



## **ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**Trabajo fin de Grado**

**Sistema Wearable de Detección de Caídas**

**Grado en Ingeniería Informática - Ingeniería de Computadores**

**Realizado por**  
**Manuel Martínez Bernal**

**Dirigido por**  
**Alberto Jesús Molina Cantero**

**Departamento**  
**Tecnología Electrónica**

**Sevilla, 29 de Mayo de 2024**

# ÍNDICE

|   |           |
|---|-----------|
| <b>Resumen.....</b>   | <b>5</b>  |
| <b>Agradecimientos.....</b>                                 | <b>6</b>  |
| <b>Introducción.....</b>                                    | <b>7</b>  |
| <b>Objetivos.....</b>                                       | <b>8</b>  |
| Obj1: Desarrollo y Optimización de la Red Neuronal.....     | 8         |
| Obj2: Pruebas de Comunicación y Alerta.....                 | 9         |
| Obj3: Prototipo Hardware y Realización de Pruebas.....      | 9         |
| <b>Estado del Arte.....</b>                                 | <b>10</b> |
| Introducción al Estado del Arte.....                        | 10        |
| Dispositivos y Tecnologías para la Detección de Caídas..... | 11        |
| Dispositivos Portátiles.....                                | 12        |
| Dispositivos de Ambiente.....                               | 13        |
| Dispositivos de Vídeo.....                                  | 15        |
| Técnicas y Algoritmos para la Detección de Caídas.....      | 16        |
| Algoritmos Basados en Umbrales.....                         | 16        |
| Redes Neuronales.....                                       | 17        |
| Comparativa y Evaluación de Tecnologías y Métodos.....      | 18        |
| Dispositivos Portátiles.....                                | 19        |
| Dispositivos Integrados en el Entorno.....                  | 19        |
| Análisis de Video.....                                      | 19        |
| Conclusiones del Estado del Arte.....                       | 20        |
| Diversidad de Enfoques Tecnológicos.....                    | 21        |
| Relevancia para el Proyecto de TFG.....                     | 21        |
| <b>Planificación.....</b>                                   | <b>22</b> |
| Fase 1 - Investigación y Planificación.....                 | 22        |
| Tarea 1.1 - Definición del Alcance.....                     | 22        |
| Tarea 1.2 - Investigación de Tecnologías.....               | 22        |
| Tarea 1.3 - Estudio del Estado del Arte.....                | 22        |
| Fase 2 - Desarrollo de Redes Neuronales.....                | 23        |
| Tarea 2.1 - Pruebas de Concepto.....                        | 23        |
| Tarea 2.2 - Entrenamiento de Redes Neuronales.....          | 23        |
| Tarea 2.3 - Evaluación y Ajustes.....                       | 23        |
| Hito 1 - Desarrollo y Optimización de la Red Neuronal.....  | 23        |
| Fase 3 - Desarrollo del Dispositivo Wearable.....           | 24        |
| Tarea 3.1 - Selección del Hardware.....                     | 24        |
| Tarea 3.2 - Desarrollo del Software.....                    | 24        |
| Tarea 3.3 - Integración de la Red Neuronal.....             | 24        |
| Hito 2 - Prototipo Hardware y Realización de Pruebas.....   | 24        |
| Fase 4 - Pruebas y Validación.....                          | 25        |
| Tarea 4.1 - Pruebas de Redes Neuronales.....                | 25        |
| Tarea 4.2 - Pruebas del Dispositivo.....                    | 25        |
| Tarea 4.3 - Validación en Entornos Reales.....              | 25        |

|   |           |
|---|-----------|
| Hito 3 - Pruebas de Comunicación y Alerta.....                  | 25        |
| Fase 5 - Documentación y Entrega.....                           | 26        |
| Tarea 5.1 - Documentación Técnica.....                          | 26        |
| Tarea 5.2 - Manuales de Usuario.....                            | 26        |
| Tarea 5.3 - Entrega Final del Proyecto.....                     | 26        |
| <b>Metodología de Trabajo.....</b>                              | <b>28</b> |
| Pruebas de Concepto.....  | 28        |
| Herramientas Utilizadas.....                                    | 28        |
| Metodología.....  | 29        |
| Procedimientos.....   | 29        |
| Entrenamiento de Redes Neuronales.....                          | 30        |
| Herramientas Utilizadas.....                                    | 30        |
| Metodología.....  | 32        |
| Procedimientos.....   | 32        |
| Evaluación y Ajustes.....                                       | 37        |
| Herramientas Utilizadas.....                                    | 38        |
| Metodología.....  | 38        |
| Procedimientos.....   | 38        |
| Selección del Hardware.....                                     | 52        |
| Herramientas Utilizadas.....                                    | 52        |
| Metodología.....  | 53        |
| Procedimientos.....   | 54        |
| Desarrollo del Software.....                                    | 55        |
| Herramientas Utilizadas.....                                    | 55        |
| Metodología.....  | 57        |
| Procedimientos.....   | 58        |
| Integración de la Red Neuronal.....                             | 61        |
| Herramientas Utilizadas.....                                    | 61        |
| Metodología.....  | 61        |
| Procedimientos.....   | 61        |
| <b>Pruebas, Resultados y Discusión.....</b>                     | <b>64</b> |
| Pruebas de las Redes Neuronales Entrenadas.....                 | 64        |
| Entrenamiento de las Redes Neuronales.....                      | 64        |
| Selección del Mejor Umbral de Clasificación.....                | 65        |
| Problemas.....  | 66        |
| Pruebas del Funcionamiento del Dispositivo Wearable.....        | 66        |
| Visualización de la Hora y Funciones Básicas.....               | 66        |
| Recopilación de Datos del Sensor.....                           | 67        |
| Funcionamiento de la interfaz de Caídas.....                    | 68        |
| Transmisión de Datos al Servidor y Recepción de Respuestas..... | 68        |
| Problemas.....  | 69        |
| Pruebas del Servidor y la Notificación por WhatsApp.....        | 69        |
| Recepción y Procesamiento de Datos.....                         | 69        |
| Funcionamiento de las Predicciones.....                         | 72        |

|  |           |
|--|-----------|
| Problemas.....   | 73        |
| Discusión de los Resultados.....                           | 73        |
| <b>Análisis temporal.....</b>                              | <b>75</b> |
| Fase 1 - Investigación y Planificación.....                | 75        |
| Tarea 1.1 - Definición del Alcance.....                    | 75        |
| Tarea 1.2 - Investigación de Tecnologías.....              | 75        |
| Tarea 1.3 - Estudio del Estado del Arte.....               | 75        |
| Fase 2 - Desarrollo de Redes Neuronales.....               | 76        |
| Tarea 2.1 - Pruebas de Concepto.....                       | 76        |
| Tarea 2.2 - Entrenamiento de Redes Neuronales.....         | 76        |
| Tarea 2.3 - Evaluación y Ajustes.....                      | 76        |
| Hito 1 - Desarrollo y Optimización de la Red Neuronal..... | 77        |
| Fase 3 - Desarrollo del Dispositivo Wearable.....          | 77        |
| Tarea 3.1 - Selección del Hardware.....                    | 77        |
| Tarea 3.2 - Desarrollo del Software.....                   | 78        |
| Tarea 3.3 - Integración de la Red Neuronal.....            | 78        |
| Hito 2 - Prototipo Hardware y Realización de Pruebas.....  | 78        |
| Fase 4 - Pruebas y Validación.....                         | 79        |
| Tarea 4.1 - Pruebas de Redes Neuronales.....               | 79        |
| Tarea 4.2 - Pruebas del Dispositivo.....                   | 79        |
| Tarea 4.3 - Validación en Entornos Reales.....             | 79        |
| Hito 3 - Pruebas de Comunicación y Alerta.....             | 80        |
| Fase 5 - Documentación y Entrega.....                      | 80        |
| Tarea 5.1 - Documentación Técnica.....                     | 80        |
| Tarea 5.2 - Manuales de Usuario.....                       | 80        |
| Tarea 5.3 - Entrega Final del Proyecto.....                | 81        |
| <b>Análisis económico.....</b>                             | <b>84</b> |
| Costes del Desarrollo del Proyecto.....                    | 84        |
| Coste de horas de trabajo.....                             | 84        |
| Coste de Entrenamiento de la Red Neuronal.....             | 84        |
| Coste de Componentes del Prototipo.....                    | 85        |
| Coste del Producto Final.....                              | 85        |
| <b>Viabilidad.....</b>                                     | <b>86</b> |
| Reducción de costes de producción.....                     | 86        |
| Establecimiento de un precio competitivo.....              | 86        |
| Análisis del retorno de la inversión.....                  | 87        |
| <b>Aplicabilidad.....</b>                                  | <b>87</b> |
| Consideraciones de Personal.....                           | 87        |
| Mejoras Tecnológicas y Modelo de Negocio.....              | 88        |
| Proyecciones Financieras.....                              | 88        |
| <b>Conclusiones.....</b>                                   | <b>89</b> |
| <b>Bibliografía.....</b>                                   | <b>91</b> |
| <b>Anexos.....</b>   | <b>93</b> |
| Anexo 1 - Manual de Configuración del Entorno Virtual..... | 93        |

|  |     |
|--|-----|
| Instalación de Anaconda.....   | 93  |
| Creación del Entorno Virtual.....  | 96  |
| Configuración de Dependencias.....   | 97  |
| Resolución de Problemas Comunes.....   | 99  |
| Anexo 2 - Configuración Arduino IDE y características de la placa de desarrollo. | 101 |
| Descarga e Instalación de Arduino IDE.....                                       | 101 |
| Configuración del Entorno.....   | 101 |
| Instalación de Placas Adicionales.....   | 102 |
| Selección de Opciones de Configuración.....                                      | 102 |
| Descripción General.....   | 103 |
| Parámetros.....  | 105 |
| Anexo 3 - Entrega del Código Fuente y Documentación.....                         | 106 |
| Código Fuente Completo:.....   | 106 |
| Manual de Instrucciones - README.md:.....  | 106 |
| Instrucciones para Acceder al Código Fuente.....                                 | 106 |

# Resumen

El presente Trabajo de Fin de Grado (TFG) tiene como objetivo el desarrollo de un dispositivo wearable capaz de detectar caídas en tiempo real utilizando redes neuronales. Este proyecto está orientado principalmente a proporcionar una solución de asistencia para personas mayores o aquellas con alto riesgo de sufrir caídas, contribuyendo a mejorar su seguridad y calidad de vida.

La finalidad del proyecto es diseñar y construir un sistema que, mediante el uso de sensores inerciales (acelerómetro y giroscopio) integrados en un dispositivo wearable, recopile datos de movimientos y posturas del usuario. Estos datos son procesados por una red neuronal bilateral Long Short-Term Memory (LSTM), entrenada específicamente para identificar patrones que indiquen una posible caída.

El núcleo del proyecto reside en el desarrollo y entrenamiento de una red neuronal (RN) específica para la detección de caídas. En particular, se ha implementado una red neuronal bilateral Long Short-Term Memory (LSTM), conocida por su capacidad para capturar dependencias a largo plazo en secuencias de datos, como es el caso de los datos recopilados por los sensores inerciales del dispositivo wearable.

El dispositivo wearable, además de la detección de caídas, incluye un proceso de verificación de doble factor que notifica al usuario a través de una interfaz intuitiva, permitiéndole confirmar o cancelar la detección. En caso de confirmación, el dispositivo envía una señal a un servidor central que, a través de una API, envía una notificación de emergencia a un contacto predefinido mediante WhatsApp.

Este proyecto representa una contribución significativa en el ámbito de la salud y la seguridad, utilizando tecnologías avanzadas de sensores y aprendizaje automático para proporcionar una solución efectiva y accesible para la detección y gestión de caídas.

# Agradecimientos

Quiero expresar mi más profundo agradecimiento a todas las personas que han sido parte de mi trayectoria académica y personal, brindándome su apoyo incondicional y su amistad.

A mis amigos, les agradezco por estar siempre ahí, compartiendo conocimientos, experiencias y momentos inolvidables. Su compañerismo y apoyo han sido fundamentales para superar los desafíos académicos y seguir adelante.

A mi pareja, quiero agradecerle por ser mi mayor apoyo y mi motivación constante. Su amor y comprensión han sido mi fuerza en los momentos difíciles y mi alegría en los momentos felices.

A mis padres, les debo un agradecimiento enorme por su incondicionalidad, su sacrificio y su constante apoyo. Gracias por creer en mí, por guiarme y por enseñarme los valores que me han hecho ser quien soy hoy.

A mis profesores y mentores, les agradezco por su dedicación y sus enseñanzas. Su orientación ha sido fundamental en mi formación académica y profesional.

Finalmente, quiero agradecer a todas las personas que de alguna manera han contribuido a mi crecimiento y desarrollo. Su generosidad y amabilidad han dejado una huella imborrable en mi vida.

A todos ustedes, ¡muchas gracias!

# Introducción

La elección de explorar la detección de caídas como tema central surge de la creciente conciencia sobre los desafíos que enfrentan las personas mayores. Las caídas en esta población representan un problema significativo, ya que no solo impactan la salud física, sino que también afectan la calidad de vida y a su autonomía.

El envejecimiento conlleva una serie de cambios fisiológicos y físicos que aumentan la vulnerabilidad de las personas mayores a las caídas. Desde la disminución de la fuerza muscular hasta problemas de equilibrio y deterioro de la visión, varios factores contribuyen a la aparición de este tipo de incidentes. Las consecuencias de las caídas pueden ser graves, abarcando desde fracturas y lesiones traumáticas hasta el deterioro de la confianza y la independencia.

La detección temprana de caídas adquiere una relevancia crítica. La prontitud en la identificación de estos eventos puede marcar la diferencia entre una intervención inmediata y el riesgo de complicaciones severas. Las caídas no detectadas o sin respuesta rápida pueden llevar a un empeoramiento de las lesiones, afectar la movilidad y, en última instancia, comprometer la calidad de vida de la persona mayor.

La implementación de un sistema de detección de caídas basado en tecnología wearable no solo busca mitigar las consecuencias físicas de estos incidentes, sino también proporcionar tranquilidad y seguridad a los familiares y cuidadores. En muchos casos, las personas mayores pueden estar solas cuando ocurre una caída, y la capacidad de alertar a familiares o servicios de emergencia de manera automática puede marcar la diferencia entre una recuperación rápida y complicaciones mayores.

La notificación inmediata a familiares o cuidadores permite una intervención eficaz, ya sea a través de asistencia directa o la movilización de recursos médicos necesarios. Esto no solo acelera el tiempo de respuesta ante emergencias, sino que también contribuye a la tranquilidad emocional de los familiares, quienes pueden tomar medidas proactivas para garantizar el bienestar continuo de la persona mayor.

Por estos motivos se propone diseñar y evaluar un sistema integral de detección de caídas que no solo considere la precisión técnica de la detección, sino también el impacto directo en la salud y la calidad de vida de las personas mayores.

La aplicación de tecnologías avanzadas no solo se orienta a la prevención de lesiones físicas, sino que también busca preservar la independencia y la dignidad de una población que merece vivir de manera plena y segura en sus entornos cotidianos.

## Objetivos

En esta sección, definiremos los objetivos del proyecto. Los objetivos actúan como hitos clave de nuestro proyecto, marcando las metas específicas que buscamos alcanzar. Son fundamentales porque nos ayudan a enfocar nuestros esfuerzos y a medir nuestro progreso a lo largo del desarrollo del dispositivo para la detección de caídas.

### Obj1: Desarrollo y Optimización de la Red Neuronal

El primer objetivo del proyecto es implementar, entrenar y optimizar una red neuronal específicamente diseñada para la detección de caídas. Esto implica utilizar un conjunto de datos existente para entrenar el modelo, ajustar la arquitectura de la red y los hiperparámetros para maximizar su rendimiento, y realizar pruebas exhaustivas en diferentes escenarios para evaluar su robustez y precisión. Además, se buscará identificar áreas de mejora y realizar ajustes adicionales para aumentar aún más la precisión y la fiabilidad del modelo.

El desarrollo y la optimización de la red neuronal constituyen un paso fundamental en el proyecto, ya que la eficacia del sistema de detección de caídas dependerá en gran medida de la capacidad del modelo para identificar con precisión los patrones asociados con las caídas.

## **Obj2: Pruebas de Comunicación y Alerta**

El segundo objetivo se centra en evaluar diferentes métodos de comunicación para alertar sobre caídas detectadas por el sistema. Se considerarán opciones como el desarrollo de una aplicación móvil para notificaciones, el envío de mensajes de texto, llamadas telefónicas o mensajes instantáneos. El objetivo es seleccionar la opción más eficiente y efectiva en términos de entrega rápida y fiable de alertas en casos de detección de caídas.

Una vez seleccionado el método de comunicación adecuado, se procederá con el desarrollo completo de la opción elegida, asegurando su integración perfecta con el sistema de detección de caídas. Posteriormente, se llevarán a cabo pruebas exhaustivas para garantizar su funcionalidad y eficacia en la entrega de alertas en situaciones reales.

## **Obj3: Prototipo Hardware y Realización de Pruebas**

El tercer objetivo consiste en diseñar y desarrollar un prototipo de hardware basado en los requerimientos del sistema de detección de caídas. Esto implicará la selección y configuración de los componentes necesarios, así como la implementación del sistema completo en un dispositivo físico. Una vez construido el prototipo, se realizarán pruebas exhaustivas para garantizar su correcto funcionamiento en situaciones del mundo real.

Durante las pruebas, se identificarán posibles mejoras y ajustes necesarios tanto en el hardware como en el software para optimizar el rendimiento y la fiabilidad del sistema de detección de caídas. El objetivo final es obtener un prototipo completamente funcional y listo para su posterior implementación y despliegue en entornos reales.

# Estado del Arte

## Introducción al Estado del Arte

La detección de caídas es un área de investigación crucial en el campo de la salud y el bienestar, especialmente para personas mayores o aquellos con condiciones médicas que aumentan el riesgo de caídas. Existe interés en el desarrollo de dispositivos y tecnologías que podrían tener la capacidad de detectar automáticamente estos eventos, lo que potencialmente podría resultar en una asistencia más rápida y una reducción en los tiempos de respuesta durante situaciones de emergencia..

El presente trabajo se centra en el diseño y desarrollo de un dispositivo wearable tipo pulsera para la detección de caídas, basado en el empleo de redes neuronales. Este enfoque innovador busca aprovechar los avances en inteligencia artificial y tecnología portátil para proporcionar una solución precisa y confiable para la detección temprana de caídas.

Para comprender plenamente el contexto en el que se enmarca este proyecto, es fundamental realizar un análisis exhaustivo del estado actual de las tecnologías y métodos utilizados en la detección de caídas. Este análisis, conocido como Estado del Arte, permite identificar las principales tendencias, avances y desafíos en este campo, proporcionando una base sólida para el desarrollo de nuevas soluciones.

A lo largo de este documento, se explorarán los dispositivos y tecnologías más relevantes para la detección de caídas, así como las técnicas y algoritmos utilizados para procesar los datos y reconocer patrones asociados con este tipo de eventos. Además, se realizará una comparativa entre diferentes enfoques, destacando sus fortalezas y limitaciones, con el objetivo de fundamentar la elección de la metodología propuesta en este trabajo.

En última instancia, se espera que este estudio contribuya al avance del conocimiento en el campo de la detección de caídas y sirva como punto de partida para el desarrollo de dispositivos y sistemas más efectivos y accesibles para la población en riesgo.

# Dispositivos y Tecnologías para la Detección de Caídas

Las caídas son un grave problema de salud pública a nivel global, siendo la segunda causa principal de muerte por lesiones no intencionales después de los accidentes de tráfico. Más del 80% de las muertes por caídas ocurren en países de ingresos bajos y medianos, siendo las regiones del Pacífico Occidental y del Sudeste Asiático las más afectadas. Cada año, aproximadamente 37.3 millones de caídas requieren atención médica, resultando en más de 38 millones de años de vida ajustados por discapacidad perdidos. Aunque el 40% de estos años perdidos ocurren en niños, las personas mayores enfrentan un mayor riesgo de discapacidad y necesidad de cuidados a largo plazo después de una caída [1].

La Figura 1 muestra la evolución del número de estudios científicos relacionados con la detección de caídas a lo largo de los años, basada en datos recopilados de una búsqueda en PubMed utilizando el término "fall detection" (detección de caídas) (<https://pubmed.ncbi.nlm.nih.gov/?term=fall+detection>). Este diagrama proporciona una visión general del crecimiento y desarrollo de la investigación en este campo crucial de la salud y el bienestar.

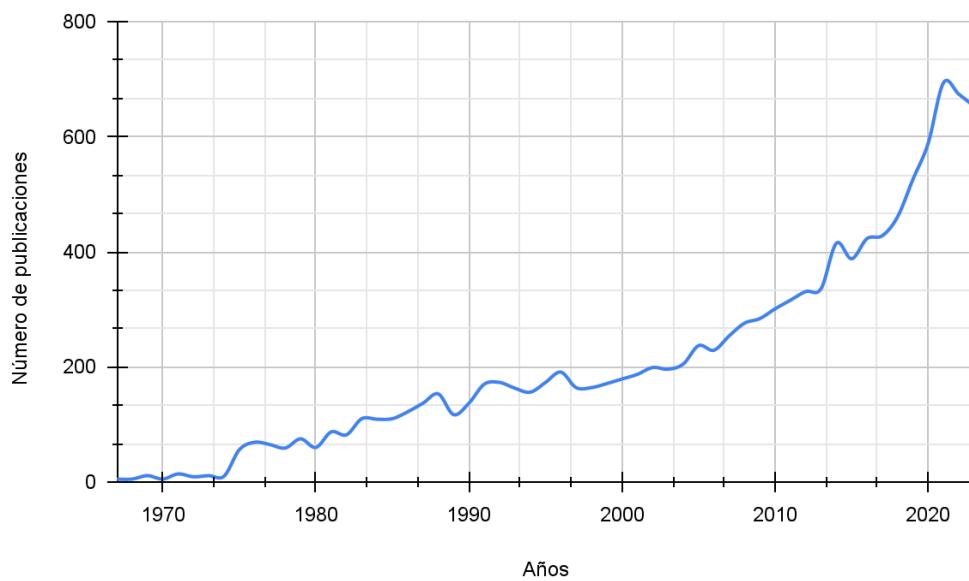


Figura 1: Evolución del Número de Estudios Científicos sobre la Detección de Caídas a lo largo de los Años

La detección de caídas es un área de investigación en constante evolución. Este diagrama destaca el aumento significativo en el interés y la actividad investigadora en este campo, reflejando el creciente reconocimiento de la importancia de abordar este problema de salud pública.

La detección de caídas se puede dividir en tres categorías principales: dispositivos portátiles, dispositivos basados en cámaras y dispositivos de ambiente, como se muestra en la Figura 2 [2].

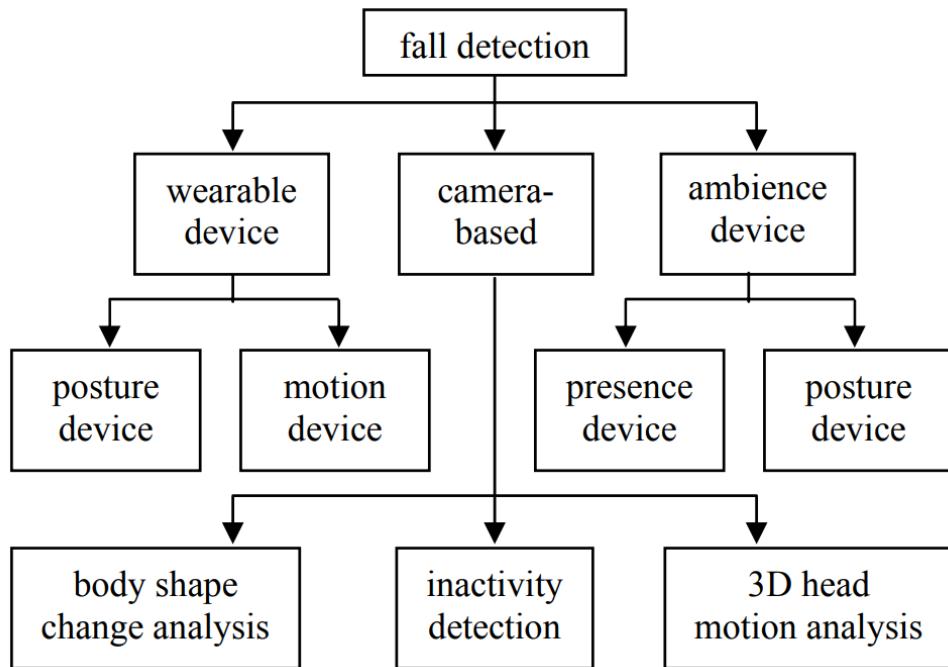


Figura 2: Clasificación de dispositivos de detección de caídas

## **Dispositivos Portátiles**

Los dispositivos portátiles, como pulseras, relojes inteligentes y sensores adheridos al cuerpo, se han convertido en una opción popular para la detección de caídas debido a su comodidad, discreción y facilidad de uso. Cuando ocurre una caída, un sensor de aceleración incorporado en estos dispositivos detecta la velocidad a la que una persona se dirige hacia el suelo. Un algoritmo determina si la persona ha sufrido una caída. Si es así, el dispositivo envía una señal al equipo de vigilancia del fabricante. Si la persona confirma que se ha caído, el agente envía una notificación al contacto de emergencia de la persona que figura en el sistema. Si la persona que

potencialmente se ha caído no responde, se notifica automáticamente a su contacto de emergencia. Muchos dispositivos con sensores para llevar puestos también cuentan con un botón manual para que la persona pueda notificar por sí misma una caída cuando esté en condiciones de hacerlo [3]. Podemos observar un ejemplo en la Figura 3, que se encuentra a continuación.

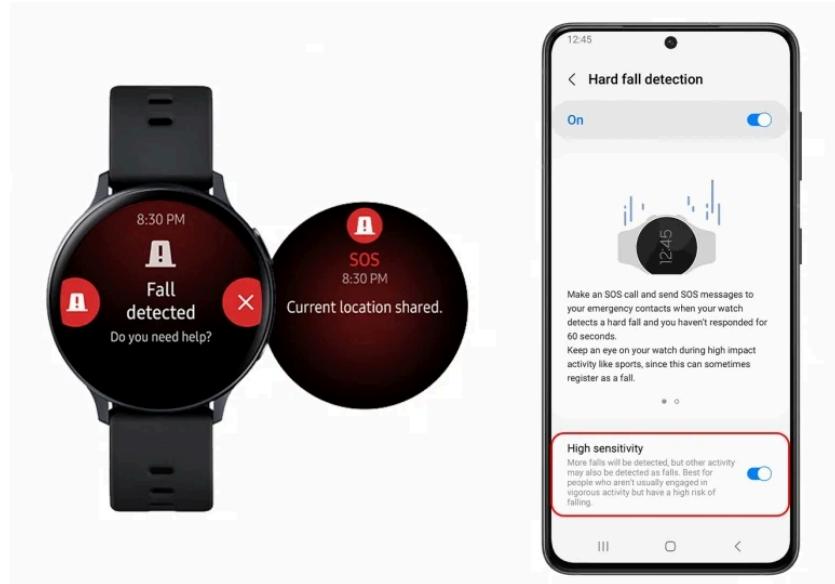


Figura 3: Ejemplo de dispositivo portátil para la detección de caídas

## ***Dispositivos de Ambiente***

Los sistemas de detección de caídas basados en sensores ambientales se centran en la captura de señales del entorno circundante para identificar eventos de caídas. Estos sistemas utilizan una variedad de sensores, como micrófonos, radar Doppler, sensores de vibración del suelo, sensores de presión y cámaras, para monitorear activamente el espacio donde ocurren las caídas.

Los micrófonos se utilizan para registrar sonidos en el entorno, lo que permite detectar eventos de caídas mediante el análisis de las características acústicas. El radar Doppler puede detectar cambios en el movimiento dentro de un área determinada, mientras que los sensores de vibración del suelo capturan vibraciones generadas por impactos, como el de una caída. Los sensores de presión pueden identificar cambios en la distribución del peso corporal, lo que puede indicar una caída,

y las cámaras pueden proporcionar imágenes visuales para confirmar la ocurrencia de una caída.

Estos sistemas tienen la ventaja de ser menos intrusivos que los sistemas basados en sensores portátiles, ya que no requieren que el usuario lleve consigo un dispositivo. Sin embargo, pueden tener limitaciones en términos de cobertura de área y costes de instalación [4].

En la Figura 4, se presenta un gráfico que muestra cómo se registra y visualiza una caída a través de las vibraciones detectadas por estos dispositivos.

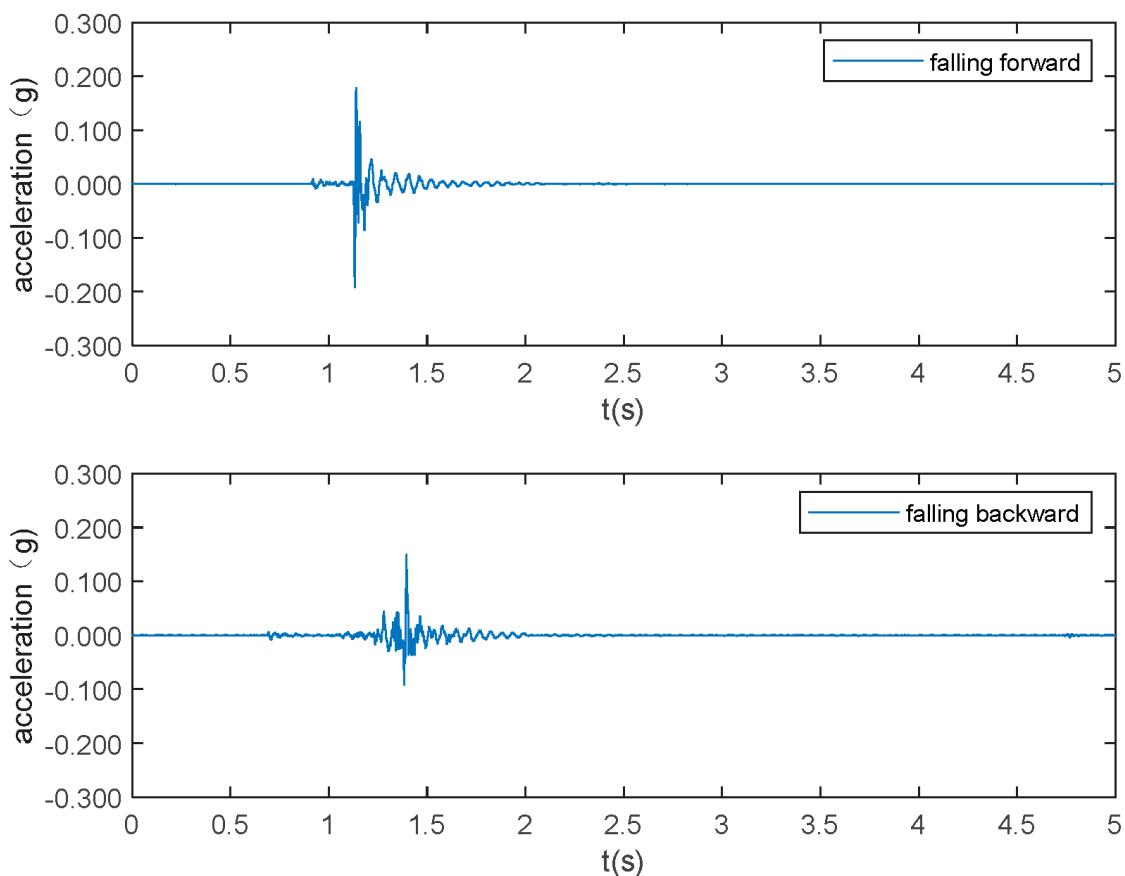


Figura 4: Vibración registrada con la caída de un maniquí de 75 kg

## **Dispositivos de Vídeo**

Los dispositivos basados en cámaras o análisis de vídeo constituyen una categoría importante en la detección de caídas, ofreciendo una alternativa menos intrusiva que los dispositivos portátiles. Estos sistemas utilizan cámaras para monitorear continuamente el entorno de una habitación y detectar la presencia de personas, así como identificar caídas potenciales.

Una de las ventajas de estos sistemas es su capacidad para ejecutar algoritmos en tiempo real utilizando plataformas de computación estándar y cámaras de bajo coste. Por ejemplo, en el estudio presentado [5], se propone un sistema de detección de caídas basado en una cámara de bajo coste integrada en un dispositivo Raspberry Pi. Este sistema puede ser instalado en paredes o techos para monitorear una habitación sin necesidad de intervención humana. Además, no requiere que las personas monitoreadas usen dispositivos adicionales, lo que permite un monitoreo continuo las 24 horas del día.

El sistema se basa en algoritmos de visión artificial que analizan la presencia de personas en una habitación y detectan si una persona ha sufrido una caída. Cuando se detecta una caída, se envía un mensaje de alerta al cuidador junto con una imagen capturada. Este enfoque ofrece una forma efectiva de detectar caídas sin comprometer la privacidad, ya que las imágenes solo se envían en caso de una detección de caída y pueden ser fácilmente difuminadas para proteger la identidad de las personas monitoreadas.

Podemos observar un ejemplo concreto en la Figura 5. En esta imagen, se muestra una representación visual de una caída capturada desde el punto de vista de un dispositivo de detección de caídas basado en cámara.



Figura 5: Caída captura por un dispositivo de detección de caídas basado en vídeo

## Técnicas y Algoritmos para la Detección de Caídas

La detección de caídas es un proceso crucial para garantizar la seguridad y el bienestar de las personas en riesgo, especialmente para aquellos que son vulnerables a accidentes relacionados con la movilidad. Para abordar este desafío, se han desarrollado una variedad de técnicas y algoritmos, cada uno con sus propias características y aplicaciones específicas. Algunas de las técnicas más comunes utilizadas para la detección de caídas son las que veremos a continuación:

### ***Algoritmos Basados en Umbrales***

Los algoritmos basados en umbrales son una de las técnicas más simples y comúnmente utilizadas para la detección de caídas. Estos algoritmos funcionan estableciendo umbrales predefinidos para ciertas variables, como la aceleración o la orientación angular, y activando una alerta cuando estos umbrales son superados.

Una variedad de estudios han explorado el uso de algoritmos de detección de caídas basados en umbrales, cada uno con resultados prometedores. Ren (2015) [6] propuso un enfoque de extracción de umbrales personalizado, logrando una alta precisión del 96.76% para la detección de caídas. Fudickar (2014) [7] evaluó algoritmos basados en teléfonos inteligentes, demostrando su potencial para ayudar a

las personas mayores. Dumitrache (2013) [8] diseñó un algoritmo de detección de caídas utilizando un acelerómetro triaxial, logrando una sensibilidad del 97.05% y una especificidad del 99%. Shi (2016) [9] mejoró aún más estos resultados al incorporar un magnetómetro y observar un aumento en la precisión de detección. Estos estudios destacan colectivamente el potencial de los algoritmos basados en umbrales en la detección de caídas, especialmente en el contexto del cuidado de personas mayores.

## **Redes Neuronales**

Las redes neuronales son modelos computacionales inspirados en la estructura y el funcionamiento del cerebro humano. Están compuestas por unidades interconectadas llamadas neuronas, que procesan y transmiten información a través de conexiones ponderadas. Estas conexiones, que tienen pesos asociados, permiten a la red neuronal aprender patrones complejos a partir de datos de entrada y generar salidas útiles.

En la detección de caídas, las redes neuronales se utilizan para analizar datos sensoriales, como la aceleración, la orientación angular o incluso datos de video, con el fin de identificar patrones que indiquen una posible caída. Estos modelos pueden aprender de manera automática y adaptarse a diferentes situaciones, lo que los hace adecuados para abordar la variabilidad y complejidad de los movimientos humanos asociados con las caídas.

Una serie de estudios han explorado el uso de algoritmos de detección de caídas basados en redes neuronales. Ye (2019) [10] desarrolló una red BiLSTM para mejorar la precisión de la detección de caídas, logrando una tasa de precisión del 90.47%. De manera similar, Zhang (2013) [11] utilizó un algoritmo de red neuronal para distinguir con precisión las caídas de las actividades diarias, con alta eficiencia y fiabilidad. Huang (2007) [12] propuso un algoritmo de detección de caídas basado en video utilizando redes neuronales modulares, logrando eficacia y fiabilidad. Salleh (2020) [13] mejoró aún más la precisión de la detección de caídas mediante el uso de un algoritmo de red neuronal de autorregresión no lineal, con el mejor modelo logrando una tasa de precisión del 0.92742. Estos estudios demuestran colectivamente el potencial de los algoritmos basados en redes neuronales en la detección de caídas.

La Figura 6 presenta una comparativa entre el número de estudios de detección de caídas mediante algoritmos basados en umbrales y redes neuronales. Los datos utilizados para esta comparación han sido obtenidos de <https://pubmed.ncbi.nlm.nih.gov/>

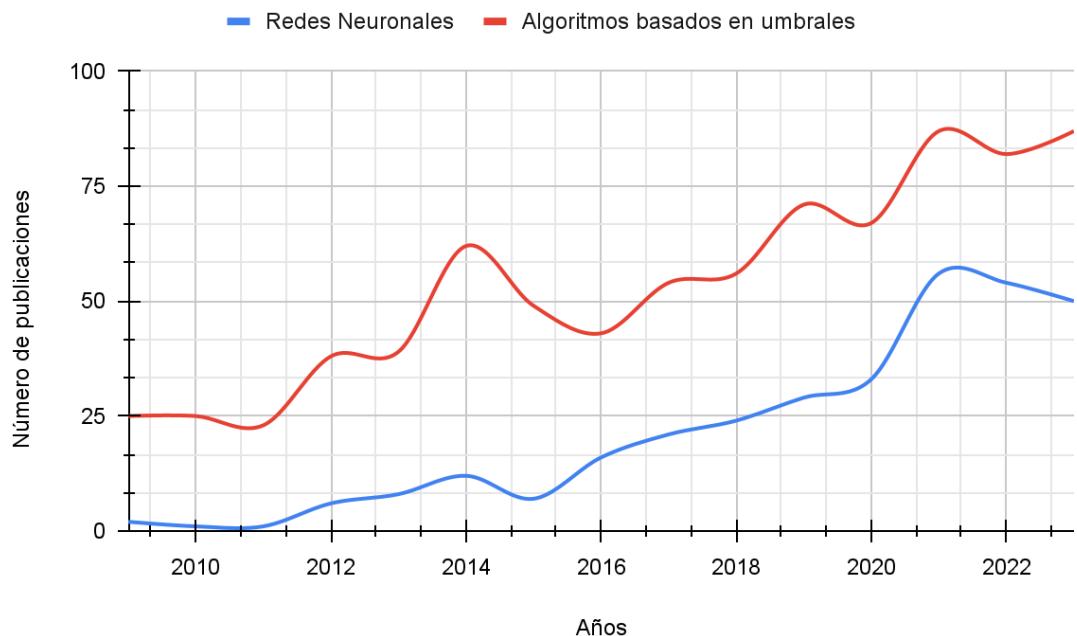


Figura 6: Gráfica comparativa entre el número de publicaciones de detección de caídas mediante algoritmos basados en umbrales y redes neuronales.

## Comparativa y Evaluación de Tecnologías y Métodos

La detección de caídas es un campo en constante evolución, y la elección de la tecnología o método adecuado depende en gran medida de las necesidades específicas del usuario, el entorno de uso y otros factores contextuales. A continuación, realizaremos una comparativa detallada entre las diferentes tecnologías y métodos mencionados, resaltando sus ventajas y desventajas, así como ejemplos de casos de uso y situaciones en las que cada uno puede ser más apropiado:

## ***Dispositivos Portátiles***

Basándonos en el trabajo realizado por SreeMadhubala (2015) [\[14\]](#) afirma que los sistemas basados en dispositivos portátiles son rentables y resistentes al ruido. Se adaptan tanto a entornos interiores como exteriores y presentan numerosas ventajas en comparación con los sistemas basados en sensores integrados en el entorno. Este sistema prescinde de la necesidad de infraestructura, eliminando así la configuración de una red de comunicación requerida por otros métodos. Sin embargo, sus inconvenientes incluyen una vida útil limitada de la batería y un monitoreo intrusivo de las personas. Además, el peso del dispositivo debe ser considerado, ya que puede resultar incómodo para el usuario.

## ***Dispositivos Integrados en el Entorno***

Estos sistemas, que emplean micrófonos y sensores infrarrojos, presentan similitudes con las técnicas de visión artificial. Su estructura es económica y sencilla, conformada por un sensor acústico o ambiental y un ordenador personal para el análisis de datos. En investigaciones relacionadas, se han propuesto varios sistemas, como SIMBAD [\[15\]](#), un monitor automático de inactividad que emplea detectores basados en matrices de infrarrojos, y un sistema de detección de caídas basado en vibraciones del suelo. También se han diseñado sistemas de detección de caídas utilizando sensores acústicos y radar Doppler [\[16\]](#). A pesar de sus ventajas en términos de privacidad y simplicidad, estos sistemas están restringidos a aplicaciones en interiores y requieren aún una extensa investigación en el área [\[14\]](#).

## ***Análisis de Video***

Los sistemas de visión por computadora son capaces de extraer información útil de imágenes sin procesar para su análisis y procesamiento. Este enfoque, que incluye la medición de características, la clasificación de patrones y el reconocimiento de patrones, se utiliza en diversos sistemas de detección de caídas. Varios estudios han propuesto soluciones que emplean cámaras para detectar caídas a través del análisis de movimiento y postura, logrando tasas de precisión que alcanzan hasta el 97.08% [\[17\]](#). Sin embargo, estos sistemas enfrentan desafíos como puntos ciegos,

problemas de ocultamiento y costes asociados con la instalación de múltiples cámaras. Aunque ofrecen ventajas sobre los sistemas basados en sensores vestibles, como la eliminación de la necesidad de recargar baterías y la comodidad de uso, requieren una inversión significativa en infraestructura y mantenimiento [14].

Después de analizar detalladamente las distintas tecnologías y métodos para la detección de caídas, es importante realizar una comparación entre sus pros y contras para entender mejor sus características y aplicaciones. En la Figura 7, podemos ver una imagen que resume esta comparativa, destacando los aspectos positivos y negativos de cada enfoque.

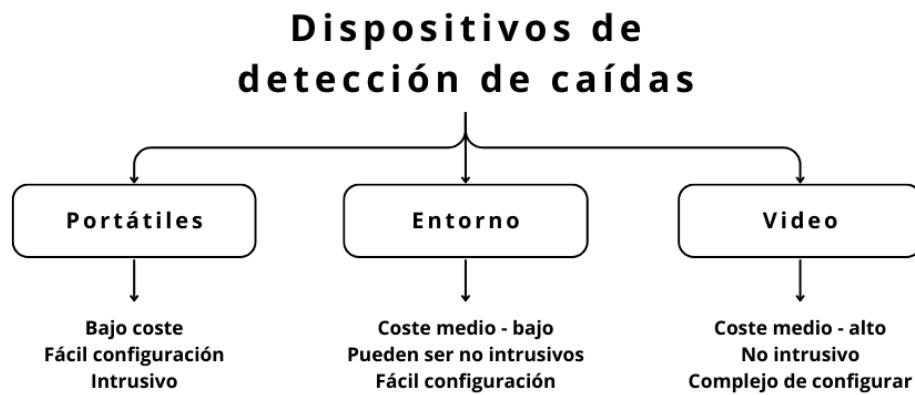


Figura 7: Comparación entre los diferentes tipos de dispositivos

## Conclusiones del Estado del Arte

El análisis del estado del arte en el campo de la detección de caídas revela una variedad de tecnologías y métodos utilizados para abordar este desafío crucial en el cuidado de la salud y la seguridad personal. La investigación revisada abarca desde dispositivos portátiles hasta sistemas integrados en el entorno, así como técnicas avanzadas de análisis de datos y aprendizaje automático. A continuación, se resumen las principales tendencias y hallazgos encontrados en este análisis, así como su relación con el proyecto de TFG centrado en el desarrollo de una pulsera portátil para la detección de caídas mediante redes neuronales entrenadas.

## ***Diversidad de Enfoques Tecnológicos***

Se observa una diversidad de enfoques tecnológicos utilizados para la detección de caídas, cada uno con sus propias ventajas y limitaciones. Tras revisar detalladamente los distintos métodos y tecnologías disponibles, se ha decidido centrar el proyecto de TFG en el desarrollo de una pulsera portátil para la detección de caídas.

## ***Relevancia para el Proyecto de TFG***

La elección de desarrollar una pulsera portátil se fundamenta en varias consideraciones. En primer lugar, los dispositivos portátiles ofrecen una solución conveniente y no intrusiva para la detección de caídas, permitiendo a los usuarios llevar el dispositivo consigo en todo momento sin interferir significativamente en sus actividades diarias. Además, la pulsera portátil se alinea con la tendencia actual hacia la autonomía y el empoderamiento de las personas mayores y otros grupos vulnerables, ya que les brinda una mayor independencia y seguridad en su vida cotidiana. Por último, la decisión de utilizar una red neuronal entrenada se basa en la eficacia demostrada de este enfoque en la identificación de patrones de caída a partir de datos de sensores [\[10\]](#), así como en el creciente auge de las inteligencias artificiales y las redes neuronales. Esta combinación promete una detección precisa y fiable de las caídas en tiempo real.

El estado del arte en la detección de caídas proporciona una base sólida para el desarrollo del proyecto de TFG, destacando la relevancia y la viabilidad de una pulsera portátil basada en redes neuronales entrenadas como una solución efectiva y prometedora para la detección de caídas en la población objetivo.

# **Planificación**

Se quiere desarrollar un dispositivo wearable para la detección de caídas, el cual se basa en redes neuronales. Se han definido diversas etapas y tareas para su ejecución, las cuales se detallan a continuación, junto con una estimación del tiempo dedicado a cada una de ellas, representada mediante un diagrama de Gantt.

## **Fase 1 - Investigación y Planificación**

Realizar investigaciones sobre tecnologías relevantes para el desarrollo del dispositivo y planificar las acciones a seguir.

### ***Tarea 1.1 - Definición del Alcance***

Identificar y definir objetivos específicos y metas a alcanzar durante el proyecto.

Duración: 1 semana.

### ***Tarea 1.2 - Investigación de Tecnologías***

Investigar tecnologías relevantes para redes neuronales y dispositivos wearables.

Duración: 2 semanas.

### ***Tarea 1.3 - Estudio del Estado del Arte***

Realizar un estudio exhaustivo del estado del arte en el campo de detección de caídas y tecnologías asociadas.

Duración: 3 semanas.

## **Fase 2 - Desarrollo de Redes Neuronales**

Realizar pruebas de concepto de diferentes arquitecturas de redes neuronales para la detección de caídas, entrenamiento de las redes seleccionadas y su implementación en el dispositivo.

### ***Tarea 2.1 - Pruebas de Concepto***

Realizar pruebas de concepto de diferentes arquitecturas de redes neuronales.

Duración: 3 semanas.

### ***Tarea 2.2 - Entrenamiento de Redes Neuronales***

Entrenar las redes neuronales seleccionadas.

Duración: 4 semanas.

### ***Tarea 2.3 - Evaluación y Ajustes***

Evaluar el rendimiento de las redes entrenadas y realizar ajustes necesarios.

Duración: 3 semanas.

### ***Hito 1 - Desarrollo y Optimización de la Red Neuronal***

Al finalizar la Fase 2, se alcanzará el Hito 1, correspondiente al Objetivo 1 del proyecto.

## **Fase 3 - Desarrollo del Dispositivo Wearable**

Seleccionar el hardware del dispositivo, desarrollar el software para la recopilación de datos del sensor, integrar la red neuronal en el dispositivo y realizar pruebas de funcionamiento.

### ***Tarea 3.1 - Selección del Hardware***

Seleccionar el hardware del dispositivo wearable.

Duración: 2 semanas.

### ***Tarea 3.2 - Desarrollo del Software***

Desarrollar el software para la recopilación de datos del sensor.

Duración: 3 semanas.

### ***Tarea 3.3 - Integración de la Red Neuronal***

Integrar la red neuronal en el dispositivo.

Duración: 2 semanas.

## ***Hito 2 - Prototipo Hardware y Realización de Pruebas***

Una vez finalizada la Fase 3, se completará el Hito 2, que corresponde al Objetivo 3 del proyecto.

## **Fase 4 - Pruebas y Validación**

Realizar pruebas de las redes neuronales y del dispositivo en diferentes situaciones de caída, así como validar la solución en entornos controlados y del mundo real.

### **Tarea 4.1 - Pruebas de Redes Neuronales**

Realizar pruebas de las redes neuronales implementadas.

Duración: 2 semanas.

### **Tarea 4.2 - Pruebas del Dispositivo**

Realizar pruebas de precisión y fiabilidad del dispositivo en la detección de caídas.

Duración: 4 semanas.

### **Tarea 4.3 - Validación en Entornos Reales**

Validar la solución en entornos del mundo real.

Duración: 3 semanas.

### **Hito 3 - Pruebas de Comunicación y Alerta**

Al concluir la Fase 4, se logrará el Hito 3, asociado al Objetivo 2 del proyecto.

## **Fase 5 - Documentación y Entrega**

Documentar detalladamente cada etapa del proyecto, crear manuales de usuario y materiales de capacitación, y finalmente entregar el proyecto completo.

### ***Tarea 5.1 - Documentación Técnica***

Documentar detalladamente cada etapa del proyecto.

Duración: 2 semanas.

### ***Tarea 5.2 - Manuales de Usuario***

Crear manuales de usuario para el dispositivo y el sistema de detección de caídas.

Duración: 2 semanas.

### ***Tarea 5.3 - Entrega Final del Proyecto***

Entregar el proyecto completo con todos los componentes y documentación asociada.

Duración: 1 semana.

A continuación en la Figura 8, se presenta el diagrama de Gantt correspondiente, que muestra la distribución del tiempo para cada una de las etapas y tareas mencionadas anteriormente.

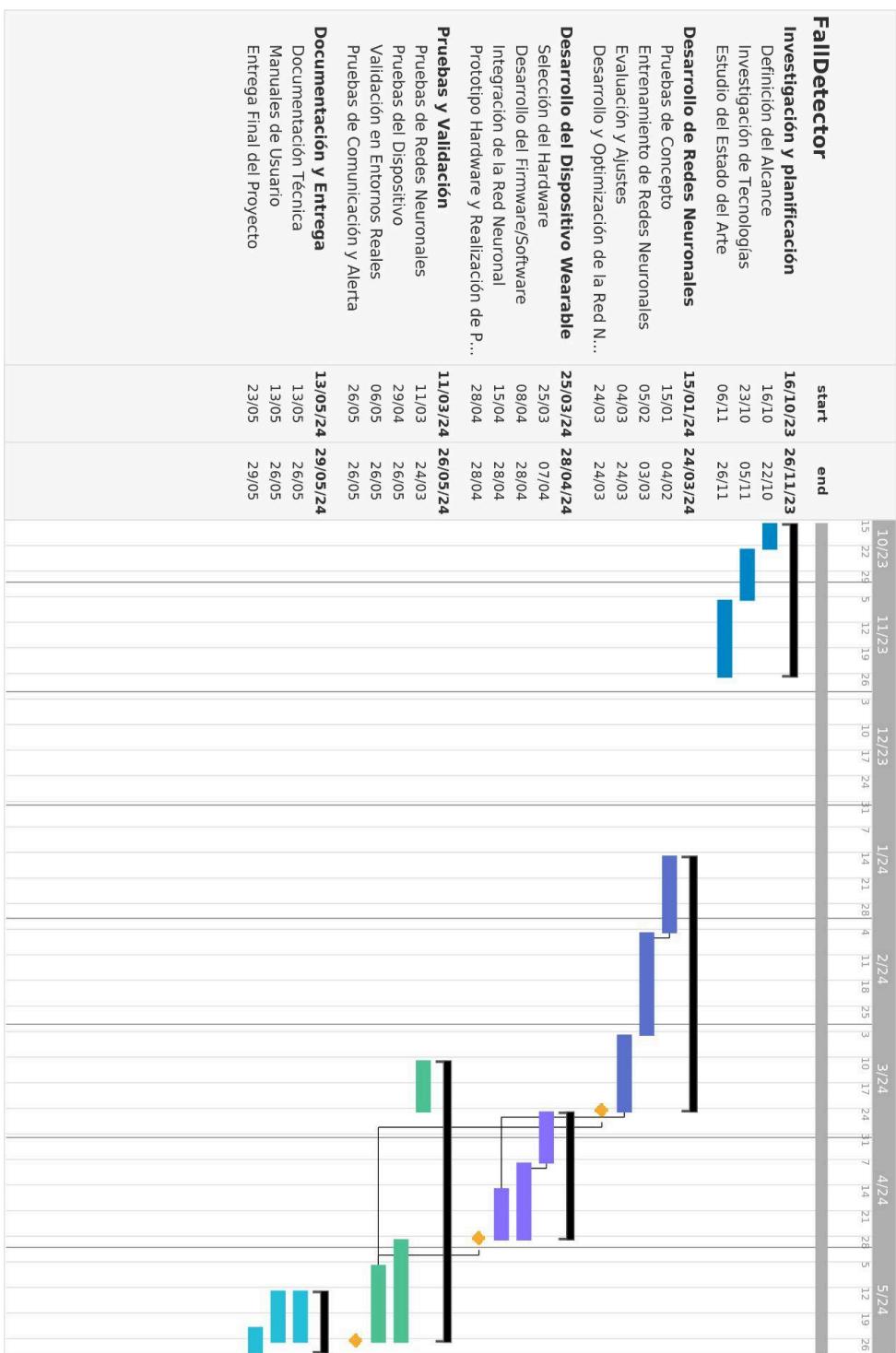


Figura 8: Diagrama de Gantt ideal

# **Metodología de Trabajo**

En esta sección, se proporciona una descripción detallada de la implementación del trabajo llevado a cabo, así como de las metodologías y procedimientos aplicados durante su ejecución. Se presenta un análisis de las herramientas utilizadas, los enfoques metodológicos adoptados y los pasos seguidos en cada fase del desarrollo.

## **Pruebas de Concepto**

En la tarea 2.1, se llevaron a cabo pruebas de concepto con el objetivo de evaluar la viabilidad de utilizar redes neuronales para la detección de caídas. Estas pruebas se realizaron siguiendo una metodología específica y utilizando un conjunto de herramientas determinadas.

## **Herramientas Utilizadas**

**Entorno de Desarrollo:** Se utilizó Visual Studio Code (VSCode) como entorno de desarrollo integrado (IDE) para escribir y ejecutar el código.

**Lenguaje de Programación:** Se empleó Python como lenguaje de programación principal para la implementación del modelo de red neuronal.

**Framework de Machine Learning:** Se hizo uso de TensorFlow, una de las bibliotecas más populares y potentes para el desarrollo de modelos de machine learning y deep learning en Python.

## ***Metodología***

### **Selección de Herramientas**

Se seleccionaron las herramientas adecuadas para el desarrollo y entrenamiento de la red neuronal. Se optó por utilizar Visual Studio Code como entorno de desarrollo, Python como lenguaje de programación principal y TensorFlow como framework de machine learning debido a su popularidad y eficacia en este tipo de tareas.

### **Desarrollo del Modelo de Red Neuronal**

Se procedió a diseñar y desarrollar un modelo básico de red neuronal utilizando TensorFlow en Python. Se definió la arquitectura de la red, incluyendo el número de capas, neuronas y funciones de activación, de acuerdo con los requisitos del problema, una arquitectura básica con 3 capas, una para la entrada, otra intermedia y finalmente una de salida.

## ***Procedimientos***

### **Preprocesamiento de Datos**

Antes de iniciar el entrenamiento del modelo, se realizó un preprocesamiento de los datos, incluyendo la unificación de los distintos archivos de mediciones y la selección de los datos relevantes para el entrenamiento del modelo, como las señales de aceleración y giroscopio.

### **Entrenamiento del Modelo**

Se llevó a cabo el entrenamiento del modelo utilizando el conjunto de datos preparado previamente. Durante este proceso, el modelo ajustó sus parámetros internos para optimizar su capacidad de detección de caídas. Se realizaron múltiples iteraciones de entrenamiento, evaluando continuamente el rendimiento del modelo.

## **Resultados y Conclusiones**

Al completar las pruebas de concepto, se lograron obtener resultados prometedores. El modelo de red neuronal demostró una capacidad satisfactoria para detectar caídas en el conjunto de datos utilizado, alcanzando un nivel de exactitud (accuracy) que cumplía con los objetivos establecidos inicialmente.

Basándonos en los resultados obtenidos y en la metodología aplicada en estas pruebas de concepto, se tomó la decisión de continuar con el desarrollo y entrenamiento de una red neuronal más avanzada para la detección de caídas, como parte del siguiente paso en la implementación del proyecto.

## **Entrenamiento de Redes Neuronales**

En esta fase del proyecto, se procedió a desarrollar y entrenar una red neuronal bilateral LSTM para la detección de caídas. Sin embargo, debido a la complejidad y extensión del nuevo conjunto de datos, se requirió un enfoque más riguroso y potente para el entrenamiento del modelo.

## **Herramientas Utilizadas**

### **Entorno Virtual con Anaconda**

Ante las limitaciones encontradas al intentar entrenar el modelo con la GPU GTX 1060, se tomó la decisión de crear un entorno virtual utilizando Anaconda. Esto nos permitió configurar un ambiente independiente con Python 3.10.13 y TensorFlow 2.10.0, asegurando la compatibilidad y evitando conflictos entre versiones de librerías. A continuación, se presenta una tabla que detalla el software y las versiones utilizadas:

| <b>Software</b>    | <b>Versión</b> | <b>Descripción</b>   |
|--------------------|----------------|--|
| Visual Studio Code | 1.87.2         | Entorno de desarrollo integrado (IDE) utilizado para escribir y ejecutar el código de Python.              |
| Python             | 3.10.13        | Lenguaje de programación principal utilizado para implementar el modelo de red neuronal.                   |
| Anaconda           | 2.5.2          | Plataforma de distribución de Python que incluye numerosas herramientas y paquetes preinstalados.          |
| TensorFlow         | 2.10.0         | Framework de machine learning utilizado para desarrollar y entrenar modelos de redes neuronales en Python. |
| TensorBoard        | 2.10.1         | Herramienta utilizada para el análisis y la visualización de las métricas de rendimiento de TensorFlow.    |
| NumPy              | 1.26.4         | Biblioteca utilizada para operaciones numéricas y computación científica en Python.                        |
| Pandas             | 2.2.1          | Biblioteca utilizada para manipulación y análisis de datos en Python.                                      |
| scikit-learn       | 1.4.1.post1    | Biblioteca utilizada para machine learning y minería de datos en Python.                                   |

Tabla 1: Componentes Software y sus versiones

Se ha elaborado un manual detallado que describe el proceso de configuración de Anaconda y su entorno virtual para la creación de un ambiente independiente compatible con Python 3.10.13 y TensorFlow 2.10.0. Este manual proporciona instrucciones paso a paso, acompañadas de capturas de pantalla, para guiar al usuario a través del proceso de instalación y configuración.

El manual se encuentra disponible en el [Anexo 1](#). Se recomienda su consulta para aquellos lectores interesados en reproducir o comprender en detalle la configuración del entorno utilizado en el desarrollo y entrenamiento de la red neuronal para la detección de caídas.

## ***Metodología***

### **Selección del Conjunto de Datos**

Se optó por utilizar un conjunto de datos más robusto y extenso, conscientes de que una mayor complejidad en los datos podría mejorar la capacidad predictiva del modelo.

### **Configuración del Entorno de Entrenamiento**

La creación de un entorno virtual con Anaconda facilitó la gestión de las dependencias y la resolución de problemas de compatibilidad, permitiendo un enfoque más eficiente en el entrenamiento del modelo.

### **Optimización de Hiperparámetros**

Se llevó a cabo una exhaustiva exploración de hiperparámetros, realizando múltiples ciclos de entrenamiento para determinar la combinación óptima que maximizara el rendimiento del modelo en la detección de caídas.

### **Evaluación de Métricas**

Reconociendo la importancia de métricas adecuadas para la detección de caídas, se dedicó tiempo a investigar y seleccionar las métricas más apropiadas para evaluar el desempeño del modelo en un contexto desbalanceado de datos.

## ***Procedimientos***

### **Preprocesamiento de Datos**

El primer paso crucial fue el preprocesamiento de los datos recopilados para entrenar la red neuronal. Para este fin, se utilizó un conjunto de datos específico diseñado por Olukunle Ojetola, Elena Gaura y James Brusey [18], que incluye eventos de caídas y actividades diarias registrados mediante sensores iniciales. Sin embargo, debido a limitaciones de potencia de procesamiento, no se empleó el dataset completo

en el entrenamiento. En su lugar, se optó por utilizar el dataset "Addendum\_November\_2016", de los mismos autores, disponible en el siguiente enlace:

[https://skuld.cs.umass.edu/traces/mmsys/2015/paper-15/Addendum\\_November\\_2016.zip.](https://skuld.cs.umass.edu/traces/mmsys/2015/paper-15/Addendum_November_2016.zip)

El conjunto de datos se compone de registros de 42 voluntarios que participaron en un experimento que incluyó una serie de protocolos guionados. Estos protocolos simularon diferentes tipos de caídas (hacia adelante, hacia atrás, lateral izquierda y lateral derecha) junto con varias actividades de la vida diaria. Los datos están estructurados y anotados, lo que facilita su uso para el desarrollo y evaluación de algoritmos de detección de caídas.

El conjunto de datos "Addendum\_November\_2016" es una versión del dataset original que proporciona una selección representativa de eventos de caídas y actividades diarias, lo que lo hace adecuado para el entrenamiento de la red neuronal en nuestro contexto.

El dataset completo, junto con los vídeos disponibles, está disponible para su acceso público en el siguiente enlace:

[http://cogentee.coventry.ac.uk/datasets/fall\\_adl\\_data.zip.](http://cogentee.coventry.ac.uk/datasets/fall_adl_data.zip)

Este paso de preprocesamiento finalizó con la consolidación de los distintos archivos de mediciones en un único conjunto de datos consolidado, que contenía más de 2 millones de filas. Este proceso de unificación nos permitió tener un dataset completo y coherente, listo para ser utilizado en el entrenamiento de nuestra red neuronal para la detección de caídas.

## **Problemas con la GPU y Solución**

El primer obstáculo surgió al intentar utilizar la GPU GTX 1060 para el entrenamiento del modelo. Se descubrió que las librerías de TensorFlow y Python instaladas previamente no eran compatibles con la GPU. Para superar este problema, se optó por crear un entorno virtual utilizando Anaconda con las versiones específicas de Python y TensorFlow requeridas.

## Estructura de la Red Neuronal

La estructura de la red neuronal utilizada desempeña un papel fundamental en su capacidad para aprender y generalizar a partir de los datos. En el contexto de la detección de caídas, se seleccionó una arquitectura de red neuronal bilateral LSTM (Long Short-Term Memory) debido a sus propiedades para capturar dependencias a largo plazo en secuencias de datos, como las series temporales. Esta arquitectura consiste en varias capas LSTM bidireccionales seguidas de capas densas para la clasificación final.

La elección de esta arquitectura se basó en la naturaleza secuencial de los datos de entrada y la necesidad de capturar relaciones temporales complejas. Las capas LSTM bidireccionales son capaces de procesar la información en ambas direcciones a lo largo de la secuencia, permitiendo una comprensión más completa de los patrones temporales en los datos. Además, las capas LSTM están diseñadas con puertas de memoria que les permiten aprender y recordar patrones a largo plazo, lo que es crucial para la detección precisa de eventos como las caídas.

El modelo especificado en la Figura 9 incluye múltiples capas bidireccionales LSTM con diferentes tamaños de unidad para capturar patrones a diferentes escalas temporales. Cada capa LSTM está seguida por una capa de dropout para regularizar el modelo y evitar el sobreajuste. La capa de salida consiste en una capa densa con una función de activación sigmoide, que produce la probabilidad de que ocurra una caída.

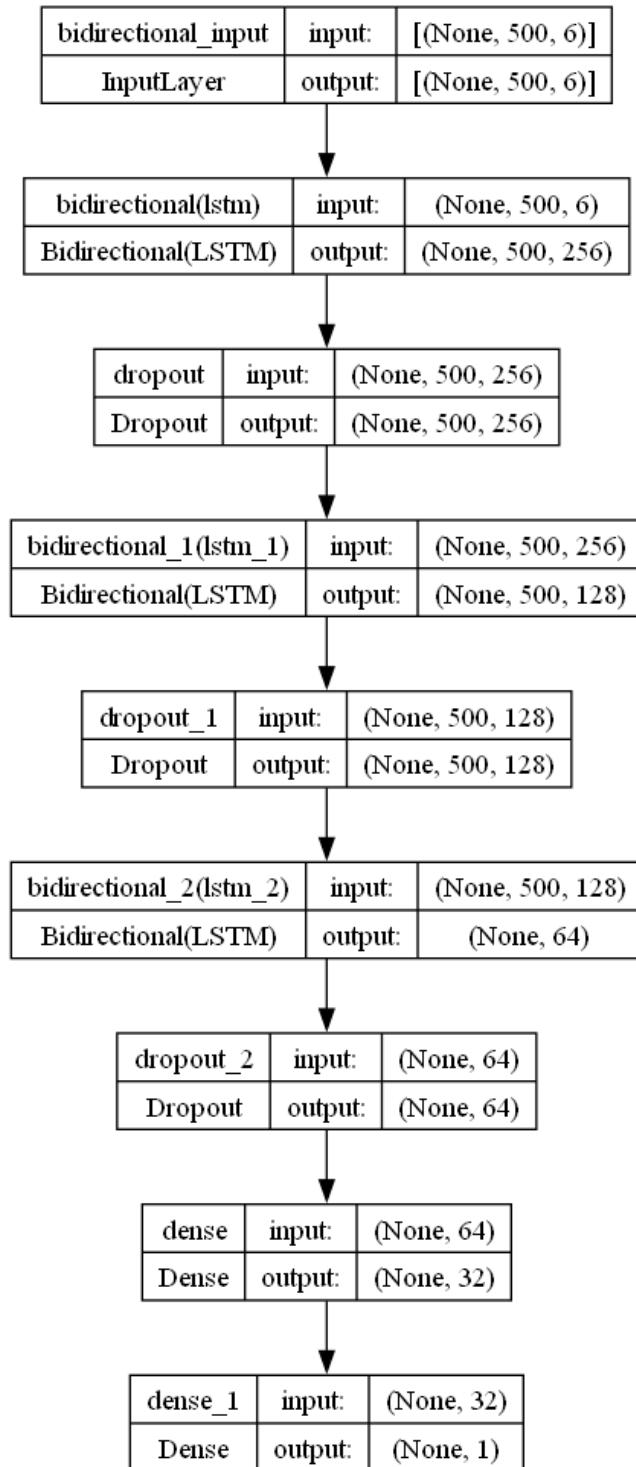


Figura 9: Estructura Red Neuronal Bilateral - LSTM

Este modelo está diseñado para procesar secuencias de datos de entrada con una forma de (500, 6), lo que significa que cada secuencia tiene una longitud de 500 pasos de tiempo y 6 características. Las capas LSTM están intercaladas con capas de dropout para mitigar el sobreajuste durante el entrenamiento. La última capa densa produce una única salida que representa la probabilidad de que ocurra una caída.

## **Procesamiento de Datos en la Red Neuronal**

Una vez preprocesados, los datos fueron alimentados a la red neuronal para su entrenamiento. Durante esta etapa, la red neuronal recibió secuencias de datos de entrada, representadas como series temporales de características. Estas secuencias, procesadas por las capas LSTM, permitieron que la red neuronal aprendiera representaciones de alto nivel de los datos a lo largo del tiempo.

Para preparar los datos para el entrenamiento, se dividió el conjunto de datos en ventanas temporales de tamaño fijo. Cada ventana contenía un conjunto de puntos de datos consecutivos, y se superponían para garantizar una cobertura adecuada de los datos. Además, se asignó una etiqueta a cada ventana, correspondiente a la clase del último punto de datos en esa ventana.

El proceso de creación de ventanas temporales se llevó a cabo utilizando una función que cargaba los datos desde el archivo CSV y luego dividía los datos en ventanas del tamaño especificado. Además, se realizó una transformación en las etiquetas para convertirlas en un formato binario, donde 1 representaba la presencia de una caída y 0 indicaba una actividad normal.

Este proceso garantizó que la red neuronal recibiera datos de entrada estructurados en forma de secuencias temporales, lo que facilitó su entrenamiento para la detección precisa de caídas.

## **Revisión de Métricas y Descubrimiento de TensorBoard**

A medida que avanzaba el proceso de entrenamiento, se identificó un problema importante relacionado con la métrica de precisión (accuracy). Se observó que esta métrica no reflejaba adecuadamente el desempeño del modelo en la detección de caídas debido al desbalance de clases en el conjunto de datos. Para abordar este problema, se dedicó tiempo a investigar y seleccionar métricas más apropiadas. Fue entonces cuando se descubrió TensorBoard, una herramienta que permitía inspeccionar visualmente las ejecuciones de TensorFlow y los resultados del entrenamiento (Figura 10).

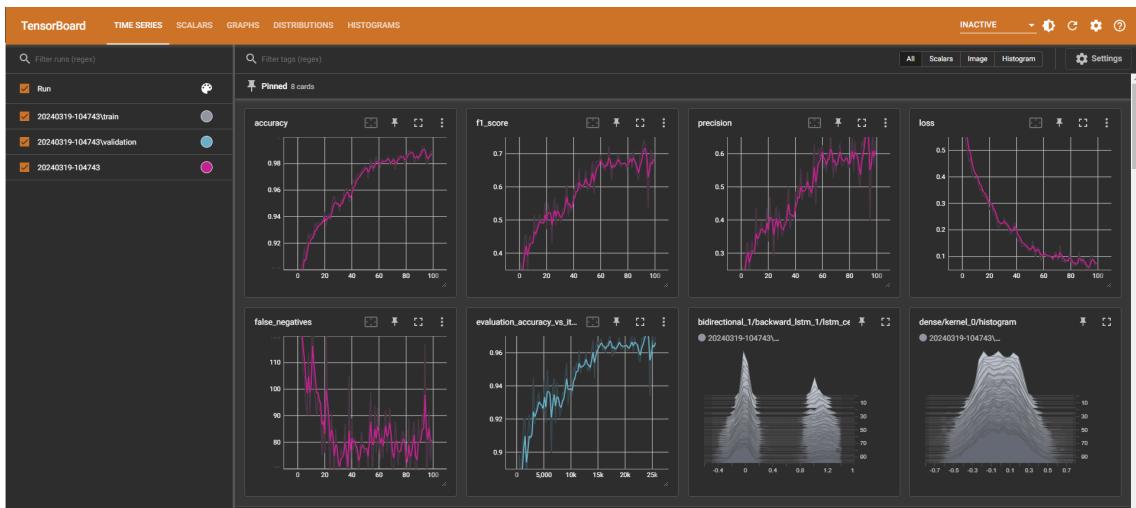


Figura 10: Visualización de gráficos en TensorBoard

## Experimentación y Ajuste Continuo

Se llevaron a cabo múltiples iteraciones de entrenamiento, experimentando con diferentes configuraciones y arquitecturas de red. El proceso de ajuste continuo permitió mejorar progresivamente el rendimiento del modelo en la detección de caídas.

## Resultados y Conclusiones

Al concluir esta fase, logramos con éxito entrenar una red neuronal bilateral LSTM para la detección de caídas, utilizando un enfoque meticuloso y riguroso. La adopción de un conjunto de datos más complejo y extenso, combinado con la optimización de hiperparámetros y el uso de métricas apropiadas, condujo a mejoras significativas en el rendimiento del modelo. El empleo de TensorBoard proporcionó una valiosa perspectiva visual sobre el proceso de entrenamiento, permitiendo una evaluación más precisa y una comprensión más profunda del desempeño del modelo.

## Evaluación y Ajustes

Durante esta fase del proyecto, se llevó a cabo la evaluación detallada de la red neuronal desarrollada, así como la realización de ajustes necesarios para mejorar su desempeño en la detección de caídas. Esta tarea se desarrolló de forma paralela a la tarea 2.2, aprovechando el proceso de entrenamiento de la red neuronal para realizar análisis continuo y ajustes iterativos.

## ***Herramientas Utilizadas***

### **TensorBoard**

Se utilizó TensorBoard como una herramienta fundamental para el análisis y la evaluación de las métricas de rendimiento de la red neuronal. Permitió visualizar y comparar el desempeño de los diferentes modelos entrenados.

## ***Metodología***

### **Análisis de Métricas de Rendimiento**

Se realizaron análisis exhaustivos de las métricas de rendimiento, como la precisión, la pérdida y la matriz de confusión entre otras métricas relevantes, utilizando TensorBoard. Este análisis proporcionó una visión detallada del desempeño de la red neuronal y ayudó a identificar áreas para mejoras.

### **Ajustes Iterativos**

Basándonos en los resultados del análisis de métricas, se llevaron a cabo ajustes iterativos en los parámetros y la arquitectura del modelo. Estos ajustes se realizaron con el objetivo de mejorar la capacidad de la red neuronal para detectar caídas con precisión y sensibilidad.

## ***Procedimientos***

### **Comparación de Modelos**

Se entrenaron y evaluaron múltiples modelos de redes neuronales con diferentes configuraciones y parámetros. Se compararon los resultados obtenidos por cada modelo para determinar cuál proporcionaba el mejor desempeño en la detección de caídas.

Todos los modelos evaluados en esta comparación comparten el mismo tamaño de ventana y el mismo tamaño de lote.

Además de estos elementos comunes, cada modelo se distingue por su configuración específica de parámetros, incluyendo el número de unidades en cada capa, el ajuste de pesos y otros hiperparámetros que influyen en su rendimiento y capacidad de detección de caídas.

| Modelo | Épocas | Overlap | Arquitectura                               | Mejor Época | Ajuste de Pesos | Tiempo  |
|--------|--------|---------|--|-------------|-----------------|---------|
| 1      | 100    | 50      | Bi-LSTM - 2 capas<br>64 y 32 unidades      | 89          | -               | 1.172 h |
| 2      | 100    | 50      | Bi-LSTM - 3 capas<br>128, 63 y 32 unidades | 80          | -               | 2.164 h |
| 3      | 100    | 50      | Bi-LSTM - 3 capas<br>128, 63 y 32 unidades | 93          | 1 a 10          | 2.15 h  |
| 4      | 73     | 30      | Bi-LSTM - 3 capas<br>128, 63 y 32 unidades | -           | 1 a 20          | 2.735 h |
| 4(2)   | 71     | 30      | Bi-LSTM - 3 capas<br>128, 63 y 32 unidades | 71          | 1 a 20          | 2.654 h |

Tabla 2: Comparativa entre los modelos entrenados

**Épocas:** Representa el número de veces que el algoritmo de entrenamiento pasa por todo el conjunto de datos de entrenamiento. Cada época implica un ciclo completo de alimentación hacia adelante y hacia atrás a través de la red neuronal, seguido de la actualización de los pesos para mejorar la precisión del modelo.

**Tamaño de ventana:** Se refiere al tamaño de la ventana de tiempo utilizada para segmentar los datos de entrada. En el contexto de este proyecto, la ventana de tiempo define el intervalo de datos sobre el cual se aplican las operaciones de procesamiento de datos y entrenamiento del modelo.

**Overlap:** Indica el grado de superposición entre ventanas adyacentes durante la segmentación de los datos de entrada. Un mayor valor de superposición significa

que hay más datos en común entre ventanas consecutivas, lo que puede influir en la capacidad del modelo para capturar patrones temporales en los datos.

**Tamaño de Lote:** Se refiere al número de muestras de datos que se utilizan en cada paso de actualización de los pesos del modelo durante el entrenamiento. El uso de lotes permite una actualización más eficiente de los pesos y puede mejorar la estabilidad del entrenamiento.

**Arquitectura:** Describe la estructura y disposición de las capas neuronales en el modelo de red neuronal. En este caso, se utiliza una arquitectura Bi-LSTM (Long Short-Term Memory Bidireccional), que es una variante de las redes neuronales recurrentes (RNN) diseñada para capturar dependencias a largo plazo en secuencias de datos. Se indica el número de unidades por cada capa.

**Mejor Época:** Se refiere a la época en la que el modelo alcanza su rendimiento más destacado durante el proceso de entrenamiento y evaluación. Este resultado se determina mediante una combinación de métricas de evaluación, como precisión, recall y F1-score. En este caso particular, se evalúa el desempeño del modelo en cada época y se selecciona la época en la que se obtiene el mejor resultado en función de la suma de la precisión, el recall y el doble del puntaje F1. Este enfoque proporciona una medida global del desempeño del modelo en un momento dado del entrenamiento.

**Ajuste de Pesos:** Se refiere a la configuración de los pesos utilizados durante el entrenamiento del modelo para compensar el desequilibrio entre las clases de datos, dando mayor importancia a la detección de caídas en comparación con otras actividades. Esta configuración es fundamental para mejorar el rendimiento del modelo al abordar el desafío de clasificar con precisión las instancias de caídas frente a las no caídas, optimizando así la sensibilidad y la precisión de la detección de caídas.

**Tiempo:** El tiempo hace referencia al tiempo requerido para entrenar completamente el modelo utilizando el conjunto de datos especificado. Este tiempo incluye el procesamiento de todas las épocas de entrenamiento. Es un factor importante a considerar, ya que puede influir en la eficiencia y la escalabilidad del proceso de entrenamiento del modelo.

## **Optimización de Umbrales**

Se ajustaron los umbrales de decisión del modelo para optimizar su desempeño en la detección de caídas. Estos ajustes se basaron en el análisis detallado de las métricas de rendimiento y la comprensión de las características específicas del conjunto de datos.

## **Gráficos Comparativos**

En los gráficos comparativos de rendimiento de los modelos, se utilizan colores específicos para representar cada modelo:

| Modelo | Modelo 1 | Modelo 2 | Modelo 3 | Modelo 4(1) | Modelo 4(2) |
|--------|----------|----------|----------|-------------|-------------|
| Color  | Rosa     | Verde    | Azul     | Morado      | Gris        |

Tabla 3: Leyenda de colores de los modelos

Modelo 4(2) - Entrenamiento repetido: Para el Modelo 4, se utiliza el color morado para representar el primer intento de entrenamiento, mientras que el gris representa el segundo intento. Esto se debe a que, debido a limitaciones de potencia de procesamiento, el Modelo 4 requirió ser entrenado dos veces para completar el proceso.

Es importante destacar que las medidas reportadas en los gráficos se basan en un valor umbral de predicción de 0.5 para clasificar las instancias. Además de las métricas comunes como precisión, recall y F1-score, también se incluyen métricas como la pérdida (loss) y el número de falsos negativos para proporcionar una evaluación exhaustiva del rendimiento de cada modelo.

Los gráficos comparativos muestran el rendimiento de cada modelo en varias métricas importantes. Cada gráfico tiene métricas específicas en el eje vertical y las épocas de entrenamiento en el eje horizontal. Las líneas o barras de diferentes colores representan los diferentes modelos, como se detalla anteriormente.

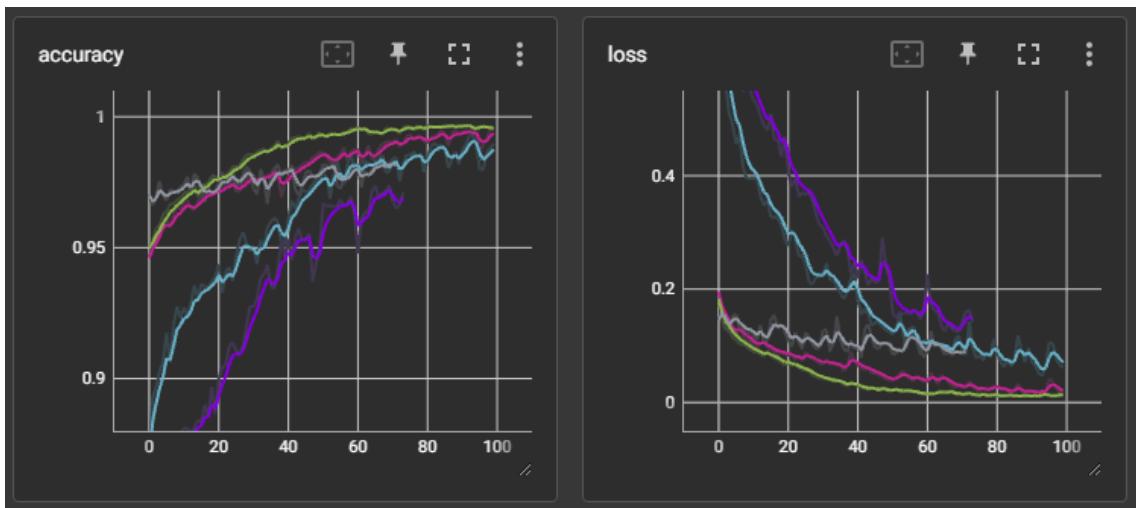


Figura 11: Comparación Accuracy y Loss de los diferentes modelos

**Accuracy (Precisión):** Mide la proporción de predicciones correctas realizadas por el modelo.

**Loss (Pérdida):** Representa la función de pérdida utilizada durante el entrenamiento del modelo, que indica cómo de bien está aprendiendo el modelo.

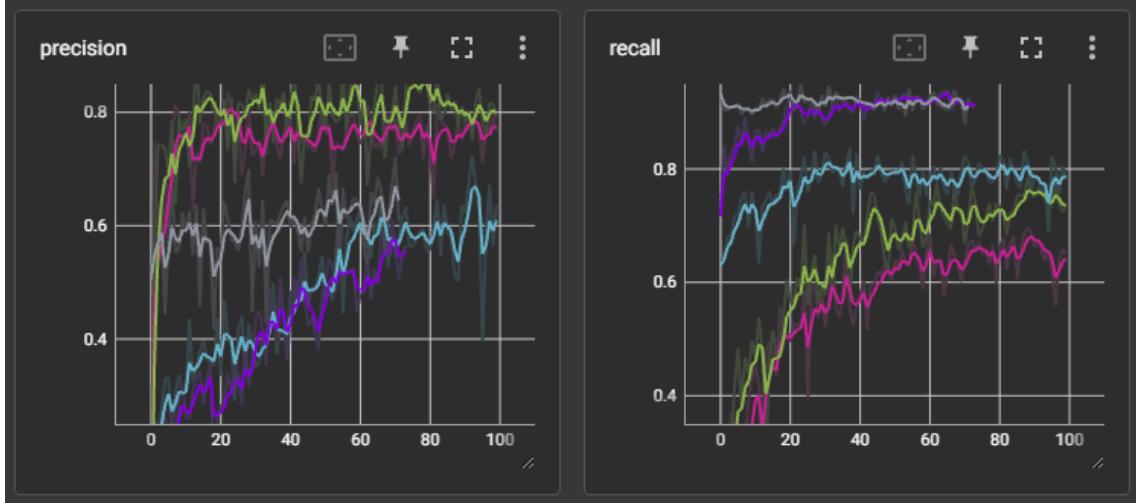


Figura 12: Comparación Precision y Recall de los diferentes modelos

**Precision (Precisión):** Indica la proporción de instancias clasificadas como positivas que son realmente positivas.

**Recall (Recuperación o Sensibilidad):** Mide la proporción de instancias positivas que fueron correctamente detectadas por el modelo.

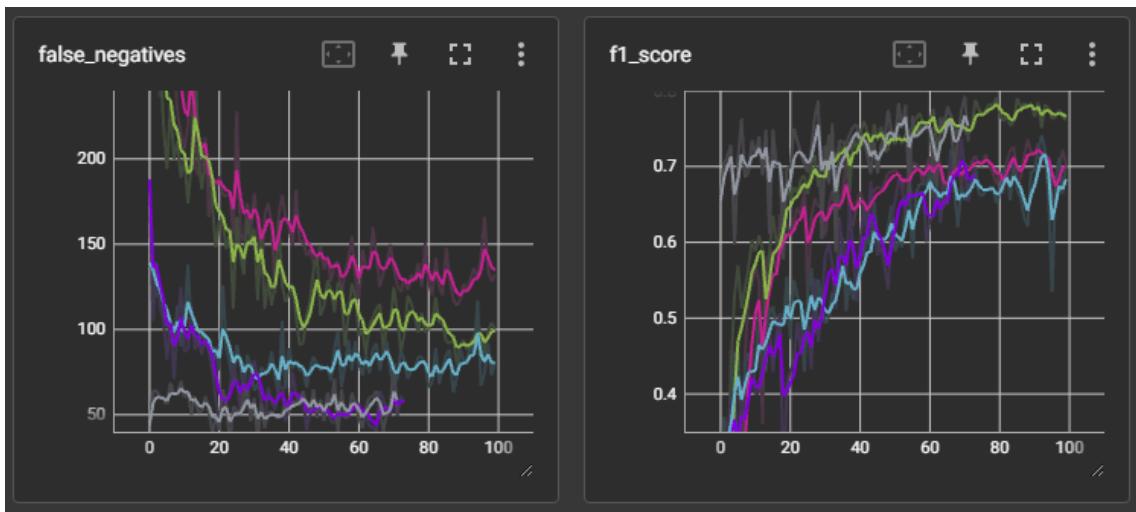


Figura 13: Comparación False Negatives y F1-Score de los diferentes modelos

**F1-score:** Es una medida que combina precisión y recall en un solo número, proporcionando una evaluación más equilibrada del rendimiento del modelo.

**False Negatives (Falsos Negativos):** Representa el número de instancias que fueron falsamente clasificadas como negativas cuando en realidad eran positivas. Esto es especialmente crítico en el contexto de detección de caídas, ya que los falsos negativos pueden significar que una caída real no fue detectada por el modelo.

### Comparación de Predicciones con Diferentes Umbrales

Para evaluar y comparar las predicciones de cada modelo de la red neuronal con diferentes umbrales de decisión, llevamos a cabo un análisis detallado que nos permitió determinar el mejor modelo y umbral para la detección de caídas.

Establecimos una serie de umbrales de decisión que van desde 0.1 hasta 0.9, con incrementos de 0.1. Estos umbrales nos permitieron ajustar la sensibilidad y especificidad del modelo para clasificar las instancias como caídas o no caídas.

Utilizando cada modelo entrenado, generamos predicciones para el conjunto de datos de prueba utilizando los umbrales definidos. Cada predicción se clasificó como caída o no caída según el umbral correspondiente.

Para cada umbral, calculamos métricas de evaluación como precisión, recall, F1-score y matriz de confusión para cada modelo. Estas métricas nos proporcionaron

información sobre el rendimiento de cada modelo en la detección de caídas con diferentes umbrales.

A continuación, se pueden ver los gráficos que representan el rendimiento de cada modelo con diferentes umbrales. En estos gráficos, cada métrica se muestra en el eje vertical, mientras que los umbrales se representan en el eje horizontal. Cada modelo se distingue por un color específico para facilitar una comparación visual clara.

## Modelo 1

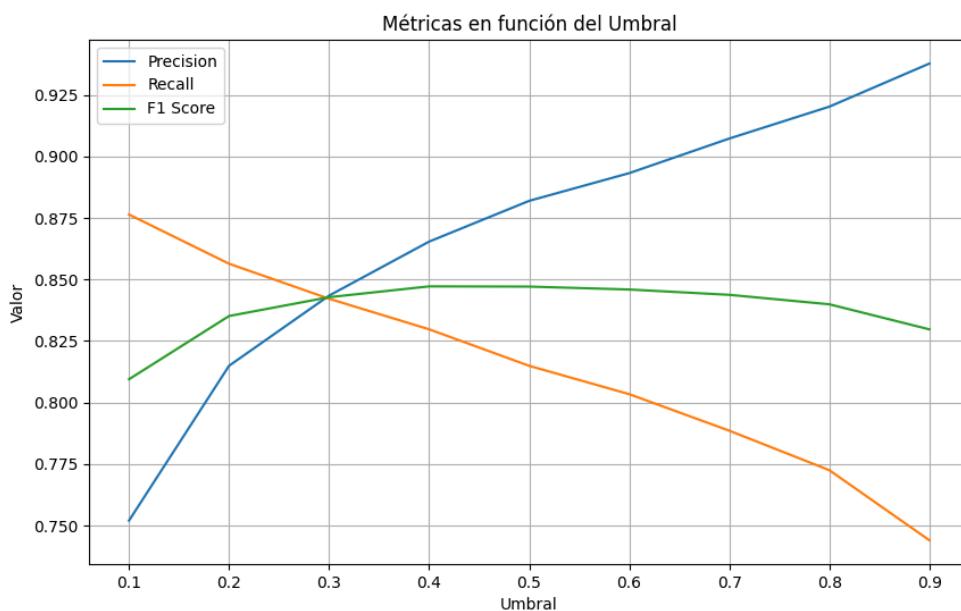


Figura 14: Comparativa Precision, Recall y F1-Score del Modelo 1

Después de revisar los datos (Figura 14), observamos que con un umbral de 0.4, logramos un recall cercano al 0.830 y una precisión de alrededor del 0.865. Estos resultados indican que, a este nivel de umbral, estamos identificando la mayoría de las caídas reales mientras mantenemos una precisión sólida.

Elegir un umbral de 0.4 nos permite optimizar la capacidad del modelo para detectar las caídas reales, reduciendo así los falsos negativos, al tiempo que mantenemos una precisión aceptable para evitar un exceso de detecciones incorrectas. Esta combinación nos permite alcanzar nuestro objetivo de minimizar los falsos negativos sin comprometer demasiado la precisión del modelo.

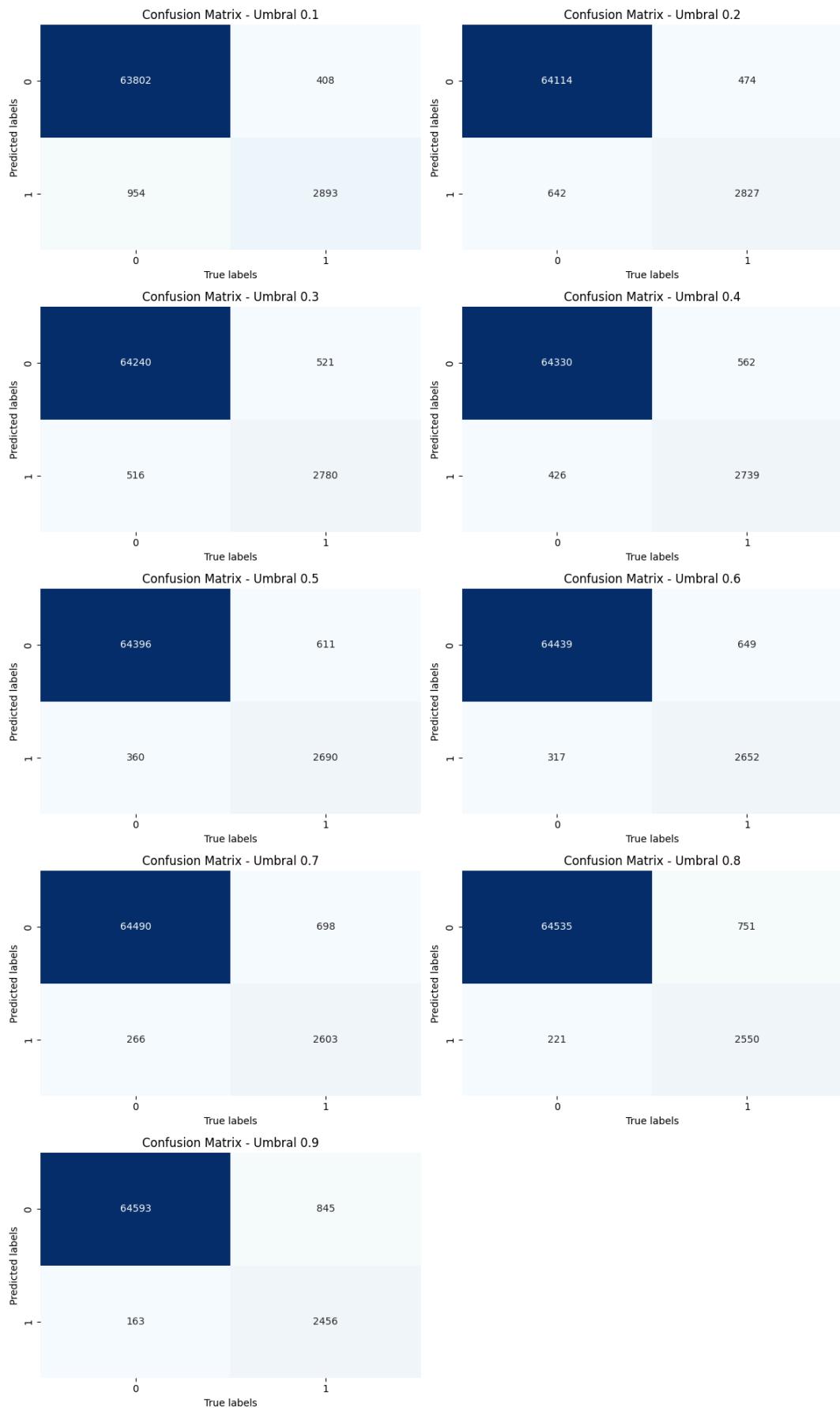


Figura 15: Matrices de Confusión por umbrales del Modelo 1

## Modelo 2

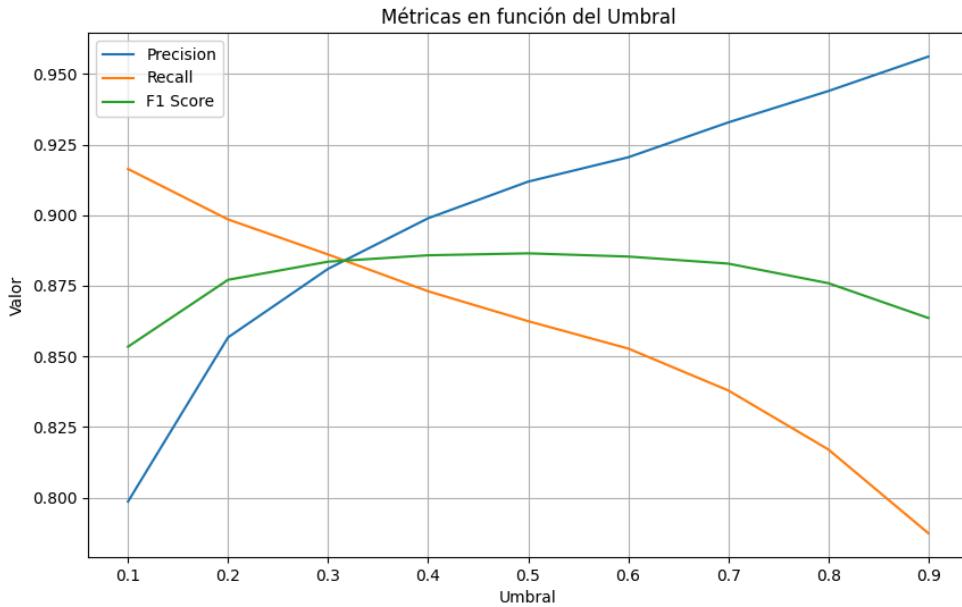


Figura 16: Comparativa Precision, Recall y F1-Score del Modelo 2

Después de analizar los datos (Figura 16), observamos que con un umbral de 0.3, alcanzamos un recall de aproximadamente 0.886 y una precisión de alrededor de 0.881. Esto sugiere que, a este nivel de umbral, estamos identificando una gran proporción de las caídas reales mientras mantenemos una precisión sólida.

Seleccionar un umbral de 0.3 nos permite maximizar la capacidad del modelo para detectar las caídas reales, reduciendo así los falsos negativos, al tiempo que mantenemos una precisión aceptable para evitar un exceso de detecciones incorrectas. Este equilibrio nos permite lograr nuestro objetivo de minimizar los falsos negativos sin comprometer demasiado la precisión del modelo.

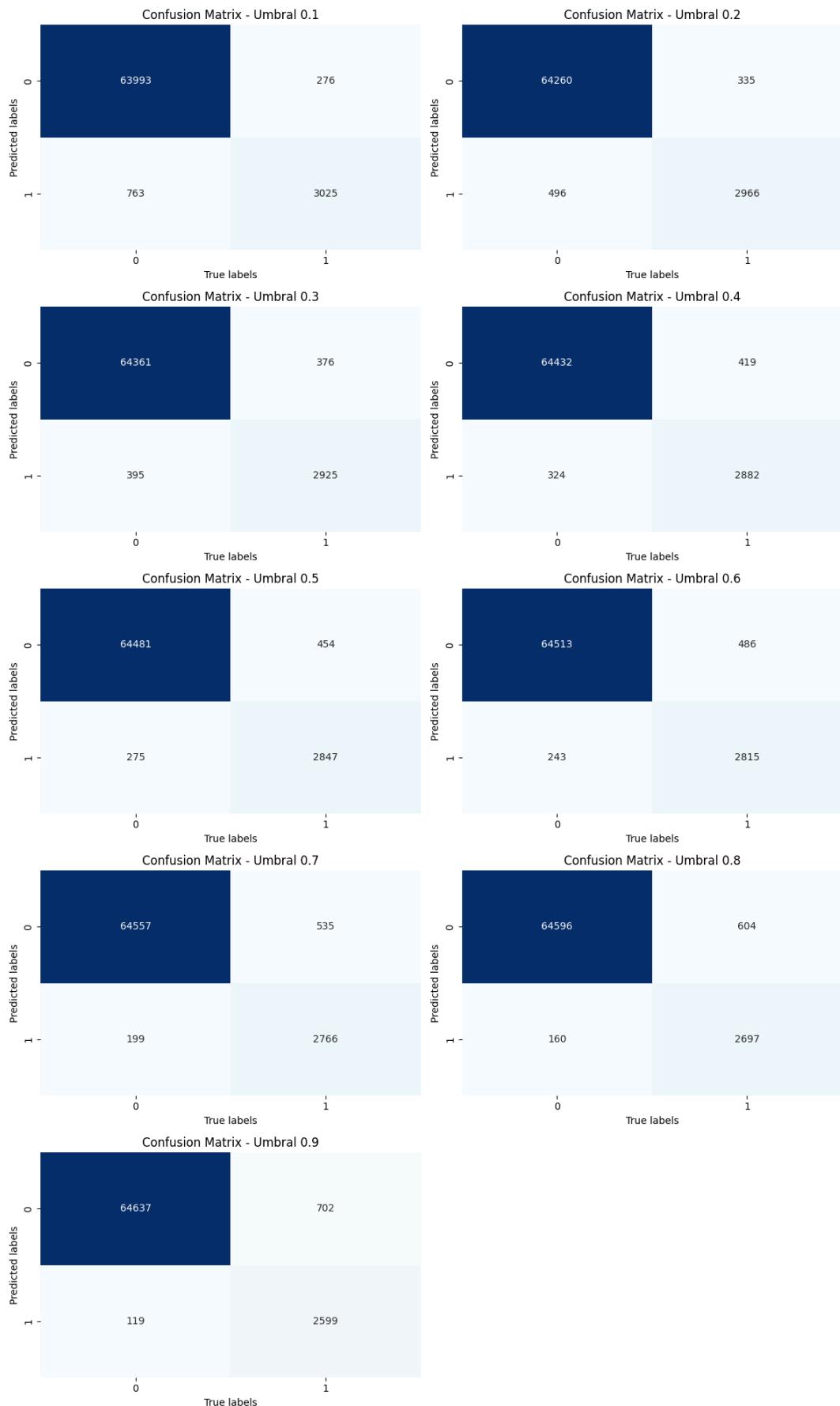


Figura 17: Matrices de Confusión por umbrales del Modelo 2

### Modelo 3

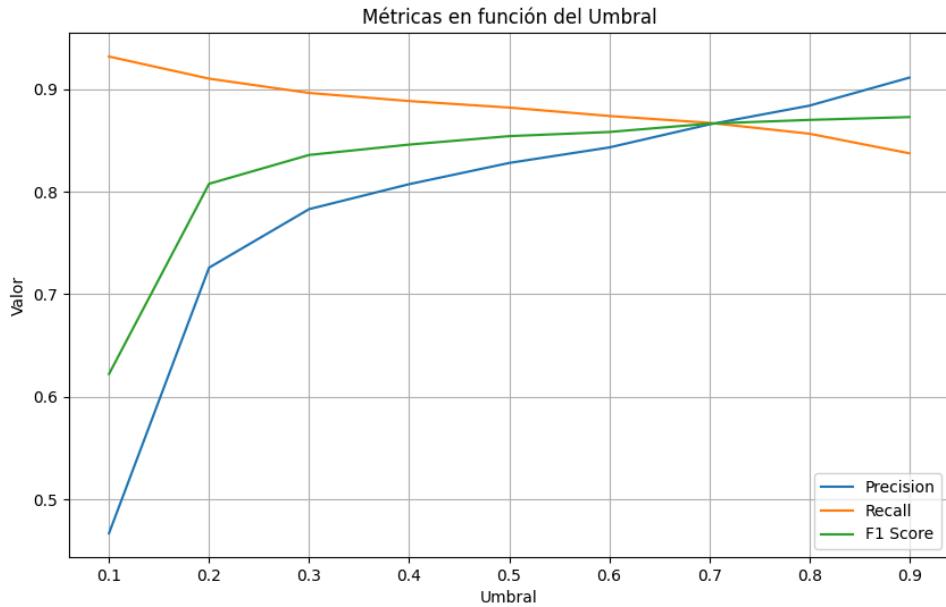


Figura 18: Comparativa Precision, Recall y F1-Score del Modelo 3

Después de revisar los datos (Figura 18), notamos que con un umbral de 0.8, la precisión es de aproximadamente 0.884 y el recall es de alrededor de 0.856. Esto indica que al utilizar este umbral, mantenemos una precisión alta mientras capturamos una proporción significativa de caídas reales.

Por lo tanto, seleccionar un umbral de 0.8 nos permite maximizar la precisión del modelo al mismo tiempo que minimizamos los falsos negativos. Aunque el recall no alcanza los niveles de algunos umbrales más bajos, la alta precisión asegura que las detecciones positivas sean altamente confiables. Esta es una consideración importante en aplicaciones donde se valora la minimización de los falsos positivos y se requiere una alta precisión en la detección de caídas.

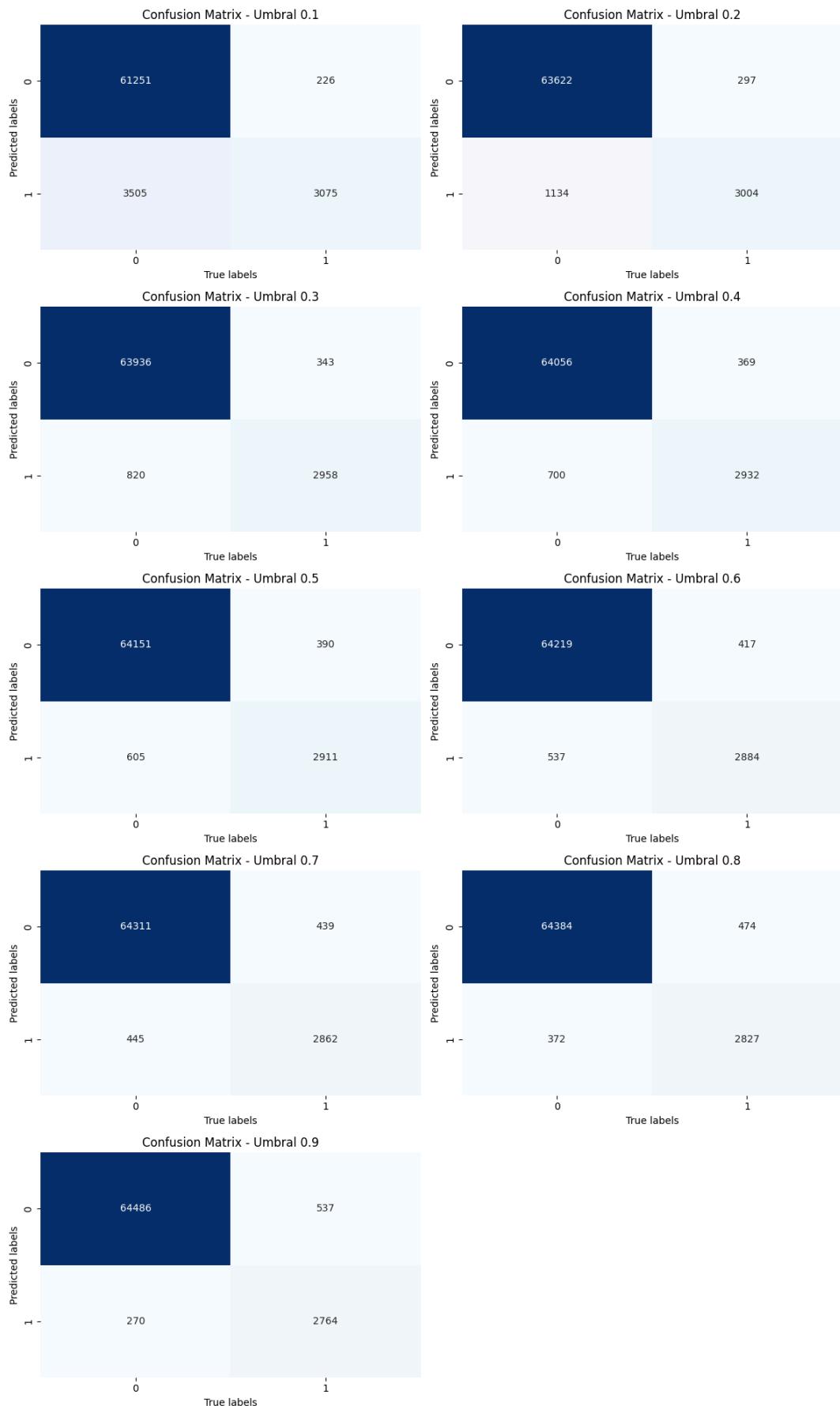


Figura 19: Matrices de Confusión por umbrales del Modelo 3

## Modelo 4

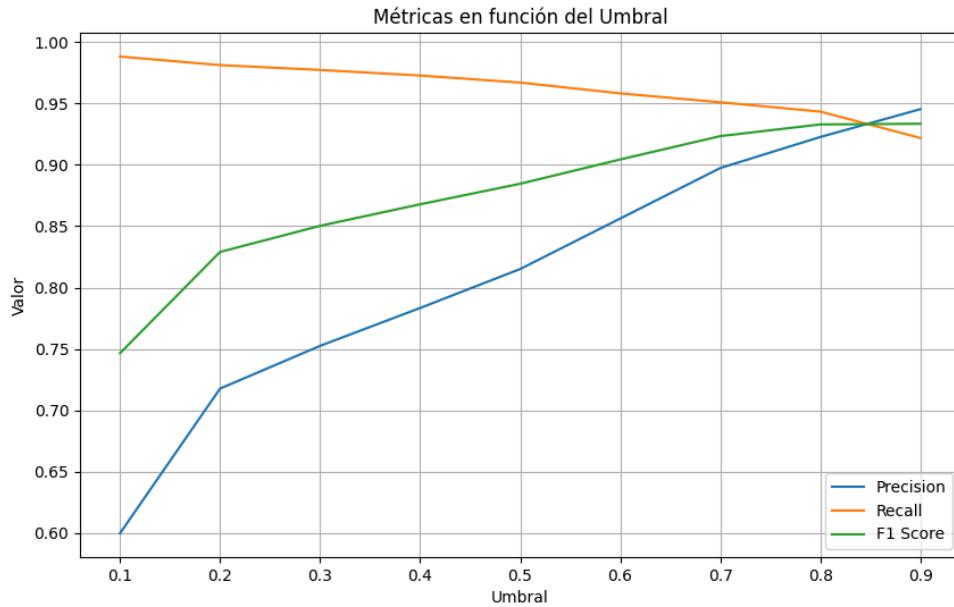


Figura 20: Comparativa Precision, Recall y F1-Score del Modelo 4

Al examinar los datos, observamos que a un umbral de 0.8, la precisión es de aproximadamente 0.923 y el recall es de aproximadamente 0.943. Esto sugiere que a este umbral, estamos manteniendo una precisión alta mientras capturamos una proporción muy alta de caídas reales.

Por lo tanto, elegir un umbral de 0.8 nos permite maximizar la precisión del modelo mientras minimizamos los falsos negativos. Aunque el recall no es perfecto, la alta precisión garantiza que las detecciones positivas sean altamente confiables. Esto es crucial en aplicaciones donde es importante minimizar los falsos positivos y mantener una alta precisión en la detección de caídas.

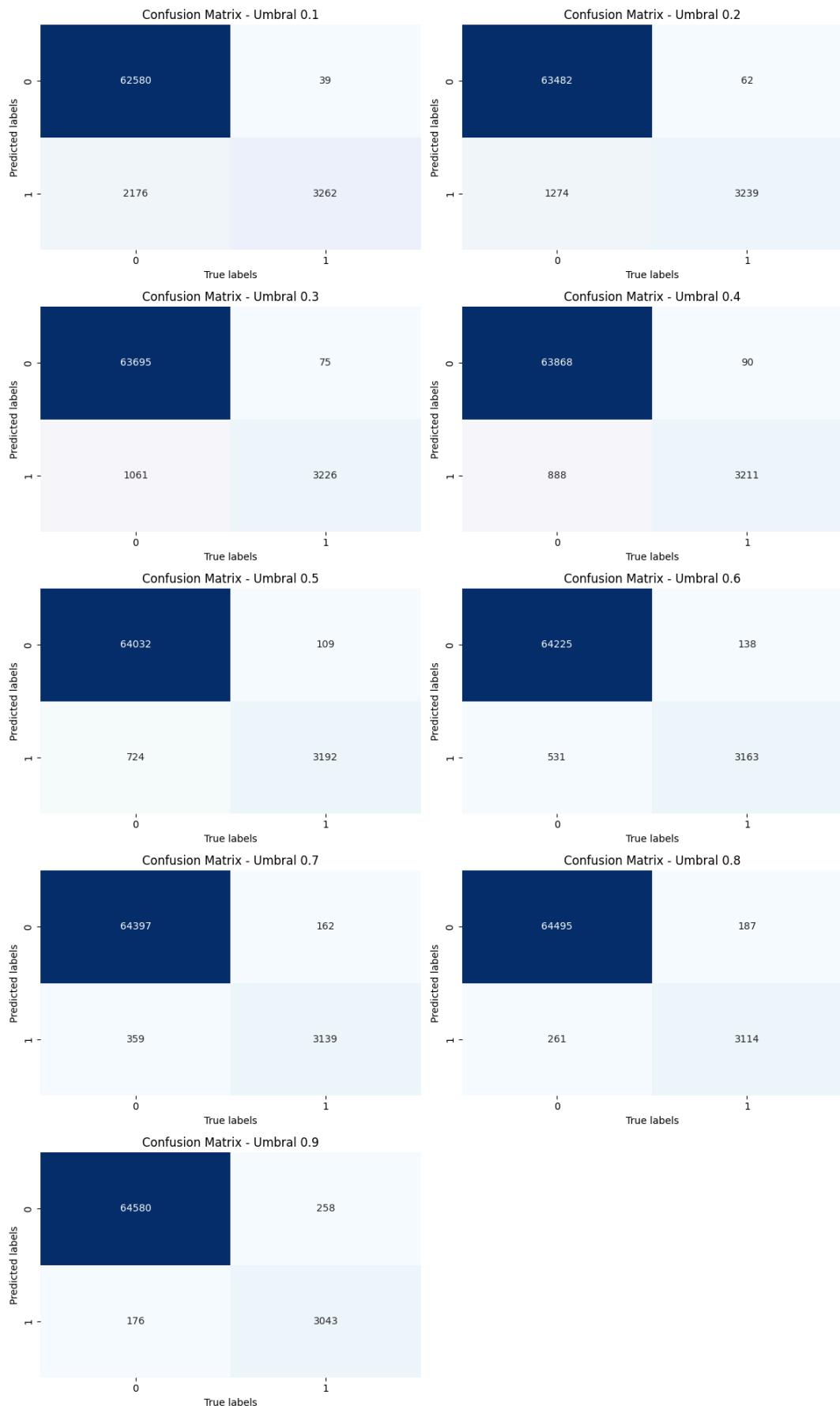


Figura 21: Matrices de Confusión por umbrales del Modelo 4

## **Resultados y Conclusiones**

Tras analizar detenidamente los resultados de los cuatro modelos entrenados, podemos concluir que el Modelo 4 ha demostrado ser el más efectivo en la detección de caídas. Este modelo alcanzó una precisión de aproximadamente 0.922 y un recall de aproximadamente 0.943, lo que indica una capacidad significativa para capturar caídas reales mientras mantiene una alta precisión en las detecciones. Comparado con los otros modelos evaluados, el Modelo 4 logra el equilibrio óptimo entre precisión y recall, lo que lo convierte en la elección preferida para la implementación final en el dispositivo wearable.

Al concluir esta fase, se ha logrado un avance sustancial en el desempeño de la red neuronal para la detección de caídas. La evaluación detallada y los ajustes iterativos nos han permitido identificar y mejorar áreas críticas, y la comparación entre múltiples modelos nos ha proporcionado una base sólida para tomar decisiones informadas. Con el Modelo 4 optimizado y listo para su implementación, estamos preparados para avanzar hacia la siguiente etapa de desarrollo del dispositivo wearable.

## **Selección del Hardware**

La selección del hardware es un papel crucial en el desarrollo de sistemas embebidos. En esta sección, se detalla el proceso de selección del hardware fundamental para la implementación del sistema de detección de caídas basado en redes neuronales. Exploraremos las diversas opciones de microcontroladores disponibles y analizaremos las alternativas de sensores iniciales para recopilar datos de aceleración y giroscopio.

## **Herramientas Utilizadas**

Para llevar a cabo el desarrollo del sistema de detección de caídas basado en redes neuronales, se emplearon las siguientes herramientas:

### **Microcontroladores:**

- ESP32
- Arduino Nano 33 Sense BLE
- ESP32-S3 Touch LCD 1.28

### **IDE Desarrollo:**

- Arduino IDE 2.3.2

### **Sensores Inerciales:**

- GY-511 (IMU)
- LSM303DLHC (Acelerómetro)
- MPU9150 (Giroscopio)
- QMI8658 (IMU)

## **Metodología**

### **Evaluación de Microcontroladores**

Se realizaron pruebas exhaustivas con los microcontroladores disponibles, centrándose en la viabilidad de integrar la red neuronal en el dispositivo seleccionado. Se evaluaron las capacidades de los microcontroladores ESP32 y Arduino Nano 33 Sense, considerando aspectos como potencia de procesamiento, memoria disponible y compatibilidad con bibliotecas de machine learning.

### **Investigación de Sensores Inerciales**

Se llevó a cabo una investigación sobre los sensores inerciales disponibles en el mercado. Se buscó identificar aquellos que pudieran capturar datos de aceleración y giroscopio con precisión y compatibilidad con el modelo de red neuronal.

### **Pruebas y Comparación de Sensores**

Se adquirieron varios sensores iniciales, incluyendo GY-511, STMicro LSM303DLHC e Invensense MPU9150. Estos sensores se sometieron a pruebas para evaluar su rendimiento en condiciones reales y su compatibilidad con los requisitos del proyecto.

## **Procedimientos**

### **Pruebas con Microcontroladores**

Se llevaron a cabo pruebas con los microcontroladores ESP32 y Arduino Nano 33 Sense para evaluar su capacidad de integrar la red neuronal en el dispositivo. Inicialmente, se consideró la opción de integrar directamente la red neuronal en el microcontrolador, lo que requeriría una conversión del modelo a una versión compatible con los recursos limitados del microcontrolador. Sin embargo, esta opción se descartó debido a las dificultades encontradas durante el proceso de migración. Algunas capas utilizadas en el modelo no eran compatibles o no existían en TensorFlow Lite, lo que limitaba su viabilidad. Se exploraron alternativas, como la posibilidad de crear una aplicación móvil para comunicarse con el microcontrolador a través de Bluetooth, pero se descartaron debido a su complejidad y coste. Finalmente, se optó por utilizar la ESP32 debido a su capacidad de conexión Wi-Fi, lo que permitiría establecer una comunicación con un servidor web de forma más sencilla.

### **Evaluación de Sensores Iniciales**

Se realizaron pruebas con varios sensores iniciales, incluyendo GY-511, STMicro LSM303DLHC e Invensense MPU9150, para determinar su idoneidad para capturar datos de aceleración y giroscopio. Al usar la IMU GY-511 se pudieron leer los valores de aceleración y giro pero fue difícil configurar estos sensores para que fueran compatibles con el conjunto de datos utilizado para entrenar la red neuronal. Se investigaron alternativas y se identificaron los sensores utilizados en el dataset deseado. Se adquirieron y probaron los sensores STMicro LSM303DLHC e Invensense MPU9150, pero se encontró que configurarlos adecuadamente resultaba complicado debido a la falta de información sobre la configuración usada para la obtención del dataset.

Durante las pruebas, se observó que ambos sensores arrojaban mediciones prácticamente idénticas, a pesar de que la IMU no era utilizada para la obtención del dataset, mientras que los otros sensores sí lo eran. Dado que no se disponía de información sobre la configuración del dataset, se concluyó que cualquier sensor inercial sería adecuado para el proyecto. Por lo tanto, se buscó una solución integral que incluyera una IMU compatible, lo que llevó a la selección de la Placa de Desarrollo ESP32-S3 Touch LCD 1.28", que incorpora la IMU QMI8658.

### **Selección de Placa de Desarrollo ESP32-S3 Touch LCD 1.28"**

Tras una exhaustiva evaluación de las opciones disponibles, se optó por adquirir y utilizar la placa de desarrollo ESP32-S3 Touch LCD 1.28". Esta placa proporcionaba una solución integral que incluía un microcontrolador ESP32-S3, un display táctil y una IMU con sensores de aceleración y giroscopio integrados, ofreciendo todas las funcionalidades necesarias para el desarrollo del dispositivo de detección de caídas.

## **Desarrollo del Software**

En esta fase, se desarrolló el software necesario para integrar la red neuronal en el hardware seleccionado y gestionar la captura, procesamiento y análisis de datos de los sensores inerciales.

### **Herramientas Utilizadas**

Para llevar a cabo el desarrollo del software, se emplearon las siguientes herramientas:

**Entorno de Desarrollo Integrado (IDE):** Arduino IDE 2.3.2

**Placa de Desarrollo:** ESP32-S3 Touch LCD 1.28"

**Librerías y placas:**

| Librería         | Descripción   | Enlace  |
|------------------|---|---|
| Adafruit_GFX     | Facilita el dibujo de gráficos en pantallas en dispositivos Arduino.                            | <a href="https://github.com/adafruit/Adafruit-GFX-Library">https://github.com/adafruit/Adafruit-GFX-Library</a> |
| Adafruit_GC9A01A | Controla el controlador de pantalla GC9A01A, común en pantallas TFT.                            | <a href="https://github.com/Adafruit/Adafruit_GC9A01A">https://github.com/Adafruit/Adafruit_GC9A01A</a>         |
| WiFi             | Permite la conexión a redes Wi-Fi y la comunicación a través de ellas.                          | -   |
| NTPClient        | Sincroniza la hora del sistema con servidores de tiempo usando el protocolo NTP.                | <a href="https://github.com/arduino-libraries/NTPClient">https://github.com/arduino-libraries/NTPClient</a>     |
| WiFiUdp          | Implementa UDP sobre Wi-Fi para comunicaciones de bajo nivel.                                   | -   |
| TimeLib          | Proporciona funciones para manipular y trabajar con el tiempo en Arduino.                       | -   |
| CST816S          | Interactúa con pantallas táctiles CST816S para procesar la entrada táctil del usuario.          | <a href="https://github.com/fbiego/CST816S">https://github.com/fbiego/CST816S</a>                               |
| SensorQMI8658    | Trabaja con el sensor de movimiento QMI8658 para detectar movimientos y cambios de orientación. | <a href="https://github.com/lewisxhe/SensorLib">https://github.com/lewisxhe/SensorLib</a>                       |

Tabla 4: Lista de librerías usadas para el desarrollo del software.

Para instalar placas adicionales en Arduino IDE, se utiliza un archivo JSON que contiene información sobre el paquete de soporte para esa placa. Este archivo se agrega a la configuración de Arduino IDE, lo que permite que el entorno reconozca y soporte la placa durante el desarrollo. Además, este archivo JSON puede incluir enlaces a bibliotecas adicionales que se instalan automáticamente junto con la placa, simplificando el proceso de desarrollo al proporcionar acceso directo a las herramientas y recursos necesarios. En mi caso, las bibliotecas que no tienen enlace se deben a que son instaladas por el fichero .json de forma automática.

[https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json)

## **Metodología**

El desarrollo del software se llevó a cabo siguiendo las siguientes fases:

### **Configuración del Entorno de Desarrollo**

Se realizó la configuración del entorno de desarrollo utilizando Arduino IDE 2.3.2 para la programación de la placa ESP32-S3 Touch LCD 1.28". Se instalaron las librerías necesarias para la comunicación con los sensores iniciales, la gestión de la pantalla táctil y la conexión a la red wifi para la posterior comunicación con el servidor.

El manual de configuración correspondiente se halla detallado en el [Anexo 2](#), abordando aspectos específicos sobre la configuración del entorno utilizado en el desarrollo, así como las características detalladas de la placa de desarrollo utilizada.

### **Desarrollo de Funcionalidades de Captura de Datos**

Se desarrollaron las funcionalidades para la captura de datos de los sensores iniciales, incluyendo la lectura de los valores de aceleración y giroscopio. Los datos capturados se almacenaron en memoria hasta alcanzar un paquete de 500 datos.

### **Uso de la Pantalla**

Se implementaron funciones para mostrar la hora actual en la pantalla, así como para mostrar un aviso visual en caso de detectar una caída. Además, se incorporó el control táctil para interactuar con la interfaz del dispositivo.

### **Conexión a Servidor Flask y Envío de Mensajes**

Se estableció una conexión Wi-Fi con el servidor Flask utilizando el protocolo POST para enviar datos de los sensores iniciales y recibir respuestas de predicción. Se desarrollaron funciones para manejar las respuestas del servidor y tomar acciones correspondientes según la predicción recibida.

## **Procedimientos**

### **Configuración del Entorno de Desarrollo**

Se configuró Arduino IDE 2.3.2 para la programación de la placa ESP32-S3 Touch LCD 1.28". Se instalaron las librerías necesarias para la comunicación con los sensores iniciales y la gestión de la pantalla táctil.

### **Desarrollo de Funcionalidades de Captura de Datos**

El desarrollo de las funcionalidades para la captura de datos de los sensores iniciales implicó una serie de pasos. En primer lugar, se procedió a utilizar librerías especializadas para la lectura de los valores de aceleración y giroscopio provenientes de los sensores conectados a la placa ESP32. Esta etapa requirió una cuidadosa implementación para asegurar la precisión y consistencia de los datos recolectados.

Una vez que los datos fueron adquiridos, se diseñó e implementó la lógica necesaria para su almacenamiento y envío. Se estableció un sistema que permitió agrupar los datos capturados en paquetes de 500 datos, lo que facilitaría su posterior procesamiento por el servidor.

Para garantizar el correcto funcionamiento del sistema, se llevaron a cabo una gran cantidad de pruebas. Estas pruebas se centraron en verificar la precisión de la captura de datos de los sensores iniciales y la fiabilidad de su transmisión. Se realizaron ajustes y mejoras según los resultados obtenidos en las pruebas, asegurando así la calidad y confiabilidad de la captura de datos del sistema.

### **Uso de la Pantalla**

Se hizo un gran trabajo en la experimentación y búsqueda de librerías para poder manejar correctamente la pantalla y poder mostrar la hora actual de manera precisa. Esto implicó la conexión a [pool.ntp.org](http://pool.ntp.org) para sincronizar el reloj interno de la ESP32 para garantizar que la hora mostrada fuera siempre precisa y actualizada.

Además, se diseñó y se incorporó un mecanismo de aviso visual que se activaría en caso de detectar una caída. Se desarrollaron funciones específicas que

permitieran mostrar una alerta visual en la pantalla en el momento en que se recibiera una predicción de caída del servidor.

Por último, se llevó a cabo la implementación del control táctil para posibilitar la interacción del usuario con la interfaz del dispositivo. Se crearon funciones encargadas de detectar y procesar las entradas táctiles, como pulsaciones y deslizamientos, permitiendo así una experiencia de usuario fluida e intuitiva.

### **Conexión a Servidor Flask y Envío de Mensajes**

Durante esta etapa, se llevó a cabo la configuración de una conexión Wi-Fi con el servidor Flask, empleando el protocolo POST para la transmisión de datos provenientes de los sensores iniciales, así como para recibir las respuestas de predicción del servidor.

Con el objetivo de gestionar de manera eficiente las respuestas del servidor, se procedió al desarrollo de funciones específicas. Estas funciones se encargaron de interpretar los mensajes recibidos y de ejecutar las acciones correspondientes según la predicción obtenida. Entre estas acciones se incluyó la activación de alertas visuales en caso de detectar una posible caída.

Posteriormente, se llevaron a cabo pruebas de integración para validar la adecuada comunicación con el servidor Flask. Durante estas pruebas, se emplearon herramientas de monitoreo de red para evaluar la estabilidad y la velocidad de la conexión, asegurando así un funcionamiento óptimo del sistema de comunicación con el servidor.

### **Resultados y Conclusiones**

El funcionamiento detallado del dispositivo se encuentra representado de manera gráfica en el siguiente diagrama de flujo (Figura 22). Este diagrama proporciona una visión general de cómo interactúan las diferentes partes del sistema, desde la captura de datos de los sensores hasta la comunicación con el servidor Flask y la activación de alertas visuales en caso de detectar una caída. El diagrama de flujo sirve como una guía visual que complementa la descripción detallada de cada fase del desarrollo del dispositivo.

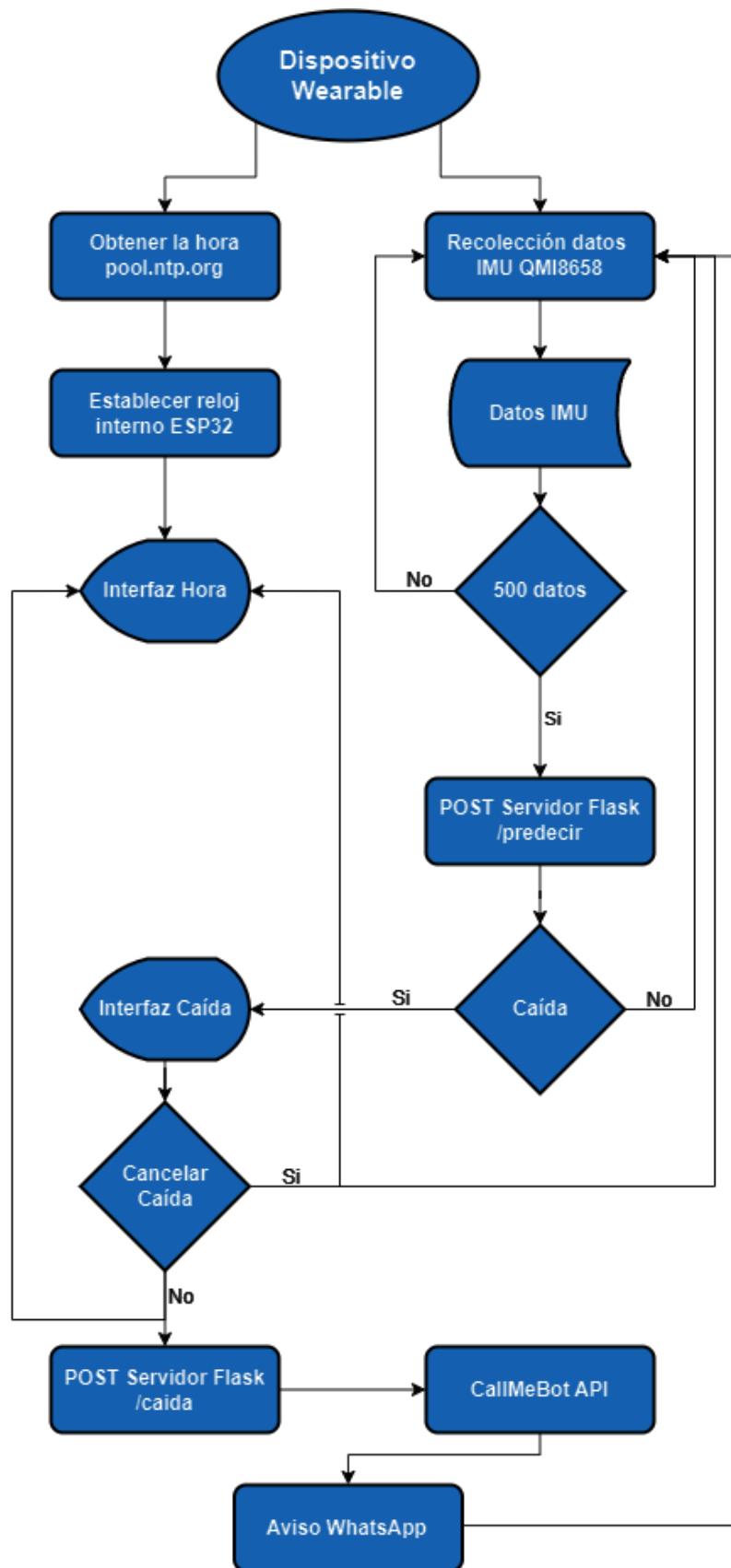


Figura 22: Diagrama de Flujo del Dispositivo Wearable

# **Integración de la Red Neuronal**

## ***Herramientas Utilizadas***

### **Entorno de Desarrollo:**

- Visual Studio Code

### **Framework de Machine Learning:**

- TensorFlow

### **Bibliotecas de Python:**

- Flask, numpy, time, requests

## ***Metodología***

### **Integración de la Red Neuronal**

Se procedió a cargar el modelo de red neuronal pre-entrenado en el servidor Flask. Utilizando TensorFlow, se cargó el modelo previamente entrenado para la detección de caídas. Se configuraron las rutas del servidor para manejar las solicitudes entrantes desde los dispositivos clientes y ejecutar el modelo sobre los datos recibidos.

## ***Procedimientos***

### **Carga del Modelo de Red Neuronal**

Se realizó la carga del modelo de red neuronal previamente entrenado en el servidor Flask. Esto se llevó a cabo utilizando TensorFlow y las funciones

proporcionadas por la biblioteca Keras para cargar el modelo guardado desde el archivo h5.

### **Configuración de Rutas en el Servidor Flask**

Se implementaron dos rutas principales en el servidor Flask para manejar las solicitudes entrantes desde los dispositivos clientes:

Ruta "/predecir": Esta ruta recibe los datos de los sensores enviados por los dispositivos clientes. Los datos son procesados y adaptados al formato necesario para realizar una predicción de caída utilizando el modelo de red neuronal cargado. Una vez completada la predicción, se devuelve el resultado al cliente.

Ruta "/caida": Cuando se recibe la confirmación de una caída en esta ruta, se realiza una conexión a la API de CallMeBot para enviar un mensaje de WhatsApp que notifica la caída a los contactos de emergencia previamente configurados.

### **Pruebas de Integración y Funcionamiento**

Se llevaron a cabo pruebas para verificar el correcto funcionamiento de las rutas implementadas en el servidor Flask. Se utilizaron herramientas de depuración y monitoreo para evaluar la estabilidad y la eficacia del sistema en situaciones simuladas de detección de caídas y notificación de emergencias.

### **Resultados y Conclusiones**

La integración efectiva de la red neuronal en el servidor Flask, junto con la configuración adecuada de rutas para manejar las solicitudes entrantes y las acciones posteriores a la detección de caídas, proporciona un sistema robusto y eficiente para la detección de caídas en tiempo real. Este enfoque integral garantiza una respuesta rápida y adecuada ante situaciones de emergencia, lo que mejora la seguridad y el bienestar de los usuarios.

En la Figura 23 se presenta el diagrama de flujo del servidor, que ilustra de manera visual el proceso detallado en la metodología y los procedimientos. Este diagrama proporciona una representación gráfica clara de cómo se gestionan las

solicitudes entrantes, se procesan los datos de los sensores, se realiza la predicción de caídas y se activa la notificación de emergencia en caso necesario.

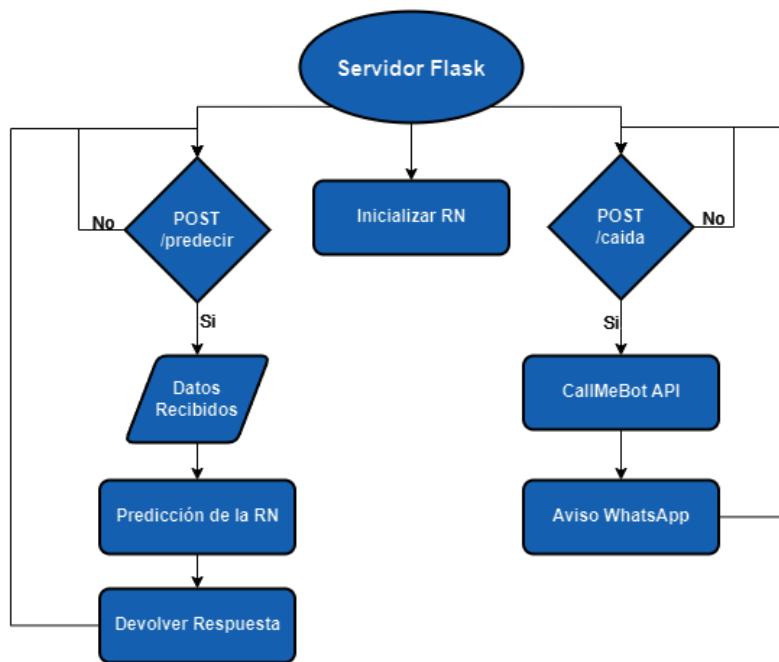


Figura 23: Diagrama de Flujo del Servidor Flask

# Pruebas, Resultados y Discusión

## Pruebas de las Redes Neuronales Entrenadas

Se llevaron a cabo numerosas pruebas de las redes neuronales entrenadas para la detección de caídas. El proceso de evaluación se dividió en varias etapas, que incluyeron el entrenamiento de las redes neuronales, la selección del mejor umbral para la clasificación de las predicciones y la evaluación del rendimiento de cada modelo.

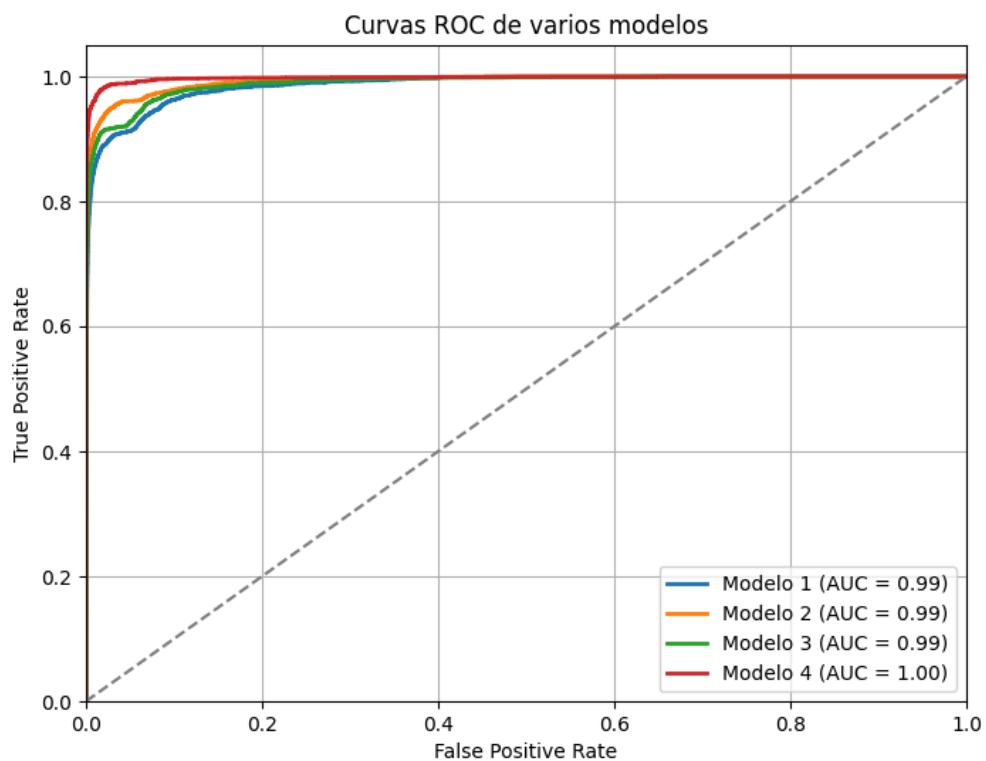


Figura 24: Comparativa Curva ROC entre los modelos entrenados

## Entrenamiento de las Redes Neuronales

Inicialmente, se entrenaron cuatro redes neuronales utilizando el conjunto de datos mencionado anteriormente. Se dividió el conjunto de datos en un 80% para entrenamiento y un 20% para pruebas. Sin embargo, surgieron algunos problemas

durante esta fase del proceso. La limitada potencia de cálculo y la memoria disponible en el hardware utilizado (una GTX 1060 6GB en un ordenador personal) resultaron ser obstáculos significativos. Cuando se intentaba forzar la complejidad o el tamaño de la red neuronal, se superaba la capacidad de la tarjeta gráfica, lo que provocaba la interrupción del entrenamiento y la necesidad de reiniciar el proceso. Además, la prolongada duración de los entrenamientos debido a la limitada potencia de cálculo fue otro inconveniente importante.

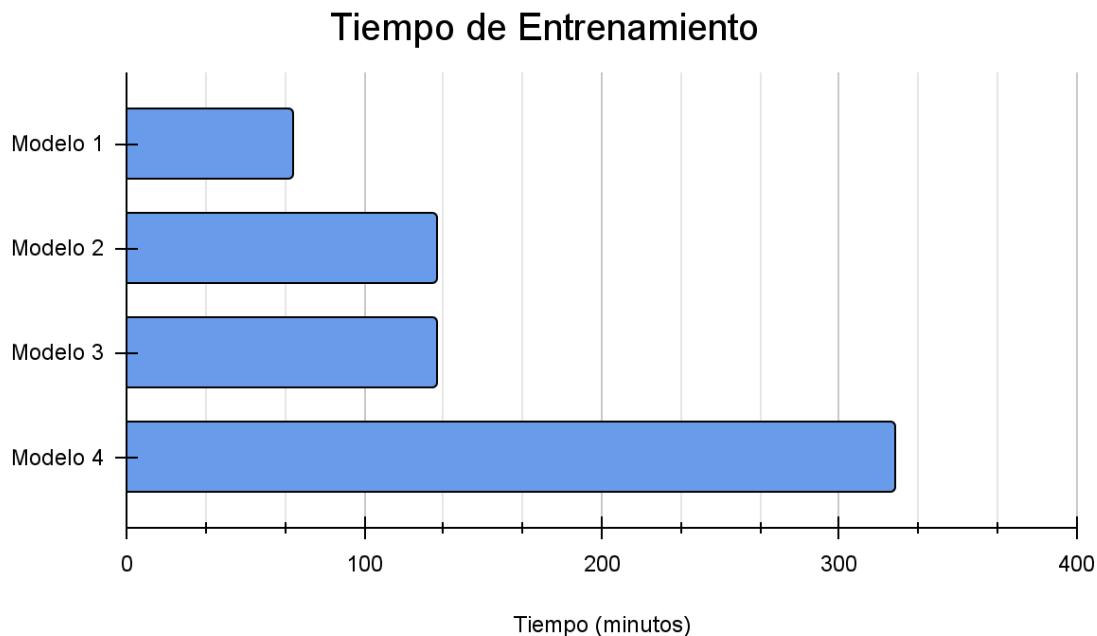


Figura 25: Gráfica comparativa del tiempo de entrenamiento

### ***Selección del Mejor Umbral de Clasificación***

Una vez entrenadas las redes neuronales, se procedió a evaluar su rendimiento utilizando el conjunto de datos de prueba. Se aplicaron diferentes umbrales de clasificación, que variaban desde 0.1 hasta 0.9, para determinar el mejor umbral para cada modelo. Este proceso de selección del umbral óptimo fue crucial para garantizar la precisión y la fiabilidad de las predicciones realizadas por cada red neuronal. Posteriormente, se evaluó el rendimiento de cada red neuronal utilizando el mejor umbral de clasificación obtenido en la etapa anterior. Se analizaron métricas como la precisión, la sensibilidad y la especificidad para evaluar la capacidad de cada modelo para detectar caídas con precisión y minimizar los falsos positivos.

## ***Problemas***

Durante el proceso de pruebas, se identificaron varios problemas importantes. La limitación de potencia y memoria del hardware utilizado resultó ser un obstáculo significativo, afectando tanto el proceso de entrenamiento como la evaluación del rendimiento de las redes neuronales. Además, la necesidad de encontrar un equilibrio adecuado entre la complejidad del modelo y los recursos disponibles representó un desafío persistente.

## **Pruebas del Funcionamiento del Dispositivo Wearable**

En esta sección, se presentan los resultados de las pruebas realizadas para evaluar el funcionamiento del dispositivo wearable desarrollado. Durante esta fase, se llevaron a cabo pruebas para verificar todas las características y funcionalidades del dispositivo, incluyendo la visualización de la hora, el funcionamiento táctil, la interfaz de alerta, la detección de caídas, así como la recolección y transmisión de datos al servidor.

### ***Visualización de la Hora y Funciones Básicas***

Se verificó que el dispositivo fuera capaz de conectarse al servidor NTP y mostrarla en pantalla. Se comprobó que la hora se actualizara de forma automática sin necesidad de intervención por parte del usuario y sin la necesidad de estar constantemente conectando al servidor. Además de la visualización de la hora, se comprobó la capacidad del dispositivo para detectar y responder de manera precisa a las pulsaciones y gestos en la pantalla táctil, incluyendo funciones de desplazamiento.



Figura 26: Imagen del dispositivo mostrando la hora

### ***Recopilación de Datos del Sensor***

Se evaluó la capacidad del dispositivo para recopilar datos de los sensores integrados. Se verificó que el dispositivo fuera capaz de leer con precisión los datos provenientes de la IMU, en concreto el acelerómetro y el giroscopio, utilizados para la detección de caídas. Se evaluó la estabilidad y precisión de la lectura de los datos del sensor en diferentes condiciones y posiciones del dispositivo, asegurando así la fiabilidad de la información recopilada para su posterior análisis y procesamiento.

```
13:08:43.885 -> {ACCEL: -0.47,-0.11,-3.18}
13:08:43.885 -> {GYRO: 20.69,-482.00,-1.66}
13:08:43.925 -> {ACCEL: -0.51,-0.08,-3.27}
13:08:43.925 -> {GYRO: 3.81,-376.94,-11.13}
13:08:43.925 -> {ACCEL: -0.47,-0.01,-2.97}
13:08:43.925 -> {GYRO: -40.81,-90.59,-14.06}
13:08:43.963 -> {ACCEL: -0.36,0.07,-2.25}
13:08:43.963 -> {GYRO: -63.16,55.16,-41.50}
13:08:44.009 -> {ACCEL: -0.24,0.11,-1.63}
13:08:44.009 -> {GYRO: -46.22,42.91,-53.06}
```

Figura 27: Captura de las lecturas de datos de los sensores

## ***Funcionamiento de la interfaz de Caídas***

El dispositivo cuenta con una interfaz diseñada para activarse como una alerta en caso de caída. Se llevaron a cabo pruebas exhaustivas para evaluar la interfaz. Se verificó que el botón táctil respondiera correctamente y se probó la funcionalidad para cancelar una alerta de caída en caso de una falsa alarma. Además, se comprobó que el dispositivo iniciara una cuenta atrás de 20 segundos para enviar la alerta de caída al servidor.



Figura 28: Imagen de la Interfaz de Caída

## ***Transmisión de Datos al Servidor y Recepción de Respuestas***

Se realizaron pruebas para verificar la capacidad del dispositivo para recolectar y transmitir los datos obtenidos al servidor. Se evaluó la capacidad del servidor para recibir y procesar los datos enviados por el dispositivo.

## ***Problemas***

Durante las pruebas del dispositivo, se identificaron varios problemas que afectaron en su desarrollo:

Se encontraron dificultades al trabajar con las librerías para crear interfaces gráficas en el dispositivo. La configuración y el uso de estas librerías resultaron ser complicados, y se necesitó de tiempo y considerables pruebas para obtener un resultado visual aceptable.

Se experimentaron dificultades con la detección de la pantalla táctil, especialmente con la librería utilizada para este propósito. En particular, se encontró que algunas funciones de la librería, como la detección de gestos en la pantalla, no cumplían con su cometido y requirieron implementaciones adicionales para su correcto funcionamiento.

## ***Pruebas del Servidor y la Notificación por WhatsApp***

A continuación se presentan los resultados de las pruebas realizadas para evaluar el funcionamiento del servidor desarrollado.

### ***Recepción y Procesamiento de Datos***

Se realizó una prueba de la capacidad del servidor para recibir datos enviados por las dos rutas POST designadas (/predecir y /caida). Se verificó que los datos recibidos en la ruta /predecir pudieran ser utilizados por la Red Neuronal para realizar predicciones de manera efectiva. Además, se confirmó que los datos enviados a la ruta /caida activaran la llamada a la API de CallMeBot para enviar una notificación por WhatsApp en caso de una alerta de caída.

The screenshot shows a POST request to the endpoint `192.168.0.19:5000/predecir`. The request body contains the following data:

```
0.03,-0.61,-0.81,4.28,3.69,-0.06
0.03,-0.61,-0.81,4.09,3.69,0.09
0.03,-0.61,-0.81,4.00,3.69,0.00
0.03,-0.61,-0.81,4.28,3.69,0.03
0.03,-0.61,-0.81,3.94,3.81,0.00
0.03,-0.61,-0.81,3.97,3.81,-0.06
0.03,-0.61,-0.81,4.41,3.72,0.06
0.03,-0.61,-0.81,4.19,3.72,0.03
0.03,-0.61,-0.81,4.00,3.78,0.00
0.03,-0.61,-0.81,4.31,3.66,0.06
```

The response status is **200 (OK)**, time taken is **126 ms**, and size is **0.01 kb**.

Below the request details, the response headers are shown:

```
HTTP/1.1 200 OK
connection: close
content-length: 6
content-type: text/html; charset=utf-8
date: Tue, 30 Apr 2024 11:29:44 GMT
server: Werkzeug/3.0.1 Python/3.10.13
```

The response body is labeled **NORMAL**.

Figura 29: Captura petición POST a “/predecir”

The screenshot shows a web-based application interface for sending a POST request. At the top, there is a URL input field containing "192.168.0.19:5000/caida", followed by a "POST" button, an "EXT" button, and a "Send" button. Below the URL, there are tabs for "Content (1)", "Authorization", "Headers", and "Raw (6)". The "Content (1)" tab is selected, showing a dropdown menu set to "TEXT (text/plain)" with a single line of text: "CAIDA". To the right of the content area are several icons: a share icon, a copy/paste icon, a refresh/circular arrow icon, a left arrow icon, and a file/folder icon. Below the content area, a horizontal line separates it from the response section. The response section displays the status message "Status: 200 (OK) Time: 533 ms Size: 0.07 kb". Underneath this, there are tabs for "Content (1)", "Headers (6)", "Raw (8)", and "HTML". The "Content (1)" tab is selected again, showing the response body: "Mensaje de caída procesado y mensaje de WhatsApp enviado correctamente".

Figura 30: Captura petición POST a “/caida”



Figura 31: Captura Aviso de Caída por WhatsApp

## **Funcionamiento de las Predicciones**

Se llevaron a cabo varias pruebas para confirmar el correcto funcionamiento de las predicciones realizadas por la Red Neuronal. Sin embargo, se encontró que las predicciones no fueron precisas ni consistentes. Se comprobó que las predicciones no se ajustaban adecuadamente a los datos recibidos por el servidor para realizar la detección de caídas de manera efectiva.

```
192.168.0.26 - - [30/Apr/2024 13:16:38] "POST /predecir HTTP/1.1" 200 -
Tiempo transcurrido desde la última petición: 12.753653287887573 segundos
Forma de los datos: (500, 6)
1/1 [=====] - 0s 80ms/step
[[[ 0.02 -0.61 -0.81  3.72  3.87  0.03]
 [ 0.02 -0.61 -0.81  3.69  3.84  0.   ]
 [ 0.02 -0.61 -0.81  4.12  3.81  0.09]
 ...
 [ 0.02 -0.61 -0.81  3.69  3.84  0.03]
 [ 0.02 -0.61 -0.81  4.06  3.78  0.03]
 [ 0.02 -0.61 -0.81  3.34  3.91 -0.06]]]
[[0.00285515]]
Normal
```

Figura 32: Captura de una predicción de la RN en el Servidor

## Problemas

Durante las pruebas del servidor, se identificó un problema significativo relacionado con el funcionamiento de las predicciones. El principal problema encontrado fue la disparidad entre la configuración de los sensores y el tratamiento de los datos en el dataset de entrenamiento y la configuración de los sensores y el tratamiento de los datos en el dispositivo real. Esta discrepancia dificultó la capacidad del dispositivo para detectar caídas de manera precisa y consistente.

## Discusión de los Resultados

Las pruebas de las redes neuronales entrenadas para la detección de caídas revelaron varios hallazgos significativos. A pesar de haber entrenado múltiples modelos y haber realizado ajustes en los umbrales de clasificación, se observó una falta de precisión y consistencia en las predicciones. La principal limitación encontrada fue la discrepancia entre la configuración de los sensores y el tratamiento de los datos en el proceso de entrenamiento y en el dispositivo real. Esto plantea la necesidad de una mayor atención en la recolección y preparación de datos, así como en la adaptación de los modelos a las condiciones del entorno real.

Por otro lado, las pruebas del dispositivo wearable mostraron avances significativos en cuanto a la funcionalidad básica, como la visualización de la hora y la

detección de la pantalla táctil. Sin embargo, se encontraron desafíos importantes relacionados con la interfaz gráfica y la detección precisa de la pantalla táctil. Estos problemas afectaron la usabilidad y confiabilidad del dispositivo, lo que resalta la importancia de mejorar el diseño y la implementación de la interfaz de usuario en futuras iteraciones del proyecto.

En cuanto a las pruebas del servidor, demostraron una capacidad adecuada para recibir y procesar datos enviados por el dispositivo. Sin embargo, sería beneficioso realizar pruebas adicionales para evaluar el rendimiento del servidor bajo conexiones simultáneas, así como considerar la actualización del medio de conectividad WiFi a una SIM con conectividad 4G para mejorar la disponibilidad y estabilidad del servicio en futuras implementaciones.

# Análisis temporal

El análisis temporal revela desviaciones significativas entre la planificación prevista y el tiempo real dedicado a cada tarea y fase. A continuación, se detallan estas desviaciones y las razones detrás de ellas:

## Fase 1 - Investigación y Planificación

### ***Tarea 1.1 - Definición del Alcance***

El objetivo inicial de esta fase era identificar y definir los objetivos específicos y metas a alcanzar durante el proyecto en el período del 16 de octubre al 22 de octubre. Aunque se logró cumplir con el objetivo de duración establecido, se observó una desviación temporal debido al inicio tardío de la tarea, que comenzó el 30 de octubre. Esta demora inicial no impactó en la cronología general del proyecto, pero sí, en la de esta fase, afectando así el desarrollo de las tareas siguientes.

### ***Tarea 1.2 - Investigación de Tecnologías***

Con el objetivo de investigar tecnologías relevantes para redes neuronales y dispositivos wearables, se planificó la ejecución de esta tarea del 23 de octubre al 5 de noviembre. A pesar de haber alcanzado el objetivo deseado, la demora en la tarea precedente influyó en el inicio de esta etapa, que también se retrasó hasta el 30 de octubre. Sin embargo, se implementaron medidas para llevar a cabo esta tarea en paralelo con la Tarea 1.1, permitiendo un progreso continuo en el proyecto.

### ***Tarea 1.3 - Estudio del Estado del Arte***

Durante esta fase, se llevó a cabo un estudio bastante profundo del estado del arte en el campo de detección de caídas y tecnologías asociadas, con una duración

planificada del 6 de noviembre al 26 de noviembre. A pesar de las demoras en las tareas anteriores, esta actividad pudo comenzar según lo previsto y se logró completar satisfactoriamente la tarea en la duración prevista y gracias a la ejecución simultánea con la Tarea 1.2, lo que permitió avanzar en la investigación de manera consistente.

## **Fase 2 - Desarrollo de Redes Neuronales**

### ***Tarea 2.1 - Pruebas de Concepto***

Durante esta etapa, se llevaron a cabo pruebas de concepto de diferentes arquitecturas de redes neuronales para la detección de caídas. La duración prevista era de 3 semanas, del 15 de enero al 4 de febrero. Sin embargo, debido a la complejidad de las pruebas y la necesidad de experimentar con múltiples modelos y conjuntos de datos, esta tarea se extendió una semana adicional, finalizando el 11 de febrero. Este retraso se justifica por la naturaleza iterativa de la experimentación y la búsqueda de la arquitectura óptima para el proyecto.

### ***Tarea 2.2 - Entrenamiento de Redes Neuronales***

El objetivo de esta tarea era el entrenamiento de las redes neuronales seleccionadas, con una duración planificada de 4 semanas, del 5 de febrero al 3 de marzo. Sin embargo, debido al retraso acumulado de la tarea anterior, esta etapa se inició una semana más tarde. Además, la complejidad del proceso de entrenamiento, que requería largas sesiones y ajustes continuos, prolongó la duración de la tarea en 2 semanas adicionales. La necesidad de realizar la Tarea 2.3 en paralelo, para evaluar y ajustar las redes entrenadas, también contribuyó a este retraso.

### ***Tarea 2.3 - Evaluación y Ajustes***

Durante esta etapa, se evaluó el rendimiento de las redes neuronales entrenadas y se realizaron los ajustes necesarios para mejorar su eficacia. La duración

planificada era de 3 semanas, del 4 de marzo al 24 de marzo. Finalmente, esta tarea se llevó a cabo en paralelo con la Tarea 2.2 debido a la necesidad de realizar ajustes continuos durante el proceso de entrenamiento. A pesar de la ejecución simultánea, se logró completar esta etapa dentro del plazo previsto gracias a una gestión eficiente del tiempo y recursos.

### ***Hito 1 - Desarrollo y Optimización de la Red Neuronal***

A pesar de que la Fase 2 acumuló un retraso de 3 semanas en su ejecución debido a la complejidad y extensión de las tareas, se logró concluir en la fecha prevista gracias a la ejecución de tareas en paralelo. Este enfoque permitió maximizar la eficiencia y optimizar el tiempo disponible, asegurando así el cumplimiento de los hitos y objetivos establecidos dentro del cronograma establecido para el proyecto.

Al finalizar la Fase 2 del proyecto, se alcanzó el Hito 1, que corresponde al objetivo principal de desarrollar y optimizar la red neuronal para la detección de caídas en el dispositivo wearable. Este hito marca un avance significativo en el progreso del proyecto hacia su objetivo final.

## **Fase 3 - Desarrollo del Dispositivo Wearable**

### ***Tarea 3.1 - Selección del Hardware***

Durante esta etapa, se llevó a cabo la selección del hardware del dispositivo wearable, con una duración prevista de 2 semanas, del 25 de marzo al 7 de abril. Sin embargo, debido a la necesidad de adquirir sensores y una placa de desarrollo específica, la tarea se prolongó una semana adicional. Este retraso se debió a la gestión de compras y tiempos de espera asociada con la adquisición de los componentes necesarios.

## **Tarea 3.2 - Desarrollo del Software**

El objetivo de esta tarea era desarrollar el software para la recopilación de datos del sensor, con una duración planificada de 3 semanas, del 8 de abril al 28 de abril. Aunque la tarea se inició una semana más tarde debido a los retrasos acumulados de la tarea anterior, se logró completar en solo 2 semanas, gracias a una eficiente asignación de recursos y esfuerzos. Esta ejecución anticipada permitió mantener el proyecto en línea con el cronograma establecido.

## **Tarea 3.3 - Integración de la Red Neuronal**

Durante esta etapa, se llevó a cabo la integración de la red neuronal en el dispositivo, con una duración planificada de 2 semanas, del 15 de abril al 28 de abril. Sin embargo, la tarea se adelantó al 1 de marzo debido a los tiempos de espera asociados a la adquisición de los componentes necesarios. Este adelanto permitió realizar la tarea en paralelo con la Tarea 3.1, manteniendo así el proyecto en línea con el cronograma establecido. A pesar de haberse adelantado, la tarea se completó en el tiempo previsto.

## **Hito 2 - Prototipo Hardware y Realización de Pruebas**

A pesar de enfrentar un retraso de 1 semana en la ejecución de la Tarea 3.1 debido a la necesidad de adquirir componentes específicos, se logró mantener el proyecto dentro del cronograma previsto. Esto se debió al acortamiento de la duración de la Tarea 3.2, que se completó en 2 semanas en lugar de las 3 inicialmente planificadas. Además, el adelantamiento de la fecha de inicio de la Tarea 3.3 permitió que se llevara a cabo en paralelo con la Tarea 3.1. Gracias a esta ejecución simultánea y a una gestión eficiente del tiempo, se logró realizar la Fase 3 en la duración estimada a pesar del retraso inicial.

Al concluir la Fase 3, se alcanzó el Hito 2, correspondiente al Objetivo 3 del proyecto. Este hito marca un avance significativo en el desarrollo del dispositivo wearable, al lograr la integración exitosa de la red neuronal y la realización de pruebas funcionales del prototipo.

## **Fase 4 - Pruebas y Validación**

### ***Tarea 4.1 - Pruebas de Redes Neuronales***

Durante esta etapa, se llevaron a cabo pruebas de las redes neuronales implementadas, con una duración prevista de 2 semanas, del 11 de marzo al 24 de marzo. La tarea se realizó de forma paralela a las Tareas 2.2 y 2.3, manteniendo así el tiempo estimado y asegurando el avance continuo del proyecto.

### ***Tarea 4.2 - Pruebas del Dispositivo***

El objetivo de esta tarea era realizar pruebas de precisión y fiabilidad del dispositivo en la detección de caídas, con una duración planificada de 4 semanas, del 29 de abril al 26 de mayo. Sin embargo, gracias a la realización conjunta de las pruebas con el desarrollo del dispositivo, estas se completaron en 1 semana, del 22 de abril al 28 de abril. Este enfoque de desarrollo y pruebas simultáneas permitió optimizar el tiempo y asegurar un avance eficiente del proyecto.

### ***Tarea 4.3 - Validación en Entornos Reales***

Durante esta etapa, se validó la solución en entornos del mundo real, con una duración planificada de 3 semanas, del 6 de mayo al 26 de mayo. Similar a la Tarea 4.2, las pruebas se llevaron a cabo en conjunto con el desarrollo, lo que permitió completar la validación en 1 semana en lugar de las 3 semanas previstas originalmente. Este enfoque de desarrollo y validación simultáneas garantizó una integración fluida del dispositivo en entornos reales y una respuesta ágil a los posibles problemas identificados durante las pruebas.

## ***Hito 3 - Pruebas de Comunicación y Alerta***

Las fechas de inicio de las tareas 4.2 y 4.3 se adelantaron significativamente, lo que resultó en una reducción considerable de los tiempos de ejecución. Ambas tareas, que originalmente estaban programadas para concluir el 24 de mayo, se completaron el 28 de abril, 4 semanas antes de lo previsto. Este adelanto en la ejecución de las pruebas proporcionó un amplio margen de tiempo adicional, brindándonos una valiosa flexibilidad y tranquilidad para la fase final del proyecto y su entrega.

Al concluir la Fase 4 del proyecto, se alcanzó el Hito 3, asociado al Objetivo 2 del proyecto. Este hito representa un paso importante hacia la finalización del proyecto, al validar la comunicación y las capacidades de alerta del dispositivo en entornos reales.

## **Fase 5 - Documentación y Entrega**

### ***Tarea 5.1 - Documentación Técnica***

Esta tarea, destinada a documentar detalladamente cada etapa del proyecto, se llevó a cabo de manera incremental a lo largo del desarrollo del proyecto. Aunque la duración planificada fue de 2 semanas, del 13 de mayo al 26 de mayo, se estima que se dedicaron aproximadamente 4 semanas, una por cada fase del proyecto. Esta documentación se realizó en paralelo a las actividades de cada fase, asegurando así una cobertura exhaustiva de todo el proceso.

### ***Tarea 5.2 - Manuales de Usuario***

El objetivo de esta tarea era crear manuales de usuario para el dispositivo y el sistema de detección de caídas. Aunque la duración planificada fue de 2 semanas, del 13 de mayo al 26 de mayo, la tarea se completó en 1 semana, comenzando el 29 de abril. Esta tarea se realizó una vez finalizado todo el proyecto, proporcionando una

guía detallada para comprender el funcionamiento del dispositivo y replicar el trabajo realizado.

### ***Tarea 5.3 - Entrega Final del Proyecto***

La entrega final del proyecto, que incluía todos los componentes y documentación asociada, se adelantó en 2 semanas y media. Aunque la duración planificada fue de 1 semana, del 23 de mayo al 29 de mayo, se inició el 6 de mayo para asegurar la puntualidad en la entrega. Este adelanto en la fecha de entrega proporcionó margen adicional para ajustes finales y garantizó una finalización exitosa del proyecto.

A pesar de los desafíos y desviaciones temporales encontrados a lo largo del proyecto, la implementación de estrategias de gestión del tiempo, ejecución en paralelo y adaptaciones continuas permitieron mantener el proyecto dentro de los plazos establecidos y lograr sus objetivos con éxito. Desde la fase inicial de investigación y planificación hasta la documentación final y la entrega del proyecto, se demostró una capacidad constante para superar obstáculos y ajustarse a las circunstancias cambiantes.

Es notable destacar que el proyecto se completó antes de lo previsto. A pesar de que se planificó inicialmente un período de 25 semanas y media, se logró concluir en tan solo 22 semanas. Este logro refleja no solo la eficiencia en la ejecución de tareas, sino también la dedicación y el compromiso del equipo en alcanzar los objetivos establecidos.

Además, es importante mencionar que la determinación del tiempo dedicado es difícil de cuantificar debido a las variaciones en la disponibilidad de tiempo. En algunos días, circunstancias externas limitaron la cantidad de tiempo que se podía dedicar al proyecto, mientras que en otros, se dedicaba el día completo al trabajo. Sin embargo, se podría decir que se dedicaron aproximadamente 15 horas semanales al proyecto, lo que equivale a unas 3 horas diarias de lunes a viernes. En total, se estima que se dedicaron alrededor de 330 horas al proyecto.

Este nuevo diagrama refleja con precisión el tiempo real dedicado a cada etapa del proyecto, proporcionando una visión clara de las fechas de inicio y finalización de cada tarea. A pesar de los desafíos y desviaciones encontrados a lo largo del proyecto, el equipo logró mantenerse dentro de los plazos establecidos y cumplir con los objetivos con éxito.

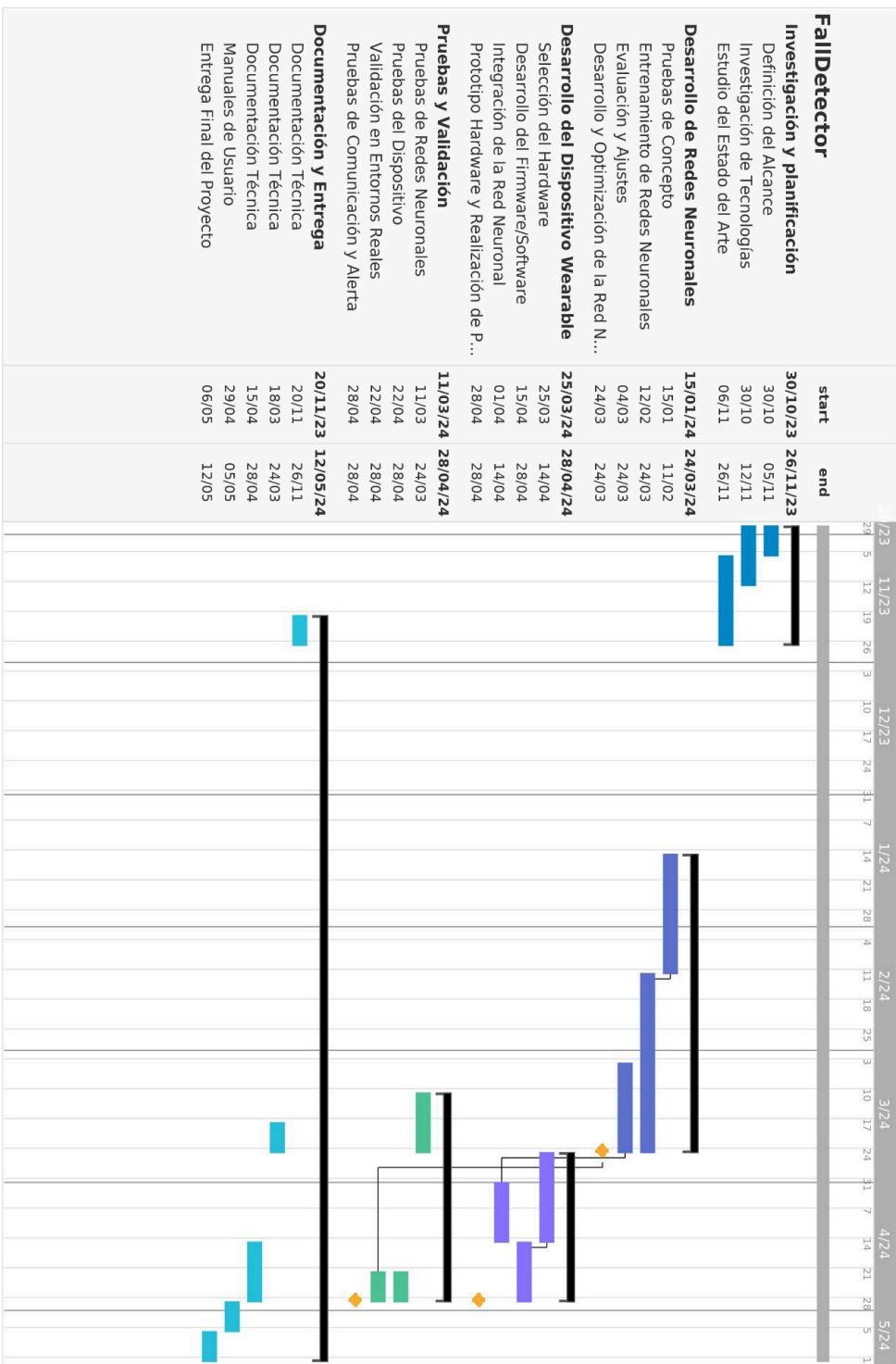


Figura 33: Diagrama de Gantt Real

# Análisis económico

## Costes del Desarrollo del Proyecto

### ***Coste de horas de trabajo***

Durante la ejecución del proyecto, se dedicaron aproximadamente 300 horas de trabajo. Estas horas incluyen investigación, desarrollo, pruebas y documentación del proyecto.

Si consideramos un salario promedio para un ingeniero informático junior, que suele rondar los 24,000 € anuales brutos, equivalente a unos 1,700 € netos al mes, podemos calcular el coste del tiempo dedicado al proyecto.

Suponiendo una jornada laboral estándar de 8 horas al día y 20 días laborables al mes, esto representa unas 160 horas de trabajo al mes. Por tanto, las 300 horas totales del proyecto equivaldrían aproximadamente a dos meses de trabajo dedicado. Considerando este período de dos meses de trabajo a jornada completa, el coste total del tiempo invertido sería de alrededor de 3,400 €.

### ***Coste de Entrenamiento de la Red Neuronal***

Para el entrenamiento de la red neuronal, se utilizó una GPU NVIDIA GTX 1060 6GB, que consume alrededor de 250W a pleno rendimiento. El tiempo total dedicado al entrenamiento, incluyendo pruebas previas y ajustes de hiperparámetros, fue de aproximadamente 40 horas. Considerando un precio medio de electricidad de 0,15 €/kWh, el coste total de electricidad para el entrenamiento de la red sería de aproximadamente 1,5 €. Este coste representa una fracción mínima del gasto total del proyecto, pero es importante considerarlo para obtener una visión completa de los costes asociados.

## **Coste de Componentes del Prototipo**

Se utilizó una serie de componentes electrónicos para la construcción y pruebas del prototipo, incluyendo microcontroladores y sensores.

El desglose detallado de los costes de los componentes puede verse en la siguiente tabla:

| Componentes               | Precio |
|---------------------------|--------|
| Arduino Nano 33 BLE Sense | 45€    |
| ESP32                     | 5€     |
| IMU Gy-521                | 2€     |
| Sensor MPU-9150           | 17€    |
| IMU Gy-511                | 8€     |
| ESP32-S3-Touch-LCD-1.28   | 23€    |
|                           | 100€   |

Tabla 5: Desglose Precio Componentes

El coste total de los componentes utilizados en el prototipo asciende a 100 €. Estos componentes han sido esenciales para la implementación y las pruebas del dispositivo y representan una parte significativa de los costes directos del proyecto.

## **Coste del Producto Final**

Es importante destacar que, si bien se adquirieron varios componentes para el prototipo y pruebas, en la implementación final del dispositivo solo se utilizó el ESP32-S3-Touch-LCD-1.28, con un coste de 23 €. Por lo tanto, el coste del producto final se reduce a 23 €, que representa el gasto directo asociado con el hardware utilizado en la solución final.

## **Viabilidad**

El análisis de la viabilidad económica del proyecto revela importantes oportunidades para maximizar el retorno de la inversión y garantizar la sostenibilidad financiera del mismo. A pesar de que el gasto inicial del proyecto fue de 100€, el coste real del prototipo se sitúa en aproximadamente 23€. Esta brecha entre el presupuesto y el coste real del producto final abre la puerta a estrategias adicionales para mejorar la rentabilidad del proyecto.

## **Reducción de costes de producción**

A través de la optimización de procesos y la búsqueda de alternativas de fabricación más eficientes, se pueden reducir aún más los costes de producción del dispositivo. Una opción sería externalizar la fabricación de las placas del dispositivo a un proveedor especializado, lo que podría resultar en ahorros significativos en los costes de producción. Además, se podría negociar con los proveedores para obtener precios más competitivos al comprar los componentes al por mayor, lo que también contribuiría a reducir los costes de producción.

## **Establecimiento de un precio competitivo**

Considerando el coste real del prototipo y los potenciales ahorros en los costes de producción, se propone establecer un precio de venta al público que sea competitivo en el mercado. Basándonos en estos cálculos, un precio ajustado para el dispositivo podría rondar los 50€. Este enfoque no solo garantizaría márgenes de beneficio saludables, sino que también permitiría posicionar el producto de manera atractiva frente a la competencia, lo que aumentaría las posibilidades de éxito en el mercado.

## Análisis del retorno de la inversión

Con un coste de producción reducido y un precio de venta competitivo, se espera que el proyecto genere un retorno de la inversión significativo. Si consideramos un coste total de producción de 23€ y un precio de venta de 50€, el beneficio por unidad sería de más del 50%, lo que representa una oportunidad lucrativa para los inversores.

## Aplicabilidad

El potencial comercialización del dispositivo y la eventual creación de una empresa dedicada a su desarrollo y venta implicaría la consideración de diversos factores adicionales, tanto a nivel operativo como financiero.

## Consideraciones de Personal

Especialistas en Ingeniería Informática: Se requeriría al menos un especialista en redes neuronales para la mejora continua del funcionamiento de la tecnología de detección de caídas y otro ingeniero informático para el desarrollo y mejora del software del dispositivo.

Diseñador UX/UI: Un profesional especializado en diseño de experiencia de usuario (UX) y diseño de interfaz de usuario (UI) sería fundamental para crear interfaces intuitivas y atractivas para el dispositivo.

Servidores y Soporte Técnico: Sería necesario contar con infraestructura de servidor para alojar la red neuronal y brindar soporte técnico a los usuarios.

Equipo de Comercio y Marketing: Un equipo dedicado a las estrategias de comercio y marketing sería esencial para promover y posicionar el producto en el mercado.

## **Mejoras Tecnológicas y Modelo de Negocio**

**Conectividad 4G:** La modificación de la conectividad del dispositivo de Wi-Fi a 4G con una tarjeta SIM permitiría a los usuarios utilizar el dispositivo fuera de sus hogares sin depender de una conexión Wi-Fi.

**Modelo de Membresía:** Se podría implementar un modelo de negocio basado en una membresía mensual para cubrir los costes asociados con la conectividad 4G y los servicios de alerta de caídas, lo que proporcionaría una fuente de ingresos recurrentes.

## **Proyecciones Financieras**

**Gastos Mensuales:** Considerando los gastos de personal y otros costes operativos, se estima un gasto mensual total de 10000€.

**Ingresos Potenciales:** Con un precio de venta del dispositivo de 50€ y una ganancia de 25€ por unidad, así como una membresía mensual de 15€ por usuario, se proyecta un beneficio significativo de más de 200€ anuales por dispositivo.

**Rentabilidad:** Para alcanzar la rentabilidad, se estima que sería necesario vender aproximadamente 1000 dispositivos y mantener una base de usuarios suscritos a las membresías mensuales. Solamente en membresías, se obtendría 15000€/mes, una cantidad superior al gasto mensual en personal.

La comercialización del dispositivo y la creación de una empresa dedicada a su desarrollo y venta ofrecen oportunidades de crecimiento y rentabilidad. Sin embargo, es crucial realizar un análisis exhaustivo del mercado y desarrollar estrategias sólidas para alcanzar el éxito empresarial a largo plazo.

# Conclusiones

En este trabajo se ha abordado el desarrollo de un sistema de detección de caídas utilizando redes neuronales y dispositivos wearables. A través de la realización de pruebas tanto en las redes neuronales como en el dispositivo wearable y el servidor, se han obtenido una serie de conclusiones que resumen el trabajo realizado y proponen posibles mejoras para futuras investigaciones.

El desarrollo y las pruebas de las redes neuronales para la detección de caídas han destacado la importancia de una cuidadosa recolección y preparación de datos, así como la adaptación de los modelos a las condiciones del entorno real. A pesar de los esfuerzos realizados, las dificultades encontradas en el proceso de entrenamiento y las limitaciones de precisión y consistencia en las predicciones subrayan la necesidad de investigaciones adicionales para mejorar la eficacia del sistema.

En cuanto al dispositivo wearable, si bien se han logrado avances significativos en la funcionalidad básica, como la visualización, la detección de la pantalla táctil y la recolección de datos de los sensores, se han identificado desafíos importantes en la interfaz gráfica y la detección precisa de la pantalla táctil. Estos problemas afectan la usabilidad y confiabilidad del dispositivo, destacando la importancia de mejoras en el diseño y la implementación de la interfaz de usuario para futuras iteraciones del proyecto.

Por último, las pruebas del servidor han demostrado una capacidad adecuada para recibir y procesar datos enviados por el dispositivo, pero se han identificado dificultades en el funcionamiento de las predicciones. Además, se señala la necesidad de realizar pruebas adicionales para evaluar el rendimiento del servidor bajo conexiones simultáneas y considerar la actualización del medio de conectividad para mejorar la disponibilidad y estabilidad del servicio.

En resumen, este trabajo ha representado un avance significativo en el desarrollo de un sistema de detección de caídas. Se ha identificado la necesidad de abordar áreas clave para mejorar la fiabilidad y la eficacia del sistema. En particular, se destaca la importancia de la creación de un dataset propio utilizando los mismos sensores y dispositivo que se utilizarán en el sistema final. Esta iniciativa mejoraría la

precisión de las predicciones, las cuales son fundamentales para lograr un sistema de detección de caídas más preciso y confiable en entornos reales.

# Bibliografía

- [1] OMS (26 de abril de 2021). "Caídas" [En línea]. Disponible en:  
<https://www.who.int/es/news-room/fact-sheets/detail/falls>
- [2] Yu, X. Approaches and principles of fall detection for elderly and patient. In Proceedings of the HealthCom 2008—10th International Conference on e-Health Networking, Applications and Services, Singapore, 7–9 July 2008; pp. 42–47.
- [3] Forbes (13 febrero de 2024). "The Best Fall Detection Devices of 2024." [En línea]. Disponible en:  
<https://www.forbes.com/health/medical-alert-systems/fall-detection-devices/>
- [4] D. Droghini, E. Principi, S. Squartini, P. Olivetti, and F. Piazza, "Human fall detection by using an innovative floor acoustic sensor," *Smart Innov.*, vol. 69, pp. 97–107, Aug. 2017
- [5] K. de Miguel, A. Brunete, M. Hernando, and E. Gambao, "Home camerabased fall detection system for the elderly," *Sensors*, vol. 17, no. 12, p. 2864, Dec. 2017.
- [6] Ren, L., Shi, W., Yu, Z., & Cao, J. (2015). ALARM: A novel fall detection algorithm based on personalized threshold. In 2015 17th International Conference on E-health Networking, Application & Services (HealthCom) (pp. 410-415).
- [7] Fudickar, S. J. F., Lindemann, A., & Schnor, B. (2014). Threshold-based Fall Detection on Smart Phones. In International Conference on Health Informatics.
- [8] Dumitache, M., & Pasca, S. (2013). Fall detection algorithm based on triaxial accelerometer data. In 2013 E-Health and Bioengineering Conference (EHB) (pp. 1-4).
- [9] Shi, T., Sun, X., Xia, Z., Chen, L., & Liu, J. (2016). Fall Detection Algorithm Based on Triaxial Accelerometer and Magnetometer.
- [10] Ye, B., & Yu, L. (2019). Fall detection system based on BiLSTM neural network. *Transactions on Machine Learning and Artificial Intelligence*.
- [11] Zhang, Qingbin, Guohui Tian, Nana Ding and Yanru Zhang. "A fall detection study based on neural network algorithm using AHRS." 2013 IEEE International Conference on Information and Automation (ICIA) (2013): 773-779.
- [12] Huang, Chieh-Ling, E-Liang Chen and P. C. Chung. "FALL DETECTION USING MODULAR NEURAL NETWORKS WITH BACK-PROJECTED OPTICAL FLOW." *Biomedical Engineering: Applications, Basis and Communications* 19 (2007): 415-424.
- [13] Salleh, Salihatun Md., Ainul Husna Mohd Yusoff, Khairul Nizam Ngadimon and Cheng Zhi Koh. "Neural Network Algorithm-based Fall Detection Modelling." *International Journal of Integrated Engineering* 12 (2020): 138-150.

- [14] J, Sree, Umamakeswari A, & Jenita B. (2015). A survey on technical approaches in fall detection systems. National Journal of Physiology, Pharmacy, and Pharmacology, 5, 275-279.
- [15] Sixsmith, A., & Johnson, N. (2004). A smart sensor for detecting falls in the elderly. IEEE Pervasive Computing, 3(2), 42–47.
- [16] Liu, L., Popescu, M., Ho, K., Skubic, M., & Rantz, M. (2012). Doppler radar sensor placement in a fall detection system. In Conference Proceedings IEEE Engineering in Medicine and Biology Society (pp. 256–259).
- [17] Thome N, Miguet S, Ambellouis S. (2008). A real-time, multiview fall detection system: A LHMM-based approach. IEEE Transactions on Circuits and Systems for Video Technology, 18(11), 1522–1532.
- [18] Ojetola, Olukunle, Gaura, Elena, & Brusey, James. (2015). Data set for fall events and daily activities from inertial sensors. In Proceedings of the 6th ACM Multimedia Systems Conference, MMSys 2015. doi:10.1145/2713168.2713198.

# Anexos

## Anexo 1 - Manual de Configuración del Entorno Virtual

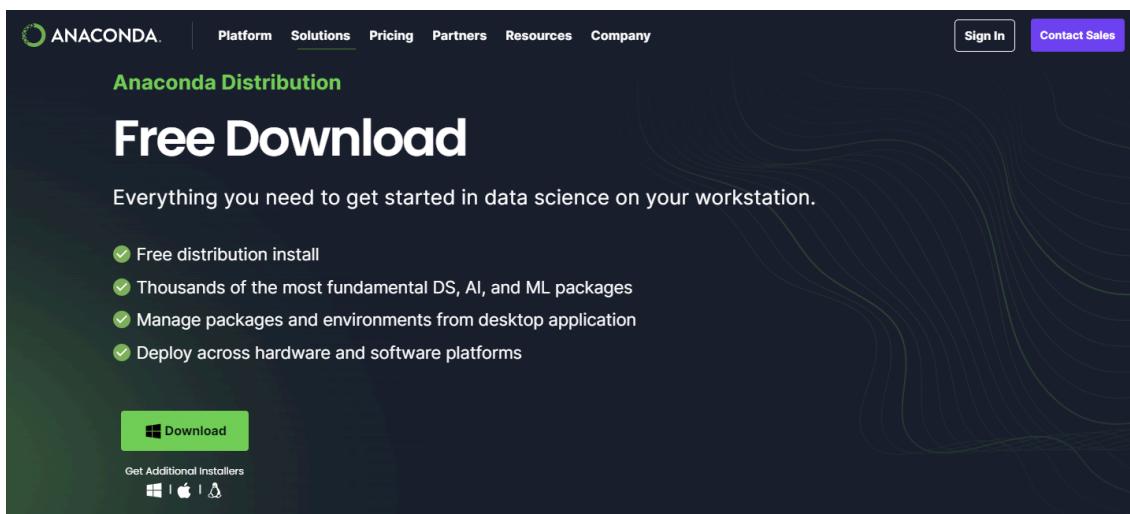
En este anexo se presenta un manual detallado que describe el proceso de configuración de Anaconda y su entorno virtual para la creación de un ambiente independiente compatible con Python 3.10.13 y TensorFlow 2.10.0. Este manual proporciona instrucciones paso a paso, acompañadas de capturas de pantalla, para guiar al usuario a través del proceso de instalación y configuración.

### *Instalación de Anaconda*

#### Paso 1: Descarga de Anaconda

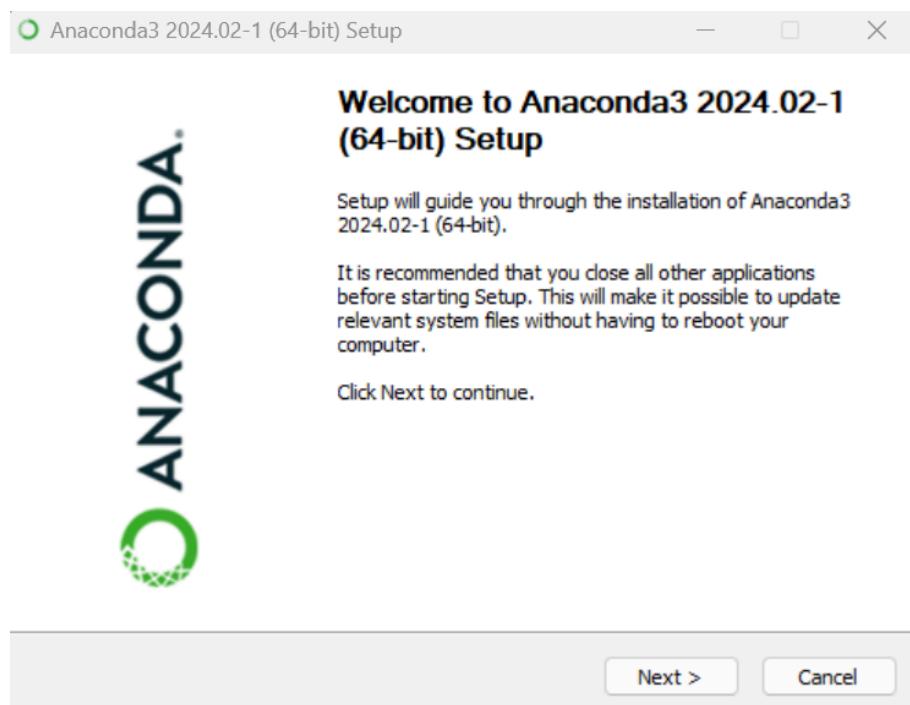
Visite el sitio web oficial de Anaconda: <https://www.anaconda.com/download>

Descargue la versión adecuada de Anaconda para su sistema operativo (Windows, macOS, Linux).

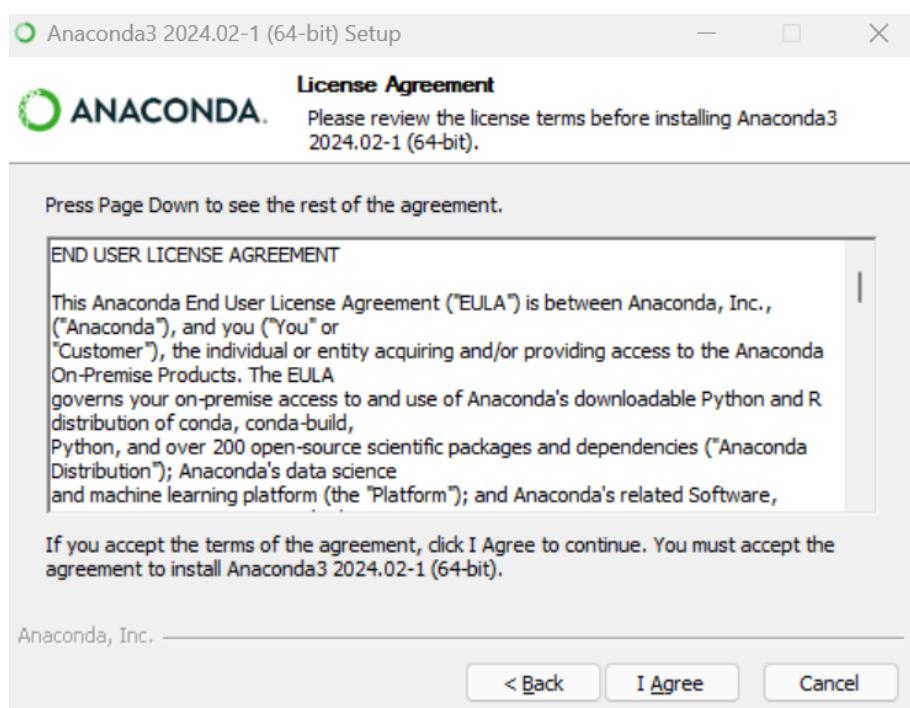


## Paso 2: Instalación de Anaconda

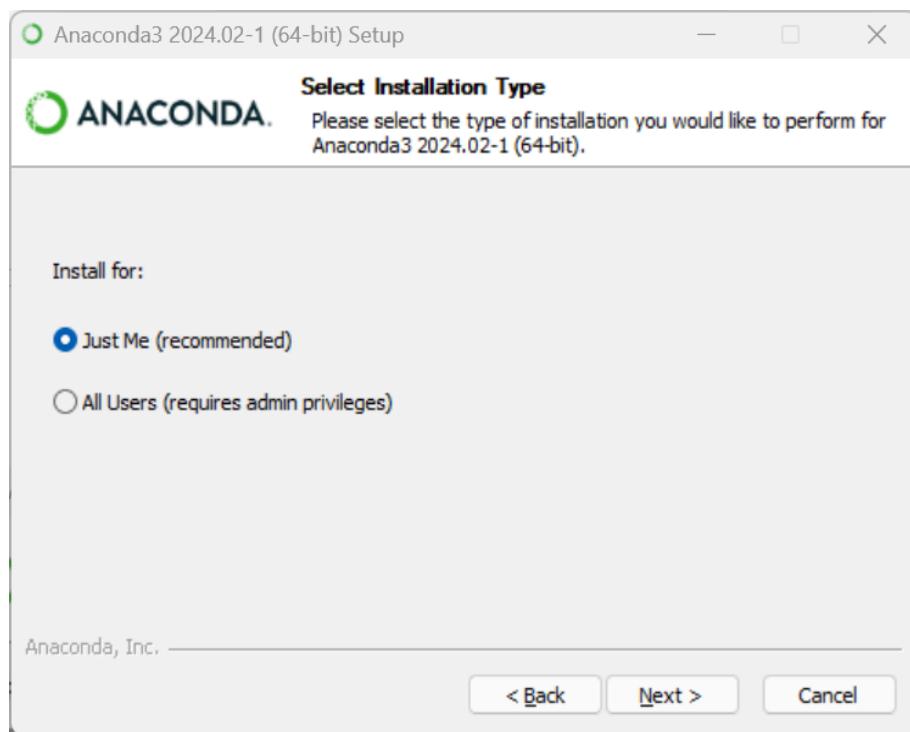
Ejecute el instalador descargado y siga las instrucciones del asistente de instalación.



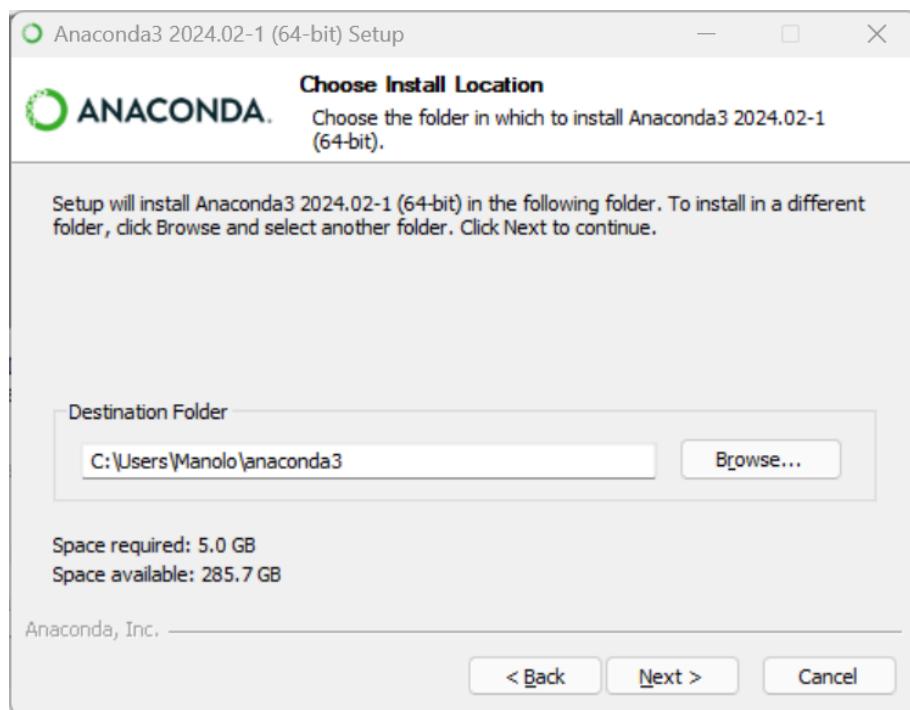
Acepte los términos y condiciones de la licencia.



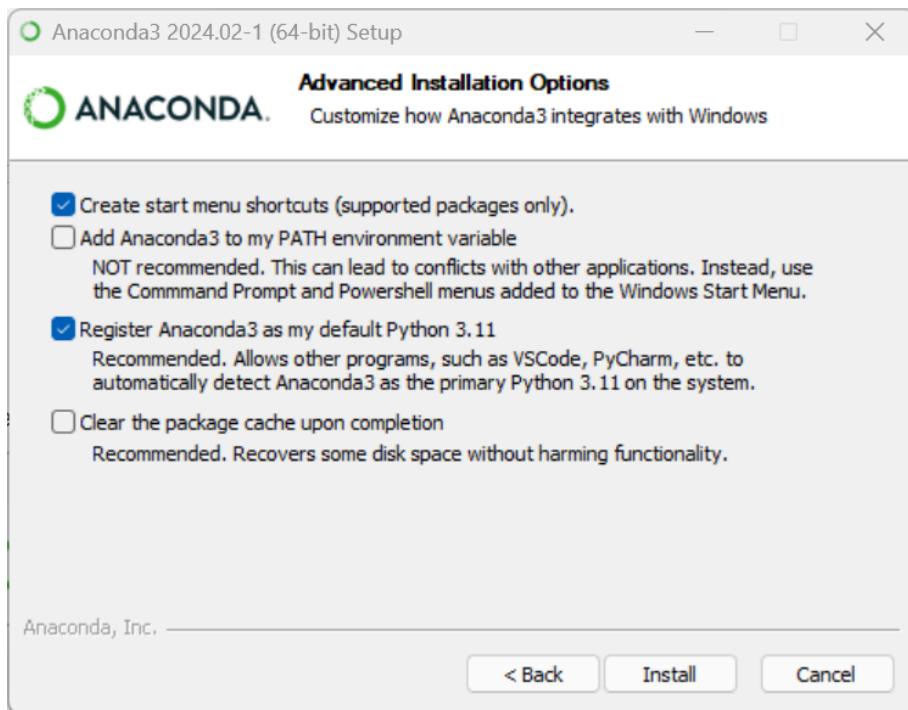
Elija la opción "Instalar solo para mí" cuando se le solicite.



Seleccione la ubicación de instalación y haga clic en "Instalar".



Elija si desea agregar Anaconda a su variable de entorno PATH o registrar Anaconda como su Python predeterminado. (No es lo recomendado)



Haga clic en Instalar. Después de una instalación exitosa, verá el cuadro de diálogo "Gracias por instalar Anaconda".

## ***Creación del Entorno Virtual***

### **Paso 1: Abrir Anaconda PowerShell Prompt**

Busque y abra Anaconda Powershell Prompt desde el menú de inicio o la barra de aplicaciones.



## Paso 2: Crear un Nuevo Entorno Virtual

Para crear un nuevo entorno, solamente necesitamos ejecutar el siguiente comando:

- `conda create --name [Nombre del Entorno] python=3.10.13`

## Paso 3: Activar el Entorno Virtual

Una vez creado, debemos de activar el entorno, para ello usaremos el siguiente comando:

- `conda activate [Nombre del Entorno]`

## ***Configuración de Dependencias***

### Paso 1: Instalación de TensorFlow

Desde el mismo terminal en el que nos encontrábamos en el paso anterior, escribiremos lo siguiente:

- `conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0`

Al terminar su instalación debe verse en el terminal algo así.

The screenshot shows a terminal window titled "Anaconda Powershell Prompt". The output of the command is as follows:

```
vs2015_runtime-14.38.33130 | hcb4865c_18 17 KB conda-forge
Total: 1.23 GB

The following NEW packages will be INSTALLED:
cudatoolkit      conda-forge/win-64::cudatoolkit-11.2.2-h7d7167e_13
cudnn           conda-forge/win-64::cudnn-8.1.0.77-h3e0f4f4_0
ucrt            conda-forge/win-64::ucrt-10.0.22621.0-h57928b3_0
vc14_runtime     conda-forge/win-64::vc14_runtime-14.38.33130-h82b7239_18

The following packages will be UPDATED:
openssl          pkgs/main::openssl-3.0.13-h2bbff1b_0 --> conda-forge::openssl-3.2.1-hcfcfb64_1
vs2015_runtime    pkgs/main::vs2015_runtime-14.27.29016~ --> conda-forge::vs2015_runtime-14.38.33130-hcb4865c_18

Proceed ([y]/n)? y

Downloading and Extracting Packages:
Preparing transaction: done
Verifying transaction: done
Executing transaction: - "By downloading and using the CUDA Toolkit conda packages, you accept the terms and conditions
- "By downloading and using the cuDNN conda packages, you accept the terms and conditions of the NVIDIA cuDNN EULA - ht
done
(FallDetectorEnv) PS C:\Users\TheNo\Desktop\TFG>
```

Luego instalamos TensorFlow forzando que la versión sea anterior a la 2.11 por problemas de compatibilidad en Windows

- `python -m pip install "tensorflow<2.11"`

Podemos comprobar que la instalación ha sido exitosa si al ejecutar el siguiente comando nos detecta la GPU de nuestro sistema

- `python -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"`

La respuesta debe ser similar a:

- `[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]`

## Paso 2: Instalación de Otras Dependencias

Aún en el terminal, instalaremos el resto de dependencias necesarias como TensorBoard, NumPy, Pandas y scikit-learn utilizando el siguiente comando:

- pip install tensorboard numpy pandas scikit-learn

Usando el comando pip list podremos ver una lista de todas las dependencias que tenemos instaladas, en mi caso son las siguientes:

| Package                      | Version     |
|------------------------------|-------------|
| absl-py                      | 2.1.0       |
| astunparse                   | 1.6.3       |
| cachetools                   | 5.3.3       |
| certifi                      | 2024.2.2    |
| charset-normalizer           | 3.3.2       |
| contourpy                    | 1.2.0       |
| cycler                       | 0.12.1      |
| flatbuffers                  | 24.3.7      |
| fonttools                    | 4.49.0      |
| gast                         | 0.4.0       |
| google-auth                  | 2.28.1      |
| google-auth-oauthlib         | 0.4.6       |
| google-pasta                 | 0.2.0       |
| grpcio                       | 1.62.0      |
| h5py                         | 3.10.0      |
| idna                         | 3.6         |
| joblib                       | 1.3.2       |
| keras                        | 2.10.0      |
| Keras-Preprocessing          | 1.1.2       |
| kiwisolver                   | 1.4.5       |
| libclang                     | 16.0.6      |
| Markdown                     | 3.5.2       |
| MarkupSafe                   | 2.1.5       |
| matplotlib                   | 3.8.3       |
| numpy                        | 1.26.4      |
| oauthlib                     | 3.2.2       |
| opt-einsum                   | 3.3.0       |
| packaging                    | 23.2        |
| pandas                       | 2.2.1       |
| pillow                       | 10.2.0      |
| pip                          | 23.3.1      |
| protobuf                     | 3.19.6      |
| pyasn1                       | 0.5.1       |
| pyasn1-modules               | 0.3.0       |
| pydot                        | 2.0.0       |
| pyparsing                    | 3.1.2       |
| python-dateutil              | 2.9.0.post0 |
| pytz                         | 2024.1      |
| requests                     | 2.31.0      |
| requests-oauthlib            | 1.3.1       |
| rsa                          | 4.9         |
| scikit-learn                 | 1.4.1.post1 |
| scipy                        | 1.12.0      |
| setuptools                   | 68.2.2      |
| six                          | 1.16.0      |
| tensorboard                  | 2.10.1      |
| tensorboard-data-server      | 0.6.1       |
| tensorboard-plugin-wit       | 1.8.1       |
| tensorflow                   | 2.10.0      |
| tensorflow-estimator         | 2.10.0      |
| tensorflow-io-gcs-filesystem | 0.31.0      |
| termcolor                    | 2.4.0       |
| threadpoolctl                | 3.3.0       |
| typing_extensions            | 4.10.0      |
| tzdata                       | 2024.1      |
| urllib3                      | 2.2.1       |
| Werkzeug                     | 3.0.1       |
| wheel                        | 0.41.2      |
| wrapt                        | 1.16.0      |

## Resolución de Problemas Comunes

Si encuentra algún problema durante el proceso de configuración o uso del entorno virtual, consulte la documentación oficial de Anaconda <https://docs.anaconda.com/free/anaconda/install/>.

Si encuentra algún problema con el uso de TensorFlow revise la versión de python, la versión de TensorFlow y el sistema operativo que está usando. Además, puedes consultar la documentación oficial de TensorFlow en <https://www.tensorflow.org/install?hl=es>

Una vez configurado correctamente, puede utilizar este entorno virtual para desarrollar y entrenar modelos de machine learning con TensorFlow y otras bibliotecas de Python de manera eficiente y sin conflictos de dependencias.

## **Anexo 2 - Configuración Arduino IDE y características de la placa de desarrollo**

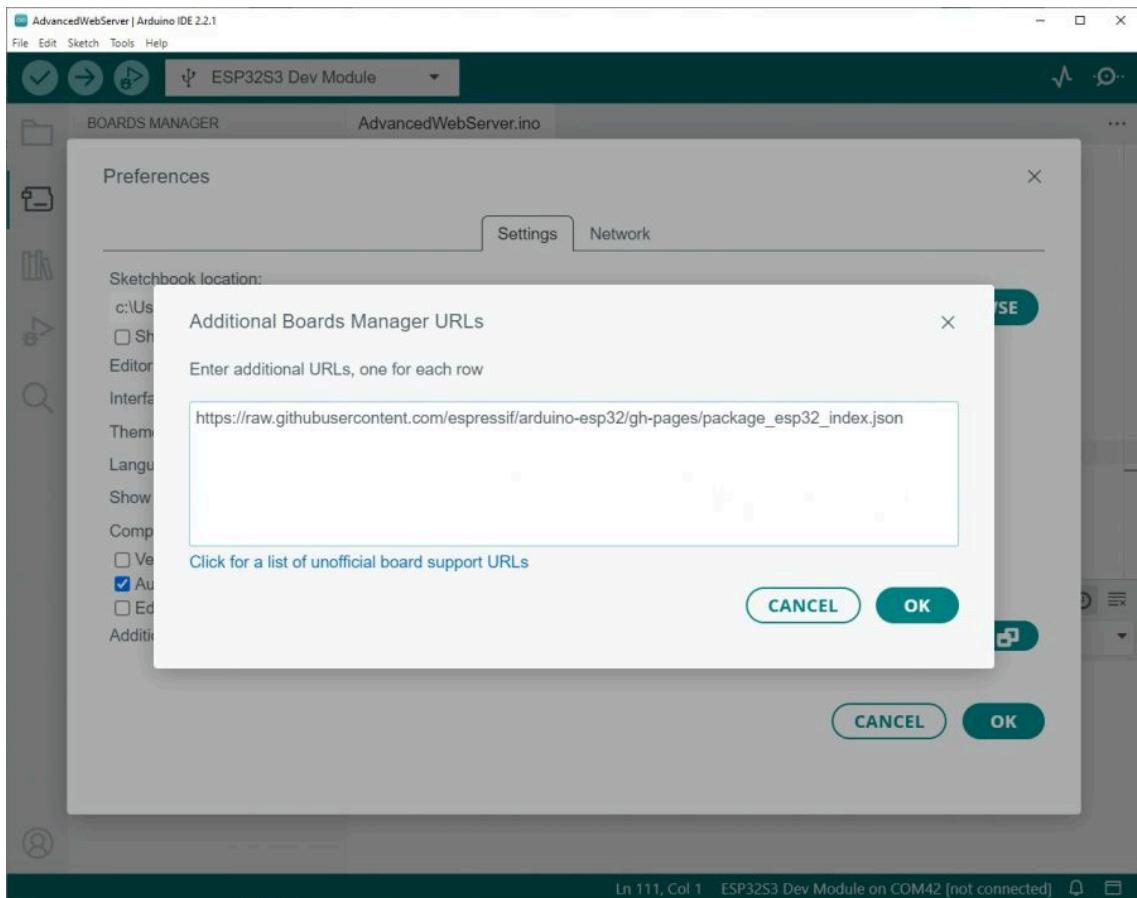
### ***Descarga e Instalación de Arduino IDE***

Descarga Arduino IDE desde el sitio web oficial. <https://www.arduino.cc/en/software>

Instala Arduino IDE siguiendo las instrucciones proporcionadas en el sitio web.

### ***Configuración del Entorno***

- Abre Arduino IDE.
- Ve a Archivo → Preferencias.
- En la sección de "URL de Administrador de Placas Adicionales", ingresa el siguiente enlace:
- [https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json)
- Haz clic en "OK" para guardar la configuración.



## ***Instalación de Placas Adicionales***

- Ve a Herramientas → Placa → Administrador de Placas
- Busca "esp32" en el Administrador de Placas.
- Selecciona la opción correspondiente a la placa ESP32 y haz clic en "Instalar".
- Reinicia Arduino IDE para que los cambios surtan efecto

## ***Selección de Opciones de Configuración***

- Abre Arduino IDE.
- Ve a Herramientas → Placa y selecciona “ESP32S3 DevModule” como placa de desarrollo.
- Ve a Herramientas → Flash Size y elige "16MB Flash".
- Habilita "QSPI PSRAM" en las opciones de configuración.

|  |
|--|
| Events Run On: "Core 1"  |
| Flash Mode: "QIO 80MHz"  |
| Flash Size: "16MB (128Mb)"   |
| JTAG Adapter: "Disabled"   |
| Arduino Runs On: "Core 1"  |
| USB Firmware MSC On Boot: "Disabled"                                 |
| Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)" |
| PSRAM: "QSPI PSRAM"  |
| Upload Mode: "UART0 / Hardware CDC"                                  |
| Upload Speed: "921600"   |

## ***Descripción General***

La ESP32-S3-Touch-LCD-1.28 es una placa de desarrollo de microcontroladores de bajo coste y alto rendimiento diseñada por Waveshare. Presenta una pantalla LCD táctil capacitiva de 1.28 pulgadas, un chip de carga de batería de litio, un sensor de seis ejes (acelerómetro de tres ejes y giroscopio de tres ejes) y otros periféricos. La placa está basada en el ESP32-S3R2, que es un sistema en chip (SoC) integrado con Wi-Fi de bajo consumo y BLE 5.0. También incluye una memoria Flash externa de 16 MB y 2 MB de PSRAM. El SoC cuenta con aceleradores de cifrado de hardware, RNG, HMAC y módulos de firma digital, cumpliendo con los requisitos de seguridad de Internet de las Cosas (IoT). Sus diversos modos de operación de bajo consumo son adecuados para aplicaciones en IoT, dispositivos móviles, dispositivos electrónicos portátiles, aplicaciones de hogar inteligente y otros escenarios.



## Parámetros

### Parámetros de la Pantalla LCD

|                          |                       |                             |                         |
|--------------------------|-----------------------|-----------------------------|-------------------------|
| <b>Chip Táctil</b>       | CST816S               | <b>Interfaz Táctil</b>      | I2C                     |
| <b>Chip de Pantalla</b>  | GC9A01A               | <b>Interfaz de Pantalla</b> | SPI                     |
| <b>Resolución</b>        | 240(H)RGB x<br>240(V) | <b>Tamaño de Pantalla</b>   | Φ32.4mm                 |
| <b>Panel de Pantalla</b> | IPS                   | <b>Tamaño de Pixel</b>      | 0.135(H)x0.135(V)<br>mm |

### Parámetros de la IMU

|                     |  |
|---------------------|--|
| <b>Sensor</b>       | QMI8658  |
| <b>Acelerómetro</b> | Resolución: 16 bits<br><br>Rango (Opcional): ±2, ±4, ±8, ±16g                                      |
| <b>Giroscopio</b>   | Resolución: 16 bits<br><br>Rango (Opcional): ±16, ±32, ±64, ±128, ±256, ±512, ±1024,<br>±2048°/sec |

Podemos obtener información adicional sobre la placa y su uso accediendo a la wiki oficial de Waveshare, donde se pueden encontrar más detalles y recursos útiles.

<https://www.waveshare.com/wiki/ESP32-S3-Touch-LCD-1.28>

## **Anexo 3 - Entrega del Código Fuente y Documentación**

En este anexo se detalla la entrega del código fuente y la documentación correspondiente al proyecto. Todo el material relacionado con el desarrollo del proyecto ha sido comprimido en un archivo .zip y entregado junto a la memoria del TFG.

El archivo .zip entregado contiene todo el repositorio del proyecto, el cual incluye:

### **Código Fuente Completo:**

Todo el código desarrollado a lo largo del TFG, incluyendo scripts, configuraciones y archivos de soporte. El repositorio incluye tanto la parte funcional del proyecto como el código desarrollado durante el proceso de investigación y pruebas.

### **Manual de Instrucciones - README.md:**

Dentro del archivo .zip se encuentra un archivo README.md que actúa como manual de instrucciones. Este archivo proporciona una guía detallada sobre cómo instalar, configurar y utilizar el proyecto.

### **Instrucciones para Acceder al Código Fuente**

Para acceder al contenido del archivo .zip y utilizar el proyecto, sigue estos pasos:

Utiliza un programa de descompresión (como WinRAR, 7-Zip, o la herramienta de descompresión de tu sistema operativo) para extraer el contenido del archivo .zip.

Una vez descomprimido, navega por las diferentes carpetas (/models, /resources, /src) para explorar el código fuente y la documentación.

Abre el archivo README.md para obtener instrucciones detalladas sobre la instalación, configuración y uso del proyecto.

La entrega del código fuente en formato .zip garantiza que todo el trabajo realizado durante el TFG esté documentado y accesible. Esto incluye no solo la versión final funcional del proyecto, sino también el historial completo de desarrollo, pruebas y ajustes realizados. El README.md actúa como una guía completa para cualquier persona que desee entender y utilizar el proyecto.