Técnicas de Aprendizaje Profundo para reconocimiento de acciones de movimiento mediante los sensores de aceleración de un dispositivo móvil
Deep Learning techniques for motion action recognition using the acceleration sensors of a mobile device

Trabajo de Fin de Grado
Curso 2023–2024

Autores

Rodrigo Gómez Serrano

Miguel Manzano Rodríguez

Director

Gonzalo Pajares Martinsanz

Colaboradora externa

Clara Isabel López González

Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Informática

Universidad Complutense de Madrid

Técnicas de Aprendizaje Profundo para reconocimiento de acciones de movimiento mediante los sensores de aceleración de un dispositivo móvil
Deep Learning techniques for motion action recognition using the acceleration sensors of a mobile device

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Rodrigo Gómez Serrano
Miguel Manzano Rodríguez

Director
Gonzalo Pajares Martinsanz

Colaborador
Clara Isabel López González

Dirigida por el Doctor
Gonzalo Pajares Martinsanz

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

5 de junio de 2024

Dedicatoria

A nuestras familias, por apoyarnos incondicionalmente.

Agradecimientos

Agradecemos a la Universidad Complutense de Madrid y en particular a la Facultad de Informática por la formación recibida durante toda la carrera.

Nuestro más profundo agradecimiento a Gonzalo por asesorarnos y guiarnos durante todo el trabajo con total disponibilidad y siempre con amabilidad y conocimiento. Gracias también a Clara por toda la ayuda que nos ha brindado.

Por último, agradecer a nuestras familias y amigos más cercanos por acompañarnos durante este proyecto y, en general, durante toda la carrera sirviendo como apoyo y guía en todo momento.

Resumen

Técnicas de Aprendizaje Profundo para reconocimiento de acciones de movimiento mediante los sensores de aceleración de un dispositivo móvil

Clasificar acciones de movimiento mediante Aprendizaje Profundo es un problema desafiante y en constante desarrollo, y es un tema de investigación en campos muy diversos como deportes y rehabilitación. En este trabajo, se presenta un tipo de redes neuronales, las LSTM (Long Short Term Memory), y se muestra una visión general de su aplicación en el reconocimiento de acciones de movimiento capturadas mediante los sensores de aceleración de un dispositivo móvil. Con esto en mente, se emplea la plataforma de programación y cálculo numérico MATLAB para desarrollar dos vertientes de investigación con el fin de abordar un espectro más amplio. En primer lugar, mediante App Designer de MATLAB se elabora la aplicación estática, la cual permite entrenar los modelos LSTM y clasifica los datos de movimiento que estén previamente almacenados en el ordenador. A continuación, se introduce la aplicación dinámica, que clasifica datos de movimiento recogidos en el momento por los sensores de aceleración del movil mediante MATLAB Mobile. También se comparan varios resultados del entrenamiento de redes LSTM para resaltar la importancia que conlleva la elección de los parámetros involucrados en dichos modelos. Por útlimo, se plantea un enfoque global acerca de la comunicación entre un dispositivo móvil, encargado de registrar acciones de movimiento, y una aplicación en la "nube", responsable de almacenar, procesar e incluso interpretar estas acciones de manera remota. Para ello se emplea una plataforma de IoT llamada ThingSpeak que esta integrada en el entorno MATLAB.

Palabras clave

Inteligencia Artificial, aprendizaje profundo, redes neuronales recurrentes, Long Short Term Memory (LSTM), Entrenamiento, MATLAB, IoT, Computación en la nube.

Abstract

Deep Learning techniques for motion action recognition using the acceleration sensors of a mobile device

Classifying motion actions through Deep Learning is a challenging and ongoing problem, and it is the subject matter in many different fields such as sports and rehabilitation. In this work, a type of recurrent neural networks, LSTM (Long Short-term Memory), is presented followed by an overview of their application in recognizing motion actions, which are captured by the acceleration sensors of a mobile device. With this in mind, the MATLAB programming and numerical computing platform is employed to develop two lines of research, addressing a broader spectrum. First, using MATLAB's App Designer, a static application is developed, allowing the training of neural networks and the classification of motion data previously stored on the computer. Next, a dynamic application is introduced, which classifies motion data collected in real-time by the mobile's acceleration sensors using MATLAB Mobile. Various LSTM network training results are also compared to highlight the importance of parameter selection. Finally, a comprehensive approach to communication between a mobile device, responsible for recording motion actions, and a "cloud" application, responsible for storing, processing, and even interpreting these actions remotely, is proposed. For this purpose, an IoT (Internet of Things) platform, called ThingSpeak, integrated with MATLAB, is used.

Keywords

Artificial Intelligence, Deep Learning, Recurrent neural networks, Long Short Term Memory (LSTM), Training, MATLAB, IoT, Cloud computing.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo se expone el punto de partida del Trabajo Fin de Grado, diferenciándose cinco apartados: se comienza con la motivación y los incentivos que han llevado a la elección del tema. A continuación, se tratan los objetivos que se espera alcanzar, seguidos del plan de trabajo que se llevará a cabo. Después se comenta el estado de la cuestión, es decir, la situación actual en relación con el tema elegido. Finalmente, se expone la organización de la memoria, explicando el propósito de cada capítulo.

1.1. Motivación

La actividad humana es fundamental en nuestra vida cotidiana y tiene un impacto más que significativo en nuestra salud y bienestar. Desde los movimientos conscientes hasta los realizados de manera automática durante, por ejemplo, la vigilia, los seres humanos llevamos a cabo una amplia gama de acciones físicas. Se estima que un adulto promedio realiza entre 3000 y 5000 movimientos al día, entre los que se incluyen actividades como caminar, estirarse, levantarse o sentarse.

Por otro lado, el uso generalizado de dispositivos inteligentes, como teléfonos móviles, relojes y pulseras, ha aumentado exponencialmente con el paso de los años desde que se introdujo el primer *smartphone*. Además de ofrecer diversas funcionalidades como mensajería o recreación, estos dispositivos también recopilan multitud de datos sobre los movimientos y hábitos cotidianos. Esta cantidad masiva de datos brinda una oportunidad única para poder analizar y estudiar las acciones diarias que realiza el ser humano.

Con este trabajo, se pretende capitalizar la vasta cantidad de datos generados por el uso de dispositivos inteligentes, ofreciendo así una oportunidad única para analizar y comprender más a fondo las acciones diarias del ser humano mediante el desarrollo de un clasificador de movimientos humanos mediante redes neuronales.

De manera sucinta, las redes neuronales consisten en modelos matemáticos que aceptan unos datos de entrada y devuelven una serie de salidas. Las redes neuronales tienen un amplio rango de aplicaciones, y resultan muy beneficiosas a la hora de predecir y clasificar datos frente a modelos estadísticos tradicionales. Esta diferencia queda especialmente patente a la hora de enfrentarse a problemas no lineales, con interferencias que afecten al sistema.

En definitiva, en este Trabajo se estudiará la aplicación de una red neuronal (concretamente, Redes Neuronales Recurrentes del tipo *LSTM*), para clasificar datos de aceleración, obtenidos de un dispositivo móvil o *smartphone*. Se desarrollará una aplicación que va a emplear estos datos para, por un lado, entrenar redes, y por el otro, clasificar movimientos en una serie de categorías preestablecidas. Además estará conectada con una plataforma en la nube para analizar y tratar los movimientos adquiridos.

1.2. Objetivos

En este apartado se plantean los objetivos que se pretenden alcanzar durante el desarrollo del proyecto. Se van a cubrir tanto aspectos teóricos, que son aquellos ligados al estudio de conceptos relacionados con la implantación de las redes LSTM, como prácticos, que serán cuestiones más enfocadas en el desarrollo de la aplicación.

A continuación, se listan los objetivos:

- Aprender a trabajar en equipo en un proyecto de una extensión considerable, buscando coordinación, eficiencia y poner en común habilidades individuales.
- Profundizar en los tipos de Redes Neuronales existentes, concretamente en las Redes Neuronales Recurrentes, y dentro de ellas las *Long Short-Term Memory* o LSTM.
- Adquirir un conocimiento sólido sobre el significado y la importancia de los conceptos fundamentales en el proceso de entrenamiento de redes neuronales.
- Estudiar la aplicación de las redes LSTM a problemas prácticos.
- Resolver un problema de clasificación de movimientos mediante estas redes.
- Aprender y comprender la implementación de redes neuronales en MATLAB, concretamente las capas empleadas y sus parámetros.
- Aprender a usar las funciones de MATLAB para el entrenamiento y clasificación secuencia a secuencia, como *trainingOptions*.
- Desarrollar una aplicación para que el usuario pueda realizar la tarea de clasificación propuesta de forma intuitiva. Para ello se aprenderá a usar *MATLAB App Designer*.

- Comprender la conexión entre MATLAB y MATLAB Mobile. Orientarlo a este trabajo y conseguir clasificar datos provenientes del dispositivo móvil.
- Exponer en profundidad la metodología y herramientas utilizadas.
- Exponer algún ejemplo del rendimiento y los resultados de la aplicación tras entrenar una red.
- Investigar y aprender nociones generales sobre el Internet de las cosas.
- Comprender cómo funciona la plataforma de la nube ThingSpeak y la conexión que tiene con MATLAB.
- Realizar un programa de MATLAB que refleje las nociones de IoT aprendidas, recibiendo y enviando datos desde y hacia la nube (ThingSpeak).

1.3. Plan de trabajo

En esta sección se describe la hoja de ruta a seguir para la consecución de los objetivos descritos en el apartado anterior. Este plan se expone paso por paso, mostrando el hilo de hitos trazado para la realización del presente trabajo.

A continuación, se listan una serie de tareas a realizar. Su orden es cronológico, desde el inicio hasta la conclusión del trabajo. Destaca el desarrollo de la memoria, como una tarea consubstancial al resto.

- Desarrollo de la memoria del Trabajo Fin de Grado. La escritura de la presente memoria se realizará conforme se vayan llevando a cabo el resto de tareas de este Plan de trabajo, pudiendo así documentar de manera más fidedigna el trabajo realizado, en cuanto se materializa.
- Estudio de alternativas al planteamiento fundamental del trabajo. En concreto, la posible utilización de Python como una alternativa más versátil y utilizada que MATLAB. De igual forma, analizar la posibilidad de desarrollo de alguna aplicación desde cero, explorando el posible uso de Android Studio para el desarrollo de una aplicación para el sistema operativo *Android*.
- En cuanto al trabajo en cuestión, inicio con un estudio del problema a resolver, principalmente de los conceptos teóricos, adquiriendo una base firme desde la que poder desarrollar el proyecto. Para ello, buscar información en diversos formatos, principalmente escritas.
- Estudiar la aplicacion de los conceptos teóricos al problema en cuestión, viendo soluciones ya creadas y pensando en la adaptación concreta al problema de clasificación de movimientos.

- Ampliar la habilidad en el uso de MATLAB. Sin ser la primera vez que se utiliza el lenguaje, estudiar el desarrollo concreto con este lenguaje y herramientas útiles para el problema.

- Desarrollo minucioso de una aplicación estática y otra dinámica, asegurando plasmar el fundamento teórico en el código y su correcto funcionamiento. Para ello, forman parte del desarrollo la adaptación del modelo al problema, así como la realización de pruebas de entrenamiento y clasificación con el modelo ya desarrollado.

- En cuanto al desarrollo de la interfaz estática de la aplicación, estudio de la herramienta de creación de interfaces MATLAB App Designer y sus posibilidades, aprendiendo su funcionamiento y analizando a su vez la conexión con el código ya realizado.

- Para ilustrar de forma adecuada el funcionamiento y aplicabilidad del modelo, plasmar los resultados del mismo, mediante la obtención de gráficas acerca del entrenamiento.

- Como extensión del Trabajo, y como una manera de mostrar las posibilidades de la aplicación desarrollada, se lleva a cabo un estudio de la herramienta de IoT llamada ThingSpeak. El uso de la herramienta se explica en la memoria mediante gráficas y otras ilustraciones que muestran la interconexión llevada a cabo.

- Finalmente, depuración de la memoria, añadiendo y perfilando además partes fundamentales para la comprensión de la misma y del Trabajo Fin de Grado, como son el resumen, capítulos 1 (Introducción) y 6 (Conclusiones y trabajo futuro), y apéndices A y B.

1.4. Revision de modelos LSTM y aplicaciones

El objetivo de este apartado consiste en contextualizar el trabajo, es decir, comprender el panorama actual de investigación y aplicación de los conceptos expuestos. Esto resulta muy útil para posteriormente desarrollar un proyecto en línea con el panorama vigente. Primeramente, se hará una breve revisión histórica, detallando algunos avances que llevan a la situación actual.

Las redes neuronales *Long Short-term Memory* o LSTM fueron introducidas por Hochreiter y Schmidhuber en 1997, en un *paper* titulado *LONG SHORT-TERM MEMORY* (Hochreiter y Schmidhuber, 1997). En el texto se expone el fundamento teórico de estas redes y se presentan varios experimentos con los que consiguieron mostrar desde un primer momento la aplicabilidad del nuevo modelo. Avanzando hasta tiempos más actuales, cerca de la última década, estas redes se han estudiado y mejorado mucho. Analizando el marco actual de aplicaciones, y tal como se indica en un artículo de la editorial Springer (Van Houdt et al., 2020), existen diversas

aplicaciones donde el uso de redes LSTM resulta de gran utilidad. Algunas de estas son:

1. Las redes LSTM son utilizadas por Google para mejorar su reconocimiento de voz.
2. Las redes LSTM fueron usadas por Google Deepmind para crear AlphaStar, una inteligencia artificial que domina el juego de estrategia en tiempo real Starcraft II.
3. Las redes LSTM son utilizadas por Amazon Alexa para proporcionar respuestas más precisas y contextuales.
4. Las redes LSTM se emplean a menudo para tareas como etiquetado de partes del discurso, análisis de sentimientos y generación de texto.
5. Las redes LSTM se utilizan comunmente para crear subtítulos de vídeos, componer música y generar texto creativo.

Como puede apreciarse, las aplicaciones de este tipo de redes son numerosas y variadas. Además de las listadas anteriormente, existen aplicaciones de clasificación cotidianas, similares a la clasificación de movimientos presentada en este trabajo.

Una aplicación que resulta de gran interés es la clasificación de etapas de sueño, tanto en ratones (Yamabe et al., 2019), como en humanos, mediante redes CNN y LSTM. En el trabajo de (Fedorin y Slyusarenko, 2021), se estudia la clasificación de etapas del sueño en humanos, utilizando un sistema portátil de monitorización de bioseñales, es decir, un *wearable* con el que obtener los datos. Gracias al uso de redes neuronales de tipo LSTM, se consigue una precisión de más del 80 % en la clasificación de eventos respiratorios época-por-época.

Esta aplicación de las redes neuronales de tipo LSTM resulta de interés, al ser una clasificación similar a la presentada en este trabajo. La diferencia radica principalmente en los datos procesados, que serán de aceleración, así como de los elementos a clasificar, que serán clases de movimientos.

A continuación, se presenta una serie de investigaciones de resultados similares al trabajo presentado. Existen múltiples aplicaciones y estudios que proponen soluciones ante este problema. Pese a que no se han tomado ideas idénticas de otros proyectos, la estructura de los mismos es similar a la de este trabajo, pues tratan el mismo problema. La principal diferencia radicará en el uso de MATLAB, en contraste con el uso principal de Python que realizan la mayoría de estos proyectos. Un listado de los mismos se presenta a continuación:

- *LSTM Networks Using Smartphone Data for Sensor-Based Human Activity Recognition in Smart Homes* (Mekruksavanich y Jitpattanakul, 2021)

- *Time Series Classification for Human Activity Recognition with LSTMs using TensorFlow 2 and Keras* (Valkov, 2020)
- *Implementing LSTM for Human Activity Recognition using Smartphone Accelerometer data* (Nabriya, 2021)
- *Classification of Human Motion Activities using Mobile Phone Sensors and Deep Learning Model* (Khan et al., 2022)

1.5. Organización de la memoria

En definitiva, habiendo expuesto la motivación para el desarrollo de la aplicación, definidos los objetivos y el plan de trabajo, y habiendo revisado una serie de aplicaciones en el mismo ámbito del trabajo existentes en la actualidad, el resto de la memoria se organiza como sigue.

En el capítulo 2 se expone el fundamento teórico en el que se basa la tecnología utilizada, en este caso las redes LSTM. El capítulo 3 describe los detalles relativos al desarrollo de la aplicación, incluyendo los recursos necesarios y las metodologías utilizadas. El capítulo 4 contiene los resultados obtenidos en los procesos de entrenamiento de los modelos y la clasificación de resultados a través de los mismos. En el capítulo 5 se presenta una ampliación de las tecnologías utilizadas, adentrándose en el paradigma de Internet de las Cosas (Internet of Things, IoT), incluyendo posibilidades de aplicación y uso concreto en relación a la clasificación de acciones. Finalmente, el capítulo 6 contiene las conclusiones y el trabajo futuro.

Introduction

In this first chapter, the foundations of this bachelor's thesis are established, distinguishing five sections: it begins with the rationale and and incentives behind the election of the topic. Subsequently, the objectives that are expected to be achieved are discussed, followed by the work plan that will be carried out. Afterwards, the current state of the art, namely, the current state of knowledge and research regarding the chosen topic, is presented. Finally, the structure of this thesis is outlined, explaining the purpose of each chapter.

1.6. Rationale

Human activity is fundamental in our daily lives and has a more than significant impact on our health and well-being. From conscious movements to those performed automatically, such as those carried out during wakefulness, humans carry out a wide range of physical actions. It is estimated that an average adult performs between 3,000 and 5,000 movements a day, including activities like walking, stretching, standing up or sitting down.

On the other hand, the widespread use of smart devices, such as mobile phones, watches and wristbands, has increased exponentially over the years since the introduction of the smartphone. In addition to offering various functionalities such as messaging and recreation, these devices also collect a multitude of data about movements and habits. This massive amount of data provides a unique opportunity to analyze and study daily human actions.

This work aims to capitalize on the vast amount of data generated by the use of smart devices, thus offering a unique opportunity to analyze and better understand daily human actions through the development of a human movement classifier using neural networks.

In short, neural networks consist of mathematical models that accept input data and return a series of outputs. Neural networks have a wide range of applications and are very beneficial when it comes to predicting and classifying data compared

to traditional statistical models. This difference is especially evident when dealing with non-linear problems with interferences that affect the system.

In summary, this work will study the application of a neural network (specifically, Long Short-Term Memory Recurrent Neural Networks, or LSTM) to classify acceleration data obtained from a mobile device or smartphone. An application will be developed that will use this data to, on the one hand, train networks, and on the other hand, classify movements into a series of pre-established categories.

1.7. Objectives

In this section, the objectives to be achieved during the development of this project are outlined. These objectives align with the structure of the work, as they cover both theoretical and practical aspects. The theoretical aspects will relate to the study of concepts associated with the implementation, while the practical aspects will involve issues such as the development of the application.

In the following, the objectives are listed:

In this section, the objectives to be achieved during the development of the project are outlined. Both theoretical aspects, which are related to the study of concepts associated with the implementation of LSTM networks, and practical aspects, which are more focused on the development of the application, will be covered. The objectives are listed below:Learn to work as a team on a substantial project, seeking coordination, efficiency, and sharing individual skills. Deepen understanding of existing types of Neural Networks, specifically Recurrent Neural Networks, and within them, Long Short-Term Memory (LSTM) networks. Acquire a solid knowledge of the meaning and importance of fundamental concepts in the neural network training process. Study the application of LSTM networks to practical problems. Solve a movement classification problem using these networks. Learn and understand the implementation of neural networks in MATLAB, specifically the layers used and their parameters. Learn to use MATLAB functions for sequence-to-sequence training and classification, such as *trainingOptions*. Develop an application so that the user can perform the proposed classification task intuitively. For this, the use of *MATLAB App Designer* will be learned. Understand the connection between MATLAB and MATLAB Mobile. Orient it to this project towards classifying data from the mobile device. Thoroughly present the methodology and tools used. Present examples of the performance and results of the application after training a network. Research and learn general notions about the Internet of Things. Understand how the ThingSpeak cloud platform works and its connection with MATLAB. Develop a MATLAB program that reflects the learned IoT concepts, by receiving and sending data to and from the cloud (ThingSpeak).

1.8. Working plan

- Development of the Final Degree Project report. The writing of this report will be carried out as the other tasks of this work plan are performed, thus being able to document the work done more accurately as it materializes.

- Study of alternatives to the fundamental approach of the work. Specifically, the possible use of Python as a more versatile and widely used alternative to MATLAB. Similarly, analyzing the possibility of developing an application from scratch, exploring the potential use of Android Studio for developing an application for the Android operating system.

- As for the specific work, starting with a study of the problem to be solved, primarily focusing on theoretical concepts, acquiring a solid foundation from which to develop the project. For this, seeking information in various formats, mainly written.

- Studying the application of theoretical concepts to the specific problem, reviewing existing solutions, and considering specific adaptations to the motion classification problem.

- Upon beginning the development, and after ruling out the mentioned development alternatives, enhancing skills in using MATLAB. Although it is not the first time using the language, studying specific developments and tools for the problem.

- In-depth development of the application, ensuring that the theoretical foundation is reflected in the code and that it functions correctly. This includes adapting the model to the problem and conducting training and classification tests with the developed model.

- Regarding the development of the application's static interface, studying the MATLAB App Designer tool and its capabilities, learning its operation, and also analyzing the connection with the already written code.

- To adequately illustrate the model's functionality and applicability, presenting its results through graphs depicting the training process.

- As an extension of the work, and as a way to demonstrate the possibilities of the developed application, conducting a study of the IoT tool called ThingSpeak. The use of this tool is documented in the report through graphs and other illustrations showing the carried-out interconnection.

- Finally, refining the report, adding and fine-tuning fundamental parts for its understanding and that of the Final Degree Project, such as an abstract, chapters 1 (Introduction) and 6 (Conclusions and future work), and appendices A and B.

1.9. LSTM model revision and applications

The objective of this section is to contextualize the work, that is, to understand the current landscape of research and application of the concepts presented. This is very useful for subsequently developing a project in line with the current landscape. Firstly, a brief historical overview will be provided, detailing some advances that lead to the current situation.

Long Short-term Memory (LSTM) networks were introduced by Hochreiter and Schmidhuber in 1997, in a paper titled "LONG SHORT-TERM MEMORY"(Hochreiter y Schmidhuber, 1997). The text outlines the theoretical foundation of these networks and presents several experiments that initially demonstrated the applicability of the new model. Moving to more recent times, over the past decade, these networks have been extensively studied and improved. Looking at common applications, as indicated in an article by Springer (Van Houdt et al., 2020), there are various applications where the use of LSTM networks proves to be very useful. Some of these applications are:

- LSTM networks are used by Google to improve its voice recognition.
- LSTM networks were used by Google DeepMind to create AlphaStar, an artificial intelligence which excels in the real-time strategy game Starcraft II.
- LSTM networks are used by Amazon Alexa to provide more precise and contextual responses.
- LSTM networks are often employed for tasks such as part-of-speech tagging, sentiment analysis, and text generation.
- LSTM networks are commonly used to create video captions, compose music, and generate creative text.

As can be seen, the applications of these types of networks are numerous and varied. Besides those listed above, there are applications similar to the motion classification concept presented in this work.

One of these applications of interest is the classification of sleep stages, both in mice (Yamabe et al., 2019) and in humans, using CNN and LSTM networks. In the work by (Fedorin y Slyusarenko, 2021), the classification of sleep stages in humans is studied using a portable biosignal monitoring system, i.e., a wearable device to obtain the data. Thanks to the use of LSTM neural networks, a precision of more than 80

This application of LSTM neural networks is of interest, as it involves classification similar to the one presented in this work. The main difference lies in the data processed, which will be acceleration data, as well as the elements to be classified, which will be classes of movements.

Next, the focus will shift to more specific sources, similar to the presented work. There are multiple applications and studies that propose solutions to the problem. Although identical ideas from other projects have not been adopted, their structure is similar to this work, as they address the same problem. The main difference will lie in the use of MATLAB, in contrast to the predominant use of Python in most of these projects. A list of them is presented below:

- *LSTM Networks Using Smartphone Data for Sensor-Based Human Activity Recognition in Smart Homes* (Mekruksavanich y Jitpattanakul, 2021)
- *Time Series Classification for Human Activity Recognition with LSTMs using TensorFlow 2 and Keras* (Valkov, 2020)
- *Implementing LSTM for Human Activity Recognition using Smartphone Accelerometer data* (Nabriya, 2021)
- *Classification of Human Motion Activities using Mobile Phone Sensors and Deep Learning Model* (Khan et al., 2022)

1.10. Report organization

In conclusion, having presented the rationale for the development of the application, defined the objectives and the work plan, and having reviewed a seies of applications in the same field of knowledge, the rest of the report is organized as follows.

Chapter 2 presents the theoretical foundation on which the technology used, LSTM networks in this case, is based. Chapter 3 describes the details regarding the development of the application, including the necessary resources and the methodologies used. Chapter 4 contains the results obtained in the model training processes and the classification of those results through them. In Chapter 5, an extension of the technology used is presented, delving into the Internet of Things (IoT) paradigm, including possibilities of concrete application and use in relation to action classification. Finally, Chapter 6 contains the conclusions and future work.

Capítulo 2

Fundamento Teórico

En este capítulo se introducen los conceptos teóricos computacionales requeridos y que se aplican a lo largo del trabajo, en particular los correspondinetes a las redes RNC (Redes Neuronales Convolucionales). Los fundamentos teóricos que se exponen a continuación están basados en el trabajo de Pajares y col. (Pajares et al., 2021).

2.1. Conceptos generales

Intuitivamente, los algoritmos de aprendizaje supervisado son aquellos que aprenden a asociar un *input* con un *output* partiendo de un conjunto de datos de entrenamiento (datos de entrada ya etiquetados).

Por debajo de estos se distinguen dos tipos de problemas, regresión y clasificación. La principal diferencia entre ellos radica en la naturaleza de la variable de salida, siendo en el primer caso un valor numérico (continua) mientras que para la clasificación se devuelve una categoría (discreta). Para este último, hay que resaltar que la salida que se obtiene no es un valor que indique que esa entrada pertenece a la categoría A. En cambio, se obtiene un número real y se establece un rango de valores para poder diferenciar entre las categorías.

2.1.1. Concepto de optimización

El concepto de *optimización* de una función $f(x)$ consiste en minimizar o maximizar su valor modificando x. En general, se hablará de minimización, pues la maximización equivale a minimizar $-f(x)$.

La función $f(x)$ se denomina *función objetivo* o *criterio*. Algunos nombres que se suelen otorgar a estas funciones, es decir, las funciones a minimizar, son *función de coste*, *función de pérdida* o *función de error*, y se utilizarán estos términos indistinta-

mente (Goodfellow et al., 2016). De esta manera, se busca un valor $x^* = argmin f(x)$, que es el valor resultante de la minimización de la función.

Para poder encontrar este valor mínimo, la *función de pérdida* ha de ser diferenciable. Cabe destacar el caso de la utilización de ciertas *funciones de activación* como *ReLU*, que no son diferenciables, por lo que se debe aplicar alguna aproximación derivativa para poder utilizar la función en los procesos de aprendizaje.

Durante el proceso de entrenamiento de la red neuronal, se compara la salida de la misma (predicción) con la verdadera etiqueta del objetivo (*ground-truth*). Una función de pérdida es una medida de cuán bueno es un modelo de predicción en términos de poder aproximarse lo más posible (predecir) al resultado esperado. Uno de los métodos más utilizados para encontrar el punto mínimo de una función es el denominado *descenso de gradiente*. De esta forma, la optimización realizada en la función se denomina *optimización mediante el gradiente* (*gradient-based optimization*). Después de calcular la pérdida, se ha de mejorar el modelo, propagando el error hacia atrás a través de la estructura del modelo. Finalmente, se ajustan los pesos del modelo de red neuronal utilizada.

Como síntesis, el ciclo de aprendizaje consiste en el envío de los datos hacia adelante, la generación de predicciones y la mejora de los pesos durante la propagación hacia atrás o retropropagación. Al adaptar los pesos, el modelo probablemente mejore (en mayor o menor medida) y, por lo tanto, se dice que se ha realizado el aprendizaje, finalizando normalmente de acuerdo a un criterio de convergencia previamente establecido.

2.1.2. Optimización basada en el gradiente

Como se mencionó en la anterior sección, se quiere encontrar un valor resultante de la minimización de la función objetivo, $f(x)$, y para ello se introduce el concepto de *descenso de gradiente*. La *derivada de una función en un punto* permite escalar un pequeño cambio en la entrada para obtener el correspondiente cambio en la salida: $f(x + \epsilon) \sim f(x) + \epsilon f'(x)$. De esta manera, se puede disminuir $f(x)$ moviendo x en pequeños pasos con el signo opuesto de la derivada.

El gradiente generaliza la noción de derivada al caso de un vector: el *gradiente de f* es el vector que contiene en la posición i cada derivada parcial, $\frac{\partial}{\partial x_i} f(x)$, y se denota como $\nabla_x f(x)$. Para minimizar f en este caso, de la misma manera que en $\mathbb{R}$, buscamos la dirección donde f disminuye más rápidamente, que coincide con la opuesta del gradiente. Se propone:

$$x' = x - \epsilon \nabla_x f(x) \tag{2.1}$$

siendo ϵ la *razón de aprendizaje*, que es el valor escalar que determina el tamaño del paso (entre 0 y 1). La elección de ϵ es variada y existen métodos como la búsqueda

en línea (*linear search*) para hallar el más adecuado.

Los problemas de este método de optimización surgen cuando hay mínimos locales que no son globalmente óptimos, es decir, se optimizan funciones con muchas limitaciones locales que no son óptimas. Esto provoca una optimización difícil, especialmente si la entrada a la función es multidimensional, y obliga a conformarse con valores muy bajos pero no necesariamente con un mínimo formal. Además, existen ciertos inconvenientes que merecen una atención más detallada, especialmente la abundante cantidad de cálculos que se realizan en cada iteración.

Supóngase un sistema con 2000 muestras y 5 características. La suma de los residuos cuadrados (estimación del error) consta de tantos términos como muestras haya, en este caso, serían 2000 términos. Hay que calcular la derivada de esta función respecto a cada una de las características, lo que se traduce en $2000 \times 5 = 10000$ cálculos por iteración. Es común que se realicen 1000 iteraciones, lo que se traduce en un total de $10000 \times 1{,}000 = 10000000$ cálculos para completar el algoritmo. Esta carga de trabajo adicional resulta significativa, haciendo que el *descenso de gradiente* sea lento cuando se trata de datos extensos. A continuación se introducen dos de los métodos, que por estos motivos son más utilizados en la práctica.

2.1.3. Gradiente Descendente Estocástico (SGD)

El cálculo de la derivada parcial de la función de coste respecto a cada uno de los pesos de la red para cada observación es, dado el número de diferentes pesos y observaciones, inviable. Por lo tanto, una posible optimización consiste en la introducción de un comportamiento estocástico (aleatorio).

El método del *Gradiente Descendiente Estocástico*, o SGD (*Stochastic Gradient Descent*) consiste en minimizar una función objetivo que tiene la forma definida en en la ecuación (2.2) (Bishop, 2006; Ruder, 2017):,

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} J_i(w) \tag{2.2}$$

donde el parámetro w, que minimiza $J(w)$, debe estimarse. J_i se asocia con la i-ésima observación en el conjunto de datos utilizados para el entrenamiento (ajuste). $J_i(w)$ es el valor de la *función de pérdida* (*loss function*) en el i-ésimo ejemplo y $J(w)$ es el riesgo empírico.

El término estocástico proviene de que la elección de las muestras (observaciones) para el ajuste se hace de manera aleatoria. SGD hace algo tan simple cómo limitar el cálculo de la derivada a tan solo una observación por iteración. Existen algunas variaciones del método basadas, por ejemplo, en seleccionar varias muestras en vez de una (*mini-batch SGD*).

El *gradiente descendente* se usa para minimizar la siguiente función de forma

iterativa, con iteraciones t. Sin entrar en un detalle excesivo:

$$w(t+1) = w(t) - \epsilon \frac{1}{n} \sum_{i=1}^{n} \nabla J_i(w) \tag{2.3}$$

De forma iterativa, el método recorre el conjunto de entrenamiento y realiza la actualización indicada en la anterior ecuación para cada muestra de entrenamiento.

Además del algoritmo elemental del *SGD*, existen diversas variantes, extensiones y mejoras del algoritmo. Resulta de particular interés el hecho de fijar la *razón de aprendizaje*, ya que valores altos tienden hacia la divergencia de los datos, mientras que valores demasiado pequeños hacen que la convergencia sea lenta.

Una forma de abordar este problema es que dicha razón sea *variable*, disminuyendo a medida que se incorporan nuevas muestras (mayor aprendizaje). Esto se consigue haciéndola dependiente del número de datos o iteraciones. Una de tales extensiones es la del *método del momentum* o *momento* (Rumelhart et al., 1986; Murphy, 2012).

Intuitivamente, el *momentum* acelera el descenso en direcciones similares a las anteriores. Para ello, se guarda un vector que representa la media en ventana de los anteriores vectores de descenso, y si el nuevo vector es similar al *vector de momentum* aceleramos su descenso.

Este método recuerda la actualización Δw en cada iteración, determinando la siguiente actualización como una combinación lineal del gradiente y la actualización previa (Rumelhart et al., 1986; Sutskever et al., 2013):

$$\begin{aligned} \Delta w(t) &= \alpha \Delta w(t-1) - \epsilon \nabla J_i(w(t-1)) \\ w(t) &= w(t-1) + \Delta w(t-1) \end{aligned} \tag{2.4}$$

La anterior expresión conduce a:

$$w(t) = w(t-1) + \alpha \Delta w(t-1) - \epsilon \nabla J_i(w(t-1)) \tag{2.5}$$

donde $\epsilon > 0$ es la *razón de aprendizaje*, $\alpha \in [0, 1]$ es el *momento constante* que controla la velocidad de actualización de $\Delta w(t-1)$.

El vector de parámetros, w, que minimiza $J(w)$ es realmente el que se estima en el proceso de optimización.

2.1.4. Estimación del Momento Adaptativo (Adam)

Otra de las posibilidades en cuanto métodos de optimización, es *Adam*, cuyo nombre deriva de *Adaptive Moment Estimation* (Kingma y Ba, 2017). Este método mantiene una actualización de la media móvil, elemento a elemento, tanto de los gradientes de los parámetros como de sus valores al cuadrado.

$$\begin{aligned} m(t) &= \beta_1 m(t-1) + (1-\beta_1)\nabla J_i(w(t-1)) \\ v(t) &= \beta_2 v(t-1) + (1-\beta_2)(\nabla J_i(w(t-1)))^2 \\ w(t) &= w(t-1) - \frac{\alpha m(t-1)}{\sqrt{v(t-1)+\epsilon}} \end{aligned} \tag{2.6}$$

Una variante de *Adam* es *AdaMax* de forma que la norma L_2 se generaliza a L_p a la vez que se parametriza β_2 a β_2^p. Las normas con valores altos de p se hacen generalmente inestables, razón por la que se utilizan con frecuencia en la práctica las normas L_1 y L_2. Por este motivo Kingma y Ba (Kingma y Ba, 2017) proponen *AdaMax* y demuestran que $v(t)$ con L_∞ converge al siguiente valor más estable. Se define $u(t)$ para designar la utilización de esta norma en $u(t)$,

$$\begin{aligned} u(t) &= \beta_2^\infty v(t-1) + (1-\beta_2^\infty)(\nabla J_i(w(t-1))^\infty \\ &= max((\beta_2 * v(t-1)), \nabla J_i(w(t-1)) \end{aligned} \tag{2.7}$$

con esta modificación se puede reemplazar el término $\sqrt{v(t-1)} + \epsilon$ en la regla de actualización, quedando como sigue,

$$w(t) = w(t-1) - \frac{\alpha m(t-1)}{u(t-1)} \tag{2.8}$$

2.1.5. Parámetros generales

Durante el proceso de *actualización de los pesos* de la red, existen básicamente tres tipos de estrategias para el cálculo del error. En todas ellas, se dispone de N muestras de entrenamiento, la diferencia estriba en cómo se realiza la actualización (Kim, 2017; Brownlee, 2018). En este apartado se hará hincapié en ellas.

El proceso de optimización es *iterativo*, lo que quiere decir que cuando se busca el mínimo se realizan una serie de pasos, siendo el objetivo de cada paso ajustar lo mejor posible los pesos. Cada paso consiste en utilizar el modelo con los parámetros actuales, realizar una predicción sobre algunas muestras, comparar las predicciones con ellas, calcular el error y propagar el error hacia atrás para actualizar de nuevo los pesos del modelo.

Una *muestra*, *instancia* u *observación* es un dato simple que se suministra al algoritmo, junto con la salida correcta. Un *conjunto de muestras de entrenamiento* engloba un cierto número n de muestras.

Un *lote* (*batch*) o *mini-lote* (*mini-batch*) se define como el número de muestras utilizadas antes de llevar a cabo una actualización de los pesos de la red en el modelo. Esto significa que hasta que no han pasado todas las muestras del lote no se actualizan dichos pesos.

Un conjunto de datos de entrenamiento se puede dividir en uno o más *lotes*. Cuando todas las muestras de entrenamiento se utilizan para crear un *lote*, el algoritmo de aprendizaje se denomina *descenso del gradiente por lotes*.

Cuando el *lote* es del tamaño de una muestra, el algoritmo de aprendizaje se denomina, simplemente, *descenso de gradiente*, *estocástico* o de otro tipo.

Cuando el *tamaño del lote* posee más de una muestra y es menor que el tamaño del conjunto de datos de entrenamiento, el algoritmo de aprendizaje se denomina *descenso de gradiente de mini-lote* (*mini-batch*). Dimensiones habituales de los *mini-lotes* son 32, 64 y 128, lo que suele derivar en el hecho de que el último *lote* a veces tiene un tamaño menor que el resto, al no coincidir el número de muestras totales con un múltiplo de esas dimensiones. Una opción es eliminar algunas muestras (las sobrantes) para que esto no ocurra.

Por otra parte, una *época* (*epoch*) representa el número de veces que el conjunto total de muestras son procesadas por el algoritmo. Esto significa que cada muestra del conjunto ha tenido una oportunidad de actualizar los pesos en cada *epoch*. Una *epoch* puede tener uno o más *lotes*.

Para mayor claridad, se puede pensar en un bucle con tantas vueltas como número de epochs se disponga donde en cada vuelta se opera sobre las N muestras. Dentro de cada iteración, existe otro bucle anidado que da $\frac{N}{TL}$ vueltas, siendo TL el *tamaño del lote*. Es decir, da tantas vueltas como *lotes* haya.

Para ver un ejemplo, suponiendo que se tiene un conjunto de 208 *muestras de entrenamiento* y *dimensión de mini-lote* a 10 con 4 *épocas*. El *número de minilotes* por tanto será $208/10 \approx 20'8$, luego se formarán 20 *minilotes* de 10 muestras y 1 *minilote* con las 8 sobrantes. De esta forma habrá 21 iteraciones por época y así, en total $21 \times 4 = 84$ iteraciones en el entrenamiento.

En resumen, la *dimensión del lote* expresa el número de muestras procesadas antes de actualizar los pesos de la red.

El *número de epochs* es el número de pases completos de todo el conjunto de muestras de entrenamiento N.

Una *iteración* viene definida por el paso de un *lote* o *mini-lote*, de manera que tras ella se actualizan los pesos.

Como última anotación, cabe destacar que se puede establecer que la *razón de aprendizaje* varíe cada cierto número de *epochs*, por ejemplo, cada 5 *epochs* que disminuya un 10 % con respecto al valor inicial. Se puede ejecutar el algoritmo todo el tiempo que se desee e incluso detenerlo utilizando otros criterios además de un número fijo de *epochs*, tal como que el *error del modelo* o que la *función objetivo* no cambia con el número de iteraciones (se ha estabilizado). En estos casos se dice que se ha alcanzado la *convergencia* y los criterios así establecidos se denominan de convergencia.

2.2. Redes Neuronales Recurrentes

En este apartado se justifica y desarrolla la elección de *Redes Neuronales Recurrentes* o RNN (*Recurring Neural Network*) para resolver el problema, frente a otras opciones como podrían ser las *Redes Neuronales Convolucionales* o CNN (*Convolutional Neural Network*). Las CNN se caracterizan por la circulación de la información en una sola dirección, además de que requieren de una entrada de tamaño fijo y datos no relacionados. Sin embargo, la entrada para el tipo de problema a estudiar consiste en secuencias variables de datos, impidiendo así el uso de las CNN.

Las RNN, en cambio, se caracterizan por su posibilidad de abordar problemas que involucran datos secuenciales, de forma que pueden retener información de un estado en la secuencia de una iteración a la siguiente (Rumelhart et al., 1986). Siguiendo las ideas expuestas en (Goodfellow et al., 2016), cabe señalar que, así como con las CNN se pueden procesar imágenes de dimensiones variables, con las redes RNN se pueden escalar secuencias de datos de grandes dimensiones, que no serían prácticas para otro tipo de redes.

Dependiendo de la conexión entre las distintas capas, se establecen diferentes topologías de red, destacando para el caso a estudiar en este trabajo, las topologías *muchos a muchos* (Figura 2.1):

- Muchos a muchos (secuencia-secuencia)
- Muchos a muchos (secuencia-secuencia sincronizada)

En concreto, resultará de especial interés la segunda de estas topologías, donde tenemos un número determinado de celdas, y de pares entrada-salida para cada celda. Esta topología es precisamente la que utiliza una red *Long Short-Term Memory* (LSTM).

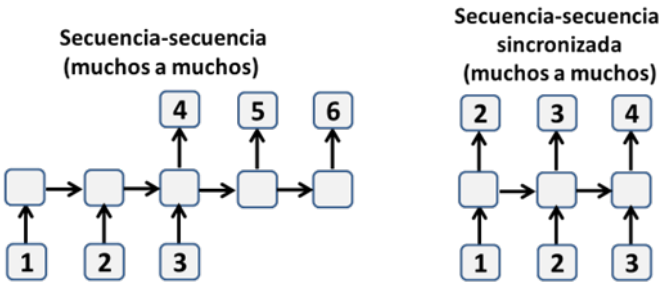

Figura 2.1: Topología de red *muchos a muchos*

2.3. Redes LSTM

En las redes neuronales profundas como las presentadas con anterioridad, pueden ocurrir *dependencias* conocidas como *long-term*. El problema se resume en el hecho de que los gradientes que se retropropagan sobre muchos estados tienden o bien a desaparecer, la mayoría de las veces, o a una explosión computacional, menos a menudo, que afecta claramente al proceso de optimización. Esto se debe principalmente a que el gradiente depende de los errores, tanto actuales como pasados, de forma que la acumulación continuada de errores origina problemas importantes para memorizar dependencias a largo alcance, de aquí la terminología denominada *long-term*. En (Goodfellow et al., 2016) se proporcionan algunas propuestas para resolver esta problemática. Dentro de las citadas propuestas destaca la que se describe a continuación, que se basa en el concepto denominado *Long Short-Term-Memory* (LSTM) que permite especificar la información que debe preservarse o eliminarse, esto es, almacenarse en forma de memoria, de ahí el término *memory*, o descartarse.

Las LSTM fueron introducidas por Hochreiter y Schmidhuber ((Hochreiter y Schmidhuber, 1997)) con el fin de resolver el problema del gradiente indicado previamente. Se trata de una red RNN específica que procesa una secuencia de pares de entrada-salida $(x_t, y_t)_{t=1}^{T}$. Para cada par de valores (x_t, y_t), la correspondiente celda LSTM toma una nueva entrada x_t y el valor oculto h_{t-1} obtenido en el paso previo, y con ello produce una estima para la salida objetivo y_i establecida al efecto y dada la secuencia de entrada previa $x_1, x_2, \ldots, x_t$ también con un nuevo valor oculto h_t y un nuevo valor del estado de la celda o memoria C_t.

La estructura de una celda LSTM (figura 2.2) consiste en lo siguiente. Cada celda temporal recibe, en cada paso de tiempo, una muestra de la secuencia x procedente de la capa de entrada, y envía el estado de celda actualizado, así como el valor de salida h al siguiente paso, es decir, a la siguiente celda. Por otra perte, en cada paso también se envía el correspondiente valor h a la capa de salida. Estos términos están indexados por el subíndice t correspondiente que establecen los pasos concretos de la secuencia en cada una de las distintas unifdades que conforman el modelo.

El estado de cada celda viene a ser una línea de transporte que atraviesa la celda con las interacciones que se indican explícitamente en la figura 2.3, de modo

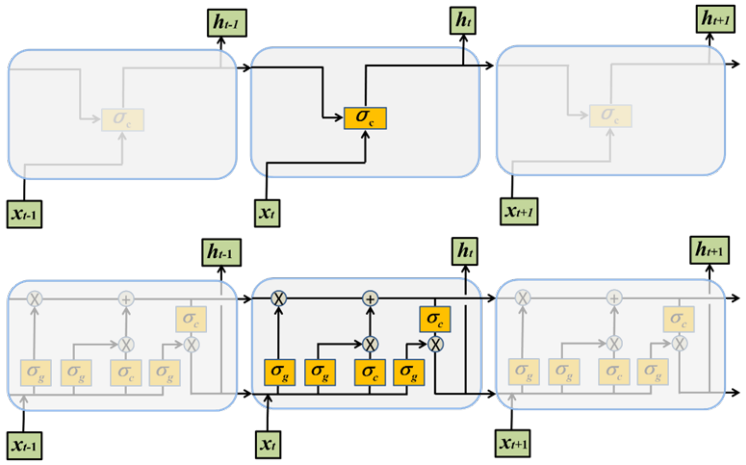

Figura 2.2: Estructura general y conexión de las celdas LSTM

que la LSTM puede eliminar o añadir información al estado de la celda mediante estructuras que se denominan en la terminología especializada como *gates*, formadas por operaciones del tipo mostrado en la subfigura de la parte derecha de la figura 2.3 con el flujo de información indicado por las flechas en dicha subfigura.

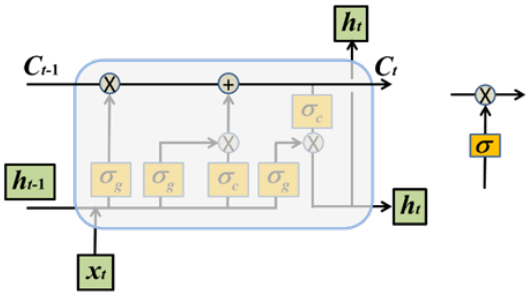

Figura 2.3: Línea de estado de la celda

El primer paso en una estructura LSTM es decidir qué información eliminar del estado de la celda, cuya responsable es una capa sigmoidal denominada puerta de olvido o forget gate, f_t, definida según la ecuación 2.9 y la figura 2.4,

$$f_t = \sigma_g(U_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.9}$$

de forma que teniendo en cuenta los pesos U_{xf} y W_{hf} y un bias b_f, a través de la función sigmoide (σ_g) se genera un valor entre 0 y 1 para cada estado de la celda C_{t-1}, de forma que un 1 significa mantener este estado por completo y un 0 deshacerse de él.

El siguiente paso consiste en decidir qué información nueva se va a almacenar en el estado de la celda, lo cual tiene dos partes. Primero, una capa sigmoide llamada puerta de entrada (input gate) i_t, figura 2.5, decide qué valores se actualizan.

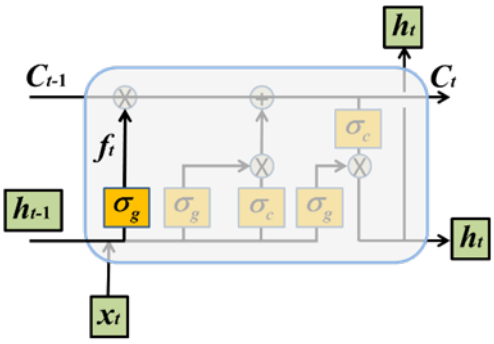

Figura 2.4: *Forget gate*

$$\begin{aligned} i_t &= \sigma_g(U_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ g_t &= \sigma_c(U_{xc}x_t + W_{hc}h_{t-1} + b_c) \end{aligned} \tag{2.10}$$

Seguidamente, una capa σ_c (generalmente de tipo tangente hiperbólica, *tanh*) crea nuevos valores candidatos g_t, que podrían agregarse al estado, teniendo en cuenta los pesos U_{xi}, W_{hi}, U_{xc} y W_{hc}, junto con los correspondientes *bias* b_i y b_c. En el siguiente paso, se combinan ambos para generar una actualización del estado.

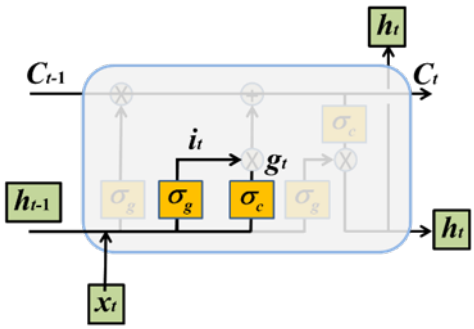

Figura 2.5: *Input gate* y estado de la celda

Ahora es el momento de actualizar el estado previo de la celda, g_{t-1}, al nuevo estado de esa celda g_t. Los pasos anteriores ya decidieron qué hacer, solo es necesario hacerlo ahora, para ello se multiplica el estado antiguo por f_t, olvidando los elementos que se decidieron olvidar antes. Luego se suma el término, figura 2.6, que son los nuevos valores candidatos, en función de cuánto se decide actualizar cada valor de estado.

$$C_t = f_t \circ C_{t-1} + i_t \circ g_t \tag{2.11}$$

Finalmente, es necesario obtener el resultado de la salida, que se basa en el estado de la celda, si bien, se trata de una versión filtrada, figura 2.7. Primero, se aplica la función *sigmoide* que decide qué partes del estado de la celda van a contribuir a la salida. Luego, se aplica la función *tanh*, que sitúa los valores en el rango -1 y 1 y se

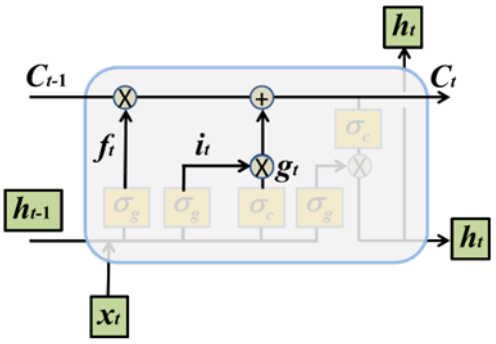

Figura 2.6: Actualización del estado de la celda

multiplica por la salida de la puerta de tipo sigmoidal, de modo que solo contribuyen a la salida las partes seleccionadas.

$$\begin{aligned} o_t &= \sigma_g(U_{xo}x_t + W_{ho}h_{t-1} + b_o) \\ h_t &= o_t \circ \sigma_c(C_t) \end{aligned} \tag{2.12}$$

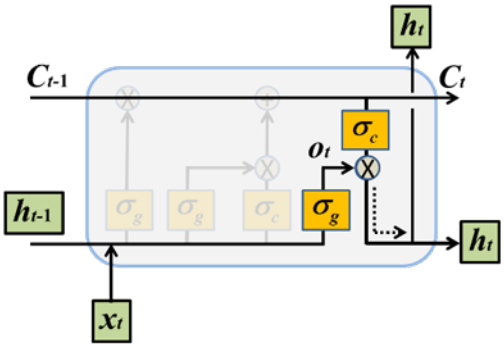

Figura 2.7: Generación de la salida

Por tanto, teniendo en cuenta las relaciones anteriores, la simbología asociada se interpreta como sigue:

1. $x_t \in \mathbb{R}^d$ representa la entrada a la celda dada, con d siendo su dimensionalidad, que se corresponde con el número de características de entrada.

2. $h_t \in \mathbb{R}^h$ representa el estado *short-term* de la celda, que es igual a la salida y_t de la celda. Su dimensión es h que se corresponde con el número de unidades ocultas de la red. Su valor inicial se establece a cero.

3. $f_t \in \mathbb{R}^h$, $i_t \in \mathbb{R}^h$, $g_t \in \mathbb{R}^h$, $o_t \in \mathbb{R}^h$ denotan, respectivamente, las puertas de olvido (*forget gate*), entrada (*input*), celda candidata (*cell candidate*) y salida (*output*).

4. El estado de la celda en el instante de tiempo t, viene dado por la ecuación (4.8), con e representando el producto de Hadamard, que consiste en la multiplicación

de vectores elemento a elemento, con la dimensión apropiada, siendo en este caso C_t de la misma dimensión que sus componentes (dimensión $\mathbb{R}^h$).

5. Las matrices de los pesos de entrada de la red asociados con la entrada x_t tienen las dimensiones $U_{xf}, U_{xi}, U_{xc}, U_{xo} \in \mathbb{R}^{h \times d}$, así como los pesos recurrentes $W_{hf}, W_{hi}, W_{hc}, W_{ho} \in \mathbb{R}^{h \times h}$ y los *bias* b_f, b_i, b_c, $b_0 \in \mathbb{R}^h$ que en determinadas implementaciones se concatenan como sigue en una única matriz.

$$U = \begin{bmatrix} U_{xi} \\ U_{xf} \\ U_{xc} \\ U_{xo} \end{bmatrix} \in \mathbb{R}^{4h \times d} \qquad W = \begin{bmatrix} W_{hi} \\ W_{hf} \\ W_{hc} \\ W_{ho} \end{bmatrix} \in \mathbb{R}^{4h \times h} \qquad b = \begin{bmatrix} b_i \\ b_f \\ b_c \\ b_o \end{bmatrix} \in \mathbb{R}^{4h \times l} \tag{2.13}$$

2.4. Clasificación utilizando LSTM

El problema estudiado en este trabajo, requiere del uso de las redes LSTM para clasificar secuencias de entrada, asignándole una determinada categoría a la que pertenece.

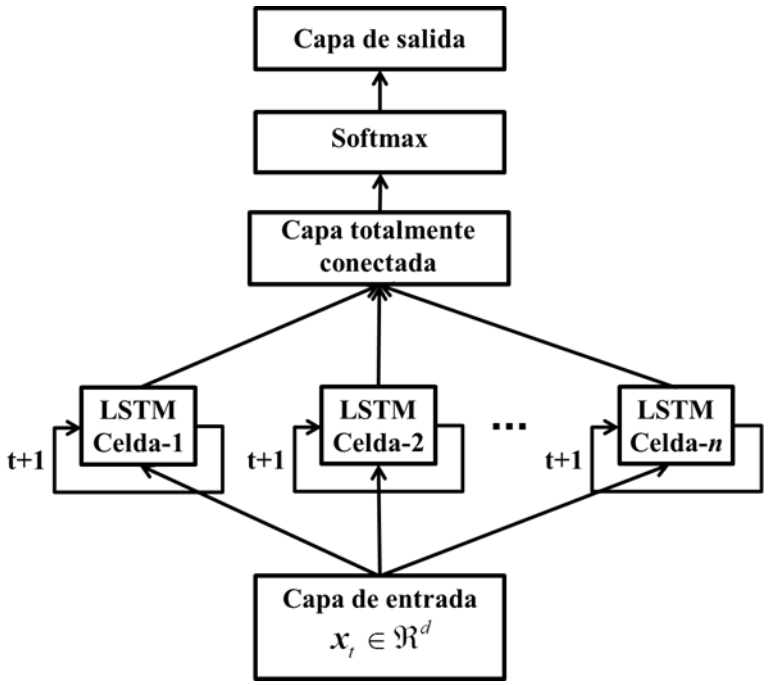

Figura 2.8: Topología de la red

La arquitectura de red utilizada en este caso es la mostrada en la figura 2.8, de forma que, en primera instancia, posee una capa de entrada con las características definidas en cada valor x_t de entrada y por tanto de la dimensión que le corresponda.

A continuación encontramos las siguientes capas intermedias. A la capa de entrada le sigue una capa oculta compuesta por C celdas o neuronas del tipo LSTM. Las neuronas ocultas se conectan a una capa totalmente conectada y a su vez a una capa *softmax* que finaliza en la capa de salida.

Finalmente, la última capa, la capa de salida, tiene la dimensión de un vector de etiquetas asociado a cada clase, de forma que cada componente de este vector representa la probabilidad, entre 0 y 1, de un tipo de movimiento definido por la correspondiente clase, es decir para cada acción de movimiento definido por las clases. De esta manera, la clase en la que se clasifiquen los valores de entrada será aquella con una mayor probabilidad.

El funcionamiento de la red en su conjunto es el siguiente. En cada paso de tiempo t, cada celda LSTM recibe el correspondiente x_t junto con la salida previa h_{t-1} de forma que se actualiza el estado de la red, como corresponde a este tipo de redes, en el siguiente paso de tiempo $t + 1$. En el último paso de tiempo de la secuencia cada neurona genera la correspondiente salida, que envía a la capa de proyección. Esta capa codifica los resultados de las salidas de todas las neuronas en un vector que representa a cada una de las clases o acción de movimiento, de esta forma se obtiene el vector final a la salida.

Capítulo 3

Desarrollo de la Aplicación

En este capítulo, se afronta el núcleo de esta investigación, detallándose los aspectos más pragmáticos y funcionales de la misma. Por tanto, se detallan aspectos cruciales como los recursos necesarios para el funcionamiento de la aplicación, la explicación técnica que sustenta su operatividad y el diseño que guía la experiencia del usuario.

El desarrollo de la aplicación se aborda desde dos vertientes. Por un lado, en la sección **??** se cuenta con una parte estática, es decir, se implementará una aplicación capaz de entrenar una red neuronal *LSTM* gracias a conjuntos de datos que previamente se encuentran en el ordenador donde se ejecuta la app. Además, mediante esta aplicación, se podrán utilizar estas redes entrenadas para clasificar conjuntos de datos estáticos (obtenidos del mismo modo que los de entrenamiento).

Por otro lado, en la sección 3.3, se explora la vertiente dinámica, que consiste en utilizar las redes ya entrenadas en la aplicación estática para clasificar datos que se obtienen a través de un dispositivo móvil en el momento de la clasificación.

3.1. Recursos Hardware y Software utilizados

En esta sección, se especifican los recursos de Hardware o Software utilizados para la aplicación. Estos recursos consisten en lo siguiente:

- *Dataset* utilizado
- Teléfono móvil, aplicación móvil de MATLAB (MATLAB Mobile)
- Programa MATLAB
- MATLAB App Designer
- MATLAB Drive

3.1.1. Recursos Hardware

Se utilizan recursos básicos, a disposición de cualquier usuario, como son un ordenador con o sin capacidad de instalar MATLAB localmente, y un dispositivo móvil, de cara a la obtención de datos.

El **ordenador** servirá para el desarrollo del código y la interfaz. Este desarrollo se realiza en principio mediante el programa de MATLAB instalado de forma local en el dispositivo. Si bien esto no sería absolutamente necesario, debido a que, gracias al uso de MATLAB Drive, el desarrollo puede realizarse *online*.

Además, cabe señalar el papel del **smartphone** en el suministro de datos a nuestro modelo, de cara a su clasificación según el modelo de red entrenado, gracias a la aplicación de MATLAB, obtenida con licencia de la Universidad Complutense desde Google Play en caso de Android o AppStore en caso de iOS.

3.1.2. Recursos Software

Seguidamente, se describen los recursos de software utilizados.

Para entrenar el modelo, el **conjunto de datos o *dataset*** utilizado, *HumanActivityTrain*, forma parte de la colección Deep Learning Toolbox de MATLAB. Este conjunto de datos contiene siete series de secuencias de datos de aceleración, obtenidas mediante el sensor de un teléfono inteligente portado por la persona que ha de moverse. Las secuencias varían en longitud, y tienen tres atributos: las componentes (x, y, z) de la aceleración.

Los datos se dividen en **XTrain** conteniendo los conjuntos de aceleraciones, formados por 6 secuencias, que se representan por matrices de 3 filas (x, y, z), y un número variable de columnas (dependiendo del número de datos por secuencia). **YTrain** está formado por 6 secuencias, de nuevo representadas por listas con una longitud variable y equivalente al mismo número de elementos que el conjunto de aceleración correspondiente de XTrain.

Como se ha mencionado anteriormente, en el dispositivo móvil se instala la aplicación MATLAB, que obtiene diversos datos procedentes de los sensores del dispositivo. En concreto, para nuestro modelo solo nos serán de utilidad los tres atributos anteriormente mencionados: las componentes (x, y, z) de la aceleración.

Como intermediario en todo este proceso, se utiliza la plataforma **MATLAB Drive**. En esta plataforma se guardan los archivos tanto de datos como *scripts* de código, que en conjunto contienen los recursos necesarios para ejecutar la aplicación. De esta manera, se puede acceder a ellos tanto desde el programa local de MATLAB en un ordenador, como el teléfono utilizado. De esta última manera, el código obtiene directamente los datos del dispositivo móvil.

El organigrama de los archivos es el siguiente. Primeramente, la interfaz está

contenida en archivos con el formato *.mlapp*, consistentes en varios archivos que definen las pantallas de nuestra aplicación de MATLAB AppDesigner: *pantallaInicio*, *pantallaTrain*, *pantallaTest*, *pantallaResultados* y *pantallaParametros*.

Seguidamente, se dispone de varios ficheros *.m*, que contienen código en MATLAB. Estos archivos tienen varias funciones, que se detallan a continuación:

- Con el código presente en MostrarDatosIniciales.m , se muestran por pantalla tres gráficas, representando las tres coordenadas de la aceleración frente al tiempo, según la actividad correspondiente.
- Con el código presente en *EntrenarParamLSTM.m*, se realizan varias tareas. Primeramente, se cargan los datos, y se define la arquitectura de red LSTM, determinando todos los parámetros de las capas y opciones de entrenamiento. Posteriormente, definimos la red, bien realizando el entrenamiento desde cero, utilizando los parámetros establecidos, o cargando una red ya entrenada. Finalmente, encontramos código para testear una red ya entrenada.
- Con el código en *PruebaThingSpeak.m*, se establece una conexión con un canal de ThingSpeak para leer y escribir datos.
- Con el código en *TrabajoFinalLSTM.m*, se crea un objeto que permite la captura de los datos de aceleración del dispositivo móvil, se utiliza la red ya entrenada para clasificarlos, y se imprime por pantalla la clasificación realizada.

Para finalizar la explicación de los recursos utilizados en la aplicación, cabe destacar que todo el código mencionado se encuentra disponible en el Apéndice B.

3.2. Explicación técnica de la aplicación estática

En este apartado se explica la aplicación estática. En concreto, se muestra cómo es el entrenamiento de una red neuronal *Long-short Term Memory (LSTM)* y cómo es posible clasificar movimientos en el caso de tener datos estáticos, es decir, que no se obtienen en el momento de la clasificación.

Primeramente, se explica el proceso a la vez que se muestra la interfaz de la aplicación implementada, comenzando con la pantalla de inicio (Figura 3.1), en la que se elige entre comenzar el entrenamiento de una red o usar una ya entrenada para testear una serie de movimientos.

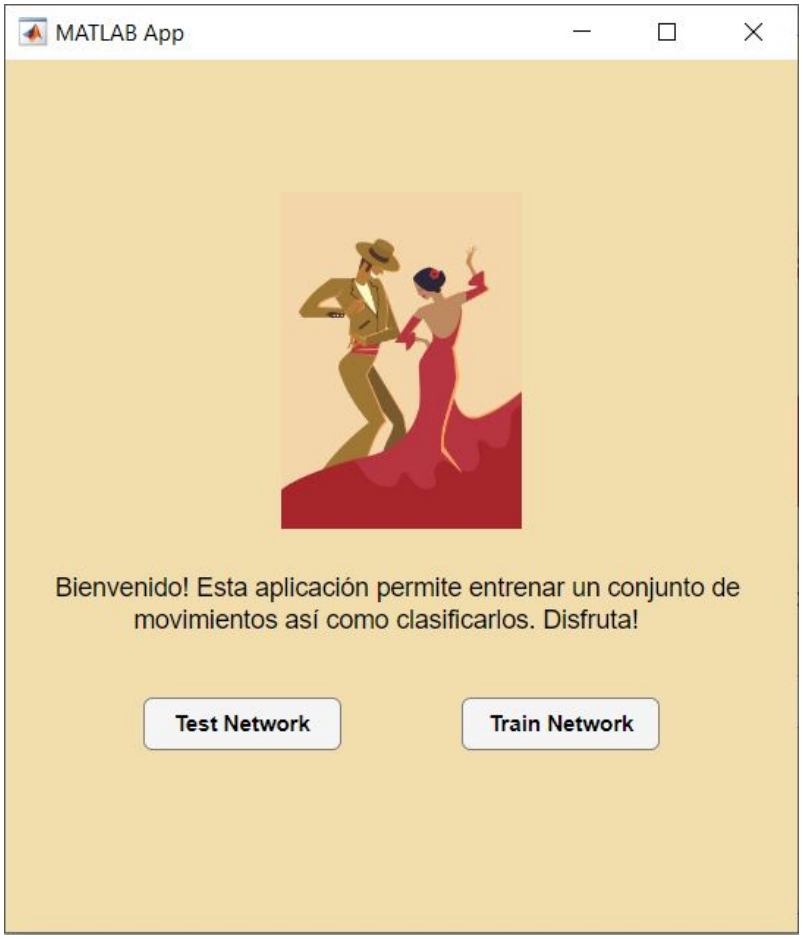

Figura 3.1: Pantalla de inicio.

Si se selecciona la opción *Train Network* (entrenar una red), se abrirá una nueva pantalla, como se aprecia en la Figura 3.2, que funciona de la siguiente manera.

Como se observa en la parte superior de la Figura 3.2, el primer paso consiste en seleccionar un fichero para cargar los datos de entrenamiento. Este se puede elegir o bien mediante el explorador de archivos, o bien seleccionando uno de los archivos predeterminados que se muestran en la interfaz. Entre estos últimos se encuentra el

fichero *HumanActivityTrain*, mencionado con anterioridad, que es el archivo empleado por defecto, con sus correspondientes variables XTrain (aceleraciones) e YTrain (categorías).

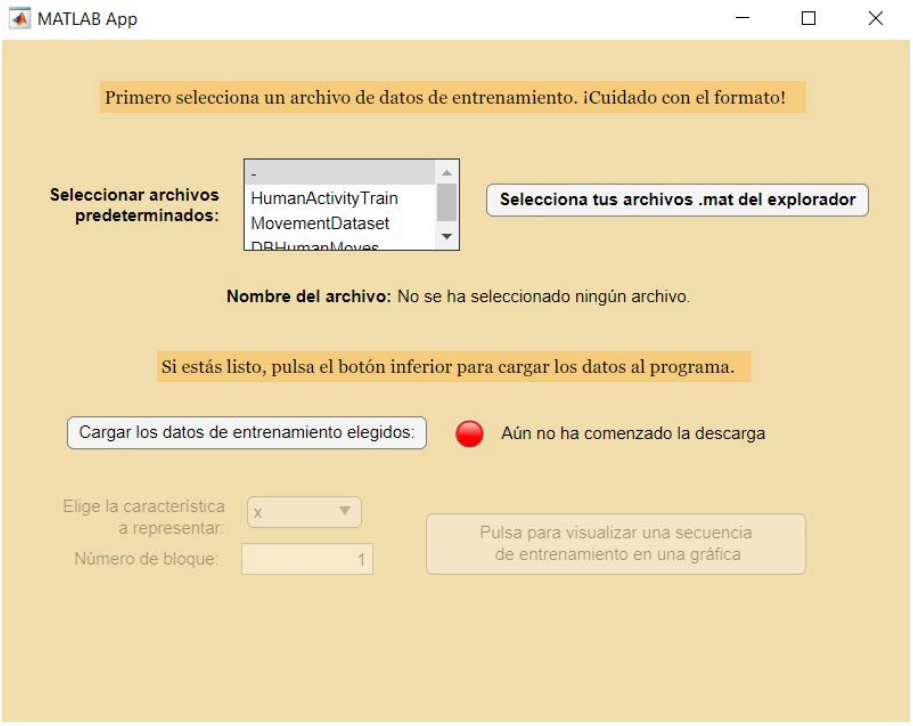

Figura 3.2: Pantalla de carga de datos de entrenamiento.

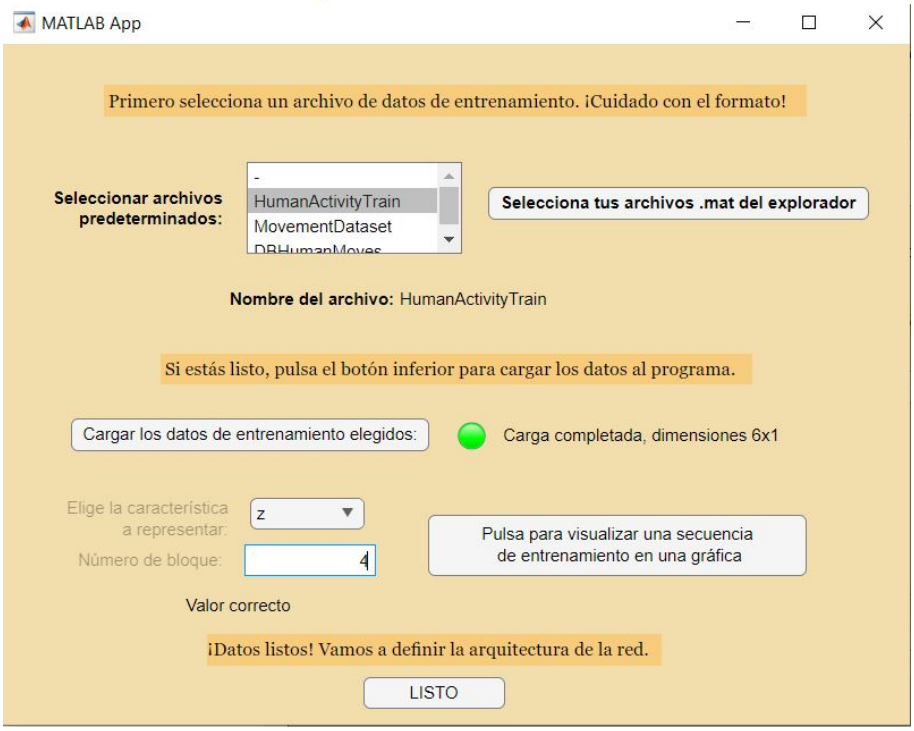

Figura 3.3: Fichero seleccionado y datos cargados.

En la Figura 3.3 se observa cómo al seleccionar un fichero, se actualiza un *label* con el nombre del mismo. De igual modo se contempla cómo evoluciona la interfaz (Figura 3.3) tras pulsar el botón «Cargar los datos de entrenamiento elegidos», que como su nombre indica, extrae los datos del fichero elegido (siempre que sea válido). Se habilita la parte opcional de representación y se puede avanzar a la siguiente

Además, en la parte inferior de la Figura 3.3 existe la zona de visualización de los datos, que se activa únicamente una vez cargados. A la izquierda, aparecen varios parámetros que se comentarán a continuación; a la derecha, un botón que, tras ser pulsado, hace que se despliegue una visualización de las secuencias de movimientos antes de ser entrenadas, tal y como se muestran en las gráficas de las figuras 3.4 y 3.5. Más específicamente, esta zona funciona de manera que al cargar el fichero, se obtiene el número de secuencias NS en los que vienen divididos los datos. A partir de este valor, se establece un rango $[1, NS]$. De entre estos valores, se selecciona una de las secuencias de entrenamiento (número de bloque en 3.3) de XTrain y, por otro lado se elige entre los atributos de aceleración (x, y o z) (característica en 3.3). Tras ello, se pulsa el botón y se despliega una pantalla secundaria, que representa la aceleración de la coordenada elegida frente al tiempo según el movimiento correspondiente. De esta manera, se permite visualizar y entender mejor la forma del *dataset* escogido.

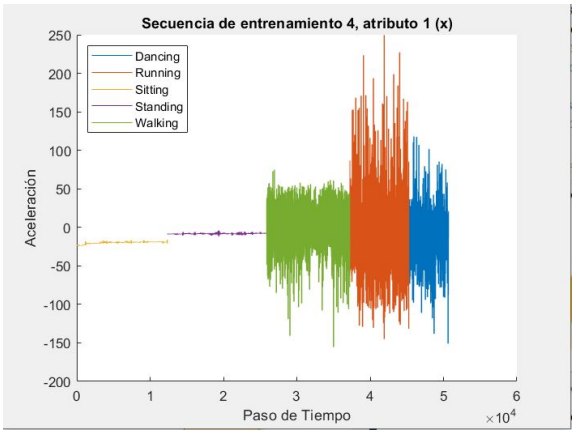

Figura 3.4: Secuencia 4, coordenada x.

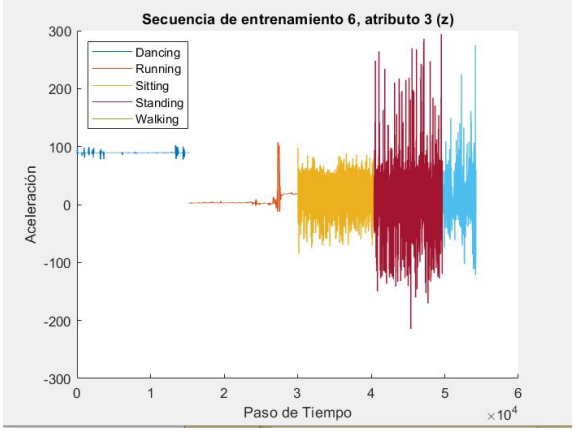

Figura 3.5: Secuencia 6, coordenada z.

3.2.1. Definir la arquitectura de red de LSTM

Una vez incorporados los datos con los que se entrena el modelo, se pasa a describir la arquitectura de la red LSTM utilizada.

Primeramente, la aplicación define una serie de **parámetros** que se van a utilizar en las capas de la arquitectura. Se establecen *numFeatures*, *numHiddenUnits* y *numClases*. Estos parámetros representan, respectivamente, que la entrada llega en secuencias de tamaño 3 (x,y,z), que el número de unidades ocultas de la capa LSTM es 200 y que el número de clases o categorías es 5 (movimientos).

Este aspecto no se verá reflejado en la interfaz, pues estos parámetros se establecen implícitamente, y por defecto, y no es necesario modificarlos para este trabajo. Cabe destacar que el número de unidades ocultas ciertamente influye en el entrenamiento de una red. En general, a mayor número de neuronas, mejores resultados (siempre que no se llege al sobreajuste), aunque esto implicará un mayor número de parámetros y tiempo de entrenamiento. Sin embargo, se reitera que en este caso no es necesaria su modificación.

3.2.2. Especificación de las capas

A continuación, se precisan y detallan las **capas**, ya introducidas conceptualmente en el capítulo anterior. Antes de empezar, cabe destacar que existen diferentes formas de definir cada capa, debido a propiedades opcionales que existen según los atributos de cada método, que se especifican mediante pares nombre-valor.

Respecto a la interfaz, ocurre lo mismo que en el apartado anterior. No se verá afectada por los cambios, pues las capas se establecen también implícitamente, al ser parámetros que no hace falta modificar para este trabajo.

En primer lugar, se establece la ***SequenceInputLayer***, que es una capa de entrada por la que ingresan datos secuenciales a la red neuronal, aplicando normalización de datos si es necesario. En este caso, solo se tiene como atributo el *InputSize*, que consiste en el tamaño de los datos de entrada, y por tanto se establece a *numFeatures*. En este ejemplo no se no se realiza un tratamiento previo o preparación de los datos de entrada antes del entrenamiento, pero se va a explicar cómo funciona.

La normalización de los datos tiene lugar cada vez que estos se propagan hacia adelante atravesando la capa de entrada y se aplica a todos, incluyendo los datos de relleno, garantizando uniformidad. Se selecciona mediante el atributo *Normalization* y hay varias opciones, destacando '*none*', que no aplica cambios, y *'zscore'*, que estandariza según la media *'Mean'* y la desviación típica *'StandardDeviation'*. Además, en las funciones *TrainNet* y *TrainNetwork*, el software calcula automáticamente estas estadísticas, aunque también se pueden especificar manualmente, en caso de querer ahorrar tiempo.

La siguiente es la ***LSTMLayer***, una capa RNN que aprende las dependencias a largo plazo de secuencias de datos en los distintos pasos de tiempo. Utiliza operaciones aditivas que contribuyen en la mejora del flujo de gradiente en secuencias largas durante el entrenamiento. La función presenta dos atributos, *numHiddenUnits* y el par nombre-valor *'OutputMode'-'sequence'*. En el segundo, se controlan las salidas de la capa, que pueden ser *'sequence'*, secuencia completa, o *'last'*, que solo genera como salida el último paso de la secuencia.

El número de unidades ocultas corresponde a la cantidad de información que la capa recuerda entre pasos de tiempo, que se conoce como **estado oculto**. En cuanto a sus características principales, esta información puede ser de cualquier paso previo.

Además, hay que tener cuidado de no fijar un número muy elevado, pues puede llevar a un sobreajuste de los pasos de la red con los datos de entrenamiento. Es decir, el modelo puede llegar a aprender demasiado bien los datos suministrados, en lugar de capturar patrones generales y dejar cierto espacio a características aleatorias. Por último, es destacable que el estado oculto no limita la longitud de las secuencias a procesar.

Acto seguido, se introduce la ***fullyConnectedLayer***, cuya función es multiplicar la entrada por una matriz de pesos y añadir un vector de sesgo. En esta función existe la propiedad obligatoria *OutputSize,* que fija el tamaño de salida y se establece al valor *numClasses*. Las filas de la matriz representan los pesos de las conexiones de entrada para cada neurona de la capa actual, mientras que el sesgo aporta adaptabilidad en el ajuste de las salidas sin importar las entradas. En ambos casos se determinan por defecto de la siguiente manera.

La función *WeightsInitializer* inicializa la matriz de pesos con el método *'glorot'*, mediante el método Glorot (también conocido como Xavier) que utiliza una distribución uniforme, una media cero y varianza $\frac{2}{T_E+T_S}$ siendo T_E el tamaño de entrada de la capa y T_S el de salida. Sin embargo, este no es el único, ya que existen otros métodos como *'he'*, '*orthogonal'* o *'zeros'*. Siguiendo el mismo método, la función *BiasInitializer* inicializa el vector de sesgos a *'zeros'*.

Los pesos y los sesgos de la capa son parámetros que se van modificando según el aprendizaje. Cuando se entrena una red, se pueden fijar los valores de las propiedades *weights* y *bias* que se usarán como iniciales, pero en el caso de que estén vacías, *TrainNet* and *TrainNetwork* usan los inicializadores especificados en las funciones mencionadas en el párrafo anterior.

Seguidamente, se tiene la capa ***softmaxLayer***, que aplica la función SoftMax a la entrada. Esta es una función de activación y se usa en la capa de salida de redes neuronales para la clasificación. Su funcionamiento consiste en convertir un conjunto de valores (salidas de la capa anterior) en probabilidades que suman uno y por tanto, es útil para conocer la probabilidad de que una instancia pertenezca a una clase

La última capa es la ***classificationLayer***, que se encarga del cálculo de la pérdida de entropía cruzada de las tareas de clasificación, es decir, cómo de bien coincide la salida con las etiquetas reales. La capa infiere el número de clases a partir del tamaño de salida de la capa anterior, siendo este el motivo por el cuál se incluye una capa *softmax* antes de la capa de clasificación.

3.2.3. Especificación de las opciones de entrenamiento

El código continúa con la declaración de la función ***trainingOptions*** y, por tanto, se establecen también los parámetros u opciones de entrenamiento. Todos los argumentos de la función siguen la estructura par-valor menos el que se explica

seguidamente. Esto se contempla en la Figura 3.6, que representa los parámetros modificables que se detallan y corresponden al umbral del gradiente, optimizador, razón de aprendizaje, número de épocas y tamaño de los minilotes.

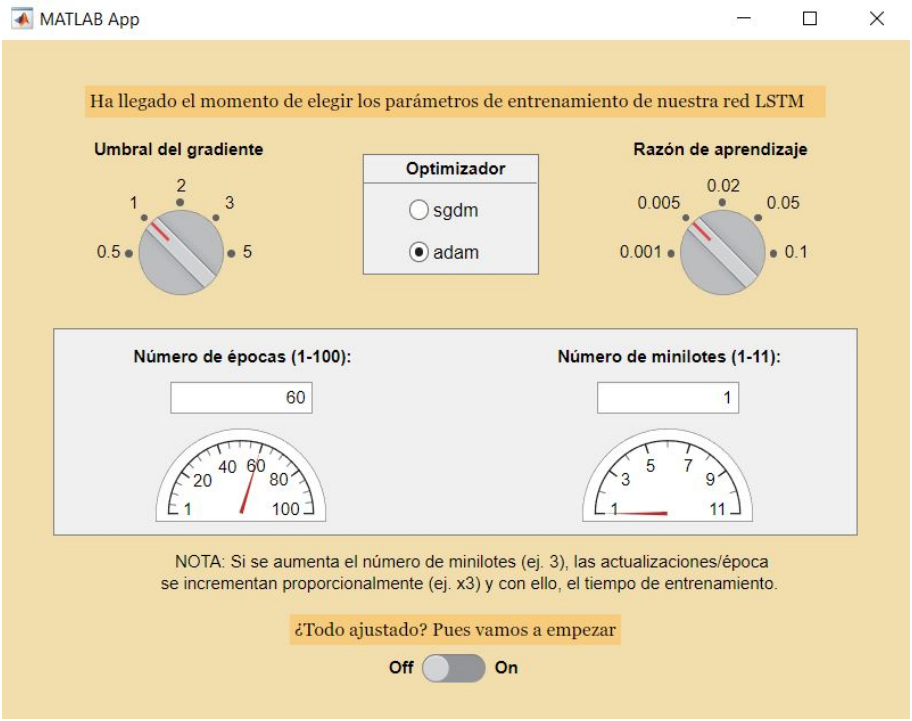

Figura 3.6: Pantalla de selección de parámetros.

Esta función tiene un argumento de entrada obligatorio *solverName*, que es el optimizador que se utiliza para entrenar la red neuronal. Se tienen tres opciones: *'adam'*, *'sgdm'* y *'rmsprop'*, siendo los dos primeros, ya explicados teóricamente en el capítulo 2, los que se utilizan en este trabajo, y entre los que se tendrá la opción de elegir en la interfaz (Figura 3.6). El tercero no es útil para este trabajo, pues suele utilizarse para otro tipo de modelos más enfocados en la predicción de los movimientos que en su clasificación.

Para establecer el número de minilotes, se tiene que hacer de manera indirecta, pues no es un parámetro modificable. Ya se vio en la parte teórica que el número de minilotes es igual al tamaño de los datos de entrada dividido entre el tamaño del minilote. Por tanto, como el tamaño de los datos de entrada es fijo, si se especifica el tamaño del minilote (Figura 3.6) mediante el argumento de par nombre-valor con nombre *'MiniBatchSize'* de *trainingOptions*, se tiene el número de minilotes deseado.

En cuanto a la razón de aprendizaje, se puede especificar la tasa de aprendizaje global (Figura 3.6) con el argumento de par nombre-valor con nombre '*InitialLearnRate'* de *trainingOptions*. Es posible modificar la tasa de aprendizaje cada cierto número de épocas multiplicándola por un factor, como vimos en la parte teórica del capítulo previo. En lugar de utilizar una tasa de aprendizaje pequeña y fija durante todo el proceso de entrenamiento, se elige una mayor al principio del entrenamien-

to, para reducir gradualmente este valor durante la optimización. De este modo, se puede acortar el tiempo de entrenamiento, a la vez que se dan pasos más pequeños hacia el mínimo de la pérdida a medida que avanza el entrenamiento.

Aunque en este trabajo no se ha incluido, y no se pueda ver reflejado en sus resultados, se va a explicar cómo se implementaría. Se emplea el argumento de par nombre-valor *'LearnRateSchedule'-'piecewise'*. Una vez seleccionado, *trainNetwork* multiplica la tasa de aprendizaje inicial por un factor de 0,1 cada 10 épocas. De hecho, también se puede especificar el factor por el que reducir la tasa de aprendizaje inicial, y cada cuántas épocas se realiza, usando los argumentos de par nombre-valor *'LearnRateDropFactor'* y *'LearnRateDropPeriod'*, respectivamente.

El siguiente parámetro a fijar es *'MaxEpochs'*, con un valor de 60 por defecto, y se refiere al máximo número de épocas. El ajuste de este hiperparámetro es importante y es una funcionalidad de la interfaz (Figura 3.6), con la que se comprueba la disparidad de los resultados según su valor, que se refleja en el rendimiento del modelo. Un número demasiado bajo puede significar un modelo que converge totalmente; sin embargo, tampoco puede ser muy elevado pues, además de que se prolongaría mucho el entrenamiento, podría provocar un sobreajuste.

También se eligen las gráficas que se desean visualizar durante el entrenamiento de red neuronal mediante *Plots* con el valor *training-progress*. El argumento *Verbose*, que se establece por defecto a 1, se pone a 0 para desactivar el indicador que muestra información sobre el progreso del entrenamiento.

El argumento restante es el correspondiente al umbral del gradiente *GradientThreshold*, que por defecto viene ajustado a un valor infinito pero en el trabajo se establece a 2 y es modificable (Figura 3.6). La función principal de este valor es evitar la explosión de gradiente y sus problemas asociados, para lo cual, si el gradiente supera el valor de *GradientThreshold*, este se recorta de acuerdo con la opción de entrenamiento *GradientThresholdMethod*, que está predeterminada a *'12norm'*. Al modificar el umbral, la velocidad a la que la red neuronal alcanza una solución óptima varía. Si es muy bajo puede retrasar demasiado el proceso de entrenamiento, mientras que uno muy alto puede resultar en una convergencia rápida pero de menor calidad.

El siguiente paso es entrenar la red *LSTM* con las opciones especificadas y la función *trainNetwork*. Como se contempla en la figura 3.6, solo hace falta activar el botón inferior de la pantalla a *'on'* y se dará comienzo al entrenamiento. Si se continúa con las opciones por defecto, cada minilote contiene el conjunto de entrenamiento completo, de manera que la gráfica (pesos) se actualiza una vez por época. Las secuencias son muy largas, por lo que se puede llevar un tiempo en procesar cada minilote y actualizar la gráfica.

Tras activar el *switch* en la figura 3.6, se desplegará una pantalla secundaria con los datos sobre la evolución del entrenamiento, que podemos ver en la figura 4.1. En el capítulo 4 se comentarán los resultados obtenidos y se explica dicha pantalla de entrenamiento.

3.2.4. Definir la arquitectura de red LSTM

En esta sección se explica la otra opción que se observa en la pantalla de inicio (Figura 3.1), es decir, el caso ***Test Network***. Se explica el funcionamiento de esta parte de la aplicación tanto interna como externamente, tal y como se ha hecho en las secciones anteriores. La idea general es cargar un conjunto de datos *test data* y clasificarlos según la actividad que se realiza en cada paso de tiempo. Visualmente, se entra tras pulsar el botón correspondiente en la interfaz, apareciendo una nueva pantalla representada en la figura 3.7 que, como se observa, está dividida en dos paneles, que se emplean para seleccionar la red entrenada y el conjunto de datos a testear.

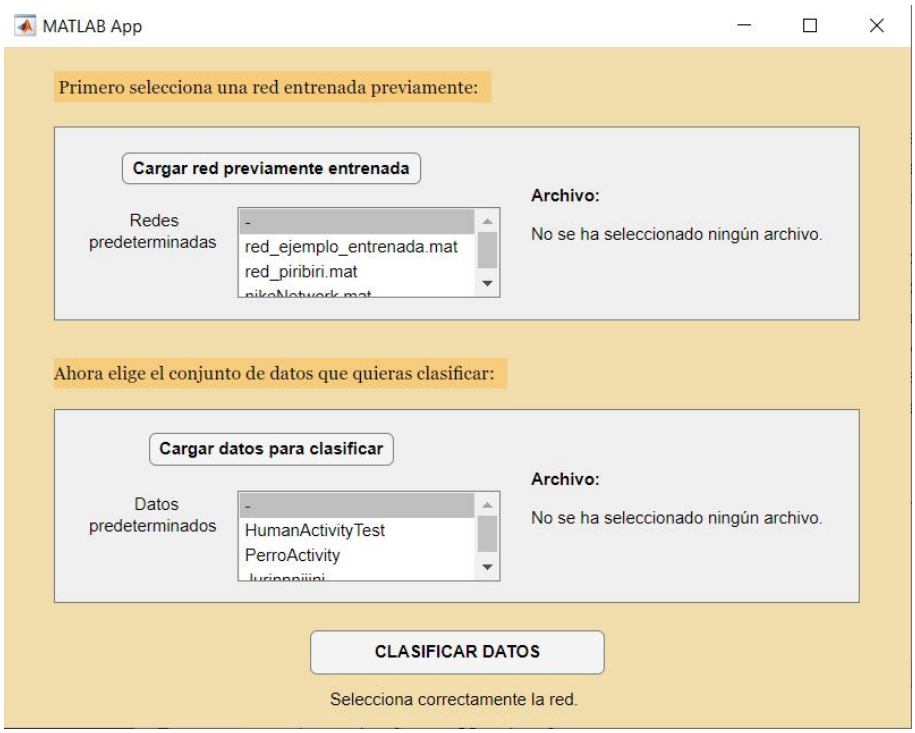

Figura 3.7: Pantalla de clasficación.

El panel de selección de la red, ubicado en la parte superior de la figura 3.7, aporta una doble función. En primer lugar, permite cargar una red previamente entrenada del explorador de archivos. Esto implica que el funcionamiento presupuesto de la aplicación consistirá primeramente en entrenar una red con la opción *Train Network* con su posterior almacenamiento. Tras esto, la aplicación cerrará las ventanas abiertas por ese proceso y se volverá a la pantalla de inicio (Figura 3.1), desde la cual se navegará hasta *Test Network* y ahí se elegirá dicha red ya entrenada. Por otro lado, también se admite la opción de elegir redes predeterminadas ya entrenadas, para una mayor comodidad y poder realizar pruebas del funcionamiento de la aplicación.

El otro panel de selección de la figura 3.7 sirve para cargar el fichero que contie-

ne los datos que se van a clasificar. Son las muestras con las que se comprueba el funcionamiento real de la red. El archivo que se va a emplear para la explicación se llama *HumanActivityTest* (aunque todos los ficheros que se usen deberían seguir esta estructura), y consiste en dos conjuntos denominados *XTest* e *YTest*, que contienen una secuencia de datos de dimensión tres (las tres coordenadas de aceleración) y una secuencia de categorías o etiquetas (uno de los movimientos posibles) correspondientes a la actividad que se realiza en cada paso de tiempo. Para una mayor comprensión de los datos que se emplean, se incluye la funcionalidad opcional para graficar los datos que se van a clasificar. En este caso, se ha optado por representar las tres coordenadas en una misma gráfica, pues solo se tiene una secuencia de datos y se considera más limpio. Los resultados se observan en la figura 3.8.

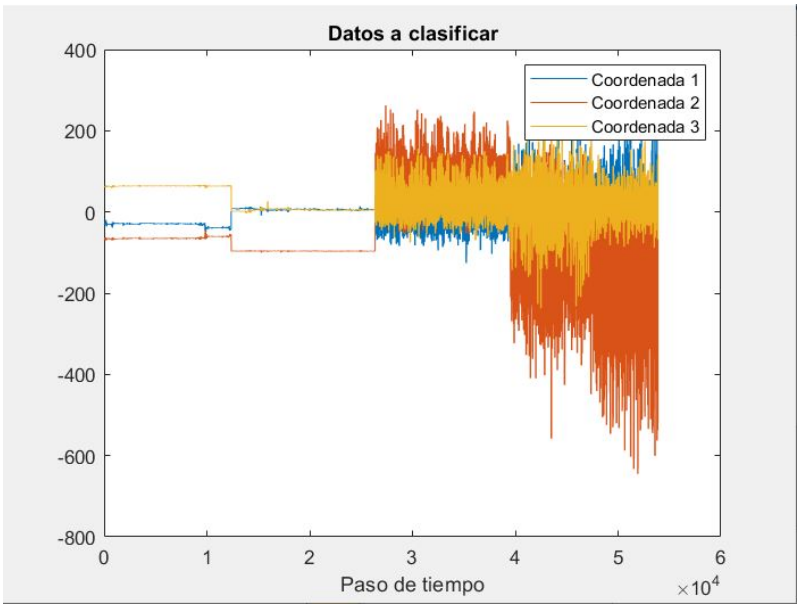

Figura 3.8: Gráfica de los datos que se van a clasificar.

Una vez se han seleccionado los dos archivos que se piden (Figura 3.7), se podrá pulsar un botón que sirve para desplegar una nueva pantalla 3.9 y comenzar la clasificación de los datos (si se pulsa cuando no están bien elegidos la red o los datos, no permite continuar e indica el fallo). Para ello, se emplea la función *classify* con los argumentos *net* y *features* que corresponden a la red entrenada y los datos de *YTest*. Esta función devuelve un conjunto de etiquetas (en el ejemplo *YPred*), que son las categorías que la red *net* predice de los datos que se han cargado para clasificar en cada instante de tiempo.

Para comprobar la efectividad del entrenamiento de la red, es común calcular la precisión. Esto se hace del siguiente modo: primeramente, se comparan los valores predichos por la red *YPred* con los valores correctos *YTest* en cada paso de tiempo. Si estos coinciden, se suma una unidad y este sumatorio se divide entre el número de elementos que se han clasificado. En otras palabras, se divide el número de aciertos entre el número total de clasificaciones. En la figura 3.9 se puede observar la fórmula y el resultado obtenido. Cuanto más cercano a 1 es el valor de *accuracy*,

mejor entrenada está la red, pues los aciertos se acercan más al número total de clasificaciones.

El problema es que de esta manera no se sabe en qué situaciones está fallando la red. Para ello se incluye un gráfico de comparación. En realidad, son dos gráficos superpuestos, mostrando los valores de *YPred* contra los de *YTest*. En el eje de abscisas se representan los pasos de tiempo, y en el de ordenadas las categorías del modelo. De esta manera, se puede observar visualmente qué movimiento ha fallado, es decir, cuál es el movimiento predicho erróneamente y cuál debería ser, y en qué momento del tiempo (elemento de la secuencia) ocurre dicho fallo. Todo esto se observa en las figuras 3.9 y 3.10, proporcionando una visión mucho más clara y comprensible para el usuario.

Figura 3.9: Pantalla de los resultados de clasificación.

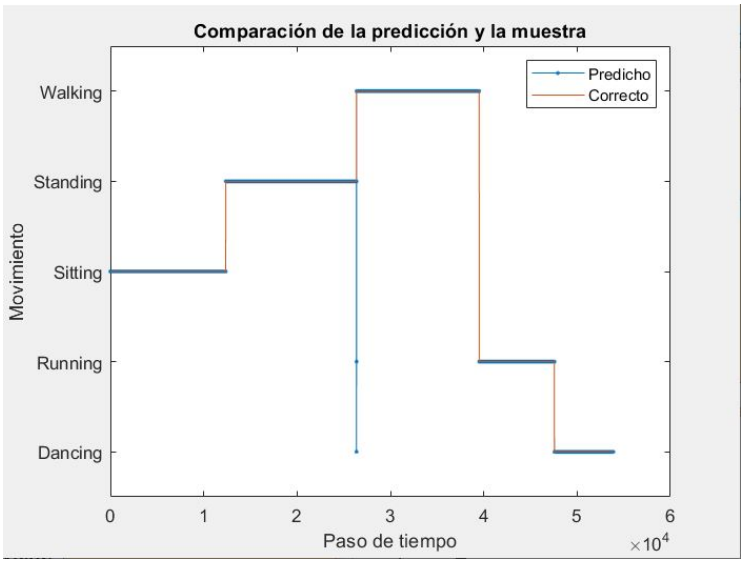

Figura 3.10: Pantalla de comparación de los resultados.

3.3. Clasificación dinámica de movimientos

En esta sección se explica cómo se pueden clasificar unos datos obtenidos instantáneamente a través de un dispositivo móvil. Para una mayor funcionalidad, se podría relacionar la aplicación estática desarrollada en la sección anterior con los datos de dispositivos inteligentes. En particular, se podrían obtener los datos de estos dispositivos, guardarlos en un archivo y luego enviarlos a la aplicación alojada en el ordenador para que puedan ser clasificados. Además, de un modo prácticamente idéntico a ese, se puede obtener un conjunto de datos para entrenar la red.

Pero ese no es el caso. Aquí, se explica cómo se obtienen los datos de estos dispositivos y cómo dichos datos son clasificados en tiempo de ejecución, gracias a una red previamente entrenada en la aplicación estática. El resultado es que obtenemos las categorías de los movimientos que hemos realizado hace apenas unos instantes.

La herramienta fundamental para esta conexión es el entorno de programación **MATLAB**, de MathWorks. En concreto se utiliza MATLAB Mobile, que es la aplicación móvil que complementa el entorno de MATLAB y permite ejecutar scripts y comandos desde dispositivos móviles dotados de sensores de movimiento, tales como smartphones o tablets.

La pantalla de la figura 3.11 es la que aparece al abrir la aplicación en el dispositivo inteligente, y funciona como línea de comandos. Dentro de MATLAB Mobile, si se pulsa en la esquina superior izquierda, se despliega un menú con varias opciones. Entre ellas, existe una pestaña llamada 'Sensores' en la que hay que activar una opción para transmitir en tiempo real a MATLAB (figura 3.12) y los sensores que se vayan a utilizar (figura 3.13). Para este trabajo se usará únicamente el acelerómetro, aunque hay otras opciones disponibles como la velocidad angular, la orientación, el campo magnético o la posición.

Figura 3.11: Pantalla de inicio de MATLAB Mobile.

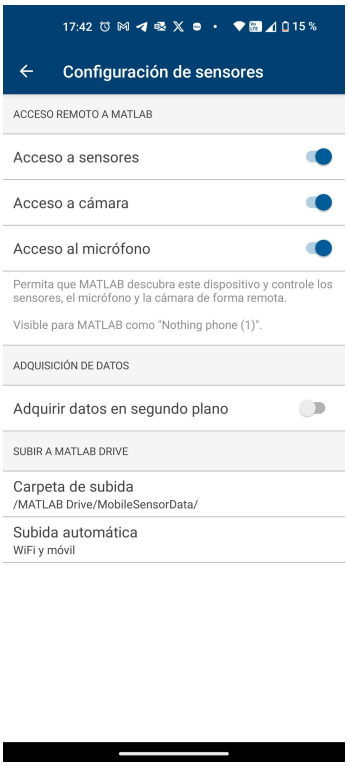

Figura 3.12: Configuración de los sensores.

La comunicación entre la aplicación móvil y MATLAB (en el ordenador) se realiza

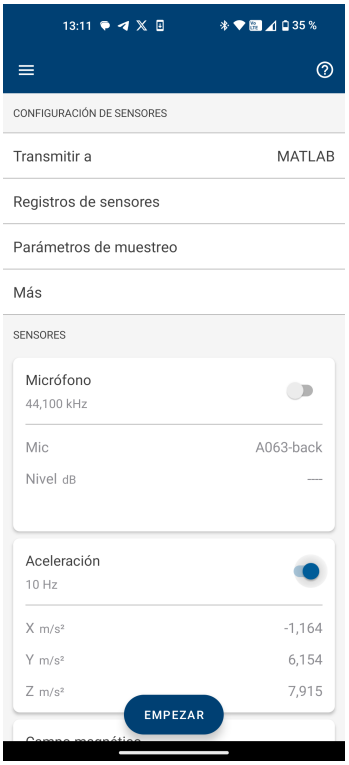

Figura 3.13: Sensor de aceleración activado.

a través de MATLAB Drive, cuyo símbolo es una nube (Figura 3.14). MATLAB Drive es un servicio en la nube que permite almacenar, acceder y colaborar en archivos de MATLAB desde cualquier lugar mediante conexión a Internet o de datos móviles. Es aquí donde se debe guardar todo fichero que se quiera ver o acceder desde el dispositivo móvil.

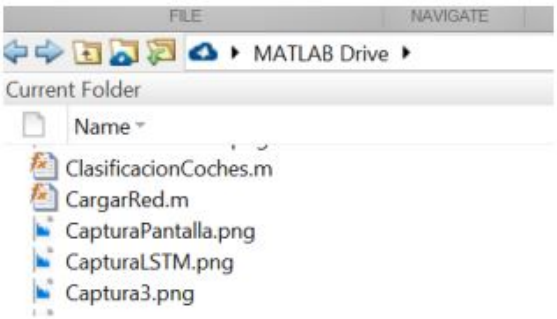

Figura 3.14: Símbolo de MATLAB Drive.

Para la correcta obtención de los datos, se usa un fichero que contiene una red ya entrenada por la aplicación estática (sección 3.2), por ejemplo, *RedLSTM.mat* y dos ficheros de código fuente llamados *InicioLSTM.m* y *ActividadLSTM.m*. Todos estos están ubicados en MATLAB Drive.

El primero de estos dos prepara el dispositivo y los datos de la siguiente manera: comprueba que no hay objetos previos activos para la captura de los datos sensoriales

y crea uno nuevo mediante la función *mobiledev*, que representa la conexión entre MATLAB y el dispositivo móvil. Hace lo mismo con la red y, si no existe, la vuelve a cargar mediante *load*. Este archivo se ejecuta cuando se han activado los sensores y se hace desde el dispositivo y una única vez.

A continuación se ejecuta *ActividadLSTM.m* tantas veces como se quiera. Este programa activa la captura de datos sensoriales durante el tiempo que se establezca (por defecto 10 segundos), y a continuación se recuperan los datos almacenados procedentes del dispositivo. Mientras discurren estos segundos, el usuario puede hacer cualquiera de los movimientos que se estudian en este trabajo, para luego comprobar los resultados. Pasado este tiempo, se obtiene una matriz que se denomina 'aceleracion' y que contiene los M movimientos capturados representados mediante M filas y $N = 3$ columnas (coordenadas x, y, z). Por último, se clasifican los datos capturados en 'aceleracion' mediante la función *classify* con la red preentrenada *RedLSTM.mat*.

Tras esto, se obtiene el resultado esperado. En la pantalla del dispositivo aparece una lista con las categorías que se han asignado a cada uno de los movimientos capturados (figura 3.15). Es importante mencionar que la red funciona mejor cuanto más largos sean los intervalos, es decir, si se cambia de movimiento cada segundo es posible que se encuentren fallos pues no son las series con las que se entreno la red.

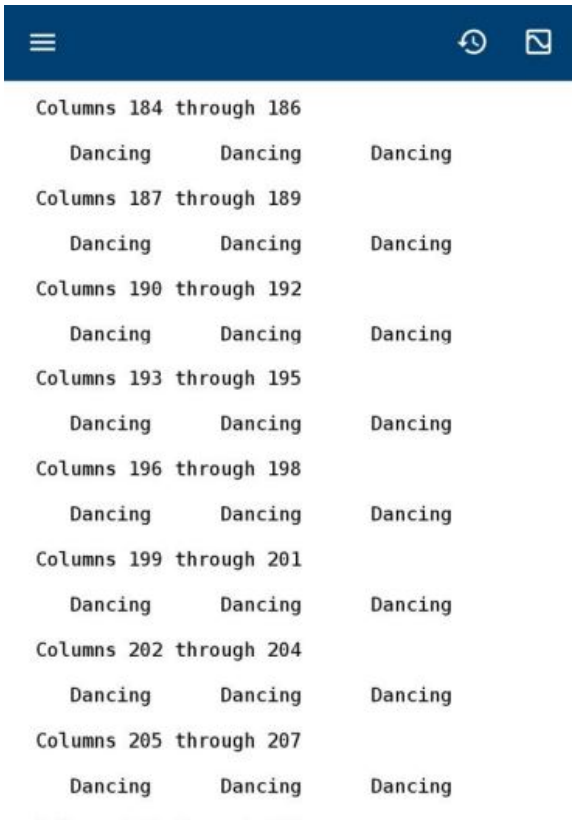

Figura 3.15: Resultados de la clasificación.

Capítulo 4

Resultados

Una vez se ha explicado la base teórica de las redes empleadas en el trabajo y se ha desarrollado el funcionamiento de la aplicación de clasificación de movimientos, es el momento de comentar cuáles son los resultados que se pueden obtener al usar la app y cómo se deben leer correctamente. Cabe destacar que los resultados que se muestran se refieren exclusivamente al proceso de entrenamiento, ya que los de clasificación son los indicados previamente.

4.1. Visualización del entrenamiento

Con fines ilustrativos, se analiza un ejemplo muy completo sobre cómo monitorizar el progreso de entrenamiento para redes entrenadas usando la función *trainnet*, es decir, se explica la gráfica de entrenamiento de la red para su mejor comprensión. El objetivo es explicar con detalle todos los conceptos posibles, aunque luego en los ejemplos de entrenamiento de este trabajo no se incluyan todas las posibilidades.

Cuando se entrenan redes para *deep learning*, monitorizar el progreso del entrenamiento es una tarea muy útil e importante pues aporta mucha información de cara a mejorar en futuras ejecuciones. La idea principal consiste en representar varias métricas que se van graficando durante todo el proceso. Algunas de las ventajas más comunes son que puede determinar si la precisión de la red está mejorando y con qué rapidez, y si la red está empezando a sobreajustar los datos de entrenamiento.

En el momento en que se establece la opción de entrenamiento *Plots* a *"training-progress"* para *trainingOptions* (mencionado en la sección anterior) y comienza el entrenamiento de la red, la función *trainnet* crea una nueva pantalla (figura 4.1), que muestra partes gráficas con distintas métricas de entrenamiento, que van variando en cada iteración. Cabe repetir que cada iteración consiste en una nueva estimación del gradiente y una actualización de los parámetros de la red. Si se especifican los datos de validación en *trainingOptions*, la figura también muestra las métricas de

validación cada vez que *trainnet* valida la red. A contunuación se describe con detalle la figura 4.1:

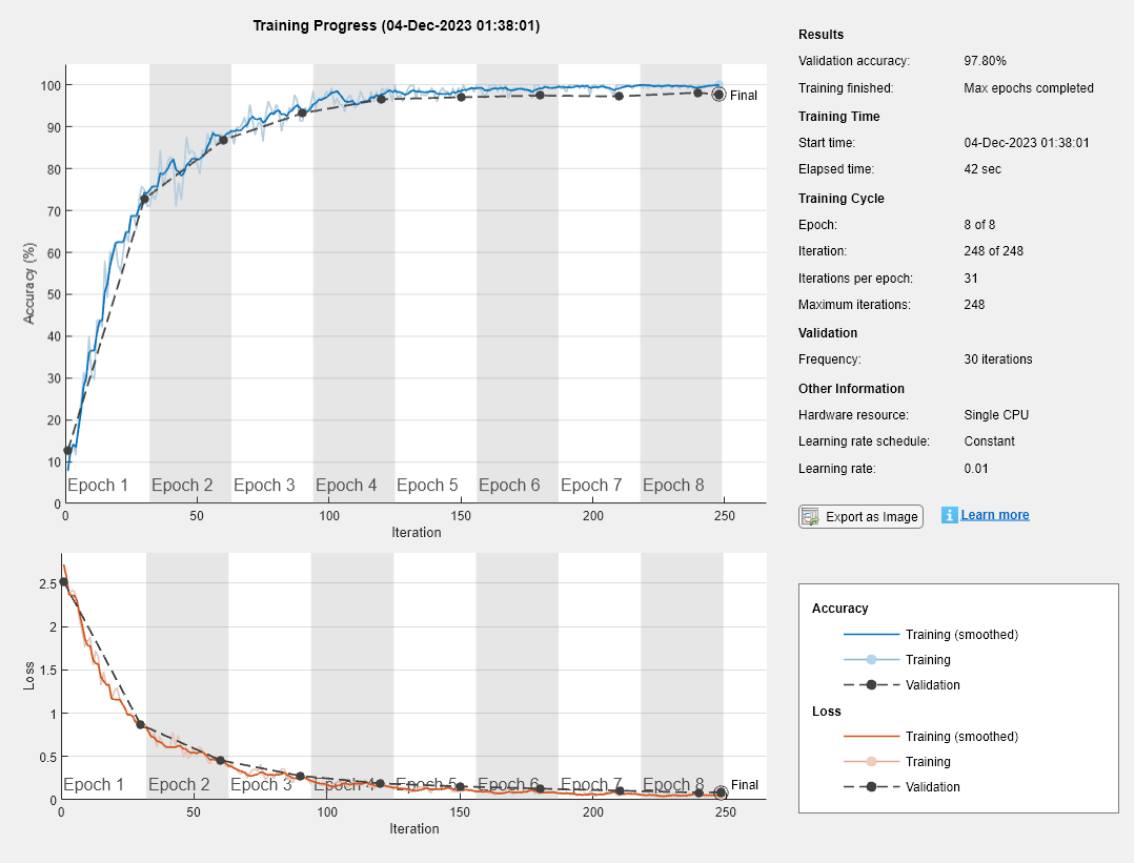

Figura 4.1: Entrenamiento general para explicación.

De manera predeterminada, y como se puede ver en la figura 4.1, la aplicación emplea una escala lineal para las gráficas (los valores aumentan o disminuyen de manera uniforme en incrementos iguales). Si se quiere utilizar una escala logarítmica para el eje Y (valores en el eje Y se representarían utilizando logaritmos en lugar de valores lineales), hay que seleccionar el botón de escala logarítmica en la barra de herramientas de los ejes.

Volviendo a la figura 4.1, el entrenamiento se puede detener y devolver el estado actual de la red en cualquier momento, tan solo haciendo *clic* en el botón de stop de la esquina superior derecha. Esta funcionalidad es muy útil, sobre todo porque es muy común que la precisión se estabilice antes del final del entrenamiento y se quiera dejar de entrenar, pues está claro que no mejorará más. Una vez que haya hecho *clic* en el botón de *stop*, es posible que la red tarde un poco antes de finalizar pues estará terminando ciertos cálculos de entrenamiento. Una vez completado, *trainNetwork* devuelve la red entrenada.

Otra característica de la figura 4.1 son que marca cada Época de entrenamiento con un fondo sombreado (las columnas blancas y grises). Cabe recordar que una época es un paso completo por el conjunto de datos.

La figura 4.1 representa los siguientes datos en las dos gráficas que se observan, teniendo ambas en el eje X las iteraciones y en el eje Y, el porcentaje de precisión (superior) y la pérdida (inferior):

- Precisión del entrenamiento (gráfica superior): precisión de la clasificación en cada minilote individual.
- Precisión de entrenamiento suavizada (gráfica superior): se obtiene aplicando un algoritmo de suavizado a la precisión del entrenamiento. Hay menos ruido que en la precisión sin suavizar, lo que facilita la detección de tendencias.
- Precisión de validación (gráfica superior): precisión de la clasificación en todo el conjunto de validación.
- Pérdida de entrenamiento, pérdida de entrenamiento suavizada y pérdida de validación (gráfica inferior): la pérdida en cada minilote. La capa final de la red es una classificationLayer, por lo que la función de pérdida es la pérdida de entropía cruzada.

Una vez se ha terminado el entrenamiento, se puede observar la precisión de validación y el motivo del final en la esquina superior derecha de la figura 4.1. Este último en el apartado *Training finished:*, que en condiciones normales es *Max epochs completed* (si se ha completado el entrenamiento). Además, durante la elección de parámetros podíamos elegir dos opciones para analizar las métricas; si la opción de entrenamiento *OutputNetwork* está en *"last-iteration"* (por defecto), las métricas finales corresponden a la última iteración de entrenamiento. Si en cambio *OutputNetwork* está en *"best-validation-loss"*, las métricas finales corresponden a la iteración con la pérdida de validación más baja. La iteración a partir de la cual se calculan las métricas de validación finales se etiqueta como *Final* en las gráficas (en este caso se puede observar en la figura 4.1 cerca de la iteración 250).

El resto de la información de la parte derecha, que corresponde a *Results*, *Training Time*, *Training Cycle*, *Validation* y *Other information*, se explicará en cada ejemplo con sus correspondientes valores.

Por último, un caso especial es que la red que se usa contenga capas de normalización de lotes (cosa que no ocurre en nuestro trabajo, pero es interesante), la métrica de validación final puede ser diferente a la métrica de validación evaluada durante el entrenamiento. El motivo de esto es que las estadísticas de la media y la varianza utilizadas para la normalización de lotes pueden diferir después de completar el entrenamiento.

La validación es una etapa muy importante en el entrenamiento (aunque no esencial) que todavía no se ha tratado en este trabajo y por tanto, se describe ahora brevemente. Los datos de validación, por ejemplo ($XVal$, $YVal$), se utilizan para monitorizar y prevenir el sobreajuste. El conjunto de datos de validación no se utiliza en el proceso de entrenamiento como tal, sino que sirve como un conjunto independiente para evaluar el rendimiento de la red. Periódicamente, es muy común que

después de cada época tenga lugar dicha evaluación, que implica hacer predicciones en los datos de validación y calcular métricas relevantes para controlar y evitar que la red aprenda demasiado.

4.2. Ejemplos de entrenamiento

A continuación se muestran tres ejemplos de entrenamiento realizados mediante la aplicación desarrollada. Es importante mencionar que el objetivo de este trabajo no es el análisis del entrenamiento para encontrar los mejores parámetros y conseguir un entrenamiento óptimo. En cambio, con este apartado se pretende aportar una visión general de cómo se desarrolla el entrenamiento de la red y obtener evidencias de que la aplicación funciona correctamente.

4.2.1. Ejemplo de entrenamiento 1

La gráfica que se comenta en este primer ejemplo es consecuencia de una ejecución de la aplicación con lo mencionado en la sección 3.2, es decir, con la elección de los parámetros modificables de la aplicación. Para este primer ejemplo, tras navegar por las pantallas de *TrainNetwork* por la app (figura 3.6) se han elegido los siguientes valores:

- *solverName* = 'adam'
- *MaxEpochs* = 60
- *GradientThreshold* = 2
- *InitialLearnRate* = 0.001
- *MiniBatchSize* = 128

En la gráfica generada tras el entrenamiento de este primer ejemplo (figura 4.2) podemos observar las métricas de precisión (azul) y pérdida (rojo) para el entrenamiento tanto suavizadas (color claro), como no (oscuro). La diferencia con lo visto en el apartado 4.1 es que no aparecen las métricas correspondientes a la validación. Esto se debe a que este concepto no se ha incluído en el entrenamiento de las redes de este trabajo.

Respecto al desarrollo de la gráfica, a medida que se van produciendo las iteraciones, la precisión (*Accuracy (%)*) aumenta de forma logarítmica, a la vez que la pérdida (*Loss*) disminuye, de manera inversa al aumento a la precisión. Es destacable que las métricas suavizadas son prácticamente idénticas a las que no lo están. También se observa que el campo *Training finished* tiene el valor *Max epochs completed*, lo que implica que la ejecución ha sido completada correctamente.

Además en las últimas iteraciones, vemos que el modelo se aproxima a un 100 % de precisión, y una pérdida 0. Esto quizás sea la característica más importante de esta gráfica, pues se observa que en las últimas iteraciones (a partir de la 50) se obtiene prácticamente beneficio nulo por lo que sería una buena idea cortar el entrenamiento en dicha iteración. Esto se traduciría en primer lugar en un menor tiempo de entrenamiento (con resultados prácticamente iguales) e incluso puede evitar un posible sobreentrenamiento.

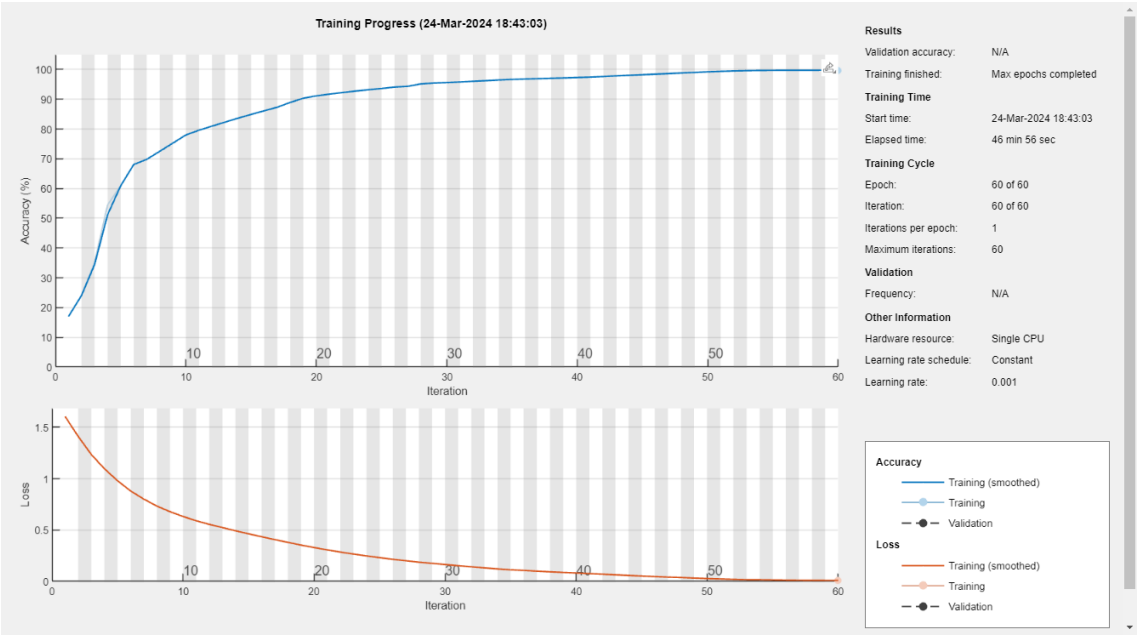

Figura 4.2: Primer ejemplo de entrenamiento de la aplicación.

4.3. Ejemplo de entrenamiento 2

Para el segundo ejemplo de entrenamiento, se produce una segunda ejecución de la aplicación con cambios en la elección de los parámetros. Para este caso se ha cambiado el *solver* y el *InitialLearnRate*, que se mantiene constante durante todo el entrenamiento por defecto.

- *solverName* = 'sgdm'
- *MaxEpochs* = 60
- *GradientThreshold* = 2
- *InitialLearnRate* = 0.01
- *MiniBatchSize* = 128

Tras el entrenamiento de este ejemplo (figura 4.3) se aprecian ambas gráficas algo más abruptas, haciendo referencia a que existen algunos cambios de pendiente no presentes en el anterior ejemplo. En cuanto a la gráfica de la precisión, crece de manera más rápida en las iteraciones 0 a 10, llegando a una precisión del 100 % de manera más lenta en las iteraciones siguientes, de la 20 a la 60. En cuanto a la gráfica de la función de pérdida, se puede apreciar una gráfica muy similar al ejemplo anterior, variando en todo caso debido a la escala mostrada, algo diferente debido al escalado de la imagen.

En este caso también se completa el entrenamiento sin interrupciones, aunque se observa que este dura cerca de 15 minutos más. Realmente esto no indica nada, pues lo más probable es que dependa de factores ajenos al entrenamiento. Por otro lado, es cierto que las últimas iteraciones son menos signficativas en cuanto a mejora con respecto a las primeras (como en el apartado anterior), pero es cierto que en este ejemplo quizás sería más conveniente retrasar el corte mencionado en el ejemplo anterior hasta la iteración 55 más o menos.

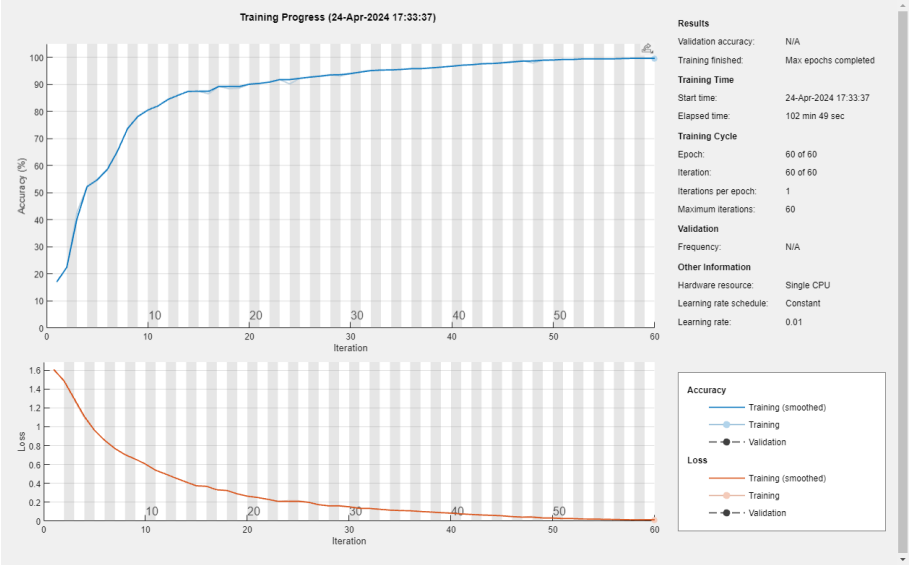

Figura 4.3: Segundo ejemplo de entrenamiento de la aplicación.

4.4. Ejemplo de entrenamiento 3

Para el tercer y último ejemplo, se ha decidido realizar una ejecución más corta para ver mejor los cambios de pendiente en las gráficas. Durante la preparación del entrenamiento, y para conseguir el objetivo mencionado, se han modificado los parámetros *MaxEpochs* y *MiniBatchSize*, obteniendo:

- *solverName* = 'sgdm'
- *MaxEpochs* = 30
- *GradientThreshold* = 2
- *InitialLearnRate* = 0.01
- *MiniBatchSize* = 64

Las decisiones tomadas han tenido el efecto deseado en las gráficas generadas en el último ejemplo de entrenamiento (figura 4.4). Se tiene una gráfica muy similar a la del segundo ejemplo pero con la escala al doble. Al fijarse en las 14 primeras iteraciones se observan los cambios de pendientes de una manera mucho más clara y además, se ve la métrica de precisión suavizada levemente por debajo de la otra en estas situaciones de cambio brusco. A partir de la iteración 15, el entrenamiento sigue con un trazado sin mucha perturbación ni pendiente. Es claro que ahora faltan iteraciones pues la precisión alcanzada en la última iteración no se acerca tanto al 100 % como se desearía.

Destaca también la reducción en el tiempo de ejecución, 10 veces menos que el ejemplo 2 y, realmente, la diferencia en la precisión final es mínima. En un trabajo que estudiase el mejor entrenamiento, sería interesante valorar también la eficiencia.

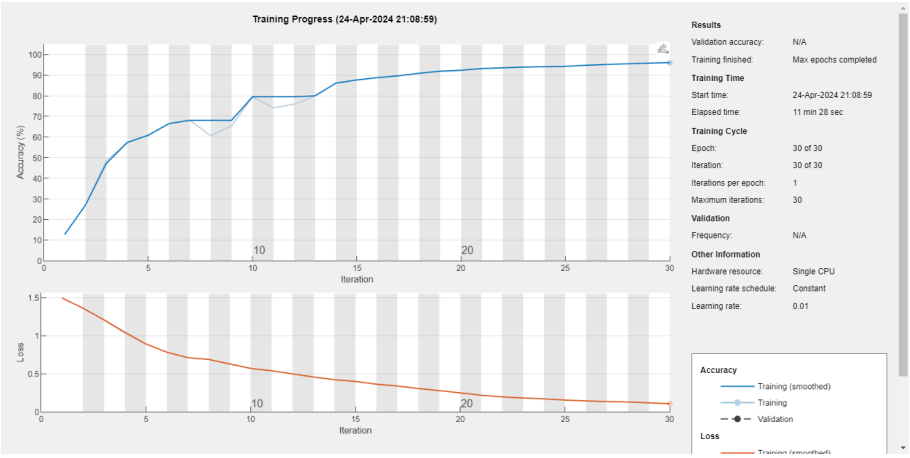

Figura 4.4: Tercer ejemplo de entrenamiento de la aplicación.

Capítulo 5

Extensión de la aplicación hacia IoT

En este capítulo se realiza una ampliación del trabajo que consiste en la introducción del concepto de **Internet de las Cosas** (**Internet of Things**, **IoT**) y su aplicabilidad en el proyecto.

La idea subyacente en relación a la ampliación hacia **IoT** estriba en la posibilidad de establecer comunicación entre un dispositivo móvil, que captura acciones de movimento y una aplicación en la "nube"de forma que los datos capturados y analizados mediante los modelos LSTM pueden almacenarse, procesarse e incluso interpretarse en la nube. De esta forma, cabría la posibilidad de que un usuario pueda visualizar datos procesados y enviados por otra persona, que es la portadora del dispositivo. Por ejemlo, se podría pensar en monitorizar de forma remota la actividad de una persona realizando rehabilitación, o vigilar el movimiento de una persona con movilidad reducida. Así pues, este es el sentido de la extensión de la aplicación que se desarrolla en el presente capítulo.

5.1. IoT y ThingSpeak

Para entender cómo funciona la conexión entre el dispositivo móvil (con su capacidad sensorial) y la nube se emplea la plataforma *ThingSpeak* de *MATLAB Analytics*, que recibe datos desde sus dispositivos (llamados *Smart Devices*, como *doorbells* o relojes), crea visualizaciones instantáneas de datos en vivo y puede enviar alertas utilizando servicios web como *Twitter* o enviar datos de vuelta a los dispositivos. Desde la nube se pueden procesar los datos y por tanto tomar decisiones según las distintas funcionalidades asociadas en cualquier momento.

ThingSpeak tiene una interfaz para cada *Smart Device* y emplea canales para para almacenar los datos crudos o procesados externa o internamente (en la propia plataforma). Los canales tienen capacidad de transferencia de la información a través de las comunicaciones establecidas entre la plataforma y los dispositivos, o incluso

entre cuentas de usuarios diferentes dentro de la propia plataforma.

Cada canal cuenta con un identificador numérico único para su referencia unívoca (*Channel ID*) y otras características opcionales tales como una descripción de su utilidad o la capcidad de asociación con *GitHub* y *YouTube*. Dentro de cada uno de estos canales, se pueden tener varios campos de datos (*Fields*) que representan los distintos datos crudos o procesados (icluyendo métricas asociadas) que se están monitorizando. En el siguiente apartado se verá la estructura particular del canal creado para este proyecto. Además, *ThingSpeak* permite ver y analizar datos en la plataforma mediante *MATLAB Visualization*.

Quizás la parte más interesante de *ThingSpeak* no sea el hecho de almacenar y visualizar datos, sino que permite configurar acciones automatizadas basadas en los datos recibidos mediante una app integrada llamada *React*. Para su funcionamiento se emplea otra aplicación también integrada que se denomina *MATLAB Analysis* y que permite escribir los fragmentos de código que luego serán ejecutados cuando se activen los *reacts*.

Por otro lado, el programa utiliza claves *API* para autorizar el acceso a sus servicios. Estas son esenciales para que la comunicación dispostivo-plataforma se realice de manera segura y controlada. Para cada canal en particular, existen dos tipos principales de claves *API* en *ThingSpeak*, la de escritura (*Write API Key*) y la de lectura (*Read API Keys*), que se utilizan para enviar datos a los canales y acceder a ellos respectivamente. Además, el usuario cuenta con dos claves propias, *User API Key*, necesaria para crear y borrar canales y poder acceder a la información privada de estos. La otra clave es *Alerts API Key*, que sirve para enviar *emails* a la cuenta de correo asociada al perfil.

5.2. Aplicación IoT en el análisis de acciones de movimiento

El objetivo de esta sección es describir el código correspondiente en *MATLAB* para representar de la mejor manera posible la aplicabilidad del IoT a este trabajo. Para ello se ha decidido captar una serie de movimientos desde un *Smart Device* y visualizarlos en la nube. Tras la recepción de los datos en la plataforma, éstos se procesan y mediante la activación de varios *Reacts*, se cuentan todos los movimientos clasificados como *Sitting* y si en alguna de las ejecuciones este número es mayor que la mitad del total de movimientos capturados, entonces se manda un correo al usuario por falta de actividad.

Como se ha indicado en los capítulos precedentes, son varias las acciones que se identifican mediante el modelo LSTM, siendo la de *Sitting* una de ellas. Se elige esta actividad para ilustrar el procesamiento de los datos y la comunicación entre ThingSpeak y otros dispositivos. Para cualquiera del resto de acciones se puden diseñar procesos asociados de la misma naturaleza.

También, con carácter ilustrativo se toman dos decisiones (propias de cada ejecución) para facilitar la comprensión al lector del proceso implementado. En primer lugar, se decide capturar 20 segundos, mientras que la idea sería aplicar este código a periodos largos de tiempo (como 8 horas) para llegar a una utilidad real. En segundo lugar, la ejecución de ejemplo que se ilustra con imágenes a continuación será un caso trivial en el que se permanece sentado durante toda la captura de datos. De esta manera el número de movimientos se corresponde con la acción de *Sitting*.

Antes de empezar la ejecución, hay que preparar el entorno de *ThingSpeak*. Se comienza con la creación de un canal *RedesLSTM* con 8 campos de valor (*Fields*), estando los 3 primeros destinados para almacenar los datos crudos de aceleración según los ejes X, Y, Z capturados (figuras 5.1): *Aceleración X*, *Aceleración Y* y *Aceleración Z*. Además, estas tres componentes de aceleración se seleccionan con la opción *dynamic*, que hace que los gráficos integrados de *ThingSpeak* se actualicen automáticamente en la cantidad de segundos o resultados que se establezca. En este caso se eligen 190 resultados, pues se van a tomar datos durante 20 segundos y se suelen adquirir entre 190-200 datos de aceleración de cada coordenada por ejecución. Por tanto, cada vez que se reciben 190 datos en el canal, y por tanto en los campos asociados, la gráfica se actualiza en cada ejecución, permitiendo visualizar dichos datos.

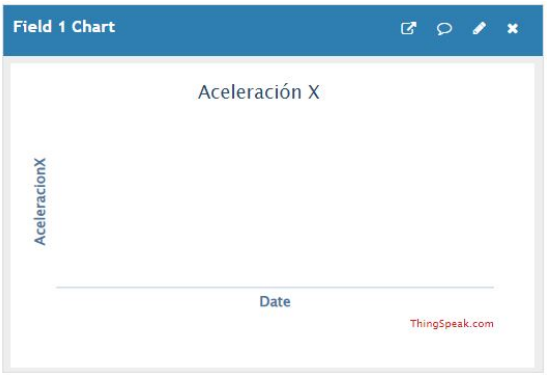

Figura 5.1: Campo *aceleración X* del canal.

Los datos que se visualizan en las tres gráficas siguientes (figuras 5.2 y 5.3) muestran el número de ejecuciones, el número de movimientos capturados por ejecucuón y el número de acciones de *Sitting* capturados por ejecución.

Los otros dos campos visualizados gráficamente (figura 5.4) son *Clasificar LSTM* y *Activar React Clasificación* y sirven para activar los *Reacts* y con ello la clasificación. Por otro lado, hay que activar en *MATLAB Mobile* el sensor de aceleración para la captura de datos, tal y como se explicó en la sección 3.3.

El código comienza con la creación de un objeto *m* mediante *mobiledev*. Este sirve para la lectura de datos del sensor de cualquier dispositivo que esté ejecutando *MATLAB Mobile* con la misma cuenta de *MathWorks*. Cuando hay varios dispositivos conectados a la cuenta, se puede añadir un argumento *devname* a *mobiledev* que permite leer datos del dispositivo indicado por *devname*. A continuación se carga

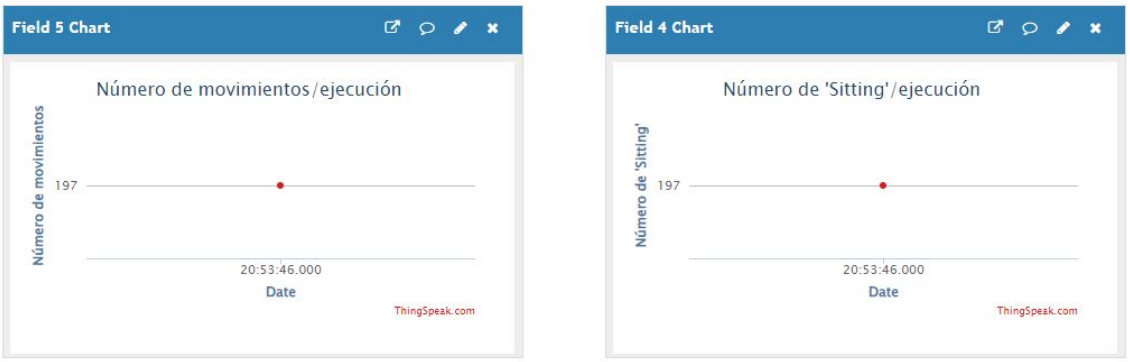

Figura 5.2: Nº de movimientos y *Sitting* por ejecución (caso trivial).

Figura 5.3: Ejecución número 2 (máx. 5).

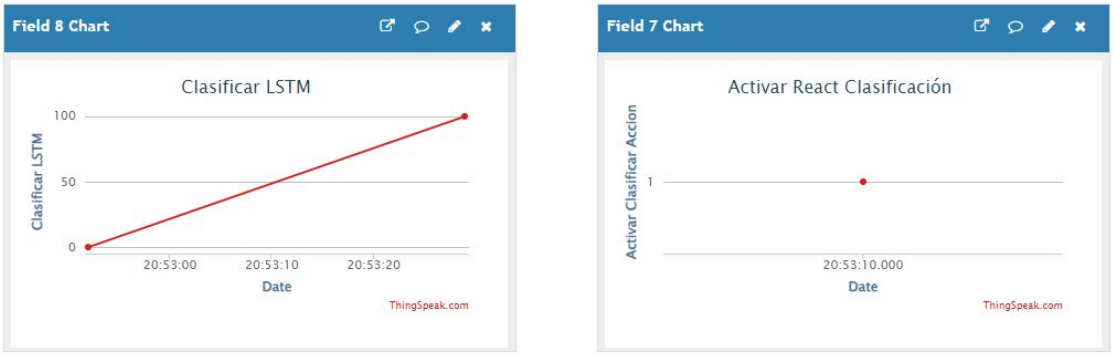

Figura 5.4: Representación de los *flags* una vez activados.

el modelo de red LSTM que se quiera utilizar mediante el comando *load*. Esto está pensado de manera que se cargue una red que haya sido entrenada en la aplicación estática desarrollada en el proyecto.

El siguiente paso es incluir en el código el ID y las claves de escritura y lectura propias del canal. Como paso previo, se comprueba el número de ejecuciones que se ha hecho leyendo el *Field Chart* correspondiente. Si ya se han contabilizado 5 ejecuciones, se emplea la función *delete* a través de *webwrite* para limpiar el canal completo. Si el número es menor que 5, se suma 1 y se comienza la ejecución. Esto

evita que el canal reciba demasiados datos y de esta manera se consigue que sea menos propenso a fallos.

Con esto, se activa la habilitación para la captura de datos sensoriales mediante la función *Logging* propia del objeto *m*. El tiempo de captura es variable (se prefija a 20 segundos) y por limitaciones de *MATLAB* se tiene que establecer acumulando pausas de 2 segundos. La función *accellog(m)* captura los datos según la siguiente estructura: [datos registrados (de aceleración), marcas de tiempo]. En concreto, se emplea la funcion *log* con el prefijo *acel*, por lo que de la misma manera se puede emplear *angvel* o *pos* para la velocidad angular y la posición. Tras esto, se desactiva *logging* y se dejan de capturar datos.

A continuación hay que preparar los datos antes de enviarlos a la nube para así poder representarlos correctamente. Se construye una tabla con 4 columnas que son los tiempos de captura y las 3 coordenadas de aceleración correspondientes a cada intervalo de tiempo. Con esto ya se envían los datos mediante *thingSpeakWrite* y se establece una pausa de 15 segundos. Esta se introduce cada vez que se accede al canal en modo escritura y se debe a la licencia de estudiante de *MathWorks*, que puede dar fallos por envío de peticiones muy continuado, así como para garantizar la consolidación de los mismos en la plataforma remota. Los datos recibidos en la nube son representados como se muestra en las figuras 5.5, 5.6 y 5.7.

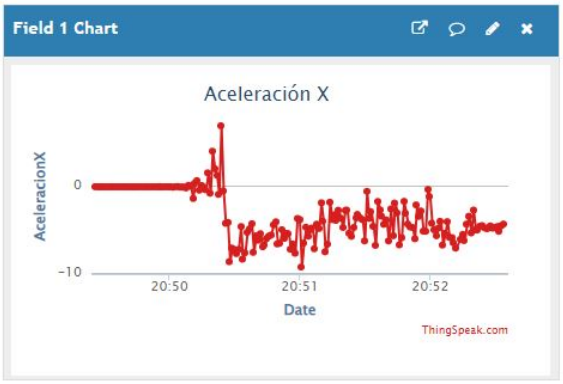

Figura 5.5: Aceleración X.

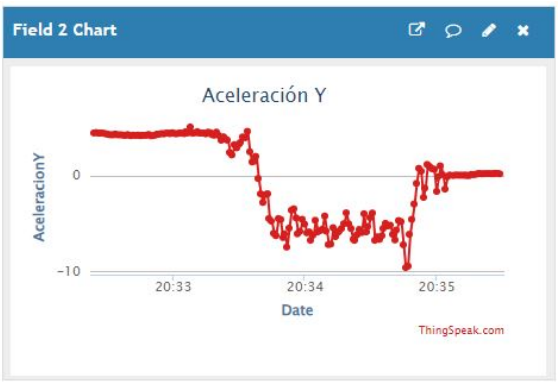

Figura 5.6: Aceleración Y.

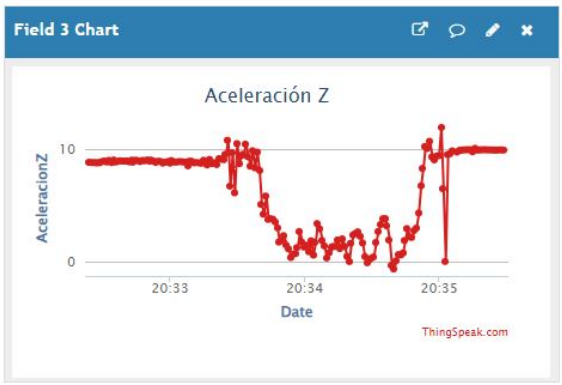

Figura 5.7: Aceleración Z.

Como comprobación de que los datos se han almacenado correctamente en la plataforma, se emplea *thingSpeakRead* para recuperar los datos de la nube. Esta función además devuelve información relativa al canal sobre el que se ha realizado la lectura. La verificación consiste en ver que el tamaño de los datos leídos corresponda con el número de datos enviados previamente.

La siguiente sección del código se enfoca en el funcionamiento de los *Reacts*, mediante la escritura de varios *flags* que se van escribiendo en los 2 canales de activación mencionados previamente. La figura 5.8 muestra los dos *Reacts* disponibles.

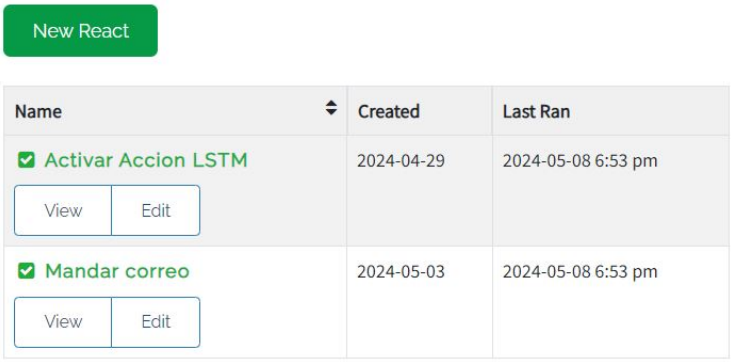

Figura 5.8: *Reacts* disponibles.

El proceso de clasificación comienza con la escritura de un 0 en el campo *Clasificar LSTM*. Tras esto, se escribe un 1 en *Activar Clasificar Acción*, que desencadena el *react Activar Accion LSTM*, como se observa en la figura 5.9. Esta respuesta automatizada comprueba que hay un 0 escrito en *Clasificar LSTM* y si es así, escribe un 100 en ese mismo campo. Mientras tanto, el código permanece en escucha de lectura (mediante un bucle) hasta que el *React* escriba el valor 100. En ese momento comienza la clasificación de las acciones subyacentes en las componentes de aceleración recibidas previamente, cuyo resultado se mostrará en el *Smart Device*.

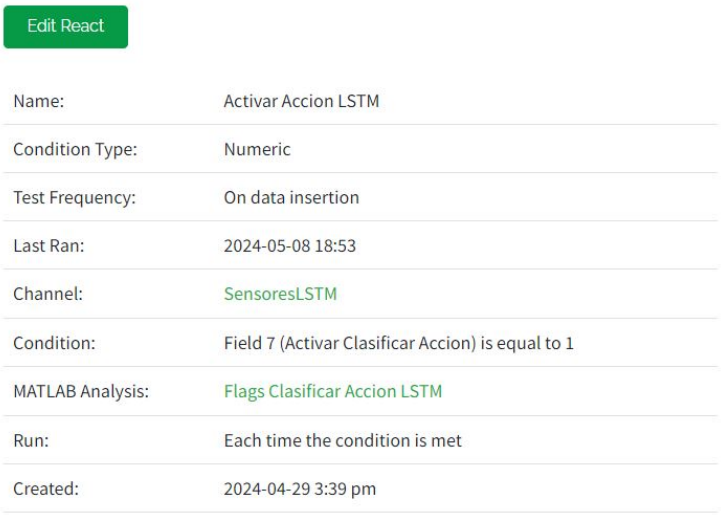

Figura 5.9: *React* Activar Accion LSTM.

Después, se recuenta el número total de movimientos clasificados y el número de acciones de*Sitting* etiquetados y se escriben en la plataforma, esto es, la nube.. Como se puede observar en la figura 5.10, el *react* de *Mandar correo* se desencadena en caso de que el número de *Sitting* escritos en la nube sea mayor que 100 (pues en 20 segundos se capturan entre 190-200 movimientos).

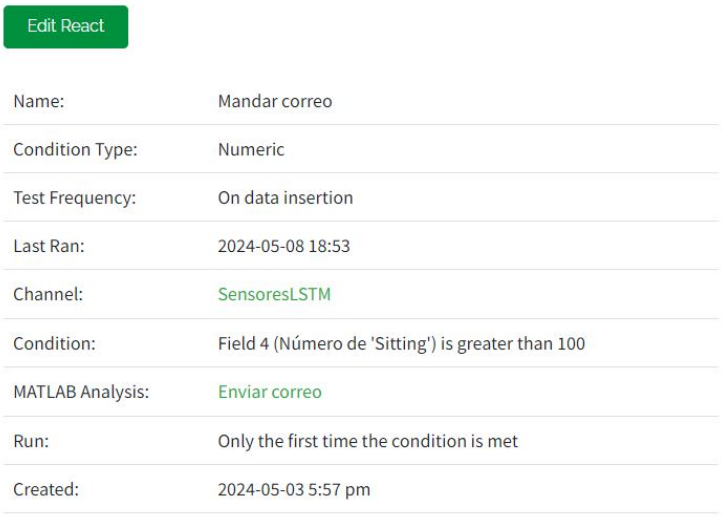

Figura 5.10: *React* correo.

Esta acción provoca el envío de un correo (figura 5.11) a la cuenta asociada al perfil de *MathWorks* en el que se indica al usuario que lleva demasiado tiempo sentado y le recomienda realizar una mayor actividad física.

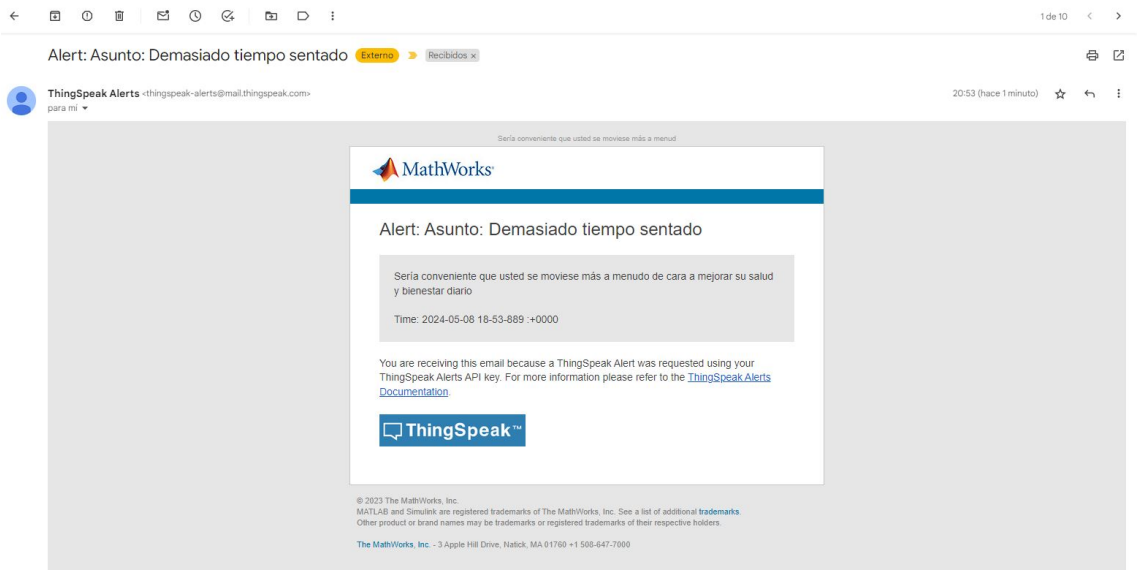

Figura 5.11: Email enviado tras la activación de *Mandar Correo*.

De esta forma se cierra el ciclo de comunicación y procesamiento de datos entre dispositivos, desde su captura al almacenamiento y manejo en la nube, para poste-

riormente recibir el mensaje de aviso procedente de la nube y en otro dispositivo o el propio móvil.

Finalmente, como colofón, cabe añadir que la programación de los *Reacts* admite la posibilidad de ejecutar código *MATLAB* instalado dentro de la propia nube para procesamiento interno de los datos, del mismo modo a como se puede realizar fuera de la nube. Para ello lo que procede es implementar el código correspondiente, almacenarlo como cualquier *script* con su nombre correspondiente y en el campo *MATLAB Analysis* del React especificar el nombre del mencionado *script*.

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo se presenta la conclusión que sintetiza la memoria de este Trabajo Fin de Grado, haciendo un repaso sobre los temas tratados en la misma. Además, se incluye una revisión y análisis de las posibles líneas de trabajo que se podrían seguir para ampliar el proyecto. Este trabajo futuro puede dividirse en dos grandes ramas. Primeramente, es menester tratar posibles mejoras de la propuesta presentada. Por otro lado, se plantean posibles aplicaciones o propuestas de trabajo, de cara a ampliar su alcance, estudiando su concurrencia con diversos sectores.

Conclusiones

En este trabajo se ha explorado la clasificación de acciones de movimiento utilizando técnicas de Aprendizaje Profundo, específicamente redes neuronales LSTM (*Long Short-term Memory*), un tipo específico de RNN (Redes Neuronales Recurrentes). Se ha demostrado que las LSTM son eficaces para reconocer y clasificar acciones de movimiento capturadas por los sensores de aceleración de dispositivos móviles, desarrollando y utilizando una aplicación que lo hace posible.

Para comenzar el trabajo, tras la introducción con sus objetivos y motivación, se han considerado, en el capítulo 2, los conceptos teóricos necesarios para comprender el funcionamiento de las redes neuronales, introduciendo conceptos específicos de las redes neuronales recurrentes, así como de las redes LSTM. Además de explicar el fundamento teórico del modelo utilizado, también se ha explicado el marco teórico de aplicación de las redes LSTM a la clasificación, especificando las capas pertenecientes a la topología de red.

Posteriormente, en el capítulo 3 se ha explicado el desarrollo de la aplicación, que consta de dos partes. Por un lado, se tiene una aplicación estática que permite entrenar y probar las redes LSTM, realizada con *App Designer* de MATLAB. Consta de dos partes. La primera, permite entrenar una red LSTM a partir de un conjunto

de datos; la segunda, permite clasificar un conjunto de datos de prueba con una red ya entrenada. Además, como aplicación dinámica, se utiliza la aplicación *MATLAB Mobile*, con la que se posibilita clasificar los movimientos en tiempo real, utilizando datos provenientes del acelerómetro del dispositivo.

Profundizando en el entrenamiento de la red LSTM, en el capítulo 4 de resultados se ha profundizado en los parámetros de entrenamiento de la misma, así como en el proceso de entrenamiento y la gráfica de entrenamiento proporcionada. De forma complementaria, se proporcionan varios ejemplos de entrenamientos consistentes en cambiar varios de los parámetros, y analizar la gráfica resultante del entrenamiento con esos parámetros modificados.

Finalmente, se ha realizado una extensión de la aplicación hacia el IoT (Internet de las Cosas), explicada en el capítulo 5. Se ha utilizado la plataforma *ThingSpeak* de MATLAB Analytics para enviar los datos del dispositivo móvil a la nube y procesarlos allí. Se han creado varios *Reacts* que permiten realizar acciones automatizadas basadas en los datos recibidos.

Gracias al compendio anterior, es posible apreciar cómo se ha ido trabajando en el concepto de la metodología presentada, el desarrollo de una aplicación para clasificar movimientos humanos en base a secuencias de datos de aceleración. Mediante los diversos capítulos de la actual memoria, en orden, se fundamenta teóricamente la propuesta, se desarrollan aplicaciones estática y dinámica para implementarla, se analizan los resultados del entrenamiento, y se amplía la propuesta mediante la extensión IoT.

Trabajo futuro

En primer lugar, se exponen una serie de mejoras, que van en la misma dirección del trabajo realizado y presentado en esta memoria. Un listado de estas posibles mejoras es:

1. *Optimización de la red*: Explorar un ajuste más fino de los parámetros de entrenamiento de la red LSTM, de cara a mejorar aún más su precisión y eficiencia.

2. *Integración multisensorial*: Incorporar datos de otros sensores, como datos de giroscopio o campo magnético, para poder entender, mediante la red, mejor las acciones de movimiento capturadas. Esta información adicional podría incorporarse sin problemas mediante el mismo método móvil con que se incorporan los datos de aceleración.

3. *Interfaz de Usuario y UX mejoradas*: Mejorar las interfaces de usuario de las aplicaciones estáticas y dinámicas, así como la experiencia de usuario (UX o *User Experience*), para facilitar su uso por parte de personas con diferentes

niveles de experiencia técnica, así como personas con algún tipo de discapacidad.

4. *Aplicación móvil*: En línea con la mejora de la UX, se podría realizar una aplicación móvil. Esta aplicación podría ser bien enfocada a Android, donde se habían explorado nociones de desarrollo con anterioridad, o multiplataforma.

Eventualmente, surgen posibles aplicaciones, adaptaciones e integraciones del trabajo dentro de diversas áreas. Algunos de los ámbitos a los que podría extrapolarse el trabajo son:

- *Medicina y rehabilitación*: Aplicar la detección y clasificación de movimientos de cara a monitorizar el progreso de pacientes en rehabilitación, evaluando no solo el tipo de movimientos sino su calidad y consistencia a lo largo del tiempo.
- *Diagnóstico de enfermedades neurodegenerativas* Predecir la presencia de enfermedades neurodegenerativas como la Enfermedad de Parkinson o el Alzheimer, mediante la identificación de patrones anormales de movimiento y clasificación de enfermedades basada en los movimientos realizados.
- *Entrenamiento personal*: análisis y evaluación de la técnica de ejecución de un amplio rango de ejercicios y movimientos deportivos, proporcionando retroalimentación y propuestas de programas de entrenamiento. Adaptación de las propuestas en función de los movimientos detectados y clasificados, e individualizarlas en consecuencia.
- *Análisis de posturas y movimientos en el trabajo*: identificar posturas y movimientos que puedan aumentar el riesgo de lesiones musculoesqueléticas en el entorno laboral, y proponer medidas correctivas. Eventual uso de *gadgets* para obtención de los mismos datos, en lugar de un dispositivo móvil, ajustándose estos dispositivos complementarios mejor a esta propuesta y propuestas similares.

Para finalizar, y como puede apreciarse en algunas de las propuestas, sería interesante evaluar la introducción de *gadgets* o dispositivos complementarios al móvil, facilitando una obtención más fidedigna y eficaz de los datos con los que realizar el entrenamiento y eventual clasificación.

Conclusions and Future Work

This chapter, a translation of chapter 6, presents a conclusion summarizing the report of this Final Project, providing a **review of the topics covered**. Additionally, it includes a review and analysis of possible **future work** that may arise. This future work can be divided into two main branches. Firstly, it is necessary to address possible **improvements** to the proposed solution. On the other hand, **possible applications** or future work proposals are discussed to expand its scope, exploring its relevance in various sectors.

Conclusions

In conclusion, this work has explored the classification of movement actions using Deep Learning techniques, specifically LSTM (Long Short-term Memory) neural networks, a specific type of RNN (Recurrent Neural Networks). It has been demonstrated that LSTMs are effective in recognizing and classifying movement actions captured by the acceleration sensors of mobile devices, developing and applying an application that makes this possible.

To begin the work, after its introduction, the chapter 2 introduced the theoretical concepts necessary to understand the functioning of neural networks, introducing specific concepts of recurrent neural networks, as well as LSTM networks. Besides explaining the theoretical foundation of the model used, the theoretical framework of applying LSTM networks to classification was also explained, specifying the layers belonging to the network topology.

Subsequently, in the chapter 3, the development of the application was explained, which consists of two parts. On one hand, there is a static application that allows training and testing the LSTM networks, developed with MATLAB's *App Designer*. It consists of two parts: the first part allows training an LSTM network from a data set; the second part allows classifying a test data set with an already trained network. Additionally, as a dynamic application, *MATLAB Mobile* is used to enable real-time classification of movements using data from the device's accelerometer.

Delving into the training of the LSTM network, the chapter 4 of results detailed the network's training parameters, the training process, and the provided training graph. Complementarily, several training examples are provided, consisting of changing various parameters and analyzing the resulting training graph with these changed parameters.

Finally, an extension of the application towards IoT (Internet of Things) was carried out, explained in the chapter 5. The *ThingSpeak* platform from MATLAB Analytics was used to send data from the mobile device to the cloud and process them there. Several *Reacts* were created to allow automated actions based on the received data.

Through the above summary, it is possible to see how the concept of the project was developed: an application to classify human movements based on sequences of acceleration data. Through the various chapters of the current report, the theoretical proposal is grounded, static and dynamic applications are developed to implement it, the training results are analyzed, and the proposal is extended through IoT.

Future Work

Firstly, a series of improvements are proposed, in line with the work done and presented in this report. A list of these possible improvements is:

1. *Network Optimization*: Explore finer tuning of the LSTM network training parameters to further improve its accuracy and efficiency.

2. *Multisensory Integration*: Incorporate data from other sensors, such as gyroscope or magnetic field data, to better understand the captured movement actions through the network. This additional information could be incorporated seamlessly using the same mobile method as the acceleration data.

3. *Improved User Interface and UX*: Enhance the user interfaces of the static and dynamic applications, as well as the user experience (UX), to facilitate use by people with different levels of technical experience, as well as people with disabilities.

4. *Mobile Application*: In line with UX improvement, a mobile application could be developed. This application could be either focused on Android, where notions of development had been previously explored, or cross-platform.

Eventually, potential applications, adaptations, and integrations of the work within various areas arise. Some of the fields to which the work could be extrapolated are:

- *Medicine and Rehabilitation*: Apply movement detection and classification to monitor the progress of patients in rehabilitation, evaluating not only the type of movements but also their quality and consistency over time.
- *Diagnosis of Neurodegenerative Diseases*: Predict the presence of neurodegenerative diseases such as Parkinson's Disease or Alzheimer's by identifying abnormal movement patterns and classifying diseases based on the movements performed.
- *Personal Training*: Analyze and evaluate the technique of performing a wide range of exercises and sports movements, providing feedback and training program proposals. Adapt the proposals based on the detected and classified movements, and individualize them accordingly.
- *Posture and Movement Analysis at Work*: Identify postures and movements that may increase the risk of musculoskeletal injuries in the workplace and propose corrective measures. Eventual use of *gadgets* for data collection, instead of a mobile device, adjusting these complementary devices better to this proposal and similar ones.

Finally, as can be seen in some of the proposals, it would be interesting to evaluate the introduction of *gadgets* or devices complementary to the mobile device, facilitating a more accurate and efficient collection of the data used for training and eventual classification.

Contribuciones Personales

Rodrigo Gómez Serrano

Mi trabajo comenzó con una investigación básica acerca de las redes neuronales LSTM (Long Short Term Memory) que se iban a emplear más adelante, con el objetivo claro de obtener una visión general e intuitiva de estas redes. Esto me sirvió para realizar varios esquemas que posteriormente fueron de gran utilidad. Además, Miguel y yo hicimos un ademán de comenzar a escribir la introducción del trabajo. Yo en particular escribí un boceto de la motivación y del estado de la cuestión, que posteriormente serían brevemente retocados para la versión final.

Una vez entendidos los conceptos principales, procedí a estudiar más a fondo la parte teórica del trabajo. En concreto me centré en dos libros aportados por el tutor que incluían gran cantidad de contenidos relacionados con el tema de las redes LSTM. Esta parte fue realizada junto con Miguel y tras varias lecturas, reuniones y borradores, nos dispusimos a comenzar la memoria. Fuimos escribiendo los conceptos teóricos en el capítulo 2 de la manera más comprensible posible, pero siempre incluyendo los contenidos fundamentales. En cuanto a la redacción, yo me centre más en la parte de los parámetros de las redes en general.

Después de finalizar este capítulo, comencé a estudiar cómo funcionan las redes LSTM en MATLAB gracias a un ejemplo que tiene MathWorks en su propia página web y elaboré un borrador con cada una de las capas de red, funciones y parámetros que nos podían servir de ayuda. A continuación empecé a adaptar y escribir un fichero de código con el objetivo de entrenar una red y emplearla para clasificar unos datos de movimiento. Mientras hacía esto, también lo explicaba en la memoria (en lo que correspondía en un principio el capítulo 3.2) incluyendo las características más importantes de las funciones y capas de la red.

En ese momento planifiqué una reunión tutorial para ver cuál era la mejor manera de enlazar el código y la teoría con la captura de acciones de movimiento mediante los sensores del dispositivo móvil, pues me estaba costando encontrar la conexión. Decidimos dividir el capítulo 3 en el apartado 3.2, una aplicación estática

(diseñada mediante App Designer) y el apartado 3.3, que trataría la parte dinámica. Por tanto, aprendí a utilizar App Designer desde cero y comencé a diseñar una interfaz (imágenes en 3.2), que integré con el código desarrollado previamente. De esta manera, teníamos una aplicación en el ordenador (estática) capaz de entrenar una red y clasificar acciones de movimiento siempre y cuando los datos estuviesen previamente cargados en el ordenador. Como hubo un pequeño cambio de dirección en el trabajp, tuve que adaptar la memoria a los cambios y en concreto a la nueva aplicación.

El siguiente paso fue estudiar un ejemplo de conexión entre un dispositivo móvil y MATLAB. Para ello se usa MATLAB Mobile, cuyo funcionamiento tuvimos que aprender a pesar de que se trata de una tarea relativamente sencilla. Tras varias pruebas, adapté el código para desarrollar un programa ejecutable desde el dispositivo móvil (dinámico), que captura una serie de datos de aceleración a través del dispositivo apropiado y posteriormente, las clasifica. Además muestra todos los resultados sobre ciertas indicaciones por la pantalla del dispositivo. Una vez era completamente funcional, redacté la explicación correspondiente en la memoria explicando su funcionamiento paso a paso.

A continuación, me dispuse a explicar en la memoria cómo se puede observar y controlar el entrenamiento de una red desde la aplicación estática en general. Después, Miguel y yo nos dispusimos a ejecutar ejemplos concretos en la aplicación con distintos parámetros para mostrar que funcionaba correctamente. Realizamos muchas ejecuciones pero decidimos que era suficiente mostrar tres, que son las realmente explicadas en la memoria.

En esta situación, decidimos dar un repaso general a la memoria y avanzar el máximo posible con la introducción antes de realizar nuevas tareas. Miguel y yo dimos una buena forma a la introducción y realizamos correcciones específicas, gran parte de ellas mencionadas por el tutor. El objetivo era tener una versión presentable antes de ampliar más.

Con el objetivo que acabamos de mencionar cumplido, y tras una nueva reunión con el tutor de cara a la ampliación, comencé a investigar a cerca del Internet de las cosas y de ThingSpeak. El concepto de IoT era algo totalmente nuevo y tampoco conocía la plataforma ThingSpeak, por lo que aprendí a través de tutoriales del propio MathWorks y de un ejemplo propio del tutor, entre otras fuentes. Del mismo modo que con los ficheros de código anteriores, desarrollé un archivo que trataba los conceptos más generales de conexión dispositivo-nube. Tuve que gestionar los canales de ThingSpeak, además de escribir los scripts de React y Analysis en dicha plataforma. Con todo esto, probé el código y cuando conseguí una versión funcional, me dispuse a escribirlo en la memoria (capítulo 6).

Por útlimo, decidimos dar por finalizado el contenido principal del trabajo y nos dispusimos a pulir detalles. En mi caso, corregí errores y erratas a lo largo de todo el texto y aporté la cohesión que faltaba entre distintas secciones y apartados. Además, finalicé el resumen con su traducción al *abstract* y creé el repositorio del trabajo, en el que se incluye el código y el manual de usuario, que también lo escribí yo.

Miguel Manzano Rodríguez

Primeramente, en cuanto a la elección del tema y grupo para realizar el trabajo, hablé con Rodrigo debido a nuestra estrecha colaboración en otros trabajos con anterioridad. En cuanto al tema, tras barajar diversas opciones, optamos por escoger el tema del actual trabajo, con Gonzalo Pajares Martinsanz como tutor. Yo aporté a la elección del tema en cuanto que me resultaba un tema interesante que tratar, y captaba mi interés. En cuanto al primer contacto con el tutor y las primeras reuniones, estuve presente y contribuí a desarrollar la idea y darle forma.

Durante los primeros meses de la elaboración del trabajo, desde septiembre hasta enero, estuve investigando junto a Rodrigo acerca de diversas opciones que implementar. Una idea que pasó por la cabeza de ambos fue la de realizar el proyecto en *Python*, y no *MATLAB*. Esta idea, si bien es más típica al ser el lenguaje de programación más utilizado, conllevaba algunas ventajas. Dentro de esas ventajas estaba la posible realización de una aplicación de *Android* más versátil y personalizable. Precisamente en este ámbito estuve explorando personalmente, siempre consultando con Rodrigo, mediante la instalación de la herramienta *Android Studio*, utilizada para desarrollar aplicaciones en Android. Mediante esta herramienta, fue posible realizar algunas pruebas de integración con la idea de aprendizaje automático ideada.

Sin embargo, se puso de manifiesto la gran complejidad de realizar un desarrollo completo de una aplicación de *Android*, por varios motivos. Primeramente, fue evidente el esfuerzo que se habría de emplear en desarrollar la *app* mediante *Android Studio*, siendo realmente un complemento al desarrollo principal. Además, resultaba más conveniente para mí, al disponer de un dispositivo *Android*, pero no tanto así para Rodrigo, que dispone de un dispositivo móvil con sistema operativo *iOS*. Evaluando la situación, el desarrollo multiplataforma pareció fuera de los límites del trabajo. Por tanto, tras esta exploración de los miembros del grupo, donde Miguel realizó pruebas con *Android Studio*, se decidió desarrollar la propuesta que se muestra en este Trabajo.

Primeramente, se decidió desarrollar el fundamento teórico (capítulo 2. En cuanto a la investigación al respecto, sirvió de gran utilidad a Miguel un primer resumen de los conceptos teóricos que realizó Rodrigo, en papel. Basándome en estos resúmenes, y en información de varios libros, algunos referenciados a partir de otros libros, y otros accedidos mediante la Biblioteca de la UCM, escribimos esta parte más conceptual del trabajo. Dentro de esta parte más conceptual, me involucré bastante en su redacción, siempre con la ayuda de Rodrigo. Para realizar dicha redacción, hicimos primero varios borradores, hasta que tuvimos la versión final, incorporando imágenes y acabando de dar formato a los apartados. Personalmente, creo que fui de especial utilidad a la hora de recopilar toda la información, y de escribirla de tal manera que resultara atractiva al lector y estuviera bien organizada. Sin embargo, Rodrigo tenía una gran habilidad a la hora de organizar los apartados del capítulo, y pensar en cómo organizar la información en secciones.

Seguidamente, aunque paralelamente al desarrollo del capítulo anterior, estuvimos comentando con Gonzalo el desarrollo de la aplicación (capítulo 3), centrándonos en el desarrollo y haciendo uso de algunos ejemplos ya existentes. Ayudé a perfilar este desarrollo, y a explicar junto a Rodrigo las distintas partes del código adaptado. En cuanto al capítulo de resultados (capítulo 4), realicé algunas ejecuciones locales en mi ordenador, variando los parámetros con respecto al entrenamiento de prueba con los parámetros predefinidos. Pese a que estos entrenamientos llevaban entre 30 y 60 minutos, no resultó un problema pues el entrenamiento no se detenía incluso si el ordenador dejaba de estar activo a mitad del proceso. Además, escribí una explicación de estos diferentes ejemplos en la memoria, analizando las diferencias entre las gráficas de entrenamiento de los ejemplos.

En cuanto a la extensión del trabajo planteado originalmente (capítulo 5), también se me pasó por la cabeza hacer una extensión realcionada con los relojes inteligentes o *smartwatches*. Sin embargo, tras estar investigando personalmente, y analizar los datos de mi reloj personal, vi que no era una opción viable para nuestro trabajo, y no la presenté ni a Rodrigo ni a Gonzalo.

Como conclusión a la actual memoria, me encargué de dar forma al capítulo de conclusiones del trabajo, incluyendo el trabajo futuro (capítulo 6) a partir del mismo. Para este capítulo, simplemente me pareció buena idea ir repasando lo expuesto en cada capítulo del trabajo, y hacer un resumen en base a esa información. Para el trabajo futuro, tuve en cuenta algunas aplicaciones, que además sabía que nos son de interés a ambos componentes del grupo, debido a lo que hemos hablado durante la realización del trabajo acerca del mismo.

Como contribución intrínseca al trabajo, cabe mencionar el apartado bibliográfico, que realicé al completo, basándome en las fuentes bibliográficas que habíamos ido anotando tanto Rodrigo como yo, y aportando asimismo fuentes nuevas, que abarcaran toda la información externa presente en el Trabajo Fin de Grado. De este modo, para todas las fuentes bibliográficas, me aseguré de obtener todos los atributos de cada fuente, juntándolos en formato *BibTeX*, de manera que su presentación en la memoria fuera la correcta. Además, me aseguré de referenciar de manera correcta estas fuentes en el texto de la memoria, con un acceso interactiva a estas.

Además, me encargué de limpiar el código del proyecto, así como de recopilarlo en su totalidad y agregarlo al apéndice B, con un formato apropiado para MATLAB. Tras pensarlo y consultarlo con Rodrigo, decidí agregar todo el código, incluyendo aquel de AppDesigner (en formato .mlapp), de manera que quedara expuesto en su totalidad en la memoria, pese a su longitud.

Por último, también contribuí en otros aspectos del trabajo. En las reuniones presenciales con el tutor, traté de entender su punto de vista (por ejemplo, tomando siempre notas, que más tarde nos sirvieron de utilidad para recordar la posición de Gonzalo respecto a un tema concreto). Además, contribuí a las revisiones que planteó el tutor. En este sentido, cuando había revisiones o correcciones que hacer, tanto Rodrigo como yo realizamos una revisión exhaustiva, repartiéndonos el trabajo y cubriendo todas las correcciones a realizar.

Bibliografía

El que lee mucho y anda mucho ve mucho y sabe mucho.

Miguel de Cervantes Saavedra

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006.

Brownlee, J. Difference Between a Batch and an Epoch in a Neural Network. 2018. Disponible en `https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/` (último acceso, Mayo, 2024).

Fedorin, I. y Slyusarenko, K. Consumer Smartwatches As a Portable PSG: LSTM Based Neural Networks for a Sleep-Related Physiological Parameters Estimation. *Annu Int Conf IEEE Eng Med Biol Soc*, vol. 2021, páginas 849–452, 2021.

Goodfellow, I. J., Bengio, Y. y Courville, A. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

Hochreiter, S. y Schmidhuber, J. Long Short-term Memory. *Neural Computation*, vol. 9(8), páginas 1735–1780, 1997. Disponible en `https://www.bioinf.jku.at/publications/older/2604.pdf` (último acceso, Mayo, 2024).

Khan, Y., Imaduddin, S., Prabhat, R. y Wajid, M. Classification of Human Motion Activities using Mobile Phone Sensors and Deep Learning Model. páginas 1381–1386. 2022.

Kim, P. *MATLAB Deep Learning*. 2017. ISBN 978-1-4842-2844-9.

Kingma, D. P. y Ba, J. Adam: A method for stochastic optimization. 2017.

Mekruksavanich, S. y Jitpattanakul, A. LSTM Networks Using Smartphone Data for Sensor-Based Human Activity Recognition in Smart Homes. *Sensors (Basel)*, vol. 21(5), 2021. Disponible en `https://www.mdpi.com/1424-8220/21/5/1636` (último acceso, Mayo, 2024).

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, Massachusetts, USA, 2012.

Nabriya, P. Implementing LSTM for Human Activity Recognition using Smartphone Accelerometer data. 2021. Disponible en `https://tinyurl.com/mrxbmvw7` (último acceso, Mayo, 2024).

Pajares, G., Herrera, P. y Besada, E. *Aprendizaje Profundo*. RC-Libros, 2021. ISBN 9788412106985.

Ruder, S. An overview of gradient descent optimization algorithms. 2017.

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. Learning representations by back-propagating errors. *Nature*, vol. 323(6088), páginas 533–536, 1986. ISSN 1476-4687.

Sutskever, I., Martens, J., Dahl, G. y Hinton, G. On the importance of initialization and momentum in deep learning. En *Proceedings of the 30th International Conference on Machine Learning* (editado por S. Dasgupta y D. McAllester), vol. 28 de *Proceedings of Machine Learning Research*, páginas 1139–1147. PMLR, Atlanta, Georgia, USA, 2013.

Valkov, V. Time Series Classification for Human Activity Recognition with LSTMs using Tensorflow 2 and Keras. 2020. Disponible en `https://tinyurl.com/mvh7vt8u` (último acceso, Mayo, 2024).

Van Houdt, G., Mosquera, C. y Nápoles, G. A review on the long short-term memory model. *Artificial Intelligence Review*, vol. 53(8), páginas 5929–5955, 2020. ISSN 1573-7462. Disponible en `https://link.springer.com/article/10.1007/s10462-020-09838-1` (último acceso, Mayo, 2024).

Yamabe, M., Horie, K., Shiokawa, H., Funato, H., Yanagisawa, M. y Kitagawa, H. MC-Sleepnet: Large-scale Sleep Stage Scoring in Mice by Deep Neural Networks. *Scientific Reports*, vol. 9(15793), página 15793, 2019. Disponible en `https://doi.org/10.1038/s41598-019-51269-8` (último acceso, Mayo, 2024).

Apéndice A

Manual de usuario

En este primer apéndice, se incluye un manual de usuario, que tiene como objetivo proporcionar a los lectores una guía para ejecutar las distintas aplicaciones del trabajo correctamente.

Para consultar tanto los ficheros de código como este manual, se ha habilitado un repositorio de GitHub al cual se accede mediante el siguiente *link*:

github.com/ROGOSE/TFGClasificacionDeMovimientosLSTM

A.1. Material necesario

En primer lugar, debe contar con un dispositivo inteligente *Smartphone* y un PC (Computador Personal).

Además, es necesario instalar en el ordenador los programas MATLAB (disponible en ssii.ucm.es), válido tanto en PC como *On-Line* y MATLAB Drive Connector (disponible en es.mathworks.com). Esto último proporciona acceso a MATLAB Drive, que lleva a cabo la comunicación entre la aplicación y el programa principal de Matlab.

También debe instalar en el dispositivo móvil la aplicación MATLAB Mobile. Para la nube, debe tener acceso a ThingSpeak (thingspeak.com).

A.2. Aplicación estática

Para ejecutar esta aplicación hay que abrir MATLAB en el ordenador y abrir App Designer. Para ello la opción más sencilla es escribir *appdesigner* en la línea de comandos (aunque también se puede acceder desde Apps, pestaña que se ubica en

la parte superior.

Una vez el usuario se encuentra en App Designer, hay que abrir los archivos *pantallaInicio.mlapp*, *pantallaTest.mlapp*, *pantallaTrain.mlapp*, *pantallaResultados.mlapp* y *pantallaParametros.mlapp*. Los archivos se abren en la parte superior izquierda de la interfaz, pinchando en el icono *Open*.

A continuación, hay que situarse en el archivo *pantallaInicio.mlapp* y para ello, se debe pinchar en la pestaña correspondiente. Una vez ahí, se pulsa el botón *Run* en la parte superior izquierda de la interfaz, lo que desencadena el inicio de la aplicación. En la sección 3.2 se desarrolla paso a paso y se explica con detalle la funcionalidad de la aplicación.

Es importante mencionar que entre que se abre una pestaña y la siguiente pueden discurrir unos segundos.

A.3. Aplicación dinámica

Para ejecutar este programa son necesarios al menos los siguientes *toolboxes*: *Image Processing*, *Computer Vision* y *Deep Learning*. Si se requiere de algún *toolbox* más, ya lo pide el propio MATLAB y lo instala, siempre con la cuenta institucional. Se requiere la cuenta institucional de la UCM.

Una vez tenemos la app MATLAB Mobile instalada, la abrimos. En ella, pulsamos en el icono superior izquierdo que son tres barras horizontales para desplegar un menú con varias opciones. Elegimos Sensores y una vez ahí seleccionamos:

- Transmitir a → MATLAB (para transmitir en tiempo real a MATLAB).

- Registro de sensores → Configurar → Acceso a sensores (activar al menos la aceleración).

En la carpeta MATLAB Drive tiene que haber un fichero con el modelo de red entrenada, como por ejemplo, *RedLSTM.mat*, que está pre-entrenada para las siguientes acciones *Dancing, Running, Sitting, Standing, Walking*. El otro ejemplo de red es *red_ ejemplo_ entrenada.mat*. Ambos archivos se encuentran en el repositorio del trabajo.

En esta misma carpeta hay que copiar el fichero *TrabajoFinalLSTM.m*, que contiene el código a ejecutar y en el cuál hay que modificar las dos siguientes líneas según la red que se quiera utilizar. En concreto hay que cambiar la cadena *'RedLSTM'* y el archivo que se carga meidante *load*.

```
if not(exist('RedLSTM','var'))
    load RedLSTM;
```

Una vez ejecutado el código, se muestra por la pantalla del dispositivo el resultado, es decir, aparece la clasificación correspondiente a cada una de las acciones.

A.4. Aplicación en la nube

Para ejecutar este programa son necesarios al menos los siguientes *toolboxes*: *Image Processing*, *Computer Vision* y *Deep Learning*. Si se requiere de algún *toolbox* más, ya lo pide el propio MATLAB y lo instala, siempre con la cuenta institucional. Se requiere la cuenta institucional de la UCM. Además, en este caso es necesario contar con *MATLAB Support Package for Android Sensors* o *MATLAB Support Package for Apple iOS Sensors*.

Una vez tenemos la app MATLAB Mobile instalada, la abrimos. En ella, pulsamos en el icono superior izquierdo que son tres barras horizontales para desplegar un menú con varias opciones. Elegimos Sensores y una vez ahí seleccionamos:

- Transmitir a → MATLAB (para transmitir en tiempo real a MATLAB).

- Registro de sensores → Configurar → Acceso a sensores (activar al menos la aceleración).

En la carpeta MATLAB Drive tiene que haber un fichero con el modelo de red entrenada, como por ejemplo, *RedLSTM.mat*, que está pre-entrenada para las siguientes acciones *Dancing, Running, Sitting, Standing, Walking*. El otro ejemplo de red es *red_ ejemplo_ entrenada.mat*. Ambos archivos se encuentran en el repositorio del trabajo.

En esta misma carpeta hay que copiar el fichero *extensionIoT.m*, que contiene el código a ejecutar y en el cuál hay que modificar las dos siguientes líneas según la red que se quiera utilizar. En concreto hay que cambiar la cadena *'RedLSTM'* y el archivo que se carga mediante *load*.

```
if not(exist('RedLSTM','var'))
    load RedLSTM;
```

Según se ejecuta el código, se muestra por la pantalla del dispositivo varias indicaciones de lo que está ocurriendo y, por último, el resultado, es decir, aparece la clasificación correspondiente a cada una de las acciones.

Apéndice B

Script/código utilizado

B.1. Código de MATLAB (entrenamiento y visualización)

B.1.1. Muestra de datos iniciales (MostrarDatosIniciales.m)

```
%%  CARGAMOS LOS DATOS

load HumanActivityTrain
XTrain % aceleraciones
YTrain % categor as de cada aceleraci n

%%  Visualizamos una secuencia de entrenamiento en una
   gr fica. Solo los datos del 1er lote.

%%  Representamos la aceleracion de la 1a coord. frente
   al tiempo seg n la actividad correspondiente.

X = XTrain{6}(1,:);
classes = categories(YTrain{6});
colores = {'blue', 'green', 'red', 'cyan', 'magenta'};

figure(1)
for j = 1:numel(classes)
    label = classes(j);
    idx = find(YTrain{6} == label);
    hold on
    plot(idx,X(idx), 'Color', colores{j})
end
```

```
hold off

xlabel("Paso de Tiempo")
ylabel("Aceleraci n")
title("Secuencia de Entrenamiento 1, Atributo 1(x)")
legend(classes,'Location','northwest')

%% Representamos la aceleracion de la 2a coord. frente al
    tiempo seg n la actividad correspondiente.

Y = XTrain{1}(2,:);

figure(2)
for j = 1:numel(classes)
    label = classes(j);
    idy = find(YTrain{1} == label); %
    hold on
    plot(idy,Y(idy), colores{j})
end
hold off

xlabel("Paso de Tiempo")
ylabel("Aceleraci n")
title("Secuencia de Entrenamiento 1, Atributo 2(y)")
legend(classes,'Location','southwest')

%% Representamos la aceleracion de la 3a coord. frente al
    tiempo seg n la actividad correspondiente.

Z = XTrain{1}(3,:);

figure(3)
for j = 1:numel(classes)
    label = classes(j);
    idz = find(YTrain{1} == label);
    hold on
    plot(idz,Z(idz), colores{j})
end
hold off

xlabel("Paso de Tiempo")
ylabel("Aceleraci n")
title("Secuencia de Entrenamiento 1, Atributo 3(z)")
legend(classes,'Location','northwest')
```

B.1.2. Entrenamiento y testeo (EntrenarParamLSTM.m)

```
%% Cargar datos secuenciales
load HumanActivityTrain
XTrain

%% Definir la arquitectura de red LSTM
numFeatures = 3;
numHiddenUnits = 200;
numClasses = 5;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm', ...
    'MaxEpochs',30, ...
    'GradientThreshold',2, ...
    'Verbose',0, ...
    'MiniBatchSize',64, ...
    'Plots','training-progress');


%% Descomentar si se desea llevar a cabo el entrenamiento
    de la red.

%net = trainNetwork(XTrain,YTrain,layers,options);


%% Descomentar si se desea utilizar una red ya entrenada.

load RedLSTM;
RedLSTM = net;


%%  Test LSTM Network. Para testear una red ya entrenada.

load HumanActivityTest
figure
plot(XTest{1}')
xlabel("Time Step")
```

```
legend("Feature " + (1:numFeatures))
title("Test Data")

% Clasificar los datos de test usando classify.
YPred = classify(net,XTest{1});

% Calcular la precisi n de las predicciones.

acc = sum(YPred == YTest{1})./numel(YTest{1})

% Comparar las predicciones con los datos de test
   utilizando una gr fica.

figure
plot(YPred,'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])
```

B.1.3. Clasificación con red entrenada (TrabajoFinalLSTM.m)

```
%% Preparacion del dispositivo y datos

%% Borrado de objetos previos
if exist('m','var')
    clear m;
end

%% Crear objeto
m = mobiledev;

%% Cargar la RedLSTM si no existe la variable
if not(exist('RedLSTM','var'))
    load RedLSTM;
    RedLSTM = net;
end

%% Activar la habilitaci n para la captura de datos
   sensoriales durante el tiempo que dura el bucle
```

```
m.Logging = 1;

t = 3;
for i = 0:1:t
    pause(1); % MATLAB Mobile solo permite pausas de
       m ximo 2 segundos
end

%% Desactivar la captura
m.Logging = 0;

%% Recuperar los datos almacenados
[aceleracion, taceleracion] = accellog(m);   % Logged
   acceleration data

%% Clasificaci n de la acci n mediante el sensor
   aceleraci n
[M,N] = size(aceleracion);

if M == 0
  disp('No se han capturado datos: repetir');
else
  X = aceleracion';
  Actividades = classify(RedLSTM,X)
end

discardlogs(m);
```

B.1.4. Prueba de ThingSpeak (PruebaThingSpeak.m)

```
readChannelID = 2521644;
writeChannelID = 2521644;
APIKey = '1SG5UE41TW4LCTEP';

% resultaData = thingSpeakRead(readChannelID, 'ReadKey',
   APIKey, 'Fields',1)

% avgResults = mean(resultData);

valoresAEscribir=[[0,0,0]]; % [field1,field2,field3] SOLO
    DEJA 1 VALOR por FIELD, no vectores
```

```
writeResponse = thingSpeakWrite(writeChannelID,
   valoresAEscribir, 'WriteKey', APIKey);
```

B.2. Código de AppDesigner

B.2.1. Pantalla de inicio (pantallaInicio.mlapp)

```
classdef pantallaInicio < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure             matlab.ui.Figure
        TestNetworkButton    matlab.ui.control.Button
        Image                matlab.ui.control.Image
        BienvenidoEstaaplicacintepermitLabel  matlab.ui.
           control.Label
        TrainNetworkButton   matlab.ui.control.Button
    end


    properties (Access = private)
        pantallaTrain % Description
        pantallaTest % Description
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: TrainNetworkButton
        function TrainNetworkButtonPushed(app, event)
            app.UIFigure.Visible = 'off';
            app.pantallaTrain = pantallaTrain();
            app.pantallaTrain.UIFigure.Visible = 'on';
        end

        % Button pushed function: TestNetworkButton
        function TestNetworkButtonPushed(app, event)
            app.UIFigure.Visible = 'off';
            app.pantallaTest = pantallaTest();
            app.pantallaTest.UIFigure.Visible = 'on';
        end
    end
```

```
% Component initialization
methods (Access = private)

    % Create UIFigure and components
    function createComponents(app)

        % Create UIFigure and hide until all
           components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.Color = [0.949 0.8706 0.6784];
        app.UIFigure.Position = [100 100 432 476];
        app.UIFigure.Name = 'MATLAB App';

        % Create TrainNetworkButton
        app.TrainNetworkButton = uibutton(app.
           UIFigure, 'push');
        app.TrainNetworkButton.ButtonPushedFcn =
           createCallbackFcn(app, @
           TrainNetworkButtonPushed, true);
        app.TrainNetworkButton.FontWeight = 'bold';
        app.TrainNetworkButton.Position = [250 100
           108 29];
        app.TrainNetworkButton.Text = 'Train Network'
           ;

        % Create BienvenidoEstaaplicacintepermitLabel
        app.BienvenidoEstaaplicacintepermitLabel =
           uilabel(app.UIFigure);
        app.BienvenidoEstaaplicacintepermitLabel.
           FontSize = 14;
        app.BienvenidoEstaaplicacintepermitLabel.
           Position = [28 147 378 65];
        app.BienvenidoEstaaplicacintepermitLabel.Text
            = {'Bienvenido! Esta aplicaci n permite
           entrenar un conjunto de '; '
           movimientos as  como clasificarlos.
           Disfruta!'};

        % Create Image
        app.Image = uiimage(app.UIFigure);
        app.Image.Position = [147 221 140 184];
        app.Image.ImageSource = 'D:\Users\migma\
           MATLAB Drive\MATLAB\Im genes\
           SevillanasConCopyright.JPG';
```

```
            % Create TestNetworkButton
            app.TestNetworkButton = uibutton(app.UIFigure
                , 'push');
            app.TestNetworkButton.ButtonPushedFcn =
                createCallbackFcn(app, @
                TestNetworkButtonPushed, true);
            app.TestNetworkButton.FontWeight = 'bold';
            app.TestNetworkButton.Position = [76 100 108
                29];
            app.TestNetworkButton.Text = 'Test Network';

            % Show the figure after all components are
                created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = pantallaInicio

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

B.2.2. Pantalla de Test (pantallaTest.mlapp)

```
classdef pantallaTest < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        ClasificarOKLabel               matlab.ui.control.
           Label
        CLASIFICARDATOSButton           matlab.ui.control.
           Button
        SegundoPasoLabel                matlab.ui.control.
           Label
        PrimerPasoLabel                 matlab.ui.control.
           Label
        PanelCargarDatosClasif          matlab.ui.container.
           Panel
        MostrarlosdatosButton           matlab.ui.control.
           Button
        DatosPredetListBox              matlab.ui.control.
           ListBox
        RedespredeterminadasLabel_2     matlab.ui.control.
           Label
        LabelDatosRuta                  matlab.ui.control.
           Label
        ArchivoDatosLabel               matlab.ui.control.
           Label
        ExplorarDatosButton             matlab.ui.control.
           Button
        PanelCargarRed                  matlab.ui.container.
           Panel
        RedesPredetListBox              matlab.ui.control.
           ListBox
        RedespredeterminadasLabel       matlab.ui.control.
           Label
        LabelRedesRuta                  matlab.ui.control.
           Label
        ArchivoRedesLabel               matlab.ui.control.
           Label
        ExplorarRedesButton             matlab.ui.control.
           Button
    end


    properties (Access = private)
        pantallaResultados % Description
        Red
```

```
        DatosTest % Description
        YPred
        Acc
        YT
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: ExplorarRedesButton
        function ExplorarRedesButtonPushed(app, event)
            %[filename, filepath] = uigetfile(filter,
               dialogTitle);
            %filter es para el tipo de archivo como '*.
               txt'
            [filename, ~] = uigetfile('*.*', 'Selecciona
               un archivo');

            % Comprueba si el usuario ha seleccionado un
               archivo
            if isequal(filename, 0) %compara 2 vars. A y
               B
                app.LabelRedesRuta.Text = 'No se ha
                   seleccionado ning n archivo.';
                app.RedesPredetListBox.Value = '-';
            else
                %app.NosehaseleccionadoningnarchivoLabel.
                   Text = ['Archivo seleccionado: ',
                   fullfile(filepath, filename)];
                app.LabelRedesRuta.Text = filename;
                app.RedesPredetListBox.Value = '-';
                %%hay que cargar la red
                %app.Red = load(filename);
                app.Red = load('red_ejemplo_entrenada.mat
                   ');
                app.Red
                %load filename;
            end
        end

        % Value changed function: RedesPredetListBox
        function RedesPredetListBoxValueChanged(app,
           event)
            value = app.RedesPredetListBox.Value;
            if isequal(value, '-') %si se elige '-'
```

```
            app.LabelRedesRuta.Text = 'No se ha
               seleccionado ning n archivo.';
        else
            app.LabelRedesRuta.Text = value + "
               cargada.";
            %%hay que cargar la red
            app.Red = load(value); %%da error si
               cargamos una red que no existe
        end
    end

    % Button pushed function: ExplorarDatosButton
    function ExplorarDatosButtonPushed(app, event)
         %[filename, filepath] = uigetfile(filter,
            dialogTitle);
        %filter es para el tipo de archivo como '*.
           txt'
        [filename, ~] = uigetfile('*.*', 'Selecciona
           un archivo');

        % Comprueba si el usuario ha seleccionado un
           archivo
        if isequal(filename, 0) %compara 2 vars. A y
           B
            app.LabelDatosRuta.Text = 'No se ha
               seleccionado ning n archivo.';
            app.DatosPredetListBox.Value = '-';
            app.MostrarlosdatosButton.Enable = "off";
        else
            %app.NosehaseleccionadoningnarchivoLabel.
               Text = ['Archivo seleccionado: ',
               fullfile(filepath, filename)];
            app.LabelDatosRuta.Text = filename;
            app.DatosPredetListBox.Value = '-';
            app.MostrarlosdatosButton.Enable = "on";
            %%cargar los datos,

            %%SE PODRIA PONER UN CATCH PARA EXCEPCION
                SI NO SE CARGA
            app.DatosTest = load(filename, 'XTest', '
               YTest');
        end
    end

    % Value changed function: DatosPredetListBox
    function DatosPredetListBoxValueChanged(app,
```

```
    event)
      value = app.DatosPredetListBox.Value;
      if isequal(value, '-') %si se elige '-'
          app.LabelDatosRuta.Text = 'No se ha
             seleccionado ning n archivo.';
          app.MostrarlosdatosButton.Enable = "off";
      else
          app.LabelDatosRuta.Text = value + "
             cargado.";
          app.MostrarlosdatosButton.Enable = "on";
          %cargar los datos
          app.DatosTest = load(value, 'XTest', '
             YTest');
      end
end

% Button pushed function: CLASIFICARDATOSButton
function CLASIFICARDATOSButtonPushed(app, event)

      if isequal(app.RedesPredetListBox.Value, '-')
          app.ClasificarOKLabel.Text = 'Selecciona
             correctamente la red.';
          %%nada
      else
          if isequal(app.DatosPredetListBox.Value,
             '-')
              app.ClasificarOKLabel.Text = '
                 Selecciona correctamente los datos
                 .';
              %%nada
          else
              app.ClasificarOKLabel.Text = '
                 CARGANDO NUEVA PANTALLA...';
              %%NUEVA PANTALLA DE CLASIF DATOS,
                 pero primero
              %%clasificamos y calculamos accuracy
              app.YPred = classify(app.Red.net, app
                 .DatosTest.XTest{1});
              %acc = sum(YPred == YTest{1})./numel(
                 YTest{1})
              suma = sum(app.YPred == app.DatosTest
                 .YTest{1})./numel(app.DatosTest.
                 YTest{1});
              app.Acc = suma; %%de aqui sale bien
                 el valor
```

```
                    app.UIFigure.Visible = 'off';
                    app.pantallaResultados =
                       pantallaResultados();
                    app.pantallaResultados.YT = app.
                       DatosTest.YTest{1};
                    app.pantallaResultados.YPred = app.
                       YPred;
                    app.pantallaResultados.Acc = app.Acc;
                    app.pantallaResultados.UIFigure.
                       Visible = 'on';
                end
            end

        end

        % Button pushed function: MostrarlosdatosButton
        function MostrarlosdatosButtonPushed(app, event)
            numFeatures = 3;  %%suponemos que esto es 3
               siempre
            figure
            plot(app.DatosTest.XTest{1}')
            xlabel("Paso de tiempo")
            legend("Coordenada " + (1:numFeatures))
            title("Datos a clasificar")
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all
               components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [0.949 0.8706 0.6784];
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create PanelCargarRed
            app.PanelCargarRed = uipanel(app.UIFigure);
            app.PanelCargarRed.Position = [36 288 569
               137];

            % Create ExplorarRedesButton
```

```
app.ExplorarRedesButton = uibutton(app.
    PanelCargarRed, 'push');
app.ExplorarRedesButton.ButtonPushedFcn =
    createCallbackFcn(app, @
    ExplorarRedesButtonPushed, true);
app.ExplorarRedesButton.FontWeight = 'bold';
app.ExplorarRedesButton.Position = [48 96 211
     23];
app.ExplorarRedesButton.Text = 'Cargar red
    previamente entrenada';

% Create ArchivoRedesLabel
app.ArchivoRedesLabel = uilabel(app.
    PanelCargarRed);
app.ArchivoRedesLabel.FontWeight = 'bold';
app.ArchivoRedesLabel.Position = [337 77 54
    22];
app.ArchivoRedesLabel.Text = 'Archivo:';

% Create LabelRedesRuta
app.LabelRedesRuta = uilabel(app.
    PanelCargarRed);
app.LabelRedesRuta.Position = [337 50 216
    22];
app.LabelRedesRuta.Text = 'No se ha
    seleccionado ning n archivo.';

% Create RedespredeterminadasLabel
app.RedespredeterminadasLabel = uilabel(app.
    PanelCargarRed);
app.RedespredeterminadasLabel.
    HorizontalAlignment = 'right';
app.RedespredeterminadasLabel.Position = [20
    48 95 30];
app.RedespredeterminadasLabel.Text = {'Redes
          '; 'predeterminadas'};

% Create RedesPredetListBox
app.RedesPredetListBox = uilistbox(app.
    PanelCargarRed);
app.RedesPredetListBox.Items = {'-', '
    red_ejemplo_entrenada.mat', 'red_piribiri.
    mat', 'nikeNetwork.mat'};
app.RedesPredetListBox.ValueChangedFcn =
    createCallbackFcn(app, @
    RedesPredetListBoxValueChanged, true);
```

```
app.RedesPredetListBox.Position = [130 16 185
    64];
app.RedesPredetListBox.Value = '-';

% Create PanelCargarDatosClasif
app.PanelCargarDatosClasif = uipanel(app.
   UIFigure);
app.PanelCargarDatosClasif.Position = [36 89
   569 137];

% Create ExplorarDatosButton
app.ExplorarDatosButton = uibutton(app.
   PanelCargarDatosClasif, 'push');
app.ExplorarDatosButton.ButtonPushedFcn =
   createCallbackFcn(app, @
   ExplorarDatosButtonPushed, true);
app.ExplorarDatosButton.FontWeight = 'bold';
app.ExplorarDatosButton.Position = [67 97 173
    23];
app.ExplorarDatosButton.Text = 'Cargar datos
   para clasificar';

% Create ArchivoDatosLabel
app.ArchivoDatosLabel = uilabel(app.
   PanelCargarDatosClasif);
app.ArchivoDatosLabel.FontWeight = 'bold';
app.ArchivoDatosLabel.Position = [337 76 54
   22];
app.ArchivoDatosLabel.Text = 'Archivo:';

% Create LabelDatosRuta
app.LabelDatosRuta = uilabel(app.
   PanelCargarDatosClasif);
app.LabelDatosRuta.Position = [337 49 216
   22];
app.LabelDatosRuta.Text = 'No se ha
   seleccionado ning n archivo.';

% Create RedespredeterminadasLabel_2
app.RedespredeterminadasLabel_2 = uilabel(app
   .PanelCargarDatosClasif);
app.RedespredeterminadasLabel_2.
   HorizontalAlignment = 'right';
app.RedespredeterminadasLabel_2.Position =
   [20 47 95 30];
app.RedespredeterminadasLabel_2.Text = {'
```

```
    Datos         '; 'predeterminados'};

% Create DatosPredetListBox
app.DatosPredetListBox = uilistbox(app.
    PanelCargarDatosClasif);
app.DatosPredetListBox.Items = {'-', '
    HumanActivityTest', 'PerroActivity', '
    Jurinnniiini'};
app.DatosPredetListBox.ValueChangedFcn =
    createCallbackFcn(app, @
    DatosPredetListBoxValueChanged, true);
app.DatosPredetListBox.Position = [130 15 185
     64];
app.DatosPredetListBox.Value = '-';

% Create MostrarlosdatosButton
app.MostrarlosdatosButton = uibutton(app.
    PanelCargarDatosClasif, 'push');
app.MostrarlosdatosButton.ButtonPushedFcn =
    createCallbackFcn(app, @
    MostrarlosdatosButtonPushed, true);
app.MostrarlosdatosButton.FontWeight = 'bold'
    ;
app.MostrarlosdatosButton.Enable = 'off';
app.MostrarlosdatosButton.Position = [383 14
    115 23];
app.MostrarlosdatosButton.Text = 'Mostrar los
     datos';

% Create PrimerPasoLabel
app.PrimerPasoLabel = uilabel(app.UIFigure);
app.PrimerPasoLabel.BackgroundColor = [0.9686
     0.8 0.4902];
app.PrimerPasoLabel.FontName = 'Georgia Pro';
app.PrimerPasoLabel.FontSize = 13;
app.PrimerPasoLabel.Position = [36 442 309
    22];
app.PrimerPasoLabel.Text = ' Primero
    selecciona una red entrenada previamente:'
    ;

% Create SegundoPasoLabel
app.SegundoPasoLabel = uilabel(app.UIFigure);
app.SegundoPasoLabel.BackgroundColor =
    [0.9686 0.8 0.4902];
app.SegundoPasoLabel.FontName = 'Georgia Pro'
```

```
            ;
        app.SegundoPasoLabel.FontSize = 13;
        app.SegundoPasoLabel.Position = [36 240 320
            22];
        app.SegundoPasoLabel.Text = 'Ahora elige el
            conjunto de datos que quieras clasificar:'
            ;

        % Create CLASIFICARDATOSButton
        app.CLASIFICARDATOSButton = uibutton(app.
            UIFigure, 'push');
        app.CLASIFICARDATOSButton.ButtonPushedFcn =
            createCallbackFcn(app, @
            CLASIFICARDATOSButtonPushed, true);
        app.CLASIFICARDATOSButton.FontWeight = 'bold'
            ;
        app.CLASIFICARDATOSButton.Position = [217 39
            208 31];
        app.CLASIFICARDATOSButton.Text = 'CLASIFICAR
            DATOS';

        % Create ClasificarOKLabel
        app.ClasificarOKLabel = uilabel(app.UIFigure)
            ;
        app.ClasificarOKLabel.Position = [231 10 194
            22];
        app.ClasificarOKLabel.Text = '';

        % Show the figure after all components are
            created
        app.UIFigure.Visible = 'on';
    end
end

% App creation and deletion
methods (Access = public)

    % Construct app
    function app = pantallaTest

        % Create UIFigure and components
        createComponents(app)

        % Register the app with App Designer
        registerApp(app, app.UIFigure)
```

```
            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

B.2.3. Pantalla de resultados (pantallaResultados.mlapp)

```
classdef pantallaResultados < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure           matlab.ui.Figure
        CalcularButton     matlab.ui.control.Button
        PrecLabel          matlab.ui.control.Label
        PrecGauge          matlab.ui.control.LinearGauge
        PRECISINLabel      matlab.ui.control.Label
        Label_2            matlab.ui.control.Label
        CompararButton     matlab.ui.control.Button
        Label              matlab.ui.control.Label
        SegundoPasoLabel   matlab.ui.control.Label
    end


    properties (Access = public)
        YT % Description
        YPred
        Acc % Description
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: CompararButton
        function CompararButtonPushed(app, event)
```

```
            Yp = app.YPred;
            Yt = app.YT;

            Ac = app.Acc;
            Ac

            figure
            plot(Yp,'.-')
            hold on
            plot(Yt)
            hold off
            xlabel("Paso de tiempo")
            ylabel("Movimiento")
            title("Comparaci n de la predicci n y la
               muestra")
            legend(["Predicho" "Correcto"])
        end

        % Button pushed function: CalcularButton
        function CalcularButtonPushed(app, event)
            Ac = app.Acc;
            app.PrecGauge.Value = Ac; %%dara error si no
               lo llamamos desde Test
            app.PrecLabel.Text = num2str(Ac);
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all
               components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [0.949 0.8706 0.6784];
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create SegundoPasoLabel
            app.SegundoPasoLabel = uilabel(app.UIFigure);
            app.SegundoPasoLabel.BackgroundColor =
               [0.9686 0.8 0.4902];
            app.SegundoPasoLabel.FontName = 'Georgia Pro'
               ;
```

```
app.SegundoPasoLabel.FontSize = 13;
app.SegundoPasoLabel.Position = [34 432 469
   22];
app.SegundoPasoLabel.Text = ' Los datos se
   han clasificado correctamente. Vamos a
   comprobar los resultados:';

% Create Label
app.Label = uilabel(app.UIFigure);
app.Label.BackgroundColor = [0.7098 0.9216
   0.9608];
app.Label.Position = [91 368 470 30];
app.Label.Text = {'        La precisi n es
   un buen indicador de la calidad del
   entrenamiento de la red.'; '   Se calcula
   dividiendo el n mero de aciertos entre el
    n mero total de clasificaciones.'};

% Create CompararButton
app.CompararButton = uibutton(app.UIFigure, '
   push');
app.CompararButton.ButtonPushedFcn =
   createCallbackFcn(app, @
   CompararButtonPushed, true);
app.CompararButton.FontWeight = 'bold';
app.CompararButton.Position = [202 114 233
   50];
app.CompararButton.Text = 'VER LA GR FICA DE
    COMPARACI N';

% Create Label_2
app.Label_2 = uilabel(app.UIFigure);
app.Label_2.BackgroundColor = [0.7098 0.9216
   0.9608];
app.Label_2.Position = [91 183 474 30];
app.Label_2.Text = {'  Visualizar los
   movimientos predichos en comparaci n con
   los movimientos correctos '; '
   es la mejor manera de ubicar los fallos y
   lograr un mayor entendimiento. '};

% Create PRECISINLabel
app.PRECISINLabel = uilabel(app.UIFigure);
app.PRECISINLabel.HorizontalAlignment = '
   center';
app.PRECISINLabel.FontWeight = 'bold';
```

```
            app.PRECISINLabel.Position = [262 267 75 22];
            app.PRECISINLabel.Text = 'PRECISI N:';

            % Create PrecGauge
            app.PrecGauge = uigauge(app.UIFigure, 'linear
               ');
            app.PrecGauge.Limits = [0.99 1];
            app.PrecGauge.Position = [34 304 567 41];

            % Create PrecLabel
            app.PrecLabel = uilabel(app.UIFigure);
            app.PrecLabel.FontWeight = 'bold';
            app.PrecLabel.Position = [337 267 72 22];
            app.PrecLabel.Text = '';

            % Create CalcularButton
            app.CalcularButton = uibutton(app.UIFigure, '
               push');
            app.CalcularButton.ButtonPushedFcn =
               createCallbackFcn(app, @
               CalcularButtonPushed, true);
            app.CalcularButton.Position = [146 267 100
               23];
            app.CalcularButton.Text = 'Calcular';

            % Show the figure after all components are
               created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = pantallaResultados

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
```

```
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

B.2.4. Pantalla de entrenamiento (pantallaTrain.mlapp)

```
classdef pantallaTrain < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        ListoButton                     matlab.ui.control.
           StateButton
        TercerPasoLabel                 matlab.ui.control.
           Label
        SegundPasoLabel                 matlab.ui.control.
           Label
        PrimerPasoLabel                 matlab.ui.control.
           Label
        CargarPanel                     matlab.ui.container.
           Panel
        NumBloqueEditField              matlab.ui.control.
           NumericEditField
        NmerodebloqueEditFieldLabel     matlab.ui.control.
           Label
        CaracteristicaDropDown          matlab.ui.control.
           DropDown
        EligelacaractersticaarepresentarDropDownLabel
           matlab.ui.control.Label
        NumBloquesLabel                 matlab.ui.control.
           Label
        GraficaButton                   matlab.ui.control.
           Button
        Label                           matlab.ui.control.
           Label
        ProcesoCargaLamp                matlab.ui.control.
           Lamp
```

```
        CargarLabel                     matlab.ui.control.
            Label
        CargarButton                    matlab.ui.control.
            Button
        ElegirDatosPanel                matlab.ui.container.
            Panel
        CargarPredetListBox             matlab.ui.control.
            ListBox
        SeleccionardatospredeterminadosLabel  matlab.ui.
            control.Label
        NombredelarchivoLabel           matlab.ui.control.
            Label
        LabelRuta                       matlab.ui.control.
            Label
        ExplorarButton                  matlab.ui.control.
            Button
    end


    properties (Access = private)  %variables globales
        Datos % Description
        pantallaParametros % Description
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: CargarButton
        function CargarButtonPushed(app, event)
            %value = app.CargarButton.Value;
            nombreArchivo = app.LabelRuta.Text;
            if isequal(nombreArchivo, 'No se ha
                seleccionado ning n archivo.') %si hay
                archivo
                  app.ProcesoCargaLamp.Color = 'red';
            else
                 app.ProcesoCargaLamp.Color = 'yellow';
                 try
                     app.Datos = load(nombreArchivo, '
                        XTrain', 'YTrain'); %VARIABLE
                        GLOBAL
                     %size(app.Datos.XTrain,1) %muestra
                        num de filas Xtrain
                     app.ProcesoCargaLamp.Color = 'green';
                     app.CargarLabel.Text = ['Carga
```

```
                completetada, dimensiones ', num2str
                (size(app.Datos.XTrain,1)), 'x',
                num2str(size(app.Datos.XTrain,2))
                ];
            app.GraficaButton.Enable = "on";
            app.NumBloqueEditField.Enable = "on";
            app.CaracteristicaDropDown.Enable = "
                on";
            app.ListoButton.Visible = "on";
            app.TercerPasoLabel.Visible = "on";
        catch ME
            % Capturar el error y mostrar un
                mensaje adecuado
            if strcmp(ME.identifier, 'MATLAB:load
                :couldNotReadFile')
                app.CargarLabel.Text = ['No se
                    encontr  el archivo ',
                    nombreArchivo];
            else
                app.CargarLabel.Text = ['Error al
                    cargar el archivo ',
                    nombreArchivo];
            end
            app.ProcesoCargaLamp.Color = 'red';
            app.GraficaButton.Enable = "off";
            app.NumBloqueEditField.Enable = "off
                ";
            app.CaracteristicaDropDown.Enable = "
                off";
            app.ListoButton.Visible = "off";
            app.TercerPasoLabel.Visible = "off";
        end
    end

end

% Button pushed function: ExplorarButton
function ExplorarButtonPushed(app, event)
     %[filename, filepath] = uigetfile(filter,
        dialogTitle);
    %filter es para el tipo de archivo como '*.
       txt'
    [filename, ~] = uigetfile('*.*', 'Selecciona
       un archivo');

    % Comprueba si el usuario ha seleccionado un
```

```
        archivo
    if isequal(filename, 0) %compara 2 vars. A y
        B
        app.LabelRuta.Text = 'No se ha
            seleccionado ning n archivo.';
        app.CargarPredetListBox.Value = '-';
    else
        %app.NosehaseleccionadoningnarchivoLabel.
            Text = ['Archivo seleccionado: ',
            fullfile(filepath, filename)];
        app.LabelRuta.Text = filename;
        app.CargarPredetListBox.Value = '-';
        %load filename;
    end
end

% Value changed function: CargarPredetListBox
function CargarPredetListBoxValueChanged(app,
    event)
    value = app.CargarPredetListBox.Value;
    if isequal(value, '-') %si se elige '-'
        app.LabelRuta.Text = 'No se ha
            seleccionado ning n archivo.';
    else
        app.LabelRuta.Text = value;
    end
end

% Button pushed function: GraficaButton
function GraficaButtonPushed(app, event)
    %inicializar XTRAIN e YTRAIN
    coord = app.CaracteristicaDropDown.ValueIndex
        ;
    coordLetra = 'x';
    switch coord
        case 2
            coordLetra = 'y';
        case 3
            coordLetra = 'z';
    end

    numBloque = app.NumBloqueEditField.Value;

    X = app.Datos.XTrain{numBloque}(coord,:);
    classes = categories(app.Datos.YTrain{
        numBloque});
```

```
            figure(1)
            for j = 1:numel(classes)
                label = classes(j);
                idx = find(app.Datos.YTrain{numBloque} ==
                    label);
                hold on
                plot(idx,X(idx))
            end
            hold off

            xlabel("Paso de Tiempo")
            ylabel("Aceleraci n")
            title(['Secuencia de entrenamiento ' num2str(
               numBloque) ', atributo ' num2str(coord) '
               (' coordLetra ')'])
            legend(classes,'Location','northwest')
        end

        % Value changed function: ListoButton
        function ListoButtonValueChanged(app, event)

            app.UIFigure.Visible = 'off';
            app.pantallaParametros = pantallaParametros()
               ;
            app.pantallaParametros.Datos = app.Datos;
            app.pantallaParametros.UIFigure.Visible = 'on
               ';
        end

        % Value changed function: NumBloqueEditField
        function NumBloqueEditFieldValueChanged(app,
           event)
            value = app.NumBloqueEditField.Value;
            numBloques = size(app.Datos.XTrain,1);
            if value > numBloques
                app.NumBloquesLabel.Text = ['Introduce un
                    n mero entre 1 y ', num2str(
                   numBloques)];
                app.NumBloqueEditField.Value = 1;
            else
                app.NumBloquesLabel.Text = ['Valor
                   correcto'];
            end
        end
    end
```

```
% Component initialization
methods (Access = private)

    % Create UIFigure and components
    function createComponents(app)

        % Create UIFigure and hide until all
           components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.Color = [0.949 0.8706 0.6784];
        app.UIFigure.Position = [100 100 640 480];
        app.UIFigure.Name = 'MATLAB App';

        % Create ElegirDatosPanel
        app.ElegirDatosPanel = uipanel(app.UIFigure);
        app.ElegirDatosPanel.BorderType = 'none';
        app.ElegirDatosPanel.BackgroundColor = [0.949
            0.8706 0.6784];
        app.ElegirDatosPanel.Position = [23 281 595
           134];

        % Create ExplorarButton
        app.ExplorarButton = uibutton(app.
           ElegirDatosPanel, 'push');
        app.ExplorarButton.ButtonPushedFcn =
           createCallbackFcn(app, @
           ExplorarButtonPushed, true);
        app.ExplorarButton.FontWeight = 'bold';
        app.ExplorarButton.Position = [321 77 269
           23];
        app.ExplorarButton.Text = 'Selecciona tus
           archivos .mat del explorador';

        % Create LabelRuta
        app.LabelRuta = uilabel(app.ElegirDatosPanel)
           ;
        app.LabelRuta.Position = [259 10 216 22];
        app.LabelRuta.Text = 'No se ha seleccionado
           ning n archivo.';

        % Create NombredelarchivoLabel
        app.NombredelarchivoLabel = uilabel(app.
           ElegirDatosPanel);
        app.NombredelarchivoLabel.FontWeight = 'bold'
           ;
```

```
app.NombredelarchivoLabel.Position = [139 10
    121 22];
app.NombredelarchivoLabel.Text = 'Nombre del
    archivo:';

% Create SeleccionardatospredeterminadosLabel
app.SeleccionardatospredeterminadosLabel =
    uilabel(app.ElegirDatosPanel);
app.SeleccionardatospredeterminadosLabel.
    HorizontalAlignment = 'right';
app.SeleccionardatospredeterminadosLabel.
    FontWeight = 'bold';
app.SeleccionardatospredeterminadosLabel.
    Position = [9 70 125 30];
app.SeleccionardatospredeterminadosLabel.Text
     = {'Seleccionar archivos'; '
    predeterminados:'};

% Create CargarPredetListBox
app.CargarPredetListBox = uilistbox(app.
    ElegirDatosPanel);
app.CargarPredetListBox.Items = {'-', '
    HumanActivityTrain', 'MovementDataset', '
    DBHumanMoves'};
app.CargarPredetListBox.ValueChangedFcn =
    createCallbackFcn(app, @
    CargarPredetListBoxValueChanged, true);
app.CargarPredetListBox.Position = [151 53
    152 65];
app.CargarPredetListBox.Value = '-';

% Create CargarPanel
app.CargarPanel = uipanel(app.UIFigure);
app.CargarPanel.ForegroundColor = [0.902
    0.902 0.902];
app.CargarPanel.BorderType = 'none';
app.CargarPanel.BackgroundColor = [0.949
    0.8706 0.6784];
app.CargarPanel.Position = [21 64 570 165];

% Create CargarButton
app.CargarButton = uibutton(app.CargarPanel,
    'push');
app.CargarButton.ButtonPushedFcn =
    createCallbackFcn(app, @CargarButtonPushed
    , true);
```

```
app.CargarButton.Position = [29 131 253 23];
app.CargarButton.Text = 'Cargar los datos de
    entrenamiento elegidos:';

% Create CargarLabel
app.CargarLabel = uilabel(app.CargarPanel);
app.CargarLabel.Position = [334 131 225 22];
app.CargarLabel.Text = 'A n no ha comenzado
    la descarga';

% Create ProcesoCargaLamp
app.ProcesoCargaLamp = uilamp(app.CargarPanel
    );
app.ProcesoCargaLamp.Position = [302 132 20
    20];
app.ProcesoCargaLamp.Color = [1 0 0];

% Create Label
app.Label = uilabel(app.CargarPanel);
app.Label.HorizontalAlignment = 'right';
app.Label.Position = [286 131 49 22];
app.Label.Text = {''; ''};

% Create GraficaButton
app.GraficaButton = uibutton(app.CargarPanel,
    'push');
app.GraficaButton.ButtonPushedFcn =
    createCallbackFcn(app, @
    GraficaButtonPushed, true);
app.GraficaButton.Enable = 'off';
app.GraficaButton.Position = [281 43 267 43];
app.GraficaButton.Text = {'Pulsa para
    visualizar una secuencia'; ' de
    entrenamiento en una gr fica'};

% Create NumBloquesLabel
app.NumBloquesLabel = uilabel(app.CargarPanel
    );
app.NumBloquesLabel.Position = [110 11 214
    22];
app.NumBloquesLabel.Text = '';

% Create
    EligelacaractersticaarepresentarDropDownLabel

app.
```

```
    EligelacaractersticaarepresentarDropDownLabel
     = uilabel(app.CargarPanel);
app.
    EligelacaractersticaarepresentarDropDownLabel
    .HorizontalAlignment = 'right';
app.
    EligelacaractersticaarepresentarDropDownLabel
    .Enable = 'off';
app.
    EligelacaractersticaarepresentarDropDownLabel
    .Position = [21 68 119 30];
app.
    EligelacaractersticaarepresentarDropDownLabel
    .Text = {'Elige la caracter stica'; ' a
    representar:'};

% Create CaracteristicaDropDown
app.CaracteristicaDropDown = uidropdown(app.
    CargarPanel);
app.CaracteristicaDropDown.Items = {'x', 'y',
     'z'};
app.CaracteristicaDropDown.Enable = 'off';
app.CaracteristicaDropDown.Position = [155 76
     81 22];
app.CaracteristicaDropDown.Value = 'x';

% Create NmerodebloqueEditFieldLabel
app.NmerodebloqueEditFieldLabel = uilabel(app
    .CargarPanel);
app.NmerodebloqueEditFieldLabel.
    HorizontalAlignment = 'right';
app.NmerodebloqueEditFieldLabel.Enable = 'off
    ';
app.NmerodebloqueEditFieldLabel.Position =
    [29 43 107 22];
app.NmerodebloqueEditFieldLabel.Text = '
    N mero de bloque:';

% Create NumBloqueEditField
app.NumBloqueEditField = uieditfield(app.
    CargarPanel, 'numeric');
app.NumBloqueEditField.Limits = [1 Inf];
app.NumBloqueEditField.ValueChangedFcn =
    createCallbackFcn(app, @
    NumBloqueEditFieldValueChanged, true);
app.NumBloqueEditField.Enable = 'off';
```

```
app.NumBloqueEditField.Position = [151 43 93
   22];
app.NumBloqueEditField.Value = 1;

% Create PrimerPasoLabel
app.PrimerPasoLabel = uilabel(app.UIFigure);
app.PrimerPasoLabel.BackgroundColor = [0.9686
    0.8 0.4902];
app.PrimerPasoLabel.FontName = 'Georgia Pro';
app.PrimerPasoLabel.FontSize = 13;
app.PrimerPasoLabel.Position = [72 430 496
   22];
app.PrimerPasoLabel.Text = ' Primero
   selecciona un archivo de datos de
   entrenamiento.  Cuidado  con el formato!';

% Create SegundPasoLabel
app.SegundPasoLabel = uilabel(app.UIFigure);
app.SegundPasoLabel.BackgroundColor = [0.9686
    0.8 0.4902];
app.SegundPasoLabel.FontName = 'Georgia Pro';
app.SegundPasoLabel.FontSize = 13;
app.SegundPasoLabel.Position = [112 241 418
   22];
app.SegundPasoLabel.Text = ' Si est s listo,
    pulsa el bot n inferior para cargar los
   datos al programa.';

% Create TercerPasoLabel
app.TercerPasoLabel = uilabel(app.UIFigure);
app.TercerPasoLabel.BackgroundColor = [0.9686
    0.8 0.4902];
app.TercerPasoLabel.FontName = 'Georgia Pro';
app.TercerPasoLabel.FontSize = 13;
app.TercerPasoLabel.Visible = 'off';
app.TercerPasoLabel.Position = [145 43 321
   22];
app.TercerPasoLabel.Text = ' Datos  listos!
   Vamos a definir la arquitectura de la red.
   ';

% Create ListoButton
app.ListoButton = uibutton(app.UIFigure, '
   state');
app.ListoButton.ValueChangedFcn =
   createCallbackFcn(app, @
```

```
                ListoButtonValueChanged, true);
            app.ListoButton.Visible = 'off';
            app.ListoButton.Text = 'LISTO';
            app.ListoButton.Position = [255 12 100 23];

            % Show the figure after all components are
                created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = pantallaTrain

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

B.2.5. Pantalla de parámetros de entrenamiento (pantallaParametros.mlapp)

```
classdef pantallaParametros < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
```

```
        UIFigure                          matlab.ui.Figure
        RazonAprendKnob                   matlab.ui.control
           .DiscreteKnob
        RazndeaprendizajeKnobLabel_2      matlab.ui.control
           .Label
        Panel                             matlab.ui.
           container.Panel
        NumMinilotesGauge                 matlab.ui.control
           .SemicircularGauge
        NumEpocasGauge                    matlab.ui.control
           .SemicircularGauge
        NumMinilotesEditField             matlab.ui.control
           .NumericEditField
        Nmerodeminilotes111EditFieldLabel  matlab.ui.
           control.Label
        NumEpocasEditField                matlab.ui.control
           .NumericEditField
        Nmerodepocas1100EditFieldLabel    matlab.ui.control
           .Label
        ComenzarEntrenSwitch              matlab.ui.control
           .Switch
        TodoajustadoPuesvamosaempezarLabel  matlab.ui.
           control.Label
        NotaLabel                         matlab.ui.control
           .Label
        UmbralGradienteKnob               matlab.ui.control
           .DiscreteKnob
        UmbraldelgradienteLabel           matlab.ui.control
           .Label
        OptimizadorButtonGroup            matlab.ui.
           container.ButtonGroup
        adamButton                        matlab.ui.control
           .RadioButton
        sgdmButton                        matlab.ui.control
           .RadioButton
        PrimerPasoLabel                   matlab.ui.control
           .Label
    end


    properties (Access = public)
        Datos %Public para poder pasarlos desde la otra
           pantalla
    end

    properties (Access = private)
```

```
        layers % Description
    end



    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            numFeatures = 3;
            numHiddenUnits = 200;
            numClasses = 5;

            app.layers = [ ...
                sequenceInputLayer(numFeatures)
                lstmLayer(numHiddenUnits,'OutputMode','
                   sequence')
                fullyConnectedLayer(numClasses)
                softmaxLayer
                classificationLayer];
        end

        % Value changed function: ComenzarEntrenSwitch
        function ComenzarEntrenSwitchValueChanged(app,
           event)
            value = app.ComenzarEntrenSwitch.Value;
            %cuando empezamos el entrenamiento,
               bloqueamos tdos los botones
            %----PONER BOTON PARAR ENTRENAMIENTO QUE LOS
               ACTIVE OTRA VEZ
            app.ComenzarEntrenSwitch.Enable = "off";
            app.OptimizadorButtonGroup.Enable = "off";
            app.UmbralGradienteKnob.Enable = "off";
            app.RazonAprendKnob.Enable = "off";
            app.NumEpocasEditField.Enable = "off";
            app.NumMinilotesEditField.Enable = "off";

            %comenzar entrenamiento
            %NOTA: LOS GAUGE NO SE MODIFICAN CON EL CLICK
                SINO QUE HAY QUE
            %JUNTARLOS CON UN TEXTLABEL Y REPRESENTAN EL
               VALOR QUE HAYA.
            optimizador = app.OptimizadorButtonGroup.
               SelectedObject.Text; %dev el string(nombre
               ) del boton seleccionado
```

```
        epocas = app.NumEpocasEditField.Value;
                    %dev un int
        umbralGrad = str2double(app.
           UmbralGradienteKnob.Value);   %value dev.
           string y lo pasamos a double
        razonAp = str2double(app.RazonAprendKnob.
           Value);   %value dev. string y lo pasamos
           a double

        %ESTA HAY QUE CAMBIARLA AL TAMA O DEL
           MINILOTE
        minilotes = app.NumMinilotesEditField.Value;
                   %dev un int

        options = trainingOptions(optimizador, ...
            'MaxEpochs',epocas, ...
            'GradientThreshold',umbralGrad, ...
            'Verbose',0, ...
            'Plots','training-progress',...
            'InitialLearnRate', razonAp);

        %ENTRENAMOS
        Xt = app.Datos.XTrain;
        Yt = app.Datos.YTrain;
        capas = app.layers;
        net = trainNetwork(Xt, Yt, capas, options);

    end

    % Value changed function: NumEpocasEditField
    function NumEpocasEditFieldValueChanged(app,
       event)
        value = app.NumEpocasEditField.Value;
        if value > 0 && value < 101
            app.NumEpocasGauge.Value = value;
        else
            app.NumEpocasGauge.Value = 60;
            app.NumEpocasEditField.Value = 60;
        end
    end

    % Value changed function: NumMinilotesEditField
    function NumMinilotesEditFieldValueChanged(app,
       event)
        value = app.NumMinilotesEditField.Value;
        if value > 0 && value < 12
```

```
                app.NumMinilotesGauge.Value = value;
            else
                app.NumMinilotessGauge.Value = 1;
                app.NumMinilotesEditField.Value = 1;
            end
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all
               components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [0.949 0.8706 0.6784];
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create PrimerPasoLabel
            app.PrimerPasoLabel = uilabel(app.UIFigure);
            app.PrimerPasoLabel.BackgroundColor = [0.9686
                0.8 0.4902];
            app.PrimerPasoLabel.FontName = 'Georgia Pro';
            app.PrimerPasoLabel.FontSize = 13;
            app.PrimerPasoLabel.Position = [60 427 521
               22];
            app.PrimerPasoLabel.Text = ' Ha llegado el
               momento de elegir los par metros de
               entrenamiento de nuestra red LSTM';

            % Create OptimizadorButtonGroup
            app.OptimizadorButtonGroup = uibuttongroup(
               app.UIFigure);
            app.OptimizadorButtonGroup.Title = '
               Optimizador';
            app.OptimizadorButtonGroup.FontWeight = 'bold
               ';
            app.OptimizadorButtonGroup.Position = [255
               316 124 85];

            % Create sgdmButton
            app.sgdmButton = uiradiobutton(app.
               OptimizadorButtonGroup);
```

```
app.sgdmButton.Text = 'sgdm';
app.sgdmButton.FontSize = 13;
app.sgdmButton.Position = [33 32 68 27];
app.sgdmButton.Value = true;

% Create adamButton
app.adamButton = uiradiobutton(app.
   OptimizadorButtonGroup);
app.adamButton.Text = 'adam';
app.adamButton.FontSize = 13;
app.adamButton.Position = [33 3 68 26];

% Create UmbraldelgradienteLabel
app.UmbraldelgradienteLabel = uilabel(app.
   UIFigure);
app.UmbraldelgradienteLabel.
   HorizontalAlignment = 'center';
app.UmbraldelgradienteLabel.FontWeight = '
   bold';
app.UmbraldelgradienteLabel.Position = [64
   393 124 22];
app.UmbraldelgradienteLabel.Text = 'Umbral
   del gradiente';

% Create UmbralGradienteKnob
app.UmbralGradienteKnob = uiknob(app.UIFigure
   , 'discrete');
app.UmbralGradienteKnob.Items = {'0.5', '1',
   '2', '3', '5'};
app.UmbralGradienteKnob.Position = [98 302 60
    60];
app.UmbralGradienteKnob.Value = '2';

% Create NotaLabel
app.NotaLabel = uilabel(app.UIFigure);
app.NotaLabel.Position = [110 92 451 30];
app.NotaLabel.Text = {'    NOTA: Si se
   aumenta el n mero de minilotes (ej. 3),
   las actualizaciones/ poca '; ' se
   incrementan proporcionalmente (ej. x3) y
   con ello, el tiempo de entrenamiento.'};

% Create TodoajustadoPuesvamosaempezarLabel
app.TodoajustadoPuesvamosaempezarLabel =
   uilabel(app.UIFigure);
app.TodoajustadoPuesvamosaempezarLabel.
```

```
    BackgroundColor = [0.9686 0.8 0.4902];
app.TodoajustadoPuesvamosaempezarLabel.
    HorizontalAlignment = 'center';
app.TodoajustadoPuesvamosaempezarLabel.
    FontName = 'Georgia Pro';
app.TodoajustadoPuesvamosaempezarLabel.
    FontSize = 13;
app.TodoajustadoPuesvamosaempezarLabel.
    Position = [204 56 233 22];
app.TodoajustadoPuesvamosaempezarLabel.Text =
     ' Todo  ajustado? Pues vamos a empezar';

% Create ComenzarEntrenSwitch
app.ComenzarEntrenSwitch = uiswitch(app.
    UIFigure, 'slider');
app.ComenzarEntrenSwitch.ValueChangedFcn =
    createCallbackFcn(app, @
    ComenzarEntrenSwitchValueChanged, true);
app.ComenzarEntrenSwitch.FontWeight = 'bold';
app.ComenzarEntrenSwitch.Position = [297 30
    45 20];

% Create Panel
app.Panel = uipanel(app.UIFigure);
app.Panel.Position = [38 133 565 146];

% Create Nmerodepocas1100EditFieldLabel
app.Nmerodepocas1100EditFieldLabel = uilabel(
    app.Panel);
app.Nmerodepocas1100EditFieldLabel.
    HorizontalAlignment = 'right';
app.Nmerodepocas1100EditFieldLabel.FontWeight
     = 'bold';
app.Nmerodepocas1100EditFieldLabel.Position =
     [51 115 162 22];
app.Nmerodepocas1100EditFieldLabel.Text = '
    N mero de  pocas  (1-100): ';

% Create NumEpocasEditField
app.NumEpocasEditField = uieditfield(app.
    Panel, 'numeric');
app.NumEpocasEditField.Limits = [1 100];
app.NumEpocasEditField.ValueChangedFcn =
    createCallbackFcn(app, @
    NumEpocasEditFieldValueChanged, true);
app.NumEpocasEditField.Position = [82 86 100
```

```
    22];
app.NumEpocasEditField.Value = 60;

% Create Nmerodeminilotes111EditFieldLabel
app.Nmerodeminilotes111EditFieldLabel =
    uilabel(app.Panel);
app.Nmerodeminilotes111EditFieldLabel.
    HorizontalAlignment = 'right';
app.Nmerodeminilotes111EditFieldLabel.
    FontWeight = 'bold';
app.Nmerodeminilotes111EditFieldLabel.
    Position = [349 115 166 22];
app.Nmerodeminilotes111EditFieldLabel.Text =
    'N mero de minilotes (1-11): ';

% Create NumMinilotesEditField
app.NumMinilotesEditField = uieditfield(app.
    Panel, 'numeric');
app.NumMinilotesEditField.Limits = [1 11];
app.NumMinilotesEditField.ValueChangedFcn =
    createCallbackFcn(app, @
    NumMinilotesEditFieldValueChanged, true);
app.NumMinilotesEditField.Position = [382 86
    100 22];
app.NumMinilotesEditField.Value = 1;

% Create NumEpocasGauge
app.NumEpocasGauge = uigauge(app.Panel, '
    semicircular');
app.NumEpocasGauge.Limits = [1 100];
app.NumEpocasGauge.MajorTicks = [1 20 40 60
    80 100];
app.NumEpocasGauge.MinorTicks = [1 5 10 15 20
     25 30 35 40 45 50 55 60 65 70 75 80 85 90
     95 100];
app.NumEpocasGauge.Position = [72 10 120 65];
app.NumEpocasGauge.Value = 60;

% Create NumMinilotesGauge
app.NumMinilotesGauge = uigauge(app.Panel, '
    semicircular');
app.NumMinilotesGauge.Limits = [1 11];
app.NumMinilotesGauge.MajorTicks = [1 3 5 7 9
     11];
app.NumMinilotesGauge.MinorTicks = [1 2 3 4 5
     6 7 8 9 10];
```

```
            app.NumMinilotesGauge.Position = [372 10 120
               65];
            app.NumMinilotesGauge.Value = 1;

            % Create RazndeaprendizajeKnobLabel_2
            app.RazndeaprendizajeKnobLabel_2 = uilabel(
               app.UIFigure);
            app.RazndeaprendizajeKnobLabel_2.
               HorizontalAlignment = 'center';
            app.RazndeaprendizajeKnobLabel_2.FontWeight =
                'bold';
            app.RazndeaprendizajeKnobLabel_2.Position =
               [443 393 128 22];
            app.RazndeaprendizajeKnobLabel_2.Text = '
               Raz n de aprendizaje';

            % Create RazonAprendKnob
            app.RazonAprendKnob = uiknob(app.UIFigure, '
               discrete');
            app.RazonAprendKnob.Items = {'0.001', '0.005'
               , '0.02', '0.05', '  0.1'};
            app.RazonAprendKnob.Position = [479 302 60
               60];
            app.RazonAprendKnob.Value = '0.001';

            % Show the figure after all components are
               created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = pantallaParametros

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)
```

```
            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Este texto se puede encontrar en el fichero Cascaras/fin.tex. Si deseas eliminarlo, basta con comentar la línea correspondiente al final del fichero TFGTeXiS.tex.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –

Bien podrán los encantadores quitarme la ventura,

pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero

Don Quijote de la Mancha

Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.

–No es menester firmarla – dijo Don Quijote–,

sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero

Don Quijote de la Mancha

Miguel de Cervantes