar objetos nuevos no considerados en la base de datos de entrenamiento.

En este TFG se aborda este problema aplicando un enfoque de aprendizaje incremental para adaptar el dominio del modelo y mejorar así su rendimiento de detección ante un entorno nuevo. Se ha utilizado un pipeline no supervisado que localiza objetos usando una red de detección, un flujo óptico pre-calculado y un clasificador de imágenes que asigna las clases a esos objetos. Posteriormente, se ha añadido aprendizaje incremental al clasificador de imágenes para adaptar el dominio del mismo a la tarea de destino.

Se han realizado diversos entrenamientos utilizando un *dataset* que incluye secuencias de vídeo tomadas en la zona de carga y descarga de un aeropuerto. En este tipo de entorno, aparecen objetos nuevos que no son conocidos y los objetos conocidos no aparecen en primer plano o, incluso, su tamaño es muy pequeño como para ser detectados con enfoques tradicionales.

Finalmente, se ha llevado a cabo una evaluación y comparativa de los resultados obtenidos.

Palabras Claves: Aprendizaje Incremental, Deep Learning, Red neuronal convolucional

Abstract

Technological advances in fields such as Artificial Intelligence and computing have led to a growing interest in object identification in video. This is a research area within computer vision applications with numerous possibilities, such as object tracking, surveillance, or autonomous driving. In recent years, there has been a significant transformation in the way objects are detected, moving away from conventional methods to adopt more complex ones, such as Deep Learning.

Deep Learning use for object detection has become an standard because it achieves unprecedented results with higher accuracy than humans. However, in realistic environments, this performance drops drastically since the domain can change considerably and its training is done using academic datasets with little information close to reality. This needs domain adaptation from the model to handle new situations with different viewpoints of already known objects, or to detect new objects not considered in the training dataset.

This TFG addresses this problem by applying an incremental learning approach to adapt the model's domain and thus improve its detection performance in a new environment. An unsupervised pipeline that locates objects using a detection network, pre-calculated optical flow, and an image classifier that assigns classes to those objects were used. Subsequently, incremental learning was added to the image classifier to adapt its domain to the target task.

Several trainings have been carried out using a dataset that includes video sequences taken on the loading and unloading area of an airport. In this type of environment, appear new objects that are unknown, and the known ones do not appear in the foreground and or are even too small to be detected with traditional approaches.

Finally, an evaluation and comparison of the obtained results was developed.

Key Words: Incremental learning, Deep learning, Aprendizaje Incremental, Deep Learning, Convolutional neural networks

Índice

Índice de Figuras

Índice de Tablas

Acrónimos y Abreviaturas

AP	Average Precision
CNN	Convolutional Neural Network
CorLoc	Correct Location
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
iCaRL	Incremental Classifier and Representation Learning
IoU	Intersection over Union
mAP	mean Average Precisiom
NMC	Nearest Mean Classifier
OF	Optical Flow
OFRP	Optical Flow Region Proposal
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
RNN	Recurrent Neural Network
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
TFG	Trabajo Fin de Grado
YOLO	You Only Look Once

Capítulo 1

Introducción

Este primer capítulo sirve como prefacio de este trabajo; en él se explican diferentes aspectos como forma de contextualización. Se tratarán la motivación que ha impulsado a su realización, los objetivos que se quieren conseguir y la metodología utilizada para ello. Además, se presenta una breve descripción de la estructura del trabajo.

1.1. Motivación

La inteligencia artificial está a la orden del día, su uso está cada vez más integrado en la sociedad por sus numerosas aplicaciones, entre ellas, la automatización de procesos o acciones del día a día, como pueden ser las recomendaciones de contenido en plataformas de entretenimiento o los asistentes virtuales.

Entre sus aplicaciones más interesantes se encuentra el reconocimiento de objetos. El uso del Deep Learning es habitual en este campo, ya que en muchas ocasiones alcanza una precisión que puede superar a la de los humanos.

Sin embargo, en entornos realistas, este rendimiento puede disminuir debido a que los modelos utilizados son entrenados con bases de datos académicas, que contienen pocas situaciones que se acerquen a la realidad, y el dominio puede cambiar completamente. Como forma de mejorar el rendimiento, es necesaria una adaptación del dominio del modelo para que pueda manejar situaciones nuevas con diferentes puntos de vista de objetos ya conocidos, o al contrario, detectar objetos nuevos no considerados en la base de datos utilizada para el entrenamiento.

Este Trabajo de Fin de Grado (TFG) se centra en la detección de objetos en secuencias de vídeo tomadas con una cámara RGB en la zona de carga y descarga

del aeropuerto de Gdansk. En la figura 1.1 se puede observar un ejemplo de las imágenes utilizadas.

Figura 1.1: Imagen de una de las cámaras del aeropuerto de Gdansk

En este tipo de entornos, aparecen nuevos objetos que los enfoques tradicionales no son capaces de reconocer. Además, los objetos conocidos no siempre aparecen en primer plano e incluso, su tamaño es muy pequeño. Por lo tanto, se va a aplicar un enfoque de aprendizaje incremental para adaptar el dominio del modelo y mejorar así su rendimiento de detección en un entorno nuevo. Además, se utiliza información de flujo óptico para facilitar la localización de objetos pequeños en movimiento.

1.2. Objetivos

El objetivo principal de este TFG es investigar y desarrollar un modelo capaz de detectar objetos en secuencias de vídeo tomadas en la zona de carga y descarga de un aeropuerto utilizando aprendizaje incremental. Para ello, se pretenden desarrollar uno o varios modelos de Deep Learning para detectar objetos a partir de información RGB y flujo óptico.

Para conseguir el objetivo final, el trabajo desarrollado se ha dividido en una serie de objetivos secundarios:

- Diseño de un pipeline de detección de regiones de interés, extracción de características y agrupamiento no supervisado en secuencias de vídeo RGB.
- Incorporación de nuevas clases mediante etiquetado manual de grupos obtenidos por *clustering*.
- Implementar un proceso de entrenamiento incremental basado en la destilación de conocimiento y uso de memoria representativa con el fin de evitar el olvido catastrófico.

- Utilizar información combinada de imagen RGB y flujo óptico para mejorar la detección: la imagen RGB permite reconocer objetos grandes o en primer plano, que suelen ser fácilmente reconocibles por modelos tradicionales preentrenados, mientras que el flujo óptico facilita la localización de objetos pequeños en movimiento.

1.3. Plan de Trabajo

A continuación, se enumeran cada uno de los pasos que se establecieron previa realización de este trabajo para la ejecución del mismo:

1. Lectura de artículos seleccionados para entender la problemática a resolver con el trabajo.
2. Familiarización con las herramientas y adquisición de conocimientos en las ramas en las que se va a profundizar: Python, Machine Learning, Deep Learning, etc.
3. Preparación de la base de datos que se utilizará en el trabajo.
4. Estudio de las posibles arquitecturas que pueden ser aplicadas, tanto para la detección y clasificación de imágenes como para el aprendizaje incremental.
5. Redacción del TFG y presentación.

1.4. Estructura del documento

Este trabajo se ha estructurado en una serie de capítulos, seguidos por la bibliografía correspondiente. A continuación, se muestra un breve resumen de cada uno de los mismos:

En el Capítulo 1, se realiza una introducción del trabajo. En él se muestran la motivación, los objetivos y cómo se ha llevado a cabo el proceso para cumplirlos.

En el Capítulo 2, se presenta el estado del arte sobre la detección de objetos en secuencias de vídeo y el aprendizaje incremental.

En el Capítulo 3, se introducen las herramientas utilizadas para la realización del trabajo, el lenguaje de programación Python junto a las librerías más importantes utilizadas.

En el Capítulo 4, se presentan los fundamentos teóricos en los que se ha basado el TFG.

En el Capítulo 5, se muestra la metodología seguida para realizar el TFG.

En el Capítulo 6, se exponen los resultados experimentales que se han obtenido.

En el Capítulo 7, se recopilan las conclusiones extraídas de la realización del trabajo, así como una serie de posibles líneas futuras del mismo.

Capítulo 2

Estado del Arte

La detección de objetos en vídeo es un área de investigación dentro de las aplicaciones de la visión por computador con multitud de posibilidades, como pueden ser el seguimiento de objetos, la vigilancia o la conducción autónoma. En los últimos años, ha tenido lugar una transformación significativa en la forma en la que se detectan objetos, abandonando métodos convencionales para adoptar otros más complejos, como el Deep Learning. En este contexto, aparece el aprendizaje incremental como una técnica para mejorar la precisión y eficiencia de los modelos de detección de objetos en vídeo.

Desde los inicios del siglo XXI se ha abordado la detección de objetos en vídeo utilizando diferentes métodos que han sentado las bases para la investigación en este campo. Las primeras contribuciones se basaban en diferentes técnicas como el método de sustracción de fondo [2] o la utilización de métodos basados en el descriptor morfológico conocido como histograma de Gradientes Orientados (HOG) [3], que alimenta una máquina de vectores de soporte que permite realizar la clasificación del objeto deseado. Otro de los métodos convencionales estaba basado en la segmentación [4], haciendo uso de la información de color para detectar objetos.

Con la llegada del *Deep Learning* o Aprendizaje Profundo, la detección de objetos en vídeo ha sufrido grandes avances gracias a la creación de modelos altamente precisos y adaptables que proporcionan una detección más rápida y certera. Se han propuesto multitud de proyectos basados en el uso de Redes Neuronales Convolucionales (CNNs) que trabajan con diferentes datos de entrada, como por ejemplo, RGB o flujo óptico. Las CNNs son capaces de identificar y aprender características complejas de los objetos observados. Además, la utilización de técnicas ampliamente conocidas como el *fine-tuning* o ajuste fino en español, permiten que funcione de forma óptima y robusta para múltiples conjuntos de datos diferentes.

En [5] se propone el uso de redes neuronales convolucionales; en este caso, se

utilizan algoritmos de flujo óptico de los que se obtienen regiones de interés, es decir, regiones de la imagen donde se encuentran objetos y, posteriormente, utilizando dichas redes neuronales convolucionales, se obtiene información de los objetos detectados. Por otro lado, en [6] se utiliza el algoritmo YOLO (por sus siglas en inglés, *You Only Look Once*), caracterizado por la velocidad en la detección de objetos debido a que solo observa la imagen una sola vez.

En los últimos años, el uso de redes neuronales *Transformer* ha tomado fuerza debido a su excelente rendimiento en el procesamiento del lenguaje natural [7]. Estas redes han sido adaptadas para el procesamiento de imágenes, así como para tareas de reconocimiento y clasificación de objetos. Arquitecturas como ViT (Vision Transformer) utilizada en [8] o DeiT (Data-efficient Image Transformer) utilizada en [9] han alcanzado resultados competitivos e incluso superiores a los obtenidos por las redes convolucionales.

A pesar de la eficacia de los modelos de *Deep Learning* en la detección de objetos en vídeo, aparece cierta problemática a la hora de realizar un aprendizaje de manera incremental. Estos modelos carecen de capacidad para adaptarse a nuevas clases de objetos, así como a las posibles variaciones del entorno. Es por ello que el uso del aprendizaje incremental está en auge.

En [10, 11, 12] se utilizan una serie de métodos que permiten aprender de forma conjunta tanto características específicas de una tarea como los clasificadores asociados. Sin embargo, cuando se busca adaptar este enfoque al aprendizaje incremental, aparece el problema conocido como olvido catastrófico o *Catastrophic Forgetting*, en el cual el rendimiento sobre las clases previamente aprendidas se deteriora notablemente al introducir nuevas clases [13, 14, 15, 16].

Los primeros esfuerzos por abordar este fenómeno se centraron en redes conexionistas [13, 14, 15], un enfoque de redes neuronales artificiales inspirado en el funcionamiento del cerebro humano. Aunque fueron pioneras en el estudio del aprendizaje incremental, sus limitaciones estructurales y de capacidad las hacen poco adecuadas para enfrentar los retos actuales del *deep learning*, especialmente en aplicaciones complejas como la visión por computador. Con el tiempo, surgieron nuevas propuestas más acordes con este tipo de modelos. Un ejemplo destacado es el trabajo realizado en [16], donde se combina la pérdida estándar de entropía cruzada y la pérdida por destilación, originalmente concebida para transferir conocimiento entre diferentes redes neuronales [17]. En este caso, dicha pérdida se utiliza para mantener el rendimiento del modelo, incluso cuando se actualiza con nuevas muestras de entrenamiento.

Aunque esta estrategia logró mitigar parcialmente el olvido catastrófico, en particular en contextos simples donde las muestras antiguas y nuevas provienen de diferentes conjuntos de datos con poca confusión entre ellas, su rendimiento sigue siendo limitado. Una posible causa de esta limitación es la débil representación del

conocimiento relativo a las clases antiguas. Trabajos como [11] demostraron esta debilidad de [16], mostrando errores significativos en un escenario de aprendizaje secuencial continuo, y en particular cuando las muestras nuevas y antiguas provienen de distribuciones relacionadas.

Otros enfoques que utilizan la pérdida por destilación, como [18], proponen congelar algunas de las capas correspondientes al modelo original, limitando así su adaptabilidad a nuevos datos. En [19] se basan en el método de [16] mediante el uso de un autoencoder como mecanismo para retener el conocimiento de las tareas antiguas, en lugar de la pérdida por destilación. Este método también fue evaluado en un escenario restrictivo, donde las redes antiguas y nuevas se entrenan en diferentes conjuntos de datos. La pérdida por destilación también se utilizó para el aprendizaje incremental de detectores de objetos [20]. A pesar de los buenos resultados obtenidos en ese contexto específico, todavía no está claro si este tipo de arquitectura es adecuada para escenarios más generales de aprendizaje incremental.

Estrategias alternativas para reducir el olvido catastrófico han explorado distintas vías, entre ellas el aumento progresivo de capas en la red con el fin de aprender nuevas representaciones específicas para las clases añadidas [21, 22]. Otra propuesta es ralentizar la tasa de aprendizaje de manera selectiva mediante la regularización por parámetro [23]. Se utiliza un enfoque relacionado en [24] introducen una arquitectura estructurada en forma de árbol que se expande de manera incremental conforme se introducen nuevas clases. El principal inconveniente de todos estos enfoques es el rápido aumento en el número de parámetros, que crece con el número total de pesos, tareas y nuevas capas.

Un enfoque representativo en el ámbito del aprendizaje incremental es iCaRL[25], donde se desacoplan las tareas de aprendizaje del clasificador y de la representación de datos. Para clasificar nuevas muestras, iCaRL utiliza un NMC (Nearest Mean Classifier) tradicional, es decir, mantiene un conjunto auxiliar que contiene muestras de datos antiguas y nuevas. El modelo de representación de datos, que es una red neuronal estándar, se actualiza a medida que nuevas muestras están disponibles, usando una combinación de pérdidas de destilación y clasificación [17, 16].

En la actualidad, se han desarrollado nuevas técnicas que buscan superar las limitaciones de los enfoques clásicos del aprendizaje incremental mediante arquitecturas más adaptativas y eficientes. En [26] se introduce Efficient-CLS, un módulo que facilita la detección continua en vídeo con mínima supervisión, reduciendo la necesidad de anotaciones exhaustivas. En [27] presentan CL-DETR (Continual Learning Detection Transformer), una variante de los transformadores para detección incremental que combina una pérdida de destilación adaptativa con calibración posterior para preservar el conocimiento adquirido. Por otra parte, en [28] se propone PIVOT (Prompting for Video Continual Learning), una técnica basada en *prompts* y modelos multimodales preentrenados que permiten una actualización eficiente sin reentrenamiento completo. Por último, en [29] se presenta DyQ-DETR (Dynamic

Query DETR), una arquitectura que introduce consultas de objetos dinámicas en transformadores para detección incremental, permitiendo una expansión eficiente del modelo y reduciendo la interferencia entre clases antiguas y nuevas.

Capítulo 3

Herramientas Utilizadas

Este capítulo tiene el objetivo de explicar las herramientas empleadas para llevar a cabo este proyecto. Se realizará una introducción del lenguaje de programación Python y de algunas librerías utilizadas, así como del procesador utilizado.

3.1. Python

Python [30] es un lenguaje de programación interpretado ampliamente utilizado para inteligencia artificial. Una de sus principales características es que posee código abierto y es gratuito. Además, se trata de un lenguaje de programación multiparadigma, es decir, se utiliza para programación funcional, programación imperativa y soporta orientación a objetos. Otra particularidad es que se trata de un lenguaje interpretado, dinámico y multiplataforma. Este conjunto de características ha hecho posible el uso de Python en tareas de Inteligencia Artificial.

Como se ha mencionado anteriormente, Python es un lenguaje interpretado, es decir, que no necesita ser compilado, es por ello que permite su uso de forma directa. Posee características útiles para el procesamiento de datos, como pueden ser la manipulación de matrices, objetos y variables complejas. Además, cuenta con una serie de bibliotecas como TensorFlow, Numpy o Keras, utilizadas para la computación avanzada y el aprendizaje automático.

Python es compatible con todos los sistemas operativos y goza de una gran popularidad entre los desarrolladores a nivel mundial.

3.2. PyTorch

PyTorch [31] es una popular biblioteca tanto de procesamiento de datos como de aprendizaje automático. Una de sus características principales es su flexibilidad y capacidad para realizar cálculos en tensores; es por ello adecuado su uso en proyectos enfocados en la inteligencia artificial y el aprendizaje profundo.

Esta biblioteca está diseñada para aprovechar al máximo el poder de cálculo de unidades de procesamiento gráfico (GPUs), lo que permite procesar volúmenes altos de datos, así como acelerar el entrenamiento de modelos de *Deep Learning*. Además, PyTorch se integra bien con otras bibliotecas populares de Python, como NumPy o SciPy.

Pytorch es una biblioteca perfecta para el aprendizaje automático, que ofrece facilidad de uso, flexibilidad y un rendimiento excelente, lo que la convierte en una opción popular entre los investigadores y desarrolladores.

3.3. Servidor

El servidor utilizado pertenece al departamento de Arquitectura de computadores de la Universidad de Málaga. Ha sido diseñado para tareas intensivas de computación y procesamiento de datos. En su núcleo, cuenta con un procesador Intel(R) Xeon(R) Silver 4314 que ofrece 64 núcleos funcionando a una frecuencia de 2.40 GHz [32]. Este procesador de alto rendimiento está optimizado para manejar cargas de trabajo multitarea y aplicaciones que requieren un procesamiento paralelo robusto, ideal para entornos de servidor como es su caso.

Además, está equipado con una RAM de 500 GB. Esta amplia memoria permite manejar aplicaciones y *datasets* extremadamente grandes, ofreciendo un rendimiento fluido y eficiente incluso bajo las cargas de trabajo más demandantes. La capacidad de RAM es beneficiosa para tareas de análisis de datos, simulaciones complejas y aplicaciones de inteligencia artificial que requieren un acceso rápido y simultáneo a grandes volúmenes de datos.

El servidor incluye cuatro tarjetas gráficas NVIDIA 3090 [33], cada una con 24 GB de memoria dedicada. Estas GPUs de última generación están diseñadas para aplicaciones de *Deep Learning*, renderizado 3D y simulaciones científicas. Permiten un procesamiento paralelo eficiente, acelerando significativamente el tiempo de cómputo para tareas que van desde el entrenamiento de modelos de inteligencia artificial hasta la edición de vídeo en alta resolución.

Capítulo 4

Fundamentos Teóricos

En este capítulo se presentan los fundamentos teóricos que se han tomado como base de este TFG, y que se han utilizado para su ejecución. Todos ellos giran en torno al Aprendizaje Incremental y las Redes Neuronales Convolucionales (CNNs).

4.1. Introducción a las redes neuronales

Las redes neuronales artificiales son una herramienta fundamental en los campos de *Machine Learning* y *Deep Learning*. Están formadas por capas de entrada y salida y, comúnmente, por capas ocultas que poseen unidades que transforman la entrada en información utilizada a la salida. Existen diversos tipos de redes neuronales, cada una con diferente complejidad y funcionalidad. Algunas de las principales son las siguientes:

- **Redes neuronales *feed-forward*** o **prealimentadas** en español. Se trata de las primeras redes creadas, y además, son las más simples. En estas redes, la información fluye en una única dirección, de la entrada a la salida. Las redes neuronales prealimentadas incluyen varias clases como son el perceptrón simple o el perceptrón multicapa.
- **Redes neuronales recurrentes (RNNs)**. Se conocen por su capacidad de transmisión de datos en varias direcciones, debido a que integran bucles de realimentación que permiten que la información no desaparezca durante algunas épocas de entrenamiento. Gracias a su capacidad de aprendizaje, estas redes se utilizan para tareas complejas como pueden ser el reconocimiento de escritura o lenguaje.

- **Redes neuronales convolucionales (CNNs)**. Se trata de un tipo de red neuronal multicapa cuya arquitectura permite identificar las características más complejas de los datos. Algunas de sus utilidades más comunes son el reconocimiento de imágenes, en vehículos autónomos o reconocimiento facial, entre muchas otras.

4.2. Redes neuronales convolucionales

Las CNNs son redes neuronales profundas diseñadas específicamente para procesar imágenes mediante el uso de filtros o convoluciones, para extraer características. En las capas iniciales, se capturan características de bajo nivel como bordes, esquinas y otros patrones simples. Por otro lado, las capas más profundas se especializan en identificar características de alto nivel como las ruedas del avión o pequeños objetos, como se ilustra en la Figura 4.1.

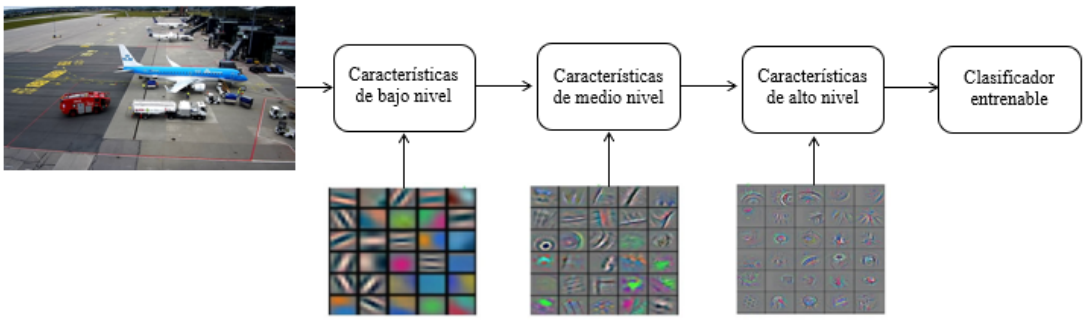

Figura 4.1: Representación gráfica del proceso de extracción de características de bajo, medio y alto nivel en una CNN. Fuente: [1]

Para iniciar el entrenamiento de una CNN, es crucial seguir una serie de pasos previos. Primero, es fundamental asegurarse de que el conjunto de datos utilizado esté adecuadamente normalizado. Además, se debe definir la arquitectura de la red, incluyendo el número y tipo de capas, así como la elección de la función de pérdida o función de *loss* en inglés, y la configuración de hiperparámetros. La selección del optimizador apropiado también es crucial para el éxito del entrenamiento. Estos aspectos se abordarán detalladamente en la Sección 4.2.1 de este capítulo.

Una vez configurados los parámetros que caracterizan la red, se procede con el entrenamiento. Este proceso implica varias etapas hasta que la red converge. Inicialmente, se introducen los datos en la red para calcular las activaciones y determinar el error de pérdida (*loss*) . Luego, se lleva a cabo la propagación hacia atrás (*back-propagation*) para calcular las derivadas parciales de cada parámetro (los pesos de la red) con respecto al error cometido. Estas derivadas indican la dirección y el valor

en el cual cada parámetro debe actualizarse para minimizar el error anterior. Este ciclo se repite iterativamente hasta que la red converja.

4.2.1. Arquitectura de las CNNs

Las redes neuronales convolucionales están formadas, a grandes rasgos, por capas, funciones de activación y funciones de pérdida, entre otros aspectos.

4.2.2. Capas

El número y tipo de capas utilizadas va a definir el comportamiento del modelo, los parámetros requeridos, así como la capacidad de aprendizaje de la red. A continuación, se explican las capas más utilizadas a la hora de definir las CNNs.

4.2.2.1. Capa Convolucional

Se trata de la capa más importante de las CNNs y la que realiza la mayor parte de los cálculos. Su función es transformar los datos de entrada utilizando la operación matemática de convolución. Esta operación se realiza tomando un grupo de píxeles de la imagen de entrada y posteriormente, ir realizando un producto escalar con un *kernel* o filtro. Este procesará todos los datos de entrada, generando un mapa de activación de dos o tres dimensiones, que servirá como entrada de la siguiente capa.

Los parámetros que definen una capa convolucional son los siguientes:

- Profundidad (*depth*): Indica la cantidad de filtros en una capa convolucional. Cada filtro genera un mapa de activación que se apila para formar el volumen total de activación.
- Relleno (*padding*): Especifica la cantidad de ceros añadidos alrededor de la muestra de entrada. Controla el tamaño del volumen de salida después de aplicar la convolución.
- Paso (*stride*): Determina el desplazamiento de la ventana deslizante sobre los datos de entrada. Valores mayores resultan en menos características locales capturadas en la salida.
- Tamaño del kernel: Define las dimensiones del filtro, especificando tanto el ancho como el alto del área de interacción con los datos de entrada.

Cada filtro en una red convolucional está representado por una neurona cuyos pesos definen el *kernel*. Durante la convolución, estas neuronas se distribuyen a lo largo de las dos dimensiones de la entrada para calcular simultáneamente características locales. Esta distribución implica la optimización de múltiples parámetros, lo cual puede llevar a problemas de sobreajuste (*overfitting*). Para evitar este problema, se reduce el número de parámetros asumiendo que si un kernel es efectivo en una posición (*x, y*), también lo será en otras posiciones (*x2, y2*). De esta manera, todas las neuronas dentro de un mismo kernel comparten los mismos pesos, reduciendo así la cantidad total de parámetros entrenables en la red.

4.2.2.2. Capa de Muestreo o *Pooling Layer*

Esta capa es la encargada de reducir el tamaño de los datos de entrada para así reducir el número de parámetros de la red y prevenir el sobreajuste. Comúnmente se sitúa detrás de las capas convolucionales para reducir el tamaño de los datos de forma progresiva. Su cálculo se realiza de forma similar al de la capa convolucional. Se ejecuta una operación sobre la entrada utilizando una ventana deslizante, lo que genera un volumen de salida. En este proceso, la operación produce un único valor para cada posición de la ventana, lo que resulta en que el tamaño final de la salida sea menor que el de la entrada original.

Los parámetros que definen las capas de *pooling* son los siguientes:

- Paso (*stride*): Indica el tamaño del desplazamiento de la ventana deslizante sobre los datos de entrada. Valores mayores resultan en la captura de menos características locales en la salida.
- Tamaño de la ventana: Determina las dimensiones de la ventana utilizada para realizar la operación sobre los datos de entrada, especificando tanto el ancho como el alto.
- Operación: Define la operación realizada sobre la ventana durante el proceso. La operación más común es el máximo (*max*), aunque también existen otras como el mínimo (*min*), la media (*mean*), L2, entre otras.
- Relleno (*padding*): Especifica la cantidad de ceros añadidos alrededor de la muestra de entrada para controlar el tamaño del volumen de salida después de aplicar la operación.

4.2.2.3. Capa Completamente Conectada o *Fully-Connected Layer*

Se trata de uno de los elementos más importantes tanto de las redes neuronales tradicionales como de las redes convolucionales. En esta capa, a diferencia de las ca-

pas convolucionales que solo tienen conexiones locales, cada neurona está conectada a todas las activaciones de la capa anterior. Esta capa está situada en las últimas capas de la red y posee los parámetros de la misma. Además, almacena información de alto nivel que será utilizada por el clasificador.

Dada la alta cantidad de parámetros en las capas completamente conectadas, es crucial limitar su número para evitar el sobreajuste. Usualmente, estas capas están precedidas por una capa de *flatten*, que extrae y transforma las salidas de las capas anteriores en un único vector, que será la entrada de la siguiente capa.

4.2.2.4. Capa *Dropout*

El *dropout* es una estrategia de regularización utilizada en redes neuronales para mitigar el sobreajuste. Se implementa en las capas conectadas a la capa *Dropout*. Durante el entrenamiento, algunas neuronas individuales se descartan de manera aleatoria, lo cual reduce la dependencia entre ellas y ayuda a prevenir el sobreajuste. Por el contrario, durante la fase de evaluación (*test*), el *dropout* no se aplica y se consideran todas las neuronas activas.

Esta técnica se suele emplear después de las capas completamente conectadas, ya que estas capas tienden a tener más parámetros y, por lo tanto, son más susceptibles al sobreajuste..

4.2.3. Funciones de activación

Las funciones de activación son elementos fundamentales en las redes neuronales, ya que introducen la no linealidad necesaria para así aprender patrones complejos. Se añaden tras una capa convolucional. Algunas de las funciones más comunes son:

- **Softmax**: convierte los valores de entrada en activaciones dentro del rango [0, 1], de manera que la suma total sea 1. Esto permite interpretarlos como una distribución de probabilidad, donde cada valor representa la probabilidad de que la entrada pertenezca a una clase específica, Esta función se emplea habitualmente como activación en la capa de salida de redes neuronales destinadas a tareas de clasificación. Su formulación matemática es la siguiente:

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{4.1}$$

- **Sigmoide**: convierte los valores de entrada en activaciones que se sitúan dentro del intervalo [0, 1]. Sin embargo, cuando estos valores se acercan demasiado

a los extremos 0 o 1, surge un inconveniente: el gradiente en esas regiones se vuelve casi nulo. Esto supone que durante la propagación hacia atrás (*backpropagation*), los gradientes tienden a cero y la convergencia se verá penalizada. Además, la inicialización de parámetros desempeña un papel importante en esta función para evitar que la función se sature. La función viene representada por la siguiente ecuación:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

- **ReLU (*Rectified Linear Unit*)**: convierte los valores de entrada en activaciones dentro del rango [0, ∞]. Esta función es ampliamente empleada en la actualidad, ya que favorece una convergencia más rápida y su cálculo resulta más eficiente en comparación con otras funciones de activación. La función se define mediante la siguiente ecuación:

$$f(x) = \max(0, x) \tag{4.3}$$

- **Tanh**: convierte los valores de entrada en activaciones dentro del rango [-1, 1]. Aunque sus activaciones pueden saturarse, su uso es preferido porque está centrada en el cero. La función se representa mediante la siguiente ecuación:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4.4}$$

A continuación, en la Figura 4.2 se muestra la representación gráfica de las funciones de activación, exceptuando la función Softmax.

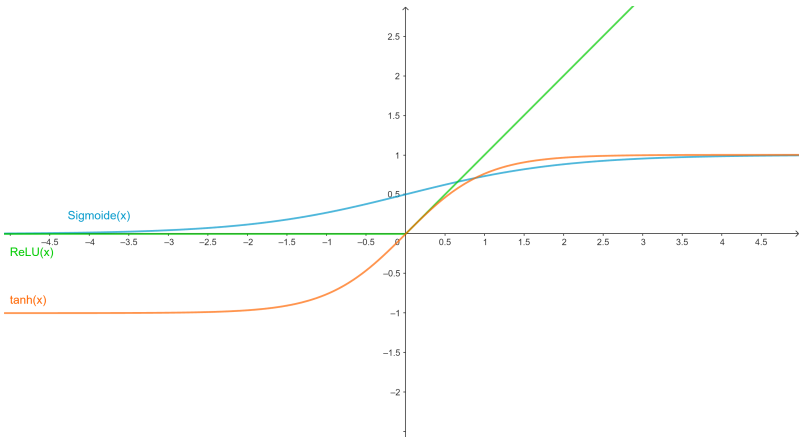

Figura 4.2: Funciones de activación. Sigmoide en azul, Tanh en naranja y ReLU en verde.

4.2.4. Funciones de pérdida

Las funciones de pérdida (*loss functions*), también conocidas como funciones de coste, son herramientas que calculan un valor numérico que representa el error entre la salida del modelo y el resultado esperado. Durante el proceso de entrenamiento, el objetivo del optimizador es reducir este error, ajustando así el modelo para mejorar su precisión. En problemas de clasificación, la función de pérdida cuantifica la penalización por las predicciones incorrectas, ayudando a guiar el ajuste del modelo hacia predicciones más precisas.

A continuación, se explican algunas de las funciones de pérdida más importantes.

4.2.4.1. Función de Pérdida de Entropía Cruzada (*Cross-Entropy Loss Function*)

La función de pérdida de entropía cruzada evalúa la discrepancia entre dos distribuciones de probabilidad: la predicha por el modelo y la distribución real. En esta función, la distribución real o la de etiquetas, se representa mediante un vector *one-hot*. Este vector tiene un tamaño igual al número de clases, con un único valor de 1 en la posición que corresponde a la clase correcta, y 0 en todas las demás posiciones.

La pérdida de entropía cruzada para un problema de clasificación con múltiples clases se calcula con la siguiente fórmula:

$$\mathcal{L}_{\mathcal{C}}(p, q) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} p_{ij} \log q_{ij} \tag{4.5}$$

donde N es el número total de muestras, C es el número total de clases, p_i es el valor real de la muestra i y q_i es la probabilidad que se obtiene aplicando la función *softmax* a la salida de una capa de clasificación para la muestra i.

4.2.4.2. Función de Pérdida de Destilación de Conocimiento (*Knowledge Distillation Loss Function*)

La destilación de conocimiento es un método para comprimir modelos, donde se transfiere el conocimiento de un modelo grande a uno más pequeño.

La pérdida de destilación se define con la siguiente ecuación:

$$L_D(\omega) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} pdist_{ij} \log qdist_{ij} \tag{4.6}$$

donde N y C representan el número de muestras y clases, respectivamente. El término $pdist_i$ es una versión modificada de la distribución p_i y $qdist_i$ de q_i. Estas distribuciones se obtienen aplicando una función *Softmax* modificada (Sección 4.2.3), donde se incluye un parámetro de destilación T dividiendo a las activaciones x_j y x_k.

Cuando $T = 1$ la función coincide con la función de pérdida de entropía cruzada (Sección 4.2.4.1). En este caso, la clase con la puntuación más alta domina el valor de la pérdida, mientras que las clases restantes contribuyen en menor medida. Sin embargo, cuando $T > 1$ las clases menos probables influyen más, ya que sus valores de pérdida son más altos y deben ser minimizados. Esto motiva a la red a aprender una separación más precisa entre las clases.

4.2.4.3. Función de Pérdida de Destilación de Conocimiento cruzada (*Cross-Distilled Loss Function*)

Esta función de pérdida combina una pérdida por destilación de conocimiento (Sección 4.2.4.2), con una pérdida de entropía cruzada multiclase (Sección 4.2.4.1). La pérdida por destilación se aplica a las capas de clasificación de las clases antiguas, mientras que la entropía cruzada multiclase se usa en todas las capas de clasificación. Esto permite al modelo actualizar los límites de decisión de las clases.

La pérdida de destilación de conocimiento cruzada está definida por la siguiente ecuación:

$$L(\omega) = L_C(\omega) + \sum_{j=1}^{F} L_{D_f}(\omega) \tag{4.7}$$

donde F es el número total de capas de clasificación correspondientes a las clases antiguas, $L_{D_f}(\omega)$ representa la pérdida por destilación aplicada en la capa de clasificación f, y $L_C(\omega)$ es la pérdida de entropía cruzada utilizada para las muestras tanto de las clases antiguas como de las nuevas.

Esta función de pérdida se utiliza de forma habitual como forma de evitar la pérdida de conocimiento dramática [13, 14].

4.2.5. Optimizadores

Este algoritmo se encarga de optimizar los pesos del modelo durante el entrenamiento de la red neuronal. Este proceso se lleva a cabo calculando los gradientes de una función de pérdida y aplicando estos gradientes para la actualización de parámetros del modelo.

4.2.5.1. Descenso del gradiente estocástico (*Stochastic Gradient Descent (SGD)*)

Se trata del algoritmo más popular para el proceso de optimización. Su objetivo es minimizar una función objetivo $J(\theta)$, donde $\theta \in \mathbb{R}^d$ son los pesos del modelo. Este método actualiza los pesos en la dirección contraria al gradiente de la función objetivo $\nabla_\theta J(\theta)$.

En el ámbito de los algoritmos de descenso del gradiente, el más utilizado para la optimización de modelos de aprendizaje profundo es el descenso del gradiente por mini-lotes (*mini-batch gradient descent*). Este método actualiza los pesos del modelo utilizando mini-lotes de tamaño n extraídos del conjunto de entrenamiento, conforme a la Ecuación 4.8. Esta estrategia permite reducir la varianza de las actualizaciones, lo que contribuye a una convergencia más estable y eficiente.

$$\theta \leftarrow \theta - \alpha \sum_{i=1}^{n} (h_\theta(x_i) - y_i) x_i \tag{4.8}$$

donde n es el tamaño del mini-lote, α representa la tasa de aprendizaje, y_i es la etiqueta correspondiente a la muestra i, $h_\theta(x_i)$ es la salida del modelo para la entrada x_i, y θ son los parámetros del modelo.

4.2.5.2. Adam (*Adaptive Moment Estimation*)

Se trata de un método de optimización que ajusta la tasa de aprendizaje para cada peso del modelo. Este algoritmo mantiene un promedio exponencialmente decreciente de los cuadrados de los gradientes pasados (v_t). Además, también guarda un promedio exponencialmente decreciente de los gradientes pasados (m_t), que funciona de manera similar al momento. Las ecuaciones 4.9 y 4.10 se usan para calcular m_t y v_t respectivamente.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.9}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.10}$$

donde g_t representa los gradientes actuales y g_t^2 los gradientes actuales al cuadrado. Los parámetros mt y v_t son las estimaciones del primer momento (media) y del segundo momento (varianza), respectivamente. m_{t-1} y v_{t-1} corresponden al promedio descendente de los gradientes pasados y al promedio descendente de los gradientes pasados al cuadrado. Finalmente, β_1 y β_2 son constantes que controlan la tasa de decaimiento exponencial de estas estimaciones.

La ecuación que define la regla de actualización de Adam es la siguiente:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.11}$$

donde α es la tasa de aprendizaje, $\hat{v}_t$ representa la estimación corregida del sesgo del segundo momento, $\hat{m}_t$ es la estimación corregida del sesgo del primer momento, ϵ es un término de suavizado que evita divisiones por cero, θ_t son los parámetros actuales y θ_{t+1} son los parámetros actualizados en el momento $t + 1$.

4.2.6. Hiperparámetros

Los hiperparámetros son variables cuya función es definir conceptos de alto nivel de las redes. Estos parámetros se definen antes del entrenamiento e inicialmente no se puede conocer su valor para un problema determinado; es por ello que se utiliza la validación cruzada para buscarlos.

Algunos de los hiperparámetros más comunes son:

- **Arquitectura de la Red.** Incluye el diseño de capas, tipos específicos de capas y el número de filtros en cada una de ellas.
- **Tasa de Aprendizaje (*Learning Rate*).** Define la velocidad de ajuste de los pesos durante el entrenamiento. Una tasa baja puede llevar a una convergencia lenta o evitar el mínimo local, mientras que una alta puede acelerar la convergencia pero corre el riesgo de no converger adecuadamente.
- **Decaída del peso (*Weight Decay*).** Se trata de un parámetro que provoca que los pesos disminuyan exponencialmente si no se aplican otras actualizaciones. Contribuye a mitigar el sobreajuste.
- **Tamaño del Lote (*Batch Size*).** Indica cuántas muestras se introducen en la red en cada iteración de entrenamiento.

- **Criterio de Parada (*Stop Criterion*).** Se trata del criterio utilizado para finalizar el entrenamiento de la red neuronal.
- **Tasa de *dropout* (*Dropout Rate*).** Porcentaje de neuronas o nodos que se eliminan de manera temporal y aleatoria durante el entrenamiento para prevenir el sobreajuste.

4.3. Introducción al aprendizaje incremental

El aprendizaje incremental es un paradigma del aprendizaje automático donde el proceso de aprendizaje tiene lugar en cualquier momento en el que aparecen nuevos datos de entrada, ajustando el conocimiento aprendido en función de esta nueva información. La diferencia más notable del aprendizaje incremental respecto al *Machine Learning* tradicional es que no asume la disponibilidad de un conjunto de entrenamiento suficiente antes del proceso de aprendizaje, sino que el conjunto se va ampliando con el tiempo [34].

Este tipo de aprendizaje obtiene su esencia de la forma en la que los humanos aprendemos. Mediante la observación, extraemos nuevas características y de esta manera le vamos dando una forma más precisa a nuestro conocimiento.

Existen diferentes técnicas de aprendizaje incremental que utilizan diversas técnicas de clasificación de datos; a continuación, se mencionan algunas de ellas:

- **Bosques aleatorios incrementales**. Un bosque aleatorio es un clasificador que consiste en una colección de clasificadores estructurados en árboles

  $$h(\mathbf{x}, \ominus_k), k = 1, ...$$

  donde los $\ominus_k$ son vectores aleatorios independientes e idénticamente distribuidos y cada árbol emite un voto unitario para la clase más popular en la entrada $\mathbf{x}$ [35].

  Los bosques aleatorios incrementales son extensiones de los bosques aleatorios estándar que permiten la actualización del modelo a medida que se reciben nuevos datos, sin necesidad de reconstruir completamente el modelo desde cero. Combinan la capacidad de los árboles aleatorios para manejar grandes conjuntos de datos, con la flexibilidad de actualización que aporta el aprendizaje incremental.

  Ante nuevos datos de entrada, para cada árbol en el bosque, se evalúa el nuevo conjunto de datos y se actualiza la estructura del árbol según sea necesario. Esto puede implicar ajustar los umbrales de división de los nodos existentes o agregar nuevos nodos para acomodar la nueva información [36].

- ***Support Vector Machines* (SVM) o Máquinas de vectores de soporte incrementales**. El objetivo del algoritmo SVM es encontrar un hiperplano que separe de la mejor forma posible dos clases de puntos de datos diferentes. Se busca el hiperplano con el margen más amplio entre las dos clases. Solo puede encontrarse este hiperplano en problemas que permiten separación lineal [37].

  Los SVM incrementales permiten adaptar el modelo ante nuevos datos de entrada de forma continua sin necesidad de entrenar desde cero todo el conjunto de datos.

  La actualización incremental se realiza ajustando los parámetros del SVM para incorporar nuevos datos. Esto incluye métodos como el descenso de gradiente estocástico para SVM lineales o variantes de la optimización secuencial mínima para SVM no lineales, que permiten ajustar los parámetros del modelo de manera eficiente con cada nueva entrada de datos [38].

- **Redes neuronales incrementales**. Las redes neuronales incrementales, también conocidas como continuas o adaptativas, son un modelo de *Deep Learning* diseñado para aprender a medida que se recibe nueva información, sin tener que entrenar desde cero el modelo. Se caracterizan por su eficiencia, flexibilidad y adaptabilidad.

Capítulo 5

Metodología

En este capítulo se expone la metodología que se ha seguido para la realización de los experimentos de este TFG en los que se aborda la detección de objetos en vídeo utilizando aprendizaje incremental.

5.1. Sistema de detección con integración de aprendizaje incremental

En este apartado se realiza una descripción en detalle de cada elemento del sistema propuesto. El flujo de datos se observa en la Figura 5.1

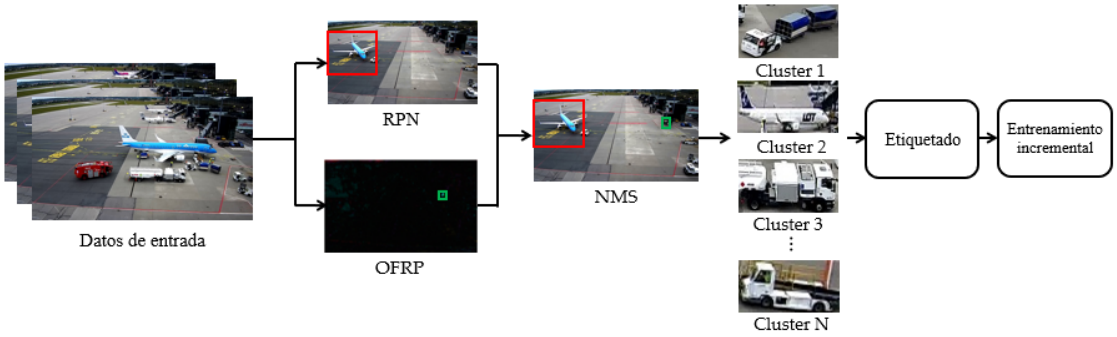

Figura 5.1: La entrada es un conjunto de frames de vídeos RGB. La localización de objetos se realiza mediante el uso de la red de propuesta de regiones (RPN) y el método de propuesta de regiones usando Flujo óptico (OFRP). Se aplica supresión de no máximos (NMS) para evitar repeticiones. Posteriormente, se realiza el proceso de *clustering*. Se etiquetan los *clusters*. Finalmente, se realiza un entrenamiento incremental del clasificador.

5.1.1. Datos de Entrada

Teniendo en cuenta el tamaño de entrada requerido por la RPN, los fotogramas originales del vídeo se han redimensionado a 640×384 píxeles. Por otro lado, el flujo óptico se calcula con una resolución de 720×405 píxeles. Ambas resoluciones se han tomado de [39] [40], donde se establecen como valores óptimos para equilibrar precisión y eficiencia computacional.

5.1.2. Propuesta de regiones

En el sistema propuesto se combinan dos algoritmos para la generación de propuestas de regiones, con el objetivo de capturar un mayor número de regiones de interés. La red de propuesta de regiones (RPN) resulta eficaz detectando objetos grandes y bien definidos, mientras que el método basado en flujo óptico (OFRP) es más adecuado para detectar objetos pequeños o sutiles que se encuentran en movimiento.

- **Red de propuesta de regiones (RPN):** Se utiliza una *Region Proposal Network* (RPN) o red de propuesta de regiones en Español, en este caso una YOLOv4 preentrenada. Además, para mejorar el rendimiento se utiliza ResNet50 como *backbone* (red encargada de extraer las características visuales). Esta configuración se basa en el trabajo realizado en [40].
- **Propuesta de regiones usando Flujo óptico (OFRP):** Para generar propuestas de regiones basadas en flujo óptico (OF), se emplean mapas de flujo calculados utilizando la técnica de *Farneback Dense* [41]. Dichos mapas F_t se calculan cada 5 fotogramas. Con el fin de eliminar el ruido que puede surgir por variaciones en las condiciones del escenario, se anulan aquellas posiciones en las que las componentes x e y del flujo óptico sean inferiores a un umbral T_f. Posteriormente, se encuentran los contornos que contienen los objetos de la escena utilizando el algoritmo propuesto en [42]. Finalmente, se eliminan aquellas regiones propuestas cuya área sea menor que un umbral T_A, para evitar regiones excesivamente pequeñas.

5.1.3. Supresión de no máximos

Para evitar localizaciones duplicadas de un mismo objeto que han sido detectadas por la RPN y por el OFRP simultáneamente, es necesario combinar ambas localizaciones. Para ello, se calcula la métrica de intersección sobre la unión o *Intersection over Union* (IoU) en inglés, entre ambas regiones propuestas. Si la IoU

es mayor que un umbral determinado T_I y la relación de aspecto de la región más grande dividida entre la del más pequeño es mayor que un umbral T_{AR}, se mantiene la región propuesta por el método RPN debido a que es más fiable que la región propuesta por el método OFRP. Además, se aplica supresión de no máximos a cada algoritmo como forma de eliminar regiones repetidas en las que su IoU es mayor que el umbral T_I. Una vez finalizado este paso, se tiene el conjunto de regiones que utilizará el algoritmo de *clustering*.

5.1.4. Extracción de descriptores

El paso previo al *clustering* es la representación mediante descriptores de las regiones detectadas en el paso anterior. Para describir las regiones se utiliza un modelo ResNet-50 [43] preentrenado, donde los descriptores son proporcionados por las activaciones del *Average Pooling* antes de la capa de clasificación.

Una vez que se han obtenido las características de cada región, se normalizan mediante la norma L2 para homogeneizar la escala de los vectores. A continuación, se reduce su dimensionalidad a 128 con el algoritmo UMAP [44]. Esta técnica resulta eficaz, ya que logra conservar la estructura global de los datos, al tiempo que mantiene la proximidad entre vecinos a nivel local.

Posteriormente, en los siguientes ciclos de entrenamiento, no se utiliza la ResNet50 inicial preentrenada, sino que se utiliza el modelo ResNet50 previamente afinado de forma incremental. De esta forma, se transmite el conocimiento acumulado en etapas anteriores, parte esencial en el proceso de aprendizaje incremental.

5.1.5. Clustering

Este paso tiene como finalidad agrupar aquellas regiones que tienen características similares en *clusters*. De este modo, en lugar de entrenar un clasificador que asigne etiquetas a cada región en función de sus características, se asigna el *cluster* más cercano a cada una de ellas.

A continuación, las representaciones resultantes se utilizan como entrada para un algoritmo de *clustering*. Para ello, se recurre a HDBSCAN [45], un método capaz de identificar *clusters* con formas y densidades variadas, mostrando un buen rendimiento incluso en contextos complejos, algo esencial en este trabajo.

5.1.6. Proceso de etiquetado

El siguiente paso del proceso es el etiquetado de *clusters*. Para ello, de forma manual, se observa un conjunto de N muestras pertenecientes a cada *cluster* y se determina cuál es su etiqueta correspondiente. Para etiquetar un *cluster*, al menos, la mitad de las muestras deben pertenecer a una determinada clase, de esta forma, se crea un sistema robusto ante la presencia de muestras atípicas. Si un *cluster* solo contiene muestras erróneas, se elimina por completo.

5.1.7. Entrenamiento incremental

Una vez obtenidos los *clusters* correctamente etiquetados y clasificados, se procede al entrenamiento incremental del modelo clasificador que se ha basado en el trabajo realizado en [46].

5.1.7.1. Memoria representativa

En el momento en el que se añade una nueva clase al modelo actual, se recoge un conjunto de las muestras más representativas y se almacena en la memoria representativa. Se considera una memoria con una capacidad limitada de muestras K. Dado que esta capacidad no depende del número de clases, a medida que se incrementa la cantidad de clases almacenadas, disminuye el número de muestras disponibles por cada una. El número de muestras por clase, n, se obtiene utilizando la Ecuación 5.1, donde K es la capacidad de la memoria y c es el número de clases almacenadas en la memoria.

$$n = \frac{K}{c} \tag{5.1}$$

La selección de muestras se realiza utilizando el método de selección por agrupamiento conocido como *herding* en inglés, que ordena los ejemplos de cada clase en función de su proximidad al vector medio de la clase. Posteriormente, se seleccionan los primeros n ejemplos, considerados como los más representativos del conjunto. Esta estrategia ha sido elegida en base al trabajo realizado en [46].

Una vez finalizado el entrenamiento, se realiza un proceso de eliminación de muestras de la memoria para así liberar espacio para las nuevas muestras. Debido a que las muestras se almacenan en una lista, la memoria elimina muestras del final del conjunto perteneciente a cada clase.

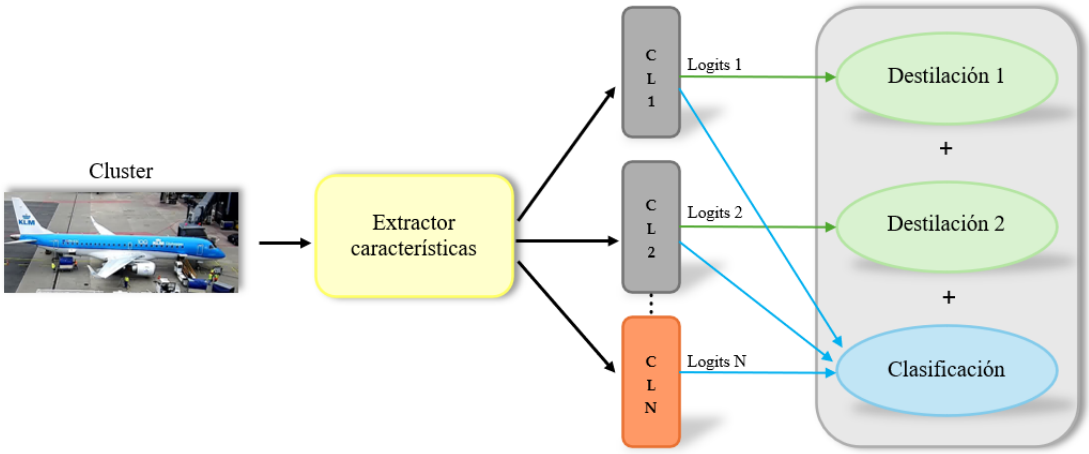

Figura 5.2: Se introduce una imagen de entrada, de la que se extraen características que se utilizan en los bloques de clasificación para generar un conjunto de *logits*. Las capas de clasificación en gris contienen clases antiguas y sus *logits* se utilizan para destilación y clasificación. La capa de clasificación en naranja contiene nuevas clases y sus *logits* toman parte únicamente en la clasificación

5.1.7.2. Arquitectura de la red

La arquitectura de la red de clasificación está compuesta por una serie de elementos, como queda reflejado en la Figura 5.2. El primero de ellos es el extractor de características, compuesto por un grupo de capas cuya función es transformar la imagen de entrada en un vector de características. A continuación, se encuentra la capa de clasificación; se trata de la última capa completamente conectada del modelo, con la misma cantidad de salidas que el número de clases con el que se trabaja. Dicha capa extrae las características y produce un conjunto de *logits*. En la fase de entrenamiento, los gradientes utilizados para actualizar los pesos de la red se calculan a partir de la combinación de una pérdida de clasificación basada en los *logits* y una pérdida de destilación de conocimiento cruzada, explicada en la sección 4.2.4.3.

5.1.7.3. Aprendizaje incremental

Este proceso consta de cuatro etapas, como se puede observar en la Figura 5.3. A continuación, se explica detalladamente cada una de ellas:

- **Construcción del conjunto de entrenamiento:** El conjunto de entrenamiento está formado tanto por muestras de las nuevas clases como por muestras de las clases antiguas que se almacenan en la memoria representativa. El modelo utiliza dos funciones de pérdida, clasificación y destilación, por lo tanto,

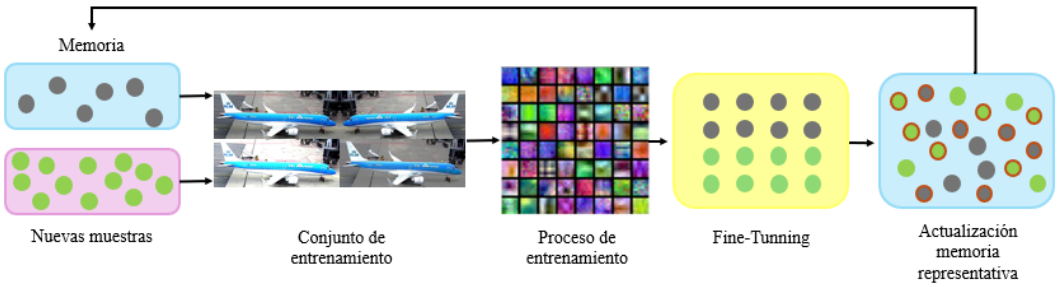

Figura 5.3: Entrenamiento utilizando aprendizaje incremental. Los puntos grises corresponden a las muestras de la memoria representativa. Los puntos verdes corresponden a muestras de las clases nuevas. Los puntos rodeados en rojo corresponden a las muestras finalmente seleccionadas para ser almacenadas en la memoria.

se le aplican dos etiquetas asociadas a dichas pérdidas, a cada muestra. Para la clasificación, se utiliza el vector *one-hot* que indica la clase que aparece en la imagen. En cambio, para la destilación,los *logits* generados por cada una de las capas de clasificación que contienen clases ya conocidas, se utilizan como etiquetas. Esto implica que, por cada muestra, se generan tantas etiquetas de destilación como capas de clasificación con clases previas haya en el modelo.

Con el objetivo de preservar el conocimiento adquirido en fases anteriores, incluso las muestras pertenecientes a nuevas clases son utilizadas en el proceso de destilación. De esta manera, todas las imágenes producen gradientes para ambas pérdidas. Así, al evaluar una imagen, la salida del modelo refleja el comportamiento de los pesos en cada capa, sin depender de la etiqueta concreta de esa imagen. Este procedimiento de extracción de etiquetas se repite en cada paso incremental.

- **Proceso de entrenamiento:** La función de Pérdida de Destilación de Conocimiento cruzada explicada en la Sección 4.2.4.3 toma el conjunto de entrenamiento obtenido utilizando *Data Augmentation*, con su etiquetado correspondiente y produce un conjunto de gradientes para optimizar el modelo. Durante el entrenamiento, todos los pesos del modelo se actualizan, de esta forma, las características obtenidas a partir del extractor de características varían en cada paso del aprendizaje incremental. Esto provoca que las capas de clasificación deben adaptar sus pesos para lidiar con las nuevas características.

- **Fine-Tunning:** Debido a que no se conservan todas las muestras de las clases antiguas, estas están subrepresentadas en comparación con las muestras de las clases nuevas. Para corregir este desequilibrio, se incorpora una etapa adicional de ajuste fino o *fine-tunning* en inglés, utilizando una tasa de aprendizaje reducida y un subconjunto de datos balanceado.

  Este nuevo subconjunto de entrenamiento, contiene el mismo número de muestras por clase, independiente de si pertenecen a clases nuevas o antiguas, man-

teniendo solo las muestras más representativas de cada clase. Con esta eliminación de muestras de las nuevas clases, el modelo puede olvidar el conocimiento adquirido durante la etapa de entrenamiento anterior. Este posible problema se elimina añadiendo una función de pérdida de destilación, definida en la Sección 4.2.4.2, a la capa de clasificación correspondiente a las nuevas clases.

- **Actualización de la memoria representativa:** Una vez finalizada la etapa de *fine-tunning*, la memoria representativa debe actualizarse para incluir muestras de las nuevas clases. Este proceso se lleva a cabo utilizando las operaciones de selección explicadas en las sección 5.1.7.1. En primer lugar, la memoria elimina muestras de las clases almacenadas para obtener espacio para las muestras de las nuevas clases. Posteriormente, se seleccionan las muestras más representativas de las nuevas clases y se almacenan en la memoria según el método de selección explicado en la Sección 5.1.7.1.

5.2. Detalles de implementación

Para cada paso incremental, se realizan 300 iteraciones de entrenamiento seguidas de 200 iteraciones adicionales para el *fine-tunning*. La tasa de aprendizaje comienza en 0.05 y se reduce a la mitad cada 50 iteraciones. La misma reducción se utiliza en el caso del *fine-tunning*, en este caso se establece una tasa de aprendizaje de 0.005. Se entrenan las redes mediante el uso del descenso del gradiente estocástico, explicado en la Sección 4.2.5.1, con un decaimiento de peso de 0.0001 y un momento de 0.9. Se aplica regularización L2 en los gradientes para minimizar el sobre-ajuste.

Se utiliza una red convolucional (CNN) con arquitectura ResNet de 32 capas. Se almacenan $K = 4000$ muestras de destilación en la memoria representativa. Por otra parte, los datos de entrada se normalizan dividiendo los valores de los píxeles por 255 y restando la imagen media del conjunto de entrenamiento.

En cuanto a los parámetros mencionados en la Sección 5.1.2, se establecen los siguientes valores: $T_f = 1$ y $T_A = 600$. Y para los parámetros de la Sección 5.1.3 se establecen los siguientes valores: $T_I = 0.4$ y $T_{AR} = 0.5$.

5.2.1. Data Augmentation

En las etapas de entrenamiento y *fine-tunning* del modelo incremental, explicadas en la Sección 5.1.7.3, se aplican técnicas de *data augmentation* con el objetivo de aumentar la variabilidad del conjunto de entrenamiento y mejorar la generalización del modelo. Las transformaciones utilizadas son:

- **Efecto espejo:** se genera la imagen reflejada horizontalmente a partir de todas las variantes previas.
- **Ajuste de brillo:** se modifica la intensidad de la imagen original sumando un valor aleatorio en el rango $[-63, 63]$.
- **Recorte aleatorio:** se realiza un recorte aleatorio sobre cada una de las imágenes generadas.
- **Normalización del contraste:** el contraste de la imagen original se altera mediante un valor aleatorio en el rango $[0.2, 1.8]$.

$$im_{alterada} = (im - media) \times contraste + media \tag{5.2}$$

Donde im es la imagen original, $media$ es la media de los píxeles por canal, y $contraste$ es el valor aleatorio de contraste.

5.3. Entrenamiento de la red

El entrenamiento se ha llevado a cabo siguiendo el esquema que se muestra en la Figura 5.4. El *dataset* de entrada que consta de 15,000 imágenes se ha dividido en tres subconjuntos de 5,000 imágenes; por lo tanto, se han realizado tres entrenamientos progresivos de la red. En primer lugar, se ha procedido a la obtención de *clusters* utilizando las técnicas explicadas en la Sección 5.1. Una vez se consigue el conjunto de clusters, se procede a su etiquetado y clasificación de forma manual. Posteriormente, se realiza el entrenamiento de la red incremental, obteniendo su correspondiente modelo.

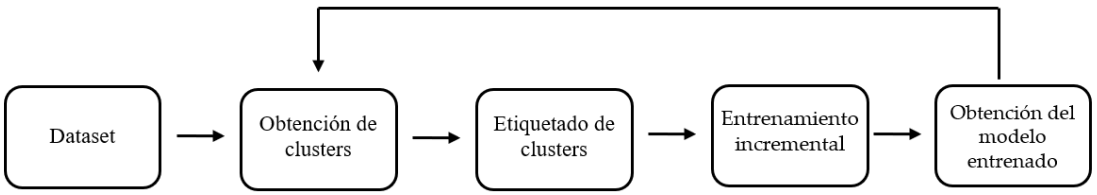

Figura 5.4: Obtención de clusters, etiquetado y entrenamiento de la red incremental

Una vez obtenido el modelo, este se utiliza para la obtención de clusters del segundo conjunto de entrenamiento y se sigue el mismo flujo de trabajo que con el primer conjunto de imágenes. De esta manera, se mejora la detección de forma progresiva y se obtiene un modelo final optimizado.

5.4. Testeo de la red

El testeo se ha realizado utilizando cada uno de los tres modelos obtenidos en el proceso con el fin de comprobar la mejora incremental en el funcionamiento de la red. El *dataset* de entrada para el test consta de 15.000 imágenes diferentes a las utilizadas durante el entrenamiento; de esta forma se observa el desempeño de cada modelo ante escenarios no conocidos.

5.4.1. Métricas

Con el objetivo de evaluar el rendimiento de los tres modelos obtenidos, se utilizan tres métricas diferentes.

En primer lugar, se utiliza la métrica de Localización correcta (CorLoc) que mide la precisión de la localización de regiones utilizando la Ecuación 5.3. En dicha ecuación N representa el número total de objetos, P_i se corresponde con la región predicha, G_i es la región real (ground-truth), y por último, $\mathbb{1}$ es una función que equivale a 1 en el caso en el que la condición en su interior sea verdadera, y 0 en caso contrario.

$$\text{CorLoc} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1} \left(\text{IoU}(P_i, G_i) > \tau \right) \tag{5.3}$$

Según este criterio, la predicción se considera correcta si la Intersección sobre la Unión (IoU) entre la región predicha y la región real (ground-truth) supera un valor τ de 0.5 en el caso del *RPN* y un valor de 0.1 para el *OFRP*.

Por otra parte, se utiliza la métrica de la precisión media promedio (mAP), que representa el promedio de la precisión media (AP) calculada para cada clase. La Ecuación 5.4 representa dicho método, donde C es el número total de clases de objetos y AP_c es la precisión media para una clase determinada c que se calcula como el área bajo la curva *Precision-recall* para dicha clase.

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^{C} AP_c \tag{5.4}$$

Por último, se utiliza la matriz de confusión que permite analizar la capacidad del modelo para diferenciar distintas clases de objetos. Cada fila de la matriz representa los valores reales de una clase determinada (ground-truth), y cada columna

representa los valores predichos de una clase determinada. Cada celda (i, j) define la cantidad de veces que un elemento de una clase real j se clasifica como clase i. De esta forma, en la diagonal principal de la matriz aparecen los aciertos, es decir, cuando coincide la predicción con la clase real, y las celdas restantes quedan ocupadas con los errores en la predicción.

Al igual que en la métrica de precisión media (mAP) la predicción se considera correcta si la Intersección sobre la Unión (IoU) entre la región predicha y la región real (ground-truth) supera un valor τ de 0.5 en el caso del *RPN* y un valor de 0.1 para el *OFRP*.

En concreto, se utiliza la matriz de confusión normalizada en porcentaje, en la que cada fila se divide por el total de muestras reales de esa clase y se multiplica por 100. Así, cada celda refleja el porcentaje de veces que una clase real fue clasificada como una clase determinada por el modelo. Esta normalización permite interpretar más fácilmente el rendimiento del modelo en cada clase, incluso en presencia de clases desbalanceadas.

Capítulo 6

Resultados experimentales

En este capítulo se expone la base de datos utilizada para la realización de este TFG. Posteriormente, se presentan los resultados alcanzados al realizar el test de los modelos obtenidos mediante entrenamiento incremental.

6.1. Base de Datos

Para la realización de los experimentos se ha utilizado un *dataset* formado por vídeos obtenidos de una cámara RGB que graba continuamente la rampa aeroportuaria o plataforma (zona donde aparcan los aviones para cargar equipaje y pasajeros) del Aeropuerto de Gdansk, ubicado en Polonia. Estas grabaciones en tiempo real están disponibles de forma pública [47].

El *dataset* consta de 96 clips de vídeo de un minuto de duración, grabados con una cámara FullHD que proporciona una transmisión de vídeo con una resolución de 1920×1080 y 15 FPS. El 60 % de los vídeos se grabaron durante la mañana y el otro 40 % se grabaron durante la tarde/noche; de esta forma se trabaja con diferentes condiciones de iluminación.

En la Figura 6.1 se muestra un conjunto de imágenes procedentes de este *dataset*, en el que aparecen diferentes elementos, como pueden ser un avión, trabajadores del aeropuerto o un camión de transporte de combustible.

En los experimentos realizados, se consideran las siguientes categorías de objetos: coche ('coche'), camión cisterna ('cist'), persona ('pe'), avión ('av'), remolcador de avión ('rem'), furgoneta ('furg'), camión de bomberos ('bomb') y escaleras ('esc'). El nombre de cada clase de objeto aparece de forma abreviada entre paréntesis.

Figura 6.1: Imágenes procedentes del *dataset* utilizado, obtenidas mediante una cámara RGB que graba la rampa aeroportuaria del Aeropuerto de Gdansk.

6.2. Resultados Experimentales

En este apartado se exponen los resultados experimentales obtenidos tras la evaluación del sistema de detección entrenado de forma incremental. El análisis se lleva a cabo con el fin de estudiar la capacidad del sistema para generar propuestas de regiones correctas y clasificarlas. Se evalúan tres modelos correspondientes a distintas etapas del proceso de entrenamiento, para así observar la evolución e impacto del aprendizaje incremental. El análisis de resultados se ha dividido en dos apartados principales que son la evaluación de los métodos de propuesta de regiones y la evaluación de la clasificación.

6.2.1. Comparativa de propuesta de regiones

En esta sección se exponen los resultados obtenidos mediante el uso de la métrica CorLoc definida en la Sección 5.4.1, desglosada por clase, para cada uno de los dos algoritmos de propuesta de regiones (RPN y OFRP). Se analiza en cada caso la capacidad de localización del último modelo obtenido durante el entrenamiento incremental, permitiendo realizar una comparación del comportamiento del sistema bajo distintos algoritmos de propuesta de regiones.

En la Tabla 6.1 aparecen los resultados obtenidos, donde un número mayor

significa un mejor desempeño, utilizando la métrica CorLoc para el algoritmo de propuesta de regiones RPN, desglosado por clase para el modelo del último paso incremental.

En este caso, el modelo presenta un rendimiento elevado en determinadas clases como 'camión cisterna', 'avión' o 'camión de bomberos', obteniendo valores superiores a 0.75. Esto indica una alta capacidad de localización de objetos grandes o de alta visibilidad. Sin embargo, aparecen puntuaciones más bajas en otras clases de menor tamaño, como pueden ser 'coche', 'furgoneta' o 'escaleras'.

Modelo	coche	cist	pe	av	rem	furg	bomb	esc
Modelo 3	0.09	0.84	0.17	0.77	0.35	0.11	0.85	0.1

Tabla 6.1: Se muestran los resultados obtenidos según la métrica CorLoc por clase para el algoritmo de propuesta de regiones RPN para el último paso incremental. Las columnas representan las clases evaluadas en el conjunto de prueba.

En la Tabla 6.2 aparecen los resultados obtenidos, donde un número mayor significa un mejor desempeño, utilizando la métrica CorLoc para el algoritmo de propuesta de regiones OFRP, desglosado por clase.

En este caso, el algoritmo presenta unos resultados bajos en todas las clases, en general inferiores a los obtenidos con el algoritmo RPN, algo esperable teniendo en cuenta que el método basado en flujo óptico solo es capaz de identificar regiones que presenten movimiento entre fotogramas.

Modelo	coche	cist	pe	av	rem	furg	bomb	esc
Modelo 3	0.02	0.15	0.04	0.15	0.11	0.02	0.15	0.07

Tabla 6.2: Se muestran los resultados obtenidos según la métrica CorLoc por clase para el algoritmo de propuesta de regiones OFRP para el último paso incremental. Las columnas representan las clases evaluadas en el conjunto de prueba.

6.2.2. Comparativa de predicciones de clases

En esta sección se exponen los resultados en la evaluación de la capacidad de los modelos para predecir correctamente las clases de objetos que se detectan. Para ello se han utilizado la métrica mAP, y la métrica AP, que permite un análisis detallado del rendimiento para cada clase, y la Matriz de Confusión definidas en la Sección 5.4.1.

En la Tabla 6.3 aparecen los resultados de la métrica mAP para cada uno de los modelos obtenidos. Los resultados muestran una mejora moderada pero progresiva en el rendimiento, alcanzando el valor más alto para el Modelo 3. Esto refleja evolución en el sistema en cuanto a su capacidad para discriminar entre clases y en la asignación de etiquetas a las regiones correctamente localizadas.

Modelo	mAP
Modelo 1	0.41
Modelo 2	0.42
Modelo 3	0.43

Tabla 6.3: La métrica mAP refleja la precisión global del modelo en detección. Cada fila muestra el valor correspondiente a cada uno de los modelos evaluados.

En la Tabla 6.4 aparecen los resultados de la métrica AP desglosada para cada clase, lo que permite identificar patrones específicos de comportamiento en cada uno de los modelos.

En este caso, los tres modelos obtenidos de forma incremental presentan puntuaciones altas en las clases 'camión cisterna', 'avión' y 'camión de bomberos', lo que indica que se trata de categorías fácilmente reconocibles debido a su forma, tamaño o frecuencia de aparición en el *dataset*. En concreto, la clase 'avión' presenta una mejora progresiva, alcanzando su valor máximo para el Modelo 3. Sin embargo, otras clases como 'remolcador de avión', 'furgoneta' o 'escaleras' obtienen unos resultados bajos, poniendo de manifiesto la dificultad del sistema para identificar clases con menor representación en el *dataset* o mayor dificultad de visualización de dichos objetos en las imágenes. Este comportamiento puede explicarse por la naturaleza del aprendizaje incremental, ya que, aunque el sistema utiliza una memoria representativa para evitar la pérdida de conocimiento adquirido en fases anteriores, si una clase no vuelve a aparecer en los pasos incrementales posteriores, su representación puede debilitarse, produciendo esa ligera degradación en su rendimiento.

A pesar de que las diferencias entre modelos no son uniformes en todas las clases presentes, el Modelo 3 tiene una mejora en varias categorías. Estos resultados evidencian el impacto del entrenamiento incremental realizado no solo en su capacidad de detección de regiones relevantes, como se observó en la Sección 6.2.1, sino también en la capacidad para afinar sus predicciones de clase.

Modelo	coche	cist	pe	av	rem	furg	bomb	esc
Modelo 1	0.16	0.62	0.30	0.76	0.08	0.39	0.99	0.00
Modelo 2	0.15	0.57	0.30	0.87	0.08	0.39	0.99	0.00
Modelo 3	0.16	0.56	0.29	0.89	0.10	0.39	0.99	0.01

Tabla 6.4: La métrica de precisión media AP calculada por clase representa el rendimiento en la detección de objetos para cada categoría. Cada fila contiene los resultados para cada modelo. Las columnas corresponden a las diferentes clases del conjunto.

Por último, se analiza el rendimiento de cada uno de los modelos obtenidos utilizando las matrices de confusión correspondientes.

En la Tabla 6.5, correspondiente al Modelo 1, se observa un rendimiento muy

alto con valores cercanos o iguales al 100 % en las clases 'coche', 'camión cisterna', 'persona', 'remolcador de avión' y 'camión de bomberos'. Por otra parte, la clase 'avión' alcanza también un valor alto del 84.2 %; sin embargo, presenta confusión principalmente con las clases 'camión cisterna' y 'remolcador de avión'. La clase 'furgoneta' se clasifica correctamente en el 42 % de los casos, pero muestra una alta confusión con la clase 'coche', debido al alto parecido entre ambos vehículos. En rasgos generales, el primer modelo se comporta de forma bastante precisa en clases con mayor representación, pero tiene algunos problemas para clasificar clases que pueden aparecer con menor frecuencia o son parecidas entre ellas.

	coche	**cist**	**pe**	**av**	**rem**	**furg**	**bomb**	**esc**
coche	**99.8**	0.0	0.0	0.0	0.1	0.1	0.0	0.0
cist	0.0	**100.0**	0.0	0.0	0.0	0.0	0.0	0.0
pe	0.0	0.0	**99.7**	0.0	0.0	0.0	0.2	0.1
av	0.9	7.6	1.6	**84.2**	3.4	1.0	1.1	0.2
rem	0.2	0.2	0.0	0.6	**98.9**	0.0	0.0	0.0
furg	58.0	0.0	0.0	0.0	0.0	**42.0**	0.0	0.0
bomb	0.0	0.0	0.0	0.0	0.0	0.0	**100.0**	0.0
esc	0.0	0.0	100.0	0.0	0.0	0.0	0.0	**0.0**

Tabla 6.5: Matriz de confusión normalizada en porcentaje para el Modelo 1

En la Tabla 6.6, correspondiente al Modelo 2, se mantiene un rendimiento muy alto con valores cercanos o iguales al 100 % en las clases 'coche', 'camión cisterna', 'persona', 'avión' y 'camión de bomberos'. Sin embargo, la clase 'remolcador de avión' sufre una bajada, mostrando confusión principalmente con la clase 'avión'. Por otra parte, la clase 'furgoneta' mantiene la confusión presentada en el Modelo 1. El segundo modelo presenta una mejora global en la clasificación, manteniendo ciertos errores en clases con mayor parecido a otras.

	coche	**cist**	**pe**	**av**	**rem**	**furg**	**bomb**	**esc**
coche	**99.9**	0.0	0.0	0.0	0.0	0.1	0.0	0.0
cist	0.0	**100.0**	0.0	0.0	0.0	0.0	0.0	0.0
pe	0.0	0.0	**99.9**	0.0	0.0	0.0	0.0	0.1
av	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	0.0
rem	2.2	4.1	9.2	28.0	**47.0**	4.1	5.2	0.5
furg	57.6	0.0	0.0	0.0	0.0	**42.4**	0.0	0.0
bomb	0.0	0.0	0.0	0.0	0.0	0.0	**100.0**	0.0
esc	0.0	0.0	100.0	0.0	0.0	0.0	0.0	**0.0**

Tabla 6.6: Matriz de confusión normalizada en porcentaje para el Modelo 2

En la Tabla 6.7, correspondiente al Modelo 3 y final, se consolida la precisión, obteniendo valores iguales o cercanos al 100 % en las clases 'coche', 'camión cisterna', 'persona', 'avión', 'remolcador de avión' y 'camión de bomberos'. La clase 'escaleras' en los modelos anteriores se clasificaba de forma incorrecta; sin embargo, en el tercer

modelo, alcanza un valor de 0.8 % de aciertos; aunque se trata de un valor bajo, el modelo comienza a aproximarse a una representación útil de esta clase.

	coche	**cist**	**pe**	**av**	**rem**	**furg**	**bomb**	**esc**
coche	**99.8**	0.0	0.0	0.0	0.0	0.1	0.0	0.0
cist	0.0	**100.0**	0.0	0.0	0.0	0.0	0.0	0.0
pe	0.0	0.1	**99.5**	0.2	0.0	0.0	0.2	0.0
av	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	0.0
rem	0.2	0.0	0.0	0.5	**96.8**	1.1	0.0	1.4
furg	58.0	0.0	0.0	0.0	0.0	**42.0**	0.0	0.0
bomb	0.0	0.0	0.0	0.0	0.0	0.0	**100.0**	0.0
esc	5.0	5.1	17.9	46.2	12.3	5.0	7.8	**0.8**

Tabla 6.7: Matriz de confusión normalizada en porcentaje para el Modelo 3

En conjunto, los resultados experimentales reflejan que el proceso de aprendizaje incremental ha sido efectivo para mejorar el rendimiento, reteniendo el conocimiento aprendido en fases anteriores y adaptándose a nuevas clases.

Capítulo 7

Conclusiones y Líneas Futuras

En este capítulo se exponen las conclusiones obtenidas tras realizar este TFG, así como un conjunto de líneas futuras que se podrían llevar a cabo tras la finalización del mismo.

7.1. Conclusiones

En este TFG se ha abordado el desarrollo, entrenamiento y evaluación de un sistema de detección de objetos en vídeo basado en aprendizaje incremental, con el objetivo de construir un modelo capaz de adaptarse de forma progresiva a nuevas clases sin sufrir el olvido de conocimiento catastrófico. A partir de un modelo base de detección, se ha generado un proceso de entrenamiento incremental que permite al modelo mejorar tanto la detección como la clasificación de objetos, evaluando el impacto de cada fase del entrenamiento en el rendimiento global del sistema. Tras la evaluación del trabajo, se han extraído una serie de conclusiones:

- Se ha demostrado que el uso de un entrenamiento estructurado en fases incrementales contribuye a una mejora progresiva y equilibrada del sistema, sin necesidad de volver a entrenar el modelo completo desde cero.
- Se ha observado que la clase 'escaleras', inicialmente ignorada por el modelo, comienza a ser reconocida de forma parcial en la última etapa, lo que refleja una mejora progresiva en su representación interna.
- A pesar de las mejoras, persiste una confusión notable entre las clases 'furgoneta' y 'coche' en todos los modelos, lo que sugiere la necesidad de estrategias adicionales de discriminación para clases visualmente similares.

- La combinación de detección mediante redes convolucionales, flujo óptico, descriptores visuales y técnicas de agrupamiento (HDBSCAN + UMAP) ha resultado eficaz para representar de forma compacta y robusta los objetos detectados, beneficiando así el posterior entrenamiento incremental sin necesidad de reentrenamiento completo desde cero.
- El uso de memoria representativa, y reorganización de clases en el entrenamiento incremental ha sido clave para preservar el conocimiento anterior y al mismo tiempo aprender nuevas clases de forma eficiente.
- El modelo final presenta mejoras generalizadas en su rendimiento, alcanzando una mayor precisión media que los modelos anteriores. No solo mejora su capacidad de localización, sino también su capacidad para clasificar correctamente objetos, incluso aquellos poco representados.

7.2. Líneas Futuras

A continuación, se muestran una serie de líneas futuras como posibles continuaciones al proyecto presentado en este TFG.

- Búsqueda y empleo de nuevos *datasets* más variados para incrementar la robustez y capacidad de generalización del modelo.
- Diseño y evaluación de nuevas arquitecturas de redes neuronales más especializadas que permitan mejorar la eficiencia sin sacrificar precisión.
- Evaluación de los modelos haciendo uso de distintos datos de entrada, con el fin de validar su comportamiento en entornos variados.
- Extender y profundizar en los resultados obtenidos en este TFG con el objetivo de publicar un artículo de investigación.

Bibliografía

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] L. Gervasoni, J. D'Amato, R. Barbuzza, M. Vénere, G. Bertolino, M. Cantero, M. Storti, and F. Teruel, "Un método eficiente para la sustracción de fondo en vídeos usando gpu," in *Mecánica Computacional*, vol. 33, no. 1, Asociación Argentina de Mecánica Computacional. AMCA, 2014, pp. 1721–1731.

[3] E. Osorio Arroyave, "Detección automatizada de objetos en secuencias de video utilizando histogramas de gradientes orientados," Trabajo de Fin de Grado, Universidad Tecnológica de Pereira, 2015.

[4] P. G. Vázquez, F. T. Medina, and F. G. O. Zamora, "Detección de objetos por segmentación multinivel combinada de espacios de color," in *Proceedings of the 25th Conference on Automática (XXV Jornadas de Automática)*, 2004, p. 19.

[5] F. Pérez Gutiérrez, "Detección de vehículos en circulación mediante visión artificial y redes neuronales convolucionales," Trabajo de Fin de Máster, Universidad Complutense de Madrid, 2020.

[6] M. Massiris, C. Delrieux, and J. Á. Fernández Muñoz, "Detección de equipos de protección personal mediante red neuronal convolucional yolo," in *XXXIX Jornadas de Automática*, 2018, pp. 1022–1029.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, 2023.

[8] H. E. Espinola Rivera, "Sistema inteligente basado en arquitectura visual transformer (vit) para clasificación de imágenes de cáncer de piel," Trabajo de Fin de Grado, Universidad Nacional de Trujillo, 2024.

[9] F. Soler Guiral, "Optimización de los procesos de triaje en servicios de urgencias mediante transformadores de visión sobre radiografías de tórax," Trabajo de Fin de Grado, Universitat Politècnica de València, 2023.

[10] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[12] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *Advances in neural information processing systems*, vol. 27, 2014.

[13] B. Ans, S. Rousset, R. M. French, and S. Musca, "Self-refreshing memory in artificial neural networks: Learning temporal sequences without catastrophic forgetting," *Connection Science*, vol. 16, no. 2, pp. 71–99, 2004.

[14] R. French, "Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference," *Proceedings of the 16th Annual Cognitive Science Society Conference*, 08 1994.

[15] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[16] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[18] H. Jung, J. Ju, M. Jung, and J. Kim, "Less-forgetting learning in deep neural networks," *arXiv preprint arXiv:1607.00122*, 2016.

[19] A. Rannen Triki, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," *arXiv e-prints*, pp. arXiv–1704, 2017.

[20] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 3400–3409.

[21] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[22] A. V. Terekhov, G. Montone, and J. K. O'Regan, "Knowledge transfer in deep block-modular neural networks," in *Proceedings of the 4th International Conference on Biomimetic and Biohybrid Systems (Living Machines 2015)*. Springer, 2015, pp. 268–279.

[23] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[24] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, "Error-driven incremental learning in deep convolutional neural network for large-scale image classification," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 177–186.

[25] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[26] Z. Wu, Y. Liu, K. Zhao, J. Fu, and Z.-J. Zhang, "Label-efficient online continual object detection in streaming video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2023, pp. 11 765–11 775.

[27] S. Liu, C. Lu, M. Xu, and Y. Xia, "Continual detection transformer for incremental object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023, pp. 3404–3414.

[28] X. Villa and J. Pajarinen, "Pivot: Prompting for video continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023, pp. 6408–6418.

[29] J. Zhang, W. Li, S. Cheng, Y.-L. Li, and S. Wang, "Dynamic object queries for transformer-based incremental object detection," *arXiv preprint arXiv:2407.21687*, 2024.

[30] *Welcome to Pyhton.org*, Python.org, Último acceso: 14 Mayo, 2024 [Online]. Disponible en: https://www.python.org/.

[31] *Welcome to pytorch.org*, pytorch.org, Último acceso: 14 Mayo, 2024 [Online]. Disponible en: https://www.python.org/.

[32] Intel Corporation, "Intel Xeon Silver 4314 Processor (24M Cache, 2.40 GHz)," 2024, Último acceso: 21 de junio de 2024 [Online]. Disponble en: https://ark.intel.com/content/www/xl/es/ark/products/215269/intel-xeon-silver-4314-processor-24m-cache-2-40-ghz.html.

[33] NVIDIA Corporation, "GeForce RTX 3090 y 3090 Ti Graphics Cards," 2024, Último acceso: 21 de junio de 2024 [Online]. Disponible en: https://www.nvidia.com/es-es/geforce/graphics-cards/30-series/rtx-3090-3090ti/.

[34] R. Ade and P. Deshmukh, "Methods for incremental learning: a survey," *International Journal of Data Mining & Knowledge Management Process*, vol. 3, no. 4, p. 119, 2013.

[35] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[36] C. Hu, Y. Chen, X. Peng, H. Yu, C. Gao, and L. Hu, "A novel feature incremental learning method for sensor-based activity recognition," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 6, pp. 1038–1050, 2019.

[37] MathWorks, "Support vector machine (svm)," 2024, Último acceso: 20 de mayo de 2024 [Online]. Disponible en: https://es.mathworks.com/discovery/support-vector-machine.html.

[38] J.-L. An, Z.-O. Wang, and Z.-P. Ma, "An incremental learning algorithm for support vector machine," in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, vol. 2, 2003, pp. 1153–1156 Vol.2.

[39] P. Ruiz-Barroso, F. M. Castro, and N. Guil, "Real-time unsupervised video object detection on the edge," *Future Generation Computer Systems*, vol. 167, p. 107737, 2025.

[40] ——, "Real-time unsupervised object localization on the edge for airport video surveillance," in *Proceedings of the 11th Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA 2023)*. Springer-Verlag, 2023, p. 466–478.

[41] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*. Springer, 2003, pp. 363–370.

[42] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[44] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[45] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–51, 2015.

[46] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," *arXiv preprint arXiv:1807.09536*, 2018.

[47] Gdańsk Lech Wałęsa Airport, "Kamery internetowe," 2024, Último acceso: 18 de mayo de 2024 [Online]. Disponible en: https://www.airport.gdansk.pl/lotnisko/kamery-internetowe-p30.html.