Universidad de Cantabria

Facultad de Ciencias

MODELOS DE APRENDIZAJE PROFUNDO APLICADOS AL RECONOCIMIENTO DEL PARKINSON EN PATRONES DE MARCHA CON SENSORES INERCIALES

(Deep Learning Approaches for Recognizing Parkinson's Disease through Gait Analysis with Inertial Sensors)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Gonzalo Peña Manterola

Directora: Cristina Tirnauca

Junio – 2025

Resumen

Este Trabajo de Fin de Grado presenta el desarrollo de un sistema automático para la detección de la enfermedad de Parkinson a partir de datos recogidos mediante sensores inerciales durante la realización de distintas tareas motoras. El sistema se basa en redes neuronales bidireccionales de memoria a largo y corto plazo, especialmente eficaces en el análisis de series temporales. El proceso incluye una fase de integración y preprocesamiento de datos, seguida del diseño y entrenamiento del modelo. Además, incorpora una interfaz gráfica que permite cargar nuevos datos y obtener predicciones de forma intuitiva. El rendimiento del sistema se evalúa mediante diversas métricas.

Palabras clave: inteligencia artificial, aprendizaje automático, redes neuronales recurrentes, redes neuronales bidireccionales de memoria a largo y corto plazo, detección de Parkinson, sensores inerciales

Abstract

This Final Degree Project presents the development of an automatic system for the detection of Parkinson's disease based on data collected through inertial sensors during the execution of various motor tasks. The system is built upon bidirectional Long Short-Term Memory neural networks (Bi-LSTM), which are particularly effective in time series analysis. The process involves a phase of data integration and preprocessing, followed by the design and training of the model. Additionally, a graphical user interface is included, allowing new data to be loaded and predictions to be obtained in an intuitive manner. The system's performance is evaluated using various metrics.

Keywords: artificial intelligence, machine learning, recurrent neural networks, Bi-LSTM, Parkinson's disease detection, inertial sensors

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El diagnóstico y seguimiento de enfermedades neurodegenerativas como el Parkinson representa un desafío clínico significativo debido a la variabilidad interindividual de los síntomas y la necesidad de evaluaciones objetivas. En este contexto, el uso de sensores para registrar datos de movimiento ha abierto nuevas posibilidades para el análisis automatizado de patrones, permitiendo identificar alteraciones características de esta patología mediante técnicas de inteligencia artificial.

El presente Trabajo de Fin de Grado (TFG) se enmarca en esta línea de investigación y tiene como objetivo desarrollar un sistema de clasificación automática de sujetos (con o sin Parkinson), utilizando como fuente de datos señales de movimiento capturadas por sensores distribuidos en diferentes segmentos corporales. Para ello, se propone una arquitectura de red neuronal recurrente de tipo *Bidirectional Long Short-Term Memory* (Bi-LSTM), entrenada sobre secuencias obtenidas a partir de tres tareas motoras realizadas por cada sujeto.

A diferencia de las arquitecturas unidireccionales tradicionales, las redes Bi-LSTM permiten procesar simultáneamente información pasada y futura, lo cual resulta especialmente útil en señales temporales complejas. En este proyecto, dicha capacidad se ha aprovechado para mejorar la capacidad del modelo de detectar patrones motores asociados al Parkinson.

Junto con el modelo de clasificación, se ha desarrollado una aplicación de escritorio en Python, basada en la biblioteca `tkinter`, que permite automatizar el flujo de trabajo completo: desde la selección de la carpeta con datos hasta la predicción del diagnóstico. Esta herramienta busca facilitar la interacción del usuario con el sistema, sin requerir conocimientos avanzados en programación o aprendizaje automático.

El código completo del proyecto, incluyendo el preprocesamiento de datos, la arquitectura del modelo, la implementación de la interfaz gráfica y las herramientas de evaluación, se encuentra disponible en el repositorio de GitHub del autor en la siguiente dirección: https://

github.com/gonzalopenamanterola/TFG_Gonzalo_Pe-a_Parkinson.git. De esta manera, se fomenta la reproducibilidad y la transparencia de la investigación, permitiendo a otros investigadores y profesionales explorar, adaptar y mejorar el sistema.

El resto del documento se estructura de la siguiente manera. El Capítulo 2 describe en detalle el conjunto de datos utilizado y el flujo de preprocesamiento aplicado para adecuar los datos a las necesidades del modelo. El Capítulo 3 expone el proceso metodológico seguido, desde la planificación inicial hasta el desarrollo de la herramienta final, incluyendo las decisiones de diseño y las etapas de validación. A continuación, el Capítulo 4 presenta una revisión teórica sobre redes neuronales recurrentes, LSTM y Bi-LSTM, abordando sus fundamentos y aplicaciones relevantes en el análisis de series temporales. En el Capítulo 5 se detalla la arquitectura del modelo desarrollado y la implementación de la aplicación gráfica, destacando sus características y funcionalidades. Por último, el Capítulo 6 recoge las conclusiones derivadas del trabajo, así como posibles líneas de trabajo futuro.

Capítulo 2

Origen, estructura y características del dataset utilizado

En el presente TFG se utilizan los datos obtenidos a partir de un experimento desarrollado en el marco de un estudio clínico. La muestra del estudio está compuesta por un total de 41 pacientes diagnosticados con enfermedad de Parkinson, 20 de ellos portadores de la mutación LRRK2 G2019S[1] y 21 con Parkinson idiopático[2], y 36 individuos sanos, de los cuales 17 presentan algún vínculo familiar con personas portadoras de la mutación LRRK2. La recolección de los datos se llevó a cabo en el año 2018.

Los participantes del estudio se encuentran organizados en diferentes carpetas de datos, clasificadas de la siguiente manera:

- AsG2019S-: individuos sanos con algún vínculo familiar con portadores de la mutación LRRK2.
- CONTROLES: sujetos sanos sin relación familiar con personas portadoras de la mutación.
- EPG2019S: pacientes diagnosticados con enfermedad de Parkinson idiopática.
- EPI: pacientes con enfermedad de Parkinson portadores de la mutación LRRK2 G2019S, siendo estos los únicos participantes con dicha mutación.

Adicionalmente, se utilizarán los datos correspondientes a otros 27 sujetos portadores de la mutación LRRK2 G2019S que, al momento del experimento, no presentaban síntomas

[1] La mutación LRRK2 G2019S es una de las variantes genéticas más comunes asociadas al Parkinson hereditario

[2] El Parkinson idiopático se refiere a los casos de enfermedad de Parkinson cuya causa es desconocida, sin una mutación genética identificable ni antecedentes familiares claros.

clínicos de la enfermedad de Parkinson. La inclusión de estos individuos tiene un doble propósito: por un lado, permite explorar la capacidad del modelo para distinguir entre sujetos asintomáticos con riesgo genético y sujetos diagnosticados; por otro, refuerza la motivación principal del presente trabajo, que radica en la posible utilidad de sistemas automáticos de clasificación para la detección temprana o el seguimiento preventivo en poblaciones de riesgo. En este sentido, el modelo desarrollado podría servir como herramienta de apoyo en estudios longitudinales, donde el objetivo es anticipar la aparición de síntomas a partir de patrones sutiles en los datos de movimiento.

2.1. Descripción del sistema de adquisición y del protocolo de pruebas

Los participantes fueron equipados con un total de 16 sensores ligeros (STT-IWS, STTSystems, San Sebastián, España), distribuidos como se puede ver en la Figura 2.1: uno en la frente (1), uno en el torso (1), uno en cada hombro (2), uno en cada codo (2), uno en cada muñeca (2), uno en cada cadera (2), uno en cada rodilla (2), uno en cada tobillo (2), uno en la nuca (1) y uno en la pelvis (1).

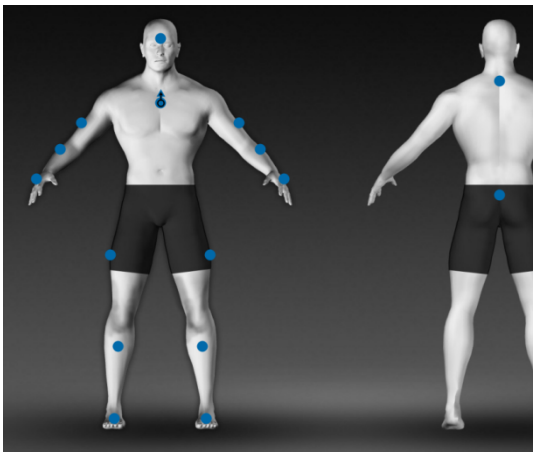

Figura 2.1: Distribución anatómica de sensores inerciales utilizados durante captura de datos.

Los sensores utilizados por cada sujeto están sincronizados y precalibrados con referencia al eje vertical y a su posición anatómica. Recogen información discreta cada 0,01 segundos sobre los ángulos de cada segmento respecto a su posición inicial.

Se realizaron tres experimentos por participante, todos ellos desarrollados en un entorno controlado consistente en un pasillo de 15 metros de longitud. Durante las pruebas, los recorridos se efectuaron caminando, con giros realizados según fuera necesario. Cada sujeto fue equipado con los 16 sensores descritos anteriormente.

- **1 MINUTO MARCHA**: el sujeto tiene que recorrer el pasillo durante un minuto caminando normal, girando las veces que sea necesario.
- **15 METROS RAPIDO**: el sujeto tiene que caminar los 15 metros lo más rápido que pueda, sin darse la vuelta.
- **DOUBLETASK**: consiste en caminar durante un minuto mientras se cuenta mentalmente de 100 a 0 en intervalos de tres, permitiéndose realizar giros durante la tarea.

2.2. Tratamiento y exploración visual de los datos

Para cada experimento se almacenan los datos en archivos en formato .csv, que contienen las mediciones capturadas por los sensores ubicados en las respectivas partes del cuerpo. Hay nueve ficheros: `Caderas.csv`, `Codos.csv`, `Cuello.csv`, `Hombros.csv`, `Muñecas.csv`, `Pelvis.csv`, `Rodillas.csv`, `Tobillos.csv` y `Torso.csv`. Estos ficheros incluyen columnas que representan distintas variables, tales como los ángulos en cada uno de los planos de movimiento de la articulación correspondiente y los tiempos (en el caso del cuello, por ejemplo, se registran parámetros como la flexión lateral, la flexo-extensión y la rotación). Estas características permiten describir de manera precisa el comportamiento dinámico de cada articulación durante la marcha. Por otro lado, aproximadamente la mitad de las columnas originales están relacionadas con marcas temporales, por lo que son eliminadas en las etapas iniciales del preprocesamiento.

Para la visualización de los datos se selecciona el sujeto AGF en el experimento **1 MINUTO MARCHA**.

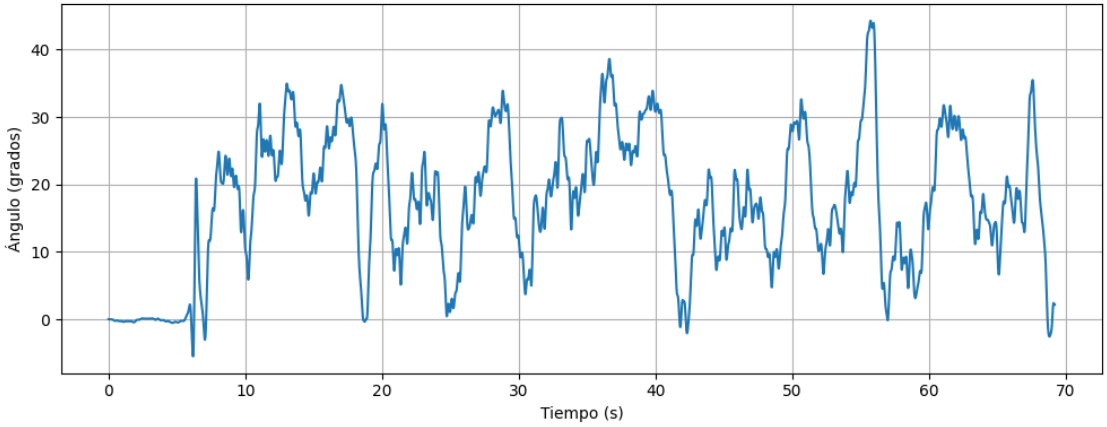

Figura 2.2: Gráfica del ángulo de flexo-extensión del cuello a lo largo del tiempo.

Se emplea el archivo `Cuello.csv` como ejemplo para ilustrar la estructura de los datos registrados. En este ejemplo se observa la flexo-extensión cervical (Figura 2.2) y la rotación vertical del cuello del sujeto (Figura 2.3). Estos datos, sin la componente del tiempo, son los utilizados para entrenar el modelo.

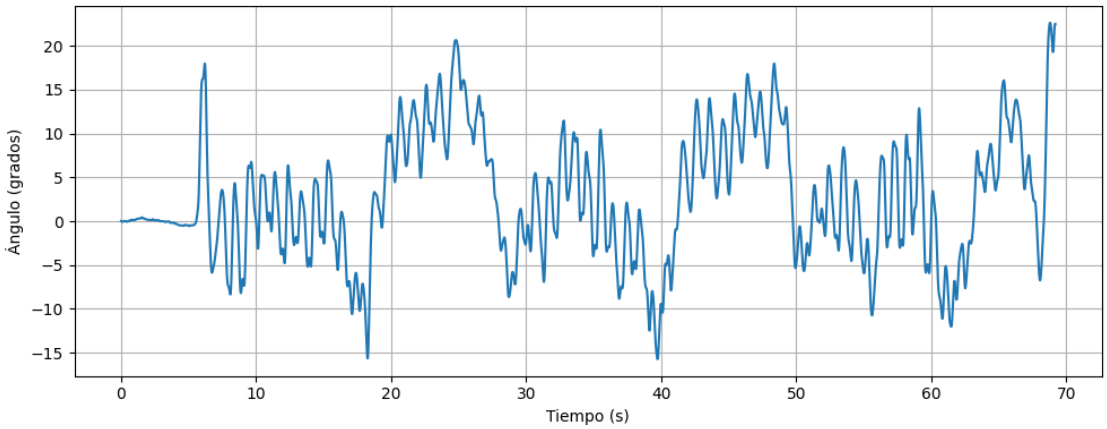

Figura 2.3: Gráfica del ángulo de rotación vertical del cuello a lo largo del tiempo.

Para cada sujeto, los datos recogidos en los distintos experimentos se integraron en un único archivo denominado `AllData.csv`, que contiene la información combinada de todos los ficheros generados por los sensores corporales. Este archivo unificado incluye una columna adicional que identifica el origen de cada fila de datos según el segmento corporal registrado. Dado que cada sujeto realiza tres experimentos diferentes, los archivos individuales de cada experimento se agruparon y concatenaron por filas en un único archivo por sujeto, de modo que el modelo opera sobre una única muestra por paciente. Esta estrategia permite aprovechar toda la información disponible en las distintas tareas motoras y garantiza que el proceso de clasificación se realice a nivel de individuo, y no de experimento aislado.

Los pacientes provenientes de las carpetas CONTROLES y AsG2019S- reciben la etiqueta **0**, mientras que los de las carpetas EPI y EPG2019S reciben la etiqueta **1**.

Los modelos utilizados en este estudio no procesan una única fila de características por paciente, sino múltiples secuencias temporales de datos. Cada una de estas secuencias tiene una dimensión de 5000×44, correspondiente a 5000 pasos de tiempo con 44 variables biomecánicas.

El número de secuencias por paciente depende de la duración total de la muestra concatenada (formada a partir de las tres tareas) y del método de segmentación, que se realiza mediante una ventana deslizante de tamaño 5000 y un paso de 2500. Cada secuencia es procesada individualmente por la red Bi-LSTM, generando una probabilidad de pertenencia a una clase. Finalmente, se calcula la media de todas las predicciones de un paciente, y en función de este valor se determina su clasificación binaria final.

Cabe destacar que los 5000 pasos temporales de cada secuencia no equivalen directamente a

5000 segundos. El intervalo temporal real depende de la frecuencia de muestreo del sistema de captura de movimiento. Si dicha frecuencia es f muestras por segundo, el tiempo total que representa una ventana es el de la ecuación (2.1):

$$\text{Duración (segundos)} = \frac{5000}{f} \tag{2.1}$$

Por ejemplo, si el sistema opera a 100 Hz ($f = 100$), cada secuencia abarca un intervalo temporal de 50 segundos. En este estudio, la frecuencia de muestreo utilizada es de 100 Hz.

El conjunto total de datos se divide en un 80 % para entrenamiento y un 20 % para validación, garantizando una evaluación robusta del rendimiento del modelo.

Capítulo 3

Proceso metodológico: del análisis de datos al desarrollo de la herramienta final

El primer paso consistió en una fase de planificación, donde se definieron los objetivos del proyecto y se seleccionó el enfoque de aprendizaje automático, concretamente el uso de redes neuronales Bi-LSTM para abordar la naturaleza secuencial de los datos. En paralelo, se realizó una revisión bibliográfica exhaustiva para fundamentar la aplicabilidad de estas arquitecturas en el ámbito biomédico [3, 6].

Posteriormente, se inició una fase de exploración de los datos, donde se analizó la estructura de los ficheros proporcionados, organizados por sujeto y subdivididos en tres pruebas motoras. Se identificaron inconsistencias en las carpetas y nomenclaturas de archivo que fueron tenidas en cuenta en el diseño del procesamiento posterior [7].

3.1. Selección de la metodología de desarrollo

Durante la fase de diseño metodológico, se valoraron distintas metodologías de desarrollo de software para la construcción de la herramienta final. Entre las opciones analizadas se incluyeron metodologías tradicionales como el modelo en cascada (*waterfall*), el prototipado y las metodologías iterativas e incrementales, así como metodologías ágiles como Scrum, Kanban y Lean.

En este trabajo, se optó por una metodología de desarrollo incremental, combinada con elementos de prototipado. Esta decisión se fundamenta en las siguientes razones:

- **Naturaleza exploratoria del proyecto:** en el contexto de investigación biomédica y análisis de señales, es habitual que las hipótesis iniciales evolucionen conforme se avanza en el análisis de los datos. La metodología incremental permite ir validando y ajustando cada módulo de manera iterativa.
- **Necesidad de construir prototipos funcionales:** antes de llegar a la herramienta final, se desarrollaron prototipos parciales (por ejemplo, para el preprocesamiento, para la fusión de datos y para la visualización de resultados) que permitieron validar la viabilidad técnica y recibir retroalimentación.
- **Modularidad y flexibilidad:** al trabajar con módulos independientes, la metodología incremental facilitó la integración progresiva y el testeo individual de cada componente.

Se descartó el uso de metodologías puramente en cascada debido a que los requisitos y las pruebas evolucionaron a medida que se profundizó en los datos y se identificaron problemas como la falta de homogeneidad en los ficheros o la necesidad de ajustar hiperparámetros del modelo. Por otro lado, se consideró que la metodología Scrum, al requerir reuniones diarias y una estructura de equipo más amplia, no se adaptaba a las características individuales de este TFG.

3.2. Desarrollo y validación del sistema

La preparación de los datos implicó el desarrollo de un *script* llamado `fusionar_datos.py`, encargado de combinar todos los archivos `.csv` de un mismo sujeto en un único fichero `AllData.csv`. Este *script* fue adaptado para manejar la estructura de carpetas identificada en la fase de exploración. Además, se implementó la normalización de datos, el filtrado de columnas no informativas y la generación de secuencias temporales de longitud uniforme, de cara a su posterior integración en la red neuronal.

En la fase de modelado, se construyó una red neuronal Bi-LSTM utilizando la biblioteca Keras. Inicialmente se validó el entorno de trabajo ejecutando modelos sencillos con datos sintéticos para verificar las dependencias necesarias, como `TensorFlow`. Una vez procesados los datos reales, se realizaron experimentos iterativos ajustando el número de capas, el tamaño de las capas LSTM y las técnicas de regularización (`Dropout`, `Batch Normalization` y penalización `L2`). Esta experimentación permitió identificar la mejor arquitectura para el problema de clasificación binaria.

3.3. Evaluación del modelo

Para evaluar el rendimiento del modelo, se utilizó el conjunto de prueba previamente mencionado (correspondiente al 20 % de los datos), separado de forma estratificada para preservar la distribución de clases. Adicionalmente, se realizó un análisis con 27 sujetos sanos en 2018, de los cuales se sabe retrospectivamente que tres de ellos desarrollaron Parkinson antes del 2024. Los resultados permitieron analizar la capacidad del modelo para distinguir entre sujetos sanos y afectados, así como comprender los posibles sesgos y desequilibrios en las predicciones. Se dedicó especial atención al análisis de falsos positivos y falsos negativos, considerando sus implicaciones clínicas y la posible utilidad de las predicciones como herramienta de apoyo en la monitorización de pacientes.

3.4. Implementación de la herramienta final

Con el objetivo de transferir el modelo a un entorno de uso práctico, se desarrolló una interfaz gráfica de usuario utilizando `Tkinter`. Esta herramienta facilita la carga de carpetas con datos de nuevos pacientes, ejecuta automáticamente el preprocesamiento, realiza la clasificación y muestra los resultados de manera clara y accesible. Esta interfaz se diseñó para ser utilizada por personal clínico o técnico sin necesidad de conocimientos avanzados en programación.

En conjunto, la combinación de un enfoque incremental con prototipado y la validación continua de cada módulo permitió adaptar el sistema a las necesidades y limitaciones de los datos, así como mejorar su fiabilidad y facilidad de uso.

Capítulo 4

Arquitectura y configuración del modelo

En el contexto del análisis de datos secuenciales, como los derivados de series temporales o movimientos en sujetos con Parkinson, las redes neuronales recurrentes (RNN) han demostrado ser modelos eficaces para capturar dependencias temporales. Estos modelos resultan particularmente útiles en tareas de clasificación y predicción de patrones complejos. Dentro de los avances más relevantes en este tipo de redes, las LSTM ofrecen una solución a las limitaciones de las RNN tradicionales, especialmente en lo que respecta a la capacidad para gestionar dependencias a largo plazo. Adicionalmente, las Bi-LSTM extienden esta capacidad al procesar las secuencias en ambas direcciones, lo que mejora la captura de información contextual. Este capítulo presenta una descripción general de los modelos empleados en el presente estudio, abordando las características y el potencial de mejora de cada uno.

4.1. Redes neuronales densas

Las RNN incorporan capas densas, también conocidas como *fully connected layers*, que constituyen una de las estructuras más fundamentales en las redes neuronales artificiales. En este tipo de capa, cada neurona está conectada a la totalidad de las neuronas de la capa anterior, lo que permite una integración completa de la información extraída en etapas previas del modelo. Estas capas son especialmente útiles en la fase final de las arquitecturas complejas, donde se requiere transformar las representaciones de alto nivel generadas por capas anteriores en decisiones concretas, como una clasificación binaria o multiclase.

Aunque la transformación principal que realizan estas capas es lineal —mediante la multi-

plicación de una matriz de pesos y la adición de un sesgo—, esta operación se complementa con una función de activación no lineal como se ve en la ecuación (4.1):

$$y = f(Wx + b) \tag{4.1}$$

donde x es el vector de entrada, W representa la matriz de pesos, b el vector de sesgos, y f una función de activación no lineal (como ReLU[1] o sigmoide[2].). Esta estructura (Figura 4.1) le permite al modelo aprender combinaciones no lineales de las características extraídas, lo que resulta fundamental para tareas de predicción y toma de decisiones.

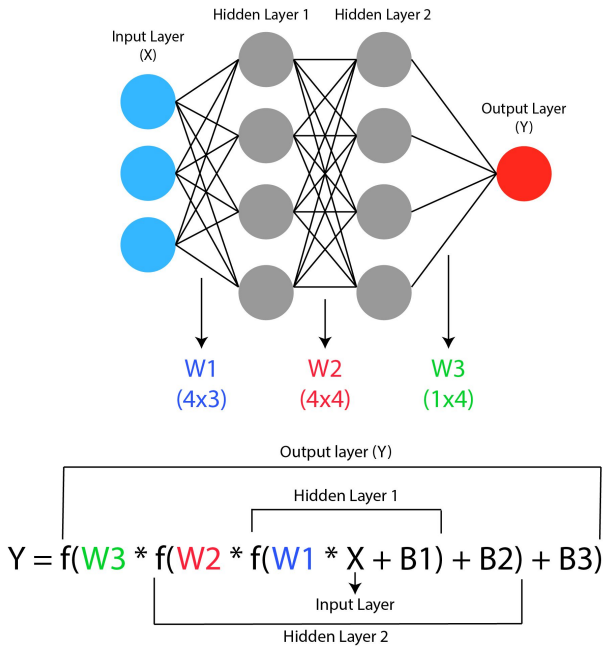

Figura 4.1: Esquema de una red neuronal densa, en la que cada neurona de una capa está conectada a todas las de la siguiente. Fuente: https://www.cs.us.es/~fsancho/Blog/posts/Redes_Neuronales/

En el presente trabajo, las capas densas se integran posteriormente al bloque Bi-LSTM (cuya arquitectura se detalla en la sección 4.4) para sintetizar la información secuencial capturada por la red y generar la salida final de clasificación.

[1] La activación `ReLU` (*Rectified Linear Unit*) se define como $f(x) = \max(0, x)$.
[2] La función sigmoide$(x) = 1/(1 + \exp(-x))$

4.2. Redes neuronales recurrentes

Una RNN es una arquitectura de red neuronal que contiene ciclos en sus conexiones, lo que significa que la salida de una unidad depende de sus propias salidas anteriores. Esta estructura permite que la red tenga una **memoria** de entradas pasadas. Estas redes son útiles por sí mismas y sirven como base para enfoques más complejos [1].

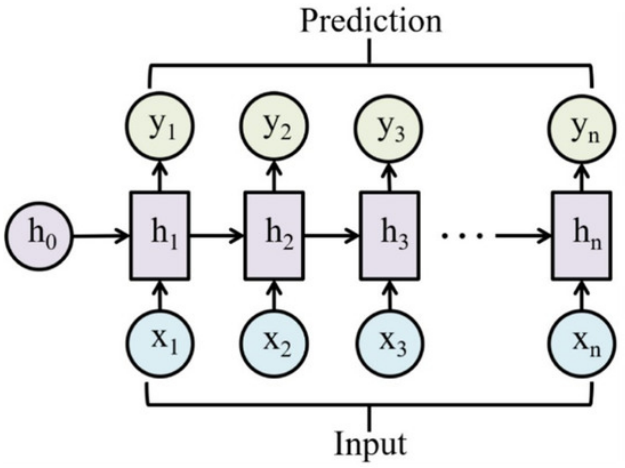

Figura 4.2: Estructura conceptual de una RNN, mostrando cómo el estado oculto se retroalimenta a sí mismo a lo largo de la secuencia temporal. Fuente: [4]

La estructura básica de una RNN está representada en la Figura 4.2.

- Entrada x_t: representación vectorial de la entrada en tiempo t
- Capa oculta h_t: calculada a partir de la entrada actual y salida de la capa oculta del tiempo anterior, esto hace que sea una especie de memoria o contexto.
- Salida y_t: generada a partir de la capa oculta actual.

En las RNN, la inferencia hacia adelante consiste en mapear una secuencia de entradas a una secuencia de salidas. El cálculo de la salida en el tiempo t requiere conocer el valor de activación de la capa oculta h_t. Para obtener este valor, se multiplican la entrada x_t y la capa oculta del paso anterior h_{t-1} por las respectivas matrices de pesos W y U, y se suman los resultados. Este valor es procesado mediante una función de activación g para determinar la activación de la capa oculta en el tiempo t (véase la ecuación (4.2)).

$$h_t = g(Uh_{t-1} + Wx_t) \tag{4.2}$$

Posteriormente, el valor h_t se utiliza para generar la salida y_t, que se calcula aplicando una función de activación f, generalmente una función softmax[3]. Es decir, la salida y_t se calcula mediante la fórmula (4.3).

$$y_t = \text{softmax}(V h_t) \tag{4.3}$$

En cuanto a la dimensión de las matrices que intervienen en las operaciones, se tiene:

- $U \in \mathbb{R}^{d_h \times d_h}$, matriz de pesos que conecta el estado oculto anterior con la puerta actual.
- $W \in \mathbb{R}^{d_h \times d_{in}}$, matriz de pesos que conecta la entrada de la red con la puerta actual.
- $V \in \mathbb{R}^{d_{out} \times d_h}$, matriz de pesos que conecta la capa oculta con la salida final.

Aquí, d_{in} representa el número de características de la entrada, d_h el número de unidades de la capa oculta y d_{out} el tamaño de la salida (por ejemplo, 1 para clasificación binaria).

Las RNN se distinguen por su capacidad teórica para conectar la información pasada con la predicción presente, lo cual resulta particularmente útil en tareas que involucran secuencias de datos, como es el caso de las series temporales. No obstante, en la práctica, las RNN enfrentan un desafío significativo cuando se trata de modelar dependencias a largo plazo en estas secuencias. Si bien las RNN pueden manejar eficientemente contextos donde la relación entre los valores previos es cercana, su rendimiento se ve afectado cuando la distancia entre los datos relevantes y el punto donde se requiere la predicción es considerable. Este problema se debe a limitaciones inherentes al entrenamiento de las RNN, como el desvanecimiento del gradiente [2], lo que dificulta la captura de dependencias temporales a largo plazo.

Estas limitaciones se evidencian en tareas donde la relación entre elementos distantes dentro de una secuencia temporal es crucial para realizar una predicción correcta. Para superar estos problemas, se han desarrollado alternativas como las LSTM, que están específicamente diseñadas para mejorar el modelado de dependencias a largo plazo. Las LSTM incorporan mecanismos de memoria que permiten que la red retenga información relevante durante períodos más largos, superando las deficiencias observadas en las RNN tradicionales.

[3]La función $\text{softmax}(x)_i = \dfrac{e^{x_i}}{\sum_j e^{x_j}}$ genera una distribución de probabilidad sobre las posibles clases de salida.

4.3. Redes LSTM

Estas redes extienden la arquitectura básica de las RNN mediante la introducción de una estructura interna denominada **celda de memoria**, junto con mecanismos de control denominados **puertas** (*gates*), que permiten regular el flujo de información en cada unidad, facilitando así la retención y el olvido de datos de manera controlada a lo largo del tiempo.

La arquitectura de una celda LSTM consta de tres puertas principales:

- **Puerta de olvido** (*forget gate*): decide qué información debe ser descartada del estado de la celda.
- **Puerta de entrada** (*input gate*): determina qué nueva información debe almacenarse en la celda.
- **Puerta de salida** (*output gate*): controla qué parte de la información contenida en la celda se transmite a la salida de la unidad.

En la Figura 4.3 se muestra un esquema de la unidad LSTM, donde se ilustran las puertas y la celda de memoria interna.

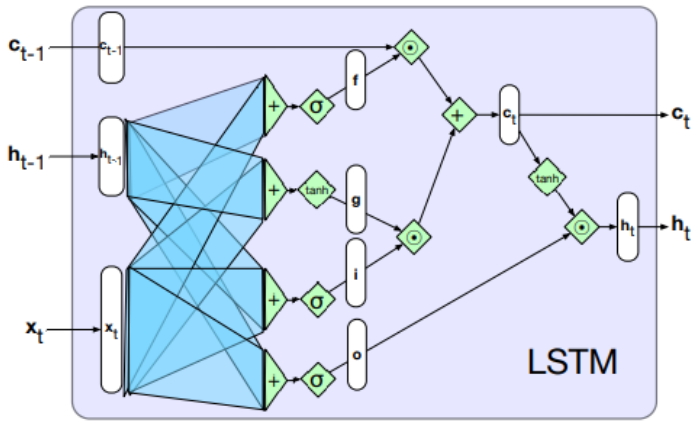

Figura 4.3: Diagrama de una unidad LSTM, donde se ilustran las tres puertas principales (entrada, olvido y salida) y la celda de memoria interna. Fuente: https://eva.fing.edu.uy/pluginfile.php/408885/mod_resource/content/1/Tema5_3.pdf

Estas puertas se definen mediante funciones sigmoides y producto escalar, permitiendo a la red aprender dinámicamente qué información conservar o descartar. Las operaciones que describen el funcionamiento de una unidad LSTM típica están descritas en las ecuaciones (4.4)-(4.9).

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \quad \text{(puerta de olvido)} \quad (4.4)$$
$$g_t = \tanh(U_g h_{t-1} + W_g x_t) \quad \text{(candidatos a nueva información)} \quad (4.5)$$
$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \quad \text{(puerta de entrada)} \quad (4.6)$$
$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \quad \text{(puerta de salida)} \quad (4.7)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad \text{(estado de la celda)} \quad (4.8)$$
$$h_t = o_t \odot \tanh(c_t) \quad \text{(salida de la unidad)} \quad (4.9)$$

En estas ecuaciones, $U \in \mathbb{R}^{d_h \times d_h}$ es la matriz de pesos que conecta el estado oculto anterior con las diferentes puertas, mientras que $W \in \mathbb{R}^{d_h \times d_{in}}$ conecta la entrada x_t con cada puerta. Aquí, d_{in} es la dimensión de las características de entrada, d_h el número de unidades en la capa oculta y d_{out} la dimensión de la salida final. A su vez, σ es la función sigmoide, tanh la tangente hiperbólica, $\odot$ denota el producto elemento a elemento, x_t es la entrada en el tiempo t, h_{t-1} es el estado oculto anterior, y c_t es el estado de la celda en el tiempo t.

La inclusión de una celda de memoria explícita permite a las LSTM mitigar eficazmente el problema del desvanecimiento del gradiente, facilitando el aprendizaje de relaciones a largo plazo en secuencias. Gracias a esta capacidad, las LSTM han sustituido en gran medida a las RNN tradicionales en aplicaciones donde las dependencias temporales prolongadas son esenciales.

Por tanto, las redes LSTM representan una evolución estructural de las RNN, cuya arquitectura especializada las convierte en una herramienta fundamental en el modelado de secuencias complejas y de largo alcance temporal.

4.4. Redes Bi-LSTM

Las redes Bi-LSTM constituyen una extensión de las redes LSTM tradicionales. Mientras que una red LSTM estándar procesa la información de una secuencia únicamente en una dirección (generalmente del pasado hacia el futuro), las Bi-LSTM están diseñadas para procesar simultáneamente la secuencia en ambas direcciones: una red LSTM hacia adelante y otra hacia atrás, como se muestra en la Figura 4.4. Esto permite a la arquitectura capturar tanto el contexto pasado como el futuro para cada punto de la secuencia, lo que resulta especialmente útil en tareas donde la comprensión completa del entorno de un dato requiere información de ambos extremos de la secuencia.

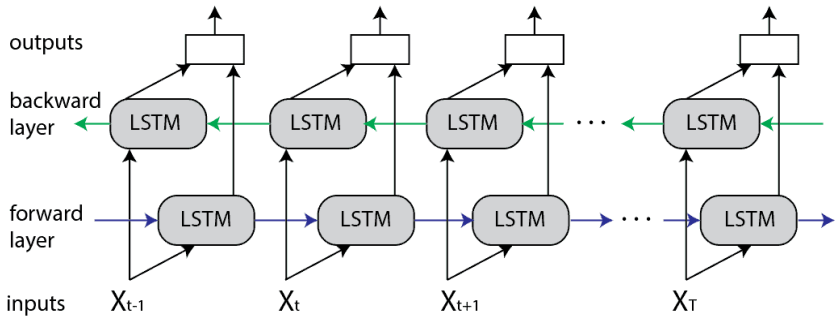

Figura 4.4: Representación de una red Bi-LSTM, en la que la información se procesa tanto hacia adelante como hacia atrás. Fuente: https://www.researchgate.net/figure/ntuition-Behind-Bidirectional-LSTM-In-Figure-5-we-illustrate-with-an-example-Suppose_fig2_365313794

En una red Bi-LSTM se utilizan dos capas LSTM independientes:

- Una **LSTM hacia adelante**, que procesa la secuencia desde el primer elemento hasta el último.
- Una **LSTM hacia atrás**, que procesa la misma secuencia en orden inverso, desde el último elemento hasta el primero.

Cada una de estas capas genera su propio conjunto de salidas ocultas, y posteriormente, las salidas de ambas direcciones se concatenan o combinan para formar la representación final en cada instante de tiempo. Formalmente, si $\overrightarrow{h_t}$ y $\overleftarrow{h_t}$ son las salidas de las LSTM hacia adelante y hacia atrás en el tiempo t, entonces la salida final de la Bi-LSTM se calcula como en la ecuación (4.10).

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}] \tag{4.10}$$

donde el símbolo ; representa la concatenación de vectores.

4.5. Desarrollo y validación del modelo Bi-LSTM

El presente apartado describe el proceso de desarrollo del modelo final empleado en este trabajo, basado en una arquitectura de red neuronal bidireccional de tipo Bi-LSTM. El objetivo principal del modelo es clasificar registros derivados de tareas motoras realizadas

por sujetos con y sin diagnóstico de Parkinson, a partir de datos cuantitativos recogidos durante la ejecución de tres pruebas distintas por sujeto.

Para ello, se ha diseñado un modelo secuencial en Keras que integra múltiples capas Bi-LSTM, junto con técnicas de regularización, normalización por lotes y una estrategia de *early stopping*, con el fin de mitigar el sobreajuste y mejorar la capacidad de generalización del sistema.

A lo largo de esta sección, se detallan la arquitectura final adoptada, el método de preparación y la entrada de los datos, los modelos alternativos considerados durante el proceso experimental y las optimizaciones implementadas en la configuración definitiva. Finalmente, se presentan los resultados obtenidos y se analizan en función de su desempeño sobre los conjuntos de entrenamiento y validación.

4.5.1. Arquitectura y estructura general del modelo

En esta sección se describe la arquitectura del modelo final propuesto, junto con los detalles de su configuración y preparación para el entrenamiento. El diseño del modelo se basa en una arquitectura secuencial de tipo Bi-LSTM, compuesta por múltiples capas bidireccionales. La red incluye además mecanismos de normalización y regularización, así como capas densas para la proyección y la salida final. La implementación se realizó utilizando la API de `Keras` sobre la plataforma `TensorFlow`.

El modelo está diseñado para procesar secuencias de longitud temporal máxima de 5000 pasos, con 44 características por paso. Estas características corresponden a variables de medición extraídas de los sensores utilizados en los experimentos y se registran en cada instante temporal, independientemente de la prueba motora realizada. Cada sujeto realiza tres pruebas motoras (1 MINUTO MARCHA, 15 M RAPIDO y DOBLETASK) cuyos datos se concatenan por filas (en el tiempo) en un único archivo `AllData.csv`, de modo que el modelo opera sobre la secuencia combinada. Así, las 44 características no se dividen en grupos según las pruebas, sino que describen el estado de los sensores en cada instante temporal, permitiendo que el modelo capture las variaciones de la señal a lo largo de todas las tareas motoras de forma conjunta. A continuación, se describen los bloques que conforman la arquitectura y las decisiones de diseño adoptadas.

Primera capa Bi-LSTM. Esta capa inicial consta de una LSTM bidireccional con 128 unidades y salida en cada paso temporal (`return_sequences=True`), generando una salida de dimensión $(5000, 256)$. Esta estructura facilita la captura de dependencias en ambas direcciones de la secuencia, lo cual es especialmente importante en contextos donde la información clave no está localizada de forma contigua. La elección de 128 unidades se realizó tras comparar empíricamente distintos tamaños de capa (64 y 256), buscando el mejor equilibrio entre capacidad de aprendizaje y complejidad computacional.

Regularización y normalización intermedia. Tras esta capa inicial, se incorpora normalización por lotes (`Batch Normalization`) y regularización mediante `Dropout` con una tasa de 0,5. La normalización por lotes estabiliza la distribución de activaciones durante el entrenamiento, reduciendo la sensibilidad a la inicialización de pesos y acelerando la convergencia. Por su parte, el `Dropout` desactiva aleatoriamente un porcentaje de neuronas que en este caso es el 50 % en cada iteración, evitando que el modelo dependa en exceso de subconjuntos de características y reduciendo así el riesgo de sobreajuste. Este uso combinado se prefiere a estrategias individuales como `L1` o `L2` porque permite aprovechar las ventajas de cada técnica de forma complementaria.

Regularización adicional. En este trabajo también se ha implementado regularización `L2` en las capas LSTM y densas. Esta técnica añade una penalización a la suma de los cuadrados de los pesos en la función de pérdida, favoreciendo soluciones con pesos más pequeños y reduciendo el sobreajuste. Dicha penalización consiste en añadir un término adicional a la función de pérdida que incrementa su valor cuando los pesos son grandes, forzando así a la red a mantenerlos controlados durante el entrenamiento. En comparación, la regularización `L1` penaliza la suma de los valores absolutos de los pesos, promoviendo la esparsidad de la red (es decir, forzando a que algunos pesos sean exactamente cero), lo cual puede ser útil para la selección de características. Sin embargo, en el presente trabajo la regularización `L2` ha resultado más adecuada para preservar un aprendizaje distribuido y robusto.

Segunda capa Bi-LSTM. Tras el primer bloque de regularización, se introduce una segunda capa Bi-LSTM con 32 unidades. Esta capa resume la información temporal completa, reduciendo la dimensionalidad a una representación de tamaño 64. Este diseño responde a una estrategia de compresión progresiva de la representación aprendida, de manera que la red retenga las características más relevantes antes de pasar a las capas densas.

Bloque final: capas densas. Después de las capas recurrentes, se aplica de nuevo normalización por lotes y regularización mediante `Dropout` (tasa 0,3). Seguidamente, se incorpora una capa densa de 8 unidades con activación `ReLU`, que actúa como un embudo no lineal que proyecta la representación aprendida a un espacio de menor dimensión. La activación `ReLU` permite introducir no linealidad en la red sin saturarse para valores positivos, lo cual facilita la propagación del gradiente y acelera la convergencia. Esta propiedad la convierte en una elección especialmente adecuada en redes profundas para modelar relaciones complejas entre las características aprendidas. Finalmente, la salida se obtiene mediante una capa densa con una única neurona y activación sigmoide, generando una probabilidad entre 0 y 1, que indica la probabilidad de pertenecer a la clase Parkinson o control.

Compilación y entrenamiento. El modelo fue compilado utilizando el optimizador `Adam` y la función de pérdida `binary_crossentropy`, adecuadas para problemas de clasificación binaria. La métrica de `accuracy` se empleó para evaluar el rendimiento del modelo. Durante el entrenamiento se utilizó un conjunto de datos previamente normalizado y particionado, aplicando la técnica de *early stopping* para detener el proceso si la pérdida de validación no

mejoraba tras varias épocas, evitando así el sobreajuste.

Resumen de resultados experimentales. Durante la fase de diseño del modelo se evaluaron diversas configuraciones arquitectónicas, ajustando el número de neuronas, el número de capas, las técnicas de regularización y las dimensiones de las capas densas. La Tabla 4.1 resume los resultados obtenidos en términos de precisión y pérdida sobre el conjunto de validación, ilustrando la evolución del rendimiento a lo largo del proceso de ajuste experimental.

Tabla 4.1: Comparación de distintas configuraciones de la red Bi-LSTM durante el proceso experimental.

Modelo	**Precisión**	**Pérdida**
Bi-LSTM (64, 32, 32, 1) con `Dropout`(0,3) sin `Normalización`	0,54	0,923
Bi-LSTM (64, 32, 32, 1) con `Dropout`(0,5) sin `Normalización`	0,59	0,874
Bi-LSTM (128, 64, 32, 1) con `Dropout`(0,5) + `Dropout`(0,3)	0,69	0,816
Bi-LSTM (256, 126, 32, 1) con `Normalización` + `Dropout`(0,5)	0,67	0,732
Modelo final (128, 32, 8, 1) + `BatchNorm` + `Dropout` + L2	**0,875**	**0,653**

El análisis de la tabla anterior refleja una evolución estructurada en la búsqueda de la configuración final, con decisiones justificadas basadas en principios de diseño de redes profundas y observaciones empíricas durante el entrenamiento. Estas decisiones incluyeron ajustes en el tamaño de las capas, la aplicación progresiva de técnicas de regularización y la integración de capas densas como embudos no lineales para optimizar la separación de clases.

En resumen, la arquitectura final adoptada demuestra un equilibrio eficaz entre capacidad de aprendizaje, robustez frente al sobreajuste y rendimiento predictivo, lo cual respalda su idoneidad para la tarea de clasificación binaria entre sujetos con y sin Parkinson.

4.5.2. Procesamiento y entrada de los datos

El conjunto de datos empleado en el presente trabajo ha sido procesado siguiendo un flujo sistemático que garantiza la uniformidad, calidad y adecuación de las muestras para su utilización en el modelo Bi-LSTM.

Inicialmente, los datos fueron extraídos de archivos `AllData.csv` almacenados en subdirectorios correspondientes a cada sujeto, clasificados en función de su condición clínica (sano o afectado por Parkinson). Cada sujeto cuenta con registros derivados de tres tareas motoras distintas: marcha normal, marcha rápida y marcha bajo doble tarea. Estos registros, de naturaleza secuencial y multivariable, fueron concatenados para cada sujeto, generando una única muestra de datos temporalmente continua.

Posteriormente, se aplicó un proceso de ajuste de longitud a todas las muestras, truncándolas

a un máximo de 5000 pasos temporales o rellenándolas mediante *padding* con ceros cuando el número de pasos era inferior. Esta operación garantiza que todas las secuencias presenten una longitud constante, condición necesaria para su introducción en el modelo de aprendizaje.

A continuación, cada muestra fue normalizada individualmente utilizando un `Standard Scaler`, con el objetivo de estandarizar las características de entrada y reducir los efectos de escalas de medición dispares entre las variables. Esta normalización consiste en transformar cada característica x mediante la ecuación (4.11).

$$x' = \frac{x - \mu}{\sigma} \tag{4.11}$$

donde μ representa la media y σ la desviación estándar de dicha característica. Este procedimiento asegura que todas las variables presenten una distribución con media cero y desviación estándar unitaria, lo cual es especialmente relevante en redes neuronales, ya que evita que ciertas características dominen sobre otras debido a diferencias de escala. Además, mejora la estabilidad numérica durante el entrenamiento y facilita la convergencia del modelo.

Finalmente, el conjunto de datos resultante fue dividido en dos subconjuntos de entrenamiento y prueba, siguiendo una proporción 80/20, mediante una partición aleatoria. Las entradas al modelo se estructuraron como tensores tridimensionales $(n_m, 5000, 44)$, donde 5000 representa el número máximo de pasos temporales y 44 corresponde al número de características por instante de tiempo.

Este flujo de procesamiento de datos permite que el modelo reciba secuencias temporales normalizadas y de longitud uniforme, optimizando tanto el rendimiento como la fiabilidad del entrenamiento.

4.6. Comparativa de modelos evaluados

Durante el desarrollo del trabajo se exploraron diferentes arquitecturas de redes neuronales con el objetivo de seleccionar el modelo más adecuado para la clasificación binaria entre sujetos con y sin Parkinson. En esta sección se comparan los tres enfoques arquitectónicos evaluados: redes LSTM unidireccionales, redes Bi-LSTM bidireccionales y un modelo basado en Transformer, presentando tanto su estructura como su rendimiento en términos de precisión y pérdida.

Las redes LSTM y Bi-LSTM ya han sido descritas en detalle en las secciones 4.3 y 4.4, respectivamente, donde se analizan sus fundamentos teóricos y ventajas para el modelado de secuencias con dependencias temporales. En este apartado, por tanto, se retoma su uso en el

contexto aplicado y se justifica la incorporación de una tercera arquitectura.

Modelo LSTM. Este modelo se construye a partir de una arquitectura secuencial que incluye dos capas LSTM unidireccionales. La primera capa contiene 128 unidades y está configurada con `return_sequences=True`, lo que permite que su salida sea secuencial y pueda ser alimentada a una segunda capa LSTM de 32 unidades. A cada capa se le aplica regularización `L2` para mitigar el sobreajuste, así como normalización por lotes y `Dropout` con tasas de 0.5 y 0.3, respectivamente. El bloque final consta de una capa densa intermedia con 8 neuronas y activación `ReLU`, seguida de una capa de salida con activación sigmoide. Esta configuración permite capturar patrones temporales unidireccionales, siendo más eficiente computacionalmente que las redes bidireccionales, pero limitada en su capacidad para modelar contextos simétricos o con dependencias temporales más amplias.

Modelo Bi-LSTM. La arquitectura del modelo utilizado se describe en detalle en la Sección 4.5.1.

Modelo Transformer. Con el auge de las arquitecturas Transformer [8] en el procesamiento de lenguaje natural y visión por computador, se decidió experimentar también con una versión simplificada de esta arquitectura para el análisis de series temporales. El modelo Transformer utilizado está compuesto por un bloque de atención multi-cabeza (`MultiHeadAttention`) con 4 cabezas y una dimensión de proyección de 128, seguido de un bloque de alimentación hacia adelante (`feed-forward`) también de dimensión 128. La entrada al modelo es una secuencia tridimensional $(n_m, 5000, 44)$, que primero se proyecta a 128 dimensiones mediante una capa densa, antes de ser procesada por el bloque de atención. Tras esto, se aplica `GlobalAveragePooling1D` para resumir la secuencia, seguido de `Dropout`, una capa densa con activación `ReLU` y una salida `sigmoide`.

La capa `GlobalAveragePooling1D`[4] constituye una operación de reducción de dimensionalidad que se emplea habitualmente en modelos diseñados para procesar secuencias temporales. Su principal objetivo es condensar la información contenida en una secuencia, calculando el promedio de cada característica a lo largo de todos los pasos temporales. Dada una entrada con forma (`batch_size`, `timesteps`, `features`), esta capa produce una salida de forma (`batch_size`, `features`), donde cada componente representa la media de una característica específica a lo largo del tiempo. Esta transformación convierte una secuencia de longitud variable en un vector de tamaño fijo, lo cual resulta especialmente útil como paso intermedio antes de aplicar capas densas para tareas de clasificación o regresión. Su uso permite reducir considerablemente el número de parámetros del modelo, disminuyendo así la complejidad computacional y el riesgo de sobreajuste, sin perder la información representativa esencial contenida en la secuencia original.

A diferencia de las redes LSTM, el Transformer no procesa secuencias de forma recurren-

[4]Una descripción más detallada de cómo funciona esta capa se encuentra en https://keras.io/api/layers/pooling_layers/global_average_pooling1d/

te, sino que utiliza mecanismos de atención que permiten acceder directamente a cualquier punto de la secuencia, con independencia de su posición. Esta propiedad ofrece una gran capacidad para modelar dependencias de largo alcance y facilita el paralelismo durante el entrenamiento. Su inclusión en este trabajo responde a su creciente adopción como estándar en tareas secuenciales complejas.

Comparativa de resultados. En la Tabla 4.2 se resume la comparación entre las tres arquitecturas evaluadas. Cada modelo fue entrenado bajo condiciones similares utilizando el mismo conjunto de datos, preprocesado y dividido con una proporción 80/20 para entrenamiento y validación. Se utilizaron funciones de pérdida tipo `binary_crossentropy` y el optimizador `Adam`, manteniendo coherencia experimental entre las pruebas para una comparación justa.

Tabla 4.2: Comparación de los modelos evaluados con sus respectivas configuraciones y métricas.

Modelo	**Arquitectura**	**Precisión**	**Pérdida**
Bi-LSTM	2 capas Bi-LSTM (128, 32) + `BatchNorm` + `Dropout` (0,5; 0,3) + Dense (8, `ReLU`) + Salida `Sigmoide`	0,875	0,653
LSTM	2 capas LSTM (128, 32) + `BatchNorm` + `Dropout` (0,5; 0,3) + Dense (8, `ReLU`) + Salida `Sigmoide`	0,8125	0,680
Transformer	Bloque Multi-Head Attention (4 cabezas, 128 dim) + `Pooling` + `Dropout` (0,5) + Dense (32, `ReLU`) + Salida `Sigmoide`	0,750	0,669

Los resultados indican que el modelo Bi-LSTM obtiene el mejor rendimiento tanto en precisión como en pérdida, superando al LSTM unidireccional y al Transformer. La superioridad de la arquitectura bidireccional puede atribuirse a su capacidad para incorporar información contextual más amplia en ambos sentidos temporales, lo cual resulta particularmente valioso en tareas de análisis de movimiento humano. Si bien el Transformer ofrece ventajas teóricas en el tratamiento de secuencias largas y en la paralelización, su rendimiento en este escenario específico fue inferior.

También es necesario comentar los tiempos de entrenamiento de cada modelo a lo largo de un máximo de 50 *epochs* para estimar su coste computacional. El modelo Bi-LSTM requirió aproximadamente 852,04 segundos, el modelo LSTM alrededor de 563,93 segundos y el Transformer tan solo 24,81 segundos. Sin embargo, es importante señalar que este último detuvo su entrenamiento tras completar 10 *epochs* (al no observar mejoras adicionales), por lo que se estima que, de haber alcanzado los 50 *epochs*, su tiempo de entrenamiento se hubiera aproximado a los 124 segundos.

Esta comparación destaca la mayor complejidad computacional de los modelos basados en redes recurrentes (Bi-LSTM y LSTM), mientras que la arquitectura Transformer mostró una notable rapidez en su convergencia. No obstante, la diferencia de rendimiento final y la precisión obtenida justifican la elección del modelo Bi-LSTM como arquitectura final, priorizando su capacidad de aprendizaje y generalización sobre la velocidad de entrenamiento.

Conclusión de esta comparativa. La elección final del modelo Bi-LSTM se justifica no sólo por sus mejores resultados empíricos, sino también por su idoneidad para procesar datos secuenciales como los analizados en este trabajo. Aunque su tiempo de entrenamiento fue el más alto de los modelos evaluados (852,04 segundos para 50 *epochs*), esta inversión de tiempo se ve compensada por su rendimiento superior. Por su parte, el modelo LSTM demostró un tiempo de entrenamiento intermedio (563,93 segundos), mientras que el modelo Transformer, aunque extremadamente rápido (24,81 segundos en 10 *epochs* y aproximadamente 124 segundos estimados para 50 *epochs*), no alcanzó la misma precisión. Este hallazgo sugiere que, si bien el Transformer destaca por su eficiencia, aún requiere ajustes específicos para mejorar su rendimiento en este dominio. Estos resultados abren la puerta a futuras investigaciones, especialmente si se dispone de un mayor volumen de datos o se exploran técnicas como la atención jerárquica o el preentrenamiento.

4.7. Evaluación de resultados

Para validar el rendimiento del modelo final más allá de su precisión global, se llevó a cabo una evaluación detallada utilizando un conjunto de datos reservado para pruebas (20 % del total). Esta evaluación se realizó mediante el uso de diversas métricas estándar en clasificación supervisada, como la **precisión**, ***recall***, la puntuación **F1** y la **matriz de confusión**.

La matriz de confusión es una herramienta fundamental para analizar el comportamiento de un modelo de clasificación, ya que permite identificar de forma detallada las predicciones correctas e incorrectas. En su estructura, cada fila representa las instancias reales de una clase, mientras que cada columna muestra las predicciones del modelo para esa clase. De este modo, se pueden detectar errores específicos, como confusiones entre clases con características similares o solapadas.

En un problema de clasificación binaria como el que nos ocupa (diferenciar sujetos con Parkinson de sujetos sanos), la matriz de confusión se organiza típicamente de la siguiente manera:

	Predicho: Positivo	**Predicho: Negativo**
Real: Positivo	Verdadero Positivo (TP)	Falso Negativo (FN)
Real: Negativo	Falso Positivo (FP)	Verdadero Negativo (TN)

En la Figura 4.5 se muestra la matriz de confusión obtenida con el conjunto de datos de prueba, lo que permite visualizar la distribución de aciertos y errores en la predicción.

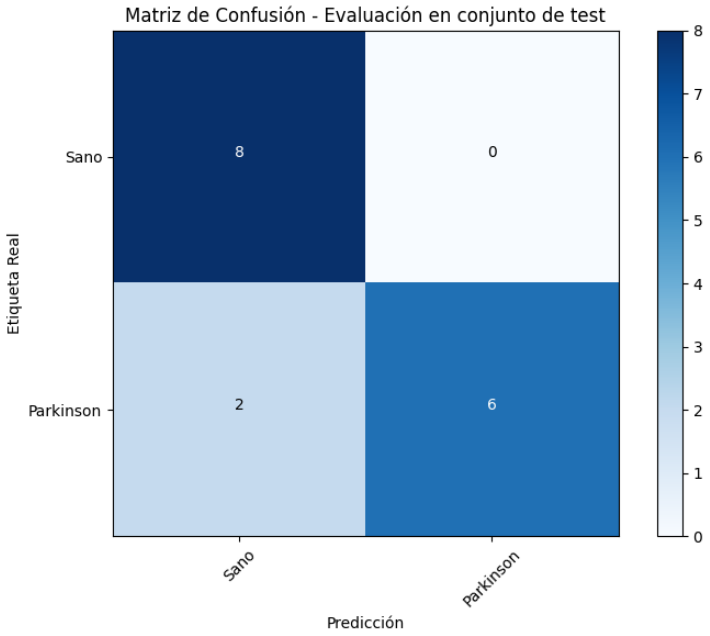

Figura 4.5: Matriz de confusión obtenida sobre el conjunto de test (20 % de los datos).

A partir de esta estructura se derivan las métricas más relevantes:

- *Precision*: proporción de las predicciones positivas que son realmente positivas. Se calcula como $\frac{TP}{TP+FP}$.
- *Recall* (sensibilidad): proporción de instancias positivas correctamente identificadas. Se calcula como $\frac{TP}{TP+FN}$.
- *Specificity* (especificidad): proporción de instancias negativas correctamente identificadas. Se calcula como $\frac{TN}{TN+FP}$.
- *Accuracy* (exactitud): proporción total de predicciones correctas.
  Se calcula como $\frac{TP+TN}{TP+TN+FP+FN}$.
- *F1-score*: media armónica entre *precision* y *recall*, útil en casos de clases desbalanceadas.

En la Tabla 4.3 se presenta el informe de clasificación, donde se muestran las métricas de rendimiento calculadas para cada clase, considerando cada clase como positiva en cada fila (es decir, con un enfoque uno-contra-resto). Las métricas que se muestran –precisión, sensibilidad y *F1-Score*– se derivan de la matriz de confusión.

Tabla 4.3: Informe de clasificación: métricas por clase.

Clase	**Precisión**	**Sensibilidad**	**F1-Score**	**Soporte**
Sano	0,80	1,00	0,89	8
Parkinson	1,00	0,75	0,86	8
Accuracy	0,85			

Además, esta evaluación cuantitativa se completa con un análisis detallado de un subconjunto específico de sujetos del estudio original de 2018 (listados en la Tabla 4.4). Se utilizan los datos recogidos ese año, cuando todos los sujetos estaban sanos, para evaluar si el modelo es capaz de identificar posibles casos de riesgo. Para el año 2024, tres de estos sujetos (MZB, FCM e IOC) habían sido diagnosticados con la enfermedad de Parkinson, lo que convierte a este análisis en una oportunidad interesante para validar la utilidad del modelo como herramienta de cribado temprano.

Tabla 4.4: Probabilidades medias y predicciones de los sujetos del experimento de 2018.

Sujeto	**Probabilidad Media**	**Predicción**	**Etiqueta Real**
ACS	0,6554	Sano	Sano
APP	0,6633	Sano	Sano
ASD	0,6803	Sano	Sano
CAR	0,6069	Sano	Sano
CBC	0,6349	Sano	Sano
CGG	0,7058	Parkinson	Sano
COR	0,6441	Sano	Sano
ESSC	0,5962	Sano	Sano
EVCS	0,6199	Sano	Sano
FCM	0,6596	Sano	Parkinson
FER	0,6810	Sano	Sano
IOC	0,6401	Sano	Parkinson
JGG	0,6388	Sano	Sano
JGLL	0,6268	Sano	Sano
JHA	0,6538	Sano	Sano
JLSC	0,6937	Sano	Sano
LCS	0,6249	Sano	Sano
LOC	0,6136	Sano	Sano
LSC	0,6184	Sano	Sano
MARR	0,6491	Sano	Sano
MCF	0,6360	Sano	Sano
MNP	0,5998	Sano	Sano
MZB	0,5966	Sano	Parkinson
NFC	0,6361	Sano	Sano
ROC	0,6773	Sano	Sano
RRSE	0,6642	Sano	Sano
TAR	0,6218	Sano	Sano

Para la realización de este estudio, interesa especialmente el valor de la probabilidad media por sujeto, ya que proporciona una indicación más continua de la certeza del modelo y permite

comparar entre sujetos sanos y aquellos que posteriormente han desarrollado la enfermedad.

Con el fin de minimizar el riesgo de falsos positivos en la clasificación de sujetos sanos, se ha decidido emplear un umbral de decisión más conservador para esta evaluación. En lugar de utilizar el umbral estándar de 0.5 (que suele maximizar la sensibilidad en detrimento de la especificidad), se ha establecido un umbral de 0.7 para considerar una predicción como positiva (Parkinson). Esta decisión se justifica en el contexto clínico de cribado, donde se prioriza reducir la probabilidad de clasificar erróneamente a un sujeto sano como enfermo.

4.7.1. Análisis de resultados

En la Tabla 4.4 se presentan los resultados de las probabilidades medias de predicción obtenidas por cada uno de los 27 sujetos. Se incluye la predicción binaria generada a partir del umbral (0,7) y la etiqueta real retrospectiva (Sano o Parkinson). De los 27 sujetos analizados, se observa que:

- 1 sujeto (CGG) supera el umbral de 0,7, lo que lo clasificaría como Parkinson a pesar de estar clínicamente sano en 2018 y 2024. Este puede considerarse un falso positivo.
- 18 sujetos superan el umbral del 0,6, lo que sugiere que un porcentaje significativo de individuos, aunque sanos, presentan una probabilidad relativamente alta de ser clasificados como Parkinson. Esto sugiere que, aunque no se consideren casos positivos según el criterio actual, podrían ser objeto de seguimiento clínico para detectar tempranamente cualquier evolución futura hacia la enfermedad.
- Los tres sujetos que desarrollaron Parkinson (MZB, FCM e IOC) presentan probabilidades medias en torno al 0,60–0,66.

En este contexto, resulta relevante destacar que ninguno de estos tres sujetos hubiera sido clasificado como Parkinson en 2018 usando el umbral utilizado. Sin embargo, sus probabilidades medias no son especialmente bajas, situándose en el rango intermedio-alto de la distribución.

4.7.2. Discusión y posibles conclusiones

El análisis sugiere que el modelo podría utilizarse como herramienta de cribado preliminar, especialmente para aquellos sujetos que presenten probabilidades medias superiores al 0.6. Este umbral podría considerarse un indicio de posible riesgo, sugiriendo un seguimiento clínico más cercano a lo largo del tiempo.

Aunque los sujetos MZB, FCM e IOC no alcanzaron la clasificación automática como Parkinson, sus probabilidades medias se sitúan en un rango que podría justificar un seguimiento adicional.

En conjunto, estos resultados muestran el potencial del modelo como herramienta de apoyo a la investigación clínica, si bien se requieren ajustes y estudios longitudinales para confirmar su utilidad predictiva. Una estrategia futura podría consistir en ajustar el umbral de decisión en función del riesgo que se desee cubrir, equilibrando la sensibilidad y la especificidad del sistema.

Capítulo 5

Diseño e implementación de la aplicación

Este capítulo describe el proceso de desarrollo de una aplicación interactiva destinada a facilitar el uso del modelo de clasificación previamente entrenado. El objetivo principal de dicha aplicación es ofrecer una herramienta práctica que permita cargar datos experimentales, procesarlos automáticamente y obtener una predicción sobre la condición del sujeto evaluado. Para ello, se ha diseñado una arquitectura modular que separa los componentes gráficos, de procesamiento y de inferencia, garantizando así la escalabilidad y mantenibilidad del sistema. A lo largo de este capítulo se detallan la estructura interna de la aplicación, la interfaz gráfica, el flujo automatizado de procesamiento de datos, la ejecución del modelo y las decisiones de diseño fundamentadas en principios de ingeniería del software.

5.1. Objetivo de la aplicación

La aplicación desarrollada en el marco de este trabajo tiene como finalidad proporcionar una herramienta funcional que permita utilizar el modelo entrenado de clasificación binaria de sujetos con o sin enfermedad de Parkinson en un entorno accesible, sin necesidad de intervención técnica adicional. Esta herramienta fue concebida con la intención de facilitar la evaluación de nuevos datos obtenidos a partir de sensores inerciales, automatizando las tareas de preprocesamiento, inferencia y visualización de resultados.

Desde el punto de vista funcional, el propósito principal de la aplicación es permitir que un usuario final, sin conocimientos avanzados en programación ni aprendizaje automático, pueda cargar una carpeta con datos recogidos durante la realización de tareas motoras, ejecutar el

procesamiento de forma automática y obtener una predicción clara del estado de salud del sujeto analizado.

La aplicación integra todo el proceso implementado durante el proyecto, incluyendo:

- La selección de carpetas que contienen los datos en bruto recogidos por los sensores.
- La fusión y limpieza de dichos datos, generando un archivo `AllData.csv` por experimento y concatenando en un solo fichero los archivos por sujeto.
- La predicción mediante el modelo Bi-LSTM entrenado previamente.
- La presentación del resultado junto con información adicional como la probabilidad media de la predicción y una representación gráfica intuitiva.

El diseño de la interfaz se ha planteado siguiendo criterios de claridad, simplicidad de uso y modularidad, en línea con los principios propuestos en el libro *Software Engineering* [5]. El sistema se compone de varios módulos diferenciados según su responsabilidad: interfaz de usuario, procesamiento de datos y gestión del modelo. Esta organización permite mejorar la mantenibilidad.

Desde el punto de vista metodológico, la implementación ha sido orientada a encapsular toda la lógica crítica del procesamiento y del modelo en archivos independientes del código de la interfaz, de modo que el núcleo de la aplicación pueda mantenerse o adaptarse sin necesidad de modificar la lógica gráfica. Esto también facilita la validación de los módulos por separado, y permite realizar pruebas unitarias más específicas.

En cuanto al entorno de ejecución, la aplicación ha sido desarrollada en Python 3.10, utilizando las bibliotecas `tkinter`, `pandas`, `numpy`, `joblib` y `keras`. El modelo ha sido entrenado previamente en `TensorFlow` y exportado en formato `keras` para ser cargado en tiempo de ejecución. El preprocesamiento incluye una normalización con un `StandardScaler` previamente ajustado y almacenado con `joblib`, garantizando que los datos introducidos en el modelo sigan la misma distribución estadística que los datos con los que fue entrenado.

La lógica interna del sistema permite detectar automáticamente si los datos cargados son válidos, asegurando que todos los archivos necesarios están presentes y que el número de características es el esperado (44 por instante temporal). Asimismo, se ha incluido un control de errores para mostrar advertencias si los archivos `AllData.csv` generados presentan una dimensión inesperada, lo cual permite identificar de forma temprana errores en el origen de los datos.

En la Figura 5.1 se muestra una vista general de la interfaz de usuario desarrollada, en la que se aprecia la estructura modular y las funcionalidades de carga de datos, predicción y visualización de resultados.

Figura 5.1: Interfaz usada para determinar si un sujeto está sano o tiene Parkinson.

Con esta solución, el modelo desarrollado a lo largo del trabajo se transforma en una herramienta práctica y accesible, con potencial de ser utilizada como apoyo a la investigación o en entornos clínicos experimentales. La experiencia de uso se ve mejorada mediante elementos visuales como la visualización de imágenes que representan un resultado positivo o negativo, así como una sección donde se detallan las probabilidades obtenidas por cada segmento de los datos.

5.2. Requisitos de la aplicación

En este apartado se recogen los requisitos funcionales y no funcionales que guían el desarrollo de la aplicación de escritorio diseñada para la predicción del estado de salud de sujetos a partir de datos de sensores inerciales.

5.2.1. Requisitos funcionales

Los requisitos funcionales definen las acciones que el sistema debe ser capaz de realizar:

- **RF1.** Permitir al usuario seleccionar una carpeta que contenga los datos de un sujeto,

organizados en subdirectorios con las tareas motoras correspondientes.

- **RF2.** Procesar los archivos `AllData.csv` de cada tarea, fusionándolos en un único archivo por sujeto y asegurando que cumplen con el número de características esperado.
- **RF3.** Cargar el modelo Bi-LSTM previamente entrenado y su `StandardScaler` asociado para garantizar la coherencia en la normalización.
- **RF4.** Ejecutar la predicción del estado de salud sobre los datos del sujeto.
- **RF5.** Mostrar el resultado de la clasificación de manera clara e intuitiva en la interfaz, junto con la probabilidad media y las probabilidades por segmento.
- **RF6.** Gestionar y mostrar mensajes de error si se detectan problemas de formato, estructura o integridad en los datos de entrada.
- **RF7.** Ofrecer al usuario una navegación guiada, donde cada acción habilita las siguientes de forma progresiva.

5.2.2. Requisitos no funcionales

Los requisitos no funcionales establecen criterios de calidad y limitaciones técnicas de la aplicación:

- **RNF1.** La aplicación debe estar implementada en Python 3.10 y ser compatible con el sistema operativo Windows.
- **RNF2.** El procesamiento y la predicción deben ejecutarse en un tiempo razonable (menos de 5 segundos en un equipo estándar) para garantizar la fluidez de uso.
- **RNF3.** La arquitectura del sistema debe estar modularizada, separando la interfaz gráfica, el procesamiento de datos y el manejo del modelo, facilitando así la mantenibilidad y escalabilidad.
- **RNF4.** Los resultados deben presentarse con claridad, utilizando colores diferenciados e iconos para reforzar el diagnóstico.
- **RNF5.** El código debe seguir buenas prácticas de programación, incluyendo comentarios explicativos y una estructura que permita realizar pruebas unitarias sobre cada módulo.

Con el fin de ilustrar la estructura modular de la aplicación, se incluye en el presente capítulo un **diagrama de clases** (UML) que detalla las relaciones entre los componentes desarrollados.

En la Figura 5.2 se presenta el diagrama de clases de la aplicación, en el que se destacan los módulos principales, sus responsabilidades y sus relaciones de colaboración. Este diagrama facilita la comprensión de la estructura modular del sistema y respalda el cumplimiento de los requisitos de mantenibilidad y escalabilidad mencionados anteriormente.

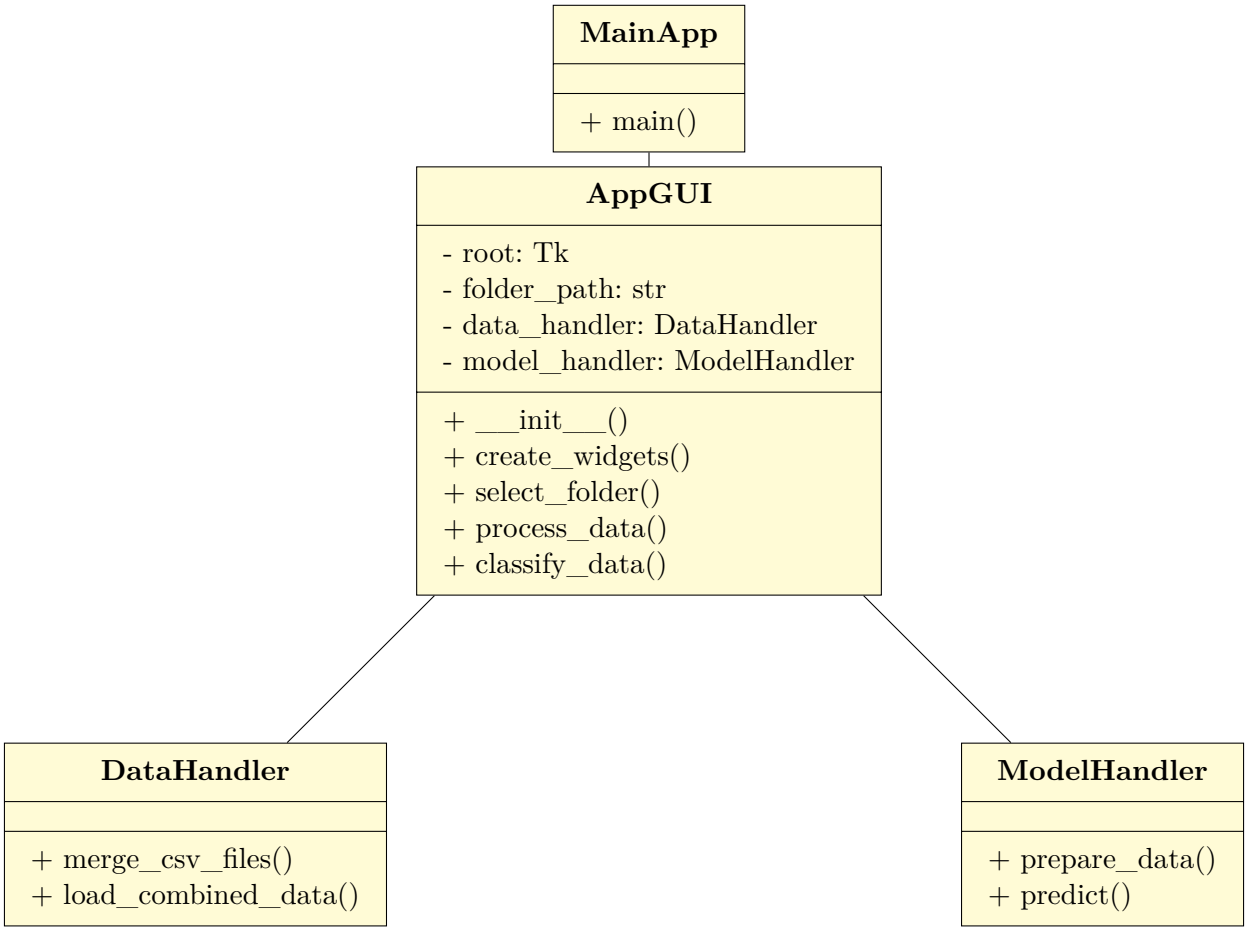

Figura 5.2: Diagrama UML simplificado que muestra las clases principales de la aplicación y sus relaciones.

5.3. Diseño de arquitectura

Para favorecer la mantenibilidad, escalabilidad y claridad del código, se optó por una estructura modular en el desarrollo de la aplicación. Esta estrategia se basa en la división del sistema en componentes independientes que se encargan de tareas específicas, permitiendo una separación clara de responsabilidades y una mayor facilidad a la hora de realizar

modificaciones o ampliaciones.

El sistema se estructura en cuatro archivos principales:

- `main.py`: contiene el punto de entrada a la aplicación. Su única responsabilidad es iniciar la interfaz gráfica, permitiendo mantener el control centralizado.
- `app_gui.py`: gestiona todos los elementos gráficos mediante la biblioteca `Tkinter`. Aquí se define el layout de la ventana principal, los botones de interacción, las etiquetas informativas y los eventos asociados a cada acción del usuario.
- `data_handler.py`: encapsula todas las funciones relacionadas con la carga, validación, fusión y normalización de los datos. Esto incluye tanto la lectura de archivos `.csv` individuales como la construcción del archivo `AllData.csv` final y la preparación de los datos para el modelo.
- `model_handler.py`: se encarga de la carga del modelo previamente entrenado en formato `.keras`, así como de la segmentación de las secuencias de entrada, la ejecución de las predicciones y el cálculo del resultado global.

En la Figura 5.3 se observa la organización de los módulos de manera visual.

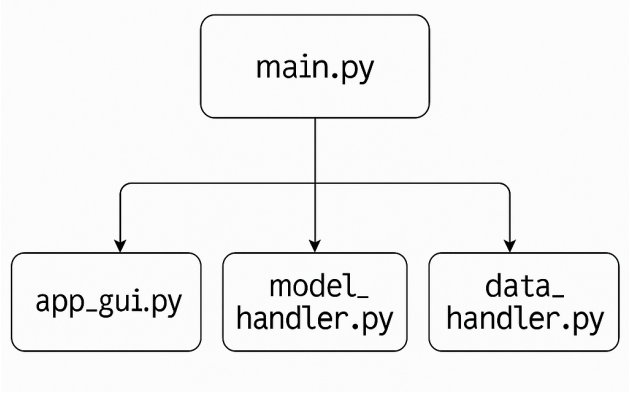

Figura 5.3: Módulos usados en la interfaz.

Esta división del sistema permite trabajar de forma independiente en cada una de las partes, facilitando tanto las pruebas unitarias como la comprensión del flujo de trabajo. Por ejemplo, un cambio en el modelo puede llevarse a cabo simplemente actualizando el archivo en `model_handler.py` sin necesidad de alterar la lógica de la interfaz. Asimismo, cualquier mejora en la estrategia de procesamiento de datos puede implementarse directamente en `data_handler.py`, manteniendo el resto del sistema inalterado.

El uso de clases en cada uno de los módulos también contribuye a encapsular el estado interno y a facilitar la reutilización del código. Por ejemplo, el objeto principal de la clase encargada de

gestionar el modelo mantiene cargado en memoria el modelo neuronal y el `scaler`, evitando repetir la operación para cada predicción. Esta optimización permite mejorar el rendimiento y reducir el tiempo de espera para el usuario.

Además, esta arquitectura modular ha permitido implementar mecanismos de control de errores de forma localizada. Por ejemplo, si durante el preprocesamiento se detecta que un archivo `AllData.csv` contiene un número de columnas diferente al esperado (por ejemplo, 43 en lugar de 44), se lanza una advertencia sin interrumpir el resto de la ejecución, permitiendo al usuario decidir si continuar o revisar los datos.

Otra ventaja destacable de esta estructura es que facilita futuras extensiones del sistema. Por ejemplo, si se desea añadir otro modelo para comparar resultados, bastaría con modificar el archivo `model_handler.py` para cargar otro modelo distinto. Asimismo, si en versiones futuras se desea reemplazar la interfaz gráfica por una interfaz web, se podría mantener el procesamiento y la lógica de predicción intactas, reemplazando únicamente el módulo de `app_gui.py`.

En definitiva, el diseño modular seguido en esta aplicación no solo responde a principios de buena práctica en ingeniería del software, sino que ha demostrado ser una elección acertada para facilitar la evolución y robustez del sistema. En los siguientes apartados se describen con más detalle los módulos gráficos, de procesamiento y predicción.

5.4. Interfaz gráfica con `tkinter`

Con el objetivo de facilitar la interacción con el sistema, se desarrolló una interfaz gráfica de usuario utilizando la biblioteca estándar `Tkinter` de Python. Esta herramienta permite crear aplicaciones de escritorio multiplataforma con un diseño funcional y ligero, sin necesidad de instalar librerías externas adicionales. La elección de `Tkinter` respondió tanto a criterios de simplicidad como de compatibilidad, garantizando que el sistema pudiera ejecutarse fácilmente en Windows.

La interfaz se diseñó en forma de ventana principal dividida en tres zonas funcionales:

- Un panel lateral izquierdo destinado a las utilidades principales: selección de carpeta, procesamiento de datos y clasificación.
- Un área central para mostrar la carpeta seleccionada, el resultado de la clasificación y una imagen representativa asociada.
- Un panel derecho con información adicional como los valores reales de salida o las probabilidades obtenidas por segmento.

El diseño se centró en la simplicidad de uso, de manera que el usuario solo debe realizar tres acciones para obtener una predicción:

1. Seleccionar una carpeta que contenga los datos de un sujeto con las tres tareas motoras.
2. Procesar automáticamente los datos para generar los archivos `AllData.csv`.
3. Ejecutar la clasificación mediante el modelo entrenado y visualizar el resultado.

Figura 5.4: Secuencia de la pulsación de botones para obtener el resultado.

Cada uno de estos pasos está asociado a un botón con un texto descriptivo, y las acciones se encuentran deshabilitadas hasta que el paso anterior se haya completado correctamente, evitando así posibles errores de ejecución. Por ejemplo, el botón de clasificación permanece inactivo hasta que se haya ejecutado el procesamiento de datos con éxito.

Los elementos visuales incluyen un mensaje con el diagnóstico (“Sano” o “Parkinson”) que aparece tras la clasificación, y una imagen que refuerza la interpretación de manera intuitiva (por ejemplo, un icono de validación o advertencia). Esta funcionalidad mejora la experiencia del usuario y facilita la comprensión del resultado sin necesidad de analizar cifras o métricas técnicas.

Asimismo, en el panel de la derecha se muestran métricas adicionales como las probabilidades obtenidas por cada segmento de datos y el valor promedio final antes de ser redondeado. Esta información resulta útil tanto para fines de depuración como para interpretar mejor el grado de certeza del modelo en cada clasificación.

Todos los elementos visuales, incluidas las fuentes y estilos, se configuraron para proporcionar una experiencia visual homogénea, utilizando tipografía tipo `Arial` cuando estuvo disponible para dar un estilo técnico y moderno a la aplicación.

Desde el punto de vista de implementación, los eventos de los botones están vinculados a funciones definidas en los módulos externos, respetando así el principio de separación de responsabilidades. Por ejemplo, al pulsar el botón "Procesar datos", la interfaz invoca una función del módulo `data_handler.py` que se encarga de recorrer las carpetas, fusionar los archivos y generar los archivos `AllData.csv`. Del mismo modo, al pulsar "Clasificar datos", la aplicación recupera la información mediante el módulo `model_handler.py`, carga el modelo previamente entrenado, transforma los datos y realiza la predicción.

Además, se añadieron cuadros de diálogo para mostrar mensajes de error o advertencia cuando se detectan inconsistencias en los datos (por ejemplo, si un archivo contiene menos columnas de las esperadas o si la carpeta seleccionada no tiene la estructura adecuada). Estos mensajes son gestionados mediante funciones estándar de `Tkinter`, lo que permite informar al usuario sin interrumpir el flujo normal de ejecución.

En términos de accesibilidad y portabilidad, el uso de una interfaz de escritorio local garantiza que la aplicación puede ejecutarse en equipos sin conexión a Internet ni dependencia de plataformas externas, lo que resulta especialmente útil en contextos clínicos o de laboratorio donde se manejan datos sensibles.

En conclusión, la interfaz gráfica diseñada con `Tkinter` proporciona una solución eficiente y amigable para interactuar con el modelo desarrollado, facilitando tanto el procesamiento de nuevos datos como la interpretación de los resultados. Esta herramienta constituye un puente entre el modelo técnico entrenado y el usuario final, integrando los componentes de forma intuitiva y funcional.

5.5. Pruebas realizadas

Con el objetivo de garantizar la calidad, la estabilidad y la correcta funcionalidad del sistema desarrollado, se llevaron a cabo diferentes tipos de pruebas a lo largo de todo el ciclo de vida del proyecto. Cada una de ellas se diseñó para cubrir aspectos específicos de la aplicación y verificar su correcto comportamiento, desde la validación de componentes individuales hasta la integración de módulos y el manejo de errores en la interfaz gráfica.

5.5.1. Pruebas unitarias

Se diseñaron pruebas unitarias para el módulo de interfaz gráfica de la aplicación (`app_gui.py`), utilizando el framework `unittest` de Python. Estas pruebas tienen como objetivo garantizar el correcto funcionamiento de cada método y componente visual de forma aislada, sin depender de otros módulos externos o de la lógica de procesamiento de datos.

Las pruebas unitarias incluyeron la verificación de:

- Correcta actualización de las etiquetas y botones de la interfaz tras la selección de una carpeta.
- Habilitación y deshabilitación dinámica de los botones de procesamiento y clasificación según el estado de la aplicación.
- Manejo de excepciones en las funciones críticas, garantizando que los errores se informan adecuadamente al usuario mediante mensajes emergentes.
- Carga de datos simulada y predicción de resultados con técnicas de `mocking`[1], evitando la dependencia de archivos externos.

5.5.2. Pruebas de integración

Además de las pruebas unitarias, se realizaron pruebas de integración centradas en la interacción entre los distintos módulos de la aplicación: la interfaz gráfica, el módulo de procesamiento de datos y el módulo de predicción. Estas se diseñaron para validar que la secuencia completa de acciones (desde la selección de carpeta hasta la obtención de resultados) funciona de forma coherente y sin errores, garantizando una experiencia de usuario fluida.

5.5.3. Pruebas de errores y recuperación

Con el fin de verificar la robustez del sistema frente a situaciones imprevistas, se incluyeron pruebas específicas de manejo de errores. Estas pruebas simularon condiciones de fallo en el procesamiento de datos y en la predicción del modelo, comprobando que el sistema reacciona de forma controlada y presenta mensajes claros al usuario. Asimismo, se aseguró que la aplicación no se bloquee ni se cierre de forma inesperada ante errores internos.

[1] Creación de reemplazos artificiales (*mock objects*) para dependencias externas, como APIs o servicios, que son necesarios para que una unidad de código (función, clase, etc.) funcione correctamente

5.5.4. Cobertura y resultados

En total, se definieron y ejecutaron 9 pruebas unitarias que cubren los distintos métodos críticos de la clase `app_gui.py`. Se implementaron técnicas de simulación de dependencias (mediante `unittest.mock`) para aislar los métodos y verificar su comportamiento de manera independiente.

En el momento de la redacción de este documento, todas las pruebas se ejecutaron con éxito, confirmando el correcto funcionamiento de las funciones de la interfaz gráfica, la gestión de errores y la integración con los módulos de datos y modelo. Estos resultados refuerzan la confianza en la estabilidad y fiabilidad del sistema desarrollado.

5.5.5. Conclusiones de las pruebas

El conjunto de pruebas implementadas ha permitido validar de manera exhaustiva el comportamiento de la aplicación y su correcta integración entre módulos. Se ha confirmado que la interfaz gráfica gestiona adecuadamente las diferentes etapas de procesamiento y predicción, informando al usuario en cada paso y manejando de forma adecuada posibles errores. Gracias a este proceso de verificación, se ha fortalecido la robustez del sistema y se ha sentado la base para futuras ampliaciones y mantenibilidad de la herramienta.

5.6. Prácticas y principios de ingeniería del software

Durante el desarrollo de la aplicación se aplicaron diversos principios fundamentales de la ingeniería del software con el objetivo de garantizar un sistema robusto, mantenible y extensible. Estas decisiones metodológicas están fundamentadas en los enfoques descritos por Sommerville [5], particularmente en lo relativo al diseño modular, la separación de responsabilidades, la reutilización de componentes y la facilidad de mantenimiento.

En primer lugar, se adoptó una arquitectura modular compuesta por archivos independientes, cada uno con una función claramente definida: `app_gui.py` para la interfaz gráfica, `data_handler.py` para el procesamiento de datos, `model_handler.py` para la carga y uso del modelo de clasificación, y `main.py` como punto de entrada de la aplicación. Esta separación de responsabilidades reduce el acoplamiento entre componentes y permite modificar una parte del sistema sin afectar las demás.

Asimismo, cada módulo fue implementado siguiendo el principio de cohesión funcional, es decir, asegurando que todas las funciones y clases dentro de un archivo contribuyen a una

única finalidad. Por ejemplo, todas las funciones relacionadas con el preprocesamiento de datos (fusión de archivos, eliminación de columnas no necesarias, normalización y segmentación) están agrupadas exclusivamente en `data_handler.py`. Esta organización mejora la legibilidad y simplifica el proceso de depuración o ampliación de funcionalidades.

Se utilizó también el principio de reutilización de componentes. Tanto el modelo de red neuronal como el objeto de normalización (`StandardScaler`) fueron guardados como archivos externos y reutilizados en la aplicación, evitando la necesidad de redefinir su comportamiento. Esta reutilización no solo mejora la eficiencia del desarrollo, sino que garantiza la coherencia entre el entorno de entrenamiento y el entorno de inferencia.

Desde el punto de vista del diseño orientado a la interfaz de usuario, se priorizó la simplicidad y claridad funcional. El número de interacciones requeridas al usuario se mantuvo al mínimo, y cada acción está claramente delimitada y guiada por la interfaz, respetando el principio de usabilidad. Además, los botones de acción están habilitados o deshabilitados en función del estado del sistema, evitando errores por uso inadecuado o fuera de orden.

Otro principio importante aplicado fue la validación anticipada de datos. Antes de ejecutar operaciones críticas, como la predicción mediante el modelo, se verifica que las dimensiones y características de los datos sean las esperadas. Esta verificación reduce drásticamente el riesgo de errores en tiempo de ejecución y permite identificar problemas en los datos desde etapas tempranas. En caso de detectar inconsistencias, el sistema proporciona mensajes de advertencia comprensibles para el usuario, mejorando la experiencia general.

En términos de portabilidad y compatibilidad, se optó por bibliotecas estándar del lenguaje Python, como `tkinter` para la interfaz gráfica, `pandas` y `numpy` para el manejo de datos, y `joblib` para la serialización de objetos. Esto garantiza que el sistema pueda ser ejecutado en múltiples plataformas sin dependencias complejas, siguiendo el principio de independencia del entorno. Asimismo, se emplearon rutas relativas para acceder a los modelos y recursos gráficos, permitiendo que la aplicación funcione correctamente sin necesidad de configuraciones adicionales.

A nivel de mantenimiento, el sistema fue documentado internamente mediante comentarios explicativos que acompañan a cada función y clase, facilitando así el trabajo futuro en caso de necesitar ajustes o ampliaciones. Se adoptaron convenciones de nombres descriptivos para las funciones, variables y archivos, mejorando la legibilidad del código y su comprensión por parte de otros desarrolladores.

Finalmente, se tuvieron en cuenta criterios de eficiencia computacional. Por ejemplo, tanto el modelo como el *scaler* se cargan una única vez durante la ejecución y se mantienen en memoria para su reutilización, lo cual reduce los tiempos de espera y mejora el rendimiento.

Capítulo 6

Conclusiones

El presente trabajo ha demostrado la viabilidad de aplicar redes neuronales bidireccionales con memoria a largo y corto plazo para la clasificación binaria de sujetos con y sin enfermedad de Parkinson, a partir de señales temporales generadas durante la realización de tareas motoras controladas. Gracias a la estructura bidireccional, el modelo ha sido capaz de capturar dependencias tanto pasadas como futuras en las series temporales, mejorando la capacidad de predicción frente a enfoques unidireccionales.

Desde el punto de vista de la metodología de desarrollo, se ha seguido una estrategia incremental con prototipado, que ha permitido construir y validar de manera iterativa cada uno de los módulos del sistema: procesamiento de datos, entrenamiento del modelo y desarrollo de la interfaz gráfica. Este enfoque ha resultado especialmente valioso para adaptarse a los datos, así como para realizar ajustes basados en resultados experimentales. La modularidad lograda facilita la mantenibilidad del sistema y abre la puerta a futuras extensiones.

En cuanto a la implementación práctica, se ha desarrollado una herramienta accesible basada en `Tkinter` que permite al usuario cargar carpetas de datos, procesarlas automáticamente, ejecutar la clasificación con el modelo entrenado y visualizar los resultados de forma clara y comprensible. Esta solución constituye una aportación significativa, ya que traduce el modelo técnico en una herramienta que puede ser utilizada en entornos clínicos o de investigación sin necesidad de conocimientos avanzados en programación.

La evaluación retrospectiva de los 27 sujetos del experimento de 2018, todos ellos sanos en el momento de la grabación, ha permitido analizar la capacidad del modelo para detectar posibles indicios tempranos de Parkinson. A través de las probabilidades medias de predicción, se ha identificado que tres sujetos (MZB, FCM e IOC) que desarrollaron la enfermedad posteriormente al estudio del 2018 obtuvieron probabilidades relativamente altas, lo que sugiere que el modelo podría ser útil como herramienta de apoyo para la monitorización de pacientes. Sin embargo, también se han observado probabilidades elevadas en sujetos sanos,

lo que resalta la necesidad de ajustar el umbral de decisión (fijado en este estudio en 0.7) para reducir los falsos positivos y mejorar la especificidad del sistema.

En definitiva, este proyecto ha combinado teoría, desarrollo de software y análisis de datos para diseñar e implementar una solución completa y funcional que permite la detección automática de Parkinson a partir de datos de sensores inerciales. El sistema resultante representa un paso importante hacia la integración de la inteligencia artificial en la investigación biomédica y ofrece una base sólida para desarrollos futuros orientados a la mejora de la precisión y la aplicabilidad clínica.

Bibliografía

[1] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. por Alessandro Moschitti, Bo Pang y Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, oct. de 2014, págs. 1724-1734. DOI: 10.3115/v1/D14-1179. URL: https://aclanthology.org/D14-1179/.

[2] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". En: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06.02 (1998), págs. 107-116. DOI: 10.1142/S0218488598000094. eprint: https://doi.org/10.1142/S0218488598000094. URL: https://doi.org/10.1142/S0218488598000094.

[3] Kevin Clark Minh-Thang Luong Christopher D. Manning Quoc V. Le. *Semi-Supervised Sequence Modeling with Cross-View Training*. 2018. URL: https://arxiv.org/pdf/1809.08370v1 (visitado 22-09-2018).

[4] Ibomoiye Domor Mienye, Theo G. Swart y George Obaido. "Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications". En: *Information* 15.9 (2024). ISSN: 2078-2489. DOI: 10.3390/info15090517. URL: https://www.mdpi.com/2078-2489/15/9/517.

[5] Ian Sommerville. *Software Engineering*. 9th. Boston: Addison-Wesley, 2011. ISBN: 978-0-13-703515-1.

[6] Ralf C. Staudemeyer y Eric Rothstein Morris. "Understanding LSTM a tutorial into Long Short-Term Memory Recurrent Neural Networks". En: *Mathematics* (2019). URL: https://arxiv.org/pdf/1909.09586.pdf.

[7] Cristina Tîrnăucă et al. "A Machine Learning Approach to Detect Parkinson's Disease by Looking at Gait Alterations". En: *Mathematics* 10.19 (2022). ISSN: 2227-7390. DOI: 10.3390/math10193500. URL: https://www.mdpi.com/2227-7390/10/19/3500.

[8] Ashish Vaswani et al. "Attention Is All You Need". En: *Advances in Neural Information Processing Systems* 30 (2017). URL: https://arxiv.org/abs/1706.03762.