FACULTAD DE INFORMÁTICA

UNIVERSIDAD NACIONAL DE LA PLATA

TÍTULO

EXPLORACIÓN DEL POSICIONAMIENTO INDOOR MEDIANTE EL RECONOCIMIENTO DE OBJETOS CON EL FIN DE CO-DISEÑAR Y CO-TESTEAR APLICACIONES MÓVILES SENSIBLES AL CONTEXTO

TESIS PRESENTADA PARA OBTENER EL GRADO DE

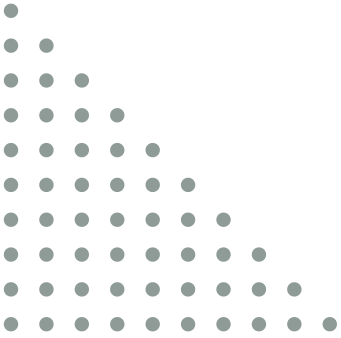
MAGISTER EN INTELIGENCIA DE DATOS ORIENTADA A BIG DATA

TESISTA

FRANCO MARTÍN BORRELLI [1]

DIRECTORA

DRA. CECILIA CHALLIOL



MARZO DE 2024

[1] BECARIO DE MAESTRÍA DE LA UNLP. LIFIA, FACULTAD DE INFORMÁTICA, UNLP

Índice

1. Introducción

1.1 Motivación

En los últimos años, la evolución de la tecnología móvil ha sido exponencial, transformando la forma en que interactuamos con el mundo que nos rodea y revolucionando la manera en que realizamos nuestras tareas diarias. Estos avances han sido impulsados por una combinación de innovaciones de hardware, software y conectividad, que han permitido el desarrollo de aplicaciones cada vez más poderosas y sofisticadas. Uno de los ámbitos más favorecidos por estos avances es el de las aplicaciones móviles sensibles al contexto [Alegre-Ibarra et al., 2018]; las cuales utilizan, por ejemplo, datos del entorno que rodea al usuario para brindar diversos tipos de servicios o información.

Las aplicaciones móviles sensibles al contexto vienen siendo objeto de investigación desde hace más de veinte años. Por ese entonces [Dey, 2000] sentaba las bases de este tipo de aplicaciones. Sin embargo, en esas primeras investigaciones, la tecnología no permitía hacer pruebas reales, sino más bien simuladas o montando laboratorios específicos. Fue recién en los últimos años que los avances tecnológicos han permitido poner en práctica estas aplicaciones en diferentes áreas de la Ciencia de la Computación [Augusto et al., 2017], como por ejemplo inteligencia artificial, hogares inteligentes, monitoreos médicos, asistencia al usuario y/o educación.

Un contexto relevante dentro de este tipo de aplicaciones es el posicionamiento, el cual se puede obtener utilizando distintas tecnologías como GPS, Wi-Fi o Bluetooth [Xiao et al., 2018]. Si bien este contexto viene siendo ampliamente utilizado en espacios abiertos (*outdoor*) gracias a las ventajas del GPS, todavía sigue siendo todo un interrogante cómo resolver el posicionamiento de los usuarios en espacios interiores (*indoor*) donde la señal del GPS no es precisa [Xiao et al., 2018]. Cabe mencionar que el autor de esta tesis estuvo explorando esta problemática utilizando distintos mecanismos de posicionamiento; por ejemplo, en [Borrelli et al., 2018] se investigó el uso de *Beacons*[1] mientras que en [Challiol et al., 2019] se usó *fingerprinting*[2] de redes Wi-Fi para posicionar al usuario. Es importante tener en cuenta que estas alternativas tienen una desventaja común: para funcionar, requieren de infraestructura adicional instalada (y muchas veces costosa) en el lugar. Esto motiva a seguir explorando este tipo de problemática con el foco puesto en esta tesis en el reconocimiento de objetos.

El reconocimiento de objetos, se refiere al procedimiento que busca identificar y categorizar objetos dentro de imágenes o videos mediante algoritmos de aprendizaje automático. Gracias a los avances en el aprendizaje automático, y en particular en el área de las *redes neuronales profundas* (o *deep neural networks)*, ahora los dispositivos móviles son capaces de ejecutar

[1] Un *"beacon"* es un dispositivo de hardware pequeño que emite señales de radio de corto alcance para transmitir información a dispositivos cercanos, como teléfonos móviles o tablets. Estos dispositivos se utilizan para determinar la ubicación de un objeto o persona en un entorno específico, como una tienda, un museo o un centro comercial. Los *beacon*s utilizan tecnologías como Bluetooth Low Energy (BLE) o Bluetooth 4.0 para enviar señales.

[2] El "*fingerprint*" de redes Wi-Fi es una colección única de señales y parámetros que caracterizan una red inalámbrica. Se utiliza para el posicionamiento indoor comparando las características de las redes Wi-Fi en un área para determinar la ubicación de un dispositivo. Es efectivo, pero requiere de una calibración precisa.

este tipo de algoritmos, abriendo un abanico de oportunidades en áreas muy diversas. Por ejemplo, para el área de la medicina y la salud se describe en [Taqi et al., 2019] un modelo entrenado para ayudar a dermatólogos y pacientes a identificar fácilmente el cáncer de piel. Así mismo, asociado al ámbito de la educación, en [Ozdemir and Kunduraci, 2022] se presenta una aplicación capaz de reconocer en tiempo real distintos tipos de insectos y poder así clasificarlos de acuerdo a sus características. En el ámbito del entretenimiento, el ejemplo más conocido podría ser quizás *Pokemon GO* [LeBlanc and Chaput, 2017], que utiliza la cámara para reconocer objetos del ambiente combinados con elementos de realidad aumentada.

Como se menciona en [Xiao et al., 2018], un uso interesante para el reconocimiento de objetos es el que se conoce como *vision-based indoor positioning*, que busca usar el reconocimiento de objetos para orientar y ubicar al usuario dentro de espacios indoor. Esta alternativa de posicionamiento indoor, en comparación a las otras mencionadas anteriormente, presenta una ventaja interesante: no necesita infraestructura adicional instalada en el lugar y solo requiere de un *Smartphone*, un dispositivo que todo el mundo posee hoy en día.

De acuerdo a [Xiao et al., 2018], el *vision-based indoor positioning* se puede lograr de tres formas:

- La primera se conoce como "*referencia en base a objetos*" y se centra en la detección de objetos estáticos en imágenes, para luego comparar estos objetos con una base de datos de edificios (muchas veces representadas como modelos 3D), las cuales contienen información de la posición de estos objetos dentro del edificio.
- La segunda alternativa se conoce como "*referencia en base a imágenes*" y plantea el uso de imágenes previamente tomadas y etiquetas de rutas específicas del edificio. Estas imágenes son utilizadas luego para comparar con la vista actual del dispositivo y así posicionar al usuario. Utilizando esta estrategia, no se identifican objetos particulares, sino ambientes específicos en lugares particulares.
- Por último, el uso de "*referencia de objetivos codificados*" plantea utilizar objetos tales como códigos de barra, QR y patrones de punto. Esta última forma es la más sencilla, ya que a diferencia de las otras dos no requiere reconocimiento previo del entorno. Cabe mencionar que esta tercera forma ya ha sido explorada en [Challiol et al., 2019], donde se utilizaron códigos QR como mecanismo de posicionamiento.

Este trabajo de tesis se centra en explorar la primera alternativa, "*referencia en base a objetos*", ya que es más flexible al permitir detectar objetos genéricos en diferentes posiciones independientemente del espacio físico donde se utilice.

En [Xu et al., 2020] se presenta otro punto interesante a analizar al abrir la interrogante de cómo se deberían ejecutar estos modelos de detección mencionados. En el enfoque tradicional, la ejecución de estos modelos de reconocimiento se vinculaba a centros de datos y grandes clústeres de máquinas con potentes GPU. Sin embargo, esta opción puede resultar costosa y además requiere transferir todos los datos del dispositivo y enviarlos a través de una conexión de red, lo cual puede demorar mucho tiempo. Gracias a los avances tecnológicos, hoy en día es posible ejecutar modelos localmente en dispositivos móviles, minimizando la latencia y mejorando la privacidad al evitar transferencias de datos por red. No obstante, implementar algoritmos eficientes en los dispositivos móviles, considerando restricciones de recursos y demandas de precisión en tiempo real, sigue siendo todo un

desafío, como se discute en [Xu et al., 2020], [Cai et al., 2021] y [Zou et al., 2023].

El posicionamiento basado en el reconocimiento de objetos podría utilizarse tanto en aplicaciones móviles como en herramientas que asistan al co-diseño in-situ de estas aplicaciones. En esta tesis, estas herramientas van a ser el marco para la exploración de éste tipo de posicionamiento.

De acuerdo a [Sanders and Stappers, 2008], el término "co-diseño" se utiliza para definir una actividad colaborativa en la cual un grupo de individuos participa en un proceso creativo para desarrollar conjuntamente un producto o servicio. En el marco del co-diseño, se utiliza el término "*in situ*" para referirse a las actividades, observaciones o estudios que se realizan en el entorno real donde ocurren los fenómenos que se están estudiando o para los cuales se están diseñando soluciones.

Es importante mencionar que el co-diseño in-situ (asistido por herramientas de creación) todavía está en una etapa temprana de investigación, existiendo muy pocos trabajos que aborden este tipo de diseño, más aún en espacios indoor. El co-diseño in-situ requiere recorrer el espacio físico e identificar los lugares relevantes donde luego se brindará información o servicios. Si bien esto consume tiempo, se logra aprovechar mejor las características del espacio físico y se tiene una visión más real de las vivencias de los usuarios para los cuales se está diseñando.

En [Hargood et al., 2018] se propone una herramienta para co-diseñar in-situ narrativas posicionadas pero en espacios outdoor usando el GPS. Los autores destacan que co-diseñar in-situ es enriquecedor porque permite que se consideren las distintas visiones de las personas involucradas en dicho proceso. Sin embargo, es complejo ponerlo en práctica, ya que implica un consenso entre todos los participantes.

Cabe mencionar que el autor de esta tesis estuvo participando en la creación de una herramienta de soporte para el co-diseño in-situ de juegos móviles basados en posicionamiento, en particular para espacios indoor. La herramienta junto a algunos casos de estudio fueron presentados en [Challiol et al., 2019] y [Challiol et al., 2020]. Además, participó en la modificación de esta herramienta para dar soporte al co-diseño distribuido de este tipo de juegos, esta modificación junto a algunas experiencias de uso fueron publicadas en [Borrelli et al., 2021] y [Borrelli et al., 2022]. Contar con esta base sobre la temática de co-diseño ha motivado el objetivo de esta tesis.

En relación al testeo de aplicaciones móviles sensibles al contexto, en [Luo et al., 2020] se realizó un estado del arte sobre distintas herramientas que pueden ser usadas para llevar a cabo el mismo. Sin embargo, estas herramientas relevadas están diseñadas para ser utilizadas por usuarios expertos. Es decir, el testeo de este tipo de aplicaciones por parte de usuarios no expertos está muy poco explorado. Por otro lado, en [Sarker, 2021] se menciona también cómo la recolección de datos y el análisis de datos al utilizar las aplicaciones puede mejorar el resultado final a partir de una mejor toma de decisiones. Es en esta dirección que se explora el co-testing (o testeo colaborativo) en esta tesis.

Por lo antes mencionado, se han podido apreciar las bases conceptuales del autor de esta tesis, las cuales han motivado el tema de investigación que explora este trabajo.

1.2 Objetivos

El objetivo general de este trabajo es explorar la utilización de modelos de reconocimiento de objetos como mecanismo para lograr el posicionamiento de usuarios en espacios indoor (cerrados) mediante el uso de dispositivos móviles. Esta exploración se enmarca dentro de un abordaje dirigido a usuarios no expertos, con el propósito de facilitar el co-diseño y el co-testeo in-situ de aplicaciones móviles sensibles al contexto.

En base al objetivo general se abordan los siguientes objetivos específicos:

- Investigar el estado actual de la tecnología de reconocimiento de objetos, sus aplicaciones en relación con el posicionamiento indoor y los dispositivos móviles. Esto incluye un análisis de las metodologías y algoritmos disponibles, así como la evaluación de sus ventajas y limitaciones.
- Explorar la utilización de modelos de reconocimiento de objetos existentes y la posibilidad de crear un modelo de reconocimiento propio (capaz de reconocer objetos específicos). Se busca desarrollar aplicaciones prototípicas con el objetivo de evaluar el funcionamiento de estos modelos y la viabilidad de su utilización para el posicionamiento indoor. En particular, se investiga y se hacen pruebas con el framework *Tensorflow* y sus versiones *Tensorflow Lite* (orientadas a espacios móviles y de edge computing) y *Tensorflow.js* (orientada a navegadores web).
- Diseñar y desarrollar una solución que permita posicionar indoor (a los usuarios) utilizando modelos de reconocimiento de objetos. Esto implica definir criterios de selección de objetos de referencia, establecer técnicas de captura y análisis de imágenes, y diseñar o utilizar algoritmos existentes para lograr una correspondencia efectiva entre los objetos detectados y la posición del usuario.
- Analizar cómo utilizar la solución de posicionamiento indoor propuesta en el marco de un abordaje dirigido a usuarios no expertos, con el propósito de facilitar el co-diseño y el co-testeo in-situ de aplicaciones móviles sensibles al contexto. Es decir, que usuarios no expertos puedan crear este tipo de aplicaciones.
- Tomando en consideración el análisis anterior, diseñar y desarrollar una herramienta prototípica que incorpore modelos de reconocimiento de objetos para el posicionamiento indoor. Esta herramienta debe estar especialmente diseñada para usuarios no expertos, con una interfaz que permita el co-diseño y co-testeo de aplicaciones móviles sensibles al contexto.
- Realizar pruebas sistemáticas de la herramienta propuesta para evaluar su eficacia en relación con el posicionamiento indoor. Esto va a permitir analizar la viabilidad de usar este tipo de posicionamiento en el marco de este tipo de aplicaciones. Para lograr esto, se crean distintas experiencias de co-diseño y co-testeo in-situ con la herramienta implementada.
- Recopilar métricas de uso con la herramienta para luego analizarlas y poder generar información relevante para que se pueda utilizar en el proceso de co-testeo de las aplicaciones. Mediante el uso de técnicas de análisis, como minería de datos y visualización, se busca identificar patrones y tendencias en estos datos que puedan ayudar a una mejor toma de decisiones.

1.3 Estructura de la tesis

A continuación, se describen los capítulos de la presente tesis.

El Capítulo 2 realiza un análisis detallado sobre cómo ha evolucionado el reconocimiento de objetos a lo largo del tiempo. Se revisan las técnicas tradicionales de *machine learning* y se explora cómo el campo ha avanzado con la adopción de *Redes Neuronales Convolucionales* (CNN) en *deep learning*. Se examinan arquitecturas específicas de CNN. Además, se profundiza en el concepto de "*vision-based indoor positioning*".

El Capítulo 3 aborda *TensorFlow* en relación con la implementación de modelos de reconocimiento y detección de objetos en dispositivos móviles. Se proporciona una visión general de *TensorFlow* y se analizan las características que lo hacen adecuado para su uso en dispositivos móviles, como *TensorFlow Lite* y *TensorFlow.js*. Se discuten las ventajas y limitaciones de utilizar *TensorFlow* en el contexto de aplicaciones móviles sensibles al contexto. Se entra en detalle sobre el proceso de entrenamiento de nuevos modelos.

El Capítulo 4 presenta pruebas realizadas de aplicaciones prototípicas en dispositivos móviles. Se emplean tanto modelos de reconocimiento y de detección de objetos pre-entrenados provistos por *TensorFlow*, como así también modelos propios entrenados para reconocer objetos específicos. Se exploran dos enfoques para la implementación en dispositivos móviles utilizando *TensorFlow Lite* y *TensorFlow.js*. Se analizan los resultados obtenidos para determinar la viabilidad y eficacia de estos enfoques en aplicaciones móviles.

El Capítulo 5 propone una arquitectura genérica que utiliza modelos de reconocimiento y detección de objetos para lograr el posicionamiento de los usuarios en espacios indoor. Se describe la arquitectura en detalle, incluyendo los componentes principales y la integración con sistemas de posicionamiento existentes. Además, se presenta una implementación concreta de esta arquitectura.

El Capítulo 6 presenta el diseño de una herramienta de autor para el co-diseño y co-testeo de aplicaciones móviles sensibles al contexto, la cual usa la implementación del *posicionamiento por objetos* en espacios indoor propuesta en el Capítulo 5. Se explican las características y funcionalidades de esta herramienta en relación con el *posicionamiento por objetos.*

El Capítulo 7 presenta experiencias de co-diseño y co-testeo utilizando la herramienta propuesta en el capítulo anterior. Se analizan los resultados obtenidos en términos del *posicionamiento por objetos* y se discuten las implicaciones prácticas y potenciales mejoras.

El Capítulo 8 presenta las conclusiones en relación con la tesis, como así también algunos trabajos futuros que se desprenden de la misma.

2. Reconocimiento de objetos

En este capítulo se introduce el concepto de reconocimiento de objetos y sus posibles aplicaciones. Particularmente, se analiza la aplicación de modelos de *deep learning* (aprendizaje profundo) para el reconocimiento de objetos, examinando algunos de los algoritmos más relevantes que han surgido a lo largo de los años. Asimismo, se profundiza en algunos de los datasets disponibles, que desempeñan un papel crucial en el entrenamiento y evaluación de modelos de reconocimiento de objetos.

Por otro lado, acorde al objetivo de esta tesis, se aborda el tema del reconocimiento de objetos para posicionar a un usuario. Específicamente se explora el concepto *vision-based indoor positioning*.

2.1 ¿Qué es el reconocimiento de objetos?

El *reconocimiento de objetos* busca darle a los sistemas informáticos la habilidad de identificar y clasificar objetos en imágenes y videos de manera automatizada y precisa [Zaidi et al., 2022]. Cuando los seres humanos miramos una fotografía o un video, podemos detectar fácilmente personas, objetos, escenas y detalles visuales. El objetivo de la visión por computadora es enseñarle a la computadora a hacer lo mismo. Es decir, lograr un nivel de comprensión de lo que contiene una imagen o video.

Actualmente, la base del *reconocimiento de objetos* radica en la implementación de algoritmos de *machine learning* y *deep learning* y el uso de técnicas de procesamiento de imágenes [Zou et al., 2023]. Estos algoritmos se entrenan para reconocer patrones y características específicas en las imágenes, permitiendo así identificar objetos y asignarles etiquetas de clase correspondientes. La *precisión de la clasificación*[3] (es decir la capacidad de asignar correctamente etiquetas de clase a objetos en imágenes), y la *velocidad de procesamiento* (que determina la eficiencia en la ejecución de esta tarea) son dos métricas esenciales que miden el desempeño de los sistemas de *reconocimiento de objetos*.

El *reconocimiento de objetos* hoy en día cuenta con una amplia gama de aplicaciones en diversas industrias. Desde la seguridad en automóviles autónomos, donde se utiliza para detectar peatones, vehículos y señales de tráfico, hasta la vigilancia y seguridad, donde identifica comportamientos sospechosos en tiempo real [Choudhari et al., 2021]. En el ámbito de la medicina, se emplea, por ejemplo, en la detección de tumores y anomalías en imágenes médicas [Taqi et al., 2019] y para el seguimiento de pacientes [Al-Azzoa et al., 2018]. Además, la realidad aumentada lo aprovecha para superponer objetos virtuales en el mundo real [LeBlanc and Chaput, 2017]. En aplicaciones de redes sociales, se utiliza para etiquetar y clasificar automáticamente objetos y personas en imágenes y videos [Mehta et al., 2018]. También juega un papel crucial en la accesibilidad, asistiendo a personas con discapacidades visuales [Salunkhe et al., 2021].

El *reconocimiento de objetos* puede clasificarse en dos enfoques principales de acuerdo a lo indicado en [Wei et al., 2014]:

[3] Formalmente se define la "*clasificación"* como el proceso que busca especificar a cuál de las k categorías posibles pertenece una entrada x. Esto se describe como la generación de una función f: $R^n \rightarrow \{1,\ldots,k\}$. La salida podría ser la clase predicha y/o un vector Y con la distribución de probabilidad de todas las k clases.

- Por un lado se encuentra el *reconocimiento de clase única*. Básicamente se centra en identificar y clasificar una sola clase de objeto específica en una imagen o video. En esencia, cuando se presenta una imagen como entrada, el sistema evalúa su contenido y emite una etiqueta (o *label*), que indica la categoría a la que pertenece el objeto identificado. Este enfoque es valioso en situaciones donde se necesita determinar la presencia o ausencia de un objeto particular en una imagen, y su correspondiente clasificación. Los problemas clásicos de clasificación de imágenes se incluyen en esta categoría.
- Por otro lado se encuentra el *reconocimiento de objetos de multiclase*. Se enfoca en reconocer y clasificar todas las clases de objetos presentes en una imagen o video en lugar de centrarse en una sola categoría. En otras palabras, su objetivo es identificar y etiquetar todos los objetos de interés en la escena visual. Este enfoque es especialmente valioso cuando se trata de situaciones más complejas en las que pueden coexistir varios objetos de diferentes clases en una misma imagen. Resulta fundamental en aplicaciones como las de seguridad de tránsito, donde es esencial, por ejemplo, identificar tanto vehículos como peatones, señales de tráfico y obstáculos.

El *reconocimiento de objetos* tiene otros conceptos relacionados que son importantes mencionar, los cuales enriquecen la comprensión y manipulación de la información visual [Hafiz and Bhat, 2020]. Estos son:

- *La localización*: se ocupa de establecer la posición espacial precisa de los objetos reconocidos, proporcionando coordenadas que indican su ubicación en la escena.
- *La detección*: va más allá de solo reconocer la presencia de objetos, sino que también busca delimitarlos y ubicarlos en la imagen o video. Generalmente esto se hace mediante cuadros delimitadores (*bounding boxes*), aunque existen otras técnicas.
- *La segmentación de objetos*: logra la asignación de etiquetas a cada píxel de la imagen, permitiendo una comprensión detallada de la estructura y forma de los objetos detectados. Esta segmentación puede ser *semántica* o por *instancia*:
    - El objetivo de la *segmentación semántica* es obtener una inferencia detallada al predecir etiquetas para cada píxel de la imagen. Cada píxel se etiqueta según el objeto o región en la que está contenido.
    - Siguiendo esta dirección, la *segmentación de instancias* proporciona etiquetas diferentes para instancias separadas de objetos pertenecientes a la misma clase de objeto. Por ese motivo, la *segmentación de instancias* puede definirse como la tarea de encontrar una solución simultánea para la detección de objetos y la *segmentación semántica*.

En la Figura 2.1 se observan los posibles resultados que se podrían obtener de una imagen si se realiza el *reconocimiento de objetos* (Figura 2.1.a), la *detección de objetos* (Figura 2.1.b*)*, una *segmentación semántica* (Figura 2.1.c) o *por instancia* (Figura 2.1.d).

En relación a la segmentación, al comparar las Figuras 2.1.c y Figuras 2.1.d se puede apreciar cómo la *segmentación* semántica asigna colores distintos a las lapiceras (*pen*), mientras la *segmentación por instancias* identifica que es el mismo objeto, y le asigna el mismo color (verde).

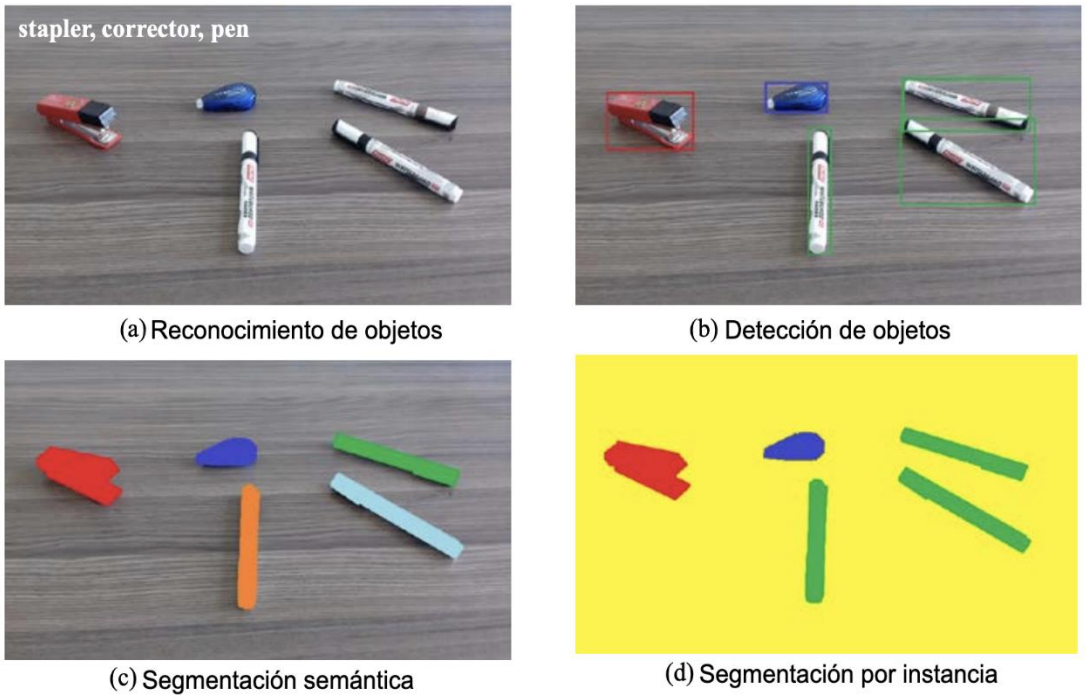


Figura 2.1: Comparación de resultados entre *reconocimiento*, *detección* y *segmentación.* Figura adaptada y traducida de [Hafiz and Bhat, 2020].

Si bien en los últimos años han existido avances continuos en esta área, el *reconocimiento de objetos* todavía presenta una serie de desafíos [Zaidi et al., 2022]:

- La inferencia de objetos puede verse considerablemente complicada debido a lo que se conoce en la literatura como "*Variación intra-clase*" o "*Intra-class variation*". Este concepto se refiere a la variabilidad que puede existir dentro de cada clase de objetos. Una baja *variación intra-clase* implica que los miembros de una clase son bastante similares entre sí, mientras que una alta *variación intra-clase* indica que hay una mayor diversidad o diferencia entre los miembros de la misma clase. Estas variaciones no se limitan solamente a características físicas de los objetos sino también a las condiciones de captura de las imágenes. Esto incluye factores como cambios en la iluminación, la presencia de objetos parcialmente ocultos, etc. Es importante tener en cuenta que los objetos que se intentan detectar podrían presentar deformaciones, estar en diferentes orientaciones, escalados o incluso podrían estar borrosos.
- Además, el auge en la disponibilidad de datos visuales en forma de imágenes y videos a gran escala requiere el desarrollo de técnicas de procesamiento eficientes que puedan lidiar con esta proliferación de información, y además mantener una alta precisión.

2.2 Enfoques para el reconocimiento de objetos

En las últimas dos décadas el campo del reconocimiento y la detección de objetos ha experimentado una evolución notable [Zaidi et al., 2022] marcada por dos períodos históricos significativos: el "*período de detección de objetos tradicional*" (antes de 2014) y el "*período de detección basado en aprendizaje profundo*" (después de 2014).

Durante el primer período, que abarcó los años anteriores a 2014, las técnicas y enfoques

para la detección y el reconocimiento de objetos se basaban principalmente en métodos clásicos de procesamiento de imágenes y aprendizaje automático convencional (*Machine Learning*) [Zaidi et al. 2022]. Los algoritmos se centran en la extracción de características manuales de las imágenes, como bordes, texturas y colores, y luego utilizando clasificadores para identificar objetos en base a estas características. A pesar de que estos enfoques lograron avances notables en la tarea, enfrentan desafíos significativos en la capacidad para lidiar con la variabilidad en la apariencia de los objetos y la adaptación a nuevas clases de objetos.

Sin embargo, el panorama cambió de manera drástica con la llegada del aprendizaje profundo (*deep learning*), y en particular, con las *redes neuronales convolucionales* [Zaidi et al. 2022]. Una *red neuronal convolucional* (CNN) es un tipo especializado de red neuronal diseñada para procesar datos bidimensionales, como imágenes. Su característica distintiva está en la capa de *convolución*, que escanea la entrada recibida para aprender patrones y características. Estas redes demostraron una capacidad sin precedentes para aprender representaciones de alto nivel directamente de los datos, lo que permitió la extracción automática de características relevantes para la identificación de objetos. Esto condujo a una mejora significativa en la precisión y la robustez de los sistemas de detección y reconocimiento de objetos.

En la Figura 2.2 se puede observar una representación de cómo funcionan ambos enfoques descriptos anteriormente.

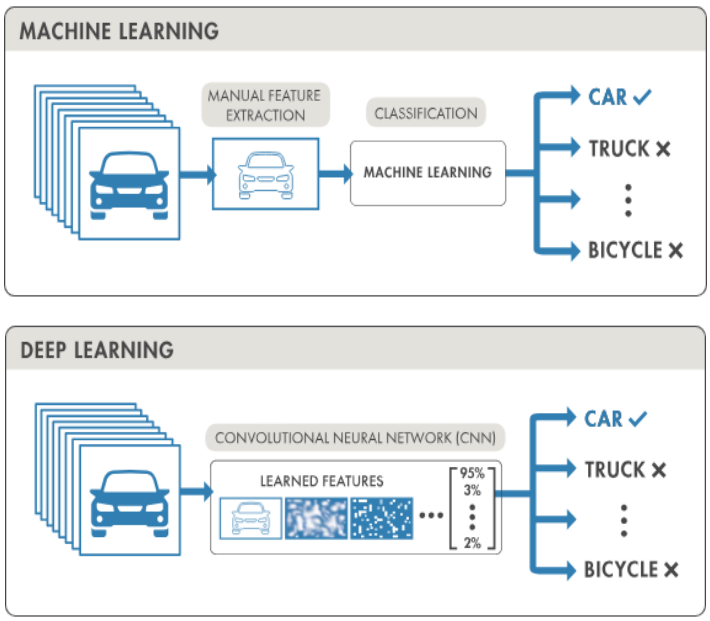


Figura 2.2: Diferencias entre reconocimiento de objetos usando *machine learning* y *deep learning*[4].

En muchos casos, las técnicas de *machine learning* pueden resultar adecuadas, especialmente, si se conoce de antemano cuáles son las características de interés de las imágenes a procesar.

[4] Imágenes extraídas de la página "*Introductoria al procesamiento de imagen y a visión de computadoras de Mathworks*".
https://www.mathworks.com/solutions/image-video-processing/object-recognition.html (último acceso: 3/09/2023).

Por otro lado, la consideración principal a tener en cuenta al elegir entre *machine learning* y *deep learning* es considerar si se cuenta con una GPU potente y un *dataset* grande para el entrenamiento. Si la respuesta a alguna de estas preguntas es no, un enfoque de *machine learning* podría ser la mejor opción. En general, las técnicas de *deep learning* tienden a funcionar mejor con más imágenes, y una GPU ayuda a disminuir el tiempo necesario para entrenar el modelo.

2.2.1 Enfoques basados en *Deep Learning*

Como se menciona en la sección anterior, una de las técnicas de aprendizaje profundo o *deep learning* más populares para tareas de *detección y reconocimiento de objetos* son las redes neuronales convolucionales o *CNN (*de sus siglas en inglés, *Convolutional Neural Network)*. Estas redes buscan aprender automáticamente características propias de un objeto con el fin de poder identificarlo. Por ejemplo, una CNN puede aprender a identificar diferencias entre perros y gatos analizando miles de imágenes de entrenamiento y aprendiendo las características que hacen que los perros y los gatos sean diferentes.

El concepto de *redes neuronales convolucionales* fue introducido por primera vez en 1989. Originalmente estas redes demostraron ser efectivas para el reconocimiento de dígitos en imágenes, como se presenta en [LeCun et al., 1989]. Sin embargo, tras el éxito en esta tarea, el interés en las CNN aumentó, expandiendo su funcionamiento a tareas más desafiantes. Las CNN han demostrado resultados más efectivos que aquellos incluso realizados por humanos [He et al., 2015]. La idea subyacente de las CNN se basa en trabajos anteriores de *reconocimiento de patrones visuales*, donde se ha demostrado que es útil extraer y combinar características locales para obtener características abstractas de orden superior [LeCun et al., 1989].

La primera capa de la CNN, la capa *convolucional*, puede emplear uno o varios filtros de dimensiones *Y* por *Y*, donde *Y* es un valor entero. Estos filtros se desplazan a lo largo de la imagen de entrada original, realizando una multiplicación elemento por elemento entre los valores de los píxeles del filtro y la subregión correspondiente de la imagen original. El resultado de esta operación de *convolución* se suma para obtener un valor único y este proceso se repite para cada posición posible del filtro en la imagen original. Este proceso se repite en toda la imagen, moviendo el filtro píxel por píxel. Al final, se obtiene uno o más mapas de características que representan diferentes aspectos de la imagen original. Cada punto en estos mapas es como una versión resumida de la información en un área específica de la imagen original. Así es como la *red neuronal* va identificando patrones y características importantes en las imágenes.

Un ejemplo de *convolución* puede apreciarse en la Figura 2.3, donde se observa cómo la subparte de la entrada original se multiplica, elemento por elemento, con los valores del filtro (kernel) para construir una nueva salida.

Luego de la capa *convolucional*, suele encontrarse la capa de *activación*, que aplica comúnmente la función ReLU[5] (*Rectified Linear Unit*) para introducir no linealidad a la red. Esta capa ayuda a la red a aprender representaciones más complejas y a capturar patrones más abstractos.

[5] ReLU es una función de activación común en redes neuronales. Su operación es simple: si la entrada es positiva, devuelve la entrada; si es negativa, devuelve cero.

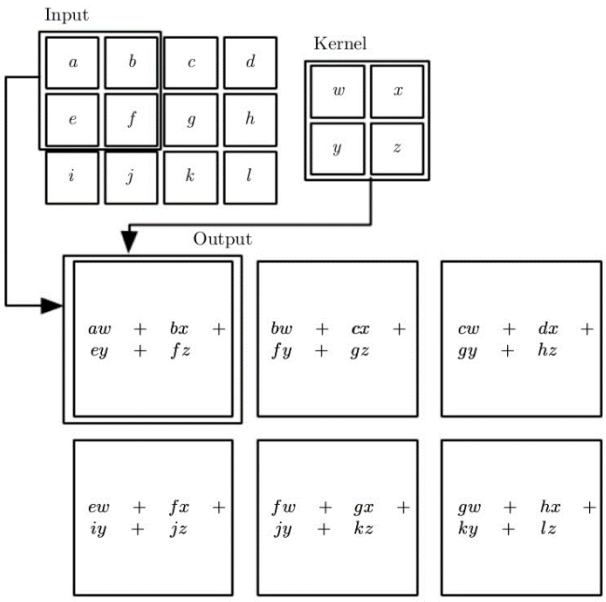


Figura 2.3: Ejemplo de una *convolución* 2D en una CNN [Goodfellow et al., 2016].

Posteriormente, las capas *de agrupación* (*pooling layers*) se insertan para reducir la dimensión espacial de las características extraídas, disminuyendo así la cantidad de parámetros en la red y previniendo el sobreajuste[6]. La técnica comúnmente utilizada (en la capa de agrupación) es el *max pooling*, que selecciona el valor máximo dentro de un área específica, como se muestra en la Figura 2.4. En el ejemplo de la figura, los datos se reducen de 4x4 a 2x2.

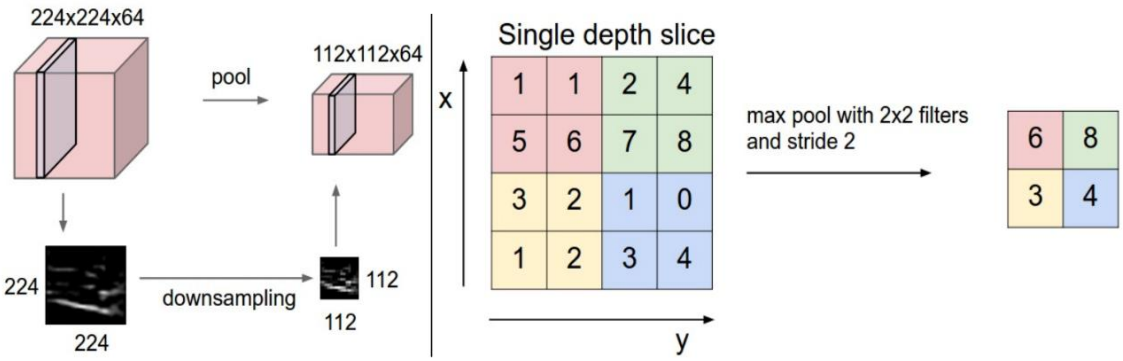


Figura 2.4: Operación de *Max Pooling* en una CNN[7].

Luego de las capas *de agrupación*, las CNN suelen contar con una o varias capas densas (*dense layers*) o también llamadas capas completamente conectadas. Estas capas establecen conexiones entre cada nodo de la capa anterior y todos los nodos de la capa actual. La función principal de estas capas es realizar operaciones de combinación y aprendizaje de características más complejas basadas en la información extraída por las capas *convolucionales* y de *agrupación*. En términos formales, las *capas densas* pueden conceptualizarse como una red de nodos interconectados, donde cada nodo representa una unidad de procesamiento y está conectado con todas las unidades en la capa anterior. Esta

[6] El sobreajuste acontece cuando se entrena demasiado una red o cuando se utilizan datos anómalos.

[7] Imagen extraída de la página "*CS231n Convolutional Neural Networks for Visual Recognition*". http://cs231n.github.io/convolutional-networks (último acceso: 3/09/2023).

conectividad total implica que cada nodo en la *capa densa* recibe información de cada nodo en la capa anterior. Además, cada conexión entre nodos tiene un peso asociado, determinando la ponderación de la información transmitida de un nodo a otro. Después de recibir información de todas las conexiones, se lleva a cabo una operación matemática, específicamente una combinación lineal, para calcular nueva información.

Posteriormente, se aplica una *función de activación* para determinar la relevancia de la información recién calculada en relación con la tarea específica para la cual la red está diseñada. Este proceso de combinación lineal seguido de una *función de activación* ayuda a la red a aprender representaciones más abstractas y complejas de los datos. Dependiendo del problema a resolver (clasificación, detección, etc.), la *función de activación* puede variar (*softmax para clasificación*, *sigmoid para problemas binarios*, etc.). Finalmente, en la capa *de salida* se produce la predicción final.

En la Figura 2.5 se puede observar un ejemplo puntual de una CNN donde se pueden apreciar las distintas capas y cómo las mismas están conectadas entre sí. La red contiene capas *convolucionales* y capas *de agrupación*, así como capas *densas* seguidas de una capa *softmax para la clasificación de imágenes* [Krizhevsky et al., 2012].

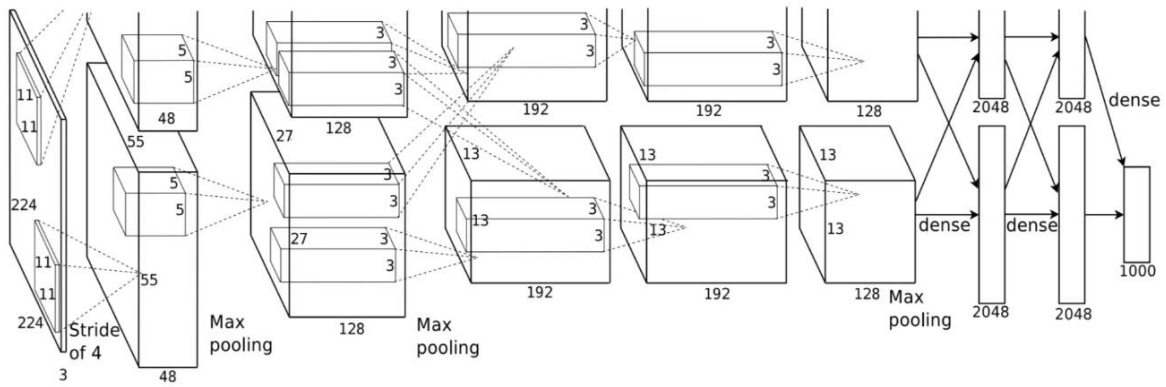


Figura 2.5: Arquitectura de ejemplo de una red CNN [Krizhevsky et al., 2012].

Las CNN se construyen utilizando múltiples capas *convolucionales* y *de agrupación*. A medida que se avanza en las capas de la red, la CNN aprende a distinguir patrones cada vez más complejos. Las capas iniciales se especializan en identificar líneas y formas simples (por ejemplo, formas geométricas simples), mientras que las capas subsiguientes se vuelven más complejas, siguiendo naturalmente la no linealidad entre capas. A medida que se avanza hacia las capas finales de la CNN, se construyen patrones aún más complejos; esto subraya la importancia de la profundidad en una CNN, ya que la ausencia de esta profundidad impediría a la red distinguir y clasificar patrones e imágenes más complejas. En la Figura 2.6 puede apreciarse cómo, a medida que se va avanzando en las capas, la red es capaz de identificar características cada vez más específicas sobre los datos.

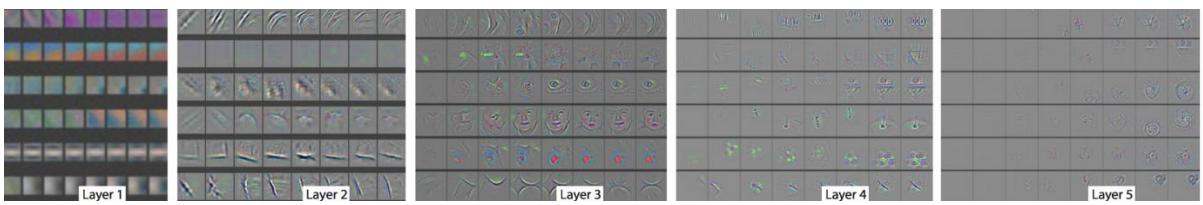


Figura 2.6: Evolución de las capas de una CNN [Zeiler and Fergus, 2014].

El surgimiento de las CNN marcó un hito en la forma de abordar el reconocimiento y la detección de objetos. En los estudios llevados a cabo por [Cai et al., 2021], [Zaidi et al., 2022] y [Zou et al., 2023] se presentan distintos análisis sobre la evolución y mejoras que han surgido en este campo a lo largo del tiempo. Estos autores utilizan el concepto de "*detector"* para hablar de las diferentes soluciones arquitecturales que buscan lograr la *detección de objetos* mediante el uso de CNN.

Estos "*detectores*" pueden clasificarse en dos categorías en función de la estrategia de detección que utilizan [Zaidi et al., 2022; Zou et al., 2023]:

- Por un lado, se encuentran los detectores de dos etapas (*two-stage detectors*). Como se puede deducir por su nombre, este tipo de detectores dividen la detección en dos etapas: primero extraen la *Región de Interés* y luego realizan las tareas de clasificación y regresión de cuadros delimitadores basadas en estas regiones. En términos más sencillos, estos modelos buscan proponer áreas que podrían contener objetos en la imagen y luego clasificarlos y localizarlos en la segunda. Como estos sistemas tienen dos pasos separados, generalmente tardan más en generar propuestas y tienen una arquitectura complicada, aunque suelen tener resultados mucho más precisos.
- Por el otro, se encuentran los detectores de una etapa (*single-stage o one-stage detector*). Estos detectores realizan la detección y clasificación en un único paso. Dividen la imagen en una cuadrícula y asignan cajas delimitadoras, y clases directamente a cada celda de la cuadrícula.

  Dentro de la clasificación de los detectores *one-stage*, en [Zaidi et al., 2022] y [Zou et al., 2023] se plantea la existencia de una sub-rama de investigación llamada *light-weight models*; la cual se viene explorando desde hace muy pocos años. Estas investigaciones tienen como objetivo diseñar redes pequeñas y eficientes para entornos con recursos limitados como celulares, tablets u otros dispositivos del Internet de las cosas (IoT).

La elección entre un tipo de detector u otro puede depender de los requisitos específicos de la aplicación. Los detectores de una etapa (*one-stage*) son preferibles en situaciones donde la velocidad es crucial, como en aplicaciones en tiempo real; mientras que los detectores de dos etapas (*two-stage*) son más adecuados cuando la precisión es prioritaria, aunque a costa de una mayor carga computacional. En las subsecciones siguientes, se aborda con mayor profundidad cada uno de estos tipos de detectores, destacando aquellos más significativos de acuerdo a [Zaidi et al., 2022] y [Zou et al., 2023].

2.2.1.1 Detectores de dos etapas (two-stage detectors) [Zaidi et al., 2022], [Zou et al., 2023]

Uno de los detectores más representativos dentro de esta categoría es *Region-based Convolutional Neural Network (o RCNN)*. Este detector demostró cómo las *CNN* pueden usarse para mejorar el rendimiento de detección. La idea detrás de RCNN es simple: primero, divide la imagen en pequeñas "*Regiones de Interés*" (ROIs). Estas son áreas en la imagen que podrían contener un objeto candidato. Luego, para cada ROI, se extraen características utilizando una CNN; la cual ha sido entrenada previamente con una gran cantidad de datos para aprender a reconocer patrones visuales en las imágenes, como bordes, texturas y formas. Finalmente, los clasificadores SVM (*Support Vector Machine*) lineales se utilizan para

predecir la presencia de un objeto dentro de cada región y para reconocer categorías de objetos. El *BB Regressor (Bounding boxes Regressor)* sirve para predecir las coordenadas de los cuadros delimitadores de un objeto. En la Figura 2.7 se presenta una representación gráfica del proceso de detección usando RCNN.

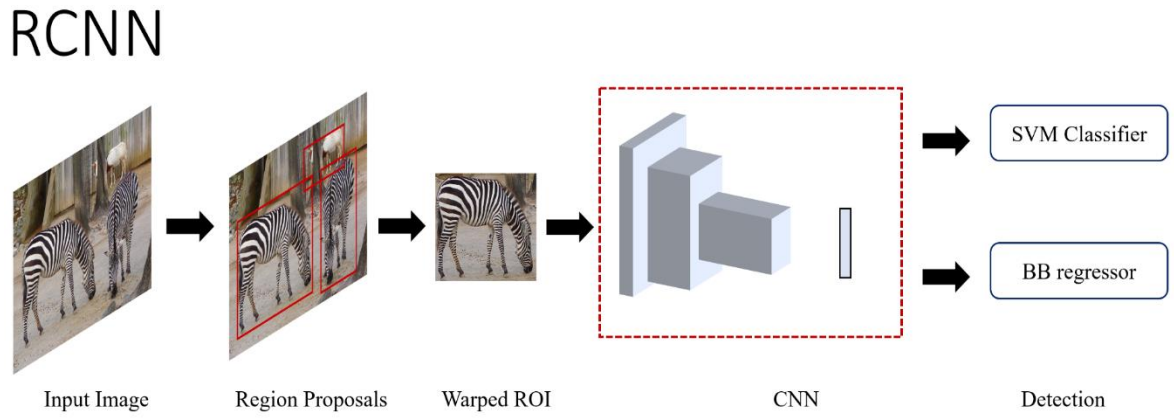


Figura 2.7: Ilustración de la arquitectura interna de RCNN *(2014)* [Zaidi et al., 2022].

Una de las principales desventajas de RCNN radica en su lento entrenamiento y evaluación, ya que procesar cada región propuesta de manera individual requiere mucho tiempo. Además, RCNN depende de algoritmos externos, como *Selective Search*, para la generación de propuestas de regiones, introduciendo complejidad adicional. Otra limitación es la falta de eficiencia en el uso de recursos computacionales debido a la incapacidad de compartir características aprendidas entre regiones propuestas.

Estas limitaciones motivaron el desarrollo de variantes más avanzadas, como *Fast RCNN* y *Faster RCNN*. A continuación se describe cómo cada uno las resuelve:

- *Fast RCNN* logra abordar las limitaciones computacionales de RCNN al proponer una arquitectura más eficiente que mejora significativamente la velocidad de entrenamiento y predicción. En lugar de procesar cada región por separado, utiliza una única CNN para extraer características de la imagen completa y luego genera regiones de interés. Estas regiones se extraen de la salida de la CNN y se utilizan para la clasificación y regresión de cuadros delimitadores. Este proceso puede observarse de forma gráfica en la Figura 2.8.

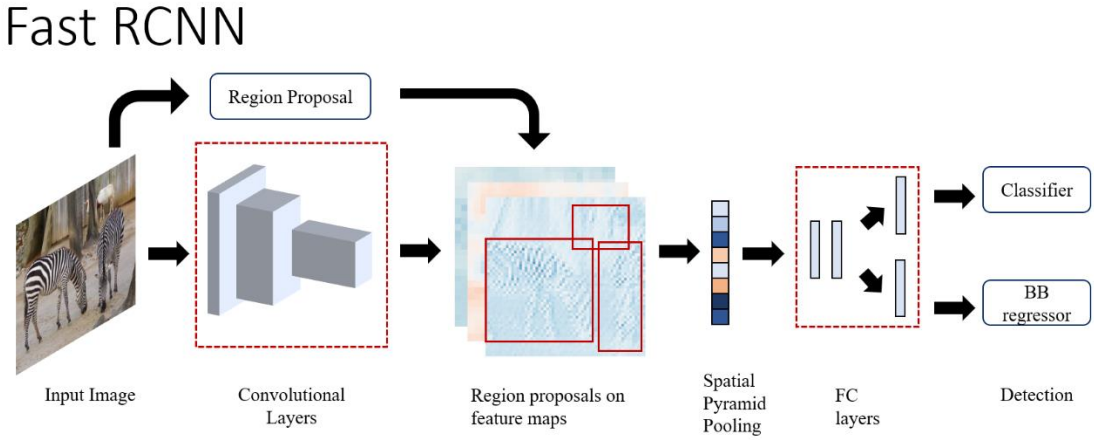


Figura 2.8. Ilustración de la arquitectura interna de *Fast-RCNN (2015)* [Zaidi et al., 2022].

- *Faster RCNN* aborda eficientemente el proceso de detección de objetos al integrar un componente clave llamado *Region Proposal Network* (o RPN) directamente en su arquitectura. Es decir, lleva la eficiencia un paso más allá que *Fast RCNN*.

  La RPN genera automáticamente regiones de interés sin la necesidad de algoritmos externos (como *Selective Search*). Esta mejora hizo que el proceso de detección sea más rápido y preciso. De acuerdo a [Zou et al., 2023], *Faster RCNN* es el primer detector de *deep learning* capaz de lograr detecciones casi en tiempo real. En la Figura 2.9 puede observarse cómo se integra la RPN a la arquitectura que había sido propuesta previamente por *Fast RCNN*.

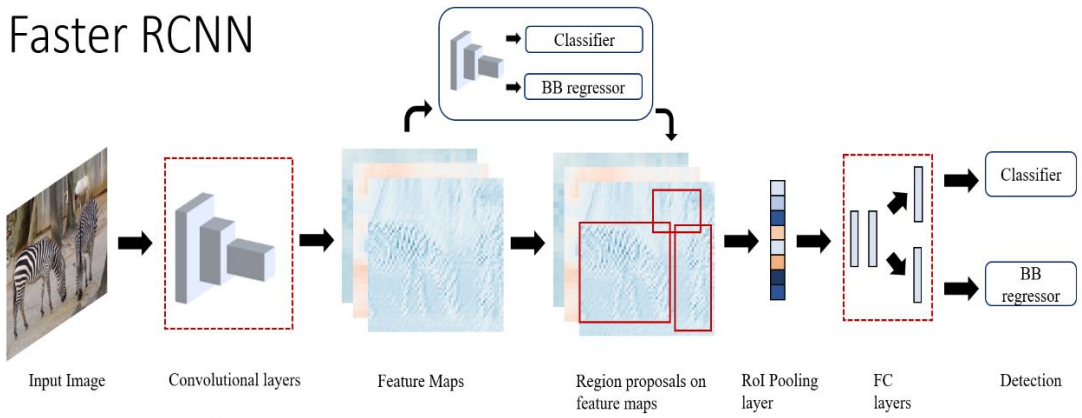


Figura 2.9: Ilustración de la arquitectura interna de *Faster RCNN (2015)* [Zaidi et al., 2022].

Existen otros detectores de dos etapas, pero esa temática queda fuera del alcance de esta tesis. Solo a modo de mención se pueden listar: *Spatial Pyramid Pooling Networks* (*SPP-Net*), *Feature Pyramid Networks* (FPN) y *Region-based Fully Convolutional Network* (*R-FCN*).

2.2.1.2 Detectores de una etapa (*one-stage detectors*)

Como ya se mencionó anteriormente, los detectores de una sola etapa (*one-stage*) clasifican y localizan objetos semánticos en un solo paso utilizando un muestreo denso [Zaidi et al., 2022]. Para esto, emplean cuadros/puntos claves predefinidos de varias escalas y relaciones de aspecto para localizar objetos.

Algunos de los detectores *one-stage* más importantes de acuerdo a [Zou et al., 2023] y [Zaidi et al., 2022] son: *You Only Look Once (YOLO)*, *Single Shot MultiBox Detector (SSD)*, *RetinaNet*, *CenterNet* y *EfficientDet*. A continuación, se detalla cada uno de estos.

- *You Only Look Once (YOLO)*

  YOLO fue el primer detector *one-stage* de la era de *deep learning*. La red YOLO divide las imágenes en una cuadrícula con celdas de tamaño GxG, y luego la cuadrícula genera N predicciones para cajas delimitadoras (en total, GxGxN cajas). Cada caja delimitadora se limita a tener solo una clase durante el momento de la predicción, lo que restringe la capacidad de la red para detectar objetos más pequeños. Aunque esto fue revisado y tratado en versiones posteriores. YOLO unifica la tarea de detección de objetos y el enmarcado de los objetos detectados, ya que la ubicación espacial de las cajas delimitadoras se trata como un problema de regresión.

La Figura 2.10 proporciona una representación gráfica del proceso de ejecución de YOLO. En la imagen izquierda, se observa cómo la arquitectura de YOLO divide la imagen de entrada en una cuadrícula. Esta cuadrícula se utiliza para evaluar múltiples ventanas de la imagen, como se muestra en la imagen del medio. Cada celda de la cuadrícula se encarga de predecir las cajas delimitadoras y las probabilidades de clase para los objetos contenidos en esa región específica. Finalmente, en la imagen derecha, se presentan las detecciones realizadas por YOLO en la imagen original, donde las cajas delimitadoras resaltan la ubicación y extensión de los objetos detectados.

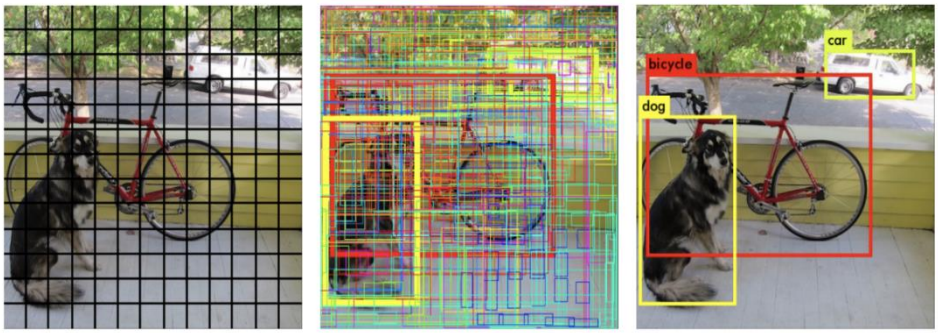


Figura 2.10: Ejemplificación del funcionamiento de YOLO [Redmon and Farhadi, 2017].

En la Figura 2.11 puede apreciarse un diagrama donde se muestra de forma muy general cómo funciona internamente YOLO. En la figura se observa como una imagen de entrada es pasada a una CNN, la cual se encarga de extraer características de la misma. La capa de *convolución final* (*Convultional Layer*), destacada en la figura, se encarga de producir las predicciones finales para la detección de objetos en la imagen de entrada. Esta capa actúa como un clasificador para asignar probabilidades a cada clase y calcular las puntuaciones de confianza asociadas (*Classifier*) y como un regresor para predecir las coordenadas de los cuadros delimitadores (*BB Regressor - Bounding boxes Regressor*).

A pesar de su gran mejora en la velocidad de detección, YOLO en su versión original, experimenta una disminución en la precisión de la localización en comparación con detectores de dos etapas, especialmente para algunos objetos pequeños. Esto fue revisado en las siguientes versiones. La versión más reciente (a la fecha de la escritura de esta tesis) es YOLO *V7* [Wang et al., 2022]; esta versión supera a la mayoría de los detectores de objetos existentes en términos de velocidad y precisión (rango de 5 FPS a 160 FPS) mediante la introducción de estructuras optimizadas como la asignación dinámica de etiquetas y la re-parametrización de la estructura del modelo.

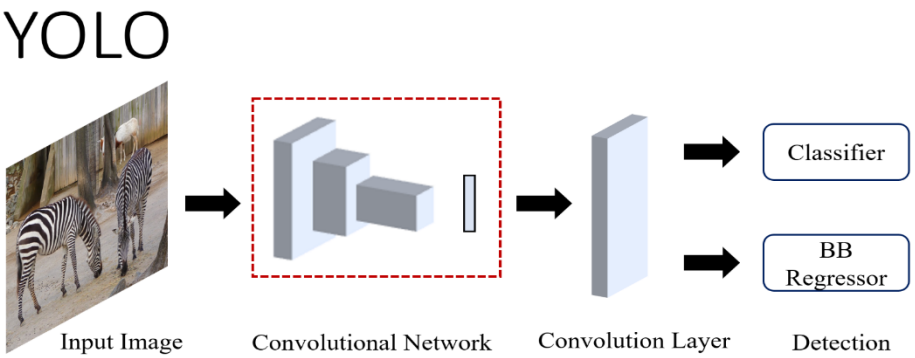


Figura 2.11: Ilustración simplificada de la arquitectura de *YOLO (2015)* [Zaidi et al., 2022].

- *Single Shot MultiBox Detector (SSD)*

  SSD fue el primer detector *one-stage* en igualar la precisión de los detectores de dos etapas como *Faster RCNN*, manteniendo una velocidad de respuesta en tiempo real. Mientras que la red *Faster RCNN* ejecuta detecciones lentamente (a alrededor de 7 *FPS*) la red SSD utilizando el mismo hardware, es capaz de ejecutarse a 59 *FPS*. Esto es debido a la eliminación de la necesidad de volver a muestrear píxeles o características, lo que redujo el número de cálculos por detección, manteniendo un alto rendimiento. Como se puede apreciar en la Figura 2.12, *SSD* utiliza una red convolucional para extraer características fundamentales de la imagen. Incorpora capas de detección en diferentes escalas, permitiendo la identificación de objetos de diversos tamaños. Cada una de estas capas realiza predicciones simultáneas de clasificación y regresión, abordando tanto la categorización como la ubicación precisa de los objetos en una misma iteración.

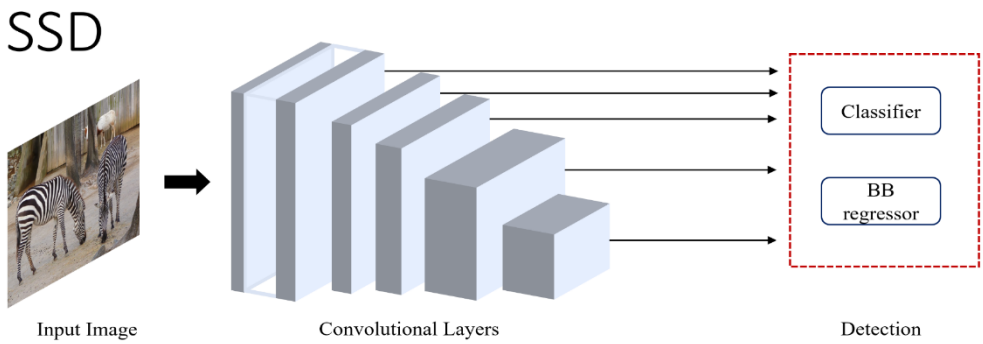


Figura 2.12: Ilustración simplificada de la arquitectura del detector *SSD* [Zaidi et al., 2022].

- *RetinaNet*

  La fortaleza de *RetinaNet* radica en poder identificar tanto objetos pequeños como grandes con mayor precisión. Utiliza una estructura llamada pirámide de características o *FPN* [Lin et al., 2017] de sus siglas en inglés (*Feature Pyramid Network*) para adaptarse a diferentes tamaños de objetos y una técnica llamada *Focal Loss* para mejorar la detección de objetos menos comunes.

  En la Figura 2.13 puede observarse cómo funciona internamente *RetinaNet*.

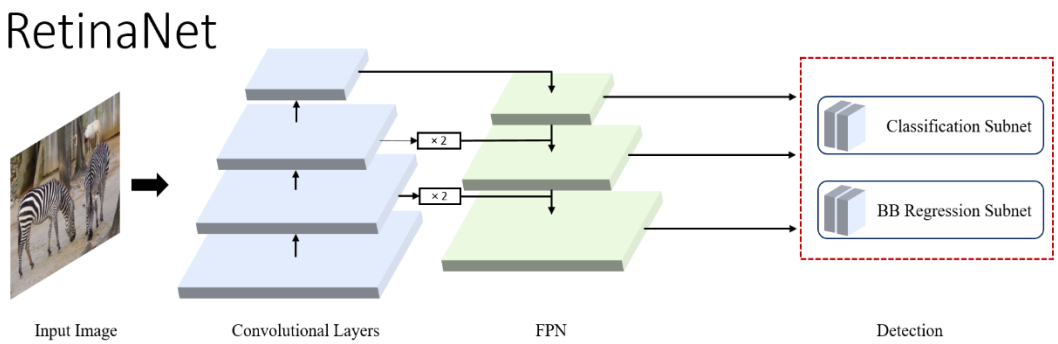


Figura 2.13: Ilustración de la arquitectura de RetinaNet *(2018)* [Zaidi et al., 2022].

En primer lugar, utiliza una CNN para extraer características esenciales de la imagen. Luego, incorpora una *FPN*, que combina información de diferentes niveles de la red convolucional para adaptarse a objetos de diversos tamaños. Luego, la red se ramifica en dos: por un lado la subred de clasificación y por otro la subred de regresión. Esto es una

característica interesante que la diferencia de *YOLO* y de *SSD*, donde estas tareas se hacían en la misma red. Mientras que la subred de clasificación se encarga de predecir las probabilidades de pertenencia a diferentes clases para cada región propuesta por la *FPN*, la subred de regresión realiza predicciones para ajustar las coordenadas de las cajas delimitadoras propuestas. Cada capa del *FPN* se pasa a estas subredes, lo que le permite detectar objetos en varias escalas.

- *CenterNet*

  Como se puede apreciar en la Figura 2.14, el proceso que realiza *CenterNet* comienza tomando una imagen como entrada que luego se procesa a través de una CNN. En particular, *CenterNet* a menudo utiliza una arquitectura basada en la CNN *Hourglass*[8] para extraer características claves de la imagen.

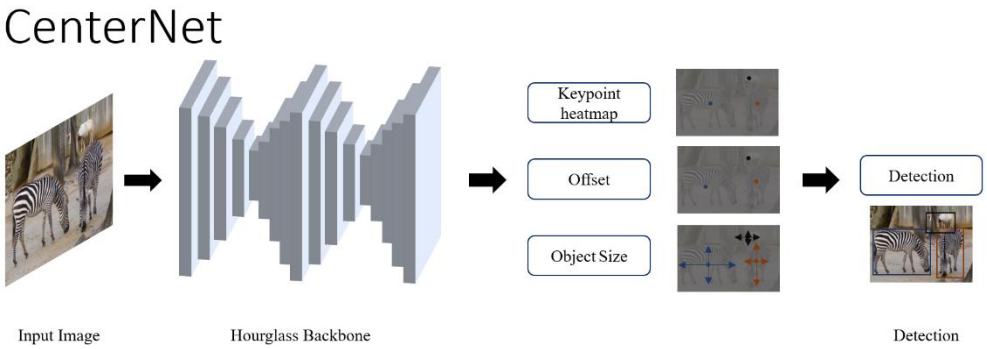


Figura 2.14: Ilustración de la arquitectura de CenterNet *(2019)* [Zaidi et al., 2022].

A partir de las características extraídas, se genera un mapa de calor que resalta los posibles centros de los objetos presentes en la imagen. Este mapa de calor se utiliza junto con múltiples "*cabezas*" de predicción, que son salidas específicas de la red neuronal. Estas *cabezas* desempeñan funciones especializadas: una identifica los centros potenciales de los objetos mediante un mapa de puntos claves (*keypoint heatmap*), otra estima las dimensiones de los objetos (*object size*) y una tercera corrige los desplazamientos (*offset*). Durante la fase de inferencia, la salida de la *cabeza* de desplazamiento se utiliza para precisar la ubicación exacta del objeto, generando finalmente un cuadro alrededor de la región detectada.

Como se puede notar de esta explicación, *CenterNet* adopta un enfoque muy diferente de los presentados hasta ahora. En lugar de realizar tareas para obtener la representación de cuadros delimitadores convencionales. Utiliza un modelado de objetos como puntos. *CenterNet* considera que un objeto es un punto único (el centro del objeto) y realiza la regresión de todos sus atributos (como tamaño, orientación, ubicación, pose, etc.) en función del punto central de referencia.

- *EfficientDet*

  La principal innovación de *EfficientDet* radica en su enfoque que permite encontrar un equilibrio entre la precisión y la eficiencia computacional en la detección de objetos. A medida que las arquitecturas de detección de objetos se volvían más complejas para mejorar la precisión, también aumentaron considerablemente los requisitos de potencia y

[8] La arquitectura *Hourglass* es conocida por su capacidad para realizar tareas que implican la detección de puntos clave o estructuras en una imagen, como las articulaciones en la detección de pose o los centros de objetos en la detección de objetos.

recursos computacionales. *EfficientDet* aborda este problema al utilizar una estrategia de escalado compuesta que ajusta el tamaño de la red y los parámetros relacionados de manera eficiente para lograr un mejor rendimiento con un menor costo computacional.

Como se puede observar en la Figura 2.15, la arquitectura *EfficientDet* combina diferentes componentes y técnicas, como la red neuronal convolucional para la extracción de características y un enfoque novedoso llamado *BiFPN* (*Bidirectional Feature Pyramid Network*) para fusionar y propagar información a través de múltiples escalas de la imagen. Esto permite que *EfficientDet* sea altamente escalable y eficiente en términos de uso de recursos.

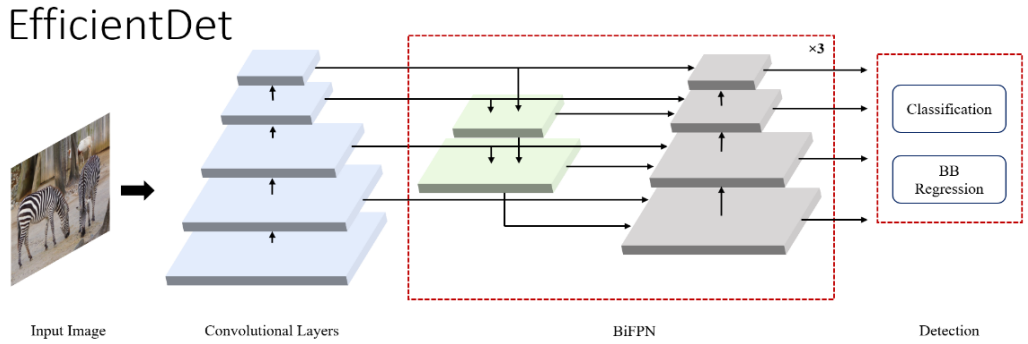


Figura 2.15: Ilustración simplificada de la arquitectura de EfficientDet *(2020)* [Zaidi et al., 2022].

2.2.1.3 Lightweight Networks

De acuerdo a [Zou et al., 2022] y [Zaidi et al., 2022], en los últimos años (2017 en adelante) ha aparecido una nueva rama de la investigación dentro del área de los detectores one-stage. El objetivo de estas investigaciones es diseñar redes pequeñas y eficientes para entornos con recursos limitados, como dispositivos móviles y otros dispositivos asociados a Internet de las Cosas (IoT). Este tipo de redes se conoce con el nombre de *lightweight networks*.

Generalmente, estas redes utilizan técnicas como *pruning* (podar), *quantization* (cuantificación) y *hashing* (mezclar) para mejorar la eficiencia de los modelos. Una característica común de todas estas redes es que sacrifican la precisión (aún más que los detectores de una etapa) ante una mayor velocidad en la detección.

Como se menciona en [Zaidi et al., 2022], existen varios detectores que hacen uso de las *lightweight networks* como *MobileNet*, *ShuffleNet* y *Once-For-All* (OFA). Estos modelos, conocidos como *lightweight models*, se detallan a continuación:

- *ShuffleNet*

  El objetivo principal de *ShuffleNet* es lograr una eficiencia extremadamente alta en términos de cálculos y parámetros sin comprometer el rendimiento. La idea clave detrás de *ShuffleNet* es la "*operación de reorganización*" o "*shuffle*", que permite mezclar las características extraídas de diferentes canales para introducir información cruzada de manera eficiente, y así mejorar la representación aprendida por la red. Esto es crucial para obtener buenos resultados de clasificación y detección, lo cual permite mejorar la representación de características sin usar demasiados recursos.

  ShuffleNet usa bloques *Shuffle* que dividen, convolucionan y reorganizan canales, como se puede apreciar en la Figura 2.16. A medida que la red avanza, usa capas de agrupación y convoluciones 1x1 para reducir la complejidad.

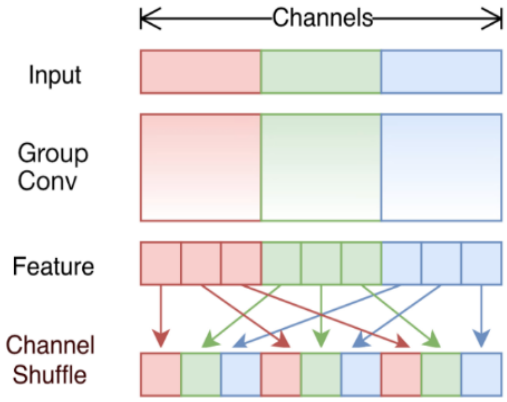


Figura 2.16: Funcionamiento de los canales en *ShuffleNet*[9].

- *MobileNet*

  Esta arquitectura de red surgió en 2017 y desde entonces ha experimentado varias mejoras significativas a lo largo de los años. La particularidad de *MobileNet* es que hace uso de *convoluciones separables* en lugar de convoluciones tradicionales. Las *convoluciones separables* dividen la convolución en dos etapas: una convolución en profundidad (1x1) seguida de una convolución espacial (3x3). Esto reduce significativamente la cantidad de cálculos necesarios, lo que hace que la red sea más rápida y eficiente. En la Figura 2.17 se presenta de manera gráfica la arquitectura de *MobileNet*, específicamente su versión 2.

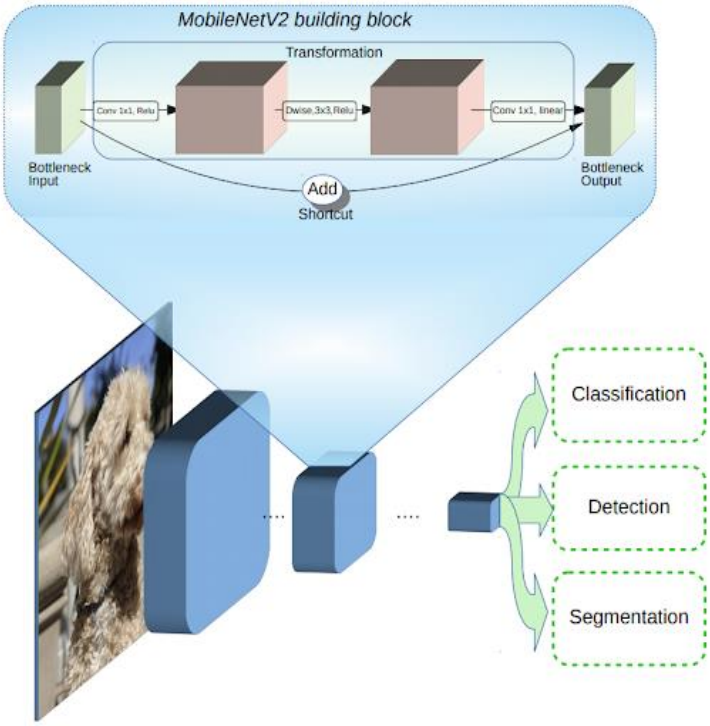


Figura 2.17: Ilustración simplificada de la arquitectura interna de *MobileNet-V2*[10].

[9] Imagen extraída de la página "*New mobile neural network architectures*". https://machinethink.net/blog/mobile-architectures (último acceso: 3/09/2023).

[10] Imagen extraída de la página "*MobileNetV2: The Next Generation of On-Device Computer Vision Networks*". https://blog.research.google/2018/04/mobilenetv2-next-generation-of-on.html (último acceso; 30/10/2023).

MobileNet también utiliza capas de agrupación (*pooling*) para reducir las dimensiones espaciales y convoluciones 1x1 para ajustar la dimensionalidad de los canales. Estas estrategias ayudan a reducir la complejidad de la red mientras se mantiene la calidad de las representaciones aprendidas.

- *Once-For-All (OFA)*

  OFA propone un método novedoso para desacoplar la etapa de entrenamiento del modelo y la etapa de búsqueda. El modelo se entrena solo una vez y las subredes se pueden extraer de él según los requisitos. La red única (OFA) proporciona flexibilidad para tales subredes en cuatro dimensiones importantes de una red neuronal convolucional: profundidad, ancho (*width*), tamaño del kernel y dimensión.

  La idea central de OFA es entrenar una única arquitectura "universal" en una amplia variedad de tamaños y restricciones de recursos (como la cantidad de parámetros o la cantidad de operaciones de punto flotante por segundo) en un proceso llamado "*entrenamiento gradual*".

  En la Figura 2.18 se puede observar que la arquitectura base (universal) se adapta a diferentes escenarios utilizando un proceso de "*ajuste de recursos*" en el que se ajustan los *hiper-parámetros* para cumplir con requisitos específicos del entorno donde se ejecutará la red. Esto permite obtener diferentes sub-arquitecturas que funcionan bien en diferentes dispositivos o restricciones de cómputo.

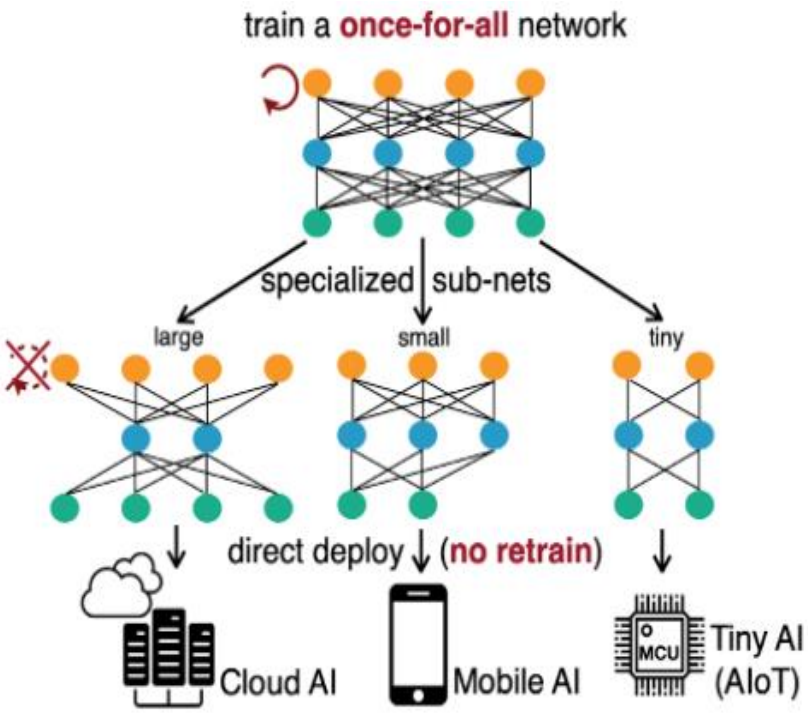


Figura 2.18: Ilustración simplificada del funcionamiento de OFA[11].

En [Zaidi et al., 2022] se realiza una comparación de estos modelos, donde se muestran que presentan resultados aceptables en términos de precisión y latencia, utilizando *ImageNet Top-1* [Russakovsky et at., 2015] para la comparación. Los resultados de la comparación

[11] Imagen extraída de la página "*Bringing OFA (Once-for-All) to FPGA*". https://www.xilinx.com/developer/articles/advantages-of-using-ofa.html (último acceso: 3/09/2023).

pueden observarse en la Tabla 2.1. Se puede apreciar cómo a medida que pasan los años, los avances en los modelos han permitido aumentar el nivel de precisión en la detección, al mismo tiempo que se reduce la latencia. De acuerdo a [Zaidi et al., 2022], se espera que los modelos con un MFLOP[12] menor a 600 puedan funcionar adecuadamente en dispositivos móviles promedio.

Tabla 2.1: Comparación de *Lightweight* models. Versión adaptada de [Zaidi et al., 2022].

Modelo	Año	Precisión %	Latencia (ms)	Parámetros (Millones)	MFLOPs
MobileNet	2017	70.6	113	4.2	569
ShuffleNet	2017	73.3	108	5.4	524
MobileNet v2	2018	74.7	143	6.9	300
ShuffleNet v2	2018	75.4	178	7.4	597
MobileNet v3	2019	75.2	58	5.4	219
OFA	2020	80.0	58	7.7	595

Si bien las *lightweight networks* resultan prometedoras, todavía presentan algunas dificultades pendientes a resolver. Una de las principales problemáticas tiene que ver con que los *detectores* dentro de esta categoría todavía no son adecuados para el reconocimiento en tiempo real o en el análisis de streams de video [Zou et al., 2023]. Esto se debe a que:

- Por un lado, la aplicación de un *detector* en todos los frames de video introduce un costo computacional excesivo y, a menudo, violaría los requisitos de latencia que las aplicaciones móviles necesitan.
- Por otro lado, los *detectores de objetos basados en imágenes* (lo cual aplica a todos los detectores mencionados en este capítulo) no son conscientes de la importancia que tiene la continuidad temporal que existe entre los frames de video sucesivos.

Es por esto que en [Xu et al., 2020], [Bagchi et al., 2020], [Zou et al., 2023] y [Zaidi et al., 2022] se menciona que todavía falta encontrar una solución que se ajuste a estos dispositivos para resolver la detección sobre streams de video. De todas formas, el uso de los *lightweight models* podría resultar adecuado si es que la aplicación puede permitirse analizar una cantidad limitada de frames por segundo, como se plantea en [Al-Azzoa et al., 2018].

2.3 Datasets

En el ámbito del reconocimiento de objetos, un *dataset*[13] es una colección de datos que consiste en imágenes o videos junto con las etiquetas asociadas que indican la presencia y

[12] El término MFLOPs (o *Million floating point operations per second*) se utiliza para referenciar a la cantidad de millones de operaciones de punto flotante que realiza un algoritmo por segundo. Los MFLOPS son una medida utilizada en informática para medir la velocidad de procesamiento.

[13] Es importante destacar que en este trabajo los *datasets* se enmarcan dentro del ámbito del *reconocimiento de objetos*. Sin embargo, estos pueden ser usados en un espectro más amplio de aplicaciones. En términos generales, un *dataset* se refiere a una colección de datos que puede incluir diversos tipos de información, como texto, sonido, datos tabulares, entre otros.

la ubicación de objetos específicos en esas imágenes o en los cuadros de esos videos. Por ejemplo, si una imagen representa una cocina, las etiquetas podrían incluir "refrigerador", "horno", "mesa", etc. Asimismo, algunas etiquetas pueden incluir información sobre la posición y el tamaño de los objetos en la imagen. Cabe destacar que las imágenes o videos de un *dataset* pueden variar en resolución, iluminación, fondo y otros factores para reflejar la diversidad de condiciones del mundo real.

Por lo general, los *datasets* se suelen dividir en conjuntos de entrenamiento, validación y prueba. El conjunto de datos de entrenamiento se utiliza para entrenar un modelo, el conjunto de validación se utiliza para ajustar parámetros y evitar el sobreajuste, y el conjunto de prueba se utiliza para evaluar el rendimiento del modelo en datos no vistos. Aunque no todos los *datasets* siguen esta división, es una práctica bastante común.

La disponibilidad de *datasets* extensos y con el menor grado de sesgo posible es fundamental para el desarrollo de modelos de reconocimiento y detección de objetos de calidad. A lo largo de la última década, la comunidad científica y la industria han trabajado para recopilar y compartir conjuntos de datos grandes y de alta calidad que se han convertido en pilares para la investigación y el desarrollo en este campo. Muchos de estos, incluso, se han convertido en referencias estándar en la investigación y se utilizan como *benchmarks* para comparar el rendimiento de diferentes modelos y algoritmos [Zou et al., 2023]. A continuación se describen algunos de estos *datasets*:

- Los datasets *PASCAL* (*VOC*)[14] (de 2005 a 2012), surgieron en el marco de las competencias más importantes en la comunidad de visión por computadora en sus primeras etapas. Dos versiones de Pascal-VOC se utilizan principalmente en la detección de objetos: VOC07 y VOC12, donde la primera consta de 5.000 imágenes de entrenamiento con más de 12.000 objetos etiquetados, y la segunda consta de 11.000 imágenes de entrenamiento con más de 27.000 objetos. Estos dos conjuntos de datos contienen anotaciones para 20 clases de objetos comunes en la vida cotidiana, como "persona", "gato", "bicicleta", "sofá", entre otros.
- Los datasets de reconocimiento visual a gran escala de *ImageNet* (*ILSVRC*)[15] (de 2010 a 2017) [Russakovsky et at., 2015], se convirtieron en un punto de referencia para evaluar el rendimiento de algoritmos. El conjunto de datos de detección de ILSVRC contiene 200 clases de objetos visuales. La cantidad de imágenes/instancias de objetos en este conjunto de datos es dos órdenes de magnitud mayor que la de Pascal-VOC.
- Microsoft *COCO* (*Common Objects in COntext*)[16] es un conjunto de datos que contiene 91 objetos comunes que un niño de 4 años puede reconocer fácilmente. Fue lanzado en 2015 y su popularidad ha aumentado constantemente desde entonces. Este dataset incluye más de dos millones de instancias y un promedio de 3.5 clases por imagen. Asimismo, contiene 7.7 instancias por imagen, lo cual es considerablemente más grande que otros conjuntos de datos populares. Uno de los

[14] Página con información de *Pascal-VOC*: http://host.robots.ox.ac.uk/pascal/VOC/ (último acceso: 25/09/2023)
[15] Página con información de *ImageNet*: https://www.image-net.org/challenges/LSVRC/ (último acceso: 25/09/2023)
[16] Página con información de *Microsoft COCO*: https://cocodataset.org/#home (último acceso: 25/09/2023)

avances más significativos de COCO es que, además de las anotaciones de cuadros delimitadores, cada objeto también está etiquetado mediante segmentación por instancia, lo que ayuda en la localización precisa. Además, COCO contiene más objetos pequeños (cuya área es menor al 1% de la imagen) y objetos ubicados de manera más densa en comparación con otros conjuntos de datos.

- El dataset de *Open Images de Google*[17] está compuesto por 9.2 millones de imágenes, que cuentan con anotaciones a nivel de imagen, cuadros delimitadores de objetos, y máscaras de segmentación, entre otros. Fue lanzado en 2017 y ha recibido seis actualizaciones. Para la detección de objetos, *Open Images* dispone de 16 millones de cuadros delimitadores para 600 categorías en 1.9 millones de imágenes, lo que lo convierte en el conjunto de datos más grande para la localización de objetos. Los creadores del conjunto de datos se esforzaron por seleccionar imágenes interesantes, complejas y diversas, con un promedio de 8.3 categorías de objetos por imagen.

En la Tabla 2.2 se proporciona una síntesis de las características de los *datasets* mencionados, en la cual se detalla la distribución de datos destinados al entrenamiento y la validación. Se puede apreciar, además, que existe una variación significativa en la cantidad de imágenes, objetos anotados y clases entre los diversos conjuntos de datos.

Tabla 2.2: Comparación entre los datasets Pascal VOC07, Pascal VOC12, *ImageNet*, Microsoft COCO y Open Images de Google. Versión adaptada de [Zou et al., 2023].

Dataset	Clases	Entrenamiento			Validación		
		Imágenes	Objetos	Objetos/ Imagen	Imágenes	Objetos	Objetos/ Imagen
PASCAL VOC07	20	2.501	6.301	2,52	2.510	6.307	2,51
PASCAL VOC12	20	5.717	13.609	2,38	5.823	13.841	2,37
ImageNet (ILSVRC)	200	456.567	478.807	1,05	20.121	55.502	2,76
Microsoft COCO	91	118.287	860.001	7,27	5.000	36.781	7,35
Open Images de Google	600	1.743.042	14.610.229	8,38	41.620	204.21	4,92

Cabe mencionar que si bien estos *datasets* son algunos de los más utilizados, y avalados por la comunidad, solo representan una fracción muy pequeña de todos los recursos públicos disponibles en internet.

2.4 Implementación de modelos de reconocimiento de objetos

Principalmente existen dos opciones a la hora de generar un modelo de reconocimiento de objetos utilizando *deep learning*, cada uno con sus propias ventajas y desventajas. La primera

[17]Página con información de *Open Images de Google*: https://storage.googleapis.com/openimages/web/index.html (último acceso: 25/09/2023)

alternativa implica entrenar un modelo desde cero (*from scratch*), mientras que la segunda implica realizar el proceso conocido como aprendizaje por transferencia (*o transfer learning*).

Entrenar un modelo desde cero implica reunir un dataset grande que contenga ejemplos de los objetos que se desean reconocer, y diseñar una arquitectura de red neuronal que aprenda a extraer características relevantes de los datos y realizar predicciones. Si bien los resultados obtenidos con esta alternativa pueden llegar a ser muy buenos, existen algunos desafíos que se deben mencionar [Shallu and Mehra, 2018]:

- *Dataset a utilizar*: entrenar una red desde cero requiere de un dataset grande y diverso. Los modelos de *deep learning* se desempeñan mejor cuando tienen una gran cantidad de datos para mejorar las capacidades de reconocimiento y generalizar mejor a los objetos. Recopilar y etiquetar dicho dataset puede ser un proceso largo y costoso.
- *Arquitectura del modelo*: se debe diseñar la arquitectura de la CNN, lo que implica determinar el número de capas, tipos de capas (por ejemplo, convolucionales, de pooling, completamente conectadas) y configurar sus pesos y parámetros. Diseñar una arquitectura efectiva requiere experiencia y experimentación.
- *Tiempo de entrenamiento*: puede ser costoso en recursos computacionales y lleva mucho tiempo, especialmente en datasets grandes. A menudo se requiere hardware potente como GPUs o TPUs para acelerar el proceso de entrenamiento.
- *Preocupaciones por el sobreajuste*: sin una cantidad suficiente de datos y técnicas de regularización cuidadosas, existe el riesgo de sobreajuste, donde el modelo funciona bien en los datos de entrenamiento, pero mal en datos no vistos.

Debido a esto, una posible alternativa podría ser utilizar modelos pre-entrenados como inicialización o extractor de características para generar nuevos modelos que resuelvan la tarea que esperamos. Esta acción de reutilizar una red pre-entrenada se conoce como *transfer learning* o aprendizaje por transferencia, e implica transferir conocimientos de un dominio a otro. Es decir, implica tomar un modelo de red neuronal que generalmente se ha entrenado en un conjunto de datos extenso, y ajustarlo para una tarea o conjunto de datos específicos. El enfoque de *transfer learning* ofrece varias ventajas:

- *Eficiencia de datos*: no se necesitan tantos datos etiquetados como en la primera alternativa, ya que el modelo pre-entrenado ya ha aprendido características significativas de su conjunto de datos original.
- *Desarrollo más rápido*: ajustar finamente un modelo pre-entrenado suele ser más rápido que entrenar uno desde cero, ya que el modelo ya ha aprendido muchas características útiles y puede proporcionar un buen punto de partida.
- *Mejor generalización*: los modelos pre-entrenados han aprendido a capturar características generales como bordes, texturas y formas, que pueden ser valiosas para diversas tareas de reconocimiento. Con esta información, el nuevo modelo puede adaptarse a un conjunto de datos específico, incluso si contiene clases previamente no conocidas.
- *Menor necesidad de recursos computacionales*: generar un nuevo modelo de reconocimiento a través de aprendizaje por transferencia requiere menos poder de cómputo en comparación con el entrenamiento de modelos grandes desde cero.

Sin embargo, también poseen algunas desventajas:

- Los modelos pre-entrenados suelen estar diseñados para tareas específicas, lo que significa que no se pueden aplicar directamente a todas las tareas de aprendizaje automático. Si la tarea que el nuevo modelo busca resolver es muy diferente de la tarea del modelo pre-entrenado original, el rendimiento puede ser limitado.
- El rendimiento del *transfer learning* está vinculado a la calidad del modelo pre-entrenado. Si el modelo pre-entrenado tiene limitaciones o sesgos, estos pueden transferirse a la tarea de destino.
- La arquitectura del modelo pre-entrenado y la arquitectura requerida para la tarea que el nuevo modelo busca resolver pueden no ser compatibles, lo que requiere adaptaciones adicionales.

En términos generales, se puede decir que existen dos enfoques de *transfer learning*, la extracción de características y el *fine-tuning* (ajuste fino):

- En el primer enfoque, el de extracción de características, se utilizan las capas convolucionales pre-entrenadas de un modelo CNN existente, que ya ha aprendido patrones y características útiles de un dataset. Estas capas convolucionales se "*congelan*", lo que significa que sus pesos no se actualizan durante el entrenamiento del nuevo modelo. Solo se añaden capas adicionales al final del modelo para adaptarse a la tarea específica que se está abordando. Este enfoque es útil cuando se dispone de un conjunto de datos pequeño o cuando el conjunto de datos de destino es similar al conjunto de datos original utilizado para pre-entrenar el modelo.

  Un ejemplo puede visualizarse en la Figura 2.19, donde se elimina la última capa de la CNN base, y se agregan dos nuevas capas de adaptación a la red. Estas dos capas se ajustan posteriormente para afinarse a la nueva tarea.

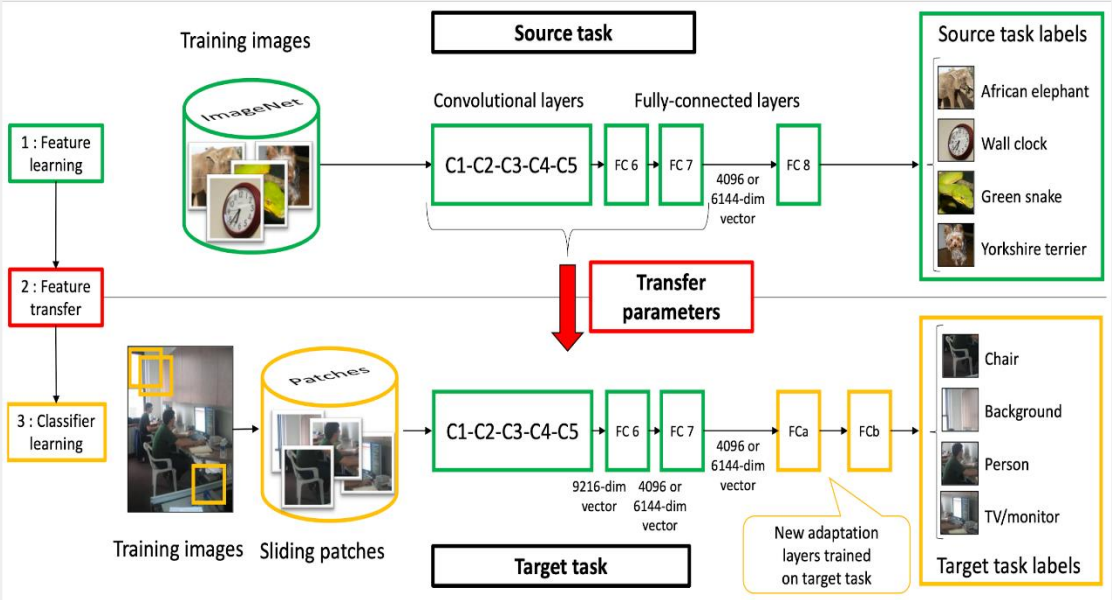


Figura 2.19: Ejemplo de *transfer learning* usando extracción de características [Oquab et al., 2014].

- En el enfoque de *fine-tuning*, se toma un modelo pre-entrenado y se ajustan no sólo las capas adicionales agregadas para la tarea específica, sino también algunas o

todas las capas convolucionales originales. Los pesos de estas capas convolucionales se actualizan durante el entrenamiento para adaptarse al nuevo conjunto de datos. Este método se emplea cuando el conjunto de datos de destino es más diverso o significativamente diferente al conjunto de datos original, y se busca adaptar el modelo a características más específicas de la nueva tarea.

Es importante mencionar que la implementación de modelos de reconocimiento de objetos puede ser simplificada mediante el uso de diversas herramientas y librerías especializadas existentes. Esto facilita la implementación de sistemas de detección y reconocimiento de objetos al proporcionar herramientas para construir, entrenar y desplegar modelos de manera más eficiente. También proveen modelos pre-entrenados, soporte para hardware acelerado y herramientas de evaluación.

2.5 Reconocimiento de objetos para el posicionamiento

Un uso interesante que se viene explorando en los últimos años en el campo del reconocimiento de objetos, como se menciona en [Xiao et al., 2018], es lo que se conoce como *posicionamiento indoor basado en visión* (o *vision-based indoor positioning*), que busca emplear la detección de objetos para guiar y ubicar al usuario dentro de espacios interiores (*indoor*). Es interesante primero hacer una mención breve sobre algunos estudios relacionados al posicionamiento indoor en general, para luego sí pasar a contextualizar qué acontece con *vision-based indoor positioning*.

Los dispositivos móviles (*smartphones*) están actualmente equipados con diversos sensores y permiten posicionar de diversas maneras. A continuación se detallan tres categorías [Ruizhi and Liang, 2017]:

- *Receptores de señales GNSS* (*Global Navigation Satellite System* o Sistema global de navegación por satélite): que incluyen GPS, BDS, GLONASS y Galileo. Si bien estos sistemas funcionan muy bien para espacios exteriores (*outdoor*), su funcionamiento en espacios indoor suele ser poco preciso debido a la interferencia de las construcciones e infraestructura de los edificios.
- *Señales de radio frecuencia:* como Wi-Fi, Bluetooth y comunicaciones inalámbricas celulares, etc. Para que estas señales se puedan utilizar para posicionar, se requiere de una grilla de señales pre-armadas para poder luego analizar la señal recibida del dispositivo móvil, y en base a los datos de la grilla determinar la posición. También se requiere algún algoritmo que realice todos estos cálculos.

  Dentro de esta categoría se encuentra el *fingerprinting* de redes WI-FI y los *beacons*, los cuales fueron investigados previamente por el autor de este trabajo, como se menciona en la Introducción (Capítulo 1). Estos mecanismos pueden alcanzar precisiones de posicionamiento de 2 a 5 metros, pero son fácilmente interferidos por cambios en el entorno y la presencia de personas cercanas. Ambas alternativas (*fingerprinting* y *beacons*) presentan una desventaja común, requieren equipamiento adicional instalado en el lugar para poder funcionar. En el caso de *fingerprinting* de WI-FI se debe contar con al menos tres redes de WI-FI, mientras que la alternativa de Bluetooth requiere de la instalación de *Beacons*.
- *Sensores incorporados en los smartphones*, como por ejemplo, acelerómetros o giroscopios. Estos se usan como complementos de los dos mecanismos de

posicionamientos anteriores, porque permiten sumar información, por ejemplo, del ángulo de visión que tiene la persona (que está usando un dispositivo móvil). Al utilizar los sensores propios del dispositivo no se requieren equipamientos adicionales, lo que lo convierte en una solución que brinda bajo costo y escalabilidad. Sin embargo, por sí solos no posicionan a la persona. Dentro de esta categoría podría incluirse también la cámara del dispositivo, para obtener información complementaria sobre lo que está viendo la persona.

Veamos ahora en qué consiste *vision-based indoor positioning*, concepto acuñado por primera vez por Microsoft en 1998 [Shafer et al., 1998]. En ese entonces, el grupo de tecnología de visión de Microsoft analizó cómo la tecnología podría facilitar la vida en el futuro, y una de las tecnologías fundamentales propuestas era ubicar a las personas en el hogar mediante el uso de diversas técnicas de videovigilancia. Desde entonces, este concepto de acuerdo a [Xiao et al., 2018] ha evolucionado dando lugar a tres posibles opciones de implementación que varían en la forma de almacenar y procesar la información:

- *Referencia en base a objetos*

  Estos sistemas se basan en la detección de objetos en imágenes y en la correspondencia de estos objetos con una base de datos del edificio, que contiene información de su ubicación en el interior del mismo. Es muy común almacenar la información de los objetos en bases de datos de edificios como CityGML[18], que permite generar una representación 3D del mismo. Este proceso de generación de modelos CityGML implica la recopilación de datos geoespaciales, como detalles topográficos e información de infraestructuras. Con tecnologías como escáneres láser, se capturan datos 3D que representan edificios, calles y otros elementos clave. Estos datos se organizan y estructuran, asignándoles atributos semánticos para describir su función y tipo. El resultado es un modelo geoespacial tridimensional estandarizado y semántico, útil para la planificación urbana y la gestión territorial.

  En [Mautz, 2012] se plantea un ejemplo concreto donde se explica una posible implementación de este enfoque. Primero se identifica la habitación correcta vista por la cámara del dispositivo, constatándola con una base de datos CityGML. Esto se realiza a partir de la nube de puntos 3D obtenida por el sensor de imágenes, se detectan objetos fijos como ventanas y puertas, y sus propiedades geométricas se pueden comparar con la base de datos. La Figura 2.20 muestra una de estas habitaciones modelada en CityGML por [Mautz, 2012] junto con sus diferentes objetos de interés.

  Luego de identificar la habitación, se realiza el posicionamiento preciso a nivel de modelo digital mediante una técnica que combina trilateración y resección espacial:

  - La trilateración es un método para determinar la posición de un punto en el espacio utilizando mediciones de distancias desde ese punto hacia tres ubicaciones conocidas. Se miden las distancias, se crean círculos o esferas alrededor de las ubicaciones conocidas, y la intersección de estas proporciona la posición estimada del punto.
  - La resección espacial es un método utilizado para determinar la posición y orientación de una cámara o sensor en el espacio tridimensional. Implica medir los

[18] Página de CityGML en *Open Geospatial Consortium:* https://www.ogc.org/standard/citygml (último acceso: 25/09/2023).

ángulos entre puntos conocidos en la escena y la cámara, y luego utilizar esta información para calcular la posición y orientación de la cámara.

En base a esta combinación se establece la posición en el espacio físico del dispositivo.

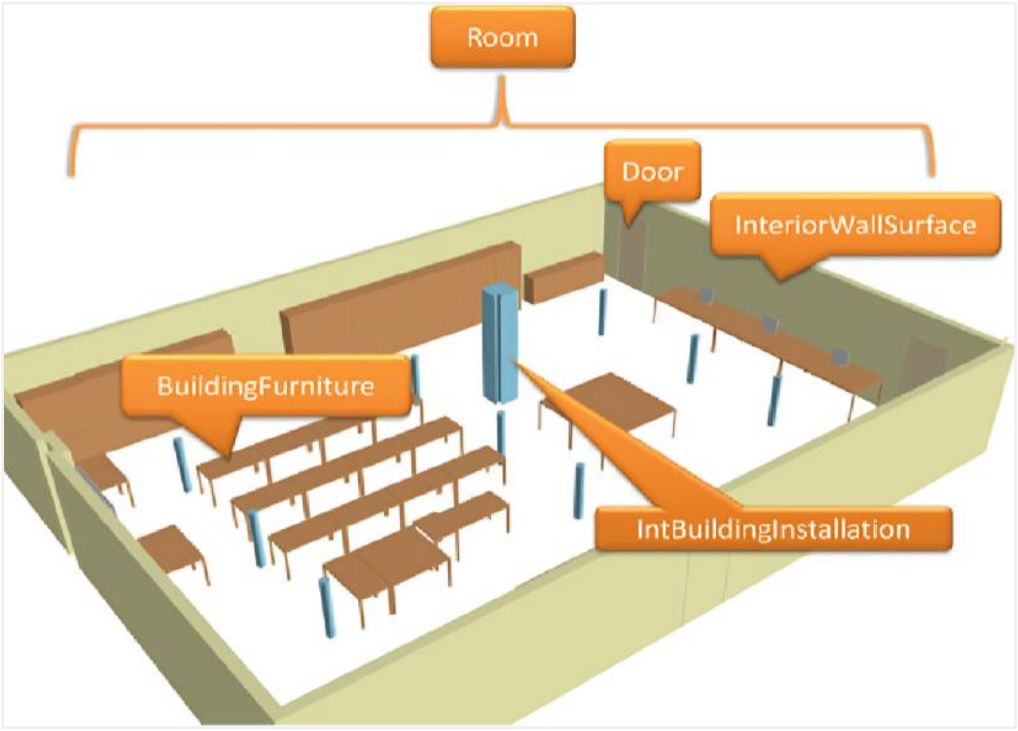


Figura 2.20: Ejemplo de modelo en CityGML para representar una habitación y la ubicación de los objetos de interés [Mautz, 2012].

Otra propuesta es la planteada en [Xiao et al., 2018]. En este trabajo se presenta un método para determinar la posición mediante el uso de "puntos de control". Cada objeto de interés en el espacio está compuesto por una serie de estos puntos, los cuales representan características físicas con coordenadas precisas asociadas. En este enfoque, la cámara del dispositivo, mediante el uso de un modelo, reconoce los objetos y sus respectivos puntos de control. Posteriormente, un algoritmo compara las distancias entre las coordenadas detectadas por la cámara con las registradas en una base de datos, permitiendo así calcular la distancia del usuario respecto a dichos objetos. Aunque los resultados obtenidos en este trabajo son prometedores, es importante señalar una limitación significativa: el modelo utilizado requiere ser entrenado con datos específicos del entorno para identificar correctamente los puntos de control. Esta exigencia podría dificultar la aplicación generalizada de esta solución.

- *Referencia en base a imágenes*

  La idea de este enfoque es comparar imágenes previamente tomadas y etiquetas de rutas específicas del edificio con la vista actual de la cámara del dispositivo. En este sentido, este tipo de enfoque requiere de una etapa inicial de recolección de datos donde se deberán tomar imágenes que abarquen las ubicaciones claves del entorno. Luego, estas imágenes deberán ser etiquetadas, estableciendo conexiones entre elementos visuales y ubicaciones específicas dentro del edificio. Posteriormente, durante la implementación en tiempo real, la vista actual de la cámara del dispositivo se compara con las secuencias de imágenes previamente capturadas. El sistema utiliza algoritmos para identificar

correspondencias entre la escena actual y las referencias almacenadas.

Un ejemplo de esto se puede apreciar en la Figura 2.21, donde se visualizan las secuencias de imágenes previamente capturadas (Figura 2.21.a) y la vista actual de una cámara del dispositivo (Figura 2.21.b).

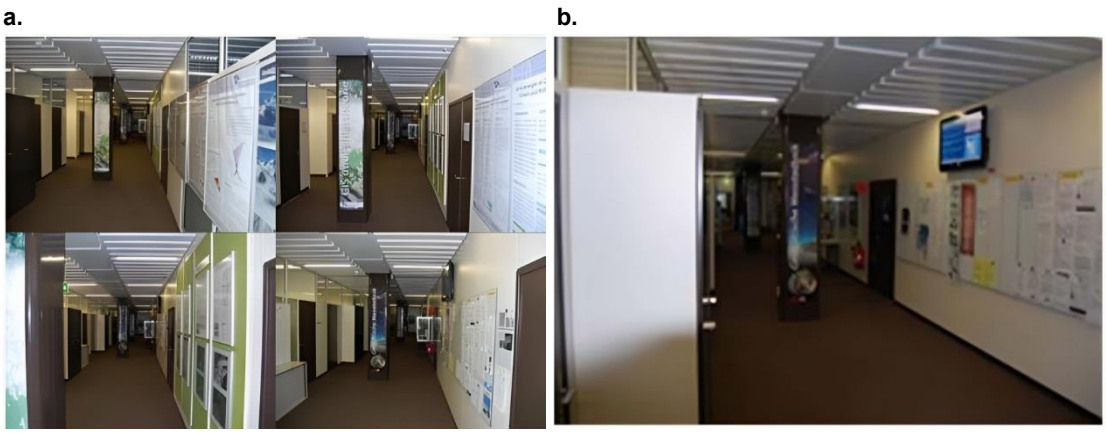


Figura 2.21: Ejemplo de *Referencia en base a imágenes.* (a) Capturas tomadas sobre un pasillo durante el proceso de reconocimiento del ambiente. (b) Vista actual de la cámara del dispositivo. [Mautz, 2012].

El principal desafío de este enfoque radica en lograr la capacidad de posicionamiento en tiempo real. Para la "*Referencia en base a imágenes*" no se identifican objetos particulares, sino ambientes específicos en lugares particulares. Esta correspondencia de imágenes implica una carga computacional particularmente alta.

- *Referencia de objetivos codificados*

  Esta opción plantea utilizar objetos "marcadores" tales como códigos de barra, QR y patrones de punto como los que se pueden ver en la Figura 2.22. La generación de este tipo de marcadores tiene un costo relativamente bajo. Para implementar la técnica, solo se requiere generar los marcadores, vincularlos con información pertinente del lugar y luego ubicarlos físicamente en su posición correspondiente.

Figura 2.22: Tres ejemplos de objetivos codificados utilizados para la identificación de puntos. [Mautz, 2012].

Los marcadores cumplen tres propósitos en el desarrollo algorítmico:

  - *Simplificación de la detección automática de puntos correspondientes*. Estos patrones sirven como puntos de referencia visualmente reconocibles, simplificando el proceso de identificación y correspondencia entre diferentes vistas o imágenes.

- *Introducción del concepto de escala del sistema*. Si a todos los puntos se les da un tamaño o dimensiones preestablecidas, es más fácil determinar la distancia al objeto.
- *Distinción e identificación de objetivos mediante el uso de un código único para cada marcador.* Si cada objeto tiene un código único, cada instancia será fácilmente reconocible, independientemente de que pertenezca a una misma clase de objetos.

En [Mautz, 2012] se plantea un ejemplo de un sistema de posicionamiento indoor de bajo costo para dispositivos móviles mediante el uso de marcadores fiduciarios con códigos de barras. Estos marcadores se colocan en paredes, carteles u objetos específicos. Si se saca una foto de estos marcadores, la posición del dispositivo puede determinarse con una precisión de "unos pocos centímetros". Además, se puede mostrar información adicional basada en la ubicación, como la próxima sala de conferencias o la próxima sesión.

2.6 Análisis de los conceptos abordados en este capítulo

El reconocimiento de objetos se ha convertido en un campo en constante expansión, ofreciendo muchas oportunidades para el avance de la inteligencia artificial y la visión por computadora. En este capítulo se buscó proporcionar una base para comprender los conceptos, técnicas y tecnologías esenciales que sustentan este campo de estudio.

Una de las áreas más relevantes en el reconocimiento de objetos ha surgido con *deep learning* y las redes neuronales convolucionales (CNN). Estas redes tienen la capacidad de aprender a identificar características visuales complejas en los datos, como se menciona en la Sección 2.2. Para este trabajo son de interés los detectores *one-stage*, y más específicamente, la subcategoría dentro de estos conocida como *lightweight models* (Sección 2.2.2.3). Este tipo de detectores son los que más se ajustan al objetivo del presente trabajo, al estar diseñados específicamente para ejecutarse en dispositivos con recursos limitados como los *smartphones*. Los *lightweight networks* presentan dos ventajas interesantes:

- Son capaces de reconocer objetos de manera adecuada y en tiempo real, tanto en dispositivos de gama media y alta. Esto garantiza que una aplicación que usa estos modelos pueda ejecutarse en una amplia variedad de dispositivos móviles, maximizando así su utilidad y accesibilidad.
- Estos modelos se ejecutan de forma local, directamente en el dispositivo. Esto elimina posibles retrasos adicionales que podrían surgir al depender de la transferencia y procesamiento de datos en un servidor externo. Este enfoque no solo optimiza la velocidad de respuesta, sino que también reduce la carga de la red, proporcionando así una experiencia de usuario más eficiente. Es decir, es independiente de la conectividad a Internet. Al prescindir de la necesidad de estar conectados, estos modelos garantizan un funcionamiento ininterrumpido incluso en escenarios donde la conectividad puede ser un desafío.

Al ser los *lightweight models* un área nueva de investigación, todavía hay que seguir evaluando su viabilidad real para usos particulares, como se plantea en [Zaidi et al., 2022]. Más aún, los *lightweight models* no proporcionan mecanismos o facilidades para llevar a cabo el posicionamiento de los usuarios en espacios *indoor*. Se espera aportar en esta dirección

con esta tesis.

En relación a esto, en la Sección 2.5 se abordó el concepto de *vision-based indoor positioning;* en donde el reconocimiento de objetos se podría extender más allá de la identificación de objetos individuales, enfocándose en usarlo para posicionar a los usuarios en espacios indoor. Aunque se han desarrollado soluciones ad-hoc [Xiao et al., 2018], todavía no existen implementaciones generalizadas que puedan aplicarse de manera efectiva y sistemática. Esto presenta una oportunidad de investigación que será abordada en esta tesis.

En particular, para este trabajo será de interés explorar la opción de *vision-based indoor positioning* conocida como "*Referencia en base a objetos*" (y detallada en la Sección 2.5), la cual se caracteriza por la generación de bases de datos de objetos con información adicional para posicionar al usuario. Para esta opción se podría lograr una implementación más flexible y adaptable a diversos espacios físicos, ya que se basa en objetos que podrían estar presentes en varios lugares distintos.

Cabe mencionar que las otras dos alternativas de *vision-based indoor positioning* presentan más limitantes para lograr una solución escalable:

- La opción "*Referencia en base a imágenes*" se encuentra muy acoplada a entornos específicos, ya que plantea la necesidad de desarrollar modelos capaces de reconocer ambientes específicos de un edificio en particular, lo cual lo convierte en una alternativa poco versátil y muy compleja como para proponer una solución escalable.
- Por otro lado, la opción "*Referencia de objetivos codificados*" presenta los desafíos propios del uso de códigos QR como mecanismo de posicionamiento, ya explorados por el autor de este trabajo en [Challiol et al., 2019].

En la actualidad existen distintas plataformas que facilitan la implementación y ejecución práctica de los conceptos descriptos en este capítulo. Una de estas plataformas es *Tensorflow*, la cual se abordará con más detalle en el próximo capítulo, ya que se utilizará para implementar la solución propuesta en esta tesis.

3. Tensorflow

Tensorflow [Tensorflow] es una de las plataformas de código abierto más importantes y robustas en el mundo del *deep learning* y la inteligencia artificial. Fue desarrollada y publicada por Google en 2015. Su importancia radica en su capacidad para impulsar la construcción y el entrenamiento de redes neuronales de manera eficiente. Asimismo, cuenta con una comunidad activa de desarrolladores y científicos de datos que contribuyen con una amplia gama de herramientas y extensiones.

Una ventaja importante que tiene *Tensorflow* es que es multiplataforma. También tiene la capacidad de poder aprovechar hardware especializado, como unidades de procesamiento de gráficos (GPU) y unidades de procesamiento tensorial (TPU). Esto permite acelerar significativamente el proceso de entrenamiento de modelos y permite manejar conjuntos de datos más grandes y modelos más complejos.

Tensorflow busca presentar soluciones y facilidades para la totalidad del proceso de desarrollo y despliegue de modelos. Como se muestra en la Figura 3.1, *Tensorflow* separa este proceso en cuatro etapas:

1. *Preparación de datos: Tensorflow* facilita este proceso mediante funciones que permiten la carga, transformación y normalización de datos, asegurando que estén en condiciones óptimas para su utilización en la construcción y entrenamiento de modelos. Además, provee datasets estandarizados que se encuentran preparados para tareas iniciales de entrenamiento y validación.
2. *Creación de modelos de Aprendizaje Automático (AA):* ofrece una amplia gama de herramientas y librerías que facilitan la creación de modelos de *machine learning* y *deep learning*. Con *Tensorflow* se pueden construir y ejecutar redes para reconocimiento de imágenes, clasificación de dígitos escritos a mano, redes neuronales recurrentes, embedding de palabras, procesamiento de lenguaje natural, detección de video y muchas otras aplicaciones.
3. *Implementación de modelos:* una vez que los modelos han sido entrenados, *Tensorflow* facilita su implementación en diversos entornos como servicios web, aplicaciones móviles o entornos de producción. Ofrece herramientas para exportar modelos entrenados y asegurar su interoperabilidad con diferentes plataformas, simplificando la integración de modelos en aplicaciones del mundo real.
4. *Implementación de operaciones de Machine Learning (MLOps):* proporciona funcionalidades y herramientas que respaldan las mejores prácticas de MLOps, como la integración con *TensorBoard* para visualizar métricas de entrenamiento y la compatibilidad con *Tensorflow Extended* (TFX) para la orquestación de flujos de trabajo en entornos de producción.

Es importante mencionar que *Tensorflow* no obliga a seguir todas estas etapas, sino que permite iniciar por cualquiera de ellas.

En el ámbito de la visión por computadora y del reconocimiento de objetos, uno de los tipos de redes más populares que se pueden construir con *Tensorflow* son las redes neuronales convolucionales (*CNN*), las cuales fueron presentadas en la Sección 2.2 de esta tesis.

Figura 3.1: Parte de la pantalla principal de *Tensorflow* [Tensorflow].

3.1 Tensorflow Object Detection API

Tensorflow cuenta con un paquete llamado *Tensorflow Object Detection API*[19] *que* facilita la creación, entrenamiento y despliegue de modelos de detección de objetos. Esto es muy interesante por su capacidad para simplificar drásticamente el proceso de construcción de modelos de detección de objetos. Antes de su existencia, crear un *detector* de objetos desde cero era una tarea monumental que requería conocimientos profundos en *deep learning* y programación. Sin embargo, con este paquete (API) esa barrera se ha reducido significativamente, ya que permite a los desarrolladores centrarse en ajustar y adaptar los modelos en lugar de crear toda la infraestructura desde cero.

A continuación se detallan algunas de las características más importantes de *Tensorflow Object Detection API.*

- *Entrenamiento de modelos desde cero (from scratch)*

  Tensorflow permite a los usuarios entrenar su propia red neuronal de detección de objetos desde cero. La API proporciona funciones para que los usuarios puedan crear su propia base de datos y realizar el entrenamiento correspondiente de la red. Sin embargo, este trabajo consume mucho tiempo y requiere computadoras con potentes GPU (*Unidades de Procesamiento Gráfico*) para entrenar o afinar estas redes.

- *Modelos pre-entrenados*

  La API proporciona varios modelos de reconocimiento y de detección de objetos pre-entrenados que se encuentran listos para ser utilizados. Esta colección se agrupa bajo el nombre de "*Model Zoo*". Estos modelos fueron entrenados utilizando el conjunto de datos de objetos comunes en contexto (COCO) y están diseñados para diferentes necesidades en términos de velocidad y precisión. Se puede elegir un modelo en función de la relación entre el rendimiento y la eficiencia que mejor se adapte a cada caso de uso específico.

  Se debe mencionar que los modelos de detección entrenados se basan en *detectores* que se mencionaron en el Capítulo 2 como, por ejemplo, Faster R-CNN, *CenterNet,*

[19] Documentación oficial de la API de *Object Detection de Tensorflow*. https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest (último acceso: 17/10/2023).

EfficientDet, y *MobileNet V1* y *V2*. En la Tabla 3.1 se presentan algunos de estos modelos, junto a su tiempo promedio de respuesta y su nivel medio de precisión (mAP de sus siglas en inglés *Mean Average Precision*).

Como se puede ver en la tabla, muchos de los modelos presentados proveen varias alternativas, ya sea porque utilizan distintas versiones del mismo detector o porque fueron entrenados con imágenes en distintas resoluciones. También se puede apreciar cómo, a medida que aumenta la resolución de las imágenes utilizadas, aumenta el nivel de precisión del modelo y se aumentan los tiempos de espera para obtener una respuesta.

Tabla 3.1: Listado de algunos de los modelos pre-entrenados brindados por *Tensorflow*[20].

Modelo	mAP	Velocidad (ms)
Faster R-CNN ResNet50 V1 640x640	29.3	53
Faster R-CNN Inception ResNet V2 640x640	37.7	206
Faster R-CNN Inception ResNet V2 1024x1024	38.7	236
SSD MobileNet v2 320x320	20.2	19
SSD MobileNet V1 FPN 640x640	29.1	48
SSD MobileNet V2 FPNLite 320x320	22.2	22
SSD MobileNet V2 FPNLite 640x640	28.2	39
CenterNet HourGlass104 Keypoints 512x512	40.0/61.4	76
CenterNet Resnet50 V1 FPN 512x512	31.2	27
EfficientDet D0 512x512	33.6	39
EfficientDet D1 640x640	38.4	54
EfficientDet D2 768x768	41.8	67

En [Salunkhe et al., 2021] se presenta un ejemplo concreto de cómo los modelos pre-entrenados de la *Tensorflow Object Detection API* pueden ser aplicados en situaciones del mundo real. En este caso, los autores desarrollaron una aplicación para dispositivos Android con el propósito de asistir a personas con discapacidades visuales. La aplicación en cuestión tiene la capacidad de reconocer objetos cotidianos para luego nombrarlos en voz alta (usando la librería *text-to-speech de Android*). En esta aplicación se utilizó un modelo pre-entrenado basado en *SSD-MobileNet* sin realizar modificaciones sobre el mismo. De acuerdo a los autores, el grado de precisión fue de un 90%. Esto demuestra cómo la API de *Tensorflow Object Detection* facilita la creación de aplicaciones útiles sin la necesidad de ajustes extensos en el modelo pre-entrenado.

[20] *Modelos pre-entrenados disponibles en Tensorflow*: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md (último acceso: 17/10/2023).

- *Capacidad de Transfer Learning*

  Al utilizar la técnica de *transfer learning* (detallada en la Sección 2.4), se pueden generar nuevos modelos a partir de modelos pre-existentes ajustándolos a una tarea más específica, como por ejemplo reconocer objetos previamente desconocidos. Esto ahorra tiempo y recursos, ya que los desarrolladores no tienen que empezar desde cero y pueden aprovechar el conocimiento previo del modelo.

  En [Taqi et al., 2019] se presenta un ejemplo concreto que destaca el proceso de *fine-tuning* de un modelo para la detección de cáncer de piel. Los autores detallan el proceso de adaptación de un modelo pre-entrenado, basado en *SSD-Mobilenet*, para que pueda identificar eficazmente esta enfermedad a partir de imágenes médicas. El proceso de *fine-tuning* se realizó utilizando el *dataset* de imágenes del *ISIC Challenge 2018*. De acuerdo a los autores, en los estudios experimentales realizados, el modelo alcanzó niveles de precisión del 100% al utilizarlo en un celular *Android*.

  Cabe mencionar que la documentación de *Tensorflow* explica de forma detallada el paso a paso que se debe seguir para re-entrenar y exportar estos modelos.

- *Componentes configurables*

  La API permite ajustar fácilmente varios aspectos del modelo, como la arquitectura de la red, los hiper-parámetros de entrenamiento y los detalles de la inferencia.

  Los modelos de detección de objetos pueden variar en términos de complejidad y profundidad, y la API brinda a los usuarios la flexibilidad para elegir la arquitectura que mejor se adapta a sus necesidades. Esto es especialmente útil cuando se trata de equilibrar el rendimiento y la eficiencia en una aplicación particular. Por ejemplo, en aplicaciones donde se requiere una detección rápida, se puede optar por una arquitectura más ligera, mientras que en aplicaciones donde la precisión es primordial, se puede seleccionar una arquitectura más compleja.

  Además de la arquitectura de la red, los hiper-parámetros de entrenamiento también son configurables. Los hiper-parámetros influyen en cómo se ajusta el modelo durante el proceso de entrenamiento. Los usuarios pueden ajustar valores como la tasa de aprendizaje, el tamaño del lote (*batch size*), la cantidad de épocas de entrenamiento y otros parámetros relacionados. Esta flexibilidad permite afinar el modelo para que se adapte mejor a los datos y las necesidades específicas de la aplicación.

  Por último, la capacidad de configurar los *detalles de la inferencia* (es decir, cómo se utiliza el modelo una vez que está entrenado y en producción) es clave para adaptar el modelo a la aplicación final. Esto puede incluir aspectos como la velocidad de procesamiento, la eficiencia en el uso de recursos computacionales y la precisión de la detección. Los usuarios pueden ajustar estos parámetros para garantizar que el modelo funcione de manera óptima en el entorno real de la aplicación.

- *Soporte para múltiples formatos de datos*

  La API es compatible con una variedad de formatos de datos comunes utilizados en la detección de objetos. Esto facilita la integración de datasets existentes y la utilización de datos etiquetados en diferentes formatos. Por ejemplo, si un desarrollador tiene un dataset en formato JSON y otro en formato CSV, la API puede manejar ambos formatos sin problemas. Esto ahorra tiempo y esfuerzo, ya que los usuarios no tienen que preocuparse

por la compatibilidad de formatos y pueden centrarse en la tarea de entrenar y evaluar modelos. La API además facilita esta tarea al proporcionar herramientas y utilidades que simplifican la carga, procesamiento y etiquetado de datos en diferentes formatos.

- *Evaluación y métricas*

  La API proporciona métricas de evaluación para medir el rendimiento del modelo en la detección de objetos. Algunas de las métricas más comunes incluyen la precisión, el recall[21] y F1-score[22]. Estas métricas permiten a los desarrolladores y evaluadores medir el desempeño de un modelo de detección de objetos de manera cuantitativa y detallada. Identificar las métricas relevantes y comprender sus resultados es esencial para perfeccionar un modelo a lo largo del tiempo y garantizar que cumpla con los requisitos específicos de una aplicación. En una última instancia, estas métricas ayudan a impulsar mejoras en la precisión y la eficiencia de los modelos de detección de objetos.

- *Despliegue de modelos*

  La API permite exportar los modelos entrenados (desde cero o a partir de los modelos pre-entrenados) para su implementación en aplicaciones de producción. Estos modelos pueden utilizarse luego en distintos entornos.

3.2 Entrenamiento de modelos en *Tensorflow* por *Transfer Learning*

El proceso de entrenamiento de un modelo de reconocimiento de objetos utilizando *Tensorflow* se divide en cuatro etapas, como se puede visualizar en la Figura 3.2. A continuación, se describen cada una de ellas:

- *Dataset*: esta primera etapa implica la recolección de las imágenes y la anotación de las mismas. La anotación de imágenes es el proceso de agregar información adicional o metadatos a una imagen para indicar la presencia y ubicación de objetos específicos dentro de la misma. Las anotaciones suelen incluir la delimitación de contornos alrededor de los objetos de interés mediante cajas delimitadoras (*bounding boxes*). Existen herramientas de software gratuitas como *LabelImg* que permiten a los usuarios realizar este proceso fácilmente. Generalmente, estas anotaciones quedan almacenadas en archivos externos a las imágenes, utilizando algún formato estándar como XML. Durante el entrenamiento, la anotación ayudará al modelo a localizar precisamente el o los objetos presentes en la imagen. El contorno generará y guardará puntos de coordenadas en un archivo XML.

[21] En el contexto de deep learning, *recall* es una métrica utilizada para evaluar el rendimiento de modelos de clasificación. Representa la proporción de instancias positivas correctamente identificadas con respecto al total de instancias positivas en el dataset, y se calcula como el cociente entre los verdaderos positivos y la suma de verdaderos positivos y falsos negativos. Un alto recall indica que el modelo es capaz de capturar la mayoría de las instancias positivas.

[22] F1-score se utiliza como una métrica de evaluación. Esta puntuación proporciona una medida equilibrada entre la capacidad del modelo para identificar correctamente las instancias positivas y evitar falsos positivos. La fórmula del F1-score integra tanto precisión como recall, ofreciendo una valoración conjunta del rendimiento del modelo. Un F1-score más alto indica un mejor equilibrio entre la precisión y el *recall* del modelo.

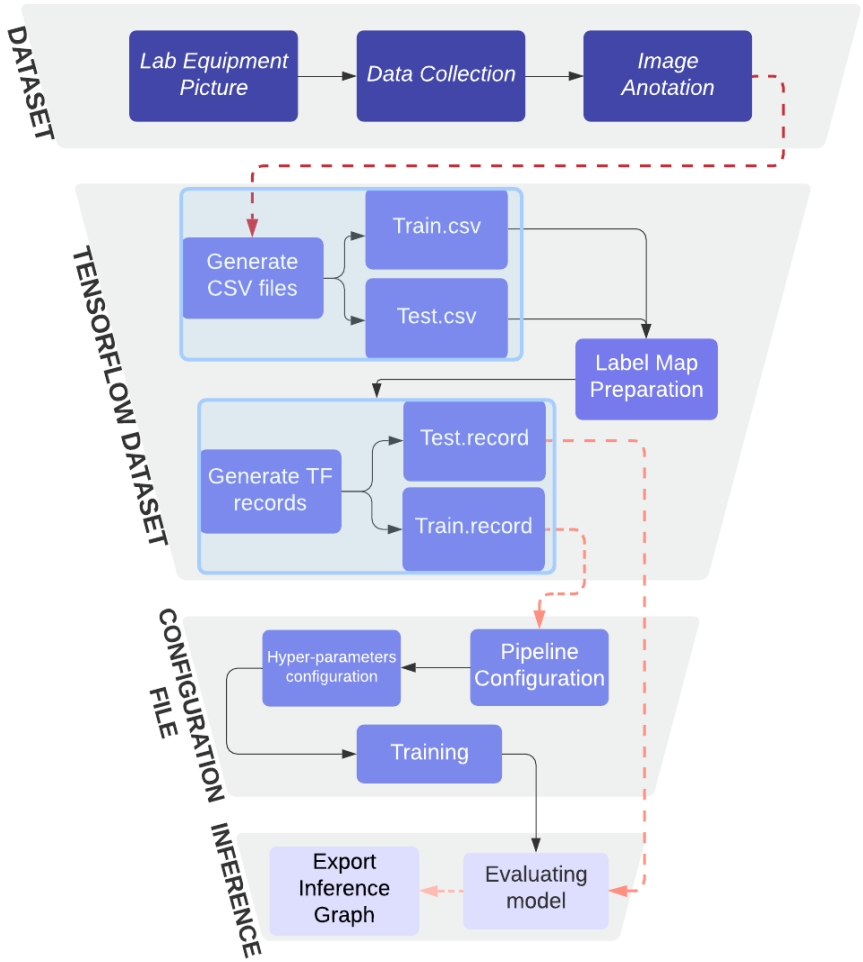


Figura 3.2: Workflow de entrenamiento de modelos por transfer-learning en Tensorflow [Estrada et al., 2022].

- *Dataset de Tensorflow* (*Tensorflow Dataset*)*:* para hacer eficientemente los cálculos, *Tensorflow* utiliza un tipo de registro binario especial conocido como *TFRecord*. Estos registros permiten que el dataset se almacene como una secuencia de cadenas binarias que mejora el rendimiento del modelo, utilizando menos espacio en disco. *Tensorflow* provee un script para generar estos *TFRecord* a partir de las imágenes y sus anotaciones en formato CSV, por lo que muchas veces es necesario realizar un paso previo de transformación para convertir las anotaciones obtenidas en el paso anterior de XML a CSV.

  El último paso en la configuración de estos registros *TFRecord* es crear un archivo .pbtxt que contenga todas las clases de etiquetas que se almacenarán en un archivo de registro.

  Como se puede apreciar en la Figura 3.2, un punto importante a destacar de esta etapa, es que el dataset de imágenes se separa en datos de entrenamiento (training) y datos de prueba (testing), generando archivos CSV y TF para cada uno de estos respectivamente.

- *Archivo de configuración* (*Configuration File*)*:* como se mencionó anteriormente, *Tensorflow* provee varios modelos pre-entrenados basados en el dataset Microsoft *COCO*. Estos modelos se pueden utilizar para configurar nuevos modelos de *deep learning*. Algunos de estos fueron presentados en la Tabla 3.1.

  Además del modelo pre-entrenado, se requiere un archivo de configuración. Este archivo

de configuración especifica entre otras cosas los detalles del modelo a utilizar, como la arquitectura de la red, los hiper-parámetros, la ubicación de los archivos *TFRecord* y otros ajustes.

Luego, el proceso de entrenamiento utiliza este archivo de configuración para cargar el modelo pre-entrenado y entrenarlo con el nuevo dataset. Durante el entrenamiento, los pesos del modelo se ajustan para adaptarse a las características del nuevo dataset, utilizando la información aprendida previamente. Después del entrenamiento, la API genera un archivo que sirve como punto de control de entrenamiento en un formato específico llamado .ckpt. Este archivo es un archivo binario que contiene todos los pesos, sesgos y valores de otras variables.

- *Inferencia* (*Inference*): después de entrenar el modelo, el paso final es ponerlo en ejecución para realizar inferencias sobre los nuevos datos. La etapa de inferencia es donde se puede evaluar la capacidad del modelo para generalizar datos no vistos previamente. Se puede medir la precisión de las predicciones utilizando métricas como la precisión, el recall o el F1-score.

3.3 Tensorflow en diferentes entornos

Como se mencionó anteriormente, *Tensorflow* está diseñado para ser compatible con una gran variedad de plataformas y dispositivos. De acuerdo a su página oficial, como se puede observar en la Figura 3.3, *Tensorflow* presenta herramientas que permiten ejecutar modelos en entornos tales como páginas web, dispositivos móviles, y en servidores. Los modelos provistos por *Tensorflow*, así como los modelos propios generados y exportados utilizando la API de *Object Detection,* pueden ser utilizados con cualquiera de estas variantes (páginas web, dispositivos móviles y en servidores).

Figura 3.3: Entornos de ejecución de *Tensorflow* [Tensorflow].

A continuación se describen brevemente cada uno de los entornos de la Figura 3.3:

- *Tensorflow.js [TensorflowJS]*

  Tensorflow.js ofrece herramientas para cargar, entrenar y ejecutar modelos utilizando el lenguaje de programación *JavaScript*, permitiendo su utilización en entornos web.

  En la Figura 3.4 se puede observar parte de la página principal de *Tensorflow.js*, donde se listan algunas de sus funcionalidades más importantes:

  - *Tensorflow.js* permite ejecutar modelos directamente en el navegador web y en entornos *Node.js*. Esto significa que se pueden crear aplicaciones que ejecuten modelos de forma nativa en el navegador del usuario, sin necesidad de servidores externos. Otra alternativa, implica aprovechar la capacidad de procesamiento de un

servidor, lo cual se puede hacer mediante Node.js. Además, *Tensorflow.js* es compatible con la importación de modelos pre-entrenados desde *Tensorflow* o *Keras*. Esto es útil cuando se desea aprovechar modelos ya existentes y entrenados en tareas específicas, como modelos de visión por computadora o modelos de procesamiento de lenguaje natural.

- En términos de modelado, *Tensorflow.js* permite construir y entrenar redes neuronales desde cero utilizando la API que provee. Este entrenamiento de modelos podría realizarse directamente en el navegador o sobre un servidor en Node.js.
- Además, es posible realizar *transfer learning*, aprovechando los modelos pre-entrenados para adaptarlos a tareas específicas.

Figura 3.4: Funcionalidades de *Tensorflow.js* [Tensorflow.js].

Tensorflow.js puede ser utilizado también en dispositivos móviles en desarrollos híbridos como *React Native [ReactNative]* y *Ionic*[23]. Este tipo de frameworks combinan tecnologías web (HTML, CSS, JavaScript) con tecnologías nativas de los dispositivos. En particular, para *React Native*, existe un paquete[24] específico desarrollado por *Tensorflow*, que aprovecha la aceleración por GPU utilizando WebGL a través de expo-gl. Esto es especialmente beneficioso para mejorar el rendimiento de las operaciones de *Tensorflow.js* en dispositivos móviles, ya que la GPU puede procesar cálculos de manera más eficiente que la CPU.

- *Tensorflow Lite [TensorflowLite]*

  Tensorflow Lite es una versión optimizada de *Tensorflow* diseñada para dispositivos móviles y otros dispositivos con recursos limitados como tablets y dispositivos IoT (*Internet of Things - Internet de las Cosas*).

  La finalidad principal de *Tensorflow Lite* es permitir la ejecución de modelos en tiempo real en este tipo de dispositivos. Además, es beneficiosa para dispositivos que no pueden depender completamente de la conexión a Internet, ya que el modelo corre de forma local.

  Dado que estos dispositivos tienen limitaciones de almacenamiento, *Tensorflow Lite* está optimizado para tener un tamaño reducido. Además, *Tensorflow Lite* incluye herramientas para convertir y optimizar modelos entrenados en formatos compatibles con dispositivos móviles, y proporciona una API para cargar y ejecutar estos modelos de manera eficiente en hardware específico, como CPUs, GPUs y unidades de procesamiento tensorial (TPUs).

[23] Documentación oficial de *Ionic*. https://ionicframework.com/ (último acceso: 17/10/2023).

[24] Documentación del paquete @tensorflow/tfjs-react-native: https://github.com/tensorflow/tfjs/tree/master/tfjs-react-native#readme (último acceso: 17/10/2023).

En la Figura 3.5 se pueden observar los pasos a seguir para utilizar *Tensorflow Lite*. En primer lugar se realiza la selección del modelo, siguiendo por su conversión a una versión comprimida. Luego, este archivo comprimido se carga en el dispositivo móvil para posteriormente ejecutarlo (que puede implicar una optimización).

Figura 3.5: Pasos para usar *Tensorflow Lite* [Tensorflow Lite].

- *Tensorflow Extended (TFX)*

  TFX es una plataforma integral desarrollada por *Tensorflow* para abordar el ciclo completo de vida de los modelos de *aprendizaje automático* en entornos de producción. Ofrece una variedad de componentes integrados que cubren desde la ingestión y preparación de datos hasta el despliegue y monitorización de modelos en producción. Su enfoque integral permite a los equipos gestionar de manera efectiva aspectos críticos como la calidad de datos, la selección de modelos, la optimización de hiper-parámetros y la compatibilidad con la infraestructura de producción, proporcionando así una solución completa y eficiente para implementar modelos de *aprendizaje automático* a gran escala.

3.4 Análisis de los conceptos abordados en este capítulo

Tensorflow es de interés para esta tesis por diversos motivos:

- Uno de los puntos más fuertes de *Tensorflow*, como se menciona en la Sección 3.3, es su versatilidad y compatibilidad con distintos entornos y plataformas. En particular, su aplicabilidad en el desarrollo móvil se presenta como un foco de interés a explorar de acuerdo a los objetivos de este trabajo.

  Como se mencionó anteriormente, *Tensorflow* provee dos librerías que permiten exportar y ejecutar modelos de forma local en dispositivos móviles y otros sistemas con recursos limitados: *Tensorflow.js y Tensorflow Lite.* Si bien en la teoría ambas alternativas resultan interesantes, será importante analizar en la práctica cómo funcionan ambas librerías tanto en aspectos de implementación (como facilidad de configuración de los modelos) y de ejecución (revisando cuestiones tales como usabilidad y tiempos promedio de respuesta en inferencias).

- Otro aspecto interesante de *Tensorflow* es la disponibilidad de modelos pre-entrenados listos para ser utilizados. Estos modelos (entre los que se encuentran los mencionados en la Sección 3.1) son capaces de reconocer y detectar objetos, eliminando la necesidad de entrenar modelos propios. Aunque la cantidad de objetos reconocibles por estos modelos es acotada, resultan interesantes, ya que permiten realizar pruebas iniciales y evaluaciones rápidas del rendimiento de *Tensorflow.js* y *Tensorflow Lite*.

- Como se plantea en la Sección 3.1, los modelos pre-entrenados además se pueden

utilizar para entrenar nuevos modelos que se ajusten a dominios o problemas específicos. En el marco de este trabajo, surge el interés por investigar sobre este tema. Particularmente, se buscará experimentar en el desarrollo de modelos capaces de detectar nuevos objetos en entornos específicos.

- Se debe destacar también de *Tensorflow* su robusta documentación, contando con una amplia cantidad de tutoriales y ejemplos concretos, además de contar con una comunidad muy activa. Todos estos recursos van a permitir, en el marco de esta tesis, guiar y agilizar el proceso completo de desarrollo, entrenamiento y ejecución de modelos, donde es probable que surjan varios desafíos o problemas.

Es importante señalar que, si bien *Tensorflow* proporciona mecanismos para desarrollar y ejecutar modelos de reconocimiento de objetos en entornos móviles, no incluye inherentemente funcionalidades específicas para el posicionamiento de usuarios en el espacio. Es decir, es necesario analizar en esta tesis cómo lograr una solución de posicionamiento escalable.

4. Pruebas de modelos de reconocimiento y detección

En los capítulos anteriores se sentaron las bases teóricas relacionadas al reconocimiento y a la detección de objetos. Por un lado, en el Capítulo 2 se exploraron los conceptos de *deep learning* y la arquitectura de las CNN. Asimismo, se detalló el funcionamiento de los *detectores* más utilizados para la detección de objetos. Por otro lado, en el Capítulo 3 se presentó *Tensorflow*, una plataforma con la que se puede, entre otras cosas, entrenar y ejecutar modelos de reconocimiento de objetos.

Tomando de base los conceptos ya abordados, en este capítulo se realizan una serie de pruebas utilizando tanto *TensorFlow Lite* como *TensorFlow.js*. Para esto se implementan algunas aplicaciones prototípicas. En particular, se explora la facilidad de despliegue y de ejecución de modelos de detección de objetos en entornos móviles, usando tanto modelos pre-entrenados como modelos propios entrenados.

Resulta importante mencionar que el autor de este trabajo posee una sólida experiencia en el desarrollo de aplicaciones móviles utilizando *JavaScript* y en entornos de desarrollo híbridos, particularmente en *React Native [ReactNative]*. Se debe también señalar que cuenta con experiencia limitada en el desarrollo nativo de aplicaciones móviles, especialmente en lenguajes como *Java* y *Swift*. Esto hace que trabajar con herramientas como *Tensorflow Lite*, sea un desafío mayor para el autor de esta tesis. A pesar de esto, se tomó la decisión de realizar pruebas tanto con *Tensorflow Lite* como con *Tensorflow.js* para analizar de manera objetiva si existían diferencias significativas entre ambos.

4.1 Pruebas con modelos pre-entrenados

En esta sección se presentan pruebas con modelos pre-entrenados usando tanto *TensorFlow Lite* como *TensorFlow.js.*

4.1.1 Tensorflow Lite

Teniendo en cuenta el poco conocimiento en el ámbito del desarrollo nativo en lo que respecta al autor de esta tesis, se tomó la decisión de explorar proyectos de código abierto que hicieran uso de *Tensorflow Lite*, los cuales pudieran servir como punto de partida. Durante esta búsqueda, se encontró el proyecto *TensorFlow Lite Object Detection Android Demo*[25], incluido en la documentación oficial de *Tensorflow Lite*. Este proyecto contiene una aplicación desarrollada para *Android* que permite realizar la detección de múltiples objetos en tiempo real utilizando la cámara trasera del dispositivo.

En la Figura 4.1 se presenta una captura de pantalla de la aplicación corriendo, en particular, utilizando el detector *MobileNet*. Se puede observar como la aplicación es capaz de identificar varios objetos en la pantalla y cómo utiliza *bounding boxes* para enmarcar la ubicación de cada objeto detectado. En la esquina superior izquierda de cada *bounding box*, se muestra el *label* (etiqueta) del objeto identificado junto con el porcentaje de confianza asociado,

[25] Página de *TensorFlow Lite Object Detection Android Demo*: https://www.tensorflow.org /lite/android/tutorials/object_detection?hl=es-419. (último acceso: 27/10/2023).

proporcionando así información sobre el grado de certeza con el que se ha realizado la detección.

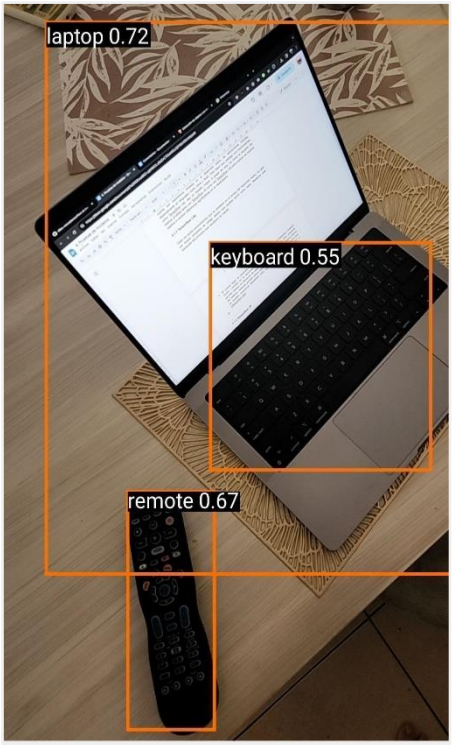


Figura 4.1: Ejemplo de detección de objetos usando la aplicación *TensorFlow Lite Object Detection Android Demo*.

Un punto interesante de la aplicación *TensorFlow Lite Object Detection Android Demo*, es que provee una interfaz que permite configurar varios parámetros de la API de *Tensorflow Lite*. A continuación, se describen cada uno de ellos:

- Permite la selección entre cuatro modelos de detección pre-entrenados. Estos modelos, provistos por *Tensorflow*, se basan en las arquitecturas *MobileNet SSD V1*, *EfficientDet Lite 0*, *EfficientDet Lite1* y *EfficientDet Lite2*.

  Aunque estos cuatro modelos son capaces de reconocer los mismos 80 objetos, ya que fueron entrenados con el mismo conjunto de datos (Microsoft *COCO*, descripto en la Sección 2.3), la arquitectura que utilizan cada uno por debajo provoca que existan diferencias en términos de tamaño, velocidad de inferencia y niveles de precisión. Esto se puede observar en mayor detalle en la Figura 4.2[26], donde se presentan dos gráficos que comparan por un lado la precisión de los cuatro modelos en relación con su tamaño (medido en MB), y por otro lado la precisión respecto del tiempo de latencia (o de inferencia). Se puede apreciar en la figura que si bien los cuatro modelos fueron entrenados utilizando el mismo dataset (Microsoft *COCO*), el tamaño de los mismos varía en función de la arquitectura que tiene cada uno.

[26] La información mostrada en la figura se obtuvo de pruebas realizadas corriendo los cuatro modelos sobre la CPU (*Unidad Central de Procesamiento*) de un dispositivo *Google Pixel 4* utilizando 4 *threads*.

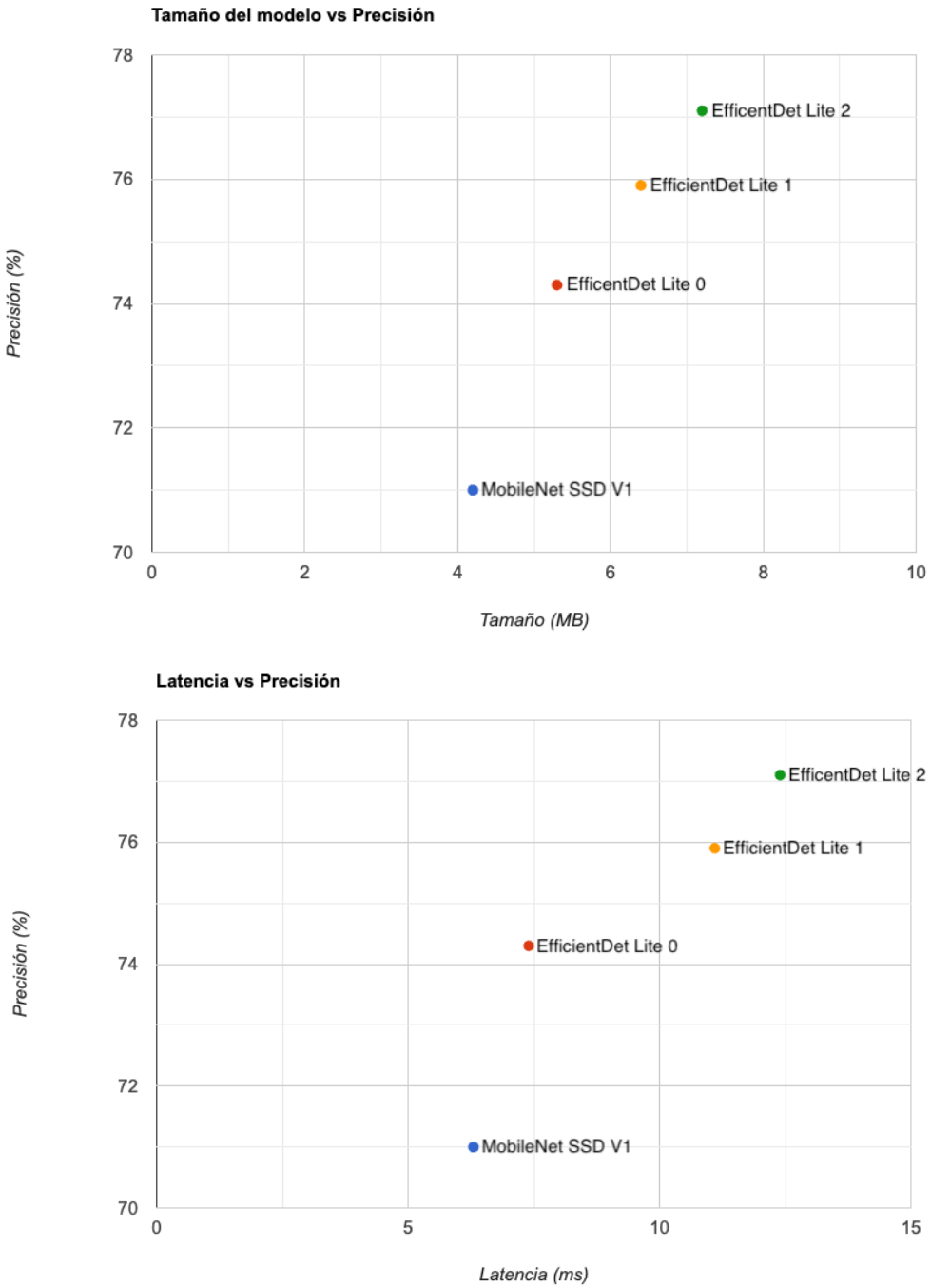


Figura 4.2: Comparación de modelos en términos de precisión vs tamaño y precisión vs latencia de distintos modelos pre-entrenados para *Tensorflow Lite*. Gráficos adaptados de "*Higher accuracy on vision models with EfficientNet-Lite*" [27].

Se puede deducir a partir de la Figura 4.2 que el modelo basado en *MobileNet SSD V1* se destaca por tener la tasa de latencia más baja, aunque su precisión es inferior en comparación con los otros modelos. Además, es el modelo más liviano. Por otro lado, el modelo basado en *EfficientDet Lite2* se caracteriza por tener la precisión más

[27] Comparación de modelos de *Tensorflow Lite*: https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html (último acceso: 20/10/2023).

alta, a pesar de ser el más pesado y de tener el tiempo de latencia más alto.

Con la Figura 4.2 se puede concluir que hay una correlación entre los niveles de precisión respecto al tamaño (a mayor tamaño, mayor precisión), pero esto impacta de forma directa en el tiempo de latencia (donde a mayor precisión se incrementa la latencia).

- En la aplicación se puede configurar la cantidad máxima de resultados a mostrar en pantalla y el umbral de detección (o *threshold*) mínimo a considerar en las detecciones.
- Asimismo, se puede definir el número de *threads* (hilos) de procesamiento a utilizar. Esto permite brindar un control más granular sobre el rendimiento y la eficiencia computacional de la aplicación. El número máximo de *threads* a configurar depende de las limitaciones de hardware de cada dispositivo.
- La aplicación permite elegir entre CPU, GPU y NNAPI (*Android Neural Networks API*). Es decir, se puede determinar qué tipo de hardware o acelerador de hardware se utilizará para llevar a cabo las operaciones de inferencia en la aplicación de detección de objetos:
    - CPU (*Unidad Central de Procesamiento*): este es el procesador principal del dispositivo. Utilizar la CPU para la inferencia implica realizar los cálculos en el procesador general del dispositivo. Esto puede ser adecuado para dispositivos con CPU potentes.
    - GPU (*Unidad de Procesamiento Gráfico*): las GPU están diseñadas para manejar tareas costosas en paralelo, como las operaciones requeridas en la inferencia de modelos. Utilizar la GPU puede acelerar significativamente el proceso de inferencia, especialmente para modelos más grandes y complejos.
    - NNAPI (*Android Neural Networks API*): es una interfaz de programación de aplicaciones *Android* que proporciona acceso a aceleradores de hardware específicos para tareas de aprendizaje profundo. NNAPI puede aprovechar hardware especializado para mejorar aún más la velocidad de inferencia, siempre que el dispositivo admita esta API.

  Cabe mencionar que, si bien la presencia de una GPU se ha vuelto estándar en muchos smartphones, la potencia y la capacidad de la GPU pueden variar significativamente entre diferentes modelos y marcas. Por otra parte, no todos los dispositivos con sistema operativo *Android* cuentan con NNAPI.

En la Figura 4.3 se presentan algunas capturas de pantalla de la aplicación (*TensorFlow Lite Object Detection Android Demo*), donde se puede observar cómo se pueden aplicar las distintas configuraciones mencionadas anteriormente.

Asimismo, se puede ver en la Figura 4.3 que la aplicación provee información sobre el tiempo promedio que se tarda en hacer las inferencias (bajo el nombre de *Inference Time*). Esto resulta de gran utilidad, ya que de esa forma se puede evaluar de forma precisa el comportamiento en distintos dispositivos.

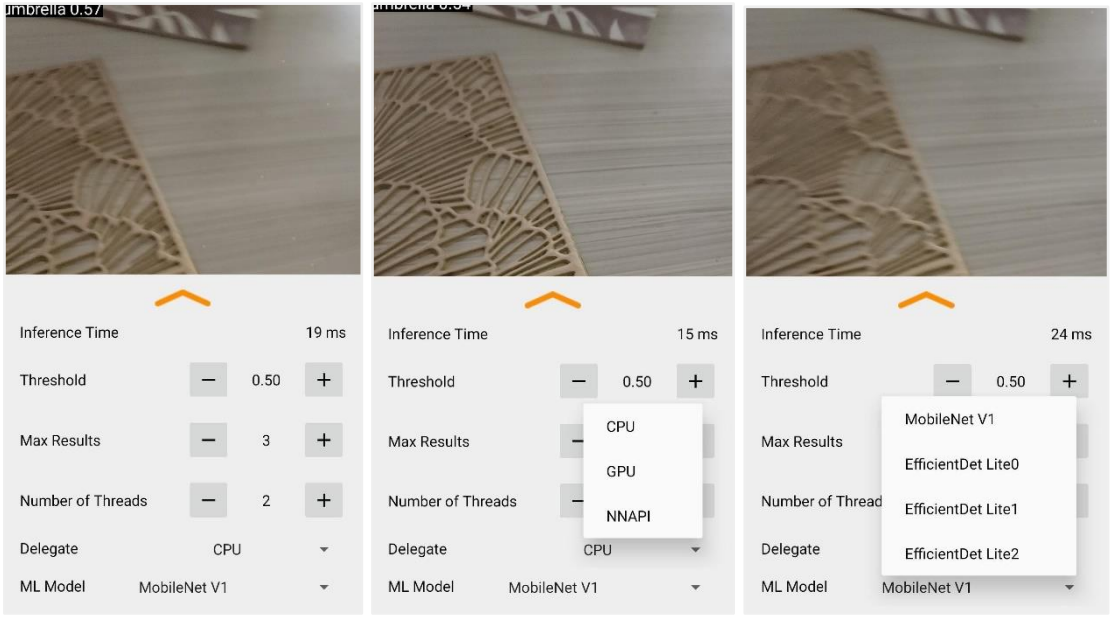


Figura 4.3: Configuraciones de la aplicación *TensorFlow Lite Object Detection Android Demo*.

Considerando estas posibles configuraciones, se tomó la decisión de realizar una evaluación propia con el propósito de examinar, en distintos dispositivos, las variaciones en los tiempos de inferencia de utilizar los detectores *MobileNet SSD V1* versus *EfficientDet Lite2*. Esta elección se fundamenta en los gráficos de la Figura 4.2 donde estos modelos se encontraban en los dos extremos en cuanto precisión y latencia (inferencia).

Teniendo en cuenta lo antes mencionado, se instaló la aplicación "*TensorFlow Lite Object Detection Android Demo"* en dos dispositivos distintos, uno de gama alta (Samsung S22) y uno de gama media (Samsung A20). En la Tabla 4.1 se presentan las especificaciones técnicas más importantes de cada dispositivo.

Tabla 4.1: Comparación de dispositivos utilizados en las pruebas.

Modelo	Versión Android	Procesador	GPU	RAM
Samsung S22 (2022)	14	Octa-core (1x2.8 GHz Cortex-X2 & 3x2.50 GHz Cortex-A710 & 4x1.8 GHz Cortex-A510)	Xclipse 920	8GB
Samsung A20 (2015)	11	Octa-core (2x1.6 GHz Cortex-A73 & 6x1.35 GHz Cortex-A53)	Mali-G71 MP2	3GB

En la Figura 4.4 se presenta un gráfico de barras que contiene los resultados obtenidos en términos de tiempo promedio de inferencia al correr la aplicación sobre el *Samsung S22*. El gráfico separa los resultados por modelo (*MobileNet SSD* o *EfficientDet Lite2*), por tipo de procesamiento (CPU o NNAPI) y por *threads* ejecutados en paralelo (de 1 a 4). Cabe mencionar que la aplicación indicaba que el dispositivo no estaba habilitado para correr con GPU, por lo que los resultados con este tipo de procesamiento no pudieron ser incluidos en la figura.

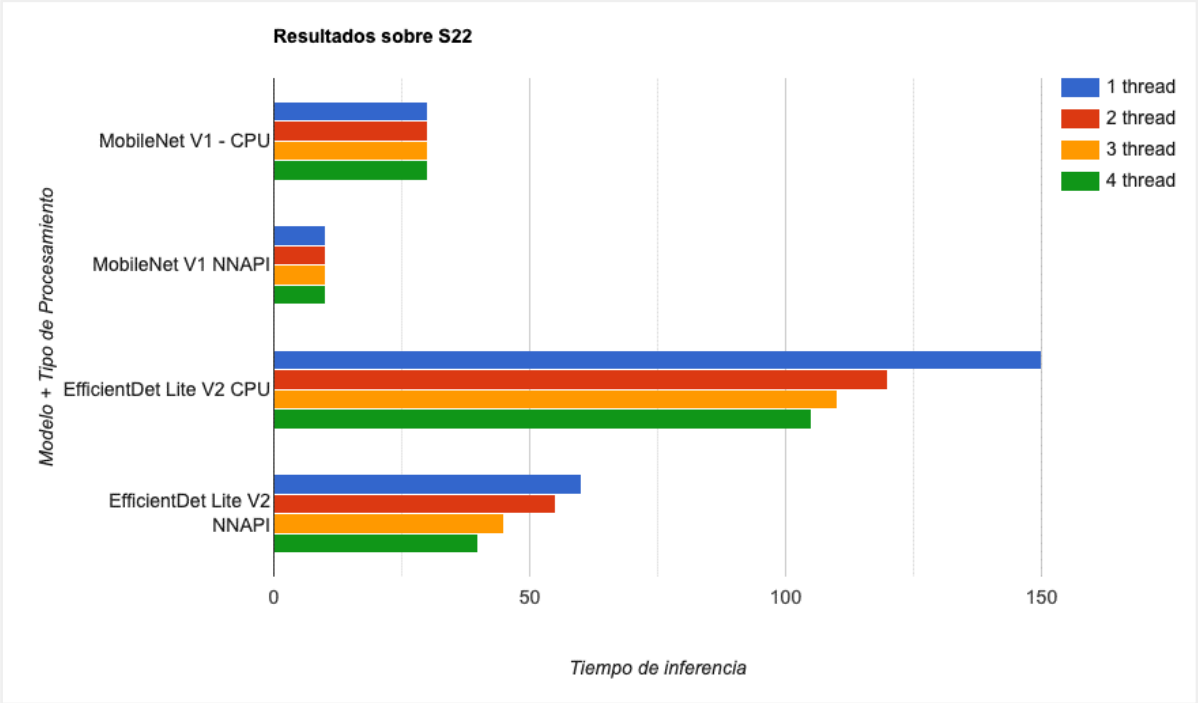


Figura 4.4: Resultados del tiempo de inferencia (en milisegundos) sobre el *Samsung S22* con *Tensorflow Lite*.

Al analizar los resultados obtenidos en la Figura 4.4 se pueden sacar las siguientes conclusiones:

- Los tiempos de inferencia obtenidos con *MobileNet* se mantuvieron constantes al incrementar el número de *threads*, mientras que con *EfficientDet* el tiempo se va achicando al aumentar la cantidad de *threads.*
- Los tiempos de inferencia se reducen a más de la mitad usando la NNAPI en lugar de CPU.
- Como era de esperar, los resultados con *MobileNet* fueron mucho mejores en cuanto al tiempo de inferencia que aquellos obtenidos por *EfficientDet*. De todos modos, los resultados obtenidos con este último fueron bastante buenos, siendo el peor caso de 150ms.

En la Figura 4.5 se presentan los resultados relacionados a los tiempos de inferencia al ejecutar la aplicación sobre el *Samsung* A20. En este caso, no fue posible realizar pruebas sobre GPU ni NNAPI, ya que este dispositivo no lo permitía.

Se puede notar en esta figura que a medida que se aumenta el número de *threads*, el tiempo de inferencia se incrementa, contrario a lo que acontece con el *Samsung* S22 donde los tiempos se mantienen constantes o se reducen. Por otro lado, se puede notar como los tiempos de inferencia son significativamente mayores en comparación con los resultados del *Samsung* S22. Si bien los tiempos obtenidos con *MobileNet* para el *Samsung* A20 son aceptables, los tiempos obtenidos con *EfficientDet* lo convierten en una opción totalmente inviable para su utilización en detecciones en tiempo real en este tipo de dispositivos. De

acuerdo con la documentación de *Tensorflow Lite*, para una ejecución en tiempo real es deseable que el tiempo de latencia en las inferencias no supere los 650 ms.

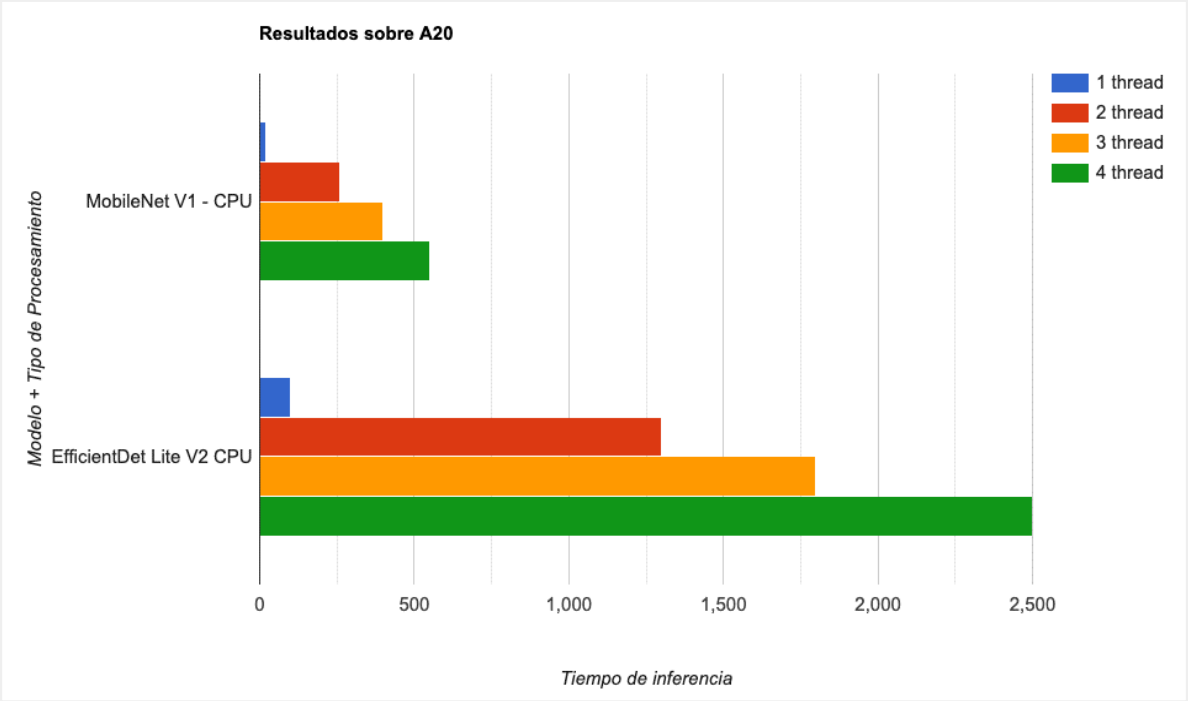


Figura 4.5: Resultados del tiempo de inferencia (en milisegundos) sobre el *Samsung A20* con *Tensorflow Lite*.

En cuanto a niveles de precisión, en términos generales fue bastante bueno y similar en ambos dispositivos para los objetos evaluados. La aplicación fue capaz de detectar correctamente objetos con niveles de precisión que oscilaban entre el 55 y 80%.

4.1.2 Tensorflow.js

Para llevar a cabo las pruebas con *Tensorflow.js*, se tomó la decisión de desarrollar una aplicación prototípica desde cero utilizando *React Native*. Esta decisión se vio respaldada por la familiaridad del autor de esta tesis con el lenguaje *JavaScript* y la motivación de explorar aspectos no abordados anteriormente como: la forma de configurar modelos, el manejo de la API proporcionada por *Tensorflow.js* y la implementación del renderizado de los *bounding boxes*. Esta decisión permitió apreciar en detalle los procesos internos asociados a *Tensorflow* y la personalización de los modelos, algo que no se pudo hacer en profundidad con el prototipo de *Tensorflow Lite*, donde estas cuestiones ya estaban resueltas en la aplicación usada (*TensorFlow Lite Object Detection Android Demo*).

La decisión de usar *React Native* para el desarrollo se vio avalada también gracias a que *Tensorflow* provee un adaptador específico para *React Native*[28], y para otras alternativas de desarrollo híbrido como *Ionic* o *Phonegap*. Este adaptador permite hacer uso de las características de hardware de los dispositivos móviles como, por ejemplo, aceleramiento por

[28] Documentación de *Tensorflow.js* para *React Native*: https://www.npmjs.com/package/@tensorflow/tfjs-react-native (último acceso: 27/10/2023).

GPU. Este tipo de optimizaciones son gestionadas directamente por *Tensorflow*, sin necesidad de realizar configuraciones adicionales.

De acuerdo a la documentación oficial de *Tensorflow.js*[29], existen varios modelos pre-entrenados de reconocimiento y de detección de objetos que resultan de interés para este trabajo:

- *Modelos de reconocimiento*: este tipo de modelo se especializa en tareas de clasificación, lo que significa que puede identificar objetos presentes en una imagen, pero no proporciona información detallada sobre su ubicación (como se menciona en la Sección 2.1). *Tensorflow.js* provee una librería que permite ejecutar dos modelos basados en *MobileNet*[30] (tanto la versión 1 como la versión 2). Además, la librería provee un parámetro *alfa*, que posibilita controlar el ancho de la red. Ajustar este parámetro implica un intercambio entre precisión y rendimiento. Un valor más pequeño de *alfa* disminuye la precisión, pero aumenta el rendimiento del modelo.

  Uno de los puntos más interesantes de estos modelos es que fueron entrenados utilizando el dataset *ImageNet* (descripto en la Sección 2.3). Gracias a esto, estos modelos son capaces de reconocer casi 1000 objetos diferentes (que abarcan desde objetos comunes del hogar y oficina, hasta diversos medios de transporte y una amplia variedad de alimentos).

  En la Figura 4.6, se presenta la respuesta generada al ejecutar este tipo de modelo sobre una imagen. La respuesta obtenida se representa como una colección de diccionarios que constan de dos claves principales. La primera clave, "*className*", identifica el nombre del objeto detectado, mientras que la segunda clave, "*probability*", indica el nivel de certeza asociado a la detección. Cabe destacar que los resultados se organizan de manera ascendente según el grado de certeza, proporcionando así una visualización clara y ordenada de las predicciones realizadas por el modelo.

```
[{
  className: "Egyptian cat",
  probability: 0.8380282521247864
}, {
  className: "tabby, tabby cat",
  probability: 0.04644153267145157
}, {
  className: "Siamese cat, Siamese",
  probability: 0.024488523602485657
}]
```

Figura 4.6: Respuesta obtenida al ejecutar *MobileNet* de *Tensorflow.js*.

- *Modelos de detección de objetos*: estos modelos no solamente tienen la capacidad de reconocer objetos en imágenes, sino que también son capaces de identificar dónde

[29] Documentación de modelos de *Tensorflow.js*: https://www.tensorflow.org/js/models?hl=es-419 (último acceso: 27/10/2023).
[30] Documentación del modelo *Mobilenet* de *Tensorflow.js*: https://github.com/tensorflow/tfjs-models/tree/master/mobilenet (último acceso: 27/10/2023).

se encuentran ubicados (como se menciona en la Sección 2.1). *Tensorflow.js* provee una librería que permite ejecutar tres modelos basados en SSD[31] que fueron entrenados utilizando el dataset *Microsoft COCO (*descripto en la Sección 2.3). Esta librería denomina a estos modelos de la siguiente: *mobilenet_v1*, *mobilenet_v2* y *lite_mobilenet_v2*. La decisión del modelo a utilizar afecta tanto en los tiempos de inferencia como en los niveles de precisión en las detecciones. De acuerdo con la documentación oficial, *lite_mobilenet_v2 es* el modelo predeterminado, ya que se destaca por su tamaño reducido y la velocidad de inferencia rápida.

Cuando estos modelos se ejecutan con una imagen específica, la respuesta generada se presenta como una colección de clases (class), posiciones (bbox) y probabilidades (score), como se muestra en la Figura 4.7.

```
[{
  bbox: [x, y, width, height],
  class: "person",
  score: 0.8380282521247864
}, {
  bbox: [x, y, width, height],
  class: "kite",
  score: 0.74644153267145157
}]
```

Figura 4.7: Respuesta obtenida al ejecutar modelos basados en *SSD* de *TensorFlow.js.*

Al comparar las Figuras 4.6 y 4.7 se puede notar una diferencia notable, y es que los modelos de detección, a diferencia de los modelos de reconocimiento, sí incluyen información de la ubicación específica de los objetos en la imagen.

Considerando las dos alternativas presentadas, se optó por que la aplicación prototípica incluyera ambas opciones. En términos generales, el desarrollo de la aplicación fue bastante ágil, tomando solo unas pocas horas. Esto refleja lo sencillo que resultó configurar e inicializar ambos tipos de modelos. El punto más crítico en el desarrollo fue quizás implementar la visualización de los *bounding boxes* en la interfaz, utilizando los resultados obtenidos de los modelos de detección.

En la Figura 4.8 se muestran dos capturas de pantalla de la aplicación prototípica desarrollada, tomadas desde un Samsung S22. En la Figura 4.8.a se pueden observar los resultados obtenidos con el modelo de reconocimiento *MobileNet versión 2* (*mobilenet_v2*), donde la aplicación lista los objetos detectados junto con sus respectivas probabilidades. Por otro lado, la Figura 4.8.b muestra el resultado de ejecutar el *modelo de detección* basado en SSD, también utilizando *mobilenet_v2*. En este caso, la aplicación representa gráficamente la información de los *bounding boxes* que enmarcan los objetos detectados, además de mostrar el nombre del objeto junto a las probabilidades asociadas.

[31] Modelo COCO-SSD para *Tensorflow.js*: https://github.com/tensorflow/tfjs-models/blob/master/coco-ssd/README.md (último acceso: 27/10/2023).

a. Modelo de reconocimiento **b. Modelo de detección**

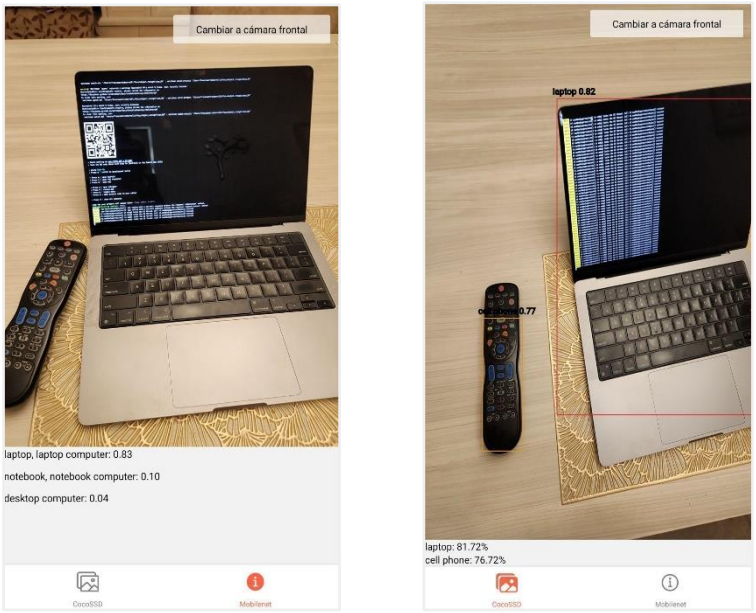


Figura 4.8: Aplicación prototípica implementada con *Tensorflow.js* y dos modelos pre-entrenados.

Para llevar a cabo las pruebas, se eligieron los mismos dos dispositivos utilizados en las pruebas de TensorFlow Lite: el Samsung S22 y el Samsung A20. Específicamente, se buscó evaluar los tiempos de inferencia de ambos tipos de modelos pre-entrenados de *TensorFlow.js*, utilizando todas las configuraciones disponibles.

La Figura 4.9 presenta los resultados obtenidos al ejecutar en ambos dispositivos los *modelos de reconocimiento* disponibles para *Tensorflow.js* sobre la aplicación prototípica. El gráfico presentado en la figura muestra el tiempo promedio (medido en milisegundos) que demoró cada modelo en lograr dar una respuesta de inferencia a partir de una imagen. Cabe señalar que para la configuración de los modelos se mantuvo el parámetro alfa en su valor por defecto (1), el cual otorga el mayor nivel de precisión.

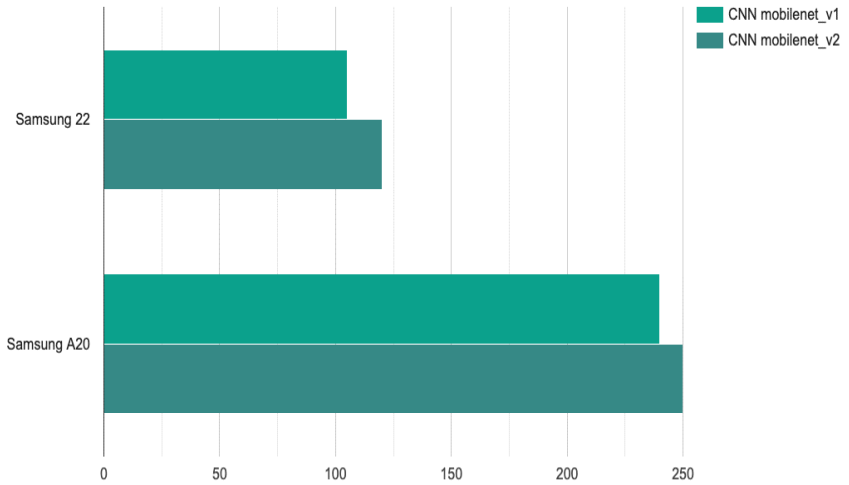


Figura 4.9: Resultados del tiempo de inferencia (en milisegundos) de *modelos de reconocimiento* con *Tensorflow.js*.

Por otra parte, en la Figura 4.10 se presentan los resultados obtenidos de la ejecución de la aplicación prototípica utilizando los *modelos de detección* previamente mencionados. En esta figura se presenta un gráfico con características similares al presentado en la Figura 4.9.

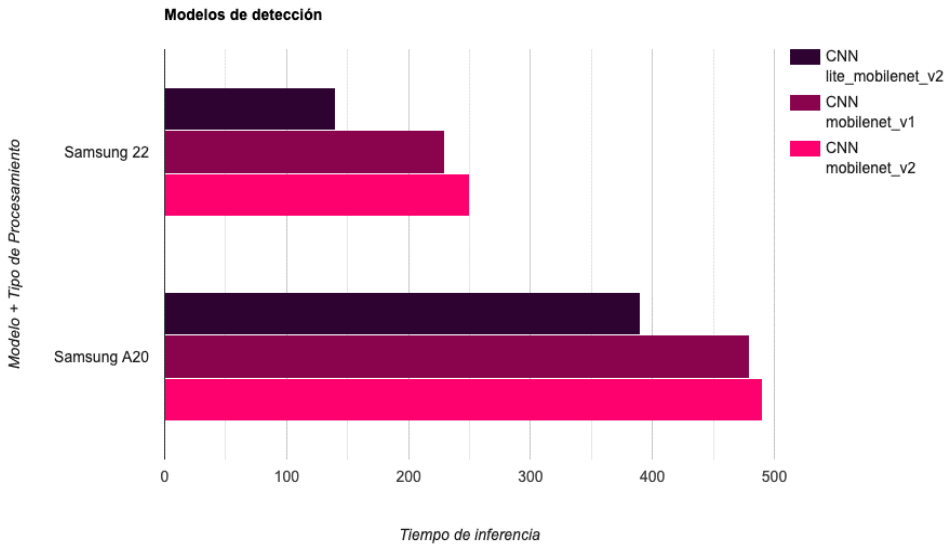


Figura 4.10: Resultados del tiempo de inferencia (en milisegundos) de *modelos de detección* en *Tensorflow.js*.

Al realizar una comparación entre las Figuras 4.9 y 4.10, se evidencia que los tiempos de inferencia se duplicaron cuando se usan *modelos de detección*. Por otro lado, se puede observar que al comparar los resultados obtenidos con *Tensorflow Lite* (presentados en la Sección 4.1.1), los tiempos de inferencia con *Tensorflow.js* fueron mayores, aunque se mantienen en un rango aceptable (por debajo de los 650 ms).

4.2 Entrenamiento de un modelo propio

En esta sección se exploran dos opciones para entrenar un modelo de detección de objetos propio. La primera se basa en el uso directo de *TensorFlow* con *transfer learning*, una técnica que se mencionó en los capítulos anteriores que aprovecha modelos pre-entrenados para mejorar la eficiencia del entrenamiento en tareas específicas. La segunda opción se centra en la utilización de *Microsoft Custom Vision*, un servicio en la nube que ofrece una interfaz intuitiva y funciones automatizadas para simplificar y agilizar el proceso de entrenamiento, etiquetado y despliegue de modelos.

4.2.1 Entrenamiento con *Tensorflow* y *transfer learning*

La implementación de un modelo de detección implica varios desafíos significativos. En primer lugar, es esencial contar con un *dataset* de imágenes representativas de los objetos que se quiere que el modelo reconozca. Un *dataset* bien diseñado debe incluir una amplia gama de escenarios, iluminaciones y condiciones. Esto permite que el modelo generalice de manera efectiva y no se vea restringido a situaciones específicas.

Dados los pocos conocimientos en esta área, se tomó la decisión de (en una fase inicial) entrenar un modelo capaz de identificar un tipo de objeto específico. Esto facilita la

familiarización con el proceso de implementación y prueba del modelo de una forma más controlada, permitiendo luego una expansión hacia la identificación de objetos más diversos y específicos. Teniendo en cuenta que la intención era realizar pruebas in-situ dentro de la Facultad de Informática de la UNLP, se eligió un objeto lo suficientemente común a dicho espacio. Por consiguiente, se definió que el modelo tendría la capacidad de reconocer *puertas*.

Otro desafío importante radica en etiquetar de forma precisa las imágenes en el *dataset*. Este proceso implica asignar correctamente las categorías o clases a cada objeto presente en las imágenes, lo cual puede ser laborioso y propenso a errores humanos. La calidad de estas etiquetas afecta directamente la capacidad del modelo para aprender y generalizar de manera precisa. Con el fin de simplificar este proceso, se optó por buscar en Internet un *dataset* estandarizado que tuviera imágenes de puertas debidamente etiquetadas. De esta forma se encontró un *dataset*[32] que superaba las 2000 imágenes y con alrededor de 6000 puertas ya etiquetadas.

Para enriquecer un poco más este *dataset de puertas*, se decidió incluir también imágenes tomadas sobre los pasillos de la Facultad de Informática de la UNLP, que fueron previamente etiquetadas utilizando la herramienta *LabelImg*[33]. En la Figura 4.11 se pueden observar algunas de las imágenes incorporadas al *dataset*. Se debe mencionar que previo al entrenamiento, algunas imágenes del *dataset* fueron separadas para poder utilizarlas luego en las etapas de validación y de prueba.

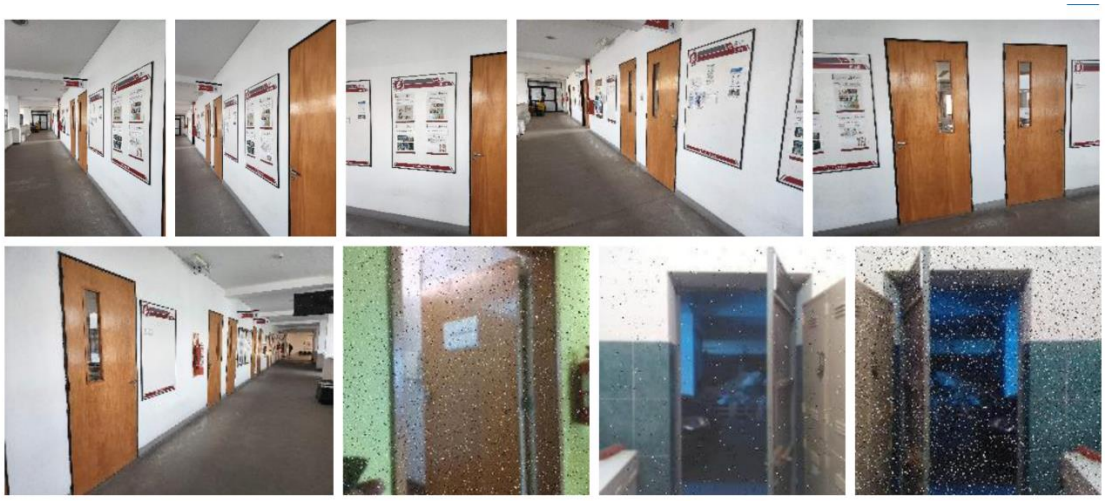


Figura 4.11: Imágenes dentro del *dataset* de entrenamiento generado.

Por otro lado, un desafío adicional es la elección del modelo que se utilizará de base. Como se menciona en el Capítulo 3, *Tensorflow* provee múltiples modelos pre-entrenados, cada uno de los cuales tiene sus particularidades, las cuales se ven reflejadas en la velocidad de inferencia y en la mAP. Teniendo en cuenta que en esta etapa inicial se va a resolver un problema de detección de objetos de clase única, el modelo más rápido como es *SSD MobileNet v2 320x320* debería ser suficiente.

[32] Dataset de puertas encontrado: https://universe.roboflow.com/mohammed-naji/doors-6g8eb/dataset/1 (último acceso: 31/10/2023).
[33] Documentación de *LabelImg*: https://github.com/HumanSignal/labelImg (último acceso: 31/10/2023).

Por último, la fase de entrenamiento del modelo también presenta desafíos, ya que es necesario ajustar correctamente los hiper-parámetros y controlar el sobreajuste para lograr un rendimiento óptimo. Esto requiere tener bastante conocimiento de la teoría subyacente y la capacidad de interpretar los resultados para realizar ajustes necesarios.

Considerando todos los aspectos mencionados, se optó por iniciar el entrenamiento del modelo propuesto. *Tensorflow* proporciona en su documentación[34] una plantilla de código escrita en *Python* que puede servir como punto de partida para el entrenamiento mediante *transfer learning.* Aunque esta plantilla está diseñada para simplificar el proceso de entrenamiento, se requiere realizar ciertos ajustes para adaptarla a las necesidades específicas del modelo que se está entrenando. La ejecución de la plantilla y el proceso de entrenamiento del modelo se llevaron a cabo de manera integral en la plataforma *Google Colab*[35].

En la Figura 4.12 se pueden observar algunas imágenes que se usaron para el testeo del modelo al ejecutarlo sobre *Google Colab*. Al evaluar todas las imágenes de prueba, el modelo logró alcanzar una precisión media del 78%.

Figura 4.12: Resultados obtenidos con el modelo pre-entrenado usando *Google Colab*.

Se debe destacar que, aunque se logró entrenar y ejecutar el modelo con éxito utilizando *Google Colab* en la computadora, se encontraron dificultades al intentar hacerlo funcionar tanto en *Tensorflow Lite* como en *Tensorflow.js*. Este modelo se logró exportar para ambas opciones, y fue integrado a las aplicaciones presentadas en las Secciones 4.1.1 y 4.1.2 respectivamente. Sin embargo, cada una de las aplicaciones cuando usaba estos modelos se congelaba sin razones aparentes.

4.2.2 Entrenamiento con *Microsoft Custom Vision*

Teniendo en cuenta el inconveniente mencionado en la Sección 4.2.1, se optó por investigar otras alternativas para entrenar un modelo. Durante esta búsqueda se dio con la existencia

[34] Entrenamiento por *Transfer Learning con Tensorflow*: https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html (último acceso: 31/10/2023).
[35] Página oficial de *Google Colab*: https://colab.research.google.com/ (último acceso: 31/10/2023).

de varios servicios en la nube como: *Microsoft Custom Vision*[36], *Amazon Sagemaker*[37] y *Google AI Platform*[38], los cuales permiten entrenar y exportar modelos a *Tensorflow*.

En líneas generales, estos servicios (*Microsoft Custom Vision, Amazon Sagemaker y Google AI Plataform*) buscan simplificar considerablemente el proceso de entrenamiento y despliegue de modelos de aprendizaje automático. Su principal objetivo es hacer que la tecnología de aprendizaje automático sea más accesible al eliminar obstáculos para aquellos que no tienen mucha experiencia en frameworks como *Tensorflow*. En lugar de requerir conocimientos profundos de programación y configuración técnica, estos servicios ofrecen interfaces intuitivas y herramientas automatizadas para que cualquier persona, independientemente de su nivel de experiencia, pueda aprovechar las capacidades de aprendizaje automático de manera más rápida y eficiente. El objetivo es permitir que los desarrolladores se centren en el diseño y mejora del modelo, sin preocuparse por configuraciones técnicas específicas.

Aunque estos servicios suelen ser pagos, a menudo ofrecen periodos de prueba gratuitos o créditos disponibles para estudiantes. Es precisamente por esta razón que se eligió probar y utilizar *Microsoft Custom Vision*.

Para entrenar un modelo en *Microsoft Custom Vision*, el primer paso es crear un nuevo proyecto. Después de la configuración inicial, el siguiente paso implica cargar las imágenes relevantes para las categorías que se desean reconocer. Algo interesante de *Microsoft Custom Vision* es que es muy fácil ver y etiquetar las imágenes cargadas. Al abrir una imagen, la interfaz ofrece sugerencias automáticas de objetos para etiquetar, agilizando de manera notable el procedimiento. Un ejemplo de este proceso se puede apreciar en la Figura 4.13.

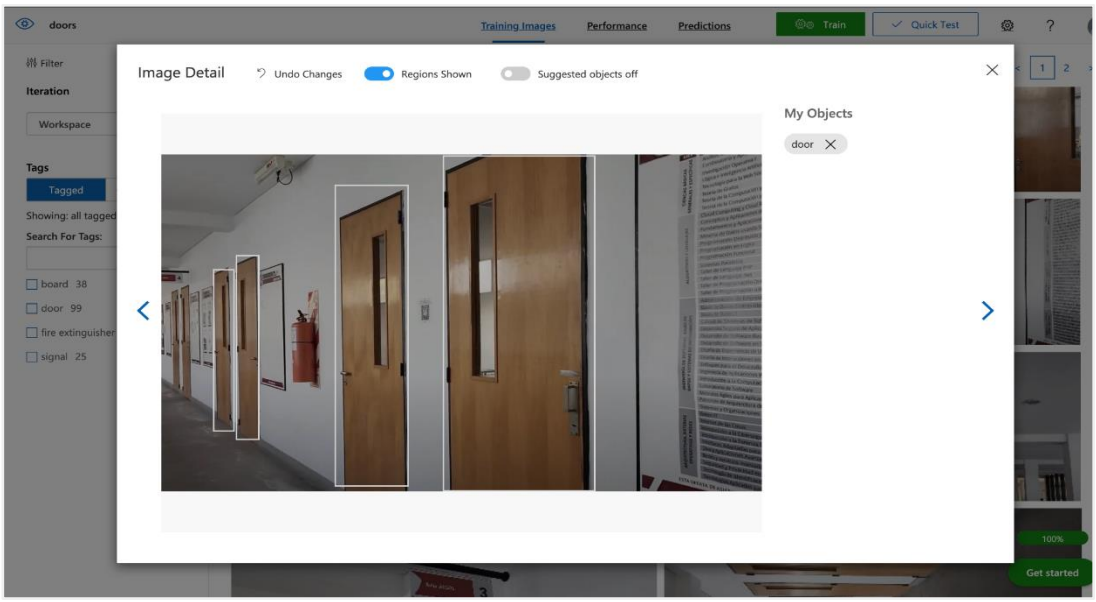


Figura 4.13: Etiquetado de imágenes en *Microsoft Custom Vision*.

[36] Página de *Microsoft Custom* Vision: https://azure.microsoft.com/es-es/products/ai-services/ai-custom-vision (último acceso: 31/10/2023).
[37] Página oficial de *Amazon Sagemaker*: https://aws.amazon.com/es/sagemaker/ (último acceso: 31/10/2023).
[38] Página oficial de *Google AI Platform*: https://cloud.google.com/ai-platform/docs/technical-overview?hl=es-419 (último acceso: 31/10/2023).

Una vez etiquetadas las imágenes, se inicia el proceso de entrenamiento del modelo haciendo clic en la opción "*Train*". Después del entrenamiento, el cual demora varios minutos, se evalúa el modelo utilizando un *dataset* de validación. Las métricas de rendimiento proporcionadas por *Custom Vision*, como la precisión y el recall, ayudan a comprender la eficacia del modelo. En la Figura 4.14 se pueden observar los resultados obtenidos para el modelo entrenado.

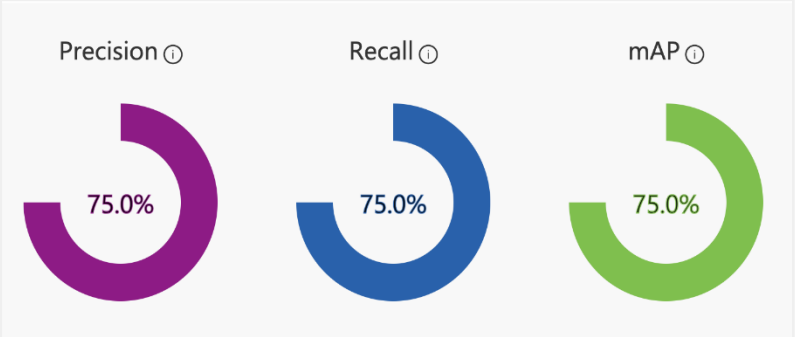


Figura 4.14: Resultados obtenidos al entrenar utilizando *Microsoft Custom Vision*.

En caso de ser necesario, se puede realizar una nueva iteración para mejorar el modelo ajustando parámetros, agregando más imágenes o realizando cambios en las etiquetas. Una vez satisfechos con el rendimiento del modelo, el mismo puede exportarse para utilizarlo en aplicaciones o proyectos específicos. Como se puede apreciar en la Figura 4.15, *Microsoft Custom Vision* provee soporte para *Tensorflow Lite* y *Tensorflow.js*.

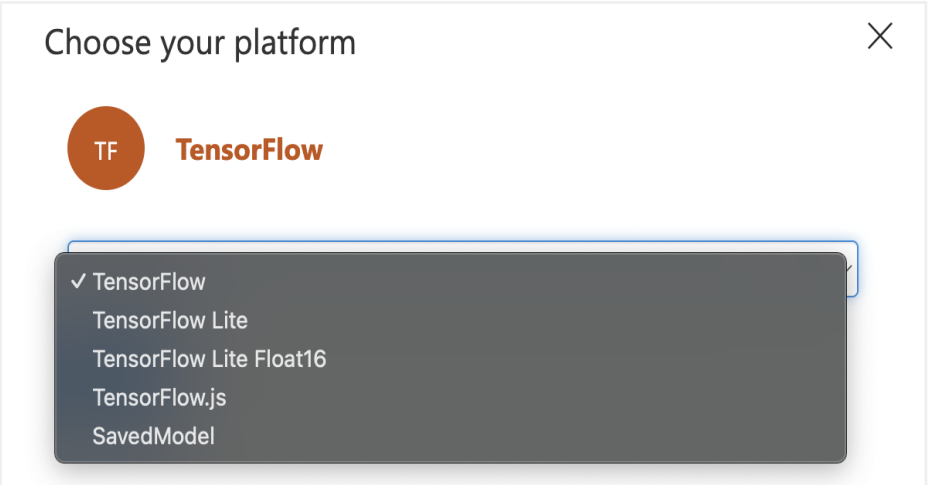


Figura 4.15: Exportación de modelos desde *Microsoft Custom Vision*.

Dado el conocimiento previo del autor de esta tesis en *JavaScript* y *React Native*, la decisión fue exportar y ejecutar el modelo entrenado en *TensorFlow.js*. Para esto, se exportó la versión del modelo entrenado para esta plataforma para luego integrarlo en la aplicación prototípica presentada en la Sección 4.1.2. A pesar de enfrentar ciertas complejidades durante la configuración y despliegue del modelo, debido a una configuración inicial incorrecta[39], finalmente se logró ejecutar con éxito el modelo desde un dispositivo móvil. En la Figura 4.16

[39] En particular, los problemas que acontecieron estuvieron relacionados a la integración y lectura del archivo del modelo en React Native.

se puede observar la aplicación funcionando en el *Samsung S22* y usando para la detección el modelo generado con *Microsoft Custom Vision*.

Figura 4.16: Modelo generado en ejecución usando *Tensorflow.js* desde el *Samsung S22*.

Es importante mencionar que el tiempo de inferencia promedio del modelo generado con *Microsoft Custom Vision* se situó en 285ms en el *Samsung S22* y de 520ms en el *Samsung A20*, demostrando viabilidad de uso en ambos dispositivos.

4.3 Análisis de las pruebas realizadas

A lo largo de este capítulo, se presentaron distintas pruebas que tenían como objetivo evaluar el rendimiento de *TensorFlow Lite* y *TensorFlow.js* sobre dispositivos móviles específicos. Durante estas pruebas, se pudo observar cómo ambas opciones fueron capaces de ejecutar distintos modelos, demostrando tiempos de inferencia adecuados en la mayoría de los casos. Estos resultados destacan la eficacia tanto de *TensorFlow Lite* y *TensorFlow.js* y sugieren que cualquiera de las dos opciones podría ser utilizada sin inconvenientes sobre los dispositivos móviles.

La elección entre *TensorFlow Lite* y *TensorFlow.js* en este trabajo de tesis es evidente que no puede tomarse en base a esta característica. Es por esto que entra en juego otro factor crucial a considerar, y en este caso, es la experticia con las tecnologías que utilizan estas herramientas. Como se mencionó anteriormente, el autor de esta tesis cuenta con experiencia en el desarrollo tanto de sistemas web como de aplicaciones móviles (híbridas) con *Javascript*; por esta razón, *Tensorflow.js* resulta la opción más adecuada para continuar con el desarrollo que se va a realizar acorde al objetivo del presente trabajo.

Cabe destacar que las pruebas realizadas en este capítulo no tuvieron en cuenta ningún aspecto de posicionamiento del usuario. Esto será abordado como parte de la solución propuesta en esta tesis en el siguiente capítulo.

5. Diseño de solución para el posicionamiento por objetos

En este capítulo se presenta el diseño de una arquitectura que se focaliza en plasmar cómo se podría llevar a cabo el posicionamiento en espacios indoor utilizando modelos de reconocimiento y detección de objetos, de una manera genérica. La solución propuesta en relación al posicionamiento por objetos está diseñada para funcionar de forma desacoplada, pudiendo ser embebida en diferentes tipos de aplicaciones. Además, se detalla cómo fue implementada esta solución.

5.1 Consideraciones iniciales

De acuerdo a lo analizado en la Sección 2.6, la solución propuesta en este capítulo utiliza el enfoque de *vision based indoor positioning* conocido como "*referencia en base a objetos"*. Como ya se menciona en la Sección 2.5, este método se caracteriza por el uso de bases de datos "*contextuales*" que registran información sobre los objetos en el espacio físico. Esta información se enriquece con datos adicionales para facilitar el posicionamiento del usuario. Cabe destacar que hasta el momento no hay una solución genérica que permita dar solución a este enfoque, y esta tesis aporta en esta dirección.

Teniendo en cuenta lo antes mencionado, para implementar la solución es necesario contar con dos aspectos claves:

- Por un lado, se requiere un mecanismo que permita reconocer los objetos presentes en el espacio físico. Esta tarea se puede abordar mediante el uso de modelos de reconocimiento y de detección de objetos (como los presentados en el Capítulo 4), los cuales son capaces de identificar una amplia variedad de objetos.
- Por otro lado, es fundamental contar con más información para que a partir de los objetos detectados se pueda ubicar al usuario en un determinado espacio físico. En este trabajo se elige descartar tecnologías que requieran la instalación de una infraestructura previa en el lugar, como *fingerprinting* con redes WiFi o el uso de *beacons* (analizadas en la Sección 2.5). Por esta razón, se escogió explorar el GPS, que si bien no cuenta con buena precisión en espacios indoor, es un sensor presente en la gran mayoría de los *smartphones*. Se espera que al combinar el GPS con el mecanismo que permita reconocer objetos (presentes en el espacio físico) se logre posicionar al usuario en un espacio indoor con una precisión lo más real posible.

Para lograr que ambos aspectos mencionados anteriormente (reconocer objetos y GPS) funcionen para posicionar al usuario por objetos, se necesitan dos estadíos, primero *crear* y luego *usar*. A continuación, se detalla lo que implica cada uno.

- *Crear*: es necesario que la solución provea un mecanismo que permita a los usuarios registrar in-situ datos en la base de datos "*contextual*", la cual almacenará información detallada sobre cada objeto detectado (y que sea relevante), vinculando cada uno con su respectiva posición espacial y otros datos de interés. Esta posición podría ser tomada con cualquier mecanismo de posicionamiento (en particular en esta tesis se explorará el GPS). La solución debe ser flexible y extensible para poder ir enriqueciendo los datos que se podrían cargar en un futuro.

- *Usar*: una vez construida la *base de datos contextual de objetos*, se podrá aprovechar la información disponible para posicionar a un usuario. Cuando el usuario apunta la cámara del dispositivo hacia un objeto, se determina si el objeto se encuentra registrado en la *base de datos contextual* y si, además, el usuario se encuentra "cerca" de la posición registrada para ese objeto. De esta forma, en caso de acierto, una aplicación (que usa este tipo de posicionamiento) puede, por ejemplo, proporcionar información relevante o servicios específicos asociados con esa posición o con ese objeto reconocido.

5.2 Arquitectura propuesta para el posicionamiento por objetos

En esta sección se analiza en detalle la arquitectura propuesta para lograr el posicionamiento mediante el reconocimiento de objetos. Como se puede apreciar en la Figura 5.1 el componente principal de esta arquitectura recibe el nombre de "*Posicionamiento por objetos*". Este posicionamiento funciona de forma completamente desacoplada de la lógica de las aplicaciones que lo usan, como se puede apreciar en la figura. Es decir, es una librería que puede ser embebida en diferentes tipos de aplicaciones.

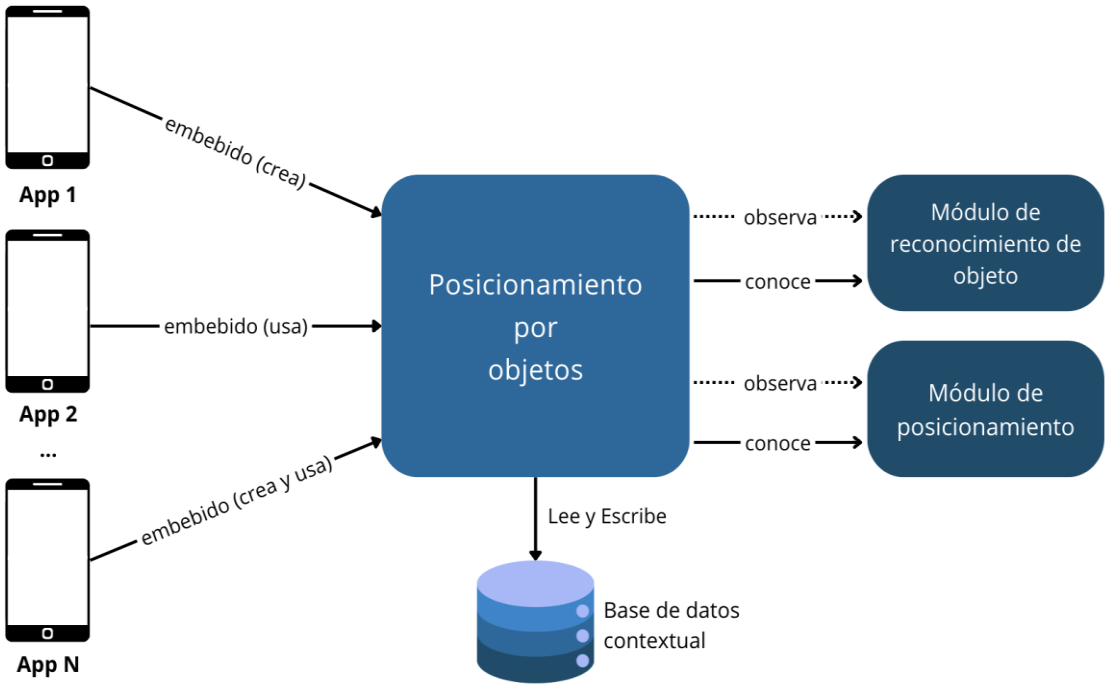


Figura 5.1: Arquitectura propuesta para el *Posicionamiento por objetos*.

Como se puede observar en la Figura 5.1 podrían desarrollarse aplicaciones que embeban al "*Posicionamiento por objetos*" para cargar la base de datos contextual (*crear*), otras para acceder a los datos previamente cargados (*usar*) y así hacer uso de este mecanismo para posicionar a un usuario, o incluso se podría contar con aplicaciones que hagan uso de ambas funcionalidades (*crear y usar*).

Es importante mencionar que la base de datos contextual podría ser local o no al dispositivo, dependiendo de las necesidades y de la complejidad de lo que se quiera implementar.

Por otro lado, en la Figura 5.1 se puede observar que el componente "*Posicionamiento por objetos*" *conoce* y *observa* dos módulos: uno de reconocimiento de objetos y otro de

posicionamiento. El primero tiene la responsabilidad de llevar a cabo el reconocimiento de los objetos presentes en el entorno mediante la cámara del teléfono, mientras que el segundo se encarga de recopilar información relativa a la ubicación del usuario. Se indica en la arquitectura que el "*Posicionamiento por objetos*" *observa* a los módulos porque está constantemente "escuchando" a cualquier cambio que acontece (ya sea un nuevo objeto detectado o una nueva posición). Pero también el "*Posicionamiento por objetos*" puede comunicarse directamente con los módulos (mediante la relación *conoce*) para solicitar información, lo cual suele ocurrir como respuesta a una acción por parte del usuario.

5.2.1 Módulo de reconocimiento de objetos

En la Figura 5.2 se presenta una representación simplificada que permite explicar cómo funciona el *módulo de reconocimiento de objetos*.

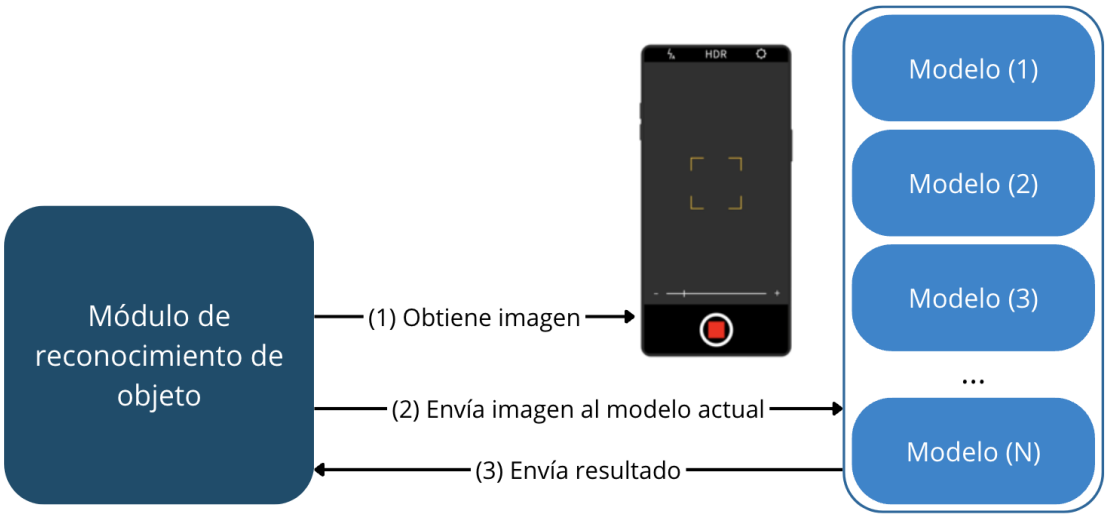


Figura 5.2: Funcionamiento del módulo de reconocimiento de objetos.

Como se observa en la Figura 5.2, el funcionamiento de este módulo cuenta con tres pasos, los cuales se detallan a continuación:

1. *Obtiene imagen*. Este paso implica establecer una comunicación con la cámara del teléfono. De esta forma, el *módulo de reconocimiento de objetos* puede obtener una captura de la vista actual de la cámara. Con el objetivo de optimizar el procesamiento y el rendimiento de los dispositivos, este módulo puede considerar sólo uno de cada N frames capturados por la cámara siguiendo la estrategia planteada en [Al-Azzoa et al., 2018].
2. *Envía imagen al modelo actual*. En este paso se envía la imagen capturada a uno de los modelos de reconocimiento o detección para su procesamiento. En función del modelo elegido, podría ser necesario realizar algún tipo de procesamiento o transformación en esta etapa para adecuar la imagen al formato esperado por el modelo. El *módulo de reconocimiento de objetos* puede conocer a varios modelos, aunque en un momento de tiempo determinado solo uno de ellos se encontrará en ejecución.

   Al momento de usar el "*Posicionamiento por objetos*" se podrá configurar el *módulo de reconocimiento de objetos,* eligiendo el modelo a utilizar y especificando qué

objetos dentro del mismo se consideran de interés durante el proceso de inferencia. Esto es necesario, ya que no todos los objetos del modelo poseen la misma relevancia en el contexto en el que se utiliza este tipo de posicionamiento.

3. *Envía resultado*. Se recibe el resultado de la inferencia del modelo en ejecución. Como se ejemplifica en las Figuras 4.6 y 4.7 (del Capítulo 4), el formato y la estructura de la respuesta obtenida en la inferencia podrían variar de un modelo a otro. Por esta razón, el *módulo de reconocimiento de objetos*, antes de devolver una respuesta final, se encargará de estandarizar la respuesta a una interfaz común, facilitando así la comunicación con distintos *modelos de reconocimiento y/o de detección de objetos*.

5.2.2 Módulo de posicionamiento

El objetivo del *módulo de posicionamiento* es obtener información acerca de la ubicación del usuario. Esta información podría obtenerse a través de diferentes mecanismos de sensado de posicionamiento, como se puede apreciar en la Figura 5.3.

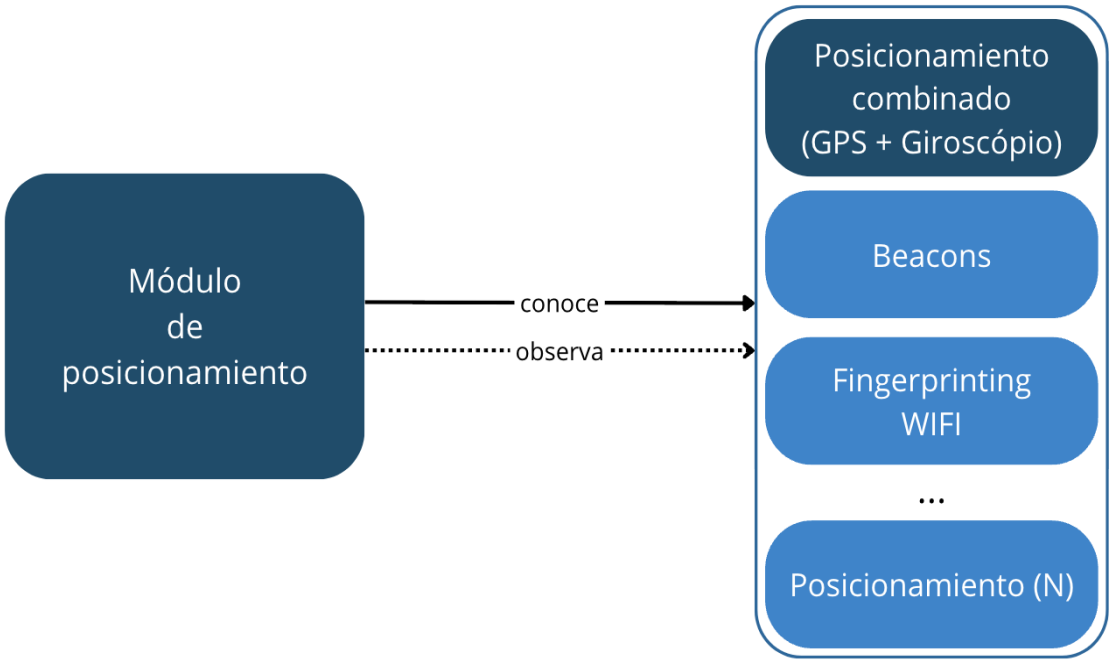


Figura 5.3: Representación gráfica del módulo de posicionamiento.

Como se puede observar en la Figura 5.3, el *módulo de posicionamiento* está siempre atento (*observa*) a los cambios que se producen en el *mecanismo de sensado de posicionamiento* seleccionado. Esta constante observación le permite detectar eventos importantes, como podría ser el desplazamiento del usuario. Además, el módulo tiene la capacidad de solicitar información actualizada (mediante la relación *conoce*) en momentos específicos, según sea necesario.

Dado que cada uno de los *mecanismos de sensado de posicionamiento* existentes (algunos detallados en la Figura 5.3) puede proporcionar una respuesta en un formato específico, el *módulo de posicionamiento* ofrece una capa de estandarización en relación con la respuesta que brinda. Esto garantizará coherencia y uniformidad en los datos de ubicación independientemente del mecanismo utilizado, lo que facilitará su uso.

Cabe mencionar que la elección del *mecanismo de sensado de posicionamiento* dependerá de varios factores, como la precisión requerida, la disponibilidad de infraestructura y las limitaciones del entorno. En esta tesis, particularmente, se utilizará como mecanismo una solución que combina la información obtenida de dos sensores: el GPS y el giroscopio.

Este mecanismo, al cual denominamos "*Posicionamiento combinado (GPS + Giroscopio)",* se grafica en la Figura 5.4.

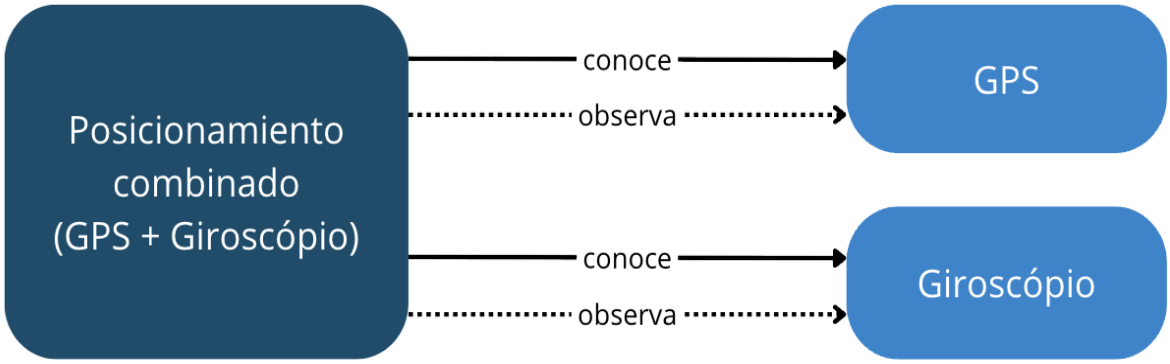


Figura 5.4: Representación gráfica del módulo de posicionamiento.

A continuación se detalla la información que se puede obtener a partir de estos sensores:

- *GPS*: proporciona información sobre la ubicación geográfica del teléfono, pero a veces sufre de imprecisiones, especialmente en entornos *indoor* o en presencia de obstáculos.
- *Giroscopio*: el giroscopio mide la orientación del teléfono en el espacio. Esto significa que se puede conocer la dirección en la que apunta la cámara. Muchas veces se combina con el GPS para dar mayor nivel de detalle a la posición del usuario, como se detalla en la Sección 2.5. Por otra parte, al combinar esta información con el *reconocimiento de objetos*, se podrían resolver situaciones en las que hay varios objetos similares cercanos, y desambiguar entre ellos.

5.3 Registro de evento y uso de métricas

Como se ha planteado anteriormente, otro de los objetivos de esta tesis es recolectar datos de uso de la herramienta que, posteriormente, se puedan analizar para generar información nueva y valiosa. Acorde a esto, se define un nuevo componente denominado "*Monitoreo de Eventos*", el cual se puede observar en la Figura 5.5. Este monitoreo tiene como principal objetivo escuchar, procesar y registrar información relevante de distintos eventos que acontezcan en relación con el *Posicionamiento por objetos*.

Se puede apreciar en la Figura 5.5 que el "*Monitoreo de Eventos*" observa (y conoce) diversas fuentes, esto es extensible. Cada vez que acontece un evento de interés de una de estas fuentes, se registra en la *base de datos de métricas* la información recibida, o se realiza algún tipo de cálculo o procesamiento para generar nueva información antes de ser almacenada. Posteriormente, la información almacenada en esta base de datos puede ser consultada y procesada por diversas aplicaciones e interfaces web.

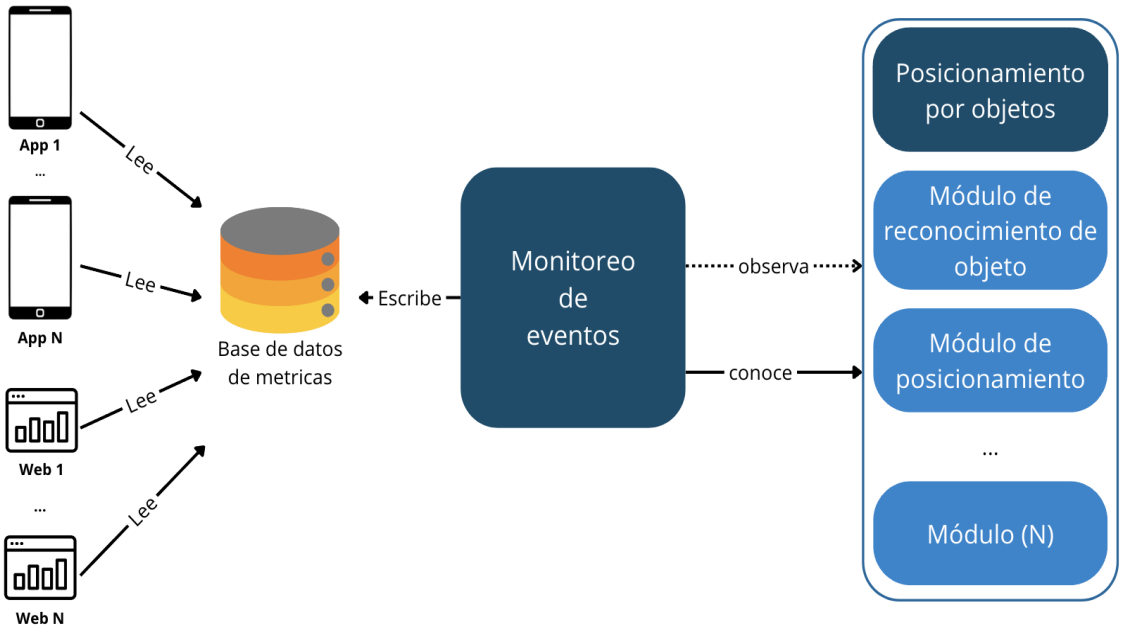


Figura 5.5: Representación gráfica del *Monitoreo de eventos*.

En particular, el "*Monitoreo de Eventos*" podría observar (y conocer) los componentes que conforman la arquitectura propuesta en la Sección 5.2, como son el "*Posicionamiento por objetos*", el "*Módulo de reconocimiento de objetos*" y el "*Módulo de posicionamiento*" como se aprecia en la Figura 5.5. Esto podría permitir, por ejemplo:

- Almacenar alguna acción desencadenada por el usuario, ya sea durante los estadíos de *creación* y/o *uso* del *Posicionamiento por objetos*.
- En cuanto al *módulo de reconocimiento por objetos,* se podrían registrar los objetos que se detectan con mayor frecuencia.
- Del *módulo de posicionamiento* se podrían registrar todas las posiciones del usuario.

El registro de eventos es de lo más variado y dependerá de qué métricas son útiles para cada aplicación.

5.4 Implementación propuesta

Las propuestas realizadas en las Secciones 5.2 y 5.3 son genéricas y pueden ser implementadas en cualquier lenguaje de programación. Sin embargo, como se menciona en la Sección 4.3, para este trabajo se decide utilizar el lenguaje de programación JavaScript. A continuación, se describe cómo se implementaron tanto *el Posicionamiento por objetos* y el *Monitoreo de eventos*.

5.4.1 Implementación del *Posicionamiento por objetos*

La arquitectura propuesta para el *Posicionamiento por objetos* es implementada como una librería para el framework *React Native* [ReactNative]. Este framework se destaca por permitir a los desarrolladores crear aplicaciones híbridas capaces de funcionar tanto para iOS como para Android utilizando un mismo código fuente. Esta capacidad de desarrollo multiplataforma facilita el proceso de creación de aplicaciones móviles.

Además, *React Native* tiene una robusta cantidad de módulos y librerías que facilitan la integración con funcionalidades nativas del teléfono. Algunas de estas librerías serán de gran ayuda para implementar los *módulos* de *reconocimiento de objetos y de posicionamiento*:

- Para implementar el *módulo de reconocimiento por objetos* (presentado en la Sección 5.2.1) se puede:
    - Utilizar la librería "*React Native Camera*", la cual permite acceder a la cámara del dispositivo.
    - En lo que respecta a la ejecución de los modelos, se opta por utilizar la librería específica de *Tensorflow.js* para *React Native*. Esto posibilita integrar tanto los modelos pre-entrenados descriptos en la Sección 4.1.2 como el modelo que ha sido entrenado mediante *Microsoft Custom Vision* detallado en la Sección 4.2.2.
- Para implementar el *módulo de posicionamiento* (presentado en la Sección 5.2.2), *React Native* cuenta con dos librerías:
    - *React Native Geolocation*: posibilita el acceso al GPS. Permitiendo obtener la posición actual del usuario y monitorear en tiempo real los cambios de la misma.
    - *React Native Sensors*: proporciona acceso a los sensores del dispositivo, como el acelerómetro, el giroscopio, el magnetómetro, entre otros.

Para la implementación de la *base de datos contextual*, se opta por utilizar *Firestore*[40], una base de datos NoSQL en la nube proporcionada por *Google Firebase*, a la cual se accede mediante una API. Esta decisión se fundamenta en varios puntos:

- Emplear bases de datos en la nube permite que la información almacenada pueda ser compartida entre diversos usuarios y dispositivos. Esto no es viable en una base de datos local del dispositivo. Además, *Firestore* evita la necesidad de implementar un servicio propio, así como la contratación de un servidor dedicado para ejecutar este servicio.
- En segundo lugar, *Firestore* se destaca por su eficiencia y escalabilidad. Es particularmente eficaz para manejar datos en tiempo real (tiene una latencia muy baja), algo que en el posicionamiento se vuelve fundamental.
- Otro factor determinante es la disponibilidad de un *Software Development Kit* (SDK) para múltiples plataformas, como iOS, Android, C++, Unity y JavaScript. En particular, la SDK para *JavaScript* permite implementar la conexión con *Firestore* de una manera ágil.
- Por otro lado, el autor de esta tesis contaba con conocimiento de *Firebase* (que incluye *Firestore*) al haberlo utilizado en trabajos previos como [Borrelli, 2021].

Es relevante señalar que *Firestore* es un servicio de pago que se basa en la tasa de transferencia de datos para asignar planes de pago a los usuarios. Sin embargo, ofrece

[40] Documentación de *Firestore*: https://firebase.google.com/docs/firestore?hl=es-419 (último acceso 12/11/2023).

planes gratuitos para realizar pruebas, y estos planes gratuitos se ajustaban perfectamente a los objetivos de esta tesis, permitiendo desarrollar y probar sin inconvenientes.

En la Figura 5.6 se detallan las tecnologías mencionadas anteriormente en relación con la implementación de la arquitectura propuesta para el *Posicionamiento por objetos*. En la figura se muestra que se interactúa (lee y escribe) con la "API de *Firebase*", la cual incluye a *Firestore.*

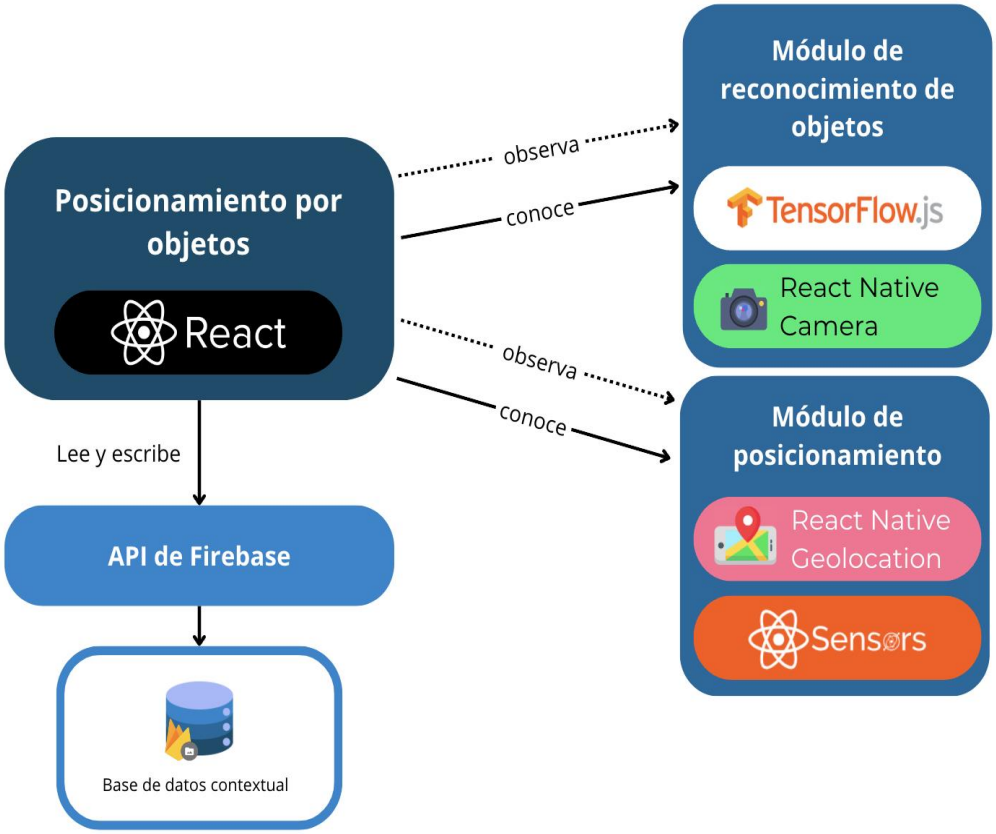


Figura 5.6: Tecnologías usadas en la implementación del *Posicionamiento por objetos*.

Asimismo, se debe destacar que la solución implementada no solo almacena la información vinculada a los *módulos de reconocimiento de objetos y posicionamiento*, sino que también permite la inclusión de datos contextuales adicionales. Esta funcionalidad se logra a través de un campo en los registros de *Firebase*, diseñado para almacenar cualquier otro tipo de información relevante. Esto resultará especialmente útil para futuras consultas a la base de datos, ya que facilitará la recuperación selectiva de datos. Por ejemplo, será posible recuperar únicamente los registros asociados a un piso específico dentro de un edificio.

5.4.2 Implementación del *Monitoreo de Eventos*

En esta tesis se implementa una versión simplificada del *Monitoreo de Eventos*, donde la función principal es observar y registrar eventos sin realizar procesamiento adicional. Además, los eventos serán solo los provenientes del componente "*Posicionamiento por objetos*" como se puede apreciar en la Figura 5.7. El monitoreo también está implementado en *React Native* y usa *Firestore* para la gestión de la base de datos de métricas a la cual se accede mediante la API de *Firebase* (en la base solo se escribe la información de los eventos).

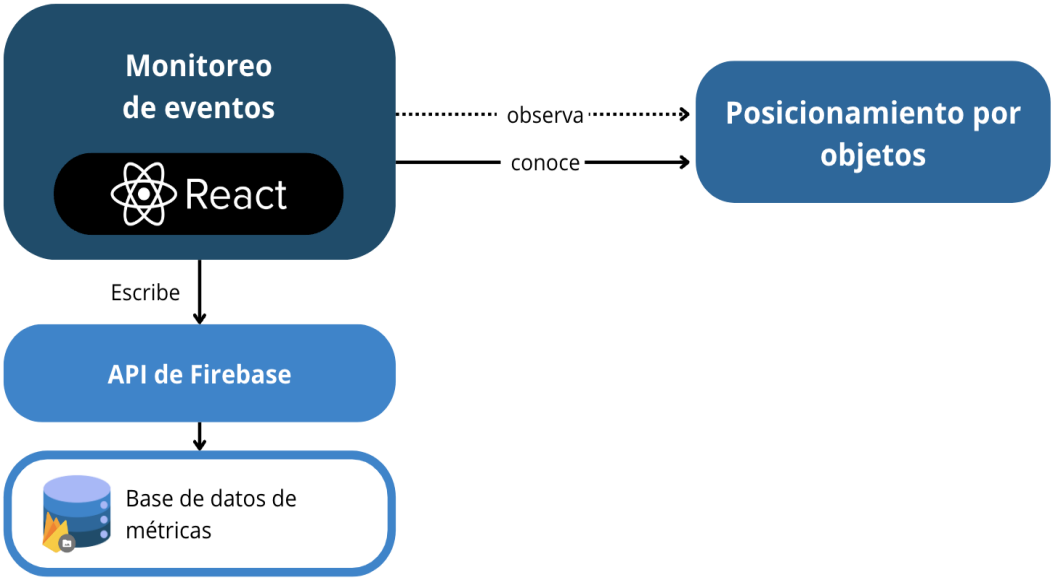


Figura 5.7: Tecnologías usadas en la implementación del *Monitoreo de eventos*.

La versión del *Monitoreo de eventos* presentada en la Figura 5.7 tiene la capacidad de registrar dos tipos específicos de eventos:

- *Evento de Detección*: este evento se registra cada vez que un usuario, utilizando la cámara del teléfono, logra identificar un objeto válido en la *base de datos contextual*.
- *Evento de Actualización*: periódicamente se registran datos sobre los últimos objetos detectados, la última posición conocida y el ángulo de visión de la cámara.

Estos eventos además se pueden configurar para incluir información contextual adicional. Por ejemplo, podría ser deseable para cada evento registrar nombre o versión de la aplicación o información del usuario que la está utilizando.

Cabe destacar también que esta versión del *Monitoreo de Eventos* está implementada para ser extensible en un futuro, por ejemplo, para registrar otros tipos de eventos.

5.5 Análisis del *Posicionamiento por Objetos* y *Monitoreo de Eventos*

En este capítulo se diseñó una arquitectura genérica para el posicionamiento de usuarios en espacios indoor utilizando modelos de reconocimiento y/o detección de objetos. Esta arquitectura está planteada para funcionar de forma desacoplada al resto de la lógica de la aplicación que la utiliza, buscando que esta solución pueda integrarse (embeberse) fácilmente en variados dominios de uso y tipos de aplicaciones.

En particular, para el *módulo de posicionamiento* se eligió explorar la combinación del GPS y el giroscopio. Esto no exige la instalación o presencia de infraestructura adicional en el espacio físico. Es decir, todo se resuelve desde el dispositivo, lo cual simplifica la utilización de esta arquitectura en cualquier lugar.

De hecho, la arquitectura está pensada para usar cualquier *modelo de reconocimiento y/o detección de objetos* genérico. Es decir, los objetos de estos modelos no están vinculados específicamente a un lugar en particular. Esto trae como ventaja que no hay necesidad de

entrenar los modelos para cada entorno, ya que los mismos modelos pueden ser reutilizados. A lo sumo se deben seleccionar cuáles son los objetos de interés de ese modelo.

La arquitectura genérica propuesta se implementó con tecnologías particulares, y la misma puede ser embebida en cualquier aplicación que tenga compatibilidad con *React Native*. También es posible, en un futuro, implementar la arquitectura en otras plataformas.

En cuanto al "*Monitoreo de Eventos*" se planteó tanto una solución genérica como una implementación simplificada orientada a registrar los eventos del *Posicionamiento por objetos,* que es el foco de interés de esta tesis. Este monitoreo también está planteado para funcionar de forma desacoplada al resto de la lógica de la aplicación que lo utiliza.

En el siguiente capítulo se aborda cómo las implementaciones descriptas ("*Posicionamiento por Objetos*" y "*Monitoreo de Eventos*") pueden ser integradas dentro de una aplicación concreta, en particular, una herramienta de co-diseño en alineación con los objetivos de esta tesis.

6. Posicionamiento por objetos embebido en una Herramienta para el co-diseño y co-testeo in-situ

En este capítulo se presenta una herramienta prototípica de co-diseño y co-testeo in-situ desarrollada que utiliza las implementaciones de la Sección 5.4 tanto del *Posicionamiento por objetos* como del *Monitoreo de eventos*.

Si bien el funcionamiento de las implementaciones del *Posicionamiento por objetos* y del *Monitoreo de eventos* podrían ser probadas en herramientas más sencillas, existe una motivación adicional para desarrollar una que facilite el co-diseño y co-testeo in-situ. Esta elección se justifica principalmente en su alineación con los objetivos del plan de investigación de la beca de maestría del autor de esta tesis.

6.1 Aspectos asociados al co-diseño

Como se menciona en el Capítulo 1, el autor de esta tesis cuenta con antecedentes en relación a la temática de herramientas de autor para el co-diseño [Challiol et al., 2020], [Borrelli, 2021] y [Borrelli, et al., 2021], los cuales pudieron ser aprovechados como base para este trabajo.

En el Anexo A se brinda un resumen que enmarca los aspectos que se describen a continuación, para las herramientas de autor para el co-diseño in-situ, acorde a la implementación puntual que se va a realizar en esta tesis:

- *Espacios de trabajo*: contiene toda la información que se co-diseña en una aplicación. Al ser in-situ, cada aplicación está vinculada a un edificio o lugar físico.
- *Plantilla*: al ser provistas por una herramienta, facilitan la creación de diferentes tipos de aplicaciones dentro de un *Espacio de trabajo.* Por ejemplo, la plantilla "*Información Posicionada*" requiere definir un nombre o título de la aplicación, una descripción, un color (que se utilizará en la interfaz al acceder al *Espacio de trabajo*) y podría tener otras configuraciones para el *reconocimiento de objetos* (tales como el modelo utilizado y los objetos de interés).
- *Perfiles de Usuario*: es beneficioso que una herramienta tenga la capacidad de incorporar diversos perfiles de usuarios, asignando a cada perfil permisos específicos para llevar a cabo acciones particulares. Por ejemplo, se podrían contar con tres perfiles:
    - Los *Creadores:* tienen la capacidad de establecer nuevos *Espacios de Trabajo* con la opción de agregar colaboradores. Además, pueden modificar los estados de dichos espacios.
    - Los *Colaboradores:* ayudan a co-diseñar contenido dependiendo del tipo de aplicación en la que estén participando. En el caso de la plantilla "*Información Posicionada*" completan en determinadas posiciones información relevante.
    - Los *Usuarios Finales:* sólo pueden visualizar la versión final de la aplicación co-diseñada.
- *Estados*: es fundamental considerar que el co-diseño de cada aplicación puede

atravesar diversas etapas o estados, dependiendo de sus características particulares. En cada una de estas fases, se van desarrollando, incorporando o decidiendo distintos aspectos específicos de la aplicación.

Por ejemplo, la plantilla "*Información Posicionada*" podría establecer 5 estados (y una herramienta de co-diseño debería permitir avanzar o retroceder a un estado anterior):

- *Estado inicial*: en este estado, solo el usuario con perfil *Creador* puede acceder y actuar sobre el *Espacio de Trabajo*.
- *Estado colaborativo*: en este estado, tanto el usuario con perfil *Creador* como los *Colaboradores* (asociados a este espacio) pueden definir in-situ información posicionada.
- *Estado decisivo*: después de que el grupo (conformado por el *Creador* y los *Colaboradores*) haya definido información posicionada, el *Creador* tiene la capacidad de hacer visible o no esta información en la versión final. Es importante destacar que la fase de acuerdo o consenso generalmente se lleva a cabo fuera de la propia herramienta. Esto implica una interacción activa y una discusión entre los miembros del equipo que participan en el *Espacio de trabajo*.
- *Estado de prueba (*co-testeo*)*: este estado permite al *Creador* y a los *Colaboradores* probar la aplicación y realizar los ajustes necesarios. Es decir, co-testean utilizando la aplicación y las métricas recolectadas.
- *Estado final*: los usuarios finales pueden probar la aplicación co-diseñada.

6.2 Implementación de la herramienta de autor

Al momento de implementar la herramienta de autor, se presentaron dos alternativas: adaptar una herramienta existente para incluir la solución de *Posicionamiento por objetos* o desarrollar una desde cero. La primera opción podría parecer lógica, especialmente considerando que el autor de este trabajo ha desarrollado herramientas similares en el pasado, como en [Borrelli, 2021] y [Borrelli, et al., 2022]. Sin embargo, las tecnologías utilizadas para desarrollarlas, muchas de las cuales ya no se utilizan actualmente, las vuelven poco escalables a futuro. Por ese motivo, se tomó la decisión de iniciar el desarrollo de una nueva herramienta.

Esta herramienta, se diseñó especialmente para usuarios no expertos, contando con una interfaz que permita el co-diseño y co-testeo in-situ de aplicaciones móviles sensibles al contexto, las cuales posteriormente podrán ser utilizadas desde la misma herramienta. Para el desarrollo de esta herramienta, se tuvieron en cuenta los conceptos presentados en la Sección 6.1. Cabe destacar, además, que en el marco de este trabajo, se tomó la decisión de centrarse en el co-diseño y co-testeo in-situ de un tipo de aplicación en particular capaz de brindar *Información Posicionada*, es decir, capaz de *posicionar por objetos* y mostrar información en relación con la posición detectada. La herramienta se diseña con la idea en mente de poder extenderla en el futuro para incluir otros tipos de aplicaciones.

La implementación de la herramienta utiliza distintas librerías y tecnologías, las cuales se presentan en la Figura 6.1. Asimismo, se puede observar como la herramienta utiliza las

soluciones presentadas de "*Posicionamiento por objetos*" y "*Monitoreo de eventos*", haciendo uso específico de las implementaciones detalladas en la Sección 5.4.

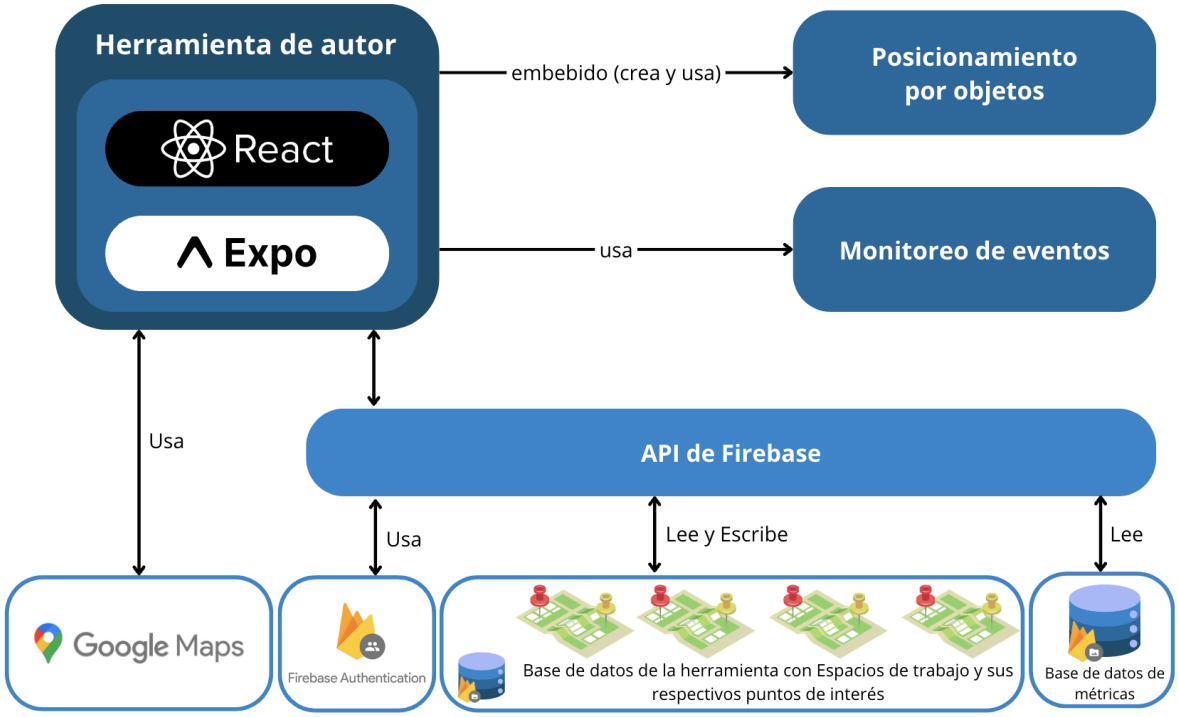


Figura 6.1: Arquitectura de la herramienta de autor desarrollada.

Viendo la Figura 6.1 se puede notar que la codificación de la herramienta se hizo utilizando *React Native* [ReactNative] y *Expo*[41]. Esta última (*Expo*) es una plataforma que simplifica el desarrollo y compilación de aplicaciones móviles, utilizando por debajo *React Native*. *Expo* agrega una capa adicional para hacer que el proceso de desarrollo sea más accesible y eficiente. Una de las principales características de *Expo* es que proporciona un entorno de desarrollo unificado. Esto significa que los desarrolladores pueden crear aplicaciones para iOS y Android desde un solo código base, evitando la necesidad de configurar entornos de desarrollo separados para cada plataforma.

De igual manera, se puede observar en la Figura 6.1 que la herramienta de autor hace uso de dos librerías:

- *Google Maps*[42]: en particular se usó la librería para *JavaScript*, la cual permite a los desarrolladores integrar mapas interactivos utilizando este lenguaje de programación. Además, ofrece funciones como visualización de distintos tipos de mapas, marcadores, información geográfica, trazado de rutas y servicios de geocodificación. Los desarrolladores pueden personalizar la apariencia y el comportamiento del mapa, así como superponer capas adicionales. Es ampliamente utilizada en aplicaciones para mostrar ubicaciones, rutas y datos geoespaciales. Para su utilización, es necesario contar con una clave de API de Google.

[41] Página oficial de *Expo*: https://expo.dev/ (último acceso 27/11/2023).
[42] Página oficial de la librería de *Google Maps para JavaScript*: https://developers.google.com/maps/documentation/javascript/overview (último acceso 27/11/2023).

Se utiliza la librería *Google Maps* para *JavaScript* para lograr la representación visual del espacio físico. Cabe destacar, que la herramienta se desarrolló para permitir cargar fácilmente planos y diagramas de los edificios (y sus diferentes pisos), para poder visualizarlos sobre el mapa.

- *Firebase*: en particular se usaron dos servicios provistos por la *API de Firebase*.
    - *Firebase Authentication*[43]: un sistema de autenticación que simplifica la gestión de usuarios y la implementación de funciones de inicio de sesión. Esto elimina la necesidad de programar estas funcionalidades desde cero, ahorrando tiempo y recursos.
    - *Firestore*: de la misma forma que se utiliza *Firestore* para la base de datos contextual del *Posicionamiento por objetos* y la base de datos de métricas del *Monitoreo de eventos*, en la herramienta se lo emplea para almacenar toda la información de los *Espacios de Trabajo* creados. Esto comprende, entre otras cosas, el registro de la información posicionada, los parámetros de configuración del modelo de reconocimiento de objetos, la lista de usuarios vinculados a un *Espacio de Trabajo*, entre otros datos de interés.

      En la Sección 5.4.1 se detallaron varias ventajas de *Firestore*, pero una en particular lo hace especialmente interesante para la herramienta desarrollada: su capacidad para mantener una sincronización constante mediante el uso de *observers*. Esta característica permite que *Firestore* identifique y notifique de manera prácticamente transparente cuando se producen cambios en los datos. Esto es fundamental para una aplicación en la que varias personas trabajan juntas y necesitan ver la información siempre actualizada.

      Para ilustrar este punto, consideremos el siguiente ejemplo: si un *Colaborador* realiza una modificación en un *Espacio de Trabajo* agregando una nueva información posicionada, el *Creador* de este espacio verá en su dispositivo automáticamente este nuevo elemento.

      Adicionalmente, la herramienta tiene acceso (de lectura) a la base de datos de métricas. De esta forma, es capaz de presentar información estadística y métricas a partir de los datos recolectados por el *Monitor de Eventos*.

6.3 Funcionamiento de la herramienta de autor

Para utilizar la herramienta de autor, los usuarios deben registrarse y autenticarse. Estos usuarios tienen la capacidad de crear sus propios *Espacios de trabajo* y colaborar en los de otros. Cuando un usuario inicia sesión en la herramienta, se realiza una validación para asegurarse de que sea un usuario registrado. Posteriormente, se accede a la base de datos de la herramienta en *Firestore* para obtener los *Espacios de trabajo* creados por este usuario, así como aquellos en los que participa como *Colaborador* o *Usuario final*.

De esta manera, la herramienta no solo cumple la función de crear, co-diseñar y coordinar entre el equipo de trabajo, sino que también puede ser accedida por usuarios que consumen la información de los *Espacios de Trabajo* una vez que estos alcanzan el *estado final*. Este enfoque permite aprovechar la misma aplicación (herramienta) para que los usuarios puedan

[43] Documentación oficial de *Firebase Authentication*: https://firebase.google.com/docs/auth (último acceso 27/11/2023).

participar en el co-diseño en algunos casos, mientras que en otros sólo actúan como *consumidores* de lo generado. Esta flexibilidad elimina la necesidad de desarrollar aplicaciones separadas, una dedicada al co-diseño y otra destinada al consumo de la información creada.

Otro punto a mencionar es que, al igual que la herramienta de autor presentada en [Borrelli, 2021], si bien la herramienta provee la posibilidad de poder cambiar de piso del edificio asociado a un *Espacio de trabajo*, el usuario de la herramienta debe realizar este cambio de forma manual. Aunque existen mecanismos que podrían explorarse para lograr este cambio de piso de manera automática, como el uso del sensor de presión barométrica (que puede medir la altitud y que se encuentra incorporado en algunos teléfonos), esto queda fuera del alcance de esta tesis.

A continuación, se presentan algunas pantallas de la herramienta desarrollada, destacando aquellas de mayor relevancia para las acciones de creación y uso en el contexto de la librería de *Posicionamiento por objetos* presentada. Para obtener información más detallada sobre otros aspectos de la herramienta se puede consultar el Anexo B.

6.3.1 Configuraciones asociadas al *Posicionamiento por objetos*

Cuando un usuario (de la herramienta) crea un *Espacio de Trabajo* utilizando la plantilla de *Información Posicionada*, puede configurar algunos parámetros relacionados al *Posicionamiento por objetos*. En la Figura 6.2 se presentan todas las configuraciones disponibles.

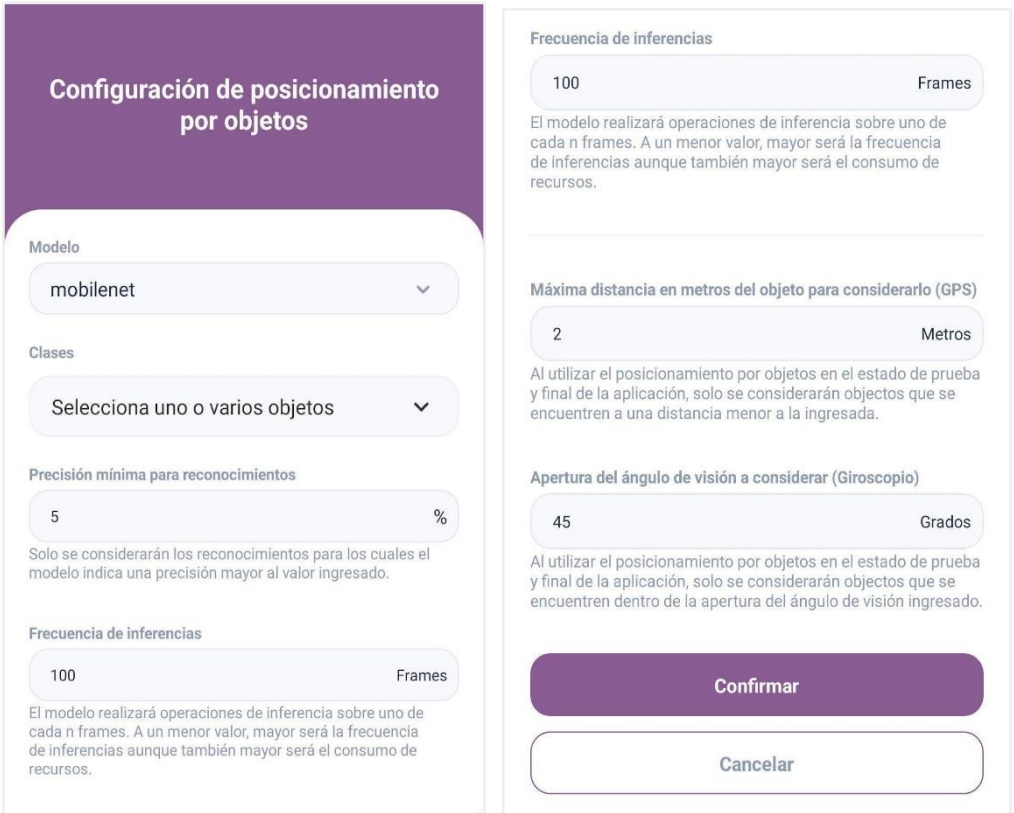


Figura 6.2: Pantalla de configuración para el *Posicionamiento por objetos*.

A continuación se explica cada una de las configuraciones de la Figura 6.2:

- *Modelo*: es posible elegir el *modelo de reconocimiento o detección de objetos* que se utiliza dentro del *Espacio de Trabajo*. En particular, la herramienta en su versión inicial permite utilizar los siguientes modelos:
    - El modelo *COCO-SSD* corresponde al modelo de detección explicado en la Sección 4.1.2, utilizando la arquitectura *lite_mobilenet_v2*. Este modelo es capaz de reconocer alrededor de 80 objetos.
    - Los modelos *MobileNet v1* y *MobileNet* v2 se corresponden al modelo de reconocimiento presentado en la Sección 4.1.2. Estos modelos fueron configurados con el valor alpha por defecto (1), el cual otorga el mayor grado de precisión, aunque sacrifica un poco el rendimiento. Ambos modelos son capaces de reconocer cerca de mil objetos.
    - El modelo "*Puertas*" hace referencia al modelo entrenado en la Sección 4.2.2 utilizando *Microsoft Custom Vision*.

- *Clases*: según el modelo seleccionado, el usuario puede elegir entre los objetos disponibles aquellos que son de interés en el *Espacio de Trabajo*. En la Figura 6.3, se presentan algunos de los objetos disponibles para el modelo *COCO-SSD*. Asimismo, se puede observar en la figura que, para facilitar el uso de la aplicación, se opta por organizar los objetos en distintas categorías.

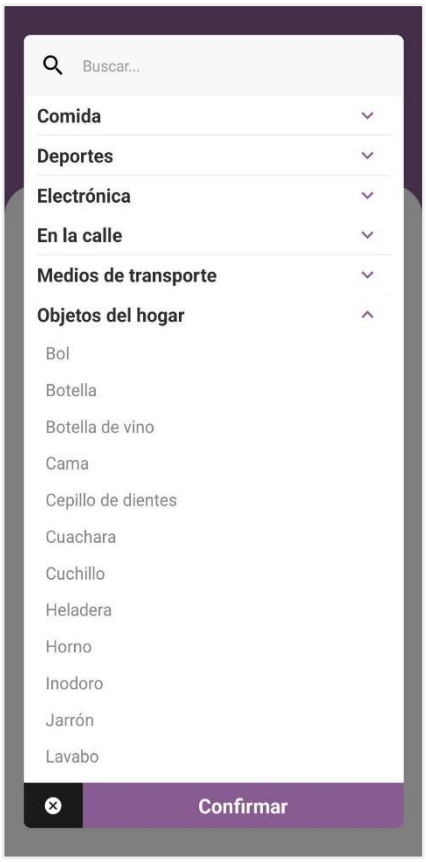


Figura 6.3: Pantalla de configuración del modelo de detección.

Es importante señalar que el modelo seleccionado, así como los objetos de interés, no pueden modificarse una vez que se crea *Información Posicionada* en el *Espacio de Trabajo*. Esta restricción se implementa para prevenir posibles inconsistencias.

- *Precisión mínima en inferencias*: este parámetro permite definir el nivel mínimo de precisión requerido para que los resultados del modelo sean considerados inferencias válidas. Este campo espera que se ingrese un valor porcentual entre 0 y 99, y cuenta con un valor por defecto de 30%.
- *Frecuencia de inferencias*: como se detalla en la Sección 5.2.1, los *modelos de reconocimiento y detección de objetos* están configurados para realizar inferencias en 1 de cada N frames obtenidos de la cámara del teléfono. Este parámetro permite configurar dicho valor. A un menor valor, mayor será la frecuencia de inferencias, aunque también será mayor el consumo de recursos. El valor por defecto para este campo es 100 frames.
- *Máxima distancia en metros del objeto para considerarlo*: cuando se *usa* el *Posicionamiento por objetos* en los *estados de prueba* y *final*, únicamente se tomarán en cuenta aquellos objetos que estén registrados en la base de datos contextual y que se encuentren a una distancia menor que la especificada. Esto permite desambiguar cuando un mismo objeto se encuentra registrado más de una vez. Para calcular esta distancia, se comparan las coordenadas de latitud y longitud registradas para los objetos en la base de datos contextual con las coordenadas actuales del usuario, obtenidas a través del GPS. El valor por defecto para este campo es 2 metros.
- *Apertura del ángulo de visión a considerar*: se evalúa la diferencia entre el ángulo de visión registrado para el objeto en la base de datos contextual y el ángulo de visión actual del teléfono (obtenido del giroscopio). Si esta diferencia es menor al ángulo de apertura establecido, se considera la inferencia como válida. El valor por defecto para este campo es 45 grados.

Una vez que el usuario ha configurado estos parámetros, tiene la capacidad (a través de una opción dentro de la herramienta) de activar la cámara del teléfono para iniciar el *Posicionamiento por objetos*. Dependiendo del estado actual del *Espacio de Trabajo* se pueden desencadenar dos flujos de acción distintos: registrar información posicionada (crear) o detectar información posicionada (usar). En las siguientes sub-secciones se detallarán cada una de estas.

6.3.2 Registrar información posicionada (crear)

En los estados de co-diseño (inicial, colaborativo y decisivo, descriptos en la Sección 6.1), los usuarios pueden utilizar el *Posicionamiento por objetos* para identificar objetos en posiciones del espacio y registrar allí *Información Posicionada*.

Bajo esta modalidad, cuando se prende la cámara del dispositivo, el modelo de reconocimiento o de detección seleccionado empieza a recibir imágenes. A partir de estas imágenes, el modelo realiza inferencias y proporciona los resultados correspondientes. La frecuencia de las inferencias, así como los objetos de interés dentro del *Espacio de trabajo*, se determinan en función de las configuraciones establecidas (las cuales se detallan en la Sección 6.3.1).

En la Figura 6.4 se presentan algunas capturas de pantallas donde se puede apreciar esta funcionalidad de la herramienta. En la primera captura (Figura 6.4.a), se muestra la cámara de la herramienta recién inicializada. También, se puede observar el mensaje que muestra la herramienta cuando todavía no se detectaron objetos. Las capturas presentadas en las

Figuras 6.4.b y 6.4.c muestran cómo se comporta la herramienta con modelos de detección y de reconocimiento respectivamente.

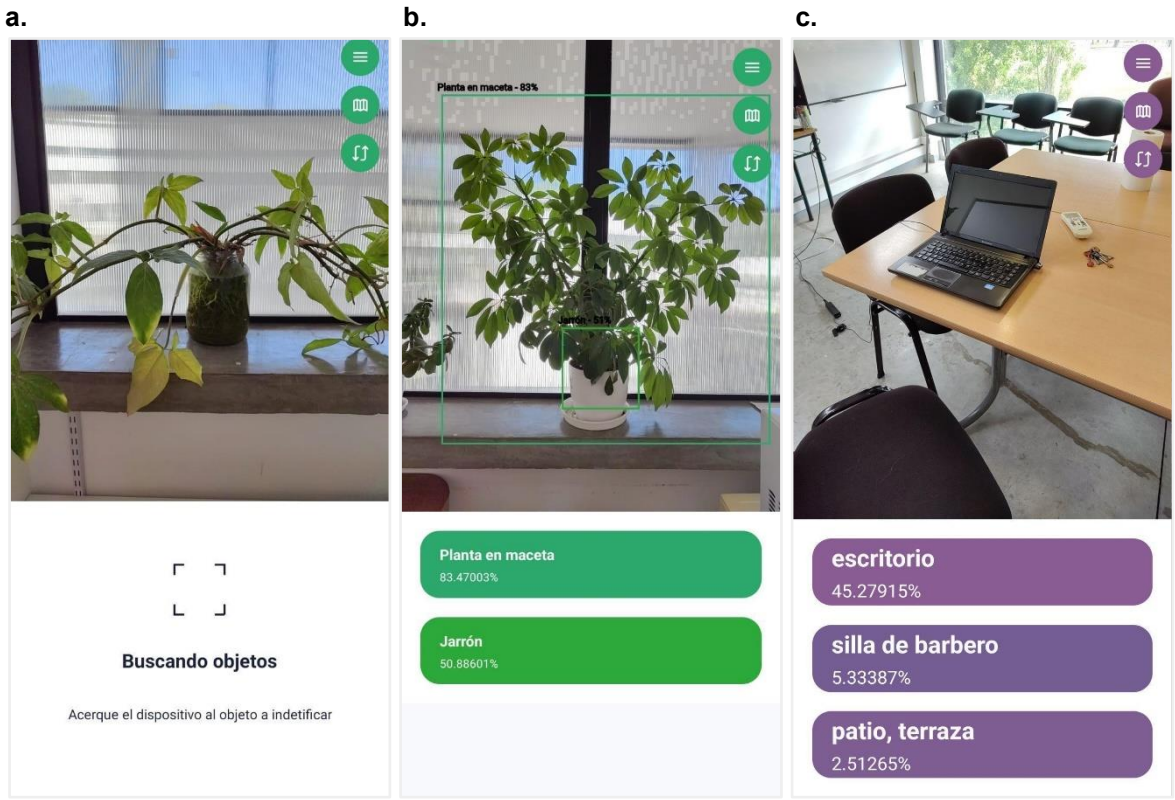


Figura 6.4: Inferencia de objetos desde la herramienta[44].

Al observar las Figuras 6.4.b y 6.4.c se puede apreciar que, una vez que el modelo logra inferir objetos, la herramienta proporciona un listado de objetos reconocidos junto con detalles sobre el nivel de certeza (o precisión) en la inferencia.

En la Figura 6.4.b se utiliza el *modelo de detección COCO-SSD*, el cual no solo identifica el objeto, sino que también ofrece su ubicación precisa en la pantalla mediante el uso de *bounding boxes*. Si bien el modelo devuelve la información para generar los *bounding boxes*, la implementación para visualizarlos desde la herramienta tuvo que ser desarrollada. En cambio, en la Figura 6.4.c, se presenta una inferencia realizada con el *modelo de reconocimiento MobileNet v2*.

Cuando un usuario selecciona un objeto de los listados (como los presentados en las Figuras 6.4.b o 6.4.c), es dirigido a otra pantalla para comenzar el proceso de registro de nueva *Información Posicionada* (esto es un tipo de *punto de interés* particular, acorde a la plantilla que usa el *Espacio de Trabajo*). Esta pantalla se presenta en la Figura 6.5, donde se muestran los campos que el usuario debe completar y, por otro lado, los campos con la información obtenida a través del *Posicionamiento por objetos,* al iniciar el proceso de registro. Esta información incluye los detalles del objeto identificado, así como los datos del GPS (latitud y

[44] La diferencia de proporciones y tamaño de las letras entre las capturas de pantalla se debe a las variaciones en la resolución de los dispositivos utilizados para capturarlas. Las capturas de las Figuras 6.4.a y 6.4.b fueron tomadas con un teléfono Samsung S22+, mientras que la Figuras 6.4.c fue capturada con un Xiaomi 12.

longitud) y del giroscopio (ángulo de visión). Asimismo, en este caso particular, tiene sentido registrar también el piso del edificio.

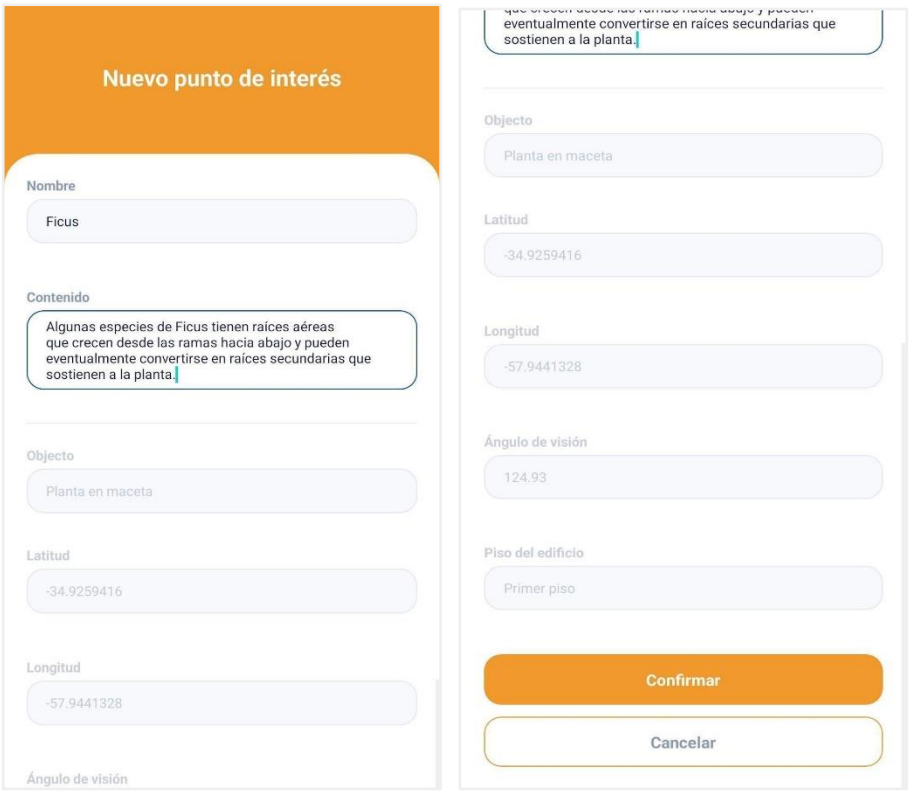


Figura 6.5: Formulario de registro de un nuevo punto de interés.

Es importante destacar que al confirmar el formulario presentado en la Figura 6.5 ocurren dos operaciones distintas pero interrelacionadas:

1. La primera operación implica registrar en la *base de datos contextual* del *Posicionamiento por objetos* la referencia al objeto con su posición exacta, lo que permite su posterior identificación y utilización. De acuerdo a lo mencionado en el Capítulo 5, en esta base de datos se guarda el objeto identificado, la posición del usuario al momento de registrarlo (GPS) y el ángulo de visión del dispositivo (giroscopio). A su vez, para cada registro se genera un ID o identificador único y se registra el edificio y piso asociado.
2. Al mismo tiempo, se registra un "*Punto de Interés*". Un tipo de dato vinculado al *Espacio de Trabajo* que se registra en la base de datos de la herramienta. En el caso de la plantilla de "*Información Posicionada*", el usuario puede cargar en este registro un título identificatorio y un texto informativo sobre el punto de interés. Además, en este registro se guarda la información de posicionamiento (que se registra de forma redundante para evitar consultas innecesarias) y el ID del objeto en la base de datos contextual, garantizando así la correspondencia entre ambos conjuntos de datos. Adicionalmente, se incluyen otros datos relevantes, como el identificador del *Espacio de Trabajo* actual, el usuario que lo creó y el piso del edificio al que está asociado.

   En un futuro, con nuevas plantillas, se espera poder crear *puntos de interés* más robustos con más valores contextuales.

Los *puntos de interés* creados (en este caso, *Información Posicionada*) se muestran en el mapa como marcadores, tal y como se muestra en la Figura 6.6. Cada marcador tiene un color que representa al usuario que lo registró. Esto facilita la identificación visual de las contribuciones individuales. Asimismo, se puede observar que cada marcador tiene en su interior una flecha que indica el ángulo de visión del teléfono al momento del registro, ofreciendo así una representación visual adicional de la orientación del dispositivo.

Los usuarios con perfil de *Creador* tienen la capacidad de visualizar en el mapa, en todo momento, todos los objetos registrados en el *Espacio de Trabajo*, como se aprecia en la Figura 6.6.a. En cambio, los usuarios con perfil de *Colaborador* solo pueden ver los objetos que ellos mismos registraron, como se ve en la Figura 6.6.b.

a. **b.**

Figura 6.6: Visualización de los puntos de interés en la herramienta.

Al hacer clic sobre cualquiera de estos marcadores, se puede acceder rápidamente a la información asociada al mismo, tal como se ejemplifica en la Figura 6.7. Esta función permite a los usuarios obtener detalles específicos sobre cada *punto de interés* en el mapa con facilidad.

En la Figura 6.7, además, se presentan dos botones en la parte inferior de la pantalla. El primero de ellos facilita el acceso a un formulario de edición, donde tanto el usuario con perfil de *Creador* como el usuario que registró el marcador podrán editar el título, la descripción cargada y marcarlo como visible u oculto en la versión final de la aplicación. Esto es especialmente útil en el *estado decisivo*, donde se acuerdan los puntos que son realmente relevantes para la aplicación.

El creador del marcador tiene la posibilidad de eliminarlo en caso de considerarlo conveniente, usando el botón que se aprecia en la Figura 6.7. Al eliminar un *punto de interés*

se borra también la referencia en la base de datos contextual del *Posicionamiento por objetos*.

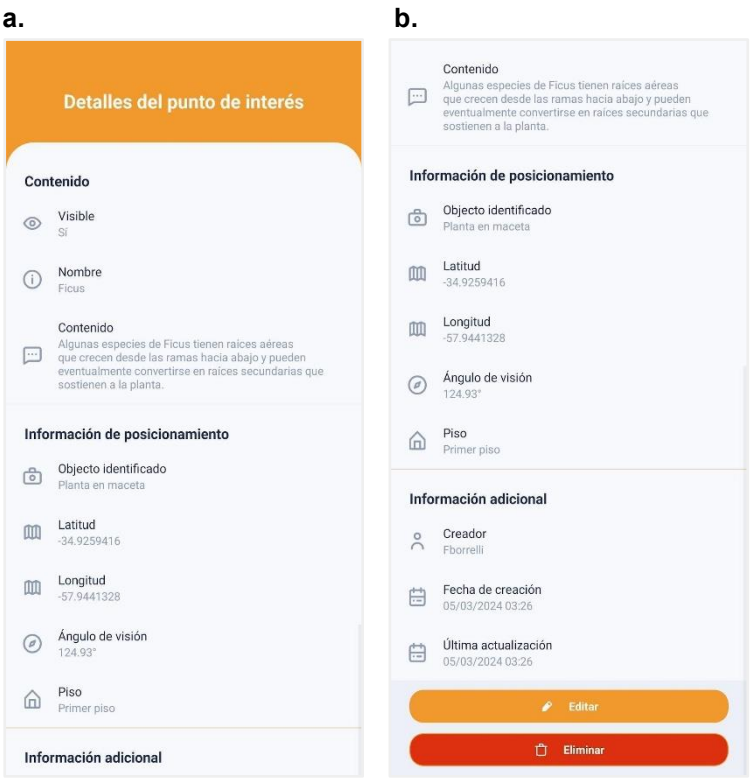


Figura 6.7: Vista detalle de un punto de interés creado.

6.3.3 Detectar información posicionada (usar)

Durante los *estados de prueba* y *final*, los usuarios pueden utilizar la solución de *Posicionamiento por objetos* para identificar objetos posicionados que hayan sido previamente registrados en la base de datos contextual.

Este proceso dentro del *Posicionamiento por objetos* se resuelve de la siguiente manera:

1. *Inferencia de objetos*: en este primer paso se busca identificar objetos en el espacio de forma similar a lo explicado en la Sección 6.3.2. Esto implica activar la cámara del dispositivo y enviar imágenes al modelo de reconocimiento o detección seleccionado. Este modelo devuelve las inferencias obtenidas, las cuales pueden ser filtradas según la configuración establecida (como se detalla en la Sección 6.3.1), como el nivel mínimo de precisión o los objetos disponibles.
2. *Validación en la base de datos contextual*: utilizando los objetos identificados en el paso anterior junto con la información de la posición actual del dispositivo, se realiza una consulta en la base de datos contextual de *Firestore* para obtener los registros relevantes. Durante esta consulta, se buscan registros que contengan los objetos identificados, posean un ángulo de visión similar al utilizado en el momento de la creación del registro y se encuentren en una posición cercana. Es importante destacar que *Firestore* facilita este proceso gracias a su capacidad para realizar consultas geoespaciales[45]. Adicionalmente, el uso de la caché de *Firestore* en el dispositivo

[45] Consultas geoespaciales en *Firestore*: https://firebase.google.com/docs/firestore/solutions/geoqueries (último acceso 27/11/2023).

mejora significativamente la eficiencia de este proceso.

3. *Envío de resultados*: la solución de *Posicionamiento por objetos* proporciona dos listados. Uno que incluye los objetos inferidos durante el *paso 1*, y otro que contiene los registros válidos obtenidos en la consulta realizada en el *paso 2*.

A partir de los resultados obtenidos de este proceso, la herramienta de autor es capaz de establecer una correspondencia con los *puntos de interés* asociados al *Espacio de Trabajo* actual. Esto se logra utilizando los ID de los registros, lo que permite identificar con exactitud los *puntos de interés* (*Información Posicionada)* cercanos.

Aunque la herramienta muestra todos los objetos identificados, solo se permite acceder a aquellos que están asociados con un *punto de interés* cercano. En la Figura 6.8.a se ilustra un ejemplo de esto. En esta figura se observa que el objeto "*Bol*" está bloqueado debido a la falta de *Información Posicionada* cercana asociada al mismo. En contraposición, sí se puede hacer clic en la opción "*Planta en maceta*", ya que esta sí cuenta con información registrada.

Al hacer clic en un objeto identificado en el listado de la Figura 6.8.a, el usuario es redirigido a otra pantalla desde la cual puede ver la información contextual asociada, como se observa en la Figura 6.8.b. Esta figura presenta la información relacionada con el *punto de interés* creado en la Figura 6.5 (de la Sección 6.3.2).

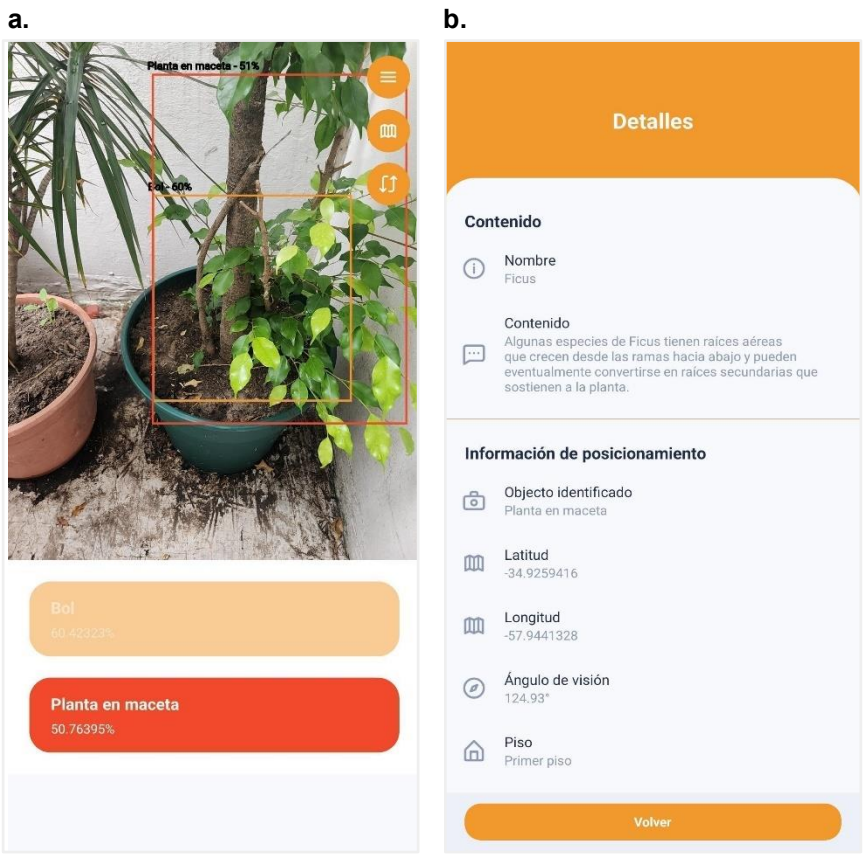


Figura 6.8: Uso del Posicionamiento por objetos en la herramienta.

6.3.4 Monitoreo de Eventos

Debido a las restricciones impuestas por la capa de uso gratuita de *Firebase*, se tomó la decisión de restringir el funcionamiento del *Monitoreo de Eventos* exclusivamente a los

Espacios de Trabajo que se encuentran en *estado de prueba (*co-testeo*)*. Al registrar eventos únicamente en esta fase, se consigue reducir significativamente el número de operaciones de escritura que se llevan a cabo sobre la base de datos de métricas.

Como se detalla en la Sección 5.4.2, la implementación del *Monitoreo de Eventos* ofrece la posibilidad de configurar los eventos ("*Eventos de detección*" y "*Eventos de actualización*") para registrar información adicional según sea necesario. Siguiendo esta línea, se optó por incluir en estos eventos el ID del *Espacio de Trabajo* actual y el ID del usuario que está usando la herramienta.

Además, considerando que el co-testeo podría contar con múltiples iteraciones, donde la *Información Posicionada* o las configuraciones del Espacio de trabajo podrían modificarse, se decidió también almacenar un número de versión, el cual se actualiza cada vez que se inicia un nuevo co-testeo. Esto es importante, dado que los datos recolectados en una iteración podrían ya no ser de interés en la siguiente. Al registrar versiones se facilita el filtrado de resultados y se asegura que la herramienta presente únicamente la información correspondiente a la última iteración.

6.3.5 Métricas y estadísticas

Una vez co-diseñada una aplicación, es necesario evaluar su funcionamiento y (posiblemente) realizar ajustes para lograr que el usuario tenga una buena experiencia de uso. Estos ajustes podrían estar vinculados a cuestiones asociadas a la *Información Posicionada* creada (como por ejemplo modificar el ángulo de visión o el objeto a inferir), o podrían implicar cambios más generales como modificaciones en las configuraciones del *Posicionamiento por objetos* (explicadas en la Sección 6.3.1).

En este sentido, la utilización de métricas y gráficos (que permitan ver datos cuantitativos de una forma amena) podrían resultar de gran ayuda para identificar estos ajustes de una manera más precisa, facilitando así la toma de decisiones, como se menciona en [Sarker, 2021], donde a estos "ajustes" se lo cataloga como co-testeo.

Con la información disponible en la herramienta y los datos recopilados con el *Monitoreo de Eventos,* podría surgir información de utilidad. Por ejemplo, podría ser de interés:

- Realizar un análisis de la *Información Posicionada* cargada en el *Espacio de Trabajo* que podría servir para brindar una visión general del nivel de actividad y la densidad de información dentro del entorno de trabajo, permitiendo así tomar decisiones sobre qué información incluir para los usuarios finales.
- Del mismo modo, la distribución de esta información cargada en función de sus ángulos de visión registrados podría facilitar información valiosa para ajustar la navegación del usuario.
- Además, la identificación de los objetos más recurrentes captados por la cámara del teléfono podría permitir detectar nuevos objetos de interés y ajustar los existentes en base a la cantidad de inferencias recibidas.
- Por otro lado, analizar el tiempo que tarda el usuario en detectar la *Información Posicionada* podría ayudar a identificar posibles problemas en la configuración del *Posicionamiento por objetos* y a realizar ajustes para mejorar la experiencia del usuario.

Estos puntos, junto a otros, permiten comprender mejor cómo se realizó el proceso de co-diseño, y permiten ayudar al equipo a evaluar si se requiere o no un rediseño. Es precisamente por esta razón que se tomó la decisión de incorporar una sección dentro de la herramienta dedicada a visualizar y analizar este tipo de información. Al proporcionar una manera fácil y accesible de ver estos datos directamente desde la herramienta, se busca agilizar el proceso de co-testeo y facilitar la toma de decisiones.

En la herramienta, estas cuestiones se analizarán durante el *estado de prueba* de los *Espacios de Trabajo*, donde se espera que se realice el co-testeo de la aplicación. Cuando se alcanza este estado *(de prueba)*, la herramienta habilita una nueva sección en el menú del *Espacio de Trabajo* llamada "*Métricas y Estadísticas*", la cual está disponible exclusivamente para los usuarios con perfil *Creador*. Esto se puede observar en la Figura 6.9.a. Al hacer clic en esta opción del menú, el usuario es dirigido a una pantalla similar a la mostrada en la Figura 6.9.b.

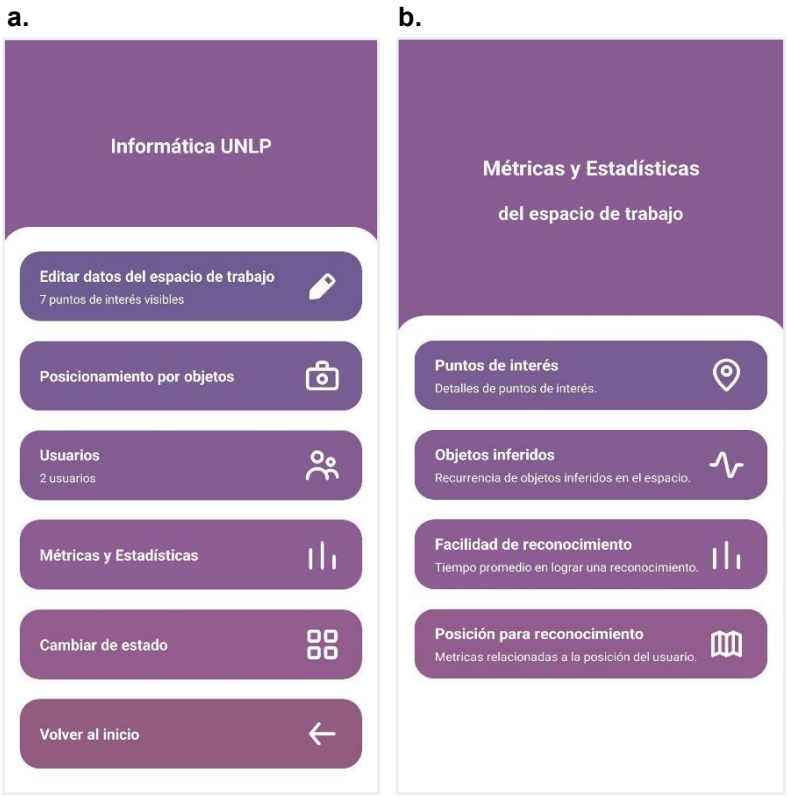


Figura 6.9: Visualización de métricas desde la herramienta.

Cada una de las secciones mostradas en la Figura 6.9.b proporciona al usuario acceso a una pantalla donde se presentan diversas métricas y gráficos relacionados con varios aspectos del co-diseño y/o co-testeo. Si bien esta herramienta cuenta con una única plantilla (la cual permite crear puntos de interés del tipo *Información Posicionada*), se espera que la información presentada pueda ser utilizada por cualquier plantilla que utilice el *Posicionamiento por objetos*.

A continuación se brinda una descripción general de cada una de las secciones de la Figura 6.9.b:

- *Puntos de interés*: ofrece una visión general de la actividad de co-diseño y la densidad de información en el *Espacio de Trabajo*. Ayuda a conocer la cantidad y visibilidad de

los *puntos de interés* (en este caso particular, *Información Posicionada*), así como a identificar posibles conflictos espaciales que podrían afectar la usabilidad. La información presentada en esta sección se obtiene exclusivamente de los datos almacenados en la base de datos de la herramienta.

- *Objetos inferidos*: permite identificar los objetos más recurrentes detectados por el modelo de reconocimiento. Dependiendo del dominio de la aplicación a co-diseñar, la información brindada por esta sección podría ayudar a comprender mejor el entorno del usuario y a ajustar el diseño en consecuencia, modificando los objetos relevantes y lograr así una experiencia más apropiada para el usuario. Cabe mencionar que la información presentada en esta sección se obtiene de los registros de "*Eventos de actualización*".
- *Facilidad de reconocimiento*: evalúa el tiempo que los usuarios tardan en detectar *puntos de interés* específicos. Con esta información se puede comprender mejor la efectividad del diseño planteado, permitiendo detectar si hay que realizar ajustes en las configuraciones del *Posicionamiento por objetos*.
- *Posición para reconocimiento*: realiza una comparación por distancia (en metros) a la que se encontraban los usuarios de la *información posicionada* al momento de la detección. Esto ayuda a comprender mejor la interacción entre el usuario y el entorno.

En el Anexo C, se presenta una explicación extendida de estas secciones, donde se presentan cada una de las métricas y gráficos que comprenden las mismas.

6.4 Conclusiones del capítulo

En este capítulo se presentó la herramienta de autor que brinda soporte al co-diseño y co-testeo in-situ de aplicaciones sensibles al contexto. Si bien la plantilla que se utiliza en los *Espacios de Trabajo* permite crear *Información Posicionada*, la herramienta está diseñada para ser extensible en un futuro con otro tipo de contenido más robusto.

Con la herramienta se pudo apreciar un uso concreto de las implementaciones de la Sección 5.4 tanto del *Posicionamiento por objetos* como del *Monitoreo de eventos*. Por un lado, se concretó la creación de *Información Posicionada* mediante el *Posicionamiento por objetos* y luego su posterior uso. Por otro lado, mediante el *Monitoreo de eventos* se registra información que permite obtener métricas y gráficos estadísticos que facilitan la toma de decisiones en la etapa de co-testeo.

En este capítulo se presentó una versión inicial de la herramienta, la cual posteriormente con distintas pruebas sistemáticas se ha ido mejorando, como se mencionará en el próximo capítulo.

7. Experiencias de co-diseño y co-testeo usando la herramienta desarrollada

En este capítulo se presentan varias experiencias de co-diseño y co-testeo utilizando la herramienta de autor presentada en el Capítulo 6. Estas experiencias tuvieron como objetivo analizar y evaluar la viabilidad del *Posicionamiento por objetos* en casos de uso realistas.

En primer lugar, se presenta una experiencia que hace uso de la versión inicial de la herramienta presentada en el Capítulo 6. A partir del feedback obtenido se identificaron varios problemas de funcionamiento y mejoras en términos de usabilidad, dando lugar a una segunda versión de la herramienta, la cual aborda y corrige estos aspectos. Usando esta segunda versión se realizaron dos experiencias, las cuales también se presentan en este capítulo.

7.1 Pruebas con la primera versión de la herramienta

En esta sección, se presenta una primera experiencia de co-diseño y co-testeo utilizando la herramienta de autor tal como se la presentó en el Capítulo 6. Esta experiencia representó la primera prueba real de la herramienta siendo utilizada de forma colaborativa.

Es importante destacar que en esta primera experiencia se decidió que participaran únicamente dos personas, ambas con experticia en diseño y codificación de aplicaciones móviles sensibles al contexto. Además, ambas personas contaban con conocimiento en co-diseño, habiendo participado previamente en otras experiencias. Esto permitió agilizar las pruebas realizadas.

En esta experiencia los participantes realizaron el proceso completo de co-diseño y co-testeo con la herramienta, lo que proporcionó información valiosa sobre su funcionalidad. El feedback recolectado durante esta experiencia, sirvió luego para generar una nueva versión donde se mejoran problemas de funcionamiento y algunos aspectos de usabilidad.

7.1.1 Aplicación para un congreso

Para esta primera experiencia, se planteó el co-diseño de una aplicación prototípica destinada a ser utilizada durante un congreso en la Facultad de Informática de la Universidad Nacional de La Plata (UNLP). Específicamente, un evento que tendría lugar entre las aulas 1-2 y 1-4 del primer piso, asignando un track (tema) específico a cada aula. La aplicación proporcionaría *Información Posicionada* de cada uno de estos tracks.

Para la experiencia se establecieron objetivos generales y específicos. En términos generales, la experiencia se centró en analizar el funcionamiento del *Posicionamiento por Objetos* al registrar *Información Posicionada* utilizando múltiples objetos del mismo tipo. Particularmente, se tomó la decisión de registrar las puertas de las distintas aulas.

En cuanto a los objetivos específicos, se esperaba que esta experiencia sirva para poner a prueba la solución del *Posicionamiento por Objetos*, analizando varios aspectos claves como:

- Observar cómo se comporta el *Posicionamiento por Objetos* cuando se cuenta con objetos del mismo tipo que se encuentran muy cerca uno del otro, como ocurre con las puertas de las aulas 1-2 y 1-3 de la Facultad de Informática (UNLP).

- Analizar cómo el *Posicionamiento por Objetos* maneja los cambios en la orientación del usuario (*ángulo de visión*) con respecto al objeto detectado:
    - Comprobando si la detección sigue siendo precisa cuando el usuario se acerca a la puerta desde diferentes ángulos.
    - Viendo qué ocurre cuando la *Información Posicionada* es registrada considerando distintos ángulos. Para esto se plantea la creación de dos *Espacios de Trabajo* distintos, donde en uno la información se registra utilizando siempre el mismo ángulo de visión y otro donde el ángulo varía.

7.1.1.1 Armado de la información para el co-diseño

Dado que el objetivo principal de la experiencia no era la creación de contenido, sino la evaluación del funcionamiento del *Posicionamiento por Objetos*, se optó por generar, previamente, la información de los *tracks* que debían registrarse en cada aula. Esta información fue provista a los participantes durante la experiencia, lo que permitió agilizar los tiempos y evitar la necesidad de que los participantes tuvieran que pensar qué contenido definir.

En la Tabla D.1 del Anexo D se detalla la información proporcionada a los participantes, incluyendo el nombre y el contenido a mostrar de cada track. Por otro lado, en la Tabla D.2 se especifica la puerta del espacio físico que se debía utilizar para registrar cada *Información Posicionada*, indicando también desde dónde debían posicionarse para llevar a cabo el registro. En la Figura 7.1 se visualizan remarcadas con círculos las tres puertas elegidas. El color asociado a cada círculo representa al *Colaborador* encargado de registrar la información dentro de la herramienta de autor. Se puede observar, además, que los círculos correspondientes a las puertas de las aulas 1-2 y 1-3 se encuentran muy cerca entre sí.

Figura 7.1: Plano del primer piso de la Facultad de Informática de la UNLP donde se ven las puertas que van a tener *Información Posicionada*.

7.1.1.2 Decisiones al crear el *Espacio de Trabajo*

Como se mencionó anteriormente, con el objetivo de poder probar el *Posicionamiento por Objetos* considerando distintos ángulos cuando se registra la *Información Posicionada,* se definieron dos *Espacios de Trabajo*. En ambos espacios, la información del congreso es la

misma (ver Tabla D.1 del Anexo D) y lo que cambia son las condiciones al momento de crear la *Información Posicionada*:

- En el primer *Espacio de Trabajo* (Congreso - Versión 1), los participantes debían registrar la información utilizando siempre el mismo ángulo de visión. Es decir, apuntando a las puertas siempre desde la misma dirección. En particular, se decidió que las puertas debían registrarse de frente. A este *Espacio de Trabajo* se le asignó el color "verde" en la herramienta.
- En el segundo *Espacio de Trabajo* (Congreso - Versión 2), se buscaba que la *Información Posicionada* fuera registrada utilizando diferentes ángulos de visión. Específicamente se decidió:
    - Para las aulas 1-2 y 1-3: pararse en el medio de ambas aulas y registrar el contenido desde allí, apuntando el celular hacia la derecha y hacia la izquierda, respectivamente.
    - Para el aula 1-4: registrar el contenido desde el lado izquierdo.

  A este *Espacio de Trabajo* se le asignó el color "naranja" en la herramienta.

Además, dentro de estos *Espacios de Trabajo* se decidió configurar el *Posicionamiento por objetos*, con las siguientes configuraciones (las cuales son explicadas en detalle en la Sección 6.3.1):

- *Modelo:* se utilizó el modelo de "*Puertas*" presentado en la Sección 4.2.2. De esta forma, cuando un usuario se acerca a la puerta de un aula, puede recibir la información correspondiente. Cabe recordar que este modelo había sido entrenado utilizando (entre otras) imágenes de puertas del edificio de la Facultad de Informática de la UNLP.
- *Clases*: el modelo es capaz de reconocer únicamente puertas, por lo que no es necesario restringirlo para que solo reconozca ciertos objetos.
- *Precisión mínima en inferencias:* se definió una precisión mínima de 15%. Es decir, que el modelo deberá ser capaz de reconocer las puertas con un nivel de certeza igual o superior a este porcentaje.
- *Frecuencia de inferencias:* al modelo se envían, para las inferencias, 1 de cada 100 imágenes (*frames*) obtenidas desde la cámara del dispositivo.
- *Máxima distancia en metros del objeto para considerarlo:* se decidió utilizar una distancia de 1 metro.
- *Apertura del ángulo de visión a considerar:* se decidió utilizar un ángulo de apertura de 70 grados.

Finalmente, se decidió registrar en la herramienta dos usuarios, los cuales fueron agregados como *Colaboradores* a los dos *Espacios de Trabajo* creados ("Congreso - Versión 1" y "Congreso - Versión 2"). A cada Colaborador se le asignó un color específico dentro del *Espacio de Trabajo*. Mientras que al *Colaborador 1* se le asoció el color azul, al *Colaborador 2* se le asignó el color rosa. De esta forma, sería más sencillo identificar qué cargó cada usuario dentro de los *Espacios de Trabajo*. Estos colores se corresponden con los presentados en la Figura 7.1 (de la Sección 7.1.1).

7.1.1.3 Co-diseñar la aplicación para un congreso (crear *Información Posicionada*)

Se decidió realizar la experiencia en una única jornada de 2 horas, de forma in-situ, en la Facultad de Informática de la UNLP. Al comenzar la experiencia se le brindó a cada uno de los participantes los datos necesarios para iniciar sesión junto con un link para descargar la herramienta de autor y poder así crear la *Información Posicionada*. En particular, a uno de los participantes se le dio acceso como "*Colaborador 1*", mientras que al otro se le dio acceso como "*Colaborador 2*". Las características de los teléfonos utilizados se encuentran detalladas en la Tabla D.3 del Anexo D.

Como los participantes tenían experiencia en este tipo de co-diseño, solo fue necesario explicar algunos aspectos técnicos relacionados al uso de la herramienta.

Para iniciar el co-diseño, se les pidió a los participantes que accedan primero al *Espacio de Trabajo* "Congreso - Versión 1", para empezar a registrar la *Información Posicionada* estando de frente a las puertas de las aulas. En la Figura 7.2 se muestra cómo quedó registrada la *Información Posicionada* en la herramienta.

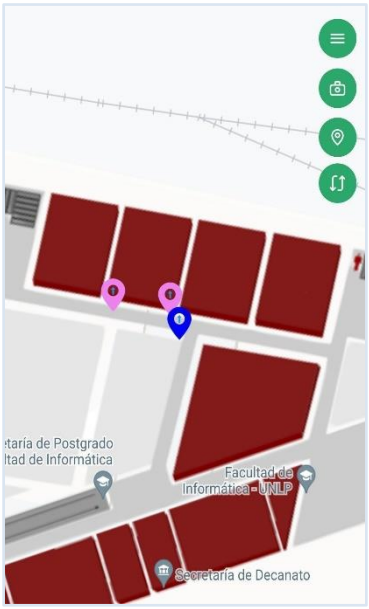


Figura 7.2: *Información Posicionada* del *Espacio de Trabajo* "Congreso - Versión 1" (vista desde el perfil *Creador*).

Como se puede observar en la Figura 7.2, todos los marcadores del mapa tienen un ángulo de visión similar, indicando en este caso que el registro se hizo de frente. Se debe mencionar que, durante el proceso de registro, la información proporcionada por el *Giroscopio* siempre se mantuvo constante y no presentó diferencias significativas entre los distintos teléfonos.

En cuanto al funcionamiento del GPS, se puede observar en la Figura 7.2, que no fue muy preciso. Para entender mejor qué aconteció, se presenta en la Figura 7.3 una comparación entre la ubicación de los participantes al momento del registro de la *Información Posicionada* (Figura 7.3.a) y el valor registrado por el GPS al momento de almacenar la información en la base de datos (Figura 7.3.b). Se puede ver que en todos los casos hubo un desfase en el posicionamiento. Cabe resaltar que la *Información Posicionada* cargada por el "*Colaborador 2"* (correspondiente a las aulas 1-2 y 1-4) mostró un desfase significativamente menor en comparación con el observado en la carga realizada por el "*Colaborador 1"* (correspondiente al aula 1-3).

Figura 7.3: Comparación entre la ubicación real de los participantes al momento del registro (a) y la posición obtenida con el GPS al momento de registrar la *Información Posicionada* (b).

Si bien se anticipaba un cierto margen de error que podría tener el GPS (dado que está documentado que su funcionamiento no es óptimo en entornos *indoor*), la magnitud de la diferencia observada entre dos dispositivos resultó ser muy significativa.

A partir de esto, se decidió investigar qué estaba ocurriendo con el GPS de cada dispositivo. Para determinar si esta disparidad era atribuible a un problema en la aplicación o si realmente era una limitación del GPS, se llevó a cabo una prueba adicional. Para esto se utilizó otra aplicación[46] que muestra la latitud y longitud actuales de un dispositivo. Colocando los dos teléfonos juntos sobre una mesa, se buscó comparar las lecturas proporcionadas por esta aplicación, y corroborar si reflejaban una discrepancia similar a la observada al crear la *Información Posicionada*. En la Figura 7.4 se puede apreciar una diferencia notable en la información de latitud de ambos dispositivos, aunque la longitud mantuvo valores similares.

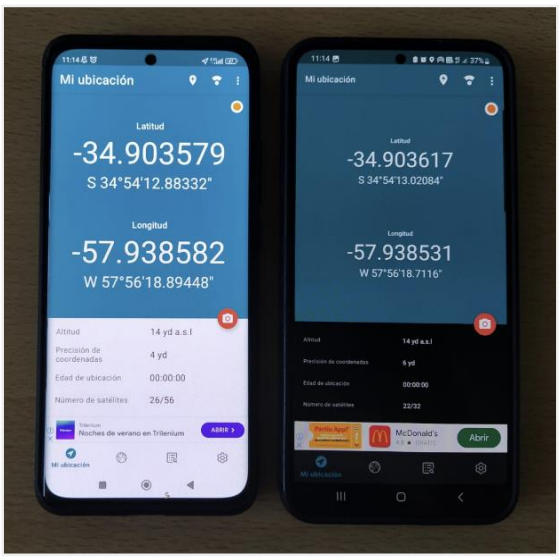


Figura 7.4: Comparación del funcionamiento del GPS entre dispositivos Xiaomi 12 (izquierda) y Samsung S22 (derecha).

[46] Aplicación *Coordenadas de mi GPS*: https://play.google.com/store/apps/details?id=com.freemium.android.apps.gps.coordinates (último acceso: 15/12/2023).

Con solo mirar los valores de la Figura 7.4 no se puede determinar la diferencia en metros entre ambos dispositivos. Por lo tanto, se decidió realizar un cálculo[47] de distancia, obteniendo como resultado una diferencia de 6 metros entre uno y otro.

Después de esta primera experiencia, se llevó a cabo un análisis que reveló que las discrepancias observadas en la Figura 7.4 podrían atribuirse a varias cuestiones[48], entre las que se incluyen: la posición de la antena en el dispositivo y el tipo de tecnología empleada en la misma (SiRF3 o SiRF4), la necesidad de calibración del GPS, la activación del modo de ahorro de energía en el dispositivo, así como el uso de fundas que puedan interferir con la recepción de señales GPS, entre otros. Estos factores, individualmente o en combinación, podrían haber contribuido a las diferencias significativas detectadas durante la carga de *Información Posicionada* en el *Espacio de Trabajo:* “Congreso - Versión 1”.

Considerando estas disparidades en el uso del GPS, se decidió que un único *Colaborador* sea responsable de cargar toda la información en un nuevo *Espacio de Trabajo* llamado "Congreso - Versión 3". En este nuevo espacio se siguieron las mismas condiciones de creación utilizadas para el "Congreso - Versión 1" (descriptas en la Sección 7.1.1.2), donde la información se registró estando de frente a las puertas. Específicamente, se designó al *Colaborador 2*, cuyo desfasaje fue el más reducido, para llevar a cabo esta tarea, como se puede apreciar en la Figura 7.5.

Se puede observar que al comparar los marcadores mostrados en la Figura 7.5 con los de la Figura 7.2 (donde el *Colaborador* 2 cargó la información de las Aulas 1-2 y 1-4), aún utilizando el mismo teléfono y estando físicamente parado en el mismo lugar, las posiciones cargadas presentaron diferencias.

Figura 7.5: *Información Posicionada* registrada por el *Colaborador 2* en el *Espacio de Trabajo* “Congreso - Versión 3” (con mismo ángulo de visión).

[47] Cálculo de distancia realizado: https://codesandbox.io/p/sandbox/geolib-tests-forked-rw4jrx (último acceso: 15/12/2023).
[48] How to troubleshoot GPS problems on android devices: https://help.aware360.com/en/knowledge-base/how-to-troubleshoot-gps-problems-android (último acceso: 15/12/2023).

Después de completar el registro de datos en el *Espacio de Trabajo* "Congreso - Versión 3", se le indicó al *Colaborador 2* que acceda al *Espacio de Trabajo* "Congreso - Versión 2" para registrar la misma información, pero desde diferentes ángulos de visión, como se detalló en la Sección 7.1.1.2. Los resultados de este proceso se pueden apreciar en la Figura 7.6.

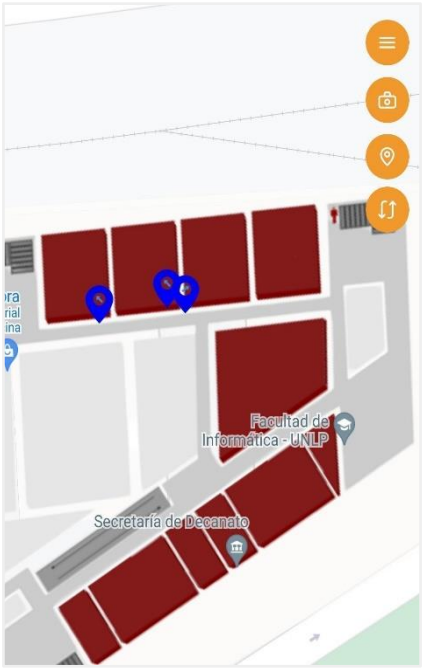


Figura 7.6: *Información Posicionada* registrada por el *Colaborador 2* en el *Espacio de Trabajo* “Congreso - Versión 2” (con distinto ángulo de visión).

7.1.1.4 Co-testear la aplicación co-diseñada (usar la *Información Posicionada*)

Después de registrar la *Información Posicionada* en los *Espacios de Trabajo*, se avanzó a la siguiente fase, que consistió en poner a prueba el funcionamiento del *Posicionamiento por Objetos* para ver si era posible acceder correctamente al contenido registrado.

Se decidió evaluar específicamente qué ocurría cuando los participantes accedían al contenido de los *Espacios de Trabajo* "Congreso - Versión 2" y "Congreso - Versión 3", donde el *Colaborador 2* había registrado toda la *Información Posicionada*. Además, para evaluar el funcionamiento de la forma más realista posible, se establecieron varios escenarios para que cada *Colaborador* pruebe en cada *Espacio de Trabajo*:

- Observar qué acontecía al caminar desde el Aula 1-2 hasta el Aula 1-4, así como en sentido inverso.
- Analizar cómo reaccionaba la aplicación al caminar a diferentes velocidades, tanto lento como rápido.

A partir de estos escenarios, se definieron diferentes pruebas. En cada una de estas, los participantes recorrieron el pasillo del primer piso de la facultad y probaron con la primera versión de la herramienta (presentada en el Capítulo 6). Al finalizar cada prueba y antes de iniciar la siguiente, se les pidió a los participantes que relataran su experiencia para poder registrar lo acontecido. Estas pruebas se llevaron a cabo en los *Espacios de Trabajo* "Congreso - Versión 2" y "Congreso - Versión 3", aplicando las configuraciones previamente establecidas (de acuerdo con lo indicado en la Sección 7.1.1.2).

Debido a las diferencias identificadas en el funcionamiento del GPS entre los dispositivos de los participantes, era de esperar que el *Colaborador 1* no pueda acceder al contenido registrado en ninguna de estas pruebas (ya que tenía un desfasaje estimado de 6 metros con el otro dispositivo móvil). No obstante, algo que llamó la atención es que el *Colaborador 2* tampoco pudo hacerlo.

Por ese motivo se decidió investigar qué ocurría al cambiar la configuración del *Posicionamiento por Objetos* en los dos *Espacios de Trabajo*. Específicamente, se decidió ir actualizando el valor de la configuración de "*Máxima distancia en metros del objeto para considerarlo"* de 1 metro a un valor mayor. Luego de hacer esto, se repitieron los escenarios de pruebas.

Solamente al aumentar el valor de la "*Máxima distancia en metros del objeto para considerarlo"* a 4 metros, el *Colaborador 2* logró acceder a toda la *Información Posicionada* de ambos *Espacios de Trabajo*. Sin embargo, con esta configuración el *Colaborador 1* no pudo acceder al contenido de ninguno de los espacios.

En la Figura 7.7 se puede observar un ejemplo donde se ve al *Colaborador 2* accediendo al contenido del Aula 1-2 en el *Espacio de Trabajo* "Congreso - Versión 2".

Algo todavía más llamativo fue que al aumentar la "*Máxima distancia en metros del objeto para considerarlo"* configurándola con 10 metros, *el Colaborador 2* siguió sin poder acceder a la *Información Posicionada* en ambos *Espacios de Trabajo.*

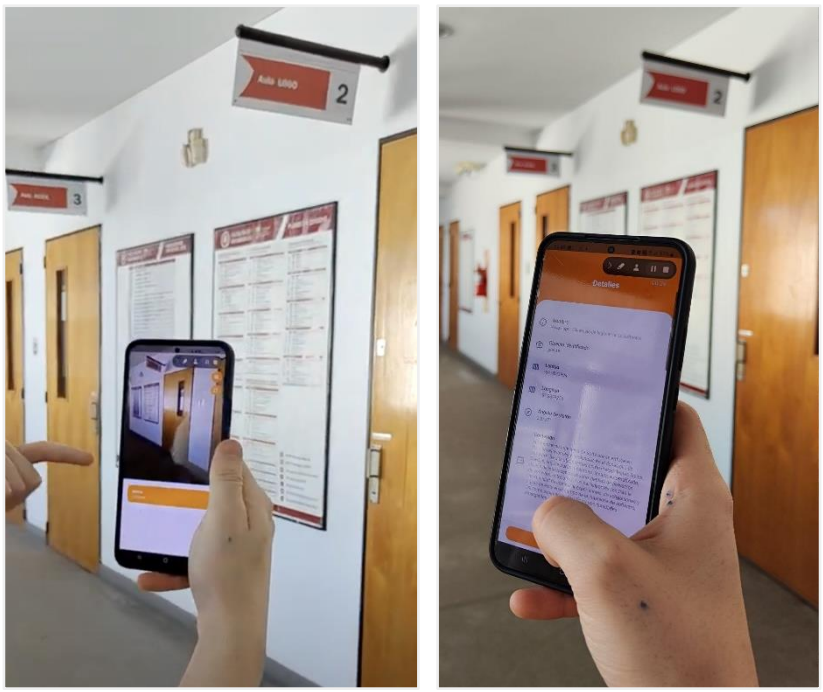


Figura 7.7: *Colaborador 2* accediendo a la *Información Posicionada* asociada al Aula 1-2 (en el *Espacio de Trabajo* "Congreso - Versión 2").

Para obtener una comprensión más clara de lo que estaba pasando, se tomó la decisión de acceder a uno de los *Espacios de Trabajo* con el perfil de *Creador* y examinar las métricas registradas. Se revisaron especialmente las métricas asociadas a la pantalla *"Posición para reconocimiento*". Desde esta pantalla se puede analizar qué tan cerca consideraba la herramienta de autor que el usuario logró estar de la *Información Posicionada*. En la Sección

6.3.5 y en el Anexo C se pueden encontrar más detalles de esta métrica.

En la Figura 7.8 se presenta una captura de la métrica "*Posición para reconocimiento*" dentro del *Espacio de Trabajo* “Congreso - Versión 2”. Observando los filtros de la figura, se puede notar que se están mostrando los resultados asociados al *Colaborador 2* y para el Aula 1-2 (Aula Logo). Con el gráfico de la figura se puede apreciar que para la herramienta lo más cerca que el *Colaborador 2* estuvo del Aula 1-2 fueron 11 metros, cuando en la realidad estuvo mucho más cerca observando su posición en el espacio físico respecto de la puerta de esa aula.

Para investigar las posibles causas de este problema, se llevaron a cabo varias pruebas. Una de las hipótesis sugería que podría existir un error en la forma en que la herramienta calculaba la distancia entre el usuario y la *Información Posicionada*. Para realizar este cálculo, la herramienta empleaba una librería de *JavaScript* llamada geolib[49], que contiene una función diseñada para calcular la distancia en metros entre dos pares de coordenadas geográficas. Esta hipótesis ganó más credibilidad cuando se observó que, al visualizar la posición en el mapa de la herramienta, los usuarios parecían estar relativamente cerca de los marcadores.

Figura 7.8: Registros de distancia del *Colaborador 2* y del Aula 1-2 dentro de la herramienta.

Para analizar lo antes mencionado con más detalle, se le pidió al *Colaborador 1* que instale en su teléfono una herramienta llamada Fake GPS[50], que permite simular la ubicación del usuario, aunque no esté físicamente presente en ese lugar. Esta herramienta se configuró proporcionando la latitud y la longitud asociada a una *Información Posicionada*, lo que permitió ubicarlo muy cerca de una de las puertas, como se puede observar en la Figura 7.9.

[49] Documentación oficial de la librería *Geolib*: https://github.com/manuelbieh/geolib#readme (último acceso: 15/12/2023).

[50] Aplicación de *Fake GPS*: https://play.google.com/store/apps/details?id=com.lexa.fakegps&hl=es&gl=US (último acceso: 15/12/2023).

Al no poder acceder a la *Información Posicionada*, en esta situación simulada de GPS, se pudo verificar que el problema se debía al cálculo de la distancia.

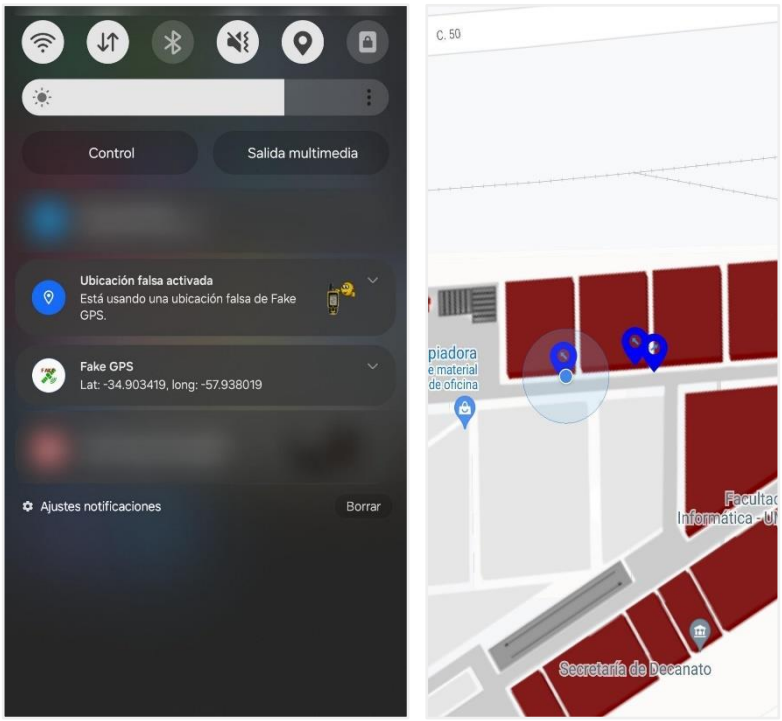


Figura 7.9: Posicionamiento por Fake GPS.

7.1.2 Análisis de las pruebas realizadas

Las pruebas realizadas en la Sección 7.1.1 proporcionaron información valiosa que permitió identificar varios aspectos a mejorar en la herramienta. Aunque uno de los colaboradores no pudo acceder al contenido, los resultados obtenidos y el feedback de los participantes permitieron identificar aspectos a revisar en la herramienta. A continuación, se presentan algunos puntos clave identificados:

- *Limitaciones del GPS*: las diferencias significativas en el funcionamiento del GPS entre los colaboradores destacaron las limitaciones de esta tecnología, especialmente en espacios indoor. Esto resalta la necesidad de considerar alternativas o ajustes para mejorar la precisión de la detección de ubicación.
- *Cálculo de Distancia*: la incapacidad para acceder a la *Información Posicionada* incluso cuando los usuarios estaban en proximidad física con los objetos resaltó la importancia de revisar el método de cálculo de distancia utilizado en la herramienta.
- *Feedback del Usuario*: la posibilidad de recopilar observaciones y comentarios directamente de los usuarios durante la experiencia resultó sumamente importante. Este feedback permitió detectar aspectos de usabilidad de la herramienta y ayudó a identificar áreas de mejora. Por ejemplo, se destacó la necesidad de modificar la forma de presentar los objetos detectados por la cámara, destacando aún más aquellos con *Información Posicionada* asociada por sobre aquellos que no la tienen.

Otro punto interesante a mencionar es que los participantes indicaron que no tuvieron problemas con el modelo utilizado para reconocer las puertas de la Facultad. Durante las pruebas registradas, este modelo alcanzó una precisión media cercana al 40% al detectar las

puertas de frente y un 70% al detectarlas de costado. En la Figura 7.10 se pueden ver dos ejemplos registrados por el *Colaborador 2.* Así mismo, ambos participantes coincidieron en que para lograr el reconocimiento siempre fue necesario apuntar a la puerta durante un par de segundos.

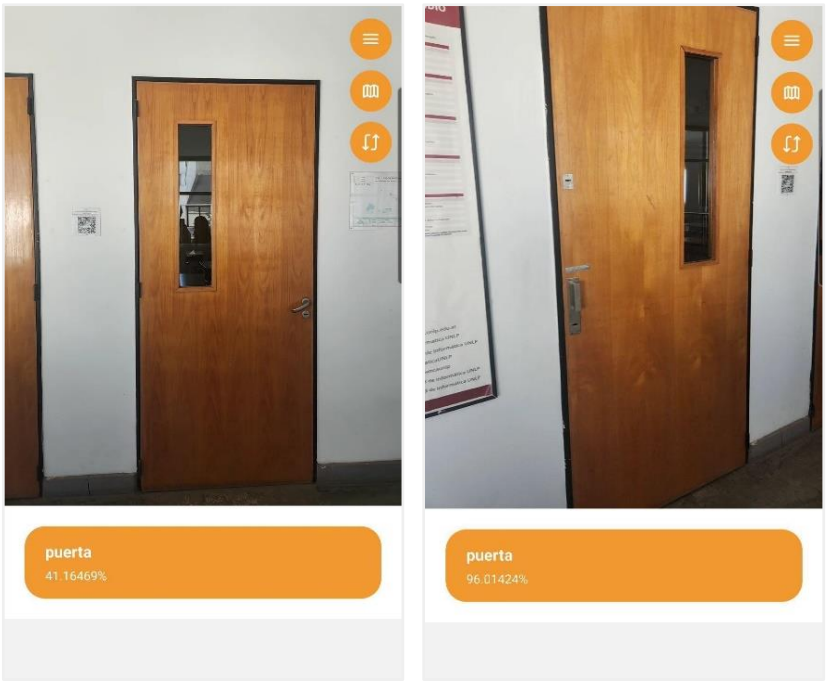


Figura 7.10: Confianza del modelo en las detecciones de frente (izquierda) y de costado (derecha).

7.2 Segunda versión de la herramienta

A partir de los problemas y sugerencias identificadas durante las experiencias de la Sección 7.1, se generó una nueva versión de la herramienta de autor, implementando varios cambios. Concretamente se realizaron las siguientes modificaciones:

- *Configuración del GPS*: se ajustó la configuración de la librería utilizada para obtener datos del GPS. En la versión del Capítulo 6 se empleaba la configuración "*Highest*", la cual ofrecía el mejor nivel de precisión disponible. Sin embargo, se encontró que esta configuración no siempre proporcionaba los resultados más precisos, especialmente en entornos interiores. Por lo tanto, se cambió a "*Best for Navigation*", que ofrece la máxima precisión posible al aprovechar datos adicionales de sensores para facilitar aplicaciones de navegación.
- *Cálculo de Distancia*: se detectó que la función *getDistance*[51] proporcionada por la librería *geolib* no era lo suficientemente precisa en todas las situaciones. Para solucionar este problema, se optó por utilizar ahora la función *isPointWithinRadius*[52], la cual valida si un punto se encuentra dentro de un radio específico de otro punto. El tamaño del radio es un valor numérico que se pasa como parámetro a la función y que

[51] Documentación de la función *getDistance*: https://www.npmjs.com/package/geolib#getdistancestart-end-accuracy--1 (último acceso: 15/12/2023).
[52] Documentación de la función *isPointWithinRadius*: https://www.npmjs.com/package/geolib#ispointwithinradiuspoint-centerpoint-radius (último acceso: 15/12/2023).

mide una distancia en metros. Esta función utiliza internamente otra función llamada *getPreciseDistance,* que también está provista por *geolib*. Cabe mencionar que esta nueva función (*isPointWithinRadius*) demostró ofrecer resultados más precisos en las pruebas realizadas con esta nueva versión de la herramienta.

- *Representación Visual de Objetos Detectados*: en esta nueva versión, los objetos detectados con la cámara del teléfono se muestran en el listado con distintas tonalidades de un mismo color, donde una tonalidad más oscura indica una detección con mayor nivel de seguridad (o *accuracy*). Esta modificación permite a los usuarios discernir rápidamente la fiabilidad de cada detección.
- *Visualización de Objetos sin Información Posicionada asociada*: ahora estos objetos se representan en color gris, lo que facilita su identificación y distinción de los objetos con *Información Posicionada* asociada.
- *Corrección de Problemas de Visualización*: se abordaron varios problemas relacionados con la visualización, como la incorrecta presentación de las flechas en algunos marcadores del mapa, donde no coincidía lo mostrado con el ángulo de visión cargado. Además, se solucionó un problema ocasional que ocurría al intentar acceder al mapa, lo que mejoró la estabilidad y la confiabilidad de la herramienta en su conjunto.

7.3 Pruebas con la segunda versión de la herramienta

En esta sección se presentan dos experiencias que hacen uso de la segunda versión de la herramienta de autor. Por un lado, se describe una segunda iteración de la experiencia enfocada en el desarrollo de la aplicación para un congreso. Por otro lado, se presenta una experiencia destinada a la creación de una aplicación dirigida a niños pequeños, la cual ofrece curiosidades relacionadas con la informática.

Esta segunda fase de pruebas tuvo como objetivo principal evaluar la efectividad de las mejoras realizadas a la herramienta y determinar si los cambios lograron abordar los desafíos identificados durante las pruebas de la primera versión.

7.3.1 Aplicación para un congreso

Esta segunda etapa de pruebas se realizó con los mismos dos participantes y nuevamente de forma in-situ en la Facultad de Informática de la Universidad Nacional de La Plata (UNLP). En esta nueva iteración, el enfoque principal estuvo puesto en lo que ocurría al co-testear las aplicaciones creadas con la primera versión de la herramienta, específicamente usando los *Espacios de Trabajo* "Congreso - Versión 2" y "Congreso - Versión 3".

La decisión de reutilizar estos *Espacios de Trabajo* se justifica en estos puntos clave:

- En primer lugar, aunque surgieron problemas con el GPS, al observar cómo quedaron los marcadores en el mapa (Figuras 7.5 y 7.6), se puede apreciar que el desfase es bastante pequeño y dentro de un margen de error aceptable.
- Además, los cambios realizados en la segunda versión de la herramienta no afectaron la forma de acceder a la estructura de la información en la base de datos. La información cargada en los *Espacios de Trabajo* creados con la primera versión de la

herramienta se mantuvo intacta. La segunda versión se limitó a cambios que afectaron cuestiones visuales o de procesamiento de la información registrada.

Es importante destacar que antes de comenzar las pruebas, se realizaron modificaciones en la configuración de ambos *Espacios de Trabajo*, específicamente en aquellas relacionadas con el *Posicionamiento por Objetos*, restableciéndolas a los valores iniciales definidos en la Sección 7.1.1.2. Por ejemplo, la "*Máxima distancia en metros del objeto para considerarlo"* volvió a ser de 1 metro.

7.3.1.1 Co-testear la aplicación co-diseñada (usar la *Información Posicionada*)

Al comenzar la experiencia, se solicitó a los participantes que instalaran la nueva versión de la herramienta en sus dispositivos. Se les indicó que utilizaran las mismas credenciales proporcionadas en la primera experiencia para acceder al *Espacio de Trabajo* "Congreso - v3", donde la información se registró de frente.

Una vez completado este paso, se les pidió a los participantes que recorrieran el pasillo del primer piso de la Facultad, repitiendo todas las pruebas mencionadas en la Sección 7.1.1.4. Después de cada prueba y antes de comenzar la siguiente, los participantes compartieron sus observaciones sobre el comportamiento de la herramienta. Este proceso fue repetido luego para el *Espacio de Trabajo* "Congreso - Versión 2", donde la información había sido registrada utilizando distintos ángulos de visión.

En esta ocasión, el *Colaborador 2*, responsable de haber cargado la *Información Posicionada* con su teléfono, logró acceder correctamente a la información en todas las pruebas utilizando una configuración de "*Máxima distancia en metros del objeto para considerarlo"* de 1 metro de distancia. Sin embargo, el *Colaborador 1* experimentó dificultades para acceder a esta misma información. Por lo tanto, se optó entonces por aumentar el valor de esta configuración y llevarlo a un valor superior, repitiendo luego las pruebas. Al fijar una distancia máxima de 3 metros, el *Colaborador 1* pudo acceder al contenido sin dificultades en ambos *Espacios de Trabajo*. Aunque esta distancia sigue siendo bastante grande, este avance fue significativo, ya que con esa misma distancia con la primera versión de la herramienta, este colaborador, no había podido acceder a ninguna información.

Es relevante destacar que, durante todas las pruebas que se llevaron a cabo, la herramienta logró mostrar siempre la información correcta para cada aula. Esto es interesante porque el contenido asociado a las puertas de las aulas 1-2 y 1-3 había sido registrado con mucha proximidad. Se decidió continuar con las pruebas, incrementando gradualmente el valor de la configuración "*Máxima distancia en metros del objeto para considerarlo",* con el fin de determinar si al aumentar aún más la distancia la herramienta comenzaba a confundir el contenido de estas aulas*:*

- En el *Espacio de Trabajo* "Congreso - Versión 3", donde toda la información había sido cargada utilizando el mismo ángulo de visión, se logró alcanzar esta situación de confusión de aulas con un valor de la configuración a 6 metros de distancia.
- En el *Espacio de Trabajo* "Congreso - Versión 2", incluso al aumentar la distancia máxima a 15 metros, la información continuaba mostrándose correctamente sin problemas. Esto se debe principalmente a que en este *Espacio de Trabajo* el contenido de las puertas 1-2 y 1-3 se registró con ángulos de visión muy distintos, lo cual permite desambiguar y mostrar la información correcta según la posición del usuario.

Solo al modificar otra configuración del *Posicionamiento por Objetos* como es la *"Apertura del ángulo de visión a considerar"* a un valor mucho mayor (de 70 grados a 250 grados), la herramienta comenzó a confundir el contenido de las aulas.

Estas pruebas llevadas a cabo en las Aulas 1-2 y 1-3 permiten comprender de forma más clara cómo el registro de varios objetos del mismo tipo desde diferentes ángulos puede contribuir a desambiguar la información que se muestra a los usuarios cuando se configura correctamente el *Posicionamiento por Objetos*.

Dado que con esta segunda experiencia ambos participantes lograron acceder correctamente al contenido registrado (algo que no habían podido hacer con la primera versión de la herramienta), se pudieron llevar a cabo otras pruebas para evaluar otros aspectos relacionados con el *Posicionamiento por Objetos*, las cuales se describen a continuación.

En ambos *Espacios de Trabajo* se volvió a configurar la "*Máxima distancia en metros del objeto para considerarlo*" a 3 metros. Se decidió realizar una exploración para comprender qué ocurría al reducir la apertura del ángulo de visión. Para esto, la *"Apertura del ángulo de visión a considerar*" se modificó a 30 grados. Se solicitó a los participantes que accedieran al *Espacio de Trabajo* "Congreso - Versión 2", donde la información se registró desde diferentes ángulos. De esta forma se observó que ya no era posible detectar la *Información Posicionada*. Sin embargo, al aumentar la apertura a 45 grados, la información podía detectarse si los participantes se posicionaban exactamente de la misma manera que al momento del registro de la información.

Otro aspecto identificado se relaciona con la capacidad de la herramienta para reconocer objetos cuando los usuarios se desplazan a distintas velocidades. Se observó que la herramienta no es capaz de reconocer los objetos cuando los usuarios caminan muy rápido, ya que el modelo requiere de un par de segundos para reconocer y procesar el objeto. Esto aconteció en ambos *Espacios de Trabajo*, aun con la configuración óptima que fue la "*Máxima distancia en metros del objeto para considerarlo*" a 3 metros y la *"Apertura del ángulo de visión a considerar"* a 70 grados.

7.3.2 Aplicación con curiosidades de la informática

Esta nueva experiencia se centró en la creación de una aplicación dirigida a niños pequeños que visitan la Facultad de Informática de la Universidad Nacional de La Plata (UNLP). Para esta nueva aplicación se propuso definir en el espacio físico un *stand* (mesa) donde se presentarían distintos objetos relacionados a la tecnología y la informática, tales como un mouse, un teclado y un disco duro. Esta aplicación debería ofrecer información sobre curiosidades relacionadas con estos objetos.

Siguiendo la misma línea que las experiencias previas, esta ocasión también contó con la participación de dos individuos, quienes llevaron a cabo todo el proceso de co-diseño y co-testeo utilizando la herramienta. La experiencia se realizó in-situ y tuvo una duración de un poco más de una hora.

Cabe mencionar que uno de los participantes había formado parte de las experiencias previamente presentadas en este capítulo, el cual participó como *Colaborador 2* (que tenía asignado el color rosa) y siguió usando esas credenciales para esta nueva experiencia. Para la otra persona se registró en la herramienta un nuevo colaborador, el *Colaborador 3,* asignándole el color verde.

El foco de esta experiencia fue analizar cómo se comportaba el *Posicionamiento por Objetos* cuando se usan muchos objetos distintos en un espacio acotado. Específicamente, fue de interés ver cómo se comportaba el GPS al registrar mucha información cercana.

7.3.2.1 Armado de la información para el co-diseño

En esta experiencia también se resolvió definir previamente la información que los participantes debían cargar en la herramienta durante el co-diseño. En este caso, se enfocó en seleccionar datos que resultaran interesantes para niños, centrándose en objetos relacionados con la tecnología y la informática que los modelos disponibles en la herramienta pudieran reconocer.

Todos los detalles de la información de esta experiencia se encuentran en la Tabla D.4 del Anexo D. De todas formas, en la Figura 7.11 se puede apreciar la distribución definida para los objetos en la mesa del stand. El color asociado a cada objeto representa al *Colaborador* que cargó la información dentro de la herramienta de autor: rosa para el *Colaborador 2* y verde para el *Colaborador 3*.

Figura 7.11: Distribución decidida para los objetos en el stand a registrar en la aplicación.

7.3.2.2 Decisiones al crear el *Espacio de Trabajo*

Se definió un único *Espacio de Trabajo (“Curiosidades - Versión 1”)*, el cual se asoció al edificio de la Facultad de Informática de la Universidad Nacional de La Plata (UNLP). Este *Espacio de Trabajo* se configuró con el color rosa.

Dentro del *Espacio de Trabajo* creado, el *Posicionamiento por objetos* fue configurado con los siguientes parámetros:

- *Modelo:* se utilizó el modelo de reconocimiento “MobileNet V2” presentado en la Sección 4.1.2.
- *Clases*: se configuró el modelo para reconocer los siguientes objetos: “Mouse”, “Teclado”, “Disco Duro”, “Computadora portátil”, “Disco Duro” y “Cassette VHS”.

- *Precisión mínima en inferencias:* se usó una precisión mínima de 20%. Es decir, que el modelo debe ser capaz de reconocer los objetos sobre la mesa con un nivel de certeza igual o superior a este porcentaje.
- *Frecuencia de inferencias:* al modelo realiza inferencias sobre 1 de cada 100 frames obtenidos desde la cámara del dispositivo.
- *Máxima distancia en metros del objeto para considerarlo:* se decidió utilizar una distancia de 1 metro.
- *Apertura del ángulo de visión a considerar:* se decidió utilizar un ángulo de apertura de 70 grados.

Tanto al *Colaborador 2* como al *Colaborador 3* se les dio acceso al *Espacio de Trabajo* creado.

7.3.2.3 Co-diseñar la aplicación con Curiosidades de la Informática (crear *Información Posicionada*)

Al iniciar la experiencia, se solicitó a los dos participantes que instalaran la segunda versión de la herramienta de autor. Posteriormente, se les proporcionaron las credenciales de los usuarios creados y se les pidió que accedieran al *Espacio de trabajo*. Los detalles de los dispositivos utilizados pueden encontrarse en la Tabla D.5 del Anexo D.

Después de completar estos pasos, se mostraron a los participantes los diferentes objetos que se exhibirían en el stand (en base a lo definido en la Figura 7.11), como se puede apreciar en la Figura 7.12.

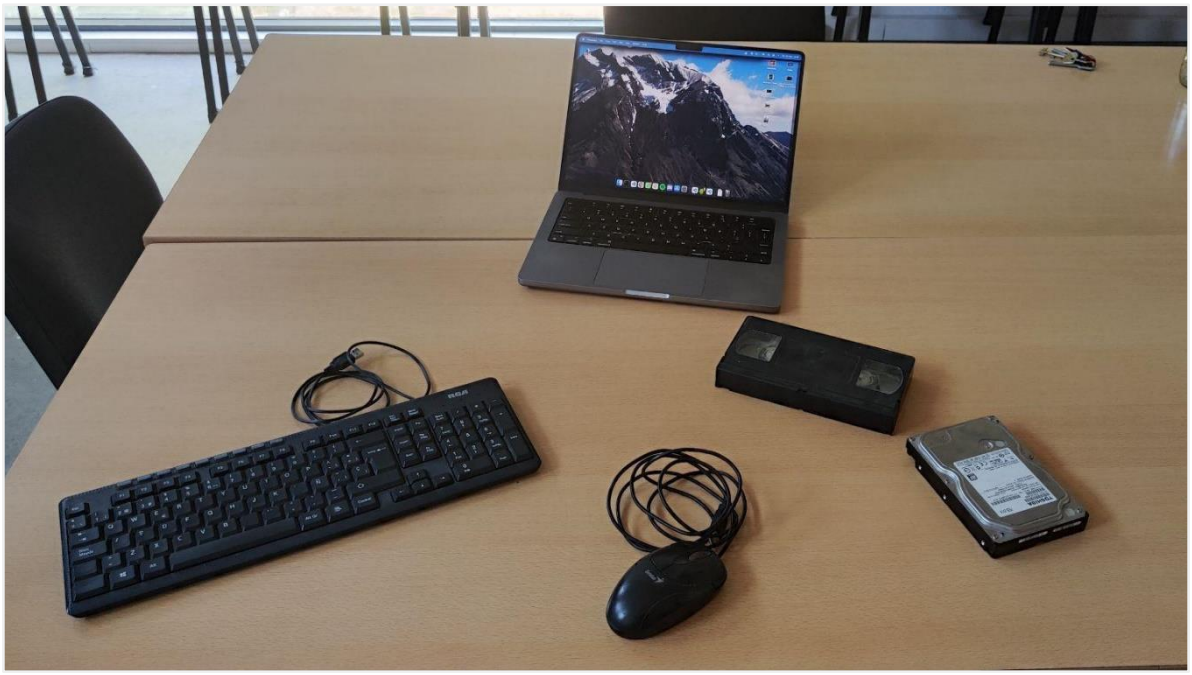


Figura 7.12: Objetos en el stand.

Posteriormente, se les proporcionó el contenido que debían registrar en la herramienta para cada uno de los objetos. El *Colaborador 3* se encargó de registrar el contenido asociado a tres objetos (teclado, mouse y VHS), mientras que el *Colaborador 2* se encargó de los dos restantes (disco duro y computadora portátil). Cabe resaltar que se les pidió que registren el contenido estando de frente al *stand*. En la Figura 7.13 se presenta la información registrada

en la herramienta. Los marcadores de color rosa fueron cargados por el *Colaborador 2*, mientras que los verdes fueron añadidos por el *Colaborador 3.*

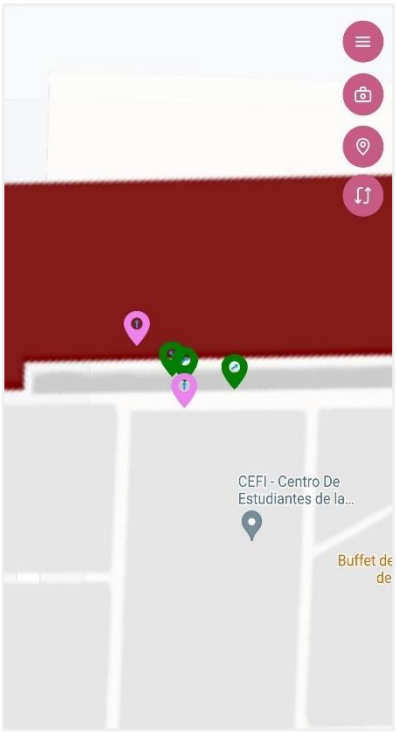


Figura 7.13: *Información Posicionada* registrada en la herramienta.

Adicionalmente, se puede observar en la Figura 7.13 que la precisión del GPS resultó ser, nuevamente, menos exacta de lo esperado. Aunque los objetos registrados estaban muy cerca entre sí (como se muestra en la Figura 7.12) el GPS de los dispositivos utilizados los ubicó en posiciones distantes, algunas incluso a más de 3 metros de distancia entre sí.

Para poder comprender mejor cómo quedó distribuida la información dentro del *Espacio de Trabajo*, se presenta la Figura 7.14. Esta figura presenta una comparación entre la distribución real de los objetos sobre la mesa (Figura 7.14.a) y lo registrado por el GPS de los dispositivos (Figura 7.14.b).

a. **b.**

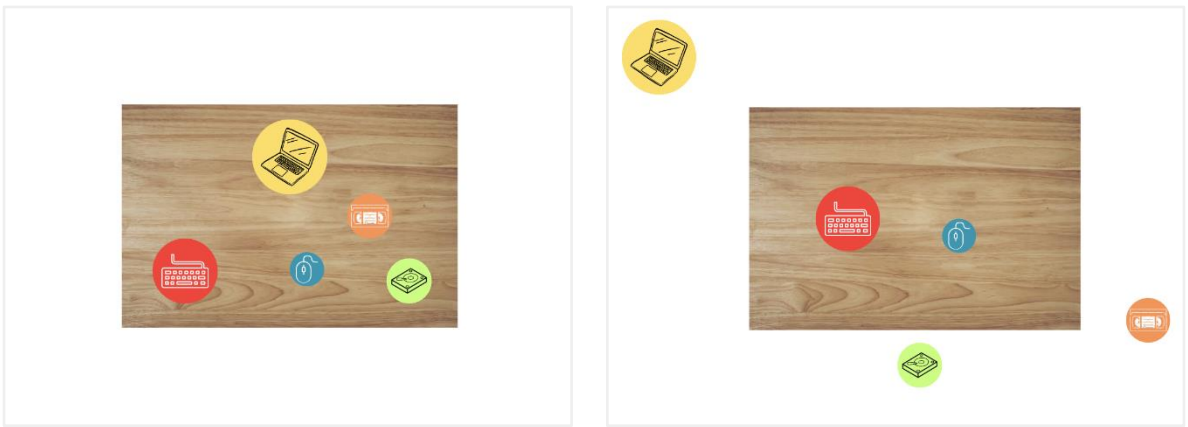


Figura 7.14: Comparación entre la ubicación real de los objetos al momento del registro (a) y lo indicado por el GPS al momento de registrar la *Información Posiciona*da (b).

Cabe resaltar que si bien la precisión del GPS presentó fallas al posicionar los objetos, el rango de error presentado se mantiene dentro de los límites aceptables.

7.3.2.4 Co-testear la aplicación co-diseñada (usar la *Información Posicionada*)

Para llevar a cabo las pruebas, se utilizó inicialmente la configuración establecida en la Sección 7.3.2.2.

Durante estas pruebas, se solicitó a ambos participantes que intentaran acceder a la *Información Posicionada* mientras se encontraban de frente al stand. Posteriormente, se solicitó a los participantes que compartieran su experiencia utilizando la herramienta.

En relación al funcionamiento del modelo de reconocimiento, los participantes destacaron que la herramienta pudo reconocer todos los objetos presentes en el stand con un nivel de precisión que oscilaba entre el 40% y el 80% en la mayoría de los casos. Sin embargo, debido a los problemas previamente identificados con el GPS, no siempre pudieron acceder a la *Información Posicionada* asociada a esos objetos.

Específicamente, utilizando la configuración establecida previamente (según se detalla en la Sección 7.3.2.2), se encontraron resultados dispares entre los participantes. Por un lado, el *Colaborador 2* reportó que solo pudo reconocer la notebook, un objeto que él mismo había registrado. Por otro lado, el *Colaborador 3* experimentó dificultades aún mayores, ya que no logró reconocer ninguno de los objetos presentes en el stand.

A partir de esto, se tomó la decisión de revisar y ajustar la configuración asociada al parámetro "*Máxima distancia en metros del objeto para considerarlo"* para llevarlo a un valor mayor*:*

- Con una configuración de 2 metros de distancia, se observó que el *Colaborador 2* logró reconocer exitosamente los dos objetos que había registrado previamente en el sistema, es decir, el disco duro y la computadora portátil. Sin embargo, no pudo identificar los objetos registrados por el *Colaborador 3*, incluyendo el teclado, el mouse y el VHS. De manera similar, el *Colaborador 3* solo pudo reconocer los objetos que había registrado él mismo, sin lograr identificar los objetos registrados por el otro colaborador.
- Por otro lado, al ajustar la configuración a una distancia de 3 metros, se observó una mejora significativa en los resultados. Ambos colaboradores pudieron acceder a toda la información asociada a los objetos presentes en el stand.

Estas pruebas permitieron apreciar que también para esta experiencia la "*Máxima distancia en metros del objeto para considerarlo"* de 3 metros es la más óptima.

Adicionalmente, al analizar el funcionamiento del modelo de *MobileNet*, se observó una particularidad interesante: con la configuración de "*Precisión mínima en inferencias*" en su valor por defecto, establecido en un 20%, se logró el reconocimiento de todos los objetos, aunque con una leve demora de aproximadamente 5 segundos. Sin embargo, al reducir el umbral mínimo de precisión al 10%, se observó que el reconocimiento se llevaba a cabo de manera prácticamente instantánea, aunque con niveles de precisión ligeramente inferiores, alcanzando alrededor del 17%.

Al analizar más detenidamente este fenómeno, se descubrió un patrón interesante: cada vez que el modelo se encontraba con un nuevo objeto, era capaz de reconocerlo, pero con un porcentaje de precisión inicialmente bajo. A medida que transcurría el tiempo, este porcentaje de precisión tendía a aumentar gradualmente. Este comportamiento se ilustra de manera

gráfica en la Figura 7.13, donde se presenta un ejemplo concreto de lo anteriormente descripto.

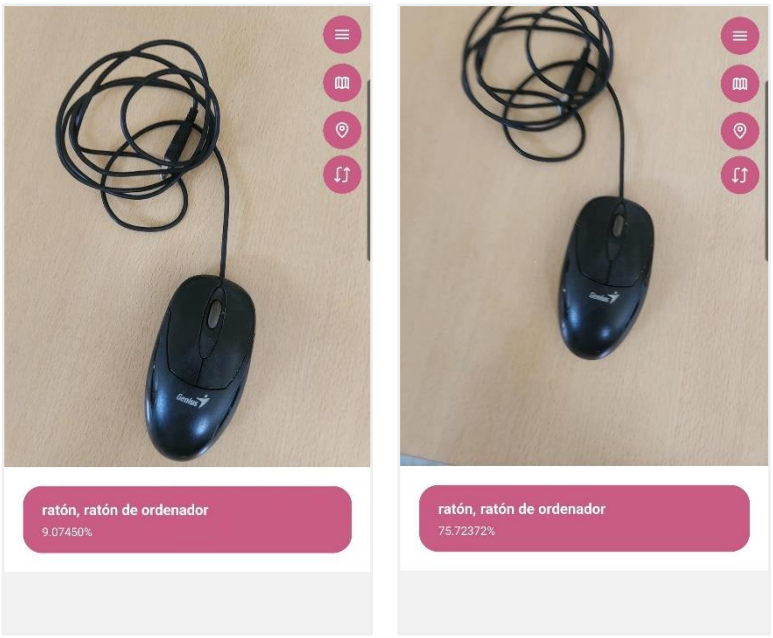


Figura 7.13: Cambios de precisión en el modelo de *MobileNet* luego de unos segundos.

Otro aspecto a destacar fue que al modificar la configuración de la "Frecuencia de inferencias" de su valor por defecto (100 frames) a un valor más bajo, como 10 frames, la herramienta comenzaba a generar resultados con precisiones más altas de manera casi instantánea para el Colaborador 2. Sin embargo, esta modificación ocasionaba que la aplicación empezara a experimentar ciertas demoras en las respuestas o incluso congelamientos de la aplicación con el teléfono del Colaborador 3. Este teléfono era de gama media, con 4GB de memoria.

7.3.3 Análisis de las pruebas realizadas

En la Sección 7.3 se llevó a cabo el co-diseño y co-testeo de dos aplicaciones utilizando la segunda versión de la herramienta de autor desarrollada. Estas pruebas permitieron identificar varios aspectos significativos relacionados al funcionamiento del *Posicionamiento por Objetos.*

Por un lado, la experiencia de la aplicación del *Congreso* (Sección 7.3.1) demostró cómo los cambios introducidos en la segunda versión de la herramienta representaron una mejora notable en la usabilidad durante la etapa de co-testeo. Todos los participantes pudieron acceder a la *Información Posicionada*, algo que no habían logrado hacer con la primera versión de la herramienta (como se detalla en la Sección 7.1.1.4). Además, se observó que la solución de *Posicionamiento por Objeto* resulta eficaz incluso cuando se registran múltiples objetos del mismo tipo en un espacio limitado, siempre y cuando se utilicen diversos ángulos de visión. No obstante, esto implica que los usuarios deben reconocer los objetos desde una orientación específica.

Por otro lado, la experiencia de la aplicación de *Curiosidades de la Informática* (presentada en la Sección 7.3.2) reveló que, a pesar de las mejoras implementadas en el GPS en la segunda versión de la herramienta, aún persistían diferencias en su funcionamiento entre distintos dispositivos, un hecho que se hizo evidente durante el proceso de co-diseño en la

Sección 7.3.2.3. Estas discrepancias señalan que aún quedan desafíos por resolver con respecto al GPS.

Una posible solución para este problema podría ser incorporar a la herramienta una etapa donde se pueda evaluar y corregir el desfase del GPS del usuario. Esta fase podría involucrar la comparación de la ubicación indicada por el GPS con un punto de referencia conocido, permitiendo así realizar las correcciones para mejorar la precisión del posicionamiento.

Aunque otra opción viable podría ser emplear un mecanismo de posicionamiento distinto, el uso del GPS demostró ser adecuado a pesar del desfase observado. Tanto en la experiencia de la aplicación del *Congreso* (Sección 7.3.1) como en la de *Curiosidades de la Informática* (Sección 7.3.2), se observó que con una configuración de "*Máxima distancia en metros del objeto para considerarlo"* de distancia de 3 metros, todos los participantes podían acceder a la *Información Posicionada* registrada en los *Espacios de Trabajo*, lo que sugiere que esta configuración garantiza una experiencia adecuada para todos los usuarios.

Una limitación identificada de la herramienta se presenta cuando se recorre el ambiente de forma muy rápida. En ambas experiencias, se observó que los modelos utilizados requieren unos segundos para reconocer adecuadamente los objetos. Esto implica la necesidad de caminar lentamente o, al menos, detenerse durante un par de segundos frente al objeto para permitir que la herramienta realice correctamente su función. Si bien esto puede mejorarse con cambios en las configuraciones del *Posicionamiento por Objetos*, por ejemplo, reduciendo el valor de la configuración de la "*Frecuencia de inferencias"* o el valor de la "*Precisión mínima en inferencias"*. *E*sto puede impactar en el funcionamiento de algunos dispositivos, como ocurrió con el *Colaborador 3* (lo cual se detalla en la Sección 7.3.2.4).

Si bien estos ajustes en las configuraciones podrían mejorar aún más la experiencia para dispositivos de gama alta, los participantes (*Colaboradores 1, 2 y 3*) destacaron el buen rendimiento general de la segunda versión de la herramienta de autor.

8. Conclusiones y Trabajos futuros

En este capítulo, se enuncian las conclusiones derivadas de la presente tesis, para luego dar lugar a algunos trabajos futuros que se desprenden de la misma.

8.1 Conclusiones

En este trabajo se presentó una arquitectura para llevar a cabo el posicionamiento en espacios *indoor* utilizando modelos de reconocimiento y detección de objetos, de una manera genérica. Esta solución toma el nombre de *Posicionamiento por Objetos y* fue diseñada para funcionar de forma desacoplada, pudiendo ser embebida en diferentes tipos de aplicaciones. Además, se presentó una posible implementación de esta solución que combina el uso de modelos de reconocimiento y detección de objetos (que se ejecutan localmente en dispositivos móviles), con la información del GPS y Giroscopio para lograr el posicionamiento de los usuarios. Finalmente, para poder probar esta implementación, se desarrolló una herramienta prototípica que permite el co-diseño y co-testeo in-situ de un tipo particular de aplicación móvil sensible al contexto. Esta herramienta tiene embebido el *Posicionamiento por Objetos.*

Alcanzar la solución de *Posicionamiento por Objetos* y desarrollar la herramienta propuesta requirió abordar diversos aspectos a lo largo de la tesis, estableciendo así las bases necesarias para su diseño y desarrollo. A continuación, se presentan algunas conclusiones específicas en relación con cada capítulo de la tesis.

En primer lugar, fue esencial adquirir conocimientos sólidos sobre el uso de modelos de reconocimiento y detección de objetos. Si bien en el curso de "*Aprendizaje Automático*" se abordaron conceptos como el de *Visión por Computadora*, clasificación de imágenes (tanto con *machine learning* como con *deep learning*) y *transfer learning*, fue necesario profundizar más en estos temas. Por lo tanto, se decidió ampliar y fortalecer los conceptos aprendidos y explorar con mayor detalle los fundamentos teóricos, las aplicaciones prácticas y las últimas tendencias en relación con los modelos de reconocimiento y detección de objetos. Esto permitió sentar bases para el análisis presentado en el Capítulo 2, donde se describe cómo estos modelos fueron evolucionando con el tiempo, desde las técnicas tradicionales de *machine learning* hasta la adopción de *Redes Neuronales Convolucionales* (CNN) en *deep learning*.

De la exploración mencionada anteriormente, también surgieron dos nuevos conceptos de gran importancia para este trabajo, los cuales también fueron detallados en el Capítulo 2. Por un lado, se introdujo el concepto de *lightweight networks*, que se refiere a modelos que utilizan redes diseñadas para funcionar de manera eficiente en dispositivos con recursos limitados, como los smartphones. Por otro lado, se abordó el concepto de *vision-based indoor positioning*, el cual propone el uso de técnicas de visión por computadora para lograr el posicionamiento del usuario en entornos *indoor*. Estos dos conceptos permitieron sentar las bases teóricas necesarias para el posterior diseño y desarrollo de la solución propuesta de *Posicionamiento por Objetos*.

Para evaluar la viabilidad de implementar modelos de reconocimiento y detección de objetos que corren localmente en los dispositivos móviles, se llevaron a cabo pruebas utilizando aplicaciones prototípicas. Este proceso implicó realizar previamente una investigación (la cual

se detalla en el Capítulo 3) centrada en las herramientas disponibles, con un enfoque especial en *TensorFlow* y sus variantes, *TensorFlow Lite* y *TensorFlow.js*.

Estas tecnologías permitieron comprender cómo se comportan los modelos de reconocimiento y detección de objetos en dispositivos reales, así como su rendimiento en términos de precisión y eficiencia. Durante el análisis de las aplicaciones prototípicas, el cual se describe en el Capítulo 4, se confirmó que tanto *TensorFlow Lite* como *TensorFlow.js* eran adecuados para su uso en los dispositivos. Esta conclusión permitió tomar la decisión de optar por *TensorFlow.js*, en parte debido a la familiaridad del autor con el lenguaje de programación *JavaScript*. Esta elección facilitó el proceso de desarrollo e implementación de la solución y la herramienta.

Otra parte importante de la investigación fue aprender cómo crear un modelo propio, tal como se detalla en el Capítulo 4. Este proceso permitió poner en práctica muchos de los conceptos teóricos aprendidos durante el curso de "*Aprendizaje Automático*". Cabe mencionar que la curva de aprendizaje resultó ser más compleja de lo anticipado. Por ejemplo, se encontraron desafíos al definir el *dataset* de imágenes y sobre todo al comprender y ajustar los parámetros para entrenar a la red neuronal.

Tomando de base la investigación y las pruebas realizadas en los capítulos anteriores, en el Capítulo 5 se presentó el diseño de la solución de *Posicionamiento por Objetos*. Esta solución aprovecha modelos de reconocimiento y de detección de objetos para el posicionamiento *indoor* de los usuarios. Aunque la arquitectura presentada es genérica y permite la combinación de los modelos de reconocimiento y detección de objetos con cualquier mecanismo de sensado, se optó por presentar una implementación específica que utiliza el GPS y el Giroscopio del dispositivo.

Asimismo, en el Capítulo 5 se introdujo el diseño de la solución de *Monitoreo de Eventos*, cuyo propósito es recopilar datos relevantes para su posterior análisis y visualización. Aunque la arquitectura de esta solución es genérica, se desarrolló una implementación específica que registra eventos particulares relacionados con la implementación del *Posicionamiento por Objetos*.

En las implementaciones del *Posicionamiento por Objetos* y el *Monitoreo de Eventos* se pudieron aprovechar conceptos incorporados durante el curso de "*Captura y Almacenamiento de la Información*". Específicamente en lo relacionado a la captura de información desde sensores y al manejo de bases de datos noSQL, como lo es *Firebase*.

Para evaluar la viabilidad de la solución del *Posicionamiento por Objetos*, se desarrolló una herramienta prototípica (detallada en el Capítulo 6 de este trabajo). Esta herramienta fue desarrollada con el propósito de facilitar el co-diseño y co-testeo in-situ de aplicaciones móviles sensibles al contexto, en línea con los objetivos de investigación de la beca de maestría del autor de este trabajo.

Además, la herramienta integra una implementación del *Monitoreo de Eventos*, añadiendo una capa adicional de funcionalidad para recopilar datos relevantes durante el proceso de prueba y desarrollo de aplicaciones. Con el objetivo de proporcionar información útil para la toma de decisiones durante el proceso de co-testeo, la herramienta permite visualizar diversas métricas y gráficos estadísticos basados en la información recopilada del *Monitoreo de Eventos.* Para lograr esto, se aprovecharon varias técnicas de preprocesamiento de datos y transformaciones adquiridas durante el curso de "*Minería de Datos*", así como métodos para detectar outliers y otros algoritmos. Además, del curso de "*Visualización de Grandes*

Volúmenes de Datos", se tuvieron en cuenta aspectos de diseño visual y técnicas de visualización para definir los gráficos de la herramienta, garantizando así una presentación efectiva y comprensible de los datos recopilados.

En el Capítulo 7, por su parte, se presentaron varias experiencias donde se buscó analizar la eficacia del *Posicionamiento por Objetos* dentro de la herramienta desarrollada. La primera de estas experiencias, detallada en la Sección 7.1.1, permitió identificar algunos problemas y áreas de mejora en la herramienta. Si bien los problemas detectados no permitieron probar en detalle el funcionamiento del *Posicionamiento por Objetos*, los hallazgos obtenidos condujeron a la implementación de una segunda versión de la herramienta. Esta nueva versión logró abordar muchos de los problemas detectados y mejoró significativamente la funcionalidad general.

Con las modificaciones introducidas en la segunda versión de la herramienta, se llevaron a cabo dos nuevas experiencias (detalladas en la Sección 7.3). En la primera de estas pruebas, los participantes pudieron completar el proceso de co-diseño y co-testeo, logrando acceder al contenido que habían creado previamente con la primera versión de la herramienta. Con las nuevas experiencias se comprobó, además, que la solución de *Posicionamiento por Objetos* funciona para reconocer objetos distintos en espacios limitados, así como para detectar objetos del mismo tipo registrados desde diferentes ángulos de visión. Sin embargo, se observó que la solución no es adecuada cuando se utiliza un mismo tipo de objeto en un espacio reducido (menos de 3 metros) y utilizando ángulos de visión similares.

Algo que se debe tener en cuenta, y que se observó durante las pruebas, es la necesidad de revisar y ajustar aspectos específicos de la configuración del *Posicionamiento por Objetos* para garantizar un funcionamiento óptimo. Por ejemplo, en los escenarios probados, se detectó que el contenido podía ser accedido correctamente cuando se usaba una configuración de "*Máxima distancia en metros del objeto para considerarlo"* de 3 metros y una configuración de "*Apertura del ángulo de visión a considerar"* de 70 grados. Esto, de todas formas, puede variar de acuerdo con las necesidades y al espacio sobre el que se va a co-diseñar. Además, se podrían ajustar los valores de las configuraciones de "*Frecuencia de inferencias"* según las capacidades de los dispositivos utilizados. Este ajuste podría mejorar la experiencia del usuario a costa de un mayor consumo computacional.

Es relevante resaltar que, durante las pruebas, se pudo observar la efectividad del *Giroscopio* del teléfono para desambiguar la información cercana. Este aspecto se hizo evidente en la Sección 7.3.1.1, donde se constató que la aplicación de "Congreso - Versión 3" siempre mostraba la información correcta. Esto se debió a que la *Información Posicionada* en este *Espacio de Trabajo* se registró con diferentes ángulos de visión. Además, cabe destacar que tanto el modelo entrenado (para reconocer puertas) como el modelo preentrenado de *MobileNet* de *TensorFlow* demostraron ser eficaces para reconocer los objetos propuestos, con un nivel de confianza adecuado.

Sin embargo, en relación con el GPS, se pudo evidenciar con las pruebas de co-diseño de la Sección 7.3.2 que todavía persisten problemas para posicionar correctamente al usuario en espacios indoor. También se evidenció que existen variaciones en el funcionamiento de este entre diferentes dispositivos, con distintos niveles de desfasaje. Lo que presenta un foco a investigar en el futuro. A pesar de esta problemática, los desfases detectados en el GPS se mantuvieron dentro de rangos de error aceptables, permitiendo a los participantes de las experiencias acceder al contenido registrado.

Por otra parte, se identificó una limitación en el rendimiento de la herramienta cuando se recorre el ambiente de forma muy rápida, ya que los modelos utilizados requieren unos segundos para reconocer adecuadamente los objetos. Esto sugiere la necesidad de caminar lentamente o detenerse frente al objeto para permitir que la herramienta realice su función correctamente.

Por último, es relevante mencionar que parte de la investigación de esta tesis fue presentada en el congreso JCC-BD&ET 2024 y ha sido aceptada para su publicación por la editorial Springer Nature. Esta publicación no se incluyó en la bibliografía, ya que la aceptación de la misma aconteció una vez entregada la tesis para su evaluación. A continuación, se proporciona la referencia de esta publicación:

> Borrelli, F.M., Challiol, C. (2025). Object Recognition Models for Indoor Users' Location. In: Naiouf, M., De Giusti, L., Chichizola, F., Libutti, L. (eds) Cloud Computing, Big Data and Emerging Topics. JCC-BD&ET 2024. Communications in Computer and Information Science, vol 2189. Springer, Cham.

8.2 Trabajos Futuros

A continuación, se mencionan algunos posibles trabajos futuros que se desprenden de la presente tesis. En relación con la solución de *Posicionamiento por objetos* presentada en el Capítulo 5*:*

- *Mejoras en el funcionamiento del GPS*

  Con la experiencia presentada en la Sección 7.3.2, se pudo notar que aún con los cambios introducidos en la segunda versión de la herramienta, el GPS sigue presentando dificultades para posicionarse de forma exacta dentro de espacios *indoor*. Esto se notó sobre todo durante el proceso de co-diseño, donde los participantes tuvieron que registrar la *Información Posicionada*.

  Con esta experiencia se evidenció no solo una discrepancia entre la posición real del usuario y la proporcionada por el GPS, sino también la variabilidad de esta discrepancia según el dispositivo utilizado. Ante esta situación, una posible línea de investigación podría ser la búsqueda de estrategias para mejorar la precisión y reducir el desfase del GPS. Para ello, se podría considerar la implementación de una función de calibración del GPS a nivel de usuario dentro de la herramienta de autor. Esta función permitiría a los usuarios realizar ajustes manuales o automáticos para corregir las discrepancias de posicionamiento, basándose en puntos de referencia conocidos y así mejorar la precisión de la ubicación.

- *Investigar otros mecanismos de sensado para el Posicionamiento por Objetos*

  Dadas las problemáticas identificadas con el GPS, sería interesante explorar cómo funciona la solución de *Posicionamiento por Objetos* en conjunto con tecnologías complementarias como el posicionamiento por Wi-Fi, Bluetooth o sensores de proximidad. Este enfoque podría ofrecer una alternativa viable para mejorar la precisión y la consistencia del posicionamiento en entornos *indoor* donde el GPS no resulta confiable. Sin embargo, podrían requerir una infraestructura adicional, por lo que será necesario evaluar los costos asociados a su implementación, así como encontrar soluciones escalables que se adapten a cualquier espacio físico.

- *Definir otras herramientas que hagan uso de la solución de Posicionamiento por Objetos*

  Como se describe en el Capítulo 5, la solución de *Posicionamiento por Objetos* es versátil y puede ser embebida en diferentes tipos de aplicaciones. La herramienta presentada en el Capítulo 6, en particular, utiliza el *Posicionamiento por Objetos* para llevar a cabo las acciones de crear (para registrar contenido) y usar (para acceder al mismo). Un trabajo futuro podría ser investigar si al incorporar la solución de *Posicionamiento por Objetos* en dos aplicaciones distintas (una para crear y otra para usar), se encuentran aspectos no considerados que puedan tener un impacto significativo en su viabilidad.

En relación con la herramienta de autor presentada en el Capítulo 6:

- *Realizar más experiencias de usuarios*

  Las pruebas con usuarios resultan de gran relevancia, ya que permiten, a partir del feedback obtenido, poder mejorar la herramienta agregando o modificando funcionalidades, mejorando así la experiencia del usuario y el proceso de co-diseño y co-testeo. En particular, será de interés realizar nuevas experiencias donde participe un mayor número de usuarios. Cabe mencionar que en las experiencias presentadas en el Capítulo 7, solo participaron dos usuarios en cada una de estas.

  De esta forma, además, se podrá evaluar el rendimiento de la herramienta, (y sobre todo los modelos de reconocimiento y/o detección utilizados) en una gama mayor de dispositivos con diferentes capacidades de hardware y sistemas operativos. Esto ayudaría a comprender mejor los diferentes entornos y cómo se puede optimizar para ofrecer una experiencia consistente.

- *Métricas y estadísticas de la herramienta*

  Aunque las pruebas presentadas en este trabajo se centraron en analizar el rendimiento del *Posicionamiento por Objetos*, quedaron aspectos abiertos que podrían requerir una investigación adicional, especialmente en relación con cómo las métricas de la herramienta.

  Se espera que, al llevar a cabo nuevas experiencias, se pueda examinar más detenidamente si las métricas y estadísticas proporcionadas por la herramienta son adecuadas para el proceso de co-testeo y para la toma de decisiones. Estas nuevas experiencias podrían ayudar a validar la relevancia y utilidad de las métricas existentes, así como a identificar áreas donde se necesiten métricas adicionales para capturar aspectos importantes del rendimiento de la herramienta.

Bibliografía

[Al-Azzoa et al., 2018] Al-Azzoa F., Taqia A.M., Milanovab M. (2018). Human Related-Health Actions Detection using Android Camera based on TensorFlow Object Detection API. International Journal of Advanced Computer Science and Applications, 9, pp. 9-23.

[Alegre-Ibarra et al., 2018] Alegre-Ibarra, U., Augusto, J. C., Evans, C. (2018). Perspectives on engineering more usable context-aware systems. Journal of Ambient Intelligence and Humanized Computing 9(5), pp. 1593-1609.

[Augusto et al., 2017] Augusto, J., Aztiria, A., Kramer, D., Alegre, U. (2017). A survey on the evolution of the notion of context-awareness. Applied Artificial Intelligence 31(7-8), pp. 613-642.

[Bagchi et al., 2020] Bagchi, S., Aggarwal, V., Chaterji, S., Douglis, F., El Gamal, A., Han, J., Henz, B. J., Hoffmann, H., Jana, S., Kulkarni, M., Lin, F. X., Marais, K. B., Mittal, P., Mou, S., Qiu, X., Scutari, G. (2020). Vision Paper: Grand Challenges in Resilience: Autonomous System Resilience through Design and Runtime Measures. IEEE Open Journal of the Coputer Society 1 (2020), pp. 155–172.

[Borrelli, 2021] Borrelli, F.M. (2021). Co-diseño in-situ y creación de juegos móviles basados en posicionamiento en espacios indoor (Universidad Nacional de La Plata).

[Borrelli et al., 2018] Borrelli, F.M., Brost, P., Challiol, C., Orellano, D.H., Mendibru, F.I., Santoleri, M.A., Alconada Verzini, F.M. (2018). Desarrollo multiplataforma de Aplicaciones Móviles combinadas con el uso de Beacons. Libro de Actas del XXIV Congreso Argentino de Ciencias de la Computación (CACIC 2018), pp. 847-856.

[Borrelli et al., 2021] Borrelli, F.M., Rouaux Servat, C.M., Goin Plexevi, F., Challiol, C. (2021). Co-diseño distribuido combinando una Herramienta de Autor con recursos de Design Thinking. Simposio Argentino de Ingeniería de Software (ASSE 2021) – JAIIO, pp. 82-95.

[Borrelli et al., 2022] Borrelli, F.M., Rouaux Servat, C.M., Goin Plexevi, F., Challiol, C. (2022). Co-diseño distribuido sincrónico-asincrónico: combinar una Herramienta de Autor con recursos de Design Thinking. Electronic Journal of SADIO (EJS) 21(2), pp. 60-78.

[Cai et al., 2021] Cai, Y., Li, H., Yuan, G., Niu, W., Li, Y., Tang, X., Ren, B., Wang, Y. (2021). YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design. In Proceedings of the AAAI Conference on Artificial Intelligence 35(2), pp. 955-963.

[Challiol et al., 2019] Challiol, C., Borrelli, F.M., Mendiburu, F.I., Rouaux Servat, C.M., Goin Plexevi, F., Orellano, D.H., Gomez-Torres, E., Gordillo, S.E. (2019). Design Thinking's resources for in-situ co-design of mobile games. In Proceedings of 4th International Conference on Information Systems and Computer Science – INCISCO 2019. IEEE, pp. 339 - 345.

[Challiol et al., 2020] Challiol, C., Borrelli, F.M., Mendiburu, F.I., Goin Plexevi, F., Rouaux Servat, C.M., Orellano, D.H., Gomez-Torres, E., Gordillo, S.E. (2019). Co-diseño in-situ de Juegos Móviles usando un abordaje con recursos de Design Thinking. Enfoque UTE 11(1), pp. 1-14.

[Choudhari et al., 2021] Choudhari, V., Phadtare, M., Pedram, R., Vartak, S. (2021). Comparison between YOLO and SSD MobileNet for Object Detection in a Surveillance

Drone. International Journal Of Scientific Research In Engineering And Management (IJSREM), 5(10), pp. 1-5.

[Dey, 2000] Dey, A.K. (2000). Providing Architectural Support for Building Context-Aware Applications. PhD Thesis. Georgia: Georgia Institute of Technology.

[Estrada et al., 2022] Estrada, J., Paheding, S., Yang, X., Niyaz, Q. (2022). Deep-learning-incorporated augmented reality application for engineering lab training. Applied Sciences, 12(10), pp. 5159.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

[Hafiz and Bhat, 2020] Hafiz, A., Bhat, G. (2020). A survey on instance segmentation: state of the art. International Journal of Multimedia Information Retrieval, 9, pp. 171–189.

[Hargood et al., 2018] Hargood, C., Weal, M.J., Millard, D.E. (2018). The storyplaces platform: Building a web-based locative hypertext system. In Proceedings of the 29th ACM Conference on Hypertext and Social Media (HT'18). Association for Computing Machinery, pp. 128-135.

[He et al., 2015] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, pp. 1026-1034.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems (NeurIPS), 25, pp. 1097-1105.

[LeBlanc and Chaput, 2017] LeBlanc, A.G., Chaput, J.P. (2017). Pokémon Go: A game changer for the physical inactivity crisis?. Preventive medicine, 101, pp. 235-237.

[LeCun et al. 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), pp. 541-551.

[Lin et al., 2017] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S. (2017). Feature pyramid networks for object detection. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 936-944.

[Luo et al., 2020] Luo, C., Goncalves, J., Velloso, E., Kostakos, V. (2020). A Survey of Context Simulation for Testing Mobile Context-Aware Applications. ACM Computing Surveys (CSUR) 53(1), pp. 1-39.

[Mautz, 2012] Mautz, R. (2012). Indoor positioning technologies. ETH Zurich, Department of Civil, (Environmental and Geomatic Engineering, Institute of Geodesy and Photogrammetry).

[Shallu and Mehra, 2018] Shallu S., Mehra, R. (2018). Breast cancer histology images classification: Training from scratch or transfer learning?. ICT Express, 4(4), pp. 247-254.

[Mehta et al., 2018] Mehta J., Ramnani E., Singh S. (2018) Face Detection and Tagging Using Deep Learning," 2018 International Conference on Computer, Communication, and Signal Processing (ICCCSP), Chennai, India, 2018, pp. 1-6.

[Oquab et al., 2014] Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 1717-1724.

[Ozdemir and Kunduraci, 2022] Ozdemir, D., Kunduraci, M.S. (2022). Comparison of Deep Learning Techniques for Classification of the Insects in Order Level With Mobile Software Application. IEEE Access 10, pp. 35675-35684.

[ReactNative] Página oficial de React Native. https://reactnative.dev/ (último acceso: 10/12/2023)

[Redmon and Farhadi, 2017] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 6517-6525.

[Ruizhi and Liang, 2017] Ruizhi, C.; Liang, C. (2017). Indoor Positioning with Smartphones: The State-of-the-art and the Challenges. Acta Geod.Cartogr. Sin. 2017, 46, pp. 1316–1326.

[Russakovsky et at., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., & Fei-Fei, L. (2014). (2015) ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), pp. 211–252.

[Salunkhe et al., 2021] Salunkhe A., Raut M., Santra S., Bhagwat S. (2021) Android-based object recognition application for visually impaired. ITM Web of Conferences, 40(4), 03001.

[Sanders and Stappers, 2008] Sanders E.B.N, Stappers P. (2008). Co-creation and the New Landscapes of Design. CoDesign, 4(1), pp. 5-18.

[Sarker, 2021] Sarker, I.H. (2021). Data Science and Analytics: An Overview from Data-Driven Smart Computing, Decision-Making and Applications Perspective. SN Computer Science, 2(5), 377.

[Shafer et al., 1998] Shafer, S.A., Krumm, J., Brumitt, B., Meyers, B., Czerwinski, M., & Robbins, D.C. (1998). The New EasyLiving Project. Microsoft Research.

[Taqi et al., 2019] Taqi, A., Azzo, F., Awad, A., Milanova, M. (2019). Skin Lesion Detection by Android Camera based on SSD-Mobilenet and TensorFlow Object Detection API. Journal of Advanced Research, 3(1), pp. 6-12.

[Tensorflow] Página oficial de la documentación de Tensorflow. www.tensorflow.org (último acceso: 17/09/2023)

[TensorflowLite] Página oficial de la documentación de Tensorflow Lite. www.tensorflow.org/lite (último acceso: 17/09/2023)

[TensorflowJS] Página oficial de la documentación de Tensorflow JS. https://www.tensorflow.org/js (último acceso: 17/09/2023).

[Wang et al., 2022] Wang C.Y., Bochkovskiy A., Liao H.Y, M. (2022). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7464-7475.

[Wei et al., 2014] Wei, Y., Xia, W., Huang, J., Ni, B., Dong, J., Zhao, Y., & Yan, S. (2014). HCP: A Flexible CNN Framework for Multi-Label Image Classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38, pp. 1901-1907.

[Xiao et al., 2018] Xiao, A., Chen, R., Li D., Chen, Y., Wu, D. (2018). An Indoor Positioning System Based on Static Objects in Large Indoor Scenes by Using Smartphone Cameras. Sensors (Basel), 18(7), 2229.

[Xu et al., 2020] Xu, R., Zhang, C. L., Wang, P., Lee, J., Mitra, S., Chaterji, S., Li, Y., Bagchi, S. (2020). ApproxDet: content and contention-aware approximate object detection for mobiles. In Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20). Association for Computing Machinery, New York, NY, USA, pp. 449-462.

[Zaidi et al., 2022] Zaidi, S., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., Lee, B. (2022). A survey of modern deep learning based object detection models. Digital Signal Processing, 126, 103514.

[Zeiler and Fergus, 2014] Zeiler, M. D., Fergus, R. (2014). Visualizing and understanding convolutional networks. In European Conference on Computer Vision (ECCV 2014). Springer, Cham, pp.818-833.

[Zou et al., 2023] Zou Z., Chen K., Shi Z., Guo Y., Ye J. (2023). Object Detection in 20 Years: A Survey. In Proceedings of the IEEE,111(3), pp. 257-276.

Anexo A. Framework conceptual para crear Herramientas de co-diseño in-situ

En este anexo se describen los conceptos usados de base para el presente trabajo, sobre los cuales se busca brindar soporte al co-diseño in-situ, en particular, en espacios indoor. Estos conceptos fueron explorados originalmente en [Borrelli, 2021], otro trabajo del autor de esta tesis.

En este anexo se describe el framework conceptual presentado en [Borrelli, 2021], el cual permite analizar las características principales que deberían tener las herramientas de autor destinadas a la construcción in-situ de Aplicaciones Móviles basadas en Posicionamiento, que son un subconjunto de las sensibles al contexto. Cabe mencionar que la herramienta presentada en [Borrelli, 2021] como la herramienta de esta tesis fueron diseñadas usando de base el framework mencionado anteriormente.

En [Borrelli, 2021] se mencionan los siguientes conceptos relevantes al momento de diseñar herramientas de autor destinadas a crear in-situ Aplicaciones Móviles basadas en Posicionamiento:

- Considerar los mecanismos de sensado de posicionamiento y la representación del espacio como características desacopladas.
- Permitir la posibilidad de representar las características de cada dominio o tipo de aplicación, por ejemplo, usando *Plantillas* (Templates). Esto no solo permite extensibilidad sino que también facilita la definición de aplicaciones por parte de los usuarios que usan estas plantillas de guía para definir el contenido de las aplicaciones que crean.
- Contar con la posibilidad de tener diferentes perfiles del usuario participando en el uso de este tipo de herramientas.
- Permitir crear en simultáneo distintos tipos de aplicaciones, es decir, contar con distintos espacios de trabajo donde cada uno podría estar en etapas (o estados) diferentes del diseño de la aplicación.

En base a estos puntos se presentó en [Borrelli, 2021] un framework conceptual (ver Figura A.1) para facilitar la creación de nuevas herramientas de autor para co-diseñar in-situ Aplicaciones Móviles basadas en Posicionamiento.

Como se puede apreciar en la Figura A.1, el framework fue definido de manera abstracta para cubrir herramientas destinadas a la creación in-situ de diferentes tipos de aplicaciones móviles y para distintos dominios. En la Figura A.1 se observa cómo están desacoplados tanto los mecanismos de sensado como la representación del espacio físico; de esta manera, este punto de variabilidad puede ser extendido.

Además, en la Figura A.1 se puede observar el concepto de *Espacio de Trabajo,* el cual usa *Plantillas* que permiten el co-diseño de las aplicaciones. Este co-diseño puede involucrar diferentes etapas y esto es representado usando diferentes estados, donde en cada uno de ellos los distintos perfiles de usuarios pueden realizar diferentes acciones.

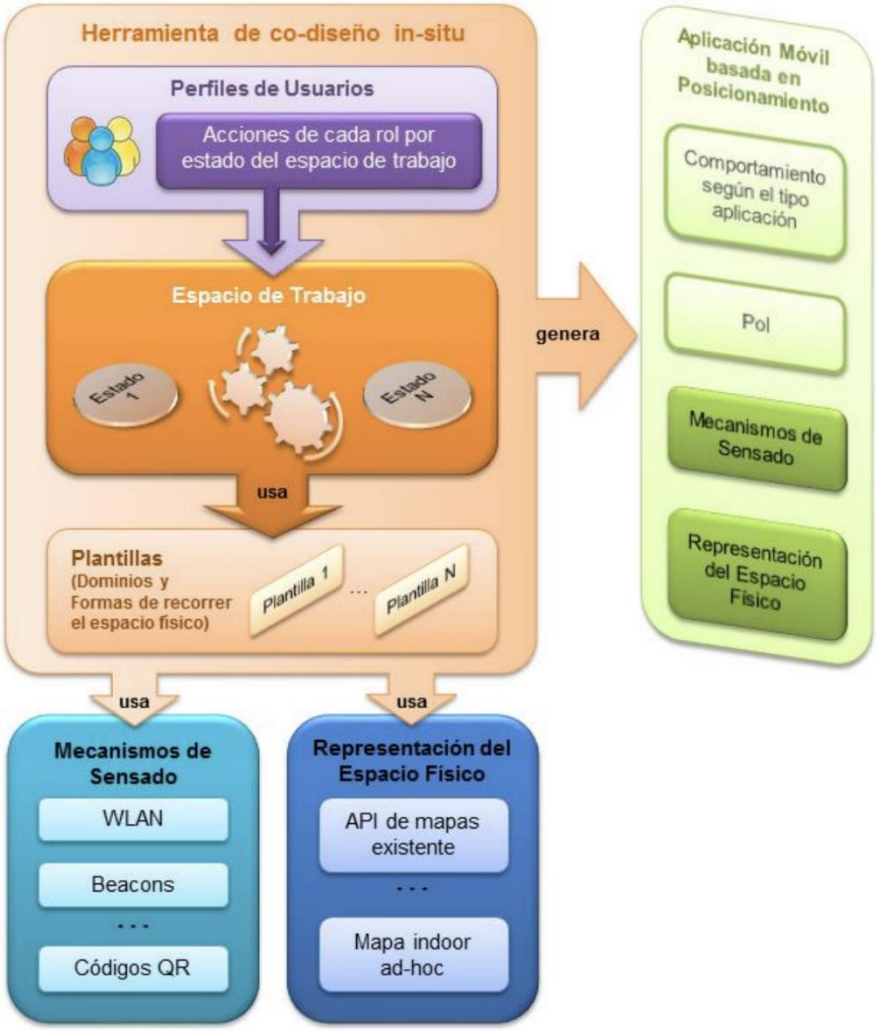


Figura A.1: Framework Conceptual para crear Herramientas de co-diseño in-situ de Aplicaciones Móviles basadas en Posicionamiento [Borrelli, 2021].

Anexo B. Herramienta de autor desarrollada para el co-diseño in-situ

El objetivo principal de este anexo es proporcionar una explicación detallada de las diversas funcionalidades asociadas a la herramienta de autor desarrollada para el co-diseño y co-testeo in-situ, la cual se presentó en el Capítulo 6 de este trabajo.

B.1 Registro e inicio de sesión

Para poder utilizar la aplicación es necesario contar con un usuario. En caso de que el usuario no haya completado un registro previo, se le ofrece la opción de registrarse como nuevo usuario, tal como se puede apreciar en la Figura B.1.b. También se puede apreciar en la Figura B.1.a la pantalla de inicio de sesión de la herramienta.

Es relevante destacar que toda la lógica de registro de usuarios y autenticación se gestiona de manera transparente gracias a la utilización de *Firebase*.

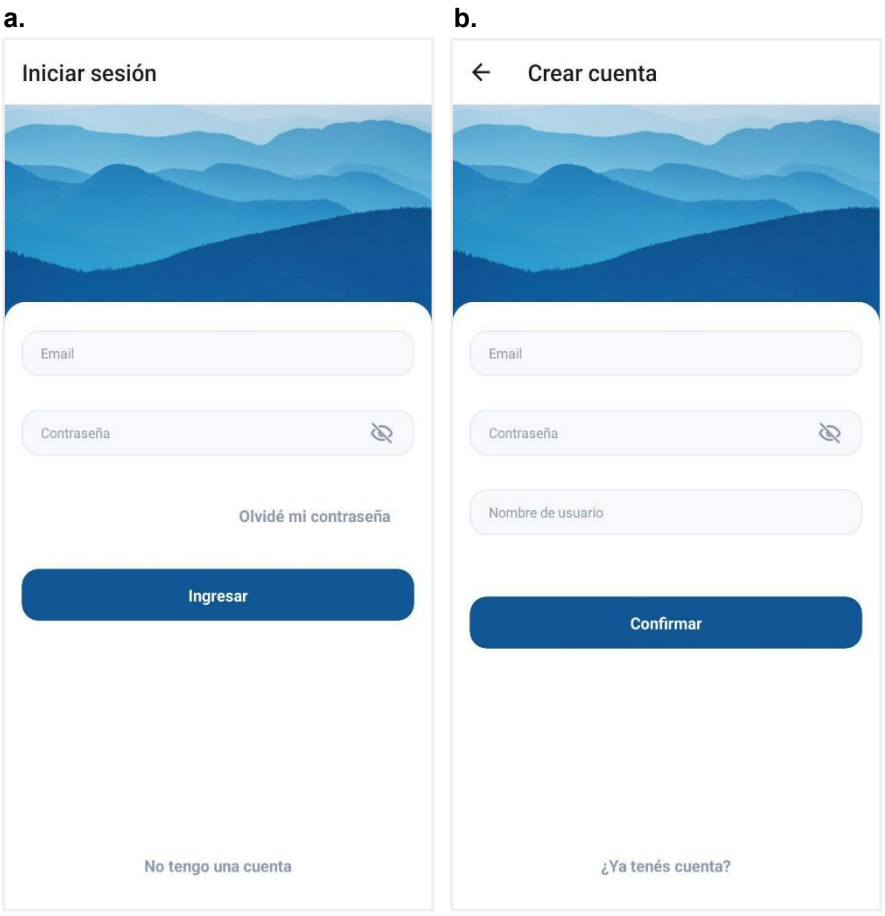


Figura B.1: Pantalla de autenticación de la herramienta.

Tras el registro e inicio de sesión en la herramienta, el usuario puede visualizar la pantalla que se observa en la Figura B.2; se puede apreciar que en la parte superior, se presenta el nombre del usuario y la opción para cerrar sesión. Justo debajo, se lista un menú con diversas opciones para interactuar con la herramienta.

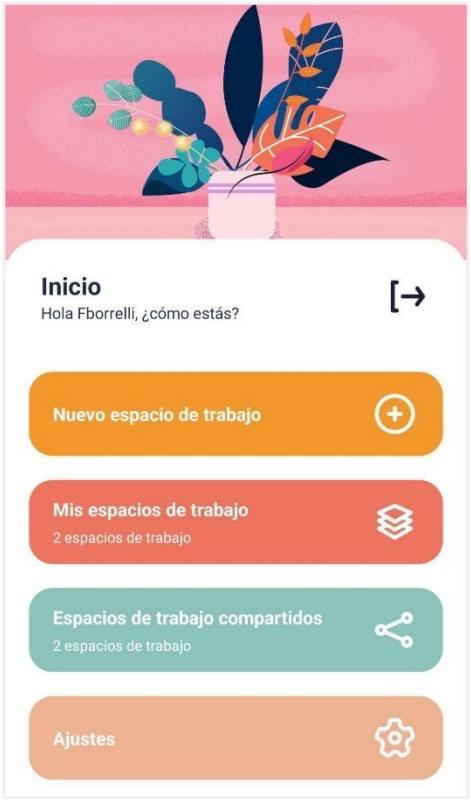


Figura B.2: Pantalla de inicio de la aplicación.

En la sección de ajustes de la Figura B.2, no solo se presenta información acerca de la versión actual de la aplicación, sino que también se brinda a los usuarios la capacidad de personalizar su experiencia mediante la realización de modificaciones en la interfaz. Entre las opciones disponibles se encuentra la posibilidad de alternar entre los modos claro y oscuro. Además, la pantalla de ajustes facilita la gestión de datos al ofrecer la opción de borrar datos en caché, lo cual puede ser útil en caso de encontrar errores inesperados.

A continuación se detallan las demás secciones de la Figura B.2, como son "*Nuevo espacio de trabajo*", "*Mis espacios de trabajo*" y "*Espacios de trabajo compartidos*".

B.2 Nuevo espacio de trabajo

Como su nombre indica, esta acción posibilita la creación de un nuevo *Espacio de trabajo*. El usuario que lleva a cabo el registro del *Espacio de Trabajo* adquiere el perfil de *Creador* de dicho espacio. Al hacer click en esta opción del menú, el usuario es dirigido a una pantalla que presenta un formulario, el cual incluye los campos necesarios que deben completarse para definir las características y detalles del nuevo espacio. En la Figura B.3, se puede observar detalladamente el formulario que el usuario debe completar.

Se pueden apreciar los siguientes campos en la Figura B.3:

- *Color:* al crear un *Espacio de Trabajo* debe definirse el color asociado; este color determinará cómo se visualizará la interfaz dentro del *Espacio de Trabajo*.
- *Nombre:* un nombre que se le otorgará a la aplicación a co-diseñar. Es un campo obligatorio.
- *Tipo de aplicación:* esta opción permite seleccionar el tipo de aplicación a co-diseñar siguiendo el concepto presentado de *Plantillas*. Si bien la herramienta sólo permite

utilizar la plantilla “*Información Posicionada*”, se espera que la herramienta permita en el futuro co-diseñar otro tipo de aplicaciones.

- *Edificio:* esta opción permite definir el edificio sobre el que se realizará el co-diseño in-situ. Como se puede observar en la Figura B.3, la información de la *Facultad de Informática* de la UNLP fue cargada en la herramienta.
- *Tipo de posicionamiento:* si bien en este trabajo se presenta una solución de posicionamiento por objetos, se espera que la herramienta admita otros tipos de posicionamientos en el futuro.
- *Descripción*: campo opcional que permite describir un poco sobre la aplicación a co-diseñar.

Figura B.3: Formulario de registro de un nuevo espacio de trabajo.

B.3 Mis espacios de trabajo

Después de crear un *Espacio de Trabajo*, este puede ser accedido a través de la sección "*Mis espacios de trabajo*", como se mostró en la Figura B.2. Al seleccionar esta opción, se muestra una pantalla como se puede observar en la Figura B.4, donde se enumeran todos los *Espacios de Trabajo* creados por el usuario. En esta pantalla, se proporcionan detalles de cada espacio, incluyendo su nombre, su descripción y el estado actual en el que se encuentra.

Al seleccionar el *Espacio de trabajo* listado en la Figura B.4, el usuario puede acceder al mismo. La vista inicial de este espacio se presenta en la Figura B.5, donde se presenta el mapa del edificio asociado (en este caso la Facultad de Informática de la UNLP), acompañado de una serie de botones en la parte superior derecha de la pantalla.

Figura B.4: Pantalla de los *Espacios de trabajo* creados.

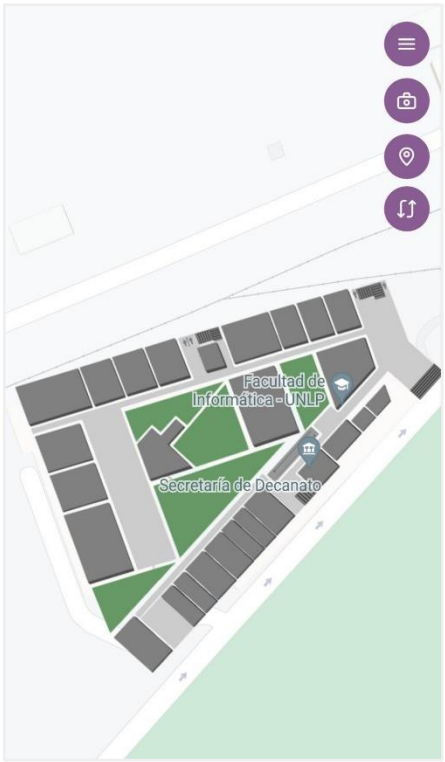


Figura B.5: Pantalla de mapa del espacio de trabajo.

Como se puede observar en la Figura B.5, en la parte superior derecha se muestran varios botones. A continuación, se explica brevemente el propósito de cada uno, detallando de arriba hacia abajo:

- *Menú*: el primero de estos botones permite abrir el menú del *Espacio de Trabajo.* Este menú permite acceder a todas las configuraciones e información del *Espacio de trabajo* actual. Más adelante se brindarán más detalles de esta pantalla.
- *Cámara*: el segundo botón en la Figura B.5 activa la cámara del teléfono para iniciar las detecciones. Durante los estados de co-diseño, se espera que el *Creador* y los *Colaboradores* seleccionen objetos en el espacio que sean relevantes y los registren con *Información Posicionada*. En el *estado de prueba y final*, los usuarios podrán utilizar esta opción para buscar los objetos registrados. Ambas acciones se describen en detalle en las Secciones 6.3.1 y 6.3.2 respectivamente.
- *Filtro de puntos de interés*: la tercera opción que se observa en la Figura B.5 actúa como un filtro, permitiendo mostrar en el mapa exclusivamente los marcadores creados por un usuario en particular. Esta función está disponible únicamente para el usuario *Creador* del *Espacio de trabajo*.
- *Cambio de piso*: finalmente, el cuarto botón de la Figura B.5 se emplea para realizar el cambio de piso. En la Figura B.6 se ilustra cómo se lleva a cabo este proceso dentro del edificio de la Facultad de Informática de la UNLP, que consta de tres pisos. Se observa en la Figura B.6.a que el mapa de fondo presenta el plano de la planta baja de la facultad. Al realizar el cambio al primer piso del edificio (Figura B.6.b), este plano se actualiza para mostrar la disposición del primer piso.

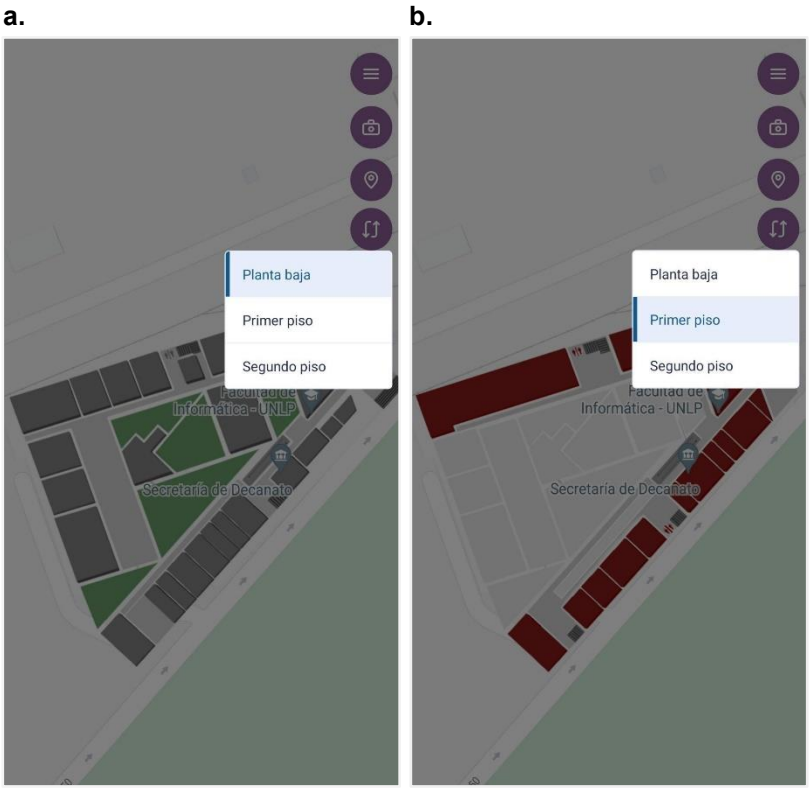


Figura B.6: Cambio de piso en la herramienta.

Cuando se accede al menú de un *Espacio de Trabajo* (de acuerdo con lo presentado en la Figura B.5 con el icono), el usuario es redireccionado a una pantalla como la que se muestra en la Figura B.7.

Figura B.7: Menú de un *Espacio de trabajo* con perfil de *Creador*.

De acuerdo a lo presentado en la Figura B.7 se detallará brevemente qué funcionalidades se encuentran asociadas a cada opción del menú:

- *Editar datos del espacio de trabajo*: al hacer click en esta opción, el usuario creador del espacio de trabajo es redirigido a un formulario similar al presentado en la Sección B.2. Desde este formulario, el usuario creador del *Espacio de Trabajo* podrá editar la información básica como el nombre y la descripción del mismo.
- *Posicionamiento por objetos:* desde esta pantalla, se pueden ajustar los parámetros relacionados a la solución de *posicionamiento por objetos* presentada en la Sección 5.4.1. En la Sección 6.3.1 se describe en detalle esta pantalla.
- *Usuarios:* desde esta pantalla se puede acceder a un listado de los usuarios *Colaboradores* y *Participantes* que tienen acceso al espacio de trabajo.
- *Compartir como colaborador*: al usar esta función, aparece en pantalla un código QR que otro usuario puede leer con su teléfono. Este código concede al nuevo usuario el acceso al *Espacio de Trabajo* con un perfil de *colaborador*. Es destacable que la aplicación permite definir un color distintivo para identificar al usuario dentro del *Espacio de Trabajo*. Además, brinda la opción de guardar el código QR en la galería del teléfono o compartirlo. En la Figura B.8 se observa un ejemplo de esto.

Figura B.8: Pantalla con código QR para unirse al *Espacio de trabajo* como *Colaborador*.

- *Cambiar de estado*: como se sugiere por su nombre, esta función permite al usuario con perfil de *Creador* modificar el estado actual del *Espacio de Trabajo*. En la Figura B.9 se visualizan los 5 estados definidos en la Sección 6.1 de este trabajo. Es importante destacar que, en algunos casos, cambiar de un estado a otro implica cumplir con ciertas condiciones. Por ejemplo, para pasar al *estado de prueba*, es necesario que haya al menos un objeto registrado en la base de datos contextual.

Figura B.9: Pantalla de cambio de estados.

- *Compartir como participante:* cabe mencionar también que cuando el *Espacio de trabajo* se encuentra en su estado final, se habilita una nueva opción en el menú para el usuario con perfil de *Creador*. Esta opción permite dar acceso a usuarios como participantes. En esta pantalla, se genera un código QR similar al presentado en la Figura B.8 para los colaboradores.

B.4 Espacios de trabajo compartidos

En esta sección, el usuario puede visualizar los *Espacios de trabajo* a los que tiene acceso, ya sea como *colaborador* o como *usuario final.* La disponibilidad de acceso depende del estado actual de cada *Espacio de Trabajo*, pudiendo haber casos en los que el usuario no pueda acceder. Los *Colaboradores* solo pueden acceder a los *Espacios de Trabajo* en estado colaborativo o de prueba, mientras que los usuarios finales sólo pueden acceder a aquellos en estado final.

La Figura B.10.a se puede observar el listado de los *Espacios de trabajo* en los que colabora el usuario. Además, en esta figura se destaca un botón con forma de cámara en la parte superior. Este botón permite unirse a un nuevo *Espacio de Trabajo*: al presionarlo, se activa la cámara del teléfono, permitiendo al usuario escanear un código QR (como el presentado en la Figura B.7) que facilitará su incorporación. En la Figura B.10.b se muestra el mensaje de confirmación que muestra la herramienta tras unirse al *Espacio de trabajo* como *Colaborador*.

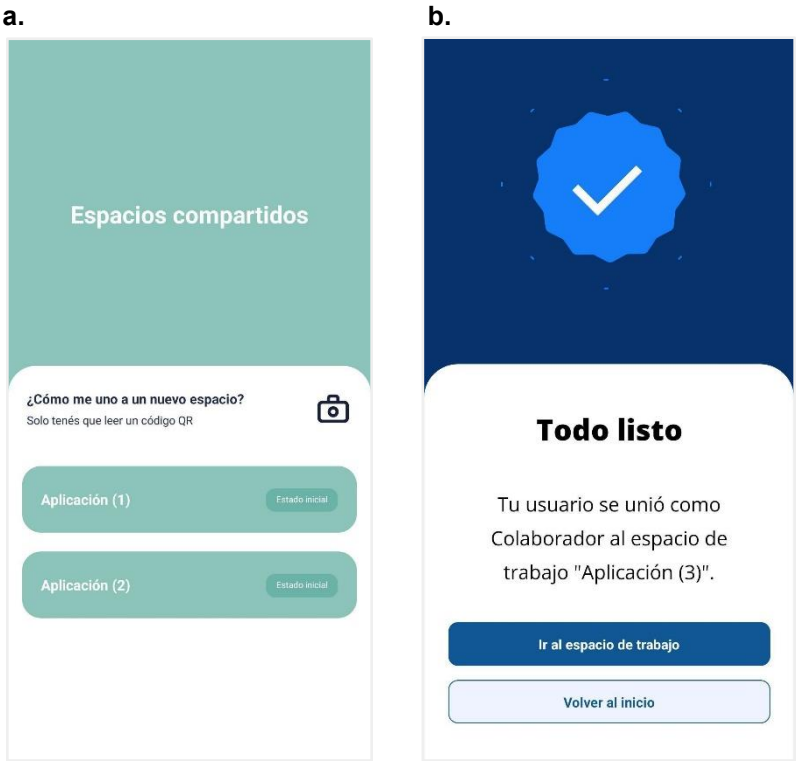


Figura B.10: Pantalla de visualización de los Espacios de trabajo compartidos.

Es importante destacar también que, al acceder como *Colaborador* a un *Espacio de trabajo*, las acciones que el usuario puede realizar se ven limitadas. Al entrar en el menú del *Espacio de Trabajo* (Figura B.11.a), se observa que las opciones de configuración de la aplicación y modelos no están disponibles. En su lugar, se habilita una opción para ver los detalles del

Espacio de Trabajo (Figura B.11.b), donde se puede visualizar la información relevante establecida por el creador.

Figura B.11: Vistas del *Espacio de trabajo* como *Colaborador*.

Anexo C. Métricas y gráficos disponibles en la herramienta

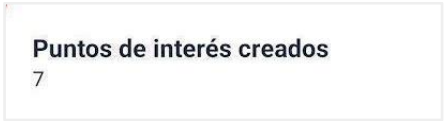
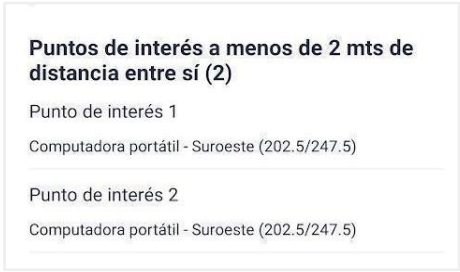
Este anexo brinda una explicación detallada de las diversas métricas y gráficos disponibles en la herramienta de autor presentada en el Capítulo 6 de este trabajo. Con la información mostrada en la herramienta se espera ayudar a los usuarios en el proceso de co-testeo.

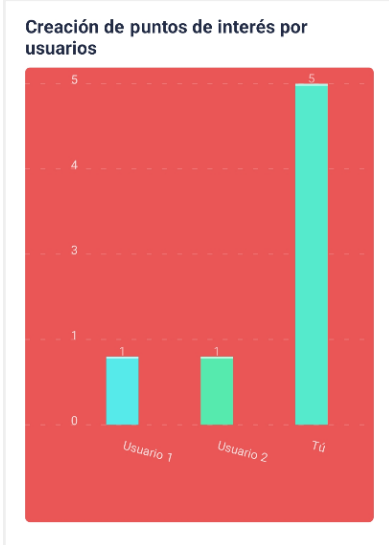
Esta información puede verse en detalle en la Tabla C.1 donde para cada métrica se describe:

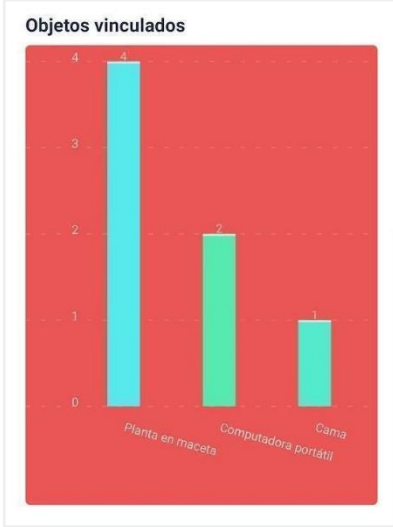
- *Pantalla:* Indica la pantalla de la herramienta correspondiente, seleccionada entre las 4 opciones presentadas en la Sección 6.3.5 ("*Puntos de interés*", "*Objetos inferidos*", "*Facilidad de reconocimiento*" o "*Posición para reconocimiento*").
- *Nombre de la métrica / gráfico*: Título asignado al gráfico o métrica dentro de la herramienta.
- *Descripción*: Proporciona una explicación detallada de la información presentada y cómo esta contribuye al co-testeo.
- *Representación en la herramienta*: Muestra una captura de pantalla que ejemplifica cómo se visualiza la información en la herramienta. Es importante mencionar que, debido a las limitaciones de tamaño de las pantallas de los dispositivos, se ha restringido el número de resultados que se muestran en algunos gráficos a un máximo de 7. Cuando hay más de 7 resultados, se ha implementado la opción de desplazarse hacia la izquierda en el gráfico para ver el resto de los resultados. Esta limitación se aplica específicamente a los gráficos de barras y de líneas.

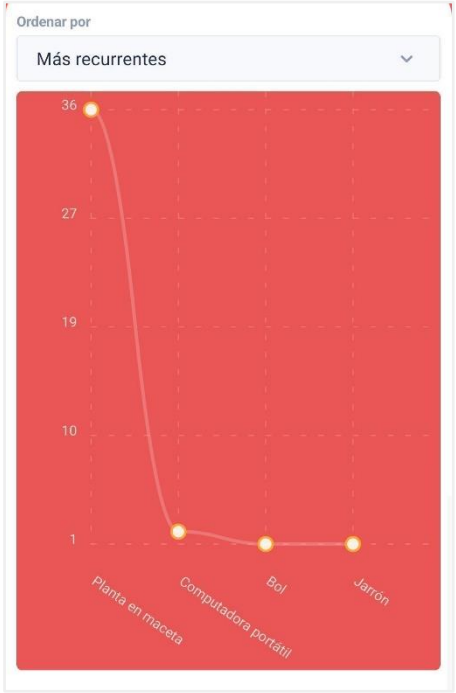
Adicionalmente, en la Tabla C.2 se proporciona una explicación detallada de aquellas métricas y gráficos que implican un procesamiento de datos más complejo.

Tabla C.1: Métricas y gráficos disponibles en la herramienta.

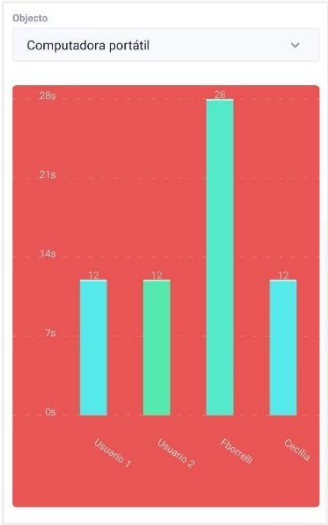
Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Puntos de interés	Puntos de interés creados	Se muestra la cantidad total de puntos de interés creados, lo que proporciona una visión general del nivel de actividad y la densidad de información dentro del *Espacio de Trabajo*.	
	Puntos de interés visibles	Se muestra la cantidad de puntos de interés que son visibles en la versión final del *Espacio de Trabajo*; lo que refleja las decisiones tomadas por el equipo de co-diseño respecto a qué información debe incluirse para los usuarios finales.	
	Puntos de interés a menos de X metros de distancia entre sí	Durante el proceso de co-diseño, es posible que se creen múltiples *puntos de interés* utilizando el mismo tipo de objeto y que estos estén ubicados a una distancia de menos de X metros entre sí. Esta situación podría llegar a generar ambigüedades al utilizar el *Posicionamiento por objetos*. El objetivo es mostrar los *puntos de interés* que cumplan estas condiciones, indicando su título, el objeto asociado y el ángulo de visión utilizado al momento de registrarse. El cálculo de la distancia entre cada par de puntos de interés se lleva a cabo mediante la comparación de las coordenadas geográficas (latitud y longitud), asociadas a cada objeto. Es importante destacar que el valor de X se determina en función del parámetro "*Máxima distancia en metros del objeto para considerarlo*", el cual se detalló en la Sección 6.3.1.	

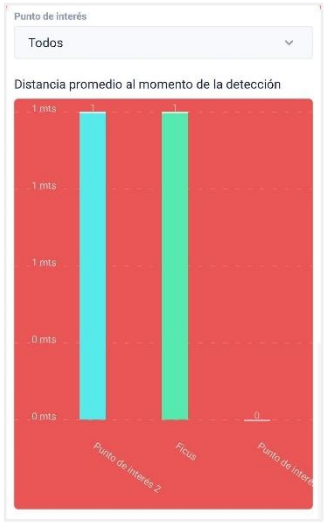
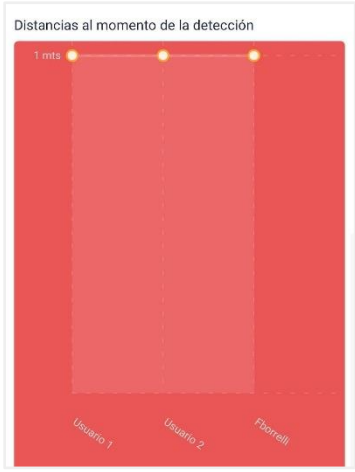
Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Puntos de interés	Orientación de los *puntos de interés*	Muestra cómo se distribuyen los *puntos de interés* en función de sus ángulos de visión registrados. Esta información se presenta mediante un gráfico de torta, lo que permite identificar fácilmente cuántos puntos de interés caen dentro de una misma categoría. La información provista permite realizar ajustes en la navegación del usuario, permitiendo mejorar el recorrido y la experiencia de los usuarios al utilizar la aplicación.	
	Creación de *puntos de interés* por usuarios	Indica la distribución de la creación de *puntos de interés* por usuario, lo que proporciona información sobre la contribución individual dentro del equipo de co-diseño. La información se presenta en un gráfico de barras, mostrando en orden descendente las contribuciones de cada usuario.	

Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Puntos de interés	Objetos vinculados	Muestra la asociación entre *puntos de interés* y sus objetos vinculados del entorno. Mediante un gráfico de barras se proporciona una representación visual de la frecuencia con la que cada objeto se utiliza en diferentes puntos de interés. En este gráfico, cada barra representa un objeto específico, y la altura de la barra indica la cantidad de veces que dicho objeto ha sido utilizado en los diferentes puntos de interés. Las barras están ordenadas de manera descendente, lo que significa que el objeto más repetido se encuentra en la parte superior y el menos repetido en la parte inferior. Esta información permite identificar patrones de uso, como la preferencia por ciertos objetos en ciertos contextos o la necesidad de mayor diversidad de objetos.	

Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Objetos inferidos	Objetos inferidos	Presenta los objetos más recurrentes al recorrer el entorno. Es decir, facilita la identificación de los objetos más (y menos) inferidos por el modelo de reconocimiento o detección configurado en el *Espacio de Trabajo*. Se implementa un gráfico de líneas; el cual presenta los resultados para cada uno de los objetos inferidos. Cada objeto está marcado en el gráfico por un punto interactivo, el cual puede ser clickeado para visualizar el valor exacto asociado. Los resultados se ordenan de forma descendente por defecto. Esto significa que los objetos más frecuentemente inferidos se ubican en la parte superior del gráfico, seguidos por los menos frecuentes hacia abajo. Además, se puede notar que se cuenta con una opción para cambiar el orden de la visualización, permitiendo al usuario alternar entre ver los objetos de forma ascendente (de menos a más frecuente) o descendente (de más a menos frecuente) según sus preferencias y necesidades. La información presentada resulta de utilidad, ya que permite detectar nuevos objetos no considerados previamente, los cuales podrían ser de interés para el *Espacio de Trabajo*. Además, posibilita la revisión y ajuste de objetos existentes, en base a la cantidad (alta o baja) de inferencias que reciben.	

Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Facilidad de reconocimiento	Facilidad de reconocimiento - General	Da la posibilidad de poder realizar un análisis de los resultados obtenidos al utilizar el *Posicionamiento por objetos*. Teniendo en cuenta el rango de cercanía establecido por el valor "*Máxima distancia en metros del objeto para considerarlo*" (explicado en detalle en la Sección 6.3.1), se evalúa el tiempo que tarda el usuario desde que entra en el radio del punto de interés hasta detectarlo. El gráfico de barra presentado muestra el tiempo medio para cada *punto de interés* del *Espacio de Trabajo*, junto con el título correspondiente a cada uno. Además, presenta dos selectores: el primero permite ordenar los resultados en orden ascendente o descendente según el tiempo que los usuarios tardaron y el segundo define la escala de visualización, permitiendo ver los resultados en términos de segundos o minutos.	

Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Facilidad de reconocimiento	Facilidad de reconocimiento - Detalle por punto de interés	Los usuarios cuentan con la opción de seleccionar un objeto específico, lo que permite visualizar todos los tiempos registrados para ese objeto por los distintos usuarios. Si se encuentran tiempos de detección significativamente largos para ciertos *puntos de interés*, esto podría indicar posibles problemas con las configuraciones del Posicionamiento por objetos que se tienen que ajustar. Además, puede influir en decisiones futuras sobre qué objetos o lugares son más adecuados para colocar puntos de interés en la aplicación.	

Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
Posición para reconocimiento	Posición para reconocimiento - General	Se muestra un gráfico de barras con la distancia media (en metros) a la que se encontraban los usuarios del *punto de interés* en el momento de su reconocimiento. La información presentada se obtiene a partir de la comparación entre las coordenadas geográficas (latitud y longitud) registradas en los "Eventos de detección" y en los puntos de interés. Esta comparación permite calcular la distancia a la que se encontraba cada usuario en el momento del reconocimiento. A partir de estos datos, se calculan medias generales. Al igual que en la métrica anterior, se realiza un cálculo para eliminar valores atípicos (*outliners*) y asegurar la precisión de los resultados.	
Posición para reconocimiento Posición para reconocimiento	Posición para reconocimiento - Detalle por punto de interés (distancia en metros)	Los usuarios cuentan con un selector que permite elegir un punto de interés específico. Al seleccionar esta opción, se puede ver información detallada del punto de interés, mostrando la distancia en metros en que se encontraba cada usuario al momento de realizar la detección. Esto permite examinar cómo cada usuario interactúa con ese punto de interés en particular.	

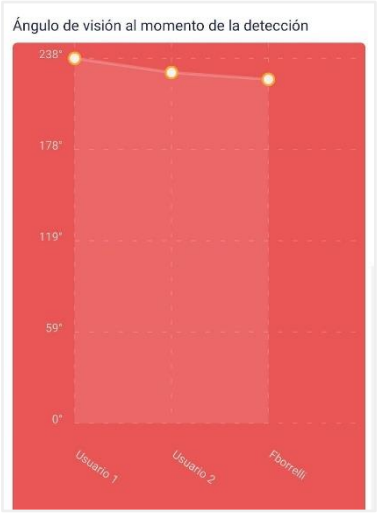
Sección	Nombre de la métrica / gráfico	Descripción	Representación en la herramienta
	Posición para reconocimiento - Detalle por punto de interés (diferencia en ángulo de visión)	Los usuarios cuentan con otra opción que les permite visualizar el ángulo de visión registrado por cada usuario al momento de realizar la detección. Esto permite ver y ajustar la forma en que los objetos del espacio fueron registrados.	

Tabla C.2: Detalles sobre el procesamiento de la información.

Sección	Nombre de la métrica / gráfico	Descripción
Puntos de interés	Orientación de los *puntos de interés*	El procedimiento para obtener esta información es simple. Primero, se determina la categoría a la que pertenece cada punto de interés. Se establecen ocho categorías principales: ● Noreste: 22.5° a 67.5° ● Este: 67.5° a 112.5° ● Sureste: 112.5° a 157.5° ● Sur: 157.5° a 202.5° ● Suroeste: 202.5° - 247.5° ● Oeste: 247.5° a 292.5° ● Noroeste: 292.5° a 337.5° Luego, se agrupan los resultados por categoría y se cuenta la cantidad de puntos de interés en cada una.
Objetos inferidos	Objetos inferidos	La información presentada en este gráfico se obtiene exclusivamente de los registros de "*Eventos de actualización*". Durante el procesamiento de estos registros, se realiza una verificación que tiene en cuenta las coordenadas geográficas asociadas a cada registro (es decir, la latitud y la longitud). El objetivo de esta verificación es eliminar posibles objetos duplicados que podrían aparecer en más de un registro. Esta medida garantiza una representación más precisa y coherente de los objetos inferidos en el entorno.
Facilidad de reconocimiento	Facilidad de reconocimiento - General	La información en esta sección se recopila procesando registros tanto de "*Eventos de detección*" como de "*Eventos de actualización*". Lo cual se hace de la siguiente manera: 1. Para calcular los tiempos medios, se comienza obteniendo los registros (tanto de detección como de actualización) asociados al *Espacio de Trabajo* actual, los cuales se agrupan por usuario. 2. Luego, es necesario llevar a cabo un proceso de filtrado sobre los registros de "*Eventos de actualización*" de cada usuario. Este proceso tiene como objetivo seleccionar únicamente los registros que sean de utilidad para determinar el momento en que un usuario entra dentro del radio de proximidad de un punto de interés (o *Información Posicionada*). En este filtrado, se compara la información almacenada en el *Espacio de Trabajo* con la de los registros de actualización, verificando que coincidan en el objeto asociado y que están geográficamente cerca (a una corta distancia en metros).

		3. A partir de la información obtenida en el paso anterior, ahora podemos saber el momento en que cada usuario entró al radio de proximidad de cada punto de interés. El paso siguiente será comparar ese momento con el momento de detección, obtenido con los registros de detección, pudiendo así determinar el tiempo total que el usuario tarda en reconocer el punto de interés. 4. Posteriormente, se calcula la media por objeto para todos los usuarios. Es importante destacar que, además, para realizar el cálculo se lleva a cabo una evaluación previa para identificar posibles valores atípicos (outliners) y eliminarlos, con el fin de evitar sesgos en los resultados.
Posición para reconocimiento	Posición para reconocimiento	La información presentada se obtiene a partir de la comparación entre las coordenadas geográficas (latitud y longitud) registradas en los "*Eventos de detección*" y en los puntos de interés. Esta comparación permite calcular la distancia a la que se encontraba cada usuario en el momento del reconocimiento. A partir de estos datos, se calculan medias generales. Además, se realiza un cálculo para eliminar valores atípicos (*outliners*) y asegurar la precisión de los resultados.

Anexo D. Detalles de las experiencias de co-diseño

El objetivo de este anexo es explicar con más detalle algunos aspectos técnicos, así como decisiones tomadas durante la realización de las distintas experiencias de co-diseño presentadas en el Capítulo 7 de este trabajo.

D.1 Aplicación para el congreso

En la Tabla D.1 se presenta el contenido utilizado para registrar la *Información Posicionada* en la aplicación del Congreso descripta en la Sección 7.1.1. Esta tabla fue provista a los participantes de la experiencia durante la etapa de co-diseño. La tabla incluye el nombre del contenido y su descripción. Además, indica qué Colaborador se debe encargar de registrar esta información. Por otro lado, en la Tabla D.2 se detallan las puertas a registrar, indicando el ángulo de visión a utilizar en cada *Espacio de Trabajo*.

Se describen en la Tabla D.3 las características de los dispositivos utilizados en las dos iteraciones de la experiencia del Congreso. La primera de estas iteraciones (Sección 7.1) utilizó la primera versión de la herramienta de autor, mientras que la otra (Sección 7.3.1) usó la segunda versión.

Tabla D.1: Contenido a registrar al co-diseñar la aplicación del Congreso.

Usuario	**Datos de la *Información Posicionada* a registrar**	
	Título	**Contenido**
Colaborador 1	Aula Logo - Simposio de Ingeniería de Software	El simposio de Ingeniería de Software se enfoca en los últimos avances y tendencias en el desarrollo de software. Se discutirán temas como metodologías ágiles, DevOps, arquitecturas modernas, testing automatizado, diseño de interfaces de usuario, gestión de proyectos de software, entre otros. Los participantes tendrán la oportunidad de compartir experiencias, investigaciones y casos de éxito en el campo de la ingeniería de software, así como explorar los desafíos y oportunidades emergentes en este ámbito.
Colaborador 2	AULA ALGOL - Simposio de Ciencia de Datos e Inteligencia Artificial	El simposio de Ciencia de Datos e Inteligencia Artificial se centra en el impacto de estas disciplinas en diversos sectores, incluyendo la industria, la salud, la educación y el gobierno. Se abordarán temas como aprendizaje automático, análisis de datos, procesamiento del lenguaje natural, visión por computadora, ética en la IA, entre otros. Los ponentes compartirán investigaciones punteras, aplicaciones prácticas y reflexiones sobre el futuro de la ciencia de datos y la inteligencia artificial, promoviendo un intercambio de conocimientos y experiencias entre académicos, profesionales y expertos en la materia.
Colaborador 2	AULA BASIC - Simposio de Ciberseguridad	El simposio de ciberseguridad se centra en los desafíos y soluciones relacionados con la protección de sistemas, redes y datos en un mundo digital cada vez más interconectado. Se discutirán temas como ataques cibernéticos, protección de la privacidad, seguridad en la nube, análisis forense digital, blockchain, entre otros. Los expertos compartirán investigaciones, buenas prácticas y estrategias para mitigar los riesgos de seguridad informática, así como reflexiones sobre la importancia de la ciberseguridad en la era digital actual. El simposio ofrecerá un espacio para el intercambio de conocimientos, la colaboración y la concienciación sobre la importancia de la seguridad cibernética en todos los ámbitos de la sociedad moderna.

Tabla D.2: Posiciones a registrar durante el co-diseño de la aplicación del congreso.

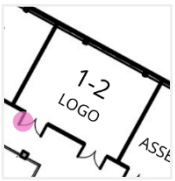
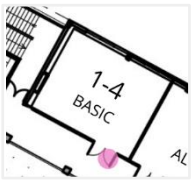
Información Posicionada	Espacio de Trabajo	Posición	Mapa
Aula Logo - Simposio de Ingeniería de Software	*Congreso - Versión 1 y 3*	De frente a la primera puerta de la izquierda del aula.	**Aula 1-2** 
	Congreso - Versión 2	Estando parado entre las aulas 1-2 y 1-3, se apunta a la primera puerta de la izquierda.	
AULA ALGOL - Simposio de Ciencia de datos e Inteligencia Artificial	*Congreso - Versión 1 y 3*	De frente a la puerta primera puerta de la derecha del aula.	**`Aula 1-3** 
	Congreso - Versión 2	Estando parado entre las aulas 1-2 y 1-3, se apunta a la primera puerta de la derecha del aula.	
AULA BASIC - Simposio de Ciberseguridad	*Congreso - Versión 1 y 3*	De frente a la puerta primera puerta de la derecha del aula.	**Aula 1-4** 
	Congreso - Versión 2	Estando parado del lado del aula 1-3, se apunta a la primera puerta de la derecha del aula.	

Tabla D.3: Detalles de los dispositivos utilizados.

	Colaborador 1	**Colaborador 2**
Modelo	Xiaomi MI 12	Samsung S22+
Sistema operativo	MIUI Global 14.0.8 (Android 13)	One UI 6.0 (Android 14)
Memoria	8gb	8gb
Procesador	3x2.50 GHz Cortex-A710, 1x3.00 GHz Cortex-X2, 4x1.80 GHz Cortex-A510, Adreno 730, Octa-core, Qualcomm SM8450 Snapdragon 8 Gen 1 (4 nm)	1x2.8 GHz Cortex-X2, 3x2.50 GHz Cortex-A710, 4x1.8 GHz Cortex-A510, Exynos 2200 (4 nm), Octa-core, Xclipse 920
Sensores	Acelerómetro, Brújula, Espectro de color, Giroscopio, Huella dactilar debajo de la pantalla, Huella dactilar óptica, Sensor Proximidad Virtual	Acelerómetro, Barómetro, Brújula, Comandos y dictados en lenguaje natural de Bixby, Giroscopio, Huella dactilar Sensor de proximidad, Soporte Banda Ultra Ancha (UWB)

D.2 Aplicación de Curiosidades de la Informática

En la Tabla D.4 se detalla el contenido utilizado para el co-diseño de la aplicación de *Curiosidades de la Informática*, descrita en la Sección 7.3.2. En esta tabla se incluye el nombre y la descripción del contenido a cargar en cada punto de interés, junto con la asociación al objeto correspondiente y la asignación del colaborador responsable de registrar la información.

Tabla D.4: Contenido a registrar al co-diseñar la aplicación de Curiosidades de la Informática.

Usuario	Datos de los *Puntos de interés* a crear		
	Objeto	**Título**	**Contenido**
Colaborador 2	Disco Duro	Dato #1	Aunque son pequeños, ¡los discos duros externos pueden guardar muchísima información! Algunos pueden almacenar hasta terabytes de datos, ¡lo que es como tener una enorme biblioteca en un bolsillo!
	Computadora portátil	Dato #2	Las notebooks son muy útiles para tomar notas en clase, pero también pueden ser usadas por los astronautas en el espacio para hacer experimentos y comunicarse con la Tierra.
Colaborador 3	Mouse	Dato #3	¿Sabías que el primer mouse fue inventado por Douglas Engelbart en 1964? Era muy diferente a los que usamos hoy en día, ¡se parecía más a una caja de madera con una rueda!
	Teclado	Dato #4	¿Notaste que las letras en el teclado no están en orden alfabético como en el abecedario? ¡Están organizadas de manera especial para que sea más fácil escribir rápido!
	VHS	Dato #5	Los cassettes VHS fueron una forma popular de grabar y ver películas y programas de televisión antes de que aparezca Netflix.

En la Tabla D.5 se detalla la información de los dispositivos que utilizaron los colaboradores que participaron en esta experiencia.

Tabla D.5: Detalles de los dispositivos utilizados.

	Colaborador 2	**Colaborador 3**
Modelo	Samsung S22+	Xiaomi Mi A3
Sistema operativo	One UI 6.0 (Android 14)	Android 9 Pie (One Edition)
Memoria	8gb	4gb
Procesador	1x2.8 GHz Cortex-X2, 3x2.50 GHz Cortex-A710, 4x1.8 GHz Cortex-A510, Exynos 2200 (4 nm), Octa-core, Xclipse 920	Qualcomm Snapdragon 665 GPU Adreno 610
Sensores	Acelerómetro, Barómetro, Brújula, Comandos y dictados en lenguaje natural de Bixby, Giroscopio, Huella dactilar, Sensor de proximidad, Soporte Banda Ultra Ancha (UWB)	Acelerómetro, Brújula, Espectro de color, Giroscopio, Huella dactilar debajo de la pantalla, Huella dactilar óptica, Sensor Proximidad Virtual