UNIVERSIDAD
COMPLUTENSE
MADRID

FACULTAD DE INFORMÁTICA

GRADO EN INGENIERÍA DEL SOFTWARE

TRABAJO DE FIN DE GRADO

TÍTULO

Técnicas de Deep Learning para el reconocimiento de movimientos corporales

Deep Learning techniques for recognizing body movements

AUTOR

Roberto Pavón Benítez

DIRECTOR

Gonzalo Pajares Martinsanz

RESUMEN

En este trabajo se detalla el diseño y desarrollo de un sistema de reconocimiento de actividades humanas (*HAR, human activity recognition*) mediante redes neuronales, así como la labor de investigación realizada sobre técnicas y proyectos similares realizados sobre este mismo campo. El sistema desarrollado consta de varios módulos que hacen uso e interactúan a través de internet, estando en parte alojados en la nube.

Concretamente, el diseño de la aplicación contiene dos módulos principales, una aplicación móvil para sistemas Android y una aplicación web alojada en un servidor. La primera, instalada en un dispositivo móvil, recaba datos de los sensores del dispositivo (sensores de aceleración y movimiento) y los envía al servidor por internet para ser analizados. La aplicación web consiste en una api REST, que hace uso de una red neuronal entrenada para realizar predicciones sobre la actividad que se está realizando, basándose en los datos recibidos. El tipo de red empleado es una red recurrente de tipo *Long-Short-Term Memory (LSTM),* basada en los trabajos publicados por Matlab sobre clasificación y predicción *sequence-to-sequence*. Esta herramienta es también la que se ha empleado para desarrollar el modelo de red y realizar los entrenamientos necesarios.

El proceso de predicción se inicia cuando el usuario selecciona la opción correspondiente en su dispositivo móvil. Desde ese momento, cada dos centésimas de segundo se envía por internet un paquete de datos al servidor, conteniendo información sobre la situación actual en la que se encuentra el aparato según la actividad que se esté realizando. El servidor analiza cada uno de estos paquetes, los procesa y se los pasa a la red neuronal. Ésta realiza una predicción, con cierto grado de seguridad, que se almacena en la base de datos del servidor. Además de realizar predicciones, la aplicación móvil permite también enviar datos con una clase asociada relativa a la actividad que se está realizando. Estos datos se almacenan también en la base de datos y se pueden utilizar posteriormente para reentrenar la red y realizar nuevas pruebas.

Al acceder a la aplicación web a través de una URL desde un navegador, se puede abrir una página web donde se muestra, en tiempo real, la actividad que la red predice que está realizando actualmente el usuario que lleva el dispositivo móvil, junto con el porcentaje de acierto respecto de la predicción realizada.

Se describen las fases de desarrollo de la aplicación comenzando por el estudio y análisis de algoritmos de *deep-learning* a este ámbito de reconocimiento de actividades humanas, así como todo el proceso de ensayo y pruebas realizado sobre distintos modelos de redes para obtener los resultados más óptimos, con resultados satisfactorios.

PALABRAS CLAVE

Redes neuronales recurrentes (RNN) – Memoria a corto y largo plazo (LSTM) – Reconocimiento de actividades humanas (HAR) – Interfaz de programación de aplicaciones (API) – Aprendizaje profundo – Predicción

ABSTRACT

This work explains the design and development of a human activity recognition system (HAR) using neural networks, as well as the research work carried out over this specific field during the process. The developed system consists of several modules that use and interact through the internet, being partly hosted in the cloud.

Specifically, the design of the application consists of two main modules, a smartphone application developed for Android systems and a web application deployed on a server. The Smartphone app collects data from the sensors on the mobile device (acceleration and movement sensors), sending them to the server through the internet to be analysed. The web application consists of a REST api, that uses a previously trained neural network to make predictions about the activity that its being performed based on the received data. The type of network used is a recurrent network of type long-short-term memory (LSTM), based on the works published by Matlab about sequence-to-sequence classification and prediction. This is also the tool used to develop the network model and to perform the required training processes.

The prediction process starts when the user selects the corresponding option on his smartphone. From that moment, every two hundredths of a second a data package is sent through the internet to the server, containing information about the current situation of the device, depending on the activity being carried out. The server analyses every one of this packages, processes them, and feeds them to the neural network. It then makes a prediction, with a certain degree of reliability, which is stored on the server database. Besides the prediction making, the app also allows for sending data tagged with the corresponding class describing the activity currently being performed. This data is also stored in the database and can be used later to train and test the network.

When accessing the web application through a URL in a browser a webpage can be expanded displaying, in real time, the current activity predicted by the network, which is carried out by the user bearing the smartphone, is shown, alongside the certainty of the prediction made.

The development phases of the application are described, starting with the study and analysis of deep-learning algorithms in this field of HAR, as well as the entire trial and testing process carried out on different network models to obtain the optimal solutions, with satisfactory results.

KEYWORDS

Recurrent neural networks (RNN) – Long-short-term memory (LSTM) – Human activity recognition (HAR) – Application programing interface (API) – Deep learning – Prediction

CONTENIDO

1- INTRODUCCIÓN

1.1- RECONOCIMIENTO DE ACTIVIDADES HUMANAS

El reconocimiento de la actividad humana (*HAR, Human Activity Recognition*) constituye un campo de interés de las tecnologías de aprendizaje automático. Define de forma general el campo de predicción de acciones y movimientos físicos que un agente humano lleva a cabo en un momento determinado, tales como caminar, correr, permanecer sentado, entre otras varias, que pueden definirse convenientemente.

Con tal propósito, se toman datos sensoriales relacionados con el movimiento. En el presente proyecto, estos datos de movimiento se obtienen utilizando los sensores contenidos en un *smartphone*. Generalmente, los sensores acoplados en estos dispositivos son: Aceleración, Velocidad Angular, Orientación, Campo Magnético y Posición. Si bien, para el caso de detección de la actividad son los relacionados con el movimiento los realmente eficientes. En el presente proyecto se utilizan los datos de Aceleración, Velocidad Angular y Orientación como base para la determinación de la Actividad pretendida. Así, esta forma de realizar predicciones en base a sensores es distintivamente diferente de otros sistemas basados en visión artificial.

Para el análisis de estos datos se utilizan técnicas de aprendizaje profundo (deep learning), que se procesan en remoto.

1.2- ESTADO DEL ARTE

Actualmente existen diversas aplicaciones de este tipo de tecnologías, que van desde el empleo directo de la información obtenida, por ejemplo, como parte del proceso de rehabilitación de un paciente con lesiones cerebrales mediante asistencia remota o el cuidado de personas ancianas (Kozina y col., 2013); hasta la inserción de estos sistemas en otros más complejos que llevan a cabo procesos automáticos, por ejemplo averiguar en qué piso de un edificio se encuentra alguien analizando simplemente si ha subido escaleras o tomado un ascensor, o deduciendo el número de escalones que ha ascendido y el número de pasos que ha dado (Ye y col., 2012) .

Existen distintos métodos de realizar HAR, pero la forma más efectiva es utilizar algoritmos de deep-learning. Existen dos tipos principales de redes neuronales aplicables a esta actividad: CNN y RNN. Las redes CNN (convolucionales) se desarrollaron para el reconocimiento de imágenes, y son hábiles a la hora de destilar una entrada compleja en elementos esenciales, efectivamente abstrayendo los datos y eliminando detalles innecesarios. Las RNN (recurrentes) son redes en las que la salida de una neurona vuelve a la propia neurona como entrada, añadiendo un componente de tiempo en el modelo. Tienen la peculiaridad de que pueden aprender no solo de datos si no de secuencias de los mismos.

Un caso de proyecto HAR que utiliza redes convolucionales para realizar el reconocimiento es el que describen Zhou y col. (2019). En su proyecto, detallan un método de recogida de datos utilizando los sensores de un Smartphone de forma similar al de este proyecto, segmentando los datos en unidades temporales que sirven de entrada para la red neuronal convolucional. Este sistema es capaz de identificar con un 98% de precisión, en alrededor de dos segundos, nueve tipos distintos de actividades.

Otros proyectos de reconocimiento de actividades, como el propuesto por Abdulmajid-Murad (2017), hacen uso de las redes LSTM, un tipo de RNN (ver punto 2.2), como alternativa superior a las CNN. El uso de estas redes soluciona algunas limitaciones de las redes convolucionales, principalmente el hecho de que las CNN utilizan una matriz de características denominada *kernel* a través de la cual filtran una entrada, lo que implica que la entrada de datos debe tener una longitud fija. Las redes recurrentes utilizadas en este artículo no tienen esta dependencia, lo que les permite utilizar secuencias de entrada de datos de longitud variables, entre otras ventajas como poder capturar dependencias a largo plazo entre los datos de entrada.

Los experimentos descritos en este último artículo indican que las redes recurrentes de tipo LSTM otorgan mejores resultados que métodos de aprendizaje automático convencional, e incluso que otros métodos de *deep learning* como las redes CNN, por lo que en este proyecto se utilizan este tipo de redes recurrentes para realizar las clasificaciones.

1.3- OBJETIVOS

El objetivo final es poder predecir en tiempo real, con un índice aceptable de éxito, la actividad que está llevando a cabo una persona en base a los datos que se reciben desde su smartphone, con especial énfasis en la exploración y comparación de distintas tecnologías y métodos de envío de datos.

Para ello se plantean los siguientes objetivos específicos:

1. Desarrollar un módulo de captura de datos sensoriales. Debe ser capaz de capturar y almacenar muestras de forma consistente, así como realizar capturas en tiempo real para poder realizar predicciones.

2. Diseñar y desarrollar un modelo de red neuronal de tipo Long-Short-Term Memory que se adapte a las necesidades de la aplicación y sea capaz de realizar clasificaciones acertadas de forma consistente.

3. Integrar las técnicas y módulos anteriores para desarrollar la aplicación final.

El plan de trabajo se estructura según estos mismos puntos, que constituyen las tareas del mismo secuenciadas en el tiempo a lo largo del desarrollo de esta propuesta.

1.4- ORGANIZACIÓN DE LA MEMORIA

La memoria se organiza como sigue, además de esta misma sección y su traducción, contiene las siguientes. En la sección tres se describen los métodos y conceptos teóricos aplicados. La sección cuatro describe el diseño de la aplicación. La sección cinco incluye los resultados obtenidos. La sección seis contiene las conclusiones y trabajos futuros. Además, se incluye un apéndice que describe la forma de instalar la aplicación.

2 – INTRODUCTION

2.1- HUMAN ACTIVITY RECOGNITION

Human activity recognition is a field of interest for machine learning technology. It broadly defines the fields of predicting action and physical movement of a human agent in a specific movement, such as walking, running, laying, among others that can be defined as needed.

With such purpose, it takes use of movement related to data acquired from sensors. In this project, the data collected comes from the sensors embedded in a smartphone. Normally, this kind of device contains at least sensors for: acceleration, angular velocity, orientation, magnetic field and position. However, for this specific use of activity detection, only the ones related to movement are really needed. In this project the data used are from acceleration, angular velocity and orientation sensors. This way of using sensors is distinctly different from artificial vision related techniques.

For the analysis of this data, deep-learning techniques are used, by processing the data remotely.

2.2- STATE OF THE ART

Currently, there are several uses for this type of technology, ranging from the direct use of the obtained information, for example, as part of the rehabilitation process for patients with brain lesions using remote assistance, to the insertion of this kind of systems into bigger and more complex ones, that automatize some processes, such as finding out the location of someone inside a building analysing simple data like how many stairs or elevator have they taken or how many steps have they climbed.

There are different ways of performing HAR, but the most effective way is to use deep-learning algorithms. There are two main types of neural networks that can perform this function: CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks). CNN were developed for image recognition, and are exceptionally good for data abstraction and for discriminating unnecessary details. RNN are networks in which the output of a neuron comes back to it as part of a new input, adding a temporal component to the model. They have the peculiarity of being able to learn not only from individual sets of data, but from full sequences as well.

An example of HAR using CNN would be the one described by Zhou y col. (2019). Here they detail a method for using smartphone sensors to acquire data in a similar way than the one used in this project, segmenting those datasets in temporal units that work as inputs for the CNN. This system is able to identify with a precision of 98%, in around two seconds, nine different activities.

Other HAR projects, such as the one reported in (Abdulmajid Murad, 2017), propose using LSTM, a type of RNN, as an alternative superior in performance to CNN. This kind of networks solve some problems arisen in CNN, mostly the fact that CNN use matrices of characteristics, named kernels, to filter inputs. This means such input must have a fixed length. RNN don’t need these structures, allowing them to accept inputs of variable length. This represents an advantage, among others, such as its ability to capture dependencies between long-term inputs.

Experiments described in such reference indicate that LSTM obtain better results than more conventional machine learning methods and even better than other deep learning ones, such as CNN. This justifies the proposed method on this project in order to make predictions.

2.3- OBJECTIVES

The final objective is to be able to predict, in real time, with an acceptable performance, the activity that a person is carrying out based on the data received from the smartphone, with special emphasis on the exploration and comparison of different technologies and data submission methods.

For this, the following specific objectives are proposed:

1. Develop a sensory data capture module, being able to consistently capture and store data samples, as well as capture them, in real time in order to make predictions.
2. Design and develop a Long-Short-Term Memory neural network model adapted to the application requirements, and being able, of consistently, making accurate classifications.
3. Integrate the previous techniques and modules to develop the final application.

The work plan is structured according to these same points, which constitute the tasks sequenced in time throughout the development of this proposal.

3- REVISIÓN DE MÉTODOS Y TÉCNICAS

3.1- MÉTODOS EMPLEADOS

En este proyecto se utilizan redes LSTM (Long-Short Term Memory), que son una subcategoría de RNN que almacena en su memoria una secuencia anterior de entradas. Estas redes prestan atención especial a errores y predicciones realizadas a lo largo del tiempo en una misma secuencia, esto se llama retropropagación a través del tiempo.

En la figura 1 aparece representada graficamente la estructura de una red recursiva. En la mitad inferior se representa con su capa oculta desplegada. En el eje inferior se representa el tiempo. Las conexiones entre unidades de la capa oculta, junto con su peso asociado, se representan de forma horizontal dando a entender la característica secuencial de la red.

3.2- REDES LSTM

Las redes LSTM surgen como un intento de afrontar el mayor problema que surge de implementar una red con memoria long-term, la dificultad para mantener gradientes que se retro-propagan a lo largo de muchos estados. Estos tienden a desaparecer o a explotar, lo cual afecta a la optimización.

En la figura 2 se representa la estructura de una red LSTM de forma muy básica. Se puede ver como las unidades de LSTM están conectadas consigo mismas, siendo esta entrada la salida de la propia unidad en el paso anterior.

La función softmax consiste en una función de normalización, que convierte un vector de números reales en un vector de probabilidades de similar dimensión, asignando a cada valor una probabiliad proporcional de forma exponencial. En definitiva, convierte la salida de un conjunto de pesos a una distribución de probabilidades exponencial.

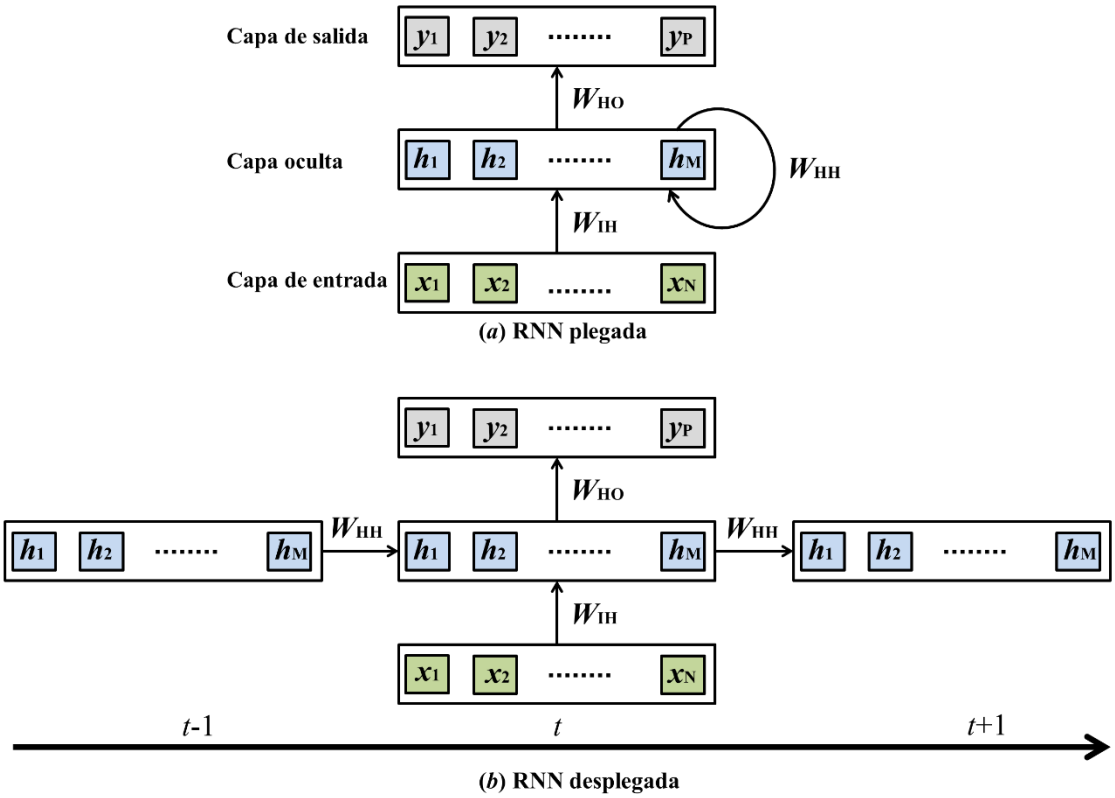


FIGURA 1- ESTRUCTURA DE UNA RNN

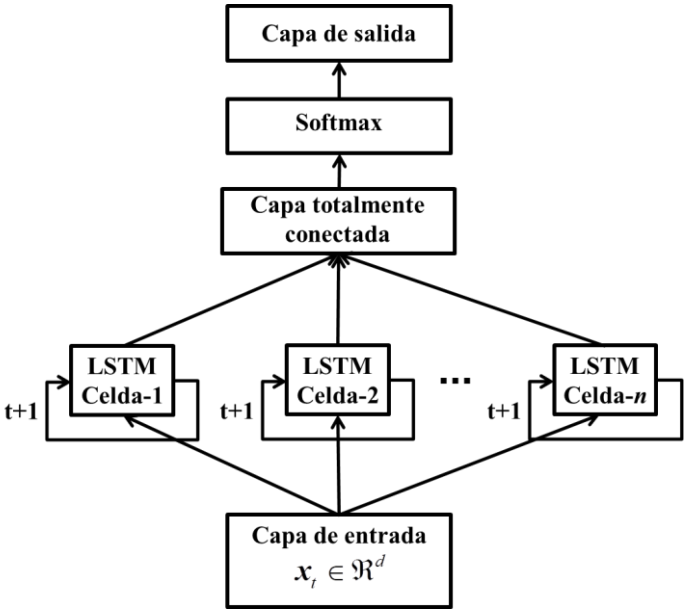


FIGURA 2- ESTRUCTURA DE UNA RED LSTM

Las LSTM fueron introducidas por Hochreiter y Schmidhuber (1997) añadiendo al paradigma de las RNN la funcionalidad de procesar pares de entrada salida.

En la figura 3 se muestra el proceso por el que pasa una celda a lo largo del tiempo. Las variables $\boldsymbol{h}_t$ y $\boldsymbol{x}_t$ respresentan el valor del estado de cada unidad y de la entrada de datos, respectivamente, en el momento de tiempo t. σ_c representa las transformaciones que se realizan sobre la entrada, teniendo en cuenta el valor del salida de la celda anterior y la entrada de datos. Estas transformaciones consisten en añadir o modificar información mediante estructuras llamadas *gates*. Estas estructuras aparecen en el resto de figuras representadas con el símbolo σ, en la figura 4 se puede ver una representación de las *gates* que aparecen en una unidad de este tipo.

La entrada en cada unidad consiste en un par de valores, la entrada de la unidad y la salida anterior, representados $(\boldsymbol{x}_t, \boldsymbol{y}_t)_{t=1}^{T}$. Para cada par $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, la celda LSTM toma una nueva entrada $\boldsymbol{x}_t$ junto con el valor $\boldsymbol{h}_{t-1}$ producido en el paso temporal anterior, y produce una estimación $\hat{\boldsymbol{y}}_t$ para la salida objetivo $\boldsymbol{y}_i$ juntco con un nuevo valor $\boldsymbol{h}_t$ y un nuevo valor del estado de la celda, o memoria.

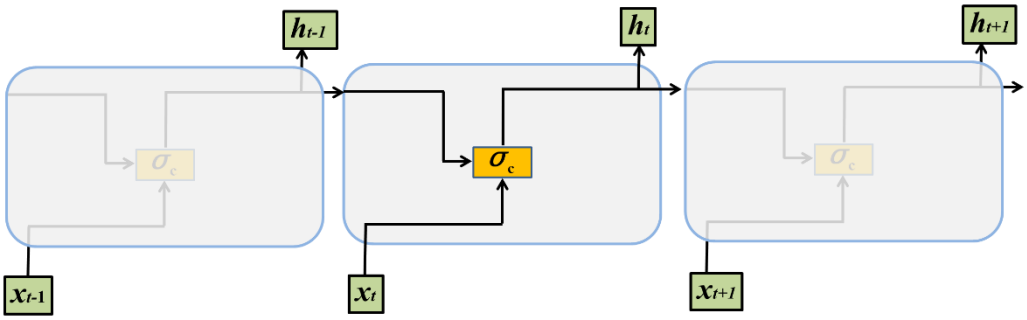


FIGURA 3- ESTRUCTURA DE UNA CELDA SOBRE EL TIEMPO (GRAVES, 2013)

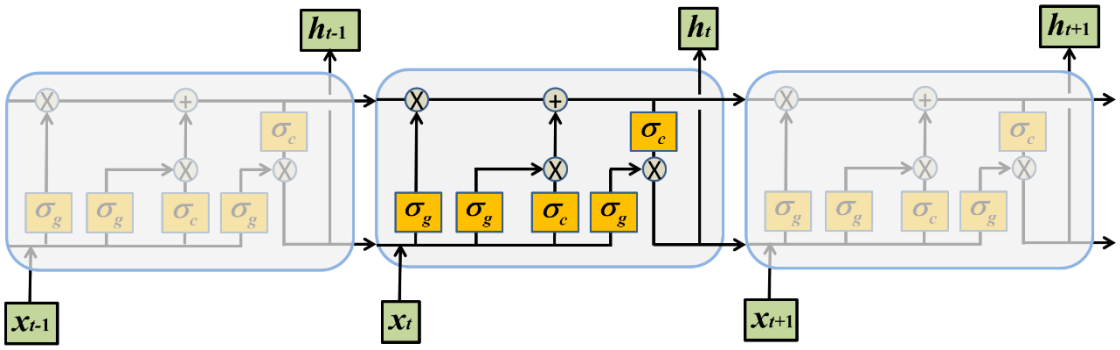


FIGURA 4- ESTRUCTURA CARACTERÍSTICA DE UNA CELDA LSTM (GERS Y COL., 2002)

En la figura 5 se ve la línea de transporte que atraviesa la celda y que compone el estado de la misma. A esta línea de transporte o estado, se le añade información utilizando las *gates* tal y como se ve en la figura 4.

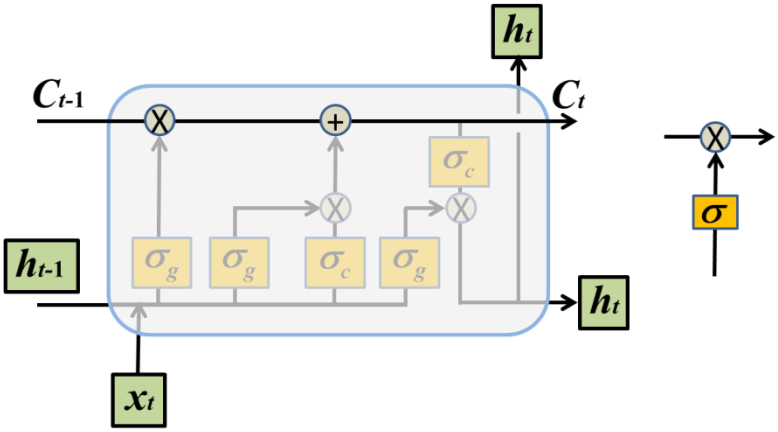


FIGURA 5- LÍNEA DE ESTADO DE LA CELDA

3.2.1- ENTRENAMIENTO

El primer paso de la LSTM es decidir qué información eliminar del estado de la celda. De esto se encarga la *forget gate*, de forma que teniendo en cuenta los pesos de sus neuronas (U_{xf} y W_{hf}) y un *bias* determinado (b_f), utiliza una función sigmoide (σ_g) para generar un valor entre 0 y 1 para cada estado de la celda $\boldsymbol{C}_{t-1}$, de forma que un 1 significa mantener el estado por completo y un 0 deshacerse de él (fig. 6).

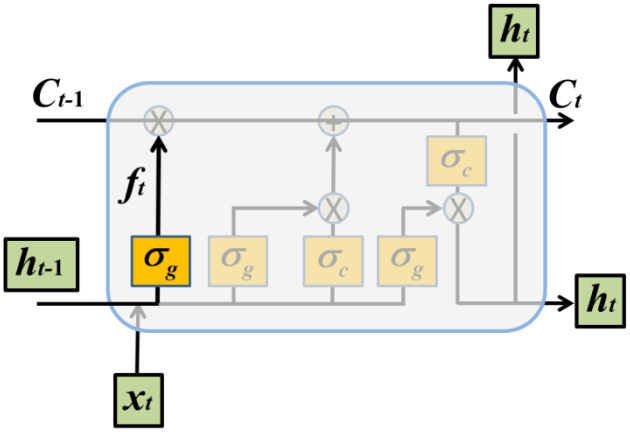


FIGURA 6- FORGET GATE

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

El siguiente paso consiste en decidir qué información se almacena en la celda. Esto se hace en dos partes, primero una capa sigmoide llamada *input gate* (i_t) decide qué valores se actualizan. Seguidamente una capa (σ_c) crea nuevos valores candidatos teniendo en cuenta los pesos U_{xi}, W_{hi}, U_{xc} y W_{hc} junto con los correspondientes *bias*. Ambos se combinan para generar el nuevo estado (fig. 7).

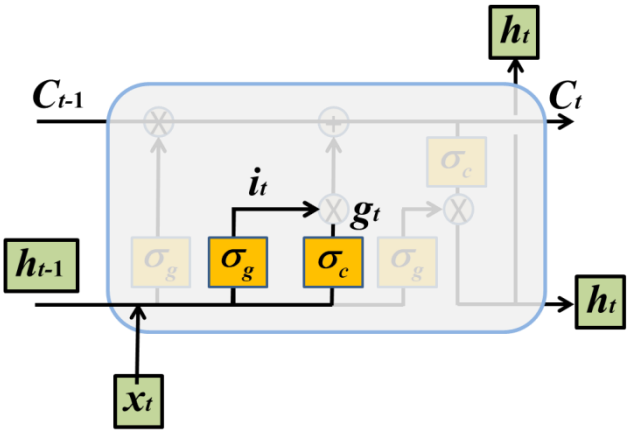


FIGURA 7- *INPUT GATE* Y CAPA DE NUEVOS VALORES

$$i_t = \sigma_g \left(U_{xi} x_t + W_{hi} h_{t-1} + b_i \right)$$
$$g_t = \sigma_c \left(U_{xc} x_t + W_{hc} h_{t-1} + b_c \right)$$

Ahora hay que actualizar el estado de la celda anterior, g_{t-1}, al nuevo estado de la celda g_t. Para ello se multiplica el estado antiguo por f_t, olvidando las cosas que decidimos olvidar antes. Luego se suma el término i_t e g_t (fig. 8), que son los nuevos valores candidatos, en función de cuánto se decide actualizar cada valor de estado.

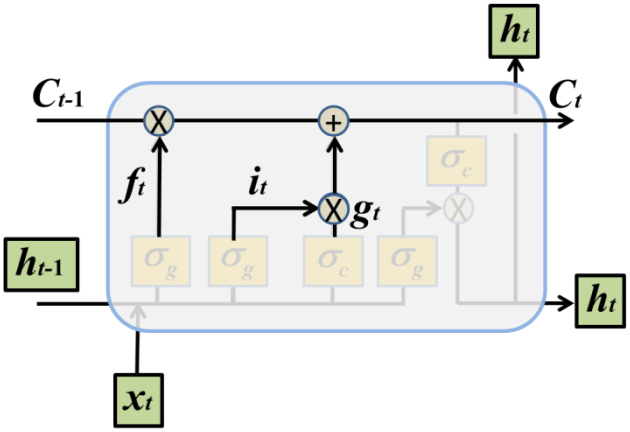


FIGURA 8- ACTUALIZACIÓN DEL ESTADO DE LA CELDA

$$C_t = f_t \text{ e } C_{t-1} + i_t \text{ e } g_t$$

Finalmente, es necesario obtener el resultado de la salida, que se basará en el estado de la celda, si bien, será una versión filtrada. Primero, se aplica la función sigmoidea 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 multiplica por la salida de la puerta de tipo sigmoidal (fig. 9), de modo que sólo contribuyen a la salida las partes seleccionadas.

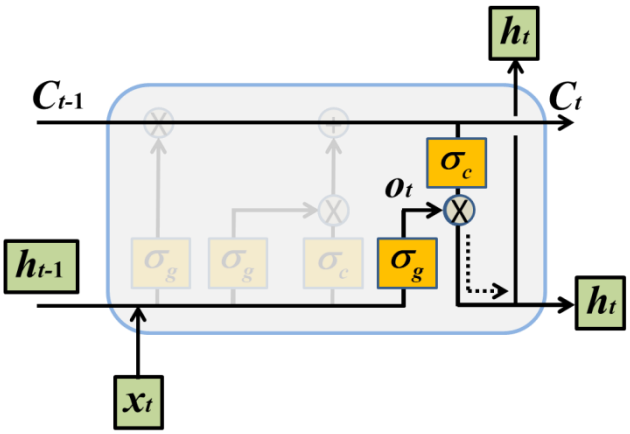


FIGURA 9- GENERACIÓN DE LA SALIDA

$$o_t = \sigma_g \left(U_{xo} x_t + W_{ho} h_{t-1} + b_o \right)$$
$$h_t = o_t \text{ e } \sigma_c \left(C_t \right)$$

3.2.2- CLASIFICACIÓN

Para realizar una clasificación cada secuencia de entrada tiene asociada una determina da categoría a la que pertenece. En el caso de nuestra aplicación concreta, En cada instante de tiempo *t* se puede definir un vector de entrada a la red que caracteriza las

aceleraciones, tal como: $\boldsymbol{x}_t \equiv (a_x, a_y, a_z)_t \in \Re^3$ de forma que a cada secuencia se le asocia una etiqueta que en realidad es una clase $\boldsymbol{y}_t \equiv c_i$, siendo i el número de clases definidas. De esta forma, el objetivo puede consistir en determinar el estado de movimiento de una persona equipada con un móvil según la actividad que esté desarrollando en cada instante, por ejemplo $c_1 \equiv$ sentada; $c_2 \equiv$ saltando; $c_3 \equiv$ corriendo o $c_4 \equiv$ caminando. En este caso se trataría de un sistema con cuatro clases y por tanto i es cuatro en este caso. El vector $\boldsymbol{x}_t$ puede definirse con los nueve valores, tres de orientación, tres de velocidad angular y otros tres de aceleración, de forma que en este caso $\boldsymbol{x}_t \in \Re^9$.

Esta es la idea expuesta en el trabajo expuesto en Drumond y col. (2018), donde los valores de orientación, velocidad angular y aceleración se obtienen mediante dispositivos del tipo IMU (*Inertial Measurement Units*), esto es, unidades de medición inercial.

4- DISEÑO DE LA APLICACIÓN

En el diagrama de la figura 10 se explica el funcionamiento de la aplicación de forma simplificada. El servidor aloja simultáneamente los datos que se utilizan para entrenar la red y el propio modelo entrenado de la misma. Un dispositivo móvil se conecta al servidor para mandar datos destinados al entrenamiento y para realizar el proceso de predicción.

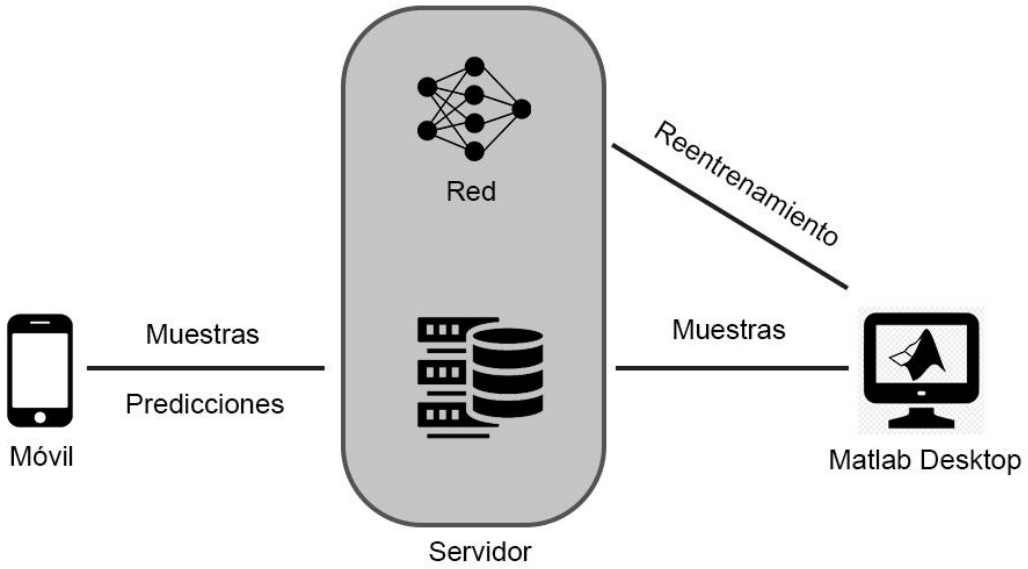


FIGURA 10- ESQUEMA DE LA ESTRUCTURA

Desde un ordenador con Matlab, se pueden solicitar estos datos para reentrenar la red siempre que sea necesario. Sin embargo la aplicación está diseñada para utilizar siempre el mismo modelo de red y no requerir de reentrenamientos constantes, debido a que la red debe ser compilada, junto con scripts necesarios para realizar las predicciones, en una librería que debe subirse al servidor, y este es el único proceso que no puede ser automatizado.

4.1- CAPTURA DE DATOS

4.1.1- MODELO

Se utilizan tres sensores: un acelerómetro, un giroscopio y un sensor de orientación, que registran aceleración lineal, aceleración angular y orientación del dispositivo, respectivamente. Estas medidas se toman en grupos de valores, que corresponden a estas magnitudes en cada uno de los tres ejes cartesianos, formando un total de nueve valores por cada medida. Que conforman la entrada de datos, tal y como se describe en el apartado 2.2.2.

El modelo de arquitectura diseñado es el que se muestra en la Figura 11. Consta de una unidad que proporciona la secuencia de entrada temporal, de forma que cada elemento de entrada $\boldsymbol{x}_t$, definido previamente es tridimensional, que contiene los correspondientes valores de aceleración en los tres ejes X,Y y Z. Son entradas proporcionadas por el sensor de captura de datos. La segunda estructura de tipo LSTM con 200 unidades o celdas de activación. Como se ha definido previamente (sección 2.2.1), cada una de estas celdas está caracterizada por los correspondientes matrices de pesos, U_{xf} y W_{hf} con las siguientes dimensiones 400×9 y 400×200 respectivamente, además del *bias* b_i de dimensión 400×1. A esta estructura le sigue una capa totalmente conectada (*FC*) con la correspondiente matriz de pesos de conexión de dimensión 3×200, donde la primera dimensión representa el número de acciones y por tanto las clases de la red. El modelo continúa con una capa de tipo *softmax*, con los valores correspondientes a cada clase conveniente transformados a través de la función de activación del mismo nombre y por tanto, correspondiéndose también con cada una de las acciones a clasificar: *walking*, *running* y *sitting*.

FIGURA 11- ESTRUCTURA DE LA RED

En la figura 12 se muestra la representación gráfica de una secuencia de entrada mostrando la evolución a lo largo del tiempo (eje X) y los datos de orientación y aceleración a izquierda y derecha respectivamente (eje Y).

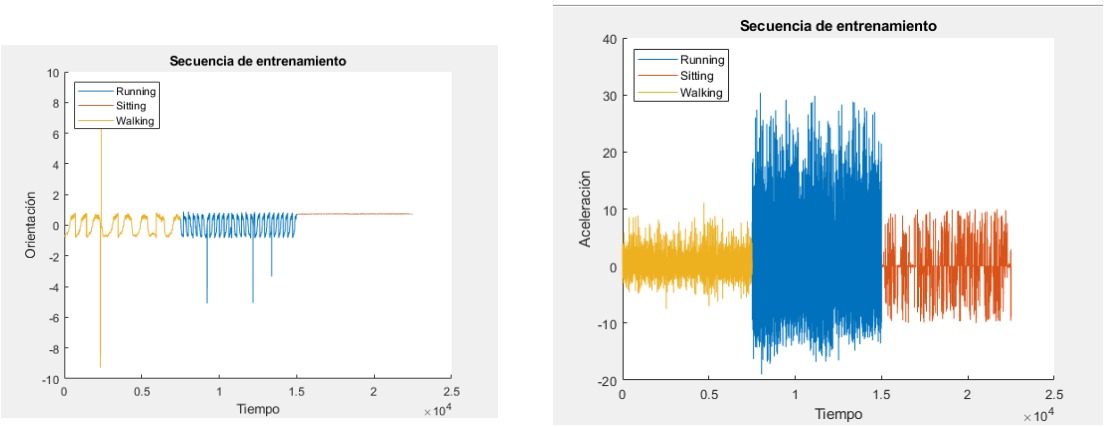


FIGURA 12- DATOS CAPTURADOS EN EL PLANO Z A LO LARGO DEL TIEMPO

4.1.2- PROCESO DE CAPTURA

Para mantener una frecuencia en la captura de los datos, esta se realiza de forma asíncrona a las lecturas de los sensores, que se actualizan cada vez que se detecta un cambio. Esto significa que se debe emplear una frecuencia de captura que permita mantener la precisión de los mismos, pero no sea tan alta que se capturen datos antes de que los sensores hayan tenido tiempo de actualizar sus lecturas, mandando datos repetidos. Cabe destacar que los sensores son muy sensibles y detectaran cambios incluso en estado de reposo.

Se toman como referencia repositorios de dominio público que han capturado datos utilizando métodos similares a los propuestos en este proyecto (Anguita y col., 2013), los cuales emplean una frecuencia de 50 capturas por segundo con buenos resultados. La frecuencia de actualización de los sensores es mucho mayor, de modo que se puede emplear esta misma frecuencia de captura.

Junto con los datos capturados por los sensores, se envía una secuencia de caracteres que describe la actividad sobre la que se están capturando datos; y un identificador único para poder diferenciar secuencias de datos distintas. Este identificador se genera al inicio de una sesión de captura.

4.1.3- PREPROCESAMIENTO DE DATOS

Antes de enviar los datos al servidor, han de ser convenientemente pre-procesados y preparados. Se realizan dos tipos principales de pre-procesamiento:

FILTRADO DE RUIDO:

Los sensores de aceleración (lineal y angular) son especialmente ruidosos. Este tipo de ruido consiste en capturas que tienen valores exageradamente altos con respecto al resto de datos. El método más efectivo para eliminar el ruido, es realizar varias capturas en un periodo corto de tiempo, y hacer la media entre todas ellas. Es decir, si se envía un total de 60 datos por segundo, y los sensores pueden realizar 120 capturas por segundo, se pueden realizar dos capturas y hacer la media por cada valor del sensor que enviamos al servidor. En la práctica, la aplicación puede hacer la media de entre 100 y 180 capturas, dependiendo de la frecuencia con la que detecta cambios, lo cual asegura una buena precisión.

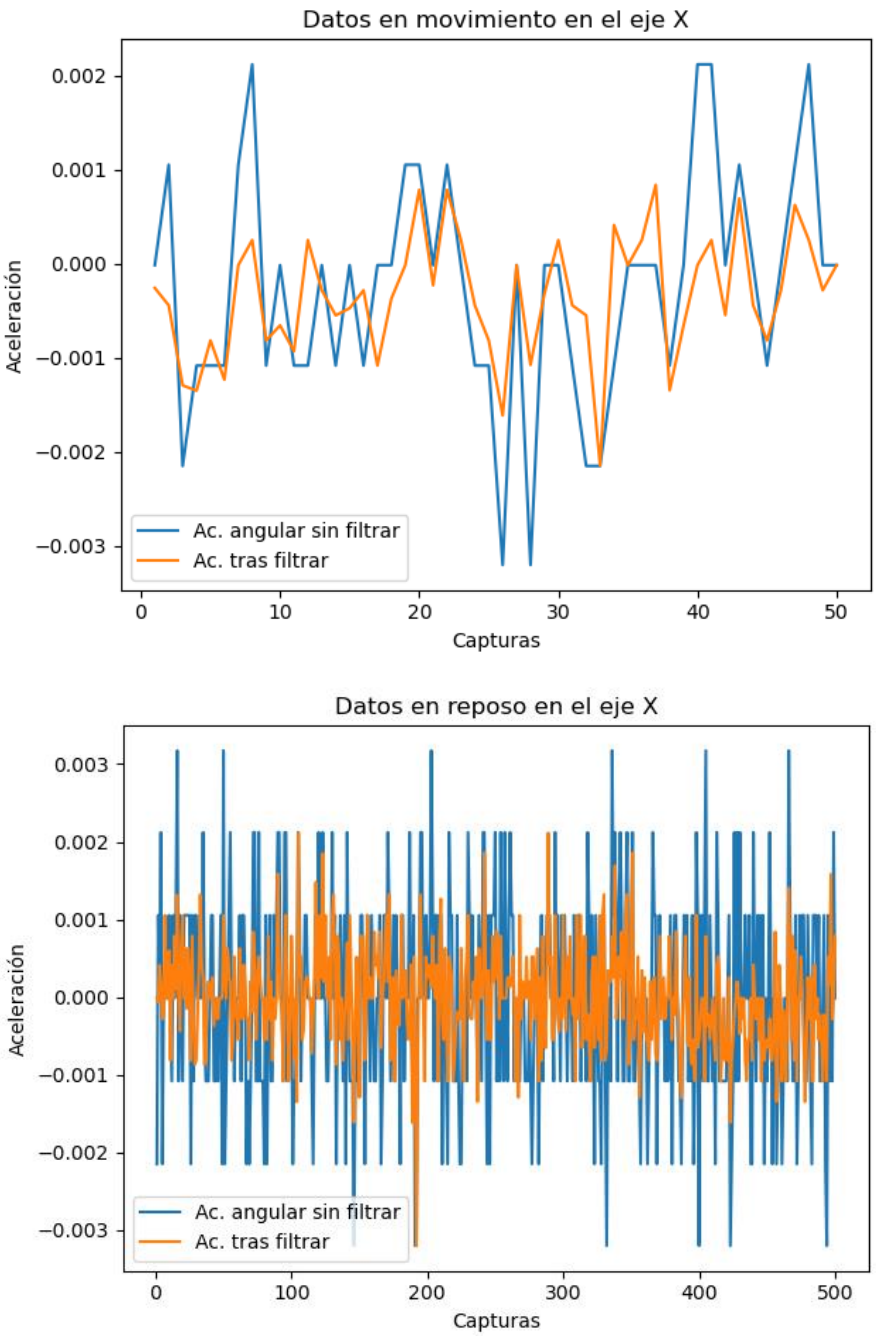


FIGURA 13- SUPERPOSICIÓN DE LOS DATOS DE ACELERACIÓN ANGULAR DURANTE 1 Y 10 SEGUNDOS

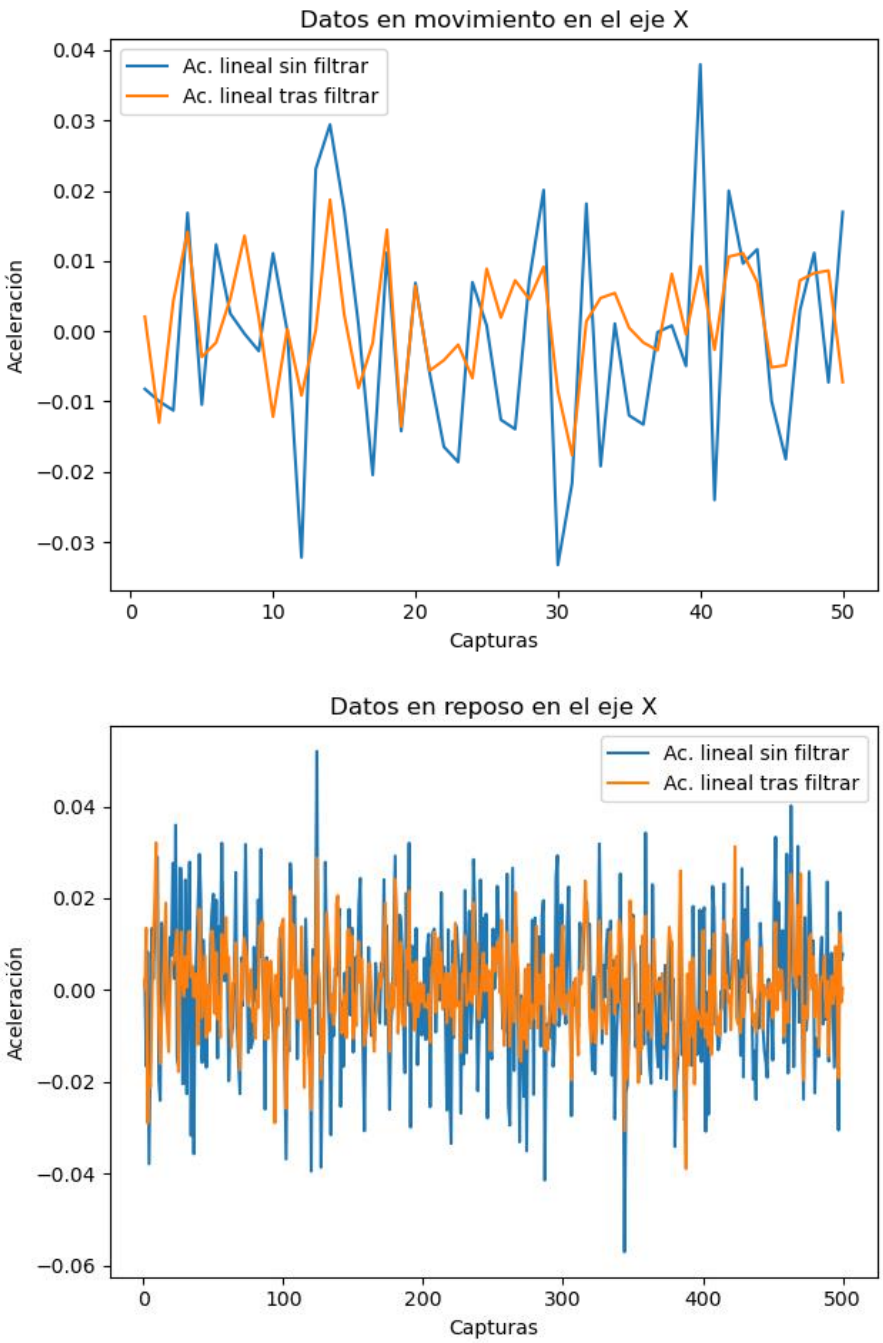


FIGURA 14- SUPERPOSICIÓN DE LOS DATOS DE ACELERACIÓN LINEAL DURANTE 1 Y 10 SEGUNDOS

En las figuras 13 y 14 se pueden ver comparaciones de los datos antes y después de ser procesados. Estos datos se corresponden a los recogidos en el eje x en una situación de reposo, con lo que idealmente deberían registrar un valor de 0. En las figuras situadas en la parte superior se pueden ver los datos capturados durante 1 segundo mientras que en las situadas en la parte inferior se muestran los capturados a lo largo de 10 segundos. En estas figuras se puede apreciar también la diferencia de precisión de los distintos sensores, siendo que los datos de aceleración lineal registran ruidos un orden de magnitud mayores que los de aceleración angular.

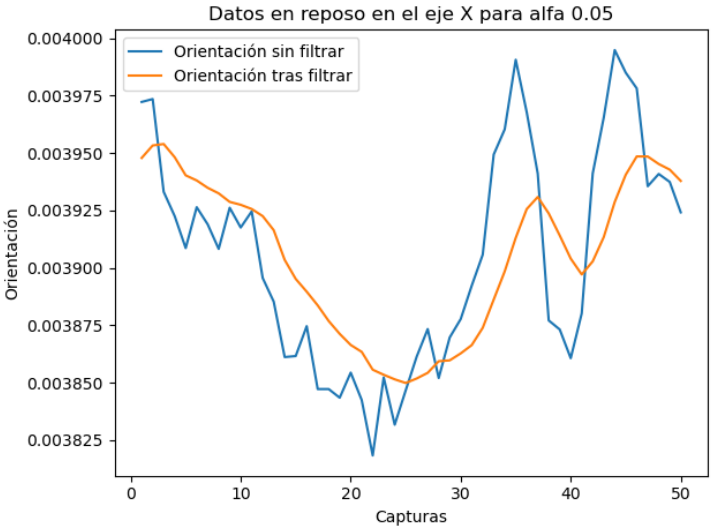
DEDUCCIÓN DE LA ORIENTACIÓN

Para calcular la orientación, se utiliza el giroscopio interno del teléfono, que devuelve la posición del dispositivo en radianes, con respecto a los ejes cartesianos. Aunque este es un sensor mucho menos propenso a capturar ruido, tiene sus propios problemas.

Por la propia naturaleza de los movimientos que captura, se trata de un proceso mucho más discreto y menos continuo que medir la aceleración. Mientras podemos suponer que la aceleración se mantiene constante en una cantidad lo suficientemente pequeña de tiempo, lo mismo no se cumple para la orientación, que puede dar cambios bruscos y cortos en un instante. De modo que para evitar ruidos en la rotación, la aplicación implementa un filtro de paso bajo que bloquea valores que se disparan de forma exagerada, manteniendo intactos los valores que varían poco. El filtro se ajusta al siguiente modelo de filtro digital:

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1}$$

Donde α tiene un valor apropiado para no suavizar en exceso la señal, pero descartando valores que no resulten realistas. Este valor suele ser entre 0.1 y 0.2 y se llama “factor de suavizado”. Realizando pruebas se puede encontrar un valor de este factor apropiado.



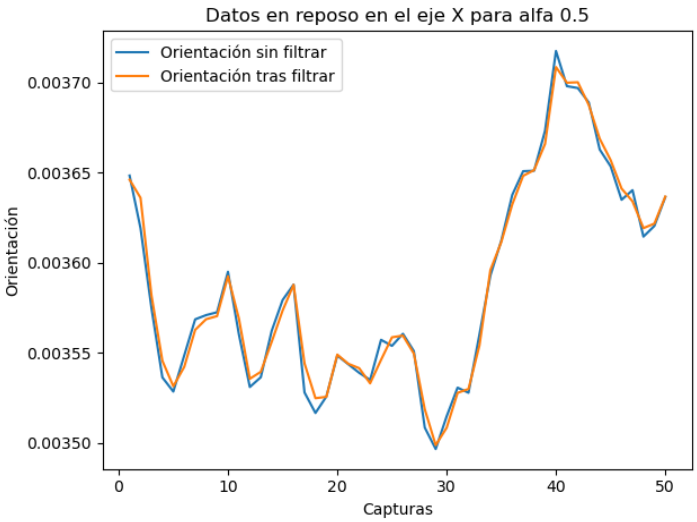
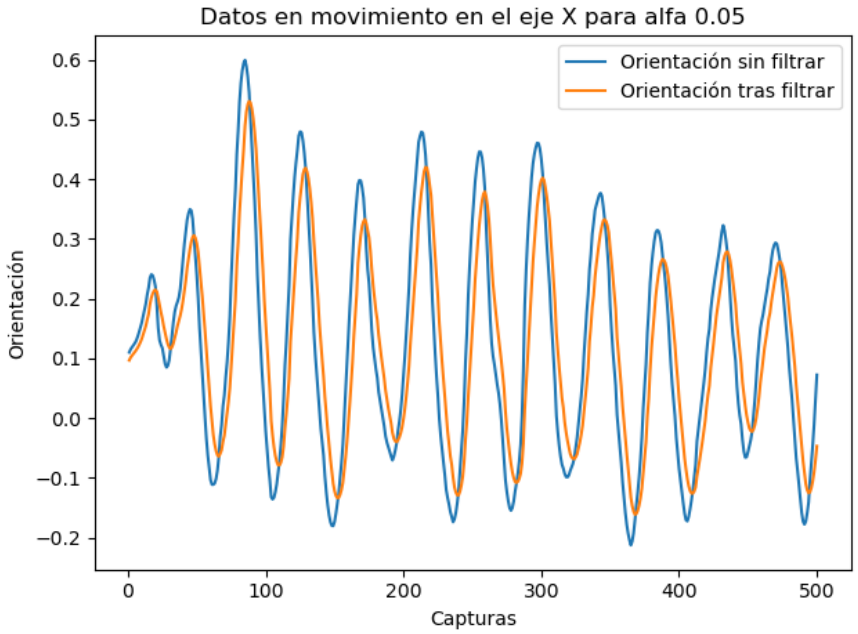


FIGURA 15- DATOS EN REPOSO DE ORIENTACIÓN A LO LARGO DE UN SEGUNDO PARA DISTINTOS VALORES DE ALFA

En las gráficas anteriores se puede comprobar como para valores cada vez más altos de α la muestra filtrada se acerca más a la señal sin filtrar. Se puede comprobar también como en comparación con los sensores de aceleración, el de orientación es mucho más estable. En la aplicación se utiliza un valor de α de 0.2, que es suficiente para deshacerse del ruido.



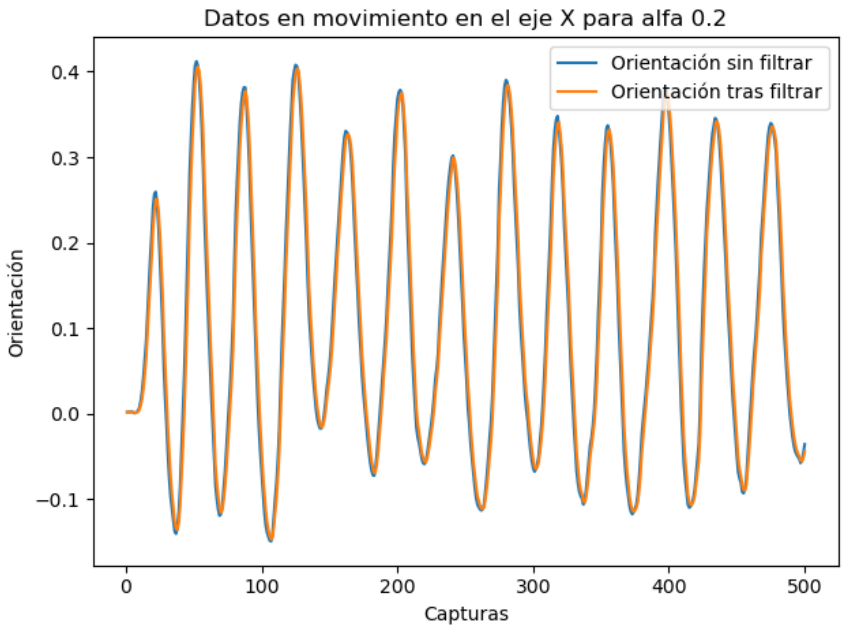


FIGURA 16- DATOS DE ORIENTACIÓN EN MOVIMIENTO A LO LARGO DE 10 SEGUNDOS PARA DISTINTOS VALORES DE ALFA

En la figura 16 se muestra como los datos pueden verse deformados por un valor de α inadecuado. Si bien un valor demasiado alto fallaría a la hora de deshacerse del ruido, uno demasiado bajo filtraría datos no ruidosos, comprometiendo la precisión del sensor.

4.1.4- ESTRUCTURAS DE DATOS EMPLEADAS

Cuando los datos están listos para ser enviados al servidor, son convertidos a cadenas caracteres de longitud constante para que el tamaño de los datos no varíe demasiado al transformarse a formato json. Una vez en el servidor, las muestras son transformadas de nuevo a vectores de numpy, lo que asegura tiempos óptimos de almacenamiento en la base de datos.

Dado que la red neuronal está implementada de tal forma que se ejecuta en un entorno virtual de Matlab que el servidor crea en tiempo de ejecución, los datos deben transformarse por última vez a un formato legible por Matlab. Para ello del vector *numpy* se extraen primero los datos a una lista simple, se castean a tipo *float* y se almacenan en una estructura en lista de formato correcto, utilizando la librería Matlab de Python.

4.2- ARQUITECTURA

A lo largo del desarrollo del proyecto, se han implementado tres modelos distintos con arquitecturas diferentes:

4.2.1- MATLAB Y MATLAB DRIVE

La primera idea fue utilizar la funcion de Matlab Mobile (Matlab, https://es.mathworks.com/products/matlab-mobile.html, s.f.) para transmitir datos desde la aplicación para Android de Matlab a una red ejecutándose en un ordenador. Sin

embargo, a pesar de que se podían mandar datos a través de Matlab Drive por medio de ficheros, la funcionalidad para lectura de datos directamente de los sensores que anteriormente existía fue desactivada en la última versión y sus servidores ya no soportan esta opción. De modo que si bien se puede entrenar una red de esta manera, y realizar predicciones sobre datos ya recogidos, la predicción a tiempo real no es posible. Para sortear este problema, se desarrolló una aplicación para Android que fuera capaz de recoger datos de los sensores y mandarlos a un servidor remoto. Para el envío de datos, se decidió utilizar Thingspeak (Thingspeak, s.f.)

4.2.2- ANDROID, MATLAB Y THINGSPEAK

En esta segunda iteración, se desarrolló una aplicación Android capaz de realizar lecturas de los sensores del móvil y mandarlos a un servidor remoto, para lo cual se empleó Thingspeak.

Thingspeak es una plataforma de analítica para IOT que permite visualizar y analizar datos online. También tiene la habilidad de ejecutar código Matlab desde la propia plataforma, lo que permitiría realizar operaciones con el modelo de red desde la nube.

Thingspeak guarda los datos en distintos canales. La forma en la que funcionan estos canales es a través de una serie de campos, que se escriben cada vez que llega un mensaje. Estos mensajes se mandan utilizando protocolo MQTT, que resulta muy rápido y ligero, con bajo coste para la batería del dispositivo móvil que los envía. Thingspeak está perfectamente integrado con Matlab, y permite recoger desde un ordenador datos de estos canales en tiempo real para realizar predicciones.

Esta opción permitiría además subir el modelo de la red re-entrenado (alojada en un fichero .mat) al propio canal de Thingspeak, donde se podrían realizar las predicciones sin tener que descargar de nuevo los datos en otro lugar.

Aunque esta opción es muy buena y sería la más apropiada en circunstancias normales, Thingspeak es un servicio con licencia. Las características del servicio que ofrece la versión gratuita son insuficientes para que el sistema funcione de forma correcta, ya que limita la frecuencia con la que se pueden enviar mensajes, e impone un tamaño máximo muy restrictivo sobre los datos que se pueden enviar. Por desgracia estas limitaciones han probado ser un serio inconveniente, la red no puede funcionar bajo las mismas, de modo que la siguiente opción fue desarrollar un servidor propio que permitiera recibir estos datos sin ningún tipo de limitación.

4.2.3- ANDROID Y DJANGO

En el tercer modelo, el planteamiento que se formula consiste en disponer de un servidor propio que almacene, procese y envíe los datos necesarios. Con esto, todas las partes de la arquitectura pasan a ser de implementación propia, lo que permite un control total de la funcionalidad y la predicción en tiempo real es finalmente posible.

El servidor se implementa como una rest API de Django utilizando Python; y la red neuronal, desarrollada y entrenada en Matlab, está integrada en el propio servidor en un paquete compilado, lo cual agiliza mucho el proceso.

4.3- IMPLEMENTACIÓN

En esta sección se detallará el funcionamiento interno de los diversos módulos que componen el tercer modelo implementado. Se ha dividido en tres elementos principales: la red neuronal, la aplicación de móvil y el servidor.

La red neuronal es entrenada en un equipo utilizando Matlab, y después debe ser compilada para ser alojada en el servidor. La aplicación de móvil realiza peticiones al servidor donde se envían los datos que la red utiliza para realizar sus predicciones. De la misma forma, envía datos de muestra para ser alojados en la base de datos, de donde pueden ser requeridos en caso de querer reentrenar la red.

4.3.1- RED NEURONAL

La red se ha implementado utilizando el toolbox "Machine Learning and Deep Learning" de Matlab, que contiene herramientas que permiten realizar diseños básicos de redes. El punto de partida es un ejemplo de clasificación utilizando redes LSTM de Matlab: "Sequence-to-Sequence Classification Using Deep Learning" (Matlab, s.f.)

Se han implementado dos modelos de red distintos

- El primero conste de una única red con una capa de entrada de 9 unidades, siendo 3 unidades destinadas a los tres ejes cartesianos detectados por cada sensor; 200 unidades ocultas y 3 unidades en la capa de salida. En la tabla 1 se puede ver una descripción en detalle de cada capa.

	Unidades	**Activaciones**	**Parámetros a aprender**	**Estados**
Secuencia entrada	**9**		**0**	
LSTM	200	Pesos Entrada 800×200 Pesos Recurrentes 800×200 Bias 800×1	168000	Estados ocultos 200×1 Estados de las celdas 200×1
Totalmente conectada (fc)	3	Pesos 3×200 Bias 3×1	603	
Softmax	3		0	
Salida			0	

TABLA 1- ESTRUCTURA DE LA RED PRINCIPAL

- El segundo consiste en 3 redes, con 3 unidades en sus capas de entrada, equivalentes a cada uno de los sensores. La idea es equiparar y comparar los resultados de cada red y estimar sus grados de confianza, para tomar una decisión final. En la tabla 2 se puede ver una descripción en detalle de cada capa.

	Unidades	Activaciones	Parámetros a aprender	Estados
Secuencia entrada	3		0	
LSTM	200	Pesos Entrada 800×200 Pesos Recurrentes 800×200 Bias 800×1	168000	Estados ocultos 200×1 Estados de las celdas 200×1
Totalmente conectada (fc)	3	Pesos 3×200 Bias 3×1	603	
Softmax	3		0	
Salida			0	

TABLA 2- ESTRUCTURA DE CADA UNA DE LAS REDES PEQUEÑAS

ENTRENAMIENTO

El entrenamiento de la red se realiza utilizando un script de Matlab, que realiza las siguientes acciones:

1. Realiza una petición a la API, que le devuelve todas las capturas de datos obtenidas, junto con una etiqueta que indica a qué actividad corresponden
2. Descarta los datos etiquetados con la actividad "Testing", estos datos se utilizan para desarrollo y testing de las herramientas y no se deben incluir en el proceso de entrenamiento.
3. Se convierte la estructura de datos en un cell array de Matlab, que contiene a su vez estructuras con las capturas de los sensores, indexadas por las etiquetas de actividades.
4. Se recorre cada una de las etiquetas y se convierten las estructuras con las capturas de datos a un tipo de dato numérico con el que puede trabajar la red.
5. Se descartan las 100 primeras y últimas capturas, que se corresponden a los primeros y últimos dos segundos del proceso de captura. De esta forma se ignora el momento en el que se pulsa el botón y se introduce el móvil en el bolsillo justo antes de empezar a capturar los datos, y el de sacar el móvil del bolsillo para interrumpir el envío de datos.
6. Se separa los datos de manera que el 75% se destinará a entrenar la red, y el 25% restante se utilizará para realizar pruebas
7. Se crea el modelo de red neuronal, adaptando las unidades en las capas de entrada y salida al tamaño de los datos de entrada y el número de actividades respectivamente, y se entrena la red obtenida con los datos.
8. Se comprueba el porcentaje de acierto sobre los datos de prueba

Este script genera una red variable, con distintas neuronas en las capas de entrada y salida dependiendo de las dimensiones de los datos obtenidos. Esto garantiza máxima simplicidad a la hora de entrenar nuevas redes, ya que si se añaden al servidor nuevas muestras de nuevas actividades, lo único que hay que hacer es relanzar este script y se generará y entrenará una nueva red con la nueva topología.

4.3.2- APLICACIÓN MÓVIL

La figura 17 muestra la estructura de la aplicación, con sus correspondientes módulos. En la parte central, se pueden ver los dos procesos de la aplicación, uno que actualiza la interfaz y otro que registra los cambios. El *looper* de Android es el gestor de procesos subyacente que emplean todas las aplicaciones de Android. Éste itera sobre estos procesos y los ejecuta siguiendo cierta prioridad, por lo que la aplicación utiliza *handlers* para asegurarse de que el tiempo que transcurre entre cada ejecución del hilo

es siempre constante. De esta forma no se compromete la frecuencia con la que se capturan los datos y la integridad de los mismos.

En la parte izquierda se muestra el esquema que sigue la recogida de datos: cuando el *sensor manager* detecta un cambio en los sensores, registra ese cambio en sus valores internos. Cada vez que se lanza la actividad del proceso de captura, se capturan esos datos almacenados en el *sensor manager*, se pasan por un filtro de paso bajo (ver sección 4.1.3) y se almacenan en variables internas. Asíncronamente, el proceso que actualiza la interfaz consulta los últimos valores guardados en estas variables internas y los muestra por pantalla cada cierto tiempo. Una vez que se ha realizado un número predeterminado de capturas, se vuelven a procesar para eliminar ruido y se envían al servidor, tras lo que se reinician las variables internas.

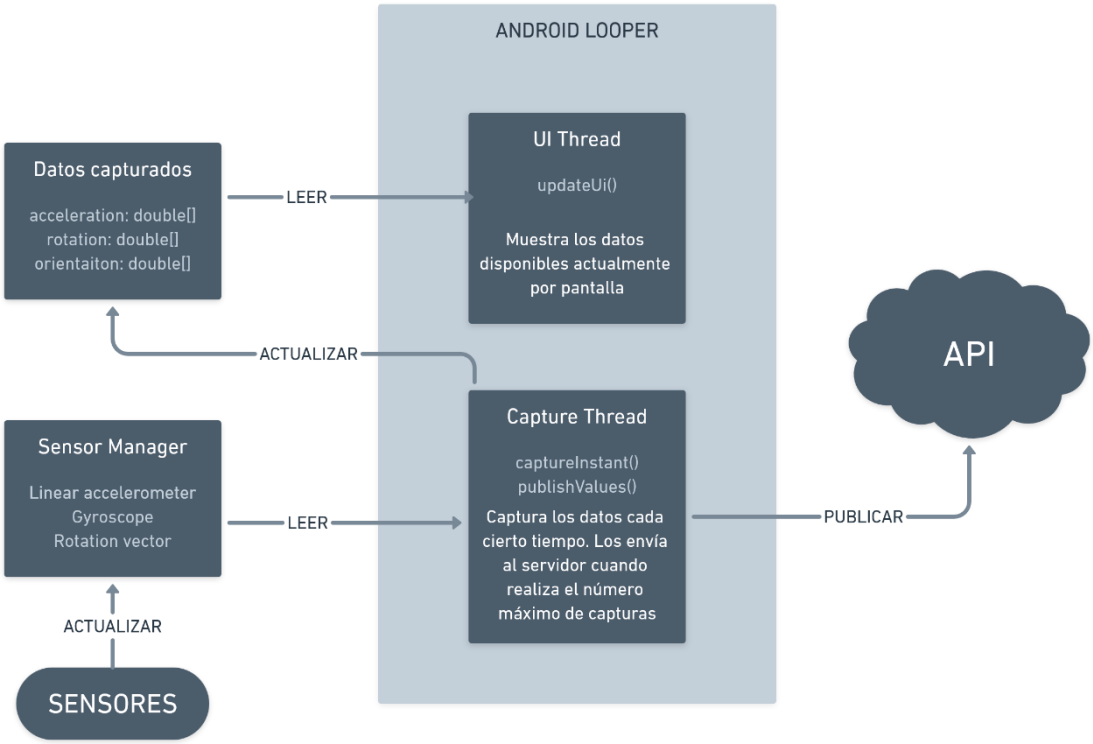


FIGURA 17- ESTRUCTURA DE LA APLICACIÓN

DISEÑO

La aplicación implementa dos funcionalidades diferentes pero que utilizan el mismo sistema de captura de datos. La primera funcionalidad está pensada para el desarrollo y es la de captura y envío de datos al servidor, donde se almacenan para un posterior entrenamiento de la red. La segunda es la de predicción en tiempo real, la cual sería la funcionalidad que utilizaría un usuario final.

En la figura 18 se ven las dos pantallas, de envío de datos para captura y de envío de datos para realizar predicciones. En ambas pantallas se muestra en tiempo real los datos que están capturando los sensores en la parte superior, y la dirección donde se enviarán los datos en la parte central. En la parte inferior de la pantalla se puede ver un texto que indica el estado de conexión con el servidor.

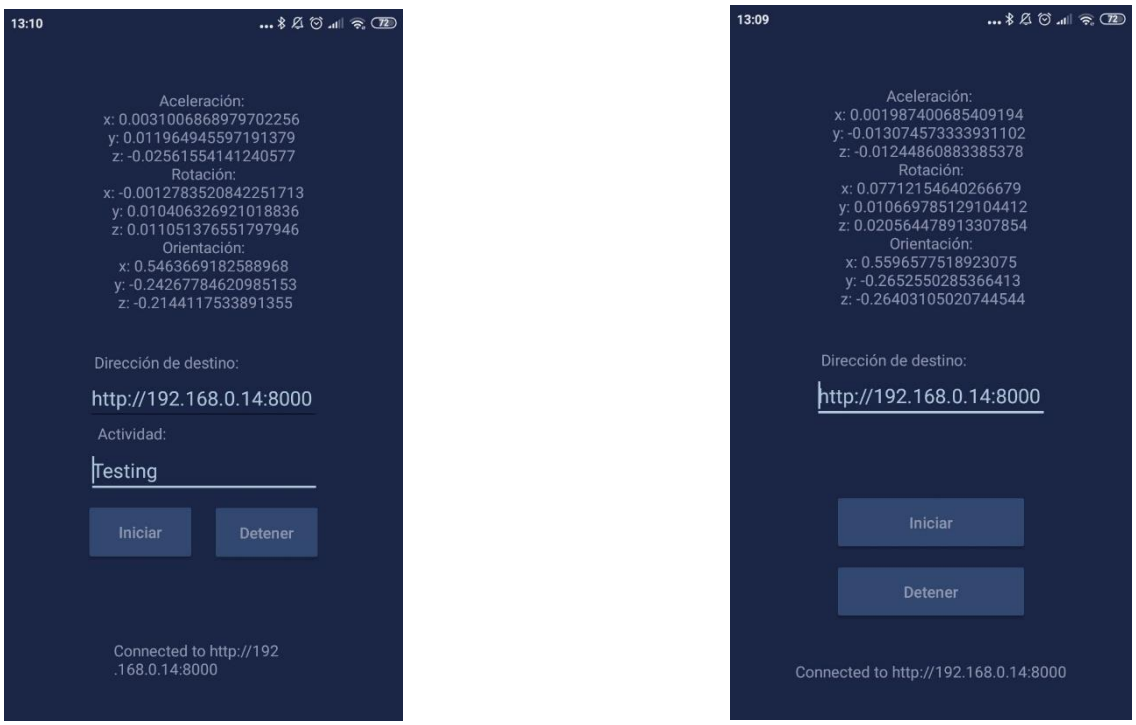


FIGURA 18- PANTALLA DE CAPTURA DE MUESTRAS (IZDA) Y PANTALLA DE ENVÍO DE DATOS

La aplicación está diseñada para funcionar en segundo plano, minimizando el uso de recursos. Para ello emplea una hebra que maneja los sensores separada de la aplicación principal. Cada vez que se detecta un cambio en estos, se actualizan los valores actuales de los mismos. Simultáneamente, cada cierto tiempo se va a realizar una captura de los datos tal y como existen en un momento concreto. Estos dos procesos son asíncronos el uno del otro, garantizando que la frecuencia de envío de datos se mantenga estable, lo que es esencial para el correcto entrenamiento de la red.

Tanto en el proceso de captura como en el de predicción, se envían paquetes con varias capturas en cada llamada al servidor. En el proceso de predicción, enviar una cantidad grande de capturas aumenta la precisión de la red, a cambio de una frecuencia de actualización menor.

TECNOLOGÍA

Se ha implementado en java utilizando Android Studio, el principal IDE para aplicaciones en Android.

Los datos pueden ser enviados a un canal de Thingspeak utilizando un cliente de MQTT, para lo cual se emplea una librería de java llamada Paho.

Para el envío de datos al servidor por HTTP se utiliza Volley, una librería que optimiza el proceso de peticiones y respuestas y facilita mucho el proceso de desarrollo. A la librería se delegan las peticiones de forma asíncrona. Volley además garantiza una secuencialidad correcta del envío de datos, lo cual es esencial ya que el orden correcto de almacenamiento de las muestras depende de que el orden de envío sea también correcto.

4.3.3- SERVIDOR

En la figura 19 se muestra la estructura del servidor. Teniendo en cuenta el esquema de Django, éste está diseñado en base al estilo de arquitectura Modelo Vista Controlador. Los modelos se corresponden con las dos funcionalidades del servidor. Uno para los datos almacenados en la base de datos para entrenamiento y otro para las predicciones que se están realizando.

Las vistas implementan principalmente también estas dos funcionalidades, dos vistas permiten insertar y extraer datos para entrenamientos y otra solicita que se realice una predicción en base a unos determinados datos de entrada.

El servidor está desarrollado para ser concurrente, de modo que en un futuro varias aplicaciones móviles pudieran hacer uso de un mismo servidor. Para ello, cada solicitud de predicción vendría acompañada de un identificador único para el dispositivo, que sería asignado por el propio servidor en el momento de su primera conexión.

Al acceder al servidor desde una URL se cargaría una página donde se puede ver el último valor introducido en el modelo de *Actividad actual*. Nuevamente, en caso de introducir concurrencia en el servidor, sería necesario implementar un sistema de usuarios que permita identificarse en la página para acceder a las predicciones realizadas exclusivamente sobre el dispositivo del usuario en cuestión.

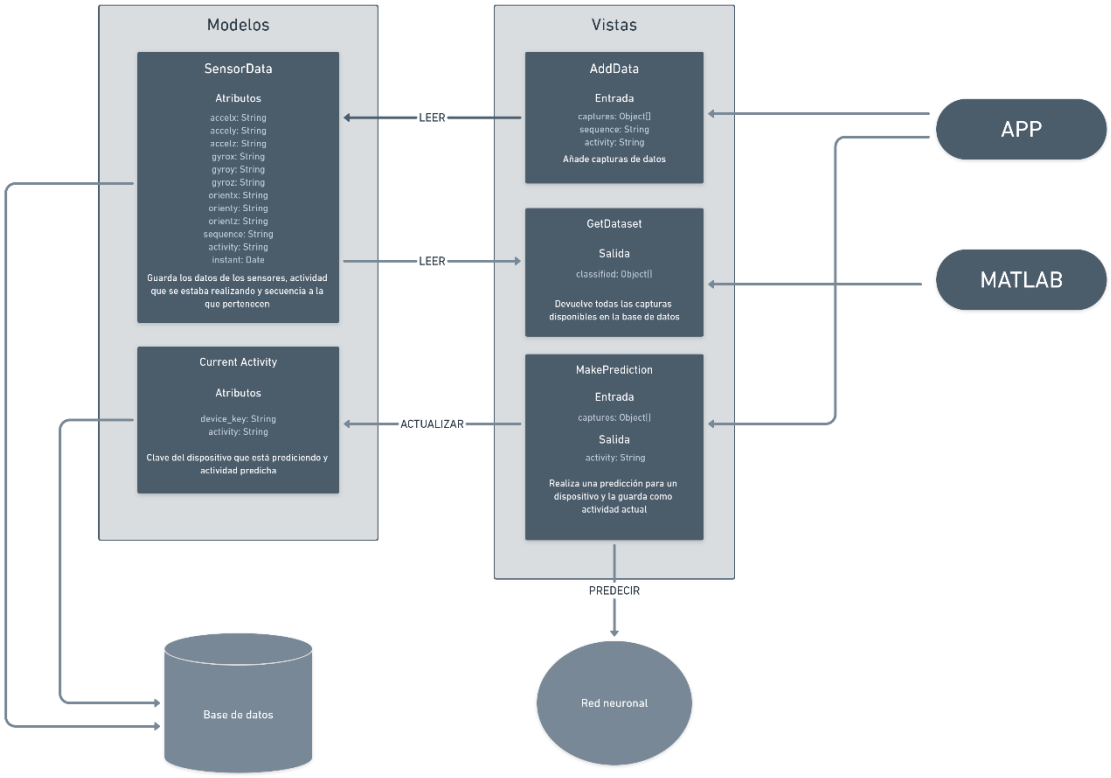


FIGURA 19- ESTRUCTURA DE LA API

DISEÑO

El servidor consiste en una API REST que implementa tres funcionalidades principales:

- Recibe volúmenes grandes de capturas de datos con una clase asociada. Estos datos se guardan en la base de datos con su clase correspondiente, con el propósito de ser accedidos en cualquier momento para servir como datos para entrenar la red.
- Recibe capturas de datos en tiempo real, y utiliza la red para generar predicciones y devolverlas al dispositivo. Estas predicciones se guardan también en la sesión del servidor, de modo que son accesibles tanto por el usuario del dispositivo como por cualquiera con acceso al servidor. El usuario puede consultar la actividad que se está realizando desde una página web.
  La red neuronal se ejecuta desde un script de Matlab compilado como un paquete de Python. Para ejecutar este script, se instancia un servidor de Matlab al iniciarse el servidor Django, y el propio script hace el manejo de los datos nativos de Matlab y devuelve una cadena de caracteres con la clase.
- Recibe peticiones de datos para entrenamiento de la red. En ese caso devuelve una estructura de datos con todas las capturas realizadas, ordenadas en el mismo orden en el que el servidor las recibió, junto con información sobre la clase a la que pertenecen.

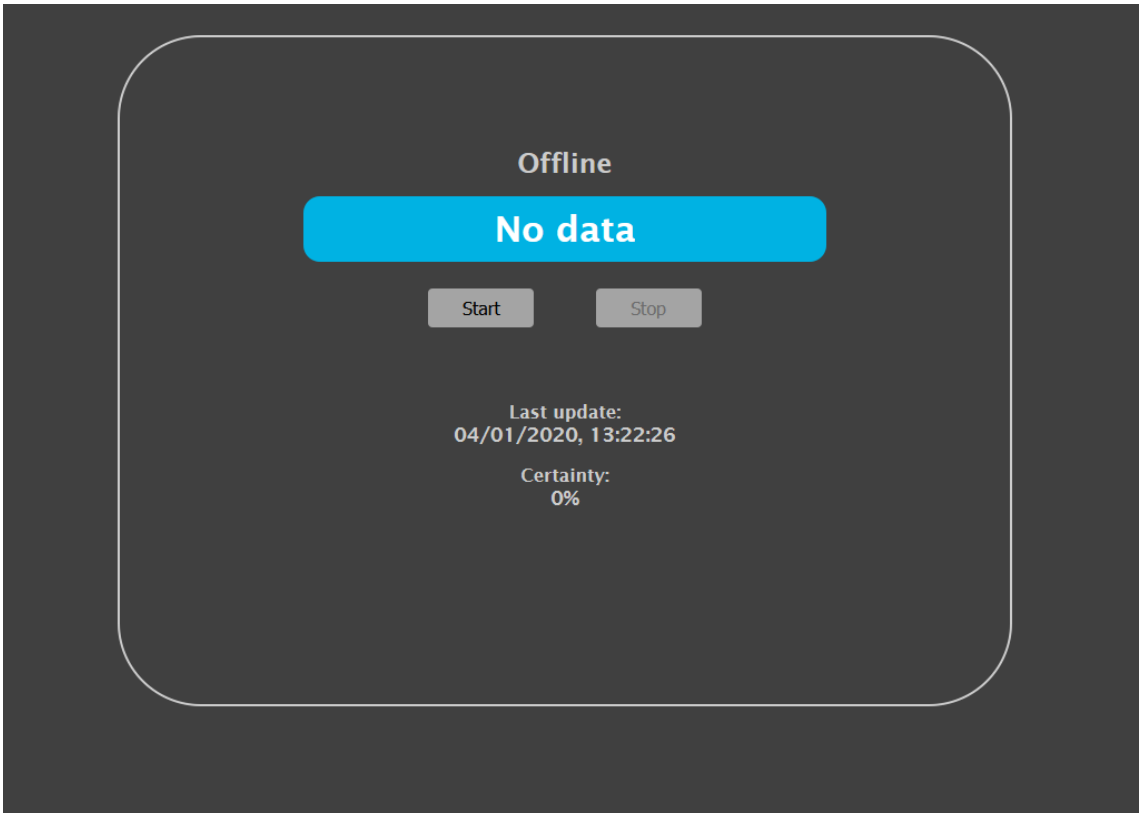


FIGURA 20- PÁGINA WEB DE LA APLICACIÓN

TECNOLOGÍA

La implementación se ha realizado en Python utilizando Django como *framework*. Se utiliza el *toolkit* "Django REST framework" para implementar la funcionalidad de API REST. La ventaja de utilizar Django es la facilidad para generar una aplicación funcional en poco tiempo, que sigue siendo lo suficiente flexible como para adaptarse a cualquier funcionalidad que se necesite implementar en el servidor.

Python es un lenguaje ideal para el manejo de grandes volúmenes de datos, gracias a sus numerosas librerías como *numpy* que permiten añadir cualquier funcionalidad de transformación o análisis que sea necesaria. Todas las gráficas presentes en esta memoria han sido generadas utilizando matplot.

También resulta ideal de *python* la facilidad para añadir librerías externas a la aplicación. El modelo de red neuronal entrenado se integra en el servidor mediante una librería previamente compilada. El entorno virtual de Matlab que utiliza la aplicación para ejecutar scripts de Matlab también se integra en el servidor gracias a una librería externa de Matlab.

La persistencia de los datos se lleva a cabo en una base de datos *sqlite*. Esta es una opción muy ligera, portable, ideal para el uso que se le da al servidor, ya que el formato de datos que almacenamos en la base de datos es muy ligero y permite realizar operaciones con rapidez.

4.4- ALOJAMIENTO DEL CÓDIGO

Tanto para el alojamiento como para el control de versiones del código fuente de las aplicaciones se ha utilizado GitHub. El código se encuentra disponible en los siguientes repositorios:

- Servidor: https://github.com/Rpavon297/HumanActivityRecognition
- Aplicación de móvil: https://github.com/Rpavon297/HumanActivityRecognition

Los scripts empleados para entrenar la red neuronal y realizar pruebas sobre la misma se pueden encontrar en el siguiente enlace de Google Drive: https://drive.google.com/open?id=1nv_Lp0gMYVlXhNddpGQwiO4INjRsywfj

5- RESULTADOS

5.1- RECURSOS UTILIZADOS

Se han utilizado dos piezas de equipo para llevar a cabo las pruebas que se detallan a continuación: un ordenador y un *smartphone*. Las comunicaciones entre la aplicación móvil y el servidor se han realizado a través de una red local vía *WiFi*, simulando las condiciones en las que se comunicarían en un entorno real, en el que el servidor estaría alojado en la nube y la comunicación sería vía internet.

5.1.1- ORDENADOR

Se ha empleado un ordenador de sobremesa con SO Windows 10 para alojar el servidor y realizar los entrenamientos de la red. Utiliza una CPU Intel Core de 3.70GHz y 16 GB de memoria RAM. Para el entrenamiento de la red se utiliza una GPU de 640 cores CUDA, 2048 MB de memoria y 5,4 Gbps de velocidad.

El servidor se ejecuta en un entorno virtual utilizando Python 3.7 como intérprete.

5.1.2- SMARTPHONE

La aplicación de móvil se ejecuta en un dispositivo con sistema operativo Android 7.1.2 modelo Xiaomi Redmi 4X, con CPU octa-core de 1.40 GHz y 3,00 GB de RAM.

5.2- RECOGIDA DE DATOS

Para tomar las muestras, se han realizado las actividades durante 30 minutos, con el móvil en el bolsillo derecho, de pie y con la pantalla mirando hacia afuera. Para mayor consistencia de los datos, se han recogido durante la mayor cantidad de tiempo ininterrumpida posible. De todos los datos recogidos, un 70% se han utilizado para entrenar la red y un 30% para comprobar el funcionamiento.

5.3- PROCESO DE ENTRENAMIENTO

Todos los procesos necesarios para el entrenamiento de la red se han automatizado en un script de Matlab. El script descarga los datos, los ordena de forma correcta y los reagrupa en el formato que requiere la red. Después separa el 75% de la muestra que se destinará al entrenamiento y con ello entrena las cuatro redes que se utilizan en la aplicación.

Se realizan cuatro entrenamientos diferentes. En el primero se entrena el modelo de red con 9 unidades de entrada y en el resto se entrenan las redes de 3 unidades de entrada.

En la figura 21 se muestra la evolución del proceso de entrenamiento. En el eje horizontal de dicha figura se representa el número de iteraciones del proceso, habiendo definido un máximo número de iteraciones, en este caso 60, llegando a alcanzarse dicho valor. En estos modelos se definen lo que se denominan *epochs* que representan el paso de todas las muestras disponibles para el entrenamiento, que por otra parte éstas se dividen en lotes, donde cada lote se denomina técnicamente *batch*, y el conjunto de muestras contenidas en un lote cuando pasan por el modelo constituyen una iteración. Para el caso del proceso mostrado en la mencionada figura, se ha definido una iteración por *epoch*, pasando en cada iteración todas las muestras disponibles en el entrenamiento, lo que hace un total de 60 iteraciones con las 60 *epochs* definidas. El proceso total se completa en 13 minutos y 18 segundos.

No se ha especificado el parámetro *patience*. Cuando se fija este valor lo que realmente se indica es que el proceso se detenga después de que tras el número especificado de *epochs* no se consigan mejoras significativas o relevantes en la precisión. Por otra parte, la razón de aprendizaje se establece constante para todo el proceso de entrenamiento con un valor de 0.001.

En la parte superior de la figura 21, aparece la representación de la precisión (*accuracy*) del proceso, consistente en determinar el grado de ajuste de la red en función de los datos de validación, que contienen las acciones esperadas en relación a los datos de entrenamiento, de suerte que cuanto mayor es el número de aciertos, en términos de porcentaje, mayor es la precisión. Con el modelo propuesto y para los datos utilizados se consigue una precisión de validación final del 100%.

En la parte inferior de la figura 21 se muestra la evolución del ajuste de la función de pérdida (*loss*). Esta función se establece como objetivo a minimizar durante el proceso de entrenamiento, de forma que valores mínimos representan los mejores ajustes, dado que el mínimo absoluto se establece en cero. A la vista de la evolución de dicha figura

se observa cómo al final del proceso y hacia la iteración se alcanza dicho mínimo, sin variar hasta la finalización.

Así pues, la eficiencia del proceso de entrenamiento se determina en función de la máxima precisión, en este caso 100% y mínimo valor de la función de pérdida, cero en el proceso mostrado. Se infiere que el proceso en su conjunto ha sido suficientemente satisfactorio. Inicialmente ambos valores están alejados de los valores óptimos, 100% y 0, evolucionando con relativa rapidez hacia los mencionados óptimos, lo que se alcanza sobre la iteración número 12, produciéndose sendos desajustes, aproximadamente sobre la iteración 36, bastante acusado en el caso de la función de pérdida, con el pico que se aleja del valor 0 hasta el 0.6. El desajuste en la precisión es menor bajando hasta el aproximadamente el 95%, alejándose por tanto un 5% del óptimo. Dado que en cada iteración pasan todas las muestras disponibles no es posible encontrar una explicación clara, tan sólo cabe pensar que el desajuste proviene del orden de entrada de las muestras, ya que si bien en cada iteración se procesan todas, no lo hacen en la mismo orden en todas las iteraciones, ya que el orden se establece de forma aleatoria. En cualquier caso, lo realmente importante es que el proceso final acabe de forma satisfactoria, como es el caso.

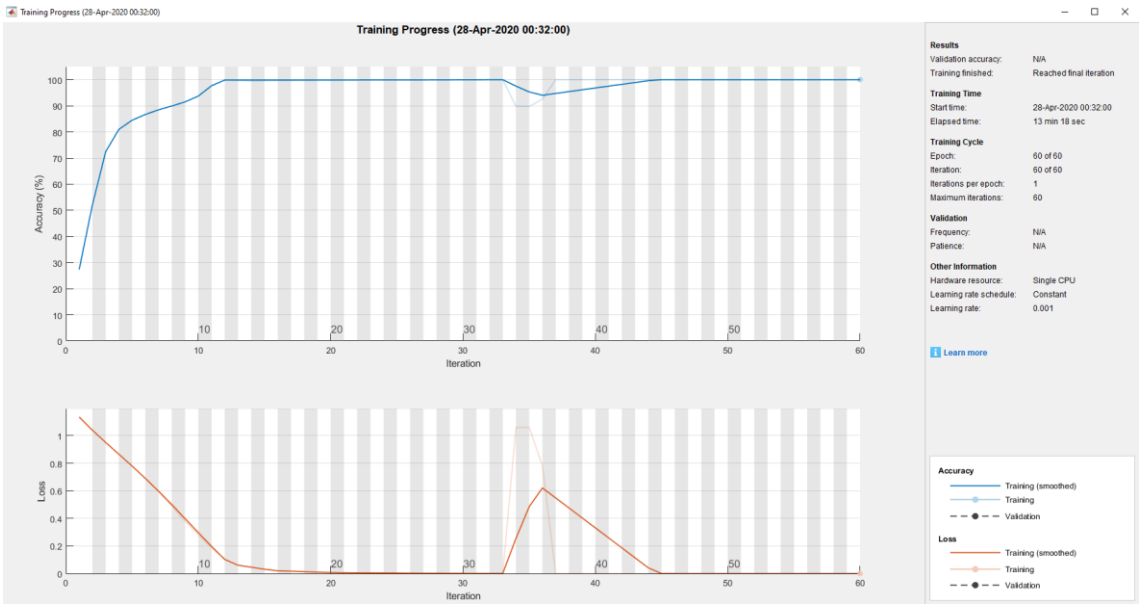


FIGURA 21- ENTRENAMIENTO DE LA RED PRINCIPAL

A continuación se muestran los resultados del entrenamiento de los modelos de red para cada uno de los sensores por separado en las figuras 18, 19 y 20. Todos los parámetros de entrenamiento son similares, pero los resultados son muy diferentes. Ninguna de las redes puede mantener una precisión del 100% tras las 60 épocas del entrenamiento, y tampoco muestran señales de estar convergiendo ni reduciendo su error, en su lugar la función de pérdida muestra continuas subidas y bajadas.

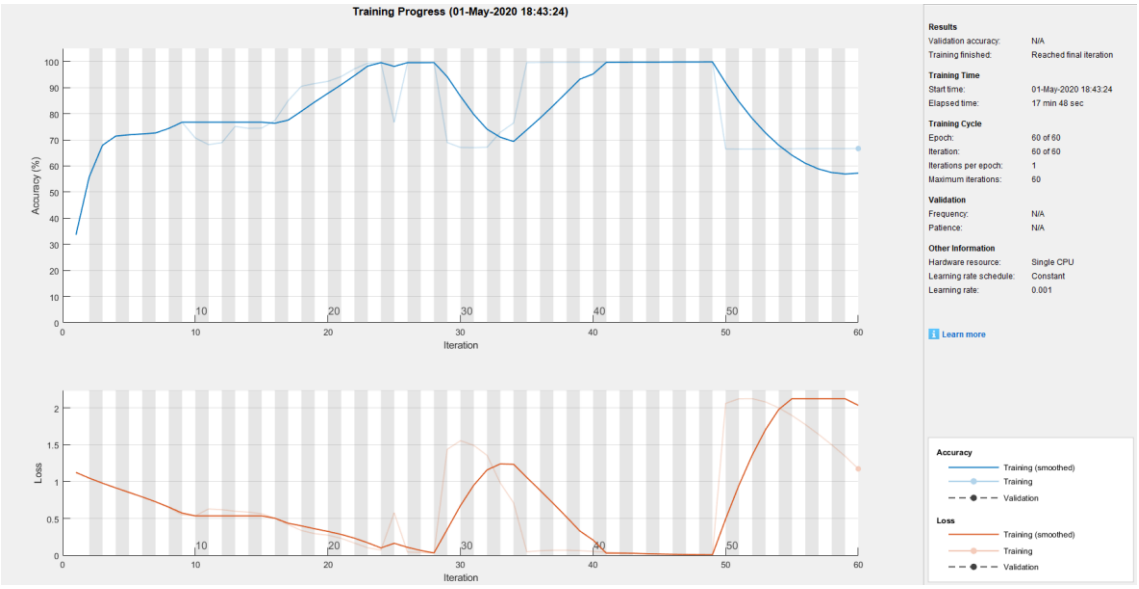


FIGURA 22- ENTRENAMIENTO DE LA RED DE ACELERACIÓN LINEAL

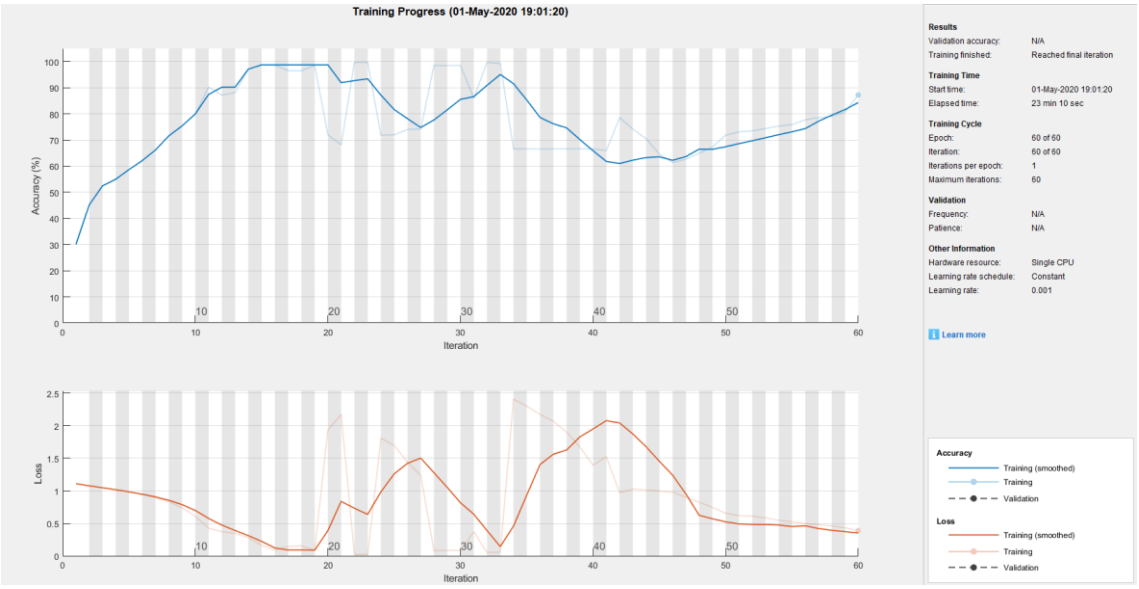


FIGURA 23- ENTRENAMIENTO DE LA RED DE ACELERACIÓN ANGULAR

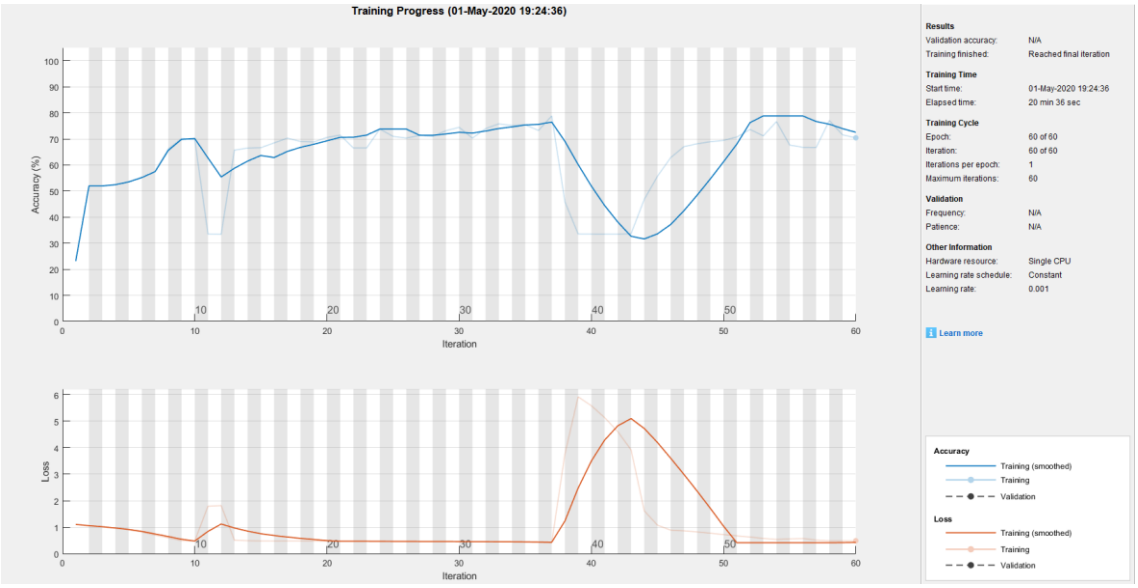


FIGURA 24- ENTRENAMIENTO DE LA RED DE ORIENTACIÓN

De estos resultados podemos discernir que la información de cada uno de los sensores por separado es insuficiente para poder realizar predicciones, pero juntos pueden dar muy buenos resultados. En la figura 25 podemos ver una comparación de los resultados obtenidos por la red de 9 unidades de entrada con las actividades que se estaban realizando. En las figuras 26, 27 y 28 se muestran los resultados de las redes de 3 unidades de aceleración lineal, angular y orientación respectivamente.

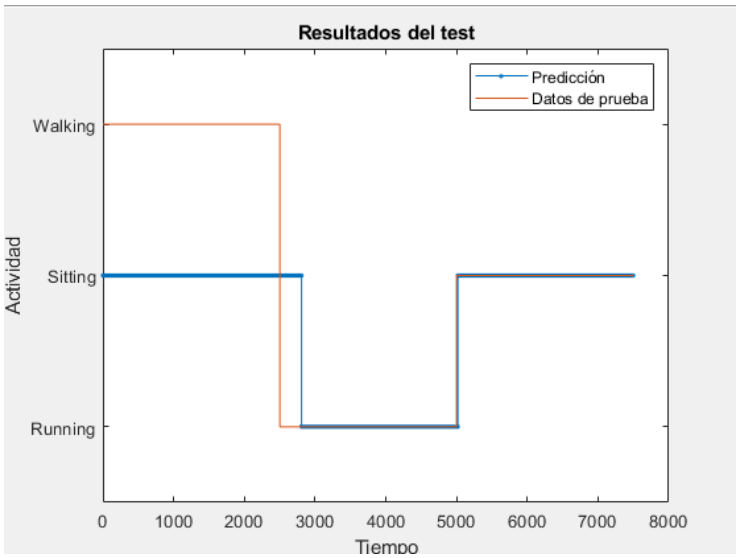


FIGURA 25- RESULTADOS DE LA RED DE 9 UNIDADES DE ENTRADA

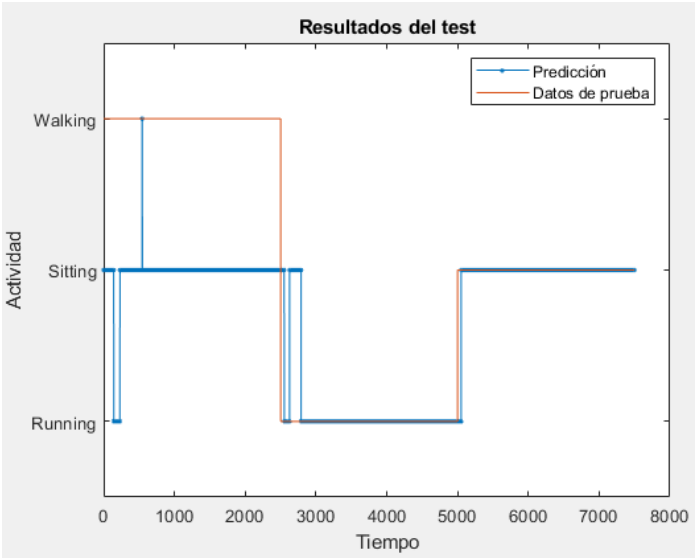


FIGURA 26- RESULTADOS DE LA RED DE ACELERACIÓN LINEAL

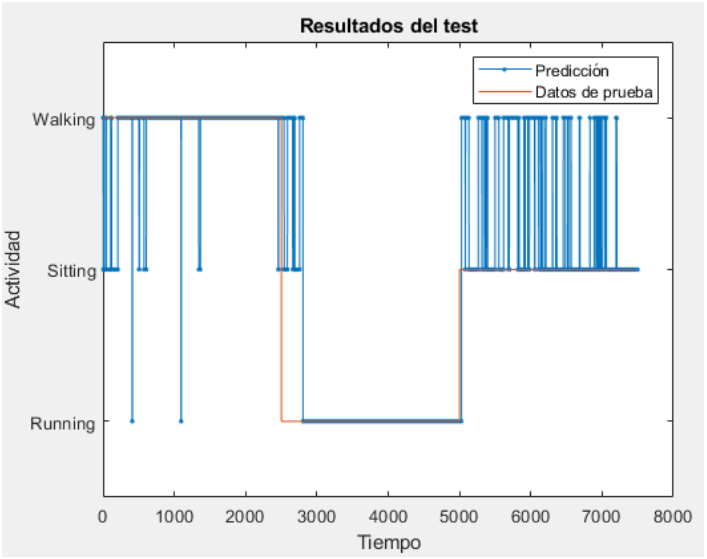


FIGURA 27- RESULTADOS DE LA RED DE ACELERACIÓN ANGULAR

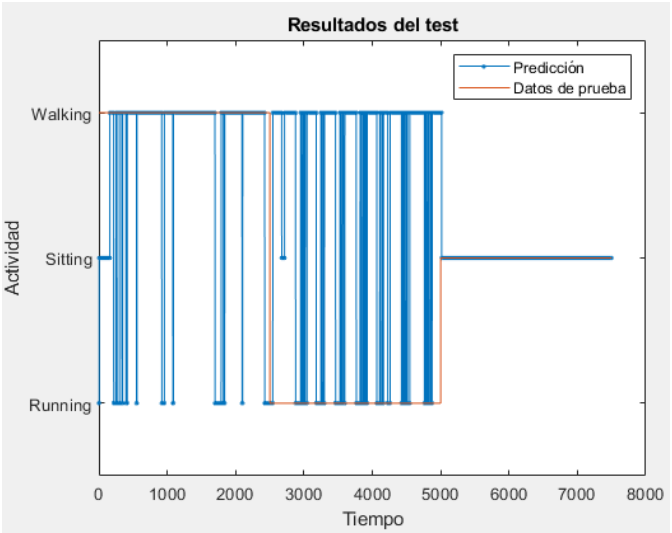


FIGURA 28- RESULTADOS DE LA RED DE ORIENTACIÓN

Se puede ver la diferencia entre la efectividad de las 4 redes en la tabla 3. En ella se muestran los resultados de la efectividad de la red para predecir casos. En las pruebas realizadas, se han utilizado 7500 casos de prueba, divididos a partes iguales entre cada actividad, esto es, 2500 casos por actividad. Para cada actividad se muestra en la columna de la izquierda el número de predicciones acertadas realizadas sobre los 2500 casos presentados, y en la de la derecha el porcentaje de acierto que representa. En la última columna, se muestran esos mismos datos para el total de los casos presentados.

La red que emplea datos de aceleración lineal es la única que da resultados más o menos constantes en el tiempo independientemente de la actividad que se esté realizando. La red que utiliza aceleración angular pude identificar la actividad de correr y la red de orientación puede identificar la actividad de sentar, pero no tienen resultados constantes para las otras dos actividades.

La red que combina datos de todos los sensores sin embargo otorga resultados constantes, si bien no identifica correctamente la actividad *Walking*.

	Walking		**Running**		**Sitting**		**Total**	
Red Principal	0	0%	2191	87,64%	2482	99,28%	4673	66.75%
Red Ac. Lineal	3	0,12%	2285	91,4%	2449	97,96%	4737	67.67%
Red Ac. Angular	2261	90,44%	2196	87,84%	1630	65,2%	6087	86.95%
Red Orientación	1995	79,8%	729	29,16%	2486	99,49%	5210	74.42%

TABLA 3- RESULTADOS DE LA PRUEBA

Cabe destacar que si bien la red tiene el menor porcentaje de acierto, sus resultados son los más constantes, tal y como se puede observar en las figuras 25-28; y su proceso de aprendizaje es el más prometedor, por lo que se espera que su rendimiento pueda mejorar mucho con mejores algoritmos de aprendizaje y muestras de entrenamiento más grandes.

6- CONCLUSIONES Y TRABAJO FUTURO

A lo largo de este proyecto se ha explorado una posible solución para el problema *HAR* con fines de monitorización de esas acciones humanas de forma remota. Se propone un sistema factible de reconocimiento de actividades humanas en tiempo real mediante el uso de dispositivos móviles equipados con sensores de movimiento y tecnología propia de *deep learning*.

Esta solución consiste en utilizar redes neuronales operando en la nube para analizar, en tiempo real, los movimientos de un usuario bajo monitorización, registrados por sensores, que aquel lleva consigo. Se propone un sistema funcional, con despliegue de una aplicación a nivel del dispositivo móvil y del servidor, que demuestra ser capaz de manejar el volumen de datos necesario para realizar las operaciones necesarias encaminadas a la detección de las actividades mencionadas. A pesar de que las predicciones realizadas con los modelos de redes neuronales desarrollados en este proyecto no son tan precisas como cabría esperar, una mejora de estas mismas es posible tal y como se propone en otros proyectos de similar naturaleza (punto 1.2).

Tras el desarrollo del sistema y las pruebas realizadas, se puede concluir lo siguiente:

1. Es posible comunicar un dispositivo móvil común con un servidor remoto, bien mediante tecnología 4G o mediante *WiFi*, a una frecuencia de intercambio de datos suficiente como para realizar predicciones en tiempo real sin pérdida de datos (punto 4.1.2).

2. Es posible obtener suficiente información para realizar predicciones acertadas utilizando los sensores presentes en un dispositivo móvil, si bien es necesario procesar adecuadamente estos datos (punto 4.1.3).

3. La red es lo suficientemente efectiva como para obtener resultados satisfactorios para un conjunto de datos suficientes (punto 5.3), aunque como se indica anteriormente es posible mejorar este rendimiento.

De cara al futuro, el sistema puede ser mejorado sustancialmente aumentando su índice de acierto. Mejorar la efectividad de la red es una propuesta razonable. Con tal propósito se han identificado las siguientes mejoras que se pueden implementar fácilmente en la misma:

1. Explorar y probar otras diferentes arquitecturas de red LSTM. Para ello se propone variar el número de capas ocultas y neuronas en éstas, mediante la realización de pruebas y contraste de resultados.

2. Explorar distintas configuraciones de los parámetros de entrenamiento, incluyendo el método de optimización del gradiente, el número de iteraciones y épocas o la razón de aprendizaje, entre otras.

3. Aumentar el tamaño de los lotes de entrenamiento, recogiendo más datos. Para que esta propuesta sea razonable, probablemente haya que optimizar el proceso de entrenamiento para que pueda aceptar volúmenes grandes de datos sin provocar fallos estructurales no soportados por el sistema, probando distintos algoritmos de entrenamiento y utilizando procesadores más potentes para realizarlo.

4. Mejorar la calidad de los datos recogidos utilizando distintas personas y dispositivos para su captura, aumentando la variedad en los mismos. También se podrían probar otras formas de dividir los datos a la hora de alimentar la red con ellos, dividiendo las épocas en iteraciones en el proceso de entrenamiento e introduciendo solapamiento en los datos.

5. Cambiar el modelo de red. Una opción que se ha investigado pero no se ha implementado es utilizar una CNN adaptada para abstraer los lotes de entrada antes de suministrarlos a la red LSTM, una técnica que da buenos resultados en otras aplicaciones similares de este tipo de redes.

7- CONCLUSIONS AND FUTURE WORK

Throughout this project, a possible solution to the HAR problem has been explored for the purpose of monitoring these human actions remotely. A feasible system for recognizing human activities in real time is proposed by using mobile devices equipped with motion sensors and deep learning approaches.

This solution consists of using neural networks operating in the cloud to analyse, in real time, the movements of a user under monitoring, recorded by sensors, which he carries with him. A functional system is proposed, with deployment of an application at the mobile device and at a server level, which shows that it is capable of handling the volume of necessary data to carry out the required operations aimed at detecting the aforementioned activities. Despite the fact that the predictions made with the neural network models developed in this project are not as accurate as might be expected, an improvement is still possible as proposed in other projects of a similar nature and content (point 1.2).

After the development of the system and the tests carried out, the following can be concluded:

1. It is possible to communicate a common mobile device with a remote server, either by 4G technology or by *WiFi*, at a data exchange frequency sufficient to make real-time predictions without data loss (point 4.1.2).

2. It is possible to obtain enough information to make accurate predictions using the sensors present in a mobile device, although it is necessary to adequately process these data (point 4.1.3).

3. The network is effective enough to obtain satisfactory results for a sufficient data set (point 5.3), although as indicated above it is possible to improve this performance.

Looking ahead, the system can be substantially improved by increasing its success rate. Improving the effectiveness of the network is a reasonable proposal. In this regard, the following improvements have been identified as feasible that can be easily implemented on the system:

1. Explore and test other different LSTM network architectures. To do this, it is proposed to vary the number of hidden layers and neurons on them, by carrying out tests and comparing results.

2. Explore different configurations of training parameters, including the gradient optimization method, the number of iterations and times, or the learning rate, among others.

3. Increase the size of training batches, collecting more data. In order to do this proposal reasonable, the training process will probably have to be optimized so that it can accept large volumes of data without causing structural failures, not supported by the system, testing different training algorithms and using more powerful processors to carry it out.

4. Improve the quality of the collected data, increasing the variability of it. Data can be organized during the network feeding process with them, dividing the epochs into iterations during the training process and introducing overlapping in the data.

5. Change the network model. One option that has been investigated but has not been implemented is to use a tailored CNN to abstract the input batches before supplying them to the LSTM network, a technique that works well in other similar applications of this type of network.

REFERENCIAS

Abdulmajid-Murad, J.Y.P. (2017). Deep Recurrent Neural Networks for Human Activity Recognition. Dong-gu, Corea.

Zhou, B., Yang, J., Li, Q. (2019). Smartphone-Based Activity Recognition for Indoor Localization Using a Convolutional Neural Network. *Sensors*, 19, 621.

Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz. J.L. (2013). A Public Domain Dataset for Human Activity Recognition Using Smartphones. In *Proc. European Symposium on Artificial Neural Networks*, Computational Intelligence and Machine Learning (ESANN 2013). Bruges (Belgium), 24-26 April 2013.

Gers, F.A., Schraudolph, N.N., Schmidhuber, J. (2002). Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research*, 3, 115-143..

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs.NE]*.

Kozina, S., Gjoreski, H., Gams, M., Luštrek, M. (2013). Efficient Activity Recognition and Fall Detection Using Accelerometers. In: Botía J.A., Álvarez-García J.A., Fujinami K., Barsocchi P., Riedel T. (eds) Evaluating AAL Systems Through Competitive Benchmarking. EvAAL 2013. Communications in Computer and Information Science, vol 386. Springer, Berlin, Heidelberg..

Matlab. (s.f.). Disponible on-line: https://es.mathworks.com/help/deeplearning/ug/sequence-to-sequence-classification-using-deep-learning.html (Accedido, Enero 2020).

Matlab. (s.f.). Disponible on-line: *https://es.mathworks.com/products/matlab-mobile.html* (Accedido, Enero 2020).

Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780.

Thingspeak. (s.f.). Disponible on-line: *https://thingspeak.com/pages/learn_more* (Accedido, Enero 2020).

Ye, H., Gu, T., Zhu, X., Xu, J., Tao, X., Lu, J., Jin, N. (2012). FTrack: Infrastructure-free floor localization via mobile phone sensing, in Silvia Giordano, Marc Langheinrich (ed.) Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom), Lugano, Switzerland, 19-23 March 2012, pp. 2-10.

APÉNDICE

PROCEDIMIENTO PARA INSTALACIÓN Y EJECUCIÓN DE LA APLICACIÓN

Este apéndice tiene por objeto proporcionar los enlaces necesarios a los repositorios donde se ubica, así como las pautas necesarias para su instalación en las distintas modalidades y plataformas locales y remotas. Además proporciona la correspondiente guía de usuario de la aplicación.

A) INSTALACIÓN DE LA APLICACIÓN WEB

VERSIÓN ABREVIADA

1. Es necesario tener una versión de Python 3.7 o posterior, y la versión de Matlab 2019a.

2. El código de la aplicación se encuentra en el siguiente enlace: https://github.com/Rpavon297/HumanActivityRecognition

3. Las librerías necesarias que pueden instalarse con *pip* están disponibles en el documento *requirements.txt*. También es necesaria la librería con la red que se encuentra en *HumanActivityRecognition/predictor/scripts/net* y se puede instalar manualmente haciendo mediante el comando *python setup.py install*.

4. La aplicación se inicia con el comando *python manage.py runserver* (el script manage.py se encuentra en la carpeta raíz del proyecto)

VERSIÓN DETALLADA

1. **Instalación del software base.** Es necesario tener Python y el entorno de ejecución de Matlab o una versión completa de Matlab, en la que dicho entorne se encuentra incluido. Los pasos concretos son los siguientes:

a) Instalar Python 3.7 o superior, disponible en el siguiente link: https://www.python.org/downloads/

b) Para poder ejecutar paquetes compilados de Matlab con los scripts requeridos para realizar predicciones, es necesario instalar en el ordenador el entorno de ejecución de Matlab, versión 2019a. No es necesario si ya existe una versión de Matlab instalada. La descarga está disponible en el siguiente link: https://es.mathworks.com/products/compiler/matlab-runtime.html.

c) En el caso de estar utilizando los sistemas operativos Linux o bien OS X, hay que asegurarse de que están configuradas las siguientes variables de entorno:

Linux:
```
setenv LD_LIBRARY_PATH
$LD_LIBRARY_PATH:mcrroot/runtime/glnxa64:
mcrroot/bin/glnxa64:mcrroot/sys/os/glnxa64:
mcrroot/sys/opengl/lib/glnxa64
```

OS X: `setenv DYLD_LIBRARY_PATH $DYLD_LIBRARY_PATH:`*`mcrroot`*`/runtime/maci64:`*`mcrroot`*`/sys/os/maci64:`*`mcrroot`*`/bin/maci64`

mcroot es la ruta de instalación del entorno de ejecución. El comando debe ser introducido sin espacios en blanco.

Más información, puede encontrarse en:
https://es.mathworks.com/help/compiler_sdk/python/install-a-matlab-compiler-sdk-python-package.html

2. **Descargar el código alojado en el siguiente repositorio**:

   https://github.com/Rpavon297/HumanActivityRecognition

3. **Crear y configurar un entorno virtual de Python.** Es necesario tener un entorno de ejecución donde poder instalar los paquetes necesarios para el funcionamiento de la aplicación.

a) Abrir la consola de comandos
b) Instalar pip (viene por defecto en caso de estar utilizando Windows) ejecutando el siguiente comando:
```
python3 -m pip install --user --upgrade pip
```

c) Instalar el módulo de python *virtualenv* ejecutando el siguiente comando:

   En macOS y Linux:
```
python3 -m pip install --user virtualenv
```

   En Windows:
```
py -m pip install --user virtualenv
```

d) Crear el entorno virtual, navegando con la consola hasta el directorio donde se encuentra la carpeta del proyecto y ejecutando el siguiente comando:

   En macOS y Linux: `python3 -m venv env`

   En Windows: `py -m venv env`

e) Activar el entorno virtual:

   En macOS y Linux: `source env/bin/activate`

   En Windows: `.\env\Scripts\activate`

f) Instalar las librerías necesarias en el entorno virtual:
```
pip3 install -r requirements.txt
```

g) Instalar la librería de Matlab para Python manualmente, abriendo en la consola la ruta *HumanActivityRecognition/predictor/scripts/net* y ejecutando el siguiente comando:
```
py setup.py install
```

4. **Iniciar el servidor** navegando desde la consola a la carpeta raíz del proyecto y ejecutando el comando:

```
py manage.py runserver
```

5. Se puede acceder al servidor desde un navegador a través de la ruta 127.0.0.1:8000

B) INSTALACIÓN DE LA APLICACIÓN ANDROID

1. Descargar el proyecto desde el siguiente link: https://github.com/Rpavon297/AndroidMotionTracker

2. Abrir desde un dispositivo móvil el archivo .apk disponible en la carpeta *app/outputs/apk/debug*

C) MANUAL DE USO

Para monitorizar un usuario, es necesario llevar consigo el dispositivo móvil con la funcionalidad sensoria activada:

1. En la pantalla principal, seleccionar el botón de “Predecir”

2. En el campo de texto marcado con “dirección de destino”, introducir la dirección donde se está ejecutando el servidor. En caso de querer conectarse a un servidor en una red local, introducir la ip y el puerto de la máquina donde se está ejecutando el servidor.

3. Al pulsar en “Iniciar”, empiezan a mostrarse en la parte superior de la pantalla los valores que están siendo capturados por los tres sensores de forma instantánea.

4. En la parte inferior se muestra el estado de conexión con el servidor.

5. El botón “Detener” interrumpe el proceso de predicción.

Para consultar la actividad que está realizando el usuario bajo monitorización, basta con acceder a la página web del servidor:

1. Introducir desde un navegador la dirección del servidor. En caso de estar ejecutándose localmente, introducir la dirección ip seguida del puerto.

2. Si es la primera vez que se accede y se está ejecutando localmente, puede tardar unos segundos en cargar mientras se inicializa la sesión.

3. Al pulsar el botón “start”, se carga en pantalla la predicción actual.

4. A partir de entonces se actualiza la página regularmente. Debajo se puede comprobar la hora de la última predicción realizada.

5. Debajo de la hora de la última predicción, se muestra el grado de seguridad con el que se ha realizado la predicción.