Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Aplicación del aprendizaje profundo al reconocimiento de la actividad humana

Trabajo Fin de Máster

Presentado por: Piqueras Segura, Óscar

Director/a: Villalonga Palliser, Claudia

Ciudad: Valencia, España
Fecha: 16 de julio de 2020

Resumen

Este trabajo tiene como objetivo aplicar técnicas de aprendizaje profundo para el reconocimiento de la actividad humana. Se han definido e implementado cuatro modelos de red neuronal y se han entrenado con los datos obtenidos a partir de sensores vestibles, colocados en personas que estaban realizando distintas actividades físicas. El conjunto de datos de entrenamiento contiene información realista, teniendo en cuenta el concepto de desplazamiento gradual. Tras la evaluación de los modelos de red propuestos, se observa que las redes recurrentes, al gestionar mejor las series temporales de datos, tienen mayor precisión a la hora de reconocer las actividades. Por otra parte, se ha comprobado que las redes convolucionales mejoran la extracción de características y, estableciendo modelos híbridos, pueden potenciar la capacidad expresiva de las redes recurrentes.

Palabras Clave: reconocimiento de la actividad humana, serie temporal, red neuronal recurrente, red neuronal convolucional

Abstract

This paper aims to apply deep learning techniques for the recognition of human activity. Four neural network models have been defined and implemented and have been trained with the data obtained from wearable sensors, placed on people who were carrying out different physical activities. The training dataset contains realistic information, taking into account the concept of gradual displacement. After evaluating the proposed network models, it is observed that the recurrent networks, by better managing the time series of data, have greater precision in recognizing the activities. On the other hand, convolutional networks have been shown to improve the extraction of characteristics and, by establishing hybrid models, can enhance the expressive capacity of recurrent networks.

Keywords: human activity recognition, time series, recurrent neural network, convolutional neural network

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

El reconocimiento de la actividad humana (en inglés Human Activity Recognition, HAR) es una área de investigación clave en la interacción humano-computadora (en inglés, Human-Computer Interaction), así como en la computación móvil y ubicua (Hsu et al., 2017).

Tradicionalmente, la investigación en visión artificial ha estado a la vanguardia en este campo. El reconocimiento automático de gestos y actividades a partir de imágenes fijas o video ha sido estudiado por muchos investigadores (Bulling et al., 2014). Pero con el tiempo, los esfuerzos para reconocer actividades de la vida diaria en entornos sin restricciones se han dirigido hacia el uso de sensores inerciales (como acelerómetros o giroscopios) colocados sobre el cuerpo. Esto se debe a varios motivos (Hsu et al., 2017):

- Los sensores reducen las limitaciones ambientales y de configuración que sufren las cámaras de video.
- La adquisición de las señales puede realizarse de una manera más precisa y efectiva, haciendo uso de múltiples sensores.
- La información recogida por los sensores es específica para una persona, por lo que la privacidad es más alta que en las señales adquiridas por una cámara, que pueden contener información de otros sujetos que no estén bajo estudio.

Los investigadores realizaron los primeros estudios sobre el reconocimiento de la actividad utilizando sensores vestibles. Inicialmente la elección de las actividades era arbitraria y no siempre relevante para ser aplicada al mundo real. Pero los avances en el reconocimiento hicieron que cada vez se propusieran escenarios más complejos y orientados a la aplicación real (Bulling et al., 2014).

Existen múltiples dominios que se pueden beneficiar del reconocimiento de la actividad humana (Hsu et al., 2017) (Wang et al., 2019):

- Sector industrial: El reconocimiento de actividades puede ayudar en la gestión del uso de las oficinas o los almacenes. También es de gran utilidad en el entrenamiento de los empleados en determinadas tareas, principalmente relacionadas con líneas de producción.

- Deportes y entretenimiento: Existen multitud de aplicaciones en el campo de los videojuegos inmersivos, así como en la monitorización de actividades deportivas (siendo las pulseras cuantificadoras un claro ejemplo).

- Asistencia sanitaria: Los posibles usos de las técnicas de reconocimiento de actividades van desde la supervisión del lavado de manos o dientes, hasta el seguimiento de actividades que permiten respaldar los métodos de diagnosis o mejorar cómo se realizan los procesos de rehabilitación.

1.1 Motivación

El reconocimiento de la actividad humana se ha tratado habitualmente como un problema de reconocimiento de patrones. Esta estrategia ha avanzado, en gran medida, gracias a la adopción de algoritmos de aprendizaje automático como (Lara & Labrador, 2012):

- Árboles de decisión (en inglés Decision Trees, DTs).

- Máquinas de vector de soporte (en inglés Support Vector Machines, SVMs).

- Clasificadores bayesianos ingenuos (en inglés Naive Bayes Classifiers, NBCs).

- Modelos ocultos de Márkov (en inglés Hidden Markov Models, HMMs).

Sin embargo, este tipo de soluciones basadas en el reconocimiento de patrones, aunque pueden ser satisfactorias en algunos escenarios concretos, tienen una serie de desventajas (Wang et al., 2019):

- Requieren de una extracción heurística y manual de las características. Es decir, se necesita un gran conocimiento y mucha experiencia en el dominio, para poder realizar una extracción de características adecuada. Este conocimiento humano puede ayudar en determinadas tareas muy específicas. Pero para entornos complejos y tareas más generales, se tarda mucho más tiempo en construir un sistema de reconocimiento de actividades y la probabilidad de hacerlo con éxito es mucho menor (Bengio, 2013).

- La experiencia humana necesaria a la hora de extraer características, hace que utilizando estas técnicas sólo se puedan aprender características muy superficiales (Jianbo Yang et al., 2015). Esto provoca que sólo se puedan reconocer actividades de bajo nivel (como caminar o correr) y hace muy difícil reconocer actividades de alto nivel que requieren de contexto (como tomar un café) (Q. Yang, 2009).

- Las técnicas de reconocimiento de patrones suelen necesitar una gran cantidad de datos correctamente etiquetados para entrenar el modelo. Sin embargo, en las aplicaciones reales la mayoría de los datos de actividad no están etiquetados. Esto provoca una disminución del rendimiento de los modelos cuando deben realizar tareas basadas en aprendizaje no supervisado (Bengio, 2013).

- La mayoría de los modelos basados en el reconocimiento de patrones están pensados para aprender de datos estáticos. Pero en la vida real, realizar una actividad genera un flujo continuo de información.

1.2 Planteamiento del trabajo

En la sección 1.1 se han presentado los métodos habituales para enfrentarse al problema del HAR: utilizar técnicas de reconocimiento de patrones mediante algoritmos de aprendizaje automático. También se han mencionado los principales inconvenientes de estos procedimientos.

El aprendizaje profundo ayuda a superar esas limitaciones (Bulling et al., 2014). En las técnicas de aprendizaje profundo, los procedimientos de extracción de características y construcción de modelos a menudo se realizan simultáneamente. Las características se pueden aprender automáticamente a través de una red neuronal en lugar de ser diseñadas manualmente. Además, una red neuronal profunda también puede extraer una representación de alto nivel, lo que la hace más adecuada para tareas complejas de reconocimiento de actividades que requieren de contexto. Cuando se enfrentan a una gran cantidad de datos sin etiquetar, los modelos generativos profundos pueden hacer uso de los datos sin etiquetar para realizar el entrenamiento (Wang et al., 2019).

Aunque los modelos de aprendizaje profundo han logrado resultados notables en la visión artificial, el procesamiento del lenguaje natural y el reconocimiento del habla, no se han explotado completamente en el campo del HAR (Hsu et al., 2017).

En el presente trabajo se propone la aplicación de determinadas técnicas de aprendizaje profundo para intentar solucionar el problema del reconocimiento de la actividad humana. En concreto, se ha optado por la utilización de redes neuronales profundas. Existen dos enfoques principales, apropiados para la clasificación de series de datos temporales como es el caso del HAR: las redes neuronales recurrentes y las redes neuronales convolucionales.

La propuesta consiste en el diseño, implementación y evaluación de cuatro redes neuronales profundas con distintas arquitecturas, capaces de resolver el problema del HAR basado en sensores a partir de un conjunto de datos público que recoja información sobre el reconocimiento de actividades. Los modelos de red que se proponen son los siguientes:

- Red neuronal profunda completamente conectada.
- Red neuronal recurrente.
- Red neuronal convolucional seguida de una red neuronal recurrente.
- Red neuronal que combina elementos recurrentes y elementos convolucionales.

1.3 Estructura de la memoria

Después de la introducción, el capítulo 2 describe el estado del arte en cuanto al problema del reconocimiento de la actividad humana y las técnicas que se utilizan para resolverlo. Se contextualizan tanto los métodos basados en aprendizaje automático como en aprendizaje profundo, mostrando una visión general de las líneas de trabajo actuales. Además, se realiza una investigación sobre los principales estudios existentes que han utilizado el mismo conjunto de datos que el presente trabajo.

Una vez realizado el estudio del estado del arte, en el capítulo 3 se establece el objetivo que pretende lograr la contribución propuesta en este trabajo. Este objetivo principal se descompone en un conjunto de objetivos específicos, alcanzables y analizables por separado.

También en el capítulo 3 se define la metodología de trabajo empleada para conseguir los objetivos. Se introduce el modelo CRISP-DM (del inglés, CRoss-Industry Standard Process for Data Mining), considerado una referencia en la representación del ciclo de vida de proyectos basados en el análisis de datos.

El capítulo 4 presenta con detalle el trabajo realizado en cada una de las primeras fases especificadas en CRISP-DM: comprensión del negocio, comprensión de los datos, preparación de los datos y modelado. Se muestran las tareas realizadas y qué resultados se han obtenido.

En el capítulo 5 se describe la implementación de los modelos propuestos, su entrenamiento, optimización y evaluación. Para ello se hace uso de las herramientas y técnicas descritas en el capítulo anterior. De esta manera se completa la fase de evaluación de CRISP-DM, tras

comparar las diferentes arquitecturas propuestas y poder así decidir si se ha logrado el objetivo establecido.

Tras la presentación objetiva de los resultados, el capítulo 6 aporta una discusión sobre los mismos. En este capítulo se analiza la relevancia de los resultados del experimento, prestando especial atención en cómo afectan los hiperparámetros de los modelos a los resultados obtenidos.

El capítulo 7 presenta el resumen del trabajo. Este último capítulo expone las conclusiones a las que se ha llegado, relacionándose los resultados obtenidos con los objetivos que se han planteado. La sección final de este capítulo trata las posibles líneas de trabajo futuro.

2. Contexto y estado del arte

Este capítulo ofrece una visión general del estado del arte, contextualizando el problema del reconocimiento de la actividad humana y las técnicas que se utilizan para resolverlo. Se describen tanto los métodos basados en aprendizaje automático como en aprendizaje profundo, mostrando una visión global de las líneas de trabajo actuales. También se realiza una investigación sobre los principales estudios existentes que han utilizado como base el mismo conjunto de datos que el presente trabajo.

2.1. Reconocimiento de las actividades humanas

El reconocimiento de la actividad humana (en inglés Human Activity Recognition, HAR) es una tecnología que se engloba dentro del ámbito de la interacción persona-computadora (Y. Zhang et al., 2018). Como indican Wang, Chen, Hao, Peng y Hu (2019), empezó a aplicarse con éxito en análisis comportamental, videovigilancia, análisis de la marcha y reconocimiento de gestos.

El reconocimiento de la actividad humana se puede dividir en dos categorías, atendiendo a qué método se utiliza para obtener los datos que representan el movimiento (Y. Zhang et al., 2018).

- HAR basado en visión artificial: Se analizan imágenes estáticas o videos capturados por una cámara, que registran los movimientos realizados por el sujeto que se está estudiando.

- HAR basado en sensores: Los datos del movimiento se obtienen de sensores como acelerómetros o giroscopios, normalmente colocados en el cuerpo del sujeto que está bajo estudio.

Como se ha introducido ya en el capítulo 1, los sensores tienen una serie de ventajas que han provocado que el HAR basado en visión artificial quede relegado a un segundo plano. Resumiendo, el uso de sensores inerciales colocados sobre el cuerpo reduce las limitaciones ambientales de las cámaras de video, permite adquirir las señales de manera más precisa, y aumenta el nivel de privacidad y especificidad de los datos, al obtener información exclusivamente de un único sujeto.

Aunque el reconocimiento de actividades humanas comparte muchos desafíos metodológicos con otros campos (como la visión artificial, el procesamiento del lenguaje natural o el

reconocimiento del habla) se enfrenta a una serie de desafíos específicos. La visión artificial y el reconocimiento de voz tienen definiciones claras de los problemas, como detectar un objeto en una imagen o detectar una palabra en una oración. Por el contrario, el HAR ofrece más grados de libertad en términos de diseño e implementación del sistema (Bulling et al., 2014).

Primero, no existe una forma estándar para definir las actividades humanas que permita formular el problema de una manera clara y común: ¿qué actividad debe reconocerse? ¿cómo se caracteriza una actividad en concreto?

En segundo lugar, la actividad humana es muy diversa y, por lo tanto, su reconocimiento requiere una selección cuidadosa de los sensores encargados de realizar las mediciones.

2.2. Aprendizaje automático y HAR

El reconocimiento de la actividad humana se ha tratado habitualmente como un problema de reconocimiento de patrones. Los avances logrados con esta estrategia se deben principalmente a la utilización de algoritmos de aprendizaje automático como árboles de decisión, máquinas de vector de soporte, clasificadores bayesianos o modelos ocultos de Márkov (Lara & Labrador, 2012).

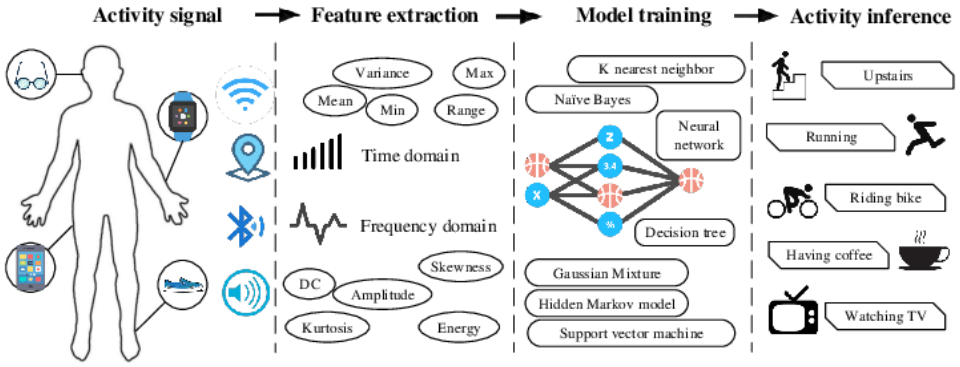

Figura 1. Reconocimiento de la actividad humana usando reconocimiento de patrones

Fuente: (Wang et al., 2019)

La figura 1 muestra el diagrama de flujo de cómo se aborda el HAR usando el reconocimiento de patrones. El primer paso es obtener los datos de actividad sin procesar, a partir de los distintos sensores. A continuación, se extraen manualmente las características o *features* de esas lecturas, utilizando el conocimiento y la experiencia humana. Por último, las

características se utilizan como entrada para entrenar un modelo de reconocimiento de patrones y poder utilizarlo para inferir actividades en aplicaciones reales (Wang et al., 2019).

Pero este tipo de soluciones basadas en el reconocimiento de patrones, como se ha expuesto en el capítulo 1, tienen una serie de desventajas (Wang et al., 2019).

- Requieren de una extracción heurística y manual de las características.
- La experiencia humana necesaria a la hora de extraer características, hace que utilizando estas técnicas sólo se puedan aprender *features* muy superficiales.
- Las técnicas de reconocimiento de patrones suelen necesitar una gran cantidad de datos correctamente etiquetados para entrenar el modelo.
- La mayoría de los modelos basados en el reconocimiento de patrones están pensados para aprender de datos estáticos.

2.3. Aprendizaje profundo y HAR

El aprendizaje profundo ayuda a mitigar las limitaciones de las propuestas basadas en el reconocimiento de patrones. La figura 2 muestra cómo funciona el aprendizaje profundo para el HAR, empleando diferentes tipos de redes neuronales.

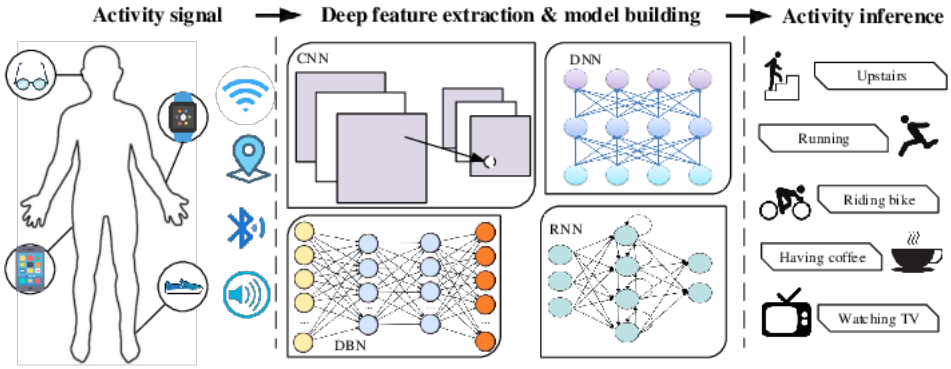

Figura 2. Reconocimiento de la actividad humana usando aprendizaje profundo

Fuente: (Wang et al., 2019)

En comparación con la figura 1, los procedimientos de extracción de características y construcción de modelos se realizan simultáneamente en los modelos de aprendizaje

profundo. Las características se pueden aprender automáticamente a través de la red en lugar de ser diseñadas manualmente. Además, una red neuronal profunda también puede extraer una representación de alto nivel, lo que la hace más adecuada para tareas complejas de reconocimiento de actividades que requieren de contexto. Cuando se enfrentan a una gran cantidad de datos sin etiquetar, los modelos generativos profundos pueden hacer uso de estos datos para realizar el entrenamiento (Wang et al., 2019).

2.3.1. Redes neuronales profundas

Las redes neuronales profundas (en inglés Deep Neural Networks, DNNs) se pueden considerar una evolución de las redes neuronales artificiales (en inglés, Artificial Neural Networks, ANNs). Las ANNs suelen estar formadas por pocas capas ocultas, por lo que se hace referencia a ellas como redes superficiales o poco profundas. Las DNNs tienen más capas ocultas, por lo tanto se consideran más profundas, siendo capaces de encontrar patrones más complejos en los datos de entrada.

Los primeros trabajos en los que se utilizan DNNs para solucionar el problema del HAR, lo hacen con redes poco profundas. En el trabajo de (Vepakomma et al., 2015), primero extraen manualmente las características a partir de los datos de los sensores, para posteriormente alimentar un modelo de red neuronal profunda. En (Walse et al., 2016) se procede de la misma manera, pero realizando un análisis de los componentes principales antes de usar la red neuronal. En esos trabajos, la DNN sólo sirve como modelo de clasificación tras una extracción de características manual, por lo tanto, no generalizan muy bien.

En el trabajo de (Hammerla et al., 2016) ya utilizan una DNN con 5 capas ocultas para realizar tanto el aprendizaje automático de las características como la clasificación. De esta manera, mejora el rendimiento y se pueden reconocer actividades más complejas.

2.3.2. Redes neuronales recurrentes

Las redes neuronales recurrentes (en inglés, Recurrent Neural Networks, RNNs) utilizan correlaciones temporales entre sus neuronas. Se usan ampliamente en el reconocimiento del habla y el procesamiento del lenguaje natural.

Hay pocos trabajos que utilicen RNNs aplicadas al problema del reconocimiento de la actividad humana. El enfoque principal de estas líneas de investigación se centran en entornos con recursos limitados que al mismo tiempo quieren obtener un buen rendimiento (Wang et al., 2019).

(Hammerla et al., 2016), (Inoue et al., 2018), (Edel & Köppe, 2016) y (Guan & Plötz, 2017) han propuesto redes recurrentes en el campo del HAR.

2.3.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (en inglés, Convolutional Neural Networks, CNNs) son especialmente idóneas para extraer características de las señales temporales de datos, y han logrado resultados prometedores en clasificación de imágenes, reconocimiento de voz y análisis de texto.

Cuando se aplican a la clasificación de series temporales, como es el caso del HAR, las CNNs tienen dos ventajas sobre otros modelos: dependencia local e invariancia de escala (Wang et al., 2019). La dependencia local significa que las señales cercanas en el tiempo probablemente estén correlacionadas (y por lo tanto pertenezcan a la misma actividad), mientras que la invariancia de escala se refiere a la diferencia de escala para diferentes ritmos o frecuencias (a las que se puede realizar una misma actividad).

Debido a la efectividad de las CNNs, la mayoría de los trabajos se centran en esta área. Por ejemplo tenemos los trabajos de (Zheng et al., 2014), (Zebin et al., 2016), (Q. Yang, 2009) o (Hannink et al., 2016).

2.3.4. Modelos híbridos

Los modelos híbridos se obtienen a partir de la combinación de varios modelos profundos. Un modelo híbrido emergente es la combinación de CNNs y RNNs (Wang et al., 2019).

En (Ordóñez & Roggen, 2016) y (Yao et al., 2017) se proporcionan ejemplos de cómo combinar CNNs y RNNs, demostrándose su buen rendimiento. La razón es que las CNNs pueden capturar mejor la relación espacial, mientras que las RNNs pueden hacer uso de la relación temporal. Esta combinación mejora la capacidad de reconocer actividades que tienen distribuciones de señal y lapsos de tiempo variados (Wang et al., 2019).

2.4. Conjunto de datos REALDISP

Este conjunto de datos se recopiló con el propósito inicial de investigar cómo afecta el desplazamiento de los sensores al proceso de reconocimiento de la actividad humana. El nombre de REALDISP proviene de *REAListic sensor DISPlacement*. Pero los datos se prestan perfectamente para ser utilizados en la evaluación comparativa de técnicas de reconocimiento

de la actividad en condiciones ideales, es decir, sin desplazamiento en los sensores (Banos, s. f.).

Existen varios trabajos realizados partiendo de este conjunto de datos. Uno de ellos es de los propios creadores de REALDISP, en el que se utilizan algoritmos de aprendizaje automático para estudiar el efecto que tiene el desplazamiento de los sensores a la hora de realizar el reconocimiento de las actividades (Baños et al., 2012). Empleando el algoritmo del centro de clase más cercano (en inglés Nearest Class Center, NCC), el método de los k vecinos más cercanos (en inglés K-Nearest Neighbors, KNN) y árboles de decisión (en inglés Decision Trees, DTs) se realiza la clasificación en cada uno de los escenarios de desplazamiento. En la figura 3 se muestran los resultados que obtienen para cada técnica y escenario.

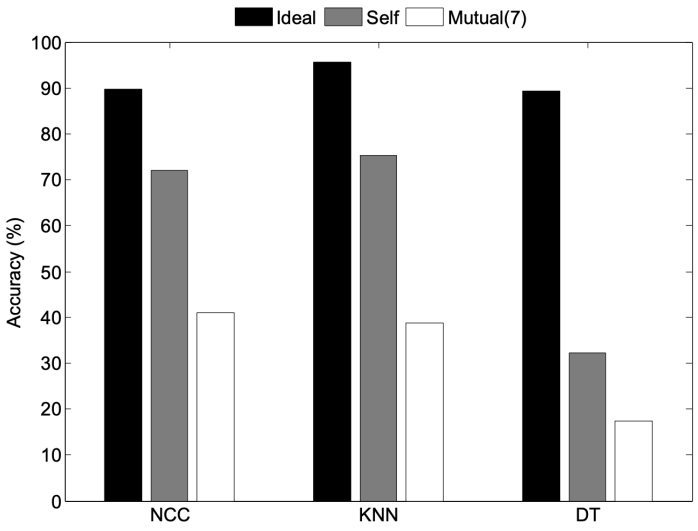

Figura 3. Reconocimiento de actividades en REALDISP usando NCC, KNN y DT

Fuente: (Baños et al., 2012)

Posteriormente, el mismo equipo realiza otro trabajo (Banos, Toth, et al., 2014) sobre REALDISP. También utilizan algoritmos de aprendizaje automático para hacer el reconocimiento, en esta ocasión el método de los k vecinos más cercanos (en inglés K-Nearest Neighbors, KNN), árboles de decisión (en inglés Decision Trees, DTs) y clasificadores bayesianos ingenuos (en inglés Naive Bayes, NB). Se diferencian tres marcos de trabajo, uno en el que se utilizan sólo 10 actividades, otro en el que se emplean 20 y un último en el que se considera el total de las 33 actividades. En la figura 4 se presenta uno de los resultados obtenidos, en concreto el alcanzado utilizando un único sensor de movimiento.

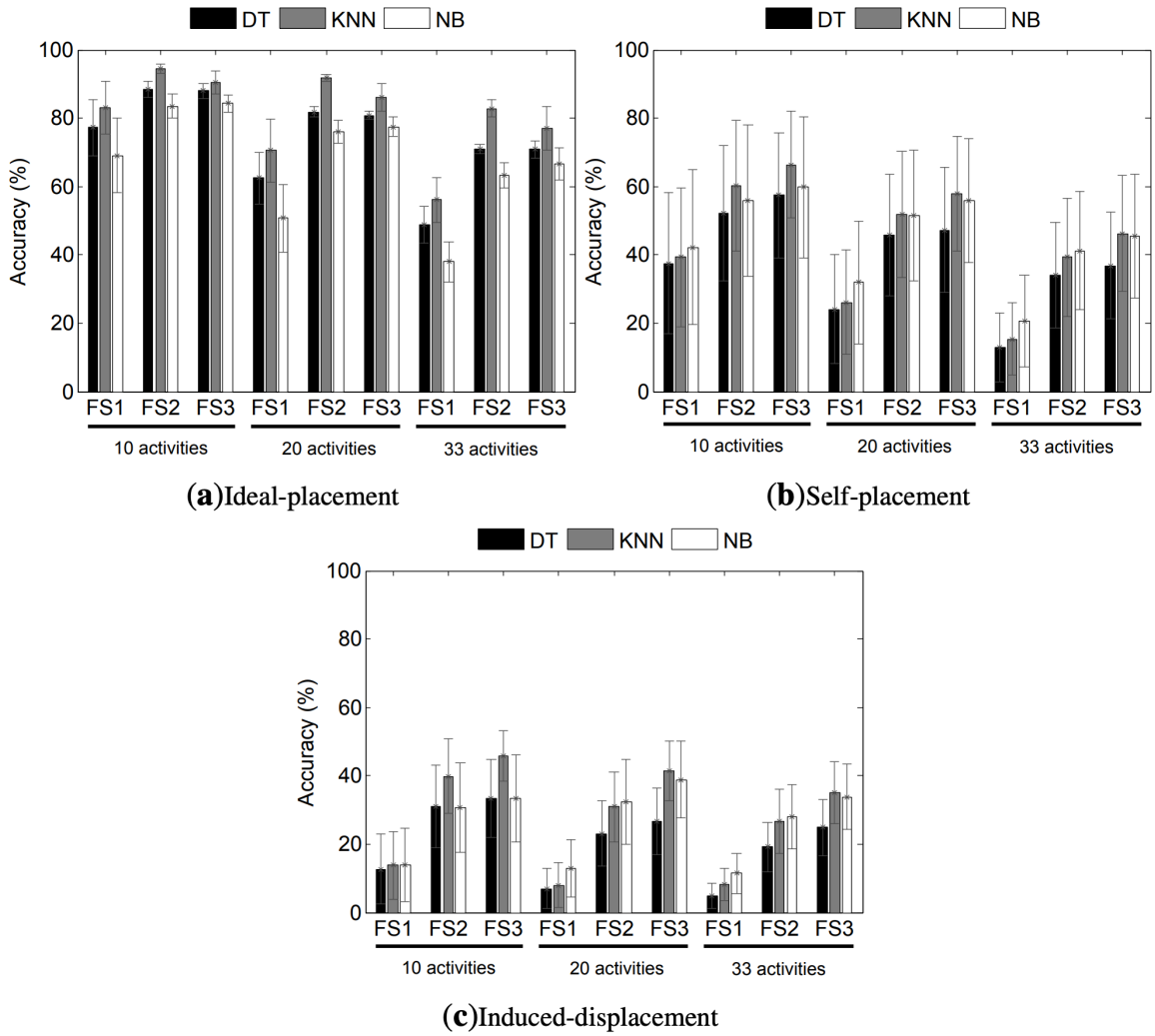

Figura 4. Reconocimiento de actividades en REALDISP usando DT, KNN y NB

Fuente: (Banos, Toth, et al., 2014)

Sobre REALDISP, también se puede destacar el trabajo de (Zhu et al., 2017), donde se comparan los resultados de (Banos, Toth, et al., 2014) con los obtenidos aplicando la técnica de aprendizaje automático denominada bosques aleatorios (en inglés, Random Forest, RF). Este trabajo se centra en la importancia de la extracción de las características. Utiliza *features* en el dominio del tiempo, extraídas de las señales de los sensores. Pero también en el dominio de la frecuencia, a partir de la transformada rápida de Fourier de las señales en el dominio del tiempo.

Por último, hay que nombrar el trabajo de (San et al., 2017), en el que ya se aplican técnicas de aprendizaje profundo sobre el conjunto de datos REALDISP. Como se puede ver en la

figura 5, en este trabajo se propone una red neuronal convolucional para resolver el problema del reconocimiento de las actividades. Se llega a la conclusión de que mejora en gran medida respecto a los métodos basados en aprendizaje automático, ya que la red convolucional es capaz de realizar una extracción de las características más eficiente que las efectuadas de forma manual. Adicionalmente, se tratan sucintamente las redes neuronales recurrentes y se indica el potencial que tienen para resolver el problema del HAR.

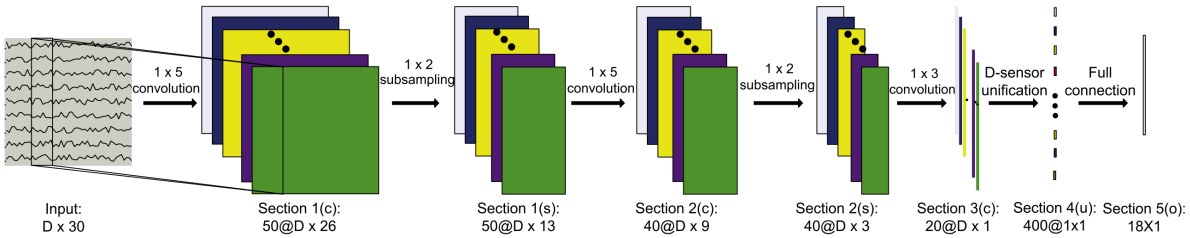

Figura 5. Modelo de red convolucional propuesto en *(San et al., 2017)*

Fuente: (San et al., 2017)

2.5. Redes neuronales profundas

En esta sección se introducen las redes neuronales profundas. Esta información de contexto es importante a la hora de plantear la aportación descrita en los capítulos 4 y 5. Se exponen aspectos muy generales sobre las redes neuronales, como la arquitectura de red, la función de activación o el algoritmo de entrenamiento.

2.5.1. Arquitectura de red

El diseño de las capas de entrada y salida de una red neuronal suele ser sencillo, pero no ocurre lo mismo con la organización de sus capas ocultas. La arquitectura o topología de una red neuronal define la estructura interna de la misma, y está directamente relacionada con la complejidad de los patrones que es capaz de aprender (Nielsen, 2015).

Por ejemplo, en las llamadas redes neuronales prealimentadas (en inglés, feed-forward neural networks), la salida de una capa se utiliza directamente como entrada para la siguiente. Esto significa que no hay bucles en la red: la información siempre se transmite, nunca se retroalimenta. Sin embargo, existen otros modelos de redes neuronales en las que son

posibles los circuitos de retroalimentación, como en las redes neuronales recurrentes (Nielsen, 2015).

Por lo tanto, la arquitectura de una red determina, entre otras, las siguientes características:

- El número de capas.
- El número de neuronas por capa.
- La función de activación de las neuronas.
- La forma en la que están interconectadas las neuronas.

2.5.2. Función de activación

La función de activación establece cómo se procesa la información en las neuronas y cómo se propaga entre las capas. Es uno de los elementos más críticos a la hora de establecer la topología de una red neuronal, ya que afecta en gran medida al resultado de su entrenamiento (Nielsen, 2015).

La función de activación más utilizada históricamente es la de tipo sigmoide. Se define como:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Su comportamiento puede verse en la figura 6. Este tipo de unidad de activación comprime su entrada a un valor de salida entre 0 y 1. Los valores positivos grandes tienden a 1, mientras que los valores negativos grandes tienden a 0.

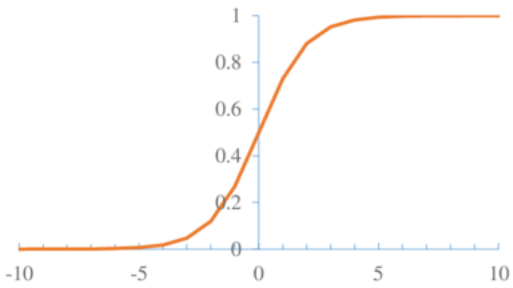

Figura 6. Función sigmoide

Fuente: (Jing Yang & Yang, 2018)

La función sigmoide fue la base de la mayoría de las redes neuronales, pero ha ido perdiendo popularidad. Cuando se utiliza en redes neuronales profundas, el entrenamiento se hace muy complicado debido al problema de la desaparición del gradiente.

Actualmente, la mayoría de las redes neuronales usan otro tipo de funciones de activación que permiten entrenar modelos cada vez más complejos, como la Rectified Linear Unit o ReLU. Su definición es la siguiente:

$$R(x) = max(0, x)$$

Su comportamiento puede verse en la figura 7. La función ReLU permite que pasen los valores positivos, sin cambiarlos, pero siempre devuelve 0 para valores negativos de entrada.

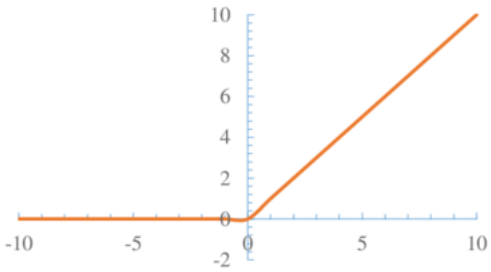

Figura 7. Función ReLU
Fuente: (Jing Yang & Yang, 2018)

2.5.3. Algoritmo de entrenamiento

La finalidad de los algoritmos de entrenamiento de redes neuronales es encontrar los valores adecuados para los parámetros de cada una de las neuronas. En definitiva, se trata de un problema de optimización.

El descenso de gradiente es una forma de minimizar una función objetivo caracterizada por los parámetros de un modelo, actualizando el valor de los parámetros en la dirección opuesta del gradiente de la función objetivo. La tasa de aprendizaje η determina la magnitud de los pasos que se realizan para alcanzar un mínimo (que puede ser local). De forma gráfica, se sigue la dirección de la pendiente de la superficie creada por la función objetivo, cuesta abajo, hasta llegar a un valle (Ruder, 2016).

El algoritmo *stochastic gradient descent* (SGD) aplica la técnica del descenso de gradiente, utilizando cada una de las muestras de entrenamiento para efectuar pequeños pasos en la dirección de máximo descenso, acercándose a un mínimo (de nuevo, es posible que sea local). Aunque cumple su cometido, SDG es un algoritmo con múltiples puntos débiles (Ruder, 2016).

Adaptative Moment Estimation (Adam) es un método que utiliza tasas de aprendizaje de forma adaptativa para cada parámetro de la función objetivo, combinando técnicas de otros algoritmos y métodos, como SDG con *momentum*, Adadelta y RMSprop (Kingma & Ba, 2014). Adam reduce la posibilidad de acabar en una posición intermedia, que no permita avanzar hacia un mínimo, ya sea local o global. Se dice que mientras SGD con *momentum* puede compararse con una bola que cae por una pendiente, Adam se comporta como una bola pesada con fricción (Heusel et al., 2017).

2.5.4. Técnicas de regularización

Dos de las causas más comunes de la baja precisión de un modelo predictivo son el subajuste (*underfitting*) y el sobreajuste (*overfitting*). Se puede estimar cómo de ajustado está un modelo a los datos de entrenamiento observando el error de predicción en las muestras de entrenamiento y prueba. La figura 8 muestra los tres posibles niveles de ajuste.

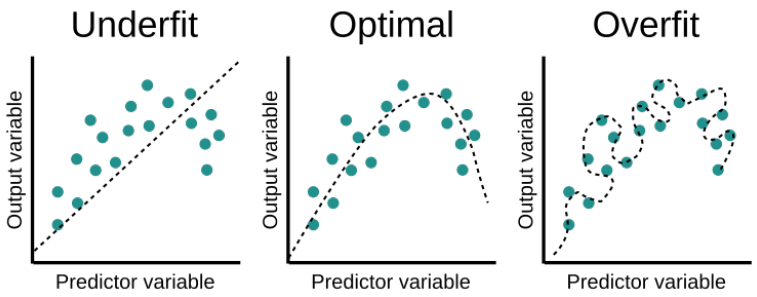

Figura 8. Nivel de ajuste de un modelo predictivo

Fuente: (Overfitting and Underfitting, s. f.)

Un modelo con *underfitting* tiene una tasa de error alta, tanto para los datos de entrenamiento como para los de prueba. Esto sucede cuando el modelo es demasiado sencillo y no es capaz de establecer una relación entre los valores de entrada y las variables objetivo.

Un modelo con *overfitting* tiene un error de predicción muy bajo en los datos de entrenamiento, pero un error de predicción muy alto en los datos de prueba. Se debe a que ha memorizado

el conjunto de datos de entrenamiento y no es capaz de generalizar lo aprendido a un conjunto de datos nuevo. Normalmente sucede con modelos muy complejos.

Ambos extremos deben ser evitados. Sin embargo, con el objetivo de encontrar el modelo de red neuronal con mayor nivel predictivo, lo más habitual es caer en el error de diseñar modelos excesivamente complejos, sobreajustados. El sobreajuste ocurre cuando el modelo aprende demasiado de los datos con los que se ha entrenado, perdiendo su aplicabilidad a cualquier otro conjunto de datos. Esto se debe a que las redes neuronales tienen un gran poder de representación. Este poder permite que una red pueda llegar a memorizar todos los detalles de los datos del conjunto de entrenamiento, perdiendo su capacidad de generalización (Hawkins, 2004).

Para intentar solucionar el problema del *overfitting*, se puede hacer uso de técnicas de regularización. Se trata de métodos que intentan minimizar al mismo tiempo la complejidad del modelo y la función de coste. Esto facilita la tarea de encontrar modelos más simples pero que generalizan mejor. Algunos de los procedimientos más utilizados son las siguientes:

- Early stopping: Consiste utilizar un conjunto de datos de validación, independientes de los de prueba, con los que se evalúa cada iteración en el entrenamiento del modelo. Cuando se observa que el error en el conjunto de validación aumenta en varias iteraciones consecutivas, se detiene el entrenamiento para evitar el sobreajuste.

- Dropout: Es una técnica reciente y con gran aceptación. Se basa en la idea de desactivar un número aleatorio de neuronas durante el entrenamiento. Esas neuronas tendrán un valor de salida igual a 0, dificultando la memorización de las características de las variables de entrada. Esto hace que la red tenga que aprender nuevos patrones, facilitando la regularización (Srivastava, 2013).

- Regularización L2: Se fundamenta en la inclusión de una ligera penalización en la función de coste, para impedir que el modelo aprenda excesivamente rápido los detalles de las variables de entrada. Es un método de regularización sencillo, ya que no modifica la topología de la red, pero muy efectivo.

2.5.5. Conjuntos de datos de entrenamiento, validación y pruebas

Para poder aplicar la técnica de regulación *early stopping* descrita en la sección 2.5.4, y evitar el sobreajuste en los modelos, el total de datos disponibles se divide en tres conjuntos: entrenamiento, validación y pruebas.

El conjunto de datos de entrenamiento se utiliza para alimentar el modelo de aprendizaje profundo, procediendo a la identificación de los patrones ocultos que relacionan las variables de entrada con la etiqueta asociada a los registros. El conjunto de validación sirve para poder comprobar que el modelo no se está sobreajustando a los datos de entrenamiento. Tras cada iteración se comprueba el error obtenido en este conjunto y, si comienza a aumentar, se detiene el proceso de aprendizaje. Una vez se considera acabado el entrenamiento del modelo, el conjunto de datos de prueba permite evaluar su comportamiento final. La figura 9 muestra gráficamente en qué consiste esta técnica.

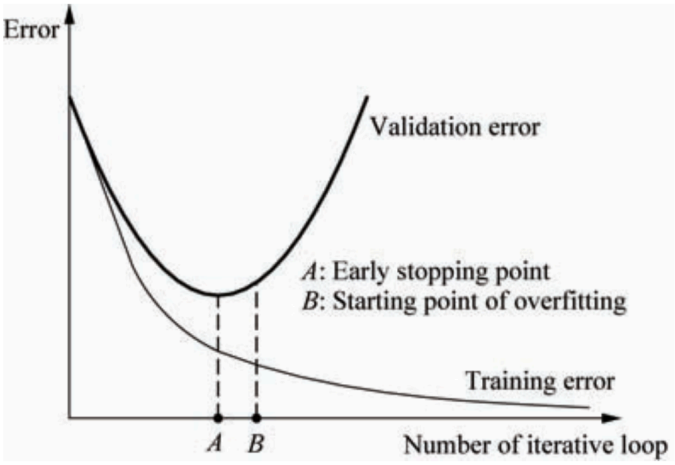

Figura 9. Técnica de *early stopping*
Fuente: (Yuan & Guangchen, 2011)

2.5.6. Métricas de evaluación

Existen múltiples métricas para evaluar la capacidad predictiva de los modelos de clasificación. Se suele comparar los aciertos y los errores para cada clase a predecir, sobre cada ejemplo del conjunto de pruebas. Así, se tienen en cuenta medidas como las siguientes:

- *Accuracy* o exactitud: También conocida como el ratio de éxito. Es el porcentaje de predicciones correctas.
- *Precision* o precisión: Indica la proporción de aciertos para una clase. Cuando el modelo predice una determinada clase, cuántas veces está en lo cierto.
- *Recall* o exhaustividad: Es el porcentaje de aciertos, para cada una de las posibles clases.

- *F1-score*: Es una medida que combina precisión y exhaustividad, utilizando la media armónica. Se usa con mucha frecuencia, ya que simplifica la evaluación de un modelo clasificador a una única métrica.

2.5.7. Herramientas para el modelado de redes neuronales

A continuación se especifica con más detalle el conjunto de tecnologías y herramientas empleadas para realizar el diseño, implementación y evaluación de las redes neuronales profundas propuestas en este trabajo:

- Python: Se trata de un lenguaje de programación con amplia aceptación en el mundo del análisis de datos y de la inteligencia artificial.
- TensorFlow: Es una biblioteca de código abierto creada por Google, centrada en tareas de aprendizaje automático. Es una biblioteca de bajo nivel muy potente, capaz de definir redes neuronales utilizando grafos de computación.
- Keras: Biblioteca de alto nivel, que puede funcionar por encima de TensorFlow, para definir redes neuronales de una forma más sencilla y modular.
- Google Colab: Entorno gratuito que se ejecuta en la nube y que soporta el uso de cuadernos compatibles con Jupyter. Este tipo de cuadernos permite alternar el uso de bloques de código fuente y bloques de texto, característica que los convierte en una opción muy interesante para integrar y compartir un desarrollo de software y su documentación asociada.

2.6. Técnica de la ventana deslizante

La segmentación de señales es uno de los procesos más importantes a la hora de intentar resolver el problema del reconocimiento de la actividad humana. Uno de los enfoques que se usan normalmente para realizar la segmentación es la técnica de la ventana deslizante.

Esta técnica consiste en dividir los datos de la señal de entrada en tramos de longitud fija, que pueden contener varios segundos de observación. A la hora de aplicar este enfoque se deben seleccionar dos parámetros: el tamaño de la ventana y el desplazamiento (Laguna et al., 2011).

La segmentación mediante ventana deslizante puede resumirse en el siguiente proceso iterativo, siendo *s* el tamaño de la ventana y *d* el desplazamiento:

1. Se toman *s* muestras, registradas consecutivamente y etiquetadas con la actividad correspondiente. Como etiqueta del segmento, se selecciona la que más se repita en ese tramo.

2. Se realiza un desplazamiento de *d* muestras, avanzando en la señal.

3. Se vuelve al primer punto para definir un nuevo segmento.

La figura 10 ilustra esta técnica de segmentación, mediante un ejemplo para *s* = *3* y *d* = *2*.

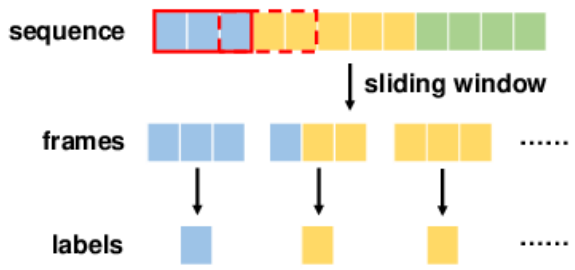

Figura 10. Técnica de la ventana deslizante

Fuente: (W. Zhang et al., 2019)

Dependiendo del tamaño de la ventana y del desplazamiento, se presentan dos posibles escenarios:

- Sin solapamiento: Sucede cuando el desplazamiento es igual al tamaño de la ventana, es decir, cuando *s* = *d*. Se toman las muestras de *s* en *s*, dividiendo así la señal en segmentos de tamaño *s*.

- Con solapamiento: El solapamiento aparece cuando el desplazamiento es menor que el tamaño de la ventana, cuando se cumple que *0* < *d* < *s*. La figura 10 muestra un ejemplo de segmentación con un 33% de solapamiento, puesto que cada segmento contiene un tercio de la información del segmento anterior.

3. Objetivos y metodología de trabajo

Tras haber realizado el estudio del estado del arte, es necesario establecer los objetivos que pretende lograr la contribución propuesta en este trabajo.

Además, en este capítulo se define la metodología de trabajo empleada para conseguir los objetivos más específicos. Alcanzar el objetivo general será una consecuencia directa de conseguir los objetivos parciales.

3.1. Objetivo general

La intención final es aplicar técnicas de aprendizaje profundo para resolver el problema del reconocimiento de la actividad humana (en inglés Human Activity Recognition, HAR) basada en sensores inerciales vestibles.

Este objetivo debe ser específico, medible y alcanzable. Por lo tanto, las técnicas de aprendizaje profundo que se proponen incluyen un conjunto de cuatro redes neuronales profundas con distintas arquitecturas internas, que pueden ser aplicadas y evaluadas de manera independiente.

3.2. Objetivos específicos

El objetivo principal de este trabajo se ha descompuesto en una serie de objetivos más específicos y analizables por separado. Son los siguientes:

- Explorar los distintos repositorios de datos públicos con información relacionada con el HAR.
- Seleccionar los métodos de aprendizaje profundo que se quieren aplicar a la resolución del problema.
- Diseñar e implementar redes neuronales con diferentes topologías, para ser entrenadas utilizando los datos etiquetados disponibles.
- Evaluar y comparar las distintas soluciones propuestas.

3.2.1. Exploración de los conjuntos de datos disponibles

La recolección de datos propios es una opción que se ha descartado, ya que no se dispone de los recursos necesarios y su recopilación excede el alcance de este trabajo. Pero como se ha visto en el estudio sobre el estado del arte, existen repositorios de datos públicos con información sobre el reconocimiento de actividades.

En este trabajo se justifica el uso del conjunto de datos REALDISP. Como se ha mencionado en la sección 2.4, originalmente estaba enfocado al estudio del efecto que tiene el desplazamiento de los sensores en el reconocimiento de las actividades. Pero también ha sido utilizado como referencia para la aplicación de técnicas de reconocimiento, usando la información relativa a las condiciones ideales.

3.2.2. Selección de los métodos de aprendizaje profundo

En la sección 2.5 se han introducido las redes neuronales. Existen otras arquitecturas en el campo del aprendizaje profundo, como las redes de creencia profunda (en inglés Deep Belief Network, DBN) o el aprendizaje profundo por refuerzo (en inglés Deep Reinforcement Learning, DRL).

En este trabajo, para acotar su alcance, los métodos propuestos se centran en el ámbito de las redes neuronales profundas, específicamente en las redes neuronales recurrentes (en inglés Recurrent Neural Networks, RNNs) y las redes neuronales convolucionales (en inglés Convolutional Neural Networks, CNNs).

3.2.3. Diseño e implementación de las redes neuronales

La topología o arquitectura de una red neuronal establece su organización interna y define su comportamiento.

Como se explica a lo largo de este trabajo, se han propuesto cuatro arquitecturas de red neuronal diferentes, que solucionan en mayor o menor medida el problema del HAR:

- Red neuronal profunda completamente conectada.
- Red neuronal recurrente.
- Red neuronal convolucional seguida de una red neuronal recurrente.
- Red neuronal que combina elementos recurrentes y elementos convolucionales.

3.2.4. Evaluación y comparación de los resultados

Los modelos de red implementados son evaluados, para obtener su precisión a la hora de reconocer actividades. Se calculan las principales métricas empleadas en la evaluación de los algoritmos de clasificación en aprendizaje supervisado, que a su vez sirven para realizar una comparación entre las cuatro arquitecturas propuestas.

3.3. Metodología del trabajo

Para poder alcanzar los objetivos específicos de una manera ordenada, siguiendo el método científico, se ha decidido utilizar CRISP-DM.

CRISP-DM (del inglés CRoss-Industry Standard Process for Data Mining) se considera un modelo de referencia para representar el ciclo de vida de proyectos basados en el análisis de datos. Es un marco de trabajo agnóstico, independiente de la industria, de las herramientas y de las aplicaciones (Shearer, 2000). Surge en el ámbito de la minería de datos, pero su flexibilidad hace que sea fácilmente adaptable a cualquier proyecto en el que se gestionen grandes cantidades de datos, como es el caso del aprendizaje profundo.

El modelo CRISP-DM establece en cuántas fases debe descomponerse el desarrollo de un proyecto, especificando incluso qué tareas y qué resultados se espera de cada una de esas fases. En la figura 11 se muestra el ciclo de vida de los proyectos centrados en datos.

Las seis fases estándar son:

- Comprensión del negocio.
- Comprensión de los datos.
- Preparación de los datos.
- Modelado.
- Evaluación.
- Despliegue.

Las fases no siguen una determinada secuencia de forma estricta. Las flechas de transición entre fases que se ven en la figura 11 indican las dependencias más importantes, pero no son las únicas. Las relaciones que se establecen realmente entre cada una de las fases del modelo, dependen principalmente del proyecto concreto que se esté realizando (Wirth & Hipp, 2000).

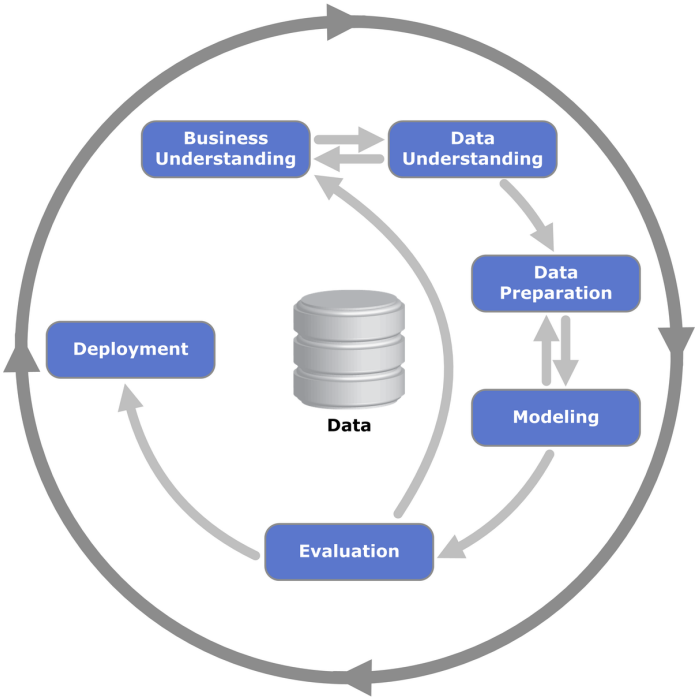

Figura 11. Fases del modelo CRISP-DM

Fuente: Creado por Kenneth Jensen, basado en (IBM, 2016)

El círculo exterior de la figura 11, alude a la naturaleza cíclica intrínseca al proceso de análisis de los datos. Todo lo que se aprende durante el proceso, unido a lo observado tras el despliegue de la solución, sirve como punto de partida para una nueva iteración en el proceso, incidiendo en cuestiones más específicas (Shearer, 2000).

A continuación, se describen sucintamente cada una de las fases y se contextualizan en el marco de los objetivos de este trabajo.

3.3.1. Comprensión del negocio

Es la fase inicial, orientada a comprender los objetivos del proyecto desde un punto de vista comercial o de producto. Este conocimiento es la base para establecer cuál es el problema a resolver y definir un plan preliminar para encontrar una solución (Wirth & Hipp, 2000).

3.3.2. Comprensión de los datos

En esta fase se realiza una recopilación inicial de los datos y se ejecutan las actividades necesarias para familiarizarse con los mismos. Esta primera toma de contacto ayuda a identificar problemas relacionados con la calidad de los datos, propiciando incluso los primeros descubrimientos (Shearer, 2000).

Existe un vínculo estrecho entre esta fase y la de comprensión del negocio. Para poder definir cuál es el problema y qué plan se va a seguir para solucionarlo, es necesario cierto nivel de comprensión de los datos disponibles (Wirth & Hipp, 2000).

3.3.3. Preparación de los datos

Durante la fase de preparación de los datos, se llevan a cabo todas las actividades que permiten construir el conjunto de datos final, a partir de los datos iniciales sin procesar. El resultado de estas acciones alimentará, en última instancia, a las herramientas de modelado. Estas actividades incluyen la selección de los registros y atributos a utilizar, la limpieza de los datos, así como posibles transformaciones para facilitar el trabajo de las herramientas de modelado (Wirth & Hipp, 2000).

3.3.4. Modelado

Como regla general, existen múltiples técnicas de modelado aplicables a un mismo problema de análisis de datos. En esta fase se seleccionan y aplican las técnicas que se consideran más adecuadas, y se ajustan sus parámetros de funcionamiento para intentar que sean los óptimos (Shearer, 2000).

También aquí existe una relación muy importante entre esta fase y la de preparación de los datos, ya que algunas técnicas de modelado requieren que los datos tengan un formato específico (Wirth & Hipp, 2000).

3.3.5. Evaluación

Tras la creación de uno o varios modelos capaces de ofrecer una solución, es importante evaluarlos en profundidad y revisar los pasos ejecutados para la construcción de los mismos. La intención final de esta fase es decidir si se ha conseguido obtener un modelo capaz de lograr los objetivos comerciales establecidos (Wirth & Hipp, 2000).

3.3.6. Despliegue

La creación del modelo no siempre es el paso final. El conocimiento obtenido debe ser organizado y presentado para facilitar su explotación. Como en muchas ocasiones es el propio cliente quien lleva a cabo las acciones de despliegue, y no un analista de datos, es importante comprender qué tareas deben llevarse a cabo para poder utilizar los modelos implementados (Wirth & Hipp, 2000).

4. Aprendizaje profundo aplicado al reconocimiento de la actividad humana

En el capítulo 3 se han definido los objetivos, tanto el principal como los específicos. Asimismo, se ha establecido CRISP-DM como metodología de trabajo, por su extendido uso en proyectos relacionados con el análisis de datos.

A lo largo de este capítulo se presenta con detalle el trabajo elaborado en cada una de las fases especificadas en CRISP-DM. Se muestran las tareas realizadas y qué resultados se han obtenido.

4.1. Comprensión del negocio

Conocer en qué consiste el problema del HAR es la base para poder encontrar una solución al mismo. En esta sección se expone el desafío que supone reconocer la actividad humana basándose en la información recogida por sensores vestibles. Además, se justica la selección de un conjunto de datos específico como es REALDISP, ya que como se ha comentado en la sección 3.3.2, existe un vínculo muy importante entre la comprensión del negocio y la comprensión de los datos.

4.1.1. El problema del reconocimiento de la actividad humana

El reconocimiento de la actividad humana es el campo de estudio relacionado con la identificación del movimiento o la acción específica que está realizando una persona.

Cuando se habla de actividades, se suele hacer referencia conjuntos de movimientos o acciones relacionadas, que tienen sentido en un determinado contexto:

- Actividades diarias, como caminar, hablar o sentarse.
- Actividades realizadas en la cocina, como cortar, pelar o hervir.
- Actividades deportivas, como correr, nadar o saltar.

Una de las formas de intentar llevar a cabo el reconocimiento de la actividad que está realizando una persona, implica predecir el tipo de movimiento que está efectuando basándose en los datos que proporciona un conjunto de sensores. Los datos del sensor pueden registrarse de forma remota, por métodos inalámbricos como Bluetooth o tecnología

wifi. Alternativamente, la información de los sensores se puede almacenar directamente en un dispositivo, como sucede por ejemplo con los teléfonos inteligentes o las pulseras cuantificadoras.

Puede decirse que el reconocimiento de la actividad basada en sensores busca un conocimiento a alto nivel sobre las actividades, a partir de múltiples lecturas de bajo nivel realizadas por sensores (Wang et al., 2019).

Por lo tanto, reconocer la actividad se convierte en un problema de clasificación a partir de series temporales de datos. Una serie temporal es una secuencia observaciones de una variable, medidas en determinados instantes de tiempo y ordenadas cronológicamente.

Para poder realizar la extracción de características a partir de los datos en bruto de los sensores, es necesario aplicar métodos y técnicas provenientes del campo del procesamiento de la señal. Es importante tener un conocimiento profundo del dominio ya que, aunque las señales recopiladas por los sensores son series temporales, son diferentes de otras series de datos relacionadas con el tiempo, como las señales de voz. Sólo ciertas partes del flujo continuo de la señal son relevantes para el HAR; la parte de la señal que carece de interés es la que corresponde a lo que se considera la ausencia de actividad. También hay que tener en cuenta que no todos los sensores son relevantes para distinguir cada actividad, dependiendo de su posición y configuración. Además, considerando cómo se realiza la actividad humana en la realidad, se observa que cada actividad es una combinación de varios movimientos continuos básicos. Por lo general, una actividad humana puede durar varios segundos en la práctica, pero involucrar sólo unos pocos movimientos básicos. Ejemplos que evidencian esta particularidad pueden ser saltar a la comba o hacer sentadillas (San et al., 2017).

En definitiva, se trata de un problema complejo, en el que no existe una forma obvia de relacionar los datos registrados por los sensores, con la actividad humana que se está realizando.

4.1.2. Selección del conjunto de datos

La comprensión del negocio está íntimamente ligada a la comprensión de los datos. Pero antes de familiarizarse con los datos hay que llevar a cabo el proceso de adquisición de los mismos.

A la hora de obtener los datos necesarios para aplicar técnicas de aprendizaje profundo al problema del HAR, existen principalmente dos esquemas de actuación: realizar una adquisición propia de los datos necesarios, o usar un conjunto de datos público (Wang et al., 2019).

- Recolección de datos propios: Muchos trabajos efectúan su propia recolección de datos. El principal problema de esta forma de actuar, es el gran esfuerzo que se necesita realizar para registrar todos los datos. Además, las tareas de procesamiento de los datos obtenidos son tediosas y requieren de mucho tiempo.

- Conjuntos de datos públicos: La mayoría de los investigadores adoptan el uso de alguno de los múltiples conjuntos de datos públicos existentes.

En esta última dirección, la tabla 1 resume las principales características de varios conjuntos de datos abiertos y accesibles públicamente, que pueden utilizarse como referencia para evaluar las soluciones propuestas al problema del HAR.

OPPORTUNITY (Ordóñez & Roggen, 2016) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio, magnetómetro, sensor en objeto, sensor de ambiente			
4 sujetos	32 Hz	16 actividades	701.366 muestras
Skoda Checkpoint (Plötz et al., 2011) Reconocimiento de actividades realizadas en fábricas Acelerómetro			
1 sujeto	96 Hz	10 actividades	22.000 muestras
UCI Smartphone (Almaslukh et al., 2017) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio			
30 sujetos	50 Hz	6 actividades	10.299 muestras
PAMAP2 (Zheng et al., 2014) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio, magnetómetro			
9 sujetos	100 Hz	18 actividades	2.844.868 muestras
USC-HAD (Jiang & Yin, 2015) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio			
14 sujetos	100 Hz	12 actividades	2.520.000 muestras
WISDM (Alsheikh et al., 2016) Reconocimiento de actividades de la vida diaria Acelerómetro			
29 sujetos	20 Hz	6 actividades	1.098.207 muestras
DSADS (L. Zhang et al., 2015) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio, magnetómetro			
8 sujetos	25 Hz	19 actividades	1.140.000 muestras

Ambient kitchen (Plötz et al., 2011) Reconocimiento de actividades relacionadas con la preparación de comida Sensor en objeto			
20 sujetos	40 Hz	2 actividades	55.000 muestras
Darmstadt Daily Routines (Plötz et al., 2011) Reconocimiento de actividades de la vida diaria Acelerómetro			
1 sujeto	100 Hz	35 actividades	24.000 muestras
Actitracker (Zheng et al., 2014) Reconocimiento de actividades de la vida diaria Acelerómetro			
36 sujetos	20 Hz	6 actividades	2.980.765 muestras
SHO (Jiang & Yin, 2015) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio, magnetómetro			
10 sujetos	50 Hz	7 actividades	630.000 muestras
BIDMC (Zheng et al., 2014) Reconocimiento de la insuficiencia cardíaca Electrocardiograma			
15 sujetos	125 Hz	2 actividades	> 20.000 muestras
MHEALTH (Banos et al., 2015) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio, magnetómetro			
10 sujetos	50 Hz	12 actividades	16.740 muestras
Daphnet Gait (Hammerla et al., 2016) Reconocimiento de la forma de andar Acelerómetro			
10 sujetos	64 Hz	2 actividades	1.917.887 muestras
ActiveMiles (Ravi et al., 2016) Reconocimiento de actividades de la vida diaria Acelerómetro			
10 sujetos	50-200 Hz	7 actividades	4.390.726 muestras
HASC (Hayashi et al., 2015) Reconocimiento de actividades de la vida diaria Acelerómetro			
1 sujeto	200 Hz	13 actividades	
PAF (Pourbabaee et al., 2018) Reconocimiento de la fibrilación auricular paroxística Electroencefalograma			
48 sujetos	128 Hz	2 actividades	230.400 muestras
ActRecTut (Jianbo Yang et al., 2015) Reconocimiento de gestos			

Acelerómetro, giroscopio			
2 sujetos	32 Hz	12 actividades	102.613 muestras
Heterogeneous (Yao et al., 2017) Reconocimiento de actividades de la vida diaria Acelerómetro, giroscopio			
9 sujetos	100-200 Hz	6 actividades	43.930.257 muestras

Tabla 1. Conjuntos de datos públicos relacionados con el HAR

Fuente: Elaboración propia basada en (Wang et al., 2019)

4.1.3. Conjunto de datos REALDISP

Para el problema del reconocimiento de actividades no existe un conjunto de datos estándar sobre el que trabajar, como sí sucede en otras áreas dentro del campo de la minería de datos. Esto supone un obstáculo, ya que cada grupo de investigación recopila sus propios datos, escoge los sujetos, establece el conjunto de actividades a reconocer y utiliza diferentes metodologías de evaluación.

En este trabajo, se ha decidido hacer uso del conjunto de datos REALDISP (UCI Machine Learning Repository: REALDISP Activity Recognition Dataset Data Set, s. f.). Los principales motivos de esta elección son el gran número de actividades físicas que contempla, la diversidad de información que proporcionan los sensores utilizados en la recopilación de datos, y el considerable número de sujetos participantes en el proceso.

En la tabla 2 se muestra información que permite comparar este conjunto de datos con los repositorios de acceso público listados en la sección 4.1.2.

REALDISP (UCI Machine Learning Repository: REALDISP Activity Recognition Dataset Data Set, s. f.) Reconocimiento de actividades fitness Acelerómetro, giroscopio, magnetómetro, orientación por cuaterniones			
17 sujetos	50 Hz	33 actividades	7.355.749 muestras

Tabla 2. Información básica sobre REALDISP

Fuente: Elaboración propia

Este conjunto de datos se recopiló con el propósito inicial de investigar cómo afecta el desplazamiento de los sensores al proceso de reconocimiento de la actividad humana. De

hecho, el nombre de REALDISP proviene de *REAListic sensor DISPlacement*. Pero los datos se prestan perfectamente para ser utilizados en la evaluación comparativa de técnicas de reconocimiento de la actividad en condiciones ideales, es decir, sin desplazamiento en los sensores (Banos, s. f.).

4.2. Comprensión de los datos

La exploración del conjunto de datos es fundamental para familiarizarse con ellos y empezar a identificar cómo pueden ser utilizados para conseguir el objetivo de negocio.

En esta sección se profundiza en la estructura y contenido del conjunto de datos REALDISP, centrándose en los datos registrados en el escenario de la colocación ideal de los sensores.

4.2.1. Sensores

En el estudio realizado para obtener los datos de REALDISP, se utilizaron 9 unidades de medición inercial de tipo *Xsens*, colocadas sobre el cuerpo del sujeto tal y como se muestra en la figura 12.

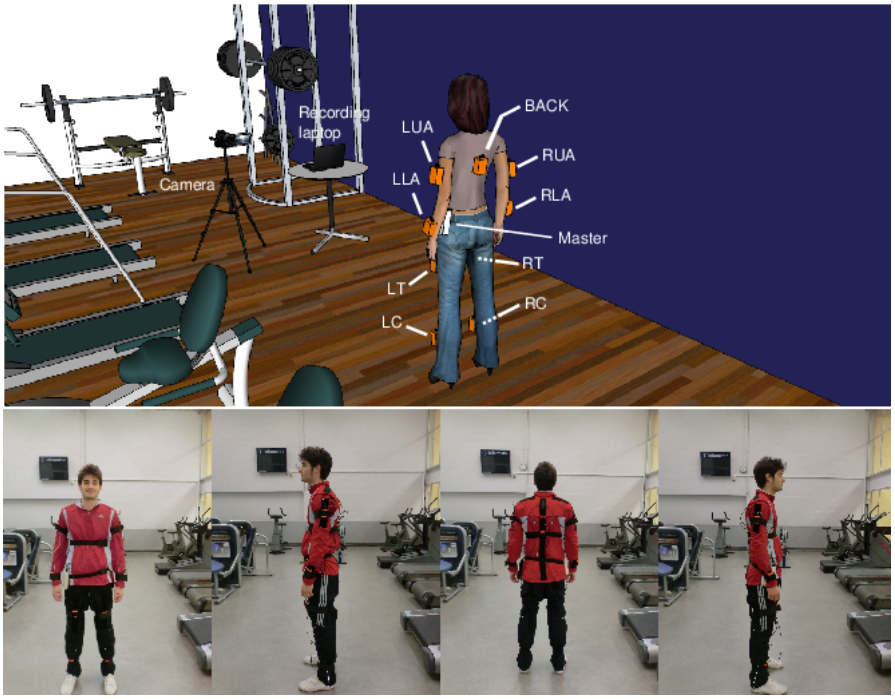

Figura 12. Colocación de las unidades Xsens

Fuente: (Banos, Toth, et al., 2014)

Cada sensor tiene asociado un código, utilizado posteriormente para su identificación a lo largo del estudio. La tabla 3 muestra los sensores, sus códigos y la parte del cuerpo en la que están situados.

Código	Posición
LC	Left calf (pantorrilla izquierda)
RC	Right calf (pantorrilla derecha)
LT	Left thigh (muslo izquierdo)
RT	Right thigh (muslo derecho)
LLA	Left lower arm (antebrazo izquierdo)
RLA	Right lower arm (antebrazo derecho)
LUA	Left upper arm (brazo superior izquierdo)
RUA	Right upper arm (brazo superior derecho)
BACK	Back (espalda)

Tabla 3. Sensores utilizados en REALDISP

Fuente: Elaboración propia basada en (Banos, Toth, et al., 2014)

Las medidas registradas por las unidades *Xsens* proporcionan información de la aceleración tridireccional, la orientación espacial mediante giroscopio y datos sobre el campo magnético. Adicionalmente, ofrecen una estimación de la orientación en cuatro dimensiones, en formato de cuaternión (Baños et al., 2012). En la tabla 4 se muestran todos los datos registrados por cada sensor.

Código	Dato del sensor
ACC:X	Componente X del acelerómetro
ACC:Y	Componente Y del acelerómetro
ACC:Z	Componente Z del acelerómetro
GYR:X	Componente X del giroscopio
GYR:Y	Componente Y del giroscopio
GYR:Z	Componente Z del giroscopio
MAG:X	Componente X del medidor de campo magnético
MAG:Y	Componente Y del medidor de campo magnético

MAG:Z	Componente Z del medidor del campo magnético
QUAT:1	Componente 1 del cuaternión (componente 1)
QUAT:2	Componente 2 del cuaternión (componente i)
QUAT:3	Componente 3 del cuaternión (componente j)
QUAT:4	Componente 4 del cuaternión (componente k)

Tabla 4. Datos registrados por cada sensor
Fuente: Elaboración propia basada en (Banos & Toth, 2014)

Por lo tanto, cada sensor genera 13 valores en cada medición, lo que en total produce un conjunto global de 117 señales registradas en cada instante de tiempo. Las grabaciones fueron muestreadas a una frecuencia de 50 Hz, es decir, realizando una medición cada 0,02 segundos (Banos & Toth, 2014).

4.2.2. Escenarios de desplazamiento

A la hora de realizar el estudio para realizar la recogida de datos que da lugar a REALDISP, se definen tres escenarios relativos a cómo están se posicionan los sensores sobre el sujeto. De esta forma, se presenta el concepto de desplazamiento gradual (Baños et al., 2012).

- Colocación ideal: Un instructor se encarga de colocar correctamente cada uno de los sensores en los sujetos. Es el escenario predeterminado.

- Auto-colocación: Se le pide al sujeto que sea él mismo el que coloque 3 de los 9 sensores. Este escenario intenta simular de forma más realista el uso diario de un sistema de reconocimiento de la actividad.

- Desplazamiento inducido: El instructor posiciona de forma errónea entre 4 y 7 sensores. Es un escenario para investigar cómo se degrada el rendimiento a medida que las condiciones se distancian de las ideales.

La figura 13 muestra un ejemplo de colocación del sensor de la parte superior del brazo izquierdo (el sensor con código *LUA*) en cada uno de los escenarios descritos.

Figura 13. Ejemplos de los posibles escenarios de colocación para un sensor
(a) Colocación ideal (b) Auto-colocación (c) Desplazamiento inducido
Fuente: (Banos, Toth, et al., 2014)

4.2.3. Actividades

El estudio registra los datos obtenidos mientras los sujetos realizan una serie de actividades deportivas o *fitness*. Como se muestra en la tabla 5, se trata de un conjunto de ejercicios frecuentes en el acondicionamiento físico.

Código	Actividad	
L0	Inactividad	
L1	Caminar	1 minuto
L2	Trotar	1 minuto
L3	Correr	1 minuto
L4	Saltar	20 repeticiones
L5	Saltar adelante y atrás	20 repeticiones
L6	Saltar a los lados	20 repeticiones
L7	Saltar abriendo y cerrando brazos y piernas	20 repeticiones
L8	Saltar a la comba	20 repeticiones
L9	Girar el tronco con los brazos extendidos	20 repeticiones
L10	Girar el tronco con los brazos recogidos	20 repeticiones
L11	Doblar la cintura hacia adelante	20 repeticiones
L12	Mover en círculos la cintura	20 repeticiones

L13	Doblar la cintura tocando el pie con la mano contraria	20 repeticiones
L14	Tocar los talones	20 repeticiones
L15	Doblar la cintura hacia los lados	10 repeticiones a la derecha 10 repeticiones a la izquierda
L16	Doblar la cintura hacia los lados levantando un brazo	10 repeticiones a la derecha 10 repeticiones a la izquierda
L17	Estiramiento repetitivo hacia adelante	20 repeticiones
L18	Giro opuesto de torso y cadera	20 repeticiones
L19	Elevación lateral de brazos	20 repeticiones
L20	Elevación frontal de brazos	20 repeticiones
L21	Palmadas frontales	20 repeticiones
L22	Cruce frontal de brazos	20 repeticiones
L23	Rotación de hombros con gran amplitud	20 repeticiones
L24	Rotación de hombros con poca amplitud	20 repeticiones
L25	Rotación interna de brazos	20 repeticiones
L26	Rodillas al pecho de forma alternativa	20 repeticiones
L27	Talones al glúteo de forma alternativa	20 repeticiones
L28	Agacharse en cuclillas	20 repeticiones
L29	Levantar las rodillas alternativamente	20 repeticiones
L30	Rotación de rodillas	20 repeticiones
L31	Remo	1 minuto
L32	Bicicleta elíptica	1 minuto
L33	Bicicleta normal	1 minuto

Tabla 5. Actividades registradas en REALDISP

Fuente: Elaboración propia basada en (Banos, Toth, et al., 2014)

Se incluyen actividades que involucran todo el cuerpo, como caminar (*L1*), saltar a la comba (*L8*) o ir en bicicleta (*L33*). Otras actividades son específicas para el tronco, como los giros con los brazos extendidos (*L9*) o doblar la cintura para intentar tocar cada pie con la mano contraria (*L13*). También se tienen en cuenta actividades para las extremidades superiores, como la elevación lateral de los brazos (*L19*) o las palmadas frontales (*L21*). Por último, hay actividades que sólo se relacionan con las extremidades inferiores, como llevar las rodillas al pecho de forma alterna (*L26*) o llevar lo talones al glúteo (*L27*) (Banos & Toth, 2014).

En este trabajo se ha decidido añadir de forma explícita la *inactividad*, el estado en el que se encuentra el sujeto cuando no está realizando ninguna de las otras actividades. En la tabla 5

aparece representada como *L0*, pasando de tener 33 actividades a 34. Como se justificará más adelante, tratar esta información como una actividad más, facilita las tareas de preparación de los datos y el entrenamiento de los modelos propuestos.

4.2.4. Ficheros de *log*

Los archivos con todos los datos de REALDISP están disponibles en la web de UC Irvine Machine Learning Repository (UCI Machine Learning Repository: REALDISP Activity Recognition Dataset Data Set, s. f.).

El conjunto de datos está formado por 46 ficheros con extensión *log*, cuyos nombres proporcionan la identificación del sujeto, el escenario de desplazamiento de los sensores y, en caso de tratarse de un desplazamiento inducido, cuántos sensores han sido colocados erróneamente. Se proporcionan algunos ejemplos, para entender mejor el patrón que sigue el nombre de estos archivos:

- *subject2_ideal.log* contiene los datos del sujeto número 2 en el contexto de unas condiciones ideales, en las que el instructor ha colocado los sensores de forma correcta.

- *subject9_self.log* almacena los datos del sujeto número 9, siendo él mismo el que ha procedido a la colocación de los sensores según su propio criterio.

- *subject15_mutual7.log* recoge los datos para el sujeto 15, al que se le han colocado 7 sensores de forma incorrecta a propósito.

Los archivos están estructurados como un conjunto de filas y columnas. Cada fila del archivo se corresponde con un registro de los valores proporcionados por los sensores, muestreados a una frecuencia de 50 Hz. De esta manera, una fila se divide en 120 columnas según se muestra en la tabla 6.

Las dos primeras columnas indican cuándo se ha recogido la muestra, almacenando por separado los segundos y los microsegundos de ese instante de tiempo.

Desde la columna 3 hasta la 119, aparecen los datos correspondientes a las medidas de los sensores. Se han utilizado 9 sensores capaces de registrar 13 mediciones diferentes cada uno, de ahí estas 117 columnas de información. El orden en el que aparecen los sensores es el especificado en la tabla 6 y, para cada uno de ellos, el orden de las mediciones es el que aparece en la tabla 4.

Columna	Descripción
1	Marca de tiempo (segundos)
2	Marca de tiempo (microsegundos)
3 - 15	Datos del sensor RLA
16 - 28	Datos del sensor RUA
29 - 41	Datos del sensor BACK
42 - 54	Datos del sensor LUA
55 - 67	Datos del sensor LLA
68 - 80	Datos del sensor RC
81 - 93	Datos del sensor RT
94 - 106	Datos del sensor LT
107 - 119	Datos del sensor LC
120	Etiqueta de la actividad

Tabla 6. Formato de los registros de los ficheros de *log*
Fuente: Elaboración propia basada en (Banos & Toth, 2014)

La última columna de cada registro de datos se corresponde con la etiqueta de la actividad que estaba realizando el sujeto en el momento de la medición. Es un entero positivo que se corresponde con los códigos de actividad mostrados en la tabla 5, denotando el valor 0 que no se estaba efectuando ninguna actividad; esto justifica la inclusión de *L0*, *inactividad*, en el conjunto de actividades.

4.2.5. Análisis previo

El propio trabajo en el que se presenta REALDISP, realiza una evaluación inicial de los datos a nivel estadístico (Baños et al., 2012).

El conjunto de datos completo contiene información registrada durante unas 39 horas. De ese tiempo, algo más de 10 horas se corresponden a la realización de actividades, siendo la diferencia la cantidad de datos correspondiente a lo que hemos llamado anteriormente *inactividad*. Las sesiones registradas en el escenario de colocación ideal y en el de auto-colocación, tienen una duración aproximada de 15 horas cada una. Las sesiones en las que se induce el desplazamiento duran un poco más de 10 horas.

El promedio de minutos transcurridos mientras se registra la actividad que realiza un sujeto es de 13,02 ± 5,26 en el escenario ideal (51,31 ± 20,35 teniendo en cuenta la inactividad), 13,96 ± 3,78 en el escenario de auto-colocación (50,59 ± 17,41 contando con la inactividad) y 14,75 ± 5,36 durante el desplazamiento inducido (49,24 ± 22,43 contando con la inactividad).

La tabla 7 muestra un resumen del tiempo dedicado a realizar las actividades, respecto al tiempo total registrado, según el tipo de escenario de desplazamiento.

Escenario	**Sujetos**	**Duración por sujeto (en minutos)** duración actividades / duración con inactividad	**Duración total (en minutos)** duración actividades / duración con inactividad
Colocación ideal	17	13,02 ± 5,26 / 51,31 ± 20,35	226,84 / 860,19
Auto-colocación	17	13,96 ± 3,78 / 50,59 ± 17,41	220,73 / 895,42
Desplazamiento inducido	3	14,75 ± 5,36 / 49,24 ± 22,43	188,45 / 632,28

Tabla 7. Resumen de los tiempos de registro de actividad

Fuente: Elaboración propia basada en (Baños et al., 2012)

Durante el análisis posterior de los datos, se identificaron algunas partes de las grabaciones como corruptas (Banos, Toth, et al., 2014). La figura 14 (a) muestra los datos de actividad que faltan para cada sujeto. Se aprecia que no hay datos de actividad disponibles para los sujetos 6 y 13, en el escenario de auto-colocación de los sensores. Para el participante 7 apenas existen datos para el escenario ideal. Además, faltan datos de ciertas actividades para algunos de los sujetos restantes.

Para el conjunto de datos relacionado con el desplazamiento inducido, se puede ver en la figura 14 (b) que la mayor pérdida de información sucede con el sujeto 3, ya que faltan todas las actividades desde la *L13* a la *L29*. Pero también faltan algunas actividades para el resto de los sujetos.

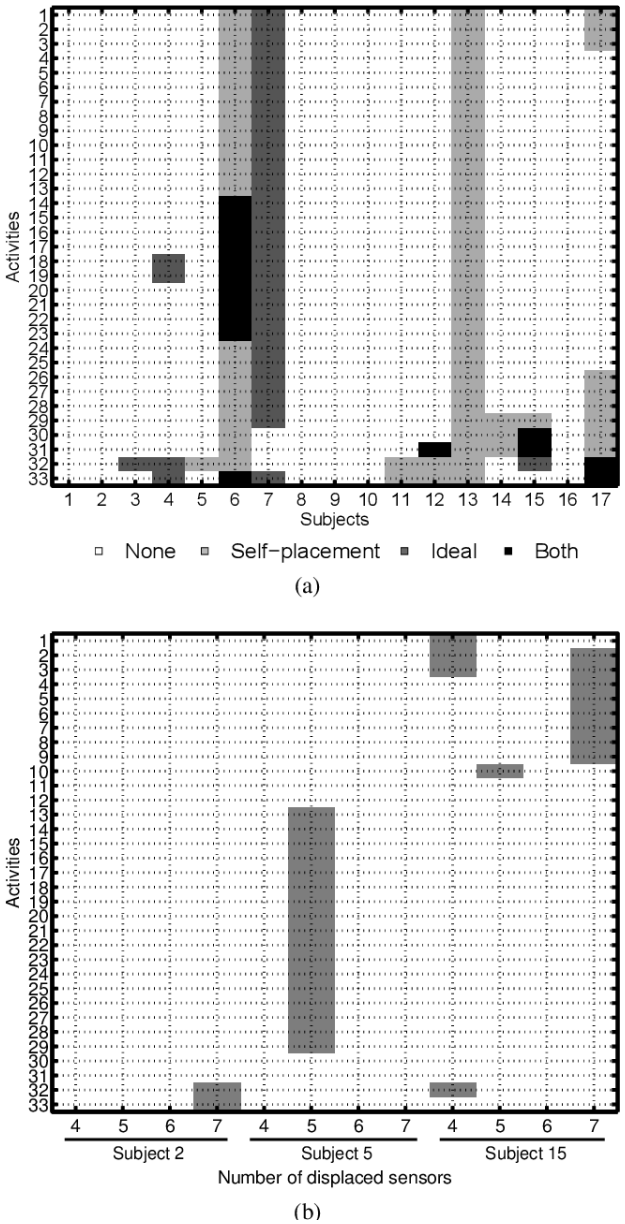

Figura 14. Datos de actividad faltantes para cada sujeto
(a) Colocación ideal y auto-colocación (b) Desplazamiento inducido
Fuente: (Banos, Toth, et al., 2014)

4.2.6. Análisis exploratorio de los datos

El análisis exploratorio ayuda entender grandes conjuntos de datos al condensar las principales propiedades de cada una de las variables que lo definen, muchas veces de una manera visual. En el caso de enfrentarse a problemas relacionados con el aprendizaje profundo, es una tarea fundamental previa al modelado de una posible solución.

En este trabajo, se han utilizado las siguientes técnicas para realizar el análisis exploratorio de los datos de REALDISP:

- Visualización: Es muy útil visualizar los datos de una variable, ver su distribución, observar sus valores máximos y mínimos (para variables numéricas) o su dominio (en variables categóricas).
- Búsqueda de valores faltantes: La cantidad de datos faltantes o vacíos para una variable es un indicador de su calidad y afecta a los modelos predictivos que la utilizan.
- Búsqueda de correlaciones: Estudiar la relación existente entre las variables que forman un conjunto de datos ayuda a seleccionar las características con las que entrenar un modelo de aprendizaje profundo.

Para realizar el análisis exploratorio se ha utilizado el lenguaje de programación Python en un cuaderno de Google Colaboratory. El código implementado para realizar este análisis puede consultarse íntegramente en el apéndice 1.

Google Colaboratory, también conocido como Google Colab, es un entorno gratuito que se ejecuta en la nube y que soporta el uso de cuadernos compatibles con Jupyter. Este tipo de cuadernos permite alternar el uso de bloques de código fuente y bloques de texto, característica que los convierte en una opción muy interesante para integrar y compartir un desarrollo de software y su documentación asociada.

Visualización

Como se ha indicado, los datos de REALDISP están repartidos entre 46 ficheros de *log*. En la tabla 8 pueden verse los cinco primeros registros para el sujeto 9 en condiciones ideales, que se encuentran en el archivo *subject9_ideal.log*. Para abreviar, se han mostrado sólo los datos de uno de los nueve sensores, el situado en la pantorrilla izquierda (código *LC*).

Esta breve inspección de las primeras líneas del fichero permite corroborar la información que ya se dispone sobre el conjunto de datos REALDISP. Aparece el segundo y el microsegundo en el que se toma la muestra, confirmando una frecuencia de muestreo de 50 Hz, ya que se realiza una medida cada 0,02 segundos. Seguidamente aparecen los datos del acelerómetro, el giroscopio, el campo magnético y los cuaterniones para el sensor de la pantorrilla izquierda. Y, por último, vemos la etiqueta asociada a la actividad, que toma valores entre 0 y 33 (siendo 0 la ausencia de actividad).

	0	1	2	3	4
second	0	0	0	0	0
microsecond	20000	40000	60000	80000	10000
lc_acc_x	-9,88	-9,6181	-9,7734	-9,9618	-9,6218
lc_acc_y	-0,17708	-0,23491	-0,19615	-0,6473	-0,45262
lc_acc_z	0,13546	0,20644	0.13697	0,2076	0,18776
lc_gyr_x	0,10506	0,098139	0,090249	0,12718	0,12957
lc_gyr_y	0,092358	0,071559	0,03494	0,0087181	0,0017662
lc_gyr_z	-0,045062	-0,075091	-0,10177	-0,10903	-0,082474
lc_mag_x	0,70093	0,69994	0,70005	0,69973	0,69865
lc_mag_y	0,06701	0,071177	0,07332	0,073504	0,075524
lc_mag_z	0,34438	0,34621	0,34598	0,34741	0,34738
lc_quat_1	0,70935	0,70886	0,70832	0,70853	0,70853
lc_quat_2	0,047252	0,047365	0,049326	0,047826	0,046855
lc_quat_3	0,70065	0,70103	0,70113	0,70071	0,70076
lc_quat_4	-0,060625	-0,061913	-0,065284	-0,068577	-0,068743
label_id	0	0	0	0	0

Tabla 8. Extracto de los primeros cinco registros de *subject9_ideal.log*

Fuente: Elaboración propia

Al *dataframe* resultante de la lectura del fichero de *log*, se le han añadido dos nuevas columnas para facilitar la exploración y visualización de los datos.

- *label_name*: La descripción asociada al código de la actividad etiquetada.
- *timestamp*: El instante de tiempo completo en el que se realiza la medición de los sensores, calculado a partir de los segundos y microsegundos originales.

En la figura 15 se observa que el número de muestras tomadas para el sujeto 9 en condiciones ideales, es de 205.498, lo que supone un total de 68,49 minutos registrados mientras realizaba algún tipo de actividad.

```
[9] data = read_realdisp_log(9, 'ideal')
    data.shape
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive"
Loaded gdrive/My Drive/realistic_sensor_displacement/subject9_ideal.log
(205498, 122)
```

Figura 15. Número de registros del fichero *subject9_ideal.log*

Fuente: Elaboración propia

Se puede apreciar en la figura 16 cómo se distribuye el número de muestras según la actividad realizada, incluyendo el estado en el que el sujeto se encuentra inactivo.

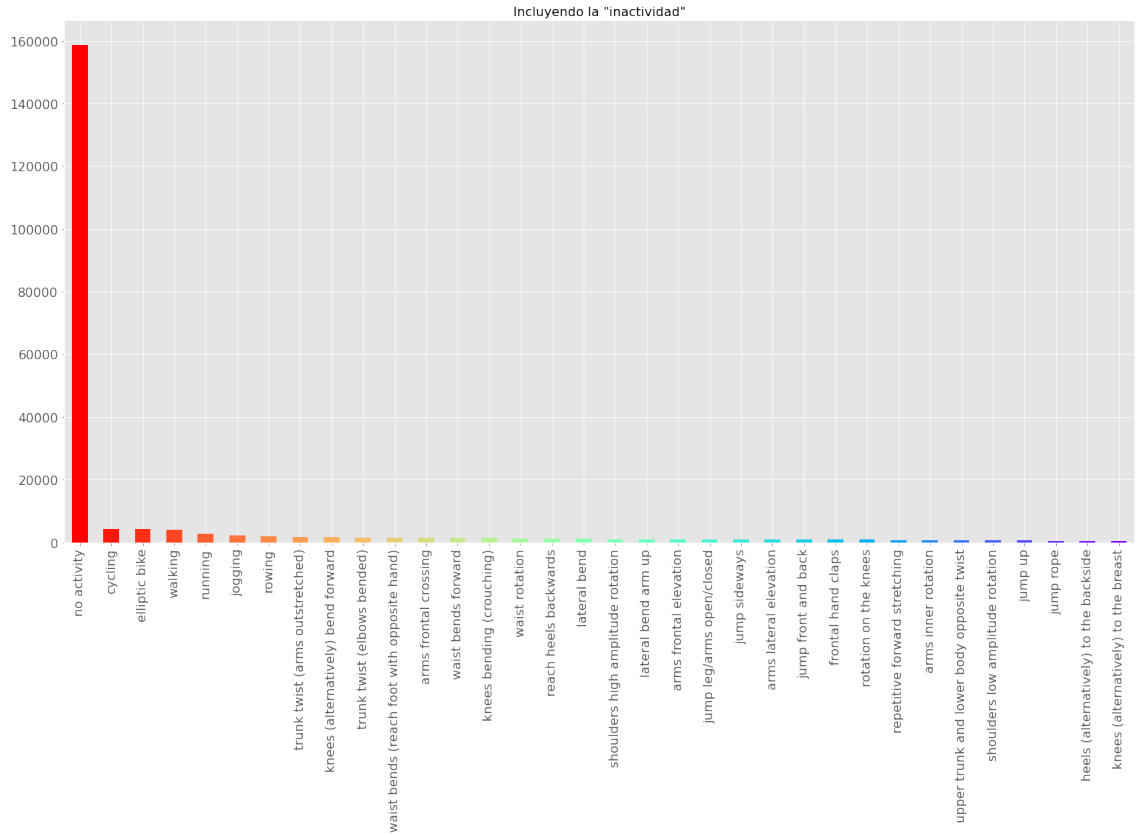

Figura 16. Número de muestras por actividad, incluyendo la *inactividad*

Fuente: Elaboración propia

Se observa que el número de muestras por actividad está muy desbalanceado, la mayoría pertenecen a la que se considera actividad *L0*, cuando no se está realizando ninguna actividad física concreta.

Es importante tener en cuenta esta información, puesto que a la hora de realizar el procesamiento de los datos y la selección de características, es necesario filtrar las muestras para quedarnos sólo con las que están etiquetadas con una actividad concreta (aquellas cuyos códigos van del *L1* al *L33*).

La figura 17 muestra la misma información, pero descartando el estado de *inactividad*.

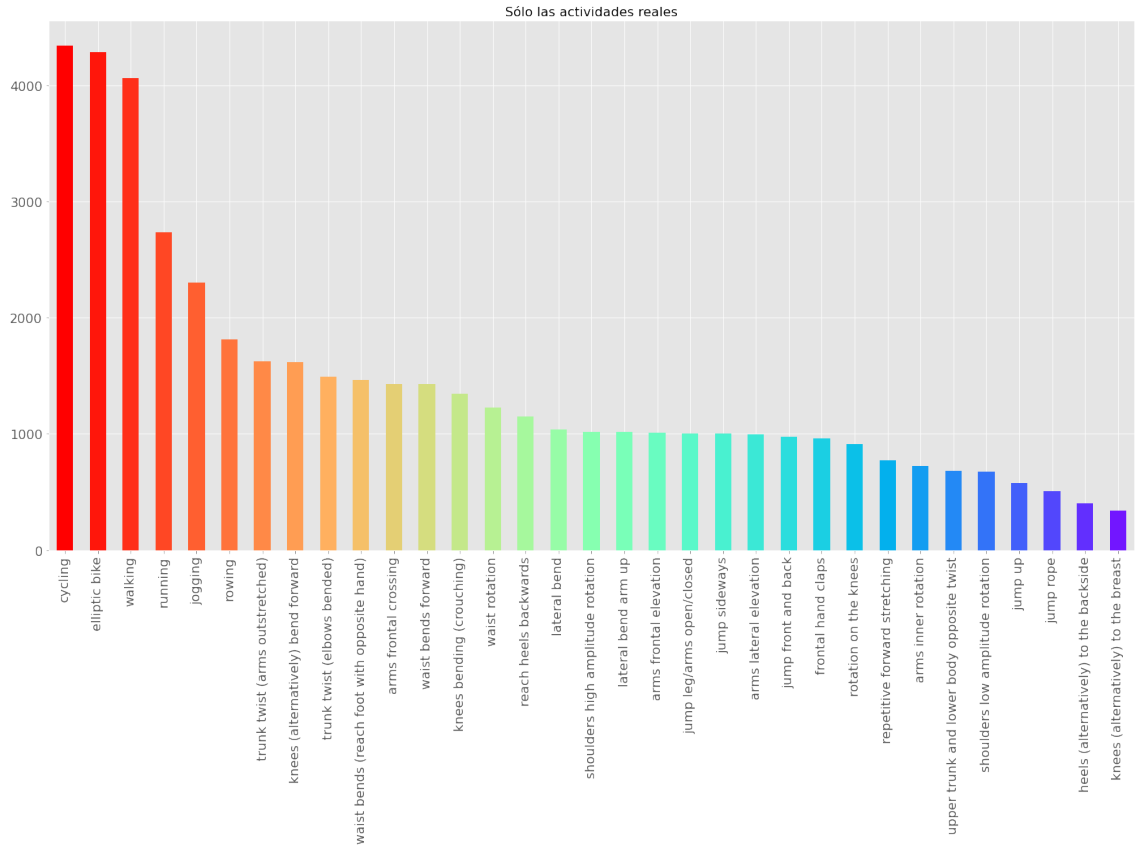

Figura 17. Número de muestras por actividad, sólo para las actividades reales

Fuente: Elaboración propia

Otra visualización interesante es la evolución de una medición concreta para un sensor a lo largo del tiempo de muestreo. En las figuras 18 y 19 puede verse cómo cambian los valores de la coordenada *x* de la medición del campo magnético, para el sensor de la pantorrilla izquierda. Las muestras están coloreadas según la actividad realizada.

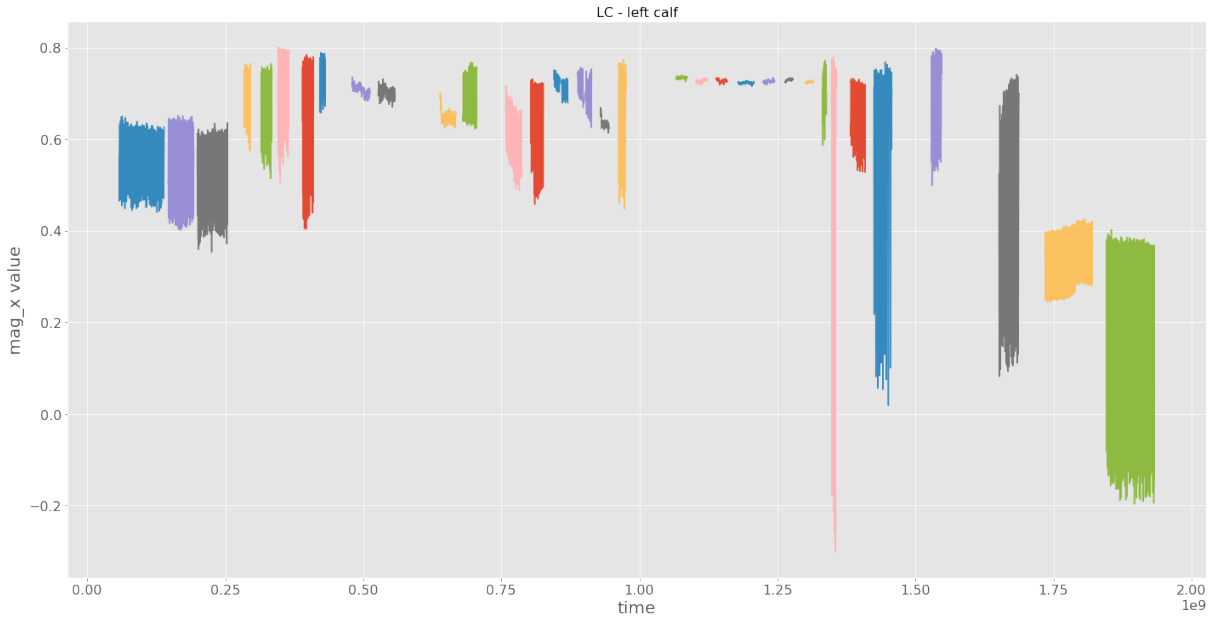

Figura 18. Evolución del valor *x* del campo magnético para el sensor *LC*

Fuente: Elaboración propia

Figura 19. Leyenda con el color de cada actividad de la figura 18

Fuente: Elaboración propia

En la figura 20 se muestra la evolución, a lo largo del tiempo de medición, para las tres variables que definen el campo magnético del sensor de la pantorrilla izquierda. Se aprecia cómo los datos varían según la actividad realizada; incluso se pueden distinguir los periodos en los que se está realizando una actividad que no implica el movimiento de las piernas. En el intervalo entre los 1.000 y los 1.300 segundos, aproximadamente, los valores apenas fluctúan. Este es justo el momento en el que se realizan las actividades de elevación lateral de brazos (*L19*), elevación frontal de brazos (*L20*), palmadas frontales (*L21*), cruce frontal de brazos (*L22*), rotación de hombros con gran amplitud (*L23*), rotación de hombros con poca amplitud (*L24*) y rotación interna de los brazos (*L25*).

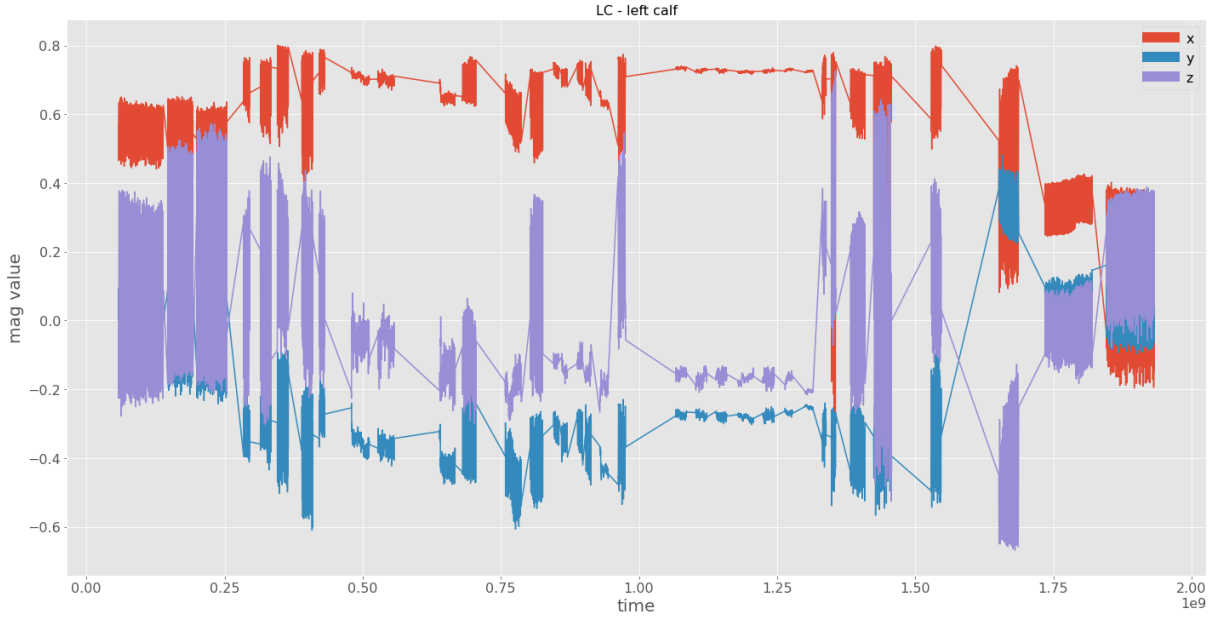

Figura 20. Evolución del valor del campo magnético para el sensor *LC*

Fuente: Elaboración propia

La figura 21 muestra una comparativa de la evolución de los valores de la coordenada *z* del campo magnético para dos sensores diferentes: el de la pantorrilla izquierda y el de la pantorrilla derecha. Se puede ver cómo existen actividades que implican a ambas piernas a la vez, mientras que en otras se mueven de forma alterna. En el intervalo entre 0 y 400 segundos, aproximadamente, se realizan las primeras actividades. Las tres iniciales se corresponden con caminar (*L1*), trotar (*L2*) y correr (*L3*), los valores de ambas pantorrillas cambian alternativamente, solapándose. Las cinco siguientes son saltar (*L4*), saltar adelante y atrás (*L5*), saltar a los lados (*L6*), saltar abriendo y cerrando los brazos y las piernas (*L7*) y saltar a la comba (*L8*), en todas ellas los valores cambian al unísono.

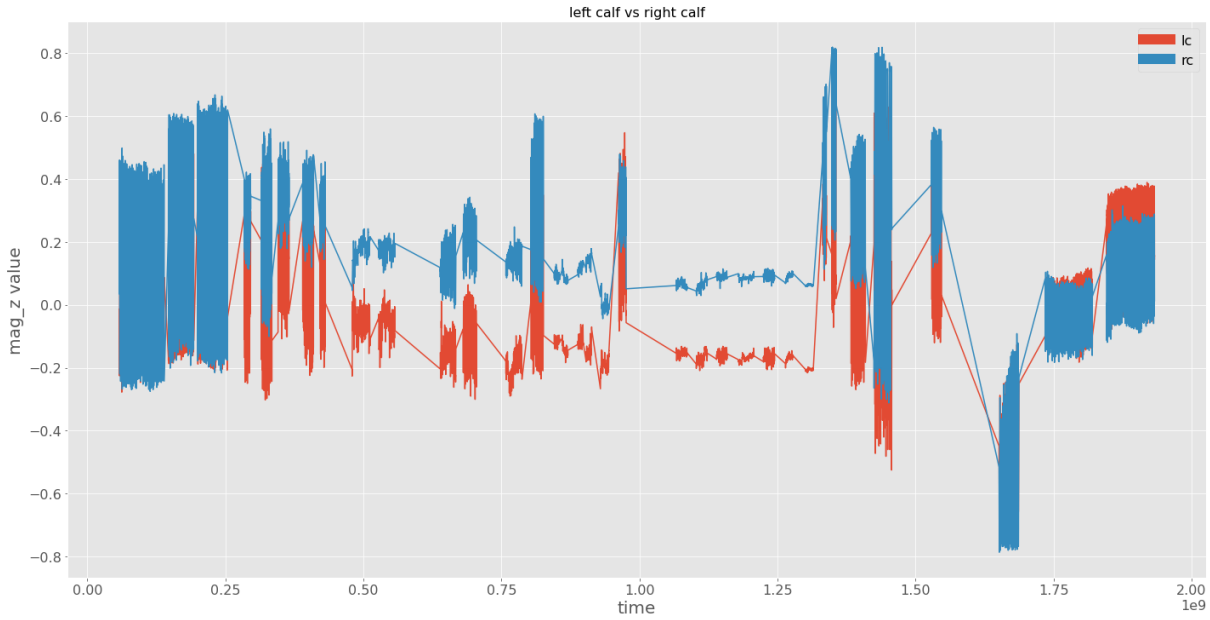

Figura 21. Evolución del valor *z* del campo magnético para los sensores *LC* y *RC*

Fuente: Elaboración propia

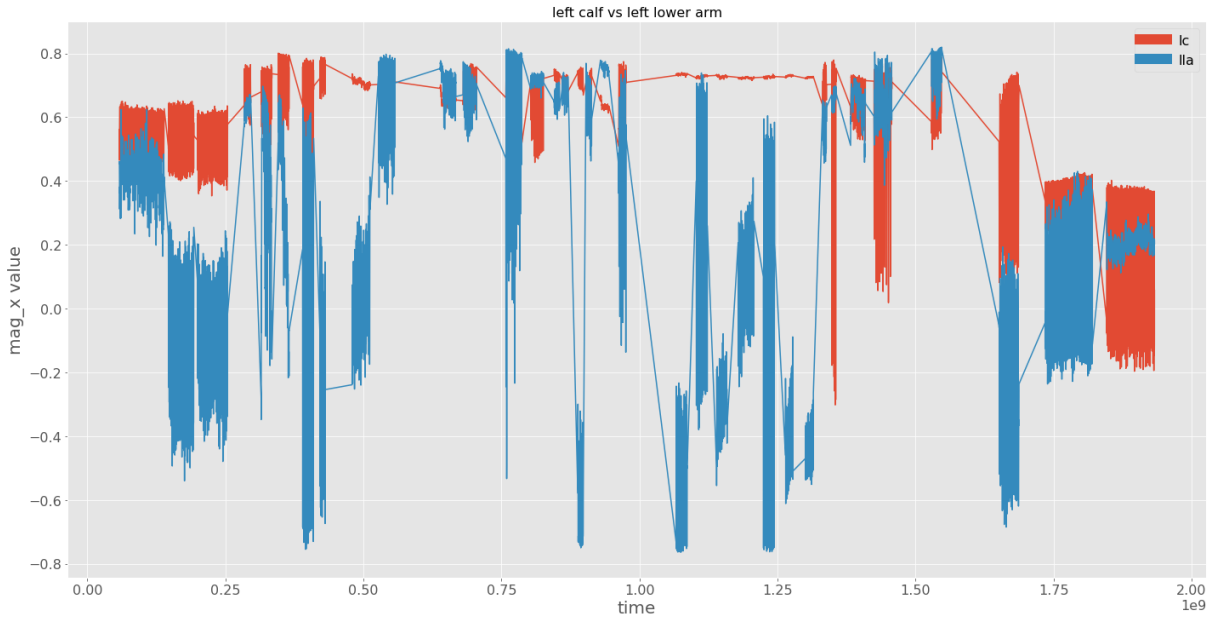

Figura 22. Evolución del valor *x* del campo magnético para los sensores LC y LLA

Fuente: Elaboración propia

Por último, se puede comparar la evolución del valor de la coordenada *x* del campo magnético para el sensor de la pantorrilla izquierda y la del sensor del antebrazo izquierdo. Como se observa en la figura 22, existe una mayor diferencia entre los valores. Además, se pueden aprecian actividades en las que sólo participan los antebrazos. En el intervalo entre los 1.000 y los 1.300 segundos, aproximadamente, los valores apenas varían en el sensor de la pantorrilla pero fluctúan de manera importante para el antebrazo. Este es justo el momento en el que se realizan las actividades de elevación lateral de brazos (*L19*), elevación frontal de brazos (*L20*), palmadas frontales (*L21*), cruce frontal de brazos (*L22*), rotación de hombros con gran amplitud (*L23*), rotación de hombros con poca amplitud (*L24*) y rotación interna de los brazos (*L25*).

Valores faltantes

Los valores nulos o faltantes en un conjunto de datos, son uno de los grandes problemas a los que se enfrenta un analista. Su gestión tiene un impacto directo sobre el entrenamiento de los modelos de aprendizaje profundo que los utilizan como entrada

A la hora de recopilar datos del mundo real, existen casos en los que la información obtenida no puede ser usada. Por ejemplo, los datos pueden corromperse si no son registrados de forma correcta y completa.

En el caso de REALDISP, ya se ha comentado en la sección 4.2.5 el análisis previo de los datos realizado para comprobar la validez de los mismos. Sabemos que existen casos como el del sujeto 7, que apenas dispone de datos bajo las condiciones ideales de posicionamiento de los sensores.

```
[41] data.isnull().sum()

     second         0
     microsecond    0
     RLA_ACC_X      0
     RLA_ACC_Y      0
     RLA_ACC_Z      0
                   ..
     lc_quat_3      0
     lc_quat_4      0
     label_id       0
     label_name     0
     timestamp      0
     Length: 122, dtype: int64
```

Figura 23. Comprobación de valores nulos en el fichero *subject9_ideal.log*

Fuente: Elaboración propia

Gracias a este estudio previo, el presente trabajo se centra en el uso de sujetos sin datos corruptos, como sucede con el participante 9. La figura 23 muestra el uso de la función *isnull* de la librería *Pandas*, para comprobar que no existen valores nulos en este subconjunto de los datos.

Correlaciones

En el apartado de visualización, se ha comprobado que existe cierta relación entre los datos proporcionados por los 9 sensores. Esta relación está estrechamente ligada a la actividad que se realiza. Según qué actividad se esté efectuando, se involucran unas partes del cuerpo u otras. Además, dependiendo de la actividad, también pueden establecerse relaciones de alternancia entre los sensores.

Para comprobar si existen una dependencia entre los sensores, se puede utilizar un mapa de calor. El mapa de calor o *correlograma*, es un tipo de diagrama que facilita la búsqueda de variables dependientes. En la figura 24 se muestra el nivel de dependencia de la componente *x* del acelerómetro, para los 9 sensores inerciales.

Existe cierto grado de dependencia del sensor de la espalda (*BACK*) con los sensores de los muslos (*LT* y *RT*) y los sensores de la parte superior de los brazos (*LUA* y *RUA*). También aparece una ligera dependencia entre los propios sensores de la parte superior de los brazos (entre *LUA* y *RUA*) y de los muslos (entre *LT* y *RT*). Por último, se observa un nivel de dependencia apreciable entre los sensores de la parte inferior de los brazos (entre *LLA* y *RLA*).

Por lo tanto, las dependencias que se reflejan en el mapa de calor, reafirman el hecho de que existe cierta lateralidad en las actividades registradas. Sin embargo, no puede decirse que aparezca una correlación directa entre los sensores: cada sensor es independiente del resto, está colocado en una posición del cuerpo diferente y ayuda a reconocer determinadas actividades.

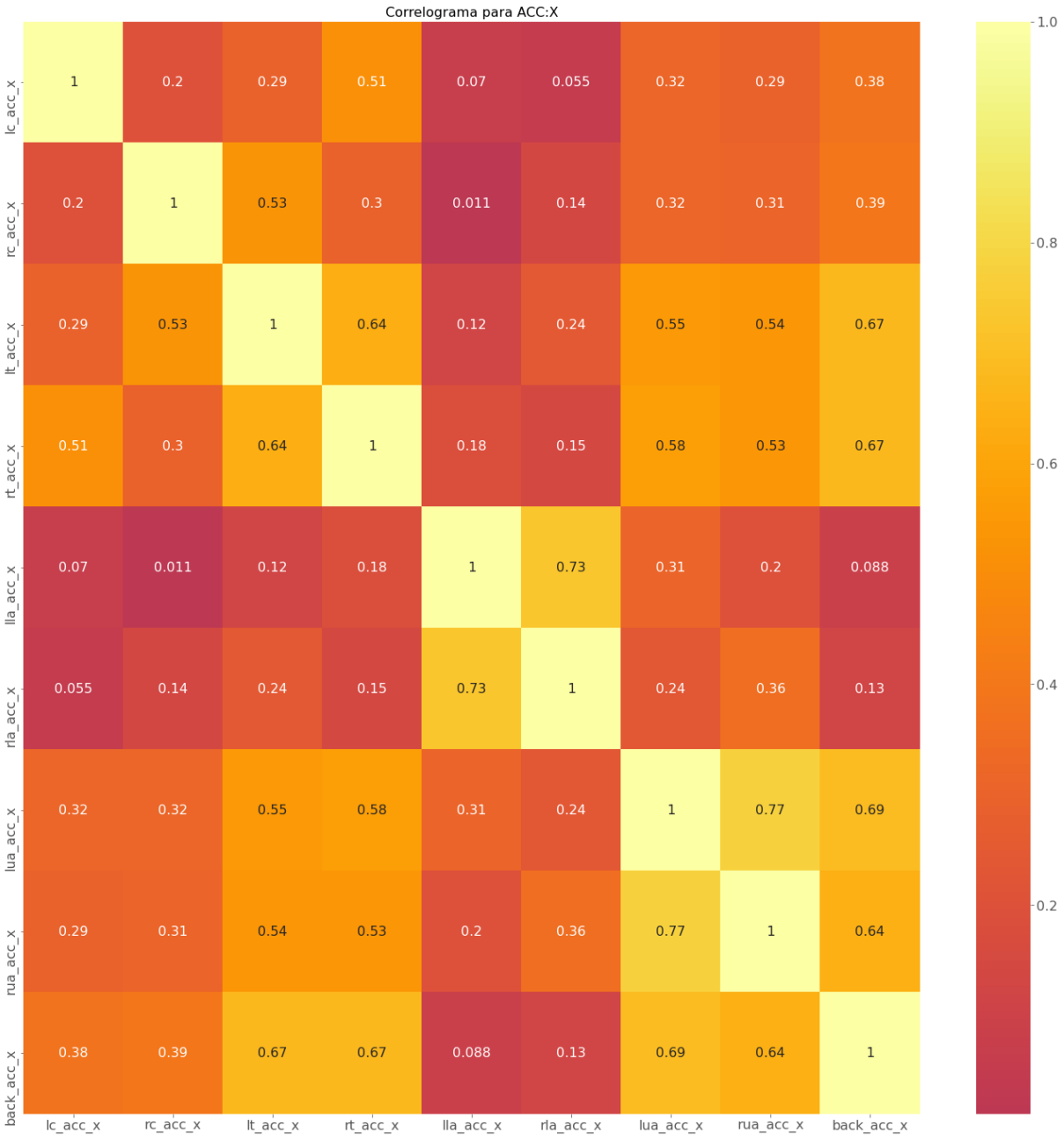

Figura 24. Mapa de calor para el valor *x* del acelerómetro de todos los sensores

Fuente: Elaboración propia

También se puede estudiar la correlación entre los datos de un único sensor. En la figura 25 se muestra el nivel de dependencia para los valores proporcionados por el sensor de la pantorrilla izquierda.

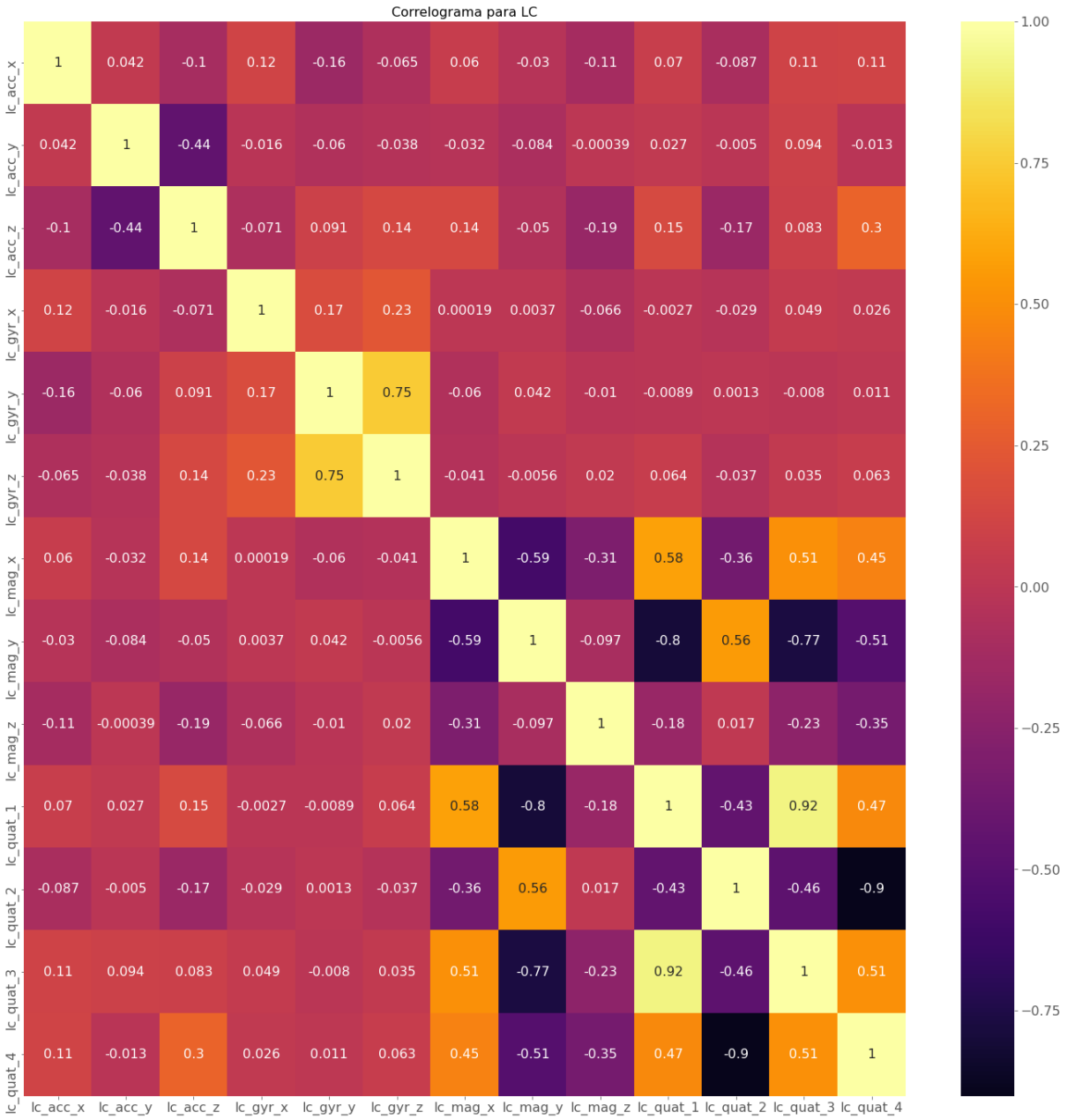

Figura 25. Mapa de calor para los datos del sensor LC

Fuente: Elaboración propia

Existe un grado de dependencia elevado (0,92) entre la primera y la tercera componente del cuaternión de orientación. También nos encontramos, aunque en menor medida (0,75), cierto nivel de dependencia entre la componente *x* y la componente *z* del campo magnético. Esta información es importante a la hora de realizar la preparación de los datos que alimentan el entrenamiento de un modelo de aprendizaje profundo.

4.3. Preparación de los datos

Las tareas de limpieza y preparación de los datos son fundamentales, y repercuten en la fase posterior de modelado establecida por CRISP-DM. Realizan la transformación de los datos originales en bruto, para obtener un nuevo conjunto que sirva de entrada al proceso de entrenamiento de los modelos de aprendizaje profundo propuestos.

Para este trabajo se han realizado operaciones estándar de preparación de los datos sobre REALDISP, como son la selección, transformación y normalización de los datos. También se han considerado otros procedimientos más específicos, que incluyen la aplicación de la técnica de la ventana deslizante (ver sección 2.6).

4.3.1. Selección de los datos

Sujetos y escenarios

En REALDISP disponemos de datos de 17 sujetos, en tres escenarios distintos, atendiendo al concepto de desplazamiento gradual de los sensores que se ha introducido previamente. Por lo tanto, existen muchas opciones posibles a la hora de seleccionar los datos para generar los conjuntos de entrenamiento, validación y pruebas.

Para este trabajo se ha decidido utilizar los datos de los sujetos 1, 2, 8, 9 y 10, bajo condiciones ideales de desplazamiento de los sensores. Por lo tanto, se seleccionan los registros de los ficheros:

- *subject1_ideal.log*
- *subject2_ideal.log*
- *subject8_ideal.log*
- *subject9_ideal.log*
- *subject10_ideal.log*

Se han seleccionado estos sujetos porque, como se vio en el análisis de la sección 4.2.5, en su caso se realizó el registro de todas las actividades en el escenario de colocación ideal sin datos corruptos o inválidos. En la decisión de usar este subconjunto entre todos los datos disponibles, también influye el intento de no sobrepasar los límites de memoria y tiempo de ejecución establecidos por el entorno de Google Colab. Explorar otras opciones, como

emplear los datos de todos los sujetos o utilizar otros escenarios de colocación de los sensores, queda para posibles trabajos futuros.

Actividades

En REALDISP se tienen en cuenta un total de 33 actividades deportivas diferentes (34 si tenemos en cuenta el estado de *inactividad*). Para este trabajo se ha decidido utilizar el conjunto completo de actividades, filtrando las muestras de datos para no tener en cuenta aquellas con la etiqueta *L0*, como se muestra en la figura 26. De esta manera, se descartan los datos proporcionados por los sensores cuando los sujetos no están realizando ninguna actividad real. De nuevo queda abierta para trabajos futuros la opción de seleccionar sólo determinadas actividades (por ejemplo, aquellas relacionadas con el tren superior).

```
[45] data_activities = data[data['label_id'] != 0]
```

Figura 26. Selección de las muestras etiquetadas con actividades reales
Fuente: Elaboración propia

Sensores

El conjunto de datos REALDISP está formado por las muestras recogidas usando 9 sensores *Xsens*. Cada uno de estos sensores proporciona una lectura compuesta por 13 variables (relacionadas con el acelerómetro, el giroscopio, el campo magnético y la orientación basada en cuaterniones).

Para este trabajo se ha optado por utilizar toda la información que ofrecen los sensores. Como trabajo futuro, se abre la puerta a realizar investigaciones utilizando sólo determinados sensores o ciertas variables de los mismos (por ejemplo, usar sólo la estimación de la orientación que ofrecen los cuaterniones).

4.3.2. Transformaciones y normalización de los datos

Un pre-procesamiento muy común al preparar conjuntos de datos para abordar el problema del HAR, es la estandarización de los valores (Panwar et al., 2017). La normalización es una actividad de gran importancia cuando se aplican técnicas de aprendizaje profundo que manipulan series temporales de datos (Baldominos et al., 2019).

Existen trabajos que abordan la problemática de la normalización utilizando específicamente el conjunto de datos REALDISP. En (Zhu et al., 2017) se evalúan seis métodos de normalización y se llega a la conclusión de que la que mejores resultados proporciona es la normalización vectorial. La normalización vectorial consiste en dividir cada elemento del vector entre la normal vectorial.

En este trabajo se ha aplicado la normalización vectorial a los datos que proporcionan las 117 señales temporales de todos los sensores. Se facilita así la fase de entrenamiento de los modelos de aprendizaje profundo propuestos más adelante. En la figura 27 se muestra el código en Python que permite realizar la normalización.

```
[ ]  # Normalización de los datos
     def normalize_data(df):
       df_to_normalize = df.drop(
                           columns = [
                             'second',
                             'microsecond',
                             'timestamp',
                             'label_id',
                             'label_name'
                           ]
                         )
       normalized = sklearn.preprocessing.normalize(df_to_normalize)
       df_normalized = pd.DataFrame(
                         data = normalized,
                         columns = df_to_normalize.columns,
                         index = df_to_normalize.index
                       )
       result = pd.concat(
                  objs = [
                    df_normalized,
                    df['second'],
                    df['microsecond'],
                    df['timestamp'],
                    df['label_id'],
                    df['label_name']
                  ],
                  axis = 1
                )
       return result
```

Figura 27. Función en Python encargada de la normalización de datos

Fuente: Elaboración propia

En la función *normalize_data*, primero se descartan las columnas que no se van a normalizar. A todas las demás se les aplica la función *sklearn.preprocessing.normalize* de la librería *scikit-learn*, que realiza la normalización vectorial. Por último, se crea un *dataframe* de *Pandas* con los datos ya normalizados y las columnas que se habían omitido.

Para finalizar con las transformaciones de datos, se aplica la técnica *one-hot encoding* (Potdar et al., 2017) a la variable categórica que representa la etiqueta asociada a cada muestra de datos, utilizando la función *keras.utils.to_categorical* de la librería *Keras*. Durante la fase de

implementación y evaluación de los modelos de aprendizaje profundo se justifica el uso de esta técnica, ya que las neuronas de la última capa de las arquitecturas propuestas utilizan SoftMax como función de activación.

4.3.3. Técnica de la ventana deslizante

La segmentación de señales es uno de los procesos más importantes a la hora de intentar resolver el problema del reconocimiento de la actividad humana. Uno de los enfoques que se usan normalmente para realizar la segmentación es la técnica de la ventana deslizante, introducida en la sección 2.6 del capítulo de contexto y estado del arte.

Es muy importante seleccionar un tamaño de ventana deslizante adecuado al problema que se está intentando resolver. Disminuir el tamaño de la ventana permite una detección más rápida de la actividad que se está realizando, además de requerir de menos recursos para hacerlo. Por el contrario, si las actividades a reconocer son complejas, normalmente es recomendable utilizar ventanas más grandes (Banos, Galvez, et al., 2014).

El desplazamiento también juega un papel interesante, ya que puede utilizarse como técnica de *data augmentation*. Se pasa de tener un conjunto de *n* muestras, cada una en un instante de tiempo concreto (cada 0,02 segundos en REALDISP, debido a la tasa de muestreo de 50 Hz), a tener un número de muestras *n'* que depende del tamaño de la ventana (*s*) y de cuánto solapamiento exista entre ellas (*d*):

$$n' = \frac{n - s}{d}$$

Por lo tanto, a mayor solapamiento, mayor número de muestras tendrá el conjunto de datos final que se utilizará para entrenar los modelos de aprendizaje automático.

En (Banos, Galvez, et al., 2014) se demuestra que en el caso de REALDISP, las ventanas reducidas (de 2 segundos o menos) proporcionan el mejor nivel de rendimiento y detección de actividades.

En este trabajo de fin de máster se han realizado diferentes pruebas experimentando con el tamaño de la ventana y el desplazamiento, aunque en la implementación final se ha decidido utilizar una ventana de 4 segundos (200 muestras) con un desplazamiento de 4 segundos (200 muestras), lo que supone que no existe solapamiento. Queda abierto a trabajos futuros el poder comparar distintos parámetros de segmentación, profundizando en cómo afecta a los resultados de los modelos.

```
[ ]  # Segmentación de los datos
     def slice_by_activity(df, window_size, window_shift, activity):
       df_activity = df[df.label_id == activity]
       segments = []
       labels = []
       for i in range(0, len(df_activity) - window_size, window_shift):
         segment = df_activity.drop(
                     columns = ['second', 'microsecond', 'timestamp', 'label_id', 'label_name']
                   ).values[i: i + window_size]
         segments.append(segment)
         labels.append(activity)
       reshaped_segments = np.asarray(segments).reshape(-1, window_size, FEATURES)
       labels = np.asarray(labels)
       return reshaped_segments, labels
```

Figura 28. Función de segmentación utilizando la técnica de ventana deslizante

Fuente: Elaboración propia

La figura 28 muestra la implementación en Python de la función encargada de realizar la segmentación. Para cada actividad, se recorre el conjunto de datos, seleccionando bloques del tamaño especificado por *window_size* y desplazándose tantos registros como indique *window_shift*. Por lo tanto, los parámetros *window_size* y *window_shift* se corresponden, respectivamente, con el tamaño de la ventana (*s*) y el desplazamiento (*d*).

En el código se aprecia que la segmentación se realiza para una actividad en concreto. Esto es debido a dos motivos:

- Se facilita el proceso de etiquetado de cada segmento. Al tratar con registros que siempre pertenecen a la misma actividad, no es necesario realizar ningún cálculo para conocer la etiqueta de los segmentos que se están generando.
- Evitamos problemas con el uso de la memoria en Google Colab. Este entorno establece unos límites de memoria fácilmente superables si se intenta hacer la segmentación de todas las actividades en un único paso.

Aunque no deja de ser obvio, es necesario destacar la importancia de realizar la segmentación de los datos antes de aplicar cualquier técnica de mezcla de datos o *data shuffling*. Estos procedimientos son fundamentales para el entrenamiento de los modelos de aprendizaje profundo, pero aplicarlos previamente a la segmentación hace que ésta deje de tener sentido.

4.3.4. Conjuntos de datos de entrenamiento, validación y pruebas

En la sección 2.5.4 se introduce el concepto de *early stopping*, que ayuda a evitar el sobreajuste en los modelos. Para poder utilizar esta técnica de regularización, la sección 2.5.5

indica cómo dividir el total de datos disponibles en tres conjuntos: entrenamiento, validación y pruebas.

En este trabajo, el conjunto total de datos de REALDISP se divide en estos tres grupos, tras realizar la segmentación utilizando la técnica de la ventana deslizante.

Como se muestra en la figura 29, la función *sklearn.model_selection.train_test_split* de la librería *scikit-learn*, nos permite seleccionar el 20% de las muestras para obtener un conjunto de pruebas. Esta función también realiza la mezcla o *shuffling* de datos. Esto es muy importante, ya que todos los registros están ordenados temporalmente y por actividad, debido al método de segmentación.

```
[ ]   x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

Figura 29. Creación de los conjuntos de entrenamiento y de pruebas
Fuente: Elaboración propia

Más adelante, durante el proceso de entrenamiento, en cada iteración se selecciona de nuevo el 20% de los datos de entrenamiento, para obtener un conjunto de validación. Esto significa que en cada iteración se utiliza un conjunto de entrenamiento y de validación diferente, lo que también actúa como técnica de regularización.

4.4. Modelado

Las redes neuronales poseen un enfoque bio-inspirado, ya que su funcionamiento se basa en las conexiones que se establecen en las neuronas del cerebro humano. Están formadas por neuronas artificiales, agrupadas en capas. Cada neurona de la red aplica una función matemática a los valores que recibe como entrada, siendo su salida la entrada de otras neuronas. Una red neuronal profunda (en inglés Deep Neural Network, DNN) es una red neuronal artificial con múltiples capas ocultas, entre la capa de entrada y la de salida (Schmidhuber, 2015). Se habla de profundidad en dos sentidos:

- Son profundas por su arquitectura, están formadas por múltiples capas con un gran número neuronas y de relaciones entre ellas.

- Son profundas porque conforme aumenta el número de capas y neuronas, la red es capaz de aprender conceptos cada vez más complejos.

El objetivo de este trabajo es evaluar y comparar modelos de redes profundas con distintas arquitecturas, que sean capaces de resolver el problema del reconocimiento de la actividad humana.

4.4.1. Modelo Fully Connected Deep Neural Network

El primer modelo diseñado para solucionar el problema del HAR es una red neuronal profunda completamente conectada (en inglés Fully Connected Deep Neural Network, FCDNN). La topología de una red neuronal de este tipo cuenta con una capa de entrada, varias capas ocultas (de ahí que se considere profunda) y una capa de salida.

Las neuronas de la capa de entrada se alimentan directamente con las características del conjunto de datos. Por lo tanto, el número de neuronas de la capa de entrada es igual al número de *features* seleccionadas.

Las siguientes capas son las ocultas, cuyas neuronas aplican la función de activación a las entradas que reciben, y generan una salida. Al ser completamente conectada, la salida de cada una de las neuronas de una capa está conectada con la entrada de todas las neuronas de la capa siguiente.

Por último, la capa de salida está formada por tantas neuronas como clases a predecir tenga el problema de clasificación.

La arquitectura del modelo FCDNN propuesta para este trabajo se compone de una capa de entrada y cuatro capas ocultas, como puede verse en la figura 30. Se distribuyen de la siguiente manera:

- La capa de entrada, que no aparece en la figura, está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).

- La primera capa de la figura se corresponde con la primera capa oculta de la red. Esta capa está formada por 128 neuronas, usando ReLU como función de activación.

- La segunda es una capa de regularización, de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste del modelo.

- La tercera capa, de nuevo es una capa densa, completamente conectada, en este caso con 64 neuronas y de nuevo con ReLU como función de activación.
- Por último, la cuarta capa define la que es considerada capa de salida. Está compuesta por tantas neuronas como etiquetas a predecir, es decir, tantas neuronas como actividades. En nuestro caso, son 33 (ya que no se tiene en cuenta la *inactividad*). La función de activación de estas neuronas es SoftMax, utilizada ampliamente en los problemas de clasificación multiclase como es el caso.

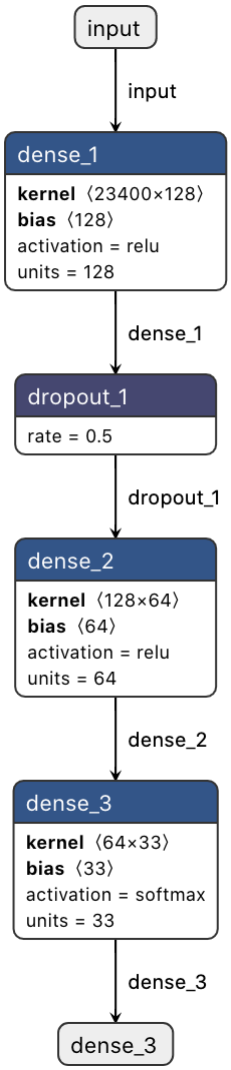

Figura 30. Arquitectura del modelo de red FCDNN

Fuente: Elaboración propia

4.4.2. Modelo Long Short-Term Memory

La siguiente arquitectura modelada para realizar el reconocimiento de actividades humanas es una red de gran memoria a corto plazo (en inglés Long Short-Term Memory, LSTM).

Los modelos de red LSTM son un tipo de red neuronal recurrente que puede aprender y recordar patrones durante largas secuencias de datos. Por lo tanto, suelen emplearse cuando los datos de entrada consisten en series temporales prolongadas, como sucede en el HAR.

La arquitectura de la red LSTM propuesta para este trabajo se describe en la figura 31, está formada pos las siguientes capas:

- La capa de entrada, que no aparece en la figura, está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).

- La primera capa de la figura, de tipo *long short-term memory*, se corresponde con la primera capa oculta de la red. Está formada por 128 neuronas, que se activan mediante la función *tanh*. La función de activación ReLU parece inapropiada para las RNNs, ya que pueden generar valores de salida muy amplios y por lo tanto aumentar la probabilidad de divergencia (Le et al., 2015).

- La segunda capa es de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste a los datos de entrenamiento.

- La tercera capa es de tipo denso, completamente conectada, que interpretará las características extraídas por la capa recurrente. Esta capa tiene 128 neuronas y utiliza ReLU como función de activación.

- Por último, la cuarta capa define la que se considera capa de salida. Está compuesta por tantas neuronas como etiquetas a predecir, es decir, tantas neuronas como actividades. En nuestro caso, son 33 (ya que no se tiene en cuenta la *inactividad*). La función de activación de estas neuronas es SoftMax, utilizada ampliamente en los problemas de clasificación multiclase.

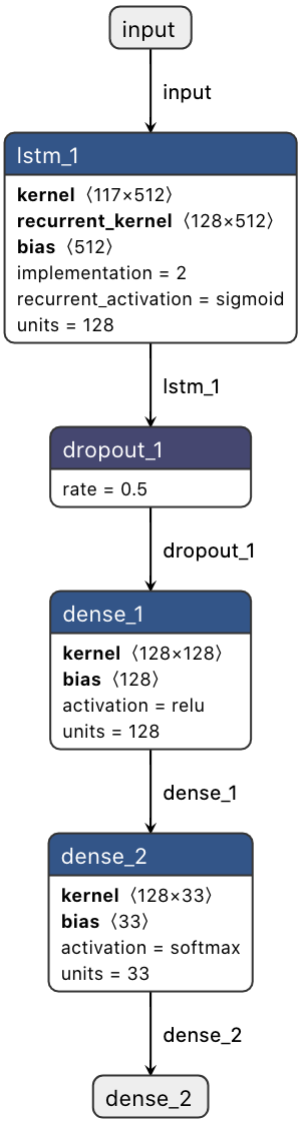

Figura 31. Arquitectura del modelo de red LSTM

Fuente: Elaboración propia

4.4.3. Modelo Convolutional Neural Network + Long Short-Term Memory

Una arquitectura de red más compleja que la Long Short-Term Memory, es la arquitectura CNN + LSTM. Utiliza capas de una red neuronal convolucional (en inglés Convolutional Neural Network, CNN) para realizar la extracción de características a partir de los datos de entrada y combina su funcionamiento con una red LSTM para completar la predicción del resultado (Sainath et al., 2015).

Las redes CNN + LSTM se desarrollaron para resolver problemas de predicción de series temporales visuales, así como para generar descripciones textuales a partir de secuencias de imágenes (Donahue et al., 2015). En especial, problemas como:

- Reconocimiento de la actividad, generar una descripción textual a partir de una secuencia de imágenes que muestran una actividad.
- Descripción de imágenes, generar una descripción textual de una única imagen.
- Descripción de video, generar una descripción textual a partir de una secuencia de imágenes.

En la figura 32 se muestra la arquitectura establecida para este tercer modelo de aprendizaje profundo. Las primeras cinco capas definen la parte convolucional de la red, que lee bloques de la secuencia y realiza la extracción de características. El resultado alimenta a las capas de una red LSTM definida con la misma topología que el modelo de la sección 4.4.2, y se encarga de interpretar las características recibidas de cada bloque de datos y realizar las predicciones.

La parte del modelo que representa una CNN está envuelta en una capa de tipo *TimeDistributed*, lo que permite ir leyendo en bloques cada uno de los grupos en los que ha resultado la segmentación de los datos de entrada. Sigue un patrón estándar, con dos capas convolucionales consecutivas seguidas de una capa de *dropout* y una de *max pooling*:

- La capa de entrada, que no aparece en la figura, está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).
- La primera y la segunda capa son de tipo convolucional. Ambas tienen 64 filtros, un tamaño de *kernel* de 3 y utilizan ReLU como función de activación de sus neuronas.
- La tercera capa es de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste.
- La cuarta capa realiza un *max pooling* de tamaño 2, reduce el número de variables, preparando y adaptando la salida para la red LSTM.
- Por último, una quinta capa aplana los datos para convertirlos en una serie temporal que puede manejar la red LSTM que viene a continuación.

La segunda parte del modelo es una red LSTM como la definida en la sección 4.4.2.

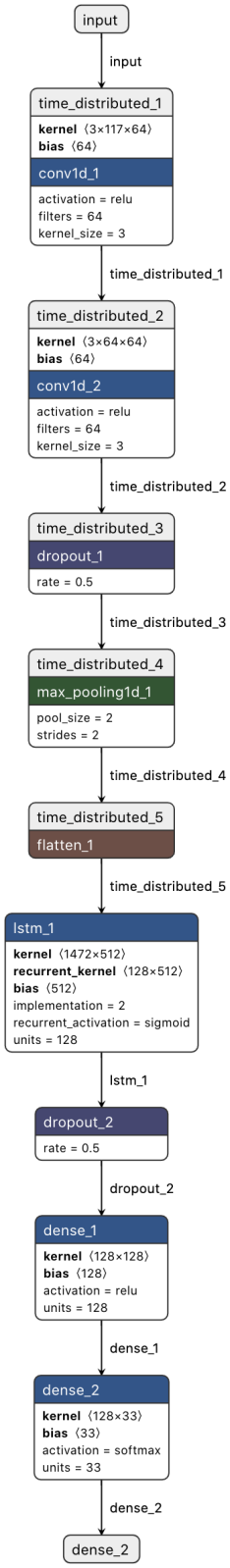

Figura 32. Arquitectura del modelo de red CNN + LSTM

Fuente: Elaboración propia

4.4.4. Modelo Convolutional Long Short-Term Memory

Las ideas propuestas por los modelos CNN + LSTM pueden avanzar un paso más. Una posible modificación es que las convoluciones de la CNN formen parte de la red LSTM. Esta combinación se llama Convolutional LSTM o ConvLSTM (Xingjian et al., 2015).

A diferencia de un modelo LSTM, que lee los datos directamente, y de CNN + LSTM, que interpreta la salida del modelo CNN, ConvLSTM utiliza las convoluciones como parte de la lectura de los datos de entrada en las unidades LSTM.

En la figura 33 se muestra la arquitectura establecida para este último modelo de aprendizaje profundo. Se utiliza una capa de tipo Convolutional LSTM y luego se aplana para poder aplicar una capa densa totalmente conectada, antes de que se use una capa de salida formada por 33 neuronas, una por cada posible clase a predecir. En concreto, se tienen las siguientes capas:

- La capa de entrada, que no aparece en la figura, está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).
- La primera capa es de tipo Convolutional LSTM. Tiene 64 filtros, un tamaño de *kernel* de 3 y utiliza ReLU como función de activación de sus neuronas.
- La segunda es una capa de regularización, de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste.
- La tercera capa aplana los datos para que puedan ser gestionados por las capas densas que vienen a continuación.
- La cuarta capa es de tipo denso, completamente conectada. Tiene 128 neuronas y utiliza ReLU como función de activación.
- Por último, la quinta capa es la considerada como capa de salida de la red. Está compuesta por tantas neuronas como etiquetas a predecir, es decir, tantas neuronas como actividades. En nuestro caso, son 33 (ya que no se tiene en cuenta la *inactividad*). La función de activación de estas neuronas es SoftMax, como en los modelos anteriores.

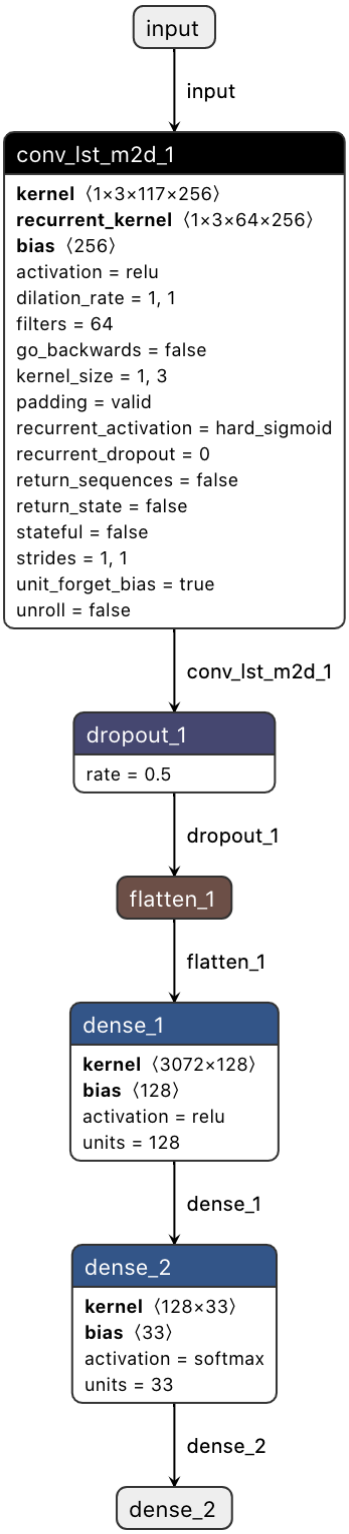

Figura 33. Arquitectura del modelo de red ConvLSTM

Fuente: Elaboración propia

5. Descripción de los resultados

Se han propuesto cuatro modelos de aprendizaje profundo para solucionar el problema del reconocimiento de la actividad humana:

- Fully Connected Deep Neural Network.
- Long Short-Term Memory.
- Convolutional Neural Network + Long Short-Term Memory.
- Convolutional Long Short-Term Memory.

En este capítulo se describe la implementación de los modelos, su entrenamiento, optimización y evaluación. Para ello se hace uso de las herramientas y técnicas descritas en el capítulo de contexto y estado del arte. De esta manera se completa la fase de evaluación de CRISP-DM, tras comparar las diferentes arquitecturas propuestas y decidir si se ha logrado el objetivo establecido.

5.1. Modelo Fully Connected Deep Neural Network

5.1.1. Entrenamiento y validación

La implementación del modelo de red neuronal FCDNN se lleva a cabo utilizando la librería de alto nivel *Keras*. La clase *Sequential* permite definir redes neuronales como una pila de capas, de manera que se conecta la salida de una con la entrada de la siguiente. Además, proporciona herramientas que permiten entrenar la red y utilizarla para hacer predicciones.

La figura 34 muestra el código en Python que implementa el modelo FCDNN definido en la sección 4.4.1.

```
[ ] # Implementación del modelo FCDNN
    model_fcdnn = Sequential()
    model_fcdnn.add(Dense(units = 128, activation = 'relu', input_shape = (input_shape, )))
    model_fcdnn.add(Dropout(rate = 0.5))
    model_fcdnn.add(Dense(units = 64, activation = 'relu'))
    model_fcdnn.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
```

Figura 34. Implementación del modelo FCDNN

Fuente: Elaboración propia

Tras definir el modelo el siguiente paso es compilarlo. Es en este momento cuando se especifica el algoritmo de optimización y la función de pérdida empleada. La figura 35 muestra la compilación del modelo utilizando la función *compile* de *Keras*. Se especifica Adam como algoritmo de optimización, entropía cruzada categórica como función de pérdida, y la exactitud o *accuracy* como métrica de evaluación durante el entrenamiento.

```
[ ] # Compilación del modelo FCDNN
    model_fcdnn.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Figura 35. Compilación del modelo FCDNN

Fuente: Elaboración propia

La tabla 9 muestra el resumen de la arquitectura del modelo, incluyendo el número de parámetros por capa y el número total.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	2995328
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 33)	2145
Total params: 3,005,729 **Trainable params: 3,005,729** **Non-trainable params: 0**		

Tabla 9. Resumen de la arquitectura del modelo FCDNN

Fuente: Elaboración propia

El número de parámetros de una capa se calcula multiplicando el número de parámetros por neurona, por el número de neuronas que hay en la capa. Al ser una red completamente conectada, el número de parámetros de una neurona es el número de neuronas de la capa anterior más el bias de la propia neurona. A modo de ejemplo, se calcula la cantidad de parámetros para el modelo propuesto:

- La primera capa tiene 128 neuronas, y cada neurona tiene 23401 parámetros, lo que da un total de 2995328. El número de parámetros de estas neuronas viene del número de variables de entrada (117 valores proporcionados en cada medida de los sensores), multiplicado por el tamaño de la ventana deslizante (200 en este caso), más 1 por el bias de la neurona.
- La segunda capa es la de *dropout* y no tiene parámetros.
- La tercera capa tiene 64 neuronas con 129 parámetros cada una (128 de la primera capa más 1 por el bias), un total de 8256.
- La cuarta capa tiene 33 neuronas con 65 parámetros cada una (64 de la capa anterior más 1 por el bias), un total de 2145.

En total el modelo propuesto tiene 3.005.729 parámetros, todos ajustables durante el proceso de entrenamiento.

Es el momento de proceder al entrenamiento de la red, que se realiza con la función *fit* de *Keras*. Los parámetros principales de la función son el conjunto de registros de entrenamiento y el conjunto de etiquetas asociadas a los mismos. También se especifica el número de épocas o iteraciones que se van a realizar durante el entrenamiento, y el tamaño de lote, que establece cómo se fraccionará el conjunto de datos de entrenamiento. Además, también se especifica el tamaño del conjunto de validación. En la figura 36 se muestra el entrenamiento para el modelo, estableciendo que se quiere utilizar el 20% de los datos para realizar la validación. El número de épocas y el tamaño del lote son hiperparámetros con los que se ha experimentado, dejándolos en sus valores definitivos (para este modelo) de 10 épocas y 64 elementos por lote.

```
[ ]  # Entrenamiento del modelo FCDNN
     history_fcdnn = model_fcdnn.fit(x_train_fcdnn,
                                     y_train_fcdnn,
                                     batch_size = batch_size,
                                     epochs = epochs,
                                     validation_split = 0.2,
                                     verbose = 1)
```

Figura 36. Entrenamiento del modelo FCDNN

Fuente: Elaboración propia

La figura 37 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento. Se aprecia claramente que la red deja de mejorar a partir de la cuarta época, así que haciendo uso de la técnica de *early stopping* descrita en la sección 2.5.4, se puede parar el entrenamiento.

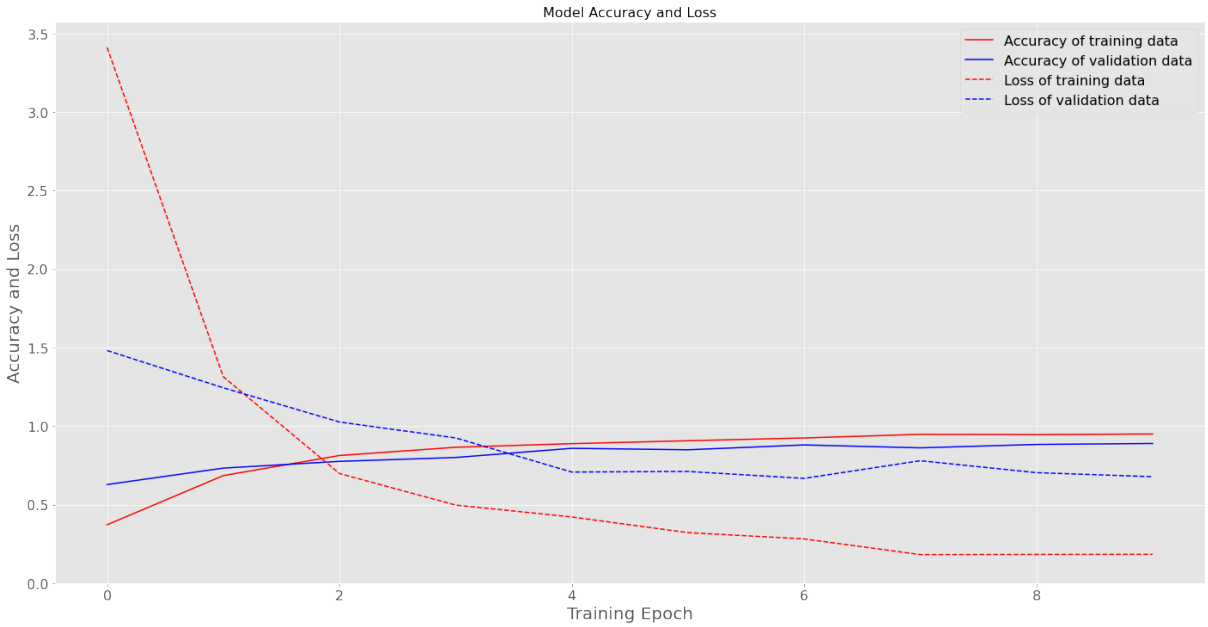

Figura 37. Métricas de *accuracy* y de *loss* en el entrenamiento del modelo FCDNN

Fuente: Elaboración propia

5.1.2. Evaluación

Una vez entrenado, se puede utilizar la función *predict* de *Keras* para evaluar cómo se comporta el modelo con los datos de prueba.

La figura 38 muestra el resultado de la evaluación del modelo entrenado durante 4 épocas, usando la función *classification_report* de *Keras*. El modelo tiene una exactitud global del 85%. A nivel de clases, se puede ver cómo en general funciona bastante bien. Es capaz de acertar y diferenciar las actividades de bicicleta elíptica (*L32*) y bicicleta normal (*L33*), por poner sólo un ejemplo. Sin embargo, tiene problemas con la actividad de llevar los talones al glúteo (*L27*).

	precision	recall	f1-score	support
1	1.00	0.97	0.99	37
2	0.93	0.74	0.82	19
3	0.84	1.00	0.91	32
4	1.00	0.14	0.25	7
5	0.50	0.67	0.57	3
6	1.00	0.67	0.80	9
7	1.00	0.86	0.92	7
8	0.00	0.00	0.00	1
9	0.95	0.87	0.91	23
10	0.67	0.89	0.76	9
11	1.00	0.90	0.95	10
12	0.80	1.00	0.89	12
13	0.90	0.90	0.90	20
14	0.70	0.58	0.64	12
15	1.00	0.44	0.62	9
16	0.93	0.93	0.93	15
17	0.71	1.00	0.83	5
18	1.00	0.29	0.44	7
19	0.86	1.00	0.92	6
20	0.56	0.45	0.50	11
21	0.67	0.80	0.73	5
22	0.73	0.62	0.67	13
23	0.88	1.00	0.93	7
24	0.40	0.80	0.53	5
25	0.29	0.67	0.40	3
26	0.00	0.00	0.00	1
27	0.00	0.00	0.00	4
28	0.68	0.94	0.79	18
29	0.76	1.00	0.87	13
30	1.00	1.00	1.00	2
31	1.00	1.00	1.00	18
32	1.00	1.00	1.00	39
33	1.00	1.00	1.00	24
accuracy			0.85	406
macro avg	0.75	0.73	0.71	406
weighted avg	0.87	0.85	0.84	406

Figura 38. Evaluación del modelo FCDNN

Fuente: Elaboración propia

La figura 39 muestra la matriz de confusión para los datos de prueba. Se puede apreciar que, aunque tiene un alto grado de exactitud, el modelo tiene ciertos problemas para diferenciar entre las actividades físicas de trotar (*L2*) y correr (*L3*), por ejemplo.

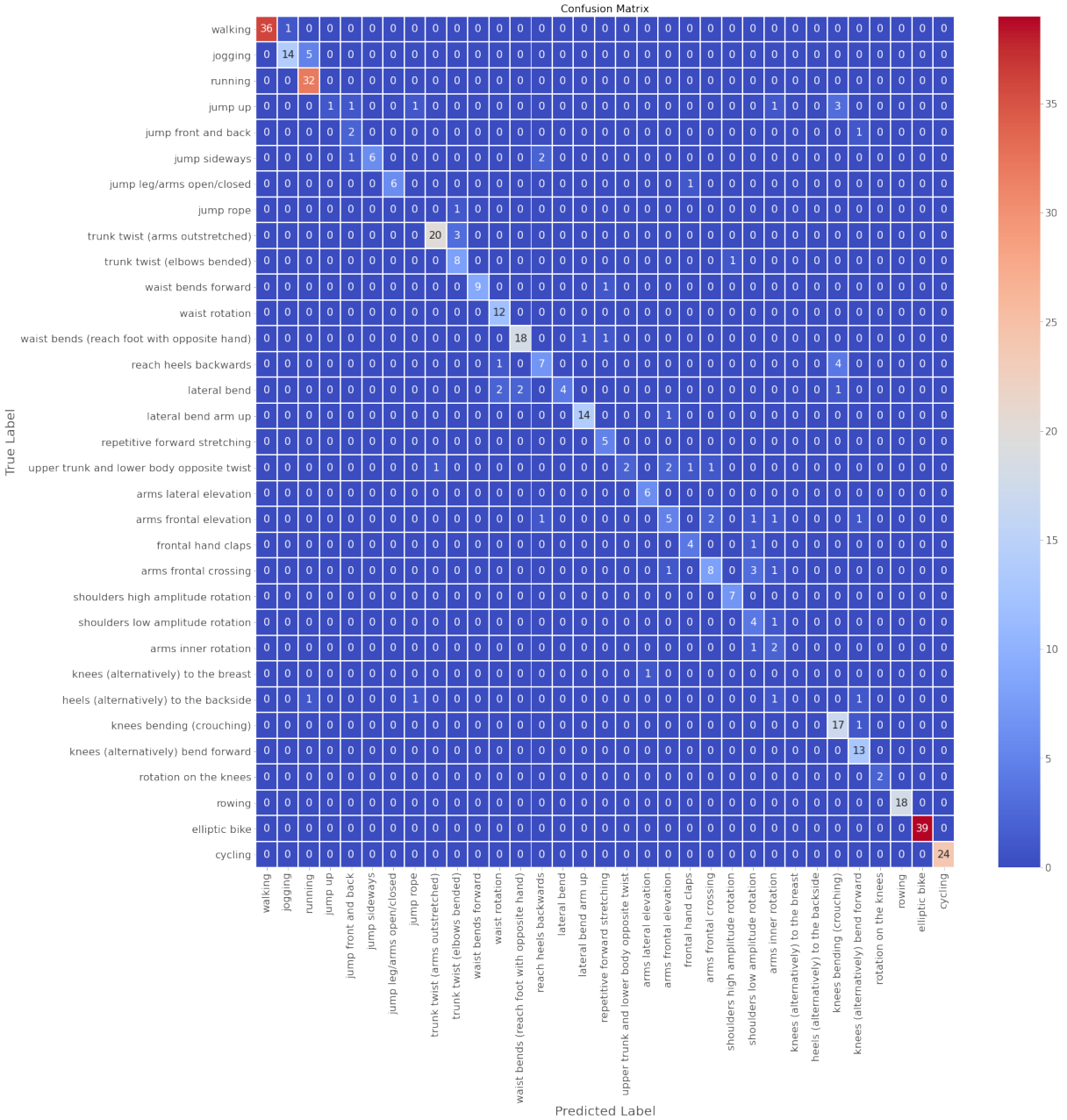

Figura 39. Matriz de confusión para el modelo FCDNN

Fuente: Elaboración propia

5.2. Modelo Long Short-Term Memory

5.2.1. Entrenamiento y validación

La implementación de este modelo de red neuronal también se lleva a cabo utilizando la librería *Keras*. La figura 40 muestra el código completo en Python que implementa el modelo LSTM definido en la sección 4.4.2.

```
[ ] # Implementación del modelo LSTM
    model_lstm = Sequential()
    model_lstm.add(LSTM(units = 128, input_shape = (window_size, FEATURES)))
    model_lstm.add(Dropout(rate = 0.5))
    model_lstm.add(Dense(units = 128, activation = 'relu'))
    model_lstm.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
```

Figura 40. Implementación del modelo LSTM

Fuente: Elaboración propia

La figura 41 muestra la compilación del modelo utilizando la función *compile* de *Keras*. Se especifica Adam como algoritmo de optimización, entropía cruzada categórica como función de pérdida, y la exactitud o *accuracy* como métrica de evaluación durante el entrenamiento.

```
[ ] # Compilación del modelo LSTM
    model_lstm.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Figura 41. Compilación del modelo LSTM

Fuente: Elaboración propia

La tabla 10 muestra el resumen de la arquitectura del modelo, incluyendo el número de parámetros por capa y el número total.

En total el modelo propuesto tiene 146.721 parámetros, todos ajustables durante el proceso de entrenamiento.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 128)	125952
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 33)	4257
Total params: 146,721 **Trainable params: 146,721** **Non-trainable params: 0**		

Tabla 10. Resumen de la arquitectura del modelo LSTM

Fuente: Elaboración propia

En la figura 42 se muestra el entrenamiento para el modelo, estableciendo que se quiere utilizar el 20% de los datos para realizar la validación. De nuevo el número de épocas y el tamaño del lote son hiperparámetros con los que se ha experimentado, dejándolos para este modelo en 20 épocas y 64 elementos por lote.

```
[ ]  # Entrenamiento del modelo LSTM
     history_lstm = model_lstm.fit(x_train_lstm,
                                   y_train_lstm,
                                   batch_size = batch_size,
                                   epochs = epochs,
                                   validation_split = 0.2,
                                   verbose = 1)
```

Figura 42. Entrenamiento del modelo LSTM

Fuente: Elaboración propia

La figura 43 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento. Se aprecia claramente que la red deja de mejorar a partir de la época 9, así que haciendo uso de la técnica de *early stopping*, se puede parar el entrenamiento en ese momento.

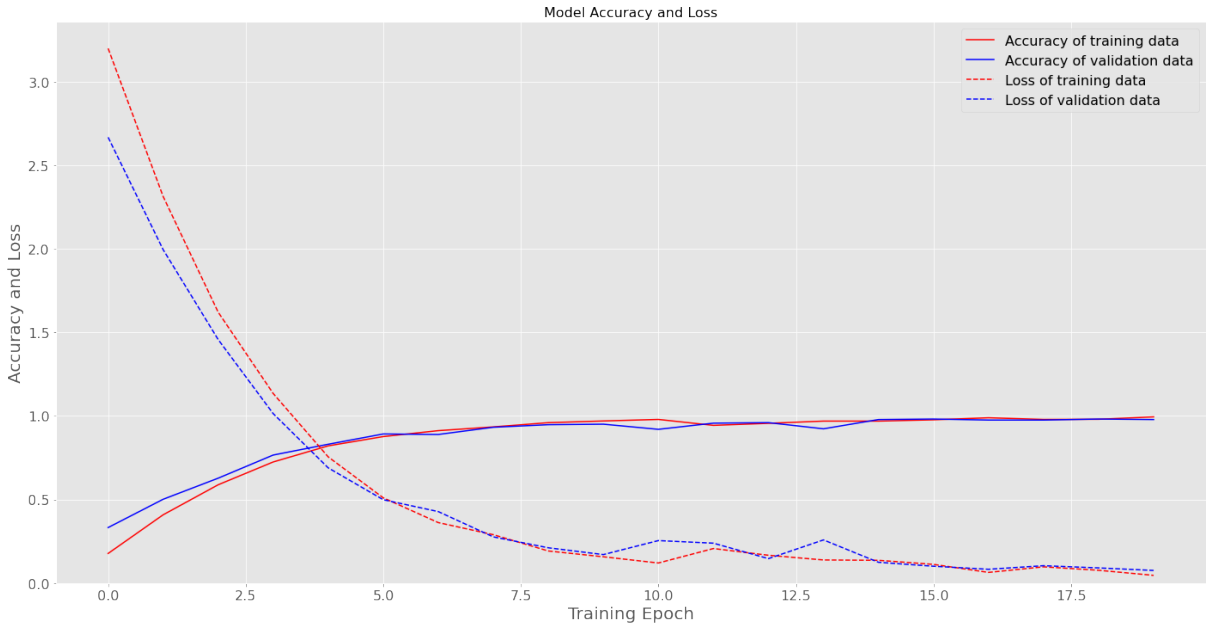

Figura 43. Métricas de *accuracy* y de *loss* en el entrenamiento del modelo LSTM

Fuente: Elaboración propia

5.2.2. Evaluación

Una vez entrenado, se puede utilizar la función *predict* de *Keras* para evaluar cómo se comporta el modelo con los datos de prueba. La figura 44 muestra el resultado de la evaluación del modelo entrenado durante 9 épocas, usando la función *classification_report*. El modelo tiene una exactitud del 96%.

	precision	recall	f1-score	support
1	1.00	0.97	0.99	37
2	0.95	1.00	0.97	19
3	0.97	1.00	0.98	32
4	0.67	0.29	0.40	7
5	0.30	1.00	0.46	3
6	0.88	0.78	0.82	9
7	1.00	1.00	1.00	7
8	1.00	1.00	1.00	1
9	1.00	1.00	1.00	23
10	1.00	1.00	1.00	9
11	1.00	1.00	1.00	10
12	1.00	1.00	1.00	12
13	1.00	1.00	1.00	20
14	0.86	1.00	0.92	12
15	1.00	1.00	1.00	9
16	1.00	1.00	1.00	15
17	1.00	1.00	1.00	5
18	1.00	0.86	0.92	7
19	1.00	1.00	1.00	6
20	1.00	1.00	1.00	11
21	1.00	1.00	1.00	5
22	0.87	1.00	0.93	13
23	1.00	1.00	1.00	7
24	1.00	0.60	0.75	5
25	1.00	1.00	1.00	3
26	1.00	1.00	1.00	1
27	0.00	0.00	0.00	4
28	1.00	1.00	1.00	18
29	1.00	1.00	1.00	13
30	1.00	1.00	1.00	2
31	1.00	1.00	1.00	18
32	1.00	1.00	1.00	39
33	1.00	1.00	1.00	24
accuracy			0.96	406
macro avg	0.92	0.92	0.91	406
weighted avg	0.96	0.96	0.96	406

Figura 44. Evaluación del modelo LSTM

Fuente: Elaboración propia

La figura 45 muestra la matriz de confusión para los datos de prueba. Se puede observar que este modelo reconoce y diferencia perfectamente entre las actividades de andar (*L1*), trotar (*L2*) y correr (*L3*). Sin embargo, tiene problemas con los saltos, confundiendo el salto normal (*L4*) con el salto adelante y atrás (*L5*).

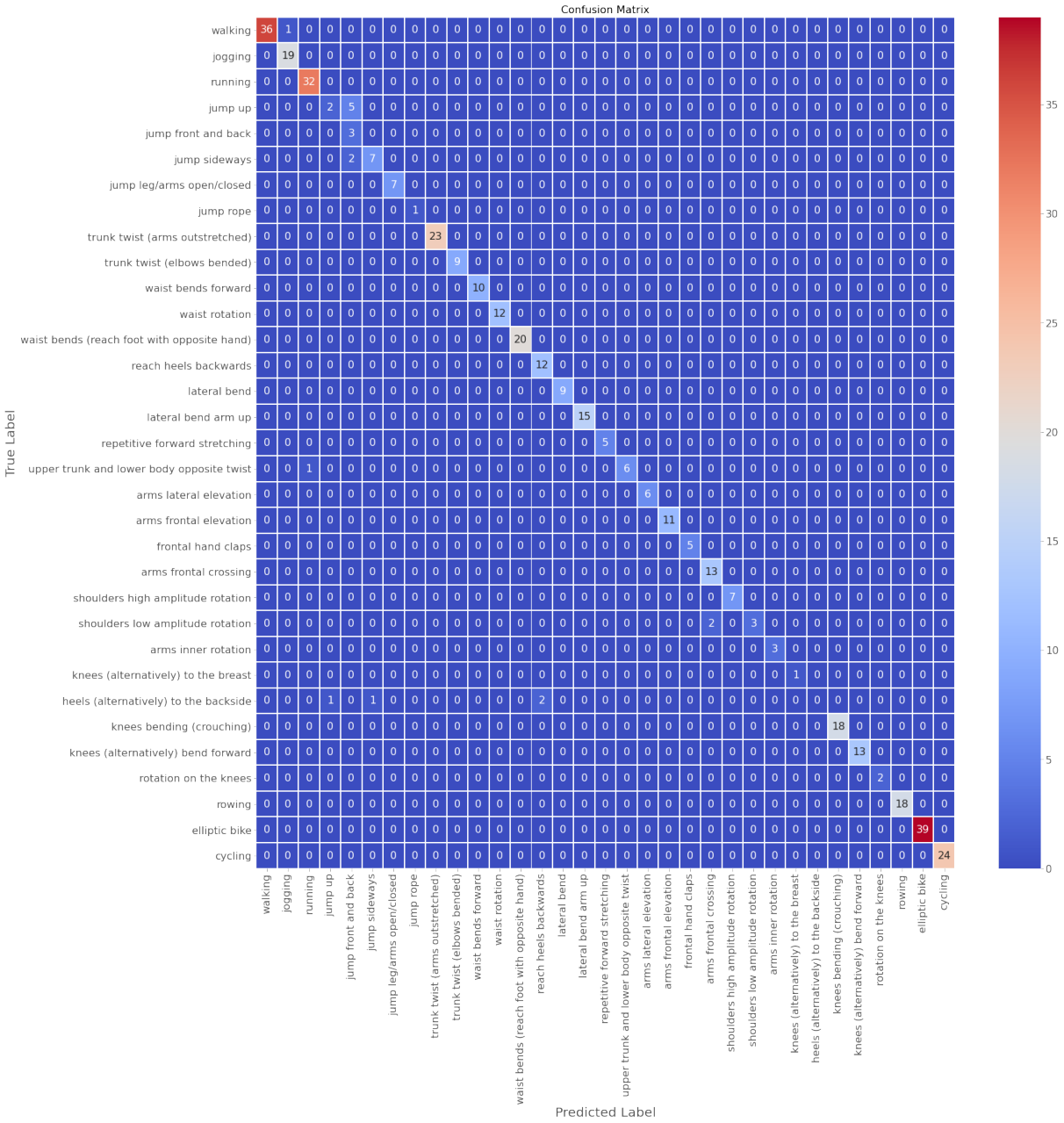

Figura 45. Matriz de confusión para el modelo LSTM

Fuente: Elaboración propia

5.3. Modelo Convolutional Neural Network + Long Short-Term Memory

5.3.1. Entrenamiento y validación

La implementación de este modelo de red neuronal también se lleva a cabo utilizando la librería *Keras*. La figura 46 muestra el código en Python necesario para definir el modelo CNN + LSTM definido en la sección 4.4.3. Las capas se agrupan en dos grandes bloques, que definen la parte convolucional y la parte recurrente, respectivamente.

```
[ ] # Implementación del modelo CNN + LSTM
    model_cnn_lstm = Sequential()
    model_cnn_lstm.add(TimeDistributed(
                          Conv1D(filters = 64, kernel_size = 3, activation = 'relu'),
                          input_shape = (None, block_size, FEATURES)
                      ))
    model_cnn_lstm.add(TimeDistributed(
                          Conv1D(filters = 64, kernel_size = 3, activation = 'relu')
                      ))
    model_cnn_lstm.add(TimeDistributed(
                          Dropout(rate = 0.5)
                      ))
    model_cnn_lstm.add(TimeDistributed(
                          MaxPooling1D(pool_size = 2)
                      ))
    model_cnn_lstm.add(TimeDistributed(
                          Flatten()
                      ))
    model_cnn_lstm.add(LSTM(units = 128))
    model_cnn_lstm.add(Dropout(rate = 0.5))
    model_cnn_lstm.add(Dense(128, activation = 'relu'))
    model_cnn_lstm.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
```

Figura 46. Implementación del modelo CNN + LSTM

Fuente: Elaboración propia

La parte del modelo que representa la CNN está envuelta en una capa *TimeDistributed* específica de *Keras*, lo que permite ir leyendo en bloques cada uno de los grupos en los que ha resultado la segmentación de los datos de entrada. Sigue un patrón estándar, con dos capas convolucionales consecutivas seguidas de una capa de *dropout* y una de *max pooling*. La segunda parte del modelo es una red recurrente como la definida en el modelo LSTM.

La figura 47 muestra la compilación del modelo utilizando la función *compile* de *Keras*. Se especifica Adam como algoritmo de optimización, entropía cruzada categórica como función de pérdida, y la exactitud o *accuracy* como métrica de evaluación durante el entrenamiento.

```
[ ]  # Compilación del modelo CNN + LSTM
     model_cnn_lstm.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Figura 47. Compilación del modelo CNN + LSTM

Fuente: Elaboración propia

La tabla 11 muestra el resumen de la arquitectura del modelo, incluyendo el número de parámetros por capa y el número total.

Layer (type)	Output Shape	Param #
time_distributed_1 (TimeDistributed) *Internamente: conv1d_1 (Conv1D)*	(None, None, 48, 64)	22528
time_distributed_2 (TimeDistributed) *Internamente: conv1d_2 (Conv1D)*	(None, None, 46, 64)	12352
time_distributed_3 (TimeDistributed) *Internamente: dropout_1 (Dropout)*	(None, None, 46, 64)	0
time_distributed_4 (TimeDistributed) *Internamente: max_pooling1d_1 (MaxPooling1D)*	(None, None, 23, 64)	0
time_distributed_5 (TimeDistributed) *Internamente: flatten_1 (Flatten)*	(None, None, 1472)	0
lstm_1 (LSTM)	(None, 128)	819712
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 33)	4257
Total params: 875,361 **Trainable params: 875,361** **Non-trainable params: 0**		

Tabla 11. Resumen de la arquitectura del modelo CNN + LSTM

Fuente: Elaboración propia

En total el modelo propuesto tiene 875.361 parámetros, todos ajustables durante el proceso de entrenamiento.

En la figura 48 se muestra el entrenamiento para el modelo, estableciendo que se quiere utilizar el 20% de los datos para realizar la validación. El número de épocas y el tamaño del lote son hiperparámetros con los que se ha experimentado, dejándolos en sus valores definitivos de 14 épocas y 64 elementos por lote, para este modelo.

```
[ ]  # Entrenamiento del modelo CNN + LSTM
     history_cnn_lstm = model_cnn_lstm.fit(x_train_cnn_lstm,
                                           y_train_cnn_lstm,
                                           batch_size = batch_size,
                                           epochs = epochs,
                                           validation_split = 0.2,
                                           verbose = 1)
```

Figura 48. Entrenamiento del modelo CNN + LSTM

Fuente: Elaboración propia

La figura 49 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento. Se aprecia claramente que la red deja de mejorar a partir de la época 9, así que haciendo uso de la técnica de *early stopping* se puede parar el entrenamiento.

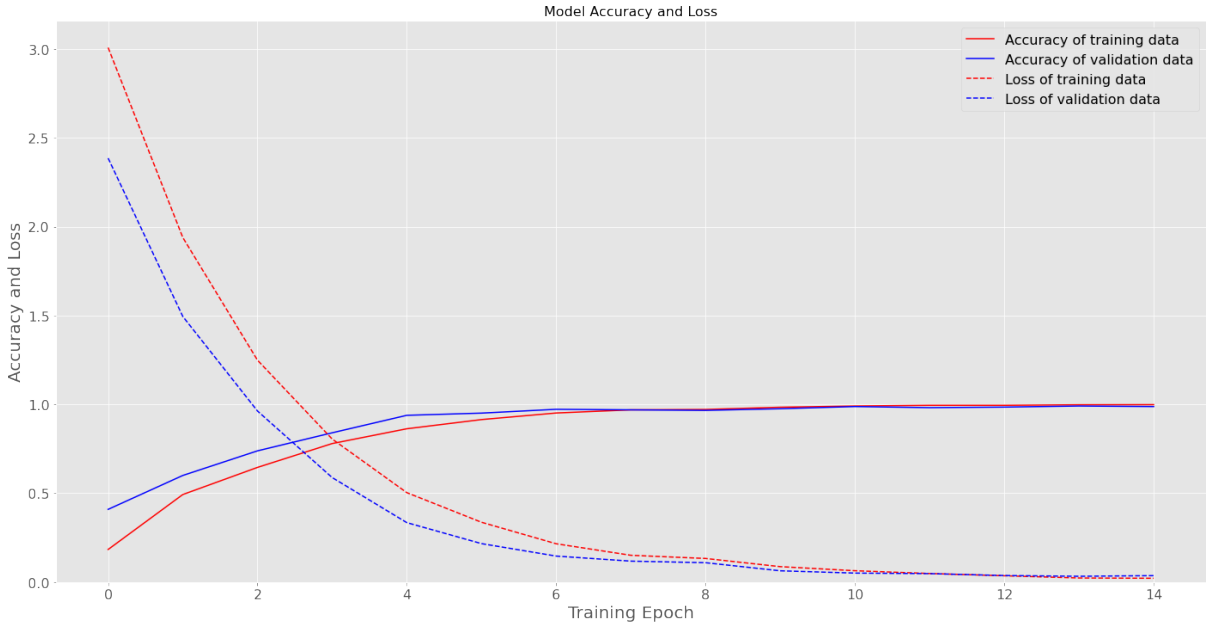

Figura 49. Métricas de *accuracy* y de *loss* en el entrenamiento del modelo CNN + LSTM

Fuente: Elaboración propia

5.3.2. Evaluación

Una vez entrenado, se puede utilizar la función *predict* de *Keras* para evaluar cómo se comporta el modelo con los datos de prueba. La figura 50 muestra el resultado de la evaluación del modelo entrenado durante 9 épocas, usando la función *classification_report*. El modelo tiene una exactitud del 97%.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	37
2	1.00	1.00	1.00	19
3	1.00	1.00	1.00	32
4	1.00	0.14	0.25	7
5	0.21	1.00	0.35	3
6	1.00	0.89	0.94	9
7	1.00	1.00	1.00	7
8	1.00	1.00	1.00	1
9	1.00	1.00	1.00	23
10	1.00	1.00	1.00	9
11	1.00	1.00	1.00	10
12	1.00	1.00	1.00	12
13	1.00	1.00	1.00	20
14	1.00	1.00	1.00	12
15	1.00	1.00	1.00	9
16	1.00	1.00	1.00	15
17	1.00	1.00	1.00	5
18	1.00	1.00	1.00	7
19	1.00	1.00	1.00	6
20	1.00	1.00	1.00	11
21	1.00	1.00	1.00	5
22	1.00	1.00	1.00	13
23	1.00	1.00	1.00	7
24	1.00	1.00	1.00	5
25	1.00	1.00	1.00	3
26	1.00	1.00	1.00	1
27	0.00	0.00	0.00	4
28	1.00	1.00	1.00	18
29	1.00	1.00	1.00	13
30	1.00	1.00	1.00	2
31	1.00	1.00	1.00	18
32	1.00	1.00	1.00	39
33	1.00	1.00	1.00	24
accuracy			0.97	406
macro avg	0.95	0.94	0.93	406
weighted avg	0.98	0.97	0.97	406

Figura 50. Evaluación del modelo CNN + LSTM

Fuente: Elaboración propia

La figura 51 muestra la matriz de confusión para los datos de prueba. Se puede observar que este modelo reconoce y diferencia perfectamente entre las actividades de andar (*L1*), trotar (*L2*) y correr (*L3*). Pero al igual que el modelo LSTM, tiene problemas con los saltos, confundiendo el salto normal (*L4*) con el salto adelante y atrás (*L5*). También tiene dificultad a la hora de reconocer la actividad de tocar el glúteo con los talones (*L27*), que confunde con los saltos adelante y atrás (*L5*).

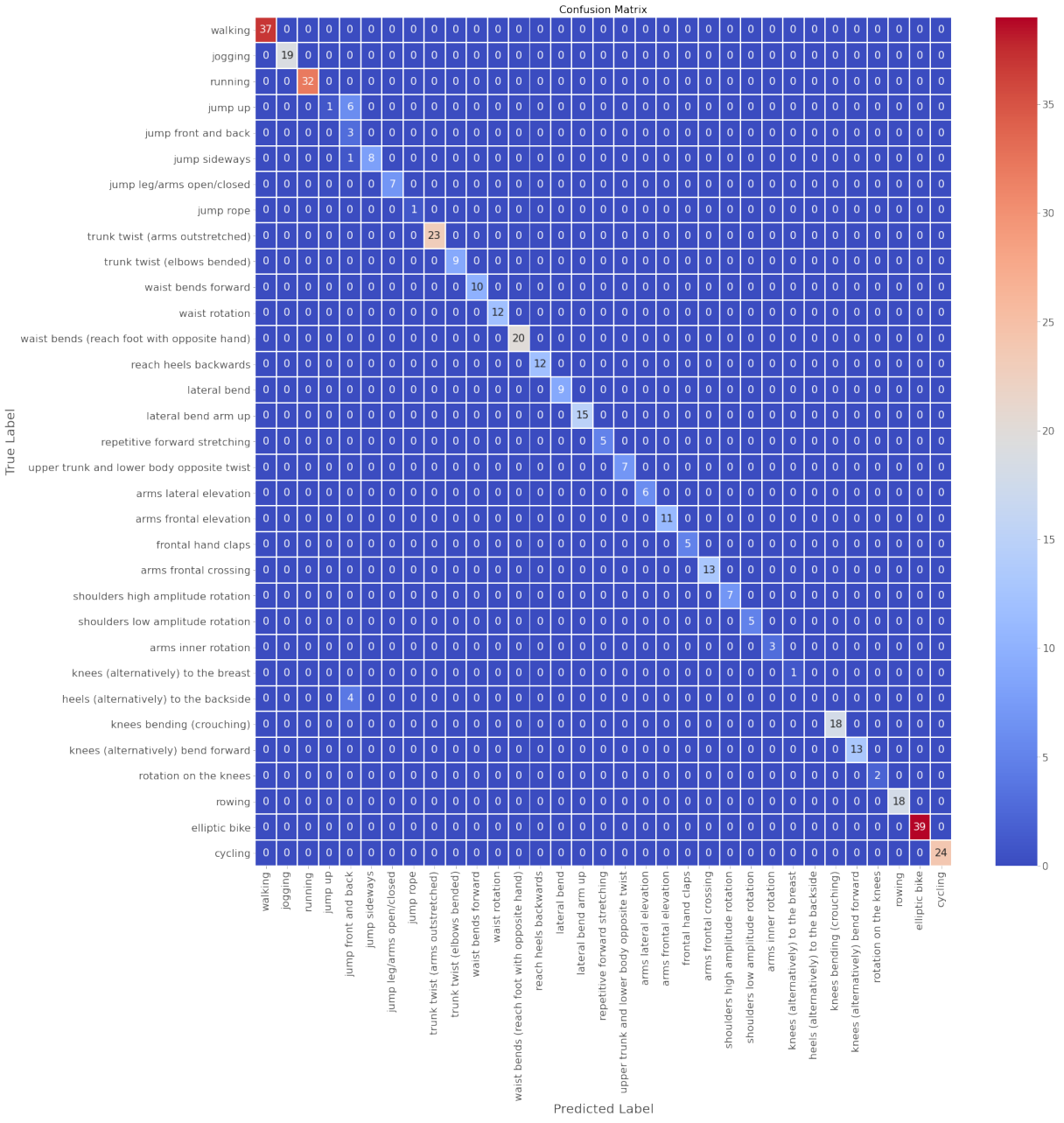

Figura 51. Matriz de confusión para el modelo CNN + LSTM

Fuente: Elaboración propia

5.4. Modelo Convolutional Long Short-Term Memory

5.4.1. Entrenamiento y validación

Este último modelo de nuevo aprovecha las capacidades de *Keras* para definir redes neuronales de alto nivel. La figura 52 muestra el código en Python que implementa el modelo ConvLSTM definido en la sección 4.4.4. Para ello, se utiliza la clase *ConvLSTM2D*, que admite tanto datos en dos dimensiones como series temporales multivariadas unidimensionales.

```
[ ] # Implementación del modelo ConvLSTM
    model_convlstm = Sequential()
    model_convlstm.add(ConvLSTM2D(
                          filters = 64,
                          kernel_size = (1, 3),
                          activation = 'relu',
                          input_shape = (num_blocks, 1, block_size, FEATURES)))
    model_convlstm.add(Dropout(rate = 0.5))
    model_convlstm.add(Flatten())
    model_convlstm.add(Dense(units = 128, activation = 'relu'))
    model_convlstm.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
```

Figura 52. Implementación del modelo ConvLSTM

Fuente: Elaboración propia

La figura 53 muestra la compilación del modelo utilizando la función *compile* de Keras. Se especifica Adam como algoritmo de optimización, entropía cruzada categórica como función de pérdida, y la exactitud o *accuracy* como métrica de evaluación durante el entrenamiento.

```
[ ] # Compilación del modelo ConvLSTM
    model_convlstm.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Figura 53. Compilación del modelo ConvLSTM

Fuente: Elaboración propia

La tabla 12 muestra el resumen de la arquitectura del modelo, incluyendo el número de parámetros por capa y el número total.

Layer (type)	Output Shape	Param #
conv_lst_m2d_1 (ConvLSTM2D)	(None, 1, 48, 64)	139264
dropout_1 (Dropout)	(None, 1, 48, 64)	0
flatten_1 (Flatten)	(None, 3072)	0
dense_1 (Dense)	(None, 128)	393344
dense_2 (Dense)	(None, 33)	4257
Total params: 536,865 **Trainable params: 536,865** **Non-trainable params: 0**		

Tabla 12. Resumen de la arquitectura del modelo ConvLSTM

Fuente: Elaboración propia

En total el modelo propuesto tiene 536.865 parámetros, todos ajustables durante el proceso de entrenamiento.

En la figura 54 se muestra el entrenamiento para el modelo, estableciendo que se quiere utilizar el 20% de los datos para realizar la validación. El número de épocas y el tamaño del lote son hiperparámetros con los que se ha experimentado, dejándolos en sus valores definitivos (para este modelo) de 9 épocas y 64 elementos por lote.

```
[ ] # Entrenamiento del modelo ConvLSTM
    history_convlstm = model_convlstm.fit(x_train_convlstm,
                                          y_train_convlstm,
                                          batch_size = batch_size,
                                          epochs = epochs,
                                          validation_split = 0.2,
                                          verbose = 1)
```

Figura 54. Entrenamiento del modelo ConvLSTM

Fuente: Elaboración propia

La figura 55 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento. Se aprecia que la red deja de mejorar a partir de la época 4, así que haciendo uso de la técnica de *early stopping*, se puede parar el entrenamiento.

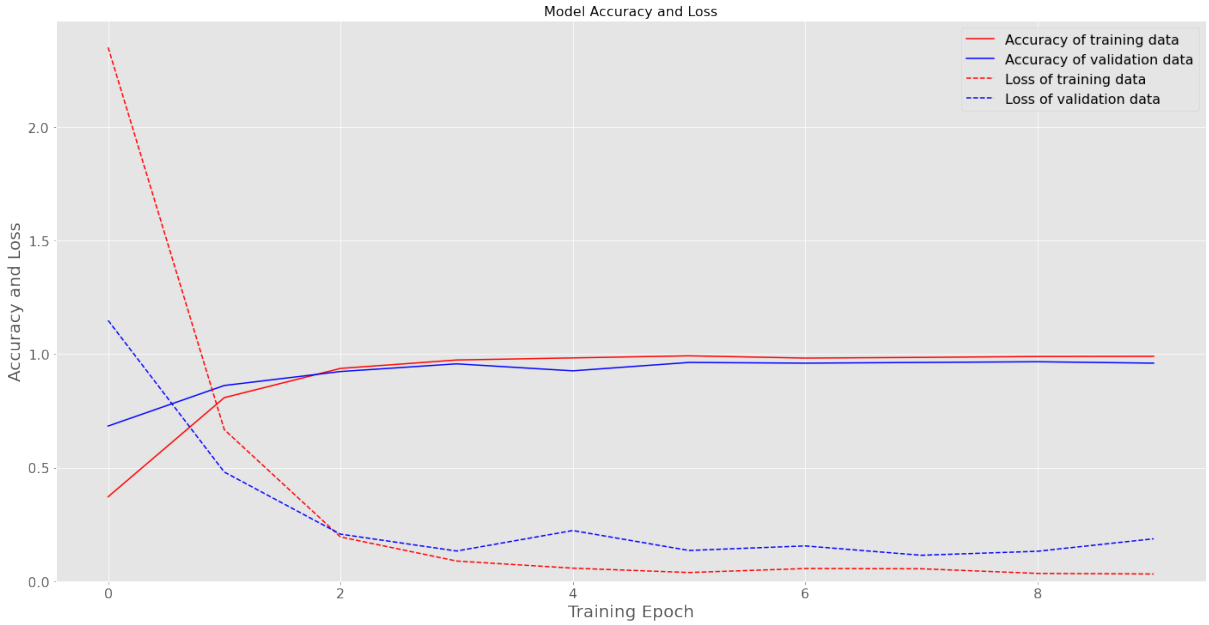

Figura 55. Métricas de *accuracy* y de *loss* en el entrenamiento del modelo ConvLSTM

Fuente: Elaboración propia

5.4.2. Evaluación

Una vez entrenado, se utiliza la función *predict* de *Keras* para evaluar cómo se comporta el modelo con los datos de prueba. La figura 56 muestra el resultado de la evaluación del modelo entrenado durante 4 épocas, usando la función *classification_report*. El modelo tiene una exactitud del 97%.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	37
2	1.00	1.00	1.00	19
3	1.00	1.00	1.00	32
4	1.00	0.14	0.25	7
5	0.25	0.67	0.36	3
6	0.89	0.89	0.89	9
7	1.00	1.00	1.00	7
8	0.50	1.00	0.67	1
9	1.00	1.00	1.00	23
10	1.00	1.00	1.00	9
11	1.00	1.00	1.00	10
12	0.92	1.00	0.96	12
13	1.00	1.00	1.00	20
14	1.00	1.00	1.00	12
15	1.00	1.00	1.00	9
16	1.00	1.00	1.00	15
17	1.00	1.00	1.00	5
18	1.00	1.00	1.00	7
19	1.00	1.00	1.00	6
20	1.00	1.00	1.00	11
21	1.00	1.00	1.00	5
22	1.00	1.00	1.00	13
23	1.00	1.00	1.00	7
24	1.00	1.00	1.00	5
25	1.00	1.00	1.00	3
26	0.50	1.00	0.67	1
27	0.00	0.00	0.00	4
28	0.95	1.00	0.97	18
29	0.93	1.00	0.96	13
30	1.00	1.00	1.00	2
31	1.00	1.00	1.00	18
32	1.00	1.00	1.00	39
33	1.00	1.00	1.00	24
accuracy			0.97	406
macro avg	0.91	0.93	0.90	406
weighted avg	0.97	0.97	0.96	406

Figura 56. Evaluación del modelo ConvLSTM

Fuente: Elaboración propia

La figura 57 muestra la matriz de confusión para los datos de prueba. Se observa que este modelo básicamente sólo tiene problemas con los saltos, confundiendo el salto normal (*L4*) con el salto adelante y atrás (*L5*).

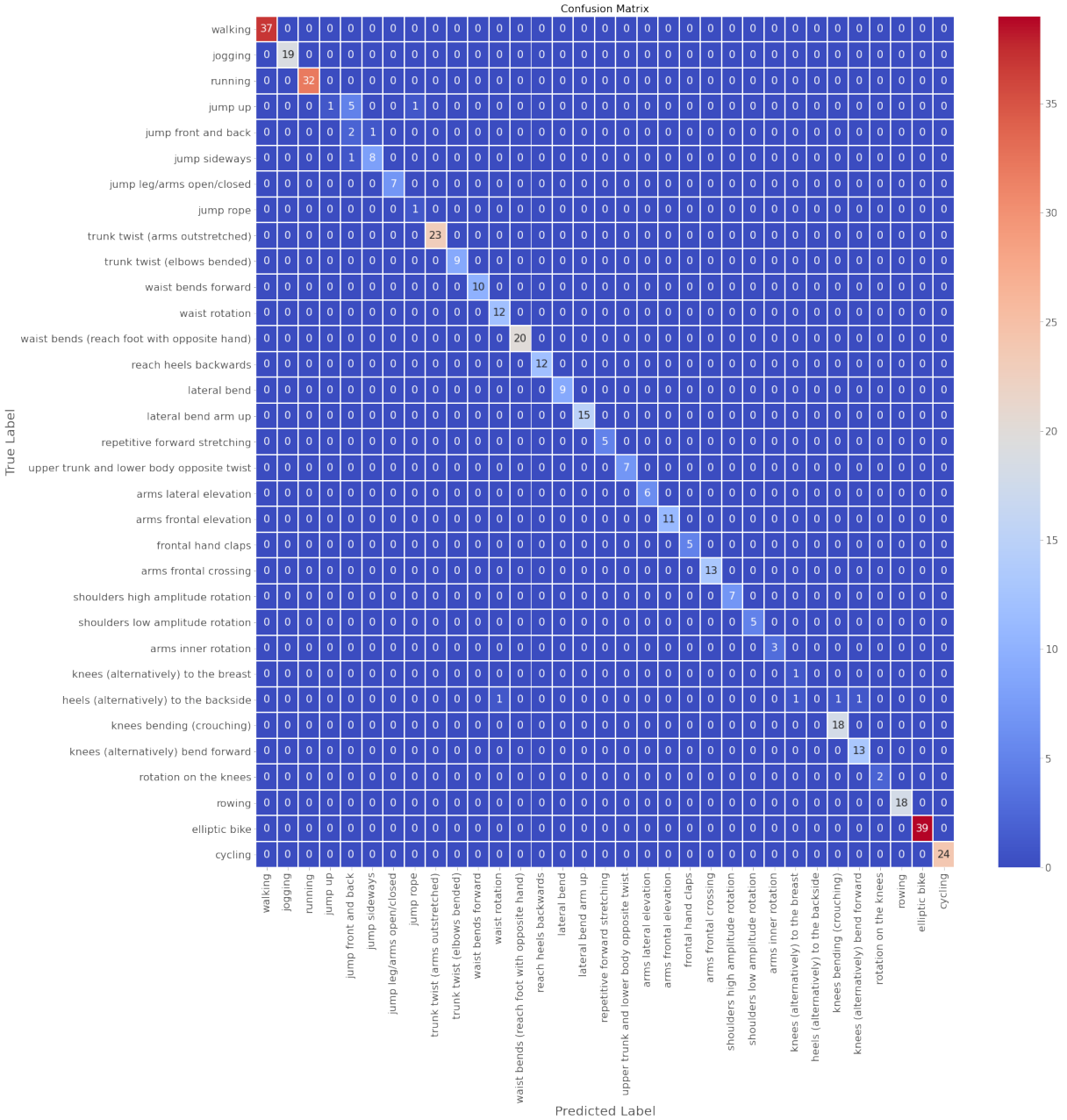

Figura 57. Matriz de confusión para el modelo ConvLSTM

Fuente: Elaboración propia

5.5. Evaluación de los modelos

En la tabla 13 se muestran las principales métricas usadas para evaluar el funcionamiento de los modelos propuestos. Estos datos se han obtenido entrenando y evaluando los modelos múltiples veces, ya que las redes neuronales son modelos estocásticos, por lo que no siempre se obtienen los mismos resultados utilizando los mismos datos. Se ha hecho uso de la función *evaluate_model* de *Keras* para obtener la media y la desviación típica del conjunto de evaluaciones.

	FCDNN	LSTM	CNN + LSTM	ConvLSTM
precision (media ponderada)	86,% ± 3,544	96,% ± 2,287	97,% ± 1,992	96,% ± 2,113
recall (media ponderada)	84,% ± 2,887	95,% ± 2,893	96,% ± 2,361	95,% ± 2,576
f1-score (media ponderada)	83,% ± 3,021	95,% ± 3,229	96,% ± 2,843	95,% ± 2,884
accuracy	**84,241% ± 2,332**	**94,632% ± 3,121**	**96,827% ± 2,901**	**97,156% ± 2,645**

Tabla 13. Métricas de los modelos evaluados

Fuente: Elaboración propia

La complejidad de los modelos de red neuronal propuestos puede evaluarse principalmente a partir del número de parámetros que tienen y del tiempo requerido para su entrenamiento. La tabla 14 muestra esta información para cada propuesta.

	FCDNN	LSTM	CNN + LSTM	ConvLSTM
Número de parámetros	3.005.729	146.721	875.361	536.865
Número de épocas	4	9	9	9
Tiempo de entrenamiento (en milisegundos)	1,163 ms	27.045 ms	1.005,910 ms	10.010 ms

Tabla 14. Complejidad de los modelos evaluados

Fuente: Elaboración propia

Los resultados condensados en las tablas de esta sección están presentados de manera objetiva. El análisis y discusión de los mismos se realiza en el capítulo 6.

6. Discusión

En el capítulo 5 se ha presentado la implementación y evaluación de cuatro modelos de redes neuronales profundas para resolver el problema del reconocimiento de la actividad humana. Los resultados muestran que todos los modelos de red son capaces de resolver el problema del HAR.

El modelo FCDNN inicial es el que peor se comporta, con una precisión del 84,241%. Además, es el modelo más complejo atendiendo al número de parámetros, necesitando 3.005.729 parámetros ajustables. Aun así, proporciona un reconocimiento de actividades notable, aunque con algunas dificultades a la hora de distinguir ejercicios con una ejecución muy parecida, como es el caso de andar, trotar y correr.

El modelo de red LSTM, al ser un tipo de red neuronal recurrente, se adapta mejor a problemas en los que la entrada de datos es una serie temporal. Esto se aprecia en la gran mejora que proporciona, llegando a un 94,632% de precisión. También es varios órdenes de magnitud menos compleja a nivel de parametrización, ya que sólo tiene 146.721 parámetros. De los cuatro modelos propuestos, es la que tiene menos parámetros ajustables, con diferencia. Al alcanzar este nivel de precisión, desaparecen los errores al etiquetar incorrectamente actividades muy similares, como sucedía en el caso de la red FCDNN. Este modelo reconoce y diferencia perfectamente entre las actividades de andar, trotar y correr. Sin embargo, tiene problemas con los saltos, confundiendo el salto normal con el salto adelante y atrás, por ejemplo.

La red neuronal CNN + LSTM mejora más la precisión, llegando al 96,827%. Esto se debe principalmente a que utiliza dos redes neuronales en serie, una tras otra. Primero la parte convolucional de la red se encarga de la extracción de las características y posteriormente la parte recurrente se encarga de hacer la predicción. Esta mejora en la precisión tiene como contrapartida el aumento de la complejidad de la red, que está compuesta de 875.361 parámetros. Este modelo, aunque mejora el reconocimiento de actividades respecto al modelo LSTM, sigue teniendo problemas con los saltos. Por ejemplo, no es capaz de distinguir perfectamente entre el salto normal y el salto adelante y atrás.

El modelo ConvLSTM evoluciona el concepto de CNN + LSTM, realizando los procesos de la red LSTM dentro de la propia convolución. Pero a pesar de su teórica superioridad, en el caso del problema del HAR no parece marcar una gran diferencia en cuanto a precisión. El modelo alcanza una precisión del 97,156%, que no es aumento excesivo respecto a la red CNN + LSTM. Donde sí se aprecia una mejora es en la complejidad, ya que sólo tiene 536.865

parámetros. Los problemas de este modelo a la hora de reconocer actividades, prácticamente se limitan a diferenciar en algunos casos el salto normal con el salto adelante y atrás, como sucede con la red CNN + LSTM.

El comportamiento de estos modelos es muy dependiente de las condiciones en las que se evalúan. Además de la evaluación realizada y detallada en el capítulo 5, se han realizado diferentes experimentos, cambiando los hiperparámetros de los modelos, algo que obviamente afecta a su precisión.

Por otra parte, también hay que tener en cuenta que las arquitecturas propuestas se han evaluado realizando una segmentación concreta: segmentos de 4 segundos sin solapamiento. La segmentación es un elemento crucial para cualquier problema que trate con señales que evolucionan en el tiempo, y especialmente en el caso del reconocimiento de actividades humanas.

7. Conclusiones y trabajo futuro

Este último capítulo presenta el resumen final de este trabajo de fin de máster. En primer lugar se presentan las conclusiones, relacionándolas con los objetivos planteados en el capítulo 3. A continuación, se comentan las posibles líneas de trabajo futuro que pueden aportar un valor añadido.

7.1. Conclusiones

El reconocimiento de la actividad que está realizando una persona, implica predecir el tipo de movimiento que está efectuando basándose en los datos que proporciona un conjunto de sensores. Los datos del sensor pueden registrarse de forma remota, por métodos inalámbricos como Bluetooth o tecnología wifi. Alternativamente, la información de los sensores se puede almacenar directamente en un dispositivo, como sucede por ejemplo con los teléfonos inteligentes o las pulseras cuantificadoras.

Históricamente el HAR se ha tratado como un problema de reconocimiento de patrones, estrategia que ha avanzado en gran medida gracias a la adopción de algoritmos de aprendizaje automático como árboles de decisión, máquinas de vector de soporte, clasificadores bayesianos y modelos ocultos de Márkov. Sin embargo, este tipo de soluciones basadas en reconocimiento de patrones, aunque pueden ser satisfactorias en algunos escenarios concretos, requieren de una extracción heurística y manual de las características. Esto tiene una serie de desventajas:

- Se necesita mucho conocimiento y experiencia en el dominio, para realizar una extracción de características adecuada.
- Se aprenden características muy superficiales, lo que sólo permite reconocer actividades de bajo nivel y hace difícil reconocer actividades de alto nivel que requieren de contexto.
- Es necesaria una gran cantidad de datos correctamente etiquetados para poder entrenar los modelos.
- Son métodos pensados para datos estáticos, mientras que en la vida real las actividades generan un flujo continuo de datos.

Para intentar evitar todas estas desventajas, se ha propuesto como objetivo principal de este trabajo el aplicar técnicas de aprendizaje profundo para resolver el problema del HAR. Este objetivo general se ha descompuesto en un conjunto de objetivos más específicos, abordables y analizables por separado

El primer objetivo específico ha sido explorar los distintos repositorios de datos públicos con información relacionada con el HAR. Una vez descartada la opción de realizar una recolección de datos propios, se ha realizado un estudio de los distintos repositorios accesibles con información sobre el reconocimiento de actividades. Se ha decidido hacer uso del conjunto de datos REALDISP, ya que contempla un gran número de actividades físicas, recoge información de un número importante de sujetos participantes y, además, los sensores utilizados para realizar las mediciones aportan mucha información.

El segundo objetivo específico ha sido seleccionar qué métodos de aprendizaje profundo aplicar a la resolución del problema del HAR. Una vez descartadas otras arquitecturas en el campo del aprendizaje profundo, como las redes de creencia o el aprendizaje por refuerzo, se han propuesto métodos centrados en el ámbito de las redes neuronales profundas convolucionales y recurrentes.

Como tercer objetivo específico, se han diseñado e implementado varias redes neuronales con diferentes topologías, para ser entrenadas utilizando los datos del conjunto REALDISP. En concreto, se han escogido las siguientes cuatro arquitecturas de red:

- Fully Connected Deep Neural Network.
- Long Short-Term Memory.
- Convolutional Neural Network + Long Short-Term Memory.
- Convolutional Long Short-Term Memory.

El cuarto y último objetivo específico ha sido realizar la evaluación y comparación de las distintas soluciones propuestas. Se han calculado las principales métricas empleadas en la evaluación de algoritmos de clasificación en aprendizaje supervisado. Los resultados obtenidos muestran que todos los modelos de red son capaces de resolver el problema con solvencia.

El modelo FCDNN es el que peor resultado obtiene, con una precisión del 84,241%. Además, es el modelo más complejo, con 3.005.729 parámetros.

El modelo LSTM, al tener una arquitectura recurrente y ser más adecuada para problemas con series de datos temporales, aumenta la precisión en gran medida respecto al modelo FCDNN, llegando al 94,632%. También es mucho menos complejo a nivel de parametrización, ya que sólo tiene 146.721 parámetros.

La arquitectura CNN + LSTM mejora ligeramente los resultados, al hacer uso de capas convolucionales para alimentar la subred LSTM. Alcanza el 96,827% de precisión. Como contrapartida aumenta la complejidad de la red, que está compuesta de 875.361 parámetros.

El modelo ConvLSTM, que realiza los procesos de la red recurrente dentro de la propia convolución, no supone un avance significativo. Obtiene una precisión del 97,156%. Donde sí se aprecia una mejora importante respecto a CNN + LSTM es en la complejidad, puesto que tiene 536.865 parámetros.

Una vez estudiado el problema del reconocimiento de las actividades humanas, se llega a la conclusión de que, aún siendo un problema ampliamente estudiado mediante técnicas de aprendizaje automático, queda margen de mejora utilizando métodos de aprendizaje profundo. El avance en la extracción de características que supone utilizar técnicas de aprendizaje profundo hace que sea menos relevante tener un gran conocimiento del dominio de los datos. Además, se pueden aprender características de alto nivel, lo que facilita el reconocimiento de actividades más complejas.

También se concluye que no hay un único modelo de aprendizaje profundo adecuado para resolver el problema del HAR. Tras la evaluación de los modelos de red propuestos en este trabajo, se observa que las redes recurrentes gestionan mejor las series temporales de datos y, por lo tanto, tienen mayor precisión a la hora de reconocer las actividades. Por otra parte, se ha visto que las redes convolucionales mejoran la extracción de características y que pueden potenciar la capacidad expresiva de las RNNs.

7.2. Líneas de trabajo futuro

La evaluación de las técnicas de aprendizaje profundo propuestas e implementadas en este trabajo, se ha realizado estableciendo una serie de condiciones que acotan su alcance. Los siguientes son ejemplos de posibles trabajos posteriores para profundizar con estas técnicas en la resolución del problema del HAR:

- Utilizar los datos de todos los sujetos de REALDISP: Este trabajo se ha limitado a usar los datos de 5 sujetos que no contenían información corrupta para el escenario de

colocación ideal (ayudando también a no sobrepasar los límites de memoria y tiempo de ejecución establecidos por el entorno de Google Colab).

- Seleccionar sólo determinadas actividades: Se puede investigar el diseño de modelos más sencillos, que reconozcan un pequeño conjunto de actividades, para implantarse en dispositivos como pulseras de cuantificación.

- Hacer uso de los escenarios de desplazamiento gradual de los sensores: Este trabajo se ha centrado en las condiciones ideales de posicionamiento, pero las técnicas desarrolladas pueden tener aplicación en el escenario de auto-posicionamiento definido por el conjunto de datos REALDISP.

- Se abre la puerta a realizar investigaciones utilizando sólo determinados sensores o ciertas variables de los mismos: Por ejemplo, se pueden plantear modelos más sencillos utilizando sólo la estimación de la orientación que ofrecen los cuaterniones.

- Experimentar con la parametrización de la ventana deslizante. El proceso de segmentación es muy importante en problemas que tratan con señales temporales de datos, por lo que el tamaño de la ventana deslizante y su desplazamiento juegan un papel fundamental en los resultados obtenidos por los modelos de red.

Adicionalmente, este trabajo puede ser una base sobre la que aplicar conceptos más avanzados de reconocimiento de la actividad humana utilizando aprendizaje profundo, como la técnica *Iss2Image* (Hur, Bang, Lee, et al., 2018).

8. Bibliografía

Almaslukh, B., AlMuhtadi, J., & Artoli, A. (2017). *An effective deep autoencoder approach for online smartphone-based human activity recognition*. Int. J. Comput. Sci. Netw. Secur, 17(4), 160–165.

Alsheikh, M. A., Selim, A., Niyato, D., Doyle, L., Lin, S., & Tan, H.-P. (2016). *Deep activity recognition models with triaxial accelerometers*. Workshops at the Thirtieth AAAI Conference on Artificial Intelligence.

Baldominos, A., Cervantes, A., Saez, Y., & Isasi, P. (2019). *A comparison of machine learning and deep learning techniques for activity recognition using mobile devices*. Sensors, 19(3), 521.

Banos, O. (s. f.). *REALDISP dataset*. Datasets. Recuperado 19 de abril de 2020, de http://orestibanos.com/datasets.htm

Banos, O., Galvez, J.-M., Damas, M., Pomares, H., & Rojas, I. (2014). *Window size impact in human activity recognition*. Sensors, 14(4), 6474–6499.

Banos, O., & Toth, M. A. (2014). *Realistic sensor displacement benchmark dataset*. 4.

Banos, O., Toth, M. A., Damas, M., Pomares, H., & Rojas, I. (2014). *Dealing with the effects of sensor displacement in wearable activity recognition*. Sensors, 14(6), 9995–10023.

Banos, O., Villalonga, C., Garcia, R., Saez, A., Damas, M., Holgado-Terriza, J. A., Lee, S., Pomares, H., & Rojas, I. (2015). *Design, implementation and validation of a novel open framework for agile development of mobile health applications*. Biomedical engineering online, 14(S2), S6.

Baños, O., Damas, M., Pomares, H., Rojas, I., Tóth, M. A., & Amft, O. (2012). *A benchmark dataset to evaluate sensor displacement in activity recognition*. Proceedings of the 2012 ACM Conference on Ubiquitous Computing, 1026–1035.

Bengio, Y. (2013). *Deep learning of representations: Looking forward*. International Conference on Statistical Language and Speech Processing, 1–37.

Bulling, A., Blanke, U., & Schiele, B. (2014). *A tutorial on human activity recognition using body-worn inertial sensors*. ACM Computing Surveys (CSUR), 46(3), 1–33.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). *Long-term recurrent convolutional networks for visual recognition and description*. Proceedings of the IEEE conference on computer vision and pattern recognition, 2625–2634.

Edel, M., & Köppe, E. (2016). *Binarized-blstm-rnn based human activity recognition*. 2016 International conference on indoor positioning and indoor navigation (IPIN), 1–7.

Guan, Y., & Plötz, T. (2017). *Ensembles of deep lstm learners for activity recognition using wearables*. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 1(2), 1–28.

Hammerla, N. Y., Halloran, S., & Plötz, T. (2016). *Deep, convolutional, and recurrent models for human activity recognition using wearables*. arXiv preprint arXiv:1604.08880.

Hannink, J., Kautz, T., Pasluosta, C. F., Gaßmann, K.-G., Klucken, J., & Eskofier, B. M. (2016). *Sensor-based gait parameter extraction with deep convolutional neural networks*. IEEE journal of biomedical and health informatics, 21(1), 85–93.

Hawkins, D. M. (2004). *The problem of overfitting. Journal of chemical information and computer sciences*, 44(1), 1–12.

Hayashi, T., Nishida, M., Kitaoka, N., & Takeda, K. (2015). *Daily activity recognition based on DNN using environmental sound and acceleration signals*. 2015 23rd European Signal Processing Conference (EUSIPCO), 2306–2310.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). *Gans trained by a two time-scale update rule converge to a local nash equilibrium*. Advances in neural information processing systems, 6626–6637.

Hsu, H.-H., Chang, C.-Y., & Hsu, C.-H. (2017). *Big Data Analytics for Sensor-Network Collected Intelligence*. Morgan Kaufmann.

Hur, T., Bang, J., Lee, J., Kim, J.-I., Lee, S., & others. (2018). *Iss2Image: A novel signal-encoding technique for CNN-based human activity recognition*. Sensors, 18(11), 3910.

IBM. (2016). *Guía de CRISP-DM de IBM SPSS Modeler*. 52.

Inoue, M., Inoue, S., & Nishida, T. (2018). *Deep recurrent neural network for mobile human activity recognition with high throughput*. Artificial Life and Robotics, 23(2), 173–185.

Jiang, W., & Yin, Z. (2015). *Human activity recognition using wearable sensors by deep convolutional neural networks*. Proceedings of the 23rd ACM international conference on Multimedia, 1307–1310.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

Laguna, J. O., Olaya, A. G., & Borrajo, D. (2011). *A dynamic sliding window approach for activity recognition*. International Conference on User Modeling, Adaptation, and Personalization, 219–230.

Lara, O. D., & Labrador, M. A. (2012). *A survey on human activity recognition using wearable sensors*. IEEE communications surveys & tutorials, 15(3), 1192–1209.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). *A simple way to initialize recurrent networks of rectified linear units*. arXiv preprint arXiv:1504.00941.

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 2018). Determination press San Francisco, CA.

Ordóñez, F. J., & Roggen, D. (2016). *Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition*. Sensors, 16(1), 115.

Overfitting and underfitting. (s. f.). *Educative: Interactive Courses for Software Developers.* Recuperado 2 de junio de 2020, de https://www.educative.io/edpresso/overfitting-and-underfitting

Panwar, M., Dyuthi, S. R., Prakash, K. C., Biswas, D., Acharyya, A., Maharatna, K., Gautam, A., & Naik, G. R. (2017). *CNN based approach for activity recognition using a wrist-worn accelerometer*. 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2438–2441.

Plötz, T., Hammerla, N. Y., & Olivier, P. L. (2011). *Feature learning for activity recognition in ubiquitous computing*. Twenty-second international joint conference on artificial intelligence.

Potdar, K., Pardawala, T. S., & Pai, C. D. (2017). *A comparative study of categorical variable encoding techniques for neural network classifiers*. International journal of computer applications, 175(4), 7–9.

Pourbabaee, B., Roshtkhari, M. J., & Khorasani, K. (2018). *Deep convolutional neural networks and learning ECG features for screening paroxysmal atrial fibrillation patients*. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 48(12), 2095–2104.

Ravi, D., Wong, C., Lo, B., & Yang, G.-Z. (2016). *A deep learning approach to on-node sensor data analytics for mobile or wearable devices*. IEEE journal of biomedical and health informatics, 21(1), 56–64.

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. arXiv preprint arXiv:1609.04747.

Sainath, T. N., Vinyals, O., Senior, A., & Sak, H. (2015). *Convolutional, long short-term memory, fully connected deep neural networks*. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 4580–4584.

San, P. P., Kakar, P., Li, X.-L., Krishnaswamy, S., Yang, J.-B., & Nguyen, M. N. (2017). *Deep learning for human activity recognition*. En Big Data Analytics for Sensor-Network Collected Intelligence (pp. 186–204). Elsevier.

Schmidhuber, J. (2015). *Deep learning in neural networks: An overview*. Neural networks, 61, 85–117.

Shearer, C. (2000). *The CRISP-DM Model: The New Blueprint for Data Mining*. Journal of Data Warehousing, 5(4), 13-22.

Srivastava, N. (2013). *Improving neural networks with dropout*. University of Toronto, 182(566), 7.

UCI Machine Learning Repository: REALDISP Activity Recognition Dataset Data Set. (s. f.). Recuperado 19 de abril de 2020, de https://archive.ics.uci.edu/ml/datasets/REALDISP+Activity+Recognition+Dataset

Vepakomma, P., De, D., Das, S. K., & Bhansali, S. (2015). *A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities*. 2015 IEEE 12th International conference on wearable and implantable body sensor networks (BSN), 1–6.

Walse, K. H., Dharaskar, R. V., & Thakare, V. M. (2016). *Pca based optimal ann classifiers for human activity recognition using mobile sensors data*. Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 1, 429–436.

Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L. (2019). *Deep learning for sensor-based activity recognition: A survey*. Pattern Recognition Letters, 119, 3–11.

Wirth, R., & Hipp, J. (2000). *CRISP-DM: Towards a standard process model for data mining*. Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining, 29–39.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W. (2015). *Convolutional LSTM network: A machine learning approach for precipitation nowcasting*. Advances in neural information processing systems, 802–810.

Yang, Jianbo, Nguyen, M. N., San, P. P., Li, X. L., & Krishnaswamy, S. (2015). *Deep convolutional neural networks on multichannel time series for human activity recognition*. Twenty-Fourth International Joint Conference on Artificial Intelligence.

Yang, Jing, & Yang, G. (2018). *Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer*. Algorithms, 11(3), 28.

Yang, Q. (2009). *Activity recognition: Linking low-level sensors to high-level intelligence*. Twenty-First International Joint Conference on Artificial Intelligence.

Yao, S., Hu, S., Zhao, Y., Zhang, A., & Abdelzaher, T. (2017). *Deepsense: A unified deep learning framework for time-series mobile sensing data processing*. Proceedings of the 26th International Conference on World Wide Web, 351–360.

Yuan, R., & Guangchen, B. (2011). *New neural network response surface methods for reliability analysis*. Chinese Journal of Aeronautics, 24(1), 25–31.

Zebin, T., Scully, P. J., & Ozanyan, K. B. (2016). *Human activity recognition with inertial sensors using a deep learning approach*. 2016 IEEE SENSORS, 1–3.

Zhang, L., Wu, X., & Luo, D. (2015). *Recognizing human activities from raw accelerometer data using deep neural networks*. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 865–870.

Zhang, W., Qin, L., Zhong, W., Guo, X., & Wang, G. (2019). *Framework of sequence chunking for human activity recognition using wearables*. Proceedings of the 2019 International Conference on Image, Video and Signal Processing, 93–98.

Zhang, Y., Zhang, Y., Zhang, Z., Bao, J., & Song, Y. (2018). *Human activity recognition based on time series analysis using U-Net*. arXiv preprint arXiv:1809.08113.

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). *Time series classification using multi-channels deep convolutional neural networks*. International Conference on Web-Age Information Management, 298–310.

Zhu, J., San-Segundo, R., & Pardo, J. M. (2017). *Feature extraction for robust physical activity recognition*. Human-centric Computing and Information Sciences, 7(1), 16.

Anexos

Anexo I. Código fuente

```
# IMPORTACIÓN DE LIBRERÍAS

from google.colab import drive
import pandas as pd
import numpy as np
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import seaborn as sns
from scipy import stats
import keras
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape, TimeDistributed, Conv1D
from keras.layers import Conv2D, MaxPooling2D, MaxPooling1D, ConvLSTM2D
from keras.layers import LSTM
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics


# DEFINICIÓN DE CONSTANTES

GOOGLE_DRIVE_PATH = '/content/gdrive'

REALDISP_PATH = 'gdrive/My Drive/realistic_sensor_displacement'

LOG_COLUMN_NAMES = [
  'second',
  'microsecond',
  'rla_acc_x',
  'rla_acc_y',
  'rla_acc_z',
  'rla_gyr_x',
  'rla_gyr_y',
  'rla_gyr_z',
  'rla_mag_x',
  'rla_mag_y',
  'rla_mag_z',
  'rla_quat_1',
  'rla_quat_2',
  'rla_quat_3',
  'rla_quat_4',
  'rua_acc_x',
  'rua_acc_y',
  'rua_acc_z',
  'rua_gyr_x',
  'rua_gyr_y',
  'rua_gyr_z',
  'rua_mag_x',
  'rua_mag_y',
  'rua_mag_z',
  'rua_quat_1',
```

```
    'rua_quat_2',
    'rua_quat_3',
    'rua_quat_4',
    'back_acc_x',
    'back_acc_y',
    'back_acc_z',
    'back_gyr_x',
    'back_gyr_y',
    'back_gyr_z',
    'back_mag_x',
    'back_mag_y',
    'back_mag_z',
    'back_quat_1',
    'back_quat_2',
    'back_quat_3',
    'back_quat_4',
    'lua_acc_x',
    'lua_acc_y',
    'lua_acc_z',
    'lua_gyr_x',
    'lua_gyr_y',
    'lua_gyr_z',
    'lua_mag_x',
    'lua_mag_y',
    'lua_mag_z',
    'lua_quat_1',
    'lua_quat_2',
    'lua_quat_3',
    'lua_quat_4',
    'lla_acc_x',
    'lla_acc_y',
    'lla_acc_z',
    'lla_gyr_x',
    'lla_gyr_y',
    'lla_gyr_z',
    'lla_mag_x',
    'lla_mag_y',
    'lla_mag_z',
    'lla_quat_1',
    'lla_quat_2',
    'lla_quat_3',
    'lla_quat_4',
        'rc_acc_x',
        'rc_acc_y',
        'rc_acc_z',
        'rc_gyr_x',
        'rc_gyr_y',
        'rc_gyr_z',
        'rc_mag_x',
        'rc_mag_y',
        'rc_mag_z',
        'rc_quat_1',
        'rc_quat_2',
        'rc_quat_3',
        'rc_quat_4',
        'rt_acc_x',
        'rt_acc_y',
        'rt_acc_z',
        'rt_gyr_x',
        'rt_gyr_y',
        'rt_gyr_z',
        'rt_mag_x',
        'rt_mag_y',
        'rt_mag_z',
        'rt_quat_1',
        'rt_quat_2',
```

```
    'rt_quat_3',
    'rt_quat_4',
    'lt_acc_x',
    'lt_acc_y',
    'lt_acc_z',
    'lt_gyr_x',
    'lt_gyr_y',
    'lt_gyr_z',
    'lt_mag_x',
    'lt_mag_y',
    'lt_mag_z',
    'lt_quat_1',
    'lt_quat_2',
    'lt_quat_3',
    'lt_quat_4',
    'lc_acc_x',
    'lc_acc_y',
    'lc_acc_z',
    'lc_gyr_x',
    'lc_gyr_y',
    'lc_gyr_z',
    'lc_mag_x',
    'lc_mag_y',
    'lc_mag_z',
    'lc_quat_1',
    'lc_quat_2',
    'lc_quat_3',
    'lc_quat_4',
    'label_id'
  ]

  ACTIVITIES = [
    'no activity',
    'walking',
    'jogging',
    'running',
    'jump up',
    'jump front and back',
    'jump sideways',
    'jump leg/arms open/closed',
    'jump rope',
    'trunk twist (arms outstretched)',
    'trunk twist (elbows bended)',
    'waist bends forward',
    'waist rotation',
    'waist bends (reach foot with opposite hand)',
    'reach heels backwards',
    'lateral bend',
    'lateral bend arm up',
    'repetitive forward stretching',
    'upper trunk and lower body opposite twist',
    'arms lateral elevation',
    'arms frontal elevation',
    'frontal hand claps',
    'arms frontal crossing',
    'shoulders high amplitude rotation',
    'shoulders low amplitude rotation',
    'arms inner rotation',
    'knees (alternatively) to the breast',
    'heels (alternatively) to the backside',
    'knees bending (crouching)',
    'knees (alternatively) bend forward',
    'rotation on the knees',
    'rowing',
    'elliptic bike',
    'cycling'
```

```
190.   ]
191.
192.   FEATURES = 117
193.
194.   NUM_CLASSES = len(ACTIVITIES)
195.
196.
197.
198.   # FUNCIONES AUXILIARES
199.
200.   # Montar Google Drive en Collaboratory
201.   def mount_google_drive(force_remount = False):
202.     drive.mount(mountpoint = GOOGLE_DRIVE_PATH, force_remount = force_remount)
203.
204.   # Lectura de los datos de un sujeto, en un escenario de colocación concreto
205.   def read_realdisp_log(subject, displacement):
206.     mount_google_drive()
207.     log_file_path = f'{REALDISP_PATH}/subject{subject}_{displacement}.log'
208.     df = pd.read_csv(filepath_or_buffer = log_file_path, sep = '\t', header = None, na
    mes = LOG_COLUMN_NAMES)
209.     df['label_name'] = np.array(ACTIVITIES)[df['label_id']]
210.     df['timestamp'] = df.apply(lambda row: (row['second'] * 1000000) + row['microsecon
    d'], axis = 1)
211.     print(f'Loaded {REALDISP_PATH}/subject{subject}_{displacement}.log')
212.     return df
213.
214.   # Mostrar datos básicos de un dataframe
215.   def show_dataframe_info(df):
216.     print(f'Número de columnas: {df.shape[1]}')
217.     print(f'Número de filas: {df.shape[0]}')
218.
219.   # Obtener mapa de color para las actividades
220.   def get_activities_colormap(df):
221.     # Normalizamos
222.     scaler = StandardScaler()
223.     activities = df['label_id'].unique()
224.     activities.sort()
225.     activities_scaled = scaler.fit_transform(activities.reshape(-1, 1))
226.     # Creamos el mapa de color
227.     return plt.cm.tab20(activities_scaled.flatten())
228.
229.   # Normalización de los datos
230.   def normalize_data(df):
231.     df_to_normalize = df.drop(
232.                           columns = [
233.                             'second',
234.                             'microsecond',
235.                             'timestamp',
236.                             'label_id',
237.                             'label_name'
238.                           ]
239.                         )
240.     standard_scaler = StandardScaler()
241.     normalized = standard_scaler.fit_transform(df_to_normalize)
242.     df_normalized = pd.DataFrame(
243.                           data = normalized,
244.                           columns = df_to_normalize.columns,
245.                           index = df_to_normalize.index
246.                         )
247.     result = pd.concat(
248.                   objs = [
249.                     df_normalized,
250.                     df['second'],
251.                     df['microsecond'],
252.                     df['timestamp'],
253.                     df['label_id'],
```

```
                  df['label_name']
                ],
                axis = 1
              )
    return result

  # Segmentación utilizando la técnica de la ventana deslizante, para una actividad
  def slice_by_activity(df, window_size, window_shift, activity):
    df_activity = df[df.label_id == activity]
    segments = []
    labels = []
    for i in range(0, len(df_activity) - window_size, window_shift):
      segment = df_activity.drop(
                  columns = ['second', 'microsecond', 'timestamp', 'label_id', 'label_name']
                ).values[i: i + window_size]
      segments.append(segment)
      labels.append(activity)
    reshaped_segments = np.asarray(segments).reshape(-1, window_size, FEATURES)
    labels = np.asarray(labels)
    return reshaped_segments, labels

  # Segmentación utilizando la técnica de la ventana deslizante, para todas las activi dades
  def slice(df, window_size, window_distance):
    for activity_index, activity_name in enumerate(ACTIVITIES, start = 1 ):
      (x_train_activity, y_train_activity) = slice_by_activity(df, window_size, window_distance, activity_index)
      if (activity_index == 1):
        x_train = x_train_activity
        y_train = y_train_activity
      else:
        x_train = np.concatenate((x_train, x_train_activity), axis = 0)
        y_train = np.concatenate((y_train, y_train_activity), axis = 0)
      print(activity_index)
    return x_train, y_train

  # Mostrar la matriz de confusión
  def show_confusion_matrix(validations, predictions):
    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize = (24, 24))
    sns.heatmap(matrix,
                cmap = 'coolwarm',
                linecolor = 'white',
                linewidths = 1,
                xticklabels = ACTIVITIES[1:],
                yticklabels = ACTIVITIES[1:],
                annot = True,
                fmt = 'd')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

  # Mostrar la curva de aprendizaje
  def show_learning_curve(history):
    plt.figure(figsize = (24, 12))
    plt.plot(history.history['accuracy'], 'r', label = 'Accuracy of training data')
    plt.plot(history.history['val_accuracy'], 'b', label = 'Accuracy of validation data')
    plt.plot(history.history['loss'], 'r--', label = 'Loss of training data')
    plt.plot(history.history['val_loss'], 'b--', label = 'Loss of validation data')
    plt.title('Model Accuracy and Loss')
    plt.ylabel('Accuracy and Loss')
    plt.xlabel('Training Epoch')
    plt.ylim(0)
```

```
316.      plt.legend()
317.      plt.show()
318.
319.
320.
321.    # CONFIGURACIÓN DE LOS GRÁFICOS
322.
323.    SMALL_SIZE = 16
324.    MEDIUM_SIZE = 20
325.    BIGGER_SIZE = 48
326.
327.    plt.rc('font', size = SMALL_SIZE)
328.    plt.rc('axes', titlesize = SMALL_SIZE)
329.    plt.rc('axes', labelsize = MEDIUM_SIZE)
330.    plt.rc('xtick', labelsize = SMALL_SIZE)
331.    plt.rc('ytick', labelsize = SMALL_SIZE)
332.    plt.rc('legend', fontsize = SMALL_SIZE)
333.    plt.rc('figure', titlesize = BIGGER_SIZE)
334.    plt.rc('figure', titlesize = BIGGER_SIZE)
335.
336.
337.
338.    # EXPLORACIÓN DE LOS DATOS
339.
340.    # Leemos los datos del sujeto 9 en condiciones ideales
341.    data = read_realdisp_log(9, 'ideal')
342.    pd.set_option('display.max_rows', 1000)
343.    show_dataframe_info(data)
344.
345.    # Número de muestras por actividad
346.    activities_colormap = get_activities_colormap(data)
347.    data_all = data
348.    data_all['label_name'].value_counts().plot.bar(figsize = (24, 12), color = activitie
    s_colormap, title = 'Incluyendo la "inactividad"')
349.    plt.show()
350.    data_activities = data[data['label_id'] != 0]
351.    data_activities['label_name'].value_counts().plot.bar(figsize = (24, 12), color = ac
    tivities_colormap, title = 'Sólo las actividades reales')
352.    plt.show()
353.
354.    # Eliminamos las muestras de la "inactividad"
355.    data_activities = data[data['label_id'] != 0]
356.
357.    # Gráficos exploratorios
358.
359.    fig, ax = plt.subplots(figsize = (24, 12))
360.    for index, activity in enumerate(ACTIVITIES, start = 0):
361.      y1 = data_activities[data_activities.label_id == index]['lc_mag_x']
362.      x1 = data_activities[data_activities.label_id == index]['timestamp']
363.      plt.plot(x1, y1, label = activity)
364.
365.    plt.xlabel('time')
366.    plt.ylabel('mag_x value')
367.    plt.title('LC - left calf')
368.    legend = ax.legend(loc='left', bbox_to_anchor = (0.0, -0.0))
369.    for l in legend.legendHandles:
370.        l.set_linewidth(12.0)
371.    plt.show()
372.
373.    y1 = data_activities['lc_mag_x']
374.    y2 = data_activities['lc_mag_y']
375.    y3 = data_activities['lc_mag_z']
376.    x1 = data_activities['timestamp']
377.
378.    fig, ax = plt.subplots(figsize = (24, 12))
379.    plt.plot(x1, y1, label = 'x')
```

```
    plt.plot(x1, y2, label = 'y')
    plt.plot(x1, y3, label = 'z')

    plt.xlabel('time')
    plt.ylabel('mag value')
    plt.title('LC - left calf')
    legend = plt.legend()
    for l in legend.legendHandles:
        l.set_linewidth(12.0)
    plt.show()

    y1 = data_activities['lc_mag_z']
    y2 = data_activities['rc_mag_z']
    x1 = data_activities['timestamp']

    fig, ax = plt.subplots(figsize = (24, 12))
    plt.plot(x1, y1, label = 'lc')
    plt.plot(x1, y2, label = 'rc')

    plt.xlabel('time')
    plt.ylabel('mag_z value')
    plt.title('left calf vs right calf')
    legend = plt.legend()
    for l in legend.legendHandles:
        l.set_linewidth(12.0)
    plt.show()

    y1 = data_activities['lc_mag_y']
    y2 = data_activities['lla_mag_y']
    x1 = data_activities['timestamp']

    fig, ax = plt.subplots(figsize = (24, 12))
    plt.plot(x1, y1, label = 'lc')
    plt.plot(x1, y2, label = 'lla')

    plt.xlabel('time')
    plt.ylabel('mag_y value')
    plt.title('left calf vs left lower arm')
    legend = plt.legend()
    for l in legend.legendHandles:
        l.set_linewidth(12.0)
    plt.show()

    plt.figure(figsize = (24, 24))
    corr = data_activities.corr()
    sns.heatmap(corr[corr >= 0.9], cmap = 'inferno', center = 0, annot = True)
    plt.title('Correlograma')
    plt.show()

    plt.figure(figsize = (24, 12))
    for index, activity in enumerate(ACTIVITIES, start = 0):
      sns.distplot(data_activities[data_activities.label_id == index]['RLA_ACC_X']);

    plt.figure(figsize = (24, 12))
    sns.distplot(data_activities['RLA_ACC_X']);

    df_num = data_activities.select_dtypes(include = ['float64'])
    df_num.hist(figsize = (24, 24))

    d = data_activities[['lc_acc_x', 'lc_acc_y', 'lc_acc_z', 'lc_gyr_x', 'lc_gyr_y', 'lc
_gyr_z', 'lc_mag_x', 'lc_mag_y', 'lc_mag_z', 'lc_quat_1', 'lc_quat_2', 'lc_quat_3', 'lc_
quat_4']]
    plt.figure(figsize=(24, 24))
    corr = d.corr()
    sns.heatmap(corr, cmap='inferno', center = 0, annot = True)
    plt.title('Correlograma para LC')
```

```
444.    plt.show()
445.
446.    d = data_activities[['lc_acc_x', 'rc_acc_x', 'lt_acc_x', 'rt_acc_x', 'lla_acc_x', 'r
    la_acc_x', 'lua_acc_x', 'rua_acc_x', 'back_acc_x']]
447.    plt.figure(figsize=(24, 24))
448.    corr = d.corr()
449.    sns.heatmap(corr, cmap='inferno', center = 0, annot = True)
450.    plt.title('Correlograma para ACC:X')
451.    plt.show()
452.
453.    # Comprobación de valores nulos
454.    data.isnull().sum()
455.
456.
457.
458.    # PREPARACIÓN DE LOS CONJUNTOS DE DATOS DE PRUEBAS Y TEST
459.
460.    def get_train_and_test_data(files, window_size, window_distance):
461.      for index, file in enumerate(files):
462.        df_file = read_realdisp_log(file[0], file[1])
463.        normalized_df_file = normalize_data(df_file)
464.        x_file, y_file = slice(normalized_df_file, window_size, window_distance)
465.        if (index == 0):
466.          x = x_file
467.          y = y_file
468.        else:
469.          x = np.concatenate((x, x_file), axis = 0)
470.          y = np.concatenate((y, y_file), axis = 0)
471.      # Hay que "eliminar" la clase 0, no-actividad
472.      y = y - 1
473.      y = np_utils.to_categorical(y, NUM_CLASSES - 1)
474.      # Importantísimo hacer shuffle
475.      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_
    state = 42)
476.      return x_train, x_test, y_train, y_test
477.
478.    # Parámetros de la ventana deslizante
479.    window_size = 200
480.    window_distance = 200
481.
482.    # Sujetos
483.    files = [
484.      [1, 'ideal'],
485.      [2, 'ideal'],
486.      [8, 'ideal'],
487.      [9, 'ideal'],
488.      [10, 'ideal']
489.    ]
490.
491.    x_train, x_test, y_train, y_test = get_train_and_test_data(files, window_size, windo
    w_distance)
492.
493.
494.
495.    # MODELO FCDNN
496.
497.    input_shape = window_size * FEATURES
498.
499.    # Hiperparámetros
500.    batch_size = 64
501.    epochs = 4
502.
503.    # Datos del modelo
504.    x_train_fcdnn = x_train.reshape(x_train.shape[0], input_shape)
505.    y_train_fcdnn = y_train
506.    x_test_fcdnn = x_test.reshape(x_test.shape[0], input_shape)
```

```
507.    y_test_fcdnn = y_test
508.
509.    # Modelo
510.    model_fcdnn = Sequential()
511.    model_fcdnn.add(Dense(units = 128, activation = 'relu', input_shape = (input_shape,
    )))
512.    model_fcdnn.add(Dropout(rate = 0.5))
513.    model_fcdnn.add(Dense(units = 64, activation = 'relu'))
514.    model_fcdnn.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
515.    print(model_fcdnn.summary())
516.
517.    # Compilación
518.    model_fcdnn.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics =
     ['accuracy'])
519.
520.    # Entrenamiento
521.    history_fcdnn = model_fcdnn.fit(x_train_dnn,
522.                                    y_train_dnn,
523.                                    batch_size = batch_size,
524.                                    epochs = epochs,
525.                                    validation_split = 0.2,
526.                                    verbose = 1)
527.
528.    # Gráfico del entrenamiento
529.    show_learning_curve(history_fcdnn)
530.
531.    # Matriz de confusión
532.    y_test_predicted_fcdnn = model_fcdnn.predict(x_test_fcdnn)
533.    max_y_test_predicted_fcdnn = np.argmax(y_test_predicted_fcdnn, axis = 1)
534.    max_y_test_predicted_fcdnn = max_y_test_predicted_fcdnn + 1 # Sumamos uno, ya que ha
    bía restado uno por la no-actividad
535.    max_y_test_fcdnn = np.argmax(y_test_fcdnn, axis = 1)
536.    max_y_test_fcdnn = max_y_test_dnn + 1 # Sumamos uno, ya que había restado uno por la
     no-actividad
537.    show_confusion_matrix(max_y_test_dnn, max_y_test_predicted_dnn)
538.
539.    print(classification_report(max_y_test_dnn, max_y_test_predicted_dnn))
540.    _, accuracy = model_fcdnn.evaluate(x_test_dnn, y_test_dnn, batch_size=batch_size, ve
    rbose=0)
541.    print(accuracy)
542.
543.    print('--- MEDIAS ---')
544.    scores = list()
545.    for r in range(10):
546.      model_fcdnn = Sequential()
547.      model_fcdnn.add(Dense(units = 128, activation = 'relu', input_shape = (input_shape
    , )))
548.      model_fcdnn.add(Dropout(rate = 0.5))
549.      model_fcdnn.add(Dense(units = 64, activation = 'relu'))
550.      model_fcdnn.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
551.      model_fcdnn.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics
     = ['accuracy'])
552.      history_fcdnn = model_fcdnn.fit(x_train_fcdnn,
553.                                      y_train_fcdnn,
554.                                      batch_size = batch_size,
555.                                      epochs = epochs,
556.                                      validation_split = 0.2,
557.                                      verbose = 1)
558.      _, accuracy = model_fcdnn.evaluate(x_test_fcdnn, y_test_dnn, batch_size = batch_si
    ze, verbose = 0)
559.      score = accuracy
560.      score = score * 100.0
561.      print('>#%d: %.3f' % (r+1, score))
562.      scores.append(score)
563.    print(scores)
564.    m, s = mean(scores), std(scores)
```

```
565.    print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))
566.
567.
568.
569.    # MODELO LSTM
570.
571.    # Hiperparámetros
572.    batch_size = 64
573.    epochs = 9
574.
575.    # Datos del modelo
576.    x_train_lstm = x_train
577.    y_train_lstm = y_train
578.    x_test_lstm = x_test
579.    y_test_lstm = y_test
580.
581.    # Modelo
582.    model_lstm = Sequential()
583.    model_lstm.add(LSTM(units = 128, input_shape = (window_size, FEATURES)))
584.    model_lstm.add(Dropout(rate = 0.5))
585.    model_lstm.add(Dense(units = 128, activation = 'relu'))
586.    model_lstm.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
587.    print(model_lstm.summary())
588.
589.    # Compilación
590.    model_lstm.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics =
    ['accuracy'])
591.
592.    # Entrenamiento
593.    history_lstm = model_lstm.fit(x_train_lstm,
594.                                  y_train_lstm,
595.                                  batch_size = batch_size,
596.                                  epochs = epochs,
597.                                  validation_split = 0.2,
598.                                  verbose = 1)
599.
600.    # Gráfico del entrenamiento
601.    show_learning_curve(history_lstm)
602.
603.    # Matriz de confusión
604.    y_test_predicted_lstm = model_lstm.predict(x_test_lstm)
605.    max_y_test_predicted_lstm = np.argmax(y_test_predicted_lstm, axis = 1)
606.    max_y_test_predicted_lstm = max_y_test_predicted_lstm + 1 # Sumamos uno, ya que habí
    a restado uno por la no-actividad
607.    max_y_test_lstm = np.argmax(y_test_lstm, axis = 1)
608.    max_y_test_lstm = max_y_test_lstm + 1 # Sumamos uno, ya que había restado uno por la
     no-actividad
609.
610.    show_confusion_matrix(max_y_test_lstm, max_y_test_predicted_lstm)
611.    print(classification_report(max_y_test_lstm, max_y_test_predicted_lstm))
612.
613.    print('--- MEDIAS ---')
614.    from numpy import mean
615.    from numpy import std
616.    scores = list()
617.    for r in range(10):
618.      history_lstm = model_lstm.fit(x_train_lstm,
619.                                    y_train_lstm,
620.                                    batch_size = batch_size,
621.                                    epochs = epochs,
622.                                    validation_split = 0.2,
623.                                    verbose = 1)
624.      _, accuracy = model_fcdnn.evaluate(x_test_dnn, y_test_dnn, batch_size=batch_size,
    verbose=0)
625.      score = accuracy
626.      score = score * 100.0
```

```
627.     print('>#%d: %.3f' % (r+1, score))
628.     scores.append(score)
629.   print(scores)
630.   m, s = mean(scores), std(scores)
631.   print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))
632.
633.
634.
635.   # MODELO CNN-LSTM
636.
637.   # Hiperparámetros
638.   batch_size = 64
639.   epochs = 9
640.   num_blocks = 4
641.   block_size = 50
642.
643.   # Datos del modelo
644.   x_train_lstm = x_train.reshape((x_train.shape[0], num_blocks, block_size, FEATURES))
645.   y_train_lstm = y_train
646.   x_test_lstm = x_test.reshape((x_test.shape[0], num_blocks, block_size, FEATURES))
647.   y_test_lstm = y_test
648.
649.   # Modelo
650.   model_cnn_lstm = Sequential()
651.   model_cnn_lstm.add(TimeDistributed(Conv1D(filters = 64, kernel_size = 3, activation
    = 'relu'), input_shape = (None, block_size, FEATURES)))
652.   model_cnn_lstm.add(TimeDistributed(Conv1D(filters = 64, kernel_size = 3, activation
    = 'relu')))
653.   model_cnn_lstm.add(TimeDistributed(Dropout(rate = 0.5)))
654.   model_cnn_lstm.add(TimeDistributed(MaxPooling1D(pool_size = 2)))
655.   model_cnn_lstm.add(TimeDistributed(Flatten()))
656.   model_cnn_lstm.add(LSTM(units = 128))
657.   model_cnn_lstm.add(Dropout(rate = 0.5))
658.   model_cnn_lstm.add(Dense(128, activation = 'relu'))
659.   model_cnn_lstm.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
660.   print(model_cnn_lstm.summary(line_length = 200))
661.
662.   # Compilación
663.   model_cnn_lstm.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['
    accuracy'])
664.
665.   # Entrenamiento
666.   history_cnn_lstm = model_cnn_lstm.fit(x_train_lstm,
667.                                         y_train_lstm,
668.                                         batch_size = batch_size,
669.                                         epochs = epochs,
670.                                         validation_split = 0.2,
671.                                         verbose = 1)
672.   # Gráfico del entrenamiento
673.   show_learning_curve(history_cnn_lstm)
674.
675.   # Matriz de confusión
676.   y_test_predicted_lstm = model_cnn_lstm.predict(x_test_lstm)
677.   max_y_test_predicted_lstm = np.argmax(y_test_predicted_lstm, axis = 1)
678.   max_y_test_predicted_lstm = max_y_test_predicted_lstm + 1 # Sumamos uno, ya que habí
    a restado uno por la no-actividad
679.   max_y_test_lstm = np.argmax(y_test_lstm, axis = 1)
680.   max_y_test_lstm = max_y_test_lstm + 1 # Sumamos uno, ya que había restado uno por la
     no-actividad
681.
682.   show_confusion_matrix(max_y_test_lstm, max_y_test_predicted_lstm)
683.   print(classification_report(max_y_test_lstm, max_y_test_predicted_lstm))
684.
685.
686.
```

```
687.    # MODELO ConvLSTM
688.
689.    # Hiperparámetros
690.    batch_size = 64
691.    epochs = 9
692.    num_blocks = 4
693.    block_size = 50
694.
695.    # Datos del modelo
696.    x_train_convlstm = x_train.reshape((x_train.shape[0], num_blocks, 1, block_size, FEA
    TURES))
697.    y_train_convlstm = y_train
698.    x_test_convlstm = x_test.reshape((x_test.shape[0], num_blocks, 1, block_size, FEATUR
    ES))
699.    y_test_convlstm = y_test
700.
701.    # Modelo
702.    model_convlstm = Sequential()
703.    model_convlstm.add(ConvLSTM2D(filters = 64, kernel_size=(1, 3), activation = 'relu',
     input_shape = (num_blocks, 1, block_size, FEATURES)))
704.    model_convlstm.add(Dropout( rate = 0.5))
705.    model_convlstm.add(Flatten())
706.    model_convlstm.add(Dense(units = 128, activation = 'relu'))
707.    model_convlstm.add(Dense(units = NUM_CLASSES - 1, activation = 'softmax'))
708.    print(model_convlstm.summary())
709.
710.    # Compilación
711.    model_convlstm.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metric
    s = ['accuracy'])
712.
713.    # Entrenamiento
714.    history_convlstm = model_convlstm.fit(x_train_convlstm,
715.                                          y_train_convlstm,
716.                                          batch_size = batch_size,
717.                                          epochs = epochs,
718.                                          validation_split = 0.2,
719.                                          verbose = 1)
720.
721.    # Gráfico del entrenamiento
722.    show_learning_curve(history_convlstm)
723.
724.    # Matriz de confusión
725.    y_test_predicted_convlstm = model_convlstm.predict(x_test_convlstm)
726.    max_y_test_predicted_convlstm = np.argmax(y_test_predicted_convlstm, axis = 1)
727.    max_y_test_predicted_convlstm = max_y_test_predicted_convlstm + 1 # Sumamos uno, ya
    que había restado uno por la no-actividad
728.    max_y_test_convlstm = np.argmax(y_test_convlstm, axis = 1)
729.    max_y_test_convlstm = max_y_test_convlstm + 1 # Sumamos uno, ya que había restado un
    o por la no-actividad
730.
731.    show_confusion_matrix(max_y_test_convlstm, max_y_test_predicted_convlstm)
732.    print(classification_report(max_y_test_convlstm, max_y_test_predicted_convlstm))
```

Anexo II. Artículo de investigación

Aplicación del aprendizaje profundo al reconocimiento de la actividad humana

Óscar Piqueras Segura

Universidad Internacional de la Rioja, Logroño (España)

16 de julio de 2020

Resumen

Este trabajo tiene como objetivo aplicar técnicas de aprendizaje profundo para el reconocimiento de la actividad humana. Se han definido e implementado cuatro modelos de red neuronal y se han entrenado con los datos obtenidos a partir de sensores vestibles, colocados en personas que estaban realizando distintas actividades físicas. El conjunto de datos de entrenamiento contiene información realista, teniendo en cuenta el concepto de desplazamiento gradual. Tras la evaluación de los modelos de red propuestos, se observa que las redes recurrentes, al gestionar mejor las series temporales de datos, tienen mayor precisión a la hora de reconocer las actividades. Por otra parte, se ha comprobado que las redes convolucionales mejoran la extracción de características y, estableciendo modelos híbridos, pueden potenciar la capacidad expresiva de las redes recurrentes.

Palabras Clave

reconocimiento de la actividad humana, serie temporal, red neuronal recurrente, red neuronal convolucional

I. Introducción

El reconocimiento de la actividad humana (en inglés Human Activity Recognition, HAR) es una área de investigación clave en la interacción humano-computadora (en inglés, Human-Computer Interaction), así como en la computación móvil y ubicua [1].

Tradicionalmente, la investigación en visión artificial ha estado a la vanguardia en este campo. El reconocimiento automático de gestos y actividades a partir de imágenes fijas o video ha sido estudiado por muchos investigadores [2]. Pero con el tiempo, los esfuerzos para reconocer actividades de la vida diaria en entornos sin restricciones se han dirigido hacia el uso de sensores inerciales (como acelerómetros o giroscopios) colocados sobre el cuerpo. Esto se debe a varios motivos [1]:

- Los sensores reducen las limitaciones ambientales y de configuración que sufren las cámaras de video.
- La adquisición de las señales puede realizarse de una manera más precisa y efectiva, haciendo uso de múltiples sensores.
- La información recogida por los sensores es específica para una persona, por lo que la privacidad es más alta que en las señales adquiridas por una cámara, que pueden contener información de otros sujetos que no estén bajo estudio.

Los investigadores realizaron los primeros estudios sobre el reconocimiento de la actividad utilizando sensores vestibles. Inicialmente la elección de las actividades era arbitraria y no siempre relevante para ser aplicada al mundo real. Pero los avances en el reconocimiento hicieron que cada vez se propusieran escenarios más complejos y orientados a la aplicación real [2]. Actualmente existen múltiples dominios que se pueden beneficiar del reconocimiento de la actividad humana [1] [3], como el sector industrial, los deportes, el entretenimiento y la asistencia sanitaria.

El reconocimiento de la actividad humana se ha tratado habitualmente como un problema de reconocimiento de patrones. Esta estrategia ha avanzado, en gran medida, gracias a la adopción de algoritmos de aprendizaje automático como [4]:

- Árboles de decisión.
- Máquinas de vector de soporte.
- Clasificadores bayesianos ingenuos.
- Modelos ocultos de Márkov.

Sin embargo, este tipo de soluciones basadas en el reconocimiento de patrones, aunque pueden ser satisfactorias en algunos escenarios concretos, tienen una serie de desventajas [3]:

- Requieren de una extracción heurística y manual de las características. Se necesita mucho conocimiento y experiencia en el dominio, para poder realizar una extracción de características adecuada. Este conocimiento humano puede ayudar en determinadas tareas muy específicas. Pero para entornos complejos y tareas más generales, se tarda más tiempo en construir un sistema de reconocimiento y la probabilidad de hacerlo con éxito es menor [5].
- La experiencia humana necesaria a la hora de extraer características, hace que utilizando estas técnicas sólo se puedan aprender características muy superficiales [6]. Sólo se pueden reconocer actividades de bajo nivel (como caminar o correr) y se hace muy difícil reconocer actividades de alto nivel que requieren de contexto (como tomar un café) [7].
- Las técnicas de reconocimiento de patrones suelen necesitar una gran cantidad de datos correctamente etiquetados para entrenar el modelo. Pero en las aplicaciones reales la mayoría de los datos de actividad no están etiquetados, lo que provoca una disminución del rendimiento de los modelos cuando deben realizar tareas basadas en aprendizaje no supervisado [5].
- La mayoría de los modelos basados en el reconocimiento de patrones están pensados para aprender de datos estáticos. Pero en la vida real, realizar una actividad genera un flujo continuo de información.

El aprendizaje profundo ayuda a superar esas limitaciones [2]. Los procedimientos de extracción de características y construcción de modelos a menudo se realizan simultáneamente en las técnicas de aprendizaje profundo. Las características se pueden aprender automáticamente a través de una red neuronal en lugar de ser diseñadas manualmente. Además, una red neuronal profunda también puede extraer una representación de alto nivel, lo que la hace más adecuada para tareas complejas de reconocimiento de actividades que requieren de contexto. Cuando se enfrentan a una gran cantidad de datos sin etiquetar, los modelos generativos profundos pueden hacer uso de los datos sin etiquetar para realizar el entrenamiento [3].

Aunque los modelos de aprendizaje profundo han logrado resultados notables en la visión artificial, el procesamiento del lenguaje natural y el reconocimiento del habla, no se han explotado completamente en el campo del HAR [1].

En el presente articulo se aplican técnicas de aprendizaje profundo para intentar solucionar el problema del reconocimiento de la actividad humana. La propuesta consiste en el diseño, implementación y evaluación de cuatro redes neuronales profundas con distintas arquitecturas, capaces de resolver el problema del HAR basado en sensores a partir de un conjunto de datos público que recoja información sobre el reconocimiento de actividades. Los modelos de red que se proponen son:

- Red neuronal profunda completamente conectada.
- Red neuronal recurrente.
- Red neuronal convolucional seguida de una red neuronal recurrente.
- Red neuronal que combina elementos recurrentes y elementos convolucionales.

II. Estado del Arte

Reconocimiento de las actividades humanas

Aunque el reconocimiento de actividades humanas comparte muchos desafíos metodológicos con otros campos (como la visión artificial, el procesamiento del lenguaje natural o el reconocimiento del habla) se enfrenta a una serie de desafíos específicos. La visión artificial y el reconocimiento de voz tienen definiciones claras de los problemas, como detectar un objeto en una imagen o detectar una palabra en una oración. Por el contrario, el HAR ofrece más grados de libertad en términos de diseño e implementación del sistema [2].

Primero, no existe una forma estándar para definir las actividades humanas que permita formular el problema de una manera clara y común: ¿qué actividad debe reconocerse? ¿cómo se caracteriza una actividad en concreto?

En segundo lugar, la actividad humana es muy diversa y, por lo tanto, su reconocimiento requiere una selección cuidadosa de los sensores encargados de realizar las mediciones.

Aprendizaje automático y HAR

La figura 1 muestra el diagrama de flujo de cómo se aborda el HAR usando el reconocimiento de patrones. El primer paso es obtener los datos de actividad sin procesar, a partir de los distintos sensores. A continuación, se extraen manualmente las características o *features* de esas lecturas, utilizando el conocimiento y la experiencia humana. Por último, las características se utilizan como entrada para entrenar un modelo de reconocimiento de patrones y poder utilizarlo para inferir actividades en aplicaciones reales [3].

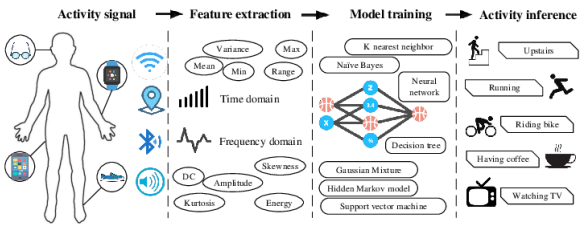

Fig 1. Reconocimiento de la actividad humana usando reconocimiento de patrones. Fuente: [3]

Pero este tipo de soluciones, basadas en el reconocimiento de patrones, tienen los puntos débiles expuestos en la introducción. Su principal problema es que es necesaria una extracción manual de las características y un amplio conocimiento del dominio [3].

Aprendizaje profundo y HAR

El aprendizaje profundo ayuda a mitigar las limitaciones de las propuestas basadas en el reconocimiento de patrones. La figura 2 muestra cómo funciona el aprendizaje profundo para el HAR, empleando diferentes tipos de redes neuronales.

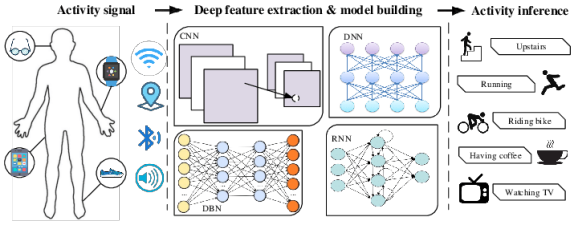

Fig 2. Reconocimiento de la actividad humana usando aprendizaje profundo. Fuente: [3]

Las redes neuronales profundas (en inglés Deep Neural Networks, DNNs) se pueden considerar una evolución de las redes neuronales artificiales (en inglés, Artificial Neural Networks, ANNs). Los primeros trabajos en los que se utilizan DNNs para solucionar el problema del HAR, lo hacen con redes poco profundas. En [8], primero extraen manualmente las características a partir de los datos de los sensores, para posteriormente alimentar un modelo de red neuronal profunda. En [9] se procede de la misma manera, pero realizando un análisis de los componentes principales antes de usar la red neuronal. En esos trabajos, la DNN sólo sirve como modelo de clasificación tras una extracción de características manual, por lo tanto, no generalizan muy bien. En [10] ya utilizan una DNN con 5 capas ocultas para realizar tanto el aprendizaje automático de las características como la clasificación. De esta manera, mejora el rendimiento y se pueden reconocer actividades más complejas.

Las redes neuronales recurrentes (en inglés, Recurrent Neural Networks, RNNs) utilizan correlaciones temporales entre sus neuronas. Se usan ampliamente en el reconocimiento del habla y el procesamiento del lenguaje natural. Hay pocos trabajos que utilicen RNNs aplicadas al problema del reconocimiento de la actividad humana. El enfoque principal de estas líneas de investigación se centra en entornos con recursos limitados que al mismo tiempo quieren obtener un buen rendimiento [3]. En [10] [11] [12] [13] se han propuesto redes recurrentes en el campo del HAR.

Las redes neuronales convolucionales (en inglés, Convolutional Neural Networks, CNNs) son especialmente idóneas para extraer características de las señales temporales de datos, y han logrado resultados prometedores en clasificación de

imágenes, reconocimiento de voz y análisis de texto. Cuando se aplican a la clasificación de series temporales, como es el caso del HAR, las CNNs tienen dos ventajas sobre otros modelos: dependencia local e invariancia de escala [3]. La dependencia local significa que las señales cercanas en el tiempo probablemente estén correlacionadas (y por lo tanto pertenezcan a la misma actividad), mientras que la invariancia de escala se refiere a la diferencia de escala para diferentes ritmos o frecuencias (a las que se puede realizar una misma actividad). Debido a la efectividad de las CNNs, la mayoría de los trabajos se centran en esta área [14] [15] [7] [16].

Los modelos híbridos se obtienen a partir de la combinación de varios modelos profundos. Un modelo híbrido emergente es la combinación de CNNs y RNNs [3]. En [17] [18] se proporcionan ejemplos de cómo combinar CNNs y RNNs, demostrándose su buen rendimiento. La razón es que las CNNs pueden capturar mejor la relación espacial, mientras que las RNNs pueden hacer uso de la relación temporal. Esta combinación mejorar la capacidad de reconocer actividades que tienen distribuciones de señal y lapsos de tiempo variados [3].

Conjunto de datos REALDISP

Este conjunto de datos se recopiló con el propósito inicial de investigar cómo afecta el desplazamiento de los sensores al proceso de reconocimiento de la actividad humana. El nombre de REALDISP proviene de *REAListic sensor DISPlacement*. Pero los datos se prestan perfectamente para ser utilizados en la evaluación comparativa de técnicas de reconocimiento de la actividad en condiciones ideales, es decir, sin desplazamiento en los sensores [19].

Existen varios trabajos realizados partiendo de este conjunto de datos. Uno de ellos es de los propios creadores de REALDISP, en el que se utilizan algoritmos de aprendizaje automático para estudiar el efecto que tiene el desplazamiento de los sensores a la hora de realizar el reconocimiento de las actividades [20]. Empleando el algoritmo del centro de clase más cercano (en inglés Nearest Class Center, NCC), el método de los k vecinos más cercanos (en inglés K-Nearest Neighbors, KNN) y árboles de decisión (en inglés Decision Trees, DTs) se realiza la clasificación en cada uno de los escenarios de desplazamiento.

Posteriormente, el mismo equipo realiza otro trabajo [21] sobre REALDISP. También utilizan algoritmos de aprendizaje automático para hacer el reconocimiento, en esta ocasión el método KNN, DTs y clasificadores bayesianos ingenuos (en inglés Naive Bayes, NB). Se diferencian tres marcos de trabajo, uno en el que se utilizan sólo 10 actividades, otro en el que se emplean 20 y un último en el que se considera el total de las 33 actividades.

Sobre REALDISP, también se puede destacar el trabajo [22], donde se compara los resultados de [21] con los obtenidos aplicando la técnica de aprendizaje automático denominada bosques aleatorios (en inglés, Random Forest, RF). Este trabajo se centra en la importancia de la extracción de las características. Utiliza *features* en el dominio del tiempo, extraídas de las señales de los sensores. Pero también en el dominio de la frecuencia, a partir de la transformada rápida de Fourier de las señales en el dominio del tiempo.

Por último, hay que nombrar el trabajo [23], en el que ya se aplican técnicas de aprendizaje profundo sobre el conjunto de datos REALDISP. Se propone una red neuronal convolucional para resolver el problema del reconocimiento de las actividades. Se llega a la conclusión de que mejora en gran medida respecto a los métodos basados en aprendizaje automático, ya que la red convolucional es capaz de realizar una extracción de las características más eficiente que las efectuadas de forma manual.

III. Objetivos y Metodología

La intención final es aplicar técnicas de aprendizaje profundo para resolver el problema del reconocimiento de la actividad humana basada en sensores inerciales vestibles.

Este objetivo principal se ha descompuesto en una serie de objetivos más específicos:

- Explorar los distintos repositorios de datos públicos con información relacionada con el HAR.
- Seleccionar los métodos de aprendizaje profundo que se quieren aplicar a la resolución del problema.
- Diseñar e implementar redes neuronales con diferentes topologías, para ser entrenadas utilizando los datos etiquetados disponibles.
- Evaluar y comparar las distintas soluciones propuestas.

Para poder alcanzar los objetivos específicos de una manera ordenada, siguiendo el método científico, se ha decidido utilizar CRISP-DM.

CRISP-DM (del inglés CRoss-Industry Standard Process for Data Mining) se considera un modelo de referencia para representar el ciclo de vida de proyectos basados en el análisis de datos. Es un marco de trabajo agnóstico, independiente de la industria, de las herramientas y de las aplicaciones [24]. Surge en el ámbito de la minería de datos, pero su flexibilidad hace que sea fácilmente adaptable a cualquier proyecto en el que se gestionen grandes cantidades de datos, como es el caso del aprendizaje profundo. En la figura 3 se muestran las seis fases estándar del modelo: comprensión del negocio, comprensión de los datos, preparación de los datos, modelado, evaluación y despliegue.

Fig 3. CRISP-DM. Fuente: Creado por Kenneth Jensen, basado en [25]

IV. Contribución

Comprensión del negocio

Una de las formas de intentar llevar a cabo el reconocimiento de la actividad que está realizando una persona, implica predecir el tipo de movimiento que está efectuando basándose en los datos que proporciona un conjunto de sensores. Puede decirse que este tipo de reconocimiento de la actividad busca un conocimiento a alto nivel sobre las actividades, a partir de múltiples lecturas de bajo nivel realizadas por sensores [3]. Reconocer la actividad se convierte en un problema de clasificación a partir de series temporales de datos.

Para el problema del reconocimiento de actividades no existe un conjunto de datos estándar sobre el que trabajar, como sí sucede en otras áreas dentro del campo de la minería de datos. En este trabajo, se ha decidido hacer uso del conjunto de datos REALDISP [26]. Los principales motivos de esta elección son el

gran número de actividades físicas que contempla, la diversidad de información que proporcionan los sensores utilizados en la recopilación de datos, y el considerable número de sujetos participantes en el proceso.

Comprensión de los datos

REALDISP registra los datos de 17 sujetos mientras realizan una serie de 33 actividades deportivas o *fitness*. Se utilizaron 9 unidades de medición inercial de tipo *Xsens*. Proporcionan información de la aceleración tridireccional, la orientación espacial mediante giroscopio y datos sobre el campo magnético. Adicionalmente, ofrecen una estimación de la orientación en cuatro dimensiones, en formato de cuaternión [20].

Por lo tanto, cada sensor genera 13 valores en cada medición, lo que en total produce un conjunto global de 117 señales registradas en cada instante de tiempo. Las grabaciones fueron muestreadas a una frecuencia de 50 Hz, es decir, realizando una medición cada 0,02 segundos [21].

A la hora de realizar el estudio para realizar la recogida de datos que da lugar a REALDISP, se definen tres escenarios relativos a cómo están se posicionan los sensores sobre el sujeto: colocación ideal, auto-colocación y desplazamiento inducido. De esta forma, se presenta el concepto de desplazamiento gradual [20].

Preparación de los datos

Para este trabajo se ha decidido utilizar los datos de los sujetos 1, 2, 8, 9 y 10, bajo condiciones ideales de desplazamiento de los sensores. Se han seleccionado estos sujetos porque se realizó el registro de todas las actividades en el escenario de colocación ideal sin datos corruptos o inválidos [20]. Se utiliza el conjunto completo de las 33 actividades, descartando los datos proporcionados por los sensores cuando los sujetos no están realizando ninguna actividad real. Además, se ha optado por utilizar toda la información que ofrecen los sensores. Como trabajo futuro, se abre la puerta a realizar investigaciones utilizando sólo determinados subconjuntos de estos datos.

Un pre-procesamiento muy común al preparar conjuntos de datos para abordar el problema del HAR, es la estandarización de los valores [27]. La normalización es una actividad de gran importancia cuando se aplican técnicas de aprendizaje profundo que manipulan series temporales de datos [28].

Existen trabajos que abordan la problemática de la normalización utilizando específicamente el conjunto de datos REALDISP. En [22] se evalúan seis métodos de normalización y se llega a la conclusión de que la que mejores resultados proporciona es la normalización vectorial. Por lo tanto, en este trabajo se ha aplicado la normalización vectorial a los datos que proporcionan las 117 señales temporales de todos los sensores. Se facilita así la fase de entrenamiento de los modelos de aprendizaje profundo propuestos.

Un preprocesamiento más específico es la segmentación de señales, uno de los procesos más importantes a la hora de intentar resolver el problema del reconocimiento de la actividad humana. Un enfoque que se usa normalmente para realizar la segmentación es la técnica de la ventana deslizante.

Es muy importante seleccionar un tamaño de ventana deslizante adecuado al problema que se está intentando resolver. Disminuir el tamaño de la ventana permite una detección más rápida de la actividad que se está realizando, además de requerir de menos recursos para hacerlo. Por el contrario, si las actividades a reconocer son complejas, normalmente es recomendable utilizar ventanas más grandes [29]. El desplazamiento también juega un papel interesante, ya que puede utilizarse como técnica de *data augmentation*: a mayor solapamiento, mayor número de muestras tendrá el conjunto de datos final que se utilizará para entrenar los modelos de aprendizaje automático.

En [29] se demuestra que en el caso de REALDISP, las ventanas reducidas proporcionan el mejor nivel de rendimiento y detección de actividades. En este trabajo se ha decidido utilizar una ventana de 4 segundos (200 muestras) con un desplazamiento de 4 segundos (200 muestras), lo que supone que no existe solapamiento. Queda abierto a trabajos futuros el poder comparar distintos parámetros de segmentación, profundizando en cómo afecta a los resultados de los modelos.

Modelo Fully Connected Deep Neural Network

El primer modelo diseñado para solucionar el problema del HAR es una red neuronal profunda completamente conectada (en inglés Fully Connected Deep Neural Network, FCDNN). La topología de una red neuronal de este tipo cuenta con una capa de entrada, varias capas ocultas y una capa de salida. Las neuronas de la capa de entrada se alimentan directamente con las características del conjunto de datos. Por lo tanto, el número de neuronas de la capa de entrada es igual al número de *features* seleccionadas. Las siguientes capas son las ocultas, cuyas neuronas aplican la función de activación a las entradas que reciben, y generan una salida. Al ser completamente conectada, la salida de cada una de las neuronas de una capa está conectada con la entrada de todas las neuronas de la capa siguiente. Por último, la capa de salida está formada por tantas neuronas como clases a predecir tenga el problema de clasificación.

La arquitectura del modelo FCDNN propuesta para este trabajo se compone de una capa de entrada y cuatro capas ocultas. Se distribuyen de la siguiente manera:

- La capa de entrada está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).
- La primera capa oculta está formada por 128 neuronas, usando ReLU como función de activación.
- La segunda oculta es una capa de regularización, de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste del modelo.
- La tercera capa, de nuevo es una capa densa, completamente conectada, en este caso con 64 neuronas y de nuevo con ReLU como función de activación.
- Por último, la cuarta capa define la que es considerada capa de salida. Está compuesta por tantas neuronas como etiquetas a predecir, es decir, tantas neuronas como actividades. La función de activación de estas neuronas es SoftMax, utilizada ampliamente en los problemas de clasificación multiclase.

Modelo Long Short-Term Memory

La siguiente arquitectura modelada para realizar el reconocimiento de actividades humanas es una red de gran memoria a corto plazo (en inglés Long Short-Term Memory, LSTM). Los modelos de red LSTM son un tipo de red neuronal recurrente que puede aprender y recordar patrones durante largas secuencias de datos. Por lo tanto, suelen emplearse cuando los datos de entrada consisten en series temporales prolongadas, como sucede en el HAR.

La arquitectura de la red LSTM propuesta para este trabajo está formada pos las siguientes capas:

- La capa de entrada está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la

ventana deslizante (200 muestras por segmento).

- La primera capa oculta es de tipo *long short-term memory*. Está formada por 128 neuronas, que se activan mediante la función *tanh*. La función de activación ReLU parece inapropiada para las RNNs, ya que pueden generar valores de salida muy amplios y por lo tanto aumentar la probabilidad de divergencia [30].
- La segunda capa es de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste a los datos de entrenamiento.
- La tercera capa es de tipo denso, completamente conectada, que interpretará las características extraídas por la capa recurrente. Esta capa tiene 128 neuronas y utiliza ReLU como función de activación.
- Por último, la cuarta capa define la que se considera capa de salida. Está compuesta por tantas neuronas como etiquetas a predecir, es decir, tantas neuronas como actividades. La función de activación de estas neuronas es de nuevo SoftMax.

Modelo Convolutional Neural Network + LSTM

Una arquitectura de red más compleja que la Long Short-Term Memory, es la arquitectura CNN + LSTM. Utiliza capas de una red neuronal convolucional (en inglés Convolutional Neural Network, CNN) para realizar la extracción de características a partir de los datos de entrada y combina su funcionamiento con una red LSTM para completar la predicción del resultado [31]. Las redes CNN + LSTM se desarrollaron para resolver problemas de predicción de series temporales visuales, así como para generar descripciones textuales a partir de secuencias de imágenes [32].

La arquitectura propuesta para este modelo está dividida en dos bloques. Las primeras cinco capas definen la parte convolucional de la red, que lee bloques de la secuencia y realiza la extracción de características. El resultado alimenta a las capas de una red recurrente definida con la misma topología que el modelo LSTM propuesto anteriormente, y se encarga de interpretar las características recibidas de cada bloque de datos y realizar las predicciones.

La parte del modelo que representa una CNN está envuelta en una capa de tipo *TimeDistributed*, lo que permite ir leyendo en bloques cada uno de los grupos en los que ha resultado la segmentación de los datos de entrada. Sigue un patrón estándar, con dos capas convolucionales consecutivas seguidas de una capa de *dropout* y una de *max pooling*:

- La capa de entrada está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).
- La primera y la segunda capa son de tipo convolucional. Ambas tienen 64 filtros, un tamaño de *kernel* de 3 y utilizan ReLU como función de activación de sus neuronas.
- La tercera capa es de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste.
- La cuarta realiza un *max pooling* de tamaño 2, reduce el número de variables, preparando la salida para la red LSTM.
- Por último, una quinta capa aplana los datos para convertirlos en una serie temporal que puede manejar la red LSTM que viene a continuación.

La segunda parte del modelo es una red recurrente como la definida en el modelo LSTM propuesto previamente.

Modelo Convolutional Long Short-Term Memory

Las ideas propuestas por los modelos CNN + LSTM pueden avanzar un paso más. Una posible modificación es que las convoluciones de la CNN formen parte de la red LSTM. Esta combinación se llama Convolutional LSTM o ConvLSTM [33]. A diferencia de un modelo LSTM, que lee los datos directamente, y de CNN + LSTM, que interpreta la salida del modelo CNN, ConvLSTM utiliza las convoluciones directamente como parte de la lectura de los datos de entrada en las unidades LSTM.

En la arquitectura establecida para este último modelo de aprendizaje profundo, se utiliza una capa de tipo Convolutional LSTM. El resultado se aplana para poder aplicar una capa densa totalmente conectada, antes de que se use una capa de salida formada por 33 neuronas, una por cada posible clase a predecir. En concreto, se tienen las siguientes capas:

- La capa de entrada está compuesta por 23.400 neuronas, resultado de multiplicar el número de características (117 variables proporcionadas por los sensores) por el tamaño de la ventana deslizante (200 muestras por segmento).
- La primera capa es de tipo Convolutional LSTM. Tiene 64 filtros, un tamaño de *kernel* de 3 y utiliza ReLU como función de activación de sus neuronas.
- La segunda es una capa de regularización, de tipo *dropout*. Desactiva el 50% de las unidades en cada iteración del entrenamiento, para evitar así el sobreajuste.
- La tercera capa aplana los datos para que puedan ser gestionados por las capas densas que vienen a continuación.
- La cuarta capa es de tipo denso, completamente conectada. Tiene 128 neuronas y utiliza ReLU como activación.
- La quinta capa es la considerada como capa de salida de la red. Está compuesta por tantas neuronas como etiquetas a predecir, es decir, tantas neuronas como actividades. La función de activación de estas neuronas es SoftMax.

V. RESULTADOS

Modelo Fully Connected Deep Neural Network

La figura 4 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento del modelo FCDNN. Se aprecia que la red deja de mejorar a partir de la cuarta época.

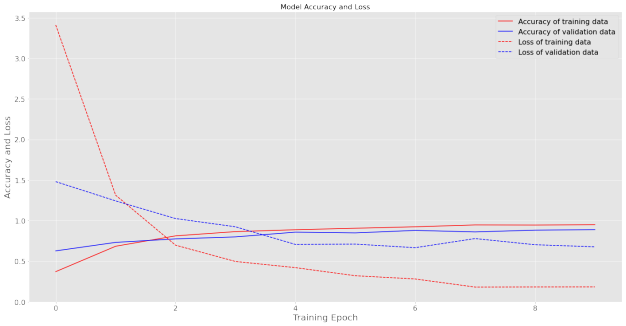

Fig 4. Métricas de accuracy y de loss en el entrenamiento del modelo FCDNN. Fuente: Elaboración propia.

Al evaluar el modelo entrenado durante 4 épocas, usando la función *classification_report* de *Keras*, se obtiene una exactitud del 85%.

Modelo Long Short-Term Memory

La figura 5 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento del modelo LSTM. Se aprecia que la red deja de mejorar a partir de la época 9.

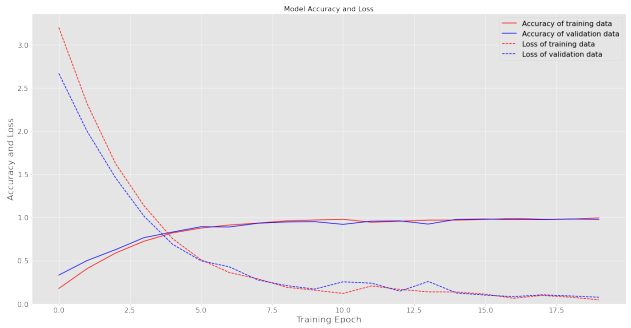

Fig 5. Métricas de accuracy y de loss en el entrenamiento del modelo LSTM. Fuente: Elaboración propia.

Al evaluar el modelo entrenado durante 9 épocas, usando la función *classification_report* de *Keras*, se obtiene una exactitud del 96%.

Modelo Convolutional Neural Network + LSTM

La figura 6 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento del modelo CNN + LSTM. Se aprecia que la red deja de mejorar a partir de la época 9.

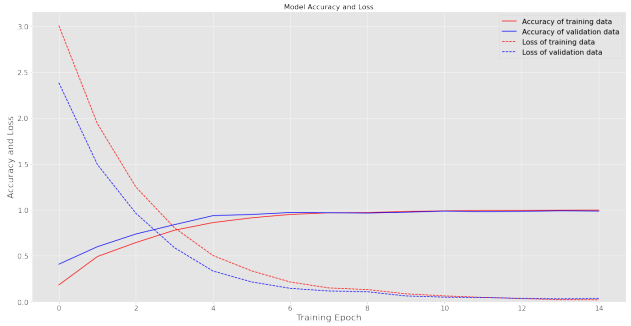

Fig 6. Métricas de accuracy y de loss en el entrenamiento del modelo CNN + LSTM. Fuente: Elaboración propia.

Al evaluar el modelo entrenado durante 9 épocas, usando la función *classification_report* de *Keras*, se obtiene una exactitud del 97%.

Modelo Convolutional Long Short-Term Memory

La figura 7 muestra la evolución de la *loss* y la *accuracy* en cada una de las épocas o iteraciones del entrenamiento del modelo ConvLSTM. Se aprecia que la red deja de mejorar a partir de la época 4.

Al evaluar el modelo entrenado durante 4 épocas, usando la función *classification_report* de *Keras*, se obtiene una exactitud del 97%.

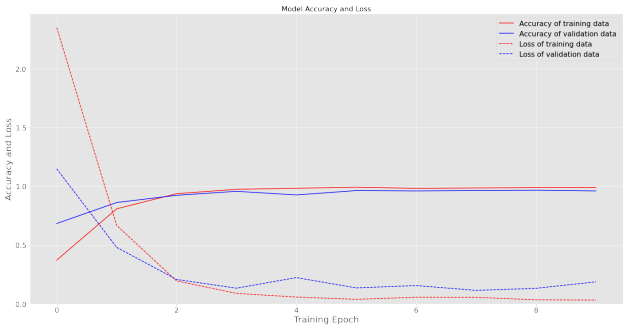

Fig 7. Métricas de accuracy y de loss en el entrenamiento del modelo ConvLSTM. Fuente: Elaboración propia.

Evaluación de los modelos

En la tabla I se muestran las principales métricas usadas para evaluar el funcionamiento de los modelos propuestos. Estos datos se han obtenido entrenando y evaluando los modelos múltiples veces, ya que las redes neuronales son modelos estocásticos, por lo que no siempre se obtienen los mismos resultados utilizando los mismos datos. Se ha hecho uso de la función *evaluate_model* de *Keras* para obtener la media y la desviación típica del conjunto de evaluaciones.

FCDNN	**LSTM**	**CNN + LSTM**	**ConvLSTM**
86,% ± 3,544	96,% ± 2,287	97,% ± 1,992	96,% ± 2,113
precision	precision	precision	precision
84,% ± 2,887	95,% ± 2,893	96,% ± 2,361	95,% ± 2,576
recall	recall	recall	recall
83,% ± 3,021	95,% ± 3,229	96,% ± 2,843	95,% ± 2,884
f1-score	f1-score	f1-score	f1-score
84,241% ± 2,332	**94,632% ± 3,121**	**96,827% ± 2,901**	**97,156% ± 2,645**
accuracy	**accuracy**	**accuracy**	**accuracy**

Tabla I. Métricas de los modelos. Fuente: Elaboración propia

La complejidad de los modelos de red neuronal propuestos puede evaluarse principalmente a partir del número de parámetros que tienen y del tiempo requerido para su entrenamiento. La tabla II muestra esta información para cada propuesta.

FCDNN	**LSTM**	**CNN + LSTM**	**ConvLSTM**
3.005.729	146.721	875.361	536.865
parámetros	parámetros	parámetros	parámetros
4	9	9	9
épocas	épocas	épocas	écpocas
1,163 ms	27.045 ms	1.005,910 ms	10.010 ms
entrenamiento	entrenamiento	entrenamiento	entrenamiento

Tabla II. Complejidad de los modelos. Fuente: Elaboración propia

VI. Discusión

El modelo FCDNN inicial es el que peor se comporta, con una precisión del 84,241%. Además, es el modelo más complejo atendiendo al número de parámetros, necesitando 3.005.729 parámetros ajustables. Aun así, proporciona un reconocimiento de actividades notable, aunque con algunas dificultades a la hora de distinguir ejercicios con una ejecución muy parecida, como es el caso de andar, trotar y correr.

El modelo de red LSTM, al ser un tipo de red neuronal recurrente, se adapta mejor a problemas en los que la entrada de datos es una serie temporal. Esto se aprecia en la gran mejora que proporciona, llegando a un 94,632% de precisión. También es varios órdenes de magnitud menos compleja a nivel de

parametrización, ya que sólo tiene 146.721 parámetros. De los cuatro modelos propuestos, es la que tiene menos parámetros ajustables, con diferencia. Al alcanzar este nivel de precisión, desaparecen los errores al etiquetar incorrectamente actividades muy similares, como sucedía en el caso de la red FCDNN. Este modelo reconoce y diferencia perfectamente entre las actividades de andar, trotar y correr. Sin embargo, tiene problemas con los saltos, confundiendo el salto normal con el salto adelante y atrás, por ejemplo.

La red neuronal CNN + LSTM mejora más la precisión, llegando al 96,827%. Esto se debe principalmente a que utiliza dos redes neuronales en serie, una tras otra. Primero la parte convolucional de la red se encarga de la extracción de las características y posteriormente la parte recurrente se encarga de hacer la predicción. Esta mejora en la precisión tiene como contrapartida el aumento de la complejidad de la red, que está compuesta de 875.361 parámetros. Este modelo, aunque mejora el reconocimiento de actividades respecto al modelo LSTM, sigue teniendo problemas con los saltos. No es capaz de distinguir perfectamente entre el salto normal y el salto adelante y atrás.

El modelo ConvLSTM evoluciona el concepto de CNN + LSTM, realizando los procesos de la red LSTM dentro de la propia convolución. Pero a pesar de su teórica superioridad, en el caso del problema del HAR no parece marcar una gran diferencia en cuanto a precisión. El modelo alcanza una precisión del 97,156%, que no es aumento excesivo respecto a la red CNN + LSTM. Donde sí se aprecia una mejora es en la complejidad, ya que sólo tiene 536.865 parámetros. Los problemas de este modelo a la hora de reconocer actividades, prácticamente se limitan a diferenciar en algunos casos el salto normal con el salto adelante y atrás, como sucede con la red CNN + LSTM.

VII. Conclusiones

Se ha propuesto como objetivo principal de este trabajo el aplicar técnicas de aprendizaje profundo para resolver el problema del HAR. Este objetivo general se ha descompuesto en un conjunto de objetivos más específicos, abordables y analizables por separado

El primer objetivo específico ha sido explorar los distintos repositorios de datos públicos con información relacionada con el HAR. Una vez descartada la opción de realizar una recolección de datos propios, se ha realizado un estudio de los distintos repositorios accesibles con información sobre el reconocimiento de actividades. Se ha decidido hacer uso del conjunto de datos REALDISP, ya que contempla un gran número de actividades físicas, recoge información de un número importante de sujetos participantes y, además, los sensores utilizados para realizar las mediciones aportan mucha información.

El segundo objetivo específico ha sido seleccionar qué métodos de aprendizaje profundo aplicar a la resolución del problema del HAR. Una vez descartadas otras arquitecturas en el campo del aprendizaje profundo, como las redes de creencia o el aprendizaje por refuerzo, se han propuesto métodos centrados en el ámbito de las redes neuronales profundas convolucionales y recurrentes.

Como tercer objetivo específico, se han diseñado e implementado varias redes neuronales con diferentes topologías, para ser entrenadas utilizando los datos del conjunto REALDISP. En concreto, se han escogido las siguientes cuatro arquitecturas:

- Fully Connected Deep Neural Network.
- Long Short-Term Memory.
- Convolutional Neural Network + Long Short-Term Memory.
- Convolutional Long Short-Term Memory.

El cuarto y último objetivo específico ha sido realizar la evaluación y comparación de las distintas soluciones propuestas. Se han calculado las principales métricas empleadas en la evaluación de algoritmos de clasificación en aprendizaje supervisado. Los resultados obtenidos muestran que todos los modelos de red son capaces de resolver el problema.

El modelo FCDNN es el que peor resultado obtiene, con una precisión del 84,241%. Además, es el modelo más complejo, con 3.005.729 parámetros.

El modelo LSTM, al tener una arquitectura recurrente y ser más adecuada para problemas con series de datos temporales, aumenta la precisión en gran medida respecto al modelo FCDNN, llegando al 94,632%. También es mucho menos complejo a nivel de parametrización, ya que sólo tiene 146.721 parámetros.

La arquitectura CNN + LSTM mejora ligeramente los resultados, al hacer uso de capas convolucionales para alimentar la subred LSTM. Alcanza el 96,827% de precisión. Como contrapartida aumenta la complejidad de la red, que está compuesta de 875.361 parámetros.

El modelo ConvLSTM, que realiza los procesos de la red recurrente dentro de la propia convolución, no supone un avance significativo. Obtiene una precisión del 97,156%. Donde sí se aprecia una mejora importante respecto a CNN + LSTM es en la complejidad, puesto que tiene 536.865 parámetros.

Una vez estudiado el problema del reconocimiento de las actividades humanas, se llega a la conclusión de que, aún siendo un problema ampliamente estudiado mediante técnicas de aprendizaje automático, queda margen de mejora utilizando métodos de aprendizaje profundo. El avance en la extracción de características que supone utilizar técnicas de aprendizaje profundo hace que sea menos relevante tener un gran conocimiento del dominio de los datos. Además, se pueden aprender características de alto nivel, lo que facilita el reconocimiento de actividades más complejas.

También se concluye que no hay un único modelo de aprendizaje profundo adecuado para resolver el problema del HAR. Tras la evaluación de los modelos de red propuestos en este trabajo, se observa que las redes recurrentes gestionan mejor las series temporales de datos y, por lo tanto, tienen mayor precisión a la hora de reconocer las actividades. Por otra parte, se ha visto que las redes convolucionales mejoran la extracción de características y que pueden potenciar la capacidad expresiva de las RNNs.

La evaluación de las técnicas de aprendizaje profundo propuestas e implementadas en este trabajo, se ha realizado estableciendo una serie de condiciones que acotan su alcance. Los siguientes son posibles trabajos futuros para profundizar con estas técnicas en la resolución del problema del HAR:

- Utilizar los datos de todos los sujetos de REALDISP: Este trabajo se ha limitado a usar los datos de 5 sujetos que no contenían información corrupta para el escenario de colocación ideal.
- Seleccionar sólo determinadas actividades: Se puede investigar el diseño de modelos más sencillos, que reconozcan un pequeño conjunto de actividades, para implantarse en dispositivos como pulseras de cuantificación.
- Hacer uso de los escenarios de desplazamiento gradual de los sensores: Este trabajo se ha centrado en las condiciones ideales de posicionamiento, pero las técnicas desarrolladas pueden tener aplicación en el escenario de auto-posicionamiento definido por el conjunto de datos REALDISP.
- Se abre la puerta a realizar investigaciones utilizando sólo determinados sensores o ciertas variables de los mismos: Por ejemplo, se pueden plantear modelos más sencillos utilizando

sólo la estimación de la orientación que ofrecen los cuaterniones.

- Experimentar con la parametrización de la ventana deslizante. El proceso de segmentación es muy importante en problemas que tratan con señales temporales de datos, por lo que el tamaño de la ventana deslizante y su desplazamiento juegan un papel fundamental en los resultados obtenidos por los modelos de red.

Referencias

[1] Hsu, H.-H., Chang, C.-Y., & Hsu, C.-H. (2017). *Big Data Analytics for Sensor-Network Collected Intelligence*. Morgan Kaufmann.

[2] Bulling, A., Blanke, U., & Schiele, B. (2014). *A tutorial on human activity recognition using body-worn inertial sensors*. ACM Computing Surveys (CSUR), 46(3), 1–33.

[3] Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L. (2019). *Deep learning for sensor-based activity recognition: A survey*. Pattern Recognition Letters, 119, 3–11.

[4] Lara, O. D., & Labrador, M. A. (2012). *A survey on human activity recognition using wearable sensors*. IEEE communications surveys & tutorials, 15(3), 1192–1209.

[5] Bengio, Y. (2013). *Deep learning of representations: Looking forward*. International Conference on Statistical Language and Speech Processing, 1–37.

[6] Yang, Jianbo, Nguyen, M. N., San, P. P., Li, X. L., & Krishnaswamy, S. (2015). *Deep convolutional neural networks on multichannel time series for human activity recognition*. Twenty-Fourth International Joint Conference on Artificial Intelligence.

[7] Yang, Q. (2009). *Activity recognition: Linking low-level sensors to high-level intelligence*. Twenty-First International Joint Conference on Artificial Intelligence.

[8] Vepakomma, P., De, D., Das, S. K., & Bhansali, S. (2015). *A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities*. 2015 IEEE 12th International conference on wearable and implantable body sensor networks (BSN), 1–6.

[9] Walse, K. H., Dharaskar, R. V., & Thakare, V. M. (2016). *Pca based optimal ann classifiers for human activity recognition using mobile sensors data*. Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 1, 429–436.

[10] Hammerla, N. Y., Halloran, S., & Plötz, T. (2016). *Deep, convolutional, and recurrent models for human activity recognition using wearables*. arXiv preprint arXiv:1604.08880.

[11] Inoue, M., Inoue, S., & Nishida, T. (2018). *Deep recurrent neural network for mobile human activity recognition with high throughput*. Artificial Life and Robotics, 23(2), 173–185.

[12] Edel, M., & Köppe, E. (2016). *Binarized-blstm-rnn based human activity recognition*. 2016 International conference on indoor positioning and indoor navigation (IPIN), 1–7.

[13] Guan, Y., & Plötz, T. (2017). *Ensembles of deep lstm learners for activity recognition using wearables*. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 1(2), 1–28.

[14] Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). *Time series classification using multi-channels deep convolutional neural networks*. International Conference on Web-Age Information Management, 298–310.

[15] Zebin, T., Scully, P. J., & Ozanyan, K. B. (2016). *Human activity recognition with inertial sensors using a deep learning approach*. 2016 IEEE SENSORS, 1–3.

[16] Hannink, J., Kautz, T., Pasluosta, C. F., Gaßmann, K.-G., Klucken, J., & Eskofier, B. M. (2016). *Sensor-based gait parameter extraction with deep convolutional neural networks*. IEEE journal of biomedical and health informatics, 21(1), 85–93.

[17] Ordóñez, F. J., & Roggen, D. (2016). *Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition*. Sensors, 16(1), 115.

[18] Yao, S., Hu, S., Zhao, Y., Zhang, A., & Abdelzaher, T. (2017). *Deepsense: A unified deep learning framework for time-series mobile sensing data processing*. Proceedings of the 26th International Conference on World Wide Web, 351–360.

[19] Banos, O. (s. f.). *REALDISP dataset*. Datasets. Recuperado 19 de abril de 2020, de http://orestibanos.com/datasets.htm

[20] Baños, O., Damas, M., Pomares, H., Rojas, I., Tóth, M. A., & Amft, O. (2012). *A benchmark dataset to evaluate sensor displacement in activity recognition*. Proceedings of the 2012 ACM Conference on Ubiquitous Computing, 1026–1035.

[21] Banos, O., Toth, M. A., Damas, M., Pomares, H., & Rojas, I. (2014). *Dealing with the effects of sensor displacement in wearable activity recognition*. Sensors, 14(6), 9995–10023.

[22] Zhu, J., San-Segundo, R., & Pardo, J. M. (2017). *Feature extraction for robust physical activity recognition*. Human-centric Computing and Information Sciences, 7(1), 16.

[23] San, P. P., Kakar, P., Li, X.-L., Krishnaswamy, S., Yang, J.-B., & Nguyen, M. N. (2017). *Deep learning for human activity recognition*. En Big Data Analytics for Sensor-Network Collected Intelligence (pp. 186–204). Elsevier.

[24] Shearer, C. (2000). *The CRISP-DM Model: The New Blueprint for Data Mining*. Journal of Data Warehousing, 5(4), 13-22.

[25] IBM. (2016). *Guía de CRISP-DM de IBM SPSS Modeler*. 52.

[26] UCI Machine Learning Repository: REALDISP Activity Recognition Dataset Data Set. (s. f.). Recuperado 19 de abril de 2020, de https://archive.ics.uci.edu/ml/datasets/REALDISP+Activity+Recognition+Dataset

[27] Panwar, M., Dyuthi, S. R., Prakash, K. C., Biswas, D., Acharyya, A., Maharatna, K., Gautam, A., & Naik, G. R. (2017). *CNN based approach for activity recognition using a wrist-worn accelerometer*. 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2438–2441.

[28] Baldominos, A., Cervantes, A., Saez, Y., & Isasi, P. (2019). *A comparison of machine learning and deep learning techniques for activity recognition using mobile devices*. Sensors, 19(3), 521.

[29] Banos, O., Galvez, J.-M., Damas, M., Pomares, H., & Rojas, I. (2014). *Window size impact in human activity recognition*. Sensors, 14(4), 6474–6499.

[30] Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). *A simple way to initialize recurrent networks of rectified linear units*. arXiv preprint arXiv:1504.00941.

[31] Sainath, T. N., Vinyals, O., Senior, A., & Sak, H. (2015). *Convolutional, long short-term memory, fully connected deep neural networks*. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 4580–4584.

[32] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). *Long-term recurrent convolutional networks for visual recognition and description*. Proceedings of the IEEE conference on computer vision and pattern recognition, 2625–2634.

[33] Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W. (2015). *Convolutional LSTM network: A machine learning approach for precipitation nowcasting*. Advances in neural information processing systems, 802–810.