Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN

Ingeniería Informática

Trabajo Fin de Grado

Procesado de imagen mediante software para dispositivos Android

Autor: José María Berzal Gómez

Tutor/es: José María Gutiérrez Martínez

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN

Trabajo Fin de Grado
Titulo del TFG

Autor:

Director/es:

TRIBUNAL:

(Espacio para la firma)

Presidente:

(Espacio para la firma)

Vocal 1º:

(Espacio para la firma)

Vocal 2º:

CALIFICACIÓN: .....
FECHA: .....

Agradecimiento y dedicatoria

Me gustaría agradecer en primer lugar a mis profesores del grado de ingeniería informática, quienes me han ayudado a desarrollar una base de conocimientos muy amplia gracias a la cual puedo desenvolverme sin problemas en casi cualquiera de las ramas de este campo y en especial a mi tutor, José María, el cual me ha ayudado a poder encauzar este proyecto y a poder llevarlo a cabo, ayudándome a buscar soluciones alternativas cuando lo necesitaba.

También me gustaría agradecer a mi familia, en especial a mis padres y a mi hermana, quienes me han ayudado a ser quien soy y encontrar mi camino en la vida y gracias a los cuales puedo estar hoy escribiendo estas palabras.

Quiero hacer una especial mención a mi abuelo Ramiro, quien murió en febrero del año pasado y al cual se le saltarían las lágrimas si me viera aquí.

No puedo olvidarme de mis amigos, quienes siempre que veo consiguen hacerme olvidar todos los problemas que pudiera tener y con quienes comparto muy buenos recuerdos.

Finalmente quiero agradecer a mi novia, la cual es un punto de apoyo fundamental en mi vida y está ahí siempre que la necesito, celebrando mis logros y apoyándome en los malos momentos.

Resumen

La finalidad de este proyecto es el estudio de la viabilidad del desarrollo de una aplicación mediante Android Studio que se encargue de detectar objetos en tiempo real mediante el uso de una red neuronal en un dispositivo móvil. Este estudio en Android Studio será llevado a cabo en el lenguaje de programación Java, implementando varios modelos de redes neuronales y probando el funcionamiento de estos.

Se realizará un análisis sobre los resultados obtenidos por estos modelos pudiendo determinar así si la aplicación tendría un buen rendimiento en un entorno real.

Palabras clave

Redes neuronales, Android Studio, detección, modelo, tiempo real.

Abstract

The purpose of this project is to study the feasibility of developing an application using Android Studio that is responsible for detecting objects in real time using a neural network on a mobile device. This study in Android Studio will be carried out in the Java programming language, implementing various models of neural networks, and testing their operation.

An analysis will be carried out on the results obtained by these models, being able to determine if the application would have a good performance in a real environment.

Keywords

Neural network, Android Studio, detection, model, real-time.

Índice

Índice de figuras

Índice de tablas

1. Introducción

El campo de la Inteligencia Artificial ha sufrido un avance enorme a lo largo de estos últimos años, el cual se debe, en gran parte, a la aparición de la rama de Machine Learning. El Machine Learning son aquellos algoritmos que permiten a una máquina aprender imitando cómo aprenden los humanos, es decir, es una forma de conseguir inteligencia artificial. Esta aproximación permite resolver problemas mediante diferentes métodos, como la identificación, la clasificación o la predicción.

El otro gran avance que ha recibido el campo de la Inteligencia Artificial es el llamado Deep Learning, el cual es considerado por muchos como la evolución del Machine Learning. El Deep Learning utiliza también algoritmos, pero se diferencia de su rama sucesora en que estos, en lugar de estar basados en árboles de decisión o algoritmos de regresión utiliza una novedosa aproximación, las redes neuronales.

Las redes neuronales tienen un comportamiento muy parecido a las neuronas del ser humano, estableciendo conexiones entre las mismas y formando una red, permitiendo en multitud de ocasiones llegar a ser tan eficiente como el cerebro humano o incluso más. Las redes neuronales cumplen funcionalidades muy diversas y que tienen gran importancia hoy en día.

Una de las principales funcionalidades de las redes neuronales es el procesamiento de imágenes, así como su clasificación. Esto da lugar a una posible automatización de la detección de piezas defectuosas en una fábrica, así como para el control de aforos, entre otras muchas.

He escogido este como tema de mi trabajo de fin de grado, junto al desarrollo de aplicaciones en dispositivos móviles, dado que son dos de los campos de la informática que más me han gustado en mi desarrollo universitario. Además, son dos campos que están en continuo crecimiento actualmente y sobre los cuales es muy importante formarse de cara al desarrollo en el mundo laboral.

2. Objetivos

El objetivo de este TFG es: comprobar la viabilidad de utilizar una red neuronal para la detección de objetos en imágenes sobre dispositivos Android.

Desarrollo del objetivo principal

Como definición de viabilidad se entiende la capacidad para finalizar un proyecto satisfactoriamente, entregando los resultados esperados en un plazo de tiempo aceptable. En este proyecto en concreto, se entiende por viabilidad que la mayoría de los dispositivos Android, es decir, aquellos con unas prestaciones hardware medias, pueda no solo utilizar esta aplicación si no obtener un rendimiento aceptable de la misma.

Subobjetivos

Para lograr este objetivo se han desarrollado los siguientes subobjetivos:

- Desarrollo de una aplicación Android que haga uso de la cámara:

Para lograr este subobjetivo se ha llevado a cabo su desarrollo mediante Android Studio, para dicha aplicación se ha hecho uso de la librería CameraX, la cual está desarrollada por Google. Esta librería otorga varios usos de diversos casos de usos para cumplir diferentes funcionalidades de la cámara, con un nivel de abstracción bastante elevado, lo cual permite acceder a la cámara y hacer uso de esta de una manera muy intuitiva y práctica.

- Integración de la API de Tensorflow Lite:

Este subobjetivo es de vital importancia al permitir el análisis por una red neuronal de la información del entorno obtenida mediante la cámara gracias a uno de los casos de uso de CameraX. Esta API recibe el frame a analizar y devuelve cada uno de los objetos detectados, así como su posición, los cuales serán indicados en la vista previa de la cámara.

- Prueba de los distintos modelos:

La finalidad de este objetivo es la preparación de la aplicación para poder ejecutar varios modelos, de los cuales uno será entrenado por medio del framework Darknet así como la selección e inclusión de los mismos.

- Estudio de los resultados:

Este último subobjetivo está en gran parte relacionado con el subobjetivo anterior, en este se estudiarán los diversos resultados obtenidos por los clasificadores, comprobando de esta manera la influencia del clasificador en el rendimiento de la aplicación. Además, se concluirá sobre la viabilidad de la creación de una aplicación de detección de objetos en un dispositivo Android.

3. Estado del Arte

En esta sección se procederá a dar una explicación detallada sobre llas tecnologías utilizadas para el desarrollo de este proyecto, así como el motivo que indica por qué se han utilizado dichas tecnologías y no otras alternativas.

Android:

Android es el sistema operativo en el cual se desarrollará la aplicación, fue desarrollado por Android Inc, empresa la cual sería adquirida por Google en 2005. Se trata de un sistema operativo desarrollado principalmente para smartphones, tablets, relojes inteligentes o incluso automóviles. Este sistema operativo está basado en núcleo Linux, así como otros softwares open-source. Además, se trata del sistema operativo con mayor cuota de mercado, superando el 90% en 2018, muy por encima de IOS.

Para el desarrollo de aplicaciones en Android se utilizan dos lenguajes de programación con bastante importancia hoy en día, el primero de ellos es Java, lenguaje con el cual se comenzó el desarrollo de aplicaciones para el sistema operativo. El segundo de los lenguajes es Kotlin, nacido a partir de Java, este sería introducido como lenguaje oficial por Google a partir de 2017.

La gran importancia de Android hoy reside en ser un sistema operativo desarrollado para la comodidad de los desarrolladores de aplicaciones, esto se debe a que Android dispone de interfaces de acceso a todas las funcionalidades del teléfono permitiendo utilizar las mismas de una manera sencilla y cómoda.

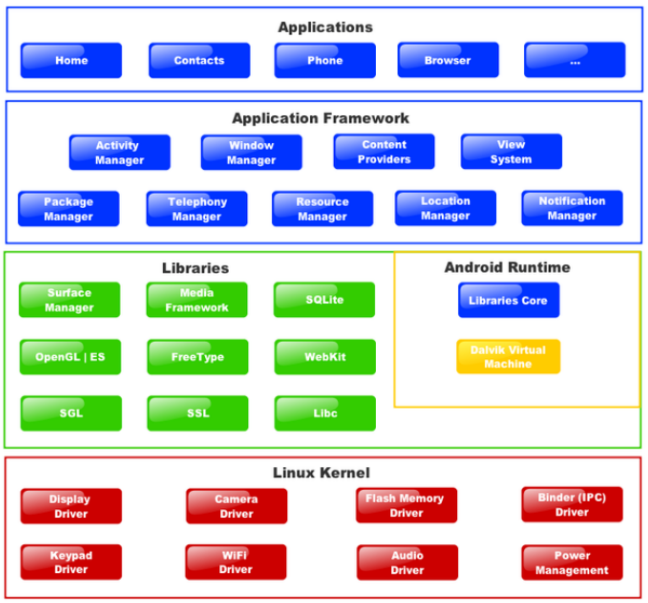

Ilustración 1 Herramientas Android

También destaca la gran cantidad de herramientas gratuitas en las que es posible apoyarse para desarrollar aplicaciones. Es debido a estas dos ventajas que existen una infinidad de aplicaciones para Android, lo cual atrae a muchos usuarios siendo esta una de las principales causas de su gran cuota de mercado.

Otro punto que destacar de este Sistema Operativo es que es completamente open-source, es decir, se permite utilizar este sistema operativo en un teléfono de manera completamente gratuita. También se permite modificar el mismo para crear tu propio sistema operativo, como es el caso de muchos fabricantes de dispositivos móviles en la actualidad. Algunos de ellos son Emui de Huawei o Miui de Xiaomi.

Ilustración 2 Android 9

Actualmente Android, tras ser comprada la empresa que llevó a cabo su desarrollo incluye de forma nativa gran cantidad de las aplicaciones desarrolladas por Google, como pueden ser Google Maps o Google Chrome, a su vez tienen integración con las cuentas de Google, de manera que al introducir la cuenta en el dispositivo se iniciará sesión automáticamente en el resto de las aplicaciones de Google.

Ilustración 3 Android 11

¿Por qué se ha escogido Android?:

Se ha escogido como lenguaje de programación para dispositivos móviles Android ya que es sistema operativo mayoritario hoy en día en dispositivos móviles y tablets, permitiendo así que el proyecto tenga un posible mayor avance. También se ha escogido Android debido a que la programación en este S.O es una variación de Java, lenguaje con el cual ya tengo cierta práctica.

Android Studio:

Android Studio es el entorno de desarrollo integrado (IDE) que se va a utilizar, este IDE se encuentra destinado al desarrollo de aplicaciones para la plataforma móvil Android y está basado en el IDE IntelliJ IDEA.

Android Studio posee un potente editor de código con texto predictivo que permite alcanzar una gran rapidez en el desarrollo de código, así como un flexible sistema de compilación. También se incluyen plantillas de código que también permitirán agilizar el desarrollo de código.

Por último, cuenta con un sistema que permite probar las aplicaciones desarrolladas ya sea por medio de un emulador de teléfono Android, el cual incluye el propio IDE o permitiendo cargar la aplicación en un teléfono real conectado al ordenador mediante un cable USB. A esto se le conoce como ADB o Android Debug Drive.

El emulador incluido con Android Studio permite comprobar el funcionamiento de la aplicación en diferentes teléfonos o tabletas de resoluciones muy variadas, permitiendo comprobar si se observa un correcto funcionamiento en todos ellos, así como si se muestra correctamente la aplicación.

Este IDE facilita también el desarrollo de interfaces gráficas propias, al contar con un conjunto bastante destacable de objetos que se pueden incluir dentro de la aplicación mediante un sencillo sistema de drag and drop. Al hacer drag and drop el nuevo objeto de la interfaz se incluirá inmediatamente en el fichero xml de la interfaz, quedando reflejado entre otras cosas su posición, su tamaño o el color. Esto será renderizado en tiempo real por el renderizador de layouts, quedando reflejado en la vista previa de la APP.

Por último, Android Studio cuenta con integración con servicios como GitHub, el cual indicará qué ficheros se encuentran subidos en la cuenta de GitHub del usuario, así como cuales de ellos presentan cambios desde la última vez que se subieron. Android Studio también tiene soporte para la construcción de proyectos basada en Gradle, lo cual le proporciona al entorno de desarrollo mucha versatilidad.

¿Por qué se ha escogido Android Studio?:

- Como se ha visto anteriormente, es un entorno muy amigable con el usuario con una gran versatilidad, en concreto los principales puntos que han llevado a escoger este entorno de desarrollo han sido los siguientes:

- La integración con GitHub permite acceder rápidamente a los repositorios, así como cargar diferentes ramas del proyecto de manera bastante intuitiva. Esto ayuda en gran medida a la toma de decisiones.

- El sistema ADB, gracias al mismo se han podido realizar múltiples pruebas con la aplicación en tiempo real, visualizándola en el dispositivo móvil, mientras en el ordenador se establecen puntos de ruptura en los cuales se puede observar claramente el estado actual de las variables. Esto permite no solo que aparezcan menos bugs, además permite solucionarlos con rapidez.

- El desarrollo de interfaces en tiempo real permite un aprendizaje rápido, sin tener que interiorizar mucho en el contenido, lo cual me ha venido muy bien ya que a pesar de haber llevado a cabo varios proyectos en Java era mi primera vez llevando a cabo un proyecto en Android.

- Gracias a la integración con Gradle se vuelve muy trivial la tarea de integrar librerías en el proyecto, no teniendo más que declarar la dependencia. El IDE detectará la nueva dependencia generada e integrará al proyecto las librerías necesarias para satisfacer dicha dependencia.

- El último punto que me ha ayudado a decantarme por este IDE es el hecho de que es el IDE más conocido para el desarrollo de aplicaciones Android. Esto supone que haya una gran documentación, foros de ayuda o tutoriales en los cuales se expliquen diferentes funcionalidades de este entorno.

CameraX:

CameraX es el framework sucesor de Camera2, antiguo framework desarrollado por Google, inc que permitía a los desarrolladores el acceso a la cámara del dispositivo móvil.

El motivo del desarrollo de este nuevo framework es aportar una capa mayor de abstracción, simplificando en gran medida el acceso a la cámara y permitiendo obtener un rendimiento mucho más elevado que utilizando el framework Camera2.

Este framework se basa en un modelo de casos de uso, permitiendo al desarollador integrar únicamente aquellos casos de uso necesitados. Esta integración se realiza de manera muy sencilla, declarando el objeto en cuestión, seleccionando la configuración deseada para el caso de uso mediante el uso de métodos .set y por último añadiendo este al ciclo de vida del objeto encargado de instanciar los objetos Camera.

Hoy en día existen 3 casos de uso que se pueden utilizar con CameraX, los cuales son los siguientes:

- Vista Previa

  Mediante este caso de uso se permite el uso de un elemento de la interfaz de usuario de la aplicación Android renderizar la vista de la cámara.

- Análisis de imágenes

  Este caso de uso permite establecer un método (setAnalyzer) que recibirá en cada frame un objeto que representa la captura de la cámara y que podrá ser utilizado para tratamiento de la información.

- Captura de imágenes

  El tercer y último caso de uso incluye un método que tomará una foto y se la pasará al método introducido como parámetro, consiguiendo en este último un objeto que representa la imagen para su guardado o posterior tratamiento.

Machine Learning:

El Machine Learning o aprendizaje automático es la rama de la Inteligencia Artificial cuya finalidad es el desarrollo de técnicas que permitan a los ordenadores aprender. Es decir, consiste en la creación de algoritmos con la capacidad para la generalización de comportamientos, así como el reconocimiento de patrones basándose en una cantidad limitada de ejemplos suministrados por el desarrollador.

Esto no es más que un proceso de inducción del conocimiento, es decir, permite obtener por medio de la generalización un caso general a partir de múltiples casos particulares. Esto, aplicado a este proyecto en concreto podría ser por ejemplo la detección general de una manzana en una grabación en tiempo real, lo cual sería el caso general, a partir de múltiples fotos de manzanas.

Para considerar una generalización válida es necesario obtener una inducción completa, esta es lograble a partir de haber observado todos los casos particulares. Esto en general, pero sobre todo en el ámbito del Machine Learning es imposible debido a la cantidad de casos que existen, esta situación tiene como consecuencia directa que exista un grado de incertidumbre en las afirmaciones y que por tanto se pueden justificar las mismas empíricamente.

El objetivo principal del Machine Learning es, por tanto, extraer conocimiento que le permita detectar características o propiedades no observables de un objeto a partir de otras propiedades que si se hayan observado del mismo o de otros objetos parecidos. A este proceso es al cual se llama aprendizaje.

El Machine Learning permitirá por tanto a una máquina aprender ciertos comportamientos para luego poder generalizarlos y así aplicarlos a cualquier situación. Para llevar a cabo este proceso de aprendizaje existen múltiples algoritmos, los cuales se clasifican principalmente en dos grandes grupos:

- Aprendizaje supervisado:

El aprendizaje supervisado trabaja con datos etiquetados, esto quiere decir que el usuario presenta a la máquina un dataset de entrada en el cual se indica la solución o salida que se debe obtener al presentar cada uno de esos datos del dataset. El algoritmo se entrenará con este dataset o histórico de datos con soluciones y desarrollará un algoritmo capaz de asignar la etiqueta adecuada, de las cuales se le han enseñado, a un nuevo valor de entrada.

Este aprendizaje es el utilizado en dos tipos de problemas muy definido, estos son los problemas de clasificación y los problemas de regresión.

  - Problemas de clasificación:

En los problemas de clasificación el resultado producido por la inteligencia artificial debe ser una clase entre un número limitado de clases.

En este tipo de problemas se obtiene un porcentaje de pertenencia para cada clase y el algoritmo clasifica el objeto en el grupo con mayor porcentaje de pertenencia. Una práctica muy común en la resolución de este tipo de problema suele ser definir un límite de aceptación inferior, con este medio se logra que si no haya ningún porcentaje de pertenencia que supere dicho umbral la imagen se clasifique como indefinido. Esta funcionalidad descrita previamente tiene gran importancia permitiendo evitar posibles errores al clasificar indebidamente los datos de entrada.

Una técnica específica de clasificación es la regresión logística, la cual se verá con el resto de las técnicas de Machine Learning.

  - Problemas de regresión:

La principal diferencia de este grupo con respecto a los problemas de clasificación es que el resultado de estos es un valor numérico, es decir, a partir de los datos de entrada la inteligencia artificial tratará de predecir el valor resultado. El propósito de este análisis es establecer un modelo que permita relacionar un número determinado de características con una variable objetivo.

Por ejemplo, se puede establecer un problema de regresión que, utilizando como datos de entrada la gente que ha realizado el mismo trayecto que se va a realizar calcule el tiempo aproximado que se va a tardar.

Así como los problemas de clasificación, los problemas de regresión utilizan sus técnicas específicas de Machine Learning, las cuales son la regresión lineal y no lineal.

- Aprendizaje no supervisado:

Este tipo de aprendizaje, al contrario que el aprendizaje supervisado, no requiere que el conjunto de datos de entrada se encuentre etiquetado. Estos modelos analizarán los datos infiriendo características de estos y clasificarán los datos mediante el uso de dichas características o puntos en común, permitiendo extrapolar las características para después utilizarlas en la selección de la clase de pertenencia de nuevos datos.

A estos modelos se les conoce como modelos de aprendizaje no supervisado ya que no requieren ser guiados hacia la solución, esto conlleva que los modelos de aprendizaje supervisado y los modelos de aprendizaje no supervisado tengan utilidades muy diferentes, utilizando los segundos principalmente para descubrir relaciones ocultas entre los datos o la estructura subyacente del conjunto de datos, hecho que fuera difícil de lograr sin estos modelos.

Los algoritmos de aprendizaje no supervisado permiten llevar a cabo tareas mucho más complejas que el aprendizaje supervisado, no obstante, el aprendizaje no supervisado puede producir una salida impredecible en muchos casos, es más, al no saber previamente los resultados deseados no se puede determinar cómo de precisos son los mismos.

Por tanto, el momento óptimo para utilizar el aprendizaje no supervisado es cuando no se dispone de los resultados para los datos deseados o cuando se trata de buscar nuevos tipos de patrones entre los datos. Este tipo de aprendizaje es utilizado para la resolución de dos problemas principalmente:

  - Agrupamiento o clustering:

Esta es la funcionalidad principal de los algoritmos de aprendizaje no supervisado, se presentan los datos y mediante el uso de un algoritmo se trata de agrupar los datos en diferentes clústeres o grupos, si existen. Su finalidad es la de encontrar un patrón o estructura a partir de un conjunto de datos no categorizados. Estos algoritmos permiten modificar el número de grupos en los que se dividirán los datos.

  - Asociación:

La finalidad de estos problemas es la de establecer asociaciones entre diversos datos dentro de grandes bases de datos. Es decir, permite descubrir relaciones interesantes entre variables de grandes bases de datos. Un ejemplo de esto sería establecer una relación entre los artículos que se compran por internet para sugerir al usuario artículos de compra recomendados en función del artículo que se va a comprar.

- Aprendizaje por refuerzo:

Aparte de las dos técnicas clásicas de Machine Learning existe una novedosa técnica conocida como aprendizaje por refuerzo. Esta técnica, a diferencia de las dos técnicas anteriores tratará de lograr un aprendizaje en la máquina utilizando un enfoque distinto, este enfoque nuevo consiste en aplicar un esquema de recompensas y castigos, "premiando" a la máquina cuando se obtenga un mejor resultado.

Esto tendrá como consecuencia un cambio sustancial el objetivo final del algoritmo, buscándose maximizar dicha recompensa en lugar de minimizar el error, como se podía ver tanto en el aprendizaje supervisado como en el no supervisado.

En este enfoque se encuentran principalmente dos componentes, estos son el agente, el cual realizará las acciones obteniendo las recompensas o castigos y el ambiente, el cual reaccionará a las acciones del agente cambiando de estado y notificando al agente de estos cambios además de premiando o castigando al agente.

Este tipo de aprendizaje se suele utilizar cuando se tiene un entorno muy amplio, dentro del cual existen una gran variedad de factores que hay que tener en cuenta, con lo cual sería muy complicado definir las etiquetas de salida como en el aprendizaje supervisado debido a que la respuesta debe basarse en un número demasiado elevado de factores y por tanto sería un proceso muy lento, mientras que el aprendizaje no supervisado no encaja en este tipo de problemas.

El aprendizaje por refuerzo no deja de ser un método de entrenamiento basado en la fuerza bruta mediante el cual, por ejemplo, en el caso de un coche autónomo para que aprenda a conducir será necesario dejar que se choque en un entorno simulado para castigarle por ello y que no lo repita fuera del mismo.

¿Por qué machine learning supervisado?:

Se ha utilizado Machine Learning Supervisado para este proyecto debido a que en tanto en la clasificación de imágenes como en la detección de objetos, la cual es un subtipo de la clasificación de imágenes, se sabe en todo momento cuál es la salida esperada o el resultado a alcanzar, siendo esta salida esperada la correcta etiquetación de la imagen o del objeto deseado.

También se ha optado por el uso de esta técnica ya que el Machine Learning no Supervisado, sería únicamente capaz de identificar qué objetos presentan una gran semejanza, pero no permitiendo la etiquetación de estos.

Se ha optado por el aprendizaje supervisado frente al aprendizaje de refuerzo ya que al no haber muchos factores dentro del entorno de los cuales dependa la salida, en el caso a tratar únicamente la localización de un objeto para el cual se haya entrenado la red, es menos complicado y tedioso el uso de técnicas de aprendizaje supervisado.

Añadiéndoles peso a los tres motivos previos, se ha decidido por esta tecnología debido al uso de dispositivos móviles para la ejecución de la aplicación a desarrollar, los cuales no disponen de la misma potencia de procesado que un ordenador y por tanto podrían presentar problemas de rendimiento al ejecutar en tiempo real algoritmos de Machine Learning no Supervisado a la vez que la ejecución de la propia aplicación, mientras que los algoritmos de Machine Learning Supervisado no cuentan con este problema al permitir entrenar previamente el modelo.

Técnicas de Machine Learning Supervisado:

A continuación, se verán los diferentes tipos de algoritmos existentes que se pueden utilizar para la resolución de problemas de Machine Learning Supervisado:

Regresión logística:

La regresión logística es una técnica de aprendizaje supervisado que solamente se utiliza para la clasificación. Está técnica está conformada por la unión de dos partes bien diferenciadas:

- Una combinación lineal: Está combinación es la suma de cada una de las entradas por sus respectivos pesos.

- Función logística: Acto seguido, se aplica la función logística al resultado obtenido por la combinación lineal. Esta función únicamente permite valores entre 0 y 1, los cuales se pueden ver como porcentajes de pertenencia. Es por esta razón que esta técnica se utiliza para los problemas de clasificación.

La regresión logística es una técnica muy sencilla de implementar, pero no por ello menos eficiente, se suele utilizar para problemas de clasificación binaria. En estos hay dos clases de pertenencia mutuamente excluyentes y se sitúa la frontera entre las clases en 0.5. En caso de que la regresión lineal devuelva un valor superior a 0.5, la clase de pertenencia para el nuevo dato es la clase 1, en caso de que se devuelva un valor inferior a la clase 0.

Esta regresión logística es una pequeña introducción a las redes neuronales, las cuales se verán posteriormente, contando con los mismos elementos. Valores de entrada, cada uno con su peso y una función de activación, en el caso de la regresión lineal la logística.

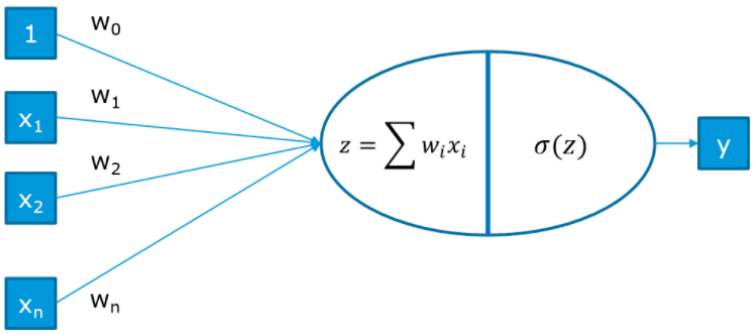

Ilustración 4 Regresión Logística

Regresión lineal:

Este algoritmo tratará de establecer un modelo que le permita ajustar la relación de dependencia entre un valor o característica y su correspondiente resultado. Es decir, una vez ajustada la recta se podrá calcular el valor Y que le corresponde a cualquier X. Para este algoritmo se posicionan todos los valores y su correspondiente valor resultado y se posiciona una recta arbitraria. Posteriormente se calcula la distancia de dicha recta a todos los puntos, conociendo a esta distancia como el residuo o error de predicción.

En cada iteración el algoritmo recolocará la recta tratando de disminuir dicho error al mínimo, el algoritmo finalizará cuando el algoritmo haya alcanzado aquella recta con el mínimo error, en otras palabras, aquella que sea más cercana a la mayoría de los puntos colocados inicialmente.

Para lograr esta aproximación, así como minimizar el error de la recta en cada iteración existen dos métodos:

- Desplazamiento de la recta:

  Para este método en primer lugar se calculará la posición de la recta utilizando dos puntos W1 y W2, que son la inclinación de la recta y el punto de corte con el eje de abcisas respectivamente. Tras saber la posición actual de la recta a partir de estos dos puntos se calcula la distancia a los puntos que representan los datos y se procede a aproximar la recta, tras realizar la aproximación se repite el proceso hasta que no se pueda aproximar mas la recta a los puntos.

- Descenso por el gradiente:

  Este método se apoya en el uso del gradiente, el gradiente indica la dirección de la máxima variación de un campo. Aplicando este concepto al error de predicción se puede ver que para obtener el mínimo valor del error es posible utilizar el gradiente con signo negativo, permitiendo descender así por la función del error. Este algoritmo, al obtener un valor como salida, es principalmente utilizado para los problemas de regresión.

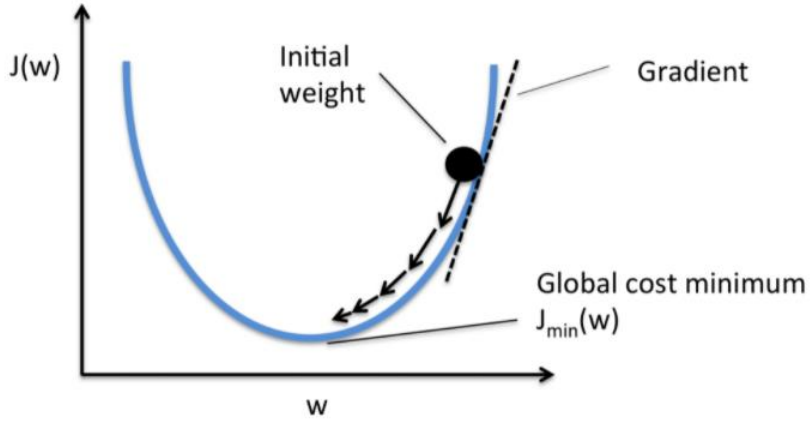

Ilustración 5 Descenso por el gradiente

Regresión no lineal:

Este algoritmo se basa en los mismos principios que la regresión lineal, ajustarse al mayor número de puntos posible, tomando como diferencia que en este caso la función que modela los datos no es lineal. Entre este tipo de funciones se consideran las funciones exponenciales, logarítmicas y trigonométricas entre otras. Esto nos permite hacer una aproximación certera en algunos casos en los cuales la aproximación lineal presente un alto error de predicción.

En esta regresión se vuelve un poco más complejo hallar el mínimo global para el error, ya que existen múltiples mínimos locales. Para intentar obtener un menor valor del error se usan valores aproximados en los parámetros junto con el algoritmo de optimización, apoyándose también en el método de mínimos cuadrados ponderados. Este algoritmo, junto con la regresión lineal al obtener un valor como salida, se utilizan en los problemas de regresión.

Máquinas de vectores de soporte

Este algoritmo nace de la necesidad para satisfacer los problemas de clasificación. Esta máquina trata de encontrar la separación entre dos o más clases, para hacer esto encontrará el hiperplano (debido a que estos problemas no tienen por qué darse en dos dimensiones) que maximice el margen de separación entre clases, o lo que es lo mismo, encontrará el hiperplano que se encuentre más separado de cualquier punto.

Para lograr este objetivo el algoritmo se apoya en los determinados vectores de soporte, estos vectores de soporte son aquellos datos que se encuentran más próximos al otro clúster y son los que, por tanto, definirán el margen máximo. El margen máximo indica la separación entre el grupo y el hiperplano.

Es habitual y completamente normal que algún dato se encuentre mal clasificado o sea un dato anómalo, para tratar estos datos anómalos y seguir encontrando una separación entre los dos grupos, se les puede indicar a este tipo de máquinas un valor C, el cual indica la cantidad de puntos que se pueden encontrar mal clasificados.

Por último, se hará una mención a una técnica con gran importancia para las máquinas de vectores de soporte, esta técnica es conocida como el truco del kernel. Este truco consiste en inventar una nueva dimensión auxiliar en nuestro problema de clasificación, permitiendo gracias a esto encontrar solución para ciertos problemas de clasificación que, con un menor número de dimensiones, no tendrían.

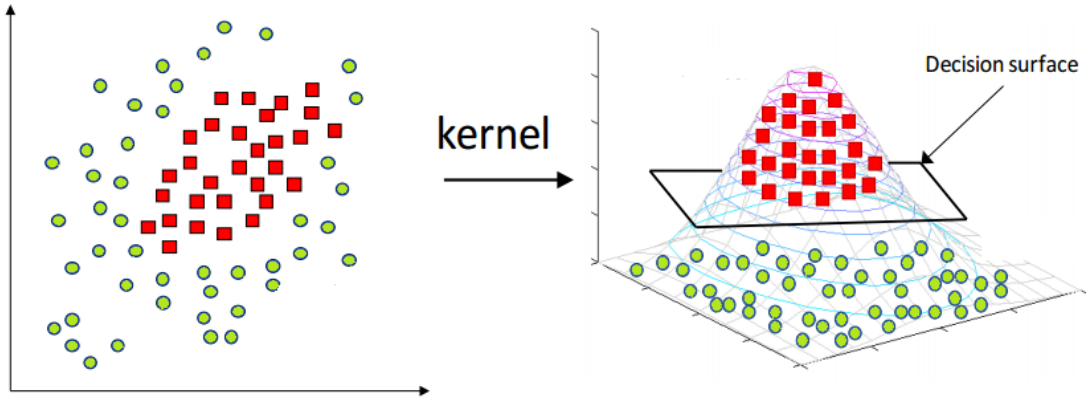

Ilustración 6 Truco del kernel

Árboles de decisión:

Los árboles de decisión son una técnica de aprendizaje supervisado que permitirá predecir la salida gracias a agrupar las observaciones similares en un mismo subgrupo o clase, estableciendo reglas para pasar del nodo raíz a los hijos de este.

Es una técnica utilizada tanto para la clasificación, obteniendo buenos resultados con un coste de modelación muy bajo como para la regresión, en la cual no son muy aconsejables. Sin embargo, no toleran los datos outliers y romperán por completo la estructura del árbol, ya que es necesario llevar a cabo una generalización de los datos dentro de las clases. Esto a su vez tiene como consecuencia una tendencia al overfitting de los datos, haciendo que los datos con anomalías sean difícilmente clasificables.

En primer lugar, se clasifican las observaciones en las diferentes clases y se procede a crear el árbol de decisión. Partiendo del nodo raíz se dividirán todos los individuos en dos subgrupos, utilizando una regla o decisión para indicar a cada nuevo dato hacia qué subgrupo debe dirigirse. Esto repetirá hasta que ninguno de los nodos hijos contenga datos de varias clases.

Bosques aleatorios:

Un bosque aleatorio es un conjunto de árboles de decisión combinados con la técnica bagging. La técnica de bagging consiste en entregarle a cada árbol un subconjunto de los datos formado a partir de escoger una muestra aleatoria del mismo para de esta forma tratar de obtener una respuesta lo más generalizada posible, solventando así una de las limitaciones de los árboles de decisión y compensando los errores de la salida de unos árboles con otros.

La técnica de bagging puede utilizarse para cualquier modelo de machine learning aunque son utilizados principalmente para los bosques aleatorios debido a que los árboles de decisión son muy rápidos de construir, obteniendo una mejor generalización sin sacrificar un buen rendimiento.

Tras aplicar bagging se obtiene un resultado para cada árbol de decisión a partir de la muestra aplicada al mismo, para obtener una respuesta final se aplicará una técnica de votación entre las respuestas obtenidas.

Dentro de estas técnicas de votación destacan tanto la votación suave, en la cual no todos los votos valen lo mismo, dando más importancia a aquellos árboles que den como respuesta valores muy próximos al 0% o al 100% para los problemas de clasificación, así como la media aritmética para los problemas de regresión.

Las desventajas de estos bosques aleatorios es que tienden al overfitting de los datos, como ocurría con los árboles de decisión.

Red Neuronal Artificial:

Una red neuronal artificial es un modelo computacional simple que emula el funcionamiento de un cerebro biológico estableciendo conexiones con diferentes pesos entre los nodos de la red neuronal.

La información de entrada atraviesa la red neuronal, dentro de la cual se somete a varias operaciones, obteniendo unos valores de salida, esto se consigue gracias a los enlaces establecidos entre las neuronas. Cada uno de estos enlaces tiene un peso, los cuales incrementarán o inhibirán los valores de las neuronas adyacentes. Aparte de esto se suele disponer de una función de activación a la salida de cada neurona, esta función limitará el valor máximo o modificará el resultado antes de propagarse a la siguiente neurona.

La gran ventaja de las redes neuronales es que aprenden y se forman a sí mismas, logrando destacar notablemente en áreas donde es difícil expresar mediante la programación convencional el objetivo o la detección de soluciones. Entre estas áreas destacan la visión por computador o el reconocimiento de voz, dos tareas bastante difíciles de resolver utilizando la programación basada en reglas.

La forma de aprender de dichas redes neuronales es mediante la minimización de la función de pérdida, tal y como otras técnicas de aprendizaje supervisado vistas previamente, para esto en las redes neuronales se utiliza un algoritmo conocido como backpropagation o propagación hacia atrás de errores.

- Backpropagation:

Se conoce como backpropagation o propagación hacia atrás de errores al método utilizado para el entrenamiento de redes neuronales. Este método utiliza un ciclo basado en dos fases, conocidas como propagación y adaptación.

En la primera fase o fase de propagación se escogen varios patrones que se aplicarán a la entrada, estos patrones se propagarán por las capas intermedias de la red neuronal hasta generar sus respectivas salidas. Por último, se comparan estas salidas con la salida deseada y se calcula el error entre la salida deseada y las obtenidas, lo cual será de vital importancia para la segunda fase.

En la segunda fase se propagan lo errores hacia atrás, comenzando por la capa de salida, llegando a la entrada de la red y pasando por todas las capas ocultas en su camino. Estas neuronas recibirán una fracción de ese error, basándose dicha fracción en la contribución de la neurona para la salida original. Esto permite que, al alcanzar la entrada de la red cada neurona habrá recibido una fracción de error que indicará su contribución relativa al error obtenido.

Al saber la contribución relativa de cada neurona al error se pueden ajustar los pesos con mucha mayor facilidad, con la finalidad de reducir el error de la red en la medida de lo posible.

¿Por qué redes neuronales?:

Se han elegido las redes neuronales como técnica de machine learning supervisado a utilizar ya que facilita mucho la detección de objetos al tratrarse de un problema muy complejo para la mayoría de las técnicas descritas previamente. Las redes neuonales permiten que el propio sistema determine la forma más eficiente, así como los patrones utilizados para reconocer los objetos, evitando que deba hacerlo el usuario.

Aparte, se han escogido las redes neuronales al tratarse de una de las técnicas más utilizadas actualmente, tratándose de una tecnología novedosa y muy eficiente.

Detección de objetos:

La detección de objetos nace a partir de la visión artificial con la intencionalidad de detectar casos de objetos semánticos de una misma clase en vídeos o imágenes. En la detección de objetos se incluyen la detección de caras, así como la de personas. La detección de objetos tiene gran importancia en la sociedad actual, utilizándose entre otras cosas para la conducción autónoma de un vehículo, la ayuda a personas invidentes o la videovigilancia.

Un uso no tan destacable a simple vista, pero no por ello menos importante es el seguimiento de objetos en tiempo real, permitiendo seguir un balón durante un partido o una persona durante un video.

La detección de objetos se basa en el reconocimiento de características comunes a todos los elementos de un grupo y posteriormente utilizar esas características para comprobar si ese objeto se encuentra en el entorno en cuestión. Por ejemplo, en el caso del círculo, una característica común a todos los círculos sería que todo el contorno de este es equidistante al centro. Esto permitiría que a pesar de cambios en el color o tamaño se pueda detectar siempre un círculo.

Lo que se acaba de ver es un ejemplo muy simple de cómo se produce la detección de objetos. No obstante, con ejemplos bastante más complicados como la detección de una cara, un objeto con bastantes más características que es necesario concretar para que la detección se produzca correctamente, es muy tedioso programar dichas características.

Es por esta razón que en este campo tienen gran influencia las redes neuronales, permitiéndonos que la máquina establezca esas reglas que nos permitan detectar el objeto en cuestión sin esfuerzo alguno.

TensorFlow Lite:

TensorFlowLite o TFLite es el framework que se utilizará para implementar los modelos de redes neuronales, se trata de una de las únicas opciones que permite integrar las mismas en un dispositivo Android. Este framewrok permite la creación de un objeto ObjectDetector que pasándole la imagen a analizar (como un mapa de Bits) retornará una estructura de datos que contiene todas las detecciones realizadas en la imagen.

Estas detecciones son representadas mediante objetos Detection, los cuales contienen tanto la boundary box del objeto (rectángulo que indica su posición) como la clase en la que se ha clasificado ese objeto o el porcentaje de pertenencia a esta clase.

A este objeto ObjectDetector será necesario también indicarle como parámetro durante su creación el modelo que se va a utilizar para dichas detecciones, el cual debe estar previamente en formato .tflite.

4. Desarrollo

En esta sección se procederá a explicar con todo detalle el completo desarrollo de la aplicación. Previamente a esto, es necesario indicar que este proyecto se ha desarrollado siguiendo las directrices del desarrollo iterativo del software, técnica del software basada en desarrollar un producto por medio de iteraciones, en cada una de dichas iteraciones se obtiene un producto funcional que puede ser probado. De esta manera el sistema se desarrolla poco a poco y se puede obtener feedback del producto desarrollado continuamente.

Este proceso iterativo llevado a cabo se ha dividido en tres fases, estas fases deben repetirse en todas las iteraciones sobre el producto que se lleven a cabo:

- Planteamiento:

  En esta fase se fijarán los objetivos a llevar a cabo en la iteración actual, debiendo ser estos de tipo incremental salvo en la primera iteración, en la cual se establecerá la base.

- Desarrollo:

  Segunda fase de la iteración, en esta se procederá con la implementación de los objetivos establecidos en la fase de planteamiento.

- Pruebas:

  Tercera y última fase, en esta se realizarán pruebas sobre el dispositivo Android para comprobar si se han implementado correctamente los objetivos establecidos en esta iteración. En esta fase también se solventarán los bugs pertinentes para el correcto funcionamiento del proyecto.

¿Por qué se ha escogido el desarrollo iterativo?

Se ha optado por el desarrollo iterativo debido principalmente a la obtención de productos funcionales en cada iteración, facilitando así tanto llevar a cabo las pruebas de lo implementado en esa iteración y permitiéndome de esta manera llevar a cabo una mayor depuración del código. He considerado que esto me sería de gran ayuda dado que estoy utilizando tecnologías que no había utilizado previamente.

Iteración 1:
Planteamiento:

El objetivo de esta primera iteración fue el de desarrollar una aplicación Android simple que permitiera el funcionamiento de la cámara.

La aplicación Android simple, como se ha especificado previamente, se desarrollará en el IDE Android Studio. Esta aplicación estará conformada por dos interfaces gráficas, siendo la primera el menú desde el cual se podrá acceder a la cámara. La otra interfaz será la interfaz de la cámara y mostrará la vista previa de la misma en pantalla completa.
Para lograr el objetivo de permitir el funcionamiento de la vista previa de la cámara se utilizará la librería CameraX.

Por último, para probar el correcto funcionamiento de lo anterior se utilizará el debug de Android Studio, ejecutando la aplicación en mi dispositivo personal por medio del modo de debug ADB.

Desarrollo:

Lo primero que hubo que llevar a cabo tras la instalación de Android Studio fue la creación de un nuevo proyecto y su integración con GitHub, conocida herramienta que se utiliza para el control de versiones de un componente software. Esto además de por su funcionalidad de control de versiones es importante ya que permite mantener una copia de seguridad del proyecto en cuestión en la nube, asegurando así el proyecto.

Tras la integración con GitHub y la subida del commit inicial se procedió a crear la interfaz del menú principal, conocida como MainActivity. Al crear la clase se crea su fichero xml asociado a la interfaz gráfica, el cual permitirá modificar de forma más precisa que mediante el uso del editor gráfico la posición el tamaño o el color de los componentes de la interfaz gráfica que conformará el menú principal.

Esta interfaz se encontrará, por el momento, compuesta de únicamente un botón que llevará al usuario hasta la cámara. Para la creación del botón, mediante el menú de componentes gráficos de Android Studio se puede añadir el mismo con una gran facilidad haciendo drag and drop.

El siguiente paso es la implementación de un listener, este elemento se encargará de recoger una acción en concreto del usuario sobre el dispositivo y dará una respuesta a dicha acción. En este caso concreto, se utilizará el listener para redirigir el usuario a la interfaz que contiene la cámara cuando este pulse el botón.

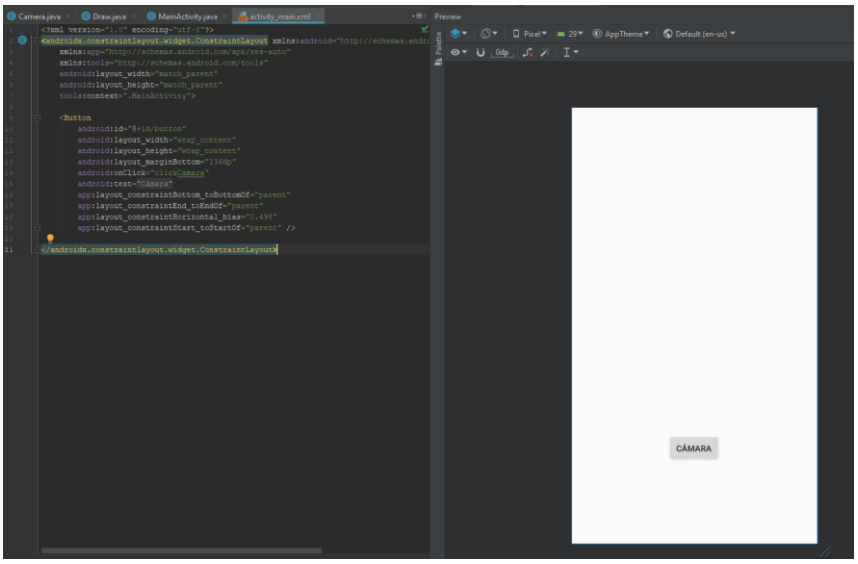

Ilustración 7 XML interfaz principal

A continuación, se llevará a cabo la creación de la interfaz de la cámara, interfaz principal de la aplicación. En esta interfaz se realiza tanto la captura de los fotogramas en tiempo real como la indicación al usuario de los objetos reconocidos en la imagen en tiempo real. El primer paso es la inclusión de la librería de CameraX, declarando las dependencias en el fichero build.gradle asociado al proyecto.

```
dependencies {
    // Librería core de CameraX
    def camerax_version = "1.0.0"
    // CameraX Lifecycle para poder asignar casos de uso.
    implementation "androidx.camera:camera-lifecycle:$camerax_version"
    // Vista previa de CameraX
    implementation "androidx.camera:camera-view:1.0.0-alpha19"
    //Implementacion de Camera2 para las dependencias de CameraX
    implementation "androidx.camera:camera-camera2:${camerax_version}"
```

Ilustración 8 Dependencias necesarias

Posteriormente, se creará el layout de esta interfaz, en esta primera iteración se opta por un layout muy simple, el cual consta únicamente de un objeto PreviewView. Este objeto, integrado por la librería de CameraX, es una vista que permite mostrar el contenido de la cámara en caso de que se instancie un objeto con el caso de uso del mismo nombre. Para este proyecto se ha optado por no dar opción al usuario de mostrar la cámara frontal y solo centrarse en la trasera.

Esto se debe a que las cámaras traseras disponen de una mayor resolución que la frontal, además de incluir otras funcionalidades como el zoom, el flash o el gran angular. A estas características se le incluye una mayor facilidad de uso, al ser la sujeción del dispositivo mas intuitiva. Estas características, junto a que no se obtiene ventaja alguna por el uso de la cámara frontal han hecho que se haya optado por no permitir al usuario utilizar la cámara frontal en vez de la trasera.

El siguiente paso es escribir el código que permitirá a la vista previa presentarse correctamente, para lo cual se declara un objeto PreviewView, que se utilizará para guardar la instancia de este en el layout, pudiendo acceder a este posteriormente. Tras la creación de la interfaz de la cámara se hará una llamada a el método iniciarCamara(), este se encarga de instanciar el objeto que hará verse la vista previa.

Como se indicó en el estado del arte, CameraX funciona mediante casos de uso, definiéndose los casos de uso a utilizar al instanciar el objeto de la cámara. Para instanciar uno de estos objetos es necesario instanciar previamente un ProcessCameraProvider, en el cual se definirán los casos de uso a utilizar por las posteriores instancias de la cámara, la rotación, la resolución o la cámara seleccionada entre otras.

A continuación, es necesario instanciar los casos de uso que cargará la cámara, por el momento únicamente se instanciarán la clase CameraSelector (necesario instanciarlo para que funcione correctamente la cámara) así como la clase Preview, caso de uso que permitirá obtener la vista previa de la cámara en tiempo real sobre un contenedor de la GUI.

El último paso llevado a cabo en esta función es la creación de un objeto Camera, su vinculación a los casos de uso definidos previamente y la asociación del caso de uso Preview al objeto PreviewView del layout.

Pruebas:

Se realizan pruebas para observar si hasta el momento la aplicación muestra los layouts correctamente, el funcionamiento del botón, así como el de la vista previa.

Para realizar las pruebas es necesario en primer lugar cargar el teléfono en el ordenador, conéctandolo mediante un cable USB y seleccionando la opción ADB en las opciones de ejecución del proyecto, lo cual cargará la aplicación en el teléfono, arrancándola.

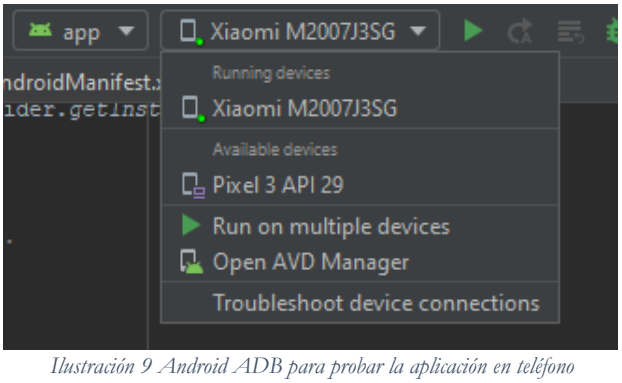

Ilustración 9 Android ADB para probar la aplicación en teléfono

Se observa lo siguiente:

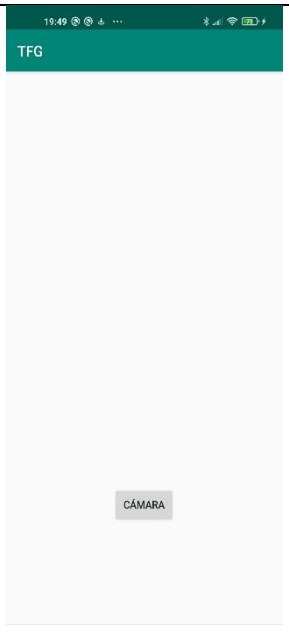

Ilustración 10 Interfaz principal

Ilustración 11Interfaz cámara

La aplicación se inicia sin problemas, mostrando la interfaz main junto con su correspondiente botón, bien situado dentro del layout. Al pulsar el mismo se pasa a la interfaz de la cámara, activándose la cámara trasera y mostrando la vista previa correctamente.

Este subobjetivo es de vital importancia al permitir el análisis por una red neuronal de la información del entorno obtenida mediante la cámara gracias a uno de los casos de uso de CameraX. Esta API recibe el frame a analizar y devuelve cada uno de los objetos detectados, así como su posición, los cuales serán indicados en la vista previa de la cámara.

Iteración 2:

Planteamiento:

El objetivo de la segunda iteración es el de integrar la librería de Android de TensorFlow lite, que en conjunto con el caso de uso ImageAnalysis de CameraX permitirá utilizar el modelo cargado en el dispositivo para analizar los fotogramas en busca de objetos que se encuentren dentro del dataset en tiempo real.

Además de la integración de la librería es necesario establecer algún mecanismo que se encargue de mostrar en la vista previa qué objetos se han detectado y donde se han visto. Para lograr este objetivo se ha decidido dibujar un cuadrado sobre la vista previa indicando la posición de cada uno de los objetos detectados. A su vez se dibujará el nombre del objeto detectado encima del cuadrado. Todo esto permitirá reconocer de forma simple la posición de los objetos detectados, así como que objetos son los detectados.

Desarrollo:

El primer paso que llevar a cabo para esta iteración es la definición de las dependencias de TFLite, gracias a las cuales se podrán cargar modelos de TensorFlow y utilizar los mismos para la detección de objetos.

En este caso será necesaria únicamente la librería visión, encargada de todo lo relacionado con la clasificación de imágenes, detección de objetos y segmentación de imágenes. Para un uso correcto de la librería será necesario establecer dos parámetros, el primero evitará que los ficheros .tflite (modelos de TFLite) sean comprimidos para su correcta lectura por la aplicación. El segundo activará la vinculación de datos, permitiendo que se establezcan relaciones entre los modelos y sus respectivos metadatos.

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    //Se compila con Java 8
    compileOptions {
        sourceCompatibility = 1.8
        targetCompatibility = 1.8
    }
    aaptOptions {
        noCompress "tflite" // Evita la compresión de los modelos TFLite
    }
    dataBinding {
        enabled = true // Activa la vinculación de datos.
    }
}

dependencies {
    // Librería core de CameraX
    def camerax_version = "1.0.0"
    // CameraX Lifecycle para poder asignar casos de uso.
    implementation "androidx.camera:camera-lifecycle:$camerax_version"
    // Vista previa de CameraX
    implementation "androidx.camera:camera-view:1.0.0-alpha19"
    //TFLite dependencies
    implementation 'org.tensorflow:tensorflow-lite-task-vision:0.1.0'
```

Ilustración 12 Valores de parámetros para el correcto uso del framework

Tras la declaración de las dependencias se procede con la inclusión del caso de uso imageAnalysis de CameraX, este caso de uso procesará los frames en tiempo real y permitirá el análisis de dichos frames.

En primer lugar, es necesario instanciar un objeto de la clase ImageAnalysis. Esto se debe hacer mediante el método Builder() de la clase. El método Builder() devuelve un objeto de tipo Builder, el cual contendrá los valores del objeto ImageAnalysis que se desea crear. Este objeto de clase Builder contiene a su vez una gran variedad de métodos que devuelven a su vez otro objeto de clase Builder, incluyendo esta vez información referente al método que se llamó.

Gracias a esto se puedan encadenar llamadas a estos métodos, obteniendo finalmente un objeto de tipo Builder que contenga toda la información especificada. En último lugar, será necesario una llamada al método build() para que se retorne un objeto de clase ImageAnalysis, creado siguiendo las directrices del objeto Builder.

En el caso de este proyecto se realiza una llamada tanto a .setTargetResolution(Size) como a .setBackpressureStrategy(int). El primer método definirá la resolución objetivo de las imágenes que se analizarán, es decir, se utilizará la resolución del dispositivo. Con respecto al segundo método, indica que estrategia seguirá el caso de uso en caso de encontrarse con una situación de "backpressure".

El "backpressure" es un problema que se refiere principalmente a computación debido al cual, en un proceso de transformación de una entrada a una salida, se forma un cuello de botella en el propio proceso, obteniendo nuevos datos en la entrada a una velocidad mayor que la velocidad de transformación de estos. Esta situación trae como consecuencia que se forme una cola de datos en el buffer de entrada.

En el método setBackpressureStrategy(int) se ha escogido como valor "ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST", la estrategia indicada por este valor descarta toda la cola de frames, quedándose únicamente con el último frame. Este valor permite al dispositivo tener una menor carga al no tener que analizar todos los frames encolados, quedándose únicamente con el frame más actual.

Gracias a este método se descartan frames intermedios los cuales son innecesarios para el problema a tratar, teniendo mayor importancia definir correctamente el estado actual del entorno, cuya mejor representación es el último frame.

```
// Se prepara el caso de uso ImageAnalysis.
//Es necesario indicar la resolución de la pantalla al caso de uso.
DisplayMetrics displayMetrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
int height = displayMetrics.heightPixels;
int width = displayMetrics.widthPixels;


ImageAnalysis imageAnalysis =
        new ImageAnalysis.Builder()
                .setTargetResolution(new Size(width,height))
                //EStrategia utilizada ante problemas de back pressure.
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .build();
```

Ilustración 13 Creación de objeto ImageAnalysis

Para utilizar correctamente este caso de uso será necesario asignarle un objeto de tipo Analyzer, mediante el método setAnalyzer. Se utilizará este objeto cuando el frame se haya generado correctamente y esté listo para su análisis, realizando el caso de uso una llamada a su método analyze(ImageProxy).

Se creará un nuevo objeto de tipo Analyzer junto con su respectivo método analyze(ImageProxy) haciendo uso del tag @override, reemplazando así el código de dicha clase para esta instancia en concreto.

Para analizar estos frames es necesario instanciar un objeto ObjectDetector, este objeto posee el método detect(TensorImage), el cual retorna una lista de objetos Detection, estos contienen tanto el identificador del objeto que se ha detectado como la posición en la cual se ha encontrado.

Este objeto ObjectDetector se instancia a partir del método createFromBufferAndOptions(MappedByteBuffer, ObjectDetectorOptions), siendo el primer parámetro un buffer de bytes que cargará el modelo de TFLite que utilizará el detector de objetos y el segundo las opciones del ObjectDetector que se desea instanciar.

La creación de un objeto ObjectDetectorOptions funciona de manera muy similar a la de un objeto ImageAnalysis, accediendo al Builder que incluye las opciones por defecto mediante el método build(). Utilizando después los respectivos métodos set para cambiar dichos valores por defecto.

En el caso de este proyecto se ha decidido cambiar el número máximo de resultados, estableciéndolo en 5 en lugar de los 10 que vienen por defecto. Se ha considerado este valor ya que producirá un incremento del rendimiento al permitir al detector de objetos dejar de buscar nuevos objetos que clasificar antes en cada fotograma.

También se ha optado por incrementar el valor del umbral de puntuación del detector, estableciendo el mínimo porcentaje de pertenencia del supuesto objeto en un 70% para poder asegurar que dicho objeto pertenece a la clase. Este incremento del valor del umbral permitirá obtener una mayor precisión al ser más exigente con el detector de objetos, otra consecuencia de esta mayor exigencia es un decremento en la cantidad de objetos detectados en el fotograma, permitiendo que no se supere el umbral de 5 objetos detectados en pantalla en la mayoría de las situaciones.

Una vez instanciado el ObjectDetector será necesario llamar al método detect del mismo y este utilizará el modelo para buscar los objetos en la imagen, retornándolos en la lista de objetos Detection. Como modelo se ha escogido un modelo de ejemplo dispuesto por la documentación de TensorFlow utilizándose para probar la detección de objetos, sustituyéndolo posteriormente por el modelo que se desarollará posteriormente.

El último paso restante sería iterar sobre cada uno de estos objetos Detect para dibujar sobre la pantalla tanto el nombre del objeto como su boundary box.

Para dibujar esta información en pantalla es necesario superponer objetos de tipo View sobre la vista previa de la cámara, por lo cual se ha definido una clase Draw que hereda de la clase View y en la que se hará override del método onDraw para incluir los elementos en pantalla. Esta clase Draw contiene además un método inicializar() que establecerá los valores de los parámetros con los cuales se va a dibujar en la imagen, como el color o el tamaño del borde.

```
public class Draw extends View {

    private Context context;
    private RectF rect;
    private float threshold;
    private String tag;
    public Paint paint;
    public Paint tagPaint;
// Se pasan como parámetros el boundary box, el umbral y el tag que indica el nombre del objeto.
public Draw(Context context, RectF rect, float threshold,String tag) {
    super(context);
    this.context = context;
    this.rect = rect;
    this.threshold = threshold;
    this.tag = tag;
    inicializar();
}

private void inicializar() {
        paint = new Paint();
        paint.setColor(Color.BLUE);
        paint.setStrokeWidth(10f);
        paint.setStyle(Paint.Style.STROKE);
        tagPaint = new Paint();
        tagPaint.setColor(Color.BLACK);
        tagPaint.setTextSize(200f);
        tagPaint.setStrokeWidth(40f);
        tagPaint.setStyle(Paint.Style.FILL);
        }

        @Override
        public void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        String label = ""+threshold+"   "+tag;
        canvas.drawText(label, rect.centerX(), rect.top, tagPaint);
        canvas.drawRect(rect.left, rect.top, rect.right, rect.bottom, paint);
        }
```

Ilustración 14 Clase Draw

Finalmente se incluye un bucle para que se itere sobre los objetos detectados y se dibujen todos ellos, cumpliendo con esto el objetivo de la iteración 2.

```
imageAnalysis.setAnalyzer(ContextCompat.getMainExecutor( context: this), new ImageAnalysis.Analyzer() {
    @Override
    public void analyze(@NonNull ImageProxy image) {
        Bitmap imagen = imageToBitmap(image.getImage());
        ObjectDetector objectDetector;
        ObjectDetector.ObjectDetectorOptions.Builder optionsBuilder;
        MappedByteBuffer modelBuffer;
        try {
            modelBuffer = FileUtil.loadMappedFile(context,model);
            optionsBuilder = ObjectDetector.ObjectDetectorOptions.builder().setMaxResults(5).setScoreThreshold(0.7f);
            objectDetector = ObjectDetector.createFromBufferAndOptions(modelBuffer,optionsBuilder.build());
           List<Detection> detectados =  objectDetector.detect(TensorImage.fromBitmap(imagen));
            for(Detection detectado: detectados){
                Draw dibujar = new Draw(context,detectado.getBoundingBox(),detectado.getCategories().get(0).getScore(),detectado.getCategories().get(0).getLabel());
                vista.addView(dibujar);
                views++;

            }
            //Es necesario cerrar la imagen asociado al frame para poder analizar los siguientes,
            //Si no se hace esto se congela la imagen.
            image.close();
        }catch (IOException e)
        {
            System.out.println("Error de I/O");
        }


    }
});
```

Ilustración 15 Método analyze

Pruebas:

Se procede con la fase de pruebas, para lo cual se instala la aplicación en el dispositivo móvil por medio de Android Studio y se obtiene el siguiente resultado:

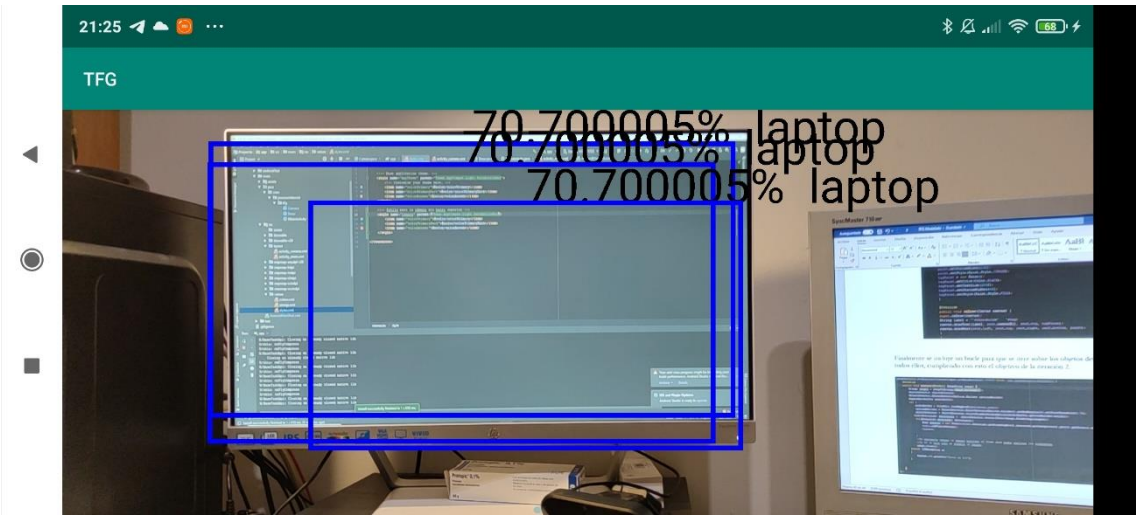

Ilustración 16 Prueba detección de objetos

Al realizar estas pruebas se han podido observar varios problemas:

- No se borran las vistas de frames anteriores ya que en cada frame se van dibujando más boundary boxes sobre el objeto en cuestión.

- La UI se superpone sobre parte del campo del Analyzer y la PreviewView, causando situaciones como que aparezcan boundary boxes tapadas por esta UI.

Para solucionar el primer problema se ha optado por inicializar atributo que contabilice el número de vistas activas en cada momento sobre la vista previa de la cámara (el objeto PreviewView es un contenedor de vistas que incluye la propia vista de la cámara), eliminando todas estas y dejando únicamente la vista que dibuja la cámara. Esto se lleva a cabo al principio de cada frame.

Con respecto al segundo problema es necesario utilizar flags que harán que cuando se presente la pantalla de la cámara desaparezca la banda verde superior y que se esconda la barra de navegación del teléfono. Además, se ha usado otra flag para que cuando se muestre esta barra sea de modo transparente, evitando así que interfiera en la visibilidad del usuario.

Iteración 3:

Planteamiento:

El objetivo de esta tercera iteración es la preparación de la aplicación para poder seleccionar entre más de un modelo entrenado, así como la selección de tres modelos que serán introducidos en la aplicación.

Desarrollo:

Para la selección del modelo se ha optado por la inclusión de varios botones de cámara, cada uno para un modelo de los incluidos en la aplicación. Estos botones se apoyarán en un Enum que facilitará en gran medida la codificación del código dependiente del modelo, al ser mucho más intuitivo que el nombre del modelo en cuestión.

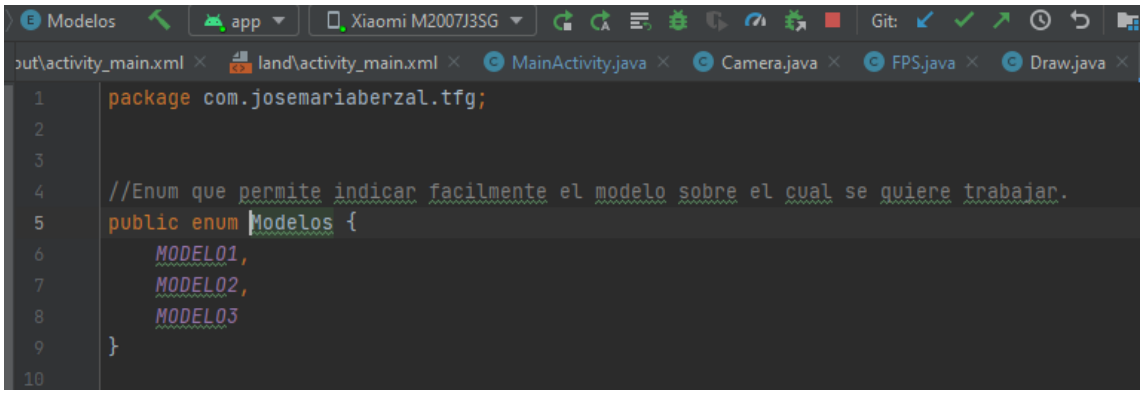

Ilustración 17 Creación Enum Modelos

El siguiente paso consta de la mejora en la interfaz gráfica principal, añadiendo además los tres botones que permiten seleccionar entre los distintos modelos que se integrarán posteriormente, además de introducir el título del TFG como texto en la aplicación.

Para cumplir esta funcionalidad se han creado tres listeners diferentes, cada uno de los cuales seleccionará un modelo que será enviado a la Activity Camera mediante el método .putExtra(name, Enum). Este método enviará a la siguiente Activity que se abra el valor de la Enum Modelos seleccionado, el cual se trata mediante un switch.

```
model = (Modelos)getIntent().getSerializableExtra( name: "model");
//Se selecciona un fichero en función del valor para Modelos seleccionado.
switch (model){
    case MODELO1:
        fileModelo = "modelo1.tflite";
        break;
    case MODELO2:
        fileModelo = "modelo2.tflite";
        break;
    case MODELO3:
        fileModelo = "modelo3.tflite";
        break;
}
```

Ilustración 18 Switch selección de modelo

Con esto ya se encuentra las interfaces preparadas para soportar los distintos modelos y se abrirá la cámara con el modelo seleccionado.

El siguiente paso consta de la selección de 3 modelos con la finalidad de probar el rendimiento de la aplicación, uno de los cuales va a estar basado en el clasificador YoloV4, este es uno de los más conocidos para dispositivos de bajo rendimiento debido al algoritmo utilizado para utilizar la detección.

Este algoritmo utiliza redes neuronales convolucionales para la detección de objetos, por lo cual requiere únicamente de una propagación completa dentro de la red neuronal para ser capaz de detectar los objetos. El algoritmo YOLO combina tres técnicas muy conocidas del Machine Learning que aplica simultáneamente para lograr una detección a tiempo real y con buena precisión.

Para generar el modelo que se integrará en la aplicación Android es necesario llevar a cabo tres diferenciados pasos:

- Generación del dataset y etiquetación de este utilizando el software Roboflow.
- Creación del respectivo modelo entrenando el dataset, previamente etiquetado, mediante el framework Darknet y utilizando para ello el entorno Google Colab.
- Conversión del modelo .weights generado por el framework a .tflite, permitiendo su inclusión en la aplicación.

El primer paso, como se ha indicado en el apartado anterior, consiste en llevar a cabo la creación del dataset y su posterior etiquetación. Para esto es necesario primeramente disponer de las imágenes que se incluirán al dataset, por lo que se han tomado dichas fotografías.

Este dataset contiene 45 imágenes de latas de refresco, dispuestas en distintas posiciones y con distintas orientaciones, cambiando la superficie y no situando siempre las mismas en el centro de la imagen. Para la creación del dataset se ha seguido el siguiente procedimiento:

1) Se capturan las imágenes en las diferentes posiciones y se cargan las mismas en Roboflow.com y se lleva a cabo la etiquetación de estas en la herramienta de la aplicación.

   Esta etiquetación es manual y se debe ir identificando dónde se encuentra el objeto / los objetos a clasificar dentro de la imagen marcando los mismos y se indica a qué clase pertenecen. En este caso solo podían pertenecer al conjunto lata.

2) En segundo lugar, es necesario seleccionar cómo se distribuirán las imágenes en los distintos sets. Para este caso se ha optado por la distribución por defecto de la página, siendo un 70% de las imágenes para entrenar, un 20% para validaciones y un 10% para pruebas posteriores.

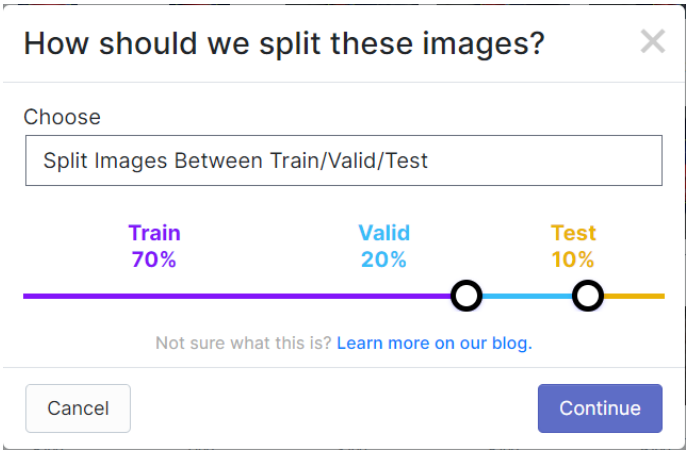

Ilustración 19 Distribución de porcentajes de entrenamiento

3) Tras repartir las imágenes se pasa a la selección de las configuraciones de preprocesamiento, en este caso se usarán varias de estas, las cuales son el resize de las imágenes a 416x416 pixeles, la orientación automática de las imágenes, así como el ajuste automático del contraste. Estas tres configuraciones seleccionadas permiten reducir en gran cantidad el tiempo empleado en el entrenamiento, sin reducir la eficacia de este.

4) Una vez pasado el preprocesamiento se pueden seleccionar opciones de "Data Augmentation", estas tienen la finalidad de aumentar el dataset gracias a introducir imágenes que son resultado de otras imágenes pertenecientes al dataset modificadas. En el caso de este proyecto se han seleccionado tanto un cambio en la rotación de las imágenes de 90º, como cambios en la saturación de estas y la creación de nuevas imágenes a partir de establecer cuatro imágenes del dataset como un mosaico.

5) Por último, al dar por finalizada la configuración se genera un zip que se entrega al usuario mediante una URL, conteniendo tanto las imágenes como las bounding boxes de cada una de las imágenes.

Tras haber generado las etiquetas se da por concluido el primer paso y se procede al paso de entrenar la red neuronal utilizando el framework Darknet.

Para este entrenamiento se ha utilizado la herramienta Google Colab, esta herramienta permite hacer uso de recursos informáticos como puede ser una GPU o una CPU. A su vez permite la ejecución de código Python, esto la hacen una herramienta muy atractiva para la creación de redes neuronales.

De cara a proceder con el entrenamiento de los datasets es necesario realizar una configuración previa:

1) Se comprueban las características de la GPU aportada al entorno de prueba por Google Colab. Esto es debido a que para el entrenamiento se usará la GPU en lugar de la CPU.

   El entrenamiento se realiza utilizando la GPU ya que la CPU obtiene un mejor rendimiento para tareas complicadas. Sin embargo, la GPU contiene un mayor número de núcleos mucho más simples que los de una CPU, esto conlleva que se obtenga una rapidez mucho mayor en la realización de operaciones simples pero repetitivas, caso del entrenamiento de la red neuronal.

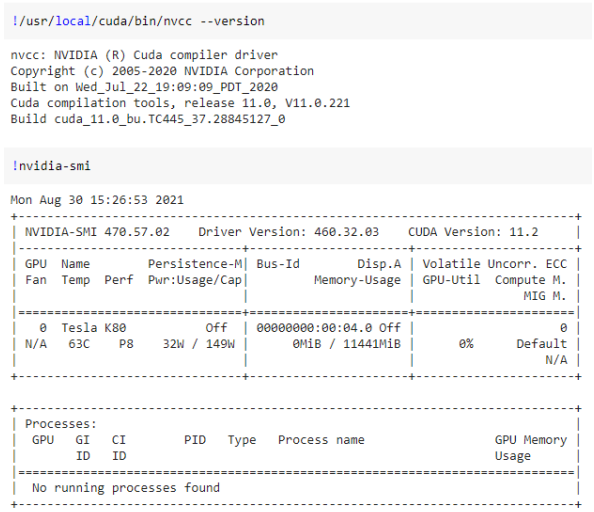

Ilustración 20 Propiedades tarjeta gráfica Google Colab

2) El segundo paso es la descarga del repositorio Github del framework Darknet, este Framework es ampliamente utilizado para el entrenamiento de redes neuronales, obteniendo un rendimiento muy elevado y optimizando el entrenamiento. Así se logran muy buenos resultados en poco tiempo de entrenamiento.

   Este framework, construido en C utiliza código CUDA para hacer uso de la GPU, por lo cual es necesario indicarle la GPU que se está utilizando, cambiando la configuración por defecto para el modelo concreto de GPU. Tras cambiar la configuración por defecto para indicar la GPU utilizada se compila el código C mediante el comando make.

3) Se descargan los pesos iniciales para la red neuronal, gracias a esto se obtienen unos mejores resultados en un menor tiempo, al guiar a la red neuronal inicialmente hacia el resultado obtenido. Estos pesos pueden ser los pesos finales obtenidos en un entrenamiento similar, en este caso se han descargado los pesos recomendados por los creadores del algoritmo YoloV4 para detección de objetos.

4) Una vez finalizados los pasos previos se procede con la carga de los datasets, estos deberán contener 4 tensores o variables de salida, al ser los únicos aceptados por la librería TFLite de AndroidStudio.

5) A continuación, se utiliza el comando "darknet detect train" para comenzar a entrenar el modelo en cuestión. La duración del entrenamiento depende en gran medida de los parámetros utilizados para el entrenamiento, así como la cantidad de imágenes o la potencia de la GPU utilizada.

6) Tras entrenar el dataset se obtendrán varios ficheros .weights, estos ficheros contienen la información de los pesos del modelo, que permitirán reproducir el modelo entrenado en cualquier entorno.

   Darknet genera múltiples ficheros .weights, haciendo distintos backups de la red neuronal según la va entrenando(cada 1000 iteraciones). También se genera un modelo con el mejor resultado obtenido, así como los pesos finales. Estos pesos finales, al ser los más maduros serán los que se utilizarán para la aplicación.

Una vez se ha entrenado el modelo se procederá con la tercera parte de la creación del modelo, en esta parte se transformarán los ficheros .weights a formato .tflite, ya que es el único formato admitido por el framework TFLite que está siendo utilizando en la aplicación Android.

Para este proceso, existe una librería de Python llamada "tensorflow-yolov4-tflite" que realiza esta conversión de una manera muy eficiente. Esta librería permitirá pasar los pesos generados por el framework Darknet a TensorFlow y de TensorFlow a TensorFlowLite, permitiendo conseguir los ficheros necesarios en solo dos pasos.

```
!python save_model.py \
  --weights /content/darknet/backup/custom-yolov4-tiny-detector_final.weights \
  --output ./checkpoints/yolov4-tiny-pretflite-416 \
  --input_size 416 \
  --model yolov4 \
  --tiny \
  --framework tflite \

!python convert_tflite.py --weights ./checkpoints/yolov4-tiny-pretflite-416 --output ./checkpoints/yolov4-tiny-416.tflite
```

Ilustración 21 Librería para convertir formato Darknet a TFLite

Al intentar importar este modelo dentro de la aplicación Android y proceder con su uso se obtiene el siguiente error:

```
java.lang.AssertionError: Error occurred when initializing ObjectDetector: Input tensor has type kTfLiteFloat32: it requires specifying NormalizationOptions metadata to preprocess input images.
    at org.tensorflow.lite.task.vision.detector.ObjectDetector.initJniWithModelFdAndOptions(Native Method)
    at org.tensorflow.lite.task.vision.detector.ObjectDetector.access$000(ObjectDetector.java:86)
    at org.tensorflow.lite.task.vision.detector.ObjectDetector$1.createHandle(ObjectDetector.java:152)
    at org.tensorflow.lite.task.vision.detector.ObjectDetector$1.createHandle(ObjectDetector.java:145)
    at org.tensorflow.lite.task.core.TaskJniUtils$1.createHandle(TaskJniUtils.java:70)
    at org.tensorflow.lite.task.core.TaskJniUtils.createHandleFromLibrary(TaskJniUtils.java:91)
    at org.tensorflow.lite.task.core.TaskJniUtils.createHandleFromFdAndOptions(TaskJniUtils.java:66)
    at org.tensorflow.lite.task.vision.detector.ObjectDetector.createFromFileAndOptions(ObjectDetector.java:143)
    at com.josemariaberzal.tfg.Camera$2.analyze(Camera.java:192)
```

Ilustración 22 Error tipo de dato kTfLiteFloat32

Replanificación Iteracion 3:

En este subapartado se tratará de solventar el problema referente al tipo de dato de los tensores de entrada, explicando cada una de las alternativas descubiertas investigando y en caso de no lograr una solución, la solución alternativa a implementar.

Este error es producido debido a que el framework no ha generado los metadatos asociados al modelo correctamente, algo necesario ya que se deben especificar las "NormalizationOptions" en los metadatos debido a que los datos de entrada y salida del modelo son de tipo float32. Estas "NormalizationOptions" permiten definir la media y varianza del modelo, lo cual es necesario cuando se utiliza float32.

Estos metadatos se pueden comprobar desde Android Studio y efectivamente no se encuentran asociados al archivo. Realizando una búsqueda de información se han encontrado únicamente dos maneras diferentes de poder sortear este problema:

- Convertir el modelo reemplazando el tipo de los valores de Float32 a int8:

De esta manera no será necesario vincular unos metadatos al modelo ni utilizar los mismos, para esto el script "convert_tflite.py" dispone de un parámetro que permite especificar el tipo de dato deseado. Entre estos tipos de datos se encuentra int8, el único otro tipo de dato aceptado por TFLite aparte de float32.

Tras convertir el modelo utilizando el parámetro se observa que el modelo es todavía float32. Produciendo el mismo error que se ha mostrado previamente.

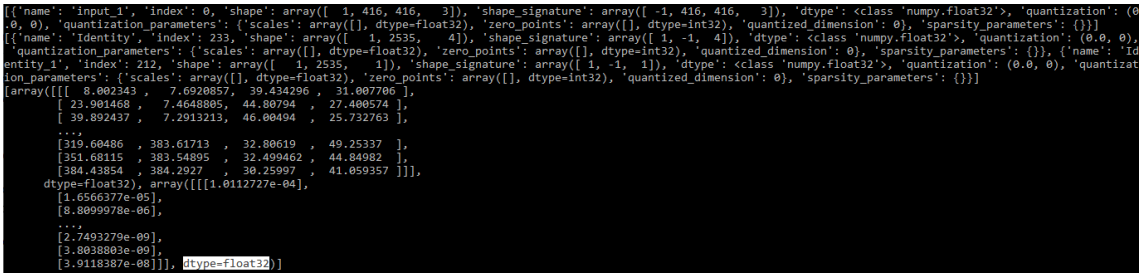

Ilustración 23 Tipo de valores del modelo tras seleccionar int8 como tipo de valores

- Generar los metadatos asociados al modelo:

Para llevar a cabo este paso se ha utilizado un script generado por TensorFlow que permite que se generen automáticamente los metadatos asociados al modelo.

```
from tflite_support.metadata_writers import object_detector
from tflite_support.metadata_writers import writer_utils

ObjectDetectorWriter = object_detector.MetadataWriter
_MODEL_PATH = "yolov4.tflite"
# Task Library expects label files that are in the same format as the one below.
_LABEL_FILE = "labelmap.txt"
_SAVE_TO_PATH = "modellYolo.tflite"
# Normalization parameters is required when reprocessing the image. It is
# optional if the image pixel values are in range of [0, 255] and the input
# tensor is quantized to uint8. See the introduction for normalization and
# quantization parameters below for more details.
# https://www.tensorflow.org/lite/convert/metadata#normalization_and_quantization_parameters)
_INPUT_NORM_MEAN = 127.5
_INPUT_NORM_STD = 127.5

# Create the metadata writer.
writer = ObjectDetectorWriter.create_for_inference(
    writer_utils.load_file(_MODEL_PATH), [_INPUT_NORM_MEAN], [_INPUT_NORM_STD],
    [_LABEL_FILE])

# Verify the metadata generated by metadata writer.
print(writer.get_metadata_json())

# Populate the metadata into the model.
writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
```

Ilustración 24 Script para generar metadatos de un modelo

Este script utiliza una librería desarollada por TensorFlow que genera automáticamente los metadatos, sin embargo, al utilizar esa herramienta con el modelo generado sigue sin obtenerse ningún resultado positivo pues se genera un nuevo error:

```
Traceback (most recent call last):
  File "script.py", line 26, in <module>
    writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
  File "C:\Users\JOSE MARIA BERZAL GO\anaconda3\envs\yolov4-cpu\lib\site-packages\tensorflow_lite_support\metadata\python\metadata_writers\metadata_writer.py
", line 162, in populate
    populator.load_metadata_buffer(self._metadata_buffer)
  File "C:\Users\JOSE MARIA BERZAL GO\anaconda3\envs\yolov4-cpu\lib\site-packages\tensorflow_lite_support\metadata\python\metadata.py", line 302, in load_met
adata_buffer
    self._validate_metadata(metadata_buf)
  File "C:\Users\JOSE MARIA BERZAL GO\anaconda3\envs\yolov4-cpu\lib\site-packages\tensorflow_lite_support\metadata\python\metadata.py", line 642, in _validat
e_metadata
    num_output_meta))
ValueError: The number of output tensors (2) should match the number of output tensor metadata (4)
```

Ilustración 25 Error tensores de salida no coincide con tensores de salida de los metadatos

Este error significa que el modelo entrenado no cuenta con los tensores de salida especificados en los metadatos, los cuales deben ser 4, mientras que en el modelo únicamente se aprecian 2.

El número de tensores necesario se encuentra especificado en la documentación y son como indica el script, 4 en lugar de 2. Sin embargo, no existe ninguna opción dentro del framework para elegir ese número de tensores, con lo cual no hay manera de elegir el número de tensores.

Esta opción se ha probado con múltiples modelos subidos al repositorio de modelos de TensorFlowLite y hay varios de ellos que presentan el mismo problema y tampoco se pueden implementar dentro del framework TensorFlowLite para Android Studio, vienen sin metadatos y al tratar de generarlos se obtiene el mismo código de error.

Se han probado dichos modelos con la APP de ejemplo creada por TensorFlow y también se ha presentado el mismo problema, descartando así cualquier problema relacionado con la APP.

Por último, se han probado los diversos modelos directamente en TensorFlowLite desde Google Colab y se han podido ejecutar sin problemas, esto es debido a que la única variante de TensorFlowLite con este problema es la versión desarollada para Android.

Ilustración 26 Prueba de detección con el modelo entrenado en Google Colab

Tras intentar aplicar numerosas vías alternas para tratar de solucionar el problema descrito anteriormente se llega a la conclusión de que hay una incompatibilidad entre Darknet y TFLite, generando el primero modelos que no necesariamente tienen 4 tensores de salida, lo que causa problemas con TFLite para Android.

Para solucionar este problema se ha optado por que los 3 modelos seleccionados para el análisis sean 3 modelos completamente distintos del repositorio de modelos para detección de objetos de TensorFlow y se han descargado sus versiones TensorFlowLite con metadatos.

Los 3 modelos que se han escogido tienen una cantidad de clases bastante amplia, y esto es debido a que un número de clases elevado incrementa las posibilidades que debe comprobar el modelo entrenado, produciendo una disminución en el rendimiento. Además, aumenta en gran medida la facilidad del modelo para producir una salida errónea, al disponer de objetos dentro del dataset que pueden parecer similares para el modelo.

Para este proyecto se han escogido los siguientes modelos:

- Modelo1 “SSDMobilenetV1”:

Mobilenet se trata de una red neuronal que se encuentra optimizada para las aplicaciones de visión dentro de los dispositivos móviles, como pueden ser la detección de objetos o la clasificación de imágenes. Estas redes neuronales dosponen de dos hiper-parámetros los cuales permitirán reemplazar latencia por precisión, adaptándose de esta manera a las necesidades particulares del uso que se le vaya a dar a la red neuronal.

Esta variabilidad de los parámetros persigue poder ser utilizadas en todo tipo de dispositivo, ajustando una menor latencia para dispositivos menos potentes (aunque disminuya la precisión del modelo).

- Modelo2 “SSDMobilenetV3”:

Gracias a incluir tanto MobilenetV3 como MobilenetV3 se podrán visualizar las diferencias de rendimiento entre ambos. Este modelo se encuentra orientado hacia CPUs de teléfonos móviles, indicando así que se debería obtener un rendimiento más que aceptable. Esto lo consigue mediante el uso de una arquitectura orientada al hardware, basada además en el algoritmo “NetAdapt”.

- Modelo3 “EfficientDet Lite3 V1”

Como tercer y último modelo se ha escogido EfficientDet Lite3 V1, este modelo es un modelo que no se encuentra optimizado para teléfonos móviles y es de carácter general. Esto permitirá confirmar si es necesario que los modelos se encuentren específicamente orientados a teléfonos móviles o se puede obtener un rendimiento aceptable con un modelo de carácter general.

Pruebas:

Las pruebas llevadas a cabo en esta iteración no conllevan mucha complicación, siendo necesario únicamente abrir la interfaz principal de la misma y probar uno a uno cada uno de los botones introducidos. Se observa que los 3 botones abren correctamente la Activity Camera, cada uno con su respectivo modelo.

Iteración 4:

Planteamiento:

El objetivo de la cuarta y última iteración es la comprobación del rendimiento de los diferentes modelos seleccionados. Con la finalidad de medir este rendimiento se utilizarán principalmente los FPS o Frames Per Second (frames por segundo).

Esta medida es definida como la cantidad de imágenes que el dispositivo es capaz de renderizar en cada segundo, la cual variará en función de la carga que esté soportando el dispositivo en ese momento.

Para que el ojo humano vea un vídeo con movimiento y no como una secuencia de imágenes superpuestas se necesita que el video esté siendo renderizado a una media de 12 FPS, medida mínima que se ha decidido considerar para indicar que un modelo es válido.

Desarrollo:

El primer paso para llevar a cabo para poder realizar la correcta medición de los FPS es crear una nueva clase llamada "FPS". Esta clase, bastante simple guardará únicamente el valor actual de los FPS, atributo público al que posteriormente se accederá desde la cámara, así como el nanosegundo en el cual se han calculado por última vez los FPS.

Para el cálculo de los FPS será necesario únicamente dividir $\frac{1s}{tiempo\_transcurrido}$, siendo tiempo_transcurrido el tiempo que ha pasado desde el instante en el que se calcularon la última vez los FPS. Este cálculo se lleva a cabo en el método logFrame(), en el cual se sustituirá el antiguo valor del atributo FPS por el recién calculado, reemplazando también el antiguo valor de la variable que guarda el nanosegundo en el cual se realizó por última vez el cálculo.

Una vez actualizado correctamente el valor de los FPS se incluye la llamada al método descrito previamente dentro del método analyze de la cámara, de tal manera que se refresque el valor de los FPS cada vez que el analizador reciba un frame nuevo, satisfaciendo así la definición de este valor.

Con la finalidad de poder hacer un seguimiento a tiempo real de este valor se ha incluido en la Activity Camera un TextView, elemento gráfico que mostrará un texto en la pantalla del dispositivo, en este caso el mensaje FPS: "numero_FPS".

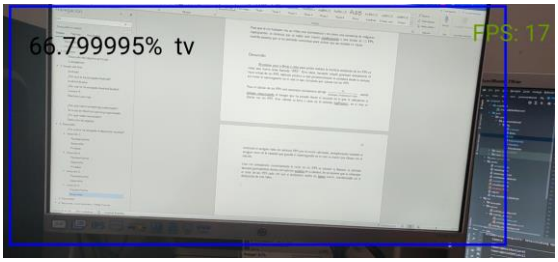

Ilustración 27 FPS mostrados durante la ejecución de la cámara

El siguiente paso consiste en la generación de un nuevo Activity para mostrar la información recogida de manera más visual, facilitando así su estudio. Para esto se ha utilizado la librería HelloCharts, la cual permite implementar de manera bastante sencilla un gráfico de líneas que muestre dichos valores.

Esta activity se llamará "Monitorizar" y contendrá la información de hasta 30 valores de FPS, se ha escogido este valor ya que considero que será suficiente para observar por si hubiera algún patrón en los FPS medidos.

La activity dispone de 3 TextView, los cuales indican tanto el valor máximo, el mínimo y la media en los valores de los FPS medidos. A su vez se incluirá un botón que permitirá llevar a cabo un borrado de este registro para poder volver a iniciar la medición desde cero. Además, se ha programado el layout de esta activity en horizontal en lugar de en vertical, consiguiendo así que se pueda visualizar una mayor parte del gráfico.

También se ha recogido desde el MainActiviy la información referente al modelo, ya que cada modelo tendrá sus mediciones de FPS por separado, a fin de probar independientemente cada uno de ellos.

El último paso a llevar a cabo para poder representar los FPS es pasar dicha información a la Activity "Monitorizar", para hacer esto se ha optado por el uso de serialización de los logs en ficheros de texto plano, permitiendo al usuario ver un registro de todos los logs que se han guardado hasta la fecha(pese a la gráfica solo mostrar los últimos 30).

Esta serialización se llevará a cabo en 3 ficheros separados, dependientes del modelo que se esté utilizando para analizar en ese momento y realizará un guardado cada vez que se llame a la función .logFrame() del objeto FPS.

La serialización tendrá lugar mediante la instancia de un FileWritter para abrir el flujo de datos con el fichero y un objeto PrintWritter que se encargará de escribir lo que se le indique dentro del fichero, añadiendo además un método cerrarfichero() que se activará cuando se utilice la flecha de navegación "Back" dentro de la interfaz Camera, cerrando el fichero y guardando todos los cambios realizados en el mismo.

El formato que se ha seleccionado para llevar a cabo esa serialización consta de incluir el timestamp completo del momento que se loguea esa información, seguido del carácter "," y por último del valor de los FPS en ese momento. Gracias a esto se podrá de serializar fácilmente la información, utilizando el carácter "," como separador y presentarlo más fácilmente en el gráfico.

Cabe destacar que se ha decidido guardar únicamente los frames cada 20 segundos, no obteniendo un flujo excesivo de logs y sin tener unos tiempos de espera exagerados entre el guardado de un log y el siguiente, para lo cual se ha añadido un atributo denominado "ultimoFrameGuardado", el cual como su nombre indica guarda el tiempo transcurrido desde la última vez que se guardó un frame en el fichero de texto asociado.

```
}
private long ultimoFrame = System.nanoTime();
//Se guardará para su posterior análisis un dato de FPS cada 1 minuto.
private long ultimoFrameGuardado = 0;
public Integer FPS = 0;

public void logFrame() {
    //Para calcular los FPS se calcula el tiempo transcurrido entre la última vez que se calcularon los FPS y el tiempo actual.
    long time = (System.nanoTime() - ultimoFrame);
    FPS = (int) (1000000000.0f/time);
    ultimoFrame = System.nanoTime();

    //Se guarda un registro cada 20 segundos de los FPS.
    if(Math.abs(System.currentTimeMillis() - ultimoFrameGuardado) >= 20000) {
        //Se loguea el nuevo registro en el fichero.
        String horaFormateada = Instant.now().toString().replace( oldChar: 'T', newChar: '/').split( regex: "Z")[0];
        pw.println(horaFormateada + "," + FPS);

        ultimoFrameGuardado = System.currentTimeMillis();
    }
}
//Para poder cerrar el fichero al cerrar la Activity.
public void cerrarfichero(){
    try {
        fw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Ilustración 28 Clase FPS tras implementar la serialización

A continuación, se procede con la definición de la deserialización de los datos pertenecientes a los ficheros de texto desde la Activity "Monitorizar", así como la declaración del contenido y las propiedades del gráfico, para poder representar los datos correctamente en la gráfica.

Para deserializar esta información se accede a el fichero de texto mediante un objeto File, a partir del cual se genera un FileReader para poder acceder a la información contenida en el mismo y posteriormente se utiliza el FileReader para instanciar un objeto ReadBuffer que, mediante un buffer permite representar la información obtenida dentro de la aplicación.

Esta información será tratada mediante el método extraerDatos(), el cual separará ambos valores(Timestamp e Integer que contiene el valor de los FPS) gracias al separador que se ha definido previamente, además se realizará un tratamiento más exhaustivo del valor Timestamp, dejando únicamente la fecha y la hora y eliminando el resto de datos del Timestamp. Tras su tratamiento, tanto las fechas y hora de cada una de las entradas como los Integers que contienen los FPS se almacenarán por separado en dos ArrayList definidos como atributos de la Activity.

El siguiente paso consiste en el tratamiento de esos datos para poder representarlos correctamente, aquí se diferencian dos flujos de código:

- Existen más de 30 medidas dentro del fichero:

  En este caso se generan dos nuevos ArrayList que contendrán los últimos 30 registros tanto de los Timestamps como de los valores de los FPS, estos se definirán mediante el método subList(). Este método permite definir sublistas a partir de una lista ya existente indicando el índice del primer elemento y del último.

Tras esto se preparan los datos iterando en un bucle for para que a partir de las listas de los 30 elementos se generen objetos PointValue(que guardarán la información referente a los FPS) y AxisValue(que dispondrán en el eje los timestamps que corresponden a cada FPS)

Este bucle además tendrá la funcionalidad de calcular tanto el máximo y el mínimo de los datos representados en la gráfica como la media de todos estos valores.

- Existen 30 medidas o menos dentro del fichero:

  Las únicas diferencias que se presentan con respecto a la situación anterior es que como no se disponen de más de 30 medidas (el máximo establecido para el gráfico) se iterará sobre el bucle un número de veces igual a la cantidad de valores incluidos en el fichero de texto, además de utilizar esta cantidad de valores para el cálculo de la media.

```
//Se guardaran los últimos 30 datos

int menorFPS = fps.get(0);
int mayorFPS = fps.get(0);
int valorMediaFPS = 0;

//Si es mayor que 30 se cogen los últimos 30 valores.
if(timestamps.size()>30)
{
    //Se crea un subarray que contenga esos ultimos 30 elementos y se utiliza este en lugar del completo.
    List<String> ultimosTimestamps = timestamps.subList(timestamps.size()-31,timestamps.size()-1);
    List<Integer> ultimosFPS = fps.subList(fps.size()-31,fps.size()-1);
    for(int i=0; i < 30; i++) {
        eje.add(new AxisValue(i).setLabel(ultimosTimestamps.get(i)));
        valores.add(new PointValue(i,ultimosFPS.get(i)));

        if(menorFPS > ultimosFPS.get(i)) menorFPS = ultimosFPS.get(i);
        if(mayorFPS < ultimosFPS.get(i)) mayorFPS = ultimosFPS.get(i);
        valorMediaFPS += ultimosFPS.get(i);

    }
    valorMediaFPS = valorMediaFPS/30;
}
else{
    for(int i=0; i < timestamps.size() ; i++) {
        eje.add(new AxisValue(i).setLabel(timestamps.get(i)));
        valores.add(new PointValue(i,fps.get(i)));

        if(menorFPS > fps.get(i)) menorFPS = fps.get(i);
        if(mayorFPS < fps.get(i)) mayorFPS = fps.get(i);
        valorMediaFPS += fps.get(i);

    }
    valorMediaFPS = valorMediaFPS/fps.size();

}

maxFPS.setText("FPS MAXIMOS: "+mayorFPS);
minFPS.setText("FPS MINIMOS: "+menorFPS);
mediaFPS.setText("MEDIA FPS: "+valorMediaFPS);
```

Ilustración 29 Deserialización de los valores de los FPS de los ficheros

El último requisito para poder mostrar los datos en la gráfica es su asignación al gráfico deseado, para lo cual es necesario definir tanto un Array de objetos Line como los propios Line que serán incluidos dentro del Array anteriormente mencionado. Estos objetos Line permiten entre otras cosas establecer el color de la línea o el tipo de línea deseada.

En este caso particular se ha decidido definir un objeto Line extra, aparte del que es necesario definir para mostrar los puntos, permitiendo así comprobar donde se sitúa la media de manera gráfica. Para esto se ha definido un objeto Line que únicamente contenga dos puntos, situando el primero al inicio del gráfico y el segundo al final, ambos teniendo como valor del eje Y la media. A este objeto Line no se le dibujarán los puntos ya que no aporta nada en específico.

Además, es necesario definir cada uno de los ejes, el ejeX se formará a partir de los valores de hora y minutos incluidos previamente en el Array eje, mientras que el eje Y se

deja vacío para que se autocomplete con los valores de las alturas de los puntos(estos se corresponden con los FPS alcanzados en cada punto).

Como último paso se crea un objeto LineChartData, el cual permite añadir una capa de parametrización al gráfico, añadiendo el Array de objetos Line dentro y por último se selecciona este LineChartData como el LineChartData que utilizará el gráfico.

```
maxFPS.setText("FPS MAXIMOS: "+mayorFPS);
minFPS.setText("FPS MINIMOS: "+menorFPS);
mediaFPS.setText("MEDIA FPS: "+valorMediaFPS);


//Se crea la linea que representará la media.
ArrayList<PointValue> valoresMedia = new ArrayList<>();
valoresMedia.add(new PointValue( x: 0,valorMediaFPS));
valoresMedia.add(new PointValue( x: fps.size()-1,valorMediaFPS));
Line media = new Line(valoresMedia).setHasLabels(false).setColor(Color.GREEN).setCubic(true).setHasPoints(false);

//Se define la linea que guardará los valores de los FPS.
Line linea = new Line(valores).setColor(Color.BLUE).setCubic(false).setHasLabels(true);

// Se añaden las lineas al Array de Lineas.
ArrayList<Line> lineas = new ArrayList<>();
lineas.add(linea);
lineas.add(media);
LineChartData datos = new LineChartData();
datos.setLines(lineas);

//Se crean los ejes, cambiando parámetros como el color o la disposición de los labels del eje.
datos.setAxisXBottom(new Axis(eje).setHasTiltedLabels(true).setMaxLabelChars(7).setName("Hora").setHasLines(true));
datos.setAxisYLeft(new Axis().setHasLines(true).setName("FPS"));



grafico.setLineChartData(datos);
```

Ilustración 30 Inclusión de los valores en LineChart

Tras acabar de implementar la funcionalidad principal de la iteración se han seguido otros objetivos más pequeños que permiten terminar de completar el funcionamiento de la aplicación:

- Se ha definido un botón de borrado en la Activity "Monitorizar", el cual presenta un AlertDialog para confirmar si se desea borrar o no el archivo del modelo sobre el cual se está viendo el grafico.

- Se han creado los botones "Resultados" en la Activity principal, que permiten acceder a la Activity "Monitorizacion", se han creado 3 botones, cada uno de los cuales permite acceder a los resultados de uno de los modelos.

- Se han realizado ajustes de la GUI en la Activity "Main", reposicionando los botones y TextView para que se vea correctamente si el dispositivo está tanto en horizontal como en vertical, eliminando también la ActionBar que incluía el nombre del proyecto.

Pruebas:

Tras finalizar por completo la aplicación se han realizado múltiples pruebas para asegurar el correcto funcionamiento de todos los componentes integrados, navegando por las Activities de esta y probando todas las funcionalidades.

La aplicación funciona a la perfección salvo por un bug producido por el borrado de los valores recogidos para los FPS, tras esto si se trata de volver a acceder a la Activity "Monitorizar" la aplicación provoca un crash al no poder leer del archivo en cuestión, ya que como se ha borrado no puede encontrarlo.

Para solucionar esto se ha establecido un listener en la Activity "Main" de tal forma que cada vez que el usuario llegue a esta interfaz (ya sea por iniciar la aplicación de nuevo o por el retorno de una de las otras dos Activities) comprobará de los tres ficheros de logging cual de ellos no existe y deshabilitará el botón asociado.

5. Resultados

Para poder llevar a cabo exitosamente este análisis se ha dejado ejecutar los modelos el tiempo suficiente para poder generar una gráfica completa con 30 puntos para cada uno de estos modelos.

- Modelo1:

Ilustración 31 Captura FPS y Gráfico FPS modelo1

- Modelo2:

Ilustración 32 Captura FPS y Gráfico FPS modelo2

- Modelo3:

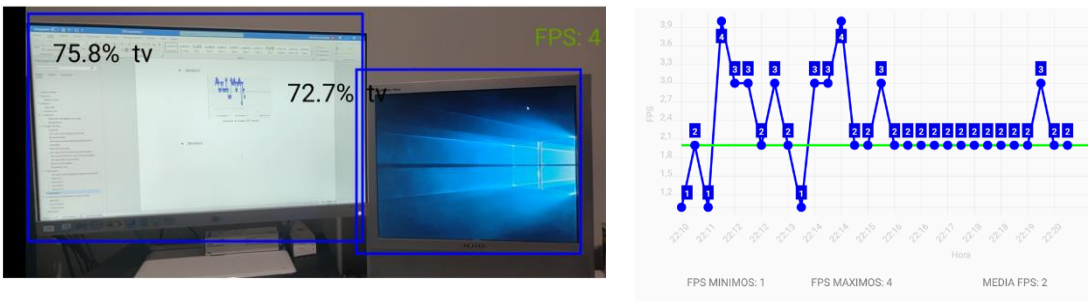

Ilustración 33 Captura FPS y Gráfico FPS modelo3

También se puede establecer una comparativa aproximada de los valores de confianza en función de la detección de un mismo objeto:

Tabla 1 Comparativa Tasas de Confianza y FPS en captura

	Pantalla 1	Pantalla 2	FPS
Modelo1	64,9%	67,6%	18
Modelo2	73,7%	72,8%	13
Modelo3	75,8%	72,7%	4

Se ha capturado un frame aleatorio y se han obtenido los resultados mostrados en la tabla superior, como se puede observar a mayor tasa de confianza del modelo se obtiene un menor número de FPS, logrando aun así tasas de confianza bastante elevadas en los 3 casos.

A su vez se han monitorizado los FPS durante el tiempo de uso de la aplicación y se han detectado los siguientes problemas:

- Se observan bajadas repentinas de FPS a 1 o 2.

  Este nivel tan bajo de puntos coincide con los instantes en los que se ha salido de la Activity y se ha vuelto a entrar. Provocando que tenga que volver a cargar los componentes y tarde más de lo debido en poder llamar al método logFrame() de la clase FPS la primera vez.

- Se observa un rendimiento muy bajo en el tercer modelo.

  Este modelo, conocido como EfficientNet Lite3, es un modelo que no se encuentra optimizado para dispositivos móviles y que consume una gran cantidad de recursos en la detección, sin embargo, partiendo de una misma posición alcanza más que el resto de los modelos en cuanto a probabilidad de pertenencia al grupo.

  Esto significa que pese a que EfficientNet Lite3 tiene una tasa de confianza bastante mayor no puede ser utilizado para la detección de objetos en tiempo real, sin embargo, si que podría utilizarse en otros casos, como la detección de objetos en imágenes ya tomadas.

  No obstante, si se trata de realizar un proceso de detección de objetos sobre una imagen resultaría mas fructífero llevarlo a cabo utilizando el ordenador, debido a que si no hay necesidad de que se realice en tiempo real se deberían aprovechar la gran superioridad de recursos de los ordenadores.

  Considero que un proceso de detección de objetos debe realizarse únicamente desde el teléfono móvil si hay necesidad de analizar ese proceso en tiempo real, en casos como los que se explicarán en líneas futuras.

Quitando estos dos puntos descritos anteriormente se obtiene un rendimiento constante que se mantiene a unos 16FPS de media(sin contar las bajadas a 1FPS) utilizando el modelo SSDMobileNet V1 mientras que SSDMobileNet V3, aunque también se mantiene estable disminuye la cantidad de FPS a una media de 11FPS.

Como se puede ver, estas cantidades son aceptables ya que en ambos casos se trata de modelos entrenados en COCO Dataset, este dataset contiene 80 clases diferentes de objetos que sabe identificar y clasificar. Esto, aparte de reducir la tasa de acierto tiene como consecuencia un decremento del rendimiento.

Considero que por tanto se ha obtenido un resultado positivo de este estudio, pues a partir de entre 10 y 12FPS el ojo no es capaz de distinguir los frames individuales que forman el video que se renderiza en la vista previa del teléfono. Además, no se ha observado un retardo excesivo en ninguno de los 3 casos en referencia a los tiempos necesarios para detectar los objetos que se han utilizado en las pruebas, detectando casi todos los objetos con los que se ha probado (para lo cual es necesario una confianza del 60% mínimo) salvo en casos muy concretos como que no se encontrase el objeto completamente dentro de la imagen o situaciones con poca iluminación.

Además, se ha comprobado utilizando el dataset entrenado con YoloV4-tiny que se obtiene una tasa de acierto mucho mayor cuanto menor sea el número de clases, siendo en el caso de la imagen de la lata de refresco un 100%.

Por lo cual, se puede concluir finalmente que la detección de objetos sobre dispositivos móviles es viable siempre y cuando se utilice un modelo cuyo rendimiento se encuentre optimizado para la detección de objetos en dispositivos móviles, obteniendo situaciones parecidas a las analizadas en los modelos 1 y 2 en lugar de lo sucedido con el modelo 3.

6. Resumen, Conclusiones y Líneas Futuras

Resumen

En este proyecto se han cumplido todos los objetivos buscados y se ha desarrollado una aplicación mediante el uso de Android Studio y frameworks como CameraX y TFLite capaz de cargar hasta 3 modelos de red neuronal y probar los mismos en tiempo real. La aplicación ha sido desarrollada en el lenguaje de programación Java.

Para este proyecto se han seleccionado 3 modelos de red neuronal previamente entrenados basados en distintos clasificadores y de esta forma comprobar el funcionamiento de manera más genérica.

La aplicación mencionada previamente es además capaz de capturar los frames por segundo del detector de objetos, permitiendo hacer un seguimiento del rendimiento de los diferentes clasificadores y de esta forma probar si existen modelos válidos para el uso de redes neuronales en dispositivos Android.

Conclusiones

En cuanto a conclusiones del proyecto, me ha resultado muy interesante aprender la tecnología Android mediante el uso de Android Studio. No me ha resultado excesivamente complejo, dado que se parece al lenguaje de programación "Java" con el cual en trabajado mucho durante la carrera. Sin embargo, aprender a utilizar Android Studio, así como su interfaz gráfica me ha gustado. Además, he aprendido a tratar redes neuronales mediante esta herramienta, lo cual también ha sido algo completamente nuevo para mí.

Respecto a las conclusiones técnicas, como se ha indicado más en detalle en los resultados se obtiene un rendimiento bastante aceptable con un dispositivo de gama media, siempre y cuando se utilice un modelo orientado a dispositivos móviles. Es un tipo de aplicaciones que no se encuentran lo suficientemente afianzadas en el mercado y en unos años se observarán grandes avances, permitiendo alcanzar los 60FPS utilizando datasets bastante amplios.

Con respecto a las conclusiones personales, el trabajo realizado me ha servido para crecer como informático. Me ha resultado muy interesante introducirme en las redes neuronales, ya que es un tema el cual llevaba llamándome la atención muchísimo tiempo. Además, he aprendido a investigar, dado que mi trabajo consta de mucha base de recopilación de información y extracción de conclusiones. Aprender sobre estos temas me ha permitido formarme en temas como las redes neuronales y la detección de objetos, los cuales había podido ver brevemente en la carrera, pero no había profundizado como me hubiera gustado.

También ha sido una experiencia complicada, he podido darme cuenta de lo costoso que es el desarrollo de un proyecto en el cuál solo hay un único miembro que debe encargarse

de todo el proceso. El trabajo de investigación, en un principio, me resultó complejo, dado que no sabía cómo extraer las conclusiones de aquello que leía, pero una vez aprendí cada vez me resultaba más sencillo.

He afrontado diversos problemas a lo largo del proyecto, muchos de ellos tienen que ver con la programación en Android y el Framework utilizado. Debido a estos problemas, la implementación del proyecto se ha ralentizado más de lo que me hubiera gustado y, por tanto, ha habido mejoras que no he podido implementar. Sin embargo, estos errores han servido para cultivar mi paciencia y para ayudarme a aprender a buscar información acerca de dichos errores.

Finalmente, cabe decir que el Trabajo de Fin de Grado que he realizado pone el broche unos estudios que me han gustado mucho cursar y de los cuales he aprendido muchísimo.

Líneas futuras

Este proyecto podría ser utilizado como base para el desarrollo de una aplicación Android que permitiera a los empleados de una fábrica, mediante el uso de un modelo personalizado únicamente enfocado para ello, distinguir entre cuáles de las piezas que tienen en stock se encuentran en buen estado y cuáles de ellas son defectuosas.

A su vez podría ser empleado por una empresa con inventario para poder realizar un inventariado de manera más efectiva, conectando este proyecto con una base de datos de tal forma que cada vez que se detecte un objeto, la aplicación incremente en una unidad la cantidad de objetos de ese tipo disponibles en el inventario.

Otro posible uso de este proyecto podría ser para apoyar a la gente con cierto tipo de discapacidad visual, en el caso del daltonismo se podría utilizar para ayudarles a distinguir los colores, indicando mediante texto qué color se está viendo en pantalla.

Estos solo son dos ejemplos de una gran cantidad de aplicaciones que se podrían crear después del estudio realizado acerca de la viabilidad de las redes neuronales en las aplicaciones móviles. Este tipo de aplicaciones serían muy útiles en muchos ámbitos de la vida cotidiana de muchísimas personas, así como en el ámbito industrial. Se ha podido observar que las aplicaciones con redes neuronales en los móviles serán cruciales en el futuro.

Bibliografía

[1] https://developer.android.com/training/camerax
[2] https://www.tensorflow.org/lite/api_docs
[3] https://www.iartificial.net/clasificacion-o-regresion/
[4] https://aprendeia.com/aprendizaje-no-supervisado-machine-learning/
[5] https://medium.com/datos-y-ciencia/aprendizaje-no-supervisado-en-machine-learning-agrupaci%C3%B3n-bb8f25813edc
[6] https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/
[7] https://www.iartificial.net/regresion-logistica-para-clasificacion/
[8]https://medium.com/datos-y-ciencia/machine-learning-supervisado-fundamentos-de-la-regresi%C3%B3n-lineal-bbcb07fe7fd
[9] https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/
[10] https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/
[11] https://developer.android.com/guide/topics/ui/look-and-feel/themes
[12] https://arxiv.org/abs/1704.04861
[13] https://tfhub.dev/s?deployment-format=lite
[14] https://www.ibm.com/docs/es/elm/6.0.3?topic=scenarios-iterative-development
[15] https://towardsdatascience.com/custom-object-detection-using-darknet-9779170faca2
[16] https://ruc.udc.es/dspace/handle/2183/24891

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá