Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación Android y Servicio Web Spring para la detección y registro de señales de tráfico de velocidad usando Deep Learning con tiny-yolov3 y OpenCV

Autor: Lucía Reina López
Tutor: María Teresa Ariza Gómez y Antonio Jesús Sierra Collado

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación Android y Servicio Web Spring para la detección y registro de señales de tráfico de velocidad usando Deep Learning con tiny-yolov3 y OpenCV

Autor:

Lucía Reina López

Tutor:

María Teresa Ariza Gómez

Profesor titular

Antonio Jesús Sierra Collado

Profesor Sustituto Interino

Dpto. de Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado: Aplicación Android y Servicio Web Spring para la detección y registro de señales de tráfico de velocidad usando Deep Learning con tiny-yolov3 y OpenCV

Autor: Lucía Reina López

Tutor: María Teresa Ariza Gómez y
Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

A mi gran familia, que me han enseñado más de lo que ninguna escuela podrá enseñarme, a mis amigos, por apoyarme, escucharme y compartir todas mis experiencias, a Abril, que tiene la capacidad de alegrarme cualquier día, y por último, a mis tutores, María Teresa Ariza Gómez y Antonio Jesús Sierra Collado, por su colaboración, paciencia, consejos y su empeño en ayudarme en la realización de este proyecto.

Lucía Reina López

Coripe, 2020

Resumen

La Inteligencia Artificial (IA) ha presenciado un crecimiento monumental en la reducción de la brecha entre las capacidades de las personas y las máquinas. La IA es una rama de la ciencia, en concreto de la informática, que hace referencia a la capacidad de una máquina para resolver un problema tal y como lo haría una persona. Si además de resolver un problema tal y como lo haría una persona, queremos que la máquina aprenda por sí sola, entonces hablamos de un segundo término cuyo uso ha experimentado también un gran crecimiento: *Machine Learning*, o lo que es lo mismo, Aprendizaje Automático.

En los últimos años, el uso de estos dos términos, Inteligencia Artificial y Aprendizaje Automático, han experimientado un crecimiento exponencial, ayudando a resolver problemas como la detección de spam, recomendación personalizada de productos, diagnósticos médicos y detección de peatones e imágenes médicas con visión artificial entre otros. Sin embargo, hay otros problemas para los que no se ha encontrado solución todavía, como son los accidentes de tráfico. Según la Dirección General de Tráfico (DGT), en el último año documentado, 2018, en España, hubo un total de 102.299 accidentes con víctimas, lo que ha supuesto un 1% más con respecto del año 2017 [1]. Debemos añadir que, en estos accidentes, la velocidad excesiva o inadecuada sigue siendo una de las principales causas, concretamente, el 22%, sólo por detrás de la conducción distraída y por delante del alcohol [2].

En este proyecto se ha indagado en estas técnicas, en concreto se ha usado la tecnología del *Machine Learning* para la detección de objetos, en este caso, detección de señales de tráfico de velocidad desde el vehículo mientras es conducido en carretera, usando redes neuronales convolucionales como es Yolo (*You Only Look Once*). Se ha diseñado e implementado software que detecta el objeto deseado, su posición (longitud y latitud), velocidad máxima permitida y cercanía a través de la cámara de un dispositivo móvil. Para ello se ha creado una aplicación móvil en Android que envía la información recogida al servicio web Spring creado y queda almacenada en una base de datos para la libre consulta y uso de cada usuario de la aplicación.

Abstract

Artificial Intelligence (AI) has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. AI is a branch of computer science which refers to the ability of a machine to solve a problem just as a person would. If in addition to solving a problem just as a person would do, we want the machine to learn by itself, then we are talking about a second term whose use has also experienced great growth: Machine Learning.

In recent years, the use of these two terms, Artificial Intelligence and Machine Learning, have experienced exponential growth, helping to solve problems such as spam detection, personalized product recommendation, medical diagnostics, and pedestrian detection with artificial vision among others. However, there are other problems whose solutions have not yet been found, such as traffic accidents. According to Dirección General de Tráfico (DGT), in the last documented year, 2018, in Spain, there were a total of 102,299 accidents with victims, which was 1% more than in 2017 [1]. Furthermore, in these accidents, excessive or inadequate speed continues to be one of the main causes, specifically, 22%, only behind distracted driving and ahead of alcohol [2].

In this project, these techniques have been investigated, specifically Machine Learning technology has been used to detect objects, in this case, detect speed traffic signs from the vehicle while driving it, using convolutional neural networks such as Yolo (You Only Look Once). Software has been designed and implemented that detects the desired object, its position (longitude and latitude), maximum allowed speed and proximity through the camera of a mobile device. For this, a mobile application has been created on Android that sends the information collected to the Spring web service created and it is stored in a database for free consultation and use by each user of the application.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.

- Albert Einstein -

Usamos el Aprendizaje Automático decenas de veces al día sin darnos cuenta. Cada vez que realizamos una búsqueda en Google, funciona bien porque su software de Aprendizaje Automático ha aprendido a indexar las páginas. Cuando Facebook u otras aplicaciones reconocen a nuestros amigos en las fotos, usan aprendizaje automático. Cada vez que leemos el correo electrónico y un filtro de spam evita que tengamos que leer correo no deseado, es porque nuestro ordenador ha aprendido a distinguir entre correo deseado y no deseado. Todo esto es *Machine Learning*, la ciencia que posibilita que un ordenador aprenda sin ser explícitamente programado, que adquiere día a día más importancia gracias a la gran variedad de aplicaciones que tiene.

En los diversos capítulos que conforman este Trabajo de Fin de Grado se explicará qué es el Aprendizaje Automático, las redes neuronales, y la solución final adoptada para aplicar estos conocimientos a la idea de un vehículo más seguro y autónomo.

1.1 Motivación y Objetivos

Este Trabajo de Fin de Grado se ha realizado con el propósito de diseñar un sistema que proporcione al vehículo una ayuda en la conducción, con la finalidad de hacerlo más seguro y autónomo. El sistema desarrollado permitirá:

- Detectar señales de tráfico de velocidad en carretera a tiempo real gracias al uso del Aprendizaje Automático y las redes neuronales artificiales.
- Permitir al usuario recopilar estos datos y consultarlos.
- Informar al conductor de la velocidad que deberá respetar en cada tramo de su recorrido.
- Avisar al conductor si se supera la velocidad máxima permitida.
- Hacer uso de los datos recopilados entre todos los usuarios para la señalización de velocidad en la navegación GPS.

Para desarrollarlo, se han hecho uso de múltiples tecnologías, con el objetivo de indagar en el funcionamiento de las mismas y en las posibilidades que ofrecen. Entre ellas destacamos YOLO (*You Only Look Once)*, que es una red neuronal capaz de detectar objetos a tiempo real, además de ser uno de los algoritmos más rápidos, precisos y completos. Spring, un framework basado en Eclipse para el desarrollo de aplicaciones Java, con el que se ha desarrollado el servicio web de este proyecto. Es el marco de trabajo más popular y extendido dentro

del desarrollo de aplicaciones web debido a su facilidad de uso. Por último, Android, para el desarrollo de una aplicación móvil a través de la cuál el usuario podrá acceder a todas las funcionalidades que ofrecerá la solución final de este proyecto. La continua evolución, desarrollo y el incremento del uso de los teléfonos inteligentes en el día a día de cada ser humano, han logrado que se conviertan en objetos casi imprescindibles en nuestras vidas, llevándolos siempre con nosotros. Android es el sistema operativo de Google que da vida a casi el 90% de los *smartphones* de la actualidad, estando en constante evolución y desarrollo en este ámbito.

1.2 Antecedentes

Existen diversas aplicaciones móviles que permiten obtener información de las señales de tráfico a través de un mapa previamente cargado y haciendo uso del GPS del dispositivo móvil, además de diversos coches que cuentan con esta funcionalidad incorporada también. Estas aplicaciones no cuentan con la ventaja que el Aprendizaje Automático nos proporciona gracias a la detección de objetos en tiempo real ya que las señales, rutas y otros factores en carreteras cambian debido a diversas causas como las obras, entre otros.

Además se parte de la base de otros proyectos realizados para dotar de más autonomía y seguridad a la conducción del vehículo, como son los proyectos realizados por Sergio Mellado Contioso titulado *Aplicación Android y Servicio Web Spring para la monitorización de datos obtenidos en un vehículo haciendo uso de la plataforma FIWARE* [3], Álvaro Carmona Palomares con *Securización mediante Keyrock y Wilma de Aplicación y Servicios Web para Vehículo Inteligente* [4], Luis Martínez Ruíz con *Aplicación web con Freeboard para la recolección de datos de sensores de acelerómetro y gps mediante Orion Context Broker de Fiware* [5] y Pablo Bermejo Pérez con *Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring* [6], en los cuáles se desarrolla la idea del vehículo conectado gracias a la tecnología del Internet de las Cosas (*Internet Of Things, IoT).*

1.3 Descripción de la solución

Describimos a continuación la solución adoptada para poder alcanzar los objetivos propuestos en este proyecto.

1.3.1 Objetivos específicos

- **Obtención de los datos desde el dispositivo móvil.**

  Desde la aplicación móvil Android se usará la red neuronal previamente entrenada YOLO y la cámara del móvil para la detección de señales de tráfico de velocidad, obteniendo la velocidad máxima permitida, su posición (longitud y latitud), dimensiones con respecto a la imagen (alto y ancho) y usuario que ha realizado la detección.

- **Envío y almacenamiento de los datos.**

  La aplicación Android mantiene una comunicación con el servicio web Spring, el cuál procesa los datos recibidos y los almacena en una base de datos MySQL. Almacenará las señales de tráfico según la siguiente lógica: Sólo habrá un señal de tráfico en la misma posición, esta será, la mayor de todas, que es aquella cuyas dimensiones sean mayores con respecto a la imagen para asegurar que la posición de la señal en el mapa es lo más precisa posible. Además se almacenarán las mejores detecciones del usuario individual que usa la aplicación, y por otro lado las mejores detecciones escogidas entre todos los usuarios de la aplicación.

- **Visualización de los datos.**

  En la aplicación Android es posible visualizar los datos gracias a *Maps SDK for Android* [21]. El usuario podrá ver las distintas señales de tráfico almacenadas en la base de datos en su posición correspondiente en el mapa. Además, podrá elegir si visualizar las mejores detecciones hechas por sí mismo mediante el uso de la aplicación, o si ver las detecciones realizadas entre todos los usuarios.

- **Uso de los datos.**

  Los datos almacenados por todos los usuarios podrán ser usados para la navegación GPS. Podrán trazar rutas hasta sus destinos, guiarse por ellas durante la conducción y además usar los datos almacenados

para ser advertidos de las señales de tráfico que deben respetar en cada tramo de su recorrido.

- **Registro e inicio de sesión de los usuarios.**

  Para poder acceder a la aplicación web y todas sus funcionalidades, es necesario realizar un registro previo y luego iniciar sesión en la aplicación Android. Toda la información de los usuarios registrados se almacenará también en la base de datos MySQL, así como la información sobre las sesiones que mantienen abiertas.

1.3.2 Funcionalidades

El sistema presenta las siguientes funcionalidades para cada usuario que haga uso de él a través de la aplicación Android:

- Registro como nuevo usuario.
- Autenticación e inicio de sesión de usuario.
- Cierre de la sesión del usuario que previamente se ha autenticado e iniciado sesión.
- Detección de señales de tráfico de velocidad a través de la cámara del teléfono móvil y obtención de su posición (longitud y latitud), dimensiones relativas a la imagen (alto y ancho), tipo de señal de velocidad y usuario que la ha detectado.
- Registro de las detecciones en servidor Spring y base de datos MySQL.
- Aviso por pantalla de la detección de una señal, esto es, la velocidad máxima permitida en ese momento.
- Aviso por pantalla de la velocidad en cada instante.
- Aviso sonoro y por pantalla si la velocidad en un instante supera la velocidad máxima permitida.
- Consulta de las detecciones registradas. Se pueden consultar tanto las detecciones registradas del usuario individual como las detecciones en conjunto de todos los usuarios de la aplicación.
- Consulta/búsqueda de detecciones registradas por ciudad o zona.
- Navegación GPS. El usuario podrá consultar rutas desde un punto A a un punto B introducidos, mostrando en cada momento la ubicación actual en el recorrido. Así mismo, también contará con avisos sobre las señales de tráfico previamente registradas por los usuarios que se encontrara en su recorrido, sobre la velocidad en cada instante y avisos si esta sobrepasase la velocidad máxima permitida.
- Consulta de las señales registradas provisionalmente en la base de datos SQLite debido a pérdidas de conexión a la red, o pérdidas de conexión con el servidor. Estos datos no registrados permanecen recogidos en la base de datos local SQLite hasta que sea posible volver a establecer conexión con el servicio web Spring.

1.3.3 Esquema de la arquitectura

En la Ilustración 1 podemos ver el esquema de la arquitectura, con el que se da una idea de todos los componentes principales que participan en la solución final adoptada para este proyecto y de sus interacciones:

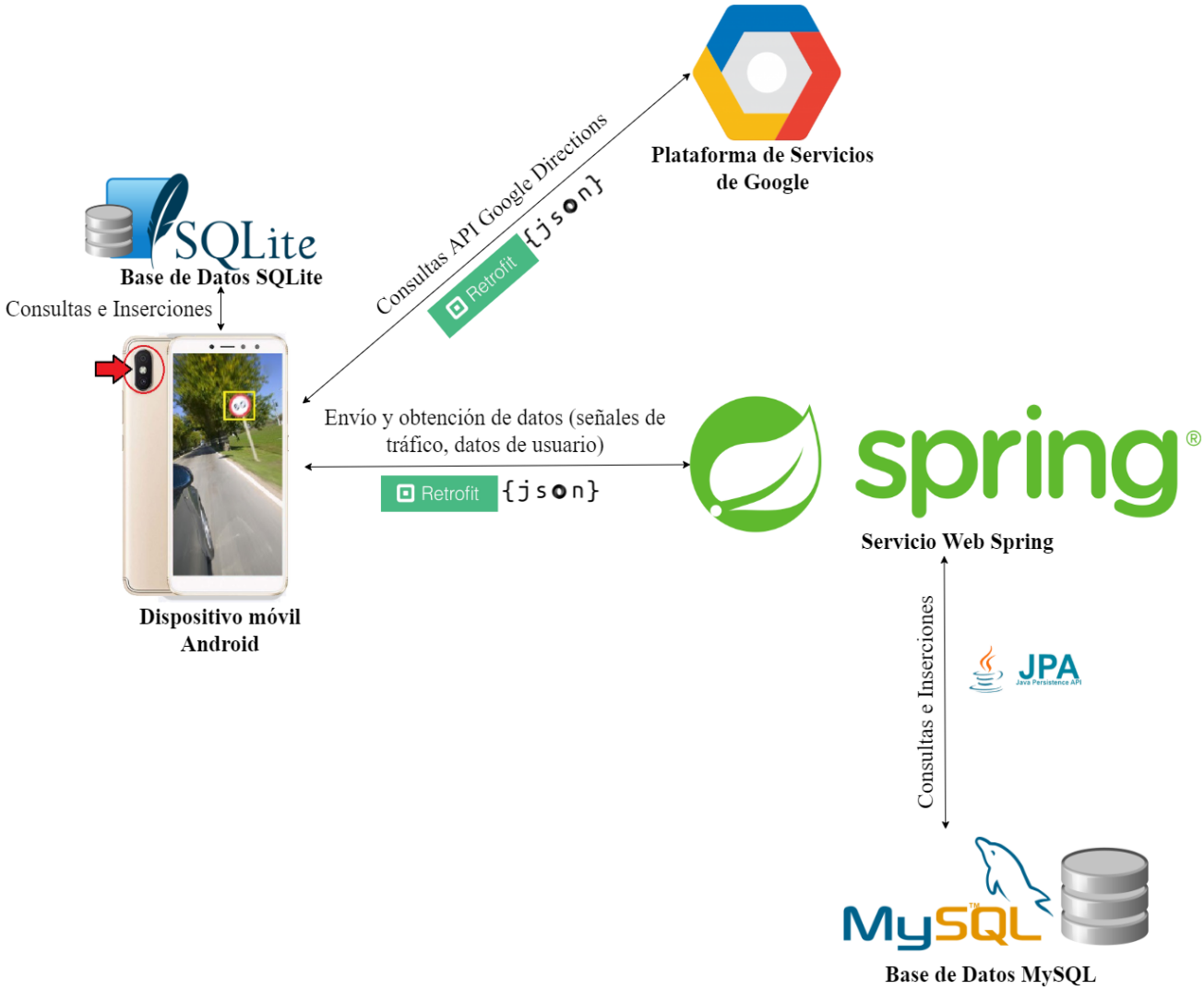

Ilustración 1: Esquema de la Arquitectura

1.4 Estructura de la memoria

A continuación, se expone un breve resumen de las partes que contiene esta memoria para que pueda servir como guía y facilitar así la lectura de la misma:

1. **Introducción:** Presentación del problema y objetivos a alcanzar, así como la solución adoptada para satisfacer los mismos.
2. **Recursos utilizados:** Presentación y breve descripción de los recursos hardware y los recursos software usados en este proyecto.
3. **Tecnologías utilizadas:** Presentación y breve descripción de las tecnologías usadas en la realización de este proyecto.
4. **Estado del arte: YOLO, Real-Time Object Detection:** Presentación de la investigación realizada sobre el estado del arte en la detección de objetos usando redes neuronales, además de profundizar en el funcionamiento del sistema de detección de objetos a tiempo real YOLO (*You Only Look Once*).
5. **Entrenamiento de la red neuronal tiny-yolov3:** Explicación del proceso de entrenamiento de la red neuronal escogida para hacerla capaz de llevar a cabo la tarea de detectar señales de velocidad.
6. **Servicio web Spring y Base de datos:** Descripción de la base de datos y servicio web Spring en detalle, así como el proceso de creación de estos.

7. **Aplicación Android iCodriver:** Se detalla el diseño y la interfaz de usuario de la aplicación Android desarrollada, *iCodriver*.
8. **Conclusiones y líneas futuras:** Se exponen nuevas ideas para mejorar o completar el sistema diseñado en versiones futuras.

2 RECURSOS UTILIZADOS

Los grandes conocimientos engendran las grandes dudas.

- Aristóteles -

En este capítulo se describen brevemente los recursos, tanto hardware como software, utilizados en este proyecto. Se expondrán sus características más relevantes, así como el uso que se les ha dado para la realización de este Trabajo Fin de Grado.

2.1 Recursos hardware

2.1.1 Ordenador Portátil

Ordenador portátil usado para el despliegue del servicio web y la base de datos para las pruebas, además de para el desarrollo de todos las partes de este Proyecto.

Ilustración 2: Intel ® Core ™ i7

- Modelo: Asus X555LJ.
- Procesador: Intel® Core™ i7-5500U CPU @ 2.4GHz.
- Memoria RAM: 8GB.
- Tarjeta Gráfica: NVIDIA GeForce 920m.
- Almacenamiento: 1TB HDD + 256GB SSD.
- Sistema Operativo: Windows 10 Home 64bits.

2.1.2 Teléfono Móvil Xiaomi Redmi S2

Usado para alojar la aplicación Android desarrollada.

Ilustración 3: Teléfono Móvil Xiaomi Redmi S2

- Modelo: M1803E6G.
- Sistema operativo: Android.
- Tamaño pantalla: 5.99", 720 x 1440 pixels
- Procesador: Snapdragon 625 2GHz (8 núcleos).
- Memoria RAM: 3GB.
- Almacenamiento: 32GB.
- Batería: Litio 3080mAh.

2.2 Recursos software

2.2.1 Navegador Web Opera

Opera es un navegador web creado por la empresa noruega Opera Software. Opera ha sido pionero en originar características que han sido posteriormente adoptadas por otros navegadores web, como por ejemplo el Acceso Rápido (Speed Dial), además de muchas otras con las que no cuentan otros navegadores.

Ilustración 4: Navegador Opera

2.2.2 Spring Tool Suite

Spring Tool Suite (STS) es un entorno de desarrollo (IDE) gratuito y de código abierto basado en Eclipse que está optimizado para desarrollar proyectos basados en el framework Spring. Puede instalarse como un IDE independiente o como un complemento en Eclipse.

En nuestro caso, como ya teníamos Eclipse, instalamos STS como un complemento a través de Eclipse Marketplace, ya que es más rápido que descargar STS por separado.

Mediante Spring Tool Suite 4 [7] se desarrolla el servicio web basado en el uso de Spring con JPA.

Ilustración 5: Spring Tool Suite 4

2.2.3 Eclipse

Eclipse es un entorno de desarrollo integrado, de código abierto y multiplataforma. Es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java [8]. El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.

Ilustración 6: Eclipse

2.2.4 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y está basado en IntelliJ IDEA de JetBrains [9]. A diferencia de Eclipse, Android studio utiliza un sistema de compilación flexible basado en Gradle.

Se ha utilizado para el completo desarrollo de la aplicación Android *iCodriver,* encargada de la detección y clasificación de las señales captadas, además de la comunicación con el servicio web y la base de datos.

Ilustración 7: Android Studio

2.2.5 MySQL Workbench

MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software,

administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL [10].

MySQL Workbench ha sido usada para la administración de la base de datos creada para el almacenamiento de las detecciones de señales realizadas por los usuarios desde la aplicación Android, así como los datos de cada usuario.

Ilustración 8: MySQL Workbench

2.2.6 LabelImg

LabelImg es una herramienta de anotación de imágenes gráficas escrita en Python y que usa Qt para su interfaz gráfica [11]. Las anotaciones se guardan como archivos XML en formato PASCAL VOC, el formato utilizado por ImageNet. Además, también es compatible con el formato YOLO, generando archivos txt que necesitamos para etiquetar las imágenes de entrada con las que alimentaremos nuestra red neuronal YOLO para su entrenamiento.

Ilustración 9: LabelImg

2.2.7 Darknet

Darknet es un marco de trabajo de red neuronal de código abierto escrito en C y CUDA [12]. Es rápido, fácil de instalar y admite cálculos de CPU y GPU.

Se ha utilizado como marco de trabajo para entrenar nuestra red neuronal YOLO, estableciendo la arquitectura de la misma.

Ilustración 10: Darknet framework

3 TECNOLOGÍAS UTILIZADAS

La tecnología hizo posible las grandes poblaciones;
ahora las grandes poblaciones hacen que la tecnología
sea indispensable.

- José Krutch -

En este capítulo se describen brevemente las tecnologías utilizadas en este proyecto, así como el uso que se les ha dado en este Trabajo de Fin de Grado.

3.1 Spring

Spring es un framework y un contenedor de inversión de control e inyección de dependencias para la plataforma de Java. Posee estructura modular y gran flexibilidad para implementar diferentes tipos de arquitectura según la aplicación [13].

Se ha utilizado la tecnología Spring para realizar el desarrollo del servicio web. Para la ejecución se ha usado Spring Boot, conjunto de herramientas para construir y ejecutar las aplicaciones Spring.

Tanto Spring como Spring boot están integradas dentro de Spring Tool Suite 4.

Ilustración 11: Spring

3.2 Java Persistence API

JPA (*Java Persistence API*) es la API de persistencia desarrollada por la plataforma Java EE definida en el paquete *javax.persistence.* Permite establecer una correlación entre una base de datos relacional y un sistema orientado a objetos estableciendo una interfaz común que es implementada por una base de datos de nuestra elección. Se encarga de automatizar dentro de lo posible la persistencia de nuestros objetos en una base de datos para no perder las ventajas de la orientación a objetos al interactuar con una base de datos [14]. Destacamos los siguientes conceptos sobre el uso de JPA:

- Uso de anotaciones como *@Entity*, *@Table* o *@Column* entre otras para especificar propiedades.

- Entidades persistentes y relaciones entre entidades.
- Mapeado Objeto-Relacional.
- Lenguaje de consulta Java Persistence Query Language (JPQL).

Ilustración 12: Java Persistence API

3.3 MySQL

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo [15].

Ilustración 13: MySQL

3.4 SQLite

SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones [16]. Es de dominio público y por lo tanto libre para el uso para cualquier propósito, comercial o privado.

SQLite está integrado en todos los teléfonos móviles y en la mayoría de las computadoras y viene incluido dentro de innumerables otras aplicaciones que la gente usa todos los días.

Se ha usado como base de datos provisional en la aplicación Android en el caso de que esta no pudiese comunicarse con el servidor web debido a la pérdida de conexión de red.

Ilustración 14: SQLite

3.5 Android

Android es un sistema operativo basado en Linux que está diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tabletas, admite una gran cantidad de aplicaciones en teléfonos inteligentes y es de código abierto, lo que significa que es gratuito y cualquiera puede usarlo.

Ilustración 15: Android

3.6 REST

REST (Representational State Transfer, o Transferencia de Estado Representacional) es una interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.

Esta tecnología ha sido usada para desarrollar un servicio web Restful, es decir, basado en la arquitectura REST.

3.7 JSON

JSON (acrónimo de *JavaScript Object Notation*) es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

Nuestro servidor REST usa JSON para el intercambio de datos.

Ilustración 16: JSON

3.8 Retrofit

Retrofit es un cliente REST para Java y Android [17]. Hace que sea relativamente fácil recuperar y cargar JSON (u otros datos estructurados) a través de un servicio web basado en REST. En Retrofit, se configura qué convertidor se usa para la serialización de datos. Por lo general para JSON se usa GSon, pero se pueden agregar otros convertidores personalizados para procesar XML u otros protocolos. Retrofit usa la biblioteca OkHttp para solicitudes HTTP.

Se ha usado Retrofit en la aplicación Android para poder enviar solicitudes de red con OkHttp y así poder comunicarnos con el servidor REST.

Ilustración 17: Retrofit

3.9 OpenCV

OpenCV (*Open Source Computer Vision Library)* es una biblioteca de software de visión artificial y aprendizaje automático de código abierto. OpenCV se creó para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto libre, con licencia BSD, OpenCV facilita que las empresas utilicen y modifiquen el código [18].

OpenCV ofrece soporte para varios sistemas operativos, entre ellos Android. Se ha usado en nuestra aplicación Android, junto con la red neuronal YOLO para la detección y clasificación de objetos en las imágenes captadas por la cámara del dispositivo móvil.

Ilustración 18: OpenCV

3.10 YOLO

You only look once (YOLO) es un sistema de detección de objetos en tiempo real de última generación [19]. Es la red neuronal elegida para nuestro proyecto que será entrenada y usada para la detección de señales de tráfico de velocidad a tiempo real. YOLO se explicará en más detalle en el capítulo 4.

Ilustración 19: YOLO

3.11 Google Colab

Google Colab es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube. Permite el uso gratuito de las GPUs y TPUs de Google, con librerías como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV [20].

Se ha usado en este proyecto para entrenar nuestra red neuronal tiny-yolov3 ya que podemos usar la potente 12GB NVIDIA Tesla K80 GPU durante 12h seguidas.

Ilustración 20: Google Colab

3.12 Google Cloud Platform

Google Cloud Platform (GCP) es una plataforma que ofrece más de 90 servicios/productos a empresas, profesionales y desarrolladores. Te permite crear, implementar y escalar aplicaciones, sitios web y servicios en la misma infraestructura que Google.

En concreto, haremos uso de los servicios de *Maps SDK for Android* [21] y *Directions API* [22] para la implementación de nuestra aplicación Android, los cuáles ofrecen imágenes de mapas desplazables, rutas y direcciones entre dos puntos del mapa.

Ilustración 21: Google Cloud Platform

3.13 AWS

Amazon Web Services (AWS) es la plataforma en la nube más adoptada y completa en el mundo, que ofrece más de 175 servicios integrales de centros de datos a nivel global [23]

Contendrá el servicio web Spring desarrollado y la base de datos.

Ilustración 22: Amazon Web Services

4 ESTADO DEL ARTE: YOLO, REAL-TIME OBJECT DETECTION

El cambio es siempre el resultado final de todo verdadero aprendizaje. Aprender no es prepararse para la vida. Aprender es la vida misma.

- John Dewey -

En este capítulo se presenta la investigación realizada sobre el estado del arte en la detección de objetos usando redes neuronales, además de profundizar en el funcionamiento del sistema de detección de objetos a tiempo real YOLO (*You Only Look Once*) usado en este Trabajo de Fin de Grado.

El campo de estudio que da a las máquinas la habilidad de aprender sin ser programadas de manera explícita (o *Machine Learning*) cuenta con diversas técnicas y algoritmos que podemos clasificar de dos formas: algoritmos supervisados y no supervisados, y algoritmos de regresión y clasificación.

Los algoritmos supervisados son los más comunes, son aquellos que aprenden a partir de un conjunto de datos correctamente etiquetados, esto es, dada una determinada entrada, le indicamos cuál debería ser la salida correcta, o, dicho de otro modo, le enseñamos al algoritmo lo que está bien y lo que está mal para que aprenda. Por otro lado, los algoritmos no supervisados, recibirán entradas sin etiquetar y buscarán alguna estructura o configuración presente en los datos o detectarán similitudes entre las entradas proporcionadas para separarlas en grupos a su propia elección.

En cuanto a los algoritmos de regresión y clasificación, los primeros tienen el objetivo de predecir valores continuos, como, por ejemplo, precios de viviendas en función del número de habitaciones, metros cuadrados, etc. Los algoritmos de clasificación en cambio predicen valores discretos. Proporcionan como salida la clase a la que pertenecen un conjunto de datos dados como entrada al algoritmo.

En este proyecto, vamos a tratar con algoritmos supervisados y de clasificación, pues proporcionaremos como entrada imágenes de señales de tráfico de velocidad etiquetadas con su clase correspondiente, para que el algoritmo o red neuronal aprenda a detectarlas y clasificarlas correctamente.

A continuación, se explicará el funcionamiento más básico de una red neuronal, después se procederá a comentar lo que son las Redes Neuronales Convolucionales (CNN, *Convolutional Neural Network*) para después hacer énfasis en las Redes Neuronales Convolucionales basadas en Regiones (R-CNN, *Region-based Convolutional Neural Networks*), asentando así las bases de la red neuronal utilizada en nuestro proyecto, YoloV3.

4.1 Redes neuronales artificiales

Las redes neuronales artificiales son modelos computacionales basados en el comportamiento de las redes neuronales biológicas que tienen la capacidad de aprender y formarse a si mismas de forma autónoma.

Una neurona arficial es la unidad básica de procesamiento que se puede encontrar dentro de la red neuronal, representada en la Ilustración 23:

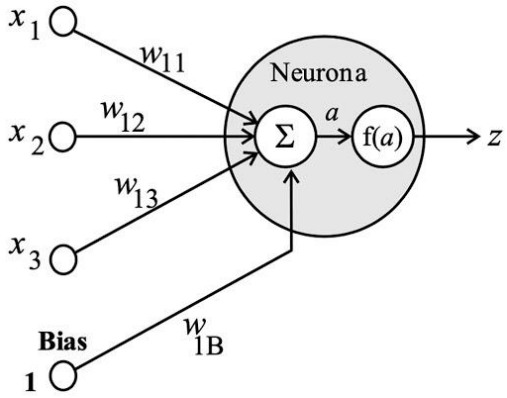

Ilustración 23: Esquema Neurona Artificial

Como se observa, estas reciben un conjunto de entradas, calcula la suma ponderada de las mismas y devuelve un valor de salida. La razón por la que es ponderada es debido a que cada entrada se multiplica por un peso asociado, que definirá la importancia relativa de la misma.

Las redes neuronales consisten en un conjunto de estas neuronas artificiales organizadas en capas, como se observa en la Ilustración 24.

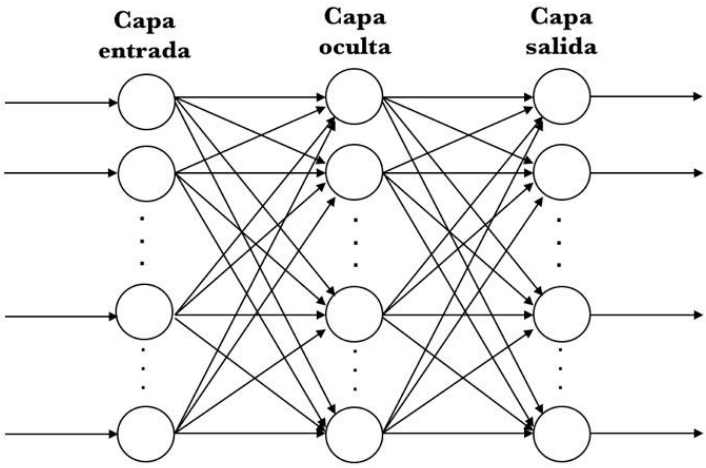

Ilustración 24: Esquema de una red neuronal

Tradicionalmente, a la primera capa se le conoce como capa de entrada, y la última como capa de salida. A todas las capas del medio se las conoce como capas ocultas totalmente conectadas (*fully-connected*). En esta estructura comentada, todas las neuronas se encuentran completamente conectadas, es decir, por cada dos capas de neuronas adyacentes, cada par de neuronas tiene una conexión por la que se transmite información. Por ejemplo, si una capa tiene m neuronas y la siguiente n, el número total de conexiones sería m*n. Esto significa que las salidas de una capa son las entradas de cada una de las neuronas de la capa adyacente, por lo que el cálculo de una neurona dependería de los valores de salida de su capa anterior, realizándose este una única vez.

Las capas pueden tener distintos tamaños y cada neurona distintas funciones incluso dentro de la primera capa. Todo esto dependerá del problema. La capa de entrada poseerá tantas neuronas como componentes de entrada haya y la capa de salida tantas neuronas como valores se quiera generar. En el proceso, cada capa va tomando una serie de decisiones y cuanto más profunda sea la red, más inteligentes y complejas serán las decisiones ya que se irán basando en las tomadas por las anteriores capas. Esto se conoce como aprendizaje profundo (Deep Learning) y gracias a este se han logrado resolver problemas complejos como el de reconocimiento del habla, el procesado de lenguaje natural o la creación de sistemas de recomendación.

4.2 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) se distinguen del resto de redes neuronales por usar capas convolucionales, que realizan la operación llamada convolución que les da nombre. Las CNN son redes neuronales muy efectivas para la detección de objetos y clasificación y segmentación de imágenes.

En la Ilustración 25, podemos ver que consta de diversas capas: capa de entrada, capa de convolución, capa de reducción y capa de conexión total.

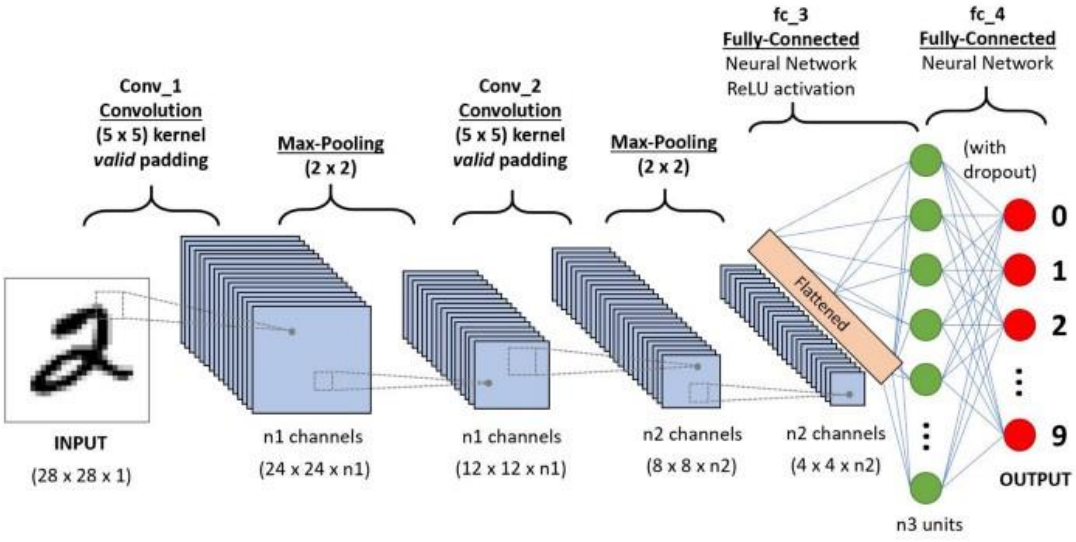

Ilustración 25: Esquema Red Neuronal Convolucional

4.2.1 Capa de entrada

Contiene los valores brutos de la imagen, teniendo un volumen igual a las dimensiones de esta.

4.2.2 Capa de convolución

En esta capa se realiza la operación de convolución que da nombre a estas redes neuronales. Esta capa posee un filtro de convolución, es decir, una matriz llena de pesos que, incialmente, poseen valores aleatorios, pero durante el entrenamiento, se irán ajustando para detectar patrones dentro de la imagen. Así, las primeras capas pueden detectar formas simples como líneas y curvas, y se van especializando hasta llegar a capas más profundas donde se reconocen formas complejas como la forma de una cara o un animal. La idea detrás de esto es que cada filtro aprende un atributo concreto y lo busca a través de toda la imagen:

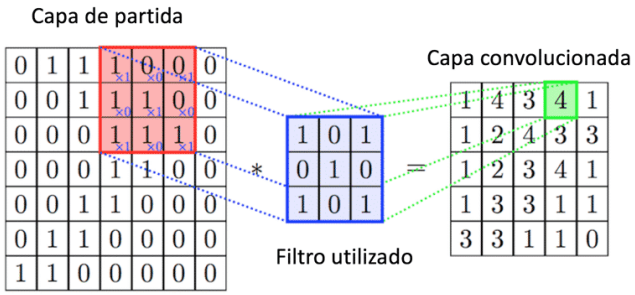

Ilustración 26: Operación de Convolución

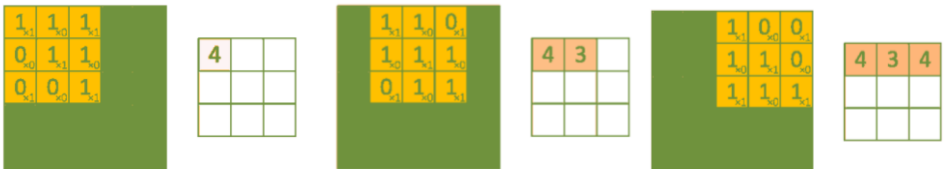

Ilustración 27: Pasos Operación de Convolución

Como vemos en la Ilustración 26 y la Ilustración 27, dada una matriz inicial y un filtro de convolución, en este caso de 3x3, para calcular la matriz resultante se debe ir multiplicando cada submatriz de la imagen de entrada por el filtro de convolución, obteniendo como resultado una matriz más reducida que la original que filtre una serie de características de la matriz de entrada.

Añadir, que la forma en que recorre el filtro la matriz inicial depende de dos parámetros que son el paso (*stride)* y el relleno (*padding)*.

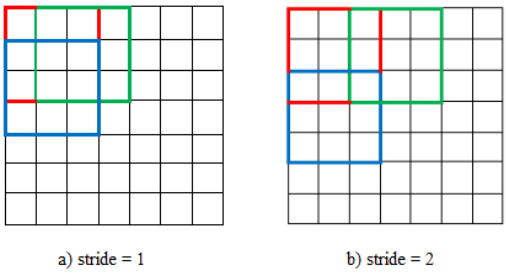

Ilustración 28: Parámetro *stride*

El parámetro paso (*stride)* se refiere al tamaño de salto que da el filtro al moverse por la imagen.

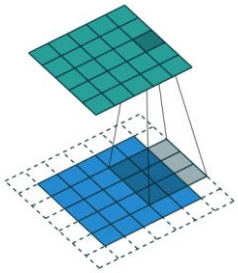

Ilustración 29: Parámetro *padding*

Por otro lado, el parámetro relleno (*padding)* no es más que un contorno de ceros alrededor de la matriz para solventar el problema de que los laterales de la imagen quedan menos observados que los valores de su interior.

4.2.3 Capa de reducción o *pooling*

La capa de reducción o *pooling* es responsable de reducir el tamaño espacial de la matriz resultante de la operación de convolución, y con ello, reducir también la carga computacional requerida para procesar los datos. Además, es útil para extraer características dominantes que son invariantes rotacionales y posicionales, manteniendo así el proceso de entrenamiento efectivo del modelo.

Los dos modelos que más se utilizan son *average-pooling* y *max-pooling* y se muestran en la Ilustración 30.

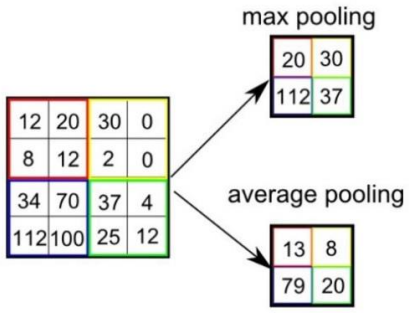

Ilustración 30: Tipos de *pooling*

Max-pooling devuelve los valores máximos de las partes de la imagen seleccionadas, mientras que *average-pooling* devuelve un promedio de estos valores.

4.2.4 Capa de conexión total (*fully connected*)

También conocida como capa clasificadora, se encarga de calcular las puntuaciones de cada categoría o clase a la que puede pertenecer la imagen de entrada, lo que dará como resultado un vector con una profundidad igual al número de clases. Entendiendo como clases en este caso, las distintas categorías o número de objetos distintos que podemos detectar en una imagen.

En la Ilustración 31, se puede ver como todas las neuronas de la capa están conectadas a todas las neuronas de la capa anterior, de ahí su nombre, capa de conexión total. Se observa también que, primero, se toman los resultados del proceso de convolución y se aplanan en un solo vector de valores, cada uno de los cuales representa la probabilidad de que una determinada característica pertenezca a una clase. Después pasan a través de una función de activación (típicamente ReLU, *Rectified Lineal Unit),* y por último obtenemos la salida, en la que cada neurona representa las puntuaciones de cada clase. La función de activación ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran. Cuando procesamos una imagen, cada capa de convolución debe capturar algún patrón en la imagen y pasarla en la siguiente capa de convolución. Los valores negativos no son importantes en el procesamiento de imágenes y, por lo tanto, se establecen en 0. Pero los valores positivos después de la convolución deben pasar a la siguiente capa.

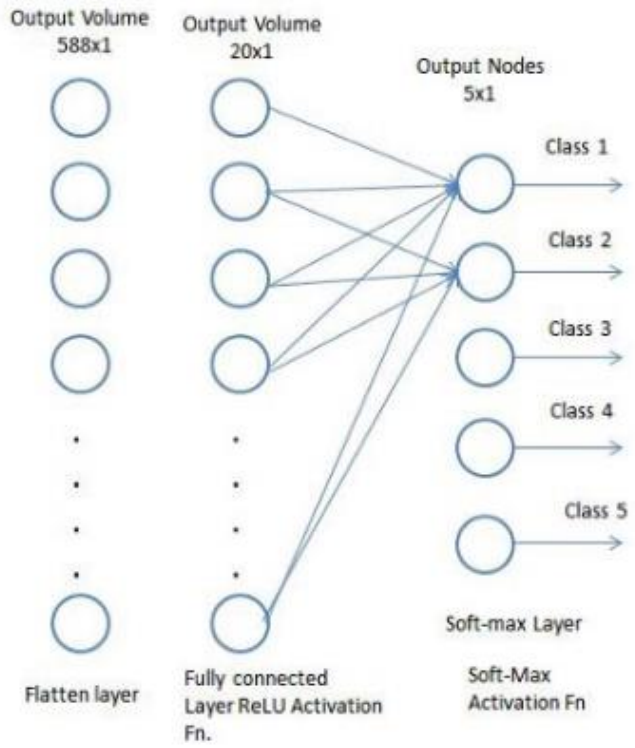

Ilustración 31: Capa de Conexión total o Clasificadora

Las capas totalmente conectadas han demostrado ser muy exitosas en el reconocimiento y clasificación de imágenes para visión por ordenador.

4.3 YOLOv3

You only look once (YOLO) es un sistema de detección de objetos a tiempo real [19]. Es un tipo de R-CNN (Region-Based CNNs), esto es, redes neuronales convolucionales basadas en regiones, que son un tipo de redes neuronales cuya idea principal es detectar objetos dentro de subregiones de la imagen principal.

Tal y como su nombre indica, Yolo solo requiere "ver" la imagen una sola vez, consiguiendo gran velocidad a cambio de una pequeña pérdida de precisión. En comparación con algoritmos de reconocimiento, un algoritmo de detección no solo predice la clase a la que pertenece el objeto, sino también la localización del objeto en la imagen. YoloV3 es capaz de detectar múltiples objetos en una imagen a través de una única red neuronal. Además, es uno de los algoritmos de detección más rápidos, posicionándose en una de las mejores elecciones cuando se busca detección en tiempo real.

YoloV3 es la última versión de Yolo cuya fecha de salida tuvo lugar en 2018. Hace uso de una única red neuronal convolucional llamada Darknet-53, ya que posee 53 capas, 24 convolucionales de 3x3, seguidas cada una por dos de conexión completa, además de capas de reducción de 1x1 usadas por algunas de las capas convolucionales para reducir la profundidad de los mapas de características. Se puede observar la estructura de Darknet-53 en la Ilustración 32.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Ilustración 32: Darknet-53

A continuación, veremos el funcionamiento de YoloV3.

4.3.1 Funcionamiento

Como ya hemos mencionado anteriormente, Yolo es capaz de detectar la posición y la clase a la que pertenece un determinado objeto. Esto lo hace de la siguiente manera:

1. Nuestro sistema divide la imagen de entrada en una cuadrícula de S × S celdas, tal como se muestra en la Ilustración 33. Si el centro de un objeto cae en una celda de la cuadrícula, esta es responsable de detectar ese objeto.

Ilustración 33: Cuadrícula SxS

2. En cada celda de la matriz se predice la posibilidad de que haya un objeto dentro de la misma y se crea un cuadro delimitador (*bounding-box)* que rodearía a ese posible objeto junto a las celdas adyacentes.

Cada cuadro delimitador consta de 5 predicciones: x (posición horizontal), y (posición vertical), w (ancho), h (alto), y confianza. Cada celda podrá contener hasta un número B de cuadros delimitadores. El cuadro delimitador no da información sobre la clase del objeto en cuestión, pero sí una confianza (*confident score*) que refleja qué tan seguro está el modelo de que el cuadro delimitador contiene un objeto y también cómo de preciso cree que es el cuadro. En la Ilustración 34, el grosor de los cuadros delimitadores será mayor cuanto más alta sea la confianza.

Ilustración 34: Cuadros delimitadores y confianza

3. Por cada cuadro delimitador, se predice el objeto que puede haber dentro del mismo. Ahora estos pasarán a estar etiquetados según la clase que contienen, como se observa en la imagen de la Ilustración 35.

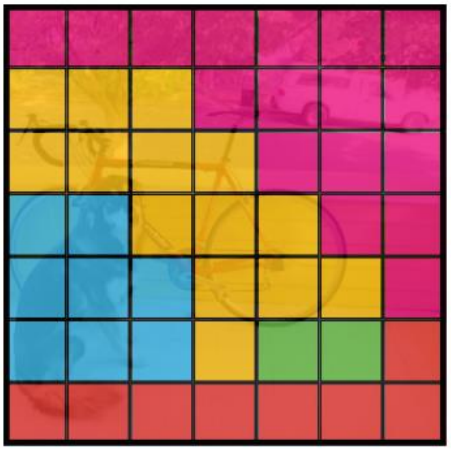

Ilustración 35: Predicción de clases en cada cuadro

4. Finalmente, se fija un umbral mínimo de probabilidad que debe tener un cuadro delimitador para que realmente contenga una clase y se eliminan todos los cuadros que no superen ese umbral. Esto devolvería el resultado esperado, que se puede ver en la Ilustración 36.

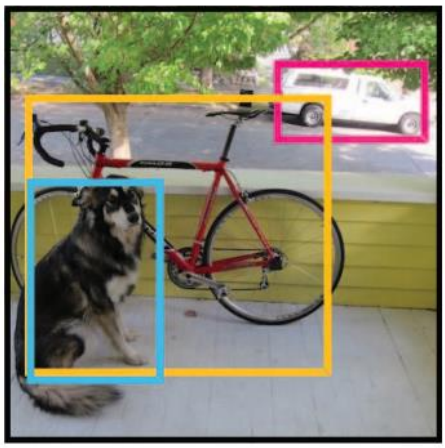

Ilustración 36: Predicción final

4.3.2 Ventajas

Aquí enumeraremos algunas de sus ventajas y motivos por los que hemos escogido Yolo para nuestro proyecto:

- Extremadamente rápido, se ejecuta a 45 frames por segundo en una GPU Titan X y de 220 fps en su versión ligera (tiny-yolov3). Esto significa que podemos procesar la transmisión de vídeo en tiempo real con menos de 25 milisegundos de latencia. Además, YOLO logra más del doble de la precisión promedio de otros sistemas en tiempo real.
- Elevado porcentaje de aciertos.
- A diferencia de otros sistemas de clasificación, Yolo considera la imagen completa, de modo que sus predicciones tienen en cuenta el contexto completo de la imagen, es decir, codifica implícitamente información contextual sobre las distintas clases además de sus características.
- Hace predicciones con una sola red neuronal, mientras que sistemas como R-CNN requieren miles para una sola imagen, lo cuál hace a Yolo extremadamente rápido.
- Proporciona la posición del objeto detectado en la imagen.
- Integración sencilla con OpenCV.

Como ya se ha mencionado anteriormente, Yolo se encuentra en la versión 3, que es la versión que usaremos en este proyecto. Con respecto a las versiones anteriores, presenta mejoras en cuanto a la eficiencia del algoritmo y una mayor configuración.

En concreto, en este proyecto se ha usado la versión ligera de YoloV3, esto es, Tiny-yoloV3. Las principales diferencias entre ambas arquitecturas son la cantidad de capas que las componen y el número de operaciones de coma flotante que realizan por segundo (flops), calculando la versión normal 38.97 billones y la ligera 5.56 billones. Todo esto provoca que la versión ligera obtenga una precisión más baja y pueda detectar un número menor de clases (la versión normal puede detectar hasta 9000), pero su tasa de detección de imágenes por segundo es casi 5 veces más rápida. Dado que no necesitamos detectar una gran cantidad de clases (sólo 10 como ya veremos), se usará tiny-yolov3 para la detección ya que ofrece una mejora significativa del rendimiento.

5 ENTRENAMIENTO DE LA RED NEURONAL TINY-YOLOV3

Locura es hacer lo mismo una y otra vez esperando obtener resultados diferentes.

- Albert Einstein -

En este capítulo detallaremos todos los pasos seguidos para el entrenamiento de la red neuronal escogida, tiny-yolov3, para hacerla capaz de detectar y clasificar las señales de tráfico de velocidad que se presenten en carretera.

5.1 Conceptos clave

A continuación, se definen algunos de los conceptos claves para el entrenamiento de la red:

- Entrenamiento: Entrenar es aplicar un número definido de imágenes etiquetadas a nuestro modelo para obtener y corregir los pesos para el propósito deseado.
- Pesos o *weights*: Parámetros ajustables de la red neuronal de forma que, dada una determinada entrada, obtengamos la salida deseada. Estos parámetros se ajustan en el proceso de aprendizaje o entrenamiento de la red neuronal.
- *Dataset*: Conjunto de pares de imágenes y etiquetas.
- Etiquetas: Anotaciones preparadas que indicarán al modelo qué va a encontrar en cada imagen.
- Conjunto de entrenamiento (*Train data set)*: Conjunto de pares de imágenes y etiquetas que usaremos para entrenar la red neuronal.
- Conjunto de validación (*Test data set)*: Conjunto de pares de imágenes y etiquetas que usaremos para validar los pesos de la red neuronal y así obtener el error que se está produciendo con el ajuste de pesos en cada momento del entrenamiento.
- Clases: Distintos objetos que queremos detectar en nuestro modelo.

5.2 Introducción

El proceso de aprendizaje o entrenamiento es el proceso por el cual una red neuronal aprende como llevar a cabo una tarea, en nuestro caso, la tarea de detección y clasificación de objetos en una imagen. Como comentamos en el Capítulo 4, el entrenamiento puede ser supervisado o no supervisado.

En nuestro caso, la red neuronal llevará a cabo el proceso de aprendizaje supervisado, de forma que le proporcionaremos imágenes de entrada y sus correspondientes salidas (posición de la detección y clasificación). La red neuronal aprenderá de sus errores entre la clase predicha y la real, y, tras el estudio de características propias de la imagen, ajustará sus pesos para reducir ese error al mínimo. Esta diferencia entre la clase predicha y la real es calculada haciendo uso de la Función de Pérdida (*Function Error Loss*) que veremos más adelante.

5.3 Clases

Dado nuestro conjunto de imágenes de señales de velocidad, el primer paso es generar las etiquetas correspondientes. Darknet, el framework usado para entrenar tiny-yolov3, precisa un fichero *.txt* por cada imagen con una línea por cada uno de los objetos a detectar en la imagen. Cada línea tendrá el siguiente formato:

<*object-class*> <*x*> <*y*> <*width*> <*height*>

Donde *x*, *y*, *width* y *height* son las posiciones relativas a la imagen, mientras que *object-class* se refiere a la clase que representa el objeto señalado. En nuestro caso, hemos separado el número de objetos a detectar en 10 clases que se muestran en la Ilustración 37.

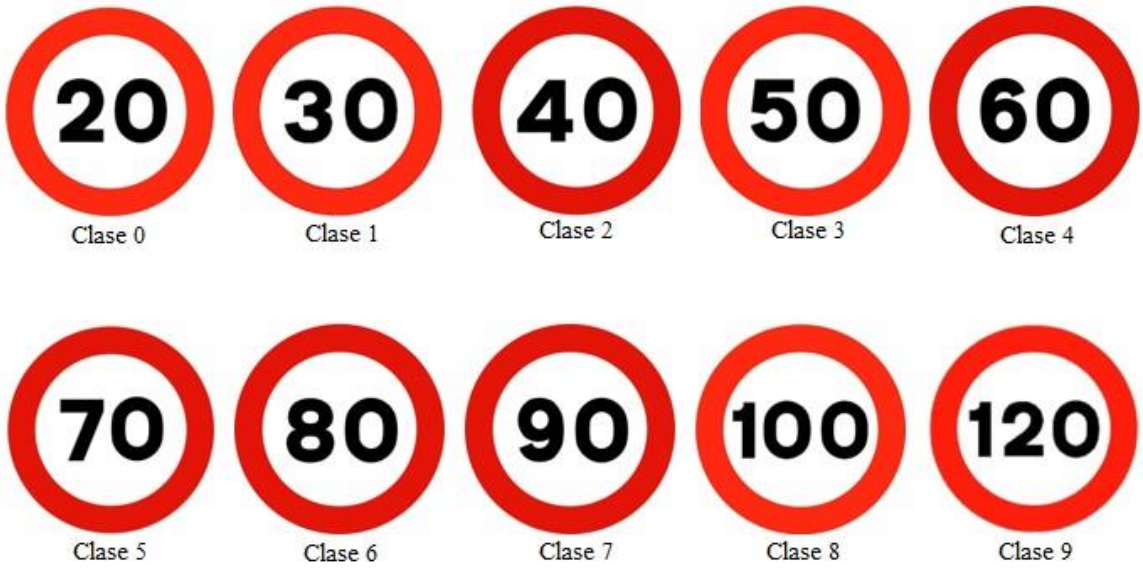

Ilustración 37: Clases

Se prodece a numerar las clases mostradas en la Ilustración 37 a continuación:

0. Velocidad máxima 20Km/h
1. Velocidad máxima 30Km/h
2. Velocidad máxima 40Km/h
3. Velocidad máxima 50Km/h
4. Velocidad máxima 60Km/h
5. Velocidad máxima 70Km/h
6. Velocidad máxima 80Km/h
7. Velocidad máxima 90Km/h
8. Velocidad máxima 100Km/h
9. Velocidad máxima 120Km/h

Las etiquetas de las 4.560 imágenes proporcionadas al modelo se han generado manualmente usando el software LabelImg, cuya interfaz gráfica observamos en la Ilustración 38.

Ilustración 38: Software LabelImg

5.4 Ficheros de configuración

Una vez tengamos el conjunto de imágenes y etiquetas, descargamos el framework Darknet desde su página oficial [29] y modificamos sus ficheros de configuración para ajustar la red a nuestro caso particular.

Partiendo del fichero situado en *cfg/tiny-yolov3.cfg*, hacemos una copia de este y la renombramos como *traffic-yolov3-tiny.cfg.* Dentro de este fichero, establecemos las variables según la Ilustración 39.

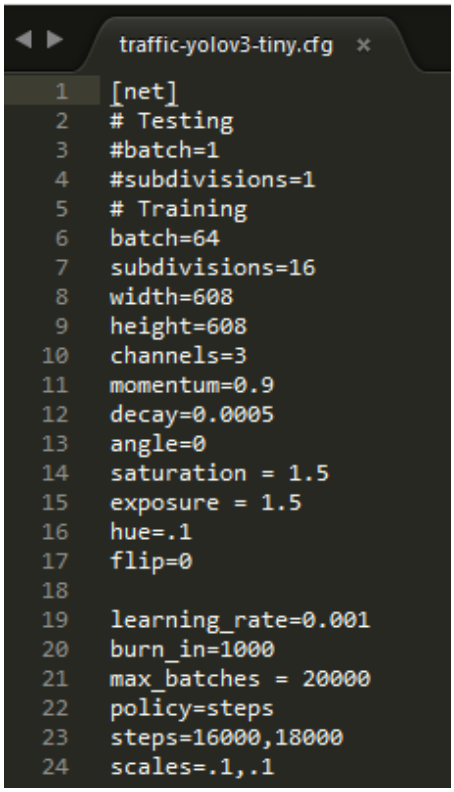

Ilustración 39: Parámetros de configuración tiny-yoloV3

El parámetro *batch,* que se refiere al lote de imágenes o número de imágenes que se propagarán por la red neuronal en una iteración durante el entrenamiento, ha sido modificada de 1 a 64. Al elegir un número más pequeño que el número total de imágenes hacemos que el espacio de memoria usado durante el entrenamiento sea menor y que así sea más rápido ya que los pesos se van actualizando con más frecuencia. Por otro lado, cuanto más pequeño es el *batch* escogido, menor es la precisión de la estimación del gradiente. Por defecto está establecido a 1, pero hemos elegido cambiarlo a 64 porque ese tamaño se comporta mejor con la arquitectura de tiny-yolov3. A continuación modificamos el valor de la variable *max_batches* por el valor de 2000 multiplicado por el número de clases, que en nuestro caso son 10. Esto es así porque se considera que 2000 iteraciones por clase son necesarias para entrenar la red adecuadamente.

Durante el entrenamiento, después que el 80% y el 90% de las iteraciones hayan sido completadas, la tasa de aprendizaje (*learning rate)* se ajustará, así que la variable steps se ha definido en '16000,18000'. La tasa de aprendizaje controla cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo. Elegir la tasa de aprendizaje es un desafío, ya que un valor demasiado pequeño puede resultar en un proceso de entrenamiento largo que podría atascarse, mientras que un valor demasiado grande puede resultar en aprender un conjunto de pesos demasiado rápido o un proceso de entrenamiento inestable. Por todo esto hemos dejado su valor por defecto.

Por otro lado, hemos aumentado la resolución de la red neuronal, en lugar de 416x416, a 608x608, así incrementamos la precisión y permitirá a la red neuronal ser capaz de detectar objetos más pequeños en las imágenes, y, además, se ha añadido el parámetro *flip* a 0. De esta forma, no se podrá dar la vuelta a las imágenes durante el entrenamiento. Esto se hace normalmente para obtener más variabilidad, pero en nuestro caso daba lugar a problemas, ya que las señales de 60Km/h y 90Km/h pueden ser confundidas.

Una vez establecidos los parámetros generales, nos centramos en los parámetros de cada capa, establecidos según la Ilustración 40.

```
124  [convolutional]
125  size=1
126  stride=1
127  pad=1
128  filters=45
129  activation=linear
130
131
132
133  [yolo]
134  mask = 3,4,5
135  anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
136  classes=10
137  num=6
138  jitter=.3
139  ignore_thresh = .7
140  truth_thresh = 1
141  random=1
142
```

Ilustración 40: Parámetros capas tiny-yoloV3

En cada una de las tres capas YOLO (estas son las capas de salida), hemos cambiado el número de *classes* a 10, que es el número de objetos distintos que queremos que nuestra red sea capaz de detectar, como ya hemos mencionado anteriormente. También, en cada capa convolucional anterior a una capa Yolo, hemos modificado el número de filtros de esa capa atendiendo a la ecuación: *filters=(classes+5) *3*, y, por último, el valor *random* ha sido establecido a 1 para incrementar la precisión de yolo para entrenarlo en distintas resoluciones, tamaños de imagen.

Finalmente, para terminar de configurar el marco Darknet completamente, hemos colocado nuestras imágenes con sus correspondientes etiquetas en la carpeta *data/images* y *data/labels* respectivamente. También se ha modificado el fichero Makefile para permitir el uso de GPU a la hora del entrenamiento, además de la creación de los ficheros: *data/traffic.names*, que contiene una línea por cada una de las clases a detectar, *data/traffic-train.txt* que contiene la ruta al conjunto de imágenes que se usarán como entrada para el entrenamiento de la red neuronal (conjunto de entrenamiento), *data/traffic-test.txt,* que contiene la ruta al conjunto de imágenes que usarán para validar el entrenamiento (conjunto de validación) y, por último, *cfg/traffic.data,* que especifica la localización de los archivos del conjunto de validación y entrenamiento y qué archivo contiene los nombres de las categorías (clases) que queremos detectar.

Nuestro marco de trabajo ya estaría preparado para comenzar el entrenamiento de la red. Partiremos del fichero *darknet53.conv.74,* que es el extractor de características pre-entrenado para YoloV3, esto es, los pesos iniciales de YOLO usados para entrenar datos personalizados.

5.5 Google Colab

Para entrenar tiny-yolov3, hemos usado la herramienta Google Colab, accessible a través de cualquier navegador, que nos permite ejecutar código y procesarlo en la nube, además de hacer uso de recursos más potentes. Este servicio gratuito nos da acceso a la Tesla K80 GPU con 12GB de RAM y más de 350GB de almacenamiento. Una vez comenzado el procedimiento, podemos usar estos recursos durante 12 horas seguidas, lo cuál ha sido suficiente para entrenar nuestra red neuronal.

Para realizar el entrenamiento y validación, preparamos el entorno de Google Colab tal y como se indica en el Anexo D de este documento. Primero, establecemos el uso de GPU para acelerar el proceso. El entrenamiento de una red neuronal con el uso de CPU es extremandamente lento, por eso hemos usado esta opción. A continuación, se ejecuta el código de la Ilustración 41 para preparar y comenzar el entrenamiento.

Trainning

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comentario Compartir

+ Código + Texto Conectar Editar

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

[ ] !unzip "/content/drive/My Drive/darknet.zip"

[ ] %cd /content/darknet
    !make
    !chmod +x ./darknet

[ ] !rm /content/darknet/backup -r
    !ln -s /content/drive/'My Drive'/backup /content/darknet

[ ] !sudo apt install dos2unix

    !dos2unix ./data/traffic-train.txt
    !dos2unix ./data/traffic-test.txt
    !dos2unix ./cfg/traffic.data
    !dos2unix ./data/traffic.names
    !dos2unix ./cfg/traffic-yolov3-tiny.cfg

[ ] %cd /content/darknet
    !./darknet detector train cfg/traffic.data cfg/traffic-yolov3-tiny.cfg darknet53.conv.74
```

Ilustración 41: Entorno Google Colab

En nuestro caso, el entrenamiento se paró al alcanzar unos determinados valores estadísticos presentados en la salida del entrenamiento en Google Colab.

5.6 Estadísiticos

Procedemos a definir los estadísticos más importantes con el fin de interpretar el comportamiento de la red durante su entrenamiento:

- Verdadero positivo (TP, True Positive): Acierto que se da cuando la red detecta un objeto en la posición y la clase correcta.
- Falso positivo (FP, False Positive): Error que se da cuando la red detecta un objeto que no existe en la imagen, o que, sí existe, pero de una clase distinta a la predicha.
- Falso negativo (FN, False Negative): Error que se da cuando la red no detecta un objeto que aparece en la imagen.
- Intersección sobre la Unión (IoU, *Intersection over Union*): Mide la precisión con la que nuestra red detectó un determinado objeto. En la Ilustración 42 podemos ver cómo calcular este porcentaje de superposición entre nuestro cuadro delimitador y el objeto real. Depende del umbral establecido, es decir, si se establece un umbral del 50% y el cuadro predicho por la red no cubre al menos el 50% del cuadro real, se considera como Falso Positivo.

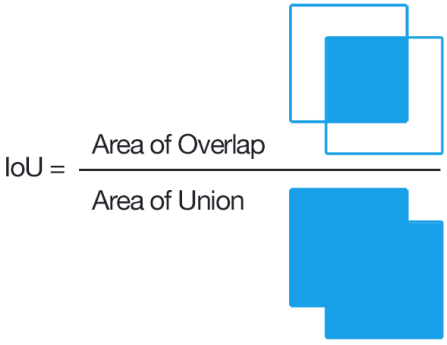

Ilustración 42: Intersección sobre la Unión

- Función de pérdida o *Loss Function*: Es el estadístico más importante para conocer el comportamiento de la red durante su entrenamiento. Como hemos dicho, YOLO predice hasta B cuadros delimitadores por cada celda, pero sólo queremos que uno de ellos sea responsable de predecir cierto objeto, y para ello seleccionamos el cuadro delimitador con el mayor IoU. A partir de aquí, YOLO usa el error de suma cuadrada entre las predicciones y el objeto de verdad para calcular la función de pérdida. Esta función tiene las siguientes partes:
    - Función de pérdida de clasificación (*Classification Loss*): Si se detecta un objeto, el error de clasificación de cada celda es el error al cuadrado de las probabilidades de clase condicionales por cada clase. Podemos ver mejor la función en la Ilustración 43:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

donde:

$\mathbb{1}_i^{\text{obj}}$ es 1 si hay un objeto en la celda i, y 0 si no lo hay.

$\hat{p}_i(c)$ denota la probabilidad condicional de la clase c para la celda i.

Ilustración 43: Función Pérdida de Clasificación

    - Función de pérdida de localización (*Localization Loss*): Mide el error entre la posición y el tamaño del cuadro delimitador y el objeto real:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

donde:

$\mathbb{1}_{ij}^{\text{obj}}$ es 1 si el cuadro delimitador nº j en la celda i es el responsable de predecir el objeto. Si no, es 0.

λ_{coord} aumenta el peso de la pérdida en las coordenadas del cuadro delimitador. Esto es para poner más énfasis en la precisión del cuadro delimitador.

Ilustración 44: Función de Pérdida de Localización

    - Función de Pérdida de Confianza (*Confidence Loss*): Simplemente mide la confianza (*confident score*). Si se detecta un objeto en el cuadro delimitador, la pérdida de confianza sería:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2$$

donde:

$\mathbb{1}_{ij}^{\text{obj}}$ es 1 si el cuadro delimitador nº j en la celda i es el responsable de predecir el objeto. Si no, es 0.

$\hat{C}_i$ es la confianza del cuadro delimitador j en la celda i

Ilustración 45: Función de Pérdida de Confianza

Si no se detecta ningún objeto sería:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2$$

donde:

$\mathbb{1}_{ij}^{\text{noobj}}$ será lo contrario de $\mathbb{1}_{ij}^{\text{obj}}$

$\hat{C}_i$ es la confianza del cuadro delimitador j en la celda i

λ_{noobj} disminuye el peso de la pérdida cuando no se detecta nada. Esto es debido a que cuando entrenamos normalmente hay más parte de la imagen que no contiene nada.

Ilustración 46: Función de Pérdida de Confianza (2)

De este modo, la función de pérdida total sería la mostrada en la Ilustración 47:

$$\begin{aligned}
&\lambda_{\textbf{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]\\
&+\lambda_{\textbf{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]\\
&+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2\\
&+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2\\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{\text{obj}}\sum_{c\in\text{classes}}(p_i(c)-\hat{p}_i(c))^2
\end{aligned}$$

Ilustración 47: Función de Pérdida

- Promedio de precisión media o *Mean Average Precision* (mAP): Mide la precisión media para cada clase y realiza un promedio. Recordemos que el entrenamiento de una red neuronal consiste en modificar los pesos de las neuronas de la red para que la función de pérdida sea mínima. Sin embargo,

debemos de tener en cuenta el sobreajuste de pesos u *overfitting*, que se produce cuando los pesos se ajustan demasiado al conjunto de entrenamiento, lo que implica tener un mAP muy elevado, pero, al estar sobreajustado y no generalizar correctamente, se tendría un mAP bajo para el conjunto de validación.

En la Ilustración 48 se puede observar una parte de la salida del entrenamiento de YOLO. A continuación, detallaremos qué significa.

```
19575: 0.064533, 0.101819 avg, 0.000010 rate, 1.599550 seconds, 1252800 images
Loaded: 0.000061 seconds
Region 16 Avg IOU: 0.927697, Class: 0.957139, Obj: 0.994984, No Obj: 0.001889, .5R: 1.000000, .75R: 1.000000,  count: 3
```

Ilustración 48: Salida Entrenamiento YOLO

Esta muestra presenta los siguientes datos de interés:

- *19575* indica la iteración o lote de entrenamiento actual.
- *0.064533* es la pérdida total.
- *0.101819 avg*, es la Función de Pérdida (Loss Function), que debe ser lo más bajo posible. Como regla general, una vez que llegue a menos de 0.060730 avg, podemos dejar de entrenar, que es lo que hemos hecho en este caso.
- *0.001000* representa la tasa de aprendizaje definida en el fichero de configuración anteriormente visto.
- *1.599550 seconds*: representa el tiempo en procesar este lote.
- *Region Avg IOU: 0.927697* es la media de IoU de cada imagen en la región actual. El porcentaje es elevado, 90%, lo que nos indica que el entrenamiento está casi completado.
- *count:* 3 es la cantidad de Verdaderos Positivos en la subdivisión de la imagen actual.

6 SERVICIO WEB SPRING Y BASE DE DATOS

La imaginación es más importante que el conocimiento. Mientras que el conocimiento es limitado, la imaginación abarca el mundo entero, estimulando el progreso, dando a luz a la evolución

-Albert Einstein-

Una vez tenemos tiny-yoloV3 entrenada para la detección y correcta clasificación de señales de tráfico de velocidad que se presentan en carretera, procedemos a explicar el tratamiento de estos datos obtenidos, así como los datos de los usuarios de nuestro sistema. Para ello, se ha desarrollado un servicio web RESTful con Spring Boot y una base de datos MySQL para la persistencia de los datos.

En primer lugar, veremos la estructura de la base de datos MySQL creada para después proceder a explicar la estructura y el funcionamiento del servicio web Spring.

6.1 Servidor de base de datos

La base de datos diseñada para este proyecto se compone de cinco tablas que se pueden ver en el modelo Entidad-Relación de la Ilustración 49, así como sus correspondientes campos y las relaciones entre ellas.

A continuación, se procede a explicar con más detalle cada una de estas tablas:

- Tabla "usuarios": Almacena toda la información de los usuarios registrados a través de la aplicación Android. Está formada por cuatro columnas que son las siguientes:
    - username: Nombre de usuario de la persona que se registra en la aplicación. Es la clave primaria de la tabla y es de tipo VARCHAR. Es uno de los datos pedidos para iniciar sesión en la aplicación Android.
    - nombre: Nombre completo del usuario registrado. Es de tipo VARCHAR.
    - email: Correo electrónico del usuario registrado. Es de tipo VARCHAR.
    - contraseña: Contraseña introducida por el usuario al registrarse en la aplicación. Es de tipo VARCHAR y se guarda codificada en la base de datos. Se utiliza junto con el username para iniciar sesión en la aplicación.

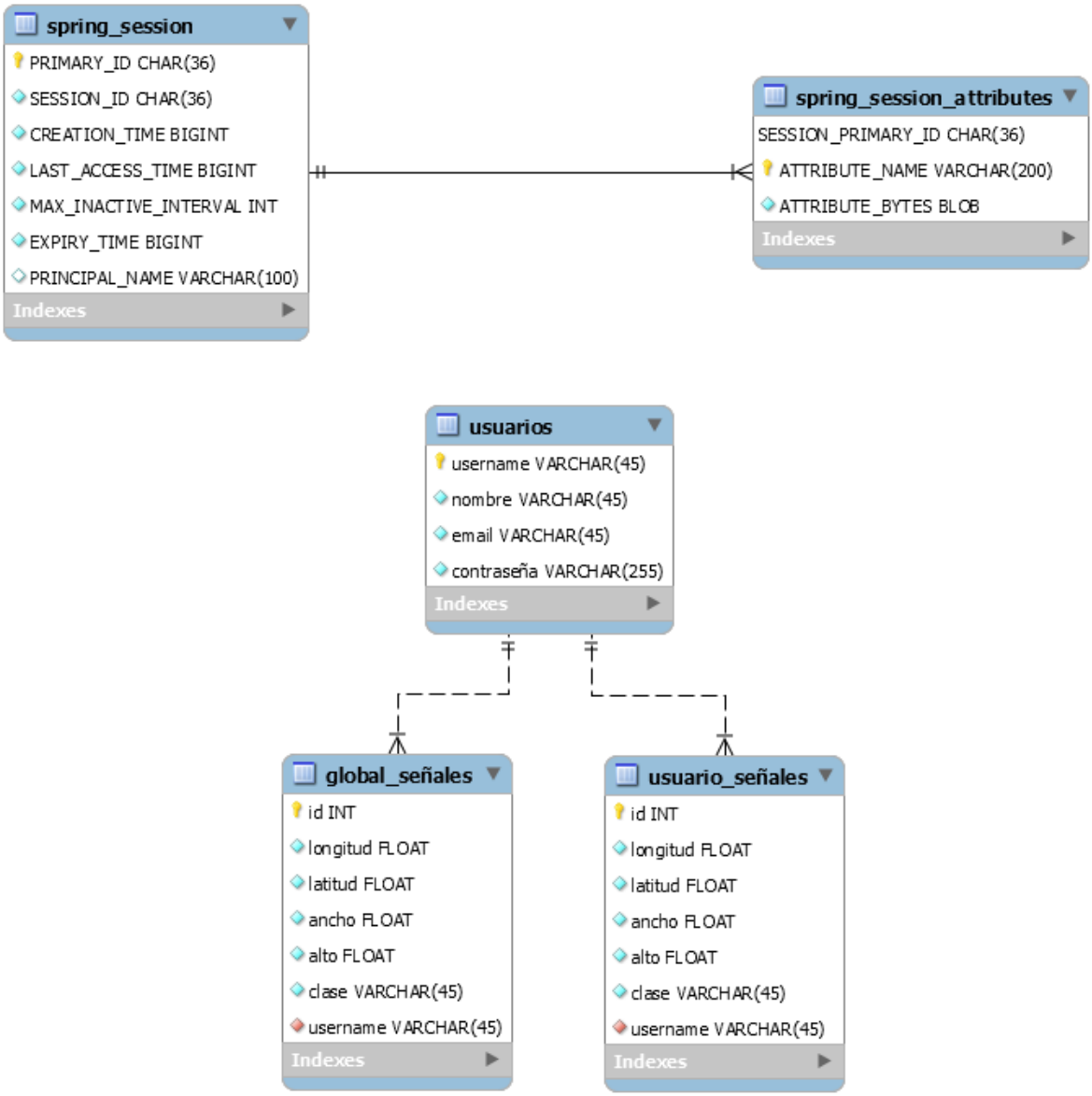

Ilustración 49: Modelo Entidad-Relación de la Base de Datos

- Tabla “usuario_señales”: Almacena toda la información de las mejores señales de tráfico detectadas por cada usuario en la aplicación Android. Está formada por siete columnas:
    - id: Es el identificador de cada inserción en la tabla, se incrementa automáticamente conforme se van insertando más filas. Se trata de la clave primaria y es de tipo INTEGER.
    - longitud: Almacena la coordenada longitud de la posición GPS en la que se detectó la señal de tráfico. Es de tipo FLOAT.
    - latitud: Almacena la coordenada latitud de la posición GPS en la que se detectó la señal de tráfico. Es de tipo FLOAT.
    - ancho: Almacena el parámetro ancho de la dimensión de la señal de tráfico detectada en píxeles. Es de tipo FLOAT.
    - alto: Almacena el parámetro alto de la dimension de la señal de tráfico detectada en píxeles. Es de tipo FLOAT.
    - clase: Almacena la clasificación de la señal detectada. Es de tipo VARCHAR.

  - username: Almacena el nombre de usuario del usuario que realizó la detección. Se trata de una clave externa que referencia a la columna "username" de la tabla "usuarios" y es de tipo VARCHAR.

- Tabla "global_señales": Es igual a la tabla "usuario_señales" pero almacena las mejores señales detectadas entre todos los usuarios. Tiene exactamente las mismas columnas y las mismas relaciones con otras tablas que "usuario_señales".
- Tablas "spring_session" y "spring_session_attributes": Almacenan los datos de las distintas sesiones actuales abiertas en la aplicación Android. La creación de estas tablas era indispensable para el uso del módulo Spring Session, que explicaremos a continuación.

6.2 Clases

Se procede ahora a detallar las distintas clases creadas para el funcionamiento del servicio web Spring.

En primer lugar, para la comunicación con la base de datos, se ha creado una clase con la anotación *@Entity* por cada una de las tablas creadas en MySQL. Esta anotación indica que una determinada clase es una entidad JPA. También podemos usar la notación *@Table* para indicarle a JPA la tabla de la base de datos con la que se corresponde en el caso de que la clase y la tabla no tengan el mismo nombre. Podemos hacer lo mismo con cada uno de los atributos de la clase y las columnas de una tabla con la notación *@Column*, pero en nuestro caso no será necesario ya que los atributos definidos coinciden con las columnas de la tabla a la que representan, además de ser del mismo tipo.

- Clase **Usuario**. Esta clase implementa la tabla "usuarios" en la base de datos y contiene información para poder identificar al usuario dentro del sistema. Se muestra en la Ilustración 50 y posee los siguientes atributos:
  - Private String username. Hace referencia al nombre de usuario escogido e introducido en la aplicación por parte del usuario.
  - Private String nombre: Hace referencia al nombre completo introducido por el usuario.
  - Private String email: Hace referencia al email introducido por el usuario.
  - Private String contraseña: Hace referencia a la contraseña elegida e introducida por el usuario.

```
1  package com.icodriver;
2
3  import java.io.Serializable;
9
10
11 @Entity
12 @Table(name="usuarios")
13 public class Usuario implements Serializable{
14
15     private static final long serialVersionUID = 2L;
16
17     @Id
18     private String username;
19     @NotEmpty
20     private String nombre;
21     @NotEmpty
22     private String email;
23     @NotEmpty
24     private String contraseña;
```

Ilustración 50: Clase Usuario – Servicio Web Spring

- Clase **TrafficSign.** Implementa la tabla "usuario_señales" en la base de datos y contiene información sobre cada una de las señales detectadas por cada usuario individualmente mediante el uso de la aplicación Android. Se muestra en la Ilustración 51 y posee los siguientes atributos:
    - Private int id: Identificador de la señal de tráfico registrada. Es único para cada una de las señales.
    - Private float longitud: Hace referencia a la coordenada longitud de la posición en la que se detectó la señal de tráfico.
    - Private float latitud: Hace referencia a la coordenada latitud de la posición en la que se detectó la señal de tráfico.
    - Private float ancho: Hace referencia al parámetro ancho de la dimensión de la señal detectada.
    - Private float alto: Hace referencia al parámetro alto de las dimensiones de la señal detectada.
    - Private String clase: Hace referencia a la clasificación de la señal detectada según el apartado 5.3 del capítulo anterior.
    - Private String username: Hace referencia al nombre del usuario que detectó la señal de tráfico registrada.

```
1  package com.icodriver;
2
3⊕ import java.io.Serializable;
13
14 @Entity
15 @Table(name="usuario_señales")
16 public class TrafficSign implements Serializable{
17
18     private static final long serialVersionUID = 1L;
19
20⊖    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private int id;
22⊖    @NotNull
23     private float longitud;
24⊖    @NotNull
25     private float latitud;
26⊖    @NotNull
27     private float ancho;
28⊖    @NotNull
29     private float alto;
30⊖    @NotEmpty
31     private String clase;
32⊖    @NotEmpty
33     private String username;
```

Ilustración 51: Clase TrafficSign – Servicio Web Spring

- Clase **GlobalTrafficSign.** Implementa la tabla "global_señales". Posee los mismos atributos que la clase definida en el punto anterior. Se ha creado otra entidad debido a que las señales de tráfico que se guardarán en la tabla "global_señales" serán seleccionadas de un modo distinto a las que se registran en la tabla "usuario_señales". Se muestra en la Ilustración 52.

```java
package com.icodriver;

import java.io.Serializable;

@Entity
@Table(name="global_señales")
public class GlobalTrafficSign implements Serializable{

    private static final long serialVersionUID = 1L;

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @NotNull
    private float longitud;
    @NotNull
    private float latitud;
    @NotNull
    private float ancho;
    @NotNull
    private float alto;
    @NotEmpty
    private String clase;
    @NotEmpty
    private String username;
```

Ilustración 52: Clase GlobalTrafficSign – Servicio Web Spring

Por cada una de estas entidades se ha creado una interfaz Repositorio encargada de recuperar la información de la base de datos o de introducir información en ella. Con estas interfaces, la tarea repetitiva de crear las implementaciones concretas para el acceso a la base de datos se simplifican, pues sólo vamos a necesitar definir la interfaz. Estas interfaces heredan de *JPARepository*, que ya por sí misma proporciona una serie de métodos genéricos como *save()* o *findAll(),* para guardar un objeto en la base de datos u obtener todos los objetos guardados en una tabla respectivamente, entre otros.

- **Interfaz UsuarioRepository**: Repositorio encargado del acceso de la base de datos para objetos de tipo "Usuario", es decir, se encargará de introducir u obtener información sobre los usuarios en la tabla "usuarios". Como vemos en la Ilustración 53, además de los métodos genéricos, existen métodos derivados que tienen dos partes principales separadas por palabras clave:
    - Usuario findByUsername: Devuelve un objeto tipo "Usuario" de la tabla "usuarios" cuyo "username" coincida con el pasado por parámetro. La palabra clave es "By", esto se podría hacer con cualquier columna de la tabla, poniendo el nombre de esta detrás de la palabra clave.
    - boolean existsUsuarioByUsername: Devuelve un tipo boolean si existe un usuario con un "username" pasado como parámetro. En este caso las palabras clave son "By" y "exists".

```java
package com.icodriver;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UsuarioRepository extends JpaRepository<Usuario, String>{

    Usuario findByUsername(String username);

    boolean existsUsuarioByUsername(String username);
}
```

Ilustración 53: Interfaz UsuarioRepository – Servicio Web Spring

- **Interfaz SignRepository:** Encargado del acceso a la base de datos para objetos de tipo "TrafficSign" de la tabla "usuario_señales".

- public List<Integer> mismoRadio: Devuelve una lista de enteros que se corresponden con los valores de las columnas "id" de los objetos de tipo "TrafficSign" almacenados en la tabla "usuario_señales" cuya posición se encuentra dentro de un radio de 1 metro de la posición pasada como parámetro y cuya columna "username" coincida con la pasada como parámetro también. Al no ser un método genérico para acceder a la base de datos, podemos especificar la sentencia con la etiqueta *@Query* encima del método como vemos en la Ilustración 54.
- public List<Interger> masPequeña: Devuelve lo mismo que el método "mismoRadio" pero además filtra esas señales de tráfico y devuelve solo aquellas cuyas dimensiones (ancho x alto) son menores a las pasadas por parámetro.
- public List<TrafficSign> findByUsername: Devuelve una lista de los objetos de tipo "TrafficSign" almacenados cuyo campo "username" coincide con el pasado por parámetro.

```
1  package com.icodriver;
2
3  import java.util.List;
7
8  public interface SignRepository extends JpaRepository<TrafficSign, Integer>{
9
10     @Query(nativeQuery = true,
11             value="SELECT id, "
12                     + "( 6371 * acos(cos(radians(?1)) * cos(radians(latitud)) * cos(radians(longitud) "
13                     + "- radians(?2)) + sin(radians(?1)) * sin(radians(latitud))))"
14                     + " AS distance FROM usuario_señales"
15                     + " WHERE ancho*alto < ?3 * ?4 AND username=?5 HAVING distance < 0.1 ")
16     public List<Integer> masPequeña (float latitud, float longitud, float ancho, float alto, String username);
17
18     @Query(nativeQuery = true,
19             value="SELECT id, "
20                     + "( 6371 * acos(cos(radians(?1)) * cos(radians(latitud)) * cos(radians(longitud) "
21                     + "- radians(?2)) + sin(radians(?1)) * sin(radians(latitud)))) "
22                     + "AS distance FROM  usuario_señales "
23                     + "WHERE username=?3 HAVING distance < 0.1 ORDER BY distance")
24     public List<Integer> mismoRadio(float latitud, float longitud, String username);
25
26     public List<TrafficSign> findByUsername(String username);
27 }
```

Ilustración 54: Interfaz SignRepository – Servicio Web Spring

- **Interfaz GlobalSignRepository:** Encargado del acceso a la base de datos para objetos de tipo "GlobalTrafficSign" de la tabla "global_señales". Los métodos definidos son los mismos que los de la interfaz SignRepository con la salvedad de que no filtra por "username", es decir, busca en la tabla entre todas las almacenadas por cualquier usuario, en lugar de por uno en concreto.

```
1  package com.icodriver;
2
3  import java.util.List;
7
8  public interface GlobalSignRepository extends JpaRepository<GlobalTrafficSign, Integer>{
9
10     @Query(nativeQuery = true,
11             value="SELECT id, "
12                     + "( 6371 * acos(cos(radians(?1)) * cos(radians(latitud)) * cos(radians(longitud) "
13             + "- radians(?2)) + sin(radians(?1)) * sin(radians(latitud)))) "
14             + "AS distance FROM global_señales WHERE ancho*alto < ?3 * ?4 HAVING distance < 0.1 ")
15     public List<Integer> masPequeña (float latitud, float longitud, float ancho, float alto);
16
17     @Query(nativeQuery = true,
18             value="SELECT id, "
19                     + "( 6371 * acos(cos(radians(?1)) * cos(radians(latitud)) * cos(radians(longitud) "
20             + "- radians(?2)) + sin(radians(?1)) * sin(radians(latitud)))) "
21             + "AS distance FROM  global_señales HAVING distance < 0.1 ORDER BY distance")
22     public List<Integer> mismoRadio(float latitud, float longitud);
23 }
```

Ilustración 55: Interfaz GlobalSignRepository – Servicio Web Spring

Por último, mostraremos las clases implementadas para controlar el funcionamiento de nuestro servicio:

- **Application:** Clase principal de la aplicación que declara el método "main". Esta clase se ejecuta en primer lugar al poner en marcha la aplicación. Contiene las anotaciones *@SpringBootApplication* que es una combinación de las anotaciones *@Configuration*, *@ComponentScan* y *@EnableAutoConfiguration*, de forma que proporciona la configuración de la aplicación con una sóla línea de código. Se muestra en la Ilustración 56.

```
1  package com.icodriver;
2
3  import org.springframework.boot.SpringApplication;
7
8  @EnableJpaAuditing
9  @EnableJdbcHttpSession
10 @SpringBootApplication
11 public class WebServiceApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(WebServiceApplication.class, args);
15     }
16
17 }
18
```

Ilustración 56: Clase Application – Servicio Web Spring

- **Clase UsuarioController:** Esta clase es un controlador que se encarga del manejo de las solicitudes de usuarios a través de la ruta */usuarios* y se muestra en la Ilustración 57. Las clases marcadas con la anotación *@Controller* son aquellas que controlan el servicio web manejando las solicitudes recibidas. En nuestro caso la anotación usada es *@RestController* ya que el servicio web es RESTful. Esta anotación es una combinación de *@Controller* y *@ResponseBody*, lo que elimina la necesidad de anotar cada método de manejo de solicitudes de la clase controlador con la anotación *@ResponseBody*.
    - public Usuario login: Busca en la tabla "usuarios" el usuario cuyo "username" se pasa como parámetro.
    - public String registrar: Registra la información de un nuevo usuario en la tabla "usuarios". Si el campo "username" del usuario a insertar en la tabla ya existe, no se podrá hacer el registro. El "username" de un usuario debe ser único.
    - public List<Usuario> getUsuarios: Devuelve una lista de todos los usuarios existentes en la tabla "usuarios".
    - public String cerrarsesion: Elimina la sesión abierta del usuario y elimina las cookies.

```java
@RestController
@RequestMapping("/usuarios")
public class UsuarioController {

    @Autowired
    UsuarioService usuarioService;

    @Autowired
    private BCryptPasswordEncoder encoder;

    @GetMapping("/login")
    public Usuario login(String username){
        Usuario usuario = usuarioService.findByUsername(username);
        return usuario;
    }

    @PostMapping("/registrar")
    public @ResponseBody String registrar(@RequestBody Usuario usuario) {
        if(usuarioService.existUsername(usuario.getUsername())) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Username already exists");
        }else {
            usuario.setContraseña(encoder.encode(usuario.getContraseña()));
            usuarioService.save(usuario);
            return "";
        }
    }

    @GetMapping("/listar")
    public List<Usuario> getUsuarios() {
        return usuarioService.findAll();
    }

    @PostMapping("/cerrarsesion")
    public String cerrarsesion(HttpServletRequest request) {
```

Ilustración 57: Clase UsuarioController – Servicio Web Spring

- **Clase SecurityConfig:** Clase de configuración para implementar la autenticación y control de acceso de usuarios en nuestro sistema gracias al módulo Spring Security (todos los módulos usados en el desarrollo de este servicio web y cómo añadirlos, está detallado en el Anexo B de este documento).

  Configuramos nuestro sistema como muestra la Ilustración 58, de forma que no se pueda acceder a nuestros servicios si el usuario no está autenticado, excepto en el caso de registro de usuarios, al que puede acceder cualquiera. Se apoya en la clase "UsuarioService" de donde obtiene los datos de autenticación que deben ser introducidos por el usuario con el fin de poder usar los servicios. Esto lo podemos ver en la Ilustración 59. Se sobreescribe el método que obtiene la información de autenticación. En este caso si un usuario introduce un "username" y "contraseña" existente en la base de datos "usuarios", podrá autenticarse y acceder al servicio.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private UsuarioService service;

    @Autowired
    private BCryptPasswordEncoder bCrypt;

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
        return bCryptPasswordEncoder;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(service).passwordEncoder(bCrypt);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // Especifica que cualquier petición que entra debe estar autenticada.
        http.cors().and().csrf().disable();
        http.authorizeRequests()
        .anyRequest()
        .authenticated()
        .and()
        .httpBasic();
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        // Petición que no tiene que estar autenticada.
        web.ignoring().antMatchers("/usuarios/registrar");
    }
```

Ilustración 58: Clase SecurityConfig – Servicio Web Spring

```java
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    Usuario usuario = usuarioRepository.findByUsername(username);

    List<GrantedAuthority> roles = new ArrayList<>();
    roles.add(new SimpleGrantedAuthority("USER"));

    UserDetails userDet = new User(usuario.getUsername(), usuario.getContraseña(), roles);
    return userDet;
}
```

Ilustración 59: Autenticación en Clase UsuarioService

- **Clase SignController:** Controlador que se encarga de manejar las peticiones correspondientes a la ruta */signs*. Se han creado dos controladores separados para diferenciar el manejo de los usuarios con el manejo de las señales de tráfico.
    - public String saveDetection: Método que se encarga de guardar tanto en la tabla "global_señales" como en la tabla "usuario_señales" las distintas señales de tráfico detectadas por el usuario en la aplicación Android. En "usuario_señales" se registra una señal si no hay una ya registrada en un radio de menos de un metro o si las señales que ya están registradas tienen las dimensiones más pequeñas, en cuyo caso se borran las que ya están registradas y se guarda la nueva. Todo esto se hace entre las señales detectadas por un mismo usuario. En la tabla "global_señales" se procede de la misma forma para el registro de una señal, pero se tienen en cuenta las señales de todos los usuarios.
    - public List<TrafficSign> getSeñales: Método que devuelve una lista con todas las señales registradas en la tabla "usuario_señales" cuya columna "username" coincida con el nombre

de usuario pasado como parámetro.

- public List<GlobalTrafficSign> getGlobalSeñales: Método que devuelve una lista con todas las señales registradas en la tabla "global_señales".

Se puede observa la clase "SignController" en la Ilustración 60:

```
@RestController
@RequestMapping("/signs")
public class SignController {

    @Autowired
    SignRepository signRepository;

    @Autowired
    GlobalSignRepository globalRepository;

    @PostMapping("/deteccion")
    public @ResponseBody String saveDeteccion(@RequestBody GlobalTrafficSign ts) {
```

Ilustración 60: Clase SignController – Servicio Web Spring

6.3 API REST

En este apartado mostraremos la API REST, esto es, los distintos URI y métodos implementados para poder acceder a las distintas funciones que proporciona nuestro servicio web.

En el marco del proyecto todas las peticiones realizadas al servicio web mediante su API son mediante peticiones HTTP con cuerpo con formato JSON. Dividiermos las peticiones en dos grupos, uno para las peticiones relativas a usuarios y otro para las peticiones relativas a las señales de tráfico.

6.3.1 Consultas de Usuarios

Método	URI	Parámetros	Descripción
GET	/usuarios/listar	No aplica	Devuelve una lista con todos los usuarios almacenados en la tabla "usuarios".
GET	/usuarios/login?[parámetro]=[valor]	"username"=Nombre de usuario.	Autenticación de usuario. Las creedenciales se envían codificadas en la cabecera "Authorization" de la petición.
POST	/usuarios/registrar	{"username":"test", "nombre":"test", "email":" test","contraseña":"test"}	Registro de un nuevo usuario. Se insertan los datos del nuevo usuario en la tabla "usuarios".
POST	/usuario/cerrarsesion	No aplica	Elimina la sesión abierta del usuario y las cookies de la tabla. Elimina la sesión de la tabla "spring_session" y "spring_session_attributes".

Tabla 1: API REST – Consulta usuarios

6.3.2 Consultas de Señales

Método	URI	Parámetros	Descripción
GET	/signs/listar/{username}	"username" es el nombre de usuario del usuario actual.	Devuelve una lista con todas las señales de la tabla "usuario_señales" cuya columna "username" coincide con el parámetro pasado en la petición.
GET	/signs/listarglobal	No aplica.	Devuelve una lista con todas las señales de tráfico almacenadas en la tabla "global_señales".

POST	/signs/deteccion	{
		"id":"test"
		"longitud":"test"
		"latitud":"test"
		"ancho":"test"
		"alto":"test"
		"clase":"test"
		"username":"test"
		}
			Almacena la señal de tráfico detectada en "usuario_señales" y "global_señales" si no hay ninguna otra de mayores dimensiones en la misma posición.

Tabla 2: API REST – Consultas señales

6.4 Despliegue en AWS

Con el fin de proporcionar acceso a nuestro servicio web desde cualquier punto y momento, hemos optado por hacer uso de los servicios en la nube. Para ello hemos escogido la plataforma AWS (Amazon Web Services), que, gracias a su plan gratuito de 12 meses, se ha podido desplegar nuestro servidor en la nube sin coste alguno, lo cual lo convierte en la mejor opción para este Trabajo de Fin de Grado.

Los pasos para el despliegue de este servicio Spring y la base de datos en la nube están detallados en el Anexo D de este documento.

A continuación, se detallan los dos servicios proporcionados por AWS usados en este proyecto: AWS Elastic Beanstalk y Amazon RDS.

6.4.1 AWS Elastic Beanstalk

AWS Elastic Beanstalk es un servicio fácil de utilizar para implementar y escalar servicios y aplicaciones web desarrollados con Java, .NET, PHP, Node.js, Python, Ruby, Go y Docker en servidores familiares como Apache, Nginx, Passenger e IIS.

Para usar este servicio, sólo hemos necesitado cargar nuestro código y Elastic Beanstalk se encarga de administrar de manera automática la implementación, desde el aprovisionamiento de la capacidad, el equilibrio de carga y el escalado automático hasta la monitorización del estado de la aplicación. Al mismo tiempo, tendrá el control absoluto de los recursos de AWS que alimentan su aplicación y podrá acceder a los recursos subyacentes cuando quiera.

En la Ilustración 61 podemos ver una imagen de la consola o interfaz gráfica de AWS Elastic Beanstalk:

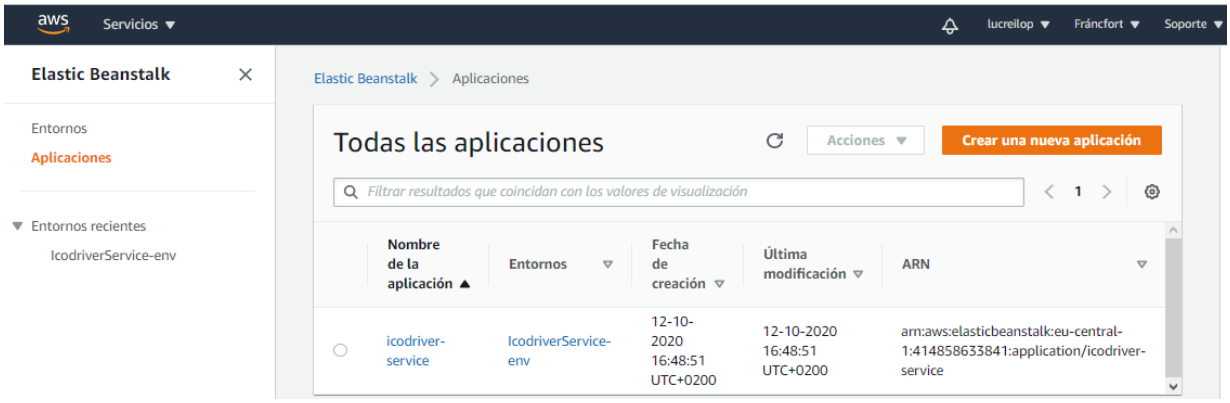

Ilustración 61: Interfaz AWS Elastic Beanstalk

6.4.2 Amazon RDS

Amazon Relational Database Service (Amazon RDS) es un servicio sencillo de configurar, utilizar y escalar una base de datos relacional en la nube.

Amazon RDS está disponible para varios tipos de instancias de base de datos (optimizadas para memoria, rendimiento u operaciones de E/S) y le proporciona seis motores de bases de datos conocidos entre los que elegir, incluidos Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database y SQL Server.

En la Ilustración 62 podemos ver una imagen de la consola o interfaz gráfica de Amazon RDS:

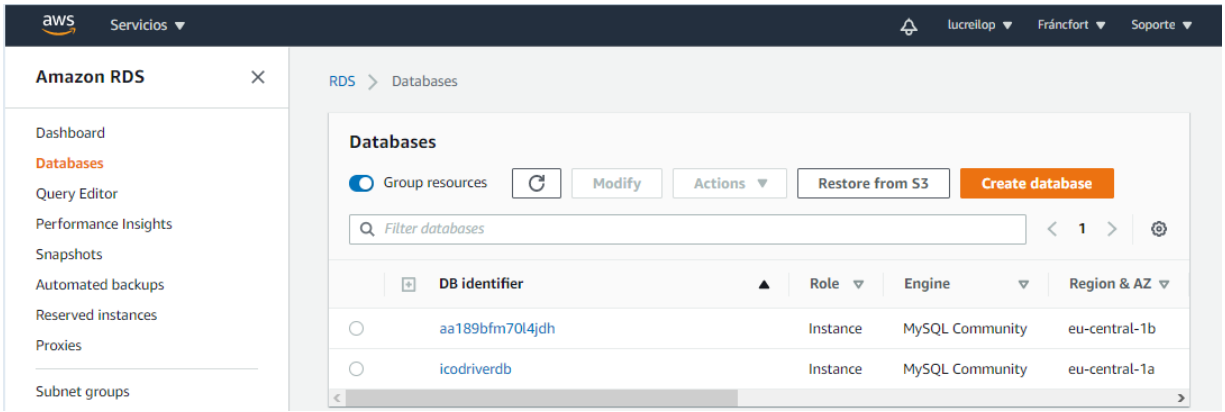

Ilustración 62: Interfaz Amazon RDS

7 APLICACIÓN ANDROID ICODRIVER

La ciencia se compone de errores, que, a su vez, son los pasos hacia la verdad.

- Julio Verne -

En este capítulo se detallará el diseño y la implementación de la Aplicación Android iCodriver. A través de ella, el usuario podrá acceder a todas las funcionalidades que ofrece nuestro sistema. Para dar una idea de la solución final que se ha adoptado, primero se pasará a mencionar las distintas partes que componen nuestro sistema completo y la interacción entre las mismas.

En total consta de 3 partes, las cuales son el servicio web RESTful desarrollado con Spring Boot y Eclipse, la base de datos MySQL administrada mediante MySQL Workbench, ambos mencionados anteriormente, y la aplicación móvil construida en Android. La arquitectura de este sistema sigue un patrón Modelo-Vista-Controlador (MVC). Este patrón es utilizado para separar los datos de una aplicación, la interfaz de usuario y la lógica de control, así separándolo en modelo, vista y controlador. En la Ilustración 63, se presenta mejor este tipo de arquitectura.

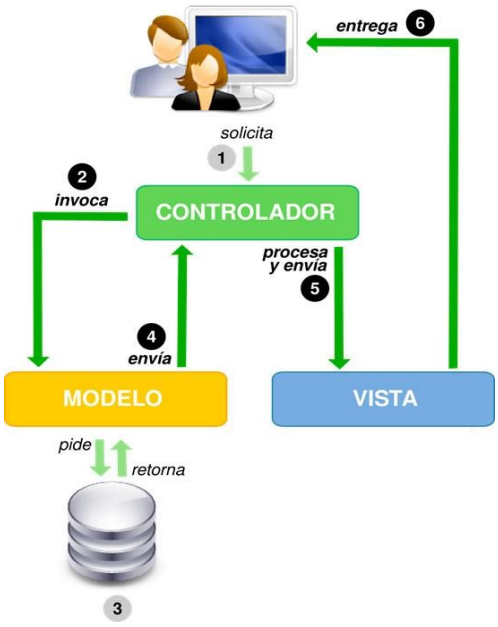

Ilustración 63: Patrón MVC

En el Modelo nos encontramos una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. La Vista, o también conocida como interfaz de usuario, contiene la información que es enviada al cliente y los mecanismos para interactuar con él, en nuestro caso, la vista se presenta en la aplicación Android desarrollada con Android Studio, que se tratará en este capítulo. Por último, el Controlador, actúa como un intermediario entre el Modelo y la Vista, gestionando así el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno. Spring Web MVC [31] proporciona una arquitectura Modelo-Vista-Controlador, por lo que nos facilita los distintos componentes y un acoplamiento sencillo y flexible entre ellos.

A continuación, presentamos una serie de diagramas de secuencia para aclarar el funcionamiento y las interacciones que se darán entre las distintas partes de nuestro sistema.

El primer diagrama de secuencia mostrado en la Ilustración 64, muestra el registro de nuevos usuarios al sistema.

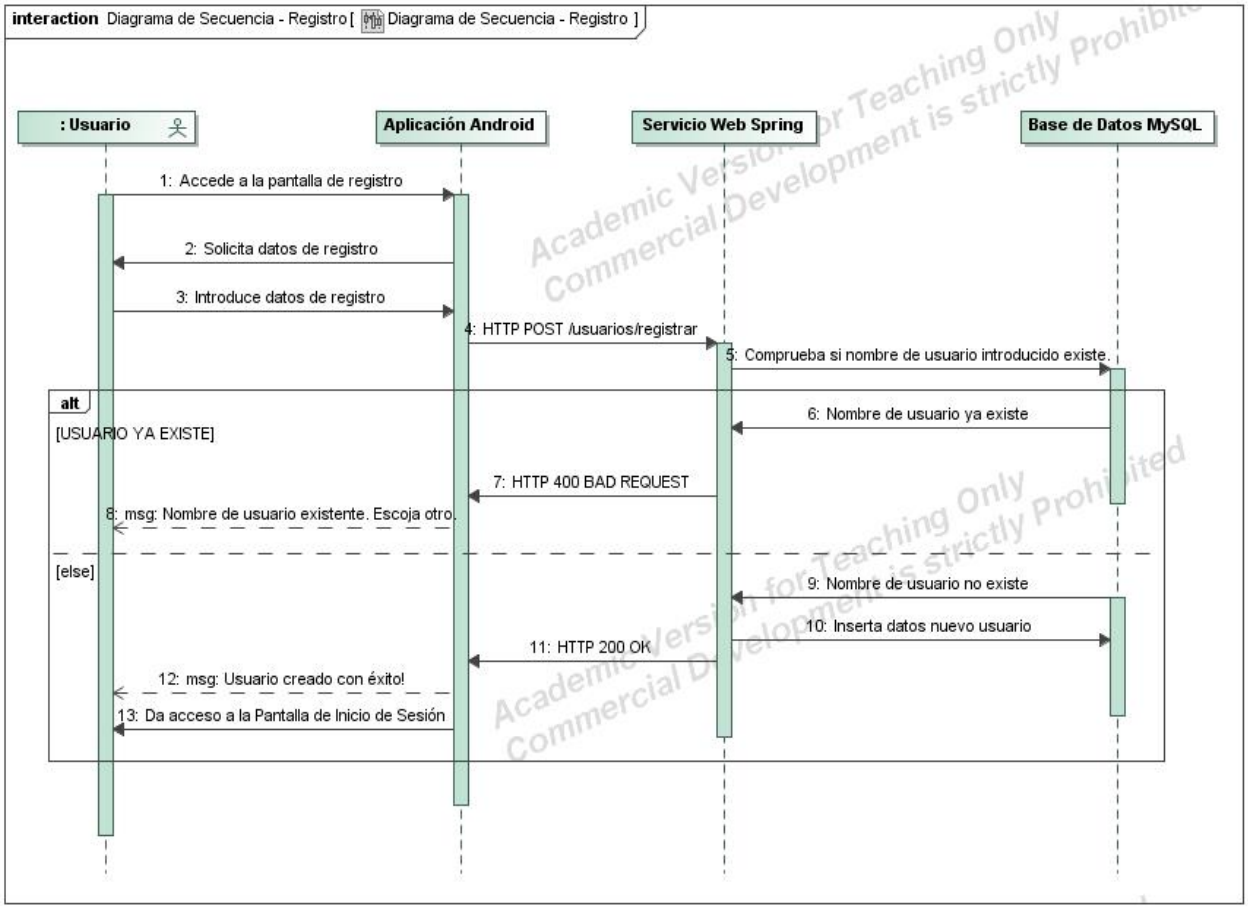

Ilustración 64: Diagrama de Secuencia – Registro

A continuación, en la Ilustración 65, se puede observar el diagrama de secuencia de inicio de sesión de un usuario ya registrado en el sistema.

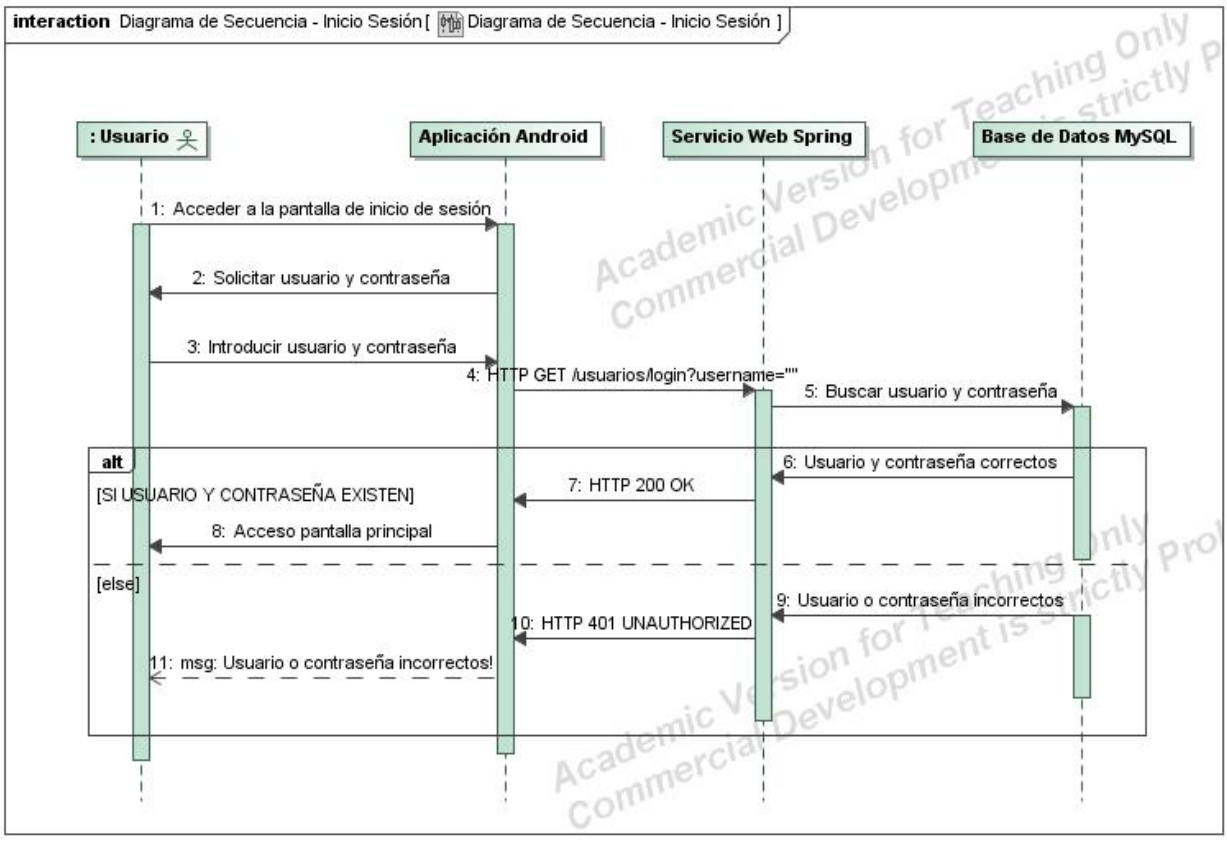

Ilustración 65: Diagrama de Secuencia – Inicio de Sesión

En la Ilustración 66, se presenta el diagrama de secuencia de cierre de sesión de un usuario que previamente ha iniciado sesión.

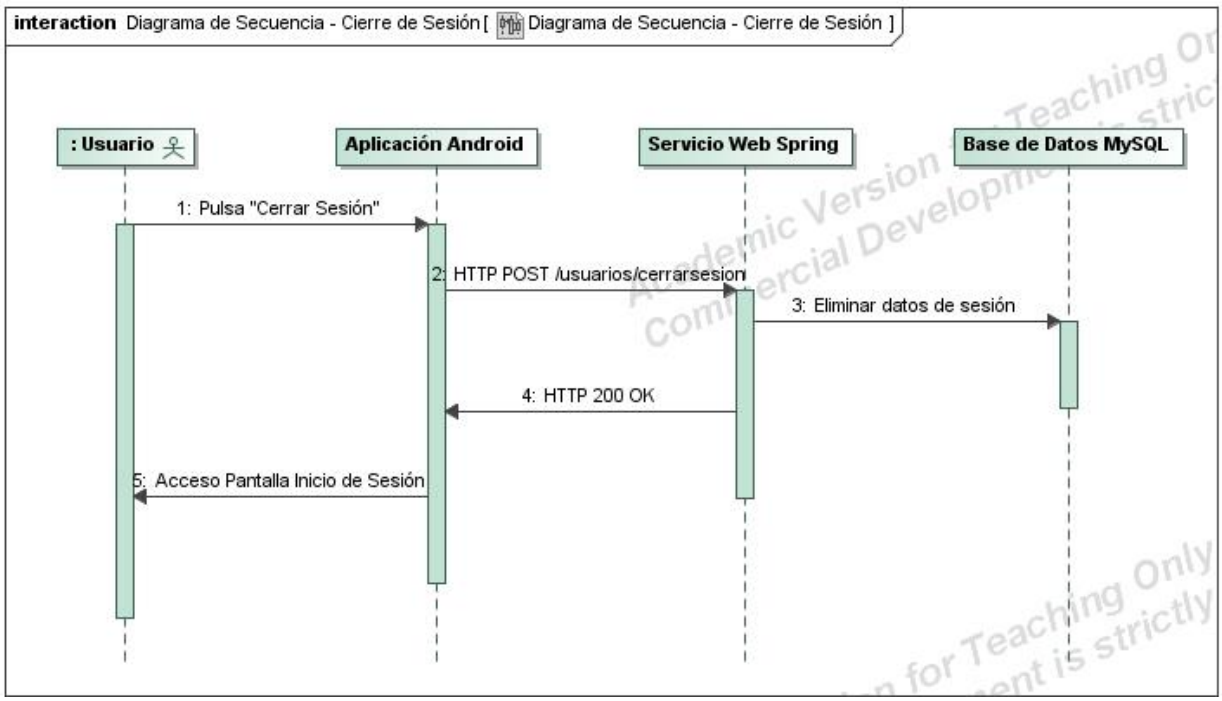

Ilustración 66: Diagrama de Secuencia – Cierre de Sesión

En el siguiente diagrama de secuencia de la Ilustración 67, se muestra el registro de señales de tráfico detectadas mediante el uso de la aplicación Android en la base de datos MySQL. Se observa la lógica y selección de las señales a registrar tras como se ha explicado en capítulos anteriores: Sólo puede haber una señal registrada en la misma posición, y esta será de todas la de mayores dimensiones.

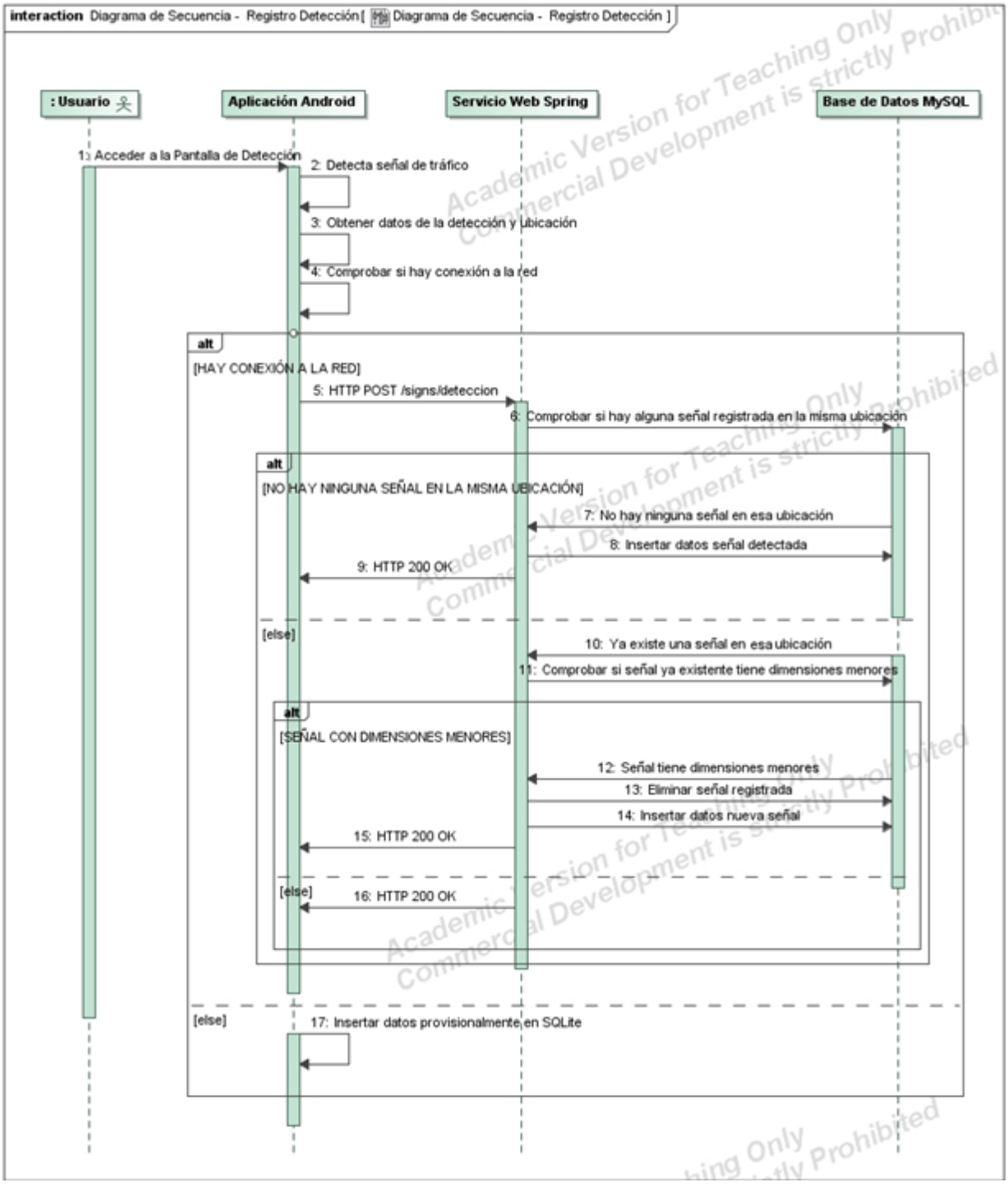

Ilustración 67: Diagrama de Secuencia – Registro Detección

Por último, en la Ilustración 68, se presenta el diagrama de secuencia de la sincronización entre la base de datos SQLite y la de MySQL. El almacenamiento de las señales de tráfico en MySQL se hace con la misma lógica que en el diagrama anterior, como se puede observar.

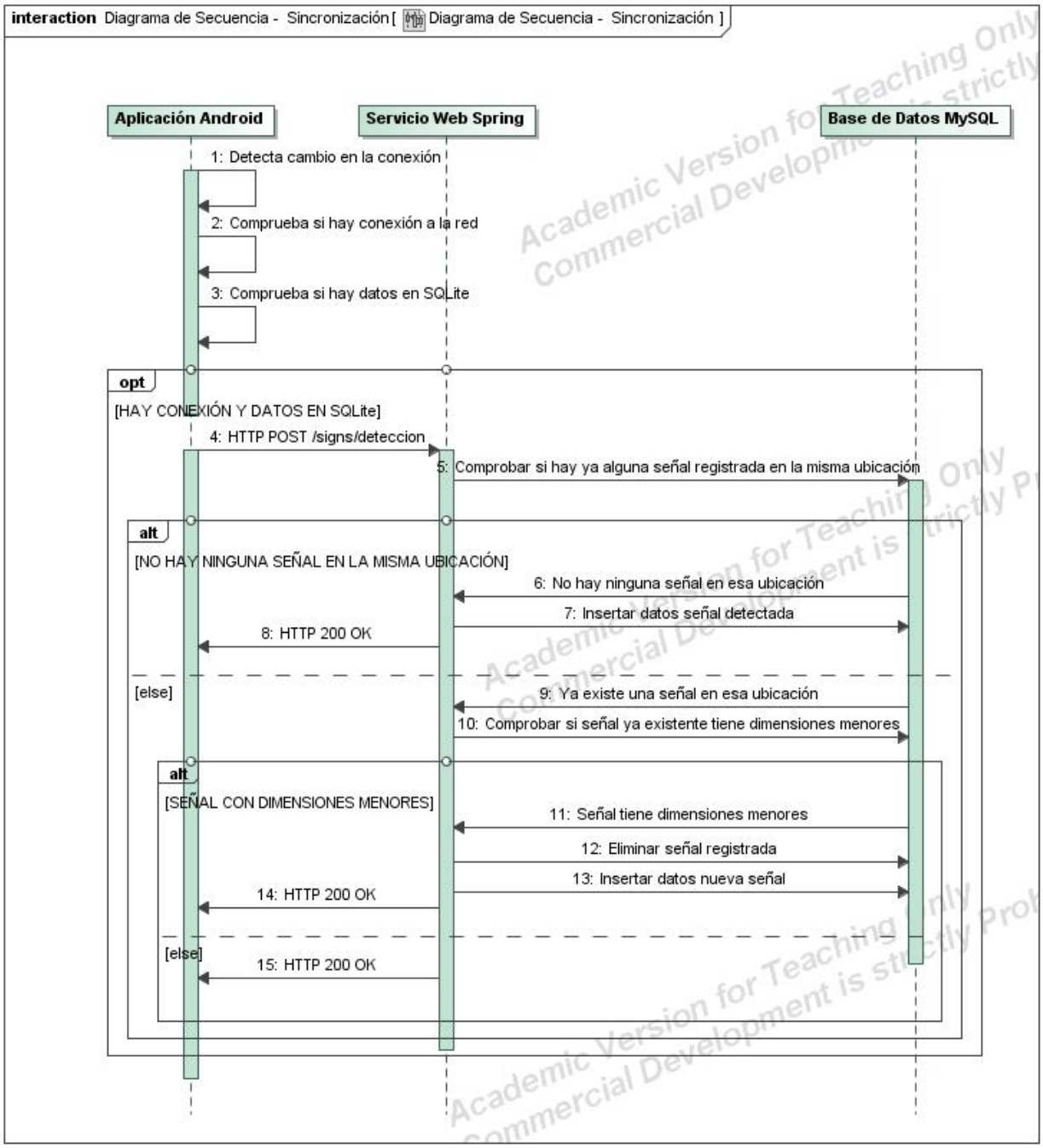

Ilustración 68: Diagrama de Secuencia – Sincronización

Nos centraremos ahora en el funcionamiento de la aplicación Android. En la Ilustración 69, podemos ver un diagrama de casos de uso en el que se especifican todas las funcionalidades a las que el usuario puede acceder a través de iCodriver:

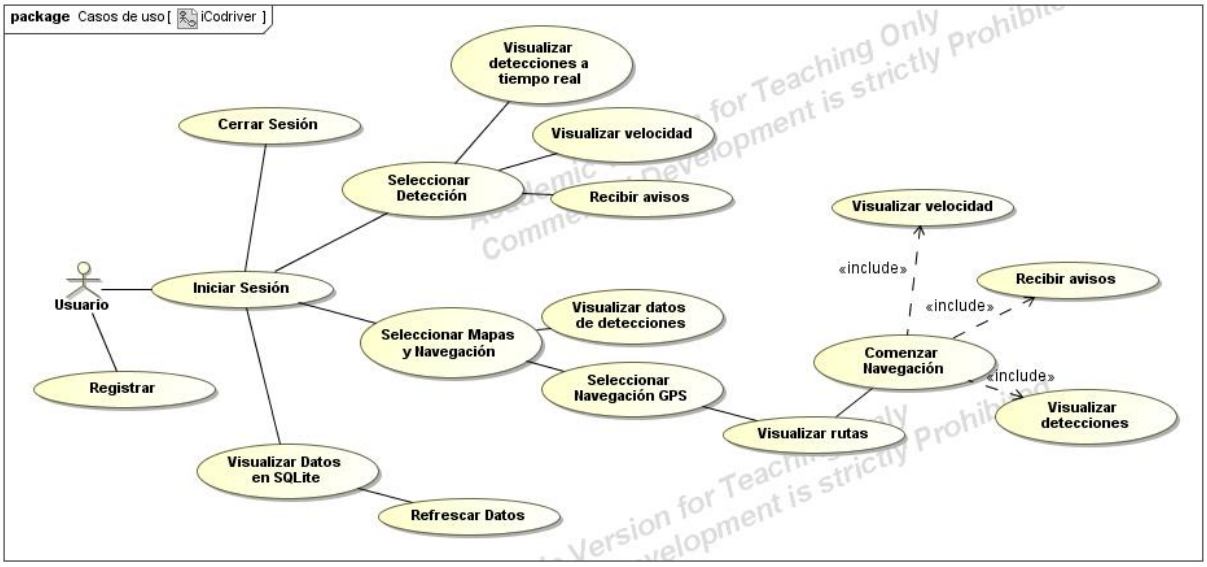

Ilustración 69: Diagrama de Casos de Uso – Aplicación Android iCodriver

En los apartados siguientes, primero detallaremos las clases usadas para el correcto funcionamiento de la aplicación, así como sus ficheros de configuración, y, por último, mostraremos la interfaz de usuario diseñada y explicaremos las funcionalidades o casos de uso que se presentan en la misma.

7.1 Clases

Para detallar la utilidad de cada una de las clases empleadas para la aplicación Android, se han dividido en 3 categorías según su funcionamiento: clases para la conexión con el servicio web, clases para el funcionamiento principal, y clases para la conexión con la base de datos SQLite.

Clases para la conexión con el servicio web

- **Usuario** (Ilustración 70): Contiene el nombre de usuario, nombre completo, email y contraseña del usuario registrado en la aplicación, además de los *getters* y *setters* de estos atributos y el constructor de la clase. Se corresponde con la entidad "Usuario" creada para el servicio web Spring.

```
import ...

public class Usuario implements Serializable {

    @SerializedName("username")
    @Expose
    private String username;
    @SerializedName("nombre")
    @Expose
    private String nombre;
    @SerializedName("email")
    @Expose
    private String email;
    @SerializedName("contraseña")
    @Expose
    private String contraseña;
```

Ilustración 70: Clase Usuario – Aplicación Android

- **TrafficSign** (Ilustración 71)**:** Contiene la información de una señal de tráfico, esto es, id, posición (latitud y longitud), dimensiones (alto y ancho), clase y username, además de los *getters* y *setters* de estos atributos y los constructores de la clase. Se corresponde con las entidades "TrafficSign" y "GlobalTrafficSign" creadas para el servicio web y mencionadas en el capítulo anterior.

```
public class TrafficSign {
    private int id;
    private Double longitud;
    private Double latitud;
    private Double ancho;
    private Double alto;
    private String clase;
    private String username;
    private String hora;

    public TrafficSign() { super(); }

    public TrafficSign(Double longitud, Double latitud, Double ancho, Double alto, String clase,
                       String username) {
```

Ilustración 71: Clase TrafficSign – Aplicación Android

- **Service (**Ilustración 72**):** Clase en la que se define la declaración API, que se corresponde con la API REST mostrada en la Tabla 1 y Tabla 2 en el capítulo anterior. Cada uno de estos métodos de solicitudes HTTP, debe tener una anotación que proporcione el método de solicitud (GET, POST, PUT, PATCH, DELETE, OPTIONS y HEAD) y la URL relativa, como podemos observar en la Ilustración 72. También se pueden especificar los parámetros de la consulta URL.

  De esta forma, y gracias a Retrofit que facilita la creación de solicitudes HTTP en Android, así como la serialización de datos, cada vez que necesitemos hacer un registro o consulta en el servicio web Spring, lo único que debemos hacer es construir la clase "Retrofit", como vemos en la Ilustración 73, con la URL base y llamar a uno de estos métodos especificados en la clase "Service".

```
public interface Service {

    // Registrar un nuevo usuario.
    @POST("registrar")
    Call<ResponseBody> save(@Body Usuario user);

    // Iniciar sesión.
    @GET("login")
    Call<Usuario> login(@Header("Authorization") String authHeader, @Query("username") String username);

    // Guardar señal detectada
    @POST("deteccion")
    Call<ResponseBody> saveDetection(@Header("Cookie") String sessionid, @Body TrafficSign trafficSign);

    // Obtener señales detectadas por el usuario actual.
    @GET("listar/{username}")
    Call<List<TrafficSign>> getSeñales(@Header("Cookie") String sessionid, @Path("username") String username);

    // Obtener todas las señales detectadas por todos los usuarios.
    @GET("listarglobal")
    Call<List<TrafficSign>> getGlobalSeñales(@Header("Cookie") String sessionid);
```

Ilustración 72: Clase Service – Aplicación Android

```
Retrofit retrofit = new Retrofit.Builder().baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
Service service = retrofit.create(Service.class);
Call<ResponseBody> call = service.saveDetection(sessionid, ts);
call.enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
```

Ilustración 73: Instanciación Clase Retrofit

Clases para el funcionamiento principal

- **SplashActivity** (Ilustración 74)**:** Esta clase es la que define la funcionalidad de la primera pantalla visible para el usuario en la aplicación. Al iniciar la aplicación, comprueba si el usuario ya ha iniciado sesión anteriormente, si tiene una sesión abierta. Si la respuesta es sí, instancia un objeto de la clase *Intent* que lleva al usuario hacia la clase "MainActivity" y si la respuesta es no, hacia "LoginActivity".

```
public class SplashActivity extends AppCompatActivity {

    private static String TAG = "SplashActivity";
```

Ilustración 74: Clase SplashActivity – Aplicación Android

- **RegisterActivity** (Ilustración 75): Esta clase se define para el registro de nuevos usuarios en la aplicación Android. Realiza una petición al servicio web para insertar los datos del nuevo registro en la tabla "usuarios". Antes de realizar la petición al servicio web, se comprueba si las dos contraseñas introducidas coinciden, en caso contrario, se genera un aviso para informar al usuario. Por otro lado, si al realizar la petición, el servicio web devuelve un error indicando que el parámetro "username" introducido por el usuario ya existe, se generará un aviso para informar al usuario y el registro no se realizará.

```
public class RegisterActivity extends AppCompatActivity {

    private static String TAG = "RegisterActivity";
```

Ilustración 75: Clase RegisterActivity – Aplicación Android

- **LoginActivity** (Ilustración 76)**:** Esta clase se define para el inicio de sesión de los usuarios en la aplicación Android, además de permitir a un usuario no registrado acceder a la pantalla de registro, cuyo funcionamiento maneja la clase "RegisterActivity" mencionada anteriormente. Realiza una petición al servicio web para comprobar que las credenciales introducidas son correctas, de manera que el usuario quede autenticado y pueda acceder a las distintas funcionalidades de la aplicación con sus datos, además de crear una nueva sesión que se mantiene abierta hasta que el mismo usuario la cierre.

```
public class LoginActivity extends AppCompatActivity {

    private static String TAG = "LoginActivity";
```

Ilustración 76: Clase LoginActivity – Aplicación Android

- **MainActivity** (Ilustración 77)**:** Es la clase central, la cuál proporciona acceso a las funcionalidades más importantes de la aplicación. Crea una instancia de las clases "DetectionFragment", que da acceso a la clase "DetectionActivity", "MapsFragment", que da acceso a la clase "MapsActivity" y "SyncFragment". Para su funcionamiento, se ha hecho uso del componente *Navigation* de Android Jetpack, que permite implementar la navegación, esto es, la barra de la aplicación y el panel lateral de navegación, que da acceso a las clases mencionadas. A través de esta clase se puede realizar también el cierre de sesión, en el cuál se realizaría una petición al servicio web, que borraría las cookies y la sesión actual del usuario. Si la petición se realiza correctamente, el usuario volvería a la clase "LoginActivity" al cerrar sesión.

```
public class MainActivity extends AppCompatActivity {

    private static String TAG = "MainActivity";
```

Ilustración 77: Clase MainActivity – Aplicación Android

- **DetectionActivity** (Ilustración 78)**:** Desde esta clase se realiza la funcionalidad de la detección de señales de tráfico de velocidad a tiempo real a través de las imágenes captadas por la cámara del teléfono móvil. Para el tratamiento de las imágenes recibidas por la cámara y la detección con YOLO, hemos usado la librería OpenCV, previamente descargada desde su página official [32] e instalada en nuestro proyecto de Android Studio. También partimos de la configuración y los pesos de tiny-yolov3 entrenados como se indica en el Capítulo 5, incluidos en la carpeta *assets* de nuestro proyecto Android.

```
public class DetectionActivity extends AppCompatActivity implements CameraBridgeViewBase.CvCameraViewListener2 {

    private static String TAG = "DetectionActivity";
```

Ilustración 78: Clase DetectionActivity – Android Studio

En la Ilustración 79 podemos ver el método sobreescrito *onCameraFrame,* que es llamado cada vez que se recibe una imagen instantánea o *frame.* Esta imagen instantánea de entrada será convertida en una matriz de la clase *Mat.* La clase *Mat* es la clase de matriz principal en OpenCV, que puede ser de distintos tipos. A continuación, hacemos una conversión de color, de 4 canales a 3, esto es, a formato RGB. RGB (*Red, Green, Blue*) es un formato de 3 canales que contiene datos para rojo, verde y azul. RGBA (*Red, Green, Blue, Alfa*) es un formato de 4 canales que contiene datos para rojo, verde, azul y alfa. El canal Alfa no tiene otra utilidad que hacer que el color sea transparente u opaco (o parcialmente transparente; translúcido), lo cual no nos hará falta en nuestro caso.

Mediante el método *blobFromImage,* del módulo *Dnn* (*Deep Neural Networks*) de OpenCV, que vemos en la Ilustración 79 también, crearemos un BLOB de 4 dimensiones a partir de la imagen. Un BLOB (*Binary Large Objects),* son elementos utilizados en las bases de datos para almacenar datos de gran tamaño, como imágenes, archivos de sonido y otros objetos multimedia. En nuestro caso lo

usaremos para almacenar la imagen instantánea, además, a través del método, podemos también, opcionalmente, redimensionar y cortar la imagen entre otras cosas, estos parámetros opcionales los podemos ver detallados en la Tabla 3.

Por último, alimentamos la red neuronal YOLO previamente cargada, con la entrada.

```
// Trigger: Every frame received by camera
@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    Mat frame = inputFrame.rgba(); // Convertir el frame de entrada en una matriz de la clase Mat.
    Imgproc.cvtColor(frame, frame, Imgproc.COLOR_RGBA2RGB); // Conversión de color. De 4 canales a 3.
    Mat imageBlob = Dnn.blobFromImage(frame,
             scalefactor: 0.00392,
            new Size( width: 608, height: 608),
            new Scalar(0, 0, 0),
             swapRB: false,
             crop: false); // Crea un BLOB de 4 dimensiones a partir de la imagen.

    // Establecemos el valor de entrada para la red.
    yolo.setInput(imageBlob);
```

Ilustración 79: Código DetectionActivity (1)

size	Tamaño especial de la imagen de salida.
mean	Escalar con valores medios que se restan de los canales de color. Se pretende que los valores estén en orden (media-R, media-G, media-B) si la imagen tiene un orden BGR y swapRB es verdadero.
scalefactor	Multiplicador de valores de imagen.
swapRB	Bandera que indica que es necesario intercambiar el primer y el último canal en la imagen de 3 canales.
crop	Bandera que indica si la imagen se recortará después del cambio de tamaño o no

Tabla 3: Parámetros método *blobFromImage*

Una vez la red neuronal ha recibido el BLOB de la imagen como entrada y ha realizado la tarea de detección para la que ha sido entrenada, obtenemos los resultados. En la Ilustración 80, vemos que primero debemos obtener el nombre de las capas de nuestra red neuronal cuyas salidas no están conectadas a ninguna otra capa de la red, pues estas serán las capas de salida que contendrán los resultados de la detección, las predicciones. Una vez obtenidas estas capas, con el método *forward*, se calcula las salidas de las mismas y se almacenan en una variable. Los parámetros que debemos pasar a este método *forward* están detallados en la Tabla 4.

```
// Obtenemos el resultado de las capas de salida.
java.util.List<Mat> result = new java.util.ArrayList<Mat>( initialCapacity: 2); // Para almacenar el resultado de la detección.
List<String> outBlobNames = new java.util.ArrayList<>();
outBlobNames = yolo.getUnconnectedOutLayersNames(); // Devuelve los nombres de las capas con salidas no conectadas.
yolo.forward(result,outBlobNames);
```

Ilustración 80: Código DetectionActivity (2)

outputBlobs	Variable donde se almacenarán todos los BLOBs de salida de cada capa especificada en el parámetro *outBlobNames.*
outBlobNames	Nombres de las capas cuyas salidas son necesarias para conseguir los resultados de la detección.

Tabla 4: Parámetros método *forward*

El siguiente paso será procesar el resultado de la detección proporcionado por la red neuronal. Básicamente la red neuronal actúa como se describió en el Apartado 4.3.1., donde se obtienen los cuadros delimitadores con sus respectivas confianzas. Estos cuadros delimitadores a las salidas de la red están representados por un vector de número de clases además de 5 elementos. Los primeros 4 elementos representan la posición en el eje horizontal del centro, la posición en el eje vertical, ancho y alto. El quinto, representa la confianza de que el cuadro delimitador encierre un objeto. El resto de los elementos son la confianza asociada con cada clase, (es decir, el tipo de objeto). Al cuadro se le asigna la clase correspondiente a la puntuación más alta. El puntuaje más alto para un cuadro delimitador también se llama confianza. Si la confianza de un cuadro es inferior al umbral dado, el cuál se ha establecido en 60% y es representado por la variable *confThreshold* en la Ilustración 81, el cuadro delimitador se descarta y no se considera para su posterior procesamiento. Podemos ver todo este proceso en la Ilustración 81. Este código se repite para cada uno de los cuadros delimitadores, por lo que está dentro de un bucle.

```
Mat row = level.row(j);
Mat scores = row.colRange(5, level.cols()); // Crea una matriz con center_x, center_y,
                                            // ancho, alto y confianza
Core.MinMaxLocResult mm = Core.minMaxLoc(scores); // Obtenemos el valor max y min de las puntuaciones.
float confidence = (float)mm.maxVal;
Point classIdPoint = mm.maxLoc;  // Asignamos la clase con mayor puntuación al cuadro delimitador.

if (confidence > confThreshold)
{
    int centerX = (int)(row.get( row: 0, col: 0)[0] * frame.cols());
    int centerY = (int)(row.get( row: 0, col: 1)[0] * frame.rows());
    width   = (row.get( row: 0, col: 2)[0] * frame.cols());
    height  = (row.get( row: 0, col: 3)[0] * frame.rows());

    int left    = centerX - width.intValue()  / 2;
    int top     = centerY - (height.intValue()) / 2;

    clsIds.add((int)classIdPoint.x);
    confs.add((float)confidence);

    // Almacenamos todas las detecciones (con confianza mayor a umbral)
    rects.add(new Rect(left, top, width.intValue(), height.intValue()));
}
```

Ilustración 81: Código DetectionActivity (3)

Los resultados con confianza mayor al umbral quedan almacenados y serán sometidos al proceso de supresión no máxima (*Non-maximum procedure, NMS).* La supresión no máxima es el proceso por el cuál se reduce el número de cuadros delimitadores superpuestos. Se controla mediante el parámetro *nmsThresh* que se observa en la Ilustración 82. Si *nmsThresh* se establece demasiado bajo, por ejemplo en 0.1, es posible que no detectemos objetos superpuestos de la misma clase o clases diferentes. Pero si se establece demasiado alto, por ejemplo 1, entonces obtendremos múltiples cajas para el mismo objeto. Realizamos este procedimiento con el método *NMSBoxes* del módulo Dnn de OpenCV, cuyos

parámetros pasados se describen en Tabla 5.

```
// Aplicamos non-maximum suppression procedure.
int ArrayLength = confs.size();
if (ArrayLength>=1) {
    float nmsThresh = 0.2f; // Parámetro que controla la supresión no máxima.

    MatOfFloat confidences = new MatOfFloat(Converters.vector_float_to_Mat(confs));
    Rect[] boxesArray = rects.toArray(new Rect[0]);
    MatOfRect boxes = new MatOfRect(boxesArray);
    MatOfInt indices = new MatOfInt();
    // Realiza la supresión no máxima dados los cuadros y puntuaciones correspondientes.
    Dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThresh, indices);
```

Ilustración 82: Código DetectionActivity (4)

bboxes	Conjunto de cuadros delimitadores a los que aplicar NMS.
scores	Conjunto de confianzas/puntuaciones correspondientes.
score_threshold	Umbral usado para filtrar los cuadros delimitadores por puntuación.
nms_threshold	Umbral usado en NMS.
indices	Los índices guardados de los cuadros delimitadores (bboxes) después de NMS.

Tabla 5: Parámetros método NMSBoxes

Por último, para completar el proceso de detección, dibujamos los cuadros resultantes en la imagen, así como la clase final detectada y la confianza final, tal como vemos en la Ilustración 83.

```
// Dibujamos los cuadros resultantes.
int[] ind = indices.toArray();
for (int i = 0; i < ind.length; ++i) {

    int idx = ind[i];
    Rect box = boxesArray[idx];

    int idGuy = clsIds.get(idx);

    float conf = confs.get(idx);

    int intConf = (int) (conf * 100);

    Imgproc.putText(frame, text: "Límite de velocidad: "+classesNames.get(idGuy)+"Km/h" + " " + intConf + "%",
            new Point( x: 5, y: 50), Imgproc.FONT_HERSHEY_SIMPLEX, fontScale: 1, new Scalar(255,255,0), thickness: 2);
    Imgproc.rectangle(frame, box.tl(), box.br(), new Scalar(255, 0, 0), thickness: 2);
```

Ilustración 83: Código DetectionActivity (5)

En esta clase, además de llevar a cabo la tarea de detección ya detallada, se registran estas detecciones en la base de datos, de modo que para cada detección, se comprueba la conexión de red, y si es posible contactar con el servidor web Spring visto en el Capítulo 6, se realiza la petición al mismo para insertar los datos de las señales de tráficos detectadas, su clase, dimensiones del cuadro delimitador (ancho y alto), el nombre de usuario y la localización (latitud y longitud) captada por los sensores, en este caso el GPS del dispositivo móvil. Si la conexión con el servicio web no es posible, se introducen estos mismos datos provisionalmente en la base de datos local SQLite.

Gracias a la función de GPS también, se mide la velocidad en cada momento y se presenta por pantalla, dando avisos si esta supera la velocidad máxima de la señal detectada.

- **MapsActivity** (Ilustración 84)**:** Esta clase muestra un mapa con las mejores detecciones hechas por todos los usuarios en su posición correspondiente en el mapa y las mejores detecciones hechas por el mismo usuario a través de un botón que cambia entre estos dos estados. Además implementa una selección de búsqueda donde se muestran solo las señales detectadas en la zona/ciudad introducida. Para obtener estos mapas basados en datos de Google Maps a la aplicación, se ha usado *Maps SDK for Android* [21], servicio de *Google Platform*.

```
public class MapsActivity extends FragmentActivity implements
        OnMapReadyCallback,
        GoogleMap.OnMarkerClickListener {

    private static String TAG = "MapsActivity";
```

Ilustración 84: Clase MapsActivity – Android Studio

Además, esta clase da acceso también a la navegación GPS. A partir de un origen y destino introducido por el usuario, dibujará la ruta en el mapa e irá indicando en cada momento la velocidad y las señales de tráfico registradas previamente que el usuario va encontrando a lo largo del recorrido, así como avisos si se supera la velocidad máxima permitida. Para obtener estos mapas se ha usado también *Maps SDK for Android*, y para obtener las rutas entre dos puntos se ha usado la *Directions API* [22], un servicio de *Google Platform* también que calcula direcciones entre ubicaciones mediante una solicitud HTTP. Para almacenar las respuestas a esta solicitud, debemos instanciar un objeto de tipo "DirectionResults", y otro de tipo "RouteDecode", como se observa en Ilustración 85, que contendrá el método de decodificación para poder crear una polilínea (polyline) que representará el recorrido. Una polilínea en el context de Google Maps, es una lista de puntos, donde se dibujan segmentos de línea entre puntos consecutivos. La creación de esta polilínea se observa en la Ilustración 86.

```
public void onResponse(Call<DirectionResults> call, Response<DirectionResults> response) {
    if (response.isSuccessful()){
        DirectionResults directionResults = response.body();
        ArrayList<LatLng> routelist = new ArrayList<>();
        if(directionResults.getRoutes().size()>0){
            ArrayList<LatLng> decodelist;
            Route routeA = directionResults.getRoutes().get(0);
            if(routeA.getLegs().size()>0){
                List<Steps> steps= routeA.getLegs().get(0).getSteps();
                Steps step;
                com.example.opencvapplication.Entities.Directions.Location location;
                String polyline;
                for(int i=0 ; i<steps.size();i++){
                    step = steps.get(i);
                    location =step.getStart_location();
                    routelist.add(new LatLng(location.getLat(), location.getLng()));
                    Log.i(TAG,  msg: "Start Location :" + location.getLat() + ", " + location.getLng());
                    polyline = step.getPolyline().getPoints();
                    decodelist = RouteDecode.decodePoly(polyline);
                    routelist.addAll(decodelist);
                    location =step.getEnd_location();
                    routelist.add(new LatLng(location.getLat() ,location.getLng()));
                    Log.i(TAG, msg: "End Location :"+location.getLat() +", "+location.getLng());
                }
            }
        }else{
            Log.d(TAG,  msg: "No se han encontrado rutas");
        }
```

Ilustración 85: Código MapsActivity (1)

```
if(routelist.size()>0){
    PolylineOptions rectLine = new PolylineOptions().width(10).color(
            Color.RED);

    for (int i = 0; i < routelist.size(); i++) {
        rectLine.add(routelist.get(i));
    }
    // Adding route on the map
    mMap.addPolyline(rectLine);
    MarkerOptions markerOptions = new MarkerOptions();
    markerOptions.position(destinolatLng[0]);
    markerOptions.draggable(true);
    mMap.addMarker(markerOptions);
}
```

Ilustración 86: Código MapsActivity (2)

Clases para la conexión con la base de datos SQLite

SQLite es un sistema de bases de datos que se ajusta perfectamente a las aplicaciones móviles. Este sólo requiere un fichero para almacenar los datos, ya que la lógica de funcionamiento se debe implementar en la plataforma que quiera usar estos datos, en este caso Android, cuyo SDK posee soporte completo para SQLite.

A continuación, veremos las distintas clases que usa Android para la comunicación con SQLite, así como para la sincronización entre esta base de datos y la base de datos de MySQL.

- **SyncFragment** (Ilustración 87)**:** Carga una lista con todas las señales de tráfico almacenadas en la base de datos local SQLite con su fecha y hora de detección. Además cuenta con la funcionalidad de refrescar, con lo que se vuelve a cargar la lista por si se han producido cambios.

```
public class SyncFragment extends Fragment {

    private static String TAG = "SyncFragment";
```

Ilustración 87: Clase SyncFragment – Android Studio

- **SyncAdapter:** Esta clase sirve como adaptador de la vista de la clase anterior.
- **ConexionSQLiteHelper** (Ilustración 88)**:** En esta clase se crea la tabla "usuario_señales" en la base de datos SQLite, donde se guardarán provisionalmente los datos de las señales de tráfico detectadas mediante el uso de la aplicación en el caso de que no haya conexión a la red o imposibilidad de conectar con el servicio web.

```
public class ConexionSQLiteHelper extends SQLiteOpenHelper {

    public ConexionSQLiteHelper(@Nullable Context context,
                                @Nullable String name,
                                @Nullable SQLiteDatabase.CursorFactory factory,
                                int version) {

        super(context, name, factory, version);
    }
```

Ilustración 88: Clase ConexionSQLiteHelper – Android Studio

- **SQLUtils** (Ilustración 89)**:** Contiene todas las cadenas relativas a la base de datos SQLite, esto es, nombre de la base de datos, nombre de las tablas, nombre de los campos y sentencias SQL usadas para insertar o consultar los datos de la misma.

```
public class SQLUtils {

    // Nombre de la base de datos
    public static final String NOMBRE_BD = "trafficsign_db";

    // Campos tabla señales
    public static final String TABLA_SEÑALES="usuario_señales";
    public static final String CAMPO_ID="id";
    public static final String CAMPO_LONGITUD="longitud";
    public static final String CAMPO_LATITUD="latitud";
    public static final String CAMPO_CLASE="clase";
    public static final String CAMPO_ANCHO="ancho";
    public static final String CAMPO_ALTO="alto";
    public static final String CAMPO_HORA="hora";
    public static final String CAMPO_USERNAME="username";

    // Sentencias SQL
    public static final String CREAR_TABLA_SEÑALES="CREATE TABLE usuario_señales (\n" +
            "  `id` INTEGER PRIMARY KEY autoincrement,\n" +
```

Ilustración 89: Clase SQLUtils – Android Studio

- **NetworkMonitor** (Ilustración 90)**:** Clase que implementa la interfaz BroadcastReceiver, cuyo método onReceive, es llamado cada vez que se produce un cambio en el estado de conexión a la red, esto es, si se pierde la conexión, si vuelve la conexión, si pasamos de usar datos móviles a usar conexión Wi-fi y viceversa. De este modo, sabremos cuando el dispositivo móvil vuelve a tener conexión con el servidor web, para consultar si hay alguna señal en la base de datos local SQLite que debamos insertar en la base de datos MySQL. Una vez esta inserción se ha hecho correctamente, borramos los datos ya sincronizados de SQLite.

```
public class NetworkMonitor extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if(checkNetworkConnection(context)){
```

Ilustración 90: Clase NetworkMonitor – Android Studio

En nuestro caso, hemos registrado dinámicamente una instancia de esta clase en "MainActivity", "DetectionActivity" y "SyncFragment" con el método *registerReceiver*, como vemos en la Ilustración 91.

```
// NetworkMonitor: Listener SYNC BBDD
networkMonitor = new NetworkMonitor();
networkMonitor.setDataSession( context: this, TAG, "http://192.168.0.20:8080/signs/", usuario.getUsername(), sessionid);
intentFilter = new IntentFilter();
intentFilter.addAction(ConnectivityManager.CONNECTIVITY_ACTION);
registerReceiver(networkMonitor, intentFilter);
```

Ilustración 91: Registro NetworkMonitor

7.2 Ficheros de configuración

Los ficheros de configuración que se usan en la Aplicación Android son:

- **AndroidManifest.xml**

  Todos los proyectos Android deben tener un archivo "AndroidManifest.xml" situado en la raíz de la fuente del proyecto. Describe información esencial de la aplicación y, entre otras cosas, debe declarar: nombre del paquete de la aplicación, componentes de la aplicación, esto es, actividades, servicios, receptores de emisiones y proveedores de contenido, permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones, funciones de hardware y software que requiere la aplicación. Este fichero, al usar Android Studio, se generó automáticamente, y la mayoría de los elementos se fueron agregando a medida que se compila la aplicación.

  En la Ilustración 92, podemos ver las subclases de *Activity* declaradas utilizando el componente de aplicación <*activity*>. Si una subclase no se declara en este fichero de configuración, el sistema no puede inciarlo. El nombre de la subclase se debe indicar con el atributo "name".

  Además, para la primera actividad que se lanza al iniciar la aplicación, se han indicado dos tipos de intent, "Android.intent.action.MAIN", que indica que esa actividad es el punto de entrada de la aplicación, al iniciarse la aplicación se crea esta actividad, y "android.intent.category.LAUNCHER", que indica que la actividad marcada como punto de entrada debe aparecer en el lanzador de la aplicación.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.opencvapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="iCodriver"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name=".View.MainActivity"
            android:label="MainActivity"
            android:theme="@style/AppTheme.NoActionBar"></activity>
        <activity android:name=".View.SyncActivity" />
        <activity android:name=".View.SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".View.RegisterActivity"
            android:windowSoftInputMode="adjustPan"/>
        <activity android:name=".View.LoginActivity" />
```

Ilustración 92: Archivo AndroidManifest.xml

En la Ilustración 93, podemos ver la declaración de los permisos que necesita la aplicación, entre ellos: permiso para el uso de la cámara del dispositivo, permiso para acceder a la ubicación del dispositivo, permiso para conectarse a la red y al estado de la misma, y permiso para leer y escribir en la memoria del dispositivo.

```xml
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Ilustración 93: Permisos declarados en AndroidManifest.xml

Android usa Gradle como herramienta que permite la automatización de compilación, además de crear y empaquetar tu aplicación. Se utilizan dos archivos de compilación “build.gradle”, ambos ubicados en el directorio raíz del proyecto. Estos son:

- **build.gradle (Project: iCodriver):** En este fichero se añaden opciones de configuración que serán comunes a todos los módulos del proyecto. Se puede ver en la Ilustración 94.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.0.0"
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        mavenCentral()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Ilustración 94: Fichero build.gradle (Project: iCodriver)

- **build.gradle (Module: App):** En este fichero de configuración de nivel superior, se configuran los módulos específicos. En la Ilustración 95, se pueden ver los distintos módulos o dependencias.

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
    implementation project(path: ':java')
    implementation 'com.google.android.gms:play-services-maps:17.0.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.navigation:navigation-fragment:2.3.0'
    implementation 'androidx.navigation:navigation-ui:2.3.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    implementation 'com.ankuryadav.lib:volleylib:1.0.3'
    implementation 'com.loopj.android:android-async-http:1.4.9'
    implementation 'com.google.code.gson:gson:2.8.5'
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'de.hdodenhof:circleimageview:3.1.0'

}
```

Ilustración 95: Fichero build.gradle (Module: App)

- **AndroidManifest.xml (OpenCV):** Para añadir la librería de OpenCV, hemos importado su módulo para Android descargado desde la página official [32]. Este módulo, posee su propio manifiesto, el cuál podemos ver en la Ilustración 96, donde se declara entre otras cosas, el paquete y la versión utilizada de OpenCV.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.opencv"
    android:versionCode="4010"
    android:versionName="4.0.1">
</manifest>
```

Ilustración 96: AndroidManifest.xml para OpenCV

7.3 Interfaz de usuario

En este apartado detallaremos la interfaz de usuario de la aplicación Android, esto es, cada una de las pantallas por las que el usuario navegará en su interacción con la aplicación y sus funcionalidades.

Pantalla de Bienvenida:

La pantalla de bienvenida o *Splash Screen* es la primera pantalla visible para el usuario cuando se inicia la aplicación. Dura dos segundos y cambia a la siguiente pantalla. Muestra el icono de la aplicación iCodriver como se puede ver en la Ilustración 97. La siguiente pantalla a la que se accede después de estos dos segundos será la pantalla principal o la pantalla de inicio de sesión, dependiendo de si el usuario de la aplicación ha iniciado sesión ya anteriormente o no, respectivamente.

Ilustración 97: Pantalla de Bienvenida

Pantalla de Inicio de Sesión:

La pantalla de Inicio de Sesión se muestra en la Ilustración 98, en ella se introducen las credenciales del usuario para autenticarse y da acceso al resto de pantallas de la aplicación. Permite:

- Rellenar los campos de usuario y contraseña y pulsar el botón INICIAR SESIÓN para iniciar sesión con un usuario ya registrado en el sistema. Si las creedenciales introducidas son correctas, se accederá a la pantalla principal.
- Pulsar el texto "¿Aún no estás registrado? REGISTRATE" para acceder a la pantalla de Registro.

Pantalla de Registro:

La pantalla de registro se puede ver en la Ilustración 99, donde se permite introducir los datos para registrar un nuevo usuario en la aplicación.

Se deben rellenar todos los campos: Nombre completo, username, email y contraseña. Además hay un campo "Repite contraseña" que debe coincidir con el campo "Contraseña" para que se produzca el nuevo registro.

Una vez rellenados todos los campos se pulsa el botón "REGISTRAR" para insertar el nuevo usuario en la base de datos MySQL y volver a la pantalla de Inicio de Sesión donde se debe introducir las credenciales del usuario que se acaba de crear.

Ilustración 98: Pantalla de Inicio de Sesión

Ilustración 99: Pantalla de Registro

Pantalla Principal:

En esta pantalla se podrá navegar a tres distintas ventanas que nos dan acceso a tres funcionalidades distintas de la aplicación. Si es la primera vez que accedemos a la aplicación, se solicitará permitir el acceso a la cámara y la ubicación del dispositivo, como se puede observar en la Ilustración 100. En la Ilustración 101, se puede ver que para su diseño hemos usado el elemento de interfaz Navigation Drawer, que consiste en un menú deslizante desde la izquierda que se despliega a través de un botón o con un gesto de desplazamiento.

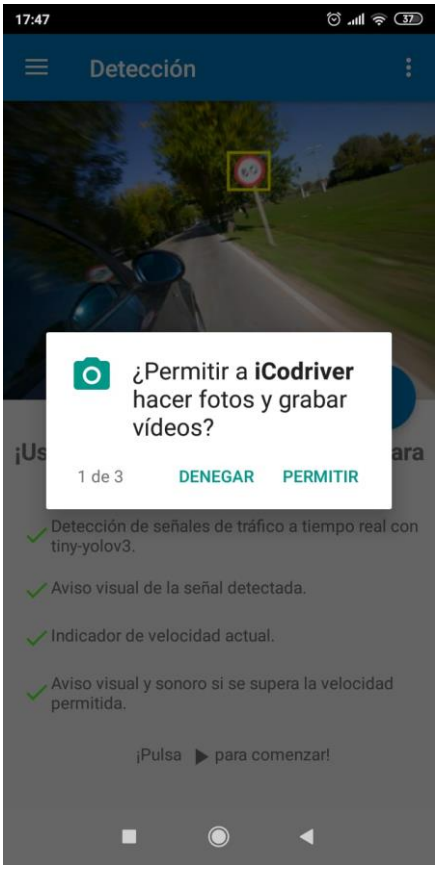

Ilustración 100: Pantalla Principal – Permisos

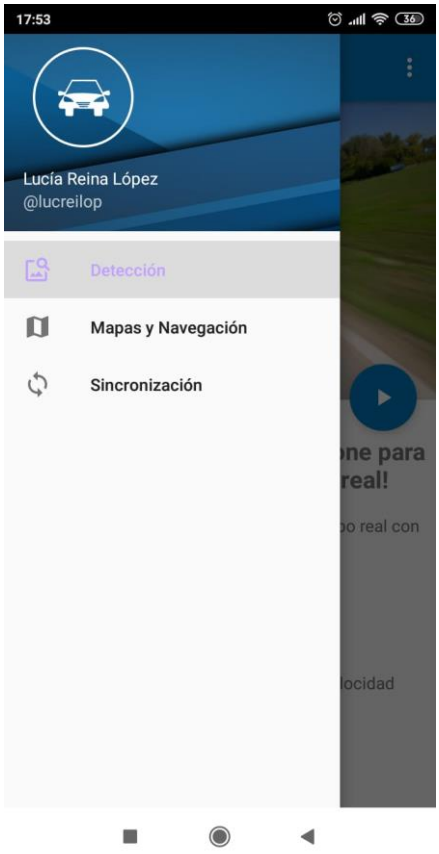

Ilustración 101: Pantalla Principal – Menú

En este menú se muestra el icono de la aplicación, el nombre completo y nombre de usuario del usuario autenticado y presentan tres opciones:

- Detección: En esta ventana se presenta la primera funcionalidad de la aplicación, como se puede ver en la Ilustración 102. Pulsando en el botón que aparece se accede a dicha funcionalidad, esto es, a la pantalla de detección.
- Mapas y Navegación: En esta ventana se presenta otra funcionalidad de la aplicación, como se observa en la Ilustración 103. Pulsando el botón que aparece se accede a dicha funcionalidad, esto es, a la pantalla de visualización en mapas y navegación.
- Sincronización: En esta ventana que se puede observar en las Ilustraciones 104 y 105, se presenta una lista con los datos provisionalmente guardados en la base de datos SQLite de las señales de tráfico detectadas con la hora de su detección. Además, en la esquina inferior derecha se presenta un botón el cuál sirve para refrescar la lista y ver qué detecciones se han ido guardando en la base de datos MySQL y a la vez borrando de SQLite.

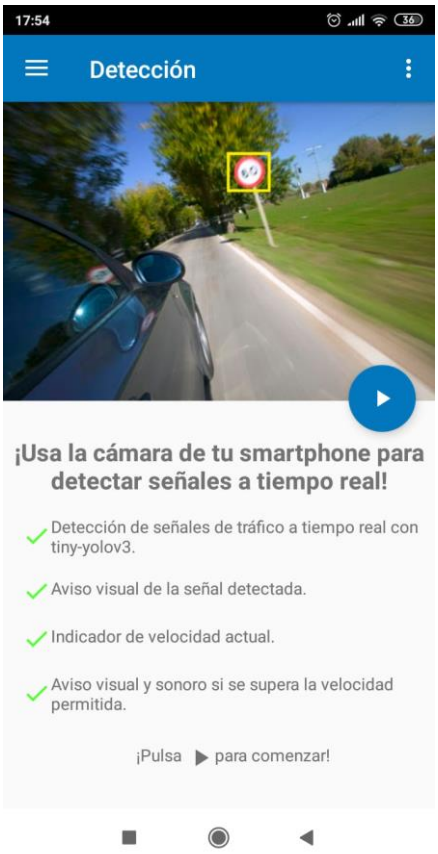

Ilustración 102: Pantalla Principal – Detección

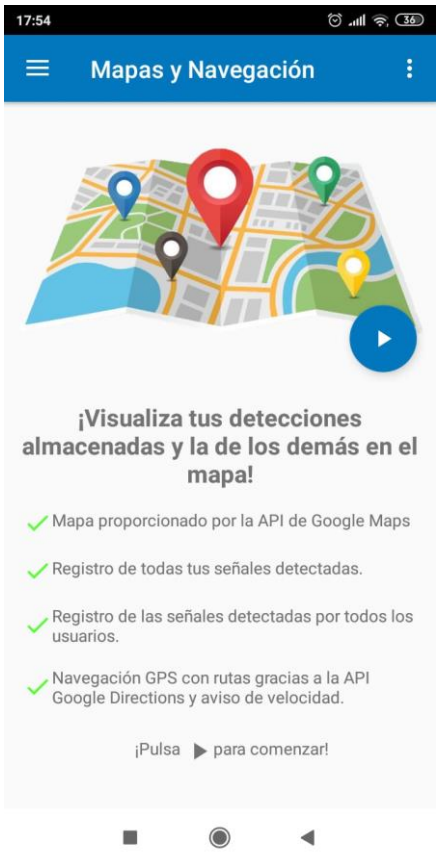

Ilustración 103: Pantalla Principal – Mapas y Navegación

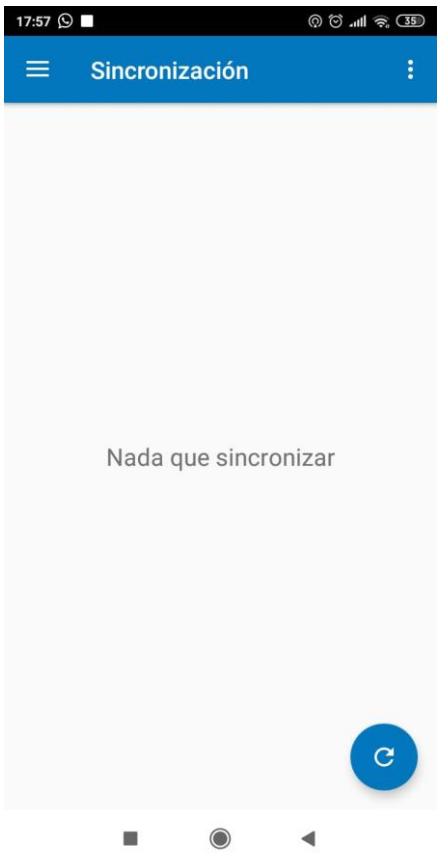

Ilustración 104: Pantalla Principal – Sincronización

Ilustración 105: Pantalla Principal – Sincronización (2)

Parte de este elemento *Navigation Drawer* es también la barra superior que se presenta en cada una de las vistas de las Ilustraciones 101, 102, 103 y 104, donde aparece el nombre de la selección del menu y la posibilidad de cerrar sesión que se muestra en la Ilustración 105. Si pulsamos esta opción, la sesión del usuario actual se eliminará de la base de datos MySQL y se volverá a la pantalla de inicio de sesión.

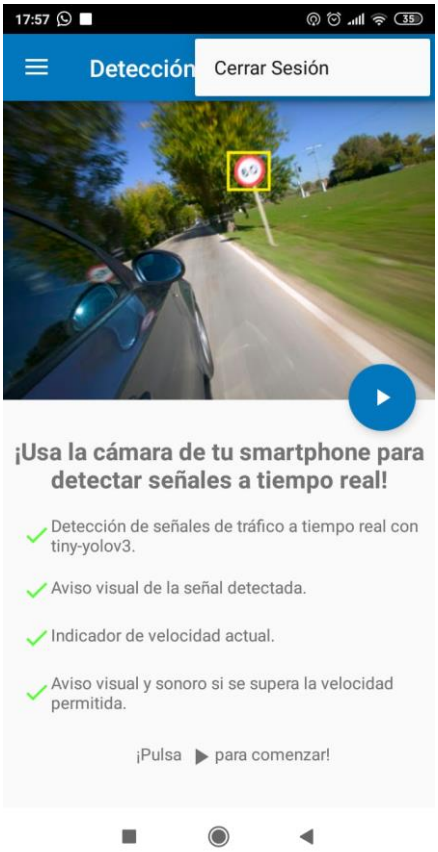

Ilustración 106: Pantalla Principal – Cerrar Sesión

Pantalla de Detección:

En esta pantalla se lleva a cabo la tarea de detección de señales de tráfico de velocidad. Para ello se ha usado el elemento JavaCameraView de la librería OpenCV, con el que la aplicación tomará los fotogramas de la vista previa de la cámara del dispositivo en tiempo real y los mostrará en modo pantalla completa. Permite:

- Realizar la detección de señales de tráfico de velocidad a tiempo real a cada fotograma que se capta por la cámara del dispositivo. Como se aprecia en la Ilustración 107, la detección se muestra en la misma imagen, indicada con un cuadrado rojo la posición de la señal, y en letras amarillas la clase a la que pertenece y la confianza en la parte superior derecha.
- Mostrar la imagen de la señal detectada como recordatorio de la velocidad máxima a la que se debe circular durante un tramo determinado del recorrido, tal como se ve en la esquina inferior izquierda de la pantalla en la Ilustración 107.
- Medir la velocidad a la que circula el vehículo en cada momento.
- Aviso si la velocidad actual supera a la velocidad máxima permitida detectada. Este aviso se produce tanto en una señal sonora al superar la velocidad máxima, como visual, pues la velocidad actual aparecerá de color rojo mientras esté situada por encima del límite que se debe respetar, como se ve en la Ilustración 108.

Ilustración 107: Pantalla de Detección – Detección

Ilustración 108: Pantalla de Detección – Velocidad máxima sobrepasada

Pantalla de Visualización en Mapas y Navegación.

En esta pantalla se presentan dos funcionalidades. La primera permite visualizar las señales detectadas en su posición correspondiente en el mapa, como se muestra en la Ilustración 109. En concreto:

- Al pulsar cualquier señal representada en el mapa, se puede ver toda su información el panel que se muestra en la Ilustración 110. Para cerrar el panel hay que pulsar el icono que aparece en la parte superior izquierda del mismo.
- Mostrar las señales detectadas en una ciudad o zona concreta del mapa al introducir el nombre de esta ciudad o zona en la barra superior con el icono de la lupa y el texto que dice “Buscar…”.
- Visualizar las mejores detecciones realizadas por todos los usuarios, o solo las realizadas por el usuario actual autenticado en la aplicación mediante el uso del botón situado en la parte central derecha de la pantalla con el icono de varias personas. Si se pulsa este botón, cambiará la imagen del mismo para mostrar el icono de una sola persona individual, indicando que se están mostrando solo las señales detectadas por el usuario actual.
- Pulsar el botón con el icono de navegación situado en la parte central derecha, lo cuál nos lleva a la segunda funcionalidad de esta pantalla.

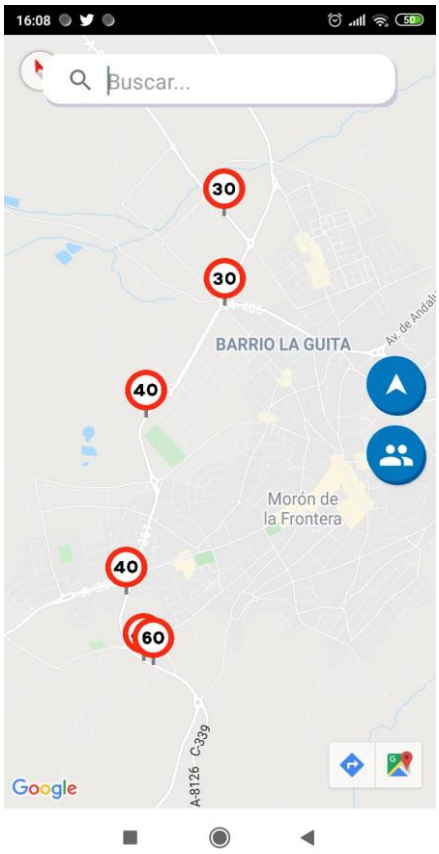

Ilustración 109: Pantalla Mapas – Visualización

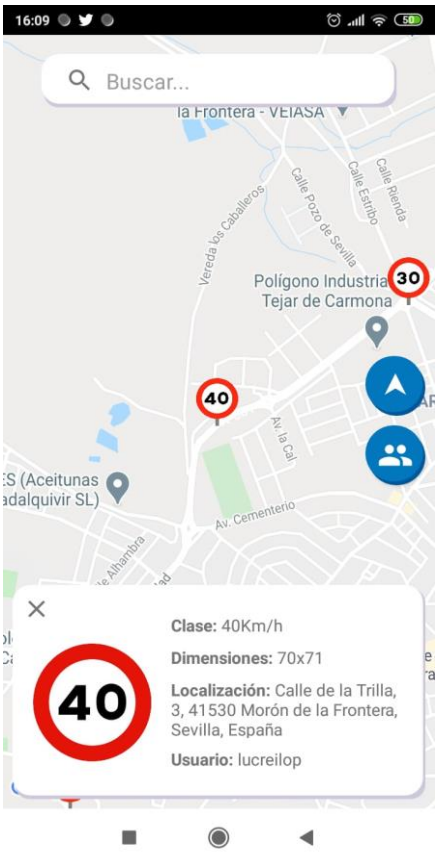

Ilustración 110: Pantalla Mapas – Panel de información

La segunda funcionalidad permite la navegación GPS, esto es, los usuarios podrán trazar rutas hacia sus destinos y guiarse por ellas durante la conducción, además de usar los datos almacenados para ser advertidos de las señales de tráfico de velocidad que deben respetar en cada tramo del recorrido. Concretamente permite:

- Al pulsar el botón con el icono de navegación, como se indicó anteriormente, aparecerá el panel superior mostrado en la Ilustración 111, que permite introducir el destino y origen (por defecto, la ubicación actual del usuario).
- La ruta de este recorrido podrá verse pulsando el botón "VER RUTA" del panel, tras lo cuál se mostrará también el botón "COMENZAR" que vemos en la Ilustración 112.
- Al pulsar el botón "COMENZAR", el mapa se centrará en la posición actual del usuario, como vemos en Ilustración 113, e irá mostrando su posición a lo largo de todo el recorrido.
- Avisos sobre las señales de tráfico previamente registradas por los usuarios que se encontrará durante su recorrido. Como vemos en la Ilustración 113, en la parte inferior izquierda de la pantalla, se mostrará la señal de tráfico presente en el tramo actual que el usuario esté recorriendo.
- Visualización de la velocidad actual que lleva el vehículo en cada momento.
- Avisos si se sobrepasa la velocidad máxima permitida. Este aviso se produce tanto en una señal sonora, como visual, pues la velocidad actual aparecerá de color rojo mientras esté situada por encima del límite que se debe respetar, como se ve en la Ilustración 114.

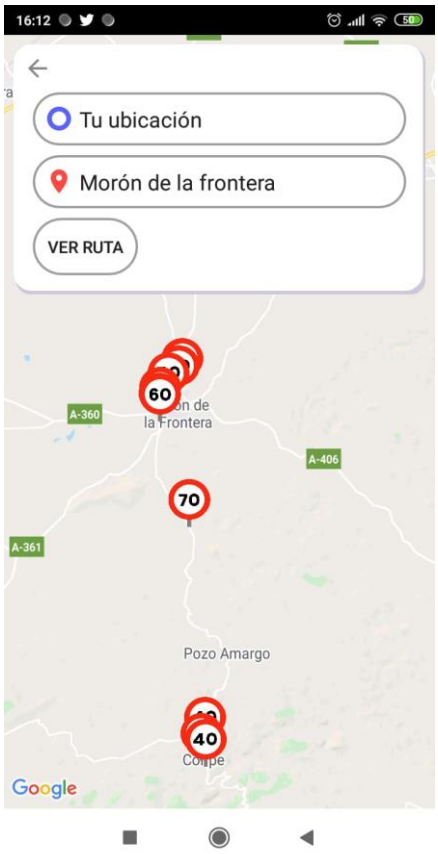

Ilustración 111: Pantalla de Navegación – Panel

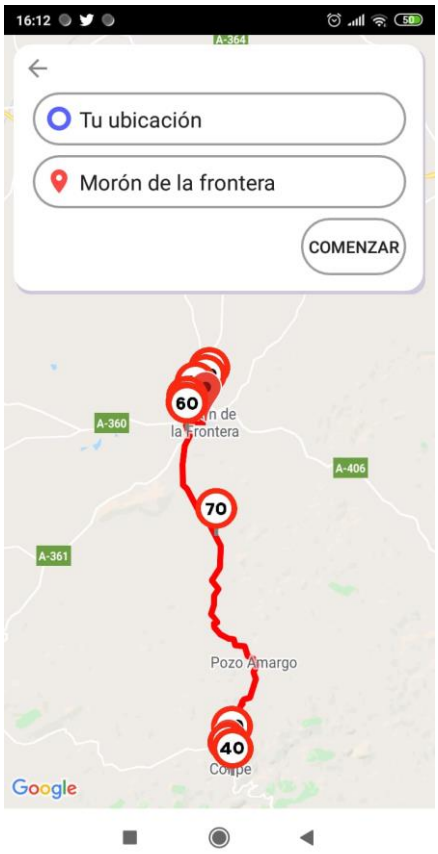

Ilustración 112: Pantalla de Navegación – Ruta

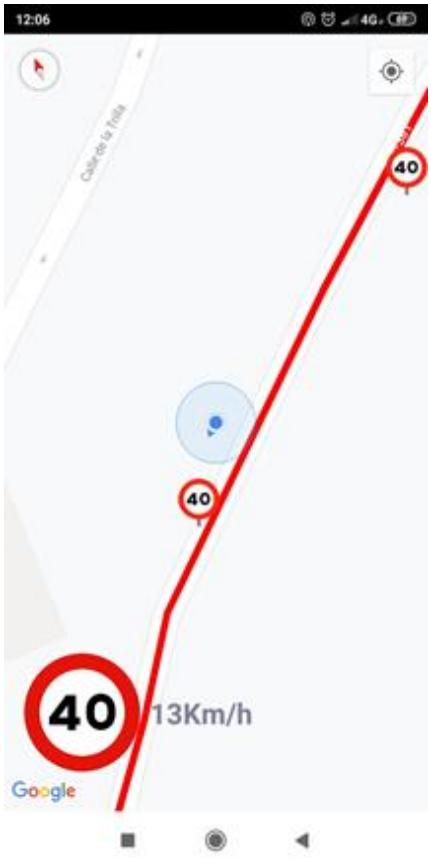

Ilustración 113: Pantalla Navegación – Aviso señal

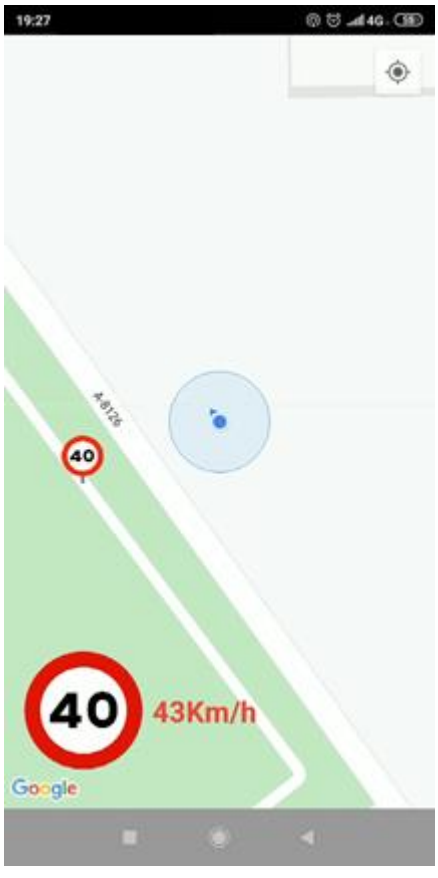

Ilustración 114: Pantalla de Navegación – Aviso velocidad

8 CONCLUSIONES Y LÍNEAS FUTURAS

Muchas cosas son improbables, solo unas pocas son imposibles.

- Elon Musk -

El Aprendizaje Profundo o *Deep Learning* ya es utilizado para dotar de autonomía a los automóviles comerciales. Vehículos de la mayoría de las empresas automovilísticas ya cuentan con sistemas de cámaras y sensores que utilizan *Deep Learning* para reconocer señales de tráfico, como, por ejemplo, Audi o Mercedes. El sistema Autopilot de Tesla también lo utiliza, ofreciendo un nivel 3 de conducción autónoma (el máximo es 5), es decir, el vehículo está equipado con radares, sensores y cámaras que le permiten reaccionar automáticamente para, por ejemplo, cambiar de carril o realizar una frenada que evite una colisión. Además, el Modelo S de Tesla ya tiene todo el equipo necesario para la conducción autónoma, por el momento se encuentra recopilando experiencias de los automóviles en las calles y aprendiendo, para que cuando se determine que ha aprendido lo suficiente se pueda activar la conducción autónoma de niveles superiores.

Estos son sólo algunos de los ejemplos, pero de cara al futuro, la mayoría de las empresas automovilísticas han anunciado que planean tener vehículos autónomos en el mercado. Los expertos creen que actualmente estamos entre los niveles 2 y 3. La automatización completa, donde el control dependerá exclusivamente del vehículo, está aún a décadas de distancia, aunque en muchos coches ya exista la opción de automatizar acciones básicas como la dirección, la aceleración, la desaceleración y el estacionamiento. Esto permite al conductor tener las manos fuera del volante, pero sólo temporalmente.

En este Trabajo de Fin de Grado se ha hecho un estudio sobre el *Deep Learning* y se ha aplicado a la idea de un vehículo más autónomo y seguro junto con otras tecnologías punteras de la actualidad como son Android, Spring Boot y los sistemas de gestión de bases de datos, así como el uso de marcos de trabajo, IDEs y diversos protocolos que hoy día todo ingeniero debe conocer. No obstante, además de todas estas funcionalidades ofrecidas en este proyecto, hay posibilidad de añadir un gran número de mejoras.

Nuestro sistema está centrado en la detección de señales de tráfico de velocidad para hacer cumplir los límites, pero la red neuronal podría ser entrenada para detectar un mayor número de señales, ya sean señales de prohibición, de obligación, de peligro u otras. Para esto sólo deberíamos entrenar YOLO con un mayor número de imágenes etiquetadas con señales de estos tipos. Además, podríamos dotarle de la capacidad de detectar también otros vehículos o peatones, pues estos son factores importantes a tener en cuenta para la conducción.

De cara al futuro también, además de ser capaz de recopilar toda esta información durante la conducción, esta podría ser integrada con otras tecnologías de forma que el vehículo fuese capaz de dar una respuesta autónoma, es decir, que sea capaz de reaccionar para frenar, acelerar, entre otras cosas.

Por último, en cuanto a mejoras técnicas de la aplicación y el servicio web, un punto muy importante es dotar de seguridad al sistema. Aunque se han desarrollado mecanismos de autenticación mediante el módulo de Spring Security, las peticiones al servidor deberían estar cifradas para así proteger la confidencialidad de los datos. De esta forma se evitarían muchos problemas, desde accesos no permitidos a la aplicación hasta adulterar los datos para conseguir fines maliciosos.

ANEXO A: INSTALACIÓN Y CONFIGURACIÓN DE ANDROID STUDIO

En este primer anexo se muestra cómo instalar y configurar Android Studio, cómo actualizarlo a versiones más recientes y cómo integrar la librería OpenCV dentro de este IDE.

A.1 Instalación de Android Studio

1. Accedemos a la página oficial de descargas de Android Studio [35] y descargamos el fichero ejecutable de la versión.
2. Lanzamos el fichero ejecutable de la Ilustración 115 y aparece el instalador de Android Studio de la Ilustración 116.

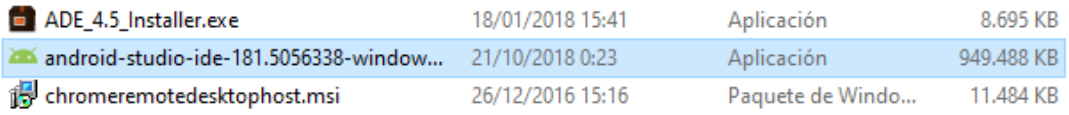

Ilustración 115: Fichero ejecutable Android Studio

Ilustración 116: Instalador Android Studio

3. Para la instalación:
    a. Seleccionamos "Android Virtual Device" como componente a instalar y pulsamos el botón "Next" (Ilustración 117).

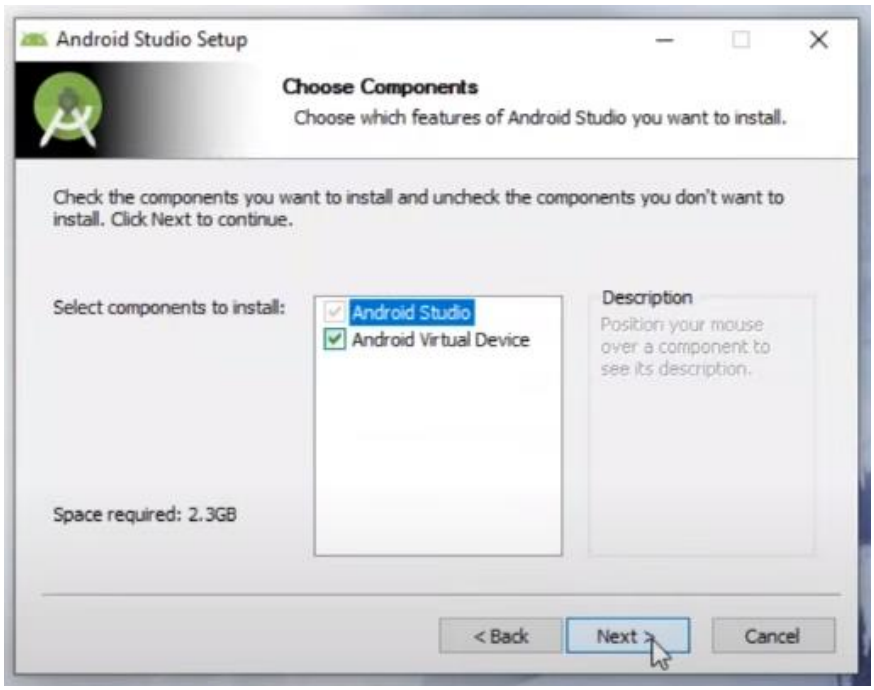

Ilustración 117: Instalación Android Studio – Elegir componentes

b. Seleccionamos el directorio donde se instalará Android Studio (Ilustración 118).

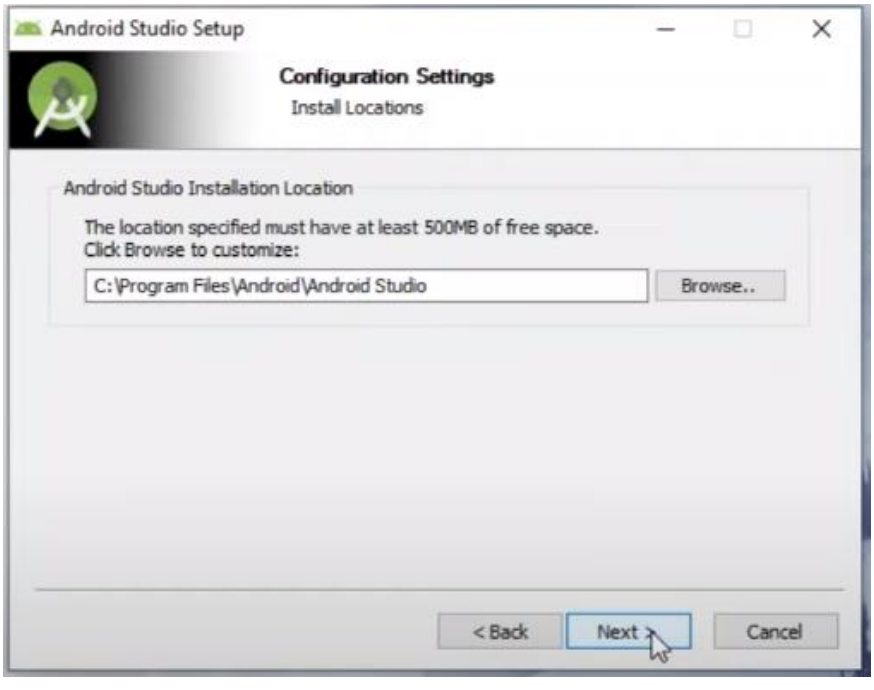

Ilustración 118: Instalación Android Studio – Localización

c. Seleccionamos la carpeta de menú de inicio y pulsamos “Install” (Ilustración 119).

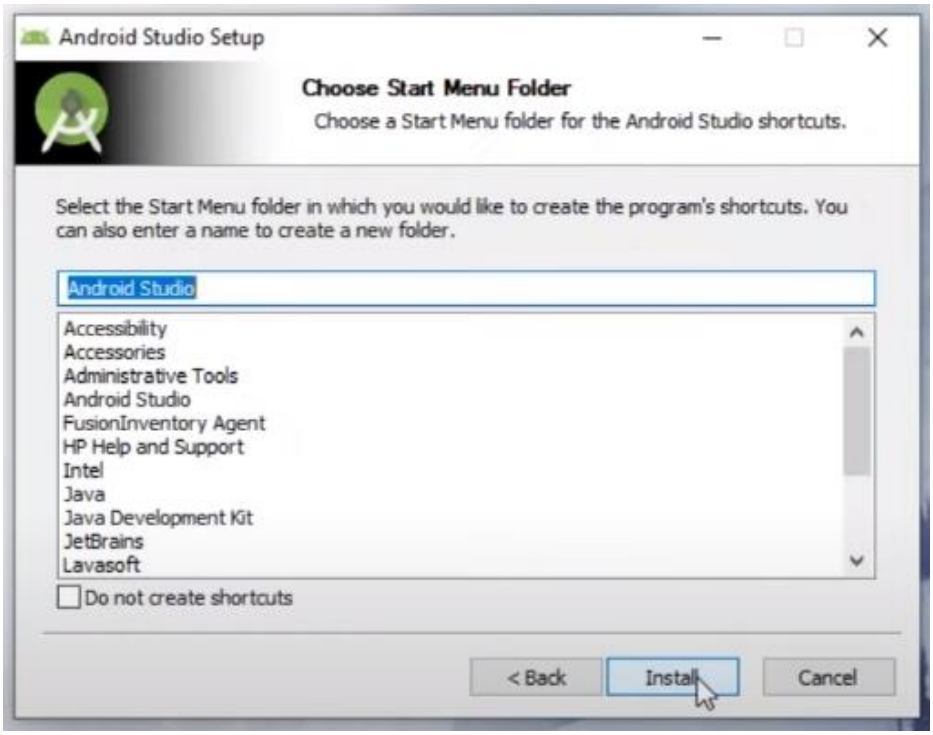

Ilustración 119: Instalación Android Studio – Elegir carpeta del Menú de Inicio

d. Esperamos el tiempo que tarde la instalación y una vez completada, nos aparecerá una pantalla como la de la Ilustración 120, dándonos la opción de arrancar Android Studio.

Ilustración 120: Instalación Android Studio completada

A.2 Configuración de Android Studio

1. Tras la correcta instalación, iniciamos Android Studio y se nos mostrará una pantalla como la de Ilustración 121 para elegir el tipo de configuración, en nuestro caso “Standard”.

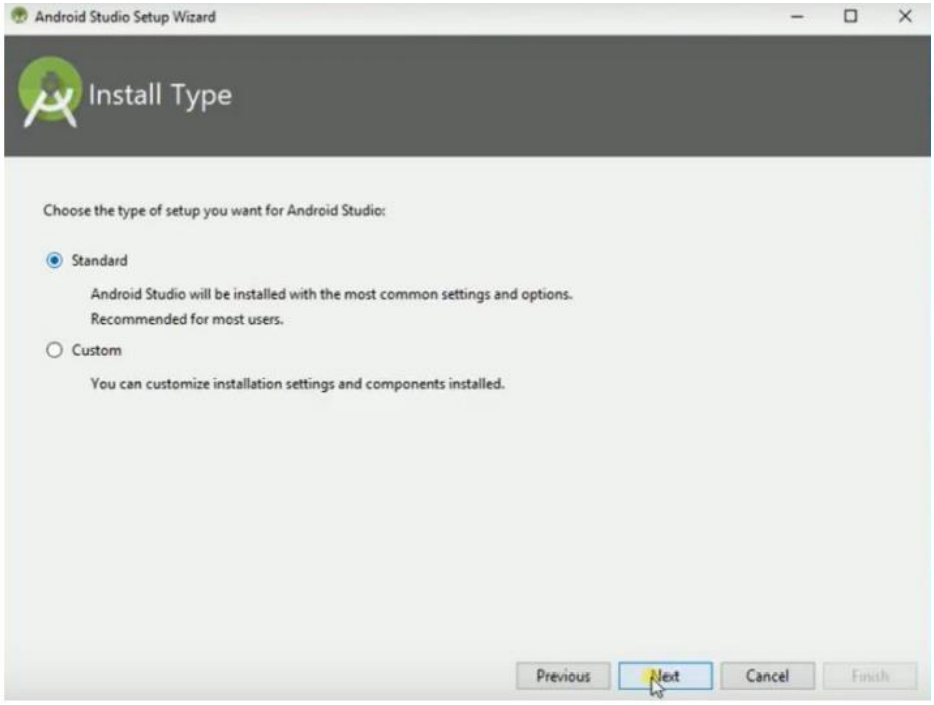

Ilustración 121: Configuración Android Studio – Selección tipo configuración

2. A continuación, seleccionamos el tema y pulsamos el botón “Next” de la Ilustración 122.

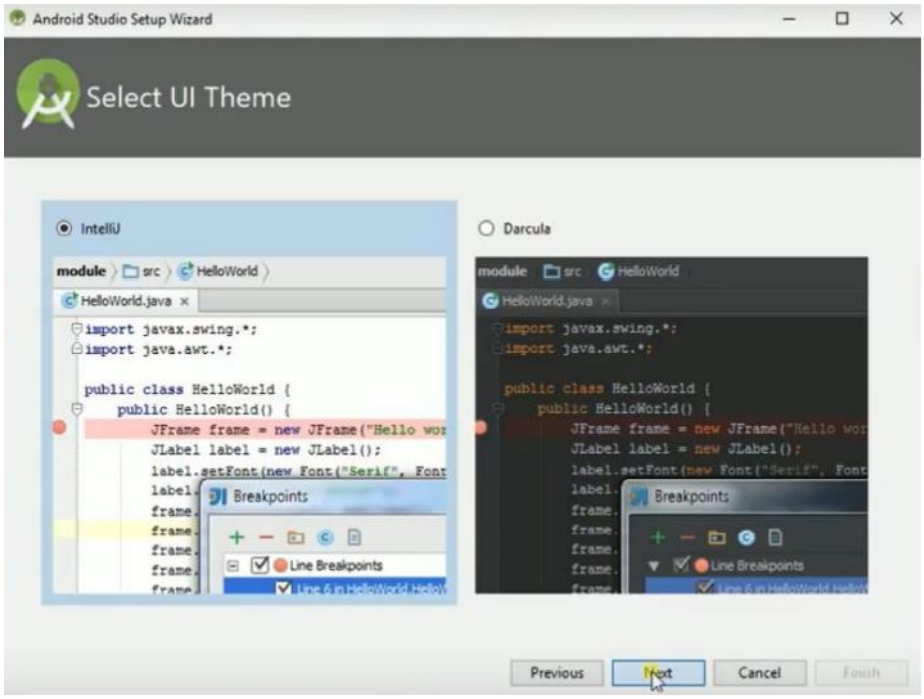

Ilustración 122: Configuración Android Studio – Selección tema

3. Por último, se cargarán todos los componentes seleccionados en la configuración y finalmente aparecerá una pantalla como la de la Ilustración 123, donde podremos comenzar un nuevo proyecto.

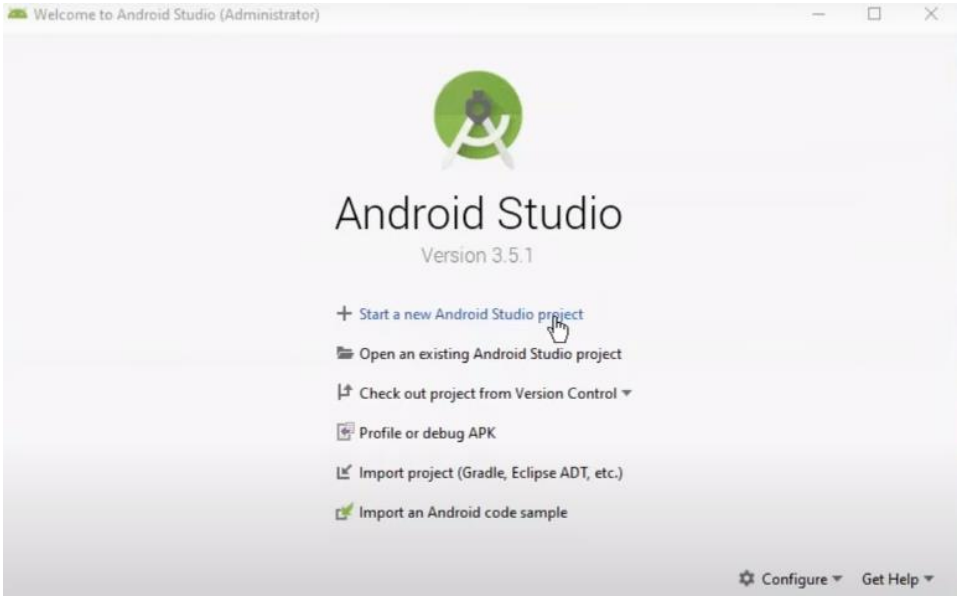

Ilustración 123: Página de Bienvenida de Android Studio

A.3 Actualización de Android Studio

En nuestro caso, ya teníamos Android Studio instalado y configurado de años anteriores, por lo que simplemente se ha procedido a actualizer este IDE a una versión más reciente. Para ello:

1. Una vez iniciado Android Studio seleccionamos *Help* → *Check for Updates* (Ilustración 124).

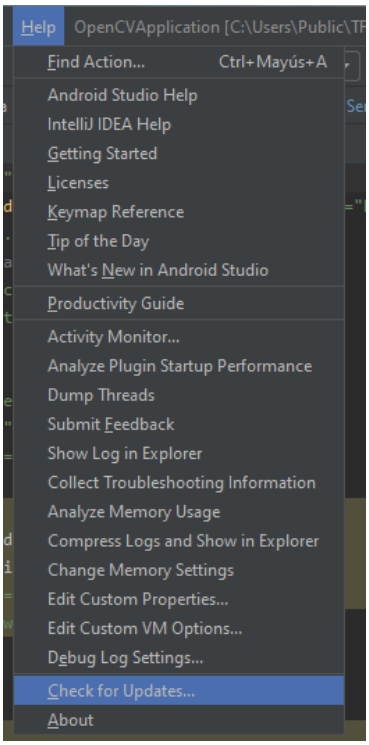

Ilustración 124: Actualización Android Studio – Buscar actualizaciones

2. A continuación, Android Studio buscará la última versión y nos aparecerá una pantalla como la de la Ilustración 125, donde se puede observar la última actualización encontrada y las mejoras que presenta. Pulsaremos en "Update and Restart".

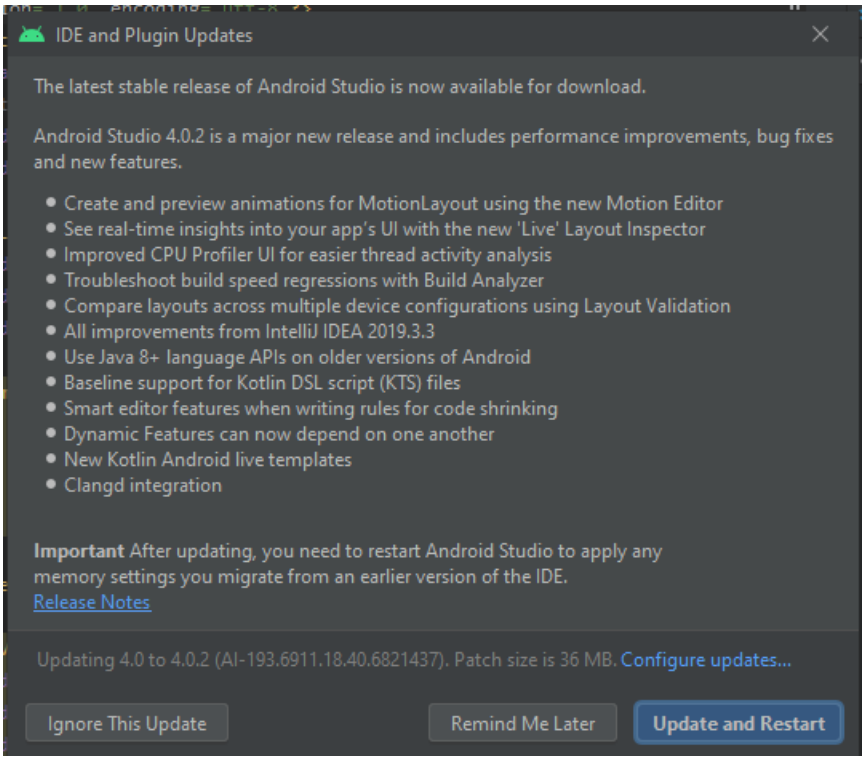

Ilustración 125: Actualización Android Studio – Actualización

3. Después de esperar el tiempo de la actualización, tendremos la última versión de Android Studio lista para usar (Ilustración 126).

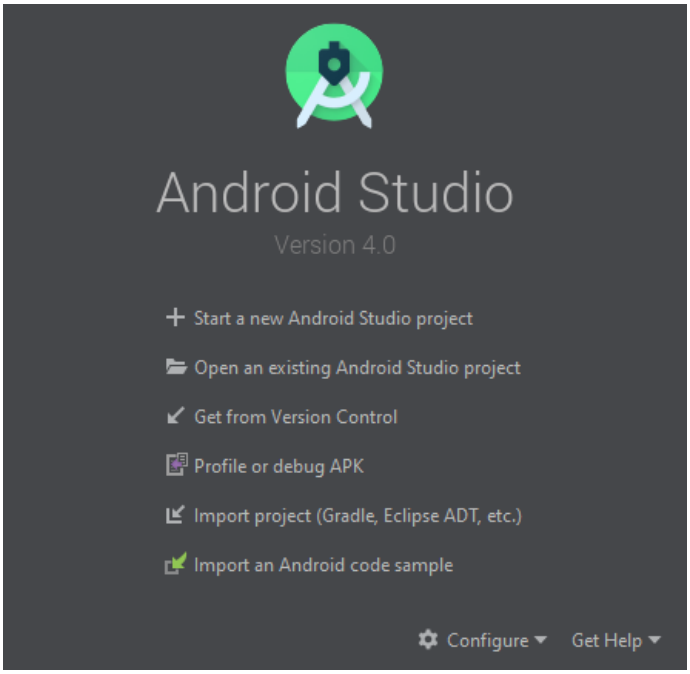

Ilustración 126: Pantalla de Bienvenida Android Studio v4.0

A.4 Integración de OpenCV en Android Studio

1. Accedemos a la página oficial de descargas de OpenCV [32] y seleccionamos la versión de OpenCV para Android que coincida con la versión de Android Studio que estamos usando, que en nuestro caso es 4.0. (Ilustración 127).

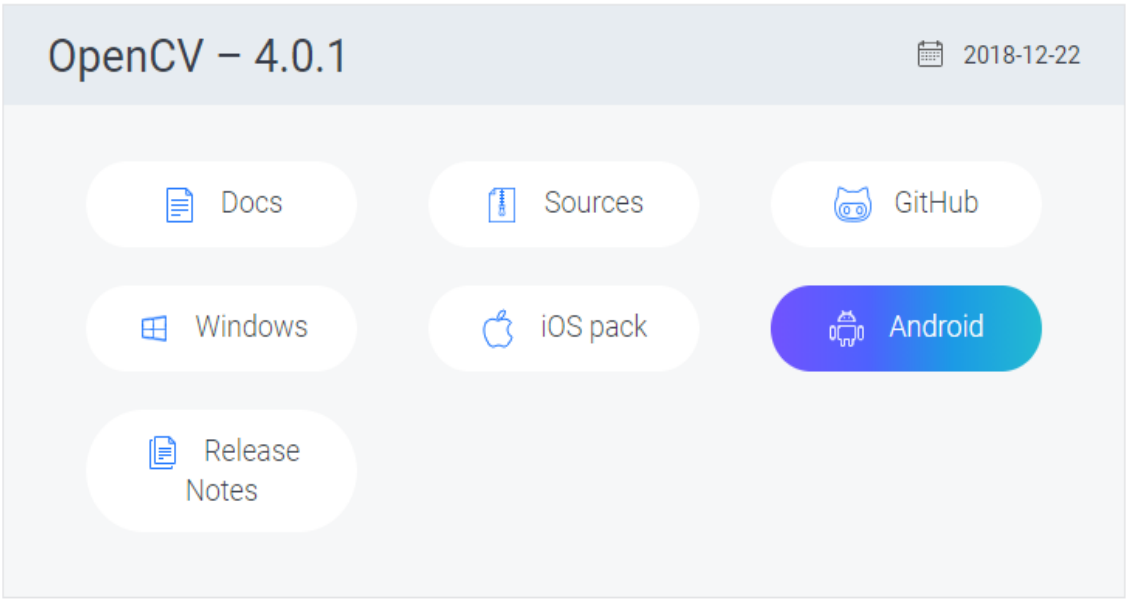

Ilustración 127: Selección Descarga OpenCV

2. Una vez descargado el archivo, lo descomprimimos (Ilustración 128).

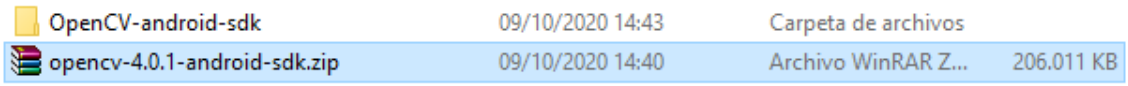

Ilustración 128: Carpeta OpenCV descomprimida

3. A continuación, abrimos nuestro proyecto en Android Studio y seleccionamos *File→New→Import Module* y seleccionamos la carpeta "java", que se encuentra dentro de la carpeta "sdk", que a su vez se encuentra dentro de la carpeta "OpenCV-android-sdk", que es la carpeta que hemos descargado y descomprimido (Ilustración 129).

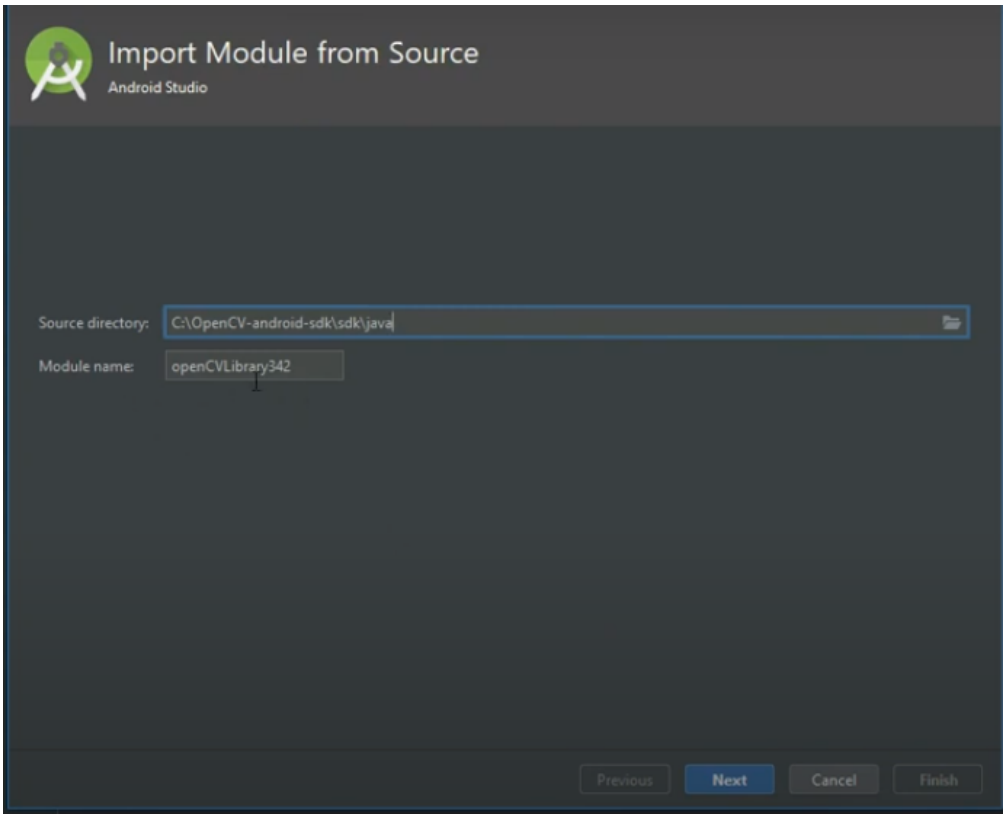

Ilustración 129: Importar módulo OpenCV en Android Studio

4. Una vez que Android Studio ha añadido el módulo y ha construido y sincronizado el proyecto sin errores, seleccionamos *File→Project Structure* y navegamos hacia el apartado de "Dependencies", donde debemos añadir OpenCV, tal como se indica en la Ilustración 130, seleccionando la opción "Module Dependency".

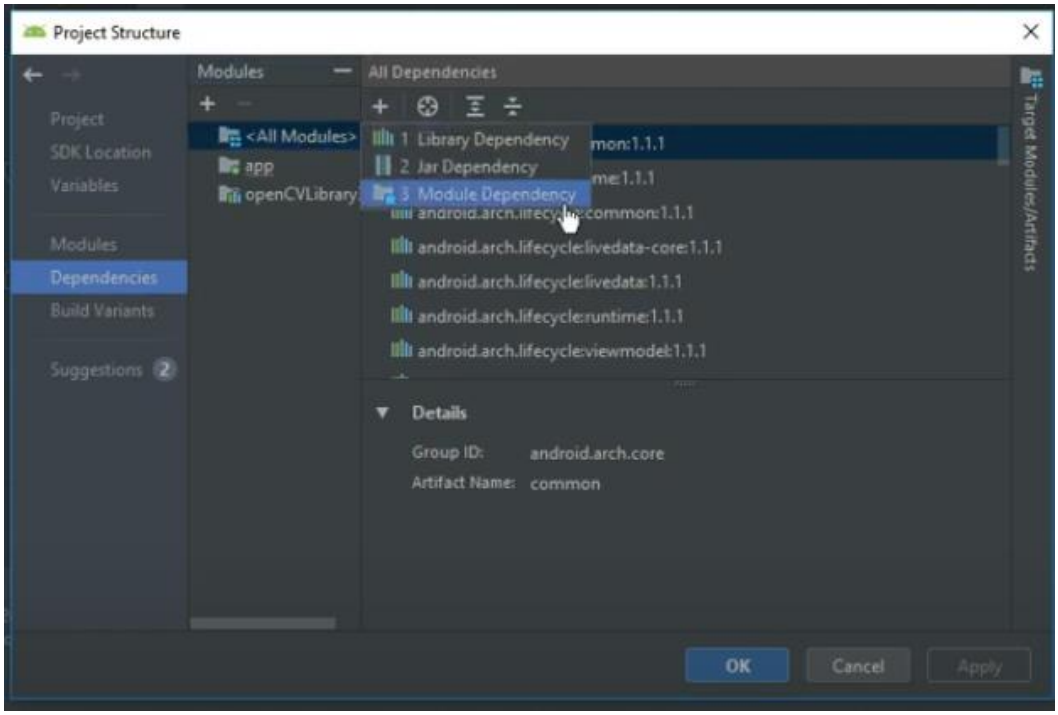

Ilustración 130: Añadir dependencia OpenCV en Android Studio

5. Una vez seleccionada esta opción, nos aparecerá una pantalla como la de la Ilustración 131, donde se debe seleccionar la librería de OpenCV. Pulsaremos entonces el botón "Apply" y a continuación el botón "Ok". Nos aparecerá a partir de ahora, en la ventana de herramientas en la izquierda, la carpeta "java", además de las otras dos que ya estaban (Ilustración 132).

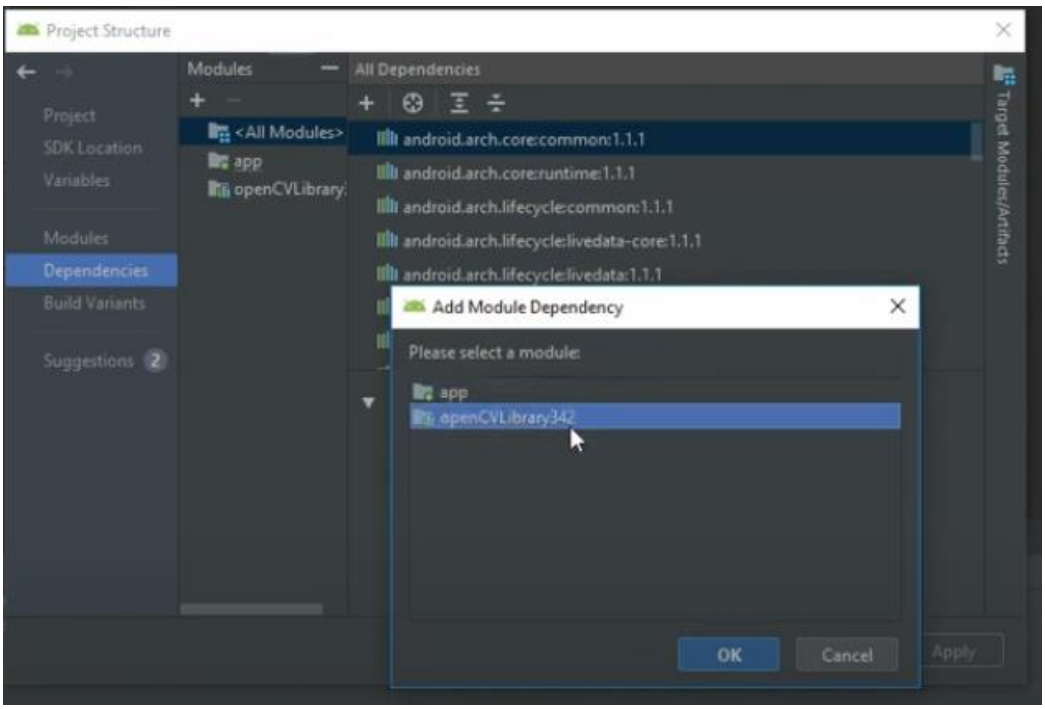

Ilustración 131: Añadir dependencia OpenCV en Android Studio (2)

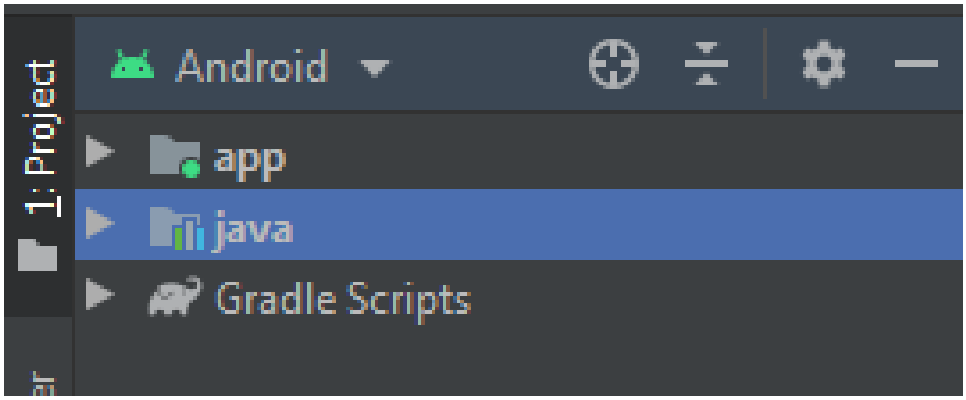

Ilustración 132: Ventana de herramientas de Android Studio

6. Una vez añadida la dependencia, en la ventana de herramientas, hacemos click derecho en la carpeta "app" y seleccionamos *New→Folder→JNI Folder* y cambiamos la localización de la nueva carpeta a crear por la que se ve en la Ilustración 133.

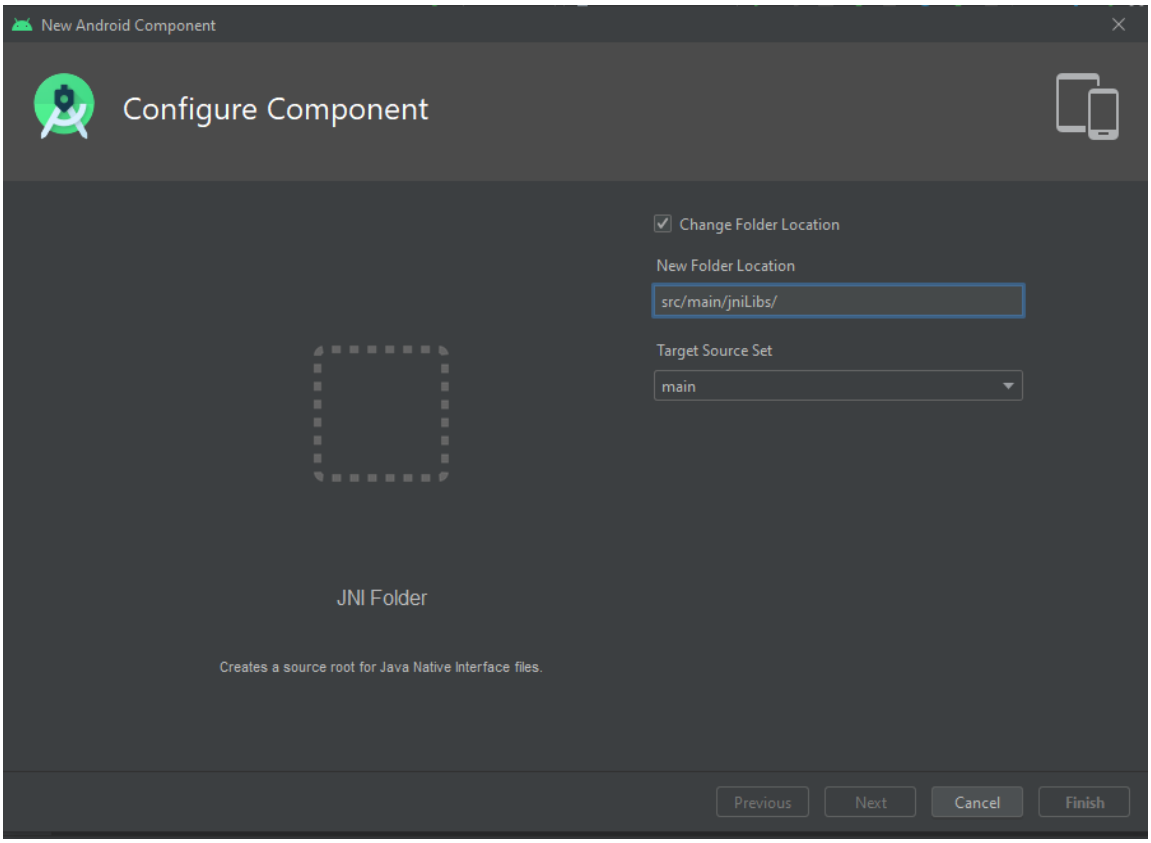

Ilustración 133: Crear nuevo *JNI Folder*

7. A continuación, copiamos dentro de esta carpeta creada los archivos que se encuentran dentro de la carpeta *sdk→native→libs* dentro de la carpeta que descargamos y descomprimimos de OpenCV en un principio (Ilustración 134). Para pegar estos archivos simplemente debemos volver a hacer click derecho en la carpeta "app" que aparece en la ventana de herramientas y seleccionar la opción "Paste", con lo que se nos mostrará la pantalla de la Ilustración 135 para seleccionar la carpeta destino.

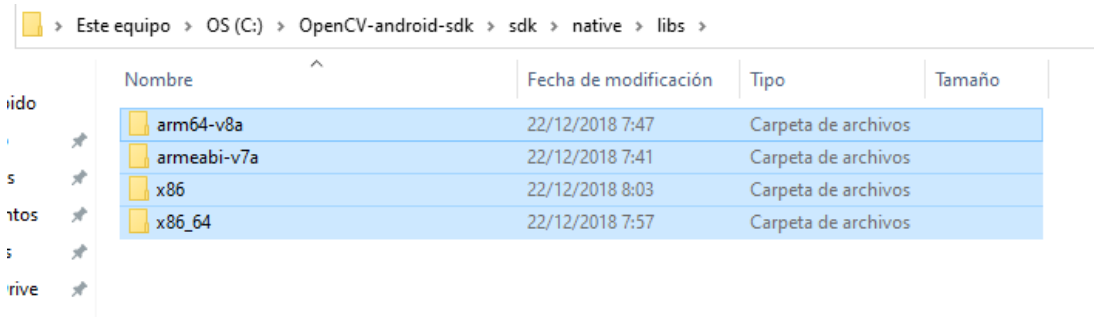

Ilustración 134: Archivos carpeta *libs*

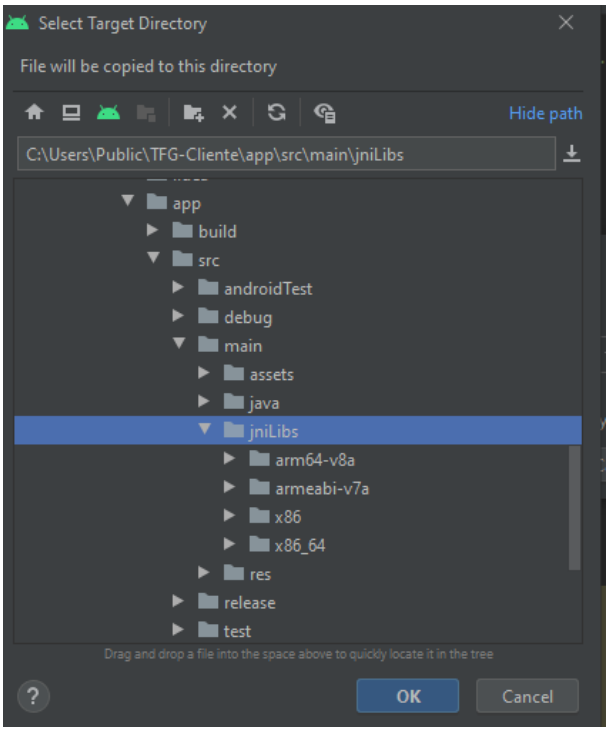

Ilustración 135: Selección carpeta destino de la opción *Paste*

8. Para comprobar que OpenCV se ha integrado correctamente con Android Studio, podemos ejecutar el código mostrado en la Ilustración 136.

```
if (OpenCVLoader.initDebug())
{
    Log.d(TAG, msg: "OpenCV is Configured or Connected Successfully.");
}
else
{
    Log.d(TAG, msg: "OpenCV not Working or Loaded.");
}
```

Ilustración 136: Comprobación Integración OpenCV con Android Studio

ANEXO B: INSTALACIÓN Y CONFIGURACIÓN DE ECLIPSE Y SPRING TOOL SUITE

En este anexo se detalla la instalación y configuración del entorno de desarrollo Eclipse y cómo añadir Spring Tool Suite a este IDE.

B.1 Instalación de Eclipse

1. Accedemos a la página oficial de descargas de Eclipse [36] y descargamos el fichero ejecutable de la versión.
2. Lanzamos el fichero ejecutable de la Ilustración 137 y aparecerá el instalador de Eclipse.

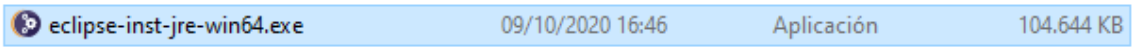

Ilustración 137: Ejecutable Eclipse

3. Seleccionamos la opción "Eclipse IDE for Enterprise Java Developers" en el instalador de Eclipse (Ilustración 138).

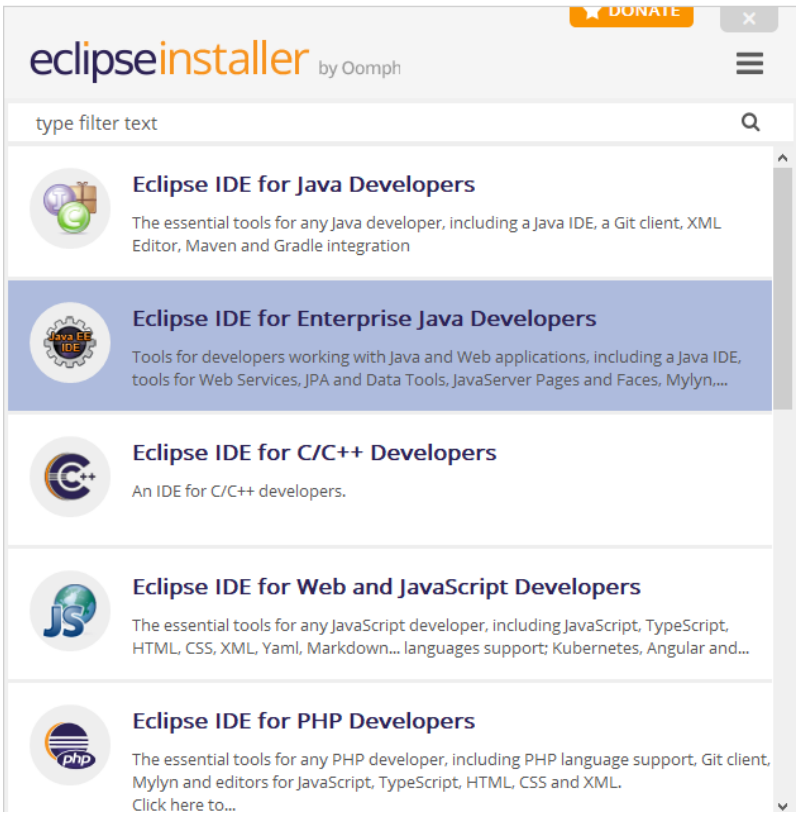

Ilustración 138: Instalador Eclipse

4. Seleccionamos el directorio donde se instalará Eclipse (Ilustración 139) y pulsamos el botón "INSTALL"

.

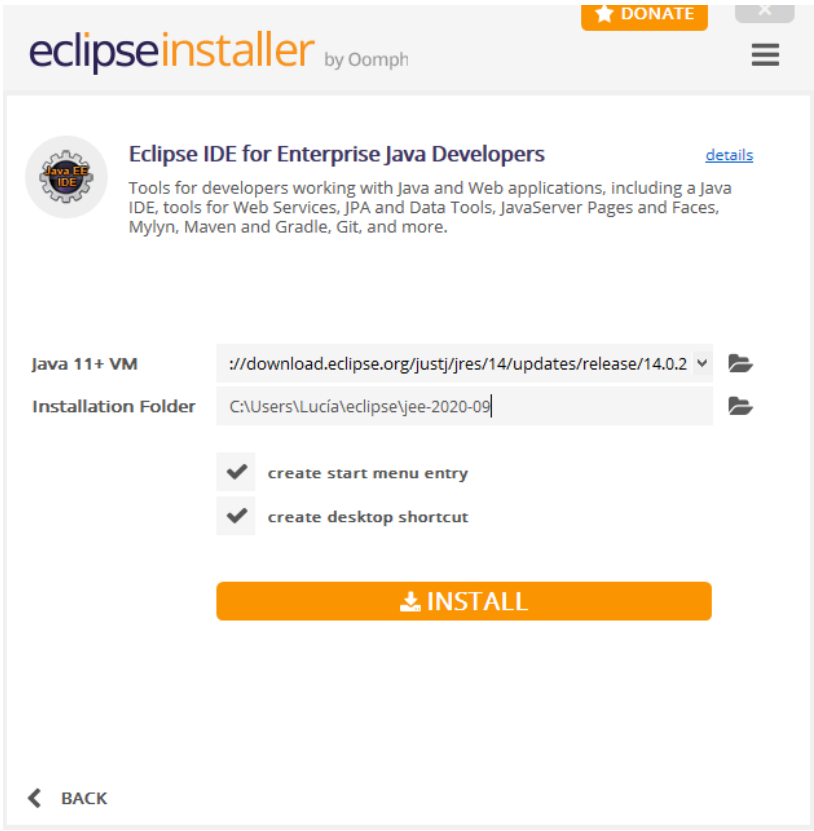

Ilustración 139: Instalación Eclipse – Localización

5. Durante el proceso de instalación nos aparecerán algunas ventanas para aceptar los términos y condiciones y las licencias de Eclipse. Una vez terminada la instalación, tendremos la opción de lanzar Eclipse (Ilustración 140).

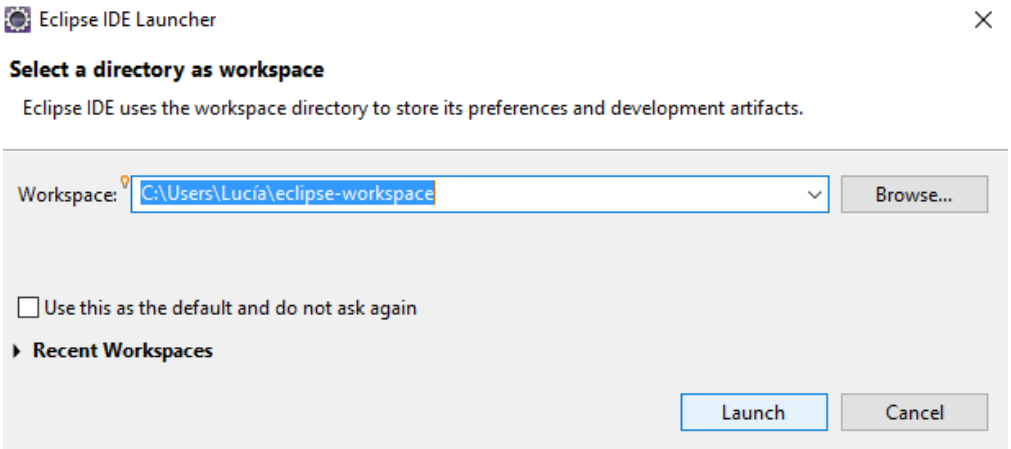

Ilustración 140: Instalación Eclipse – Eclipse IDE Launcher

B.2 Configuración de Eclipse con STS

1. Una vez lanzado Eclipse, seleccionamos *Help→Eclipse Marketplace* e instalamos "Spring Tool Suite 4" (Ilustración 141).

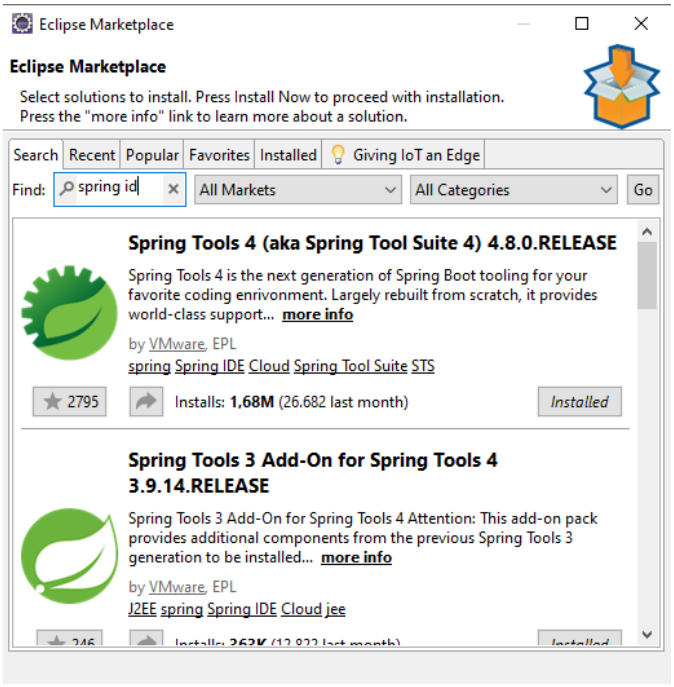

Ilustración 141: Eclipse Marketplace

2. Una vez instalado Spring Tool Suite, ya podemos crear un proyecto Spring. Para ello seleccionamos en Eclipse *File→New→Spring Starter Project* (Ilustración 142).

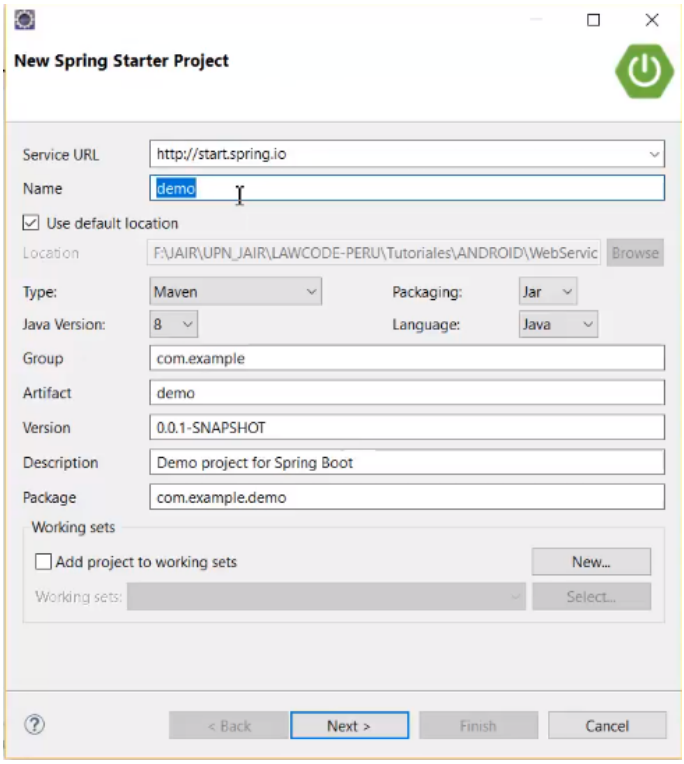

Ilustración 142: Ventana de creación Proyecto Spring

B.3 Añadir Dependencias

Para añadir distintos módulos y dependencias de Spring, simplemente debemos declararlas en el archivo *pom.xml*, que se puede ver en la Ilustración 143.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>WebService</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>WebService</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

Ilustración 143: Archivo *pom.xml*

Para nuestras necesidades concretas, hemos añadido las siguientes dependencias:

Módulo Spring Data JPA:

Spring Data es uno de los frameworks que se encuentra dentro de la plataforma de Spring cuyo objetivo es facilitar al desarrollador la persistencia de datos contra distintos repositorios de información. En concreto Spring Data JPA [37], nos facilitará la implementación de repositorios basados en JPA, simplemente tendremos que escribir las interfaces de nuestro repositorio, incluyendo los métodos de búsqueda personalizados, y Spring nos proporcionará la implementación automáticamente. De esta forma no tendremos que escribir demasiado código repetitivo para ejecutar consultas simples, así como para realizar la paginación y auditoría.

Debe ser añadido como dependencia como se observa en la Ilustración 144, dentro del archivo *pom.xml.*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Ilustración 144: Dependencia módulo Spring Data JPA

Módulo Spring Security:

Spring Security [38] es el módulo estándar de facto para proteger las aplicaciones basadas en Spring, que nos proporciona un marco de autenticación y control de acceso potente. Además, es altamente personalizable, lo cuál nos ha permitido alcanzar nuestros objetivos concretos.

Debe ser añadido como dependencia como se observa en la Ilustración 145, dentro del archivo *pom.xml.*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Ilustración 145: Dependencia módulo Spring Security

Módulo Spring Session:

Spring Session [39] proporciona una API e implementaciones para administrar la información de la sesión de un usuario. Spring Session nos ha proporcionado identificadores de sesión en las cabeceras de las llamadas REST, de esta forma, cada vez que debamos acceder a un recurso protegido del servidor, el usuario puede usar este identificador de forma transparente, en lugar de tener que autenticarse con su usuario y contraseña en cada paso.

Debe ser añadido como dependencia como se observa en la Ilustración 146, dentro del archivo *pom.xml.*

```
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-jdbc</artifactId>
</dependency>
```

Ilustración 146: Dependencia módulo Spring Session

MySQL Connector

Para enlazar el servicio web con nuestro sistema de gestión de bases de datos elegido, MySQL [40].

Debe ser añadido como dependencia como se observa en la Ilustración 147, dentro del archivo *pom.xml.*

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Ilustración 147: Dependencia MySQL Connector

Además, para establecer la conexión con la base de datos MySQL debemos añadir en el archivo *application.properties* de nuestro proyecto lo indicado en la Ilustración 148:

application.properties | WebServiceApplication.java | WebService/pom.xml | UsuarioController.java

```
1 spring.jpa.hibernate.ddl-auto=update
2 #spring.datasource.url=jdbc:mysql://localhost:3306/trafficsign_db?useUnicode=true&useJDBCCompliantTimezoneShift
3 spring.datasource.url=jdbc:mysql://icodriverdb.ckxtmwksllpr.eu-central-1.rds.amazonaws.com:3306/icodriverdb
4 spring.datasource.username=lucreilop
5 spring.datasource.password=
6
7 #server.servlet.session.timeout=60m
8
9 server.port=5000
```

Ilustración 148: Fichero *application.properties*

ANEXO C: INSTALACIÓN MYSQL WORKBECH Y CONFIGURACIÓN DE LA BASE DE DATOS

C.1 Instalación MySQL Workbench

1. Accedemos a la página oficial de descargas de MySQL [41] y descargamos el fichero ejecutable de la versión.
2. Lanzamos el fichero ejecutable de la Ilustración 149 y aparecerá el instalador de MySQL Workbench.

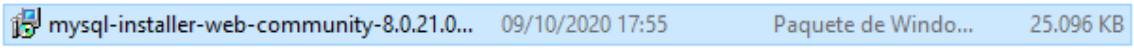

Ilustración 149: Ejecutable MySQL Workbench

3. Seleccionamos la opción "Custom" en el instalador de MySQL (Ilustración 150).

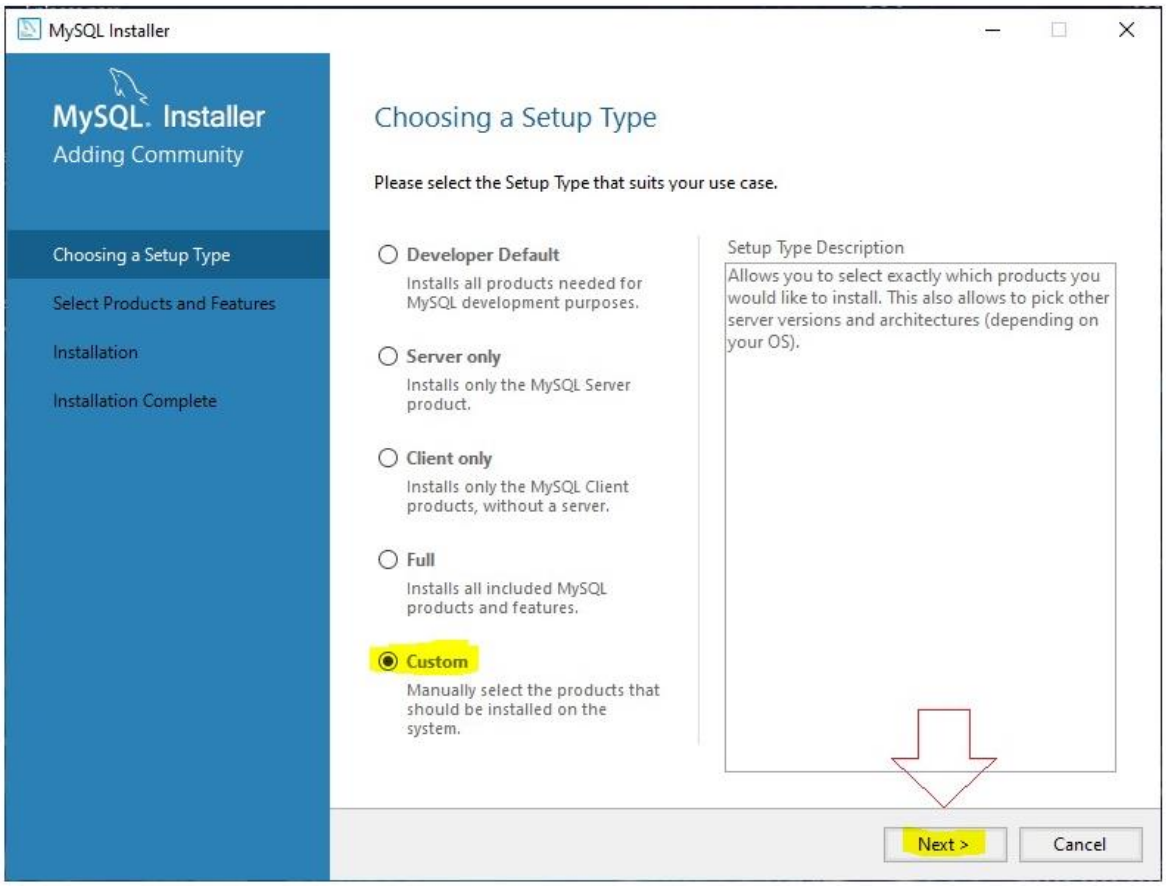

Ilustración 150: Instalador MySQL – Selección de un tipo de configuración

4. A continuación, seleccionamos los productos y características que queremos instalar. En nuestro caso, seleccionaremos MySQL Workbench y el Servidor de MySQL (Ilustración 151).

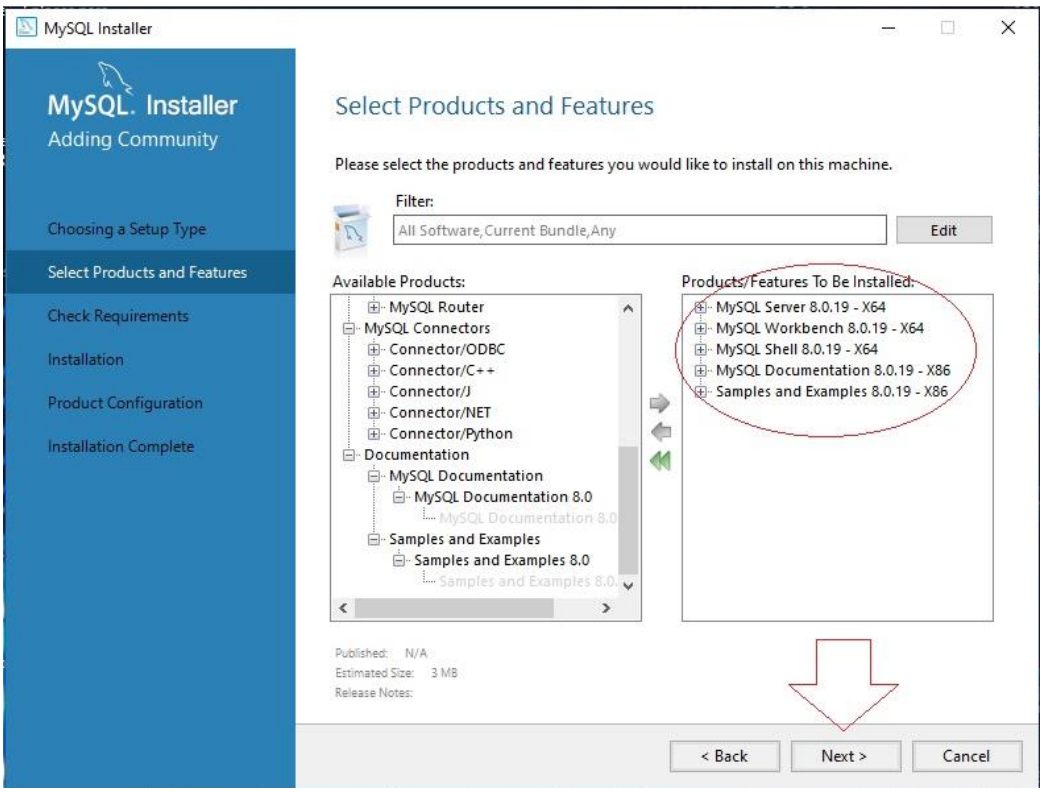

Ilustración 151: Instalador MySQL – Selección de productos y características

5. Una vez repasados los productos a instalar, llegaremos a la pantalla mostrada en la Ilustración 152, donde debemos pulsar “Execute” para comenzar con la instalación.

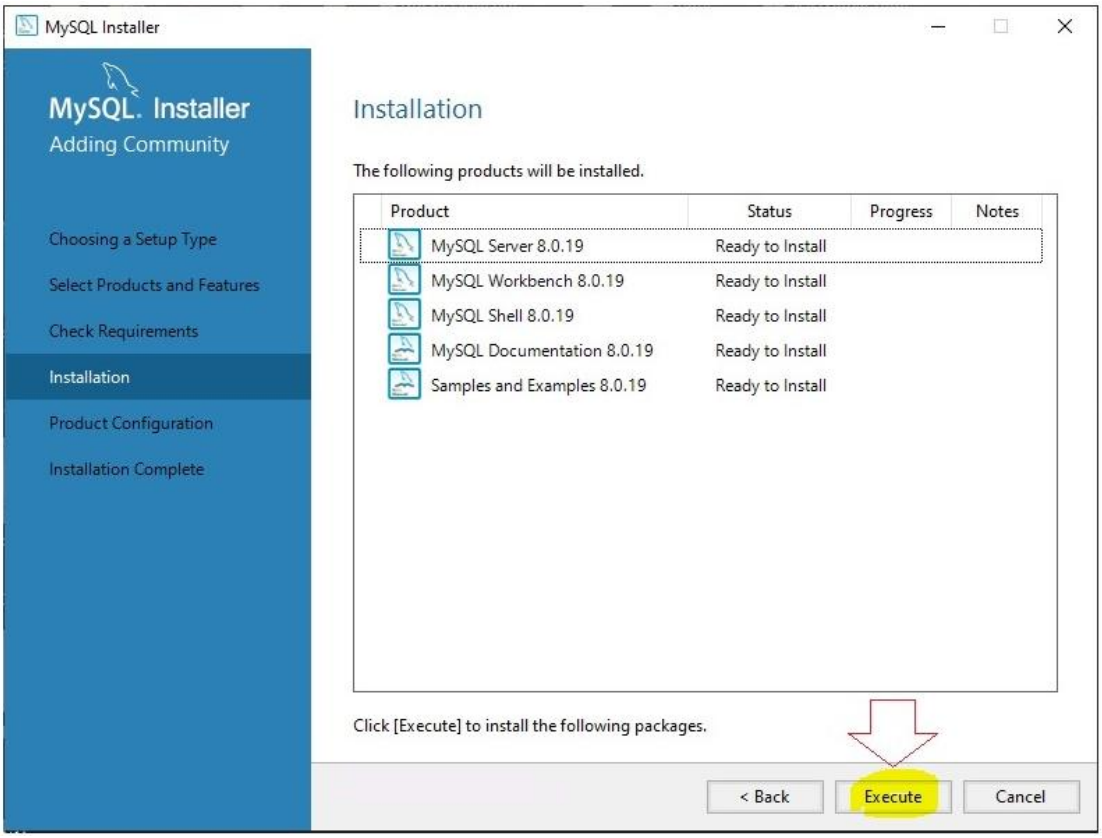

Ilustración 152: Instalador MySQL – Instalación

C.1 Configuración Base de Datos

1. Lanzamos MySQL Workbench. Cuando abrimos MySQL Workbench por primera vez después de ser instalado, se crea una instancia local (Ilustración 153).

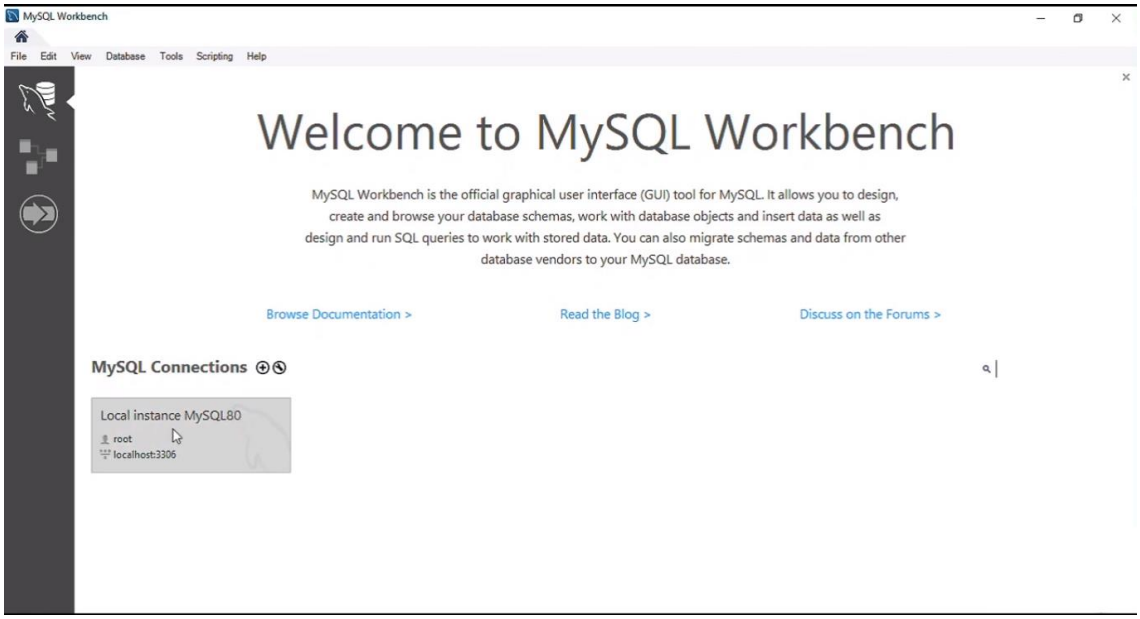

Ilustración 153: Pantalla de Bienvenida MySQL Workbench

2. Dejamos esta conexión local con sus valores por defecto (Ilustración 154).

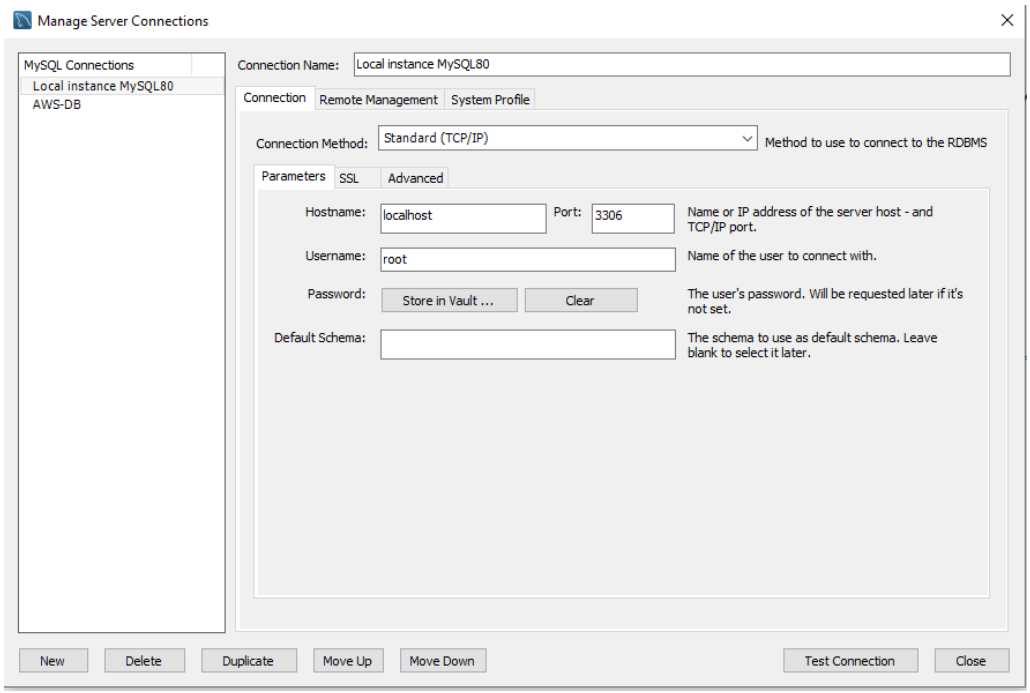

Ilustración 154: Parámetros conexión local MySQL Workbench

3. Abrimos la conexión local y creamos la base de datos a partir del fichero creatablas.sql, cuyo contenido se muestra en la Ilustración 155 y 156.

```
CREATE SCHEMA `trafficsign_db` DEFAULT CHARACTER SET utf8 ;

USE trafficsign_db;

CREATE TABLE `trafficsign_db`.`usuarios` (
  `username` VARCHAR(45) NOT NULL,
  `nombre` VARCHAR(45) NOT NULL,
  `apellidos` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `contraseña` VARCHAR(255) NOT NULL,
  `ciudad` VARCHAR(45) NOT NULL,
  `país` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

CREATE TABLE `trafficsign_db`.`usuario_señales` (
  `id` INTEGER NOT NULL auto_increment,
  `longitud` FLOAT NOT NULL,
  `latitud` FLOAT NOT NULL,
  `ancho` FLOAT NOT NULL,
  `alto` FLOAT NOT NULL,
  `clase` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `username_idx` (`username` ASC) VISIBLE,
  CONSTRAINT `username`
    FOREIGN KEY (`username`)
    REFERENCES `trafficsign_db`.`usuarios` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Ilustración 155: Fichero *creatablas.sql*

```
CREATE TABLE `trafficsign_db`.`global_señales` (
  `id` INTEGER NOT NULL auto_increment,
  `longitud` FLOAT NOT NULL,
  `latitud` FLOAT NOT NULL,
  `ancho` FLOAT NOT NULL,
  `alto` FLOAT NOT NULL,
  `clase` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `username_idx` (`username` ASC) VISIBLE,
  CONSTRAINT `username`
    FOREIGN KEY (`username`)
    REFERENCES `trafficsign_db`.`usuarios` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

CREATE TABLE SPRING_SESSION (
    PRIMARY_ID CHAR(36) NOT NULL,
    SESSION_ID CHAR(36) NOT NULL,
    CREATION_TIME BIGINT NOT NULL,
    LAST_ACCESS_TIME BIGINT NOT NULL,
    MAX_INACTIVE_INTERVAL INT NOT NULL,
    EXPIRY_TIME BIGINT NOT NULL,
    PRINCIPAL_NAME VARCHAR(100),
    CONSTRAINT SPRING_SESSION_PK PRIMARY KEY (PRIMARY_ID)
) ENGINE=InnoDB ROW_FORMAT=DYNAMIC;

CREATE UNIQUE INDEX SPRING_SESSION_IX1 ON SPRING_SESSION (SESSION_ID);
CREATE INDEX SPRING_SESSION_IX2 ON SPRING_SESSION (EXPIRY_TIME);
CREATE INDEX SPRING_SESSION_IX3 ON SPRING_SESSION (PRINCIPAL_NAME);

CREATE TABLE SPRING_SESSION_ATTRIBUTES (
    SESSION_PRIMARY_ID CHAR(36) NOT NULL,
    ATTRIBUTE_NAME VARCHAR(200) NOT NULL,
    ATTRIBUTE_BYTES BLOB NOT NULL,
    CONSTRAINT SPRING_SESSION_ATTRIBUTES_PK PRIMARY KEY (SESSION_PRIMARY_ID, ATTRIBUTE_NAME),
    CONSTRAINT SPRING_SESSION_ATTRIBUTES_FK FOREIGN KEY (SESSION_PRIMARY_ID) REFERENCES SPRING_SESSION(PRIMARY_ID) ON DELETE CASCADE
) ENGINE=InnoDB ROW_FORMAT=DYNAMIC;
```

Ilustración 156: Fichero *creatablas.sql* (Continuación)

4. A partir de aquí, podemos ver la base de datos creada, así como sus tablas y columnas, además podremos modificarla y gestionarla (Ilustración 157).

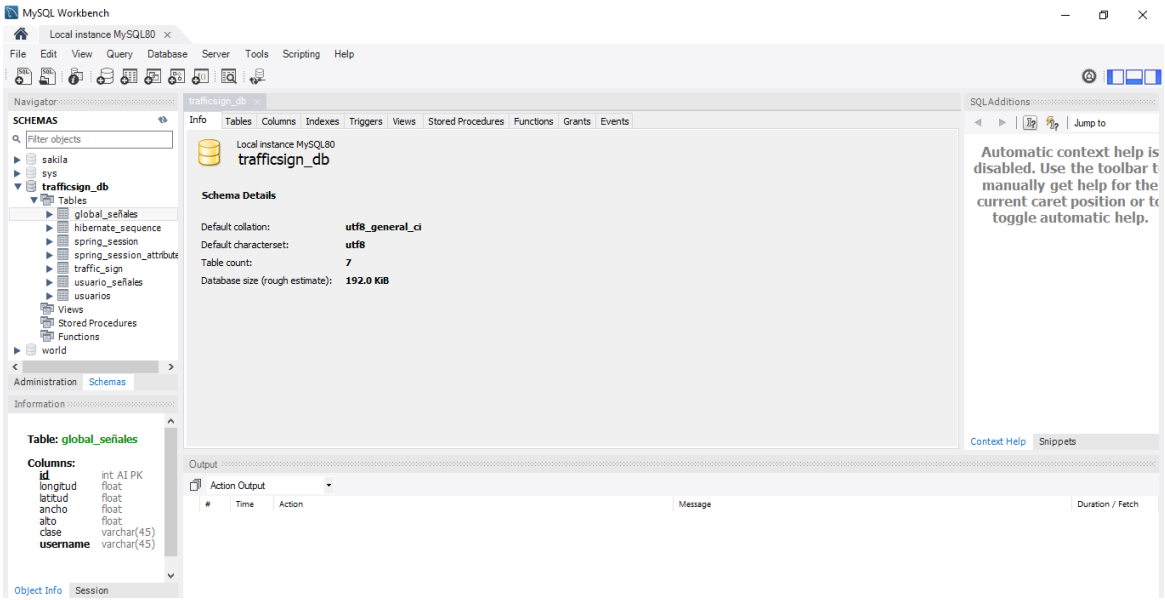

Ilustración 157: MySQL Workbench – Interfaz gráfica

ANEXO D: DESPLIEGUE SERVICIO WEB Y BASE DE DATOS EN AWS

En este Anexo detallamos el despliegue del servicio web Spring desarrollado y la base de datos MySQL en la nube de Amazon Web Services.

D.1 Base de Datos MySQL en AWS.

1. Accedemos a la consola de AWS [42] y en el buscador de servicios introducimos "RDS" (Ilustración 158).

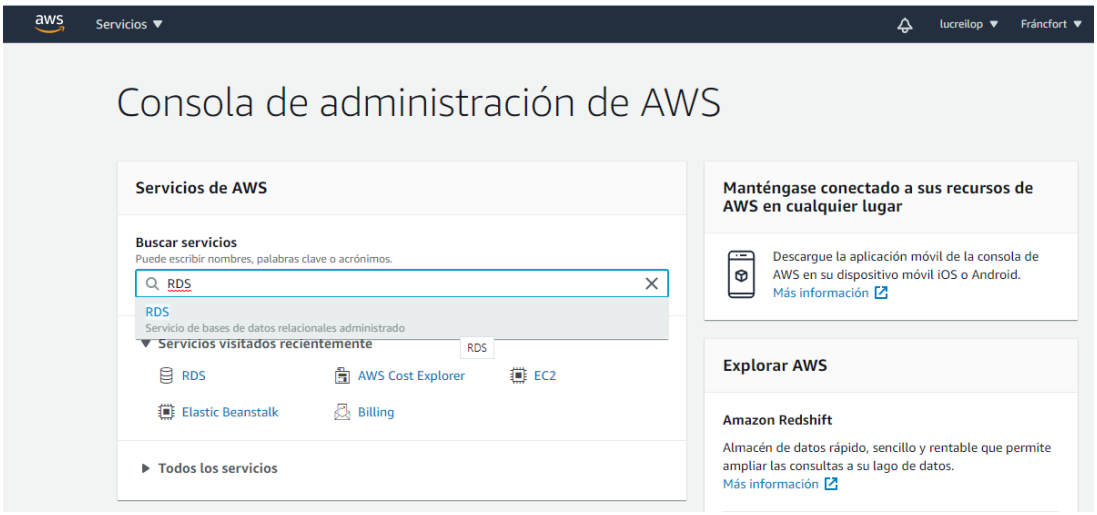

Ilustración 158: Consola AWS – Amazon RDS

2. Una vez accedemos al servicio Amazon RDS (Servicio de bases de datos relacionales administrado), pulsamos la opción "Databases" en el panel de la izquierda de la Ilustración 159. Como vemos, no hay ninguna base de datos creada todavía, para ello pulsamos "Create database".

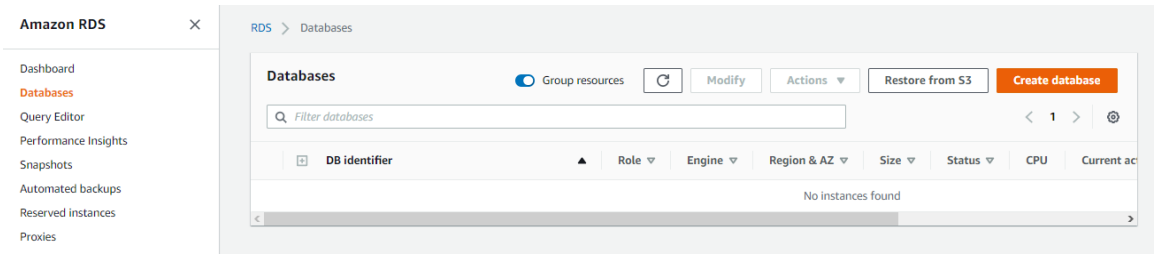

Ilustración 159: Amazon RDS – Databases

3. A continuación, seleccionamos las distintas opciones que nos aparecerán para la configuración de nuestra base de datos. En la Ilustración 160, se observa como seleccionamos MySQL, en la Ilustración 161, seleccionamos la opción "Free tier" para poder usar RDS de manera gratuita. Y por último, en la Ilustración 162, se elije un nombre para la base de datos, un usuario y una contraseña.

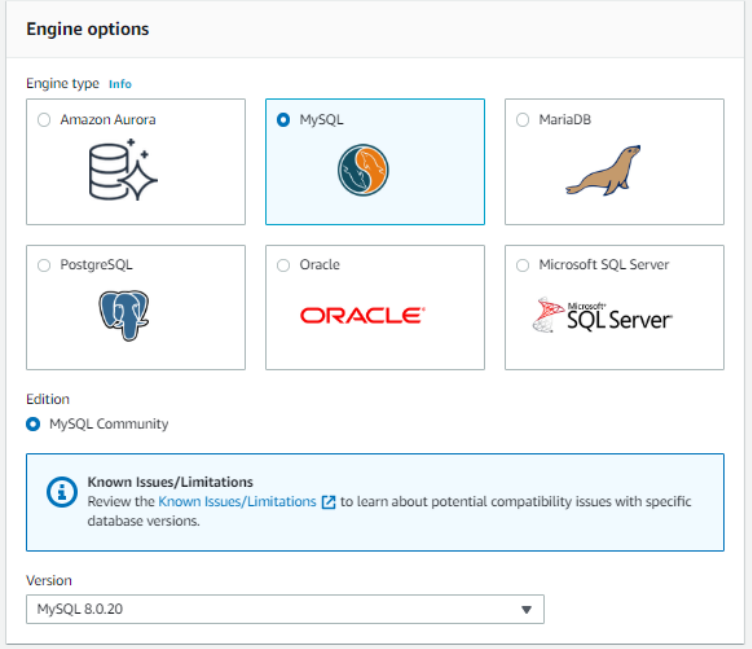

Ilustración 160: Amazon RDS – Engine options

Ilustración 161: Amazon RDS – Templates

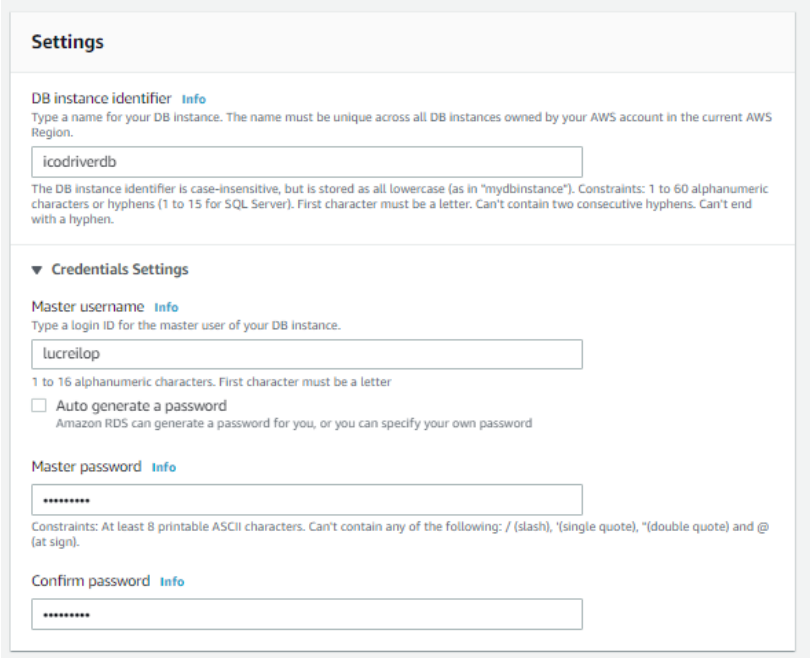

Ilustración 162: Amazon RDS – Configuración

4. En el apartado "Connectivity" desplegamos la opción "Additional connectivity configuration" y seleccionamos "Yes" en el apartado "Public Access". El resto se deja como está (Ilustración 163).

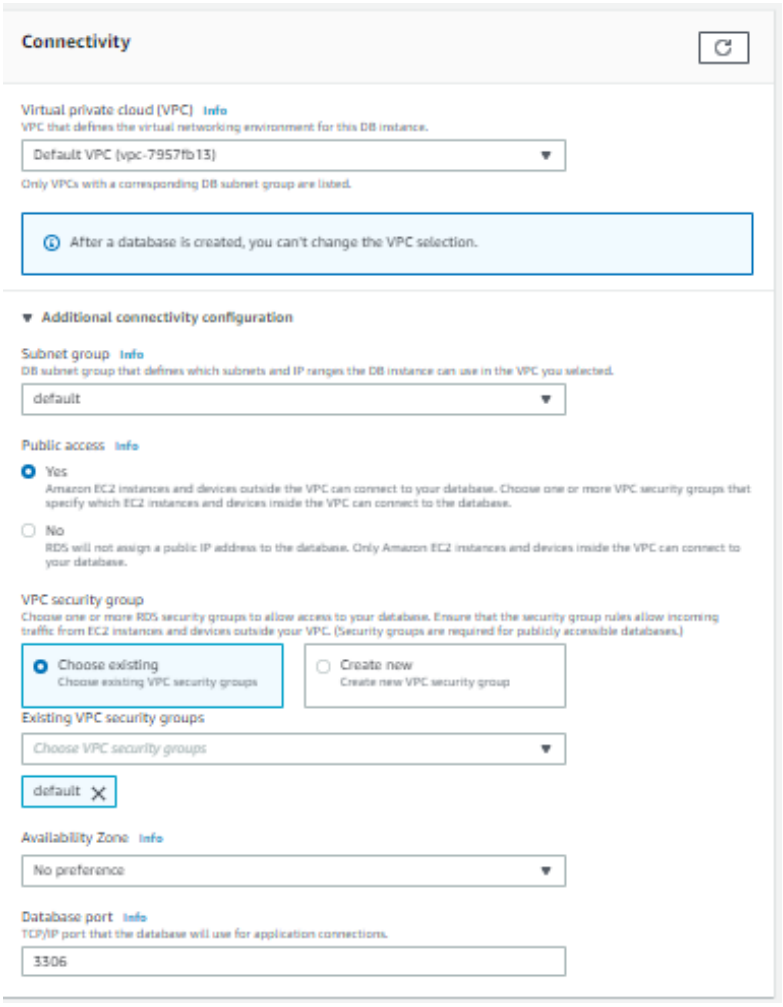

Ilustración 163: Amazon RDS – Conectividad

5. Para terminar con la creación, seleccionamos el desplegable "Additional configuration" y volvemos a escribir el nombre de la base de datos en "Initial database name" (Ilustración 164). Este nombre se debe recordar posteriormente. El resto de campos los dejamos como están y pulsamos finalmente el botón "Create database" al final de la página.

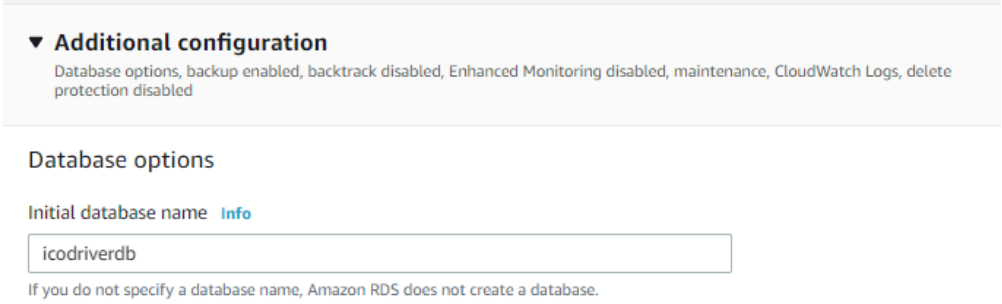

Ilustración 164: Amazon RDS – Configuración adicional

6. La creación de la base de datos tardará unos minutos (Ilustración 165). Una vez este creada, el campo "Status" aparecerá como "Available" en verde.

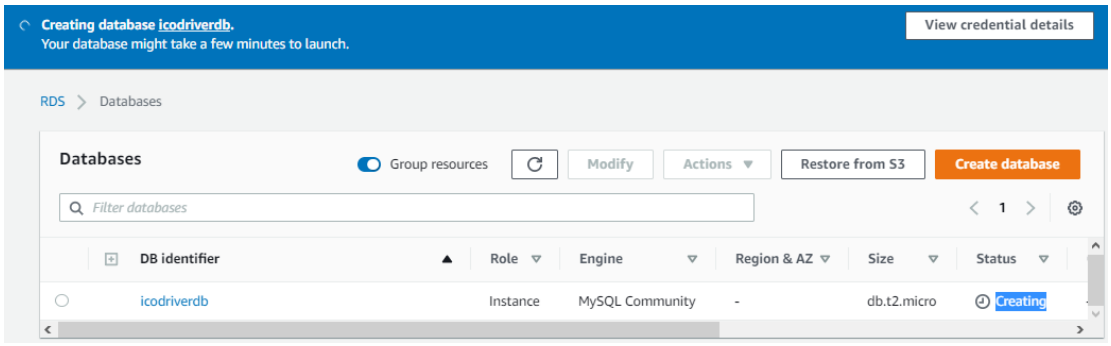

Ilustración 165: Amazon RDS – Creando base de datos

7. Una vez creada, seleccionamos el "DB identifier" para ver los detalles y en el apartado "Connectivity & port" copiamos el "Endpoint" y el "Port" (Ilustración 166).

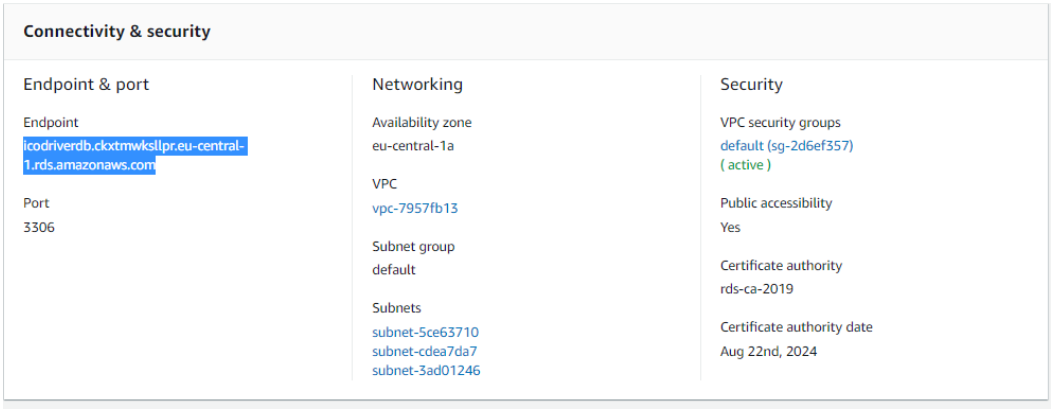

Ilustración 166: Amazon RDS – Conectividad y Seguridad

8. Modificamos el archive *application.properties* de nuestro servicio web en Eclipse, añadiendo el "Endpoint", el "Port" y el nombre de la base de datos que introdujimos en el campo "Initial database name" como se muestra en la Ilustración 167, así como el usuario y contraseña que introdujimos también anteriormente. También establecemos como puerto del servidor el 5000, para que no haya problemas con la pasarela.

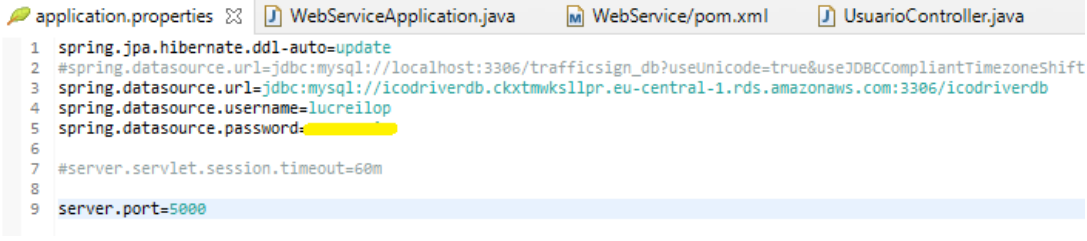

Ilustración 167: Archivo *application.properties*

9. En el apartado "Security" de la Ilustración 166, si seleccionamos la opción que se encuentra debajo de "VPC security groups" podremos editar las reglas de entrada y así restringir por IP quién puede acceder a la base de datos. En nuestro caso, dejaremos las reglas tal como se indica en la Ilustración 168, para que se pueda acceder desde cualquier IP (siempre que se tenga el usuario y la contraseña).

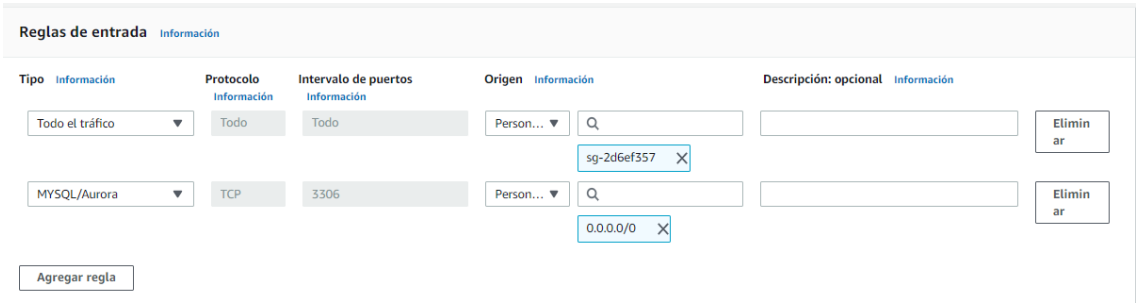

Ilustración 168: Amazon RDS – Editar reglas de entrada

10. A continuación, en MySQL Workbench, añadiremos una conexión con esta instancia para poder administrar la base de datos más cómodamente. Para ello lanzamos MySQL Workbench, pulsamos la opción "+" al lado de "MySQL Connections" y asignaremos un nombre a la conexión, además de configurar la conexión con el "Endpoint", "Port", nombre de usuario y contraseña tal como se muestra en la Ilustración 169.

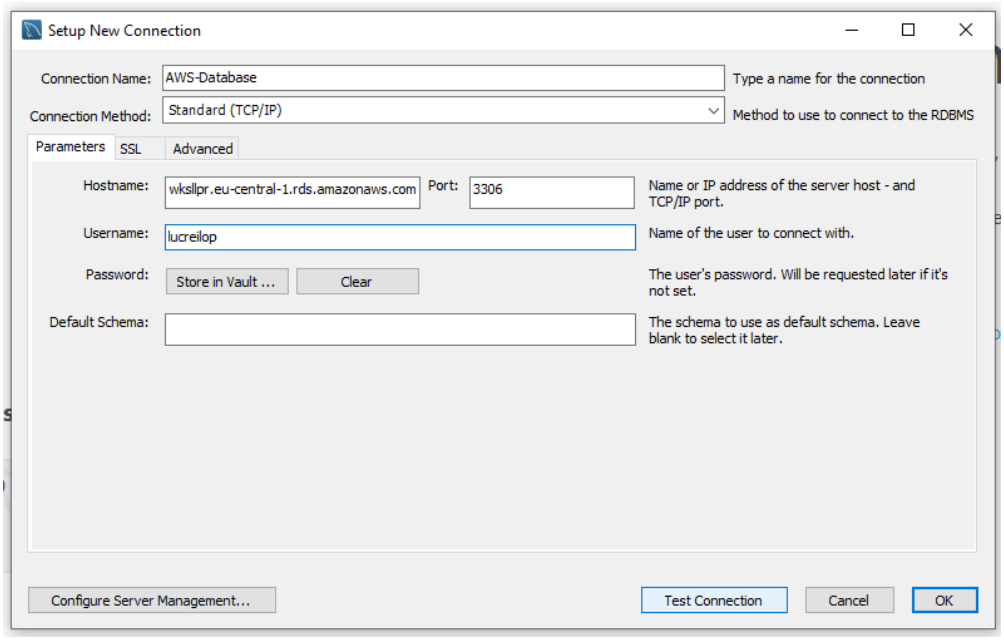

Ilustración 169: MySQL Workbench – Añadir nueva conexión

11. Creamos manualmente las tablas "spring_session" y "spring_session_attributes", ya que al no tener una clase entidad asociada, no se crearán automáticamente cuando conectemos el servicio web desplegado en AWS a este base de datos. Para ello, en MySQL Workbench y en la conexión que acabamos de añadir, ejecutamos el código que se muestra en la Ilustración 170.

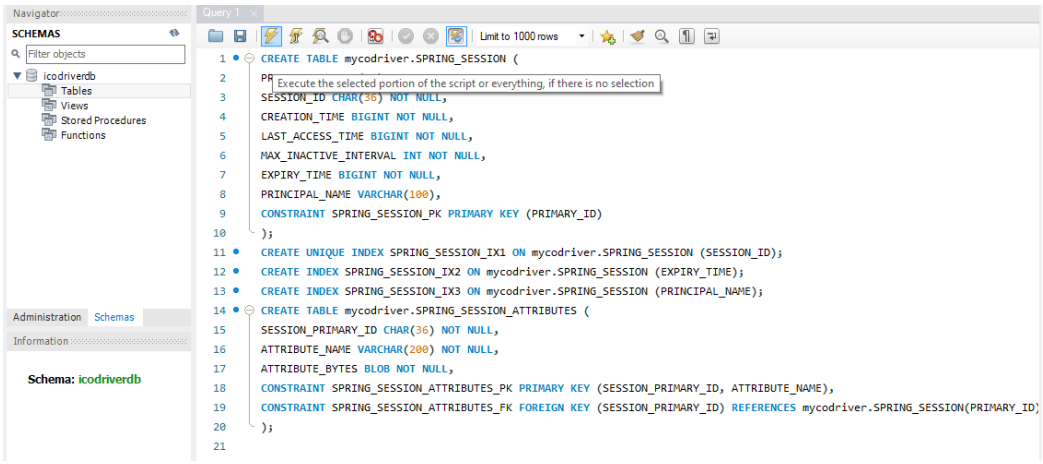

Ilustración 170: MySQL Workbench – Creación manual tablas Spring Session

D.1 Despliegue Servicio Web Spring en AWS

1. Accedemos a la Consola de Administración de AWS [42], para ello, debemos iniciar sesión con nuestra cuenta o registrarnos si no poseemos una.
2. Dentro de la consola, en el apartado "Buscar Servicios", navegaremos hacia *Elastic Beanstalk* y pulsaremos en el botón "Create Application" (Ilustración 171).

Ilustración 171: AWS – Elastic Beanstalk

3. Primero, seleccionaremos un nombre (Ilustración 172).

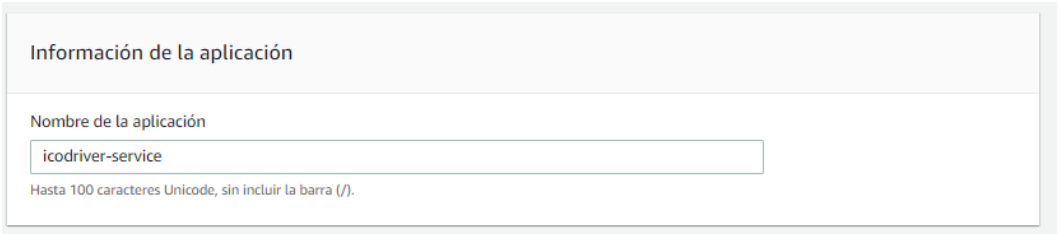

Ilustración 172: Elastic Beanstalk – Seleccionar nombre aplicación

4. En el apartado “Pataforma”, seleccionamos Java tal y como se muestra en la Ilustración 173.

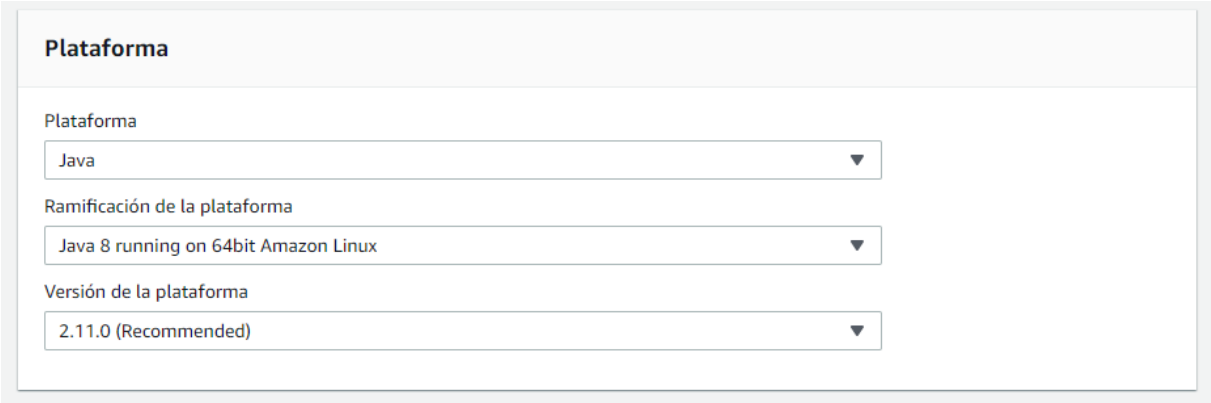

Ilustración 173: Elastic Beanstalk – Seleccionar plataforma

5. Pulsamos la opción “Configurar más opciones” que aparece al final de la página y seleccionamos “Base de datos” (Ilustración 174), rellenamos los campos para configurar nuestra base de datos MySQL (Ilustración 175) y pulsamos el botón “Guardar”. Hay que tener en cuenta que el usuario y contraseña que seleccionemos para la base de datos deben coincidir con los introducidos en el apartado anterior.

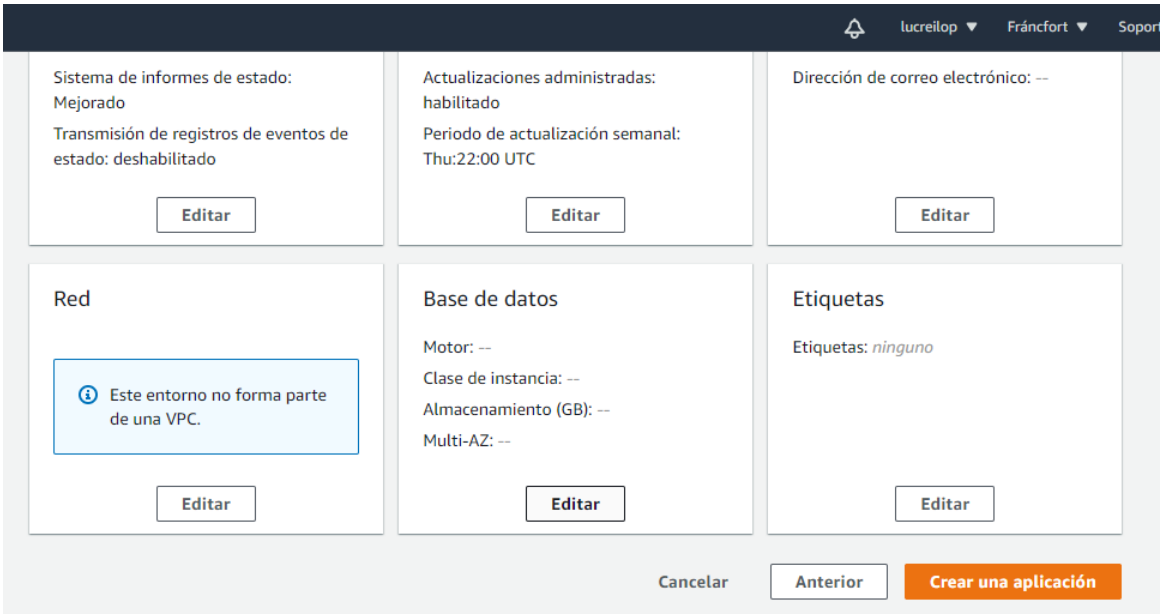

Ilustración 174: Elastic Beanstalk – Configurar más opciones

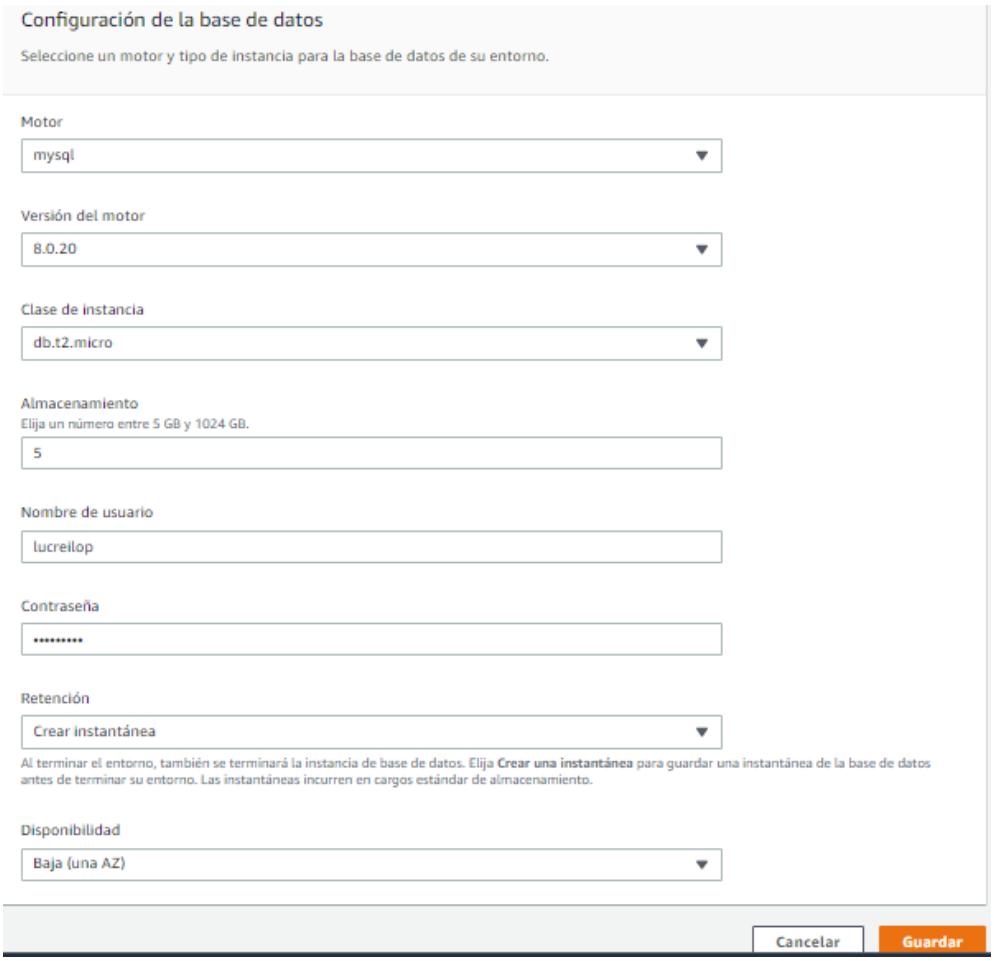

Ilustración 175: Elastic Beanstalk – Configuración de la base de datos

6. A continuación, pulsamos en la opción "Crear una aplicación" y comenzará la creación de la aplicación web, lo que tardará un tiempo (Ilustración 176).

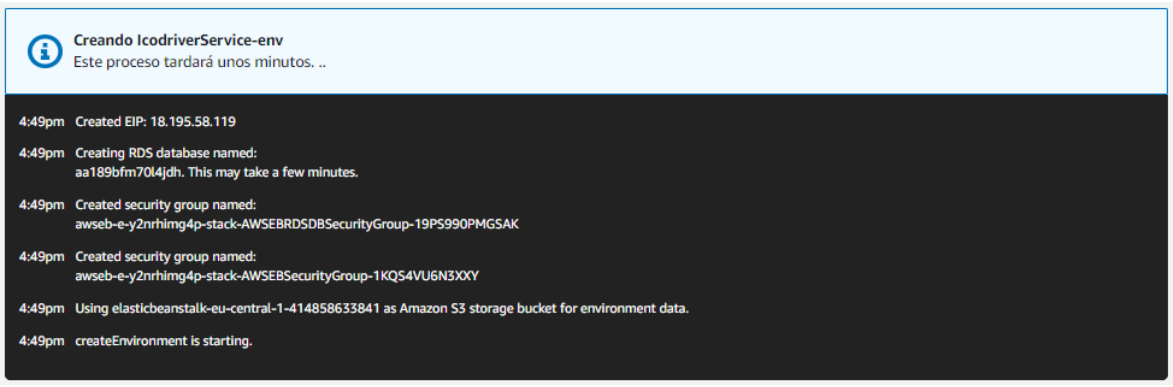

Ilustración 176: Elastic Beanstalk – Creando aplicación

7. Una vez creada la aplicación, nos aparecerá la pantalla mostrada en la Ilustración 177, con la opción "Cargar e implementar" para subir nuestro fichero *.jar* contenedor de nuestro servicio web Spring.

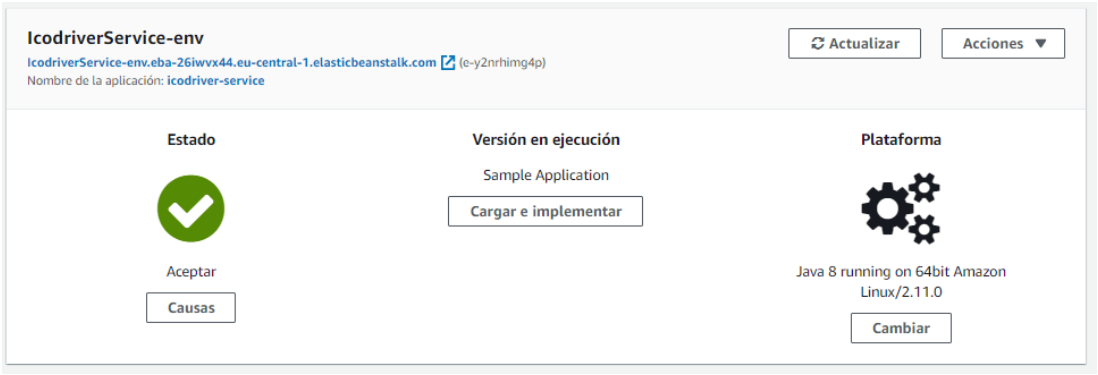

Ilustración 177: Elastic Beanstalk – Aplicación creada

8. Crearemos el archivo *.jar* de nuestra aplicación desarrollada en Spring Boot. Para ello, en Eclipse seleccionamos *Run →Run as →Maven Build.* Nos aparecerá una pantalla como la de la Ilustración 178, donde rellenamos el apartado "Goals" con "package" y pulsamos "Run". El archivo *.jar* se encontrará en la carpeta "target" de nuestro proyecto.

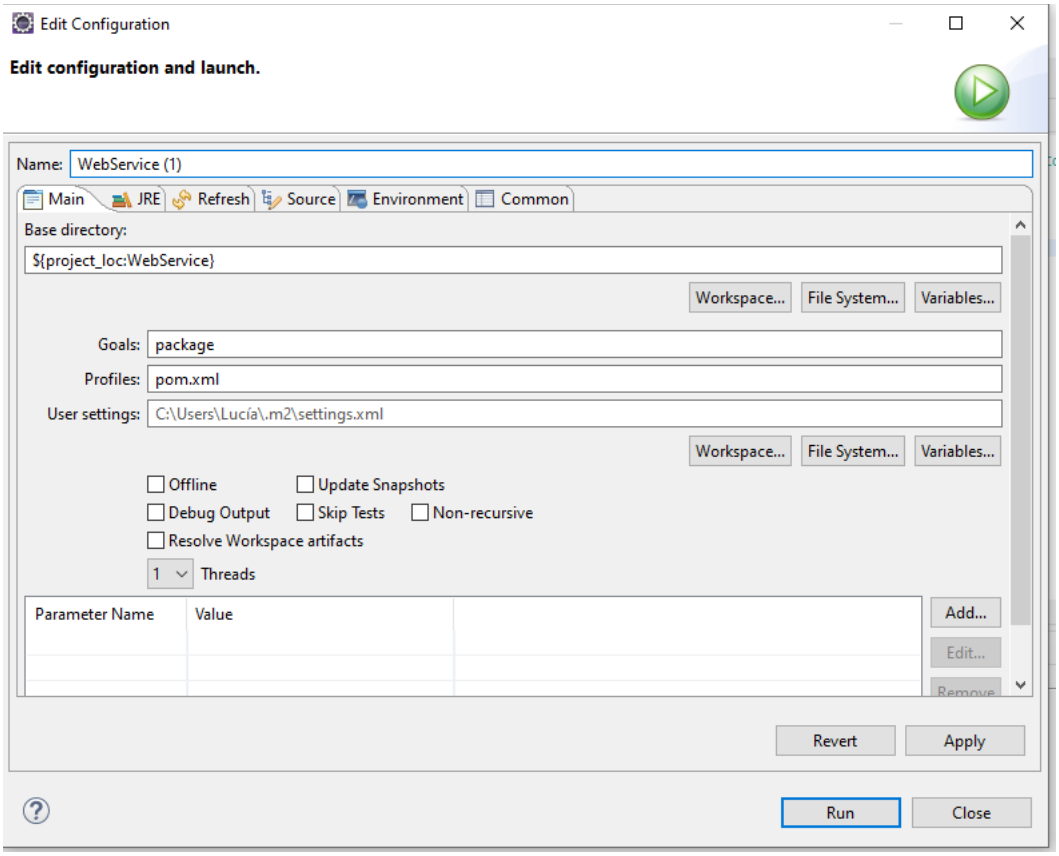

Ilustración 178: Creando el archivo .jar

9. A continuación, continuamos en *Elastic Beanstalk* con la opción "Cargar e implementar" de la Ilustración 177, seleccionamos el *.jar* creado en el paso anterior. Tardará unos minutos en cargar la aplicación, pero una vez hecho, podemos ver un link en la Ilustración 179, a través del cuál podremos realizar las peticiones HTTP y acceder a las funcionalidades implementadas del servicio web.

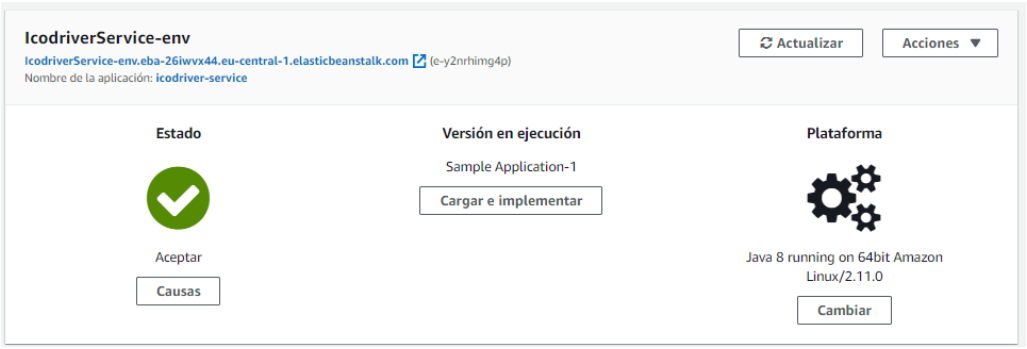

Ilustración 179: Elastic Beanstalk – Aplicación subida y lista para usar

ANEXO E: PREPARACIÓN DEL ENTORNO DE GOOGLE COLAB

En este anexo se detalla cómo preparar el entorno de Google Colab antes de comenzar con el entrenamiento de la red neuronal YOLO.

1. Accedemos a la página oficial de Google Colab [20] a través de nuestro navegador y seleccionamos *Archivo→Nuevo Cuaderno* (Ilustración 180).

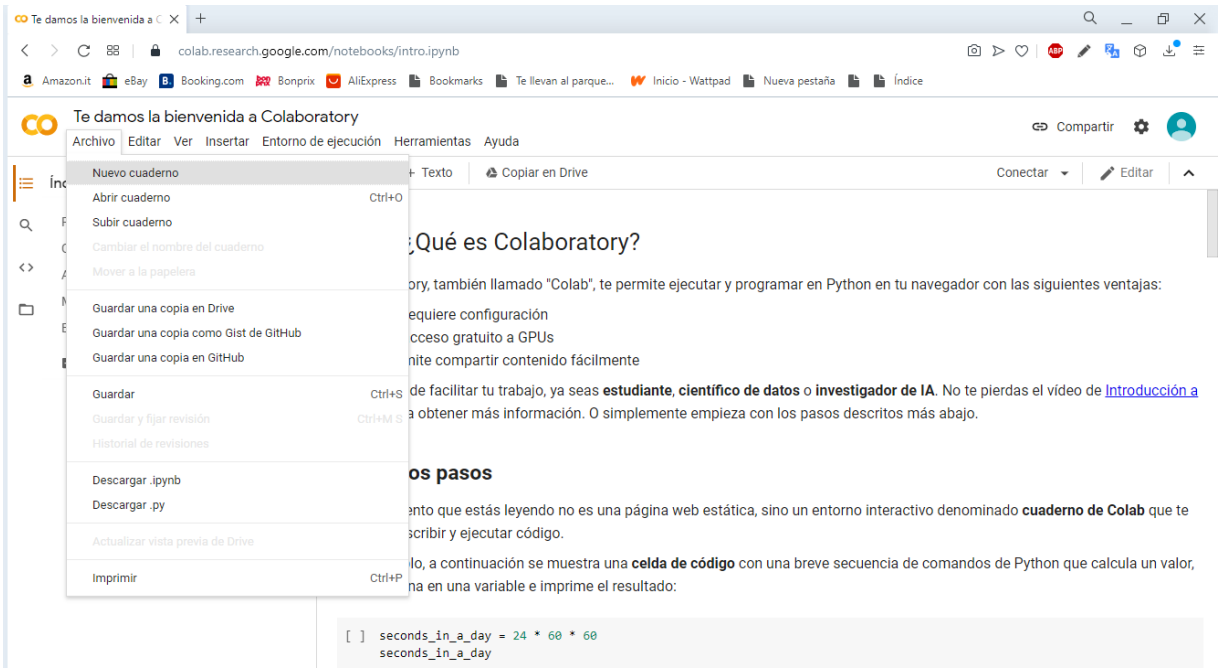

Ilustración 180: Página Principal Google Colaboratory

2. A continuación, en el nuevo entorno de ejecución creado o cuaderno, seleccionamos ->*Configuración del Cuaderno* y elegimos como acelerador por hardware "GPU" y seleccionamos "Guardar" (Ilustración 181).

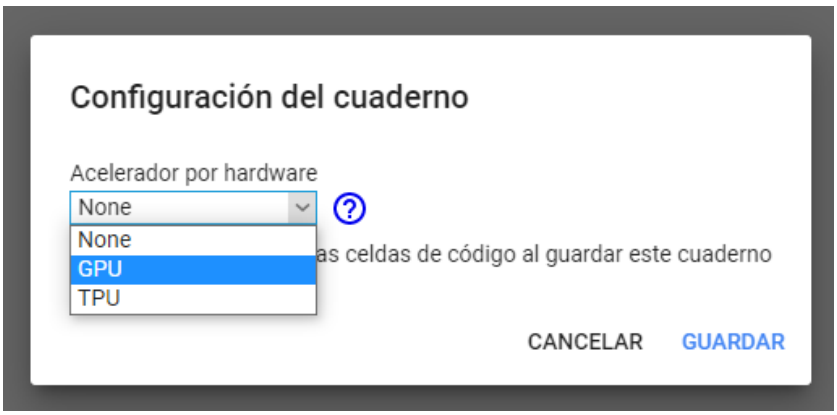

Ilustración 181: Google Colab – Configuración del Cuaderno

3. Enlazamos el cuaderno con una cuenta de Google Drive para poder guardar los pesos que vamos obteniendo del entrenamiento de Yolo, ya que el entorno de Google Colab se reiniciará en 12h y perderemos todo el trabajo hecho. Para esto, escribimos en una celda de código la Ilustración 182 y ejecutamos dicha celda a través del botón de reproducir situado a la izquierda del código o usando la combinación de teclas "Comando/Ctrl + Intro".

```
from google.colab import drive
drive.mount('/content/drive')
```

Ilustración 182: Google Colab – Celda de código para enlazar con Google Drive

4. Una vez enlazada nuestra cuenta de Google Drive, podemos acceder a todos los archivos dentro de esta, que estarán ubicados en el directorio */content/drive* dentro del entorno de ejecución. En la ventana de la izquierda podemos ver estos directorios en el apartado "Archivos" (Ilustración 183).

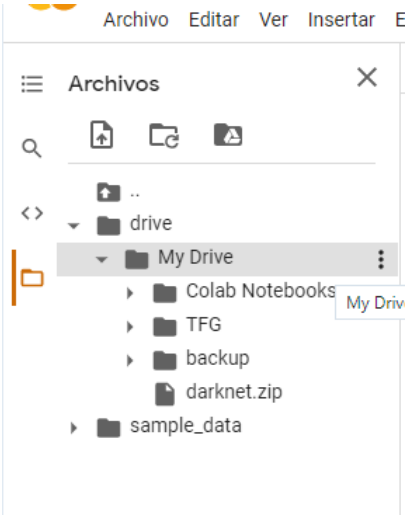

Ilustración 183: Google Colab – Ventana Archivos

5. A continuación, subimos la carpeta *darknet.zip* a nuestra cuenta de Google Drive enlazada y la descomprimos ejecutando la celda de código de la Ilustración 184.

```
[ ]  !unzip "/content/drive/My Drive/darknet.zip"
```

Ilustración 184: Google Colab – Celda de código para descomprimir Darknet

6. Compilamos Darknet a través del comando *make* y le damos permisos de ejecución. Para ello ejecutamos la celda de código de la Ilustración 185.

```
[ ]  %cd /content/darknet
     !make
     !chmod +x ./darknet
```

Ilustración 185: Google Colab – Celda de código para compilar Darknet

7. Creamos un enlace simbólico a la carpeta “Backup” que se encuentra dentro de nuestro Google Drive para guardar ahí los pesos que se vayan generando del entrenamiento de Yolo. Para ello ejecutamos la celda de código de la Ilustración 186.

```
[ ]  !rm /content/darknet/backup -r
     !ln -s /content/drive/'My Drive'/backup /content/darknet
```

Ilustración 186: Google Colab – Celda de código para crear enlace simbólico a “Backup”

8. Ya estaría preparado el entorno de Google Colab y podríamos comenzar a entrenar la red neuronal tiny-yolov3 ejecutando la celda de código de la Ilustración .

```
[ ]  %cd /content/darknet
     !./darknet detector train cfg/traffic.data cfg/traffic-yolov3-tiny.cfg darknet53.conv.74
```

Ilustración 187: Google Colab – Celda de código para entrenamiento

REFERENCIAS

[1] Dirección General de Tráfico, «Anuario Estadístico de Accidentes 2018», [En línea] Disponible: http://www.dgt.es/es/seguridad-vial/estadisticas-e-indicadores/publicaciones/anuario-estadistico-accidentes/

[2] Dirección General de Tráfico, «La velocidad excesiva o inadecuada sigue siendo una de las principales causas de los accidentes de tráfico» *Notas de prensa (01/04/2019),* [En línea] Disponible: http://www.dgt.es/es/prensa/notas-de-prensa/2019/La-velocidad-excesiva-o-inadecuada-sigue-siendo-una-de-las-principales-causas-de-los-accidentes-de-trafico.shtml

[3] S. Mellado Contigioso, *Aplicación Android y Servicio Web Spring para la monitorización de datos obtenidos en un vehículo haciendo uso de la plataforma FIWARE,* Sevilla: Universidad de Sevilla, 2020

[4] A. Carmona Palomares, *Securización mediante Keyrock y Wilma de Aplicación y servicios Web para Vehículo Inteligente,* Sevilla: Universidad de Sevilla, 2020

[5] L. Martínez Ruíz, *Aplicación web con Freeboard para la recolección de datos de sensores de acelerómetro y gps mediante Orion Context Broker de Fiware,* Sevilla: Universidad de Sevilla, 2018

[6] P. Bermejo Pérez, *Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring*, Sevilla: Universidad de Sevilla, 2019

[7] Spring Tool Suite 4, «Spring Tool Suite 4» [En línea] Disponible: https://spring.io/tools

[8] Eclipse, «Desktop IDEs», [En línea] Disponible: https://www.eclipse.org/ide/

[9] Android Studio, «Introducción a Android Studio», [En línea] Disponible: https://developer.android.com/studio/intro

[10] MySQL Workbench, «MySQL Workbench», [En línea] Disponible: https://www.mysql.com/products/workbench/

[11] LabelImg, «LabelImg», [En línea] Disponible: https://github.com/tzutalin/labelImg

[12] Darknet, «Darknet: Open Source Neural Networks in C», [En línea] Disponible: https://pjreddie.com/darknet/

[13] Spring, «Why Spring?», [En línea] Disponible: https://spring.io/why-spring

[14] JPA, «Java Persistence API», [En línea] Disponible: https://www.oracle.com/java/technologies/persistence-jsp.html

[15] MySQL, «MySQL 8.0 Reference Manual», [En línea] Disponible: https://dev.mysql.com/doc/refman/8.0/en/

[16] SQLite, «What Is SQLite?», [En línea] Disponible: https://www.sqlite.org/index.html

[17] Retrofit, «Retrofit», [En línea] Disponible: https://square.github.io/retrofit/

[18] OpenCV, «About», [En línea] Disponible: https://opencv.org/about/

[19] YOLO, «YOLO: Real-Time Object Detection» [En línea] Disponible: https://pjreddie.com/darknet/yolo/

[20] Google Colab, «¿Qué es Colaboratory?», [En línea] Disponible: https://colab.research.google.com/notebooks/intro.ipynb

[21] Maps SDK for Android, «Descripción general», [En línea] Disponible: https://developers.google.com/maps/documentation/android-sdk/overview

[22] Directions API, «Overview», [En línea] Disponible: https://developers.google.com/maps/documentation/directions/overview

[23] AWS, «Overview of Amazon Web Services», [En línea] Disponible: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html

[24] Simon Haykin, *Neural Networks,* Prentice-Hall, New Yersey, USA, 1999

[25] Keiron O'Shea y Ryan Nash, *An Introduction to Convolutional Neural Networks,* 2015

[26] Sumit Saha, *A Comprehensive Guide to Convolutional Neural Networks,* 2015 [En línea] Disponible: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[27] Redmon, Joseph y Farhadi, Ali, *YOLOv3: An Incremental Improvement*, arXiv, 2018

[28] Redmon, Joseph y Farhadi, Ali, *You Only Look Once: Unified, Real-Time Object Detection*, arXiv, 2016

[29] Darknet, «Installing Darknet» [En línea] Disponible: https://pjreddie.com/darknet/install/

[30] Jonathan Hui, *Real-Time Object Detection with YOLO, YOLOv2 and now YOLOv3*, 2018 [En línea] Disponible:https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[31] Spring Web MVC, «Web MVC Framework», [En línea] Disponible: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[32] OpenCV, «Releases» [En línea] Disponible: https://opencv.org/releases/

[33] Tesla, «Piloto Automático», [En línea] Disponible: https://www.tesla.com/es_ES/autopilotAI

[34] Mercedes Benz, «Autonomous», [En línea] Disponible: https://www.mercedes-benz.com/en/innovation/autonomous/

[35] Android Studio, «Android Studio» [En línea] Disponible: https://developer.android.com/studio?hl=es

[36] Eclipse, «Download Eclipse Technology that is right for you» [En línea] Disponible:

https://www.eclipse.org/downloads/

[37] Spring Data JPA, «Accessing Data with JPA» [En línea] Disponible: https://spring.io/guides/gs/accessing-data-jpa/

[38] Spring Security, «Spring Security» [En línea] Disponible: https://spring.io/projects/spring-security

[39] Spring Session, «Spring Session» [En línea] Disponible: https://spring.io/projects/spring-session

[40] MySQL Connector, «Accessing Data with MySQL» [En línea] Disponible: https://spring.io/guides/gs/accessing-data-mysql/

[41] MySQL, «MySQL Community Downloads» [En línea] Disponible: https://dev.mysql.com/downloads/

[42] AWS, «Consola de administración de AWS» [En línea] Disponible: https://aws.amazon.com/es/console/