UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Sistema de Aprendizaje Profundo para reconocimiento de actividades con sensores de captura de movimientos

Autor:

D. Sergio Sáez Bombín

Tutor:

Dr. D. Mario Martínez Zarzuela

Valladolid, 29 de enero de 2020

TÍTULO:	**Sistema de aprendizaje Profundo para reconocimiento de actividades con sensores de captura de movimientos**
AUTOR:	**D. Sergio Sáez Bombín**
TUTOR:	**Dr. D. Mario Martínez Zarzuela**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Javier Manuel Aguiar Pérez**
SECRETARIO:	**Dr. D. David González Ortega**
VOCAL:	**Dra. Dª. Míriam Antón Rodríguez**
SUPLENTE 1:	**Dra. Dª. María Ángeles Pérez Juárez**
SUPLENTE 2:	**Dra. Dª. María García Gadañón**
SUPLENTE 3:	**Dr. D. Jaime Gómez Gil**

FECHA:	**29 de enero de 2020**
CALIFICACIÓN:	

Agradecimientos

Me gustaría manifestar mi agradecimiento al Dr. D. Mario Martínez Zarzuela por sus consejos y ayuda en la realización de este Trabajo Fin de Máster.

A mis amigos y a mi pareja, dispuestos siempre a escucharme y a darme su punto de vista, así como a apoyarme en mis decisiones.

Por último, a mis padres, mi hermano y mi hermana, por su interés, ánimo y ayuda en todo momento.

Resumen

En este este Trabajo Fin de Máster se desarrolla un sistema para el reconocimiento de actividades humanas (HAR) a partir de lo que se conoce como redes neuronales y sensores inerciales. El sistema es capaz de distinguir entre 11 actividades a partir de los datos que indican la orientación del cuerpo (cuaterniones) provenientes únicamente de 5 sensores. Las pruebas han sido realizadas con un conjunto de datos públicos conocido como REALDISP, ampliamente utilizado en la resolución de problemas HAR. También se aborda el problema de la generación de datos de movimiento a partir de redes neuronales, como complemento a la resolución del problema HAR.

Además, a lo largo de este Trabajo Fin de Máster se expone la situación en la que se encuentra hoy en día el reconocimiento de actividades físicas mediante el uso de la Inteligencia Artificial y más en concreto, del Deep Learning, así como los fundamentos matemáticos y teóricos en los que se basa el diseño de redes neuronales, con el objetivo de justificar las decisiones de diseño que se han llevado a cabo. Finalmente, se describen las redes neuronales diseñadas presentando los resultados obtenidos.

Palabras clave

Reconocimiento de actividades humanas, sensores inerciales, Inteligencia Artificial, Deep Learning, Redes Neuronales.

Abstract

In this End of Master Project a system for Human Action Recognition (HAR) is developed with Artificial Neural Networks and inertial sensors. The system can distinguish between 11 activities from data that indicate the body's orientation, that is, quaternions, which come from only 5 sensors. The tests have been done with a public dataset known as REALDISP, widely used in solving HAR problems. The generation of movement data is also addressed using neural networks, as a complement of HAR problem.

In addition, throughout this End of Master Project, the nowadays' situation of the recognition of physical activities through the use of Artificial Intelligence and, more specifically, Deep Learning, is exposed, as well as the mathematical and theoretical foundations on which the design of neural networks is based, in order to justify the design decisions that have been carried out. Finally, the neural networks designed by presenting the resulst obtained are described.

Keywords

Human activity recognition, inertial sensors, Artificial Intelligence, Deep Learning, Neural Networks.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

En este capítulo se ubica el marco de trabajo de este Trabajo Fin de Máster, describiendo los objetivos abordados en él.

Este Trabajo de Fin de Máster se ha desarrollado dentro del Grupo de Telemática e Imagen (GTI) de la Universidad de Valladolid. Este grupo utiliza desde hace varios años los sensores inerciales (IMUs, *Inertial Measurement Units*) como un elemento base de sus investigaciones, con la intención de determinar la correcta realización de un movimiento en juegos serios interactivos para ayudar a la recuperación de la movilidad de usuarios (rehabilitación) mediante su utilización. Ya en el Trabajo Fin de Grado (Sáez Bombín, S., 2018) se hizo una primera incursión al campo del aprendizaje automático con este tipo de datos para mejorar la detección de los movimientos.

Este Trabajo Fin de Máster es una continuación de ese trabajo (Sáez Bombín, S., 2018) que se limitó a sentar las bases del reconocimiento de actividades de manera automática mediante redes neuronales profundas (DNN, *Deep Neural Network*). El valor de este Trabajo Fin de Máster es el de mejorar el aprendizaje realizado por esas redes neuronales y llegar a puntos en los que no se llegó durante el Trabajo Fin de Grado (Sáez Bombín, S., 2018), como estudiar su viabilidad para su inclusión en un sistema embebido y de tiempo real y la apertura de otra vía de estudio, como es la generación de datos de manera sintética.

Las redes neuronales pertenecen al ámbito del Deep Learning, que a su vez forma parte del Machine Learning y que se engloba dentro del amplio campo conocido como Inteligencia Artificial.

1.1 Motivación y objetivos

El reconocimiento automático de actividades físicas humanas se enmarca dentro del campo de investigación del Reconocimiento de la Actividad Humana (HAR, *Human Activity Recognition*), y es un área de gran interés en diversos sectores, como son los deportes, el entretenimiento, la industria y, el que más se acerca al ámbito de este Trabajo Fin de Máster, el sector de la salud (San, P. *et al.*, 2017, pp. 186-204).

Este campo cobra cada vez más interés, favorecido por el desarrollo de sistemas multisensoriales en los que se utiliza un conjunto de sensores inerciales sobre el cuerpo (*wearable on-body sensors*), que suelen ser acelerómetros, giroscopios o sensores de campo magnético (Sáez Bombín, S., 2018).

Como se indica en (Ordóñez, F. J. y Roggen, D., 2016), el problema HAR está basado en la premisa de que los movimientos del cuerpo se trasladan a patrones específicos de señales provenientes de sensores, y estos pueden ser detectados y clasificados mediante algoritmos de aprendizaje de Machine Learning.

Este último detalle es el que ha hecho que el HAR basado en sensores sobre el cuerpo humano haya tenido un gran avance en aplicaciones como videoconsolas, *fitness*, toma

de medicamentos y monitorización de la salud durante los últimos años, ya que la utilización de sensores sobre el cuerpo elimina la importancia de la ubicación del usuario en el reconocimiento de las actividades (Ordóñez, F. J. y Roggen, D., 2016), a diferencia de las cámaras de vídeo, utilizadas anteriormente en este problema (Hassan, M. M. *et al*, 2018).

Sin embargo, tal y como se indica en (San, P. *et al*., 2017, pp. 186-204), según (Bulling, A. *et al.*, 2014), es preferible utilizar las señales procedentes de los sensores en vez de las de una cámara de vídeo para el problema HAR, debido principalmente a 3 razones:

- Los sensores *wearable* reducen las limitaciones del entorno, además de evitar los marcos estacionarios que sufren las cámaras de vídeo.
- Utilizar varios sensores *wearable* permite una captura más precisa y efectiva de la señal.
- Los sensores *wearable* aportan mayor privacidad. Esto es así porque los sensores únicamente dan información del sujeto que los lleva, mientras que al utilizar cámaras de vídeo también puede haber información de otros sujetos que aparezcan en escena que no son objeto de estudio.

En contrapartida, los sensores son más sensibles a la variabilidad entre clases, que se refiere a que distintos sujetos realizan la misma actividad con movimientos distintos, así como a la semejanza entre clases, por ejemplo, entre el *jogging* y el *running*, en lo que el movimiento del cuerpo es semejante, pero son actividades distintas (San, P. *et al.*, 2017, pp. 186-204).

El éxito del HAR tiene como factor clave la representación efectiva de las series temporales recogidas por los sensores *wearable*. De hecho, convencionalmente, el problema HAR es considerado una aplicación específica del análisis de series temporales (Sáez Bombín, S., 2018).

En este punto es en el que tiene fuerza el Deep Learning (las redes neuronales de aprendizaje profundo), ya que ayuda a modelar abstracciones de alto nivel sobre los datos (Bengio Y., 2009, citado en San, P. *et al.*, 2017, pp. 188). La forma en la que consigue esto es mediante el procesado no lineal de información para la extracción de características, organizadas de forma jerárquica, y su posterior clasificación, en la que cada capa de la red procesa los datos de la capa anterior (Sáez Bombín, S., 2018). Además, tiene como ventaja con respecto a otros algoritmos de Machine Learning que es la propia red la que realiza la extracción de características, y no tiene que hacerse un procesado previo exhaustivo (Sáez Bombín, S., 2018).

El hecho de tratar con sensores *wearable*, aunque reduzca la dificultad de obtención de los datos al no tener en cuenta el entorno, hace que la señal recogida sea susceptible a fallos que pueden provenir de unas condiciones inadecuadas del entorno, en el que pueda haber interferencias que afecten a los sensores. Por ello es de interés el planteamiento de generar, en ciertas ocasiones, datos a partir de una red neuronal.

Por lo tanto, este Trabajo Fin de Máster presenta **dos objetivos principales**:

1. **Realizar un sistema de reconocimiento de actividades mediante modelos de redes neuronales de aprendizaje profundo con datos provenientes de un dataset de movimientos capturado con sensores inerciales y estudiar su**

viabilidad de inclusión en un sistema embebido y en tiempo real, con la intención de servir de base para una red neuronal que permita, mediante los sensores del Grupo de Telemática e Imagen, el reconocimiento de movimientos o actividades físicas para los juegos serios que en él se desarrollan y en definitiva, para resolver el problema HAR.

2. **Desarrollar una red neuronal que permita la generación de datos de los movimientos deseados sin necesidad de ser recogidos por un sensor**.

El sistema que se propone tendría la estructura que se muestra en la Figura 1. Constaría de un módulo de obtención de datos, en el que se realizaría la recolección de las señales provenientes de los sensores, otro de preprocesamiento y otro en el que se realice la clasificación.

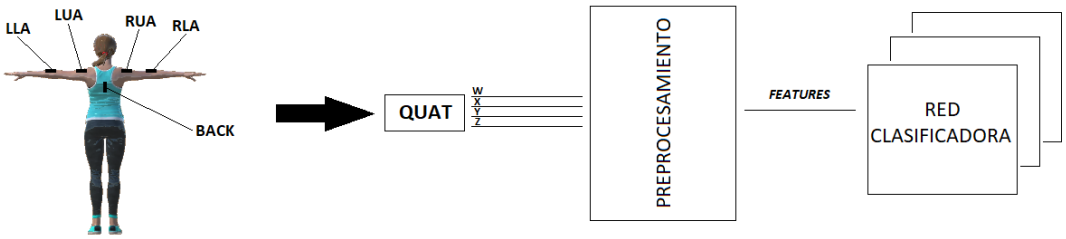


Figura 1. *Idea de sistema propuesto.*

1.2 Fases y métodos

Las fases en las que se ha dividido este Trabajo Fin de Máster se exponen brevemente a continuación:

- **Búsqueda de datasets**: Esta fase es inmediata, ya que se utiliza el mismo dataset que se utilizó en (Sáez Bombín, S., 2018).
- **Visualización de datos**: Se realiza una reconstrucción 3D de los datos para conocer mejor qué se está utilizando y tomar decisiones sobre ellos, es decir, permite realizar un estudio profundo de los datos del dataset.
- **Limpieza de dataset**: La fase anterior permite encontrar imperfecciones o datos que no son de interés y que no se tienen en cuenta para este Trabajo Fin de Máster. Por ello, en esta fase se toman una serie de decisiones que van a permitir elegir los datos de mayor interés para el sistema Deep Learning.
- **Revisión bibliográfica de trabajos HAR**: Para realizar un sistema de Deep Learning que tiene como objetivo resolver el problema HAR es necesario conocer el Estado del Arte de este y cómo se puede abordar desde los puntos de vista del Machine Learning, así como del Deep Learning.
- **Estudio de Keras**: Partiendo de que se conocen los conceptos teóricos del Deep Learning, solamente es necesario realizar un estudio de las bibliotecas de funciones que se utilizan. En este caso, esta es Keras, que va a permitir implementar redes neuronales desde un punto de vista de alto nivel.

- **Acondicionamiento de dataset**: Una vez cumplidas las fases anteriores, se está en condiciones de diseñar el sistema de Deep Learning. Además del estudio y limpieza del dataset, se realiza esta fase a modo de preprocesamiento de los datos, con el fin de adecuarlos a la red y que se optimice su rendimiento.
- **Propuesta de redes**: Ya con los datos en su formato final, se pueden empezar a tomar decisiones de diseño de las redes neuronales.
- **Pruebas realizadas**: Por cada una de las decisiones de diseño se realizan pruebas y un estudio de los resultados. De esta forma se pueden ajustar los parámetros de esta de manera iterativa y una vez que se considera que se obtiene un resultado óptimo con una red, se vuelve a la fase anterior para seguir realizando otras pruebas, hasta conseguir un resultado satisfactorio.

1.3 Medios disponibles

1.3.1 Hardware

El equipamiento hardware utilizado para este Trabajo Fin de Máster consiste en un servidor alojado en la ETSIT de la Universidad de Valladolid que consta de los siguientes componentes:

- Procesador CPU Intel® Core™ i5-4690 con las siguientes características (Intel Corporation, n.d.):
  - Frecuencia básica del procesador: 3.50 GHz
  - Frecuencia turbo máxima: 3.90 GHz
  - Número de núcleos: 4
  - Cantidad de subprocesos: 4
- Memoria RAM: 8 GB
- Una tarjeta gráfica GPU Nvidia® GeForce GTX 1060 con chip GP106 y 6GB de RAM (NVIDIA Corporation, 2017):
  - Arquitectura de GPU: Pascal
  - NVIDIA CUDA® Cores: 1280
  - Frecuencia de reloj normal: 1506 MHz
  - Frecuencia de reloj acelerada: 1708 MHz
  - Memoria de vídeo: 6 GB GDDR5/X
  - Frecuencia de la memoria: 8Gbps
  - Ancho de la interfaz de memoria: 192-bit
  - Ancho de banda de memoria: 192 GB/s

1.3.2 Software

En cuanto al software utilizado en este Trabajo Fin de Máster cabe destacar lo siguiente:

Python

El lenguaje de programación utilizado en este Trabajo Fin de Máster ha sido Python.

Python es un lenguaje de programación interpretado y orientado a objetos, caracterizado por su sintaxis clara y legible. Puede ser interpretado por un gran número de Sistemas Operativos (los basados en UNIX, Mac OS, MS-DOS, 0S/2 y Windows) y actualmente es considerado el lenguaje de programación más popular, recibiendo un impulso por parte de grandes empresas como Google (Python, 2019).

Fue creado por Guido van Rossum, procedente de Holanda, y el código es de libre acceso.

Una característica acerca de su sintaxis clara es el indentado que utiliza, evitando el uso de llaves y dejando más limpio el código. Además, Python ofrece tipos de datos dinámicos, clases de lectura e interfaces para varias llamadas al sistema y bibliotecas. Puede ser extendido, usando el lenguaje C o C++ (Sáez Bombín, S., 2018).

En la realización de este Trabajo Fin de Máster se ha hecho uso de la versión 3.6.8 de este lenguaje.

Figura 2. *Logotipo de Python.*

Tensorflow

Tensorflow es una biblioteca software de código abierto para computación numérica utilizando grafos de flujos de datos. Es una plataforma *end-to-end* que hace muy sencillo el despliegue de modelos de Machine Learning, tanto a gente experta como a principiantes (Tensorflow, 2019).

Originalmente desarrollada por Google Brain Team en la organización de investigación de Machine Intelligence de Google para la investigación en Machine Learning y Redes Neuronales Profundas, se ha convertido en una biblioteca potente que nos permite enfrentarnos a cualquier problema de cualquier dominio del mundo real (Tensorflow, 2019).

Es una plataforma que puede funcionar sobre CPUs, GPUs, móviles y sistemas embebidos (en su versión *Lite*) e incluso en unidades de procesamiento de tensores (TPUs), que son hardware especializado sobre el que realizar matemática de tensores, haciéndola así una biblioteca muy demandada en el sector del Machine Learning.

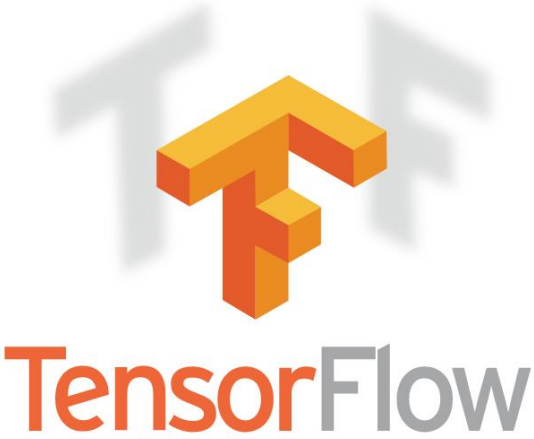


Figura 3. *Logotipo de Tensorflow.*

Jupyter

El Proyecto Jupyter existe para desarrollar software de código abierto, estándares abiertos y servicios para computación interactiva a través de docenas de lenguajes de programación, en concreto más de 40, entre los que se incluyen algunos como Python, R, Julia y Scala.

En concreto, se ha hecho uso del Notebook de Jupyter, aplicación web de código abierto que permite crear y compartir documentos que contienen código *"vivo",* ecuaciones, visualizaciones y texto narrativo. Las salidas de estos Notebooks son interactivas (se puede ir ejecutando el código "paso a paso" y son muy ricas en cuanto a variedad, ya que permiten salidas del tipo HTML, imágenes, vídeos, LaTeX y tipos personalizados de MIME (Jupyter, 2019).

Además, tienen la ventaja de que permiten hacer uso de herramientas y bibliotecas del mundo del Big Data y la Ciencia de Datos, como pueden ser: pandas, scikit-learn y Tensorflow, todas ellas utilizadas durante este Trabajo Fin de Máster.

Para este trabajo Fin de Máster se ha utilizado sus versiones de JupyterHub y JupyterLab. Mientras que JupyterHub es una versión multi-usuario que permite la distribución de Jupyter Notebooks a distintos usuarios dentro de una estructura centralizada (compatible con Docker y Kubernetes), JupyterLab es definida como la próxima generación de interfaces de Jupyter Notebooks, es decir, es un entorno interactivo, no solamente para Jupyter Notebooks sino también para código y datos, siendo totalmente flexible y modular, permitiendo la adición e integración de nuevos componentes (Jupyter, 2019).

Figura 4. *Logotipos de Jupyter y JupyterHub.*

Docker

Docker es la compañía líder en el mercado de los contenedores y permite una independencia verdadera entre aplicaciones e infraestructura.

Docker desbloquea el potencial de las organizaciones, al permitir la construcción de todo tipo de aplicaciones y microservicios sobre cualquier infraestructura (Windows, Linux e incluso *mainframes*). Esto trae consigo una mejora del rendimiento de las organizaciones de un 300% además de una reducción de los costes (optimiza el uso de los recursos de las infraestructuras y las operaciones de optimización para reducir en un 50% los costes) (Docker, 2019).

La plataforma de contenedores Docker es utilizada por millones de desarrolladores y por clientes comerciales como ADP, GE, MetLife, PayPal y Societe Generale (Docker, 2019).

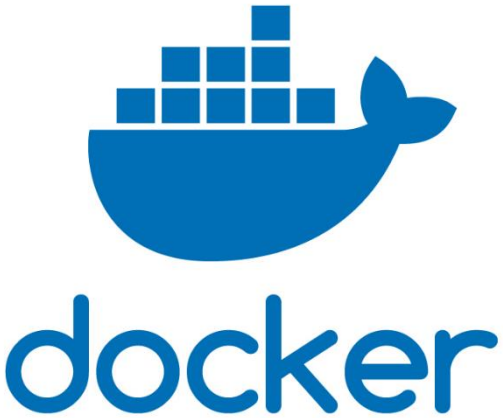


Figura 5. *Logotipo de Docker.*

Keras

Keras es una API de alto nivel (también se puede ver como un *meta-framework*) para redes neuronales, es decir, para Deep Learning, escrita en Python y con posibilidad de correr sobre Tensorflow, CNTK o Theano. La idea del uso de Keras es la posibilidad de poder desarrollar modelos de una manera rápida y así aumentar la experimentación (Keras, 2019).

Según ellos mismo indican, aconsejan usar Keras si se busca un prototipado fácil y rápido, si se van a utilizar redes convolucionales y recurrentes (y sus combinaciones) y si se quiere utilizar tanto en CPU como en GPU (Keras, 2019).

En este Trabajo Fin de Máster se ha utilizado la versión 2.2.4 contenida en Tensorflow 1.14.0.

Figura 6. *Logotipo de Keras.*

Unity

Unity es un motor de juegos multiplataforma desarrollado por Unity Technologies. Se utiliza para desarrollo 3D, 2D, realidad virtual y aumentada, lo que permite que se utilice no solamente en el mundo del videojuego sino prácticamente en cualquier ámbito. En el caso de este Trabajo Fin de Máster se ha utilizado para realizar un modelado 3D de los movimientos.

Con respecto a plataforma 3D, Unity es el líder mundial permitiendo crear y monetizar experiencias en tiempo real a través de una serie de herramientas que ofrecen. Por esto y por los socios de los que dispone (Google, Facebook, Oculus, Autodesk, Microsoft…) se mantiene al frente en el desarrollo de aplicaciones 3D, siendo utilizado por 3 billones de dispositivos en todo el mundo (Unity, 2019).

Figura 7. *Logotipo de Unity.*

1.4 Estructura de la memoria

Esta memoria se estructura en varios capítulos en los que se abordan los diferentes pasos y acciones realizados para la consecución de los objetivos de este Trabajo Fin de Máster.

En el **primer capítulo** se ha presentado una introducción al reconocimiento de actividades, ámbito en el que se enmarca este trabajo, y se han definido las motivaciones y objetivos de este.

En el **segundo capítulo**, se ahonda más en el reconocimiento de actividades, ya que en él se estudiarán los principios básicos de los sensores inerciales, fuente de los datos usados en todo problema de reconocimiento de actividades físicas, y los fundamentos matemáticos de los cuaterniones, que son los datos utilizados durante todo este trabajo.

El **tercer capítulo** se centra en el Deep Learning en todas sus dimensiones, ya que se aborda desde el punto de vista histórico, teórico y práctico. En ese capítulo se realizará una introducción a lo que se conoce como Deep Learning y se mostrarán los hitos principales de su historia para conocer su evolución y su potencial. Además, se explicará la base teórica del Deep Learning, mostrando algunos de sus fundamentos matemáticos y poniendo ejemplos para afianzar y aclarar las explicaciones más teóricas. Finalmente, se comentarán algunos trabajos en los que se ha basado este Trabajo Fin de Máster.

El **cuarto capítulo** es el referente a los resultados obtenidos. Además, en él se muestra el estudio y diseño de distintos tipos de redes neuronales de aprendizaje profundo para el reconocimiento y generación de actividades físicas.

Finalmente, en el **quinto capítulo**, se discutirán ciertos aspectos de los resultados obtenidos y se mostrarán las conclusiones sacadas de este Trabajo Fin de Máster, así como las líneas futuras que se pueden seguir en trabajos sucesivos que sirvan de complemento a este.

Además, se incluyen dos anexos:

- **ANEXO I**: En este anexo se muestran las matrices de confusión de los mejores resultados obtenidos. Estas matrices son una forma habitual de representar los resultados en trabajos que manejan algoritmos de Machine Learning y Deep Learning**.**
- **ANEXO II**: Aquí se presenta un resumen enviado a la **ASPAI'2020 Conference** de Berlín (1-3 de abril), a la espera de su aceptación y el desarrollo de un artículo para esa misma conferencia.

Capítulo 2. Reconocimiento de movimientos con sensores inerciales

Los sensores inerciales son parte fundamental en el problema del Reconocimiento de Actividades Humanas, tanto que se le dedica un capítulo en esta memoria. En este, se explica su principio de funcionamiento y la forma de obtener los datos de interés, los cuaterniones, a partir de las medidas del sensor.

2.1 IMUs

En primer lugar, se explica qué es un sensor inercial o Unidad de Medida Inercial (IMU, *Inertial Measurement Unit*):

Los sensores inerciales son dispositivos de medida electrónicos que permiten estimar la orientación de un cuerpo a partir de las fuerzas inerciales que experimenta este. El principio de funcionamiento consiste en la medida de aceleración, velocidad angular, orientación del campo magnético y los cuaterniones a partir del acelerómetro, giróscopo o giroscopio, magnetómetro y una combinación de estos, respectivamente (Sáez Bombín, S., 2018).

Es el componente principal de sistemas de guía inercial utilizados en vehículos aéreos, espaciales, marinos y aplicaciones robóticas (González Alonso, J., 2017, pp. 103-106).

Las IMUs más básicas están compuestas únicamente por un acelerómetro y un giroscopio, generalmente triaxiales (dan valores en los ejes X, Y, Z), y estiman su orientación en el espacio a partir de una aceleración y una velocidad angular concretas (González Alonso, J., 2017, pp. 103-106).
En este Trabajo Fin de Máster se disponen de datos procedentes de los siguientes componentes de un sensor inercial:

- **Acelerómetro**: Instrumento capaz de medir la aceleración en uno, dos o tres ejes. En su forma más básica consiste en una masa suspendida por un muelle que puede moverse en la dirección de medida del acelerómetro, por lo que un acelerómetro triaxial está formado por tres acelerómetros orientados ortogonalmente entre sí, dando información de manera independiente en cada una de las direcciones del espacio. Sin embargo, las IMUs incorporan acelerómetros integrados en silicio a través de la tecnología llamada MEMS (*Micro-machined Electro Mechanical System*). Su principio de funcionamiento es a partir de la medida del voltaje obtenido entre dos placas, variando una de ellas su posición dependiendo del movimiento, lo que los hace de carácter capacitivo (González Alonso, J., 2017, pp. 103-106).

- **Giroscopio o giróscopo**: Este instrumento es capaz de medir la velocidad angular de rotación del cuerpo con respecto al sistema de referencia inercial. En las IMUs están también implementados con la tecnología MEMS, dando como salida un voltaje variable en función de la velocidad angular que experimenta el sensor (González Alonso, J., 2017, pp. 103-106). Esta salida se da en grados por segundo (V/º/s) e integrándola, se puede obtener el ángulo de rotación del cuerpo sobre un eje. Al igual que en el caso del acelerómetro, para obtener un giroscopio triaxial se deben combinar tres giroscopios uniaxiales orientados ortogonalmente entre sí (Sáez Bombín, S., 2018).
- **Magnetómetro**: Incluido en algunas IMUs, es el elemento sensible al campo magnético, capaz de medir la fuerza y/o dirección del campo magnético con respecto al campo magnético terrestre (González Alonso, J., 2017, pp.103-106).

Al igual que toda la electrónica, las IMUs se han ido miniaturizando a lo largo de los años y son cada vez más accesibles a nivel económico, lo que permite disponer de manera más o menos sencilla de sistemas de medida que pueden ser colocados por todo el cuerpo (*wearable on-body sensors*) obteniendo su posición y movimiento en cualquier entorno (Sáez Bombín, S., 2018), de ahí su aplicación al problema HAR.

Así, esta tecnología permite extraer patrones cinemáticos del movimiento humano de una forma más sencilla, a diferencia de cuando se utilizan cámaras de vídeo, ya que requieren de algoritmos complejos, requiriendo un procesado de la imagen y un modelado muy condicionados por el entorno, como pueden ser las condiciones de iluminación, tal y como se ha indicado en la sección anterior (Sáez Bombín, S., 2018).

Finalmente, a partir de las medidas de estos tres elementos de los sensores inerciales se obtienen los datos que se utilizan en este Trabajo Fin de Máster, los **cuaterniones**.

2.2 Cuaterniones

2.2.1 Definición

Los cuaterniones son números hipercomplejos, denominados así los números complejos cuyo rango es superior a 2. En concreto, tienen un rango de 4 y presentan la forma mostrada en la ecuación (1).

$$q = q_0 + \mathbf{q} = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 \tag{1}$$

Además, en esta ecuación es necesario tener en cuenta (2).

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{2}$$

2.2.2 Notación

La forma de denotar las bases ortonormales del espacio tridimensional $\mathbb{R}^3$ es mediante **i**, **j** y **k**, y se pueden escribir como se muestra en (3).

$$\mathbf{i} = (1,0,0) \qquad \mathbf{j} = (0,1,0) \qquad \mathbf{k} = (0,0,1) \tag{3}$$

Por lo tanto, un cuaternión define un elemento en $\mathbb{R}^4$ (4).

$$q = (q_0, q_1, q_2, q_3) \tag{4}$$

Siendo q_0, q_1, q_2 y q_3 números reales. Combinando la ecuación de las bases ortonormales del espacio tridimensional (3) y la del cuaternión (4), se obtiene (5).

$$q = q_0 + \mathbf{q} = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 \tag{5}$$

Donde q_0 es la parte escalar (y real) del cuaternión mientras que **q** es la parte vector del cuaternión, siendo q_0, q_1, q_2 y q_3 las componentes del cuaternión (Sáez Bombín, S., 2018).

2.2.3 Rotación de cuaterniones

Es de interés conocer la forma en la que se obtienen los cuaterniones de las medidas del acelerómetro, giroscopio y magnetómetro (Sáez Bombín, S., 2018).

Siguiendo la explicación de (Valenti, R. G. *et al*., 2015, pp. 19308-19314), se toma esta notación:

- Lo referente al sensor se denota con el superíndice L (local), mientras que lo referente a la tierra se denota con el superíndice G (global).
- Para el acelerómetro, la aceleración medida se denota como $\boldsymbol{a}^{\mathrm{L}}$, y la aceleración gravitacional como $\boldsymbol{g}^{\mathrm{G}}$ y se definen en las ecuaciones (6) y (7) respectivamente.

$$\boldsymbol{a}^{\mathrm{L}} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T, \quad \|\boldsymbol{a}\| = 1 \tag{6}$$

$$\boldsymbol{g}^{\mathrm{G}} = [0 \quad 0 \quad 1]^T \tag{7}$$

- Para el giroscopio, la medida de la velocidad angular $\boldsymbol{w}^{\mathrm{L}}$ (8).

$$\boldsymbol{w}^{\mathrm{L}} = \begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^T \tag{8}$$

- Y la medida del campo magnético por el magnetómetro es $\boldsymbol{m}^{\mathrm{L}}$ (9), mientras que el verdadero campo magnético es $\boldsymbol{h}^{\mathrm{G}}$ (10).

$$\boldsymbol{m}^{\mathrm{L}} = \begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T, \ \|\boldsymbol{m}\| = 1 \tag{9}$$

$$\boldsymbol{h}^{\mathrm{G}} = \begin{bmatrix} h_x & h_y & h_z \end{bmatrix}^T, \ \|\boldsymbol{h}\| = 1 \tag{10}$$

En (Valenti, R. G. *et al*., 2015, pp. 19308-19314), la derivación algebraica del cuaternión $\boldsymbol{q}_{\mathrm{G}}^{\mathrm{L}}$, del marco global (G, la tierra) con respecto al marco local (L, el sensor), se presenta como una función de $\boldsymbol{a}^{\mathrm{L}}$ y $\boldsymbol{m}^{\mathrm{L}}$, lo que genera el sistema de ecuaciones (11).

$$\begin{cases} R^T(\boldsymbol{q}_G^L)\boldsymbol{a}^{\mathrm{L}} = \boldsymbol{g}^{\mathrm{G}} \\ R^T(\boldsymbol{q}_G^L)\boldsymbol{m}^{\mathrm{L}} = \boldsymbol{h}^{\mathrm{G}} \end{cases} \tag{11}$$

En el que hay que tener en cuenta que la ecuación de rotación de un vector entre dos marcos A y B mediante un cuaternión (12).

$$\boldsymbol{v}^{\mathrm{B}} = \mathrm{R}(\boldsymbol{q}_{\mathrm{A}}^{\mathrm{B}})\boldsymbol{v}^{\mathrm{A}} \tag{12}$$

Con todo esto, una solución posible para hallar $\boldsymbol{q}_{\mathrm{G}}^{\mathrm{L}}$ es mediante su descomposición en dos cuaterniones auxiliares, referentes al acelerómetro $\boldsymbol{q}_{acc}$ y magnetómetro $\boldsymbol{q}_{mag}$ (13).

$$\boldsymbol{q}_{\mathrm{G}}^{\mathrm{L}} = \boldsymbol{q}_{acc} \otimes \boldsymbol{q}_{mag} \tag{13}$$

Con (14) como sigue.

$$R(\boldsymbol{q}_G^L) = R(\boldsymbol{q}_{acc})R(\boldsymbol{q}_{mag}) \tag{14}$$

Cuaternión del acelerómetro

El cuaternión auxiliar del acelerómetro ($\boldsymbol{q}_{acc}$) (Valenti, R. G. *et al*., 2015, pp. 19308-19314) realiza una transformación entre las dos observaciones del vector de gravedad en los dos marcos de referencia, global y local (15).

$$\mathrm{R}(\boldsymbol{q}_{\mathrm{G}}^{\mathrm{L}})\, \boldsymbol{g}^{\mathrm{G}} = \boldsymbol{a}^{\mathrm{L}} \tag{15}$$

Y utilizando las dos últimas ecuaciones vistas, (14) y (15), se obtiene (16).

$$R(\boldsymbol{q}_{acc})R(\boldsymbol{q}_{mag}) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{16}$$

No se entra en el desarrollo de esta ecuación, pero en él se genera un sistema totalmente determinado, y tras unas consideraciones, se llega a que se tienen dos soluciones reales, mostradas en (17).

$$\boldsymbol{q}_{acc} = \begin{cases} \left[\sqrt{\frac{a_z + 1}{2}} \quad -\frac{a_y}{\sqrt{2(a_z + 1)}} \quad \frac{a_x}{\sqrt{2(a_z + 1)}} \quad 0\right]^T & , a_z \geq 0 \\ \left[-\frac{a_y}{\sqrt{2(1 - a_z)}} \quad \sqrt{\frac{1 - a_z}{2}} \quad 0 \quad \frac{a_x}{\sqrt{2(1 - a_z)}}\right]^T & , a_z < 0 \end{cases} \tag{17}$$

Cuaternión del magnetómetro

Para el cálculo del cuaternión auxiliar del magnetómetro ($\boldsymbol{q}_{mag}$) (Valenti, R. G. *et al.*, 2015, pp. 19308-19314), primero se utiliza el cuaternión $\boldsymbol{q}_{acc}$. Con él se rota $\boldsymbol{m}^{\mathrm{L}}$ en un marco intermedio cuyo eje Z sea el mismo que el del marco global (G), mostrado en (18).

$$\mathrm{R}^T(\boldsymbol{q}_{acc})^{\mathrm{L}}\boldsymbol{m} = \boldsymbol{l} \tag{18}$$

Lo que lleva a la ecuación (19).

$$\mathrm{R}^T(\boldsymbol{q}_{mag}) \begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} = \begin{bmatrix} {l_x}^2 + {l_y}^2 \\ 0 \\ l_z \end{bmatrix} \tag{19}$$

Al igual que antes, el desarrollo de esta ecuación conduce a un sistema totalmente determinado, teniendo como soluciones reales las que se ven en (20).

$$\boldsymbol{q}_{mag} = \begin{cases} \left[\frac{\sqrt{\Gamma + l_x\sqrt{\Gamma}}}{\sqrt{2\Gamma}} \quad 0 \quad 0 \quad \frac{l_y}{\sqrt{2}\sqrt{\Gamma + l_x\sqrt{\Gamma}}}\right]^T & , l_x \geq 0 \\ \left[\frac{l_y}{\sqrt{2}\sqrt{\Gamma - l_x\sqrt{\Gamma}}} \quad 0 \quad 0 \quad \frac{\sqrt{\Gamma - l_x\sqrt{\Gamma}}}{\sqrt{2\Gamma}}\right]^T & , l_x < 0 \end{cases} \tag{20}$$

Con $\Gamma = {l_x}^2 + {l_y}^2$

Finalmente, el cuaternión de orientación del marco global (G, la tierra) con respecto al marco local (L, el sensor) quedará como en la ecuación (21).

$$\boldsymbol{q}_{\mathrm{G}}^{\mathrm{L}} = \boldsymbol{q}_{acc} \otimes \boldsymbol{q}_{mag} \tag{21}$$

Si bien esta es una forma de calcular los cuaterniones, cada sensor tiene su algoritmo de cálculo, y en concreto, el algoritmo del cálculo de cuaterniones de Xsens (sensores inerciales de los que se obtienen los datos en este Trabajo Fin de Máster) es propietario y no se sabe con seguridad qué pasos siguen para ello.

2.2.4 Cuaterniones en el problema HAR

La motivación de utilizar cuaterniones para resolver el problema HAR, tanto en el Trabajo Fin de Grado (Sáez Bombín, S., 2018) como en este Trabajo Fin de Máster viene del potencial que tienen para poder representar los movimientos. La predicción de movimientos mediante rotaciones 3D (cuaterniones) permite el uso de un esqueleto parametrizado (Pavllo, D. *et al.*, 2018) para poder visualizar qué se le introduce a la red.

En el caso de este Trabajo Fin de Máster, se ha realizado un programa en Unity que permite visualizar qué movimiento exacto se está introduciendo a la red (en el caso en el que se utiliza la red para clasificación) y qué movimiento genera la red (cuando se utiliza para predicción o generación sintética). En la Figura 8 se muestra un ejemplo.

Figura 8. *Representación de las actividades realizada con el programa de Unity.*

Además, es conocida la existencia de trabajos que utilizan **QNNs** (*Quaternion-based Neural Networks*), como (Gaudet, C. J. y Maida A. S., 2018) que modifican los cálculos de errores, pesos, derivadas… y todo el álgebra interno de la red neuronal para poder trabajar con números hipercomplejos como los cuaterniones, dando una serie de ventajas, como la obtención de manera más sencilla de relaciones internas entre los datos (Parcollet, T. *et al.*, 2019). Sin embargo, esta no es una vía que se ha explorado en este Trabajo Fin de Máster, y los cuaterniones se ven desde el punto de vista de una red neuronal que utiliza números reales, considerando cada elemento del cuaternión como una dimensión diferente (de manera semejante a como se hace en *Computer Vision* con imágenes RGB).

Otro motivo por el que utilizar los cuaterniones en este Trabajo Fin de Máster es el de que se quiere enfocar todo desde el punto de vista de un sistema embebido de tiempo real, y esto supone utilizar los menos datos posibles. Con los cuaterniones se está enviando la información más completa con la menor cantidad de datos. Si bien tanto acelerómetro, como giroscopio y magnetómetro tienen únicamente 3 ejes de datos y los cuaterniones 4, una reconstrucción del movimiento con ellos no sería posible a no ser que se añadiera información sobre la posición del sujeto, mientras que con los cuaterniones no hace falta nada más que los propios cuaterniones, por lo que estamos teniendo una información más completa que con los otros 3 sensores.

2.3 Dataset: REALDISP

Durante la realización del Trabajo Fin de Grado (Sáez Bombín, S., 2018) se llevó a cabo una búsqueda bibliográfica enfocada al reconocimiento de actividades humanas, encontrando una serie de bases de datos o datasets, ampliamente utilizados en los trabajos de este problema y que tienen suficientes datos e información para ser la base de un sistema de Deep Learning.

Se consideró que sería idóneo tener en el dataset muchas actividades distintas, que dichas actividades se asemejaran al caso práctico que se le podría dar a la red resultante del trabajo (juego serio del grupo GTI para rehabilitación de personas) y que proporcionara los datos de forma semejante a como los proporcionan los sensores del grupo GTI, es decir, que además de los datos del acelerómetro, giroscopio y magnetómetro, estuvieran en ese dataset los cuaterniones calculados (Sáez Bombín, S., 2018).

Con todo esto, el dataset que cumplía todos los requisitos era el dataset REALDISP (Baños, O. *et al*., 2012), así que fue el elegido para trabajar en ese Trabajo Fin de Grado, y es con el que se sigue trabajando en este Trabajo Fin de Máster.

El dataset REALDISP es un dataset abierto (Baños, O. *et al*., 2012), disponible en un repositorio de datasets (UCI Machine Learning Repository, n.d) en el que se consideran 33 actividades *fitness* diferentes, grabadas usando 9 sensores inerciales (de marca Xsens, como se ha comentado anteriormente) colocados en 17 participantes.

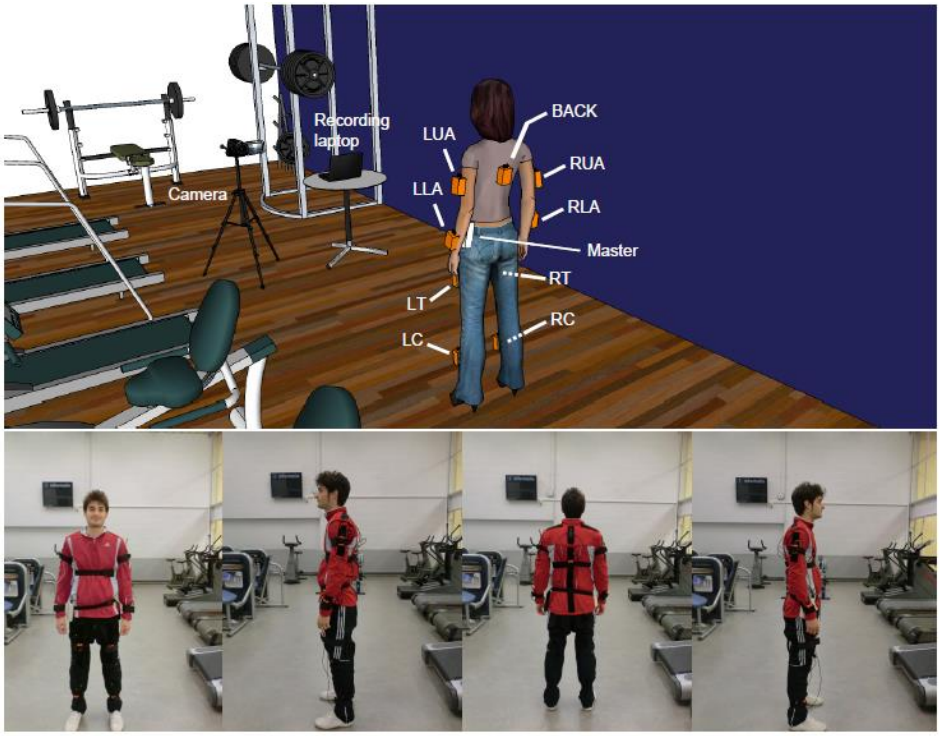


Figura 9. *Disposición de los sensores en el dataset REALDISP.*

El set de actividades del dataset es el mostrado en la Figura 10.

Activity set		
L1: Walking (1 min)	L12: Waist rotation (20x)	L23: Shoulders high amplitude rotation (20x)
L2: Jogging (1 min)	L13: Waist bends (reach foot with opposite hand) (20x)	L24: Shoulders low amplitude rotation (20x)
L3: Running (1 min)	L14: Reach heels backwards (20x)	L25: Arms inner rotation (20x)
L4: Jump up (20x)	L15: Lateral bend (10x to the left + 10x to the right)	L26: Knees (alternatively) to the breast (20x)
L5: Jump front & back (20x)	L16: Lateral bend arm up (10x to the left + 10x to the right)	L27: Heels (alternatively) to the backside (20x)
L6: Jump sideways (20x)	L17: Repetitive forward stretching (20x)	L28: Knees bending (crouching) (20x)
L7: Jump leg/arms open/closed (20x)	L18: Upper trunk and lower body opposite twist (20x)	L29: Knees (alternatively) bend forward (20x)
L8: Jump rope (20x)	L19: Arms lateral elevation (20x)	L30: Rotation on the knees (20x)
L9: Trunk twist (arms outstretched) (20x)	L20: Arms frontal elevation (20x)	L31: Rowing (1 min)
L10: Trunk twist (elbows bended) (20x)	L21: Frontal hand claps (20x)	L32: Elliptic bike (1 min)
L11: Waist bends forward (20x)	L22: Arms frontal crossing (20x)	L33: Cycling (1 min)

Figura 10. *Actividades en el dataset REALDISP.*

A diferencia del Trabajo Fin de Grado y de algunas publicaciones encontradas que utilizan este dataset, no se incluye la actividad L0, correspondiente al periodo de inactividad ("L0: No actividad"), ya que son momentos en los que los sujetos pueden realizar varias actividades (andar, hablar, girarse…) y quedan etiquetadas como una sola.

Además, en el dataset se diferencian tres tipos de datos:

- ***Ideal***: Los datos son recogidos con los sensores colocados idealmente. En (Baños, O. *et al*., 2014a) se indica que estos datos pueden considerarse como el set de entrenamiento para sistemas de reconocimiento de actividades. Este es el único grupo de datos que se utiliza en este Trabajo Fin de Máster.
- ***Self***: Los datos son recogidos con tres de los sensores colocados por el propio sujeto, es decir, sin la supervisión de un profesional, así que no tendrán una colocación ideal. Este escenario intenta simular la variabilidad que puede ocurrir en el uso del día a día en un sistema de reconocimiento de actividades (Baños, O. *et al*., 2014a).
- ***Mutual***: Los datos son recogidos con hasta 7 sensores descolocados (algunos los coloca el sujeto y otros el profesional de manera intencionada) en cuanto a rotaciones y traslaciones con respecto a la colocación ideal (Baños, O. *et al*., 2014a).

Los sensores utilizados en este dataset proporcionan medidas de aceleración, velocidad angular y campo magnético en las 3 dimensiones (X, Y, Z) además de una estimación de la orientación en 4D (los cuaterniones). Por lo tanto, cada sensor da 13 valores a la salida en cada medida, lo que lleva a un set de 117 señales, recogidas a 50 Hz, es decir, cada 0,02 segundos (Sáez Bombín, S., 2018).

De esta forma el dataset consta de 46 ficheros *.log*, separados por sujeto y tipo de datos (*ideal*, *self* o *mutual*) y la forma en la que están distribuidos los datos dentro de cada fichero es en forma de matriz, con tantas filas como muestras se tengan y 120 columnas:

- Columnas 1 y 2: contienen los *timestamps* (tiempos de captura de los sensores) con la parte entera en segundos (en la 1ª columna) y el resto en microsegundos (en la 2ª columna).
- Columnas 3-119: contienen las medidas del acelerómetro (ACC), giroscopio (GYR), campo magnético (MAG) y cuaterniones (QUAT) como se ha indicado anteriormente. En este orden están: ACC (X, Y, Z), GYR (X, Y, Z), MAG (X, Y, Z) y QUAT (Re, i, j, k), es decir, 13 columnas para cada uno de los sensores, los cuales van en el siguiente orden: RLA, RUA, BACK, LUA, LLA, RC, RT, LT, LC.
- Columna 120: contiene las etiquetas de actividad.

Sin embargo, tras una inspección y estudio de los datos a través del programa de Unity anteriormente comentado, se ha podido comprobar una serie de puntos a destacar y que van a modificar el dataset de cara a este Trabajo Fin de Máster:

- Tal y como se indica en (Baños, O. *et al*., 2014a), hay varios ficheros corruptos, como son el "subject6_self.log", "subject13_self.log" y "subject15_mutual4.log". Aunque finalmente solamente se hace uso de los datos *ideal*.
- Hay algunos errores en las medidas de los sensores (especialmente en las piernas) que se pudieron ver a partir de la reconstrucción de los movimientos en Unity. Estos errores fueron comentados con el autor del dataset REALDISP, el Dr. D. Oresti Baños, y se tomó la decisión de prescindir de los sensores de las piernas, reduciendo los datos a los sensores de los brazos y espalda (5 de los 9 iniciales).
- El anterior punto conlleva la reducción de la cantidad de actividades, ya que algunas de ellas son principalmente realizadas con los sensores de las piernas, como andar, correr…
- Del nuevo set reducido de actividades, se prescindió de algunos sujetos que no tenían datos en algunas de ellas.

Finalmente, el dataset considerado consta de 12 sujetos, 11 actividades y los cuaterniones de 5 de los sensores (RUA, RLA, LUA, LLA, BACK), lo que supone la reducción del número de clases considerado, pero también de los datos de los que se dispone.

Este hecho supone un reto, pero va a dar una visión más real de la capacidad del sistema que se realice para su inclusión en un sistema embebido de tiempo real, ya que, idealmente, este requiere un buen rendimiento con pocos datos y con un modelo que no ocupe mucha memoria.

Capítulo 3. Deep Learning

La técnica utilizada en este Trabajo Fin de Máster para la resolución del problema HAR es el Deep Learning. En este capítulo se desglosan los fundamentos teóricos de este campo, analizados previamente al comienzo de la toma de decisiones y el inicio del estudio de las redes.

3.1 Introducción

Para comenzar a hablar del Deep Learning es necesario saber de dónde viene, la Inteligencia Artificial (AI, *Artificial Intelligence*) y el Machine Learning.

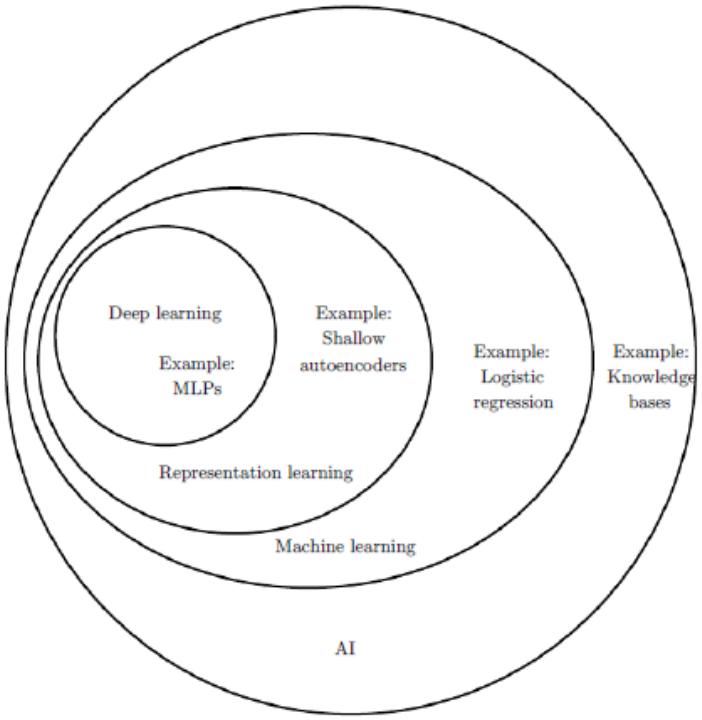


Figura 11. *Representación en de la Inteligencia Artificial.*

En la Figura 11, de (Goodfellow, I. *et al*., 2016), se representa un diagrama de Venn, en el que se puede ver cómo el Deep Learning es un subconjunto del Machine Learning, y, por lo tanto, de la Inteligencia Artificial. Por ello, es necesario hablar de la Inteligencia Artificial para después hacerlo del Deep Learning.

La Inteligencia Artificial es un conjunto de técnicas que permite realizar *software* que aprenda de modelos matemáticos, sin la necesidad de preprogramar reglas que abarquen todas las infinitas combinaciones de posibilidades dentro de un problema (Sáez Bombín, S., 2018).

En los comienzos de la Inteligencia Artificial, su objetivo era la resolución de manera rápida de tareas intelectualmente difíciles para los humanos, pero que pueden describirse mediante una serie de reglas matemáticas, lo que las hacía sencillas para los ordenadores. Sin embargo, el verdadero reto de la Inteligencia Artificial es resolver tareas que son fáciles de realizar para las personas, pero difíciles de describir formalmente por ellas, ya que las resolvemos de forma intuitiva, como puede ser reconocer ciertas palabras cuando las escuchamos o caras e imágenes cuando las vemos (Goodfellow, I. *et al*., 2016).

En nuestros días, la Inteligencia Artificial es un campo próspero con muchas aplicaciones prácticas y temas de investigación activos, como la automatización de labores rutinarias, el entendimiento del lenguaje o imágenes o la realización de diagnósticos en medicina (Goodfellow, I. *et al*., 2016).

La complejidad de estas tareas (una misma tarea puede realizarse de muchas maneras diferentes) hace que las máquinas necesiten extraer patrones de los datos que se les proporcionan mediante el aprendizaje automático, definido como Machine Learning (Bengio, Y. *et al*., 2016, citado en Goodfellow, I. *et al*., 2016).

El Machine Learning hace uso de un conjunto de técnicas y algoritmos que le permiten conseguir sus objetivos. Entre estas técnicas destacan las redes neuronales, que son sistemas artificiales formados por "neuronas", definidas así por inspiración del funcionamiento bilógico del cerebro humano. Las neuronas realizan una operación concreta sobre los datos que pasan por ellas, además, se organizan en capas, habiendo múltiples conexiones entre ellas y una dirección de propagación de los datos por las distintas capas (Sáez Bombín, S., 2018).

El hecho de necesitar extraer patrones, realizar clasificación de los datos y categorizar resultados con una buena precisión requiere una gran cantidad de datos para proporcionarlos a las neuronas de la red y ayudarlas en su cometido (Sáez Bombín, S., 2018).

Del desarrollo de estas redes neuronales, surgió el Deep Learning:

El Deep Learning es una rama del Machine Learning que se caracteriza por utilizar una gran cantidad de datos y por proporcionar una alta adaptabilidad y una estructura y lenguaje comunes para describir los problemas a resolver (Udacity, 2018).

La clave es que permite a los ordenadores entender el mundo en términos de una jerarquía de conceptos, definiéndolos a partir de otros más simples y aprendiendo de la experiencia, lo que evita la necesidad de especificar de forma precisa, por parte de los humanos, todo el conocimiento que el ordenador necesita. La jerarquía de conceptos permite al ordenador aprender conceptos complicados construyéndolos a partir de conceptos más sencillos (Goodfellow, I. *et al*., 2016).

El Deep Learning se ha convertido en la técnica líder del Machine Learning especialmente en campos como *Computer Vision* (reconocimiento de objetos, animales o personas de una imagen o vídeo) y *Speech Recognition* (reconocimiento de lo que dice una persona a partir de la grabación de su voz), gracias al desarrollo y a la utilización de las GPUs, que permiten un procesamiento mucho más rápido (LeCun *et al*., 2015, citado en Goodfellow, I. *et al*., 2016).

Si se dibujara un gráfico de un concepto, construyéndose sobre otros más sencillos como realiza esta técnica, se podría apreciar un gráfico profundo, de muchas capas. Por esto se llama a esta aproximación de Inteligencia Artificial, Deep Learning (Goodfellow, I. *et al*., 2016).

En este Trabajo de Fin de Máster se utiliza el Deep Learning para la resolución del problema HAR como se ha indicado anteriormente:

Reconocer actividades humanas y el contexto en el ocurren a partir de los datos procedentes de un sensor es el núcleo de las tecnologías inteligentes asistidas. El problema HAR está basado en la asunción de que los movimientos del cuerpo se traducen en unos patrones característicos de las señales de los sensores, que pueden ser detectados por los mismos y clasificados utilizando técnicas de Machine Learning, en concreto el Deep Learning (Ordóñez, F. J. y Roggen, D., 2016).

Por ello y por lo indicado en la sección 1.1, en este Trabajo Fin de Máster se utilizan los datos procedentes de la combinación de varios sensores colocados sobre el cuerpo y sobre cuyas señales se aplicarán técnicas de Deep Learning, ya que puede mejorar el rendimiento se mejora sobre las técnicas existentes de reconocimiento y puede tener potencial para descubrir características que están ligadas a la dinámica del movimiento humano (Ordóñez, F. J. y Roggen, D., 2016).

3.2 Historia del Deep Learning

En primer lugar, la historia del Deep Learning comienza con la Inteligencia Artificial y con las primeras ideas de *red neuronal*. Estas pueden remontarse a 1943, cuando Walter Pitts y Warren McCulloch crearon un modelo de ordenador (neurona artificial) basado en las redes neuronales del cerebro humano, utilizando una combinación de algoritmos y matemáticas para imitar el proceso del pensamiento humano (McCulloch, W., S. y Pitts, W., 1943).

Más tarde, en 1957, comenzó el desarrollo del **Perceptron**, la primera red neuronal, de la mano de Frank Rosenblatt (Rosenblatt, F., 1958). Esta red supuso un gran avance ya que podría reconocer letras y números (San José, R., 2019), pero no dejaba de ser muy limitada, por lo que, en los siguientes años, los esfuerzos de la investigación de este campo se centraron la mejora de redes neuronales basadas en el Perceptron pero que utilizaran distintas reglas de aprendizaje. Las limitaciones del Perceptron fueron expuestas por Marvin Minsky y Seymour Papert, al demostrar que esta red neuronal no era capaz de aprender a partir de una función no lineal, un tipo de función muy utilizada (Matich, D. J., 2001, pp. 6-7).

El 1962 se desarrollan las bases de un algoritmo de aprendizaje denominado *backpropagation* (propagación hacia atrás), en un intento de Rosenblatt de generalizar el algoritmo de aprendizaje del Perceptron a múltiples capas en 1962. En realidad, este no fue el único, ya que entre los años 60 y 70, hubo muchos intentos, como por ejemplo el de Seppo Linnainmaa, en 1970, que escribió su tesis incluyendo un código FORTRAN para la *backpropagation*, pero en una versión básica. Por otro lado, los primeros esfuerzos de desarrollo de algoritmos de Deep Learning vinieron de Alexey Grigoryevich Ivakhnenko (Ivakhnenko, A. G. e Ivakhnenko, G. A., 1995) junto a Valentin Grigorevich Lapa en 1967.

El desarrollo de la Inteligencia Artificial y, por tanto, del Deep Learning, ha tenido muchos ciclos de crecimiento y estancamiento debido a no cumplir con las expectativas que se tenían de ello. Los ciclos de estancamiento se conocen como *AI Winters* y el primero de ellos (y más largo) comenzó en 1969, cuando Marvin Minsky y Seymor Papert escribieron un libro sobre las limitaciones insalvables del Perceptron (y todos sus semejantes) provocando un frenazo en la investigación de las redes neuronales (San José, R., 2019).

Sin embargo, esto no implica que se abandonara por completo la investigación en este campo, de hecho, la versión moderna y que tuvo éxito del algoritmo *backpropagation*, que parece tener tres inventores distintos, fue desarrollada por Paul Werbos en 1974 en el título *"Beyond Regression"*, aunque no fue hasta 1982 cuando David Parker y David Rumelhart la desarrollaron a la vez. Esto dio pie a la publicación del *paper" Parallel distributed processing: Explorations in the microstructure of cognition"* por Rumelhart, Hinton y Williams en 1986, mostrando sus aplicaciones en las redes neuronales y la Inteligencia Artificial, y un gran número de investigadores se interesaron por este algoritmo, poniendo fin al *AI Winter* más largo (Chauvin, Y. y Rumelhart, D. E., 1995, pp 1-2).

Posteriormente, durante los años 80 se realizaron varios trabajos muy importantes para las redes neuronales (Udacity, 2018). Las primeras redes neuronales convolucionales fueron usadas por Kunihiko Fukushima y en 1982 se culminó el desarrollo de una red neuronal artificial, llamada **Neocognitron** (Fukushima, K. y Miyake, S., 1982), la cual utilizaba un diseño multicapa jerárquico, que permitía al ordenador "aprender" a reconocer patrones visuales.

Ya en 1989, Yann LeCun proporcionó la primera demostración práctica de la *backpropagation* en los Laboratorios Bell (LeCun, Y. *et al*., 1989) y posteriormente crearía la red LeNet 5, en la que combinó redes neuronales convolucionales con la *backpropagation* para clasificar dígitos escritos a mano (el conocido MNIST).

En 1995, Corinna Cortes y Vladimir Vapnik desarrollaron el ***Support Vector Machine*** (SVM) (Cortes, C. y Vapnik, V., 1995) y en 1997 se desarrollaron las **celdas LSTM** (*Long short-term memory*) para las redes neuronales recurrentes, por Sepp Hochreiter y Juergen Scmidhuber (Hochreiter, S. y Schmidhuber, J., 1997).

A pesar de estos grandes avances, "en los años 90, los ordenadores eran muy lentos, y los datasets eran muy pequeños, además de que la investigación no conseguía encontrar demasiadas aplicaciones en el mundo real" (Udacity, 2018). Por ello, a comienzos de los 2000, las redes neuronales no eran una opción viable ni interesante para el mundo del Machine Learning (Udacity, 2018), dando pie a otro pequeño *AI Winter*.

Este *AI Winter* duró apenas unos pocos años, ya que poco después del comienzo del siglo XXI se comenzó a tener más datos, los ordenadores empezaron a ser más rápidos en el procesamiento de datos y las GPUs se desarrollaron para comenzar a procesar datos no relacionados con los gráficos (Buck, I., 2010). Además, en 2006 hubo un *rebranding* y comenzó a conocerse como **Deep Learning** a todo algoritmo que utiliza redes neuronales (San José, R., 2019). Gracias a estos avances, las redes neuronales compitieron con el *Support Vector Machine*, ya que, a pesar de ser más lentas, ofrecían mejores resultados usando los mismos datos, además de ser capaces de mejorar cuantos más datos se utilicen.

Ha sido en los últimos años en los que se han tenido más avances en las redes neuronales. En 2009, las redes neuronales irrumpieron con fuerza en el campo de la *Speech Recognition*, y un poco más tarde, en 2012, lo hicieron en el campo de la *Computer Vision*. Ya en 2014, lo hicieron en el campo de *Machine Translation*, convirtiéndose, en los tres casos, en la técnica líder para la resolución de esos problemas (Udacity, 2018). A esta época, y hasta nuestros días se la conoce como *Great AI Awakening* (el gran despertar de la IA) (San José, R., 2019).

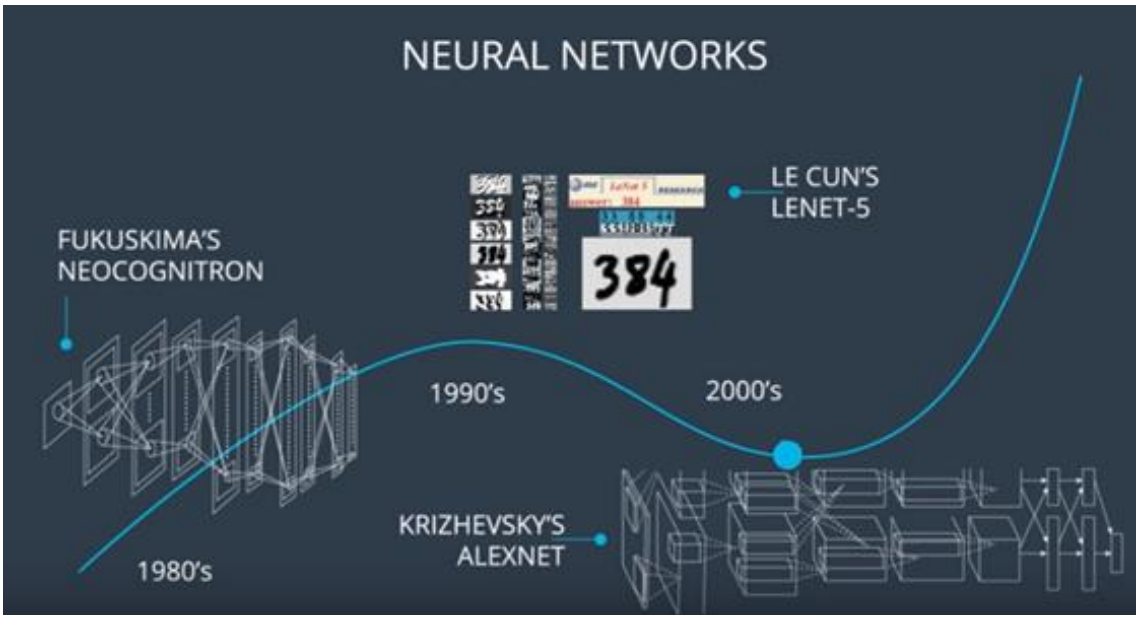


Figura 12. *Línea temporal del Deep Learning.*

En la Figura 12 de (Udacity, 2018), se muestran varios eventos importantes de la historia más actual del Deep Learning, como el **Neocognitron** de Fukushima, la **LeNet-5** de LeCun o la **AlexNet** de Krizhevsky (Krizhevsky, A., *et al*., 2012), que ganó varias competiciones con una red convolucional utilizando múltiples GPUs.

Como complemento a la anterior figura, se añade la Figura 13 (San José, R., 2019), en la que se muestran varios hitos desde los inicios del Deep Learning.

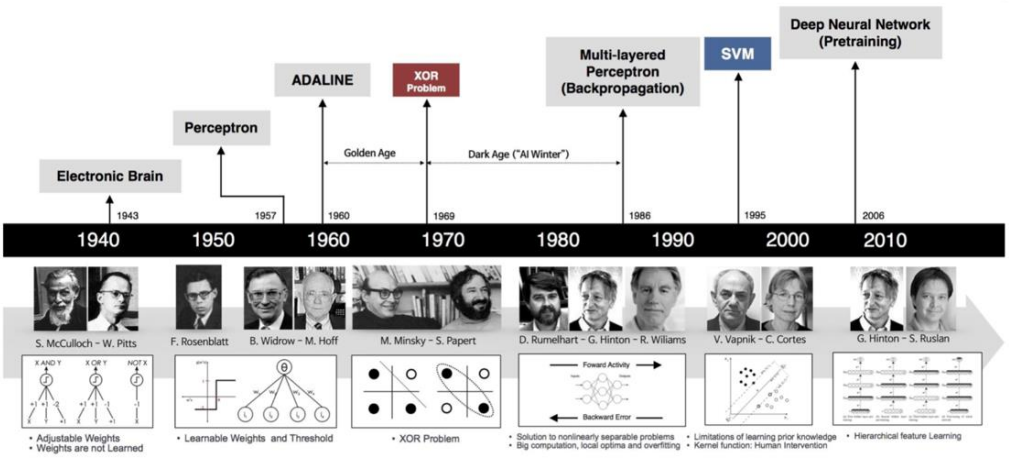


Figura 13. *Visión más detallada de la línea temporal del Deep Learning.*

Actualmente, tanto el Big Data como la evolución de la Inteligencia Artificial dependen del desarrollo del Deep Learning, cada vez más apoyado y cada vez con mayor interés (Udacity, 2018).

3.3 Teoría

A lo largo de esta sección se explica la teoría relacionada con este Trabajo Fin de Máster. Esta explicación se realiza desde los modelos y conceptos más sencillos, hasta los más complejos (los utilizados en este Trabajo Fin de Máster) y su aplicación.

3.3.1 Conceptos básicos: de Machine Learning a Deep Learning

Como se ha indicado anteriormente, el Deep Learning es un campo del Machine Learning, por lo que, para entenderlo, es necesario tener claros los conceptos básicos del Machine Learning.

Un algoritmo de aprendizaje es aquel que es capaz de aprender de los datos, y tal y como se indica en (Goodfellow, I. *et al*., 2016 citando a Mitchell, 1997): "Se dice que un programa de ordenador aprende de la experiencia ***E*** con respecto a alguna clase de objetivos ***T*** y una medida de rendimiento ***P***, si su rendimiento en los objetivos *T*, medido por *P*, mejora con la experiencia *E*." Con esto se refiere a:

- Objetivos (***T***): Los objetivos habitualmente son aquellos que son demasiado difíciles de resolver a través de programas fijos escritos y diseñados por seres humanos (Goodfellow, I. *et al*., 2016). Algunos de los objetivos típicos (aunque no únicos) de los algoritmos de aprendizaje de Machine Learning son: clasificación, regresión, transcripción, traducción, estimación de densidades de probabilidad…
- Medida de rendimiento (***P***): Esta es una media cuantitativa, habitualmente específica del objetivo que esté llevando a cabo el sistema. Por ejemplo, un objetivo de clasificación usará como medida la precisión del modelo (proporción de ejemplos para los que la salida es correcta) (Goodfellow, I. *et al*., 2016).
- Experiencia (***E***): Se refiere a qué datos y en qué forma ve esos datos el algoritmo durante el proceso de entrenamiento (Goodfellow, I. *et al*., 2016). En este sentido, los algoritmos de Machine Learning se diferencian en 4 tipos de aprendizaje:
    - Supervisado (***Supervised Learning***): Mediante el uso de etiquetas (***labels***), se indica a la máquina qué es cada dato.
    - No supervisado (***Unsupervised Learning***): En este tipo de aprendizaje no se utilizan las etiquetas, es la máquina la que debe extraer y aprender la estructura del conjunto de datos (habitualmente, la distribución de probabilidad completa de la que vienen esos datos) (Goodfellow, I. *et al*., 2016).
    - Pseudo-supervisado (***Self-supervised Learning***): Es un tipo de aprendizaje en el que se mezclan datos etiquetados por humanos con datos que etiqueta la propia máquina, una vez que esta ha aprendido la estructura del conjunto de datos, mediante un aprendizaje no supervisado, mejorando el rendimiento (San José, R., 2019).
    - Reforzado (***Reinforcement Learning***): Se basa en la idea de causa-efecto, es decir, la máquina aprende en base a su interacción con los datos (Sutton, R. S. y Barto, A. G., 2014).

En todos los casos, estos algoritmos de aprendizaje lo que realizan es una ***feature extraction*** o extracción de características, es decir, consiguen sacar aquellas características importantes, que tengan un "significado" y saquen información de los datos. En muchos casos, las salidas de un algoritmo de Machine Learning se utilizan como entradas para otro, con el fin de mejorar el entrenamiento, aprendizaje y entendimiento (Sáez Bombín, S., 2018).

Otro elemento básico de Machine Learning, y por lo tanto de Deep Learning es la **capacidad** del modelo. El reto principal de los algoritmos de aprendizaje es que deben tener un buen rendimiento sobre datos nuevos, que no hayan visto previamente, y si lo consigue, se dice que es capaz de **generalizar** (Goodfellow, I. *et al*., 2016). Por ello, en Machine Learning, los datos de los que se dispone se separan en 3 sets de datos:

- **Set de entrenamiento**: Contiene la mayor parte de los datos. Se utiliza para que el modelo o algoritmo aprenda y generalice (Sáez Bombín, S., 2018).
- **Set de validación**: Permite comprobar el rendimiento del entrenamiento. Este set se extrae de los datos de entrenamiento y se utilizan para evaluar el modelo cada cierto número de iteraciones de entrenamiento (Sáez Bombín, S., 2018).
- **Set de test**: Este set debe ser totalmente disjunto a los otros 2. Es el que nos permite saber si el modelo es capaz de generalizar o no, ya que son datos que no ha visto previamente, es decir, nos indica el rendimiento real del modelo (Sáez Bombín, S., 2018).

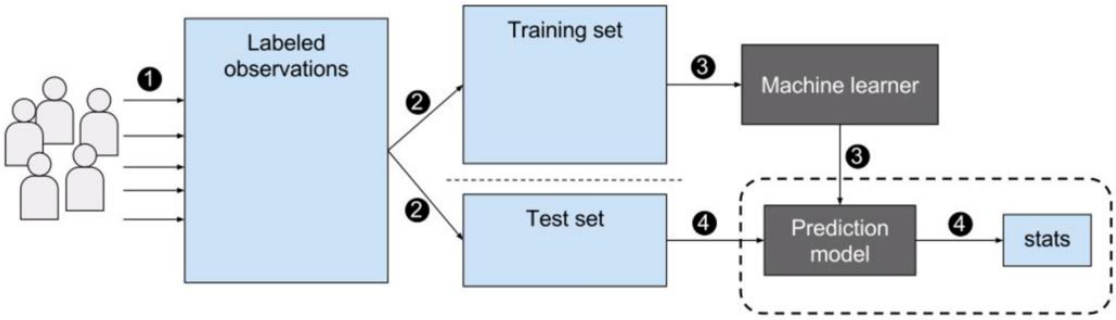


Figura 14. *Ejemplo de separación de los datos (San José, R., 2019).*

El test de validación no siempre se utiliza, pero permite detectar durante el entrenamiento dos fenómenos que nos hacen indicar que no está yendo bien:

- ***Underfitting***: Se produce cuando el modelo tiene poca capacidad y no es capaz de aprender nada de los datos de entrenamiento. En el caso de estar en una clasificación esto se traduce en que el modelo tiene poca precisión tanto en el set de entrenamiento como en el de test.
- ***Overfitting***: Este problema se produce cuando la capacidad del modelo es demasiado elevada comparada con el número de datos de los que se dispone, y lleva al modelo a memorizar las propiedades del set de entrenamiento, siendo incapaz de generalizar (Udacity, 2018). Siguiendo con el ejemplo de clasificación esto se traduce en una alta precisión en el set de entrenamiento, pero baja en el de test. En este caso, el test de validación permite detectarlo si la pérdida en entrenamiento va bajando, pero la de validación es cada vez mayor, tal y como se muestra en la Figura 15.

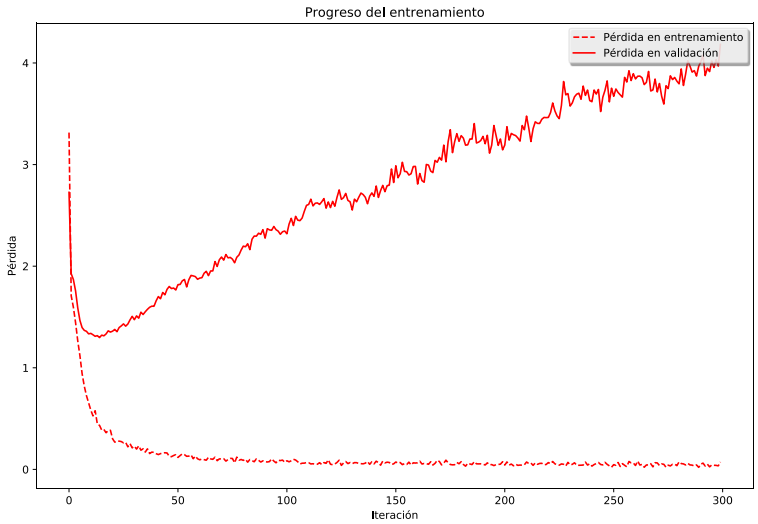


Figura 15. *Ejemplo de un caso de overfitting.*

Por lo tanto, en este Trabajo Fin de Máster se tienen 2 objetivos (***T***), uno de ellos es la clasificación del conjunto de datos, y el segundo de ellos, es la generación de nuevos datos de manera sintética (enfocado desde dos puntos de vista), que se realizan bajo una experiencia (***E***) de *Supervised Learning* y donde se utilizan distintas medidas de rendimiento (***P***) intentando no caer en problemas como el *underfitting* o el *overfitting*.

Clasificación

En este caso, el objetivo del aprendizaje es que la máquina sea capaz de clasificar a partir de un aprendizaje supervisado. A partir de esta clasificación podría realizar otra serie de actividades, como puede ser la detección (para saber si hay peatones cruzando un paso de cebra con un clasificador binario) o el "ranking" (para crear una lista de preferencias, por ejemplo, de páginas web, mediante la clasificación del par petición-página web) (Sáez Bombín, S., 2018).

El clasificador más sencillo que puede haber es un **clasificador lineal**. En él, se realiza un ajuste de pesos y sesgos, a partir del cual la máquina es capaz de distinguir qué es cada dato (Sáez Bombín, S., 2018). Este clasificador sigue una función lineal tan sencilla como la mostrada en (22).

$$WX + b = Y \tag{22}$$

Donde W es la matriz de pesos, b es un vector de sesgos (estos son los parámetros que se entrenan para después realizar una predicción correcta) y X son los datos de entrada (Udacity, 2018).

La salida Y, será un vector de números (denominados ***scores***, es decir, resultados o ***logits***, en el ámbito de la regresión lineal), cada uno correspondiente a la posibilidad de que se tenga un resultado en concreto, y se deberá convertir a unas probabilidades entre 0 y 1 a través de la función conocida como ***softmax*** (23).

$$S(y_i) = \frac{e^{y_i}}{\sum_j y_j} \tag{23}$$

Por lo tanto, para cada posible resultado se obtiene una probabilidad, es decir, como salida tenemos un vector de probabilidades. La máquina tomará como predicción aquella que tenga la probabilidad más alta y el objetivo es que esa salida coincida con lo que son realmente los datos, es decir, con la etiqueta (Udacity, 2018).

Este vector de probabilidades *Y* se compara con el vector de etiquetas que le correspondería a esos datos (aprendizaje supervisado), expresado en el formato ***One-hot encoding***, que expresa las etiquetas o *labels*, que corresponden a cada una de las clases como un vector en el que todos los elementos son 0, excepto el que se corresponde con la clase correcta, que está expresado como un 1.

Para comparar los vectores se utiliza la entropía cruzada, que permite calcular la distancia entre ambos. La ecuación de la entropía cruzada se muestra en (24).

$$D(S, L) = -\sum_i L_i \, log(S_i) \tag{24}$$

Siendo *S* el vector de probabilidades (la salida de la función *softmax*) y *L* el vector de las etiquetas.

Con lo que, el proceso de clasificación lineal consistiría en el siguiente proceso: la entrada *X* es convertida a unos *logits Y* mediante el uso de un modelo lineal (matriz multiplicadora *W* y sesgos *b*) (Sáez Bombín, S., 2018). Tras este proceso, se utiliza la función *softmax* para obtener las probabilidades de los *logits* o *scores* (*S(Y)*), y se comparan esas probabilidades con el vector de etiquetas (*L*) mediante la entropía cruzada (*D(S, L)*) (Udacity, 2018).

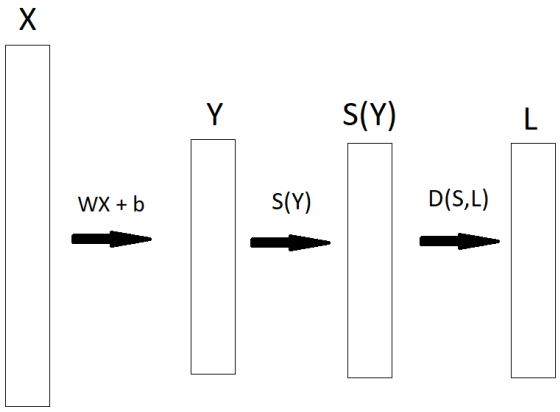


Figura 16. *Modelo de clasificación lineal.*

Lo que se intenta es minimizar la entropía cruzada, es decir, la distancia entre *S(Y)* y *L* (conocido como error, pérdida o coste), por lo que requerimos de una **optimización** del error calculado. En este proceso de optimización se busca que la distancia del vector de probabilidades sea pequeña para la clase correcta y muy grande para la clase incorrecta (Sáez Bombín, S., 2018). De esta forma tenemos la ecuación (25).

$$\varepsilon = \frac{1}{N}\sum_i D(S(wx_i + b), L_i) \tag{25}$$

Es decir, la función error total es el promedio de la entropía cruzada a través de todos los ejemplos del set de datos. Así que, el problema de optimización se resuelve buscando los pesos que minimicen esta función de error. Y la forma más sencilla de realizarlo es utilizando la derivada, que indica la dirección de máxima variación, por lo que si se toma el sentido opuesto (la derivada cambiada de signo), se está yendo en la dirección y sentido hasta alcanzar el mínimo. Por lo tanto, para cada iteración en el entrenamiento, los pesos y sesgos se actualizan como se muestra en (26) y (27) respectivamente.

$$w \leftarrow w - \alpha \Delta w \mathcal{E} \tag{26}$$

$$b \leftarrow b - \alpha \Delta b \mathcal{E} \tag{27}$$

$$\alpha = \boldsymbol{learning\ rate}\ (tasa\ de\ aprendizaje)$$

Este procedimiento o algoritmo de medida y optimización del error se denomina ***Gradient Descent***, sin embargo, tiene una gran limitación, el escalado, ya que al aumentar el número de parámetros a ajustar en el modelo (es decir, el número de datos en nuestro set de entrenamiento), aumenta en gran medida el número de operaciones a realizar debido a que este algoritmo utiliza todos los datos de los que se dispone, haciendo que no sea eficiente del todo. Por ello, el algoritmo más conocido y que supera esta limitación es el ***Stochastic Gradient Descent*** (Udacity, 2018), que es el optimizador (o variaciones sobre él) más utilizado en todos los problemas de Deep Learning, dándole un gran impulso (Goodfellow, I. *et al*., 2016).

Como su propio nombre indica es un *Gradient Descent* estocástico, en el que, en vez de calcular el error con todos los datos, lo estima a partir de unos pocos, un conjunto denominado *batch* o *minibatch*, y que representa un pequeño grupo de ejemplos (de 1 a unos cientos). Así que estamos ajustando el modelo a un set de datos de, por ejemplo, millones de datos, utilizando actualizaciones de los pesos calculadas sobre unos cientos de datos, reduciendo así la carga computacional (Goodfellow, I. *et al*., 2016). Obviamente esta estimación no va a ser buena comparada con lo que podríamos conseguir utilizando todos los datos, pero se toma como buena, llevando a realizar numerosos, pero pequeños y sencillos pasos en el descenso del gradiente en vez de realizar pocos que nos lleven a tener que realizar operaciones muy costosas (Udacity, 2018).

En este Trabajo Fin de Máster se han utilizado dos variaciones del *Stochastic Gradient Descent*, como son el ***RMSProp*** *(Root Mean Square Propagation*) y el ***Adam*** (*Adaptive moment estimation*), para fines distintos (para clasificación se ha utilizado únicamente el *Adam*).

Este último algoritmo es una extensión del *Stochastic Gradient Descent* en la que, a diferencia de éste, mantiene una *learning rate* diferente para cada parámetro de la red (no una única fija para todos), adaptándolos de manera individual mientras se va desarrollando el aprendizaje de la red (Kingma, D. P. y Lei Ba, M., 2015).

Específicamente, "el algoritmo calcula un movimiento promedio exponencial del gradiente y de su cuadrado, con los hiperparámetros β_1 y β_2 controlando las tasas de decaimiento de estos promedios de movimiento" (Kingma, D. P. y Lei Ba, M., 2015).

Este algoritmo combina las ventajas de otras dos extensiones del *Stochastic Gradient Descent*, como son AdaGrad y RMSProp (ya mencionado):

- *Adaptive Gradient Algorithm* (AdaGrad): este algoritmo funciona bien para gradientes dispersos (Duchi, J. *et al*., 2011, citado en Kingma, D. P. y Lei Ba, M., 2015). Se correspondería con un algoritmo Adam de parámetros $\beta_1 = 0$, β_2 infinitesimal y otra *learning rate* α modificada (Kingma, D. P. y Lei Ba, M., 2015).
- *Root Mean Square Propagation* (RMSProp): el algoritmo realiza las actualizaciones de sus parámetros usando un momento del gradiente, mientras que Adam utiliza los momentos de primer y segundo orden (Kingma, D. P. y Lei Ba, M., 2015).

A continuación, se muestra una comparativa de los costes de entrenamiento de este algoritmo con otros realizada por sus autores, Figura 17.

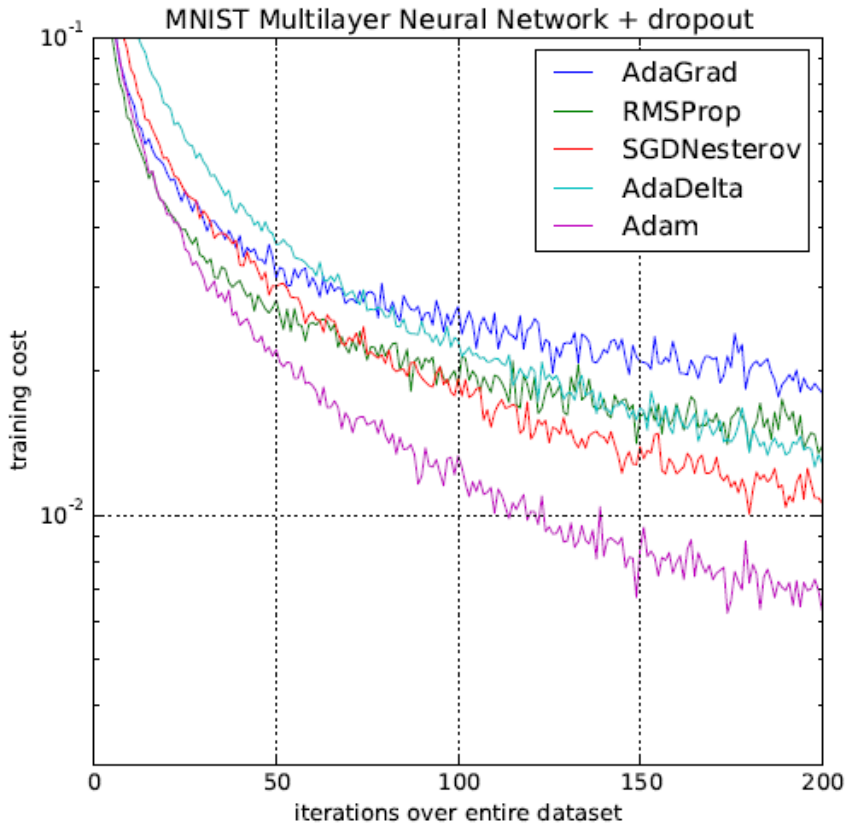


Figura 17. *Comparativa del optimizador Adam con otros optimizadores conocidos.*

Como se puede observar en esta Figura 17 (Kingma, D. P. y Lei Ba, M., 2015) el algoritmo *Adam* es el que da un coste más bajo durante el entrenamiento.

Generación

Este objetivo se afronta desde dos puntos de vista diferentes:

1. En el primero de ellos, se intenta conseguir que el modelo sea capaz de generar nuevos datos semejantes (con el mismo significado) que los de entrenamiento a partir de números aleatorios.
2. Y en el segundo de ellos lo que se hace es intentar que el modelo encuentre la forma de predecir datos nuevos a partir de una secuencia de datos dada.

A estos dos métodos se les puede atribuir todo lo relacionado con la clasificación comentado previamente, especialmente al primero de ellos. Con respecto al segundo, hay varias diferencias, por ejemplo, no se utilizan *labels*, sino que lo que se le proporciona al modelo durante el entrenamiento es una secuencia de datos en el instante de tiempo *t* y la secuencia de datos en el instante de tiempo *t+1* para que cuando se realice el test, se den como respuesta los datos en instantes temporales posteriores a la secuencia que se le introduzca.

Por lo tanto, no se compara con un vector de probabilidades de ciertas clases como antes, sino que se compara con una matriz de datos, así que el error se va a calcular de manera distinta a como se hacía anteriormente. En este caso se utiliza el *Mean Square Error **(MSE)*** en vez de la entropía cruzada, y su ecuación es (28).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)^2 \tag{28}$$

Donde *n* es el número de datos, Y_i es el dato real *i*, $\widehat{Y}_i$ es la predicción *i*. La utilización de esta función de error lleva a utilizar otro optimizador que sea más efectivo, como es el ***RMSProp***.

En Deep Learning, el modelo al que se hace referencia a lo largo de esta sección son las Redes Neuronales Profundas (*Deep Neural Networks*) que se explican a lo largo de la siguiente sección.

3.3.2 Redes Neuronales Profundas (DNN)

Tal y como se hace en (Goodfellow, I. *et al*., 2016) se van a explicar las redes neuronales a partir de su versión más básica, las ***Feedforward Neural Networks***, también conocidas como Perceptron multicapa (**MLP**, *multi-layer Perceptron*), que suponen la extensión al concepto de "red neuronal" del modelo lineal del apartado anterior. Las redes neuronales están formadas por **nodos**, agrupados en **capas**, en las que cada nodo de la red está asociado con el resto de los nodos mediante conexiones con **distintos pesos** (Sáez Bombín, S., 2018). Estos nodos transforman los datos que pasan por ellos mediante operaciones no lineales para crear un umbral de decisión para la entrada, proyectándolo en un espacio donde se convierte linealmente separable (Goodfellow, I. *et al*., 2016).

Se denominan ***redes*** porque normalmente se representan como una composición de varias funciones que sigue una estructura en cadena, cuya longitud determina la ***profundidad*** del modelo (de esta terminología viene Deep Learning) y se puede ver como la función (29).

$$f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))) \tag{29}$$

Como se ha indicado en el apartado anterior, la salida del modelo se espera que se parezca lo más posible a lo que estamos buscando, bien sea una clasificación, regresión…, es decir, los ejemplos del entrenamiento indican lo que la capa de salida de la red debe hacer, que es producir un valor cercano a lo que debería ser. El comportamiento del resto de las capas de la red no lo determinan los ejemplos de entrenamiento, sino que es el propio algoritmo de aprendizaje el que determina cómo utilizar esas capas para obtener el resultado deseado. Como no se muestra el resultado de las capas intermedias (aunque hoy en día sí es posible, o al menos saber más información acerca de ellas), estas capas se denominan capas escondidas (***hidden layers***) (Goodfellow, I. *et al*., 2016).

Estas redes se denominan ***neuronales*** porque están inspiradas por la neurociencia: cada capa escondida obtiene valores en forma de vector y la dimensión de estos vectores determina la *anchura* del modelo, pudiéndose interpretar que cada elemento del vector está realizando un papel similar al de una neurona (Goodfellow, I. *et al*., 2016).

Es por ello, que se puede ver cada capa como un conjunto de nodos denominados ***units*** que actúan en paralelo, pareciéndose cada una de estas *units* a una neurona, en el sentido de que reciben a la entrada datos de otras muchas *units* y calculan su propio **valor de activación**, es decir, en el caso de las *Feedforward Neural Networks*, primero se pasará la entrada por el modelo lineal, y al resultado, se le aplicará la **función de activación** (no lineal) de la *unit* (Sáez Bombín, S., 2018).

Esta función de activación o ***activation*** a secas introduce no linealidades. De esta forma, los parámetros de cada capa están dentro de un modelo lineal, con las ventajas que ello lleva, como es la eficiencia (uso de GPUs) y estabilidad (pequeños cambios en la entrada producen pequeños cambios en la salida) (Udacity, 2018), mientras que la capacidad del modelo no se ve limitada por las funciones lineales y gracias a las *activations* es capaz de entender la interacción entre **cualquier** par de variables de entrada (no solamente de aquellas relacionadas linealmente) (Goodfellow, I. *et al*., 2016).

Este valor de activación dependerá de la *unit* o *hidden unit* (al pertenecer a las *hidden layers*) que utilicemos. Hay varios tipos dependiendo de la función de activación, y no es posible predecir cuál de ellos funcionará mejor. Se comentan solamente los utilizados en este Trabajo Fin de Máster:

- **ReLU** (*Rectified Linear Unit*): Es la opción por defecto típica. Este tipo de *unit* utiliza la función de activación (30).

$$g(z) = \max\{0, z\} \tag{30}$$

  Las ReLUs son fáciles de optimizar porque son muy similares a una *unit* lineal y no introducen efectos de segundo orden: la primera derivada de esta *unit* es 1 para

todos los valores en los que la *unit* está activa y la segunda derivada de esta *unit* es 0 para todos los valores (Goodfellow, I. *et al*., 2016).
En un modelo lineal, como el visto en el anterior apartado, sería la ecuación (31).

$$h = g(WX + b) \tag{31}$$

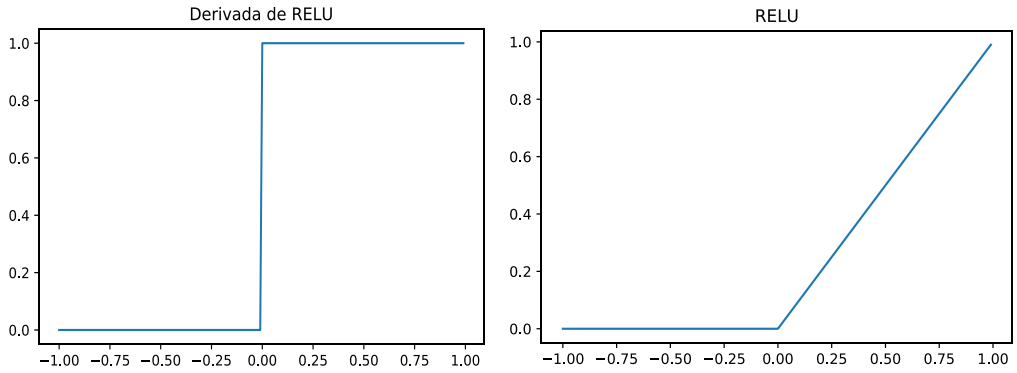


Figura 18. *Función de activación de una ReLU y su derivada.*

- **Leaky ReLU**: Igual que la ReLU pero con la diferencia de que cuando $x<0$ su pendiente en vez de ser *0*, es $0.01x$.

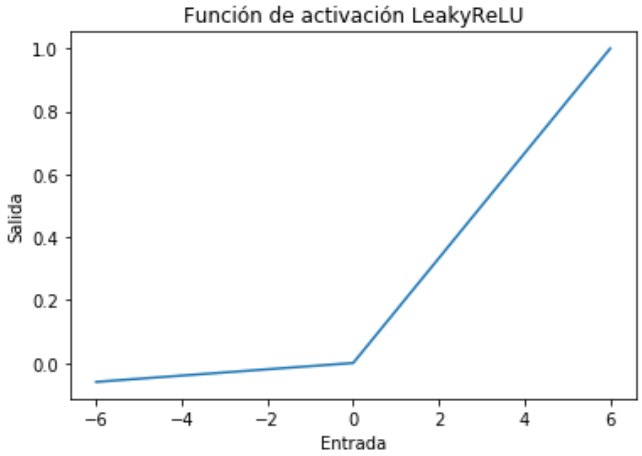


Figura 19. *Función de activación LeakyReLU.*

- **Función sigmoide** (*Logistic Sigmoid*): Previas a la introducción de las ReLUs. Su función de activación (sigmoide estándar) es (32).

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{32}$$

Estas *units* tienen el problema de que la función toma un valor muy alto para z muy positivas y muy bajo para z muy negativas, lo que su rango de sensibilidad es muy pequeño, donde z es cercano a 0 (Sáez Bombín, S., 2018). Esto provoca que el aprendizaje basado en el gradiente sea más difícil. Sin embargo, las redes

neuronales recurrentes que se verán más adelante tienen unos requisitos que hacen que estas *units* sean mejor opción que las ReLUs (Goodfellow, I. *et al.*, 2016).

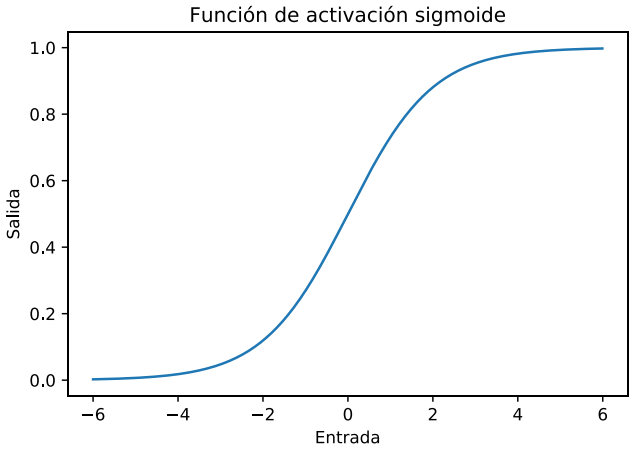


Figura 20. *Función de activación sigmoide.*

- **Función tangente hiperbólica** (*Hyperbolic Tangent*): La función de activación de esta *unit* es la mostrada en la ecuación (33).

$$g(z) = tanh(z) \tag{33}$$

Y está relacionada con la sigmoide, que es una función tangente hiperbólica escalada y con offset, mostrada en la ecuación (34).

$$\tanh(z) = 2\sigma(2z) - 1 \tag{34}$$

Esta relación hace que en los casos en los que se pueda utilizar la función sigmoide, la función de activación tangente hiperbólica rinde mejor (Sáez Bombín, S., 2018). Esto ocurre debido a que tanh(0) = 0 mientras que $\sigma(0)=\frac{1}{2}$, lo que hace que el entrenamiento de una red neuronal profunda se parezca al entrenamiento de un modelo lineal haciéndolo más sencillo (Goodfellow, I. *et al.*, 2016).

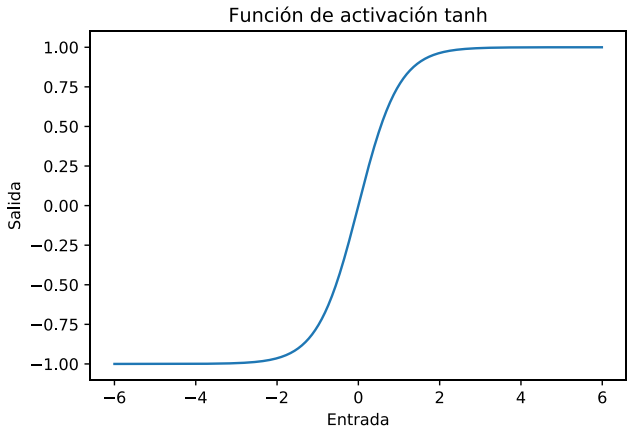


Figura 21. *Función de activación de tangente hiperbólica.*

Una vez que cada *unit* tiene su resultado calculado y los costes obtenidos, se realiza lo que se conoce como la ***back-propagation***, es decir, la propagación hacia las capas anteriores de esos resultados y costes con el objetivo de que cada *unit* ajuste los parámetros en busca de un mejor resultado (error o coste mínimo). Así, durante el entrenamiento, la red va modificando el valor de sus parámetros hasta encontrar la configuración que consiga el resultado óptimo (Udacity, 2018).

Otro aspecto importante en las redes neuronales es la **regularización** (Udacity, 2018). La regularización es una ayuda para la optimización de la red, que va a permitir a esta rendir bien no solamente en los datos de entrenamiento, sino en datos nuevos, por lo tanto, permite reducir el error en el test (a expensas de aumentarlo en el entrenamiento) (Goodfellow, I. *et al*., 2016).

Una red neuronal mejora su eficiencia cuantos más datos se tengan, pero esto a su vez hace aumentar el número de parámetros de la red, pero a través de la regularización se puede reducir el número de parámetros libres, sin aumentar la complejidad de la red, ayudando a evitar en gran medida problemas como el *overfitting*. Hay varios tipos o técnicas de regularización, y a continuación se presentan algunas de ellas:

- **Early stopping**: Esta técnica consiste en parar el entrenamiento cuando se detecta que el error en validación no ha mejorado el mínimo alcanzado durante varias épocas seguidas, quedándose con el modelo (pesos y sesgos de la red actualizados) que mejor precisión ha alcanzado hasta dicho (Sáez Bombín, S., 2018).

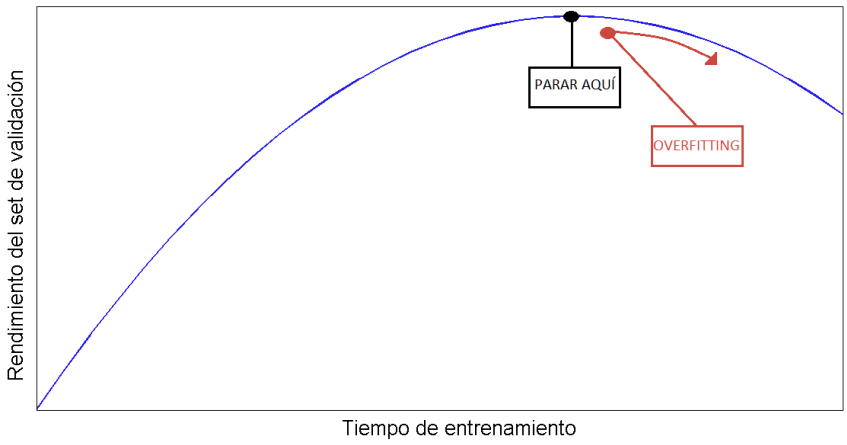


Figura 22. *Técnica de Early Stopping.*

- **Regularización L2:** Consiste en añadir una penalización a la función de coste, obteniendo la función (35).

$$\varepsilon' = \varepsilon + \beta \frac{1}{2} \|w\|_2^2 \tag{35}$$

  Es una regularización simple, ya que solamente se añade la penalización a la función de coste, sin cambiar la estructura de la red.

Esta estrategia también es conocida como *weight decay*, *ridge regression* o *Tikhonov regularitazion* (Goodfellow, I. *et al*., 2016).

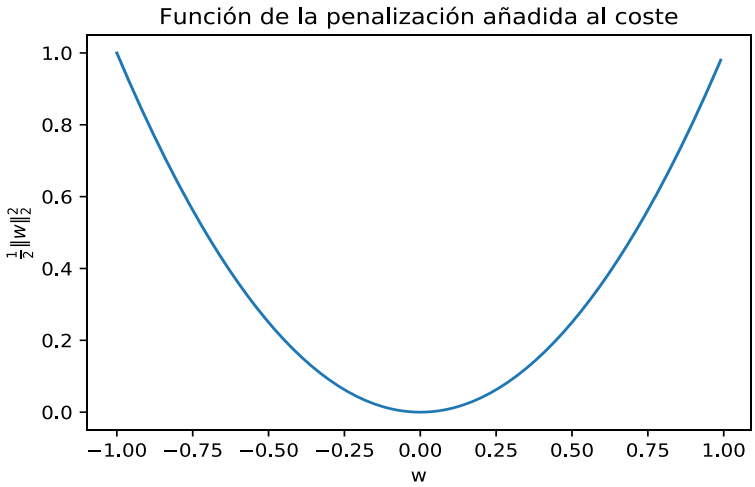


Figura 23. *Función de penalización en la regularización L2.*

- **Dropout**: En esta técnica se ponen a cero varias *activations* (el porcentaje que se elija) de manera aleatoria con el fin de tener un resultado como consenso de varias redes si consideramos cada uno de los resultados obtenidos como el resultado de una red independiente. De esta forma se tiene un aprendizaje redundante y robusto y una mejora del rendimiento (Udacity, 2018).
- **Data augmentation**: Permite la creación de más datos "falsos" para aumentar la cantidad general de datos. Esta es una técnica muy utilizada en clasificación ya que sirve simplemente añadir algo de ruido o realizar algunas transformaciones sobre los datos e introducirlos como nuevos, ayudando a la red a generalizar (Goodfellow, I. *et al*., 2016).
- **Adaptive Learning Rate**: Esta técnica permite modificar el valor de la *learning rate* durante el entrenamiento de la forma deseada y atendiendo a las medidas que se consideren oportunas.
- **Batch normalization**: Esta es una técnica de regularización y optimización. Realiza un reparametrizado adaptativo con el objetivo de mejorar el entrenamiento de las redes. Lo que realiza es normalizar la salida de las *activations* de la capa anterior, es decir, un *batch* completo, restándole su media y dividiéndole por su desviación típica (Goodfellow, I. *et al*., 2016).

Todas estas técnicas de regularización son controladas en el momento del entrenamiento y decididas por el diseñador de la red en base a su experiencia. De cara a entrenar una red, es importante saber con qué datos se está trabajando y la información relevante de estos para así facilitar el aprendizaje de la red y mejorar su rendimiento. De esta forma, se utilizan distintos tipos de redes en función de lo que se quiera aprender, entre las que se destacan las redes convolucionales, redes recurrentes, GANs y *Autoencoders*, que son los tipos de redes con los que se han realizado pruebas en este Trabajo Fin de Máster.

Redes Convolucionales (CNN)

Una red convolucional es simplemente una red neuronal que utiliza la convolución en lugar de la habitual multiplicación de matrices en al menos una de sus capas (Goodfellow, I. *et al*., 2016).

A este tipo de redes se les puede considerar el corazón o núcleo del Deep Learning, convirtiéndose en las redes más populares en investigación (San José, R., 2019).

En primer lugar, se explica la operación de la **convolución** (Goodfellow, I. *et al*., 2016). En su forma más general, una convolución es una operación que realiza un promediado de una función de pesos en cada instante temporal (36), y se puede representar como (37).

$$s(t) = \int x(a)w(t-a)da \tag{36}$$

$$s(t) = (x * w)(t) = x(t) * w(t) \tag{37}$$

Donde $x(t)$ es la entrada, $w(t)$ es el *kernel* (también conocido como *patch*) y $s(t)$ es la salida, que se refiere a un *feature map* o mapa de características. Estas características serán **espacialmente invariantes** (San José, R., 2019).

Al trabajar con señales muestreadas en tiempo, tenemos señales discretas, no continuas, por lo que la expresión de la convolución pasará a ser (38).

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{38}$$

En Machine Learning, las entradas a estas redes son a*rrays* multidimensionales, denominados **tensores** (es un *array* de 3 o más dimensiones). Además, en este trabajo, las convolucionales no se realizan en una única dimensión (por la naturaleza multidimensional de la entrada), sino en dos, es decir, vamos a utilizar un *kernel* bidimensional (Goodfellow, I. *et al*., 2016). Denominando a la entrada como I y al *kernel* como K, la expresión de la convolución bidimensional será (39).

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{39}$$

Y gracias a la propiedad conmutativa de la convolución, se obtiene una expresión más utilizada en los algoritmos de Machine Learning al ser más directa de implementar por contener menos variaciones en el rango de valores válidos de *m* y *n* (Goodfellow, I. *et al*., 2016), la expresión (40).

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(i,j) \tag{40}$$

Como se puede observar, el *kernel* se ha dado la vuelta con respecto a la entrada para obtener esta propiedad conmutativa. Sin embargo, no es importante en la implementación de las redes neuronales, por lo que muchas bibliotecas de redes neuronales implementan lo que se conoce como la correlación cruzada (Goodfellow, I. *et al*., 2016), que es como una convolución, pero sin dar la vuelta al *kernel*. Esta se muestra en la expresión (41).

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(i,j) \tag{41}$$

Técnicamente, este volteo del *kernel* se debería especificar para mantener ese convenio a lo largo de todas las operaciones de convolución, pero esta no es una operación que se realice de manera aislada, sino que se combina con otras para formar la red neuronal, por lo tanto, este volteo del *kernel* no es relevante en las redes neuronales (Goodfellow, I. *et al*., 2016). De esta forma, la convolución en una matriz se puede ver como:

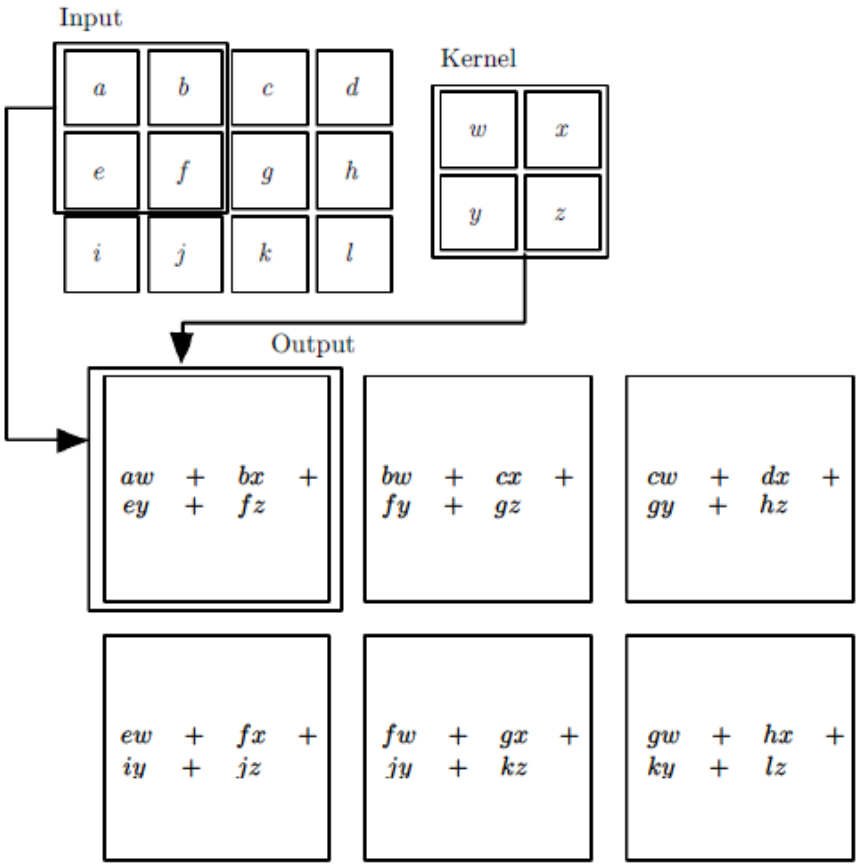


Figura 24. *Operación que realiza una capa convolucional en una red.*

En el caso mostrado en la Figura 24 (Goodfellow, I. *et al*., 2016) se tiene una convolución 2x2 (el *kernel* es una matriz 2x2) y con un *stride* de 1 (el *stride* es el número de pasos que se mueve el *kernel* de una sola vez por la matriz de datos).

Este tipo de redes neuronales formadas por una serie de capas, comparten los pesos entre las entradas que contengan la misma información y se entrenan de manera conjunta (*"Weight sharing"*). Esto es lo que le va a dar a las características extraídas (*features*) esa **invarianza espacial** comentada anteriormente, ya que, por ejemplo, en imagen, si en dos ejemplos que se introducen a la red aparece un mismo objeto ubicado en una zona distinta en cada ejemplo, los pesos de esas dos entradas se compartirán y entrenarán de forma conjunta, llegando a ser capaz de generalizar y llegar a aprender lo que es el objeto, **independientemente de su posición** (Sáez Bombín, S., 2018).

El efecto que tiene una capa convolucional sobre el tamaño o la cantidad de datos es el de añadir ***feature maps***, es decir, se generan tantos *feautre maps* como *kernels*, lo que representa la profundidad de la capa (Sáez Bombín, S., 2018).

Esto se muestra en la Figura 25, de (San, P. *et al.*, 2017, pp. 186-204), que representa ría solamente una capa convolucional, con J *kernels* de tamaño kx1, generando J *feature maps*.

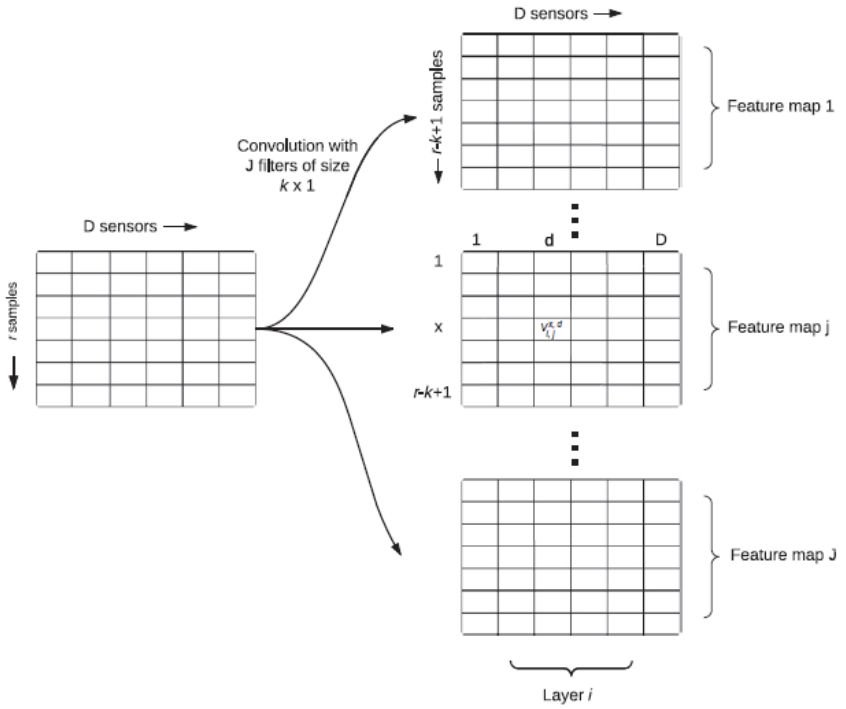
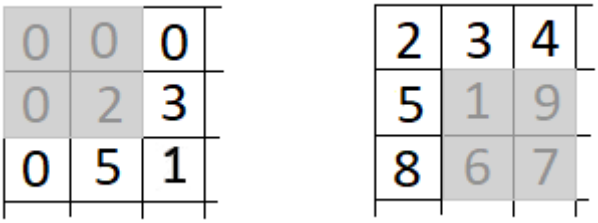


Figura 25. *Salida de una capa convolucional.*

Como se ha comentado anteriormente y como se explicará más a fondo en secciones posteriores, en este Trabajo Fin de Máster, se trabajan con convolucionales 2D debido a la naturaleza multidimensional de las entradas, que son tensores 4D en los que una dimensión tiene un tamaño de 1, que irá cambiando en función del número de *feature maps* que se quieran en cada capa, con lo que el valor de esta dimensión a la entrada de la red se puede entender como que tenemos solamente 1 *feature map* de las características (*features*) con las que entrenamos (Sáez Bombín, S., 2018).

Otro aspecto en las capas convolucionales a tener en cuenta es el ***padding*** o padeado (Udacity, 2018) que permite ajustar el *kernel* al tamaño de los datos. Se distinguen dos tipos:

- ***Same***: En este caso, en los límites de la matriz de datos, se añaden una fila y columna a cada lado, rellenado con ceros (Figura 24.a).
- ***Valid***: En vez de rellenar con ceros, lo que se hace es que no se tiene en cuenta la última y fila y columna en cada caso (Figura 24.b), es decir, se eliminan.



(a) (b)

Figura 26. *Padding same (a) y valid (b).*

Para terminar con las redes convolucionales, una forma habitual de ayudar al rendimiento de la red es mediante lo que se conoce como ***pooling***. Esta técnica permite reducir la dimensionalidad de los datos de varias maneras, y habitualmente, las capas convolucionales van seguidas de una capa de *pooling* (Sáez Bombín, S., 2018). Hay dos tipos:

- ***Max-pooling***: Es la forma más común (y la que se utiliza en este trabajo). Consiste en quedarse con el máximo de los datos de la ventana (*kernel*) que se elija (2x2, 3x3…). Esta ventana se irá moviendo también con un *stride* definido (Udacity, 2018).
- ***Average-pooling***: En este caso se realiza un promedio de dichos datos (Udacity, 2018).

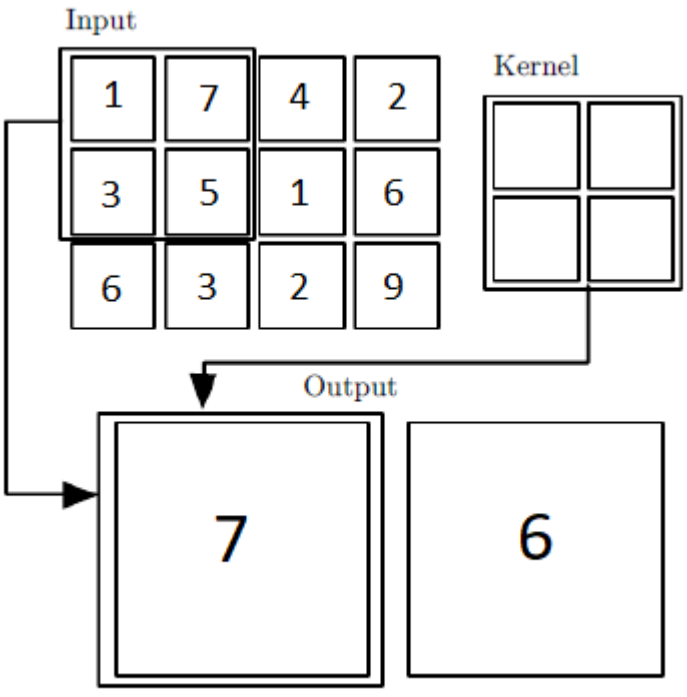


Figura 27. *Operación de max-pooling.*

Por lo tanto, al diseñar una capa convolucional, hay que definir: el tamaño del *kernel*, el *stride*, el tipo de *padding* y el número de *feature maps* que se van a tener a la salida. En la capa en la que se realiza el *pooling* también hay que definir el tamaño del *kernel* y el

stride (Sáez Bombín, S., 2018). Además, muchas veces estas dos capas van acompañadas de una capa de *Batch normalization* siendo habitual, en las redes convolucionales, el bloque (San José, R., 2019):

$$x \rightarrow Conv(x) \rightarrow Pool(Conv(x)) \rightarrow Batch\_norm(Pool(Conv(x)))$$

Redes Recurrentes (RNN)

Este tipo de redes se utilizan para procesar datos secuenciales (Goodfellow, I. *et al*., 2016). Al igual que las redes neuronales convolucionales, este tipo de redes utiliza el *"Weight sharing"*, pero en este caso, en vez de espacial, se comparte de manera temporal, no espacial (Sáez Bombín, S., 2018).

Por simplicidad, se dice que una RNN es aquella que opera una secuencia que contiene vectores $\boldsymbol{x}^{(t)}$ con un índice de tiempo *t* que va de 1 a τ. Normalmente trabajan en *batches* como todas las redes neuronales que pueden contener secuencias de distintas longitudes (Goodfellow, I. *et al*., 2016). Además, pueden ser utilizadas con 2 dimensiones, como en el caso de esta Trabajo Fin de Máster, y cuando tiene datos relacionados temporalmente, mantiene conexiones que van hacia instantes temporales anteriores (Goodfellow, I. *et al*., 2016).

En todas las redes neuronales, al definirlas, se generan grafos computacionales que mantienen la arquitectura y estado de la red (se puede considerar como la red en sí), pero en este tipo de redes, estos grafos computacionales se "desenrollan" por su carácter temporal (Goodfellow, I. *et al*., 2016). Esto se muestra en la Figura 28, de (Goodfellow, I. *et al*., 2016).

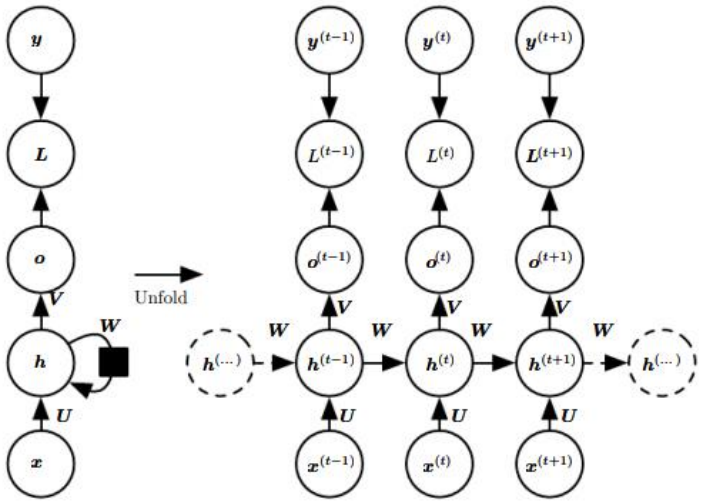


Figura 28. *Grafo computacional desenrollado.*

En esta Figura 28 se puede ver cómo el grafo computacional, debido a la presencia de una parte recurrente, puede verse como un grafo con un camino para cada instante temporal considerado.

También se puede observar que la red está compuesta conceptualmente por una serie de celdas que mantendrán un cierto estado correspondiente a cada unidad de tiempo

considerada (Sáez Bombín, S., 2018). Al tratarse de una secuencia, la *back-propagation* con la que actualizamos los pesos de la red, interesaría poder hacerla hasta el inicio de la secuencia para tener una idea de la variación de toda la secuencia a lo largo del tiempo, sin embargo, en la práctica, con secuencias largas, esto no va a ser posible, realizándose esta propagación hasta donde se pueda (Udacity, 2018).

Debido a la utilización de *"Weight sharing"*, toda esta propagación actualiza los pesos de cada celda de la red aplicándose sobre los mismos parámetros, provocando que haya muchas actualizaciones correladas (Sáez Bombín, S., 2018), algo que nos puede llevar a los fenómenos conocidos como:

- ***Exploding gradient***: A cada actualización el peso se va incrementando, tendiendo a infinito y llevando a la red a ser incapaz de aprender (Sáez Bombín, S., 2018).
- ***Vanishing gradient***: Es el fenómeno opuesto, en el que llega un momento que la actualización tiende a 0, siendo incapaz de aprender más la red (Sáez Bombín, S., 2018).

Sin embargo, hay dos técnicas que superan estos problemas (Udacity, 2018):

- ***Gradient clipping***: Para evitar el *exploding gradient* sin un gran coste. Se define un valor de actualización máximo (Δmax) y no se deja superarlo (42).

$$\Delta w \leftarrow \Delta w \frac{\Delta max}{\max(|\Delta w|, \Delta max)} \tag{42}$$

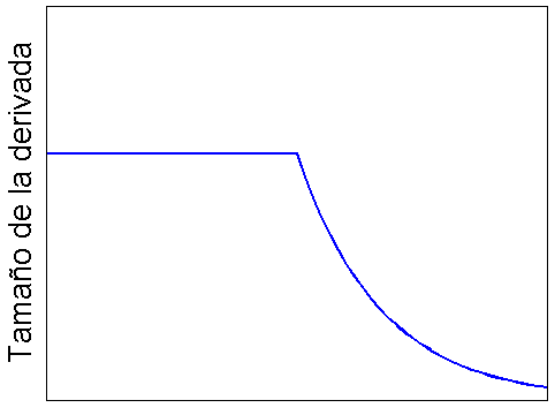


Figura 29. *Ejemplo de gradient clipping aplicado a un gradiente que empieza a tender a infinito.*

  Como se muestra en la Figura 29, a medida que la actualización se propaga hacia atrás, el tamaño de la derivada va aumentando, lo que provocaría el problema de *exploding gradient*, pero gracias a este sistema *gradient clipping*, este aumento se ve frenado en Δmax (Udacity, 2018).
- ***"Memory loss"***: Para evitar el *vanishing gradient*. Cada celda de la red mantiene solamente el estado con respecto a las celdas más cercanas a ella (las que se corresponden con instantes temporales cercanos) en vez de intentar mantener el estado de toda la red, olvidándose de las celdas que se corresponden con instantes temporales muy alejados. Sin embargo, este método es costoso (Udacity, 2018).

Por esto último existen las celdas **LSTM** (*Long-Short-Term Memory*). Las celdas que forman conceptualmente la estructura de una red recurrente normal podrían considerarse, en su forma más básica, como en la Figura 30 (Sáez Bombín, S., 2018).

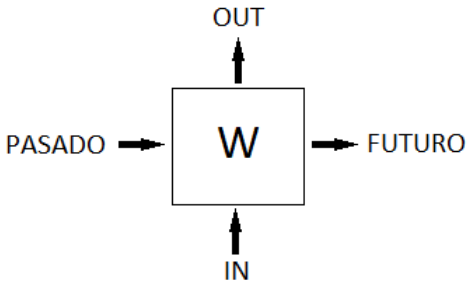


Figura 30. *Ejemplo de una celda recurrente básica.*

Esta celda se corresponde con una red neuronal en sí misma (Sáez Bombín, S., 2018), y esto es algo que se va a cumplir con todos los tipos de celda de las redes recurrentes (por el grafo computacional que se desenrolla), Figura 31.

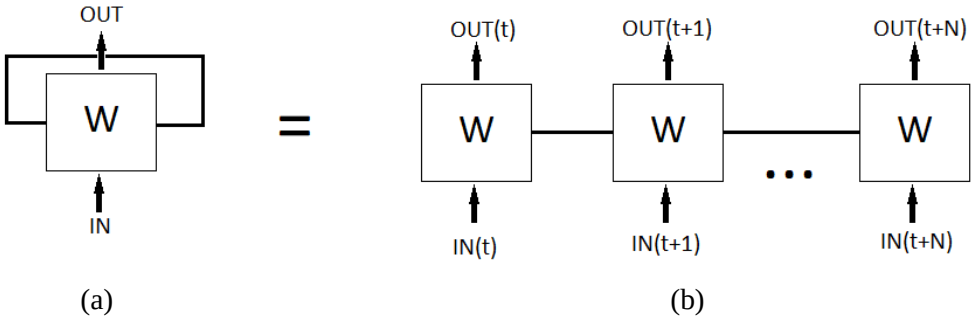


Figura 31. *Red neuronal recurrente en su versión "comprimida" (a) y unfolded ("desenrrollada") (b).*

Sin embargo, en las redes neuronales recurrentes LSTM, la celda pasa a ser:

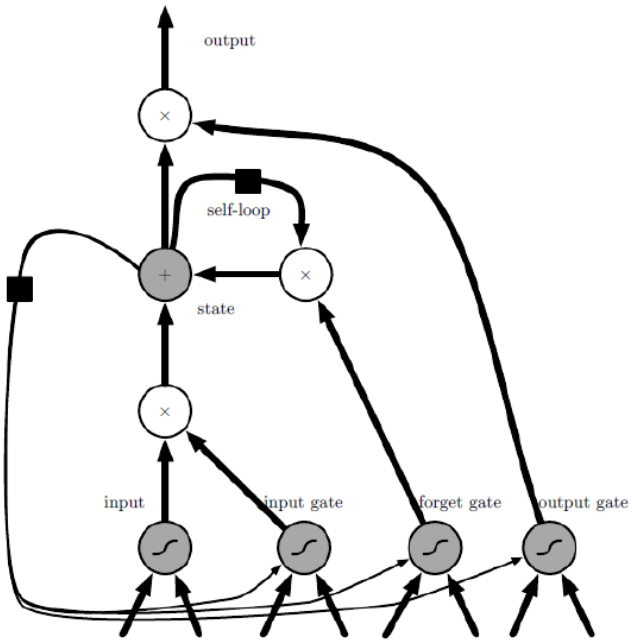


Figura 32. *Celda LSTM.*

En la Figura 32 (Goodfellow, I. *et al*., 2016) se muestra la celda LSTM, que trabaja como si fuera una pequeña red neuronal, permitiendo reducir el problema del *vanishing gradient*. Está constituida por una pequeña unidad de memoria que permite recordar el estado temporal de la red, además de unas unidades multiplicativas llamadas puertas para el control del flujo de la información. Cada celda contiene:

- Una puerta de entrada (***input gate***): controla el flujo de entrada a la unidad de memoria.
- Una puerta de salida (***output gate***): controla el flujo de salida de la unidad de memoria al resto de la red.
- Una puerta de olvido (***forget gate***): escala el estado interno de la celda antes de añadirlo como una entrada a la celda a través de conexiones recurrentes propias. Permite resetear de forma adaptativa la memoria (el estado) de la celda (Sak, H. *et al.*, 2014), lo que hace que evite el problema del *vanishing gradient*.

Además, las conexiones entre la unidad de memoria y las puertas se conocen como "conexiones mirilla" (***peephole connections***) porque las acciones indicadas en las puertas se realizan con una cierta probabilidad dependiendo del contexto de la red y de las entradas y salidas anteriores, no como una decisión binaria (Sáez Bombín, S., 2018). Como todas las redes, permite la aplicación de regularizaciones como L2 y Dropout (Zaremba, W. *et al*., 2015).

Pero las LSTM no son las únicas celdas disponibles para redes recurrentes. En este Trabajo Fin de Máster también se han utilizado las celdas **GRU** (*Gated Recurrent Unit*), similares a las LSTM, pero más sencillas, al tener únicamente 2 puertas (***update gate***, que selecciona cuándo el estado h es actualizado por el nuevo estado $\tilde{h}$ y ***reset gate***, que decide cuándo el estado previo es ignorado) y por lo tanto menos parámetros que ajustar, lo que

las hace una mejor opción para cuando se tienen menos datos (Cho, K. *et al*., 2014). La *unit* de este tipo se muestra en la Figura 33.

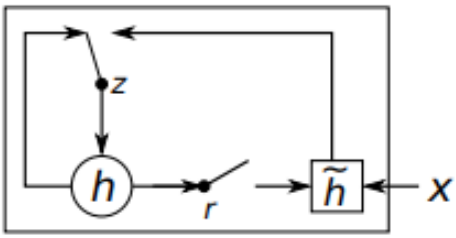


Figura 33. *Celda GRU.*

En definitiva, las redes neuronales recurrentes son un modelo que permite mapear secuencias de longitud variable a vectores de longitud fija, o bien secuencias de longitud fija a vectores de longitud variable, así como secuencias de longitud variable a vectores de longitud variable (Sáez Bombín, S., 2018). Son una extensión natural de las MLP, pero que añaden memoria al problema (San José, R., 2019).

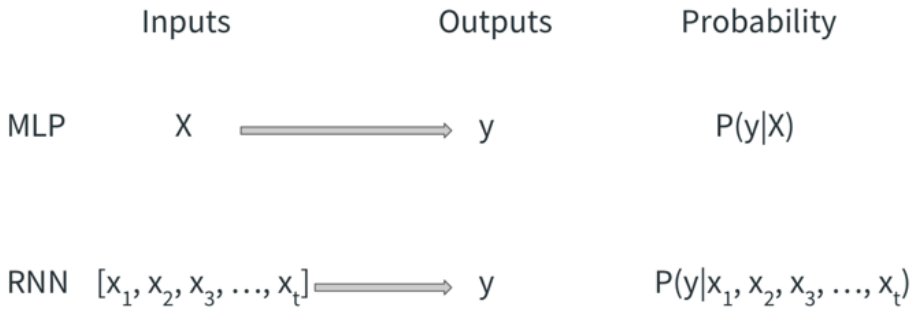


Figura 34. *MLP vs. RNN.*

Es por ello por lo que se utilizan en gran medida para *Machine Translation* o *Speech Recognition*, pero en el caso de este Trabajo Fin de Máster se utilizan para trabajar con la secuencia temporal de los datos provenientes de los sensores (Udacity, 2018), y tanto para clasificación como para generación (predicción de movimientos y actividades).

Generative Adversarial Network (GAN)

En este tipo de redes se entrenan de manera simultánea dos modelos: un **modelo generador *G*** que captura la distribución de los datos y un **modelo discriminador *D*** que estima la probabilidad de que una muestra venga de los datos de entrenamiento en vez del generador *G* (Goodfellow, I. et al., 2014).

Por lo tanto, permite la generación de datos sintéticos de un gran realismo a través de forzar que esos datos generados sean casi indistinguibles estadísticamente de los reales (San José, R., 2019).

El funcionamiento consistiría básicamente en:

- Red generadora: toma como entrada un vector aleatorio y lo transforma en datos sintéticos (depende de con lo que estemos trabajando, pueden ser imágenes o como en el caso de este trabajo, una matriz de cuaterniones) (San José, R., 2019).
- Red discriminadora: toma como entrada los datos (imagen/matriz de cuaterniones) reales o sintéticos, y predice si esos datos vienen del set de entrenamiento o han sido generados por el generador *G* (San José, R., 2019).

Un ejemplo de este funcionamiento se muestra en la Figura 35 (San José, R., 2019).

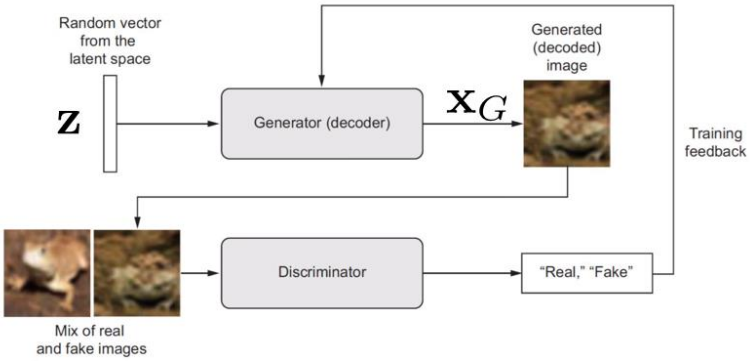


Figura 35. *Ejemplo de GAN.*

Donde ***z*** es ruido aleatorio, ***x*** son los datos reales y $\boldsymbol{x}_G$ son los datos generados.

La red generadora es entrenada con el objetivo de engañar a la red discriminadora, y la red discriminadora es entrenada con el objetivo de no ser engañada. Por lo tanto, se tiene un juego ***minimax*** de dos jugadores (*G* y *D*), ya que estamos intentando minimizar y maximizar una función de coste. Esta función será la mostrada en (43).

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right] \quad (43)$$

En esta función $p_{data}(x)$ es la distribución de los datos, que no es conocida pero sí tenemos muestras de ella (los datos reales), $p_z(z)$ es la distribución del ruido aleatorio que introducimos a *G* (espacio latente), *D(x)* es la probabilidad de que *x* provenga de los datos y no del generador y *D(G(z))* es la probabilidad de que a la entrada del discriminador tengamos datos del generador *G* (Goodfellow, I. et al., 2014).

Por lo tanto, el juego *minimax* viene de:

- El generador *G* quiere generar datos que sigan la distribución $p\,(x)$ y engañar al discriminador *D* (San José, R., 2019). Esto se traduce en (44).

$$\begin{matrix} D(G(z)) \rightarrow 1 \\ \log D(x) + \log(1-1) = -\infty \textbf{ (minimización del coste)} \end{matrix} \quad (44)$$

- El discriminador *D* quiere aprender que *(G(z))* es falso (San José, R., 2019), lo que lleva a (45).

$$\begin{matrix} D(x) \rightarrow 1 \\ D(G(z)) \rightarrow 0 \\ \log(1) + \log(1) = 0 \textbf{ (maximización del coste)} \end{matrix} \quad (45)$$

Este proceso no puede realizarse optimizando por separado el generador *G* y el discriminador *D*, porque nos llevaría a tener *overfitting*. Por ello, se realiza un entrenamiento en el que se alternan *k* pasos optimizando *D* y un paso optimizando *G* (Goodfellow, I. et al., 2014). Así, el discriminador *D* se está adaptando constantemente a la mejora gradual del generador *G* y al final del entrenamiento este último es capaz de generar datos creíbles para el primero (San José, R., 2019).

Esto se muestra en la Figura 36, en la que la línea azul representa la distribución de *D*, la línea negra punteada es la distribución de los datos reales y la línea verde la distribución de los datos generados por *G*. Además, las líneas horizontales representan el mapeo de la distribución de *z* (de la que se muestrea) a la de *x* (datos reales) que realiza *G*. De esta forma, se ve cómo al inicio del entrenamiento (Figura 36.a) el discriminador es capaz de distinguir los datos reales de los falsos, ya que sus distribuciones están claramente diferenciadas, pero a medida que el entrenamiento avanza, las distribuciones de los datos reales y falsos van siendo cada vez más parecidas (el generador está modificando la distribución de los datos que genera) hasta que son exactamente iguales al final del entrenamiento (Figura 36.d) y el discriminador no es capaz de diferenciarlas ($D(x) = \frac{1}{2}$) (Goodfellow, I. et al., 2014).

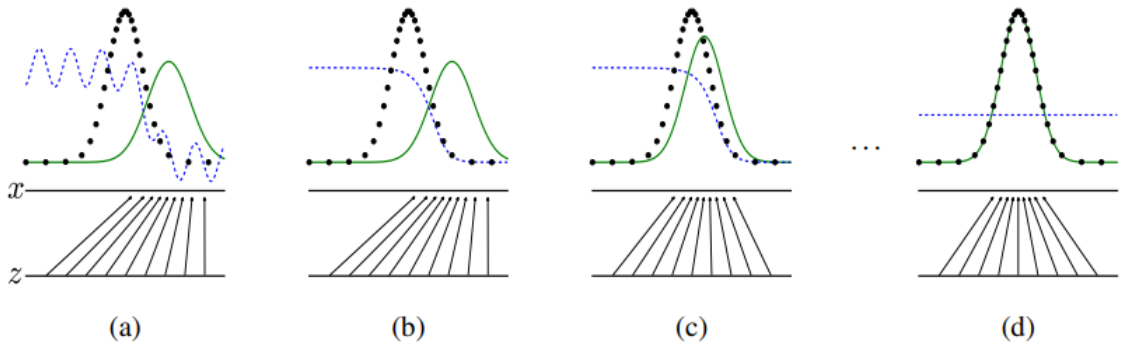


Figura 36. *Proceso de aprendizaje de una GAN.*

Tanto el modelo generador como el discriminador pueden ser redes neuronales convolucionales, recurrentes y MLPs. En concreto, es habitual que el generador *G* sea un el *decoder* de un *autoencoder* (San José, R., 2019).

Autoencoder

Un *autoencoder* es una red neuronal que es entrenada para conseguir copiar su entrada a su salida. Internamente tiene una capa ***h*** que describe un código utilizado para representar la entrada (Goodfellow, I. et al., 2014). La red se puede dividir en dos partes:

- Función *encoder*: Que traduce la entrada a un código que la representa.

$$\boldsymbol{h} = f(\boldsymbol{x}) \tag{46}$$

- Función *decoder*: Produce la reconstrucción de la entrada a partir del código proporcionado por el *encoder*.

$$\boldsymbol{r} = g(\boldsymbol{h}) \tag{47}$$

El objetivo final no es que copie perfectamente la entrada a la salida, sino que se le fuerza a que copie solamente algunos aspectos de la entrada que puedan resultar de interés (Goodfellow, I. et al., 2014).

En realidad, lo que se está aprendiendo es una transformación de mapeo del espacio original de datos a través de una representación dimensional reducida (código), que se puede ver como una compresión de los datos. Esta idea se representa en la expresión (48) y en la Figura 37 (San José, R., 2019).

$$\hat{x} = D(E(x)) \tag{48}$$

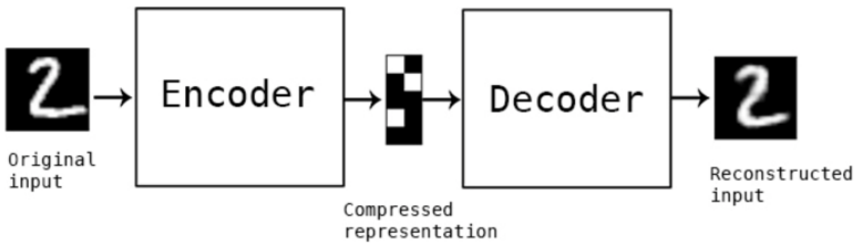


Figura 37. *Funcionamiento del Autoencoder.*

Algunas de las **características** de los *autoencoders* son las siguientes (San José, R., 2019):

- Son específicos para los datos para los que se entrena, es decir, solamente funcionarán bien para datos del mismo o semejante espacio de datos que los datos con los que se entrene.
- Producen pérdidas, lo que supone que la salida descomprimida estará degradad con respecto a la entrada (esto se puede ver en la Figura 37).
- Producen transformaciones no lineales.

Y de entre sus aplicaciones (San José, R., 2019) destacan:

- *Denoising*: limpiar de ruido los datos originales.
- Reducción de las dimensiones: el código generado es de menor dimensión que los datos originales, pero mantiene sus características e información principal.
- **Generador de datos**: Esta es la aplicación que interesa para este trabajo, de cara a introducirlo en una GAN como generador *G*.

Un tipo de *autoencoders* son los ***Variational Autoencoders (VAEs)***. Estos permiten aprender espacios latentes (espacios de datos) altamente estructurados y generar datos nuevos (los *autoencoders* normales no son capaces de generar anda "nuevo") (San José, R., 2019).

Los VAEs aprenden **parámetros de una distribución estadística** de un espacio latente, como pueden ser su media y su varianza (San José, R., 2019). Esta idea se muestra en la Figura 38 (San José, R., 2019).

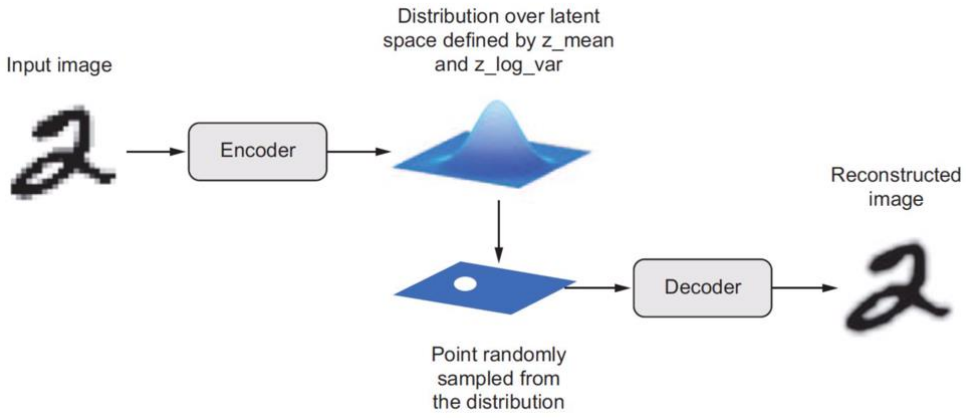


Figura 38. *Funcionamiento de un VAE.*

En definitiva, lo que se consigue es que el espacio latente generado por el *encoder* sea una distribución conocida (generalmente normal), lo que hará que cualquier punto aleatorio de esa distribución tenga una salida por el *decoder* que siga la distribución del espacio de los datos de entrada, es decir, se pueden generar datos totalmente nuevos a partir de números aleatorios (San José, R., 2019).

3.4 Estado del Arte

Esta sección se va a dividir en dos subsecciones, distinguiendo entre el Estado del Arte en clasificación y generación de movimientos.

3.4.1 Estado del Arte en Clasificación

Se presenta una serie de publicaciones y trabajos que realizan el reconocimiento de actividades a través del Deep Learning y con sensores de captura de movimientos, resaltando trabajos en los que se trabaja con cuaterniones, que son la magnitud a utilizar.

En los últimos años, el problema del Reconocimiento de la Actividad Humana (HAR), ha tenido un gran impulso y se ha visto favorecido por la inclusión de técnicas de Deep Learning y de los sensores de captura de movimientos, tal y como se indicó en (Sáez Bombín, S., 2018).

Además, se ha visto un aumento en el interés de trabajar y de resolver el problema de HAR mediante la utilización de cuaterniones. En este sentido, caben destacar los siguientes trabajos:

En primer lugar, se presenta el trabajo (Gaudet, C. J. y Maida A. S., 2018) en el que se propone la utilización de los cuaterniones, no como números complejos en 4 dimensiones, sino como 4 números "reales", uno en cada dimensión (w, x, y, z), representando igualmente una rotación en el espacio 3D y la utilización de una red convolucional para sacar características y poder, posteriormente, clasificar esos datos. Esta línea se asemeja a la que se utilizó en (Sáez Bombín, S., 2018), donde se utilizaba esta misma representación de los cuaterniones y donde además de las redes convolucionales, se propuso la utilización de redes recurrentes y de una combinación de ambas.

Por otro lado, se destaca el trabajo (Pavllo, D. *et al.*, 2018) en el que se presenta una red recurrente, que se utiliza para clasificación de poses humanas (también a partir del uso de los cuaterniones) así como la predicción o generación de poses futuras dado un segmento de actividad, a partir también de los cuaterniones.

Para terminar con trabajos que tengan relación con cuaterniones en Deep Learning, cabe mencionar una publicación (Parcollet, T. *et al.*, 2019) en la que se propone utilizar los cuaterniones como lo que son, números complejos con una parte real y 3 partes imaginarias, teniendo que modificar y haciendo más complejo el cálculo de gradientes y de la función de pérdidas, entre otras cosas. Sin embargo, el número de parámetros de la red disminuye por un factor 4, ya que se reduce el problema a una sola dimensión (en lo que respecta al cuaternión).

También es importante hablar de publicaciones que busquen resolver el problema HAR a través del dataset REALDISP, para así poder comparar los resultados obtenidos en el Trabajo Fin de Máster con las publicaciones que trabajen sobre un mismo dataset. En este sentido, se presentan las mismas que se presentaron en (Sáez Bombín, S., 2018) incluyendo además el propio Trabajo Fin de Grado y alguna publicación más, ya que no se han encontrado más publicaciones relacionadas con este dataset.

El primer trabajo relacionado con REALDISP es de los propios diseñadores del dataset. En él, utilizan algoritmos de Machine Learning con la intención de ver el efecto que tiene el desplazamiento o mal posicionamiento de los sensores a la hora de resolver el problema HAR (Baños, O. *et al*., 2012). Los algoritmos que se utilizan son: *Nearest Class Center* (NCC), *K-nearest neighbors* (KNN) y *Decision Trees* (DT), con los que se realiza la clasificación. En la Figura 39 se muestran los resultados que obtienen para cada algoritmo y tipo de set de datos.

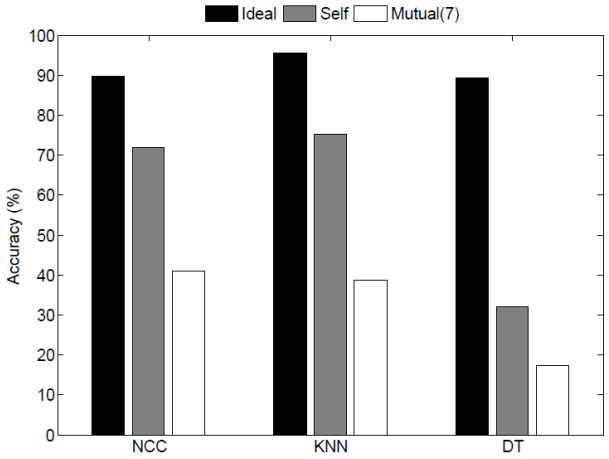


Figura 39. *Resultados del reconocimiento de movimientos en (Baños, O. et al., 2012).*

De los mismos autores de este primer trabajo, se encuentra el segundo trabajo o publicación a destacar relacionado con este dataset (Baños, O. *et al*. 2014b, pp. 9995-10023). En esta ocasión, los algoritmos de Machine Learning utilizados son: *K-nearest neighbors* (KNN), *Decision Trees* (DT) y *Naive-Bayes* (NB). De este trabajo cabe destacar una idea utilizada en este Trabajo Fin de Máster en relación al preprocesamiento de los datos, ya que utiliza una *sliding window* (en este caso sin *overlap*). Como métricas en este trabajo utilizan: FS1 = "media", FS2 = "media y desviación estándar y FS3 = "media, desviación estándar y *crossing rate* máxima, mínima y media" (la *crossing rate* es la tasa a la que cambian los símbolos de una señal) y se consideran dos tipos de tratamiento previos a la clasificación, que son *Feature-Fusion Multi-sensor Activity Recognition Chain* (FFMARC) y *Hierarchical Weighted Classifier* (HWC) (Sáez Bombín, S., 2018). Diferencian distintos escenarios considerando distinta cantidad de actividades cada vez y separan por completo, como en la anterior publicación, los datos ideales de los no ideales. En la Figura 40 se muestran los resultados de este trabajo considerando todos los sensores.

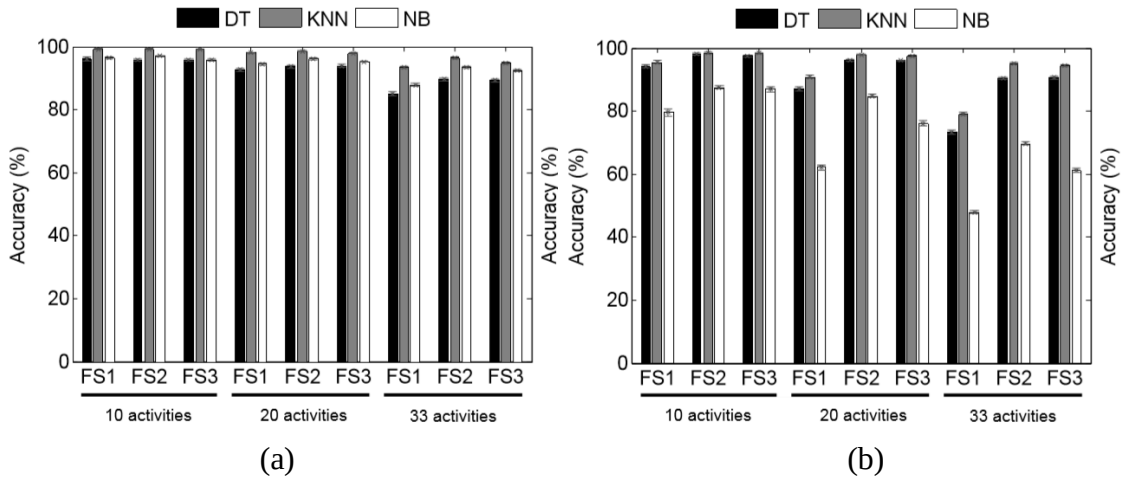


Figura 40. *Resultados del trabajo (Baños, O. et al. 2014b, pp. 9995-10023) para FFMARC (a) y HWC (b).*

Las conclusiones que se pueden sacar de la Figura 40 son dos: 1) en el caso FFMARC, el algoritmo KNN es el que mayor precisión alcanza (95% para colocación ideal de sensores y caso de FS3 y 33 actividades) y el más fiable, ya que es al que menos le afecta a la precisión la mala colocación de sensores; 2) mientras que en el caso HWC, el algoritmo más fiable y robusto es, otra vez, el KNN (alcanza también un 95% de precisión en condiciones similares). Sin embargo, lo que hace que el modelo HWC supere al FFMARC es su buen rendimiento en los casos en los que los sensores no tienen una colocación ideal.

También cabe destacar un trabajo (Zhu, J. *et al*., 2017) que se centra en la extracción de las características y compara sus resultados con los del trabajo de (Baños, O. *et al*. 2014b, pp. 9995-10023). En este caso, utilizan el algoritmo de Machine Learning, *Random Forest*, obteniendo mejor resultados que los del trabajo de (Baños, O. *et al*. 2014b, pp. 9995-10023). Hace hincapié en la extracción de características (*features*) tanto en el dominio del tiempo (señales provenientes de los sensores y algunas señales procesadas) como en el dominio de la frecuencia (FFTs (*Fast Fourier Transforms*) de las señales en el dominio del tiempo) (Sáez Bombín, S., 2018).

En este sentido, para el domino del tiempo utiliza *features* como son la media, la desviación típica o la energía de la señal y para el dominio de la frecuencia utiliza otras como el índice de la componente en frecuencia con la magnitud mayor, una media en frecuencia o la energía de algunas bandas de la FFT (Sáez Bombín, S., 2018).

Para terminar lo referente a este trabajo, cabe destacar que las precisiones máximas que consiguen son un 99,4% de precisión para los datos ideales y separando por sujetos los sets de entrenamiento, validación y test, incluyendo la "No actividad", y un 99,1% de precisión para el mismo escenario, pero incluyendo la actividad "No actividad" (Sáez Bombín, S., 2018). En ambos casos realizan previamente un preprocesamiento sobre los datos.

Siguiendo con trabajos relacionados con el dataset READLISP, cabe destacar uno que sí utiliza técnicas de Deep Learning (San, P. *et al.*, 2017, pp. 186-204), ya que se elabora una red convolucional, mostrada en la Figura 41, que permite obtener un 90,1% de precisión para el conjunto de datos en los que los usuarios se colocan los sensores (*self*), sin incluir la "No actividad". En este caso también se utiliza una *sliding window* (sin *overlap*), siendo esta de 1 segundo de duración, es decir, muy pequeña, por lo que no se obtienen capturas completas de las actividades, teniendo datos "menos significativos" o que contienen menos información que si la ventana fuera más grande, lo que produce que se dificulte el entrenamiento.

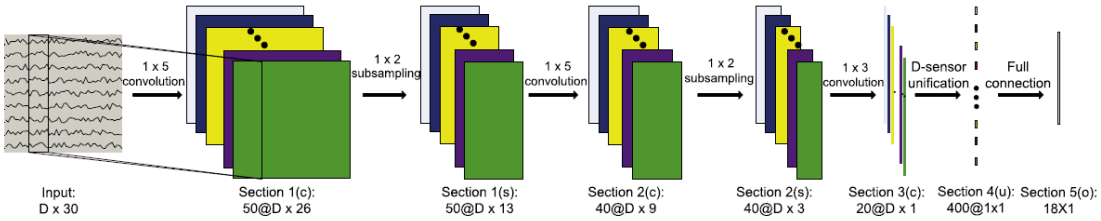


Figura 41. *Red convolucional de (San, P. et al., 2017, pp. 186-204).*

El hecho de obtener una precisión tan elevada sin haber hecho ningún preprocesamiento de los datos para una extracción de características ("a mano") como se realizaba en las anteriores publicaciones, lleva a los autores de este trabajo a afirmar que "la red convolucional es capaz de descubrir representaciones de características mejores que las realizadas a mano" (San, P. *et al.*, 2017, pp. 186-204). Además, terminan indicando las posibilidades y potencial de las redes neuronales (tanto convolucionales como recurrentes) en la resolución del problema HAR (Sáez Bombín, S., 2018).

Ya para terminar con esta sección de Estado del Arte, se presenta el Trabajo Fin de Grado (Sáez Bombín, S., 2018) previo a este Trabajo Fin de Máster. En este trabajo se utilizaron redes neuronales convolucionales, recurrentes (LSTM) y convolucionales-recurrentes para clasificar 34 actividades (se incluyó la "No actividad") del dataset REALDISP. Como paso previo al entrenamiento, se realizaba un preprocesamiento sencillo que consistía en estandarizar los datos para que tuvieran una distribución con media 0 y desviación típica 1, y posteriormente utilizar una *sliding window* tanto con *overlap* como sin él para obtener los fragmentos de actividad con los que se alimentaría la red. Como resultado, se obtuvo una precisión máxima de casi el 99% pero en la que no había una separación de sujetos entre entrenamiento, validación y test (en ese caso, la precisión bajaba hasta un 80%).

3.4.2 Estado del Arte en Generación

En cuanto a trabajos relacionados con generación de movimientos en el dominio de las señales de los sensores, apenas se ha podido encontrar alguno, así que también se presentan trabajos que pueden indicar ideas acerca de cómo realizarlo.

Se destacan 3 trabajos que se ocupan de distintos problemas, pero que se pueden resumir de la siguiente manera:

- Se pueden generar vídeos *frame* a *frame* (Vondrick, C., Pirsiavash, H. y Torralba, A., 2016), lo que demuestra que se puede generar un "movimiento" muestra a muestra.
- Se pueden trabajar con datos en el dominio de las señales de los sensores. Esto es así porque en (Soleimani, E. y Nazerfard, E., 2019) utilizan una GAN para generar datos de un sujeto en un sistema en el que se realiza *transfer learning*.
- En (Kulharia, V. *et al.*, 2017) se muestra que además de las típicas MLP y redes convolucionales, las redes recurrentes también se pueden utilizar dentro de las GANs.

Por lo tanto, tras esta revisión bibliográfica, en este Trabajo Fin de Máster se optó por intentar conseguir una buena clasificación (mejorar los resultados anteriores) pero separando por sujetos cada uno de los sets de datos, para conseguir una mayor generalización en el entrenamiento y asegurando un correcto funcionamiento independientemente del sujeto que realice la acción. Además, se decidió prescindir de la "No actividad" al no aportar información "real" del movimiento, ya que en los tiempos de descanso los sujetos podrían haber realizado distintos movimientos y actividades y, sin embargo, estarían todos ellos etiquetados como una misma actividad, pudiendo llevar a la red a confusión y entorpeciendo el entrenamiento. Todas estas decisiones referentes a la clasificación hacen que este Trabajo Fin de Máster se asemeje a lo realizado en (Zhu, J. *et al*., 2017), siendo el número de actividades, el preprocesamiento de los datos y los algoritmos utilizados las mayores diferencias entre ambos. Finalmente, también se optó por proponer una red que sea capaz de generar las señales de movimientos, como si las estuvieran generando los sensores.

Capítulo 4. Desarrollo de redes neuronales

Durante este capítulo se indican y explican los distintos tipos de **arquitecturas de red neuronal** desarrolladas durante este Trabajo Fin de Máster, el **preprocesamiento** que se realiza sobre los datos, cómo es **el entrenamiento** de cada red y los **resultados** obtenidos.

Como punto de partida para este Trabajo Fin de Máster se tomó (Sáez Bombín, S., 2018) donde ya se realizó un estudio con distintos tipos de redes neuronales, aunque, en este caso, se busca generalizar más los resultados obtenidos en dicho trabajo.

En este sentido, hay 2 partes claramente diferenciadas en este Trabajo Fin de Máster: una primera parte centrada en la clasificación de las actividades, y una segunda parte centrada en la generación de nuevas muestras de las actividades de manera sintética.

Así, primero se explica el preprocesamiento que se realizó sobre los datos del dataset y después, las partes de clasificación y generación, donde también se explicará la adecuación de los datos para cada tipo de red.

4.1 Preprocesamiento de los datos del dataset

A continuación, se explican las transformaciones que se realizan sobre los datos, adecuándolos a la red y realizando una serie de operaciones que se consideran necesarias y de interés.

4.1.1 Limpieza de los datos

En primer lugar, se realizó un estudio intensivo de los datos disponibles en el dataset. Tal y como se indica en la sección 2.3, los datos del dataset REALDISP están repartidos en 43 ficheros de datos en forma de matriz de 120 columnas y tantas filas como muestras se tengan. La 1ª y 2ª columna indican los instantes de muestreo, de la 3ª a la 119ª columna son las medidas del acelerómetro, giroscopio, campo magnético y cuaterniones para cada uno de los 9 sensores, y por último, la 120ª contiene las etiquetas de actividad.

De estos ficheros se sacan los cuaterniones separando por actividad y sujeto para un manejo más cómodo de los datos. La razón por la que se utilizan únicamente los cuaterniones se explica en la sección 2.3.

Centrando el foco en los cuaterniones, se decidió reconstruir los movimientos a partir de estos y mediante un programa en Unity. De esta forma, realizando la misma reconstrucción para todos los sujetos y actividades, se pudieron observar una serie de inconsistencias en los datos tal y como se indica en la sección 2.3.

Debido a estas inconsistencias se optó por utilizar únicamente los cuaterniones provenientes de 5 de los sensores (BACK, RUA, RLA, LUA y LLA), es decir, el sensor

de la espalda y los 4 de las extremidades superiores, ya que eran los que menos inconsistencias presentaban. Además, se realizaron una serie de operaciones en los que se conseguía que todos los sujetos estuvieran orientados en la misma dirección y sentido y que las pequeñas inconsistencias en algunos de los sujetos quedaran corregidas:

- Se calcula el cuaternión inverso de la primera muestra de un sensor y se aplica a cada uno de los cuaterniones sucesivos de dicho sensor, consiguiendo así la misma orientación para todos los sujetos.
- Se aplican unas ligeras rotaciones sensor a sensor para conseguir un movimiento correcto en los casos en los que los datos tenían alguna inconsistencia menor.

Por esta razón, en vez de hacer uso de las 33 actividades disponibles en el dataset original, se utilizaron hasta 11 actividades, aquellas en las que los brazos presentaban un movimiento suficientemente característico como para considerarlo como una actividad independiente. En concreto, los sujetos utilizados (ya que en algunos no hay datos para alguna de las actividades) son S1, S2, S3, S5, S8, S9, S10, S11, S13, S14, S16, S17, un total de 12 sujetos y las actividades utilizadas son:

- L9: Rotación del tronco (brazos extendidos).
- L10: Rotación del tronco (brazos doblados a cintura).
- L11: Flexión frontal de cintura.
- L12: Rotación de cintura.
- L13: Flexión frontal de cintura (alcanzando el pie con la mano del brazo opuesto).
- L19: Elevación lateral de los brazos.
- L20: Elevación frontal de los brazos.
- L21: Aplauso frontal.
- L24: Rotación de los hombros con mucha amplitud.
- L25: Rotación de los hombros con poca amplitud.
- L31: Remo.

Por lo tanto, para cada una de las pruebas se leen los cuaterniones en función del sujeto y la actividad de interés. Los cuaterniones de los que se disponen son unitarios, es decir, su norma es 1, y se encuentran en el espacio de valores [-1,1]. Si bien en Deep Learning la estandarización de los datos (forzar a los datos para que sigan una distribución normal *N(0,1)*) es una práctica extendida con el fin de facilitar la minimización del error, no siempre es aconsejable realizarla, y, en este caso, no se ha realizado, ya que a partir de unas pruebas iniciales se pudo observar que las redes funcionaban mejor con los datos no estandarizados.

4.1.2 Técnicas de preprocesamiento consideradas

Técnica de ventana deslizante

Tras la lectura de los datos y previo al entrenamiento, teniendo como base los trabajos de la sección 3.4, en todas las pruebas se considera la **técnica de la ventana deslizante**:

- Ventana deslizante (***sliding window***) **con *overlap***: Consiste en utilizar una ventana para producir “segmentos” de los datos con los que se alimentará la entrada de la

red. La longitud de la ventana deslizante se toma de ***n_time_steps***, fijado (salvo excepciones en algunas pruebas) en 128 muestras (2,56 segundos de actividad) al ser una cantidad de muestras que recoge la suficiente cantidad de información representativa de una actividad, y se desplaza ***step*** muestras para ir recorriendo toda la secuencia. El valor de *step* debe ser más pequeño que el de *n_time_steps* para no dejar muestras de actividad sin leer.

Los "segmentos" generados se almacenan en un tensor de dimensiones (49):

$$Tensor = [filas, canales, n\_time\_steps, features] \quad (49)$$

Siendo *filas* la cantidad de segmentos, *canales* el número de dimensiones a considerar, *n_time_steps* el número de muestras de cada segmento y *features* las distintas dimensiones de los cuaterniones de cada sensor. Se pueden considerar distintos formatos de estos tensores, según el número de *canales* considerados, como se muestra en (50).

$$Tensor = \begin{cases} [filas, 1, 128, 20] \\ [filas, 4, 128, 5] \\ [filas, 5, 128, 4] \end{cases} \quad (50)$$

En el caso de $[filas, 4, 128, 5]$ se consideran segmentos que contienen 4 matrices de 128 filas (muestras) y 5 columnas (cada uno de los sensores utilizados), por lo tanto, aquí *canales* representa cada una de las 4 dimensiones del cuaternión ($w + \boldsymbol{i}x + \boldsymbol{j}y + \boldsymbol{k}z$).
Si se considera $[filas, 5, 128, 4]$, *canales* correspondería a cada uno de los **sensores** utilizados (BACK, RUA, RLA, LUA, LLA), mientras que las columnas de las 5 matrices se corresponderían con las dimensiones del cuaternión.
Sin embargo, la configuración utilizada ha sido la de $[filas, 1, 128, 20]$ en la que se utilizan matrices con 20 columnas (cada uno de los 5 sensores con sus 4 dimensiones de cuaterniones) al permitirnos utilizar redes más profundas, ya que, al aumentar la profundidad de la red, el tamaño de los datos se suele ir reduciendo. Además, se realizaron pruebas con las otras dos dimensiones y no suponían ventaja alguna con respecto a este formato.

- Ventana deslizante (***sliding window***) **sin *overlap***: El procesamiento es el mismo que en el anterior punto, pero en este caso el parámetro ***step*** es igual al parámetro ***n_time_steps***, así que los "segmentos" generados no contienen información que ya haya en otros "segmentos", son disjuntos (se elimina el *overlap*) (Sáez Bombín, S., 2018).

Generalmente, utilizar la ventana deslizante con *overlap* se traduce en un mejor rendimiento de las redes en la evaluación, ya que se están generando más ejemplos con los que poder entrenarla.

En la Figura 42 se muestra la idea conceptual de la técnica de *sliding window*, que genera los segmentos a medida que recorre los datos.

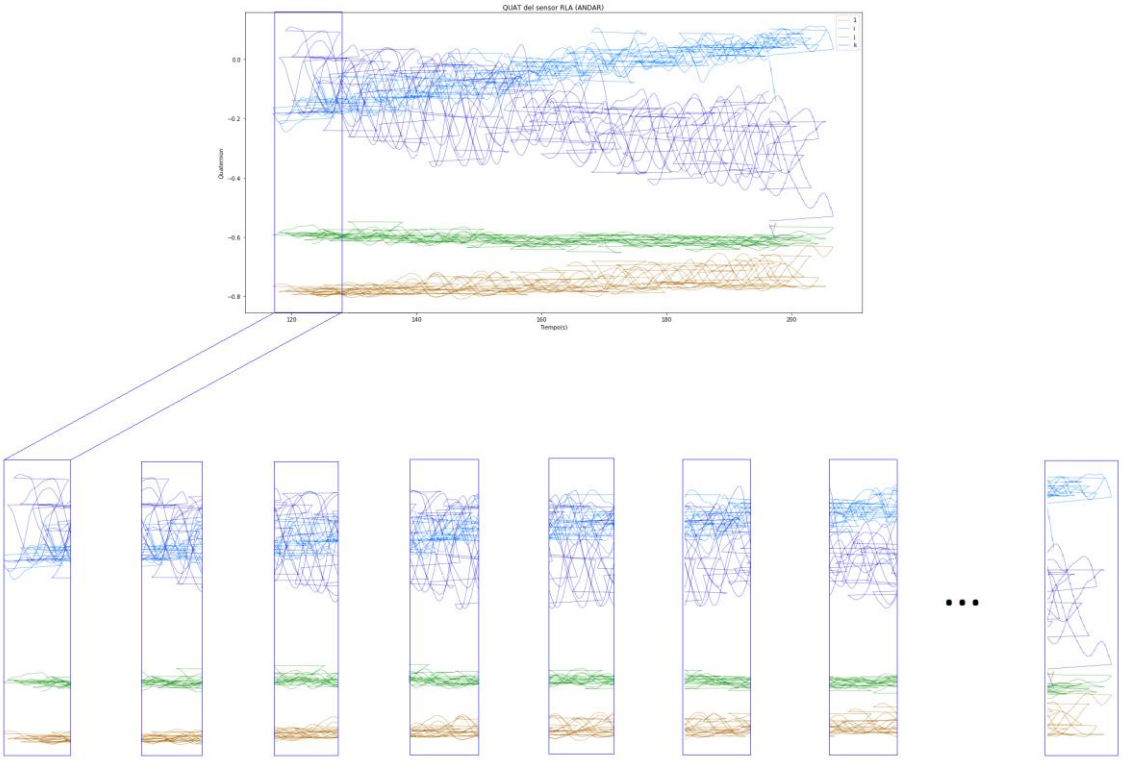


Figura 42. *Técnica de ventana deslizante.*

Data Augmentation

Posteriormente, debido a los pocos datos disponibles y la gran reducción de datos que se realiza del dataset (se simplifica el problema al tener menos actividades, pero a su vez se añade complejidad al reducir el número de sujetos y datos que se utilizan), se opta por incluir un *Data augmentation* que, en este caso, se realiza mediante rotaciones de los sujetos para tenerlos orientados en distintas direcciones. En concreto, se tienen todos los sujetos en orientaciones que distan 15º, es decir, se tienen 24 orientaciones de cada sujeto (0º, 15º, 30º, 45º, 60º, 75º, 90º, 105º, 120º, 135º, 150º, 165º, 180º, -165º, -150º, -135º, -120º, -105º, -90º, -75º, -60º, -45º, -30º, -15º con respecto a la orientación inicial del sujeto). Esto se muestra en la Figura 43, de izquierda a derecha y de arriba a abajo el orden de orientaciones indicado.

Este *Data augmentation* no solamente sirve para obtener más datos con los que entrenar, sino que con él se persigue conseguir invarianza a la orientación en el posterior reconocimiento de actividades.

Normalización de los datos

Esta normalización se lleva a cabo debido a la variabilidad entre clases que se tiene y con la intención de mejorar el entrenamiento y evaluación de la red neuronal.

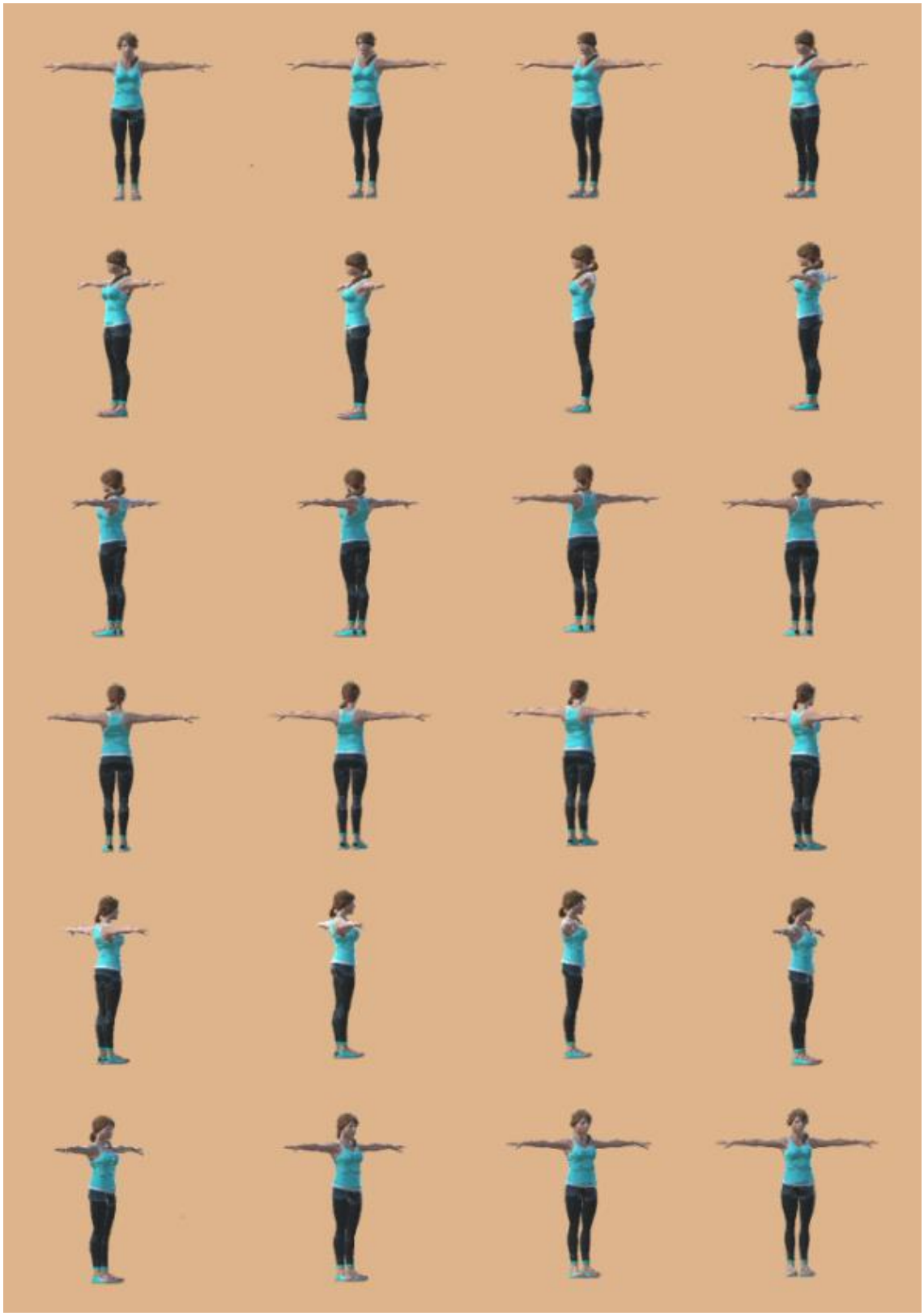


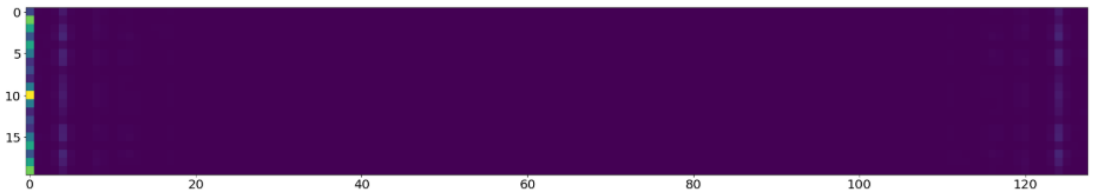
Figura 43. *Data Augmentation realizado a través de Unity.*

Extracción de características frecuenciales

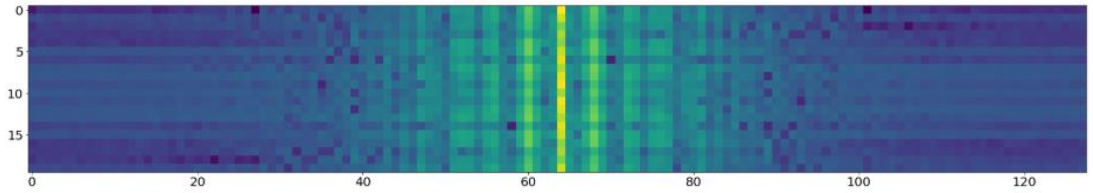
Siguiendo la estrategia utilizada en (Zhu, J. *et al*., 2017), se considera añadir información en el domino **frecuencial** a nivel **espacial**.

Esto se consigue a través del cálculo de la *Discrete Fourier Transform 2D* (**DFT-2D**) de cada uno de los segmentos, como si de una imagen se tratara. De esta forma se aumenta el número de *features*. Considerando a cada segmento como una imagen $f(x,y)$ de tamaño $M \times N$, la DFT-2D se calcula como en (51).

$$F(m,n) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \exp\left(-2\pi i \left(\frac{x}{M}m + \frac{y}{N}n\right)\right) \qquad (51)$$



(a)



(b)

Figura 44. *DFT-2D de una matriz de cuaterniones (a) y su potencia (b).*

En la Figura 44.a) se muestra un ejemplo de DFT de una de las matrices de datos de [128,20] y en la Figura 44.b) se muestra su potencia, para una mejor visualización de los patrones que pueda llevar asociados (actividad 25). Lo que se introduce a la red es la DFT, no su potencia, ya que, aunque parezca más significativa visualmente, tras unas pruebas se pudo comprobar que la red neuronal rendía mejor con la DFT.

La DFT normalmente es implementada por la *Fast Fourier Transform* (FFT). En este punto cabría preguntarse si una red neuronal es capaz de calcular el espectro frecuencial de una señal o imagen y en ese caso, si tiene sentido calcular la DFT a través de la FFT o si es mejor utilizar una red neuronal para su cálculo (o en el mejor de los casos, intentar incluir ese cálculo en una sola capa de nuestra red).

La respuesta a esa pregunta nos la da (Velik, R., n.d.), al asegurar, gracias al trabajo de (Keerthipala, W. W. L., Chong, L. T. y Leong., T. C., 1995), que sí es posible calcular la DFT a través de una red neuronal, pero que en ningún caso va a mejorar el rendimiento de la FFT. De hecho, intentar calcular la DFT mediante una red neuronal supondría un gasto elevado de tiempo de computación para conseguir algo que ya consigue la FFT de

manera rápida, que es un algoritmo bien definido y nos asegura la correcta realización de la DFT.

Esto supone una ligera extracción de características, no habitual dentro del Deep Learning, pero que por la justificación comentada lleva a tenerla en cuenta, ya que podría proporcionar información que puede ayudar a la red a acometer su función.

4.1.3 Preprocesamiento específico de Clasificación

En el caso de clasificación se utiliza la **técnica de ventana deslizante** con un 75% de ***overlap***, obteniendo tensores de tamaño $[filas, 1, 128, 20]$. Además, se utilizan el resto de las técnicas de preprocesamiento comentadas, obteniendo resultados para las siguientes combinaciones de técnicas:

1. Datos "en crudo": son los datos que se tienen del dataset, con las transformaciones indicadas en el apartado 4.1.1.
2. Datos "en crudo" y con *Data Augmentation*: en este caso, se orientan hacia 24 orientaciones diferentes los datos del anterior grupo.
3. Datos "en crudo", con *Data Augmentation* y calculando la DFT-2D: aquí, además, se añade el cálculo de la DFT-2D.
4. Datos normalizados y con *Data Augmentation*: como el tipo 2, pero los datos normalizados.
5. Datos normalizados, con *Data Augmentation* y calculando la DFT-2D: como el tipo 3, pero con los datos normalizados (previo al cálculo de la DFT-2D).
6. Datos únicamente de la DFT-2D.

En los casos en los que se añade la DFT junto a los datos temporales, los tensores pasan a tener un tamaño de $[filas, 1, 128, 40]$. Además, al ser un *Aprendizaje Supervisado* se hace uso de las etiquetas. Este vector de etiquetas de actividad se pasa al formato *One-hot encoding*, produciendo una matriz del mismo número de filas que los datos que se utilicen y con tantas columnas como número de clases tengamos (en este caso 11 actividades), apareciendo un 1 en la clase correcta y un 0 en el resto de las clases.

Finalmente, una vez realizada la transformación al formato $[filas, 1, 128, 20]$ o $[filas, 1, 128, 40]$ (dependiendo del caso en el que nos encontremos) los datos se separan en set de entrenamiento y test, y dentro de cada uno de ellos se realiza un barajeo de los segmentos y las etiquetas (manteniendo la correspondencia inicial), dejando los datos listos para el entrenamiento y posterior evaluación de la red. Este barajeo permite no dar ninguna "pista" a la red acerca de la secuencia de datos que se le proporciona.

4.1.4 Preprocesamiento específico de Generación

La generación sintética de actividades se enfoca desde dos puntos de vista: GANs y RNNs como predictoras.

Para el caso de las RNNs como redes predictoras no se realiza la transformación de los datos al formato $[filas, 1, 128, 20]$ mediante la técnica de ventana deslizante con *overlap*

y tampoco se calcula la DFT-2D. Por el contrario, se leen los datos, sin ningún tipo de *overlap* quedando un tensor de tamaño $[filas, 1, 20]$ (adecuado para la red recurrente). Los datos se pasan del espacio de valores [-1,1] al [0,1] para facilitar la predicción y no se utilizan etiquetas, sino que lo que se comparan son las muestras predichas con las reales (se está realizando una regresión sobre los datos). Además, los datos se separan en 3 sets (entrenamiento, validación y test).

Sin embargo, si se utilizan las GANs no se hace distinción entre 3 sets, solamente tenemos uno (utilizamos todos los datos), pero sí se generan los segmentos de tamaño $[filas, 1,128,20]$ con un 99% de *overlap*. Tampoco se calcula la DFT-2D ni se normalizan los vectores de entrada y no se utilizan las etiquetas que indican la actividad, sino que se generan durante el entrenamiento y en ellas, un 1 indica al discriminador *D* que el segmento al que está asignado proviene de los datos reales y un 0 indica que ese segmento ha sido generado sintéticamente por el generador *G*.

4.2 Diseño y estudio de redes neuronales

El diseño de la red va a depender de la cantidad de datos y del número de *features* que se tengan, así como de todas las consideraciones que se crean oportunas.

En el diseño de la red entra en juego no solamente la arquitectura de la red, sino también una serie de hiperparámetros que determinan el comportamiento de la red durante el entrenamiento.

4.2.1 Entrenamiento de la red

Una vez elegido el diseño de la red, la forma de entrenarla es mediante el ajuste de los hiperparámetros, con el objetivo de obtener el mejor resultado para cada una de las arquitecturas de las redes. Los hiperparámetros sobre los que se ha actuado en las distintas redes para la mejora del resultado han sido:

- ***n_time_steps***: Este parámetro indica la longitud de la ventana para generar los segmentos (matrices) que van a alimentar la red (Sáez Bombín, S., 2018).
- ***n_classes***: Número de clases a aprender (Sáez Bombín, S., 2018).
- ***n_channels***: Las dimensiones consideradas (Sáez Bombín, S., 2018).

Estos tres hiperparámetros son los que están ajustados a los datos y afectan a la estructura de la red neuronal. Sin embargo, hay otra serie de hiperparámetros que no están ajustados a la estructura de la red y que son los que se modificarán en busca de un mejor resultado (Sáez Bombín, S., 2018):

- ***epochs***: Es el número de veces que el set de entrenamiento completo va a pasar por la red neuronal.
- ***batch_size***: Número de segmentos con los que se alimenta a la red de una vez, lo que determinará el número de iteraciones que se realiza en cada época de

entrenamiento (*epochs*). En este caso, si nuestro set de entrenamiento tuviera un tamaño de $[100, 1, 128, 20]$ y un *batch_size*=2, una época de entrenamiento constaría de 50 iteraciones en las que va viendo de 2 en 2 los segmentos de tamaño $[128, 20]$ del set de entrenamiento (Sáez Bombín, S., 2018). Después de cada *batch* se produce el ajuste de los pesos de la red, por lo que es un hiperparámetro importante y que tiene influencia en el problema del *overfitting*.

- ***learning_rate***: Tasa de entrenamiento. También es importante en el *overfitting*, ya que no debe ser muy grande para que no se ajuste únicamente a los datos de entrenamiento y tampoco muy pequeña para que no se quede en un mínimo local del gradiente del que no sea capaz de salir y no mejore el entrenamiento.

En el caso de trabajar con alguna capa con recurrente, hay que utilizar otro hiperparámetro adicional:

- ***units***: Indica el número de *units* (LSTM o GRU) que tendrá cada una de las capas recurrentes (Sáez Bombín, S., 2018).

Por último, al utilizar técnicas de regularización como el *Early Stopping* y *Adaptive Learning Rate*, hay que definir varios hiperparámetros:

- *Early Stopping*: Solamente utilizado cuando se está realizando validación durante el entrenamiento. Requiere el ajuste de un parámetro:
  - ***patience***: es el número de épocas que se espera para parar el entrenamiento (número de épocas consecutivas en las que la pérdida en validación no se reduce por debajo del mínimo alcanzado).
- *Adaptive Learning Rate*: Se puede utilizar para reducir la *learning rate* en función de la evolución de la pérdida en validación o de una manera fija, si no se está realizando validación. Requiere el ajuste de 3 parámetros:
  - ***factor***: es el número por el que se multiplica la *learning rate* cada vez que se cumple la condición establecida.
  - ***patience***: es el número de épocas que se espera para aplicar ***factor***.
  - ***min_lr***: mínimo valor de *learning rate* que se puede alcanzar

Como comentario adicional en esta parte, cabe destacar que tanto las redes utilizadas para clasificación como para generación a través de predicción (RNNs que realizan una regresión) realizan el proceso de entrenamiento de manera similar, es decir, al final van a comparar su resultado (etiqueta y segmento respectivamente) con los datos reales, mientras que en el caso de generación mediante GANs hay que ir entrenando al generador *G* y al discriminador *D* de manera alternativa.

Más específicamente, una GAN se entrena de la siguiente manera:

1. El generador *G* produce datos a partir de números aleatorios y estos datos generados, etiquetados como 0, se mezclan con datos reales, etiquetados como 1 y se pasan al discriminador *D*.
2. El discriminador *D* se entrena con los datos reales y los ficticios (generados) para ir ajustando sus pesos.
3. De nuevo, se generan datos con el generador *G*, pero esta vez se etiquetan como 1, es decir, como reales, con la intención de engañar al discriminador. Con estos

datos se entrena toda la GAN para que el generador *G* ajuste sus pesos (en este paso el discriminador *D* no realiza ningún ajuste de pesos, es decir, no entrena).

4.2.2 Evaluación de la red

Clasificación

La evaluación de la red arrojará un porcentaje indicando la precisión de la red, y será en base a esta precisión por lo que cambiaremos los valores de algunos de los hiperparámetros para un reentrenamiento de la red.

Para la parte de clasificación, en este trabajo Fin de Máster se han seguido los métodos de evaluación realizados en (Zhu, J. *et al*., 2017) que consisten en la técnica de ***K-fold cross-validation***. En esta técnica se divide el dataset en *k* bloques de igual tamaño, se entrena a la red en $k-1$ bloques y se realiza test con el bloque restante. Esto se repite para *k* redes en la que cada una de ellas tiene en el set de test uno de los *k* bloques, pasando así todos los bloques por el set de test. El resultado obtenido es la media de los resultados de cada una de las *k* redes. Y en (Zhu, J. *et al*., 2017) se enfoca desde dos puntos de vista, que son también las utilizadas en este Trabajo Fin de Máster:

El primero de ellos es un ***10-Fold random-partitioning***, lo que significa que se tienen 10 bloques de igual tamaño con datos de todos los sujetos mezclados en todos los bloques, que esta es la técnica que se utilizó en el Trabajo Fin de Grado (Sáez Bombín, S., 2018). Sin embargo, al igual que se indica en (Zhu, J. *et al*., 2017), al mezclar los datos de todos los sujetos y estar presente un mismo sujeto tanto en el set de entrenamiento como en el de test, puede ser que la red no aprenda únicamente características del movimiento, sino también características dependientes del sujeto, lo que le daría ventaja a la hora de hacer el test.

El segundo método es un ***12-Fold subject-wise***, es decir, se tienen 12 bloques de igual tamaño, pero en este caso, cada bloque es de un único sujeto, asegurándonos así de que se están evaluando las características de movimiento aprendidas por la red, y no las dependientes del sujeto, lo cual es más restrictivo, pero más fiable. En este método, en (Zhu, J. *et al*., 2017) ponderan cada uno de los resultados obtenidos por el número de muestras de las que disponen, ya que no todos los sujetos tienen el mismo número de muestras.

Sin embargo, en este Trabajo Fin de Máster, se ha realizado un **balanceo entre clases** con el objetivo de mejorar el entrenamiento, haciendo más fiable el resultado obtenido en la evaluación de la red. Este balanceo se ha utilizado tanto en el método *10-Fold random-partitioning* como en *12-Fold subject-wise*, utilizando para el entrenamiento 2375 ejemplos de cada sujeto, es decir, un sujeto se corresponde con un tensor de $[2375, 1, 128, features]$, con una media de 216 ejemplos por actividad o clase (hay ligeras variaciones en este número, por eso se indica la media). Se ha realizado tomando tantas muestras de actividad por cada sujeto como las que tiene de máximo el sujeto que menos tiene, lo cual también complica más el entrenamiento al tener menos datos.

Al igual que en (Zhu, J. *et al*., 2017), para medir si los resultados son estadísticamente significativos y fiables, se aplica el concepto de *Intervalo de Confianza (**IC**)*. Para calcularlo, primero se necesita el *Margen de Error* δ (52).

$$\delta = \pm Z \times \sqrt{\frac{P(100-P)}{N}} \tag{52}$$

Donde *Z* es una constante que, dependiendo de qué IC utilicemos, tomará un valor u otro; *P* es la precisión media obtenida y N es el número de instancias o ejemplos de las actividades en el dataset REALDISP.

Como se trata de una muestra grande ($N > 100$), según el Teorema del Límite Central, se debe utilizar un IC del 95%, lo que hará que $Z = 1.96$. Esto significa que podemos estar seguros al 95% de que nuestro resultado estará dentro de un intervalo de $P - \delta < P < P + \delta$. Como queremos que nuestro resultado sea fiable, ese IC y δ deben ser lo más pequeños posible a la vez que obtenemos un *P* grande (en el numerador de δ), por lo tanto, tenemos que utilizar un tamaño de muestra *N* que permita minimizar δ. El concepto de intervalo de confianza también se puede ver desde el punto de vista de (Zhu, J. *et al*., 2017), donde lo utilizan para determinar si las mejoras obtenidas con un experimento son significativas con respecto a otros experimentos con el mismo número de instancias. Esto se refiere a que si el resultado *P* de dos experimentos con el mismo número de instancias distan entre sí un valor mayor que el que marca el intervalo de confianza, se puede afirmar que esa diferencia es estadísticamente significativa.

En este sentido, al realizar un *K-Fold*, obtenemos una precisión no solamente sobre un set reducido de datos (el de test) sino sobre todo el dataset, ya que consideramos la media de todos los resultados. Así que, en el cálculo de δ, *N* será 28500 considerando las 11 actividades y el *Data Augmentation*. Esto hace que $\delta = 0.528\%$ en el peor de los casos de todos los *K-Fold* con *Data Augmentation* realizados, por lo que el IC será: $P - 0.528\% < P < P + 0.528\%$, haciendo que la media de la precisión de todo el *K-Fold* caiga en ese intervalo el 95% de las veces, por lo que podemos considerar que es un muy buen IC, que se ajusta mucho al resultado obtenido.

Un punto muy importante de hacer el balanceo entre clases es que también se obtienen resultados fiables clase a clase, ya que se tienen aproximadamente el mismo número de instancias de cada clase, por lo que todas van a tener el mismo peso en el resultado final.

Para terminar, el proceso completo de la obtención de una red que arroje un resultado óptimo en clasificación mediante la evaluación *K-Fold* y que se ha seguido en este Trabajo Fin de Máster ha sido:

1. Obtención de una red (modificando arquitectura e hiperparámetros) que arroje un buen resultado utilizando únicamente un bloque en test, utilizando validación durante el entrenamiento, *Early Stopping* y *Adaptive learning rate* en función de la pérdida en validación.
2. Realización del *K-Fold* con esa red. Si no es del todo satisfactorio, se busca el bloque que ha supuesto un peor rendimiento (debido a la variabilidad entre clases que se tiene).

3. Nuevo ajuste de hiperparámetros y de arquitectura para mejorar el rendimiento en ese bloque.
4. Nuevo *K-Fold* sobre todo el dataset con la nueva red, generalmente obteniendo mejores resultados que antes.

Generación

En el caso de generación, dependiendo de si se utilizan GANs o RNNs (predicción) se utiliza una forma de evaluación u otra.

Si se trabaja con GANs la evaluación de la red es totalmente visual, es decir, una vez entrenada, se extrae el generador y se utiliza para generar nuevos datos a partir de ruido. Estos datos generados se pasan al programa de Unity que se utilizó para revisar los datos inicialmente y se ve cómo de correctos son los datos sintéticos.

Al trabajar con RNNs en modo de predicción, se tiene un set de test, con el que se va a poder comparar el resultado de la red (como se hacía en clasificación, pero en vez de comparar etiquetas lo que se compara son segmentos de datos). En este caso sí se tiene una métrica que nos va a permitir saber cómo de cerca está lo predicho por la red de lo real, esta métrica que se utiliza es la **RMSE** (*Root Mean Square Error*) de los valores predichos $\widehat{y_t}$ para t veces la regresión de la variable dependiente y_t, es decir, como se muestra en la ecuación (53).

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\widehat{y_t} - y_t)^2}{T}} \tag{53}$$

Cuanto menor sea esta RMSE, mejor es el resultado, que, además, también se revisa visualmente a través del programa de Unity.

4.3 Arquitecturas de las redes y resultados

A continuación, se presentan los resultados de este Trabajo Fin de Máster, separando para las partes de ***Clasificación*** y ***Generación*** de igual manera que en las secciones anteriores.

Se presentan más detalladamente únicamente las redes en su mejor versión, es decir, para un determinado número y tipo de capas, así como los parámetros y estructura que permiten un mayor porcentaje de acierto.

La notación que se va a seguir para las distintas estructuras es la utilizada en (Ordóñez, F. J. y Roggen, D., 2016), y consiste en:

- $C(F)$: Se refiere a una capa convolucional con *F feature maps*, es decir, el número de *kernels* utilizados.
- $CT(F)$: Es una capa convolucional traspuesta con *F feature maps*.
- $R(n)$: Se refiere a una capa recurrente con *n units* de tipo LSTM.
- $RG(n)$: Se refiere a una capa recurrente con *n units* de tipo GRU.
- $DR(d)$: Se refiere a una capa de Dropout que descarta resultados con una probabilidad *d*.
- $D(n)$: Capa *fully-connected* con *n units* en la que solamente se introduce una activación no lineal.
- S_m: Capa *Dense* que se refiere a la última capa. Constituye una capa *fully-connected* en la que se fusionan todos los resultados de la capa previa, junto con la aplicación de la fusión *softmax*.

Antes de comenzar a ver las estructuras de las redes y sus detalles, se quieren hacer dos comentarios que valen para todos los tipos de arquitecturas de red probadas:

- En primer lugar, se quiere destacar que siempre se han realizado las pruebas con un número de *units* múltiplo de 32. Esto se debe a que en CUDA la unidad ejecutable más pequeña de paralelismo es de 32 hilos (lo que se conoce como un *warp* de hilos, implementado en hardware), con lo que el *warpsize*, es 32, mejorando así la velocidad del entrenamiento (NVIDIA Corporation, 2018).
- Por otra parte, en (Baños, O. *et al*. 2014b, pp. 9995-10023) se utiliza una ventana de 6 segundos sin *overlap*, pero esto nos generaría muy pocos segmentos, y por ello, datos, con respecto a lo que se requiere para el entrenamiento de una red neuronal. Para solucionar esto se utiliza un tamaño de ventana de 128 muestras, es decir, 2.56 segundos, con un *overlap* del 75% para el caso de ***Clasificación***, y un 75% y 99% para distintos casos de ***Generación con GANs***, por el contrario, para ***Generación con RNN (predicción)*** no hay *overlap*. Este tamaño de ventana nos permite tener todos los datos cargados en RAM durante la ejecución de las redes y recoge un ciclo significativo de la actividad. Además, este tamaño de ventana ha sido elegido acorde a las dimensiones de la red y el número de *features*.

4.3.1 Resultados de Clasificación

En esta parte del Trabajo Fin de Máster se han utilizado 2 tipos de redes neuronales:

- **CNN (*Convolutional Neural Networks*)**: Redes completamente convolucionales, con un *kernel* de tamaño 4x4, siguiendo los resultados obtenidos en (Sáez Bombín, S., 2018).
- **CNN+RNN (*CNN+Recurrent Neural Networks*)**: Basadas en trabajos de reconocimiento de actividades que utilizan técnicas de Deep Learning, como (Ordóñez, F. J. y Roggen, D., 2016). Las primeras capas de estas redes con convolucionales, y las últimas recurrentes. De esta forma, las capas convolucionales realizan un preprocesamiento de los datos, por lo que podrían sacar las relaciones espaciales de los datos dejando las relaciones temporales a las capas recurrentes, y así facilitar y mejorar el rendimiento y resultado.

La forma en que se presentan los mejores resultados para cada tipo de red (CNN y CNN+RNN) es de la siguiente manera:

1. Se presenta el ajuste de hiperparámetros llevado en la red.
2. Se visualiza y comenta la arquitectura de la red.
3. Se añade la gráfica de evolución de la *learning rate* durante la fase de entrenamiento.
4. En el ANEXO I se muestran las matrices de confusión de cada una de las iteraciones de los *K-Fold*, es decir, para todos los casos. La matriz de confusión permite ver el rendimiento de una red (o de un algoritmo de Machine Learning), mostrando en cada fila el número de ejemplos de la clase verdadera y en las columnas el número de veces que la red o algoritmo ha predicho una clase, es decir, un elemento de la matriz sería el número de veces que la red predice una actividad *X*, siendo verdadera la actividad *Y* (con *X* la fila e *Y* la columna). Por lo tanto, este tipo de matrices interesa que sean diagonales, lo que significaría que la red siempre predice correctamente la actividad (Sáez Bombín, S., 2018).

La Tabla 1 presenta los resultados obtenidos en Clasificación y, en ella, caben destacar 3 puntos:

1. Hay una tendencia generalizada de que las redes CNN+RNN obtienen mejores resultados que las puramente CNN. Esto refuerza la idea presentada de que las capas convolucionales están "extrayendo" las relaciones espaciales de los datos (como si de una imagen se tratara) y las capas recurrentes obtienen información de las relaciones temporales, por ello, en este caso en el que los datos presentan relaciones tanto espaciales como temporales, parece lógico que las redes que combinan ambas capas obtengan mejor rendimiento.
2. A medida que se aumenta el número de datos y de *features*, el rendimiento mejora.
3. El método de evaluación *12-Fold subject-wise* es más restrictivo que el *10-Fold random-partitioning*, algo que ya se introdujo en la sección 4.2.2 y que aquí queda demostrado, además de las evidencias de otros trabajos que también realizan distinciones semejantes (Zhu, J. *et al*., 2017). Por lo tanto, se pueden concluir 2 aspectos: en primer lugar, que la red tiende a aprender las características de cada

sujeto, no solamente del movimiento, y, en segundo lugar, que a través del *12-Fold subject-wise* se puede tomar por seguro que las redes de este Trabajo Fin de Máster reconocen un mismo movimiento en distintos sujetos.

Debido a este punto 3, tanto la mejor red CNN como la mejor CNN+RNN que se presentan más adelante en esta sección, se explican desde el punto de vista del método *12-Fold subject-wise*, que se considera que es el más correcto de cara a un futuro uso de la red dentro de un sistema que sea independiente del sujeto. Por lo tanto, la evaluación *10-Fold random-partitioning* solamente se incluye para demostrar lo que se indica en el aspecto 3.

Por lo tanto, en la Tabla 1 se presentan los resultados (exactitud, que se corresponde con el habitual *accuracy* de los trabajos de Machine Learning) distinguiendo por tipo de red, tipo de evaluación (*10-Fold random-partitioning* y *12-Fold subject-wise*) y tipo de preprocesamiento que se hace, que recordando el apartado 4.1.3, estos tipos son:

1. Datos "en crudo".
2. Datos "en crudo" y con *Data Augmentation*.
3. Datos "en crudo", con *Data Augmentation* y calculando la DFT-2D.
4. Datos normalizados y con *Data Augmentation*.
5. Datos normalizados, con *Data Augmentation* y calculando la DFT-2D.
6. Datos únicamente de la DFT-2D.

Los resultados mostrados son los obtenidos entrenando con los datos modificados como en la lista anterior, pero cabe destacar que los datos de test que se han utilizado en esas pruebas están también modificados como en la lista anterior, cuando deberían ser con los datos originales del dataset (solamente en test), es decir, cada sujeto orientado hacia una dirección, no todos hacia la misma, y sin *Data Augmentation* (tipos de preprocesamiento 2, 3, 4, 5).

Esta modificación solamente se ha realizado para tener más seguridad a la hora de tomar decisiones en cuanto al diseño de la arquitectura de red, y obtener resultados más significativos (por ejemplo, en los casos en los que hay *Data Augmentation*, realizar el test con datos también aumentados hace que los intervalos de confianza sean más pequeños, por lo que se puede tener una mejor idea de qué mejoras son más significativas estadísticamente).

Sin embargo, para obtener unos resultados más realistas, se realiza también el test con los datos originales los tipos de preprocesamiento 3 y 5, que son los que mejor resultado han ofrecido. Así que, en este test, los sujetos están orientados cada uno en su dirección original y no han sido aumentados. Estos resultados son los que están marcados con un asterisco (*) y son los que se deben tomar para tener una visión más realista del rendimiento de la red.

Los mejores resultados de cada red (en esta última forma de test) se muestran en negrita y cursiva. El resultado en rojo es el mejor obtenido en todo el Trabajo Fin de Máster.

Tipo	Red	Preproc.	Exactitud			
			10-Fold random-partitioning		12-Fold subject-wise	
CNN	C(16)-C(32)-S_m	1	90.99%		78.06%	
		2	94.71%		70.77%	
		3	98.75%	***97.5%****	89.09%	***93.28%****
		4	94.4%		70.85%	
		5	98.65%	96.11%*	89.35%	92%*
		6	95.77%		71.11%	
	C(16)-C(32)-C(64)-C(128)-DR(0.5)-S_m	1	91.35%		77.89%	
		2	97.98%		91.87%	
		3	98.64%	***98.51%****	94.03%	95.58%*
		4	97.16%		92.69%	
		5	98.08%	97.5%*	93.95%	***95.66%****
		6	96.68%		79.78%	
CNN+ RNN	C(32)-C(64)-C(32)-DR(0.5)-RG(64)-S_m	1	92.65%		83.33%	
		2	96.26%		89.5%	
		3	98.67%	96.48%*	95.08%	95.58%*
		4	98.35%		86.83%	
		5	99.32%	***97.13%****	93.72%	***95.66%****
		6	96.35%		82.04%	
	C(32)-C(64)-C(32)-DR(0.5)-RG(96)-S_m	3	99%	***98.05%****	93.51%	94.39%*
		5	98.78%	97.22%*	95.2%	***96.26%****
	C(64)-C(64)-C(64)-DR(0.5)-RG(64)-S_m	1	94.23%		83.67%	
		2	98.14%		94.58%	
		3	99.47%	99.16%*	96.27%	***97.7%****
		4	98.65%		92.08%	
		5	99.36%	***99.44%****	95.18%	96.85%*
		6	97.54%		82.75%	
	C(64)-C(64)-C(64)-DR(0.5)-RG(128)-S_m	3	99.21%	98.98%*	96.21%	97.36%*
		5	99.32%	***99.08%****	95.89%	***97.45%****

Tabla 1. *Resultados de clasificación.*

Como se puede ver en esta Tabla 1, en los casos en los que se entrena con más datos y más *features* obtienen mejores resultados, y la combinación *features* temporales y frecuenciales (tipos 3 y 5) obtiene mejores resultados que utilizando únicamente un tipo de ellas (tipos 1, 2, 4 y 6).

Por otro lado, no se aprecia que haya una gran diferencia en el rendimiento de las redes entre utilizar datos normalizados o sin normalizar (de manera general), aunque el mejor resultado en *12-Fold subject-wise*, que es el método que nos interesa y el más realista, se obtiene con los datos sin normalizar, **97.7%**.

Como último comentario de esta tabla, cabe destacar otro aspecto que refuerza la idea de tener en cuenta el método *12-Fold subject-wise* frente al *10-Fold random-partitioning*.

En el método *12-Fold subject-wise*, además de asegurarnos de que la red nos arroja un rendimiento independiente del sujeto que realiza los movimientos, a la hora de evaluarla con los datos originales, se obtiene un rendimiento aún mejor, a diferencia del caso *10-Fold random-partitioning*, que empeora en la gran mayoría de los casos. Esto solamente refuerza lo comentado anteriormente, que la red aprende mejor el movimiento con el método *12-Fold subject-wise*.

A continuación, se presentan otras métricas de interés que se han sacado. Para presentarlas, primero es necesario conocer una serie de conceptos, ligados a todas las métricas:

- **Positivos verdaderos** (*True Positives, TP*): Son los valores tomados como positivos, siendo verdadera la condición positiva. Ejemplo: una instancia es clasificada como "Actividad X" siendo esa instancia la actividad X.
- **Negativos verdaderos** (*True Negatives, TN*): Son los valores tomados como negativos, siendo verdadera la condición negativa. Ejemplo: una instancia es clasificada como "Actividad Y", siendo esa instancia una actividad distinta a X.
- **Falsos positivos** (*False Positives, FP*): Son los valores tomados como positivos, siendo verdadera la condición negativa. Ejemplo: una instancia es clasificada como "Actividad X" siendo esa instancia una actividad distinta a X.
- **Falsos negativos** (*False Negatives, FN*): Son los valores tomados como negativos, siendo verdadera la condición positiva. Ejemplo: una instancia es clasificada como "Actividad Y", siendo esa instancia la actividad X.

En concreto, las métricas mostradas son:

- **Precisión** (*precision* o *positive predictive value, PPV*): Fracción de instancias realmente positivas de entre las instancias seleccionadas como positivas (54). Responde a la pregunta: ¿Cuántas instancias son positivas de todas las seleccionadas como positivas?

$$Precisión = \frac{TP}{TP + FP} \tag{54}$$

- **Sensibilidad** (*sensitivity*, *recall*, *hit rate* o *true positive rate, TPR*): Fracción del total de instancias que fueron seleccionadas como positivas siendo positivas de entre todas las instancias realmente positivas (55). Respondería a la pregunta: De todas las instancias realmente positivas, ¿cuántas han sido seleccionadas como positivas?

$$Recall = \frac{TP}{TP + FN} \tag{55}$$

- ***F1-score***: Considera la precisión y la sensibilidad, y es la media armónica de estas dos, siendo 1 en el mejor de los casos (precisión y sensibilidad perfectas) y 0 en el peor) (56).

$$F_1 = 2 \times \frac{PPV \times TPR}{PPV \times TPR} = \frac{2TP}{2TP + FP + FN} \quad (56)$$

- **Especificidad** (*specifity*, *selectivity* o *true negative rate*, *TNR*): Mide la proporción de negativos que son identificados correctamente como tal (57). Responde a la pregunta: De los elementos seleccionados como negativos, ¿cuántos son realmente negativos?

$$Especificidad = \frac{TN}{TN + FP} \quad (57)$$

- **Exactitud** (*accuracy*): Proporción de las instancias correctamente seleccionadas de entre todas las instancias del problema (58).

$$Exactitud = \frac{TP + TN}{TP + TN + FP + FN} \quad (58)$$

La Tabla 2 y Tabla 3 presentan estas métricas para la mejor red CNN y CNN+RNN respectivamente, con el método *12-Fold subject-wise* y los dos tipos de preprocesamiento que mejor resultado han dado, el 3 y 5.

Preprocesamiento	Precisión			Recall			F1-score			Especificidad			Exactitud
	Clase		Total	Clase		Total	Clase		Total	Clase		Total	
3	9	1	0.951	9	0.99	0.956	9	0.996	0.947	9	1	0.996	95.58%
	10	0.92		10	0.92		10	0.92		10	0.999		
	11	0.93		11	0.99		11	0.96		11	0.99		
	12	0.97		12	0.91		12	0.91		12	0.995		
	13	0.95		13	0.98		13	0.96		13	0.991		
	19	0.99		19	1		19	0.996		19	0.999		
	20	0.88		20	0.92		20	0.89		20	0.993		
	21	1		21	1		21	1		21	1		
	24	0.95		24	1		24	0.97		24	0.993		
	25	1		25	0.89		25	0.93		25	1		
	31	0.87		31	0.92		31	0.89		31	0.992		
5	9	0.97	0.96	9	0.94	0.956	9	0.94	0.95	9	0.995	0.996	95.66%
	10	0.967		10	0.99		10	0.97		10	0.994		
	11	0.943		11	1		11	0.96		11	0.99		
	12	0.88		12	0.92		12	0.89		12	0.994		
	13	1		13	0.98		13	0.99		13	1		
	19	1		19	0.95		19	0.968		19	1		
	20	0.9		20	0.92		20	0.91		20	0.997		
	21	1		21	1		21	1		21	1		
	24	0.95		24	0.97		24	0.95		24	0.993		
	25	0.995		25	0.92		25	0.94		25	0.998		
	31	0.954		31	0.93		31	0.92		31	0.991		

Tabla 2. *Métricas para la mejor CNN.*

Preprocesamiento	Precisión			Recall			F1-score			Especificidad			Exactitud
	Clase		Total	Clase		Total	Clase		Total	Clase		Total	
3	9	1	0.978	9	1	0.977	9	1	0.973	9	1	0.998	97.7%
	10	1		10	0.98		10	0.99		10	2		
	11	0.992		11	1		11	0.996		11	0.999		
	12	0.992		12	1		12	0.996		12	0.999		
	13	0.992		13	0.99		13	0.991		13	0.999		
	19	0.992		19	1		19	0.996		19	0.999		
	20	0.992		20	0.91		20	0.891		20	1		
	21	1		21	1		21	1		21	1		
	24	0.952		24	0.94		24	0.93		24	0.993		
	25	0.97		25	0.98		25	0.97		25	0.994		
	31	0.96		31	0.94		31	0.92		31	0.992		
5	9	1	0.965	9	1	0.968	9	1	0.962	9	1	0.997	96.85%
	10	1		10	0.94		10	0.958		10	1		
	11	1		11	1		11	0.976		11	1		
	12	1		12	1		12	1		12	1		
	13	0.94		13	1		13	0.964		13	0.99		
	19	0.98		19	1		19	0.985		19	0.996		
	20	0.87		20	0.88		20	0.87		20	0.993		
	21	1		21	1		21	1		21	1		
	24	0.96		24	1		24	0.98		24	0.994		
	25	1		25	0.97		25	0.985		25	1		
	31	0.873		31	0.89		31	0.88		31	0.992		

Tabla 3. *Métricas para la mejor CNN+RNN.*

Por último, se presentan más en detalle la mejor red de cada tipo, CNN y CNN+RNN:

CNN

La red **CNN** que mejor resultado ha dado es la que ha obtenido un 94.03% de precisión. La arquitectura de dicha red es *C(16)-C(32)-C(64)-C(128)-DR(0.5)-* S_m, mostrada en la Figura 45 (para el caso de preprocesamiento con el que se obtiene mejor resultado). Esta red presenta los hiperparámetros mostrados en (59).

$$\begin{gathered} batch\ size = 128 \\ learning\ rate\ inicial = 9 \times 10^{-4} \\ epochs = 100 \end{gathered} \tag{59}$$

Cada una de las capas convolucionales *C(F)* van seguidas de un *max-pooling* 2×2 y un *Batch Normalization* y tras la última capa convolucional se introduce una capa de Dropout de 0.5.

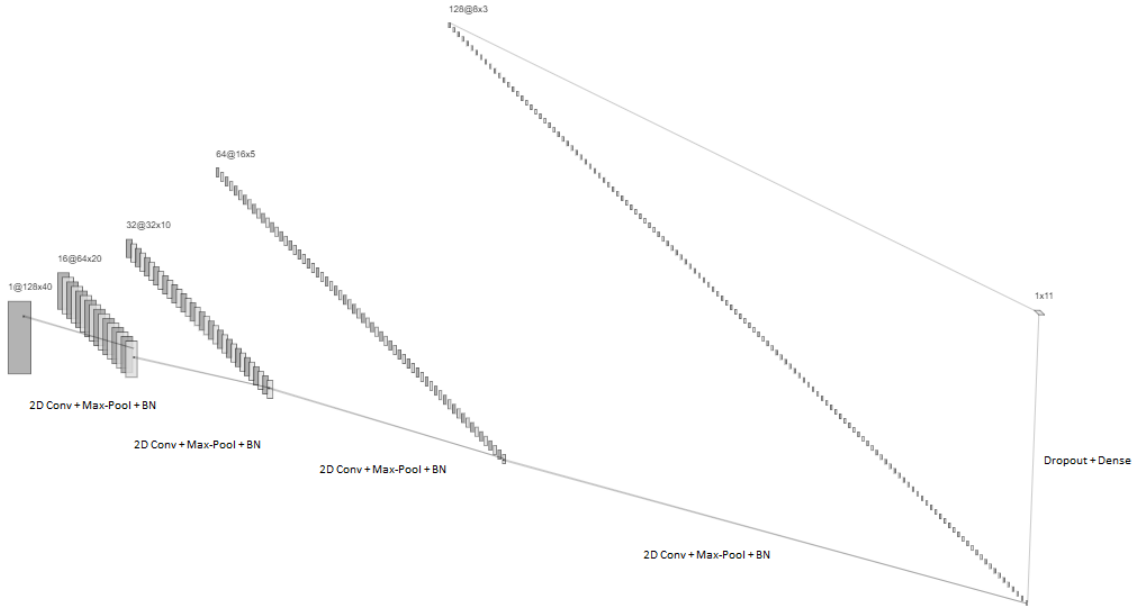


Figura 45. *Arquitectura de la CNN*

En la Figura 45 se puede ver que el tamaño de los tensores comienza siendo $1@128 \times 40$, correspondiente a las últimas 3 dimensiones del tensor de entrada $[filas, 1, 128, 40]$. A medida que aumenta el número de *feature maps F* se va reduciendo el tamaño de las matrices, para terminar como salida con un vector que indica en formato *One-hot encoding* la clase en la que ha clasificado a una determinada matriz de datos.

Además, se utiliza el optimizador *Adam* con valores de $\beta_1 = 0.9$, $\beta_2 = 0.999$ y una *learning rate* inicial de 9×10^{-4} que se va reduciendo por un factor de 0.8 cada 10 épocas de entrenamiento, lo que se muestra en la Figura 46.

La evolución de la *learning rate* quedaría como sigue:

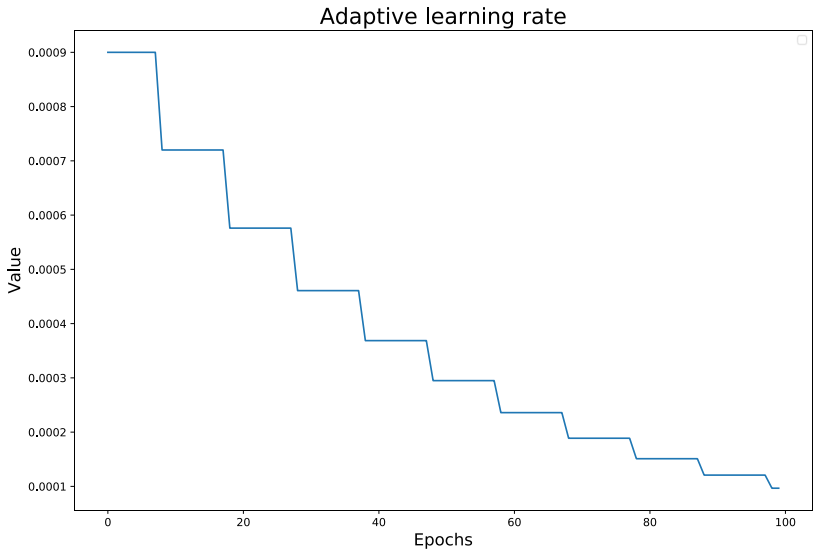


Figura 46. *Evolución de la learning rate durante el entrenamiento.*

Las matrices de confusión de todo el *12-Fold subject-wise* se incluyen en el ANEXO I.

CNN+RNN

En el caso de las **CNN+RNN** cabe destacar la red *C(64)-C(64)-C(64)-DR(0.5)-RG(64)-*S_m**,** mostrada en la Figura 47 y con la que se ha obtenido un 96.5% de precisión, el valor más alto de todo el Trabajo Fin de Máster, dentro del método de evaluación *12-Fold subject-wise*. Los hiperparámetros utilizados son los mostrados en (60).

$$\begin{gathered} batch\ size = 128 \\ learning\ rate\ inicial = 10^{-3} \\ epochs = 100 \end{gathered} \tag{60}$$

Al igual que en la CNN las capas convolucionales *C(F)* van seguidas de un *max-pooling* y un *Batch Normalization*, siendo el *max-pooling de* 2×2 para la primera capa, y de 4×4 para las dos siguientes convolucionales, introduciendo un Dropout de 0.5 justo antes de la entrada a la capa recurrente.

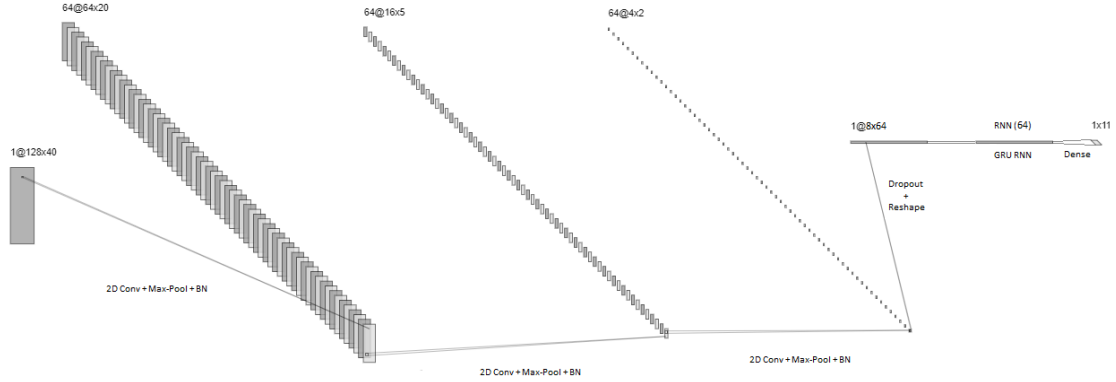


Figura 47. *Arquitectura de la CNN+RNN.*

Para esta situación, también se utiliza el optimizador *Adam* con valores de $\beta_1 = 0.9$, $\beta_2 = 0.999$ pero con una *learning rate* inicial de 10^{-3} que se va reduciendo durante el entrenamiento por un factor de 0.4 cada 10 épocas de entrenamiento, teniendo el límite también en una *learning rate* de 10^{-9} (Figura 48).

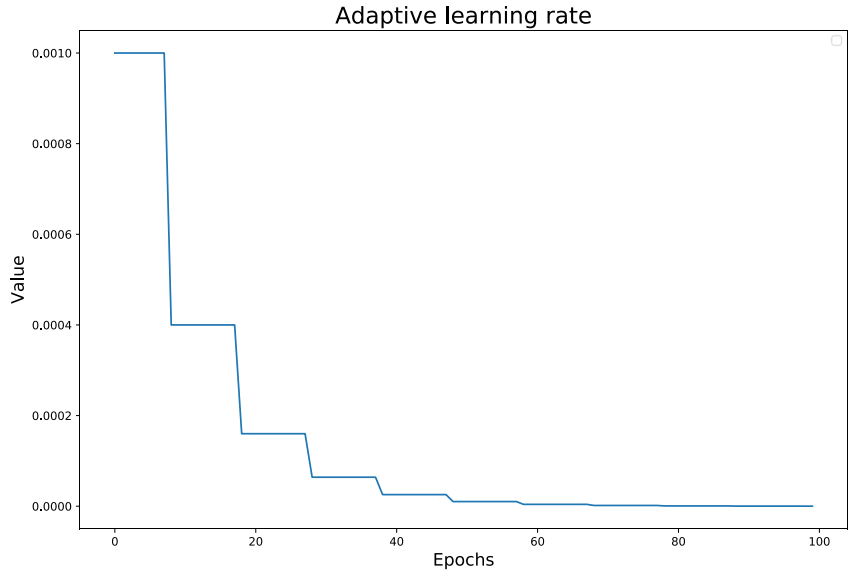


Figura 48. *Evolución de la learning rate durante el entrenamiento.*

Las matrices de confusión de todo el *12-Fold subject-wise* se incluyen en el ANEXO I.

Se deja un enlace en el que se accede a un vídeo para cada uno de los *12-Fold*, es decir, se tienen 12 vídeos para la CNN y 12 vídeos para la CNN+RNN. En ellos se puede ver cómo rinde cada una de las dos redes dejando fuera del entrenamiento uno de los sujetos cada vez. El número indicado en el nombre de los vídeos se refiere al sujeto que se ha dejado para test (fuera del entrenamiento) en esa iteración del *12-Fold*.

Una vez revisado el rendimiento de las redes, para terminar con el apartado de Clasificación, habría que comprobar su viabilidad dentro de un sistema embebido, con aplicaciones en tiempo real.

Para ello, es necesario revisar el tiempo de inferencia por cada ejemplo de la red, es decir, cuánto tarda la red en clasificar un ejemplo de tamaño $[1, 1, 128, features]$. Teniendo en cuenta que los mejores resultados han sido alcanzados con una CNN+RNN utilizando el método 3 de preprocesamiento de datos, es decir, no se normalizan los datos y es necesario calcular la DFT-2D, hay que tener en cuenta el tiempo de inferencia de dicha red ($t_{inferencia}$) y sumarle el tiempo que el ordenador tarda en calcular la DFT-2D de un segmento de tamaño $[1, 1, 128, 20]$ y lo añade al segmento de datos ($t_{DFT+concat}$). Estos dos tiempos son:

- $t_{inferencia} = 248\ \mu s$.
- $t_{DFT+concat} = 196\ \mu s \pm 230\ ns$. Este resultado consiste en una media y una varianza, ya que se ha realizado la operación para 10000 ejemplos y así obtener un resultado fiable.

Esto hace que la ecuación de tiempo total sea (61).

$$t_{TOTAL} = t_{inferencia} + t_{DFT+concat} = 444\ \mu s\ \pm 230\ ns \quad (61)$$

Por lo tanto, inicialmente, el sistema requeriría de $2.56\ s + t_{TOTAL}$ iniciales de movimiento para poder realizar su primera clasificación (segmento de tamaño $[1, 1, 128, 40]$), y una vez realizada, se podrían construir nuevos segmentos con cada nueva muestra que llegue, tomando las 127 muestras anteriores y añadiéndole la nueva. Teniendo en cuenta que en el problema HAR, habitualmente las muestras son tomadas por los sensores con una frecuencia de $f_s = 50\ Hz$, es decir, cada $t_s = \frac{1}{f_s} = 20\ ms$, hace que se pueda asegurar que no habría problema en incluir esta red en un sistema embebido de tiempo real, ya que $t_s \gg t_{TOTAL}$ y no habría ningún retardo mayor que el de la propia toma de muestras y envío de datos, que se considera suficientemente rápido.

4.3.2 Resultados de Generación

A lo largo de esta memoria se ha indicado que la generación de datos sintéticos se aborda de dos formas en este Trabajo Fin de Máster, con sus respectivas formas de evaluación.

GAN

Desde la perspectiva de las GANs se han obtenido varias redes que han dado resultados satisfactorios (solamente se ha probado con la actividad 19). A continuación, se presentan las arquitecturas de GAN que se han utilizado y la evolución de su entrenamiento:

a) Tanto el Generador *G* como el Discriminador *D* son MLPs, es decir, redes que únicamente tienen capas *fully-connected* (en estas redes las *units* solamente introducen una activación no lineal).
   En concreto, en esta red se tiene:
   A. *G* consta de 3 capas *D(256)-D(512)-D(1024)*.
   B. *D* tiene 4 capas *D(1024)-D(512)-D(256)-D(1)*.
   C. Esta GAN tiene 6.6 millones de parámetros a ajustar.
   D. frece un resultado ya satisfactorio, aunque con una sensación de "ruido".

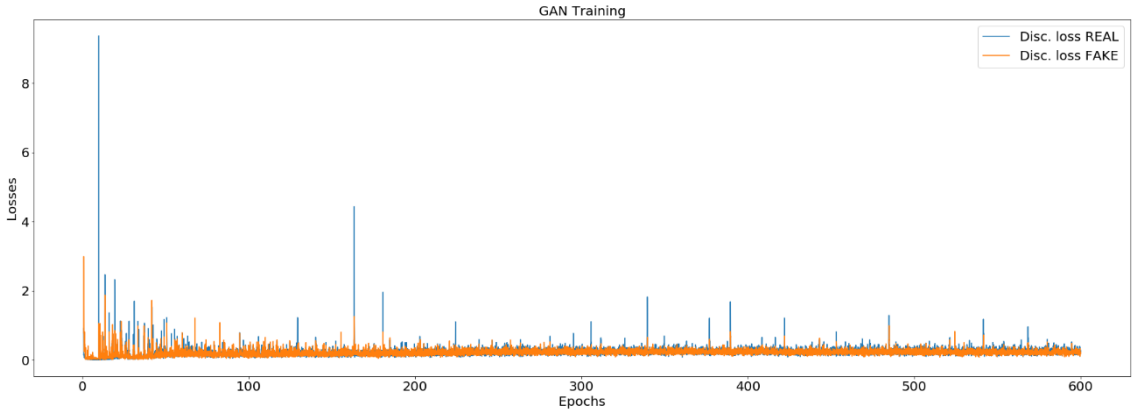


Figura 49. *Evolución de las pérdidas en la GAN durante el entrenamiento.*

b) Con el objetivo de eliminar ese “ruido" y obtener un mejor resultado se prueba con redes más complejas. Para este caso:
   A. *G* es una RNN con 3 capas GRU de *RG(1000)-RG(1000)-RG(2560)*.
   B. *D* es una *fully-connected* de 5 capas, *D(2560)-D(512)-D(256)-D(128)-D(1)*.
   C. Esta GAN, debido a la introducción de una RNN como generador, tiene 22.6 millones de parámetros a ajustar, afectando al tiempo de entrenamiento de esta.
   D. Sin embargo, el resultado tiene mayor calidad que en el caso anterior, aunque sigue apareciendo algo de ese “ruido”.

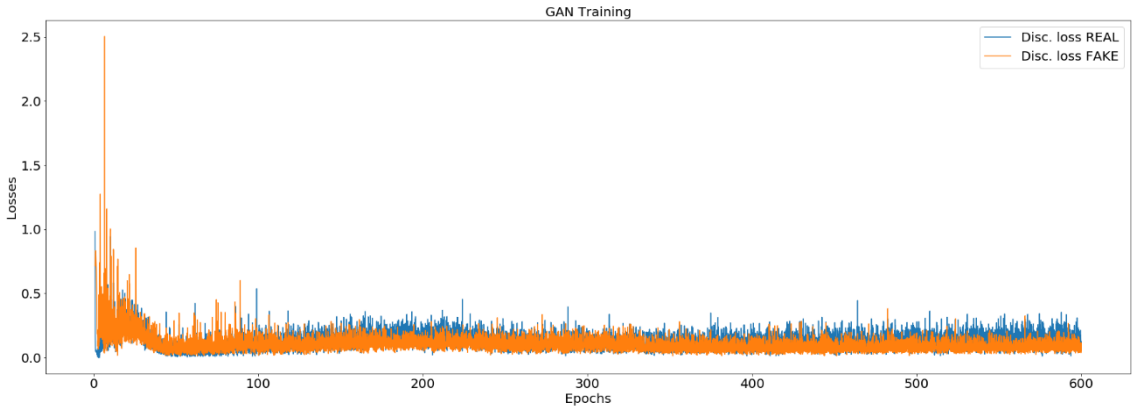


Figura 50. *Evolución de las pérdidas en la GAN durante el entrenamiento.*

c) Siguiendo la línea anterior, se buscan otros tipos de redes para obtener mejor resultado. Así:
   A. Se mantiene el *G* del caso b).
   B. *D* se sustituye por una red con una red convolucional con dos capas *fully-connected* (siendo una de ellas la que realiza la clasificación): *D(2560)-C(64)-C(128)-C(512)-D(1)*.
   C. Como se trata de 2 redes más complejas, los parámetros de la red pasan a ser 31.7 millones de parámetros.
   D. El resultado es muy satisfactorio en este caso.

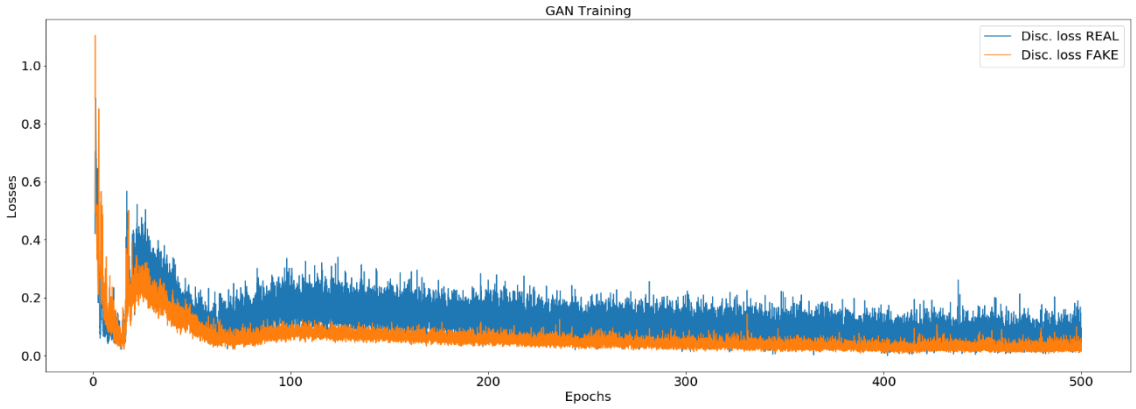


Figura 51. *Evolución de las pérdidas de la GAN durante el entrenamiento.*

d) Como última prueba que da buenos resultados, se presenta una siguiendo las ideas de (San José, R., 2019): usar como *G* el *decoder* de un *autoencoder* previamente entrenado, y usar como *D* una red también previamente entrenada. De esta manera se tiene:

   A. Se opta por un *autoencoder* con una arquitectura *C(32)-C(64)-D(100)-CT(64)-CT(32)*, en la que el *decoder* es *CT(64)-CT(32)* que funciona como *G*.
   B. Como *D* se utiliza una MLP de *D(2048)-D(1024)-D(512)-D(256)-D(128)-D(1)*.
   C. El uso de una MLP en *D* y una red convolucional tan pequeña como *G* hace que el tamaño de la red se reduzca hasta los 9.1 millones de parámetros.
   D. Además, se obtienen muy buenos resultados, mejores que en los anteriores casos.

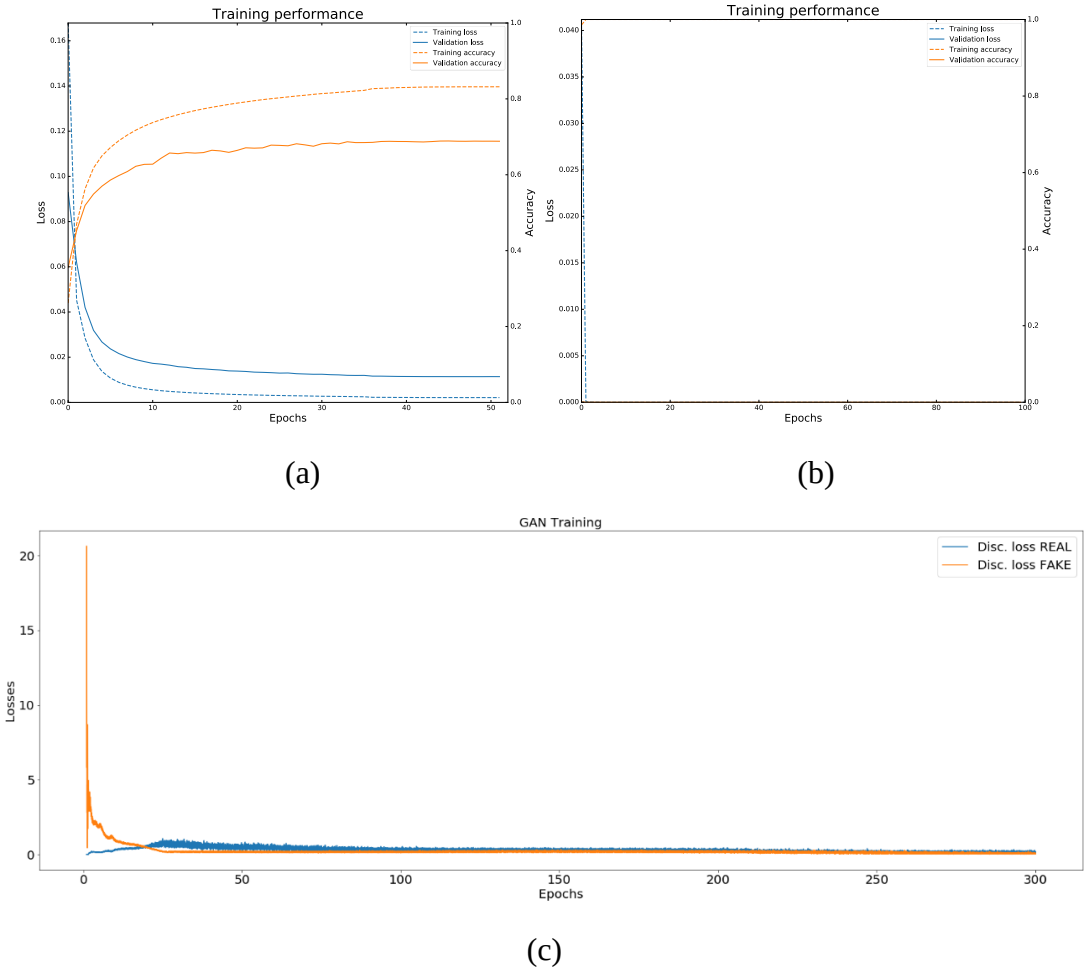


Figura 52. *Evolución de: (a) preentrenamiento del generador G, (b) preentrenamiento del discriminador D, (c) entrenamiento de la GAN.*

A continuación, en la Figura 53 se muestran una serie de capturas de los datos generados para esta última red, que es la que mejor resultado ha dado. Además, se deja un enlace para poder visualizar unos vídeos en los que se puede ver la evolución del rendimiento de esta última GAN para ciertas épocas clave del entrenamiento.

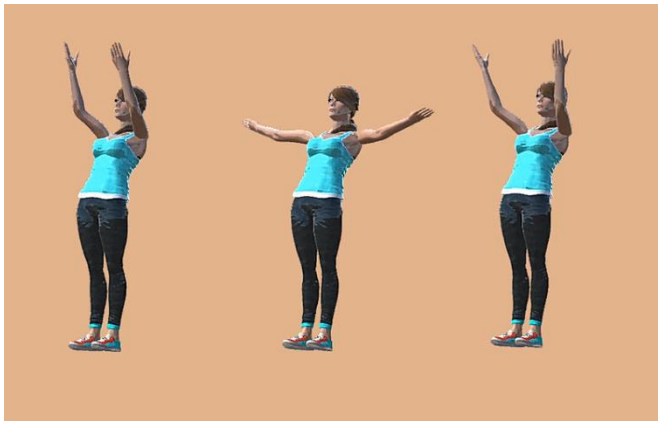


Figura 53. *Capturas del resultado de la generación con GAN.*

Para todos los casos, los resultados son visualizados a través de un programa en Unity, de donde se han podido sacar los vídeos de los enlaces indicados. En este programa se puede ver que en todos ellos hay presente en mayor o menor medida algo de "ruido", pero se consigue eliminar a través de una interpolación circular, que consiste en encontrar los puntos intermedios entre dos dados (azul y verde en la Figura 54), mediante una trayectoria circular (línea roja en la Figura 54), mientras que, en el caso de una interpolación lineal, esa trayectoria sería una recta (línea negra en la Figura 54).

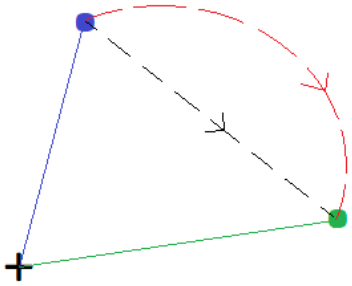


Figura 54. *Interpolación circular.*

En las Figuras 49-52 se puede ver la evolución del entrenamiento de las distintas GAN. En ellas se muestran la pérdida del *D* frente a los datos reales (REAL) y frente a los datos generados por el *G* (FAKE). Idealmente, estas dos pérdidas se van a equilibrar, siendo muy cercanas una a otra y sin ser 0, momento en el que se dice que la GAN ha convergido.

Además, en la Figura 52.a) y 52.b) se puede ver el preentrenamiento realizado a *G* y *D* respectivamente antes de entrenarlos como GAN. Este preentrenamiento no es muy exhaustivo, ya que lo que se busca es guiar a ambas redes, especialmente a *G* hacia la distribución de datos que se va a utilizar en la GAN, y así favorecer la convergencia de esta.

Por último, caben destacar una serie de comentarios que son de interés, y que han ayudado a conseguir resultados satisfactorios con distintos tipos de redes:

- La función de activación de la capa de salida del generador *G* es *tanh*, es decir, tangente hiperbólica.
- Tanto en generador *G* como en discriminador *D* se utilizan funciones de activación *LeakyReLU* (menos en la salida de *G*).
- Los datos aleatorios que se introducen en el generador provienen de una distribución Gaussiana.
- En el generador, toda capa convolucional va seguida de una capa de *Batch Normalization* además de la función de activación *LeakyReLU*.
- En el generador, en vez de utilizar una capa de *pooling* (ya se *max-pooling* o *average-pooling*), se realiza lo que se conoce como *strided convolutions*, es decir, aumentan el tamaño de la entrada mediante el desplazamiento del propio *kernel*.

RNN

Por otro lado, desde el punto de vista de las RNNs predictoras solamente se hace una introducción con alguna, como una vía posible para generar datos en un futuro.

En este caso sí se tiene una métrica que permite ver el rendimiento de la red sin necesidad de visualización de los datos en el programa de Unity. Esta métrica es el RMSE, y se ha obtenido hasta un valor de 0.03 en los datos de test, es decir, prácticamente se realiza una predicción perfecta de los datos.

En definitiva, con estas redes lo que se está realizando es una regresión de una serie temporal multivariante, al tener 20 *features* (cada uno de los 4 ejes de los 5 sensores utilizados).

La red con la que se han obtenido resultados satisfactorios en predicción ha sido una RNN con arquitectura RG(80), con los hiperparámetros de (62).

$$\begin{gathered} batch\ size = 1 \\ learning\ rate\ inicial = 10^{-4} \\ epochs = 30 \end{gathered} \tag{62}$$

Con esta red se obtiene un RMSE de 0.01 en los datos de entrenamiento, y un 0.03 en los de test, que son datos correspondientes al sujeto 16, que no es utilizado en entrenamiento, mientras que los sujetos 2 y 10 se utilizan como validación, quedando el resto para entrenamiento.

En la Figura 55 se muestra la comparativa de los datos que predice la red en test y los datos reales correspondientes. Se realiza un *scatterplot* y en el caso de que los datos predichos coincidieran exactamente con los reales, se podría apreciar una diagonal perfecta. En este caso, esta diagonal aparece ligeramente ensanchada, ya que no se obtiene un ajuste perfecto, pero sí se puede considerar satisfactorio.

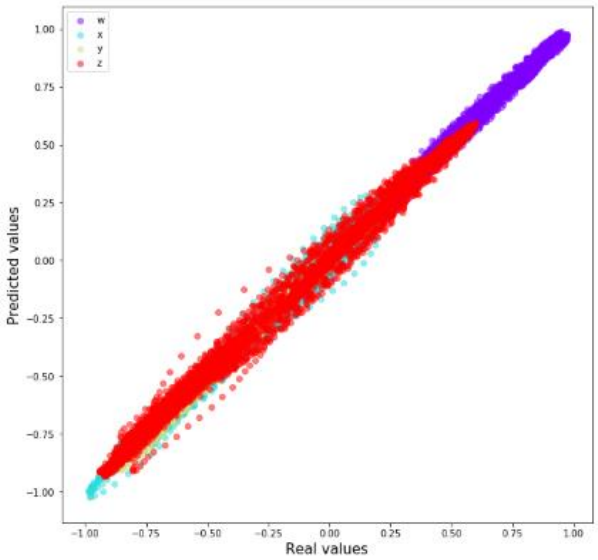


Figura 55. *Comparativa de valores reales y predichos.*

Además, en las Figuras 56-60 se incluyen los datos que se utilizan de test (discontinuo) y las predicciones de la red (continuo) para los 5 sensores utilizados.

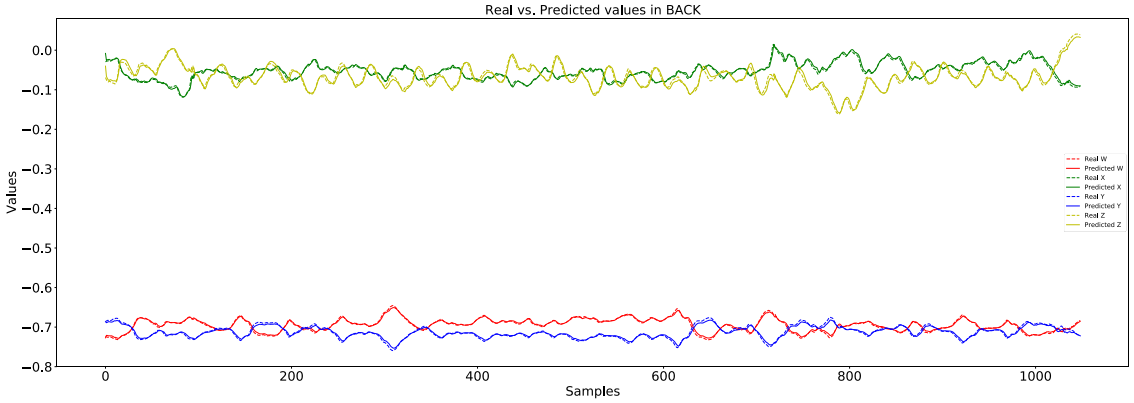


Figura 56. *Datos reales vs. predichos para el sensor de la espalda.*

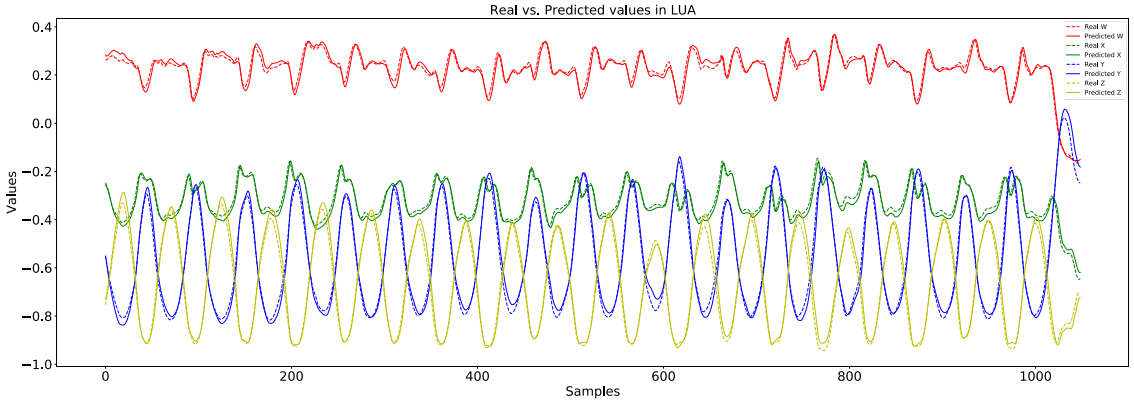


Figura 57. *Datos reales vs. predichos para el sensor de la parte superior del brazo izquierdo.*

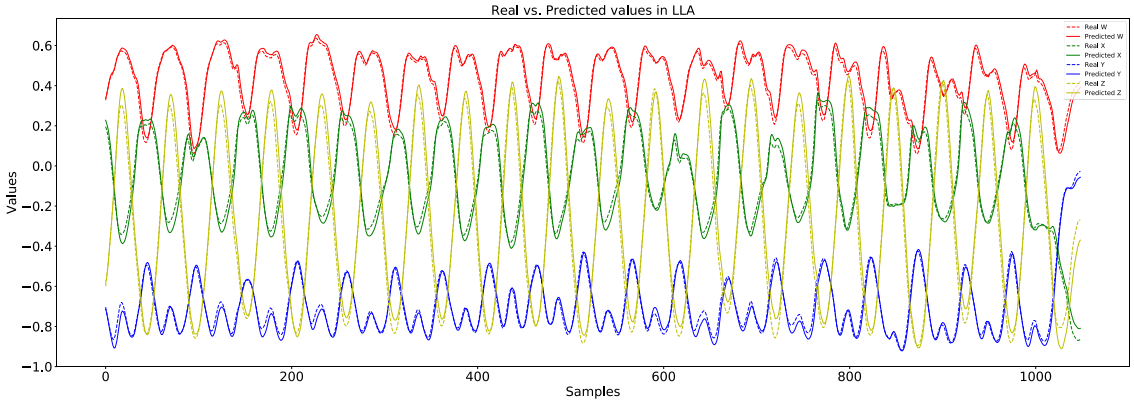


Figura 58. *Datos reales vs. predichos para el sensor de la parte inferior del brazo izquierdo.*

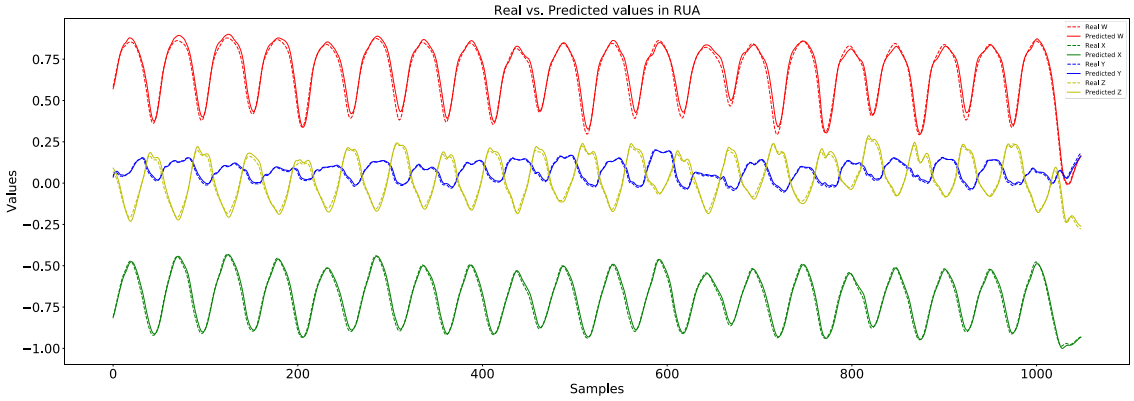


Figura 60. *Datos reales vs. predichos para el sensor de la parte superior del brazo derecho.*

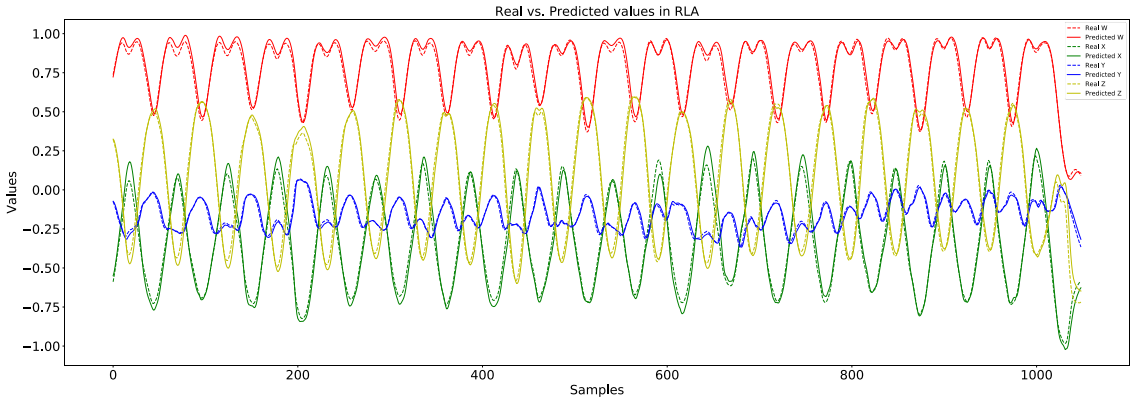


Figura 59. *Datos reales vs. predichos para el sensor de la parte inferior del brazo derecho.*

Además, como comentario adicional, cabe destacar que se ha realizado la misma prueba de esta red, pero en vez de utilizando un RNN con celdas GRU, se ha realizado con celdas LSTM, obteniendo los mismos resultados.

Por último, también se deja acceso mediante un enlace a un vídeo en el que se muestra una comparativa de los datos reales de test y lo que predice la red.

4.4 Discusión de resultados

Este apartado se centra en la comparativa de los resultados obtenidos en este Trabajo Fin de Máster, en la parte de Clasificación, con otros trabajos que utilizan el dataset REALDISP para realizar un reconocimiento de movimientos.

En este sentido, cabe destacar que se mejoran los resultados de los otros trabajos. En primer lugar, para el método *10-Fold ranodm-partitioning*, se mejora el resultado de 97% obtenido por (Baños, O. *et al*. 2014b, pp. 9995-10023), ya que en este Trabajo Fin de Máster se consigue cerca de un 99.5% de exactitud.

Por otro lado, y siendo un resultado más interesante, en *12-Fold subject-wise* se supera lo conseguido por (Zhu, J. *et al*., 2017), ya que, a pesar de que no se supera su mejor resultado (99.4%), si se tiene en cuenta su resultado utilizando únicamente los datos provenientes de los cuaterniones, solamente consiguen un 93%, ampliamente superado por lo obtenido en este Trabajo Fin de Máster, de un 97.7% de exactitud. Además, este resultado es mejorado utilizando 30 veces menos *features* y 110 veces menos instancias, y si se tiene en cuenta su mejor resultado, este TFM se queda solamente a un 1.7% de alcanzar su marca, utilizando, en ese caso, 100 veces menos *features* que en (Zhu, J. *et al*., 2017).

Hace falta recordar también que, si bien en los dos trabajos comentados se realiza una clasificación de 33 actividades y en este de 11, en este TFM se utilizan únicamente 12 sujetos frente a los 17 que utilizan en dichos trabajos, además de la gran reducción de datos que se ha tenido, como se ha explicado en secciones anteriores.

En la Tabla 4 se muestra un resumen de la comparativa aquí realizada. El primer trabajo comentado se denota por *Oresti et al.* y el segundo por *Zhu et al.*, en referencia al apellido de sus autores principales, para una mejor legibilidad de la tabla.

Network	#IMUs	Sensors	#features (per IMU)	#features (total)	#activities	#subjects	Evaluation Method	Number of instances	Accuracy
Oresti et al.	9	ACC+GYR+MAG+QUAT	-	-	33	17	1	-	97.0%
Zhu et al.	9	ACC+GYR+MAG+QUAT	106	4086	33	17	2	130000	99.4%
		QUAT		1224					93.0%
TFM	5	QUAT	8	40	11	12	1	1176	99.5%
							2		97.7%

Tabla 4. *Comparativa de este trabajo con otros publicados que utilizan el mismo dataset.*

4.5 Presupuesto

Como última sección de este capítulo, se muestra la inversión necesaria para la realización de un estudio de estas características. Esta inversión se puede dividir en gastos de equipamiento y gastos en personal.

A continuación, se desglosa y detalla la inversión necesaria. El gasto en personal (Tabla 5) se calcula a partir del tiempo empleado en el desarrollo del estudio, sin incluir la etapa de aprendizaje de la teoría y del software de desarrollo.

Horas	Precio por hora (€)	Total (€)
1200	15	18000

Tabla 5. *Presupuesto de ingeniero.*

Equipamiento	€/Unidad	Nº de unidades	Subtotal (€)
Portátil amortizado	780	1	**780**
GASTOS PERSONAL			**18000**
GASTOS SERVIDOR (amortizado)			**1514.51**
Procesador CPU Intel® Core™ i5-4690	240	1	240
GPU Nvidia® GeForce GTX 1060	224,51	1	224,51
Mantenimiento, reparaciones y otros gastos	-	-	1000
IMPORTE TOTAL			**19514.51**

Tabla 6. *Presupuesto total.*

En la Tabla 6, se han utilizado precios de venta iniciales, por lo que actualmente podrían tener valores ligeramente inferiores. Además, se ha añadido un sobrecoste de mantenimiento, reparaciones y otros gastos que contiene la estimación de los costes de los dispositivos del servidor de los que no se ha podido obtener el precio junto con los costes que supone su mantenimiento y el reemplazo de algunos componentes a lo largo de los años. De esta forma, el gasto en el servidor se estima alrededor de unos 1514.51 €.

Capítulo 5. Conclusiones y líneas futuras

La gran cantidad de avances tecnológicos que se están dando hoy en día están impulsando en gran medida tanto la Inteligencia Artificial como el Deep Learning. Estos son los campos que se están desarrollando con más notoriedad en los últimos años dentro de la Ingeniería, y están generando una serie de grandes avances que no hacen más que fomentar la investigación en estos campos. Por ello, es de gran interés la resolución del problema HAR mediante redes neuronales de aprendizaje profundo, ya que, como se ha visto en este Trabajo Fin de Máster, tienen un gran potencial, no solamente para clasificar, sino para generar datos dentro del dominio de las señales sensoriales.

5.1 Conclusiones

En el primer capítulo de esta memoria se marcaban dos objetivos principales que se debían alcanzar al finalizar este Trabajo Fin de Máster. El primero de ellos era explorar mediante redes neuronales de aprendizaje profundo el problema HAR, y que se adaptara a un sistema embebido de tiempo real.

Este objetivo se ha cumplido al conseguir unos resultados satisfactorios en la clasificación y, además, unas características de la red y los datos (tamaño de los datos y tiempo de inferencia de la red) que permiten plantearse utilizar la red con mejor resultado dentro del sistema requerido.

El segundo objetivo era la generación de datos en el dominio de las señales provenientes de sensores. Objetivo también cumplido al obtener resultados altamente satisfactorios, que permiten plantearse utilizar estos datos para futuras aplicaciones con redes neuronales.

El estudio desarrollado en este Trabajo Fin de Máster ha demostrado el potencial del Deep Learning para la resolución del problema HAR y su posibilidad de aplicación dentro de sistemas embebidos que requieran la característica de ser suficientemente rápidos como para considerarlos de tiempo real. Una aplicación de esto sería como complemento del desarrollo de un juego serio para la rehabilitación de pacientes, ayudando a determinar si los ejercicios marcados para la recuperación física están siendo correctamente realizados por el paciente o no.

La **primera conclusión** a la que se ha llegado, tras realizar un estudio del problema HAR, es que este problema está ya suficientemente estudiado mediante otras técnicas y algoritmos de Machine Learning, con tasas realmente elevadas, pero con un preprocesado de los datos muy profundo, lo que lleva a una programación subyacente muy tediosa. La irrupción del Deep Learning dentro de este problema está generando una serie de avances que no hacen más que reforzar la idea de seguir en esta línea, ya que las redes neuronales permiten obtener buenos resultados de manera más sencilla que con otros algoritmos, sin necesidad de un preprocesado de los datos tan exhaustivo.

La **segunda conclusión** es que hay infinidad de tipos de red neuronal adecuadas a la resolución del problema. Esta dependerá de la naturaleza de los propios datos que se

utilicen y el preprocesamiento que se realice sobre ellos. En este Trabajo Fin de Máster se han visto dos tipos de redes clasificadoras, con distintas configuraciones que permiten la resolución de un mismo problema con buenos resultados. Además, se han podido ver varias formas de generar datos, utilizando también redes de muy distinta arquitectura, dando resultados positivos.

La **conclusión final** a la que se ha llegado es que se ha realizado un amplio estudio de cómo resolver el problema HAR, que ha permitido ver un alto potencial de las redes neuronales tanto como clasificadoras como generadoras. En este caso, como red clasificadora ha destacado una que combina capas convolucionales y recurrentes, lo que ha permitido tener en cuenta tanto las relaciones espaciales como temporales, obteniendo **resultados competitivos** con respecto a **otros trabajos**, llegando a **superarlos** en condiciones semejantes.

5.2 Líneas futuras

Tras el cumplimiento de los objetivos de este Trabajo Fin de Máster, queda la posibilidad de afrontar varias líneas de futuro, de cara a mejorar los resultados obtenidos

La primera de las **posibilidades** en lo referente a la clasificación sería la utilización de los datos generados de manera sintética para utilizarlos dentro del entrenamiento de las redes neuronales, en la búsqueda de resultados aún mejores.

Para ello, en generación se abriría una **línea futura** que consistiría en el estudio más profundo de las redes para la generación de los datos, llegando a controlar características de los datos que se generan.

Otra **posibilidad** sería plantearse utilizar todos los datos disponibles en el dataset, es decir, utilizar en el entrenamiento los datos procedentes del acelerómetro, giroscopio y del magnetómetro junto con los cuaterniones ya utilizados, buscando un refuerzo del entrenamiento.

Finalmente, quedaría abierta la línea de, a partir de este estudio, plantearse la realización de un **dataset propio**, con las actividades específicas que se deseen, para después desarrollar completamente el sistema, incluyendo las redes aquí estudiadas implementándolo dentro de un **juego serio** diseñado para la rehabilitación.

REFERENCIAS

Baños, O., Damas, M., Pomares, H., Rojas, I., Tóth, M. A. y Amft, O. (2012) *A benchmark dataset to evaluate sensor displacement in activity recognition*.

Baños, O. y Tóth, M. A. (2014a) *Realistic sensor displacement benchmark dataset*. Manual del dataset.

Baños, O., Toth, M. A., Damas, M.m Pomares, H. y Rojas, I. (2014b) "Dealing with the Effects of Sensor Displacement in Wearable Activity Recognition", *Sensors*, 14, pp. 9995-10023.

Bengio, Y. (2009) *Learning Deep architectures for AI*. Found Trends Mach Learn

Buck, I. (2010) "The Evolution of GPUs for General Purpose Computing", *GPU Technology Conference*. San Jose, CA, 20-23 septiembre 2010.

Bulling, A., Blanke, U., Schiele, B. (2014) *A tutorial on human activity recognition using body-worn inertial sensors*.

ACM Comput Surv Center for Machine Learning and intelligent Systems (n.d.) *UCI Machine Learning Repository*. Disponible en: https://archive.ics.uci.edu/ml/datasets/REALDISP+Activity+Recognition+Dataset [Consultado 30-5-2019].

Chauvin, Y. y Rumelhart, D. E. (1995) *Backpropagation: Theory, architectures, and applications*. New Jersey: Lawrence Erlbaum Associates, Publishers.

Cho, K., Gulcehre, B. C., Bahdanu, D., Bougares, F., Schwenk, H. y Bengio, Y. (2014) *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*.

Cortes, C. y Vapnik, V. (1995) *Support-vector networks. Machine Learning*.

Docker (2019) *What is Docker?* Disponible en: https://www.docker.com/what-docker [Consultado 10-12-2019].

Duchi, J., Hazan, E. y Singer, Y. (2011) "Adaptive subgradient for online learning and stochastic optimization", *The Journal of Machine Learning Research*, 12, pp. 2121-2159.

Fukushima, K. y Miyake, S. (1982) *Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition*. Competition and Cooperation in neural nets: Springer.

Gaudet, C. J. y Maida A. S. (2018) "Deep Quaternion Networks" *2018 International Joint Conference on Neural Networks (ICJNN)*. Río, Brasil, 8-13 de julio de 2018.

González Alonso, J. (2017) *Periféricos basados en Arduino para interacción con sistemas médicos de simulación y rehabilitación*. Trabajo Fin de Grado. Universidad de Valladolid.

Goodfellow, I., Bengio, Y. y Courville A. (2016) *Deep Learning*. Cambridge, MA (EEUU): MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. y Bengio, Y. (2014) *Generatuve Adversarial Nets*.

Hassan, M. M., Huda, S., Uddin, M., Almogren, A., Alrubaian, M. (2018) "Human Activity Recognition from Body Sensor Data using Deep Lerning", *Journal of Medical Systems*.

Hochreiter, S. y Schmidhuber, J. (1997) "Long short-term memory". En: *Neural Computation*. MIT Press.

Intel Corporation (n.d.) Intel. Disponible en: https://ark.intel.com/content/www/es/es/ark/products/80810/intel-core-i5-4690-processor-6m-cache-up-to-3-90-ghz.html [Consultado 19-1-2020].

Ivakhnenko, A. G. e Ivakhnenko, G. A. (1995) "The Review of Problems Solvable by Algorithms of the Group Method of Data Handling (GMDH)". En: *Patern Recognition and Image Analysis*.

Jupyter (2019) Jupyter. Diponible en: http://jupyter.org/ [Consultado 10-12-2019].

Keras (2019) Disponible en: https://keras.io/ [Consultado 6-12-2019].

Keerthipala, W. W. L., Chong, L. T. y Leong., T. C. (1995) "Artificial Neural Network for Analysis of Power System Harmonics", *International Conference of Neural Networks IEEE*.

Kingma D. P. y Lei Ba, M. (2015) "Adam: A method for stochastic optimization". ICLR. San Diego, 7-9 mayo 2015.

Krizhevsky, I., Sutskever, I. y Hinton. G. E. (2012) *ImageNet Classification with Deep Convolutional Neural Networks*.

Kulharia, V., Ghosh, A., Mukerjee, A., Namboodiri, V. y Bansal, M. (2017) "Contextual RNN-GANs for Abstract Reasoning Diagram Generation", *Thirsty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. San Francisco, EEUU, 4-9 de febrero de 2017.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. y Jackerl, L. D. (1989) "Backpropagation Applied to Handwritten Zip Code Recognition". En: *Neural Computation*, pp. 541-551.

LeCun, Y., Bengio, Y. y Hinton, G. (2015) *Deep Learning*. Nature Publishing Group.

Matich, D. J. (2001) *Redes Neuronales. Conceptos básicos y aplicaciones*. Cátedra. Universidad Tecnológica Nacional – Facultad Regional Rosario.

McCulloch, W., S. y Pitts, W. (1943). "A logical calculus of the ideas immanent in nervous activity". *The bulletin of mathematical biopshisycs*.

NVIDIA Corporation (2017) Nvidia GeForce GTX 1060 *datasheet*. Disponible en: https://www.nvidia.com/es-es/geforce/products/10series/geforce-gtx-1060/ [Consultado 19-1-2020].

NVIDIA Corporation (2018) *CUDA Toolkit Documentation*. Disponible en: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#memory-optimizations [Consultado 4-1-2020]

Ordóñez, F. J. y Roggen, D. (2016) *Deep Convolutional ans LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition*.

Parcollet, T., Ravanelli, M., Morchid, M., Linàres, G., Trabelsi, C., De Mori, R. y Bengio, Y. (2019) "Quaternion Recurrent Neural Networks", *Seventh International Conference on Learning Representations*. Nueva Orleans, EEUU, 6-9 de mayo de 2019.

Pavllo, D., Grangier, D. y Auli, M. (2018) "Quaternet: A Quaternion-based Recurrent Model for Human Motion", *British Machine Vision Conference (BMVC)*. Newcastle, UK, 3-6 de septiembre de 2018.

Python (2019) Python. Disponible en: https://www.python.org/ [Consultado 6-12-2019].

Rosenblatt, F. (1958), *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386-408

Sáez Bombín, S. (2018) *Reconocimiento de actividades físicas con sensores inerciales y Redes Neuronales de Aprendizaje* Profundo. Trabajo Fin de Grado. Universidad de Valladolid.

Sak, H., Senior, A. y Beaufays, F. (2014) *Long short-term memory based Recurrent Neural Network Architectures for large vocabulary speech recognition*.

San José, R. (2019) "Métodos de Aprendizaje Profundo en imagen médica: principios básicos y aplicaciones".

San, P. P., Kakar, P., Li, X., Krishnaswamy, S., Yang, J., Nguyen, M. N. (2017) "Deep Learning for Human Activity Recognition". En: Hsu, Hui-Huang *et al.* Eds. *Big Data Analytics for Sensor-Network Collected Intelligence*. Singapur: *Elsevier Inc.*

Soleimani, E. y Nazerfard, E. (2019) *Cross-Subject Transfer Learning in G¡Human Activity Recognition Systems using Generative Adversarial Networks*.

Sutton, R. S. y Barto, A. G. (2014) *Reinforcement Learning: An Introduction*. Primera edición. MIT Press.

Tensorflow (2019) Tensorflow. Disponible en: https://www.tensorflow.org/ [Consultado 6-12-2019].

Udacity (2018) *Courses*. Disponible en: https://classroom.udacity.com [Consultado 10-12-2019].

Unity (2019) Disponible en: https:/unity.com/ [Consultado 6-12-2019].

Valenti, R. G., Dryanowski, I. y Xiao, J. (2015) "Keeping a Good Attitude: A Quaternions-Based Orientation Filter for IMUs and MARGs", *Sensors*, 15, pp. 19308-19314.

Velick, R. (n.d.) *Discrete Fourier Transform Computation Uisng Neural Networks*. Universidad de Tecnología de Viena.

Vondrick, C., Pirsiavash, H. y Torralba, A. (2016) "Generating Videos with Scene Dynamics", *Thitieth Conference on Neural Information Processing Systems (NIPS)*. Barcelona, España, 5-10 de diciembre de 2016.

Zaremba, W., Sutskever, I. y Vinyals, O. (2015) *Recurrent Neural Network Regularization*.

Zhu, J, San-Segundo, D. y Pardo, J. M. (2017) "Feature extraction for robust physical activity recognition", *Human-centric Computing and Information Sciences*, 7 (16). Disponible en: https://link.springer.com/content/pdf/10.1186%2Fs13673-017-0097-2.pdf [Consultado 8-9-2019].

ANEXOS

ANEXO I
Matrices de confusión

En este ANEXO I se presentan las matrices de confusión tanto de la mejor red CNN como de la mejor red CNN+RNN, para el caso de test con los datos orientados como en el dataset original.

En primer lugar, se presentan las 12 matrices de confusión de la red CNN del método *12-Fold cross-validation* y posteriormente, las 12 matrices de confusión de la red CNN+RNN también del método *12-Fold cross-validation*. Por lo tanto, se presentan todas las matrices de confusión del caso más realista, es decir, del método *12-Fold cross-validation* y del test realizado con los datos originales del dataset.

Como se ha indicado en el apartado de resultados, la matriz de confusión es una de las formas de ver el rendimiento de una red (o de un algoritmo de Machine Learning). En ella se muestra en cada fila el número de ejemplos de la clase verdadera y en las columnas el número de veces que la red o algoritmo ha predicho una clase, es decir, un elemento de la matriz sería el número de veces que la red predice una actividad *X*, siendo verdadera la actividad *Y* (con *X* la fila e *Y* la columna). Sabiendo esto, cuanto más diagonal sea una matriz de confusión, mejor rendimiento está teniendo en la clasificación de actividades.

Finalmente, las redes de las que se presentan sus matrices de confusión son:

- **CNN**: *C(16)-C(32)-C(64)-C(128)-DR(0.5)-* $S_m \rightarrow$ 95.66% de *accuracy*.
- **CNN+RNN**: *C(64)-C(64)-C(64)-DR(0.5)-RG(64)-* $S_m \rightarrow$ 97.7% de *accuracy*.

I.1. Matrices de confusión de CNN

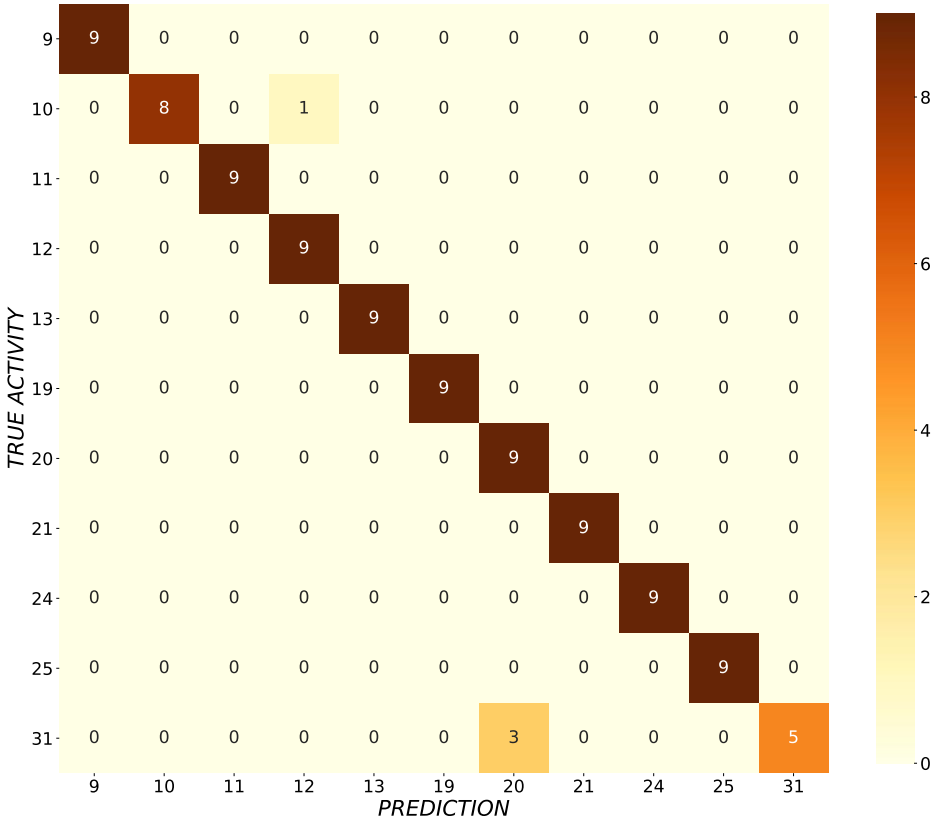


Figura. *Matriz de confusión para el sujeto 1.*

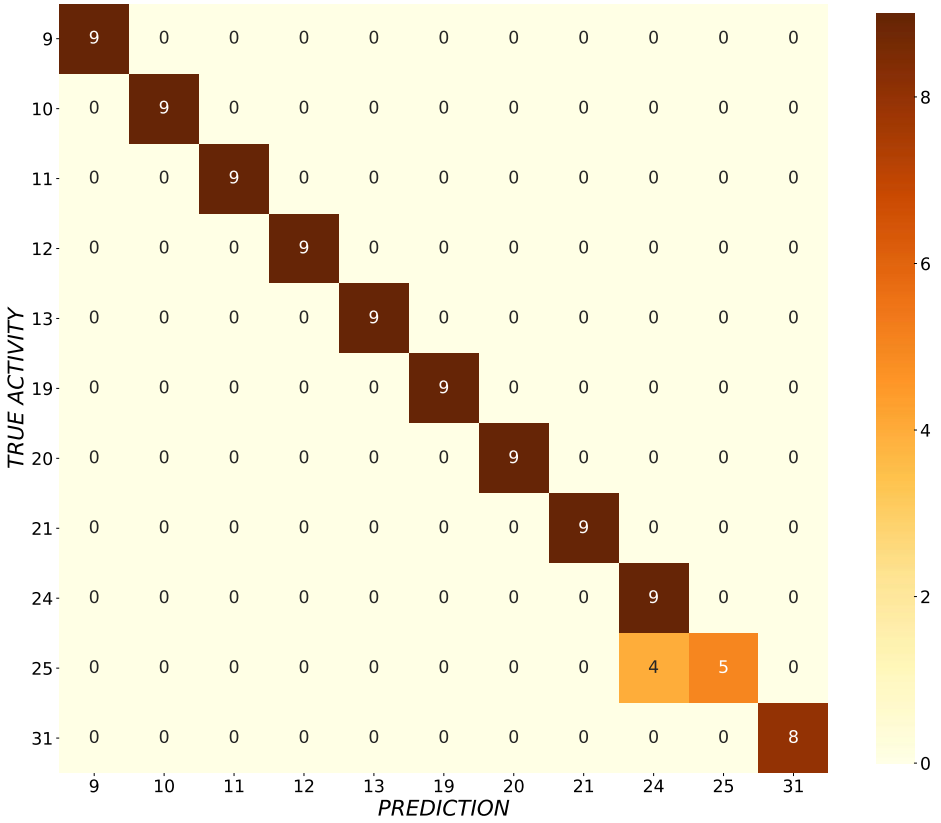


Figura. *Matriz de confusión para el sujeto 2.*

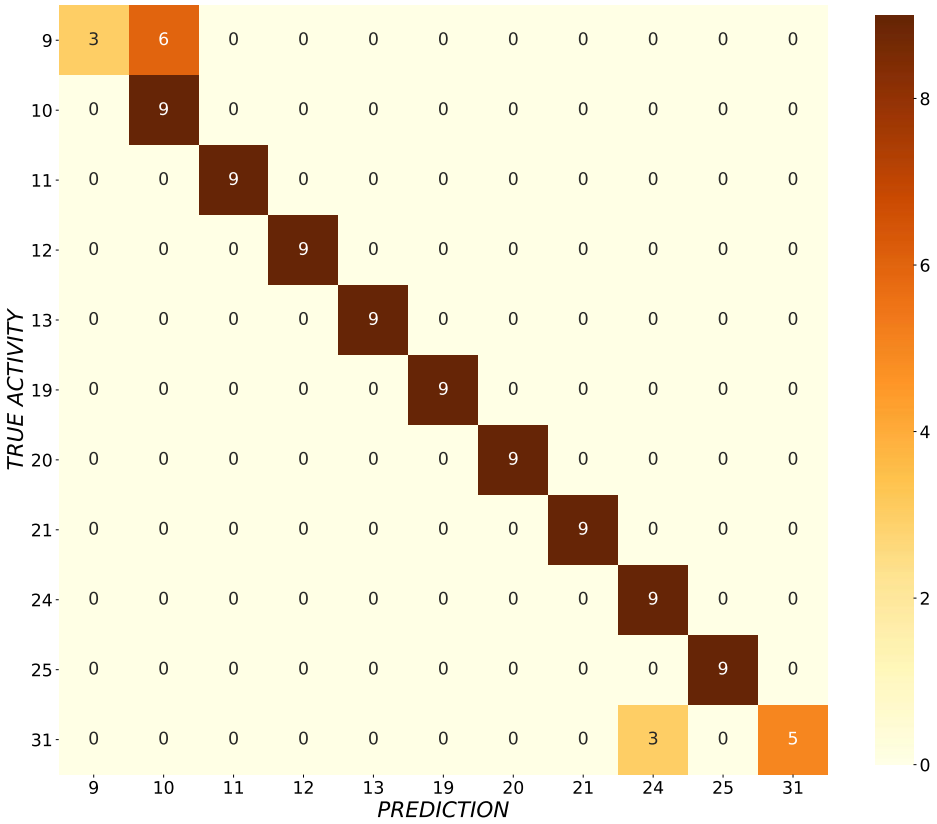


Figura. *Matriz de confusión para el sujeto 3.*

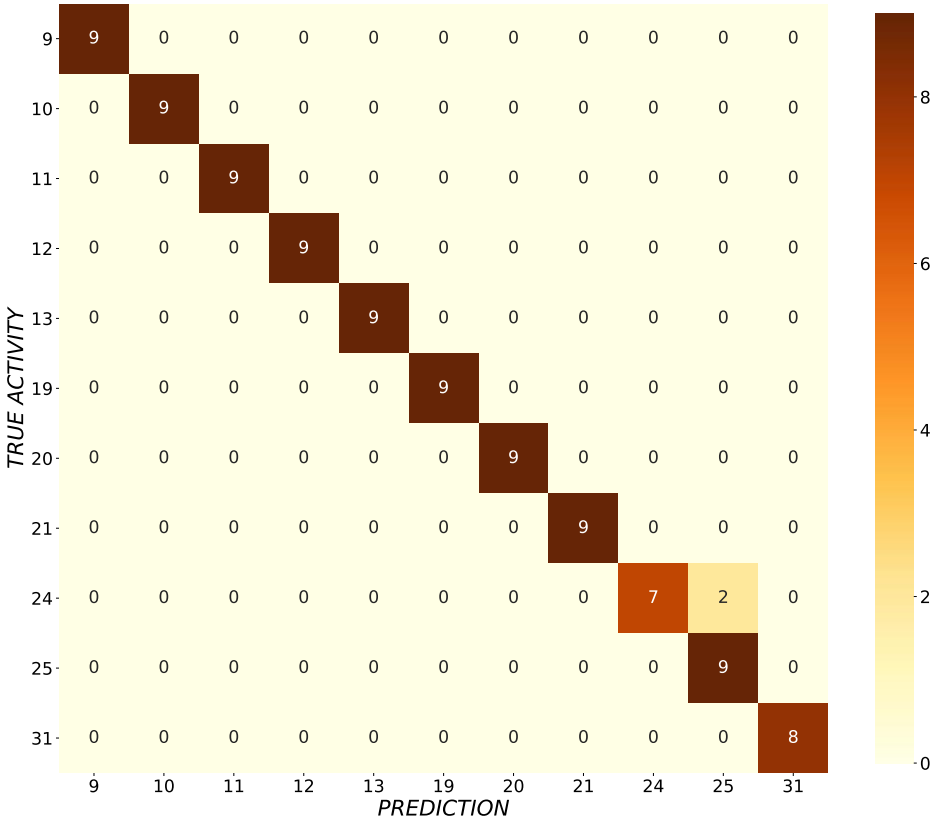


Figura. *Matriz de confusión para el sujeto 5.*

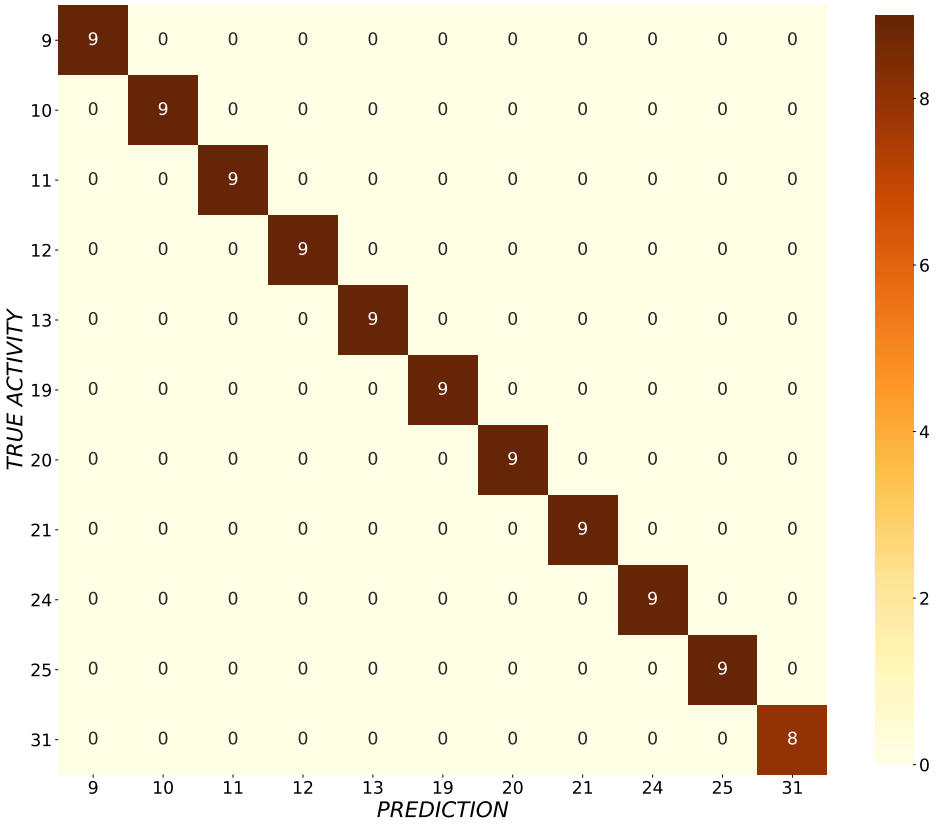


Figura. *Matriz de confusión para el sujeto 8.*

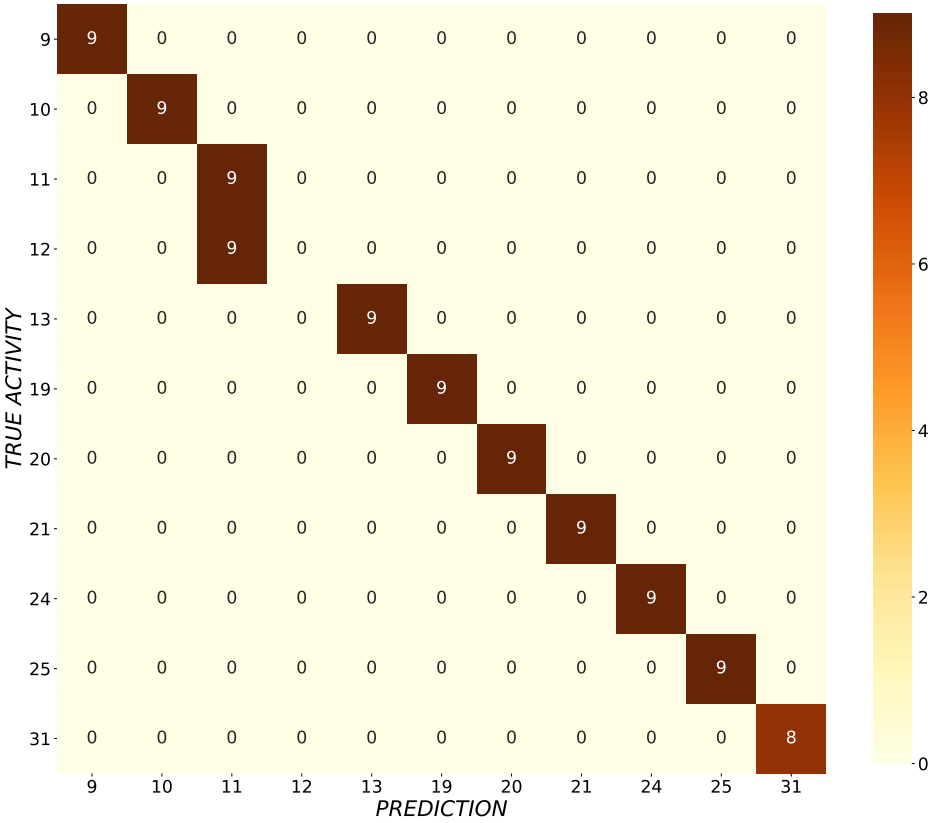


Figura. *Matriz de confusión para el sujeto 9.*

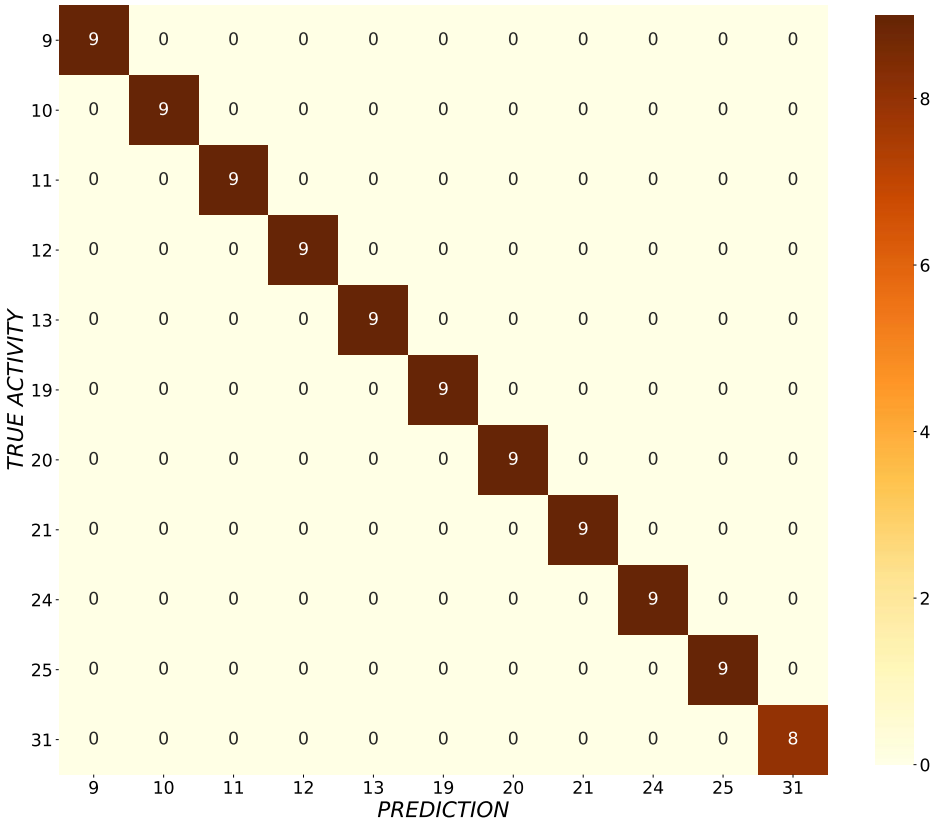


Figura. *Matriz de confusión para el sujeto 10.*

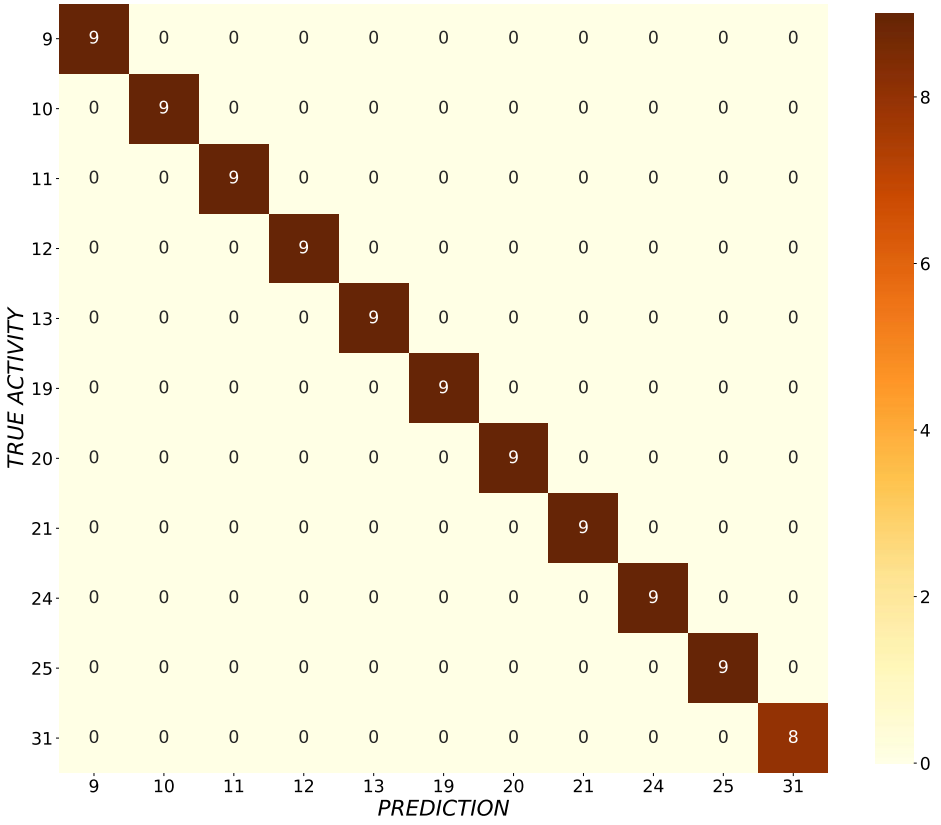


Figura. *Matriz de confusión para el sujeto 11.*

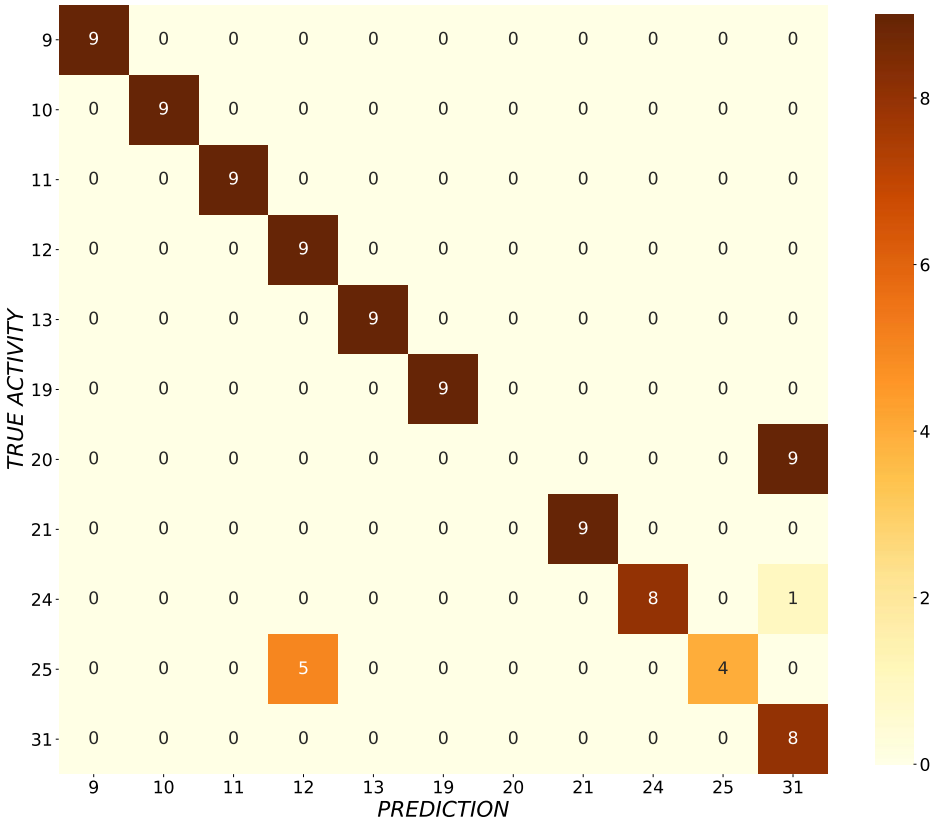


Figura. *Matriz de confusión para el sujeto 13.*

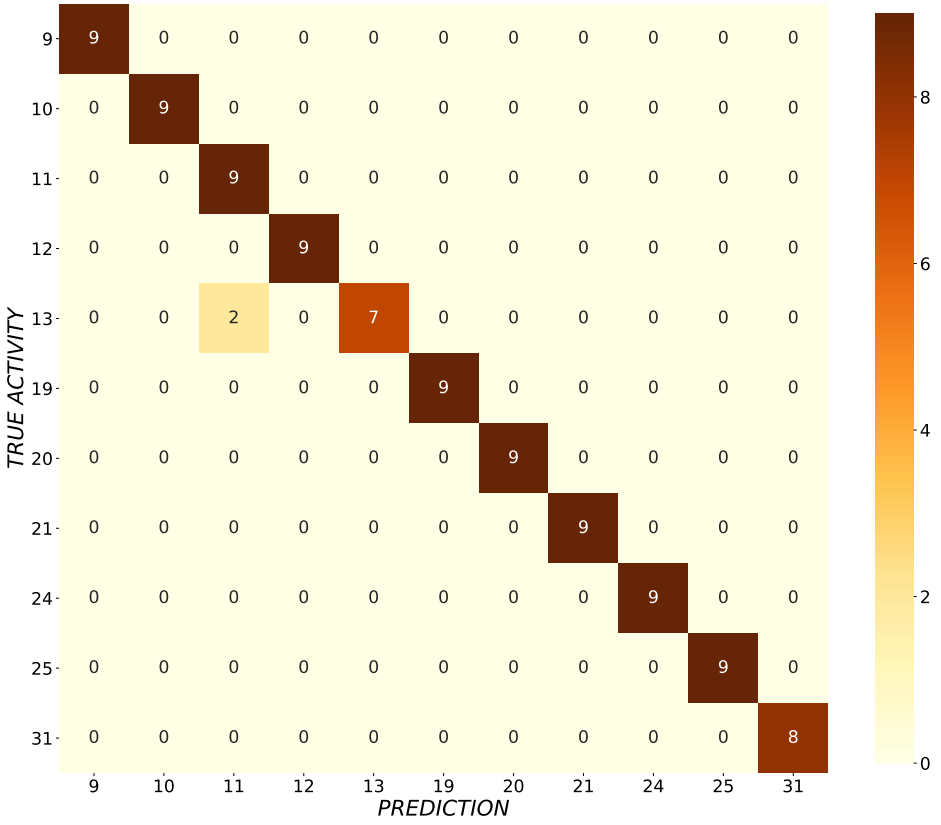


Figura. *Matriz de confusión para el sujeto 14.*

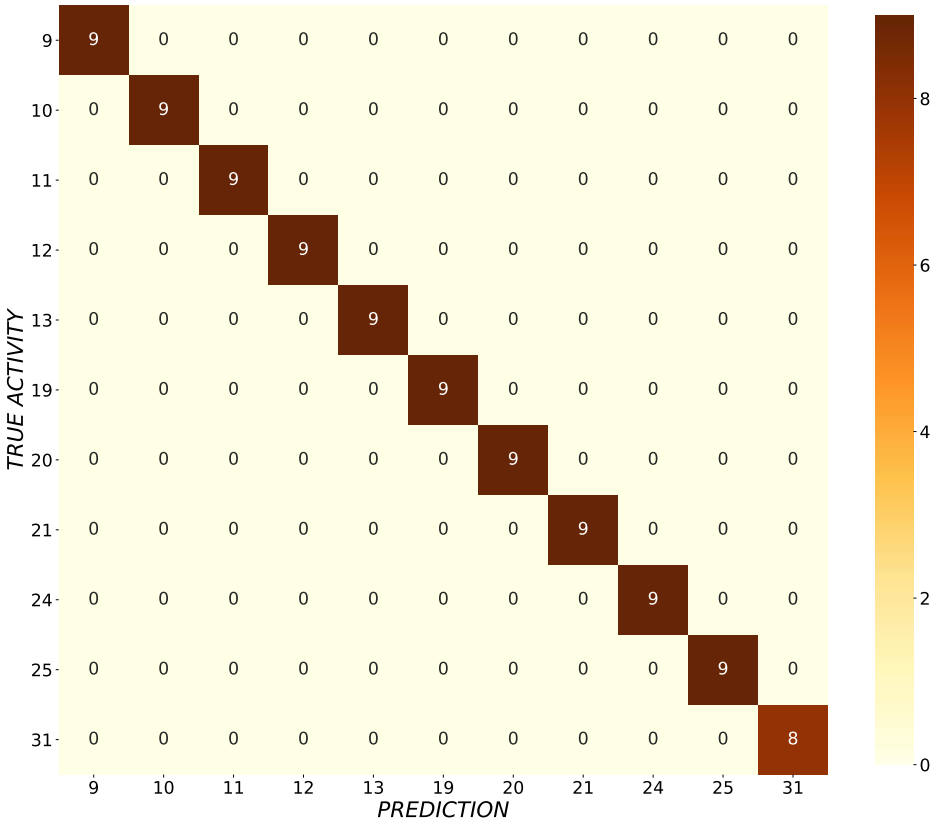


Figura. *Matriz de confusión para el sujeto 16.*

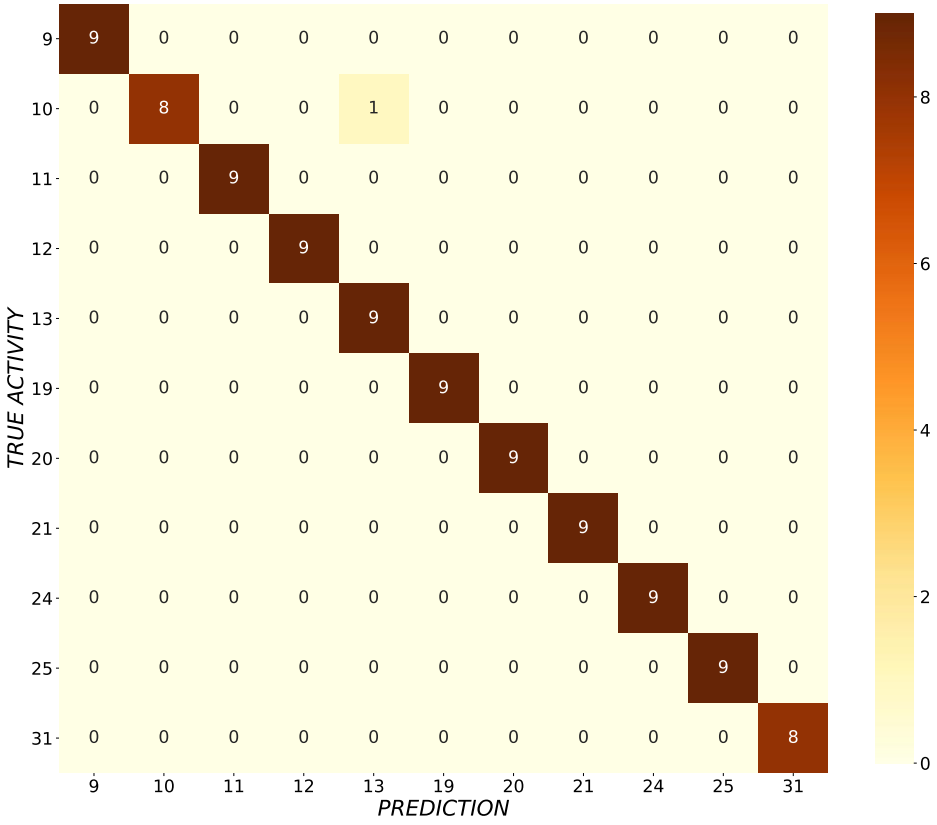


Figura. *Matriz de confusión para el sujeto 17.*

I.2. Matrices de confusión de CNN+RNN

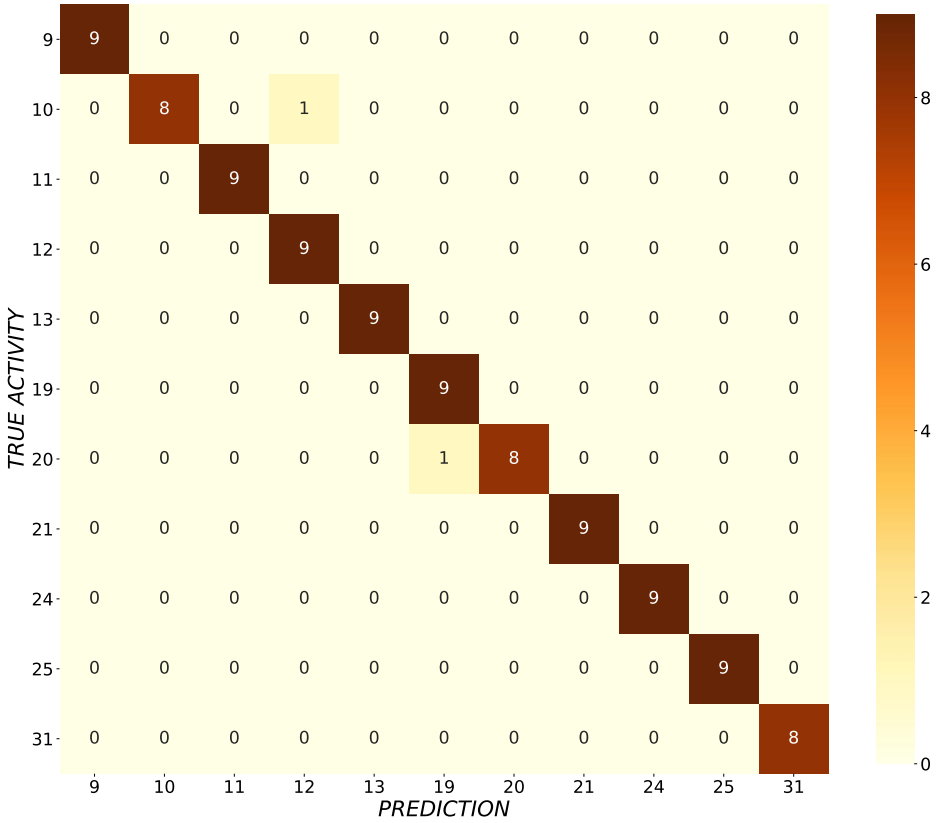


Figura. *Matriz de confusión para el sujeto 1.*

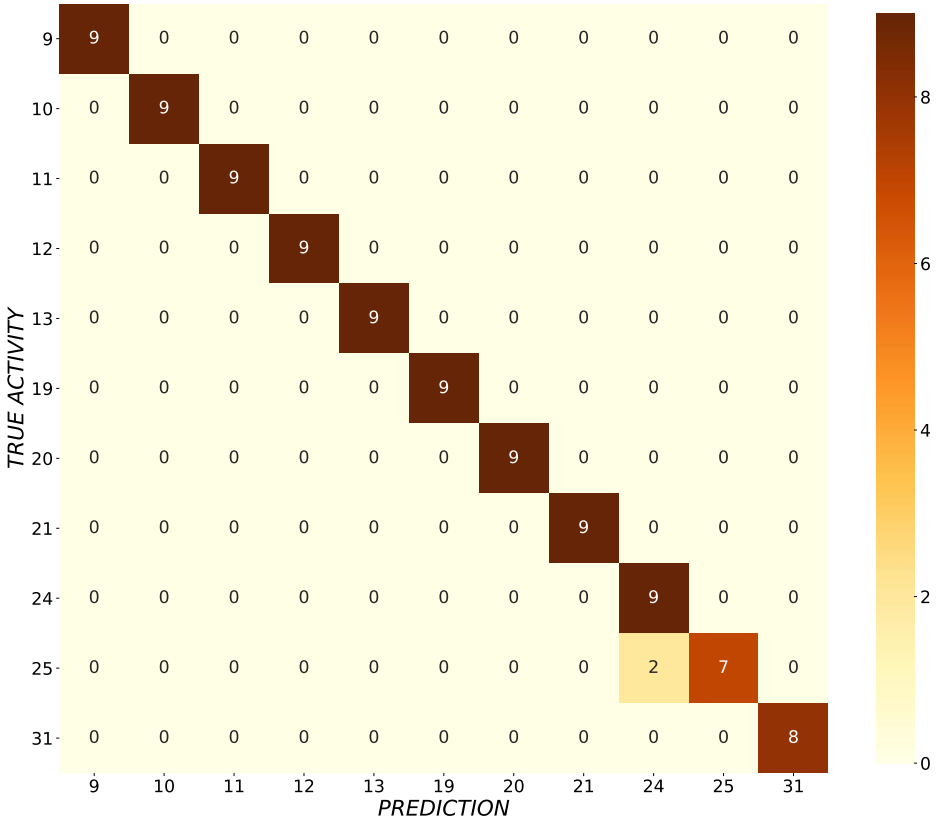


Figura. *Matriz de confusión para el sujeto 2.*

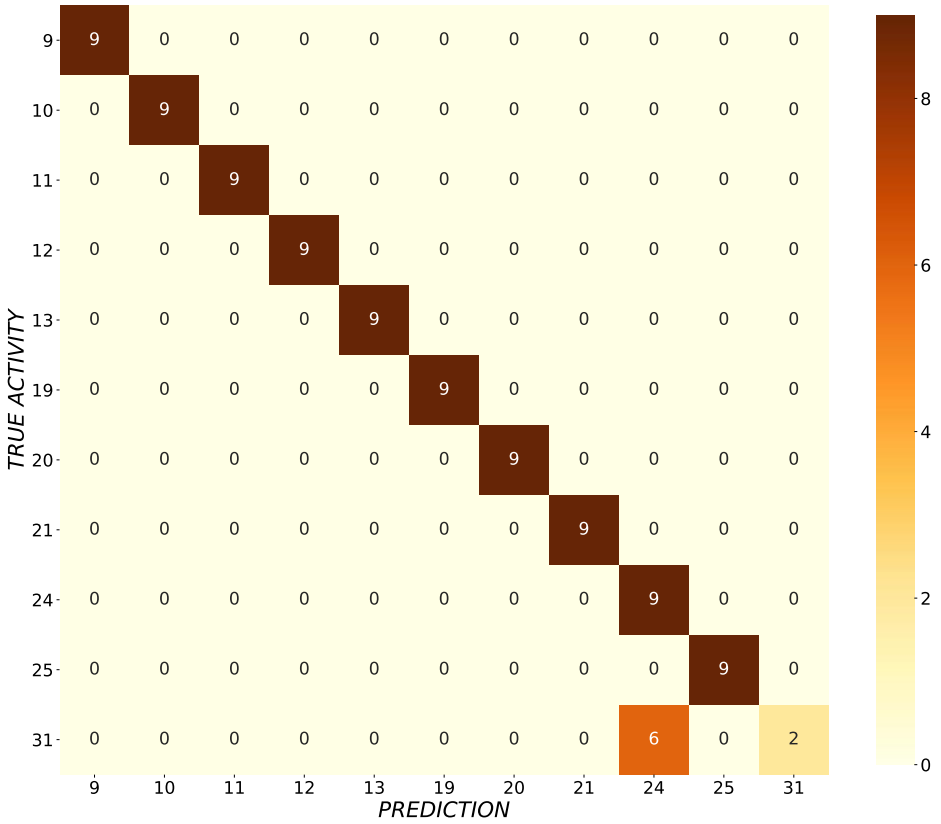


Figura. *Matriz de confusión para el sujeto 3.*

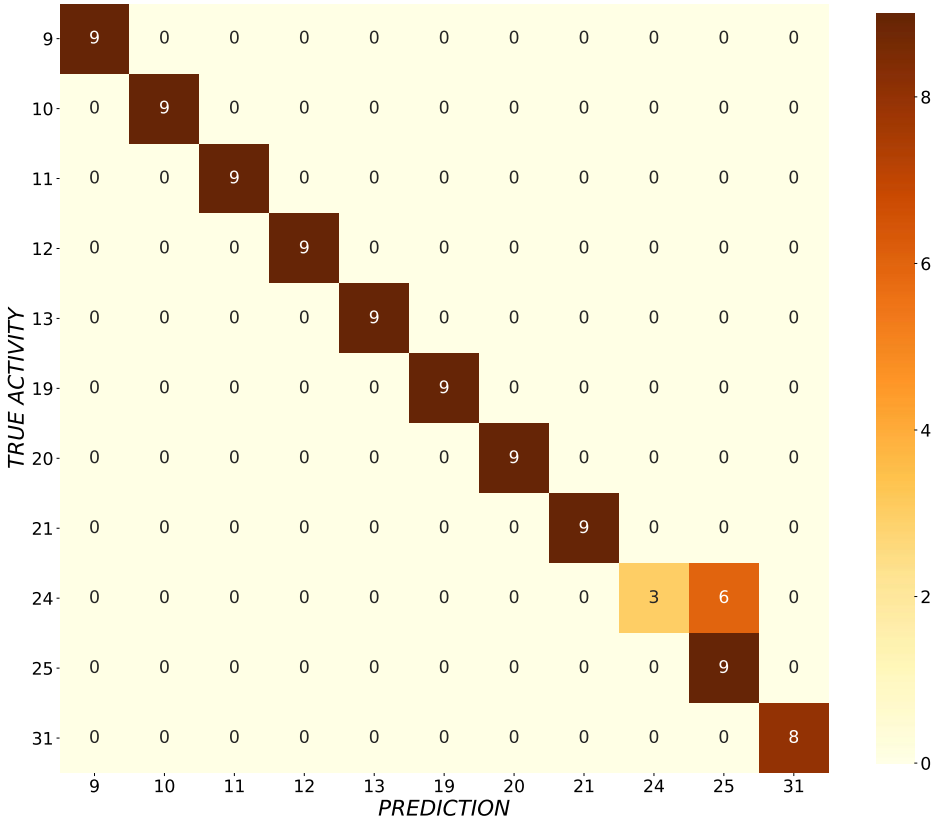


Figura. *Matriz de confusión para el sujeto 5.*

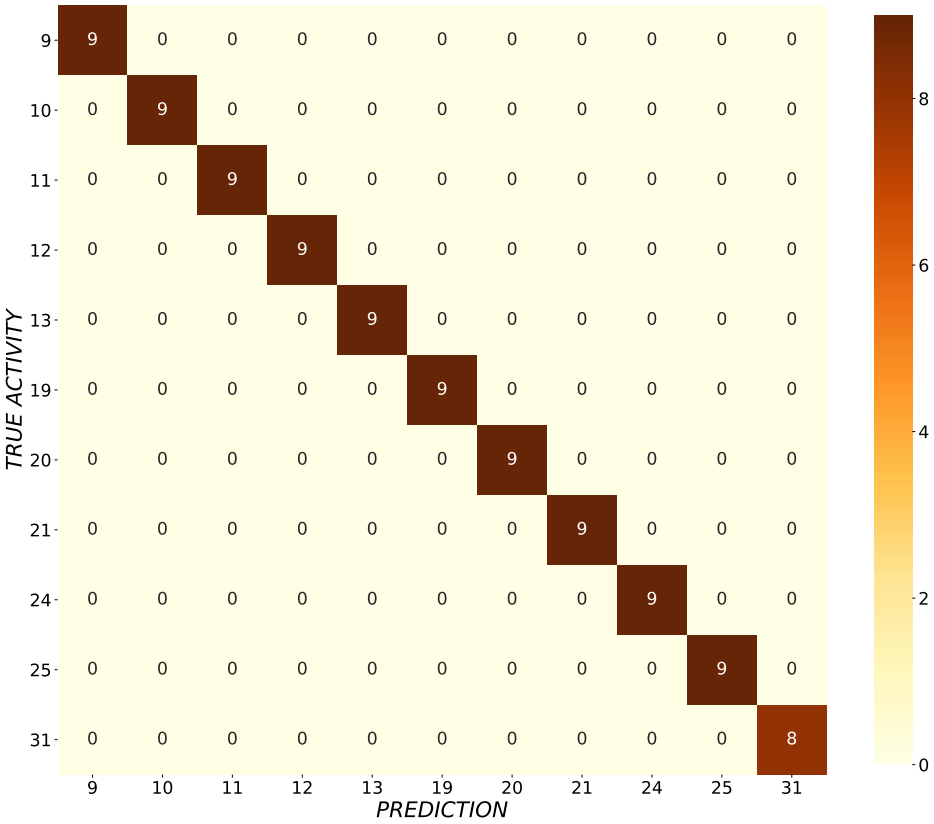


Figura. *Matriz de confusión para el sujeto 8.*

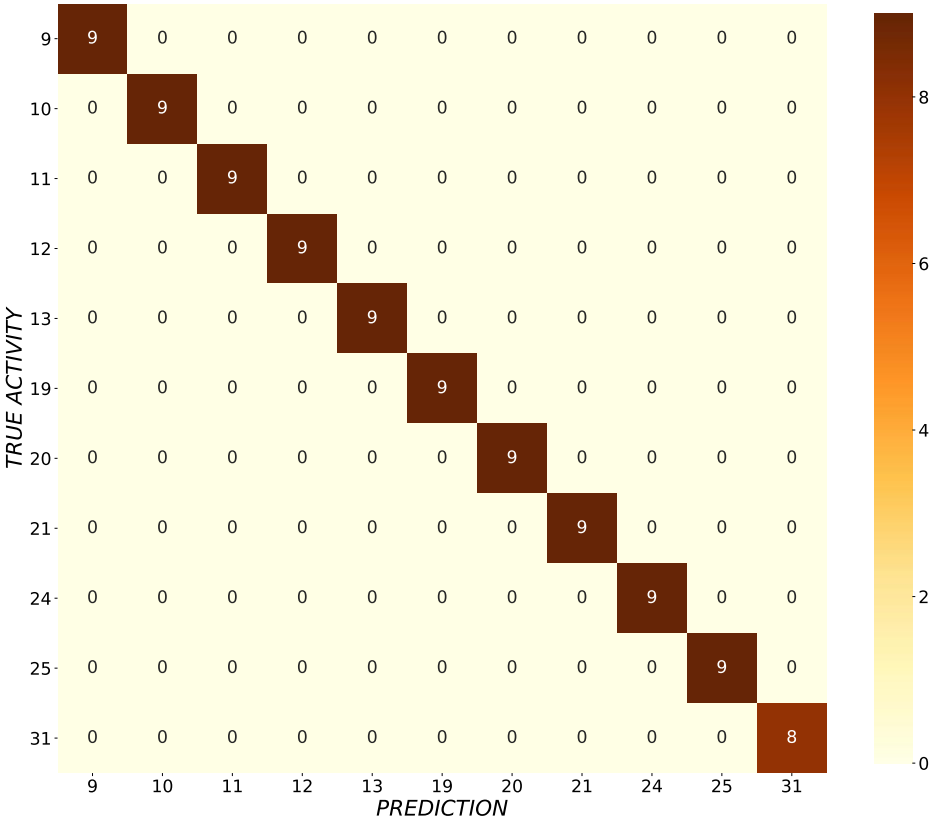


Figura. *Matriz de confusión para el sujeto 9.*

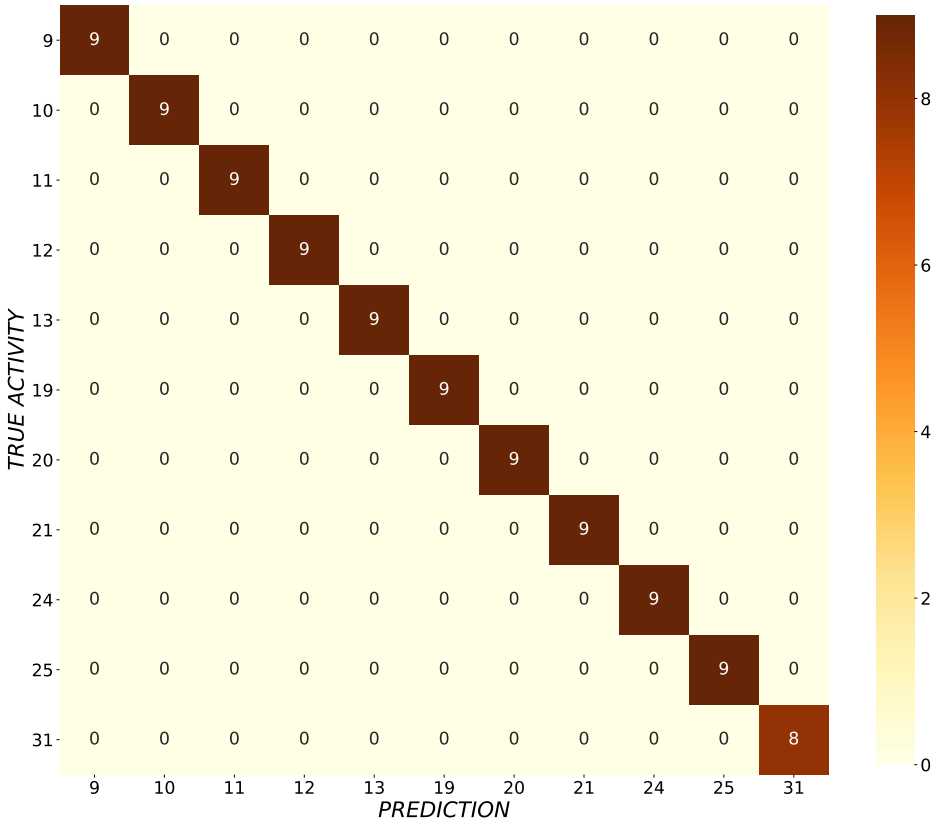


Figura. *Matriz de confusión para el sujeto 10.*

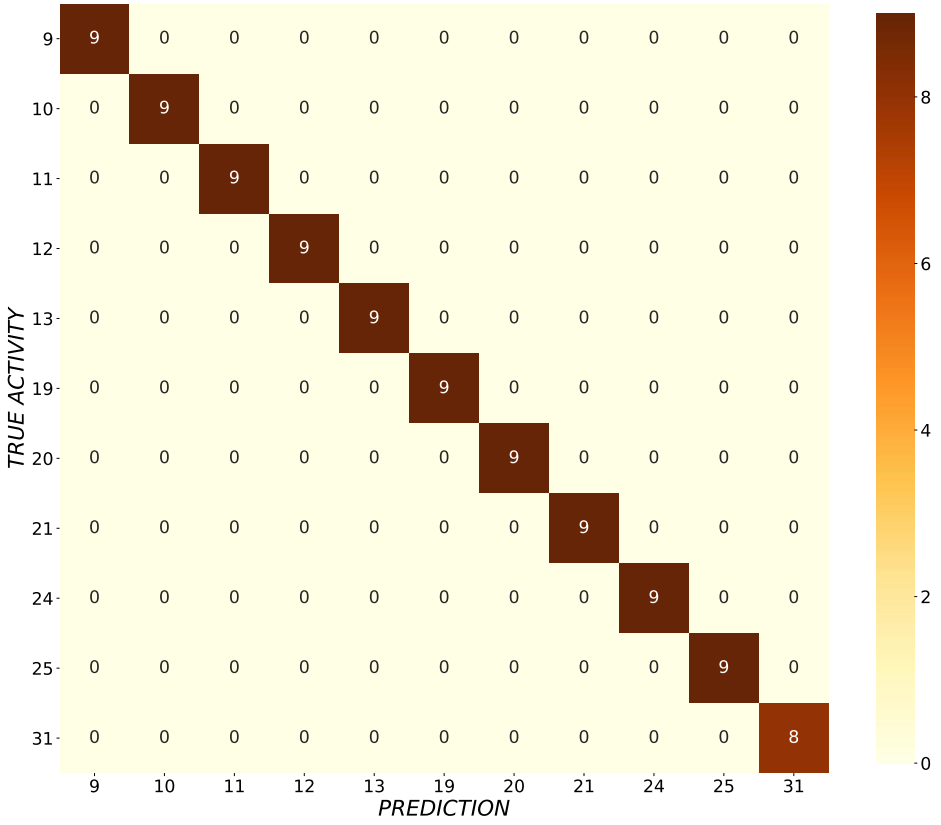


Figura. *Matriz de confusión para el sujeto 11.*

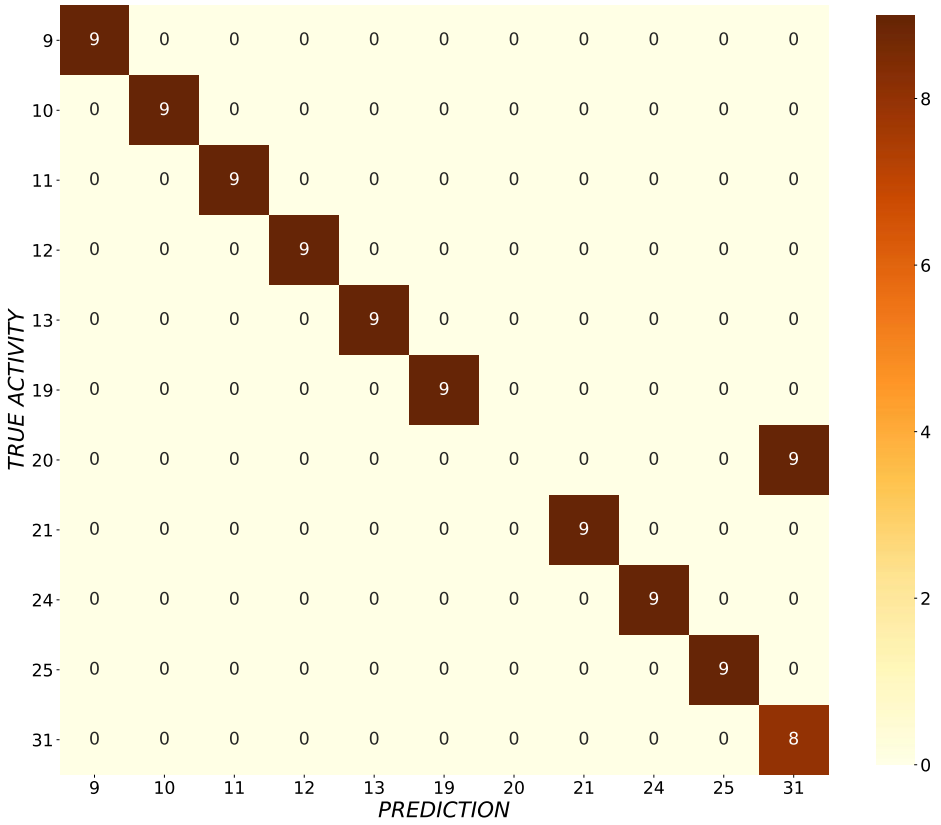


Figura. *Matriz de confusión para el sujeto 13.*

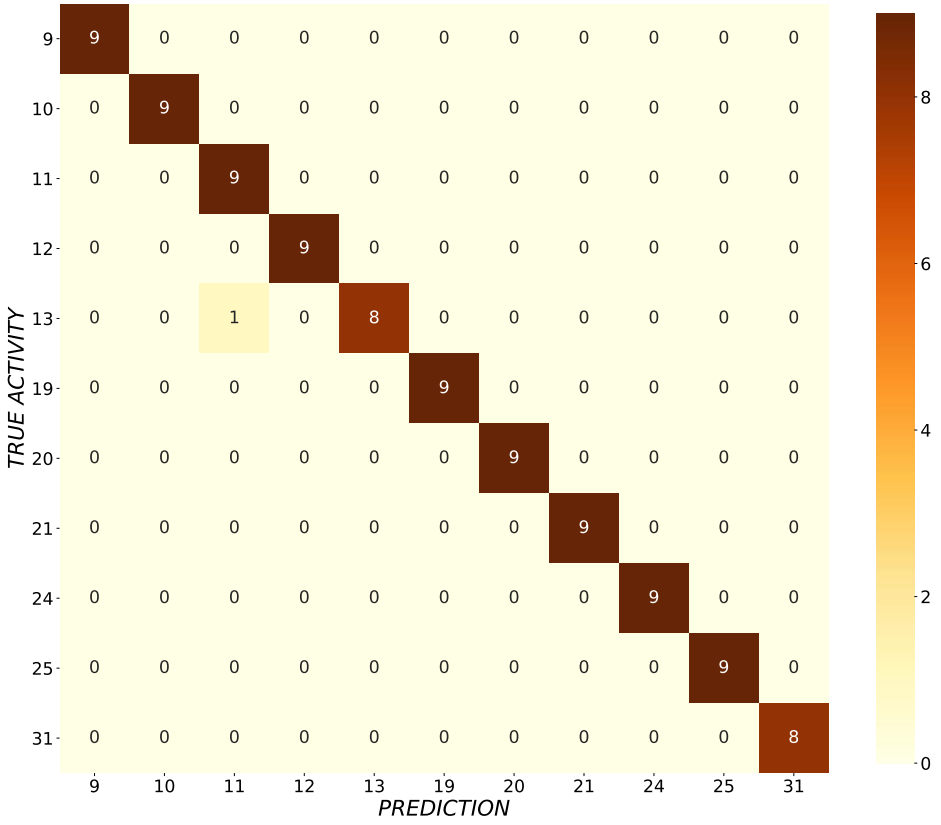


Figura. *Matriz de confusión para el sujeto 14.*

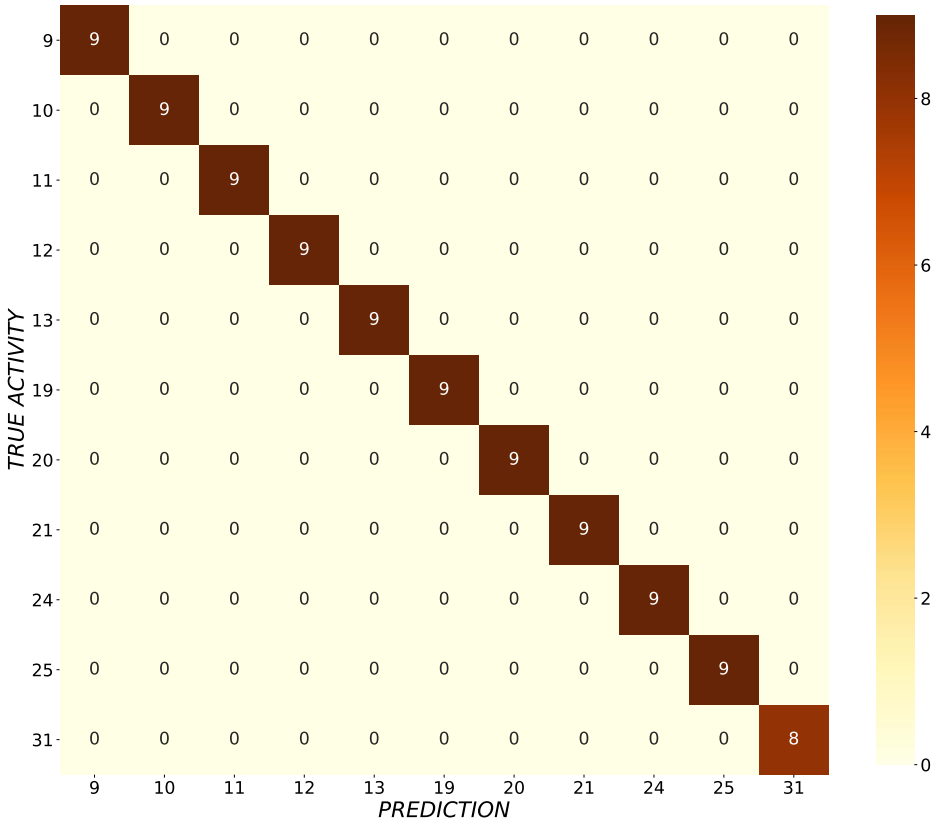


Figura. *Matriz de confusión para el sujeto 16.*

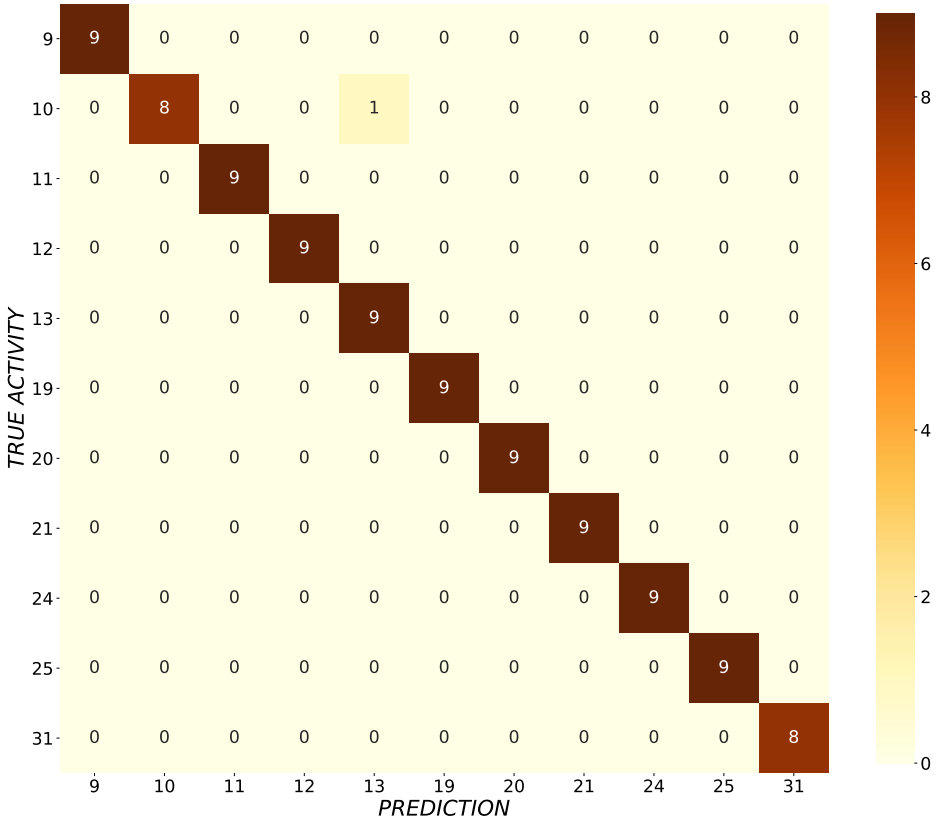


Figura. *Matriz de confusión para el sujeto 17.*

ANEXO II

Resumen de artículo

En este ANEXO II se presenta un resumen de 2 páginas enviado a la **ASPAI'2020 Conference** de Berlín que se celebra del 1 al 3 de abril de 2020 (sitio web: https://www.aspai-conference.com/).

En el caso de ser aceptado, se escribirá un artículo de mayor longitud en el que se presentarán los resultados obtenidos en este Trabajo Fin de Máster.

Oral / Poster / The same

Topic: *Deep Learning*

Deep learning model for upper-body action recognition using body-worn sensors

M. Martinez-Zarzuela[1], S. Saez-Bombín[1], F.J. Díaz-Pernas[1], D. González-Ortega[1] and M. Antón-Rodríguez[1]

[1] University of Valladolid, ETSI Telecomunicación, Paseo de Belén, 15 Valladolid, SPAIN.
Tel.: +34 983 423 000
E-mail: mario.martinez@tel.uva.es

Summary: We introduce a deep learning model for upper-body human activity recognition using body-worn sensors. Our model has been designed so that it needs only quaternion data as input and provides results in real-time, with the aim of being potentially included on a wearable system to monitor daily activities in medical applications. Moreover, we introduce a new technique for data augmentation that helps the model to learn the activities with independence of subject orientation. The proposed model was evaluated using the REALDISP database and overcomes previous methods or achieves comparative results using 30 times less features and 110 times less instances. We obtain an accuracy of 97.7% for a subject-wise clasification and 99.5% for all instances 10-fold evaluation.

Keywords: Deep Learning, Human Action Recognition, Inertial Measurement Units, Wearables, Real-time.

1. Introduction and previous works

Deep learning based methods are being widely adopted for Human Activity Recognition (HAR) using different sensor modalities, including ambient, object and body-worn sensors [1].

Wearable sensors like those that can be found on smartwatches have the potential of continuous monitoring human motion in daily environment, providing valuable information [2]. For many applications in the medical field, it is needed to track the movement of different body joints simultaneously, using a larger number of body-worn sensors [3] [4]. Increasing the number of sensors introduces new opportunities and influences systems complexity and patients discomfort. An interesting recent contribution proposed a light-weight Convolutional Neural Network (CNN) model for arm movement classification and targeting stroke rehabilitation on 3 different tasks. It uses tri-axial acceleration to monitor the progress of rehabilitated subjects under ambulatory settings [5].

In our opinion, the requeriments of an embedded, comfortable and portable system should include wireless communication among body-worn sensors and ubiquitous HAR recognition for feedback interaction. This introduces restrictions on the data size collected by the sensors that can be wirelessly transferred and processed in real-time.

However, most of the solutions for HAR using Inertial Measurement Units (IMUs) that can be found in the literature do not take these restrictions into account. Those machine learning approaches tend to use as much input data as possible and even do feature engineering, in their search to maximize recognition performance.

In this paper we introduce an upper body activity recognition model using 5 IMUs and a data-driven network that can provide real-time high-accuracy recognition rates. The contributions of this work are: (1) Robust Deep Learning model using only Quaternion data from IMUs, (2) New data augmentation technique to introduce rotation invariance of the subject and high accuracy using a very small number of samples.

The work presented here is the first step towards developing a system that in the future could be used during activities of daily living (ADL). In this sense, is similar to [5], but considering the classification of activities in which both limbs and the back participate and using the REALDISP public dataset. This database contains raw (accelerometer, gyroscope and magnetometer) and quaternion data of 33 activities and 17 subjects.

The authors of REALDISP [6] explored the use of classical machine learning algorithms (*K-nearest neighbors*, *Decision Trees* y *Naïve-Bayes*) for classification and obtained an accuracy of 97%, but not considering a *subject-wise* decomposition of samples for training and testing. In a more recent study [7], the authors achieved a *subject-wise* 99.4% accuracy, using feature engineering to generate up to 4086 features.

2. Methods

We manually selected data from 11 upper-body activities out from the 33 activities contained in the dataset. We also discarded subjects in which data for a given activity could not be found or the recorded time was too short for input data balancing, ending up with data of 12 different subjects.

In order to reduce as much as possible the number of features, only quaternion data is used as an input to the network. Instances of 128 time samples were used for training and testing. Six different deep network models based on Convolutional Neural Networks

(CNN) and concatenating a CNN with a Recurrent Neural Network (RNN) were explored. Only results of the winning model are provided in the next section.

For preventing network overfitting, we propose a new data augmentation technique. Using a self-made program on a *Game Engine*, we were able to reconstruct in an avatar the movements on the dataset and reorient the back of all the subjects towards the Earth's north direction. Then, we were able to generate x24 times more data by rotating subjects in the database every 15 degrees.

3. Results and discussion

The results have been obtained using two evaluation methods and a *K-Fold cross-validation* approach, as in [7]: (1) *10-Fold random-partitioning* and (2) *12-Fold subject-wise*.

Table 1, compares the results of this work against previous works [6], [7]. It includes the total number of IMUs, the sensors info used as input (accelerometer, gyroscope, magnetometer and quaternions), the number of features per IMU, the number of activities classified in the work, the number of subjects used for that classification, the evaluation method, the number of instances extracted from the dataset and the accuracy obtained for the classification problem. In our work we only classify 11 activities. On the other hand we use only data from 12 subjects and 5 IMUs instead of 9. More interesting is that the results should be highlighted in terms of the achieved performance using only a fraction of input features and training instances.

In the *random-partitioning* evaluation method we overcome the results obtained in [6], with a 99.5% of accuracy against the 97%. In the *subject-wise* evaluation method, we overcome or have similar results than those obtained in [7]. Using only quaternion data and 40 features we obtain a 97.7% of accuracy, while previous work achieved 93%, but using 30 times less features. We also had 110 times less data instances (1176 vs 130000). Finally, we are only 1.7% below from the best result obtained in [7], using 100 times less input features.

Our model deployed on TensorFlow takes only 500μs to preprocess data and perform the classification, on a not-last-generation commodity card NVIDIA GeForce GTX 1060. Data acquisition from IMUs occurs every 20 ms (50Hz), thus we can consider that our model meets real-time restrictions and could be potentially embedded in a system in a near future.

References

[1] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, 'Deep learning for sensor-based activity recognition: A survey', *Pattern Recognit. Lett.*, vol. 119, pp. 3–11, Mar. 2019.

[2] I. H. Lopez-Nava and A. Munoz-Melendez, 'Wearable Inertial Sensors for Human Motion Analysis: A Review', *IEEE Sens. J.*, vol. 16, no. 22, pp. 7821–7834, Nov. 2016.

[3] N. Jalloul, 'Wearable sensors for the monitoring of movement disorders', *Biomed. J.*, vol. 41, no. 4, pp. 249–253, Aug. 2018.

[4] C. J. Newman *et al.*, 'Measuring upper limb function in children with hemiparesis with 3D inertial sensors', *Childs Nerv. Syst.*, vol. 33, no. 12, pp. 2159–2168, Dec. 2017.

[5] M. Panwar *et al.*, 'Rehab-Net: Deep Learning Framework for Arm Movement Classification Using Wearable Sensors for Stroke Rehabilitation', *IEEE Trans. Biomed. Eng.*, vol. 66, no. 11, pp. 3026–3037, Nov. 2019.

[6] O. Banos, M. Toth, M. Damas, H. Pomares, and I. Rojas, 'Dealing with the Effects of Sensor Displacement in Wearable Activity Recognition', *Sensors*, vol. 14, no. 6, pp. 9995–10023, Jun. 2014.

[7] J. Zhu, R. San-Segundo, and J. M. Pardo, 'Feature extraction for robust physical activity recognition', *Hum.-Centric Comput. Inf. Sci.*, vol. 7, no. 1, p. 16, Dec. 2017.

Table 1. Classification results.

Network	#IMUs	Sensors	#features (per IMU)	#features (total)	#activities	#subjects	Evaluation Method	Number of instances	Accuracy
Oresti [6]	9	ACC+GYR+MAG+QUAT	-	-	33	17	1	-	97.0%
Zhu [7]	9	ACC+GYR+MAG+QUAT	106	4086	33	17	2	130000	99.4%
		QUAT		1224					93.0%
Our work	5	QUAT	8	40	11	12	1	1176	99.5%
							2		97.7%