Desarrollo de un sistema de asistencia Android para personas con baja visión basado en la detección de poses con Smart Glasses.

Development of an Android assistance system for people with low vision based on pose detection with Smart Glasses.

Trabajo de Fin de Grado
Curso 2024–2025

Autor

Álvaro Velasco García: 9.5
Pedro Luis Cuadra Carrión: 9.5
Samuel Álvarez Medina: 9.5

Director

Joaquín Recas Piorno
Juan Bayon Fernandez

Grado en Ingenieria del Software
Facultad de Informática
Universidad Complutense de Madrid

Desarrollo de un sistema de asistencia Android para personas con baja visión basado en la detección de poses con Smart Glasses.

Development of an Android assistance system for people with low vision based on pose detection with Smart Glasses.

Trabajo de Fin de Grado en Ingeniería del Software
Departamento de Arquitectura de Computadores y Automática

Autor
Álvaro Velasco García: 9.5
Pedro Luis Cuadra Carrión: 9.5
Samuel Álvarez Medina: 9.5

Director
Joaquín Recas Piorno
Juan Bayon Fernandez

Convocatoria: *Junio* 2025

Grado en Ingenieria del Software
Facultad de Informática
Universidad Complutense de Madrid

13-06-2025

Dedicatoria

A nuestras queridas familias, especialmente a nuestros hermanos Nacho, Javier, Lucía y Alejandro, y a nuestros padres Pedro y Emilia, Carlos y María; y Ramón y Teresa

Agradecimientos

A nuestras familias, por apoyarnos siempre en todo momento. Gracias por todo.

A nuestros tutores Joaquín Recas Piorno y Juan Bayon Fernandez por darnos la oportunidad de desarrollar este proyecto y guiarnos en el proceso.

A Jonathan José Jiménez Jiménez, por su gran apoyo, orientación constante y generosa disposición durante cada etapa de este trabajo.

Resumen

Desarrollo de un sistema de asistencia Android para personas con baja visión basado en la detección de poses con Smart Glasses.

La baja visión es una condición visual que afecta en gran medida la calidad de vida de quienes sufren de esta, limitando su capacidad para realizar actividades cotidianas debido a la reducción de la agudeza visual y también de su campo de visión. Aunque existen ayudas tradicionales como lupas o gafas de aumento, estas soluciones suelen ser estáticas y no siempre se adaptan a las necesidades individuales de los pacientes. En este contexto, las tecnologías emergentes, como las smart glasses, ofrecen un enfoque innovador al combinar dispositivos portátiles con inteligencia artificial para proporcionar asistencia visual personalizada y en tiempo real.

En este trabajo presentamos el desarrollo de una aplicación Android para ser usada con *Smart Glasses*, diseñada para asistir a pacientes con baja visión mediante la descripción de información de personas a través de la detección de poses. La aplicación, desarrollada en Android Studio, utiliza un modelo de inteligencia artificial entrenado por nosotros específicamente para reconocer y analizar posturas humanas. Este enfoque permite a los usuarios recibir descripciones auditivas y visuales de las personas en su entorno, mejorando su capacidad para interactuar socialmente y su movilidad en entornos públicos.

La integración de la aplicación con nuestras smart glasses ofrece ventajas significativas, como la portabilidad y la discreción, aspectos clave para la aceptación de estos dispositivos por parte de los usuarios. Además, el uso de un modelo de Inteligencia Artificial personalizado asegura que la detección de poses sea precisa y que sea capaz de adaptarse a las necesidades específicas de cada persona. El sistema está diseñado con un enfoque escalable, permitiendo la incorporación de nuevas poses y funcionalidades de manera sencilla, lo que facilita su adaptación a diferentes escenarios y necesidades.

En conclusión, este proyecto demuestra cómo la combinación de smart glasses, aplicaciones móviles y modelos de inteligencia artificial puede ofrecer soluciones efectivas y personalizadas para personas con baja visión.

Palabras clave

Android, Baja Visión, Smart Glasses, Inteligencia Artificial

Abstract

Development of an Android assistance system for people with low vision based on pose detection with Smart Glasses.

Low vision is a visual condition that greatly affects the quality of life of those who suffer from it, limiting their ability to perform everyday activities due to the reduction of visual acuity and also of their field of vision. Although traditional aids such as magnifying glasses or loupes are available, these solutions are often static and not always adapted to the individual needs of patients. In this context, emerging technologies such as smart glasses offer an innovative approach by combining wearable devices with artificial intelligence to provide personalised, real-time visual assistance.

In this paper we present the development of an Android application for use with smart glasses, designed to assist patients with low vision by describing information about people through pose detection. The application, developed in Android Studio, uses an artificial intelligence model trained by us specifically to recognise and analyse human poses. This approach allows users to receive auditory and visual descriptions of people in their environment, improving their ability to interact socially and their mobility in public settings.

The integration of the app with our smart glasses offers significant advantages, such as portability and discretion, which are key to the acceptance of these devices by users. In addition, the use of a customised Artificial Intelligence model ensures that pose detection is accurate and able to adapt to the specific needs of each individual. The system is designed with a scalable approach, allowing the incorporation of new poses and functionalities in a simple way, which facilitates its adaptation to different scenarios and needs.

In conclusion, this project demonstrates how the combination of smart glasses, mobile applications and artificial intelligence models can offer effective and personalised solutions for people with low vision.

Keywords

Android, Low Vision, Smart Glasses, Artificial Intelligence

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La baja visión es una condición que limita la capacidad visual de forma irreversible, afectando actividades cotidianas como reconocer expresiones faciales o posturas de otras personas. Según la OMS (OMS, 2019), más de 2.200 millones de personas en todo el planeta sufren algún tipo de discapacidad visual, siendo la baja visión una de las más discapacitantes en entornos sociales.

Este proyecto desarrolla una aplicación Android para smart glasses que ayuda a usuarios con baja visión a interpretar las posturas de las personas que les rodean. La solución utiliza inteligencia artificial para detectar poses básicas como "brazos cruzados ", "saludo mano derecha" o "señalando a la izquierda ", proporcionando retroalimentación auditiva y visual al usuario.

El sistema utiliza modelos de visión por computadora optimizados de ML Kit, específicamente diseñados para funcionar eficientemente en dispositivos wearables con recursos limitados. A través de Pose Detection de ML Kit, la aplicación identifica con precisión los puntos clave del cuerpo humano (como hombros, codos y caderas) en tiempo real, incluso en dispositivos de gama media. El modelo IA que hemos desarrollado es capaz de interpretar estos datos, con los que posteriormente ofreceremos descripciones auditivas y visuales simplificadas. Este enfoque permite a los usuarios con baja visión interpretar posturas y gestos de su entorno de forma inmediata, reduciendo su dependencia de la visión residual y mejorando su interacción social.

Una innovación clave es el procesamiento local de los datos, garantizando privacidad y rapidez al evitar la dependencia de servidores externos. La aplicación también incluye funcionalidades de accesibibilidad, entre las que se encuentran cambiar el modo de color del sistema, el tamaño de letra y el idioma.

Este trabajo representa un primer paso hacia sistemas de asistencia visual más avanzados. Futuras versiones podrían integrar reconocimiento facial, navegación ambiental o interacción por voz. El objetivo final es facilitar el día a día de las personas con baja visión mediante tecnología accesible y discreta.

El proyecto combina técnicas de machine learning, desarrollo móvil y experiencia de usuario accesible. Los resultados podrían beneficiar no solo a pacientes, sino también a terapeutas y desarrolladores de tecnologías asistivas.

1.1. Motivación

Las tecnologías de asistencia basadas en visión artificial están cambiando en los últimos años la forma en que las personas que sufren de algún tipo de discapacidad visual interactúan con su entorno. En este contexto, los dispositivos wearables, especialmente las smart glasses, han emergido como una solución prometedora gracias a su capacidad para integrar inteligencia artificial de forma discreta y portátil. Un hito reciente en este campo es el lanzamiento de dispositivos como las Ray-Ban Meta (Meta, 2023).

Si tenemos en cuenta datos oficiales de la Organización Mundial de la Salud (OMS, 2023), son prácticamente 1.300 millones de personas las que padecen algún tipo de discapacidad visual, de las cuales 260 millones presentan baja visión. Estas cifras siguen una tendencia ascendente, impulsada principalmente por el envejecimiento poblacional y el aumento de patologías oculares crónicas. Estudios recientes demuestran que las personas con baja visión enfrentan mayores riesgos de aislamiento social y dificultades laborales, con un impacto económico importante: En España el coste económico acumulado, para el periodo 2021 a 2030, que está asociado a 5 patologías evaluadas (entre ellas el glaucoma, la degeneración macular y la miopía magna) alcanzará cerca de 100 mil millones de euros al final del periodo (Fundación Porib, 2025).

Nuestro objetivo en este proyecto es mejorar la calidad de vida de las personas con baja visión todo lo que podamos,por lo que investigaremos y explicaremos posteriormente en qué consiste la baja visión propiamente dicha y cuáles son algunas de las principales patologías que la provocan.

En el presente trabajo pretendemos investigar soluciones que mejoren la calidad de vida de las personas con baja visión empleando gafas inteligentes. Realizaremos una investigación integral, primero conociendo e indagando en estas enfermedades, analizando los distintos perfiles de los pacientes con sus principales limitaciones visuales, y estudiaremos algunas soluciones actuales basadas en Realidad Aumentada. Acto seguido, investigaremos acerca del hardware y software que mejor se adapten a dicha solución y desarrollaremos una ayuda tanto visual como auditiva.

Nuestro trabajo surge como respuesta a estas necesidades, con el objetivo de desarrollar un sistema innovador que combine:

- Smart glasses como soporte visual integrado y discreto
- Técnicas avanzadas de detección de poses mediante inteligencia artificial
- Interfaz accesible e intuitiva para el uso de todos los usuarios

A diferencia de sistemas como OrCam MyEye (Orcam, 2024) centrados en lectura de texto, nuestra solución se especializará en interpretar lenguaje corporal, abordando una necesidad poco cubierta en soluciones actuales.

1.2. Objetivos

Este proyecto se plantea con una doble finalidad: por un lado, desarrollar una solución tecnológica innovadora para personas con baja visión, y por otro, sentar las bases para futuras mejoras e investigaciones en este campo. Se podría decir que nuestro objetivo principal es desarrollar un sistema integrado en smart glasses que, mediante técnicas de inteligencia artificial y detección de poses, proporcione asistencia visual y auditiva a personas con baja visión, mejorando su autonomía y facilitando su día a día.

A continuación se muestra el enlace al repositorio con el código completo del proyecto, donde está disponible toda la implementación y la documentación adicional:

`https://github.com/alvave03/TFG-2024-25`

Si nos referimos a objetivos más específicos, podríamos diferenciarlos en los siguientes:

1. Implementar nuestro propio modelo de detección de poses eficiente utilizando Google ML Kit, capaz de identificar posturas humanas en tiempo real.

2. Diseñar una interfaz lo más intuitiva y accesible posible en una aplicación Android que permita la interacción con las smart glasses.

3. Crear un sistema que, a través de las smart glasses, describa con voz y visualmente las posturas de las personas alrededor del usuario.

4. Integrar la aplicación Android con las smart glasses para garantizar una comunicación fluida y estable entre ambos dispositivos, permitiendo el envío de información en tiempo real sobre las poses detectadas.

5. Garantizar la escalabilidad del sistema para permitir la incorporación futura de nuevas poses y funcionalidades.

6. Evaluar la usabilidad y efectividad del sistema mediante pruebas en tiempo real.

7. Desarrollar e implementar la aplicación Android que contenga toda la funcionalidades descritas anteriormente.

El cumplimiento de estos objetivos permitirá crear una herramienta práctica que combine las posibilidades que nos brindan los dispositivos wearables y la inteligencia artificial para abordar un problema real, con capacidad de evolucionar hacia soluciones más completas en el futuro.

1.3. Plan de trabajo

Para garantizar el óptimo desarrollo de este proyecto, dividiremos el trabajo en varias fases con actividades específicas para cada una de ellas.

- **Investigación y Análisis:** En esta fase inicial realizaremos un estudio de las necesidades de las personas con baja visión, analizando sus limitaciones y los escenarios donde el sistema podría ser más útil. Además, estudiaremos las diferentes tecnologías disponibles para la detección de poses, comparando sus ventajas y limitaciones. Este análisis permitirá definir los requisitos técnicos y funcionales para el desarrollo posterior.

- **Desarrollo del Modelo de IA:** La parte más relevante del proyecto consiste en crear un modelo eficiente de detección de poses. Para ello, nos encargaremos de recopilar un dataset o conjunto de datos representativo, que utilizaremos para entrenar nuestro modelo y llevaremos a cabo varias implementaciones distintas para elegir la que obtenga mejores resultados.

- **Desarrollo de la Aplicación:** Con el modelo de IA ya listo, procederemos a construir la aplicación Android que hará uso de él. El desarrollo incluirá una interfaz que sea muy intuitiva, accesible y además fácil de usar. Implementaremos el sistema de ayuda visual en la cámara de las gafas y el de ayuda auditiva con sus altavoces. Un aspecto clave será la integración con las Smart Glasses, teniendo que establecer una comunicación estable para mostrar la información procesada.
- **Integración y Pruebas:** En esta fase unificaremos todos los componentes del sistema, verificando su correcto funcionamiento conjunto mediante pruebas técnicas y de usabilidad. Evaluaremos la precisión en la detección de poses, la claridad de la retroalimentación y la experiencia de usuario real, implementando los ajustes necesarios para optimizar el rendimiento global de nuestra solución.

Además, usaremos una metodología ágil con ciclos cortos de desarrollo (2-3 semanas) donde iremos: 1) construyendo funciones paso a paso, 2) probándolas con usuarios, y 3) haciendo mejoras continuas. Esto nos ayudará a crear una herramienta que realmente sirva a quienes tienen problemas de visión y a coordinarnos eficazmente.

1.4. Estructura del documento

Vamos a estructurar este documento siguiendo el siguiente esquema:

En el Capítulo 2, "Estado de la Cuestión", se explora cómo la inteligencia artificial, específicamente la detección de poses (Pose Detection), puede mejorar la calidad de vida de personas que sufren de baja visión. Se analizan las patologías causantes (como cataratas, DMAE y glaucoma), las limitaciones de las ayudas tradicionales y el potencial de las Smart Glasses como dispositivos de asistencia. Además, se detallan tecnologías clave como IoT, Google ML Kit y MediaPipe para implementar sistemas inteligentes, junto con técnicas de IA como redes neuronales para entrenar modelos personalizados.

En el Capítulo 3, "Planificación y Recursos", mostramos la planificación del proyecto, describimos todos los recursos que hemos escogido de hardware con la justificación de esta elección, los SDKs que hemos empleado, las librerías usadas para la integración de la inteligencia artificial en las diversas aplicaciones Android y los entornos de desarrollo que vamos a utilizar en el desarrollo de este proyecto.

En el Capítulo 4, "Desarrollo del Proyecto y resultados", detallamos el desarrollo completo del proyecto, explicando cómo se implementan detalladamente cada una de las funcionalidades y su integración con Android Studio y las demás herramientas y software que hemos elegido, así como los resultados del mismo.

En el Capítulo 5, "Conclusiones y trabajo futuro", expondremos las conclusiones del estudio y enumeraremos las posibles líneas de trabajo futuro.

Capítulo 2

Estado de la Cuestión

En este capítulo, vamos a explorar cómo las tecnologías de inteligencia artificial (**IA**) pueden ser utilizadas para mejorar la calidad de vida de las personas con baja visión. Nos centraremos en la detección de poses (**Pose Detection**), una técnica que permite identificar la posición de una persona a partir de puntos clave como articulaciones, y su aplicación en sistemas de asistencia que faciliten tareas cotidianas. Analizaremos frameworks como Google ML Kit y MediaPipe, que ofrecen soluciones para implementar esta tecnología de manera eficiente, así como la integración de dispositivos IoT y técnicas de realidad aumentada para crear sistemas inteligentes y accesibles.

Está organizado siguiendo este esquema: 1) primero, describimos las principales patologías que son causantes de los problemas de baja visión y cómo afectan en función de la edad; 2) mostramos las principales ayudas tradicionales que es estaban empleando hasta la fecha para solventar los problemas relacionados con la baja visión; 3) se describe el uso y funcionamiento de Smart Glasses, así como sus aplicaciones prácticas; 4) detallamos las tecnologías que vamos a utilizar en el sistema IoT; 5) exponemos las técnicas que se utilizarán en este proyecto haciendo uso de inteligencia artificial para entrenar nuestros modelos y compararemos los resultados que vamos obteniendo; 6) y por último exponemos las conclusiones.

2.1. Baja visión

La **baja visión** es una condición visual en la que la persona experimenta una pérdida significativa de su campo de visión y su agudeza visual, que no no se ha conseguido corregir en su totalidad ni con gafas convencionales, cirugía, lentes de contacto o distintos tipos de medicamento. Afecta la capacidad para realizar actividades cotidianas como leer, escribir, reconocer rostros o desplazarse con facilidad. Aunque la persona conserva cierto grado de visión, esta es insuficiente para desenvolverse con normalidad sin ayudas visuales o adaptaciones específicas.

Debemos tener en cuenta que la baja visión no es una patología en sí, es el resultado de una de ellas. Esto es lo que hace que las personas que la sufren tengas dificultades a la hora de llevar a cabo su día a día de manera normal. Debido a esto, no debemos centrarnos solo en buscar una solución para la baja visión como tal, sino que también tendremos que estudiar las causas que la hacen aparecer.

Podemos considerar que una persona sufre de baja visión cuando tiene un campo visual que no supera el 20 % o una agudeza visual menor de 0,3 en el mejor ojo incluso tras

cualquier corrección óptica previa (Alcocer Óptica, 2024).

2.1.1. Patologías causantes

La baja visión puede estar causada por patologías muy distintas, entre las que se encuentran:

2.1.1.1. Cataratas

Las cataratas son una afección ocular que se caracteriza por la opacidad del cristalino, que es la la lente natural del ojo (que suele ser transparente) y permite el paso de la luz para poder enfocar las distintas imágenes en la retina. Esta opacidad provoca una visión borrosa, dificultad para ver en situaciones en las que haya poca luz, deslumbramiento con la luz brillante, pérdida de intensidad en los colores y, en etapas avanzadas, una disminución significativa de la visión.

Las cataratas se suelen desarrollar muy lentamente y están asociadas principalmente al envejecimiento (cataratas seniles), aunque también pueden ser causadas por traumatismos oculares, enfermedades como la diabetes, exposición muy prolongada a la radiación ultravioleta y también sl uso de algunos medicamentos específicos (como pueden ser los corticosteroides) u otros factores congénitos. Con la Figura 2.1 nos podemos hacer una idea de esta condición.

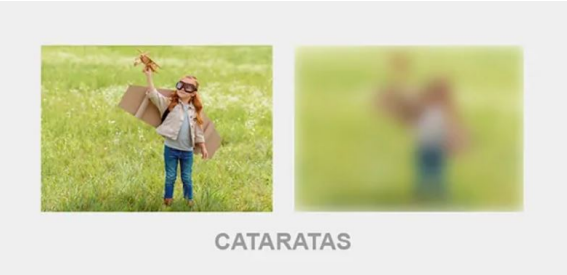

Figura 2.1: Visión de una persona con cataratas (Barañano Baja Visión, 2025).

2.1.1.2. Degeneración macular asociada a la edad (DMAE)

La DMAE o **degeneración macular** asociada a la edad, es una enfermedad del ojo que ocurre por el detrioro, daños, o la degeneración de la mácula, que es una capa amarillenta de tejido que se encuentra en la parte posterior del ojo (en el centro de la retina) y es sensible a la luz. Esta zona es la que proporciona la agudeza visual que nos permite percibir detalles muy pequeños y difíciles de apreciar. Cuando la mácula no funciona como debería, empezamos a perder nitidez en las áreas centrales de nuestro campo visual. En la Figura 2.2 que se muestra a continuación vemos un ejemplo.

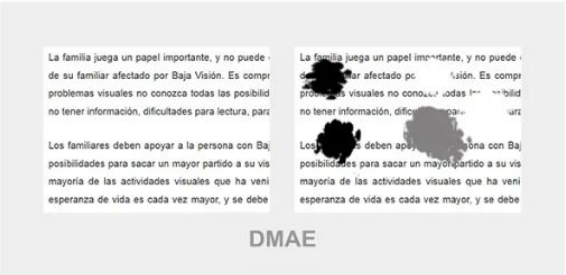

Figura 2.2: Visión de una persona con DMAE (Barañano Baja Visión, 2025).

2.1.1.3. Retinosis Pigmentaria

La retinosis pigmentaria es una enfermedad del ojo hereditaria y degenerativa que ocasiona una notable disminución de la capacidad periférica, y que puede llevar a la ceguera en muchas ocasiones. Aunque muchas personas ya nacen con esta enfermedad, es bastante extraño que se manifieste antes de la adolescencia. Además las personas que la padecen no suelen ser conscientes de ello hasta que ya se encuentra en fases avanzadas.

Es importante tener en cuenta que no todas las retinosis pigmentarias conducen a la misma pérdida de visión, no son iguales. Esta enfermedad se transmite principalmente por personas que portan ya los genes defectuosos y que actúan como transmisores de esta patología. En la Figura 2.3 se muestra un ejemplo de lo que ven las personas que la padecen.

Figura 2.3: Visión de una persona con retinosis pigmentaria (Barañano Baja Visión, 2025).

2.1.1.4. Glaucoma

El glaucoma es una enfermedad del ojo que daña el nervio óptico, generalmente debido a un aumento de la presión dentro del mismo. Este daño suele ser progresivo y puede llevar a la pérdida de visión periférica. Además, en etapas avanzadas puede conducir incluso a la ceguera de manera irreversible si no es tratado a tiempo.

Existen varios tipos de glaucoma, el más común de ellos es el de ángulo abierto. Este avanza lentamente y sin síntomas evidentes en sus primeras etapas, mientras que el de ángulo cerrado por ejemplo, se presenta de forma aguda con síntomas de todo tipo como dolor ocular intenso, enrojecimiento del ojo, naúseas o visión borrosa.

Las personas que cuentan con un historial de glaucoma en su familia tienen más posibilidades de desarrollarlo. Es recomendable para todas las personas mayores de 25 años hacerse controles de glaucoma, aumentando el número a lo largo de los años, ya que otro dato a destacar es que la mitad de las personas que tiene glaucoma ni lo saben. Tenemos un ejemplo en la Figura 2.4 que se muestra a continuación:

Figura 2.4: Visión de una persona con glaucoma (Barañano Baja Visión, 2025).

2.1.2. Baja Visión en Grupos de Edad

Debemos saber que la baja visión afecta de forma distinta a las personas. Dependiendo de la edad, el entorno y las circunstancias individuales, las necesidades y los desafíos pueden variar significativamente. Vamos a analizar a continuación tres grupos específicos de la población:

2.1.2.1. Niños/as y Adolescentes

La pérdida de la visión en los niños afecta principalmente en el ámbito social. Puede tener un efecto perjudicial en la construcción de relaciones con los amigos y la sociedad. Es importante el apoyo y ayuda de la familia para conseguir que sean capaces de llevar a cabo su día a día de manera normal (Mácula Retina, Dra. Shahid, 2012).

En el ámbito educativo, es fundamental contar con adaptaciones como material didáctico ampliado, tecnología asistida (como tablets con zoom o lectores de pantalla) y el apoyo de profesores especializados. Aportando los tipos de apoyo adecuados y buenas intervenciones, las instituciones educativas pueden garantizar que la educación no se vea limitada por los problemas de visión.

2.1.2.2. Adultos y Personas de mediana edad

Los adultos con pérdida de visión no se enfrentan a los mismos desafíos que los niños, sino a otros factores más relacionados con el ámbito social en el trabajo y económico. Una persona con visión normal suele encontrar trabajo con mucha más facilidad que las que sufren de alguna discapacidad visual.

Las personas con ceguera o algún tipo de discapacidad visual grave también suelen contar con menos años de educación formal. Esto hace que tiendan a tener ingresos más bajos y una mayor dependencia de programas como los de asistencia pública.

Muchos adultos en esta etapa se ven obligados a adaptar sus carreras o buscar nuevas formas de empleo que se ajusten a sus limitaciones visuales.

2.1.2.3. Ancianos y Personas mayores

En las personas mayores, la baja visión es más común y está frecuentemente relacionada con el envejecimiento. Suelen ser muy habituales enfermedades como la degeneración macular, de la cual hemos hablado anteriormente. Para este grupo, la pérdida de visión suele ir acompañada de otros problemas de salud, como la artritis o la diabetes, lo que complica su manejo.

La baja visión en personas mayores afecta gravemente su calidad de vida, limitando

su autonomía para realizar actividades cotidianas como cocinar, leer o desplazarse de manera segura. Esto puede llevar al aislamiento social y a un mayor riesgo de depresión. El apoyo familiar y comunitario es muy importante para ayudar a las personas mayores a ser independientes y a mantener su bienestar emocional.

2.2. Ayudas visuales tradicionales no tecnológicas

En las ayudas visuales tradicionales podemos destacar inventos como el telemicroscopio (Salud Vision, 2023), los prismas de fresnel peli o los telescopios TS (tuOptometrista, 2017). En la Figura 2.5 podemos observar una imagen de uno de estos ejemplos.

Estos dispositivos tradicionales presentan varias desventajas, como la reducción de la agudeza visual en las personas que los utilicen con frecuencia o incluso la distorsión de las imágenes percibidas. Además, carecen de la capacidad de ajustarse en tiempo real, lo que limita opciones como el zoom, el brillo o el contraste. A esto se suma su diseño poco atractivo, que puede generar rechazo social y llevar a que muchos usuarios decidan no utilizarlos. Es por ello que con el avance de las nuevas tecnologías, muchos de estos aparatos se están quedando un tanto obsoletos. En la Figura 2.5 podemos observar un ejemplo de telemicroscopio.

Figura 2.5: Imagen de un telemicroscopio (Salud Vision, 2023).

2.3. Smart Glasses

Las **Smart Glasses** son dispositivos que incorporan una tecnología en forma de pantalla, que puede proyectarse o reflejarse en las mismas lentes o también puede ser un componente separado, y es capaz de mostrarle información al usuario sin provocar distracciones cuando no haya necesidad.

Las gafas inteligentes o Smart Glasses cada vez están adquiriendo más relevancia en el panorama global. Uno de los momentos claves fue en 2012 cuando Google anunció su desarrollo de las Google Glass o también cuando aparecieron las primeras Microsoft Hololens (Microsoft, 2025).

No obstante, todavía es un producto que todavía tiene que evolucionar y desarrollarse más, pero observando el potencial que tienen estos dispositivos es innegable que en dentro de poco y cada vez más se utilizarán en aplicaciones o incluso en el día a día, ya que es muy probable que de aquí a unos años su uso esté muy normalizado. Podemos ver un ejemplo de estos dispositivos en la Figura 2.6

Figura 2.6: Ejemplo de Smart Glasses RAY-BAN META RW4006 (Cottet, 2025a).

2.3.1. Usos y beneficios

Las Smart Glasses son gafas que incorporan una tecnología que permite integrar funciones (como aplicaciones) muy parecidas a las que nos ofrece cualquier teléfono móvil.

Es cierto que gran parte de sus aplicaciones se encuentran todavía en fase de desarrollo, una de las características más relevantes que suelen incorporar todas las Smart Glasses es la realidad aumentada, que permite mezclar la realidad que un usuario percibe a través de las lentes con elementos virtuales. Podemos tomar como uno de los primeros ejemplos de esto a las Google Glass (Google, 2025) que se comercializaron en 2014, aunque sin lograr una gran cantidad de ventas.

Una de las funciones más importante de las Smart Glasses disponibles hoy en el mercado es la capacidad de hacer fotos y grabar vídeos, con la opción de compartirlos al instante en redes sociales directamente desde el propio dispositivo. Además, muchas de ellas también permiten realizar y recibir llamadas, otra funcionalidad clave especialmente para las personas que no le dan tanta importancia a las redes sociales, facilitando la comunicación (Cottet, 2025b).

Hay un gran número de empresas que se encuentran trabajando en el desarrollo de nuevas funcionalidades para mejorar la experiencia de las Smart Glasses y explotar este mercado, que pronto se prevee que sufra un crecimiento exponencial. Un ejemplo de nueva funcionalidad es el asistente Facebook View, que conecta las gafas inteligentes con una aplicación y permite integrar diversas funcionalidades de forma fácil y sencilla, como la traducción instantánea o el control por comandos de voz.

Este tipo de gafas inteligentes nos enseñan un nuevo camino para el acceso a la tecnología a personas que cuenten con distintos tipos de problemas de movilidad o de diversidad funcional. Además, como profundizaremos más adelante, son aplicables en muchos ámbitos diversos de todo tipo y mucho más útiles en situaciones en las que nos resulta difícil usar nuestro teléfono móvil, como por ejemplo cuando estamos practicando algún deporte.

Ofrecen múltiples usos y beneficios, especialmente para personas con baja visión. Estas gafas incorporan tecnología avanzada, como cámaras, pantallas integradas y software de realidad aumentada, que permiten ampliar imágenes, ajustar el contraste y el brillo, y superponer información útil en tiempo real. Esto facilita actividades cotidianas como leer, reconocer rostros o desplazarse con mayor seguridad. Además, su diseño moderno y discreto reduce el estigma asociado a las ayudas visuales tradicionales, promoviendo su aceptación y uso continuo.

2.3.2. Aplicaciones prácticas

Las smart glasses se perfilan como una tecnología clave para la industria del futuro, optimizando procesos laborales y mejorando la productividad. Estos dispositivos ya están

transformando entornos de trabajo con aplicaciones prácticas como: (ATRIA, 2025).

2.3.2.1. Aplicación en montaje industrial

Las smart glasses guían visualmente a los operarios durante procesos complejos, mostrando instrucciones paso a paso para el posicionamiento de piezas, lo que reduce errores en y mejora la eficiencia.

2.3.2.2. Seguridad

Las smart glasses permiten comparar imágenes en tiempo real con bases de datos, facilitando identificación y rastreo (como el sistema usado por la policía china). También ofrecen funciones críticas como visualizar planos de edificios o localizar equipos de emergencia, mejorando la respuesta en situaciones de riesgo. Un uso muy interesante fue una aplicación que permitía ver planos de edificios en tiempo real o localizar una boca de incendio cercana.

2.3.2.3. Realidad aumentada

Estos dispositivos transforman entornos cotidianos en interfaces interactivas: al mirar un establecimiento, muestran instantáneamente reseñas y datos relevantes, mientras que al enfocar carteles publicitarios reproducen contenido multimedia asociado, como tráilers de películas. En la Figura 2.7 observamos un ejemplo de lo que pretende aportar esta tecnología.

Figura 2.7: Realidad aumentada con Smart Glasses

2.3.2.4. Comandos de voz

Esto es una parte fundamental de las gafas inteligentes, ya que muchos de ellos se controlan mediante nuestra voz. Algunas de las funcionalidades que permiten realizar serían: realizar llamadas, fotos o incluso vídeos con un simple comando de voz, o preguntas tal y como se realizan con los asistentes personales como GoogleHome.

2.3.2.5. Discapacitados visuales

Las smart glasses ofrecen una solución transformadora para personas con baja visión o ceguera, permitiendo la lectura de textos y detección de obstáculos en tiempo real. Nuestro

proyecto se encuadra dentro de este tipo de aplicación, brindando mayor autonomía a usuarios con discapacidad visual.

2.3.2.6. Sanidad

La alta resolución de las imágenes permite su uso tanto en formación médica como en telemedicina. Las smart glasses han revolucionado este ámbito desde que en 2013 estudiantes de la Universidad de Ohio observaron una cirugía en tiempo real a través de sus dispositivos, marcando un hito en la educación médica a distancia. (ATRIA, 2025).

2.3.2.7. Deporte

El mercado de gafas inteligentes deportivas está experimentando un crecimiento notable, especialmente en el ámbito del ciclismo. Estos dispositivos especializados monitorizan métricas clave como frecuencia cardíaca, velocidad y distancia recorrida, proyectando los datos en pantalla para que el usuario pueda consultarlos sin interrumpir su actividad. En la Figura 2.8 podemos ver un ejemplo de uso en este ámbito.

Figura 2.8: Uso de Smart Glasses en deporte

2.4. Tecnologías para sistemas IoT

En este apartado vamos a describir las tecnologías, como frameworks o librerías, que hemos empleado en este proyecto para implementar este sistema **IoT** (Internet of Things o Internet de las Cosas) para baja visión.

2.4.1. IoT y sistemas de ayuda visual

Podemos definir al término IoT, o Internet de las cosas, como una red colectiva de dispositivos conectados y unas tecnologías que facilitan la comunicación entre la nube y cualquier dispositivo tecnológico, así como la comunicación entre estos dispositivos.

Gracias a los chips económicos y las redes de alta velocidad, hoy en día existen miles de millones de dispositivos, desde electrodomésticos hasta vehículos, que están conectados a Internet. Esto quiere decir que cualquier dispositivo de uso diario, ya sea una aspiradora, un coche o incluso un simple cepillo de dientes, podrían utilizar sensores para responder de forma inteligente a las personas con los datos que recojan. Mediante sensores, estos objetos se comunican entre sí y actúan de forma inteligente, apoyándose en tecnologías como la nube, big data o la movilidad, todo sin apenas intervención humana. (Oracle, 2025).

El IoT puede mejorar nuestras vidas en numerosos aspectos. Por ejemplo, en una casa que tenga implementada estas nuevas tecnologías, tu despertador no solo haría la función de despertarte por la mañana, sino que también pdoría ajustar la temperatura de la habitación, encender la ducha a la temperatura que más nos guste o prepararnos el café. La nevera también podría detectar qué alimentos faltan y ordenarlos automáticamente, mientras el asistente virtual revisa el tráfico de ese día y te dice la mejor ruta para llegar al trabajo. Incluso tu coche, si está conectado al sistema, podría ser capaz de reservar una plaza de aparcamiento cerca de la oficina él solo. Esta interconexión de dispositivos simplificará tareas cotidianas, ahorrando tiempo y mejorando la eficiencia en el día a día de las personas (AWS, 2024).

Podemos poner como ejemplo de sistema IoT diseñado para ayudar a personas con baja visión a **OrCam MyEye**, un dispositivo wearable que se acopla a las gafas y utiliza inteligencia artificial para asistir a los usuarios. Este dispositivo puede leer texto en tiempo real (libros, etiquetas, pantallas), reconocer rostros, identificar productos y hasta describir objetos o colores. Funciona de manera autónoma, sin necesidad de conexión a Internet, y se controla mediante gestos simples (Orcam, 2024).

Este tipo de tecnología mejora la calidad de vida de las personas con baja visión además de su autonomía, permitiéndoles interactuar mejor con su entorno.

2.4.2. Pose Detection

La detección de pose humana (Sergio Pérez, 2022) es un problema fundamental en visión por computador, que busca identificar la postura de una persona a partir de una imagen suya. Esta pose se define a partir de una serie de keypoints o puntos clave, como pueden ser las articulaciones, de forma que el objetivo que se pretende lograr es encontrar la posición en 2D (x, y) de cada uno de esos puntos, o en 3D (x, y, z) si queremos identificar también la profundidad en el espacio, aunque esto último aumenta su complejidad.

Para llevar a cabo esta tarea, existen diversas herramientas y frameworks que facilitan la implementación de modelos de detección de poses. Dos de las principales opciones a tener en cuenta en este ámbito son Google ML Kit y MediaPipe, cada una con sus propias características y ventajas.

En las siguientes secciones, se explorarán en detalle las capacidades de ambas herramientas de IA y cómo pueden ser utilizadas para implementar sistemas de detección de poses en aplicaciones orientadas a personas con baja visión.

2.4.2.1. Google ML Kit

Podemos definir a ML Kit (Google ML Kit, 2024a) como un framework de desarrollo de machine learning de Google, cuya principal funcionalidad es simplificar la integración de aprendizaje automático a las aplicaciones web y móviles también, tanto iOS como Android. Ejemplos de modelos preentrenados que incluye podrían ser el de reconocimiento de texto, escaneo de códigos de barras o reconocimiento de puntos clave o keypoints de la cara de una persona.

Una de las ventajas clave de ML Kit es que puede funcionar tanto en la nube como en el dispositivo, lo que permite a los desarrolladores elegir entre mayor precisión (usando la nube) o mayor privacidad y velocidad (usando solo el dispositivo). Esto lo hace ideal para cualquier tipo de aplicación que requiera procesamiento en tiempo real, como las que son diseñadas para personas con baja visión, donde la rapidez y la privacidad son cruciales.

ML Kit ofrece una serie de APIs listas para usar, que cubren tareas comunes de machine

learning, como:

- **Reconocimiento de texto (OCR):** Extrae texto de imágenes, ideal para aplicaciones de escaneo de documentos o traducción.
- **Detección de rostros:** Identifica rostros y sus características, como expresiones o puntos clave de la cara.
- **Escaneo de códigos de barras y QR:** Perfecto para aplicaciones de pago, inventario o acceso rápido a información.
- **Clasificación de imágenes:** Reconoce objetos, escenas o actividades en imágenes.
- **Traducción en dispositivo:** Traduce texto sin necesidad de conexión a Internet.
- **Etiquetado de imágenes:** Asigna etiquetas descriptivas a imágenes.

La API de detección de poses de ML Kit es una solución muy útil y versátil para que los desarrolladores de este tipo de aplicaciones sean capaz de detectar la pose del cuerpo de una persona en tiempo real a partir de un video o de una imagen que no esté en movimiento (Google ML Kit, 2024b). Al fin y al cabo, una pose puede ser considerada como un conjunto de coordenadas o datos que describen la posición del cuerpo en un momento determinado, son un conjunto de puntos de referencia del esqueleto de un cuerpo humano. Estos puntos de referencia corresponden a diferentes partes del cuerpo, como pueden ser las muñecas, los codos o las rodillas, y podemos usar las posiciones relativas de estos puntos para distinguir una pose de otra. A continuación se muestran en la Figura 2.9 los puntos de referencia que están definidos en esta funcionalidad.

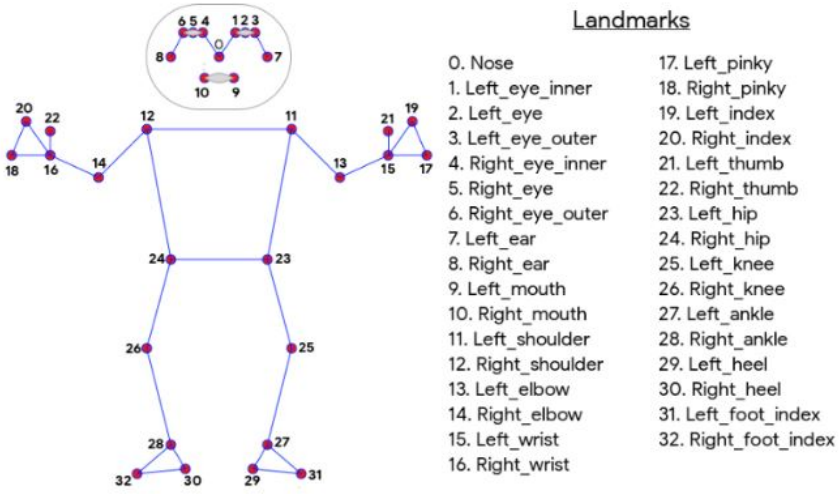

Figura 2.9: Puntos de referencia de Pose Detection ML Kit (Google ML Kit, 2024b)

Es un repertorio muy amplio como podemos observar, nos da una gran abanico de posibilidades para implementar sistemas que requieran de este tipo de funcionalidad de reconocer el cuerpo de una persona. En la Figura 2.10 se muestra un ejemplo real del uso de esta funcionalidad con algunos de los puntos de referencia mencionados anteriormente.

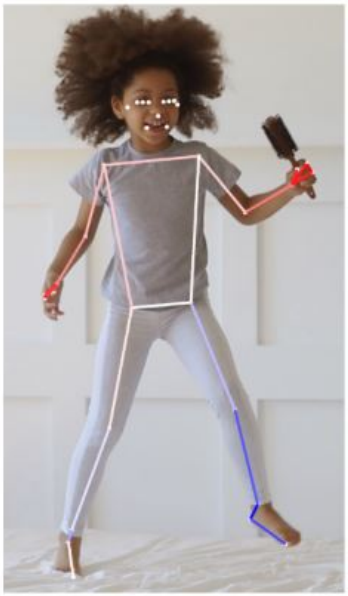

Figura 2.10: Ejemplo Real de Pose Detection (Google ML Kit, 2024b)

2.4.2.2. MediaPipe y discusión

MediaPipe es un framework desarrollado por Google de código abierto que permite crear aplicaciones de machine learning (ML) y procesamiento multimedia de manera rápida y eficiente (MediaPipe, 2025). Está diseñado para facilitar la implementación de pipelines (flujos de trabajo) que integran modelos de ML con procesamiento de videos, audios y datos en tiempo real. MediaPipe es especialmente conocido por su capacidad para trabajar con aplicaciones multiplataforma, incluyendo Android, iOS, web y dispositivos integrados.

Ofrece una serie de soluciones preconstruidas, como son la detección de rostro y puntos de referencia de la cara, detección de pose y seguimiento ocular. Además, es reconocido por su buen rendimiento en tiempo real y su personalización. Al igual que ML Kit ofrece una funcionalidad de Pose Detection, como podemos observar en la Figura 2.11:

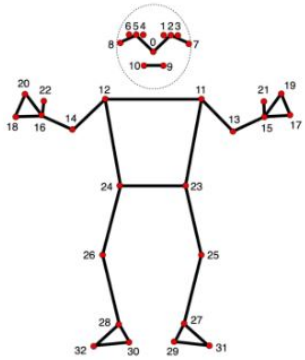

Figura 2.11: Puntos de referencia de Pose Detection MediaPipe (MediaPipe, 2025)

Llegamos a la conclusión de que ML Kit y MediaPipe eran las dos mejores opciones para llevar a cabo nuestro modelo de detección de poses. Tras este análisis, procedimos a evaluar las ventajas y limitaciones de ambas plataformas.

Una de las mayores ventajas de ML Kit es su optimización para dispositivos móviles. Está diseñado para funcionar de manera fluida en Android e iOS, incluso en dispositivos de gama media, lo que lo convierte en una gran opción para aplicaciones que requieren baja latencia pero un alto rendimiento a la vez. Además, al procesar los datos directamente en el dispositivo, garantiza la privacidad del usuario, ya que no es necesario enviar información a la nube. Otra ventaja es su facilidad de integración, ya que proporciona una API lista para usar, con una documentación clara y ejemplos de código que nos permitían implementar Pose Detection eficazmente.

MediaPipe, en cambio, al ser una herramienta más general consume más recursos, y al estar diseñada para ser una herramienta multipropósito, no está tan optimizada específicamente para tareas como Pose Detection.

Nosotros buscábamos una solución optimizada para Android, priorizando la privacidad y el procesamiento en el dispositivo y con facilidad de integración mediante una API. Por lo que, aunque es cierto que aún se encuentra en una fase beta, ML Kit ofrece justo lo que estábamos buscando para el desarrollo de nuestro proyecto. Por ello y por su fácil integración con el resto del software (Android Studio), decidimos que era la mejor opción para lograr los objetivos de nuestro proyecto.

Si bien es cierto que MediaPipe es una herramienta muy potente y flexible y que hubiese sido una opción mejor si nos hubiéramos centrado en otros aspectos similares como reconocimiento facial o seguimiento ocular, decidimos que para la detección de pose en concreto ML Kit iba a sernos más útil.

2.5. Técnicas de Inteligencia Artificial

Una de las características principales de este proyecto es la incorporación de inteligencia artificial en el desarrollo de la aplicación. Para lograrlo, exploramos diferentes técnicas para poder entrenar modelos y evaluarlos, además de frameworks y distintas herramientas que puedan aportar a este objetivo.

2.5.1. Aprendizaje automático

El aprendizaje automático (AA) es una rama fundamental de la inteligencia artificial que hace posible que los sistemas mejoren de manera autónoma mediante técnicas como redes neuronales y aprendizaje profundo, sin tener que depender exclusivamente de instrucciones programadas. Este proceso se basa en el análisis de grandes cantidades de datos para identificar patrones y poder generar modelos predictivos (Google Cloud, 2025).

A medida que estos sistemas procesan mayor cantidad y diversidad de datos, aumenta su capacidad de adaptación y precisión, optimizando su rendimiento. El hecho de que sean muy escalables y la calidad de los conjuntos de datos son factores claves que determinan el éxito de estas soluciones basadas en aprendizaje automático.

Existen tres tipos de modelos que se usan en el aprendizaje automático:

- El **aprendizaje supervisado**, que usa datos etiquetados para el entrenamiento del modelo. El algoritmo aprende a asignar entradas a salidas conocidas (por ejemplo, reconocer una manzana en una foto). Es ideal para tareas de clasificación y predicción.
- El **aprendizaje no supervisado**, que trabaja con datos sin etiquetar para identificar patrones o agrupar información. El algoritmo categoriza los datos por sí solo (por ejemplo, separar imágenes de manzanas y bananas). Es muy útil para análisis exploratorio y agrupación.

- El **aprendizaje por refuerzo**, en el que el modelo aprende mediante prueba y error, recibiendo retroalimentación positiva o negativa según sus acciones. Es como "aprender haciendo" (por ejemplo, un algoritmo que juega a un videojuego y mejora con cada partida). Es la mejor opción para tareas secuenciales y toma de decisiones.

2.5.1.1. Redes neuronales

Las **redes neuronales** (IBM, 2025b) son modelos computacionales inspirados en el funcionamiento del cerebro humano. Son capaces de aprender patrones muy complejos a partir de datos para realizar tareas de clasificación, predicción y toma de decisiones. Estas redes, que desempeñan un papel fundamental en el aprendizaje automático y la inteligencia artificial, procesan información mediante capas de neuronas artificiales que están interconectadas entre sí. Cada neurona recibe señales, las pondera según su importancia y las transmite si superan un umbral de activación, permitiendo que la red sea capaz de aprender por sí misma relaciones no lineales en los datos.

El entrenamiento de una red neuronal consiste en ajustar progresivamente los pesos y umbrales de sus conexiones mediante algoritmos, intentando minimmizar siempre los errores en sus predicciones. A medida que lo entrenamos con un mayor cantidad de datos, el modelo mejora su precisión, convirtiéndose en una herramienta muy poderosa para muchos tipos de aplicaciones que requieren procesamiento rápido y automatizado. Por ejemplo, en reconocimiento de imágenes o voz, las redes neuronales pueden analizar grandes volúmenes de información empleando infinitamente menos tiempo del que le tomaría a un humano, como demuestra el algoritmo de búsqueda de Google, que emplea estas técnicas para interpretar consultas y ofrecer resultados relevantes. Su versatilidad y capacidad de aprendizaje las están convirtiendo en los últimos años en herramientas indispensables en campos que van desde la medicina hasta la robótica.

El uso de estas redes neuronales es indispensable en nuestro proyecto para poder entrenar a nuestro propio modelo de detección de poses desde cero. En la Figura 2.12 podemos observar un ejemplo del esquema de una red neuronal.

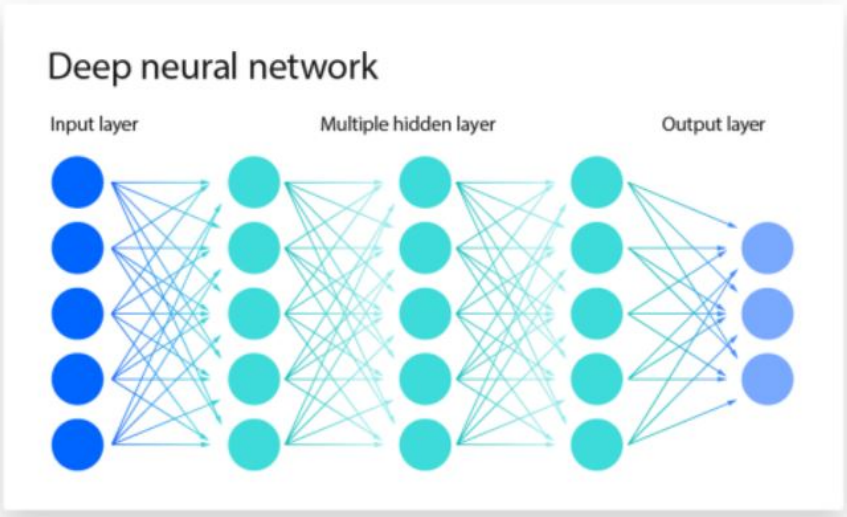

Figura 2.12: Ejemplo de red neuronal (IBM, 2025b)

2.5.2. TensorFlow Lite

TensorFlow Lite (Instituto EITCI, 2025; TensorFlow, 2021) es un marco de trabajo desarrollado por Google específicamente para implementar modelos de aprendizaje automático en dispositivos con recursos limitados, como móviles, tabletas y dispositivos IoT. Su diseño optimizado permite ejecutar modelos TensorFlow de manera eficiente, ofreciendo ventajas clave como menor latencia y mayor privacidad al procesar datos localmente. El framework incluye herramientas para convertir modelos estándar a un formato optimizado, junto con bibliotecas especializadas que facilitan su despliegue en diversas plataformas (Android, iOS y sistemas embebidos), manteniendo un equilibrio entre rendimiento y consumo de recursos. Sus características clave (TensorFlow, 2021) son las siguientes:

- Está optimizado para el aprendizaje automático integrado en el dispositivo que vayamos a emplear.
- Es compatible con diversos lenguajes.
- Tiene un alto rendimiento, con optimización de modelos y aceleración de hardware.
- Se puede emplear en un gran número de tareas útiles que tengan que ver con aprendizaje automático en múltiples plataformas, como clasificación de imágenes, estimación de poses, detección de objetos, clasificación de texto, etc.

Nosotros necesitábamos encontrar una forma de integrar nuestro modelo entrenado en nuestra aplicación de Android Studio, y decidimos usar TensorFlow Lite para darle solución a nuestro problema.

2.5.3. Random Forest

El Random Forest (Inesdi, 2025) es un algoritmo de aprendizaje supervisado que resuelve problemas de clasificación y regresión mediante conjuntos de árboles de decisión. Cada árbol es entrenado con una muestra aleatoria de datos y un subconjunto de características, lo que hace que mejore la precisión y se reduzca el sobreajuste. Para clasificación elige la clase más votada entre todos los árboles, mientras que para regresión calcula el promedio de sus predicciones.

Este es un método muy robusto y maneja bien datos complejos, identificando además la importancia de cada característica. Sin embargo, su principal desventaja es el mayor consumo de recursos, ya que requiere más tiempo y memoria que otros algoritmos más simples. A pesar de esto, su eficacia lo convierte en una muy buena opción a implementar en aplicaciones donde la precisión es algo prioritario. Estos son algunos de sus beneficios (IBM, 2025a):

- **Reduce el sobreajuste:** A diferencia de un solo árbol de decisión, que puede adaptarse demasiado a los datos de entrenamiento, el Random Forest combina múltiples árboles no correlacionados. Este enfoque promedia sus resultados, disminuyendo la varianza y minimizando errores de predicción sin comprometer la generalización del modelo.
- **Versatilidad en aplicaciones:** El algoritmo destaca por su capacidad para abordar tanto problemas de clasificación como de regresión con alta precisión. Esta flexibilidad lo convierte en una herramienta muy valiosa para diversas necesidades en ciencia de datos.

- **Identificación de características clave:** El Random Forest no solo ofrece predicciones precisas, sino que también es capaz de cuantificar la importancia de cada variable en el modelo. Esta funcionalidad ayuda a priorizar características relevantes y optimizar el análisis.

Probamos a entrenar nuestro modelo usando el algoritmo random forest en lugar de redes neuronales, pero obtuvimos mejores resultados que explicaremos más adelante. Podemos ver un ejemplo de la estructura de esta implementación en la Figura 2.13.

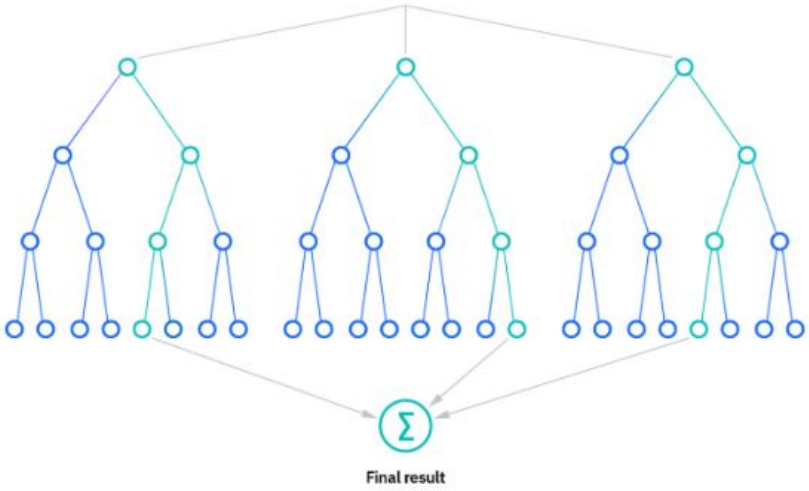

Figura 2.13: Ejemplo de Random Forest (IBM, 2025a)

2.6. Conclusiones

En resumen, las conclusiones que obtenemos tras este estudio es que la baja visión en personas es un problema muy a tener en cuenta hoy en día, y que definitivamente podemos solventar en cierta medida con los avances tecnológicos de estos tiempos. Actualmente, las gafas inteligentes con tecnología de realidad aumentada integrada, mediante sus cámaras y sensores, permiten ofrecer asistencia visual en tiempo real, mejorando significativamente la autonomía de personas con baja visión.

Es por ello que usando modelos de detección de pose (Pose Detection) y técnicas de IA como las redes neuronales, queremos ser capaces de aprovechar las posibilidades que nos ofrecen las gafas inteligentes y aportar una solución para facilitar el día a día de la gente que padece enfermedades relacionadas con la baja visión, que limita su campo visual e impide vivir con normalidad.

Con la llegada inminente de las gafas inteligentes a nuestro día a día, pretendemos crear una solución intuitiva y fácil de usar, mejorando la eficiencia de las ayudas visuales tradicionales.

Capítulo 3

Planificación y recursos

En este capítulo se presentará la planificación llevada a cabo durante el desarrollo del proyecto, así como las distintas herramientas utilizadas en su implementación. Además, se expliacarán los criterios de selección de las herramientas, justificando su elección en comparación con otras alternativas disponibles.

Gracias a las conclusiones del capítulo anterior, una vez entendida la necesidad a solucinar junto a la de investigación y entendimiento de todas las herraientas y tecnologias que tenemos a nuestro alcance, hemos desarrollado una aplicacion movíl con la capacidad de satisfacer los requerimientos del proyecto. Se utiliza en esta sección para documentar el proceso de selección de cada una de las herramientas y tecnologias.

Asimismo se expondrán los desafios encontrados durante la elaboración del proyecto y como fueron abordados, garantizando que la aplicación cumpla con los objetivos propuestos en terminos de funcionalidad, usabilidad y eficiencia.

3.1. Gestión, desarrollo y documentación del proyecto

3.1.1. GitHub

En cuanto al uso de GitHub, en nuestro proyecto empezamos a emplearla como herramienta de control de versiones y colaboración en el desarrollo de la aplicación. GitHub es un sistema de gestión de proyectos que está basado en Git, en la que el código fuente de un software se mantiene de forma eficiente, al ser un servicio de alojamiento de repositorios Git, hace fácil compartir el código en todo el mundo y a medida que el software se desarrolla a lo largo del tiempo, los desarrolladores pueden realizar un seguimiento de esas actualizaciones e incluso revertir el software a una versión anterior.

Por otro lado, se puede trabajar en diferentes versiones de ese código sin interferir en el código principal. El trabajo en GitHub se reparte en diferentes ramas que hacen que podamos estructurar el código base y trabajar de manera paralela en diferentes rama para el desarrollo de otras funcionalidades. Facilita nuestra integración con las herramientas más importantes, en este caso Android Studio.

3.1.2. Google Docs

Desde el comienzo de la creación del proyecto, utilizamos Google docs para tomar notas compartidas sobre reuniones y actualizaciones del proceso. Llevar un registro sistemático de todas las decisiones, ideas y tareas pendientes en un archivo común, lo que garantizó que

cada miembro del grupo utilizara la misma información cuando fuera necesario recordar momentos del desarrollo del proyecto.

3.1.3. Google Colab

Asimismo, otro recurso que se utilizó para el entrenamiento de los modelos aplicados en nuestro proyecto es Google Colaboratory, una plataforma basada en Jupyter Notebooks que brinda la posibilidad de ejecutar código en la nube con recursos computacionales y de almacenamiento de Google, tales como GPUs y TPUs. El uso de esta herramienta nos facilitó entrenar los modelos de manera más ágil sin tener que depender únicamente del hardware local; en consecuencia, aceleramos el proceso de desarrollo y optimizamos el rendimiento de nuestras soluciones.

3.1.4. Overleaf

Acabamos utilizando la plataforma Overleaf para desarrollar la memoria de nuestro proyecto, ya que nos fue recomendada por nuestros tutores.

Overleaf es una herramienta en línea para la edición de documentos en LaTeX, un sistema ampliamente utilizado en el ámbito académico y científico por su capacidad para gestionar textos complejos con gran precisión. Entre sus principales ventajas destacan:

- Colaboración en tiempo real, permitiendo que varios miembros del equipo trabajen simultáneamente en el documento.
- Formato profesional y estructurado, ideal para la redacción de documentos técnicos, tesis y artículos científicos.
- Manejo eficiente de referencias y bibliografía, facilitando la citación mediante herramientas como BibTeX.
- Control de versiones, asegurando un historial de cambios que permite revisar ediciones anteriores sin riesgo de perder información.

3.1.5. Justificación

Elegimos estas herramientas porque cubrian las necesidades que teniamos para empezar a desarrollar el proyecto, además consideramos que el sentirnos más cómodos con las aplicaciones era algo positivo, ya que las habíamos utilizado en diferentes asignaturas. Esto nos dio la facilidad para incluirlas en nuestro proyecto sin problemas y usar las herramientas a la perfección, es el mejor modo de volvernos productivos y eficientes en el desarrollo del proyecto lo antes posible, evitando tener que aprender el funcionamiento de nuevas herramientas.

Por otro lado, decidimos utilizar Overleaf para la memoria del proyecto, ya que nos permitió redactar y desarrollar un documento con un formato profesional y adecuadamente estructurado. Nos decidimos a utilizar esta herramienta debido a que nuestros tutores nos la recomendaron desde un primer momento, además una vez investigado y considerado las diferentes ventajas en términos de referencias, colaboración y control de versiones, no dudamos en utilizarla para desarrollar un documento final de alta calidad y adaptado a las convenciones académicas.

3.2. Recursos Hardware

Concluida la investigación sobre baja visión y sus soluciones actuales, se establece que, además de satisfacer los requisitos funcionales, las gafas deben contar con un diseño ligero y discreto, lo más próximo posible al de unas gafas convencionales. La elección del hardware es un factor clave, ya que determina la comodidad, accesibilidad y aceptación del dispositivo por parte del usuario.

3.2.1. Smart Glasses

Para nuestro proyecto, hemos elegido las Rokid Air Pro, unas gafas de realidad aumentada que se adaptan muy bien a los requisitos que buscábamos: ligereza, discreción, buena calidad de visualización y compatibilidad con Android. Su diseño se asemeja bastante al de unas gafas convencionales, lo que las hace más cómodas de usar en el día a día, especialmente para el perfil de usuario al que nos dirigimos. En la Figura 3.1 podemos observar un ejemplo de estas gafas.

Figura 3.1: Smart Glasses Rokid Air Pro.

En cuanto al precio, las Rokid Air Pro se sitúan en torno a los 500–600€, lo cual las coloca en una gama intermedia-alta, pero mucho más accesibles que otras opciones del mercado como las HoloLens o las Google Glass Enterprise, que duplican o incluso triplican esa cifra. Teniendo en cuenta su versatilidad, la calidad de sus componentes y la facilidad de integración, nos ha parecido una opción muy equilibrada.

3.2.2. Ventajas y Desventajas

Ventajas

- Alta compatibilidad con Android Studio, facilitando la integración de nuestra app sin necesidad de modificar el entorno de desarrollo.
- Diseño ergonómico y discreto, ideal para personas con baja visión que buscan evitar estigmatización social.
- Procesamiento visual de calidad, gracias a su pantalla de alta definición y sensores incorporados.
- Control simple, mediante un botón físico o a través del dispositivo Android.

Desventajas

Característica	Especificación
Peso	83 gramos, lo que las hace cómodas para un uso prolongado
Pantallas	Dual Micro OLED con resolución de 1920x1080 por ojo
Campo de visión (FOV)	43°, suficiente para integrar la información visual
Frecuencia de refresco	75 Hz
Sensores	Acelerómetro, giroscopio y magnetómetro integrados
Cámara	8 MP, con funcionalidad de grabación y captura útil para tareas futuras de interacción avanzada
Botón físico	Permite realizar acciones básicas sin necesidad de una interfaz por pantalla
Conectividad	USB-C para conexión directa con dispositivos Android compatibles

Tabla 3.1: Especificaciones técnicas del dispositivo Hardware

- Falta de autonomía propia: Al no contar con batería, requieren estar conectadas a un smartphone o mini-PC, lo que limita ligeramente la movilidad.

Aunque no cuentan con batería propia y necesitan estar conectadas a un smartphone o un mini-PC, creemos que este pequeño inconveniente queda compensado por todo lo que ofrecen. En conjunto, las Rokid Air Pro se han mostrado como una herramienta muy sólida para este tipo de proyectos, y su uso ha sido clave para poder dar forma a una solución portátil, eficaz y realista.

3.2.3. Dispositivo

Para poder utilizar las gafas Rokid Air Pro, es esencial contar con un dispositivo Android compatible.

- La conexión entre las gafas y el dispositivo se realiza mediante un puerto USB-C, que debe ser capaz de transmitir tanto datos como video de alta calidad.
- El dispositivo debe tener un procesador de gama media o alta, que permita ejecutar las aplicaciones de realidad aumentada de manera fluida. El Snapdragon 888 o superiores.
- Una batería de mínimo 4000 mAh, ya que el uso de las gafas consume bastante energía y un dispositivo con poca autonomía podría quedarse corto rápidamente.
- Recomendable un sistema Android 10 o superior para asegurarse la compatibilidad con la aplicación Rokid AR.

Este dispositivo debe cumplir con los requisitos de las smart glases y tener capacidades de cómputo suficientes para desarrollar la aplicación.

3.3. Lenguaje

Una vez elegidas las gafas de realidad aumentada, es momento de concretar los lenguajes de programación que se necesitan para realizar la aplicación. Dentro del ecosistema de desarrollo para Android, los lenguajes eficientes con el entorno son Kotlin y Java, son los únicos en los que se puede programar sin usar librerías puente. Ambos lenguajes son compatibles con las herramientas y bibliotecas imprescindibles para trabajar con RA, por lo tanto será necesario exponer sus características y ver cuál encaja mejor con nuestras necesidades.

3.3.1. Java

Java desde el principio fue nuestra primera opción ya que es un referente en la creación de aplicaciones Android y cuenta con una amplia comunidad y soporte. Al ser tan conocido existe mucha documentación, foros y guías que pueden ser útiles a la hora de resolver problemas que se puedan presentar durante el desarrollo. Otro punto que juega muy a su favor es que es uno de los lenguajes con los que más familiarizados nos sentimos, ya que lo hemos utilizado en nuestra formación académica, lo cual nos otorgaría una mayor seguridad y fluidez desde el inicio del desarrollo, ya que no tendríamos que dedicar ningún tiempo a aprender nuevas sintaxis. Además, es ampliamente compatible con las bibliotecas de realidad virtual disponibles para Android. Esto significa que no tendremos problemas para integrar las distintas herramientas necesarias.

3.3.2. Kotlin

Kotlin es un lenguaje de programación reciente que Google recomienda para desarrollar aplicaciones en Android. Es fácil de entender y usar, que permite escribir menos líneas de código. Kotlin maneja los errores de manera más eficaz, sobre todo cuando se trata de problemas con valores "null", un fallo común en Java. Una gran ventaja es su compatibilidad con Java, lo que permite utilizar librerías ya existentes sin inconvenientes. Nuestros tutores fueron los que nos dieron a conocer Kotlin y nos aconsejaron elegirlo para desarrollar nuestra aplicación, resaltando sus importantes beneficios en términos de productividad y eficiencia para crear aplicaciones Android actuales. Como desventaja encontramos que nunca hemos trabajo con Kotlin, por lo tanto tendríamos que establecer un pequeño periodo de aprendizaje y familiarización con el nuevo lenguaje.

3.3.3. Decisión final

Después de considerar ambas opciones, decidimos que Kotlin sería el lenguaje principal para implementar nuestra aplicación. A pesar de nuestra mayor experiencia con Java, lo que nos proporcionaría una mayor comodidad y sencillez a la hora de desarrollar la aplicación, las ventajas que ofrece Kotlin ya comentadas anteriormente fueron un factor determinante en nuestra decisión. Además de ser el lenguaje que nuestros tutores nos dieron a conocer y recomendaron, proporcionandonos la posibilidad de desarrollar una aplicacion más eficiente y moderna. Aunque requirió un breve período de adaptación, su compatibilidad con Java facilitó la transición, permitiéndonos aprovechar las bibliotecas existentes sin inconvenientes.

3.4. Android Studio

Hemos desarrollado nuestra aplicación de manera efectiva mediante Android Studio, que es el entorno de desarrollo integrado oficial proporcionado y respaldado por Google para el desarrollo de aplicaciones Android.

Android Studio incluirá un conjunto completo de herramientas que están diseñadas para un desarrollo del tiempo de ejecución, como el lenguaje de programación, Kotlin y Java, muy útiles para programar, depurar y probar la implementación de nuestra aplicación. Cuenta con un emulador muy avanzado, el cual nos permite probar nuestra aplicación en diferentes dispositivos sin la necesidad de hardware físico, también, tiene integraciones con bibliotecas y herramientas que nos brinda Google, lo que es fundamental para implementar funcionalidades de realidad aumentada. Entre sus ventajas cuenta con la inteligencia artificial, el autocompletado integrado, depurador y analizador de código, lo que mejora la productividad y evita errores; es el ambiente propicio para desarrollar un proyecto de estas características.

3.5. Dataset

La dataset desarrollada es uno de los elementos fundamentales del sistema, ya que de ella depende el rendimiento y la fiabilidad del modelo de detección de poses, en nuestro caso necesitabamos una base de datos muy especifica,la cual durante todo el desarrollo del proyecto ha ido evolucionando y transaformandose en función de las diferentes necesidades que han ido surgiendo. A traves de pruebas hemos acabando haciendo la seleccion de los datos más determinantes consiguiendo los resultados mas precisos gracias a nuestra dataset.

3.5.1. Datos propios o dataset existente

En primer lugar, valoramos la posibilidad de utilizar alguna bases de datos ya exitente que fuera compatible con nuestras necesidades. Estas bases ofrecen un gran volumen de imágenes y anotaciones de poses humanas. Sin embargo, presentan varios inconvenientes para nuestro caso concreto:

- Las poses específicas que necesitábamos (saludos, gestos de señalar, etc.) no estaban representadas de manera consistente.
- Muchas imágenes estaban tomadas en contextos poco relevantes para nuestra aplicación asistencial, como por ejemplo encontramos algunas relacionadas con las poses de yoga.
- Los datasets encontradas no cumplian con las dimensiones que queriamos tener, eran demasiado pequeñas.

Frente a esto, construir nuestra propia base de datos ofrecía ventajas importantes:

- Definir exactamente qué poses necesitábamos y cómo representarlas.
- Capturar la información en un entorno controlado.
- Adaptar las características extraídas (coordenadas, distancias, ángulos) a las necesidades de nuestro modelo.

Por tanto, optamos por la creación de una base propia, a pesar del mayor esfuerzo inicial requerido. En la Figura 3.2 y Figura 3.3 mostramos frames de los vídeos que nos grabamamos para crearla.

Figura 3.2: Pose saludo con la derecha

Figura 3.3: Pose brazos cruzados

3.5.2. Construcción del dataset

Cada miembro del grupo grabó vídeos realizando cada una de las ocho poses seleccionadas. Para asegurar la diversidad corporal y mejorar la generalización, intentando diversificar todo lo posible. Los vídeos fueron divididos en frames, y cada frame fue procesado para extraer las coordenadas de los puntos clave del cuerpo.

Mediante una aplicación desarrollada en Android Studio, automatizamos el proceso de extracción de datos, obteniendo archivos CSV en los que cada fila corresponde a un frame y cada dos columna a un punto del cuerpo.

3.5.3. Primer dataset

En un primer momento, cada fila del CSV correspondía a un frame y contenía las coordenadas X e Y de 33 puntos del cuerpo, lo que daba lugar a un total de 66 columnas. Esta versión inicial fue útil para comenzar los primeros entrenamientos, pero pronto identificamos limitaciones en cuanto a precisión y rendimiento del modelo. En la Figura 3.4 se puede apreciar su formato.

		AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL	BM	BN	BO
1	17883	0.71128476	0.41266954	0.72540325	0.5640313	0.90166545	0.43643096	0.9051933	0.5404408	0.9306047	0.46478584	0.9301403	0.59556603	0.95690197	0.35759002	0.9615855	Señalando a la izqu
2	81467	0.70686215	0.41517356	0.7226215	0.5625314	0.8862313	0.4368865	0.9023969	0.53778845	0.90958965	0.46622768	0.92339504	0.5941731	0.94431823	0.3620089	0.95827645	Señalando a la izqu
3	68712	0.70744365	0.41221905	0.7225087	0.56108224	0.8712158	0.4320909	0.89958143	0.53615963	0.89499307	0.4619613	0.92175925	0.5907291	0.93150073	0.3616533	0.9562018	Señalando a la izqu
4	25012	0.71246374	0.40848207	0.7263885	0.5673128	0.8640115	0.4326414	0.9025865	0.5392107	0.8650078	0.45817474	0.9274156	0.59777415	0.92534834	0.3583709	0.9592801	Señalando a la izqu
5	99114	0.7111696	0.4110753	0.7247624	0.57273644	0.8572554	0.4301687	0.89788187	0.5437908	0.878858	0.45659855	0.9245678	0.60424644	0.91514146	0.36444867	0.95657	Señalando a la izqu
6	13008	0.7098181	0.41057056	0.72264496	0.57919776	0.8457289	0.4284642	0.8990174	0.55468976	0.86285466	0.45384976	0.9246131	0.60780627	0.90774435	0.36885226	0.9563501	Señalando a la izqu
7	34179	0.7116265	0.4084906	0.72430384	0.58477527	0.84462285	0.42713115	0.89865315	0.563735	0.8602148	0.4511667	0.92472935	0.6147735	0.90748864	0.369602	0.95555604	Señalando a la izqu
8	55786	0.709238	0.41793796	0.7217532	0.5909137	0.8494275	0.43323165	0.90014935	0.57831776	0.86293364	0.4533473	0.92081654	0.61637485	0.90863675	0.3664978	0.95279276	Señalando a la izqu
9	36498	0.70751154	0.41551176	0.7219563	0.59667116	0.85158736	0.43208918	0.90053666	0.5882257	0.8640866	0.45171228	0.9214421	0.6169158	0.9091257	0.3642915	0.95456874	Señalando a la izqu
10	13603	0.7086376	0.41112852	0.72440964	0.59659225	0.8541254	0.43685812	0.90107644	0.57707703	0.8727521	0.45841822	0.919633	0.6223422	0.908816	0.36509284	0.95451796	Señalando a la izqu
11	82194	0.70703584	0.42068258	0.7196538	0.59152526	0.8512039	0.43414742	0.8992233	0.5708271	0.87110204	0.4538205	0.9189106	0.61918414	0.9085545	0.36395004	0.95317346	Señalando a la izqu
12	53866	0.7059015	0.4230386	0.7200109	0.5885187	0.85511667	0.4360966	0.90186375	0.5694079	0.87513477	0.46033812	0.92070603	0.6243973	0.9100896	0.3725645	0.95964813	Señalando a la izqu
13	54702	0.70149183	0.42409724	0.71739334	0.5890804	0.8568156	0.43399012	0.900244	0.5626668	0.8811648	0.45856333	0.920263	0.6254901	0.9091865	0.3694238	0.95831716	Señalando a la izqu

Figura 3.4: Últimas columnas de dataset inicial de "Señalando con la izq".

3.5.4. Optimización

Tras los primeros resultados, realizamos una serie de mejoras en la base de datos, que permitieron un salto significativo en la calidad de los datos y del sistema:

Normalización de coordenadas: adaptamos todos los valores para que no dependieran del tamaño del vídeo ni de la posición de la persona en la imagen. Esto permitió al modelo aprender la forma de las poses sin verse afectado por factores externos.

Eliminación de puntos redundantes: descartamos aquellos puntos que no aportaban información útil para nuestras poses, como algunos de la cara o de las piernas, reduciendo así el ruido en los datos.

Cálculo de distancias: añadimos nuevas columnas con las distancias entre puntos clave del cuerpo, como hombro–muñeca, codo–hombro o cadera–hombro. Estas medidas permiten representar mejor la estructura general de cada pose.

Cálculo de ángulos: introdujimos también los ángulos formados por tríos de puntos, lo que resultó especialmente útil para distinguir entre poses similares con pequeñas variaciones de orientación.

Gracias a estas optimizaciones, la base de datos final no solo contiene coordenadas, sino también características derivadas que aportan mucha más información y ayudan al modelo a distinguir de forma más precisa entre clases. En la Figura 3.5 se encuentra el formato de nuestro dataset final.

	A	B	C	D	E	F	G	H	I	J
1	LeftShoulderAngle	RightShoulderAngle	LeftElbowAngle	RightElbowAngle	LeftWristToHeadDist	RightWristToHeadDi	LeftElbowToHipDist	RightElbowToHipDis	WristToWristDist	Pose
2	0.55082524	0.55135274	0.9152346	0.9105839	0.92635983	0.9058204	0.67445934	0.63632375	0.9992269	1
3	0.56126744	0.55271125	0.922402	0.90870804	0.92026496	0.90579027	0.6736438	0.62754744	0.99915516	1
4	0.54905134	0.5365621	0.89008313	0.90863323	0.91525275	0.91253304	0.6769387	0.6260434	0.9991761	1
5	0.5597944	0.55563587	0.9042105	0.9199534	0.91054934	0.90818936	0.6694119	0.6259607	0.9990883	1
6	0.55689925	0.5546261	0.90528744	0.9041716	0.9066172	0.89616615	0.6600824	0.62060595	0.99890566	1
7	0.5510784	0.5389438	0.866421	0.892182	0.90382457	0.8917786	0.67313206	0.6169868	0.99883586	1
8	0.53866273	0.564832	0.8647327	0.8781064	0.88559264	0.8735601	0.6514403	0.65732646	0.9983369	1
9	0.55296874	0.55450433	0.86536634	0.84828705	0.8647716	0.8468934	0.6643291	0.6441042	0.9975126	1
10	0.5585368	0.5475297	0.844476	0.85199016	0.86072725	0.8368826	0.6723425	0.63072854	0.9972199	1

Figura 3.5: Dataset Final.

3.5.5. Análisis BD

3.5.5.1. Tamaño y contenido

- **9.260 muestras** en total, correspondientes a frames individuales.
- **10 columnas** en cada muestra, que recogen:
  - Ángulos de hombros y codos (izquierdo y derecho).
  - Distancias de muñecas a la cabeza y de codos a caderas.
  - Distancia entre ambas muñecas.
  - Una columna adicional que indica la pose de cada frame.

3.5.5.2. Distribución de clases

La base de datos contiene ejemplos de 8 poses distintas, con una distribución bastante equilibrada :

Clase	Tipo de Pose	Número de muestras
0	Pose neutral	1062
1	Brazos abiertos	1228
2	Brazos cruzados	1201
3	Manos en la cintura	1229
4	Saludo con la derecha	1115
5	Saludo con la izquierda	1107
6	Señalando a la derecha	1230
7	Señalando a la izquierda	1088

Tabla 3.2: Distribución de muestras por tipo de pose

Esto demuestra que no existe un desequilibrio crítico entre clases, lo cual es esencial para entrenar modelos fiables y sin sesgos.

3.5.6. Conclusión

La planificación detallada y la elección de cada recurso, desde el flujo colaborativo con GitHub y Google Docs hasta la selección de Kotlin y Android Studio, han sido determinantes para acelerar el desarrollo y asegurar la calidad técnica del prototipo. Integrar el SDK de Rokid y un hardware realista desde el principio evitó retrabajos costosos, mientras que diseñar una base de datos propia, normalizada y enriquecida con distancias y ángulos, garantizó que nuestro modelo aprendiera de datos precisos y relevantes. Esta combinación de buenas prácticas, decisiones tecnológicas y un dataset adecuado para el aprendizaje no solo sienta una base sólida para las siguientes fases, sino que permite abordar de forma efectiva y escalable retos de asistencia visual para baja visión.

Capítulo 4

Desarrollo del proyecto y resultados

4.1. Visión general del proyecto

Este proyecto propone el diseño e implementación de una solución basada en una aplicación móvil, potenciada por algoritmos de inteligencia artificial, que se comunica con unas gafas inteligentes ("Smart Glasses") mediante conexión USB-C para asistir a personas con dificultades visuales. Mediante la cámara incorporada en las gafas, el sistema captura en tiempo real los principales puntos de referencia corporal de las personas que rodean al usuario. A continuación, estos datos de pose se procesan y se convierten en información auditiva mediante un motor de síntesis de voz, de modo que el usuario recibe retroalimentación instantánea sobre la disposición y el movimiento de su entorno, mejorando así su autonomía y seguridad.

El desarrollo siguió una metodología iterativa que combinó prototipado rápido y validación continua. En la primera fase, se exploró el uso de la cámara del teléfono móvil junto con la librería ML Kit de Google, capaz de identificar hasta 33 puntos clave (joints) del cuerpo humano. A partir de secuencias de vídeo grabadas, se generó un conjunto inicial de datos (dataset) en el que cada pose se describía mediante coordenadas bidimensionales. Para enriquecer esta representación y aumentar la robustez del modelo, se calcularon ángulos articulares y distancias relativas entre puntos (por ejemplo, la separación hombro–codo o cadera–rodilla), lo que permitió capturar mejor la variabilidad postural y reducir la sensibilidad a la posición de la cámara.

Con el dataset refinado, se compararon varias arquitecturas de aprendizaje automático. Por un lado, se entrenaron redes neuronales profundas (deep learning) que aprendían directamente de las características angulares y espaciales; por otro, se probaron algoritmos clásicos como Random Forest para evaluar la relevancia de cada variable y la capacidad de generalización. Este análisis dual permitió optimizar tanto la precisión en la clasificación de posturas como la eficiencia computacional necesaria para su ejecución en un dispositivo móvil.

Finalmente, se integró el modelo seleccionado en la aplicación y se estableció la comunicación con las Smart Glasses. Se incorporaron funcionalidades de accesibilidad adicionales —como ajustes de idioma, modos de color en la interfaz de la app y tamaño de texto— para adecuar la experiencia a las necesidades específicas de cada usuario. El resultado es un sistema robusto y probado que demuestra la viabilidad de combinar visión por computadora en tiempo real con dispositivos de asistencia para mejorar significativamente la interacción de las personas con baja visión con su entorno.

4.2. Desarrollo de la aplicación móvil

En esta sección se describe detalladamente el proceso de diseño e implementación de la aplicación móvil desarrollada en Android Studio utilizando el lenguaje Kotlin. Se abordan los pasos iniciales de configuración del entorno de desarrollo, así como la integración de herramientas clave como ML Kit, que permite realizar tareas avanzadas de visión por computadora, incluyendo la detección y análisis de poses corporales en tiempo real.

Además, se detalla la estructura general del proyecto, las decisiones de arquitectura del software adoptadas, y las estrategias de control de versiones implementadas. Estas prácticas no solo garantizan la calidad y mantenibilidad del código, sino que también favorecen la colaboración efectiva entre desarrolladores a lo largo de las distintas fases del desarrollo.

4.2.1. Configuración del entorno de desarrollo

Para garantizar un entorno sólido y reproducible, se eligió Android Studio como IDE principal, junto con Kotlin, por ser el lenguaje de referencia moderno recomendado por Google para Android. A continuación se describen los pasos seguidos para crear y parametrizar el proyecto.

4.2.1.1. Creación del proyecto en Android Studio

El primer paso en el desarrollo consistió en la creación del proyecto mediante Android Studio. Esta plataforma proporciona múltiples plantillas prediseñadas que permiten iniciar el desarrollo para diferentes tipos de dispositivos, tales como teléfonos móviles, tabletas, relojes inteligentes, televisores e incluso automóviles. En particular, las plantillas destinadas a móviles y tabletas incluyen opciones básicas, como iniciar con una actividad vacía o sin actividad alguna, así como opciones más avanzadas, que incorporan elementos de navegación como barras inferiores o menús laterales. Dado que en esta etapa inicial no se tenía certeza sobre las funcionalidades que eventualmente serían necesarias, se optó por iniciar el proyecto con una actividad vacía, lo cual proporciona una base limpia sobre la que desarrollar de forma personalizada las características requeridas. Las distintas plantillas disponibles para dispositivos móviles y tabletas se ilustran en la Figura 4.1.

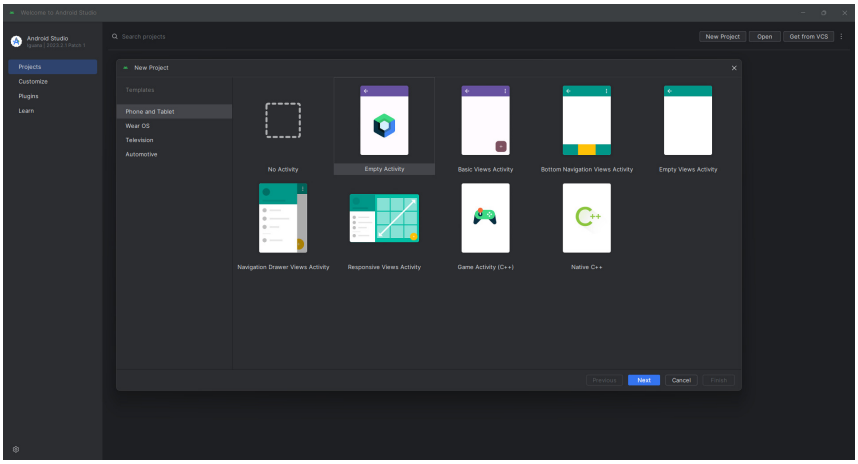

Figura 4.1: Plantillas de Android Studio.

Posteriormente, se deben definir diversos parámetros del proyecto, tales como el nombre de la aplicación, el nombre del paquete y la ubicación local donde se almacenarán

los archivos del proyecto. Asimismo, es necesario especificar la versión mínima del SDK de Android requerida para ejecutar la aplicación. En este caso, se seleccionó la API 27, correspondiente a la versión 8.1 del sistema operativo Android. Esta elección permite que la aplicación sea compatible con aproximadamente el 96.4 % de los dispositivos Android disponibles en el mercado, tal como se observa en la Figura 4.2.

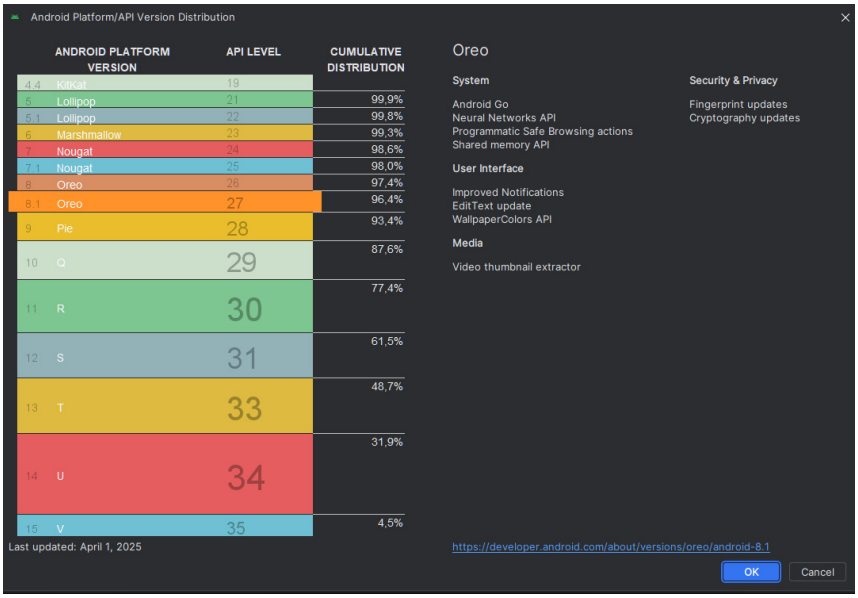

Figura 4.2: Distribución de versiones de Android.

4.2.1.2. Gestión de versiones y flujo de trabajo

Una vez creado y configurado el proyecto, se procedió a establecer un repositorio en GitHub. Tal como se describió en la Sección 3.1.1, GitHub ofrece numerosas ventajas al gestionar proyectos de software, especialmente en entornos de desarrollo colaborativo. Un componente fundamental en este contexto son las ramas (branches), ya que cada una constituye una versión independiente del código base. De esta forma, al incorporar una funcionalidad significativa, se crea una rama específica para ella, lo que permite conservar la versión principal como respaldo en caso de que surjan incidencias y, simultáneamente, facilita que distintos miembros del equipo trabajen en paralelo en funcionalidades diversas.

La estrategia inicial consistió en desarrollar una primera versión de la aplicación sin la integración directa con las Smart Glasses; en su lugar, se empleó la cámara del dispositivo móvil para la captura de imágenes. Esta decisión se fundamentó en dos motivos principales. Primero, se buscó simplificar el desarrollo en entornos remotos: de esta manera, cualquier modificación pudo evaluarse sin la necesidad de desplazarse a la facultad para probar la aplicación con las Smart Glasses. Segundo, esta aproximación permitió centrar los esfuerzos, en una primera fase, en asegurar el correcto funcionamiento del modelo de inteligencia artificial encargado de la detección y clasificación de poses, sin preocuparse por la complejidad de la integración de hardware.

Tras completar y validar exhaustivamente esta versión inicial de la aplicación, se procedió a la integración de las Smart Glasses. Este proceso se describirá con detalle en la Sección 4.4.

4.2.2. Integración de ML Kit

ML Kit es una biblioteca desarrollada por Google que ofrece diversas funcionalidades de aprendizaje automático específicas para aplicaciones móviles (véase Sección 2.4.2.1). Al estar optimizada para dispositivos móviles, su integración resulta sencilla y permite ejecutar el procesamiento directamente en el dispositivo, a diferencia de otras soluciones basadas en la nube. Este enfoque en el dispositivo mejora la velocidad de respuesta, habilita casos de uso en tiempo real—como el análisis de secuencias de cámara—y garantiza operación sin conexión, lo que convierte a ML Kit en una opción idónea para nuestro proyecto.

Entre las funcionalidades que ofrece ML Kit se incluyen el escaneo de códigos de barras, la detección de rostros, el reconocimiento de texto, el etiquetado de imágenes y el seguimiento de objetos, entre otras. La Figura 4.3 muestra un resumen de estas APIs de visión. Para nuestro desarrollo hemos seleccionado la API de detección de poses, la cual detecta en tiempo real la postura corporal a partir de un stream de vídeo o de imágenes estáticas.

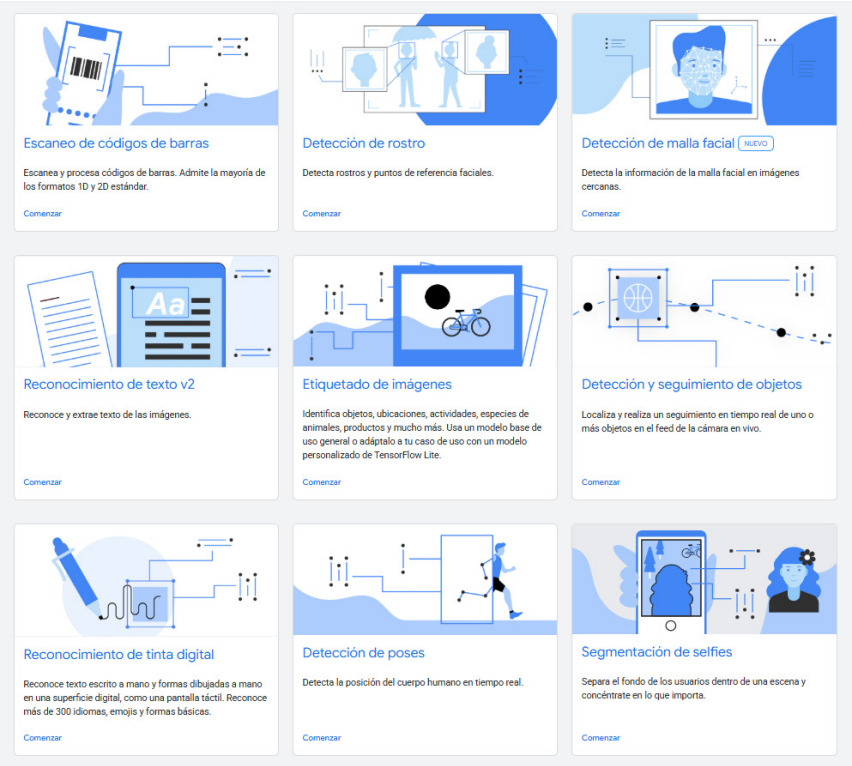

Figura 4.3: APIs de visión disponibles en ML Kit.

La API de detección de poses genera una malla esquelética compuesta por 33 puntos de referencia repartidos por todo el cuerpo, incluyendo puntos faciales (ojos, nariz, orejas y boca) y articulaciones de manos y pies. En la Figura 4.4 se ilustran los puntos clave que captura el modelo. Esta funcionalidad, aunque aún se encuentra en fase Beta, ofrece resultados estables y precisos, por lo que su uso no presentará inconvenientes para los objetivos del proyecto.

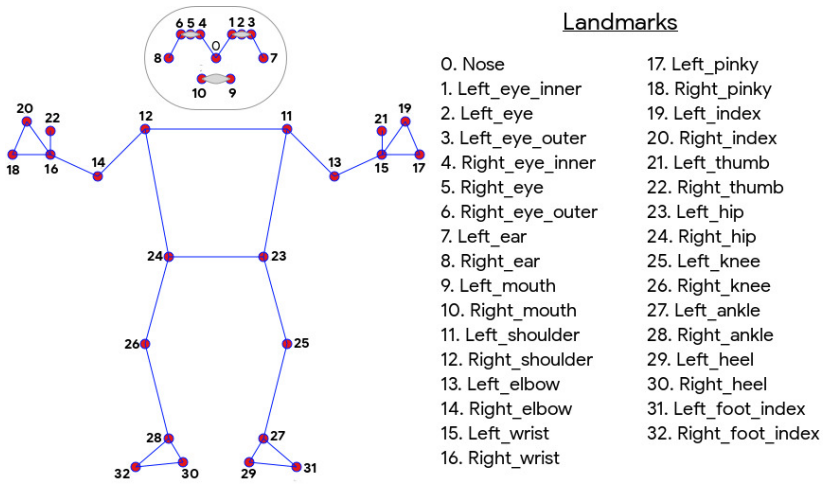

Figura 4.4: Puntos clave detectados por la API de detección de poses de ML Kit.

4.2.2.1. SDK y configuración de hardware

ML Kit pone a disposición dos variantes de SDK optimizados para la detección de poses, cada uno con características de rendimiento y precisión diferenciadas. El SDK base opera en tiempo real en dispositivos modernos, alcanzando tasas de alrededor de 30 FPS y 45 FPS, respectivamente, si bien con una ligera variabilidad en la exactitud de los puntos de referencia. En contraste, el SDK preciso ofrece una mayor fidelidad en las coordenadas de cada punto, a costa de una tasa de fotogramas inferior.

Inicialmente, se eligió el SDK base para garantizar la compatibilidad y el correcto funcionamiento de la aplicación en cualquier dispositivo de prueba. Posteriormente, una vez integrado el modelo de inteligencia artificial —el componente de mayor impacto en el rendimiento general—, se evaluó el SDK preciso en distintos escenarios y terminales. Los resultados demostraron una mejora sustancial en la exactitud de la detección de poses, con un decremento en el rendimiento prácticamente imperceptible. Por consiguiente, se adoptó el SDK preciso en la versión definitiva de la aplicación.

Adicionalmente, ML Kit admite varias configuraciones de hardware para optimizar el rendimiento de la inferencia:

- CPU: ejecución exclusiva sobre la unidad de procesamiento central.
- CPU + GPU: uso combinado de CPU y unidad de procesamiento gráfico.

La API setPreferredHardwareConfigs permite especificar las configuraciones preferentes al compilar el detector. Por defecto, todas las opciones están marcadas como preferidas, de modo que ML Kit evalúa, en tiempo de ejecución y sin bloquear la interfaz de usuario, la disponibilidad, estabilidad, exactitud y latencia de cada alternativa, eligiendo la óptima. Si ninguna resulta aplicable, recurre automáticamente a la configuración de CPU como opción de respaldo. Dado que se pretende delegar por completo la selección de hardware a ML Kit, no se realiza ninguna llamada explícita a dicha API.

4.2.2.2. Dependencias y setup en Android Studio

La integración de ML Kit en Android Studio se simplifica mediante la incorporación de una única dependencia en el archivo build.gradle a nivel de proyecto. Asimismo, es

necesario asegurar la inclusión del repositorio Maven de Google tanto en las secciones buildscript como allprojects.

Una vez configuradas estas secciones, se puede instanciar en el código la clase PoseDetector, que expone dos modos de detección:

STREAM_MODE:

- Detecta inicialmente a la persona más prominente en el fotograma y, a continuación, aplica la detección de poses.
- En los fotogramas posteriores, se omite la etapa de detección de la persona a menos que esta se oculte o ya no se detecte con alta confianza, realizando únicamente el seguimiento de la pose.
- Reduce la latencia y suaviza la detección, siendo idóneo para transmisiones de vídeo en tiempo real.

SINGLE_IMAGE_MODE:

- Ejecuta tanto la detección de persona como la de poses en cada imagen de forma independiente.
- Aumenta la latencia y no efectúa seguimiento continuo, por lo que resulta más adecuado para el análisis de imágenes estáticas o escenarios en los que no se requiera seguimiento.

En nuestro flujo de trabajo, se empleará SINGLE_IMAGE_MODE para el procesamiento de capturas estáticas y STREAM_MODE cuando la aplicación reciba un flujo de vídeo en tiempo real.

4.2.2.3. Captura de imagen con CameraX

Para proporcionar al usuario la capacidad de capturar imágenes y vídeo desde su dispositivo, se empleó la biblioteca CameraX (Google LLC, 2025b) de Android. En primer lugar, es necesario solicitar los permisos correspondientes de cámara y de lectura de almacenamiento, que en nuestro caso permiten tanto la captura de fotografías como la selección de archivos locales. (De requerirse captura de audio o escritura de archivos, se deberían añadir las solicitudes de permiso adicionales.)

En la fase inicial del desarrollo implementamos únicamente la captura de imágenes estáticas: añadimos una vista previa en tiempo real de la cámara y un botón para disparar la fotografía. Una vez validado este flujo, incorporamos el procesamiento continuo mediante la clase ImageAnalysis de CameraX, que recibe cada fotograma como una unidad independiente, permitiendo tratarlo de la misma manera que una imagen capturada manualmente.

Para que ML Kit pueda procesar una imagen, ésta debe representarse como un objeto de la clase InputImage. Dicha instancia puede generarse a partir de diferentes fuentes (por ejemplo, Bitmap, media.Image, ByteBuffer, array de bytes o fichero). Dado que CameraX nos provee fotogramas de tipo media.Image, la conversión a InputImage resultó directa. Asimismo, tras realizar varias pruebas en distintos dispositivos, comprobamos que la resolución por defecto de la cámara ofrecía un compromiso óptimo entre calidad y rendimiento, por lo que no fue necesario ajustar este parámetro.

Una vez integrado ML Kit en la aplicación, cada imagen procesada que contiene una persona devuelve un objeto Pose, que incluye una lista de 33 PoseLandmark (puntos clave corporales). Cada PoseLandmark aporta un identificador único, su tipo (por ejemplo, LEFT_SHOULDER, RIGHT_ELBOW o RIGHT_WRIST), las coordenadas X e Y en píxeles y el valor InFrameLikelihood, que indica la probabilidad de que el punto esté efectivamente dentro del fotograma. Aunque ML Kit ofrece de forma experimental una coordenada Z (perpendicular al eje de visión de la cámara, con origen entre las caderas), en nuestro caso de uso únicamente se emplearon las coordenadas X e Y.

En la Figura 4.5 se aprecian los 33 puntos de referencia superpuestos sobre una imagen real.

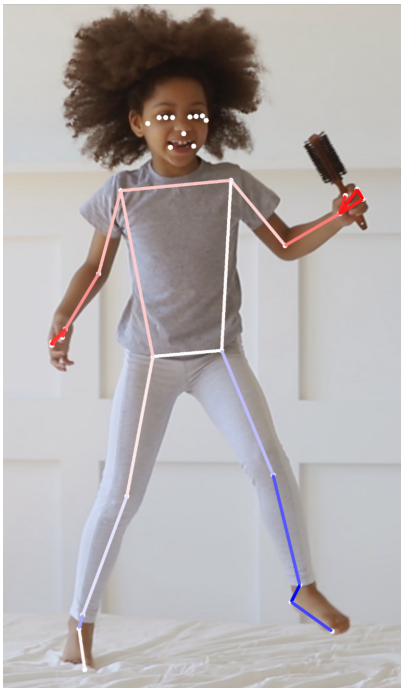

Figura 4.5: Representación gráfica de los 33 puntos clave detectados por ML Kit.

4.3. Creación y entrenamiento de modelos de IA

Este apartado describe el proceso completo de desarrollo del modelo de inteligencia artificial, desde la obtención y preparación del conjunto de datos hasta la evaluación final del modelo entrenado.

4.3.1. Extracción de datos y generación del dataset inicial

El conjunto de datos constituye una parte fundamental de cualquier proyecto de IA, dado que la calidad y la diversidad de sus instancias determinan en gran medida el desempeño del modelo entrenado.

4.3.1.1. Selección de poses y formato de datos

El primer objetivo consistió en definir la estructura del dataset. Se determinó emplear como características de entrada las coordenadas X e Y de los 33 puntos clave detectados por ML Kit. A continuación, fue preciso seleccionar las posturas que el sistema debía

reconocer: tras revisar diversas fuentes y discutir distintas propuestas, se acordó partir con ocho poses fundamentales —postura neutral, brazos abiertos, brazos cruzados, manos en la cintura, saludo con la mano derecha, saludo con la mano izquierda, señalamientos a la derecha y a la izquierda—. Aunque el prototipo definitivo requerirá un mayor número de categorías, esta selección inicial facilita la experimentación y permite una futura expansión del conjunto de clases sin alterar la arquitectura básica del sistema.

A continuación, se exploraron repositorios de datasets públicos (Kaggle, Hugging Face y GitHub) en busca de colecciones preexistentes que incluyeran coordenadas de puntos clave ya clasificadas según las posturas de interés. Si bien se localizaron conjuntos de datos de yoga y otros deportes con esquemas de puntos esqueléticos, ninguno se ajustaba a las ocho poses definidas. La inexistencia de un dataset adecuado motivó la decisión de generar uno propio, capturando imágenes de cada postura y procesándolas con ML Kit para extraer automáticamente las coordenadas de los 33 landmarks.

Se valoró la posibilidad de recopilar imágenes mediante búsquedas automáticas en Google Images, pero la heterogeneidad y el bajo ratio de relevancia de las fotografías obtenidas habrían exigido una revisión manual exhaustiva, lo cual se consideró inviable dada la escala requerida.

4.3.1.2. Captura de vídeos y extracción de fotogramas

Concluimos que la forma más eficiente de obtener una gran cantidad de imágenes para cada una de las poses era grabar vídeos en los que los propios miembros del grupo recrearan dichas posturas. Cada uno de nosotros grabó un vídeo de entre uno y dos minutos por cada pose, variando intencionadamente el ángulo y la distancia con respecto a la cámara, así como introduciendo variaciones en la ejecución de la pose. Esto permitía capturar una mayor diversidad de datos, mejorando así la capacidad de generalización del modelo. Como el objetivo era extraer únicamente las coordenadas de los puntos clave mediante ML Kit, factores como el fondo o la ropa no deberían tener un impacto relevante en los datos obtenidos.

Para procesar estos vídeos creamos una versión específica de la aplicación móvil desarrollada, diseñada exclusivamente para capturar y generar datos para el entrenamiento. Esta versión permitía seleccionar la pose a grabar, la duración del vídeo y la cámara a utilizar (frontal o trasera). Una vez configuradas estas opciones, se mostraba una cuenta atrás de tres segundos antes de iniciar la grabación. Esta cuenta atrás resultó especialmente útil para minimizar la aparición de datos ruidosos, ya que ofrecía tiempo suficiente para colocarse en la pose seleccionada antes de comenzar la grabación real. Así evitamos la necesidad de recortar manualmente los vídeos, facilitando el procesamiento automatizado posterior.

Finalizada la grabación, la aplicación permitía previsualizar el vídeo y ofrecía un botón para generar el dataset. Al pulsarlo, se extraían todos los fotogramas del vídeo, que se procesaban automáticamente con ML Kit para obtener las coordenadas X e Y de los 33 puntos clave en cada uno. Cada conjunto de coordenadas se almacenaba como una fila en un archivo CSV, junto con una columna adicional que indicaba mediante un identificador numérico la clase de pose correspondiente. En total, el archivo resultante contenía 67 columnas: 66 correspondientes a las coordenadas de los puntos (33 pares X,Y) y una con la etiqueta de la clase.

Además, para facilitar la escalabilidad de esta herramienta y ampliar las posibilidades de recopilación de datos, añadimos la opción de subir vídeos previamente grabados desde fuera de la aplicación. Esto permitía trabajar con material externo sin necesidad de grabarlo

directamente desde la app. En la Figura 4.6 se muestra la interfaz gráfica de esta versión especializada de la aplicación.

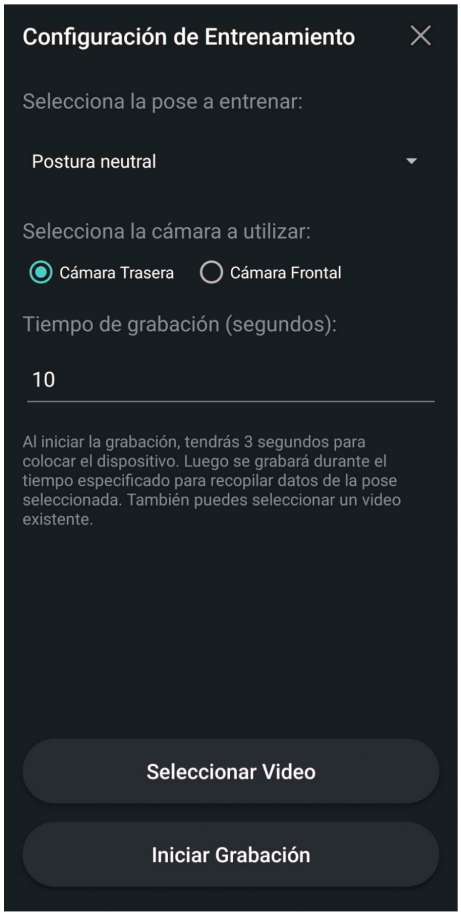

Figura 4.6: Interfaz de grabación de video.

4.3.1.3. Construcción del CSV y estadísticas básicas

Como parte del proceso de creación del dataset, cada miembro del grupo grabó un vídeo de entre uno y dos minutos para cada una de las ocho poses previamente definidas. Dado que la tasa de captura de fotogramas de los vídeos es de 30 frames por segundo, esto supone una obtención estimada de entre 5.400 y 10.800 muestras por pose. Tras procesar todos los vídeos utilizando la versión extendida de nuestra aplicación móvil, obtuvimos un total de 24 archivos en formato CSV, uno por cada vídeo grabado.

Para consolidar toda esta información en un único archivo de trabajo, desarrollamos un script en JavaScript. Este script no solo unifica el contenido de los 24 archivos, sino que además genera la cabecera del CSV final, incluyendo los nombres de todas las columnas, lo que facilitará el posterior entrenamiento del modelo. Las 66 primeras columnas corresponden a las coordenadas X e Y de los 33 puntos clave detectados por ML Kit, y la columna final representa la etiqueta de clase asociada a cada pose, en formato numérico. En la Figura 4.7 puede observarse un ejemplo del aspecto del archivo resultante con los datos estructurados y etiquetados.

	A	B	C	D	E	F	G	H	I	J	K	L
1	x0	y0	x1	y1	x2	y2	x3	y3	x4	y4	x5	y5
2	0.5067357	0.32679138	0.5193979	0.31536144	0.5293586	0.31525195	0.5360049	0.31530762	0.49517968	0.31587473	0.48875332	0.31591755
3	0.5052337	0.32513115	0.51783824	0.3145436	0.52759355	0.31457704	0.53393334	0.31467468	0.49310297	0.31517547	0.4868806	0.3154848
4	0.5044366	0.3253397	0.51698416	0.31450552	0.5267483	0.31430495	0.5332929	0.31422663	0.49179903	0.31558523	0.48566595	0.31597704
5	0.50340086	0.32728255	0.5158369	0.31593257	0.5243109	0.31594983	0.5311691	0.31584576	0.49166933	0.31654522	0.4849443	0.31681702
6	0.5027864	0.3265831	0.5156213	0.31480053	0.5244394	0.31475198	0.53138214	0.3146528	0.49112865	0.3152334	0.4842944	0.31537545
7	0.5015353	0.32777336	0.51510596	0.31604868	0.5238254	0.31600446	0.5308747	0.31587476	0.49057126	0.31658024	0.48351994	0.31676006
8	0.5002134	0.3276993	0.5136326	0.31563568	0.5233292	0.31564528	0.53012955	0.31565136	0.4887489	0.31595516	0.4820783	0.3159223
9	0.49997818	0.3287295	0.5124697	0.3166458	0.52200216	0.31642962	0.52882797	0.31615356	0.48821467	0.31703177	0.48128748	0.31697977
10	0.499499	0.3250764	0.5118406	0.3136008	0.52125496	0.3138158	0.5280068	0.31412247	0.48772514	0.3135304	0.48086858	0.31355917
11	0.4971026	0.32507223	0.51010925	0.3140958	0.5191749	0.31412965	0.52726567	0.3144143	0.48693702	0.31413537	0.47996494	0.3141173
12	0.49661368	0.32576713	0.5099378	0.31419843	0.5193035	0.314369	0.5267632	0.31480777	0.48638955	0.3140297	0.47922432	0.31403568
13	0.4982293	0.32462412	0.51085156	0.31381634	0.5200425	0.3139533	0.5274254	0.31413072	0.4876567	0.313749	0.4807823	0.31374517
14	0.49769017	0.3256963	0.5101976	0.31395102	0.51897764	0.31408384	0.52516526	0.31422138	0.48649755	0.31384346	0.47960967	0.31388214
15	0.4982536	0.32501048	0.5108959	0.31341827	0.51945716	0.3137759	0.52562666	0.31401998	0.4866292	0.3130837	0.47930688	0.31314442
16	0.4943938	0.32468522	0.5070398	0.31319416	0.51625264	0.31330442	0.5230336	0.3134413	0.48331898	0.31309432	0.47656375	0.31306025
17	0.49481788	0.326191	0.5073494	0.31399384	0.5158505	0.31397083	0.5222157	0.31395444	0.48368394	0.31426844	0.4761201	0.31437412
18	0.49547765	0.32353312	0.5078523	0.31205064	0.51609105	0.3123544	0.5226385	0.3125362	0.4835524	0.31196174	0.47600132	0.31216547
19	0.4912561	0.32386723	0.5041651	0.31295675	0.5134104	0.313052	0.52019095	0.3131622	0.4800814	0.31283447	0.47312644	0.31276625
20	0.4914067	0.32345226	0.50415087	0.31265587	0.51328015	0.3126926	0.5200289	0.3127116	0.47996503	0.3125177	0.4728439	0.31247035
21	0.4910315	0.32354495	0.5041493	0.3126601	0.5134172	0.31299055	0.5201522	0.31333977	0.47991863	0.31197298	0.47272643	0.3117253
22	0.48962143	0.32336184	0.502629	0.3123213	0.51125175	0.31269825	0.51765	0.31298125	0.4786732	0.31179708	0.4713801	0.3117442
23	0.4875547	0.32409096	0.5009549	0.31225485	0.5091878	0.31253433	0.5160558	0.3127426	0.47700998	0.31204262	0.4691937	0.31205642
24	0.48944223	0.32246283	0.5028368	0.31190574	0.51057106	0.3119392	0.5179025	0.31210268	0.47869584	0.31165507	0.47035676	0.31153324
25	0.49119258	0.32186893	0.50458467	0.31207654	0.5129972	0.3121334	0.5197391	0.31236103	0.47995132	0.31148311	0.471928	0.31117725

Figura 4.7: Ejemplo del dataset.

Además de realizar la unificación del dataset, decidimos aprovechar el script para calcular y mostrar estadísticas básicas sobre los datos recopilados. Estas estadísticas incluyen el número total de filas del archivo combinado, el número de ejemplos por cada una de las ocho clases, así como los valores máximos, mínimos y la media aritmética de todas las coordenadas del conjunto. Este análisis nos permitió verificar que la distribución de clases era razonablemente equilibrada y que no existían valores anómalos que pudieran deberse a errores en la captura o el procesamiento. Este tipo de control de calidad es esencial en cualquier proyecto de aprendizaje automático, ya que garantiza que el modelo no se entrene con datos sesgados o corruptos, lo cual podría comprometer su rendimiento y su capacidad de generalización.

El resultado de este proceso fue un único archivo CSV que contenía un total de 55.557 instancias, listas para ser utilizadas en la fase de entrenamiento. En la Figura 4.8 se muestran las estadísticas calculadas automáticamente por el script.

```
Estadísticas de los datos:
Número menor: -0.021503448
Número mayor: 1.1400898
Media: 0.4852

Estadísticas de filas:
Número total de filas: 55557

Número de filas por pose:
- Manos en la cintura: 7369 filas
- Saludo con la derecha: 6691 filas
- Postura neutral: 6376 filas
- Señalando a la izquierda: 6525 filas
- Brazos abiertos: 7365 filas
- Brazos cruzados: 7207 filas
- Saludo con la izquierda: 6645 filas
- Señalando a la derecha: 7379 filas
```

Figura 4.8: Estadísticas del dataset.

4.3.2. Diseño y entrenamiento de la red neuronal

Como se ha mencionado en apartados anteriores, todas las tareas relacionadas con el diseño y entrenamiento de modelos de inteligencia artificial se llevarán a cabo en la plataforma Google Colab. Esta herramienta resulta especialmente útil en el contexto académico

y de investigación, ya que permite desarrollar y ejecutar código Python directamente desde el navegador sin necesidad de instalar software adicional. Además, Colab proporciona acceso gratuito a recursos computacionales avanzados como GPUs y TPUs, lo cual es esencial para acelerar los procesos de entrenamiento en modelos de aprendizaje automático. Asimismo, ofrece facilidades para compartir notebooks, lo que mejora la colaboración entre distintos miembros del equipo.

Para implementar y entrenar nuestro modelo utilizaremos la biblioteca TensorFlow, una de las más utilizadas actualmente en el ámbito del aprendizaje profundo debido a su flexibilidad, escalabilidad y amplia documentación. Dado el tipo de problema que abordamos —la clasificación de poses humanas a partir de coordenadas— optamos inicialmente por una red neuronal densa (fully connected), que se adapta bien a problemas de clasificación con entradas numéricas de dimensión fija. En la Figura 4.9 se muestra un esquema representativo de esta arquitectura.

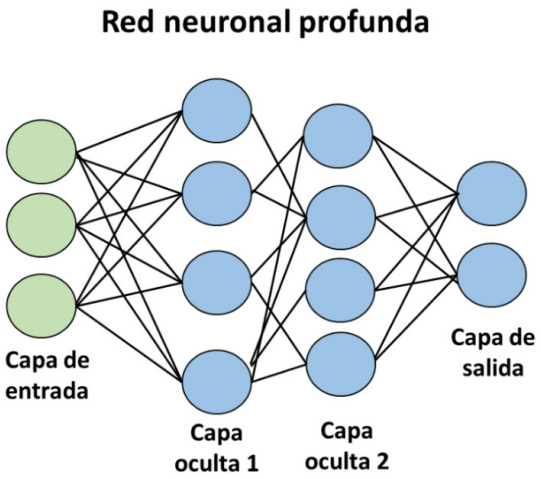

Figura 4.9: Diagrama de una red neuronal secuencial.

El primer paso en el proceso de entrenamiento fue la preparación del dataset. Importamos el archivo CSV final previamente generado y, a partir de él, realizamos una separación en dos subconjuntos: uno con los valores de la columna correspondiente a las etiquetas de clase (las poses), y otro con las coordenadas X e Y de los 33 puntos clave, es decir, las variables independientes que constituirán la entrada al modelo.

Para dividir el dataset en conjuntos de entrenamiento y validación utilizamos la función train_test_split de la biblioteca sklearn. Esta función admite varios parámetros, siendo los más relevantes en nuestro caso los siguientes: los dos primeros son los subconjuntos de entrada (X) y etiquetas (y), respectivamente; el tercer parámetro (test_size) determina el porcentaje de los datos que se destinarán a validación —establecido en un 20 %, que es el valor estándar más habitual—; el cuarto parámetro (random_state) es una semilla para garantizar la reproducibilidad del particionado; y el quinto (stratify) permite estratificar según las etiquetas, asegurando que la proporción de clases se mantenga constante en los dos subconjuntos. Este último parámetro es especialmente importante en problemas de clasificación para evitar un desequilibrio entre clases que pudiera distorsionar los resultados del modelo.

La correcta separación entre entrenamiento y validación es crucial para poder evaluar con precisión el rendimiento del modelo. Una vez entrenada la red neuronal con el 80 % de los datos, se utilizará el 20 % restante para evaluar su capacidad de generalización. El

porcentaje de aciertos sobre este subconjunto se tomará como medida objetiva de precisión (accuracy), lo cual permitirá comparar distintos modelos y configuraciones a lo largo del proceso de experimentación.

4.3.2.1. Arquitectura secuencial en TensorFlow

Una vez preparados los datos y divididos en conjuntos de entrenamiento y validación, el siguiente paso es definir la arquitectura del modelo de red neuronal. Para ello, emplearemos la API secuencial de TensorFlow, que permite construir redes capa a capa de forma sencilla y ordenada. Este tipo de arquitectura es ideal para problemas donde los datos fluyen de manera unidireccional desde la entrada hasta la salida sin necesidad de recurrir a estructuras más complejas como bifurcaciones o ciclos.

En nuestro caso, diseñamos una red neuronal feedforward que consta de una capa de entrada (definida implícitamente mediante el parámetro input_shape), seguida de tres capas densas (fully connected): las dos primeras capas densas son las capas ocultas y la última capa densa corresponde a la capa de salida. En la Figura 4.9 se muestra la estructura de la red neuronal extraída con la herramienta Netron (Roeder, 2025), un visualizador de modelos de redes neuronales, deep learning y machine learning.

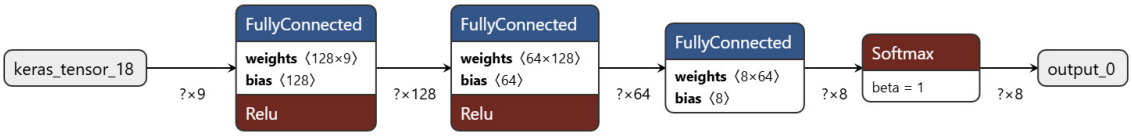

Figura 4.10: Estructura de nuestra red neuronal.

La capa de entrada no añade pesos entrenables, sino que simplemente establece que cada muestra del dataset tiene 66 valores (las coordenadas X e Y de los 33 puntos clave). A continuación, la primera capa oculta es una capa densa de 128 neuronas y la segunda capa oculta es una capa densa de 64 neuronas. La elección del número de neuronas en las capas ocultas es un proceso empírico, ya que no existe una fórmula exacta que garantice el número óptimo. Habitualmente se prueban distintas configuraciones hasta encontrar aquella que ofrezca el mejor equilibrio entre rendimiento y complejidad.

Tanto la primera como la segunda capa oculta utilizan la función de activación ReLU (Rectified Linear Unit), una de las funciones más utilizadas en redes neuronales profundas. Una función de activación es una función matemática que, aplicada a la suma ponderada de las entradas y un sesgo, calcula la salida de una neurona. Decide si la neurona "se dispara" en función de si su entrada supera cierto umbral. Introduce no linealidad al modelo, permitiendo que la red aprenda patrones complejos. La función ReLU, cuya gráfica puede observarse en la Figura 4.11, devuelve 0 si el valor de entrada es negativo y el propio valor si es positivo. Su simplicidad computacional es una de sus principales ventajas, ya que evita operaciones costosas como las exponenciales. Además, promueve activaciones dispersas al desactivar una parte significativa de las neuronas, lo que reduce el riesgo de sobreajuste, mejora la eficiencia del entrenamiento y acelera la convergencia.

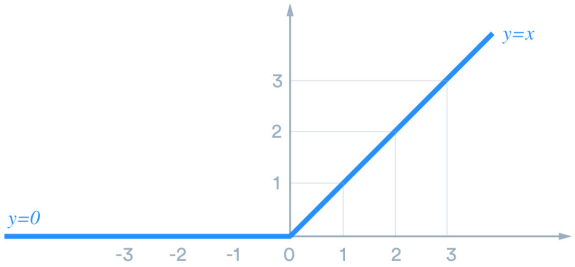

Figura 4.11: Gráfica de la función ReLU.

La tercera y última capa, es decir, la capa de salida, debe tener una neurona por cada clase del problema —ocho en total, una por cada pose a clasificar— y utiliza la función de activación Softmax. Esta función convierte los valores de salida en probabilidades, transformando el vector de salida en una distribución de probabilidad en la que todos los valores están entre 0 y 1 y su suma total es igual a 1. De este modo, la salida del modelo puede interpretarse directamente como el nivel de confianza en cada una de las posibles clases. En la Figura 4.12 se muestra la representación gráfica de la función Softmax.

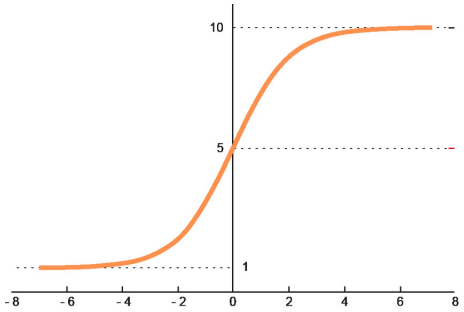

Figura 4.12: Gráfica de la función Softmax.

Con esta arquitectura secuencial sencilla, pero eficaz, pretendemos establecer una base sólida para el entrenamiento inicial del modelo. Posteriormente, en caso de ser necesario, se podrá experimentar con arquitecturas más complejas o realizar ajustes en los hiperparámetros con el fin de mejorar el rendimiento.

4.3.2.2. Parámetros de entrenamiento

Una vez definida la arquitectura de la red, el siguiente paso fue configurar los parámetros necesarios para el proceso de entrenamiento del modelo. En primer lugar seleccionamos el optimizador, que es un algoritmo encargado de ajustar iterativamente los pesos de la red neuronal con el fin de minimizar la función de coste. Optamos por el optimizador Adam (Adaptive Moment Estimation) (Kingma y Ba, 2014), uno de los más utilizados en el campo del aprendizaje profundo debido a su rendimiento eficiente y su capacidad de adaptación durante el entrenamiento. Adam combina lo mejor de dos algoritmos previos: Momentum, que acelera el aprendizaje al considerar el promedio de los gradientes anteriores, y RMSProp, que ajusta dinámicamente la tasa de aprendizaje en función de la magnitud reciente de los gradientes. Esto permite realizar actualizaciones de pesos más estables y efectivas, sobre todo en problemas complejos como el nuestro.

A continuación se definió la función de pérdida, un criterio que cuantifica la discrepancia entre las predicciones del modelo y los valores reales, y cuyo valor guía el ajuste de los pesos mediante el proceso de backpropagation: el error calculado en la salida se propaga

hacia atrás por las capas para actualizar los parámetros y minimizar la función de coste. Para nuestro problema de clasificación multiclase con etiquetas codificadas como enteros, seleccionamos SparseCategoricalCrossentropy, que compara la distribución de probabilidad resultante del modelo (gracias a la función Softmax de la capa de salida) con la clase real y penaliza de forma apropiada las salidas probabilísticas, a diferencia de funciones como MeanSquaredError, pensadas para regresión.

Seguidamente, fijamos el índice de aprendizaje o learning rate —es decir, el tamaño de los pasos con que se actualizan los pesos en cada iteración— en 0.0001. Aunque un valor bajo ralentiza el entrenamiento, favorece una convergencia más precisa al evitar saltos excesivamente grandes en la función de pérdida, reduciendo el riesgo de pasar por alto el mínimo óptimo o que la función de error diverja, haciendo el proceso más controlado y robusto.

El entrenamiento del modelo se configuró para ejecutarse durante 100 épocas (epochs), tras observar en ensayos preliminares que, pasada esta cantidad, la mejora en el rendimiento era mínima y comenzaban a evidenciarse signos de sobreajuste ("overfitting"), es decir, una pérdida de capacidad de generalización debido a la memorización excesiva de los datos de entrenamiento.

Establecimos la precisión (accuracy) como la métrica principal para evaluar el desempeño del modelo. Esta métrica calcula el porcentaje de predicciones correctas, lo que la convierte en un indicador directo y fácilmente interpretable de la eficacia del clasificador tanto en el conjunto de entrenamiento como en el de validación. No obstante, es importante tener en cuenta que, según la naturaleza del problema y el coste relativo de los distintos tipos de error, pueden resultar más adecuadas otras métricas —por ejemplo, el F1-score o el F2-score en contextos donde sea preferible penalizar con mayor severidad los falsos negativos (como en la detección de enfermedades), o la precisión y el recall por separado cuando interese controlar por separado la tasa de falsos positivos o falsos negativos. En nuestro caso, dado que las clases estaban balanceadas y el coste de un falso positivo y un falso negativo se consideró equiparable, la elección de la precisión nos proporcionó una evaluación sencilla y suficiente para comparar el rendimiento entre los distintos experimentos.

Finalmente, para el tamaño del lote (batch size), optamos por un valor de 32. Este valor es una elección común que equilibra bien el uso de recursos computacionales y la estabilidad del aprendizaje. Batches más pequeños pueden generar ruido en las actualizaciones de los pesos, mientras que batches demasiado grandes requieren más memoria y pueden ralentizar el entrenamiento en dispositivos con recursos limitados, como ocurre frecuentemente en entornos como Google Colab.

4.3.2.3. Resultados iniciales

Con todos los parámetros definidos, procedimos a entrenar nuestro modelo y analizar los primeros resultados obtenidos. Tal y como se puede observar en la Figura 4.13, el proceso de entrenamiento tuvo una duración total de 8 minutos y alcanzó una precisión (accuracy) final del 99,94 %. Para profundizar en su comportamiento, presentamos en la Figura 4.14 la evolución de la precisión por época—donde el eje horizontal representa el número de épocas y el eje vertical el porcentaje de acierto— y en la Figura 4.15 la curva de la función de pérdida (loss), que muestra cómo disminuye el valor de la pérdida calculada en cada iteración. Ambas curvas revelan que, a partir de la época 15, la precisión supera el 99 % y la pérdida se estabiliza casi en cero, un indicio claro de que el modelo deja de aprender patrones generales y se ajusta excesivamente a los datos de entrenamiento.

```
Epoch 90/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9991 - loss: 0.0050 - val_accuracy: 0.9996 - val_loss: 0.0026
Epoch 91/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9985 - loss: 0.0062 - val_accuracy: 0.9993 - val_loss: 0.0039
Epoch 92/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9990 - loss: 0.0053 - val_accuracy: 0.9996 - val_loss: 0.0026
Epoch 93/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9990 - loss: 0.0048 - val_accuracy: 0.9995 - val_loss: 0.0033
Epoch 94/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9987 - loss: 0.0053 - val_accuracy: 0.9996 - val_loss: 0.0027
Epoch 95/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9987 - loss: 0.0053 - val_accuracy: 0.9996 - val_loss: 0.0037
Epoch 96/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9990 - loss: 0.0042 - val_accuracy: 0.9996 - val_loss: 0.0026
Epoch 97/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9992 - loss: 0.0040 - val_accuracy: 0.9975 - val_loss: 0.0084
Epoch 98/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9994 - loss: 0.0038 - val_accuracy: 0.9996 - val_loss: 0.0035
Epoch 99/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9992 - loss: 0.0040 - val_accuracy: 0.9995 - val_loss: 0.0033
Epoch 100/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9988 - loss: 0.0046 - val_accuracy: 0.9996 - val_loss: 0.0022
Duración del entrenamiento: 477.77 segundos
Accuracy:
348/348 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9994 - loss: 0.0025
[0.0022295459639281034, 0.9995500445365906]
```

Figura 4.13: Entrenamiento del modelo.

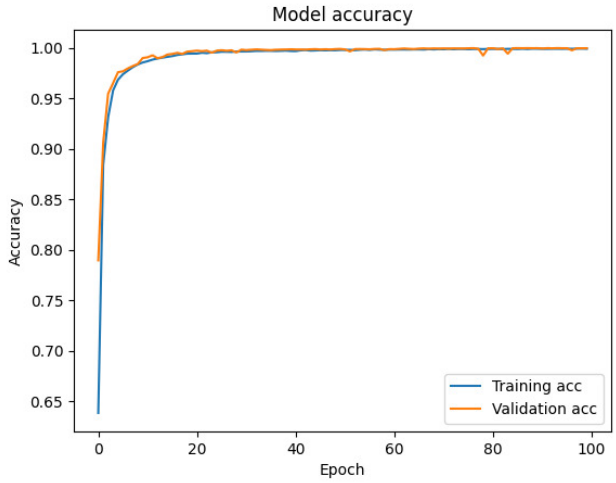

Figura 4.14: Gráfica de precisión.

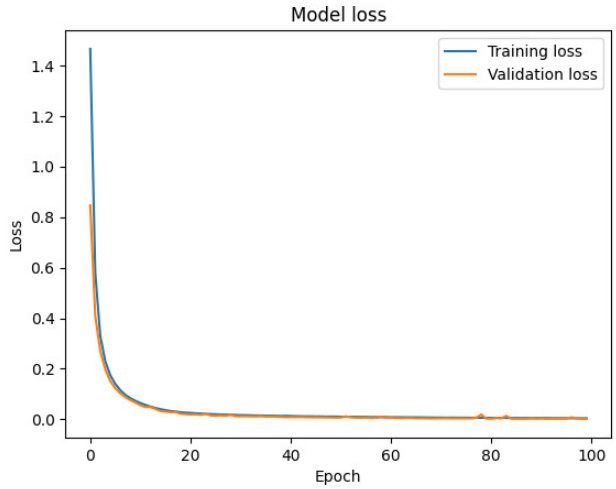

Figura 4.15: Gráfica de pérdida.

Para corroborar esta sospecha, examinamos la matriz de confusión (Figura 4.16), que resume en una tabla las predicciones frente a las etiquetas reales: cada fila corresponde a la clase verdadera y cada columna a la clase predicha, de modo que los valores en la diagonal principal indican aciertos y los valores fuera de ella, errores de clasificación. En nuestro caso, la matriz muestra prácticamente solo valores en la diagonal y casi ningún cruce, lo cual confirma un rendimiento demasiado perfecto y refuerza la hipótesis de sobreajuste.

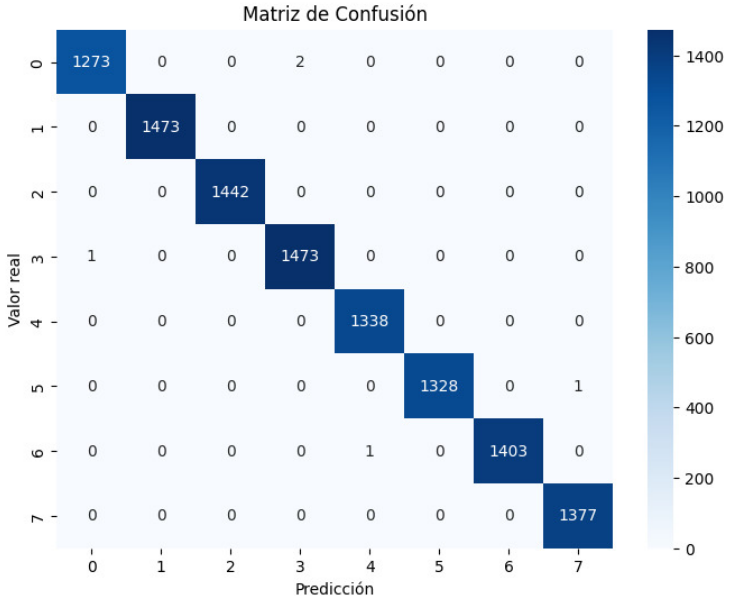

Figura 4.16: Matriz de confusión.

Una posible explicación sería que nuestro problema fuese muy sencillo y, por tanto, el modelo fuese capaz de aprenderlo en muy pocas iteraciones. Este sería el caso, por ejemplo, de una red que clasifica figuras geométricas simples y bien diferenciadas. Sin embargo, dada la naturaleza de nuestro proyecto, que implica reconocer poses humanas con variaciones en distancia, ángulo y postura, descartamos esta posibilidad. Otro problema común en estos casos es la fuga de datos (data leakage), que ocurre cuando la información del conjunto de validación aparece, directa o indirectamente, en el conjunto de entrenamiento, afectando así a la evaluación del modelo. Sin embargo, nuestra división de datos se realizó utilizando la función train_test_split de la librería sklearn, aplicando la estratificación adecuada y asegurando una separación correcta. Por tanto, tampoco parecía ser esa la causa del comportamiento observado.

Ante esta situación, decidimos probar una solución habitual para evitar el sobreajuste: introducir capas Dropout (Srivastava et al., 2014) en nuestra red neuronal. Estas capas anulan aleatoriamente un porcentaje de neuronas durante el entrenamiento, forzando al modelo a aprender representaciones más robustas y generalizables. Añadimos dos capas de este tipo: una entre la primera y la segunda capa oculta y otra entre la segunda capa oculta y la capa de salida. La primera capa de Dropout tiene una tasa de abandono (dropout rate) de 0.3, es decir, desactiva el 30 % de sus neuronas en cada iteración. La segunda tiene una tasa de 0.2. Tras reentrenar el modelo con esta arquitectura mejorada, obtuvimos una precisión final de un 99,65 %, como se puede ver en la Figura 4.17. Aunque esta nueva versión de la red muestra un leve descenso en la precisión, el cambio no fue lo suficientemente significativo como para resolver por completo el problema detectado.

```
Epoch 90/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9810 - loss: 0.0625 - val_accuracy: 0.9956 - val_loss: 0.0129
Epoch 91/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9820 - loss: 0.0620 - val_accuracy: 0.9960 - val_loss: 0.0119
Epoch 92/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9818 - loss: 0.0618 - val_accuracy: 0.9959 - val_loss: 0.0129
Epoch 93/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9814 - loss: 0.0612 - val_accuracy: 0.9960 - val_loss: 0.0125
Epoch 94/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9815 - loss: 0.0591 - val_accuracy: 0.9960 - val_loss: 0.0117
Epoch 95/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9828 - loss: 0.0593 - val_accuracy: 0.9964 - val_loss: 0.0114
Epoch 96/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9825 - loss: 0.0604 - val_accuracy: 0.9965 - val_loss: 0.0117
Epoch 97/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9828 - loss: 0.0592 - val_accuracy: 0.9961 - val_loss: 0.0117
Epoch 98/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9839 - loss: 0.0543 - val_accuracy: 0.9959 - val_loss: 0.0107
Epoch 99/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9825 - loss: 0.0592 - val_accuracy: 0.9964 - val_loss: 0.0106
Epoch 100/100
1389/1389 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9824 - loss: 0.0561 - val_accuracy: 0.9966 - val_loss: 0.0115
Duración del entrenamiento: 552.31 segundos
Accuracy:
348/348 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9965 - loss: 0.0113
[0.011488579213619232, 0.9965803027153015]
```

Figura 4.17: Entrenamiento del modelo con capas Dropout.

Finalmente, tras una revisión exhaustiva, llegamos a la conclusión de que el origen del problema estaba en el propio dataset. Al extraer los datos directamente de los vídeos, obtenemos muchísimos ejemplos muy similares entre sí. Esto se debe a que entre dos fotogramas consecutivos de un vídeo apenas hay movimiento, por lo que la información aportada por cada nuevo dato es mínima. En consecuencia, el modelo tiende a memorizar fácilmente los patrones del dataset, en lugar de aprender características generalizables. Esto explica la alta precisión en entrenamiento y validación, pero también la baja expectativa de rendimiento en datos nuevos. En la siguiente sección explicaremos cómo optimizamos el dataset para reducir estos problemas y mejorar la capacidad de generalización del modelo.

4.3.3. Optimización del dataset

Tras observar los resultados del entrenamiento inicial del modelo y comprobar su comportamiento en la aplicación móvil, consideramos que la forma más efectiva de mejorar su rendimiento era optimizando el propio dataset. Como ya se explicó, el modelo parecía estar aprendiendo demasiado rápido, lo cual, lejos de ser una ventaja, podía estar indicando una falta de variedad o exceso de datos redundantes. Por ello, comenzamos a estudiar posibles mejoras en la calidad y representatividad de los datos utilizados.

4.3.3.1. Reducción de frames

La primera decisión que tomamos fue reducir el tamaño del dataset eliminando información redundante. En su versión original, el conjunto de datos se generó extrayendo todos los frames de los vídeos grabados para cada pose. Teniendo en cuenta que los vídeos estaban grabados a una tasa de 30 frames por segundo, esta metodología producía una gran cantidad de datos prácticamente idénticos entre sí, ya que entre fotogramas consecutivos apenas existía variación en la pose, lo que dificultaba la generalización del modelo y favorecía el sobreajuste.

Para solucionar este problema, decidimos realizar un muestreo más espaciado de los vídeos, seleccionando únicamente un frame de cada seis. Esto significa que, en lugar de extraer 30 frames por segundo, pasamos a procesar sólo 5. Esta reducción permitió eliminar datos repetitivos y centrarnos en ejemplos más representativos de cada pose. Como resultado, el tamaño total del dataset se redujo considerablemente, pasando de 55.557 filas a 9.260. En la Figura 4.18 se muestran las estadísticas actualizadas del nuevo dataset.

```
Estadísticas de los datos:
Número menor: -0.016979128
Número mayor: 1.0996788
Media: 0.485209354627739

Estadísticas de filas:
Número total de filas: 9260

Número de filas por pose:
- Postura neutral: 1061 filas
- Brazos abiertos: 1227 filas
- Brazos cruzados: 1202 filas
- Manos en la cintura: 1228 filas
- Saludo con la derecha: 1116 filas
- Saludo con la izquierda: 1107 filas
- Señalando a la derecha: 1231 filas
- Señalando a la izquierda: 1088 filas
```

Figura 4.18: Estadísticas del dataset reducido.

Una vez generada esta nueva versión optimizada del conjunto de datos, entrenamos nuevamente el modelo utilizando la misma arquitectura y parámetros definidos anteriormente. Tal y como se muestra en la Figura 4.19, el proceso de entrenamiento fue significativamente más rápido, reduciéndose de 8 minutos a tan solo 2 minutos. Además, la precisión alcanzada en este nuevo entrenamiento fue del 99,19 %, cifra ligeramente inferior a la de versiones anteriores. Esta mejora se atribuye a la eliminación de datos redundantes que no aportaban valor real al aprendizaje del modelo, lo que ha favorecido una mayor capacidad de generalización. No obstante, dicho porcentaje sigue siendo excesivamente alto y poco representativo del rendimiento real del sistema, por lo que seguiremos aplicando estrategias de optimización.

```
Epoch 90/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9929 - loss: 0.0373 - val_accuracy: 0.9924 - val_loss: 0.0338
Epoch 91/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9904 - loss: 0.0405 - val_accuracy: 0.9914 - val_loss: 0.0353
Epoch 92/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9882 - loss: 0.0396 - val_accuracy: 0.9924 - val_loss: 0.0327
Epoch 93/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9880 - loss: 0.0445 - val_accuracy: 0.9930 - val_loss: 0.0318
Epoch 94/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9929 - loss: 0.0334 - val_accuracy: 0.9935 - val_loss: 0.0329
Epoch 95/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9899 - loss: 0.0370 - val_accuracy: 0.9941 - val_loss: 0.0322
Epoch 96/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9929 - loss: 0.0342 - val_accuracy: 0.9930 - val_loss: 0.0305
Epoch 97/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9907 - loss: 0.0359 - val_accuracy: 0.9930 - val_loss: 0.0318
Epoch 98/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9923 - loss: 0.0330 - val_accuracy: 0.9924 - val_loss: 0.0301
Epoch 99/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9908 - loss: 0.0362 - val_accuracy: 0.9930 - val_loss: 0.0296
Epoch 100/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9926 - loss: 0.0311 - val_accuracy: 0.9924 - val_loss: 0.0330
Duración del entrenamiento: 118.53 segundos
Accuracy:
58/58 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9919 - loss: 0.0303
[0.03302225470542908, 0.9924405813217163]
```

Figura 4.19: Entrenamiento del modelo usando dataset reducido.

Este primer paso en la optimización del dataset demostró que no siempre es beneficioso disponer de una gran cantidad de datos, especialmente si estos son muy similares entre sí. En las siguientes secciones exploraremos más estrategias para seguir mejorando la calidad del dataset y, con ello, el rendimiento del modelo.

4.3.3.2. Empleo de ángulos en lugar de coordenadas

La segunda estrategia que aplicamos para optimizar el dataset consistió en modificar completamente el tipo de datos que se usaban como entrada para el modelo. Inicialmente, cada muestra estaba compuesta por las coordenadas X e Y de los 33 puntos clave detectados en el cuerpo humano por ML Kit, lo cual generaba un total de 66 valores por muestra, más una columna adicional con la etiqueta de clase (la pose). Sin embargo, este enfoque presentaba varios inconvenientes: era muy sensible a factores externos como la altura o complexión de la persona, la distancia a la cámara o el encuadre del plano, lo cual dificultaba la generalización del modelo.

Para solucionar estos problemas decidimos sustituir las coordenadas por un conjunto reducido de características más abstractas y representativas: los ángulos formados por determinadas articulaciones del cuerpo. En lugar de usar todas las coordenadas, seleccionamos 8 ángulos significativos que capturan de manera efectiva la postura del cuerpo. Cada ángulo se calcula a partir de tres puntos clave, siendo el punto medio el vértice del ángulo. Para calcularlos implementamos una función personalizada llamada getAngle, que realiza el siguiente procedimiento:

1. Se definen dos vectores a partir del punto medio: uno desde el punto medio hacia el primer punto, y otro hacia el tercer punto.
2. Se calcula el ángulo de cada vector respecto al eje X mediante la función atan2(y, x), que devuelve un valor en radianes entre $-\pi$ y π.
3. Se obtiene la diferencia entre ambos ángulos, convirtiéndola a grados.
4. Se toma el valor absoluto de esta diferencia para eliminar signos negativos.
5. Si el resultado supera los 180°, se realiza la operación 360 - ángulo para obtener siempre el ángulo más pequeño (menor o igual a 180°), garantizando así que se devuelva el ángulo interior agudo entre los tres puntos.

Además, normalizamos estos valores para que se encuentren en el rango [0, 1], simplemente dividiendo cada ángulo por 180. Tal y como se ilustra en la Figura 4.20, los ángulos seleccionados fueron los siguientes:

- LeftShoulderAngle: formado por el codo izquierdo, el hombro izquierdo y la cadera izquierda.
- RightShoulderAngle: formado por el codo derecho, el hombro derecho y la cadera derecha.
- LeftElbowAngle: formado por la muñeca izquierda, el codo izquierdo y el hombro izquierdo.
- RightElbowAngle: formado por la muñeca derecha, el codo derecho y el hombro derecho.
- LeftHipAngle: formado por el hombro izquierdo, la cadera izquierda y la rodilla izquierda.
- RightHipAngle: formado por el hombro derecho, la cadera derecha y la rodilla derecha.

- LeftKneeAngle: formado por la cadera izquierda, la rodilla izquierda y el tobillo izquierdo.
- RightKneeAngle: formado por la cadera derecha, la rodilla derecha y el tobillo derecho.

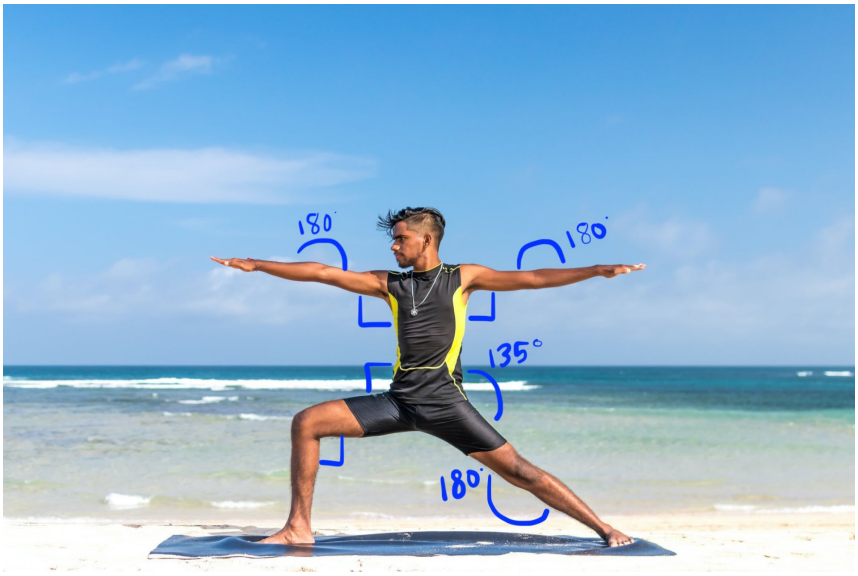

Figura 4.20: Ángulos clave.

Esta transformación del dataset presenta varias ventajas claras. En primer lugar, reduce significativamente la dimensionalidad de los datos de entrada, pasando de 67 columnas a solo 9 (8 ángulos más la etiqueta), lo que simplifica la red neuronal y acelera su entrenamiento. En segundo lugar, elimina la dependencia del modelo respecto a la escala, posición o proporciones físicas del usuario, permitiendo una generalización mucho más efectiva. En la Figura 4.21 se puede ver un ejemplo del nuevo dataset generado con ángulos como entrada.

	A	B	C	D	E	F	G	H	I
1	LeftShoulderAngle	RightShoulderAngle	LeftElbowAngle	RightElbowAngle	LeftHipAngle	RightHipAngle	LeftKneeAngle	RightKneeAngle	Pose
2	0.18316711	0.19379522	0.38349906	0.35120976	0.8668595	0.90357363	0.9816197	0.9897136	2
3	0.18997659	0.22065946	0.38635918	0.31494513	0.87666345	0.91709876	0.99720657	0.9784158	2
4	0.19123891	0.21017191	0.35819656	0.3296785	0.8764669	0.9104662	0.9550249	0.9669526	2
5	0.21447076	0.18754175	0.32983857	0.34804067	0.8680019	0.9307137	0.9922228	0.9830228	2
6	0.29869735	0.043419704	0.23970017	0.50720876	0.8531865	0.9211496	0.9620153	0.9987621	2
7	0.26640755	0.0057463143	0.27376688	0.53106713	0.8743659	0.8968163	0.96075743	0.997115	2
8	0.3283769	0.09600074	0.22110012	0.62777823	0.8671254	0.91134524	0.962077	0.9778979	2
9	0.33804756	0.1685452	0.21655782	0.71215886	0.85628676	0.9311979	0.92339945	0.9465491	2
10	0.33204153	0.1633824	0.19761612	0.715822	0.862016	0.9543041	0.95778155	0.9407845	2
11	0.3293402	0.16035028	0.20395476	0.701128	0.85078585	0.92501265	0.9130871	0.9956179	2
12	0.28602517	0.10204022	0.25660095	0.6303166	0.8929145	0.93153363	0.99468756	0.9586425	2
13	0.2675161	0.047362544	0.297458	0.5689061	0.8362967	0.9273517	0.8853477	0.9496895	2
14	0.2210996	0.019944228	0.31729206	0.5130767	0.8651278	0.90363336	0.9406013	0.9803762	2
15	0.20928407	0.059807982	0.33278617	0.473248	0.86012167	0.9170538	0.94655854	0.9964475	2
16	0.21021417	0.15685329	0.3408936	0.37438965	0.86492795	0.9569032	0.9769185	0.9503321	2
17	0.1492336	0.23499729	0.40843198	0.2971915	0.91633207	0.9115587	0.99026847	0.9530121	2
18	0.122167416	0.27874854	0.68392444	0.23666215	0.91380614	0.8738938	0.9875794	0.92015094	2
19	0.11285278	0.26432103	0.7135803	0.22845058	0.89976466	0.87376785	0.9895138	0.9267279	2
20	0.14881259	0.27522236	0.7381361	0.21497245	0.89674103	0.8939899	0.94627106	0.9206152	2
21	0.2036665	0.29727644	0.76524633	0.20133084	0.91968375	0.9259349	0.99101603	0.8844187	2
22	0.16650414	0.29026017	0.735723	0.24084102	0.95740604	0.90001655	0.9748879	0.90557235	2
23	0.15340456	0.2980083	0.73030955	0.24110323	0.99371505	0.9163477	0.97750914	0.9093093	2
24	0.119406655	0.2726181	0.7381704	0.25860775	0.9498173	0.9420748	0.9764308	0.9572099	2
25	0.04691729	0.25668824	0.65859926	0.27285278	0.93381965	0.9093991	0.9651342	0.95642066	2

Figura 4.21: Ejemplo del dataset con 8 ángulos normalizados.

Entrenamos de nuevo el modelo con este nuevo conjunto de datos, manteniendo la arquitectura secuencial y los parámetros de entrenamiento previamente establecidos. Como se muestra en la Figura 4.22, el proceso de entrenamiento fue ligeramente más rápido (1 minuto y 50 segundos) gracias a la menor complejidad de los datos.

```
Epoch 90/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9170 - loss: 1.2711 - val_accuracy: 0.9266 - val_loss: 1.2683
Epoch 91/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9181 - loss: 1.2596 - val_accuracy: 0.9282 - val_loss: 1.2562
Epoch 92/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9089 - loss: 1.2560 - val_accuracy: 0.9298 - val_loss: 1.2442
Epoch 93/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9208 - loss: 1.2377 - val_accuracy: 0.9325 - val_loss: 1.2322
Epoch 94/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9279 - loss: 1.2266 - val_accuracy: 0.9336 - val_loss: 1.2201
Epoch 95/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9216 - loss: 1.2132 - val_accuracy: 0.9352 - val_loss: 1.2081
Epoch 96/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9219 - loss: 1.2032 - val_accuracy: 0.9384 - val_loss: 1.1962
Epoch 97/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9307 - loss: 1.1930 - val_accuracy: 0.9406 - val_loss: 1.1844
Epoch 98/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9284 - loss: 1.1775 - val_accuracy: 0.9411 - val_loss: 1.1727
Epoch 99/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9373 - loss: 1.1591 - val_accuracy: 0.9444 - val_loss: 1.1610
Epoch 100/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9341 - loss: 1.1541 - val_accuracy: 0.9455 - val_loss: 1.1494
Duración del entrenamiento: 110.38 segundos
Accuracy:
58/58 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9442 - loss: 1.1546
[1.1494194269180298, 0.9454643726348877]
```

Figura 4.22: Entrenamiento del modelo usando dataset con ángulos.

La precisión final alcanzada fue del 94,42 %, lo cual, si bien es inferior a los valores anteriores, resulta mucho más realista y se aleja de los síntomas de sobreajuste detectados previamente. En las gráficas asociadas al entrenamiento (Figura 4.23: precisión; Figura 4.24: pérdida; Figura 4.25: matriz de confusión) se observa un comportamiento coherente con lo esperado en problemas reales de clasificación. No obstante, destaca un patrón de confusión especialmente pronunciado en la zona central de la matriz de confusión, tal y como se aprecia en los valores fuera de la diagonal principal. Este comportamiento fue identificado como uno de los principales retos del modelo y se analizará en detalle en la Sección 4.3.4.4.

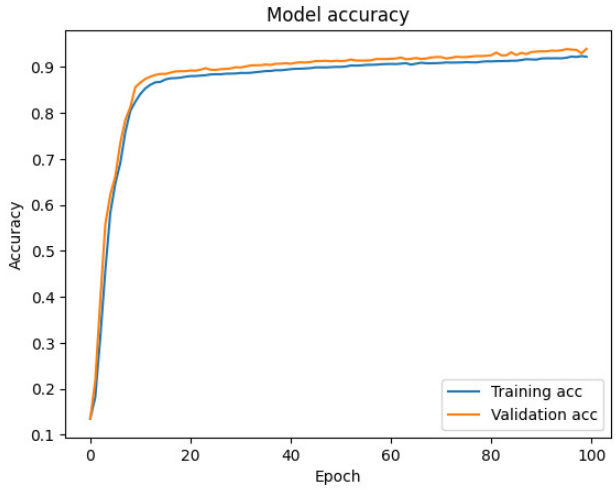

Figura 4.23: Gráfica de precisión usando dataset con ángulos.

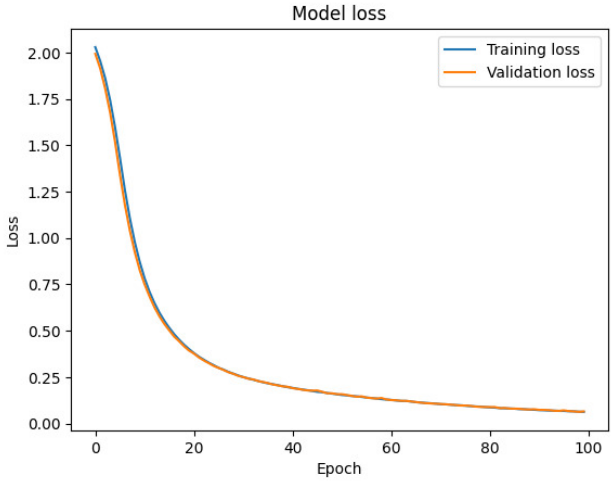

Figura 4.24: Gráfica de pérdida usando dataset con ángulos.

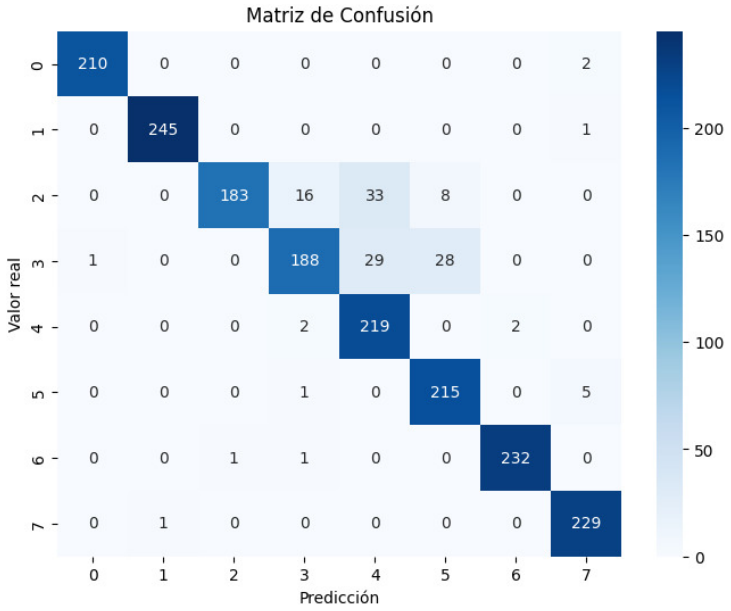

Figura 4.25: Matriz de confusión usando dataset con ángulos.

Más allá de los valores cuantitativos, esta versión del modelo demostró un comportamiento claramente superior al integrarse en la aplicación bajo condiciones reales de uso. En pruebas comparativas con versiones anteriores, observamos que las implementaciones previas tendían a "saltarse" detecciones cuando el usuario pasaba de una pose a otra de forma continua, arrojando resultados inconsistentes o retrasos perceptibles. Al adoptar el enfoque basado en ángulos, las predicciones se mantuvieron estables incluso frente a pequeñas fluctuaciones en los datos. Esta fluidez en la respuesta no solo mejoró la experiencia de usuario, sino que también reforzó nuestra confianza en emplear este método como base para futuras optimizaciones.

4.3.4. Random Forest y refinamiento del dataset

Una vez que obtuvimos un dataset optimizado y bien estructurado, decidimos explorar alternativas a las redes neuronales para comparar el rendimiento. Una de las opciones más prometedoras fue el uso del modelo Random Forest (Breiman, 2001), una técnica de aprendizaje supervisado ampliamente utilizada en tareas de clasificación por su robustez y simplicidad.

Un Random Forest está compuesto por un conjunto de árboles de decisión independientes, cada uno entrenado sobre una muestra aleatoria del conjunto de datos original. La idea principal es que, al combinar múltiples modelos débiles (árboles individuales), se obtiene un modelo fuerte con mejor capacidad de generalización y menor riesgo de sobreajuste. La Figura 4.26 ilustra la estructura básica de un Random Forest.

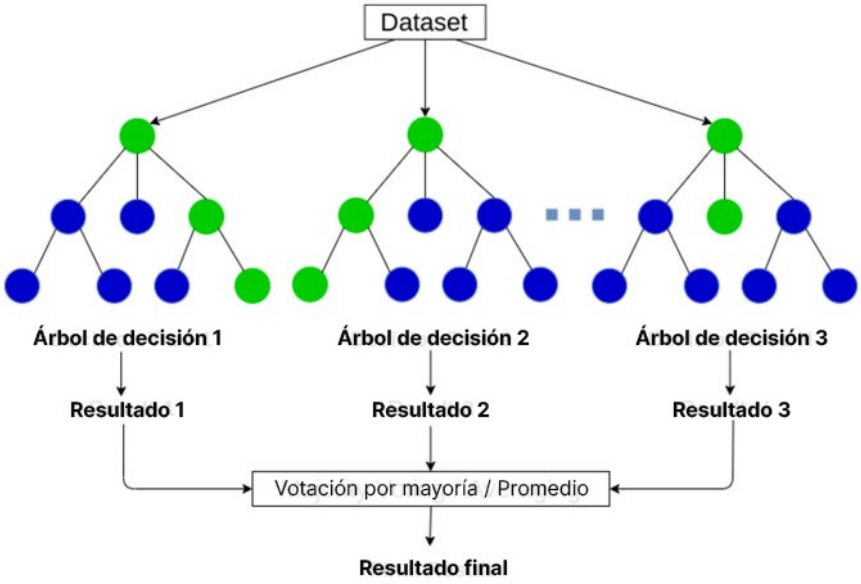

Figura 4.26: Diagrama de Random Forest.

El funcionamiento del algoritmo se puede resumir en los siguientes pasos:

1. Muestreo aleatorio con reemplazo: Para cada árbol, se genera una muestra aleatoria del dataset original. Este proceso, conocido como bootstrap, permite que algunas instancias se repitan y otras queden fuera.

2. Selección aleatoria de características: En cada división de un nodo del árbol, se considera únicamente un subconjunto aleatorio de las variables, no la totalidad. Esto introduce diversidad adicional entre los árboles.

3. Construcción de los árboles: Cada árbol se entrena de forma estándar, dividiendo los nodos según la variable y punto de corte que mejor separen las clases, hasta alcanzar una profundidad máxima o un número mínimo de muestras por hoja.

4. Votación agregada: Una vez entrenado el conjunto, cada árbol vota por una clase cuando se presenta una nueva instancia. La clase final se elige por mayoría de votos entre todos los árboles.

Este enfoque presenta diversas ventajas. Por un lado, tiene bajo coste computacional comparado con redes neuronales profundas, lo que lo hace más rápido de entrenar en datasets pequeños o medianos. También ofrece una cierta interpretabilidad, ya que es posible inspeccionar árboles individuales y evaluar la importancia relativa de cada variable para la clasificación. Sin embargo, su principal desventaja es que puede generar modelos muy grandes, con alta demanda de memoria si se utilizan demasiados árboles, lo cual puede ser un problema en entornos con recursos limitados.

Al aplicar esta técnica sobre nuestro dataset basado en ángulos, buscamos evaluar su rendimiento frente a la red neuronal previamente entrenada. Esta comparación nos permitió identificar las fortalezas de cada enfoque y sentó las bases para futuras decisiones de diseño en función del caso de uso, recursos disponibles y necesidades de precisión y generalización.

4.3.4.1. Selección de parámetros y entrenamiento

Para implementar el modelo de Random Forest, utilizamos también la librería TensorFlow, manteniendo un flujo de trabajo similar al empleado para la red neuronal. El conjunto de datos fue dividido en un 80 % para entrenamiento y un 20 % para validación.

Uno de los retos clave al trabajar con Random Forest es el ajuste de hiperparámetros, ya que un número excesivo de árboles o una profundidad demasiado grande puede generar modelos muy pesados, difíciles de manejar y propensos a consumir muchos recursos. Por esta razón, aplicamos la técnica de Grid Search, que consiste en explorar de forma sistemática distintas combinaciones de hiperparámetros para identificar la configuración óptima.

Las combinaciones evaluadas fueron las siguientes:

- Número de árboles (num_trees): 50, 100, 200, 500, 1000
- Profundidad máxima (max_depth): 5, 10, 15, 20, sin límite

Para cada combinación, medimos tanto la accuracy sobre el conjunto de validación como el tiempo de entrenamiento, con el objetivo de encontrar un equilibrio entre rendimiento predictivo y eficiencia computacional. Es importante destacar que, en muchas ocasiones, un modelo ligeramente menos preciso pero mucho más eficiente resulta preferible para aplicaciones prácticas.

Tras completar la búsqueda, seleccionamos como configuración óptima la siguiente:

- Número de árboles: 100
- Profundidad máxima: sin límite
- Accuracy obtenida: 98,81
- Tiempo de entrenamiento: 2,6 segundos

Esta configuración proporcionó un rendimiento competitivo sin requerir recursos excesivos. En la Figura 4.27 se muestra un ejemplo de uno de los árboles generados dentro del Random Forest.

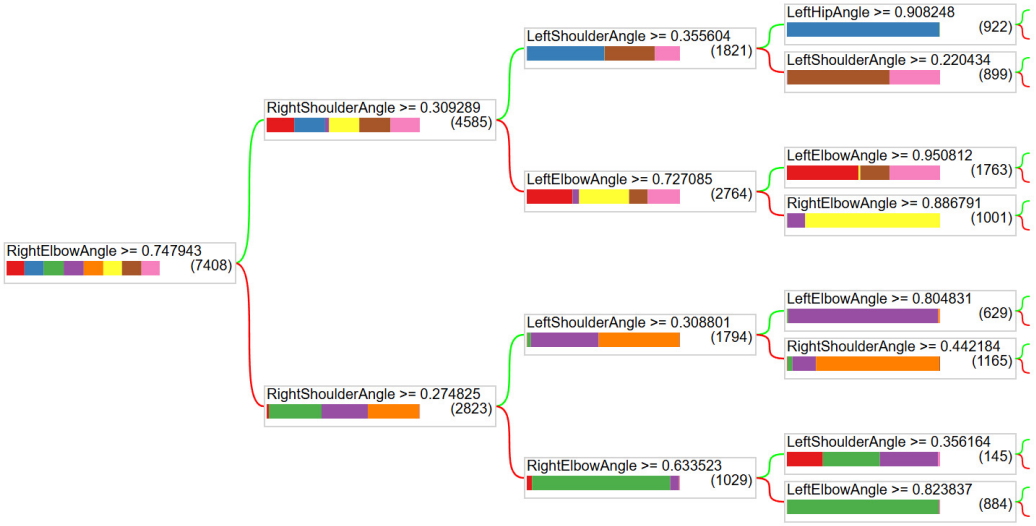

Figura 4.27: Ejemplo de árbol en el modelo Random Forest.

Es importante resaltar que el tiempo de entrenamiento fue notablemente inferior al del modelo de red neuronal. Esta diferencia se debe a la arquitectura del Random Forest, donde cada árbol se entrena de forma independiente y no es necesario realizar procesos iterativos complejos como la retropropagación. Además, los árboles pueden ser construidos en paralelo, lo que mejora significativamente la eficiencia. Este tipo de modelos es especialmente eficaz en tareas de clasificación sobre datos estructurados o tabulares, donde se puede alcanzar un alto rendimiento sin necesidad de arquitecturas profundas.

4.3.4.2. Importancia de variables

Aunque actualmente TensorFlow Decision Forests no permite exportar modelos en formato TFLite, lo cual impide su uso directo en nuestra aplicación móvil, el modelo de Random Forest nos resultó útil para obtener información valiosa sobre el dataset, en particular sobre la relevancia de las variables utilizadas.

Una de las principales ventajas de los modelos basados en árboles es que permiten evaluar la importancia de las variables mediante distintas métricas. En nuestro caso, analizamos dos de las más utilizadas:

- INV_MEAN_MIN_DEPTH: mide la profundidad promedio mínima a la que aparece una variable en los árboles del bosque. Variables que tienden a aparecer cerca de la raíz son más influyentes en la toma de decisiones.

- SUM_SCORE: cuantifica la mejora acumulada en la predicción cada vez que una variable es utilizada para dividir un nodo. Cuanto mayor sea este valor, mayor es su contribución al modelo.

Tras analizar los resultados de estas métricas (mostrados en las Figuras 4.28 y 4.29), observamos que 4 de los 8 ángulos utilizados tenían un impacto significativamente bajo en el rendimiento del modelo. Estos correspondían a los ángulos relacionados con la cintura y las piernas, lo que sugiere que podrían eliminarse en versiones futuras del dataset para simplificar el modelo sin pérdida sustancial de precisión.

```
Variable Importance: INV_MEAN_MIN_DEPTH:
    1.    "RightElbowAngle"  0.359765 ################
    2.     "LeftElbowAngle"  0.334880 ##############
    3. "RightShoulderAngle"  0.285359 ##########
    4.  "LeftShoulderAngle"  0.268758 #########
    5.       "LeftHipAngle"  0.141716 #
    6.      "RightHipAngle"  0.138496 #
    7.     "RightKneeAngle"  0.126300
    8.      "LeftKneeAngle"  0.122229
```

Figura 4.28: Métrica de importancia INV_MEAN_MIN_DEPTH.

```
Variable Importance: SUM_SCORE:
    1.    "RightElbowAngle" 770864.994574 ################
    2.     "LeftElbowAngle" 765405.409858 ###############
    3.  "LeftShoulderAngle" 696465.138838 ##############
    4. "RightShoulderAngle" 670059.430281 #############
    5.       "LeftHipAngle" 39412.521791
    6.      "RightHipAngle" 32669.739343
    7.     "RightKneeAngle" 15803.777495
    8.      "LeftKneeAngle" 7318.683071
```

Figura 4.29: Métrica de importancia SUM_SCORE.

Este análisis no solo nos permitió entender mejor la relevancia de cada variable, sino que además abrió la puerta a una futura simplificación del dataset, facilitando modelos más ligeros y eficientes.

4.3.4.3. Reducción de ángulos innecesarios

Tras analizar la importancia de variables obtenida con el modelo de Random Forest, observamos que los ángulos relacionados con la cintura y las piernas tenían una influencia muy reducida en el resultado. Esta observación tenía sentido, ya que las poses que pretendemos clasificar se caracterizan principalmente por la posición y orientación de la parte superior del cuerpo, mientras que la parte inferior tiende a permanecer más estática o uniforme entre clases.

Por tanto, optamos por eliminar del dataset los ángulos con bajo impacto, simplificándolo y reduciendo el posible ruido introducido por variables irrelevantes. Como se puede apreciar en la Figura 4.30, el nuevo dataset incluye únicamente los siguientes cuatro ángulos:

- LeftShoulderAngle
- RightShoulderAngle
- LeftElbowAngle
- RightElbowAngle

	A	B	C	D	E
1	LeftShoulderAngle	RightShoulderAngle	LeftElbowAngle	RightElbowAngle	Pose
2	0.55082524	0.55135274	0.9152346	0.9105839	1
3	0.56126744	0.55271125	0.922402	0.90870804	1
4	0.54905134	0.5365621	0.89008313	0.90863323	1
5	0.5597944	0.55563587	0.9042105	0.9199534	1
6	0.55689925	0.5546261	0.90528744	0.9041716	1
7	0.5510784	0.5389438	0.866421	0.892182	1
8	0.53866273	0.564832	0.8647327	0.8781064	1
9	0.55296874	0.55450433	0.86536634	0.84828705	1
10	0.5585368	0.5475297	0.844476	0.85199016	1
11	0.5550334	0.556252	0.854556	0.86794627	1
12	0.5760817	0.5368532	0.8565695	0.8705067	1
13	0.57802767	0.5441405	0.85551673	0.87001956	1
14	0.5791942	0.51630783	0.84104437	0.87302685	1

Figura 4.30: Ejemplo del dataset con 4 ángulos.

Con el nuevo conjunto de datos reentrenamos la red neuronal y los resultados, mostrados en la Figura 4.31, evidencian una mejora notable: el tiempo de entrenamiento se redujo a 1 minuto y 40 segundos y la precisión aumentó hasta el 96,07 %. Esto demuestra que, al eliminar datos ruidosos o redundantes, hemos logrado un modelo no solo más rápido, sino también más preciso.

```
Epoch 90/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9488 - loss: 0.2249 - val_accuracy: 0.9563 - val_loss: 0.2170
Epoch 91/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9481 - loss: 0.2284 - val_accuracy: 0.9563 - val_loss: 0.2133
Epoch 92/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9531 - loss: 0.2144 - val_accuracy: 0.9573 - val_loss: 0.2098
Epoch 93/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9525 - loss: 0.2191 - val_accuracy: 0.9573 - val_loss: 0.2063
Epoch 94/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9522 - loss: 0.2103 - val_accuracy: 0.9579 - val_loss: 0.2030
Epoch 95/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9557 - loss: 0.2036 - val_accuracy: 0.9579 - val_loss: 0.1999
Epoch 96/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9576 - loss: 0.1948 - val_accuracy: 0.9584 - val_loss: 0.1969
Epoch 97/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9555 - loss: 0.1987 - val_accuracy: 0.9584 - val_loss: 0.1940
Epoch 98/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9589 - loss: 0.1918 - val_accuracy: 0.9584 - val_loss: 0.1912
Epoch 99/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9530 - loss: 0.1983 - val_accuracy: 0.9584 - val_loss: 0.1884
Epoch 100/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9540 - loss: 0.1951 - val_accuracy: 0.9584 - val_loss: 0.1858
Duración del entrenamiento: 99.73 segundos
Accuracy:
58/58 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9607 - loss: 0.1759
[0.1858130842447281, 0.9584233164787292]
```

Figura 4.31: Entrenamiento del modelo usando dataset con 4 ángulos.

4.3.4.4. Refinamiento final: distancias clave

Aunque el modelo mostró una mejora considerable respecto a las versiones anteriores, persistían errores de clasificación concretos. En la parte central de la matriz de confusión (Figura 4.32) se observa que la pose real 3 (manos en la cintura) se confunde en varios casos con la pose 4 (saludo con la mano derecha) y la 5 (saludo con la mano izquierda).

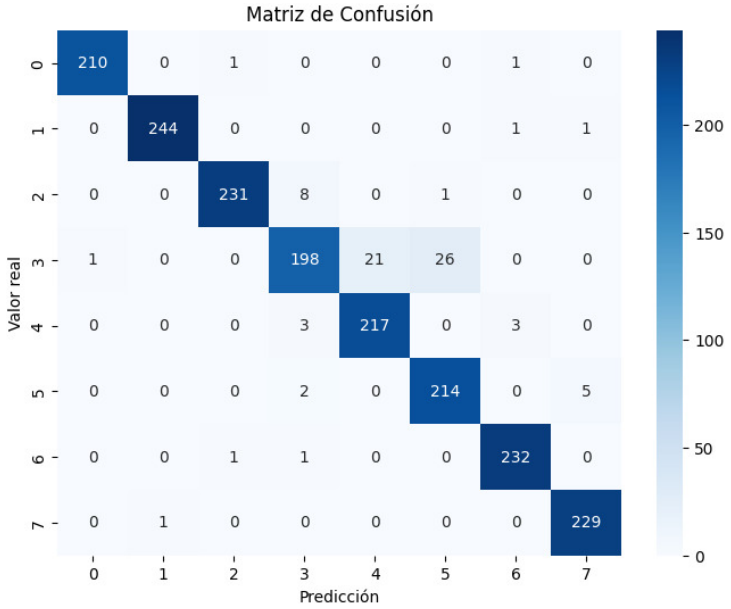

Figura 4.32: Matriz de confusión usando dataset con 4 ángulos.

Esto se debe a que, al calcular el ángulo formado por tres puntos y emplear siempre el menor de los posibles, los gestos de saludo y la postura con las manos en la cadera resultan muy parecidos desde el punto de vista angular, tal y como se aprecia en la Figura 4.33. Esto provoca fronteras de decisión difusas y errores frecuentes entre estas clases.

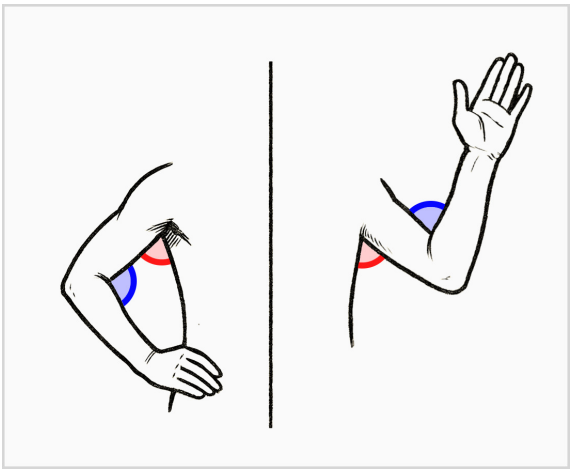

Figura 4.33: Visualización de ángulos similares.

Tras analizar la situación con los tutores del proyecto, consideramos que algunas distancias específicas entre puntos clave del cuerpo podrían aportar información adicional relevante. Estas distancias, al complementar los ángulos ya utilizados, permitirían al modelo reconocer mejor las relaciones espaciales características de ciertas poses.

Finalmente, decidimos incluir las siguientes cinco distancias, representadas en la Figura 4.34:

- Distancia entre la muñeca izquierda y la cabeza
- Distancia entre la muñeca derecha y la cabeza

- Distancia entre el codo izquierdo y la cadera izquierda
- Distancia entre el codo derecho y la cadera derecha
- Distancia entre ambas muñecas

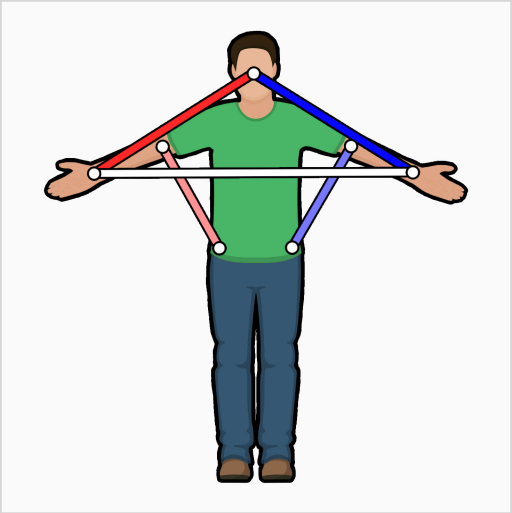

Figura 4.34: Distancias clave.

Al igual que con los ángulos, las distancias se normalizan para que sus valores queden en el rango [0, 1], garantizando que todas las características del dataset sean comparables y eviten dominar el entrenamiento. El proceso de normalización se realiza en dos pasos:

1. Escalado relativo al tamaño corporal.

   Para que las distancias sean independientes de la distancia al plano de la cámara o de la complexión de cada usuario, primero se calcula la distancia entre los hombros como factor de referencia. Todas las demás distancias (muñeca-cabeza, codo-cadera, muñeca-muñeca, etc.) se dividen entre esta distancia de hombros. De esta manera, si una persona está más cerca o más lejos de la cámara, o tiene una complexión diferente, las relaciones espaciales siguen siendo coherentes y comparables.

2. Compresión mediante función sigmoide

   Aunque el escalado lineal ya sitúa las distancias en un rango relativo, puede haber valores extremos que dificulten la convergencia del modelo. Por ello, a cada distancia ya escalada se le aplica una función sigmoide de la forma

   $$S(x) = \frac{1}{1 + e^{-k(x - x_0)}}$$

   con k=2.5 (que controla la pendiente de la curva) y x_0=1.0 (el punto medio, donde la salida es 0.5). Esta transformación "aplasta" los valores pequeños hacia 0 y los grandes hacia 1, con una transición suave alrededor de 1, como se observa en la Figura 4.35. Así se reduce aún más la influencia de valores atípicos y se consigue que todas las distancias normalizadas entren en el mismo rango, favoreciendo una convergencia más estable y rápida del entrenamiento.

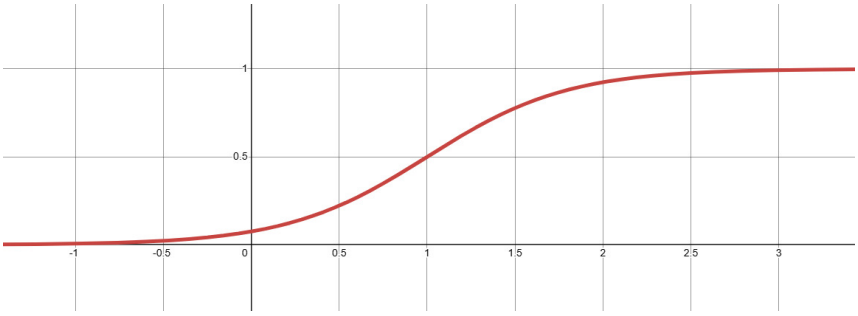

Figura 4.35: Gráfica de la función sigmoide.

En conjunto, este sistema de normalización —primero dividiendo por la distancia interhombros y luego aplicando la sigmoide— elimina efectos indeseables de escala y reduce la dispersión de los datos, garantizando que cada variable aporte de forma equilibrada al modelo.

En la Figura 4.36 se puede apreciar un ejemplo de esta versión del dataset, compuesto por 4 ángulos y 5 distancias.

	A	B	C	D	E	F	G	H	I	J
1	LeftShoulderAngle	RightShoulderAngle	LeftElbowAngle	RightElbowAngle	LeftWristToHeadDist	RightWristToHeadDi	LeftElbowToHipDist	RightElbowToHipDis	WristToWristDist	Pose
2	0.55082524	0.55135274	0.9152346	0.9105839	0.92635983	0.9058204	0.67445934	0.63632375	0.9992269	1
3	0.56126744	0.55271125	0.922402	0.90870804	0.92026496	0.90579027	0.6736438	0.62754744	0.99915516	1
4	0.54905134	0.5365621	0.89008313	0.90863323	0.91525275	0.91253304	0.6769387	0.6260434	0.9991761	1
5	0.5597944	0.55563587	0.9042105	0.9199534	0.91054934	0.90818936	0.6694119	0.6259607	0.9990883	1
6	0.55689925	0.5546261	0.90528744	0.9041716	0.9066172	0.89616615	0.6600824	0.62060595	0.99890566	1
7	0.5510784	0.5389438	0.866421	0.892182	0.90382457	0.8917786	0.67313206	0.6169868	0.99883586	1
8	0.53866273	0.564832	0.8647327	0.8781064	0.88559264	0.8735601	0.6514403	0.65732646	0.9983369	1
9	0.55296874	0.55450433	0.86536634	0.84828705	0.8647716	0.8468934	0.6643291	0.6441042	0.9975126	1
10	0.5585368	0.5475297	0.844476	0.85199016	0.86072725	0.8368826	0.6723425	0.63072854	0.9972199	1
11	0.5550334	0.556252	0.854556	0.86794627	0.865499	0.8477035	0.6681458	0.64541036	0.99754685	1
12	0.5760817	0.5368532	0.8565695	0.8705067	0.8933688	0.8608529	0.71471965	0.6271807	0.9983125	1
13	0.57802767	0.5441405	0.85551673	0.87001956	0.87693065	0.8475474	0.67835027	0.6075072	0.99779224	1
14	0.5791942	0.51630783	0.84104437	0.87302685	0.90723354	0.8666142	0.7334377	0.6141576	0.99862397	1
15	0.5643674	0.5178988	0.8218199	0.8624894	0.89290315	0.8421144	0.70534873	0.60519797	0.9980299	1
16	0.56874335	0.5355315	0.83869255	0.8735376	0.8833014	0.8474713	0.6747899	0.5938271	0.9979285	1
17	0.55009764	0.5457308	0.8504978	0.846259	0.87582374	0.8349799	0.6365181	0.59955007	0.9975672	1
18	0.54075664	0.5410437	0.8379767	0.8301841	0.87451875	0.84331656	0.6550907	0.6540427	0.9977049	1
19	0.5317648	0.55501413	0.8456477	0.8368138	0.8838751	0.86654884	0.65190256	0.6679867	0.99821293	1
20	0.5068864	0.5650357	0.8298201	0.81720674	0.8625621	0.8571866	0.6239974	0.69340694	0.99759185	1
21	0.49196362	0.5545999	0.8414204	0.79168004	0.8682936	0.868963	0.6255629	0.712384	0.997895	1
22	0.45955184	0.5728143	0.8537581	0.8061564	0.85808647	0.88441956	0.5705678	0.7659202	0.9979972	1
23	0.44399258	0.5657032	0.84185463	0.8014207	0.8573702	0.88899636	0.55639946	0.7533837	0.9980562	1
24	0.4340698	0.5689989	0.84380233	0.80728346	0.8697609	0.89095056	0.5636732	0.7809527	0.9982541	1
25	0.44321445	0.578715	0.8378246	0.78235847	0.87092113	0.8872531	0.54694086	0.76603127	0.99824226	1

Figura 4.36: Ejemplo del dataset con ángulos y distancias.

4.3.5. Resultados y evaluación del modelo final

Con este nuevo dataset, que incluye los 4 ángulos relevantes y las 5 distancias clave normalizadas, procedimos a entrenar nuevamente el modelo de red neuronal. El proceso de entrenamiento completo duró 1 minuto y 46 segundos, obteniendo una accuracy del 98,09 %, como puede observarse en la Figura 4.37.

```
Epoch 90/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9685 - loss: 0.1261 - val_accuracy: 0.9741 - val_loss: 0.1225
Epoch 91/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9722 - loss: 0.1161 - val_accuracy: 0.9746 - val_loss: 0.1202
Epoch 92/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9693 - loss: 0.1223 - val_accuracy: 0.9752 - val_loss: 0.1181
Epoch 93/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9710 - loss: 0.1172 - val_accuracy: 0.9773 - val_loss: 0.1158
Epoch 94/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9712 - loss: 0.1144 - val_accuracy: 0.9768 - val_loss: 0.1138
Epoch 95/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9750 - loss: 0.1110 - val_accuracy: 0.9773 - val_loss: 0.1116
Epoch 96/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9716 - loss: 0.1136 - val_accuracy: 0.9784 - val_loss: 0.1096
Epoch 97/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9725 - loss: 0.1109 - val_accuracy: 0.9779 - val_loss: 0.1076
Epoch 98/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9729 - loss: 0.1051 - val_accuracy: 0.9789 - val_loss: 0.1057
Epoch 99/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9732 - loss: 0.1055 - val_accuracy: 0.9784 - val_loss: 0.1039
Epoch 100/100
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9741 - loss: 0.1038 - val_accuracy: 0.9795 - val_loss: 0.1021
Duración del entrenamiento: 106.03 segundos
Accuracy:
58/58 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9809 - loss: 0.0992
[0.10208039730787277, 0.9794816374778748]
```

Figura 4.37: Entrenamiento del modelo usando dataset con ángulos y distancias.

Además del valor de accuracy, analizamos otros indicadores clave del rendimiento del modelo. La gráfica de precisión (Figura 4.38) y la gráfica de pérdida (Figura 4.39) mostraron un comportamiento estable y sin señales de sobreajuste, lo que sugiere que el modelo podría generalizar correctamente a nuevos datos.

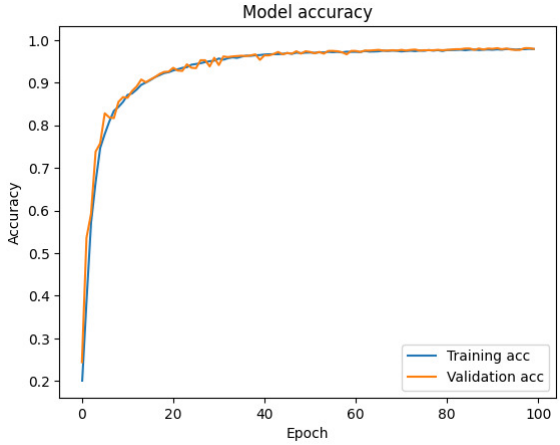

Figura 4.38: Gráfica de precisión usando dataset con ángulos y distancias.

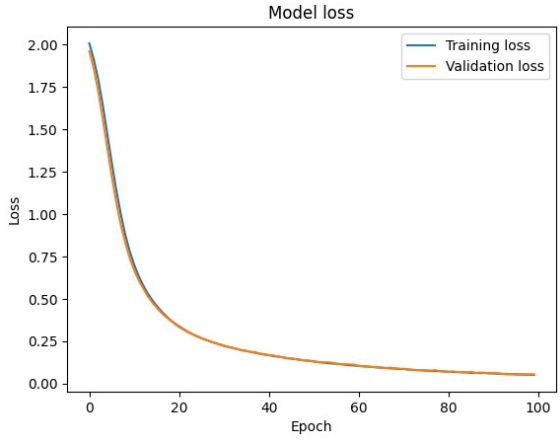

Figura 4.39: Gráfica de pérdida usando dataset con ángulos y distancias.

Como se aprecia en la matriz de confusión (Figura 4.40), tras aplicar las modificaciones en el conjunto de datos, las confusiones entre las poses 3, 4 y 5 se han reducido de manera significativa, prácticamente eliminándose, lo que indica que el modelo distingue correctamente estas posturas sin errores apreciables.

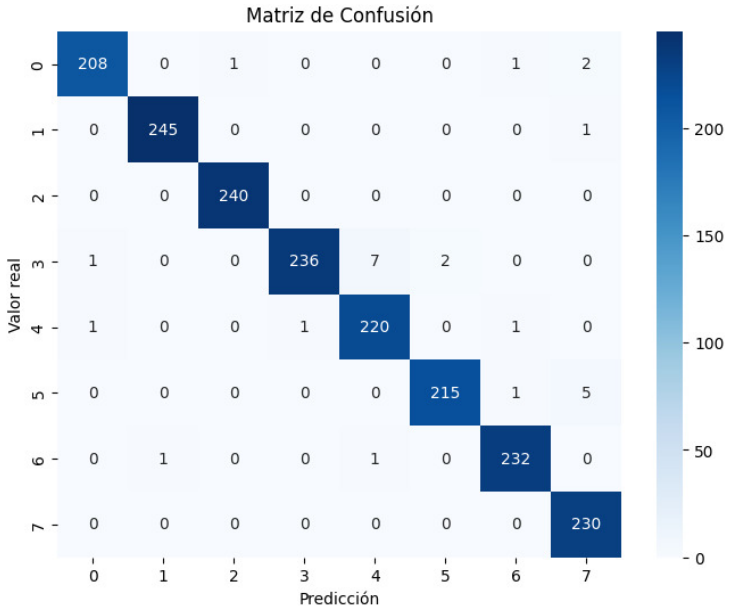

Figura 4.40: Matriz de confusión usando dataset con ángulos y distancias.

Tras compartir estos resultados con los tutores del proyecto y validar su rendimiento tanto a nivel técnico como práctico, consideramos que este modelo ofrece una solución sólida, eficiente y precisa. Por tanto, decidimos no realizar más optimizaciones sobre el dataset ni entrenamientos adicionales con esta arquitectura, y pasar a comparar su rendimiento con otras estrategias, como el uso de modelos preentrenados mediante transfer learning.

4.3.6. Transfer Learning y Fine-Tuning

Transfer learning (Pan y Yang, 2010) es una técnica que consiste en reutilizar un modelo previamente entrenado en una tarea para aplicarlo a una tarea nueva pero relacionada. Por ejemplo, un modelo entrenado para reconocer objetos comunes (como perros, autos o sillas) puede ser reutilizado y adaptado para una tarea más específica, como la identificación de razas de perros. Esto se logra aprovechando el conocimiento general aprendido en la primera tarea, evitando tener que entrenar un modelo desde cero. En la Figura 4.41 se muestra un esquema que representa este concepto.

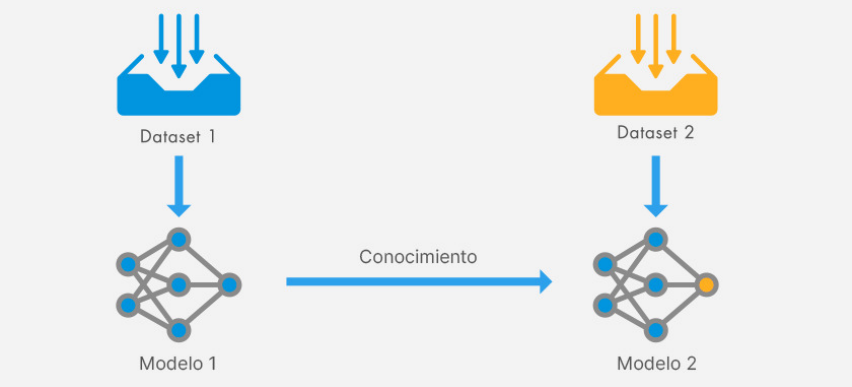

Figura 4.41: Diagrama de transfer learning.

El paso siguiente al transfer learning es el fine-tuning (ajuste fino). Consiste en realizar una nueva fase de entrenamiento sobre el modelo preentrenado, utilizando datos específicos del nuevo problema. Como se puede ver en la Figura 4.42, en lugar de mantener intacto el modelo original, se ajustan sus pesos (total o parcialmente) para adaptarlo a la nueva tarea, mejorando su precisión.

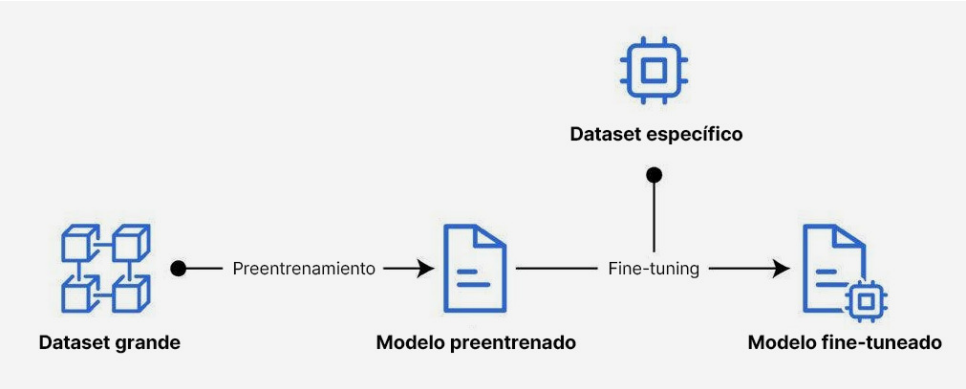

Figura 4.42: Diagrama de fine-tuning.

4.3.6.1. Adaptación de un modelo preentrenado

Después de entrenar nuestra propia red neuronal desde cero, quisimos explorar la posibilidad de aplicar transfer learning para comparar resultados y evaluar las ventajas de este enfoque. El primer paso era encontrar un modelo previamente entrenado en una tarea similar: la clasificación de poses corporales.

Tras una búsqueda exhaustiva en plataformas como Kaggle, Hugging Face y GitHub, localizamos un modelo de clasificación de poses de yoga cuya entrada coincidía exactamente con los ocho ángulos utilizados en nuestro dataset. La Figura 4.43 muestra la estructura de la red neuronal preentrenada extraída nuevamente con la herramienta Netron. Aunque este modelo no contemplaba las distancias que posteriormente añadimos, nos servía para probar la metodología de transfer learning y fine-tuning, que era el principal objetivo de este experimento.

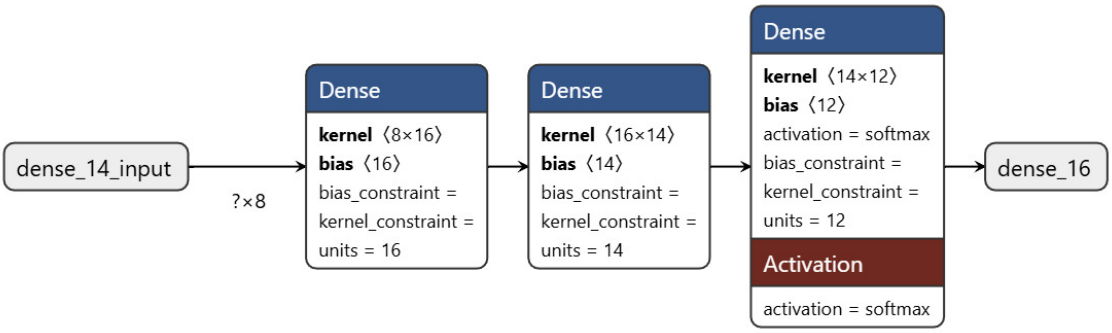

Figura 4.43: Estructura de la red neuronal preentrenada.

Para utilizar este modelo, fue necesario adaptar nuestro dataset. Eliminamos las distancias y nos quedamos solo con los ocho ángulos originales. Además, desnormalizamos los valores, ya que el modelo original no esperaba entradas normalizadas. Como nuestros ángulos estaban escalados entre 0 y 1 (al dividirse por 180), simplemente multiplicamos cada valor por 180 para volver a su rango original.

Al igual que en entrenamientos anteriores, dividimos los datos en dos conjuntos: 80 % para entrenamiento y 20 % para validación.

El modelo importado estaba formado por tres capas densas:

- Primera capa: 16 neuronas de salida y 144 parámetros, lo cual confirma una entrada de 8 características (8×16 pesos + 16 sesgos).
- Segunda capa: 14 neuronas de salida y 238 parámetros (16×14 pesos + 14 sesgos). La dimensión de entrada se infiere automáticamente de la capa anterior.
- Tercera capa (salida): 12 neuronas y 180 parámetros (14×12 pesos + 12 sesgos), una por cada una de las 12 clases de poses, con activación Softmax para generar probabilidades.

El total de parámetros del modelo es 564. A partir de la estructura, se deduce que el modelo original estaba diseñado para clasificar entre 12 poses diferentes.

Para aplicar transfer learning, congelamos todos los pesos del modelo original, de modo que no se actualicen durante el nuevo entrenamiento. Esto nos permite conservar el conocimiento aprendido previamente en las capas profundas del modelo.

A continuación, eliminamos la última capa de salida original y añadimos una nueva capa densa con 8 neuronas (correspondientes a nuestras 8 poses) y una función de activación Softmax, adecuada para clasificación multiclase. Usamos TensorFlow para crear este nuevo modelo secuencial, combinando las capas congeladas del modelo preentrenado con nuestra nueva capa de salida.

Este procedimiento nos permitió aprovechar la estructura del modelo original y adaptarlo a nuestra propia tarea con un esfuerzo computacional muy reducido.

4.3.6.2. Resultados de Transfer Learning

Una vez definida la nueva estructura del modelo con la capa de salida adaptada a nuestras 8 clases, procedimos a compilarlo y entrenarlo. Realizamos un entrenamiento de 20 épocas, ya que al tratarse de un modelo previamente entrenado, se espera que no necesite mucho tiempo para adaptarse a la nueva tarea.

Como se puede observar en la Figura 4.44, el tiempo total de entrenamiento fue de 15.55 segundos, alcanzando una accuracy del 80,28 % sobre el conjunto de validación.

```
Epoch 10/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7477 - loss: 2.3344 - val_accuracy: 0.7624 - val_loss: 1.8753
Epoch 11/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7659 - loss: 2.2417 - val_accuracy: 0.8024 - val_loss: 1.5108
Epoch 12/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7815 - loss: 2.1057 - val_accuracy: 0.8132 - val_loss: 1.4130
Epoch 13/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7810 - loss: 1.5838 - val_accuracy: 0.8013 - val_loss: 1.1061
Epoch 14/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7964 - loss: 1.4994 - val_accuracy: 0.8083 - val_loss: 1.3146
Epoch 15/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7923 - loss: 1.7594 - val_accuracy: 0.8337 - val_loss: 1.1869
Epoch 16/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7840 - loss: 1.7667 - val_accuracy: 0.8267 - val_loss: 0.9664
Epoch 17/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8044 - loss: 1.3216 - val_accuracy: 0.8245 - val_loss: 1.3611
Epoch 18/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8078 - loss: 1.4535 - val_accuracy: 0.8332 - val_loss: 0.8762
Epoch 19/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8068 - loss: 1.1206 - val_accuracy: 0.8218 - val_loss: 0.9734
Epoch 20/20
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8115 - loss: 1.1418 - val_accuracy: 0.8116 - val_loss: 1.6475
Duración del entrenamiento: 15.55 segundos
Accuracy:
58/58 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8028 - loss: 1.6788
[1.6474748849868774, 0.8115550875663757]
```

Figura 4.44: Entrenamiento del modelo usando transfer learning.

Tal y como se muestra en las gráficas de precisión y pérdida (Figura 4.45 y Figura 4.46 respectivamente), el rendimiento del modelo mejora rápidamente al principio, pero la pendiente se vuelve casi plana al acercarse a las 20 épocas. Esto indica que el modelo ya no mejora significativamente, lo cual es esperable dado que gran parte de sus pesos estaban congelados y sólo se estaba ajustando la nueva capa final.

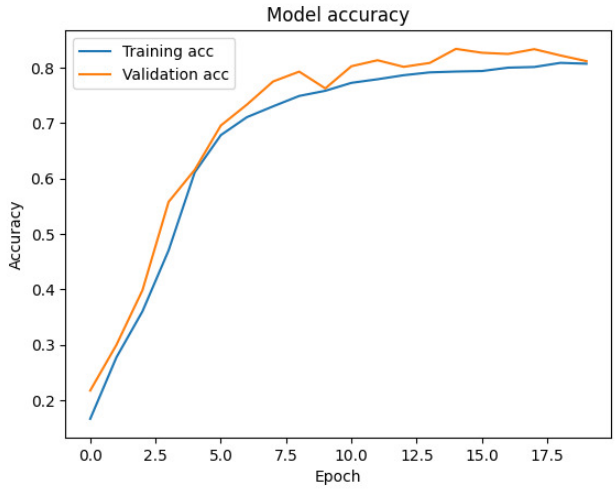

Figura 4.45: Gráfica de precisión usando transfer learning.

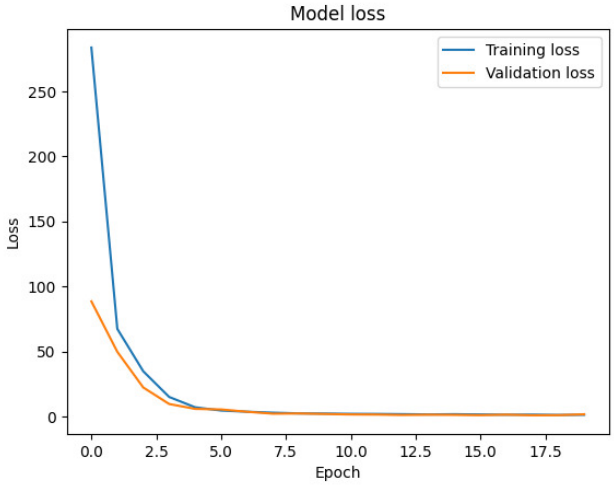

Figura 4.46: Gráfica de pérdida usando transfer learning.

4.3.6.3. Resultados de Fine-Tuning

Una vez completada la primera fase del transfer learning, procedimos a realizar el fine-tuning del modelo. Para ello, descongelamos las capas del modelo original, permitiendo que sus pesos pudieran ajustarse durante el entrenamiento. Este paso permite adaptar con mayor precisión el conocimiento del modelo preentrenado a la nueva tarea.

Dado que estas capas ya contienen información útil, no es necesario un entrenamiento extenso. Tras varias pruebas, determinamos que 10 épocas eran suficientes para observar mejoras sin sobreajuste.

Después del entrenamiento, el modelo alcanzó una accuracy de 85,27 % en tan solo 12 segundos, tal y como se muestra en la Figura 4.47.

```
Epoch 1/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8154 - loss: 0.9884 - val_accuracy: 0.8434 - val_loss: 0.6748
Epoch 2/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8267 - loss: 0.7740 - val_accuracy: 0.8391 - val_loss: 0.6249
Epoch 3/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8247 - loss: 0.8203 - val_accuracy: 0.8429 - val_loss: 0.6117
Epoch 4/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8344 - loss: 0.6928 - val_accuracy: 0.8483 - val_loss: 0.6022
Epoch 5/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8336 - loss: 0.7067 - val_accuracy: 0.8472 - val_loss: 0.5900
Epoch 6/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8357 - loss: 0.6703 - val_accuracy: 0.8504 - val_loss: 0.5892
Epoch 7/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8426 - loss: 0.7560 - val_accuracy: 0.8569 - val_loss: 0.5756
Epoch 8/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8376 - loss: 0.6962 - val_accuracy: 0.8499 - val_loss: 0.5930
Epoch 9/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8380 - loss: 0.6833 - val_accuracy: 0.8499 - val_loss: 0.6187
Epoch 10/10
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8367 - loss: 0.7407 - val_accuracy: 0.8564 - val_loss: 0.5611
Duración del entrenamiento: 12.02 segundos
Accuracy:
58/58 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8527 - loss: 0.5098
[0.5611392259597778, 0.8563714623451233]
```

Figura 4.47: Entrenamiento del modelo usando fine-tuning.

Las gráficas de precisión (Figura 4.48) y de pérdida (Figura 4.49) reflejan igualmente esta mejora, con incrementos en el porcentaje de acierto por época y una disminución estable de la pérdida.

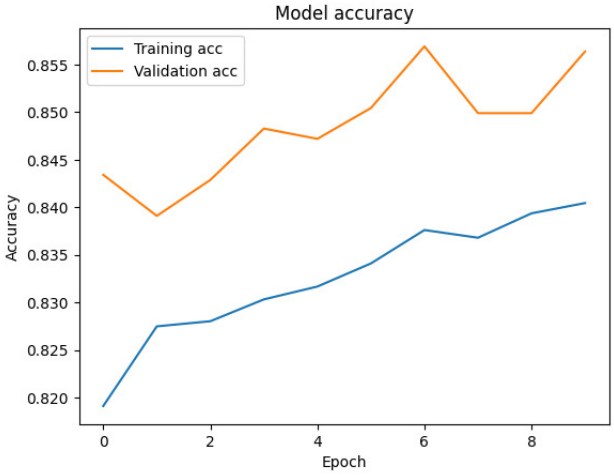

Figura 4.48: Gráfica de precisión usando fine-tuning.

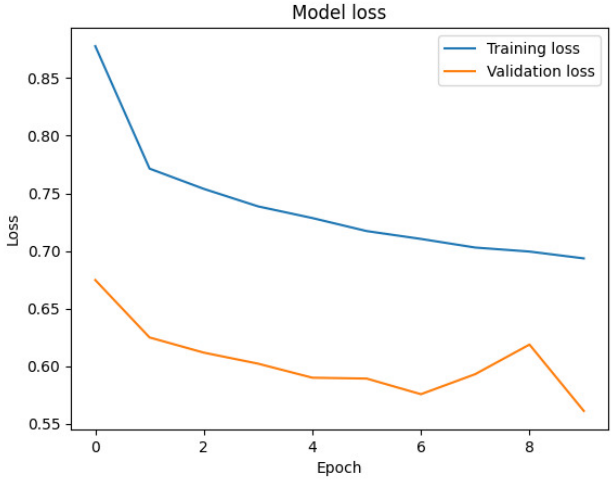

Figura 4.49: Gráfica de pérdida usando fine-tuning.

Esta mejora respecto al transfer learning simple (80,28 %) confirma que el fine-tuning fue efectivo para adaptar el modelo a nuestras poses específicas. Sin embargo, como ya habíamos anticipado, al tratarse de un modelo preentrenado pequeño y limitado en datos, el rendimiento final es inferior al de nuestra red neuronal entrenada desde cero.

Aun así, consideramos este ejercicio muy útil y didáctico. En muchos casos, especialmente cuando no se dispone de grandes cantidades de datos o recursos computacionales, el uso de transfer learning y fine-tuning puede ser una estrategia muy eficiente y poderosa para el desarrollo de modelos precisos y bien generalizados.

Tras comprobar que ninguna de estas estrategias alternativas supera el rendimiento alcanzado por nuestro modelo propio, confirmamos que este será el modelo definitivo que se empleará en la aplicación final.

4.4. Integración con Smart Glasses y funcionalidades adicionales

En esta sección detallamos el proceso de integración del modelo de reconocimiento de poses dentro de nuestra aplicación Android, así como su conexión con el hardware específico: las Smart Glasses Rokid Air Pro. También abordamos algunas funcionalidades adicionales diseñadas para mejorar la experiencia del usuario.

4.4.1. Conectividad e integración de las Smart Glasses

Para el despliegue del sistema en un entorno real, utilizamos las Smart Glasses Rokid Air Pro. La elección de este modelo se basó en dos razones principales: por un lado, se trata de uno de los dispositivos de realidad aumentada más potentes y accesibles del mercado; por otro, estaban disponibles en la facultad para su utilización durante todo el proyecto, facilitando el desarrollo, las pruebas y la integración del sistema.

Las Rokid Air Pro (Figura 4.50) son unas gafas de realidad aumentada (AR) ligeras y portátiles, diseñadas para ofrecer una experiencia inmersiva mediante la proyección de una pantalla virtual de hasta 120 pulgadas perceptible frente al usuario. Se conectan mediante cable USB-C a dispositivos compatibles (smartphones, tablets, PCs o consolas) y cuentan con altavoces integrados y un diseño plegable que facilita su transporte y almacenamiento.

Figura 4.50: Smart Glasses Rokid Air Pro.

4.4.1.1. Arquitectura del SDK

Para lograr la integración entre las gafas y nuestra aplicación Android, Rokid proporciona un conjunto de herramientas denominado AXR Phone SDK. Un SDK (Software Development Kit) es un conjunto de bibliotecas y utilidades que facilita a los desarrolladores la implementación de funcionalidades específicas en sus aplicaciones.

El SDK de Rokid se compone de tres módulos principales:

- AXR Phone SDK-glassdevice
- AXR Phone SDK-glasscamera
- AXR Phone SDK-glassvoice

A continuación se describen sus funcionalidades:

- AXR Phone SDK-glassdevice: Este módulo es esencial y proporciona control sobre el hardware y funcionalidades básicas de las gafas. Permite:

  - Obtener información del hardware.
  - Acceder a los datos del sensor de movimiento.
  - Capturar eventos de teclas y del panel táctil de las gafas.
  - Obtener datos del micrófono.
  - Ajustar el brillo de la pantalla.
  - Controlar el modo de visualización.
  - Habilitar la visualización en modo de doble pantalla.

- AXR Phone SDK-glasscamera: Este módulo permite acceder y controlar la cámara de las gafas. Sus funciones incluyen:

  - Obtener información de la cámara.
  - Habilitar o deshabilitar la cámara.
  - Obtener la vista previa (preview).
  - Tomar fotografías.
  - Grabar vídeos.
  - Configurar ajustes de la cámara.

- AXR Phone SDK-glassvoice: Este módulo habilita el reconocimiento de voz, aunque con limitaciones:

  - Reconocimiento de discurso en chino e inglés.

En nuestro caso, no utilizamos el módulo de voz, ya que no requerimos comandos por voz en la aplicación. Además, su falta de soporte para el idioma español lo hace poco útil en nuestro contexto.

4.4.1.2. Adaptación de la aplicación demo

Además de la documentación técnica, Rokid proporciona una aplicación demo desarrollada en Kotlin (Rokid, Inc., 2025), que sirve como referencia para integrar las Smart Glasses con dispositivos Android. Dado que esta aplicación ya incluía muchas de las funcionalidades básicas necesarias para trabajar con las gafas, decidimos utilizarla como base sobre la que construir e integrar nuestro sistema.

La demo oficial está organizada en varios módulos:

- Un módulo base que verifica si las Smart Glasses están correctamente conectadas al dispositivo Android. Solo en caso afirmativo permite el uso completo de la aplicación.
- Un módulo dedicado a la cámara, que permite capturar fotos, grabar vídeos y visualizar la imagen en tiempo real.
- Un módulo encargado del hardware de las gafas, que permite detectar pulsaciones de botones físicos y acceder al micrófono.
- Un módulo adicional orientado al control por voz.

Para nuestro proyecto, el módulo más relevante fue el de la cámara, concretamente la función de previsualización en tiempo real, ya que es clave para aplicar nuestro sistema de detección y clasificación de poses.

La primera tarea fue integrar nuestro sistema en este módulo. Gracias a que durante el desarrollo inicial seguimos buenas prácticas de desacoplamiento de código, la migración fue rápida y sencilla. Pudimos reutilizar la mayor parte del código de la versión anterior sin complicaciones.

Nuestro sistema necesita como entrada objetos de tipo InputImage, mientras que la cámara de las gafas genera imágenes en formato Bitmap. Afortunadamente, la clase InputImage proporciona un método para construir una instancia a partir de un Bitmap, por lo que la conversión entre formatos fue directa y sin pérdida de información.

Tras varias pruebas con las Smart Glasses en distintos entornos y condiciones, validamos que el sistema funcionaba correctamente. A continuación, llevamos a cabo una refactorización del código, eliminando los módulos innecesarios y adaptando los existentes a nuestras necesidades. Esto resultó en una aplicación más ligera, organizada y mantenible.

4.4.1.3. Compatibilidad de dispositivos

Un aspecto importante a tener en cuenta es que las Rokid Air Pro no son compatibles con todos los dispositivos Android. Tal como se muestra en la Figura 4.51, el fabricante proporciona una lista de smartphones compatibles, principalmente modelos de gama media-alta con soporte completo para conexiones USB-C con salida de vídeo (DisplayPort Alt Mode).

Brand	Model	OS	Remark
Huawei	Mate 10	EMUI 8.0(Android 8.0)/HarmonyOS	Huawei smart resolution will affect the system resolution, please make sure to use it when smart resolution is turned off.
	Mate 10 pro	EMUI 8.0(Android 8.0)/HarmonyOS	
	Mate 20	EMUI 9.0(Android 9.0)/HarmonyOS	
	Mate 20 pro	EMUI 9.0(Android 9.0)/HarmonyOS	
	Mate 20 X	EMUI 9.0(Android 9.0)/HarmonyOS	
	Mate 30 E	EMUI 10.0(Android 10.0)/HarmonyOS	
	Mate 30	EMUI 10.0(Android 10.0)/HarmonyOS	
	Mate 30	EMUI 10.0(Android 10.0)/HarmonyOS	
	Mate 30 pro	EMUI 10.0(Android 10.0)/HarmonyOS	
	Mate 40	EMUI 11.0(Android 10.0)/HarmonyOS	
	Mate 40 Pro	EMUI 11.0(Android 10.0)/HarmonyOS	
	Mate X2	EMUI 11.0(Android 10.0)/HarmonyOS	
	P20 Series	EMUI 8.1(Android 8.1)/HarmonyOS	
	P30 Series	EMUI 9.1(Android 9.0)/HarmonyOS	
	P40 Series	EMUI 10.1(Android 10)/HarmonyOS	
	P50 Series	HarmonyOS	
Honor	V20	MagicUI2.0.1(Android 9)	Honor smart resolution will affect the system resolution, please make sure to use it when smart resolution is turned off.
	Magic 3	Magic UI 5.0(Android 11)	
OnePlus	7 Series	H2OS(Android 9)	
	8 Series	HydrogenOS(Android 10)	
Black Shark	Black Shark 3	JoyUi 11(Android 10)	
	Black Shark 4	JoyUi 12.5(Android 11)	
Sony	Xperia PRO-I	Android 11	
	Xperia 1iii	Android 11	
Nubia	redmagic6	Redmagic OS 4.5	
ZTE	中兴 Axon 30 Pro	My OS 11	
OPPO	Find X2	Color OS 7.1(Android 10)	There are stability issues.
	Find X2 Pro	Color OS 7.1(Android 10)	
	Find X3	Color OS 11.2(Android 11)	
	Find X3 Pro	Color OS 11.2(Android 11)	
Samsung	S10 Series	Android 9.0	S20 series of national banks have problems that cannot be cast.
	S20 Series	One ui 2.0(Android 10)	
	S21 Series	One ui 3(Android 11)	
	Note 10 Series	One ui(Android 9.0)	
	Note 20 Series	One ui 2.5(Android 10)	
	Galaxy Z Fold2	One UI(Android 9.0)	
	Galaxy Z Fold3	One UI 3.1.1(Android 11)	
	W22	One UI 3.1.1(Android 11)	

Figura 4.51: Lista de móviles compatibles con Rokid Air Pro.

Para evitar errores o confusiones por parte del usuario, implementamos un sistema de validación en el inicio de la aplicación. Este sistema detecta el modelo de dispositivo y lo compara con la lista de terminales compatibles. Si el modelo está en la lista, el usuario puede continuar con normalidad. En caso contrario, se muestra una interfaz de advertencia, como la que se ve en la Figura 4.52, informando al usuario de que su dispositivo no es compatible y bloqueando el acceso a las funcionalidades principales de la aplicación.

Figura 4.52: Mensaje mostrado en caso de dispositivo incompatible.

4.4.2. Feedback al usuario

Una vez integrado el sistema de detección y clasificación de poses en la nueva versión de la aplicación y habiendo verificado su correcto funcionamiento con las Smart Glasses, el siguiente paso fue diseñar un sistema de feedback claro e inmediato para el usuario. Nuestro objetivo era comunicar de manera visual y auditiva los resultados del sistema, mejorando así la experiencia de uso y facilitando la comprensión de las posturas detectadas.

4.4.2.1. Visualización de esqueletos en AR

Como parte del feedback visual, decidimos implementar una representación del esqueleto corporal sobre la imagen capturada por la cámara en tiempo real. Esta superposición muestra los puntos clave detectados por el modelo y cómo se interpretan para clasificar la pose, lo que facilita una mejor comprensión de la postura. Al tratarse de una versión simplificada de la figura humana, también resulta más accesible para personas con baja visión, al resaltar únicamente los elementos esenciales de la pose.

Una vez ejecutado el análisis de la imagen mediante ML Kit, obtenemos un objeto de tipo Pose, que contiene las coordenadas (x, y) de todos los puntos clave del cuerpo (por ejemplo: hombros, codos, rodillas, etc.).

A partir de estas coordenadas —siguiendo la documentación oficial de ML Kit— se recorre cada punto para dibujar un círculo en su posición. Asimismo, empleamos la lista

predefinida de conexiones entre puntos (por ejemplo, entre hombro y codo, o rodilla y tobillo) propuesta por ML Kit para trazar las líneas que representan las extremidades y articulaciones de la persona. Esto nos permitió reutilizar la lógica ya probada de la librería, asegurando compatibilidad y reduciendo el esfuerzo de implementación.

Ajustamos parámetros visuales como el tamaño de los círculos, el grosor de las líneas y los colores utilizados para que el esqueleto sea fácilmente visible sobre la imagen del fondo, incluso en entornos con variaciones de iluminación. De esta forma conseguimos una interfaz intuitiva que permite al usuario identificar rápidamente la postura detectada.

4.4.2.2. Sistema Text-to-Speech

Para complementar la información visual, incorporamos un sistema de text-to-speech (TTS) que proporciona retroalimentación auditiva en tiempo real a través de los altavoces integrados en las monturas de las Smart Glasses. Este sistema es responsable de notificar al usuario los eventos clave de la aplicación, tales como:

- Conexión o desconexión de la cámara.
- Inicio o pausa del sistema de detección.
- Nombre de la pose detectada.

Para su implementación utilizamos la clase TextToSpeech del SDK de Android, que proporciona una interfaz completa para la síntesis de voz. Entre sus funciones más destacadas se encuentra speak(), encargada de emitir mensajes en lenguaje natural. Esta función requiere los siguientes parámetros:

1. Texto a reproducir, que puede incluir objetos TtsSpans para personalizar la entonación.
2. Modo de cola (queueMode), que determina cómo gestionar los mensajes en espera. Se puede elegir entre:
   - QUEUE_ADD: añade el nuevo mensaje al final de la cola actual.
   - QUEUE_FLUSH: vacía la cola inmediatamente y reproduce el nuevo mensaje.

   En nuestro caso utilizamos QUEUE_FLUSH, ya que es prioritario que la voz esté sincronizada con la realidad. Este mecanismo descarta inmediatamente cualquier mensaje pendiente y reproduce solo la última detección, evitando solapamientos. Aunque podría parecer que esto interrumpe bruscamente la comunicación, abordamos este aspecto en la sección siguiente, donde implementamos un sistema de estabilización de poses que filtra las transiciones no deseadas y permite un feedback más fluido. Así, si una pose se detecta justo antes de que otra la reemplace, asumimos que era una postura de transición y preferimos informar únicamente de la posición final estabilizada.
3. Parámetros adicionales, como volumen o canal de salida. En nuestro caso, se pasa null para utilizar los valores por defecto.
4. Utterance ID, un identificador único para hacer seguimiento de cada mensaje. Esto permite recibir callbacks sobre el inicio, fin o errores durante la reproducción.

Además, desde el menú de accesibilidad de la aplicación se permite seleccionar el idioma (español o inglés), lo cual afecta tanto a los textos visibles en la interfaz como al sistema de síntesis de voz, asegurando una experiencia consistente para usuarios de distintos idiomas. Esta funcionalidad será tratada con más detalle en la Sección 4.4.4.

4.4.3. Sistema de estabilización de poses

Uno de los aspectos más importantes en el desarrollo de nuestro sistema era definir cuándo se debía notificar una pose al usuario. Este paso es crucial, ya que representa la principal vía de comunicación entre el modelo y el usuario. Un sistema de detección puede alcanzar altos niveles de precisión, pero si notifica poses con demasiada frecuencia o con escasa fiabilidad, la experiencia de uso se deteriora considerablemente.

Con el objetivo de optimizar esta interacción, diseñamos un sistema de estabilización de poses que persigue dos metas fundamentales:

1. Estabilizar la clasificación de poses: evitar fluctuaciones bruscas entre clases consecutivas, incrementar la confianza del sistema y filtrar detecciones erróneas o espurias.

2. Reducir notificaciones innecesarias: garantizar que el usuario reciba únicamente información relevante, evitando un exceso de mensajes que puedan resultar molestos.

4.4.3.1. Algoritmo de buffer temporal y umbral de confianza

La solución implementada se basa en un buffer temporal deslizante, tal y como ilustra la Figura 4.53. Este buffer almacena todas las poses detectadas durante los últimos 1000 milisegundos (1 segundo). Por cada nueva detección:

- Se eliminan del buffer todas las entradas cuya antigüedad supere el segundo.

- Se añade la nueva pose detectada.

- Se cuenta la frecuencia de aparición de cada pose en ese intervalo.

A partir de esta información, se identifica la pose más frecuente en el buffer. Sin embargo, para considerarla una pose estable, no basta con ser la más repetida. Debe, además, superar un umbral de estabilidad, es decir, representar al menos el 60 % del total de poses registradas en el último segundo.

Solo cuando se cumple este criterio, y si la pose candidata difiere de la última pose notificada, el sistema procede a considerarla como válida y la comunica al usuario. En caso contrario, se ignora la nueva detección y no se emite ningún mensaje, contribuyendo así a reducir el ruido y mejorar la coherencia del sistema.

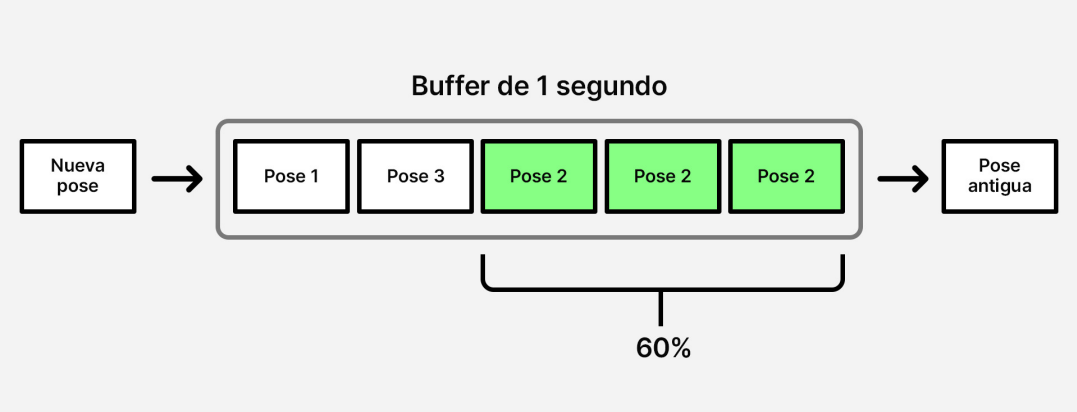

Figura 4.53: Diagrama del sistema de estabilización.

4.4.3.2. Ajuste de ventana temporal y umbral mínimo

Este sistema tiene dos parámetros clave que controlan su funcionamiento:

- Duración de la ventana temporal
- Umbral mínimo de estabilidad

Inicialmente, probamos con una ventana de 500 milisegundos. Aunque se apreciaba una clara mejora respecto a una versión sin estabilización, el sistema aún generaba notificaciones innecesarias, algunas de ellas incorrectas. Aumentamos entonces el intervalo a 1000 milisegundos, lo que permitió una mejora significativa en la precisión, reduciendo falsos positivos sin perder capacidad de respuesta. Se exploraron también ventanas mayores (2000 y 3000 ms), pero se comprobó que en estos casos se perdía la detección de poses válidas de corta duración. Por tanto, el valor óptimo quedó fijado en 1000 milisegundos, ofreciendo un equilibrio entre reactividad y estabilidad.

Respecto al umbral de estabilidad, partimos de un valor del 40 %, que permitía notificar poses si al menos el 40 % de las entradas del buffer coincidían. Aunque esta configuración descartaba muchas detecciones erróneas, en movimientos rápidos aún se notificaban poses incorrectas. Subimos el umbral al 85 %, pero esto generaba un comportamiento demasiado restrictivo: algunas poses válidas no se anunciaban por ligeras fluctuaciones en la postura. Finalmente, elegimos un valor intermedio de 60 %, que proporcionaba la mejor combinación entre precisión y sensibilidad.

Con esta estrategia se reducen significativamente las transiciones abruptas y se asegura que las poses notificadas sean realmente representativas y estables. Tras realizar numerosas pruebas con usuarios reales y en diferentes escenarios, se concluyó que el sistema proporciona una experiencia de uso notablemente más precisa, fluida y cómoda.

4.4.4. Ajustes de accesibilidad

Dado que la aplicación está dirigida a personas con dificultades visuales, consideramos esencial incorporar una serie de ajustes de accesibilidad (Santana-Mansilla et al., 2015) que permitan adaptar la experiencia de uso a las necesidades específicas de cada usuario. Nuestro objetivo es ofrecer una interfaz más cómoda, legible y personalizable, facilitando así la interacción y la autonomía durante el uso de las Smart Glasses.

Finalmente se han implementado tres características configurables:

1. Modo de color El usuario puede elegir entre tres opciones de visualización:

   - Modo claro
   - Modo oscuro
   - Modo por defecto del sistema

   Cada persona con baja visión puede tener distintas sensibilidades lumínicas. Por ejemplo, algunos usuarios pueden experimentar incomodidad con fondos blancos brillantes, mientras que otros prefieren interfaces claras porque mejoran su percepción del contenido. Por ello, ofrecemos libertad de elección, estableciendo como configuración predeterminada el modo por defecto del sistema, asumiendo que el dispositivo ya estará configurado según las preferencias del usuario. La Figura 4.54 muestra la interfaz de ajustes de accesibilidad con la comparación lado a lado de los modos claro y oscuro.

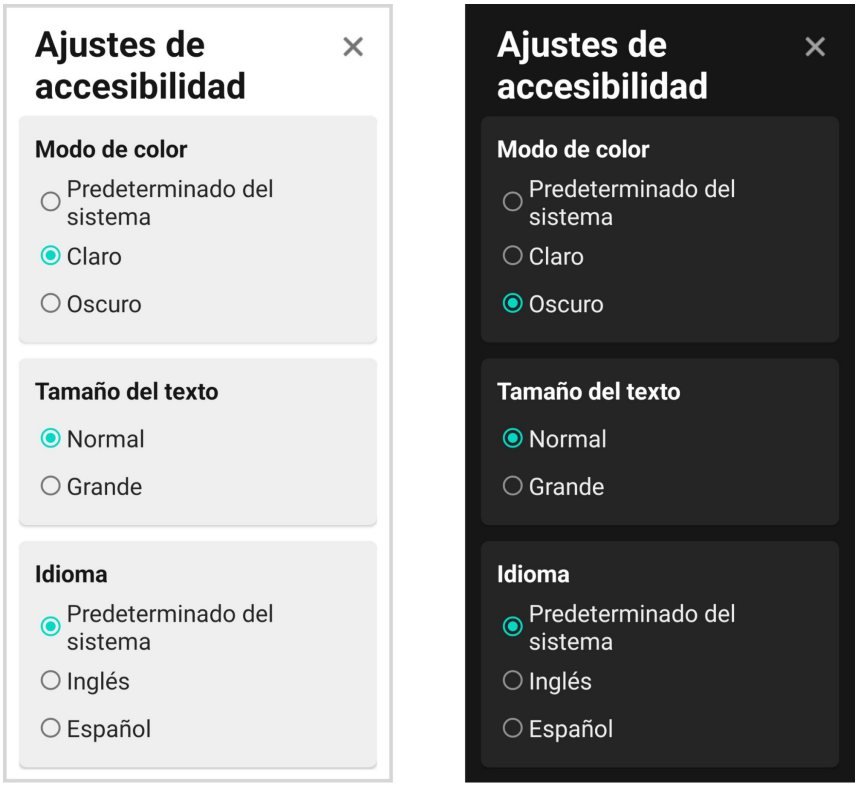

Figura 4.54: Interfaz y comparación de ajuste del modo de color.

2. Tamaño de letra Se permite seleccionar entre dos tamaños de fuente:

   - Normal (valor por defecto)
   - Grande (incremento del 30 % sobre el tamaño normal)

   Aunque la opción normal ya ha sido diseñada para garantizar una legibilidad aceptable, se incluye una versión ampliada para aquellos usuarios que requieran un mayor tamaño de texto. Esto facilita la lectura de elementos clave de la interfaz, como botones, instrucciones o resultados, sin comprometer el diseño visual. En la Figura 4.55

se puede apreciar la interfaz de ajustes de accesibilidad con la comparación lado a lado del tamaño de texto normal y grande.

Figura 4.55: Interfaz y comparación de ajuste del tamaño de texto.

3. Idioma El idioma de la aplicación puede establecerse entre:

   - Idioma por defecto del sistema (predeterminado)
   - Español
   - Inglés

Este ajuste afecta tanto a los textos visibles en la interfaz como a las notificaciones por voz generadas por el sistema text-to-speech. Además, el sistema está diseñado de manera modular y escalable, lo que facilitará la incorporación de nuevos idiomas en futuras versiones, si se considera necesario. En la Figura 4.56 se muestra, de forma comparativa, la interfaz de ajustes presentando lado a lado las opciones en español e inglés.

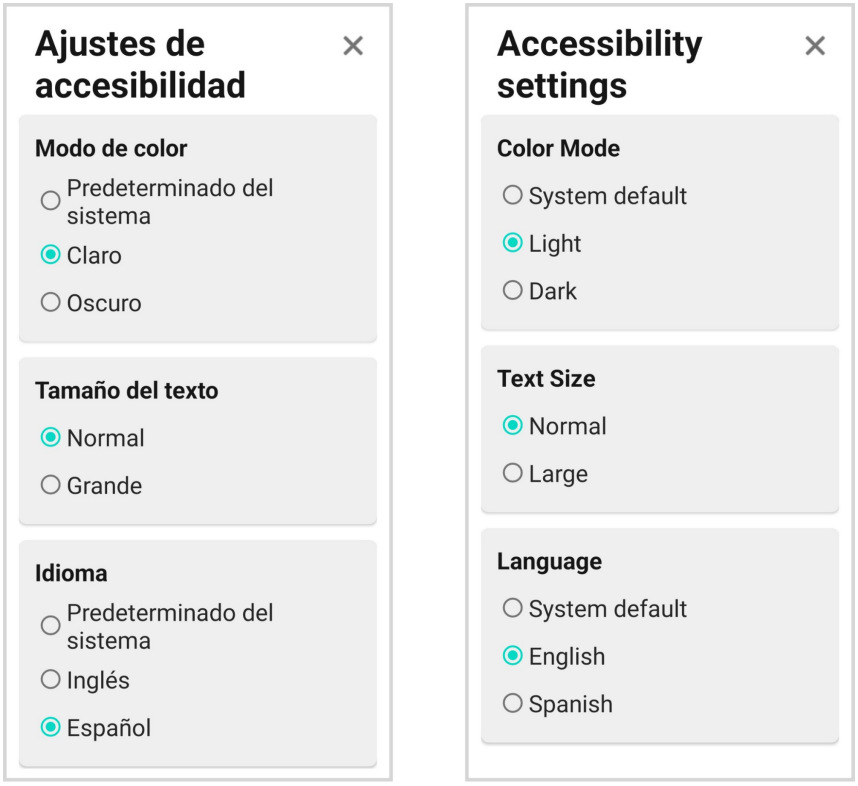

Figura 4.56: Interfaz y comparación de ajuste del idioma.

Para garantizar una experiencia personalizada y persistente, se ha implementado un sistema dedicado al almacenamiento de las preferencias del usuario. Este sistema se apoya en un mecanismo estándar de Android llamado SharedPreferences (Google LLC, 2025c), diseñado específicamente para guardar de forma segura pequeñas cantidades de datos, como las configuraciones elegidas, directamente en la memoria interna del dispositivo. Toda la gestión de estas preferencias se centraliza en una única clase organizadora, PreferencesManager, que actúa como intermediaria entre las elecciones del usuario y el almacenamiento físico.

Cuando el usuario selecciona una opción, como cambiar al tema oscuro o elegir un tamaño de texto más grande, PreferencesManager se encarga de guardar esta elección. Lo hace asociando un identificador único (una 'clave', como key_theme o key_text_size) con el valor seleccionado (por ejemplo, dark o large) dentro de un archivo de preferencias privado para la aplicación. El uso de constantes predefinidas para estos valores asegura la coherencia y evita errores.

Así, en futuros usos de la aplicación, los ajustes se restauran automáticamente al iniciarse. Esto ocurre porque, al arrancar, la aplicación consulta a PreferencesManager para recuperar los valores previamente guardados para cada clave. Si alguna preferencia no ha sido modificada por el usuario, se aplica un valor por defecto (como seguir el tema del sistema o usar el tamaño de texto normal).

Una vez recuperadas las preferencias, PreferencesManager utiliza las herramientas adecuadas de Android (AppCompatDelegate) para aplicar activamente esos ajustes, modificando la apariencia visual (colores del tema) y el idioma de la interfaz en tiempo real. En el caso particular del tamaño del texto, cuando este se cambia, se envía además una notificación interna (LocalBroadcastManager) a las pantallas activas de la aplicación, permitiéndoles reajustar su contenido inmediatamente sin necesidad de reiniciar. De esta forma,

la personalización elegida por el usuario se mantiene consistentemente entre sesiones, sin necesidad de reconfigurar manualmente.

Es relevante mencionar que las prácticas actuales de desarrollo en Android recomiendan el uso de Jetpack DataStore (Google LLC, 2025a) como una solución más moderna para el almacenamiento de preferencias, principalmente por sus mejoras en operaciones asíncronas y manejo de datos mediante Flujos (Flows). Se evaluó su implementación en este proyecto; sin embargo, como ya hemos explicado, la aplicación se basa en una demostración proporcionada por Rokid que utiliza versiones anteriores de ciertas librerías fundamentales. La introducción de DataStore y sus dependencias asociadas generaba conflictos de versiones significativos que comprometían la estabilidad general de la aplicación base. Por tanto, para asegurar la compatibilidad y evitar una refactorización profunda y potencialmente problemática del código heredado, se tomó la decisión pragmática de mantener el uso de SharedPreferences. Para las necesidades específicas de este proyecto –almacenar un conjunto limitado de configuraciones clave-valor–, SharedPreferences sigue siendo una solución perfectamente válida, robusta y eficiente.

En conjunto, estos ajustes, gestionados mediante la implementación descrita, suponen una mejora significativa en términos de accesibilidad, usabilidad y personalización, elementos clave cuando se trata de ofrecer una herramienta verdaderamente inclusiva para usuarios con diversidad visual.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

En este proyecto nos enfocamos en encontrar una manera de usar los últimos avances tecnológicos para poder ser capaces de ayudar a personas que sufren de enfermedades que provoquen la baja visión. Para ello, primero realizamos un estudio acerca de estas, y de las soluciones ya existentes (tanto tradicionales como recientes) para solventar este tipo de problemas. La idea era combinar el abanico de posibilidades que nos ofrecen los dispositivos wearable, como son las **Smart Glasses**, que poco a poco se irán integrando y normalizando en el uso cotidiano; con el uso de la **IA**, utilizando modelos relacionados como redes neuronales, para obtener una aplicación Android capaz de llevar a cabo esta labor.

El resultado ha sido una aplicación Android que, usando de la cámara de las gafas inteligentes, es capaz de reconocer la pose de la persona que se encuentra delante del usuario que las está usando. Además, se dibujan una serie de puntos que reflejan su esqueleto y una descripción por voz. Con estas funcionalidades, nuestra aplicación ofrece una ayuda visual y auditiva especialmente útil para las personas que sufren de problemas relacionados con baja visión.

Con este trabajo pretendemos demostrar el gran potencial de todas las nuevas tecnologías y todas las posibilidades que ofrece y cómo podemos utilizarlas para encontrar soluciones novedosas a problemas cotidianos.

5.2. Trabajo futuro

Como trabajo futuro y posibilidades de mejora, planteamos las siguientes opciones:

- Añadir más poses a nuestro modelo, ya que nuestro proyecto es totalmente escalable, las posibilidades son infinitas.
- Simplificar mucho más el modo entrenamiento, para que permita a los usuarios añadir nuevas poses personalizadas sin necesidad de ningún conocimiento técnico.
- Implementar un sistema colaborativo donde la comunidad de usuarios pueda compartir y validar nuevas poses, creando una base de datos colectiva por ejemplo.
- Investigar la integración con otros sensores (como cámaras térmicas o de profundidad) para mejorar la precisión en diversas condiciones de iluminación.

- Explorar la combinación de detección de poses con sistemas de navegación espacial para asistencia en movilidad, como por ejemplo para detectar escalones.

Creemos que nuestro proyecto tiene un enorme potencial de crecimiento y evolución. Las mejoras planteadas permitirán transformar esta solución en una herramienta aún más completa y accesible, que se adapte perfectamente a las necesidades cotidianas de las personas con baja visión. Aunque nuestro objetivo principal sigue siendo claro: mantener la sencillez de uso mientras mejoramos significativamente la calidad de vida de nuestros usuarios.

Chapter 6

Introduction

Low vision is a condition that irreversibly limits visual ability, affecting everyday activities such as recognising facial expressions or postures of other people. According to the OMS (OMS, 2019), more than 2.2 billion people in the world suffer from some kind of visual impairment, with low vision being one of the most disabling in social settings.

This project develops an Android application for smart glasses that helps users with low vision to interpret the postures of the people around them. The solution uses artificial intelligence to detect basic poses such as "arms crossed", "handshake right" or "pointing left", providing audio and visual feedback to the user.

The system uses ML Kit's optimised computer vision models, specifically designed to work efficiently on wearable devices with limited resources. Through ML Kit's Pose Detection, the app accurately identifies key points on the human body (such as shoulders, elbows and hips) in real time, even on mid-range devices. The AI model we have developed is able to interpret this data, which we then use to provide simplified audio and visual descriptions. This approach allows users with low vision to interpret postures and gestures in their environment immediately, reducing their reliance on residual vision and improving their social interaction.

A key innovation is the local processing of data, ensuring privacy and speed by avoiding dependence on external servers. The application also includes accessibility functionalities, including changing the system's colour mode, font size and language.

This work represents a first step towards more advanced visual assistance systems. Future versions could integrate facial recognition, ambient navigation or voice interaction. The ultimate goal is to improve the autonomy and quality of life of people with low vision through accessible and unobtrusive technology.

The project combines machine learning techniques, mobile development and accessible user experience. The results could benefit not only patients, but also therapists and assistive technology developers.

6.1. Motivation

Assistive technologies based on artificial vision are changing in recent years the way people suffering from some kind of visual impairment interact with their environment. In this context, wearable devices, especially smart glasses, have emerged as a promising solution thanks to their ability to integrate artificial intelligence in an unobtrusive and portable way. A recent milestone in this field is the launch of devices such as the Ray-Ban

Meta (Meta, 2023).

According to official figures from the World Health Organisation (OMS, 2023), almost 1.3 billion people suffer from some form of visual impairment, 260 million of whom have low vision. These figures are on an upward trend, driven mainly by an ageing population and the increase in chronic eye diseases. Recent studies show that people with low vision face higher risks of social isolation and employment difficulties, with a significant economic impact: In Spain, the cumulative economic cost, for the period 2021 to 2030, which is associated with 5 evaluated pathologies (including glaucoma, macular degeneration and myopia magna) will reach nearly 100 billion euros at the end of the period. (Fundación Porib, 2025).

Our aim in this project is to improve the quality of life of people with low vision as much as we can, so we will investigate and then explain what low vision itself consists of and what some of the main pathologies that cause it are.

In this work we intend to investigate solutions that improve the quality of life of people with low vision by using smart glasses. We will carry out a comprehensive research, firstly by getting to know and investigating these diseases, analysing the different profiles of patients with their main visual limitations, and we will study some current solutions based on Augmented Reality. Then, we will investigate about the hardware and software best suited to such a solution and we will develop both visual and auditory aids.

Our work arises as a response to these needs, with the aim of developing an innovative system that combines:

- Smart glasses as an integrated and discreet visual support
- Advanced pose detection techniques using artificial intelligence
- Accessible and intuitive interface for use by all users

Unlike systems such as OrCam MyEye (Orcam, 2024) focused on text reading, our solution will specialise in interpreting body language, addressing a need that is not very well covered in current solutions.

6.2. Project Objectives

This project has a dual purpose: on the one hand, to develop an innovative technological solution for people with low vision, and on the other, to lay the foundations for future improvements and research in this field. It could be said that our main objective is to develop a system integrated in smart glasses that, through artificial intelligence and pose detection techniques, provides visual and auditory assistance to people with low vision, improving their autonomy and facilitating their day-to-day life.

The following is the link to the repository containing the project's complete code, where the full implementation and supplementary documentation are available:

`https://github.com/alvave03/TFG-2024-25`

If we refer to more specific objectives, we could differentiate them into the following:

1. Implement our own efficient pose detection model using Google ML Kit, capable of identifying human poses in real time.
2. Design an intuitive and accessible user interface in an Android application that allows interaction with smart glasses.

3. Create a system that, through the smart glasses, describes with voice and visually the postures of the people around the user.
4. Integrate the Android application with the smart glasses to guarantee fluid and stable communication between the two devices, allowing information to be sent in real time about the poses detected.
5. Guarantee the scalability of the system to allow the future incorporation of new poses and functionalities.
6. Evaluate the usability and effectiveness of the system through real-time testing.
7. Develop and implement the Android application containing all the functionalities described above.

Fulfilling these objectives will allow the creation of a practical tool that combines the possibilities offered by wearable devices and artificial intelligence to address a real problem, with the capacity to evolve towards more complete solutions in the future.

6.3. Workplan

To ensure the optimal development of this project, we will divide the work into several phases with specific activities for each phase.

- **Research and Analysis:** In this initial phase we will carry out a study of the needs of people with low vision, analysing their limitations and the scenarios where the system could be most useful. In addition, we will study the different technologies available for pose detection, comparing their advantages and limitations. This analysis will allow us to define the technical and functional requirements for further development.
- **Development of the AI Model:** The most relevant part of the project is to create an efficient pose detection model. To do this, we will collect a representative dataset, which we will use to train our model, and we will carry out several different implementations to choose the one with the best results.
- **Application Development:** With the AI model ready, we will proceed to build the Android application that will make use of it. The development will include an interface that is very intuitive, accessible and easy to use. We will implement the visual aid system in the camera of the glasses and the auditory aid with its speakers. A key aspect will be the integration with the Smart Glasses, having to establish a stable communication to display the processed information.
- **Integration and Testing:** In this phase we will unify all the components of the system, verifying their correct functioning together through technical and usability tests. We will evaluate the accuracy of pose detection, the clarity of feedback and the actual user experience, implementing the necessary adjustments to optimise the overall performance of our solution.

In addition, we will use an agile methodology with short development cycles (2-3 weeks) where we will be: 1) building features step by step, 2) testing them with users, and 3) making continuous improvements. This will help us to create a tool that really serves the visually impaired and to coordinate effectively.

6.4. Document structure

This document is structured as follows:

In Chapter 2, "State of the Art", we explore how artificial intelligence, specifically Pose Detection, can improve the quality of life of people suffering from low vision. It analyses the causative pathologies (such as cataracts, AMD and glaucoma), the limitations of traditional aids and the potential of Smart Glasses as assistive devices. In addition, key technologies such as IoT, Google ML Kit and MediaPipe to implement smart systems are detailed, along with AI techniques such as neural networks to train personalised models.

In Chapter 3, "Planning and Resources", we show the planning of the project, we describe all the hardware resources we have chosen with the justification for this choice, the SDKs we have used, the libraries used for the integration of artificial intelligence in the various Android applications and the development environments we are going to use in the development of this project.

In the Chapter 4, "Project development and results", we detail the complete development of the project, explaining how each of the functionalities are implemented in detail and their integration with Android Studio and the other tools and software we have chosen, as well as the results of the project.

Chapter 5, "Conclusions and future work", we will present the conclusions of the study and list possible lines of future work.

Chapter 7

Conclusions and Future Work

7.1. Conclusions

In this project we focused on finding a way to use the latest technological advances to be able to help people suffering from diseases that cause low vision. To do this, we first carried out a study of these, and of existing solutions (both traditional and recent) to solve this type of problem. The idea was to combine the range of possibilities offered by wearable devices, such as **Smart Glasses**, which will gradually be integrated and normalised in everyday use, with the use of the **AI**, using related models such as neural networks, to obtain an Android application capable of carrying out this task.

The result has been an Android application that, using the camera of the smart glasses, is able to recognise the pose of the person in front of the user who is wearing them. In addition, a series of dots are drawn to reflect their skeleton and a voice description is provided. With these functionalities, our application offers a visual and auditory aid especially useful for people suffering from problems related to low vision.

With this work we aim to demonstrate the great potential of all new technologies and all the possibilities they offer and how we can use them to find novel solutions to everyday problems.

7.2. Future Work

As future work and possibilities for improvement, we propose the following options:

- Add more poses to our model, as our project is fully scalable, the possibilities are endless.
- Simplify much more the training mode, to allow users to add new custom poses without the need of any technical knowledge.
- Implement a collaborative system where the user community can share and validate new poses, creating a collective database for example.
- Investigate integration with other sensors (such as thermal or depth cameras) to improve accuracy in various lighting conditions.
- Explore combining pose detection with spatial navigation systems for mobility assistance, e.g. to detect steps.

We believe that our project has enormous potential for growth and evolution. The proposed improvements will allow us to transform this solution into an even more complete and accessible tool, perfectly adapted to the daily needs of people with low vision. Although our main objective remains clear: to maintain simplicity of use while significantly improving the quality of life of our users.

Capítulo 8

Contribuciones Personales

8.1. Pedro Luis Cuadra Carrion

Mi labor en este trabajo ha ido desde la investigación hasta la implementación de código y entrenamiento de modelos de IA. Hay que decir que en nuestro caso es complicado definir las aportaciones personales en algunos aspectos, ya que en muchas ocasiones hemos trabajado mano a mano y no podemos hacer una separación clara de muchos apartados ya que no están del todo bien definidas, pero voy a intentar detallar mi rol lo más claramente posible.

Lo primero que hice fue aprender cómo usar Android Studio y programar con Kotlin, ya que ninguno de los tres miembros teníamos experiencia previa con estas tecnologías. Creamos cada uno por nuestra cuenta un primer prototipo de aplicación simplemente con un par de botones e implementaciones visuales para ir familiarizándonos con dichas tecnologías.

Una vez hecho esto y habiendo investigado cómo íbamos a proceder con el resto del proyecto, implementé la primera versión de nuestro modelo de IA de reconocimiento de poses, mediante la implementación de un script en el mismo Android Studio que extraía las coordenadas de 33 puntos claves del cuerpo de la persona de la imagen usando Pose Detection de Google ML Kit, que posteriormente y tras varios cambios fue lo que se acabó convirtiendo en el modo entrenamiento que está disponible en la versión final de la aplicación.

Una vez extraídos los puntos clave de todas las imágenes de la carpeta que habíamos seleccionado, este código generaba un CSV con las coordenadas de cada uno de ellos y sus correspondientes etiquetas para su posterior procesamiento en Google Colab para el entrenamiento del modelo de IA.

Estas imágenes las obtuve separando en frames un video mío realizando cada pose desde distintos ángulos y perspectivas para propiciar el correcto entrenamiento. Lo realicé mediante un sitio web especializado en esto (Online Convert, 2025), aunque posteriormente acabamos implementando dicho funcionamiento en nuestro propio código (modo entrenamiento de la versión final).

Una vez obtenido el CSV, implementé nuestra primera versión en Colab del modelo entrenado. En esta versión todavía no habíamos llegado a la conclusión de que era mejor

implementarlo usando ángulos y distancias entre puntos, por lo que simplemente lo entrené en base a las 33 coordenadas de puntos clave de cada imagen. Dividí los datos en conjunto de entrenamiento y de prueba, añadí la capa de entrada (que no es una capa "oculta" propiamente dicha), pero define la dimensión de los datos de entrada: 66 neuronas, por los 33 pares de coordenadas X, Y), creé dos capas ocultas y, por último, una capa de salida. Con esto entrené el modelo y, una vez entrenado, lo convertí a formato .tflite para facilitar así su integración en nuestra aplicación.

Acto seguido lo integré en nuestra aplicación en Android Studio y, aunque eran mejorables, por lo menos pude obtener resultados. El modelo era capaz de reconocer (con poca exactitud) la pose y mostrarla en pantalla en nuestro prototipo, por lo que decidimos que este era el camino correcto a seguir y que de ese momento en adelante lo que debíamos hacer era centrarnos en mejorar este entrenamiento del modelo.

Hice también varias pruebas con otras tecnologías antes de llegar a la conclusión final. Implementé un modelo usando Teachable Machine (Teachable Machine, 2025), una herramienta gratuita de Google para entrenar modelos de IA (clasificación de imágenes, audio o posturas) sin código, mediante una interfaz web sencilla. Sin embargo, a pesar de que los resultados no fueron del todo negativos, estaban lejos de la precisión que queríamos alcanzar en nuestra versión final, por lo que llegué a la conclusión de que la mejor opción era usar Google Colab para entrenar el modelo pasándole los datos en formato CSV y TensorFlow Lite para integrarlo en nuestra aplicación porque a pesar de ser menos intuitivo, es mucho más personalizable. Mediante la implementación en Colab teníamos un abanico de posibilidades mucho mayor a la hora de modificar parámetros y probar distintas implementaciones de la red neuronal para el correcto entrenamiento de nuestro modelo.

Investigué acerca de posibles datasets que pudiésemos usar pero llegamos a la conclusión de que la mejor opción era hacerlo nosotros mismos, ya que ninguno se ajustaba a nuestras necesidades específicas para el proyecto, así que me grabé vídeos realizando cada una de las poses para poder así separarlos en frames y ampliar nuestro dataset.

Contribuí también a la hora de realizar la integración con las Smart Glasses con nuestra aplicación, realizando una investigación exhaustiva de la documentación de las Rokid Air Glasses. Tras varios intentos, pruebas y problemas; finalmente conseguimos llevar a cabo dicha integración y logramos crear una primera versión funcional.

También aporté bastante al desarrollo de esta memoria, ocupándome del capítulo 1 (Introducción), del capítulo 2 (Estado de la Cuestión), del capítulo 5 (Conclusiones y trabajo futuro), del capítulo 6 (Introduction) y del capítulo 7 (Conclusions and Future Work); así como del Resumen, Abstract y mi correspondiente parte de la Bibliografía.

Y con esto concluye, de forma resumida, mi aportación personal a este proyecto. Como he dicho al principio, creo que es complicado atribuirse de manera personal muchas cosas, ya que ha sido un proyecto colaborativo en el que hemos trabajado muchas veces de manera conjunta. Al fin y al cabo, el buen resultado final de este trabajo se debe precisamente a esta integración de perspectivas y habilidades complementarias de los tres. Aun así, espero haber podido describir mi participación de la forma más clara posible.

8.2. Álvaro Velasco García

Para iniciar nuestro trabajo hicimos una labor de estudio. Antes de comenzar con las labores prácticas de programación debíamos informarnos sobre la naturaleza del proyecto, los requisitos y necesidades, las herramientas disponibles... Este conocimiento nos permitirá tomar mejores decisiones durante el trabajo. Para ayudarnos con esto, los tutores nos facilitaron algunos enlaces de interés: el manual oficial de Android, documentación sobre las Smart Glasses y librerías específicas de machine learning para Android.

Comencé aprendiendo sobre el desarrollo en Android. Para ello, primero descargué e instalé Android Studio, el IDE recomendado por Google para el desarrollo de aplicaciones en Android. Dediqué los primeros días a explorar su interfaz y a consultar el manual proporcionado por los tutores. Tras evaluar las opciones, optamos por usar Kotlin como lenguaje de programación, dado su respaldo oficial y nuestra experiencia previa en Java. La transición resultó fluida, permitiéndonos asimilar rápidamente las bases de Kotlin.

Los tutores nos propusieron dos alternativas para la detección de poses en imágenes: ML Kit y MediaPipe. Tras comparar sus ventajas y limitaciones, decidimos integrar ML Kit en nuestro proyecto. La importación e implementación inicial en Android Studio fue sencilla, por lo que comenzamos a explorar sus capacidades.

Nuestro primer objetivo práctico fue crear una versión inicial de la aplicación capaz de capturar vídeo en tiempo real, analizar cada fotograma con ML Kit y superponer el esqueleto detectado sobre la imagen. Para el acceso a la cámara utilicé la librería CameraX de Android. Tras leer la documentación y ejemplos oficiales, integré la vista previa de la cámara en el proyecto, solicité programáticamente los permisos de cámara y almacenamiento, y verifiqué que la imagen se mostrase correctamente en pantalla. En este paso resolví un desafío inicial: la conversión de cada fotograma (media.Image) en un objeto InputImage compatible con ML Kit. Ajusté las dimensiones y orientaciones hasta lograr que, al procesar una foto, los puntos esqueléticos que devolvía ML Kit quedasen perfectamente superpuestos sobre la imagen.

Con esa base, pasé a la puesta en marcha de ML Kit para la detección de poses. Configuré la dependencia en el Gradle, importé el módulo de visión y probé distintos modos de detección hasta encontrar el flujo más estable para nuestra aplicación. Una vez completada esta etapa, teníamos una versión funcional que mostraba, en tiempo real, un esqueleto articulado sobre la imagen de la cámara.

Una vez lograda la detección, implementé un sistema alternativo de clasificación de poses sin inteligencia artificial, calculando ángulos clave del cuerpo y usándolos para distinguir entre las diferentes poses. Este prototipo nos permitió evaluar los resultados y discutir su escalabilidad en la reunión con los tutores. Aunque reconocimos sus limitaciones, la experiencia fue valiosa para aclarar dudas y definir la siguiente fase.

Decidimos entrenar un modelo de IA que, a partir de la información de puntos clave, clasificara las poses. Tras estudiar diversas librerías, seleccionamos TensorFlow por su robustez y compatibilidad con Android. Analizamos varias arquitecturas y concluimos que una red neuronal era apropiada como punto de partida; asimismo, preveíamos explorar Random Forest más adelante.

Realicé un estudio intensivo sobre redes neuronales mediante vídeos didácticos y la

documentación oficial de TensorFlow. Abordé temas como la estructura de capas, backpropagation, funciones de activación y métricas de evaluación, construyendo una base sólida para el diseño y entrenamiento del modelo.

El siguiente gran bloque de trabajo correspondió a la generación de nuestro propio dataset. Tras descartar la existencia de repositorios públicos que recogieran exactamente los datos y formato que buscábamos, diseñé y desarrollé una versión alternativa de la aplicación enfocada a la grabación de vídeo y extracción de fotogramas. Aproveché parte del código base aportado por Pedro —que ya importaba una carpeta de imágenes y generaba un CSV—, pero reescribí y extendí esa funcionalidad para ofrecer a los usuarios una experiencia más cómoda: selección de la pose, cámara frontal o trasera, duración de la grabación y una cuenta atrás de tres segundos para garantizar que cada pose quedase centrada y estable en pantalla antes de iniciar la captura. Tras pulsar "Generar dataset", el vídeo se dividía automáticamente en frames, cada uno se procesaba con ML Kit para extraer las coordenadas de 33 puntos clave y todo se volcaba en un archivo CSV. Con este sistema, cada uno de los tres miembros del equipo grabó ocho vídeos de 1–2 minutos, variando distancia y ángulo respecto a la cámara, lo que me permitió generar decenas de miles de muestras para el entrenamiento inicial.

Para unificar y analizar esos CSV individuales, programé varios scripts en JavaScript. El primero combinaba los 24 archivos generados (tres miembros × ocho poses) en uno único, añadiendo la cabecera necesaria con los nombres de las columnas. El segundo recorría ese archivo para calcular estadísticas básicas: número total de instancias, recuento por pose y valores máximos, mínimos y medios de todas las coordenadas. Este análisis de calidad de datos me permitió detectar desequilibrios o valores anómalos antes de entrar en la fase de entrenamiento.

Con el dataset consolidado, abrí un notebook en Google Colab y realicé el primer entrenamiento de una red neuronal en TensorFlow. Tomé como punto de partida el notebook inicial que había creado Pedro, ajusté la carga de datos al formato CSV que habíamos generado, y añadí gráficas de precisión y pérdida, así como una matriz de confusión. Me dediqué a experimentar con hiperparámetros: número de capas ocultas, neuronas por capa, función de activación, tasa de aprendizaje y número de épocas. Cada cambio era seguido de nuevas ejecuciones, observación de resultados y consulta de la documentación oficial para entender la repercusión de cada ajuste. Rápidamente detecté un sobreajuste: la accuracy alcanzaba el 99 % en apenas 15 épocas. Para enfrentarlo, introduje capas de tipo Dropout en la arquitectura, pero la mejora fue marginal.

Decidí entonces optimizar el dataset en origen. Primero desarrollé un script que reducía el dataset descartando cinco de cada seis filas, manteniendo la diversidad y relevancia de los datos. Esto además aceleró el entrenamiento y dio lugar a una precisión más realista. Posteriormente, transformé las coordenadas crudas en ocho ángulos corporales normalizados, simplificando la entrada de 66 variables a solo 8 y mejorando la estabilidad de la clasificación. Para ello, modifiqué la aplicación de generación de dataset para que, en lugar de volcar X e Y, calculase ángulos entre tríos de puntos esqueléticos según la documentación de ML Kit.

También decidí explorar un segundo modelo: Random Forest. Tras preparar los datos adecuados, creé un nuevo notebook de TensorFlow Decision Forests y apliqué Grid Search para ajustar num_trees y max_depth. Medí tanto la precisión de validación como el tiempo de entrenamiento y seleccioné parámetros equilibrados. Además, analicé la importancia de

variables (INV_MEAN_MIN_DEPTH y SUM_SCORE) y detecté que los ángulos de cadera y rodilla aportaban muy poco, por lo que decidimos eliminarlos en la siguiente iteración del dataset. Aunque no pudimos exportar el modelo Random Forest a TFLite para su integración en la app, este experimento amplió nuestro conocimiento y validó decisiones de diseño de features.

De vuelta a la red neuronal y, siguiendo las recomendaciones de nuestros tutores, decidimos incorporar al dataset cinco distancias clave, que normalicé primero respecto a la distancia entre los hombros y luego comprimí entre 0 y 1 usando una función sigmoide. Tras reentrenar la red neuronal, obtuvimos la mejor precisión hasta la fecha. Este conjunto de datos final constituyó la base de nuestro modelo.

Además estudié en profundidad las técnicas de transfer learning y fine-tuning, e importé un modelo preentrenado en un nuevo notebook. Tras congelar sus capas, sustituir la salida y entrenar brevemente, apliqué fine-tuning para terminar de ajustar todos los pesos. Las métricas mostraron menor precisión que nuestro modelo entrenado desde cero, pero la experiencia resultó instructiva.

Por recomendación de los tutores, para la versión final de la aplicación decidimos usar como base la demo proporcionada por Rokid. Partiendo de la demo, desacoplé cuidadosamente mi código de detección y clasificación de poses del proyecto original y lo adapté al módulo de cámara de las gafas. Tras múltiples pruebas refactoricé el proyecto, eliminé módulos de voz y hardware que no usábamos, y añadí un chequeo de compatibilidad: al arrancar la app, se compara el modelo del smartphone con la lista oficial de dispositivos Rokid; si no coincide, se muestra un mensaje informativo y se deshabilitan las funciones avanzadas.

Para enriquecer la experiencia de usuario, implementé dos canales de feedback. El primero, visual, dibuja en AR el esqueleto detectado en el visor de las gafas, pintando círculos en cada PoseLandmark y líneas entre articulaciones, con colores y grosores optimizados tras iteraciones de prueba. El segundo, auditivo, emplea un sistema de text-to-speech usando una librería de Android. Configuré el método speak() en modo QUEUE_FLUSH para garantizar inmediatez y permití la selección de idioma (español o inglés).

Dado que notificar cada detección podía resultar molesto, diseñé un sistema de estabilización de poses basado en un buffer de 1000 ms y un umbral del 60 %. Cada segundo, el buffer mantiene solo las poses recientes, calcula la más repetida y, si supera el umbral mínimo y difiere de la última notificada, la emite; en caso contrario, omite el aviso. Tras afinar ventana y umbral mediante pruebas reales, conseguimos eliminar notificaciones erráticas y ofrecer un feedback consistente.

Finalmente, implementé un módulo completo de accesibilidad que permite elegir el de modo de color, tamaño de texto e idioma. Diseñé la persistencia de preferencias para que los ajustes se apliquen y recuerden automáticamente al iniciar la app, asegurando una interfaz inclusiva y cómoda para personas con baja visión. Esta funcionalidad implicó investigar actualizaciones dinámicas de la interfaz, y resolver errores de sincronización y persistencia de preferencias, lo que implicó un exhaustivo análisis y depuración.

Disponiendo de las Smart Glasses en la facultad, realizamos pruebas frecuentes, recibiendo feedback inmediato de los tutores. Este ciclo de validación aceleró la fase final de desarrollo.

Mi responsabilidad en la redacción de la memoria abarcó el apartado 4, que recoge el desarrollo de la aplicación y los resultados. Además adapté figuras y referencias, y garanticé que la exposición fuera clara, rigurosa y fiel al trabajo realizado.

Como reflexión final, este proyecto ha sido una experiencia formativa y enriquecedora. Me ha llevado a enfrentarme a retos técnicos, desde la captura y normalización de datos hasta la creación de algoritmos de estabilización y accesibilidad, y a integrar distintos campos como la visión por computadora, el aprendizaje automático, el desarrollo de apps móviles y el diseño de interfaces AR. He aprendido a iterar rápidamente, a documentar cada paso y a tomar decisiones fundamentadas, logrando finalmente un prototipo sólido, usable y accesible para personas con baja visión.

8.3. Samuel Álvarez Medina

Mi participación en este proyecto ha sido variada y transversal, abarcando desde el aprendizaje y la planificación inicial hasta el diseño de funcionalidades, pasando por el desarrollo técnico y la integración de los dispositivos hardware. Como ya han comentado mis compañeros, muchas tareas y decisiones se han realizado en equipo y resulta complicado delimitar fronteras claras entre las aportaciones individuales, intentaré detallarlas lo más claramente posible.

Como ninguno del grupo tenía mucha experiencia en desarrollo para Android, una de mis primeras tareas fue aprender desde cero a utilizar herramientas como Android Studio y el lenguaje Kotlin. No fue un proceso inmediato, pero me resultó muy gratificante ir viendo cómo podía trasladar ideas a algo visual y funcional.

Me centré en comprender cómo podíamos diseñar una interfaz intuitiva y funcional para personas con baja visión, considerando que esta era una parte esencial de nuestro objetivo. Con este conocimiento, participé activamente en el desarrollo de la interfaz de la aplicación, buscando no solo que fuese intuitiva, sino que además facilitara el proceso de pruebas y ajustes durante el desarrollo. Incorporamos funcionalidades que nos permitían trabajar de forma más eficiente: pruebas rápidas, controles accesibles, y elementos visuales que ayudaban a interpretar las salidas del modelo.

Durante esas primeras semanas también estuve investigando diferentes opciones para desarrollar el modelo de inteligencia artificial que necesitábamos: desde el uso de ML Kit de Google, que acabó siendo la opción principal, hasta alternativas como MediaPipe evaluando ventajas, limitaciones y compatibilidad con Android.

Gracias a la primera versión del modelo que desarrolló Pedro, me resultó más fácil familiarizarme con Google Colab y empezar a trabajar en cómo podíamos optimizarlo en lugar de partir de cero. Aunque Álvaro y Pedro se encargaron de la parte más técnica del entrenamiento, yo contribuí al entendimiento global del funcionamiento de los modelos, su rendimiento en tiempo real, sus limitaciones prácticas, llevando a cabo pruebas con los diferentes datasets y modelos que ibamos desarrollando. Me aseguré de que el equipo tuviera una visión conjunta y coherente del modelo de detección y clasificación de poses.

Antes de seguir añadiendo nuevas versiones sin una dirección clara, al retomar el proyecto despues de los examenes de enero definimos bien los puntos clave a mejorar. Una de mis tareas en esa etapa fue llevar a cabo un análisis detallado de las poses que se podían

incluir en el entrenamiento del modelo. Esto implicó no solo seleccionar poses relevantes, sino también definir cómo estructurar el dataset para reflejar las distintas perspectivas que proporciona cada pose.

Participé activamente en la obtención de los datos grabando videos que sirvieron como base para el dataset. También colaboré en la optimización del mismo, eliminando puntos que no aportaban información relevante —es decir, datos vacíos o poco significativos—, y realizando pruebas para determinar qué combinaciones de ángulos y distancias entre puntos eran las más adecuadas para nuestro proyecto. Esta fase fue clave para asegurar la calidad del dataset y su utilidad en los análisis posteriores.

Una vez cerrado el diseño del dataset y con el modelo final ya entrenado, participé activamente en la integración con las Smart Glasses, algo que no fue sencillo. Para la integración utilizamos como punto de partida la demo de Rokid, la cual nos proporcionarón nuestros tutores y nos facilitó el proceso. Me enfoqué en la revisión e interpretación de partes del código, depurar errores y realizar pruebas para asegurarme de que la app respondía correctamente al usarla con las gafas. Esto incluyó tanto ajustes técnicos como validaciones en situaciones reales, tratando de asegurar una experiencia fluida.

Todas las pruebas con las gafas ROKID Pro se llevaron a cabo de manera presencial en la Facultad, ya que era el único lugar donde se podían utilizar. Esta circunstancia nos permitió contar siempre con la presencia de los tutores, quienes nos orientaban y aconsejaban durante las sesiones, lo que nos proporcionó un trabajo muy productivo y eficaz.

En cuanto a la memoria del proyecto, fui responsable de redactar el capítulo 3 (Planificaión y Recursos), donde se detalla la planificación del proyecto y las herramientas empleadas para su desarrollo. Se justifica la elección de tecnologías por su utilidad en la colaboración, documentación y entrenamiento de modelos de IA. Además, se presentan los recursos hardware utilizados, en especial las Smart Glasses Rokid Air Pro. También se abordan los desafíos enfrentados y cómo se resolvieron para cumplir con los objetivos de funcionalidad, eficiencia y accesibilidad del sistema desarrollado

También llevé un seguimiento de las reuniones con los tutores y de las internas, tomando notas que después sirvieron para documentar las decisiones importantes del desarrollo.

Como conclusión personal, este proyecto ha supuesto una oportunidad única para integrar conocimientos muy diversos: desde el desarrollo de aplicaciones móviles hasta la iniciación, al menos por mi parte, en el mundo de la inteligencia artificial, además de llevar todo proyecto a cabo teniendo en cuenta las accesibilidades un proyecto de estas características. He aprendido a trabajar en equipo, a compartir responsabilidades técnicas y a resolver problemas de manera creativa. El hecho de haber podido probar nuestra solución en dispositivos reales y recibir feedback inmediato durante todo el desarrollo del proyecto por parte de nuestros tutores ha sido clave para lograr un prototipo funcional, con potencial real de aplicación.

Bibliografía

Alcocer Óptica. ¿qué es baja visión? 2024. Disponible en https://www.alcoceroptica.com/servicios/baja-vision/#1574784659579-4da79f6d-486c (último acceso, Accessed: 2025-04-22).

ATRIA. Gafas inteligentes, todo lo que pueden hacer. 2025. Disponible en https://atriainnovation.com/blog/gafas-inteligentes/ (último acceso, Accessed: 2025-05-03).

AWS. ¿qué es iot (internet de las cosas)? 2024. Disponible en https://aws.amazon.com/es/what-is/iot/ (último acceso, Accessed: 2025-02-17).

Barañano Baja Visión. PatologÍas frecuentes en baja visiÓn. 2025. Disponible en https://baja-vision.org/servicios-de-baja-vision-baranano/baja-vision-caracteristicas/patologias-frecuentes-en-baja-vision/# (último acceso, Accessed: 2025-03-06).

Breiman, L. Random forests. *Machine Learning*, vol. 45(1), páginas 5–32, 2001.

Cottet. Gafas de sol inteligentes ray-ban meta rw4006 601/71. 2025a. Disponible en https://www.cottet.com/es/gafas-sol/gafas-de-sol-unisex-ray-ban-meta-rw4006-60171-negro-verde-50.html (último acceso, Accessed: 2025-05-12).

Cottet. ¿qué son las gafas inteligentes? 2025b. Disponible en https://www.cottet.com/es/blog/news/que-son-las-gafas-inteligentes (último acceso, Accessed: 2025-05-12).

Fundación Porib. Carga de la pérdida de visión y ceguera en españa. 2025. Disponible en https://porib.com/carga-de-la-perdida-de-vision-y-ceguera-en-espana/ (último acceso, Accessed: 2025-04-03).

Google. Google glass. 2025. Disponible en https://www.google.com/glass/photography/ (último acceso, Accessed: 2025-04-27).

Google Cloud. ¿qué es el aprendizaje automático (aa)? 2025. Disponible en https://cloud.google.com/learn/what-is-machine-learning?hl=es-419 (último acceso, Accessed: 2025-04-09).

Google LLC. Arquitectura de apps: Capa de datos - datastore. https://developer.android.com/topic/libraries/architecture/datastore?hl=es-419, 2025a. Consultado el 5 de abril de 2025.

Google LLC. CameraX Overview – Android Developers. 2025b. Disponible en https://developer.android.com/media/camera/camerax?hl=es-419 (último acceso, Accessed: 2024-10-12).

Google LLC. Cómo guardar datos simples con sharedpreferences. https://developer.android.com/training/data-storage/shared-preferences?hl=es-419, 2025c. Consultado el 2 de abril de 2025.

Google ML Kit. Aprendizaje automático destinado a desarrolladores de apps para dispositivos móviles. 2024a. Disponible en https://developers.google.com/ml-kit (último acceso, Accessed: 2025-02-17).

Google ML Kit. Detección de poses. 2024b. Disponible en https://developers.google.com/ml-kit/vision/pose-detection?hl=es-419 (último acceso, Accessed: 2025-02-17).

IBM. ¿qué es un random forest? 2025a. Disponible en https://www.ibm.com/es-es/think/topics/random-forest (último acceso, Accessed: 2025-05-04).

IBM. ¿qué son las redes neuronales? 2025b. Disponible en https://www.ibm.com/es-es/topics/neural-networks (último acceso, Accessed: 2025-05-04).

Inesdi. Random forest, la gran técnica de machine learning. 2025. Disponible en https://www.inesdi.com/blog/random-forest-que-es/ (último acceso, Accessed: 2025-04-03).

Instituto EITCI. ¿qué es tensorflow lite y cuál es su propósito? 2025. Disponible en https://es.eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/programming-tensorflow/tensorflow-lite-for-android/examination-review-tensorflow-lite-for-android/what-is-tensorflow-lite-and-what-is-its-p (último acceso, Accessed: 2025-04-22).

Kingma, D. P. y Ba, J. Adam: A method for stochastic optimization. *CoRR*, vol. abs/1412.6980, 2014.

MediaPipe. Pose landmark detection guide. 2025. Disponible en https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker (último acceso, Accessed: 2025-02-19).

Meta. Ray-ban meta smart glasses. 2023. Disponible en https://www.meta.com/es/en/ai-glasses/?srsltid=AfmBOor2-KBrLDHkwUR3Wel-rc5t4JiM_1dgDJSLHfcdV5rBOe5YLXPh (último acceso, Accessed: 2025-03-02).

Microsoft. Microsoft hololens. 2025. Disponible en https://learn.microsoft.com/es-es/hololens/ (último acceso, Accessed: 2025-05-01).

Mácula Retina, Dra. Shahid. Discapacidad visual: comprender su impacto psicosocial. 2012. Disponible en https://www.macula-retina.es/discapacidad-visual-comprender-su-impacto-psicosocial/ (último acceso, Accessed: 2025-05-13).

OMS. World report on vision. 2019. Disponible en https://www.who.int/publications/i/item/9789241516570 (último acceso, Accessed: 2025-04-12).

OMS. Informe sobre visión. 2023. Disponible en https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment (último acceso, Accessed: 2025-04-08).

Online Convert. Conversor de mp4 a jpg. 2025. Disponible en https://imagen.online-convert.com/es/convertir/mp4-a-jpg (último acceso, Accessed: 2025-03-25).

Oracle. ¿qué es el iot? 2025. Disponible en https://www.oracle.com/es/internet-of-things/ (último acceso, Accessed: 2025-03-11).

Orcam. Empowering accessibility with ai. 2024. Disponible en https://www.orcam.com/en-us/home (último acceso, Accessed: 2025-05-05).

Pan, S. J. y Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, vol. 22(10), páginas 1345–1359, 2010.

Roeder, L. Netron: Visualizer for neural network, deep learning and machine learning models. https://netron.app/, 2025. Consultado el 8 de diciembre de 2024.

Rokid, Inc. Axr phone kotlin sample (demo). Versión v1.0.4, descarga en https://ota-g.rokidcdn.com/toB/Rokid_Glass/SDK/AXR_SDK/forPhone/AXR_Phone_Kfotlin_sample_v1.0.4.zip, 2025. Aplicación demo desarrollada en Kotlin.

Salud Vision. Ayudas baja visión. 2023. Disponible en https://www.baja-vision.es/ayudas-baja-vision/ (último acceso, Accessed: 2025-04-30).

Santana-Mansilla, P., Lescano, G. y Costaguta, R. Accesibilidad de aplicaciones móviles para discapacitados visuales: Problemas y estrategias de solución. En *44 JAIIO – STS 2015 (2.º Simposio Argentino sobre Tecnología y Sociedad)*, páginas 356–375. JAIIO, Buenos Aires, Argentina, 2015.

Sergio Pérez. Data machine learning visualization. 2022. Disponible en https://blog.damavis.com/deteccion-de-poses-humanas-mediante-deep-learning/#:~:text=La%20detecci%C3%B3n%20de%20pose%20humana,a%20partir%20de%20una%20imagen. (último acceso, Accessed: 2025-04-14).

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. y Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, vol. 15, páginas 1929–1958, 2014.

Teachable Machine. Train a computer to recognize your own images, sounds, & poses. 2025. Disponible en https://teachablemachine.withgoogle.com/ (último acceso, Accessed: 2025-03-12).

TensorFlow. Tensorflow lite. 2021. Disponible en https://www.tensorflow.org/lite/guide?hl=es-419 (último acceso, Accessed: 2025-04-16).

tuOptometrista. Ayudas con sistema óptico telescopio (ts y tms). 2017. Disponible en https://www.tuoptometrista.com/solucion/ayudas-con-sistema-optico-telescopio-ts-y-tms/ (último acceso, Accessed: 2025-02-25).