TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Implementación de un algoritmo de aprendizaje automático para la detección de actividades en el hogar a partir de sensores

Autor: Francisco Ortega Peña

Grado: Ingeniería en Informática

Directores: José Luis López Ruiz y David Díaz Jiménez
Departamento del director: Informática

Fecha: 18/02/2025

UNIVERSIDAD DE JAÉN

D./Dª José Luis López Ruiz y D./Dª David Díaz Jiménez, tutor(es) del Trabajo Fin de Grado titulado: **Implementación de un algoritmo de aprendizaje automático para la detección de actividades en el hogar a partir de sensores**, que presenta Francisco Ortega Peña, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, 18/02/2025

El estudiante Los tutores

Francisco Ortega Peña José Luis López Ruiz David Díaz Jiménez

Agradecimientos

Este trabajo marca el fin de una etapa llena de aprendizaje, retos y crecimiento personal. Sin embargo, este logro no hubiese sido posible sin el apoyo incondicional de muchas personas que han estado a mi lado a lo largo de este camino.

En primer lugar, quiero agradecer a mi familia, que siempre ha creído en mí incluso en los momentos más difíciles. A mis padres y mi hermano, por ser mi apoyo constante y por sus sacrificios para ayudarme a alcanzar mis metas.

A mis amigos y compañeros de clase, quienes han hecho que estos años sean más llevaderos con su ánimo, risas y consejos. Vuestro apoyo en momentos de estrés y vuestros pequeños gestos han hecho una gran diferencia.

Por último, pero no menos importante, agradecer a mis tutores y compañeros del grupo ASIA por su paciencia, dedicación y disposición para ayudarme en todo momento.

Este proyecto no solo marca el final de una etapa, sino el comienzo de muchas más. A todos los que formaron parte de este camino, gracias por estar ahí.

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de algoritmos

Lista de listados de código

Capítulo 1

INTRODUCCION

Este proyecto se centra en la identificación y **reconocimiento de actividades humanas** en el entorno doméstico mediante el **uso de sensores** y la aplicación de técnicas de **aprendizaje automático**.

1.1. Motivación

El desarrollo de este trabajo surge del interés en mejorar los sistemas de monitorización y análisis de actividades humanas en el hogar. Estos son fundamentales en áreas como la asistencia a personas mayores, la prevención de accidentes domésticos y la gestión de hogares inteligentes. Además, la necesidad de identificar patrones de comportamiento en espacios domésticos de forma eficiente y no intrusiva ha motivado la búsqueda de soluciones mediante herramientas tecnológicas que puedan abordar estos retos de manera efectiva.

Actualmente, la detección de actividades en el hogar depende en muchos casos de métodos invasivos, que no respetan la privacidad y que, además, están limitados en términos de escalabilidad y personalización. Estos enfoques plantean desafíos técnicos y éticos debido a problemas relacionados con la privacidad de las personas residentes. Ante esta problemática, el uso de sensores distribuidos por el hogar ofrece una mejor alternativa, ya que permite recoger información relevante sin comprometer la intimidad de las personas.

Igualmente, la creciente disponibilidad de tecnologías basadas en **IA** brinda la oportunidad de diseñar sistemas más precisos y moldeables. La automatización de la detección de actividades mediante sensores y modelos avanzados de aprendizaje

automático puede aliviar tareas repetitivas y laboriosas, además de permitir a los especialistas centrarse en otros aspectos importantes. Un ejemplo de esto se da en el ámbito de la salud, mediante el uso de sensores de movimiento aplicados al cuidado de personas mayores, es posible predecir el riesgo de caídas según la actividad que realizan, lo que permite a los profesionales intervenir de manera más efectiva, centrando su atención en tareas de mayor importancia [18].

Desde el ámbito de la investigación en Inteligencia Artificial (IA), con este proyecto busco contribuir al desarrollo de sistemas que combinen la precisión del *Machine Learning* (ML) con la eficiencia y la flexibilidad de los sensores inteligentes. Una correcta implementación podría mejorar la calidad de vida de los usuarios finales y, además, podría sentar las bases para investigaciones futuras.

En este **Trabajo Fin de Grado**(TFG), se procesarán los datos obtenidos a partir de sensores distribuidos en un entorno doméstico con el objetivo de clasificar las actividades humanas realizadas en dicho espacio. A través del análisis de estos datos, se emplearán técnicas avanzadas de aprendizaje automático para detectar y etiquetar automáticamente las actividades realizadas, como dormir, comer, hacer ejercicio, etc.

1.2. Estructura del documento

La **estructura** de esta memoria consta de seis capítulos organizados de la siguiente manera:

1. **Introducción**
   En este capítulo (capítulo 1), se realiza una **introducción del trabajo** donde se presenta el objetivo principal del proyecto y se expone la motivación detrás del estudio, resaltando que muchos métodos actuales son invasivos y comprometen la privacidad, mientras que el uso de sensores distribuidos es una alternativa más simple y eficiente. Además, se habla sobre el potencial de la Inteligencia Artificial para automatizar el proceso de detección de actividades.

   Por último, se presenta la estructura del documento y se incluye una lista de acrónimos y términos relevantes para facilitar la lectura del trabajo.

2. **Fundamentos y antecedentes**
   El capítulo 2 recopila conceptos clave necesarios para comprender el caso de estudio, incluyendo una **revisión del estado del arte**. Se abordan desde los principios de la Inteligencia Artificial, el Aprendizaje Automático y el Aprendizaje

Profundo hasta su aplicación en el Reconocimiento de Actividades Humanas, analizando las técnicas más usadas en este campo.

3. **Objetivos**
   En el capítulo 3 se exponen los **objetivos** que guían el desarrollo del trabajo, primero se presenta el objetivo principal que, más tarde, se descompone en un conjunto de objetivos específicos.

4. **Materiales y métodos**
   En el capítulo 4 se describen los **materiales y métodos** utilizados para alcanzar los objetivos de este trabajo, detallando el entorno de experimentación, las características del hardware utilizado, la selección del conjunto de datos, las técnicas de preprocesamiento aplicadas, el diseño y selección del modelo y los métodos de evaluación y validación para la selección del modelo.

5. **Resultados**
   En el capítulo 5 se muestran los **resultados obtenidos** tras la experimentación de los distintos modelos implementados. Se comparan los modelos en función de métricas de rendimiento y se analizan sus fortalezas y limitaciones mediante gráficas y tablas comparativas. Por último, se muestra el modelo seleccionada y se trata de optimizar sus resultados variando los distintos hiperparámetros.

6. **Conclusiones**
   En el capítulo 6 se exponen las **conclusiones generales del estudio**, evaluando si se han cumplido los objetivos y discutiendo **posibles mejoras futuras**.

1.3. Acrónimos y términos

Para facilitar la lectura del documento, se muestra a continuación una lista con los acrónimos y términos utilizados:

- **TFG**: Trabajo Fin de Grado
- **HAR**: *Human Activity Recognition* (Reconocimiento de Actividades Humanas)
- **IA**: Inteligencia Artificial
- **ML**: *Machine Learning* (Aprendizaje Automático)
- **DL**: *Deep Learning* (Aprendizaje Profundo)

- **RNN**: *Recurrent Neural Network* (Red Neuronal Recurrente)
- **CNN**: *Convolutional Neural Network* (Red Neuronal Convolucional)
- **GRU**: *Gated Recurrent Unit* (Unidad Recurrente Controlada)
- **LSTM**: *Long Short-Term Memory* (Memoria a Largo y Corto Plazo)
- **HMM**: *Hidden Markov Models* (Modelos Ocultos de Markov)
- **SVM**: *Support Vector Machine* (Máquina de Vectores de Soporte)
- **NB**: *Naïve-Bayes*
- **k-NN**: *k-Nearest Neighbors* (K-Vecinos más cercanos)
- **SSL**: *Semi-Supervised Learning* (Aprendizaje Semi-Supervisado)
- **GSSL**: *Graph-based Semi-Supervised Learning* (Aprendizaje Semi-Supervisado basado en grafos)
- **DNN**: *Deep Neuronal Network* (Red Neuronal Profunda)
- **PUN**: *Pretrained Unsupervised Networks* (Redes No Supervisadas Preentrenadas)
- **GAN**: *Generative Adversarial Networks* (Redes Generativas Antagónicas)
- **DBN**: *Deep Belief Networks* (Redes de Creencias Profundas)
- **RFID**: *Radio Frequency Identification* (Identificación por Radiofrecuencia)
- **EEG**: *Electroencephalogram* (Electroencefalograma)
- **ECG**: *Electrocardiogram* (Electrocardiograma)
- **ReLU**: *Rectified Linear Unit* (Unidad Lineal Rectificada)
- **MSE**: *Mean Squared Error* (Error cuadrático medio)
- **MAE**: *Mean Absolute Error* (Error absoluto medio)
- **BCE**: *Binary Cross-Entropy* (Entropía cruzada binaria)
- **CCE**: *Categorical Cross-entropy* (Entropía cruzada categórica)
- **SCCE**: *Sparse Categorical Cross-entropy* (Entropía cruzada categórica dispersa)
- **GPU**: *Graphics Processing Unit* (Unidad de procesamiento gráfico)

- **BLE**: *Bluetooth Low Energy*
- **IDE**: *Integrated Development Environment* (Entorno de Desarrollo Integrado)

Capítulo 2

FUNDAMENTOS Y ANTECEDENTES

Este capítulo tiene como propósito sentar las **bases teóricas** que sustentan el desarrollo de este trabajo, proporcionado un contexto amplio que abarque los principales conceptos y avances relacionados con el *Deep Learning* (DL) y con el Reconocimiento de Actividades Humanas (HAR) mediante sensores.

En primer lugar, se presenta una introducción general a la Inteligencia Artificial, incluyendo su evolución histórica y su relación con el Reconocimiento de Actividades Humanas. A continuación, se profundiza en el Aprendizaje Automático y sus diferentes tipos, destacando su relevancia en el campo de estudio. Más tarde, se aborda el Aprendizaje Profundo y su impacto en la mejora de técnicas aplicadas al HAR, como las redes neuronales y sus variantes. Finalmente, se detalla el papel de los sensores y los avances en el HAR.

2.1. Inteligencia Artificial

Para empezar a ponernos en contexto sobre el problema de este trabajo, es imprescindible comenzar hablando de la Inteligencia Artificial. En primer lugar, se expondrán algunas de sus definiciones más relevantes y haremos un breve repaso de su evolución en la historia. Por último, veremos cómo este campo se conecta con el Reconocimiento de Actividades Humanas.

2.1.1. Definición y evolución del concepto IA

Empezaremos definiendo el término **Inteligencia Artificial**, cuya conceptualización ha ido evolucionando a lo largo de los años debido al rápido avance que ha experimentado este campo. La constante evolución y los diferentes enfoques hacen que resulte complicado establecer una única interpretación. De manera general, la IA se puede entender como el área de la informática dedicada al desarrollo de sistemas capaces de realizar procesos de pensamiento similares a los humanos, como el aprendizaje el razonamiento y la autocorrección [19].

Una definición más concisa y estricta podría describirla como la capacidad de los ordenadores para imitar la inteligencia humana [20].

Según un informe del grupo de expertos de alto nivel sobre IA de la Comisión Europea, la IA se refiere a sistemas que muestran un comportamiento inteligente mediante el análisis de su entorno y la toma de decisiones autónomas para alcanzar objetivos específicos [1].

2.1.2. Orígenes de la IA como campo científico

El desarrollo de la IA como campo formal de estudio comenzó en 1956 con la **Conferencia de Dartmouth** [21], considerada un hito fundamental en la historia de esta disciplina. Organizada por John McCarthy, Marvin Minsky, Claude, Shannon y Nathaniel Rochester, este evento reunió a un grupo de grandes investigadores para discutir y formalizar el concepto de Inteligencia Artificial. La conferencia no solo estableció este término, sino que también estableció las bases para una **nueva área científica** dedicada a explorar cómo las máquinas podían replicar o simular procesos cognitivos humanos [22].

En este mismo año, Alan Newell y Herbert Simon desarrollaron la ***Logic Theory Machine***, el primer programa considerado de IA [23]. Diseñado para demostrar teoremas en lógica simbólica, introdujo un enfoque heurístico que simulaba razonamiento humano al reducir las combinaciones posibles en la búsqueda de soluciones.

2.1.3. Relación entre IA y HAR

Dentro de la IA, el campo del Reconocimiento de Actividades Humanas ha evolucionado significativamente. En un principio, los sistemas HAR utilizaban métodos tradicionales de aprendizaje basado en características manuales extraídas de datos visuales, como podía ser secuencias de video o imágenes RGB [16]. Estos sistemas utilizaban clasificadores como los HMM (Modelos Ocultos de Markov) y SVM (Máquina de Vectores de Soporte), que demostraron ser herramientas robustas para clasificar actividades en conjuntos de datos pequeños. El problema de estas técnicas es que necesitan un esfuerzo considerable en el diseño manual de características [24].

Con el crecimiento del DL, se comenzó a abordar el HAR de manera más eficiente pudiendo procesar los datos de los sensores en bruto. Modelos como las Redes Neuronales Convolucionales (**CNN**) y las Redes Neuronales Recurrentes (**RNN**) permitieron trabajar con series temporales de sensores [25]. Más tarde, se introdujeron **modelos híbridos**, como la combinación de CNN con LSTM (*Long Short-Term Memory*) [26], [27], que aprovecharon la capacidad de extracción de características de las CNN junto con el modelado de dependencias temporales de las RNN, haciendo así que se mejorara la precisión en el HAR al capturar tantos patrones locales como relaciones a largo plazo en los datos de los sensores [28].

Dentro de la IA, surgen, como ya hemos visto y podemos ver en la Figura 2.1, disciplinas que se especializan en diferentes aspectos. Entre ellas, el ML destaca como una herramienta clave para abordar problemas complejos que no puedan resolverse fácilmente con algoritmos basados en reglas preestablecidas.

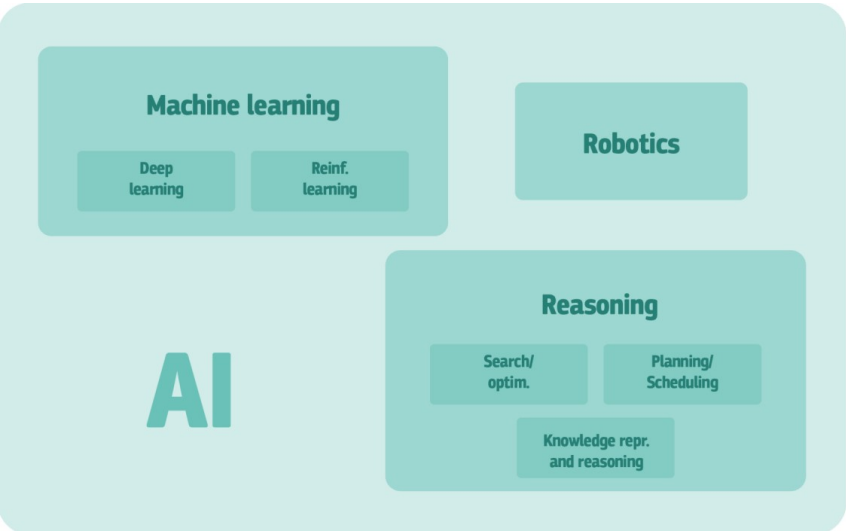

Figura 2.1: Subdisciplinas de la IA y sus relaciones. Fuente [1]

2.2. Aprendizaje Automático

Dentro de la IA encontramos el **Aprendizaje Automático** consolidado como una disciplina fundamental e interdisciplinaria, ya que combina conceptos de informática, matemáticas y estadística (Figura 2.2). Podemos definir el ML como: "Problema de aprendizaje en el que se busca mejorar alguna medida de rendimiento al ejecutar alguna tarea, a través de algún tipo de experiencia de entrenamiento". [29]

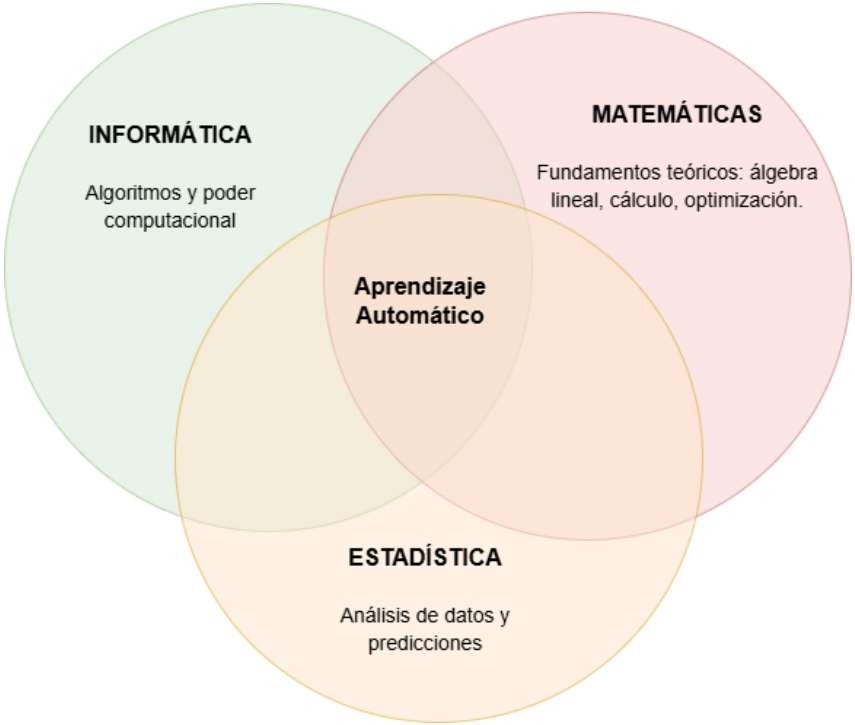

Figura 2.2: Disciplinas relacionadas con el ML. Fuente: elaboración propia

Es decir, un programa se dice que aprende de la experiencia E con respecto a alguna clase de tareas T y la medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E. Un ejemplo de este problema podrá ser jugar a las damas, donde la tarea (T) será jugar a las damas, como medida de rendimiento (P) estará el porcentaje de partidas ganadas y la experiencia de entrenamiento (E) será jugar partidas de práctica contra sí mismo [30].

Como podemos apreciar en la Figura 2.3, se pueden distinguir diferentes tipos de aprendizaje dentro del ML, cada uno de ellos representan una manera distinta de abordar los distintos problemas.

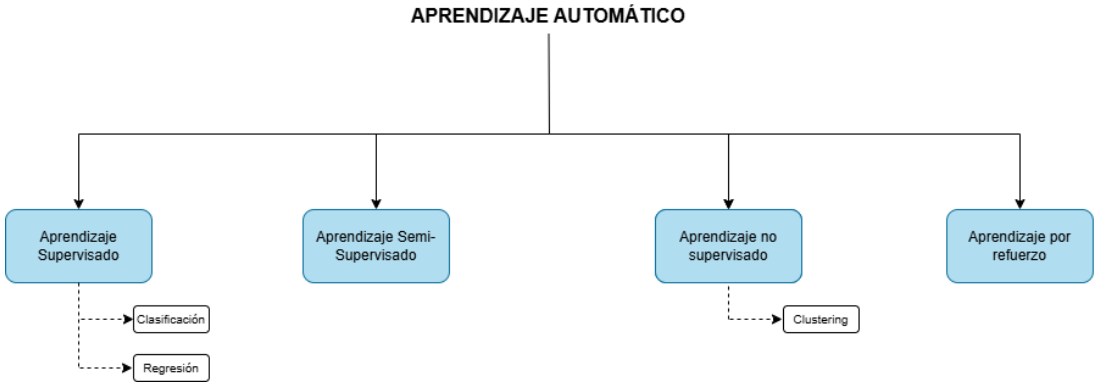

Figura 2.3: Clasificación de aprendizaje automático

2.2.1. Aprendizaje supervisado

El **Aprendizaje supervisado** se define como el proceso mediante el cual se aprende de una función que establece una correspondencia entre un conjunto de variables de entrada (X) y una variable de salida (Y). Más tarde, esta función se emplea para estimar los valores de salida correspondientes a nuevos datos no observados previamente. La característica principal del aprendizaje supervisado es la **disponibilidad de datos de entrenamiento etiquetados**, donde cada ejemplo incluye las etiquetas que el sistema debe asociar a las entradas. El objetivo es generar modelos que a partir de datos de entrenamiento sean capaces de clasificar otros datos sin etiquetar [31].

Como podemos ver en la Figura 2.4 los datos etiquetados se preprocesan y se utilizan para entrenar el modelo de ML, el cual genera predicciones sobre los datos de prueba.

Uno de los modelos más destacados y usados del aprendizaje supervisado son las **redes neuronales**, sin embargo, se abordarán con más detalle en la Sección 2.4. Otros modelos relevantes pueden ser:

- **Árbol de decisión [32]:** se trata de un clasificador que se expresa como una partición recursiva del espacio de instancias. Está compuesto por nodos que forman un árbol dirigido con un nodo denominado 'raíz' que no tiene aristas entrantes. Todos los demás nodos tienen exactamente una arista entrante y los nodos con aristas salientes se les conoce como nodos internos. Los demás nodos se denominan hoja (también nodos terminales o de decisión).

  Como ejemplo, podemos ver el árbol de decisión mostrado en la Figura 2.5 que ha sido generado utilizando el dataset de la Tabla 2.1, cuyo objetivo es predecir si el pronóstico del tiempo es favorable para jugar al tenis o no [33].

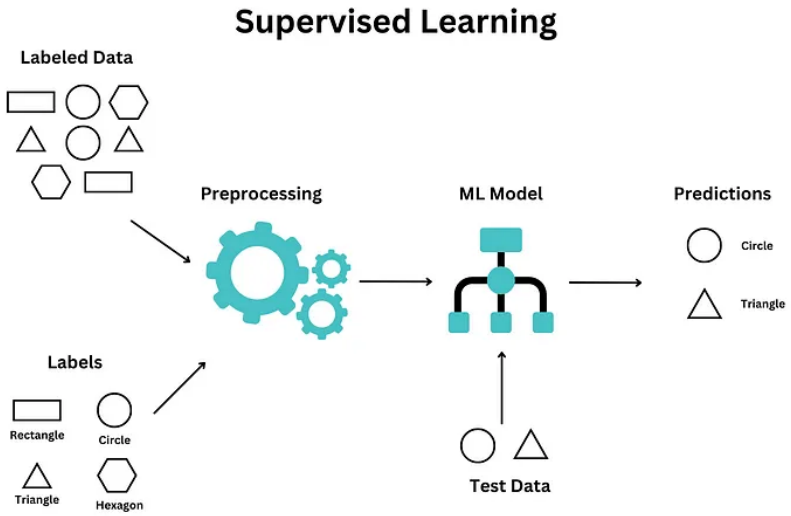

Figura 2.4: Flujo de trabajo del aprendizaje supervisado. Fuente: [2]

No.	Clima	Temperatura	Humedad	Viento	Clase
1	soleado	alta	alta	falso	No
2	soleado	alta	alta	verdadero	No
3	nublado	alta	alta	falso	Si
4	lluvia	templada	alta	falso	Si
5	lluvia	fría	normal	falso	Si
6	lluvia	fría	normal	verdadero	No
7	nublado	fría	normal	verdadero	Si
8	soleado	templada	alta	falso	No
9	soleado	fría	normal	falso	Si
10	lluvia	templada	normal	falso	Si
11	soleado	templada	normal	verdadero	Si
12	nublado	templada	alta	verdadero	Si
13	nublado	alta	normal	falso	Si
14	lluvia	templada	alta	verdadero	No

Tabla. 2.1: Dataset de condiciones climáticas para predecir si se juega al tenis.

Como se puede ver en el ejemplo mostrado, el árbol de decisión, por lo tanto, es una representación visual y lógica de las reglas aprendidas del dataset, permitiendo clasificar nuevas instancias y predecir si las condiciones meteorológicas

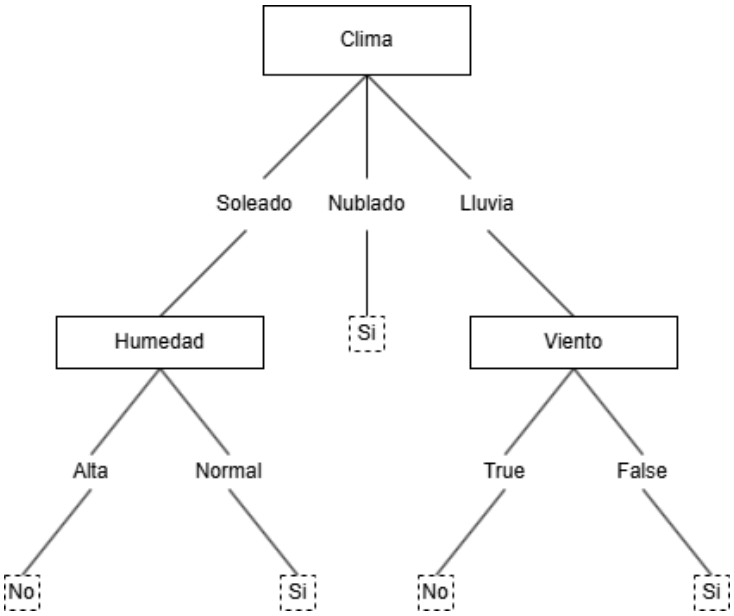

Figura 2.5: Árbol de decisión sencillo de ejemplo.

son favorables para jugar al tenis.

- **Naïve-Bayes**: Es un **clasificador probabilístico** relacionado con el teorema de Bayes (Ecuación 2.1) [34] donde X son las características e Y son las clases, $P(X)$ es la probabilidad de las características y $P(Y)$ es la probabilidad previa de la clase Y, $P(Y \mid X)$ probabilidad condicional de Y dado X (probabilidad de que el evento Y ocurra bajo la condición de que X ha ocurrido) y $P(X|Y)$ probabilidad condicional de X dado Y.

$$P(X \mid Y) = \frac{P(X)P(Y \mid X)}{P(Y)} \tag{2.1}$$

  Este modelo se basa en el **supuesto de independencia incondicional** [35], según el cual se asume que las características (X) son independientes entre sí dado el conocimiento de la clase (Y), esto significa que la presencia de una característica en particular no afecta a la probabilidad de otra, simplificando así los cálculos.

  NB se dirige principalmente a la industria de la clasificación de textos y se utiliza para fines de *clustering* y clasificación [36].

- **k-NN** (k-Nearest Neighbors): Este algoritmo de **clasificación** asigna una clase a un nuevo dato considerando la clase predominante entre sus k vecinos más cercanos en el conjunto de datos de entrenamiento, como se ilustra en la Figura 2.6. En problemas de clasificación, la clase del nuevo dato corresponde a la más representada entre los vecinos, mientras que en problemas de regresión, se

calcula el promedio de los valores de los vecinos [37].

Generalmente, la distancia entre los puntos se determina utilizando la distancia euclídea como se observa en la Ecuación 2.2, donde, p y q son los puntos en el espacio, p_i y q_i son las coordenadas de los puntos en la dimensión i, y n es el número de dimensiones.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.2}$$

Es conocido como un algoritmo de aprendizaje perezoso, ya que no se construye un modelo explícito durante la fase de entrenamiento, en su lugar, se almacenan todos los datos de entrenamiento y realiza los cálculos necesarios sólo en el momento de la clasificación [38].

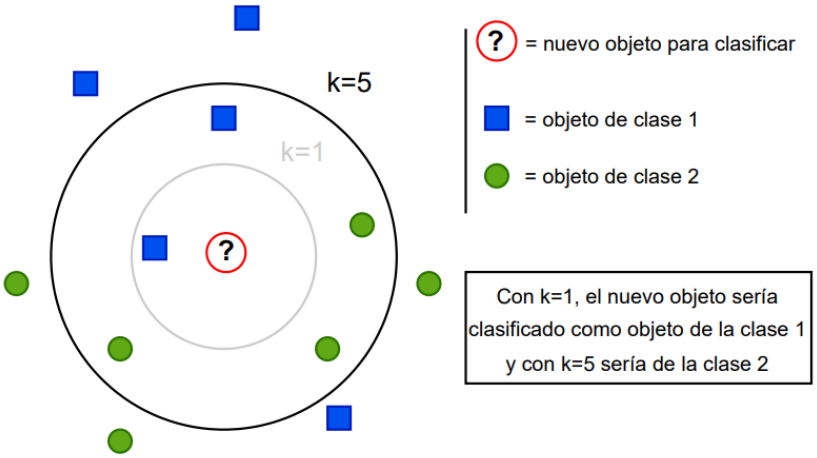

Figura 2.6: Ejemplo del funcionamiento de K-NN.

- **SVM [39]**: Se trata de un algoritmo de aprendizaje supervisado diseñado para resolver tanto problemas de clasificación como problemas de regresión. Su principal característica es la construcción de un **hiperplano** óptimo que separa las clases maximizando el margen entre ellas. Los **vectores de soporte**, que corresponden a los puntos de datos más cercanos al hiperplano, determinan su posición y orientación. Para manejar datos no linealmente separables, es decir, que no se pueden dividir las diferentes clases de datos mediante una línea recta, las SVM utilizan márgenes suaves (*soft margins*) y funciones kernel que permiten trabajar eficientemente en espacios de características de alta dimensión.

  La Figura 2.7 muestra el funcionamiento de las SVM, ilustrando el hiperplano óptimo que separa dos clases, los vectores de soporte y el margen máximo entre

las clases.

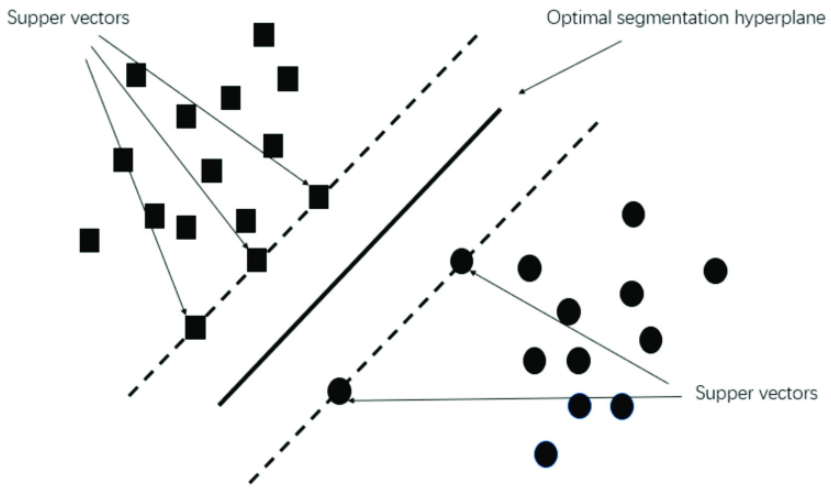

Figura 2.7: Funcionamiento de un clasificador SVM. Fuente [3]

2.2.2. Aprendizaje semi-supervisado

El Aprendizaje semi-supervisado se caracteriza por usar **datos etiquetados y no etiquetados**, donde, normalmente el número de datos no etiquetados es mayor que el de datos etiquetados. El propósito principal del SSL (*Semi-Supervised Learning*) es construir un modelo que pueda clasificar correctamente los datos no etiquetados a partir de la información proporcionada por los datos etiquetados [34]. El funcionamiento del SSL se ilustra en la Figura 2.8. Se pueden distinguir dos tipos de SSL [40]:

- ***closed-set* SSL**: Los datos etiquetados y no etiquetados comparten el mismo conjunto de clases, es decir, todas las etiquetas posibles ya están presentes en el conjunto de datos etiquetados, y el objetivo del modelo es extender el conocimiento adquirido a los datos no etiquetados.
- ***Open-set* SSL**: Los datos no etiquetados pueden contener clases que no están presentes en el conjunto de datos no etiquetados.

Algunos de los algoritmos más comunes del SSL son:

- ***Self-training*** [41]: Se lleva a cabo en tres etapas diferentes. En primer lugar, se hace un preentrenamiento, donde un modelo inicial es entrenado usando un

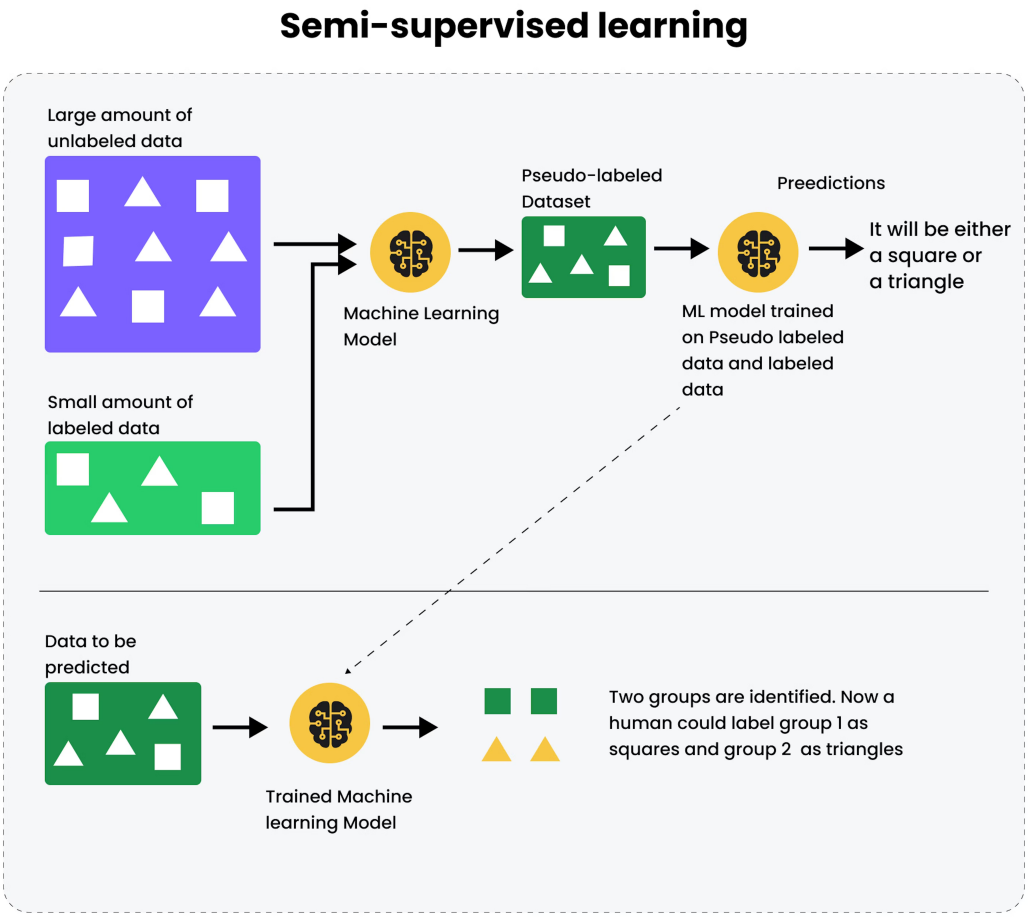

Figura 2.8: Funcionamiento de un SSL (*Self-training*). Fuente [4]

conjunto pequeño de datos etiquetados. A continuación este modelo se usa para generar pseudoetiquetas, prediciendo las etiquetas de los datos no etiquetados. Por último el modelo se vuelve a entrenar combinando tanto los datos originales etiquetados como los datos no etiquetados con las nuevas pseudoetiquetas, teniendo así mayor cantidad de información para mejorar el aprendizaje.

La principal desventaja de este enfoque está en la precisión de las pseudoetiquetas, ya que, al ser predicciones del propio modelo, pueden contener errores.

- **GSSL** (*Graph-based Semi-Supervised Learning*) [42]: Se caracteriza por la utilización de **grafos** como una representación de los datos, que permite inferir etiquetas para las muestras no etiquetadas a partir de la estructura y las relaciones del grafo.

  En primer lugar se crea el grafo, que como se observa en la Figura 2.9 los nodos representan las muestras disponibles (etiquetadas y no etiquetadas) y los bordes ponderados indican la similitud entre pares de nodos. Este enfoque se sustenta

en la **hipótesis de la variedad**, que establece que las muestras ubicadas cerca unas de otras tienden a compartir la misma etiqueta.

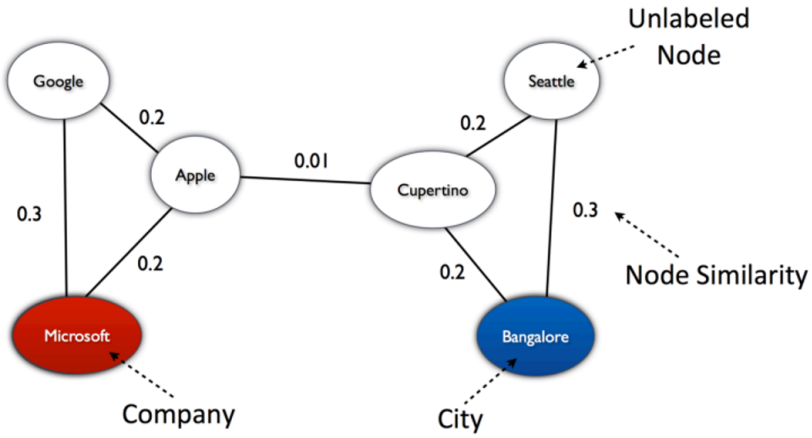

Figura 2.9: Ejemplo de representación gráfica en el GSSL. Fuente [5]

Después de construir el grafo, las etiquetas de las muestras etiquetadas se propagan hacia las no etiquetadas utilizando la información estructural del grafo. Con este proceso aprovechamos las conexiones entre los nodos para inferir con precisión las etiquetas faltantes.

2.2.3. Aprendizaje no supervisado

El aprendizaje no supervisado es un enfoque del ML en el que un modelo trabaja **con datos sin etiquetar** [43], es decir, datos que no tienen información previa sobre categorías o clasificaciones. A diferencia del aprendizaje supervisado, donde el modelo aprende con ejemplos claramente identificados, en el no supervisado el objetivo es **descubrir patrones ocultos, estructuras o relaciones dentro de los datos por sí solo** [44].

En la Figura 2.10 se muestran las diferencias clave entre el aprendizaje supervisado y el no supervisado. Mientras que el aprendizaje supervisado se enfoca en clasificar los datos en etiquetas preexistentes, como los círculos azules y las cruces rojas, el aprendizaje no supervisado busca identificar patrones o estructuras en los datos, como la formación de los grupos representados por los círculos rojos.

Uno de los algoritmos más usados del aprendizaje no supervisado es ***K-means*** cuyo objetivo principal es particionar un conjunto de datos en k grupos, asignando cada punto al grupo cuyo centroide esté más cercano. El proceso es iterativo y consta de dos pasos, en primer lugar, los datos se asignan al centroide más cercano, y

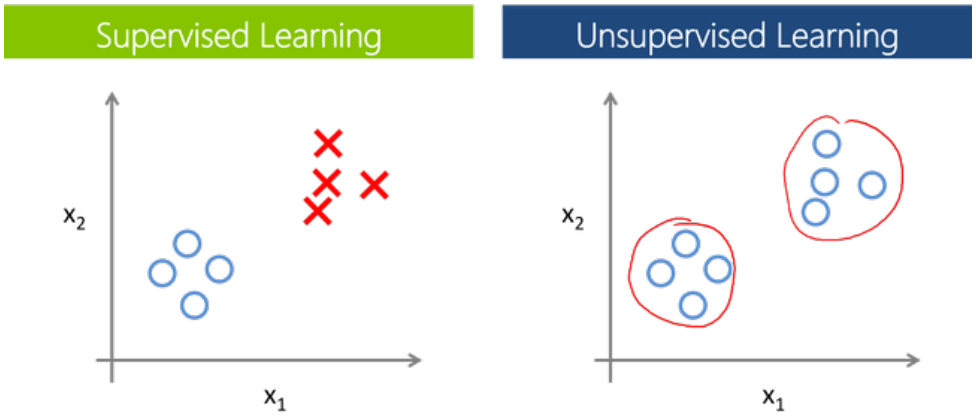

Figura 2.10: Comparación entre aprendizaje supervisado y no supervisado. Fuente [6]

en segundo lugar, los centroides se vuelven a calcular como la media de los puntos asignados a cada grupo, repitiéndose este ciclo hasta la convergencia [45]. Algunas limitaciones que presenta este algoritmo son la necesidad de definir k previamente, su sensibilidad a valores atípicos y dificultades para manejar datos mixtos [46]. En la Figura 2.11 se puede observar un ejemplo del funcionamiento del algoritmo, donde datos sin etiquetar son agrupados en clusters con sus respectivos centroides.

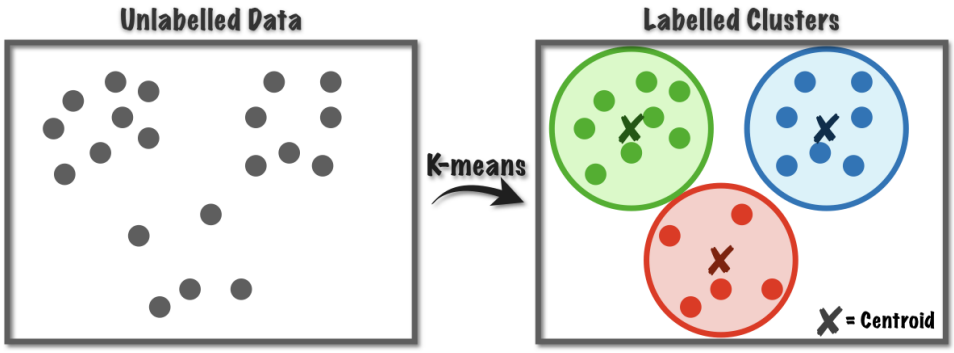

Figura 2.11: Ejemplo de funcionamiento de K-means. Fuente [7]

2.2.4. Aprendizaje por refuerzo

El **Aprendizaje por refuerzo** se refiere al desafío que enfrenta un agente al intentar aprender un comportamiento mediante interacciones de prueba y error con un entorno dinámico. Este enfoque se basa en un sistema de recompensas y castigos, que permiten al agente mejorar su rendimiento sin necesidad de instrucciones sobre cómo alcanzar el objetivo. El propósito final es que el agente descubra una política que maximice la suma acumulativa de las recompensas a largo plazo [47]. Este funcionamiento lo podemos ver ilustrado en la Figura 2.12.

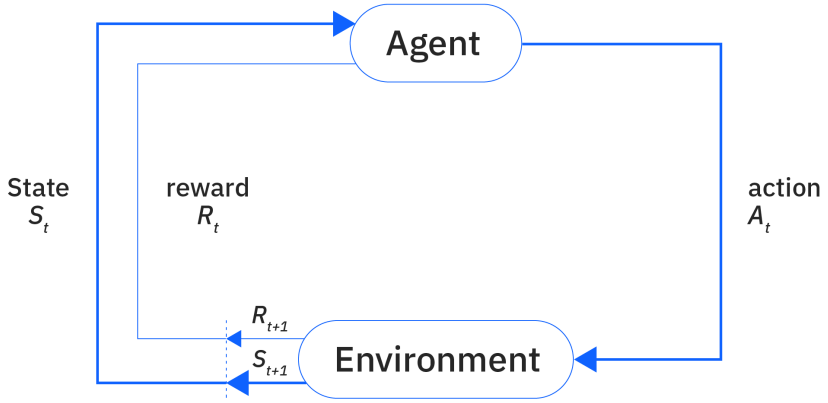

Figura 2.12: Esquema de funcionamiento del aprendizaje por refuerzo. Fuente [8]

Algunos conceptos clave del aprendizaje por refuerzo son [48]:

- **Agente**: Toma las decisiones e interactúa con el entorno.
- **Entorno**: Aquello con lo que el agente interactúa y de donde recibe retroalimentación en forma de recompensas.
- **Estado**: Representación del entorno en un momento dado.
- **Acción**: Es una decisión que el agente puede tomar. Puede llevar al agente a un nuevo estado.
- **Recompensa**: Valor escalar que el agente recibe del entorno como resultado de sus acciones, el objetivo del agente es maximizar la recompensa a largo plazo. La función de recompensa no la puede modificar el agente y define lo que es realmente “bueno” o “malo” para este.
- **Política**: Es el mapeo desde posibles estados a posibles acciones. Puede ser una política almacenada (tabla de búsqueda simple) o una política calculada (cálculos y búsquedas en un árbol de valores).
- **Función de valor**: Mapeo del conjunto de estados a las estimaciones del retorno esperado a largo plazo que se espera obtener a partir de un estado. Se actualiza continuamente según las consecuencias experimentadas por el agente.
- **Modelo**: Representación del entorno que permite al agente planificar, es decir, prever las consecuencias de secuencias de acciones.

- **Exploración**: Probar acciones nuevas para mejorar el conocimiento del entorno y la función de valor.
- **Explotación**: Utilizar el conocimiento actual para tomar acciones que maximicen la recompensa inmediata.

2.3. Aprendizaje Profundo

El **Aprendizaje Profundo** [49] es una rama dentro de la IA y del ML donde las máquinas realizan tareas que normalmente requieren inteligencia humana. Trata de imitar el aprendizaje del cerebro humano utilizando redes neuronales artificiales (hablaremos de estas con mayor detalle en la Sección 2.4), compuestas por neuronas interconectadas (estructura ilustrada en la Figura 2.13) que procesan datos como textos, imágenes, vídeos y sonidos. Estas **DNN** (*Deep Neuronal Network*) se caracterizan por tener múltiples capas ocultas que les permiten aprender características de alta complejidad y patrones en los datos.

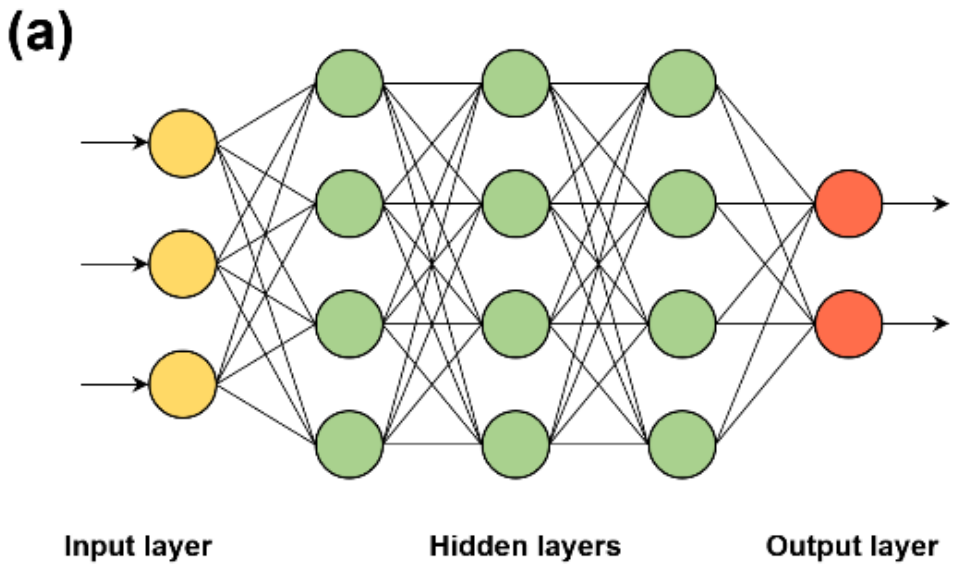

Figura 2.13: Estructura de una DNN. Fuente [9]

El proceso del aprendizaje en DNN se basa en dos pasos fundamentales:

- **Propagación hacia adelante**: Los datos de entrada se propagan a través de las capas de la red, generando predicciones.
- **Retropropagación del error (*Backpropagation)***: El error entre las predicciones y los valores reales se utilizan para ajustar los parámetros del modelo.

El DL ha llevado a las redes neuronales a superar los métodos de vanguardia en tareas complejas, demostrando ser muy útil en dominios con gran cantidad de datos y de alta dimensionalidad, como la visión por computador, procesamiento del lenguaje natural, reconocimiento de voz, texto, imágenes y audio.

La principal ventaja del DL es su capacidad para automatizar el proceso de **extracción de características**, ya que las DNN pueden alimentarse con datos en bruto y descubrir representaciones relevantes para la tarea en cuestión, reduciendo el esfuerzo humano significativamente [50].

La introducción del DL en el ámbito del HAR ha facilitado el proceso de extracción de características significativas a partir de los datos de sensores en bruto. La evolución de los modelos de DL ha logrado avances importantes gracias a enfoques como las CNN, la extensión del uso de esquemas de transferencias de pesos (que permite la reutilización del conocimiento al entrenar un modelo de reconocimiento en un conjunto de datos y aplicarlo a otro conjunto diferente), y el desarrollo de modelos híbridos, como la fusión de CNN con LSTM.

Además, el diseño de funciones de pérdida avanzadas junto con paradigmas de optimización, como la entropía cruzada y el descenso de gradiente estocástico (SGD), han contribuido a que el diseño basado en HAR sea más modular del tipo "plug-and-play"[51].

2.4. Redes Neuronales

Las **Redes Neuronales** artificiales [52], enmarcadas dentro del aprendizaje profundo, son una de las técnicas más utilizadas y con mejores resultados en problemas donde hay que manejar grandes volúmenes de datos y realizar tareas como la clasificación.

Las ANN se definen como modelos computacionales diseñados para imitar la estructura y el funcionamiento del cerebro humano, con el objetivo de realizar tareas específicas de procesamiento de información y aprendizaje.

Las ANN están compuestas por **neuronas artificiales**, que intentan simular el comportamiento de las neuronas del cerebro humano. A diferencia de las neuronas biológicas, que transmiten señales eléctricas, las neuronas artificiales funcionan a través de operaciones matemáticas. Como se observa en la Figura 2.14, las neuronas artificiales reciben un conjunto de entradas provenientes de otras neuronas o de los

datos iniciales. Cada entrada se multiplica por un peso asociado y la suma ponderada se compara con un umbral. Si la suma es igual o superior al umbral, la neurona se activa y devuelve 1; de lo contrario, devuelve 0.

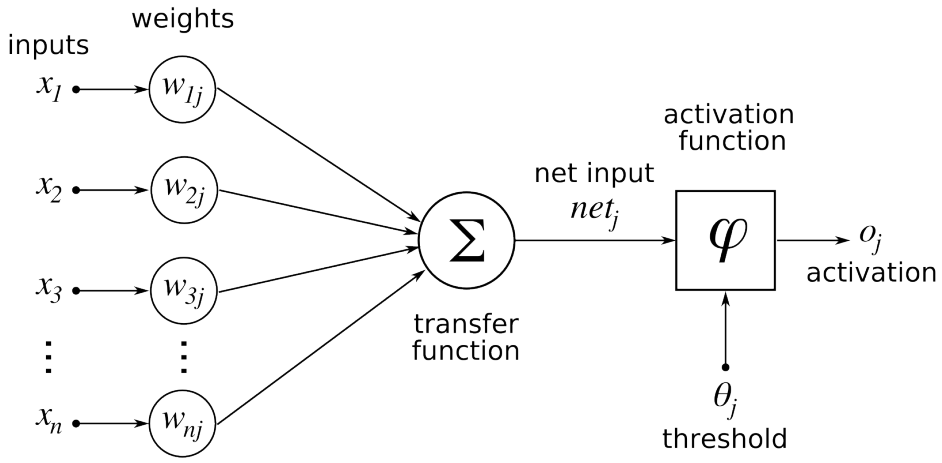

Figura 2.14: Arquitectura de una neurona artificial. Fuente [10]

Estas neuronas artificiales están conectadas entre sí, y se organizan, normalmente, en las siguientes capas:

- La **capa de entrada** es en la que se encuentran las neuronas encargadas de recibir la información del exterior.
- La **capa de salida** es la encargada de producir el resultado del procesamiento.
- Las **capas intermedias** se encuentran entre la capa de entrada y la capa de salida. En el caso de que solo haya una capa de este tipo, la red se denomina **red neuronal simple** (ilustrada en la Figura 2.15), mientras que si tiene más de una, se denomina **red neuronal profunda** (como ya hemos visto en la Figura 2.13).

Sabiendo lo que son las redes neuronales y como se organizan, a continuación se analizarán algunos componentes de las redes neuronales, como las funciones de activación, las funciones de pérdida, los optimizadores, los hiperparámetros y las métricas de rendimiento, describiendo su propósito, las opciones más comunes y cómo influyen en el aprendizaje de la red.

2.4.1. Función de activación

Las **funciones de activación** [53], también conocidas como funciones de umbral o de transferencia, son transformaciones escalares que se aplican a las entradas netas

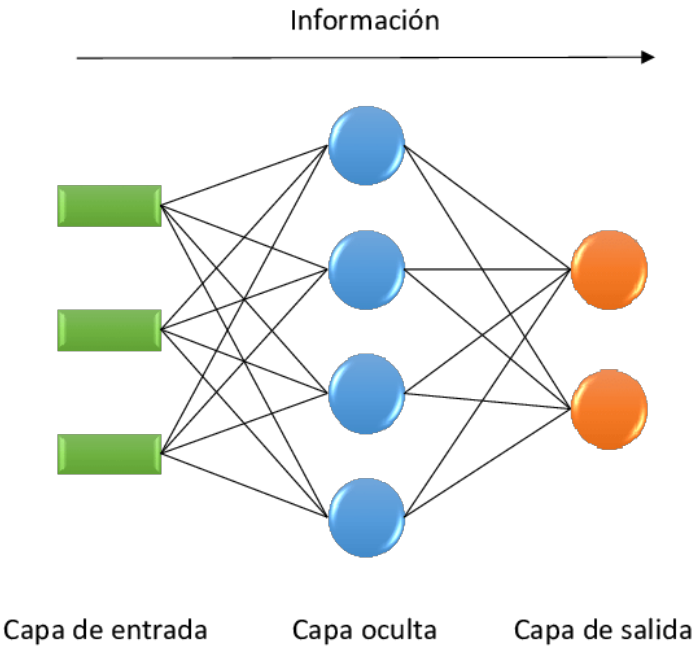

Figura 2.15: Estructura de una red neuronal simple. Fuente [11]

de las neuronas en una red neuronal. Estas funciones tienen como objetivo convertir la entrada neta en una salida, conocida como activación de la unidad, que más tarde se utiliza como entrada para la siguiente capa de la red.

El propósito principal de estas funciones es permitir que las redes aprendan relaciones complejas y no lineales entre las entradas y las salidas. Sin ellas, las redes neuronales estarían limitadas a aprender únicamente funciones lineales, lo que restringiría su capacidad para modelar datos complejos. Por estos motivos, la no linealidad de las funciones de activación es clave, ya que, gracias a esto, las redes pueden aprender funciones más complejas y modelar datos más variados y complejos, como imágenes, videos, audio, voz o texto, convirtiendo a las redes neuronales en aproximadores de funciones universales, capaces de computar y aprender cualquier función que se les proporcione.

A continuación, se analizarán los diferentes tipos de funciones de activación utilizados en redes neuronales, destacando sus características principales y los contextos en los que pueden aplicarse.

- La función de **escalón binario**, definida por la ecuación 2.3 se trata de la función de activación más simple, donde la salida es 1 si la entrada es mayor o igual a 0, y 0 si la entrada es menor que 0. Se suele utilizar para clasificadores binarios, además, su gradiente es cero lo que puede dificultar la retropropagación. En

la Figura 2.16 se puede observar su representación gráfica.

$$f(x) = \begin{cases} 1, & \text{si } x \geq 0 \\ 0, & \text{si } x < 0 \end{cases} \tag{2.3}$$

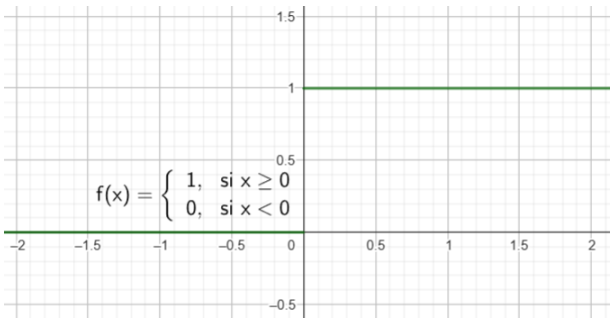

Figura 2.16: Función de activación escalón binario.

- En la función **lineal** la salida es directamente proporcional a la entrada. Está definida como indica la Ecuación 2.4 donde a es una constante, por lo tanto su gradiente es constante, lo que significa que la red no mejora el error porque el valor del gradiente es el mismo en cada iteración, por lo tanto esta función de activación no es adecuada para identificar patrones complejos. En la Figura 2.17 se muestra una representación gráfica de esta función.

$$f(x) = ax \tag{2.4}$$

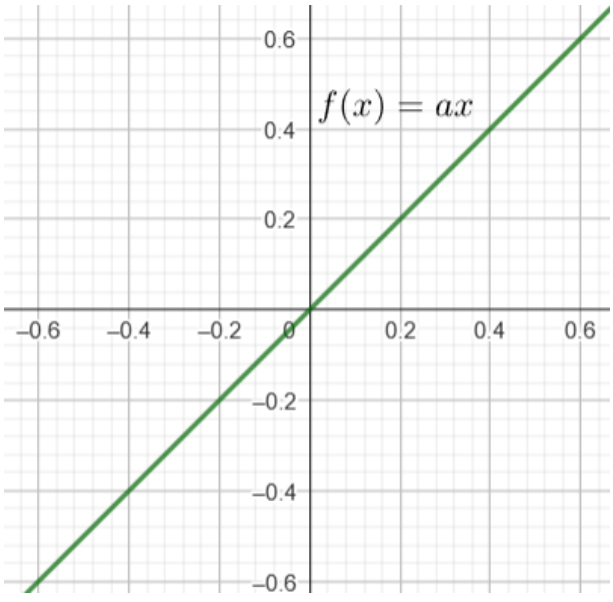

Figura 2.17: Función de activación lineal.

- La función **sigmoide** es una función no lineal que transforma los valores al rango de 0 a 1, definida por la Ecuación 2.5. Como podemos ver en la Figura 2.18 tiene una forma de S y no es simétrica con respecto a cero, lo que puede ser un problema en varios casos.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

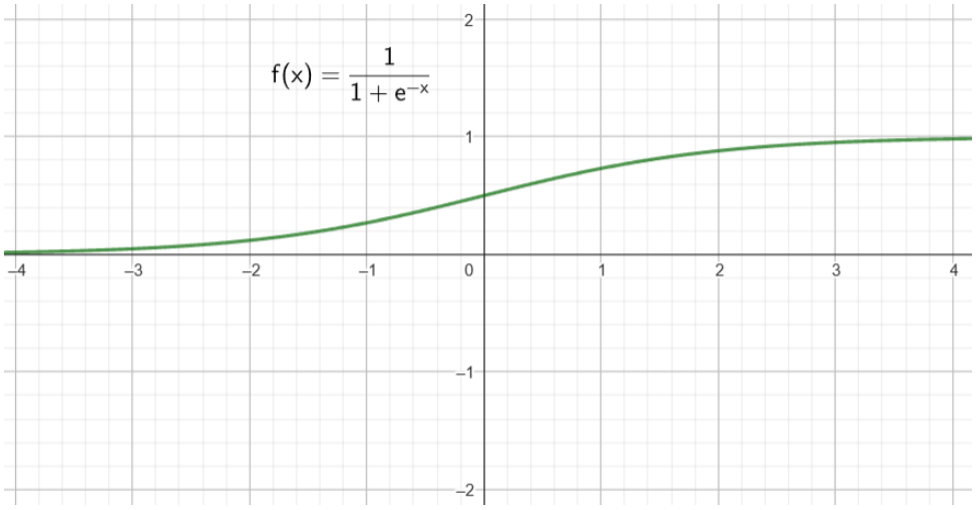

Figura 2.18: Función de activación sigmoide.

- La función **tangente hiperbólica** es similar a la función sigmoide, pero en este caso si es simétrica con respecto al origen, con un rango de valores de -1 a 1 y definida como indica la Ecuación 2.6. Normalmente se prefiere antes que la sigmoide porque sus gradientes no están restringidos a variar en una dirección determinada y, además está centrado en cero. La Figura 2.19 muestra una representación gráfica de esta función de activación.

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.6}$$

- La función **ReLU** (*Rectified Linear Unit*), definida por la ecuación 2.7, es una función no lineal muy utilizada en redes neuronales, ya que, con esta función de activación no todas las neuronas se activan a la vez. En algunos casos, su gradiente puede ser cero lo que impide la actualización de pesos y sesgos. La Figura 2.20 muestra una representación gráfica de esta función.

$$f(x) = \begin{cases} x, & \text{si } x \geq 0 \\ 0, & \text{si } x < 0 \end{cases} \tag{2.7}$$

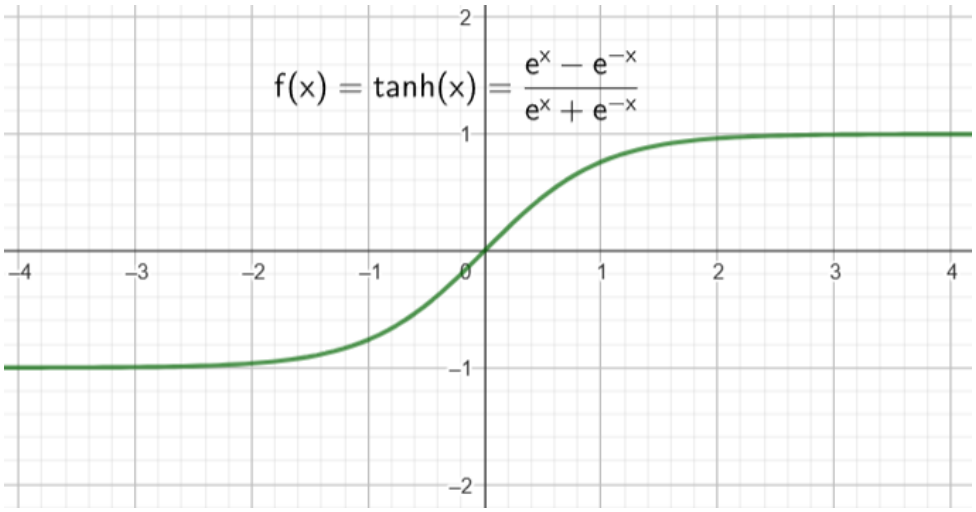

Figura 2.19: Función de activación tangente hiperbólica.

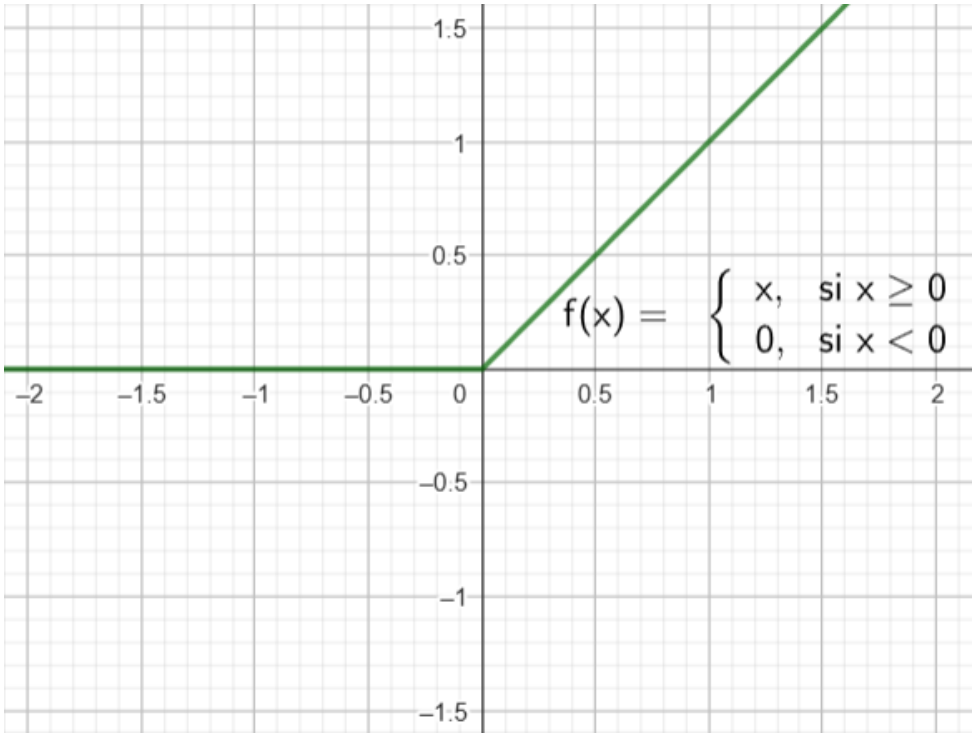

Figura 2.20: Función de activación ReLU.

2.4.2. Función de pérdida

Otra herramienta fundamental en el entrenamiento de modelos de DL son las **funciones de pérdida** [54], estas permiten medir la diferencia entre las predicciones del modelo y los valores reales esperados, sirviendo como una guía para la mejora del modelo en cada iteración. El objetivo principal de estas funciones es minimizar esa diferencia ajustando los parámetros del modelo.

La elección de una función de pérdida adecuada depende de la arquitectura del modelo y de la tarea específica que se quiera resolver, existen varias propiedades importantes que deben considerarse al seleccionar una función de pérdida. En primer lugar, la **convexidad** asegura que cualquier mínimo local encontrado durante el entrenamiento sea también el mínimo global, lo que facilita la optimización. En segundo lugar, la **diferenciabilidad** es crucial, ya que permite calcular las derivadas necesarias para el algoritmo de retropropagación. Por último, la **robustez** es una característica importante, ya que asegura que la función sea capaz de manejar valores atípicos y no se vea significativamente afectada por un pequeño número de datos extremos.

En cuanto a los diferentes tipos de funciones de pérdida, estas se dividen principalmente en aquellas diseñadas para tareas de regresión y para problemas de clasificación:

Regresión

- El **Error Cuadrático Medio (MSE)** [55] calcula el promedio de las diferencias al cuadrado entre los valores predichos y los verdaderos, penalizando fuertemente los errores grandes. Esta función de pérdida es definida como indica la ecuación 2.8, donde n es el número total de muestras en el conjunto de datos, y_i es el valor real de la i-ésima muestra e $\hat{y}_i$ es la predicción del modelo para la i-ésima muestra.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.8}$$

- El **Error Absoluto Medio (MAE)** [56] calcula el promedio de las diferencias absolutas entre las predicciones y los valores reales, siendo menos sensibles a valores atípicos en comparación con el MSE. La función de pérdida MAE puede ser definida por la ecuación 2.9, donde n es el número total de muestras en el conjunto de datos, y_i es el valor real de la i-ésima muestra e $\hat{y}_i$ es la predicción del modelo para la i-ésima muestra.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.9}$$

Clasificación Binaria

- La función de pérdida de **Entropía Cruzada Binaria (BCE)** [57] se usa en problemas de clasificación binaria para medir la diferencia entre la predicción y la etiqueta de clase verdadera. La ecuación 2.10 define matemáticamente a esta función de pérdida, donde y es el valor de la clase para la i-ésima muestra y p es la probabilidad predicha por el modelo de que la i-ésima muestra pertenezca a la clase 1.

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{2.10}$$

- La función de pérdida de **Hinge** [58] se utiliza comúnmente en modelos de clasificación binaria, como las SVM, aunque puede adaptarse para problemas de clasificación multiclase. Está definido por la siguiente ecuación 2.11, donde y representa la etiqueta verdadera de la instancia y $f(x)$ es la salida predicha por el modelo para la entrada x.

$$L(y, f(x)) = \text{máx}(0, 1 - y \cdot f(x)) \tag{2.11}$$

Clasificación Multiclase

- La función de pérdida de **Entropía Cruzada Categórica (CCE)** [59] también conocida como pérdida logarítmica multiclase, se utiliza en tareas de clasificación multiclase para medir la diferencia entre las probabilidades predichas y la distribución real de los resultados. Amplía el concepto de entropía cruzada binaria a múltiples clases. La ecuación 2.12 define matemáticamente a esta función de pérdida, donde N es el número de muestras en el conjunto de datos, C es el número de clases en el problema de clasificación, $y_{i,j}$ indica si la muestra i pertenece a la clase j (valor binario 0 o 1) y $P_{i,j}$ es la probabilidad predicha de que la muestra i pertenezca a la clase j.

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \log(p_{i,j}) \tag{2.12}$$

- La función de pérdida de **Entropía Cruzada Categórica Dispersa (SCCE)** [54] es una variación de CCE diseñada para tareas de clasificación multiclase en las que las clases se codifican como enteros. Este enfoque es especialmente útil en situaciones con un gran número de clases, en las que la codificación de una sola clase consumiría mucha memoria y sería menos eficaz. La ecuación 2.13

calcula la pérdida de una predicción individual, mientras que la pérdida global de SCCE para el conjunto de datos se calcula como la media de estas pérdidas individuales (ecuación 2.14). Donde $\hat{y}_{i,y_i}$ es la probabilidad predicha por el modelo para la clase correcta y_i de la muestra i y n es el número total de muestras.

$$H(y, \hat{y}) = -\log(\hat{y}_{i,y_i}) \tag{2.13}$$

$$H(Y, \hat{Y}) = -\frac{1}{n}\sum_{i=1}^{n}\log(\hat{y}_{i,y_i}) \tag{2.14}$$

2.4.3. Optimizadores

Los **optimizadores** se pueden definir como algoritmos utilizados en el proceso de entrenamiento de las DNN que sirven para minimizar la función de pérdida y obtener los parámetros óptimos de la red en un tiempo aceptable.

Los optimizadores [60] se pueden clasificar en dos categorías, la primera son los **optimizadores de primer orden**, estos métodos utilizan el gradiente de la función de pérdida para actualizar los parámetros de la red neuronal. Se basan en la dirección del gradiente para encontrar el mínimo de la función de pérdida. Por último, están los **optimizadores de segundo orden**, estos consideran la curvatura de la función de pérdida mediante la matriz Hessiana para lograr una convergencia más rápida.

Optimizadores de primer orden

- El **Descenso de Gradiente Estocástico (SGD)** [61] es una variante del algoritmo de Descenso de Gradiente, que ajusta los parámetros del modelo iterativamente, basándose en estimaciones del gradiente de la función de pérdida. Se representa matemáticamente como indica la ecuación 2.15, donde θ_t son los parámetros del modelo en la iteración t, α_t es la tasa de aprendizaje en la iteración t, también conocido como *learning rate* y $\nabla f(\theta_t)$ es el gradiente de la función de pérdida $f(\theta)$ con respecto a los parámetros θ en la iteración t.

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \tag{2.15}$$

- El **Descenso de Gradiente Estocástico con Momentum** [62] es la modificación de SGD con la condición de Nestreov. Matemáticamente se define como

indica la ecuación 2.16, donde θ_t son los parámetros del modelo en la iteración t, α_t es la tasa de aprendizaje en la iteración t, también conocido como *learning rate*, $\nabla f(\theta_t)$ es el gradiente de la función de pérdida $f(\theta)$ con respecto a los parámetros θ en la iteración t, μ es el coeficiente de momentum y b_{t+1} es la acumulación del gradiente ponderada por el coeficiente μ en iteraciones anteriores ("momento"de la iteración t+1).

$$\theta_{t+1} = \theta_t - \alpha_t \left(\nabla f(\theta_t) + \mu \cdot b_{t+1}\right) \tag{2.16}$$

- El optimizador **AdaGrad** [63] destaca por su capacidad para ajustar la tasa de aprendizaje de manera adaptativa para cada parámetro de la red neuronal. Esta adaptabilidad se basa en la suma de los gradientes observados hasta el momento durante el entrenamiento. La idea que respalda este optimizador es que los parámetros que reciben gradientes grandes con frecuencia deberían tener una tasa de aprendizaje menor, y viceversa. Se define matemáticamente como indica la ecuación 2.17, donde θ_{t+1} son los parámetros del modelo que se van a calcular, θ_t son los parámetros del modelo para la iteración t, α_t es la tasa de aprendizaje para la iteración t, G_{t+1} es la suma acumulada de los gradientes cuadrados hasta la iteración $t+1$ y $\nabla f(\theta_t)$ es el gradiente de la función de pérdida $f(\theta)$ con respecto a los parámetros θ en la iteración t.

$$\theta_{t+1} = \theta_t - \frac{\alpha_t}{\sqrt{G_{t+1} + \epsilon}} \cdot \nabla f(\theta_t) \tag{2.17}$$

- El optimizador **Adam** [64] se caracteriza por combinar las ventajas de otros optimizadores, como el uso del momentum y la adaptación de la tasa de aprendizaje. Esto último lo realiza utilizando no solo el gradiente de la función de pérdida en cada paso de optimización, sino también un promedio ponderado de los gradientes anteriores y un promedio ponderado de los cuadrados de esos gradientes.

  Las actualizaciones de los parámetros en Adam se realizan en diferentes pasos: primero, se calcula un estimador del primer momento (ecuación 2.18) y el segundo momento (ecuación 2.20), luego se corrigen los sesgos de estos momentos calculados (ecuación 2.19) y (ecuación 2.21) y por último se calcula la actualización de los parámetros (ecuación 2.22).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.18}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.19}$$

Donde β_1 es el factor de decaimiento para el primer momento, m_{t-1} es el valor anterior del primer momento y g_t es el gradiente de la función de pérdida con respecto a los parámetros en el tiempo t.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.20}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.21}$$

Donde β_2 es el factor de decaimiento para el segundo momento, v_{t-1} es el valor anterior del segundo momento y g_t^2 es el cuadrado del gradiente en el tiempo t.

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.22}$$

Donde θ_{t-1} son los parámetros del modelo en el paso $t-1$, α es la tasa de aprendizaje global, $\hat{m}_t$ es el primer momento corregido, $\hat{v}_t$ es el segundo momento corregido y ϵ un pequeño valor que evita la división por cero.

Optimizadores de segundo orden

- El **método de Newton** [65] es un optimizador que utiliza la matriz Hessiana para ajustar más rápidamente el descenso, teniendo en cuenta la curvatura de la función de pérdida. Su uso práctico se ve limitado debido al coste computacional del cálculo de la inversa de la Hessiana, sobre todo en redes neuronales grandes. Se define matemáticamente como indica la ecuación 2.23, donde θ_t representa los parámetros actuales del modelo en la iteración t, $\nabla f(\theta_t)$ es el gradiente de la función objetivo $f(\theta)$ evaluado en los parámetros actuales θ_t y Hess(θ_t) es la matriz Hessiana de la función $f(\theta)$ calculada en los parámetros actuales θ_t.

$$\theta_{t+1} = \theta_t - \text{Hess}(\theta_t)^{-1} \nabla f(\theta_t) \tag{2.23}$$

- Los métodos **cuasi-Newton** [66] buscan aproximar la matriz Hessiana inversa para reducir el costo computacional, además, intentan lograr una precisión similar a la del método de Newton, pero con una convergencia más rápida. Un método de este estilo es **BFGS (Broyden-Fletcher-Goldfarb-Shanno)**, que actualiza la Hessiana inversa en cada iteración utilizando la información del gradiente actual y anterior. La actualización se realiza mediante la ecuación 2.24, donde θ_t son los parámetros del modelo en la iteración actual t, α_t es la tasa de aprendizaje en la iteración t, H_t es la aproximación de la inversa de la matriz Hessiana

en la iteración t y $\nabla f(\theta_t)$ es el gradiente de la función de pérdida evaluado con los parámetros actuales θ_t.

$$\theta_{t+1} = \theta_t - \alpha_t H_t \nabla f(\theta_t) \tag{2.24}$$

2.4.4. Hiperparámetros

Los **hiperparámetros** [67] se definen como los parámetros que controlan el proceso de aprendizaje y la arquitectura del modelo de ANN. A diferencia de los pesos de la red, que se ajustan automáticamente durante el entrenamiento, los hiperparámetros se establecen antes del inicio del entrenamiento. La elección adecuada de los hiperparámetros es crucial para el rendimiento del modelo resultante.

Algunos de los hiperparámetros que deben establecerse de manera adecuada son [68]:

- El **tamaño del lote (Batch size)** define la cantidad de muestras que se procesan juntas antes de actualizar los pesos del modelo.
- La **Tasa de Dropout** es utilizado para prevenir el sobreajuste de las redes neuronales. Consiste en desactivar un porcentaje de neuronas durante el entrenamiento.
- El **Número de unidades** se refiere a la cantidad de neuronas en una capa densa.
- La **Tasa de aprendizaje** controla la magnitud de los pasos que el optimizador da al ajustar los pesos durante la retropropagación.

2.4.5. Métricas de rendimiento

Una vez el modelo de aprendizaje ha sido diseñado y entrenado, es fundamental evaluar su desempeño utilizando **métricas de rendimiento** [69] adecuadas. Estas métricas son medidas cuantitativas que se utilizan para evaluar la calidad y la eficacia de un modelo predictivo. Además, proporcionan una forma de cuantificar qué tan bien un modelo está aprendiendo de los datos de entrenamiento y qué tan bien es capaz de generalizar a datos nuevos o no vistos.

Teniendo en cuenta que **TP** es el número de verdaderos positivos, **FN** son los falsos negativos, **FP** son los falsos positivos y **TN** son los verdaderos negativos, algunas de las métricas más usadas son:

- La **sensibilidad (Recall)**, definida por la ecuación 2.25, indica la proporción de casos positivos reales que el modelo identifica correctamente.

$$\frac{TP}{TP + FN} \tag{2.25}$$

- La **Precisión**, definida por la ecuación 2.26, mide la proporción de predicciones positivas que son realmente correctas.

$$\frac{TP}{TP + FP} \tag{2.26}$$

- **F1-score** es la media armónica entre la precisión y el recall. Se calcula como indica la ecuación 2.27.

$$\frac{2 * (\text{precisión*recall})}{\text{precisión*recall}} \tag{2.27}$$

- La **Exactitud (accuracy)** representa la proporción de todas las predicciones (tanto positivas como negativas) que son correctas. Se calcula como indica la ecuación 2.28.

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{2.28}$$

- La **Curva ROC** es una gráfica que muestra el rendimiento de un clasificador binario, representando la tasa de verdaderos positivos frente a la tasa de falsos positivos. En la Figura 2.21 podemos ver un ejemplo de curva ROC, donde se observa una linea roja discontinua que sería una clasificación aleatoria, toda curva ROC que se encuentre a la izquierda de esta linea discontinua es mejor que la clasificación aleatoria y si está a la derecha es peor.

- La **matriz de confusión**, en clasificación binaria, es una matriz de 2x2 que muestran el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos (Figura 2.22). En la clasificación multiclase (Figura 2.23) con N clases, tendremos una matriz de NxN donde cada fila corresponde a la clase real y cada columna corresponde a la clase predicha, Los elementos diagonales (M[i,i]) representan el número de puntos que fueron correctamente clasificados (verdaderos positivos de cada clase)

Existen diferentes tipos [70] de redes neuronales que se diferencian en su estructura, objetivo, y forma de procesar los datos, algunas de estas son: CNN, PUNs (Auto-

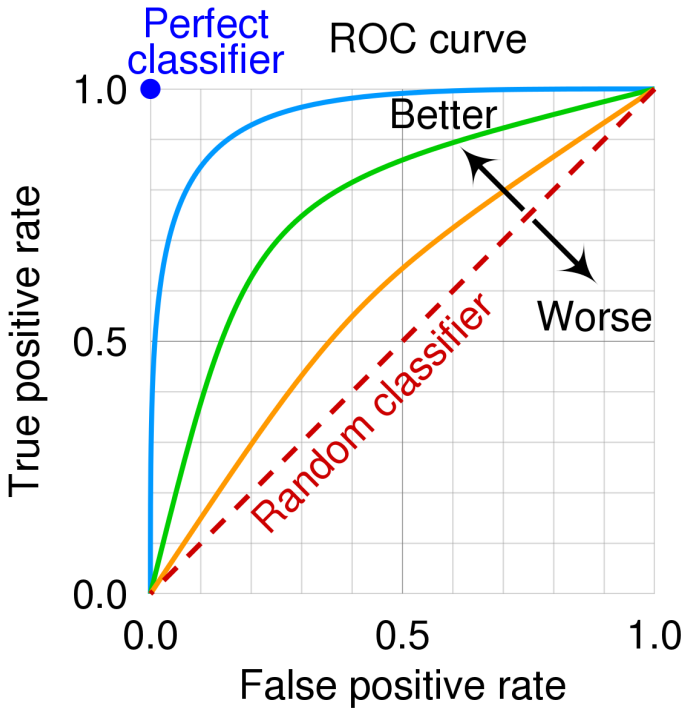

Figura 2.21: Ejemplo de curva ROC.

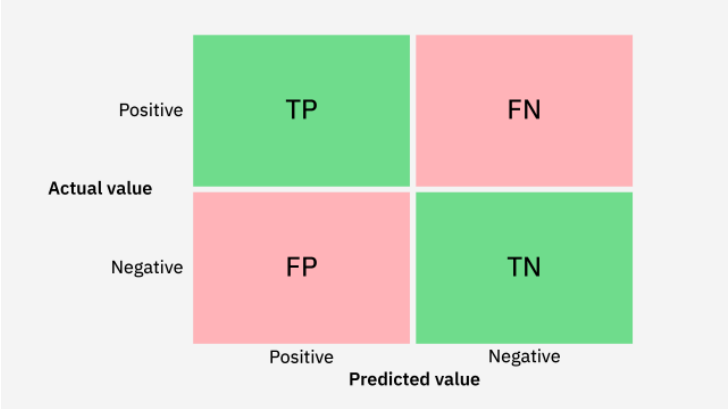

Figura 2.22: Ejemplo de matriz de confusión binaria.

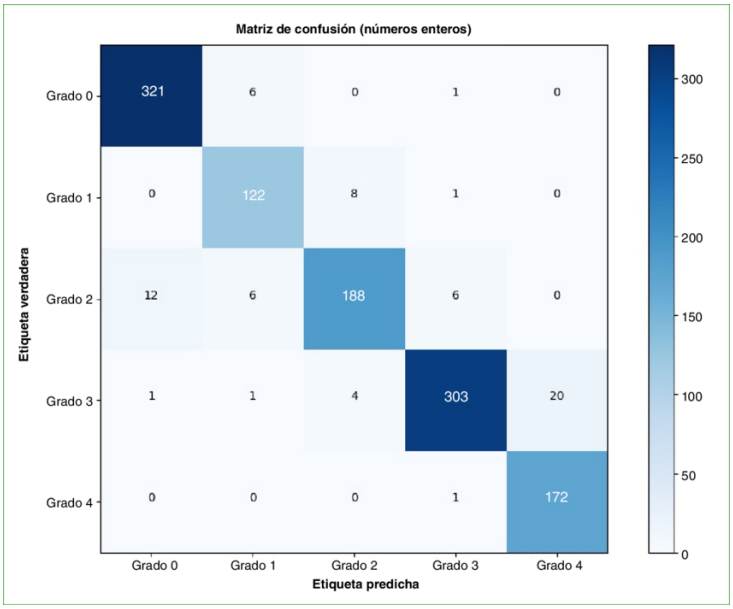

Figura 2.23: Ejemplo de matriz de confusión multiclase. Fuente [12]

encoders, GANs, DBNs) y redes recurrentes y recursivas (RNN, LSTM y redes recursivas). A continuación se describen los tipos de redes neuronales usadas para este trabajo.

2.4.6. Redes Neuronales Convolucionales

Las **CNN** [71] se definen como un tipo de red neuronal *feedforward* capaz de extraer características automáticamente a partir de los datos mediante las estructuras de convolución, eliminando la necesidad de realizar extracción de características manual.

La arquitectura de las CNN está inspirada en la percepción visual. De esta manera, una neurona biológica se corresponde con una neurona artificial. Los *kernels* de las CNN actúan como receptores que responden a diferentes características, y las funciones de activación simulan el mecanismo por el cual las señales eléctricas neuronales se transmiten únicamente si superan cierto umbral.

En comparación con las redes completamente conectadas (ilustrado en la Figura 2.24), las CNN poseen algunas ventajas. En primer lugar, poseen **conexiones locales**, donde cada neurona no está conectada a todas las neuronas de la capa anterior, sino solo a un pequeño número de ellas. Reduciendo los parámetros del modelo y acelerando la convergencia. Otra ventaja es el **reparto de pesos**, donde un grupo de conexiones comparte los mismos pesos, lo que disminuye aún más la cantidad de parámetros necesarios. Por último, se realiza una **reducción dimensional median-**

te *downsampling*, donde una capa de *pooling* utiliza el principio de correlación local de las imágenes para reducir su tamaño, disminuyendo la cantidad de datos mientras se conserva la información útil. Además, esto elimina características triviales, lo que reduce los parámetros adicionales.

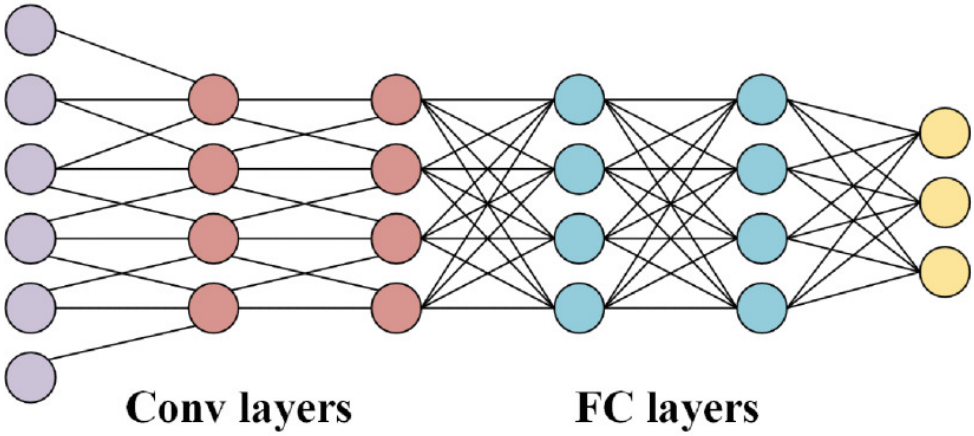

Figura 2.24: Comparación entre las capas convolucionales y las capas totalmente conectadas.

Variantes de las capas convoluciones en las CNN

Con el intento de mejorar las CNN se han desarrollado variantes que tienen el objetivo de mejorar el rendimiento y la adaptabilidad en tareas específicas de las convolucionales estándar. Algunas de estas variantes son las convolucionales deformables, las convolucionales de grupo y las convolucionales orientables, cada una con características y aplicaciones particulares.

Las **convolucionales deformables** [72] se introdujeron para abordar las limitaciones de las redes neuronales convencionales, cuya estructura geométrica es fija y las hace poco robustas frente a transformaciones geométricas. Esta ventaja es muy importante en la visión por computadora, ya que los modelos deben ajustarse a las deformaciones o cambios de pose de los objetos. Además, este tipo de convolución permite que los *kernels* enfoquen áreas de interés, generando mapas de características más representativos. Para conseguir esto se aprenden *offsets* en una capa de convolución paralela, que se añaden a la posición original del kernel para realizar transformaciones como traslación y rotación. Estos *offsets* se adaptan a las transformaciones del objeto y aumentan el campo receptivo del kernel.

En la Figura 2.25 se ilustra un ejemplo de funcionamiento de una convolución deformable, en este ejemplo se pueden observar los mapas de características de entrada

y de salida y la capa de convolución paralela donde se aplican los desplazamientos (*offsets*).

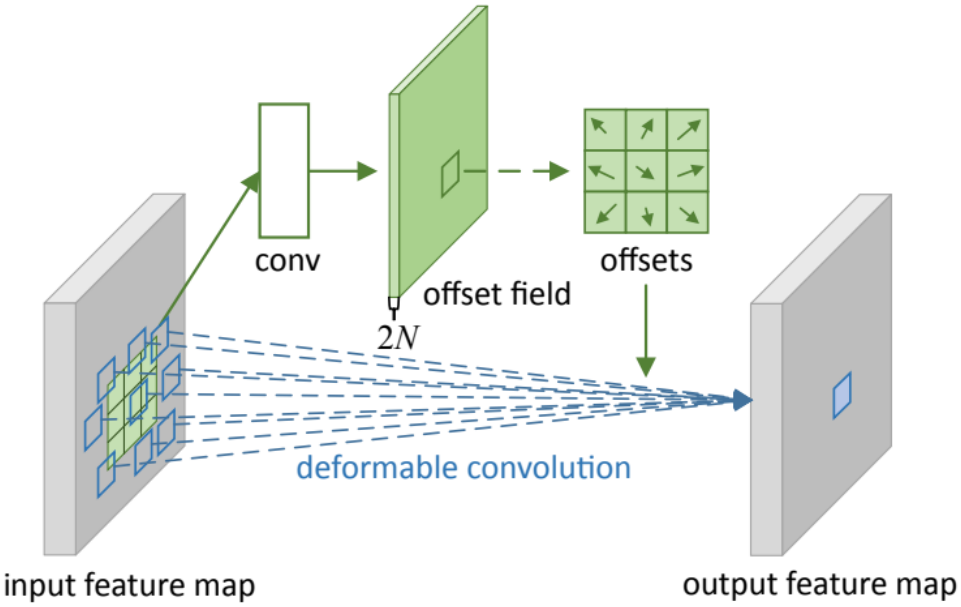

Figura 2.25: Ejemplo del funcionamiento de una capa convolucional deformable.

Otra variante es la **convolución de grupo** [73] que se define como una técnica de compresión de redes neuronales que reemplaza las capas convolucionales estándar con el fin de reducir el número de parámetros y operaciones computacionales, con una mínima pérdida de precisión. A diferencia de una convolución normal, en la que cada filtro se convoluciona con todos los canales, en la convolución de grupo, los canales de entrada se dividen en g grupos y cada grupo de filtros se aplica únicamente a su correspondiente grupo de canales de entrada. Al reducir los canales de entrada y los filtros en grupos, se reduce significativamente el número de parámetros utilizados.

Esta comparativa entre convolución estándar y convolución de grupos la podemos ver ilustrada en la Figura 2.26, donde podemos observar que en la convolución estándar (a) cada filtro se convoluciona con todos los canales de entrada, mientras que en la convolución agrupada (b), en este caso con dos grupos ($G = 2$), la mitad de los filtros se aplican a cada mitad de la entrada, reduciendo los parámetros utilizados a la mitad.

2.4.7. Redes Neuronales Recurrentes

Las **RNN** [74] son un tipo de ANN diseñadas para trabajar con datos secuenciales y series temporales. Su característica principal es la presencia de conexiones recurrentes, que forman ciclos dirigidos entre las unidades de la red. Estas conexiones

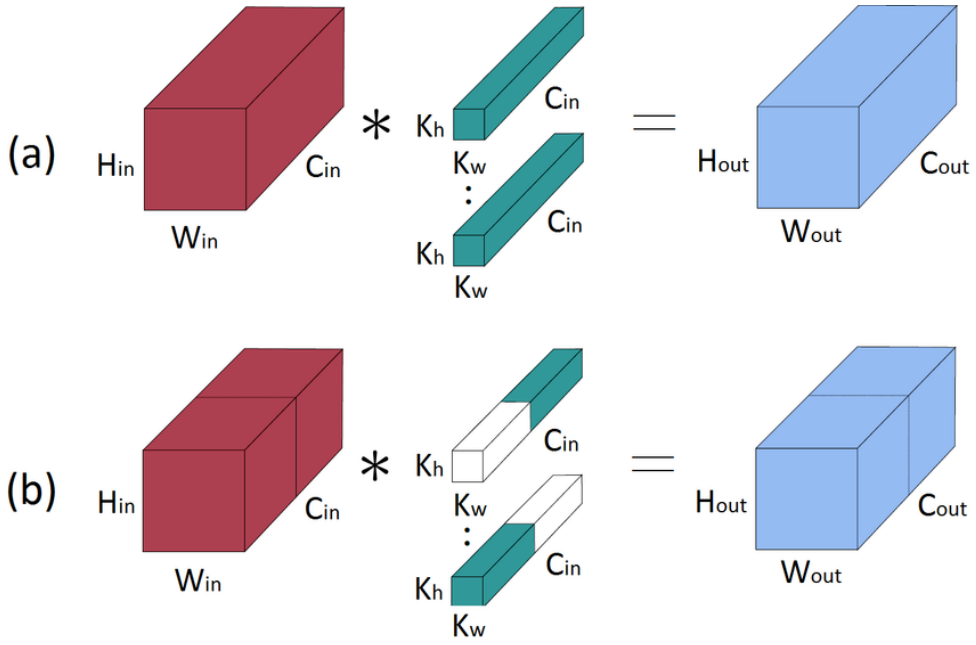

Figura 2.26: Comparación entre una convolución estándar (a) y una convolución agrupada (b).

permiten que la red mantenga un estado interno que actúa como memoria, lo que permite que pueda procesar datos que dependen de información previa en la secuencia.

Como se observa en la Figura 2.27, la estructura de una RNN simple consta de tres capas: una **capa de entrada**, una **capa oculta recurrente** y una **capa de salida**. Las unidades en la capa oculta están conectadas entre sí, lo que les permite almacenar y procesar información sobre señales complejas del pasado. En cada paso de tiempo, la salida de la red depende tanto de la entrada actual como del estado oculto generado en el paso anterior.

La **regularización** desempeña un papel importante en las RNN para evitar el sobreajuste y mejorar la capacidad de generalización del modelo. Las técnicas de regularización más usadas pueden ser L1, L2 y *dropout*.

A pesar de las ventajas que tienen las RNN, se han desarrollado variantes de RNN que mejoran su capacidad de modelar relaciones temporales más complejas y solucionar estos problemas. Algunas de estas variantes son:

LSTM

Una de las principales limitaciones de las RNN es el problema del **desvanecimiento del gradiente** [75], que ocurre durante el entrenamiento. En este proceso, el error

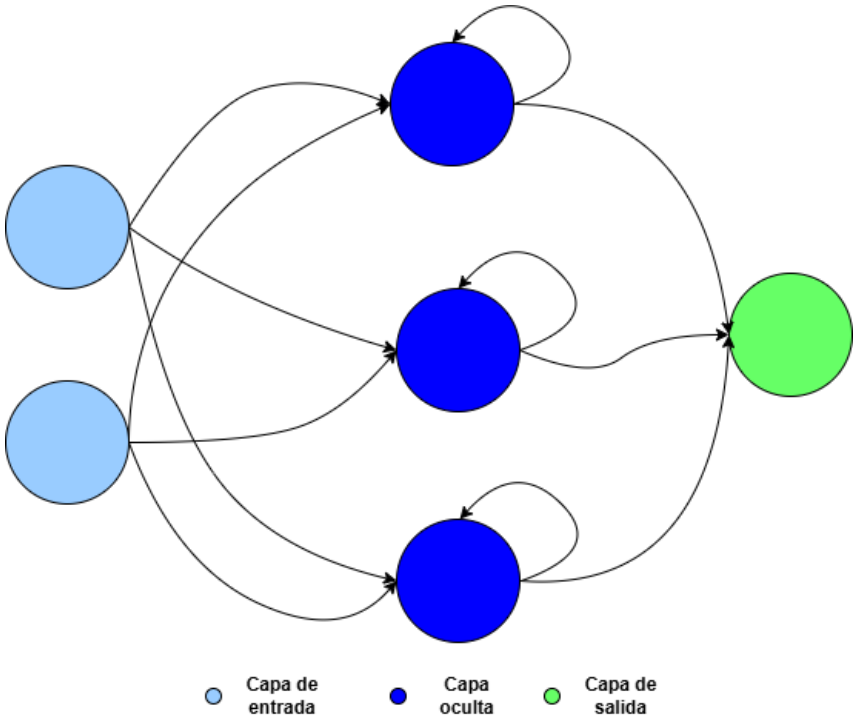

Figura 2.27: Arquitectura de una Red Neuronal Recurrente.

calculado al final de una secuencia debe propagarse hacia atrás en el tiempo para ajustar los pesos de la red mediante el cálculo de gradientes. Sin embargo, los valores de los gradientes tienden a disminuir exponencialmente si los pesos son menores que 1, provocando que las primeras capas de la red no reciban señales significativas para aprender.

Para resolver este problema, se desarrollaron las **LSTM** [76], un tipo de RNN en las que las neuronas tradicionales son reemplazadas por celdas LSTM, componentes muy importantes que incorporan una arquitectura interna compleja capaz de retener información durante períodos prolongados. Estas celdas gestionan dos tipos de memoria, una a **corto plazo** (estado oculto h_t) y una memoria a **largo plazo** (estado de la celda C_t).

Como podemos ver en la figura 2.28, las celdas LSTM utilizan tres tipos de puertas para regular el flujo de información: la **puerta de olvido** (f_t), que decide qué información eliminar del estado de la celda, la **puerta de entrada** (i_t), que regula qué nueva información debe almacenarse en el estado de la celda, y la **puerta de salida** (O_t), que combina la memoria a largo plazo con la memoria a corto plazo para generar una salida relevante en cada paso temporal.

En el Algoritmo 1 se describe una iteración a través de una celda LSTM, en primer lugar se realiza una multiplicación elemento a elemento con la puerta de olvido f_t,

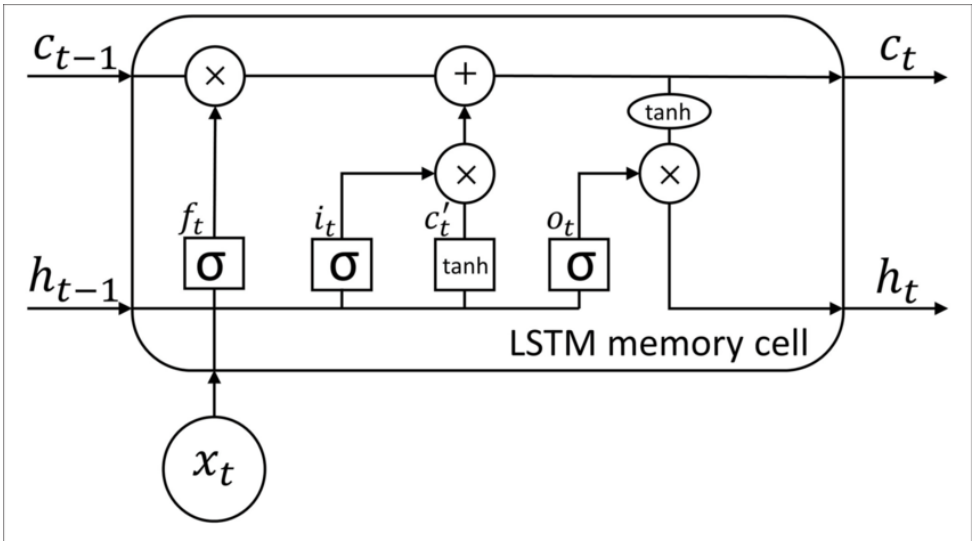

Figura 2.28: Arquitectura de una celda LSTM. Fuente [13]

más tarde, se le añade al estado actual de la celda la multiplicación entre el estado candidato $\tilde{c}_t$ y la salida de la puerta de entrada m_t. Por último se calcula la salida actual de la celda h_t como la combinación del estado de la celda c_t, la entrada x_t y la salida previa h_{t-1}. La iteración finaliza devolviendo el estado actual de la celda y la salida actual de la celda.

Algoritmo 1: Una iteración a través de una celda LSTM.

Input: c_{t-1}: Estado de la celda previo, h_{t-1}: Salida previa, x_t: Entrada actual

Output: c_t: Nuevo estado de la celda, h_t: Nueva salida de la celda

$c_t \leftarrow c_{t-1} \odot f_t$

$c_t \leftarrow c_t + (\tilde{c}_t \cdot m_t)$

$h_t \leftarrow$ Combinación de c_t, x_t y h_{t-1}

return (c_t, h_t)

Bi-LSTM

Las **LSTM bidireccionales** son una extensión de las LSTM que aborda las limitaciones de estas al considerar tanto el contexto pasado como el futuro en las tareas de modelado de secuencias. Mientras que los modelos LSTM tradicionales procesan los datos de entrada sólo hacia adelante, Bi-LSTM supera esta limitación entrenando el modelo en dos direcciones: hacia adelante y hacia atrás [77].

Como podemos ver en la figura 2.29, una red Bi-LSTM consiste en dos capas LSTM paralelas que una procesa la secuencia de entrada hacia adelante, mientras que la otra lo hace hacia atrás. Durante el entrenamiento, estas dos capas extraen ca-

racterísticas de forma independiente y actualizan sus estados internos basándose en la secuencia de entrada. Al incorporar información de ambas direcciones, los modelos Bi-LSTM captan un contexto más amplio y mejoran la capacidad del modelo para modelar dependencias temporales en datos secuenciales [17].

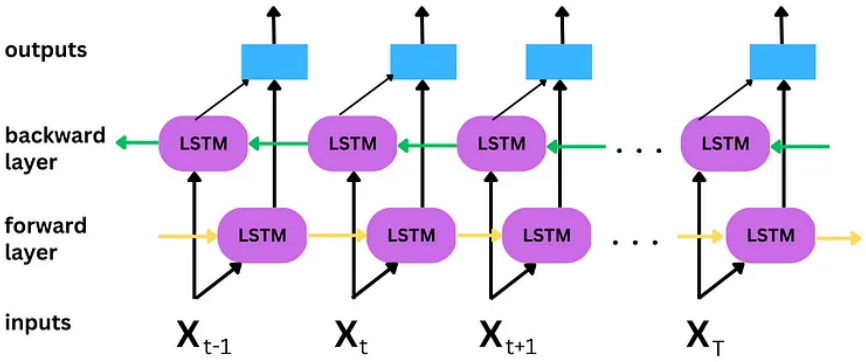

Figura 2.29: Estructura de una Red Bi-LSTM.

GRU

Otra variante de la arquitectura RNN es GRU (*Gated Recurrent Unit*) [78] que aborda el problema de la memoria a corto plazo y ofrece una estructura más simple en comparación con LSTM, ya que combina la puerta de entrada y la puerta de olvido en una única puerta de actualización. Como podemos ver en la figura 2.30, una unidad GRU consta de tres componentes principales [79]: una **puerta de actualización**, una **puerta de reinicio**, y el **contenido actual de la memoria ($\tilde{h}_t$)**. Estas puertas permiten a la red GRU actualizar y utilizar selectivamente la información de pasos temporales anteriores, permitiéndole capturar dependencias a largo plazo en secuencias.

La **puerta de actualización** (ecuación 2.29) determina cuánta información pasada debe ser retenida y combinada con la entrada actual en un paso de tiempo específico. Se calcula a partir de la combinación del estado oculto anterior h_{t-1} y la entrada actual x_t, seguida de una transformación lineal y una función de activación sigmoidea.

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \tag{2.29}$$

La **puerta de reinicio** (ecuación 2.30) decide cuánta información pasada debe

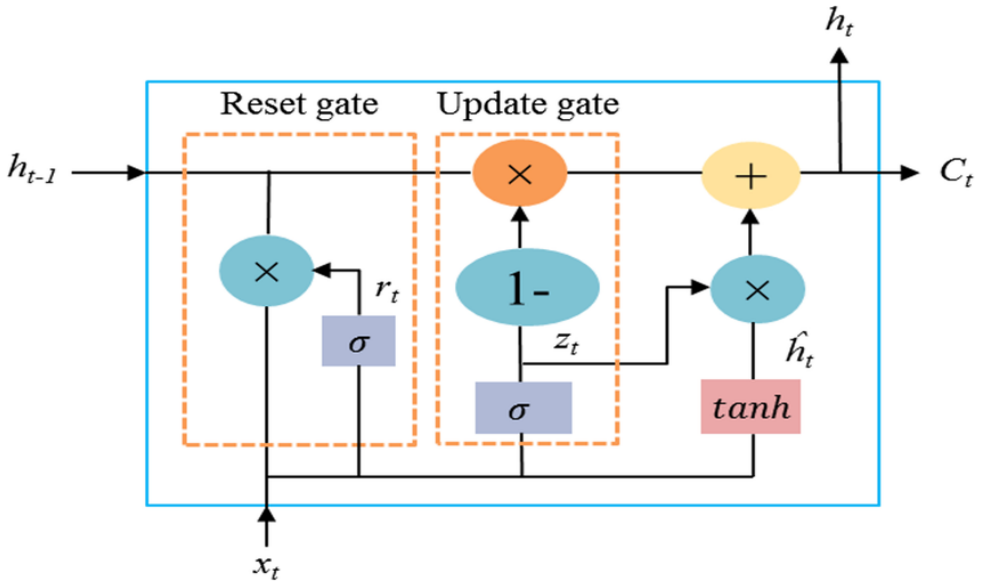

Figura 2.30: Estructura de una unidad GRU. Fuente [14]

ser olvidada. Se calcula de forma similar a la puerta de actualización, utilizando la combinación del estado oculto anterior y la entrada actual.

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \tag{2.30}$$

El **contenido de la memoria actual** (ecuación 2.31) se calcula basándose en la puerta de reinicio y la combinación del estado oculto previo y la entrada actual. El resultado pasa a través de una función de activación tangente hiperbólica para producir la activación candidata.

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t]) \tag{2.31}$$

Por último, el **estado final de la memoria** h_t viene determinado por una combinación del estado oculto anterior y la activación candidata (ecuación 2.32). La puerta de actualización determina el equilibrio entre el estado oculto previo y la activación candidata. Además, se puede introducir una puerta de salida c_t para controlar el flujo de información desde el contenido actual de la memoria a la salida (ecuación 2.33). Esta se calcula utilizando el estado actual de la memoria h_t y suele ir seguida de una función de activación.

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t \tag{2.32}$$

$$c_t = \sigma_o(W_o h_t + b_o) \tag{2.33}$$

Donde la matriz de pesos de la capa de salida es W_o y el vector de sesgo de la capa de salida es b_o.

BI-GRU

Bi-GRU [80] mejora la arquitectura GRU convencional mediante la integración de contextos del pasado y del futuro en tareas de modelado secuencial. A diferencia de GRU, que procesa exclusivamente secuencias de entrada hacia delante, la Bi-GRU gestiona secuencias en ambas direcciones. Para ello se utilizan dos capas paralelas, como muestra la figura 2.31.

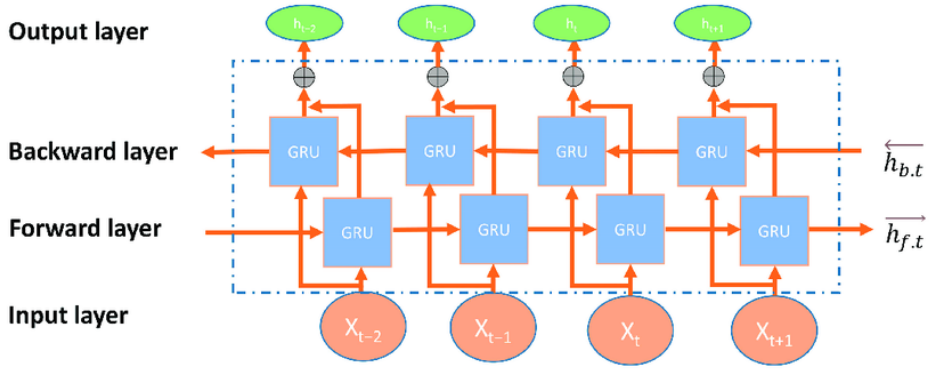

Figura 2.31: Estructura de un modelo Bi-GRU. Fuente [15]

2.4.8. Modelos híbridos

Los **modelos híbridos** [81] combinan elementos de diferentes modelos y han demostrado un potencial significativo para mejorar el rendimiento. Estas arquitecturas aprovechan las fortalezas de distintos enfoques para abordar problemas complejos del mundo real. Entre las principales categorías de modelos híbridos se encuentran:

- Combinación de varios **modelos supervisados**, como CNN+LSTM o CNN+GRU, que permite extraer características más relevantes y robustas, capturando eficazmente las dependencias temporales y espaciales dentro de los datos.
- Integración de **modelos generativos**, como la combinación de autocodificadores (AE) con redes generativas adversiales (GAN), que busca aprovechar las

fortalezas de ambos tipos de modelos y mejorar el rendimiento en diversas tareas.

- Combinación de **modelos generativos y supervisados** como GAN+CNN o AE+CNN, que combina las capacidades de ambos enfoques. Mejora el aprendizaje de características, el aumento de datos y la robustez del modelo.

Gracias a su capacidad para manejar datos secuenciales y modelar dependencias temporales, las RNN tienen un amplio rango de aplicaciones en diversos dominios como pueden ser: el aprendizaje de idiomas y la resolución de problemas de trayectoria. Además, se emplean en el modelado de sistemas dinámicos, la identificación de sistemas y el diseño de sistemas de control no lineales [82].

2.5. Reconocimiento de Actividades Humanas

El **Reconocimiento de Actividades Humanas** [83] ha sido un área de estudio de gran interés en las últimas décadas debido a su importancia en diversos ámbitos de aplicación. Entre ellos, se encuentran la salud, la monitorización remota, los videojuegos, la seguridad y vigilancia, así como la interacción entre humanos y computadoras.

El HAR se puede definir como la capacidad de reconocer o detectar la actividad que una persona está realizando en un momento determinado utilizando para ello la información proporcionada por diferentes **sensores**.

Estos sensores desempeñan un papel importante en el HAR. En la Figura 2.32 se ilustra el proceso mediante el cual se reconoce una actividad humana. Se proporciona un gesto corporal como entrada y los sensores capturan la información adquirida a través de este gesto, para que más tarde el algoritmo adecuado analice la información y determine el tipo de actividad que se ha realizado [84].

Las dos técnicas más utilizadas [85] para conseguir el objetivo de reconocer la actividad humana son el **HAR basado en visión** y el **HAR basado en sensores**. Se pueden clasificar como se muestra en la Figura 2.33.

2.5.1. Aplicaciones del Reconocimiento de Actividades Humanas

Los sistemas de HAR pueden identificar una amplia variedad de actividades, que van desde acciones simples hasta comportamientos complejos e interacciones. Estas

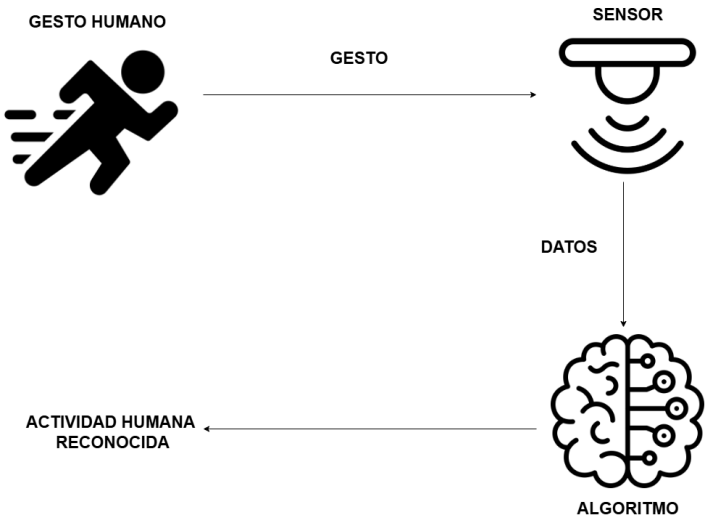

Figura 2.32: Estructura general de un sistema HAR.

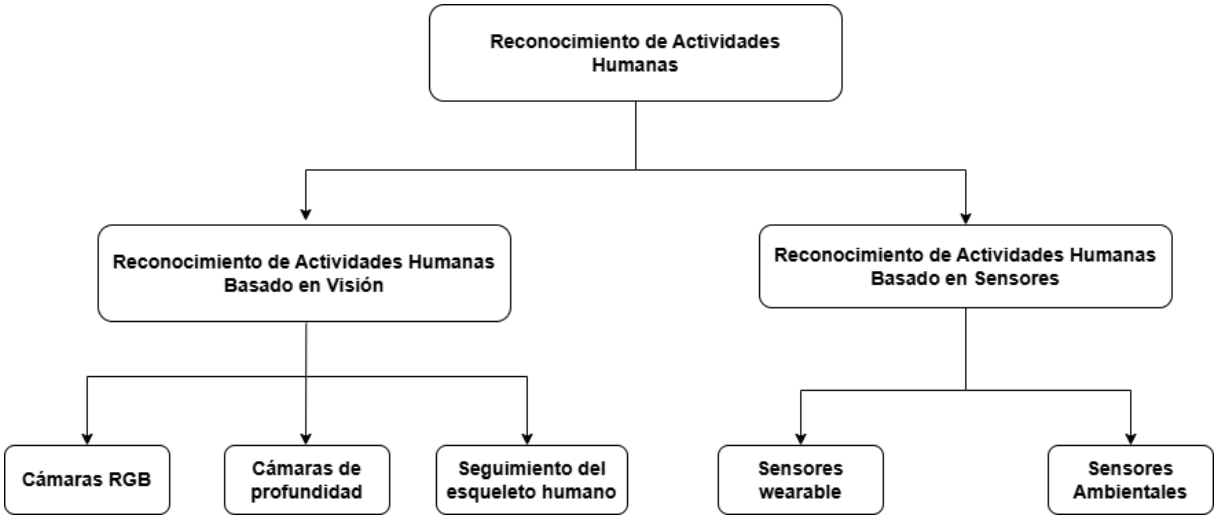

Figura 2.33: Clasificación de métodos de reconocimiento de actividades humanas basada en enfoques por visión y sensores.

se pueden clasificar en tres categorías principales: actividades **basadas en acción**, **basadas en movimiento** y **basadas en interacción** [83]. Estas tres categorías están enfocadas en HAR basado en sensores, que es el enfoque principal de este trabajo.

Actividades basadas en acción:

- **Reconocimiento de gestos**: Identificación de movimientos específicos realizados con las manos u otras partes del cuerpo. Esto puede ser útil para la interacción persona-máquina.
- **Reconocimiento de posturas**: Identificación de posiciones corporales, como estar de pie, sentado, acostado o caminando.
- **Detección de caídas**: Identificación de eventos en los que una persona cambia repentinamente de una posición normal a una posición de caída. Estos sistemas son esenciales para el cuidado de personas mayores.
- **Actividades de la vida diaria**: Reconocimiento de actividades cotidianas como comer, cocinar, dormir, bañarse, vestirse, usar el baño, etc.
- **Reconocimiento de comportamiento**: Inferencia del comportamiento de una persona en un entorno específico. Esto puede ser útil en centros de cuidados de ancianos, hogares inteligentes y tiendas.

Actividades basadas en movimiento:

- **Seguimiento (*Tracking*)**: Rastreo de la trayectoria de una persona en un área determinada, tanto en interiores como en exteriores. Útil para aplicaciones como la navegación interior. Se puede realizar con tecnologías como WiFi o RFID (*Radio Frequency Identification*)
- **Detección de movimiento**: Identificación de la presencia o el movimiento de una persona en un espacio. Esta detección es esencial para la seguridad y la vigilancia.
- **Conteo de Personas**: Conteo del número de personas presentes en un área. Puede llegar a ser útil en lugar como tiendas o eventos.

Actividades basadas en interacción:

- **Interacción Humano-Objeto**: Reconocimiento de las interacciones entre personas y objetos, como manipular objetos de la vida diaria.

Otras actividades y consideraciones:

- **Actividades con múltiples sujetos**: Los sistemas intentan reconocer actividades realizadas por varias personas simultáneamente.
- **Actividades concurrentes**: Se busca reconocer cuando una persona realiza múltiples actividades al mismo tiempo, como leer y beber.
- **Actividades anormales**: La detección de actividades que se desvían de lo normal es importante en seguridad y atención médica.
- **Predicción de actividades futuras**: La capacidad de predecir la siguiente actividad de una persona es un objetivo en aplicaciones como la prevención de caídas.

2.5.2. Etapas del HAR

El proceso de reconocimiento de actividades humanas, como observamos en la Figura 2.34, se compone de **cuatro etapas principales** [51]. La primera de ellas es la **captura de señal de una actividad**, donde se seleccionan los dispositivos HAR que mejor se adecúen a la aplicación objetivo. Por ejemplo, para vigilancia de grupos de personas se utilizan cámaras, mientras que para el monitoreo de actividades diarias de una persona se prefiere un sensor.

La siguiente etapa es el **preprocesamiento de los datos**, en la cual se realiza la limpieza de la información para eliminar el ruido, mejorar las imágenes, normalizar y segmentar los datos, etc.

En tercer lugar, se encuentra el **reconocimiento de actividad basado en IA**, etapa en la que se entrena el modelo HAR utilizando ML o DL. Debido a la extracción de características automática y la reutilización del conocimiento mediante el aprendizaje por transferencia, los modelos de DL son los más utilizados en este ámbito. Además, los modelos híbridos de DL son útiles para identificar características espaciales y temporales de manera más efectiva.

Por último, la **interfaz de usuario** para la gestión de HAR representa la etapa en la que un modelo, una vez entrenado, está listo para su aplicación o para poder realizar predicciones. Esta fase es especialmente difícil, ya que el modelo debe enfrentarse a datos reales, que pueden variar dependiendo de diferentes factores.

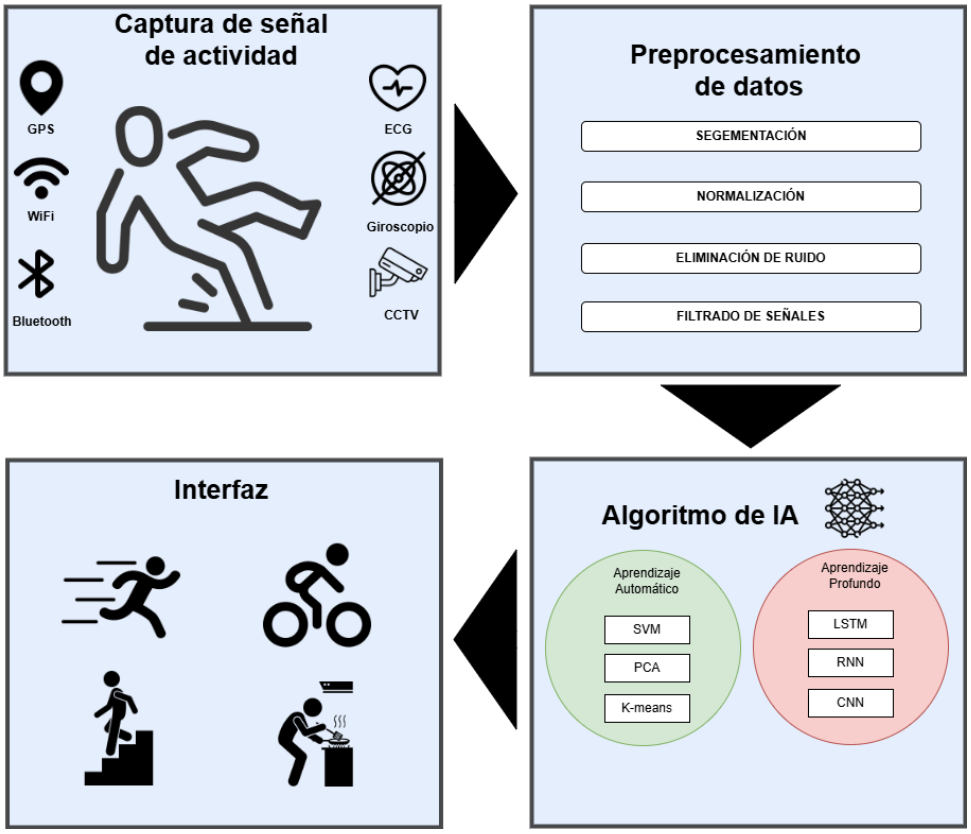

Figura 2.34: Etapas del HAR.

2.5.3. Sensores usados en HAR

Como ya se mencionó en la Subsección 2.5.1, este trabajo se centra en el HAR basado en sensores; por lo tanto, es importante conocer los diferentes tipos de **sensores** que se pueden usar y saber las razones por las que usarlos.

Por muchas razones, los sensores presentan ventajas [86] significativas frente a los demás métodos de reconocimiento de actividades humanas. Una de las principales ventajas es el **bajo consumo de energía**, ya que los sensores requieren menos recursos para funcionar. Además, su **tamaño reducido** facilita la integración en una amplia variedad de dispositivos, incluso en el cuerpo humano.

Otra ventaja destacable es su **bajo coste**, lo que permite una implementación más accesible. También ofrecen una gran **versatilidad**, ya que pueden integrarse en prendas de vestir, relojes inteligentes y otros dispositivos portátiles, mejorando la facilidad de uso y el transporte.

Además, los sensores permiten un **monitoreo virtual continuo**, sin estar limitados a un espacio de observación restringido como las cámaras. Esto posibilita la supervisión de actividades en diversas ubicaciones sin generar la sensación de vigilancia

constante. En este sentido, también se puede hablar de la **privacidad**, ya que los sensores no capturan imágenes visuales, resultando un método menos intrusivo para los usuarios.

Además de las ventajas mencionadas, los sensores utilizados en el HAR se pueden clasificar en función de su forma de despliegue y de su tipo [87].

Según su forma de despliegue, se distinguen principalmente los sensores vestibles y los sensores densos. Los **sensores vestibles** son aquellos que se llevan directamente en el cuerpo del usuario y miden sus movimientos y/o señales fisiológicas. Pueden estar integrados en dispositivos como smartphones, smartwatchs, gafas, ropa, cinturones, zapatos, etc. Algunos ejemplos son:

- **Giroscopio**: miden la velocidad angular del usuario.
- **Acelerómetros**: miden la aceleración del cuerpo y son los sensores más comunes para monitorizar movimientos repetitivos como caminar, correr o subir escaleras.
- **Sensores de presión**: miden la fuerza ejercida por un fluido sobre una superficie.
- **Biosensores**: Monitorizan signos vitales como la presión arterial, la frecuencia cardíaca, la EEG (*Electroencephalogram*) y la ECG (*Electrocardiogram*).
- **Sensores de sistema de posicionamiento global (GPS)**: monitorizan actividades basadas en la ubicación.

Por otro lado, los **sensores densos o de objeto** se fijan a objetos para monitorizar las interacciones entre el usuario y los mismos. Incluyen sensores de contacto, detectores de movimiento, sensores de presión, conmutadores de contacto, sensores RFID y sensores de voz.

En cuanto a su tipo, los sensores se clasifican en **ambientales** e **híbridos**. Los sensores ambientales se colocan en el entorno del usuario para capturar su interacción con el ambiente y recogen datos sobre factores como la luz, la temperatura, la humedad o las características ambientales básicas. Por otra parte, los sensores híbridos combinan diferentes tipos de sensores, como los vestibles, los densos y los ambientales, con el objetivo de capturar información más completa sobre las actividades humanas mediante la fusión de datos.

Cabe destacar que los sensores vestibles y los densos no son excluyentes entre sí, y su combinación permite obtener mejores resultados de reconocimiento.

2.5.4. Limitaciones del HAR con sensores

A pesar de los avances logrados y las ventajas del HAR, presenta varias limitaciones [88] que pueden dificultar su implementación. Una de las principales dificultades radica en la extracción de características manual. Los métodos tradicionales dependen en gran medida de este proceso, que requiere conocimiento y experiencia humana. Además, las características extraídas manualmente suelen ser superficiales, como la media, la varianza o la amplitud, lo que resulta insuficiente para detectar actividades complejas.

Otra limitación importante en HAR radica en la necesidad de grandes volúmenes de datos etiquetados para entrenar modelos, especialmente en técnicas basadas en aprendizaje profundo. Sin embargo, en aplicaciones reales, los datos de actividad suelen ser no etiquetados, y el proceso de **etiquetado manual es costoso** y requiere mucho tiempo, lo que dificulta la implementación eficiente de estos modelos.

Los sensores corporales no siempre son adecuados para actividades que implican interacciones con el entorno. Por otro lado, los sensores de objetos y los sensores ambientales pueden ser más efectivos en estos casos, pero su despliegue suele ser más complejo y su funcionamiento puede verse afectado por las condiciones del entorno.

Finalmente, los modelos tradicionales enfrentan dificultades para reconocer actividades complejas o dependientes del contexto, ya que las características superficiales son útiles para detectar actividades simples como caminar o correr, pero resultan poco adecuadas para identificar actividades más complejas, como tomar un café. Para abordar estas actividades más avanzadas, sería necesario añadir información adicional, como el contexto, el comportamiento o incluso las emociones del usuario.

2.5.5. Datasets relevantes para HAR

En el HAR, los datasets etiquetados son esenciales para entrenar y evaluar modelos de aprendizaje automático. Los conjuntos de datos pueden clasificarse según la fuente de datos que recopilan. Como vimos en la sección 2.5, se pueden dividir en datasets basados en visión y en datasets basados en sensores.

A continuación, en la tabla 2.2 se muestran ejemplos representativos de datasets utilizados en el ámbito de HAR basado en visión.

En la tabla 2.4 se muestran algunos conjuntos de datos relacionados con el reco-

Conjunto de datos	modalidad del dataset	Año	Clases	Sujetos	Muestra	Precisión
UPCV [89]	Skeleton	2014	10	20	400	99.20 % [90]
ActivityNet [91]	RGB	2015	203	-	27208	94.7 % [92]
Kinetics-400 [93]	RGB	2017	400	-	306245	92.1 % [94]
AVA [95]	RGB	2017	80	-	437	83.0 % [96]
MSRDaily Activity3D [97]	RGB, Skeleton	2012	16	10	320	97.50 % [98]
N-UCLA [99]	RGB, Skeleton	2014	10	10	1475	99.10 % [100]
UTD-MHAD [101]	RGB, Skeleton	2015	27	8	861	94.51 % [102]
NTU RGB+D 60 [103]	RGB, Skeleton	2016	60	40	56880	97.40 % [100]
PKU-MMD [104]	RGB, Skeleton	2017	51	66	10076	94.40 % [105]

Tabla. 2.2: Conjuntos de datos para HAR basado en visión. Fuente [16]

nocimiento de actividades humanas basado en sensores. Uno de los más importantes es el conjunto de datos **CASAS**, que se trata de un conjunto de datasets de hogares inteligentes. Este repositorio se inició obteniendo los datos de los sensores de un apartamento ubicado en el campus de la **Universidad Estatal de Washington**, compuesto por un baño, una sala de estar, tres dormitorios y una cocina. Este apartamento contaba con una gran variedad de sensores (sensores de movimiento, sensores de puerta, sensores de agua caliente y fría, sensores de temperatura, sensores de uso de electricidad, sensores de agitación, giroscopios, etc.) que servían para reconocer una gran variedad de actividades diarias [106]. Un ejemplo de un dataset del repositorio CASAS se muestra en la tabla 2.3.

Fecha y hora	Sensor	Estado del sensor	Actividad
2015-07-01 00:00:00.24076	OfficeAChairA	ON	Work_At_Desk="begin"
2015-07-01 00:00:01.932692	OfficeAChairA	OFF	-
2015-07-01 00:00:02.486752	OfficeAChairA	ON	-
2015-07-01 00:00:04.545731	OfficeAChairA	OFF	-
2015-07-01 00:01:50.078962	OfficeAChairA	ON	-
2015-07-01 00:01:51.199188	OfficeAChairA	OFF	Work_At_Desk=.^{en}d"
2015-07-01 00:04:05.607239	BedroomABed	ON	Sleep
2015-07-01 00:04:06.733944	BedroomABed	OFF	Sleep
2015-07-01 00:07:57.008161	OfficeAChairA	ON	Work_At_Desk="begin"

Tabla. 2.3: Ejemplo de un dataset de CASAS etiquetado parcialmente.

Otro conjunto de datos relevante es **Opportunity**, en este caso 4 personas fueron monitorizadas en un mismo entorno durante 30 días. Este entorno simula un estudio con cocina, tumbona y acceso al exterior. Se registraban actividades del día a día como limpiar, despertarse, asearse, comer, preparar la comida, etc.

Conjunto de datos	Número de casas	Residentes	Número de sensores	Número de actividades
CASAS [107]	7	Múltiples	20-86	11
Kasteren [108]	3	Múltiples	14-21	14-16
Domus [109]	1	Individual	78	Sentimientos del usuario
ARAS [110]	2	Múltiples	20	27
HIS [111]	1	Mútiples	20-30	7
OPPORTUNITY [112]	1	Múltiples	72	15-20

Tabla. 2.4: Conjuntos de datos para HAR basado en sensores. Fuente [17]

Capítulo 3

OBJETIVOS

En este capítulo se exponen los objetivos que guían el desarrollo de este trabajo. En primer lugar, se presenta el **objetivo general** que establece la base del proyecto y su propósito principal. Posteriormente, este objetivo se descompone en un conjunto de **objetivos específicos**, diseñados para abordar de manera estructurada y precisa las distintas etapas del trabajo.

3.1. Objetivo general

El **objetivo general** de este trabajo es estudiar, diseñar e implementar un sistema, basado en Aprendizaje Automático, capaz de **clasificar actividades humanas** realizadas en un entorno doméstico, utilizando datos recopilados por sensores distribuidos en el hogar.

Este objetivo implica la preparación, preprocesamiento y etiquetado de un conjunto de datos, el desarrollo de modelos de Aprendizaje Automático adecuados para esta tarea, su entrenamiento y evaluación y, finalmente, la comparación de los resultados obtenidos para determinar el mejor enfoque para la detección de actividades humanas.

3.2. Objetivos específicos

El objetivo general de esta investigación se desglosa en varios objetivos más específicos que nos ayudarán a avanzar en el estudio y a abordar la problemática de

manera ordenada.

Adquisición de conocimientos en reconocimiento de actividades humanas y aprendizaje automático aplicado a datos de sensores mediante una revisión del estado del arte.

Para realizar este trabajo, es necesario adquirir una base sólida de conocimientos sobre el Reconocimiento de Actividades Humanas y el Aprendizaje Automático aplicado a datos de sensores. Este objetivo implica realizar una **revisión del estado del arte** que permitirá familiarizarse con los avances más relevantes en el campo de estudio y las técnicas utilizadas en proyectos similares. Este estudio facilita la selección de modelos de Aprendizaje Automático y, además, permite identificar las herramientas necesarias para cumplir con los objetivos del proyecto.

Estudiar y analizar el conjunto de datos para comprender la estructura, los tipos de sensores, así como examinar las anotaciones.

Una vez asentadas las bases teóricas relacionadas con el campo de investigación, se deberá realizar un **estudio y análisis de los conjuntos de datos** para comprender su estructura y las características que lo definen. Esto incluye identificar las variables que recopilan y el tipo de sensores involucrados, así como examinar las anotaciones de los conjuntos de datos *Ground Truth*. Este análisis permitirá familiarizarse con el contenido del dataset y, además, garantiza su adecuación para las tareas de HAR.

Preprocesar los datos de los sensores para realizar tareas de limpieza y analizar los datos perdidos.

Con una idea más concreta de la estructura y características del conjunto de datos, se realizará el **preprocesamiento del conjunto de datos** de los sensores, una etapa muy importante que garantiza la calidad y utilidad del conjunto de datos en el desarrollo del modelo de HAR.

Este objetivo implica llevar a cabo tareas de limpieza para eliminar errores, inconsistencias y datos ruidosos que pueden afectar negativamente al rendimiento del modelo. También se llevará a cabo el ajuste de las fechas y horas del conjunto de datos de los sensores, asegurando que estén sincronizados con los conjuntos de datos

Ground Truth. Este paso es de gran importancia para obtener un conjunto de datos alineado temporalmente que sea adecuado para las etapas posteriores.

Estudiar la segmentación temporal de los datos para dividirlos en ventanas de tiempo adecuadas para reconocer las actividades.

Una vez el conjunto de datos está totalmente preprocesado y listo para ser usado, se hará una **segmentación temporal** de los datos, esto permite dividir las señales continuas de los sensores en ventanas de tiempo adecuadas para su análisis. Este objetivo se centra en estudiar y definir la mejor estrategia de segmentación temporal, teniendo en cuenta las características de las actividades a reconocer y la naturaleza de los datos recogidos.

Se analizarán diferentes tamaños de ventanas para garantizar que las actividades puedan ser capturadas de manera precisa y completa dentro de cada segmento o ventana temporal. Este objetivo es imprescindible, ya que permite identificar información relevante y mejora la capacidad del modelo identificando patrones en los datos temporales.

Seleccionar y definir un algoritmo de reconocimiento de las actividades.

Una vez que el conjunto de datos ha sido segmentado temporalmente de manera adecuada, el siguiente paso es **seleccionar y definir el algoritmo de HAR** más eficaz. Para ello se explorarán y evaluarán diversos modelos de Aprendizaje Profundo. La estrategia que se llevará a cabo es comparar el rendimiento de diferentes arquitecturas, como Redes Neuronales Convolucionales, Redes Neuronales Recurrentes y modelos híbridos, revisados en la sección 2.4. Tras analizar los resultados obtenidos en cada caso, se seleccionará el modelo que demuestre mejores resultados en diferentes métricas de rendimiento.

Entrenar y validar el algoritmo definido en diferentes actividades.

Con el algoritmo previamente seleccionado, el siguiente paso es llevar a cabo su **entrenamiento y validación** utilizando el conjunto de datos preprocesado. Este proceso implica aportar al modelo una parte del conjunto de datos etiquetados, para que así aprenda a reconocer patrones asociados a las diferentes actividades. Después se

validará el modelo utilizando otra parte del conjunto de datos totalmente independiente para evaluar su precisión y capacidad para detectar y diferenciar las actividades humanas.

Optimizar el algoritmo para proponer mejoras y ajustes basados en los resultados obtenidos.

Una vez completado el entrenamiento y la validación del algoritmo, se procederá a su **optimización** con el fin de mejorar su rendimiento y adaptabilidad. Esta etapa se centrará en analizar los resultados obtenidos durante las fases previas para identificar posibles limitaciones, como errores en la predicción de ciertas actividades donde el modelo no logra generalizar correctamente. En base a este análisis, se implementan ajustes en los hiperparámetros, modificaciones en la arquitectura del modelo, etc. El fin de esta optimización es que el algoritmo de reconocimiento de actividades humanas pueda trabajar de manera correcta bajo diversas condiciones y escenarios.

Capítulo 4

MATERIALES Y MÉTODOS

En este capítulo se describen los **materiales utilizados** y los **métodos empleados** para alcanzar los objetivos de este trabajo. Se detalla el entorno de experimentación, las características del *hardware* utilizado, la selección del conjunto de datos, las técnicas de preprocesamiento aplicadas, el diseño y selección del modelo y los métodos de evaluación y validación para la selección del modelo.

4.1. Entorno de experimentación

En esta sección se describen los **recursos utilizados** para llevar a cabo el proyecto, desde las características técnicas del equipo hasta el entorno y el lenguaje de programación.

4.1.1. Especificaciones técnicas

En la tabla 4.1 se detallan las especificaciones técnicas del equipo empleado para llevar a cabo las diferentes tareas necesarias para alcanzar los objetivos establecidos en este proyecto.

4.1.2. Entorno y lenguajes de programación

A lo largo del desarrollo de este trabajo, ha sido imprescindible hacer uso de herramientas de programación para llevar a cabo tareas fundamentales, como el tratamiento

Componente	Especificación
Procesador (CPU)	Intel Core i9-14900HX
Memoria RAM	32GB DDR4
Unidad de Procesamiento Gráfico (GPU)	NVIDIA GeForce RTX 4060 Laptop 8GB
Almacenamiento	1TB SSD NVMe
Sistema operativo	Windows 11 Pro 23H2

Tabla. 4.1: Componentes del equipo con el que se llevaron a cabo las tareas necesarias en este trabajo.

de datos, la implementación de algoritmos y el análisis de resultados.

Para realizar estas tareas, entre otras, se ha utilizado **Python** [113], como lenguaje de programación principal. Se trata de una herramienta de programación ampliamente utilizada en el ámbito del aprendizaje automático y la ciencia de datos debido a su simplicidad, versatilidad y, debido a su comunidad, la gran cantidad de bibliotecas que ofrece, algunas de las bibliotecas usadas en este trabajo son:

- **NumPy** [114]: Es una biblioteca fundamental para el **cálculo numérico** en Python. Proporciona soporte para trabajar con matrices y vectores multidimensionales y una amplia variedad de funciones matemáticas para realizar operaciones sobre ellos. Además, actúa como base para otras bibliotecas en el ecosistema de Python.

- **Pandas** [115]: Es una biblioteca de Python diseñada para facilitar el trabajo con datos estructurados, proporcionando estructuras como *DataFrame* y *Series* que permiten la manipulación de datos etiquetados de forma eficiente. Es ideal para el **análisis de datos**, debido a que tiene herramientas intuitivas para tareas como fusión y agregación. Además, destaca en el manejo de datos faltantes utilizando *NaN* y funciones como *dropna* y *fillna*.

- **Matplotlib** [116]: Se trata de la biblioteca de **visualización de datos** más utilizada en Python, sobre todo ciencia de datos, y que es compatible con otras bibliotecas ya nombradas como NumPy y Pandas.

- **Tensorflow** [117]: Se trata de una herramienta muy versátil, creada originalmente por investigadores de Google, para la implementación de **modelos de aprendizaje automático**, especialmente para redes neuronales, que proporciona flexibilidad, escalabilidad y herramientas para facilitar su desarrollo y despliegue.

- **Scikit-learn** [118]: Es una biblioteca de código abierto de **aprendizaje automático para Python**. Incluye una fácil integración de métodos de aprendizaje automático como pueden ser: Clasificación, Regresión, Reducción de dimensionalidad, prepocesamiento de datos y selección de modelos.

Para trabajar con Python se ha utilizado **Pycharm** [119] como **IDE**, una herramienta muy conocida por su potencia y versatilidad. Destaca por su capacidad para crear y gestionar entornos virtuales como Virtualenv o Conda, que permiten mantener las dependencias del proyecto organizadas y aisladas, algo primordial para proyectos más complejos. Otro aspecto en el que destaca es en su sistema integrado de control de versiones. En este trabajo, se ha utilizado un repositorio en GitHub que facilita el seguimiento de cambios y ver el código empleado en las diferentes tareas del trabajo.

Para conseguir ejecutar los modelos en el equipo descrito, ayudándonos de la GPU, es necesario instalar software adicional. En primer lugar, hay que instalar **CUDA** [120], se trata de una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA que permite el uso de GPUs para realizar cálculos, además facilita la ejecución de tareas computacionales intensivas en áreas como la ciencia de datos, inteligencia artificial, procesamiento de imágenes, etc. Por último, es necesario tener **cuDNN** (CUDA Deep Neural Network) [121], una biblioteca de aceleración de DNN desarrollada, también por NVIDIA. Está optimizada para ejecutarse en GPUs con CUDA, proporcionando mejor rendimiento y eficiencia en tareas de aprendizaje profundo.

Además, para la configuración de los hiperparámetros de los modelos empleados en este trabajo, se ha utilizado **YAML** [122] (*YAML Ain´t Markup Language*), un lenguaje de serialización de datos basado en Unicode, diseñado para ser legible por humanos y compatible con múltiples lenguajes de programación. Su estructura se basa en indentación lo que facilita su lectura y edición.

4.2. Descripción de los datasets utilizados

El conjunto de datos utilizado en este trabajo está compuesto por datos de sensores obtenidos de un piso equipado con una variedad de sensores diseñados para monitorizar actividades humanas en un entorno doméstico. Además, incluye información adicional sobre la intensidad de la señal recibida (**RSSI**) captada mediante un sistema de posicionamiento *beacon-anchor*.

4.2.1. Tipos de sensores y tecnologías utilizadas

El conjunto de sensores empleados abarca distintos tipos como sensores de movimiento (figura 4.1), que permiten detectar presencia y movimiento en las áreas donde están ubicados, sensores ambientales (figura 4.3), que registran temperatura, humedad y presión atmosférica, y sensores de apertura y cierre (figura 4.2), que detectan la apertura y cierre de cajones y puertas. Cada uno de estos dispositivos proporciona datos esenciales para analizar y reconocer actividades específicas dentro del hogar.

Figura 4.1: Sensor de movimiento utilizado.

Figura 4.2: Sensor de apertura y cierre de puertas y cajones utilizado.

Además de los sensores mencionados, el sistema empleado incluye un esquema de localización basado en tecnología BLE (*Bluetooth Low Energy*) compuesto por un dispositivo que actúa como *tag* y otro que actúa como *anchor*. Este sistema permite complementar los datos de los sensores con información de la localización dentro del entorno monitorizado.

Figura 4.3: Sensor ambiental utilizado.

Para este sistema, los residentes utilizan ***tags*** integrados en pulseras de actividad (figura 4.4) que emiten señales BLE de manera continua, y son captadas por dispositivos que actúan como *anchor*, en este caso una **Raspberry Pi** (figura 4.5), que registran la intensidad de las señales recibidas. Este valor se utiliza para estimar la distancia entre los dispositivos y permite determinar en qué zona del piso se encuentra cada residente en un momento dado. Esta información es aún más relevante en un entorno donde convivan varias personas, como es el caso de este trabajo, ya que facilita la asociación de las actividades detectadas por los sensores con un residente en particular.

Figura 4.4: Mi band 3 (tag).

Figura 4.5: Raspberry Pi 5 (ancla).

4.2.2. Distribución y ubicación de los dispositivos

Como se puede observar en la figura 4.6, los sensores se encuentran distribuidos por las diferentes habitaciones del piso, permitiendo que se capturen los comportamientos de los residentes en su vida cotidiana, y estos llevan la pulsera de actividad que manda las señales a la Raspberry Pi, situada en el comedor, que permiten medir la intensidad de la señal y localizar a la persona.

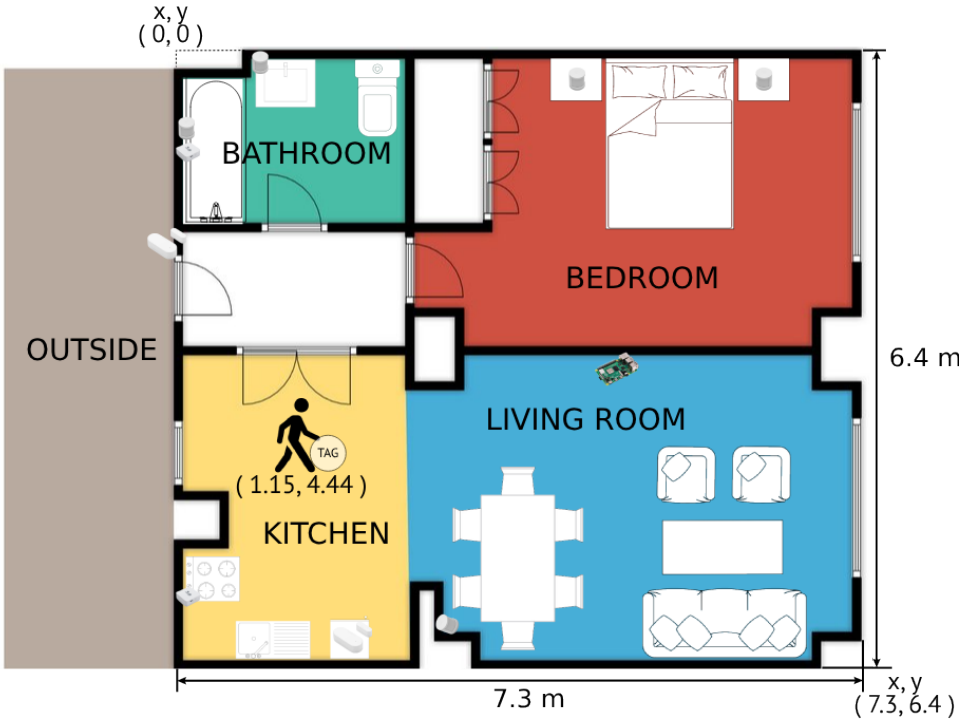

Figura 4.6: Plano del piso donde se recogieron los datos de los sensores.

4.2.3. Estructura de los datos recopilados

Los datos de los sensores y dispositivos se almacenan en formato **JSON** (*JavaScript Object Notation*) [123], un formato de datos simple, eficiente y ampliamente utilizado para la serialización e intercambio de datos entre aplicaciones, que ha ganado popularidad debido a su simplicidad y a su estrecha relación con las estructuras de datos utilizadas en la programación.

Como el conjunto de datos incluye múltiples tipos de sensores, no todos los registros contienen la misma estructura ni los mismos campos. Cada tipo de sensor captura información específica dependiendo de su funcionalidad. En la tabla 4.2 se presentan los diferentes sensores utilizados junto con sus identificadores correspondientes.

ID	Nombre sensor
52	Ambiental baño
53	Apertura y cierre del cajón de las medicinas
54	Apertura y cierre de la puerta principal
55	Movimiento cama
56	Movimiento ducha
57	Movimiento cepillo de dientes
75	Enchufe microondas
76	Apertura y cierre frigorífico
77	Movimiento comedor
78	Ambiental cocina
79	Enchufe lavadora

Tabla. 4.2: Conjunto de sensores usados junto a su ID.

A continuación se muestran ejemplos de registros de cada tipo de sensor para ilustrar su estructura:

- Como se observa en el listado 4.1, los **sensores ambientales** registran mediciones continuas de variables como la temperatura, la humedad y la presión atmosférica.

```
{
  "_id": {
    "$oid": "65d3cd25de8139cd4224d146"
  },
  "id_sensor": 52,
  "timestamp": 1708379428.116728,
  "temperature": 19.7,
  "humidity": 74.4,
  "pressure": 994
}
```

Listado 4.1: Ejemplo de un registro de un sensor ambiental

- Como se observa en el listado 4.2, los **sensores de movimiento** detectan la presencia y se activan cuando se registra movimiento en una zona cercana (campo "*state*": on) y se desactiva cuando deja de detectarse movimiento (campo "*state*": off).

```
{
  "_id": {
    "$oid": "65d3cd5dde8139cd4224d158"
    },
  "id_sensor": 55,
  "timestamp": 1708379484.854841,
  "state": "on"
}
```

Listado 4.2: Ejemplo registro sensor de movimiento

- Los **sensores de apertura y cierre** se utilizan para detectar cambios de estado en puertas y cajones, registrando si han sido abiertos o cerrados. Como se observa en el listado 4.3, estos sensores almacenan la misma información que los sensores de movimiento, con la diferencia de que el campo *state* se encuentra en *on* cuando el elemento se abre y en *off* cuando se cierra.

```
{
  "_id": {
    "$oid": "65d3cd87de8139cd4224d15f"
  },
  "id_sensor": 53,
  "timestamp": 1708379526.110686,
  "state": "on"
}
```

Listado 4.3: Ejemplo de un registro de un sensor de apertura y cierre

- Los **sensores de consumo energético** sirven para monitorizar el uso de los electrodomésticos, proporcionando información sobre el consumo de energía. Como se observa en el listado 4.4, el sensor recoge información sobre la corriente eléctrica medida en amperios (*current_a*), la potencia eléctrica consumida en vatios (*current_w*), el voltaje total de la red eléctrica en voltios (*total_voltage*), el consumo total de energía registrado por el sensor (*total_consumption*) y el consumo energético acumulado en el día actual (*today_consumption*).

```
{
  "_id": {
    "$oid": "65d477b9307305a9f6f12b82"
  },
  "id_sensor": 75,
  "timestamp": 1708423097.20396,
  "current_a": 6.12,
  "current_w": 1320.8,
  "total_voltage": 230,
  "total_consumption": 0.001,
  "today_consumption": 0.018
}
```

Listado 4.4: Ejemplo de un registro de un sensor de consumo energético

- El sistema de posicionamiento **RSSI** intenta aproximar la distancia entre los residentes y los distintos puntos de referencia dentro del hogar. Como se observa en el listado 4.5, en el conjunto de datos se almacena el identificador de la Raspberry Pi (*id_anchor*), el identificador de la pulsera de actividad (*id_beacon*) y el valor de la intensidad de la señal recibida (*rssi*), donde un valor cercano a 0 indica mayor proximidad entre la pulsera de actividad y el ancla, y valores más negativos representan una mayor distancia, con un máximo de -200.

```
{
  "_id": {
    "$oid": "65d477d1307305a9f6f12bc1"
  },
  "id_anchor": 8,
  "id_beacon": 2,
  "timestamp": 1708423121.193041,
  "rssi": -75
}
```

Listado 4.5: Ejemplo de un registro del valor RSSI

- Las **pulseras de actividad**, también conocidas como *beacon* o *tag*, proporcionan información adicional de la actividad física de los residentes. Como observamos en el listado 4.6, los registros de la pulsera de actividad registra el número

de pasos del usuario (*steps*) y el nivel de batería de la pulsera en porcentaje (*battery*).

```
{
  "_id": {
    "$oid": "65d5f1b2e1463c29e3666730"
  },
  "id_beacon": 2,
  "timestamp": 1708519858.634708,
  "steps": 204,
  "battery": 40
}
```

Listado 4.6: Ejemplo de un registro de la pulsera de actividad

Además de los campos particulares de cada tipo de sensor, existen varios campos que son comunes a todos los sensores. Estos campos incluyen el ***_id*** que se trata de un identificador único para cada registro, el campo ***id_sensor***, que indica el identificador del sensor que ha generado el dato y el campo ***timestamp*** que almacena la marca temporal del momento exacto en que se hizo el registro.

4.2.4. Conjuntos de datos *Ground Truth*

Los datasets de actividades marcadas por los residentes, conocidos como ***ground truth***, contienen información sobre las actividades realizadas por los usuarios dentro del hogar. Estos registros son muy importantes para la posterior asignación de etiquetas a los datos de los sensores, facilitando el entrenamiento de los modelos de reconocimiento de actividades.

Como se ve en el listado 4.7 y en el listado 4.8, en estos datasets se almacenan detalles sobre el **usuario** que realiza la actividad (*user*), el **intervalo de tiempo** en el que la actividad comenzó (*date_init*) y terminó (*date_end*), el tipo de actividad (*tag*) e **información adicional** (*additional_info*) proporcionada por el usuario que puede servir para ajustar el intervalo de tiempo en caso de error.

```
{
  "_id": {
    "$oid": "660d320fbdc4eab8625dfd54"
  },
  "user": "vanessa",
  "date_init": {
    "$numberLong": "1712140540933"
  },
  "date_end": {
    "$numberLong": "1712140813621"
```

```
  },
  "tag": "Ducha",
  "additional_info": ""
},
{
  "_id": {
    "$oid": "660d89c9bdc4eab8625dfd55"
  },
  "user": "vanessa",
  "date_init": {
    "$numberLong": "1712143343693"
  },
  "date_end": {
    "$numberLong": "1712163268234"
  },
  "tag": "Ejercicio",
  "additional_info": "Se␣me␣ha␣olvidado.␣aproximadamente␣hace␣15␣minutos␣"
}
```


Listado 4.7: Ejemplo de registros en el dataset de actividades de un residente


```
{
  "_id": {
    "$oid": "660cf794bdc4eab8625dfd4e"
  },
  "user": "llopez",
  "date_init": {
    "$numberLong": "1712124641448"
  },
  "date_end": {
    "$numberLong": "1712125839665"
  },
  "tag": "Comer",
  "additional_info": "Desayuno␣Café"
},
{
  "_id": {
    "$oid": "660cf795bdc4eab8625dfd4f"
  },
  "user": "llopez",
  "date_init": {
    "$numberLong": "1712124394657"
  },
  "date_end": {
    "$numberLong": "1712124411808"
  },
  "tag": "Medicación",
  "additional_info": ""
}
```


Listado 4.8: Ejemplo de registros en el dataset de actividades de otro residente

Los datos de estos datasets fueron recopilados durante un periodo de **14 días**, comprendido entre el 2 y el 16 de abril de 2024. Durante este tiempo, se registraron actividades como dormir, comer, tomar medicamentos, cepillarse los dientes, ducharse y hacer ejercicio.

4.3. Preprocesamiento de datos

Antes de pasar al uso de los modelos, hay que realizar un preprocesamiento de los datos obtenidos, algo que garantiza la coherencia y utilidad del conjunto de datos en el modelo de aprendizaje. En primer lugar, se preparan los datos para tener un dataset más consistente y, más tarde, se aplican transformaciones específicas para su uso en las diferentes redes neuronales, como normalización y codificación de variables.

4.3.1. Preprocesamiento de los datasets

Lo primero que se hizo fue ajustar las marcas temporales de los datasets, tanto de las actividades como de los sensores, convirtiendo el campo *timestamp* en fechas con un formato más legible. Después se filtraron los registros del conjunto de sensores para incluir solamente las mediciones realizadas durante las dos semanas en las que se marcaron las actividades.

Además, para simplificar el problema en un primer momento, **se separaron los datos de los sensores de las mediciones de RSSI** y de los datos de la pulsera de actividad, aunque luego más tarde se tuviesen esos datos en cuenta para obtener el campo *location* del conjunto de datos de los sensores.

Uno de los pasos más importantes era **ajustar las fechas** del dataset de actividades para solucionar pequeños fallos a la hora de anotarlas. Para conseguir esto se generaron gráficos que ayudaron a ajustar las fechas de las actividades, teniendo en cuenta tanto los datos de los sensores como las anotaciones en el campo de *adittional_info*. Algunos ejemplos de estos gráficos generados se pueden ver en las figuras 4.7, 4.8, 4.9 y 4.10.

El siguiente ajuste que se hizo fue el **borrado de los datos del sensor de consumo energético de la lavadora**, ya que no aportaba información importante para ninguna de las actividades que se van a detectar. Con el objetivo de etiquetar los datos de los sensores, se **añadió el campo *activity*** y para facilitar este proceso se unieron los dos datasets de actividades en un único conjunto de datos, ordenado por fecha. A partir de este nuevo dataset, se etiquetaron los registros de los sensores, considerando los intervalos de tiempo definidos en el dataset de actividades.

Por último, para analizar cada actividad más en detalle, se dividió el conjunto de datos en conjuntos más pequeños, separados por actividad y para cada una de ellas se eliminaron las mediciones de los sensores que resultaban irrelevantes, por ejemplo,

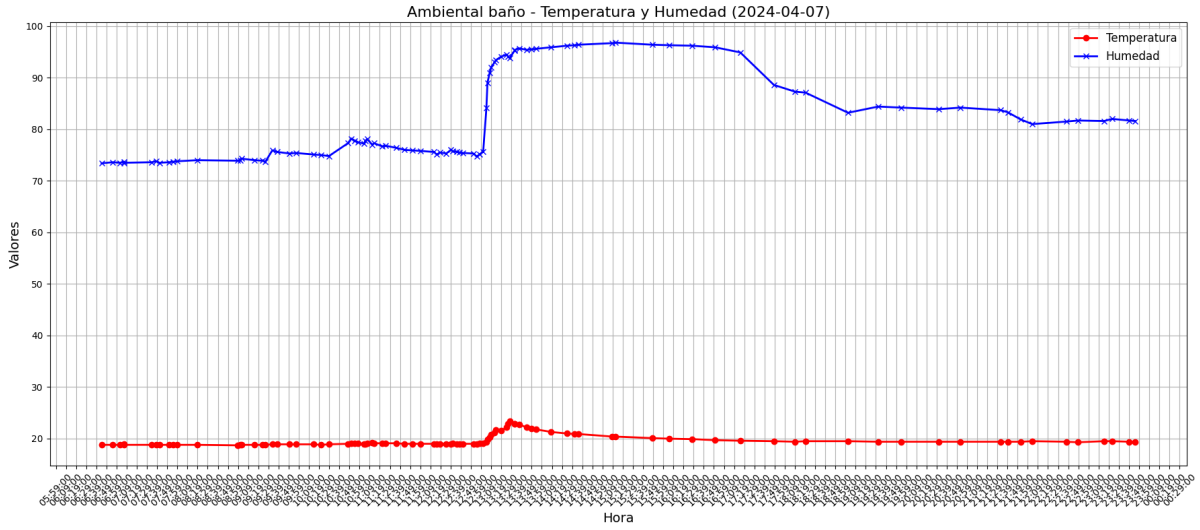

Figura 4.7: Ejemplo de gráfico usado para los sensores ambientales.

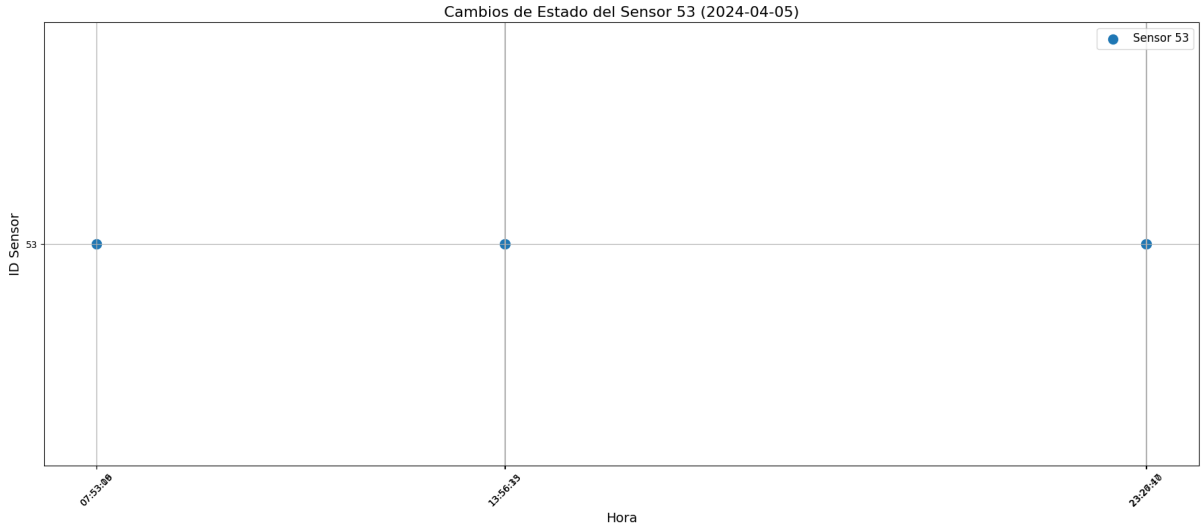

Figura 4.8: Ejemplo de gráfico usado para los sensores de apertura y cierre.

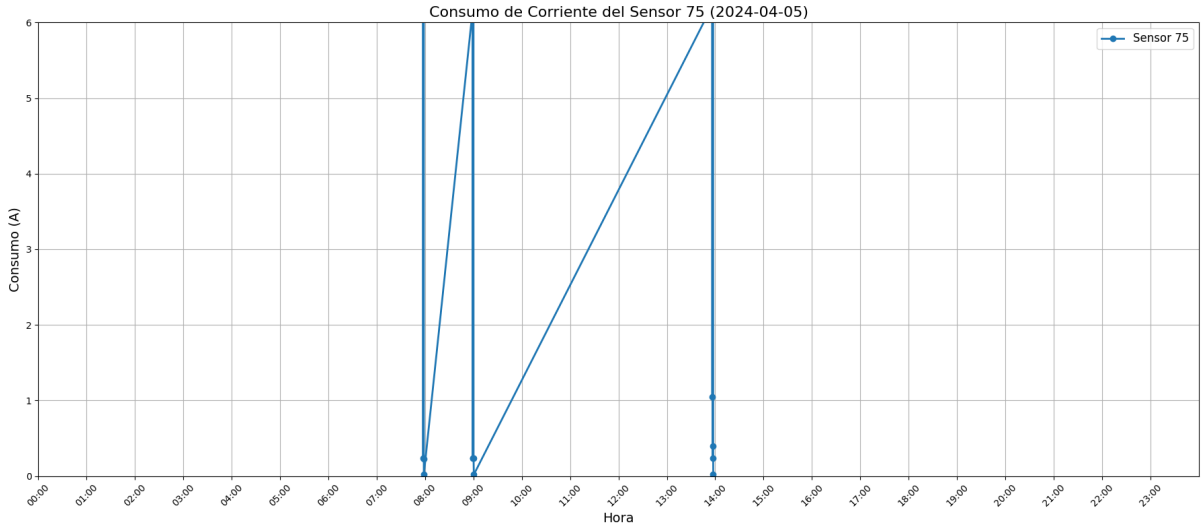

Figura 4.9: Ejemplo de gráfico usado para los sensores de consumo de energía.

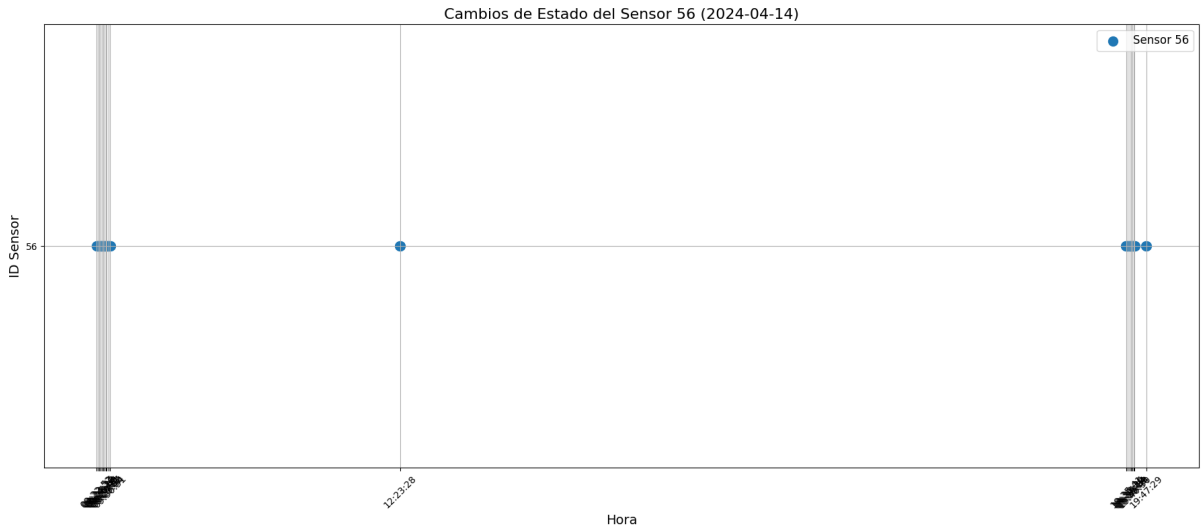

Figura 4.10: Ejemplo de gráfico usado para los sensores de movimiento.

para la actividad ‘cocinar‘ solo se tuvieron en cuenta los sensores que podrían aportar algún tipo de información relevante, como el ambiental de la cocina, el sensor de consumo del microondas y el sensor de apertura y cierre de la puerta del frigorífico.

Después de todos estos cambios, el conjunto de datos quedó como muestra el listado 4.9.


```
{
    "_id": {
        "$oid": "661a4db8a2fbb869dffe1f37"
    },
    "id_sensor": 57,
    "timestamp": "2024-04-13␣11:17:44",
    "state": "off",
    "activity": "Cepillado",
    "temperature": null,
    "humidity": null,
    "pressure": null,
    "current_a": null,
    "current_w": null,
    "total_voltage": null,
    "total_consumption": null,
    "today_consumption": null
},
{
    "_id": {
        "$oid": "661a4f00a2fbb869dffe2428"
    },
    "id_sensor": 56,
    "timestamp": "2024-04-13␣11:23:12",
    "state": "on",
    "activity": "Ducha",
    "temperature": null,
    "humidity": null,
    "pressure": null,
    "current_a": null,
    "current_w": null,
    "total_voltage": null,
    "total_consumption": null,
    "today_consumption": null
},
{
    "_id": {
        "$oid": "661a4f0fa2fbb869dffe246a"
    },
    "id_sensor": 56,
    "timestamp": "2024-04-13␣11:23:27",
    "state": "off",
    "activity": "Ducha",
    "temperature": null,
    "humidity": null,
    "pressure": null,
    "current_a": null,
    "current_w": null,
    "total_voltage": null,
    "total_consumption": null,
    "today_consumption": null
```

```
}
```

Listado 4.9: Ejemplo del conjunto de datos final sin la localización

Más tarde, **se añadió la localización** de cada uno de los residentes. Esto no solo permite detectar la actividad realizada, sino también identificar cuál de los residentes la llevó a cabo. Como muestra el listado 4.12, el conjunto de datos ahora incluye la ubicación de cada residente junto con una etiqueta que indica quién realizó la acción.

Para conseguir la localización de cada residente se ha utilizado el valor RSSI que anteriormente se había descartado. Con este valor se ha generado un conjunto de datos para cada uno de los residentes en los que, como muestran los listados 4.10 y 4.11, se registra la fecha del valor RSSI y la ubicación correspondiente. Posteriormente, a cada registro de los sensores en el dataset final se le asigna la localización del registro más cercano en el tiempo dentro de estos nuevos conjuntos de datos.

```
[
    {
        "id": "0",
        "timestamp": "2024-04-02␣23:54:33",
        "prediction": "Bedroom"
    },
    {
        "id": "1",
        "timestamp": "2024-04-02␣23:54:34",
        "prediction": "Bedroom"
    }
]
```

Listado 4.10: Dataset de localización para el residente 1.

```
[
    {
        "id": "0",
        "timestamp": "2024-04-02␣23:54:33",
        "prediction": "Bedroom"
    },
    {
        "id": "1",
        "timestamp": "2024-04-02␣23:54:34",
        "prediction": "Bedroom"
    }
]
```

Listado 4.11: Dataset de localización para el residente 2.

```
{
    "_id":{
        "$oid":"661a4da9a2fbb869dffe1efa"
    },
    "id_sensor":57,
```

```
    "timestamp":"2024-04-13 11:17:29" ,
    "state":"on",
    "activity":"Cepillado",
    "temperature":null,
    "humidity":null,
    "pressure":null,
    "current_a":null,
    "current_w":null,
    "total_voltage":null,
    "total_consumption":null,
    "today_consumption":null,
    "resident_1_location": "Bathroom",
    "resident_2_location": "LivingRoom&Kitchen",
    "resident":"Residente 1"
},
{
    "_id":{
        "$oid":"661a4db8a2fbb869dffe1f37"
    },
    "id_sensor":57,
    "timestamp":"2024-04-13 11:17:44",
    "state":"off",
    "activity":"Cepillado",
    "temperature":null,
    "humidity":null,
    "pressure":null,
    "current_a":null,
    "current_w":null,
    "total_voltage":null,
    "total_consumption":null,
    "today_consumption":null,
    "resident_1_location": "Bathroom",
    "resident_2_location": "LivingRoom&Kitchen",
    "resident":"Residente 1"
},
{
    "_id":{
        "$oid":"661a4f00a2fbb869dffe2428"
    },
    "id_sensor":56,
    "timestamp":"2024-04-13 11:23:12",
    "state":"on",
    "activity":"Ducha",
    "temperature":null,
    "humidity":null,
    "pressure":null,
    "current_a":null,
    "current_w":null,
    "total_voltage":null,
    "total_consumption":null,
    "today_consumption":null,
    "resident_1_location": "Bathroom",
    "resident_2_location": "LivingRoom&Kitchen",
    "resident":"Residente 1"
    }
```


Listado 4.12: Ejemplo del conjunto de datos final con la localización

4.3.2. Preprocesamiento de los datos para el modelo de aprendizaje automático

En primer lugar, aquellas columnas del dataset que no se consideran útiles se eliminan del dataframe, en este caso se eliminaron las columnas *pressure*, *total_voltage*, *id_sensor* y *today_consumption*, ya que no aportaban información relevante para la detección de las actividades en cuestión.

```
df = df[['timestamp','state','temperature','humidity','current_a','current_w','
    total_consumption','activity','location']]
```

Listado 4.13: Borrado de las columnas que no son relevantes para el modelo.

En cuanto a las demás características del dataset, que sí se consideran útiles para el modelo, se realizaron ciertas operaciones para que el modelo en cuestión pueda **procesarlas** de manera correcta. Primero se codifican las variables categóricas, como el campo *location*, asignando a cada localización un código, por ejemplo, la localización *Bedroom* podría ser la localización 0.

```
label_encoder = LabelEncoder()
df.loc[:, 'location'] = label_encoder.fit_transform(df['location']).astype('float64')
```

Listado 4.14: Conversión de las localizaciones a valores numéricos.

El campo *state* que solo tomaba valores *on* y *off* se transformó en valores binarios, asignando 1 y 0, respectivamente. Además, los campos numéricos como son *temperature*, *humidity*, *current_a*, *current_w* y *total_consumption* se normalizan para tener una media de 0 y una desviación estándar de 1, facilitando la convergencia durante el entrenamiento.

```
df.loc[:, "state"] = df["state"].map({"on": 1, "off": 0})

columnas_normalizar=['temperature','humidity','current_a','current_w','total_consumption'
    ]
scaler = StandardScaler()
df.loc[:, columnas_normalizar] = scaler.fit_transform(df[columnas_normalizar])
```

Listado 4.15: Conversión binaria de *state* y normalización de características numéricas.

Todos los sensores de este dataset tienen valores nulos en algunos de sus campos, por ejemplo, los sensores de movimiento no registran mediciones de temperatura ni humedad, por lo tanto, estos campos serían nulos para este tipo de sensores. Para solucionar este problema, se sustituyen los valores nulos por números muy grandes, intentando que el modelo los ignore durante el proceso de aprendizaje.

```
df.fillna(99999999999, inplace=True)
```

Listado 4.16: Procesamiento de los valores nulos.

Para que el modelo pueda procesar las fechas de una manera más efectiva, se vuelve a convertir el campo *timestamp* en una marca temporal.

```
df.loc[:, 'timestamp'] = df['timestamp'].view('int64') // 10 ** 9
```

Listado 4.17: Transformación de las fechas para que el modelo pueda procesarlas.

El campo que etiqueta la actividad (*activity*) se convierte en un formato categórico, primero se realiza el mismo proceso que el explicado con el campo *location* y, después, se utiliza la técnica *one-hot encoding* [124], con la que cada etiqueta de clase se representa como un vector binario, donde todos los elementos son cero, excepto el elemento que corresponde a la clase que se representa con un uno.

```
label_encoder = LabelEnconder()
df.loc[:, 'activity'] = label_encoder.fit_transform(df['activity']).astype('float64')

y = to_categorical(y)
```

Listado 4.18: One-hot encoding de las etiquetas.

Para conseguir capturar la información temporal se crean secuencias temporales mediante ventanas de cierto tamaño (función 4.20), que le permiten al modelo aprender patrones en función del tiempo. Por último, se calcula el tamaño del conjunto de prueba (función 4.21) en función de los días seleccionados para la validación, y se divide el conjunto de datos en un subconjunto de prueba y entrenamiento, teniendo en cuenta el tamaño de la prueba calculado.

```
X, y = crear_secuencias_temporales(df)

test_size = calcular_test_size(df)
return train_test_split(X, y, test_size=test_size, random_state=SEED)
```

Listado 4.19: Calculo del tamaño del conjunto de prueba y creación de secuencias temporales.

```
def crear_secuencias_temporales(df):
    caracteristicas = ['timestamp', 'state', 'temperature', 'humidity', 'current_a', '
        current_w', 'total_consumption', 'activity', 'location']

    secuencias = []
    etiquetas = []
    for i in range(len(df) - TAM_SECUENCIA_TEMPORAL):
        secuencia = df[caracteristicas].iloc[i:i + TAM_SECUENCIA_TEMPORAL].values
        etiqueta = df['activity'].iloc[i + TAM_SECUENCIA_TEMPORAL - 1]
```

```
        secuencias.append(secuencia)
        etiquetas.append(etiqueta)

    X = np.array(secuencias)
    y = np.array(etiquetas)
    return X, y
```

Listado 4.20: Función para crear secuencias temporales.

```
def calcular_test_size(df):
    fecha_min = df['timestamp'].min()
    fecha_max = df['timestamp'].max()
    total_dias = (fecha_max - fecha_min).days + 1

    return DIAS_PRUEBA / total_dias
```

Listado 4.21: Función para calcular el tamaño del conjunto de prueba.

4.4. Modelos para la experimentación

En esta sección se detallarán los hiperparámetros definidos en el archivo de configuración y la estructura del código utilizado, tanto la común entre los modelos como la particular de cada uno.

4.4.1. Hiperparámetros

La configuración de los diferentes hiperparámetros se ha hecho en el archivo *configuration.yaml* siguiendo la siguiente estructura:

```
localizacion: False

general:
  dias_prueba: 3
  tam_secuencia_temporal: 250
  modelo_a_ejecutar: "CNN"
  seed: 50

modelos:
  cnn:
    filters: 256
    kernel_size: 5
    pool_size: 2
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    epochs: 300
    patience: 15
```

```
  cnnbilstm:
    filters: 256
    kernel_size: 5
    pool_size: 2
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
  cnnbigru:
    filters: 256
    kernel_size: 5
    pool_size: 2
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
  bigru:
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
  bilstm:
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
  gru:
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
  lstm:
    dropout_rate: 0.4
    ff_dim: 128
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
  rnn:
    rnn_units: 256
    dropout_rate: 0.4
    ff_dim: 128
    pool_size: 2
    batch_size: 64
    embed_dim: 64
    epochs: 300
    patience: 15
```

Listado 4.22: Archivo *configuration.yaml*

Como se puede ver en el listado 4.22, en primer lugar se indica en el campo *localización* si se va a utilizar el dataset con la localización añadida (True) o no (False). A continuación, se definen variables generales como el número de días que se van a utilizar para el conjunto de test de la red neuronal, el tamaño de las secuencias temporales, el modelo que se va a ejecutar y el número que se utiliza como semilla, que garantiza la reproducibilidad de los experimentos. Por último, se definen los hiperparámetros para cada modelo utilizado, se pueden diferenciar en:

Hiperparámetros de la arquitectura

- ***Filters***: Número de filtros en las capas convolucionales, estos controlan cuántas características se detectan en la secuencia de entrada.
- ***Kernel_size***: tamaño del kernel en las Redes Neuronales Convolucionales.
- ***Pool_size***: tamaño de la capa de *pooling*.
- ***Dropout_rate***: valor que indica la probabilidad de hacer que aleatoriamente las neuronas dejen de funcionar para prevenir el sobreajuste.
- ***ff_dim***: dimensión de la capa totalmente conectada.
- ***embed_dim***: dimensión de la capa de embedding, determina el tamaño del vector que representa cada entrada.
- ***rnn_units***: número de neuronas en la capa recurrente.

Hiperparámetros del entrenamiento

- ***batch_size***: tamaño del lote, es decir, cantidad de ejemplos procesados antes de actualizar los pesos del modelo.
- ***epochs***: número de epocas, número de veces que el modelo entrenará el conjunto de datos.
- ***patience***: número de épocas sin mejora en alguna medida antes de detener el entrenamiento.

La selección de los modelos se ha centrado en el uso de redes neuronales, descritas en la sección 2.4. A lo largo de esta investigación, se han evaluado arquitecturas de las diferentes redes neuronales para determinar cuál ofrece mejor rendimiento y todos los modelos comparten una estructura base.

4.4.2. Estructura común del código

Todos los modelos comparten la misma estructura del código, únicamente se diferencian en las capas intermedias.

En primer lugar, se cargan los hiperparámetros específicos para ese modelo y se establece la configuración de la semilla aleatoria. Esta función se puede ver en el listado 4.24. Posteriormente, se registran los tiempos de inicio de la ejecución para, al acabar el entrenamiento, medir el tiempo final y obtener el tiempo total de este. Más tarde se preprocesan los datos, como se ha indicado en la sección 4.3.2 y se dividen los datos en conjuntos de entrenamiento y prueba.

Además, se define el modelo, la capa de entrada, donde se establece la dimensión de los datos que recibirá el modelo y la capa de normalización, esta se utiliza porque, al entrenar una red neuronal, las activaciones pueden tener valores muy diferentes entre sí, haciendo que el modelo tenga dificultades para mejorar. Con esta capa se evita esto, ya que se asegura de que las activaciones tengan una escala más equilibrada, normalizando cada muestra de entrada de manera independiente.

```
hiperparametros = config["modelos"]["nombre_modelo"]
establecer_semilla()
start_time = time.time()
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = preparar_datos(df)
num_clases = y_entrenamiento.shape[1]

model = Sequential()
model.add(Input(shape=(X_entrenamiento.shape[1], X_entrenamiento.shape[2])))
model.add(LayerNormalization(epsilon=1e-6))
```

Listado 4.23: Código común para los modelos antes de la definición de las capas particulares de cada uno.

```
def establecer_semilla():
    tf.keras.utils.set_random_seed(SEED)
    tf.config.experimental.enable_op_determinism()
```

Listado 4.24: Función para establecer la semilla aleatoria de Tensorflow.

A continuación de definir el modelo, se añaden las capas necesarias para cada uno de los diferentes modelos, como veremos en la sección 4.4.3.

Una vez definida la arquitectura de la red neuronal, los siguientes pasos son la configuración de las capas finales, la compilación, el entrenamiento y la evaluación del modelo.

```
model.add(Dropout(hiperparametros["dropout_rate"]))
```

```
model.add(Dense(hiperparametros["ff_dim"], activation="relu"))
model.add(Dropout(hiperparametros["dropout_rate"]))
model.add(Dense(num_clases, activation="softmax"))
```

Listado 4.25: Definición de las capas finales del modelo.

En el listado 4.25 se puede observar cómo se definen las capas finales del modelo. Se añaden capas *dropout* para evitar el sobreajuste, una capa densa oculta *dense* con la función de activación ReLU (vista en la subsección 2.4.1) para aprender relaciones no lineales en los datos de entrada y, por último, se añade una capa de salida con el número de neuronas igual al número de clases que se pueden clasificar cada registro, además, se utiliza la activación *softmax* para obtener las probabilidades de clasificación.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Listado 4.26: Compilación del modelo.

En el listado 4.26, se observa cómo se compila el modelo, y se ve que se usa la función de pérdida *categorical_crossentropy* (vista en la subsección 2.4.2) adecuada para problemas de clasificación multiclase. Además, se utiliza el optimizador Adam (estudiado en la subsección 2.4.3).

```
early_stopping = EarlyStopping(monitor='val_accuracy', patience=hiperparametros["patience
    "], restore_best_weights=True)
history = model.fit(X_entrenamiento, y_entrenamiento, epochs=hiperparametros["epochs"],
    batch_size=hiperparametros["batch_size"], validation_data=(X_prueba, y_prueba),
    callbacks=[early_stopping])
```

Listado 4.27: Early stopping y entrenamiento del modelo.

Como se observa en el listado 4.27, lo siguiente es la configuración del *Early Stopping*, donde la métrica a monitorizar es la precisión en la validación (*val_accuracy*). La paciencia en este caso está definida en el archivo de configuración e indica el número de épocas en las que que no se mejora el criterio de monitoreo antes de detener el entrenamiento. Después se entrena el modelo con los conjuntos de entrenamiento, con el número de épocas y el tamaño de lote indicado y se añade el *early stopping* configurado.

```
loss, accuracy = model.evaluate(X_prueba, y_prueba)
y_pred_prob = model.predict(X_prueba)
end_time = time.time()
execution_time = end_time - start_time
```

Listado 4.28: Evaluación del modelo, predicción sobre los datos de prueba y cálculo del tiempo de ejecución.

Por último, en el listado 4.28 podemos ver cómo se realiza la evaluación del modelo con el conjunto de prueba y se obtiene la pérdida y la precisión del modelo. También se realiza una predicción sobre los datos de prueba, útil para realizar posteriormente las matrices de confusión o las métricas ROC-AUC (vistas en la subsección 2.4.5). Por último, se calcula el tiempo de ejecución del modelo como medida de rendimiento.

4.4.3. Modelos implementados

Una vez definida la estructura común, se pueden detallar las **particularidades de cada modelo**. En las siguientes subsecciones se muestran las diferentes arquitecturas evaluadas en este trabajo.

Red Neuronal Convolucional

Como observamos en el lisado 4.29, en primer lugar, se añade al modelo tres capas convolucionales unidimensionales, acompañadas de capas de agrupación o *pooling*. Cada capa convolucional tiene un número de filtros y un tamaño de kernel definido en el archivo *configuration.yaml*. Además, utilizan la función de activación ReLU. Las capas de *pooling* tienen la funcionalidad de eliminar información redundante y resaltar las características más importantes de la salida de la capa convolucional.

Por último, se añade la capa final (*flatten*) cuyo propósito es transformar la salida multidimensional de la última capa convolucional en un vector unidimensional, permitiendo conectar con la salida de la red neuronal y con las capas densas encargadas de la clasificación final de las actividades.

```
1  model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
       kernel_size"], activation='relu'))
2  model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))
3
4  model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
       kernel_size"], activation='relu'))
5  model.add(MaxPooling1D(hiperparametros["pool_size"]))
6
7  model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
       kernel_size"], activation='relu'))
8  model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))
9
10 model.add(Flatten())
```

Listado 4.29: Capas de la Red Neuronal Convolucional.

Red Neuronal Recurrente

Sigue la misma estructura que las CNN, pero cambiando las capas convolucionales por capas recurrentes, como se observa en el lisado 4.30.

```
model.add(SimpleRNN(hiperparametros["rnn_units"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"]))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(SimpleRNN(hiperparametros["rnn_units"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"]))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(SimpleRNN(hiperparametros["rnn_units"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"]))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(Flatten())
```

Listado 4.30: Capas de la Red Neuronal Recurrente.

LSTM y GRU

Como se ve en los listados 4.31 y 4.32, la estructura de estas dos redes recurrentes es muy similar. En primer lugar, se encuentra una capa densa que facilita el aprendizaje de patrones dentro de los datos secuenciales.

A continuación, se encuentra la principal diferencia entre ambas redes que es la capa recurrente, ya que, en la LSTM se utiliza una celda de memoria que gestiona la información a largo plazo a través de la puerta de entrada, olvido y salida, como se estudió en la subsección 2.4.7. En cambio, en la arquitectura GRU se combinan las puertas de entrada y olvido en una única unidad, reduciendo la cantidad de parámetros.

Después de esta capa, ambas redes incluyen una capa de normalización con la misma intención de la capa de normalización explicada en la subsección 4.4.2.

Por último, se añade una capa de pooling, usada para reducir la cantidad de datos que maneja la red al final de las capas recurrentes.

```
model.add(Dense(hiperparametros["embed_dim"]))
model.add(LSTM(hiperparametros["embed_dim"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"]))
model.add(LayerNormalization(epsilon=1e-6))
model.add(GlobalAveragePooling1D())
```

Listado 4.31: Capas de la red neuronal LSTM.

```
model.add(Dense(hiperparametros["embed_dim"]))
model.add(GRU(hiperparametros["embed_dim"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"]))
model.add(LayerNormalization(epsilon=1e-6))
model.add(GlobalAveragePooling1D())
```

Listado 4.32: Capas de la red neuronal GRU.

Bi-LSTM y Bi-GRU

Al igual que los modelos anteriores, ambas arquitecturas comparten una estructura muy parecida y únicamente se diferencian en la capa recurrente utilizada, donde Bi-LSTM emplea celdas LSTM bidireccionales y Bi-GRU utiliza celdas GRU bidireccionales.

Además, se añade una capa densa inicial, cuyo número de neuronas es el doble que en los modelos que no son bidireccionales, ya que estas capas procesan la información en ambas direcciones, como se vio en la sección 2.4.7. Después se añade la capa recurrente, seguida de una capa de normalización que estabiliza las activaciones de la red. A continuación de esta capa se repite la combinación de capa densa y bidireccional, reforzando así el aprendizaje.

Por último, se añade al modelo una capa de pooling, que reduce la dimensionalidad y prepara los datos para las últimas capas de la red neuronal.

```
model.add(Dense(hiperparametros["embed_dim"] * 2))
model.add(Bidirectional(LSTM(hiperparametros["embed_dim"], return_sequences=True, dropout
    =hiperparametros["dropout_rate"])))

model.add(LayerNormalization(epsilon=1e-6))
model.add(Dense(hiperparametros["embed_dim"] * 2))
model.add(Bidirectional(LSTM(hiperparametros["embed_dim"], return_sequences=True, dropout
    =hiperparametros["dropout_rate"])))

model.add(GlobalAveragePooling1D())
```

Listado 4.33: Capas de la red neuronal LSTM bidireccional.

```
model.add(Dense(hiperparametros["embed_dim"] * 2))
model.add(Bidirectional(GRU(hiperparametros["embed_dim"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"])))

model.add(LayerNormalization(epsilon=1e-6))
```

```
model.add(Dense(hiperparametros["embed_dim"] * 2))
model.add(Bidirectional(GRU(hiperparametros["embed_dim"], return_sequences=True, dropout=
    hiperparametros["dropout_rate"])))

model.add(GlobalAveragePooling1D())
```

Listado 4.34: Capas de la red neuronal GRU bidireccional.

CNN-BiLSTM y CNN-BiGRU

Estos modelos, al tratarse de modelos híbridos (sección 2.4.8), son combinaciones de CNN y RNN, en este caso LSTM y GRU bidireccionales. Como se observa en los listados 4.35 y 4.36, en primer lugar, se encuentran las tres capas convolucionales acompañadas de las capas de pooling, al igual que en la CNN simple.

Antes de pasar a las capas recurrentes del modelo, se normalizan las activaciones de la red y se añade una capa totalmente conectada. A continuación, se añade la capa recurrente, seguida de una capa final de normalización y una capa densa, que prepara los datos para las capas finales de la red neuronal.

```
model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
    kernel_size"], activation='relu'))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
    kernel_size"], activation='relu'))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
    kernel_size"], activation='relu'))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(LayerNormalization(epsilon=1e-6))
model.add(Dense(hiperparametros["embed_dim"] * 2, activation="relu"))

model.add(Bidirectional(LSTM(units=hiperparametros["embed_dim"], return_sequences=False,
    dropout=hiperparametros["dropout_rate"])))

model.add(LayerNormalization(epsilon=1e-6))
model.add(Dense(hiperparametros["embed_dim"] * 2, activation="relu"))
```

Listado 4.35: Capas del modelo híbrido entre CNN y LSTM bidireccional.

```
model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
    kernel_size"], activation='relu'))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))

model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
    kernel_size"], activation='relu'))
model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))
```

```
6
7  model.add(Conv1D(filters=hiperparametros["filters"], kernel_size=hiperparametros["
       kernel_size"], activation='relu'))
8  model.add(MaxPooling1D(pool_size=hiperparametros["pool_size"]))
9
10 model.add(LayerNormalization(epsilon=1e-6))
11 model.add(Dense(hiperparametros["embed_dim"] * 2, activation="relu"))
12
13 model.add(Bidirectional(GRU(units=hiperparametros["embed_dim"], return_sequences=False,
       dropout=hiperparametros["dropout_rate"])))
14
15 model.add(LayerNormalization(epsilon=1e-6))
16 model.add(Dense(hiperparametros["embed_dim"] * 2, activation="relu"))
```

Listado 4.36: Capas del modelo híbrido entre CNN y GRU bidireccional.

4.5. Medidas y gráficos de rendimiento

Con el objetivo de **evaluar el rendimiento** de los modelos descritos anteriormente, se han utilizado distintas métricas y representaciones visuales, estudiadas en la sección 2.4.5, que pueden ayudar a analizar su comportamiento.

En primer lugar, se presentan las medidas de evaluación de cada modelo. A continuación, se muestran los gráficos de rendimiento y, por último, se habla sobre las gráficas de las comparaciones entre los modelos, que ayudan a determinar cuál ofrece mejores resultados.

Todos estos resultados, tanto métricas obtenidas y representaciones gráficas, se detallarán en el capítulo 5.

4.5.1. Medidas de rendimiento

Al finalizar la ejecución del modelo, se ejecuta la función 4.37, que muestra las siguientes medidas de evaluación:

- **Tiempo de ejecución**
- **Pérdida** (*loss*)
- **Precisión global** (*Accuracy*): porcentaje de predicciones correctas (ecuación 2.28).
- **F1-score**

- **Precisión**: porcentaje de predicciones correctas entre todas las predicciones positivas realizadas (ecuación 2.26).
- **Sensibilidad** o **Exhaustividad** (*Recall*)

```
def mostrar_resultados(y_true, y_pred, loss, accuracy, modelo, y_pred_prob, y_prueba):
    f1 = f1_score(y_true, y_pred, average='weighted')
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)

    print(f'{modelo} Loss:{loss:.4f}, Accuracy: {accuracy:.4f}')
    print(f'Test Accuracy:{accuracy * 100:.2f}%')
    print(f'F1 Score:{f1* 100:.2f}%')
    print(f'Precision:{precision* 100:.2f}%')
    print(f'Recall:{recall* 100:.2f}%')
```

Listado 4.37: Función que muestra las medidas de rendimiento del modelo.

4.5.2. Gráficos de rendimiento

Con el objetivo de obtener una mejor interpretación del comportamiento del modelo, se han generado estas representaciones gráficas, elaboradas con la biblioteca *Matplotlib* que se introdujo en la subsección 4.1.2.

- **Matriz de confusión**
- **Curva PR-AUC**: Muestra la relación entre las medidas precisión y sensibilidad (*recall*).
- **Errores por actividad**: gráfico de barras en el que se visualiza en qué actividades comete más errores el modelo.

4.5.3. Gráficos de comparación entre modelos

Para facilitar la elección entre los modelos usados, se han realizado las diferentes gráficas de comparación entre modelos:

- **Comparación de precisión en entrenamiento**: Muestra cómo evoluciona la precisión de cada modelo a lo largo de las épocas en el entrenamiento.
- **Comparación de precisión en validación**: Muestra cómo evoluciona la precisión de cada modelo a lo largo de las épocas en la validación.

- **Comparación de pérdida en entrenamiento**: Permite analizar la convergencia de los diferentes modelos y detectar posibles problemas como el sobreajuste en el entrenamiento.
- **Comparación de pérdida en validación**: Permite analizar la convergencia de los diferentes modelos y detectar posibles problemas como el sobreajuste en la validación.

Capítulo 5

RESULTADOS

Después de definir la metodología y los modelos empleados en este trabajo, en este capítulo se presentan los **resultados obtenidos en las distintas pruebas realizadas**. Se analiza el rendimiento de los modelos en dos conjuntos de datos: uno de ellos sin información de localización y otro que incorpora la ubicación de los residentes.

Primero, se evalúa el rendimiento de cada modelo en ambos conjuntos de datos, comparando métricas clave como la precisión, la puntuación F1 y el tiempo de entrenamiento. Más tarde, se comparan los diferentes enfoques para determinar cuál de ellos es el más adecuado y, por último, se lleva a cabo una optimización de la arquitectura seleccionada, afinando distintos hiperparámetros con el objetivo de mejorar su desempeño sin incrementar significativamente el coste computacional.

Durante todo el capítulo, los resultados se muestran de forma detallada mediante tablas y gráficos que permiten visualizar el impacto de cada uno de los ajustes realizados en los modelos.

5.1. Conjunto de datos sin localización

Como se ha indicado en la sección 4.3.1, primero se realizaron las pruebas en un conjunto de datos en el que no se tenía en cuenta la localización (listado 4.9), tratando de reconocer únicamente las actividades sin tener en cuenta qué residente realizaba la acción.

Los hiperparámetros que utilizan los modelos están detallados en la tabla 5.1. Para

la realización de las pruebas, únicamente se ajustarán los parámetros *patience* y el tamaño de la secuencia temporal (descritos en la sección 4.4.1), con el objetivo de obtener los valores óptimos para conseguir los mejores resultados en los modelos.

Modelo	filters	kernel_size	pool_size	dropout_rate	ff_dim	batch_size	embed_dim	rnn_units	epochs
cnn	256	5	2	0.4	128	64	-	-	300
cnnbilstm	256	5	2	0.4	128	64	64	-	300
cnnbigru	256	5	2	0.4	128	64	64	-	300
bigru	-	-	-	0.4	128	64	64	-	300
bilstm	-	-	-	0.4	128	64	64	-	300
gru	-	-	-	0.4	128	64	64	-	300
lstm	-	-	-	0.4	128	64	64	-	300
rnn	-	-	2	0.4	128	64	64	256	300

Tabla. 5.1: Hiperparámetros y sus valores para la ejecución de los modelos.

A continuación se muestran los resultados obtenidos de los modelos en el conjunto de datos sin localización. Para facilitar su lectura se resaltará en rojo el mejor resultado de cada modelo. Además, se incluirán las gráficas correspondientes al mejor resultado obtenido para cada caso.

5.1.1. CNN

En la tabla 5.2 se muestra el rendimiento de la **CNN**. Como se observa, los mejores resultados se obtienen cuando el tamaño de la secuencia temporal es 250, además con el parámetro *patience* = 20 se obtienen los mejores resultados con un 96.77 % de precisión. Mientras que, con el hiperparámetro *patience* = 15 se obtiene la pérdida con el valor más bajo (0.1596).

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.9133	0.9152	0.9133	0.9112	0.2913	30.98	46
15	**250**	0.9632	0.9636	0.9632	0.9610	0.1596	37.26	44
15	**350**	0.9561	0.9545	0.9561	0.9550	0.1698	36.06	33
20	**150**	0.9133	0.9154	0.9133	0.9128	0.3908	49.79	83
20	**250**	0.9677	0.9652	0.9677	0.9651	0.1987	64.51	85
20	**350**	0.9561	0.9545	0.9561	0.9550	0.1698	39.81	38
25	**150**	0.9133	0.9154	0.9133	0.9128	0.3908	52.92	88
25	**250**	0.9677	0.9652	0.9677	0.9651	0.1987	67.69	90
25	**350**	0.9561	0.9545	0.9561	0.9550	0.1698	44.15	43

Tabla. 5.2: Comparación de medidas de rendimiento del modelo CNN con diferentes valores de `patience` y tamaño de la secuencia temporal.

Las siguientes figuras muestran el rendimiento del que se ha considerado el mejor resultado de la CNN (tamaño de secuencia temporal = 250 y *patience* = 20). La figura 5.1 muestra los errores por actividad, la figura 5.2, muestra la matriz de confusión en el reconocimiento de actividades del modelo CNN y, por último, la figura 5.3, muestra la curva *precision-recall*.

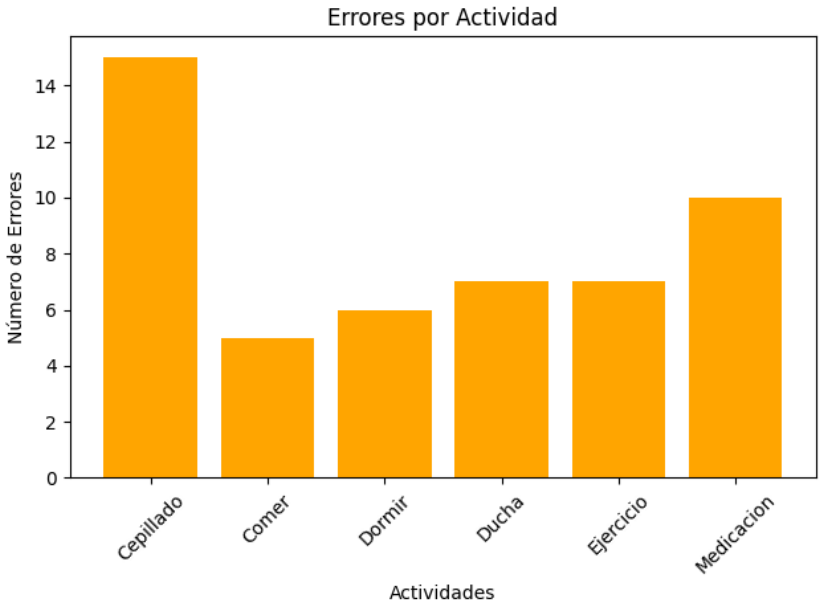

Figura 5.1: Errores por actividad obtenidos en el mejor resultado de la CNN.

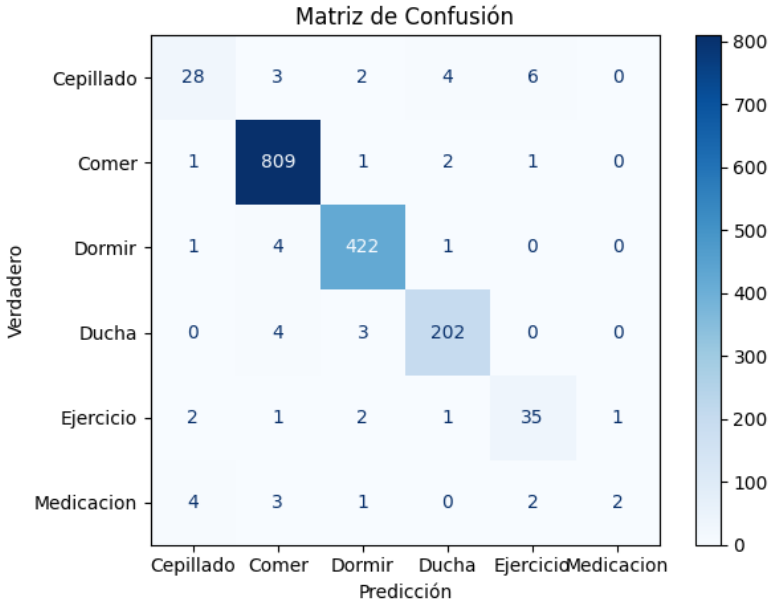

Figura 5.2: Matriz de confusión del mejor resultado de la CNN.

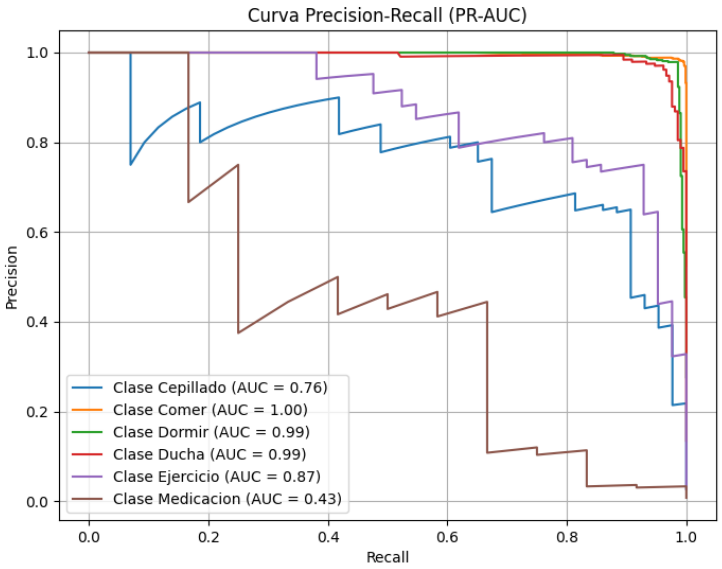

Figura 5.3: Curva *Precision-Recall* del mejor resultado de la CNN.

5.1.2. LSTM

En la tabla 5.3 se puede ver el rendimiento de la red neuronal recurrente **LSTM**. Se observa que con el tamaño de la secuencia temporal en 250 y con paciencia = 25 se obtienen los mejores resultados, llegando a alcanzar 95.09 % en la precisión global.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.8866	0.8771	0.8866	0.8751	0.3201	103.52	85
15	**250**	0.9283	0.9129	0.9283	0.9147	0.2302	174.61	106
15	**350**	0.9194	0.8925	0.9194	0.9021	0.2902	223.39	97
20	**150**	0.8936	0.8804	0.8936	0.8834	0.2966	149.15	118
20	**250**	0.9328	0.9206	0.9328	0.9175	0.2271	179.27	106
20	**350**	0.9253	0.8924	0.9253	0.9052	0.2291	376.66	180
25	**150**	0.8942	0.8833	0.8942	0.8857	0.3018	156,39	121
25	**250**	0.9509	0.9403	0.9509	0.9419	0.1613	395.40	235
25	**350**	0.9365	0.91	0.9365	0.92	0.2028	378.63	177

Tabla. 5.3: Comparación de medidas de rendimiento del modelo LSTM con diferentes valores de `patience` y tamaño de la secuencia temporal.

En las siguientes figuras se pueden ver de manera visual los resultados del mejor resultado obtenido en la red neuronal recurrente LSTM (tamaño de secuencia temporal = 250 y *patience* = 25). La figura 5.4 muestra los errores por actividad, la figura 5.5, muestra la matriz de confusión en el reconocimiento de actividades del modelo LSTM y, por último, la figura 5.6, muestra la curva *precision-recall*.

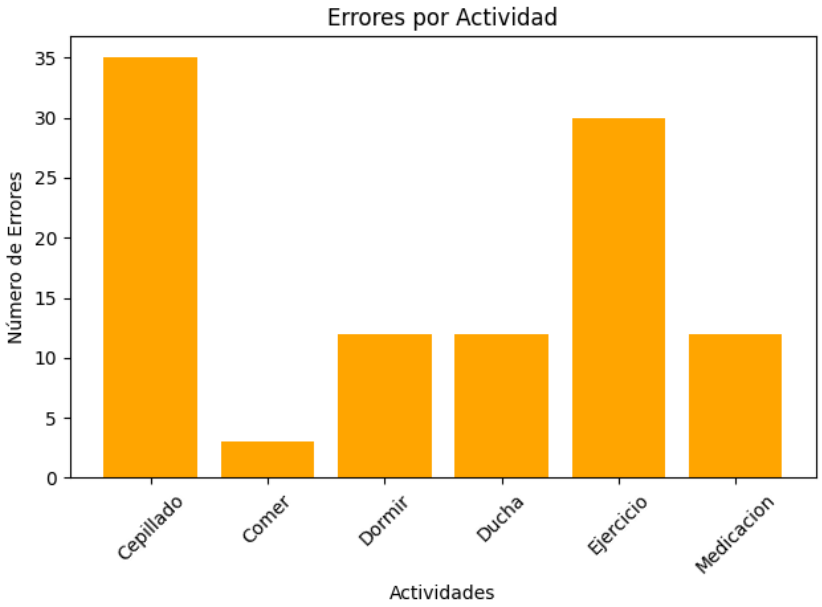

Figura 5.4: Errores por actividad obtenidos en el mejor resultado de la LSTM.

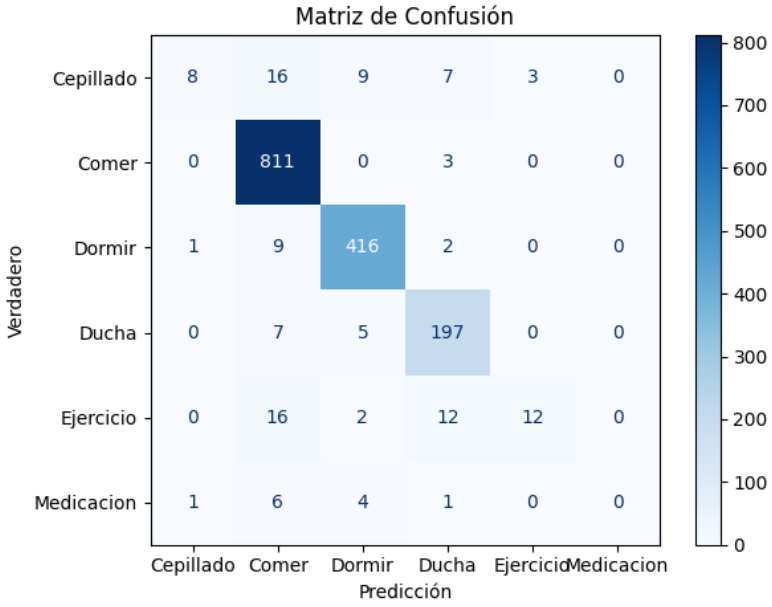

Figura 5.5: Matriz de confusión del mejor resultado de la LSTM.

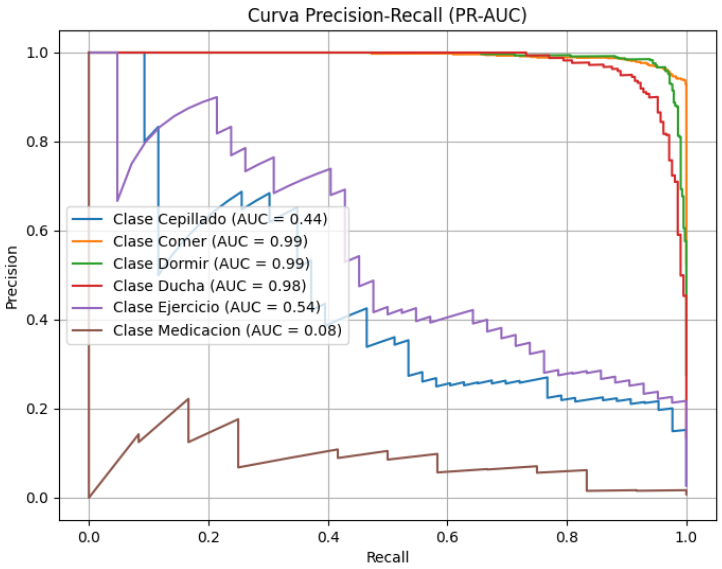

Figura 5.6: Curva *Precision-Recall* del mejor resultado de la LSTM.

5.1.3. GRU

En la tabla 5.4 se observa el rendimiento de la red neuronal recurrente **GRU**. Se puede ver que los mejores resultados se obtienen, nuevamente, con una secuencia temporal de 250 muestras, obteniendo resultados de un 92.38 % en *accuracy*. Al aumentar el tamaño de la secuencia temporal, empeoran los resultados, ya que no se mejora el rendimiento y aumenta el tiempo de ejecución. Además, se puede observar que el *Early Stopping* es efectivo con *patience* = 15 porque al aumentar este valor el rendimiento sigue siendo el mismo.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.8872	0.8741	0.8872	0.8773	0.2903	232.64	201
15	**250**	0.9238	0.9044	0.9238	0.9051	0.2432	282.30	174
15	**350**	0.9214	0.8961	0.9214	0.9055	0.2659	341.12	168
20	**150**	0.8872	0.8741	0.8872	0.8773	0.2903	238.73	206
20	**250**	0.9238	0.9044	0.9238	0.9051	0.2432	282.90	179
20	**350**	0.9214	0.8961	0.9214	0.9055	0.2659	352.64	173
25	**150**	0.8872	0.8741	0.8872	0.8773	0.2903	243.94	211
25	**250**	0.9238	0.9044	0.9238	0.9051	0.2432	295.03	184
25	**350**	0.9214	0.8961	0.9214	0.9055	0.2659	364.71	178

Tabla. 5.4: Comparación de medidas de rendimiento del modelo GRU con diferentes valores de `patience` y tamaño de la secuencia temporal.

En las siguientes figuras se pueden ver el rendimiento del mejor resultado del mo-

delo GRU (tamaño de secuencia temporal = 250 y *patience* = 15). Se pueden observar los gráficos de errores por actividad (figura 5.7), la matriz de confusión (figura 5.8) y la curva *Precision-Recall* del modelo GRU (figura 5.9).

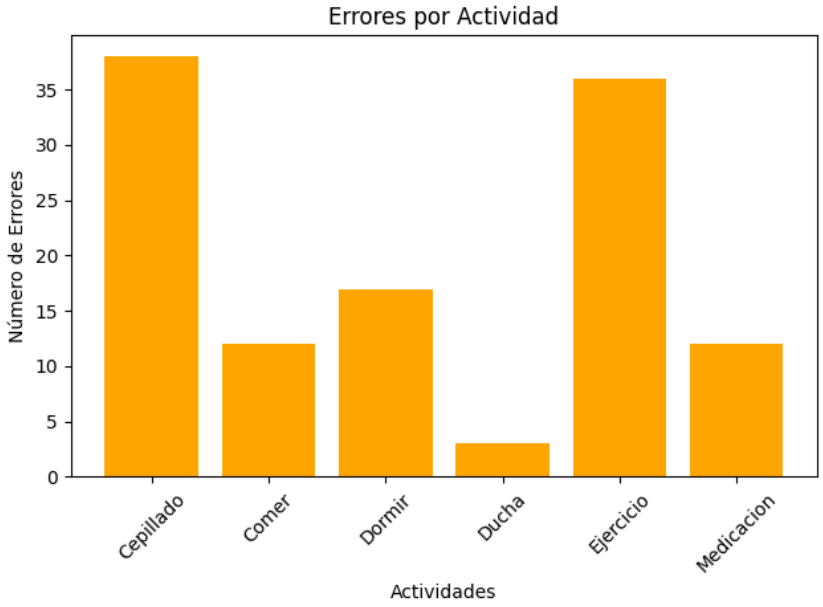

Figura 5.7: Errores por actividad obtenidos en el mejor resultado de la GRU.

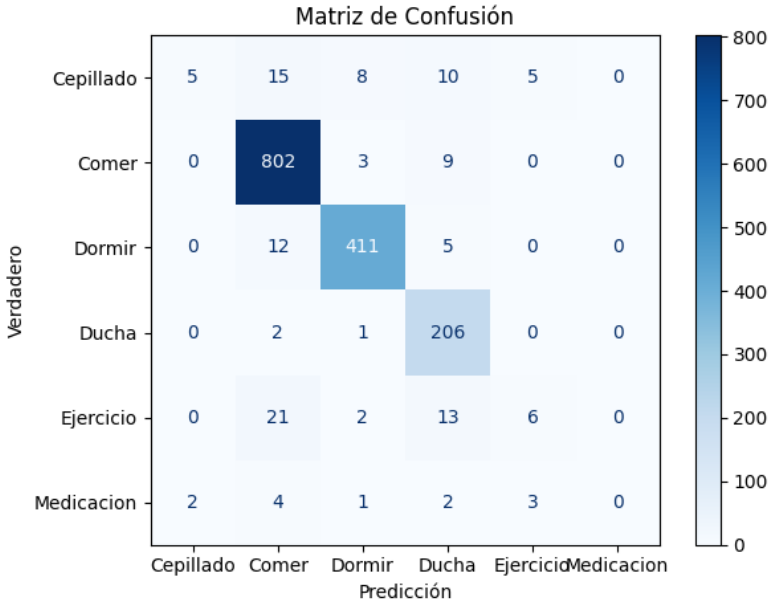

Figura 5.8: Matriz de confusión del mejor resultado de la GRU.

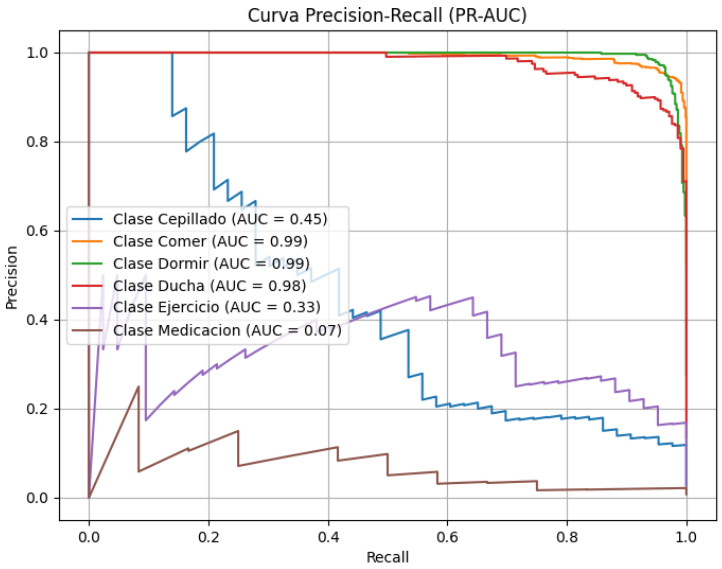

Figura 5.9: Curva *Precision-Recall* del mejor resultado de la GRU.

5.1.4. Bi-LSTM

En la tabla 5.5 se muestra el rendimiento de la red neuronal **LSTM bidireccional**. El mejor desempeño se consigue, de nuevo, con un tamaño de secuencia temporal de 250 muestras y con *patience* 25. Hay que destacar el mayor costo computacional de esta configuración, llegando a superar los 1000 segundos de ejecución.

Se puede observar también que disminuir o aumentar el tamaño de la secuencia temporal puede mejorar el tiempo de ejecución, pero a cambio de obtener peores resultados en cuánto a rendimiento.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.8961	0.8925	0.8961	0.8914	0.2878	243.58	64
15	**250**	0.9302	0.9166	0.9302	0.9204	0.2126	473.42	82
15	**350**	0.9312	0.9112	0.9312	0.9187	0.2278	593.13	76
20	**150**	0.9031	0.9032	0.9031	0.8978	0.2659	393.81	104
20	**250**	0.9302	0.9166	0.9302	0.9204	0.2126	500.52	87
20	**350**	0.9312	0.9112	0.9312	0.9187	0.2278	639.54	81
25	**150**	0.9095	0.9061	0.9095	0.9050	0.2657	509.25	131
25	**250**	0.9496	0.9432	0.9496	0.9449	0.1730	1076.83	184
25	**350**	0.9378	0.9224	0.9378	0.9292	0.1922	1039.53	129

Tabla. 5.5: Comparación de medidas de rendimiento del modelo Bi-LSTM con diferentes valores de `patience` y tamaño de la secuencia temporal.

Las siguientes figuras muestran los errores por actividad (figura 5.10), la matriz de confusión (figura 5.11) y la curva *Precision-recall* (figura 5.12) de la configuración con mejor rendimiento para el modelo LSTM bidireccional (*patience* = 25 y el tamaño de secuencia temporal = 250).

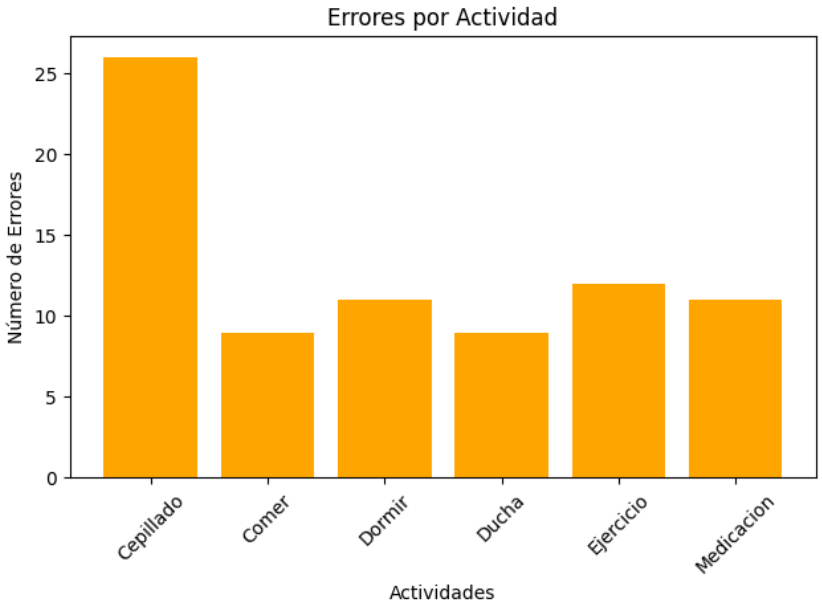

Figura 5.10: Errores por actividad obtenidos en el mejor resultado de la LSTM bidireccional.

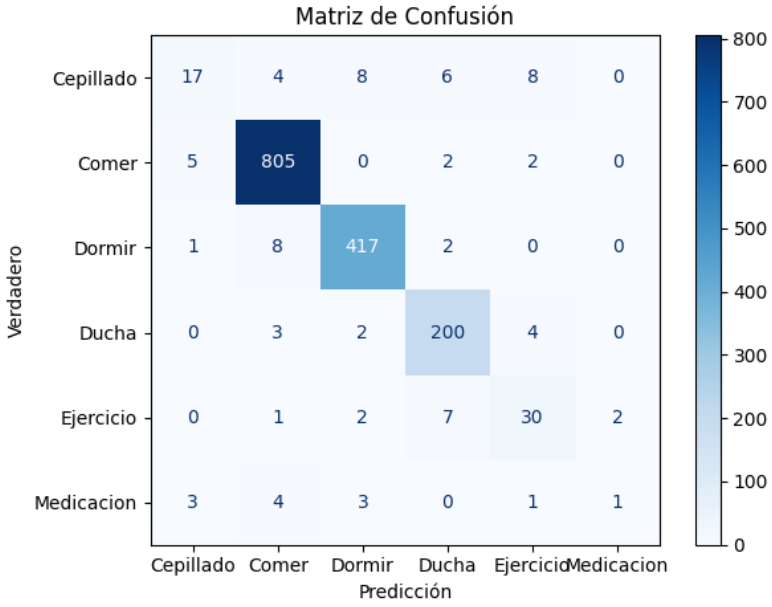

Figura 5.11: Matriz de confusión del mejor resultado de la LSTM bidireccional.

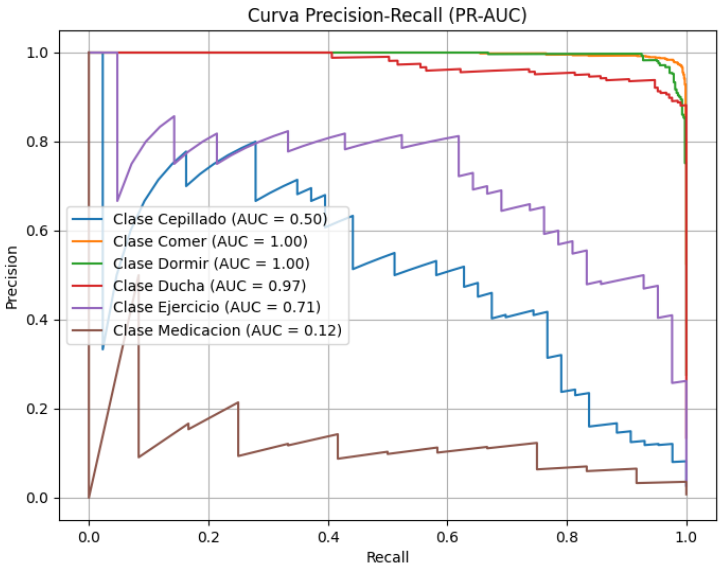

Figura 5.12: Curva *Precision-Recall* del mejor resultado de la LSTM bidireccional.

5.1.5. Bi-GRU

En la tabla 5.6 se puede ver como los mejores resultados se obtienen con la configuración de *patience* = 25 y un tamaño de secuencia de 250 muestras, llegando a un valor de *accuracy* de 0.9477. Al igual que los otros modelos, se observa que un valor intermedio del tamaño de secuencia es mejor que si se disminuyese o aumentase. Como diferencia a los demás modelos, se puede ver que un valor mayor de paciencia hace que mejoren los resultados, lo que indica que el modelo podría seguir mejorando con un número mayor de épocas.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.9012	0.8910	0.9012	0.8943	0.2708	322.90	83
15	**250**	0.9399	0.9399	0.9399	0.9357	0.1884	556.10	96
15	**350**	0.9312	0.9150	0.9312	0.9190	0.2285	614.46	77
20	**150**	0.9095	0.9081	0.9095	0.9069	0.2655	657.58	168
20	**250**	0.9399	0.9399	0.9399	0.9357	0.1884	597.69	101
20	**350**	0.9476	0.9348	0.9476	0.9408	0.2195	1386.17	175
25	**150**	0.9012	0.8910	0.9012	0.8943	0.2708	358.02	93
25	**250**	0.9477	0.9426	0.9477	0.9441	0.1631	925.87	160
25	**350**	0.9476	0.9348	0.9476	0.9408	0.2195	1416.26	180

Tabla. 5.6: Comparación de medidas de rendimiento del modelo Bi-GRU con diferentes valores de `patience` y tamaño de la secuencia temporal.

En las siguientes figuras se pueden observar los errores por actividad (figura 5.13),

la matriz de confusión (figura 5.14) y la curva PR-AUC (figura 5.15) del mejor resultado del modelo GRU bidireccional.

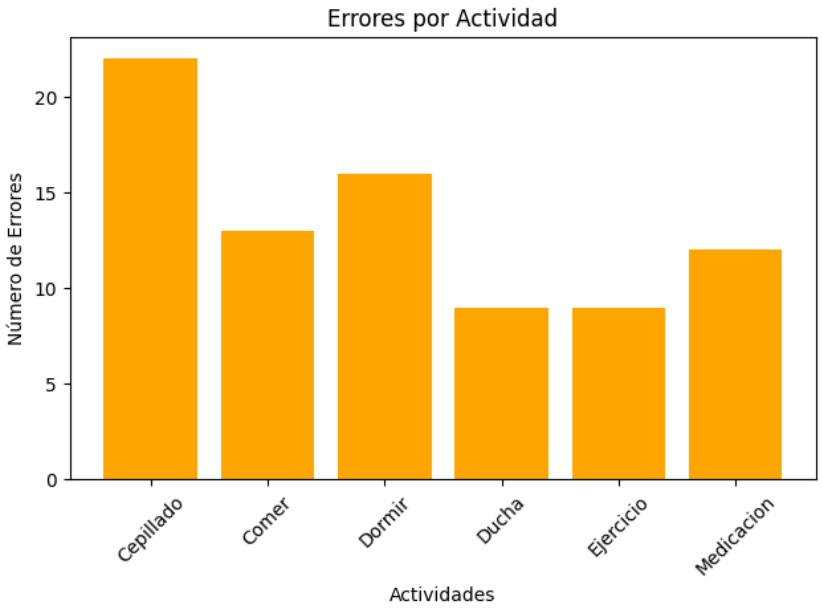

Figura 5.13: Errores por actividad obtenidos en el mejor resultado de la GRU bidireccional.

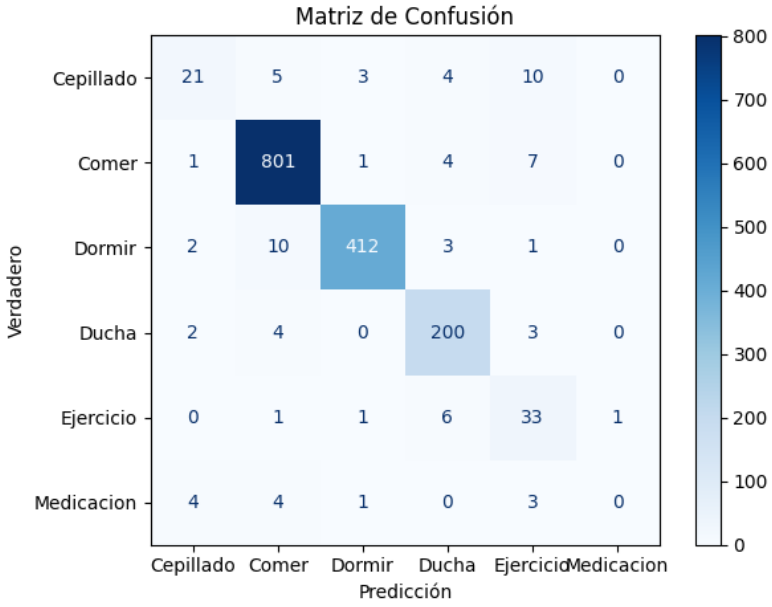

Figura 5.14: Matriz de confusión del mejor resultado de la GRU bidireccional.

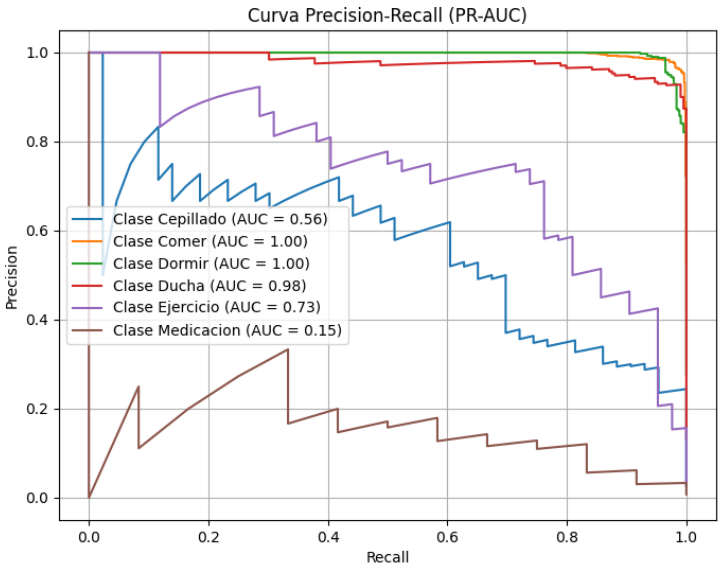

Figura 5.15: Curva *Precision-Recall* del mejor resultado de la GRU bidireccional.

5.1.6. CNN-BiLSTM

Los resultados de la tabla 5.5 muestran que el mejor resultado se obtiene con la configuración *patience* = 25 y tamaño de secuencia temporal = 250, logrando los mejores valores en todas las métricas, menos en la pérdida. De nuevo, se demuestra que un tamaño intermedio de secuencia temporal mejora los resultados.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.9069	0.9081	0.9069	0.9054	0.3394	90.32	71
15	**250**	0.9548	0.9507	0.9548	0.9519	0.1793	89.69	63
15	**350**	0.9470	0.9369	0.9470	0.9418	0.1865	76.00	43
20	**150**	0.9076	0.9056	0.9076	0.9041	0.3209	74.02	64
20	**250**	0.9522	0.9446	0.9522	0.9478	0.1479	84.69	59
20	**350**	0.9470	0.9369	0.9470	0.9418	0.1865	83.76	48
25	**150**	0.9057	0.9031	0.9057	0.9025	0.3320	94.01	83
25	**250**	0.9580	0.9543	0.9580	0.9549	0.1963	135.74	99
25	**350**	0.9470	0.9369	0.9470	0.9418	0.1865	91.52	53

Tabla. 5.7: Comparación de medidas de rendimiento del modelo CNN-BiLSTM con diferentes valores de `patience` y tamaño de la secuencia temporal.

En las siguientes figuras se pueden observar los errores por actividad (figura 5.16), la matriz de confusión (figura 5.17) y la curva *precision-recall* (figura 5.18) del mejor resultado del modelo híbrido que combina **CNN y LSTM bidireccional**.

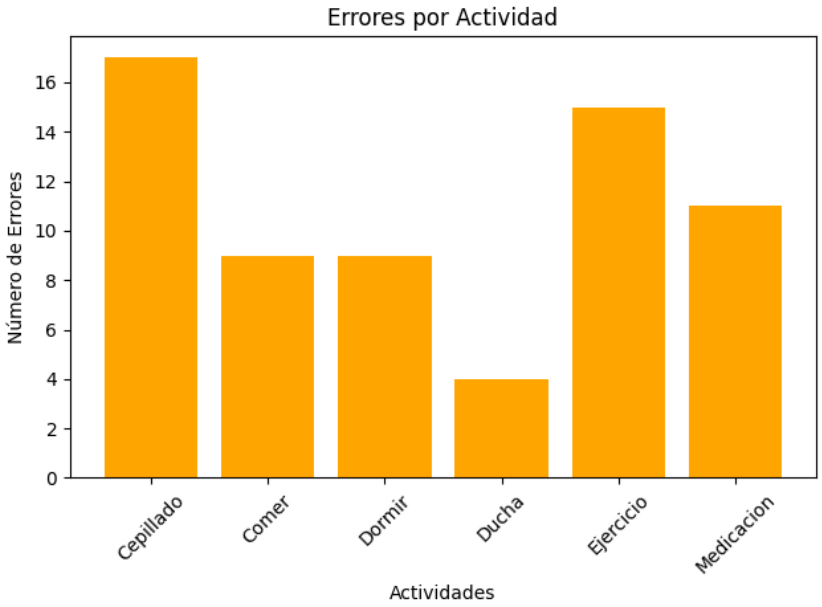

Figura 5.16: Errores por actividad obtenidos en el mejor resultado del modelo CNN-LSTM bidireccional.

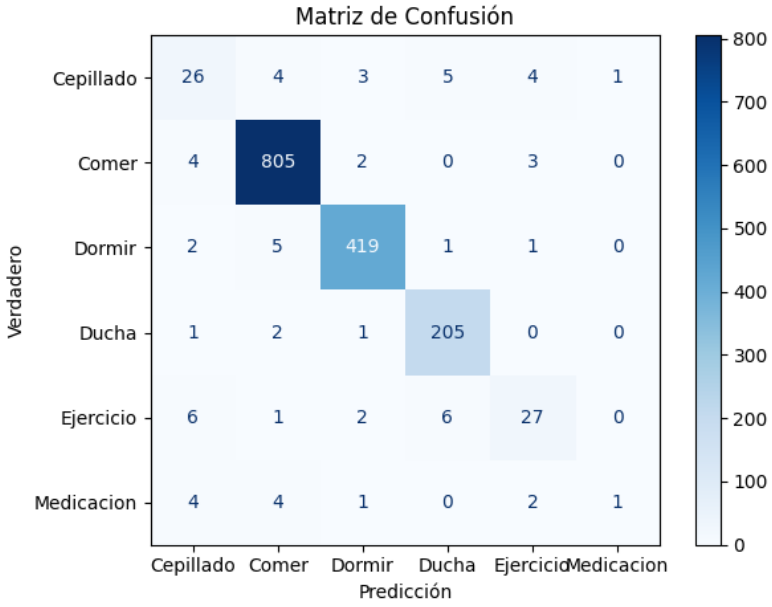

Figura 5.17: Matriz de confusión del mejor resultado del modelo CNN-LSTM bidireccional.

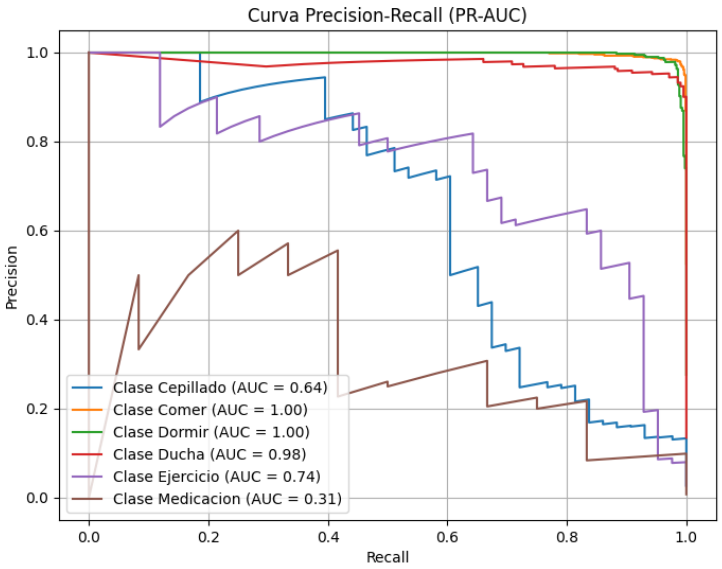

Figura 5.18: Curva *Precision-Recall* del mejor resultado del modelo CNN-LSTM bidireccional.

5.1.7. CNN-BiGRU

Como se puede observar en la tabla 5.8, el mejor rendimiento de este modelo se alcanza con el tamaño de secuencia temporal de 250 muestras, como en los demás modelos, y además, se observa que aumentar el hiperparámetro *patience* no mejora el rendimiento, indicando que el modelo converge de manera estable con un menor número de épocas.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**150**	0.9031	0.9038	0.9031	0.9011	0.3647	95.63	78
15	**250**	0.9561	0.9529	0.9561	0.9532	0.1928	119.29	88
15	**350**	0.9443	0.9370	0.9443	0.9405	0.2608	114.74	66
20	**150**	0.9031	0.9038	0.9031	0.9011	0.3647	92.34	83
20	**250**	0.9561	0.9529	0.9561	0.9532	0.1928	125.64	93
20	**350**	0.9443	0.9370	0.9443	0.9405	0.2608	122.56	71
25	**150**	0.9101	0.9108	0.9101	0.9081	0.3785	120.36	113
25	**250**	0.9561	0.9529	0.9561	0.9532	0.1928	134.02	98
25	**350**	0.9489	0.9450	0.9489	0.9444	0.2117	146.96	89

Tabla. 5.8: Comparación de medidas de rendimiento del modelo CNN-BiGRU con diferentes valores de `patience` y tamaño de la secuencia temporal.

Las siguientes figuras muestran los errores por actividad (figura 5.19), la matriz de confusión (figura 5.20) y la curva *Precision-recall* (figura 5.21) de la configuración con

mejor rendimiento para el modelo híbrido **CNN-GRU bidireccional** (*patience* = 15 y el tamaño de secuencia temporal = 250).

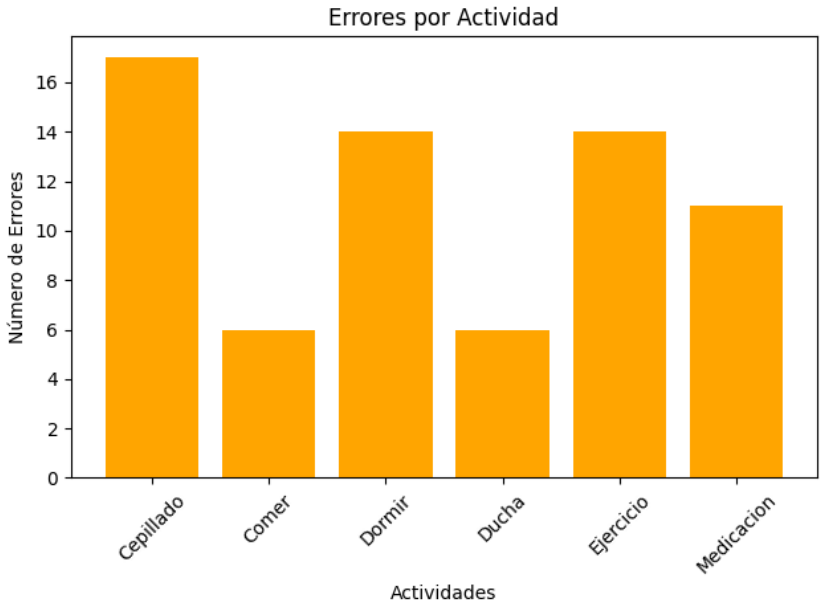

Figura 5.19: Errores por actividad obtenidos en el mejor resultado del modelo CNN-GRU bidireccional.

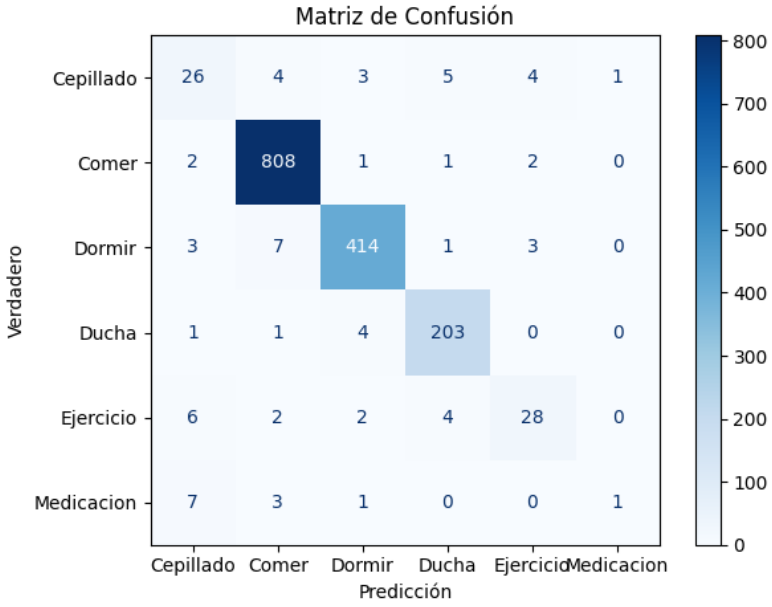

Figura 5.20: Matriz de confusión del mejor resultado del modelo CNN-GRU bidireccional.

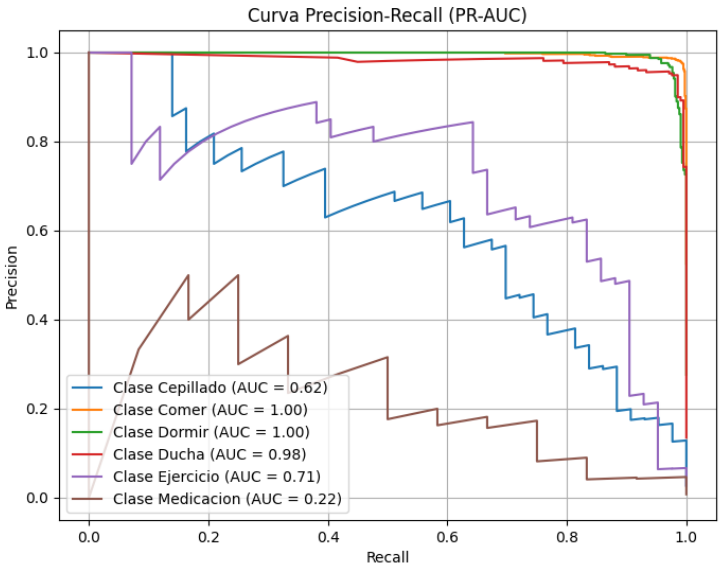

Figura 5.21: Curva *Precision-Recall* del mejor resultado del modelo CNN-GRU bidireccional.

5.1.8. RNN

Como se ve en la tabla 5.9, el coste computacional de este modelo es muy superior al de los demás modelos, superando la hora de ejecución. Por este motivo, y teniendo en cuenta que en las pruebas anteriores el tamaño de secuencia temporal óptimo ha sido de 250 muestras, las pruebas realizadas con este modelo se han limitado a este tamaño de secuencia, variando únicamente el valor de paciencia (*patience*).

Además, se observa que un valor de *patience* de 15 es suficiente para lograr la convergencia del modelo, ya que valores superiores solo aumentan el tiempo de entrenamiento sin aportar mejoras de rendimiento.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.9251	0.9103	0.9251	0.9134	0.2647	3420.01	117
20	**250**	0.9251	0.9103	0.9251	0.9134	0.2647	3635.41	122
25	**250**	0.9251	0.9103	0.9251	0.9134	0.2647	3805.19	127

Tabla. 5.9: Comparación de medidas de rendimiento del modelo RNN con diferentes valores de `patience` y tamaño de la secuencia temporal.

Las siguientes figuras muestran los errores por actividad (figura 5.22), la matriz de confusión (figura 5.23) y la curva *Precision-recall* (figura 5.24) de la configuración con mejor rendimiento para el modelo híbrido **RNN** (*patience* = 15 y el tamaño de

secuencia temporal = 250).

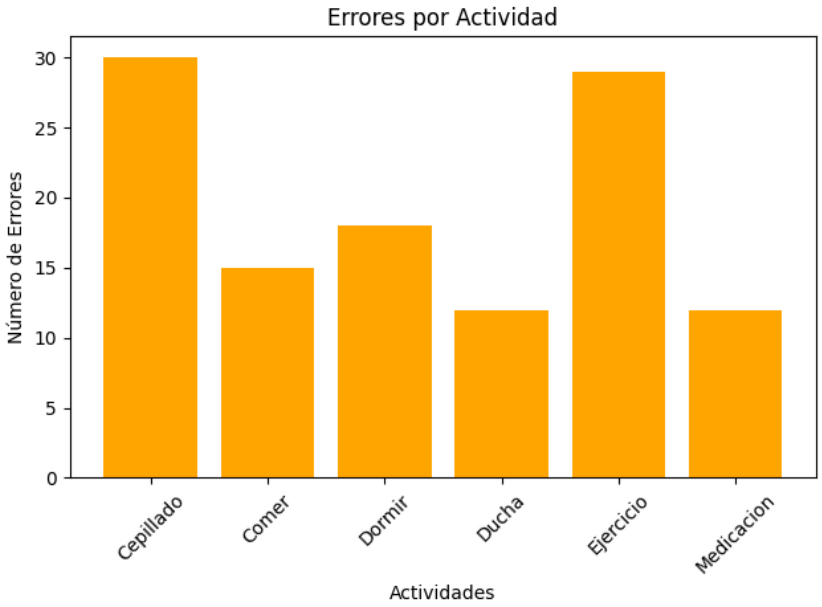

Figura 5.22: Errores por actividad obtenidos en el mejor resultado de la RNN.

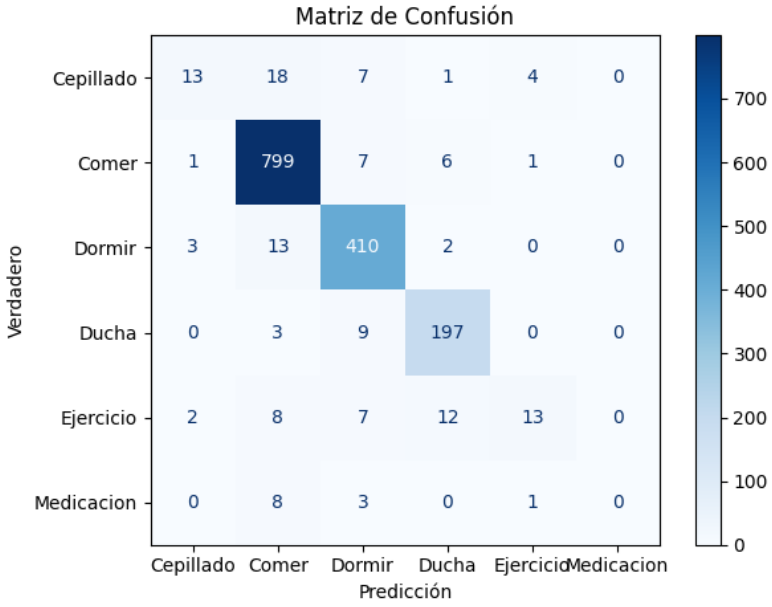

Figura 5.23: Matriz de confusión del mejor resultado de la RNN.

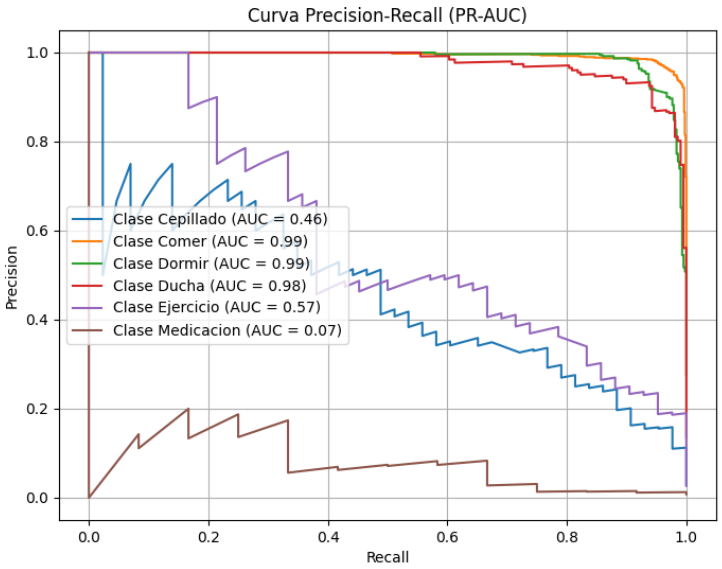

Figura 5.24: Curva *Precision-Recall* del mejor resultado de la RNN.

5.1.9. Comparación de modelos

En la tabla 5.10, se pueden ver los **mejores resultados obtenidos para cada modelo** con el dataset sin información de localización.

Se observa que las Redes Neuronales Convolucionales destacan con la mayor precisión y el menor tiempo de entrenamiento. Los modelos recurrentes como LSTM y BiLSTM presentan un buen rendimiento, pero con un mayor coste computacional, especialmente en el segundo caso, donde se superan los 1000 segundos de ejecución.

Los modelos híbridos ofrecen un buen equilibrio entre rendimiento y coste computacional, logrando grandes resultados de precisión en la detección de actividades y con tiempos de entrenamiento reducidos. Por otro lado, el modelo RNN muestra el peor rendimiento y el mayor tiempo de entrenamiento (3420 segundos).

En general, las arquitecturas CNN y sus combinaciones con redes recurrentes resultan las más efectivas, ofreciendo un gran rendimiento con un bajo coste computacional.

En las figuras 5.25 y 5.26 se pueden ver dos comparativas de los resultados de los distintos modelos. La primera gráfica muestra una comparación de la precisión de los modelos en la validación, mientras que la segunda gráfica muestra una comparación de la pérdida de los modelos.

Modelo	Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
CNN	**20**	**250**	0.9677	0.9652	0.9677	0.9651	0.1987	64.51	85
LSTM	**25**	**250**	0.9509	0.9403	0.9509	0.9419	0.1613	395.40	235
GRU	**15**	**250**	0.9238	0.9044	0.9238	0.9051	0.2432	282.30	174
BiLSTM	**25**	**250**	0.9496	0.9432	0.9496	0.9449	0.1730	1076.83	184
BiGRU	**25**	**250**	0.9477	0.9426	0.9477	0.9441	0.1631	925.87	160
CNN-BiLSTM	**25**	**250**	0.9580	0.9543	0.9580	0.9549	0.1963	135.74	99
CNN-BiGRU	**15**	**250**	0.9561	0.9529	0.9561	0.9532	0.1928	112.29	88
RNN	**15**	**250**	0.9251	0.9103	0.9251	0.9134	0.2647	3420.01	117

Tabla. 5.10: Comparación de los mejores resultados de los diferentes modelos con el conjunto de datos sin localización.

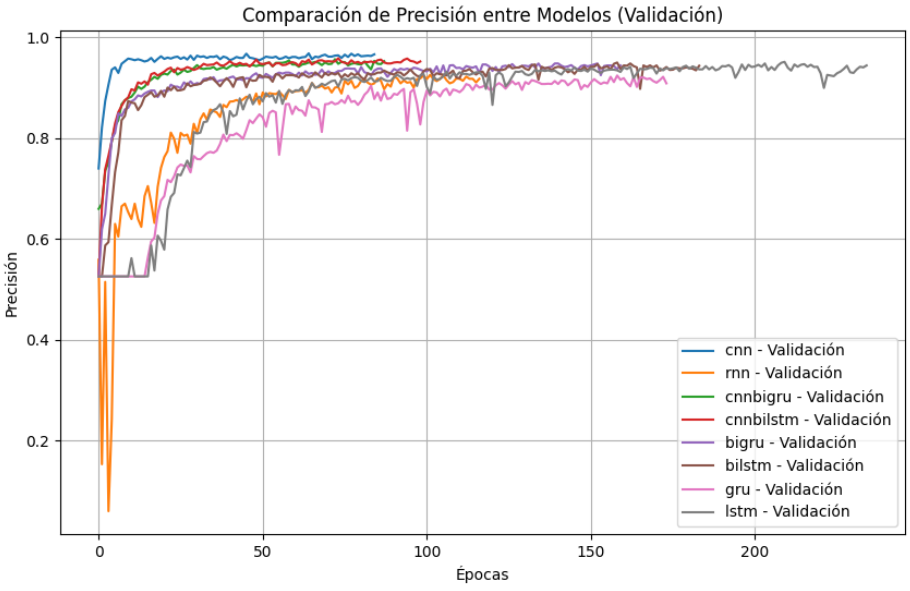

Figura 5.25: Comparación de la precisión en la validación de los diferentes modelos.

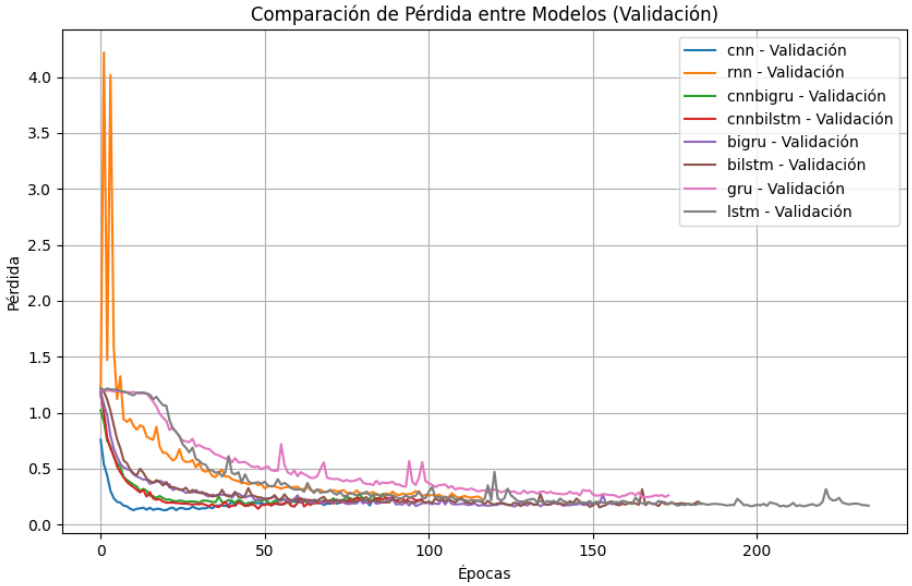

Figura 5.26: Comparación de la pérdida en la validación de los diferentes modelos.

5.2. Conjunto de datos con localización

En esta sección se muestran los resultados obtenidos de la ejecución de los distintos modelos con el conjunto de datos que incluye la localización de los residentes, tratando de reconocer qué actividad es la que se realiza y quién la realizó.

Los hiperparámetros siguen siendo los mismos que en las anteriores pruebas (tabla 5.1), pero en este caso, únicamente variará el valor del parámetro *patience*, ya que, como se ha visto en las pruebas anteriores, el valor óptimo del tamaño de la secuencia temporal ha sido 250, logrando con este valor los mejores resultados de rendimiento.

De la misma manera que en la sección anterior, se resalta en color rojo y se incluyen gráficas del mejor resultado de cada modelo.

5.2.1. CNN

Como se observa en la tabla 5.11, el modelo **CNN** alcanza una precisión del 89.15 % en todas las configuraciones, lo que demuestra que el aumento del parámetro *patience* no impacta en la mejora del rendimiento y que se encuentra rápidamente la convergencia del modelo, haciendo que el *early stopping* cumpla su función.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8915	0.8890	0.8915	0.8898	0.4619	35.68	40
20	**250**	0.8915	0.8890	0.8915	0.8898	0.4619	38.95	45
25	**250**	0.8915	0.8890	0.8915	0.8898	0.4619	41.98	50

Tabla. 5.11: Comparación de medidas de rendimiento del modelo CNN con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.27) y la matriz de confusión (5.28) del mejor resultado del modelo en el conjunto de datos con localización.

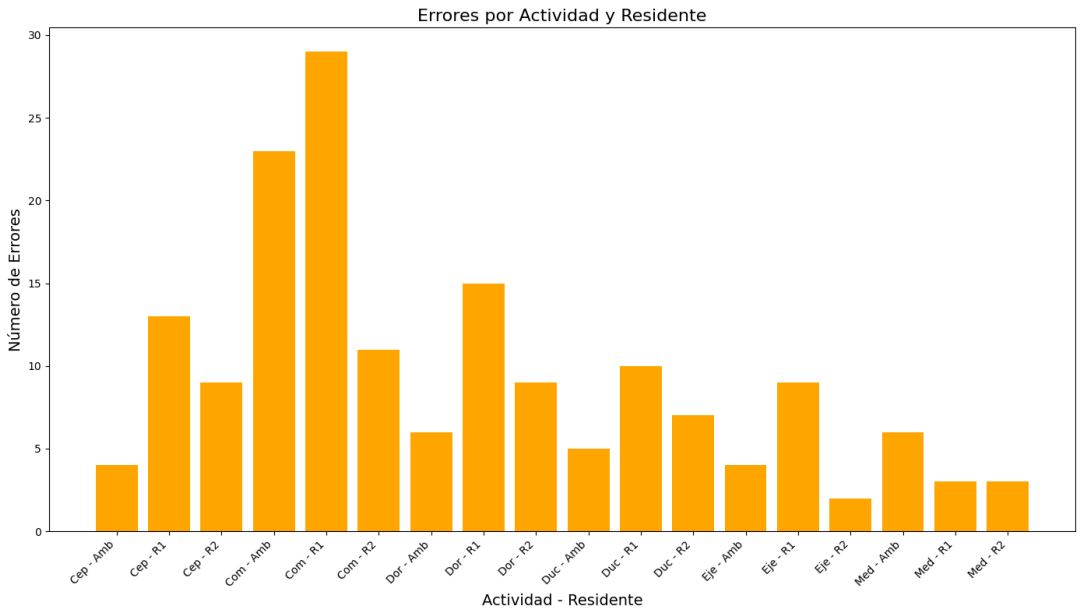

Figura 5.27: Errores por actividad obtenidos en el mejor resultado de la CNN.

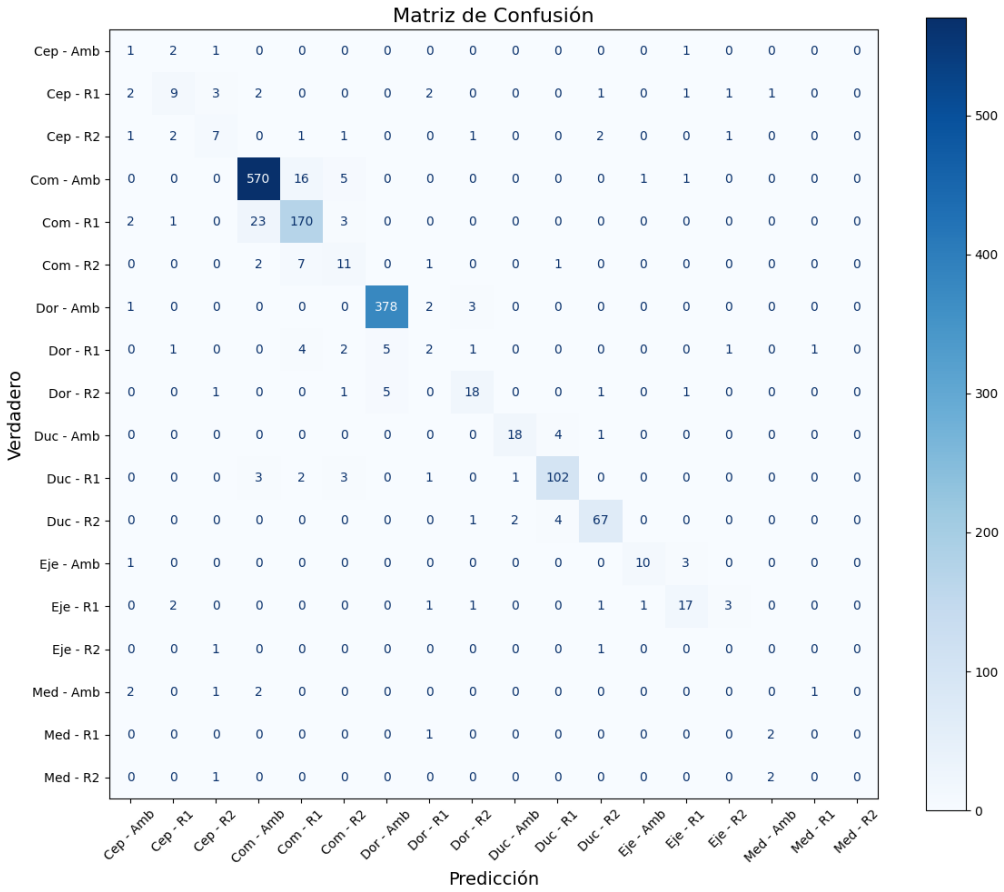

Figura 5.28: Matriz de confusión del mejor resultado de la CNN.

5.2.2. LSTM

En la tabla 5.12, se muestran los resultados del modelo **LSTM** utilizando el conjunto de datos con información de localización.

Se puede observar cómo el mejor rendimiento se alcanza con *patience* 20, obteniendo una precisión del 83.33 %. Sin embargo, al aumentar la paciencia a 25, los resultados permanecen idénticos, lo que indica que el modelo deja de mejorar a partir de ese punto.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8249	0.7769	0.8249	0.7922	0.5885	268	154
20	**250**	0.8333	0.8017	0.8333	0.8089	0.5532	347.24	200
25	**250**	0.8333	0.8017	0.8333	0.8089	0.5532	358.54	205

Tabla. 5.12: Comparación de medidas de rendimiento del modelo LSTM con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.29) y la matriz de confusión (5.30) del mejor resultado del modelo en el conjunto de datos con localización.

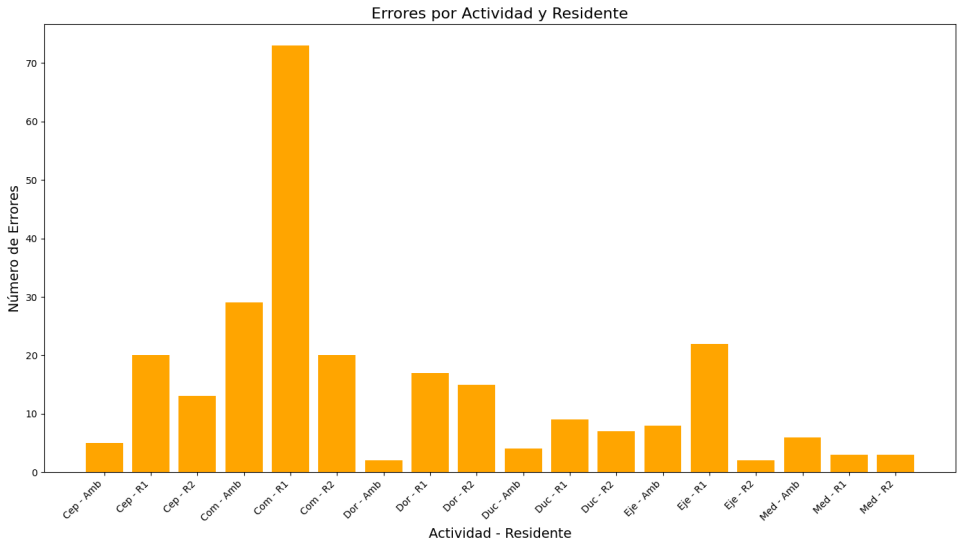

Figura 5.29: Errores por actividad obtenidos en el mejor resultado de la LSTM.

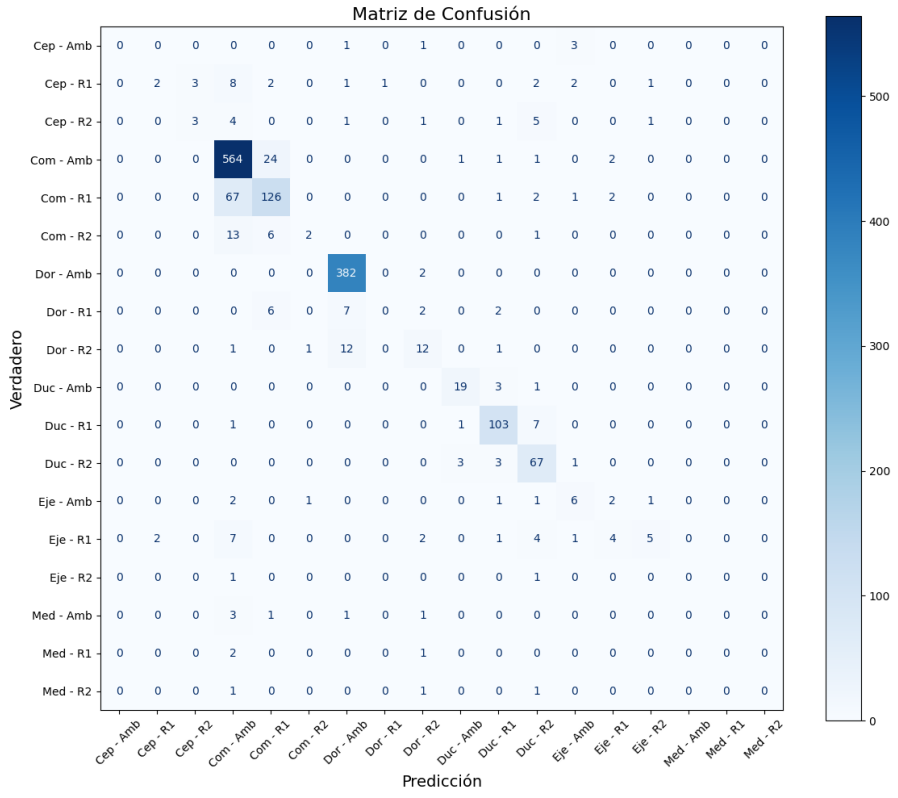

Figura 5.30: Matriz de confusión del mejor resultado de la LSTM.

5.2.3. GRU

Como se puede ver en la tabla 5.13, se presentan los resultados de rendimiento del modelo **GRU** utilizando el conjunto de datos. De igual manera que el anterior modelo, el mejor rendimiento se consigue con *pacience* 20, obteniendo un 82.43 % de precisión y aumentando este parámetro, el rendimiento no mejora y aumenta el tiempo de ejecución.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8236	0.7833	0.8236	0.7929	0.6338	262.65	160
20	**250**	0.8243	0.7701	0.8243	0.7904	0.6177	308.24	187
25	**250**	0.8243	0.7701	0.8243	0.7904	0.6177	328.22	192

Tabla. 5.13: Comparación de medidas de rendimiento del modelo GRU con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.31) y la matriz de confusión (5.32) del mejor resultado del modelo en el conjunto de datos con localización.

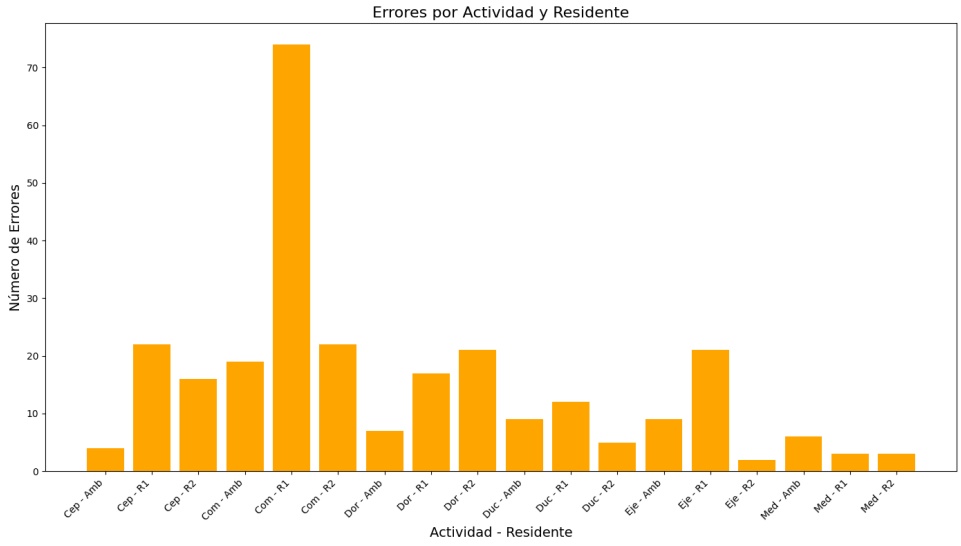

Figura 5.31: Errores por actividad obtenidos en el mejor resultado de la GRU.

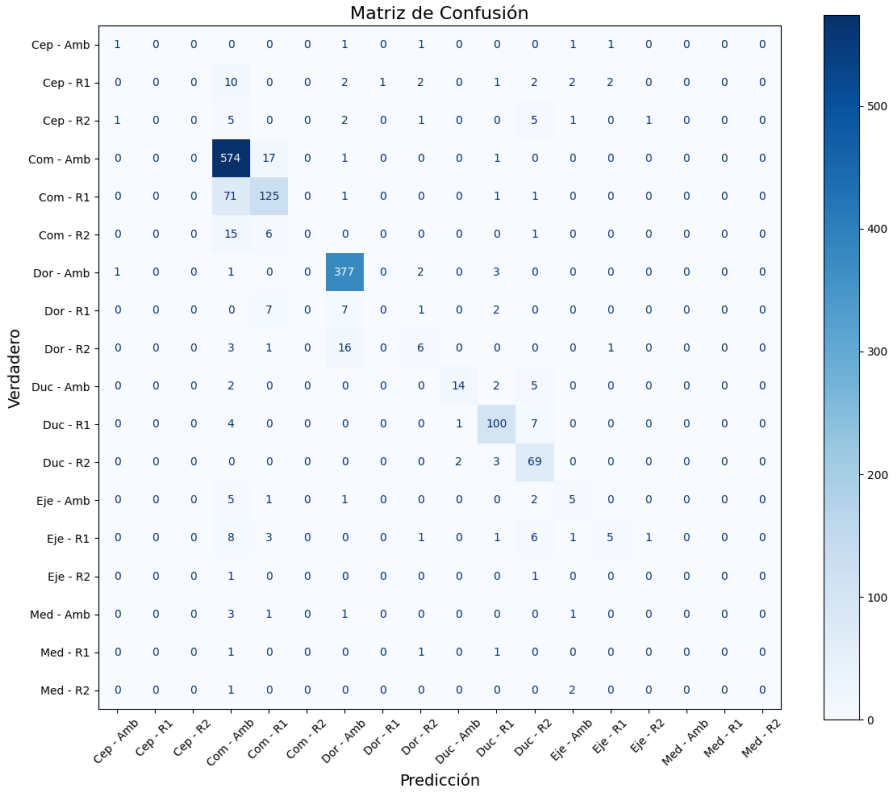

Figura 5.32: Matriz de confusión del mejor resultado de la GRU.

5.2.4. Bi-LSTM

La tabla 5.14 muestra el rendimiento del modelo **BiLSTM**, este obtiene su mejor resultado con *patience* 15, llegando a tener un 85.66 % de precisión, siendo muy poca la diferencia con las otras configuraciones.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8566	0.8261	0.8566	0.8375	0.4891	644.90	108
20	**250**	0.8566	0.8261	0.8566	0.8375	0.4891	676.33	113
25	**250**	0.8566	0.8261	0.8566	0.8375	0.4891	702.34	118

Tabla. 5.14: Comparación de medidas de rendimiento del modelo Bi-LSTM con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.33) y la matriz de confusión (5.34) del mejor resultado del modelo en el conjunto de datos con localización.

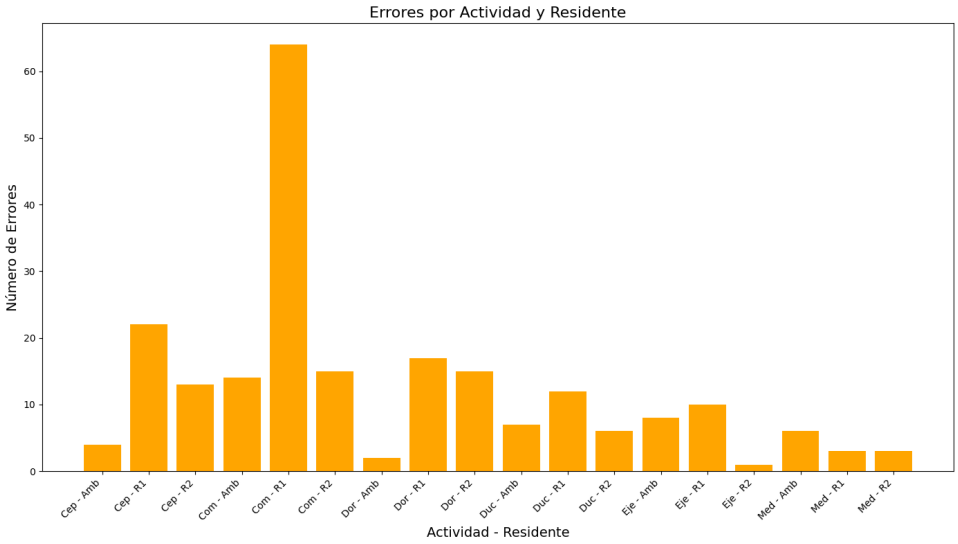

Figura 5.33: Errores por actividad obtenidos en el mejor resultado de la LSTM bidireccional.

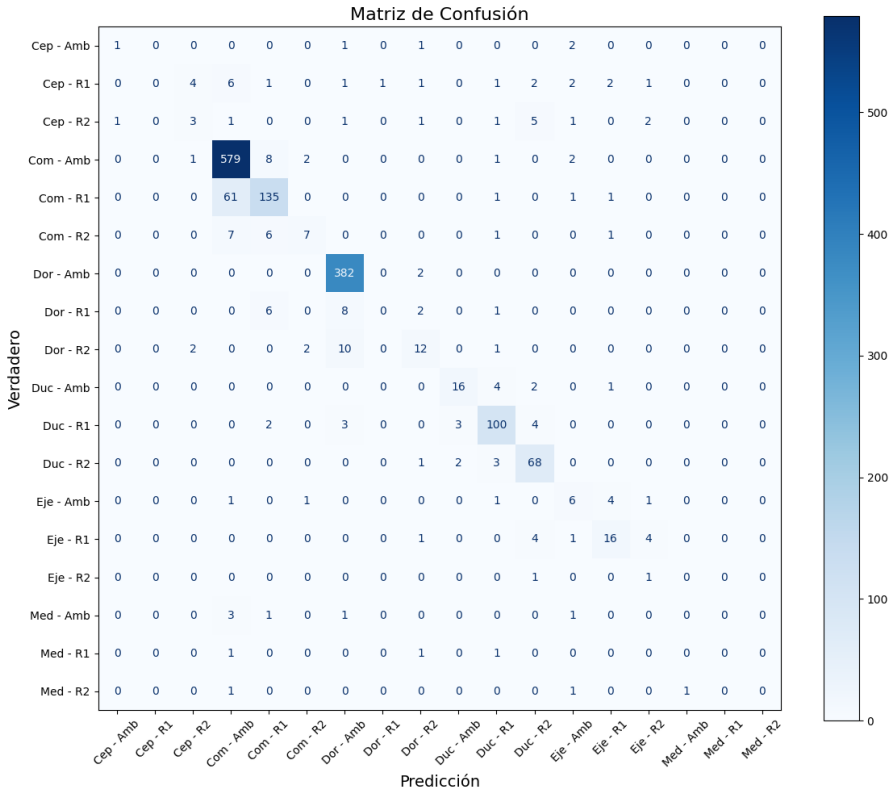

Figura 5.34: Matriz de confusión del mejor resultado de la LSTM bidireccional.

5.2.5. Bi-GRU

La tabla 5.15 muestra el rendimiento del modelo **BiGRU**, este obtiene su mejor resultado con *patience* 25, llegando a tener un 87.66 % de precisión, siendo muy poca la diferencia con las otras configuraciones. En cuanto al tiempo de ejecución, se observa cómo se incrementa bastante respecto a otros modelos, llegando a superar los 1000 segundos.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8760	0.8660	0.8760	0.8683	0.4817	914.10	155
20	**250**	0.8760	0.8660	0.8760	0.8683	0.4817	939.64	160
25	**250**	0.8766	0.8594	0.8766	0.8652	0.5213	1105.21	188

Tabla. 5.15: Comparación de medidas de rendimiento del modelo Bi-GRU con diferentes valores de `patience` y tamaño de la secuencia temporal.

En las siguientes figuras se pueden ver los errores por actividad (5.35) y la matriz de confusión (5.36) del mejor resultado del modelo en el conjunto de datos con localización.

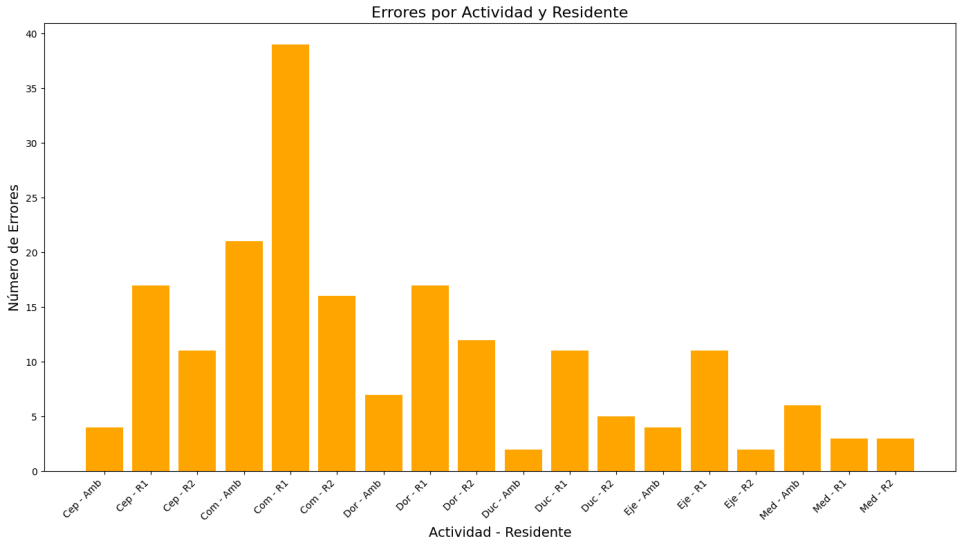

Figura 5.35: Errores por actividad obtenidos en el mejor resultado de la GRU bidireccional.

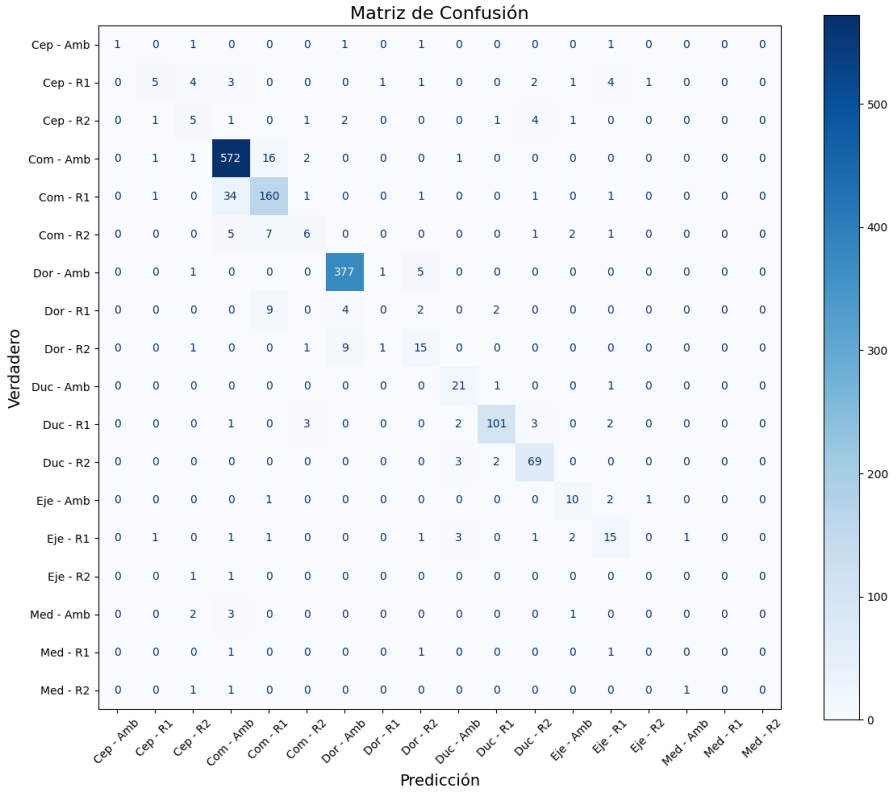

Figura 5.36: Matriz de confusión del mejor resultado de la GRU bidireccional.

5.2.6. CNN-BiLSTM

En la tabla 5.16 se observan los resultados obtenidos por el modelo **CNN-BiLSTM** utilizando el conjunto de datos con localización.

El mejor rendimiento se consigue con *patience* 20, logrando una precisión del 88.37 %. Como en los modelos LSTM y GRU, al aumentar el parámetro a 25 no hay variación en los resultados, demostrando que el modelo alcanza su límite de rendimiento.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8818	0.8661	0.8818	0.8718	0.4940	116.55	83
20	**250**	0.8837	0.8680	0.8837	0.8747	0.4348	118.25	84
25	**250**	0.8837	0.8680	0.8837	0.8747	0.4348	127.82	89

Tabla. 5.16: Comparación de medidas de rendimiento del modelo CNN-BiLSTM con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.37) y la matriz de confusión (5.38) del mejor resultado del modelo en el conjunto de datos con

localización.

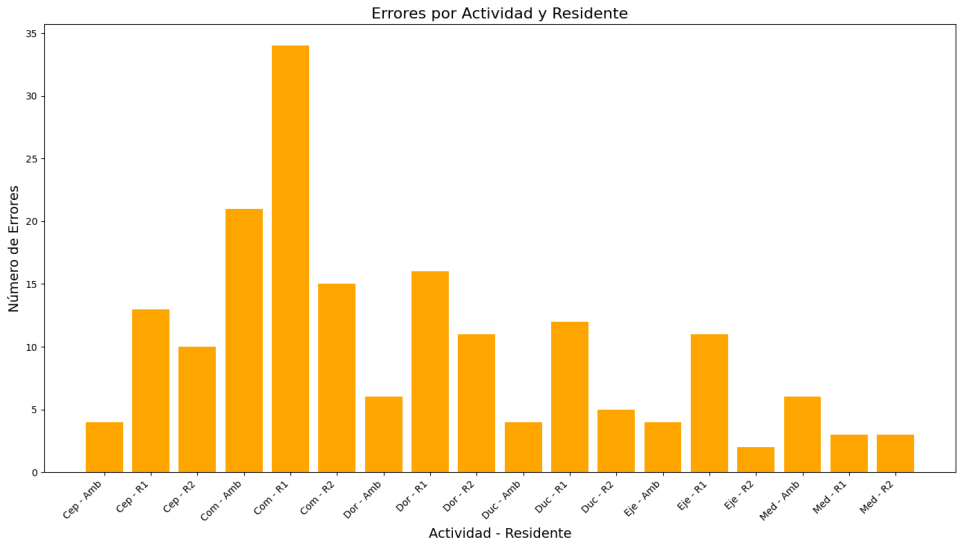

Figura 5.37: Errores por actividad obtenidos en el mejor resultado del modelo CNN-LSTM bidireccional.

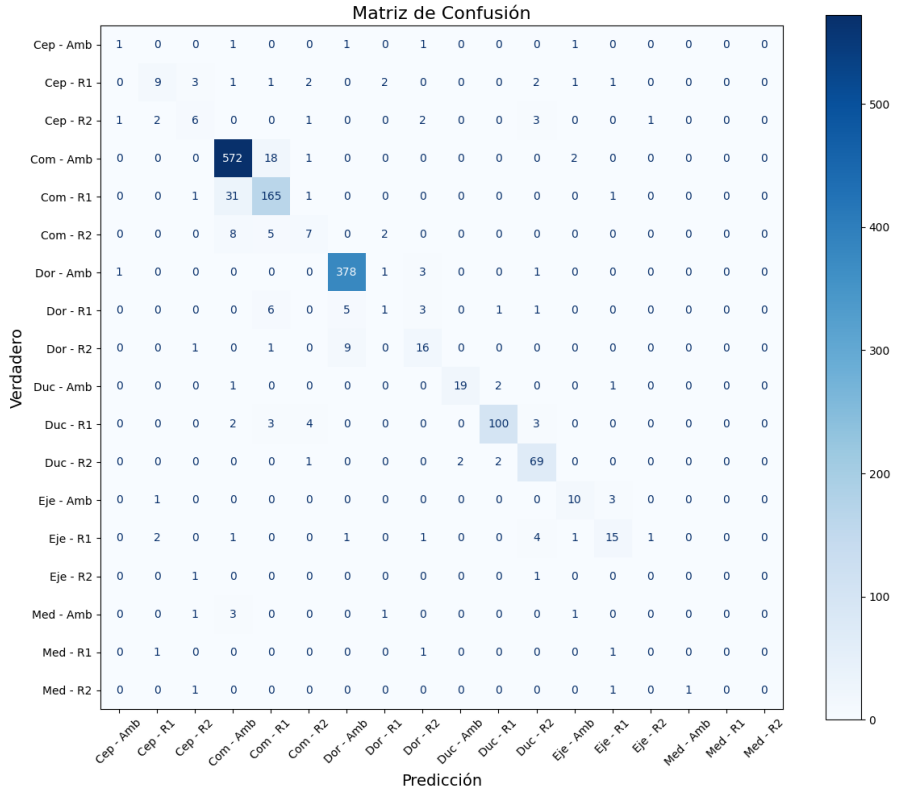

Figura 5.38: Matriz de confusión del mejor resultado del modelo CNN-LSTM bidireccional.

5.2.7. CNN-BiGRU

Como se observa en la tabla 5.17, el modelo **CNN-BiGRU** alcanza su mejor rendimiento con una *patience* de 15, llegando a obtener un 87.98 % de precisión. Además, se demuestra que aumentar el valor de *patience* no aporta ninguna mejora en las métricas de rendimiento y únicamente empeora el tiempo de ejecución.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.8798	0.8625	0.8798	0.8703	0.5447	136.85	98
20	**250**	0.8798	0.8625	0.8798	0.8703	0.5447	141.41	103
25	**250**	0.8798	0.8625	0.8798	0.8703	0.5447	148.03	108

Tabla. 5.17: Comparación de medidas de rendimiento del modelo CNN-BiGRU con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.39) y la matriz de confusión (5.40) del mejor resultado del modelo en el conjunto de datos con localización.

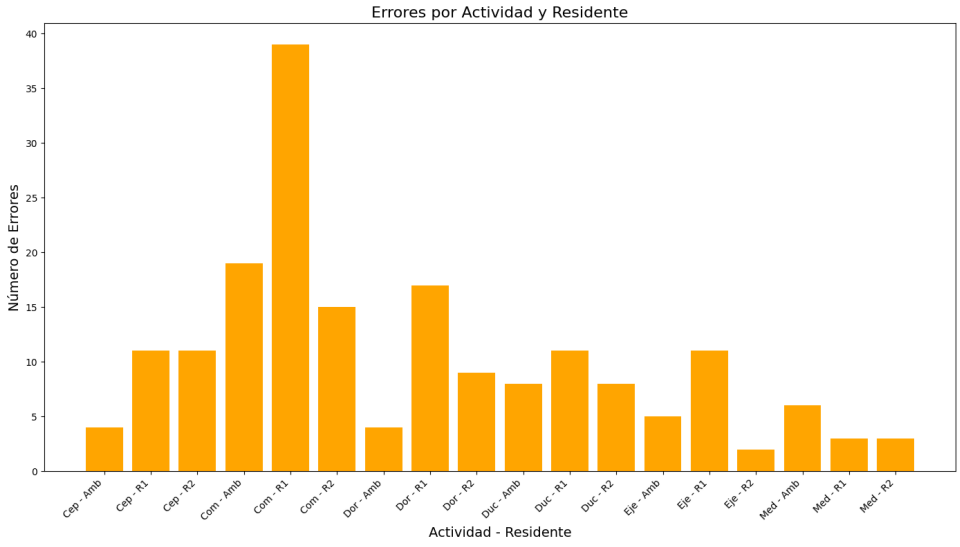

Figura 5.39: Errores por actividad obtenidos en el mejor resultado del modelo CNN-GRU bidireccional.

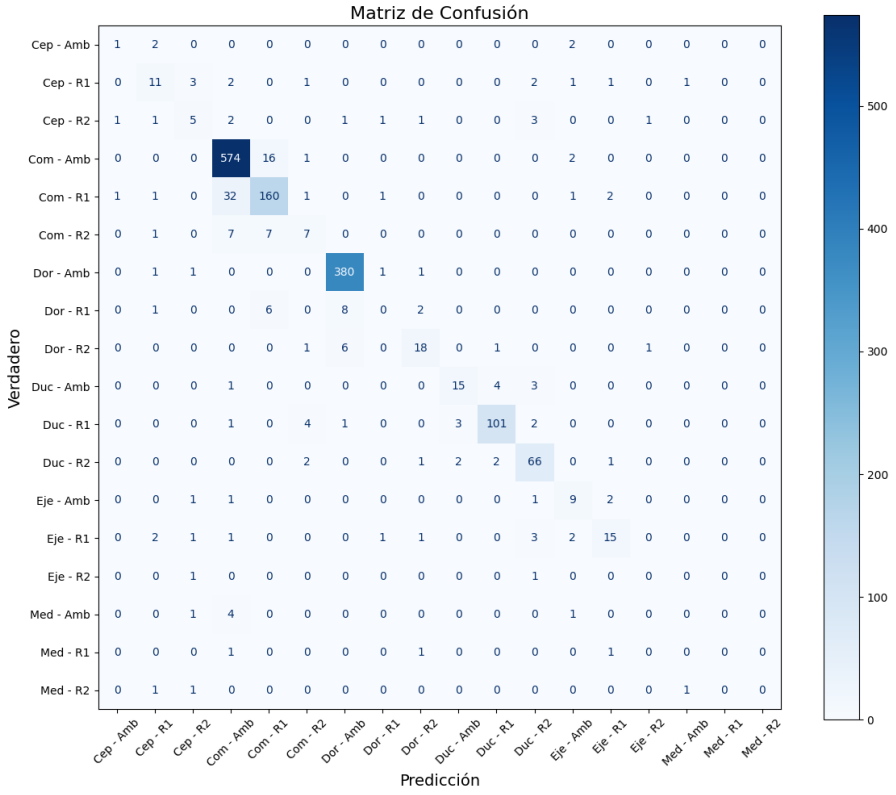

Figura 5.40: Matriz de confusión del mejor resultado del modelo CNN-GRU bidireccional.

5.2.8. RNN

En la tabla 5.18, se pueden ver los resultados de rendimiento de la **RNN**. Se observa que el mejor resultado se obtiene con *patience* 25, alcanzando un 83.14 % de precisión. Además, se observa que aumenta de manera considerable el coste computacional, ya que se superan los 5000 segundos de ejecución.

Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
15	**250**	0.7745	0.7234	0.7745	0.7315	0.7271	2622.79	88
20	**250**	0.7745	0.7234	0.7745	0.7315	0.7271	2773.54	93
25	**250**	0.8314	0.8126	0.8314	0.7921	0.5918	5212.20	180

Tabla. 5.18: Comparación de medidas de rendimiento del modelo RNN con diferentes valores de `patience` y tamaño de la secuencia temporal en el dataset con localización.

En las siguientes figuras se pueden ver los errores por actividad (5.41) y la matriz de confusión (5.42) del mejor resultado del modelo en el conjunto de datos con localización.

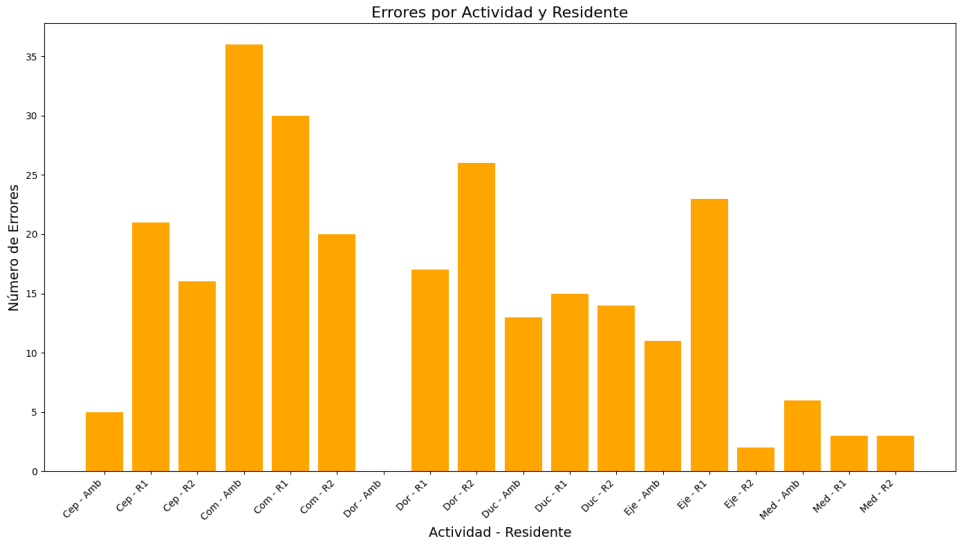

Figura 5.41: Errores por actividad obtenidos en el mejor resultado de la RNN.

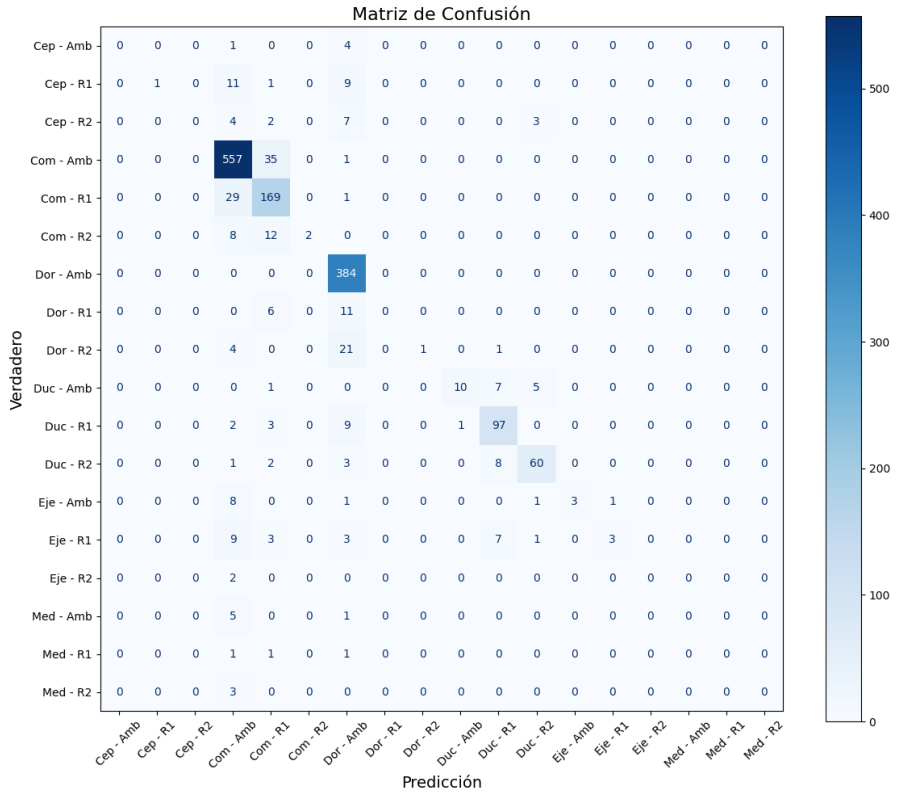

Figura 5.42: Matriz de confusión del mejor resultado de la RNN.

5.2.9. Comparación de modelos

En la tabla 5.19, se pueden ver los mejores resultados obtenidos para cada modelo con el dataset con información de localización.

Se observa que, nuevamente, el modelo **CNN** logra los mejores resultados, tanto en medidas de rendimiento como en coste computacional, ya que demuestra ser el más eficiente con tan solo 36 segundos de ejecución y 40 épocas.

Los modelos híbridos **CNN-BiLSTM** y **CNN-BiGRU** también muestran buen rendimiento, llegando a unos valores de precisión superiores al 87 % y tiempos de entrenamiento bajos en comparación con las arquitecturas basadas únicamente en redes recurrentes.

Los modelos restantes (**RNN**, **LSTM**, **GRU**, **BiLSTM** y **BiGRU**) presentan valores más bajos en cuanto a precisión, *F1-Score* y *Recall*, además de elevados tiempos de ejecución, llegando a los 5200 segundos de ejecución en el modelo RNN.

Modelo	Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
CNN	**15**	**250**	0.8915	0.8890	0.8915	0.8898	0.4619	35.68	40
LSTM	**20**	**250**	0.8333	0.8017	0.8333	0.8089	0.5532	347.24	200
GRU	**20**	**250**	0.8243	0.7701	0.8243	0.7904	0.6177	308.24	187
BiLSTM	**15**	**250**	0.8566	0.8261	0.8566	0.8375	0.4891	644.90	108
BiGRU	**25**	**250**	0.8766	0.8594	0.8766	0.8652	0.5213	1105.21	188
CNN-BiLSTM	**20**	**250**	0.8837	0.8680	0.8837	0.8747	0.4348	118.25	84
CNN-BiGRU	**15**	**250**	0.8798	0.8625	0.8798	0.8703	0.5447	136.85	98
RNN	**15**	**250**	0.8314	0.8126	0.8314	0.7921	0.5918	5212.20	180

Tabla. 5.19: Comparación de los mejores resultados de los diferentes modelos con el conjunto de datos con localización.

En las figuras 5.43 y 5.44 se pueden ver dos comparativas de los resultados de los distintos modelos. La primera gráfica muestra una comparación de la precisión de los modelos en la validación, mientras que la segunda gráfica muestra una comparación de la pérdida de los modelos.

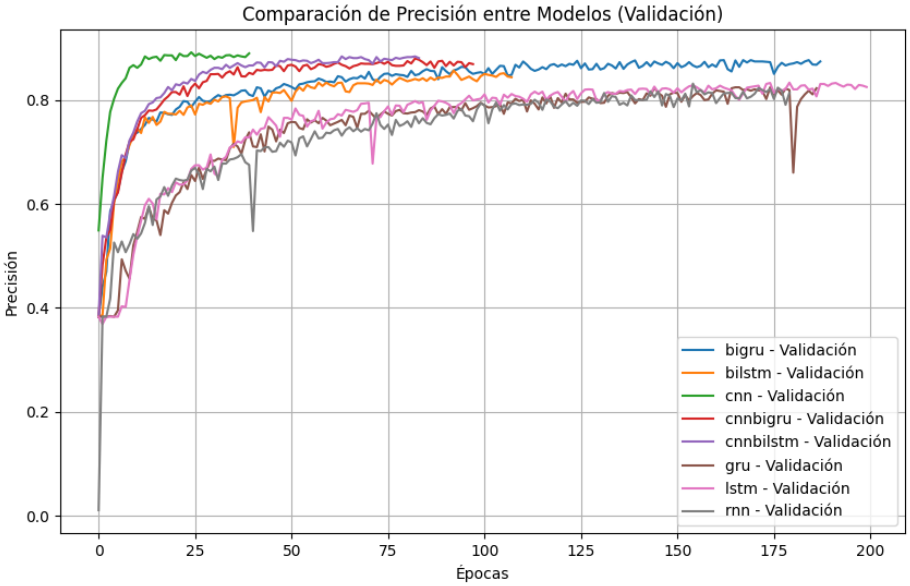

Figura 5.43: Comparación de la precisión en la validación de los diferentes modelos.

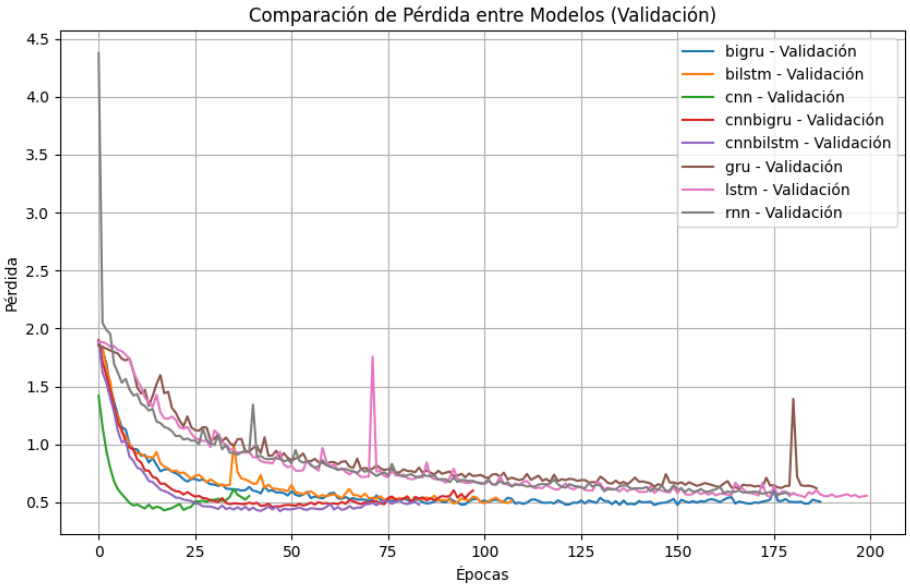

Figura 5.44: Comparación de la pérdida en la validación de los diferentes modelos.

5.3. Comparación de los modelos con los diferentes dataset

En la 5.20 se presentan los tres mejores resultados obtenidos por los modelos en función de si se ha utilizado o no la información de la localización.

Se puede ver que los modelos entrenados sin información de la localización alcanzan un mejor rendimiento en comparación con aquellos que sí tienen esta información. En particular, el modelo **CNN** sin localización logra los mejores resultados y con un tiempo de ejecución bajo.

Por otro lado, al incorporar la localización, se aprecia una disminución de un 7 % en cuanto a la precisión de los modelos. Lo que indica que la inclusión de los datos de localización no ha aportado mejoras significativas en la clasificación de actividades y ha llegado a generar un efecto negativo.

Modelo	Localización	Patience	Tamaño Secuencia	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
CNN	**No**	**20**	**250**	0.9677	0.9652	0.9677	0.9651	0.1987	64.51	85
CNN-BiLSTM	**No**	**25**	**250**	0.9580	0.9543	0.9580	0.9549	0.1963	135.71	99
CNN-BiGRU	**No**	**15**	**250**	0.9561	0.9529	0.9561	0.9532	0.1928	112.29	88
CNN	**Si**	**15**	**250**	0.8915	0.8890	0.8915	0.8898	0.4619	35.68	40
CNN-BiLSTM	**Si**	**20**	**250**	0.8837	0.8680	0.8837	0.8747	0.4348	118.25	84
CNN-BiGRU	**Si**	**15**	**250**	0.8798	0.8625	0.8798	0.8703	0.5447	136.85	98

Tabla. 5.20: Comparación de los mejores resultados de los modelos con los diferentes conjuntos de datos.

En las siguientes figuras se puede observar diferentes comparaciones de los mejores resultados de los modelos con el conjunto de datos con localización y sin localización. Comparación de precisión en el entrenamiento (5.45), comparación de precisión en el proceso de validación (5.46), comparación de la pérdida de los modelos en el entrenamiento (5.47) y comparación de la pérdida de los modelos en el conjunto de validación (5.48).

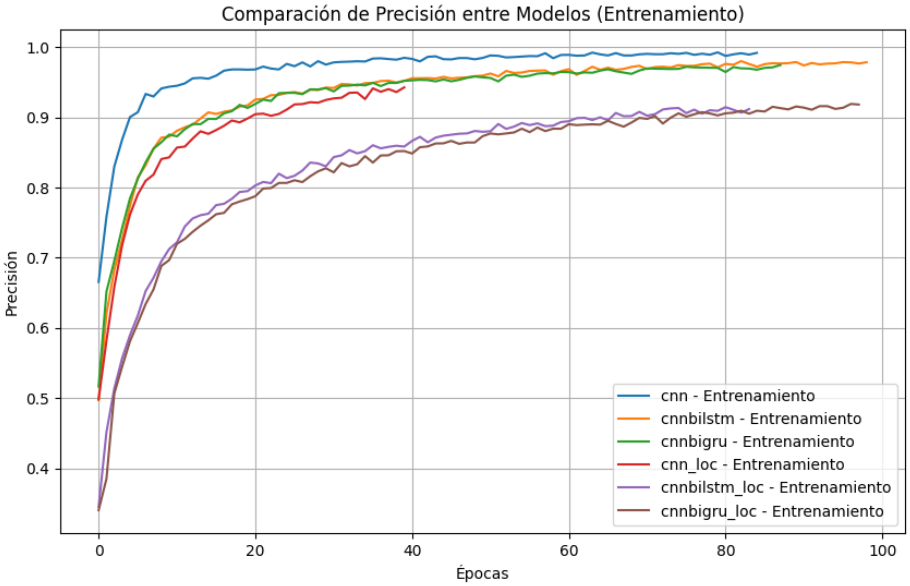

Figura 5.45: Comparación de la precisión en el entrenamiento de los diferentes modelos.

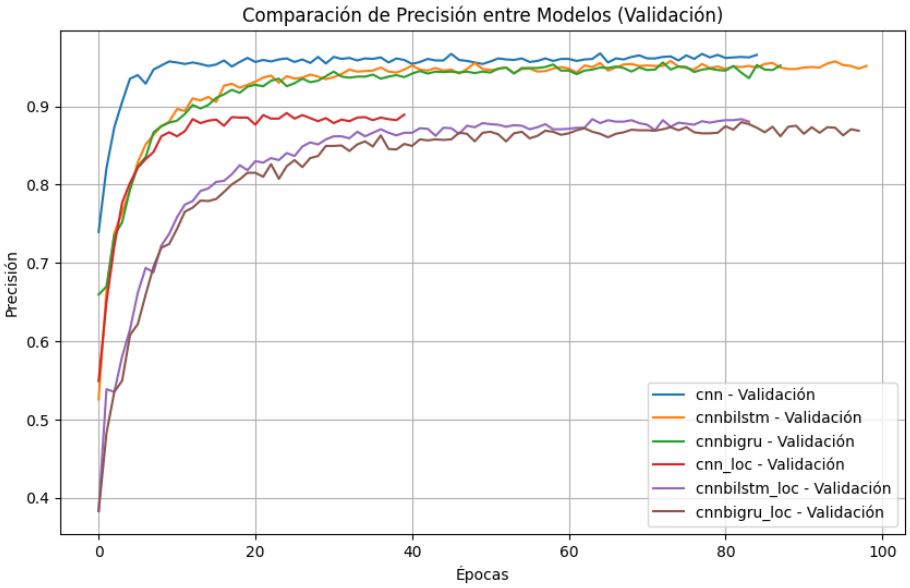

Figura 5.46: Comparación de la precisión en la validación de los diferentes modelos.

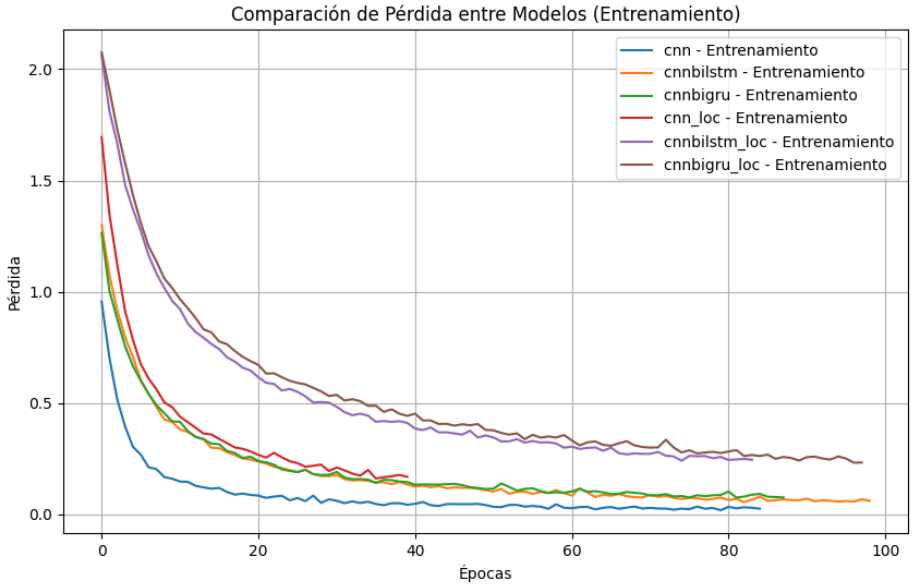

Figura 5.47: Comparación de la pérdida en el entrenamiento de los diferentes modelos.

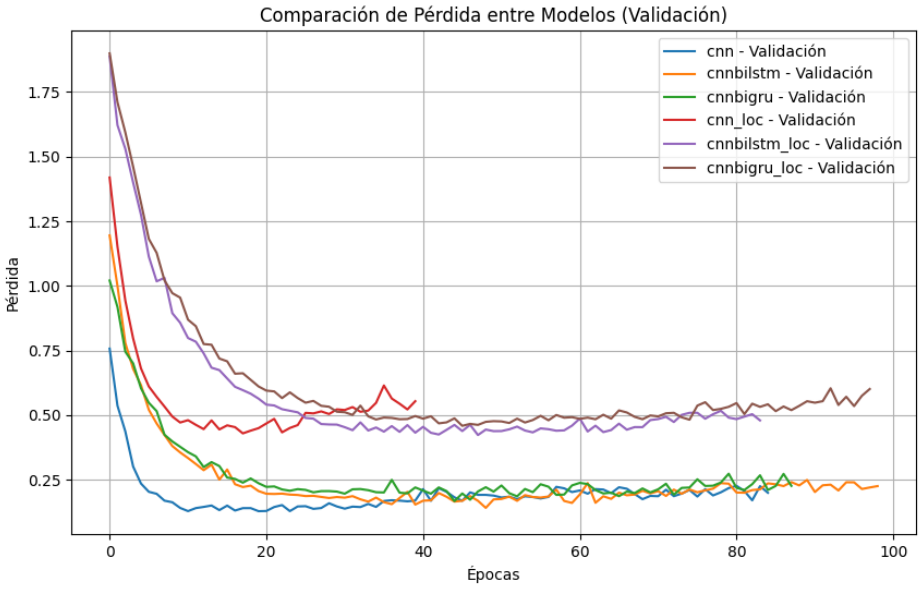

Figura 5.48: Comparación de la pérdida en la validación de los diferentes modelos.

5.4. Optimización del modelo seleccionado

Como se ha visto en la sección anterior, la Red Neuronal Convolucional ha demostrado ser el modelo más eficiente en términos de rendimiento para la detección de

actividades humanas a partir de los datos obtenidos de los sensores. Sin embargo, a pesar de estos buenos resultados, es imprescindible llevar a cabo una fase de **optimización en la que se ajusten diversos hiperparámetros** con el objetivo de mejorar aún más su rendimiento.

Para llevar a cabo esta optimización, se compararán los resultados obtenidos de la CNN mediante la variación de alguno de sus hiperparámetros. Las pruebas se realizaron tanto con el conjunto de datos con localización como con el dataset sin localización, pero dado que los valores óptimos fueron los mismos en ambos casos, solo se mostrarán los resultados del dataset sin la información de localización para evitar redundancias. Se evaluarán diferentes configuraciones y, para cada parámetro, se seleccionará el valor con mejores resultados en términos de rendimiento, resaltando el mejor resultado en rojo para facilitar la lectura.

5.4.1. Optimización del número de filtros (*filters*)

En la tabla 5.21 se muestran los resultados obtenidos al modificar el **número de filtros** en la capa convolucional de la CNN. Antes de la optimización, el modelo utilizaba 256 filtros, pero se observa como al aumentar este valor a 512 hay una ligera mejora de rendimiento en cuanto a la puntuación F1. Aumentar este valor aún más no aporta mejoras y, además, incrementa el tiempo de ejecución notablemente.

Filtros	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
64	0.9587	0.9566	0.9587	0.9563	0.1212	22.90	46
128	0.9619	0.9603	0.9619	0.9606	0.1163	23.71	42
256	0.9677	0.9652	0.9677	0.9651	0.1987	64.51	85
512	0.9677	0.9663	0.9677	0.9663	0.1572	125.06	71
1024	0.9671	0.9655	0.9671	0.9660	0.1434	324.52	53

Tabla. 5.21: Resultados del modelo CNN con distinto número de filtros en la capa convolucional.

5.4.2. Optimización del tamaño del kernel (*kernel size*)

En la tabla 5.22 se presentan los resultados obtenidos al modificar el **tamaño del *kernel* en la capa convolucional**. Se observa que el mejor rendimiento se obtiene con un tamaño de kernel de 5. Disminuir este hiperparámetro provoca una disminución de rendimiento, aunque reduce el tiempo de entrenamiento. Aumentar este valor no provoca ninguna mejora significativa, además de aumentar el tiempo de ejecución.

Kernel size	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
3	0.9612	0.9603	0.9612	0.9597	0.1469	68.05	48
5	0.9677	0.9663	0.9677	0.9663	0.1572	125.06	71
7	0.9651	0.9639	0.9651	0.9642	0.1430	118.35	55
11	0.9632	0.9602	0.9632	0.9607	0.1393	117.27	41

Tabla. 5.22: Resultados del modelo CNN con distintos tamaños de kernel.

5.4.3. Optimización del tamaño de la capa de poolig (*pool size*)

En la tabla 5.23 se ven los resultados obtenidos al variar el **tamaño de la capa de *pooling*** en la CNN. Nuevamente, un valor intermedio vuelve a dar los mejores resultados, en este caso con un tamaño de pool de 2, un valor que ya se estaba utilizando antes de iniciar la optimización. Al aumentar o disminuir este valor, el rendimiento se ve afectado negativamente, empeorando en la mayoría de métricas de rendimiento.

Pool size	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
1	0.9457	0.9413	0.9457	0.9414	0.3656	198.38	45
2	0.9677	0.9663	0.9677	0.9663	0.1572	125.06	71
3	0.9554	0.9522	0.9554	0.9526	0.1509	50.47	36

Tabla. 5.23: Resultados del modelo CNN con distintos tamaños de la capa de pooling.

5.4.4. Optimización del *dropout rate*

En la tabla 5.24 se pueden analizar los resultados obtenidos al modificar la **tasa de *dropout* en la CNN**. Se observa que el valor óptimo es 0.4, logrando el mejor rendimiento y evitando tanto el sobreajuste como la pérdida de demasiada información durante el entrenamiento.

Dropout rate	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
0.3	0.9612	0.9583	0.9612	0.9590	0.1473	76.83	39
0.4	0.9677	0.9663	0.9677	0.9663	0.1572	125.06	71
0.5	0.9651	0.9630	0.9651	0.9630	0.1817	156.70	77

Tabla. 5.24: Resultados del modelo CNN con distinto *dropout rate*.

5.4.5. Optimización de la dimensión de las capas densas (*ff dim*)

En la tabla 5.25, se muestran los resultados obtenidos al **modificar la dimensión de las capas densas** (*ff_dim*) en la CNN. Se observa que el mejor valor es 128, y disminuir este valor solo conseguiría empeorar los resultados. Al aumentar el valor se consiguen mejores resultados en cuanto a tiempo de ejecución, a cambio de empeorar el resto de métricas.

ff_dim	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
64	0.9651	0.9638	0.9651	0.9638	0.1462	128.49	68
128	0.9677	0.9663	0.9677	0.9663	0.1572	125.06	71
256	0.9638	0.9625	0.9638	0.9629	0.1283	88.53	44
512	0.9606	0.9567	0.9651	0.9573	0.1282	72.29	34

Tabla. 5.25: Resultados del modelo CNN con distintas dimensiones de las capas densas.

5.4.6. Optimización del tamaño del lote (*batch size*)

En la tabla 5.26 se muestran los resultados obtenidos al modificar el **tamaño del lote en la CNN**. Se observa que el mejor valor era el que ya se estaba utilizando (64). Valores más pequeños como 32 muestran un descenso en las medidas de rendimiento, al igual que valores más grandes como 128, 256 y 512, que aunque ofrecen menores tiempos de ejecución, no logran superar el rendimiento obtenido con *batch size* 64.

batch size	Accuracy	Precision	Recall	F1-Score	Loss	Tiempo (s)	épocas
32	0.9632	0.9611	0.9632	0.9614	0.1934	139.92	62
64	0.9677	0.9663	0.9677	0.9663	0.1572	125.06	71
128	0.9625	0.9596	0.9625	0.9598	0.1173	88.20	47
256	0.9664	0.9652	0.9664	0.9645	0.1188	88.92	56
512	0.9619	0.9598	0.9619	0.9602	0.1225	100.46	65

Tabla. 5.26: Resultados del modelo CNN con distinto tamaño de lote.

Tras la optimización de los distintos hiperparámetros, se ha logrado identificar la mejor configuración para obtener un equilibrio entre rendimiento y eficiencia computacional para la Red Neuronal Convolucional, que se muestra en la tabla 5.27.

Hiperparámetro	**Valor óptimo**
Número de filtros (*filters*)	512
Tamaño del kernel (*kernel size*)	5
Tamaño del *pooling* (*pool size*)	2
Tasa de *dropout* (*dropout rate*)	0.4
Dimensión de las capas densas (*ff_dim*)	128
Tamaño del lote (*batch size*)	64

Tabla. 5.27: Resumen de los valores óptimos de los hiperparámetros de la CNN.

Capítulo 6

CONCLUSIONES

Desde el inicio del trabajo, se plantearon las razones que motivaron su desarrollo, resaltando la importancia de detectar actividades humanas en el hogar mediante sensores y aprendizaje automático. A lo largo del documento, se fueron explicando los conceptos clave relacionados con la inteligencia artificial, aprendizaje automático y aprendizaje profundo, así como las técnicas más utilizadas en el reconocimiento de actividades humanas.

También se establecieron los objetivos que guiaron todo el proceso, desde la selección y preparación de los datos hasta la implementación de distintos modelos de redes neuronales. Se detalló cómo se trabajó con los datos recogidos por los sensores, el preprocesamiento necesario para su uso en los modelos y la configuración de las distintas arquitecturas probadas.

Por último, se analizaron los resultados obtenidos, comparando diferentes configuraciones y evaluando el impacto de incluir la información de localización en la precisión de los modelos. En este capítulo, primero se hará una evaluación de los objetivos que fueron propuestos con el fin de verificar su cumplimiento y, finalmente, se describen un conjunto de propuestas a trabajo futuro que pueden derivarse del desarrollo actual.

6.1. Evaluación del cumplimiento de los objetivos

En el capítulo 3 se definieron los objetivos generales y específicos de este trabajo, cuyo propósito era diseñar e implementar un sistema basado en Aprendizaje Automático para la detección y clasificación de actividades humanas en el hogar utilizando datos recopilados por sensores. A lo largo de esta memoria se han realizado los

distintos pasos necesarios para lograr este propósito, desde la adquisición de conocimientos teóricos, el estudio y el análisis del conjunto de datos, el preprocesamiento de este conjunto de datos, el estudio de la segmentación de los datos en ventanas temporales, hasta la selección, definición, entrenamiento y validación de un algoritmo de aprendizaje automático.

El primer objetivo específico consistía en **adquirir conocimientos en HAR y ML aplicado a sensores**. Para ello se llevó a cabo una revisión del estado del arte, abarcando los temas más importantes como pueden ser la Inteligencia Artificial, el Aprendizaje Automático, el Aprendizaje Profundo, las Redes Neuronales y, por último, el Reconocimiento de Actividades Humanas y su relación con el Aprendizaje Automático. Para recopilar toda esta información y adquirir el conocimiento necesario, se consultaron numerosos artículos, libros y recursos en línea, todos ellos citados a lo largo del documento.

El **análisis y comprensión del conjunto de datos** fue otro de los objetivos clave. Para ello, se realizó un estudio de los sensores empleados, identificando sus tipos, ubicaciones y las mediciones registradas. Se analizaron las variables recogidas por cada sensor, diferenciando entre aquellas de tipo ambiental, de movimiento, de consumo energético y de localización. Además, se tuvieron en cuenta las anotaciones registradas en los conjuntos de datos *Ground Truth*, que posteriormente nos ayudarían a la asignación de etiquetas a los datos de los sensores.

En cuanto al **preprocesamiento de los datos de los sensores**, inicialmente se ajustaron las marcas temporales y se filtraron las mediciones de los sensores al periodo de estudio de dos semanas. Más tarde, se eliminaron los datos del sensor de consumo energético de la lavadora por su irrelevancia. Después se realizó la asignación de etiquetas a los datos de los sensores, según los intervalos de tiempo definidos en los datasets *Ground Truth*. Por último, se añadió, al conjunto de datos de los sensores ya procesado y etiquetado, la localización de los residentes mediante los valores RSSI medidos.

Con los conjuntos de datos preprocesados y listos para ser usados, se procedió a la **segmentacion temporal de los datos diviéndolos en ventanas temporales**, necesaria para estructurar de manera correcta la información. Además, se han analizado y probado varios tamaños de secuencias temporales, evaluando cuál podría ser el valor óptimo para conseguir los mejores rendimientos en los modelos de ML.

Para **seleccionar un modelo de reconocimiento de actividades humanas**, se han implementado y comparado diversas arquitecturas, incluyendo CNN, RNN, LSTM, GRU, LSTM bidireccional, GRU bidireccional y combinaciones híbridas como CNN-

BiLSTM y CNN-BiGRU.

Todos estos modelos fueron **entrenados y validados** utilizando métricas estándar en aprendizaje automático (explicadas en la sección 4.5.1), con estas medidas se consiguió evaluar el rendimiento de cada modelo en la clasificación de actividades humanas.

Una vez seleccionado el modelo con mejores resultados en cuanto a rendimiento, se llevó a cabo una optimización del modelo, ajustando los diferentes hiperparámetros definidos, buscando mejorar la capacidad del modelo para identificar actividades humanas.

En conclusión, todos los objetivos planteados en este trabajo han sido alcanzados, permitiendo así demostrar la viabilidad del uso de modelos de ML en el reconocimiento de actividades humanas y estableciendo un punto de partida para futuras investigaciones.

6.2. Propuestas a futuro

A partir del trabajo desarrollado, es posible incorporar nuevas mejoras y ajustar ciertas decisiones tomadas previamente, con el objetivo de optimizar los resultados obtenidos. En esta sección, se presentan diversas estrategias que podrían implementarse en el futuro para mejorar el rendimiento del modelo seleccionado.

Mejora en la metodología de localización basada en RSSI

Para este trabajo, la localización de los residentes se ha obtenido a partir de una metodología basada en la medición de señales RSSI utilizando un esquema de balizas y anclas (explicado en la subsección 4.2.1). En este sistema, las balizas corresponden a las pulseras de actividad, que emiten señales BLE, mientras que las anclas son dispositivos fijos ubicados en distintas zonas de la vivienda y se encargan de escanear y registrar la señal emitida por las pulseras. Cada muestra de datos RSSI recogida contiene información sobre qué ancla ha detectado la señal y a qué pulsera corresponde, permitiendo inferir la ubicación aproximada del residente.

Sin embargo, este enfoque presenta ciertas limitaciones. Debido a que las pulseras utilizadas no permiten configurar parámetros como la potencia de emisión, haciendo que la precisión del modelo pueda verse afectada. Y otros dispositivos que sí se po-

drían configurar, tienen el inconveniente de que la autonomía de su batería es muy baja, llegando a durar un solo día en la mayoría de los casos y siendo poco práctico para, por ejemplo, personas mayores.

Una posible mejora para mejorar la precisión de la localización sería invertir el esquema actual, desplegando balizas fijas en distintas zonas de la vivienda y utilizando las pulseras de los residentes como dispositivos de escaneo. De esta forma, las pulseras podrían detectar y registrar las señales de todas las balizas del entorno, mejorando la precisión de la localización. Además, con el desarrollo de nuevos dispositivos enfocados en este uso, es decir, que tengan la posibilidad de configurarse y su autonomía sea mayor, se podría aplicar este nuevo esquema que podría lograr una mayor precisión en la localización y mejorar el rendimiento del sistema de reconocimiento de actividades humanas.

Generalización del modelo a otras viviendas mediante adaptación de dominio

La **adaptación de dominio** [125] es una técnica de ML que busca mejorar la capacidad de generalización de un modelo cuando los datos de entrenamiento y prueba provienen de distribuciones ligeramente diferentes. El sistema desarrollado en este trabajo ha sido diseñado y evaluado en una única vivienda, lo que puede llegar a limitar su aplicación en otros entornos, debido a la distribución del espacio y la disposición de los sensores en otras viviendas.

Para este problema sería interesante utilizar técnicas de adaptación de dominio que permitan transferir el conocimiento aprendido en un entorno a otro sin necesidad de recopilar y etiquetar grandes volúmenes de datos adicionales.

Uso de sensores complementarios para mejorar la detección de actividades

A los sensores que se utilizan se podría añadir una serie de sensores complementarios que proporcionarían información valiosa y mejorarían la precisión en el reconocimiento de actividades humanas. Algunos de estos sensores pueden ser:

- **Sensores de presión**: se pueden colocar en sillas, camas o zonas del suelo para detectar cuándo una persona se sienta, se acuesta o camina por una zona específica de la vivienda. Permitiendo distinguir con mayor precisión actividades como dormir, comer, etc.

- **Acelerómetros y giroscopios**: Se podrían añadir en sensores vestibles y podrían proporcionar información detallada sobre el movimiento del usuario, detectando actividades como caminar, sentare, levantarse e incluso caídas.

Bibliografía

[1] High-Level Expert Group on Artificial Intelligence. Ethics guidelines for trustworthy artificial intelligence. *European Commission Publications*, 2021. URL https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai.

[2] Zeeshan Arif. Building the future with supervised learning: Unleashing the potential of labeled data, 2023. URL https://medium.com/@mr-zeeshan-arif/building-the-future-with-supervised-learning-unleashing-the-potential-of-labele

[3] Yingying Liao, Lei Han, Haoyu Wang, and Hougui Zhang. Prediction models for railway track geometry degradation using machine learning methods: A review. *Sensors*, 22:7275, 09 2022. doi: 10.3390/s22197275.

[4] Roman Panarin. Semi-supervised learning explained: Techniques and real-world applications, 2024. URL https://maddevs.io/blog/semi-supervised-learning-explained/.

[5] Amarnag Subramanya and Partha Pratim Talukdar. *Graph-based semi-supervised learning*. Morgan & Claypool Publishers, 2014. URL https://www.odbms.org/wp-content/uploads/2014/08/Subramanya_Ch1.pdf.

[6] AI Planet. Introducción al aprendizaje no supervisado, 2023. URL https://aiplanet.com/learn/unsupervised-learning-es/introduccion-al-aprendizaje-no-supervisado/1616/introduccion-al-aprendizaje-no-supervisado.

[7] Fernando. K-means, 2022. URL https://rpubs.com/josefer09/864491.

[8] Jacob Murel and Eda Kavlakoglu. What is reinforcement learning?, 2024. URL https://www.ibm.com/think/topics/reinforcement-learning.

[9] Jieun Baek and Yosoon Choi. Deep neural network for predicting ore production by truck-haulage systems in open-pit mines. *Applied Sciences*, 10:1657, 03 2020. doi: 10.3390/app10051657.

[10] UC Business Analytics R Programming Guide. Artificial neural network fundamentals, 2017. URL https://uc-r.github.io/ann_fundamentals.

[11] Lino Manjarrez. *Relaciones Neuronales Para Determinar la Atenuación del Valor de la Aceleración Máxima en Superficie de Sitios en Roca Para Zonas de Subducción*. PhD thesis, 06 2014.

[12] Facundo Segura, Florencio Segura, María Zudaire, Florencio Segura, Rocío Mendía, Lucía Falco, Paula Zalazar, and Daniel Sequeira. Clasificación avanzada de la artrosis de rodilla utilizando tecnologías de inteligencia artificial. *Revista de la Asociación Argentina de Ortopedia y Traumatología*, 89:462–469, 10 2024. doi: 10.15417/issn.1852-7434.2024.89.5.1993.

[13] Amir Sahraei, Tobias Houska, and Lutz Breuer. Deep learning for isotope hydrology: The application of long short-term memory to estimate high temporal resolution of the stable isotope concentrations in stream and groundwater. *Frontiers in Water*, 3, 09 2021. doi: 10.3389/frwa.2021.740044.

[14] Saeed Mohsen. Recognition of human activity using gru deep learning algorithm. *Multimedia Tools and Applications*, 82, 05 2023. doi: 10.1007/s11042-023-15571-y.

[15] Tiantian Guo, Jianzhuo Yan, Jianhui Chen, and Yongchuan Yu. Overflow capacity prediction of pumping station based on data drive. *Water*, 15:2380, 06 2023. doi: 10.3390/w15132380.

[16] Jungpil Shin, Najmul Hassan, Abu Saleh Musa Miah, and Satoshi Nishimura. A comprehensive methodological survey of human activity recognition across diverse data modalities. *arXiv preprint*, arXiv:2409.09678, 2024. URL https://ar5iv.org/html/2409.09678.

[17] Daniele Liciotti, Michele Bernardini, Luca Romeo, and Emanuele Frontoni. A sequential deep learning application for recognising human activities in smart homes. *Neurocomputing*, 396:501–513, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.10.104. URL https://www.sciencedirect.com/science/article/pii/S0925231219304862.

[18] Ahmed Nait Aicha, Gwenn Englebienne, Kimberley S. Van Schooten, Mirjam Pijnappels, and Ben Kröse. Deep learning to predict falls in older adults based on daily-life trunk accelerometry. *Sensors*, 18(5), 2018. ISSN 1424-8220. doi: 10.3390/s18051654. URL https://www.mdpi.com/1424-8220/18/5/1654.

[19] Joost N Kok, Egbert J Boers, Walter A Kosters, Peter Van der Putten, and Mannes Poel. Artificial intelligence: definition, trends, techniques, and ca-

ses. *Artificial intelligence*, 1(270-299):51, 2009. URL https://www.eolss.net/Sample-Chapters/C15/E6-44.pdf.

[20] Haroon Sheikh, Corien Prins, and Erik Schrijvers. *Artificial Intelligence: Definition and Background*, pages 15–41. Springer International Publishing, Cham, 2023. ISBN 978-3-031-21448-6. doi: 10.1007/978-3-031-21448-6_2. URL https://doi.org/10.1007/978-3-031-21448-6_2.

[21] ICCSI Centro de Innovación. Conferencia de dartmouth: El nacimiento de la inteligencia artificial, 2024. URL https://iccsi.com.ar/dartmouth-inteligencia-artificial/.

[22] James Moor. The dartmouth college artificial intelligence conference: The next fifty years. *AI Magazine*, 27(4):87, Dec. 2006. doi: 10.1609/aimag.v27i4.1911. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1911.

[23] M. González Castillo and R. Vintimilla Vásquez. Historia y evolución de la inteligencia artificial. *Bits: Revista de ciencias de la computación con aplicaciones a ingeniería de software*, 1(1):5–18, 2017. URL https://revistasdex.uchile.cl/index.php/bits/article/view/2767/2700.

[24] Michail Kaseris, Ioannis Kostavelis, and Sotiris Malassiotis. A comprehensive survey on deep learning methods in human activity recognition. *Machine Learning and Knowledge Extraction*, 6(2):842–876, 2024. ISSN 2504-4990. doi: 10.3390/make6020040. URL https://www.mdpi.com/2504-4990/6/2/40.

[25] Wenbin Gao, Lei Zhang, Wenbo Huang, Fuhong Min, Jun He, and Aiguo Song. Deep neural networks for sensor-based human activity recognition using selective kernel convolution. *IEEE Transactions on Instrumentation and Measurement*, 70:1–13, 2021. ISSN 1557-9662. doi: 10.1109/TIM.2021.3102735.

[26] Kun Xia, Jianguang Huang, and Hanyu Wang. Lstm-cnn architecture for human activity recognition. *IEEE Access*, 8:56855–56866, 2020. URL https://api.semanticscholar.org/CorpusID:215722530.

[27] Repuri Mohan Vamsi, Neha Adapa, Dinesh Yelamanchili, Nurul Amin Choudhury, and Badal Soni. An efficient and optimized cnn-lstm framework for complex human activity recognition system using surface emg physiological sensors and feature engineering. In *2024 IEEE Students Conference on Engineering and Systems (SCES)*, pages 1–6, 2024. doi: 10.1109/SCES61914.2024.10652396.

[28] Hui Chen, Charles Gouin-Vallerand, Kévin Bouchard, Sébastien Gaboury, Mélanie Couture, Nathalie Bier, and Sylvain Giroux. Leveraging self-supervised learning for human activity recognition with ambient sensors. In *Proceedings of the 2023 ACM Conference on Information Technology for Social Good*, GoodIT '23, page 324–332, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701160. doi: 10.1145/3582515.3609551. URL https://doi.org/10.1145/3582515.3609551.

[29] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015. doi: 10.1126/science.aaa8415. URL https://www.science.org/doi/abs/10.1126/science.aaa8415.

[30] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN 9780071154673. URL https://books.google.es/books?id=EoYBngEACAAJ.

[31] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. *Supervised Learning*, pages 21–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-75171-7. doi: 10.1007/978-3-540-75171-7_2. URL https://doi.org/10.1007/978-3-540-75171-7_2.

[32] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook, 2nd ed*. 01 2010. ISBN 9780387098227.

[33] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. ISSN 1573-0565. doi: 10.1007/BF00116251. URL https://doi.org/10.1007/BF00116251.

[34] Thomas Rincy N and Roopam Gupta. A survey on machine learning approaches and its techniques:. In *2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*, pages 1–6, Feb 2020. doi: 10.1109/SCEECS48394.2020.190.

[35] Shenglei Chen, Geoffrey I. Webb, Linyuan Liu, and Xin Ma. A novel selective naïve bayes algorithm. *Knowledge-Based Systems*, 192:105361, 2020. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2019.105361. URL https://www.sciencedirect.com/science/article/pii/S0950705119306185.

[36] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001. doi: 10.1.1.330.2788. URL https://www.dors.it/documentazione/testo/201911/10.1.1.330.2788.pdf.

[37] Kashvi Taunk, Sanjukta De, Srishti Verma, and Aleena Swetapadma. A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1255–1260, May 2019. doi: 10.1109/ICCS45141.2019.9065747.

[38] Yuanming Shi, Kai Yang, Zhanpeng Yang, and Yong Zhou. *Mobile Edge Artificial Intelligence*. Academic Press, 2021. ISBN 978-0-12-823817-2. doi: 10.1016/C2020-0-00624-9. URL https://www.sciencedirect.com/book/9780128238172/mobile-edge-artificial-intelligence.

[39] Corinna Cortes. Support-vector networks. *Machine Learning*, 1995. URL https://ise.ncsu.edu/wp-content/uploads/sites/9/2022/08/Cortes-Vapnik1995_Article_Support-vectorNetworks.pdf.

[40] Yanbei Chen, Massimiliano Mancini, Xiatian Zhu, and Zeynep Akata. Semi-supervised and unsupervised deep visual learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(3):1327–1347, 2024. doi: 10.1109/TPAMI.2022.3201576.

[41] Yifan Wang, Yan Huang, Qicong Wang, Chong Zhao, Zhenchang Zhang, and Jian Chen. Graph-based self-training for semi-supervised deep similarity learning. *Sensors*, 23(8), 2023. ISSN 1424-8220. doi: 10.3390/s23083944. URL https://www.mdpi.com/1424-8220/23/8/3944.

[42] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. Graph-based semi-supervised learning: A comprehensive review. *arXiv preprint arXiv:2102.13303*, 2021. URL https://arxiv.org/pdf/2102.13303. Los autores Z. Song e I. King están afiliados al Departamento de Ciencia e Ingeniería de la Computación, La Universidad China de Hong Kong, Hong Kong, China. X. Yang está afiliado al Laboratorio SMILE, Escuela de Ciencia e Ingeniería de la Computación, Universidad de Ciencia y Tecnología Electrónica de China, Chengdu, China. Z. Xu está afiliado a la Escuela de Ciencia y Tecnología de la Computación, Instituto de Tecnología de Harbin, Shenzhen, China, y también con el Laboratorio Peng Cheng, Shenzhen, China. [1].

[43] Zoubin Ghahramani. *Unsupervised Learning*, pages 72–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-28650-9. doi: 10.1007/978-3-540-28650-9_5. URL https://doi.org/10.1007/978-3-540-28650-9_5.

[44] Peter Dayan, Maneesh Sahani, and Grégoire Deback. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, pages 857–859, 1999. URL https://web.math.princeton.edu/~sswang/developmental-diaschisis-references/dun99b.pdf.

[45] Hans-Hermann Bock. *Clustering Methods: A History of k-Means Algorithms*, pages 161–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-73560-1. doi: 10.1007/978-3-540-73560-1_15. URL https://doi.org/10.1007/978-3-540-73560-1_15.

[46] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8), 2020. ISSN 2079-9292. doi: 10.3390/electronics9081295. URL https://www.mdpi.com/2079-9292/9/8/1295.

[47] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996. URL https://doi.org/10.1613/jair.301.

[48] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999. URL https://www.researchgate.net/publication/270960086_Reinforcement_learning.

[49] Jan Egger, Antonio Pepe, Christina Gsaxner, Yuan Jin, Jianning Li, and Roman Kern. Deep learning—a first meta-survey of selected reviews across scientific disciplines, their commonalities, challenges and research impact. *PeerJ Computer Science*, 7:e773, November 2021. ISSN 2376-5992. doi: 10.7717/peerj-cs.773. URL http://dx.doi.org/10.7717/peerj-cs.773.

[50] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, April 2021. ISSN 1422-8890. doi: 10.1007/s12525-021-00475-2. URL http://dx.doi.org/10.1007/s12525-021-00475-2.

[51] Neha Gupta, Suneet K. Gupta, Rajesh K. Pathak, Vanita Jain, Parisa Rashidi, and Jasjit S. Suri. Human activity recognition in artificial intelligence framework: a narrative review. *Artificial Intelligence Review*, 55(6):4755–4808, 2022. ISSN 1573-7462. doi: 10.1007/s10462-021-10116-x. URL https://doi.org/10.1007/s10462-021-10116-x.

[52] Martin C. Nwadiugwu. Neural networks, artificial intelligence and the computational brain, 2020. URL https://arxiv.org/abs/2101.08635.

[53] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017. URL https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf.

[54] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A. Chavez-Urbiola, and Julio A. Romero-Gonzalez. Loss functions and metrics in deep learning, 2024. URL https://arxiv.org/abs/2307.02694.

[55] Timothy O. Hodson, Thomas M. Over, and Sydney S. Foks. Mean squared error, deconstructed. *Journal of Advances in Modeling Earth Systems*, 13(12):e2021MS002681, 2021. doi: https://doi.org/10.1029/2021MS002681. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021MS002681. e2021MS002681 2021MS002681.

[56] Cort J. Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005. ISSN 0936577X, 16161572. URL http://www.jstor.org/stable/24869236.

[57] Qiufu Li, Xi Jia, Jiancan Zhou, Linlin Shen, and Jinming Duan. Rediscovering bce loss for uniform classification, 2024. URL https://arxiv.org/abs/2403.07289.

[58] Claudio Gentile and Manfred K. K Warmuth. Linear hinge loss and average margin. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1998. URL https://proceedings.neurips.cc/paper_files/paper/1998/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[59] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/f2925f97bc13ad2852a7a551802feea0-Paper.pdf.

[60] Ruslan Abdulkadirov, Pavel Lyakhov, and Nikolay Nagornov. Survey of optimization algorithms in modern neural networks. *Mathematics*, 11(11), 2023. ISSN 2227-7390. doi: 10.3390/math11112466. URL https://www.mdpi.com/2227-7390/11/11/2466.

[61] Yingjie Tian, Yuqi Zhang, and Haibin Zhang. Recent advances in stochastic gradient descent in deep learning. *Mathematics*, 11(3), 2023. ISSN 2227-7390. doi: 10.3390/math11030682. URL https://www.mdpi.com/2227-7390/11/3/682.

[62] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information*

Processing Systems, volume 33, pages 18261–18271. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/d3f5d4de09ea19461dab00590df91e4f-Paper.pdf.

[63] N. Zhang, D. Lei, and J.F. Zhao. An improved adagrad gradient descent optimization algorithm. In *2018 Chinese Automation Congress (CAC)*, pages 2359–2362, 2018. doi: 10.1109/CAC.2018.8623271.

[64] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[65] J. V. Burke and M. C. Ferris. A gauss–newton method for convex composite optimization. *Mathematical Programming*, 71(2):179–194, Dec 1995. ISSN 1436-4646. doi: 10.1007/BF01585997. URL https://doi.org/10.1007/BF01585997.

[66] Yu-Hong Dai. Convergence properties of the bfgs algoritm. *SIAM Journal on Optimization*, 13(3):693–701, 2002. doi: 10.1137/S1052623401383455. URL https://doi.org/10.1137/S1052623401383455.

[67] Luis Japa, Marcello Serqueira, Israel MendonçA, Masayoshi Aritsugi, Eduardo Bezerra, and Pedro Henrique González. A population-based hybrid approach for hyperparameter optimization of neural networks. *IEEE Access*, 11: 50752–50768, 2023. ISSN 2169-3536. doi: 10.1109/access.2023.3277310. URL http://dx.doi.org/10.1109/ACCESS.2023.3277310.

[68] Ilya Loshchilov and Frank Hutter. Cma-es for hyperparameter optimization of deep neural networks, 2016. URL https://arxiv.org/abs/1604.07269.

[69] Alexandru Korotcov, Valery Tkachenko, Daniel P. Russo, and Sean Ekins. Comparison of deep learning with multiple machine learning methods and metrics using diverse drug discovery data sets. *Molecular Pharmaceutics*, 14(12):4462–4475, 2017. ISSN 1543-8384. doi: 10.1021/acs.molpharmaceut.7b00578. URL https://doi.org/10.1021/acs.molpharmaceut.7b00578.

[70] Mohammad-Parsa Hosseini, Senbao Lu, Kavin Kamaraj, Alexander Slowikowski, and Haygreev C. Venkatesh. *Deep Learning Architectures*, pages 1–24. Springer International Publishing, Cham, 2020. ISBN 978-3-030-31756-0. doi: 10.1007/978-3-030-31756-0_1. URL https://doi.org/10.1007/978-3-030-31756-0_1.

[71] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, Dec 2022. ISSN 2162-2388. doi: 10.1109/TNNLS.2021.3084827.

[72] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks, 2017. URL https://arxiv.org/abs/1703.06211.

[73] Perry Gibson, José Cano, Jack Turner, Elliot Crowley, Michael O'Boyle, and Amos Storkey. Optimizing grouped convolutions on edge devices. 06 2020. doi: 10.1109/ASAP49362.2020.00039.

[74] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks, 2018. URL https://arxiv.org/abs/1801.01078.

[75] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019. URL https://arxiv.org/abs/1909.09586.

[76] Christian Bakke Vennerød, Adrian Kjærran, and Erling Stray Bugge. Long short-term memory rnn, 2021. URL https://arxiv.org/abs/2105.06756.

[77] Theyazn H. H. Aldhyani and Hasan Alkahtani. A bidirectional long short-term memory model algorithm for predicting covid-19 in gulf countries. *Life*, 11(11), 2021. ISSN 2075-1729. doi: 10.3390/life11111118. URL https://www.mdpi.com/2075-1729/11/11/1118.

[78] Aniruddha Dutta, Saket Kumar, and Meheli Basu. A gated recurrent unit approach to bitcoin price prediction. *Journal of Risk and Financial Management*, 13(2), 2020. ISSN 1911-8074. doi: 10.3390/jrfm13020023. URL https://www.mdpi.com/1911-8074/13/2/23.

[79] Abdu Gumaei, Mohammad Mehedi Hassan, Abdulhameed Alelaiwi, and Hussain Alsalman. A hybrid deep learning model for human activity recognition using multimodal body sensing data. *IEEE Access*, 7:99152–99160, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2927134.

[80] Chunxu Chai, Chuanxiang Ren, Changchang Yin, Hui Xu, Qiu Meng, Juan Teng, and Ge Gao. A multifeature fusion short-term traffic flow prediction model based on deep learnings. *Journal of Advanced Transportation*, 2022(1):1702766, 2022. doi: https://doi.org/10.1155/2022/1702766. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/1702766.

[81] Farhad Mortezapour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru, 2024. URL https://arxiv.org/abs/2305.17473.

[82] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001. URL https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/Recurrent%20Neural%20Networks%20Design%20And%20Applications%20-%20L.R.%20Medsker.pdf.

[83] Zawar Hussain, Quan Z. Sheng, and Wei Emma Zhang. A review and categorization of techniques on device-free human activity recognition. *Journal of Network and Computer Applications*, 167:102738, October 2020. ISSN 1084-8045. doi: 10.1016/j.jnca.2020.102738. URL http://dx.doi.org/10.1016/j.jnca.2020.102738.

[84] Ong Chin Ann and Lau Bee Theng. Human activity recognition: A review. In *2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*, pages 389–393, 2014. doi: 10.1109/ICCSCE.2014.7072750.

[85] Abhay Gupta, Kuldeep Gupta, Kshama Gupta, and Kapil Gupta. A survey on human activity recognition and classification. In *2020 International Conference on Communication and Signal Processing (ICCSP)*, pages 0915–0919, July 2020. doi: 10.1109/ICCSP48568.2020.9182416.

[86] Ayokunle Olalekan Ige and Mohd Halim Mohd Noor. A survey on unsupervised learning for wearable sensor-based activity recognition. *Applied Soft Computing*, 127:109363, 2022. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2022.109363. URL https://www.sciencedirect.com/science/article/pii/S1568494622005191.

[87] Md Atiqur Rahman Ahad, Anindya Das Antar, and Masud Ahmed. Iot sensor-based activity recognition. *IoT Sensor-Based Activity Recognition*, 2, 2020. URL https://link.springer.com/content/pdf/10.1007/978-3-030-51379-5.pdf.

[88] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2018.02.010. URL https://www.sciencedirect.com/science/article/pii/S016786551830045X. Deep Learning for Pattern Recognition.

[89] Ilias Theodorakopoulos, Dimitris Kastaniotis, George Economou, and Spiros Fotopoulos. Pose-based human action recognition via sparse representation in dissimilarity space. *Journal of Visual Communication and Image Representation*, 25(1):12–23, 2014. ISSN 1047-3203. doi: https://doi.org/10.1016/j.jvcir.2013.03.008. URL https://www.sciencedirect.com/science/article/pii/S1047320313000485. Visual Understanding and Applications with RGB-D Cameras.

[90] Qing Zhou, Jarhinbek Rasol, Yuelei Xu, Zhaoxiang Zhang, and Lujuan Hu. A high-performance gait recognition method based on n-fold bernoulli theory. *IEEE Access*, 10:115744–115757, 2022. URL https://api.semanticscholar.org/CorpusID:252744243.

[91] Fabian Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding*, 06 2015. doi: 10.1109/CVPR.2015.7298698.

[92] Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Limin Wang, and Yu Qiao. Uniformerv2: Spatiotemporal learning by arming image vits with video uniformer, 2022. URL https://arxiv.org/abs/2211.09552.

[93] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017. URL https://arxiv.org/abs/1705.06950.

[94] Yi Wang, Kunchang Li, Xinhao Li, Jiashuo Yu, Yinan He, Chenting Wang, Guo Chen, Baoqi Pei, Ziang Yan, Rongkun Zheng, Jilan Xu, Zun Wang, Yansong Shi, Tianxiang Jiang, Songze Li, Hongjie Zhang, Yifei Huang, Yu Qiao, Yali Wang, and Limin Wang. Internvideo2: Scaling foundation models for multimodal video understanding, 2024. URL https://arxiv.org/abs/2403.15377.

[95] Chunhui Gu, Chen Sun, David A. Ross, Carl Vondrick, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. Ava: A video dataset of spatio-temporally localized atomic visual actions, 2018. URL https://arxiv.org/abs/1705.08421.

[96] Kekai Sheng, Weiming Dong, Chongyang Ma, Xing Mei, Feiyue Huang, and Bao-Gang Hu. Attention-based multi-patch aggregation for image aesthetic assessment. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, page 879–886, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356657. doi: 10.1145/3240508.3240554. URL https://doi.org/10.1145/3240508.3240554.

[97] Jiang Wang, Zicheng Liu, Ying Wu, and Junsong Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1290–1297, 2012. doi: 10.1109/CVPR.2012.6247813.

[98] Amir Shahroudy, Tian-Tsong Ng, Yihong Gong, and Gang Wang. Deep multimodal feature analysis for action recognition in rgb+d videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1045–1058, 2018. doi: 10.1109/TPAMI.2017.2691321.

[99] Jiang wang, Xiaohan Nie, Yin Xia, Ying Wu, and Song-Chun Zhu. Cross-view action modeling, learning and recognition, 2014. URL https://arxiv.org/abs/1405.2941.

[100] Qin Cheng, Jun Cheng, Zhen Liu, Ziliang Ren, and Jianming Liu. A dense-sparse complementary network for human action recognition based on rgb and skeleton modalities. *Expert Systems with Applications*, 244:123061, 2024. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2023.123061. URL https://www.sciencedirect.com/science/article/pii/S0957417423035637.

[101] Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. Utd-mhad: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 168–172, Sep. 2015. doi: 10.1109/ICIP.2015.7350781.

[102] Mengyuan Liu and Junsong Yuan. Recognizing human actions as the evolution of pose estimation maps. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1159–1168, June 2018. doi: 10.1109/CVPR.2018.00127.

[103] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis, 2016. URL https://arxiv.org/abs/1604.02808.

[104] Chunhui Liu, Yueyu Hu, Yanghao Li, Sijie Song, and Jiaying Liu. Pku-mmd: A large scale benchmark for continuous multi-modal human action understanding, 2017. URL https://arxiv.org/abs/1703.07475.

[105] Tianhong Li, Lijie Fan, Mingmin Zhao, Yingcheng Liu, and Dina Katabi. Making the invisible visible: Action recognition through walls and occlusions, 2019. URL https://arxiv.org/abs/1909.09300.

[106] Emiro De-La-Hoz-Franco, Paola Ariza-Colpas, Javier Medina Quero, and Macarena Espinilla. Sensor-based datasets for human activity recognition – a systematic review of literature. *IEEE Access*, 6:59192–59210, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2873502.

[107] D. J. Cook and M. Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of Information in Medicine*, 48(05):480–485, 2009. ISSN 0026-1270, 2511-705X. doi: 10.3414/ME0592. URL http://www.thieme-connect.de/products/ejournals/abstract/10.3414/ME0592.

[108] T. L. M. van Kasteren, G. Englebienne, and B. J. A. Kröse. *Human Activity Recognition from Wireless Sensor Network Data: Benchmark and Software*, pages 165–186. Atlantis Press, Paris, 2011. ISBN 978-94-91216-05-3. doi: 10.2991/978-94-91216-05-3_8. URL https://doi.org/10.2991/978-94-91216-05-3_8.

[109] Mathieu Gallissot Gallissot, Jean Caelen, Nicolas Bonnefond, Brigitte Meillon, and Sylvie Pons. Using the Multicom Domus Dataset. Research Report RR-LIG-020, LIG, 2011. URL https://hal.science/hal-01473142. Les rapports de recherche du LIG - ISSN: 2105-0422.

[110] Hande Alemdar, Halil Ertan, Ozlem Durmaz Incel, and Cem Ersoy. Aras human activity datasets in multiple homes with multiple residents. In *2013 7th International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, pages 232–235, May 2013.

[111] Anthony Fleury, Norbert Noury, and Michel Vacher. Supervised classification of activities of daily living in health smart homes using svm. In *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6099–6102, Sep. 2009. doi: 10.1109/IEMBS.2009.5334931.

[112] Daniel Roggen, Alberto Calatroni, Mirco Rossi, Thomas Holleczek, Kilian Förster, Gerhard Tröster, Paul Lukowicz, David Bannach, Gerald Pirkl, Alois Ferscha, Jakob Doppler, Clemens Holzmann, Marc Kurz, Gerald Holl, Ricardo Chavarriaga, Hesam Sagha, Hamidreza Bayati, Marco Creatura, and José del R. Millàn. Collecting complex activity datasets in highly rich networked sensor environments. In *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, pages 233–240, June 2010. doi: 10.1109/INSS.2010.5573462.

[113] Why Python. Python. *Python releases for windows*, 24, 2021. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f2ee3831eebfc97bfafd514ca2abb7e2c5c86bb.

[114] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckes-

ser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with numpy. *Nature*, 585(7825):357–362, 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[115] Wes Mckinney. pandas: a foundational python library for data analysis and statistics. *Python High Performance Science Computer*, 01 2011. URL https://www.researchgate.net/profile/Wes-Mckinney/publication/265194455_pandas_a_Foundational_Python_Library_for_Data_Analysis_and_Statistics/links/5670827c08ae0d8b0cc0f3cc/pandas-a-Foundational-Python-Library-for-Data-Analysis-and-Statistics.pdf.

[116] Ali Hassan Sial, Syed Yahya Shah Rashdi, and Abdul Hafeez Khan. Comparative analysis of data visualization libraries matplotlib and seaborn in python. *International Journal*, 10(1):277–281, 2021. URL https://www.researchgate.net/publication/349304292_Comparative_Analysis_of_Data_Visualization_Libraries_Matplotlib_and_Seaborn_in_Python_HEC_Y_Cat.

[117] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020. doi: 10.3102/1076998619872761. URL https://doi.org/10.3102/1076998619872761.

[118] Oliver Kramer. *Scikit-Learn*, pages 45–53. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33383-0. doi: 10.1007/978-3-319-33383-0_5. URL https://doi.org/10.1007/978-3-319-33383-0_5.

[119] Sayeth Saabith, T Vinothraj, and M Fareez. A review on python libraries and ides for data science. *Int. J. Res. Eng. Sci*, 9 (11):36–53, 2021. URL https://www.researchgate.net/profile/Vinothraj-Thangarajah/publication/357898994_A_Review_on_Python_Libraries_and_IDEs_for_Data_Science/links/620249344d89183b338b49c2/A-Review-on-Python-Libraries-and-IDEs-for-Data-Science.pdf.

[120] Yuancheng Luo and Ramani Duraiswami. Canny edge detection on nvidia cuda. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, June 2008. doi: 10.1109/CVPRW.2008.4563088.

[121] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning, 2014. URL https://arxiv.org/abs/1410.0759.

[122] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008*, 5(11), 2009. URL https://yaml.org/spec/history/2004-12-28/2004-12-28.pdf.

[123] Charles Severance. Discovering javascript object notation. *Computer*, 45(4): 6–8, April 2012. ISSN 1558-0814. doi: 10.1109/MC.2012.132.

[124] Pau Rodríguez, Miguel A. Bautista, Jordi Gonzàlez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018. ISSN 0262-8856. doi: https://doi.org/10.1016/j.imavis.2018.04.004. URL https://www.sciencedirect.com/science/article/pii/S0262885618300623.

[125] Hao Guan and Mingxia Liu. Domain adaptation for medical image analysis: A survey. *IEEE Transactions on Biomedical Engineering*, 69(3):1173–1185, March 2022. ISSN 1558-2531. doi: 10.1109/TBME.2021.3117407.