UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Informática de Sistemas y Computadores

Recuento de personas mediante cámaras de vídeo y ML en el edge

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Computadores y Redes

AUTOR/A: Calandín Vega, Andrés

Tutor/a: Manzoni, Pietro

Cotutor/a: Cecilia Canales, José María

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Informática de Sistemas y Computadores

Recuento de personas mediante cámaras de vídeo y ML en el edge

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Computadores y Redes

AUTOR/A: Calandín Vega, Andrés

Tutor/a: Manzoni, Pietro

Cotutor/a: Cecilia Canales, José María

CURSO ACADÉMICO:

Resum

En els últims anys, l'ús de dispositius intel·ligents ha augmentat dràsticament. Fins ara l'anàlisi de les dades recaptades els dispositius es realitzava en un servidor en el núvol amb alta disponibilitat i capacitat. Aquest creixement ha ocasionat que el trànsit de dades cap a aquests servidors se sature, generant retards i pèrdues, i per tant, perjudicant les aplicacions que requerisquen de resposta en temps real. La solució a aquesta problemàtica s'ha donat en forma del Edge Computing i el TinyML, que proposen que les dades siguen analitzades en el propi dispositiu, normalment restringit en capacitat.

En aquest marc, l'àrea del Machine Learning més problemàtica és la de l'anàlisi d'imatges, donada l'enorme quantitat d'operacions i grandària necessàries de les xarxes neuronals dedicades a aquesta tasca. El present treball empra anàlisi d'imatges amb *TinyML en una cambra com dispositiu restringit, amb l'objectiu d'identificar les persones que transiten per davant de la cambra i identificar la seua adreça per a definir les persones sortints o entrants a un espai concret. Per a aconseguir això s'empren modificacions de xarxes neuronals convolucionals com YOLO en ordre de complir els requisits de memòria i emmagatzematge del dispositiu sensor.

Paraules clau: TinyML, IoT Análisi d'images, Segmentació d'Images, Detecció de persones

Resumen

En los últimos años, el empleo de dispositivos inteligentes ha aumentado drásticamente. Hasta ahora el análisis de los datos recabados los dispositivos se realizaba en un servidor en la nube con alta disponibilidad y capacidad. Este crecimiento ha ocasionado que el tráfico de datos hacia estos servidores se sature, generando retrasos y pérdidas, y por tanto, perjudicando a las aplicaciones que requieran de respuesta en tiempo real. La solución a esta problemática se ha dado en forma del Edge Computing y el TinyML, que proponen que los datos sean analizados en el propio dispositivo, normalmente restringido en capacidad.

En este marco, el área del Machine Learning más problemática es la del análisis de imágenes, dada la enorme cantidad de operaciones y tamaño necesarias de las redes neuronales dedicadas a esta tarea. El presente trabajo emplea análisis de imágenes con TinyML en una cámara cómo dispositivo restringido, con el objetivo de identificar las personas que transiten por delante de la cámara e identificar su dirección para definir las personas salientes o entrantes a un espacio concreto. Para lograr esto se emplean modificaciones de redes neuronales convolucionales como YOLO en orden de cumplir los requisitos de memoria y almacenamiento del dispositivo sensor.

Palabras clave: TinyML, IoT, Análisis de imágenes, Segmentación de imagenes, Detección de personas

Abstract

In recent years, the use of smart devices has increased dramatically. Until now, the analysis of the data collected by the devices was performed on a cloud server with high availability and capacity. This growth has caused data traffic to these servers to become saturated, generating delays and losses, and therefore harming applications that require real-time response. The solution to this problem has come in the form of Edge Computing and TinyML, which propose that data be analyzed on the device itself, normally restricted in capacity.

In this framework, the most problematic area of Machine Learning is that of image analysis, given the enormous number of operations and size required of the neural networks dedicated to this task. The present work uses image analysis with TinyML on a camera as a constrained device, with the objective of identifying people passing in front of the camera and identifying their direction in order to define people leaving or entering a specific space. To achieve this, modifications of convolutional neural networks such as YOLO are used in order to meet the memory and storage requirements of the sensor device.

Key words: TinyML, IoT, Image analysis, Image Segmentation, Person detection

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este trabajo se empleará un dispositivo con una cámara incorporada, por lo que se asemeja a una cámara de vigilancia, aunque el objetivo de la misma difiere de la vigilancia clásica. Las cámaras de video fijas se han usado históricamente como cámaras de seguridad o de vigilancia. Gracias a las mejoras técnicas estos dispositivos han obtenido cada vez mejores resoluciones y latencia, además de conexión a Internet y otras funcionalidades. A parte de estas mejoras, en los últimos años, con el auge de los dispositivos inteligentes (o *Smart Devices*) y el Internet de las cosas (*IoT*, por sus siglas del inglés) se añaden aun más funcionalidades relacionadas, como la implementación del análisis del vídeo captado mediante el uso de redes neuronales. Este análisis se daba en un principio en el Cloud (o Nube), pero esto generó ciertos problemas al funcionamiento del IoT que la comunidad solucionó mediante la transición al *Edge Computing*, el cuál aboga por el análisis de los datos en el propio dispositivo para evitar, entre otros problemas, grandes anchos de banda.

El Internet de las cosas (IoT, por sus siglas en inglés, Internet of Things) es una tecnología que ha revolucionado la forma en que interactuamos con el mundo que nos rodea. *IoT* son las siglas en inglés de Internet de las Cosas y se refiere a la interconexión de dispositivos físicos a través de internet, permitiéndoles recopilar y compartir datos entre sí y con sistemas centrales en la nube. Estos dispositivos pueden ser prácticamente cualquier cosa: desde electrodomésticos y sensores industriales hasta vehículos y wearables, los cuales son dispositivos implementados en accesorios o ropa, como un reloj inteligente o una chaqueta que recopila datos sobre sudoración y regula la temperatura mediante refrigreación incorporada. La idea principal detrás de *IoT* es mejorar la eficiencia, la comodidad y la toma de decisiones al proporcionar acceso a datos en tiempo real y automatizar tareas cotidianas.

La noción de conectar objetos físicos a través de una red no es nueva. Los precursores de IoT se remontan a la década de 1970, cuando se desarrollaron las primeras máquinas expendedoras conectadas a la web y los sistemas de monitoreo de energía eléctrica. Sin embargo, el término 'Internet de las cosas' comenzó a ganar popularidad en la década de 1990 gracias a la visión de investigadores y empresas.

Durante la década de los 90 y 2000, la tecnología de IoT estaba en sus primeras etapas de desarrollo. Los investigadores y las empresas exploraban conceptos y creaban aplicaciones limitadas. Un hito importante fue la invención del protocolo IPv6, que proporcionó un número casi ilimitado de direcciones IP, lo que era esencial para conectar una gran cantidad de dispositivos a Internet. Esto permitió no solo que se conectaran los terminales usuales a Internet, como servidores y ordenadores, sino cada vez más dispositivos que no se relacionarían con Internet de forma normal, como termómetros, bombillas, neveras, etc.

Continuando con la década de 2010, esta marcó un punto de inflexión para el *IoT*, ya que se dieron avances significativos en hardware y conectividad. La miniaturización de componentes electrónicos permitió la creación de dispositivos más pequeños y económicos. La aparición de la tecnología 4G LTE y la posterior adopción de 5G mejoraron la conectividad, lo que permitió la transferencia de datos más rápida y confiable, sumandose esta tecnologías a las ya existentes, como Bluetooth o WiFi. A medida que avanzamos actualmente por la década de los 2020, y mirando al futuro, *IoT* continúa evolucionando a un ritmo acelerado. La integración de inteligencia artificial y aprendizaje automático en dispositivos *IoT* está impulsando la automatización y la toma de decisiones más sofisticadas. La seguridad cibernética también se ha convertido en un área crítica de enfoque a medida que la cantidad de dispositivos conectados aumenta y se enfrenta a riesgos potenciales.

Una de las características de una arquitectura IoT es el hecho de que los dispositivos que entran en ella son totalmente heterogéneos. Una red en IoT se encuentra formada por sensores y elementos muy diversos. En ella pueden encontrarse desde dispositivos domésticos que empleamos en nuestro día a día; como pueden ser electrodomésticos, cámaras de vigilancia, termómetros u otros sensores. La expansión del uso de IoT a diversos campos como la la industria, turismo, sanidad, educación, etc., ha generado que cada vez más dispositivos tengan conexión a Internet. Por tanto, a la hora de construir una red dentro de esta arquitectura estos nodos presentan diferencias notables en consumo de energía, modos de actividad, conectividad o localización. Dependiendo del uso o área en la que se encuentren los dispositivos y su red se puede clasificar en categorías como las siguientes.

Dispositivos domésticos inteligentes

Estos dispositivos entrarían dentro del campo de la *domótica* [2], pudiendo ser o no inteligentes, estando conectados entre sí. Los casos más comunes son los asistentes de voz, que se encuentran en muchos horages, como lo es Amazon Alexa; termostatos o bombillas inteligentes, o incluso persianas, robots de limpieza o incluso cerraduras inteligentes. Muchos de estos dispositivos pueden ser controlados mediante una aplicación móvil que el inquilino tenga en su *smartphone*, además de las funciones inteligentes que puedan tener. En el caso de una bombilla o termostato inteligentes, estos suelen modificar la temperatura o intensidad de la luz dependiendo de datos anteriores o de las preferencias del usuario. Un ejemplo de esta mecánica sería, en el caso de una bombilla inteligente, mantener una iluminación baja si se detecta suficiente luz en la habitación o administrar el consumo para ser más eficiente. En el caso de un termostáto, puede darse el caso de que el usuario haya desigando una temperatura ideal para ciertas habitaciones o el entorno global del hogar, a su vez, puede identificar las horas a las que se enciende o apaga y realizar esta acción de forma autónoma. Refrigerando el hogar antes de que el inquilino vuelva de trabajar en los meses de verano, por ejemplo.

Uno de los usos que se ha pensado para la domótica es el empleo de la misma en hogares con personas mayores que vivan solos y sean semidependientes [3]. En estos hogares se deplegarían ciertos sensores y dispositivos, como actuadores, que se enfoquen en facilitar la vida diaria de nuestros mayores. Los sensores recopilarían datos que variarían desde las condiciones mediambientales como la temperatura, humedad o presencia de partículas; hasta detectores de presencia, lectores biométricos o micrófonos. Estos sensores serían controlados por microcontroladores a lo largo del hogar que proporcionarían una forma de visualizar y sacar información de los datos obtenidos. En base a estos datos, de forma automatizada o manual, los actuadores desplegados en el hogar realizarían modificaciones enfocadas en la mejora de la vida del mayor.

Smart Health

En los últimos años, han aumentado de forma significativa las soluciones IoT enfocadas al ámbito sanitario y clínico. Muchas de estas soluciones son dispositivos categorizados como *wearables*, que son aquellos que pueden *vestirse* o forman parte de accesorios. Un caso muy común son los *smart watch*, que monitorizan datos tanto del entorno como del usuario, como la temperatura corporal y externa, frecuencia cardiaca, momentos de actividad o pasos realizados a lo largo del día. Estos dispositivos en concreto suponen una herramienta muy poderosa ya que permite al propio usuario mantener un seguimiento de su salud o ejercicio realizado en pos de mantener una vida sana y activa. En cuanto al seguimiento de actividad física, no solo los *smart watches* proporcionan esta utilidad, sino también los propios *smartphones*; que con aplicaciones como *Google Fit* de Google, o *Health* y *Fitness* de Apple permiten monitorizar datos de la actividad física del usuario como la distancia y pasos recorridos, altitud, velocidad o recorrido.

Los tipos de dispositivos anteriores son de uso común y se pueden comprar en tiendas estándar. Sin embargo, existen dispositivos más avanzados que requieren de un médico o una condición específica del paciente para ser expedidos. Un ejemplo son los marcapasos o medidores de glucosa. Ambos casos están relacionados con enfermedades o situaciones anormales en la salud del paciente, que require de una monitorización constante de ciertas constantes vitales o niveles de compuestos dado que un valor anormal puede causar efectos irreversibles y fatales en el paciente. Este tipo de dispositivos suelen encontrarse en el cuerpo del paciente y cuentan con conexión a Internet para poder visualizar los datos recogidos en una aplicación o enviar alertas de peligro. Otro dispositivo empleado en salud son los audífonos inteligentes. En este caso, el paciente no requiere de estos dispositivos para monitorizar sus salud, sino para mejorar su calidad de vida y aunque el uso de dispositivos de ayuda a la audición suele estar asociado a los mayores, no se limita a estos. Existen audífonos inteligentes que pueden analizar el sonido recibido y enfatizar las voces mientras que reduce la presencia de ruido ambiente y de fondo. A su vez, algunas soluciones permiten identificar localizaciones y relacionarlas con una configuración específica, como enfatizar las voces que procedan de una dirección específica.

Agricultura Inteligente

Al igual que en las áreas anteriores, el empleo de sensores y actuadores en una arquitectura IoT también se ha comenzado a emplear en el ámbito de la agricultura. Muchas veces el empleo de soluciones IoT se enfoca en monitorizar valores como la temperatura, humedad ambiental y del suelo, cantidad de nutrientes o presencia de polutantes y partículas contaminantes. Este tipo de configuración se suele encontrar en granjas con invernaderos, en los que la monitorización de las condiciones dentro del mismo es esencial para el crecimiento de los cultivos ya que el uso de los invernaderos se enfoca en cultivos que requieren de condiciones muy precisas para poder dar fruto. Este campo se asocia mucho con el cambio climático ya que este afecta negativamente a los cultivos y puede generar escasez y crisis de alimentos [4]. Iniciativas a lo largo del mundo abogan por emplear *smart argiculture* para mejorar la eficiencia de consumo de agua y energía de los cultivos, ya que con el cambio climático se dan más frecuentemente sequías y problemas medioambientales en lugares en los que antes no se daban.

Smart Cities

A lo largo de la última década, el término *Smart Cities* [5] ha ganado notoriedad y cada vez se han publicado cada vez más articulos sobre el tema. Una smart city se puede clasificar como un sistema de sistemas. Al igual que una ciudad se compone de una gran cantidad de elementos que interactúa entre sí, una smart city es un sistema IoT compuesto de diferentes subsistemas que miden y actúan sobre diferentes aspectos de la ciudad. La aparición de las smart cities, responde, como la mayoría de áreas de IoT, a la necesidad de aumentar la eficiencia y desempeño mediante la homogenización de infor-

mación obtenida de gran cantidad de sensores. Por esto último, requiere de tecnologías y técnicas como el procesamiento de datos masivos y soluciones de inteligencia artificial que puedan obtener pronósticos, análisis o explicaciones de los mismos. Las smart cities buscan encontrar soluciones eficientes en campos como la administración y gobierno, felicidad ciudadana con la calidad de vida, la economía haciendo más prospera la ciudad o el medio ambiente, convirtiendo la ciudad en sostenible en diferentes aspectos.

Varios de los aspectos de los diferentes campos que una smart city busca mejorar son, por ejemplo, la movilidad (*Smart Mobility*), respuesta a emergencias, eficiencia energética o sostenibilidad con la gestión de residuos. Por tanto, habrán diferentes sistemas operando en una smart city. En el caso de la sostenibilidad, sensores de polución, temperatura, humedad y gases se encontrarán repartidos por la ciudad, monitorizando estas métricas para poder tener un panoráma claro de las condiciones metereológicas y ambientales de la ciudad. En cuanto a la movilidad, multitud de espiras en las calles medirán la dirección y flujo del tráfico pudiendo tomar decisiones en base al tráfico existente. Este último puede jugar un papel fundamental en la asistencia en situaciones de emergencia por los sanitarios y fuerzas desplegadas, ya que se necesitaría redirigir el flujo del tráfico de forma eficiente para poder hacer llegar a las fuerzas de respuesta a la mayor brevedad posible. En cuanto a a la contaminación, con el crecimiento de las ciudades y de la población mundial se ha dado un aumento de la contaminación producida por los humanos, lo que afecta a ecosistemas de todo el mundo y causa plagas y epidémias. En pos de remediar esto, una smart city presenta sistemas de monitoreo de los desechos generados, pudiendo emplearlos o deshacerse de ellos de forma eficiente.

Existen diferentes beneficios de IoT, a la vez que desafíos que debe superar, estos serán discutidos más adelante.

1.1 TinyML

En la era actual de la tecnología, los dispositivos inteligentes se han vuelto omnipresentes en nuestras vidas diarias. Desde los asistentes virtuales en nuestros teléfonos hasta los electrodomésticos conectados en nuestros hogares, la inteligencia artificial (IA) ha dado lugar a una revolución en la forma en que interactuamos con el mundo digital. Sin embargo, esta revolución no se limita a los dispositivos de gran tamaño y potencia; se ha extendido a dispositivos más pequeños y eficientes energéticamente gracias a la llegada de 'Machine Learning' o 'TinyML'. En este contexto, la detección de objetos en el edge se destaca como un campo emocionante y en rápido crecimiento que aprovecha la potencia de TinyML para llevar la inteligencia a los dispositivos más pequeños y distribuidos en el mundo real

El aprendizaje automático, o "Machine Learning", ha sido históricamente un dominio de los centros de datos y las nubes de cómputo masivo. Esto ha limitado su aplicabilidad a dispositivos que pueden conectarse constantemente a estas fuentes de potencia de procesamiento. Sin embargo, TinyML rompe con esta tradición al llevar la inteligencia directamente a la *edge* o el borde de la red, donde los datos se generan y se utilizan en tiempo real. Este enfoque permite una serie de ventajas notables, como la latencia ultrabaja, la privacidad de los datos y la capacidad de funcionar incluso en entornos sin conexión a Internet.

Una de las aplicaciones más emocionantes de TinyML en el edge es la detección de objetos. Imagine un dron que puede identificar obstáculos en tiempo real, un sistema de seguridad en el hogar que puede reconocer intrusiones, o incluso una cámara de tráfico que puede analizar el flujo vehicular sin depender de una conexión a la nube. Todo esto es posible gracias a la capacidad de ejecutar modelos de aprendizaje automático en dis-

positivos extremadamente pequeños y con recursos limitados, como microcontroladores y sistemas embebidos.

La detección de objetos en el edge mediante TinyML puede sonar como ciencia ficción, pero en realidad, es una tecnología que ya está transformando numerosos sectores. En su núcleo, esto implica la capacitación de modelos de aprendizaje automático para identificar objetos específicos a partir de imágenes o flujos de datos capturados por sensores. Estos modelos se simplifican y optimizan para ser ejecutados en hardware altamente eficiente, como microcontroladores.

Un aspecto clave de esta tecnología es la eficiencia en el consumo de energía. Los dispositivos de edge a menudo funcionan con baterías o fuentes de energía limitadas, por lo que es esencial que los modelos de TinyML sean capaces de realizar inferencias de manera eficiente y económica en términos de energía. Esto se logra mediante técnicas de cuantización, poda y optimización de modelos, lo que permite que incluso los dispositivos más pequeños realicen tareas de detección de objetos de manera efectiva.

En resumen, TinyML con detección de objetos en el edge representa una poderosa convergencia de la inteligencia artificial y la informática embebida. Permite que dispositivos pequeños y distribuidos realicen tareas sofisticadas de detección de objetos sin depender de conexiones a la nube, lo que tiene un impacto significativo en una amplia gama de aplicaciones, desde la atención médica hasta la movilidad autónoma. A medida que esta tecnología continúa avanzando, podemos esperar ver un mundo donde la inteligencia está verdaderamente en todas partes, impulsando la próxima ola de innovación tecnológica.

1.2 Motivación

Muchos de los parques naturales españoles cuentan con lugares privilegiados desde los que observar el terreno o la fauna y flora local. Estos miradores u observatorios muchas veces son accesibles por caminos estrechos y presentan poco aforo. A su vez, no tienen poca afluencia, pero en algunos casos, aquellos que acceden al observatorio pueden quedarse en el mismo por un periodo prolongado de tiempo, por lo que sería de utilidad mantener un control o monitorizaciíon no solo de los accesos sino también de las salidas. Como no sería viable mantener personal en las entradas las 24 horas del día, dado el difícil acceso de los lugares y el coste de destinar todos los días del año a una persona al lugar, un método fiable y de bajo coste es el uso de un sensor que proporcione esta información para ser empleada posteriormente en estudios.

Por tanto, el presente trabajo busca emplear un dispositivo con acceso a pocos recursos equipado con una cámara de vídeo para así identificar las personas que entran y salen de un observatorio en el ámbito de un parque natural. Esto se realizará con un modelo de Machine Learning (ML) en el propio dispositivo, que obtenga el número de personas entrantes o salientes en un periodo de tiempo arbitrario y envíe estos contadores a un servidor.

Este método se ha elegido dada la gran carga que supone enviar archivos multimedia y el gran ancho de banda que supondría el constante envío de frames para su análisis, por lo que se ha escogido un modelo de *Edge Computing* con *TinyML* para poder incluir en el dispositivo un modelo de ML que obtenga los datos buscados con el objetivo de abaratar tiempo y recursos.

1.3 Objetivos

Dado lo explicado anteriormente, el presente trabajo tiene como objetivo general el diseñar un sistema energéticamente eficiente que permita el conteo de personas en parques naturales o en lugares remotos con difícil acceso. Esto con el fin de automatizar y hacer la tarea más eficiente. A su vez, el objetivo general se puede separar en varios objetivos específicos, que alcanzados todos supondría haber conseguido el principal y general, los caules serían los siguientes:

- Generar una red neuronal que entre dentro del campo del TinyML, que pueda emplearse en un dispositivo con pocos recursos.
- Que la red neuronal mentada en el anterior objetivo presente una precisión alta.
- El proceso general debe ser rápido, o tener poca latencia, dado que al analizarse un video y necesitarse este análisis en tiempo real, se requiere del modelo de ML que analize lo más rápido posible.
- Identificar con la máxima precisión alcanzable las personas que pasen y su dirección respecto el punto de referencia del dispositivo.
- El proceso creado en este trabajo debe ser fácilmente desplegable.

1.4 Impacto esperado

Como anteriormente se ha explicado, el implementar *Edge Computing* permite reducir el consumo de energía y el ancho de banda en la nube, permitiendo realizar más operaciones o soportar más conexiones. El presente trabajo propone emplear *Edge Computing*, y en concreto el subcampo del mismo llamado *TinyML*, para diseñar un sistema eficiente energéticamente que pueda contar personas en lugares remotos. Este sistema debería permitir a los administradores de parques naturales, fuerzas de emergencia y encargados de gobierno tener acceso a la información necesaria para esbozar la afluencia y presión en tiempo real que se ejerce sobre estos espacios protegidos.

Por tanto el impacto que este trabajo se espera que tenga atañe a diferentes áreas, entre las que se encuentran el turismo, economía y calidad de vida de las áreas cercanas, gracias a que con la información obtenida se pueden identificar lugares con gran afluencia de personas, siendo marcados por tanto como *hot spots* turísticos. A su vez, esta ayuda a la creación de modelos turísticos se espera que tenga un impacto en la economía, ya que al identificar estos lugares se pueden tomar medidas que puedan beneficiar económicamente a los alrededores. Por ejemplo, gracias al sistema ideado se identifica que uno de los parques naturales cercanos a un municipio recibe muchas visitas, siendo un potencial lugar turístico y de interés. Gracias a esto, se emprende un proyecto que puede generar más turismo en los alrededores, como la construcción de hoteles y albergues, u otros edificios destinados a albergar turistas, en localidades cercanas. Esto generaría que estas localidades experimentasen un incremento en la economía del lugar, creándose tiendas, restaurantes y oficinas de turismo que ofrezcan rutas guiadas. Todo lo anterior mejorando la economía del lugar, y por tanto, la calidad de vida de los vecinos de las áreas cercanas.

Otro de los efectos que se espera que el trabajo tenga es un impacto la conservación de la naturaleza. El diseño del sistema que se plantea en este trabajo debe ser eficiente energéticamente, por lo que el consumo de energía debe ser exiguo, esto permite el no

necesitar construir casetas para almacenar un generador para el dispositivo, reduciendo enormemente el impacto en la naturaleza que tiene el sistema, pudiendo ser suministrado de energía mediante una placa solar, que tiene poca repercusión en los alrededores. Esto es por que el tipo de lugares en los que se quiere desplegar el sistema son reservas protegidas o parques naturales, y por tanto se debe tener sumo cuidado de no perturbar el entorno. A su vez, al identificar, mediante el sensor, lugares con mucha afluencia de gente, se puede entender que este lugar tan transitado sufre una tensión respecto a la naturaleza. Esto es dado el hecho de que los humanos podemos contaminar ese lugar mediante desechos como bolsas de plástico o restos de comida. Esto puede perturbar a la fauna y flora local, por lo que gracias al trabajo de este proyecto se pueden identificar estos lugares y tomar medidas como posicionar contenedores de basura en los que depositar los residuos generados. O instalar carteles o paneles informativos en los que se recuerde la importancia de cuidar el entorno.

1.5 Estructura de la memoria

Una vez explicado brevemente el contexto del presente trabajo, los objetivos que persigue y el impacto que se espera que proporcione; el contenido restante se estructura de la siguiente forma. En el capítulo 2 se explicarán las tecnologías y metodologías que se han empleado en el trabajo, además de las que han permitido llegar a las mismas. A continuación, en el capítulo 3 se describirá el problema y la solución propuesta, para continuar explicando el estado final de esta solución y cómo se ha llegado a ella. Los resultados de los experimentos se muestran y discuten en el capítulos 4. Así, una vez se han discutido los resultados de los experimentos se explicarán las conclusiones del trabajo en el capítulo 5 y en el capítulo 6 se formularán posibles continuaciones y mejoras al trabajo realizado.

CAPÍTULO 2
Estado del arte

2.1 TinyML

2.1.1. Trasfondo

El uso del Internet de las Cosas (IoT) ha aumentado enormemente en las últimas décadas y, como consecuencia, el número de dispositivos IoT conectados a la red también se ha incrementado. Esto ha causado un problema con el ancho de banda soportado por la red debido al enorme tráfico de paquetes que estos dispositivos inyectan en la misma. Cisco señaló en el Global Cloud Index [6] que la cantidad total de datos creados por cualquier dispositivo IoT alcanzó los 847 ZB anuales en 2021, frente a los 218 ZB anuales de 2016. Esta cantidad de datos generados es dos órdenes de magnitud superior a los datos almacenados en 2021 en centros de datos, 5,9 ZB. Dada la gran cantidad de datos almacenados en la nube y el crecimiento masivo y continuo de los datos generados por los dispositivos conectados, el análisis de Big Data en la nube presenta muchas deficiencias:

En tiempo real: El número de dispositivos conectados aumenta exponencialmente. Cuando se añade una gran cantidad de dispositivos a la red, los datos se siguen transmitiendo a la nube para su procesamiento, pero aumenta la cantidad de los mismos. Debido a esto, el volumen de transmisión de datos aumenta, generando una reducción en el rendimiento del procesamiento debido al gran ancho de banda y el consiguiente retraso. En algunas aplicaciones en tiempo real esto resultaría en caídas del servicio o bajo rendimiento, empeorando la calidad del servicio y la experiencia del usuario.

Consumo de energía: Si aumenta la cantidad de datos a procesar, aumentarán los recursos dados para procesarlos, generando un crecimiento en el uso de energía. No es posible reducir el consumo de energía y abastecer la demanda de procesamiento de datos, dado el aumento previsto en la demanda de esta. El rápido desarrollo del uso de dispositivos inteligentes hará que aumenten los requisitos de consumo de energía de la nube.

Debido al aumento de la cantidad de datos y de las necesidades de computación en la nube, el Edge Computing ha surgido como una alternativa para solucionar este problema, Fig. 2.1. Esta tecnología pretende dotar de inteligencia artificial con *Machine Learning* y análisis de datos a los dispositivos conectados, y hacer que la nube utilice menos recursos, disminuyendo el estrés causado por el procesamiento de los datos generados por estos dispositivos y solucionando el problema del consumo energético haciendo que la nube procese menos datos. *Edge Computing* está más cerca de las fuentes de datos, por lo que es más rápida y segura, lo que permite a los servicios en tiempo real no depender de la nube ni de sus caídas de ancho de banda. De hecho, *Edge Computing* proporciona reducción del consumo de energía en la nube al evitar la transmisión y el procesamiento

de datos en la nube, servicios de alta capacidad de respuesta para aplicaciones móviles y sistemas altamente escalables. Gracias a la distribución de las unidades de procesamiento, *Edge Computing* proporciona ciertas políticas de privacidad garantizadas para el IoT y sistemas tolerantes a las interrupciones del servicio, ya que las interrupciones transitorias de la conexión pueden ocultarse a los usuarios.

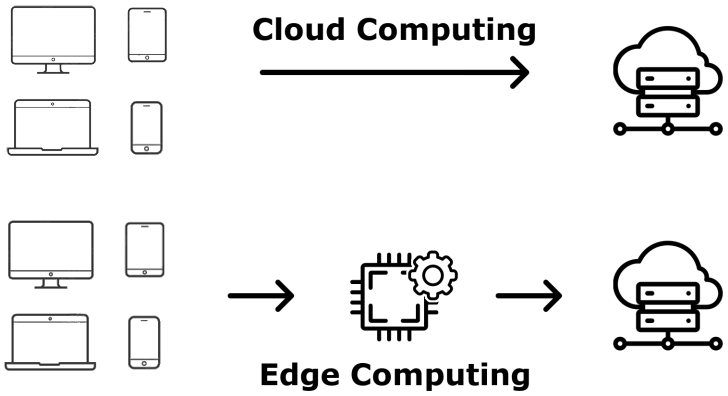

Figura 2.1: Comparación entre los esquemas de *edge* y *cloud computing*

Por lo anterior, TinyML es un paradigma que pretende facilitar la ejecución de modelos ML embebidos en los dispositivos en el *edge* presentes en una red IoT [7] [8], visible en la Fig. 2.2. TinyML permite que estos dispositivos de borde se vuelvan inteligentes y mejoren las capacidades de procesamiento mediante modelos de ML que se ejecutan en el propio dispositivo, haciendo que este paradigma sea capaz de proporcionar servicios de ML autónomos evitando la necesidad de utilizar la computación en nube para analizar estos datos. Debido a este uso potencial del ML en entornos con recursos limitados, como son los dispositivos de borde, TinyML está atrayendo mucha atención tanto en el mundo académico como en el industrial

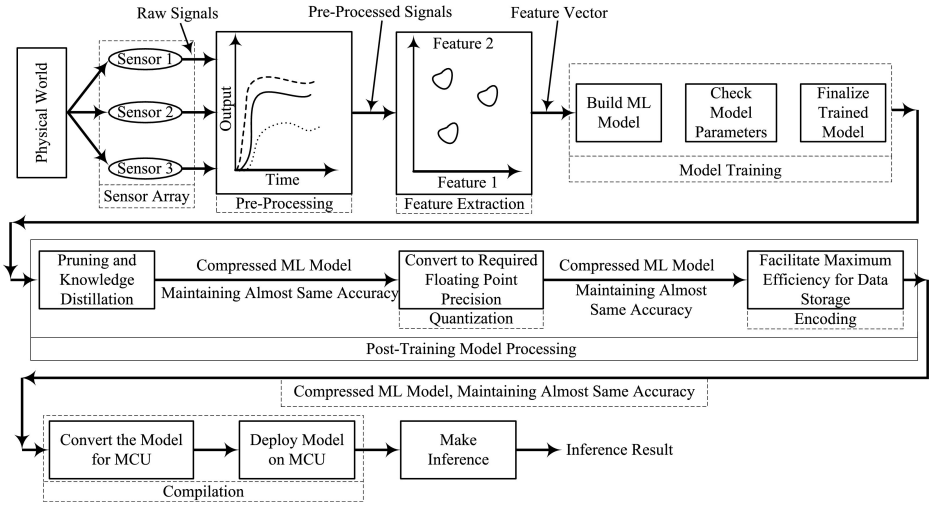

Figura 2.2: Esquema de TinyML

Es importante aclarar que TinyML no está destinado a sustituir a la computación en nube, sino a complementarla dotando a este dispositivo de borde de capacidades que puedan apoyarla. Una forma de implementar TinyML es agrupar varios dispositivos para analizar los datos obtenidos, y que cada dispositivo individual realice una parte del

análisis. Y otro de los enfoques consiste en aislar todo el análisis en un solo dispositivo, suponiendo que cada dispositivo funciona por separado de los demás. De esta forma, se reducen y filtran los datos transmitidos a la nube, ya que el único dato transmitido es el resultado de este análisis o proceso ML.

2.1.2. Beneficios

TinyML traslada gran parte de la computación al extremo de la red IoT, mejorando o introduciendo nuevos métodos de análisis de datos para los dispositivos con recursos limitados. Además, TinyML tiene características inherentes como la versatilidad, dada la gran variedad de dispositivos que pueden ejecutarlo, y la simplicidad. Además, este paradigma tiene ventajas como las que se enumeran a continuación.

Dispositivos inteligentes en el edge

Los dispositivos en el edge generan enormes cantidades de datos en bruto, lo que provoca la incapacidad de la nube para computarlos y analizarlos en tiempo real, junto con problemas con el ancho de banda de la red. De ahí que grandes cantidades de datos se desperdicien en el edge sin llegar a la nube. El paradigma TinyML permite realizar este análisis de los datos en el entorno en el que se encuentran estos dispositivos (lo más cerca de los datos en bruto), convirtiéndolos en dispositivos inteligentes que incorporan métodos de análisis avanzados como algoritmos de ML.

Ancho de banda y latencia

La arquitectura de red IoT tradicional requiere pasarelas para redirigir los datos de los sensores a Internet y servicios en la nube que procesen esos datos. Sin embargo, los enfoques TinyML reducen la presencia de la nube.

TinyML hace que los dispositivos del *edge* sean más independientes de la nube proporcionando herramientas ML para procesar la gran cantidad de datos en bruto capturados por los sensores. Esto hace que se reduzca el volumen de datos transmitidos a la nube proporcionando a la red más ancho de banda libre, junto con una mejora de la latencia. Además, servicios críticos como *Smart Health* o *Smart Security* mejoran su seguridad. Sirviendo de *backbone* proporcionando una latencia muy baja.

Seguridad, privacidad, fiabilidad y filtración de datos

Uno de los factores que afecta más negativamente a la aceptación pública de IoT es la realidad de que grandes cantidades de datos privados o sensibles son transmitidos a la nube. Los usuarios no tienen claro dónde se almacenan estos datos sensibles ni a quién pertenecen. Además, la transmisión a la nube abre la posibilidad de fugas de datos e intercepción por terceras partes malintencionadas. Por ello, en TinyML no se transmiten los datos en bruto, sino una versión procesada, evitando así los peligros mencionados.

Este procesamiento realizado por TinyML puede hacer también que se filtren los datos capturados y se envíen a la nube los datos importantes. Por ejemplo, un sistema de cámaras incrustadas en puertas de seguridad, diseñado para identificar a los empleados que quieren traspasar la puerta a un área restringida. En lugar de enviar toda la imagen capturada a la nube, transmite sólo la identidad del empleado, identificado mediante un reconocimiento facial en el *edge*. La nube solo deberá cotejar el empleado detectado en la base de datos y comprobar si tiene acceso a ese área restringida.

Además, la implantación de TinyML mejora la fiabilidad de la red gracias a la capacidad de los dispositivos para realizar tareas *in situ* en lugares con baja conectividad o difícil acceso, como islas aisladas, zonas rurales alejadas o mar abierto.

Eficiencia energética

En IoT, los dispositivos conectados están siempre activos y consumiendo energía de su fuente de energía (la cuál puede ser desde una batería, un panel solar o la propia red eléctrica), normalmente bastante pequeña. Sin embargo, estos dispositivos deben funcionar durante largos periodos de tiempo, como meses o años. Por lo tanto, a veces necesitan estar en modo de reposo o similar y despertarse cuando se activan y leen los datos. Además, transmitir los datos puede ser más costoso energéticamente que procesarlos con ML y enviar los datos procesados. En este sentido, TinyML puede mejorar el comportamiento del dispositivo en el *edge*.

Bajo coste

En ambas partes, los dispositivos en el *edge* y la nube, necesitan reducir el coste de operación. Uno debido a las limitaciones de hardware y recursos que sufre, y el otro causado por los problemas que conlleva el análisis de datos masivos. Con la tecnología TinyML es posible distribuir la tarea de computación, reduciendo el uso de recursos por ambas partes a la vez que se garantiza la escalabilidad.

Dado todo lo explicado anteriormente, la tabla 2.1 presenta un sumario de la comparación entre Cloud y Edge/TinyML.

Tabla 2.1: Comparación entre Cloud y TinyML

	Cloud	TinyML
Estructura	Centralizada	Descentralizada
Nodos inteligentes	Poca cantidad	Gran cantidad
Requisitos de ancho de banda	Altos	Bajos
Seguridad y privacidad	Baja	Alta
Fiabilidad	Depende de Internet	Alta
Capacidad computacional	Muy alta	Baja

2.1.3. Desafíos y oportunidades

Es evidente que en la situación actual, *Edge Computing* con TinyML no puede resolver por sí sola todos los problemas derivados de la computación centralizada en la nube, pero en un futuro próximo se espera que la satisfaga [9]. Múltiples grupos de investigación de todo el mundo están investigando y evaluando las plataformas TinyML actuales y su rendimiento en diversos campos [10] [11]. Evaluar el consumo de energía y el tamaño de la memoria es necesario para ejecutar modelos TinyML capaces de igualar la potencia del ML en la nube. Evaluar el consumo de energía [12] mientras se ejecuta un modelo ML es esencial en campos como UAVs para determinar si el dispositivo es capaz de realizar y análisis antes de necesitar una recarga de batería [13]. Además, los estudios sobre cómo utilizar dispositivos de baja potencia con TinyML están presentes en diversos campos como la industria [14] o la investigación en general [15]. Además, la evaluación del rendimiento de estos modelos en escenarios reales [16] es una tendencia que trata de

identificar la capacidad de TinyML en estos dispositivos de memoria limitada. Es posible resumir los retos y oportunidades que tiene este paradigma en lo siguiente.

Uso de la energía

Uno de los grandes retos de TinyML es el consumo de energía de los dispositivos en el *edge*, junto con su inconsistencia debido a los diferentes requisitos de energía de cada microprocesador. El desarrollo de la gestión de energía, TinyML es una tarea crítica para lograr la fiabilidad en estos dispositivos. Por tanto, hay margen para la investigación en este aspecto del paradigma. Además, el consumo de batería causado por los algoritmos TinyML aumenta de forma directamente proporcional a los requisitos de uso de recursos, que se incrementan exponencialmente. Además, la mejora en la investigación de la batería es lenta [17]. Sin embargo, este problema depende en parte de un reto externo que hay que superar para ofrecer servicios ML mejorados.

Restricciones de uso de memoria y de los microprocesadores

Este es otro de los grandes retos que presenta TinyML, y como se ha dicho anteriormente, hay investigaciones abiertas evaluando el uso de memoria por parte de los algoritmos TinyML en dispositivos de borde. La altísima exigencia de precisión de los modelos de ML se ha convertido en una parte fundamental de la evolución del medio. Por desgracia, la precisión implica complejidad de cálculo y tamaño del modelo. Cuanto más precisa es la solución del modelo, más compleja y grande es el mismo. Esto plantea algunos retos cuando se intenta implementar algoritmos de ML en un dispositivo con restricciones.

a) La capacidad de memoria y la potencia de cálculo del microprocesador de un dispositivo son reducidas. Por tanto, la implantación de un modelo complejo y de gran tamaño no es viable.

b) Debido a lo anterior, es necesario comprimir y reducir el tamaño del modelo ML sin perder mucha precisión.

Al comprimir un algoritmo ML, si se hace de forma agresiva, el rendimiento puede modificarse volviéndose peor que el modelo original. Además, es necesario un equilibrio entre el modelo reducido y su precisión. En el estado actual de IoT, los dispositivos dependen de la potencia de cálculo de la nube, pero investigadores de todo el mundo están trabajando para encontrar soluciones a este problema. Esto puede acelerar la búsqueda de una solución.

Diferencias entre Cloud and Edge

El funcionamiento de la nube y de las partes integradas del sistema IoT es diferente. La nube tiene potentes CPU y GPU con grandes cantidades de memoria que permiten ejecutar grandes algoritmos de ML, junto con una carga de almacenamiento masiva. Por el contrario, los dispositivos periféricos están limitados por una memoria pequeña y una potencia de cálculo reducida. Por lo tanto, el borde tiene limitaciones de computación y carga que deben tenerse en cuenta al diseñar una arquitectura con computación de borde.

2.2 Detección de objetos

La detección de objetos o personas es una tarea trivial para los humanos, la realizamos a cada segundo de nuestras vidas. Incluso usted, lector, se encuentra realizando esta tarea al identificar cada letra y palabra en este escrito y darle un significado. Los infantes de poca edad comienzan a reconocer objetos comunes pasados unos meses, sin embargo, lograr que un ordenador sea capaz de realizar esta tarea ha resultado ser harto arduo. El objetivo de los estudios e investigaciones actuales y pasadas se ha fundamentado en lograr que las máquinas sean capaces de identificar y localizar dentro de su campo de visión todos los objetos de un tipo (como coches, personas, señales de tráfico, etc.). A su vez, la clasificación, la segmentación, la estimación del movimiento, la comprensión de la escena, así como otras tareas, son otros problemas a los que se enfrenta la visión por ordenador para que pueda realizar funciones que la vida orgánica realiza por millones de miles de años.

Los primeros modelos de detección de objetos se construyeron como un conjunto de extractores de características, como el detector de Viola-Jones [18], el histograma de gradientes orientados o Histogram of Oriented Gradient (HOG) [19], etc. Estos modelos eran lentos, imprecisos y no funcionaban bien en conjuntos de datos no usados en el entrenamiento. Todo cambiío con la introducción de las redes neuronales convolucionales (CNN) y el aprendizaje profundo (*Deep Learning*) para la clasificación de imágenes. Modificando por completo el panorama de la investigación en visión por ordenador. Su uso en el desafío ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 por AlexNet [20] inspiró nuevas investigaciones sobre su aplicación en visión por ordenador. Hoy en día, la detección de objetos tiene aplicaciones que van desde los coches automáticos y la detección de personas hasta la seguridad y los usos médicos. En los últimos tiempos, el desarrollo de nuevas técnicas y herramientas ha experimentado un increíble y exponencial crecimiento

2.2.1. Trasfondo

Planteamiento del problema

La tarea de detectar objetos es una extensión natural de la clasificación de objetos, ya que en la que en la clasificación solo se reconoce un tipo de objeto por imagen. En cambio, el objetivo de la detección de objetos es identificar y reconocer múltiples objetos por imagen, sin ser necesariamente todos de la misma clase, proporcionando también la posición en la imagen de los mismos. El detector de objetos debe poder identificar todas las apariciones de las clases de objetos y proporcionar la *bouniding box*, la cuál es un rectángulo que delimita la posición del objeto en la imagen. La detección de objetos generalmente se considera un problema de aprendizaje supervisado, por lo que se se posée un conjunto de datos (imágenes) con los objetos y sus clases ya definidos y etiquetados, y mediante la comparación de la salida del modelo y la realidad se realizan ajustes que permiten evolucionar al modelo a un estado en el que su precisión sea aceptable. Por tanto, los modelos modernos de detección de objetos tienen acceso a grandes conjuntos de imágenes ya etiquetadas que son usadas en el entrenamiento.

Principales retos en la detección de objetos

Como antes se ha comentado, la visión por computador ha avanzado mucho en la última década gracias a las CNN, sin embargo, aún tiene que superar algunos retos im-

portantes. Algunos de los retos que los modelos de detección de objetos enfrentan a la hora de aplicarse en la vida real son:

- Variación intraclase: La variación intraclase entre las instancias de un mismo objeto es relativamente común en la naturaleza. Esta variación puede deberse a numerosas razones, como la oclusión, la iluminación, la pose, el punto de vista, etc. Estas variaciones y modificaciones pueden ocasionar diferencias extremas en la apariencia de un mismo objeto [21]. Es esperable, por tanto, que los objetos en un caso real presenten deformaciones y rotaciones respecto los objetos vistos en la muestra de control empleada en el entrenamiento.

- Número de categorías y datos: Este tipo de detectores requieren una cantidad ingente de objetos y clases de objetos ya etiquetadas para poder acercarse a la realidad. A su vez, estos datos deben ser de alta calidad para evitar errores o *bias* en la aplicación real.

- Eficacia: Los modelos actuales necesitan consumir grandes recursos para generar resultados de detección precisos. Con la generalización de los dispositivos móviles y en el *edge*, la eficiencia de estos detectores es crucial para el futuro desarrollo en el campo de la visión por ordenador en todas su aplicaciones, con especial énfasis en el ámbito del *edge*.

2.2.2. Datasets y técnicas de evaluación

Datasets

Como se ha explicado antes, los modelos actuales de detección requeiren de ingentes cantidades de datos ya etiquetados. A lo largo de los años han surgido diferentes tipos de datasets. Los siguientes son algunos de los más usados.

Tabla 2.2: Tabla con los datasets más usados en detección de objetos.

Dataset	Classes	Train		Validation		Test
		Images	Objects	Images	Objects	
PASCAL VOC	20	5717	13609	5823	13841	10991
MS-COCO	80	118287	860001	5000	36781	40670
OpenImage	600	1743042	14610229	41620	204621	125436

Pascal VOC. Pascal VOC o Pascal Visual Objet Classes [22] es un reto a lo largo de los años diseñado para acelerar el desarrollo de algoritmos de detección. Comenzó en 2005 para tareas tanto de clasificación como de detección. Actualmente, este dataset cuenta con 11000 imágenes de entrenamiento y más de 27000 objetos etiquetados. Estas imágenes están diferenciadas en 20 clases diferentes, entre las que se encuentra: personas, coches, bicicletas, sillas, gatos, perros, etc. Este reto también introdujo una de las medidas de evaluación más empleadas actualmente: la precisión media o *Mean Average Precision* (mAP).

MS-COCO. Microsoft Common Objects in Context (MS-COCO) [23] es uno de los conjuntos de datos más complejos actualmente. Se trata de un dataset con objetos que un niño de 4 años puede reconocer. En concreto, presenta 80 categorías, 118287 imágenes de entrenamiento con 860000 objetos detectables en total. Este dataset incluye imágenes con objetos desde diferentes ángulos, obteniendo el modelo de detección mayor precisión en un escenario real.

Open Image. Open Images es un dataset de Google [24] que se compone de 9,2 millones de imágenes, anotadas con etiquetas a nivel de imagen y máscaras de segmentación. Fué lanzado en 2017 y desde entonces ha recibido seis actualizaciones. Open Images cuenta con 16 millones de *bounding boxes* para 600 categorías en 1,9 millones de imágenes para detección. Por tanto, lo convierte en el mayor conjunto de datos para detectar objetos. En su creación se puso especial cuidado en elegir imágenes complejas y diversas. Se introdujeron varios cambios en el mAP introducido en Pascal VOC, como ignorar las clases no anotadas, el requisito de detección para la clase y su subclase, etc.

Métricas

Los modelos de detección de objetos emplean múltiples criterios para evaluar el rendimiento del mismo, siendo algunos los FPS (fotogramas por segundo), precisión o recall. Aunque el *Mean Average Precision* (mAP) es la más común actualmente. La precisión se obtiene a partir de la *Intersection Over Union* (IoU), que es la relación entre el área de intersección y el área de unión entre el *ground-truth* y la *bounding box* predicha para el objeto. En base a esta medida se establece un umbral para saber si la detección es correcta o se debe mejorar. Si es superior al umbral se clasifica como True Positive, si no, como False Positive. Por el contrario, si un objeto no es detectado se denomina False Negative. Así, la precisión mide el porcentaje de detecciones correctas y el recall mide las detecciones correctas respecto a las etiquetas.

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \\ &= \frac{\text{True Positive}}{\text{All Observations}} \end{aligned} \tag{2.1}$$

$$\begin{aligned} \text{Recall} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \\ &= \frac{\text{True Positive}}{\text{All Ground Truth}} \end{aligned} \tag{2.2}$$

La mAP se calcula por separado para cada clase y se emplea como métrica única para la evaluación final.

2.2.3. Arquitecturas backbone

Las arquitecturas troncales son uno de los componentes más importantes del detector de objetos. Estas redes extraen características de la imagen de entrada utilizada por el modelo. Aquí hemos analizado algunas de las arquitecturas troncales más utilizadas en los detectores modernos (Tabla 3). La Fig. 7 ofrece una visualización simplificada de estas redes CNN.

Tabla 2.3: Tabla con las arquitecturas base más empleadas en detección de objetos.

Modelo	Anõ	Capas	Parametros (Millón)	Top-1 acc %	FLOPs (Billion)
AlexNet	2012	7	62.4	63.3	1.5
VGG-16	2014	16	138.4	73	15.5
ResNet-50	2015	50	25.6	76	3.8
EfficientNet	2019	160	19	83	4.2

AlexNet

Krizhevsky et al. propusieron AlexNet [20], una arquitectura basada en CNN diseñada para clasificar imágenes, y ganaron el reto *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) de 2012. El modelo logró una precisión considerablemente mayor (más del 26 %) que los modelos del momento, por lo que supuso una revolución en el campo. AlexNet se compone de ocho capas entrenables: cinco capas convolucionales y tres capas *fully connected*. La última capa está conectada a un clasificador de tipo *softmax* de N vías, siendo N el número de clases a clasificar. Al ser una CNN emplea convoluciones con el fin de obtener mapas de caracterśiticas y emplea operaciones como la ReLU para cribar las características y acelerar el entrenamiento. Gracias a este modelo, las CNN se han labrado un puesto como las redes más usadas en el procesamiento y análisis de imágnes.

VGG

AlexNet y las CNN que le siguieron se enfocaron en ventanas con tamaños pequeños para afinar al precisión del modelo. En cambio, Simonyan y Zisserman se enfocaron en estudiar como variaba el rendimiento de las redes al cambiar su profundidad. Al final, propusieron la red VGG [25], que emplea filtros convolucionales pequeños en redes de distintas profundidades. El artículo en el que se propuso la red demostraba como una arquitecutra de red más profunda (16 o 19 capas, entonces) podía ser usada para clasificar o detectar con gran precisión. La arquitectura de esta red se caracteriza por añadir agrupaciones de tres capas convolucionales, y al final, tres capas *fully connected* seguidas de un *softmax*.

ResNet

La arquitectura VGG [25] cambió el paradigma de la construcción de CNN para análisis de imágenes. A partir de entonces la CNN que aparecieron fueron cada vez más profundas y contaban con mayor número de parámetros. Kaiming et al. [26] mostraron como al aumentar cada vez más la profundidad de la red la precisión se satura y luego sufre una rápida degradación. En su artículo propusieron una arquitectura en la que se daban saltos entre capas hacia adelante, concatenando las entradas y salidas de la capa para así poder aumentar la profundidad pero sin perder la precisión. Demostraron que la red VGG16, con 16 capas de profundidad presentaba peor precisión y mayor complejidad que las redes ResNet101 y ResNet152 con 101 y 152 capas respectivamente. Posteriormente, propusieron ResNetV2 [27], que usaba *Batch Normalization* y capas de activación ReLU. Esta última es más generalizada y fácil de entrenar.

EfficientNet

La red EfficientNet [28] fue propuesta en 2019 por Tan et al. al estudiar los efectos en el modelo de clasificación/detección que tenía el escalar la red, es decir, aumentar la profundidad, dimensiones de los filtros o resolución de la misma. Descubrieron que aumentar la profundidad puede ayudar a capturar características más complejas, pero dificulta el entrenamiento por el problema de desaparición del gradiente [29]. Al variar las dimensiones de los filtros se facilitará el obtener características de grano fino, pero dificultará la obtención de características de alto nivel. Por otro lado, no se encuentran ventajas significativas al aumentar la resolución del modelo ya que se saturan, es cada vez más difícil el mejorar. En su artículo proponen usar un coeficiente compuesto que es-

cale uniformemente en estas tres caracterśiticas, mediante tres parámetros. ϕ controlaría el tamaño de la red. A medida que ϕ aumenta, tanto el ancho como la profundidad de la red se incrementan proporcionalmente. Esto permite adaptar la arquitectura a diferentes niveles dependiendo de los recursos disponibles. γ se encarga de controlar cómo se escalan los canales en cada capa. Aumentarlo aumenta el número de canales en cada capa de manera uniforme. Y por último, τ controla la resolución de entrada de la red. A medida que τ aumenta, la resolución de las imágenes de entrada también lo hace. EfficientNet es una arquitectura sencilla y eficiente. Supera a los modelos existentes en precisión y velocidad a la vez que es considerablemente más pequeña. Al proporcionar un aumento monumental de la eficiencia, podría abrir potencialmente una nueva era en el campo de las redes eficientes.

2.2.4. Modelos detectores de objetos

Dentro de los modelos detectores se pueden encontrar dos categorías: modelos de dos fases y modelos de una fase. Los modelos en dos fases son aquellos que cuentan con un módulo para obtener propuestas de regiones en las que hayan objetos, y otro módulo que se encarga de obtener la localización exacta del objeto y su clase. Estos modelos son complejos y carecen del contexto global de la imagen. Por otro lado, los detectores de una fase realizan la obtención de las ubicaciones de los objetos y sus clases en una única clase. Estos modelos superan a los modelos en dos etapas por tener un diseño mucho más simple y un mejor rendimiento en tiempo real. También existen los detectores basados en transformers, los cuales requieren más datos para ser entrenados en comparación con los basados en CNN.

Detectores en dos fases

R-CNN

Rich feature hierarchies for accurate object detection and semantic segmentation [30] fue el primer artículo en proponer un modelo de la familia R-CNN (Region based CNN). En este artículo se demostró una forma de emplear CNN para mejorar enormemente el rendimiento del modelo. Los modelos R-CNN proponen una primera fase en la que se proponen regiones independientemente de la clase, buscando convertir el problema de detección en uno de clasificación. Este módulo expide 2000 regiones candidatas a objeto. Luego, estas regiones son empleadas en una CNN AlexNet para extraer características. A continuación, estas características son transmitidas a una SVM entrenada para cada clase con el fin de obtener valores de confianza. Posteriormente, se aplica una *non-max supression* a las regiones puntuadas, en función de su IoU y clase. Una vez identificada la clase, el modelo identifica su *bounding box* utilizando un regresor entrenado, que predice cuatro parámetros, es decir, las coordenadas centrales con su anchura y altura.

Como antes se ha explicado, los modelos en dos fases son complejos y esta complejitud es transladada también a su entrenamiento. La R-CNN tiene un complicado proceso de entrenamiento en varias etapas. La primera fase consiste en preentrenar la CNN con un gran conjunto de datos de clasificación. A continuación, se ajusta para la detección utilizando imágenes concretas de lo que se quiere detectar mediante la sustitución de la capa de clasificación por un clasificador de N+1 vías inicializado aleatoriamente, siendo N el número de clases, utilizando el descenso de gradiente estocástico (SGD) [31]. Para cada clase se entrena un SVM y un regresor.

La R-CNN marcó el comienzo de una nueva era en el campo de la detección de objetos, pero era lenta (47 segundos por imagen) y costosa en tiempo y espacio [28]. Su

proceso de entrenamiento era complejo y se tardaban días en entrenar conjuntos de datos pequeños, incluso cuando se compartían algunos de los cálculos.

Fast R-CNN

Como se ha explicado antes, el modelo R-CNN era lento y necesitaba entrenar varios sistemas por separado (lo que aumentaba la complejidad de entrenamiento). Fast R-CNN [32] surgió para solucionar este problema, creando un único sistema entrenable. A la red CNN procesa la imágen y es sobre los mapas de características que se aplican las regiones propuestas. Tras esto, la siguiente capa es llamada ROI, que realiza el pooling de las regiones sobre las características. La salida de esta es pasada a dos capas *fully connected* que acaban en una capa *softmax* que obtiene las clases de los objetos. Mientras que otra rama de la salida acaba en una capa de regresión que obtiene las coordenadas de cada *bounding box*. También se cambió la función de pérdida del regresor a la función L1 suave con el fin de mejorar el rendimiento. La red fue entrenada mediante descenso por gradiente estocástico y lotes de imágenes de 2, permitiendo una convergencia más rápida.

Fast R-CNN mejoró la velocidad unas 146 veces respecto a R-CNN, además de aumentar consigo la precisión, simplificó el entrenamiento y presentó una nueva función de pérdida mejorada. El detector de objetos, sin la red de propuesta de regiones, registró una velocidad cercana al tiempo real con una precisión considerable.

Faster R-CNN

Aunque, como se ha explicado antes, Fast R-CNN mejoraba enormemente el tiempo de detección, seguía siendo lento en cuanto a la categoría de tiempo real, ya que tardaba 2 segundos por imagen, en comparación con los 0.2 segundos por imagen que sería denominado como tiempo real.

En comparación con sus predecesores, antes obtiene los mapas de características de la imagen, de los cuales propone las regiones, en vez de proponer las regiones de la imagen inicial. Utiliza múltiples *bounding boxes* de diferentes relaciones de aspecto y realiza una regresión sobre ellas para localizar el objeto. A la hora de obtener las regiones propuestas, también se obtiene la clase a la cuál pertenecería mediante un clasificador. Por último se envían a la capa *fully connected*. El resultado se envía entonces al clasificador y al regresor de *bounding boxes*.

El entrenamiento de esta iteración de R-CNN es más complicado ya que hay capas que se comparten entre redes que realizan tareas diferentes. En primer lugar, la sección que propone regiones se entrena previamente en el conjunto de datos ImageNet [33] y se ajusta en el conjunto de datos PASCAL VOC [22]. Fast R-CNN es entrenada a partir de las regiones propuestas en el primer paso.

Faster R-CNN [34] mejoró la precisión de la detección en más de un 3 % con respecto a sus predecesores [32] y redujo el tiempo de detección . Corrigió el cuello de botella que suponía la lentitud de la propuesta de regiones y funcionó casi en tiempo real a 5 FPS. Al contar con una CNN en el paso de propuesta de regiones, esta red puede producir mejores propuestas, mejorando así la precisión.

Mask R-CNN

Mask R-CNN [35] se propueso para ampliar el modelo Faster R-CNN añadiendo otra rama en paralelo para la segmentación de instancias de objetos a nivel de píxel. Esta rama es una red *fully connected* que se aplica a los ROI para clasificar cada píxel en segmentos con un coste computacional total reducido. En cuanto a la propuesta de regiones emplea

la misma arquitectura que el modelo Faster R-CNN, pero añade una salida paralela a la clasificación y *bounding boxes* para proporcionar una máscara de segmentación. A diferencia de Faster R-CNN, emplea una capa ROIAlign en vez de una ROIPool para evitar problemas de alineación a nivel de pixel de cara a la segmentación. Otra diferencia es el uso de la red ResNet-101 como base para el modelo, permitiendo una mayor velocidad y precisión. La función de pérdida es actualizada con la función de pérdida empleada en la segmentación y emplea 5 *anchor boxes* y 3 *aspect ratio*. Aun así el entrenamiento de este modelo es similar a Faster R-CNN.

Mask R-CNN obtiene mejor rendimiendo que las arquitecturas de detección de única fase del momento, además de añadir la funcionalidad de segmentación por poco coste computacional. Es fácil de entrenar y flexible, pero su rendimiento sigue por debajo del tiempo real, al funcionar a más de 30 FPS.

Detectores de única fase

YOLO

Los detectores de dos etapas resuelven la detección de objetos como un problema de clasificación: un módulo presenta candidatos que la red clasifica como objeto o como fondo. Sin embargo, YOLO o You Only Look Once [36] lo replantea como un problema de regresión, prediciendo las coordenadas de la *bounding box* del objeto. En YOLO, la imagen de entrada se divide en una cuadrícula S x S y la celda donde cae el centro del objeto es la responsable de detectarlo. Una celda de la cuadrícula predice múltiples cuadros delimitadores, y cada matriz de predicción consta de 5 elementos: coordenadas del x e y de la *bounding box*, dimensiones de la misma (alto y ancho) y la puntuación de confianza.

YOLO se inspira en el modelo GoogLeNet para la clasificación de imágenes [37], que utiliza módulos en cascada de CNN más pequeñas. Se entrena previamente con datos de ImageNet [33] hasta que el modelo alcanza una gran precisión y, a continuación, añaden capas convolucionales y *fully connected*. Puede darse el caso de que un mismo objeto sea detectado multiples veces, por lo que se emplea *Non maximum suppression* (NMS), que elimina las detecciones múltiples específicas de cada clase.

YOLO superó los modelos de una única fase de entoces en tiempo real por un enorme margen tanto en precisión como en velocidad. Sin embargo, también presentaba importantes deficiencias. La precisión de la localización de objetos pequeños o agrupados y la limitación del número de objetos por celda eran sus principales inconvenientes. Estos problemas se solucionaron en versiones posteriores de YOLO [38] [39] [40].

YOLOv2 y YOLO9000

YOLOv2 [38] surgió como una mejora de YOLO. Este modelo ofrecía un buen equilibrio entre velocidad y precisión, mientras que el modelo YOLO9000 predice 9000 clases de objetos en tiempo real. Sustituyeron la arquitectura base del modelo por una Darknet-19 [41]. Entre otras técnicas, emplea *Batch Normalization* para mejorar la convergencia, entrena a la vez la clasificación y detección para aumentar la velocidad y emplea *anchor boxes* para aumentar la precisión. .

YOLOv2 ofrece mayor flexibilidad para elegir el modelo en función de la velocidad y la precisión, y la nueva arquitectura tenía menos parámetros que su predecesor.

YOLOv3

YOLOv3 presenta grandes mejoras respecto a las versiones predecesoras de YOLO [36] [38]. Se sustituyó la CNN que extrae los mapas de características por una red Darknet-

53 de mayor tamaño. También incorporaron varias técnicas como el aumento de datos, el entrenamiento multiescala y *Batch Normalization*, entre otras. La capa *softmax* en la capa de clasificación fue sustituido por un clasificador logístico.

Aunque YOLOv3 era más rápido que YOLOv2, no presenta ningún cambio significativo en cuanto a su predecesor, incluso llega a tener menos precisión que un detector de última generación.

YOLOv4

YOLOv4 [40] incorporó un montón de ideas interesantes para diseñar un detector de objetos rápido y fácil de entrenar que pudiera funcionar en los sistemas de producción existentes. Utiliza "bag of freebies"[42], es decir, métodos que sólo aumentan el tiempo de entrenamiento y no afectan al tiempo de inferencia. YOLOv4 utiliza técnicas de aumento de datos, métodos de regularización, suavizado de etiquetas de clase, CIoU, Cross mini-Batch Normalization (CmBN), Self-adversarial training, Cosine annealing scheduler [43] además de otras técncias para mejorar el entrenamiento. También se añaden a la red métodos que sólo afectan al tiempo de inferencia, llamados "Bag of Specials", como la activación Mish [44], conexiones parciales entre etapas (CSP), PAN path aggregated block, Multi input weighted residual connections (MiWRC), etc. También utiliza un algoritmo genético para la búsqueda de hiperparámetros. Cuenta con una red troncal CSPNetDarknet-53 preentrenada en ImageNet, y YOLOv3 como módulo de detección principal.

La mayoría de los algoritmos de detección existentes requieren varias GPU para entrenar el modelo, pero YOLOv4 puede entrenarse fácilmente en una sola GPU. Es el doble de rápido que EfficientDet y ofrece un rendimiento comparable.

SSD

Single Shot MultiBox Detector (SSD) [45] fue el primer detector de una sola fase que igualó la precisión de los detectores del momento de dos fases como Faster R-CNN [34], además de mantener la velocidad en tiempo real. SSD se construyó sobre VGG-16 [25], con estructuras auxiliares adicionales para mejorar el rendimiento. Estas capas auxiliares de convolución, añadidas al final del modelo, disminuyen progresivamente de tamaño. SSD detecta objetos más pequeños al principio de la red, cuando las características de la imagen no son demasiado toscas, mientras que las capas más profundas son responsables del desplazamiento de las *bounding boxes* y relaciones de aspecto predeterminadas [46]. Durante el entrenamiento, SSD empareja cada caja de verdad con la caja por defecto con el mejor solapamiento y entrena la red en consecuencia. El resultado final se obtiene realizando *Non maximum suppression*.

Aunque SSD era significativamente más rápido y preciso que redes como YOLO y Faster R-CNN, tenía dificultades para detectar objetos pequeños. Este problema se resolvió posteriormente utilizando mejores arquitecturas troncales como ResNet [26] e implementando otras pequeñas correcciones.

RetinaNet

Hay una marcada diferencia entre los detectores en una fase y dos fases, la precisión. Lin et al. sugieren en su artículo que la causa de la relativa baja precisión de los fase única frente a los de dos fases biene causado por el 'extremo desequilibrio entre las clases *foreground* y *background*' [47]. Propusieron la *Focal Loss*, un parámetro de entropía cruzada, para reducir este desequilibrio, reduciendo así la contribución a la pérdida. Para demostrar su eficacia emplearon un detector sencillo llamado RetinaNet. Este modelo lo-

gra detectar objetos mediante muestreo denso de la imagen de entrada en referencia a las coordenadas, escala y relación de aspecto. De base emplea la red ResNet [26] aumentada con un *Feature Pyramid Network* (FPN) [48], que termina en un clasificador y en un regresor.

EfficientDet

EfficientDet [49] se basa en la idea de un detector escalable con mayor precisión y eficiencia. Introduce características multiescala eficientes, BiFPN y escalado de modelos. BiFPN es una red piramidal bidireccional de características con pesos entrenables para la conexión cruzada de características de entrada a diferentes escalas. Esto elimina los nodos menos eficientes y mejora la fusión de características de alto nivel. A diferencia de los detectores existentes, que escalan con capas FPN más grandes y profundas o apiladas, EfficientDet introduce un coeficiente compuesto que se puede utilizar para 'escalar conjuntamente todas las dimensiones de la red troncal, la red BiFPN, la red de clase/caja y la resolución'. EfficientDet utiliza EfficientNet [28] como red troncal con múltiples conjuntos de capas BiFPN apiladas en serie como red de extracción de características. Cada salida de la última capa BiFPN se envía a la red de predicción de clases y cajas. El modelo se entrena utilizando el optimizador SGD junto con *Batch Normalization* y utiliza la activación swish, en lugar de la activación ReLU estándar, que es diferenciable, más eficiente y tiene un mejor rendimiento.

EfficientDet consigue una mayor eficacia y precisión que los detectores anteriores, a la vez que es más pequeño y barato desde el punto de vista computacional. Es fácil de escalar y se generaliza bien para otras tareas.

2.2.5. Detectores basados en transformers

Los transformers [50] han tenido un profundo impacto en el ámbito del Procesamiento del Lenguaje Natural (PLN) desde sus inicios. Su aplicación en modelos lingüísticos como BERT (Bidirectional Encoder Representation from Transformers) [51], GPT (Generative Pre-trained Transformer) [52], T5 (Text-To-Text Transfer Transformer) [53] etc. han impulsado el estado del arte en este campo. Los transformers utilizan el modelo de atención para establecer dependencias entre los elementos de la secuencia y pueden atender a contextos más largos que otras arquitecturas secuenciales. El éxito de los transformadores en PLN despertó el interés por su aplicación en visión por computador. En el artículo [54] se introdujo el primer modelo basado en transformers enfocado en clasificación llamado ViT. Mientras que las CNNs han sido la columna vertebral en el avance de la visión, los transformadores tienen deficiencias inherentes como la falta de sesgo inductivo, pesos fijos post-entrenamiento [55], etc.

DeTR

Detection Transformer o DeTR fue presentado por Carion et al. en [56] ya hace casi cuatro años. Utilizaba una combinación de CNN y transformers para crear un detector entrenable y eliminaba cualquier módulo hecho a mano, como *Non maximum suppression* o la generación de *anchor boxes*. La imagen de entrada pasa primero por una red troncal CNN para obtener los mapas de características de la imagen, y después por un conjunto de transformers. Las predicciones finales, las clases y *bounding boxes*, se obtienen pasando la salida del transformador a través de redes feed-forward. DeTR utilizó ResNets [26] como red troncal y transformadores. El codificador transformador toma las los mapas de características de la imagen, junto con las codificaciones de posición como entrada y dirige el resultado al decodificador. El decodificador manipula N incrustaciones de

entrada para generar la salida. Las atenciones multicabezal del descodificador modifican estas consultas de objetos con las incrustaciones del codificador para generar resultados, que en última instancia pasan por los perceptrones multicapa para predecir la clase y *bounding boxes*. Como el número de salidas en DeTR viene definido por las consultas de objeto, se asigna una clase especial 'sin objeto' o ∅ a la clase de fondo. DeTR utiliza la pérdida de correspondencia bipartita para encontrar la correspondencia óptima uno a uno entre la salida del detector y la verdad de fondo acolchada.

Se introdujo un detector simple basado en transformadores de propósito general que es competitivo con los detectores basados en CNN. Sin embargo, su rendimiento en objetos pequeños deja bastante que desear.

2.2.6. Redes *Lightweight*

En los últimos años ha surgido una nueva rama de investigación orientada al diseño de redes pequeñas y eficientes para dispositivos con recursos limitados, como es habitual en los despliegues de Internet de las Cosas (IoT), extendiéndose esta tendencia también hasta el desarrollo de detectores de objetos. Conforme esta tendencia se ha ido estableciendo y cada vez han salido más trabajos sobre este campo, se ha observado que los detectores de objetos mostrados en ellos obtienen gran precisión y velocidad en tiempo real. Sin embargo, mayoritariamente requieren de muchos recursos para funcionar, por lo que no son aptos para despliegues en dispositivos con recursos limitados, como son aquellos en el *edge*.

Muchos enfoques diferentes han mostrado resultados interesantes en el pasado. La utilización de componentes eficientes y técnicas de compresión como la poda [57] [58], la cuantización [59] [60], el hashing [61], etc. han mejorado la eficiencia de los modelos de aprendizaje profundo. El uso de redes grandes entrenadas para entrenar modelos más pequeños, denominado destilación [62], también ha mostrado resultados interesantes. Sin embargo, en esta sección, exploramos algunos ejemplos destacados de diseño eficiente de redes neuronales para lograr un alto rendimiento en dispositivos *edge*.

SqueezeNet

Los avances recientes en el campo de las CNN se habían centrado principalmente en mejorar la precisión del estado del arte en los conjuntos de datos de referencia, lo que llevó a una explosión del tamaño de los modelos y sus parámetros. Pero en 2016, Iandola et al. propusieron una red más pequeña e inteligente llamada SqueezeNet [63], que reducía los parámetros manteniendo el rendimiento. Lo consiguieron empleando tres estrategias de diseño principales, a saber, utilizando filtros más pequeños, disminuyendo el número de canales de entrada a filtros 3x3 y colocando capas de reducción de muestreo más adelante en la red. Las dos primeras estrategias reducen el número de parámetros e intentan mantener la precisión, mientras que la tercera aumenta la precisión de la red. El componente básico de SqueezeNet es el módulo de disparo, que consta de dos capas: una capa de compresión y una capa de expansión, cada una con una activación ReLU. La capa de compresión está formada por varios filtros 1x1, mientras que la capa de expansión es una mezcla de filtros 1x1 y 3x3, lo que limita el número de canales de entrada. La arquitectura de SqueezeNet se compone de una pila de 8 módulos Fire aplastados entre las capas de convolución. Inspirado en ResNet [26], también se propuso SqueezeNet con conexiones residuales, lo que aumentó la precisión respecto al modelo vainilla. Los autores también experimentaron con Deep Compression [59] y consiguieron reducir en 510 veces el tamaño del modelo en comparación con AlexNet, manteniendo la precisión de referencia. SqueezeNet se presentó como un buen candidato para mejorar la eficiencia hardware de

las arquitecturas de redes neuronales.

MobileNets

MobileNet [64] se alejó de los métodos convencionales de modelos pequeños, como la reducción, la poda, la cuantización o la compresión, y en su lugar utilizó una arquitectura de red eficiente. La red utiliza la convolución separable en profundidad, que factoriza una convolución estándar en una convolución en profundidad y una convolución puntual 1x1. Una convolución estándar utiliza kernels en todos los canales de entrada y los combina en un solo paso, mientras que la convolución en profundidad utiliza kernels diferentes para cada canal de entrada y utiliza la convolución puntual para combinar las entradas. Esta separación del filtrado y la combinación de características reduce el coste computacional y el tamaño del modelo. MobileNet consta de 28 capas convolucionales separadas, cada una de ellas seguida de una normalización por lotes y una función de activación ReLU. Howard et al. también introdujeron dos hiperparámetros de reducción del modelo: la anchura y el multiplicador de resolución, con el fin de mejorar aún más la velocidad y reducir el tamaño del modelo. El multiplicador de anchura manipula uniformemente la anchura de la red reduciendo los canales de entrada y salida, mientras que el multiplicador de resolución influye en el tamaño de la imagen de entrada y sus representaciones en toda la red. MobileNet logra una precisión comparable a la de algunos modelos completos con un tamaño muy inferior. Howard et al. también demostraron que podía generalizarse a diversas aplicaciones como la atribución de rostros, la geolocalización y la detección de objetos. Sin embargo, era demasiado simple y lineal como el VGG y, por tanto, tenía menos vías para el flujo de gradiente. Esto se solucionó en iteraciones posteriores de este modelo.

ShuffleNet

En 2017, Zhang et al. presentaron ShuffleNet [65], una arquitectura de red neuronal extremadamente eficiente desde el punto de vista computacional, diseñada específicamente para dispositivos móviles. Reconocieron que muchas redes eficientes pierden eficacia a medida que se reducen y afirmaron que se debe a las costosas convoluciones 1x1. Junto con la barajadura de canales, propusieron el uso de la convolución en grupo para sortear su inconveniente de flujo de información limitado. ShuffleNet consiste principalmente en una convolución estándar seguida de pilas de unidades ShuffleNet agrupadas en tres etapas. La unidad ShuffleNet es similar al bloque ResNet, en el que se utiliza la convolución en profundidad en la capa 3x3 y se sustituye la capa 1x1 por la convolución de grupo puntual. La capa de convolución en profundidad va precedida de una operación de barajado de canales. El coste computacional de ShuffleNet puede administrarse mediante dos hiperparámetros: el número de grupos para controlar la dispersión de las conexiones y el factor de escala para manipular el tamaño del modelo. A medida que aumenta el número de grupos, la tasa de error se satura al disminuir los canales de entrada a cada grupo, lo que puede reducir la capacidad de representación. ShuffleNet superó a los modelos contemporáneos [33] [37] [63] [64] a pesar de tener un tamaño considerablemente menor. Como el único avance de ShuffleNet fue el barajado de canales, no hay ninguna mejora en la velocidad de inferencia del modelo.

MobileNetv2

Mejorando MobileNetv1, Sandler et al. propusieron MobileNetv2 [66] en 2018. Introdujeron el residuo invertido con cuello de botella lineal, un módulo novedoso para reducir los cálculos y mejorar la precisión. El módulo expande las representaciones de baja dimensión de la entrada a dimensiones superiores, filtra con una convolución en profun-

didad y luego lo proyecta de nuevo a las dimensiones inferiores, a diferencia del bloque residual común que realiza operaciones de compresión, convolución y luego expansión. MobileNetv2 contiene una capa de convolución seguida de 19 módulos de cuello de botella residual y, a continuación, dos capas convolucionales. El módulo de cuello de botella residual tiene una conexión de atajo sólo cuando el stride es 1. Para strides superiores, el atajo no se utiliza debido a la diferencia de dimensiones. También emplearon el ReLU6 como función de no linealidad, en lugar del ReLU simple, para limitar los cálculos. Para la detección de objetos, los autores utilizaron MobileNetv2 como extractor de características para una variante computacionalmente eficiente del SSD [45]. Este modelo, denominado SSDLite, afirma tener 8 veces menos parámetros que el SSD original, al tiempo que logra una precisión competitiva. Se generaliza bien en otros conjuntos de datos, es fácil de implementar y, por lo tanto, fue bien recibido por la comunidad.

ShuffleNetv2

En 2018, Ningning Ma et al. presentan un conjunto de directrices integrales para diseñar arquitecturas de red eficientes en ShuffleNetv2 [67]. Defendieron el uso de métricas directas como la velocidad o la latencia para medir la complejidad computacional, en lugar de métricas indirectas como los FLOPs. ShuffleNetv2 se basa en cuatro principios directores:

1) Igual anchura de los canales de entrada y salida para minimizar el coste de acceso a la memoria.

2) Elección cuidadosa de la convolución de grupo en función de la plataforma y la tarea objetivo.

3) Las estructuras multitrayectoria logran una mayor precisión a costa de la eficiencia.

4) Las operaciones de elementos como la suma y la ReLU son computacionalmente no despreciables.

Siguiendo estos principios, diseñaron un nuevo bloque de construcción. Divide la entrada en dos partes mediante una capa de división de canales, seguida de tres capas convolucionales que luego se concatenan con la conexión residual y pasan por una capa de barajado de canales. En el modelo de muestreo descendente, se elimina la división de canales y la conexión residual tiene capas de convolución separables en profundidad. Un conjunto de estos bloques intercalados entre un par de capas convolucionales da como resultado ShuffleNetv2. También se experimentó con modelos grandes (50/162 capas) para obtener una precisión superior con un número considerablemente menor de FLOPs. ShuffleNetv2 superó con creces a otros modelos punteros de complejidad comparable.

MobileNetV3

El núcleo de MobileNetv3 [68] es el mismo método utilizado para crear MnasNet [69] con algunas modificaciones. Se realiza una búsqueda automatizada de arquitectura neuronal consciente de la plataforma en un espacio de búsqueda jerárquico factorizado y, en consecuencia, se optimiza mediante NetAdapt [70], que elimina los componentes infrautilizados de la red en múltiples iteraciones. Una vez obtenida una propuesta de arquitectura, recorta los canales, inicializa aleatoriamente los pesos y luego la afina para mejorar las métricas objetivo. El modelo se modificó aún más para eliminar alguna capa costosa de la arquitectura y obtener una mejora adicional de la latencia. Howard et al. argumentaron que los filtros de la arquitectura son a menudo imágenes reflejadas unos

de otros, y que la precisión puede mantenerse incluso tras eliminar la mitad de estos filtros. El uso de esta técnica redujo los cálculos. MobileNetv3 utilizó una mezcla de ReLU y hard swish como filtros de activación; este último se emplea sobre todo hacia el final del modelo. Hard swish no presenta diferencias notables respecto a la función swish, pero es más barata desde el punto de vista computacional y mantiene la precisión. Para diferentes casos de uso de recursos, [68] introdujo dos modelos: MobileNetv3-Large y MobileNetv3-Small. MobileNetv3-Large se compone de 15 bloques de cuello de botella, mientras que MobileNetv3-Small tiene 11. También incluye la capa de compresión y excitación [72] en sus bloques de construcción. Al igual que [71], este modelo actúa como detector de características en SSDLite y es un 35 % más rápido que las iteraciones anteriores [64] [66], a la vez que consigue un mAP más alto.

CAPÍTULO 3
Metodología

3.1 Descripción del problema

Como ya se ha explicado, el problema que trata este trabajo consiste de la necesidad de automatizar el conteo de personas que entran y salen de un observatorio o mirador. Dado que este tipo de lugares se encuentra en zonas remotas o de difícil acceso constante, la automatización de estas tareas mediante sensores de bajo coste y eficientes es una propuesta conveniente. Generalizando, se quiere identificar el número de personas que entren o salgan de un espacio concreto, situado en algún lugar más allá de los bordes laterales de visión de la cámara. Con el fin de mantener un estándar en cuanto a estos términos, *salir* se definirá como avanzar en dirección hacia la derecha respecto la cámara. Mientras que *entrar* consistirá de avanzar en dirección hacia la derecha respecto a la cámara. Además, dado que este tipo de zonas se encuentran dentro de espacios naturales protegidos, no se pueden desplegar sensores que requieran de instalaciones complejas. Esto quiere decir que quedan excluidas a aquellas soluciones que requieran de generadores aparatosos, y que necesiten de la construcción de un cobertizo o similar para proteger el generador de las inclemencias meteorológicas o de la fauna del lugar.

Por tanto, es necesario el emplear sensores que consuman poca energía y puedan ser alimentados por una batería o panel solar fácil de colocar. A su vez, el sensor debe ser una cámara que pueda obtener el número de personas que entren o salgan, ya que el envío de imágenes de forma constante supone saturar la red con el envío de muchos paquetes de datos, causando problemas de ancho de banda. Por tanto, es necesario que el sensor pueda ejecutar un modelo de ML que sea capaz de obtener los datos requeridos para enviarlas. Como el sensor a emplear se caracteríza por tener pocos recursos se debe emplear un modelo de detección de personas que pueda desplegarse en este tipo de dispositivos.

3.2 Descripción de la solución

Dada la explicación del problema, se ha ideado una solución que emplee un dispositivo del *edge* que pueda ejecutar un modelo de ML que realice lo siguiente: el sensor debe para cada frame del video capturado, identificar las personas en la imagen. Con estas detecciones, se debe comparar las detecciones en el frame anterior y relacionar estas detecciones en el frame actual con la ubicación de las personas ya identificadas y en seguimiento en el frame anterior. Una vez se ha identificado a que persona corresponde cada ubicación actualmente, se debe comparar el desplazamiento en el tiempo para identificar la dirección de la persona y así poder contar aquellas que entran o salen. Du-

rante un periodo definido se acumulan los contadores de personas salientes o entrantes, una vez pasado el periodo, se envían los datos recopilados. El programa central de la solución, que realiza lo anteriormente explicado se puede estudiar en el apéndice B.

3.3 Diseño de la solución propuesta y proceso

En esta sección se explicará al completo el hilo de trabajo realizado. A lo largo de la investigación se han dado multiples iteraciones relacionadas al modelo de detección y dispositivo que emplear. En la siguiente figura se puede observar un esquema del funcionamiento de la solución propuesta a la que se quiere llegar.

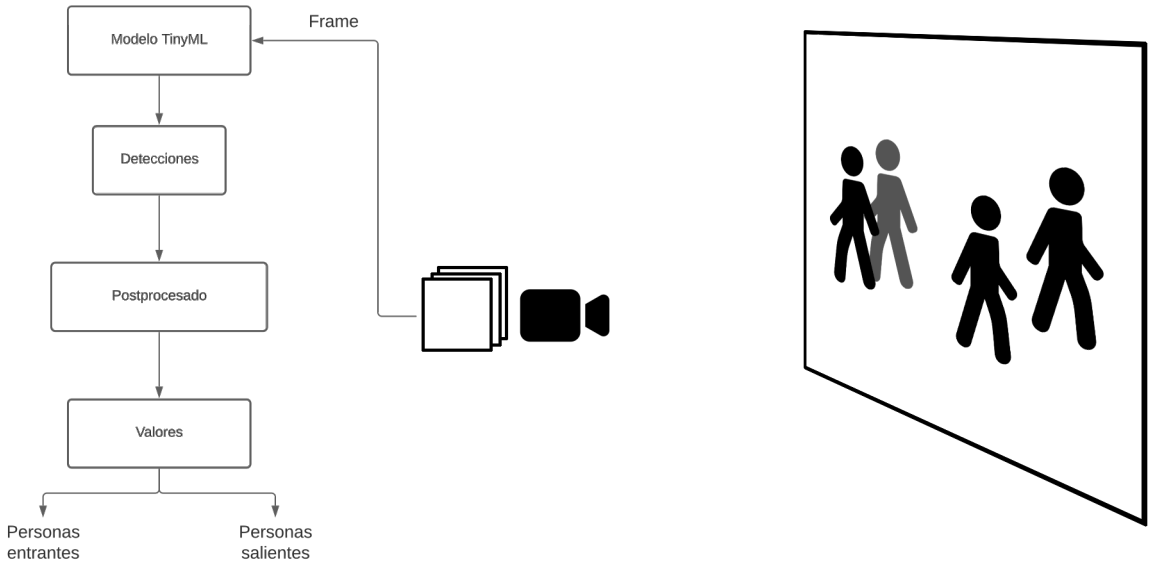

Figura 3.1: Esquema de la arquitectura de la solución

Como dispositivo inicial se pensó en emplear una cámara Nicla Vision de Arduino. Esta cámara entra en la categoría de dispositivos restringidos dada la frugal capacidad de almacenamiento de 16 MB que posee, además de la capacidad de la CPU, de hasta 480 MHz. Este dispositivo se carga mediante USB de tipo B y por su forma y tamaño consume poca batería. Estas características lo hacen idónea para funcionar como sensor remoto en el esquema de IoT ya que cuenta con conectividad inalámbrica WiFi y Bluetooth.

Su pequeño tamaño y su potente cámara de 2MP permiten que se pueda desplegar muy fácilmente en todo tipo de lugares. Solo es necesario crear una carcasa para proteger el dispositivo de las inclemencias del entorno y mantenerlo estable para que la cámara pueda desempeñar su papel, además de una fuente de alimentación, que gracias a su poco consumo de energía puede ser un panel solar o una batería de larga duración.

El modelo de detección por visión elegido fue el modelo FOMO desarrollado por EdgeImpulse [72]. Este modelo ha sido diseñado específicamente para funcionar en el *edge*. Por lo que presenta gran capacidad de análisis empleando muy pocos recursos, además de ocupar poco almacenamiento. El nombre FOMO es un acrónimo para *Faster Objects, More Objects*. Este algoritmo se basa en la premisa de detectar objetos lo más rápido posible para así poder detectar aún más objetos. El modelo en sí es muy simple, emplea las primeras capas de una red preentrenada de tipo MobileNetV2, la cuál ha

Tabla 3.1: Caracterśticas del dispositivo Arduino Nicla Vision.

Processor	STM32H747AII6 Dual Arm® Cortex® M7/M4 IC: 1x Arm® Cortex® M7 core up to 480 MHz 1x Arm® Cortex® M4 core up to 240 MHz
Sensors	2 MP Color Camera 6-Axis IMU (LSM6DSOX) Distance / Time Of Flight sensor (VL53L1CBV0FY/1) Microphone (MP34DT05)
I/O	Castellated pins with the following features: 1x I2C bus (with ESLOV connector), JTAG, Power and GPIO pin headers 1x serial port 1x SPI 2x ADC Programmable I/O voltage from 1.8-3.3V
Power	High speed USB (480Mbps) Pin Header 3.7V Li-po battery with Integrated battery charger and fuel gauge (MAX17262REWL)
Dimensions	22.86 mm x 22.86 mm
Memory	2MB Flash / 1MB RAM 16MB QSPI Flash for storage
Connectivity	Wi-Fi / Bluetooth® Low Energy 4.2 (Murata 1DX - LBEE5KL1DX-883)

obtenido obtiene una precisión cercana al modelo YOLO (de los mejores en detección de objetos) pero empleando significativamente menos parámetros. Por tanto, este modelo se puede catalogar como un modelo de TinyML, ya que es un modelo de *Aprendizaje profundo* o *Deep Learning* construido con muy pocos parámetros.

El modelo empleado en este trabajo es un modelo preentrenado e incorporado en un firmware para otro tipo de dispositivo, una OpenMV H7 Cam. Por lo que el dispositivo Nicla Vision tuvo que ser desestimado en el trabajo dado este problema de compatibilidad. Así, dado el problema de compatibilidad de la Nicla Vision, y no poder ejecutar el modelo de detección creado se contemplo el emplear el dispositivo para el que se diseñó, la OpenMV H7 [73]. Esta cámara, aun contando con poca capacidad de RAM y almacenamiento, es capaz de ejecutar algoritmos de visión y de ser programada en un lenguaje de nivel alto, como lo es Python. Además, esta cámara presenta soporte de TensorFlow Lite para microcontroladores.

En concreto, el modelo FOMO emplea las capas de la red MobileNetV2 hasta que las dimensiones de los mapas de características resultado presentan unas dimensiones ancho y alto una octava parte de las originales. Es decir, que si la imagen original es de 96x96, el mapa de calor resultante será de 12x12; si es de 320x320, el mapa de calor resultado será de 40x40. Una vez se tiene este mapa de características, este es aplicado a un clasificador estándar, el cúal no es más que una convolución de 32 filtros y la capa de salida, la cuál es un clasificador con tantos filtros como clases. Es decir, si se quiere clasificar dos tipos de objetos la salida será una matriz tridimensional en la que la anchura y altura serán la octava parte de las dimensiones de la imagen original y la profundidad (o canales) serán las clases, 2 canales, uno para cada clase. La salida, por tanto, será un mapa de calor, el que los centroides serán las detecciones. Por lo que cuanta mayor resolución y menor sea

Tabla 3.2: Caracterśticas del dispositivo OpenMV H7 Cam.

Processor	ARM® 32-bit Cortex®-M7 CPU w/ Double Precision FPU 216 MHz (462 DMIPS) Core Mark Score: 1082 (compare w/ Raspberry Pi Zero: 2060)
RAM Layout	16KB Stack 128KB .DATA/.BSS/Heap 384KB Frame Buffer/Stack (512KB Total)
Flash Layout	32KB Bootloader 96KB Embedded Flash Drive 1920KB Firmware (2MB Total)
Supported Image Formats	Grayscale RGB565 JPEG (and BAYER/YUV422)
Maximum Supported Resolutions	Grayscale: 640x480 and under RGB565: 320x240 and under Grayscale JPEG: 640x480 and under RGB565 JPEG: 640x480 and under
Lens Info	Focal Length: 2.8mm Aperture: F2.0 Format: 1/3" HFOV = 70.8°, VFOV = 55.6° Mount: M12*0.5 IR Cut Filter: 650nm (removable)

la reducción de las características, es decir, cuanto antes se corte la red MobileNetV2, más centroides serán detectables por lo que más objetos se podrán detectar.

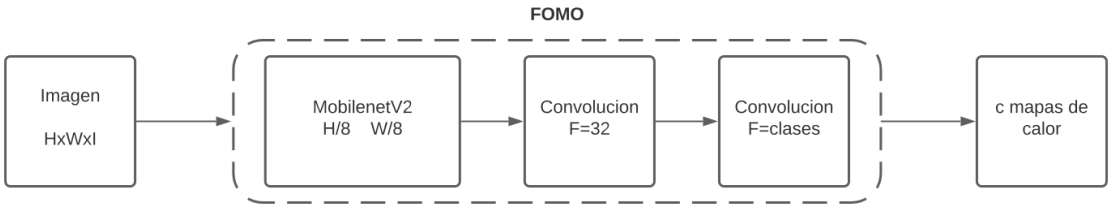

Figura 3.2: Esquema del modelo FOMO

En este caso, el modelo FOMO empleado fue un modelo ya entrenado cedido por otro grupo de investigación. El entrenamiento de este modelo se realizó mediante la herramienta de Edge Impulse, la cuál es de pago y de forma gratuita cede poco tiempo de entrenamiento. Dado esto, la precisión del modelo era aceptable, identificando correctamente los objetos necesarios, pero dado el poco tiempo de entrenamiento ocurrían inexactitudes como doble identificación de personas, al identificar tanto el tronco como las piernas por ejemplo. Además, dado que el modelo se basa en mapas de densidad, de los que se extraen centroides que indican la posición de los objetos, los objetos demasiado cercanos se identificarán como uno solo. Aunque el algoritmo, tras las detecciones del modelo, solo identifica un objeto como el mismo en el frame anterior si se encuentra a menos de 20 frames de distancia, si se daba esta duplicidad de forma continuada, el

resultado variaría de forma considerable de la realidad, ya que podrían darse multiples detecciones para el mismo objeto o una para multiples objetos, contándose varias salidas o entradas cuando solo debería haberse contado una única vez. Como se puede ver en la imagen 3.3, este problema sumado a la variación de posición entre frames, puede generar que para un mismo objeto hayan diferentes detecciones con diferente dirección identificada.

Figura 3.3: Ejemplo de imprecisión de FOMO

Dado esto se decidió que el modelo FOMO no cumplía con las necesidades de precisión, por lo que se comenzó a investigar otros modelos. Exactamente, el problema con el modelo FOMO se encontraba con la precisión a la hora de cambiar de frame, pero si fuera por la precisión en frame estático, es decir, sin tener en cuenta el cambio de frame, sería aceptable (aun con el problema surgido del uso de mapas de densidad). Por tanto, modelos que, aunque fueran ligeros, no alcanzasen una gran precisión. Esto ocasionó que los modelos de tipo *Lightweight* fueran descartados de las opciones elegibles ya que, aunque su poco peso es ideal para su uso en dispositivos restringidos, su precisión eran mucho menor de la que se estaba logrando de forma estática con el modelo FOMO. Por otro lado, los modelos basados en transformers requieren aún más datos para ser entrenados que los modelos de *Deep Learning* basados en convoluciones. Esto suponía un problema, por qué las imágenes de entrenamiento debían ser de personas de perfil, ya que la posición en la que se ha ideado que la cámara sea posicionada es de cara al perfil de los transeúntes. Las imágenes de las que se disponían eran pocas, alrededor de 800 imágenes en total entre 4 vídeos en distintos escenarios, velocidades y densidad de personas. Por esta razón, aquellos modelos basados en transformers fueron también rechazados, ya que no se disponía de los suficientes datos para el entrenamiento, y buscar los datos necesarios para el entrenamiento de este tipo de redes supondría un alto prolongado en el proyecto, lo cuál no era viable.

Esta criba dejó solo los modelos detectores de objetos basados en convoluciones en dos fases y de una única fase. Como antes se ha explicado, los detectores de dos fases presentan una arquitectura compleja, por lo que no son aptos para desplegarse en dispositivos en el *edge*, que no poseen los suficientes recursos. Este hecho inclina la balanza por completo en favor de los detectores de objetos de una única fase, por lo que ya se tiene un conjunto claro de detectores entre los que elegir. Primero, dado que existen diferentes versiones de YOLO [36], cada una mejorando ciertos aspectos de la iteración anterior, se decidió elegir una de estas versiones para representarlos.

Como se puede ver en la tabla 3.3, a cada iteración de YOLO, los parámetros aumentan, por lo que la versión 4, aunque con buena precisión, es mejor dejarla de lado en favor de las versiones anteriores. Lo mismo ocurre con la versión dos, que obtiene la mejor pre-

Tabla 3.3: Comparación entre las diferentes versiones de YOLO

	YOLO	YOLOv2	YOLOv3	YOLOv4
Parámetros	50M	67M	61M	65M
mAP (%)	63.4	77.8	63.5	64.9

cisión, pero gracias a la gran cantidad de parámetros, por lo que es descartada. Dado todo este proceso de elección entre los modelos YOLO, la decisión entre los modelos YOLO se centra entre el modelo YOLO original y el modelo YOLOv3. Al final se decidió por elegir como representante de los modelos YOLO el modelo versión tres, ya que posee una mejor precisión, y aunque sí es cierto que el tamaño es superior, es más rápido que sus anteriores iteraciones, como se puede constatar en [39].

Tabla 3.4: Comparación entre las diferentes versiones de YOLO

	SSD	EfficientDet	RetinaNet	YOLOv3
mAP (%)	52.6	51.8	57.5	63.5

En cuanto a los otros modelos, quedan los modelos SSD, EfficientDet y RetinaNet. Los tres modelos alcanzan el tiempo real, es decir, que obtienen una velocidad de más de 30 FPS. Como se puede observar en la tabla 3.4, los modelos restantes no superan en precisión a YOLOv3, por lo que este sería el mejor modelo a elegir entre en contraste a los demás. Por todo lo anterior y dado este problema con la precisión del modelo FOMO, se decidió cambiar de modelo de detección a uno que presente una precisión alta. Por esto se decidió cambiar al modelo YOLO (You Only Look Once) [39]. Este modelo es de los mejores clasificados en detección de objetos de una única fase, como se ha explicado antes. En concreto, la versión de YOLO empleada en los experimentos ha sido la versión Tiny Yolo, a su vez basada en Yolo V3.

El modelo TinyYOLO se caracteriza por emplear como base la CNN Tiny Darknet [74]. La cual es una versión de Darknet [41] que emplea solo 19 capas. Como antes se ha explicado, YOLO extrae las *bounding boxes* en tres categorías: objetos grandes, objetos medianos y objetos pequeños. Para obtener los objetos pequeños, YOLO realiza más operaciones que para obtener los medianos, por lo que en la versión Tiny se prescinde de objetos pequeños (o lejanos) para evitar aumentar la cantidad de operaciones a realizar y los parámetros de la red. Esto también tiene sentido en este proyecto, ya que la cámara se posicionará en un acceso estrecho y de lado, por lo que será necesario capturar los objetos grandes y medianos (si se da el caso de personas de baja estatura o niños). Como se puede ver en la figura 3.4, Tiny Darknet se compone de dos tipos de bloques principalmente. Los bloques CBL, compuestos de una capa convolucional, una normalización y una capa de activación. Y segundo, los bloques CBLP o bloques residuales, que añaden a los bloques CBL una capa de *pooling* al final. Tras los cinco primeros bloques se extraen los mapas de características y son aportados a las últimas capas del modelo, que las emplean para obtener los objetos de tamaño grande. Por otro lado, los objetos de tamaño medio se extraen tras realizar otra operación convolucional a la concatenación de los mapas de características empleados en los objetos grandes y el proceso anterior. La salida del modelo es un vector de dos elementos, cada uno con la información del objeto identificado.

Para cada kernel de los mapas de caracterísitcas, dependiendo de la cantidad de clases a identificar, la profundidad de la información del objeto variará, ya que para cada objeto se tiene la siguiente información que se puede ver en la figura 3.5. Como se puede observar, para cada kernel se tendrán tres *bounding boxes*, cada *bounding box* tendra $5 + c$ elementos, siendo c el número de clases a identificar. En la imagen se emplea el dataset

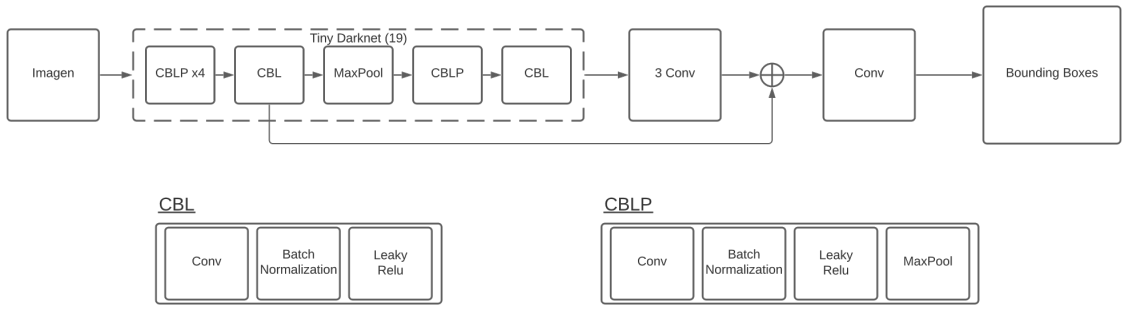

Figura 3.4: Esquema de la arquitectura del modelo TinyYOLO

de COCO, que tiene 80 clases, por lo que cada kernel tendrá hasta 255 elementos. En el caso de este trabajo solo se requiere una clase, *persona*, por lo que serán 8 elementos por kernel.

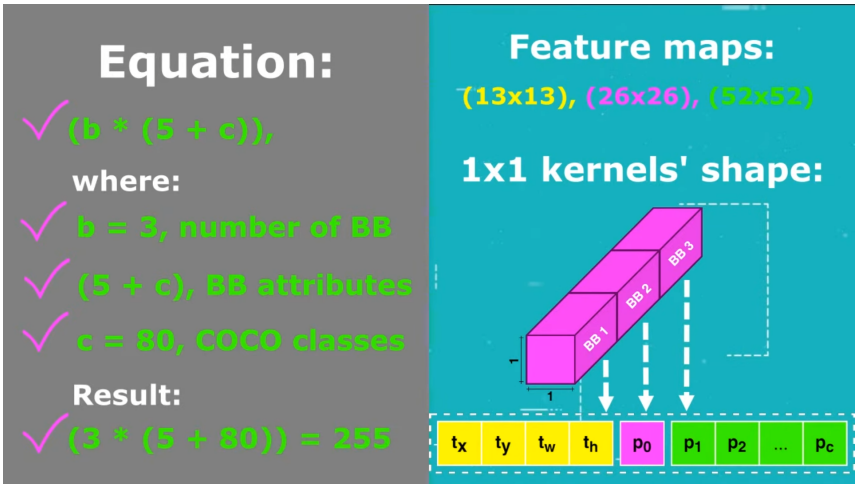

Figura 3.5: Esquema de la información de un kernel de la salida del modelo. Extraída de [1]

Una vez elegido el modelo de detección, se comenzó la fase de entrenamiento. La cual se realizó mediante programas en lenguaje Python empleando las librearías Tensorflow y Numpy. El propio programa de entrenamiento proporcionaba el mAP al proporiconarle un conjunto de prueba. Aunque este parámetro será discutido más adelante, en la sección 4, se puede adelantar que el valor de mAP para el modelo era de un 91 %. Una vez se tuvo el modelo entrenado y guardado para poder modificarlo o mejorar más adelante se comenzó la fase de compresión. Como se ha explicado antes en la sección 2, a la hora de comprimir un modelo se debe tener en cuenta que la precisión y rendimiento decaerán, por lo que se debe encontrar un equilibrio entre la compresión del modelo y el desempeño del mismo. Por suerte, Tensorflow cuenta con una librería que se especializa en compresión y creación de modelos *tiny* enfocados a ser desplegados en dispositivos restringidos o de pocos recursos, Tensorflow Lite. Para comprimir el modelo, como se puede ver en el apéndice B, se debe cargar en memoria y mediante una clase específica de la librería se definen varios parámetros, como la necesidad de optimizar el tamaño, para luego convertir el modelo a un modelo tiny. Una vez realizada esta compresión, se puede guardar el nuevo modelo, que se guarda con un formato propio de la librería: tflite.

En este momento surge otro problema, el modelo sin comprimir ocupa en disco 34MB y a la hora de comprimirlo se logra reducir este tamaño a 8MB. El problema radica este momento en el dispositivo a emplear, recordemos que la OpenMV H7 Cam posee un

almacenamiento de hasta 2MB, por lo que desgraciadamente el modelo comprimido sobrepasa por el cuádruple la capacidad de almacenamiento del dispositivo. Tras varias pruebas, y lecturas de documentación de Tensorflow Lite, y varios papers versados en compresión de modelos de ML, se llegó a la conclusión de que Tensorflow Lite ya empleaba las técnicas y herramientas que posibilitan la mayor compresión posible. Por tanto, se decide cambiar el dispositivo en el que se prueba el modelo a uno con mayores capacidades, una Jetson Orin Nano.

Tabla 3.5: Características de las Jetson Orin Nano

	Jetson Orin Nano 4GB	Jetson Orin Nano 8GB
GPU	Nvidia Ampere 512 núcleos y 16 núcleos tensor	Nvidia Ampere 512 núcleos y 16 núcleos tensor
Max GPU	625 Hz	
CPU	CPU Arm® Cortex®-A78AE 6 núcleos 64 bits	
Max CPU	1.5 GHz	
Almacenamiento	234 GB	
Memoria	LPDDR5 4 GB y 64b 34 GB/s	LPDDR5 8 GB y 128b 68 GB/s

Como se puede observar en la tabla 3.5, hay dos configuraciones de Jetson Orin Nano dependiendo de sus características. En el caso de este trabajo se ha ewmpleado la Jetson Nano de 8GB. Estos dispositivos poseen GPU, por lo que se puede emplear aceleración con la GPU. Desgraciadamente, TensorFlow Lite reestructura las redes neuronales y los métodos empleados para no ser empleados en dispositivos con GPU, sino con CPU, ya que un dispositivo con recursos limitados no posee de normal una GPU. Por lo que en los experimentos no se ha podido emplear la aceleración de la GPU de Nvidia, sino la CPU del dispositivo.

CAPÍTULO 4
Experimentos y resultados

4.1 Preparación de los experimentos

Como se explicó en el apartado de objetivos de la introducción, el proceso creado debe poderse desplegar de forma fácil en un dispositivo. Esto quiere decir que el algoritmo debe poderse integrar con el sistema del dispositivo en el que se ejecute. Esto se puede solucionar fácilmente al emplear Docker (https://www.docker.com), el cual es una herramienta muy potente a la hora de despliegue de soluciones IoT, en la que se pueden crear contenedores que pueden ser empleados en otros dispositivos sin tener en cuenta el sistema operativo del mismo. A su vez, para maximizar la velocidad del algoritmo, se ha decidido trabajar mediante paralelización gracias a la ayuda de Docker Compose, que permite gestionar diferentes imágenes al mismo tiempo. En concreto, el proceso general del algoritmo se divide en dos de forma natural. La primera parte consiste en la toma del vídeo, en la frontera física del algoritmo, que se conpartimentará en un contenedor. Este contenedor, además de ser el responsable de la inicialización de la cámara y de la toma de vídeo, guardará los frames numerados en un directorio temporal cada cierto tiempo. En cuanto al segundo contenedor, este se encargará de acceder al directorio temporal en el que se encuentran los frames, para cada frame identificar las personas en la imagen, y realizar el conteo de personas que han salido o entrado. A modo de aclaración, todo esa conpartimentación se ha realizado en base a los experimentos. En el caso de un despliege real se añadiría un tercer contenedor encargado de las comunicaciones, de enviar los datos recogidos para su consecuente almacenaje y análisis.

En los experimentos se han empleado tres vídeos pregrabados, con personas andando o montando en bici, para así poder realizar los experimentos sin tener que mover el dispositivo, ya que aun no ha sido preparado para un despliegue. Cada vídeo presenta una escena diferente, variando la iluminación, número de personas que transitan delante de la cámara, posicionamiento de la cámara y tipo de fondo y terreno, como se puede ver en la figura 4.1. El vídeo 1 muestra una escena de atardecer, con variaciones de luminosidad con las sombras y zonas iluminadas por la luz crepuscular. Estas variaciones de luminosidad pueden causar imprecisiones en las detecciones del detector de objetos, por lo que resulta útil el emplear este vídeo para evaluar el desenpeño en esta situación, ya que el lugar en el que se encuentre la cámara desplegada puede presentar claroscuros en el fondo que hagan variar la forma que la cámara captura las personas. El segundo vídeo muestra múltiples grupos de personas andando delante de la cámara. La luminosidad en la escena es neutral, no hay claroscuros y el fondo mayoritariamente se distingue perfectamente de los viandantes. A lo largo del vídeo la densidad de personas varía, por lo que este vídeo sirve para evaluar el desempeño del detector a la hora de identificar personas dentro de un grupo frente a personas individuales. En cuanto al tercer vídeo, este vuelve

a presentar un fondo que permite identidicar claramente a los viandantes, sin claroscuros a diferencia de la esquina superior izquierda con una zona oscura. La diferencia con los otros dos radica en la ubicación y ángulo de la cámara. En este caso la cámara se encuentra en una zona elevada, con una inclinación hacia el suelo, sin convertirse en una vista zenital. Esto permite evaluar el desempeño del detector al variar el ángulo, distancia y posición de la cámara, ya que el dispositivo puede ser colocado en una posición en la que se pueda tener un campo de visión óptimo de la zona de paso.

Figura 4.1: Vídeos empleados en el entrenamiento y experimentos. De izquierda a derecha: video 1, 2 y 3.

4.2 Resultados

Cada video de los empleados en los experimentos, como se ha explicado antes, presenta diferencias que permiten evaluar el detector en diferentes aspectos. Como se puede ver en la tabla 4.1, para cada vídeo se ha evaluado la velocidad con los fotogramas por segundo (FPS), el tiempo que ha tardado el análisis completo del vídeo en segundos y la mAP.

Tabla 4.1: Comparación de rendimiento del algoritmo en cada uno de los diferentes vídeos empleados.

Video	1	2	3
FPS	4.77	5.12	5.12
mAP (%)	82.4	91.3	90.8
Longitud del vídeo (s)	5	20	20
Duración del análisis (s)	18	60	64
Longitud/Duración	3.6	3.0	3.2

Se puede observar como el primer vídeo, aquel que ha sido grabado durante el atardecer y presenta variaciones en la luminosidad por las sombras, obtiene un peor desempeño. En concreto, sufre de una peor velocidad, como se puede constatar al ver los FPS a los que corre el detector. Además, la precisión que muestra se resiente, al presentar una precisión de hasta un 9 % menos que en los otros dos vídeos. Esto nos indica que el detector sufre por cambios de luminosidad bruscos o presencia de claroscuros en el entorno con los que se pueda confundir a las personas. Por tanto, sería prudente colocar el sensor en un lugar que presente poco estas características, o que en su defecto haya medidas paliativas, como colocar un foco de luz que pueda iluminar en las horas de poca luz. Otra medida cautelar sería la instalación del sensor en una zona en el que el fondo no muestre esos cambios que puedan hacer camuflarse a los viandantes.

En cuanto a los otros dos vídeos, el rendimiento del algoritmo en ambos es muy parecido, a la vez que la precisión en la detección se puede considerar casi perfecta. Dado esto, se puede afirmar que el detector no sufre, o sufre de forma mínima las aglomeraciones, con tal de que haya personas visibles. Aclarar que, obviamente, si un grupo grande o una persona muy cerca de la cámara son detectadas, estas evitarán que aquellas que se encuentren muy eclipsadas no sean detectadas. El segundo vídeo, además del rendimiento frente aglomeraciones, permite evaluar cómo se comporta frente a objetos que se muevan a diferentes velocidades, ya que las personas no andamos a la misma velocidad. Habrá quienes anden a menos velocidad que otras, causando solapamientos. Esto puede causar un solapamiento entero en el frame, o a medias, pero no afecta a la precisión.

El tercer vídeo permite evaluar si el rendimiento del detector se ve modificado por un cambio en la posición e inclinación de la cámara. Como se puede observar, la precisión se ve reducida, pero no significativamente, ya que este cambio genera una pérdida de menos del 1 %. Esto nos permite aclarar que, dentro de un margen, el detector continuará desempeñando un rendimiento correcto. Esto quiere decir que la cámara, con tal de encontrarse en una posición y dirección en las que los viandantes pasen de lado, será posible detectar correctamente a las personas en la imágen. Claro está, cambios muy significativos en la dirección y posicionamiento empeoraran el resultado. Por ejemplo, cuanto mayor el ángulo de visión se parezca a una vista zenital, peor será la detección. Igual pasará con ángulos en los que los viandantes crucen el campo de visión de la cámara de frente o espaldas.

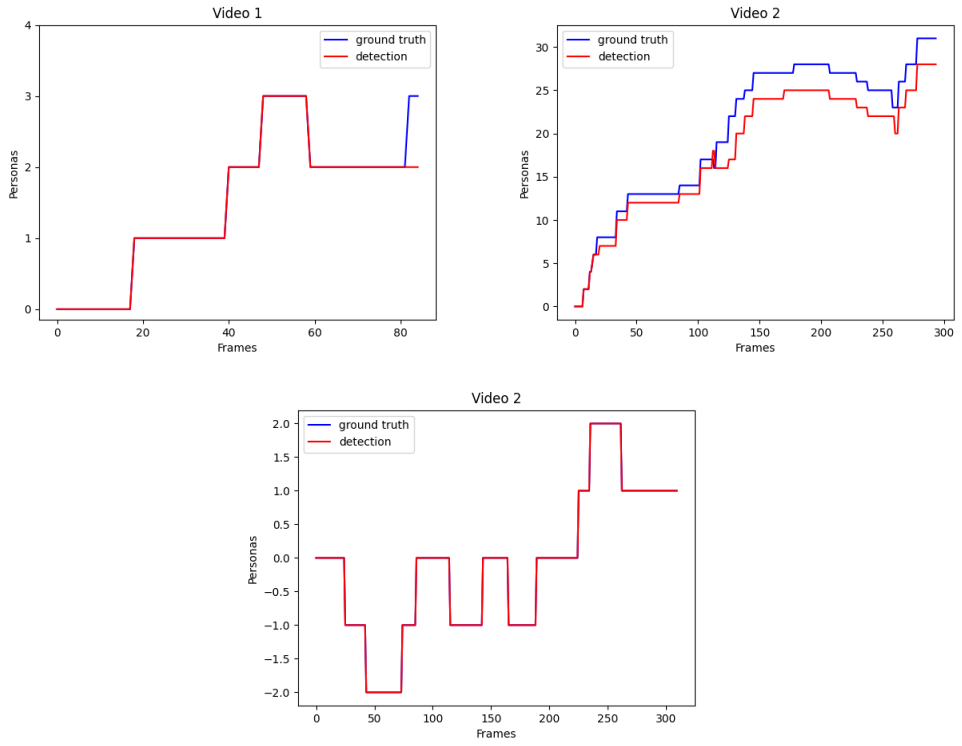

Figura 4.2: *Ground truth* y detecciones de entradas y salidas para cada vídeo.

La figura 4.2 muestra las tres gráficas de las entradas y salidas en los tres vídos. La línea roja representa la salida del algoritmo, mientras la azul la realidad. Como se puede observar, la detección de entradas y salidas no resulta perfecta, pero afortunadamente obtiene una desviación siempre de una persona, por lo que no es una error significativo.

En cuanto a la gráfica del vídeo 1, se ve cómo a pesar de la pequeña bajada de precisión en la detección de personas, la detección de entradas y salidas es muy precisa la

mayor parte del tiempo. Esto se debe mayormente a que en el vídeo siempre aparece una única persona en la pantalla, por lo que no puede haber un desajuste más allá saltos en la detección entre frames por la imprecisión con las sombras. Es decir, como no ha habido solapamientos, es posible que la imprecisión en los últimos frames se deba a que el detector no ha sido capaz de identificar a la persona y la última ubicación se encontraba fuera de la zona en la que cuenta un objeto como finalizando la acción de salir o entrar.

Por otro lado, observando el gráfico correspondiente al vídeo dos, se puede confirmar el caso contrario. A pesar de haber alcanzado una precisión de detección de personas muy alta, a la hora de detectar las salidas y entradas de personas individuales dentro de una multitud resulta menos preciso. Aun así, siempre se da una variación de una persona, por lo que no llega a ser una falta de precisión realmente significativa. Se debe tener en cuenta qué el lugar en el que se plantea desplegar el sensor no recibe gran afluencia de personas, por lo que si se da el caso de un grupo, con gran probabilidad no será tan voluminoso como para generar imprecisiones grandes. Además, por la misma razón la probabilidad de la aparición de grupos es reducida.

Todo lo contrario es el caso del vídeo 3. En este se alcanza una precisión absoluta. Cabe recalcar que en el vídeo no se dan solapamientos de personas ni multitudes. Siendo que las personas que transitan delante de la cámara siempre lo hacen en la misma dirección y a cierta distancia. A esto se suma el poco número de personas, siendo solo dos, y la idoneidad del entorno, permitiendo que no se den fallos de detección de las personas a cada frame. Se podría decir que el vídeo tres puede funcionar como muestra modelo para identificar si el algoritmo funciona correctamente.

CAPÍTULO 5
Conclusiones

Haciendo un breve resumen, en este trabajo se ha desarrollado un detector de objetos que permite detectar en una imagen personas. Con especial énfasis aquellas que se encuentren de perfil a la cámara, significando que se encuentran atravesando el campo de visión de la misma de lado a lado. A su vez, este detector ha alcanzado una alta precisión en las muestras empleadas para evaluar el modelo. Por otro lado, el detector es empleado como engranaje en un algoritmo de *Machine Learning* que permite contar las personas que han entrado o salido en un periodo de tiempo de un lugar, teniéndose como salir o entrar la derecha e izquierda de la cámara. En relación a esto, el algoritmo diseñado alcanza una precisión aceptable.

Desgraciadamente no se ha logrado el objetivo de que el algoritmo alcance una velocidad de tiempo real, que sería de 30FPS, sino que se han logrado unos pocos 5FPS, una sexta parte. Para evitar el problema del tiempo real se ha decidido no realizar el análisis completamente a tiempo real, sino que se realiza a intervalos de tiempo definido. Se acumulan frames y se analizan al finalizar el intervalo. Esta modificación se puede realizar sin decrementar el rendimiento de la solución debido a las características del lugar de despliegue, ya que la probabilidad de afluencia de multitudes es muy baja. A su vez, uno de los objetivos se enfocaba en que el modelo de detección empleado sea de la categoría de TinyML, es decir, que pueda ser empleado en un dispositivo de recursos limitados. Desgraciadamente no se ha logrado este objetivo, ya que el modelo empleado presenta un tamaño de 8MB, lo que lo hace imposible de ser desplegado en este tipo de dispositivos, que como mucho alcanzan a tener 2MB de almacenamiento. El modelo debería pesar como mucho un megabyte si se quisiera poder emplear, ya que hay que tener en cuenta que además del modelo detector se requeriría almacenar elementos del firmware y el script necesario para que el modelo funcione. Por tanto, el modelo de detección de objetos desarrollado no puede ser llamado como un modelo de TinyML, sin embargo, si puede ser catalogado cómo *Edge Computing*, ya que es capaz de ser desplegado en un dispositivo que puede desplegarse en un sistema IoT como sensor.

Por otro lado, se han podido cumplir objetivos como el de obtener un modelo detector con alta precisión y que el algoritmo creado logre identificar de forma aceptable las entradas y salidas. Como se ha explicado en la sección 4, el algoritmo muestra un buen rendimiento trabajando con multitudes y entornos con cambios de intensidad y luminosidad que puedan generar errores en la detección. Además, en un caso ideal, lo cual se dará en la inmensa mayoría de los casos, la identificación de salidas y entradas es cercana a la perfección.

También se ha logrado otro de los objetivos, el proceso debe poderse desplegar de forma sencilla en los dispositivos que puedan almacenarlo, para ello empleándose Docker. Prueba del despliegue de forma sencilla es el empleo en el dispositivo empleado en los

experimentos, ya que no ha sido en el mismo donde se ha desarrollado el algoritmo y el detector.

5.1 Problemas surgidos en el trabajo

A lo largo del proyecto se han encontrado múltiples problemas que ya han sido explicados anteriormente. Uno de ellos era la imprecisión del modelo FOMO [72]. En concreto, el problema radica no en el modelo, sino en la poca capacidad de entrenamiento que hubo. Esto es dado que el modelo FOMO se podía generar y entrenar en le herramienta Web Edge Impulse. Desgraciadamente, el entrenamiento del modelo presenta una barrera de pago, no pudiendo entrenar más de 40 ciclos o 20 minutos. Este hecho resultó en un gran problema, ya que Edge Impulse proporcionaba tanto optimización del modelo como una herramienta de despliegue del modelo optimizada para dispositivos pertenecientes al *edge*.

Un problema claro, e intrínseco del área a la que este trabajo aspiraba a proporcionar avances, el campo del TinyML, es el hecho de la limitación de recursos de los que disponen los dispositivos empleados en este campo. Para diseñar un modelo que pueda ser categorizado como TinyML hay que tener en muy en cuenta el tamaño y uso de memoria del mismo, ya que el sensor al que sea desplegado contará con poca capacidad de almacenamiento, memoria y capacidad computacional. Además el uso de técnicas de *Deep Learning*, cómo las redes convolucionales, normalmente requieren de gran cantidad de recursos para poder proporcionar un buen rendimiento, lo cual hace especialmente difícil el emplear TinyML con imágenes en dispositivos de recursos limitados.

Y por último, uno de los mayores problemas a la hora de entrenar el modelo de detección ha consistido en encontrar imágenes que pudieran emplearse, ya que se requería de imágenes muy concretas. Siendo las imágenes necesitadas, aquellas en las que hubieran personas de cuerpo completo en un plano medio de perfil a la cámara, ya que de esta forma se podría identificar más fácilmente la dirección a la que se desplazaban y definir si entraban o salían. Normalmente, las imágenes empleadas en el entrenamiento de redes de detección son muy específicas o muy generales. Además, las imágenes de personas en un plano medio se toman de frente, o en el caso de espaldas, con la persona mirado a cámara o siendo una situación específica. Ocasionando esto que las imágenes necesarias para el entrenamiento correcto del modelo de detección fueran difíciles de conseguir, suponiendo un hiato en el proyecto.

CAPÍTULO 6
Trabajos futuros

Como futuros trabajos a realizar, basados en el proyecto realizado, se podría evaluar el rendimiento de los modelos *lightweight*, que fueron desestimados por falta de tiempo. Entre estos se encuentra el modelo FOMO. Un trabajo a poder realizar sería el entrenarlo fuera de la herramienta Edge Impulse, para luego aportar el modelo entrenado a la herramiento por medio de la API y así poder hacer uso del servicio de optimización y despliegue del que dispone. En este aspecto, se podrían evaluar los otros modelos *ightweight* mentados en el estado del arte en la sección 2. Por supuesto, ya que es el objetivo que no se ha podido cumplir, múltiples trabajos que se pueden realizar consistirían en buscar un modelo de detección que pueda entrar dentro de la categoría de TinyML. En base al otro objetivo no alcanzado, se podrían realizar trabajos que intentasen mejorar el rendimiento de la solución propuesta, o una derivada de la misma, con el fin de alcanzar una velocidad de ejecución de al menos 30FPS y así alcanzar un rendimiento en tiempo real.

Otros trabajos que se podrían realizar consistirían, no en rellenar los vacíos dejados por este trabajo, sino en probar sus límites. Un ejemplo sería un trabajo que buscara determinar cual es el límite a cómo de desviado puede encontrarse el campo de visión respecto al perfil de las personas. Es decir, como de cercana la cámara puede estar a una vista zenital o una de frente sin perder precisión.

Bibliografía

[1] V. Sichkar. Introducción a yolo v3'. Youtube. [Online]. Available: https://youtu.be/vRqSO6RsptU?si=kDTPsIv5doD4T4se

[2] L. M. Amaya Fariño, A. R. Tumbaco Reyes, E. T. Roca Quirumbay, T. Villón González, B. M. Mendoza Morán, and Á. D. R. Reyes Quimís, "El iot aplicado a la domótica," 2020.

[3] L. A. Luengas, M. F. Díaz, and M. Castellanos, "Domótica para asistir adultos mayores," *Ingenio Magno*, vol. 10, no. 1, pp. 79–88, 2019.

[4] L. Lipper, P. Thornton, B. M. Campbell, T. Baedeker, A. Braimoh, M. Bwalya, P. Caron, A. Cattaneo, D. Garrity, K. Henry *et al.*, "Climate-smart agriculture for food security," *Nature climate change*, vol. 4, no. 12, pp. 1068–1072, 2014.

[5] G. Halegoua, *Smart cities*. MIT press, 2020.

[6] Cisco, "Cisco global cloud index: forecast and methodology, 2016-2021," 2021.

[7] A. B. staff, "Tinyml brings ai to smallest arm devices," Aug 2021. [Online]. Available: https://www.arm.com/blogs/blueprint/tinyml

[8] J. MSV, "How tinyml makes artificial intelligence ubiquitous," Nov 2020. [Online]. Available: https://www.forbes.com/sites/janakirammsv/2020/11/03/how-tinyml-makes-artificial-intelligence-ubiquitous/?sh=914103876227

[9] C. Guleria, K. Das, and A. Sahu, "A survey on mobile edge computing: Efficient energy management system," in *2021 Innovations in Energy Management and Renewable Resources (52042)*. IEEE, 2021, pp. 1–4.

[10] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University-Computer and Information Sciences*, 2021.

[11] V. Archana, "Tinyml for solar panels: Bringing edge computing applications to solar energy systems," Master's thesis, National Uiversity of Singapore, 2020.

[12] J. Morales-García, A. Bueno-Crespo, R. Martínez-España, J.-L. Posadas, P. Manzoni, and J. M. Cecilia, "Evaluation of edge computing platforms through tinyml workloads," this is a preprint; it has not been peer reviewed by a journal.

[13] W. Raza, A. Osman, F. Ferrini, and F. D. Natale, "Energy-efficient inference on the edge exploiting tinyml capabilities for uavs," *Drones*, vol. 5, no. 4, p. 127, 2021.

[14] M. Z. M. Shamim, "Tinyml model for classifying hazardous volatile organic compounds using low-power embedded edge sensors: Perfecting factory 5.0 using edge ai," *IEEE Sensors Letters*, vol. 6, no. 9, pp. 1–4, 2022.

[15] N. N. Alajlan and D. M. Ibrahim, "Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications," *Micromachines*, vol. 13, no. 6, p. 851, 2022.

[16] M. Holmgren and E. Holmér, "Wireless beehive monitoring: Using edge computing and tinyml to classify sounds," 2022.

[17] M.-K. Tsai, "1.2 cloud 2.0 clients and connectivity — technology and challenges," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 15–19.

[18] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I.

[19] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893 vol. 1.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[21] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, pp. 261–318, 2020.

[22] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[24] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[27] ——, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[28] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[29] Y. Hu, A. Huber, J. Anumula, and S.-C. Liu, "Overcoming the vanishing gradient problem in plain recurrent networks," *arXiv preprint arXiv:1801.06105*, 2018.

[30] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[32] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

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

[35] H. He, H. Xu, Y. Zhang, K. Gao, H. Li, L. Ma, and J. Li, "Mask r-cnn based automated identification and extraction of oil well sites," *International Journal of Applied Earth Observation and Geoinformation*, vol. 112, p. 102875, 2022.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[38] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[39] ——, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[40] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[41] J. Redmon. [Online]. Available: https://pjreddie.com/darknet/

[42] Z. Zhang, T. He, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of freebies for training object detection neural networks," 2019.

[43] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[44] D. Misra, "Mish: A self regularized non-monotonic activation function," *arXiv preprint arXiv:1908.08681*, 2019.

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[46] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2147–2154.

[47] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[48] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[49] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[52] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[53] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[54] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[55] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[56] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[57] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[58] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.

[59] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[60] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[61] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International conference on machine learning*. PMLR, 2015, pp. 2285–2294.

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

[63] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and<0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[64] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[65] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[66] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[67] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[68] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[69] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.

[70] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[71] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[72] [Online]. Available: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices

[73] [Online]. Available: https://openmv.io/products/openmv-cam-h7

[74] J. Redmon. [Online]. Available: https://pjreddie.com/darknet/tiny-darknet/

APÉNDICE A
Objetivos de desarrollo sostenible

El día 25 de septiembre de 2015, mediante la Organización de Naciones Unidas, los líderes de todo el mundo decidieron adoptar un conjunto de objetivos diseñados para modificar o erradicar problemas sociales y culturales que azotan nuestra sociedad, como son la pobreza, desigualdad, el cambio climático o la contaminación. Dichos objetivos se plantearon con el fin de proteger el la Tierra y la humanidad, para hacer posible la convivencia entre nosotros y nuestro planeta, junto todos los seres vivos que lo pueblan. Todos los objetivos definidos en esta cumbre se construyeron con metas específicas a alcanzar antes de la década de 2030. A continuación se encuentran listados e indicando el grado de relación con el presente trabajo.

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

Reflexión sobre la relación del TFG con los ODS y con los ODS más relacionados.

ODS 7. Energía asequible y no contaminante. Este proyecto se enfoca diseñar un sistema eficiente energéticamente. Por tanto, creo que el proyecto se relaciona de forma directa y clara con este objetivo, dado que el empleo de dispositivos que empleen energía de forma eficiente participan del uso de energía sostenible

> **7.b** De aquí a 2030, ampliar la infraestructura y mejorar la tecnología para prestar servicios energéticos modernos y sostenibles para todos en los países en desarrollo, en particular los países menos adelantados, los pequeños Estados insulares en desarrollo y los países en desarrollo sin litoral, en consonancia con sus respectivos programas de apoyo.

ODS 8. Trabajo decente y crecimiento económico. Uno de los impactos esperados de este trabajo se centra en el turismo y la economía. En concreto, lugares como reservas naturales o puntos de interés turísitco requieren de mano de obra para ser cuidados. Este trabajo busca automatizar tareas que requieren de mano de obra desplazada en terreno remoto y aumentar la economía. Buscando a su vez un turismo sostenible.

> **8.2** Lograr niveles más elevados de productividad económica mediante la diversificación, la modernización tecnológica y la innovación, entre otras cosas centrándose en los sectores con gran valor añadido y un uso intensivo de la mano de obra.
>
> **8.3** Promover políticas orientadas al desarrollo que apoyen las actividades productivas, la creación de puestos de trabajo decentes, el emprendimiento, la creatividad y la innovación, y fomentar la formalización y el crecimiento de las microempresas y las pequeñas y medianas empresas, incluso mediante el acceso a servicios financieros.
>
> **8.9** De aquí a 2030, elaborar y poner en práctica políticas encaminadas a promover un turismo sostenible que cree puestos de trabajo y promueva la cultura y los productos locales.

ODS 9. Industria, innovación e infraestructura. La infraestructura de Internet de las Cosas busca ser eficiente tanto energéticamente como en resultados. Como parte de una infraestructura IoT, creo que este proyecto se relaciona a este objetivo por el uso de energía limpia y sostenible que emplean todos los dispositivos de IoT.

> **9.4** De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas.

APÉNDICE B
Código empleado

B.1 Pseudocódigo del programa principal

```
Inicializacion de variables como

first_frame como 0
center_point_prev como lista vacia
person_tracker como diccionario vacio
tracking_person_stats como diccionario vacio
tracking_label como lista vacia
not_detected como diccionario vacio
track_id como 0

Inicializar enter_person como 0
Inicializar enter_bike como 0
Inicializar exit_person como 0
Inicializar exit_bike como 0

Cargar modelo de deteccion

Mientras Verdadero:
    Capturar imagen del sensor

    Inicializar center_point_current como lista vacia
    Incrementar first_frame por 1

    Para cada deteccion d en lista_detecciones:
        Obtener coordenadas (x, y, ancho, alto) de la deteccion
        Calcular coordenadas del centro (center_x, center_y) de la deteccion
        Agregar [center_x, center_y, labels[i]] a center_point_current

    Si first_frame es menor o igual a 2:
        Para cada persona_actual en center_point_current:
            Para cada persona_previa en center_point_prev:
                Calcular diferencias distance_x y distance_y entre coordenadas
                    de persona_previa y persona_actual
                Si las diferencias estan en un rango pequeno y not_detected[
                    track_id] es menor a 20:
                    Agregar persona_actual al diccionario person_tracker con
                        clave track_id
                    Agregar entrada vacia para track_id en
                        tracking_person_stats
                    Agregar 0 para track_id en not_detected
                    Incrementar track_id por 1
    Sino:
        Copiar person_tracker a person_tracker_copy
        Copiar center_point_current a center_point_current_copy
```

```
        Para cada object_id, persona_previa en person_tracker_copy:
            persona_detectada = Falso

            Para cada persona_actual en center_point_current_copy:
                Calcular diferencias distance_x y distance_y entre coordenadas
                    de persona_previa y persona_actual

                Si las diferencias estan en un rango pequeno y not_detected[
                    object_id] es menor a 20:
                    Actualizar coordenadas de object_id en person_tracker con
                        coordenadas de persona_actual
                    Reiniciar contador not_detected[object_id] a 0
                    Marcar persona_detectada como Verdadero

                    Si la persona esta en la zona de entrada o salida:
                        Actualizar estadisticas de seguimiento de persona segun
                            su direccion
                    Si la persona_actual esta en center_point_current,
                        eliminarla de la lista

                Sino:
                    Incrementar not_detected[object_id] por 1

            Si persona_detectada es Falso:
                Si not_detected[object_id] es mayor o igual a 20:
                    Eliminar object_id de person_tracker, tracking_person_stats
                        y not_detected
                Sino:
                    Incrementar not_detected[object_id] por 1

        Para cada persona_actual en center_point_current:
            Agregar persona_actual a person_tracker con clave track_id
            Agregar entrada vacia para track_id en tracking_person_stats
            Agregar 0 para track_id en not_detected
            Incrementar track_id por 1

    Actualizar center_point_prev con copia de center_point_current

    Enviar contadores
```

B.2 Compresión del modelo ML

```
import tensorflow as tf
import tensorflow_model_optimization as tfmot

saved_model_dir = 'saved_models\custom_yolo_tiny_v2'
model = tf.saved_model.load(saved_model_dir)
#model = tf.keras.models.load_model(saved_model_dir)


# Convert the model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
                                       tf.lite.OpsSet.SELECT_TF_OPS]
converter.allow_custom_ops=True

tflite_model = converter.convert()

# Save the model.
with open('saved_models/trained_samall_v2.tflite', 'wb') as f:
```

```
f.write(tflite_model)
```

Código B.1: Script de compresión del modelo ML a tiny