Universidad Nacional
de Educación a Distancia

Universidad
Complutense de Madrid

Escuela Técnica Superior de
Ingeniería Informática

Facultad de Informática

Estudio de la detección de Personas con YOLO para la mejora de un sistema de alarmas de exterior en un instituto

Pablo Cancer Casado

Director: Gonzalo Pajares Martinsanz

Trabajo de Fin de Máster

Máster Universitario
en Ingeniería de Sistemas y Control

[Curso 2023/2024 /Junio]

TÍTULO DEL TRABAJO

MASTÉR EN INGENIERÍA DE SISTEMAS Y DE CONTROL

PROYECTO TIPO B:

Autor: Pablo Cancer Casado

Director: Gonzalo Pajares Martinsanz

Convocatoria: Junio 2024

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Firma del alumno

Resumen

El propósito principal de este proyecto es llevar a cabo un análisis exhaustivo de un modelo de detección de objetos, con varias de sus variantes, mediante técnicas avanzadas de aprendizaje profundo, con el objetivo de mejorar el sistema de videovigilancia de un instituto, que constituye el objetivo final. Se realiza un estudio detallado de diversas arquitecturas de redes neuronales, centrándose específicamente en los modelos YOLOv5, YOLOv8 y YOLOvNAS, con el fin de evaluar tanto su precisión como su tiempo de procesamiento. Por tanto, se trata de un estudio conceptual de viabilidad como paso previo a su implantación en el entorno real previsto.

El desarrollo de los detectores se lleva a cabo utilizando el lenguaje de programación Python, aprovechando las bibliotecas de Ultralytics y DECI AI. Se emplean conjuntos de datos relevantes como COCO2020 y Open-Image v7. Todo el análisis se realiza utilizando hardware local para garantizar la viabilidad y la eficiencia del proyecto.

Para determinar qué modelo puede ser utilizado en futuras mejoras del sistema de videovigilancia, se lleva a cabo un análisis comparativo de los modelos YOLOv5, YOLOv8 y YOLOvNAS. Este análisis abarca la evaluación del rendimiento de los modelos en imágenes nítidas, así como en imágenes y vídeos obtenidos de cámaras de seguridad, y también se evalúa su comportamiento en tiempo real.

Además, se establece una comunicación bidireccional entre la aplicación Python y el hardware de videovigilancia mediante Node-RED. Este hardware incluye actuadores de iluminación y sonido controlados por un Relé programable Siemens Logo8.

Los resultados obtenidos en este trabajo proporcionan información crucial sobre la viabilidad de este desarrollo para su futura implementación y puesta en marcha en un entorno de videovigilancia real del instituto.

Palabras clave:
Detección de objetos, redes neuronales convolucionales, aprendizaje profundo, YOLOV5, YOLOV8, YOLONAS, NODE-RED, LOGO8.

Abstract

The goal of this project is to exhaustively analyze the performance of an object detection model (and several of its variants) for the video surveillance system of a building. Therefore, this is a feasibility study prior to its deployment in the real world. Specifically, I evaluate the accuracy and processing time of three neural network architectures: YOLOv5, YOLOv8 and YOLOvNAS models.

The models are implemented in Python, using the Ultralytics and DECI AI libraries, and tested on two datasets: COCO2020 and Open-Image v7. All analyses are carried out locally to ensure the viability and efficiency of the project. In addition, a two-way communication is established between the Python application and the video surveillance hardware using Node-RED. This piece of hardware includes lighting and sound actuators controlled by a Siemens Logo8 programmable Relay.

I compare YOLOv5, YOLOv8 and YOLOvNAS to determine which model can be used in future improvements to the video surveillance system. I evaluate the performance of the models in sharp images, as well as in images and videos obtained from security cameras. Their behaviour is also evaluated in real time. The results obtained in this work provide crucial information on the feasibility of this solution for its future implementation for video surveillance in the building.

The results obtained from this work provide crucial information about the feasibility of this development for its future implementation and deployment in a real surveillance environment at the institute.

Keywords: Object detection, convolutional neural networks, deep learning, YOLOV5, YOLOV8, YOLONAS, NODE-RED, LOGO8.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

El objeto de este proyecto es la realización de un estudio de las diferentes versiones y modelos de YOLO, una técnica muy avanzada para detección de objetos dentro del paradigma de lo que se conoce como Aprendizaje Profundo dentro de la Inteligencia Artificial, profundizando en su arquitectura, precisión, tiempo de ejecución y adaptabilidad. Para obtener las distintas características de las diferentes versiones el análisis se enfoca hacia un caso práctico como es la detección de personas en tiempo real de una cámara de seguridad y su comunicación con otros sistemas para aumentar la disuasión y realizar el aviso de intrusión. Dentro de los diferentes sistemas de detección de objetos, *You Only Look Once* (YOLO) ha destacado como un enfoque revolucionario para la detección de objetos en tiempo real, por lo que es el elegido para la realización de este trabajo. En base a esto, el propósito general del proyecto es la integración de estos modelos en un sistema de videovigilancia de un Instituto de enseñanza.

YOLO, con sus diversas versiones evolutivas, tales como YOLOv3, YOLOv5, YOLOv8 y YOLONas, ha desempeñado un papel crucial en la evolución de la detección de objetos. Cada iteración ha introducido mejoras sustanciales en la precisión y eficiencia del modelo, abordando desafíos específicos y adaptándose a las demandas cambiantes de la visión computacional. En este trabajo se propone explorar y comparar detalladamente las distintas versiones de YOLO, analizando sus características, rendimiento en una aplicación específica, la posibilidad de comunicación con otros sistemas. A través de esta investigación, se busca proporcionar una comprensión integral de la evolución de YOLO y su contribución al campo de la visión artificial, con especial énfasis en YOLOv5, YOLOv8 y YOLONas.

El resultado final del trabajo busca crear un prototipado para la mejora de un sistema de videovigilancia con alarmas de disuasión y reconocimiento de personas para evitar falsas alarmas de la zona exterior de un instituto. Además, la detección de personas permitirá evitar las falsas alarmas que se producen por los sensores de detección de presencia. Ya que el instituto tiene una gran zona exterior donde suele haber transcurso de pequeños animales, como gatos o pájaros. Por falta de la adquisición del hardware de cámaras y de sistema de videovigilancia correcto para realizar las pruebas, el análisis de los detectores de objetos realizado se lleva a cabo a través de imágenes y videos que permitan realizar la simulación pertinente con el fin de escoger el software y el modelo más apropiado para su implantación futura.

La justificación de un sistema como el previsto se justifica por la ausencia de personal de vigilancia por la noche y la falta de recursos económicos para su contratación, lo que hace necesaria la implantación de un sistema de alarmas que mejore al actual y solo se active cuando hay una intrusión de personas. Pudiendo el usuario establecer los parámetros de la alarma, en este sentido se podría ajustar de forma a que solo actuara cuando se detectan dos o más personas, debido a que la mayoría de los hurtos que se han producido han sido realizados por más de una persona.

1.1 Características del centro

El instituto para el cual se va a realizar el análisis es el recinto de los centros del IES y el CPIFP Bajo Aragón de Alcañiz, Teruel. Ambos constan de un recinto de gran zona exterior que no se encuentra vigilada por personal y sin detectores externos. La gran zona exterior se observa delimitada en azul en la Figura 1.

Figura 1 Área Instituto Bajo Aragón, Alcañiz

Todos los edificios constan de alarmas privadas con mensaje en el móvil, que llega al equipo directivo, pero debido a la gran cantidad de falsos avisos, hace que estos no se tengan en cuenta por parte de la persona encargada del sistema de alarmas y se hayan producidos hurtos en el

recinto. Por ello, este trabajo va dirigido a la mejora para una posible futura implantación de la mejora del sistema de alarmas. El cual, en el momento que detecte una o varias personas por medio de unas cámaras de seguridad, se enciendan las luces exteriores y suene un zumbador para alertar de presencia de personas en el entorno.

1.2 Antecedentes

Fruto de los hurtos producidos en el instituto, a pesar de haberse activado la alarma, se ha constatado frecuentemente la falta de confirmación de una intrusión real y la ausencia de mecanismos de disuasión. Por tal motivo, en este trabajo se opta por la realización de un estudio conceptual y su viabilidad para la implantación de un sistema que utilice la visión artificial para la mejora del sistema.

La visión artificial, como disciplina de la inteligencia artificial, ha experimentado un ascenso notable desde sus inicios. Surgió y se desarrolló en las décadas de 1960 y 1970 e incluso anteriores, marcando sus primeros pasos con el reconocimiento de patrones y la interpretación de imágenes (Infaimon, s.f.). A lo largo de los años, el desarrollo de algoritmos más sofisticados y el acceso a conjuntos de datos masivos han catalizado su progresión exponencial., (Marr, 1982).

En la última década, el auge de la visión artificial ha sido extraordinario, impulsado por avances significativos en el aprendizaje profundo y las redes neuronales convolucionales (CNN, Convolutional Neural Networks). Este período ha presenciado un cambio de paradigma, llevando la visión artificial más allá de la mera identificación de patrones para abordar tareas complejas como la detección de objetos en tiempo real. El aprendizaje profundo ha permitido la creación de modelos más sofisticados y adaptables, mientras que las CNN han desempeñado un papel fundamental al capturar relaciones espaciales y extraer características cruciales en imágenes. Este conjunto de tecnologías ha impulsado la visión artificial hacia un nuevo nivel de precisión y versatilidad, consolidándola como una herramienta indispensable en diversas aplicaciones prácticas.

La evolución de la visión artificial ha llevado el desarrollo de aplicaciones de la detección de objetos son diversas, desde la seguridad con sistemas de vigilancia avanzados hasta la conducción autónoma, la medicina con la identificación de patologías en imágenes médicas, y el comercio electrónico mediante la mejora de la experiencia de compra con sistemas de

recomendación basados en la detección de productos. Estas aplicaciones ilustran la versatilidad y el impacto de la detección de objetos en diferentes campos.

Como se menciona en Pajares et al (2021) entre los sistemas de detección de objetos que han contribuido a este panorama, se destacan Faster R-CNN, SSD (*Single Shot MultiBox Detector*), y R-FCN (*Region-based Fully Convolutional Networks*), YOLO. Estos sistemas han establecido hitos en términos de precisión y eficiencia, cada uno aportando enfoques únicos a la detección de objetos.

Este contexto histórico establece el marco para la evaluación y comparación de las versiones evolutivas de YOLO en este trabajo de fin de máster. Exploraremos cómo YOLO, a lo largo de sus iteraciones, se ha integrado y competido en este escenario dinámico de la visión artificial, contribuyendo a su continua evolución.

1.3 Objetivos

Los objetivos marcados para la realización de este trabajo son los siguientes:

- Estudio y comprensión de las redes neuronales convencionales (CNN).
- Estudio y compresión de la arquitectura YOLO para la detección de objetos.
- Utilización y estudio de diferentes Datasets (COCO, Open Images V7 Dataset) para crear los diferentes detectores.
- Desarrollo, implementación y análisis de detectores utilizando las diferentes versiones de YOLO.
- Toma de resultados.
- Estudio de herramientas de comunicación.
- Comunicación de un autómata con la detección de objetos.
- Resultado final.

1.4 Tareas realizadas

Para lograr los objetivos pretendidos en la realización de este Proyecto se tuvo que disgregar y analizar cada uno de ellos en diferentes tareas. Al partir desde un enfoque teórico que diese un marco sólido para la parte práctica, las tareas planteadas se realizaron en este orden:

- Lectura técnica sobre redes neuronales convolucionales y los mecanismos internos de generación de resultados.

- Lectura, análisis y síntesis de las redes convolucionales de las distintas arquitecturas de YOLO.
- Estudio y generación de código de las distintas arquitecturas de YOLO y realización de la detección de objetos.
- Instalación de las distintas librerías para poder realizar la detección de objetos en las distintas arquitecturas.
- Lectura y análisis de las métricas comúnmente usadas para el análisis de los detectores de objetos.
- Estudio, generación de los datasets utilizados, como COCO datasets.
- Ejecución y análisis de las distintas arquitecturas en la detección de imágenes.
- Ejecución y análisis de las distintas arquitecturas en la detección de video.
- Ejecución y análisis de las distintas arquitecturas en la detección de video en tiempo real.
- Análisis de los diferentes relés programables.
- Análisis de las comunicaciones industriales para comunicar la aplicación de detección de imágenes y un relé programable.
- Prototipado y puesta a punto del sistema de alarmas en tiempo real.

1.5 Organización de la memoria

El desarrollo de esta memoria se ha realizado siguiendo los objetivos y tareas a realizar, las secciones que componen la memoria son las siguientes:

- En la primera parte se realiza una introducción, se plantean los objetivos tareas y organización del trabajo de fin de máster a realizar.
- La segunda parte de la memoria consta del marco teórico de las redes convolucionales, haciendo hincapié en las capas que presentan las redes convolucionales y sus arquitecturas más destacadas.
- En la tercera sección se va a introducir a la detección de objetos y se van a presentar algunas de las arquitecturas más utilizadas.
- En la cuarta sección de este trabajo se incluyen las arquitecturas YOLO5, YOLOv8 y YOLONAS que se van a utilizar para comparar la aplicación real.
- La quinta sección es una introducción de las comunicaciones industriales, Node Red y los relés programables.
- La sexta sección se analizan los distintos sistemas hardware y software utilizados.

- En la séptima sección se describen los resultados que se han obtenido en las diferentes fases en las que se ha ido desarrollando el trabajo y prototipo.
- En la octava sección se incluyen las conclusiones obtenidas a partir de los resultados y se explican las posibilidades de ampliación futura de este Proyecto.
- En la sección número nueve, se incluye la bibliografía consultada.
- Finalmente, en la última sección, se añade el Anexo I con todo el código utilizado para realizar la detección de objetos.

2 Redes Neuronales Convolucionales

El aprendizaje profundo se encuentra dentro del campo más amplio del aprendizaje automático. En el proceso de entrenar modelos de aprendizaje profundo, es esencial contar con grandes volúmenes de datos. Estos datos se organizan durante su procesamiento en patrones mediante una serie de capas en el modelo. Las relaciones entre los datos originales y procesados se codifican a través de conexiones entre estas capas, cada una de las cuales contiene ponderaciones específicas o pesos. La fuerza de la relación entre los datos está determinada por el valor de estas ponderaciones. Este conjunto de capas y conexiones forma lo que se conoce como redes neuronales artificiales. La profundidad de una red se refiere al número de capas que contiene, y una mayor cantidad de capas la convierte en una red neuronal profunda.

Existen diversos tipos de redes neuronales, entre las que destacan el perceptrón multicapa (MLP, *MultiLayer Perceptron*), las redes neuronales convolucionales (CNN, *Convolutional Neural Network)* o las redes neuronales recurrentes (RNN, *Recurrent Neural Networks*) (Aggarwal, 2023). El MLP, siendo la más básica, asigna un conjunto de entradas a un conjunto de salidas y es apropiada cuando los datos carecen de un componente espacial o temporal. Las CNN utilizan capas convolucionales para procesar información espacial en los datos, siendo útil, por ejemplo, en el procesamiento de imágenes para detectar características en regiones específicas (como determinar si hay una nariz en el centro de una imagen). Por último, las RNN permiten la persistencia del estado o memoria, lo que las hace ideales para el análisis de series temporales donde la secuencia y el contexto de eventos son cruciales. (Tutorial: Detección de objetos con ONNX en ML.NET, 2023)

La piedra angular de este trabajo se encuentra en el desarrollo teórico de las Redes Neuronales Convolucionales (CNN), fundamentales para comprender la arquitectura YOLO (You Only Look Once) (Redmon & Farhadi, 2018). Al adentrarnos en el núcleo conceptual de las CNN, exploraremos su evolución histórica, características clave y aplicaciones, delineando un panorama completo de su papel en el aprendizaje profundo. Este fundamento teórico sienta las bases para desentrañar la arquitectura YOLO, la cual se erige sobre los principios de las CNN para lograr una detección de objetos eficiente y precisa. Al comprender la esencia de las redes neuronales convolucionales, realizando una comprensión profunda de la arquitectura YOLO y su impacto en el ámbito de la visión por computadora.

2.1 Teoría de las redes CNN

Las CNN tuvieron su origen a mediados de la década de 1990, pero fue a principios de los años 2010 cuando alcanzaron prominencia gracias a su aplicación exitosa en competiciones de reconocimiento de imágenes. El trabajo pionero de investigadores como Yann LeCun, Geoffrey Hinton, y Yoshua Bengio (1989) contribuyó significativamente al desarrollo y popularización de estas redes. Aunque ya había trabajos de Kunihiko Fukushima (1980) en la década de 1980, que diseño una red neuronal artificial que imitaba el funcionamiento de células simples y complejas, no es hasta el trabajo de Yann LeCun cuando tiene el origen las CNN. Desde entonces, las CNN se han convertido en una pieza fundamental del aprendizaje profundo, especialmente en tareas que implican procesamiento de datos (imágenes, videos, texto o sonido). (ICHI.PRO, s.f.)

Las CNN constituyen un hito crucial en el campo del aprendizaje profundo (Ap) o Deep Learning (DL) (LeCunn, Bengio, & Hinton, 2015), especialmente en aplicaciones relacionadas con la visión por computadora. Desarrolladas para imitar la capacidad visual humana, las CNN han transformado radicalmente la manera en que las máquinas interpretan e interactúan con imágenes. Desde su concepción, estas redes han revolucionado la capacidad de los sistemas de inteligencia artificial para comprender y extraer información visual de manera eficiente y precisa.

Antes de las CNN, se utilizaban métodos manuales de extracción de características que requerían mucho tiempo para identificar objetos en imágenes. Sin embargo, actualmente las CNN proporcionan un enfoque más escalable para las tareas de clasificación de imágenes y reconocimiento de objetos al aprovechar los principios del álgebra lineal, en concreto la multiplicación de matrices, para identificar patrones en una imagen, básicamente se trata de operaciones de convolución, de ahí su nombre. No obstante, estas redes pueden exigir un uso intensivo de recursos informáticos y requerir unidades de procesamiento gráfico (GPU) para entrenar los modelos. (IBM, s.f.)

Las CNN poseen varias características que las hacen excepcionalmente eficientes en el procesamiento de datos visuales. Desde las capas convolucionales que aplican filtros para detectar características locales hasta las capas de *pooling* que reducen la dimensionalidad, cada componente se ha diseñado para capturar información relevante y facilitar la generalización en tareas diversas.

Una de las características distintivas de las CNN es la presencia de capas *Fully Connected* al final de la arquitectura. Estas capas son responsables de fusionar las características extraídas en una representación global, permitiendo a la red realizar clasificaciones precisas. Exploraremos cómo las conexiones densas entre estas capas contribuyen a la capacidad de las CNN para modelar complejidades en los datos. (Krizhevsky et al, 2012)

Otra característica que presenta las redes CCN es la generalización y adaptabilidad. La capacidad de generalización de las CNN es uno de sus atributos más destacados. Analizaremos cómo estas redes pueden aprender patrones y características específicas en conjuntos de datos de entrenamiento y aplicar ese conocimiento a datos no vistos durante el entrenamiento. Esto no solo demuestra su capacidad para adaptarse a diferentes contextos, sino también su robustez frente a variabilidades en los datos.

Otra característica esencial es la compartición de pesos y sesgos en la CNN. Así; en contraste con las redes neuronales convencionales, las CNN adoptan un enfoque de pesos y sesgos compartidos, uniformes para todas las neuronas ocultas en una capa específica. Este principio implica que todas las neuronas ocultas tienen la capacidad de identificar las mismas características, como bordes o formas, en distintas regiones de la imagen. Este fenómeno confiere a la red una notoria tolerancia a las variaciones en la posición de los objetos en una imagen. De manera ilustrativa, consideremos una red entrenada para el reconocimiento de automóviles. Esta red podría identificar automóviles con eficacia, independientemente de su ubicación específica en la imagen, gracias a la compartición de pesos y sesgos. Este enfoque no solo optimiza el proceso de aprendizaje al reducir la cantidad de parámetros a ajustar, sino que también confiere a la CNN una capacidad única para generalizar patrones y características esenciales en imágenes de manera robusta. (Mathworks, s.f.)

Además, presentan unas estructuras jerárquicas, que son esenciales para la identificación de características a diferentes niveles de abstracción. Desde detalles finos hasta patrones más complejos, las capas convolucionales construyen representaciones jerárquicas que permiten a la red discernir información relevante. Investigaremos cómo estas estructuras escalonadas contribuyen a la eficacia de las CNN en tareas específicas. (LeCunn et al, 1998)

2.2 Funcionamiento de las CNN

Las arquitecturas de redes neuronales convolucionales son un tipo especializado de red neuronal para procesar datos que tienen una topología similar a una cuadrícula (LeCun, et al., 1989), en el contexto del procesamiento de imágenes, pueden extenderse a decenas o incluso centenares de capas. Cada una de estas capas desempeña un papel crucial al aprender a discernir y extraer diversas características de una imagen. Este proceso se inicia mediante la aplicación de filtros a las imágenes de entrenamiento, variando las resoluciones para capturar información en diferentes escalas. La salida resultante de la convolución de cada imagen se convierte en la entrada para la capa subsiguiente. Inicialmente, es decir en las primeras capas, estos filtros pueden identificar características elementales como brillo y bordes, evolucionando progresivamente hacia elementos de mayor complejidad que singularizan y definen de manera distintiva los objetos en la imagen. Este enfoque jerárquico refleja la capacidad de las redes neuronales convolucionales para aprender representaciones cada vez más sofisticadas a medida que se profundizan en las capas. En la Figura 2, tomada de TheMatworks (2024) se muestra un esquema de CNN genérico (Mathworks, s.f.)

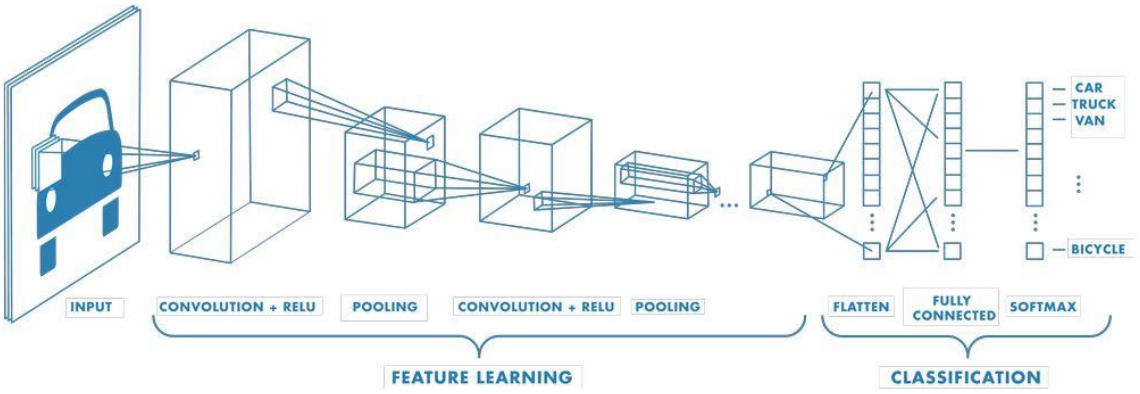

Figura 2 Ejemplo de red con múltiples capas convolucionales (The Marthworks, 2024)

Las redes neuronales convolucionales se distinguen de otras redes neuronales por su rendimiento superior con entradas de imagen, voz o señales de audio. Se componen de tres tipos principales de capas:

- Capa convolucional.
- Capa de agrupación.
- Capa totalmente conectada.
- Capa Softmax.

En la Figura 3 tomada del artículo (Lee, Yoon & Cho, 2017) muestra la distribución de las

capas de una red CNN.

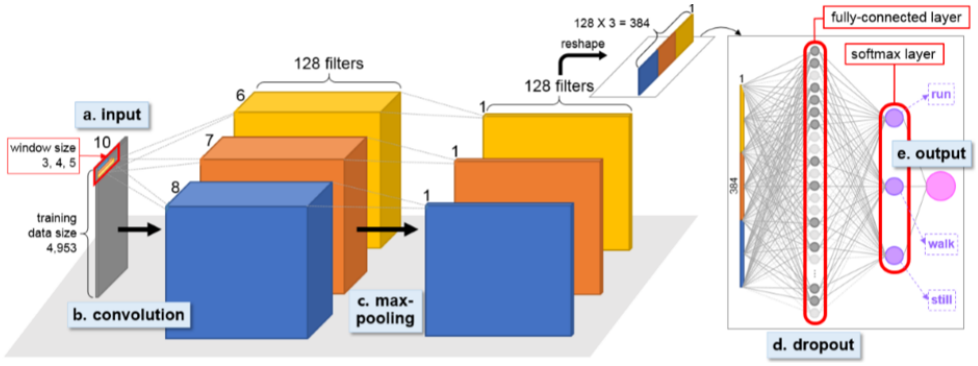

Figura 3 Capas principales de una red CNN (Lee, Yoon, & Cho, 2017)

a. Entrada de datos.
b. Capa convolucional.
c. Capa de agrupación (*pooling*)
d. Sobreajuste
e. Salida

En la Figura 3 también aparecen los conceptos de *fully-connected* y *softmax* que son elementos comunes en las redes neuronales, incluyendo las CNN y las redes neuronales profundas en general.

- **Fully Connected Layer (Capa Totalmente Conectada):**

Una capa totalmente conectada, también conocida como capa densa, es una capa en la que cada neurona o unidad está conectada a todas las neuronas de la capa anterior y a todas las de la capa siguiente. En otras palabras, cada salida de la capa anterior se conecta a cada entrada de la capa siguiente. Estas capas son comúnmente utilizadas para fusionar información y realizar operaciones de clasificación y toma de decisiones finales en la red.

En el contexto de una CNN, las capas totalmente conectadas a menudo se encuentran al final de la red, después de las capas convolucionales y de agrupación. Estas capas son responsables de tomar las características extraídas por las capas anteriores y utilizarlas para realizar la clasificación final o la tarea específica para la cual se está entrenando la red.

- **Softmax Layer (Capa Softmax):**

La capa *Softmax* aplica una función de activación que se utiliza comúnmente en la capa de salida de una red neuronal para problemas de clasificación con múltiples clases. La función *Softmax* toma un vector de números reales y lo transforma en una distribución de probabilidad, donde la suma de todas las probabilidades es igual a 1.

La ecuación de la función *Softmax* para una salida z_i en un vector de salida z es la siguiente:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (1)$$

Donde:

- e es la función exponencial.
- Z_i es la salida correspondiente a la clase i en el vector z.
- El denominador es la suma de las exponenciales de todas las salidas en el vector z.

La capa *Softmax* es crucial en problemas de clasificación multiclase, ya que convierte las puntuaciones de salida de la red en probabilidades. La clase con la probabilidad más alta se elige como la que proporciona la predicción final.
Lo que supone que la capa *fully-conected* se utiliza para combinar y transformar las características extraídas por las capas anteriores, mientras que la capa *Softmax* se utiliza para convertir las puntuaciones de salida en una distribución de probabilidad, facilitando así la clasificación en problemas con múltiples clases. (Courville, 2016)

La capa convolucional es, generalmente, la primera capa de una red convolucional. Si bien las capas convolucionales pueden ir seguidas de otras capas convolucionales o de capas de agrupación, la capa final es la capa totalmente conectada. Con cada capa, la CNN aumenta en complejidad, identificando partes cada vez más grandes de la imagen. Las primeras capas se centran en características simples, como colores y bordes. A medida que los datos de la imagen avanzan a través de las capas, la CNN comienza a reconocer elementos o formas más grandes hasta que finalmente identifica el objeto esperado. (IBM, s.f.)

Durante la parte convolucional de una CNN, la imagen proporcionada de entrada pasa a través de una **serie de filtros de convolución**. Por ejemplo, existen filtros de convolución que se suelen utilizar y que permiten extraer características más pertinentes que píxeles como la detección de bordes (**filtro derivador**) o formas geométricas. La elección y la aplicación de los filtros la hace de forma **automática** el modelo. (Departamento de Matemática Aplicada, UPM, 2021)
Las redes neuronales convolucionales modelan de forma consecutiva pequeñas piezas de información, tratando de extraer información sobre diferentes patrones de cada imagen. La primera capa intentará detectar los bordes y establecer patrones de detección de bordes, por ejemplo. Todo ello en una secuencia de mapas que constituirán el primer bloque de convolución. Luego, en secuencia de capas posteriores trataran de combinarlos en formas más

simples y, finalmente, en patrones de las diferentes posiciones de los objetos, iluminación, escalas, etc. Las capas finales intentarán hacer coincidir una imagen de entrada con todos los patrones y llegar a una predicción final como una suma ponderada de todos ellos, usando ya una red densa. Figura 4.

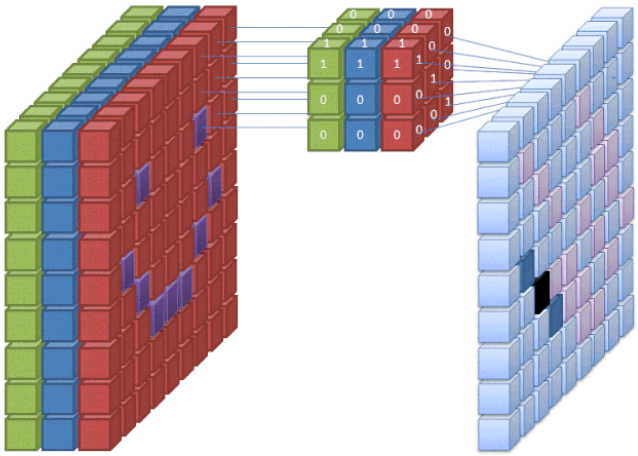

Figura 4 "Convolutional Neural Network with Color Image Filter" by Cecbur is licensed under CC BY-SA 4.0.

Una CNN organiza sus neuronas en tres dimensiones (ancho, alto, profundidad). Cada capa transforma el volumen de entrada 3D en un volumen de salida 3D de activaciones neuronales. En este ejemplo, la capa de entrada roja contiene la imagen, por lo que su ancho y alto serían las dimensiones de la imagen, y la profundidad sería de 3 (canales rojo, verde, azul). (Departamento de Matemática Aplicada, UPM, 2021)

2.2.1 Capa Convolucional

La capa convolucional constituye el núcleo esencial de una CNN, siendo el lugar donde se llevan a cabo la mayoría de los cálculos fundamentales, concretamente los filtrados mediante las correspondientes operaciones de convolución Para su funcionamiento, requiere diversos elementos, entre los cuales se encuentran los datos de entrada, un filtro y un mapa de características. Si consideramos la entrada como una imagen en color compuesta por una matriz de píxeles en 3D, estas dimensiones corresponden a la altura, anchura y profundidad, representando la composición RGB de la imagen.

Un componente clave en esta capa es el detector de características, también conocido como *kernel* o filtro. Este elemento se desplaza por los campos receptivos de la imagen para

identificar la presencia de ciertas características, dando lugar al proceso de convolución. La convolución implica la aplicación del filtro a una región de la imagen, calculando un producto escalar entre los píxeles de entrada y el filtro. El resultado de esta serie de productos escalares se denomina mapa de características, mapa de activación o característica convolucionada.
Tras cada operación de convolución, la CNN introduce una transformación de unidad lineal rectificada (ReLU, *Rectified Linear Unit*) al mapa de características, añadiendo no linealidad al modelo.

Es importante destacar que la capa de convolución inicial puede ser seguida por otra capa de convolución, generando una estructura jerárquica en la CNN. Este enfoque permite que las capas posteriores analicen los píxeles en los campos receptivos de las capas anteriores, facilitando la identificación de patrones complejos.

Los parámetros de la capa convolucional constan de un conjunto de filtros aprendibles. Cada filtro es espacialmente pequeño, pero abarca toda la profundidad del volumen de entrada. Por ejemplo, un filtro típico puede tener dimensiones de 7x7x3, representando 7 píxeles de ancho, 7 de alto y 3 canales de color, para una imagen de 3 canales. Durante el paso hacia adelante, cada filtro se desliza por el ancho y alto del volumen de entrada, calculando productos entre las entradas del filtro y la entrada en cada posición.

A diferencia de otras redes neuronales, las CNN utilizan la operación de convolución en lugar de la multiplicación de matrices, aprovechando conceptos clave como interacciones dispersas, parámetros compartidos y representaciones equivariantes. Esta operación permite trabajar con entradas de tamaño variable, lo que resulta conveniente en diversos contextos. La convolución se interpreta como la salida de una neurona que examina una región reducida en la entrada, compartiendo parámetros con las neuronas adyacentes espacialmente la Figure 5 muestra un esquema general de capa convolucional. (Departamento de Matemática Aplicada, UPM, 2021)

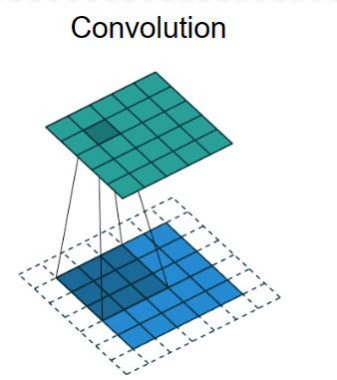

Figura 5 "ConvolutionAndPooling" by Andreas Maier is licensed under CC BY 4.0.

Veamos ahora un ejemplo de cómo sería la creación de una capa de convolución con el paquete PyTorch. **torch.nn.Conv2d** → se utiliza para definir una capa de convolución en una red neuronal convolucional (CNN). En el contexto de redes neuronales, la convolución es una operación clave para extraer características de datos bidimensionales, como imágenes. (Departamento de Matemática Aplicada, UPM, 2021)

- **torch.nn.Conv2d**(in_channels, out_channels, kernel_size, stride=1, padding=0, ...)

donde:

- ***in_channels*:** Número de canales de entrada. Para imágenes en escala de grises, esto sería 1, y para imágenes en color (RGB), sería 3.
- ***out_channels*:** Número de canales de salida. Este parámetro determina la cantidad de filtros que se aplicarán y, por lo tanto, la cantidad de canales en la salida de la capa convolucional.
- ***kernel_size*:** Tamaño del kernel o filtro que se aplicará durante la convolución. Puede ser un solo número o una tupla de dos números que representan la altura y la anchura del kernel.
- ***stride*:** Paso de desplazamiento del kernel durante la convolución. Especifica cuánto se mueve el kernel en cada paso.
- ***padding*:** Cantidad de relleno que se agrega alrededor de la entrada. El relleno se utiliza para mantener el tamaño espacial de la entrada después de la convolución.

Si completáramos la función de la convolución **Conv2d (3, 12, *kernel*_size=(7, 7), *stride*=(2, 2), *padding*=(3, 3), bias=False)** se especifica: (Departamento de Matemática Aplicada, UPM, 2021)

- La **profundidad de la entrada es 3**, por los canales RGB. Puede ser un canal único para el caso de imágenes en gris o más de 3 si se recogen datos en el espectro electromagnético con sensores especiales.
- La **profundidad de salida es 12**, se entrenan 12 características distintas para las imágenes.
- Las **dimensiones del filtro es 7 pixels de ancho por 7 de alto**.
- El **paso o *stride* es 2 por 2**. Si el paso o *stride* es 1 a 1 se mueve el filtro pixel a pixel y la salida es de igual dimensión, Si es 2 a 2 la dimensión de los mapas de salida se reduce.
- El ***padding* o relleno de ceros es 3 por 3**. De no usar el *padding*, un filtro (7, 7) tiene una pérdida de 3 filas superiores e inferiores y 3 columnas izquierda y derecha. Añadiendo un marco de ceros (3, 3) se evita tener que reducir la dimensión de los mapas de salida respecto a las imágenes de entrada
- El **parámetro Bias está anulado**. Aquí se indica si además de la matriz de pesos del *kernel* se utiliza una matriz de bias.

En el desplazamiento del filtro sobre el volumen de entrada, se genera un mapa de activación en 2 dimensiones que refleja las respuestas del filtro en cada posición espacial. La red aprende filtros que se activan ante características visuales específicas, generando una jerarquía de mapas de activación.

2.2.2 Capa de Agrupación (*Pooling*)

La agrupación, también conocida como submuestreo, permite reducir la dimensión mediante la reducción del número de parámetros de la entrada. De manera similar a la capa convolucional, la operación de agrupación barre toda la entrada con un filtro, o más bien una ventana, pero la diferencia es que este filtro no tiene ningún peso. En su lugar, el *kernel* aplica una función de agregación a los valores dentro del campo receptivo y llena así la matriz de salida. Hay dos tipos principales de agrupación:

- **Agrupación máxima:** a medida que el filtro recorre la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida. Este enfoque suele utilizarse más que la agrupación media.
- **Agrupación media:** conforme el filtro avanza por la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida.

Aunque se pierde mucha información en la capa de agrupación, esta tiene una serie de beneficios para la CNN. Ayuda a reducir la complejidad, mejora la eficiencia y limita el riesgo de sobreajuste.

La capa de agrupación reducción o *pooling* se coloca generalmente después de la capa convolucional. Su utilidad principal radica en la reducción de las dimensiones espaciales (ancho x alto) del volumen de entrada para la siguiente capa *convolucional*. No afecta a la dimensión de profundidad del volumen. La operación realizada por esta capa también se llama *reducción de muestreo*, ya que la reducción de tamaño conduce también a la pérdida de información. Sin embargo, una pérdida de este tipo puede ser beneficioso para la red por dos razones:

1. La disminución en el tamaño conduce a una menor sobrecarga de cálculo para las próximas capas de la red.
2. Sirve para reducir el sobreajuste.

La operación más generalizada que se suele utilizar en esta capa es *max-pooling*, que divide a la imagen de entrada en un conjunto de rectángulos y, respecto de cada subregión, se va quedando con el máximo valor, Figura 6.

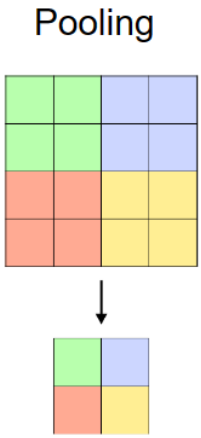

Figura 6 "ConvolutionAndPooling" by Andreas Maier is licensed under CC BY 4.0.

Deshacerse de la agrupación. Springenberg et al. (2014) proponen descartar la capa de agrupación en favor de una arquitectura que solo consiste en capas CONV repetidas. Para reducir el tamaño de la representación, sugieren usar un salto más grande en la capa CONV de vez en cuando. También se ha encontrado que descartar capas de agrupación es importante para entrenar buenos modelos generativos, como los autocodificadores variacionales (VAE) o las

redes generativas adversarias (GAN). Parece probable que las arquitecturas futuras presenten muy pocas o ninguna capa de agrupación. (Springenberg∗, 2014)

2.2.3 Reglas generales para modelar una red convolucional

- La **capa de entrada** (que contiene la imagen) debe ser divisible por 2^n. Los números comunes incluyen 32 (por ejemplo, CIFAR-10), 64, 96 (por ejemplo, STL-10) o 224 (por ejemplo, ImageNet), 384 y 512. (Departamento de Matemática Aplicada, UPM, 2021)
- Las **capas conv** deben usar filtros de dimensiones relativamente pequeñas (por ejemplo, 3x3 o como máximo 5x5, 7×7, 11×11 e incluso mayores, procurando que no superar valores del orden de 21×21, aunque esto depende de las dimensiones de las imágenes de entrada), usar un salto relativamente pequeños, por ejemplo $\mathbf{S}=1$ y, lo que es más importante, rellenar el volumen de entrada con ceros (*padding*) de tal manera que la capa conv no altere las dimensiones espaciales de la entrada (si F = 3, P = 1; si F = 5, P=2; en general $\mathbf{P} = (\mathbf{F}-1)/2$). (Departamento de Matemática Aplicada, UPM, 2021)
- Las **capas de *pooling*** se encargan de reducir el muestreo de las dimensiones espaciales de la entrada. La configuración más común es usar *max-pooling* con campos receptivos 2x2 (es decir, F = 2), y un salto de 2 (S = 2); esto es, se descarta exactamente el 75%. Más infrecuente, por la dificultad de encajarlo en la dimensión de la entrada es el uso de F = 3 y S = 2. Dimensiones de *maxpooling* superiores son muy infrecuentes ya que es muy agresiva y deficitaria y conducciendo a peores rendimentos. (Departamento de Matemática Aplicada, UPM, 2021)

2.3 Arquitectura CNN destacadas

Algunas de las redes neuronales convolucionales que se han destacado en distintas aplicaciones de visión por computadora, son las siguientes:

- **Redes Neuronales Convolucionales Tradicionales** (LeCun, y otros, 1989)**:**

Estas CNN básicas consisten en capas convolucionales seguidas por capas de agrupación, con una capa completamente conectada al final. Son fundamentales para tareas de clasificación de imágenes y reconocimiento de objetos.

- **Redes Neuronales Convolucionales Profundas (DCNN)** (Simonyan & Zisserman, 2014)**:**

También conocidas como Redes Neuronales Convolucionales Profundas, estas arquitecturas se

caracterizan por tener múltiples capas convolucionales, permitiendo una representación jerárquica más compleja de las características de la imagen.

- **Redes Neuronales Convolucionales Residuales (ResNet)** (He et al, 2016)**:**

Introducidas para abordar el desafío del entrenamiento de redes muy profundas, las ResNet incorporan conexiones residuales que facilitan el flujo de información y ayudan a prevenir el problema de desvanecimiento del gradiente.

- **Redes Neuronales Convolucionales Recurrentes (CRNN)** (Bai et al, 2018)**:**

Fusionan elementos de redes convolucionales con redes neuronales recurrentes, siendo efectivas en tareas que involucran datos secuenciales, como reconocimiento de voz, video y procesamiento de lenguaje natural.

- **Redes Generativas Convolucionales (DCGAN)** (Radford et al, 2015)**:**

DCGAN combina arquitecturas generativas y convolucionales, siendo empleadas en la generación de imágenes realistas. Su capacidad para crear contenido visualmente coherente ha sido valiosa en campos como la generación de imágenes artificiales.

- **Redes Neuronales Convolucionales de Atención (CNN de Atención)** (Chenet al, 2017)**:**

Estas redes incorporan mecanismos de atención para destacar regiones específicas de la imagen durante el proceso de clasificación, mejorando la precisión y la capacidad de interpretación.

- **YOLO (You Only Look Once)** (Redmon & Farhadi, 2018)**:**

YOLO es una arquitectura especializada en la detección de objetos en tiempo real. Su enfoque único se basa en dividir la imagen en una cuadrícula y realizar la detección en una única pasada a través de la red. YOLO se basa en una red convolucional GoogleNet, denominada Darknet.

- **VGGNet** (Simonyan & Zisserman, 2014)**:**

VGG es conocida por su simplicidad y uniformidad en la arquitectura. Consiste en bloques de capas convolucionales seguidas por capas de agrupación, logrando una representación profunda de las características.

- **Inception (GoogLeNet):**

Su principal contribución fue el desarrollo de un Módulo de Inicio que redujo drásticamente el número de parámetros en la red (4M, en comparación con AlexNet con 60M). Además, este documento utiliza Average Pooling en lugar de capas totalmente conectadas en la parte superior de la CNN, eliminando una gran cantidad de parámetros que no parecen importar mucho. También hay varias versiones de seguimiento de GoogLeNet, la más reciente Inception-v4. Desarrollada por Google, la arquitectura Inception se destaca por su módulo Inception, que utiliza filtros de diferentes tamaños en paralelo para capturar características a diferentes escalas.

- **AlexNet**:

AlexNet, presentada por Alex Krizhevsky, Ilya Sutskever, y Geoffrey Hinton en 2012, marcó un hito al ganar la competición ImageNet Large Scale Visual Recognition Challenge. Destaca por su arquitectura profunda con capas convolucionales y de agrupación, contribuyendo a la popularización de CNN en el ámbito de la visión por computadora.

- **MobileNet** (Howard et al, 2017)**:**

Diseñada para aplicaciones en dispositivos móviles, MobileNet emplea operaciones convolucionales separables en profundidad para lograr una red eficiente en términos computacionales y de almacenamiento.

3 Detección de objetos

La detección de objetos es un proceso fundamental en el campo de la visión por computadora que combina dos tareas esenciales: la clasificación y la localización de objetos en una imagen. A diferencia de los modelos de clasificación de imágenes, que simplemente etiquetan la imagen completa con una categoría específica, los modelos de detección de objetos van un paso más allá al identificar y ubicar la posición de los objetos individuales en la escena.

Cuando se habla de visión por computador se hace referencia a dos procesos, **clasificación y detección, e**stos términos, aunque comúnmente utilizados en el ámbito de la visión por computadora, no son sinónimos y se refieren a dos tareas distintas.
En primer lugar, la **clasificación** tiene como objetivo asignar etiquetas o categorías al contenido completo de una imagen. En otras palabras, responde preguntas como "¿Hay un perro en esta foto?", "¿Qué animal es este?", "¿A qué clima pertenece este paisaje?" o "¿De qué raza es este perro?".
Por otro lado, la **detección** va más allá de simplemente identificar lo que hay en la foto; busca determinar la ubicación específica del objeto de interés en la imagen. En lugar de solo decir qué elementos están presentes, proporciona información sobre dónde se encuentran esos objetos dentro de la imagen.

La detección de objetos se logra mediante el concepto de "Bounding Boxes" (cuadros delimitadores). Un "Bounding Box" es esencialmente un rectángulo que rodea la región espacial de un objeto detectado. Para definir completamente la posición de este cuadro delimitador, se utilizan coordenadas (x, y), que indican la posición del cuadro en el plano de la imagen. Además, se incluyen las dimensiones del cuadro delimitador, generalmente expresadas en términos de altura y anchura, esto es: (x, y, ancho, alto). En la Figura 7 izquierda se muestra una imagen con el fin de proceder a su clasificación, mientras en la parte derecha se trata de la identificación de objetos, con sus correspondientes Bounding Boxes.

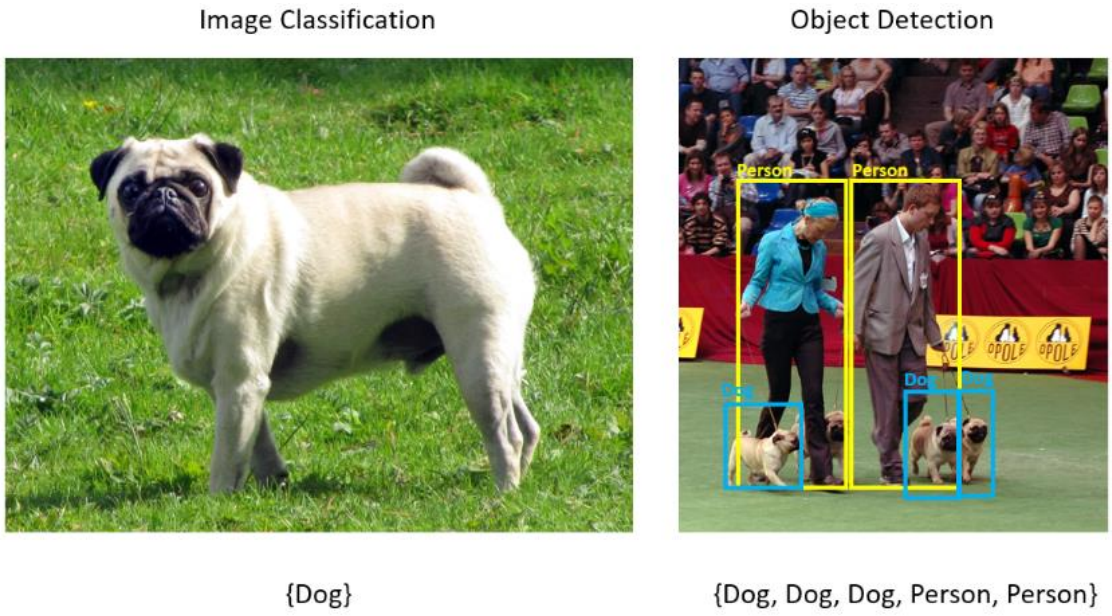

Figura 7 Detección de Imágenes (Microsoft, 2023)

El proceso de detección de objetos implica, por lo tanto, dos tareas principales:

- **Clasificación:**

Asignar una etiqueta o categoría a cada objeto detectado. Cada objeto dentro del cuadro delimitador se clasifica en una categoría específica, como automóvil, persona o perro.

- **Localización:**

Definir las coordenadas (x, y, ancho, alto) del cuadro delimitador que rodea cada objeto. Esto implica precisar la posición exacta del objeto en la imagen y proporcionar una representación espacial del mismo.

Dentro de los diferentes algoritmos que podemos encontrar para la detección de objetos, los más populares en detección de una pasada son:

1. **YOLO (You Only Look Once)** (Redmon & Farhadi, 2018)**:**
    a. **Enfoque Principal**: Detección en una sola pasada.
    b. **Características Destacadas:** Divide la imagen en una cuadrícula y realiza predicciones de cuadros delimitadores y probabilidades de clasificación simultáneamente.
2. **SSD (Single Shot MultiBox Detector)** (Pajares et al, 2021)**:**
    a. **Enfoque Principal**: Detección en una sola pasada.
    b. **Características Destacadas:** Utiliza cuadros delimitadores de diferentes escalas en capas convolucionales para abordar objetos de diferentes tamaños.

Es eficiente y adecuado para tiempo real.

Y los detectore de objetos que realizan más de una pasada:

1. **Faster R-CNN (Region-based Convolutional Neural Network)** (Pajares et al, 2021)**:**
    a. **Enfoque Principal:** Propone regiones de interés y clasificación en dos etapas.
    b. **Características Destacadas:** Introduce regiones de interés (RoIs) propuestas por una red para la clasificación y regresión de cuadros delimitadores. Ha sido mejorado con variantes como Faster R-CNN with Feature Pyramid Network (FPN).
2. **RetinaNet** (Lin et al, 2017)**:**
    a. **Enfoque Principal:** Aborda el desafío del desequilibrio de clases en la detección.
    b. **Características Destacadas**: Introduce la función de pérdida focal para dar más peso a los ejemplos difíciles y utiliza una Feature Pyramid Network (FPN) para manejar objetos de diferentes escalas.
3. **Mask R-CNN** (Pajares et al, Mask R-CNN, 2021)**:**
    a. **Enfoque Principal**: Extensión de Faster R-CNN para la segmentación de instancias.
    b. **Características Destacadas:** Además de las capas de detección, incorpora una rama de máscara para la segmentación precisa de instancias.
4. **EfficientDet** (Tan et al, 2020)**:**
    a. **Enfoque Principal**: Eficiencia en términos de recursos computacionales.
    b. **Características Destacadas**: Utiliza un enfoque escalable para equilibrar la precisión y la eficiencia, empleando la técnica de búsqueda de red para ajustar los hiperparámetros automáticamente.

El modelo de detección de objetos elegido para profundizar y desarrollar sobre el modelo YOLO en sus diferentes versiones.

4 YOLO

YOLO, acrónimo de "You Only Look Once," es un innovador algoritmo de detección de objetos desarrollado por Joseph Redmon et al (2016). A diferencia de otros enfoques que dividen la tarea de detección en múltiples etapas, YOLO aborda la detección de objetos de manera única y eficiente en una sola pasada. (Ultralytics)

YOLO es conocido por su capacidad para realizar detecciones en tiempo real al dividir la imagen en una cuadrícula y predecir cuadros delimitadores y probabilidades de clasificación simultáneamente. Esto significa que, en una sola inferencia, YOLO es capaz de identificar y localizar múltiples objetos en una escena.

Las ventajas que ofrece YOLO frente a otros algoritmos de detección de objeto son las siguientes: (Joseph Redmon, 2016)

- **Eficiencia en Tiempo Real:** YOLO destaca por su capacidad para realizar detecciones rápidas en tiempo real, haciéndolo adecuado para aplicaciones en video y sistemas en tiempo real.
- **Manejo de Múltiples Objetos:** Su capacidad para detectar y clasificar varios objetos en una sola pasada es una ventaja clave frente a métodos que dividen la tarea en múltiples etapas.
- **Generalización:** YOLO generaliza bien a diversas clases de objetos y se entrena eficazmente en conjuntos de datos amplios.

Además, YOLO se distribuye bajo **una licencia GPL** (*General Public License*). Esta licencia de código abierto permite la distribución, modificación y uso del código fuente de YOLO, siempre y cuando se respeten las condiciones de la licencia GPL. La GPL promueve la libertad del usuario y garantiza que las versiones modificadas del software también se distribuyan con una licencia de código abierto. Lo que permite el acceso y uso de YOLO, en sus diversas versiones y adaptaciones, es gratuito y está disponible para la comunidad de desarrolladores. Esto ha permitido que la comunidad haya desempeñado un papel fundamental en el desarrollo y la mejora continua de YOLO. Con contribuciones, correcciones de errores y mejoras provienen de desarrolladores de todo el mundo, enriqueciendo su funcionalidad y eficacia del sistema.

El desarrollo y la implementación de YOLO en Python son posibles gracias a la existencia de numerosas implementaciones y adaptaciones en este lenguaje. Uno de los proyectos notables es YOLOv5, desarrollado por Ultralytics, que ha ganado popularidad por su eficiencia y facilidad de uso. YOLOv5 está escrito en Python y se integra con el popular marco de aprendizaje profundo PyTorch. Para el caso de estudio de este trabajo se desarrollan las aplicaciones en lenguaje Python aprovechando la integración de estas versiones.

Algunas de las versiones que vamos a utilizar están realizadas por Ultralytics que es una empresa de software de código abierto que desarrolla herramientas y bibliotecas para la visión artificial. La empresa fue fundada en 2019 por Joseph Redmon y Ross Girshick, los creadores del modelo de detección de objetos YOLO. La empresa es responsable del desarrollo y mantenimiento de las últimas versiones de YOLO, incluyendo YOLOv3, YOLOv4, YOLOv5 y YOLOv8. (Ultralytics)

Durante la realización del Trabajo gran parte de la documentación que se va a seguir para el desarrollo es la documentación que aporta ultralytics, en su dirección web: Home - Ultralytics YOLOv8 Docs, ya que podemos encontrar una actualización continuada de los modelos y las herramientas.

Dentro de las diferentes versiones de Yolo podemos encontrar los siguientes modelos destacados: (Ultralytics)

- **YOLOv1:** Fue la primera versión, presentando el enfoque integral de detección en una sola pasada.
- **YOLOv2 (YOLO9000):** Introdujo mejoras en términos de precisión y capacidad para detectar un conjunto amplio de clases.
- **YOLOv3:** Optimizó la precisión y la velocidad, destacando la capacidad de detectar objetos pequeños y la incorporación de capas "Feature Pyramid Network" (FPN).
- **YOLOv4:** Trajo mejoras significativas en velocidad y precisión, además de optimizaciones de hardware, como la inclusión de la arquitectura CSPDarknet53.
- **YOLOv5:** Aunque no fue desarrollado por Joseph Redmon, YOLOv5 es una versión popular creada por Ultralytics, manteniendo el enfoque en la eficiencia y rendimiento.
- **YOLOv6**: Lanzado por Meituan en 2022 y en uso en muchos de los robots de reparto autónomos de la compañía.
- **YOLOv7**: Modelos YOLO actualizados lanzados en 2022 por los autores de **YOLOv4**.

- **YOLOv8**: La última versión de la familia YOLO, con capacidades mejoradas como la segmentación de instancias, la estimación de poses/puntos clave y la clasificación.
- Segment Anything Model (SAM**): Segment Anything Model (SAM)** de Meta.
- **Mobile Segment Anything Model (MobileSAM): MobileSAM** para aplicaciones móviles, de la Universidad Kyung Hee.
- **Modelo de Segmento Rápido de Cualquier Cosa (FastSAM): FastSAM** por el Grupo de Análisis de Imagen y Video, Instituto de Automatización, Academia China de Ciencias.
- **YOLO-NAS**: Modelos de búsqueda de arquitectura neuronal (NAS) de YOLO.

En los siguientes apartados profundizaremos en las arquitecturas de las versiones que se estudian en el trabajo que son: YOLOv5, YOLOv8 y YOLONAS.

4.1 YOLOv5

YOLOv5 es una versión importante de YOLO que presenta una serie de mejoras sobre las versiones anteriores. Estas mejoras incluyen:

- Una nueva arquitectura de red neuronal llamada PANet, que es más eficiente y precisa.
- La implementación de técnicas de aprendizaje profundo más avanzadas, como Cascaded Refinement Network (CRNN) y Cross Stage Partial Convolution (CSP).

YOLOv5 viene en cuatro versiones principales: pequeña (s), mediana (m), grande (l) y extragrande (x), cada una de las cuales ofrece tasas de precisión progresivamente más altas. Cada variante también requiere una cantidad diferente de tiempo para entrenar. (Solawetz, 2020)

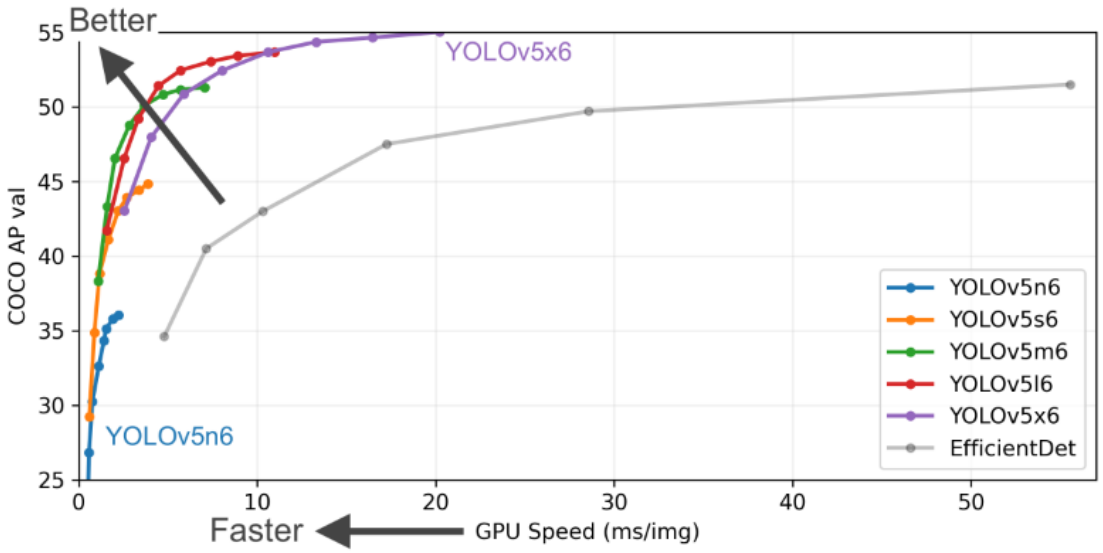

Figura 8 Comparativa versiones YOLOv5 (Github/Ultralitycs/Yolo)

El objetivo es producir un modelo de detector de objetos que sea muy eficaz (eje Y) en relación con su tiempo de inferencia (eje X). En el gráfico de la Figura 8 se puede ver que todas las variantes de YOLOv5 entrenan más rápido que EfficientDet. El modelo YOLOv5 más preciso, YOLOv5x, puede procesar imágenes varias veces más rápido con un grado de precisión similar al del modelo EfficientDet D4.

YOLOv5 deriva la mayor parte de su mejora del rendimiento de los procedimientos de entrenamiento de PyTorch, mientras que la arquitectura del modelo se mantiene cerca de YOLOv4. La mayor contribución de YOLOv5 es traducir el marco de investigación del modelo Darknet al marco PyTorch, Figura 9. El marco de la Darknet está escrito principalmente en C y ofrece un control detallado sobre las operaciones codificadas en la red. En muchos sentidos, el control del lenguaje de nivel inferior es una bendición para la investigación, pero puede hacer que sea más lento portar nuevos conocimientos de investigación, ya que se escriben cálculos de gradiente personalizados con cada nueva adición.

Figura 9 Logotipo Pytorch (Pytorch)

4.1.1 Arquitectura

La arquitectura de YOLOv5, Figura 10, consta de tres partes principales: (Ultralytics)

- **Backbone CPSDarknet:** una red neuronal convolucional que agrega y forma entidades de imagen con diferentes granularidades, es el cuerpo principal de la red. Para YOLOv5, la red troncal se diseña utilizando la estructura CSP-Darknet53, una modificación de la arquitectura Darknet utilizada en versiones anteriores.
- **Neck PA-Net**: Esta parte conecta la columna vertebral y la cabeza, es una serie de capas para mezclar y combinar entidades de imagen para pasarlas a la predicción. En YOLOv5, y se utilizan estructuras CSP-PANET.
- **Head YOLO Layer**: Esta parte se encarga de generar el resultado final. YOLOv5 utiliza tres DDHs (Double IoU-aware Decoupled Head) para este propósito. YOLOv3 Head.

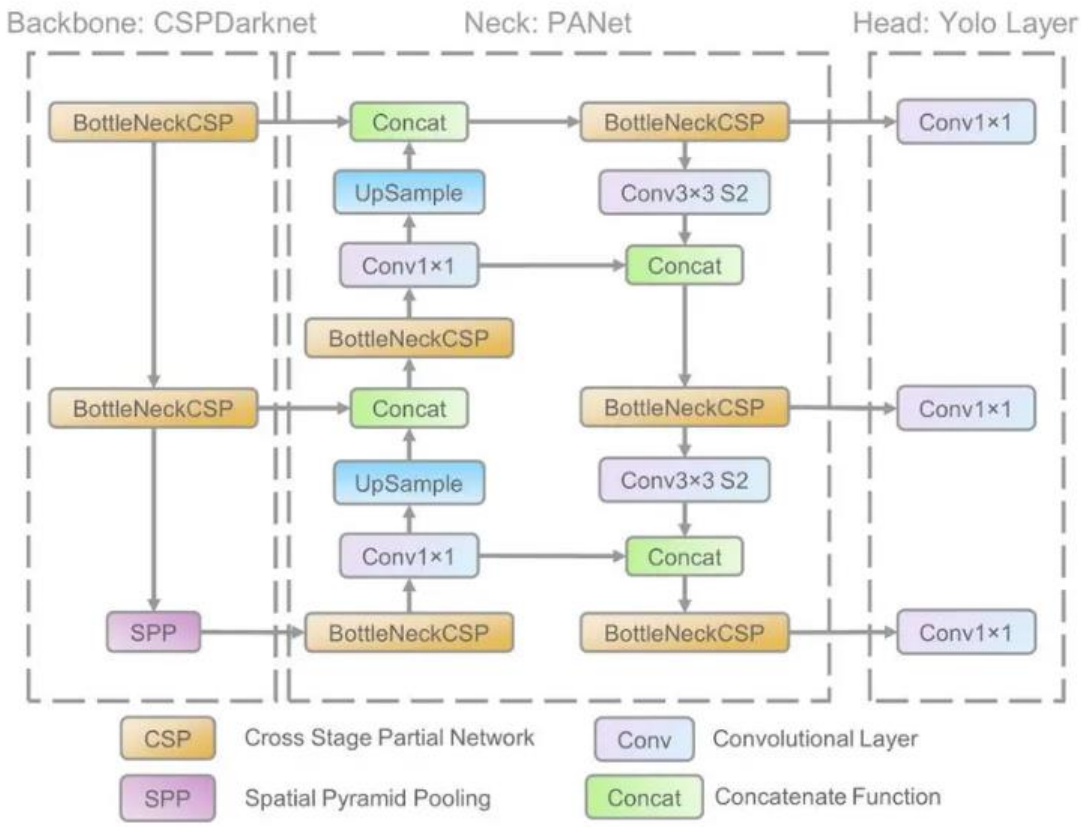

Figura 10 Arquitectura YOLOv5 (Wang, et al., 2019)

El modelo YOLO fue el primer detector de objetos en conectar el procedimiento de predicción de cuadros delimitadores con etiquetas de clase en una red diferenciable de extremo a extremo, Figura 11. (Bochkovskiy et al, 2020)

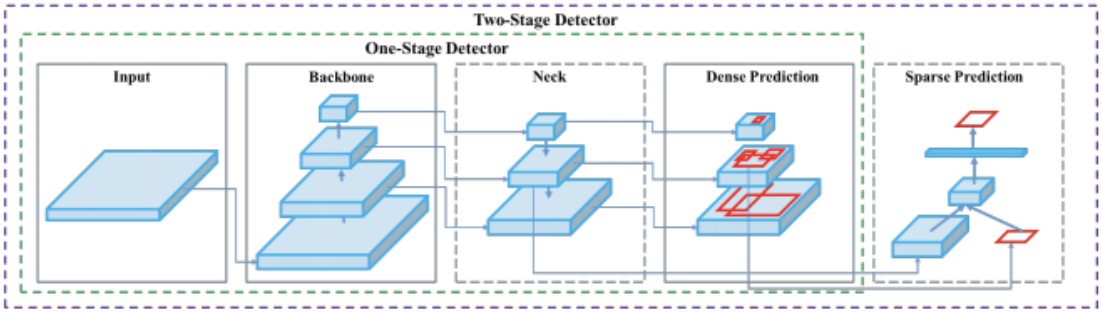

Figura 11 imagen del proceso de detección de objetos YOLOv4

4.1.1.1 Red troncal de CSP

Tanto YOLOv4 como YOLOv5 implementan el cuello de botella de CSP (**CSPDarknet53**) para formular características de imagen. El CSP aborda los problemas de gradiente duplicado en otras redes troncales ConvNet más grandes, lo que da como resultado menos parámetros y menos FLOPS para una importancia comparable. Esto es extremadamente importante para la familia YOLO, donde la velocidad de inferencia y el pequeño tamaño del modelo son de suma importancia. (Wang, et al., 2019)

Los modelos CSP se basan en DenseNet. DenseNet fue diseñado para conectar capas en redes neuronales convolucionales con las siguientes motivaciones:

- Aliviar el problema del gradiente de fuga (es difícil respaldar las señales de pérdida a través de una red muy profunda);
- Reforzar la propagación de características.
- Animar a la red a reutilizar las funciones.
- Reducir el número de parámetros de red.

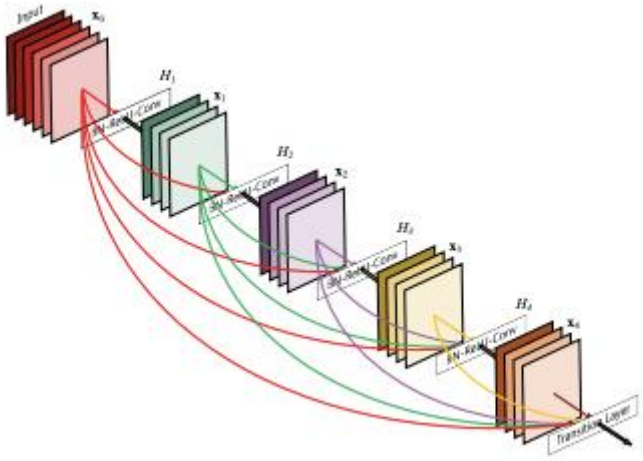

Figura 12 Capas en una densa red con conexiones entre capas (Gao Huang, 2017)

En CSPResNext50 y CSPDarknet53, DenseNet se ha modificado para separar el mapa de entidades de la capa base copiándolo y enviando una copia a través del bloque denso y enviando otra directamente a la siguiente etapa. La idea con CSPResNext50 y CSPDarknet53 es eliminar los cuellos de botella computacionales en DenseNet y mejorar el aprendizaje mediante la transmisión de una versión sin editar del mapa de características.

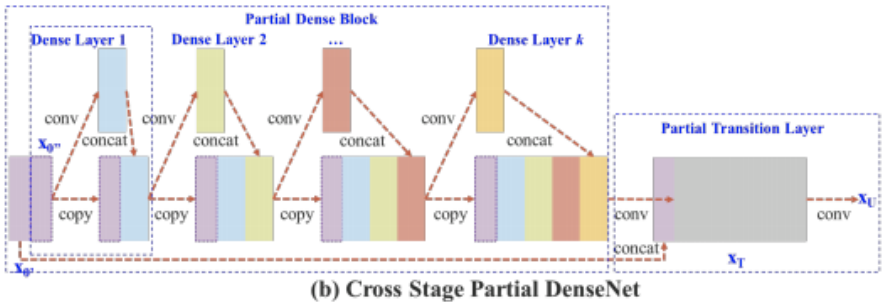

Figura 13 Pasos por los que pasan los datos en DenseNet a través de k capas en una capa de transición parcial (Wang, et al., 2019)

4.1.2 Darknet

Darknet es una plataforma de código abierto desarrollada por Joseph Redmon, Figura 14, creador de YOLO (You Only Look Once). Este entorno de desarrollo, escrito en C y CUDA, está diseñado para admitir la implementación eficiente de algoritmos de visión por computadora y aprendizaje profundo. La arquitectura se basa en una red convolucional GoogleNet y consta de 24 capas convolucionales. Darknet es conocido principalmente por ser el marco de referencia original de YOLO, un revolucionario algoritmo de detección de objetos. La documentación de Darknet se puede encontrar en [GitHub - pjreddie/darknet: Convolutional Neural Networks]

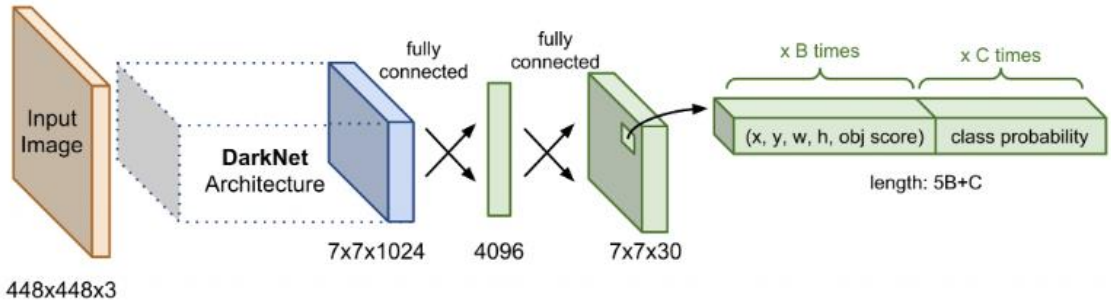

Figura 14 Capas Darknet (Bagnato, 2020)

Darknet proporciona una infraestructura robusta para el desarrollo y la implementación de redes neuronales, centrándose especialmente en aplicaciones de visión por computadora. Su

arquitectura modular permite la creación y conexión de capas de red con facilidad. YOLO, como uno de los algoritmos emblemáticos implementados en Darknet, adopta un enfoque novedoso al realizar la detección de objetos en una sola pasada, dividiendo la imagen en cuadrículas y prediciendo cuadros delimitadores y probabilidades de clasificación simultáneamente.

4.1.2.1 Darknet53:

Darknet53 (Wang, et al., 2019) es una arquitectura de red neuronal convolucional utilizada como *backbone* en modelos como YOLOv3. Darknet53 se caracteriza por tener 53 capas convolucionales.

Las características principales de Darknet53 es la utilización bloques residuales, similar a la arquitectura ResNet, para facilitar el entrenamiento profundo y ayudar en la propagación del gradiente durante el entrenamiento. Tiene conexiones residuales que permiten que la información fluya directamente a través de los bloques sin pérdida.

4.1.2.2 CSPDarknet53:

CSPDarknet53 es una variante de Darknet53 que introduce la idea de "*Cross-Stage Partial Networks*" (CSP), diseñada para mejorar la eficiencia y el rendimiento del modelo. Una de las características principales de CSPDarknet53 es la división de la red en dos partes: la entrada y la salida, que se dividen en dos mitades y se conectan cruzadamente, permitiendo una mejor propagación de la información a través de la red. La arquitectura CSP ayuda a reducir el problema de la degradación del rendimiento en redes muy profundas, permitiendo que la información fluya de manera más eficiente. (Bochkovskiy et al , 2020)

Una de las características más distintiva de CSPDarknet53 es la introducción de conexiones cruzadas entre las etapas de la red, lo que facilita la comunicación entre diferentes partes de la arquitectura, lo que permite una degradación del rendimiento en redes profundas, mejorando la eficiencia de la propagación de información a lo largo de la red.

4.1.3 Neck PA-Net

Tanto YOLOv4 como YOLOv5 implementan el cuello PA-NET para la agregación de características.

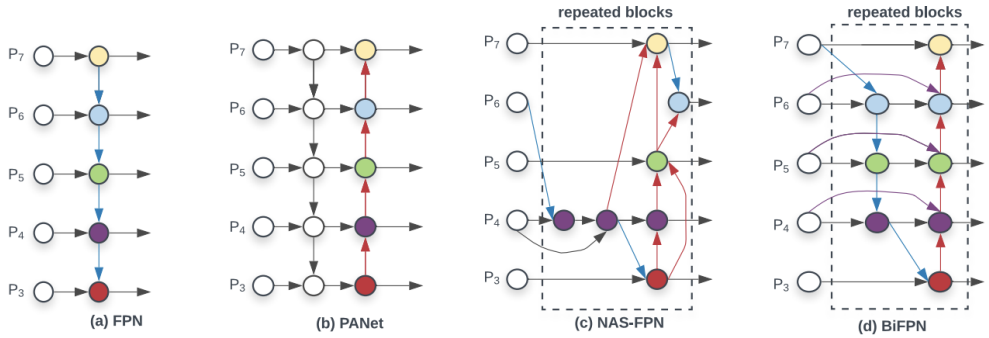

Figura 15 Diseño de Redes (EfficientDet)

Cada uno de los gráficos anteriores representa una capa de entidades en la red troncal de CSP.
La imagen de arriba proviene de una investigación realizada por Google Brain sobre la arquitectura de detección de objetos EfficientDet, Figura 15. Los autores de EfficientDet encontraron que BiFPN es la mejor opción para el cuello de detección, y puede ser un área de mayor estabilidad para YOLOv4 y YOLOv5.

La arquitectura PA-Net es una nueva arquitectura de red neuronal presentada en el trabajo (Chen, Chen, & Yu, 2023). Esta arquitectura está diseñada para mejorar el rendimiento de detección de objetos en entornos de recursos limitados, como dispositivos móviles o cámaras de seguridad de baja resolución.
La PA-Net consta de dos módulos principales: un módulo de detección y un módulo de predicción. El módulo de detección es responsable de generar las predicciones de detección, mientras que el módulo de predicción es responsable de mejorar la precisión de estas predicciones.

- El módulo de detección de la PA-Net es una red YOLOv5 modificada. Esta red utiliza una arquitectura en cascada para detectar objetos de diferentes tamaños. El módulo de detección primero genera una lista de candidatos de detección, y luego utiliza un clasificador para clasificar estos candidatos.
- El módulo de predicción de la PA-Net utiliza una red CNN para mejorar la precisión de las predicciones de detección. Esta red recibe como entrada las predicciones del módulo de detección, y utiliza estas predicciones para generar nuevas predicciones más precisas.

YOLOv5 ha demostrado ser muy preciso y eficaz en una amplia gama de aplicaciones. En particular, ha demostrado ser superior a otras versiones de YOLO anteriores.

4.2 YOLOv8

YOLOv8 es una versión aún más reciente de YOLO que presenta una serie de mejoras adicionales sobre YOLOv5. Estas mejoras incluyen:

- Una nueva arquitectura de red neuronal llamada CSPHead, que es aún más eficiente y precisa que PANet.
- La implementación de una nueva técnica de aprendizaje profundo llamada Focal Loss, que ayuda a mejorar la precisión de la detección de objetos difíciles.
- La utilización de un nuevo conjunto de datos de entrenamiento llamado COCO 2022, que es aún más grande y más diverso que COCO 2017.

YOLOv8 ha demostrado ser aún más preciso y eficaz que YOLOv5. En particular, ha demostrado ser superior a otras versiones de YOLO en la detección de objetos pequeños, difíciles y en movimiento.

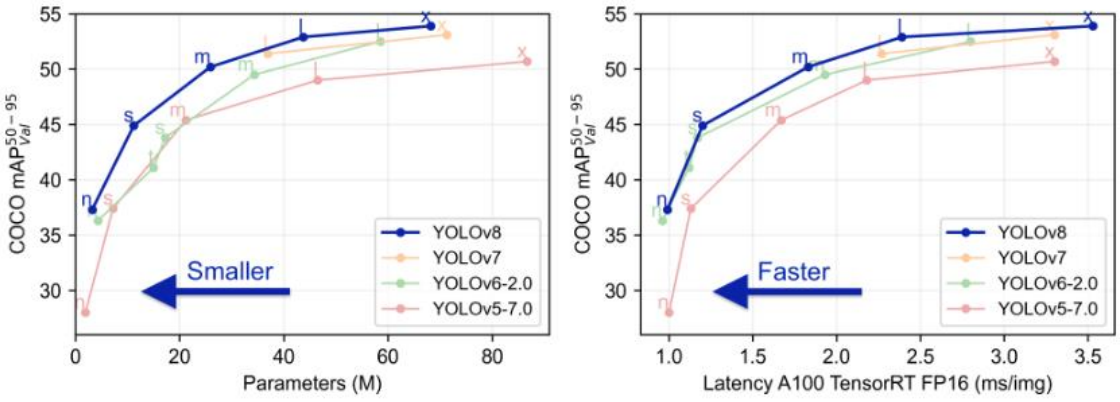

Figura 16 Comparativa Versiones Yolo (Github/Ultralitycs/Yolo, s.f.)

En la Figura 16 se representa en la gráfica con diferentes tamaños como "n", "s", "m", "l" y "x"; de las diferentes versiones de YOLO, donde se observa que las versiones con menos parámetros tienen una menor latencia y por tanto son modelos más rápidos que aquellos con más parámetros.

4.2.1 Comparación de YOLOv5 y YOLOv8

YOLOv5 y YOLOv8 son dos versiones muy potentes del algoritmo YOLO. Ambas versiones presentan una serie de mejoras sobre las versiones anteriores, lo que las hace aún más precisas y eficaces en una amplia gama de aplicaciones.

En general, YOLOv8 es una versión más reciente y avanzada que YOLOv5. Presenta una serie de mejoras adicionales que la hacen aún más precisa y eficaz. Sin embargo, YOLOv5 sigue siendo una versión muy potente que puede utilizarse para una amplia gama de aplicaciones.

La Figura 17 realizada por el usuario de GitHub RangeKing (RangeKing, 2022) muestra una visualización detallada de la arquitectura de la red.

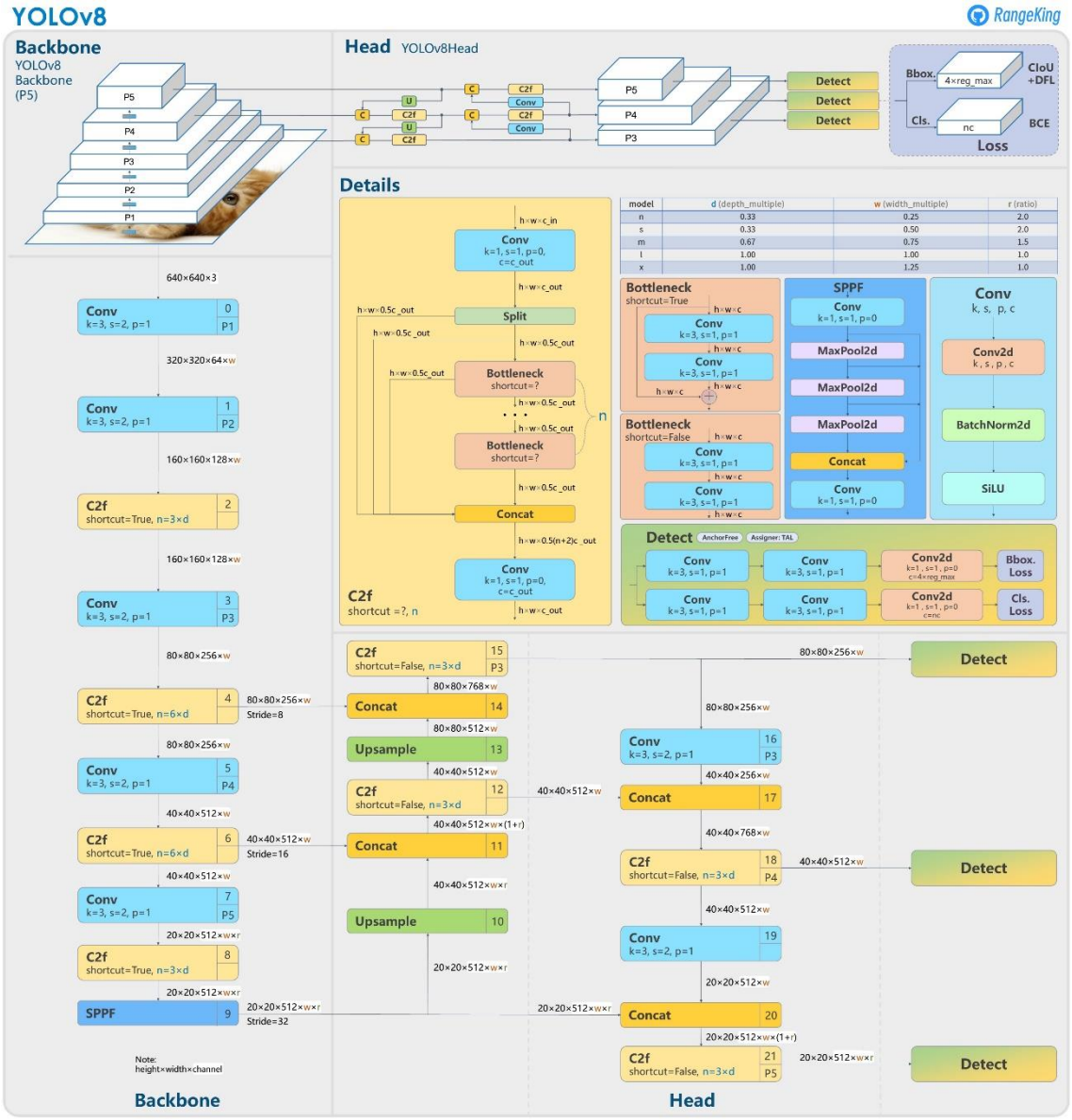

Figura 17 Arquitectura YOLOv8, visualización realizada por el usuario de GitHub RangeKing

Cambios en comparación con YOLOv5:

1. Reemplaza el módulo C3 por el módulo C2f
2. Reemplaza el primer Conv de 6x6 con Conv de 3x3 en la red troncal
3. Elimina dos Convs (No.10 y No.14 en la configuración de YOLOv5)
4. Reemplaza el primer Conv 1x1 con Conv 3x3 en el cuello de botella
5. Utiliza la cabeza desacoplada y elimine la rama de objetividad

YOLOv8 utiliza una variante del CSPDarknet53 que ha sido optimizada para la detección de objetos de diferentes tamaños. Esta variante utiliza una arquitectura en cascada que permite a la red neuronal procesar imágenes de diferentes resoluciones. Esto permite a YOLOv8 detectar objetos pequeños y grandes con mayor precisión.

En concreto, YOLOv8 ofrece las siguientes mejoras frente a YOLOv5:

- Un 20% eficiente: YOLOv8 es un 20% más eficiente que YOLOv5, tanto en términos de recursos computacionales como de memoria. Esto la hace adecuada para su uso en dispositivos móviles o cámaras de seguridad de baja resolución.
- Una mejora del 5% en la precisión: YOLOv8 ofrece una mejora del 5% en la precisión frente a YOLOv5. Esto se debe a las mejoras en el backbone CSPDarknet53, el cuello PA-Net y la cabeza YOLOv5.

4.2.2 Detección sin anclaje

YOLOv8 es un modelo sin anclaje. Esto significa que predice directamente el centro de un objeto en lugar del desplazamiento de un cuadro de anclaje conocido. (Solawetz, 2020)

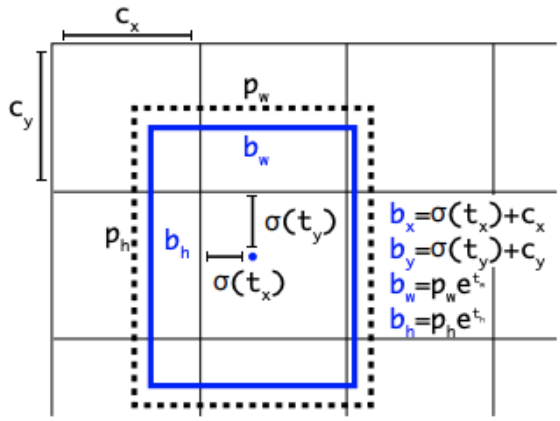

Figura 18 Visualización caja de anclaje (Redmon & Farhadi, 2018)

Visualización de una caja de anclaje en YOLO

Las cajas de anclaje eran una parte notoriamente complicada de los modelos YOLO anteriores, ya que pueden representar la distribución de las cajas del punto de referencia objetivo, pero no la distribución del conjunto de datos personalizado.

4.3 YOLONas

YOLONAS es una nueva versión del algoritmo YOLO desarrollada por Deci AI (The Deep Learning Development Platform | Deci). YOLONAS utiliza una técnica de búsqueda de arquitecturas neuronales llamada AutoNAC para encontrar la arquitectura de red neuronal óptima para la detección de objetos.

AutoNAC es una técnica de búsqueda de arquitecturas neuronales que utiliza un proceso de optimización para encontrar la arquitectura de red neuronal óptima para una tarea determinada. El proceso de optimización se basa en un algoritmo genético que se utiliza para explorar el espacio de búsqueda de arquitecturas neuronales.

YOLONAS ha demostrado ser muy preciso y eficaz en una amplia gama de aplicaciones. En particular, ha demostrado ser superior a otras versiones de YOLO en la detección de objetos pequeños, difíciles y en movimiento. (DeciAI)

YOLONAS presenta una serie de ventajas sobre otras versiones de YOLO, entre las que se incluyen:

- Mayor precisión: YOLONAS ha demostrado ser más preciso que otras versiones de YOLO en la detección de objetos pequeños, difíciles y en movimiento.
- Mayor eficiencia: YOLONAS es más eficiente que otras versiones de YOLO en términos de velocidad de ejecución.
- Mayor flexibilidad: YOLONAS es más flexible que otras versiones de YOLO, ya que puede adaptarse a una amplia gama de aplicaciones.

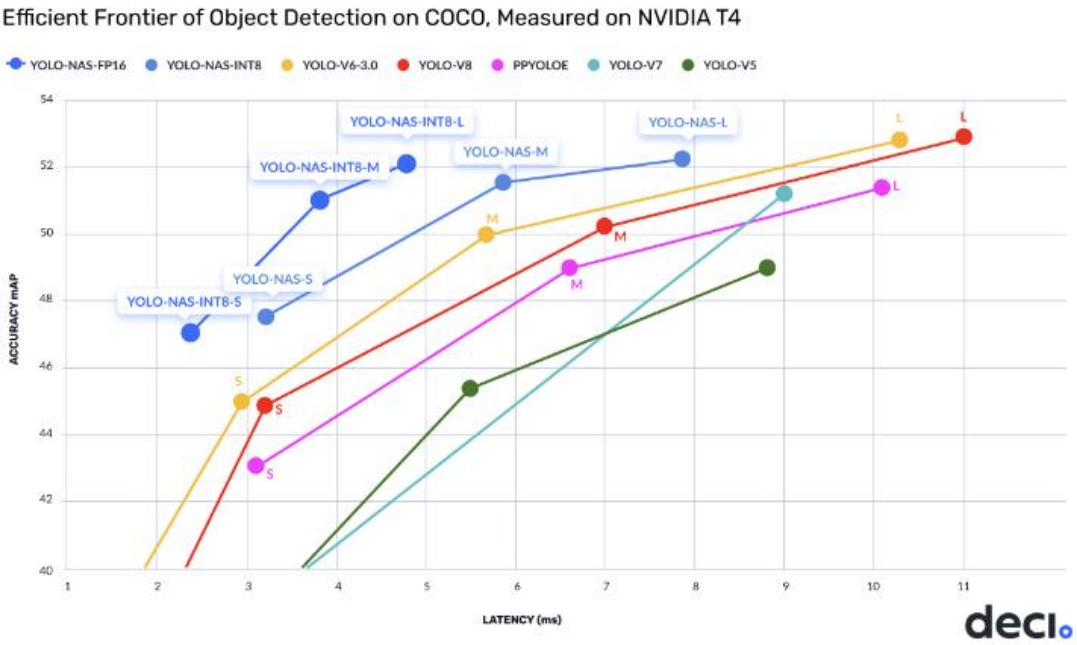

Figura 19 Comparativa YoloNas (DeciAI)

La biblioteca de YoloNas permite establecer la confianza mínima y el IoU, que se refiere a la medida de la intersección sobre la unión (Intersection over Union, IoU). Es un parámetro que controla el umbral de superposición necesario para considerar que una predicción es correcta.

La IoU se calcula como la proporción entre el área de intersección y el área de unión de dos regiones, generalmente representadas por las cajas delimitadoras predichas y las cajas delimitadoras reales de los objetos en una imagen. La expresión viene dada por:

$$IoU = \frac{Área\ de\ Intersección}{Área\ de\ Unión} \qquad (2)$$

En lo que se refiere a la implementación el parámetro **iou** en **model.predict()** se utiliza para filtrar las predicciones basándose en este criterio. Si la IoU entre una predicción y una caja delimitadora real es mayor que el valor de **iou**, se considera una detección correcta. Este valor suele establecerse en un rango de 0 a 1, siendo 0.5 un valor comúnmente utilizado.

En resumen, al establecer **iou=0.5**, estás indicando que solo se considerarán como detecciones correctas aquellas predicciones cuya IoU con las cajas delimitadoras reales sea mayor o igual a 0.5. Este umbral determina cuánta superposición debe haber entre las predicciones y las cajas reales para que la detección se considere precisa.

5 Comunicaciones Industriales

La irrupción de los autómatas programables ha representado un punto de inflexión en la industria de ensamblaje y líneas de producción, dando lugar al surgimiento de la concepción de líneas de producción automatizadas. Estos dispositivos compactos han posibilitado que los procesos industriales alcancen niveles de eficiencia y precisión superiores. Más significativamente, han permitido la reprogramación, eliminando la necesidad de costosos reemplazos inherentes al obsoleto sistema de control basado en relés y contactores, tanto en términos de tamaño como de vida útil.

No obstante, el avance en los procesos de automatización se encuentra actualmente vinculado al desarrollo de redes de comunicación. Aunque la interconexión de sistemas y procesos industriales no es una novedad, con sistemas como IEEE-488 y RS485/422 utilizados ampliamente durante más de dos décadas para satisfacer las demandas de instalaciones de baja y mediana complejidad en cuanto a capacidad de intercomunicación se refiere.

Estos enlaces, predominantemente utilizados para equipos de instrumentación y sistemas de automatización, diseñados para una baja tasa de transferencia de datos, han demostrado ser insuficientes para las crecientes necesidades de intercomunicación entre dispositivos en la actualidad. En este contexto, redes de comunicación como PROFIBUS y PROFINET han emergido como elementos indispensables en un entorno laboral donde la integración global es cada vez más esencial. En la Figura 20 se muestra un esquema de la distribución de dos ejemplos de buses de campo.

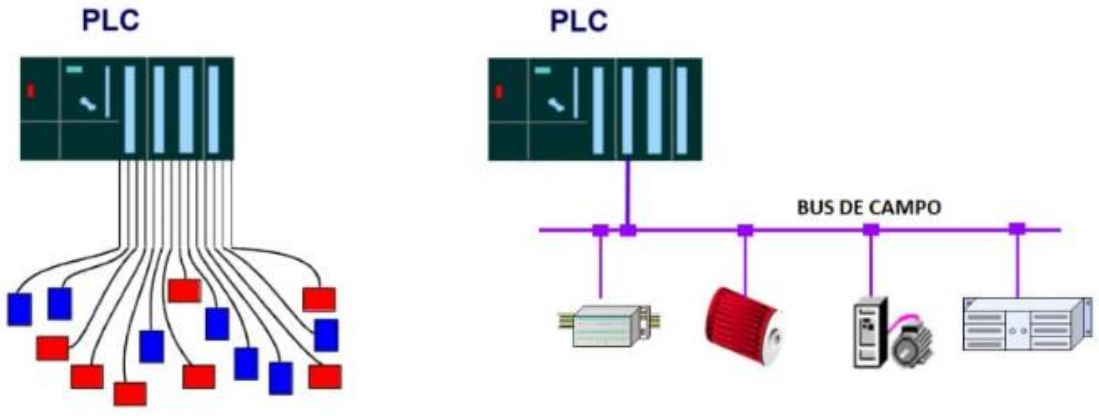

Figura 20 Ejemplo Bus de Campo (InfoPLC, 2015)

A medida que la distancia entre el instrumento y el sistema de control aumenta o cuando el proceso involucra un gran número de instrumentos, es crucial considerar los costos asociados al cableado, especialmente cuando se requiere un extenso número de conductores de reserva para futuras expansiones. Por estas razones, la filosofía de bus de campo se está consolidando como una opción definitiva en la actualidad. Este enfoque permite la substitución de voluminosos conjuntos de conductores por un único cable bifilar o de fibra óptica compartido por todos los sensores y actuadores, generando un significativo ahorro económico. La comunicación de la variable de proceso se realiza de manera completamente digital bajo este sistema. (InfoPLC, 2015)

Dentro de las comunicaciones industriales, podemos destacar las siguientes que se muestran en la Tabla 1:

BUS DE CAMPO	TOPOLOGÍA	MEDIO FÍSICO	VELOCIDAD	DISTANCIA SEGMENTO	NODOS POR SEGMENTO	ACCESO AL MEDIO
P-NET	Anillo	Par trenzado apantallado	76'8 Kbps	1.200 m	125	Paso de testigo Maestro/esclavo
PROFIBUS	Bus lineal Anillo Estrella Árbol	Par trenzado apantallado Fibra óptica	Hasta 12Mbps	Hasta 9'6 Km y 90 Km	125	Paso de testigo Maestro/esclavo
WORLDFIP	Bus lineal	Par trenzado apantallado Fibra óptica	Hasta 1 Mbps y 5Mbps	Hasta 5 Km y 20 Km	64	Arbitro de bus
HART	Bus lineal	Cable 2 hilos	1'2Kbps	3.000 m	30	Maestro/esclavo
MODBUS	Bus lineal	Par trenzado	Hasta 19'2Kbps	1 Km	248	Maestro/esclavo
INTERBUS-S	Anillo	Par trenzado	500 Kbps	400 m	256	Paso de testigo
BITBUS	Bus lineal	Par trenzado Fibra óptica	Hasta 1'5Mbps	Hasta 1.200m	29	Maestro/esclavo
CAN	Bus lineal	Par trenzado	Hasta 1 Mbps	Hasta 1.000m	127-64	CSMA/CD con arbitraje de bit
SDS	Bus lineal	Cable de 4 hilos	Hasta 1 Mbps	500 m	64	CSMA
DEVICENET	Bus lineal	Par trenzado	Hasta 500 Kbps	Hasta 500 m	64	CSMA/CDBA
CONTROLNET	Bus lineal Árbol Estrella	Coaxial Fibra óptica	5 Mbps	Hasta 3.000m	48	CTDMA
SERIPLEX	Bus lineal	Cable 4 hilos apantallado	98 Kbps	1.500m	300	Maestro/esclavo
AS-i	Bus lineal Árbol - Estrella	Cable 2 hilos	167 Kbps	Hasta 200 m	32-62	Maestro/esclavo
LON WORKS	Bus Anillo Libre	Par trenzado Fibra óptica Red eléctrica Coaxial Radio Infrarrojos	Hasta 1'25 Mbps	Hasta 2.700 m	64	CSMA/CA
ARCNET	Bus Estrella	Par trenzado Fibra óptica Coaxial	2'5 Mbps	122 m	255	Paso de testigo
M-BUS	Bus lineal	Cable 2 hilos	Hasta 9'6 Kbps	1.000 m	250	Arbitro de bus
UNI-TELWAY	Bus lineal	Par trenzado apantallado	Hasta 19'2Kbps	20 m	Hasta 28	Maestro/esclavo
COMPOBUS/S	Bus lineal	Cable de 2 ó 4 hilos	Hasta 750 Kbps	Hasta 500 m	32	Maestro/esclavo

Tabla 1 Buses de Campo (InfoPLC, 2015)

Además, de estos protocolos de comunicaciones industriales podemos encontrar la plataforma de Node-Red para la coordinación y comunicación de algunos de estos protocolos de comunicación, además de otros no industriales como webSockets, http o comunicación con distintas bases de datos.

5.1 Node Red

Node-RED es una plataforma de código abierto diseñada para la creación visual de flujos de trabajo, especialmente en el ámbito de la Internet de las cosas (IoT, *Internet of Things*). Desarrollada por IBM, Node-RED proporciona una interfaz gráfica basada en nodos que facilita la creación, conexión y despliegue de lógica de programación de manera intuitiva. Cada nodo representa una función o tarea específica, y los usuarios pueden conectarlos de manera visual para construir flujos de trabajo complejos. (NodeRed, s.f.)

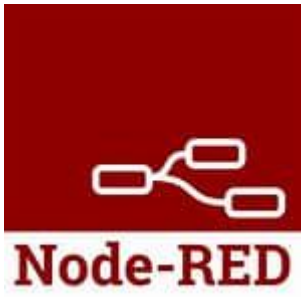

Figura 21 Logo Node-Red (NodeRed)

En el contexto de la comunicación industrial, Node-RED se utiliza como una herramienta versátil para integrar y coordinar dispositivos y sistemas heterogéneos. Su capacidad para interactuar con una amplia gama de protocolos de comunicación, como MQTT, HTTP, Modbus y otros, lo convierte en una elección popular para implementaciones en entornos industriales.

El uso de Node-RED en comunicación industrial implica la creación de flujos de trabajo que facilitan la interconexión y el intercambio de datos entre dispositivos y sistemas en una red industrial. Puede servir como un middleware eficaz para facilitar la comunicación entre PLCs (Controladores Lógicos Programables), sensores, actuadores y otros componentes industriales. Además, su interfaz gráfica facilita la adaptación de lógica y reglas comerciales, lo que resulta beneficioso en entornos industriales en constante cambio.

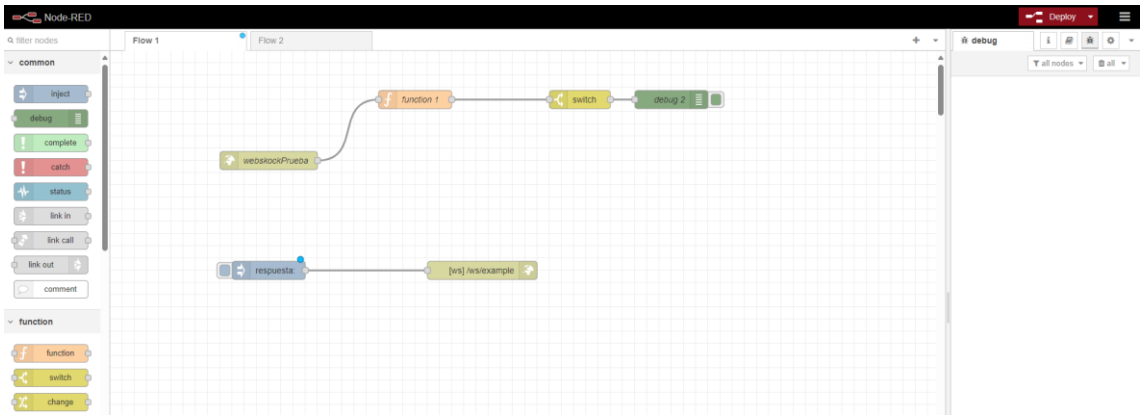

Figura 22 Ejemplo flujo de Node Red (Elaboración propia)

Node-RED también es compatible con la implementación de servicios basados en la nube, lo que permite la integración de soluciones industriales en entornos de computación en la nube, es una programación en flujo como se muestra en la Figura 22 que permite una programación rápida y sencilla. La gran cantidad servicios que permite comunicar, va a facilitar la comunicación entre el Relé Programable y la detección de objetos con YOLO.

5.2 Relés programables

Los relés programables representan una evolución en el ámbito de la automatización industrial, permitiendo la ejecución de lógica programable en lugar de depender exclusivamente de circuitos electrónicos cableados. Estos dispositivos combinan la funcionalidad de relés tradicionales con capacidades de programación, brindando flexibilidad y adaptabilidad a los sistemas de control.

En este contexto, Siemens LOGO 8 emerge como una destacada solución de relé programable. Diseñado por Siemens AG, LOGO 8 se destaca por su versatilidad y facilidad de programación. Este relé programable es ideal para aplicaciones en entornos industriales y comerciales de pequeña y mediana escala.

Figura 23 Logo 8 (Siemens)

Características de Siemens LOGO 8:

- **Programación Intuitiva:** LOGO 8 ofrece un entorno de desarrollo de software intuitivo que facilita la creación de lógica de control. Su interfaz gráfica permite a los usuarios diseñar programas de control mediante la conexión visual de bloques de funciones.
- **Conectividad:** Equipado con múltiples puertos de comunicación, LOGO 8 puede integrarse fácilmente con otros dispositivos y sistemas. Admite protocolos comunes como Modbus, facilitando la comunicación con otros equipos industriales.
- **Amplias Funcionalidades:** A pesar de su tamaño compacto, LOGO 8 proporciona una amplia gama de funciones, incluyendo temporizadores, contadores, funciones matemáticas y de comparación. Esto lo convierte en una solución robusta para diversas aplicaciones.
- **Flexibilidad de Entradas/Salidas:** LOGO 8 cuenta con entradas y salidas configurables, permitiendo a los usuarios adaptar el dispositivo a sus necesidades específicas. Esto facilita la conexión con sensores, actuadores y otros componentes del sistema.
- **Monitoreo Remoto:** La capacidad de monitoreo remoto y la posibilidad de conectividad a través de interfaces de red permiten a los usuarios supervisar y controlar los procesos a distancia, mejorando la eficiencia operativa.
- **Aplicaciones Prácticas:**
- LOGO 8 encuentra aplicaciones en diversas industrias, como control de acceso, sistemas de iluminación, automatización de pequeñas máquinas, sistemas de riego, entre otros. Su diseño compacto y su capacidad de programación intuitiva hacen que sea una opción popular para proyectos donde se requiere flexibilidad y control preciso.

6 Recursos y materiales utilizados

En este apartado se describen e identifican los diferentes sistemas utilizados, tanto la parte de hardware como el software necesario para la realización de la detección de objetos y aplicación.

6.1 Python 3.9

Para el desarrollo de la detección de objetos se utilizael lenguaje Python en la versión 3.9. La elección de Python se fundamenta en su versatilidad, extensibilidad y la disponibilidad de bibliotecas especializadas que lo posicionan como uno de los mejores lenguajes para abordar tareas complejas como la detección de objetos.

Se ha elegido este lenguaje debido a su sintaxis clara y legible facilita la implementación y comprensión de algoritmos complejos, como los utilizados en la detección de objetos, entre ellos la detección de objetos con YOLO. La implementación de la detección de objetos con YOLO se basa en la utilización de bibliotecas de ultralitycs (Ultralytics) y superGradients (DeciAI). Ultralytics es una biblioteca en Python que proporciona herramientas y utilidades para la investigación en aprendizaje profundo. Es especialmente conocida por su implementación de YOLOv5 y de YOLOv8. SuperGradients es otra biblioteca que se centra en optimizar y acelerar el entrenamiento de modelos de aprendizaje profundo, incluidos los utilizados en la detección de objetos, que se utiliza para el algoritmo de YOLONAS. Ambas bibliotecas ofrecen soluciones eficientes y simplifican la implementación de modelos de detección de objetos, brindando funcionalidades adicionales y facilitando la experimentación.

Python cuenta con una comunidad activa y gran cantidad de recursos en github en el ámbito de la IA y la visión por computadora. Esto se traduce en una abundancia de recursos, tutoriales y documentación disponibles para ayudar en el desarrollo de sistemas de detección de objetos. Para la realización de la aplicación se han utilizado las siguientes bibliotecas que se muestran en la siguiente Tabla 2.

Bibliotecas utilizadas	
Matplotlib	3.8.1
pandas	2.1.2
Opencv-python	4.8.1.78
Torch	2.2.0+cu121
Super-gradients	3.5.0
Ultralytics	8.1.10
Websockets	12.0

Tabla 2 Bibliotecas utilizadas

Destacando estas dos bibliotecas, ultralytics u super-gradients que usaremos para el desarrollo de la detección de objetos. Se utilizará la biblioteca ultralytics para los modelos de Yolov5 y Yolov8 y la biblioteca super-gradients para la implementación del modelo con YoloNas.

Algunos errores que han surgido en la instalación de las librerías en un sistema de Windows, es que es necesario tener instalado Microsoft Visual C++ para poder ejecutar la biblioteca de super-gradients en Windows.

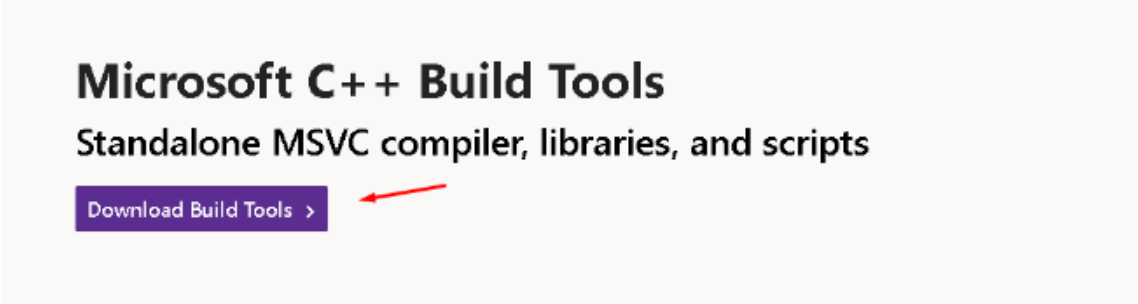

Figura 24 Microsoft C++ Build Tools (Microsoft)

6.2 Pycharm

Se ha elegido PyCharm que es un entorno de desarrollo integrado (IDE) diseñado específicamente para la programación en Python. Destaca como una opción líder en el ámbito de desarrollo Python debido a sus características robustas y su interfaz intuitiva. A continuación, se proporciona un resumen de PyCharm y las razones para considerarlo como la mejor opción para la realización del presente trabajo:

- **Interfaz amigable:** ofrece una interfaz de usuario limpia y fácil de usar, lo que facilita la navegación y la escritura de código de manera eficiente.
- **Autocompletado y sugerencias:** La función de autocompletado inteligente y las

sugerencias de código en tiempo real aumentan la productividad al reducir errores y acelerar el proceso de desarrollo.

- **Análisis estático:** realiza análisis estático del código, identificando posibles errores y ofreciendo recomendaciones para mejorar la calidad del código.
- **Herramientas de depuración avanzadas:** ofrece potentes herramientas de depuración que facilitan la identificación y resolución de problemas en el código, mejorando la eficiencia del proceso de desarrollo.
- **Integración con herramientas de construcción y entornos virtuales:** se integra perfectamente con herramientas de construcción como *setuptools* y *pip*, además de admitir la gestión de entornos virtuales para proyectos Python.
- **Soporte para marcos y bibliotecas:** Proporciona soporte nativo para una amplia variedad de marcos y bibliotecas de Python, facilitando el desarrollo en entornos diversos.
- **Control de versiones integrado:** La integración con sistemas de control de versiones como Git permite un seguimiento fácil de los cambios y la colaboración eficiente en proyectos.
- **Desarrollo web integrado:** ofrece funcionalidades de desarrollo web, lo que lo hace adecuado para proyectos que involucran tanto el *backend* como el *frontend*.

Figura 25 Logo Pycharm (Jetbrains)

6.3 Datasets Utilizados

La implementación de los algoritmos de la detección de personas con YOLO se va realiza con modelos preentrenados con los datasets COCO y Open Imagen V7

6.3.1 COCO2022

COCO 2022 (Lin, et al., 2014) es un conjunto de datos de detección de objetos de vanguardia

publicado en 2022 por el grupo de investigación de visión artificial de la Universidad de Stanford. COCO 2022 contiene más de 2 millones de imágenes y 1.5 millones de objetos etiquetados, lo que lo convierte en uno de los conjuntos de datos de detección de objetos más grandes y diversos del mundo. (COCO Datasets, s.f.)

COCO 2022 tiene una serie de características que lo convierten en un conjunto de datos de detección de objetos de vanguardia:

- Tamaño: contiene más de 2 millones de imágenes y 1.5 millones de objetos etiquetados. Esto lo convierte en uno de los conjuntos de datos de detección de objetos más grandes y diversos del mundo.
- Diversidad: contiene una amplia gama de objetos y escenas. Esto permite a los investigadores desarrollar algoritmos de detección de objetos que sean precisos en una amplia gama de aplicaciones.
- Precisión: Los datos de COCO 2022 están etiquetados con precisión. Esto permite a los investigadores evaluar el rendimiento de sus algoritmos de detección de objetos de manera precisa.
- Herramientas: COCO 2022 incluye un conjunto de herramientas para el desarrollo de algoritmos de detección de objetos. Estas herramientas facilitan a los investigadores la creación y evaluación de sus algoritmos.

6.3.2 Open-Images V7

Open Images V7 (Kuznestova, et al., 2021) es una base de datos de imágenes a gran escala que sirve como recurso fundamental en la investigación de visión por computadora y aprendizaje profundo. A continuación, se presenta un resumen de sus características clave: (Open Images V7, s.f.)

- Open Images V7 alberga un extenso conjunto de datos que consiste en millones de imágenes, lo que lo convierte en una de las bases de datos más grandes y diversificadas disponible en la actualidad.
- Cada imagen en Open Images V7 está anotada con información detallada sobre objetos y escenas presentes en la imagen. Las anotaciones incluyen etiquetas de clases, coordenadas de cuadros delimitadores y atributos específicos.
- La base de datos cubre una amplia variedad de clases que abarcan objetos, animales, personas y escenas complejas. La inclusión de una amplia gama de categorías facilita la investigación en diversas aplicaciones de visión por computadora.
- Open Images V7 se distribuye bajo una licencia abierta, permitiendo su uso gratuito

para propósitos de investigación, desarrollo y aplicación en proyectos académicos y comerciales.

- Open Images V7 incluye conjuntos de datos específicos para desafíos de detección de objetos, lo que permite a los investigadores y profesionales evaluar y comparar algoritmos y modelos de detección de manera estandarizada.
- Las imágenes en Open Images V7 se capturan en diversas condiciones de iluminación, fondos y orientaciones, proporcionando una variedad que refleja las condiciones del mundo real y desafía la robustez de los modelos de visión por computadora.
- Open Images V7 se ha convertido en un recurso esencial para la investigación en campos como reconocimiento de objetos, detección de instancias, segmentación semántica y otros temas relacionados con la visión por computadora.

6.4 Siemens Logo

Para la comprobación de la comunicación entre Node-Red y el autómata que controlara los elementos de disuasión, luz y sonido, de la parte exterior de los edificios del centro es un Siemens Logo 8.2.

6.5 Ordenador utilizado

Para la realización del proyecto fin de máster he utilizado un portátil MSI Pulse 17 con las siguientes características:

- CPU: Intel® Core™ i7-13700H
- GPU: NVIDIA® GeForce RTX™ 4070
- Memoria de Video: GDDR6 8GB
- Memoria RAM: DDR5 16GB*2
- Almacenamiento: 1TB NVMe PCIe SSD Gen4x4 w/o DRAM

Figura 26 Imagen MSI Pulse (MSI)

7 Resultados

Dentro del desarrollo práctico el análisis de resultados se divide en tres grandes bloques:

- La detección de objetos con YOLO.
    - Imágenes.
    - Videos.
    - Tiempo Real.
- Programación y cableado del relé programable.
- Comunicación del Relé Programable y la detección con YOLO.

7.1 Detección de objetos con YOLO

Es la primera parte práctica realizada en el trabajo, consta desde la detección de personas en una imagen hasta la detección de personas en tiempo real. Este análisis realiza en cada una de las arquitecturas de YOLO para visualizar los resultados y determinar si consideramos aceptables los resultados, para poder aplicar la arquitectura a la aplicación real.

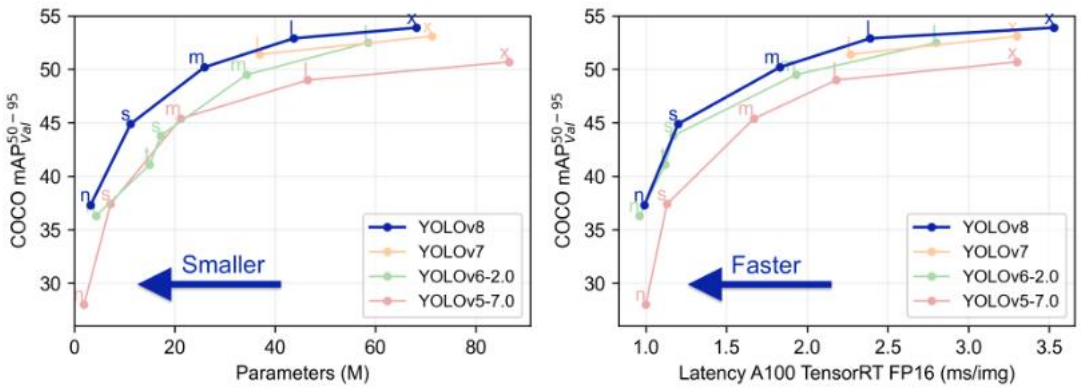

Figura 27 Comparatica Modelos YOLO (ultralytics)

En la Figura 27 se observa que a mayor mAPval, (*mean Average Precision)* en el conjunto de datos de validación mayor rendimiento del modelo y más lento es el proceso. Por lo que es muy importante elegir el modelo adecuado que mejor se adapte a nuestro sistema que queremos desarrollar.

Para la implementación de los modelos se ha seguido la documentación de docs.ultralytics y docs.deci:

- Ultralytics: https://docs.ultralytics.com/
- Deci AI: https://docs.deci.ai/

En este trabajo se va a realizar un primer análisis de distintas imágenes y videos, desde imágenes de buena calidad, a imágenes de cámaras de seguridad para posteriormente analizar los modelos con videos. Durante la realización de este trabajo no se va a tener un acceso al sistema de videovigilancia del centro por lo que será un análisis del modelo que se escogerá en un futuro.

La selección de estas imágenes y video sirven para analizar el comportamiento desde una imagen normal a una imagen de cámara de seguridad hasta analizar un video de cámara de seguridad de baja calidad. El número de imágenes utilizadas para certificar el modelo es relativamente bajo teniendo en cuenta que los modelos basados en redes neuronales convolucionales demandan un número alto de imágenes, hasta llegar incluso a millones. No obstante, las imágenes seleccionadas en este trabajo son suficientemente representativas de un conjunto mucho más amplio, que incluiría las mencionadas miles de imágenes similares en un entorno de videovigilancia. Por ejemplo, la tercera imagen y el video simulan las condiciones reales de vigilancia, donde destacan como características esenciales la calidad de imagen relativamente baja y los ángulos de visión típicos de estas cámaras.

Los modelos YOLO utilizados en este estudio están previamente pre-entrenados y han sido validados suficientemente por la comunidad científica en tareas de detección de objetos en diversas aplicaciones. Estos modelos han demostrado su eficacia en conjuntos de datos extensos y variados, lo que asegura que su rendimiento sea robusto y generalizable. Esta consideración permite llegar al objetivo principal de este trabajo, que no es otro que demostrar la viabilidad de estos modelos para su utilización en un sistema de videovigilancia, en este caso en un Instituto, con la finalidad de mejorar los sistemas existentes. Los resultados obtenidos, aunque basados en un número limitado de imágenes y videos, como se ha indicado previamente, demuestran que estos modelos pueden ser utilizados para la detección de personas con cámaras de videovigilancia de uso común existentes en el mercado. En este sentido, cabe añadir que el análisis de los resultados llevado a cabo en este trabajo es válido y extensible a imágenes de características similares en entornos de exterior y con los fines de videovigilancia pretendidos.

7.1.1 YOLOv5

Se ha comenzado con la arquitectura de YOLOv5, lo primero que se ha hecho es comprobar que se detectan personas con un datasets preentrenado para ello se utilizará uno de los siguientes modelos preentrenados en COCO 2022 (Lin, et al., 2014). En la Figura 28

se muestran los distintos modelos preentreandos remendados por ultralytics.

Modelo	YAML	tamaño (píxeles)	mAPval 50-95	Velocidad CPU ONNX (ms)	Velocidad A100 TensorRT (ms)	parámetros (M)
yolov5nu.pt	yolov5n.yaml	640	34.3	73.6	1.06	2.6
yolov5su.pt	yolov5s.yaml	640	43.0	120.7	1.27	9.1
yolov5mu.pt	yolov5m.yaml	640	49.0	233.9	1.86	25.1
yolov5lu.pt	yolov5l.yaml	640	52.2	408.4	2.50	53.2
yolov5xu.pt	yolov5x.yaml	640	53.2	763.2	3.81	97.2
yolov5n6u.pt	yolov5n6.yaml	1280	42.1	211.0	1.83	4.3
yolov5s6u.pt	yolov5s6.yaml	1280	48.6	422.6	2.34	15.3
yolov5m6u.pt	yolov5m6.yaml	1280	53.6	810.9	4.36	41.2
yolov5l6u.pt	yolov5l6.yaml	1280	55.7	1470.9	5.47	86.1
yolov5x6u.pt	yolov5x6.yaml	1280	56.8	2436.5	8.98	155.4

Figura 28 Modelos Preentrenados de Yolov5 en COCO

En la figura 28 se muestran las métricas de rendimiento de diferentes modelos de detección de objetos YOLO con los siguientes parámetros

- **Modelo:** el nombre del modelo de detección de objetos YOLOv5
- **Tamaño:** el tamaño del modelo, que puede ser indicado por el número de capas, el número total de parámetros o alguna otra métrica relacionada con la complejidad del modelo.
- **mAPval:** la métrica mAP (*Mean Average Precision*) que evalúa el rendimiento del modelo en términos de precisión de detección. Cuanto mayor sea el valor de mAP, mejor será el rendimiento del modelo.
- **Velocidad en CPU (ONNX):** la velocidad de inferencia del modelo en una CPU utilizando el formato ONNX (*Open Neural Network Exchange*). Esta métrica indica cuánto tiempo tarda el modelo en realizar inferencias en una CPU.
- **Velocidad en A100 (TensorRT):** la velocidad de inferencia del modelo en un acelerador de GPU A100 utilizando el motor de inferencia TensorRT. Esta métrica indica cuánto tiempo tarda el modelo en realizar inferencias en una GPU específica.
- **Parámetros:** el número total de parámetros entrenables en el modelo. Esta métrica proporciona información sobre la complejidad del modelo y su capacidad para capturar información durante el entrenamiento.

Para la realización de la prueba utilizan los modelos yolov5nu.pt y el modelo yolov5m6u.pt. El código utilizado para la detección de Objetos se encuentra en el Anexo I, y las imágenes de prueba usadas con todos los modelos son las siguientes:

Figura 29 Imagen Prueba bus (Github/Ultralitycs/Yolo)

Figura 30 Imagen Prueba Personas y móvil (PacoCara, 2023)

Una vez comprobada la eficacia del modelo para una imagen con gran definición, se comprueba si también es óptimo para una imagen más pixelada como sería la de una cámara de seguridad de un sistema de videovigilancia, como la mostrada en la Figura 31. De manera que se simule la respuesta al no tener acceso al mismo.

Figura 31 Imagen de cámara de seguridad (Infokrause, 2018)

Para el análisis del video para posteriormente conectar la webcam o cámara exterior para realizar la detección se ha utilizado el video de Yotube (Paso de peatones inteligente Cambrils - Smart pedestrian crossing Cambrils (youtube.com)) para que se pueda testear este trabajo con los mismos videos que se han utilizado para realizarlo. En la Figura 32 se muestra una captura del video seleccionado para realizar la detección de personas.

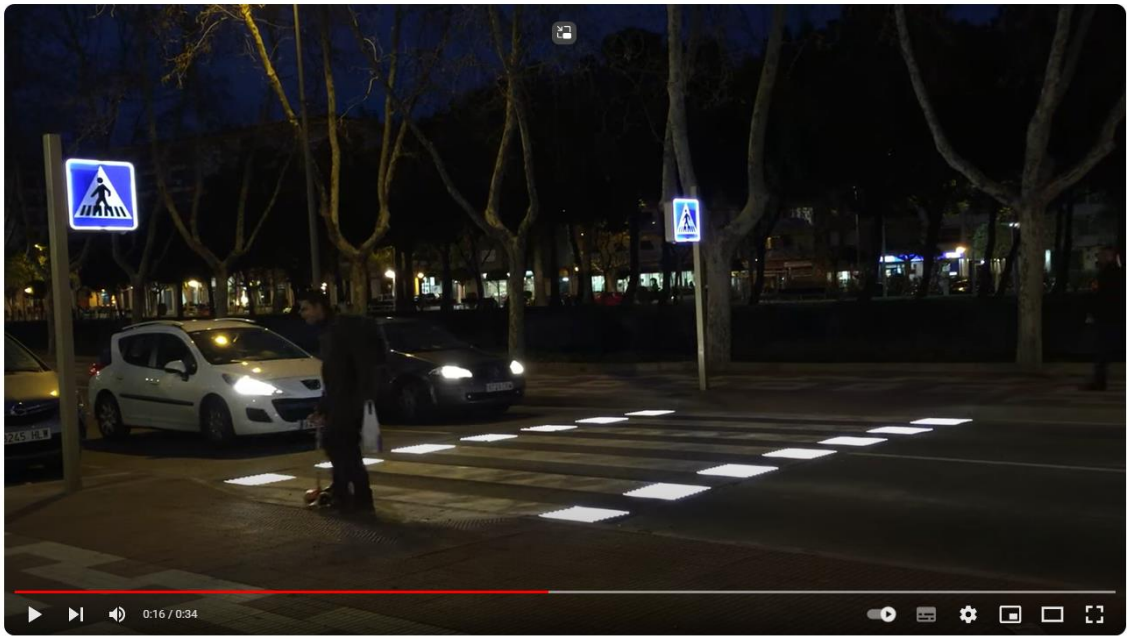

Figura 32 Captura Video Paso Peatones

Para el análisis de los diferentes modelos de Yolo se valorará el **tiempo de inferencia** que se refiere al período durante el cual un modelo de detección procesa una imagen de entrada y produce predicciones sobre la presencia y ubicación de objetos en esa imagen. Es el tiempo que lleva al modelo analizar la información visual y generar resultados.

7.1.1.1 Modelo yolov5nu.pt

7.1.1.1.1 Imagen

Aplicando este modelo se han obtenido los siguientes resultados de la implementación de YOLOv5n en Python. A continuación, se realiza un resumen de los datos proporcionados:

- **Arquitectura del modelo (Red Neuronal):**
  - **Capas:** el modelo tiene 262 capas en total.
  - **Parámetros:** hay 2,654,816 parámetros en el modelo.
- **Inferencia en una imagen específica:**
  - **Entrada:** la imagen de entrada es de tamaño 448x448 píxeles.
  - **Objetos detectados en la imagen:**
    - En la primera inferencia se detectaron 3 personas y 1 bus.
    - En la segunda inferencia se detectaron 8 personas y 1 mochila (*backpack*).
  - **Tiempo de inferencia:** la inferencia en cada imagen lleva 153.0 milisegundos.
- **Velocidades detalladas:**
  - **Preprocesamiento:** toma 2.7 milisegundos por imagen.
  - **Inferencia:** toma 76.5 milisegundos por imagen.
  - **Postprocesamiento:** toma 44.8 milisegundos por imagen.
  - **En total:** la imagen completa (desde la carga hasta la obtención de resultados) lleva 153.0 milisegundos.
- **Rendimiento General:**
  - **GFLOPs (Operaciones de Punto Flotante por Segundo):** el modelo realiza 7.8 GFLOPs durante la inferencia en una imagen.

*GFLOPs, que significa "*Giga FLoating-point Operations Per Second*" (Operaciones en Punto Flotante por Segundo en gigaoperaciones), es una medida de rendimiento utilizada para evaluar la capacidad de cálculo de una unidad de procesamiento, como una CPU o GPU, al realizar operaciones en punto flotante. En el contexto de modelos de redes neuronales, la cantidad de GFLOPs proporciona una indicación de la carga computacional requerida para realizar inferencias en una red neuronal específica. Cuantos más GFLOPs tenga un modelo, mayor será la cantidad de operaciones de punto flotante que debe realizar durante la inferencia, lo que generalmente se traduce en una mayor demanda computacional.

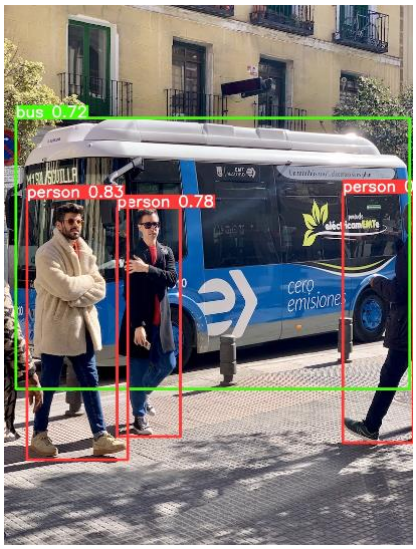

Figura 33 Detección bus Yolov5nu

Figura 34 Detección personas Yolo5nu

7.1.1.1.2 Imagen cámara seguridad

El resumen indica que el modelo YOLOv5n tiene 262 capas y 2,654,816 parámetros. En la inferencia de una imagen de tamaño 448x640, se detectó una persona. El tiempo de inferencia total es de 95.7 milisegundos, con 3.0 milisegundos dedicados al preprocesamiento y 61.2 milisegundos al postprocesamiento. El modelo tiene un rendimiento de 7.8 GFLOPs durante la inferencia en una imagen.

Figura 35 Detección cámara seguridad yolo5n

7.1.1.1.3 Video

El código del video para la detección de video se encuentra en el Anexo II, obteniendo los siguientes resultados:

- Media de inferencia: 16.98 ms
- Máximo de 'postprocess': 96.21 ms
- Máximo de 'preprocess': 9.76 ms

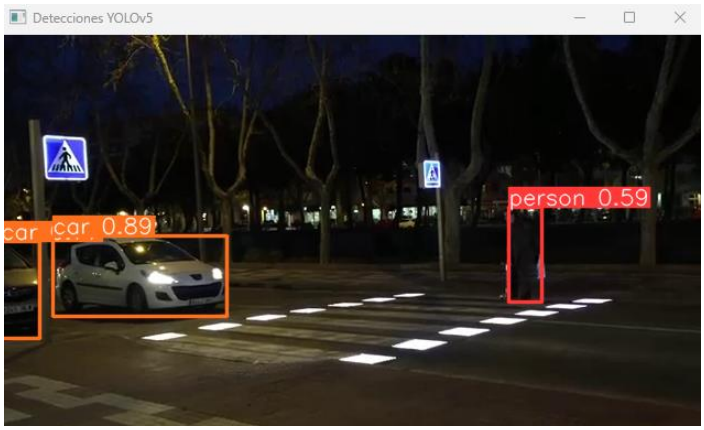

Figura 36 Detección en video Yolo5nu

7.1.1.2 Modelo yolov5m6u.pt

7.1.1.2.1 Imagen

- **Arquitectura del modelo (Red Neuronal):**
  - **Capas**: el modelo tiene 442 capas en total.
  - **Parámetros**: hay 41,222,064 parámetros en el modelo.
  - **Gradients**: no hay gradientes disponibles en estos resultados.
  - **GFLOPs**: el modelo realiza 65.7 GFLOPs durante la inferencia en una imagen.

- **Inferencia en una imagen específica:**
    - **Entrada**: la imagen de entrada es de tamaño 448x448 píxeles.
    - **Objetos detectados en la imagen:**
        - En la primera inferencia se detectaron 4 personas y 1 bus.
        - En la segunda inferencia se detectaron 8 personas, 1 banco (bench), 1 mochila (backpack), 2 bolsos de mano (handbags).
    - **Tiempo de inferencia**: La inferencia en cada imagen lleva 209.9 milisegundos.
- **Velocidades detalladas:**
    - **Preprocesamiento**: toma 2.5 milisegundos por imagen.
    - **Inferencia**: toma 105.0 milisegundos por imagen.
    - **Postprocesamiento**: toma 43.8 milisegundos por imagen.
    - **En total**: la imagen completa (desde la carga hasta la obtención de resultados) lleva 209.9 milisegundos.

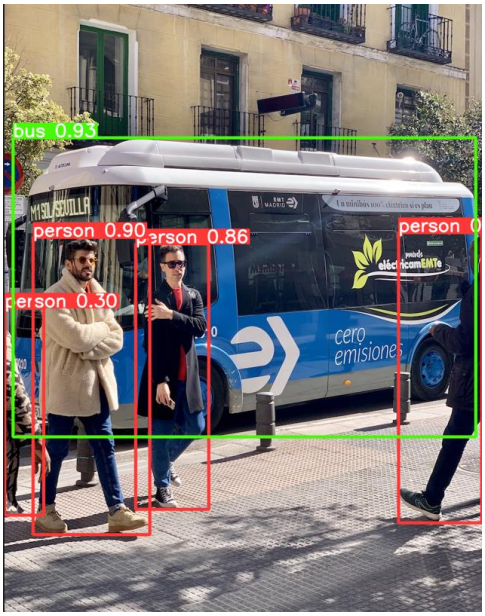

Figura 37 Detección bus Yolov5m6u

Figura 38 Detección personas Yolo5m6u

7.1.1.2.2 Imagen cámara seguridad

El modelo YOLOv5m6u tiene 442 capas y 41,222,064 parámetros. En la inferencia de una imagen de tamaño 896x1280, se detectó una persona. El tiempo de inferencia total es de 176.5 milisegundos, con 7.1 milisegundos dedicados al preprocesamiento y 157.1 milisegundos al postprocesamiento. El modelo tiene un rendimiento de 65.7 GFLOPs durante la inferencia en una imagen.

Figura 39 Detección cámara seguridad yolo5m6u

7.1.1.2.3 Video

Media de inferencia: 24.29 ms

Máximo de 'postprocess': 83.24 ms

Máximo de 'preprocess': 19.06 ms

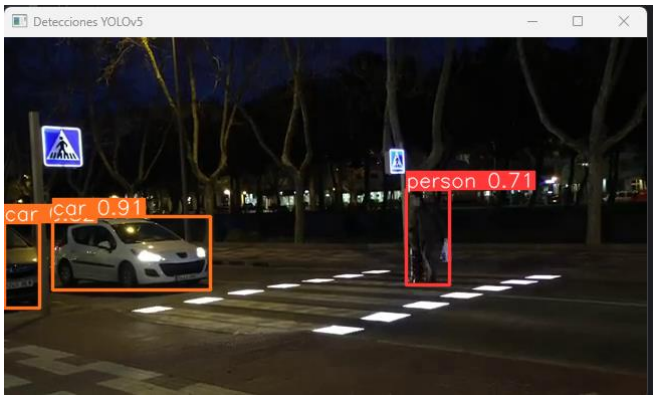

Figura 40 Detección en video Yolo5m6u

7.1.1.3 Modelo yolov5x6u.pt

7.1.1.3.1 Imagen

- **Arquitectura del modelo (Red Neuronal):**
  - **Capas**: el modelo tiene 638 capas en total.
  - **Parámetros**: hay 155,544,432 parámetros en el modelo.
  - **Gradients**: no hay gradientes disponibles en estos resultados.
  - **GFLOPs**: el modelo realiza 251.7 GFLOPs durante la inferencia en una imagen.
- **Inferencia en una imagen específica:**
  - **Entrada**: La imagen de entrada es de tamaño 448x448 píxeles.
  - **Objetos detectados en la imagen:**
    - En la primera inferencia se detectaron 4 personas y 1 bus.
    - En la segunda inferencia se detectaron 8 personas, 3 bancos (benchs), 1 mochila (backpack), 1 sombrilla (umbrella), 1 bolso de mano (handbag), 3 teléfonos celulares (cell phones).
  - **Tiempo de inferencia**: La inferencia en cada imagen lleva 405.0 milisegundos.
- **Velocidades detalladas:**
  - **Preprocesamiento**: toma 3.5 milisegundos por imagen.
  - **Inferencia**: toma 202.5 milisegundos por imagen.
  - **Postprocesamiento**: toma 47.8 milisegundos por imagen.
  - **En total**: la imagen completa (desde la carga hasta la obtención de resultados) lleva 405.0 milisegundos.

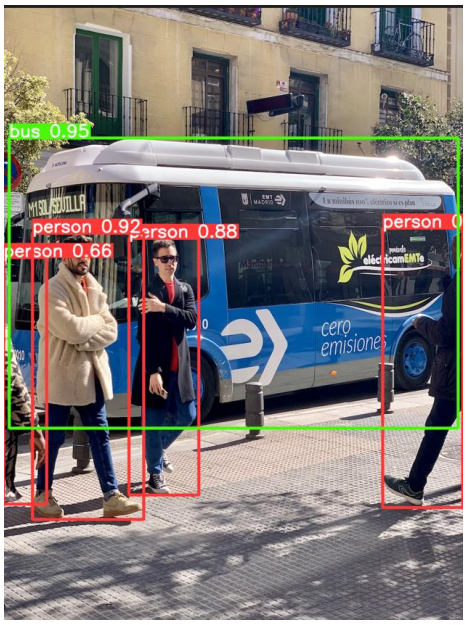

Figura 41 Detección bus Yolov5x6u

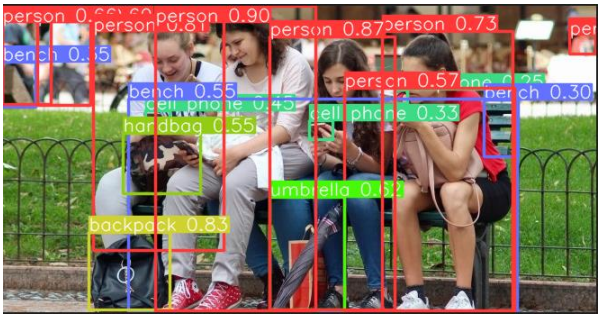

Figura 42 Detección personas Yolo5x6u

7.1.1.3.2 Imagen cámara seguridad

El modelo YOLOv5x6u tiene 638 capas y 155,544,432 parámetros. En la inferencia de una imagen de tamaño 896x1280, se detectó una persona. El tiempo de inferencia total es de 232.6 milisegundos, con 6.3 milisegundos dedicados al preprocesamiento y 148.4 milisegundos al postprocesamiento. El modelo tiene un rendimiento de 251.7 GFLOPs durante la inferencia en una imagen.

Figura 43 Detección cámara seguridad yolo5x6u

7.1.1.3.3 Video

Media de inferencia: 69.56 ms

Máximo de 'postprocess': 57.96 ms

Máximo de 'preprocess': 60.18 ms

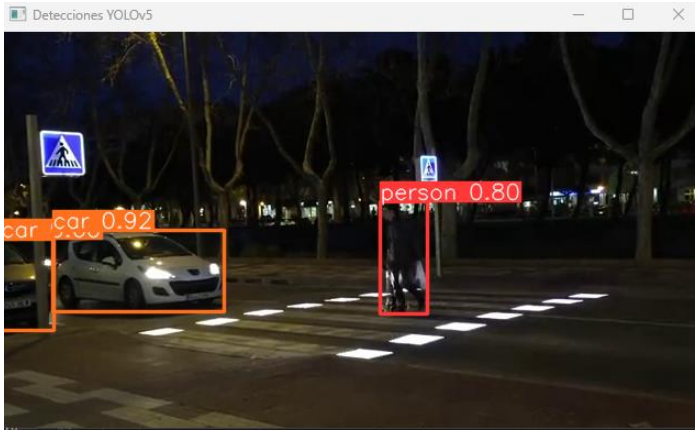

Figura 44 Detección en video Yolo5x6u

7.1.1.4 Resultados Yolov5

Tras la realización de una evaluación exhaustiva de los modelos YOLOv5, centrándose en las variantes N, M y X, con un enfoque particular en la detección de personas en cámaras de seguridad. Los principales hallazgos y conclusiones se resumen de la siguiente manera:

- **Impacto del volumen de datos en la eficiencia:**

Se ha demostrado de manera consistente que a medida que aumenta el volumen de datos del modelo YOLOv5, también se incrementan los tiempos de inferencia, así como los tiempos de preprocesado y postprocesado asociados. Estos resultados subrayan la importancia de considerar cuidadosamente la complejidad del modelo en relación con los requisitos de recursos

disponibles.

- **Comparación entre modelos N, M y X:**

La evaluación comparativa de los modelos N, M y X ha revelado diferencias sustanciales en términos de rendimiento. Aunque la complejidad de los modelos puede traducirse en una mayor precisión, se ha destacado la necesidad de encontrar un equilibrio adecuado, especialmente en aplicaciones específicas como la detección de personas en cámaras de seguridad.

- **Requisitos de Hardware y ligereza del modelo:**

La conclusión principal apunta a la importancia de elegir modelos que sean lo suficientemente ligeros como para ser ejecutados eficientemente en hardware con limitaciones de recursos. La adaptación de modelos más ligeros puede ser esencial para garantizar un rendimiento óptimo en entornos donde los requisitos de hardware son menos robustos.

- **Consideraciones prácticas para implementación en tiempo real:**

Se sugiere la inclusión de la opción ***stream = True*** como una medida práctica para mitigar problemas de memoria durante la implementación en tiempo real. Esta adición puede contribuir significativamente a la eficiencia y estabilidad del sistema.

7.1.2 YOLOV8

La serie YOLOv8 ofrece una diversa gama de modelos, cada uno especializado para tareas específicas en visión por ordenador. Estos modelos están diseñados para satisfacer distintos requisitos, desde la detección de objetos hasta tareas más complejas como la segmentación de instancias, la detección de poses/puntos clave, la detección de objetos orientados y la clasificación. En la Figura 45 se muestran los modelos de Yolov8 en función de la tarea a realizar.

Modelo	Nombres de archivo	Tarea	Inferencia	Validación	Formación	Exportar
YOLOv8	`yolov8n.pt` `yolov8s.pt` `yolov8m.pt` `yolov8l.pt` `yolov8x.pt`	Detección	✅	✅	✅	✅
YOLOv8-seg	`yolov8n-seg.pt` `yolov8s-seg.pt` `yolov8m-seg.pt` `yolov8l-seg.pt` `yolov8x-seg.pt`	Segmentación de instancias	✅	✅	✅	✅
YOLOv8-pose	`yolov8n-pose.pt` `yolov8s-pose.pt` `yolov8m-pose.pt` `yolov8l-pose.pt` `yolov8x-pose.pt` `yolov8x-pose-p6.pt`	Pose/Puntos clave	✅	✅	✅	✅
YOLOv8-obb	`yolov8n-obb.pt` `yolov8s-obb.pt` `yolov8m-obb.pt` `yolov8l-obb.pt` `yolov8x-obb.pt`	Detección Orientada	✅	✅	✅	✅
YOLOv8-cls	`yolov8n-cls.pt` `yolov8s-cls.pt` `yolov8m-cls.pt` `yolov8l-cls.pt` `yolov8x-cls.pt`	Clasificación	✅	✅	✅	✅

Figura 45 Modelos Yolov8

La Figura 45 proporciona una visión detallada de las variantes del modelo YOLOv8, destacando su aplicabilidad en tareas de detección de objetos y su compatibilidad con diversos modos operativos, como Inferencia, Validación, Entrenamiento y Exportación. Este completo soporte garantiza que los usuarios puedan aprovechar plenamente las capacidades de los modelos YOLOv5u en una amplia gama de escenarios de detección de objetos.

El código utilizado para probar los diferentes valores de yolov8 es el mismo que el utilizado con yolov5, ya que ambos utilizan la misma biblioteca de ultralytics.

7.1.2.1 Modelo yolov8n

7.1.2.1.1 Imagen

- **Inferencia en una imagen específica:**
  - En la primera inferencia (0), se detectaron 4 personas y 1 autobús (bus) en una imagen de tamaño 448x448 píxeles.
  - En la segunda inferencia (1), se detectaron 8 personas y 1 mochila (backpack) en otra imagen de tamaño 448x448 píxeles.
  - **El tiempo total** para realizar estas inferencias fue de 108.3 milisegundos.
- **Velocidades detalladas:**

- **Tiempo de preprocesamiento**: toma 1.5 milisegundos por imagen.
- **Tiempo de inferencia:** toma 54.2 milisegundos por imagen.
- **Tiempo de postprocesamiento**: toma 29.2 milisegundos por imagen.
- **En total**, el proceso completo (desde la carga hasta la obtención de resultados) para cada imagen lleva 108.3 milisegundos.

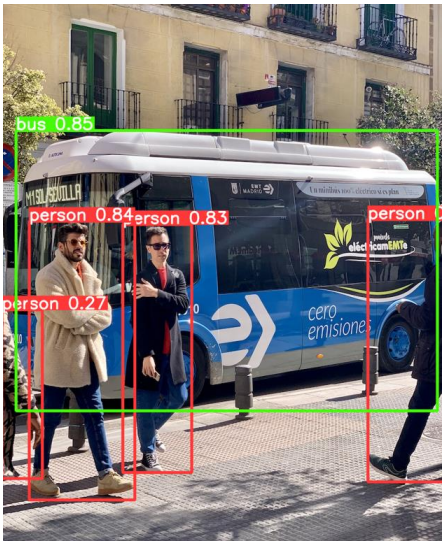

Figura 46 Detección bus Yolov8n

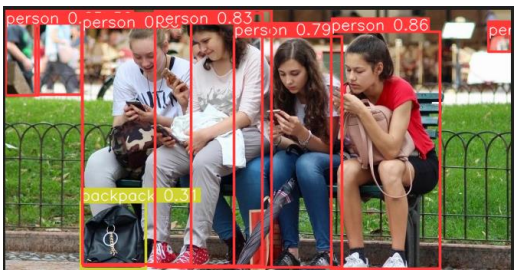

Figura 47Detección personas Yolo8n

7.1.2.1.2 Imagen cámara seguridad

El modelo YOLOv8n consta de 225 capas y tiene un total de 3,157,200 parámetros entrenables. Durante la inferencia en una imagen, el modelo realiza aproximadamente 8.9 GFLOPs, lo que sugiere una eficiencia computacional razonable.

En cuanto a la inferencia en una imagen específica, se detectó 1 persona en una imagen de tamaño 448x640 píxeles. El tiempo total de procesamiento de la imagen, desde el preprocesamiento hasta el postprocesamiento, fue de 106.4 milisegundos. Esto se desglosa en 6.6 milisegundos para el preprocesamiento, 106.4 milisegundos para la inferencia y 158.5 milisegundos para el postprocesamiento.

En resumen, el modelo YOLOv8n es capaz de detectar personas en imágenes con una eficiencia razonable, procesando una imagen en aproximadamente 106.4 milisegundos y realizando alrededor de 8.9 GFLOPs durante la inferencia.

Figura 48 Detección cámara seguridad yolo8n

7.1.2.1.3 Video

Media de inferencia: 20.238 ms

Máximo de 'postprocess': 62.160 ms

Máximo de 'preprocess': 17.124 ms

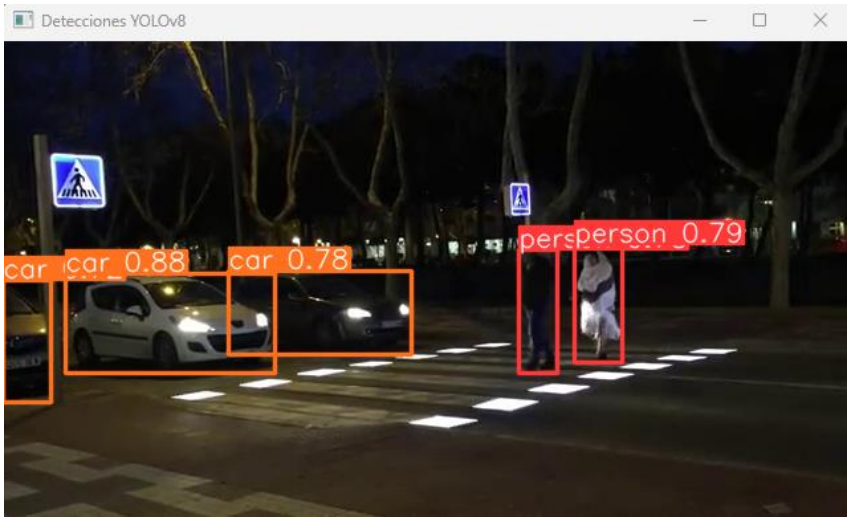

Figura 49 Detección en video Yolo8n

7.1.2.2 Modelo yolov8m

7.1.2.2.1 Imagen

- **Inferencia en una imagen específica:**
  - En la primera inferencia (0), se detectaron 4 personas y 1 autobús (bus) en una imagen de tamaño 448x448 píxeles.
  - En la segunda inferencia (1), se detectaron 8 personas, 1 banco (bench), 1 mochila (backpack), 1 bolso de mano (handbag) y 1 teléfono celular (cell phone) en otra imagen de tamaño 448x448 píxeles.
- **Velocidades detalladas:**
  - **Tiempo de preprocesamiento**: toma 1.5 milisegundos por imagen.
  - **Tiempo de inferencia:** toma 52.1 milisegundos por imagen.
  - **Tiempo de postprocesamiento**: toma 28.7 milisegundos por imagen.
  - En total, el proceso completo (desde la carga hasta la obtención de resultados) para cada imagen lleva 104.2 milisegundos.

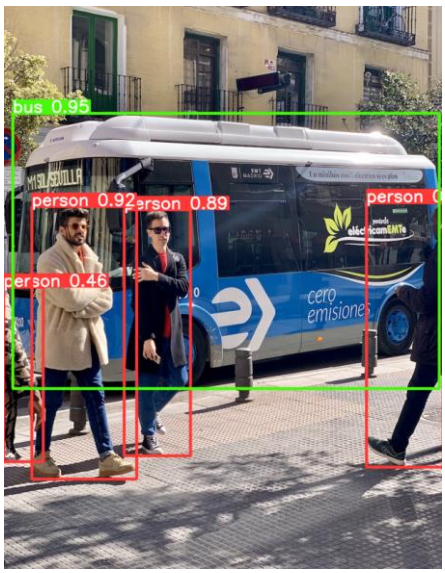

Figura 50 Detección bus Yolov8m

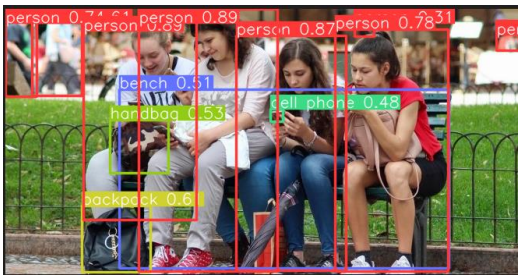

Figura 51 Detección personas Yolov8m

7.1.2.2.2 Imagen cámara seguridad

El modelo YOLOv8m tiene 295 capas y un total de 25,902,640 parámetros entrenables. Durante la inferencia en una imagen, el modelo realiza aproximadamente 79.3 GFLOPs, lo que sugiere una alta capacidad de procesamiento.

En una imagen específica de tamaño 448x640 píxeles, el modelo detectó 1 persona y 1 tren. El tiempo total de procesamiento de la imagen, desde el preprocesamiento hasta el postprocesamiento, fue de 133.6 milisegundos. Esto se desglosa en 9.1 milisegundos para el preprocesamiento, 133.6 milisegundos para la inferencia y 197.8 milisegundos para el postprocesamiento.

En resumen, el modelo YOLOv8m es capaz de detectar personas y otros objetos en imágenes con una eficiencia razonable, procesando una imagen en aproximadamente 133.6 milisegundos y realizando alrededor de 79.3 GFLOPs durante la inferencia.

Figura 52 Detección cámara seguridad yolo8m

7.1.2.2.3 Video

Media de inferencia: 23.237 ms

Máximo de 'postprocess': 58.873 ms

Máximo de 'preprocess': 12.412 ms

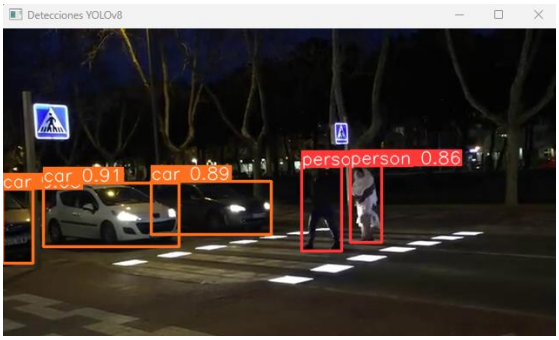

Figura 53 Detección en video Yolov8m

7.1.2.3 Modelo yolov8x

7.1.2.3.1 Imagen

- **Inferencia en una imagen específica:**
  - En la primera inferencia (0), se detectaron 4 personas y 1 autobús (bus) en una imagen de tamaño 448x448 píxeles.
  - En la segunda inferencia (1), se detectaron 13 personas, 1 banco (*bench*), 1 mochila (*backpack*), 1 sombrilla (*umbrella*), 3 bolsos de mano (*handbags*) y 3 teléfonos celulares (*cell phones*) en otra imagen de tamaño 448x448 píxeles.
- **Velocidades detalladas:**
  - **Tiempo de preprocesamiento**: toma 1.5 milisegundos por imagen.
  - **Tiempo de inferencia**: toma 52.0 milisegundos por imagen.
  - **Tiempo de postprocesamiento**: toma 30.4 milisegundos por imagen.
  - **En total**, el proceso completo (desde la carga hasta la obtención de resultados) para cada imagen lleva 104.0 milisegundos.

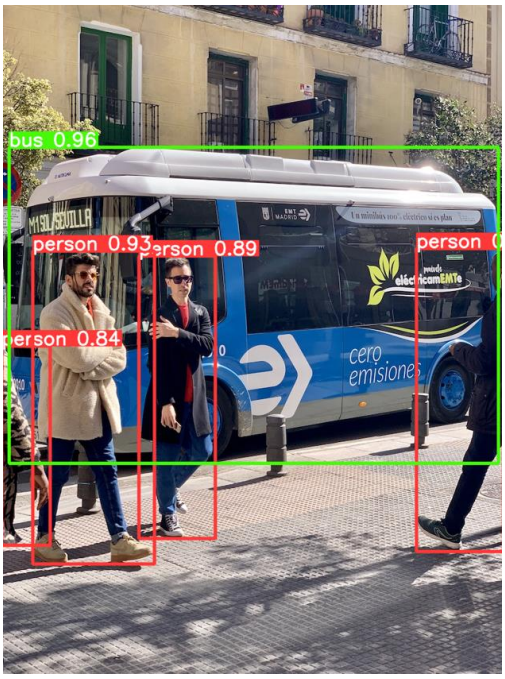

Figura 54 Detección bus Yolov8x

Figura 55Detección personas Yolov8x

7.1.2.3.2 Imagen cámara seguridad

El modelo YOLOv8x tiene 365 capas y un total de 68,229,648 parámetros entrenables. Durante la inferencia en una imagen, el modelo realiza aproximadamente 258.5 GFLOPs, lo que sugiere una alta capacidad de procesamiento.

En una imagen específica de tamaño 448x640 píxeles, el modelo detectó 1 persona y 1 tren. El tiempo total de procesamiento de la imagen, desde el preprocesamiento hasta el postprocesamiento, fue de 129.4 milisegundos. Esto se desglosa en 7.0 milisegundos para el preprocesamiento, 129.4 milisegundos para la inferencia y 141.4 milisegundos para el postprocesamiento.

En resumen, el modelo YOLOv8x es capaz de detectar personas y otros objetos en imágenes

con una eficiencia razonable, procesando una imagen en aproximadamente 129.4 milisegundos y realizando alrededor de 258.5 GFLOPs durante la inferencia.

Figura 56 Detección cámara seguridad yolo8x

7.1.2.3.3 Video

Media de inferencia: 33.489 ms

Máximo de 'postprocess': 62.355 ms

Máximo de 'preprocess': 21.971 ms

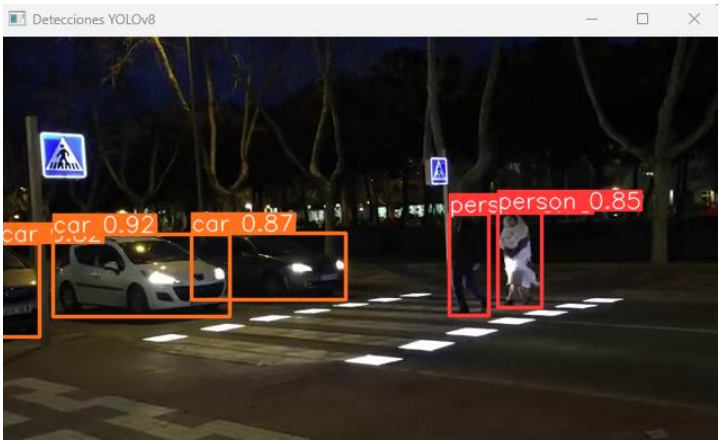

Figura 57 Detección en video Yolo8x

7.1.2.4 Modelo Open-Imagen-V7

```
yolov8x-oiv7.pt
```

7.1.2.4.1 Imagen

- **Inferencia en una imagen específica:**
  - En la primera inferencia (0), se detectó 1 autobús (bus), 5 calzados (*footwears*), 2 pantalones (*jeans*) y 3 hombres (*mans*) en una imagen de tamaño 448x448 píxeles.
  - En la segunda inferencia (1), se detectaron 7 calzados (*footwears*), 1 pantalón (*jeans*) y 4 mujeres (*womans*) en otra imagen de tamaño 448x448 píxeles.
- **Velocidades detalladas:**
  - **Tiempo de preprocesamiento**: toma 1.5 milisegundos por imagen.
  - **Tiempo de inferencia**: toma 50.0 milisegundos por imagen.
  - **Tiempo de postprocesamiento**: toma 32.1 milisegundos por imagen.
  - En total, el proceso completo (desde la carga hasta la obtención de resultados) para cada imagen lleva 100.0 milisegundos.

Figura 58 Detección bus Yolov8x-oi7

Figura 59 Detección personas Yolo8x-oi7

7.1.2.4.2 Imagen cámara seguridad

l modelo tiene 365 capas y un total de 68,731,371 parámetros entrenables. Durante la inferencia en una imagen, el modelo realiza aproximadamente 261.3 GFLOPs, lo que sugiere una alta capacidad de procesamiento.

Resultados de la inferencia: En una imagen específica de tamaño 448x640 píxeles, el modelo no detectó personas, pero detectó 1 prenda de vestir, 1 puerta y 1 hombre. El tiempo total de procesamiento de la imagen, desde el preprocesamiento hasta el postprocesamiento, fue de 92.5 milisegundos. Esto se desglosa en 2.5 milisegundos para el preprocesamiento, 92.5 milisegundos para la inferencia y 58.4 milisegundos para el postprocesamiento.

Figura 60 Detección cámara seguridad yolo8n librería Open-Imagen

7.1.2.4.3 Video

Media de inferencia: 17.129 ms

Máximo de 'postprocess': 55.718 ms

Máximo de 'preprocess': 3.561 ms

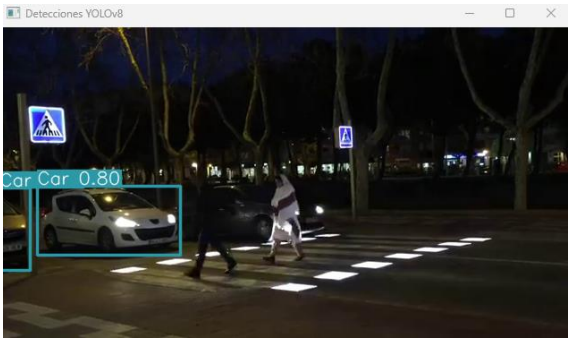

Figura 61 Detección en video Yolo8x-oi7

7.1.2.5 Resultados Yolov8

Con base en los resultados obtenidos en las imágenes y videos anteriores con los modelos YOLOv8 (n, m, x) y el modelo preentrenado de Open-Images-v7 se han obtenido los siguientes resultados:

- El modelo YOLOv8-x muestra una mejora en la precisión de aproximadamente 0.05 en comparación con el modelo YOLOv8-n. Sin embargo, este aumento en la precisión se traduce en un tiempo de inferencia significativamente mayor.
- Considerando un escenario de cámaras de seguridad con hardware limitado, donde el tiempo de procesado es crítico, la ganancia marginal en precisión no justifica el aumento en el tiempo de procesado. Se concluye que para entornos con recursos hardware restringidos, la elección de modelos con un buen equilibrio entre precisión y tiempo de inferencia es esencial.
- Los resultados obtenidos al utilizar el modelo preentrenado de Open-Images-v7 no han sido satisfactorios, especialmente en condiciones de baja iluminación donde la detección de personas fue deficiente. Esto sugiere que el modelo preentrenado puede no haber sido entrenado lo suficiente en escenarios de poca luz o puede necesitar ajustes adicionales para adaptarse a estas condiciones.

En el contexto de cámaras de seguridad, se ha procesado una imagen y un video, que se usará un servidor con un hardware limitado, la elección del modelo de detección de objetos es una decisión crítica que impacta directamente en el rendimiento del sistema. En este estudio, se evaluaron tres variantes de YOLOv8, designadas como YOLOv8-n, YOLOv8-m y YOLOv8-x,

con el objetivo de identificar la opción más adecuada para entornos con limitaciones computacionales.

Se observó que el modelo YOLOv8-x exhibe una mejora marginal en la precisión, aproximadamente 0.05 en comparación con YOLOv8-n. Sin embargo, este aumento en la precisión se traduce en un aumento sustancial en el tiempo de inferencia, lo que podría no ser justificado en aplicaciones de tiempo real con recursos hardware limitados.

7.1.3 YOLONAS

En este apartado analiza el modelo YoloNas que ofrece un rendimiento de última generación (SOTA) con un rendimiento de precisión y velocidad sin precedentes, superando a otros modelos como YOLOv5, YOLOv6, YOLOv7 y YOLOv8. Para la aplicación de este modelo se ha utilizado la biblioteca de super-gradients, ya que la biblioteca ultralitycs tiene un bug que no permite la utilización de la esta biblioteca, este bug está recogido en github (https://github.com/Deci-AI/super-gradients/issues/1021).
Los modelos preentrenados de YoloNas son los que se muestran en la figura 62:

Modelo	mapa	Latencia (ms)
YOLO-NAS S	47.5	3.21
YOLO-NAS M	51.55	5.85
YOLO-NAS L	52.22	7.87
YOLO-NAS S INT-8	47.03	2.36
YOLO-NAS M INT-8	51.0	3.78
YOLO-NAS L INT-8	52.1	4.78

Figura 62 Modelos YoloNas

La Figura 62 muestra los modelos que propone Ultralytics para obtener el mejor renidmiento en velocidad y precisión para el modelo YOLONAS. Dentro de estos modelos se muestran los siguientes:

- YOLO-NAS-s: optimizado para entornos en los que los recursos informáticos son limitados pero la eficiencia es clave.
- YOLO-NAS-m: ofrece un enfoque equilibrado, adecuado para la detección de objetos de uso general con mayor precisión.

- YOLO-NAS-l: adaptado para escenarios que requieren la máxima precisión, en los que los recursos informáticos son menos limitantes.

Para la obtención de los resultados se ha utilizado el código de los Anexos III y IV.

7.1.3.1 Modelo yoloNas-S

Figura 63 Detección cámara seguridad yoloNas S

7.1.3.1.1 Imagen

Tiempo de procesamiento para la imagen 1: 739.0532 ms

Tiempo de procesamiento para la imagen 2: 114.6309 ms

Figura 64 Detección bus YolovNas-S

Figura 65 Detección personas YoloNas-S

7.1.3.1.2 Imagen cámara seguridad

Tiempo de procesamiento para la imagen 3: 117.9333 ms

Figura 66 Detección cámara seguridad yoloNas M

7.1.3.1.3 Video

Tiempo medio procesamiento: 259.9647 ms

Tiempo de procesamiento para el video: 266983.7377 ms

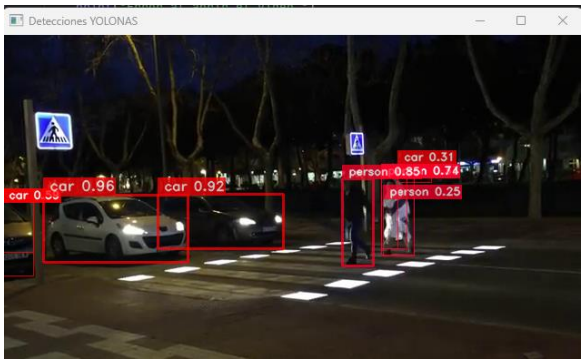

Figura 67 Detección en video YoloNas-S

7.1.3.2 Modelo yoloNas-M

7.1.3.2.1 Imagen

Tiempo de procesamiento para la imagen 1: 683.0626 ms

Tiempo de procesamiento para la imagen 2: 727.5069 ms

Figura 68 Detección bus YoloNas-M

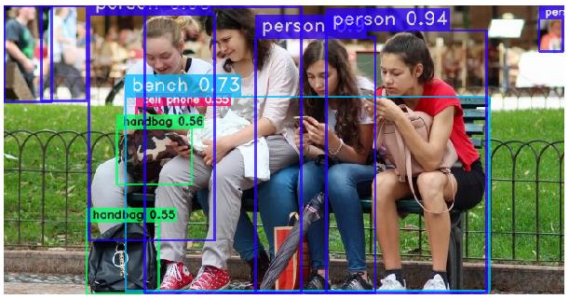

Figura 69 Detección personas YoloNas-M

7.1.3.2.2 Imagen cámara seguridad

Tiempo de procesamiento para la imagen 3: 257.7202 ms.

Figura 70 Detección cámara seguridad yoloNas M

7.1.3.2.3 Video

Tiempo medio procesamiento: 330.3489 ms

Tiempo de procesamiento para el video: 339268.3318 ms

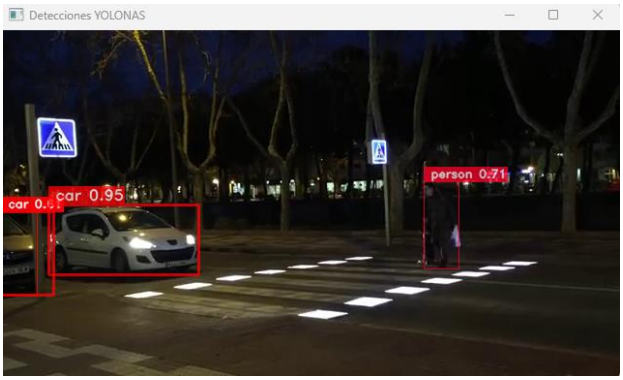

Figura 71 Detección en video YoloNas-M

7.1.3.3 Modelo yoloNas-L

7.1.3.3.1 Imagen

Tiempo de procesamiento para la imagen 1: 626.6778 ms

Tiempo de procesamiento para la imagen 2: 142.0729 ms

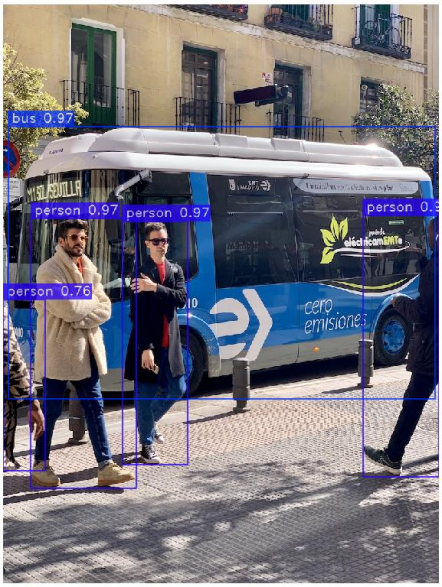

Figura 72 Detección bus YoloNas-L

7.1.3.3.2 Imagen cámara seguridad

Tiempo de procesamiento para la imagen 3: 158.2425 ms

Figura 73 Detección cámara seguridad yoloNas L

7.1.3.3.3 Video

Tiempo medio procesamiento: 227.3179 ms

Tiempo de procesamiento para el video: 233455.4482 ms

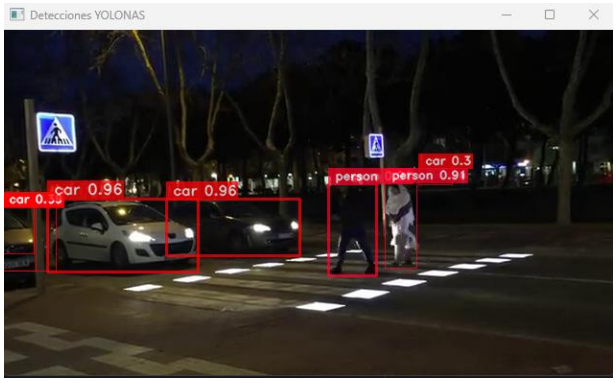

Figura 74 Detección en video YoloNas-L

7.1.3.4 Resultados YoloNas

De los resultados anteriores se obtiene que el modelo YOLOvNAS-S tiene una precisión menor pero un tiempo de procesamiento más rápido en comparación con los modelos YOLOvNAS-M y YOLOvNAS-L. Esto puede deberse a que el modelo más pequeño (S) tiene menos parámetros y capas, lo que le permite realizar inferencias más rápidas a costa de una precisión ligeramente menor. Por otro lado, los modelos M y L, al ser más complejos y tener más parámetros, pueden lograr una precisión similar, pero a costa de un mayor tiempo de procesamiento.

También se ha detectado que la velocidad del modelo L es más rápida que el modle m, cuando no debería tener ese comportamiento, esto se puede deber a:

- La implementación del modelo L puede haber sido realizada de manera más eficiente, lo que permite un procesamiento más rápido de las imágenes.
- La variabilidad en los conjuntos de datos utilizados para evaluar los modelos podría haber afectado los resultados. Es posible que el modelo L haya tenido una mejor adaptación a los datos de prueba específicos, lo que resulta en un rendimiento aparentemente mejor.

Dado los resultados obtenidos en el análisis de los modelos S, M y L con la arquitectura YOLONAS, habría que comprobar con el hardware real de videovigilancia que modelo M o L sería el más adecuado. Si solo tuviéramos en cuenta esta primara parte de análisis y elección del modelo el modelo NAS-L nos daría la mayor precisión con un tiempo de procesado similar a los modelos S o M.

7.1.4 Detección con cámara

Una vez que realizada la comprobación con video, se ha realizado la comprobación en Stream de diferentes grabaciones con la webcam del ordenador, obteniendo resultados satisfactorios,

como los mostrados en la Figura 75, en la detección de personas con todos los modelos de YOLO preentrenados con COCO.

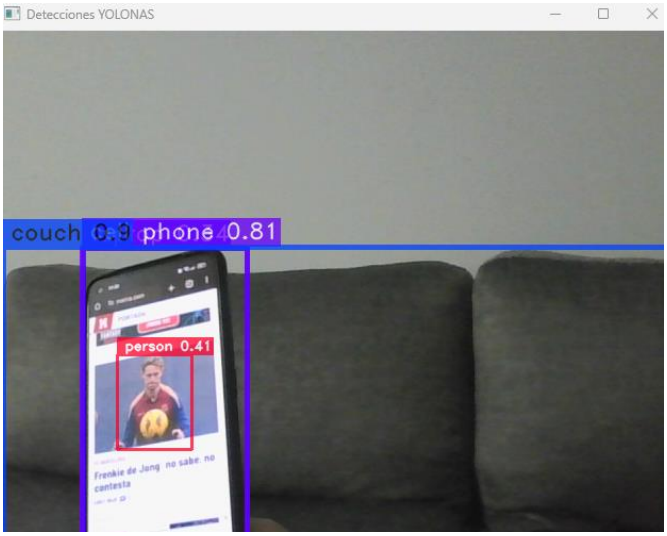

Figura 75 Detección de personas webcam (Fuente propia)

En la Figura 75 se observa que, aun grabando la imagen de una persona desde un *smartphone*, es posible la detección. De los diferentes modelos evaluados anteriormente se recomendaría la utilización de la arquitectura de YOLONAS al tener mejor confianza que el resto de los modelos. Dentro de esta arquitectura habría que analizar el hardware empleado para la elección de un modelo u otro.

Además, se ha hecho otra prueba con la imagen de una cámara de seguridad a tiempo real como es la mostrada en la figura 76.

Figura 76 Reconocimiento cámara de seguridad (Elaboración propia)

En la Figura 76 se muestra la detección de personas con el modelo YOLONAS, con el que habría que ajustar el parámetro de precisión a una precisión mayor del 0.65 para evitar falsos resultados.

7.2 Cableado y Programación Relé Programable

El cableado y programación que se ha realizado en Siemens Logo es muy sencillita, ya que en este trabajo únicamente se va a realizar el prototipado del sistema de seguridad en el que únicamente se van a conectar un piloto de iluminación y un zumbador. Lo que supone que solo se cablearan dos salidas del LOGO y ninguna entrada.

7.2.1 Cableado

En la Figura 77 se muestra el esquema de la conexión de un LOGO 8 con dos actuadores, una bombilla y un timbre.

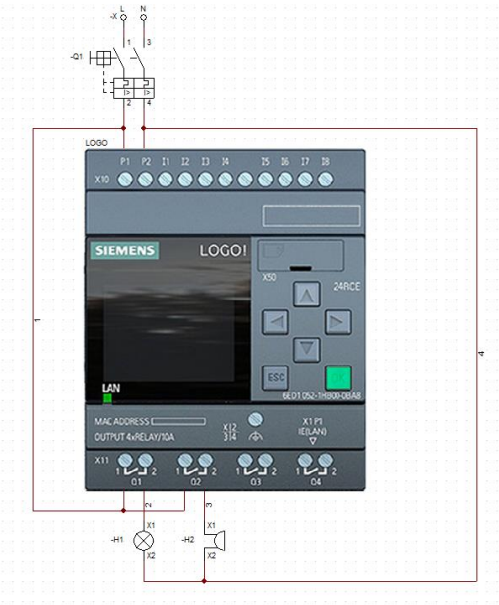

Figura 77 Esquema Logo 8 (Fuente propia)

7.2.2 Programación

Para la realización de la programación en LOGO Soft Confort se ha utilizado el lenguaje diagramas de funciones (FUP), se ha añadido además de la entrada de red una entrada I1 para poder simular el programa cuando no está el LOGO físicamente conectado. La Figura 78 representa la programación en lenguaje FUP de la lógica del LOGO.

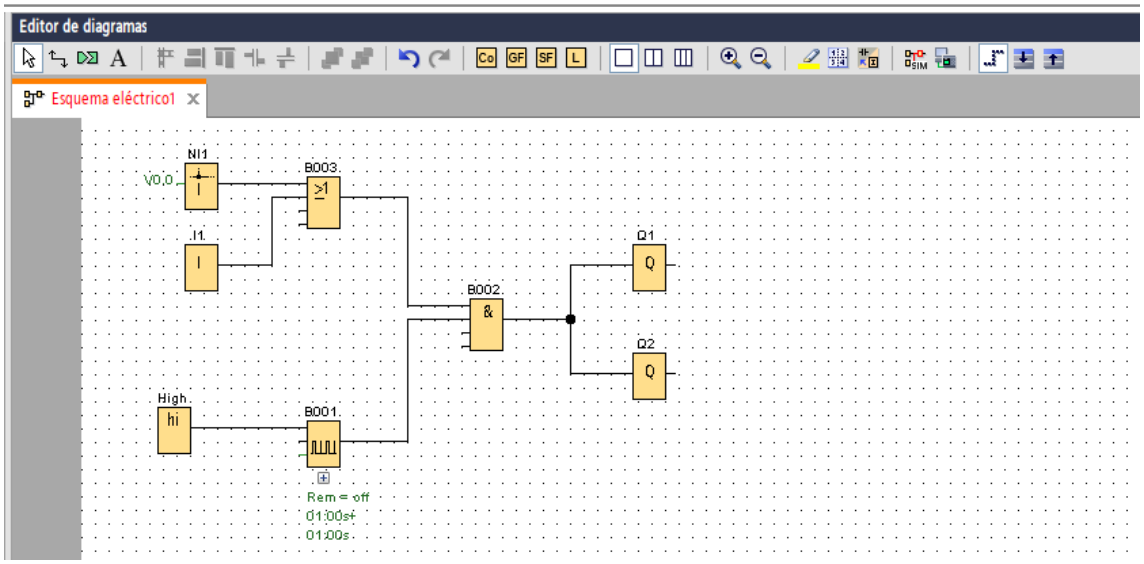

Figura 78 Código LogoSoft Comfort (Fuente propia)

La figura 78 es un ejemplo sencillo de programación donde al recibir una entrada digital virtual, procedente de la detección de objetos con YOLO activa las dos salidas conectadas, la bombilla y el timbre.

7.3 Comunicación Node-Red

Para la realización de la comunicación con Node-Red que va a ser la pasarela que va a comunicar la detección de objetos con el automatismo se ha realizado el siguiente diagrama con la utilización de dos librerías:

- La librería **node-red-contrib-s7 3.1.0 (**node-red-contrib-s7 (node) - Node-RED (nodered.org)**)** para conectar con el Siemens LOGO.
- La librería websockets **node-red-contrib-websocket** (node-red-contrib-websocket (node) - Node-RED (nodered.org)) para conectar por websockets con la detección de objetos en Python.

En el marco de este proyecto, se optará por establecer una comunicación eficiente entre la detección de objetos implementada en Python y una aplicación web mediante el uso de websockets. La propuesta es desarrollar una aplicación web utilizando un framework de Python, como FastAPI, aprovechando sus capacidades para crear interfaces web robustas y de alto rendimiento.

La elección de utilizar websockets se fundamenta en la necesidad de una comunicación bidireccional y en tiempo real entre la aplicación web y el componente de detección de objetos. Esta decisión permitirá la transmisión instantánea de datos y resultados entre ambas entidades, facilitando una experiencia de usuario dinámica y receptiva.

Para optimizar la interoperabilidad con otras herramientas y sistemas, se planea integrar la comunicación mediante websockets con Node-RED. El código para realizar la comunicación con Node-Red desde la aplicación Python es el código que se encuentra en el Anexo V. Node-RED, conocido por su enfoque visual y flexible en el desarrollo de flujos de trabajo, actuará como un nodo de recepción y procesamiento para los datos provenientes de la detección de objetos. Con esta comunicación por websockets y la librería s7 de Siemens obtenemos el siguiente flujo de comunicación. En la figura 79 se establece el flujo de programación necesario a realizar en NodeRED.

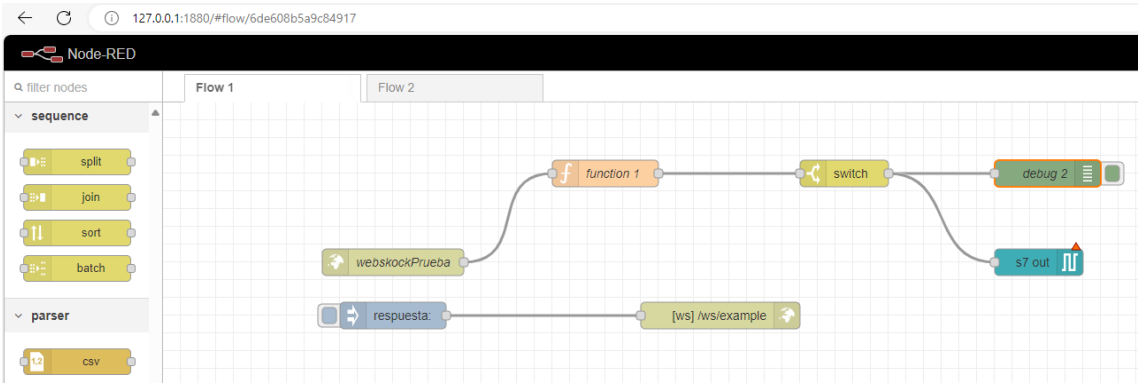

Figura 79 Flujo Node-Red (Fuente Propia)

La Figura 79 es la programación del flujo de nodeRed, donde al recibir una comunicación por webSockets del programa de detección de objetos este lo comunica con la parte física a través del envío de la información al LOGO8.

8 Conclusiones y trabajos futuros

8.1 Conclusiones

En este trabajo se ha realizado un análisis de modelos buscando la mejora del sistema de alarmas de un Instituto mediante la implementación de un sistema de detección de objetos, enfocándose principalmente en la identificación de personas. La comparación entre modelos preentrenados de las arquitecturas YOLOv5, YOLOv8 y YOLO-NAS ha arrojado resultados significativos, destacando la superioridad en términos de confianza de YOLO-NAS sobre las otras dos variantes.

Se ha profundizado en el análisis de las arquitecturas YOLOv5, YOLOv8 y YOLO-NAS, evaluando modelos específicos y considerando factores clave como la precisión, el tiempo de procesamiento de imágenes y videos y la adaptabilidad a condiciones desafiantes, como escenarios con poca iluminación. Todos los modelos fueron ya preentrenados con el conjunto de datos COCO2022, lo que les proporciona una base sólida para la detección de objetos en general. Se ha excluido un modelo preentrenado con Open Images V7 para la versión YOLOv8 debido a su falta de detección efectiva en condiciones de poca iluminación. Esta observación destaca la importancia de la calidad y diversidad del conjunto de entrenamiento en la adaptabilidad del modelo a diversas condiciones ambientales.

La elección de optar por el modelo con menor coste computacional en cada arquitectura constituye una estrategia prudente para garantizar una integración económica y eficiente del sistema de seguridad que se quiere implementar. Esta decisión ponderada no solo optimizará los recursos disponibles, sino que también asegurará la viabilidad y sostenibilidad del sistema a largo plazo.

Dado que no contamos con el hardware del sistema de videovigilancia del instituto, el enfoque de este trabajo consiste en realizar un análisis exhaustivo de algunos de los modelos más importante del detector YOLO y determinar cuál sería la más eficiente para su implementación futura. Con el objetivo de obtener resultados lo más cercanos a la realidad, se han empleado imágenes y vídeos capturados por cámaras de seguridad, garantizando así una simulación precisa y representativa de las condiciones reales de aplicación. Se trata por tanto, de una prueba de concepto de cara a su implantación en un sistema real.

La mejor elección al concluir este trabajo de fin de máster es utilizar YOLO-NAS en su versión más pequeña (YOLO-NAS-s) para una futura aplicación en un entorno real se justifica por los resultados destacados que ha demostrado en términos de confianza en la detección de personas superando a las otras versiones de YOLO.

Una vez realizada la detección y elección del modelo de detección de objetos óptimo para la aplicación buscada, se ha profundizado en la mejora de la disuasión del sistema de seguridad y se ha logrado mediante la comunicación efectiva de la detección de objetos con un relé programable Siemens LOGO 8.2, activando señales visuales y auditivas para disuadir a personas no autorizadas. Esta integración no solo refuerza la eficacia del sistema, sino que también demuestra la aplicabilidad de las soluciones tecnológicas para la mejora de la seguridad en entornos educativos.

8.2 Trabajo Futuro

El horizonte de desarrollo futuro de este proyecto se vislumbra con diversas posibilidades y mejoras potenciales:

- **Desarrollo de Aplicación Web:**

La implementación de una aplicación web en Python, preferiblemente con frameworks como FastAPI, podría mejorar significativamente la accesibilidad y la gestión remota del sistema de seguridad.

- **Comunicaciones Adicionales:**

Ampliar las capacidades del sistema mediante la inclusión de comunicaciones adicionales, como correos electrónicos o mensajes SMS, permitiría una notificación inmediata de la detección de intrusos, mejorando la capacidad de respuesta del sistema.

- **Optimización de Arquitecturas de Detección:**

Explorar y evaluar otras arquitecturas de detección de objetos podría revelar alternativas que se adapten de manera más efectiva a las características específicas de la aplicación en el entorno educativo.

- **Estudio de Eficiencia Energética:**

Realizar un análisis de eficiencia energética del sistema implementado podría proporcionar información valiosa sobre el consumo de recursos y sugerir posibles ajustes para mejorar la sostenibilidad.

- **Consideración de Escenarios Específicos:**

Evaluar el sistema en escenarios específicos, como situaciones de poca iluminación o eventos

atípicos, podría proporcionar *insights* adicionales sobre la robustez y la eficacia del sistema en condiciones diversas.

Estos trabajos futuros no solo ampliarían las capacidades del sistema implementado, sino que también contribuirían al avance continuo de la investigación en la aplicación de tecnologías de detección de objetos para mejorar la seguridad en diferentes entornos, evaluándose en este para un entorno educativos. La intersección de la tecnología y la seguridad presenta un campo en el que profundizar para la innovación y la optimización continua de los sistemas de protección y disuasión.

9 Bibliografía

Aggarwal, C. C. (2023). *Neural Networks and Deep Learning: A Textbook.* Springer; 2nd ed.

Bagnato, J. I. (2020). *Aprende Machine Learning*. Obtenido de https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/

Bai, S., Kolter, J., & Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. Obtenido de https://doi.org/10.48550/arXiv.1803.01271

Bochkovskiy, A., Wang, C. Y., & Mark Liao, H.-Y. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection.* Obtenido de https://doi.org/10.48550/arXiv.2004.10934

Chen, H., Chen, Z., & Yu, H. (2023). *Enhanced YOLOv5: An Efficient Road Object Detection Method.* Sensors 23, no. 20: 8355. https://doi.org/10.3390/s23208355.

Chen, L., Zhang, H., Xiao, J., & Chang, S.-F. (2017). Sca-CNN: Spatial and Channel-wise Attention in Convolutional Networks for Image Captioning. En *Proceedings of the IEEE Conference on Computer Vision and Pattern* (págs. 5659-5667).

COCO Datasets. (s.f.). Obtenido de https://cocodataset.org/#home

Courville, I. G. (2016). *Deep Learning.* Goodfellow: MIT Press.

Deci AI. (s.f.). *The Deep Learning Development Platform*. Obtenido de https://deci.ai/

DeciAI. (s.f.). *DECI AI docs.* Obtenido de https://docs.deci.ai/super-gradients/latest/documentation/source/ModelPredictions.html#detect-objects-using-a-webcam

Departamento de Matemática Aplicada, UPM. (2021). *Convolutional Neural Network : definición y funcionamiento*. Recuperado el 28 de 12 de 2023, de https://dcain.etsin.upm.es/~carlos/bookAA/05.7_RRNN_Convoluciones_CIFAR_10_INFORMATIVO.html#capa-convolucional

Gao Huang, Z. L. (2017). Densely Connected Convolutional Networks. Obtenido de https://doi.org/10.48550/arXiv.1608.06993

Github/Ultralitycs/Yolo. (s.f.). *Github Yolov5*. Obtenido de https://github.com/ultralytics/yolov5?tab=readme-ov-file

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern* (págs. 770-778).

Howard, A. G., Chen, B., & Kalenichenko, D. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* Obtenido de https://doi.org/10.48550/arXiv.1704.04861

IBM. (s.f.). Recuperado el 10 de 12 de 2023, de https://www.ibm.com/es-es/topics/convolutional-neural-networks

ICHI.PRO. (s.f.). Obtenido de ICHI.PRO: https://ichi.pro/es/la-breve-historia-de-las-redes-neuronales-convolucionales-76620623015807#:~:text=En%20esta%20publicaci%C3%B3n%2C%20visitaremos%20brevemente%20los%20or%C3%ADgenes%20de,de%20visi%C3%B3n%20por%20computadora%20pre-entrenados%20de%20la%20ac

Infaimon. (s.f.). *Infaimon*. Obtenido de https://infaimon.com/blog/vision-2d-3d/historia-evolucion-vision-artificial/

Infokrause. (16 de Mayo de 2018). *Cámaras de vigilancia: la importancia de Full HD*. Obtenido de https://www.infokrause.cl/en/news/camaras-vigilancia-importancia-full-hd

InfoPLC. (12 de 06 de 2015). *Introducción a las Redes de Comunicación Industrial*. Recuperado el 28 de 01 de 2024, de https://www.infoplc.net/documentacion/7-comunicaciones-industriales/2332-redes-comunicacion-industrial

InfoPLC. (12 de Julio de 2015). *Introducción a las Redes de Comunicación Industrial*. Obtenido de https://www.infoplc.net/documentacion/7-comunicaciones-industriales/2332-redes-comunicacion-industrial

Jetbrains. (s.f.). *PyCharm*. Obtenido de https://www.jetbrains.com/pycharm/

Joseph Redmon, A. F. (2016). *YOLO9000:*. University of Washington. Obtenido de https://doi.org/10.48550/arXiv.1612.08242

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems.* - Communications of the ACM, Vol. 60 (6), doi: 84–90https://doi.org/10.1145/3065386

Kunihiko, F. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36, 193–202.

Kuznestova, A., Rom, H., Alldrin, N., Ujling, J., Pont-Tuset, J., Kamali, S., . . . Ferrari, V. (2021). *The Open Images Dataset v7*. Google LLC. Obtenido de https://doi.org/10.1007/s11263-020-01316-z

Le, M. T. (s.f.). EfficientDet: Scalable and Efficient Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 10781-10790

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. En *Neural Computationl computation*, vol. 1 (4), 541-555.

LeCunn, Y., Bengio, Y., & Hinton, G. (2015). *Deep Learning.* Nature volume 521, pages436–444.

LeCunn, Y., Bottou, L., Bengio, Y., & Haffnet, P. (1998). *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, Volumen 86.

Lee, S.-M., Yoon, S. M., & Cho, H. (2017). *Human Activity Recognition From Accelerometer Data Using Convolutional Neural Network.* Jeju (South Korea): IEEE.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017). Focal Loss for Dense Object Detection. En *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (págs. 2980-2988).

Lin, T.-Y., Maire, M., Belongie, S., Boordev, L., Hays, J., Perona, P., & Dollar, P. (2014). *Microsoft COCO: Common Objects in Context.* Springer International Publishing.

Marr, D. (1982). *Vision W.H.* San Francisco: Freeman & Co.

Mathworks. (s.f.). Recuperado el 10 de 12 de 2023, de Mathworks: https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html

Microsoft. (2023). *Tutorial: Detect objects using ONNX in ML.NET*. Obtenido de

https://learn.microsoft.com/en-us/dotnet/machine-learning/tutorials/object-detection-onnx

MSI. (s.f.). *MSI PULSE 17*. Obtenido de https://es.msi.com/Laptop/Pulse-17-B13VX

NodeRed. (s.f.). Obtenido de https://nodered.org/

Open Images V7. (s.f.). Open Images Dataset V7 and Extension. Obtenido de https://storage.googleapis.com/openimages/web/index.html

PacoCara. (2023). *Dolor cervical en niños y adolescentes*. Obtenido de https://pacocara.com/dolor-cervical-en-ninos-y-adolescentes/

Pajares, G., Herrera, P. J., & Besada, E. (2021). *Aprendizaje Profundo.* Madrid: RC-Libros.

Pytorch. (s.f.). *Pytorch*. Obtenido de https://pytorch.org/

Radford, A., Metz, L., & Chintala, S. (2015). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* Obtenido de https://doi.org/10.48550/arXiv.1511.06434

RangeKing. (2022). *Github*. Obtenido de https://github.com/RangeKing?tab=repositories&q=yolo&type=&language=&sort=

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement.* Obtenido de ; https://doi.org/10.48550/arXiv.1804.02767

Siemens . (s.f.). *Logo! Basic Modules*. Obtenido de https://www.siemens.com/global/en/products/automation/systems/industrial/plc/logo/logo-basic-modules.html#BasicModuleswithdisplay

Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition.* Obtenido de: https://doi.org/10.48550/arXiv.1409.1556

Solawetz, J. (29 de 06 de 2020). *Roboflow*. Recuperado el 10 de 01 de 2024, de https://blog.roboflow.com/yolov5-improvements-and-evaluation/

Solawetz, J. (23 de 06 de 2020). *What are Anchor Boxes in Object Detection?* Obtenido de https://blog.roboflow.com/what-is-an-anchor-box/

Springenberg∗, J. T. (2014). STRIVING FOR SIMPLICITY THE ALL CONVOLUTIONAL NET. Department of Computer Science, University of Freiburg.

Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and Efficient Object Detection. En *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (págs. 10781-10790).

Tutorial: Detección de objetos con ONNX en ML.NET. (5 de 6 de 2023). Recuperado el 7 de 01 de 2024, de https://learn.microsoft.com/es-es/dotnet/machine-learning/tutorials/object-detection-onnx

Ultralytics. (s.f.). *Docs ultralytics.* Obtenido de https://docs.ultralytics.com/

Wang, C.-Y., Mark Liao, H.-Y., Yeh, H., Wu, Y.-H., Chen, P.-Y., & Hsieh, J.-W. (2019). CSPNET: A NEW BACKBONE THAT CAN ENHANCE LEARNING.

10 Anexo I -Código

10.1 Yolov5 – Detección de imágenes

En este apéndice se presenta el código utilizado para el procesamiento de todas las imágenes con las arquitecturas YOLO de Ultralytics, adaptando el modelo utilizado en función de la arquitectura bajo análisis.

```
from PIL import Image
from ultralytics import YOLO

# Load a COCO-pretrained YOLOv5n model
model = YOLO('yolov5nu.pt')

# Display model information (optional)
model.info()

# Inicializar el contador de personas
num_personas = 0

# Run inference on 'bus.jpg' with arguments
results = model.predict('images/bus.jpg', save=False, imgsz=448,
conf=0.65)


# Show the results
for r in results:
    im_array = r.plot()  # plot a BGR numpy array of predictions
    im = Image.fromarray(im_array[..., ::-1])  # RGB PIL image
    im.show()  # show image

    # Obtener el tensor de detecciones
    tensor = r.boxes.cls

    # Iterar sobre los elementos del tensor
    for element in tensor:
        if element == 0:
            num_personas += 1

# Imprimir el número de personas
print(f"Número de personas detectadas: {num_personas}")
```

10.2 Yolov5 - Detección de video

En este apéndice se presenta el código utilizado para el procesamiento de todos los videos con las arquitecturas YOLO de Ultralytics, adaptando el modelo utilizado en función de la arquitectura bajo análisis.

```
from ultralytics import YOLO
import cv2

# Cargar el modelo YOLOv5 preentrenado en COCO
model = YOLO('yolov5nu.pt')

# Abrir el video con OpenCV
cap = cv2.VideoCapture("PasoPeatones.mp4")


if not cap.isOpened():
    print("Error al abrir el video.")
else:
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Inicializar el contador de personas
        num_personas = 0

        results = model(frame)
        model.cuda()

        # Filtrar detecciones con confianza mayor al 50%
        filtered_results = results[0][results[0].boxes.conf > 0.5]

        # Crear una copia del frame para dibujar las detecciones
        frame_con_detecciones = frame.copy()

        # Dibujar las detecciones filtradas en el frame copiado
        frame_con_detecciones = filtered_results.plot()

        # Obtener el tensor de detecciones
        tensor = results[0].boxes.cls

        # Iterar sobre los elementos del tensor
        for element in tensor:
            if element == 0:
                num_personas += 1

        # Mostrar el frame con detecciones
        cv2.imshow('Detecciones YOLOv5', frame_con_detecciones)
        print(f"Número de personas detectadas: {num_personas}")

        # Esperar 30 milisegundos entre frames (ajustar según el FPS
del video)
```

```
        if cv2.waitKey(30) & 0xFF == 27:  # Presionar Esc para salir
            break

    cap.release()
    cv2.destroyAllWindows()
```

10.3 YoloNas - Detección de imágenes

En este apéndice se presenta el código utilizado para el procesamiento de todas las imágenes con las arquitecturas YOLONAS de DeciAI, adaptando el modelo utilizado en función de la arquitectura bajo análisis.

```
import torch.cuda
from super_gradients.training import models
from super_gradients.common.object_names import Models
import time
import sys

model = models.get(Models.YOLO_NAS_L, pretrained_weights="coco")
model = model.to("cuda" if torch.cuda.is_available() else "cpu")

# Evita el error de conflicto de librerías para mostrar en consola
sys.stdout = sys.__stdout__

# Create CUDA events for timing
start_event = torch.cuda.Event(enable_timing=True)
end_event = torch.cuda.Event(enable_timing=True)

# Medir el tiempo de procesamiento e inferencia
for i, image_path in enumerate(['images/bus.jpg', 'ima-
ges/imagenMovil.jpg']):
    # Registrar el tiempo de inicio
    start_time = time.time()

    # Realizar la inferencia en la imagen
    images_predictions = model.predict([image_path], iou=0.5,
conf=0.5)

    # Registrar el tiempo de finalización
    end_time = time.time()

    # Calcular el tiempo de procesamiento
    processing_time = end_time - start_time

    # Imprimir el tiempo de procesamiento e inferencia
    print(f"Tiempo de procesamiento para la imagen {i + 1}: {proces-
sing_time * 1000 :.4f} ms")

    # Abrir la imagen
    images_predictions.show(box_thickness=2, show_confidence=True)
```

10.4 YoloNas – Detección de video

En este apéndice se presenta el código utilizado para el procesamiento de todos los videos con las arquitecturas YOLONAS de DeciAI, adaptando el modelo utilizado en función de la arquitectura bajo análisis.

```
import torch.cuda
from super_gradients.training import models
from super_gradients.common.object_names import Models
import cv2
import sys
import time

model = models.get(Models.YOLO_NAS_S, pretrained_weights="coco")
model = model.to("cuda" if torch.cuda.is_available() else "cpu")

# Evita el error de conflicto de librerías para mostrar en consola
sys.stdout = sys.__stdout__

# Abrir el video con OpenCV
cap = cv2.VideoCapture(0)

num_inference = 0
total_time = 0
avg_time = 0

if not cap.isOpened():
    print("Error al abrir el video.")
else:
    while True:
        ret, frame = cap.read()

        if not ret:
            break

        # Inicializar el contador de personas
        num_personas = 0

        start_time = time.time()
        # Predecir las detecciones en el fotograma
        predictions = model.predict(frame)

        # Registrar el tiempo de finalización
        end_time = time.time()

        # Mostrar el frame con detecciones
        cv2.imshow('Detecciones YOLONAS', predictions.draw())

        # Calcular el tiempo de procesamiento
        processing_time = end_time - start_time

        total_time = total_time + processing_time
        num_inference += 1
```

```
        # Iterar sobre los elementos del tensor
        for element in predictions.prediction.labels:
            if element == 0:
                num_personas += 1
        print(f"Número de personas detectadas: {num_personas}")

        # Esperar 30 milisegundos entre frames (ajustar según el FPS
del video)
        if cv2.waitKey(30) & 0xFF == 27:  # Presionar Esc para salir
            break

# Imprimir el tiempo de procesamiento e inferencia
avg_time = total_time/num_inference
print(f"Tiempo medio procesiamiento: {avg_time * 1000 :.4f} ms")
print(f"Tiempo de procesamiento para el video: {total_time * 1000
:.4f} ms")

cap.release()
cv2.destroyAllWindows()
```

10.5 Comunicación WebSockets

En este apéndice se presenta el código utilizado para la comunicación mediante WebSockets de la aplicación en Python con Node-Red.

```python
import asyncio
import websockets

async def main():
    # Crea una instancia del objeto WebSocketClient
    ws = await websockets.connect("ws://localhost:1880/ws/example")

    # Envía un mensaje al servidor
    await ws.send("Detectando personas")

    # Recibe un mensaje del servidor
    msg = await ws.recv()

    # Imprime el mensaje recibido
    print(msg)

if __name__ == "__main__":
    # Ejecuta el código
    asyncio.run(main())
```