VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

Máster en Ciencia de Datos

Trabajo Fin de Máster

DESARROLLO DE UN SISTEMA DE BAJO COSTE PARA EL ANÁLISIS DE TRÁFICO MEDIANTE EL USO DE DEEP LEARNING

Sergio Morant Gálvez

Tutor: Valero Laparra Pérez-Muelas
Tutor: Jordi Muñoz Mari

Curso académico 2020/21

Resumen

El objetivo de este proyecto es desarrollar e implementar un sistema end-to-end de bajo coste capaz de obtener estadísticas relevantes sobre el tráfico en carretera mediante el uso de *deep learning* en un dispositivo de computación dedicado. El sistema, basado en la reciente tendencia del *edge computing*, obtendrá estadísticas sin necesidad de almacenar ningún dato sobre los vehículos respetando así las leyes de protección de datos.

Para llevar a cabo este proyecto se ha decidido aplicar *transfer learning* sobre una red neuronal ya entrenada, así como recoger y etiquetar un reducido conjunto de datos desde cero para comprender todas las fases del desarrollo de un proyecto basado en *deep learning*.

Por último, se pretende también evaluar el desempeño de diferentes componentes de hardware en el uso de modelos de *deep learning* como la RaspberryPi o el Acelerador TPU Coral USB.

Palabras clave: Deep Learning, Machine Learning, AI, tráfico, visión artificial, Raspberry Pi, TPU, redes neuronales, datos

Código disponible a través del siguiente enlace: https://github.com/smorantg2/tfm_traffic

Abstract

The aim of this project is to develop and implement a low-cost end-to-end system capable of obtaining relevant road traffic data through the use of deep learning in a dedicated computer device. The system, based on the recent trend on edge computing, will obtain statistics without storing any data regarding the vehicles or drivers. This way it also respects privacy and meets data protection laws.

In order to carry out this project, a neural network has been re-trained using transfer learning. Also, a reduced dataset has been gathered and labelled from scratch. This has been helpful to understand all the phases involved in the development of a deep learning-based project.

Finally, the performance of different hardware components will be assessed regarding the use of deep learning models such as the RaspberryPi or the USB TPU Coral Accelerator.

Keywords: Deep Learning, Machine Learning, AI, traffic, computer vision, Raspberry Pi, TPU, neural networks, data

The code can be accessed through the following link:: https://github.com/smorantg2/tfm_traffic

Índice

Figuras:

Tablas:

1. Introducción

1.1 Motivación

La inteligencia artificial (IA) está cada vez más presente en la sociedad, en todo tipo de aplicaciones: industriales, económicas, relacionadas con la salud, relacionadas con el entretenimiento, etc. Cada vez más empresas y dispositivos hacen uso de ella debido a su gran potencial, siendo las redes neuronales las grandes protagonistas de los últimos años. Su profundidad, complejidad, número de parámetros y efectividad no ha hecho más que aumentar recientemente.

Se conoce como redes neuronales a aquellos modelos computacionales vagamente inspirados en el comportamiento de un cerebro humano. En esencia, consisten en un conjunto de unidades llamadas "neuronas artificiales" que están conectadas entre sí. Estos modelos son capaces de, a partir de unos datos de entrada, producir ciertos valores de salida que corresponden a estimaciones o predicciones.

Este tipo de modelo es capaz de aprender a partir de un conjunto de datos de entrenamiento sin necesidad de ser programados de forma explícita. Su desempeño es sobresaliente en áreas donde la detección o predicción de soluciones es difícil de expresar mediante programación convencional.

Las redes neuronales se utilizan para resolver una gran variedad de tareas como el reconocimiento de voz, predicciones sobre series temporales, detección de anomalías y un largo etcétera. Uno de los campos donde más repercusión ha tenido su desarrollo es el de la visión por ordenador donde este tipo de modelos ha permitido mejorar la clasificación de imágenes, la detección de objetos en ellas, crear nuevas imágenes, transferir estilos, aumentar resoluciones y mucho más.

"*Actualmente, en 2016, una regla general es que un algoritmo de aprendizaje profundo supervisado generalmente alcanzará un rendimiento aceptable con alrededor de 5.000 ejemplos etiquetados por categoría, e igualará o superará el rendimiento humano cuando se entrene con un conjunto de datos que contenga al menos 10 millones de ejemplos etiquetados.*" - Ian Goodfellow, investigador y creador de las Generative Adversarial Networks (GAN).

Durante mucho tiempo el uso de algunas redes neuronales para aplicaciones en tiempo real no ha sido posible por la velocidad de inferencia, el coste de computación, su precisión o la falta de datos representativos con los que llevar a cabo los entrenamientos.

Por todo lo anterior, la motivación del presente proyecto es la de demostrar que los últimos avances en este campo han permitido superar estas barreras y es posible desarrollar e implementar desde cero modelos que permitan obtener soluciones a tiempo real utilizando dispositivos de bajo coste y llevando a cabo todas las fases del proyecto;

desde la búsqueda, recogida y el etiquetado de datos hasta el entrenamiento e implementación del modelo.

En la actualidad, uno de los métodos más utilizados para medición del tráfico que circula por una carretera o calle es el uso de tubos neumáticos como los de la figura 1. Se trata de cables huecos por los cuales circula el aire. Se colocan dos cables ligeramente separados y, cuando un vehículo circula por encima de ellos, un sensor manda una señal eléctrica a un sistema de conteo.

Figura 1 - Instalación de tubos neumáticos para el análisis del tráfico.
https://www.diariomotor.com/noticia/cables-carreteras-funcion/

Este tipo de sistema tiene algunos problemas que el sistema desarrollado en este proyecto podría mejorar.

1. Riesgo de afectar a la conducción de vehículos como las motos.
2. Preferencia de instalación en zonas rectas debido a lo señalado en el punto anterior.
3. Compleja diferenciación entre distintos tipos de vehículos.
4. Alto coste.
5. Dificultad de montaje.
6. Desgaste del material, tiempo de vida limitado.

También existen otros métodos basados en radar que solventan algunos de estos problemas, pero cuyo precio es todavía superior.

1.2 Objetivo

El objetivo de este proyecto consiste en desarrollar una primera aproximación a un sistema de bajo coste capaz de obtener información relevante sobre el tráfico a través de imágenes y mediante el uso de redes neuronales profundas o *deep learning* como muestra el esquema de la figura 2.

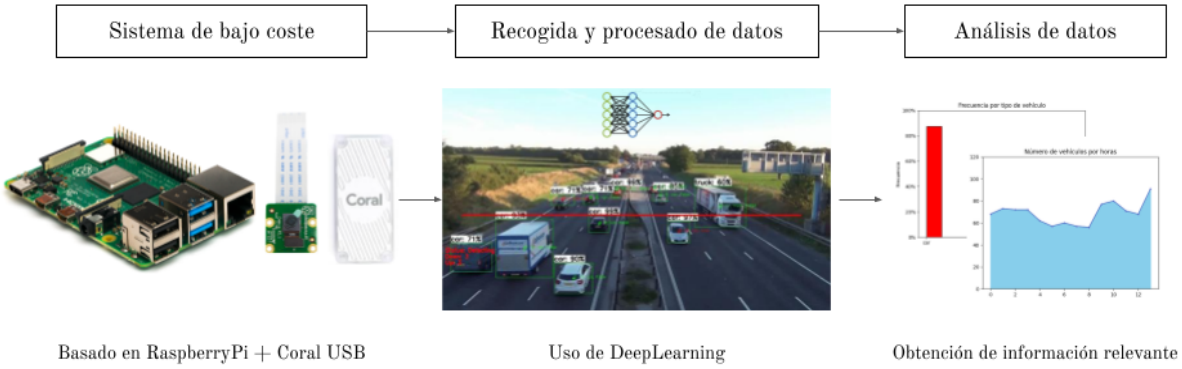

Figura 2 - Esquema explicativo del proyecto.
Elaboración propia

El uso de sistemas embebidos supone una gran limitación del rendimiento, ya que suelen ser dispositivos con menor potencia de cálculo. Sin embargo, esta limitación se puede sortear mediante el uso de componentes de hardware que permitan acelerar los cálculos. Las grandes compañías están poniendo un gran empeño en desarrollar unidades de procesamiento que permitan acelerar la inferencia en este tipo de modelos. Es así como nace la TPU (Tensor Processing Unit) y el paradigma del *Edge Computing*, es decir, la realización del procesamiento y computación cerca de sistemas embebidos, evitando grandes procesadores o el cálculo en la nube. Así se logra reducir el tiempo de respuesta, así como algunas otras ventajas en lo que se refiere a almacenamiento o privacidad.

En este proyecto se van a estudiar distintos dispositivos o hardware, ya que los resultados en cuanto a precisión y tiempos de ejecución se compararán con y sin el uso del acelerador TPU Coral USB así como con los de un ordenador portátil:

- Raspberry Pi + Acelerador TPU Coral USB
- Ordenador portátil Intel i7 de 9a generación, GPU GeForce GTX1660

También se estudiará el efecto del acelerador TPU Coral USB en el PC.

El sistema final, si el desarrollo llevado a cabo en este proyecto fuera a implementarse, constaría de los siguientes elementos cuyo precio actual aproximado se incluye:

Elemento	**Precio**
Raspberry Pi 4 Model B	48,97 €
Acelerador TPU Coral USB	53,73 €
Cámara compatible Raspberry Pi	9,98 €
Alimentación (*Power Bank)*	20,00 €
Pantalla táctil para Raspberry Pi	35,00 €
TOTAL	167,68

Tabla 1 - Precio de los diferentes elementos del sistema.
Elaboración propia.

El precio total de los componentes es de alrededor de 168 €, a los cuales habría que sumar la carcasa y el soporte para montarlo en la localización que se desee y resista las inclemencias del tiempo, por lo que el precio del sistema total se estima alrededor de los 200 €.

Los soportes podrían ser unos simples imanes si se desea colocar en una estructura metálica de iluminación, puente o similar. Respecto a la pantalla táctil, podría no ser necesaria si se mejorara el sistema para detectar la forma de la carretera y marcara una línea de paso de forma automática, lo cual queda fuera del alcance de este proyecto y se deja como sugerencia de trabajo futuro.

La transmisión de datos es posible gracias al módulo Wi-Fi que incluye la Raspberry Pi. En función del uso que se le diera al dispositivo y de su localización podría utilizarse otro tipo de conexión o incluso podrían almacenarse los datos en una memoria para extraerlos más adelante.

2. Descripción del sistema

2.1 Edge Computing

La potencia computacional de los sistemas embebidos ha ido aumentando con el tiempo y actualmente estos son capaces de ejecutar modelos y trabajar con latencias que en el pasado eran impensables. Uno de los ejemplos más claros es el caso de los *smartphones*, dispositivos de uso diario que han alcanzado niveles de computación realmente altos y que permiten el uso de redes neuronales profundas, las cuales ya forman parte de gran número de apps.
El incremento masivo de los dispositivos *IoT* (*Internet of Things*) ha producido a su vez un incremento masivo en la cantidad de datos que deben procesarse y almacenarse en *data centers*.

El *Edge Computing* consiste en evitar los problemas derivados de descentralizar los datos y cálculos moviendo la computación al borde (e*dge*) de los sistemas, lo cual tiene una serie de ventajas importantes, como las destacadas en la figura 2.

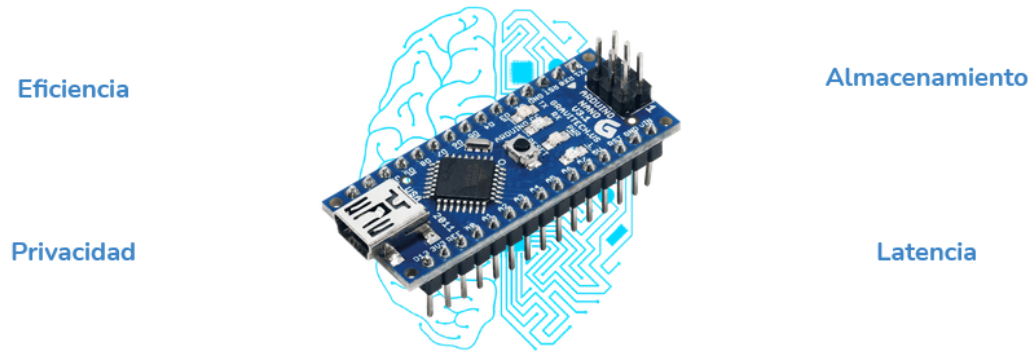

Figura 3 - Beneficios del Edge Computing.
Elaboración propia

El hecho de no tener que enviar los datos permite aumentar el tiempo de respuesta de algunos sistemas, y asegura un funcionamiento independiente de las condiciones de conectividad, lo cual es fundamental para ciertas aplicaciones como es el caso de los vehículos autónomos.
Evitar la transferencia de los datos también permite aumentar la seguridad y privacidad de los datos recogidos por el sistema, ya que son mucho menos susceptibles de ser extraídos o capturados de forma irregular. También se reduce el almacenamiento necesario para muchas aplicaciones ya que, como es el caso de este proyecto, los datos de entrada pueden utilizarse y descartarse cada vez que se ejecuta el modelo.

Por todos estos motivos, el coste de implementación de este tipo de sistemas se reduce drásticamente, tanto por los dispositivos empleados, como por la energía consumida. Es por esto que se trata de un campo en auge donde empresas e investigadores están poniendo el foco.

Como muestra se encuentran compañías como Google, NVIDIA o Intel, las cuales han decidido investigar este campo y han lanzado al mercado productos específicos relacionados con él.

Por supuesto, aunque este tipo de dispositivos sea suficiente para la implementación de muchos modelos, en general, no están pensados para llevar a cabo la fase de entrenamiento de estos, sino que se entrenan en dispositivos con mayor potencia computacional.

2.2 Hardware

En este apartado se comentan brevemente los principales elementos de hardware que forman parte del dispositivo desarrollado en este proyecto, así como algunas de sus funcionalidades y características clave.

2.2.1 Raspberry Pi

La placa de desarrollo Raspberry Pi es una de las más conocidas a nivel mundial. No está pensada específicamente para aplicaciones relacionadas con el *deep learning*, sino que se trata de una placa de carácter general.

Es un ordenador monoplaca o SBC por sus siglas en inglés *Single-Board-Computer*. La primera generación de Raspberry Pi se lanzó en el año 2012 y su sistema operativo oficial es una versión adaptada de Debian llamada Raspbian.

Figura 4 - Raspberry Pi 4 Model B.
Michael Henzler / Wikimedia Commons

Para este proyecto se ha utilizado una Raspberry Pi 4 Model B como la de la figura 3 la cual posee las siguientes especificaciones:

- Procesador ARM Cortex-172 con 4 núcleos a 1,5 GHz
- GPU VideoCore VI
- 8 GB RAM
- Conectividad Bluetooth 5.0, Wi-Fi y Ethernet
- Puertos USB 3.0
- Entrada para cámara web

La Raspberry Pi es compatible con dispositivos aceleradores de inferencia como el Movidius Neural Compute Stick de Intel o el acelerador TPU Coral USB de Google, lo cual permite el uso de aplicaciones de *deep learning*.

2.2.2 Google Coral TPU USB

El acelerador TPU por USB es uno de los productos de la gama Coral de Google enfocados al *Edge Computing*.

En este caso no se trata de una placa de desarrollo, aunque también está disponible, sino de una TPU capaz de conectarse a otros dispositivos mediante USB para acelerar la inferencia de modelos de *deep learning*.

Figura 5 - Acelerador TPU Coral USB.
https://coral.ai/products/accelerator/

El acelerador utiliza una Edge TPU y es compatible con la Raspberry Pi así como con ordenadores convencionales.

A la hora de conectarse a otros dispositivos es importante conectarlo utilizando un puerto USB 3.0. En caso de utilizar un puerto USB 2.0 este podría hacer de cuello de botella e impedir aprovechar al máximo la aceleración TPU.

A la hora de instalar la librería necesaria para su uso existen dos opciones: *Std* o *Max*. La opción Max permite usar el Coral USB en *overclock*, lo cual proporciona la máxima velocidad de cálculo, pero también un mayor estrés del procesador y una mayor temperatura.

A lo largo del desarrollo de este proyecto se ha usado siempre la versión *Std* por seguridad y de cara a una futura posible aplicación real en entornos donde la refrigeración pueda ser limitada. No obstante, una vez terminado el desarrollo se comparó la velocidad de cálculo de ambas versiones. Los resultados pueden verse en la parte final de la memoria.

2.2.2.1 TPU

La TPU (Tensor Processing Unit) es un ASIC (Application Specific Integrated Circuit), es decir, un circuito impreso diseñado para una aplicación específica, que en este caso se trata de realizar operaciones con tensores. Los tensores son una generalización de vectores de N dimensiones de datos, y son las unidades con las cuales trabajan las redes neuronales.

La mayoría de operaciones que suceden en las redes corresponden a operaciones básicas de multiplicación y suma. La peculiaridad de este tipo de aplicaciones es que deben llevarse a cabo millones de estas operaciones antes de obtener un resultado. Es por esto que las GPUs son mucho más eficientes a la hora de ejecutar redes neuronales que las CPUs.

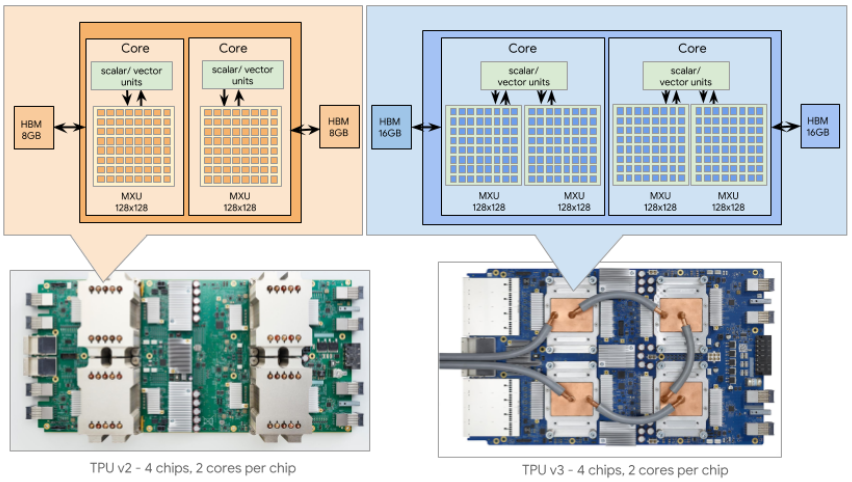

Figura 6 - TPU v2 y v3 desarrolladas por Google.
https://cloud.google.com/tpu/docs/system-architecture

Las TPUs van un paso más allá y eliminan el uso de memoria necesario tanto en CPUs como GPUs, lo cual reduce el tiempo de cálculo drásticamente. Para conseguir esto los

pesos de la red neuronal se cargan en cada una de las ALU (Arithmetic Logic Unit) y posteriormente se cargan los datos de entrada de la red neuronal para realizar las operaciones de multiplicación y suma.

Esta gran ventaja se consigue mediante una metodología llamada *systolic array*, la cual introduce los datos de entrada a la red de forma escalonada, evitando la necesidad de almacenar resultados intermedios.

Sin embargo, las TPUs son mucho menos flexibles que las GPU o las CPU y esto limita en cierta medida el tipo de operaciones que pueden realizar y, por tanto, el tipo de capas que puede tener una red neuronal que vaya a ser ejecutada en ellas.

2.2.2.2 Edge TPU

La Edge TPU es la versión más reducida de la TPU y es la que se encuentra en los productos de la serie Coral.

A pesar de su diminuto tamaño, la Edge TPU es capaz de realizar 4 $x10^{12}$ operaciones por segundo con un consumo de únicamente 2W.

Por supuesto esta restricción en el tamaño viene con mayores limitaciones a la hora de realizar cálculos, ya que la Edge TPU únicamente es capaz de acelerar operaciones hacia delante, por lo que sólo permite realizar inferencia y algunos casos de *transfer learning* ligero [1].

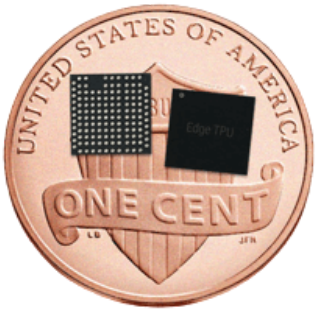

Figura 7 - Edge TPU incluida en la serie Coral.
https://coral.ai/docs/edgetpu/faq/

Además, los modelos que vayan a ser usados en la Edge TPU deben ser cuantizados, es decir, sus pesos deben reducirse de números con formato en coma flotante de 32 bits (*float32*) a enteros de 8 bits (*int8*) para acelerar los cálculos.

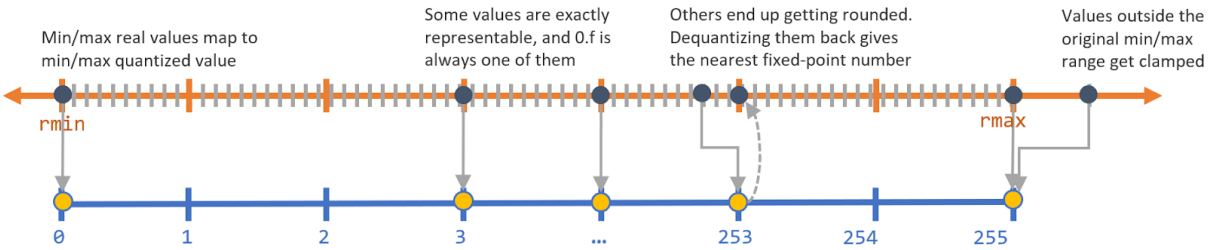

Figura 8 - Proceso de cuantización de los pesos de una red neuronal. https://heartbeat.fritz.ai/8-bit-quantization-and-tensorflow-lite-speeding-up-mobile-inference-with-low-precision-a882dfcafbbd

De esta manera, los pesos de la red original se escalan en el intervalo 0-255 y, si el valor no encaja exactamente con un entero, se redondea al más cercano, lo cual puede reducir la precisión del modelo de forma significativa si no se realizan los pasos indicados y recomendados para este tipo de conversiones.

2.3 Software

El software utilizado durante el proyecto se basa en Python y las librerías necesarias para realizar las operaciones y comandos necesarios para hacer funcionar el sistema de análisis de tráfico.

Python es un lenguaje de programación multiplataforma (Windows, Linux, Mac) con licencia de código abierto. En los últimos años se ha convertido en uno de los lenguajes de programación más populares e importantes debido a la gran cantidad de librerías disponibles para trabajar con *machine learning*, *big data*, *deep learning*, etc.

Sin embargo, a la hora de ejecutar el modelo existen diferencias entre hacerlo utilizando el acelerador TPU Coral USB (tanto en PC como en Raspberry Pi) u otros dispositivos como la placa de desarrollo NVIDIA Jetson Nano, puesto que para sacar el máximo partido de este y obtener el menor tiempo de ejecución posible debe ser optimizado en función de la plataforma en la que vaya a ser utilizado.

En el primer caso, el de este proyecto, se debeutilizar el entorno TensorflowLite, mientras que en el segundo se debería usar TensorRT desarrollado por NVIDIA.

2.3.1 Tensorflow Lite

Tensorflow es una librería de bajo nivel y código abierto diseñada para cálculo numérico de alto rendimiento. Como su propio nombre indica está pensada para trabajar con tensores, la unidad básica con la que trabajan los modelos de *deep learning*.

Su arquitectura permite la computación en los diferentes tipos de unidades de procesamiento: CPU, GPU y TPU.

Al tratarse de una librería de bajo nivel, a menudo se utiliza mediante Keras, una API de alto nivel escrita en Python y que utiliza Tensorflow para desarrollar modelos de *deep learning*. Es uno de los entornos más utilizados a la hora de desarrollar y entrenar redes neuronales.
Tensorflow Lite es una versión diseñada para utilizar los modelos desarrollados en Tensorflow en dispositivos como smartphones, microcontroladores, sistemas embebidos o relacionados con IoT (Internet of Things). Permite la optimización de modelos como redes neuronales para reducir su latencia y reducir su tamaño.

Para poder utilizar un modelo desarrollado en Tensorflow en el acelerador TPU Coral USB se deben seguir una serie de pasos que se muestran en la figura 9.

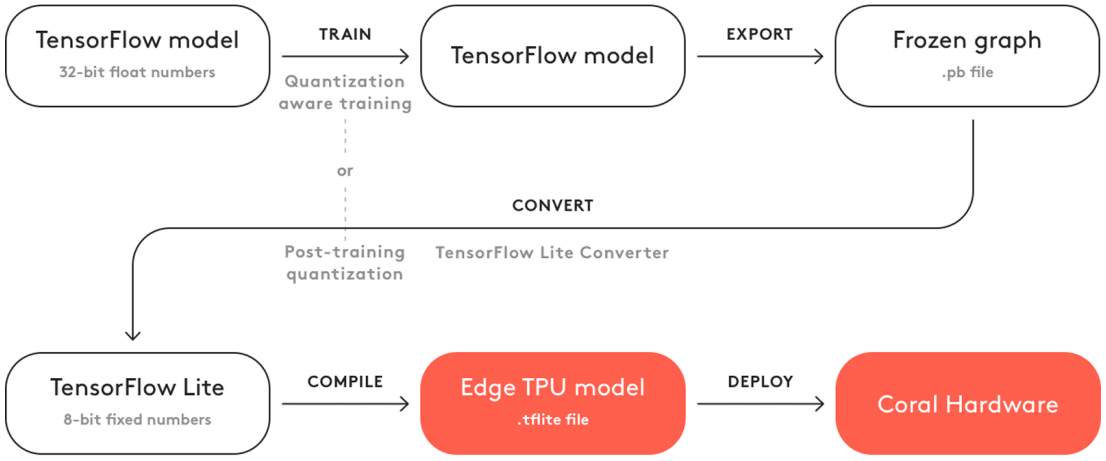

Figura 9 - Pasos a seguir para acelerar un modelo Tensorflow con una Edge TPU. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview

En primer lugar, se debe consultar la documentación tanto de Tensorflow Lite como de la Edge TPU para comprobar que todas las operaciones o capas del modelo serán soportadas por ambos.

Las operaciones no soportadas por Tensorflow Lite impedirán la conversión y las no soportadas por la Edge TPU Coral harán que todas las operaciones siguientes se ejecuten en la CPU lo cual impedirá la total aceleración por TPU como se ve en la figura 10.

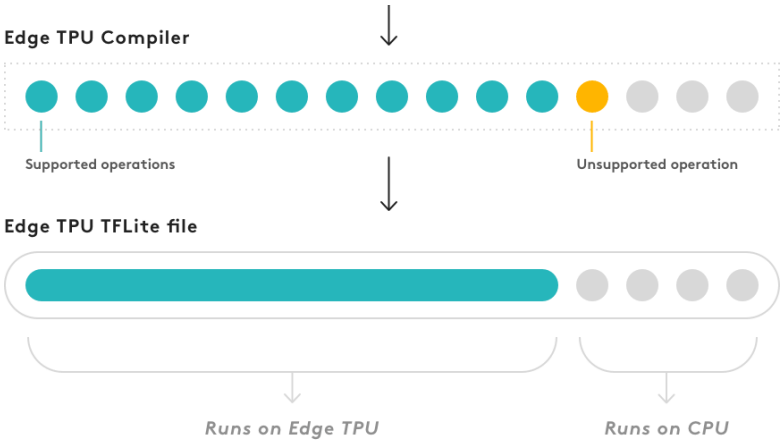

Figura 10 - Ejecución de operaciones no soportadas en Edge TPU. https://coral.ai/docs/edgetpu/models-intro/

Siguiendo las recomendaciones de la documentación de Tensorflow Lite y Coral, una vez comprobado que el modelo cumplirá los requisitos se entrenará de forma estándar utilizando la API de Keras. Esto se hace para que los pesos del modelo tras este primer entrenamiento, sirvan de punto de partida para volver a entrenar el modelo, pero esta vez *quantization aware*.

Es decir, en el segundo entrenamiento, el modelo es consciente de que sus pesos serán cuantizados, lo cual hará que la pérdida de precisión cuando se conviertan los pesos de 32 bits a 8 bits sea mínima. Sin embargo, es importante dejar claro que, a pesar de haber sido declarado *quantization aware*, los pesos del modelo siguen siendo de 32 bits hasta que se realiza la conversión a Tensorflow Lite, como indica el esquema de la figura 11.

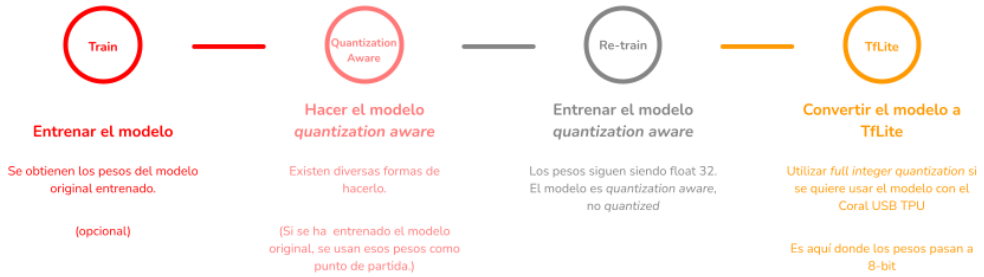

Figura 11 - Diagrama entrenamiento y cuantización modelo Tensorflow para Edge TPU. Elaboración propia

Una vez hecho esto sólo queda convertir el modelo a Tensorflow Lite. Los pesos serán cuantizados (si se desea) y se optimizará su tamaño. Es importante utilizar la versión 1.x de Tensorflow ya que utilizar la 2.x puede dar problemas de conversión.

Si se quiere utilizar este modelo en una Edge TPU, como es el caso de este proyecto, quedará un paso más que consiste en compilar el modelo .tflite para su uso en la TPU. Este paso únicamente puede realizarse en dispositivos Linux, pero se puede resolver de forma sencilla para los demás haciendo uso de un *notebook* de Google Colab.

Si se han seguido los pasos indicados la pérdida de precisión del modelo al cuantizar los pesos y optimizar el modelo será mínima.

Es posible que, por algún motivo, no pueda realizarse un entrenamiento *quantization aware*, como puede ser el caso de querer utilizar un modelo ya entrenado que no ha sido desarrollado por el usuario. En ese caso, es posible realizar una cuantización post-entrenamiento, la cual puede acarrear una mayor pérdida de precisión, aunque puede atenuarse si se dispone de un conjunto de datos representativo. La forma de proceder en este caso se describe en el diagrama de la figura 12, el cual sirve de guía para elegir la mejor optimización para cada caso concreto.

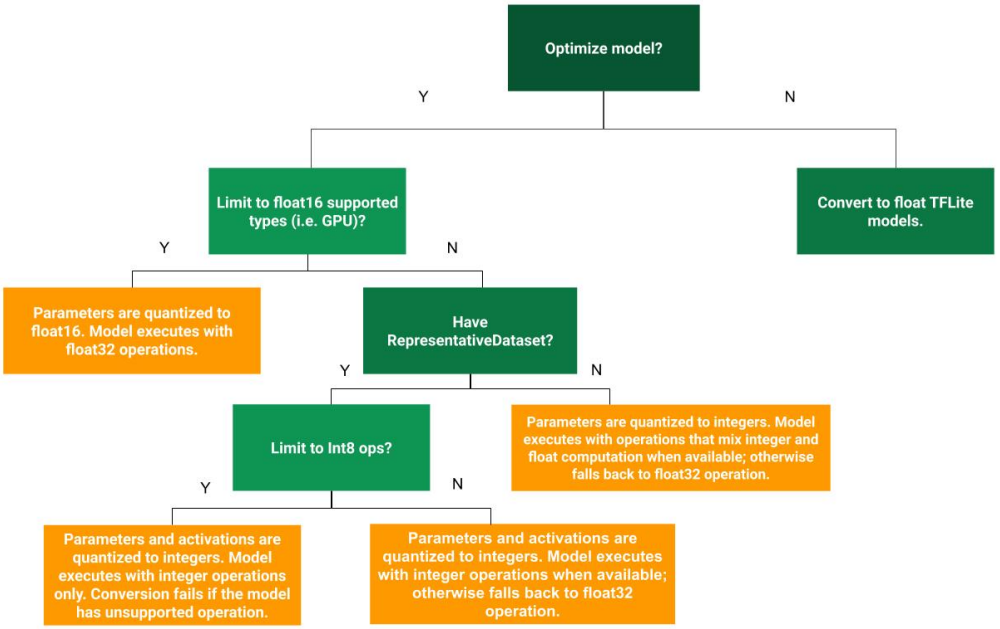

Figura 12 - Diagrama de decisión para cuantización post-entrenamiento.
https://www.tensorflow.org/lite/performance/post_training_quantization

2.4 Funcionamiento

2.4.1 Identificación y seguimiento

El sistema diseñado funciona de la siguiente manera: en primer lugar, una primera imagen se recoge a través de la cámara conectada al dispositivo que se esté utilizando, en este caso, la Raspberry Pi.

El usuario debe marcar dos puntos en la imagen que corresponderán a una línea perpendicular a la carretera.

Una vez dibujada esta línea, el sistema es completamente autónomo y comienza a funcionar.

Existen dos versiones de funcionamiento:

1. Utiliza únicamente el modelo de detección de objetos.
2. Alterna las detecciones del modelo con un *tracker*. Es el usuario quien especifica el número de *frames* que utiliza el *tracker*.

Esto es porque, en el caso de utilizar modelos más pesados, puede resultar interesante la alternancia entre la ejecución del modelo y un *tracker*. El tiempo de cálculo de este último puede ser menor y, por tanto, acelerar el cálculo o permitir el uso de modelos más complejos y precisos que de otra manera no serían capaces de ser ejecutados en tiempo real.

Un *tracker* es un algoritmo cuya función consiste en "seguir" a lo largo de una serie de imágenes algún objeto detectado en una imagen anterior. Esto se hace buscando ese primer objeto en las siguientes imágenes y muchos de estos *trackers* funcionan mediante el uso de filtros de correlación.

Se han probado 3 *trackers* distintos implementados en la librería OpenCV: MOSSE, CSRT y KCF, ya que son de los más populares.

El sistema seleccionado para este proyecto utiliza un método de seguimiento e identificación de los vehículos basado en el centroide.

Una vez se realizan las primeras detecciones, el modelo devuelve las coordenadas de la caja que delimita cada vehículo y se calcula el centroide de cada una. El centroide se calcula directamente como las coordenadas (x,y) del centro de las detecciones.

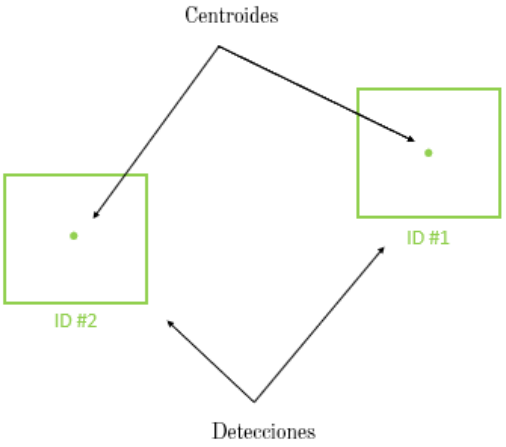

Figura 13 - Ilustración de las detecciones y sus centroides correspondientes. Elaboración propia

Si se combina la detección con *tracking*, las cajas que delimitan cada vehículo a veces vendrán dadas por el *tracker*.

Dado que estas han sido las primeras detecciones, se le asignará a cada una un identificador (ID) único.

Para cada *frame* nuevo se realiza una detección (o *tracking*) que proporciona, si los hay, nuevos centroides. Sin embargo, antes de añadir un ID nuevo a cada detección, primero se debe comprobar si es posible relacionar las nuevas detecciones (rojas) con algunas de las ya existentes (verdes) como se muestra en la figura 14.

Para esto, se calcula la distancia Euclídea entre cada pareja posible de nuevos centroides y los ya identificados anteriormente.

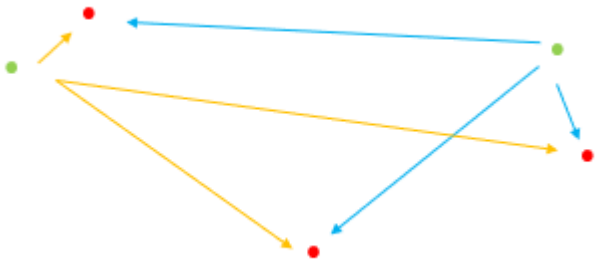

Figura 14 - Cálculo de la distancia entre nuevas detecciones y ya existentes. Elaboración propia.

Este tipo de identificación y seguimiento basado en el centroide asume que aunque el objeto detectado se mueva, la distancia entre su posición en dos *frames* consecutivos será siempre menor que la distancia con el resto de objetos detectados.

Figura 15 - Correspondencia entre las detecciones de dos frames consecutivos.
Elaboración propia.

De esta manera, se puede asociar las parejas de centroides cuya distancia Euclídea es mínima.

Si aparece un nuevo objeto que no puede ser asociado con ninguno de los anteriores, como en las figuras 14, 15 y 16, este se registra con un nuevo ID.

Figura 16 - Asignación de IDs final.
Elaboración propia.

Del mismo modo que se registran nuevos objetos cuando una detección no se puede relacionar con una de las ya identificadas, un objeto perdido o desaparecido debe eliminarse.

En este caso se ha definido que un objeto se elimina del registro cuando no se ha detectado en 10 *frames* seguidos.

2.4.2 Conteo de vehículos

Siendo capaces de identificar los vehículos detectados y seguirlos en la imagen utilizando el método del centroide, se debe definir la manera de contarlos.

En este punto se debe tener en cuenta que, en ocasiones, las detecciones del modelo pueden ser inestables por una infinidad de motivos:

- Vehículos muy lejanos difíciles de detectar
- Interferencias con otros vehículos u objetos
- Detecciones perdidas por imprecisión del modelo
- Otros

Esta cierta inestabilidad podría desembocar en identificar el mismo vehículo más de una vez a lo largo de su paso por la carretera. Es por esto que no se puede realizar el recuento de vehículos únicamente en base a su identificación.

Este es el motivo por el cual el usuario debe definir una línea perpendicular a la carretera, que servirá como una suerte de línea de meta. Para realizar el recuento de vehículos se tendrán en cuenta únicamente aquellos que crucen esta línea de meta.

Si una vez en la lejanía el vehículo se pierde y se reidentifica con un ID diferente, esto no supondrá un problema ya que no podrá volver a cruzar la línea.

La inestabilidad de las detecciones no se limita a casos en los que no se detecta el objeto de interés, sino que también puede ocurrir que la caja que delimita el vehículo detectado no sea perfecta, y por tanto, tampoco lo sea su centroide.

Esto hace que no se pueda asumir el histórico de los centroides de un objeto como su trayectoria, ya que puede que un vehículo moviéndose hacia la parte superior de la imagen dibuje en dos *frames* consecutivos un movimiento irreal hacia la parte inferior. Es este problema el que impide estimar la velocidad de los vehículos de una forma sencilla a pesar de aplicar transformaciones en la perspectiva de la imagen y/o suavizados en la trayectoria.

Para considerar un cruce por la línea definida por el usuario, a pesar de las imperfecciones de las detecciones, se utiliza el histórico de centroides de cada ID. Si la media de sus detecciones le sitúan a un lado de la meta, pero la última detección le sitúa al otro lado y a cierta distancia de la meta se clasifica este ID como detectado.

Esto impide que se pueda contar a este ID más de una vez y, si se descartara y volviera a detectar, sería irrelevante ya que no volvería a cruzar la meta.

2.4.3 Datos obtenidos

Una vez se detecta que un vehículo ha cruzado la línea, se guarda la información disponible sobre este en un archivo JSON (JavaScript Object Notation). El formato JSON es un formato de texto sencillo para el intercambio de datos cuyo uso está muy extendido, existen APIs para una gran variedad de lenguajes de programación que permiten su análisis y procesamiento.

De cada vehículo se guarda:

1. La clase a la que pertenece (coche, moto...)
2. La hora a la que cruza la línea
3. El sentido en el que circula.

Si se desarrollara un cálculo fiable se podría guardar la estimación de la velocidad a la que circula el vehículo, así como otros datos que pudieran ser relevantes para algún estudio como el color del vehículo, etc.

En ningún caso se pretende almacenar información que pudiera servir para identificar al vehículo o su conductor como podría ser su matrícula, ni imágenes del vehículo detectado. Se pretende así cumplir con las más estrictas normas de protección de datos a la vez que reducir el tamaño de los datos que deberán ser almacenados o enviados a otro sistema para su análisis.

Si se desea, es posible transferir los datos recogidos a otro dispositivo ya que la Raspberry Pi dispone de módulo WiFi. Esto puede ser un punto débil respecto a la seguridad, ya que un posible *hacker* podría conectarse a nuestro dispositivo y recoger imágenes de la cámara. Para evitar esto, se propone el uso de una conexión con un ancho de banda muy limitada, la cual impida la transferencia de vídeo o imágenes, pero sí permita el envío de los datos relacionados con el conteo de vehículos cuyo tamaño es mínimo.
Además, sería recomendable contactar con expertos en seguridad informática que dotaran al sistema de protección frente a posibles ataques.

A pesar de la sencillez de los datos recopilados, es posible realizar análisis muy interesantes como la cantidad y el tipo de vehículos que transitan una cierta carretera como los de la figura 17 o las horas, días y épocas en las que hay mayor tráfico. Incluso en el futuro podría utilizarse el sistema para detectar atascos y/o accidentes en tiempo real y reducir así el tiempo de reacción de los servicios de emergencia o de gestión del tráfico.

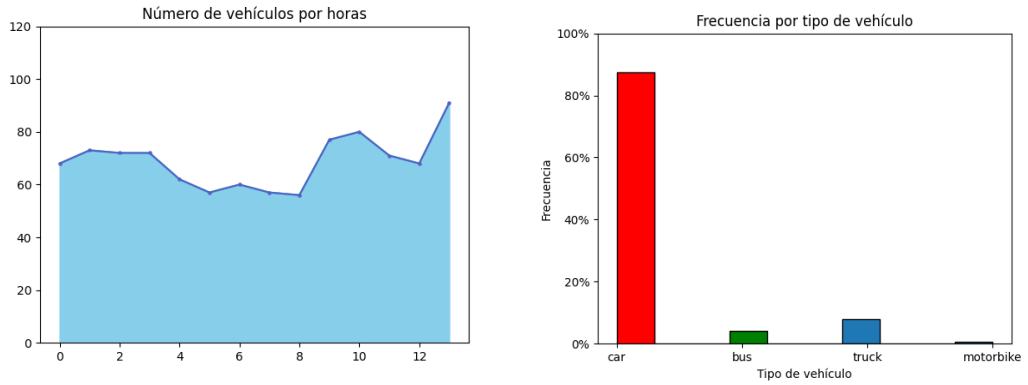

Figura 17 - Ejemplos de análisis sobre los datos obtenidos por el sistema. Elaboración propia.

3. Conjunto de datos

Uno de los objetivos de este proyecto es comprender todas las fases de un proyecto basado en *deep learning*, desde su planteamiento inicial pasando por la recogida de datos, la definición y entrenamiento del modelo hasta el procesamiento de los resultados. Por este motivo, no se utilizó ninguno de los conjuntos de datos disponibles, sino que se recopilaron, etiquetaron y procesaron desde cero. Algunos ejemplos se muestran en la figura 18. Esta parte del proyecto ocupó aproximadamente un tercio del tiempo dedicado al proyecto, a pesar de haberse recogido un número muy reducido de muestras.

El conjunto de datos original consta de 322 imágenes, con un total de 4.725 anotaciones de 4 clases distintas. Las clases son: Moto, Coche, Camión y Autobús, a las cuales les corresponden 1910, 1817, 579 y 419 anotaciones respectivamente.

Figura 18 - Ejemplos de muestras del conjunto de datos de entrenamiento. Elaboración propia.

La escasa cantidad de muestras y el desbalance entre las clases son dos problemas importantes que se deberían solucionar de cara al desarrollo de un producto final. Sin embargo, para el objetivo de este proyecto, y gracias al uso de métodos de *data augmentation*, se ha considerado suficiente en vista de los resultados obtenidos.

Durante el desarrollo del proyecto se es consciente de que el conjunto de datos empleado es quizá la debilidad más importante de este. Por esto es necesario hacer hincapié en la importancia de un conjunto de datos representativo donde se disponga de una cantidad suficiente de muestras en diferentes distancias y ángulos de cámara, condiciones meteorológicas, etc.

3.1 Recogida de datos

Para la recogida de datos se siguieron dos caminos:

1. Se descargaron diversos videos de la plataforma YouTube y se extrajeron fotogramas de ellos de forma separada para favorecer la diversidad de vehículos.

2. Se descargaron imágenes de Google Imágenes realizando búsquedas de vehículos en diferentes idiomas para aumentar la variedad de ángulos de la cámara e intentar compensar en cierta medida el desbalance de las clases.

La mayoría de los vídeos utilizados y, por tanto, de las muestras disponibles corresponden a situaciones diurnas y sin precipitaciones o condiciones meteorológicas adversas.

3.2 Etiquetado, formato y *data augmentation*

El etiquetado del conjunto de datos se realizó con la aplicación VoTT (Visual Object Tagging Tool) [2], la cual puede ser descargada desde su repositorio de GitHub y está disponible para los tres sistemas operativos principales.

Una vez se etiquetaron las imágenes que se consideraron suficientes para hacer una validación de la idea de este proyecto se utilizó la aplicación web Roboflow para darles el formato apropiado y realizar *data augmentation* de una manera sencilla.

Debido a los métodos seguidos para la recopilación de los datos de entrenamiento las resoluciones de las muestras etiquetadas eran muy diversas, por lo que se redimensionaron todas a un tamaño de 300x300 píxeles, ya que este es el tamaño de entrada del modelo que se eligió para el proyecto. Este tamaño se ha considerado adecuado ya que es relativamente ligero, lo cual acelera los cálculos a la vez que posee suficiente resolución para la detección de vehículos pequeños o lejanos en el tipo de imagen usado.

Para evitar deformaciones en la imagen o perder etiquetas al recortar partes de esta, se decidió hacer el cambio de tamaño añadiendo barras negras horizontales en la parte superior e inferior de las imágenes.

Debido a la escasez de datos etiquetados, se decidió hacer uso de tres técnicas de *data augmentation* de forma simultánea:

- **Variar la inclinación** de las muestras tanto de forma vertical como horizontal para hacer el modelo más robusto frente a variaciones en el ángulo de la cámara o la carretera.

- **Variar la exposición** de las muestras para hacer el modelo más robusto frente a cambios de iluminación.

- **Añadir** una pequeña cantidad de **ruido** para hacer el modelo más robusto frente a artefactos del vídeo o la cámara.

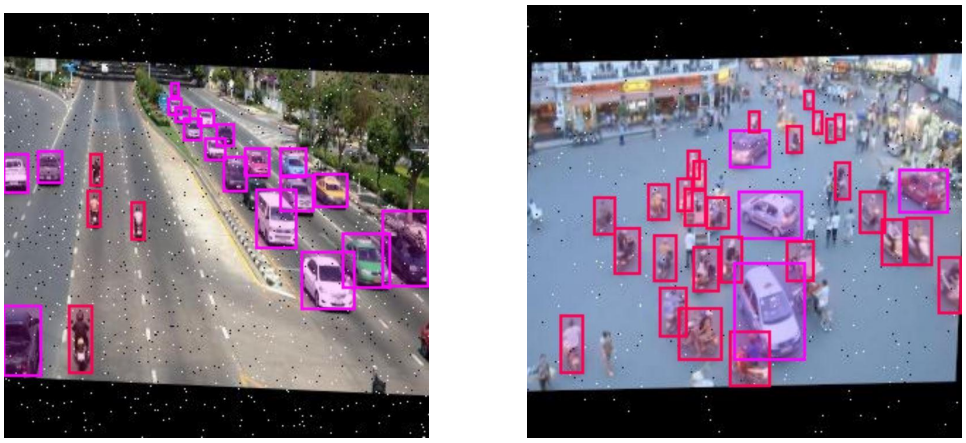

Figura 19 - Muestras de entrenamiento tras redimensionar y aplicar data augmentation. Elaboración propia.

De esta manera, el conjunto de datos final posee 846 imágenes, de las cuales un 70% se utilizó para el entrenamiento del modelo, un 15% como conjunto de validación y un 15% como conjunto de test.

4. Modelo

Debido a la naturaleza del proyecto y teniendo en cuenta que el objetivo de este es poder ejecutar un modelo en un dispositivo de bajo coste a tiempo real, se decidió utilizar un modelo que pudiera aportar unos resultados de alta precisión con baja latencia. Por esto se decidió utilizar un modelo MobileNet V2 SSD. Las características de este y los motivos que llevaron a su elección se detallan a continuación.

4.1 Algoritmo y arquitectura

A la hora de utilizar modelos de *deep learning* para tareas de detección de objetos en imágenes existen tres principales algoritmos que pueden utilizar diferentes arquitecturas para extraer información y características de las imágenes: Faster R-CNN, YOLO y SSD.

Para entender las diferencias entre ellos se debe conocer bien la tarea que se quiere desempeñar, en este caso la detección de objetos, que consiste en una combinación entre localización y clasificación de diferentes clases de objetos en una imagen.

Clasificación + localización

Esta tarea se puede enfocar como un problema de regresión, en el cual el modelo debe **clasificar y localizar**. A partir de una imagen de entrada, la red devuelve las coordenadas de la caja delimitadora (*bounding box)*, $\mathbf{b_x}$, $\mathbf{b_y}$, $\mathbf{b_h}$, $\mathbf{b_w}$, así como la clase del objeto detectado, como el ejemplo de la figura 20.

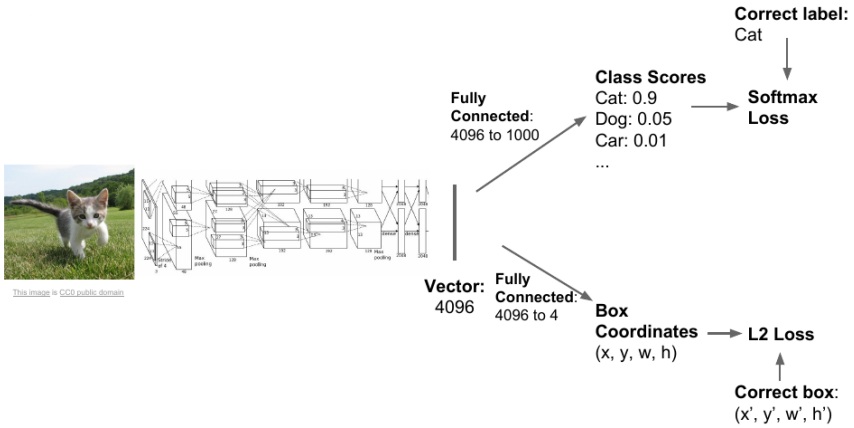

Figura 20 - Ejemplo del enfoque clasificación + localización.
https://kharshit.github.io/blog/2019/03/15/quick-intro-to-object-detection

El problema de este enfoque es que una misma imagen puede contener más de un objeto, por lo que cada imagen necesitaría un número diferente de salidas.

Sliding window

Otro método es el uso de una *sliding window (*ventana deslizante*)*, que enfoca el problema como una clasificación. La ventana se desplaza por la imagen y se pasa cada imagen recortada a una red convolucional que la clasifica como objeto o como fondo.

El problema con este enfoque es la necesidad de ejecutar una red convolucional a un gran número de ventanas de diverso tamaño y relación de aspecto para cada imagen de entrada, lo cual significa tiempos de cálculo mucho más elevados.

Region proposals

Este algoritmo funciona también en dos pasos: primero utiliza una búsqueda selectiva para proponer regiones de interés que pueden contener objetos, lo cual evita la ejecución de la red para todas las ventanas posibles como en el método anterior, y después realiza una clasificación sobre estas regiones.

En la actualidad, la versión más avanzada basada en *region proposal* es Faster R-CNN [3], la cual utiliza una red de proposición de regiones (RPN) para predecir las regiones de interés (ROI).

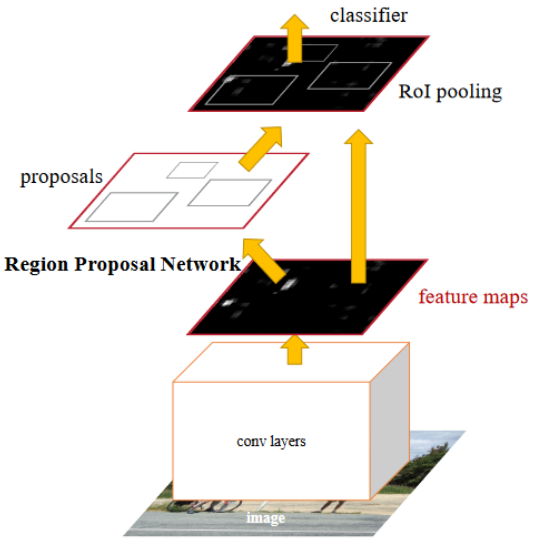

Figura 21 - Esquema Faster R-CNN.
https://arxiv.org/pdf/1506.01497.pdf [3]

YOLO (You Only Look Once)

El método YOLO [4] unifica la extracción de características, la localización y clasificación en un solo bloque, lo cual resulta en un tiempo de inferencia muy rápido.

La tercera versión, YOLOv3 [5] contiene conexiones salteadas (inspirado en la ResNet [6]) y tres salidas de predicción (inspirado en la FPN (Feature-Pyramid Network [7]), cada una procesando la imagen a diferente compresión espacial.

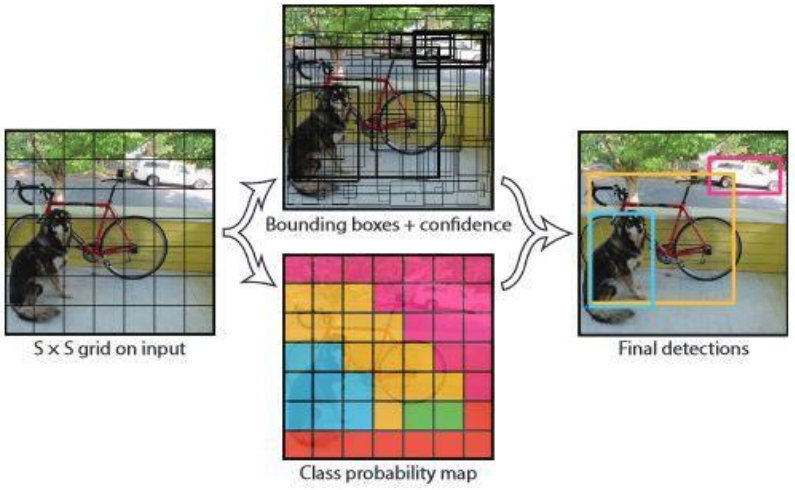

Figura 22 - Esquema funcionamiento YOLOv1.
https://arxiv.org/pdf/1506.02640.pdf [4]

A diferencia de los métodos anteriores, el modelo YOLO utiliza modelos *Fully CNN*, es decir, que solo emplean capas convolucionales, y enfoca la detección de objetos como un problema de regresión a cuadros delimitadores y probabilidades de clase asociadas.

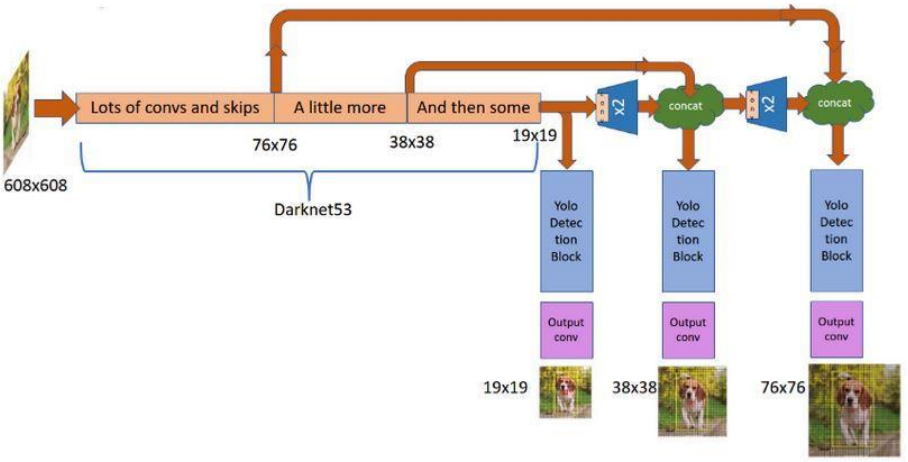

Figura 23 - Esquema funcionamiento YOLOv3.
https://towardsdatascience.com/yolo-v3-explained-ff5b850390f

La relativamente simple arquitectura del enfoque YOLO funciona gracias a una compleja función de pérdida, cuya explicación sobrepasa el alcance de este proyecto.

SSD (Single Shot Multibox Detector)

El algoritmo SSD [8], al igual que YOLO, permite la localización y clasificación de los objetos en un solo paso.

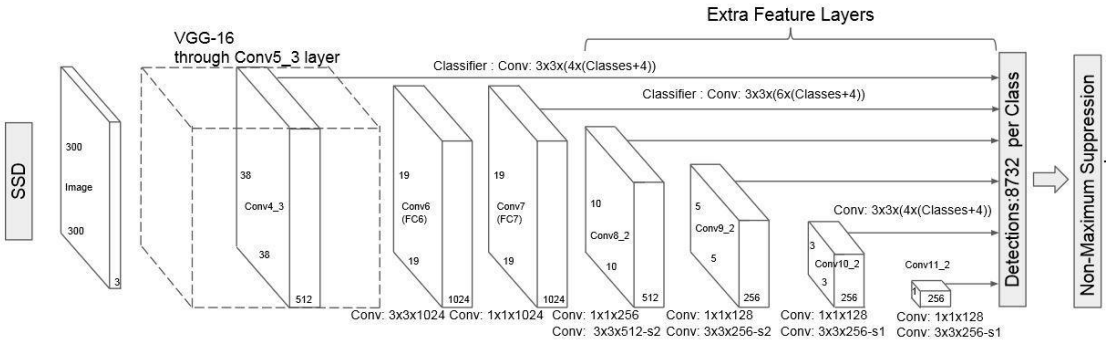

Figura 24 – Esquema de la estructura de un SSD.
https://arxiv.org/pdf/1512.02325.pdf [8]

Citando el artículo original (Liu, 2016, octubre): "Las SSD discretizan el espacio de salida de los cuadros delimitadores en un conjunto de rejillas predeterminados sobre diferentes relaciones de aspecto y escalas por ubicación de mapa de características. Durante la predicción, la red genera puntuaciones de la presencia de cada clase de objeto en cada rejilla predeterminada y realiza ajustes para adaptarse mejor a las dimensiones del objeto a detectar."

En la figura 25 se puede observar una comparación entre los diferentes algoritmos comentados realizada sobre los conjuntos de datos KITTI y Caltech para detectar carriles en carretera [9].

Algorithm	KITTI		Caltech	
	mAP (%)	Speed (ms)	mAP (%)	Speed (ms)
Fast RCNN	49.87	2271	53.13	2140
Faster RCNN	58.78	122	61.73	149
Sliding window & CNN	68.98	79,000	71.26	42,000
SSD	75.73	29.3	77.39	25.6
Context & RCNN	79.26	197	81.75	136
Yolo v1 ($S \times S$)	72.21	44.7	73.92	45.2
T-S Yolo v1 ($S \times 2S$)	74.67	45.1	75.69	45.4
Yolo v2 ($S \times S$)	81.64	59.1	82.81	58.5
T-S Yolo v2 ($S \times 2S$)	83.16	59.6	84.07	59.2
Yolo v3 ($S \times S$)	87.42	24.8	88.44	24.3
T-S Yolo v3 ($S \times 2S$)	88.39	25.2	89.32	24.7

Figura 25 - Comparación de precisión y tiempo de ejecución de los diferentes algoritmos.
https://www.mdpi.com/1424-8220/18/12/4308

Todos los algoritmos anteriores requieren de una red neuronal de base, la cual se encarga de extraer las características o *features* de las imágenes. Existe una gran cantidad de modelos diferentes como la ResNet, EfficientNet, o MobileNet.

Como ocurre a la hora de elegir algoritmo, no existe una arquitectura genuinamente mejor que el resto, y depende de diversos factores como el propósito del sistema o en qué dispositivo vaya a ser implementado.

Es por esto que para este proyecto, a pesar de que YOLOv3 parece tener mejores prestaciones, se decidió emplear la arquitectura MobileNetV2 [10] SSD. Mobilenet fue desarrollada por Google AI para ser usada específicamente en dispositivos móviles y de pocos recursos. Esta arquitectura ha sido ampliamente utilizada en este tipo de dispositivos y resulta sencillo encontrar implementaciones y documentación sobre ella.

Un factor decisivo es que se trata de un modelo de código libre disponible en la API de Tensorflow Object Detection. Esto facilitó mucho el entrenamiento y la conversión del modelo, así como el ajuste de hiperparámetros, lo cual encaja perfectamente con la filosofía del proyecto, buscando usar los elementos más sencillos y *open source* posibles para reducir la complejidad y el coste final del sistema al máximo.

Al igual que su predecesora, MobileNetV1 [11], la V2 utiliza convoluciones separables en profundidad como bloques de construcción eficientes. Sin embargo, la V2 introduce dos nuevas características a su arquitectura:

1. Cuellos de botella lineales entre las capas.
2. Conexiones de "atajo" entre los cuellos de botella.

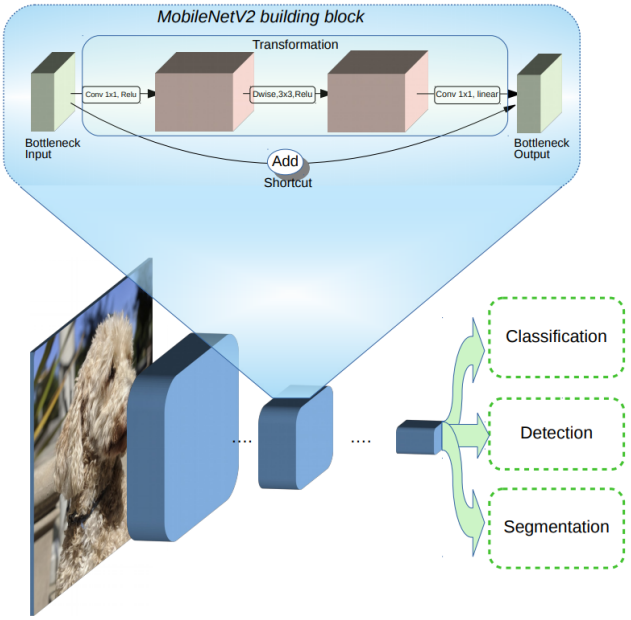

Figura 26 - Arquitectura de la MobileNetV2.
https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

Al modelo se le ha bautizado como EmbosNet, ya que la palabra "*embós*" significa atasco en valenciano.

5. Entrenamiento y resultados

5.1 Entrenamiento del modelo

Para el entrenamiento del modelo se utilizó *transfer learning* sobre el modelo MobileNetV2 cuantizado y preentrenado disponible en el repositorio de Tensorflow de Object Detection.

Se modificó el número de clases a las 4 necesarias para el proyecto, así como algunos parámetros del entrenamiento siendo los siguientes los empleados. El objetivo principal de los siguientes valores es que el modelo sea capaz de generalizar lo máximo posible teniendo en cuenta la escasez de muestras de entrenamiento. en el caso del *batch size*, fue imposible usar un número mayor por limitaciones de la GPU:

- Exponential decay learning rate: Initial = 0.004, decay factor = 0.95
- Optimizador *rmsprop*: Momentum = 0.9
- Batch Normalization = True
- Regularización L2: weight = 0.00004
- Batch size = 8

Se eligió reentrenar un modelo disponible en la API de Tensorflow Object Detection que ya estaba cuantizado ya que, al hacerlo de esta manera, se evita una pérdida de precisión posterior del modelo al cuantificar los pesos.

5.2 Métrica empleada

La métrica utilizada para evaluar el desempeño del modelo es la mAP (Mean Average Precision), ya que se ha convertido en un estándar a la hora de comparar el desempeño de modelos de detección de objetos. A pesar de su nombre, la mAP, no es la media de la precisión. Los modelos de detección de objetos dibujan una caja que delimita el objeto detectado y, para estudiar cómo de correctas son las predicciones de un modelo se comparan frente a las originales (*ground truth*). Esto se mide mediante la IoU (Intersection over Union), como ilustra la figura 27.

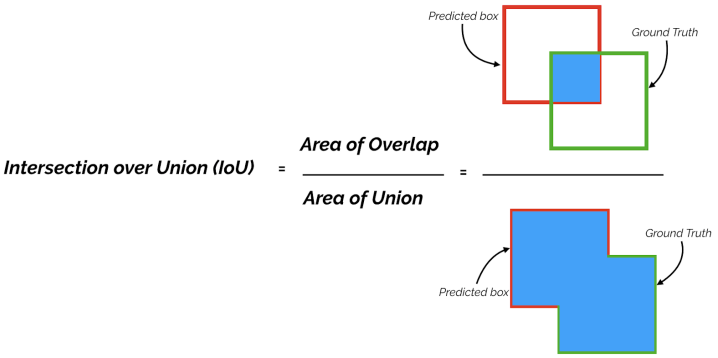

Figura 27 - Ilustración de la métrica IoU.
https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

En las tareas de detección de objetos se calcula la precisión y sensibilidad utilizando un cierto umbral de IoU. Por ejemplo, si el umbral de IoU se sitúa en 0,5 y el valor de una predicción es 0,7 se clasifica como correcta (TP). Si, por el contrario, la predicción es de 0,3 se clasifica como incorrecta (FP) como ilustra la figura 28.

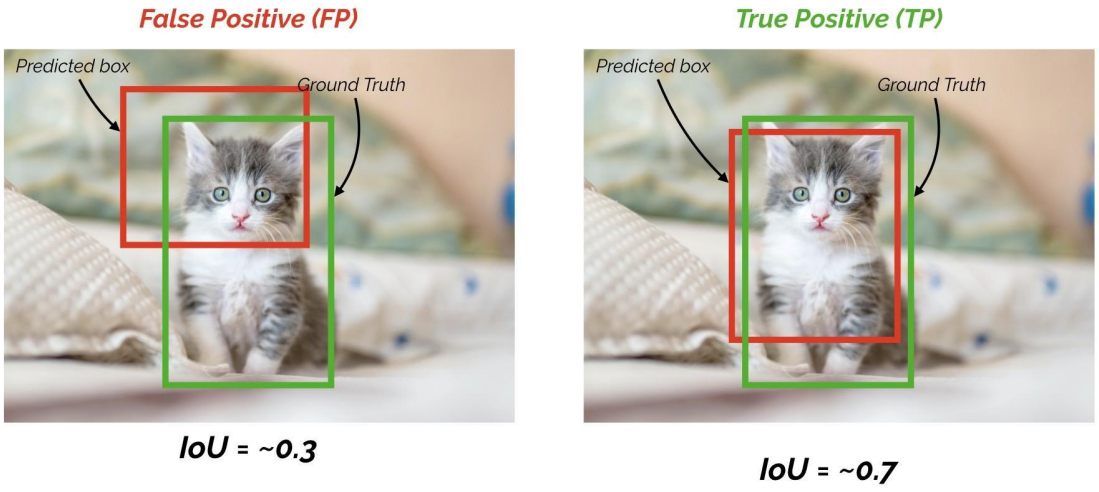

Figura 28 - Ejemplo de TP y FP con un umbral de IoU de 0,5.
https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

La precisión mide cuál es el porcentaje de predicciones correctas, es decir, cuántas de las predicciones del modelo son correctas:

$$Precisión = \frac{TP}{TP+FP}$$

TP = True Positive (Predicciones positivas correctas)

FP = False Positive (Predicciones positivas erróneas)

La precisión por sí sola no tiene en cuenta el número de objetos que el modelo no ha detectado.

La sensibilidad es otro término importante y explica cómo de bueno es el modelo encontrando todos los objetos:

$$Sensibilidad = \frac{TP}{TP+FN}$$

TP = True Positive (Predicciones positivas correctas)

FN = False Negatives (Objetos no detectados por el modelo)

Variando el umbral de IoU se obtienen diferentes parejas de precisión y sensibilidad, la unión de las cuales dibuja una curva ROC (Reciever Operating Characteristic) como las de la figura 29. Elegir un IoU concreto nos hará caer en un punto concreto de esas líneas.

El valor de F1 que se ve en la gráfica de la figura 29 corresponde a la métrica F1-Score, la cual corresponde a un valor ponderado de la sensibilidad y la precisión según la siguiente fórmula:

$$F1 - Score = 2 * \frac{precisión * sensibilidad}{precisión + sensibilidad}$$

Así, por lo general, se debe elegir un umbral de IoU que maximice el F1-Score para no sacrificar en exceso una de las dos y tener demasiados falsos positivos o falsos negativos.

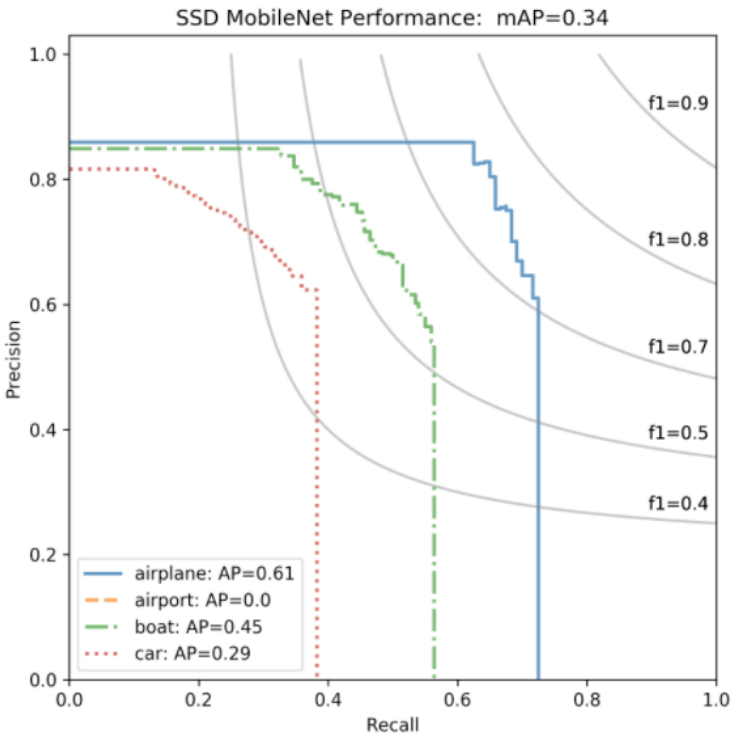

Figura 29 - Ejemplos de curvas precisión-sensibilidad.
https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

Por último, la métrica AP (*Average Precision*) representa el área bajo la curva ROC, y la mAP realiza un último promedio entre las diferentes clases de objetos a detectar por el modelo. En el caso de la figura 29 la clase *airplane* es la que tiene un mayor area bajo la curva y, por tanto mayor AP. Es decir, es la clase que mejor detecta el modelo. El promedio entre las distintas clases *airplane*, *airport*, *boat* y *car* da un resultado de mAP de 0.34.

5.3 Precisión y tiempos de ejecución

Los resultados finales del entrenamiento del modelo EmbosNet sobre los conjuntos de entrenamiento y test se muestran más abajo en la tabla 2. La notación IoU 0,5:0,95 significa que se han promediado los resultados de la mAP de los diferentes umbrales de IoU desde 0,5 hasta 0,95 y de las 4 clases que detecta el modelo.

Metric	IoU	Area	mAP (Train)	mAP (Test)
AP	0,5 : 0,95	All	0,351	0,329
AP	0,5 : 0,95	Small	0,301	0,282
AP	0,5 : 0,95	Medium	0,587	0,563

Tabla 2 - Resultados del modelo sobre los conjuntos de entrenamiento y test. Elaboración propia.

La clasificación de los objetos en áreas *Small* o *Medium* se hace siguiendo las métricas definidas en el dataset COCO [15], referente en el campo de *Object Detection:*

Small: Área $< 32^2$ píxeles

Medium: 32^2 píxeles $<$ Área $< 96^2$ píxeles

Los tiempos de ejecución varían en función de los dispositivos utilizados, así como del método utilizado, alternando detección y *tracker* o no, además del tipo de *tracker* utilizado. Algunos de estos factores afectan también a la precisión final del sistema.

En la tabla 3 se muestra una comparativa de la precisión y tiempos de ejecución de las diferentes combinaciones sobre un vídeo de tráfico en una autovía [12].En ella se puede observar la gran diferencia en los tiempos de ejecución gracias al uso del acelerador TPU Coral USB y cómo gracias a este se puede implementar el sistema en un dispositivo de bajos recursos y coste como la Raspberry Pi.

La precisión se ha evaluado como el porcentaje de los vehículos totales que aparecen en el video que es capaz de detectar el sistema y en lugar de mostrar tiempos de ejecución se muestran los *frames* por segundo (FPS) medios que es capaz de procesar el sistema tras analizar el vídeo completo.

A pesar de la reducida cantidad de datos utilizados en el entrenamiento del modelo, la sencillez del método de identificación y otros factores que podrían comprometer la efectividad del sistema, los resultados son suficientemente positivos para demostrar la validez del sistema y, por tanto, la propuesta del proyecto.

Se debe tener en cuenta que los tiempos de ejecución no se limitan al uso del modelo o modelo **+** *tracker*, sino al conjunto total del sistema con todos los cálculos, almacenamiento de datos, etc. que en él se realizan.[1]

Dispositivo	Sistema	NO TPU		TPU			Aceleración
		Precisión	**FPS**	**Precisión**	**FPS (std)**	**FPS (max)**	
PC	EmbosNet	98,51%	16,42	98,24%	**93,38**	**138.59**	**x5,7 - x8.4**
PC	EmbosNet + MOSSE	91,72%	**60,42**	91,63%	**123,91**	**166.53**	**x2,1 - x2.7**
PC	EmbosNet + CSRT	90,87%	9,93	90,87%	10,94	11.90	x1,1 - x1.2
PC	EmbosNet + KCF	92,17%	**39,30**	92,42%	**62,92**	**71.64**	**x1,6 - x1.8**
Raspberry Pi	EmbosNet	98,51%	4,03	98,24%	**29,36**	**41.17**	**x7,3 - x10.2**
Raspberry Pi	EmbosNet + MOSSE	91,72%	12,46	91,63%	**26,85**	**34.91**	**x2,2 - x2.8**

Tabla 3 - Precisión y velocidad de cálculo según los diferentes métodos y dispositivos. Elaboración propia.

De los resultados anteriores se pueden extraer diversas conclusiones. En primer lugar es posible destacar que el sistema es capaz de ser ejecutado en tiempo real (>25fps) gracias al acelerador TPU que multiplica la velocidad de cálculo del sistema en un factor de 7,3 en el caso de la Raspberry Pi sin usar *tracker*.

Esto demuestra que el acelerador por TPU es un componente esencial de este sistema si se quiere utilizar un dispositivo de bajos recursos. Incluso en el caso de utilizar un PC , el cálculo sin TPU del sistema sin uso de *trackers* no es lo suficientemente rápido para usarse en tiempo real.

[1] A la hora de realizar los cálculos de los tiempos de ejecución, el sistema realizaba también estimaciones de la velocidad de los vehículos que finalmente fueron descartadas por no ser lo suficientemente precisas.

También se puede comprobar cómo el uso de ciertos *trackers* menos pesados, como KCF y, sobre todo, MOSSE, puede resultar interesante en casos donde el modelo es mucho más pesado o no se dispone de aceleración por TPU.

Por el contrario, si se dispone de esta última, puede que su uso no sea tan interesante puesto que es un proceso que, por lo general, se ejecuta en la CPU y no se ve afectado por el acelerador TPU.

En cuanto a la precisión, el sistema ha detectado el mayor número de vehículos sin el uso de *trackers*, lo cual se podía esperar ya que se usan como alternativa durante un número de *frames* (5 en este caso) durante los cuales no se pueden detectar nuevos vehículos.

Otros enfoques como el uso de un tracker basado en las detecciones como SORT [13] o DeepSORT [14], o la combinación del modelo de detección junto con el *tracker* en cada ejecución podrían aumentar la robustez y precisión total del sistema.

6. Conclusiones

A modo de conclusión, es posible apuntar que se han cumplido los principales objetivos marcados al inicio de este proyecto. A lo largo de su desarrollo se han podido experimentar de primera mano todas las etapas de un proyecto basado en *deep learning*, así como conocer los retos y dificultades que guarda cada una de ellas.

A pesar de ser un sistema con gran margen de mejora, se ha conseguido desarrollar e implementar una primera versión capaz de analizar el tráfico utilizando dispositivos de pocos recursos y bajo coste gracias al uso de tecnologías de código abierto o *open-source*, siendo el coste total del dispositivo de alrededor de 200 €.

7. Trabajo futuro

Para finalizar, a lo largo de este trabajo han surgido muchas ideas y mejoras que han quedado fuera del alcance del proyecto y que deberían estudiarse seriamente si se quisiera perfeccionar el sistema y convertirlo en un producto final.

Una de los primeros caminos a seguir sería la recopilación y el uso de un conjunto de datos con mayor cantidad de muestras, mayor diversidad en cuanto a vehículos, meteorología, ángulos de cámara, etc. Cuanto mayor sea la diversidad de los datos mejor funcionará el modelo de detección y, por tanto, todo el sistema independientemente de la localización de la cámara, la hora del día y el tipo de vehículos.

Sería recomendable probar diferentes modelos compatibles con los dispositivos aceleradores del cálculo para extraer el máximo partido al hardware como los modelos EfficientNet, desarrollados con ese objetivo.

El hecho de implementar un detector de carril en la primera imagen del vídeo eliminaría la necesidad de una pantalla para que el usuario introdujera la línea de conteo de vehículos y, por tanto, ahorraría costes en hardware. Además, si se ejecutara cada cierto tiempo, podría hacer el sistema robusto frente a movimientos de la cámara.

En cuanto al sistema de identificación y seguimiento de los vehículos, podría combinarse el modelo de detección con el algoritmo DeepSORT, un *tracker* basado en *deep learning* que utiliza las detecciones de un modelo como punto de partida, o algún otro enfoque más sofisticado, siempre que se mantenga el rendimiento y el sistema pueda funcionar en tiempo real.

Por último, sería interesante continuar el estudio de la estimación de la velocidad de los vehículos detectados, quizá aplicando técnicas de *Optical Flow* u otros ya que las “basadas en píxeles” a partir de las detecciones o los centroides no han dado buenos resultados.

8. Bibliografía

[1] Recuperado de: https://coral.ai/docs/edgetpu/models-intro/#transfer-learning-on-device

[2] GitHub: https://github.com/microsoft/VoTT

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster R-CNN: Towards Real-Time ObjectDetection with Region Proposal Networks" (2016)

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection" (2016)

[5] Joseph RedmonAli Farhadi, "YOLOv3: An Incremental Improvement" (2018)

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition" (2015)

[7] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie, "Feature Pyramid Networks for Object Detection" (2017)

[8] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, "SSD: Single Shot MultiBox Detector" (2016)

[9] Xiang Zhang, Wei Yang, Ciaolin Tang, Jie Liu, "A Fast-Learning Method for Accurate and Robust Lane Detection Using Two-Stage Feature Extraction with YOLO v3", (2018)

[10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks" (2019)

[11] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" (2017)

[12] VK (3, jul, 2017) Relaxing highway traffic. Recuperado de: https://www.youtube.com/watch?v=nt3D26lrkho

[13] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft, "Simple Online and Realtime Tracking" (2017)

[14] Nicolai Wojke, Alex Bewley, Dietrich Paulus, "Simple Online and Realtime Tracking with a Deep association metric" (2017)

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, "Microsoft COCO: Common Objects in Context" (2015)