Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Máster Universitario en
Ingeniería Industrial

Trabajo de Fin de Máster

Diseño e implementación de un sistema deep learning para la detección de obstáculos y lectura de semáforos en tranvías usando Jetson Nano

Autor: Jorge Luis Díaz Acosta

Tutor: Cándido Caballero Gil

Cotutora: Sonia Díaz Santos

La Laguna, 6 de julio de 2024

La publicación de este Trabajo Fin de Máster solo implica que el estudiante ha obtenido al menos la nota mínima exigida para superar la asignatura correspondiente, no presupone que su contenido sea correcto, aunque si aplicable. En este sentido, la ULL no posee ningún tipo de responsabilidad hacia terceros por la aplicación total o parcial de los resultados obtenidos en este trabajo. También pone en conocimiento del lector que, según la ley de protección intelectual, los resultados son propiedad intelectual del alumno, siempre y cuando se haya procedido a los registros de propiedad intelectual o solicitud de patentes correspondientes con fecha anterior a su publicación.

D. **Cándido Caballero Gil**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

Dña. **Sonia Díaz Santos**, alumna del Doctorado de Ingeniería Industrial, Informática y Medioambiental adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora.

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Diseño e implementación de sistema deep learning de detección de obstáculos y lectura de semáforos para tranvías en Jetson Nano*"

ha sido realizada bajo su dirección por D. **Jorge Luis Díaz Acosta**, con N.I.F. 79063528Q.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2024

Agradecimientos

Quisiera agradecer en primer lugar a mi tutor D. Candido Caballero Gil y a mi cotutora Dña. Sónia Díaz Santos por haber aceptado tutorizar mi trabajo de fin de máster y haberme apoyado en el desarrollo de este trabajo, especialmente en las épocas más exigentes.

También quisiera agradecer a la compañía Metropolitano de Tenerife, S.A. por permitirme hacer este proyecto con ellos y suministrarme el material videográfico necesario para la realización de este proyecto.

Por último, quisiera agradecer a mi familia y pareja su apoyo durante estos meses de desarrollo del proyecto y animarme a seguir adelante con él.

Licencia

Resumen

En este proyecto de final de máster, se presenta el sistema de detección de obstáculos y lectura de semáforos mediante Inteligencia Artificial deep learning que ha sido diseñado para los tranvías de la red de tranvías de Tenerife, Islas Canarias, España.

Para su implementación, se ha realizado un estudio del estado del arte de los sistemas de detección de objetos mediante Inteligencia Artificial actuales, tras lo cual se ha seleccionado y entrenado un modelo YOLO de detección de objetos mediante transfer learning, para posteriormente implementarlo en un programa que ejecutará el modelo en un dispositivo Jeson Nano y generará alertas para el conductor del tranvía en función de lo que hayas sido detectado frente al tranvía.

Los sistemas basados en Inteligencia Artificial se encuentran en boga hoy en día, en gran medida gracias a los últimos avances en los modelos y la democratización de acceso de los usuarios a herramientas para el desarrollo de sistemas de Inteligencia Artificial.

Con este proyecto, se ha demostrado que es viable la implementación de un sistema de Inteligencia Artificial que detecte obstáculos y lea los semáforos de la red de Tranvías de Tenerife, de cara a la implementación de un sistema de asistencia a la conducción de los tranvías con Inteligencia Artificial.

Palabras clave: Inteligencia Artificial, Deep Learning, Transfer Learning, IA, Tranvía, YOLO, Tenerife, Detección de objetos, Jetson Nano

Abstract

This master's degree final project presents the obstacle detection and traffic light reading system using deep learning Artificial Intelligence that has been designed for trams in the tram network of Tenerife, Canary Islands, Spain.

For its implementation, a state-of-the-art study of the current Artificial Intelligence object detection systems has been carried out, after which a YOLO model of object detection by transfer learning has been selected and trained, to subsequently implement it in a program that will run the model in a Jeson Nano device and generate alerts for the tram driver depending on what has been detected in front of the tram.

AI-based systems are in vogue today, largely thanks to recent advances in modelling and the democratisation of user access to tools for developing AI systems.

This project has demonstrated the feasibility of implementing an Artificial Intelligence system that detects obstacles and reads the traffic lights on the Tenerife Tramway network, with a view to implementing an Artificial Intelligence driving assistance system for trams.

Keywords: *Artificial Intelligence, Deep Learning, Transfer Learning, IA, Tram, YOLO, Tenerife, Object detection, Jetson Nano*

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Objetivo del trabajo de fin de máster

El objetivo general de este trabajo consiste en la selección de una red neuronal y su entrenamiento para su uso en la detección de posibles obstáculos a la circulación, ya sean fijos o móviles, por parte de la red de tranvías de Tenerife, en las Islas Canarias, España, actualmente operada por la compañía Metropolitano de Tenerife, S.A., haciendo uso de un mini ordenador Jetson nano.

Para alcanzar este objetivo general, se han definido una serie de objetivos específicos, u objetivos parciales, que permitan la consecución de este objetivo:

- Estudio del estado del arte de las redes neuronales para detección de objetos: Revisión de los últimos avances realizados en elaboración de redes neuronales para detección de objetos en imágenes y técnicas de entrenamiento.
- Selección de un modelo de detección de objetos en imágenes: Selección de un modelo en función de las características de este, su rendimiento de ejecución en una Jetson nano, y su facilidad para el entrenamiento.
- Elaboración de los datasets de entrenamiento del modelo: Extracción de imágenes del material videográfico suministrado por la compañía y posterior

etiquetado para su uso en el entrenamiento del modelo de IA seleccionado.

- Entrenamiento del modelo elegido: Haciendo uso de los datasets elaborados, entrenar el modelo seleccionado hasta obtener los resultados deseados.

- Elaboración de un programa de ejecución y procesamiento de los resultados devueltos por la IA: Con el modelo ya entrenado, se elaborará un programa encargado de extraer fotogramas del medio de entrada establecido, ya sea un vídeo o una cámara, para procesarlo por el modelo de IA, utilizar los resultados devueltos para determinar los estados de los semáforos que se observen y los obstáculos que pueda haber en la vía, y la correspondiente emisión de alertas en el video devuelto con las imágenes procesadas.

1.2 Marco teórico

1.2.1 Detección de objetos

La detección de objetos es el campo de la visión artificial encargado de la identificación de objetos, ya sea en vídeos o en imágenes. Su objetivo es simple: lograr que la un ordenador sea capaz de tener un cierto nivel de comprensión de lo que contiene una imagen, de manera similar a cómo lo hace naturalmente el ser humano. Hoy en día, se usa en diferentes campos, como la conducción autónoma, la inspección insdustrial y la visión robótica [1].

Para lograrlo, un algoritmo de detección de objetos debe [2]:

- Ser capaz de detectar varios objetos.

- Determinar sus coordenadas X e Y en la imagen y generar un rectángulo a su alrededor.

- En el caso del procesamiento de videos, hacerlo en un tiempo razonable, ya que si no, pueden escaparse datos relevantes.

Figura 1.1: Ejemplo de imagen con detección de frutas [3]

En la actualidad, se usan principalmente dos tipos de algoritmos basados en Inteligencia Artificial para la detección de objetos: el Machine Learning y el Deep Learning [1].

1.2.2 Redes neuronales. Machine Learning vs Deep Learning

Tanto Machine Learning como Deep learning son técnicas de inteligencia artificial basadas en redes neuronales, que son programas que toman decisiones de una forma parecida a la que hace el cerebro humano, mediante procesos que imitan las neuronas biológicas [4]. Para ello, se basan en el uso del perceptrón, un elemento con un cierto número de entradas, con un peso cada una, que se suman en función de los pesos y una ecuación de activación de la salida para determinar si la salida del perceptrón (neurona) se activa. Los pesos asignados a cada entrada se definen durante el proceso de entrenamiento de la red [5].

En el campo de la detección de objetos, las diferencias entre Machine Learning y Deep learning son [1]:

- Machine Learning: En este caso, la red neuronal solo se encarga de la clasificación de la imagen. Para ello, recibe las características relevantes

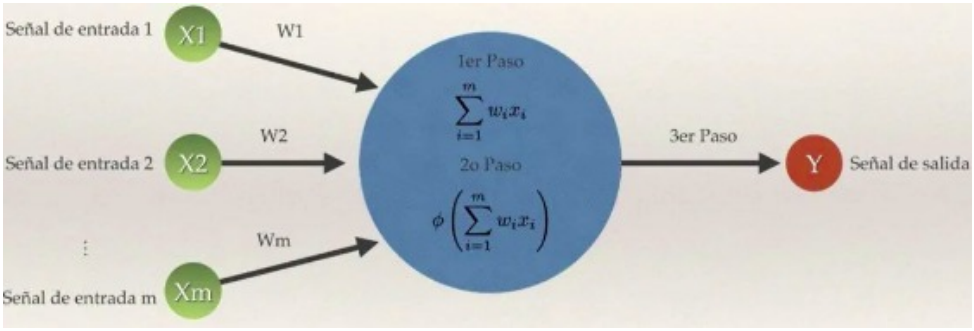


Figura 1.2: Ejemplo de perceptrón [5]

extraídas de la imagen de diversas maneras, ya sea de forma manual por el usuario o por medio de algún algoritmo, como el histograma de de gradientes orientados (HOG) o el patrón binario local (LBP), para luego aplicar el tipo de clasificador entrenado y determinar el contenido de la imagen.

- Deep Learning: En este caso, la red se encarga no solo de la parte de clasificación, sino también de la extracción de características relevantes de la imagen. Para ello, se hace uso de la llamadas redes neuronales convolucionales (CNN, por su siglas en inglés), que incluyen, junto a la red clasificadora, unas capas de neuronas, conocidas como capas convolucionales, que se encargan de extraer la información relevante para la clasificación de los elementos de la imagen. Estas capas se entrenan junto al clasificador durante el proceso de entrenamiento para que puedan trabajar en conjunto.

En la figura 1.3 podemos visualizar la diferencia entre ambos tipos:

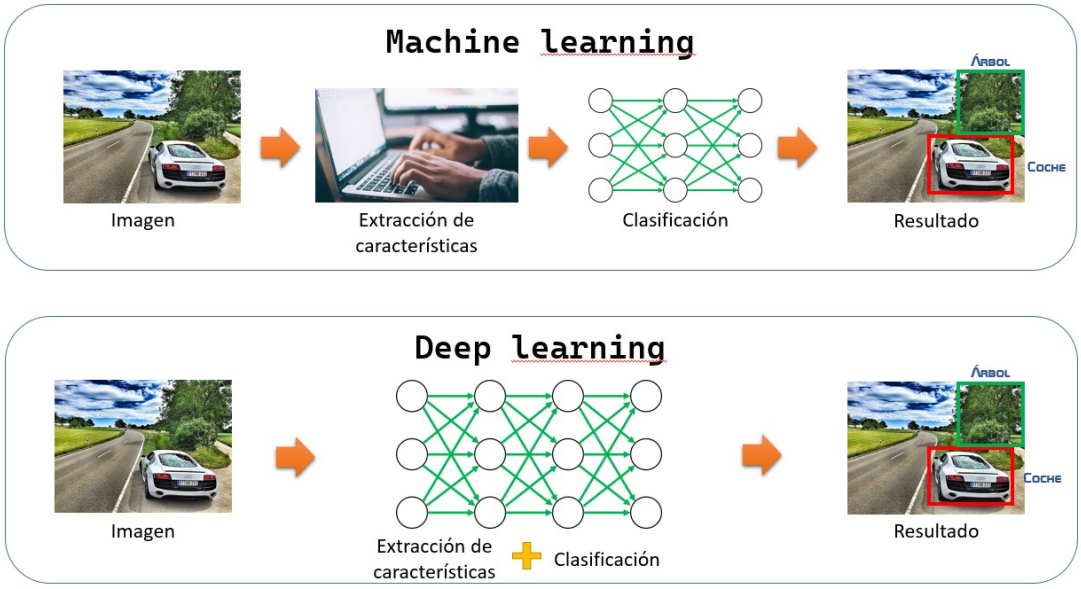


Figura 1.3: Diferencias entre Machine Learning y Deep Learning

A la hora de determinar si decantarnos por uno u otro, se depende generalmente del tipo de aplicación que se quiera realizar, pero en caso de duda, el principal criterio es si se posee una GPU potente y una gran cantidad de imágenes etiquetadas. Si alguno de estos elementos falta, será mejor decantarse por el machine learning, ya que tiene menos requisitos de potencia computacional e imágenes para su entrenamiento [1]. Teniendo en cuenta el equipo con el que se ha contado, que veremos en el capítulo 2, para este trabajo nos hemos decantado por el uso de Deep Learning.

A su vez, dentro de Deep Learning existen dos formas de aproximar el problema [1]:

- Creación de un modelo customizado: Consiste en crear un modelo CNN desde cero, configurando manualmente las diferentes capas que la conforman. Esta técnica requiere la potencia suficiente para el compilado de las capas, además de una enorme cantidad de imágenes etiquetadas para su entrenamiento, pero a cambio, ofrecen resultados destacables y ajustados a la tarea a realizar.

- Uso de modelos preentrenados: También conocido como Transfer Learning, consiste en el uso de un modelo ya entrenado para reentrenarlo para que se ajuste a nuestra aplicación concreta. Se basa en en la idea de que, si el modelo original ya estaba preparado para una tarea similar a la que estemos implementando, gran parte de la información original del modelo podrá aprovecharse para el nuevo modelo. Esta técnica permite reducir enormemente el número de imágenes necesarias para entrenar el modelo, ya que gran parte de la información necesaria para su funcionamiento ya ha sido implementada [6].

1.2.3 Medición de resultados de entrenamientos. Métricas

Una vez realizado el entrenamiento del modelo, es necesario saber cuán óptimo ha sido el entrenamiento y como de eficiente es el modelo resultante del entrenamiento. Para ello, existen una serie de métricas que buscan medir el desempeño del modelo, cuantos objetos es capaz de detectar y cuantos falsos positivos da.

Existen muchos tipos de métricas para medir el desempeño de un modelo, según el tipo de problema ha resolver y el modelo en cuestión, pero las siguientes son las más relevantes para medir el desempeño del modelo resultante del entrenamiento [7]:

- Matriz de confusión: Es una métrica que nos permite averiguar como de "confuso" está un modelo a la hora de clasificar una detección como de una clase u otra. Como su propio nombre indica, se trata de una matriz donde las filas corresponden a los valores verdaderos, y las columnas a los valores predichos. Los elementos en la diagonal de la matriz indican los

verdaderos positivos (TP) del modelo, es decir, las veces que el modelo ha acertado a la hora de predecir un objeto de una clase, mientras que los valores por encima de la diagonal indican cada vez que, siendo una clase, le ha asignado otra al objeto, causando un falso negativo (FN), y los valores por debajo de la diagonal indican cada vez que, siendo otra clase, el modelo ha asignado esa clase, dando un falso positivo (FP) a la clase. En la figura 1.4 podemos observar un esquema de la distribución de valores en una matriz de confusión.

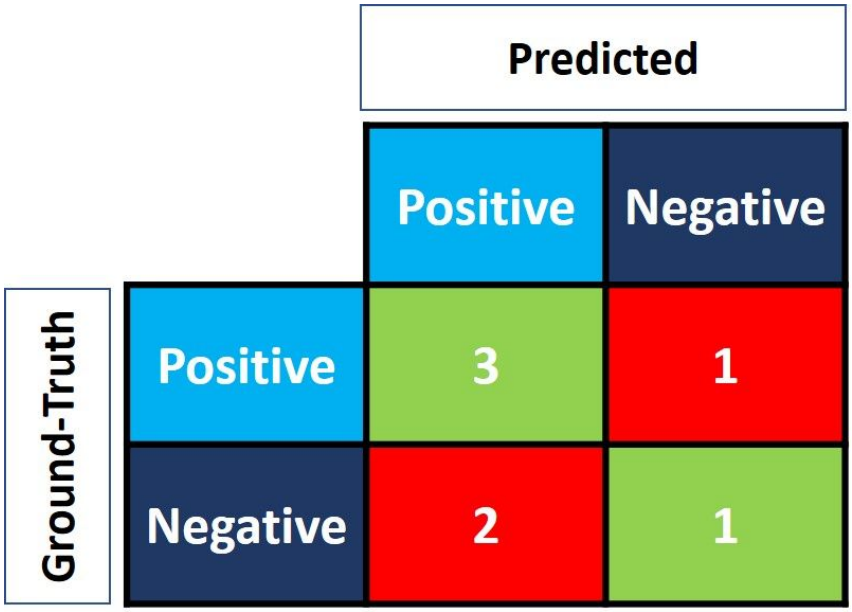


Figura 1.4: Esquema de matriz de confusión [8]

- Intersección sobre la unión (loU): Esta métrica mide el nivel de solapamiento entre el recuadro predicho alrededor del objeto localizado y el recuadro real del objeto en el etiquetado. Esta métrica permite determinar la precisión del modelo a la hora de determinar la ubicación de un objeto en la imagen y, por tanto, un valor bajo indica que el modelo tiene problemas para determinar la ubicación de un objeto [7].

- Precisión (P): Una de las métricas más utilizadas en todos los problemas

de detección y clasificación, determina que proporción de los positivos dados por el modelo son correctos. En el caso de la detección de objetos, cuando nos referimos a una clase, determina cuantas de las veces en las que el modelo asignó una clase a un objeto, esta era la clase correcta. Cuanto mayor sea este valor, menos veces se equivoca el modelo a la hora de asignar una clase a un objeto [8]. La ecuación de la precisión es la 1.1:

$$Precision = \frac{True_{positeve}(TP)}{True_{positeve}(TP) + False_{positeve}(FP)} \tag{1.1}$$

- Recall (R): También conocida como recuperación o sensibilidad, es otra de las métricas más utilizadas en todos los problemas de detección y clasificación. A diferencia de la precisión, determina cuantos positivos detecta de todos los posibles. En el caso de la detección de objetos, refiriéndonos a una clase, determina cuantos de todos los posibles objetos de una clase detecta. Cuanto mayor sea este valor, menos dificultades tiene el modelo para distinguir objetos de esa clase del resto del entorno [8] La ecuación de la sensibilidad es la 1.2:

$$Precision = \frac{True_{positeve}(TP)}{True_{positeve}(TP) + False_{negative}(FN)} \tag{1.2}$$

- Precisión promedio (AP): Esta métrica corresponde al área que queda por debajo de la curva por debajo de la curva precisión-recall, proporcionando una medida de la relación entre ambas métricas y su balance en el modelo. En un modelo donde se detecten varias clases de objetos, cada clase tiene su propio valor de AP que indica, de forma general, cuanto le cuesta al modelo detectar objetos de esa clase [7].

- Precisión promedio media (mAP): Esta métrica es una versión extendida del

AP, usada en los modelos que detectan más de una clase, y que corresponde a la media del AP de todas las clases presentes en el modelo,proporcionando así una imagen general del rendimiento del modelo. Un bajo valor en esta métrica indica una falta de rendiminto general del modelo. Esta métrica, a su vez, se divide en dos tipos [7]:

  - mAP50: En este caso, se calcula la media del AP imponiendo como condición que solo se tengan en cuenta las detecciones que obtengan un valor de loU mayor de 0,5. De esta forma, se busca medir el rendimiento del modelo solo con las detecciones “fáciles”.
  - mAP50-95: En este caso, se calcula la media del AP variando el límite de loU a partir del cual se considera la detección en un rango que va de 0,5 a 0,95. De esta manera, se trata de obtener una visión del rendimiento del modelo a lo largo de varios niveles de dificultad de detección.

- Puntación F1 (F1-score): Esta métrica se obtiene calculando la media armónica entre la precisión y el recall de una clase [9], lo que permite obtener una idea del rendimiento general del modelo, teniendo en cuenta tanto los falsos positivos como los falsos negativos. Un valor bajo o desbalanceado en esta métrica indica que hay disparidad entre la P y R [7]. Su ecuación es 1.3:

$$Precision = 2 * \frac{P * R}{P + R} \tag{1.3}$$

1.3 Metodología

Para la realización de este trabajo y poder cumplir con los objetivos marcados en el apartado 1.1, se han seguido los siguientes puntos:

1. Estudio del estado del arte: Análisis de las técnicas que existen para la detección de objetos en el ámbito de la inteligencia artificial, para la determinación de la estrategia a seguir para el desarrollo del proyecto.

2. Selección del modelo: Búsqueda de información sobre modelos compatibles con el hardware disponible y selección del más adecuado para el problema a resolver.

3. Elaboración de los datasets de entrenamiento: Se obtendrán las imágenes necesarias para el entrenamiento, se etiquetarán y se dispondrá en un dataset listo para el entrenamiento del modelo. Esta fase se repetirá tantas veces como modelos haga falta preparar hasta la obtención del modelo final.

4. Entrenamiento del modelo: uso de los datasets elaborados para entrenar el modelo elegido hasta la obtención de un modelo capaz de detectar todos los elementos considerados relevantes para nuestro sistema.

5. Elaboración del sistema de detección: Elaboración de un sistema que aproveche la información devuelta por del modelo de las imágenes suministradas para elaborar un sistema que avise visualmente de cualquier obstáculo o elemento relevante para la circulación del tranvía que haya en la imagen. Esta fase comenzará cuando se obtenga el primer modelo entrenado, pero continuará a medida que se realicen nuevos modelos, tanto para adaptar el sistema a las características del nuevo modelo como la introducción progresiva de nuevas mejoras.

Capítulo 2

Tecnologías

En este capítulo, se hablará de las tecnologías utilizadas para el desarrollo del proyecto, tanto el hardware utilizado como las librerías y programas utilizados, así como la arquitectura del sistema a implementar.

2.1 Tecnologías utilizadas

2.1.1 Hardware

Para el desarrollo del proyecto, se ha contado con dos piezas principales de hardware.

La primera es el ordenador personal del autor de este trabajo. Se trata de un ordenador custom, montado a partir de componentes adquiridos a lo largo de los años. Usa Windows 10 como sistema operativo y sus características principales son las siguientes:

- Procesador Intel Core i7-8700 a 3,2 GHz.
- 32 GB de memoria RAM.
- Disco duro SSD de 1 TB.
- Tarjeta gráfica NVIDIA RTX 4060 8GB con 242 AI TOPS [10].

Los AI TOPS es una medida para medir el rendimiento bruto de un dispositivo a la hora de realizar tareas de Inteligencia artificial. En un dispositivo dedicado a procesamiento de redes neuronales (Como una NPU, Neural Processor Unit), o con núcleos dedicados a estas tareas (como las tarjetas gráficas más relativamente modernas), los TOPS definen el número de trillones de operaciones que puede realizar el dispositivo para tareas relacionadas con la IA [11].

La segunda pieza de hardware utilizada es un kit de desarrollo jetson nano, modelo de 4GB. Fabricado por la empresa NVIDIA, se trata de un pequeño ordenador preparado para la ejecución de redes neuronales en aplicaciones como clasificación de imágenes, detección de objetos, segmentación y procesamiento del lenguaje. Fue concebido para el prototipado de aplicaciones basadas en IA, permitiendo su prueba de cara a la posterior explotación comercial [12]. Su sistema operativo es Jetpack SDK 4.6.5, que incluye NVIDIA L4T 32.6.1, una distribución de linux basada en Ubuntu 18.04, adaptada para arquitecturas ARM64 y con una capa de personalización propia [13].

De acuerdo con el fabricante, el kit de desarrollo jetson nano cuenta con las siguientes especificaciones [12]:

GPU	Arquitectura NVIDIA Maxwell™ con 128 núcleos NVIDIA CUDA
CPU	Procesador ARM® Cortex®-A57 MPCore de cuatro núcleos
Memoria	LPDDR4 de 4 GB y 64 bits
Almacenamiento	16 GB de almacenamiento Flash eMMC 5.1
Codificación de vídeo	4K a 30 cuadros (H.264/H.265)
Decodificación de vídeo	4K a 60 cuadros (H.264/H.265)
Entrada Cámara	12 vías (3 x 4 o 4 x 2) MIPI CSI-2 DPHY 1.1 (18 Gbps)
Conectividad	Gigabit Ethernet
Salida pantalla	HDMI 2.0 o DP 1.2 | eDP 1.4 | DSI (1 x 2) 2 simultáneos
UPHY	1 1/2/4 PCIE, 1 USB 3.0, 3 USB 2.0
E/S	1 SDIO / 2 SPI / 4 I2C / 2 I2S / GPIO
Tamaño	69,6 mm x 45 mm
Mecánicas	Conector de 260 pines

Tabla 2.1: Especificaciones kit de desarrollo Jeson Nano 4GB [12]

Las especificaciones de la tabla 2.1 proporcionan a este kit una potencia bruta de 472 GFLOPs [12] de GPU, o lo que es lo mismo, 10^9 operaciones de punto flotante por segundo, suficientes para el manejo básico de redes neuronales.

Figura 2.1: Kit de desarrollo Jetson Nano [12]

Sin embargo, los 16 GB de memoria Flash integrada en la Jetson Nano son insuficientes para la mayoría de aplicaciones, por lo que se le añadió una tarjeta de memoria Flash de 256 GB para ampliársela y es donde se le instaló el sistema operativo.

2.1.2 Selección del modelo de inteligencia artificial

Tal y como se vio en el apartado 1.2.2, dentro del campo del Deep Learning para detección de objetos existen dos forma de aproximar el problema: Con un modelo desde cero y con transfer learning. De estas dos opciones, se optó finalmente para este proyecto por el transfer learning, ya que nos permite no solo reducir el número de imágenes necesarias para el entrenamiento, sino también no tener que profundizar en el diseño de redes neuronales desde cero, pudiendo trabajar directamente con modelos elaborados por empresas e investigadores profesionales con más experiencia en ese campo, y cuya eficiencia en el campo de la detección de objetos está más que probada.

En un primer momento, y siguiendo las indicaciones del repositorio "Hello AI world", creado por Dustin Franklin, desarrollador de NVIDIA [14], se consideró el uso del modelo SSD-mobilenet.

SSD-mobilenet es un modelo de detección de objetos presentado en 2017 por investigadores de Google AI. Se trata de un modelo basado en capas de convolución profunda separadas, lo que permite obtener modelos bastante livianos. Posee solo un par de hiperparámetros, que permiten ajustar eficientemente el modelo en función de si lo que se busca es rapidez o precisión, permitiendo así a los construtores de modelos dar con la configuración más adecuada según la aplicación a realizar [15]. En la tabla 2.2 podemos observar la arquitectura del modelo según sus autores [15]:

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Tabla 2.2: Arquitectura modelo Mobilenet [15]

El repositorio “Hello AI world” trae todas las herramientas necesarias para el entrenamiento de modelos SSD-Mobilenet e instrucciones detalladas de como utilizarlas. Sin embargo, vienen con la limitación de que solo pueden usarse en la propia Jetson Nano, lo que sumado ha que la Jetson Nano está optimizada para ejecutar modelos, no así entrenarlos (según mediciones de la propia NVIDIA, un entrenamiento de 6375 imágenes de 35 épocas tardaría aproximadamente 8 horas, a 4,77 imagenes por segundo [3][16]) no lo hace una opción viable para el proyecto.

En un intento de paliar esta deficiencia, se buscaron formas de poder ejecutar las herramientas de entrenamiento. Se encontró la opción de usar una hoja de google colab para ejecutar los scripts necesarios, sin embargo, la necesidad de

tener que cargar el entorno cada vez que se necesitaba hacer un nuevo modelo y tratar mantener una conexión constante y estable con el entorno durante todas las horas que dure el entrenamiento para no perder los resultados del mismo hacen que, en base a la experiencia previa del autor de este trabajo en proyectos similares, no sean una opción viable, por lo que se decidió descartar este modelo y buscar otro que permita su entrenamiento de forma local en el ordenador definido en el apartado 2.1.1.

Con el criterio de poder entrenar el modelo de forma local impuesto, se consideró la opción de usar el modelo YOLO. YOLO (You Only Look Once) es un modelo creado por Joseph Redmon y Ali Farhadi de la Universidad de Washington, lanzado en 2015 y mantenido actualmente a través de la empresa Ultralytics, que se ha vuelto especialmente popular en los últimos años debido a su alta velocidad de procesamiento y su precisión [17]. Actualmente, aunque van por la versión 10 del modelo, no todos los modelos son elaborados por la propia Ultralytics, ya que, debido a su mentalidad open-source, otras entidades han colaborado en el proyecto de YOLO, aportando sus mejoras y versiones.

Para el uso de YOLO, Ultralytics ofrece dos licencias [17]:

- Licencia AGPL-3.0: Licencia open-source que permite su uso libre y gratuito para particulares y estudiantes.
- Licencia Enterprise: Orientada al uso comercial, permitiendo su uso para aplicaciones comerciales.

El modelo YOLO se mostró interesante, ya que permitía el entrenamiento de forma local haciendo uso de las librerías y repositorios oficiales de la compañía, además de la extensa documentación puesta a disposición de los usuarios por la compañía para su uso, por lo que se eligió como modelo definitivo para

desarrollar nuestro sistema. Sin embargo, a pesar de que la versión más moderna del modelo es la V10, se ha tenido que implementar la versión V5 debido a unos problemas de compatibilidad con el software de la Jetson Nano que impidieron el uso de versiones más modernas. Estos problemas de compatibilidad serán desarrollados en más profundidad en el apartado 2.1.3. En la url [18] podemos observar un diagrama de la arquitectura del modelo YOLOv5.

2.1.3 Librerías

Una vez definido el hardware y el modelo utilizado para el sistema, solo queda por definir el software utilizado en la elaboración del proyecto.

El lenguaje de programación básico usado para todos los scripts y programas utilizados es Python, versión 3.12.2 en el ordenador personal y 3.6.2 en la Jetson Nano. Python es un lenguaje orientado a objetos ampliamente utilizado para multitud de tareas, en virtud de su sencillez de lectura y escritura y la extensa cantidad de módulos disponibles para ampliar sus funcionalidades [19], entre ellas el desarrollo de inteligencia artificial, lo que, sumado a la extensa cantidad de documentación disponible sobre su uso, lo hace especialmente interesante para este tipo de proyectos.

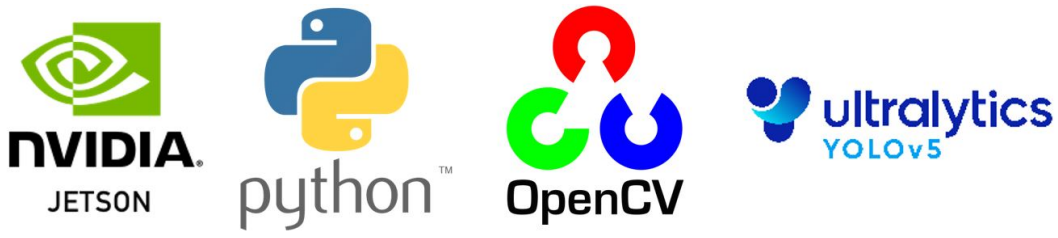


Figura 2.2: Logos de las tecnologías utilizadas. Jetson Nano, Python, OpenCV y Yolo V5

Para la extracción de las imágenes de los vídeos suministrados, se ha hecho uso del programa VLC Media Player, un programa de reproducción de vídeo open-source, que también cuenta con funcionalidades para, entre otras cosas,

extraer frames de los videos durante la reproducción y guardarlos, mediante el filtro “vídeo en escena”.

El etiquetado de las imágenes obtenidas se ha realizado con el programa labelImg, un herramienta open-source creada originalmente por el usuario de Github Tzutalin y ahora mantenido por la comunidad Label Studio. Se ejecuta sobre Python y permite el recuadramiento y anotación de objetos en imágenes de forma sencilla. Cuenta con una interfaz simple, fácil de entender y utilizar, e incorpora soporte para la mayoría de formatos de anotación usados en el entrenamiento de sistemas Deep Learning, entre ellos el formato usado por los modelos YOLO [20]. En la figura 2.3 podemos observar un ejemplo de la interfaz.

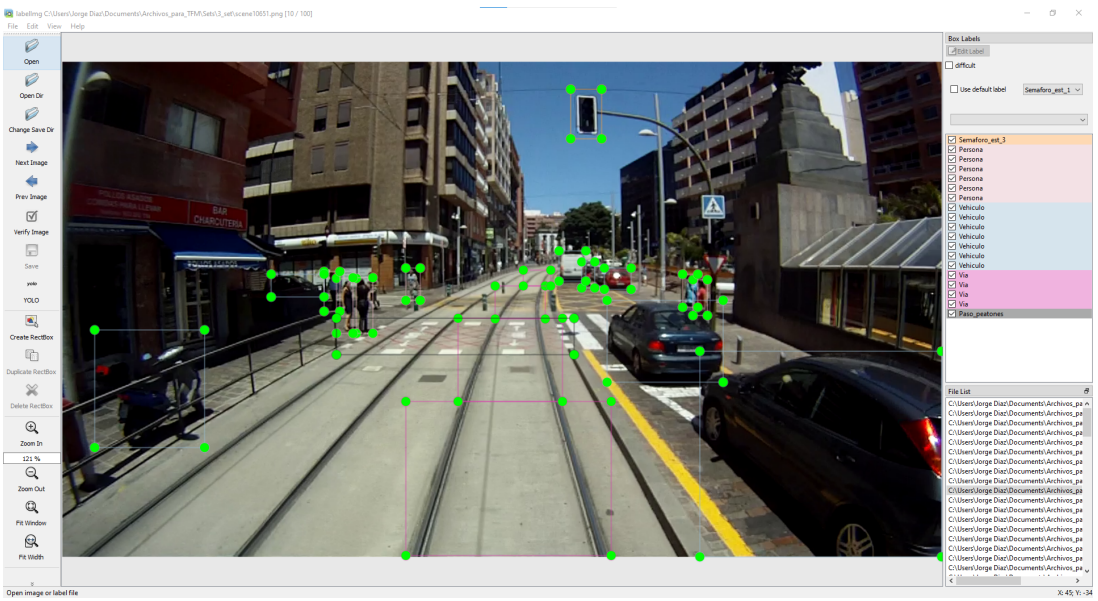


Figura 2.3: Interfaz de labelImg

Para poder entrenar con YOLO, se descargó la librería oficial de YOLO [21], liberadas por Ultralytics, en el ordenador Windows mencionado en el apartado 2.1.1. Para su correcto funcionamiento hizo falta la instalación de las siguientes librerías, que eran instaladas automáticamente al ejecutar el comando “pip install -r requirements.txt” en la carpeta raiz del repositorio [22]:

- Gitpython versión 3.1.30 o superior
- Matplotlib versión 3.3 o superior
- Numpy versión 1.23.5 o superior
- Opencv-python versión 4.1.1 o superior
- Pillow versión 10.3.0 o superior
- Psutil
- PyYAML versión 5.3.1 o superior
- Requests versión 2.32.0 o superior
- Scipy versión 1.4.1 o superior
- Thop versión 0.1.1 o superior
- Torch versión 1.8.0 o superior
- Torchvision versión 0.9.0 o superior
- Tqdm versión 4.64.0 o superior
- Ultralytics versión 8.2.34 o superior

Sin embargo, a la hora de usar esta misma librería para ejecutar un modelo YOLO en la Jetson Nano, se encontró con un problema de compatibilidad imprevisto: Muchas de las librerías necesarias para ejecutar usar el repositorio solo eran compatibles con versiones de Python 3.8.0 en adelante. Dado que, como mencionamos anteriormente, la versión de Python de la Jetson Nano es 3.6.2, y esta no puede ser actualizada, al estar intrínsecamente unida a la versión del sistema operativo, resultó imposible usar las librerías oficiales en este dispositivo.

Sin embargo, tras una búsqueda por internet, se pudo encontrar un repositorio llamado “JetsonYolov5”, creado por el usuario de Github mailrocketsystems [23], quien había adaptado las librerías y repositorio oficiales de Ultralytics para funcionar en Python 3.6.2, lo que permitía la ejecución de estos modelos en una Jetson Nano. Por contra, este repositorio solo estaba preparado para su uso con modelos de YOLOv5, razón por la que, tal y como se indicó en el aparatado 2.1.2, se haya decido usar esa versión del modelo para nuestro sistema.

Para el uso del repositorio “JetsonYolov5”, ha sido necesario instalar en la Jetson nano las siguientes librerías [23]:

- Numpy versión 1.19.0
- Pandas versión 0.22.0
- Pillow versión 8.4.0
- PyYAML versión 3.12
- Scipy versión 1.5.4
- Psutil
- Tqdm versión 4.64.1
- Imutils
- Pycuda, para cuyo correcto funcionamiento es necesario generar algunas rutas en las variables del sistema, mediante los comandos “export PATH=/usr/local/cuda-10.2/bin${PATH:+:${PATH}}” y “export LD_ LIBRARY _PATH=/usr/local/cuda-10.2/lib64:$LD_LIBRARY_PATH”
- Seaborn

- Torch versión 1.10.0, aunque en este caso, no la versión normal obtenida con el comando "pip", ya que la versión obtenida sería la versión para arquitecturas X86, sino una versión modificada, distribuida por NVIDIA, que adapta la librería a arquitecturas ARM64 [24].
- Torchvision versión 0.11.1, obtenida en este caso directamente del repositorio oficial de la librería.

Por último, se mencionan a continuación las librerías que, más allá de haber servido como dependencias de otras librerías, han sido utilizadas para la elaboración del sistema, u otras librerías relevantes para el diseño:

- Pandas y scikit-learn, para la elaboración de los datasets de entrenamiento.
- Opencv, para la lectura de las entradas de vídeo, manipulación de imágenes y generación de vídeos de salida.
- Numpy, para el manejo de arrays.
- Shutil, para el movimiento automatizado de un gran número de archivos de imágenes y anotaciones.
- Pyautogui, para la creación rápida de una interfaz de control.

2.2 Arquitectura

En este apartado, se definirá la arquitectura del sistema, definiendo el flujo de información desde que se extraen las imágenes hasta que se tiene un modelo procesando entradas de video.

Las fases del sistema son las siguientes:

1. Se extraen las imágenes de los vídeos suministrados para entrenar el modelo haciendo uso del programa VLC Media Player.

2. Se etiquetan las imágenes nuevas que hayan con el programa labelImg, teniendo en cuenta las etiquetas definidas y hasta que punto se considera que el modelo debe "ver" un objeto (Por lejanía, por obstrucción de otro obstáculo, etc.).

3. Elaboración del dataset de entrenamiento, separando las imágenes y sus correspondientes archivos de etiquetas en los correspondientes grupos de imagenes de entrenamiento y validación, tratando de mantener una proporción de un 70 % de imágenes para entrenamiento y un 30 % de imágenes para validación.

4. Se realiza el entrenamiento con los parámetros definidos.

5. Se analizan los resultados de las métricas y se realiza un primer procesado con el programa de test de detecciones suministrado por el repositorio oficial de Ultralytics. Si los resultados indican una mejora sobre modelos anteriores, se envía a la Jetson Nano para continuar el proceso. Si no, se vuelve al punto 2 para elaborar un nuevo dataset.

6. Una vez en la Jetson Nano, el modelo se convierte a formato wts.

7. Una vez obtenido el archivo en formato wts, se siguen los pasos para compilar el programa que generará el para convertir el modelo a formato tensorRT y se ejecuta.

8. Con el modelo ya en formato tensorRT, se ejecuta el programa de detección:

   a) Se lee el modelo y se inicializa la clase de detección.

   b) Se inicializa la captura de video y el archivo de video de salida del programa.

c) Si la inicialización de la captura de video es correcta, la ejecución continua. Si no, pasa al punto k para finalizar la ejecución.

d) Se lee un frame de la captura de video. Si la lectura es correcta, el programa prosigue. Si no, pasa al punto m para finalizar la ejecución.

e) Se redimensiona la imagen para ajustarla a las dimensiones esperadas por el modelo.

f) Se le suministra la imagen a la clase de detección para que procese la imagen y devuelva lo detectado, junto al frame con los resultados.

g) Se recorre el diccionario devuelto por la clase de detección para analizar que ha detectado la IA y separar lo relevante del resto.

h) Determinar si un objeto está en la vía mediante análisis del solapamiento de sus cuadros de ubicación.

i) Escribir las alarmas que se hayan podido activar en función del análisis de las clases deseadas y resaltado de los elementos que la activaron (si procede).

j) Redimensionado de la imagen para ajustarla a la resolución del archivo de salida.

k) Escritura del frame en el archivo de salida y mostrado por pantalla. Si no se ha pulsado la tecla ‘q”, se vuelve al punto d, si no, se continua para cerrar el programa.

l) Se libera el medio de captura de vídeo y el archivo de salida, se cierran todas las ventanas y se finaliza el script.

En las figuras 2.4, 2.5 y 2.6 podemos ver los flujogramas de la arquitectura y la ejecución del del programa de detección.

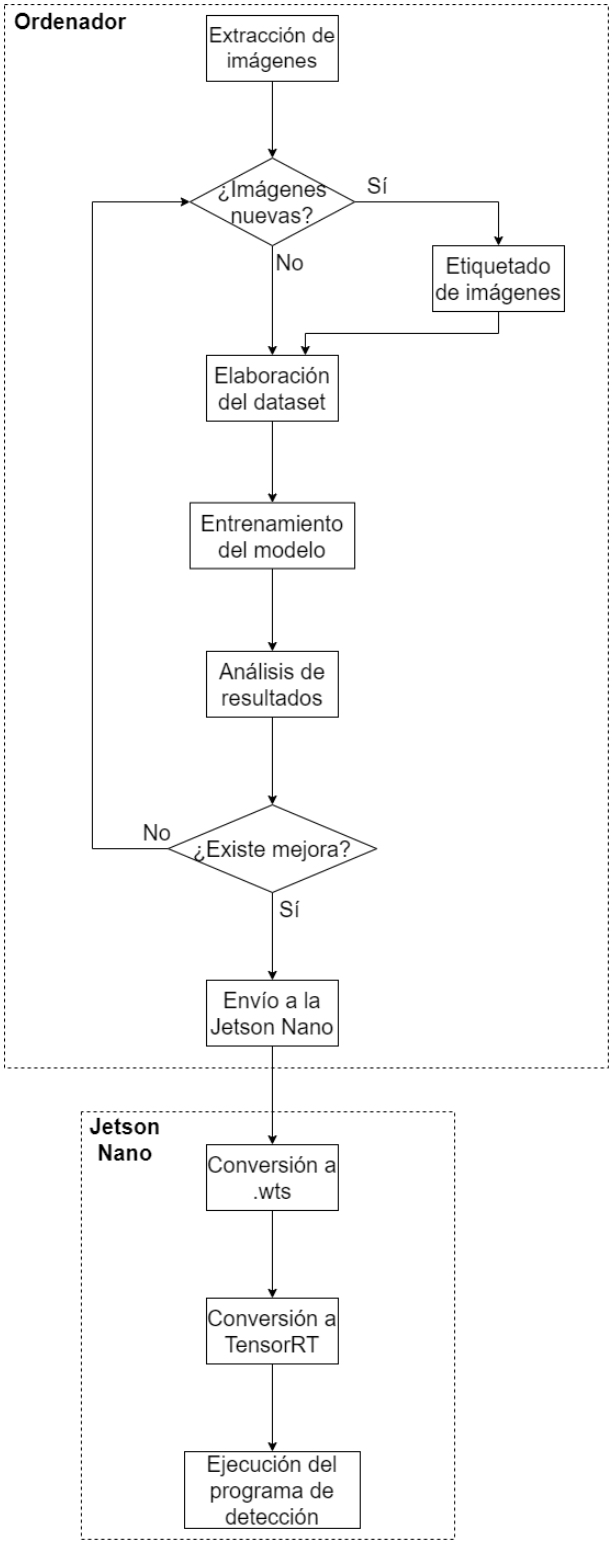


Figura 2.4: Flujograma de la arquitectura del modelo

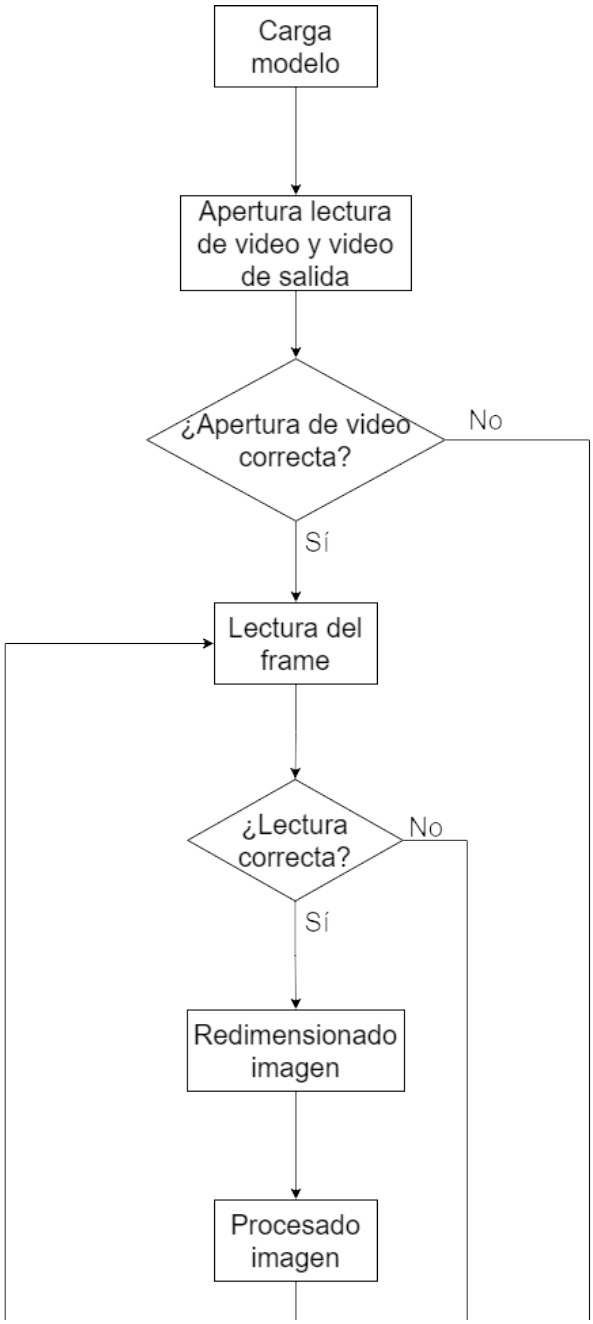


Figura 2.5: Primera parte del flujograma de la ejecución del programa

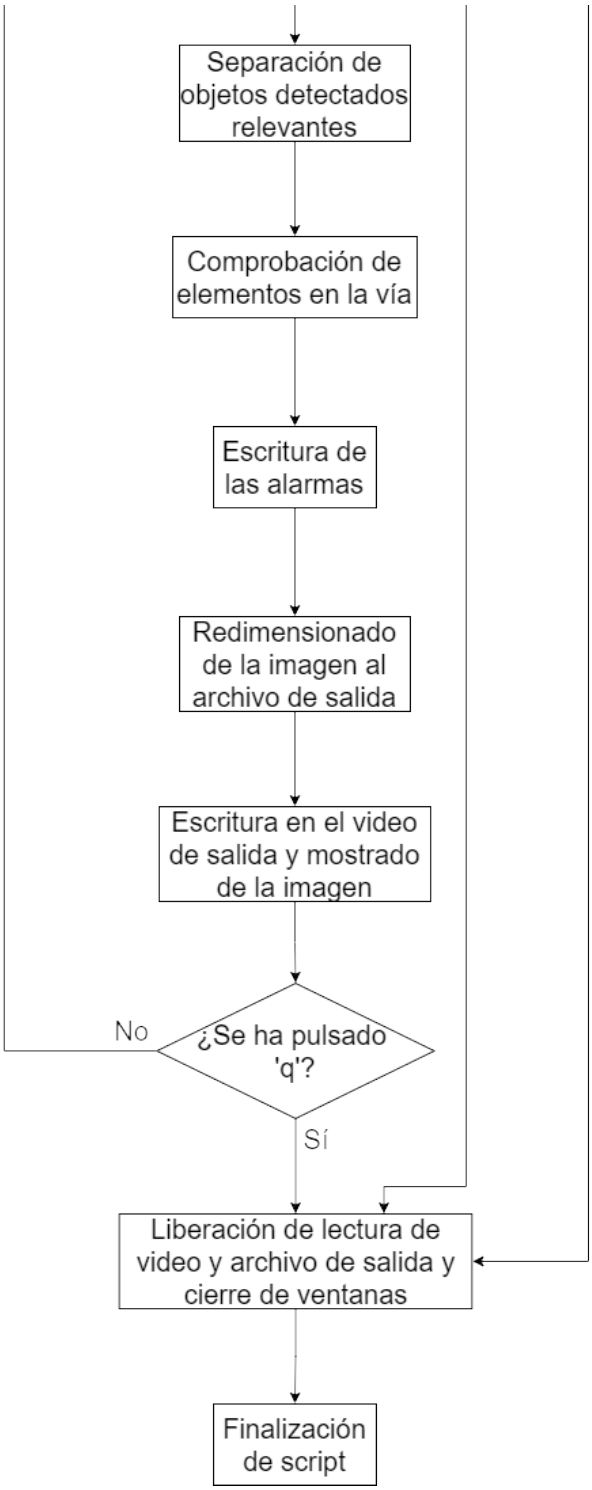


Figura 2.6: Segunda parte del flujograma de la ejecución del programa

Capítulo 3

Desarrollo del sistema

En los capítulos anteriores, se han explicado los conceptos necesarios para el desarrollo de sistemas de detección de objetos con Inteligencia Artificial y las tecnologías necesarias para el desarrollo de nuestro sistema.

En este capítulo, se mostrará el desarrollo del sistema, un resumen de los resultados intermedios obtenidos y las soluciones aplicadas en siguientes iteraciones hasta la obtención del modelo final, tanto para el desarrollo del modelo de detección como el programa de procesamiento de los datos devueltos por la IA.

3.1 Desarrollo del modelo de detección de objetos

3.1.1 Primer entrenamiento

En este primer entrenamiento, el objetivo era probar el proceso completo de etiquetado, entrenamiento, procesamiento del modelo y ejecución en la Jetson Nano, para así identificar cualquier problema que pudiera existir en cualquier fase del proceso y corregirla de cara a los siguientes entrenamientos, así como obtener una idea inicial del rendimiento del sistema en general.

Para este fin, tomando el vídeo de prueba suministrado, correspondiente a un trayecto completo en un sentido, se extrajeron imágenes de los primeros

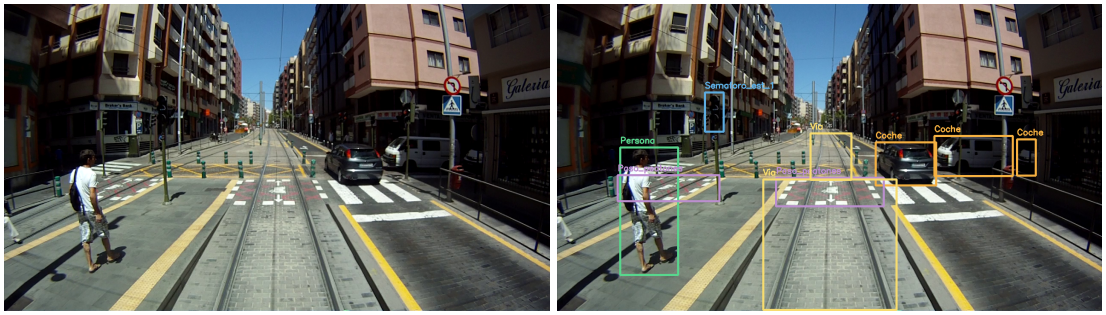
15 minutos, de entre las cuales se seleccionaron 110 para conformar el primer dataset de entrenamiento. Estas imágenes se etiquetaron usando el programa labelImg presentado en el apartado 2.1.3 y guardando los archivos de etiquetas salientes en una carpeta “labels” creada dentro del directorio de almacenamiento de las imágenes, usando las siguientes etiquetas:

- Semaforo_est_1.
- Persona.
- Coche.
- Paso_peatones.
- Via.
- Semaforo_est_2.
- Tranvia.
- Semaforo_est_3.
- Moto.
- Limite_30 (Límites de velocidad de 30 Km/h).
- Signal (Cualquier señal que no sea un límite de velocidad).

En esta fase se presentó el primer gran problema: En el archivo de etiquetas asociado a cada imagen, en formato YOLO, cada etiqueta se asocia a una clase a través de un número de clase, que se asigna a cada clase en función de la primera vez que un objeto fue etiquetado con esta clase. Sin embargo, al abrir y cerrar de nuevo el programa labelImg, esta asignación se pierde, resultando en que, si al retomar el trabajo tras reabrir el programa, no se respeta el orden de etiquetado

que se había seguido anteriormente, este sobrescribía al antiguo, corrompiendo los etiquetado ya hechos anteriormente. tras analizarlo, este problema pudo resolverse mediante la predefinición de las clases a utilizar previo al inicio del programa, mediante la introducción de estas en un archivo de configuración específico. De esta manera, las clases y su número asignado se mantienen fijos de una sesión a la siguiente.

En la figura 3.1 se puede observar la comparación entre como era una imagen sin etiquetar, y como queda la imagen en base a la información que se almacena en su archivo de etiquetas asociado tras el proceso de etiquetado de objetos.



(a) Imagen sin etiquetar *(b) Imagen etiquetada*

Figura 3.1: Comparativa de imágenes antes y después del etiquetado

Tras haber etiquetado todas las imágenes del dataset, se implementó una hoja de jupyter, llamada Separar_conjuntos_entrenamiento.ipynb, donde, tras leer los nombres de los archivos almacenados en la carpeta de almacenamiento de imágenes y generar una lista con ellos, procede a generar un dataframe con esa lista y se somete a la función train_test_split de scikit-learn, que nos garantiza una separación aleatoria de la lista de imágenes en dos conjuntos de entrenamiento y validación en la proporción deseada. Finalmente, tras reconvertir ambos dataframes de entrenamiento y validación a listas, se recorren ambas, transfiriendo cada imagen y su archivo de etiquetas correspondiente a la carpeta correspondiente del archivo del dataset, que obedece a la distribución mostrada

en la figura 3.2. El código de las celdas de la hoja de jupyter pueden consultarse en el apéndice A.1.

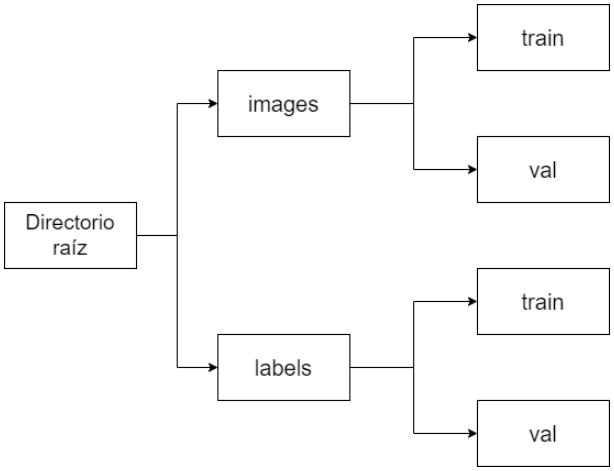


Figura 3.2: Esquema de archivos para entrenamiento de YOLOv5

Una vez distribuidas las imágenes y los archivos de etiquetas en los archivos correspondientes, se genera el archivo .yaml que indica al programa de entrenamiento las rutas al dataset y las etiquetas usadas, y finalmente se tuvo el dataset listo para su uso, quedando 77 imagenes para entrenamiento y 33 para validación.

Finalizado el dataset, se inició el entrenamiento con la versión slim de YOLOv5 de 100 épocas con una configuración de batch_size de 16. El modelo slim de YOLOv5 es la segunda versión más liviana del modelo, que permite tener una buen tiempo de respuesta al tiempo que mantiene buena precisión [22]. En la figura 3.3 podemos ver una gráfica comparativa de las diferentes versiones existentes de YOLOv5 ante datasets de diferente tamaño.

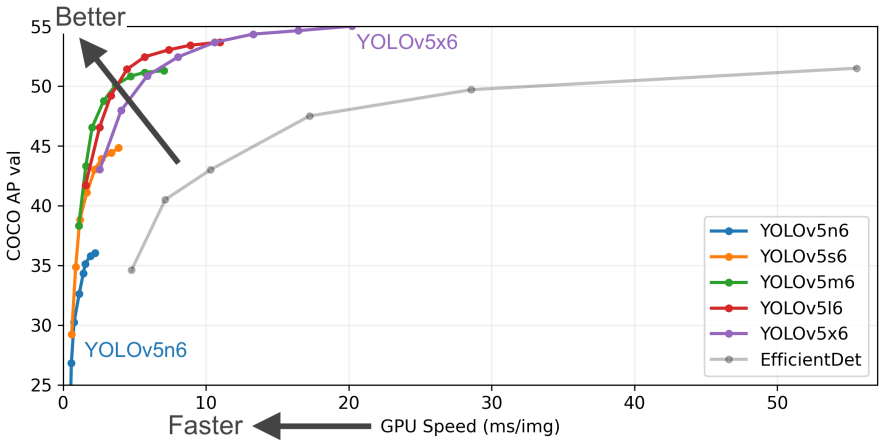


Figura 3.3: Comparación de los rendimientos de las distintas variantes de YOLOv5 para datasets entre 256 y 1536 imágenes del conjunto de imagenes COCO [21]

El entrenamiento duró aproximadamente 45 minutos, y el modelo resultante, aunque muestra resultados prometedores en ciertas áreas a pesar del bajo número de imágenes, como los coches y las personas, tenía taras importantes en otras áreas, como los semáforos, por lo que se determinó que harían falta más entrenamientos. Los resultados completos del entrenamiento se discutirán en el apartado 4.2.

A pesar de los resultados no deseados, se decidió proseguir con el proceso de exportado a la Jetson Nano para poder probar el proceso de conversión del modelo a TensorRT y su ejecución en la Jetson Nano.

Al exportar el modelo a la Jetson Nano y tratar de iniciar el proceso de conversión, se encontró un último problema: Al tratar de generar el archivo .wts, se daba un error debido a que el sistema detectaba que el modelo había sido entrenado en un sistema Windows y, por tanto, a la hora de convertirlo, el sistema trataba de generar rutas con el sistema de archivos de Windows, lo que provocaba errores al estar ejecutandose en un sistema Linux. Tras una búsqueda, se haya una solución en un foro de informática, donde se indicaba que la solución al error era forzar el uso del sistema de archivos Linux, sobrescribiendo las rutas

en la librería pathlib durante el tiempo de ejecución [25]. Con este parche, el archivo .wts se generó correctamente y finalmente se pudo generar el modelo en formato TensorRT para el comienzo de las primeras pruebas en la Jetson Nano. En la figura 3.4 puede verse una imagen procesada por el modelo resultante de este entrenamiento.

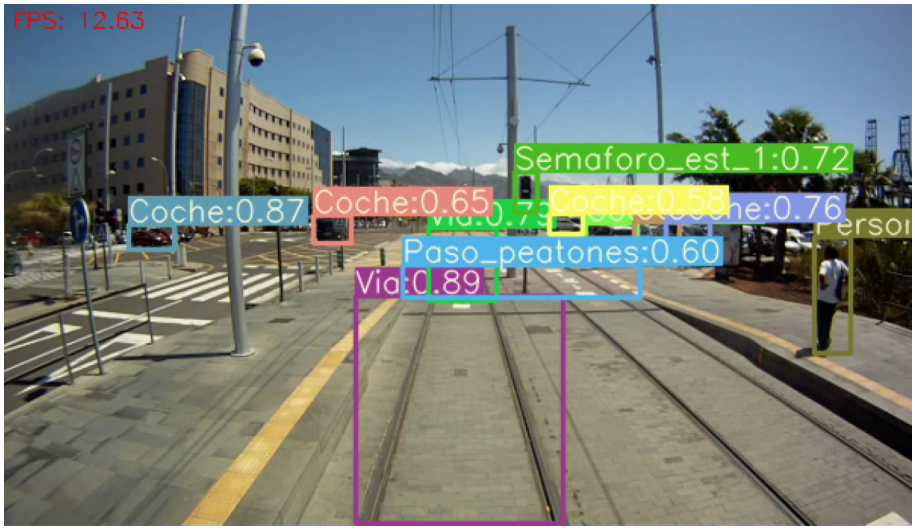


Figura 3.4: Imagen generada con el modelo resultante del primer entrenamiento

3.1.2 Segundo entrenamiento

Tras resolver los problemas surgidos en el primer entrenamiento, el objetivo de este segundo entrenamiento fue tratar de solventar las taras vistas en el modelo del primer entrenamiento, como la incapacidad de detectar otros tranvías o los problemas con los semáforos.

Para ello, se diseñó un nuevo dataset, aumentando el número de imágenes hasta las 1302 imágenes, extrayéndolas esta vez de la totalidad del vídeo suministrado. A la hora de etiquetarlas, se hicieron cambios en la forma de etiquetar las vías, reduciendo el tamaño de los recuadros y aumentando su número. Este se hizo para mejorar el ajuste al perímetro de la vía de las etiquetas, ya que se observó que los grandes cuadros del modelo entrenado en el apartado 3.1.1 podrían

causar muchos falsos positivos en el futuro a la hora de determinar si un objeto ha invadido la vía. También se retocaron levemente las etiquetas, quedando de la siguiente manera:

- Semaforo_est_1.
- Semaforo_est_2
- Semaforo_est_3.
- Persona.
- Vehículo (Cualquier vehículo que no sea el tranvía).
- Paso_peatones.
- Via.
- Tranvia.
- Limite (Cualquier señal de límite de velocidad).
- Signal (Cualquier señal que no sea un límite de velocidad).

Finalmente, tras etiquetar todas las imágenes y pasarlas por el cuaderno de jupyter diseñado en el entrenamiento del apartado 3.1.1, se obtuvo un dataset con 911 imágenes de entrenamiento y 391 imágenes de validación. En la figura 3.5 se pueden observar un ejemplo de la comparación de una imagen con su versión etiquetada.

(a) Imagen sin etiquetar *(b) Imagen etiquetada*

Figura 3.5: Comparativa de imágenes antes y después del etiquetado para segundo entrenamiento

Con el nuevo dataset, se repitió el entrenamiento con los mismos parámetros que en el aparatado 3.1.1, 100 épocas con batch_size de 16. El entrenamiento se prolongaría a lo largo de 12,30 horas aproximadamente, tras las cuales, al analizar los resultados, se observaron mejoras en ciertas clases, como la de los tranvías, además de obtenerse mejores ajustes al contorno de la vía, fruto del nuevo etiquetado. Sin embargo, se seguían teniendo graves deficiencias en la detección de los semáforos, solo siendo capaz de detectar prácticamente solo uno de los estados. Los resultados completos pueden verse en el apartado 4.3.

A pesar de los malos resultados, se decidió procesar el modelo en la Jetson Nano para, usarlo como base para empezar a desarrollar el programa de procesamiento de la información de la IA que veremos en el apartado 3.2, aprovechando el ajuste obtenido en la detección de las vías, mientras se realizaban nuevas pruebas de entrenamientos.

En la figura 3.6 puede verse una imagen procesada por el modelo resultante de este entrenamiento.

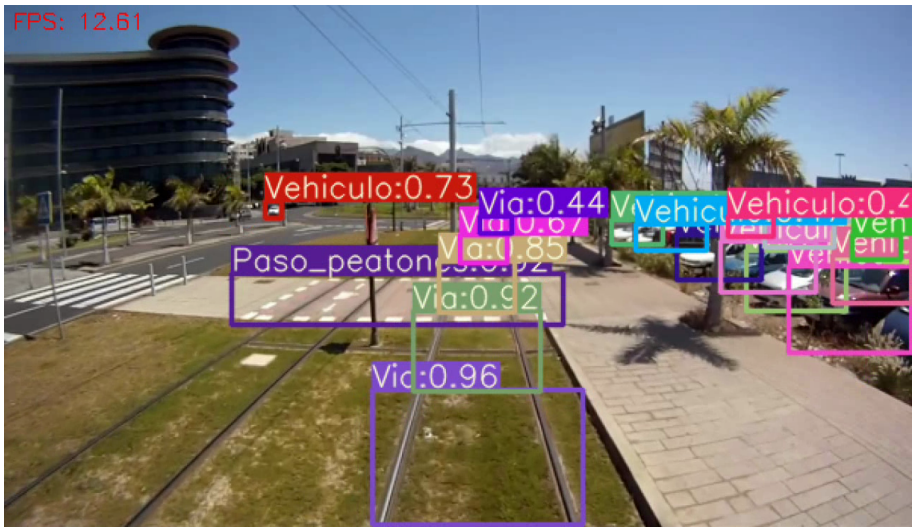


Figura 3.6: Imagen generada con el modelo resultante del segundo entrenamiento

3.1.3 Tercer entrenamiento

Tras los resultados observados en el entrenamiento del apartado 3.1.2, la primera opción que se consideró para solucionar el problema fue el uso de una versión del modelo YOLOv5 más compleja, el modelo medium. Esta opción se consideró en base a lo observado en la figura 3.3, donde para un mismo número de imágenes, un modelo de más complejo obtenía mejores resultados, por lo que se decidió probar, ya que esta opción permitía aprovechar el dataset preparado para el entrenamiento del apartado 3.1.2, ahorrando así tiempo de desarrollo.

En base a lo observado en los resultados del anterior entrenamiento (Ver apartado 4.3 para más información) se decidió reducir el número de épocas del entrenamiento a 50, manteniendo el mismo batch_size. Sin embargo, a pesar de la reducción de épocas, el entrenamiento volvió a duran en torno a las 12,30 horas aproximadamente, dado el mayor tamaño del modelo.

Tras finalizar el entrenamiento, se observaron los resultados, pero decepcionantemente, no se pudo observar ninguna mejora notable en el uso de un modelo más pesado, lo que parecería indicar que el problema se encuentra en las imáge-

nes del dataset, por lo que se descartó el uso del modelo medium y, en su lugar, se regresó al modelo slim y se decidió centrarse en mejorar el dataset utilizado.

Los resultados completos de este entrenamiento pueden consultarse en el apartado 4.4.

3.1.4 Cuarto entrenamiento

Tras los resultados del apartado 3.1.3, se decidió estudiar posibles deficiencias que pudiera tener el dataset utilizada en los dos últimos entrenamientos, en especial para mejorar la detección de los semáforos, al ser un elemento clave de nuestro sistema. Se consideraron dos opciones:

A. Que, durante el proceso de separación aleatorio de imágenes para entrenamiento y validación, la mayoría de las imágenes de semáforos se quedasen en validación, impidiendo que el modelo sea entrenado con pocas imágenes de semáforos.

B. Que el número de imágenes de semáforos sea insuficiente.

Para este cuarto entrenamiento, se ha considerado la opción A, por lo que los esfuerzos se centraron en garantizar la variedad de imágenes en el dataset de entrenamiento.

Para lograr este objetivo, se modificó el cuaderno de jupyter del apartado 3.1.1 para que leyera los archivos de etiquetas de cada imagen y le asignara a cada una en el dataframe una etiqueta de 0 a 9, en función de la clase más relevante presente en la imagen, según el siguiente criterio de prioridad:

1. Semaforo_est_1.

2. Semaforo_est_2

3. Semaforo_est_3.
4. Paso_peatones.
5. Tranvia.
6. Signal.
7. Limite.
8. Persona.
9. Vehiculo.
10. Via.

El orden de prioridad se ha asignado en función de lo relevante que es esa clase y la poca representación que hay de ella en las imágenes, motivo por el que, por ejemplo, la clase Persona está tan abajo, debido a la gran representación que ya tiene en general.

Con esta etiquetado, se hace uso del argumento stratify de la función train_test _split para garantizar el la representación de todas las clases en ambos sets de entrenamiento y validación en la proporción deseada. El código de este nuevo cuaderno de jupyter, llamado Separar_conjuntos_entrenamiento_con_clases_mas _importantes.ipynb, puede encontrarse en el apéndice A.2.

Tras la formación del dataset, se realizó el entrenamiento con 50 épocas y batch_size de 16, completándose el entrenamiento en 6 horas, aproximadamente.

Tras analizarse los resultados, se logró observar una pequeña mejora en los resultados de la detección de los semáforos con el nuevo reparto de imágenes, pero aún insuficiente como para ser aceptable.Los resultados completos de este entrenamiento pueden consultarse en el apartado 4.5.

A raíz de esto, se concluye que el origen del problema es la falta de imágenes donde se representen los diferentes estados del semáforo, lo que se intentará solucionar en el siguiente entrenamiento.

3.1.5 Quinto entrenamiento

Tras observarse los resultados de todos los entrenamientos anteriores, se concluye que el motivo de la falta de precisión del modelo se debe a la falta de imágenes en ciertas clases, entre ellas las de los semáforos. Sin la opción de obtener nuevas imágenes de los semáforos y etiquetarlas en un tiempo razonable, se decidió optar por duplicar las imágenes ya existentes, de manera que el modelo se le muestren más imágenes con semáforos durante el entrenamiento.

Para ello, se ha creado un sencillo script en python, llamado Duplicar _borrar_imagenes.py, el cual permite visualizar individualmente cada imagen del conjunto de imágenes, con las etiquetas que contiene, y duplicar tanto la imagen como el archivo de etiquetas si se se observa que alguna de las imágenes tiene elementos interesantes, y de la misma manera, borrar las que no sean interesantes. El código completo del script puede observarse en el apéndice A.3.

Haciendo uso del script, se aumento el número de imágenes en las que aparecían semáforos, provocando que el número de imágenes disponibles aumentara hasta las 2390. Tras pasar el cuaderno de jupyter desarrollado en el apartado 3.1.4, se obtuvo un dataset de 1673 imágenes de entrenamiento y 717 imágenes de validación.

Con este dataset, se entrenó el modelo durante 50 épocas, con batch_size de 16, con una duración de 12.40 horas, aproximadamente, y se analizaron los resultados. Estos resultados, que pueden consultarse de manera completa en

el apartado 4.6, esta vez sí indicaron una fuerte mejoría en la detección de los estados de los semáforos, al tiempo que mantenía los resultados obtenidos en entrenamientos posteriores para el resto de clases. Aún se observan momentos en los que el modelo presenta dificultades para determinar el estado, como la diferenciación entre estado 1 y 2, pero los resultados se consideraron lo suficientemente óptimos como para elaborar el sistema de detección con este modelo, por lo que se definió como modelo final del sistema.

3.2 Desarrollo del programa de procesamiento de la información

En el apartado anterior, se explicó el proceso de desarrollo del modelo de IA que se utilizaría para la detección de objetos. Tras lograr unos modelos aceptables, se comenzó a desarrollar, de manera paralela, un programa encargado de cargar y ejecutar el modelo de IA entrenado, y a su vez procesar la información devuelta por la IA para determinar la presencia de objetos en la vía o el estado de los semáforos.

3.2.1 Primera versión

La primera versión que se desarrolló fue detectar_objetos.py. En esta primera versión, el objetivo era crear una base funcional para leer la fuente de vídeo suministrada, extraer imágenes de estas ejecutar inferencias con el modelo sobre la imagen, mostrar los resultados por pantalla y volcar los resultados en un vídeo.

Para lograr este objetivo, se usó opencv para leer la fuente de vídeo suministrada y generar un archivo .mp4 de salida, donde se volcaría los resultados. Tras esto, en cada ciclo del programa, se captura un frame de la fuente y, tras redimensionarlo y procesarlo por el modelo, quien devuelve el frame con cada

elemento detectado recuadrado y una etiqueta con la clase y confianza de detección, es vuelto a redimiensionar al formato del vídeo de salida y escrito en el vídeo. Adicionalmente, se añadió la posibilidad de detener el programa con la letra 'q' y que el programa se cerrara automáticamente si se detectaba que algún error durante la lectura de la fuente vídeo, además de escribir en la imagen el número de frames por segundo procesados por la IA.

El principal inconveniente se encontró cuando se trató de hacer que el texto de la ventana de visualización del vídeo fuera dinámico, para mostrar datos en ella. Las primeras pruebas resultaron en la generación de infinitas ventanas, llegando incluso a bloquear la propia Jeson Nano. Sin embargo, este error se pudo solucionar rápidamente forzando a que el nombre de la ventana generada sea único, haciendo que siempre se sustituya su contenido en lugar de abrir una nueva ventana cuando se muestra una nueva imagen, y cambiar el texto tras esto.

Las figuras 3.4 y 3.6 provienen de vídeos generados con detectar _objeto.py. El código completo puede encontrarse en el apéndice A.4.

3.2.2 Segunda versión

Para esta segunda versión, llamada detectar_objetos_con_alertas.py, el objetivo era hacer uso de la información devuelta por la IA sobre los objetos detectados en la imagen, para así poder determinar si un objeto está en la via o el estado de los semáforos y generar las advertencia y alertas correspondientes.

Para ello, partiendo como base el programa desarrollado en el apartado anterior, se expandía el programa para que recorriera el diccionario devuelto por el modelo, activando el aviso correspondiente en función del estado del semáforo (si lo hubiera) y la presencia de un paso de peatones, y almacenando las coordenadas

de los recuadros correspondientes a las vías, vehículos y personas detectadas. Con estas últimas, se comprueba para cada recuadro de vía si alguno de los recuadros de vehículo o persona se le superpone, y en caso afirmativo, activar un aviso u otro en función de si el trozo de vía se encuentra o no por encima de mitad de la imagen, límite a partir del que se considera que algo es lejano en la vía.

De esta manera, la lista de avisos generados queda como puede observarse en la tabla 3.1, mostrandose por pantalla en la imagen:

Evento	**Aviso asociado**
Persona	Deténgase
Persona (lejana)	Precaución
Vehículo	Deténgase
Vehículo (lejano)	Precaución
Semáforo estado 1	Deténgase
Semáforo estado 2	Precaución
Semáforo estado 3	Continúe
Paso de peatones	Precaución

Tabla 3.1: Lista de alertas asociadas a cada clase

Durante las pruebas, se observó que se daban bastantes falsos positivos en las áreas donde los coches circulan al lado del tranvía. Esto era debido a que los recuadros de los vehículos que circulaban al lado del tranvía tendían a rozar levemente el área de los recuadros de la vía. Para solucionarlo, se añadió un offset a las comprobaciones de solapamiento por el lateral, reduciendo así los falsos positivos.

En la figura 3.7 pertenecen a un vídeo procesado por este programa. El código completo del programa se encuentra en el apéndice A.5.

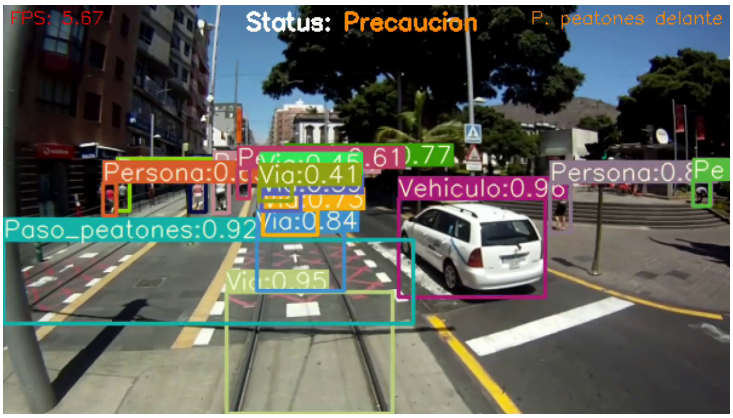


Figura 3.7: Frame procesado por el programa de detección de objetos con alertas

3.2.3 Tercera versión

En esta tercera versión, llamada detectar_objetos_con_alertas_visual_usuario .py, el objetivo era, habiendo ya implementado una sistema funcional de detección de obstáculos en el apartado anterior, crear una variante que de una imagen más limpia que la de versiones anteriores. Esto se hace dado que, si observamos la figura 3.7, podremos ver que el sistema arroja mucha información por pantalla de todos los objetos detectados, algo útil para analizar el sistema y hacernos una idea de todo lo que el sistema "ve" y con qué confianza, pero, sin embargo, de cara a su uso por un conductor del tranvía, sería mucha información que no le aporta nada.

Para solucionarlo, partiendo del código del programa del apartado 3.2.2, se implementó que se creara una copia del frame antes de procesarlo por la IA, lo que permite tener un frame limpio para escribir en él solo la información relevante. Tras esto, y durante las fases de comprobaciones del contenido de la imagen y la activación de alertas, se incluyó que en el frame limpio, aparte de la escritura de las alertas, también se dibujen exclusivamente los recuadros de los elementos que hayan hecho saltar una alerta, de tal manera que el conductor sepa exactamente sobre cuales elementos debe permanecer atento.

Además, en esta versión se ha mejorado el control del tiempo de ejecución del programa, pasando a medirse ahora la duración completa del ciclo del ciclo del programa y no solo del tiempo de ejecución de la inferencia de la IA, como en anteriores versiones. Este cambio se aplicó de forma retroactiva al programa detectar_objetos_con_alertas.py del apartado 3.2.2, al considerarse una mejora útil para ambos programas. El código final completo de esta versión con visual de usuario puede verse en el apéndice A.6.

En la figura 3.8 permite visualizar un frame de un video procesado por esta versión del programa para que pueda compararse con la figura 3.7, perteneciente a la otra versión.

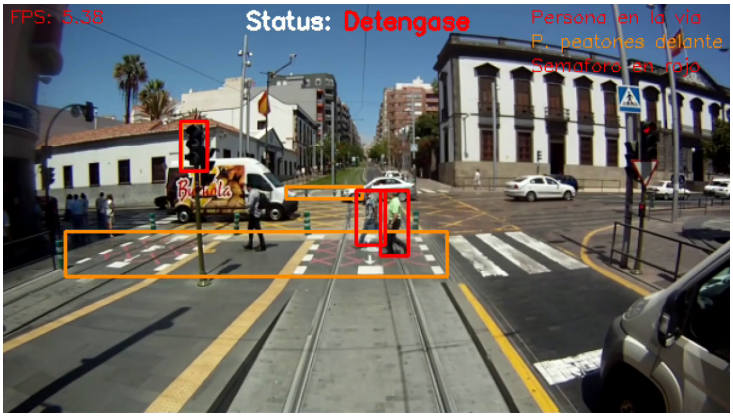


Figura 3.8: Frame procesado por el programa de detección de objetos con alertas en su versión para usuarios

Esta versión del programa, junto a la versión del apartado 3.2.2, se han considerado la versiones finales de nuestro sistema.

Capítulo 4

Resultados

En el capítulo 3, se explicó el desarrollo del sistema y los diferentes entrenamientos que se hicieron, dando un breve resumen de los resultados obtenidos en cada uno para justificar los cambios introducidos en cada entrenamiento.

En este capítulo, expondremos de forma completa los resultados de estos entrenamientos y profundizaremos más en los motivos que llevaron a la toma de las decisiones y cambios expuestos a lo largo del capítulo 3. Además, se presentan los resultados del entrenamiento del modelo final.

4.1 Consideraciones previas

Para el estudio de los resultados de los entrenamientos, aparte de las métricas presentadas en el apartado 1.2.3, que nos sirven para medir el rendimiento del modelo ya entrenado, debemos tener en cuenta otras métricas, recolectadas por el programa de entrenamiento durante el entrenamiento, y que nos permiten observar la evolución del entrenamiento durante las diferentes épocas. Estas métricas son [18]:

- Perdida de localización (box_loss): Métrica usada para medir el error cometido a la hora de detectar la posición de un objeto en la imagen.

- Perdida de objetividad (obj_loss): Métrica usada para medir el error cometido a la hora de determinar si hay presencia de algún objeto de las clases definidas en la imagen.

- Perdida de clases (class_loss): Métrica usada para medir el error cometido a la hora de asignar una clase u otra a un objeto detectado.

Para que el entrenamiento sea correcto, nos interesa que los valores de estas métricas disminuyan en cada época, terminando lo más pequeños posibles, ya que implicaría que el error que comete el modelo en el campo que representa cada métrica disminuye.

Aparte de estas métricas, que se calculan durante el transcurso de cada época, algunas de las métricas del apartado 1.2.3, como la precisión, el recall y los mAP, se calculan al final de cada época, para así poder medir su evolución por época.

Finalmente, como último apunte, resaltar que los resultados de las matrices de confusión presentadas de cada modelo se encuentran normalizados en función del número total de veces que el modelo ha asignado esa clase a una detección.

4.2 Resultados del primer entrenamiento

Tal y como se mencionó en el apartado 3.1.1, el objetivo de este entrenamiento fue la prueba del proceso de entrenamiento y generación de modelos entrenados, no tanto generar un modelo funcional. Sin embargo, el estudio de los resultados de este entrenamiento arrojaría algunos datos interesantes de cara a posteriores análisis, tanto en entendimiento de los datos devueltos por el script de entrenamiento como en mejoras en la elaboración de los datasets.

Primero observaremos en la figura 4.1 los datos de la evolución de las métricas

durante el entrenamiento, donde en el eje x tenemos el número de la época, y en el eje y tenemos los valores de las métricas para cada época:

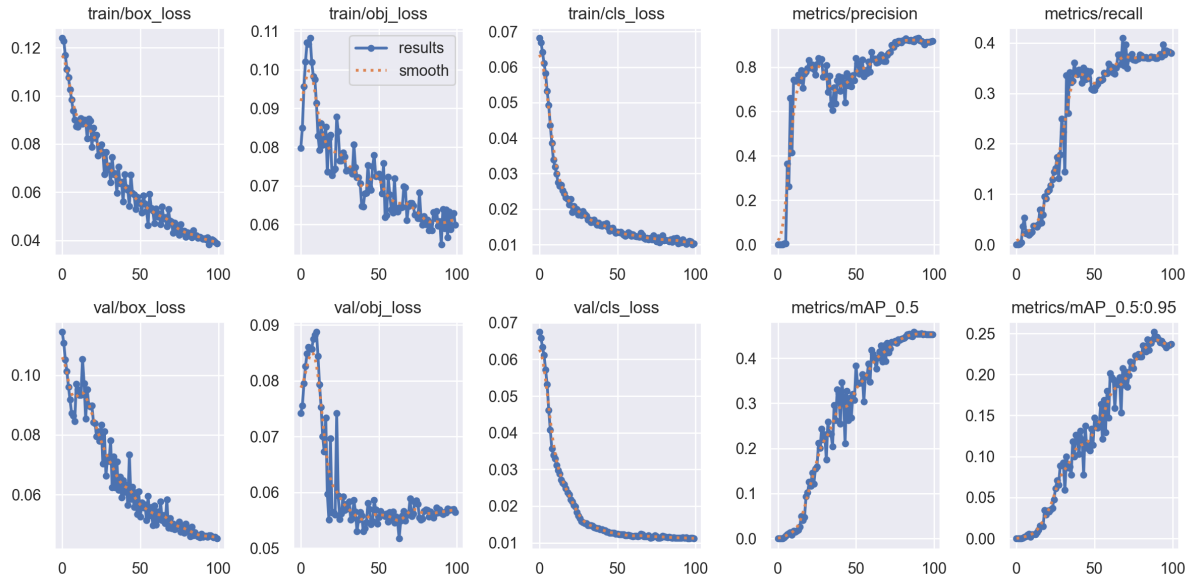


Figura 4.1: Evolución de las métricas durante el primer entrenamiento

A la hora de determinar leer los datos, los más relevantes en estas gráficas de la figura 4.1 son los que refieren a los medidos con el set de imágenes de validación (marcado como val), ya que representan el comportamiento del sistema con imágenes que el modelo no ha visto durante el entrenamiento y, por tanto, son nuevas para él, además de las métricas de rendimiento general del modelo al final de cada época (Precisión, recall y sendos mAP), ya explicadas en el apartado 1.2.3.

Atendiendo a esto, en términos generales, observamos bajos resultados en el mAP, que indican un rendimiento pobre del modelo. Lo más destacable es el hecho de que la precisión es alta, pero el recall es bajo. Esto nos indica que el modelo tiene problemas para localizar los objetos, probablemente más algunos que otros.

Esta teoría se confirma al observar la matriz de confusión en la figura 4.2,

cuya interpretación se explicó en el apartado 1.2.3:

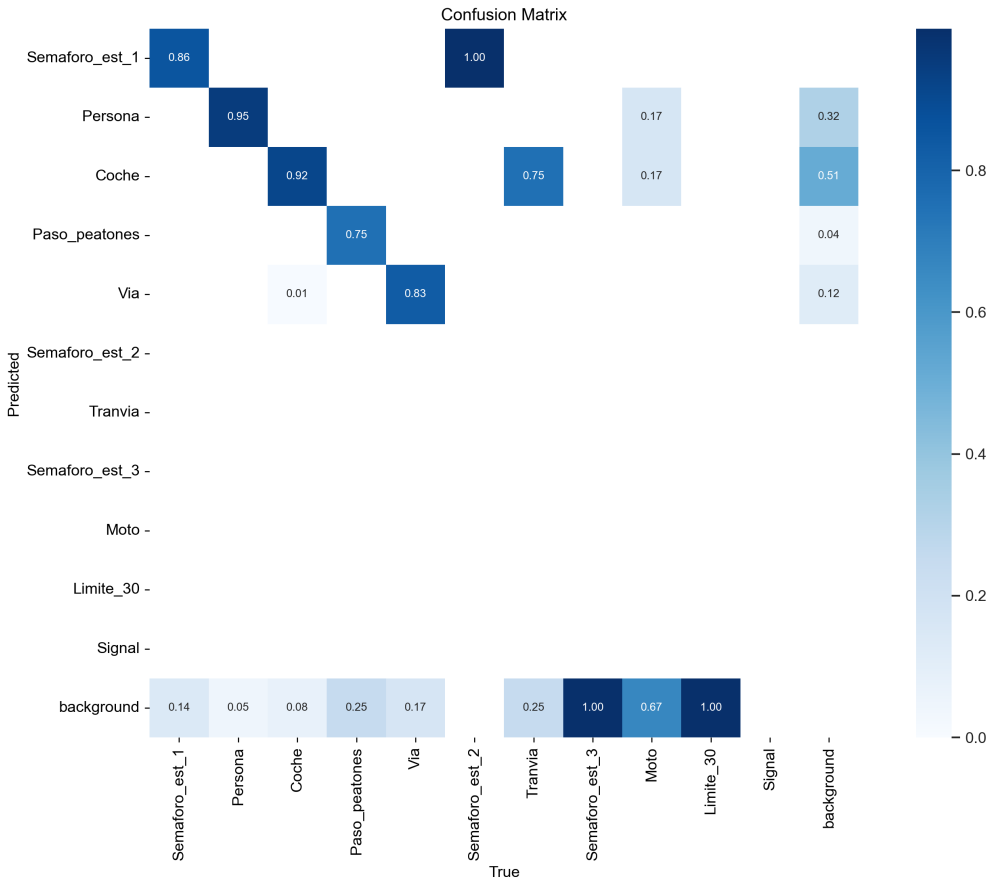
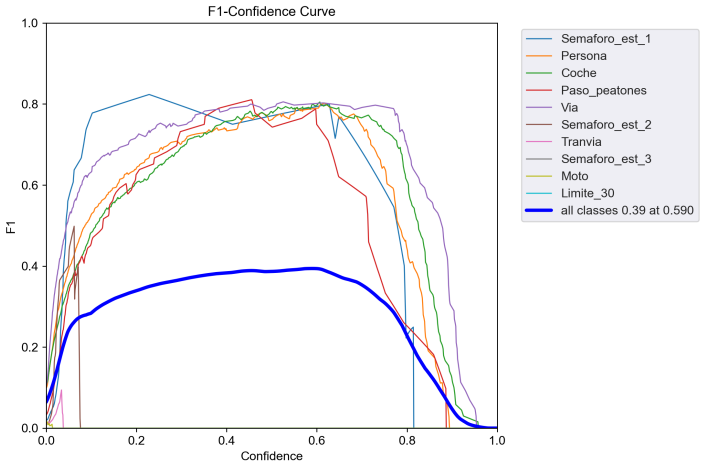


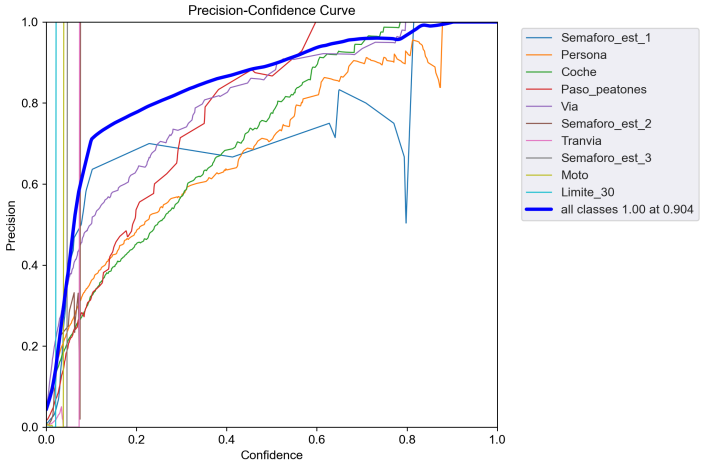
Figura 4.2: Matriz de confusión del primer modelo

Podemos observar que, salvo en las clases Semaforo_est_1, Persona, Coche, Paso_peatones y Via, el modelo es incapaz de detectar ninguna de las demás clases, aunque las clases que sí detecta lo hace con bastante solvencia, lo que explica la precisión global alta y el recall global bajo.

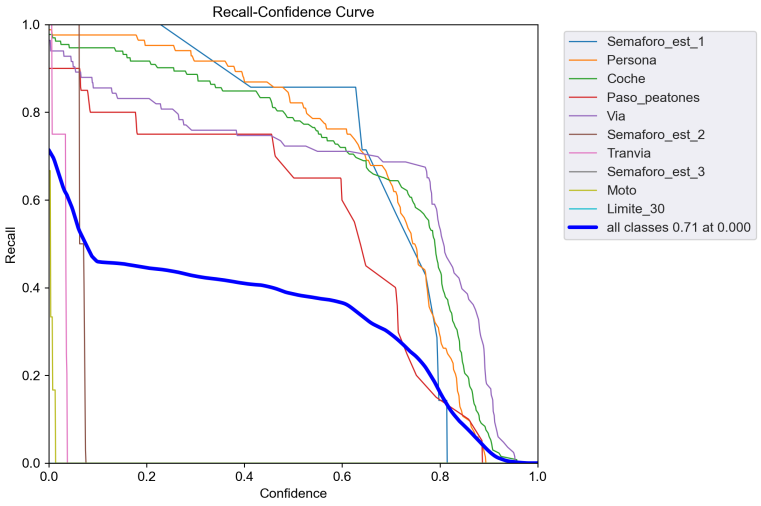
Este hecho se vuelve más notable si observamos las curvas de precisión, recall y F1 para cada clase en función de la confianza de detección, mostrada en la figura 4.3, donde podemos ver los resultados de cada una de estas métricas para cada nivel de confianza o seguridad del modelo en que un objeto detectado sea lo predicho (es decir, si damos por bueno cualquier detección por encima de ese nivel de confianza, que valor obtendríamos en cada métrica):



(a) F1-score



(b) Precisión



(c) Recall

Figura 4.3: Curvas de F1-score, Precisión y recall en función de la confianza del primer entrenamiento

Se puede ver como, para las clases antes mencionadas (Semaforo_est_1, Persona, Coche, Paso_peatones y Via), las curvas tienen valores más o menos lógicos, mientras que para el resto, los valores enseguida caen a cero, incluso a confianzas bajas.

Todos estos resultados coinciden en una falta de instancias de estas clases, algo comprensible teniendo en cuenta el pequeño tamaño del dataset, por lo que se decidió ampliarlo para los siguientes entrenamientos.

4.3 Resultados del segundo entrenamiento

A diferencia del primer entrenamiento visto en el apartado 4.2, donde se buscaba probar el modelo, este entrenamiento fue el primero en tratar de obtener un modelo más definitivo, remarcado en el mucho mayor tamaño del dataset utilizado.

Sin embargo, tal y como adelantamos en el apartado 3.1.2, los resultados obtenidos no fueron los esperados, tal y como puede verse en la evolución de las métricas en la figura 4.4:

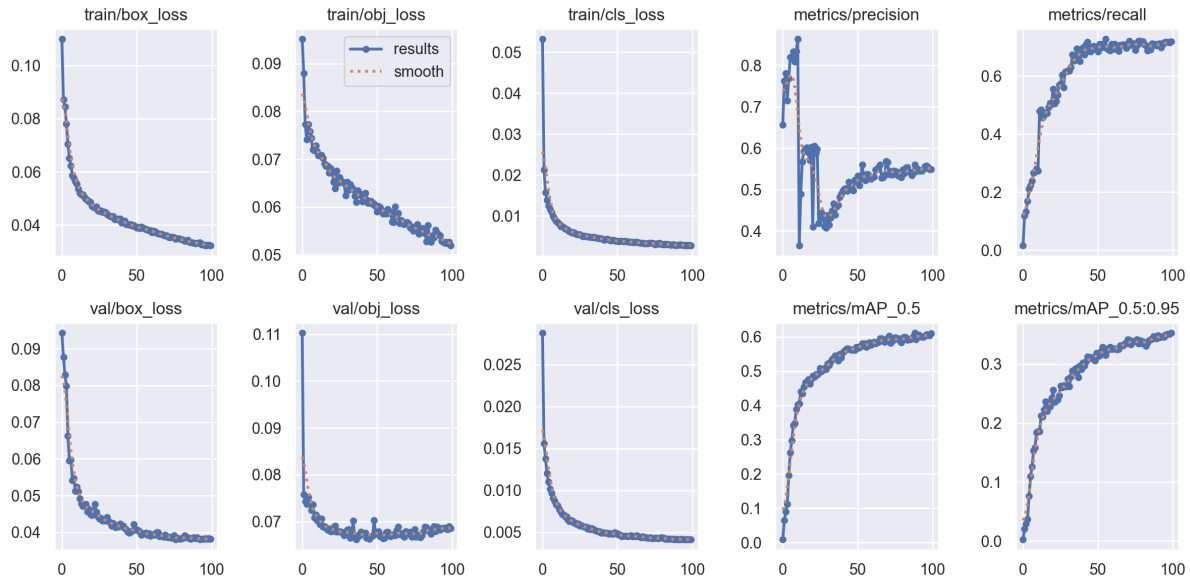


Figura 4.4: Evolución de las métricas durante el segundo entrenamiento

Como puede verse, a pesar de haber una mejora en el box_loss, el class_loss y el recall, ha habido perdidas en el obj_loss y sobre todo en la precisión, lo que ha conllevado que, en la práctica, métricas que miden el rendimiento global del modelo, como los mAP no hayan mejorado sustancialmente (aunque sí que ha habido una mejora de dos puntos en mAP0.5).

Estos resultados se tradujeron en que el modelo resultante de este entrenamiento fuera capaz de detectar más objetos, pero teniendo más problemas para clasificarlos, como puede verse en la matriz de confusión de la figura 4.5:

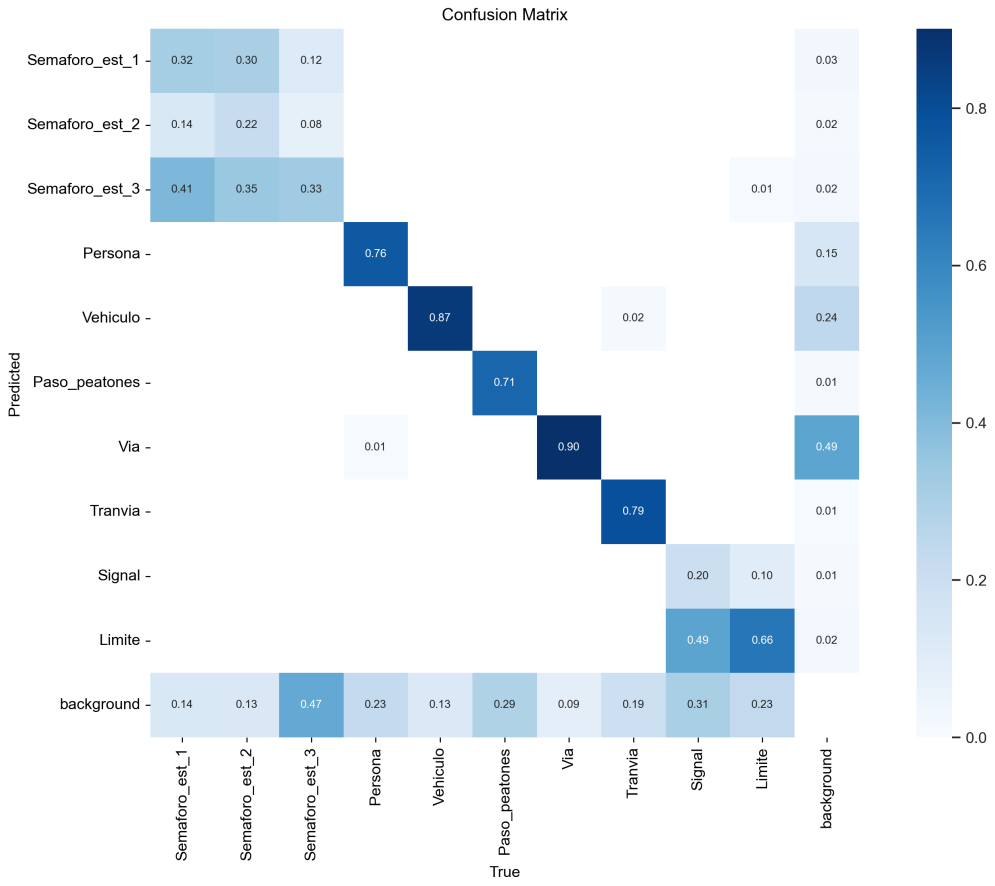
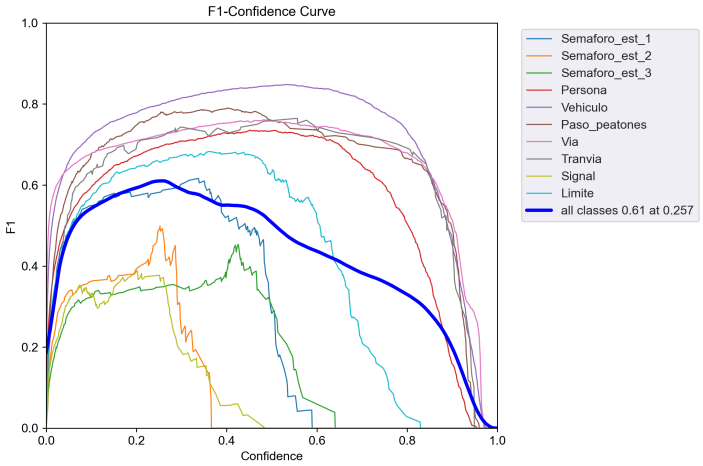
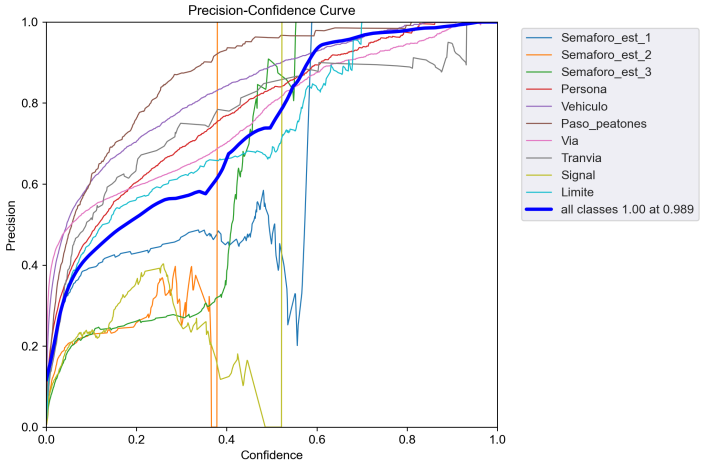


Figura 4.5: Matriz de confusión del segundo modelo

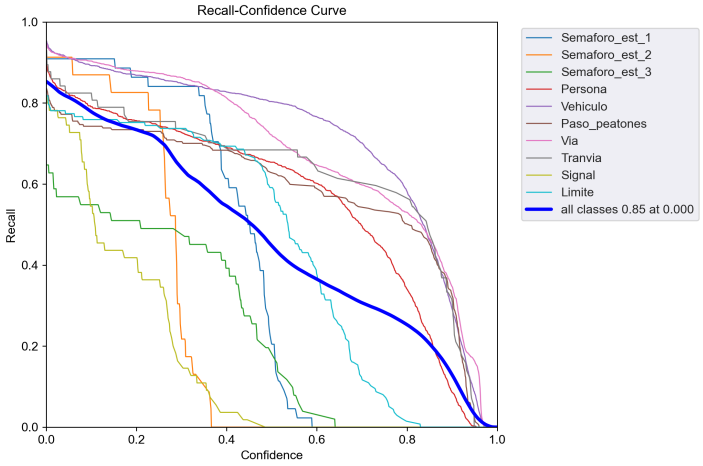
Observamos que ahora aunque, con problemas, el modelo es capaz de detectar todas las clases. Sin embargo, al observar las curvas individuales de F1, precisión y recall de la figura 4.6, observamos otro problema.



(a) F1-score



(b) Precisión



(c) Recall

Figura 4.6: Curvas de F1-score, precisión y recall en función de la confianza del segundo entrenamiento

En estas gráficas podemos observar que, en especial en el caso los semáforos, los valores de precisión y recall caen rápidamente para confianzas superiores al 0.5, lo que significa que el modelo nunca obtiene una detección que supere 0.6 en el mejor de los casos. Esto provocará que el límite de confianza de lo que se considera una detección, situado en 0.4, descarte la mayoría de las detecciones realizadas de estas clases. Es por ello que en los siguientes entrenamientos se probaron diferentes soluciones para mejorar esa confianza.

Como información más relevante de este entrenamiento, se comprobó gracias, a los datos de la figura 4.4, que un entrenamiento de 100 épocas es demasiado, ya que a partir de las 50 ya se observa un estancamiento en la progresión de las métricas con el set de validación. Por ello, se estableció que los entrenamientos fueran de 50 épocas en siguientes entrenamientos, como vimos en los apartados 3.1.3, 3.1.4 y 3.1.5

4.4 Resultados del tercer entrenamiento

Tal y como se mencionó en el apartado 3.1.3, en este entrenamiento se probó a usar la versión medium del modelo YOLOv5 para paliar los problemas observados en el segundo entrenamiento.

Sin embargo, al observar la evolución de las métricas durante el entrenamiento, como puede verse en la figura 4.7, los resultados no fueron los esperados, nuevamente.

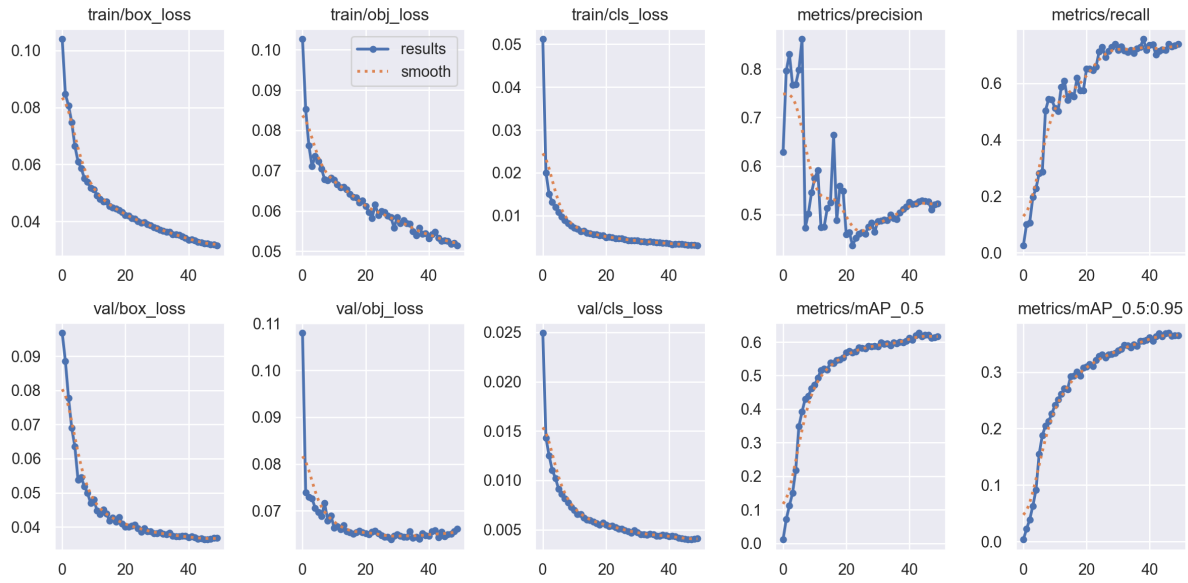
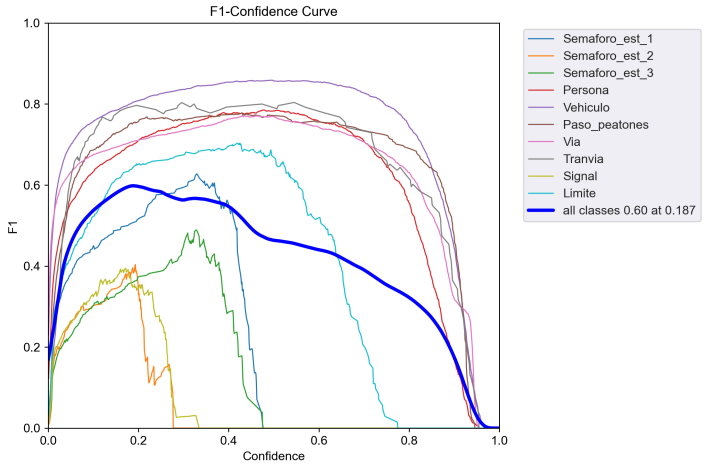


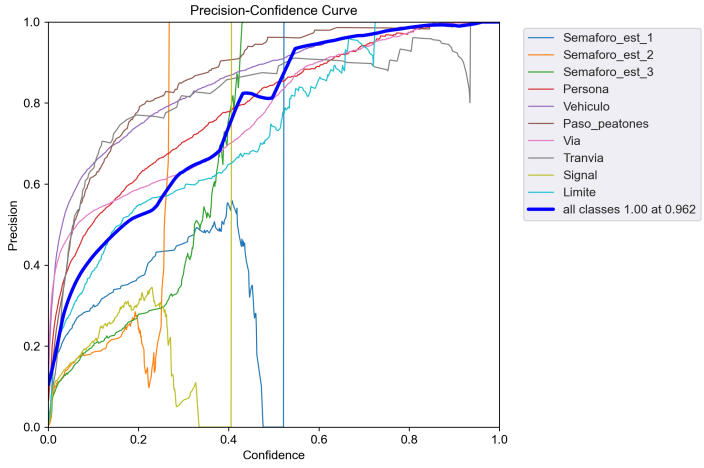
Figura 4.7: Evolución de las métricas durante el tercer entrenamiento

Como puede verse, al compararse estos resultados con los del segundo entrenamiento en la figura 4.4, que los resultados obtenidos de usar el modelo medium son similares a los obtenidos del modelo slim del segundo entrenamiento, no habiendo ninguna mejora en el proceso.

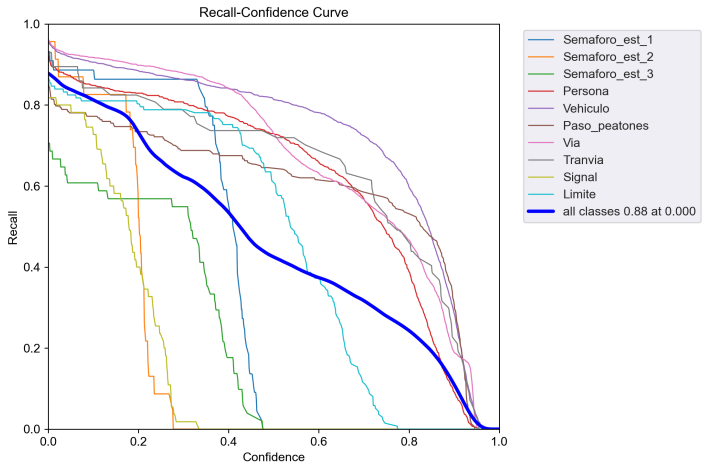
El problema es aún más evidente si observamos las curvas individuales de F1, precisión y recall de la figura 4.8.



(a) F1-score



(b) Precisión



(c) Recall

Figura 4.8: Curvas de F1-score, Precisión y recall en función de la confianza del tercer entrenamiento

A raíz de estos resultados es que se tomó la decisión de descartar este modelo entrenado y probar otras alternativas.

4.5 Resultados del cuarto entrenamiento

En este entrenamiento, como se vió en el apartado 3.1.4, se probó a entrenar el modelo con un dataset que garantizaba la variedad de presencia de instancias de cada clase, en un intento de mejorar los resultados de salida.

La la evolución de las métricas durante este entrenamiento pueden verse a continuación en la figura 4.9.

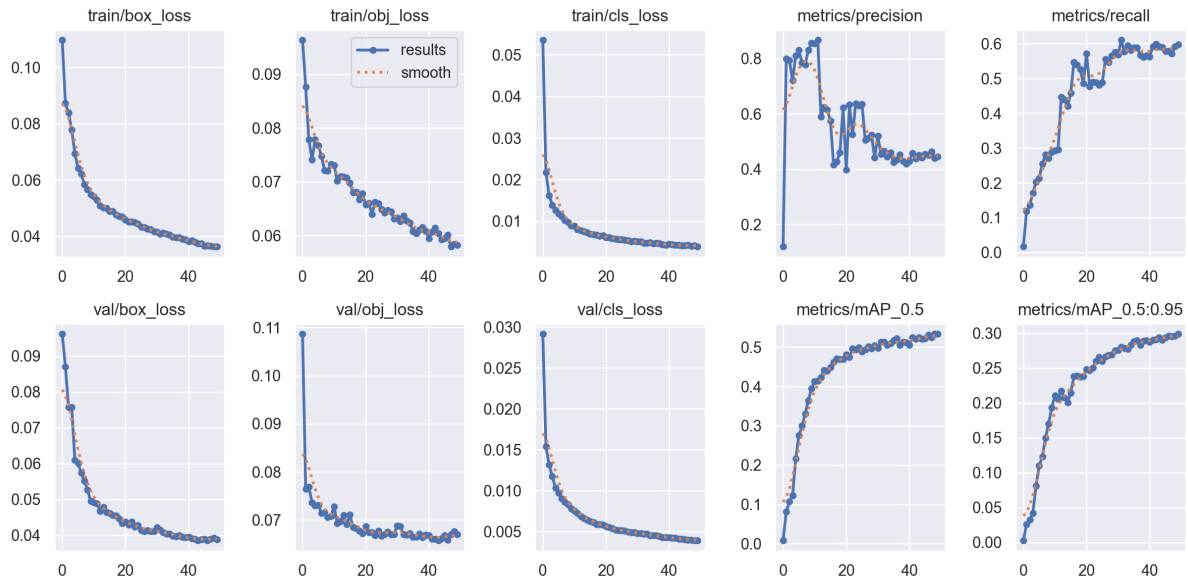


Figura 4.9: Evolución de las métricas durante el cuarto entrenamiento

Como se observa, nuevamente no se han obtenido mejoras en los resultados, llegando a verse incluso empeoramientos en áreas como el recall o la precisión, notándose dicha disminución en el mAP.

Este empeoramiento también puede vislumbrarse en la matriz de confusión del modelo, mostrada en la figura 4.10, donde pueden observarse pérdidas completas en la capacidad de detección de algunas clases, como la Semaforo_est_2.

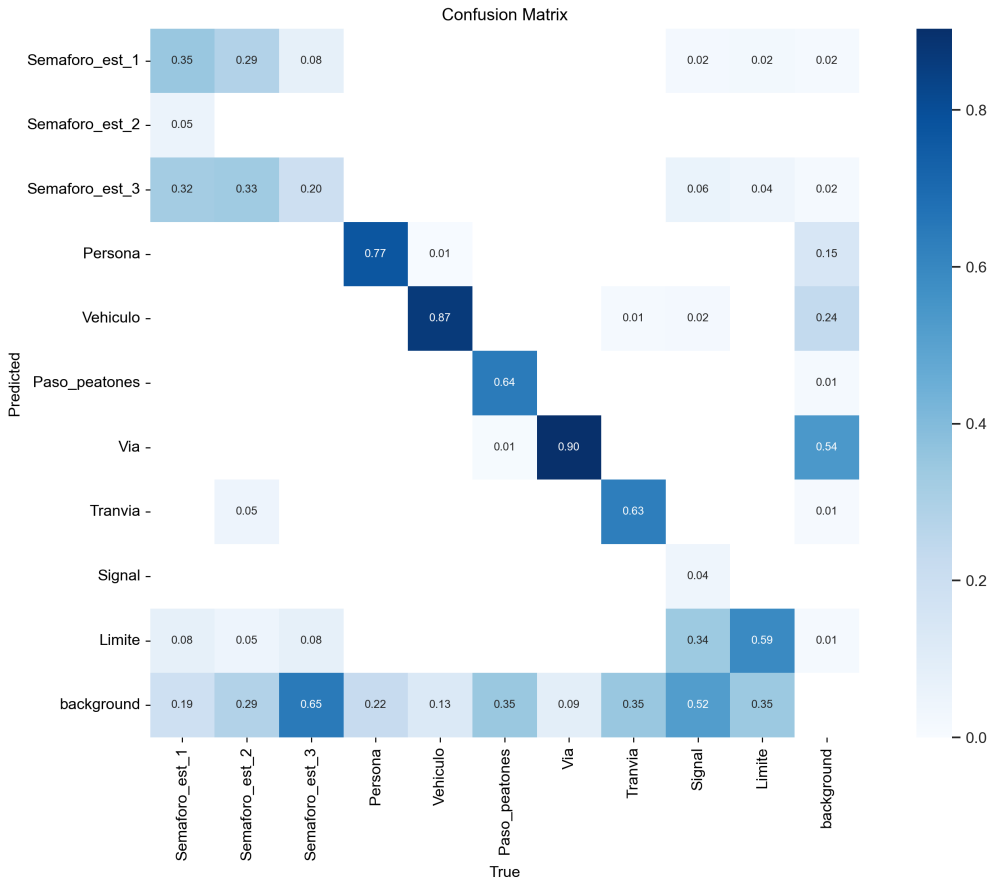
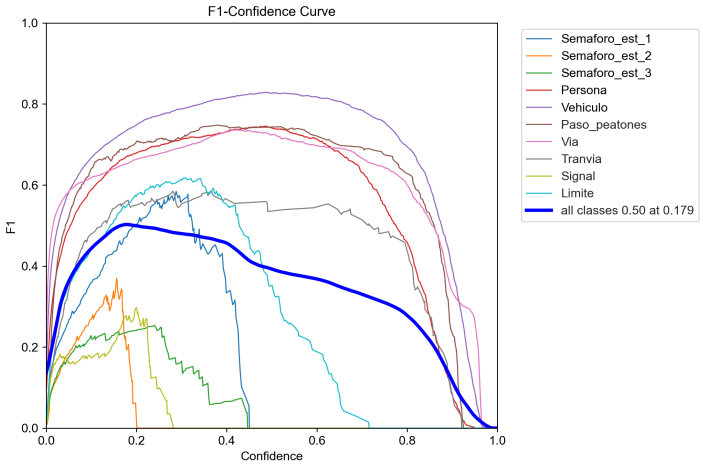
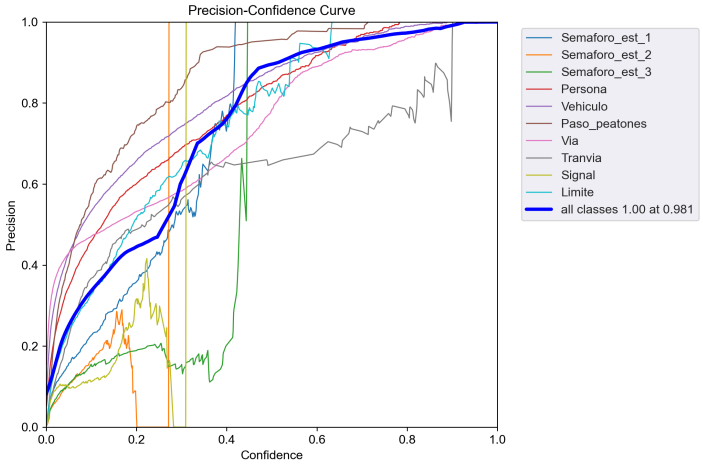


Figura 4.10: Matriz de confusión del cuarto modelo

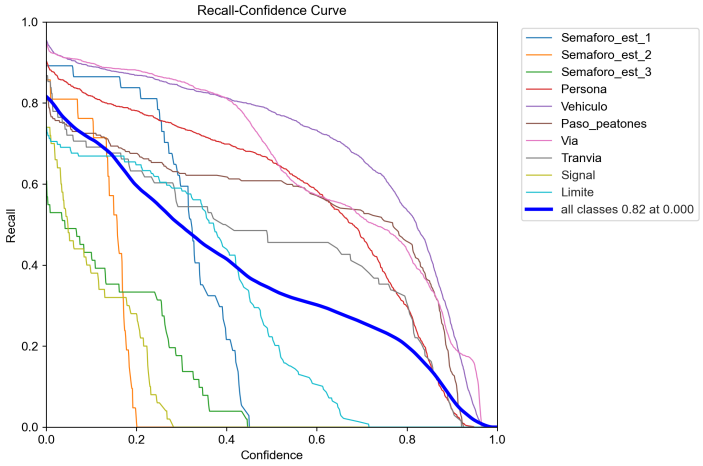
Finalmente, observando las curvas individuales de F1, precisión y recall para cada clase en la figura 4.11, podemos comprobar que no se ha producido ninguna mejora en ninguno de los parámetros, para ninguna de las clases.



(a) F1-score



(b) Precisión



(c) Recall

Figura 4.11: Curvas de F1-score, Precisión y recall en función de la confianza del cuarto entrenamiento

Estos resultados, tal y como se concluyó en el apartado 3.1.4, lo que muestran es que necesitamos entrenar el modelo con más imágenes, tal y como finalmente se hizo en el apartado 3.1.5

4.6 Resultados del entrenamiento final

Tal y como vimos en el apartado 3.1.5, y a raíz de los resultados expuestos en los apartados anteriores de este capítulo, se determinó que el origen de las deficiencias de los modelos era debido a la falta de representación de algunas clases en el dataset, por lo que se realizó este último entrenamiento aumentando el número de imágenes, esperando lograr una mejora en los resultados.

Tras observar los resultados de la evolución de las métricas durante este entrenamiento, que pueden verse en la figura 4.12, se pudo finalmente ver mejoras con respecto a los entrenamientos anteriores.

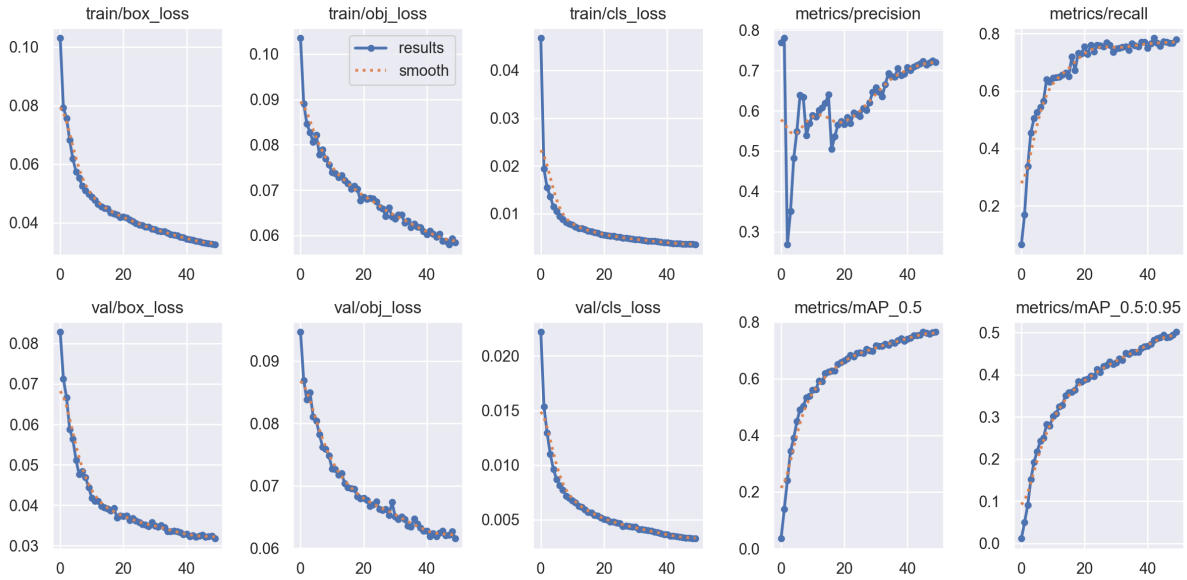


Figura 4.12: Evolución de las métricas durante el entrenamiento final

Como puede verse, en este entrenamiento se ha logrado una reducción de las pérdidas durante el entrenamiento, lo que ha repercutido en una mejora de casi 2

puntos porcentuales en los mAP. Además, se ha logrado aumentar la precisión general del modelo, llegando a alcanzar los 0.7, al tiempo que se mantenía un recall alto, rozando el 0.8.

Observando la matriz de confusión del modelo, en la figura 4.13, se puede comprobar que el modelo aún tiene algunos problemas para determinar los estados de los semáforos. Sin embargo, se puede observar una cierta mejora en los verdaderos positivos, en especial para el estado 3.

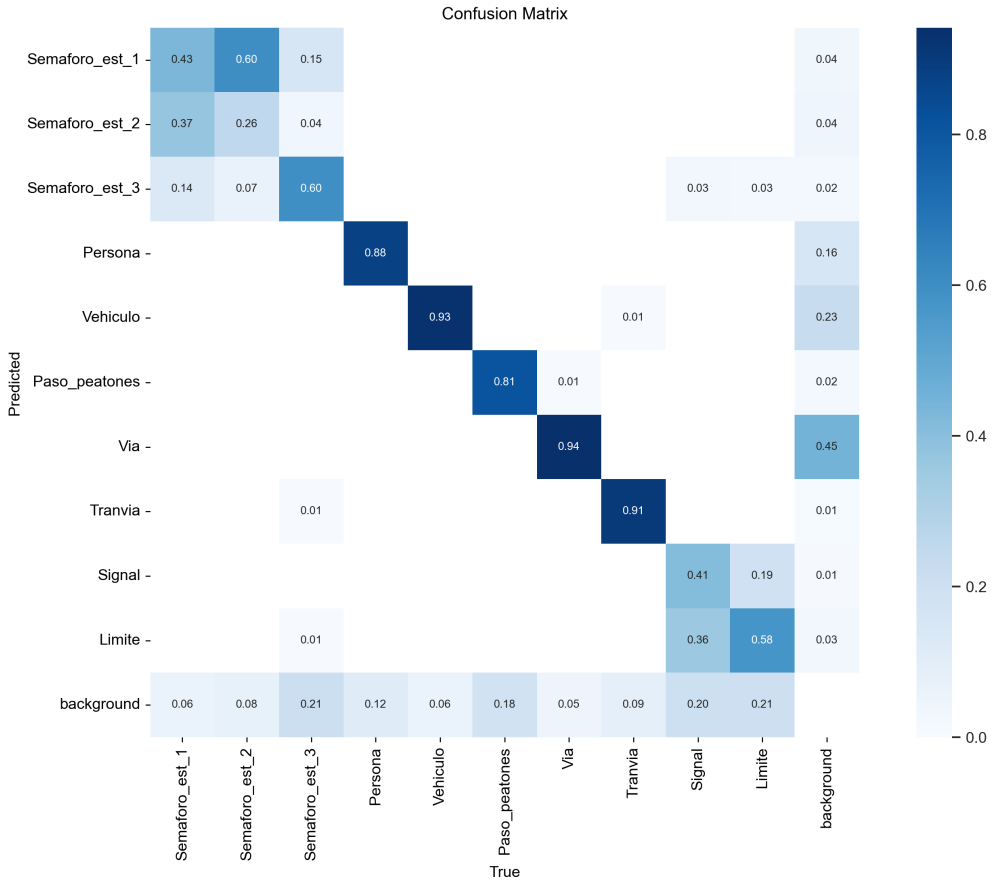


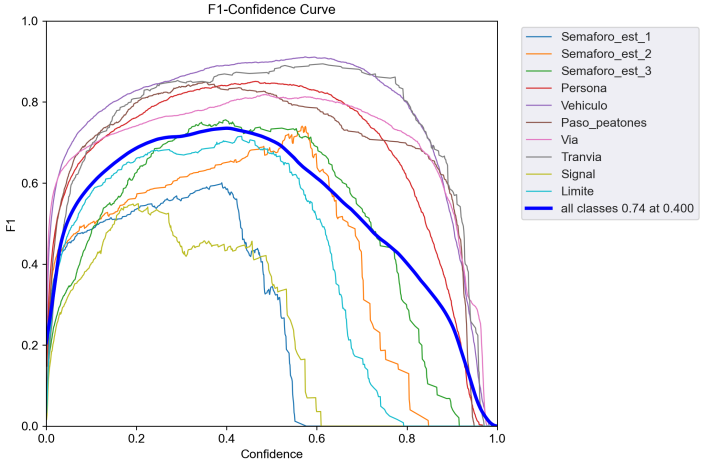
Figura 4.13: Matriz de confusión del quinto modelo

Se especula que las dificultades observadas para diferenciar semáforos en estado 2 y 1 se debe al gran parecido entre ambos indicativos, por lo que en el futuro habría que insistir en mejorar el número de imágenes de estos dos estados.

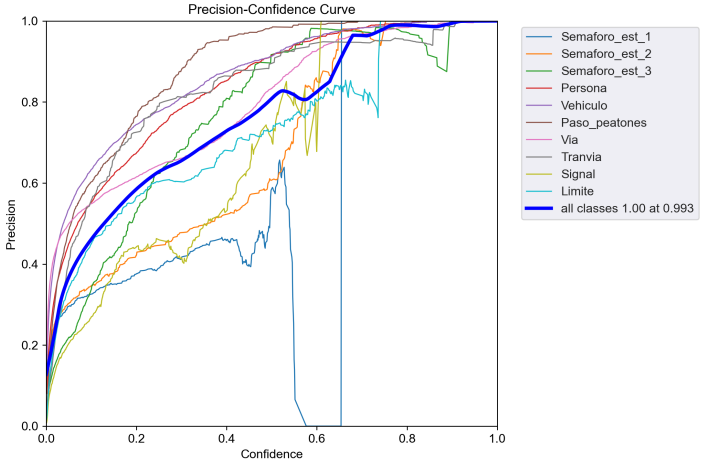
Finalmente, al observar las curvas individuales de F1, precisión y recall para

cada clase en la figura 4.14, podemos observar que, en general, ha habido un aumento en los niveles de confianza de detección del modelo, pudiendo obtener resultados notables incluso por encima del 0.5 en todas las clases más relevantes (semáforos, personas, vehículos y vías).

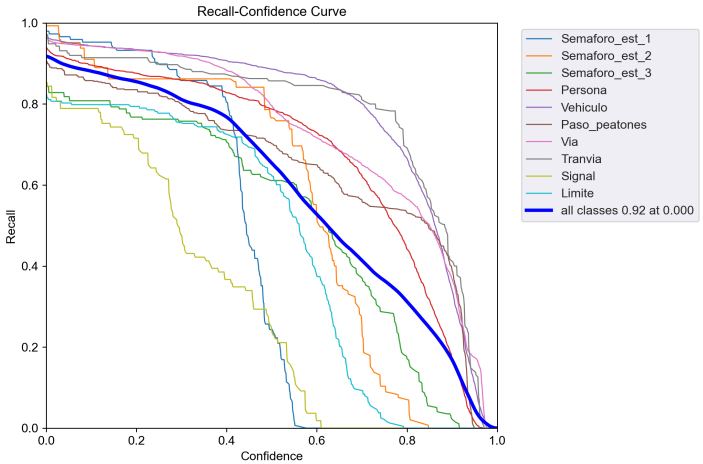
Estos resultados positivos han hecho que, a pesar de las aún evidentes taras que posee el modelo, se considere que, en terminos generales, el modelo ha alcanzado unos niveles de calidad suficientes para una primera implementación del sistema, por lo que se ha definido este modelo como el definitivo de este proyecto.



(a) F1-score



(b) Precisión



(c) Recall

Figura 4.14: Curvas de F1-score, Precisión y recall en función de la confianza del cuarto entrenamiento

Capítulo 5

Conclusiones y trabajos futuros

En este capítulo, finalizada la explicación de todo el desarrollo y los resultados obtenidos en capítulos anteriores, se presentarán la conclusiones obtenidas del desarrollo del proyecto, así como los posibles trabajos futuros que se podrían desarrollar a partir de este proyecto.

5.1 Conclusiones

En este presente proyecto se ha implementado un sistema de detección de obstáculos mediante Inteligencia Artificial para los tranvías de la red de tranvías de Tenerife, Islas Canarias, España. Para ello, se ha seleccionado y entrenado un modelo deep learning, basado en el modelo YOLOv5, para la identificación de obstáculos en la vía y la lectura de semáforos mediante una señal de vídeo, proporcionada por una cámara o vídeo, y se han implementado un programa que interpreta los datos devueltos por el modelo y emita las alertas correspondientes al conductor del tranvía, siendo ejecutado este modelo y programa de procesado de la información en un dispositivo NVIDIA Jetson Nano.

Para lograr este objetivo, se han seguido los siguientes pasos:

A) Estudio del estado del arte de los sistemas de detección de objetos mediante

inteligencia artificial.

B) Búsqueda y selección de un modelo que cumpla los requisitos de rendimiento y compatibilidad con el hardware establecidos para el proyecto.

C) Entrenamiento del modelo YOLOv5 seleccionado hasta lograr un modelo capaz de detectar los elementos deseados con la precisión deseada.

D) Diseño un programa que ejecute el modelo entrenado en una Jetson Nano y procese la información devuelta por el modelo para la emisión de alertas al conductor del tranvía, con dos versiones: uno mostrando información más completa, para las comprobaciones de funcionamiento, y otro mostrando información más simple y concisa sobre los objetos detectados que requieran atención, para su uso habitual durante la operación del tranvía.

Tras el análisis de los resultados del entrenamiento, y en especial del último, se ha podido comprobar que el modelo resultante, a pesar de sus fallas, es funcional y puede ser usado para una primera implementación del sistema de detección de obstáculos, que podrá eventualmente ayudar a reducir la tasa de incidentes en la red de tranvías de Tenerife.

Este proyecto, además, ha permitido demostrar que es posible la implementación de este sistema utilizando una Jetson Nano, permitiendo, gracias a su pequeño tamaño, bajo consumo y facilidad de uso, su uso tanto para la realización de pruebas del sistema como su eventual instalación de la red de tranvías de Tenerife sin necesidad de adaptaciones complejas de los trenes disponibles.

Para finalizar, a nivel general, podemos concluir que este proyecto demuestra la viabilidad de este sistema de detección de obstáculos, abriendo la puerta a futuros desarrollos enfocados en el pulimento de este sistema, su puesta en

marcha y explotación comercial.

5.2 Conclusions

In this project, an obstacle detection system using Artificial Intelligence has been implemented for trams in the tram network of Tenerife, Canary Islands, Spain. For this purpose, a deep learning model has been selected and trained, based on the YOLOv5 model, for the identification of obstacles on the track and the reading of traffic lights using a video signal, provided by a camera or video source, and a program has been implemented to interpret the data returned by the model and issue the corresponding alerts to the tram driver, this model and information processing program being executed on an NVIDIA Jetson Nano device.

In order to achieve this objective, the following steps have been taken:

A) State-of-the-art study of object detection systems using artificial intelligence.

B) Search and selection of a model that meets the performance and hardware compatibility requirements set for the project.

C) Training of the selected YOLOv5 model until a model capable of detecting the desired elements with the desired accuracy is achieved.

D) Design of a program that runs the trained model on a Jetson Nano and processes the information returned by the model for issuing alerts to the tram driver, with two versions: one showing more complete information, for operational checks, and another showing simpler and more concise

information about detected objects that require attention, for regular use during tram operation.

After the analysis of the training results, and especially of the last one, it has been possible to verify that the resulting model, despite its flaws, is functional and can be used for a first implementation of the obstacle detection system, which may eventually help to reduce the incident rate in the Tenerife tram network.

This project has also demonstrated that it is possible to implement this system using a Jetson Nano, allowing, thanks to its small size, low power consumption and ease of use, its use both for testing the system and its eventual installation on the Tenerife tram network without the need for complex adaptations to the available trains.

To conclude, on a general level, we can conclude that this project demonstrates the viability of this obstacle detection system, opening the door to future developments focused on the polishing of this system, its implementation and commercial exploitation.

5.3 Trabajos futuros

De cara a futuros trabajos, se han propuesto la siguientes opciones:

- Mejoras en entrenamiento del modelo: El modelo actual, aunque funcional, aún tiene bastantes deficiencias, que podrían solventarse mediante la creación de un dataset de entrenamiento más completo, incorporándo más instáncias de las clases más problemáticas, además de incluir imágenes a diferentes horas del día y condiciones climáticas, con vistas a mejorar la capacidad del modelo de generalizar, pudiendo detectar estas clases en condiciones diversas.

- Actualización del modelo a versiones más modernas de YOLO: Aunque fue descartado de este proyecto para evitar problemas de compatibilidad con los sistemas ya implementados, y centrarse en opciones cuya funcionalidad era conocida, las librerías utilizadas para la ejecución del modelo en la Jetson Nano están preparadas para utilizar modelos basados en YOLOv7, por lo que en futuros trabajos se puede estudiar la actualización a esta versión más moderna del modelo, lo que permitiría mejorar el rendimiento general del sistema.

- Mejoras de los programas de ejecución del modelo y procesamiento de la información: Los actuales programas implementados podrían ser mejorados en el futuro de diversas maneras, desde implementando formas de pasarle parámetros que actualmente están programados hard-code como argumentos a la hora de ejecutarlos hasta tratar de unificarlos en uno solo y dejar la selección de qué procesamiento hacer como un parámetro del programa.

- Implementación de sistemas para detectar obstáculos no definidos: Actualmente, el programa solo puede detectar obstáculos conocidos en la vía, como personas o vehículos. En el futuro, se podría mejorar el sistema para que sea capaz de determinar si hay algún obstáculo distinto a los conocidos en la vía, para así preparar el sistema para lidiar con situaciones raras no consideradas inicialmente, como la presencia de rocas en la vía.

- Realización de pruebas de campo: El sistema actual podría someterse a una serie de pruebas controladas montado en uno de los tranvías de la red de tranvías de Tenerife, para así comprobar el rendimiento del sistema ante situaciones reales.

- Actualización del hardware utilizado: Aunque para la realización de este

proyecto se decidió utilizar la Jetson Nano por ser un sistema maduro y con amplia documentación disponible, actualmente NVIDIA oferta versiones más modernas de su plataforma, como pueden ser las Jetson Orin Nano [26], que permitirían mejorar el rendimiento del sistema y, gracias al soporte de versiones más modernas de python, optar por otros modelos más modernos que actualmente no son compatibles con la Jetson Nano.

5.4 Agradecimientos

Esta investigación resulta de del Proyecto Estratégico SCITALA (C064/23), fruto del convenio de colaboración suscrito entre el Instituto Nacional de Ciberseguridad (INCIBE) y la Universidad de La Laguna. Esta iniciativa se realiza en el marco de los fondos del Plan de Recuperación, Transformación y Resiliencia, financiados por la Unión Europea (Next Generation).

Bibliografía

[1] MathWorksInc., "What is object detection?" 2017, accedido el 28/06/2024. [Online]. Available: https://es.mathworks.com/discovery/object-detection.html

[2] Na8, "Modelos de detección de objetos," 2020, accedido el 28/06/2024. [Online]. Available: https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/

[3] D. Franklin, "Re-training ssd-mobilenet," 2023, accedido el 28/06/2024. [Online]. Available: https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-ssd.md

[4] IBM, "What is a neural network?" 2024, accedido el 29/06/2024. [Online]. Available: https://www.ibm.com/topics/neural-networks

[5] S. D. González, "Redes neuronales caso práctico," apuntes de clase de sistemas de percepción.

[6] UNIR, "¿qué es el transfer learning y qué ventajas tiene?" *UNIR REVISTA*, 2023, accedido el 29/06/2024. [Online]. Available: https://www.unir.net/ingenieria/revista/transfer-learning/#:~:text=El%20transfer%20learning%20consiste%20en,se%20puede%20compartir%20entre%20ellos.

[7] J. Glenn, M. R. Munawar, and A. Vina, "Performance metrics deep dive," 2024, accedido el 30/06/2024. [Online]. Available: https://docs.ultralytics.com/guides/yolo-performance-metrics/#official-documentation-and-resources

[8] A. F. Gad, "Evaluating deep learning models: The confusion matrix, accuracy, precision, and recall," *Paperspace*, 2020, accedido el 30/06/2024. [Online]. Available: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/

[9] Y. Sasaki, "The truth of the f-measure," *Teach Tutor Mater*, pp. 1–5, 10 2007.

[10] *Gama GeForce RTX 4060*, NVIDIA. [Online]. Available: https://www.nvidia.com/es-es/geforce/graphics-cards/40-series/rtx-4060-4060ti/

[11] P. Burns, "A guide to ai tops and npu performance metrics," *ONQ BLOG*, 2024, accedido el 01/07/2024. [Online]. Available: https://www.qualcomm.com/news/onq/2024/04/a-guide-to-ai-tops-and-npu-performance-metrics

[12] *Jetson Nano*, NVIDIA. [Online]. Available: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/product-development/

[13] *JetPack SDK 4.6 Release Page*, NVIDIA. [Online]. Available: https://developer.nvidia.com/embedded/jetpack-sdk-46

[14] D. Franklin, "Hello ai world project repository," 2023, accedido el 01/07/2024. [Online]. Available: https://github.com/dusty-nv/jetson-inference

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[16] D. Franklin, "Transfer learning with pytorch," 2023, accedido el 01/07/2024. [Online]. Available: https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-transfer-learning.md

[17] J. Glenn, M. R. Munawar, and A. Vina, "Yolo: A brief history," 2023, accedido el 01/07/2024. [Online]. Available: https://docs.ultralytics.com/#yolo-a-brief-history

[18] F. Mattioli, J. Glenn, and S. Waxmann, "Architecture summary," 2023, accedido el 01/07/2024. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure

[19] "Beginnersguide/overview - python," 2023, accedido el 01/07/2024. [Online]. Available: https://wiki.python.org/moin/BeginnersGuide/Overview

[20] L. S. Community, "labelimg repository," 2022, accedido el 01/07/2024. [Online]. Available: https://github.com/HumanSignal/labelImg

[21] Ultralytics, "yolov5," 2023, accedido el 01/07/2024. [Online]. Available: https://github.com/ultralytics/yolov5#pretrained-checkpoints

[22] J. Glenn and Burhan, “Train custom data,” 2023, accedido el 01/07/2024. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

[23] mailrocketsystems, “Jetsonyolov5,” 2023, accedido el 01/07/2024. [Online]. Available: https://github.com/mailrocketsystems/JetsonYolov5

[24] “Pytorch for jetson,” 2023, accedido el 01/07/2024. [Online]. Available: https://forums.developer.nvidia.com/t/pytorch-for-jetson/72048

[25] “Cannot instantiate ‘windowspath’ when loading learner,” 2021, accedido el 02/07/2024. [Online]. Available: https://forums.fast.ai/t/cannot-instantiate-windowspath-when-loading-learner/86564

[26] *Jetson Modules*, NVIDIA. [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules

Apéndice A

Códigos

A.1 Separar_conjuntos_entrenamiento.ipynb

Celda 1:

```
import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split
```

Celda 2:

```
data_path = "./Segunda_prueba"

data_list = os.listdir(data_path)
data_list.remove("labels")
data_list_names = []
for i in data_list:
    data_list_names.append(i.split(".")[0])

dataset = pd.DataFrame(data_list_names)

dataset.head()
```

Celda 3:

```
data_train, data_val = train_test_split(dataset, test_size=0.3, random_state=2)

print(f"Imagenes de entrenamiento: {len(data_train)}\n",f"Imagenes de validación:
                                                        {len(data_val)}", sep="")
data_train.head()
```

Celda 4:

```
data_train_list = data_train.to_numpy().transpose().tolist()[0]
data_val_list = data_val.to_numpy().transpose().tolist()[0]

dataset_path = "./dataset_test_2"

print("Moviendo imagenes de entrenamiento...", end="")
for f in data_train_list:
    origin_images = data_path + "/" + f + ".png"
    destiny_images = dataset_path + "/images/train/" + f + ".png"
    shutil.move(origin_images, destiny_images)

    origin_labels = data_path + "/labels/" + f + ".txt"
    destiny_labels = dataset_path + "/labels/train/" + f + ".txt"
    shutil.move(origin_labels, destiny_labels)

print("Listo")

print("Moviendo imagenes de validación...", end="")
for f in data_val_list:
    origin_images = data_path + "/" + f + ".png"
    destiny_images = dataset_path + "/images/val/" + f + ".png"
    shutil.move(origin_images, destiny_images)

    origin_labels = data_path + "/labels/" + f + ".txt"
    destiny_labels = dataset_path + "/labels/val/" + f + ".txt"
    shutil.move(origin_labels, destiny_labels)

print("Listo")
```

```
print("Moviendo archivo de clases...", end="")
origin = data_path + "/labels/" + "classes.txt"
destiny_copy = dataset_path + "/labels/train/" + "classes.txt"
destiny_move = dataset_path + "/labels/val/" + "classes.txt"
shutil.copy(origin, destiny_copy)
shutil.move(origin, destiny_move)
print("Listo")

print("\nSeparación completada")
```

A.2 Separar_conjuntos_entrenamiento_con_clases_mas _importantes.ipynb

Celda 1:

```
import pandas as pd
import os
import shutil
from sklearn.model_selection import train_test_split
```

Celda 2:

```
data_path = "./Quinta_prueba"

data_list = os.listdir(data_path)
data_list.remove("labels")
data_list_names = []
for i in data_list:
    data_list_names.append(i.split(".")[0])

Most_imp_class_list = []
for i in data_list_names:
    fichero = open(data_path + "/labels/" + i + ".txt")
    lineas = fichero.readlines()
    clases = []
```

```
    for linea in lineas:
        clases.append(linea.split(" ")[0])

    #print(clases)
    fichero.close()

    if '0' in clases: #Semaforo_est_1
        Most_imp_class_list.append(0)
    elif '1' in clases: #Semaforo_est_2
        Most_imp_class_list.append(1)
    elif '2' in clases: #Semaforo_est_3
        Most_imp_class_list.append(2)
    elif '5' in clases: #Paso_peatones
        Most_imp_class_list.append(5)
    elif '7' in clases: #Tranvia
        Most_imp_class_list.append(7)
    elif '8' in clases: #Signal
        Most_imp_class_list.append(8)
    elif '9' in clases: #Limite
        Most_imp_class_list.append(9)
    elif '3' in clases: #Persona
        Most_imp_class_list.append(3)
    elif '4' in clases: #Vehiculo
        Most_imp_class_list.append(4)
    else: #Via
        Most_imp_class_list.append(6)

d = {'0':data_list_names, '1':Most_imp_class_list}
dataset = pd.DataFrame(d)

dataset.head(10)
```

Celda 3:

```
data_train, data_val, class_train,
    class_val = train_test_split(dataset['0'], dataset['1'],
    test_size=0.3, random_state=2,stratify=dataset['1'])
```

```
print(f"Imagenes de entrenamiento: {len(data_train)}\n",f"Imagenes de validación:
            {len(data_val)}", sep="")

for i in range(0,10):
    print("Imagenes clase {} de entrenamiento: {}"
                .format(i,len(np.where(class_train.values == i)[0])))
    print("Imagenes clase {} de validación: {}"
                .format(i,len(np.where(class_val.values == i)[0])))

data_train.head(10)
```

Celda 4:

```
data_train_list = data_train.values.tolist()
data_val_list = data_val.values.tolist()

dataset_path = "./dataset_test_4"

print("Moviendo imagenes de entrenamiento...", end="")
for f in data_train_list:
    origin_images = data_path + "/" + f + ".png"
    destiny_images = dataset_path + "/images/train/" + f + ".png"
    shutil.move(origin_images, destiny_images)

    origin_labels = data_path + "/labels/" + f + ".txt"
    destiny_labels = dataset_path + "/labels/train/" + f + ".txt"
    shutil.move(origin_labels, destiny_labels)

print("Listo")

print("Moviendo imagenes de validación...", end="")
for f in data_val_list:
    origin_images = data_path + "/" + f + ".png"
    destiny_images = dataset_path + "/images/val/" + f + ".png"
    shutil.move(origin_images, destiny_images)

    origin_labels = data_path + "/labels/" + f + ".txt"
    destiny_labels = dataset_path + "/labels/val/" + f + ".txt"
```

```
    shutil.move(origin_labels, destiny_labels)

print("Listo")

print("Moviendo archivo de clases...", end="")
origin = data_path + "/labels/" + "classes.txt"
destiny_copy = dataset_path + "/labels/train/" + "classes.txt"
destiny_move = dataset_path + "/labels/val/" + "classes.txt"
shutil.copy(origin, destiny_copy)
shutil.move(origin, destiny_move)
print("Listo")

print("\nSeparación completada")
```

A.3 Duplicar_borrar_imagenes.py

```
import os
import cv2
import shutil
import pyautogui

data_path = "./Quinta_prueba"
indice = 0
color = ((226, 173, 93),(141, 214, 88),(65, 176, 245),(206, 143, 187),(111, 220, 247),
        (0, 84, 211),(240, 222, 235),(86, 155, 35),(111, 44, 91),(146, 145, 131))

fichero = open(data_path + "/labels/" + "classes.txt")
clases_temp = fichero.readlines()
fichero.close()
clases = []
for i in clases_temp:
    clases.append(i.split("\n")[0])

while True:
    data_list = os.listdir(data_path)
    data_list.remove("labels")
    data_list_names = []
```

```
for i in data_list:
    data_list_names.append(i.split(".")[0])

img = cv2.imread(data_path + "/" + data_list_names[indice] + ".png")
fichero = open(data_path + "/labels/" + data_list_names[indice] + ".txt")
lineas = fichero.readlines()
fichero.close()
for linea in range(0, len(lineas)):
    clase = lineas[linea].split(" ")
    x_center = float(clase[1]) * img.shape[1]
    y_center = float(clase[2]) * img.shape[0]
    width = float(clase[3]) * img.shape[1]
    height = float(clase[4]) * img.shape[0]
    c1 = (int(x_center - width / 2), int(y_center - height / 2))
    c2 = (int(x_center + width / 2), int(y_center + height / 2))
    cv2.rectangle(img, c1, c2, color[int(clase[0])],3)
    c2 = (c1[0], c1[1] - 10)
    cv2.putText(img, clases[int(clase[0])], c2, cv2.FONT_HERSHEY_SIMPLEX, 0.65,
    color[int(clase[0])], 2)

cv2.imshow("Img", img)
cv2.setWindowTitle("Img", data_list_names[indice] + ".png")

opcion = pyautogui.confirm(text="Seleccione accion", title="Panel de control",
        buttons=["Img_anterior", "Duplicar_Img", "Borrar_Img","Img_siguiente","Salir"])

if opcion == "Img_anterior":
    if indice <= 0:
        indice = len(data_list_names) - 1
    else:
        indice = indice - 1
elif opcion == "Img_siguiente":
    if indice >= len(data_list_names) - 1:
        indice = 0
    else:
        indice = indice + 1
elif opcion == "Duplicar_Img":
    inicio = data_list_names[indice].split("_")[0]
    i = 1
```

```
        while True:
            if data_list_names[indice + i].split("_")[0] != inicio:
                if len(data_list_names[indice + i - 1].split("_")) > 1:
                    a = int(data_list_names[indice + i -1].split("_")[1])
                    nuevo_subindice = "_" + str(a + 1)
                else:
                    nuevo_subindice = "_1"
                origin_img = data_path + "/" + inicio + ".png"
                destiny_img = data_path + "/" + inicio + nuevo_subindice + ".png"
                shutil.copy(origin_img, destiny_img)

                origin_label = data_path + "/labels/" + inicio + ".txt"
                destiny_label = data_path + "/labels/" + inicio + nuevo_subindice + ".txt"
                shutil.copy(origin_label, destiny_label)

                break
            else:
                i = i + 1
        indice = indice + 1
    elif opcion == "Borrar_Img":
        os.remove(data_path + "/" + data_list_names[indice] + ".png")
        os.remove(data_path + "/labels/" + data_list_names[indice] + ".txt")
        indice = indice - 1
    elif opcion == "Salir":
        break
    else:
        pass

cv2.destroyAllWindows()
```

A.4 detectar_objeto.py

```
import sys
import cv2
import imutils
from yoloDet import YoloTRT
```

```
input_path = "/home/cryptull/Jorge_Diaz/Videos/TR2.mp4"
output_path = "TR2_procesado.mp4"
library_path = "yolov5/build/libmyplugins.so"
engine_path = "yolov5/build/test_2.engine"
winName = "Output"

# use path for library and engine file
model = YoloTRT(library=library_path, engine=engine_path, conf=0.4, yolo_ver="v5")

cap = cv2.VideoCapture(input_path)
out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), 30, (1280,720))

while (cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        frame = imutils.resize(frame, width=600)
        detections, t = model.Inference(frame)

        FPS = "FPS: {:.2f}".format(1/t)
        cv2.putText(frame, FPS, (5,15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
        resized_frame = cv2.resize(frame, (1280,720), interpolation=cv2.INTER_AREA)
        out.write(resized_frame)

        title = winName + ". " + FPS
        cv2.imshow(winName, resized_frame)
        cv2.setWindowTitle(winName,title)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            print("Cierre solicitado por el usuario")
            break
    else:
        print("Error al obtener el frame")
        break

print("Liberando archivos...", end="")
cap.release()
out.release()
print("Hecho")
```

```
print("Cerrando ventanas...", end="")
cv2.destroyAllWindows()
print("Hecho","\nVideo resultante guardado en {}".format(output_path))
```

A.5 detectar_objetos_con_alertas.py

```
"""
detectar_objetos_con_alertas.py, version 3

Este programa procesa las imagenes de un video o fuente de video suministrada y
la procesa con el modelo de IA yoloV5
suministrado, que ha sido entrenado anteriormente con las imagenes correspondientes del
recorrido del tranvia.

Esta version genera un video con todas las detecciones realizadas por el
modelo en cada frame, mostrando su etiqueta
y confianza de deteccion, así como las alertas producidas en función de lo
detectado en la via del tranvia, los estados de los semaforos y otros elementos.
"""

import sys
import cv2
import imutils
import os
import time
from yoloDet import YoloTRT

# Rutas y parametros globales para la ejecucion del programa

input_path = "/home/cryptull/Jorge_Diaz/Videos/TR5.mp4" # Ruta al archivo de video
                                                        # a procesar o a la camara
                                                        # utilizada como input
output_path = "TR5_procesado.mp4" # Ruta al archivo de video de salida
library_path = "yolov5/build/libmyplugins.so" # Ruta al archivo libmyplugins.so
engine_path = "yolov5/build/test_5.engine" # Ruta al archivo engine del modelo
winName = "Output" # Nombre para las ventanas
posiciones_alertas = ((435,15),(435,35),(435,55),(435,75)) # Posiciones para las alertas
```

```
offset = 14 # Offset para la determinacion de si hay un vehiculo en la via
lejano = 168 # Posicion en el eje y a partir de la cual se considera que el objeto
            # detectado esta cerca del tranvia

# Lectura del modelo e inicializacion de la IA
model = YoloTRT(library=library_path, engine=engine_path, conf=0.4, yolo_ver="v5")

# Inicializacion de la captura de video
cap = cv2.VideoCapture(input_path)

# Generacion del archivo de salida
# Si ya existiera, se elimina antes de generar el nuevo para evitar conflictos
if os.path.isfile(output_path):
    os.remove(output_path)

out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'x264'), 25, (1280,720))

# Inicializacion del bucle principal del programa si la captura de video se ha
# iniciado correctamente
while (cap.isOpened()):
    t_init = time.time() # toma del tiempo inical de cada ciclo para control de frames
                        # procesados

    ret, frame = cap.read() # Lectura del frame

    if ret == True: # Se comprueba si se ha obtenido un frame de la entrada de video
                    # antes de procesar

        # Redimensionado de la imagen y procesado de esta por la IA
        frame = imutils.resize(frame, width=600)
        detections, t = model.Inference(frame)

        # Desactivacion de las alarmas activadas en el ciclo anterior y vaciado de los
        # vectores de elementos detectados
        Personas = []
        Vehiculos = []
        Vias = []
        Riesgo_persona = False
        Riesgo_vehiculo = False
```

```
Advertencia_persona = False
Advertencia_vehiculo = False
Advertencia_paso_peatones = False
Sem_rojo = False
Sem_amarillo = False
Sem_verde = False

for obj in detections: # Recorrido del diccionario de elementos detectados por la
                  # IA en el frame
    # Si alguna de las clases detectada coincide con una de interes, se almacena
    # sus coordenadas
    # o se activa la alerta correspondiente, segun el caso
    if obj['class'] == 'Persona':
        Personas.append(obj['box'])
    elif obj['class'] == 'Vehiculo':
        Vehiculos.append(obj['box'])
    elif obj['class'] == 'Via':
        Vias.append(obj['box'])
    elif obj['class'] == 'Paso_peatones':
        Advertencia_paso_peatones = True
    elif obj['class'] == 'Semaforo_est_1':
        Sem_rojo = True
    elif obj['class'] == 'Semaforo_est_2':
        Sem_amarillo = True
    elif obj['class'] == 'Semaforo_est_3':
        Sem_verde = True

# Comprobacion de si hay un algo sobre las vias
for via in Vias:
    for persona in Personas:
        if persona[0] <= via[2] and persona[2] >= via[0] and
            persona[1] <= via[3] and persona[3] >= via[1]:
            if via[3] <= lejano:
                Advertencia_persona = True
            else:
                Riesgo_persona = True
    for vehiculo in Vehiculos:
        if vehiculo[0] <= via[2] - offset and vehiculo[2] >= via[0] + offset and
            vehiculo[1] <= via[3] and vehiculo[3] >= via[1]:
```

```
            if via[3] <= lejano:
                Advertencia_vehiculo = True
            else:
                Riesgo_vehiculo = True

# Indicador de advertencia general
# Muestra la accion deseada en funcion de la alerta mas seria activada
cv2.putText(frame, "Status:", (200,20), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
            (255,255,255), 2)
if Riesgo_persona or Riesgo_vehiculo or Sem_rojo:
    cv2.putText(frame, "Detengase", (280,20), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
                (0,0,255), 2)
elif Advertencia_persona or Advertencia_vehiculo or Advertencia_paso_peatones or
        Sem_amarillo:
    cv2.putText(frame, "Precaucion", (280,20), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
                (0,140,255), 2)
else:
    cv2.putText(frame, "Continue", (280,20), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
                (0,255,0), 2)

# Indicadores de advertencia específica
# Muestran el tipo de alerta especifico que se ha activado
indice = 0
if Riesgo_persona: # Persona en la via
    cv2.putText(frame, "Persona en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
    indice = indice + 1
elif Advertencia_persona: # Persona en la via a lo lejos
    cv2.putText(frame, "Persona en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
    indice = indice + 1

if Riesgo_vehiculo: # Vehiculo en la via
    cv2.putText(frame, "Vehiculo en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
    indice = indice + 1
elif Advertencia_vehiculo: # Vehiculo en la via a lo lejos
    cv2.putText(frame, "Vehiculo en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
```

```
        indice = indice + 1

    if Advertencia_paso_peatones: # Paso de peatones mas adelante
        cv2.putText(frame, "P. peatones delante", posiciones_alertas[indice],
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
        indice = indice + 1

    if Sem_rojo: # Semaforo en rojo
        cv2.putText(frame, "Semaforo en rojo", posiciones_alertas[indice],
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
        indice = indice + 1
    elif Sem_amarillo: # Semaforo en amarillo
        cv2.putText(frame, "Semaforo en amarillo", posiciones_alertas[indice],
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
        indice = indice + 1
    elif Sem_verde: # Semaforo en verde
        cv2.putText(frame, "Semaforo en verde", posiciones_alertas[indice],
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 1)
        indice = indice + 1

    # Obtencion del tiempo final y escritura de los fps obtenidos
    t_end = time.time()
    FPS = "FPS: {:.2f}".format(1/(t_end - t_init))
    cv2.putText(frame, FPS, (5,15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)

    # Ajuste de la imagen a las dimensiones del video
    resized_frame = cv2.resize(frame, (1280,720), interpolation=cv2.INTER_AREA)

    # Escritura del frame en el video
    out.write(resized_frame)

    # Visualizacion del frame
    title = winName + ". " + FPS
    cv2.imshow(winName, resized_frame)
    cv2.setWindowTitle(winName,title)

    # En caso de que el usuario pulse la tecla 'q', se cerrararia el programa
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
```

```
            print("Cierre solicitado por el usuario")
            break
    else: # Cierre del programa en caso de no poder obtenerse el fram
        print("Error al obtener el frame")
        break

# Liberacion de los archivos utilizados y cierre de todas las ventanas generadas
print("Liberando archivos...", end="")
cap.release()
out.release()
print("Hecho")
print("Cerrando ventanas...", end="")
cv2.destroyAllWindows()
print("Hecho","\nVideo resultante guardado en {}".format(output_path))
```

A.6 detectar_objetos_con_alertas_visual_usuario.py

```
"""
detectar_objetos_con_alertas_visual_usuario.py, version 4

Este programa procesa las imagenes de un video o fuente de video suministrada y la
procesa con el modelo de IA yoloV5 suministrado, que ha sido entrenado anteriormente con
las imagenes correspondientes del recorrido del tranvia.

Esta version genera un video solo mostrando los elementos que han generado una alerta,
así como las alertas producidas en función de lo detectado en la via del tranvia, los
estados de los semaforos y otros elementos.
La visualización obtenida simula a la que veria el conductor del tranvia durante una
operacion normal del tranvia.
"""

import sys
import cv2
import imutils
import os
import time
```

```
from yoloDet import YoloTRT

# Rutas y parametros globales para la ejecucion del programa

input_path = "/home/cryptull/Jorge_Diaz/Videos/TR5.mp4" # Ruta al archivo de video a
                                                        # procesar o a la camara
                                                        # utilizada como input
output_user_path = "TR5_visual_usario_procesado.mp4" # Ruta al archivo de video de salida
                                                    # de la visual del conductor
library_path = "yolov5/build/libmyplugins.so" # Ruta al archivo libmyplugins.so
engine_path = "yolov5/build/test_5.engine" # Ruta al archivo engine del modelo
winName_user = "Output_usuario" # Nombre para las ventanas de la visual del usuario
posiciones_alertas = ((435,15),(435,35),(435,55),(435,75)) # Posiciones para las alertas
offset = 14 # Offset para la determinacion de si hay un vehiculo en la via
lejano = 168 # Posicion en el eje y a partir de la cual se considera que el objeto
            # detectado esta cerca del tranvia

# Lectura del modelo e inicializacion de la IA
model = YoloTRT(library=library_path, engine=engine_path, conf=0.4, yolo_ver="v5")

# Inicializacion de la captura de video
cap = cv2.VideoCapture(input_path)

# Generacion del archivo de salida
# Si ya existiera, se elimina antes de generar el nuevo para evitar conflictos
if os.path.isfile(output_user_path):
    os.remove(output_user_path)

out_user = cv2.VideoWriter(output_user_path, cv2.VideoWriter_fourcc(*'x264'), 25,
                           (1280,720))

# Inicializacion del bucle principal del programa si la captura de video se ha iniciado
# correctamente
while (cap.isOpened()):
    t_init = time.time() # toma del tiempo inical de cada ciclo para control de frames
                        # procesados

    ret, frame = cap.read() # Lectura del frame
```

```
    if ret == True: # Se comprueba si se ha obtenido un frame de la entrada de video
                    # antes de procesar

        # Redimensionado de la imagen y procesado de esta por la IA
        frame = imutils.resize(frame, width=600)
        frame_usuario = frame.copy() # Copia del frame para el video de vision del
                                     # usuario
        detections, t = model.Inference(frame)

        # Desactivacion de las alarmas activadas en el ciclo anterior y vaciado de los
        # vectores de elementos detectados
        Personas = []
        Vehiculos = []
        Vias = []
        Riesgo_persona = False
        Riesgo_vehiculo = False
        Advertencia_persona = False
        Advertencia_vehiculo = False
        Advertencia_paso_peatones = False
        Sem_rojo = False
        Sem_amarillo = False
        Sem_verde = False

        for obj in detections: # Recorrido del diccionario de elementos detectados por
                               # la IA en el frame
            # Si alguna de las clases detectada coincide con una de interes, se almacena
            # sus coordenadas o se activa la alerta correspondiente, segun el caso
            # Tambien se resaltan los elementos que han hecho saltar la alerta
            if obj['class'] == 'Persona':
                Personas.append(obj['box'])
            elif obj['class'] == 'Vehiculo':
                Vehiculos.append(obj['box'])
            elif obj['class'] == 'Via':
                Vias.append(obj['box'])
            elif obj['class'] == 'Paso_peatones':
                Advertencia_paso_peatones = True
                c1 = (int(obj['box'][0]), int(obj['box'][1]))
                c2 = (int(obj['box'][2]), int(obj['box'][3]))
                cv2.rectangle(frame_usuario, c1, c2, (0,140,255),2)
```

```
        elif obj['class'] == 'Semaforo_est_1':
            Sem_rojo = True
            c1 = (int(obj['box'][0]), int(obj['box'][1]))
            c2 = (int(obj['box'][2]), int(obj['box'][3]))
            cv2.rectangle(frame_usuario, c1, c2, (0,0,255),2)
        elif obj['class'] == 'Semaforo_est_2':
            Sem_amarillo = True
            c1 = (int(obj['box'][0]), int(obj['box'][1]))
            c2 = (int(obj['box'][2]), int(obj['box'][3]))
            cv2.rectangle(frame_usuario, c1, c2, (0,140,255),2)
        elif obj['class'] == 'Semaforo_est_3':
            Sem_verde = True
            c1 = (int(obj['box'][0]), int(obj['box'][1]))
            c2 = (int(obj['box'][2]), int(obj['box'][3]))
            cv2.rectangle(frame_usuario, c1, c2, (0,255,0),2)

    # Comprobacion de si hay un algo sobre las vias
    for via in Vias:
        for persona in Personas:
            if persona[0] <= via[2] and persona[2] >= via[0] and
                persona[1] <= via[3] and persona[3] >= via[1]:
                if via[3] <= lejano:
                    Advertencia_persona = True
                    c1 = (int(persona[0]), int(persona[1]))
                    c2 = (int(persona[2]), int(persona[3]))
                    cv2.rectangle(frame_usuario, c1, c2, (0,140,255),2)
                else:
                    Riesgo_persona = True
                    c1 = (int(persona[0]), int(persona[1]))
                    c2 = (int(persona[2]), int(persona[3]))
                    cv2.rectangle(frame_usuario, c1, c2, (0,0,255),2)
        for vehiculo in Vehiculos:
            if vehiculo[0] <= via[2] - offset and vehiculo[2] >= via[0] + offset and
                vehiculo[1] <= via[3] and vehiculo[3] >= via[1]:
                if via[3] <= lejano:
                    Advertencia_vehiculo = True
                    c1 = (int(vehiculo[0]), int(vehiculo[1]))
                    c2 = (int(vehiculo[2]), int(vehiculo[3]))
                    cv2.rectangle(frame_usuario, c1, c2, (0,140,255),2)
```

```
            else:
                Riesgo_vehiculo = True
                c1 = (int(vehiculo[0]), int(vehiculo[1]))
                c2 = (int(vehiculo[2]), int(vehiculo[3]))
                cv2.rectangle(frame_usuario, c1, c2, (0,0,255),2)

# Indicador de advertencia general
# Muestra la accion deseada en funcion de la alerta mas seria activada
cv2.putText(frame_usuario, "Status:", (200,20), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
            (255,255,255), 2)
if Riesgo_persona or Riesgo_vehiculo or Sem_rojo:
    cv2.putText(frame_usuario, "Detengase", (280,20), cv2.FONT_HERSHEY_SIMPLEX,
                0.65, (0,0,255), 2)
elif Advertencia_persona or Advertencia_vehiculo or Advertencia_paso_peatones or
    Sem_amarillo:
    cv2.putText(frame_usuario, "Precaucion", (280,20), cv2.FONT_HERSHEY_SIMPLEX,
                0.65, (0,140,255), 2)
else:
    cv2.putText(frame_usuario, "Continue", (280,20), cv2.FONT_HERSHEY_SIMPLEX,
                0.65, (0,255,0), 2)

# Indicadores de advertencia específica
# Muestran el tipo de alerta especifico que se ha activado
indice = 0
if Riesgo_persona: # Persona en la via
    cv2.putText(frame_usuario, "Persona en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
    indice = indice + 1
elif Advertencia_persona: # Persona en la via a lo lejos
    cv2.putText(frame_usuario, "Persona en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
    indice = indice + 1

if Riesgo_vehiculo: # Vehiculo en la via
    cv2.putText(frame_usuario, "Vehiculo en la via", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
    indice = indice + 1
elif Advertencia_vehiculo: # Vehiculo en la via a lo lejos
    cv2.putText(frame_usuario, "Vehiculo en la via", posiciones_alertas[indice],
```

```
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
    indice = indice + 1

if Advertencia_paso_peatones: # Paso de peatones mas adelante
    cv2.putText(frame_usuario, "P. peatones delante", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
    indice = indice + 1

if Sem_rojo: # Semaforo en rojo
    cv2.putText(frame_usuario, "Semaforo en rojo", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
    indice = indice + 1
elif Sem_amarillo: # Semaforo en amarillo
    cv2.putText(frame_usuario, "Semaforo en amarillo", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,140,255), 1)
    indice = indice + 1
elif Sem_verde: # Semaforo en verde
    cv2.putText(frame_usuario, "Semaforo en verde", posiciones_alertas[indice],
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 1)
    indice = indice + 1

# Obtencion del tiempo final y escritura de los fps obtenidos
t_end = time.time()
FPS = "FPS: {:.2f}".format(1/(t_end - t_init))
cv2.putText(frame_usuario, FPS, (5,15), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
            (0,0,255), 1)

# Ajuste de la imagenes a las dimensiones de los videos
resized_frame_usuario = cv2.resize(frame_usuario, (1280,720),
                                   interpolation=cv2.INTER_AREA)

# Escritura de los frames en sus correspondientes videos
out_user.write(resized_frame_usuario)

# Visualizacion de los frames
title_user = winName_user + ". " + FPS
cv2.imshow(winName_user, resized_frame_usuario)
cv2.setWindowTitle(winName_user,title_user)
```

```
        # En caso de que el usuario pulse la tecla 'q', se cerrararia el programa
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            print("Cierre solicitado por el usuario")
            break
    else: # Cierre del programa en caso de no poder obtenerse el fram
        print("Error al obtener el frame")
        break

# Liberacion de los archivos utilizados y cierre de todas las ventanas generadas
print("Liberando archivos...", end="")
cap.release()
out_user.release()
print("Hecho")
print("Cerrando ventanas...", end="")
cv2.destroyAllWindows()
print("Hecho","\nVideo resultante guardado en {}".format(output_user_path))
```