UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Realidad aumentada e Inteligencia Artificial en un entorno de Tactile Internet

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Flor Damiá, Jaime

Tutor/a: Palau Salvador, Carlos Enrique

CURSO ACADÉMICO: 2022/2023

Resumen

El objetivo de este Trabajo de Fin de Grado consiste en el diseño y desarrollo de un proyecto industrial de realidad aumentada e inteligencia artificial.

En concreto, el caso de uso del que este proyecto forma parte se centra en asistir a determinados usuarios (mecánicos, integradores de sensores, etc.) para localizar posibles fallos de sensores implementados en motores automovilísticos.

Para ello, los distintos elementos proclives de fallo se mapean con posiciones físicas de la zona del motor, de forma que, al diagnosticar un fallo, un usuario es dirigido al sensor problemático por medio de herramientas de visión artificial y elementos de realidad aumentada. Este proyecto se centra en tres partes: (i) la detección de objetos en *streams* de video, (ii) la representación de dichos objetos en interfaces gráficas de usuario (PC, dispositivos móviles, gafas de AR) y (iii) la comunicación entre las distintas interfaces y el servidor de procesamiento; todo ello, en tiempo real.

Las tecnologías principales que se emplean son TensorFlow, OpenCV y Android, entre otras varias. Entre los elementos de hardware usados para las pruebas cabe destacar el uso de gafas EPSON Moverio BT-350 y un servidor de altas prestaciones con una tarjeta gráfica NVIDIA A40.

Resum

L'objectiu d'aquest Treball de Fi de Grau consisteix en el disseny i desenvolupament d'un projecte industrial de realitat augmentada i intel·ligència artificial.

En concret, el cas d'ús del qual aquest projecte forma part se centra en assistir a determinats usuaris (mecànics, integradors de sensors, etc.) per a localitzar possibles fallades de sensors implementats en motors automobilístics.

Per a això, els diferents elements proclius de fallada es mapejen amb posicions físiques de la zona del motor, de manera que, en diagnosticar una fallada, un usuari és dirigit al sensor problemàtic per mitjà d'eines de visió artificial i elements de realitat augmentada. Aquest projecte es centra en tres parts: (i) la detecció d'objectes en *streams* de vídeo, (ii) la representació d'aquests objectes en interfícies gràfiques d'usuari (PC, dispositius mòbils, ulleres de AR) i (iii) la comunicació entre les diferents interfícies i el servidor de processament; tot això, en temps real.

Les tecnologies principals que s'empren són TensorFlow, OpenCV i Android, entre alters diverses. Entre els elements de hardware usat per a les proves cal destacar l'ús d'ulleres EPSON Moverio BT-350 i un servidor d'altes prestacions amb una targeta gràfica NVIDIA A40.

Abstract

The aim of this Final Degree Project is to design and develop an industrial project using augmented reality and artificial intelligence.

Specifically, the use case of which this project is part of focuses on assisting certain users (mechanics, sensor integrators, etc.) in locating possible failures of these sensors implemented in automotive engines.

To achieve this, the different elements prone to failure are mapped with physical positions of the engine area, so that when a failure is diagnosed, a user is directed to the problematic sensor through use of computer vision tools and augmented reality elements. This project focuses on three parts: (i) the detection of objects in video streams, (ii) the representation of these objects in graphical user interfaces (PC, mobile devices, AR glasses) and (iii) the communication between those different interfaces and the processing server, all in real time.

The main technologies used are TensorFlow, OpenCV, and Android, among several others. Notably, among the hardware elements used for testing, the use of EPSON Moverio BT-350 glasses and a high-performance server with an NVIDIA A40 graphics card stands out.

Índice general

Índice de figuras

Capítulo 1. Introducción

1.1 Motivación

En la actualidad, tanto la inteligencia artificial como la realidad aumentada son dos áreas que han demostrado un gran potencial en la transformación de múltiples sectores de la sociedad. Estas tecnologías han captado la atención de académicos, profesionales y empresas de todo el mundo debido a la capacidad de estas de mejorar la forma en la que interactuamos con el entorno y procesamos la información, siendo así de gran ayuda en múltiples ámbitos como pueden ser en la salud, los servicios financieros o el sector de la fabricación.

Aunque la inteligencia artificial ha estado presente en nuestra sociedad durante varios años, como puede ser con la funcionalidad de autocompletar de los teléfonos o aplicaciones de reconocimiento de voz, la aparición de modelos como *ChatGPT* ha generado un mayor interés en el usuario medio en torno a las capacidades de esta tecnología. Por otra parte, y aunque con menor repercusión, la realidad aumentada ha vuelto a estar de actualidad con la reciente salida a mercado de las *Apple Vision Pro*. El avance y desarrollo de estas tecnologías ha llamado la atención en el usuario medio, y como consecuencia, mayores recursos pueden ser captados por parte de las entidades de investigación y desarrollo para potenciar su evolución y adopción.

Durante mi periodo de prácticas en el grupo SATRD, ubicado en la Universidad Politécnica de Valencia, tuve la oportunidad de realizar el trabajo de fin de grado (TFG en adelante) relacionado con estas dos tecnologías, concretamente en el marco de trabajo del proyecto europeo ASSIST-IoT coordinado por el grupo.

La motivación principal de este proyecto es indagar en estas dos tecnologías en auge, para así desarrollar un proyecto en el que se integran de forma conjunta con la finalidad de facilitar labores específicas en entornos profesionales.

1.2 Objetivos

El objetivo de este proyecto es el desarrollo de una aplicación para gafas de realidad aumentada con la finalidad de, junto a la configuración de un servidor de altas capacidades, desplegar un sistema de detección de objetos mediante inteligencia artificial para que a la hora de inspeccionar la zona donde se encuentra el motor de un coche, localizar una serie de sensores que han sido instalados previamente. Para ello las gafas de realidad aumentada transmiten video capturado desde su cámara en tiempo real al servidor para posteriormente ser pasado por un modelo de detección de objetos. Una vez extraída la información de este proceso, el servidor enviará dicha información a las gafas para que estas representen la posición de estos objetos. Todo este proceso está enfocado con el principal objetivo de que sea en tiempo real.

Para alcanzar este objetivo se han establecido una serie de objetivos secundarios. Primeramente, se ha investigado en profundidad las tecnologías relacionadas con inteligencia artificial y realidad aumentada. La familiarización con el entorno de pruebas y el hardware disponible ha sido otro objetivo secundario importante, ya que gracias a ello se han podido plantear las distintas posibles soluciones que son factibles, para luego elegir la que consideramos más óptima y eficaz. Por último, la comprobación de la funcionalidad del sistema en general y la prueba en un entorno realista ha sido indispensable para proponer mejoras respecto a un sistema de detección de objetos y distintas aplicaciones posibles con esta tecnología.

1.3 Estructura Memoria

La memoria del proyecto está compuesta en 7 capítulos:

- *Capítulo 1:* Se expone la motivación principal a la hora de desarrollar este TFG y los objetivos, tanto principales como secundarios.

- *Capítulo 2:* Se presentan brevemente los conceptos principales sobre los que se desarrolla el TFG, para así tener un entendimiento más amplio.

- *Capítulo 3:* Se expone la metodología empleada para el desarrollo del proyecto.

- *Capítulo 4:* Se describe el entorno sobre el que se ha trabajado, así tanto como los componentes *hardware* y *software* usados durante el transcurso del proyecto.

- *Capítulo 5:* Se indaga en profundidad en el desarrollo de las diferentes partes del proyecto.

- *Capítulo 6:* Se presentan los resultados de las pruebas realizadas una vez se ha desarrollado y desplegado el sistema.

- *Capítulo 7:* Se presentan las conclusiones obtenidas del desarrollo del TFG y posibles líneas futuras de mejora.

Capítulo 2. Fundamentos teóricos

2.1 Inteligencia artificial y aprendizaje automático.

La inteligencia artificial **[1]** (en adelante IA, por sus siglas de referencia) se basa en el principio de que la inteligencia humana puede ser definida de tal manera que una máquina pueda imitarla fácilmente y llevar a cabo tareas, desde las más simples hasta aquellas más complejas. Los objetivos de la IA incluyen imitar la actividad cognitiva humana. Los investigadores y desarrolladores en este campo están logrando avances sorprendentemente rápidos en la imitación de actividades como el aprendizaje, el razonamiento y la percepción, hasta el punto en que éstas pueden ser definidas concretamente. Algunos creen que los innovadores pronto podrán desarrollar sistemas que superen la capacidad de los seres humanos para aprender o razonar sobre cualquier tema. Sin embargo, otros se muestran escépticos porque toda actividad cognitiva está impregnada de juicios de valor que están sujetos a la experiencia humana.

Indudablemente, la IA ha experimentado un rápido crecimiento y avance en las últimas décadas, ya que este campo se ha visto impulsado por los avances en la capacidad de cálculo y procesamiento, el almacenamiento de datos y los algoritmos de aprendizaje automático. Éste último va a ser el centro de atención, ya que aparte de ser el área más destacada de la IA en los últimos años, es una de las varías tecnologías que han sido utilizadas para el desarrollo de este trabajo.

El aprendizaje automático **[2]** (*Machine Learning*) es el concepto de que un programa informático puede aprender y adaptarse a nuevos datos sin intervención humana. Los algoritmos de *Machine Learning* recurren a métodos computacionales para "aprender" información directamente a partir de datos, sin depender de una ecuación predeterminada como modelo. Los algoritmos mejoran su rendimiento de manera adaptativa a medida que aumenta el número de muestras disponibles para el aprendizaje.

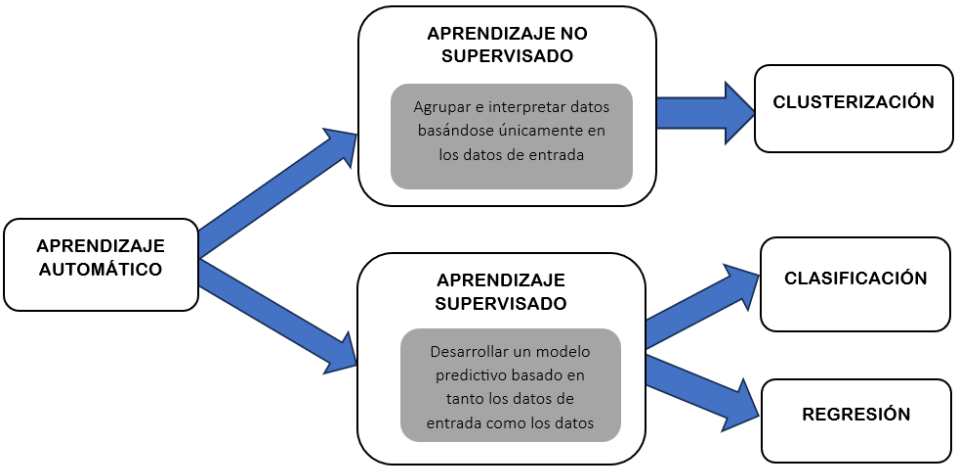

Figura 2.1: Métodos de aprendizaje

Tal y como se muestra en la Figura 2.1, existen dos tipos de aprendizaje automático, el aprendizaje supervisado y el no supervisado:

- **Aprendizaje supervisado**: Toma un conjunto conocido de datos de entrada y respuestas conocidas sobre esos datos (salidas), y entrena un modelo para generar predicciones razonables como respuesta a datos nuevos. Emplea técnicas de clasificación y regresión para desarrollar modelos.

- **Aprendizaje no supervisado**: Identifica patrones ocultos o estructuras intrínsecas en los datos. Se emplea para sacar conclusiones sobre conjuntos de datos de entrada sin respuestas etiquetadas.

Para el desarrollo de este trabajo se ha elegido utilizar el aprendizaje supervisado.

2.2 Detección de objetos en imágenes.

La detección de objetos **[3]** es una técnica de visión artificial que permite localizar instancias de objetos en imágenes o videos. Los algoritmos de detección de objetos suelen aprovechar el aprendizaje automático o el aprendizaje profundo para obtener resultados significativos. Cuando los humanos observamos imágenes o videos, podemos reconocer y localizar objetos de interés en cuestión de momentos. El objetivo de la detección de objetos es replicar esta capacidad de inteligencia utilizando un ordenador.

La detección de objetos lleva ya un tiempo considerable instaurada en nuestra sociedad. Un ejemplo claro de esto es dentro del campo de la vigilancia y seguridad, ya que esta tecnología se usa para identificar y rastrear personas, vehículos y objetos sospechosos en áreas públicas, aeropuertos, centros comerciales y otros lugares donde se requiere una supervisión constante. Otro ejemplo pueden ser las ciudades inteligentes, como es Tokio, en la cual la detección de objetos se utiliza para aplicaciones como el monitoreo de tráfico, la gestión del estacionamiento, identificar y rastrear vehículos en tiempo real u optimizar el flujo de tráfico. Esto se puede observar en la Figura 2.2.

Figura 2.2: Algoritmo de detección de objetos

Es posible acotar entre dos enfoques clave para comenzar con la detección de objetos a la hora de utilizar algoritmos de aprendizaje automático:

- **Crear y entrenar un detector de objetos personalizado**: Su funcionamiento principal se basa en entrenar un detector de objetos personalizado desde cero. Se debe diseñar una arquitectura de red para aprender las características de los objetos de interés. También es necesario recopilar un conjunto muy grande de datos etiquetados para entrenar la red neuronal convolucional (*CNN*, por sus siglas en inglés). Los resultados de un detector de objetos personalizado pueden ser notables. Sin embargo, se debe configurar manualmente las capas y los pesos en la *CNN*, lo cual requiere mucho tiempo y datos de entrenamiento.
- **Utilizar un detector de objetos pre-entrenado**: Muchos flujos de trabajo de detección de objetos utilizan el aprendizaje por transferencia, un enfoque que permite comenzar con una red entrenada previamente para luego ajustarla para tu aplicación. Este método puede proporcionar resultados más rápidos porque los detectores de objetos ya han sido entrenados en miles, e incluso millones, de imágenes.

Durante el desarrollo de este TFG, se ha hecho uso del segundo enfoque, ya que, a partir de un modelo pre-entrenado, ajustarlo para que así detectara los objetos de interés. Esto mismo lo veremos en el Capítulo 5 en profundidad.

2.3 Realidad aumentada y su aplicación en detección de objetos

La realidad aumentada **[5]** es una experiencia interactiva que mejora el mundo real con información perceptual generada por ordenador. Utilizando *software*, aplicaciones y *hardware* como pueden ser gafas de realidad aumentada, la realidad aumentada superpone contenido digital en entornos y objetos de la vida real. Esto enriquece la experiencia del usuario y convierte el entorno circundante en un entorno de aprendizaje interactivo, lo cual es especialmente valioso en los procesos de fabricación e Industria 4.0. Esta tecnología permite a los usuarios industriales "fusionarse" con los sistemas y máquinas con las que trabajan, para así optimizar y mejorar las tecnologías y redes de IoT con la ingeniosidad humana, la observación y la creatividad.

Al hablar de la realidad aumentada pueden surgir confusiones causadas por otras tecnologías desarrolladas en nuestra sociedad, como puede ser la *realidad virtual.* Para aclarar esto, se van a exponer una serie de diferencias que distinguen las diversas aproximaciones a la hora de interactuar con el mundo real:

- **Realidad Virtual**: La realidad virtual separa a las personas del mundo real y las sumerge por completo en un mundo virtual utilizando un visor o casco de visualización. En ese mundo virtual de imágenes y sonidos, los usuarios pueden moverse en todas las direcciones, manipular objetos y más. La realidad virtual se utiliza con frecuencia en el campo de la salud, arquitectura y educación.

- **Realidad Aumentada**: La realidad aumentada mejora o aumenta el mundo real con información digital. Si bien las aplicaciones de realidad aumentada funcionan a través de dispositivos móviles como teléfonos inteligentes o tabletas, en entornos de fabricación e industriales donde es beneficioso para el usuario tener las manos libres, las gafas o auriculares son las mejores opciones para experimentar la realidad aumentada.

Cuando se piensa en realidad aumentada, uno de los elementos clave a considerar es la tecnología de reconocimiento de objetos. Como se ha explicado antes, esta tecnología se refiere a la capacidad de identificar la forma y el contorno de diferentes objetos, así como su posición en el espacio capturados por la cámara del dispositivo. Dicho esto, la realidad aumentada pretende conseguir una mejora de la vista del mundo real con superposiciones de gráficos por computadora, texto, videos o sonidos. En todas las aplicaciones de realidad aumentada, el reconocimiento de objetos es especialmente importante. **[4]**

Figura 2.3: Realidad Aumentada y Realidad Virtual

Capítulo 3. Metodología y desarrollo

3.1 Metodología

Para la realización de ese Trabajo de Fin de Grado se siguió una metodología denominada *Scrum*. *Scrum* **[15]** es un marco de gestión de proyectos ágil que ayuda a los equipos a estructurar y gestionar su trabajo a través de un conjunto de valores, principios y prácticas. Esta metodología fomenta que los equipos aprendan a través de experiencias, se autoorganicen mientras trabajan en un problema y reflexionen sobre sus éxitos y fracasos para mejorar continuamente.

3.2 Planificación y desarrollo

Al comienzo del desarrollo de este trabajo fue necesario establecer una serie de tareas iniciales al mismo tiempo que se definía el entorno sobre el cuál se iba a trabajar. El objetivo era, una vez concretada la tarea inicial de la que se iba a partir, proceder con la realización de dichas actividades en orden cronológico para la correcta evolución de este proyecto. En concreto, se realizaron las siguientes actividades:

1. Estudio y familiarización de la tecnología de IA a usar.
2. Familiarización con el entorno de pruebas y el *hardware*.
3. Evaluación de las distintas aproximaciones a la hora de desarrollar el proyecto y elección de estas.
4. Desarrollo del trabajo.
5. Pruebas de campo
6. Redacción de la memoria

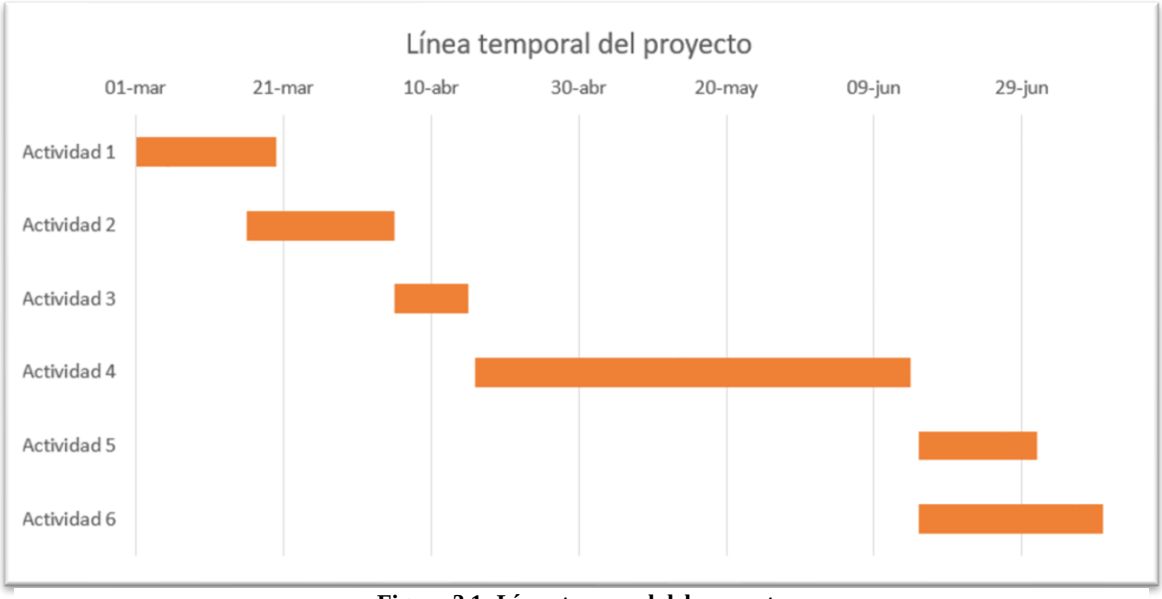

Figura 3.1: Línea temporal del proyecto

Capítulo 4. Entorno de pruebas

En este capítulo se van a describir los distintos componentes, tanto *hardware* como *software*, que han sido utilizados durante el desarrollo del proyecto para así, gracias a los componentes listados, poder realizar un caso de uso que cubra todos los objetivos y subobjetivos principales de este TFG.

4.1 Hardware

4.1.1 Gafas Epson

Las gafas de realidad aumentada EPSON MOVERIO BT-350 **[16]** son un producto desarrollado por la empresa EPSON el cual salió a la venta en el año 2017. Estas gafas vienen con un sistema operativo llamado Moverio OS, basado en un Android 5.1. Las gafas no llevan incorporadas en sí mismas este sistema operativo, si no que vienen junto a un controlador con *hardware* propio donde el sistema operativo va instalado. Las gafas van conectadas al controlador mediante un cable y estas actúan como *display* del dispositivo.

A continuación, se van a presentar las características más relevantes de este producto:

1. Aspecto
    - Tipo de dispositivo de visualización: Si-OLED
    - Tamaño de pantalla: 0,43 pulgada panel ancho (16:9)
    - Número de píxeles: 921.600 (1280x720) RGB
    - Campo de visión: aprox. 23º
    - Distancia de proyección: 80 pulgadas en 5m – 320 pulgadas en 20m
    - Frecuencia de actualización: 30 Hz
2. CPU y Memoria
    - CPU: Intel® Atom™ x5, 1.44GHz Quad Core
    - RAM: 2GB
    - Memoria Interna: 16GB
3. Conectividad del controlador
    - Puertos: 1 x microUSB 2.0
    - LAN inalámbrica: IEEE 802.11a/b/g/n/ac
    - Bluetooth

Figura 4.1: Gafas EPSON Moverio BT-350

4.1.2 Servidor *Supermicro GPU SuperServer SYS-420GP-TNR*

Para el procesado de imágenes y posterior inferencia mediante el modelo de IA, se usó un servidor alojado en la Ciudad Politécnica de la Innovación (CPI) con el que, gracias a su gran potencia, facilitó de manera notable todas las tareas relacionadas con IA permitiendo así alcanzar resultados de forma más eficiente y efectiva.

Se van a exponer las principales características del servidor al completo, y más adelante se entrará en detalle con las características de las tarjetas gráficas, ya que son el componente más importante de este para el proyecto:

- 1 TB de memoria RAM DDR4
- 104 x Intel(R) Xeon(R) Gold 5320 CPU @ 2.20GHz
- 8 x Tarjetas gráficas NVIDIA A40
- 1 TB de almacenamiento

Figura 4.3: Servidor

Figura 4.2: Servidor

En lo que respecta a las gráficas, éstas son el modelo A40 de la compañía NVIDIA. La NVIDIA A40 **[6]** acelera las cargas de trabajo de computación visual más exigentes desde el centro de datos, combinando la última arquitectura NVIDIA *Ampere* con *RT Cores*, *Tensor Cores* y *CUDA Cores*. Desde potentes estaciones de trabajo virtuales accesibles desde cualquier lugar hasta nodos de renderización dedicados, las gráficas NVIDIA A40 llevan la tecnología NVIDIA RTX de próxima generación al centro de datos para las cargas de trabajo de visualización profesional más avanzadas.

Respecto a las características de la NVIDIA A40:

- Memoria de la GPU: 48 GB DDR6
- Ancho de banda de la memoria: 696 GB/s
- Interfaz de conexión: PCIe Gen4 de 64 GB/s
- Núcleos CUDA: 10,752
- Núcleos RT: 84
- Núcleos Tensor: 336
- Rendimiento pico en FP32: 37.4 TFLOPS
- Consumo máximo de potencia: 300 W

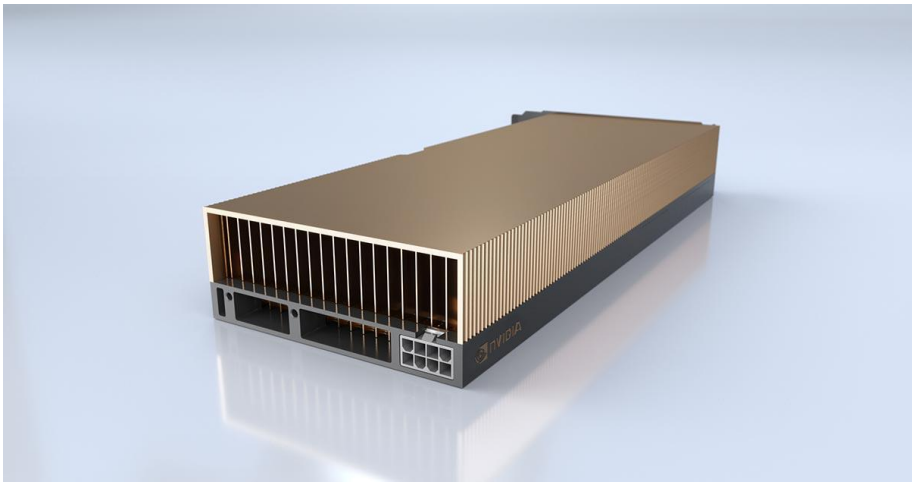

Figura 4.4: Tarjeta gráfica NVIDIA A40

4.2 Software

4.2.1 Proxmox

Proxmox VE **[7]** (*Proxmox Virtual Environment*) es una solución de gestión de virtualización de servidores de código abierto que permite a los usuarios administrar máquinas virtuales, como máquinas Windows o Linux, y contenedores de Linux. Está basado en la distribución de Linux Debian y combina dos tecnologías de virtualización, KVM (*Kernel-based Virtual Machine*) y LXC (*Linux Containers*), que se gestionan a través de una interfaz basada en web.

El servidor, debido a la cantidad de *hardware* que contiene y la potencia de este, está específicamente diseñado para ser utilizado por múltiples usuarios simultáneamente, ofreciéndoles así la posibilidad de aprovechar al máximo su rendimiento y potencial. Gracias a Proxmox se pueden crear diversas máquinas virtuales individuales a las que se les pueden asignar más o menos recursos, dependiendo del uso y la carga de trabajo que se le vaya a dar a esa máquina virtual en específico. De esta manera, se puede gestionar el *hardware* del servidor de una forma óptima y eficiente.

A la máquina virtual en la que se trabajó durante este proyecto se le asignó una tarjeta gráfica dedicada para realizar tanto el entrenamiento como la inferencia. En la Figura 4.5 se muestran las características específicas de la máquina con la que se trabajó:

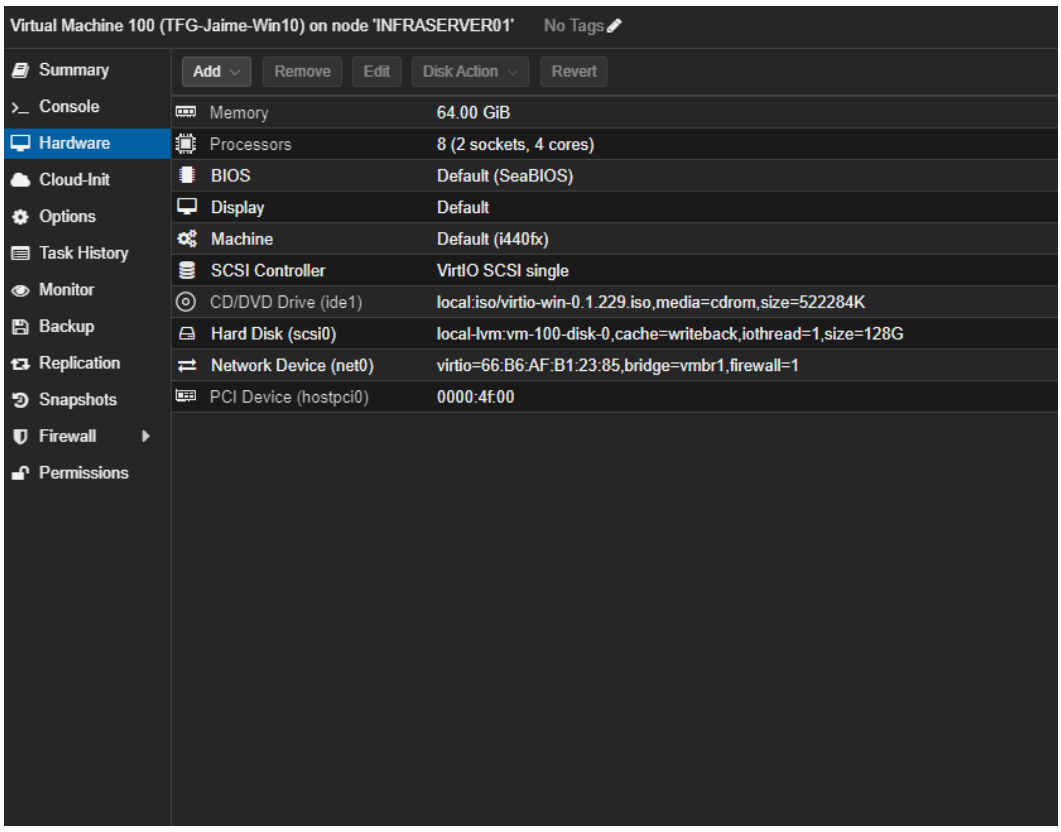

Figura 4.5: Especificaciones máquina virtual

4.2.2 Android Studio

Android Studio **[17]** es un entorno de desarrollo integrado (*IDE*, por sus siglas en inglés) con la particularidad de que está completamente enfocado a la plataforma Android. Está desarrollado por Google y facilita completamente el desarrollo y pruebas de aplicaciones dirigidas a dispositivos Android, ya que cuenta con una serie de herramientas como pueden ser el *Layout Editor* que permite crear tu diseño visual de la aplicación usando *XML (Extensible Markup Language*, lenguaje diseñado con el objetivo de almacenar y transportar información) a la par que lo puedes ver en directo mientras lo creas. Otra herramienta muy útil de este software es que proporciona un emulador de Android con el cual puedes crear dispositivos móviles emulados en el ordenador para probar las aplicaciones que vayas desarrollando, sin la necesidad de tener que disponer de un dispositivo Android físico en el que hacer estas pruebas.

Figura 4.6: Android Studio

La razón por la cual se ha decidido utilizar este software se debe a que las gafas de realidad aumentada cuentan con el sistema operativo Android 5.1, lo cual requería la necesidad de acceder y gestionar la cámara, así como manejar todas las comunicaciones requeridas con el servidor. Para asegurar una interacción cómoda y óptima con este dispositivo, se ha considerado que la mejor solución era desarrollar una aplicación específica para Android que permita trabajar de manera eficiente con todas estas funcionalidades.

4.2.3 Tensorflow

TensorFlow **[8]** es una plataforma *end-to-end* de código abierto para el aprendizaje automático. Una plataforma *end-to-end* es una plataforma de software que proporciona un conjunto completo de herramientas y servicios para construir, implementar y gestionar aplicaciones de principio a fin.

Cuenta con un ecosistema completo y flexible de herramientas, bibliotecas y recursos comunitarios que permiten a los investigadores llevar el estado del arte en ML (*Machine Learning*) y a los desarrolladores crear y desplegar fácilmente aplicaciones impulsadas por ML.

TensorFlow fue desarrollado originalmente para realizar cálculos numéricos de gran escala sin tener en cuenta específicamente el aprendizaje profundo. Sin embargo, resultó ser muy útil también para el desarrollo de aprendizaje profundo, por lo que Google lo "liberó" como software de código abierto. La plataforma acepta datos en forma de arreglos multidimensionales de dimensiones superiores llamados tensores. Los arreglos multidimensionales son muy útiles para manejar grandes cantidades de datos. **[9]**

Figura 4.7: TensorFlow

Como el objetivo principal a la hora de usar un *framework* era la detección de objetos, TensorFlow fue la mejor opción por diversas razones.

TensorFlow ofrece APIs tanto en C++ como en Python. Antes del desarrollo de bibliotecas, el mecanismo de codificación para el aprendizaje automático y el aprendizaje profundo era mucho más complicado. Esta plataforma proporciona APIs de alto nivel y no se necesita una codificación compleja para configurar o programar redes neuronales **[9]**. Debido a que el proyecto no se trataba de indagar en la tecnología de la IA si no que el objetivo era implementar esta de forma sencilla, TensorFlow ofrecía la posibilidad de desarrollar de manera simple, junto a Python y la API de detección de objetos, diversos programas para entrenar modelos y realizar la inferencia en tiempo real con las gafas.

TensorFlow admite tanto CPUs como GPUs. Las aplicaciones de aprendizaje profundo son muy complejas, y el proceso de entrenamiento requiere una gran cantidad de cálculos. Esto lleva mucho tiempo debido al gran tamaño de los datos e implica varios procesos iterativos, cálculos matemáticos, multiplicaciones de matrices, entre otros. Si se realizaran estas actividades en una CPU normal, por lo general, llevaría mucho más tiempo. Las GPUs son populares en el contexto de los juegos, donde se necesita que la pantalla y la imagen tengan una alta resolución. Las GPUs fueron diseñadas originalmente para este propósito. Sin embargo, también se están utilizando para desarrollar aplicaciones de aprendizaje automático.

Capítulo 5. Desarrollo de la solución propuesta

En este capítulo se va a tratar en profundidad el proceso de desarrollo de las 3 partes fundamentales que componen el trabajo, las cuales consisten en el desarrollo de la aplicación para las gafas de realidad aumentada, la configuración del servidor y del *framework* de IA y el proceso de entrenamiento del modelo personalizado para la detección de objetos.

5.1 Detalles técnicos de la Aplicación desarrollada para las gafas

Como bien se mencionó en el Capítulo 4, el enfoque elegido a la hora de trabajar con las gafas fue usando el IDE Android Studio, con el objetivo de desarrollar una aplicación, ya que el sistema operativo usado por las gafas está basado en un Android 5.1

A la hora de trabajar en la solución de la parte del cliente, es decir, de las gafas, se tuvieron en cuenta 3 requisitos:

1. Captura de cámara.
2. Conectividad para la transmisión de datos.
3. Representación en AR.

Para el primer punto, hay distintas librerías de Android que ofrecen un control completo de la cámara a la hora de desarrollar software para Android. Entre ellas podemos encontrarnos alguna como es *CameraX*, la cual pertenece a Jetpack.

Jetpack **[10]** es un conjunto de bibliotecas que ayuda a los desarrolladores a seguir las prácticas recomendadas, reducir el código estándar y escribir código que funcione de manera coherente en los dispositivos y las versiones de Android para que puedan enfocarse en el código que les interesa.

Otra opción que se contempló es el uso de *Camera2*, la cual es una librería para el manejo de la cámara de bajo nivel de Android que reemplaza a la clase *Camera* obsoleta. *Camera2* proporciona controles detallados para casos de uso complejos, pero requiere que administres configuraciones específicas del dispositivo.

Por último, la opción elegida por su sencillez a la hora de la programación fue la librería *Camera*. Esta librería, como hemos visto antes, ha quedado obsoleta y ha sido sustituida con el tiempo por la librería *Camera2*. Que se haya quedado obsoleta no significa que no sea funcional, si no que, debido a la evolución del desarrollo de *hardware* para dispositivos móviles que usan Android y de la propia evolución del sistema operativo Android, Google desarrolló esta librería la cual incluía muchas más características a la hora del manejo de la cámara.

A la hora de usar esta librería, en el código de la aplicación (Anexo A.1) destacan dos métodos principales para el uso y funcionamiento de la cámara:

- **startCamera()**: Este método es ejecutado al iniciarse la aplicación y, como su nombre indica, contiene toda las líneas de código relacionadas a la inicialización de la cámara. En este método se inicializa el objeto de la clase *Camera*, creado anteriormente, en el cual se almacenan los parámetros necesarios para el funcionamiento de esta, como pueden ser la resolución, el formato de las imágenes que captura, la orientación del *display* y el tamaño del *buffer*.
- **onPreviewFrame()**: Cuando la cámara comienza a capturar fotogramas, estos se almacenan en el *buffer* que se ha configurado previamente. En este *buffer* se almacena todos los datos del fotograma capturado en formato NV21. Dicho eso, el método onPreviewFrame() es llamado cada vez que un fotograma es capturado y almacena la información de este en una variable global, la cual más adelante es utilizada para enviar este fotograma por la conexión creada entre el servidor y las gafas.

Para la realización del segundo punto (la conectividad entre cliente y servidor) se plantearon diversas soluciones, ya que el objetivo principal es enviar imágenes en tiempo real para que fueran pasadas por el modelo de detección de objetos. Entre estas estuvieron consideradas RTSP (*Real Time Streaming Protocol*) o HLS (*Http Live Streaming*), ya que son protocolos enfocados a la transmisión en tiempo real de video, pero debido a que el uso de estos requería una serie de configuraciones adicionales como pueden ser la implementación un servidor RTSP exclusivamente para la transmisión de video, se optó por una solución más sencilla y práctica que fue mediante una conexión TCP usando *sockets* de programación.

Para llevar a cabo la tarea, en la parte del cliente se desarrolló un *thread* dentro de la aplicación principal, el cual se ejecutaba paralelamente a la captura de los fotogramas de la cámara. El objetivo de que funcionase paralelamente era que, cada vez que el *buffer* de la cámara se actualizaba con nuevos datos al capturar un fotograma, estos mismos fuesen tratados y enviados mediante la conexión realizada. Para ello se creó un objeto de la clase *ServerSocket* que se inicializaba al ejecutarse la aplicación y se quedaba a la escucha de una conexión, concretamente en el puerto 8080. Cuando la conexión se aceptaba, se ejecutaba un bucle infinito donde los datos de los fotogramas en formato NV21 se comprimían en formato JPEG para reducir el tamaño de estos y conseguir una latencia baja. Además de reducir el tamaño para la transmisión en tiempo real, el comprimir los datos en JPEG servía para, en la parte del servidor, poder identificar el inicio y el fin de cada fotograma enviado mediante el uso de los *Magic Numbers*, los cuales son unos valores específicos dentro de una secuencia de bytes que son usados para identificar y delimitar distintos tipos de formato de archivos.

Para el tercer y último punto, la cual es la representación en realidad aumentada de la detección de objetos, se hizo algo similar al tema de la conectividad entre cliente y servidor.

Junto al resto de la aplicación, se creó otro *thread* el cual se ejecutaba paralelamente y se encargaba de recibir una serie de coordenadas enviadas por el servidor una vez este realizaba la inferencia, ya que estas hacían referencia a los objetos detectados en los fotogramas que habían sido enviados por las gafas. Concretamente, el *thread* encargado de recibir las coordenadas recibía una lista serializada en formato JSON la cual contenía una serie de elementos, y a su vez, cada elemento contenía un total de 4 coordenadas.

La cantidad de elementos en esta lista enviada variaba dependiendo de la cantidad de objetos que el modelo de IA había detectado al procesar el fotograma, es decir, si se detectaban 3 objetos, las gafas recibirían una lista con 3 objetos con sus correspondientes coordenadas. Las 4 coordenadas representaban las coordenadas de las *bounding boxes* que el modelo de IA creaba cuando el fotograma procesado detectaba un objeto. Estas *bounding boxes* eran simplemente rectángulos que se dibujaban sobre el objeto detectado en cuestión.

El formato de la lista sería de la siguiente manera:

$$[x, y, z, \dots] \tag{1}$$

Siendo *x*, *y*, *z* los elementos que formarían la lista, los cuales tendrían el siguiente formato:

$$x = [xmin, xmax, ymin, ymax] \tag{2}$$

Siendo estos 4 valores las coordenadas de la *bounding box* del objeto.

Con estas coordenadas recibidas, se creó otra clase, la cual se denominó *CustomView* (Anexo A.2), que tenía como objetivo representar sobre el *display* de las gafas la posición de los objetos en el mundo real.

Mediante el *thread* que recibe las coordenadas, éstas se pasaban a la clase *CustomView* la cual tiene una serie de métodos sencillos que se encargan tanto de recibir la lista con las coordenadas como de representar en pantalla estos rectángulos, por lo que, al visualizar la cámara por las gafas en tiempo real, se pueden observar los rectángulos superpuestos sobre los objetos en la realidad. Toda la carga de trabajo recae sobre el servidor.

5.2 Configuración del servidor y su comunicación con las gafas de realidad aumentada

El servidor tiene como objetivo tanto entrenar el modelo de IA como realizar la inferencia con nuevos datos de entrada, siendo éstos proporcionados por el dispositivo cliente, esto es, las gafas de realidad aumentada. Para estas dos tareas se requiere una gran potencia de cómputo, por eso mismo se disponían de todas las características mencionadas en el Capítulo 4.1.2 relacionado con el *hardware*.

5.2.1 GPU Passthrough

En el servidor se creó una máquina virtual dentro del entorno de virtualización Proxmox en la que se instaló Windows 10 como sistema operativo. Esta decisión se tomó ya que Windows es un entorno considerablemente más sencillo que sistemas Linux para trabajar con gráficas de la compañía NVIDIA, concretamente para entornos de IA.

Como ya se ha visto en el Capítulo 4.1.2, en el servidor hay físicamente instaladas 8 tarjetas gráficas NVIDIA A40. No obstante, se empleó una sola ya que era suficiente para las cargas de trabajo del proyecto (entrenamiento e inferencia). Para para poder emplear dicha tarjeta gráfica desde la máquina virtual, hubo que realizar un GPU passthrough, de forma que la tarjeta se asignó a la máquina de forma dedicada.

El *GPU passthrough* **[11]** es una tecnología que permite al *kernel* de Linux presentar directamente una GPU PCI interna a una máquina virtual. El dispositivo actúa como si estuviera siendo controlado directamente por la máquina virtual y la máquina virtual detecta el dispositivo PCI como si estuviera conectado físicamente. El *GPU passthrough* también se conoce a menudo como IOMMU, aunque esto es un poco incorrecto, ya que el IOMMU es la tecnología de *hardware* que proporciona esta característica, pero también proporciona otras características como protección contra ataques de DMA o la capacidad de direccionar espacios de memoria de 64 bits con direcciones de 32 bits.

Para asignar la tarjeta gráfica a la máquina virtual en específico es necesario seguir estos pasos:

1. Asegurarse de que IOMMU está activado en la BIOS del servidor físico, para a continuación activar este mismo en Proxmox. Para eso hay que modificar el GRUB bootloader de la manera en la que se indica en la Figura 5.1:

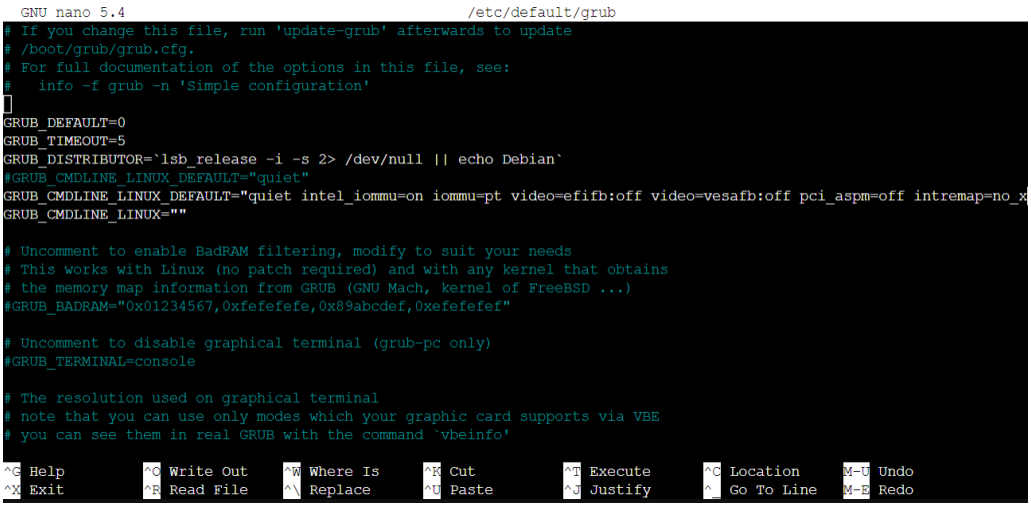

Figura 5.1: GRUB Bootloader

En nuestro caso, la línea que era necesaria modificar era:

GRUB_CMDLINE_LINUX_DEFAULT=”quiet”

La cual tenemos que dejarla de la siguiente manera, ya que en nuestro caso, hay instalados procesadores de la marca Intel:

GRUB_CMDLINE_LINUX_DEFAULT=”quiet intel_iommu=on”

Luego de esto habría que ejecutar el comando *update-grub* para que los cambios surtieran efecto.

2. Modificar el archivo *modules* dentro del directorio */etc/* de manera que se muestra en la figura 5.2 siguiente manera:

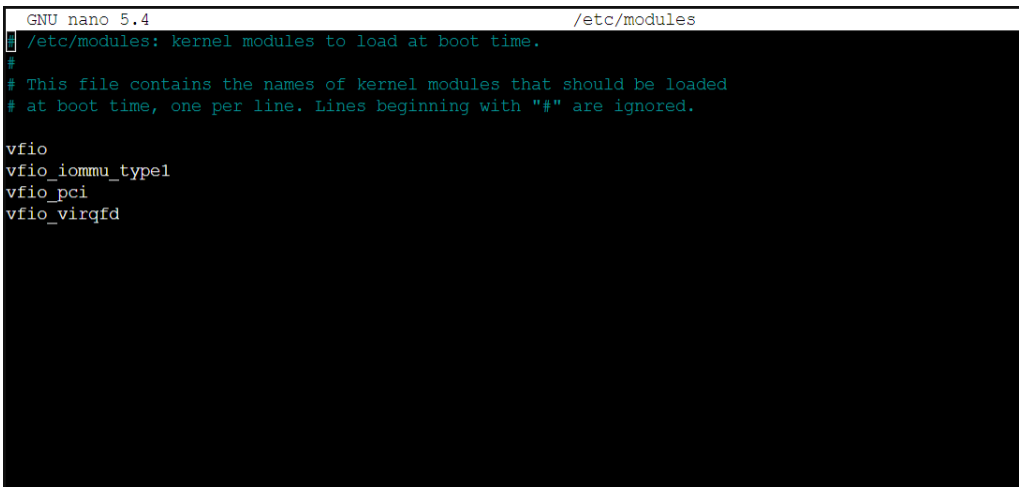

Figura 5.2: Modules

3. Asignar la tarjeta gráfica a la máquina virtual en la interfaz del Proxmox como se observa en la figura 5.3:

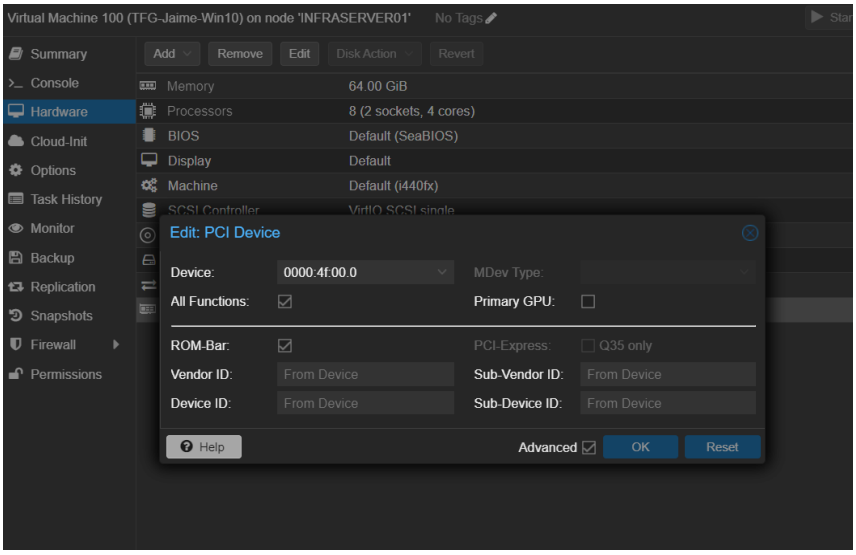

Figura 5.3: Asignación de gráfica

Con estos pasos realizados, ya tendríamos disponible en la máquina virtual la NVIDIA A40.

5.2.2 Drivers, CUDA Toolkit y cuDNN

El siguiente paso a la hora de la configuración consiste en descargar tres componentes de *software* imprescindibles para poder hacer uso de la tarjeta gráfica:

- Drivers de la tarjeta gráfica
- CUDA Toolkit
- Librería cuDNN

Los dos primeros elementos son sencillos de instalar, ya que accediendo a la página oficial de NVIDIA nos podemos descargar un instalador local en el que vendría incluidos tanto los drivers como el CUDA Toolkit compatible para los drivers elegidos.

En la Figura 5.4 se muestra una imagen del portal de descarga del instalador de NVIDIA, con las opciones seleccionadas:

Figura 5.4: Portal de descarga de CUDA Toolkit

Una vez instalado los drivers y el CUDA Toolkit, es necesario reiniciar el sistema para que los cambios surjan efecto. Tras realizar el reinicio, para comprobar que todo ha funcionado correctamente se debe de introducir el comando *nvidia-smi* en la terminal de Windows:

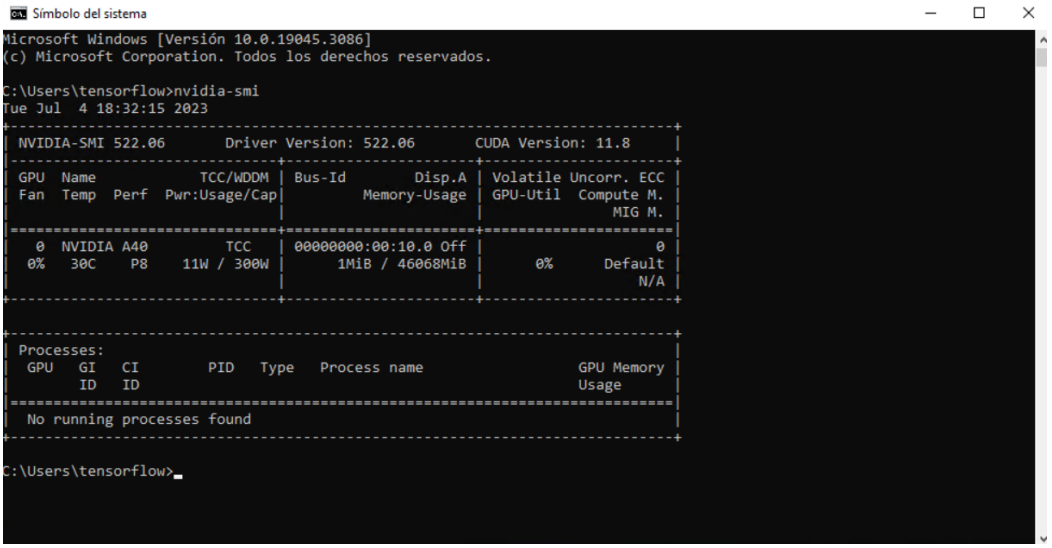

Figura 5.5: nvidia-smi

Como se puede observar, la salida al introducir el comando es una tabla con toda la información relacionada con la tarjeta gráfica, la versión de los drivers y la del CUDA Toolkit.

Por último, para completar esta parte de la configuración, es necesario descargar la librería cuDNN que también pertenece a la propia NVIDIA, ya que es necesaria para el correcto funcionamiento de la gráfica a la hora de trabajar en entornos de IA. Esta librería requiere de la creación de una cuenta en la web de NVIDIA. Cuando ya la tenemos creada, simplemente debemos de escoger la versión de la librería compatible con la versión de los drivers instalados y descargarla. Una vez descargada, el archivo contendrá una serie de carpetas, las cuales simplemente tendremos que copiar en la carpeta de instalación del CUDA Toolkit, la cual se muestra en la Figura 5.6:

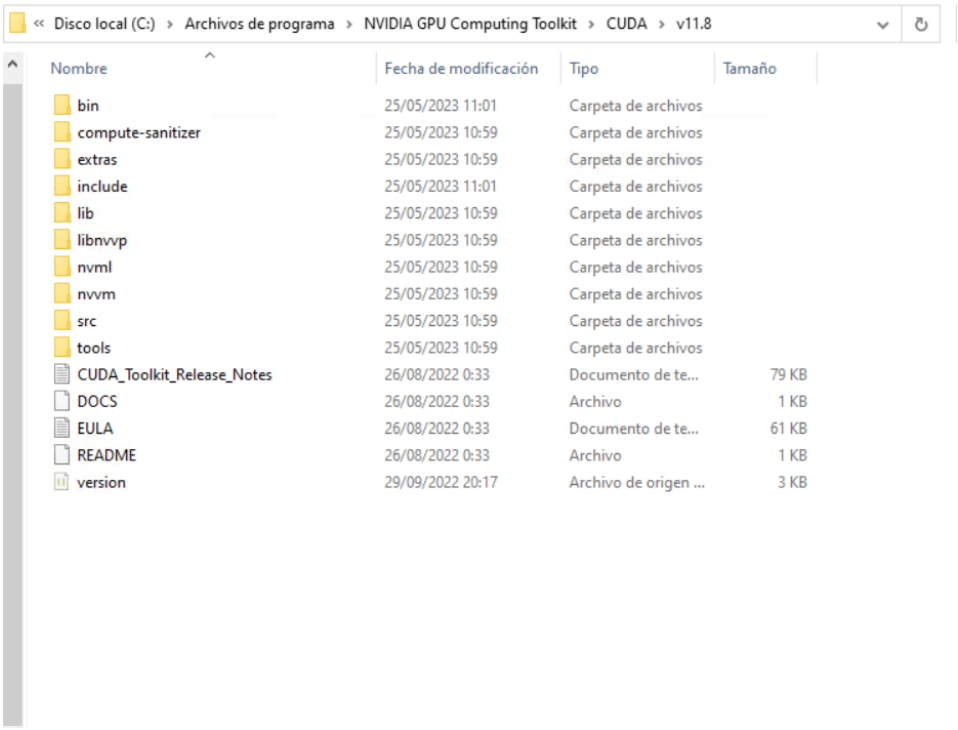

Figura 5.6: Carpeta de instalación de CUDA Toolkit

5.2.3 Tensorflow

En este último punto se van a desarrollar los pasos necesarios para configurar el framework con el que vamos a trabajar, TensorFlow. Específicamente nos vamos a centrar en como configurarlo para poder trabajar y entrenar con modelos de detección de objetos, ya que vamos a hacer uso de la API que viene incluida en el repositorio de GitHub **[18]**.

API **[19]** significa interfaz de programación de aplicaciones (*Application programing inteface*, por sus siglas en inglés). Es un conjunto de definiciones y protocolos para construir e integrar *software* de aplicaciones. Las APIs permiten que tu producto o servicio se comunique con otros productos y servicios sin necesidad de conocer cómo están implementados. Esto puede simplificar el desarrollo de aplicaciones, ahorrando tiempo y dinero.

Lo primero y más importante, vamos a necesitar tener descargado e instalado Python. Python **[20]** es un lenguaje de programación de alto nivel que es muy conocido por su facilidad a la hora de aprenderlo y por la simplicidad de este mismo a la hora de desarrollar programas, de ahí que sea una opción bastante popular para tanto principiantes como para programadores con una cierta experiencia. A parte de esto, Python es muy usado en el mundo del aprendizaje automático.

Una vez tenemos Python instalado se seguirán los siguientes pasos para la configuración de TensorFlow:

1. Se ha de clonar el repositorio *models* de la página de TensowFlow en GitHub. Esto se puede hacer mediante la herramienta Git, o simplemente podemos acceder al repositorio en cuestión mediante un navegador y seleccionar la opción de "Descargar ZIP".

2. Una vez se tiene el repositorio en nuestro ordenador, vamos al directorio donde lo hayamos guardado y dentro del directorio *models* nos iremos a *research/object_detection/packages/tf2*. Dentro de este directorio encontraremos un archivo de Python llamado *setup.py*. Copiaremos el archivo y lo moveremos a la carpeta *research.*

3. Abriremos el terminal de Windows, nos iremos a la ubicación donde hemos copiado el archivo "setup.py" e introduciremos el siguiente comando en la terminal:

   Python -m pip install .

   Este comando, al ejecutarlo, buscará en el directorio en el que está un archivo de Python con el nombre *setup* y lo ejecutará. El archivo *setup.py* es un archivo de configuración el cual instalará automáticamente las dependencias de paquetes necesarias para poder utilizar la API de detección de objetos de TensowFlow.

4. Una vez se hayan instalado todas las dependencias, habrá que comprobar que la instalación ha sido exitosa. Para ello ejecutaremos mediante el uso del terminal el archivo *model_builder_tf2_test.py* el cual se encuentra en el directorio *research/object_detection/builders/.* Al ejecutarlo empezarán a realizarse una serie de pruebas usando la API de TensorFlow. Para saber si la instalación ha sido realizada correctamente, al final de la salida al ejecutar este script de Python nos debería de salir algo similar a lo que se muestra en la Figura 5.7:

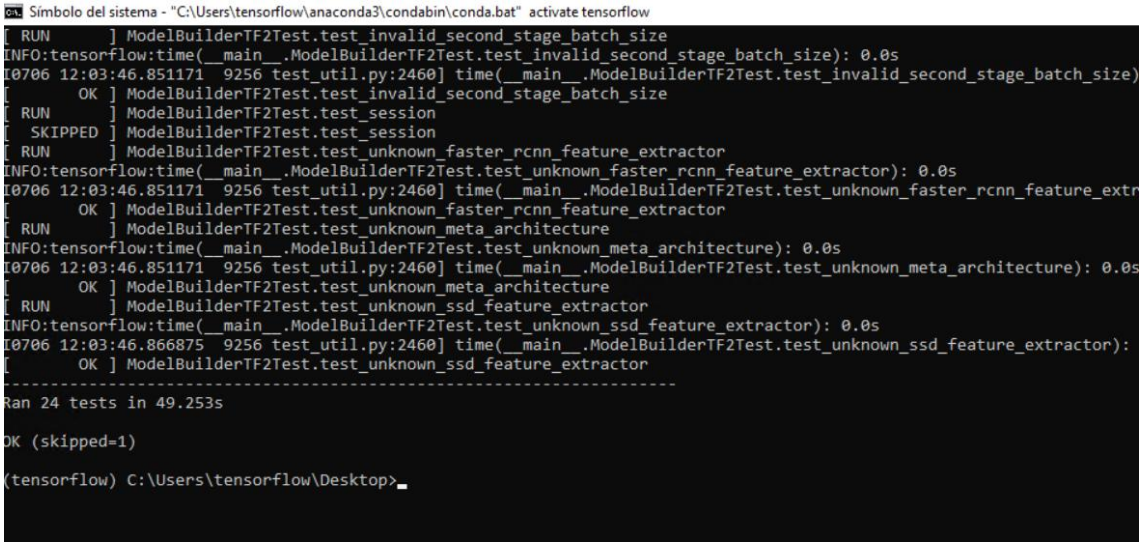

Figura 5.7: Prueba de instalación de TensorFlow

5.2.4 Script de Python

En el Capítulo 4.1.1 se ha visto el desarrollo de la aplicación de las gafas, la cual cuenta con una serie de características para su funcionamiento, como son la captura de fotogramas, el envío de estos a través de una conexión o la representación de la detección en realidad aumentada.

Para realizar la conexión, no es suficiente con la aplicación desarrollada para las gafas, si no que ha sido necesario realizar un script de Python (Anexo B) para la parte del servidor encargado de realizar una serie de tareas complementarias a la de las gafas. Estas tareas se basan en:

- Conectarse a las gafas, las cuales están a la escucha de alguna conexión mediante un *ServerSocket.*
- Realizar la inferencia del modelo entrenado sobre los fotogramas recibidos por las gafas.
- Extraer información relevante de la inferencia, para luego enviarla de vuelta a las gafas.
- Mostrar por pantalla en tiempo real las imágenes procesadas por el modelo de detección de objetos, observando los objetos detectados rodeados por las *bounding boxes.*

Cuando se ha realizado la instalación de TensorFlow y todas las dependencias, dos de ellas han sido de gran utilidad a la hora de desarrollar este script:

- **OpenCV** es una amplia biblioteca de código abierto para la visión por computadora, el aprendizaje automático y el procesamiento de imágenes, y ahora desempeña un papel importante en las operaciones en tiempo real, lo cual es muy importante en los sistemas actuales. Al utilizarlo, se puede procesar imágenes y videos para identificar objetos, caras e incluso la escritura a mano de un ser humano.**[12]**

Figura 5.8: OpenCV

- **NumPy** es un paquete de procesamiento de arreglos de propósito general. Proporciona un objeto de array multidimensional de alto rendimiento y herramientas para trabajar con estos arreglos. Es el paquete fundamental para la computación científica con Python.**[13]**

Sabiendo estos puntos clave a la hora de desarrollar el programa, vamos a ver más en detalle las funciones principales de éste mismo:

1. **receive_frames()**: Esta función es la encargada de recibir los fotogramas de las gafas. Utiliza un socket para recibir los datos y decodifica los bytes recibidos en una imagen utilizando OpenCV. Como se mencionó antes, se hacen uso de los *Magic Numbers* para poder reconstruir bien los fotogramas recibidos en *JPEG* en el lado del servidor.

2. **send_coordinates(coordinates)**: Esta función se encarga de enviar las coordenadas de detección serializadas al servidor. Convierte las coordenadas en una cadena *JSON* para luego enviar los datos a través del *socket*.

3. **load_model(model_path):** Esta función carga un modelo de TensorFlow guardado en la ruta especificada. Devuelve el modelo cargado.

4. **run_inference_for_single_image(model, image)**: Esta función ejecuta la inferencia de detección de objetos en una sola imagen utilizando el modelo cargado y la imagen proporcionada. La imagen se convierte en un tensor de TensorFlow y se realiza dicha inferencia. Los resultados de esta se procesan y se devuelven en un diccionario.

5. **run_inference(model, category_index)**: Esta función se encarga de ejecutar continuamente las inferencias de detección de objetos en tiempo real. Utiliza la función **pilla_frames()** para recibir los fotogramas del servidor, luego llama a **run_inference_for_single_image()** para obtener los resultados de la detección de objetos en cada fotograma. A continuación, se utiliza la función **vis_util.visualize_boxes_and_labels_on_image_array()** para, con los resultados recibidos al realizar la inferencia sobre la imagen, poder visualizar las cajas delimitadoras y las etiquetas de los objetos detectados en la imagen para poder mostrarlo por pantalla posteriormente mediante el uso de OpenCV. Estos resultados se obtienen del diccionario de salida de la inferencia.

Tras realizar la inferencia, se realiza un filtrado de las cajas delimitadoras y puntajes de detección para seleccionar solo aquellos objetos cuyos puntajes superen un umbral mínimo establecido (en este caso, 0.75). Esto se hace para que cuando se llame a la función **send_coordinates()** para enviar las coordenadas serializadas a las gafas, solo envíe aquellas coordenadas de los objetos detectados con un mínimo de confianza.

Por último, el programa, usando OpenCV, muestra las imágenes procesadas continuamente con sus respectivas *bounding boxes* y etiquetas por pantalla utilizando *cv2.imshow()*.

Cabe destacar que el script requiere de dos argumentos de entrada cuando se ejecuta en la terminal:

- La ruta del modelo entrenado (‘-m’ o ‘--model’)
- La ruta del archivo de mapeo de etiquetas (‘-l’ o ‘--labelmap’)

5.3 Entrenamiento y evaluación del modelo de detección de objetos.

En el Capítulo 2 se vio que existían dos formas de enfocar el entrenamiento de modelos de detección de objetos. Por un lado, existía la posibilidad de diseñar un modelo de detección objetos desde cero en el que se necesitaba diseñar una arquitectura de red y configurar manualmente las capas y los pesos. Para ello era necesario recopilar una cantidad inmensa de datos etiquetados para entrenar este modelo.

No obstante, existía una segunda opción que era utilizar un modelo de detección de objetos previamente entrenado. Esto se conoce como aprendizaje por transferencia, ya que, en lugar de entrenar un modelo desde cero, esta técnica te permite aprovechar los conocimientos aprendidos por estos modelos previamente entrenados en grandes conjuntos de datos, como puede ser el caso de *COCO Dataset* **[21]** (conjunto de imágenes de diversas escenas cotidianas, capturando una gran cantidad de objetos diversos en distintos contextos), para así luego ajustar el modelo personalizado que se esté entrenando a los datos etiquetados de interés que uno desee.

Dicho esto, la forma a proceder para crear nuestro modelo personalizado de detección ha sido mediante la técnica de aprendizaje por transferencia, en la que se ha usado un modelo pre-entrenado llamado *EfficientDet D1*, obtenido de la colección de modelos para detección de objetos de TensorFlow, entrenados con el *COCO Dataset*. Sabiendo que este conjunto incluye más de 200,000 imágenes y que cada objeto ha sido etiquetado cuidadosamente con cajas delimitadoras y posteriormente usando máscaras de segmentación para delinear precisamente el objeto, lo convierte en uno de los mejores conjuntos de datos para esta tarea.

5.3.1 Preparación de los datos

Para la preparación de los datos eran necesarias dos cosas:

- Recopilación de imágenes
- Etiquetado de imágenes

Por una parte, como el objetivo era detectar partes específicas de un motor de un coche, específicamente la posición de dos sensores que tenía instalado el coche en la zona del motor, fue necesario recopilar una cantidad generosa de imágenes en las que la variedad de estas fuera abundante. Para construir esta colección, se realizaron fotografías teniendo en cuenta diferentes posiciones y ángulos, incluyendo además imágenes en las que no estuvieran presentes ninguno de los componentes que se querían detectar, para disminuir el número de falsos positivos.

La segunda parte consistió en utilizar un *software open-source* llamado *LabelImg*, que permite etiquetar de manera manual fotografías que uno haya recopilado para posteriormente crear un conjunto de datos para entrenar un modelo de detección de objetos.

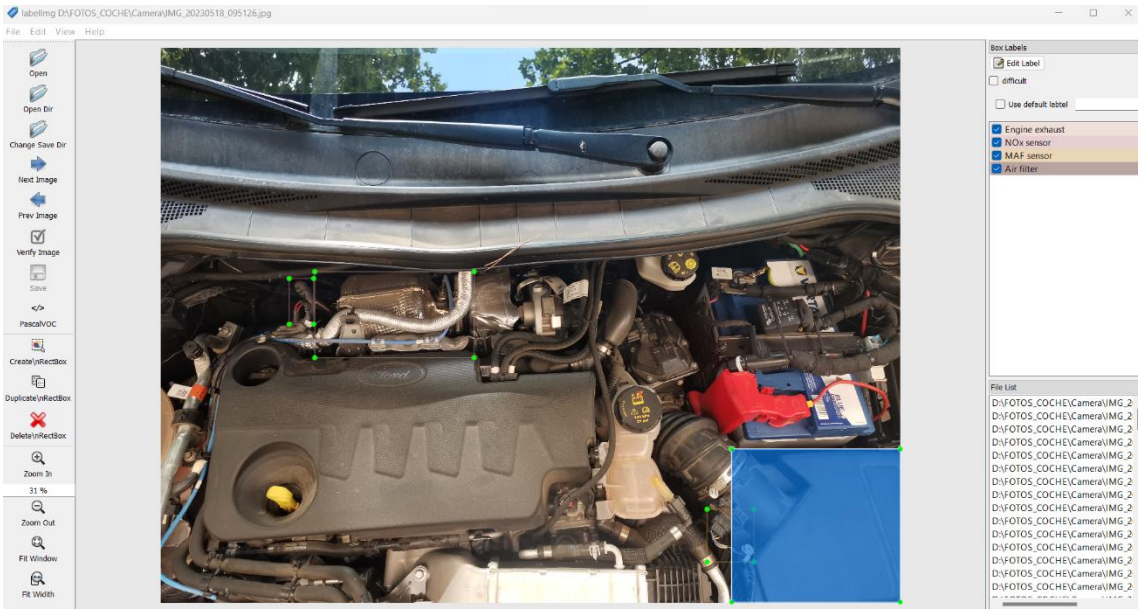

Figura 5.9: LabelImg

En la Figura 5.9 se aprecia un ejemplo las 800 imágenes que había en el conjunto que se recopiló para el conjunto de datos. En esta imagen aparecen precisamente los 4 componentes que se querían detectar:

- Engine exhaust.
- NOx sensor.
- MAF sensor.
- Air filter.

El objetivo principal era detectar los sensores instalados en la zona del motor, pero al ser de tamaño reducido, detectar estos sensores desde una cierta distancia iba a ser complejo para el modelo. Por eso mismo se decidió añadir dos componentes más a parte de los sensores, ya que así podrían servir como referencia para así, una vez se hubiesen detectado estos componentes más grandes y generales, poder acercase a la zona en cuestión y poder detectar con claridad los mismos.

Cada vez que se realizaba el etiquetado de una imagen, el *software* generaba un archivo con formato *XML* y con el mismo nombre que la imagen etiquetada, en el cual se encontraban todos los datos relacionados con la imagen y con las posiciones de las cajas delimitadoras con sus respectivas etiquetas creadas para detectar los objetos en cuestión.

Una vez se tenían estos datos etiquetados, el total de archivos resultantes había que dividirlo en dos carpetas distintas:

- Entrenamiento: 70-80%
- Evaluación: 20-30%

El objetivo es utilizar gran parte de las imágenes recopiladas para el proceso de entrenamiento del modelo, reservando una parte minoritaria de dicho conjunto de imágenes (no vistas durante el proceso) para pasarlas por el modelo obtenido, para poder ajustarse a nuevos datos y perfeccionar la precisión de este.

Para realizar el entrenamiento, éste requería de un archivo con formato *CSV* ya que este tipo de archivos contienen una gran cantidad de datos de forma concatenada y por lo tanto son capaces de almacenar toda la información relacionada con las etiquetas del conjunto entero de imágenes. Para ello se hizo uso de un script de Python denominado *xml_to_csv.py* **[22]** el cual, con las imágenes ya divididas en las dos carpetas con sus respectivos archivos *XML* correspondientes, se ejecutaba y generaba dos archivos *CSV*, uno con toda la información de las imágenes dedicadas al entrenamiento y otro con la de las imágenes dedicadas a la evaluación.

Por último, en lo que se refiere a la preparación de los datos, es necesario crear un archivo con el formato *pbtxt.* Este archivo generado tiene generalmente como nombre *labelmap.pbtxt*, ya que es un archivo utilizado en la API de detección de objetos de TensorFlow para definir las etiquetas o clases de objetos que un modelo de detección de objetos puede reconocer. El propósito principal de este archivo es establecer una correspondencia entre un identificador numérico y una etiqueta legible por humanos para cada clase de objeto. En el caso concreto de este trabajo, se creó un archivo *labelmap.pbxt* con la estructura de la Figura 5.10:

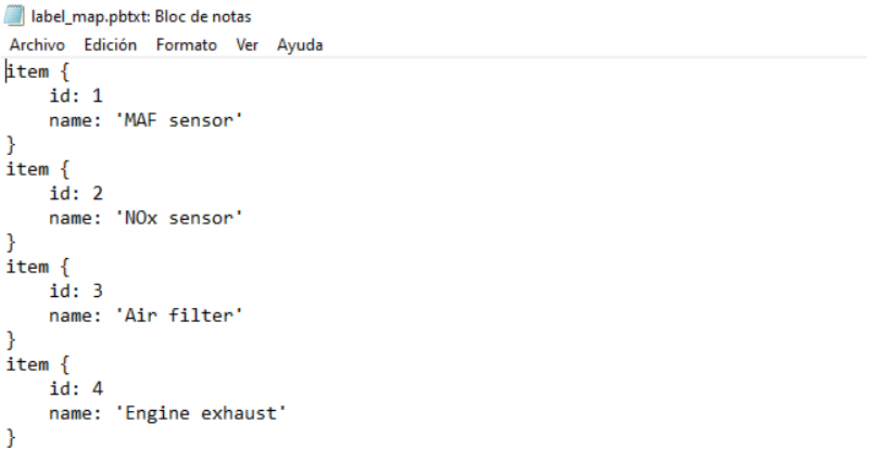

Figura 5.10: Archivo labelmap.pbtxt

5.3.2 Configuración del entrenamiento

Una vez con las imágenes divididas y con sus archivos *CSV* respectivos conteniendo la información de las etiquetas, es necesario generar, a partir de lo mencionado anteriormente, unos archivos compatibles con la librería de TensorFlow para poder proceder con el entrenamiento. Para eso se utilizó un script de Python denominado *generate_tfrecord.py* **[22]** el cual, a partir de los archivos *CSV* y los directorios que contienen las imágenes de entrenamiento, genera dos archivos *TFRecord* llamados *train.record* y *test.record*. Estos archivos serán los que se utilizarán posteriormente para el entrenamiento del modelo.

Como se ha mencionado al inicio de esta sección, al estar utilizando la técnica de aprendizaje por transferencia, se descargó el modelo *EfficientDet D1* de la colección de modelos de TensorFlow. Al descargar este modelo nos encontramos con un archivo llamado *pipeline.config* el cual describe todos los parámetros configurados para tanto el entrenamiento como la inferencia. Por eso mismo deberemos de modificarlo para ajustarlo a las necesidades de nuestro caso. Posteriormente se van a presentar los parámetros principales a modificar de este archivo, explicando estos brevemente y con los valores que se usaron en el proyecto:

- **Batch_size**: Determina la cantidad de imágenes que se procesarán en cada iteración durante el entrenamiento del modelo. Cuanto más grande sea, la velocidad del entrenamiento será mayor, pero a su vez requiere de más memoria. En el proyecto se utilizó un valor de 16.

- **Fine_tune_checkpoint:** Especifica la ruta al archivo de punto de control del modelo pre-entrenado utilizado para el aprendizaje por transferencia. Así, el entrenamiento partirá con este modelo como referencia y continuará entrenándolo con el conjunto de imágenes que se ha recopilado. En este caso, se le indicó la ruta del punto de control del modelo *EfficientDet D1*.

- **Fine_tune_checkpoint_type:** Define el tipo de punto de control que se utilizará para el ajuste fino. Se puede establecer en *classification* para modelos de clasificación o en *detection* para modelos de detección de objetos. En este caso se estableció como tipo *detection*.

- **Label_map_path:** Indica la ruta al archivo *labelmap.pbtxt* mencionado anteriormente, el cual define los identificadores de las clases con sus correspondientes etiquetas asignadas durante el proceso de etiquetado y la lista de clases que el modelo está preparado para detectar. En este caso se indicó la ruta donde estaba el archivo con las 4 clases creadas.

- **Train_record_path:** Especifica la ruta al archivo *TFRecord* que contiene los datos de entrenamiento. Se indicó la ruta donde estaba el archivo *train.record* generado a partir del archivo *CSV* y del directorio de imágenes de entrenamiento.

- **Test_record_path:** Especifica la ruta al archivo *TFRecord* que contiene los datos de evaluación. Se indicó la ruta donde estaba el archivo *test.record* generado a partir del archivo *CSV* y del directorio de imágenes de evaluación.

- **Num_classes:** Define el número de clases o etiquetas en el conjunto de datos. Al tener 4 tipos de objetos a detectar, este valor se estableció como 4.

5.3.3 Entrenamiento y exportar el modelo

Con todo ya preparado para el entrenamiento, el proceso es muy sencillo, ya que al descargar el repositorio de GitHub de TensorFlow, en el directorio de detección de objetos, se proporciona un script de Python con el nombre *model_main_tf2.py*. Este script requiere de los siguientes argumentos de entrada para su correcto funcionamiento:

- **--pipeline_config_path**: Especifica la ruta del archivo *pipeline.config* que se ha editado previamente con los valores correctos para el entrenamiento.

- **--model_dir**: Se define la ruta donde se guardará el modelo que se va a entrenar.

- **--alsologtostderr**: Este argumento de entrada permite que se muestren todos los registros de salida por consola.

- **--num_train_steps**: Define el número total de iteraciones que se realizarán en total en el entrenamiento. Cada iteración corresponde al procesamiento de un lote de datos, el cual hemos definido previamente con el *batch size*. En este caso se estableció un total de 16,000 pasos

- **--sample_1_of_n_eval_examples**: Especifica la frecuencia a la que se seleccionan ejemplos del conjunto de evaluación durante el entrenamiento. En este caso se estableció que seleccionara 1 de cada N ejemplos.

- **--num_eval_steps**: Define el número total de iteraciones que se realizarán en total en el proceso de evaluación del modelo. En este caso se estableció un total de 1,000 pasos.

Una vez ejecutado este script con todos los argumentos de entrada correspondientes, el proceso de entrenamiento comenzará. Por consola nos aparecerán actualizaciones de los valores relacionados con el entrenamiento cada 100 pasos, por lo que es posible tener un seguimiento en tiempo real del proceso de entrenamiento. No obstante, cuando se instalaron las dependencias de TensorFlow, entre ellas se instaló TensorBoard.

TensorBoard **[14]** es una herramienta para visualizar y comprender el rendimiento de los modelos de aprendizaje profundo. Es una herramienta de código abierto desarrollada por TensorFlow y se puede utilizar con cualquier marco de trabajo de aprendizaje profundo. TensorBoard permite realizar un seguimiento y visualizar métricas como la pérdida y la precisión, visualizar el grafo del modelo, ver histogramas, mostrar imágenes y optimizar tus códigos de TensorFlow/PyTorch.

A la hora de usar TensorBoard para monitorizar el entrenamiento, debemos de introducir el siguiente comando por la terminal:

"tensorboard --logdir *ruta del directorio de entrenamiento*"

Una vez introducido, se abrirá por defecto el servicio de TensorBoard en el puerto 6006 en la máquina local, donde podremos acceder mediante cualquier navegador.

En la Figura 5.11 podemos observar las distintas gráficas y su evolución a lo largo del tiempo cuando finalizó el proceso de entrenamiento del modelo:

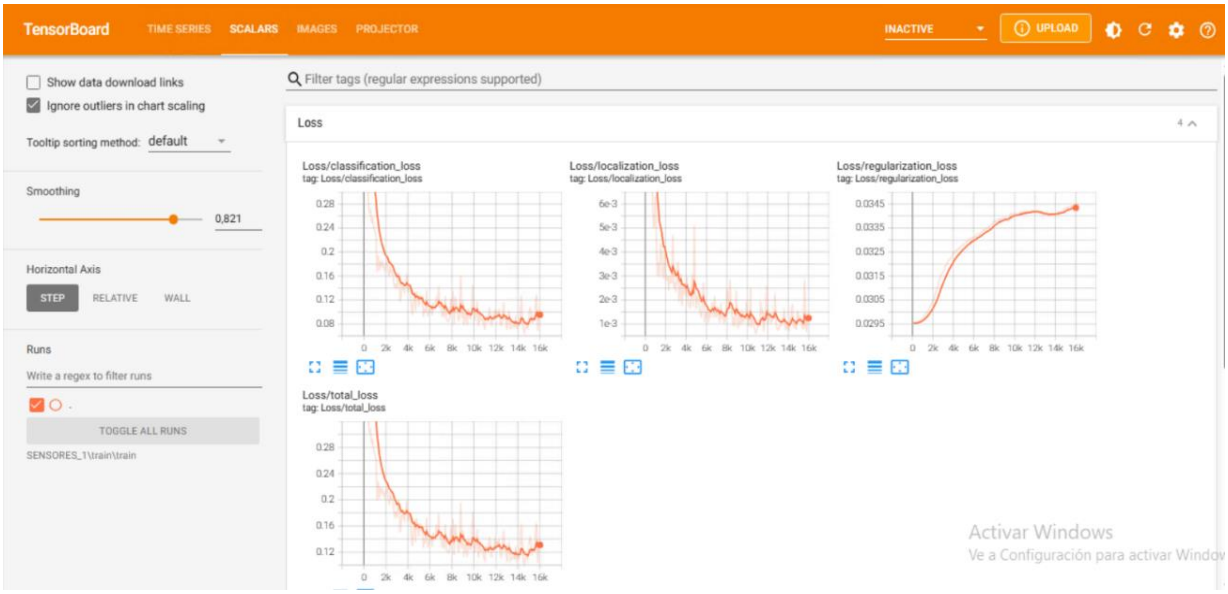

Figura 5.11: TensorBoard

De las gráficas que representa esta herramienta, la que más interesa a la hora de monitorizar un modelo de detección de modelos es la gráfica llamada *classification_loss*. Esta gráfica representa la reducción de las pérdidas a la hora de ejecutar la detección de objetos sobre el conjunto de datos introducido conforme va avanzando el procesamiento. Por una parte, se encuentra el eje horizontal, el cual representa el número de pasos que el proceso de entrenamiento lleva ya realizados, por lo que este eje sería el equivalente a un eje temporal. Por otra parte, tenemos el eje vertical que se utiliza para representar el porcentaje de pérdidas en la detección de objetos, y su escala va de 0 a 1. Al inicio del entrenamiento, es esperado que este valor se acerque al 1, lo que indica una mala precisión y éxito en la detección de objetos. Sin embargo, a medida que el entrenamiento avanza, se espera que este valor disminuya progresivamente hasta estabilizarse en un valor cercano al cero. Lo ideal a la hora de terminar el entrenamiento es cuando se observa que el valor estable ha alcanzado un valor por debajo de 0.1, ya que representa que el modelo ha alcanzado un nivel muy bajo de pérdidas en la detección de objetos y se considera ya un modelo óptimo.

Una vez se ha finalizado el proceso de entrenamiento del modelo, es necesario exportar el grafo de inferencia. Un grafo de inferencia es una representación serializada del modelo entrenado que se puede utilizar para implementación e inferencia. Normalmente contiene la arquitectura del modelo, los pesos aprendidos y otros metadatos necesarios para realizar predicciones sobre nuevos datos. Se utiliza en la fase de inferencia del modelo, donde se realiza la evaluación y se obtienen resultados a partir de estos nuevos datos de entrada.

Para generar este grafo de inferencia una vez se ha finalizado el entrenamiento, se debe utilizar un script incluido en el repositorio de GitHub de TensorFlow descargado previamente denominado *exporter_main_tf2.py*, el cual se encuentra dentro del directorio */models/research/object_detection*. Para ello sencillamente ejecutaremos el script mencionado con los siguientes argumentos de entrada:

- **--trained_checkpoint_dir**: Especifica la ruta donde se ha almacenado el modelo una vez finalizado su entrenamiento. Este será el mismo que el argumento de entrada --*model_dir* cuando se ejecutó el script *model_main_tf2.py*.

- **--output_directory**: Especifica la ruta del directorio donde se almacenará el grafo de inferencia.

- **--pipeline_config_path**: Se define la ruta del archivo de configuración *pipeline.config* que se utilizó para el entrenamiento del modelo.

Una vez realizada la exportación, se creará una carpeta con el nombre que le hemos asignado en el argumento de entrada. La carpeta contendrá el archivo *pipeline.config* que hemos especificado en el argumento de entrada y dos carpetas más. Una de ellas, llamada *checkpoint*, contendrá el último punto de control que alcanzó el modelo durante el entrenamiento. La otra carpeta, llamada *saved_model*, es la carpeta donde estará almacenado el modelo entrenado, listo para ser utilizado para la inferencia. La ruta de esta última carpeta será la que usemos como argumento de entrada para cargar el modelo en el script que se desarrolló para la conexión y realización de la inferencia en tiempo real, que está descrito en el Capítulo 5.2.4.

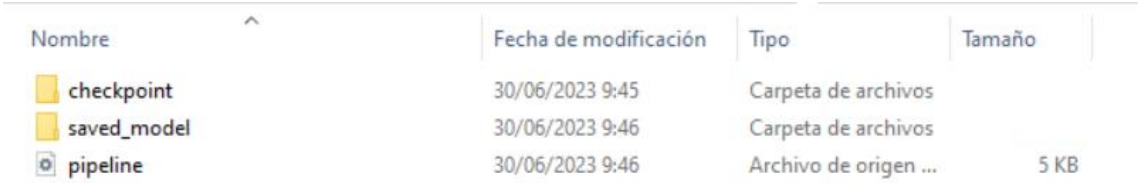

Figura 5.12: Archivos del modelo exportado

Capítulo 6. Pruebas y resultados

A la hora de probar el modelo de detección de objetos en el entorno de trabajo, se llevaron a cabo diversas evaluaciones para determinar su empeño. Durante estas pruebas, se observó que el modelo tenía una buena capacidad para detectar los componentes del coche que se suponía que debía identificar. Sin embargo, surgió un problema que afectaba a la precisión durante la inferencia: la presencia de falsos positivos.

Los falsos positivos en la detección de objetos se refieren a situaciones en las que el modelo identifica erróneamente un objeto que no está presente en la imagen procesada. Esto significa que el modelo genera resultados erróneos al reportar la existencia de un objeto que no existe realmente. Estos falsos positivos pueden ser problemáticos, ya que pueden alterar un sistema el cual use la detección de objetos de manera automática y sin ser supervisado en el momento, como pueden ser cámaras de videovigilancia.

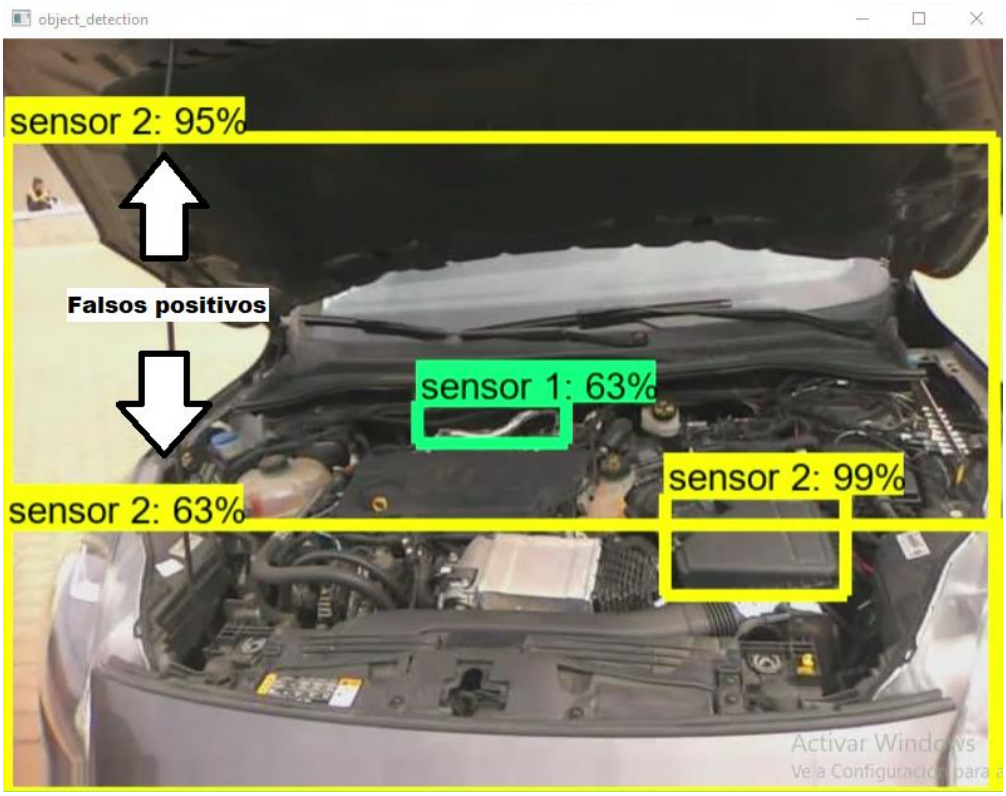

Figura 6.1: Falsos positivos

Se analizaron las posibles causas de la alta aparición de falsos positivos en la inferencia. Se llegó a la conclusión de que, al principio del desarrollo del proyecto, se habían recopilado una cantidad relativamente pequeña de imágenes para entrenar el modelo, cerca de 200. La falta de diversidad y representatividad en el conjunto de datos usados limitó la capacidad del modelo para generalizar y aprender patrones precisos.

Además, durante el entrenamiento, se realizaron un número reducido de pasos, por lo que la duración del entrenamiento fue menor. Al igual que el conjunto reducido de datos, la limitación del tiempo afectó negativamente a la capacidad del modelo para aprender y ajustarse adecuadamente a las características de los componentes que debía detectar, ya que no realizó suficientes iteraciones durante el conjunto de datos y, por ende, los falsos positivos eran abundantes.

Para abordar este problema, se tomó la decisión de ampliar tanto el conjunto de datos como el número de pasos a realizar durante el entrenamiento del modelo de detección de objetos. Se recolectaron más imágenes que abarcaban una mayor diversidad respecto a los componentes del vehículo y del escenario en general y se etiquetaron para luego añadirlas al conjunto de datos que ya se tenía, lo que permitió así una mayor representatividad del modelo.

Junto a la ampliación del conjunto de imágenes, se aumentó la cantidad de iteraciones que el entrenamiento del modelo iba a realizar sobre el conjunto de datos para que así se ajustara de manera más exhaustiva a este y aprendiera patrones más precisos. Esto conllevó un tiempo de entrenamiento más prolongado, pero se consideró una inversión necesaria para reducir la cantidad de falsos positivos presentes durante la inferencia.

Tras realizar estas dos acciones, la mejora de los resultados fue notable. El modelo seguía detectando con precisión los componentes etiquetados mientras que la presencia de falsos positivos se redujo hasta el punto de que ya no aparecían o estos aparecían durante milésimas de segundo durante la ejecución de la inferencia. Como se puede observar en la figura 6.2 y 6.3

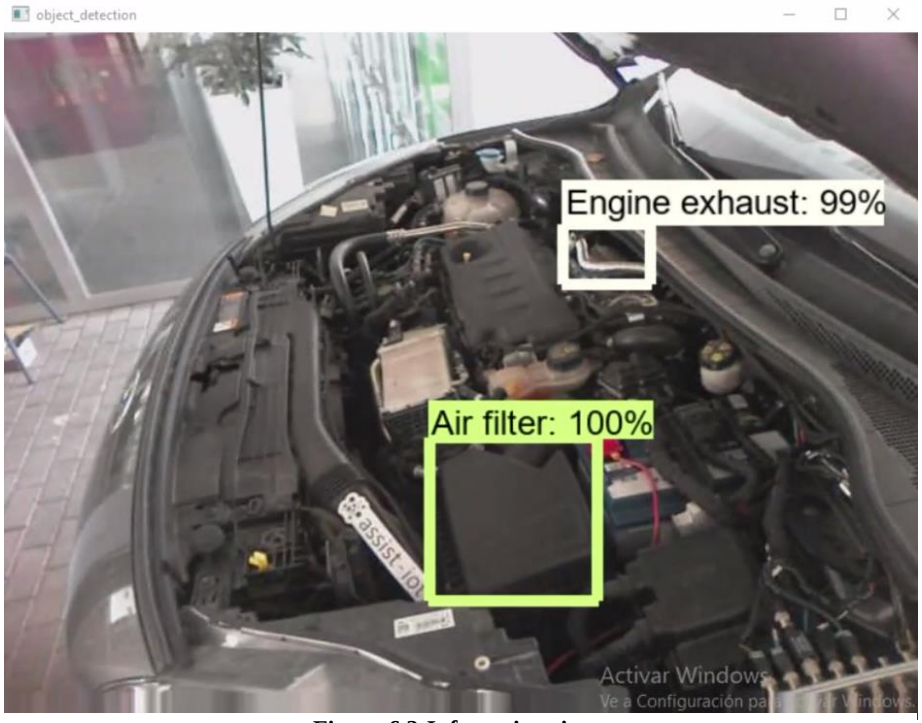

Figura 6.2:Inferencia exitosa

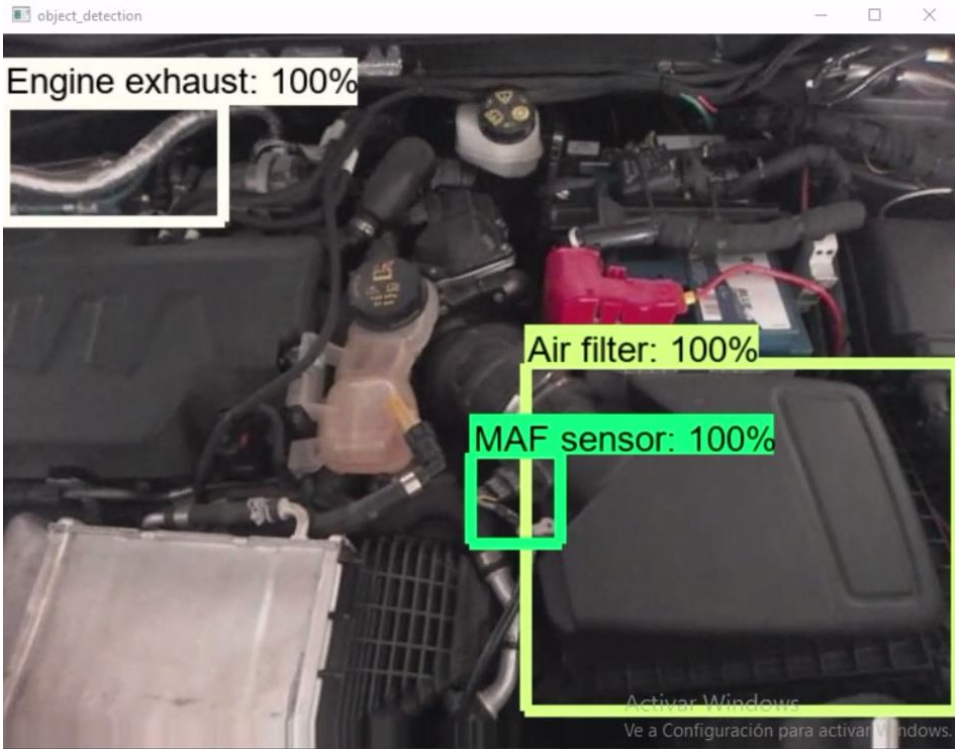

Figura 6.3: Inferencia exitosa

Capítulo 7. Conclusión y trabajo futuro

Como se ha podido observar durante el desarrollo de este Trabajo Fin de Grado, se ha conseguido abordar el diseño y desarrollo de un sistema de detección de objetos con el objetivo de asistir a usuarios especializados en la detección de fallos de sensores implementados en motores automovilísticos. Los tres puntos fundamentales que componen este proyecto se han cumplido con éxito: La transmisión en tiempo real de video, el uso de TensorFlow para la creación de un modelo de detección de objetos y la posterior representación de estos objetos en las gafas de realidad aumentada en un caso práctico.

Al disponer de una tarjeta gráfica dedicada, se ha podido entrenar un modelo eficiente y utilizarlo en tiempo real, sin la que los resultados hubieran sido mucho más pobres. El modelo habría podido ser entrenado con muchas menos imágenes, reduciendo su rendimiento, y procesar todos los *frames* de video en tiempo real no hubiera sido posible.

Aunque los objetivos principales se hayan alcanzado con bastante éxito, surgió un inconveniente durante el desarrollo por el que se plantearon otros caminos a seguir a la hora de avanzar con el proyecto. Esto fue debido a la característica de que las gafas de realidad aumentada EPSON Moverio BT-350 contaban con un campo de visión de 23º. Para trabajar con realidad aumentada, este campo de visión es muy reducido a la hora de representar imágenes en el mundo real, por lo que se tuvo que optar por la posibilidad de representar la detección de los objetos al mismo tiempo que se observaba la propia cámara de las gafas por su *display*, para poder cuadrar los resultados de la detección con lo que se veía a través de estas.

La solución y el resultado que surgieron tras plantearse este inconveniente fueron un éxito. No obstante, sería beneficioso contar con unas gafas especialmente diseñadas para la creación de entornos en realidad aumentada.

Otro aspecto para tener en cuenta de cara al futuro es la posibilidad de recopilar un conjunto de imágenes muchísimo más grande que con el que se ha trabajado para, por ejemplo, poder escalar este proyecto a una línea de producción de una compañía de automóviles para que usuarios especializados puedan trabajar con la tecnología de la realidad aumentada e inteligencia artificial.

Además, la posibilidad de integrar una base de datos y un sistema de comunicación adicional en el coche para almacenar el estado de los sensores podría mejorar el proyecto hasta un punto en el que, además de detectar la localización de dichos sensores, se pudiese adquirir información de dichos elementos, todo ello en tiempo real y sobre realidad aumentada.

Bibliografía

[1] Frankenfield, J. (2023c). Artificial intelligence: What it is and how it is used. *Investopedia*. https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp

[2] *¿Qué es machine learning? | Cómo funciona, tutoriales y ejemplos*. (s. f.-c). MATLAB & Simulink. https://es.mathworks.com/discovery/machine-learning.html

[3] *What is object detection?* (s. f.). MATLAB & Simulink. https://es.mathworks.com/discovery/object-detection.html

[4] Bobeshko, A. (2018, 27 mayo). Object recognition in augmented reality - virtual reality pop. *Medium*. https://virtualrealitypop.com/object-recognition-in-augmented-reality-8f7f17127a7a

[5] *What is augmented reality (AR)? | SAP Insights*. (s. f.). SAP. https://www.sap.com/products/scm/industry-4-0/what-is-augmented-reality.html

[6] NVIDIA. (2022, marzo). *Nvidia A40 Datasheet*. Recuperado 10 de julio de 2023, de https://images.nvidia.com/content/Solutions/data-center/a40/nvidia-a40-datasheet.pdf

[7] Brandon.Lee, & Brandon.Lee. (2023). Proxmox 8: New features and home Lab upgrade instructions. *Virtualization Howto*. https://www.virtualizationhowto.com/2023/06/proxmox-8-new-features-and-home-lab-upgrade-instructions/#h-what-is-proxmox

[8] Tensorflow. (s. f.). *GitHub - Tensorflow/Tensorflow: an open source machine learning framework for everyone*. GitHub. https://github.com/tensorflow/tensorflow

[9] Banoula, M. (2023). What is Tensorflow? Deep learning libraries & program elements. *Simplilearn.com*. https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow#what_is_tensorflow

[10] Recursos para desarrolladores de Android JetPack - Android Developers. (s. f.). *Android Developers*. https://developer.android.com/jetpack?hl=es-419

[11] *GPU passthrough with Libvirt Qemu KVM - GenToo Wiki*. (s. f.). https://wiki.gentoo.org/wiki/GPU_passthrough_with_libvirt_qemu_kvm

[12] GeeksforGeeks. (2022). OpenCV overview. *GeeksforGeeks*. https://www.geeksforgeeks.org/opencv-overview/

[13] GeeksforGeeks. (2023). Python Numpy. *GeeksforGeeks*. https://www.geeksforgeeks.org/python-numpy/

[14] Hoong, K., PhD. (2023, 11 febrero). How to use tensorboard with Pytorch - Kuan Hoong, Ph.D - medium. *Medium*. https://kuanhoong.medium.com/how-to-use-tensorboard-with-pytorch-e2b84aa55e67

[15] *¿Qué es Scrum?* (s. f.). Scrum.org. https://www.scrum.org/resources/blog/que-es-scrum

[16] *Moverio BT-350 | Visualizador móvil translúcido | Gafas inteligentes | Productos | Epson España*. (s. f.). https://www.epson.es/es_ES/productos/gafas-inteligentes/visualizador-m%C3%B3vil-transl%C3%BAcido/moverio-bt-350/p/20179

[17] Android Studio & App Tools - Android Developers. (s. f.). *Android Developers*. https://developer.android.com/studio

[18] Tensorflow. (s. f.-a). *GitHub - Tensorflow/Models: Models and examples built with TensorFlow*. GitHub. https://github.com/tensorflow/models

[19] *What is an API?* (s. f.). https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

[20] *Python.org*. (2023, 1 julio). Python.org. https://www.python.org/

[21] *Coco - Common objects in context*. (s. f.). https://cocodataset.org/#home

[22] TannerGilbert. (s. f.). *GitHub - tannergilbert/tensorflow-object-detection-API-train-model: Train a object detection model with the Tensorflow Object Detection API and Tensorflow 2*. GitHub. https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model

Anexos

Anexo A. Aplicación de las gafas

A.1. Aplicación principal

```
package com.example.opencv_cameraapi_v2;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ImageFormat;
import android.graphics.Paint;
import android.graphics.Rect;
import android.graphics.YuvImage;
import android.hardware.Camera;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.google.gson.Gson;

import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.lang.reflect.Array;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
```

```
public class MainActivity extends AppCompatActivity implements
SurfaceHolder.Callback, Camera.PreviewCallback, Camera.ErrorCallback {

    private Camera mCamera;
    private SurfaceView preview;
    private SurfaceHolder mSurfaceHolder;
    private boolean mIsCapturing;
    private Thread ServerThread, RecibeCoordenadasThread;
    private byte[] mPreviewBuffer;
    private boolean mIsServerRunning;
    private byte[] data;
    private Socket clientSocket;

    private CustomView customView;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        preview = findViewById(R.id.preview);
        mSurfaceHolder = preview.getHolder();
        mSurfaceHolder.addCallback(this);
        mSurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        customView = findViewById(R.id.custom_view);


        startCamera();

        ServerThread = new Thread(new ServerThread());
        RecibeCoordenadasThread = new Thread(new
RecibeCoordenadasThread());

        ServerThread.start();
        mIsServerRunning = true;


    }


    @Override
    public void onError(int i, Camera camera) {

    }
```

```
    @Override
    public void onPreviewFrame(byte[] data, Camera camera) {
        // Almacena los bytes de cada frame en la variable global data
        this.data = data;
        // Añade buffer para la siguiente captura de frame
        mCamera.addCallbackBuffer(mPreviewBuffer);


    }

    @Override
    public void surfaceCreated(@NonNull SurfaceHolder surfaceHolder) {

    }

    @Override
    public void surfaceChanged(@NonNull SurfaceHolder surfaceHolder, int
i, int i1, int i2) {
        if (mCamera != null) {
            try {
                mCamera.setPreviewDisplay(mSurfaceHolder);
                mCamera.startPreview();
            } catch (IOException e) {
                e.printStackTrace();
                Toast.makeText(this, "Failed to start preview.",
Toast.LENGTH_SHORT).show();
            }
        }

    }

    @Override
    public void surfaceDestroyed(@NonNull SurfaceHolder surfaceHolder) {

    }

    private void startCamera() {
        try {
            // Abre la camara
            mCamera = Camera.open();

            // Fija los parámetros
            Camera.Parameters params = mCamera.getParameters();

            params.setPreviewSize(640, 480);
            params.setPreviewFormat(ImageFormat.NV21);
```

```
            mCamera.setParameters(params);
            mCamera.setDisplayOrientation(0);


            mCamera.setPreviewDisplay(mSurfaceHolder);
            mCamera.setPreviewCallbackWithBuffer(this);
            mCamera.setErrorCallback(this);

            // Calculamos buffer para los fotogramas
            int bufferSize = params.getPreviewSize().width *
params.getPreviewSize().height *
ImageFormat.getBitsPerPixel(params.getPreviewFormat()) / 8;
            Log.d("BUFFER SIZE", Integer.toString(bufferSize));
            mPreviewBuffer = new byte[bufferSize];
            mCamera.addCallbackBuffer(mPreviewBuffer);

            // Empezamos a capturar
            mCamera.startPreview();


            mIsCapturing = true;


        } catch (Exception e) {
            e.printStackTrace();
        }

    }


    private class ServerThread implements Runnable {
        private OutputStream outputStream;

        @Override
        public void run() {
            ServerSocket serverSocket = null;
            try {
                serverSocket = new ServerSocket(8080);
                int width =
mCamera.getParameters().getPreviewSize().width;
                int height =
mCamera.getParameters().getPreviewSize().height;
                int format = mCamera.getParameters().getPreviewFormat();
                Log.d("SERVIDOR", "EL SERVER SOCKET HA ARRANCADO Y ESTÁ A
LA ESPERA DE CONEXION");
                Log.d("WIDTH", Integer.toString(width));
                Log.d("HEIGHT", Integer.toString(height));
                Log.d("FORMAT", Integer.toString(format));
```

```
                clientSocket = serverSocket.accept();
                outputStream = clientSocket.getOutputStream();
                Log.d("SERVIDOR", "HA ACEPTADO LA CONEXION");
                RecibeCoordenadasThread.start();
                while (true) {
                    YuvImage yuvImage = new YuvImage(data, format, width,
height, null);
                    ByteArrayOutputStream baos = new
ByteArrayOutputStream();
                    yuvImage.compressToJpeg(new Rect(0, 0, width,
height), 50, baos);
                    byte[] jpegData = baos.toByteArray();
                    outputStream.write(jpegData);
                    outputStream.flush();
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                if (serverSocket != null) {
                    try {
                        serverSocket.close();
                    } catch (IOException e) {
                    }
                }
            }
        }

    }

    private class RecibeCoordenadasThread implements Runnable {

        @Override
        public void run() {
            try {
                BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                while (true) {
                    try {
                        String jsonData = reader.readLine();
                        List<Rect> rectangulos = new ArrayList<>();
                        Gson gson = new Gson();
                        int[][] coordenadas = gson.fromJson(jsonData,
int[][].class);
                        for (int[] fila : coordenadas) {
                            int x = fila[0] * 2;
                            int width = fila[1] * 2;
                            int y = (int) (fila[2] * 1.5);
```

```
                            int height = (int) (fila[3] * 1.5);
                            rectangulos.add(new Rect(x,y,width,height));
                        }
                        customView.setRectangulos(rectangulos);
                        customView.postInvalidate();
                    }catch (Exception e){
                        e.printStackTrace();
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}
```

A.2 *CustomView*

```
package com.example.opencv_cameraapi_v2;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.util.AttributeSet;
import android.view.View;

import java.util.ArrayList;
import java.util.List;

public class CustomView extends View {

    private List<Rect> rectangulos;

    public CustomView(Context context, AttributeSet attrs) {
        super(context, attrs);
        rectangulos = new ArrayList<>();
    }

    public void setRectangulos(List<Rect> rectangulos){
        this.rectangulos = rectangulos;

    }
```

```
    @Override
    public void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        Paint paint = new Paint();
        paint.setColor(Color.GREEN);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(10);

        for (Rect rect : rectangulos){
            canvas.drawRect(rect, paint);
        }
    }
}
```

Anexo B. Programa del servidor

```
import numpy as np
import argparse
import tensorflow as tf
import cv2
import socket
import struct
import json

from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

# patch tf1 into `utils.ops`
utils_ops.tf = tf.compat.v1

# Patch the location of gfile
tf.gfile = tf.io.gfile

def receive_frames():

    # Receive frames from the server
    data = b''
    while b'\xff\xd8' not in data:
        data += client_socket.recv(1024)
        frame_start = data.index(b'\xff\xd8')
        data = data[frame_start:]
    while b'\xff\xd9' not in data:
        data += client_socket.recv(1024)
```

```
        frame_end = data.index(b'\xff\xd9')
        frame_bytes = data[:frame_end]
        frame = cv2.imdecode(np.fromstring(frame_bytes, dtype=np.uint8),
cv2.IMREAD_COLOR)
        return frame

def send_coordinates(coordinates):
    serialized_coordinates = json.dumps(coordinates)
    serialized_coordinates += '\n'
    client_socket.sendall(serialized_coordinates.encode("utf8"))


def load_model(model_path):
    model = tf.saved_model.load(model_path)
    return model


def run_inference_for_single_image(model, image):
    image = np.asarray(image)
    # The input needs to be a tensor, convert it using
`tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image)
    # The model expects a batch of images, so add an axis with
`tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis,...]

    # Run inference
    output_dict = model(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch
dimension.
    # We're only interested in the first num_detections.
    num_detections = int(output_dict.pop('num_detections'))
    output_dict = {key: value[0, :num_detections].numpy()
                   for key, value in output_dict.items()}
    output_dict['num_detections'] = num_detections

    # detection_classes should be ints.
    output_dict['detection_classes'] =
output_dict['detection_classes'].astype(np.int64)

    # Handle models with masks:
    if 'detection_masks' in output_dict:
        # Reframe the the bbox mask to the image size.
        detection_masks_reframed =
utils_ops.reframe_box_masks_to_image_masks(
```

```
                                    output_dict['detection_masks'],
output_dict['detection_boxes'],
                                    image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(detection_masks_reframed >
0.5, tf.uint8)
        output_dict['detection_masks_reframed'] =
detection_masks_reframed.numpy()

    return output_dict


def run_inference(model, category_index):
    while True:
        image_np = receive_frames()
        # Actual detection.
        output_dict = run_inference_for_single_image(model, image_np)
        # Visualization of the results of a detection.
        vis_util.visualize_boxes_and_labels_on_image_array(
            image_np,
            output_dict['detection_boxes'],
            output_dict['detection_classes'],
            output_dict['detection_scores'],
            category_index,
            instance_masks=output_dict.get('detection_masks_reframed',
None),
            use_normalized_coordinates=True,
            line_thickness=8)
        boxes = np.squeeze(output_dict['detection_boxes'])
        scores = np.squeeze(output_dict['detection_scores'])
        min_score_thresh = 0.75
        bboxes = boxes[scores>min_score_thresh]

        h, w, _ = image_np.shape
        final_box =[]
        for box in bboxes:
            ymin, xmin, ymax, xmax = box
            final_box.append([int(xmin * w),int(xmax * w),int(ymin *
h),int(ymax * h)])
        print(final_box)
        send_coordinates(final_box)

        cv2.imshow('object_detection', cv2.resize(image_np, (800, 600)))
        if cv2.waitKey(1) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            break


if __name__ == '__main__':
```

```
    parser = argparse.ArgumentParser(description='Detect objects inside
webcam videostream')
    parser.add_argument('-m', '--model', type=str, required=True,
help='Model Path')
    parser.add_argument('-l', '--labelmap', type=str, required=True,
help='Path to Labelmap')
    args = parser.parse_args()


    detection_model = load_model(args.model)
    category_index =
label_map_util.create_category_index_from_labelmap(args.labelmap,
use_display_name=True)

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('158.42.165.137', 8080))


    run_inference(detection_model, category_index)
```