Evaluación de modelos de Deep Learning para el control de aforos en dispositivos de bajo consumo
Evaluation of Deep Learning models for capacity control in low consumption devices

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Pablo Sánchez-Rodilla Serrano
Javier Gómez Arribas

Director
Sandra Catalán Pallarés
Sergio Bernabé García

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Evaluación de modelos de Deep Learning para el control de aforos en dispositivos de bajo consumo
Evaluation of Deep Learning models for capacity control in low consumption devices

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Pablo Sánchez-Rodilla Serrano
Javier Gómez Arribas

Director
Sandra Catalán Pallarés
Sergio Bernabé García

Convocatoria: *Mayo 2024*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

27 de Mayo de 2024

Dedicatoria

A nosotros, por siempre haber confiado el uno en el otro y nunca rendirse.

Agradecimientos

Queremos expresar nuestro más profundo agradecimiento a Sandra y Sergio, nuestros tutores, por su infinita paciencia a lo largo de todo el curso. Nos han acompañado en este largo camino brindándonos la ayuda necesaria en cada momento, siendo constantes en su apoyo y ofreciendo repetidamente valiosos consejos y directrices que han sostenido este trabajo en los momentos más difíciles.

También extendemos nuestro agradecimiento por el apoyo incondicional de nuestras familias y amigos a lo largo de todo este proceso.

Finalmente, queremos agradecer a nuestros compañeros y excompañeros de carrera, en especial a Alejandro-Daniel Díaz Román, quien nos inspiró al inicio del TFG y nos ayudó a enfocarnos en nuestros objetivos.

Resumen

Evaluación de modelos de Deep Learning para el control de aforos en dispositivos de bajo consumo

Este Trabajo de Fin de Grado (TFG) se enfoca en el desarrollo y evaluación de un sistema de gestión y control de aforos en eventos, utilizando modelos de detección de objetos y dispositivos de bajo consumo. La motivación principal del proyecto radica en la necesidad de mejorar la seguridad y eficiencia en la gestión de multitudes, especialmente en contextos donde el control del número de asistentes es crucial.

Para una mejor comprensión del funcionamiento de estos modelos y su aplicación en la detección de objetos, se presentan conceptos esenciales sobre el tema.

El TFG busca llevar estos conceptos teóricos al ámbito práctico del control de aforos. Por ello, se realiza una investigación sobre los entornos de desarrollo y ejecución en los que se trabaja, además de analizar el dispositivo de bajo consumo, la Raspberry Pi 4, que se utiliza y la arquitectura del sistema que se pretende llevar a cabo.

Finalmente, se proporciona una explicación detallada sobre los conjuntos de datos utilizados, las configuraciones aplicadas durante los entrenamientos de los diferentes modelos de redes neuronales probados, y una evaluación exhaustiva de los resultados obtenidos. Pudiendo de esta manera realizar una conclusión de todo el trabajo.

Palabras clave

Inteligencia Artificial, modelo, dataset, entrenamiento, red neuronal, arquitectura.

Abstract

Evaluation of Deep Learning models for capacity control in low consumption devices

This Final Degree Project (TFG) focuses on the development and evaluation of a management and control system for capacity at events, using object detection models and low-power devices. The main motivation of the project lies in the need to improve safety and efficiency in crowd management, especially in contexts where controlling the number of attendees is crucial.

For a better understanding of the operation of these models and their application in object detection, essential concepts on the subject are presented.

The TFG seeks to bring these theoretical concepts to the practical field of capacity control. For this reason, an investigation is carried out on the development and execution environments in which we work, in addition to analyzing the low-power device, the Raspberry Pi 4, that is used and the architecture of the system that is intended to be carried out.

Finally, a detailed explanation is provided about the data sets used, the configurations applied during the training of the different neural network models tested, and an exhaustive evaluation of the results obtained. Being able to conclude all the work in this way.

Keywords

Artificial inteligence, model, dataset, training, neural net, arquitecture.

Índice

1. Introducción **1**
1.1. Motivación . . . 2
1.2. Objetivos . . . 3
1.3. Plan de trabajo . . . 3
1.4. Estructura del documento . . . 4

2. Estado del Arte **7**
2.1. Deep Learning . . . 8
2.2. Redes Neuronales . . . 9
2.2.1. Funcionamiento de una red neuronal . . . 10
2.2.2. Tipos de redes neuronales . . . 11
2.2.3. Redes Neuronales Convolucionales para la detección de objetos (CNN) . . . 12
2.3. Modelos de CNN para la detección de objetos . . . 14
2.3.1. YOLO . . . 15
2.3.2. EfficientDet . . . 16
2.3.3. Single Shot Multibox Detector (SSD) . . . 17
2.3.4. MobileNet . . . 18

3. Arquitectura del sistema **21**
3.1. Software del sistema . . . 22
3.1.1. Interfaz gráfica - GUI . . . 22
3.1.2. Entorno de desarrollo - Google Colab . . . 22
3.1.3. Ultralytics . . . 24
3.1.4. Tensorflow y tensorflow lite . . . 25
3.1.5. Procesamiento de los datos de entrada . . . 25
3.2. Hardware del sistema . . . 26
3.2.1. Entorno de ejecución - Google Colab Pro . . . 26
3.2.2. Raspberry . . . 27

4. Procesos de Entrenamientos **29**
4.1. Conjuntos de datos . . . 29

4.1.1. Estructura de los datos 31
4.1.2. Formato de los datos 31
4.1.3. Modificaciones de los datos 33
4.2. Preparación y configuración del entrenamiento 34
4.2.1. Preparación y configuración del entrenamiento de EfficientDet y SSD-MobileNet 34
4.2.2. Preparación y configuración del entrenamiento de YOLOv8 . 37
4.3. Entrenamiento 38
4.3.1. Entrenamiento de EfficientDet y SSD-MobileNet 38
4.3.2. Entrenamiento de YOLOv8 39

5. Evaluaciones y resultados **41**
5.1. Evaluación y resultados de EfficientDet 42
5.2. Evaluación y resultados de SSD-MobileNet 43
5.3. Evaluación y resultados de YOLOv8 44
5.4. Comparación de los modelos 45

6. Conclusiones y Trabajo Futuro **47**
6.1. Conclusiones 47
6.2. Trabajo a futuro 48

Introduction **49**

Conclusions and future Work **55**

Contribuciones Personales **57**

Bibliografía **59**

Índice de figuras

1.1. Aglomeración de personas en un espacio sin control. 2
1.2. Diagrama de Gantt con la planificación del TFG. 4

2.1. Jerarquía de IA, Aprendizaje Automático y Aprendizaje Profundo. . 9
2.2. Partes de una red neuronal. 9
2.3. Desenrollamiento de capa de una RNN. 12
2.4. Fases de una CNN. 13
2.5. Proceso de una ventana de filtro. 14
2.6. Tareas realizables por YOLOv8. 16
2.7. Arquitectura EfficientDet. 16
2.8. Funcionamiento SSD. 18

3.1. Arquitectura general del sistema. 21
3.2. Interfaz gráfica del sistema. 23
3.3. Interfaz de Google Colab. 24
3.4. Raspberry Pi 4-1. 27

4.1. Imágenes de mala calidad pertenecientes al conjunto de datos utilizado. 30
4.2. Límites en relación con la memoria. 37
4.3. Pérdida total de EfficientDet y SSD-MobileNet. 39
4.4. Box Loss, Class Loss y Object Loss. 39

5.1. Ejemplo de la evaluación de IoU. 42
5.2. Inferencia EfficientDet. 43
5.3. Inferencia SSD-MobileNet. 44
5.4. Inferencia de YOLOv8. 45

6.1. Crowd of people in an uncontrolled space. 50
6.2. Gantt diagram representing TFG planification. 52

Índice de tablas

3.1. Diferencias entre arquitecturas de YOLOv8. 25
3.2. GPU de Google Colab Pro. 27

4.1. Estructura del conjunto de datos para EfficientDet. 31
4.2. Estructura del conjunto de datos para SSD-MobileNet y YOLOv8. . . 31

5.1. Métricas de EfficientDet. 42
5.2. Métricas de SSD-MobileNet. 43
5.3. Métricas de YOLOv8. 44
5.4. Comparación de precisión y velocidad. 45

Capítulo 1

Introducción

En este primer capítulo se presenta una introducción a este TFG, el cual aborda el desarrollo de una solución para el control de aforos en eventos mediante el uso de inteligencia artificial (IA) y su ejecución en sistemas de bajo consumo.

En la actualidad, la gestión y control eficiente del aforo en eventos se ha convertido en una preocupación creciente, no solo para garantizar la seguridad y comodidad de los asistentes, sino también para cumplir con las normativas establecidas y optimizar el uso de los recursos disponibles.

Cuando se trata de contar las personas que hay presentes en un lugar concreto, se puede recurrir a múltiples métodos tan rudimentarios como el hecho de realizar un conteo con los dedos de las manos, o mediante el uso de contadores analógicos muy utilizados por el personal de seguridad que forma parte de todo tipo de eventos.

En las últimas décadas se han desarrollado modelos automatizados que son capaces de controlar el aforo de personas y otros individuos en espacios con un perímetro cerrado en donde se establecen uno o varios puntos de entrada y salida. Por ejemplo, el Metro de Madrid limita las entradas y salidas colocando tornos capaces de sumar o restar una unidad al aforo total cada vez que se utilizan, también, en multitud de establecimientos se implementan sensores infrarrojos que a través del movimiento detectan la entrada o salida de un individuo del lugar, etc.

El problema surge cuando se quiere realizar el control del aforo de un evento llevado a cabo en un espacio de grandes dimensiones, abierto al público y en donde no hay un control del número de entradas y salidas (veáse Figura 1.1 (TheObjective, 2018)). Para llevar un control del aforo en este tipo de eventos se han desarrollado métodos más avanzados como sistemas de seguimiento por radiofrecuencia, aplicaciones de teléfonos inteligentes con geolocalización o modelos de inteligencia artificial capaces de detectar y contar todas las personas situadas en el área de celebración del evento. Es la implementación de este último recurso el objetivo que se ha propuesto conseguir durante el desarrollo de este TFG.

Figura 1.1: Aglomeración de personas en un espacio sin control.

1.1. Motivación

El TFG en el que nos hemos visto envueltos durante este curso ha sido una experiencia verdaderamente enriquecedora que nos ha llevado a una expansión significativa de nuestros conocimientos tanto a nivel académico como profesional. Aunque partíamos con escasos conocimientos previos, el desafío de adentrarnos en el fascinante mundo de la inteligencia artificial y su aplicación en la gestión y control de aforos mediante su posible integración en dispositivos de bajo consumo nos ha cautivado por completo.

Durante este trabajo, hemos tenido la oportunidad de sumergirnos en un universo tecnológico fascinante y novedoso, explorando las complejidades y posibilidades de la utilización de la IA en el control y gestión de eventos. Esta inmersión no solo nos ha proporcionado un profundo entendimiento teórico, sino que también nos ha permitido desarrollar habilidades prácticas en la implementación y operación de esta tecnología.

Nuestro compromiso con este TFG va más allá de la mera investigación académica. Nos impulsa una firme convicción en el potencial transformador de la tecnología cuando se emplea con responsabilidad. Al centrar nuestros esfuerzos en desarrollar una herramienta eficaz para la gestión y control de aforos en eventos, aspiramos a contribuir fuertemente en la seguridad y el bienestar de la sociedad. Por ello, visualizamos este trabajo como un paso adelante en la creación de soluciones innovadoras que aborden desafíos contemporáneos, y estamos emocionados por el impacto positivo que nuestra labor puede tener en la comunidad y más allá.

1.2. Objetivos

El principal objetivo de este TFG es la evaluación de varios modelos de inteligencia artificial preexistentes, capaces de reconocer y contar las personas presentes en un archivo de entrada, que puede ser una imagen, grabación o transmisión en directo, procesado por un dispositivo de bajo consumo.

En primer lugar, para lograr el objetivo principal, es necesario la aplicación de técnicas de IA, específicamente el uso de redes neuronales convolucionales, ya que ofrecen una vía prometedora para abordar este desafío de manera efectiva. Modelos como SSD (University, 2020b) y YOLO (YOLO, 2023) han demostrado su capacidad para realizar tareas de detección y seguimiento de objetos en tiempo real, lo que los convierte en herramientas ideales para el monitoreo y control del flujo de personas en entornos de eventos.

Al integrar estas tecnologías en el desarrollo de nuestra solución, buscamos no solo automatizar el proceso de conteo de asistentes, sino también obtener información en tiempo real sobre la densidad de personas en diferentes áreas del evento.

Además, se ha orientado el diseño del sistema para que pueda ser implementado en una placa de bajo coste y consumo, lo que permite su integración en dispositivos de bajo consumo que operen en tiempo real. Dentro de este diseño se ha desarrollado una interfaz gráfica desde la que se puede acceder a este dispositivo y probar los modelos entrenados y observar los resultados de una forma más fácil e intuitiva.

Por otro lado, es fundamental la ejecución de un correcto entrenamiento con un conjunto de datos específico para que la capacidad del modelo para la detección y conteo de personas no esté limitada por la calidad de la entrada, su perspectiva, el tamaño del sujeto, etc.

1.3. Plan de trabajo

Durante la totalidad del desarrollo de este TFG, las tareas han sido meticulosamente bien establecidas, ayudando así a dividir la carga de trabajo y responsabilidades entre ambos miembros del equipo. En la Figura 1.2 presentamos un diagrama de Gantt orientativo de la planificación de este TFG.

Inicialmente, nos basamos en el TFG llevado a cabo por Alejandro-Daniel Díaz Guzmán durante el curso académico 2022-2023. Su trabajo, centrado en la *Simulación de Vuelo Autónomo de UAV para misiones de búsqueda y rescate*, sirvió como punto de partida para situarnos en el contexto adecuado, estructurar nuestro trabajo y comenzar nuestras investigaciones.

En los primeros meses posteriores, nos dedicamos a la búsqueda e investigación de simuladores de drones gratuitos que fueran compatibles con nuestros equipos,

así como capaces de cargar contenido multimedia para su posterior procesamiento. Aunque esta búsqueda resultó no tener éxito, esta experiencia marcó una evolución en el enfoque de este TFG, dejando en segundo plano la integración de un dron en el sistema.

Posteriormente, nos centramos rápidamente en el contenido más sustancial: la búsqueda, entrenamiento y evaluación de modelos de redes neuronales convolucionales especializados en la detección de objetos.

Finalmente, este TFG ha estado enfocado en la evaluación y comparación de los resultados obtenidos por los modelos seleccionados y la realización de la memoria del trabajo.

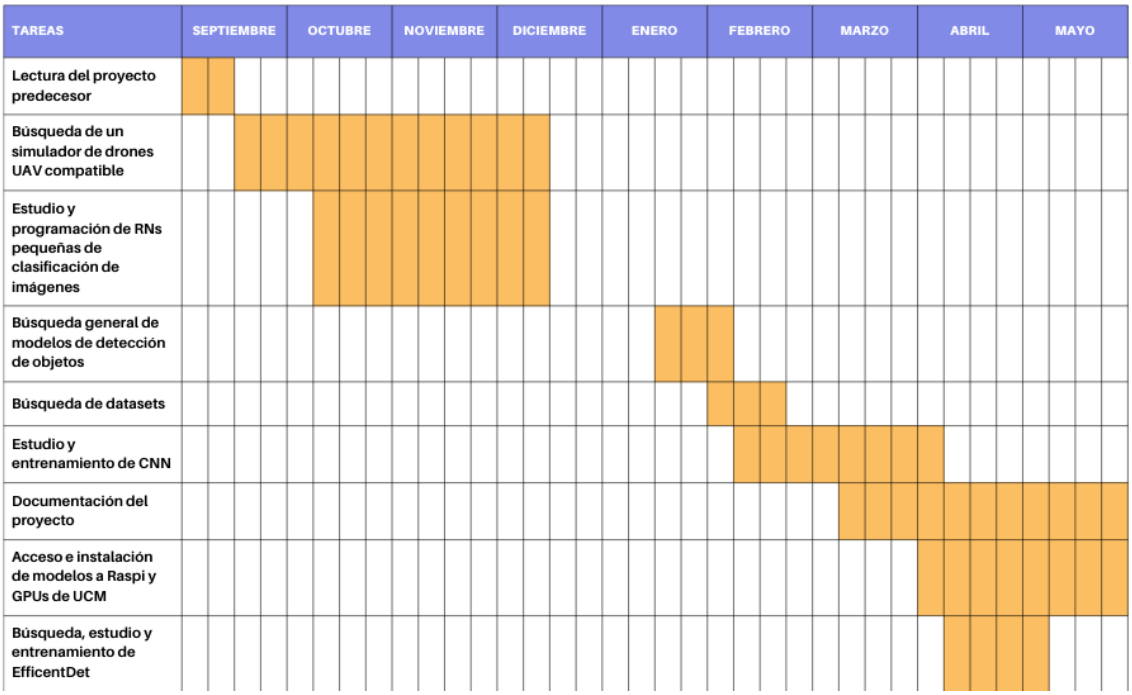

Figura 1.2: Diagrama de Gantt con la planificación del TFG.

1.4. Estructura del documento

Este documento está organizado en 6 capítulos. A continuación, se enumeran todos ellos acompañados de un breve resumen:

- **Capítulo 1: Introducción.** Este capítulo es en el que se encuentra el lector en este preciso instante. Este capítulo introduce el TFG y habla de las motivaciones, objetivos, plan de trabajo y estructura del documento.
- **Capítulo 2: Estado del Arte**. Este capítulo introduce los conceptos fundamentales para la comprensión de este TFG. Habla del Deep Learning y del

funcionamiento y tipos de las redes neuronales con las que se trabaja en la actualidad. Además, entra en detalle en el gran paradigma de la inteligencia artificial y las aplicaciones que puede llegar a tener.

- **Capítulo 3: Arquitectura del sistema** Este capítulo describe la arquitectura del sistema que se quiere implementar, trata tanto la parte de hardware como la parte de software.
- **Capítulo 4: Procesos de los entrenamientos** Este capítulo describe el proceso de entrenamiento de un modelo preexistente, desde la preparación del conjunto de datos hasta la preparación del entrenamiento y su configuración.
- **Capítulo 5: Evaluación y resultados**: Este capítulo describe los resultados obtenidos con cada uno de los modelos preparados. Además, presenta un breve análisis de los mismos haciendo distintas comparaciones técnicas.
- **Capítulo 6: Conclusiones y Trabajo a Futuro**: Este capítulo incluye las conclusiones obtenidas una vez finalizado el trabajo y propone una serie de contribuciones futuras.

Capítulo 2

Estado del Arte

La inteligencia artificial ha sido creada con el propósito de imitar el comportamiento del cerebro humano. Desde los años 50 con la *Prueba de Turing*, se lleva desarrollando conscientemente el concepto de inteligencia artificial. Se han construido sistemas que están a disposición de todos y que son muy accesibles en el día a día del ser humano. Algunos ejemplos son los sistemas de recomendaciones de películas de las plataformas de *TV on Demand*, los asistentes virtuales como *Siri* o *Alexa* o los algoritmos de reconocimiento facial. En este TFG se seleccionan una serie de modelos de inteligencia artificial que se utilizarán para la detección de objetos, más concretamente, en la detección de personas.

Han sido varios los estudios previos que han empleado modelos de detección de objetos en una amplia variedad de contextos. Tanto gobiernos como instituciones privadas han desarrollado sistemas de vigilancia urbana, control de aforo en eventos masivos, monitoreo de fronteras y puntos de control, así como aplicaciones en seguridad aeroportuaria. A continuación, se describen algunos sistemas e investigaciones actuales que ilustran el uso y la relevancia de estos modelos en diversas aplicaciones:

- Desde 2013 Elon Musk ha estado trabajando en *Tesla AutoPilot*(Tesla, 2024), un sistema de asistencia al conductor avanzado que mejora la seguridad y la comodidad al volante. Este sistema se encarga de detectar objetos y señales, además de tomar decisiones de forma automática.

- Comenzando en 2016, *Amazon* ha ido implementado en sus tiendas *Amazon Go* (BBC, 2018) un sistema avanzado de detección de objetos que permite a los clientes realizar la compra sin pasar por caja, sin tener contacto con el personal de la tienda, y sin tener que hacer cola. Amazon Go utiliza una combinación de cámaras, sensores y algoritmos de IA, que permiten identificar automáticamente los productos que el cliente selecciona para realizar un cobro directamente a su cuenta de Amazon.

- La Policía de Londres implementó en 2018 un sistema de vigilancia urbana basado en IA desarrollado en colaboración con *Facewatch* (Facewatch, 2024). Se trata de un sistema que detecta objetos para identificar actividades sospechosas o delictivas en zonas muy pobladas.

- A partir de 2021, mediante el empleo de cámaras instaladas en los barcos, y el uso de drones, The *Ocean Cleanup* (Vries, 2022) ha hecho uso de modelos de detección de objetos para identificar y rastrear plásticos dispersados a lo largo de los océanos facilitando así las labores de limpieza.

- En 2022 Vodafone lanzó la iniciativa *Inteligencia Artificial impacto en el modelo de negocio* que trata de profundizar en el estado del arte de la IA y propone una serie de proyectos. Uno de estos proyectos fue *Detección y optimización del tráfico mediante modelos de IA* de Carlos Ortiz (Vodafone, 2022), el cual está centrado en la detección de vehículos y en la optimización del tráfico en grandes aglomeraciones y lugares con un importante tránsito, como ciudades, centros logísticos o puertos.

Cada uno de los proyectos mencionados se enmarca en un ámbito específico de la inteligencia artificial. Tal como se detalló en la introducción de este TFG, este trabajo también emplea la inteligencia artificial aplicada a la detección de objetos, en particular, para realizar un control del aforo en eventos.

2.1. Deep Learning

Dentro de la inteligencia artificial, la rama del *deep learning* (Google, 2024a) permite a los sistemas aprender sin estar programados explícitamente. Mediante los procesos de pendiente de gradiente y propagación inversa, el algoritmo de deep learning se ajusta y se adapta a sí mismo para ganar precisión.

El deep learning se distingue del *machine learning* clásico por el tipo de datos con los que trabaja y los métodos de aprendizaje que emplea. Los algoritmos de machine learning utilizan datos estructurados y etiquetados para hacer predicciones, lo que requiere definir características específicas durante la entrada de datos al modelo y organizarlos en tablas. Aunque también pueden utilizar datos no estructurados, estos suelen ser procesados previamente para organizarlos en un formato estructurado.

En el caso de los algoritmos de deep learning, nos referimos a las redes neuronales, que procesan los datos de forma jerárquica. Estos algoritmos son capaces de trabajar con datos no estructurados, como texto e imágenes, y automatizan la extracción de características, reduciendo así la necesidad de intervención humana experta. El deep learning tiene numerosas aplicaciones, como la evaluación de riesgos de negocio para detectar fraudes financieros, los *chatbots* para un servicio al cliente, o el reconocimiento de patrones en imágenes para asistencia sanitaria y control de aforos por ejemplo.

En la Figura 2.1 (IAAR, 2020) se puede ver la jerarquía de todos estos conceptos, es decir, de la inteligencia artifical, el machine learning y el deep learning.

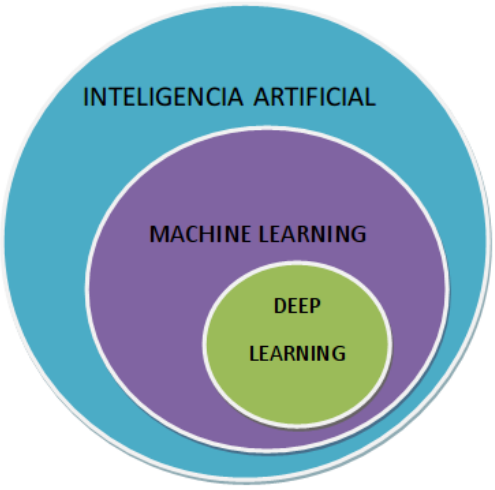

Figura 2.1: Jerarquía de IA, Aprendizaje Automático y Aprendizaje Profundo.

2.2. Redes Neuronales

Las redes neuronales (RN) son algoritmos fundamentales dentro de la inteligencia artificial y, como se menciona anteriormente, se inspiran en el cerebro humano. Estas redes consisten en un conjunto de nodos interconectados que trabajan juntos para resolver problemas mediante la extracción de patrones. Estos nodos funcionan de manera similar a las neuronas humanas, por lo que se les llama neuronas artificiales o nodos.

Estas neuronas se agrupan en capas: la capa de entrada, la capa de salida y una o más capas ocultas que realizan y mejoran el procesamiento. Véase la Figura 2.2 (Larranaga, 2019) para observar un esquema de estos conceptos.

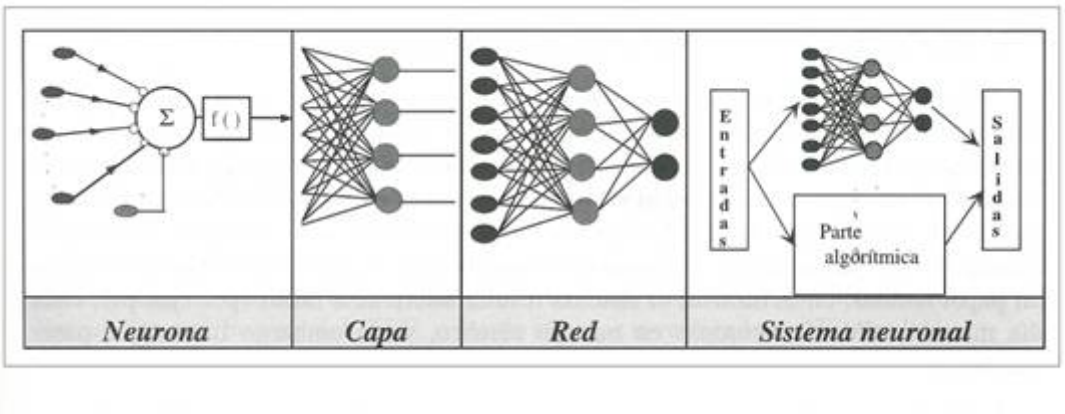

Figura 2.2: Partes de una red neuronal.

Además, las neuronas y las capas están conectadas por enlaces ponderados. Las conexiones entre las neuronas artificiales tienen asociados una serie de pesos que

determinan la importancia de las señales de entrada. Estos pesos se ajustan durante el proceso de entrenamiento de la red para optimizar su rendimiento.

2.2.1. Funcionamiento de una red neuronal

Para comprender cómo funcionan las redes neuronales, es fundamental entender el concepto de regresión lineal (IBM, 2024b). Esta técnica se emplea para predecir el valor de una variable dependiente en función de una o más variables independientes. La regresión lineal es ampliamente utilizada en diversos campos, como la biología, ciencias medioambientales y financieras. Según IBM, la fórmula de la regresión lineal para los nodos de una red neuronal se expresa en la fórmula 2.1.

$$salida = f(x) = \begin{cases} 0 \quad si \quad \sum w_i x_i + b >= 0 \\ \\ 1 \quad si \quad \sum w_i x_i + b < 0 \end{cases} \tag{2.1}$$

En esta ecuación debemos describir la salida de una neurona en función de tres valiables:

x_i : Variables binarias.

w_i : Variables no binarias.

b : Umbral

Las variables binarias determinan si se debe activar la siguiente neurona o no, es decir, la salida. Las variables no binarias son las que cuantifican la importancia de las no binarias. Es decir, se asignan ponderaciones o pesos a nuestras variables para designar un resultado. Por último se menciona el umbral, que representa el grado de inhibición de una neurona. Este umbral debe restar al sumatorio de la multiplicación de las variables binarias con su ponderación. De esta manera el nodo de se activará o no. A esta fórmula se le llama función de activación. A partir de la fórmula de regresión lineal nacen la funciones de activación, las cuales se irán modificando según cómo se desee construir la red neuronal. Existe otro parámetro llamado sesgo el cual es fácil de confundir con el umbral. Es un parámetro adicional cuya función es permitir que la red neuronal modele mejor los datos. Es un parámetro que se le suma a las entradas de las neuronas evitando que sean muchas entradas sean nulas y permitiendo un mejor resultado. Mientras el umbral se suma a la salida de la neurona, el sesgo se suma en la entrada.

Existen varios tipos de función de activación. Posteriormente, se explicará qué funciones de activación serán necesarias para un modelo de detección de personas.

Llevando este concepto a la detección de objetos, las variables van a ser cada uno de los píxeles, el tamaño y el color (dos o tres canales) de la imagen. Cuando se entrena un modelo, nos interesa evaluar su precisión usando una función de coste

(o pérdida). El propósito es minimizar la función de pérdida para garantizar un ajuste preciso en todas las observaciones. Conforme el modelo ajusta sus pesos, emplea la función de coste y el aprendizaje por refuerzo para alcanzar el punto de convergencia, conocido como mínimo local. El proceso mediante el cual el algoritmo ajusta sus pesos es a través de un descenso de gradiente. Para esto utilizaremos el descenso de gradiente *Batch* (IBM, 2024c), se calcula la suma de errores para cada punto en un conjunto de entrenamiento, y luego se actualiza el modelo hasta que todos los ejemplos de entrenamiento disponibles sean evaluados. Dividir el dataset en lotes o *batches*, lotes de imágenes en caso de referiste a la detección de objetos, proporciona una mayor eficiencia computacional en conjuntos de datos grandes como es el caso en este proyecto.

2.2.2. Tipos de redes neuronales

A día de hoy existen múltiples tipos de redes neuronales (IBM, 2024a), que se clasifican en función de la aplicación en la que se emplea la red. Teniendo en cuenta la topología de la red neuronal, se pueden clasificar en dos tipos:

- Las Redes Neuronales Convolucionales (CNN) son un tipo específico de red neuronal diseñada especialmente para el reconocimiento de patrones y la clasificación de imágenes. Estas redes constan de una capa de entrada, una capa de salida y múltiples capas ocultas intermedias.

  En primer lugar, las CNN utilizan un método conocido como propagación hacia adelante, lo que significa que la información fluye en una sola dirección: desde la capa de entrada, a través de las capas ocultas, hasta llegar a la capa de salida. Cada neurona en una capa transmite su salida a las neuronas de la siguiente capa.

  Además, el término *convolucional* proviene de la presencia de capas convolucionales dentro de la red. Estas capas aplican operaciones de convolución a la entrada, permitiendo a la red capturar y aprender características locales de la imagen, como bordes, texturas y patrones más complejos.

- Las Redes Neuronales Recurrentes (RNN) son un tipo concreto de red neuronal que se caracterizan por sus bucles de retroalimentación, que les permiten procesar secuencias de datos y mantener una memoria interna de entradas anteriores. Este tipo de arquitectura es especialmente útil para tareas que involucran datos temporales o secuenciales. Por ello, son muy utilizadas para la predicción del mercado de acciones o tareas de reconocimiento por voz y generación de texto.

  En la Figura 2.3 (Alexis, 2020) se ilustra el desenrollamiento de una capa a lo largo del tiempo. Este desenrollamiento muestra cómo una RNN procesa secuencias de datos paso a paso, donde cada neurona tiene acceso a su propio historial de resultados. Esta estructura permite que la red se retroalimente, utilizando su memoria interna para influir en las predicciones futuras.

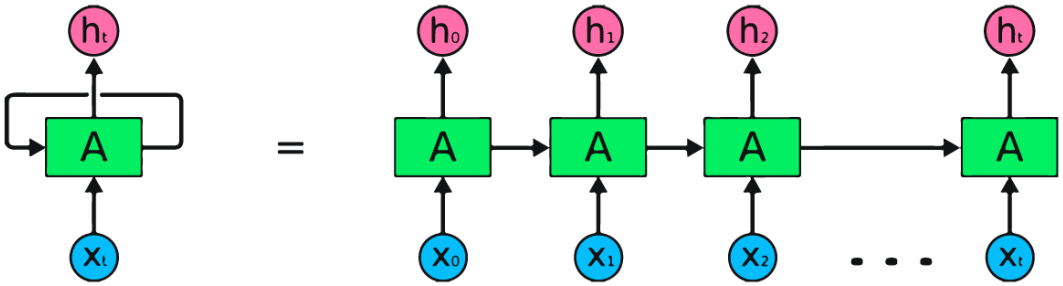

Figura 2.3: Desenrollamiento de capa de una RNN.

Existen varias diferencias clave entre las Redes Neuronales Convolucionales (CNN) y las Redes Neuronales Recurrentes (RNN).

En primer lugar, en las CNN la información a procesar viaja hacia adelante y pasa por una serie de filtros. Estos filtros permiten que las CNN capturen características espaciales y patrones dentro de los datos de entrada, como imágenes. Por el contrario, las RNN están diseñadas para manejar datos secuenciales y temporales; la información no solo viaja hacia adelante, sino que también puede retroalimentarse en la red. Esto significa que las RNN recogen los resultados finales y los vuelven a filtrar a través de la red neuronal, permitiendo que las capas recurrentes influyan en las entradas futuras.

Además, las entradas y salidas en las CNN son fijas. En cada pasada, una imagen de entrada específica produce una salida específica después de pasar por las capas convolucionales y las capas de conexión completa. Sin embargo, en las RNN, las entradas y salidas pueden variar en función de los resultados de las capas recurrentes.

Estas diferencias estructurales y funcionales conllevan a que cada tipo de red neuronal se utilice en diferentes ámbitos. Las CNN se utilizan mayoritariamente en tareas relacionadas con imágenes, como la detección de objetos, la clasificación de imágenes y la segmentación de las mismas. Son altamente eficaces en la extracción de características espaciales y patrones visuales. Por otro lado, las RNN se emplean en el procesamiento de datos secuenciales, como texto y vídeos. Son esenciales para aplicaciones que requieren un entendimiento temporal, como la traducción automática

2.2.3. Redes Neuronales Convolucionales para la detección de objetos (CNN)

Las redes neuronales convolucionales (CNN) (DataScientest, 2023) están especializadas en procesar datos en forma de matriz. En el contexto de este proyecto, necesitamos procesar imágenes para detectar patrones, y las CNN consideran la imagen como una cuadrícula bidimensional (2D) de píxeles. Estas redes han sido diseñadas específicamente para el procesamiento de imágenes, imitando el mecanismo de percepción visual humano. Además, las CNN utilizadas con este objetivo tienen

una arquitectura tridimensional (3D), esto quiere decir, que a parte de la altura y la anchura de la imagen, se toma como tercera dimensión a la profundidad de la red, es decir, las capas de procesamiento de la misma.

En primer lugar, las redes neuronales convolucionales (UPM, 2021) analizan pequeños fragmentos de información de una imagen, buscando descubrir patrones distintos en cada uno. Estas operaciones se realizan en una secuencia de fragmentos que conforman el primer bloque de convolución. Posteriormente, en las capas siguientes, las CNN intentan combinar estos bordes en formas más simples y luego en patrones que representan diferentes aspectos de los objetos, como su posición, iluminación y tamaño.

Finalmente, las capas finales buscan relacionar la imagen de entrada con todos estos patrones para llegar a una predicción final. Esta predicción se logra mediante una combinación ponderada de estos patrones, utilizando una red densa. En la Figura 2.4 (Mathworks, 2024) se puede ver un desglose de una CNN y, a continuación, se realiza una descripción de cada una de las capas que componen esta arquitectura y desempeñan un papel crucial en el procesamiento y análisis de imágenes:

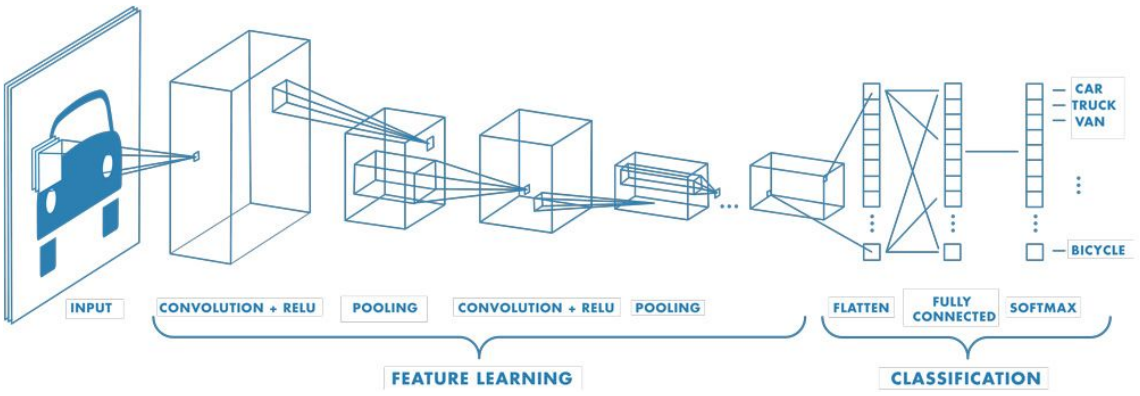

Figura 2.4: Fases de una CNN.

1. Capa de entrada: a partir de una imagen, se extraen sus dimensiones, incluyendo cada uno de los píxeles y los canales de color, generalmente RGB (Küppers, 2008).

2. Capa convolucional: se define el tamaño de la *ventana de filtro* situada en la parte superior izquierda de una imagen. La ventana de filtro es una pequeña región con un tamaño predefinido de x por y píxeles y z canales de color. A medida que el filtro se desliza sobre el ancho y la altura del volumen de imagen, se produce un mapa de activación en dos dimensiones que proporciona las respuestas de ese filtro en cada posición espacial. En otras palabras, en una capa de convolución, los píxeles se agrupan en ventanas de filtro que transforman la imagen. Durante este proceso de convolución, los valores de los píxeles en las ventanas de filtro se ajustan automáticamente. Así, cada filtro aprende a detectar patrones como bordes y ejes.

3. Capa de agrupación (Pooling): esta capa reduce el tamaño de la imagen y resalta sus características más notables mediante la operación de agrupación, que reduce un conjunto de ventanas de filtro a una ventana más pequeña, omitiendo píxeles sin contraste significativo. En el siguiente punto se explica en detalle este proceso. La operación de agrupación más común es el *Max Pooling*, en la cual, si las ventanas de filtros son de 3x3 píxeles, se selecciona el píxel con el mayor valor de color. Después de una capa de convolución, generalmente se aplica una capa de activación y una de agrupación. Se pueden añadir múltiples conjuntos de estas capas según sea necesario, permitiendo que el modelo agrupe figuras y patrones cada vez más complejos.

4. Capa de activación (ReLU): cada neurona pasa por una función de activación que considera los umbrales. La función de activación comúnmente utilizada es la ReLU (Rectified Linear Unit)(Jacar, 2023), que recoge todos los valores negativos recibidos en la entrada y los pone a 0, permitiendo que solo los valores positivos pasen. En la Figura 2.5 (UPM, 2021) se puede ver el proceso de la ventana de filtro en la capa de de activación.

5. Función SoftMax: esta función transforma la salida de los nodos en probabilidades. De este modo, en la salida de la red neuronal, se muestra la probabilidad de que una región de la imagen corresponda a un objeto específico.

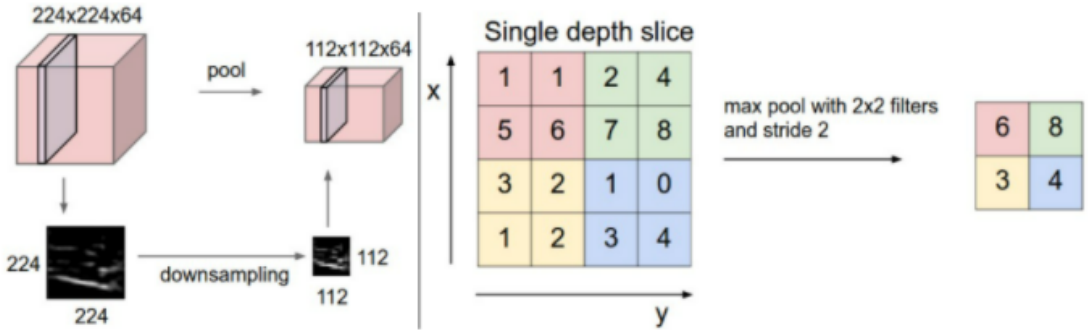

Figura 2.5: Proceso de una ventana de filtro.

2.3. Modelos de CNN para la detección de objetos

La detección de objetos (MathWorks, 2018) es una disciplina de la inteligencia artificial que capacita a los sistemas informáticos para identificar elementos dentro de una imagen al compararla con patrones previamente conocidos.

Para este TFG, se han elegido diversos modelos de CNN específicamente diseñados para la detección de objetos, centrándonos en la identificación de personas durante un evento. Estos modelos de CNN pueden realizar la detección de objetos en una o dos etapas. En este caso, se han seleccionado modelos que emplean una

sola etapa de detección debido a su mayor velocidad de procesamiento, lo que resulta fundamental para este proyecto, donde se requiere realizar las detecciones de personas prácticamente en tiempo real.

2.3.1. YOLO

YOLO (You Only Look Once) es un modelo muy popular dentro de la detección de objetos y la segmentación de imágenes (YOLO, 2023). Fue lanzado y desarrollado en 2015 por Joseph Redmon y Ali Farhadi en la Universidad de Washington y cuenta con nueve versiones disponibles.

Todas las versiones han mejorado aspectos más o menos significativos que sus predecesoras pero las más importantes son:

- YOLOv1: es el modelo primigenio que predice cuadros delimitadores y probabilidades de la clase del objeto directamente a partir de imágenes completas en una sola etapa.
- YOLOv4: en 2020 se introdujeron innovaciones como el aumento de datos mosaico y granulados, un nuevo cabezal de detección sin anclajes y una nueva función de pérdida.
- YOLOv5: mejoró el rendimiento y la optimización del modelo además de añadir el seguimiento de objetos en vídeos.
- YOLOv8: es la versión más novedosa y es la que utilizaremos para nuestro proyecto.

YOLOv8 es la mejor versión con la que cuenta actualmente la marca Ultralytics y con la que se ha trabajado durante este TFG. YOLOv8 cuenta con una gran variedad de modelos preentrenados para probar todas las tareas que puede realizar que se pueden ver en la Figura 2.6 (Ultralytics, 2024). Una propiedad destacada de este modelo es la detección sin anclaje. El modelo ya no va a contar con las ventanas de filtro previamente definidas. Ahora el modelo va a predecir el centro de un posible objeto en vez de recorrer la imagen de arriba a abajo al completo con una ventana. Con esta característica se reduce mucho el tiempo computacional y aumenta el rendimiento. Otra ventaja que tiene YOLOv8 es su propiedad de seguimiento de objetos cuando analiza vídeos. Esta propiedad permite mejorar el control de aforos mediante un pequeño trazado de la trayectoria de una persona en movimiento.

Para llevar a cabo la detección, primero se divide la imagen en ventanas de filtro. Por cada ventana predice de la forma más precisa con una probabilidad muy baja. Después de este proceso el modelo elimina esas ventanas cuyas predicciones sean menores a un valor concreto y, posteriormente, elimina también objetos detectados más de una vez.

YOLO ya se ha utilizado en algunos casos prácticos como en la aplicación *Latest Sightings* (Ossendryver, 2011), una plataforma para conectar a los aficionados

a los safaris con la vida salvaje del Parque Nacional Kruger de Sudáfrica. Ahora en vez de que los usuarios viajen por el safari virtualmente sin rumbo, con YOLO identifica avistamientos de animales y coloca marcadores en el mapa para señalar las ubicaciones a los usuarios. También se ha utilizado YOLO para evitar la caza furtiva de animales mediante su detección por visión por computadora en zonas del mundo con difícil acceso. Esto es una iniciativa de la Fundación Mundial de Cachemira (de Cachemira, 2008) para acabar con las empresas que cazan especies en peligro de extinción o difíciles de encontrar para la producción de artículos lujosos.

Figura 2.6: Tareas realizables por YOLOv8.

2.3.2. EfficientDet

EfficientDet (University, 2020a) un modelo de detección de objetos creado por Google Brain (Google, 2024b) que fusiona la eficiencia de la arquitectura EfficientNet (véase Figura 2.7) con la precisión de los algoritmos de detección de objetos más avanzados y de última generación. Este modelo primero sugiere posibles áreas de la imagen donde podrían estar ubicados los objetos antes de clasificarlos, lo que lo hace simple y eficiente, además de mejorar la capacidad de detección. Parecido a lo que hace YOLO.

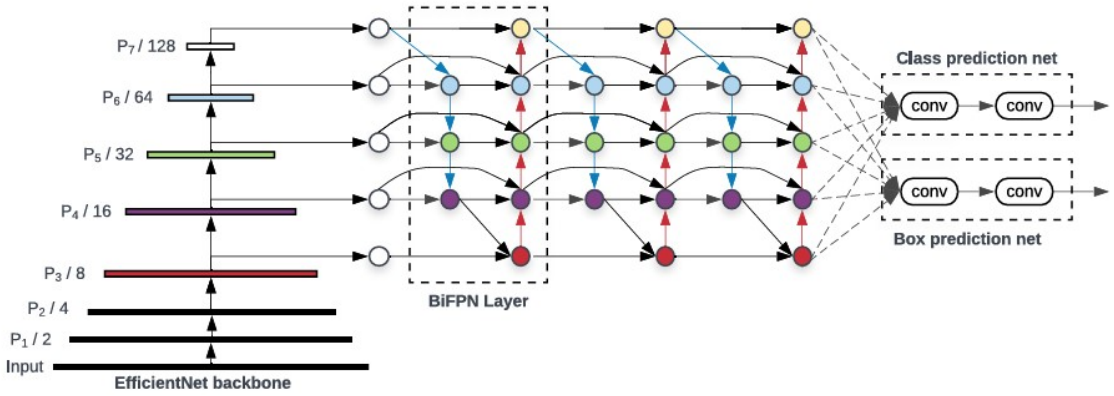

Figura 2.7: Arquitectura EfficientDet.

EfficientDet busca la combinación perfecta entre rendimiento y precisión para identificar objetos en imágenes, para ello se basa en dos optimizaciones:

- Una estructura de red piramidal de características bidireccionales ponderadas (BiFPN) que facilitan la fusión rápida y sencilla de características multiescala en ambas direcciones, de arriba hacia abajo y de abajo hacia arriba. Esto permite que la red considere tanto los detalles más pequeños como las características más generales en la detección de objetos.
- Un método de escalado combinado que ajusta simultáneamente todas las dimensiones clave del modelo EfficientDet, es decir, la red troncal (*backbone*), la estructura BiFPN y las redes de clasificación y detección de cuadros delimitadores. Este ajuste utiliza un coeficiente compuesto que equilibra tanto la precisión como la eficiencia del modelo para realizar cambios lineales y exponenciales en el ancho, la profundidad y la resolución del modelo, lo que provoca la creación de varios modelos EfficientDet, desde D0 a D7.

Como se acaba de explicar, EfficientDet presenta diferentes versiones, desde D0 hasta D7, cada una con variaciones en eficiencia y precisión. Estas versiones se distinguen por su capacidad para equilibrar la precisión en la detección de objetos con el costo computacional requerido para ejecutar el modelo.

La versión D0 es la más ligera en términos de recursos computacionales, lo que la hace adecuada para aplicaciones en las que se requiere una detección de objetos rápida y eficiente en dispositivos con recursos limitados. A medida que se avanza hacia las posteriores, la precisión en la detección de objetos aumenta, pero también lo hace la carga computacional necesaria para ejecutar el modelo. A continuación se presentan algunos detalles de dos versiones de EfficientDet:

- EfficientDet-D2 es la versión de EfficientDet que se ha implementado en este trabajo, es un modelo que espera una entrada con una resolución de 768x768 píxeles y alcanza un AP(Average Precision) de 43,9 en el conjunto de pruebas COCO (coco dataset, 2017) con 8,1 millones de parámetros y 11 mil millones de FLOP (operaciones en punto flotante por segundo).
- EfficientDet-D7, la última versión de la familia de detectores de objetos EfficientDet, es un modelo que espera una entrada con una resolución de 768x768 píxeles, alcanza un AP de 55,1 en el conjunto de pruebas COCO con 77 millones de parámetros y 410 mil millones de FLOP. Esta y otras variaciones de EfficientDet no se han podido llevar a cabo en este trabajo por falta de memoria.

EfficientDet se utilizó para el desarrollo del estudio: Clasificación de imágenes de ultrasonido mamario utilizando EfficientNetV2 y arquitecturas de redes neuronales (Nguyen, 2024), en la Universidad de Can Tho, Vietnam. . Consistía en clasificar imágenes de ultrasonidos para la prevención de cáncer de mama en tipos de tumores para identificar si eran malignos o benignos.

2.3.3. Single Shot Multibox Detector (SSD)

El método de detección de objetos SSD (*Single Shot MultiBox Detector*) utiliza una única red neuronal profunda para generar cuadros delimitadores y clasificar

objetos en imágenes. A diferencia de otros enfoques, como EfficientDet, que requieren generación de propuestas de objetos, SSD discretiza el espacio de salida de los cuadros delimitadores y combina múltiples mapas de características para manejar objetos de diferentes tamaños. Esto simplifica el proceso de detección al encapsular todos los cálculos en una sola red, facilitando tanto el entrenamiento como la integración en sistemas de detección.

La arquitectura de SSD se basa en una red convolucional que genera cuadros delimitadores de tamaño fijo y puntúa la presencia de clases de objetos en esos cuadros. Además, utiliza mapas de características a múltiples escalas para la predicción, predictores convolucionales para la detección y cuadros delimitadores predeterminados con diversas relaciones de aspecto para mejorar la eficiencia y precisión del modelo (veáse Figura 2.8).

Se ha visto en los experimentos en conjuntos de datos como PASCAL VOC (Tensorflow, 2022) o COCO que SSD logra una precisión competitiva en comparación con otros métodos, pero con una velocidad significativamente mayor.

No obstante, en este trabajo no se ha implementado una versión pura del modelo SSD, sino una variación basada en SSD-MobileNet. Esta combinación de ambas arquitecturas de redes neuronales opera con una entrada y resolución específicas. Como resultado, se obtiene mayor velocidad a costa de una menor precisión en comparación con un SSD estándar, que utiliza una arquitectura más profunda y compleja. Esta reducción en precisión se debe a que MobileNet prioriza la eficiencia computacional y la velocidad de procesamiento sobre la precisión.

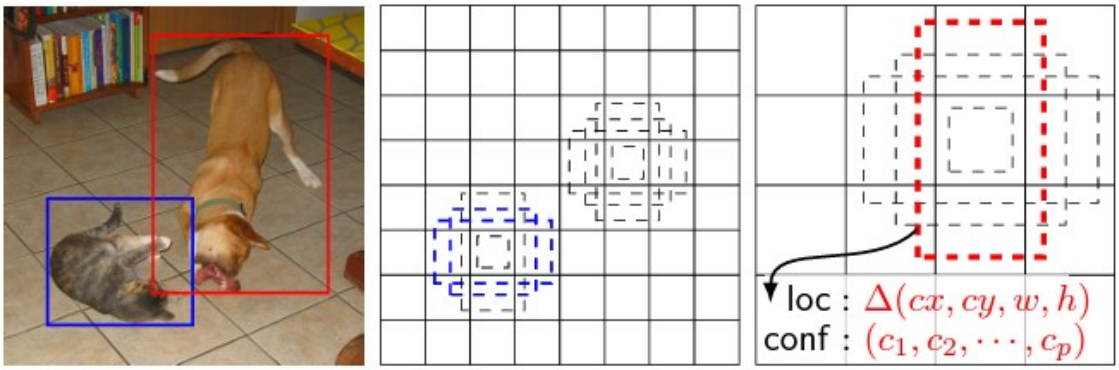

Figura 2.8: Funcionamiento SSD.

2.3.4. MobileNet

MobileNet (University, 2017) es una arquitectura de red neuronal convolucional simple pero eficiente, diseñada específicamente para aplicaciones de visión por computadora en dispositivos móviles, donde el rendimiento en tiempo real y la eficiencia computacional son críticos.

Una de las técnicas clave empleadas en MobileNet es la convolución separable en profundidad, que descompone las convoluciones convencionales en dos tipos de capas distintas:

- Capa convolucional *Depthwise*: Esta capa realiza una convolución individual en cada canal de entrada por separado, en lugar de aplicarla a todos los canales simultáneamente. De este modo, se efectúa una convolución independiente para cada canal, lo que permite capturar las características específicas de cada uno de manera más eficiente.
- Capa convolucional 1x1 *Pointwise*: Después de la capa depthwise, se lleva a cabo una convolución 1x1 para fusionar las características extraídas previamente. Aquí, la convolución 1x1 opera como una operación de mezcla, combinando las características de los distintos canales en combinaciones lineales.

Aunque la arquitectura base de MobileNet ya es pequeña y computacionalmente eficiente, ofrece dos hiperparámetros globales adicionales para reducir aún más el costo computacional:

- Multiplicador de ancho: Este parámetro se multiplica por los canales de entrada y salida para estrechar la red.
- Multiplicador de resolución: Este parámetro se multiplica por el mapa de características de entrada para ajustar la resolución de las imágenes.

SSD y MobileNet se utilizaron para la implementación de Google Lens (Lens, 2024), una aplicación que funciona como un buscador visual. En vez de introducir texto se introduce una imagen detectando objetos que aparecen en fotos, desde monumentos hasta animales, pasando por libros, CD o plantas.

Capítulo 3

Arquitectura del sistema

En este capítulo se explora la arquitectura del sistema (véase Figura 3.1) que se pretende llevar a cabo en este TFG, tanto la parte de *hardware* como la parte de *software*. La arquitectura consiste en una CPU desde la cual se ejecuta un script de python que genera una interfaz. Esta permite al usuario conectarse al dispositivo de bajo consumo, en este caso una Raspberry, y especificar un modelo de deep learning con el que se pretende realizar la detección de personas, para finalmente mostrar los resultados obtenidos.

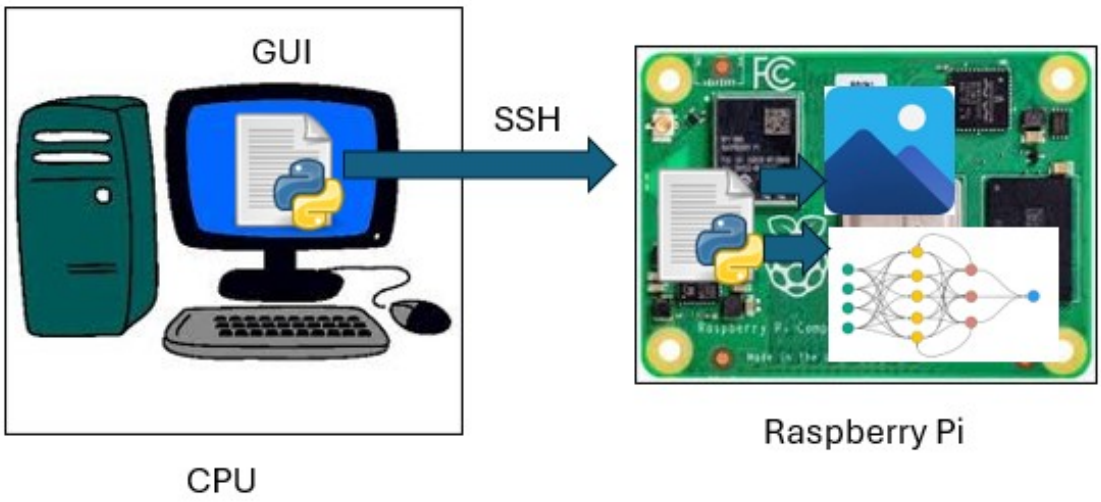

Figura 3.1: Arquitectura general del sistema.

Como se ha mencionado previamente, el objetivo de este trabajo es supervisar el aforo de un evento específico utilizando redes neuronales convolucionales. Estas redes deben ser integrables en dispositivos de bajo consumo o de *edge computing* (IBM, 2023), ya que el control del aforo requiere respuestas en tiempo real y no puede tolerar largos tiempos de espera.

El edge computing consiste en analizar y procesar la información en el punto donde nace. La escala y la complejidad de estos datos están superando todas

aquellas infraestructuras creadas para centralizar y procesar la información, hasta tal punto en el que el tiempo desde que se recopilan una serie de datos, se envían, se procesan y dan respuesta, es demasiado grande. Por ello, el edge computing es una solución a este problema, ya que permite analizar los datos al instante para proporcionar un conocimiento profundo y análisis predictivo en tiempo real.

3.1. Software del sistema

El éxito de cualquier sistema computacional depende en gran medida del software que lo impulsa. En este apartado, se exploran las herramientas y tecnologías de software utilizadas en la arquitectura del sistema, comenzando por la interfaz gráfica, programada para realizar una conexión automática con la Raspberry que permite al usuario realizar inferencias con las modelos de deep learning desarrollados. También se explican los principales entornos y librerías utilizados, finalizando así la explicación con el procesamiento de los datos de entrada al sistema.

3.1.1. Interfaz gráfica - GUI

Una interfaz gráfica de usuario (GUI) es un tipo de interfaz que permite al usuario interactuar con un sistema computacional a través de elementos visuales y gráficos en lugar de comandos textuales como se hace en una terminal.

Las GUI emplean componentes visuales de todo tipo, aunque la que se ha desarrollado para este TFG es muy sencilla (véase Figura 3.2) y únicamente muestra una lista desplegable con el nombre de los distintos modelos entrenados con los que se ha trabajado. Esta GUI está conectada a la Raspberry, por tanto, a la hora de seleccionar un modelo, se ejecuta en la placa de bajo consumo un script asociado a este, que devuelve a la interfaz un resultado, es decir, una imagen con todas sus detecciones y el número total de esas detecciones (el aforo).

El objetivo principal de desarrollar esta GUI es facilitar al usuario el acceso a la Raspberry y probar los modelos entrenados.

3.1.2. Entorno de desarrollo - Google Colab

Es importante explicar el entorno de desarrollo, que se ha utilizado para entrenar los modelos de inteligencia artificial: Google Colab (Google, 2024c).

Google Colab, un servicio alojado de Jupyter Notebook (Jupyter, 2024) en la nube, se destaca como una herramienta poderosa y versátil que elimina la necesidad de configuración previa. Proporciona acceso gratuito a recursos informáticos, incluyendo GPU (Unidades de Procesamiento Gráfico) muy potentes y rápidas y TPU (Unidades de Procesamiento Tensorial), lo que permite la ejecución de código en Python y la utilización de librerías populares en el ámbito de la inteligencia artificial, como TensorFlow (Tensorflow, 2024b), YOLO (YOLO, 2023) y PyTorch (Pytorch, 2024). Está especialmente orientado hacia el aprendizaje automático, la

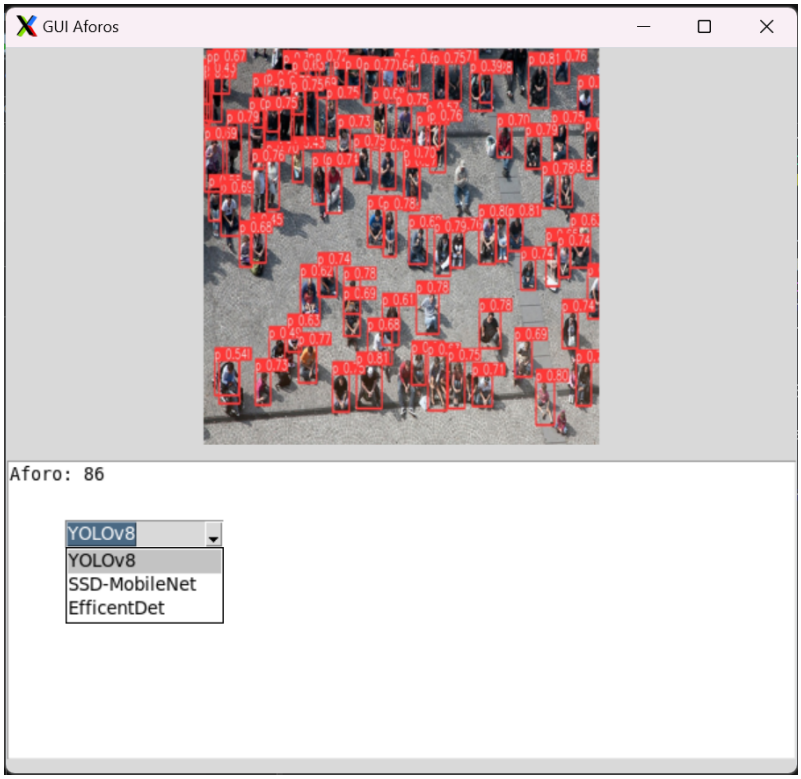

Figura 3.2: Interfaz gráfica del sistema.

ciencia de datos y la educación, tanto para principiantes como para usuarios avanzados, además de ofrecer una interfaz de usuario intuitiva y fácil de usar.

A pesar de sus ventajas, los recursos de Google Colab no son garantizados ni ilimitados, y los límites de uso pueden variar. Estas restricciones son necesarias para mantener el servicio gratuito.

En Google Colab, el código de Python se ejecuta en una máquina virtual vinculada a la cuenta de Google. Sin embargo, después de un periodo de inactividad, la sesión se desconecta. En la versión gratuita, Google Colab permite la ejecución de un *notebook* durante un máximo de 12 horas. Un notebook es un conjunto de celdas que ejecutan "trozos´´ de código en python.

Una de las características destacadas de Google Colab es su capacidad para crear y compartir notebooks de Jupyter de manera colaborativa. Esto permite trabajar en un mismo notebook con otros usuarios en tiempo real, lo que lo convierte en una herramienta ideal para proyectos colaborativos o para recibir retroalimentación.

Además, los notebooks de Google Colab se pueden guardar directamente en Google Drive, simplificando la gestión y el acceso a los proyectos desde cualquier lugar. También es posible importar y exportar datos directamente desde y hacia Google Drive, lo que facilita el manejo de conjuntos de datos y archivos. En proyectos como este TFG, esta funcionalidad resulta especialmente útil para importar

grandes conjuntos de datos utilizados para el entrenamiento del modelo.

En la Figura 3.3 se muestra una imagen de la interfaz que ofrece Google Colab. La disposición de la interfaz se detalla a continuación:

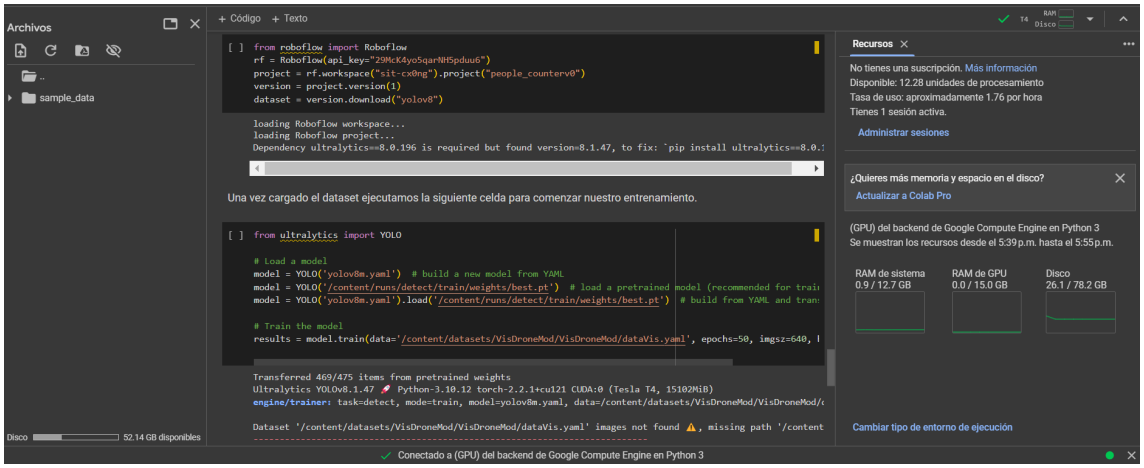

Figura 3.3: Interfaz de Google Colab.

- En el panel izquierdo se muestra el árbol de directorios del notebook en uso. Este árbol organiza los archivos necesarios para la ejecución del código, y así como los archivos descargados y los directorios clonados, según la jerarquía especificada por el usuario.
- En la esquina inferior izquierda, se muestra la cantidad de espacio en disco que el notebook consume.
- El panel central consiste en celdas que contienen instrucciones y sus salidas, las cuales son ejecutadas por el usuario.
- En el panel derecho se indican los recursos disponibles para el usuario al ejecutar las diversas celdas del Notebook. Estos recursos incluyen la RAM del sistema, la RAM asociada GPU, y la capacidad del disco. En la versión Pro, también se muestran el número de unidades informáticas disponibles junto con su tasa de uso, y así como el número de sesiones activas que indican la cantidad de proyectos en ejecución simultánea.

3.1.3. Ultralytics

Ultralytics (Ultralytics, 2022) es una empresa que desarrolla software y herramientas para aplicaciones de visión por computadora y aprendizaje automático. Su instalación es fundamentalmente necesaria para la implementación de YOLO, ya que entre sus productos se encuentra una librería de código abierto llamada "PyTorch-YOLOvX", que es una implementación en PyTorch de la arquitectura X del algoritmo YOLO.

Es importante saber que YOLOv8 cuenta con varias arquitecturas distintas. En función de las necesidades del programador es necesario escoger una u otra (YOLOv8, 2023). En la Tabla 3.1 se muestra una tabla con las características que distinguen a cada una de estas arquitecturas.

Modelo	dimensiones	mAP50-95	CPU ONNX (ms)	A100 (ms)	FLOPs
YOLOv8n	640	37,3	80,4	0,99	8,7
YOLOv8s	640	44,9	128,4	1,2	28,6
YOLOv8m	640	50,2	234,7	1,83	78,9
YOLOv8l	640	52,9	375,2	2,39	165,2
YOLOv8x	640	53,9	479,1	3,53	257,8

Tabla 3.1: Diferencias entre arquitecturas de YOLOv8.

3.1.4. Tensorflow y tensorflow lite

TensorFlow es una librería de software de código abierto desarrollada por Google para el aprendizaje automático y la inteligencia artificial que ofrece una serie de características y herramientas que lo hacen muy flexible y eficaz. Se lanzó por primera vez en 2015 y se ha convertido en una de las librerías más populares y ampliamente utilizadas en el campo del aprendizaje automático, tanto que se ha utilizado en este TFG para implementar los modelos SSD-MobileNet y EfficientDet.

Durante este TFG se ha aprovechado su flexibilidad para trabajar con redes neuronales convolucionales y su alta escalabilidad que ha permitido entrenar modelos con grandes conjuntos de datos distribuidos en múltiples dispositivos. Además, ofrece una API que proporciona una serie de modelos de detección de objetos preentrenados y herramientas para entrenar y evaluar modelos personalizados.

Por otro lado, en este trabajo también se ha hecho uso de TensorFlow Lite (Tensorflow, 2024c), una versión de TensorFlow diseñada específicamente para aplicaciones de aprendizaje automático en dispositivos móviles y sistemas integrados, como teléfonos inteligentes, tabletas, dispositivos IoT (Internet de las cosas) y otros dispositivos con recursos computacionales limitados, ya que permite ejecutar modelos de forma rápida y con un bajo consumo de energía.

Es por ello, que TensorFlow Lite puede ser una herramienta útil para la implementación de modelos de aprendizaje profundo en tiempo real en un dispositivo de bajo consumo.

3.1.5. Procesamiento de los datos de entrada

El procesamiento de los datos de entrada en el software del sistema se realiza mediante diversos scripts que comparten la misma premisa. Estos scripts varían según el tipo de datos de entrada recibidos, ya sea una imagen estática o un flujo de vídeo, y así como de la librería utilizada para desarrollar el modelo.

El propósito principal es adecuar el dato de entrada de manera que se aproveche la capacidad del modelo previamente entrenado para llevar a cabo la detección de personas. Este proceso implica dibujar un cuadro delimitador sobre cada objeto detectado en la imagen y llevar un recuento del total de detecciones.

Para acceder al código de la interfaz, de los notebooks o de los scripts, hemos creado un repositorio en Github (Sánchez-Rodilla, 2024) donde se encuentra todo el software programado del sistema. Los notebooks están explicados paso a paso, los scripts también están comentados.

3.2. Hardware del sistema

En cuanto al hardware del sistema hay que resaltar, por un lado, el hardware utilizado para la ejecución de los entrenamientos de los modelos y, por otro lado, el dispositivo de bajo consumo en el que se realizarán las inferencias.

3.2.1. Entorno de ejecución - Google Colab Pro

Es importante explicar el entorno de ejecución, Google Colab Pro (Google, 2023). Este entorno cuenta con una gran variedad de GPU para realizar los entrenamientos de los modelos. Dado que los entrenamientos son procesos de muchas horas, es necesario contar con acceso a hardware que los agilice en la medida de lo posible, siendo aún así demasiado largos.

Google Colab Pro es una versión mejorada y de Google Colab que ofrece a los usuarios la capacidad de conectarse a GPU y TPU más potentes y con mayor capacidad. Esta ventaja la puede adquirir cualquier usuario siempre que disponga de suficientes unidades informáticas. Esto supera algunas de las limitaciones presentes en la versión gratuita, y además, brinda la posibilidad de utilizar un terminal con la máquina virtual conectada.

Cualquier usuario con una cuenta de Google puede acceder a esta versión mejorada mediante una suscripción mensual que proporciona 100 unidades informáticas para conectar con las GPU con más calidad. Sin embargo, estas GPU tienen un alto consumo, lo que significa que los usuarios pueden necesitar contratar más unidades informáticas, incluso después de suscribirse para seguir disfrutando de las ventajas de Google Colab Pro.

Durante el desarrollo de este TFG, nos hemos visto en la necesidad de realizar varias suscripciones a esta versión mejorada y comprar unidades informáticas adicionales para aprovechar las ventajas que ofrece este servicio. Esto se debe a que, en ocasiones, la versión gratuita no permitía acceder a las GPU, los entrenamientos de los modelos excedían el tiempo de ejecución máximo permitido por la versión gratuita, y algunos modelos requerían más memoria de la que ofrece incluso Colab Pro.

Estas GPU sirven para entrenamiento, estaciones de trabajo de visualización remota, transcodificación de vídeo y HPC (computación de alto rendimiento). Se enumeran las GPU disponibles (Google, 2024d) en la Tabla 3.2.

Modelo de GPU	Memoria (Gb)	Rendimiento (TFLOPS)
NVIDIA Tesla T4	16	0.25
NVIDIA L4	16	0.5
NVIDIA V100	16	7.8
NVIDIA A100	16	9.7

Tabla 3.2: GPU de Google Colab Pro.

3.2.2. Raspberry

Los tutores de este TFG han proporcionado el acceso a un sistema Raspberry para evaluar los modelos utilizados durante el trabajo.

Una Raspberry Pi es un hardware versátil y compacto formado por una serie de procesadores simples de bajo coste. Todos los diseños de Raspberry Pi se basan en el hardware libre y habitualmente se utilizan también sistemas operativos libres basados en GNU/Linux, normalmente tienen una versión personalizada de Debian.

Figura 3.4: Raspberry Pi 4-1.

El dispositivo con el que se ha trabajado es, en concreto, una Raspberry Pi 4-1 (veáse Figura 3.4). Esto significa que estamos utilizando una Raspberry Pi modelo 4, específicamente el modelo identificado como “raspi4-1”. Sus propiedades son las siguientes:

- Procesador Quad Core, cuatro núcleos Cortex-A72 64-bit @ 1.8GHz.

- 8GB Memoria RAM
- 2 puertos USB 3.0
- Distribución Linux basada en Debian, Kernel 6.1.0 compilado para arquitectura ARMv8 (aarch64)

Para realizar una correcta conexión a la Raspberry Pi que hemos utilizado, desde un ordenador personal es necesario realizar dos pasos:

1. En primer lugar, conectarse a la VPN (red privada virtual) de la UCM a través del programa Global Protect. Dado que la Raspberry Pi se encuentra físicamente en la UCM, es necesario utilizar la VPN para poder acceder a la misma.
2. En segundo lugar, realizar una conexión ssh, con unas credenciales y puerto proporcionados, para acceder a la Raspberry.

Capítulo 4

Procesos de Entrenamientos

En este capítulo se detalla cómo se han preparado los conjuntos de datos y configurado los entrenamientos para los diferentes modelos utilizados en este TFG: SSD-MobileNet, EfficientDet y YOLOv8. Algunos de estos procesos se explican de forma paralela para SSD-MobileNet y EfficientDet, por la similitud de estos al utilizar ambos la librería TensorFlow. Esto significa que se ha seguido un enfoque común para preparar el conjunto de datos y entrenamiento de ambos modelos, lo que facilita la comparación y la comprensión de los resultados obtenidos con cada uno de ellos.

4.1. Conjuntos de datos

La importancia del conjunto de datos en el desempeño final de la red neuronal radica en su capacidad para influir en todos los aspectos del proceso de aprendizaje. Desde la inicialización de los parámetros de la red hasta la evaluación y ajustes finales del modelo, los datos actúan como el sustrato sobre el cual se desarrolla el conocimiento y la capacidad predictiva de la red.

En el contexto del aprendizaje automático o profundo, un conjunto de datos bien preparado es la materia prima que alimenta el proceso de entrenamiento de la red, determinando en gran medida su capacidad para generalizar y tomar decisiones precisas en entornos diversos y complejos.

Un conjunto de datos de alta calidad se caracteriza por varios aspectos cruciales. En primer lugar, debe ser representativo de los escenarios reales a los que la red se enfrentará en su aplicación. Esto implica una amplia diversidad en los datos para capturar la variabilidad inherente a las situaciones del mundo real.

Además de la representatividad, la calidad de los datos también se relaciona con su integridad y consistencia. Los datos incompletos, incorrectos o sesgados pueden introducir distorsiones en el proceso de entrenamiento, afectando negativamente a la capacidad de la red para aprender patrones precisos y generalizar adecuadamente.

Otro aspecto importante es la cantidad de datos disponibles para el entrenamiento. En general, se considera que entrenar con más datos es mejor, ya que proporcionan a la red una mayor diversidad de ejemplos para aprender y generalizar. Sin embargo, la calidad sigue siendo prioritaria sobre la cantidad; es preferible un conjunto de datos pequeño, pero bien construido a uno grande ruidoso o incompleto.

En este TFG se ha confiado en la utilización de conjuntos de datos puestos al público y ya etiquetados por parte de otros usuarios, ya que el proceso de construcción de un conjunto de datos desde cero excede los límites de este TFG. Esta confianza de la que se habla, ha afectado de forma negativa e inevitable a este trabajo. Como se observa en la Figura 4.1, algunos datos de entrada que se han utilizado en este TFG, no tienen buena calidad, por ejemplo, una imagen borrosa, o están incompletos, por ejemplo, una imagen que no está etiquetada en su totalidad.

Figura 4.1: Imágenes de mala calidad pertenecientes al conjunto de datos utilizado.

El conjunto de datos utilizado para entrenar EfficientDet, SSD-MobileNet y YOLOv8 está formado por varios subconjuntos de datos unificados todos ellos en un único conjunto. Para el modelo EfficientDet no se ha utilizado exactamente el mismo conjunto de datos, si no un conjunto más pequeño compuesto mayoritariamente de datos de entrada también pertenecientes al conjunto de datos utilizado para entrenar SSD-MobileNet y YOLOv8. La explicación de esta acción se detalla en el punto 4.2.1.3.

Todos y cada uno de los subconjuntos de datos se encuentran almacenados y han sido descargados desde Roboflow (Roboflow, 2024). Esta es una plataforma que ofrece herramientas para gestionar, etiquetar, mejorar y entrenar modelos de inteligencia artificial utilizando conjuntos de datos de imágenes, además de actuar como un repositorio de conjuntos de datos de imágenes etiquetadas.

Los conjuntos de datos descargados contienen imágenes de multitud de per-

sonas vistas desde el aire y desde diferentes ángulos, para así simular la perspectiva que tendría una imagen tomada por un dron o una cámara de vigilancia. El conjunto de datos contiene imágenes de mucha y poca calidad, en color y en blanco y negro, etc.

4.1.1. Estructura de los datos

Una vez que se han recolectado todos los datos requeridos para entrenar un modelo, se dividen estos en tres grupos diferentes: el conjunto de entrenamiento (*train*), el conjunto de validación (*valid*) y el conjunto de prueba (*test*). Esta separación nos permite realizar varias tareas importantes. En las Tablas 4.1 y 4.2 se muestra la estructura de los datos de cada conjunto utilizado para el entrenamiento y evaluación de cada modelo.

Conjunto	test	train	valid
Nº imágenes	172	1913	336

Tabla 4.1: Estructura del conjunto de datos para EfficientDet.

Conjunto	test	train	valid
Nº imágenes	793	22.627	2.241

Tabla 4.2: Estructura del conjunto de datos para SSD-MobileNet y YOLOv8.

Primero, se utiliza el conjunto de entrenamiento para enseñar a la red neuronal, permitiéndole aprender patrones y características útiles de los datos. Mientras tanto, el conjunto de validación ayuda a monitorizar el rendimiento del modelo durante el entrenamiento, evitando que se ajuste demasiado a los datos de entrenamiento (sobreajuste), lo que podría llevar a un rendimiento deficiente en datos nuevos.

Una vez que el modelo ha sido entrenado, se utiliza el conjunto de prueba para evaluar su precisión y medir su rendimiento en datos que no ha procesado antes. Esto proporciona una idea clara de como de bien generaliza el modelo a nuevos datos y nos ayuda a tener una evaluación imparcial de su desempeño final.

Como evidencian las Tablas 4.1 y 4.2, la magnitud del conjunto de datos destinado al entrenamiento y evaluación de SSD-MobileNet y YOLOv8 es notablemente mayor que la asignada para EfficientDet. La explicación de este suceso, como se ha mencionado antes, se detalla en el punto 4.2.1.3

4.1.2. Formato de los datos

Desde el repositorio de Roboflow se puede descargar un conjunto de datos en distintos formatos. Es fundamental comprender que el formato del conjunto de datos depende del modelo que se esté evaluando.

4.1.2.1. Formato de los datos para EfficientDet y SSD-MobileNet

Para poder entrenar los modelos EfficientDet y SSD-MobileNet se ha realizado la descarga en formato PASCAL VOC (*Visual Object Classes*), un conjunto de datos ampliamente utilizado en el campo de la visión por computadora y el aprendizaje profundo.

El conjunto de datos está formado por pares de archivos JPG/ JPEJ y XML. A continuación, se muestra parte de un fichero XML de anotación perteneciente al conjunto de datos utilizado, que contiene información relevante, como el nombre de la carpeta que contiene la imagen, el nombre y la ruta del archivo de la imagen, el nombre de la base de datos de la que provienen los datos, la anchura y altura de la imagen en píxeles, la profundidad del color de la imagen, el nombre del objeto, un indicador de si el objeto es difícil de reconocer o está oculto, la posición del objeto, etc.

```
<annotation>
    <folder></folder>
    <filename>img_101.jpg</filename>
    <path>img_101.jpg</path>
    <source>
        <database>roboflow.ai</database>
    </source>
    <size>
        <width>216</width>
        <height>216</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>people</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>29</xmin>
            <xmax>35</xmax>
            <ymin>50</ymin>
            <ymax>63</ymax>
        </bndbox>
    </object>
</annotation>
```

4.1.2.2. Formato de los datos para YOLOv8

El conjunto de datos utilizado para YOLOv8 tiene el mismo origen que el utilizado para SSD-MobileNet o EfficientDet. Sin embargo, el formato no es el mismo y por ello, existen diferencias entre los ficheros de anotaciones. Para YOLOv8, se

ha realizado la descarga en formato YOLOv8. Este formato implica que los ficheros de anotaciones siguen un formato de texto, en donde cada fila corresponde con un objeto seguido de información relacionada con este, como el tipo o las coordenadas. A continuación, se muestra parte de un fichero TXT de anotación perteneciente al conjunto de datos utilizado.

```
0 0.1390625 0.459375 0.01875 0.03359375
0 0.18515625 0.46484375 0.01484375 0.02890625
0 0.36640625 0.53828125 0.02578125 0.02578125
0 0.40390625 0.3765625 0.0140625 0.028125
0 0.31171875 0.28828125 0.0140625 0.0234375
0 0.3078125 0.22109375 0.01640625 0.0296875
0 0.3828125 0.190625 0.01640625 0.0296875
0 0.496875 0.19609375 0.0171875 0.0265625
```

4.1.3. Modificaciones de los datos

Se han realizado algunas modificaciones importantes en los conjuntos de datos para ejecutar un entrenamiento de calidad. Al estar formado cada conjunto de datos por varios subconjuntos de datos, cada uno de estos subconjuntos contienen anotaciones de objetos que no interesa detectar, como por ejemplo, *bicycle* o *car*. Este TFG se centra únicamente en detectar y contar las personas que aparecen en una imagen de entrada. Además, cada subconjunto establece en los ficheros de anotaciones un nombre de clase distinto para el objeto que se clasifica como persona.

En primer lugar, se ha programado un script que recorre todos los ficheros de anotaciones XML del conjunto de datos unificado, para establecer el mismo nombre en todos los objetos que cada subconjunto de datos detecta como personas, ya que en uno se clasifica como 'h', en otro como 'Human', etc. Además, se eliminan de los ficheros de anotaciones todos los objetos que no interesa detectar para conseguir un entrenamiento más eficiente y aumentar la precisión del modelo.

Por otro lado, muchas arquitecturas de redes neuronales convolucionales requieren que todas las imágenes de entrada tengan el mismo tamaño. Esto se debe a que las capas convolucionales y de agrupación de la red están diseñadas para operar sobre conjuntos de imágenes con las mismas dimensiones.

De esta forma, al tener todas las imágenes de entrada del mismo tamaño, el modelo puede aprender patrones y características de manera más efectiva, ya que no tiene que lidiar con variaciones en el tamaño de las imágenes.

Por ello, se ha programado un script, que recorre todos los pares de archivos JPG/ JPEJ y XML del conjunto de datos utilizado para SSD-MobileNet y EfficientDet, con el fin de realizar modificaciones algunas modificaciones en este. En los archivos JPG/JPEJ redimensiona la imagen al valor establecido (320x320 para SSD-MobileNet y 768x768 para EfficientDet) manteniendo la proporción entre altu-

ra y anchura de la imagen original rellenando con píxeles vacíos (color blanco) en algunos casos. Se rellena con píxeles de color blanco y no negro porque desde una perspectiva aérea, un píxel negro se puede confundir con parte de la silueta de la cabeza de una persona. Por otro lado, en el fichero de anotación XML asociado a la imagen modificada, actualiza la posición de cada objeto anotado, ya que al redimensionar la imagen, los objetos cambian de posición.

Sin embargo, no es necesario redimensionar las imágenes del conjunto de datos en formato YOLOv8. Cuando se descargan los datasets de Roboflow en este formato, todas las imágenes tienen la misma resolución: 640x640. Esto se debe a que las arquitecturas de YOLOv8 tienen fijadas estas dimensiones para la entrada. Por ello, no es necesario programar ningún script para este aspecto como se ha hecho para SSD-MobileNet y EfficientDet.

4.2. Preparación y configuración del entrenamiento

En esta sección se explica el proceso de preparación y configuración de parámetros que se utilizarán en el proceso del entrenamiento de los modelos.

La preparación y configuración del entrenamiento son aspectos críticos en el desarrollo de modelos de deep learning, ya que pueden influir en la calidad, el rendimiento y la eficiencia del modelo final. Invertir tiempo y esfuerzo en estos aspectos puede conducir a modelos más efectivos y robustos.

4.2.1. Preparación y configuración del entrenamiento de EfficientDet y SSD-MobileNet

Como ya se ha comentado al inicio del capítulo, la preparación y configuración de los entrenamientos de EfficientDet y SSD-MobileNet se harán de forma conjunta, ya que al utilizar ambos modelos la librería de Tensorflow, los procesos de entrenamiento son similares.

4.2.1.1. Conversión de los datos a TFRecord

En primera instancia es necesario convertir los datos al formato ideal utilizado en TensorFlow, *TFRecord* (Microsoft, 2024), para realizar entrenamientos de modelos de aprendizaje profundo.

TFRecord es un formato de datos binarios utilizado en TensorFlow para almacenar datos de manera eficiente y facilitar su lectura y procesamiento durante el entrenamiento de modelos de aprendizaje profundo. Está diseñado para ser rápido y escalable, especialmente cuando se trabaja con grandes conjuntos de datos.

Para conseguir esta conversión de los datos, se hace uso de un script programado que convierte los ficheros de anotaciones XML en ficheros CSV. Con este

nuevo formato y conversión intermedia, podemos realizar una transformación de los datos al formato final deseado, es decir, pasar los datos en CSV a TFRecord.

4.2.1.2. Especificación del modelo a implementar

A continuación, se especifica qué modelo preentrenado de TensorFlow se va a desarrollar desde el *Zoológico del modelo de detección de TensorFlow 2* (Team, 2021), una colección de modelos de detección de objetos previamente entrenados sobre el conjunto de datos COCO 2017.

En este apartado, por una lado, se especifica el modelo 'EfficientDet D2 768x768', una variante específica del modelo EfficientDet que trabaja con una entrada de tamaño 768x768, y que consigue una velocidad de 67ms y una precisión de 41.8 COCO mAP (esta métrica de precisión se explica el capítulo 5 de la memoria) sobre el conjunto de datos COCO 2017. No se ha podido implementar una versión más alta de EfficientDet, con mayor velocidad y precisión, por falta de memoria.

Por otro lado, se especifica el modelo 'SSD MobileNet v2 320x320', una variante específica del modelo SSD-MobileNet que trabaja con una entrada de tamaño 320x320. Este modelo preentrenado tiene una velocidad de 19 milisegundos y una precisión de 20.2 COCO mAP sobre el conjunto de datos COCO 2017.

4.2.1.3. Configuración de los parámetros del entrenamiento

En este apartado se explican algunos parámetros del fichero de configuración de cada modelo que han sido modificados para conseguir un entrenamiento eficiente y unos resultados precisos.

En primer lugar, para el fichero de configuración del modelo 'EfficientDet D2 768x768' se han establecido los siguientes valores para los parámetros correspondientes:

- *fine_tune_checkpoint_type = Detection*: Este parámetro se refiere al tipo de archivo que se utiliza como punto de partida para el ajuste fino (fine-tuning) de un modelo de aprendizaje profundo. Cuando se realiza el ajuste fino, se toma un modelo preentrenado y se adapta para un problema específico o conjunto de datos.
- *total_steps = 48325*: Este parámetro establece la cantidad total de pasos a utilizar para entrenar un modelo. No es una cantidad muy vistosa, pero es la exacta para conseguir recorrer cien veces el conjunto de datos en su totalidad. Si al terminar el entrenamiento, las métricas siguen disminuyendo, entonces conviene aumentar este valor, aunque cuantos más pasos, más tiempo llevará el entrenamiento.
- *batch_size = 4*: Este parámetro indica el número de imágenes a utilizar por paso de entrenamiento. Un tamaño de lote más grande permite que un modelo se entrene en menos pasos, pero este está limitado por la memoria de la GPU

disponible para el entrenamiento. En este TFG, no se ha podido establecer un valor mayor, ya que se ha trabajado al límite con todos los recursos, en este caso el de la memoria, como se puede observar en la Figura 4.2. Esta figura fue extraída durante un entrenamiento de EfficientDet en Google Colab.

En la sección anterior se explica que no se ha podido utilizar el mismo conjunto de datos para SSD-MobileNet que EfficientDet y YOLOv8. Esta decisión se fundamenta en el proceso de entrenamiento llevado a cabo para EfficientDet, cuyo tamaño de lote es excesivamente pequeño, como se acaba de ver, en comparación al establecido por defecto (128), lo que conlleva a un entrenamiento mucho más duradero del habitual.

Para el conjunto de datos de más de veintidós mil imágenes y un entrenamiento de más de veinte horas, EfficientDet obtenía resultados poco satisfactorios, a pesar del largo tiempo de ejecución del entrenamiento. Este suceso se producía ya que al haber un tamaño de lote tan pequeño, tanto tiempo de entrenamiento no era suficiente para procesar y aprender de tantos datos de entrada.

Por consiguiente, en este TFG, se optó por priorizar la utilización de un conjunto de datos más reducido para EfficientDet. Esta decisión se sustenta en el principio de que el modelo aprenda 'mucho de poco', en lugar de 'poco de mucho'.

- *matched_threshold = 0.5*: Este parámetro se refiere a como de similar deben ser dos o más detecciones para considerarse coincidentes y por tanto fusionarse. Se ha establecido un valor intermedio, '0.5', que significa que si la similitud entre las detecciones es mayor que el 50 %, se considerarán como una coincidencia. Para datos de entrada donde aparezcan personas muy juntas nos interesaría aumentar ese valor, ya que si hay, por ejemplo, dos cabezas muy juntas, podría darse el caso de que sus detecciones se fusionaran. En cambio, para datos de entrada donde las personas aparezcan dispersas, convendría aumentar este valor para realizar una detección más exigente.

- *max_detections_per_class, max_total_detections & max_number_of_boxes = 2000*: Estos parámetros se refieren al máximo de detecciones en total, se establece un número alto para que sea válido para todas las entradas de datos.

- *num_classes = 1*: Este parámetro indica el número de clases que el modelo está configurado para reconocer o predecir. Al estar establecido a "1", hace que el modelo esté diseñado para una tarea de detección binaria, es decir, indicar si el objeto es o no una persona en este caso.

En segundo lugar, para el fichero de configuración del modelo 'SSD MobileNet v2 320x320' se han establecido los siguientes valores para los parámetros correspondientes:

- *fine_tune_checkpoint_type = Detection*

- *total_steps = 60000*

- *batch_size = 16*
- *matched_threshold = 0.5*
- *max_detections_per_class, max_total_detections & max_number_of_boxes = 2000*:
- *num_classes = 1*:

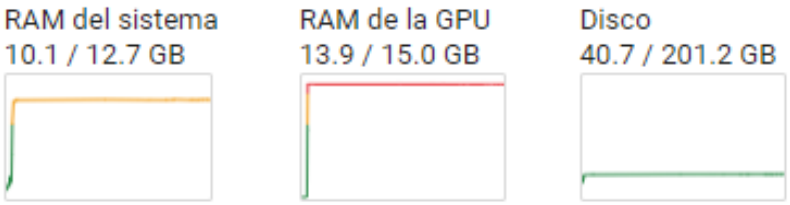

Figura 4.2: Límites en relación con la memoria.

4.2.2. Preparación y configuración del entrenamiento de YOLOv8

La configuración necesaria para el entrenamiento de YOLOv8 es menos compleja que la que se acaba de ver. El fomato YOLOv8 posee un fichero *data.yaml* de configuración programado en un lenguaje de serialización de datos. Este fichero contiene información que recoge el modelo durante el periodo de entrenamiento. Incluye las rutas de los directorios donde están las imágenes de entrenamiento, validación y prueba, además puede contener enlaces de descarga, las clases recogidas en el conjunto de datos, e incluso definiciones de los marcos que recuadran los objetos a detectar. Por lo tanto, este fichero debe estar correctamente configurado a la hora arrancar el entrenamiento.

4.2.2.1. Especificación del modelo a implementar

Como se ha explicado en el Capítulo 3, YOLOv8 posee varias arquitecturas dentro de esta versión. La arquitectura elegida para la evaluación de este modelo ha sido en concreto YOLOv8, ya que posee el mejor balance en cuanto a velocidad y precisión, lo que puede resultar ventajoso en una aplicación como es la del control de aforos mediante el uso de dispositivos de bajo consumo.

4.2.2.2. Configuración de los parámetros del entrenamiento

Cabe resaltar los parámetros modificables para obtener un entrenamiento exitoso en función del conjunto de datos que se utilice:

- *Pre-trained = Yes*: Este parámetro indica la ejecución del entrenamiento de un modelo sobre una arquitectura preentrenada con COCO.

- *Epochs = 50*: una época es un ciclo completo en el modelo recorre de principio a fin el conjunto de datos utilizado. Se utiliza este valor, ya que a aproximadamente, a partir de esta época, el modelo deja de aprender.
- *Image Size = 640*. Indica que el conjunto de datos que se le pasan al modelo tienen una resolución de 640x640 píxeles.
- *Batch Size = Auto:* fijado en *Auto*, de esta forma maximiza el tamaño de lote en función de la arquitectura utilizada.

4.3. Entrenamiento

En este apartado se explica el proceso de entrenamiento que se ha llevado a cabo en cada uno de los modelos que se han evaluado en este TFG.

El entrenamiento en redes neuronales implica refinar los ajustes internos de la red para que pueda desempeñar la tarea específica de reconocer objetos. Esto se logra exponiendo a la red a un conjunto de ejemplos de entrenamiento que contienen entradas y las salidas correctas asociadas. A medida que la red procesa estos ejemplos, ajusta gradualmente sus conexiones internas para reducir la discrepancia entre las salidas que genera y las salidas deseadas.

Los resultados ideales al finalizar cualquier entrenamiento son que cada uno de los parámetros de pérdida, que se definen en los siguientes puntos, finalicen en valores muy cercanos al cero. Los resultados serán más fiables cuando las pérdidas sean más pequeñas.

4.3.1. Entrenamiento de EfficientDet y SSD-MobileNet

En este bapartado se explica lo que ha sido el entrenamiento del modelo EfficientDet y SSD-MobileNet gracias a la herramienta TensorBoard (Tensorflow, 2024a). Esta es una herramienta incluida en TensorFlow, cuyo objetivo principal es proporcionar una interfaz gráfica para visualizar, monitorizar y analizar el proceso de entrenamiento de modelos de aprendizaje profundo.

En la Figura 4.3 se muestra la pérdida total del modelo, que es la suma de la pérdida de clasificación, la pérdida de localización y la pérdida de regularización. En estas gráficas se muestra cómo se reduce la pérdida a lo largo del entrenamiento.

- Pérdida de clasificación: indica la pérdida asociada con la clasificación de objetos, es decir, mide cómo de bien puede el modelo predecir las clases correctas de los objetos en una imagen.
- Pérdida de localización: mide cómo de preciso es el modelo al predecir las coordenadas de la ubicación de los objetos en una imagen.
- Pérdida de regularización: mide cuánto contribuyen los términos de regularización para evitar el sobreajuste.

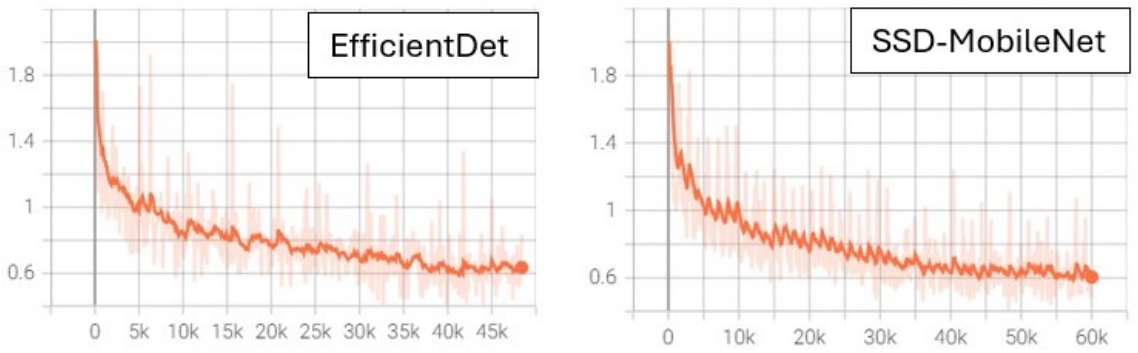

Figura 4.3: Pérdida total de EfficientDet y SSD-MobileNet.

La diferencia entre el valor inicial de 2.01 y el valor final de 0.63 para el entrenamiento de EfficientDet, y entre el valor inicial de 2.002 y el valor final de 0.6048 para el entrenamiento de SSD-MobileNet, en las gráficas de pérdida total, indica una disminución significativa en la pérdida de los modelos a lo largo de los entrenamientos. Es decir, la disminución en la pérdida total desde el primer paso hasta el último indica un progreso positivo, lo que sugiere que el modelo está aprendiendo y mejorando su capacidad para realizar la tarea de detección de objetos con mayor precisión a medida que avanza el proceso de entrenamiento.

4.3.2. Entrenamiento de YOLOv8

En este apartado se explica lo que ha sido el entrenamiento del modelo YOLOv8 gracias a la herramienta Ultralytics, que proporciona información y gráficas de las siguientes métricas:

- *Box Loss*: diferencia entre las personas detectadas y las que de verdad existen, o están etiquetadas.
- *Class Loss*: pérdida de localización, previamente explicada en el punto anterior.
- *Object Loss*: pérdida de los objetos detectados.

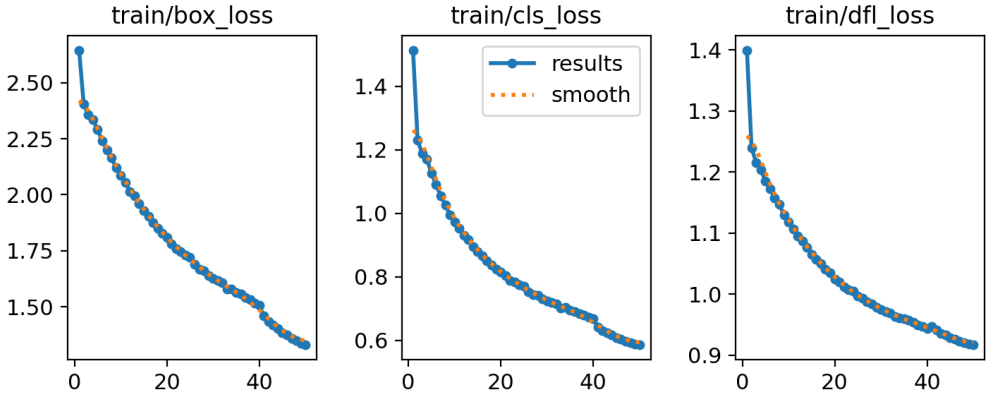

Figura 4.4: Box Loss, Class Loss y Object Loss.

La imagen de la Figura 4.4 se ha obtenido de la carpeta de resultados que genera Ultralytics después de un entrenamiento. Como se puede apreciar en la misma, los valores de pérdida han descendido mucho durante el entrenamiento, quedándose en la última época un box_loss de 1,33, un class_loss de 0,58 y un object_loss de 0,91.

Poco a poco, cuando se acercan a la última época, las funciones de pérdida se van convirtiendo en funciones constantes. Como se ha explicado antes, a partir de la época 51 estos valores se mantienen. Existen casos donde estas métricas pueden subir, síntoma de que el modelo está desaprendiendo. Es importante saber que estas métricas deben conseguir llegar a valores muy próximos al cero ya que de esta forma el modelo pierde menos información. Estas métricas dependen de la calidad del dataset, cuanto mayor calidad, se obtendrán valores más próximos al cero.

Capítulo 5

Evaluaciones y resultados

Este último capítulo describe los resultados obtenidos y una evaluación de las métricas de precisión y velocidad más relevantes tras el entrenamiento de los modelos utilizados en este TFG.

La métrica mAP, cuyas siglas significan *Mean Average Precision* (Tan, 2019), es una métrica que se utiliza para medir el rendimiento de los modelos que realizan tareas de recuperación de documentos o información y detección de objetos. Para entender su cálculo es necesario comprender la métrica IoU (Digifarm, 2024).

IoU (Intersección sobre Unión) es una métrica comúnmente utilizada para evaluar el rendimiento de algoritmos de detección de objetos. En concreto, se utiliza para evaluar la superposición entre la región predicha por el modelo y la región real del objeto en la imagen. Por ello, el IoU se define como la proporción del área de superposición entre las cajas delimitadoras predichas y las verdaderas delimitadoras al total de sus áreas de unión.

Por lo tanto, a la hora de realizar el cálculo de mAP de un modelo, se selecciona cada una de las imágenes del conjunto de datos de testeo y se evalúa cada una de las detecciones predichas que el modelo ha realizado a partir de la imagen. Si la detección en evaluación cuenta con una IoU por encima del valor fijado, se toma la detección como correcta, en caso contrario, se evalúa la detección incorrectamente. A continuación se presentan dos cálculos distintos de mAP a partir de IoU:

- *mAP 0.50*: se refiere a la forma tradicional de calcular mAP, es decir, para un umbral constante de IoU (0.5).
- *mAP 0.5:0.95*: que se refiere a el cálculo de mAP para diferentes umbrales de IoU en un rango de 0.5 a 0.95 y aumentando 0.5 su valor en cada paso, para finalmente realizar un promedio y obtener el mAP general del modelo.

A medida que aumenta el valor de IoU requerido en el cálculo de mAP, la evaluación de una detección es más exigente. Por lo tanto, como el puntaje de IoU varía de cero a uno, un puntaje de uno indica una superposición perfecta entre las

cajas delimitadoras predichas y verdaderas, y un puntaje de cero indica que no hay superposición en absoluto. En la Figura 5.1 (Pyimagesearch, 2024) se muestra el resultado de la evaluación de distintas detecciones según el IoU.

Por último, es importante medir la velocidad de cada uno de los modelos, ya que el objetivo de este TFG es detectar el aforo en tiempo real, y por tanto, obtener un modelo veloz a la hora de hacer predicciones es de vital importancia. La velocidad de un modelo se define como el tiempo que tarda en procesar una imagen en la Raspberry. Para calcular esta medida, se han procesado diferentes imágenes para realizar una media de todos los tiempos que tarda en procesar cada una de estas.

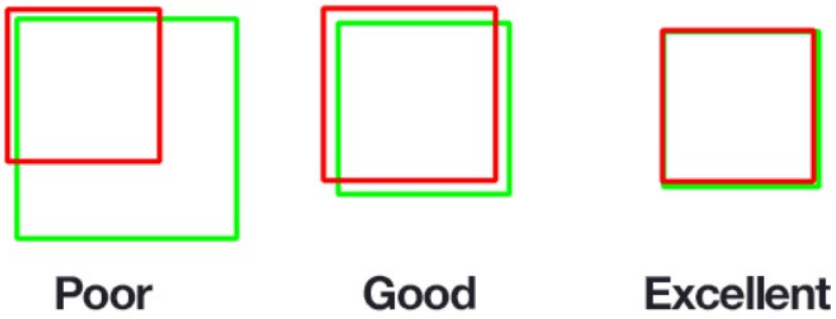

Figura 5.1: Ejemplo de la evaluación de IoU.

5.1. Evaluación y resultados de EfficientDet

En esta sección se explican los resultados conseguidos tras realizar las inferencias del modelo EfficientDet sobre un conjunto de datos de testeo en la Raspberry.

Como se observa en la Tabla 5.1, EfficientDet obtiene una precisión de 45,34 % mAP0.5 y 19,54 % mAP0.5:0.05:0.95. Estos valores se obtienen haciendo uso de la herramienta de cálculo de mAP (Cartucho, 2019).

Métrica	Valor
mAP0.5	45,34 %
mAP0.5:0.95	19,54 %

Tabla 5.1: Métricas de EfficientDet.

El hecho de que la precisión disminuya a medida que aumenta el umbral de IoU, indica que el modelo tiene dificultades para realizar predicciones precisas cuando se requiere una superposición más alta entre las detecciones y las personas reales. Es decir, EfficientDet, de manera general no es un buen modelo para realizar detecciones en situaciones donde las personas están extremadamente juntas.

Sin embargo, este modelo es más fiable para la detección de personas algo más dispersas. A continuación (veáse Figura 5.2), se muestra la inferencia de este modelo para dos situaciones distintas. En una imagen aparecen muchas personas juntas desde un ángulo frontal, y en la otra, aparecen personas más dispersas y desde un ángulo con mayor perspectiva aérea.

Figura 5.2: Inferencia EfficientDet.

Por último, en cuanto a la velocidad de EfficientDet, este modelo es capaz de procesar una imagen y realizar las detecciones en la Raspberry en un tiempo medio de 3,56 segundos.

5.2. Evaluación y resultados de SSD-MobileNet

En esta sección se explican los resultados conseguidos tras realizar la inferencia del modelo EfficientDet sobre un conjunto de datos de testeo en la Raspberry.

Como se observa en la Tabla 5.2, SSD-MobileNet obtiene una precisión de 27,01 % mAP0.5 y 13,52 % mAP0.5:0.95. Estos valores se obtienen haciendo uso de la herramienta de cálculo de mAP mencionada en la sección anterior.

Métrica	Valor
mAP0.5	27,01 %
mAP0.5:0.95	13,52 %

Tabla 5.2: Métricas de SSD-MobileNet.

En términos generales, SSD-MobileNet no resulta ser un modelo eficaz para detectar personas que no se encuentran lo suficientemente dispersas como para facilitar la detección al ojo humano y a este modelo de deep learning. A continuación (veáse Figura 5.3), se muestra la inferencia de este modelo para dos situaciones distintas.

En ambas imágenes, se puede observar que es incapaz de reconocer a muchas de las personas que aparecen.

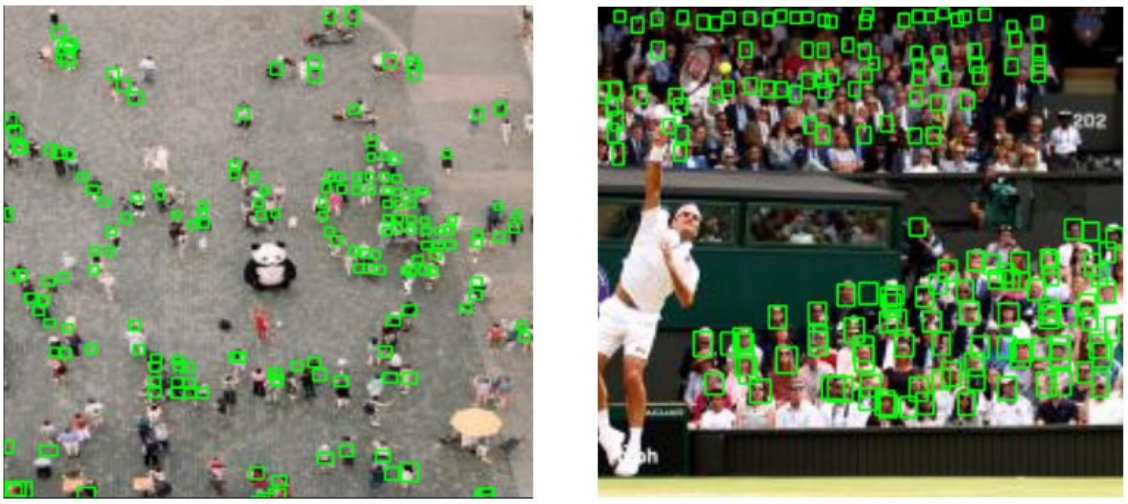

Figura 5.3: Inferencia SSD-MobileNet.

Por último, en cuanto a la velocidad de SSD-MobileNet, este modelo es capaz de procesar una imagen y realizar las detecciones en la Raspberry en un tiempo medio de 2,08 segundos.

5.3. Evaluación y resultados de YOLOv8

En esta sección se explican los resultados conseguidos tras realizar las inferencias del modelo YOLOv8 sobre un conjunto de datos de testeo en la Raspberry.

Al finalizar el entrenamiento de YOLOv8, una función de Ultralytics se encarga de almacenar todos los valores de las métricas de cada época de entrenamiento en un archivo llamado *results.csv*. Este fichero contiene tanto métricas del propio entrenamiento, como de la evolución del modelo. tal y como se observa en la Tabla 5.3, YOLOv8 obtiene una precisión de 60,568 % mAP0.5 y 33,481 % mAP0.5:0.95.

Métrica	Valor
mAP0.5	60,568 %
mAP0.5:0.95	33,481 %

Tabla 5.3: Métricas de YOLOv8.

En términos generales, YOLOv8 resulta ser un modelo sólido y eficaz para detectar personas en cualquier tipo de entorno. A continuación, (veáse Figura 5.4), se muestra la inferencia de este modelo.

En cuanto a la velocidad de YOLOv8, este modelo es capaz de procesar una imagen y realizar las detecciones en la Raspberry en un tiempo medio de 3,15 segundos

Figura 5.4: Inferencia de YOLOv8.

5.4. Comparación de los modelos

En esta última sección, se expone una comparación (veáse Tabla 5.4) de los modelos EfficientDet, SSD-MobileNet y YOLOv8. Se realiza una comparación técnica en la que se tienen en cuenta métricas de precisión y velocidad.

Modelo	mAP0.5	mAP0.5:0.95	Velocidad
EfficientDet	45,34 %	19,54 %	3,56s
SSD-Mobilenet	27,01 %	13,52 %	2,08s
YOLOv8	60,56 %	33,481 %	3,15s

Tabla 5.4: Comparación de precisión y velocidad.

En primer lugar, es posible que no se obtenga un valor mayor para la precisión en cada uno de los modelos, ya que estos han sido entrenados para ser capaces de detectar personas vistas desde todo tipo de ángulos, es decir, realiza detecciones sobre la cabeza o el cuerpo de las personas dependiendo de la situación. Es por eso, que si en una imagen aparece una persona en la que se refleja todo su cuerpo, y el modelo le detecta únicamente por su cabeza, la detección sería clasificada como incorrecta ya que no superaría la superposición mínima requerida. Este suceso daría lugar a una medida de precisión inferior a la que realmente corresponde.

Ademas, es importante señalar que el conjunto de datos utilizado para el entrenamiento de EfficientDet es más pequeño que el empleado para SSD-MobileNet y YOLOv8. Aunque la mayoría de las imágenes de este conjunto también están incluidas en el último, la precisión de EfficientDet puede resultar engañosa. Esto se debe a que el entrenamiento con un conjunto de datos más reducido podría haber favorecido su evaluación en la detección de ciertas imágenes. Sin embargo, EfficientDet no está diseñado para alcanzar los niveles de precisión que ha logrado con un conjunto de datos más amplio y diverso que incluya una mayor variedad de situaciones.

No obstante, en términos de precisión, YOLOv8 es el modelo que obtiene los mejores resultados con diferencia. Si bien este resultado no lo define como un mode-

lo extremadamente preciso, sí podemos reconocer su capacidad para ser útil en una gran variedad de situaciones sin ser excesivamente complejas.

Por otro lado, en cuanto a velocidad, el modelo SSD-MobileNet se destaca significativamente. Esto se debe a que MobileNet es una arquitectura de red neuronal ligera y eficiente, diseñada específicamente para ser utilizada en dispositivos con recursos limitados y en aplicaciones donde la rapidez es crucial. La simplicidad de MobileNet permite una rápida inferencia. Sin embargo, su eficiencia sacrifica en gran medida la precisión, lo que la hace una solución poco balanceada para escenarios que requieren tanto rapidez como razonable exactitud en la detección y clasificación de objetos.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

A lo largo de este TFG, se ha desarrollado un sistema eficiente para la gestión y el control de aforos en eventos, destinado a dispositivos de bajo consumo como la Raspberry Pi. Los resultados obtenidos demuestran que el sistema es eficaz en determinadas situaciones, proporcionando una solución viable para el conteo de personas en tiempo real y en entradas de datos mediante un modelo de detección de objetos.

El sistema desarrollado incluye una interfaz gráfica intuitiva, que permite al usuario seleccionar el modelo de detección de objetos adecuado para realizar la inferencia en un dispositivo de bajo consumo. Esta funcionalidad es crucial para adaptar el sistema a diferentes escenarios y requisitos, garantizando la flexibilidad y usabilidad necesarias en diferentes aplicaciones prácticas.

En términos de precisión y velocidad, los resultados obtenidos son satisfactorios. La detección es precisa en datos de entrada de buena calidad y en situaciones donde la cantidad de personas no es extremadamente grande. Esto se debe a la elección de modelos de detección que equilibran precisión y velocidad de procesamiento, permitiendo una rápida inferencia de los datos de entrada y salida. En particular, se ha identificado que el modelo YOLOv8 es el más adecuado para futuras investigaciones debido a su alta precisión y eficiencia en el procesamiento de datos en comparación con los otros modelos.

Además, es importante reconocer que, durante el desarrollo del sistema, se han apreciado ciertas limitaciones relacionadas con la escasez de recursos y la calidad variable de los mismos. Estas restricciones han impedido en muchas ocasiones la ejecución de entrenamientos largos y completamente libres de errores, afectando potencialmente la eficacia del sistema. A pesar de estos desafíos, el sistema ha demostrado ser funcional y efectivo dentro de los parámetros establecidos.

Otro aspecto a considerar es la calidad y etiquetado de los datasets utilizados para el entrenamiento del modelo. Los datasets no fueron construidos desde

cero, lo que implica una dependencia en la precisión y exhaustividad del etiquetado realizado por terceros. La calidad de los datos de entrada es fundamental para garantizar la precisión de la detección, y cualquier deficiencia en este aspecto puede impactar negativamente en los resultados obtenidos.

En conclusión, el sistema desarrollado ofrece una solución prometedora para la gestión y control de aforos en eventos, utilizando dispositivos de bajo consumo. Aunque existen áreas de mejora, especialmente en relación con los recursos disponibles, la calidad de los datasets y la precisión con la que detectan los modelo. Aún así, los resultados obtenidos son alentadores y establecen una base sólida para futuras investigaciones y desarrollos.

6.2. Trabajo a futuro

A pesar de los resultados prometedores obtenidos en este TFG, existen múltiples áreas que pueden ser exploradas y optimizadas en futuros trabajos para mejorar aún más la eficacia y eficiencia del sistema de gestión y control de aforos desarrollado. A continuación se detallan algunas posibles mejoras que pueden implementarse en un trabajo a futuro:

- Entrenamiento con datasets personalizados: una de las principales áreas de mejora es la creación y utilización de datasets personalizados. Desarrollar datasets desde cero asegurará que los datos de entrada estén perfectamente etiquetados y sean de alta calidad. Esto podría mejorar significativamente la precisión del modelo de detección.

- Exploración de nuevos modelos: aunque el modelo YOLOv8 ha demostrado ser adecuado, explorar otros modelos de detección de objetos que puedan ofrecer un mejor equilibrio entre precisión y velocidad también es buen camino de cara a unos resultados mejores. Modelos con dos capas de detección en lugar de una, podrían mejorar considerablemente la precisión a la hora de realizar detecciones, eso sí, sacrificando siempre algo de velocidad. También sería buena idea realizar una investigación y evaluación de YOLOv9, la nueva versión de YOLO. La razón por la que YOLOv9 no ha sido evaluado en este TFG es porque su lanzamiento fue después del comienzo de la evaluación de YOLOv8.

- Modificación de la interfaz gráfica: mejorar este aspecto puede aportar al sistema ser más interactivo de cara al usuario.

- Extrapolación a diversas industrias: considerando la aceptación positiva del sistema en escenarios de prueba, es factible plantear su aplicación en diversos tipos de industria. Por ejemplo, el sistema podría ser útil en el sector *retail* (es el sector de la venta al detalle o minorista) para gestionar aforos en tiendas y centros comerciales, en el ámbito de la salud para controlar la cantidad de personas en clínicas y hospitales, y en el sector de transporte para monitorizar la ocupación de estaciones y terminales.

Introduction

This first chapter presents an introduction to this Final Degree Project (TFG), which addresses the implementation of a solution for controlling capacity at events through the use of artificial intelligence (AI) and its execution in low-consumption systems.

Currently, the efficient management and control of capacity at events has become a growing concern, not only to guarantee the safety and comfort of attendees, but also to comply with established regulations and optimize the use of available resources.

When it comes to counting the number of people present in a specific place, multiple methods can be used, as rudimentary as counting with your fingers or by using analogic counters widely used by security personnel. which is part of all types of events.

In recent decades, automated models have been developed that are capable of controlling the capacity of people and other individuals in spaces with a closed perimeter where one or more entry and exit points are established. For example, the Madrid Metro limits entrances and exits by placing turnstiles capable of adding or subtracting one unit from the total capacity each time they are used. Also in many establishments, infrared sensors are implemented that detect entry or exit through movement of a local individual.

The problem arises when you want to control the capacity of an event held in a large space, open to the public and where there is no control of the number of entrances and exits (see Figure 1.1 (TheObjective, 2018)). To keep track of the capacity at this type of event, more advanced methods have been developed, such as radio frequency tracking systems, smartphone geolocation applications or artificial intelligence models capable of detecting and counting all the people located in the area where the event is held. event. The implementation of this last resource is the objective that has been proposed to achieve during the development of this TFG.

Figura 6.1: Crowd of people in an uncontrolled space.

Motivation

The TFG in which we have been involved during this course has been a truly enriching experience that has led us to a significant expansion of our knowledge both academically and professionally. Although we started with little prior knowledge, the challenge of entering the fascinating world of artificial intelligence and its application in the management and control of capacity through its possible integration into low-consumption devices has completely captivated us.

During this work, we have had the opportunity to immerse ourselves in a fascinating and novel technological universe, exploring the complexities and possibilities of AI in the control and management of events. This immersion has not only provided us with a deep theoretical understanding, but has also allowed us to develop practical skills in the implementation and operation of this technology.

Our commitment to this TFG goes beyond mere academic research. We are driven by a solid belief in the transforming potential of technology when used responsibly. By focusing our efforts on developing an effective tool for managing and controlling capacity at events, we want to contribute strongly to the safety and well-being of society. Therefore, we envision this work as a step forward in creating innovative solutions that address contemporary challenges, and we are excited by the positive impact our work can have in the community and beyond.

Objectives

The main objective of this TFG is the evaluation of pre-existing artificial intelligence models capable of recognizing and counting the people present in an input file, which can be an image, recording or live transmission, processed by a low-power device.

Firstly, to achieve the main objective, the application of AI techniques is necessary, specifically the use of convolutional neural networks, as they offer a promising way to address this challenge effectively. Models such as SSD (University, 2020b) and YOLO (YOLO, 2023) have demonstrated their ability to perform object detection and tracking tasks in real time, making them ideal tools for monitoring and controlling the flow of people. in event environments.

By integrating these technologies into the development of our solution, we seek to not only automate the attendee counting process, but also obtain real-time information on the density of people in different areas of the event.

Furthermore, the design of the system has been oriented so that it can be implemented on a low-cost and low-power board, which allows its integration into low-power devices that operate in real time. Within this design, a graphical interface has been implemented from which you can access this device and test the trained models in an easier and more intuitive way.

On the other hand, it is essential to execute correct training with a specific data set so that the model's capacity for detecting and counting people is not limited by the quality of the input, its perspective, the size of the subject, etc. .

Work planning

During the entire development of this TFG, the tasks have been meticulously well established, thus helping to divide the workload and responsibilities between both members of the team. In Figure 1.2 we present a Gantt diagram indicative of the planning of this TFG.

Initially, we are based on the TFG carried out by Alejandro-Daniel Díaz Guzmán during the 2022-2023 academic year. His work, focused on *UAV Autonomous Flight Simulation for search and rescue missions*, served as a starting point to place ourselves in the appropriate context, structure our work and begin our investigations.

In the first few months that followed, we dedicated ourselves to searching and researching free drone simulators that were compatible with our equipment, as well as capable of loading multimedia content for further processing. Although this search turned out to be unsuccessful, this experience marked an evolution in the approach of this TFG, leaving the integration of a drone into the system in the background.

Afterwards, we quickly focus on the most substantial content: the search, training and evaluation of convolutional neural network models specialized in object detection.

Finally, this TFG has been focused on the evaluation and comparison of the results obtained by the selected models and the completion of the working memory.

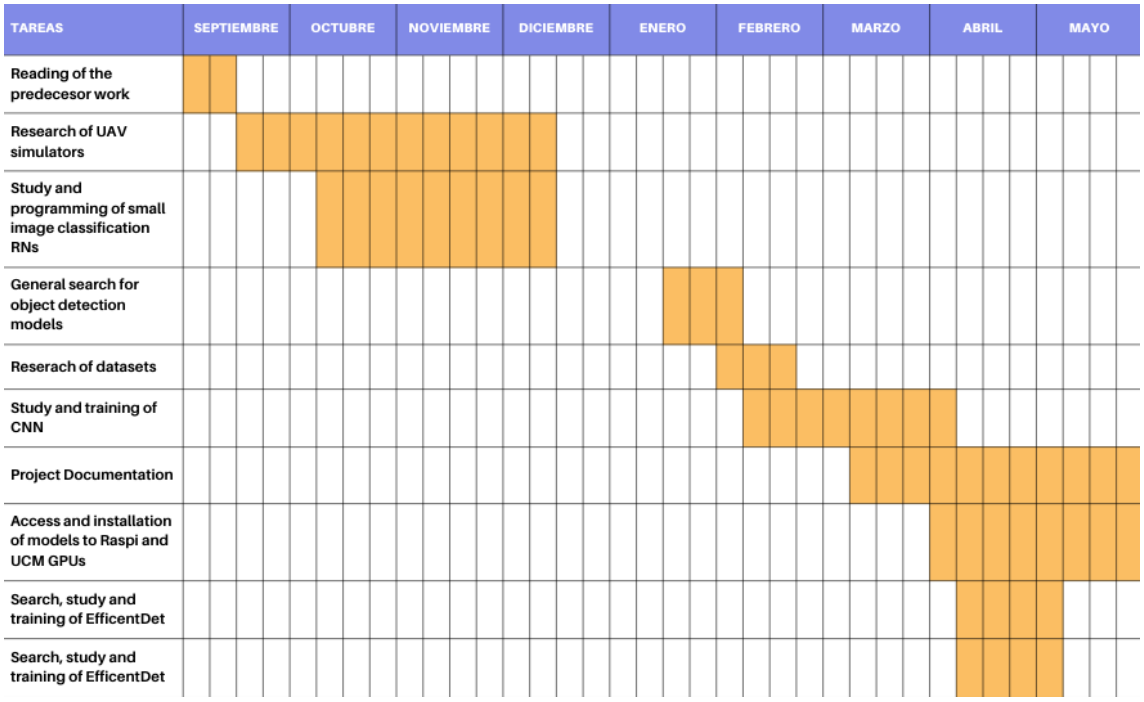

Figura 6.2: Gantt diagram representing TFG planification.

Document structure

This document is organized into 6 chapters. All of them are listed below, accompanied by a brief summary:

- **Chapter 1: Introduction**. This chapter is the one the reader is in at this very moment. This chapter introduces the TFG and talks about the motivations, objectives, work plan and structure of the document.

- **Chapter 2: State of the Art**. This chapter introduces the fundamental concepts for understanding this TFG. It talks about Deep Learning and the functioning and types of neural networks that are currently being worked with. In addition, it goes into detail about the great paradigm of artificial intelligence and the applications it may have.

- **Chapter 3: System architecture**. This chapter describes the architecture of the system that you want to implement, it deals with both the hardware part and the software part.
- **Chapter 4: Training processes**.This chapter describes the training process of a pre-existing model, from the preparation of the data set to the preparation of the training and its configuration.
- **Chapter 5: Evaluation and results**.This chapter describes the results obtained with each of the prepared models. In addition, it presents a brief analysis of them making different technical comparisons.
- **Chapter 6: Conclusions and Future Work**. This chapter includes the conclusions obtained once the work was completed and proposes a series of future contributions.

Conclusions and future Work

Conclusions

Throughout this TFG, an efficient system has been developed for the management and control of capacity at events, aimed at low-consumption devices such as the Raspberry Pi. The results obtained demonstrate that the system is effective in certain situations, providing a viable solution for counting people in real time and in data inputs using an object detection model.

The developed system includes an intuitive graphical interface, which allows the user to select the appropriate object detection model to perform inference on a low-power device. This functionality is crucial to adapt the system to different scenarios and requirements, guaranteeing the flexibility and usability necessary in different practical applications.

In terms of precision and speed, the results obtained are satisfactory. Detection is accurate in good quality input data and in situations where the number of people is not extremely large. This is due to the choice of detection models that balance accuracy and processing speed, allowing rapid inference of input and output data. In particular, the YOLOv8 model has been identified as the most suitable for future research due to its high accuracy and efficiency in data processing compared to the other models.

Furthermore, it is important to recognize that, during the development of the system, certain limitations related to the scarcity of resources and their variable quality have been appreciated. These restrictions have often prevented the execution of long, completely error-free training sessions, potentially affecting the effectiveness of the system. Despite these challenges, the system has proven to be functional and effective within the established parameters.

Another aspect to consider is the quality and labeling of the datasets used for model training. The datasets were not built from scratch, which implies a dependence on the accuracy and completeness of labeling carried out by third parties. The quality of the input data is essential to guarantee the accuracy of the detection,

and any deficiency in this aspect can negatively impact the results obtained.

In conclusion, the developed system offers a promising solution for the management and control of capacity at events, using low consumption devices. Although there are areas for improvement, especially in relation to the available resources, the quality of the datasets and the precision with which the models detect. Still, the results obtained are encouraging and establish a solid foundation for future research and development.

Future Work

Despite the promising results obtained in this TFG, there are multiple areas that can be explored and optimized in future work to further improve the effectiveness and efficiency of the capacity management and control system developed. Below there are some possible improvements that can be implemented in future work:

- Training with custom datasets: one of the main areas of improvement is the creation and use of custom datasets. Developing datasets from scratch will ensure that the input data is perfectly labeled and of high quality. This could significantly improve the accuracy of the detection model.
- Exploring new models: Although the YOLOv8 model has proven to be suitable, exploring other object detection models that can offer a better balance between accuracy and speed is also a good way to achieve better results. Models with two detection layers instead of one could considerably improve the precision when making detections, always sacrificing some speed. It would also be a good idea to do some research and evaluation of YOLOv9, the new version of Yolo. The reason why YOLOv9 was not evaluated is because its release was after the beginning of the YOLOv8 evaluation.
- Modification of the graphical interface: improving this aspect can make the system more interactive for the user.
- Extrapolation to various industries: considering the positive acceptance of the system in test scenarios, it is feasible to propose its application in various types of industries. For example, the system could be useful in the retail sector to manage capacity in stores and shopping centers, in the health field to control the number of people in clinics and hospitals, and in the transportation sector to monitor the occupancy of stations. and terminals.

Contribuciones Personales

Este TFG ha sido realizado en su mayoría de forma conjunta, debido a que se ha colaborado estrechamente desde el inicio del trabajo, especialmente en áreas desconocidas para ambos. Y es que, ninguno de los dos autores tenía conocimientos previos en inteligencia artificial, lo que hizo que el aprendizaje y la investigación inicial fueran procesos colaborativos.

Con el tiempo, y a medida que se adquirieron más conocimientos, los miembros del equipo pudieron dividir las tareas de manera efectiva para manejar la gran carga de trabajo. A continuación se detallan las contribuciones específicas de cada autor:

Javier Gómez Arribas

- Búsqueda y preparación del dataset utilizado en Yolov8: identificó y seleccionó el dataset Visdrone, adaptándolo específicamente para la detección de personas.

- Entrenamiento y evaluación del modelo Yolov8: realizó múltiples entrenamientos, pruebas y configuraciones del modelo YOLOv8. Además, evaluó los resultados obtenidos en la Raspberry Pi y ajustó los parámetros según fuese necesario, asegurando así que el sistema fuera funcional en dispositivos de bajo consumo.

- Desarrollo de la interfaz: desarrolló la interfaz que forma parte del sistema y lo hace más atractivo e intuitivo para el usuario.

Pablo Sánchez-Rodilla Serrano

- Búsqueda y preparación del dataset utilizado en EfficientDet y SSD-MobileNet: investigó el funcionamiento de Roboflow, identificó y seleccionó diferentes conjuntos de datos que posteriormente fueron unificados y adaptados para la detección específica de personas.

- Entrenamiento y evaluación de los modelos EfficientDet y SSD-MobileNet: realizó múltiples entrenamientos, pruebas y configuraciones de ambos. También evaluó los resultados obtenidos en la Raspberry Pi y ajustó los parámetros de configuración del modelo y del entrenamiento según fuese necesario para obtener mejores resultados. Además, investigó y probó otros modelos, como distintas versiones de EfficientDet o Faster-RCNN, que quedaron en el camino.

En conjunto, ambos autores han logrado avanzar significativamente en el TFG, superando la falta de conocimientos inicial y distribuyendo las tareas de manera eficiente para manejar la carga de trabajo. Sus contribuciones han sido esenciales para el desarrollo y éxito del sistema de gestión y control de aforos presentado.

Bibliografía

Alexis. Red neuronal recurrente. *https://alexis96.github.io/proyecto-RNN/* , 2020.

BBC. Amazon go. *https://www.bbc.com/mundo/noticias-42774861* , 2018.

de Cachemira, F. M. Fundación mundial de cachemira. *https://www.kashmirworldfoundation.org/* , 2008.

Cartucho. map (mean average precision). *https://github.com/Cartucho/mAP*, 2019.

DataScientest. Convolutional neural network : definición y funcionamiento. *https://datascientest.com/es/convolutional-neural-network-es*, 2023.

coco dataset. Coco. *https://cocodataset.org/home*, 2017.

Digifarm. Intersección sobre la unión. *https://digifarm.io/es/blog/intersection-over-union*, 2024.

Facewatch. Facewatch. *https://www.facewatch.co.uk/* , 2024.

Google. Google colab pro. *https://colab.research.google.com/signup*, 2023.

Google. Diferencias entre deep learning, machine learning e inteligencia artificial. *https://cloud.google.com/discover/deep-learning-vs-machine-learning: :text=Essentially*

Google. *Google brain.* https://research.google.com/teams/brain/?authuser=2*, 2024b.*

Google. *Google colaboratory.* https://research.google.com/colaboratory/intl/es/faq.html*, 2024c.*

Google. *Plataformas de gpu.* https://cloud.google.com/compute/docs/gpus?hl=es-419*, 2024d.*

IAAR. *introducción al deep learning.* https://iaarbook.github.io/deeplearning/*, 2020.*

IBM. *Edge computing.* https://www.ibm.com/es-es/topics/edge-computing*, 2023.*

IBM. *Redes neuronales.* https://www.ibm.com/es-es/topics/neural-networks, *2024a.*

IBM. *Regresión lineal.* https://www.ibm.com/es-es/topics/linear-regression, *2024b.*

IBM. *¿qué es el descenso de gradiente?* https://www.ibm.com/mx-es/topics/gradient-descent, *2024c.*

Jacar. *La función unidad rectificada uniforme (relu): Una herramienta esencial para el aprendizaje profundo.* https://jacar.es/la-funcion-unidad-rectificada-uniforme-relu-una-herramienta-esencial-para-el-aprendizaje-profundo/: :text=La

Jupyter. Jupyter notebook. *https://jupyter.org/*, 2024.

Küppers, H. Küppers, teoría del color. *https://htmlcolorcodes.com/es/*, 2008.

Larranaga, P. Redes neuronales. *http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf*, 2019.

Lens, G. Google lens. *https://lens.google/intl/es/*, 2024.

MathWorks. ¿qué es la detección de objetos? *https://es.mathworks.com/discovery/object-detection.html*, 2018.

Mathworks. ¿qué son las redes neuronales convolucionales? *https://es.mathworks.com/discovery/convolutional-neural-network.html*, 2024.

Microsoft. Preparación de datos para el aprendizaje distribuido. *https://learn.microsoft.com/es-es/azure/databricks/machine-learning/load-data/ddl-data*, 2024.

Nguyen, H. T. Clasificación de imágenes de ultrasonido mamario utilizando efficientnetv2 y arquitecturas de redes neuronales. *https://www.researchgate.net/publication/363020277_Breast_Ultrasound_Image_Classification_Using_EfficientNetV2_and_Shallow_Neural_Network_Architectures*, 2024.

Ossendryver, N. Latest sightings. *https://latestsightings.com/*, 2011.

Pyimagesearch. ntersección sobre unión (iou) para detección de objetos. *https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/*, 2024.

Pytorch. Pytorch. *https://pytorch.org/*, 2024.

Roboflow. Todo lo que necesita para construir e implementar modelos de visión por computadora. *https://roboflow.com/*, 2024.

Sánchez-Rodilla, P. Repositorio de github del proyecto. *https://github.com/PabloS-RS/Control_aforos_TFG*, 2024.

Tan, R. J. Desglose de la precisión media media (map). *https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52*, 2019.

Team, T. O. D. Tensorflow 2 detection model zoo. *https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md*, 2021.

Tensorflow. Pascal visual object classes challenge. *https://www.tensorflow.org/datasets/catalog/voc*, 2022.

Tensorflow. Tensorboard: el kit de herramientas de visualización de tensorflow. *https://www.tensorflow.org/tensorboard?hl=es-419*, 2024a.

Tensorflow. Tensorflow. *https://www.tensorflow.org/?hl=es-419*, 2024b.

Tensorflow. Tensorflow lite. *https://www.tensorflow.org/lite/api_docs?hl=es-419*, 2024c.

Tesla. Piloto automático y capacidad de conducción autónoma total. *https://www.tesla.com/es_es/support/autopilot*, 2024.

TheObjective. Aglomeración de personas en una playa. *https://theobjective.com/further/cultura/2018-06-08/playas-peligrosas-solitarias/*, 2018.

Ultralytics. Entrena modelos de ia en segundos con ultralytics yolo. *https://www.ultralytics.com/es*, 2022.

Ultralytics. Ultralytics yolov8 tasks. *https://docs.ultralytics.com/tasks/*, 2024.

University, C. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *https://arxiv.org/abs/1704.04861*, 2017.

University, C. Efficientdet: Detección de objetos escalable y eficiente. *https://arxiv.org/abs/1911.09070*, 2020a.

University, C. Ssd: Single shot multibox detector. *https://arxiv.org/abs/1512.02325*, 2020b.

UPM. Redes neuronales convoluciones. *https://dcain.etsin.upm.es/ carlos/-bookAA/ 05.7_RRNN_Convoluciones_CIFAR_10_INFORMATIVO.htmlcapa-de-agrupacion-pooling*, 2021.

Vodafone. Detección y optimización del tráfico mediante modelos de ia. *https://www.saladeprensa.vodafone.es/c/sp/statics/Informe_IA_Impacto_en_el_modelo_de_negocio/page=39zoom=100,0,0*, 2022.

Vries, R. Uso de la ia para monitorear la densidad de plástico en el océano. *https://www.bbc.com/mundo/noticias-42774861*, 2022.

YOLO. Ultralytics yolov8 docs. *https://docs.ultralytics.com/es*, 2023.

YOLOv8. Arquitecturas de yolov8. *https://docs.ultralytics.com/es/models/yolov8/performance-metrics*, 2023.