UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Redes de aprendizaje profundo para reconocimiento de actividades humanas: framework de pre-procesado y entrenamiento en Tensorflow

Autor:

D. Gonzalo Pardo Villalibre

Tutor:

Dr. D. Mario Martínez Zarzuela

Valladolid, 5 de septiembre de 2021

TÍTULO:	**Redes de aprendizaje profundo para reconocimiento de actividades humanas: framework de pre-procesado y entrenamiento en Tensorflow**
AUTOR:	**D. Gonzalo Pardo Villalibre**
TUTOR:	**Dr. D. Mario Martínez Zarzuela**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dra. Dª. Miriam Antón Rodríguez**
SECRETARIO:	**Dr. D. Mario Martínez Zarzuela**
VOCAL:	**Dr. D. David González Ortega**
SUPLENTE 1:	**Dr. D. Francisco Javier Díaz Pernas**
SUPLENTE 2:	**Dr. D. Carlos Gómez Peña**

FECHA:

CALIFICACIÓN:

"You don't understand anything until you learn it more than one way."

Marvin Minsky

Me gustaría agradecer al Dr. D. Mario Martínez Zarzuela por su dedicación y consejo en la confección de este Trabajo Fin de Grado.

Asimismo, hacer especial mención a todas aquellas personas que me han acompañado en este largo camino: padre, hermana y amigos; y en particular a mi madre que tanto dio para que llegase este momento.

Resumen

En esta memoria de Trabajo Fin de Grado se describe la elaboración de un *framework* especializado en el reconocimiento y clasificación de actividades humanas (problema HAR). Para ello, el autor se ha servido del uso de técnicas avanzas de aprendizaje automático y profundo aplicadas sobre dos bases de datos que contienen información de sensores ópticos (VICON) e inerciales (IMU). Dichos sensores aportarán información sobre la posición (vectores tridimensionales) u orientación (cuaterniones) de los sujetos bajo estudio que será utilizada por las redes neuronales en el proceso de clasificación.

Se detalla la construcción del entorno de trabajo, diseñado bajo las máximas de sencillez y versatilidad, capaz de integrar múltiples bases de datos y no sólo aquellas dos utilizadas como fundamento en la confección de este documento[1]. Los resultados obtenidos sobre los conjuntos de datos públicos mencionados anteriormente tras haber sido procesados por el sistema confirman la eficacia del mismo llegando a una solución satisfactoria del problema.

Palabras clave

Framework, reconocimiento de actividades humanas, aprendizaje automático, aprendizaje profundo, sensores ópticos, sensores inerciales, redes neuronales.

[1] *Realdisp activity recognition dataset* y *Hardvard neura sparse dataset*.

Abstract

This final master thesis describes the development of a framework specialized in the recognition and classification of human activities (HAR problem). To do this, the author has made use of advanced machine and deep learning techniques applied to two databases that contain information from optical (VICON) and inertial (IMU) sensors. These sensors will provide information on the position (three-dimensional vectors) or orientation (quaternions) of the subjects under study that will be used by the neural networks in the classification process.

The construction of the work environment will be detailed as designed under the maxims of simplicity and versatility, capable of integrating multiple databases and not only those two used as a basis in the preparation of this document. The results obtained on the aforementioned public datasets after being processed by the system confirm its effectiveness, reaching a satisfactory solution to the problem.

Keywords

Framework, human activity recognition, machine learning, deep learning, ambient sensors, inertial sensors, neural networks.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

El presente Trabajo Fin de Grado (TFG) se ha desarrollado dentro del Grupo de Telemática e Imagen (GTI) de la Universidad de Valladolid. Este grupo de trabajo tiene vasta experiencia en el uso de sensores inerciales (IMU, *Inertial Measurement Unit*) especialmente referida al reconocimiento y clasificación de actividades humanas. Una muestra de ello son el Trabajo Fin de Grado (Sáez Bombín, S., 2018) y el Trabajo Fin de Máster del miso autor (Sáez Bombín, S., 2020), cuyas conclusiones tendrán gran influencia en el presente documento.

Este TFG busca ampliar el campo de investigación mediante la inclusión de sensores ópticos VICON (*Vicon Vantage*), aumentando de esta manera el espacio de aprendizaje que será utilizado por los sistemas de clasificación automática. Las disciplinas de *Machine Learning* (ML) y *Deep Learning* (DL) adquieren, por tanto, un protagonismo esencial en esta elaboración.

Adicionalmente, se pretende crear un sistema capaz de generalizar el problema de clasificación de movimiento humano y optimizar el proceso de resolución desde el planteamiento específico de cada hipótesis hasta el hallazgo de una solución efectiva concreta.

1.1 Motivación

Se describe el problema HAR (*Human Activity Recognition*) como el ejercicio de clasificación y reconocimiento de información temporal multicanal obtenida a partir de sensores que recogen la traslación de un sujeto humano (Zmitri et al., 2019). El objetivo final consiste en determinar certeramente la actividad realizada por el protagonista en cada instante.

A lo largo de los últimos años, el foco de desarrollo se ha centrado en el uso de aprendizaje automático supervisado fundamentado en la capacidad de las redes neuronales profundas para extraer patrones y características inherentes al movimiento de todos los seres humanos (Zmitri et al., 2019).

A diferencia de los excelentes resultados obtenidos en campos similares, como el procesado de lenguaje natural, reconocimiento de lenguaje o visión artificial, el problema de clasificación y reconocimiento de movimiento humano sigue suponiendo un gran reto en la actualidad debido, entre otras razones, a la enorme varianza que introduce cada sujeto al realizar una acción (con sus peculiaridades) o a la similitud entre clases (semejanza entre las actividades realizadas).

Estas dos problemáticas, no obstante, suelen estar presentes también en todas las hipótesis de los campos de éxito anteriormente mencionados. Sin embargo, la mayor adversidad en este caso reside en que una determinada acción está a su vez constituida por un conjunto de movimientos básicos que se suceden en un breve periodo de tiempo y que están presentes del mismo modo en otro sin fin de actividades; ello dificulta sobremanera la caracterización particular de la acción original.

1.2 Hipótesis y objetivos

El movimiento de una mano, la rotación de una cadera o el cabeceo al afirmar captados mediante un sensor que opera a una determinada frecuencia pueden ser interpretados como una señal continua al constituir un flujo constante de datos. Este TFG parte de la hipótesis que asegura la existencia de patrones característicos y esenciales del movimiento humano contenidos en dichas señales y que sirven para caracterizar inequívocamente una acción concreta.

Tradicionalmente las IMUs (Unidades de Medición Inerciales) han sido utilizadas para cuantificar y medir tales señales, haciendo uso de acelerómetros y giroscopios colocados en puntos estratégicos del cuerpo humano, habitualmente donde confluyen dos extremidades: hombro, rodilla, muñeca… El documento se centra, por otra parte, en el uso de los conocidos sistemas VICON compuestos por múltiples cámaras que se disponen alrededor del sujeto y que, del mismo modo que las IMUs, aportan información sobre su posición, velocidad o aceleración. Aunque a priori el planteamiento parece similar, lo cierto es que representa un cambio de perspectiva radical.

Se considera, en definitiva, la posibilidad de abstraer las propiedades únicas de distintas actividades a partir de secuencias finitas de movimiento generadas vía sensores ambientales. Para ello, se pretende hacer uso de redes neuronales profundas que aplicarán de manera sucesiva una serie de transformaciones no lineales sobre las señales obtenidas con el fin de extraer singularidades cada vez más exclusivas.

Finalmente, tras advertir la similitud que presentan entre sí las distintas soluciones propuestas históricamente para resolver el problema HAR, se opta por construir una herramienta capaz de integrar gran parte de ellas adaptándose a las peculiaridades de cada caso de uso.

Por consiguiente, el presente documento tiene **tres objetivos principales**:

1. Acometer la **construcción de una herramienta** capaz de aplicar sin esfuerzo y repetitivamente aquellas operaciones óptimas sobre los datos de entrada que conduzcan a la mejor solución del problema HAR basado en muestras temporales.

2. **Abordar el problema HAR específico referido a la base de datos *Harvard Neura Sparse Dataset***. Se desarrollará un análisis complejo para determinar los modelos de redes neuronales óptimos aplicados a la clasificación de movimiento humano a partir de los datos obtenidos de sensores que reportan información sobre la mitad inferior del cuerpo humano.

3. **Demostrar la validez de los datos obtenidos a través de sistemas de vídeo/ambientales (Vicon Vantage)** de forma alternativa a los sistemas inerciales.

1.3 Fases y métodos

A lo largo de este TFG el autor ha seguido las siguientes fases:

1. **Formación básica en la rama de inteligencia artificial.** De cara a afrontar los retos que presenta esta investigación, el autor se formó previamente en el campo de la inteligencia artificial a través de varios cursos on-line en la plataforma Udemy (Gomila Salas, 2020b, 2020a).
2. **Elección y análisis del dataset**. La base de datos que sirve para plantear la hipótesis inicial fue facilitada por el Dr. D. Mario Martínez Zarzuela. Seguidamente se procedió a un análisis minucioso de la información disponible valorando los posibles fallos que pudiera contener.
3. **Diseño e implementación de la solución**. Con el marco de trabajo inicial establecido y tras advertir la similitud que presentan entre sí los problemas de tipo HAR, se decide crear una solución general capaz de agilizar el proceso de resolución.
4. **Evaluación global de la herramienta**. Una vez creada la herramienta, se prosigue con una fase de pruebas cuya finalidad no es obtener resultados relevantes para la investigación, sino asegurar el correcto funcionamiento de la misma. En esta fase se añaden nuevas funcionalidades potenciales y se depuran errores de implementación.
5. **Diseño e implementación de redes neuronales**. Junto con la tercera fase, es la más extensa de todas: se suceden etapas de diseño de redes neuronales, ajuste de hiperparámetros, reflexión sobre los resultados obtenidos y valoración de posibles alternativas. Además, se decide incluir en la herramienta la capacidad para elaborar métricas de rendimiento a partir de los entrenamientos realizados; de este modo es posible identificar de manera más sencilla y precisa las debilidades y puntos fuertes de cada diseño.
6. **Incorporación de datasets alternativos**. Con los resultados obtenidos tras integrar la base de datos *Neura Sparse* se procede con el estudio sobre el conjunto *Realdisp*, que ya había sido objeto de investigación por parte del alumno Sáez Bombín, S. El objetivo de esta última fase es emular los resultados obtenidos y confirmar el éxito de la herramienta.

1.4 Recursos disponibles

1.4.1 Hardware

El presente TFG ha sido desarrollado en su totalidad sobre un ordenador de sobremesa personal que ejecuta un sistema operativo Linux. Dicha máquina consta de:

- Procesador CPU Intel® Core™ i7-9700k
- Memoria RAM Corsair Vegeance PRO 32 GB a 3200 MHz.
- Tarjeta gráfica GPU Nvidia® GeForce NVIDIA RTX 3080.
- Disco duro Samsung 970 Evo Plus M.2 de 500 GB.

1.4.2 Software

La implementación software del framework es fruto de la cohesión de múltiples tecnologías comúnmente utilizadas en los entornos profesionales más competitivos, pudiendo distinguirse:

Arquitectura

Referido a la parte más estructural del entorno de trabajo (framework).

- **Docker**: tecnología fundamentada en el uso de "contenedores" que permite generar instancias basadas en Linux y que refleja su máxima expresión bajo el plugin de *docker-compose*. Su utilidad reside en la capacidad de mantener entornos de desarrollo estables e independientes de la máquina *host* donde se ejecuta el servicio.

Figura 1. *Logotipo Docker*

- **Flask**: framework desarrollado en Python que facilita la creación de aplicaciones web. Destaca por ser ligero, no necesitar de librerías externas ni requerir el uso de una base de datos.

Figura 2. *Logotipo Flask*

Desarrollo

Referido a la parte funcional del entorno de trabajo.

- **Shell**: intérprete de comandos de Linux que permite generar scripts con el fin de automatizar procesos recurrentes.

***Figura 3**. Logotipo bash*

- **Python:** es un lenguaje de programación interpretado de alto nivel. Su enfoque general y multiplataforma representa la opción más adecuada de cara a implementar una solución dinámica y adaptativa. Tiene integración total con multitud de utilidades de terceros enfocadas al ámbito de la inteligencia artificial como Tensorflow o Keras.

***Figura 4**. Logotipo Python*

- **TensorFlow: "**Es una plataforma de código abierto de extremo a extremo para el aprendizaje automático" (*TensorFlow*, n.d.). Incorpora toda la instrumentación necesaria para aplicar la amplia mayoría de conocimientos teóricos actuales a soluciones software de aprendizaje profundo.

  El código fue liberado en 2015 por Google, que lo había creado como segunda generación del proyecto "*Google Brain*". Sobre TensorFlow se desarrolló posteriormente **Keras**, con un nivel mayor de abstracción facilitando la entrada de nuevos desarrolladores. No obstante, aunque algo más complejo, Tensorflow posee mayor versatilidad y, por tanto, es la utilidad que se ha escogido en la elaboración de esta investigación.

***Figura 5**. Logotipo Tensorflow y Keras*

- **Unity: "**Unity es una herramienta de desarrollo de videojuegos creada por la empresa Unity Technologies… que también se ha utilizado para crear experiencias de Realidad Virtual interactivas e incluso miniseries" (García, 2019). La herramienta nos facilitará la comprensión y representación tridimensional del problema.

***Figura 6.** Logotipo Unity*

1.5 Estructura de la memoria.

La presente memoria ha sido dispuesta en múltiples capítulos emulando el orden lógico-temporal sucedido a lo largo de este Trabajo Fin de Grado.

En el **primer capítulo** se ha descrito la naturaleza del problema, y a continuación se ha planteado la hipótesis inicial y la causa que ha motivado el comienzo de la investigación. Finalmente, se ha recopilado el conjunto de medios teóricos y técnicos disponibles a la hora de abordar la tarea.

En el **segundo capítulo** se describe el contexto científico actual que rodea el campo del aprendizaje profundo incidiendo en la resolución del problema de detección y clasificación de movimiento humano. Se evalúan también las técnicas de ML más relevantes que serán utilizadas. Por último, se analizan las bases de datos protagonistas de este Trabajo.

A lo largo del **tercer capítulo** se realiza una detallada descripción del framework desarrollado, al constituir uno de los elementos más trascendentes. Comenzando con la visión arquitectónica del proyecto hasta los fundamentos de software más esenciales, se revisita en esta sección toda la tecnología utilizada por el autor.

En el **cuarto capítulo** se expone el discurso lógico y teórico que guía la toma de decisiones a la hora de diseñar los diferentes modelos de redes neuronales. Se incorporan los resultados más diferenciales intentando motivar la causa-efecto de los mismos y, en definitiva, se manifiesta el progreso efectivo en la resolución del problema.

Finalmente, en el **quinto capítulo** se exponen las conclusiones más notorias y se discuten las posibles líneas futuras objeto de exploración.

Adicionalmente se incorporan dos anexos:

- **ANEXO I**: presupuesto del proyecto.
- **ANEXO II**: guía de uso del framework. Escrito en inglés, se trata de un documento técnico que detalla todos los requisitos necesarios para poder utilizar el entorno de trabajo. Asimismo, se incluyen las instrucciones de uso para futuros investigadores.
- **ANEXO III**: en este apartado se muestran las matrices de confusión y las métricas obtenidas tras los entrenamientos con las redes neuronales que se han demostrado más relevantes.

Capítulo 2. Revisión del estado de conocimiento

En la última década el reconocimiento de actividades físicas humanas ha suscitado el interés de la comunidad científica, fomentado en gran parte por el auge de la computación ubicua -*ubiquitous computin*- (Zhang & Sawchuk, 2012) . Enfrentada a la tradicional computación de escritorio (*desktop computing*), esta nueva corriente engloba "todos aquellos servicios que permiten al usuario interactuar con los sistemas digitales mediante interfaces naturales como el movimiento o la voz sin necesidad de un teclado o un ratón de por medio" (FRACTALIA, 2016).

Adicionalmente, otro tipo de motivaciones como la corrección de la postura humana, la detección precoz de hábitos motrices incorrectos o la evaluación del rendimiento de los atletas profesionales se han sumado al mar de aplicaciones que generan la necesidad de abordar el problema de detección y clasificación de movimiento humano (*Human activity recognition*).

2.1 Problema HAR

Desde sus inicios, y como ya se ha mencionado en el presente documento, este problema se fundamenta sobre la posibilidad de identificar el movimiento humano a partir de señales captadas por, generalmente, dos tipos de sensores: sensores ambientales o sensores "vestibles", también conocidos como *on-body sensors*.

Los **sistemas de sensores optoelectrónicos** están constituidos esencialmente por una o varias cámaras que se colocan estratégicamente para captar el movimiento de un sujeto. La principal y evidente desventaja reside en la necesidad previa de preparar y mantener la instalación que, además, verá su uso limitado al rango de acción de las propias cámaras.

***Figura** 7. Captura de movimiento vía sensores optoelectrónicos*

Sobre el vídeo obtenido se aplica típicamente un algoritmo de procesado computacional para extraer exclusivamente la información relativa al movimiento; este proceso suele ser costoso en tiempo y dinero. Además, el contenido original de los vídeos puede incluir información que entre en conflicto con la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales, lo que degrada aún más la eficiencia de este tipo de soluciones.

Frente a ello, los **sistemas de sensores vestibles** están formados por sensores inerciales y magnéticos que contienen giroscopios, acelerómetros y magnetómetros. La principal ventaja de estas soluciones es que pueden ser fácilmente integradas en dispositivos de uso común como relojes o smartphones; así, su uso resulta transparente para el usuario final y prácticamente ilimitado en cuanto al área geográfica de aplicación.

No obstante, con el objetivo de conseguir una representación fidedigna de la realidad con valor suficiente para resolver nuestro problema, es recomendable utilizar varios sensores de manera simultánea, como se muestra en la imagen.

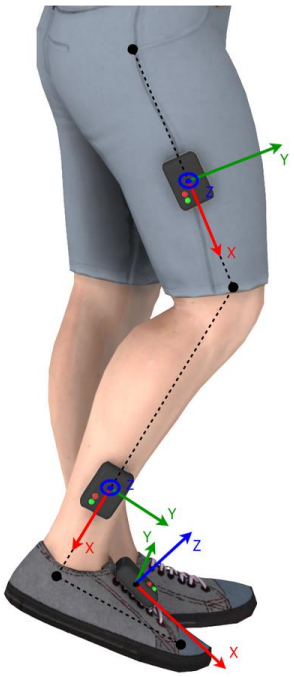


Figura 8. *Uso de múltiples sensores inerciales*

Es precisamente la necesidad del uso de múltiples sensores el Talón de Aquiles de estos sistemas; tal y como se describe en el artículo (Banos et al., 2014)**,** la propia colocación de dichos dispositivos genera una serie de errores de medición derivados que, además de tener difícil solución, son complicados de identificar. No obstante -como se concluye en el citado estudio-, a medida que aumente el número de sensores la incorrecta colocación de uno sólo de ellos tendrá un impacto menor en la decisión final.

Aunque ambas opciones tienen su desventajas y bondades, los sistemas vestibles se demuestran más populares debido esencialmente a su inferior coste y sencillo mantenimiento. Sin embargo, siempre resultará interesante comparar el desempeño de ambas alternativas para cada situación particular.

En cualquier caso, e independientemente del sistema que utilicemos para obtener los datos, el objetivo final del problema HAR será determinar certeramente a partir de las series temporales de datos qué actividad está realizando el protagonista de la acción.

2.2 Estado del arte

En todo estudio fundamentado sobre series de datos temporales generados por sensores, la comunidad científica ha coincidido habitualmente en aplicar un esquema de trabajo similar que se ve perfectamente reflejado en la imagen extraída del artículo científico "***Recent Progress in Sensing and Computing Techniques for human activity recognition and motion análisis***" (Meng et al., 2020).

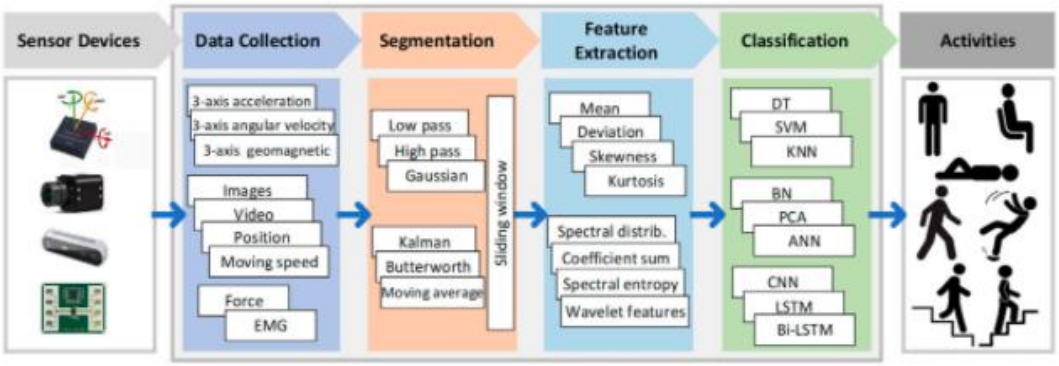


Figura 9. *Esquema de resolución común del problema HAR*

Tras la obtención de los datos es común acometer una fase de **segmentación** que se encarga de dividir las series temporales obtenidas de forma continua en intervalos más pequeños. El objetivo es generar muestras individuales limitadas en el tiempo que puedan ser procesadas y utilizadas por los distintos sistemas de clasificación. Como se explica en el artículo, el **tamaño de la ventana temporal** utilizado para dividir los datos influirá de manera determinante en el resultado del problema; es por ello que esta técnica recibe en ocasiones el nombre de ***sliding window***.

A continuación, sucede una fase de **extracción de características** (*feature extraction)* con el objetivo de resaltar las peculiaridades de los datos obtenidos. En ocasiones dicha fase se reduce a eliminar la información menos relevante de las series de datos y propagar aquella de mayor interés. Sin embargo, a lo largo de los últimos años se han desarrollado multitud de **operaciones de preprocesado** que se pueden aplicar sobre los datos y que se han demostrado certeras a la hora de facilitar el futuro proceso de clasificación:

- Normalización de los segmentos de datos/ventanas temporales:
  - Eliminación de la media: restar la media a todos los datos.
  - Z-Score: dividir los datos entre la desviación estándar.
  - Normalización: dividir cada valor entre la magnitud total del segmento.
  - Ecualización del histograma.

- Obtención de información frecuencial a partir de los segmentos temporales.

- Descarte de los instantes con velocidad nula, técnica conocida como **eliminación de la actividad nula** (Ribeiro et al., 2020).

Una vez generadas las muestras finales que se van a utilizar en el proceso de clasificación, se ha de determinar el **sistema de clasificación automática** a emplear. A pesar del tradicional uso de algoritmos de ML su popularidad ha descendido tras la reciente aparición de las redes neuronales profundas (*Deep Learning*).

Siguiendo el anterior esquema de trabajo, el documento ***Deep Convolutional Neural Networks on Multichannel Time Series For Human Activity Recognition*** (Yang et al., 2015), opta por el uso de redes neuronales convolucionales como sistema de clasificación. Los -quizá no tan alentadores- resultados obtenidos del 86.4% de precisión encuentran su explicación esencialmente en el prematuro estado de la tecnología de redes neuronales en aquel momento y en la precariedad de las operaciones de extracción de características aplicadas.

Más tarde, en el artículo ***Feature extraction for robust physical activity recongnition*** (Jiadong Zhu, Ruben San-Segundo y José M. Pardo, 2017) se propone una solución para el problema HAR haciendo uso exclusivo de algoritmos de ML sobre de la información contenida en la base de datos **REALDISP Activity Recgonition dataset**.

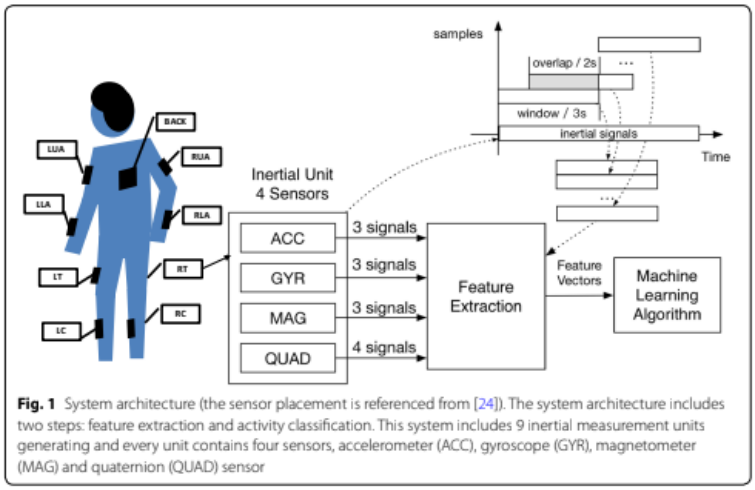


Fig. 1 System architecture (the sensor placement is referenced from [24]). The system architecture includes two steps: feature extraction and activity classification. This system includes 9 inertial measurement units generating and every unit contains four sensors, accelerometer (ACC), gyroscope (GYR), magnetometer (MAG) and quaternion (QUAD) sensor

Figura 10. *Esquema de solución propuesto*

El esquema propuesto por los compañeros de la Universidad Politécnica de Madrid (UPM) coindice, en gran medida y de nuevo, con la "manera de hacer" tradicional que se ha expuesto anteriormente. En este caso los algoritmos de clasificación seleccionados fueron **J48** y **Random Forest**, logrando el segundo resultados del 96,1% de precisión en la clasificación de movimiento.

Se ha hecho especial hincapié en el artículo anterior al poseer extrema relación con el estudio realizado por D. Sergio Sáez Bombín ya que ambos hacen uso de la misma base de datos **REALDISP Activity Recognition dataset.** Por su parte, éste propone una solución que incorpora el uso de redes neuronales profundas novedosas como la combinación CNN+LSTM (red neuronal convolucional seguida de una *Long Short Term*) en lugar de algoritmos de ML.

Los resultados obtenidos en este tercer estudio obtuvieron una precisión similar del 96%, lo que confirma el gran avance de la tecnología desde el primer estudio citado (año 2015). De hecho, las redes neuronales profundas se han demostrado actualmente más eficaces al degradarse la precisión en menor proporción a medida que aumenta el número de actividades a clasificar (en comparación a los algoritmos de ML tradicionales).

No obstante, los resultados obtenidos por D. Sergio Sáez Bombín deben aceptados con precaución al no hacer un uso total de la base de datos -a diferencia del primer estudio- filtrando algunas actividades y sujetos problemáticos.

En definitiva, independientemente del sistema de clasificación elegido queda patente el esquema de trabajo habitual (**Figura 9**) que guiará efectivamente el presente Trabajo Fin de Grado.

2.3 Análisis del caso de uso concreto

Principalmente este TFG se centra en la resolución del problema HAR empleando los datos proporcionados por el estudio "**Replication Data for Estimating Lower Limb Kinematics using a Reduced Wearable Sensor Count"** (Sy, 2019).

El experimento contiene información de siete sujetos realizando diez actividades distintas para cuya medición se han utilizado tanto el **sistema ambiental** Vicon Vantage (ocho cámaras rodeando al individuo) como el **sistema inercial** xSens (7 sensores colocados sobre el cuerpo del sujeto). Como se ha detallado anteriormente el autor empleará sus esfuerzos en el estudio proporcionado por los sensores ambientales Vicon Vantage con el fin de demostrar su valía.

Movement	Description	Time
Static	Stand stil	~10
Walk	Walk straight and back	~30
Figure of 8	Walk in figures of eight	~60
Zig-Zag	Walk zigzag	~60
5-minute walk	Undirected walk, side step, and stand	~300
Speedskater	Speedskater on the spot	~30
Jog	Jog straight and return	~30
Jumping jacks	Jumping jacks on the spot	~30
High kneee	High knee jog straight and return	~30

Tabla 1. *Actividades grabadas en la base de datos*

En cualquier caso, ambos sistemas proporcionan tablas de datos temporales que aportan información sobre la posición u orientación de puntos estratégicos de la parte inferior del cuerpo humano:

SENSOR 3D	DESCRIPCIÓN
PELV	Pelvis
RFEP	Rodilla derecha
LFEP	Rodilla izquierda
RFEO	Tobillo derecho
LFEO	Tobillo izquierdo
LTOE	Pie izquierdo
RTOE	Pie derecho

SENSOR 4D	DESCRIPCIÓN
qRPV	Pelvis
qRSK	Tibia derecha
qLSK	Tibia izquierda
qRTH	Fémur derecho
qLTH	Fémur izquierdo
qRFT	Pie derecho
qLFT	Pie izquierdo

Tabla 2. *Sensores de posición y orientación*

Cada sensor genera un valor con la correspondiente posición u orientación con una frecuencia de 100 Hz; estos valores serán almacenados en tablas individuales en función de la actividad, el sujeto o el número de intento realizado.

Adicionalmente, se integrará en la herramienta creada la ya mencionada base de datos **REALDISP Activity Recognition dataset** (UCI Machine Learning Repository, 2014) utilizada en el estudio realizado por la UPM, así como por D. Sergio Sáez Bombín con tres objetivos:

- Comprobar cuán fácil es adaptar un modelo adicional de base de datos a la herramienta creada.
- Emular los resultados obtenidos por D. Sergio Sáez Bombín.
- Comparar la solución del problema HAR en términos de precisión al entrenar las distintas redes neuronales sobre los datos proporcionados por este segundo estudio basado en la medición vía sensores inerciales en contraposición de los sistemas ambientales.

En este segundo caso, como se aprecia en la **tabla 3**, se trata de nueve **sensores inerciales** colocados a lo largo de todo el cuerpo que plasman los datos de treinta y tres actividades entre las que podemos mencionar saltar a la comba, rotar los hombros o realizar círculos con los brazos abiertos. A diferencia de la primera base de datos, en este caso coinciden los puntos donde se reporta la información de orientación y posición, incluyendo además otros valores como la aceleración o el valor del campo magnético.

SENSORES 3D, 4D Y OTROS VALORES	DESCRIPCIÓN
RLA	Inferior brazo derecho
RUA	Superior brazo derecho
BACK	Espalda
LUA	Superior brazo izquierdo
LLA	Inferior brazo izquierdo
RC	Pantorrilla derecha
RT	Fémur derecho
LT	Fémur izquierdo
LC	Pantorrilla izquierda

Tabla 3. *Sensores inerciales REALDISP*

De nuevo, la información grabada generará tablas de datos con un valor instantáneo cada 50 Hz.

En ambos casos, se ha realizado un filtrado de las actividades objeto de clasificación: de aquellas expuestas en la **Tabla 1** se han eliminado "Jog" al presentar un número de muestras insuficiente y "5-Minute-Walk" puesto que existe información suficiente relativa a la acción de andar en la propia actividad "Walk". Por su parte para las actividades en la base de datos RealDisp, al tratarse en este caso un estudio comparativo con el realizado con D. Sergio Sáez Bombín se han tenido en cuenta únicamente las actividades consideradas en su momento.

Capítulo 3. Framework de preprocesado y entrenamiento

En el capítulo anterior se determina la necesidad y conveniencia de usar técnicas avanzadas de *Machine learning* y *Deep learning* en la resolución del problema HAR. Como consecuencia de la **implementación** de un esquema global y único surge la **herramienta de preprocesado y entrenamiento**. Aunque ciertamente esté centrada en este objetivo concreto, el uso parcial de la misma podría influir directamente en el flujo de trabajo de otras muchas casuísticas o servir como inspiración en otros diseños y problemas.

3.1 Arquitectura Docker

Docker es una herramienta de virtualización capaz de empaquetar software en unidades estandarizadas denominadas "**imágenes**" que incluyen todo lo necesario para que dicho software pueda ser utilizado (*Contenedores de Docker | ¿Qué Es Docker? | AWS*, n.d.) . Cuando una imagen es ejecutada se genera una instancia o **contenedor**, que convive con el sistema host pero que actúa de manera independiente. Existen multitud de imágenes Docker ya creadas que realizan tareas como ejecutar un servidor Nginx (*NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy*, n.d.) o desplegar una pasarela de ciberseguridad como Kong (*Kong API Gateway - KongHQ*, n.d.). Típicamente estas imágenes base se encuentran alojadas en repositorios públicos -véase https://hub.docker.com/ o https://ngc.nvidia.com/catalog/containers- accesibles al para el uso de forma no comercial (o incluso en cuantiosas ocasiones también comercialmente).

En la construcción de cualquier entorno de desarrollo se comenzará esencialmente con una de las citadas imágenes que aportará los requisitos mínimos del sistema. La correcta elección de este *punto de partida* resulta crucial puesto que idealmente contendrá todas las necesidades de nuestro sistema sin incorporar utilidades excesivas que disminuyan la eficiencia del mismo. El objetivo final es conseguir la mínima versión capaz de desarrollar el trabajo deseado.

No obstante, es común que dichas imágenes no satisfagan completamente nuestros requisitos, así que, con el objetivo de completarlas, podemos hacer uso del **Dockerfile**. Este fichero de texto plano habilita la generación de contenedores ejecutando sobre las imágenes primigenias una serie de instrucciones previas que permiten al usuario perfeccionar la instancia a placer descargando las dependencias y programas adicionales.

El diseño de una arquitectura basada en contenedores debe cumplir cuatro reglas básicas (*Docker Development Best Practices | Docker Documentation*, n.d.):

- **Modularidad**: cada contenedor debe ejecutar idealmente una única tarea meridianamente concreta e independiente. El producto final deberá surgir como combinación de las diferentes instancias.
- **Transparencia**: la comunicación entre contenedores deberá reducirse a la mínima expresión; si dos instancias no se relacionan directamente no deberán tener constancia mutua.
- **Escalabilidad**: el entorno debe estar preparado para soportar un aumento de demanda sin que ello suponga un incremento desproporcionado de los recursos necesarios.
- **Manejo de entornos**: se recomienda utilizar una arquitectura adaptada para cada ámbito de despliegue. Habitualmente se puede diferenciar entre los entornos de desarrollo, pruebas, preproducción y producción.

Así pues, la arquitectura planteada para este Trabajo Fin de grado está constituida por los siguientes contenedores o "entornos": el **contenedor de preprocesado**, el **contenedor de entrenamiento** y el **contenedor de inferencia**.

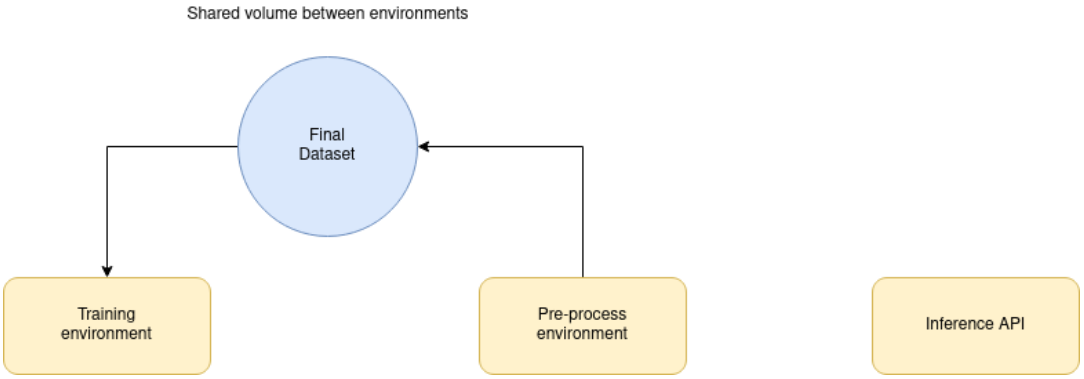


Figura 11. *Arquitectura del framework de preprocesado y entrenamiento*

El **contenedor de preprocesado**, como su propio nombre indica, se encargará de procesar los datasets que contengan información temporal relativa al movimiento de seres humanos aplicando las técnicas de **segmentación** y **extracción de características** revisadas en el **Capítulo 2**. Como podemos observar, está conectado con el **contenedor de entrenamiento** a través de un **volumen** compartido; en este directorio se depositará el resultado del pre-procesado disponiendo el conjunto de datos para comenzar la fase de entrenamiento.

Por su parte, este segundo contenedor proporciona un marco de trabajo listo y estable capaz de abordar la tarea de **clasificación** haciendo uso de las técnicas de *Deep Learning* más actuales. El resultado de los entrenamientos podrá ser cargado en el **contenedor de inferencia** que pone a disposición del investigador un pequeño API REST capaz de responder peticiones de clasificación; es decir, hacer un uso efectivo de los modelos entrenados. A lo largo del presente capítulo analizaremos en profundidad el contenido de los distintos entornos, si bien nos centraremos de momento en la arquitectura *per se*.

3.1.1 Dockerfiles

Como hemos explicado anteriormente, cada contenedor se construye a partir un Dockerfile sobre una imagen base predefinida. En el caso del entorno de preprocesado se ha elegido la imagen **python3** (*Python - Official Image | Docker Hub*). Accesoriamente se han añadido las librerías: funcional **toolz** (*Pytoolz/Toolz: A Functional Standard Library for Python*), matemática **numpy** (*NumPy*) y de análisis de datos **pandas** (*Pandas - Python Data Analysis Library*).

```
1  FROM python:3
2
3  RUN pip install toolz \
4      pip install numpy \
5      pip install pandas
```

Figura 12. *Dockerfile del contenedor de preprocesado*

Los contenedores de entrenamiento e inferencia son similares entre sí, ya que ambos están construidos sobre la imagen **nvcr.io/nvidia/tensorflow:20.12-tf2-py3** creada por Nvidia. Sobre un sistema Linux, incorpora todos los requisitos necesarios para cargar y entrenar modelos de redes neuronales basados en Tensorflow.

Por su parte, el entorno de entrenamiento contiene adicionalmente las siguientes librerías de Python 3 instaladas mediante el fichero *requirements.txt*:

- **Matplotlib**: facilita la creación de visualizaciones en Python (*Matplotlib: Python Plotting — Matplotlib 3.4.3 Documentation*).
- **Sklearn**: incorpora instrumentación para el análisis predictivo de datos (*Scikit-Learn: Machine Learning in Python — Scikit-Learn 0.24.2 Documentation*)
- **Seaborn**: herramienta para representación de datos estadísticos (Waskom, 2021).
- **Pandas**.
- **Keras**. (Capítulo 1).

```
1  FROM nvcr.io/nvidia/tensorflow:20.12-tf2-py3
2
3  COPY requirements.txt /src/
4  RUN pip install -r /src/requirements.txt
```

Figura 13. *Dockerfile del contenedor de entrenamiento*

Además, el contendor de inferencia añade las librerías:

- **Flask** y **flask_restful**. (Capítulo 1).
- **PyJWT** (versión 1.7.1): facilita el tratamiento de ficheros JSON (*Welcome to PyJWT — PyJWT 2.1.0 Documentation*).
- **Request**: incorpora todas las herramientas necesarias para generar una comunicación HTTP en Python (*Requests: HTTP for Humans™ — Requests 2.26.0 Documentation*).

```
1  FROM nvcr.io/nvidia/tensorflow:20.12-tf2-py3
2
3  COPY requirements.txt /app/
4  RUN pip install -r /app/requirements.txt
```

Figura 14. *Dockerfile del contenedor API de inferencia*

3.1.2 Docker-compose

Con el objetivo de definir las relaciones entre instancias para entornos multi-contenedor se creó el *plugin* conocido como "compose"; gracias a un fichero de configuración YAML los desarrolladores pueden caracterizar la totalidad del marco de trabajo permitiendo, entre otras muchas bondades, desplegarlo con un solo comando.

Se trata, por tanto, de un "orquestador" que facilita la coordinación entre contenedores y que será clave en el éxito de la herramienta creada al convertir la ardua tarea de despliegue de la infraestructura en un procedimiento transparente para futuros investigadores.

```yaml
version: "3"
services:

  ######################################
  # Processing environment
  ######################################
  preprocesser:
    build:
      context: ./docker/pre-processing
    volumes:
      - ./framework/pre-processing:/TFG/framework/pre-processing
      - ./framework/final-dataset:/TFG/framework/final-dataset
    working_dir: /TFG/framework/pre-processing
    command: tail -f /dev/null

  ######################################
  # Training environment
  ######################################
  trainer:
    build:
      context: ./docker/train
    volumes:
      - ./framework/train:/TFG/framework/train
      - ./framework/final-dataset:/TFG/framework/final-dataset
    working_dir: /TFG/framework/train
    command: tail -f /dev/null
    deploy:
      resources:
        reservations:
          devices:
          - capabilities: [gpu]

  ######################################
  # Inference API
  ######################################
  inferencer:
    build:
      context: ./docker/inference
    ports:
      - "8082:8082"
    volumes:
      - ./framework/inference:/TFG/framework/inference
    working_dir: /TFG/framework/inference
    command: python3 inferenceServer.py
    environment:
        LOG_LEVEL: DEBUG
        ENV: LOCAL
        SERVER_PORT: 8082
        DEFAULT_NN: N5-250-28-9-1
    deploy:
      resources:
        reservations:
          devices:
          - capabilities: [gpu]
```

Figura 15. *Docker-Compose*

Como observa en la imagen anterior, para cada entorno se define el parámetro "build" que es la ruta donde se encuentra el Dockerfile usado como referencia. A continuación, se caracterizan los volúmenes de memoria compartidos entre la máquina host y las distintas instancias; cabe destacar en este sentido el usado para almacenar el dataset tras el preprocesado que se comparte entre los entornos de entrenamiento y preprocesado mediante las líneas 12 y 24.

Por otro lado, los contenedores pueden hacer uso de los recursos hardware del sistema; por ejemplo, la tarjeta gráfica dedicada (líneas 31 y 54). También se puede definir un mapeo de los puertos TCP/IP de la máquina host con los contenedores: en este caso el entorno de inferencia hará uso del puerto local 8082 (línea 40) donde será accesible el API REST (línea 48).

3.1.3 Utilidades transversales: GNU Make y JSON

A lo largo del presente documento se mencionará en múltiples ocasiones la utilidad **Make** (*Make - GNU Project - Free Software Foundation*); se trata de "una herramienta que controla la generación de ejecutables… que toma la información a partir de un fichero de texto denominado *makefile*". De esta manera, el diseñador ahonda en la idea de transparencia haciendo el entorno mucho más accesible.

```
Usage: make <command>

Commands:
  help:                Show this help information
  develenv-up:         Launch the development environment with a docker-compose of the service
  preprocess-sh:       Access to a shell of a launched preprocessing environment
  train-sh:            Access to a shell of a launched training environment
  inference-sh:        Access to a shell of a launched inference environment
  develenv-down:       Stop the development environment
```

Figura 16. *Opciones make sobre el framework global*

Para interactuar con el framework en primera instancia sólo será necesario ejecutar el comando "*make*" seguido de cualquiera de las siguientes instrucciones:

- **Help:** desplegar lista de opciones *make.*
- **Develenv-up:** desplegar localmente el framework.
- **Develenv-down:** apagar el despliegue local del framework.
- **Preprocess-sh:** acceder al contenedor de preprocesado.
- **Train-sh:** acceder al contenedor de entrenamiento.
- **Inference-sh:** acceder al contenedor de inferencia.

Destaca el uso del formato JavaScript Object Notation (**JSON**) con el objetivo de modelar todos los parámetros configurables de la herramienta. Tal y como se describe en documentos como "Speed up development with a JSON Configuration File" (Koo, 2020) este formato permite generar código versátil de forma sencilla.

El uso de estas dos tecnologías transversales tiene como objetivo, en definitiva, facilitar el acceso a la herramienta abstrayendo los detalles más técnicos aportando robustez y transparencia.

3.2 Entorno de pre-procesado

El entorno de preprocesado dispone una variedad de técnicas de ML que serán aplicadas sobre el conjunto de datos de entrada dejando cierto margen de decisión al investigador. El objetivo es modificar el dataset original para facilitar la posterior fase de entrenamiento de redes neuronales.

Este contenedor es el más específico de todos, en lo que a la resolución del problema HAR se refiere y, en esencia, el único requisito para poder usarlo es que los datos de entrada correspondan a muestras temporales obtenidas a partir de sensores 3D o 4D. De este modo, la información de cada actividad realizada por un determinado sujeto deberá estar definida en un fichero *.csv* con el siguiente formato.

	A	B	C	D	E	F	G	H	I	J	K	L
1	PELV_1	PELV_2	PELV_3	LFEP_1	LFEP_2	LFEP_3	LFEO_1	LFEO_2	LFEO_3	LTIO_1	LTIO_2	LTIO_3
2	-0.01596490637	0.32289928	0.9415684609	-0.08953610405	0.2581864589	0.8555807335	-0.1080942134	0.2374734178	0.4575910444	-0.1008051018	0.1987609019	0.0795145425
3	-0.01609043372	0.3228450544	0.941599039	-0.08966353519	0.2581835116	0.8555738502	-0.1080693391	0.2376779534	0.4577460038	-0.1008191894	0.1987241691	0.0794928496
4	-0.01622681463	0.3227940252	0.9416261464	-0.08980601529	0.2581763648	0.8555741732	-0.1080208231	0.2379179089	0.4579481067	-0.1008288018	0.1986784834	0.07946858821
5	-0.01637253285	0.3227462632	0.9416517696	-0.0899628141	0.2581640531	0.8555827151	-0.1079498142	0.2381992218	0.4581952499	-0.1008323655	0.198620896	0.07944270094
6	-0.0165253321	0.3227013085	0.9416753984	-0.09013157972	0.2581446466	0.8556004928	-0.1078605412	0.2385286315	0.4584783394	-0.100829274	0.1985514352	0.07941598245
7	-0.01668210493	0.3226592481	0.9416980053	-0.09030920467	0.2581182324	0.8556284789	-0.1077622481	0.2389127293	0.4587863252	-0.1008194215	0.1984701164	0.0793891317
8	-0.01683929398	0.3226201787	0.9417205592	-0.09049177337	0.2580849339	0.8556686391	-0.1076632841	0.2393620896	0.4591060743	-0.1008042133	0.1983819203	0.07936266638
9	-0.01699425542	0.322585196	0.9417450176	-0.09067520907	0.2580458406	0.8557209186	-0.1075749368	0.2398852591	0.4594265169	-0.1007855896	0.198288812	0.0793369656
10	-0.01714290564	0.3225548801	0.9417708355	-0.09085506879	0.2579990498	0.8557853163	-0.1075076961	0.2404987466	0.4597334253	-0.1007667519	0.1981997142	0.07931148728
11	-0.01728317002	0.3225307956	0.9417989662	-0.09102724211	0.2579486522	0.8558617208	-0.1074719491	0.241217194	0.4600196079	-0.1007513803	0.1981225375	0.07928561243
12	-0.01741316288	0.3225140025	0.9418303742	-0.09118837199	0.2578917025	0.8559491593	-0.1074742251	0.2420613096	0.4602767259	-0.1007433597	0.1980651763	0.07925812396
13	-0.01753150144	0.3225030311	0.9418650756	-0.09133577295	0.2578292784	0.8560485895	-0.1075200257	0.2430529366	0.4605024117	-0.1007463458	0.1980385212	0.07922724612
14	-0.01763783451	0.3224968424	0.9419005867	-0.09146793774	0.2577553565	0.8561571118	-0.1076138887	0.2442159584	0.4606952514	-0.1007630404	0.1980464814	0.07919122715
15	-0.01773410823	0.3224878648	0.9419355665	-0.09158539764	0.2576619058	0.8562738825	-0.1077532902	0.2455735066	0.4608587875	-0.1007953865	0.1980949903	0.07914867081
16	-0.01782182157	0.3224689436	0.9419636615	-0.0916901169	0.2575407891	0.8563940707	-0.1079376971	0.2471486881	0.4609975797	-0.1008431483	0.1981820341	0.07909861299
17	-0.01790605907	0.3224289006	0.9419816246	-0.09178560163	0.2573778912	0.8565159719	-0.1081594679	0.2489597655	0.4611162201	-0.100904274	0.1983006519	0.07904086812
18	-0.01799030442	0.3223555465	0.941983714	-0.09187645189	0.2571609603	0.8566318547	-0.1084108506	0.2510200033	0.4612193944	-0.100974514	0.198439953	0.07897650686
19	-0.01807872575	0.3222376486	0.9419626752	-0.09196736161	0.2568827808	0.8567358931	-0.1086808592	0.2533317177	0.4613109679	-0.101047259	0.1985821114	0.07890741147
20	-0.01817395827	0.3220640827	0.9419152399	-0.09206186386	0.256537125	0.8568202328	-0.1089553481	0.2558892564	0.46139293	-0.1011142807	0.1987043548	0.07883654442
21	-0.0182787309	0.3218302231	0.9418380175	-0.092161547	0.2561248081	0.8568768698	-0.1092199705	0.258675877	0.461462477	-0.1011666211	0.1987829457	0.07876788058
22	-0.01839542808	0.3215324893	0.9417295952	-0.09226681324	0.255650616	0.8568947146	-0.1094590566	0.2616598464	0.4615160773	-0.1011956962	0.1987931457	0.07870581694
23	-0.01852408282	0.3211738432	0.941587917	-0.0923771339	0.2551244484	0.8568675745	-0.1096577179	0.2647983578	0.4615474146	-0.1011944578	0.1987122017	0.07865572597
24	-0.01866536782	0.3207608172	0.9414153653	-0.0924917214	0.2545621927	0.856788141	-0.1098028169	0.2680365274	0.461548415	-0.1011586706	0.1985222916	0.07862251029
25	-0.01881903329	0.3203049271	0.9412122206	-0.09261075965	0.2539837526	0.8566510381	-0.1098860987	0.2713133265	0.4615091613	-0.101088276	0.1982135002	0.07861085782
26	-0.01898514856	0.3198197376	0.9409817276	-0.09273572097	0.2534100784	0.856454882	-0.109905127	0.2745606544	0.4614229189	-0.1009880255	0.1977847793	0.07862382866

Figura 17. *Ejemplo de dataset válido para sensores de posición 3D*

Como se puede apreciar, las columnas contienen información de cada sensor 3D o también denominados **sensores de posición** (coordenadas X, Y, Z, respectivamente) mientras que las filas identifican los sucesivos instantes temporales. Como se ha mencionado, el framework también está preparado para trabajar con **sensores de orientación** que aportan la información en forma de cuaterniones (coordenadas W, X, Y, Z). Evidentemente ambos formatos no son de elección aleatoria puesto que son, actualmente, las maneras habituales de cuantificar el movimiento humano.

	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM
1	qRPV_1	qRPV_2	qRPV_3	qRPV_4	qRTH_1	qRTH_2	qRTH_3	qRTH_4	qLTH_1	qLTH_2	qLTH_3	qLTH_4
2	0.6896556069	-0.09788637429	0.04602977748	0.7160130314	0.7451814626	-0.03159989745	0.002034859486	0.6661095207	0.5687920519	0.004352291384	0.03460251955	0.8217416412
3	0.689758001	-0.09778041584	0.04583378003	0.7159414466	0.7457822757	-0.03103596094	0.001652863462	0.665464375	0.5691753353	0.004329490462	0.03429691361	0.8214891447
4	0.6898612128	-0.09770065485	0.04569490092	0.715861764	0.7463622452	-0.03043907071	0.001242625736	0.6648423255	0.5697477878	0.004288090473	0.03392709745	0.8211078021
5	0.6899623692	-0.09764285453	0.04560878474	0.7157776476	0.7469139669	-0.0298047763	0.0008020581559	0.6642518785	0.5704827843	0.004236390589	0.03348663301	0.8206156781
6	0.6900591017	-0.09761128259	0.04558868027	0.7156899789	0.7474726788	-0.02912830254	0.0003275016224	0.663653546	0.5713479963	0.004189025178	0.03296999535	0.8200346935
7	0.6901552241	-0.09760240246	0.04562557497	0.7155961463	0.7480703304	-0.0283986687	-0.0001887165833	0.6630115088	0.5722767345	0.004166409561	0.0323777583	0.819410557
8	0.6902472108	-0.09762102677	0.04572340009	0.715498633	0.7487291747	-0.02761311233	-0.0007480319043	0.6623002185	0.5731779379	0.004192549325	0.03170615021	0.8188065669
9	0.6903355181	-0.09766582134	0.04587505036	0.7153976094	0.7494950083	-0.02676209008	-0.001358319076	0.6614674429	0.5739401888	0.004293070801	0.03095707957	0.8183006101
10	0.6904119859	-0.09774111386	0.04607992624	0.7153003599	0.7504086778	-0.02584207072	-0.002022800434	0.6604656781	0.5744223893	0.004501735138	0.03012052921	0.8179923024
11	0.6904785255	-0.09784779342	0.04633354788	0.7152051576	0.7515201791	-0.02484467485	-0.002747907441	0.6592570183	0.5744784597	0.00484572975	0.0291946274	0.8179845304
12	0.6905293936	-0.09798669943	0.04662381156	0.7151181605	0.7528164109	-0.02376186491	-0.003539035838	0.6577919887	0.5739595294	0.00535613821	0.02816014384	0.8183818038
13	0.6905561746	-0.09815844366	0.04695686248	0.7150469514	0.7543620644	-0.02258565677	-0.004398330132	0.6560551948	0.5727314103	0.006058481828	0.02699906356	0.8192759468
14	0.6905525828	-0.09836372235	0.04732988074	0.714997616	0.756158186	-0.02129739776	-0.005334703209	0.654020458	0.5706688056	0.006980880705	0.02568140281	0.820748955
15	0.6905105715	-0.09860057755	0.04774411225	0.7149780251	0.7581967498	-0.01987680169	-0.006353897485	0.6516918208	0.5677083615	0.008139074062	0.02416821658	0.8228346548
16	0.6904196634	-0.09887803657	0.04820986693	0.7149962455	0.7604690558	-0.01829147097	-0.007468262297	0.6490735416	0.5638175134	0.009546478796	0.02241955166	0.8255398477
17	0.690273462	-0.09918702355	0.04872772659	0.7150595015	0.7629326162	-0.01650761925	-0.008686984695	0.6462088346	0.55903944	0.01120389729	0.0203869237	0.828814666
18	0.6900746701	-0.09952703492	0.04930334571	0.7151646657	0.7655514969	-0.01448357243	-0.0100234767	0.643133471	0.5534655837	0.01310295974	0.01802792494	0.8325738129
19	0.689820034	-0.09989319714	0.04993366146	0.7153155244	0.7682589765	-0.01218474897	-0.01148294655	0.639920166	0.5472615908	0.01521771214	0.01531532004	0.8366831021
20	0.6895203647	-0.1002671115	0.05059735733	0.715505472	0.7709889444	-0.009584018087	-0.01306603628	0.6366423431	0.5406388922	0.01751339899	0.0122319612	0.8409835005
21	0.6891822479	-0.100634252	0.05126383106	0.7157322098	0.7736531661	-0.006668486807	-0.01476450579	0.6334021782	0.5338468385	0.0199459991	0.008784282378	0.84530027
22	0.6888095229	-0.100963457	0.05189856599	0.715998855	0.7761612379	-0.00345388597	-0.01655653807	0.630307611	0.5271934234	0.02246236637	0.005004186963	0.8494336316
23	0.6884112327	-0.1012387735	0.05246085455	0.7163019923	0.778409957	2.23E-05	-0.01841319111	0.6274861694	0.5209765388	0.02500675363	0.0009466889271	0.8532040858
24	0.6879990255	-0.1014244067	0.05290176689	0.7166392632	0.7802807543	0.003691843893	-0.02029724813	0.6250890628	0.5154934948	0.02752151457	-0.003307180286	0.8564450278
25	0.6875828882	-0.1015026909	0.0531966242	0.7170056449	0.7816693606	0.007466899681	-0.02217107097	0.6232541212	0.5109892387	0.02995456035	-0.007659690583	0.8590308792
26	0.687176103	-0.1014499267	0.05332257735	0.7173936288	0.7825058346	0.01124568579	-0.02399967231	0.6220789091	0.5076623647	0.0322602773	-0.01199946028	0.8608682889

Figura 18. *Ejemplo de dataset válido para sensores de orientación 4D*

El investigador deberá realizar un trabajo de adaptación previo uso del framework con el objetivo de adaptar mínimamente su dataset al formato de entrada anteriormente presentado. En la **guía de uso (Anexo II)** se detallan en profundidad todos los requisitos técnicos. Como complemento, se ha facilitado una muestra de *tuneado* para dos bases de datos distintas con el fin del ilustrar a los futuros usuarios de la plataforma.

Esta fase no es en ningún caso baladí ya que además será imprescindible que los ficheros *.csv* estén correctamente nombrados identificando el sujeto, la actividad y el *trial* (intento o número de prueba). Así, el entorno de preprocesado podrá realizar el filtrado de aquellos que sean relevantes de forma automática.

En conclusión, **este entorno está preparado para trabajar con tablas que contienen series temporales de datos correspondientes al movimiento de un ser humano realizando una determinada actividad.**

Está distribuido en cuatro módulos pseudo-independientes que aplican diferentes operaciones de manera secuencial sobre los datos de entrada, y se distinguen:

- Módulo de “tablas de datos intercaladas” o interleaved-dataframe.
- Módulo de “construcción de imágenes” o image-builder.
- Módulo de “enriquecido de imágenes” o image-enricher.
- Módulo de “dataset final” o final-dataset.

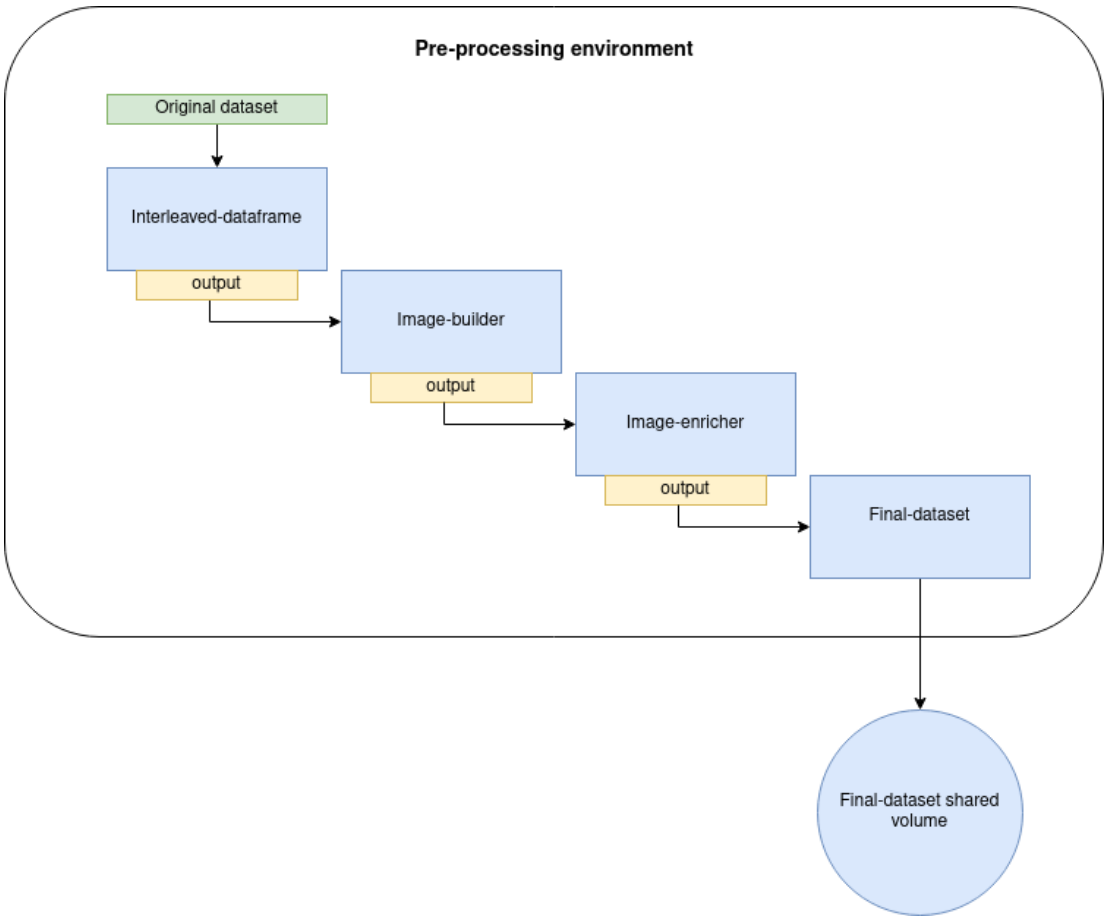


***Figura 19**. Arquitectura del entorno de pre-procesado*

3.2.1 Interleaved-dataframe

La función de este módulo es reorganizar el formato de los datos originales para representarlos con mayor facilidad en **Unity 3D**. Dicha representación se realizará a través de varios scripts desarrollados por el Doctor D. Mario Martínez Zarzuela y resultará fundamental en la resolución del problema, ayudando al investigador a analizar visualmente los movimientos grabados y así determinar posibles errores de medición o valorar diferentes hipótesis, como veremos a lo largo del **Capítulo 4**.

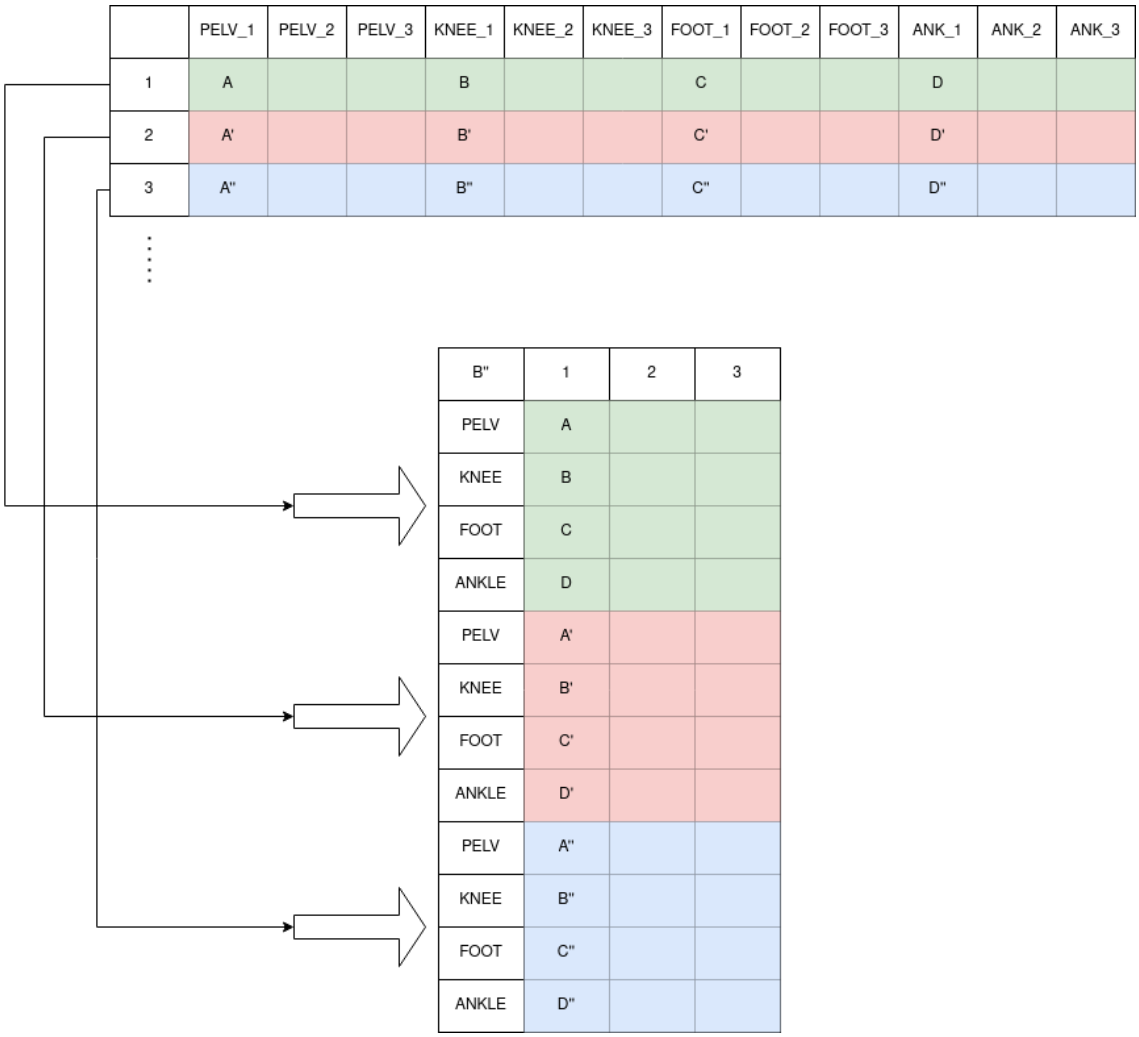


Figura 20. *Transformación efectuada por el módulo interleaved-dataframe*

La transformación consiste en trasladar cada sensor perteneciente a un mismo instante temporal a una nueva fila adicional; de este modo, obtendremos finalmente una tabla con tres o cuatro columnas (si estamos usando sensores de posición u orientación respectivamente) y tantas filas como instantes temporales teníamos inicialmente multiplicadas por el número de sensores utilizados.

Adicionalmente, se podrá realizar un filtrado del dataset original para discriminar los sujetos, actividades, sensores de posición y sensores de orientación deseados en el estudio. Así, se presenta la posibilidad de atacar también la resolución del **problema HAR minimizando el número de sensores usados**. Sin embargo, no será posible combinar información de sensores de orientación y movimiento a lo largo del framework ya que se tratarán de forma separada.

Figura 21. *Representación de varios movimientos en Unity*

3.2.2 Image-builder

La primera función de este segundo módulo es recuperar el formato de las tablas de datos originales asegurando además que no existe ningún carácter extraño y transformando la nomenclatura científica a una nomenclatura estándar para facilitar el procesado. Sin embargo, su tarea principal y que da nombre a este bloque es aplicar la técnica de **segmentación** que se ha descrito en el **Capítulo 2**.

A partir de un determinado fichero que contiene la información de un sujeto realizando una determinada actividad durante N instantes de tiempo, se generarán nuevas tablas de datos con X instantes de tiempo cada una, donde X siempre será inferior a N. Desde ahora momento nos referiremos a estas nuevas subtablas bajo el nombre de **imágenes de movimiento**.

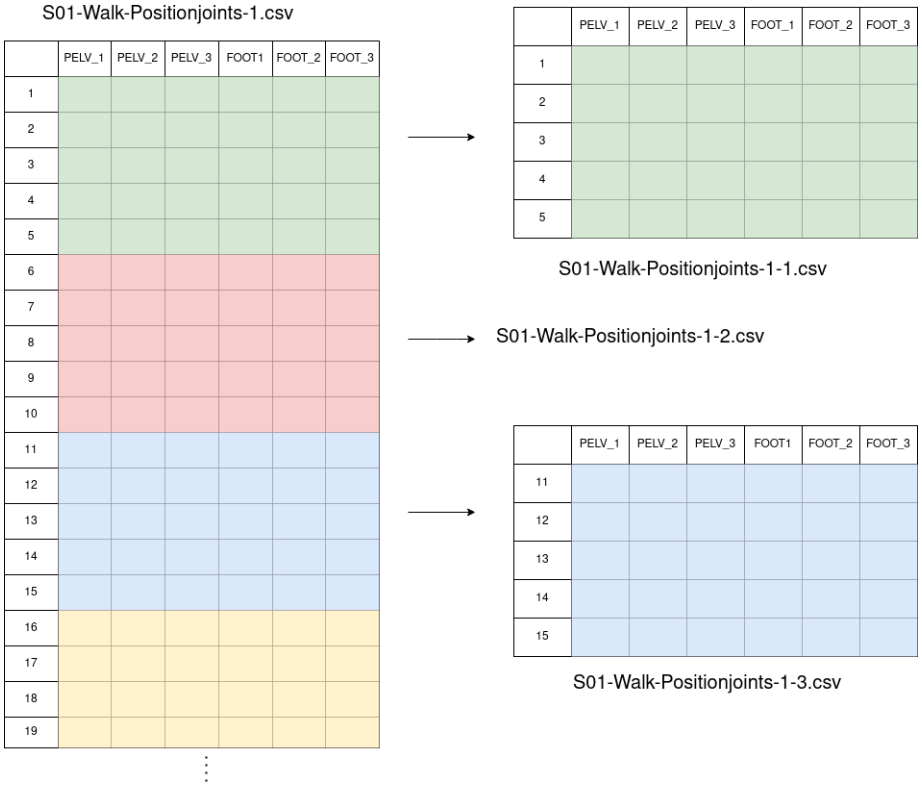


Figura 22. *Generación de imágenes a partir de ficheros originales*

Cada imagen contendrá, por tanto, una porción de información de los ficheros de movimiento originales y servirá para alimentar a las futuras redes neuronales. A fin de aumentar el número de imágenes final -puesto que se suele trabajar con bases de datos limitadas-, también se puede definir cierta superposición entre los datos incluidos en cada imagen.

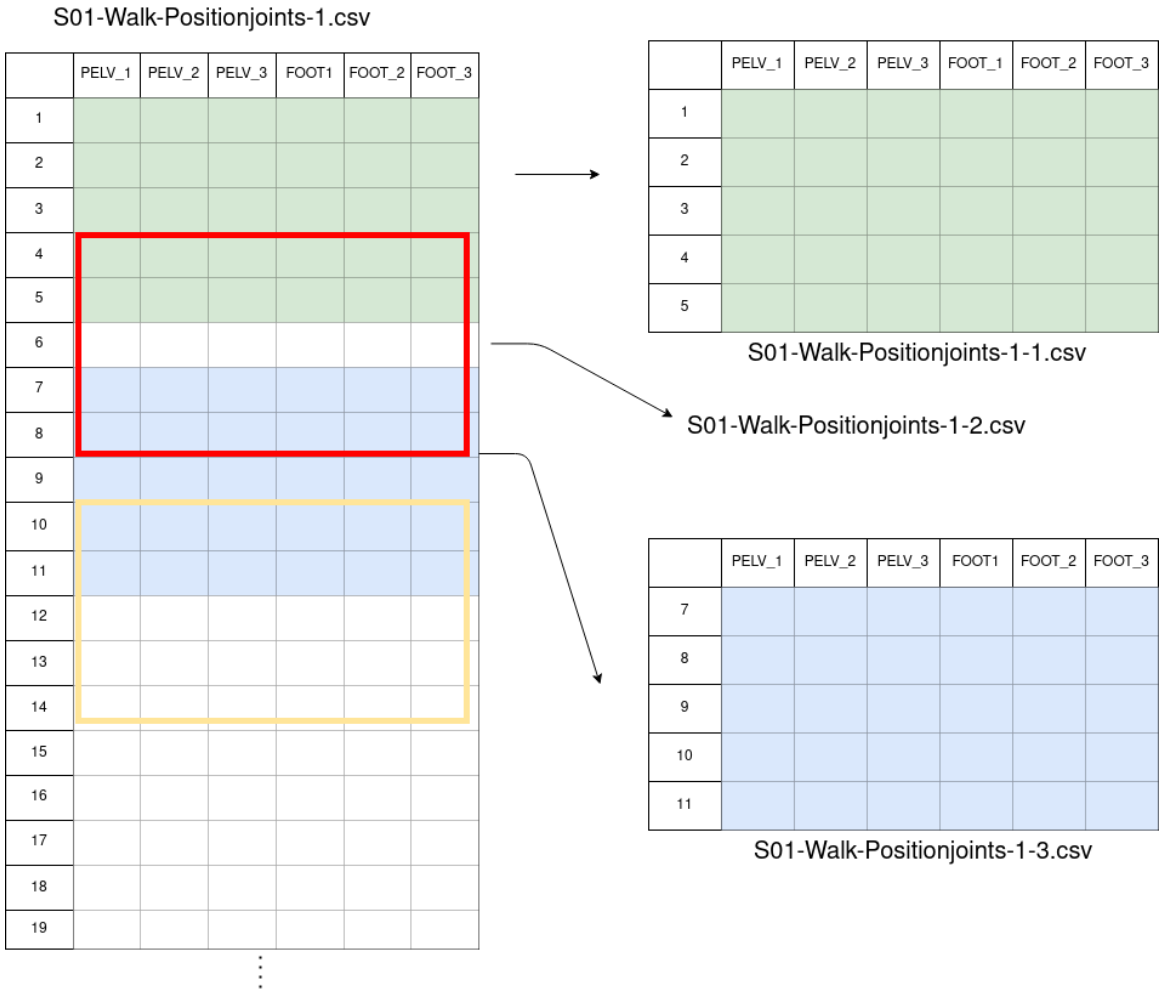


Figura 23. *Generación de imágenes con superposición a partir de ficheros originales*

A partir de este momento, todas las utilidades estarán determinadas al máximo a facilitar el aprendizaje de las redes neuronales; para ello es esencial disponer de tantas muestras en los conjuntos de entrenamiento, test y validación como sea posible. Además, dichas muestras deberán contener la información más completa y relevante.

En ese sentido, cada imagen resultante está nombrada de manera estratégica identificando totalmente su contenido. Por ejemplo, **S09-HighKneeJog-Orientationjoints-2-40-0.csv** donde cada campo significa respectivamente**:**

- **Sujeto que realiza el movimiento**: S09.
- **Actividad**: HighKneeJog.
- **Sensores utilizados**: orientación.
- **Intento/trial**: segundo intento.
- **Número de imagen generada**: se trata de la imagen número 40.
- **Grados de rotación**: al haber 0 grados de rotación se trata de una imagen que no proviene de *data augmentation*, opción que será comentada a continuación.

3.2.3 Image-enricher

El tercer módulo tiene como función enriquecer las imágenes previamente creadas habilitando la incorporación de información frecuencial, técnica que se ha descrito en el **Capítulo 2** en el apartado de extracción de características. El uso de este recurso es opcional; de hecho, es posible:

- Conservar imágenes exclusivamente temporales.
- Generar imágenes con contenido exclusivo frecuencial a partir de la FFT 2D calculada sobre las imágenes temporales.
- Combinar la información temporal y frecuencial.

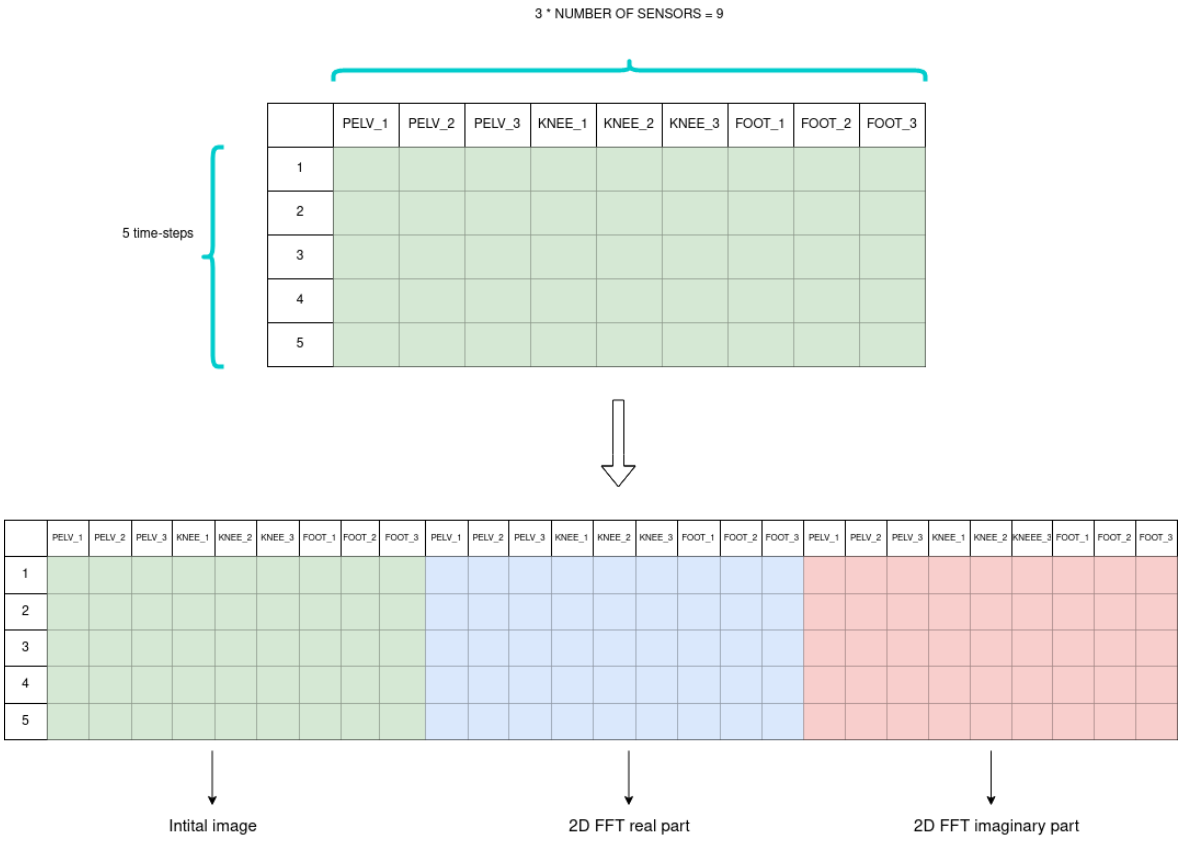


Figura 24. *Generación de imagen enriquecida combinando información temporal y frecuencial*

Adicionalmente, este módulo permite aplicar ***data augmentation*** a través de la **rotación vertical** de sujetos con el fin de aumentar el número de muestras. El investigador puede detallar una lista con diferentes grados de giro que se aplicarán sobre la imagen temporal original, de suerte que al final obtendrá una simulación del sujeto realizando la actividad en otra dirección. Seguidamente, se calcula la FFT 2D para cada una de las imágenes rotadas generadas inorgánicamente, si así se desea.

3.2.4 Final-dataset

Este último módulo es el más sencillo de todos y su única función es filtrar las imágenes que se trasladarán al volumen compartido tras el pre-procesado. Así, se podrá volver a discriminar entre actividades, además de indicar si se desea elegir las imágenes exclusivamente temporales o aquellas generadas en el módulo previo con contenido frecuencial.

3.2.5 Configuración (JSON)

Previamente se han mencionado multitud de características dependientes del criterio del investigador, entre las que destacan, por ejemplo, los sensores utilizados, la longitud temporal de las imágenes o los grados de giro en el *data augmentation*. Surge por tanto la necesidad de dotar al entorno de preprocesado de cierta versatilidad y dar rienda suelta a la "magia" del investigador. Para ello, el entorno posee un fichero de configuración JSON que permite modificar la inmensa mayoría de atributos relevantes. A continuación, se detallan las propiedades modificables de cada módulo.

ATRIBUTO	TIPO	DESCRIPCIÓN
enabled	boolean	Activa el primer módulo
subject.list	Array<String>	Lista de los sujetos incluidos
movements.list	Array<String>	Lista de las actividades incluidas
movements.samples	Array<String>	Lista de los intentos incluidos
orientationSensors.enabled	boolean	Activa el procesado de los sensores de orientación
orientationSensors.list	Array<String>	Lista con los sensores de orientación a procesar
positionSensors.enabled	boolean	Activa el procesado de sensores de posición
positionSensors.list	Array<String>	Lista con los sensores de posición a procesar

***Tabla 4**.Configuración del módulo "interleaved-dataframe"*

ATRIBUTO	TIPO	DESCRIPCIÓN
enabled	boolean	Activa el segundo módulo
orientationSensors.enabled	boolean	Activa el procesado de los sensores de orientación
positionSensors.enabled	boolean	Activa el procesado de los sensores de posición
images.window-size	int	Numero de instantes temporales por cada imagen
images.overlap	int	Instantes temporales superpuestos entre imágenes
deletePrevious	boolean	Elimina las tablas intercaladas generadas por el módulo anterior

***Tabla 5.** Configuración del módulo "image-builder"*

ATRIBUTO	TIPO	DESCRIPCIÓN
deepen-images	boolean	Activa el enriquecido de imágenes
table-images (BETA)	boolean	BETA
dataAugmentationRotation.gradeList	Array<Int>	Lista de grados de rotación para data augmentation
FFT.enabled	boolean	Activa el cálculo de la FFT 2D
FFT.combined	boolean	Activa la combinación de imágenes temporales y frecuenciales
FFT.saveWithoutFFT	boolean	Activa el guardado de imágenes exclusivamente temporales
deletePrevious	boolean	Elimina las imágenes generadas en el módulo anterior

***Tabla 6.** Configuración del módulo "image-enricher"*

ATRIBUTO	TIPO	DESCRIPCIÓN
movements.list	Array<String>	Lista de actividades para mover al volumen compartido
FFT-input	boolean	Selecciona el tipo de imágenes para trasladar al volumen compartido: frecuenciales o temporales.

***Tabla 7.** Configuración del módulo "final-dataset"*

```
{
    "in-dt":{
        "enabled": true,
        "subjects":{
            "list": []

        },
        "movements":{
            "list": [],
            "samples": []
        },
        "orientationSensors":{
            "enabled": true,
            "list":  []
        },
        "positionSensors":{
            "enabled": false,
            "list": []
        }
        },
    "image-builder":{
        "enabled": true,
        "orientationSensors":{
            "enabled": true
        },
        "positionSensors":{
            "enabled": false
        },
        "images":{
            "window-size": 5,
            "overlap": 2
        },
        "deletePrevious": true
    },
    "image-enricher":{
        "deepen-images":{
            "enabled": true
        },
        "table-images":{
            "enabled": false
        },
        "dataAugmentationRotation":{
            "gradeList":[]
        },
        "FFT":{
            "enabled": true,
            "combined": true,
            "saveWithoutFFT": false
        },
        "deletePrevious": false
    },
    "final-dataset":{
        "movements":{
            "list": []
        },
        "FFT-input": true
    }
}
```


***Figura 25**. JSON de configuración entorno de preprocesado*

3.2.6 Uso (Makefile)

De manera similar al despliegue del framework, el entorno de preprocesado incorpora un makefile que facilita su uso **una vez nos encontramos dentro del propio contenedor**. Para ello es imprescindible además que el JSON de configuración haya sido completado correctamente; a continuación, bastará con hacer uso de la utilidad make junto con cualquiera de los siguientes comandos.

```
Usage: make <command>
Commands:
  help:                                Show this help information
  build-interleaved-dataframes         Excute interleaved dataframe script.
  build-images                         Excute image builder script.
  enrich-images                        Excute image enricher script.
  build-final-dataset                  Excute final dataset script.
  build-all:                           Execute all preprocessing steps by order.

Usual order:
  1. build-interleaved-dataframes
  2. build-images
  3. enrich-images
  4. build-final-dataset
```

Figura 26. *Opciones make sobre el entorno de preprocesado*

Se distingue:

- **build-interleaved-dataframes:** generar las tablas intercaladas (primer módulo).
- **build-images:** generar las imágenes (segundo módulo).
- **enrich-images:** enriquecer las imágenes (tercer módulo).
- **build-final-dataset:** trasladar al volumen compartido el resultado final del pre-procesado (cuarto módulo).
- **build-all:** ejecutar secuencialmente los cuatro módulos como se describe en el apartado “uso habitual”.

En definitiva, para hacer uso del entorno de preprocesado y obtener el conjunto de datos listo para entrenar se deben seguir los siguientes pasos:

1. Adaptar el dataset original al formato de entrada del framework.
2. Configurar el JSON del entorno con los parámetros deseados.
3. Ejecutar secuencialmente las fases de transformación.

De forma adicional, en el **Anexo II** se incluye una guía de uso en inglés.

3.3 Entorno de entrenamiento

Este segundo entorno representa la utilidad más general del framework al habilitar el entrenamiento de redes neuronales siempre y cuando las muestras utilizadas sean ficheros *.csv* nombrados correctamente y así poder extraer la etiqueta o *tag* necesario a partir del nombre. Naturalmente, dicha especificación coincide con el formato de salida de los datos tras pasar por el entorno de preprocesado, pero podría ser utilizada para otros múltiples problemas; en el **Anexo** II, sección "**Can I still use this framework if…"** se detallan una serie de consejos con este último objeto.

En la práctica habitual de resolución del problema HAR el entorno de *train* tomará las muestras de entrenamiento alojadas en el volumen compartido, es decir, las **imágenes** generadas por el entorno de preprocesado y, en base a la configuración convenientemente ajustada, comenzará a entrenar sin necesidad de realizar ningún cambio.

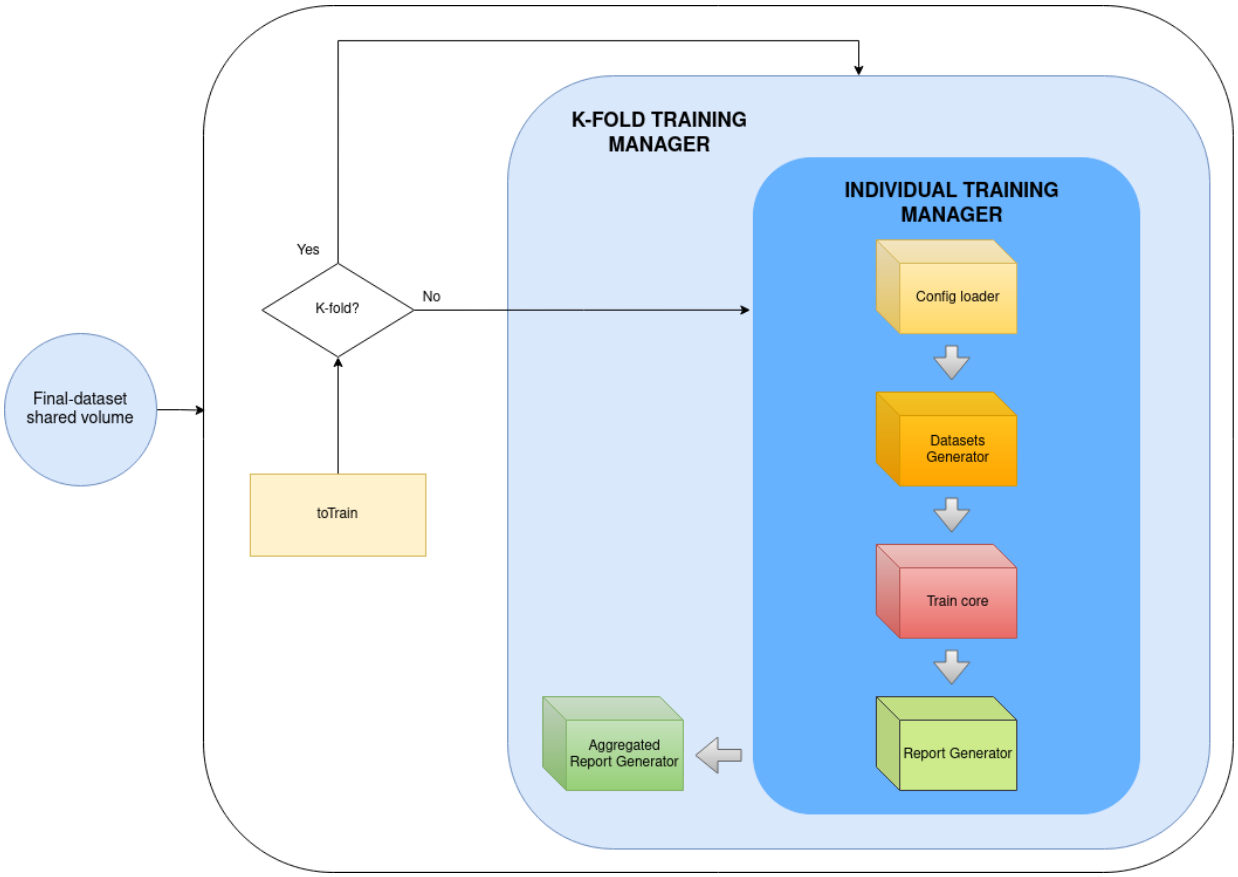


Figura 27. *Arquitectura del entorno de entrenamiento*

A diferencia del entorno de preprocesado, este marco no está dividido en módulos independientes; sin embargo, se pueden distinguir ciertos bloques lógicos que facilitan la comprensión del sistema. El uso de esta segunda herramienta es ciertamente flexible, aunque, en contraposición al preprocesado, no será sencillo incorporar aptitudes nuevas sobre las ya existentes; en este sentido se trata de un código robusto que requerirá amplios conocimientos de programación y específicamente de Tensorflow para poder elaborar nuevas funcionalidades.

3.3.1 Gestor de entrenamientos individuales

El gestor de entrenamientos individuales es la unidad lógica más básica de este contenedor y se ejecutará con cada entrenamiento. Por ello, es preciso comprender correctamente su comportamiento antes de avanzar con la estructura general.

Config loader

En primer lugar, por cada fichero de configuración albergado **directamente en el directorio *<u>/framework/toTrain</u>*** se generará un nuevo evento de entrenamiento independiente utilizando como dataset las muestras alojadas en el volumen compartido (las ya mencionadas imágenes de movimiento). El contenido de los ficheros de configuración de entrenamiento será objeto de análisis en el **apartado 3.3.3**; en esencia, a través de ellos es posible caracterizar todo lo necesario para realizar un determinado entrenamiento:

- Dimensión de las imágenes de entrada.
- Modelo de red neuronal deseada.
- Actividades/movimientos a clasificar.
- Sujetos que queremos incluir en cada conjunto de datos (entrenamiento, test y validación).
- Utilizar callbacks o no.
- Utilizar imágenes provenientes de *data augmentation.*
- ...

Dataset Generator

A continuación, se ejecutará el **generador de datos** que acomete principalmente dos funciones:

1. Dividir y discriminar las imágenes de entrenamiento, validación y test.
2. Generar al vuelo *batches* con estos subconjuntos que alimentarán a la red neuronal en cada fase.

Tradicionalmente se suele dividir el conjunto de datos disponible en tres subgrupos independientes: **entrenamiento, test y validación**. La red neuronal sólo podrá “aprender", es decir, actualizar el valor de los pesos de las neuronas, a partir de la información extraída de las imágenes de entrenamiento, mientras que los conjuntos de validación y test servirán exclusivamente para evaluar el rendimiento de la red neuronal intra-entrenamiento y post-entrenamiento, respectivamente.

Idealmente estos tres subgrupos deben tener la misma distribución y han de ser independientes entre sí. Por ello, para el caso particular del problema HAR, se ha considerado óptimo utilizar el sujeto que realiza la acción para discriminar qué imagen se destina a cada *set*.

Como se puede apreciar en la **Figura 28**, se habilita vía configuración la posibilidad de evitar el uso de datos provenientes de *data augmentation* para el conjunto de entrenamiento. Por su parte, los conjuntos de validación y test nunca usarán este tipo de datos puesto que las imágenes han sido generadas de manera **artificial** rotando ficticiamente los sujetos sobre el eje vertical (como se detalló en el **apartado 3.2.3**).

Finalmente, es posible excluir de los entrenamientos aquellos sujetos o actividades que no nos interesen a través del ya mencionado fichero de configuración de entrenamiento (**punto 3.3.3**).

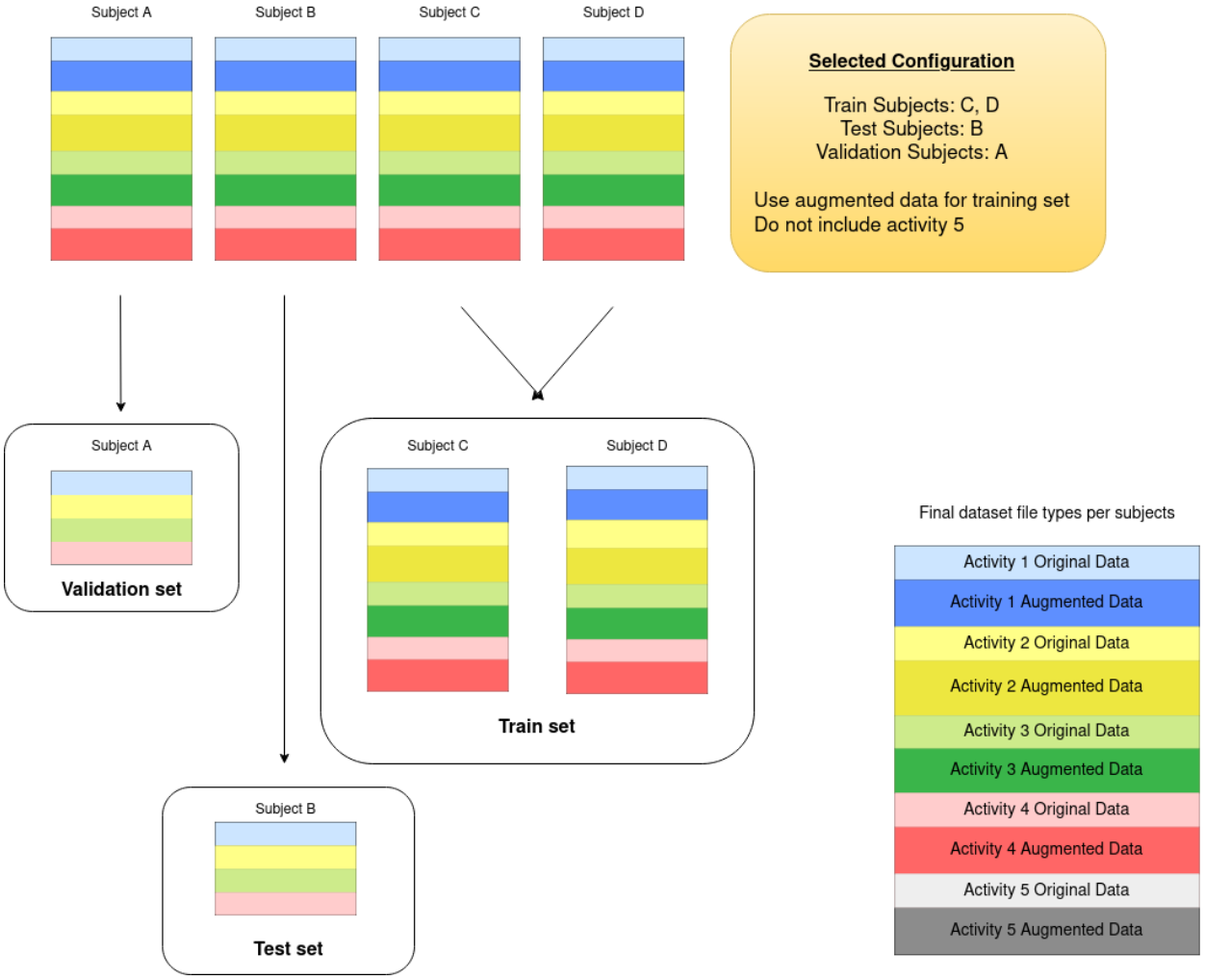


Figura 28. *División de subconjuntos en función del sujeto*

Tras haber determinado a qué *set* corresponde cada imagen debería comenzar la fase de aprendizaje; sin embargo, el gran tamaño que de ordinario ocupa en memoria RAM el conjunto de los datos de entrenamiento genera la necesidad de cargar las muestras en pequeños grupos denominados *batches*. Es deseable que a lo largo de cada época (*epoch*) se generarán tantos *batches* como sea necesario para suministrar a la red la totalidad de las muestras de entrenamiento. En un solo entrenamiento se sucederán tantas épocas como sea necesario hasta que la red sea capaz de clasificar correctamente.

Resulta así mismo imprescindible extraer la etiqueta que usará la red para identificar inequívocamente aquello que queremos aprender a clasificar; asiduamente se utiliza el nombre del fichero que contiene la muestra para determinar este valor, aunque se podría emplear cualquier otra característica.

Habitualmente tanto la generación de "lotes" como la extracción de etiquetas se encuentra integrada en Tensorflow o Keras; por ejemplo, para el caso de ficheros .png, es común utilizar la clase *image_datset_from_directory* (*Image Data Preprocessing*) que se encarga de generar los *batches* y de extraer la etiqueta correspondiente a cada muestra de forma automática.

No obstante, cuando los datos de entrada son ficheros *.csv* no existe ninguna utilidad previamente desarrollada que cubra esta necesidad. Por ello, ha sido preciso diseñar una canalización de datos de entrada en Tensorflow siguiendo la guía correspondiente (*Tf.Data: Compila Canalizaciones de Entrada de TensorFlow*). Esto ha permitido al autor del presente documento personalizar al máximo el flujo de datos de entrada de la red ideando dos canalizaciones diferenciadas:

- Canalización de imágenes 2D.
- Canalización de imágenes 3D.

La **canalización 2D** se encarga de tomar las imágenes de movimiento generadas por el entorno de preprocesado y extraer el nombre de la actividad a partir del fichero *.csv* que ha sido nombrado convenientemente como se detalló en el **punto 3.2.2**. Por ejemplo, para una imagen nombrada **S01-Walk -Orientationjoints-1-1-0.csv**, la etiqueta resultante sería "Walk". A continuación, se generan los lotes de imágenes que son entregados a la red.

Por su parte la **canalización 3D**, además de realizar las dos acciones anteriormente descritas, modifica ligeramente el formato de los datos de entrada. A lo largo del discurso lógico del **Capítulo 4** se plantea la posibilidad de mejorar los resultados obtenidos alterando el formato de las imágenes. Concretamente, se discute la ocasión de integrar cada una de las cuatro componentes que aporta un sensor de orientación en un canal o dimensión independiente. Para una imagen con sensores de orientación LFOOT, RFOOT y HIP el resultado sería el siguiente:

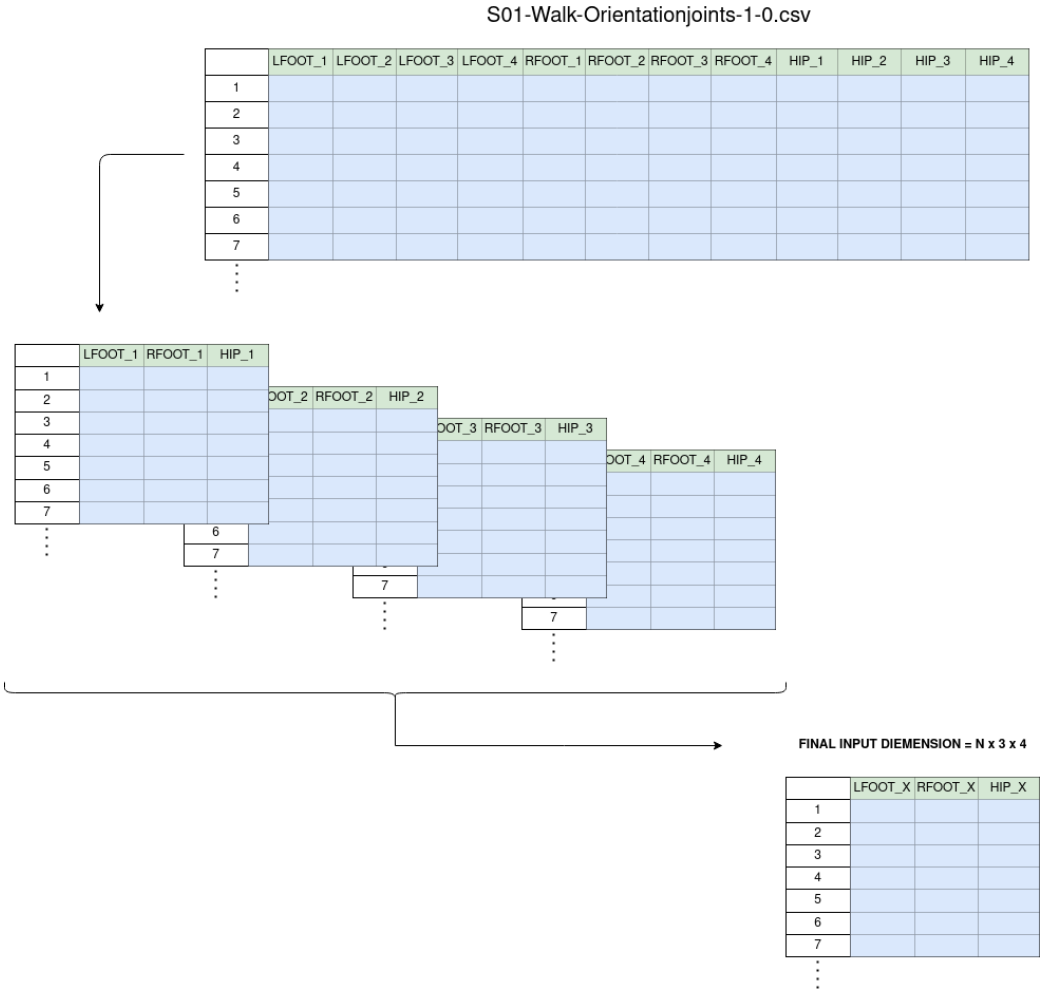


Figura 29. *Ilustración de canalización 3D*

Con ello se busca aplicar cierta similitud entre las componentes W, X, Y, Z de un cuaternión y las componentes RGB de una imagen. De manera que, si las redes neuronales convolucionales trabajan correctamente con estas últimas, quizás mejoren el resultado de nuestro problema; aunque esto será objeto de debate posterior, de momento nos conformaremos con habilitar esta posibilidad.

Core de entrenamiento

El núcleo de entrenamiento se encarga principalmente de cargar de manera dinámica el modelo de red neuronal deseado para cada ocasión, que será particular en cada fichero de configuración de entrenamiento.

A lo largo del diseño del entorno de preprocesado se ha hecho hincapié en la posibilidad de modificar la dimensión de las imágenes creadas; en otras palabras, alterar el número de sensores usados o cambiar el número de instantes temporales que contiene cada imagen, así como de aumentar o reducir el número de actividades que se quiere incluir en el dataset final de entrenamiento.

Por tanto y para que todo tenga sentido, es imprescindible que los diseños de redes neuronales sean capaces de adaptarse a las características individuales del dataset que se está utilizando en ese momento. De manera que, si decidimos modificar alguno de los parámetros anteriormente mencionados en el entorno de preprocesado, baste con transformar de igual modo la configuración de los futuros entrenamientos sin necesidad de alterar el código fuente de los diseños de redes las neuronales.

```python
import tensorflow as tf
import utils.utils as nnUtils
import numpy as np
from tensorflow.keras import layers
import math

def load_model(input_rows:int, input_columns:int, channels:int, movements_number:int):

    nnUtils.restrict_to_physcial_gpu()
    nnUtils.set_memory_growth()

    lstm_input_rows = math.ceil(input_rows/8)
    lstm_input_columns = math.ceil(input_columns/8)

    model = tf.keras.Sequential([
        layers.Conv2D(32, activation = "relu", input_shape = (input_rows,input_columns,channels), kernel_size=3,padding='same',strides=1),
        layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
        layers.BatchNormalization(axis=1),
        layers.Conv2D(64, activation='relu', kernel_size=4,padding='same',strides=1),
        layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
        layers.BatchNormalization(axis=1),
        layers.Conv2D(128, activation='relu', kernel_size=4,padding='same',strides=1),
        layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
        layers.BatchNormalization(axis=1),
        layers.Reshape((lstm_input_rows,lstm_input_columns*128)), #For each feauture(rows) squeeze all neurons
        layers.LSTM(units=256, return_sequences=False),
        layers.Dropout(0.5),
        layers.Flatten(),
        layers.Dense(516, activation = "relu"),                    #This activation function is the change
        layers.Dense(movements_number, activation="softmax")
    ])

    # model.summary()

    model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])

    return model
```

Figura 30. *Ejemplo de diseño de red neuronal dinámica*

Como se puede apreciar en la **Figura 30**, línea 7, el número de columnas, filas, canales y movimientos se fija de manera dinámica. Concretamente, para cada entrenamiento, la herramienta se encarga de tomar esta información del fichero de configuración (que deberá ser convenientemente ajustado) y cargar el modelo adaptado.

Así pues, el único requisito en general es mantener la congruencia entre ambos entornos, preprocesado y entrenamiento, para poder utilizar la herramienta satisfactoriamente.

Generador de reportes

Finalmente y tras cada entrenamiento se sucederán una **fase de test** y una **fase de predicción** que culminarán en un reporte facilitando el análisis de lo ocurrido. Cada informe contiene:

- **outcome.txt:** resumen del entrenamiento.
- **confusion-matrix.png:** matriz de confusión generada a partir de una **fase de predicción.**
- **confusion-matrix-normalized.png:** matriz de confusión normalizada.
- **confusion-matrix-metrics.png:** tabla de métricas de rendimiento calculadas a partir de la matriz de confusión.

```
################################################
#                 MODEL SUMMARY                #
################################################
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 352, 84, 64)       1088
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 176, 42, 64)       0
_________________________________________________________________
batch_normalization_16 (Batc (None, 176, 42, 64)       704
_________________________________________________________________
dropout_24 (Dropout)         (None, 176, 42, 64)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 176, 42, 64)       65600
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 88, 21, 64)        0
_________________________________________________________________
batch_normalization_17 (Batc (None, 88, 21, 64)        352
_________________________________________________________________
dropout_25 (Dropout)         (None, 88, 21, 64)        0
_________________________________________________________________
reshape_8 (Reshape)          (None, 88, 1344)          0
_________________________________________________________________
lstm_8 (LSTM)                (None, 512)               3803136
_________________________________________________________________
dropout_26 (Dropout)         (None, 512)               0
_________________________________________________________________
flatten_8 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 516)               264708
_________________________________________________________________
dense_17 (Dense)             (None, 7)                 3619
=================================================================
Total params: 4,139,207
Trainable params: 4,138,679
Non-trainable params: 528
_________________________________________________________________


################################################
#                TRAIN OUTCOME                 #
################################################
loss: 0.6072 | accuracy: 0.7370 | val_loss: 1.3192 | val_accuracy: 0.5504
loss: 0.3075 | accuracy: 0.8626 | val_loss: 0.2775 | val_accuracy: 0.8873
loss: 0.2476 | accuracy: 0.8924 | val_loss: 0.2067 | val_accuracy: 0.9226
loss: 0.1915 | accuracy: 0.9235 | val_loss: 0.3126 | val_accuracy: 0.8524
loss: 0.1671 | accuracy: 0.9351 | val_loss: 0.1801 | val_accuracy: 0.9333
loss: 0.1502 | accuracy: 0.9427 | val_loss: 0.3452 | val_accuracy: 0.8806
...
loss: 0.0110 | accuracy: 0.9954 | val_loss: 0.1984 | val_accuracy: 0.9534
loss: 0.0154 | accuracy: 0.9954 | val_loss: 0.0989 | val_accuracy: 0.9713
loss: 0.0198 | accuracy: 0.9922 | val_loss: 0.0867 | val_accuracy: 0.9697
loss: 0.0177 | accuracy: 0.9928 | val_loss: 0.0773 | val_accuracy: 0.9778


################################################
#                TEST OUTCOME                  #
################################################
loss: 0.1847 | accuracy: 0.9404

################################################
#                    NOTES                     #
################################################
K-Fold for neural network model CNN-LSTM-2 and augmented special data
```

***Figura 31**. Ejemplo del fichero de outcome.txt*

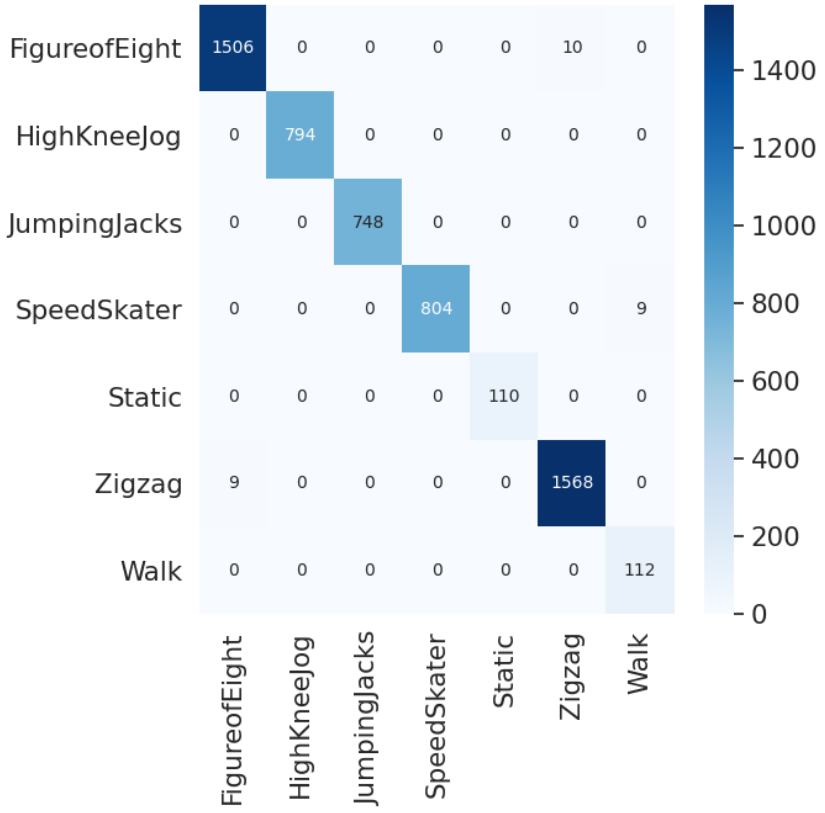


Figura 32. *Ejemplo de matriz de confusión*

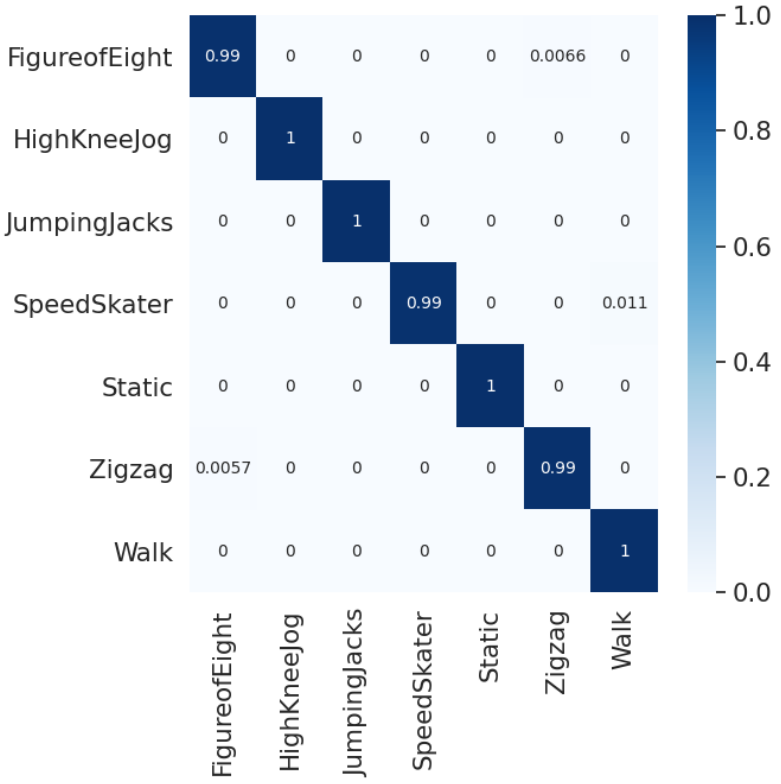


Figura 33*. Ejemplo de matriz de confusión normalizada*

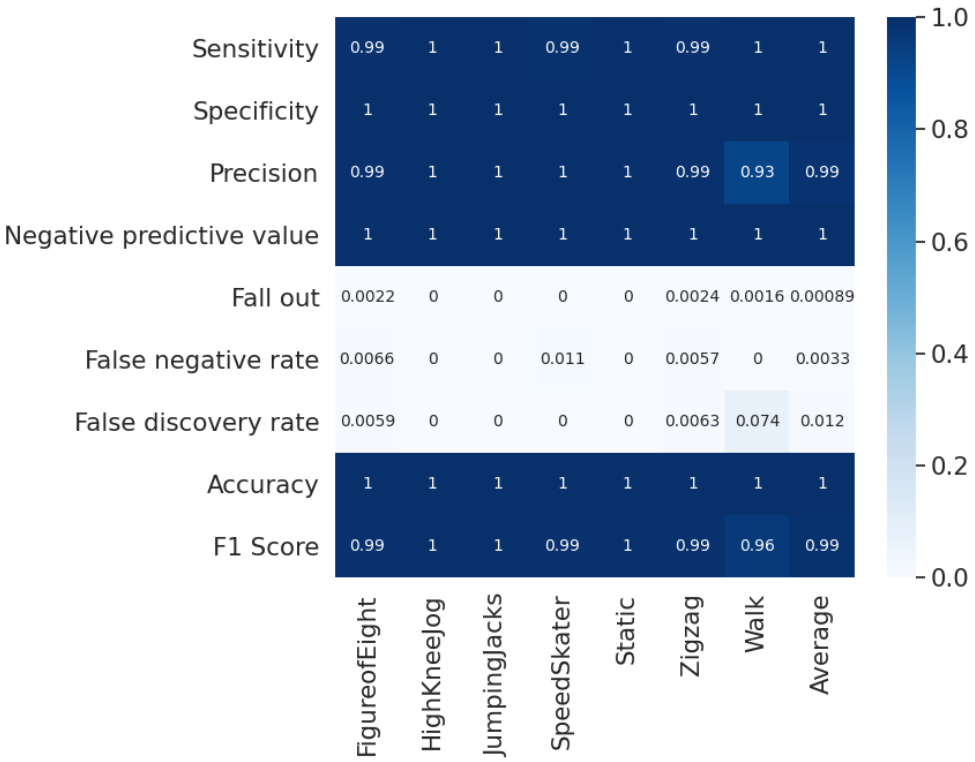


Figura 34. *Ejemplo de tabla de métricas de rendimiento*

También se incorporarán en cada reporte los ficheros **model.h5** y **model.json**, que servirán para cargar en el futuro el modelo de red neuronal ya entrenado, una copia del fichero de configuración de entrenamiento usado con el nombre **config.json**, y en ocasiones, otros ficheros como, por ejemplo, **los pesos de la red neuronal** -en caso de haber usado *model checkpoints* para guardar el estado de la red en un punto determinado- que recibirán el nombre de **best_weights.index**.

3.3.2 Gestor de K-folds

Como ya se ha dicho, se da la posibilidad de aplicar la técnica de *k-fold cross validation*. Para determinar los ficheros de configuración de entrenamiento que pertenecen a un mismo *k-fold* se debe generar una subcarpeta bajo el directorio ***/framework/toTrain*** y almacenarlos en ella. De tal modo, además de dispararse un evento de entrenamiento individual para cada uno de ellos, al concluir todos se generará un reporte agregado que contendrá esencialmente:

- **Matriz de confusión agregada.**
- **Matriz de confusión agregada normalizada.**
- **Tabla de métricas agregada.**

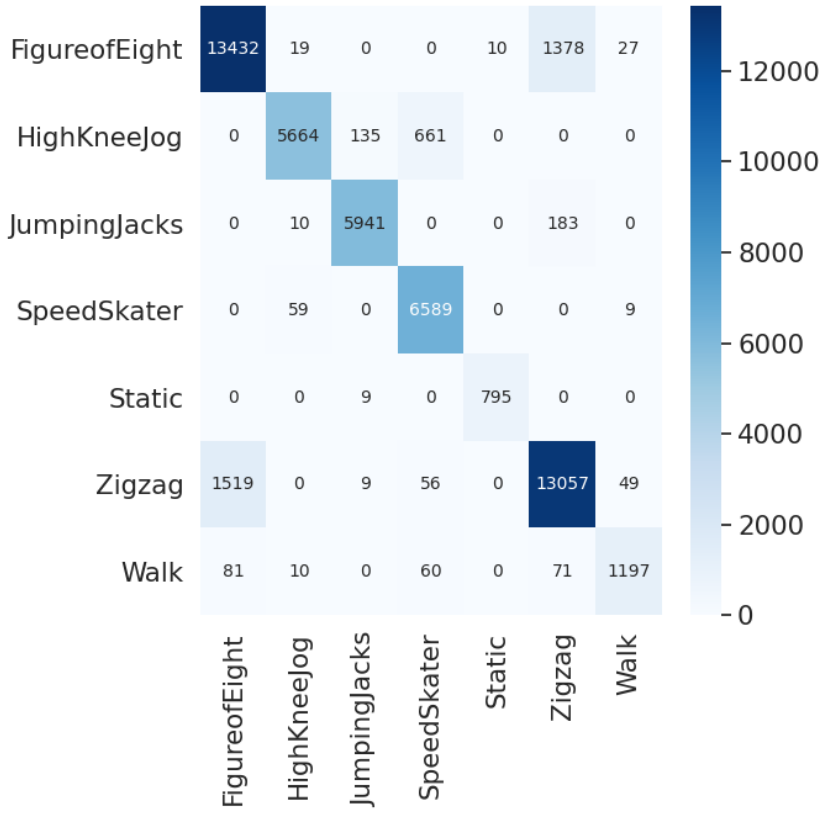


Figura 35. *Matriz de confusión agregada*

Se puede comprobar cómo la matriz de confusión agregada posee ocho veces más muestras que la matriz de confusión de un entrenamiento individual mostrada en la **Figura 32** al corresponder a un 8-Fold.

Finalmente cabe destacar que, entre los ficheros de configuración de un mismo K-fold, sólo pueden variar los conjuntos de test, validación y entrenamiento; en otras palabras, solo deben cambiar los sujetos que pertenecen a cada subconjunto. De otro modo, no se aplicaría correctamente la técnica de validación cruzada mediante K-fold.

3.3.3 Configuración de entrenamiento (JSON)

Cada entrenamiento está completamente caracterizado a través de su correspondiente fichero de configuración que posee los siguientes atributos:

ATRIBUTO	TIPO	DESCRIPCIÓN
neural-network	string	Modelo de red neuronal seleccionado
no-augmentation	boolean	Deshabilita las imágenes generadas vía *data-augmentation* para conjunto de entrenamiento
movements	Array<String>	Lista de movimientos que ha de predecir
train-subjects	Array<String>	Lista de sujetos incluidos en el conjunto de entrenamiento
validation-subjects	Array<String>	Lista de sujetos incluidos en el conjunto de validación
test-subjects	Array<String>	Lista de sujetos incluidos en el conjunto de test
callbacks.enabled	boolean	Habilita el uso de callbacks
callbacsk.list	Array<Object>*	Lista con los callbacks configurados
input-rows	int	Filas por imagen
input-columns	int	Columnas por imagen
channels	1 o 4	Determina el uso de la canalización 2D o 3D respectivamente.
batch-size	int	Tamaño del batch usado
train-steps	int	Steps por epoch en la fase de entrenamiento
validation-steps	int	Steps por epoch en la fase de validación
test-steps	int	Steps por epoch en la fase de test
epochs	int	Epochs para la fase de entrenamiento (uno por defecto para la fase de test).
comments	string	Comentarios para propagar en el reporte final

Tabla 8. *Configuración de un entrenamiento*

*Actualmente la plataforma soporta el uso de dos tipos de *callbacks* distintos: **modelCheckPoint** (*ModelCheckpoint*), que sirve para guardar el estado de la red neuronal en un momento determinado, y **earlyStopping** (*EarlyStopping*), utilizado para concluir prematuramente los entrenamientos si se da una determinada condición, típicamente que tras varias épocas la red neuronal no mejore**.**

En las **Figuras 36** y **37** se mostrará un ejemplo para la correcta configuración de cada uno de ellos.

```
{
    "neural-network": "N5",
    "no-augmentation": true,
    "movements": [
        "FigureofEight",
        "HighKneeJog",
        "Jog",
        "JumpingJacks",
        "SpeedSkater",
        "Static",
        "Zigzag",
        "Walk"
    ],
    "train-subjects": [
        "S01",
        "S02",
        "S06",
        "S07",
        "S08",
        "S09",
        "S10"
    ],
    "validation-subjects": [
        "S04"
    ],
    "test-subjects": [
        "S05"
    ],
    "callbacks": {
        "enabled": true,
        "list": [
            {
                "type": "modelCheckPoint",
                "save_weights_only":true,
                "monitor": "loss",
                "mode":"min",
                "save_best_only":true
            }
        ]
    },
    "input-rows": 350,
    "input-columns": 84,
    "channels": 1,
    "batch-size": 70,
    "train-steps": 81,
    "validation-steps": 81,
    "test-steps": 81,
    "epochs": 40,
    "comments": "K-Fold for neural network model 5. Try with a deeper network as I assume no much noise in measures."
}
```


Figura 36. *Primer ejemplo de configuración del entorno de entrenamiento*

Se presenta la configuración necesaria para entrenar el modelo de red neuronal N5. No se incluirá *data augmentation* en el conjunto de datos de entrenamiento. Mientras que los sujetos S04 y S05 serán usados para los conjuntos de validación y test respectivamente, el resto de ellos se incluirán en el de entrenamiento.

Además, se incluye el *callback* denominado *modelCheckpoint* que sirve para almacenar el estado de la red neuronal cuando ésta ofrezca el resultado con menor pérdida (*loss)*. Adicionalmente se caracteriza la dimensión de las imágenes e hiperparámetros y se completan los comentarios.

```
{
    "neural-network": "CNN+LSTM-4",
    "no-augmentation": true,
    "movements": [
        "FigureofEight",
        "HighKneeJog",
        "JumpingJacks",
        "SpeedSkater",
        "Static",
        "Zigzag",
        "Walk"
    ],
    "train-subjects": [
        "S01",
        "S02",
        "S04",
        "S05",
        "S06",
        "S07",
        "S08",
        "S10"
    ],
    "validation-subjects": [
        "S09"
    ],
    "test-subjects": [
        "S09"
    ],
    "callbacks": {
        "enabled": true,
        "list": [
            {
                "type": "modelCheckPoint",
                "save_weights_only":true,
                "monitor": "accuracy",
                "mode":"max",
                "save_best_only":true
            },
            {
                "type": "earlyStop",
                "monitor": "accuracy",
                "mode":"max",
                "patience":7
            }
        ]
    },
    "input-rows": 256,
    "input-columns": 84,
    "channels": 1,
    "batch-size": 80,
    "train-steps": 95,
    "validation-steps": 95,
    "test-steps": 95,
    "epochs": 80,
    "comments": "K-Fold for neural network model CNN+LSTM 4. No augmentation included. 8 subjects for train. 256 time-steps and FFT"
}
```


Figura 37. *Segundo ejemplo*

En este caso se presenta la configuración utilizada para entrenar el modelo CNN+LSTM-4. No se incluirá *data augmentation* en el conjunto de datos de entrenamiento. El sujeto S09 será usado para los conjuntos de validación y test, mientras que el resto se incluirán en el de entrenamiento.

Además, se incluye el *callback* denominado *modelCheckpoint* y *earlystop* -que sirve para finalizar el entrenamiento de manera prematura si la red no mejora su exactitud en 7 epochs-. Adicionalmente se caracteriza la dimensión de las imágenes e hiperparámetros y se completan los comentarios.

3.3.4 Uso (Makefile)

En esta ocasión también se dispone de un pequeño makefile que facilita la interacción con la herramienta; sin embargo, es mucho más sencillo ya que la complicación real de este entorno reside en el correcto diseño tanto de las redes que queramos utilizar como de los ficheros de configuración de entrenamientos.

```
Usage: make <command>
Commands:
  help:                         Show this help information
  train:                        Start training process
```

Figura 38. *Opciones make sobre el entorno de entrenamiento*

Se distingue en este caso únicamente el comando **train** que generará un evento de entrenamiento por cada fichero de configuración que haya bajo el directorio */framework/toTrain*; si, como hemos dicho, existe un subdirectorio con varios ficheros de entrenamiento entonces todos serán interpretados como un K-fold.

En definitiva, con el objetivo de usar correctamente el entorno de entrenamiento para la resolución del problema HAR el investigador debe acometer exclusivamente los siguientes pasos:

1. Preparar el dataset haciendo uso del entorno de preprocesado.
2. Diseñar el modelo de red neuronal que quiera utilizar (o hacer uso de uno ya desarrollado).
3. Generar un documento de configuración de entrenamiento con todos los atributos correctamente rellenados.
4. Ejecutar el comando make.

3.4 Entorno de inferencia

El entorno de inferencia tiene como función exponer un API REST capaz de realizar predicciones/clasificaciones de movimiento humano utilizando un modelo de red neuronal previamente entrenado.

En este caso se realizará una clasificación sobre imágenes de movimiento utilizadas a lo largo del presente documento y que deberán coincidir en formato con aquellas usadas para entrenar el modelo seleccionado. En este sentido, el número y tipo (3D o 4D) de sensores usados deberá ser exactamente el mismo, así como el número de *time-steps* por cada imagen. No obstante, en caso de haber decidido utilizar la *técnica de la FFT* en el entrenamiento, será el propio API REST quien se encargue de calcular esta operación, facilitando su uso en tiempo real.

El API REST está desarrollado sobre la tecnología Flask (*Welcome to Flask — Flask Documentation (2.0.X)*); se trata de un micro framework basado en Python que permite generar aplicaciones web con pocas líneas de código.

Flask no sólo incorpora la funcionalidad necesaria parar generar el servicio sino también agrega un pequeño servidor web para poder realizar el despliegue; no obstante, siguiendo las pautas recomendadas por los propios creadores, este servidor no está preparado para ser desplegado en un entorno de producción y su uso debe limitarse a marcos de desarrollo. Por tanto, el autor del presente documento recomienda alternativamente el uso de un servidor Nginx o un Apache para suplir las carencias del servidor web original en ámbitos comerciales.

3.5.1 Soporte

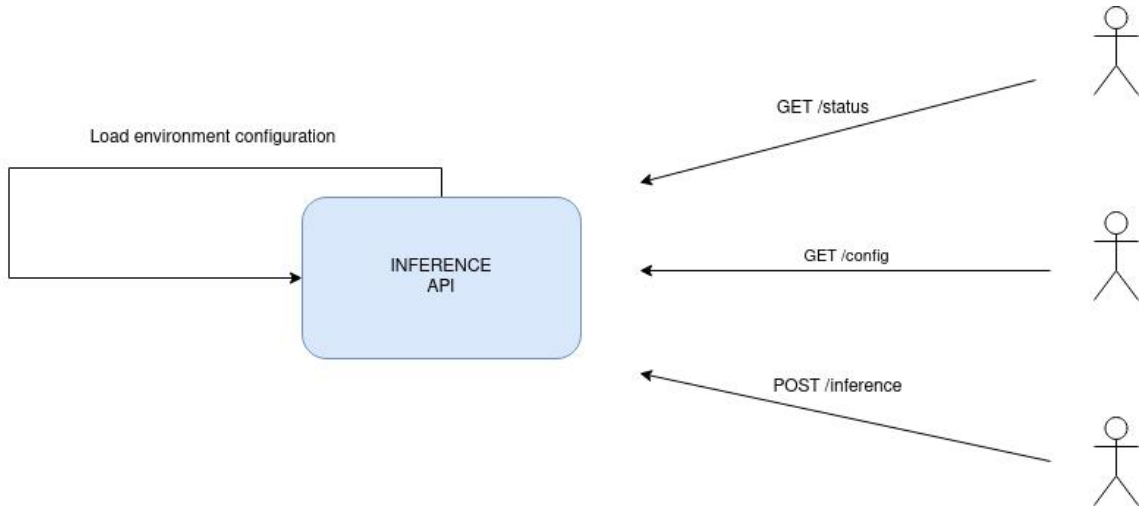


***Figura 39**. API REST de inferencia*

El servidor de inferencia expone los siguientes tres *endpoints*:

- **api/status**: consultar el estado del servidor.
- **api/config**: consultar la configuración actual del servidor, para conocer el modelo de red neuronal utilizada o el tamaño requerido de la imagen de movimiento.
- **api/inference**: realizar una petición de clasificación sobre una determinada imagen.

A continuación, se muestra un ejemplo para cada una de estas operaciones:

```
# Get service status
curl localhost:8082/api/status
# Reply: {"code":"SUCCESS","message":"Server is online"}

# Get service config
curl localhost:8082/api/config
# Reply: {
   "in-fo": {
      "FFT": true,
      "channels": 1,
      "columns": 28,
      "movementsList": [
         "FigureofEight",
         "HighKneeJog",
         "Jog",
         "JumpingJacks",
         "SpeedSkater",
         "Static",
         "Zigzag",
         "Walk"
      ],
      "rows": 250,
      "sensorslist": [
         "qRPV",
         "qRTH",
         "qRSK",
         "qRFT",
         "qLTH",
         "qLSK",
         "qLFT"
      ]
   },
   "nueral-network": "N2-350-28-9-1"
}

# Request inference
curl --location --request POST 'localhost:8082/api/inference' --form 'data_file=@"SOMEWHERE/S10-Zigzag-Orientationjoints-2-103.csv-0"'
# Reply: {"code":"SUCCESS","message":"The performed movement is: Zigzag"}
```

Figura 40. *Ejemplo de operaciones soportadas sobre el API de inferencia*

A diferencia de los endpoints de configuración y estado que esperan una interacción a través del método GET, el *endpoint* de inferencia está preparado para una comunicación vía POST. Ello se debe a la necesidad de enviar la imagen sobre la que queremos realizar la clasificación; ésta deberá ser suministrada en el cuerpo de la petición con el nombre de atributo **data-file** y en congruencia con el formato utilizado en el entorno de preprocesado y entrenamiento.

Asimismo, el API de inferencia contiene numerosos mensajes de error que alertarán al usuario de la mala praxis e impedirán el uso malintencionado de la herramienta. Además, el API *per se* es fácilmente escalable y resultará casi inmediato incorporar un nuevo recurso o funcionalidad, si fuese necesario.

3.5.2 Configuración (JSON)

De manera similar a los dos anteriores, el API de inferencia es configurable a través de un fichero JSON que soporta los siguientes parámetros:

ATRIBUTO	TIPO	DESCRIPCIÓN
neural-network	String	Red neuronal seleccionada
info.movementList	Array<String>	Lista de actividades soportadas
info.sensorsList	Array<String>	Lista de sensores usados
Rows	Int	Filas por imagen
Columns	Int	Columnas por imagen
Channels	Int	Canales por imagen
FFT	Boolean	Valor que indica el uso de la técnica de la FFT.

Tabla 9. *Parámetros de configuración del API de inferencia*

```
{
   "LOCAL": {
      "nueral-network": "N5-250-28-9-1",
      "info" : {
         "movementsList": [
            "FigureofEight",
            "HighKneeJog",
            "Jog",
            "JumpingJacks",
            "SpeedSkater",
            "Static",
            "Zigzag",
            "Walk"
         ],
         "sensorslist": ["qRPV", "qRTH", "qRSK", "qRFT", "qLTH", "qLSK", "qLFT"],
         "rows": 250,
         "columns": 28,
         "channels": 1,
         "FFT": true
      }
   },
   "PRO": {
      "nueral-network": "N2-PROD",
      "info" : {
         "movementsList": [
            "FigureofEight",
            "HighKneeJog",
            "Jog",
            "JumpingJacks",
            "SpeedSkater",
            "Static",
            "Zigzag",
            "Walk"
         ],
         "sensorslist": ["qRPV", "qRTH", "qRSK", "qRFT", "qLTH", "qLSK", "qLFT"],
         "rows": 350,
         "columns": 28,
         "channels": 1,
         "FFT": true
      }
   }
}
```

Figura 41. *JSON de configuración entorno de inferencia*

Como se aprecia en la figura anterior, al tratarse de un API pensado para ser expuesto públicamente, el contendor soporta varias configuraciones en función del ámbito donde se despliegue; véase LOCAL (desarrollo) o PRO (producción).

Dada la siguiente estructura de ficheros dentro del contenedor de inferencia, el investigador deberá colocar la red neuronal entrenada dentro una nueva carpeta bajo el directorio “nerunalNetworks”.

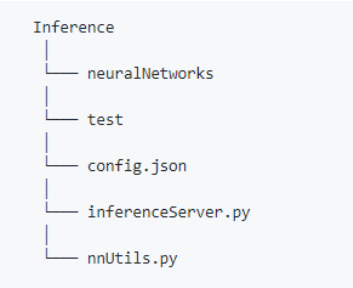


Figura 42. *Estructura de directorios entorno de inferencia*

El nombre de dicha carpeta será el usado en el atributo de configuración **neural-network** y deberá contener:

- Un fichero JSON con el modelo de la red neuronal utilizada con el nombre “**model.json**”.
- Un fichero con el valor de los pesos del modelo nombrado como “**model.h5**”.

A pesar de la complicación que esto parece suponer, la realidad es que el entorno de entrenamiento genera automáticamente ambos ficheros por cada entrenamiento realizado, como se detalla en la sección **Generador de reportes** del **apartado 3.3.1**. Así que el proceso se reduce a:

- Generar una subcarpeta con el nombre deseado, por ejemplo “**Example”**.
- Copiar los ficheros producidos tras el entrenamiento de la red neuronal.
- Modificar el fichero de configuración con el valor **“neural-network”: “Example”**.

Capítulo 4. Integración: pruebas y resultados

En este punto se exponen los resultados obtenidos tras resolver el problema HAR sobre las bases de datos citadas en el **Capítulo 2** (con sus consideraciones):

- ***Harvard Neura Sparse Dataset (HNS)***
- ***Realdisp activity recognition dataset (RAR)***

Para la primera, se ha realizado un estudio completo del problema HAR aplicando el esquema de resolución expuesto en el **Capítulo 2** y optimizando los recursos necesarios mediante el uso de la herramienta de preprocesado y entrenamiento desglosada en el **Capítulo 3**. Por su parte, para la segunda base de datos se realiza un estudio comparativo entre los resultados obtenidos (Bombín, 2020) y aquellos alcanzados al replicar el mismo estudio haciendo uso del *framework* desarrollado.

La notación utilizada para caracterizar los modelos de redes neuronales es la siguiente (Ordóñez & Roggen, 2016):

- C(F): capa convolucional con F unidades.
- R(F): capa recurrente de tipo LSTM.
- RG(F): capara recurrente de tipo GRU con F unidades.
- DR(d): capa de *dropout* con probabilidad d.
- D(n): capa totalmente conectada con activación no lineal.
- S_m: última capa densa de una red convolucional.

4.1 *Harvard Neura Sparse Dataset*

Para comenzar ha sido necesario diseñar varios modelos de redes neuronales sencillos en orden a obtener las primeras conclusiones. En primer lugar, se replica la red neuronal convolucional que mejor resultado había proporcionado sobre la BD RAR (Bombín, 2020) y que será denominada **N2** a partir de ahora.

```
model = tf.keras.Sequential([
    layers.Conv2D(16, activation = "relu", input_shape = (input_rows,input_columns,channels), kernel_size=3,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(32, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(64, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(128, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(movements_number, activation = "softmax")
])

# model.summary()

model.compile(loss = "sparse_categorical_crossentropy", optimizer = Adam(lr= 9e-4) , metrics = ["accuracy"])
```

Figura 43. *Red convolucional N2*

El modelo seleccionado con esquema **C(16)-C(32)-C(64)-C(128)-DR(0.5)-S$_m$** incorpora, además, una capa de **max pooling** seguida de una de ***batch normalization*** entre cada una de las capas convolucionales con el objetivo de regularizar los datos de salida de cada una de ellas y así evitar el *overfitting*. El optimizador utilizado es Adam con un ***learning rate*** inicial de 9 x 10^{-4}.

Como se ha mencionado en el **punto 2.3** la BD RAR contiene muestras grabadas a una frecuencia de 50Hz, mientras que la BD HNS se grabó a 100Hz. En la resolución del problema se hace uso de la técnica de segmentación; ello implica directamente que una ventana con el mismo número de muestras aplicada sobre las dos bases de datos generará imágenes/tablas con intervalos de distinta longitud temporal absoluta. Así, una ventana de 100 muestras representará 1 segundo de actividad realizada en la BD HNS, mientras que para la BD RAR supondrá 2 segundos de actividad, es decir, mucha más información en este segundo caso.

Por ende, tiene sentido que el tamaño de ventana óptimo crezca para el estudio ***Harvard Neura Sparse Dataset*** -con el fin de contener la misma cantidad de información-. En consecuencia, también deberá ser dimensionado proporcionalmente el modelo de red neuronal convolucional utilizado, al tener las imágenes generadas una "resolución" mayor (tablas de datos más grandes o más instantes *time-steps* por tabla). Así, el autor de este TFG propone el siguiente modelo:

```
model = tf.keras.Sequential([
    layers.Conv2D(32, activation = "relu", input_shape = (input_rows,input_columns,channels), kernel_size=3,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(64, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(128, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(256, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Dropout(0.5),
    layers.Flatten(),
    layers.Dense(movements_number, activation = "softmax")
])

# model.summary()

model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
```

Figura 44. *Red convolucional N5*

El modelo **N5** con esquema **C(32)-C(64)-C(128)-C(256)-DR(0.5)-S$_m$** representa una evolución del modelo N2 adaptado a las características propias del problema, aumentando únicamente el número de unidades por cada capa convolucional y eliminando el valor de partida de la tasa de aprendizaje adaptativo Adam.

Además, se propone el uso de modelos combinados CNN+LSTM planteado en numerosos artículos científicos (Mutegeki & Han, 2020). Mientras que las redes convolucionales son capaces de extraer las características **permanentes en el tiempo** de un determinado movimiento humano, los diseños recurrentes y en particular aquellos *long-short-term-memory* habilitan la posibilidad de explotar las dependencias temporales (características **dinámicas**) de dicho movimiento. Aunque el uso de modelos combinados no siempre ofrece los mejores resultados, se presenta en este caso la oportunidad perfecta para aprovechar conjuntamente las ventajas que nos ofrecen ambas metodologías.

Después de varios diseños preliminares la opción elegida es **C(64)-DR(0.2)-C(64)-DR(0.2)-R(256)-DR(0.2)-D(516)-S_m** intercalando de nuevo capas de **max pooling** y ***batch normalization*** entre aquellas convolucionales.

```
model = tf.keras.Sequential([
    layers.Conv2D(64, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Dropout(0.2),
    layers.Conv2D(64, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Dropout(0.2),
    layers.Reshape((lstm_input_rows,lstm_input_columns*64)), #For each feauture(rows) squeeze all neurons
    layers.LSTM(units=256, return_sequences=False),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(516, activation = "sigmoid"),
    layers.Dense(movements_number, activation="softmax")
])

# model.summary()

model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
```

Figura 45. *Red convolucional CNN+LSTM-4*

4.1.1 Optimización de los datos de entrenamiento

La constitución de estos modelos de referencia permite avanzar hacia la siguiente fase de la investigación cuyo objetivo es discriminar, siguiendo un método heurístico -ensayo y error-, los parámetros y operaciones que debe aplicar el entorno de preprocesado en los procesos de segmentación y extracción de características para obtener los conjuntos de datos mejor preparados de cara al entrenamiento.

En la **Tabla 10** se exponen los mejores resultados obtenidos tras entrenar distintas adaptaciones del dataset original -mediante el entorno de preprocesado- sobre las redes anteriormente descritas. En los casos representados el ajuste de hiperparámetros se ha realizado previamente y se presume muy bueno u óptimo.

3		Window size	FFT	Subject rotation	N5	CNN SERGIO (N2)	CNN +LSTM 4
4	Neuro Sparse	128	Si	No	0,838	0,83	
5		128	Si	Si	0,78		
6		256	SI	No	90.4		89,57
7		256	SI	Si	84.28		
8		256	SI	Partial*	92.14		91,14
9		352	SI	No	92,8	91,57	92,28
10		352	SI	Si	87		
11		352	Si	Partial*	92,28	91,14	94,28
12		352	No	No	83,86	85	87,85

Tabla 10. *Optimización heurística de las operaciones en el preprocesado*

En primer lugar, se busca **determinar el tamaño óptimo de ventana** a utilizar (time-steps por imagen), resultando que, conforme aumenta el tamaño (sección inferior de la **Tabla 10**) el éxito es mayor. La conclusión es meridianamente intuitiva: cuanta más información incorporamos en cada imagen, más fácil les resulta a las redes neuronales clasificar el movimiento humano. A pesar de ello, el objetivo ideal es determinar la actividad realizada utilizando el mínimo intervalo de tiempo posible, lo que supone un *trade-off* (cuello de botella) insalvable.

En segundo lugar, **se valora el uso de rotación vertical como forma de aumentar el conjunto de datos** disponibles y aportar invarianza espacial a la red neuronal. A continuación, se muestran los resultados de un 8-Fold (validación cruzada de orden 8) utilizando el modelo N5 e imágenes de 352 muestras: en el caso de la **Figura 47** se ha utilizado la técnica de aumento de datos mediante rotación con los ángulos [0,15,30,45,60,75,90.105.120,135,150,165,180], mientras que en el caso de la **Figura 46** solo se han usado imágenes originales en la fase de entrenamiento.

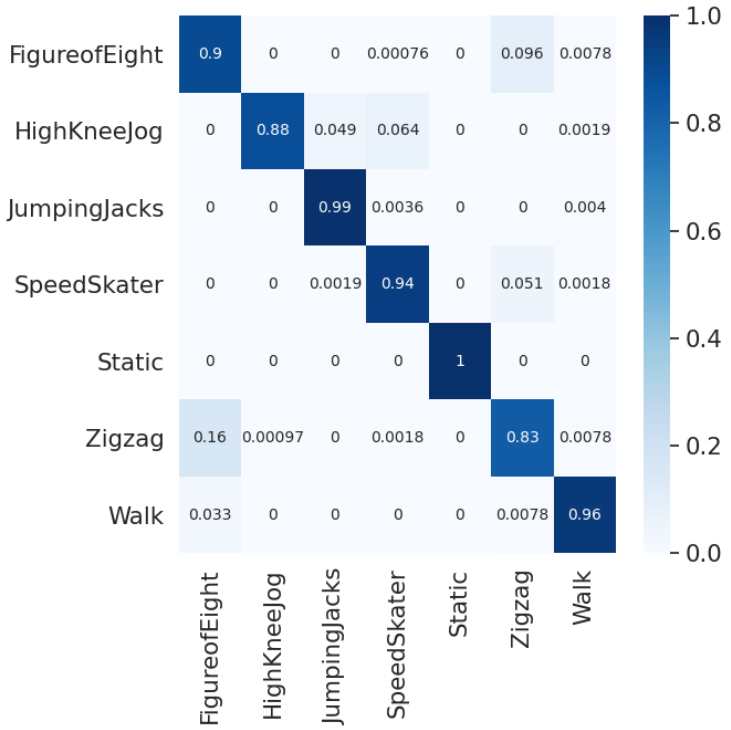


Figura 46. *Matriz de confusión sin aumento de datos*[2]

Como se aprecia en la diagonal de la **Figura 46** la precisión es prácticamente homogénea para todas las actividades, es decir, el modelo es capaz de generalizar por igual todas las actividades. Aunque existe cierta confusión entre los movimientos "FigureOfEight" y "ZigZag" -razonable al ser actividades similares en intervalos cortos de tiempo-, no es tan notoria como la dada al usar aumento de datos (**Figura 47**).

[2] Las matrices de confusión expuestas en este TFG se construyen siempre a partir de los datos obtenidos tras evaluar la red neuronal contra el conjunto de test; es decir, la red neuronal nunca tiene conocimiento previo de los datos.

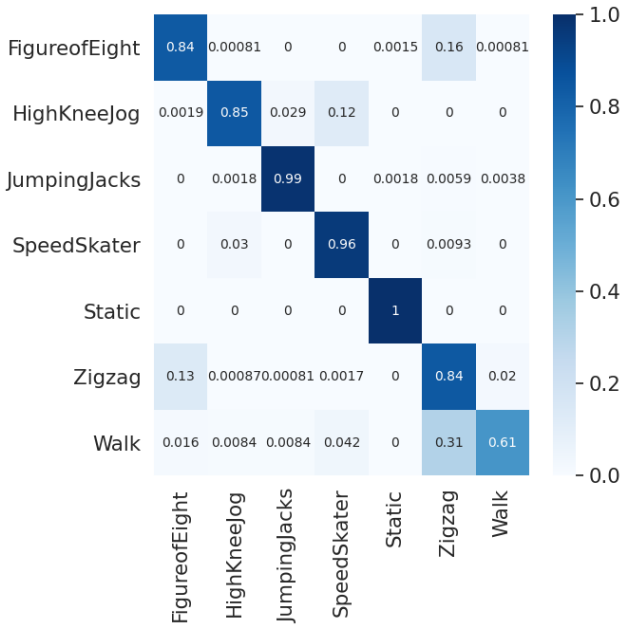


Figura 47. *Matriz de confusión con aumento de datos*

El uso de datos generados inorgánicamente perjudica, en esencia, el aprendizaje de la actividad "caminar". La hipótesis barajada es la siguiente: **al ser "caminar" una actividad tan básica, la rotación ficticia de sujetos le otorga *features* (características) adicionales que, por el hecho de ser un movimiento simple, eclipsan a sus peculiaridades inherentes; por contra, en acciones más complejas dotadas de propiedades diferenciales este fenómeno no se da**. El aumento de datos favorece además la confusión entre los movimientos "dibujar figurar de ocho" y "zigzaguear", como ya hemos comentado.

Por tanto y tras multitud de ensayos, la decisión óptima ha sido aplicar data augmentation **parcial** (*partial,* en la **Tabla 10**, **líneas 8 y 11**) - aumentar todas las actividades menos Walk, FigureofEight y ZigZag- puesto que, a pesar de no favorecer a los modelos convolucionales puros, mejora el resultado del modelo combinado CNN+LSTM-4, que ofrece la mayor precisión hasta el momento de 94,28%.

Figura 48. *Uso de data augmentation parcial*

Para concluir, en la **Tabla 10, línea 12** se analiza el impacto que tiene el uso de imágenes que combinan datos temporales y frecuenciales calculados aplicando la FFT 2D sobre los primeros (**Capítulos 2** y **3**). En contraposición a las conclusiones obtenidas en ***Feature extraction for robust physical activity recognition*** (Jiadong Zhu, Rubén San-Segundo y José M. Pardo, 2017), donde el uso de información frecuencial empeora el resultado obtenido usando algoritmos de ML en la clasificación, en este caso, al usar redes neuronales profundas, la técnica sí tiene un efecto determinante.

Por tanto, se concluye que para las imágenes de entrenamiento:

- El número óptimo de **time-steps** por imagen será de **352 para actividades medidas a 100Hz.**
- El **uso** de contenido frecuencial a través de la **FFT** ayuda en gran medida en el proceso de clasificación, por lo que se **incluirá** siempre en el preprocesado.
- El **aumento de datos vía rotación vertica**l se realizará de forma **parcial** en función de la actividad, puesto que no siempre favorece el proceso de aprendizaje.

4.1.2 Diseño de modelos avanzados

Una vez identificadas las características del dataset que finalmente se utilizará en el entrenamiento, se proponen diseños alternativos de redes neuronales buscando mejorar los resultados obtenidos.

CNN + GRU

Una alternativa al modelo combinado CNN+LSTM -teniendo en cuenta que "GRU is similar to LSTM and has shown that it performs better on smaller datasets" (Patel, 2018)- es **C(64)-C(64)-C(64)-DR(0.5)-GRU(128)-S_m**. La celda GRU al no tener puerta de salida, es decir, una puerta menos que la celda LSTM, realiza un número menor de operaciones internas por lo que se considera más sencilla; ello hace posible entrenar los modelos de forma más rápida y con conjuntos de datos de entrenamiento más pequeños.

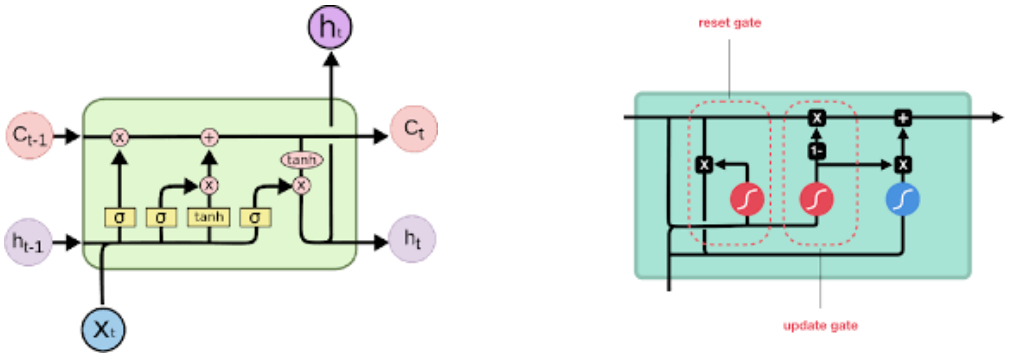


Figura 49. *Celda LSTM vs celda GRU*

LSTM

A pesar de ser un modelo sencillo, y aunque resulta evidente que un modelo LSTM puro debe comportarse peor que un modelo CNN o uno CNN+LSTM, se propone el esquema **R(256)-DR(0.5)-R(128)-S_m** para corroborar esta última hipótesis.

CNN avanzadas

Los modelos convencionales de clasificación basados en arquitecturas convolucionales se construyen sucediendo una serie de capas ***conv1D*** o ***conv2D*** seguidos de una capa **densa** final. La función de las capas convolucionales es extraer características cada vez más particulares de las matrices de datos de entrada, mientras que la capa densa final debe realizar el proceso de clasificación utilizando las características obtenidas en las capas previas.

Con el objetivo de mejorar el rendimiento de los modelos en este último paso de clasificación se incluye tradicionalmente una segunda capa densa totalmente conectada que actuará como regularizador de los datos previniendo el overfitting. Así, se propone en este TFG una evolución del modelo N5 denominado **N5-2Dense** con formato **C(32)-C(64)-C(128)-C(256)-DR(0.5)-D(516)-S_m** que añade una capa densa con 516 unidades.

De manera alternativa, en el artículo "**Network In Network**" (Lin et al., 2014) se propone sustituir esta nueva capa de tipo **D(n)** por una capa *global average pooling* que, en teoría, ofrece mejores resultados. Siguiendo las recomendaciones del citado paper se realizaron múltiples pruebas hasta dar con un modelo relativamente satisfactorio, **C(128)-C(128)-C(256)-C(256)-AveragePooling2D( )-D(516)-S_m** que denominaremos **N6**, donde la capa **D(n)** ha sido reemplazada.

```
model = tf.keras.Sequential([
    layers.Conv2D(128, activation = "relu", input_shape = (input_rows,input_columns,channels), kernel_size=3,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(128, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(256, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.BatchNormalization(axis=1),
    layers.Conv2D(256, activation='relu', kernel_size=4,padding='same',strides=1),
    layers.BatchNormalization(axis=1),
    layers.AveragePooling2D(pool_size=(44, 11), padding='same'),
    layers.Flatten(),
    layers.Dense(movements_number, activation = "softmax")
])

# model.summary()

model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
```

Figura 50. *Red convolucional N6*

CNN multicanal. Canalización 3D

Las imágenes RGB (rojo, verde o azul) son, en esencia, matrices 3D de dimensión MxNx3 donde **MxN** corresponde con **la resolución** en **píxeles** de la imagen y la componente final identifica el **porcentaje** de **cada color** que posee ese determinado píxel. El color que adquiere un píxel en la representación surgirá como la fusión de los tres valores de dicha componente, final como se muestra en la **Figura 50.**

Las redes convolucionales CNN basadas en Keras están preparadas para trabajar con matrices bidimensionales y tridimensionales de manera que, cuando lo que se quiere clasificar es una imagen RGB, basta con adaptar el atributo *input_shape* al formato anteriormente citado (MxNx3).

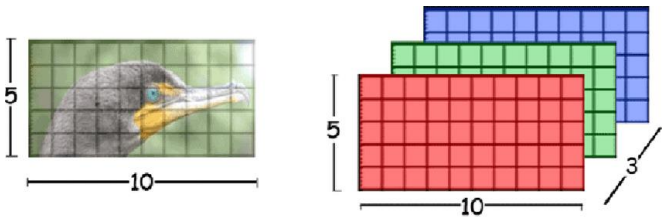


***Figura 51**. Construcción de una imagen RGB*

Análogamente, propone este TFG en el uso de sensores de orientación suministrar cada una de las componentes que conforma un cuaternión (W, X, Y, Z) a través de un canal específico (**Figura 29**).

- 5x10 en la **Figura 51**→ *time-steps x número de señores*.
- RGB en la **Figura 51** (3 componentes) → WXYZ en la imagen de movimiento (4 componentes ya que se restringe la propuesta a sensores de orientación).

Para ello se hará uso en este caso de estudio del **modelo N5** junto con la **canalización 3D**, tal y como se detalló en el **punto 3.3.1**, siendo necesario únicamente adaptar la configuración de los entrenamientos individuales como se muestra en la **Figura 52**:

***Figura 52.** Adaptación al uso de canalización 3D*

ConvLSTM2D

Finalmente, siguiendo a Shah et al., 2021 se incluye el uso de modelos basados en celdas convlstm2D cuya función es idéntica al modelo combinado CNN+LSTM: utilizar la sección convolucional para identificar las características **remanentes** y aprovechar la habilidad LSTM para reconocer las dependencias **a corto plazo** de la actividad.

```
model = tf.keras.Sequential([
    layers.Reshape((input_rows,) + input_shape, input_shape=input_shape),
    layers.ConvLSTM2D(filters=8, kernel_size=(3, 3),
                data_format= 'channels_last',
                activation='relu',
                return_sequences=True,
                padding='same'),
    layers.BatchNormalization(),
    layers.ConvLSTM2D(32, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.5),
    layers.Flatten(),
    layers.Dense(movements_number, activation = "softmax")
])

# model.summary()

model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
```

***Figura 53**. Modelo basado en capas convlstm2d*

4.1.3 Aplicación y análisis de los resultados

A continuación, se ha realizado un entrenamiento 8-Fold -validación cruzada de orden 8- con cada uno de los modelos expuestos hasta el momento (N2, N5, N5-Multicanal, N6, LSTM, CNN+LSTM-4, CNN+GRU, CON2DLSTM-1) tomando el conjunto de datos de entrenamiento generado a partir de las siguientes conclusiones obtenidas en los **apartados 2.3 y 4.1.1:**

- Ventanas temporales de 352 instantes (**4.1.1**) con un 75% de superposición entre imágenes (264 time-steps compartidos).
- Concatenación horizontal de información temporal y frecuencial a partir de la FFT 2D. (**4.1.1**)
- Uso de data augmentation parcial.
    - HighKneeJog, JumpingJacks, Static, SpeedSkater. (**4.1.1**)
    - Walk, FigureOfEight y ZigZag no serán aumentados. (**4.1.1**)
- Exclusión de los movimientos TUG y 5-Minute-Walk (**2.3**)

El resultado obtenido es el siguiente:

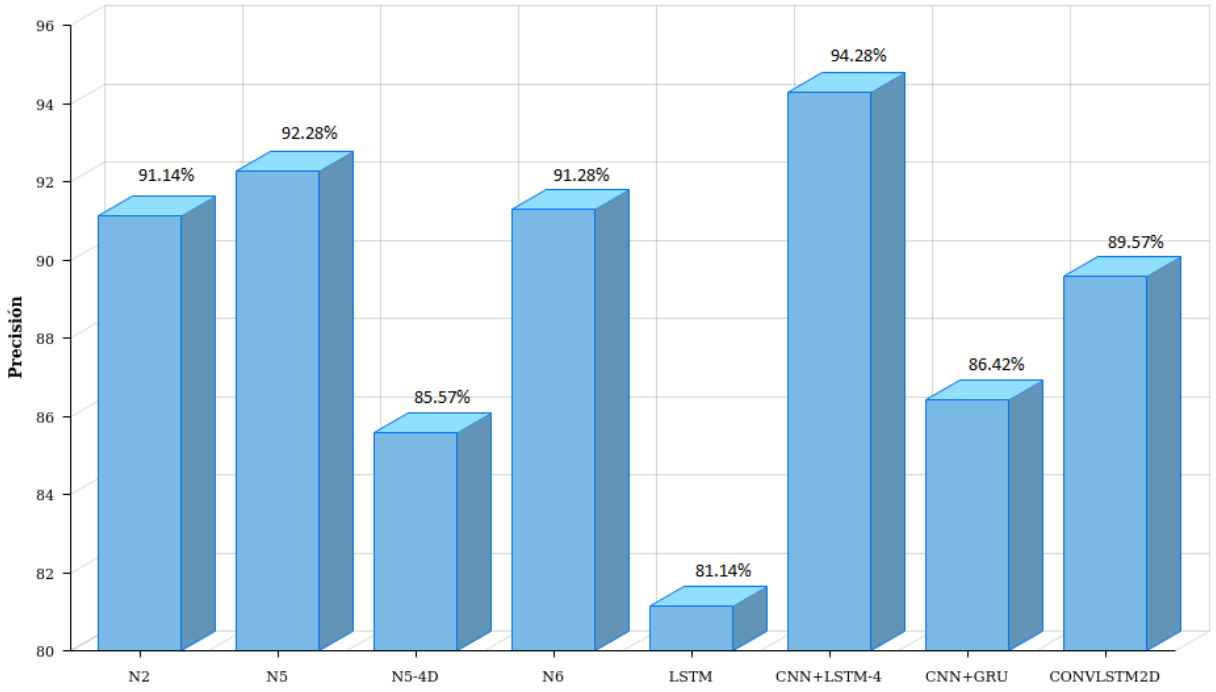


***Figura 54**. Resultados de entrenamientos usando los modelos descritos*

Como se puede observar en la **Figura 54** ninguno de los modelos complejos propuestos en el **apartado 4.1.2** ha conseguido mejorar los resultados obtenidos por los diseños más sencillos **N5** y **CNN+LSTM-4** detallados al comienzo de este capítulo. La hipótesis barajada es que, especialmente para conjuntos de datos pequeños -como es el caso-: **"As model complexity increases, performance on the data used to build the model (training data) improves. However, performance on an independent set (validation data) improves up to a point, then starts to get worse"** (Wu & Shapiro, 2006); es decir, se produce overfitting.

En consecuencia, el análisis de resultados utilizará mayoritariamente aquellos obtenidos por los modelos **N5** y **CNN+LSTM-4**, recomendando el uso de arquitecturas de redes neuronales complejas en aquellos problemas donde se disponga de un conjunto de datos más amplio.

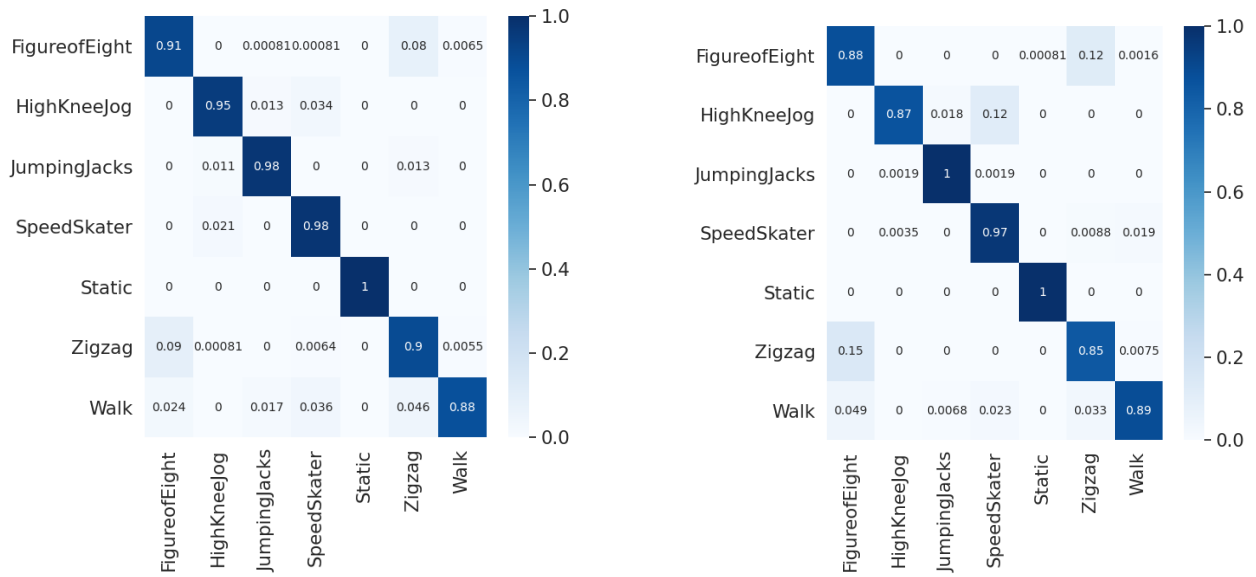


Figura 55. *Matrices de confusión CNN+LSTM-4 VS N5*

Los resultados confirman dos hipótesis formuladas en el **punto 4.1.1.** En **primer lugar**, se da cierta confusión entre los movimientos "dibujar figuras de ocho" y "zigzaguear", debido, en teoría, a la similitud de dichos movimientos en intervalos cortos de tiempo que son encapsulados dentro de las imágenes de movimeinto (por uso de la ténica de segmentación). En **segundo lugar**, las redes neuronales tienen dificultad para clasificar la actividad "caminar" al encontrarse supuestamente este movimiento presente en el resto de actividades (lo que entorpece su discriminación particular).

El efecto adverso presente en estas tres actividades se reproduce para los entrenamientos sobre los modelos N5 y CNN+LSTM-4, donde su clasifcación no alcanza en ningún lugar la media de precisión obtenida por el modelo, como se aprecia en la **Figura 56**.

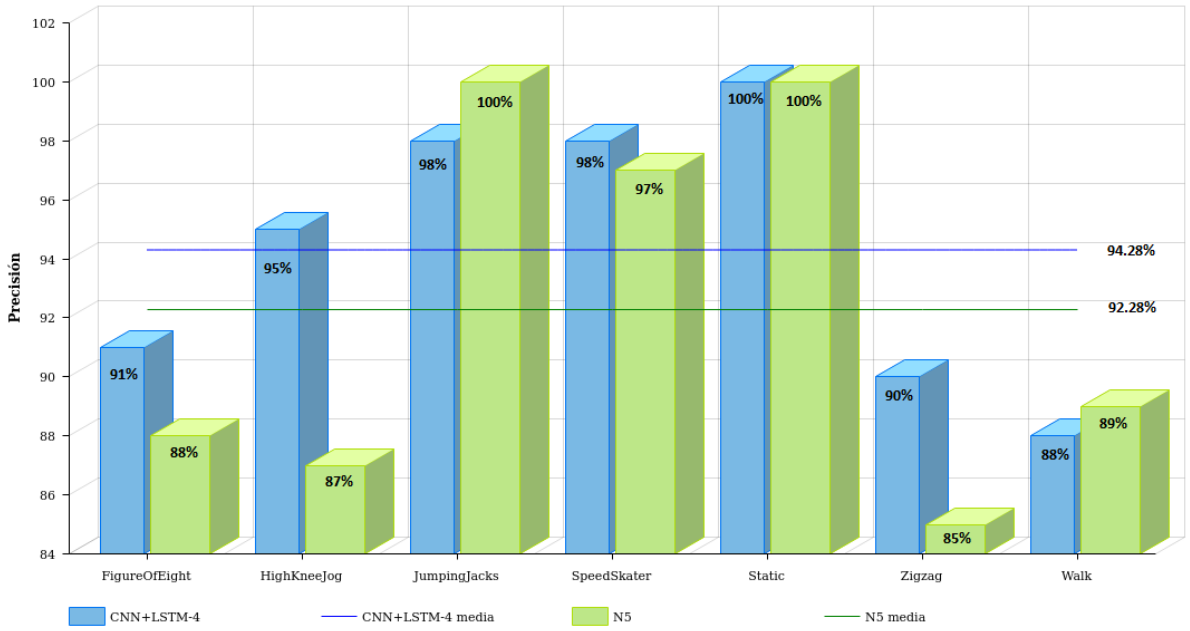


Figura 56. *Precisión por actividades*

De hecho, este resultado es prácticamente homogéneo independientemente del diseño utilizado, siendo la precisión de estas tres actividades siempre inferior a la media de cada modelo.

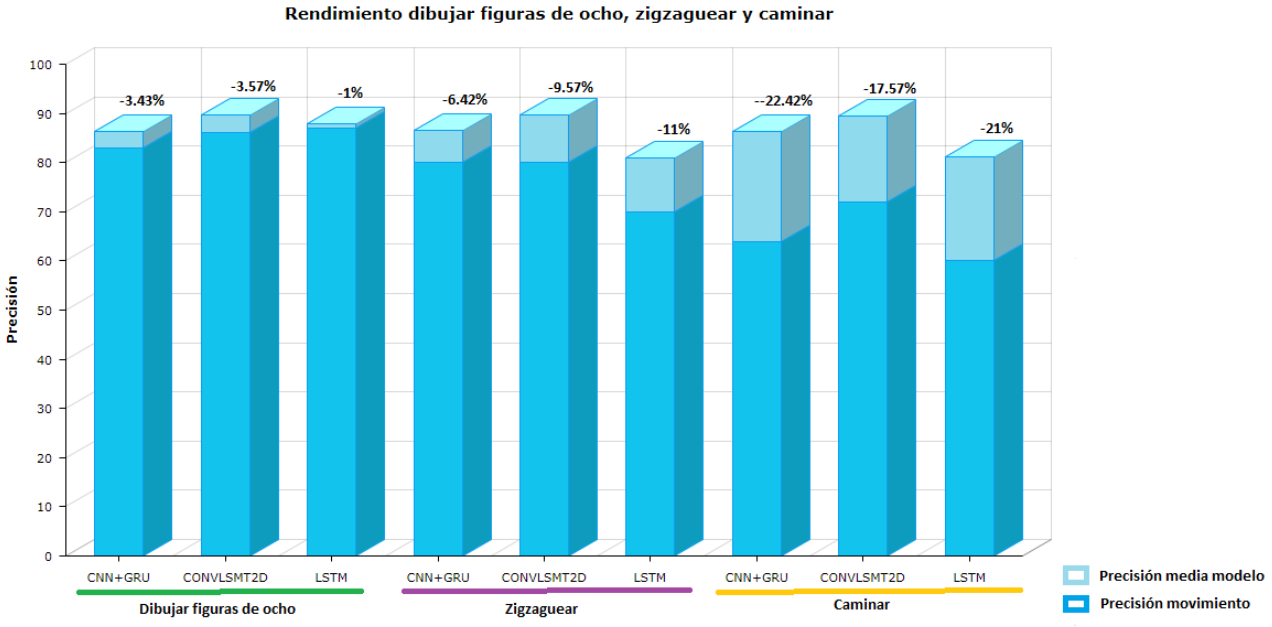


Figura 57. *Precisión subóptima de actividades*

Por otra parte, se reproduce un efecto común a todos los casos de uso del problema HAR conocido como ***inter-subject variation*** que consiste en la variación propia que introduce cada sujeto al realizar una acción. Esta característica inherente del problema HAR dificulta en gran medida su resolución puesto que, independientemente del método utilizado, resulta imprescindible a la par de complicado discriminar qué parte de la información corresponde al movimiento propiamente dicho y qué pertenece a la personalidad del sujeto que realiza la acción.

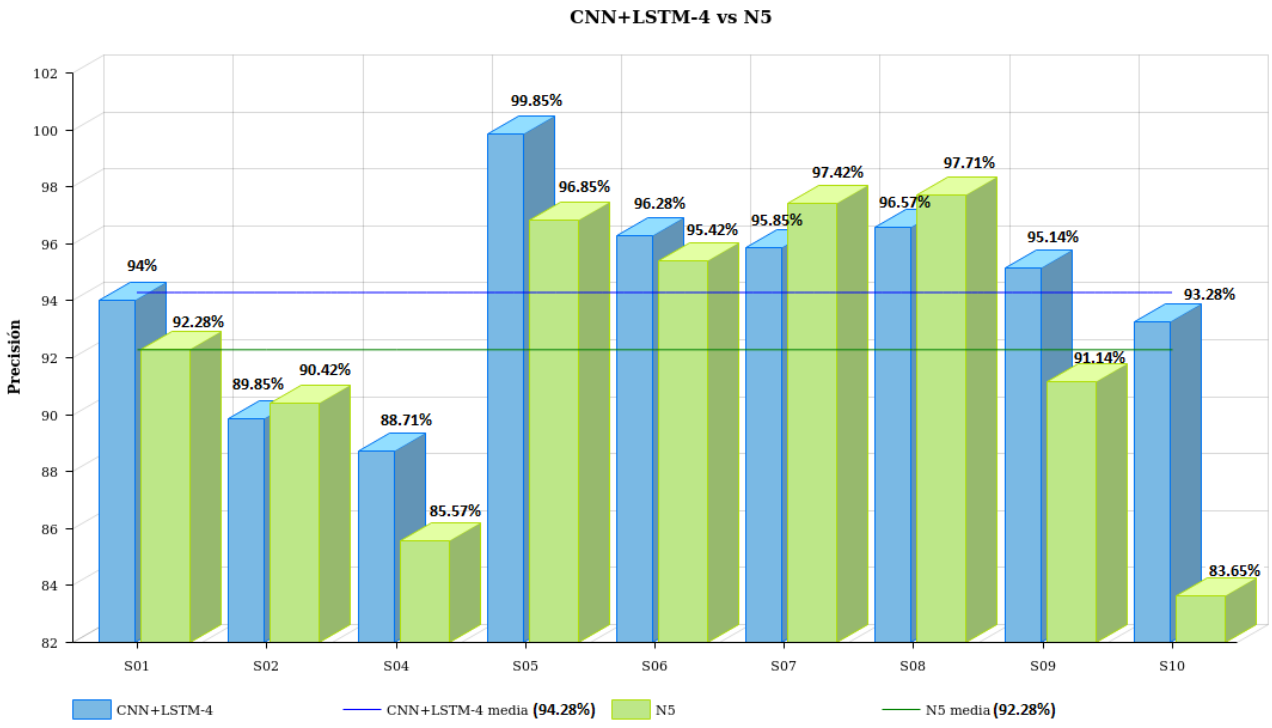


Figura 58. *Precisión por sujetos*

Se aprecia en la **Figura 58** cómo destaca la acertada clasificación de los movimientos realizados por el sujeto S05 frente a la pésima clasificación del sujeto S04. Como concluye Yurtman & Barshan, 2012 es de notoriedad que:

- No siempre el sujeto con mayor precisión es aquel que realiza mejor el movimiento, sino que **el sujeto con mayor precisión suele ser "aquel que realiza una acción de la manera más parecida al resto de sujetos"**.

- Eliminar la media de los valores aportados por los sensores afecta al resultado final (no necesariamente de manera positiva).

Finalmente, nos ayudan a comprender mejor los resultados las métricas de rendimiento extraídas de los dos resultados ganadores.

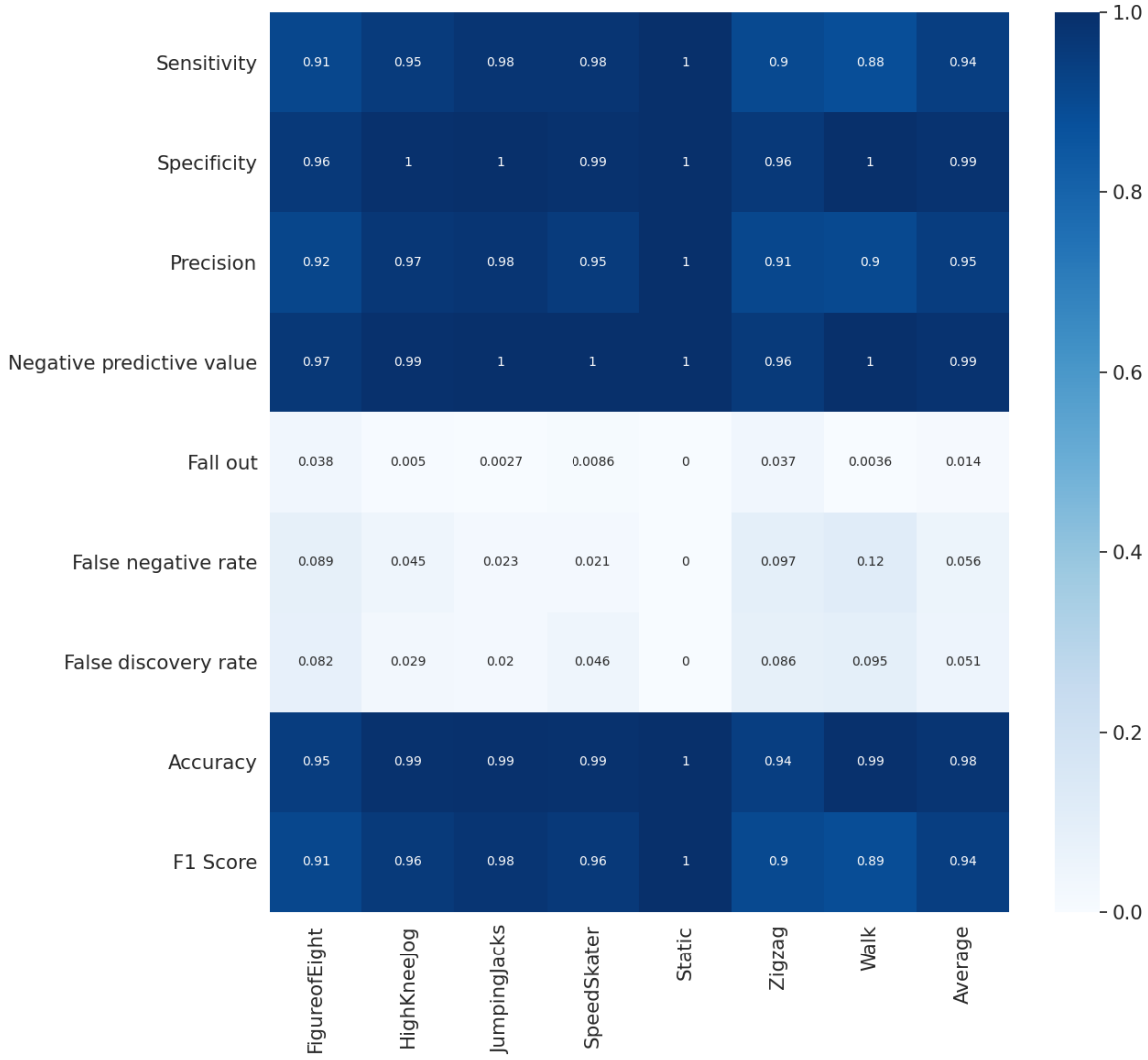


Figura 59. *Métricas de rendimiento CNN+LSTM-4*

Cita Minnen et al., 2006 "el amplio dominio de estudio que abarca el reconocimiento de actividades fuerza el uso de múltiples métricas a la hora de evaluar el rendimiento de los clasificadores". El autor de la presente investigación se ha servido principalmente de: **accuracy**, **precisión, sensitivity y F1 Score** al representar esta última métrica el perfecto equilibrio entre precisión y sensibilidad (Ping, 2018).

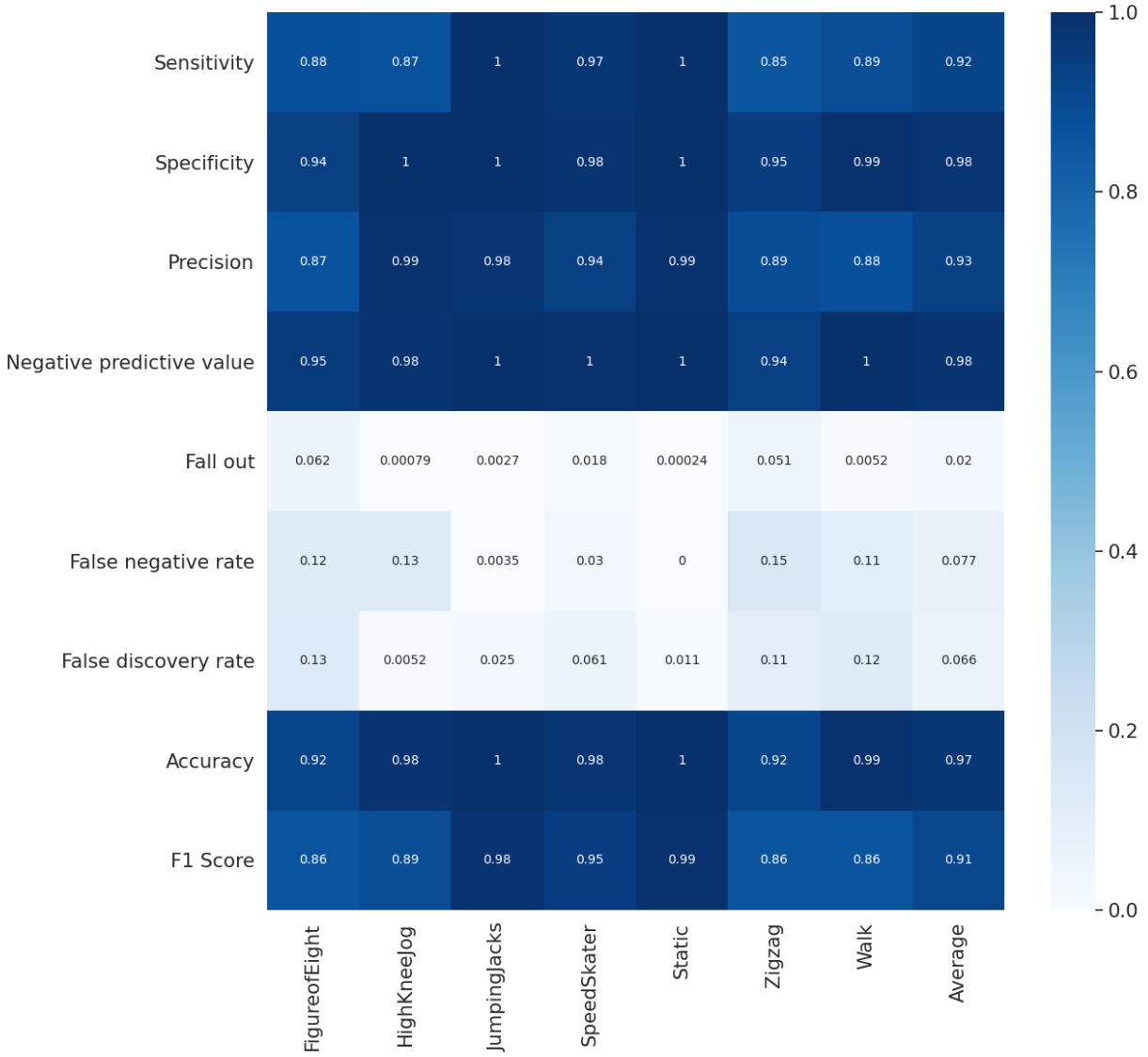


Figura 60. *Métricas de rendimiento N5*

4.2 *Realdisp activity recognition dataset*

La BD RAR supone la segunda integración efectiva en el framework de preprocesado y entrenamiento desarrollado. En primer lugar, ha sido necesario realizar un trabajo previo de adaptación del formato original de las mediciones (dispuesto en ficheros *.log*) al formato de entrada de la herramienta (ficheros *.csv* correctamente nombrados y con la información sensorial colocada en tablas de manera adecuada). El tiempo transcurrido en desarrollar el script de adaptación ha sido de cuatro horas -sirva de referencia para futuras integraciones-.

Posteriormente se ha llevado a cabo un estudio comparativo con los mejores resultados obtenidos en el pasado (Bombín, 2020). Para ello se han utilizado: el modelo **N2**, ya que es una herencia directa del estudio de referencia; un nuevo modelo de estilo CNN+GRU, **C(64)-C(64)-C(64)-DR(0.5)-RG(64)-S_m**, también utilizado por Bombín y que es similar al modelo CNN+GRU presentado en el **apartado 4.1**; y finalmente, se han evaluado también los modelos que mayor precisión ofrecen para la integración de la BD HNS: **N5** y **CNN+LSTM-4**.

Los entrenamientos realizados usando los modelos comentados son de tipo *k-fold* (validación cruzada) con las siguientes características (replicando el estudio previo):

- Imágenes de movimiento con 128 time-steps por imagen con un 75% de superposición entre sí (96 time-steps compartidos).
- Uso de *data augmentation* con grados: 0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180.
- Sujetos: S01, S02, S03, S05, S08, S09, S10, S11, S13, S14, S16, S17.
- Actividades: A09, A10, A11, A12, A13, A19, A20, A21, A24, A25, A31.

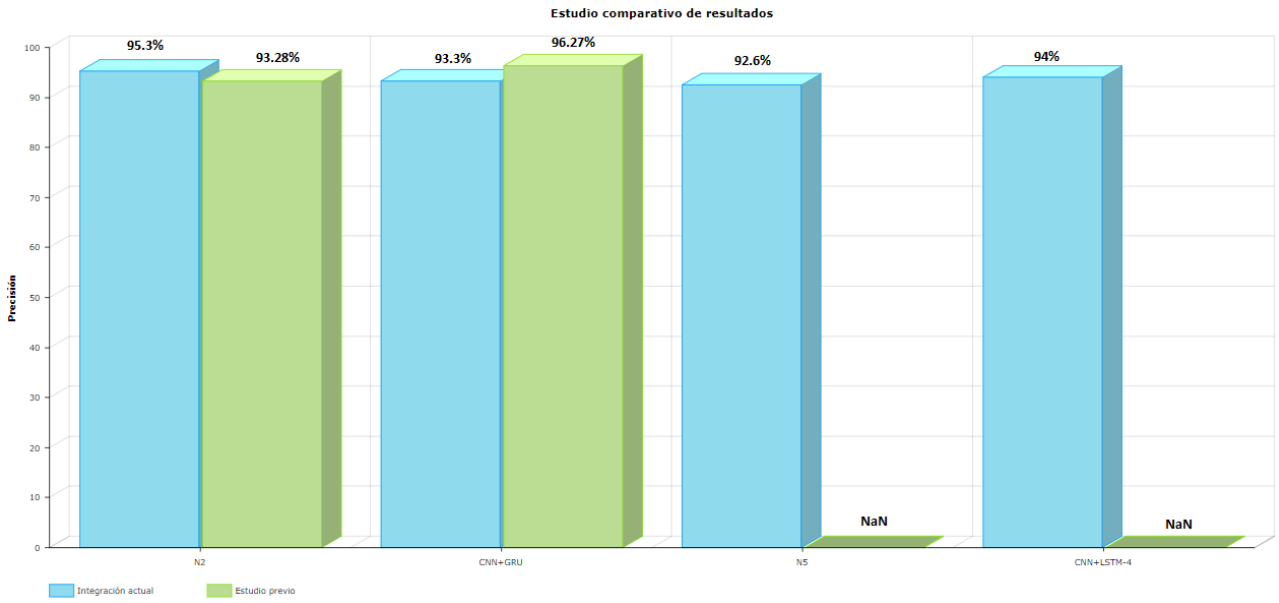


Figura 61. *Comparativa de la precisión entre ambos estudios*

Como se observa en la **Figura 61** la red neuronal que ofrece mayor precisión, del **96.27**%, es aquella basada en el modelo CNN+GRU presentado por Bombín -estudio previo (color verde)-. Por su parte, la precisión obtenida para el modelo similar en la presente investigación es aproximadamente un 3% menor, del 93.3%.

Sin embargo, en lo que respecta al modelo convolucional N2, el presente TFG ofrece un mejor resultado, del **95.3%** frente al 93.28%. Por otra parte, los modelos N5 y CNN+LSTM muestran una precisión en la media de los resultados ya comentados, aunque nos dan "menos juego" puesto que no tenemos modelos similares para comparar en el estudio previo.

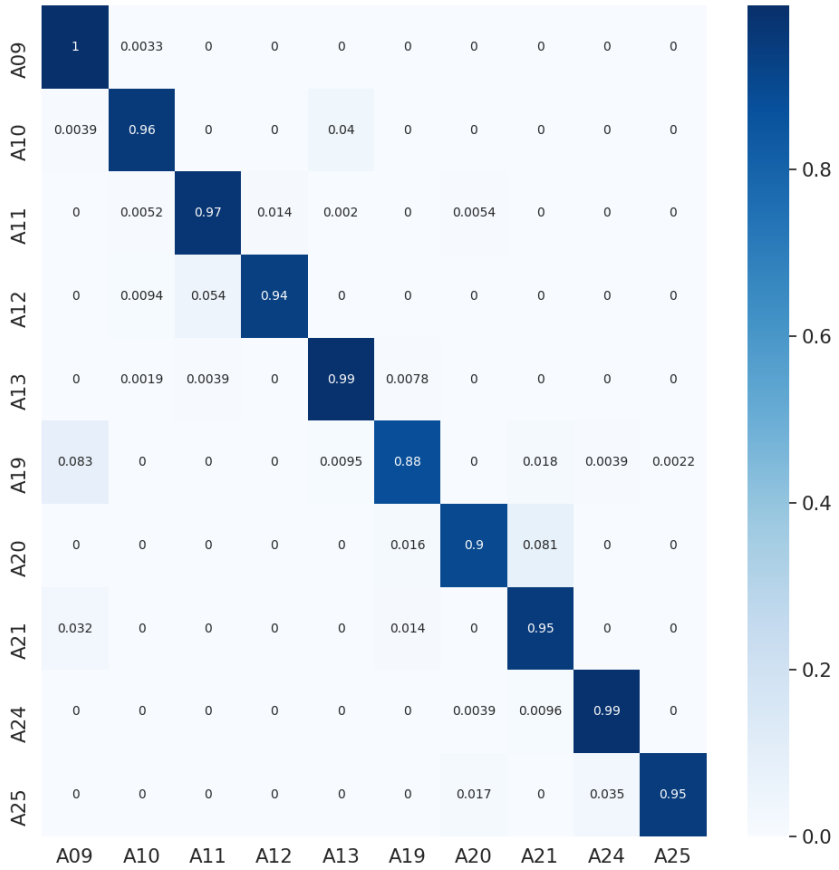


Figura 62. *Matriz de confusión modelo N2 estudio actual:* ***95.3%***

El *mismatch* entre ambos estudios -a pesar de ser ínfimo- encuentra su explicación en la siguiente causa: mientras que los datos utilizados a partir de la BD RAR en el presente estudio han sido tomados directamente de los ficheros *.log*, en la investigación realizada por Bombín se realizó previamente una representación de las actividades en Unity 3D. A través de esta herramienta Bombín calibró las medidas de los sensores y se encargó de limpiar los datos erróneos contenidos en la base de datos; tras dicha limpieza se generaron los datos que sirvieron para entrenar a las redes neuronales.

Adicionalmente, Bombín incorporó la normalización de los datos dentro de cada una de las imágenes de movimiento, lo que, como ya se ha citado, determina el resultado obtenido -no necesariamente de manera positiva- (Yurtman & Barshan, 2012). Estas dos razones explican la varianza en los resultados analizados.

Por otra parte, cabe destacar el buen rendimiento de los modelos N5 y CNN+LSTM-4 a pesar de haber sido dimensionados en origen para trabajar con imágenes de 352 time-steps, en vez de 128.

En cualquier caso, es razonable considerar satisfactoria la integración en la herramienta de la base de datos Neura Sparse, además de constatar la utilidad real del framework, al haber requerido el presente estudio sobre la BD RAR aproximadamente 50 horas, lo que supone un 4.17% del tiempo total utilizado por Bombín (1200 horas).

Capítulo 5. Conclusiones y líneas futuras

El acelerado desarrollo de soluciones software específicas para el entrenamiento de redes neuronales profundas, como Tensorflow y Keras, ha hecho posible la construcción del framework de preprocesado y entrenamiento elaborado en este Trabajo Fin de Grado y así avanzar en el estudio sobre la resolución del problema HAR.

5.1 Conclusiones

En primer lugar, se evidencia la construcción del framework para la resolución del problema HAR tras haber concluido la integración de las bases de datos HNS y RAR, objetivo principal del presente estudio -**Capítulo 1**-. Simultáneamente, se logra el segundo propósito, que no es más que la resolución específica del problema sobre la información contenida en la base de datos HNS.

Finalmente, después de comparar los resultados obtenidos sobre los datos de la BD HNS (sensores optoelectrónicos) y la BD RAR (sensores inerciales), se constata el uso indistinto de ambos tipos de sensores analizados en el **punto 2.1** -tercer objetivo-.

La **primera conclusión** obtenida es la posibilidad efectiva de desarrollar una metodología genérica para la resolución del problema HAR siguiendo la estrategia definida en el **punto 2.2**, usando en la clasificación redes neuronales profundas frente a algoritmos de ML. Bajo la premisa de utilizar siempre información temporal multicanal, el método es independiente al número de actividades, sensores y sujetos utilizados, aunque mejora sus resultados de manera proporcional a la cantidad de datos disponibles.

La **segunda conclusión** es que si, por el contrario, los datos de partida son escasos, es preferible utilizar modelos de redes neuronales sencillos para evitar el overfitting. En ese sentido se evidencia la facultad de resolver el problema HAR a través de distintas arquitecturas de redes neuronales que arrojarán resultados diferentes en función del caso de uso concreto.

La **tercera conclusión** alcanzada es el buen resultado -en términos de precisión- obtenido a pesar de trabajar, en ambos casos, con ventanas temporales no superiores a 3,5 segundos; intervalo de tiempo relativamente corto.

La **conclusión final** tras analizar los resultados obtenidos es que, **en el uso de redes neuronales profundas para la resolución del problema HAR, la clasificación de una actividad no está determinada por el sujeto que mejor realiza un movimiento sino por aquel que lo efectúa de la manera más parecida al resto de sujetos.**

5.2 Líneas futuras

Una vez copados los objetivos de este Trabajo Fin de Grado se puede plantear una multitud de líneas futuras de cara a mejorar la solución desarrollada, así como los resultados obtenidos.

En cuanto a la herramienta construida se propone la integración de técnicas de normalización de los datos en la fase de preprocesado; como se ha comentado en el **punto 4.1.3** dichas técnicas pueden alterar los resultados obtenidos habilitando su mejoría.

De cara a mejorar los resultados obtenidos el autor propone:

- Realizar una optimización automática de los hiperparámetros replicando los entrenamientos detallados en el **Capítulo 4 -**frente a la optimización heurística utilizada en este TFG-, haciendo uso, por ejemplo, de pautas como las descritas por Vasco, 2019.

- Experimentar con nuevos modelos de redes neuronales. No deben ser necesariamente diseños con capas alternativas a las ya vistas (CNN, LSTM, CONVLSTM2D), sino simplemente redimensiones de las propias arquitecturas ya planteadas.

- Eliminar la mitad de las muestras temporales de las actividades contenidas en la base de datos HNS para emular una grabación a 50 Hz (la actual es de 100Hz) y así poder comparar los resultados con la base de datos RAR.

Finalmente, y utilizando como punto de partida la tercera propuesta, se plantea la búsqueda de un modelo único válido para la resolución óptima del problema en ambas bases de datos. Se suscita la posibilidad de incluir en dicha búsqueda más bases de datos con características compatibles.

REFERENCIAS

Banos, O., Toth, M. A., Damas, M., Pomares, H., & Rojas, I. (2014). Dealing with the effects of sensor displacement in wearable activity recognition. *Sensors (Switzerland)*, *14*(6), 9995–10023. https://doi.org/10.3390/s140609995

Bombín, S. S. (2020). *Sistema de Aprendizaje Profundo para reconocimiento de actividades con sensores de captura de movimientos*.

Contenedores de Docker | ¿Qué es Docker? | AWS. (n.d.). Retrieved August 26, 2021, from https://aws.amazon.com/es/docker/

Docker development best practices | Docker Documentation. (n.d.). Retrieved August 26, 2021, from https://docs.docker.com/develop/dev-best-practices/

EarlyStopping. (n.d.). Retrieved August 26, 2021, from https://keras.io/api/callbacks/early_stopping/

FRACTALIA, G. (2016). *Computación ubicua: aplicaciones tecnológicas que mejoran nuestro bienestar*. https://www.fractaliasystems.com/computacion-ubicua-aplicaciones-mejoran-nuestro-bienestar/

García, D. (2019). *Qué es Unity y características principales | OpenWebinars*. undefined-undefined. https://www.mendeley.com/catalogue/a0c9d57c-538d-3711-bc9d-19fdd79de5d9/?utm_source=desktop&utm_medium=1.19.8&utm_campaign=open_catalog&userDocumentId=%7Bacfd6d0e-86d0-3a72-8f27-a4498170675a%7D

Gomila Salas, J. G. (2020a). *Curso completo de Inteligencia Artificial con Python | Udemy*. https://www.udemy.com/course/curso-completo-de-inteligencia-artificial/learn/lecture/12286870#overview

Gomila Salas, J. G. (2020b). *Deep Learning de A a Z:redes neuronales en Python desde cero | Udemy*. https://www.udemy.com/course/deep-learning-a-z/learn/lecture/17063574#overview

Image data preprocessing. (n.d.). Retrieved August 26, 2021, from https://keras.io/api/preprocessing/image/

Kong API Gateway - KongHQ. (n.d.). Retrieved August 26, 2021, from https://konghq.com/kong/

Koo, J. (2020, July). *Speed Up Development with a JSON Configuration File - alwaysAI Blog*. https://alwaysai.co/blog/speed-up-development-with-a-json-configuration-file

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 1–10.

Make - GNU Project - Free Software Foundation. (n.d.). Retrieved August 26, 2021, from https://www.gnu.org/software/make/

Matplotlib: Python plotting — Matplotlib 3.4.3 documentation. (n.d.). Retrieved August 26, 2021, from https://matplotlib.org/

Meng, Z., Zhang, M., Guo, C., Fan, Q., Zhang, H., Gao, N., & Zhang, Z. (2020). Recent progress in sensing and computing techniques for human activity recognition and motion analysis. *Electronics (Switzerland)*, *9*(9), 1–19. https://doi.org/10.3390/electronics9091357

Minnen, D., Westeyn, T. L., Starner, T., Ward, J. a, & Lukowicz, P. (2006). Performance Metrics and Evaluation Issues for Continuous Activity Recognition. *Proc. Int. Workshop on Performance Metrics for Intelligent Systems*, 141–148.

ModelCheckpoint. (n.d.). Retrieved August 26, 2021, from https://keras.io/api/callbacks/model_checkpoint/

Mutegeki, R., & Han, D. S. (2020). A CNN-LSTM Approach to Human Activity Recognition. *2020 International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2020*, 362–366. https://doi.org/10.1109/ICAIIC48513.2020.9065078

NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy. (n.d.). Retrieved August 26, 2021, from https://www.nginx.com/

NumPy. (n.d.). Retrieved August 26, 2021, from https://numpy.org/

Ordóñez, F. J., & Roggen, D. (2016). Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. *Sensors (Switzerland)*, *16*(1). https://doi.org/10.3390/s16010115

pandas - Python Data Analysis Library. (n.d.). Retrieved August 26, 2021, from https://pandas.pydata.org/

Patel, A. (2018). *Introduction to CNN, LSTM, GRU, OBJECT RECOGNITION & DATASETS | by Ashish Patel | Medium*. https://medium.com/@apatel67/introduction-to-cnn-lstm-gru-object-recognition-datasets-7dfa4f8ad8f6

Ping, K. (2018, March). *Accuracy, Precision, Recall or F1? | by Koo Ping Shung | Towards Data Science*. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

Python - Official Image | Docker Hub. (n.d.). Retrieved August 26, 2021, from https://hub.docker.com/_/python

pytoolz/toolz: A functional standard library for Python. (n.d.). Retrieved August 26, 2021, from https://github.com/pytoolz/toolz

Requests: HTTP for Humans™ — Requests 2.26.0 documentation. (n.d.). Retrieved August 26, 2021, from https://docs.python-requests.org/en/master/

Ribeiro, P. M. S., Matos, A. C., Santos, P. H., & Cardoso, J. S. (2020). Machine learning improvements to human motion tracking with imus. *Sensors (Switzerland)*, *20*(21), 1–21. https://doi.org/10.3390/s20216383

scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation. (n.d.). Retrieved August 26, 2021, from https://scikit-learn.org/stable/

Shah, D., Malhotra, A., & Patidar, M. (2021). Sensor Based Human Activity Recognition by Multi-Headed ConvLSTM. *2021 7th International Conference on Advanced Computing and Communication Systems, ICACCS 2021*, 1127–1129. https://doi.org/10.1109/ICACCS51430.2021.9441948

Sy, L. (2019). *Replication Data for Estimating Lower Limb Kinematics using a Reduced Wearable Sensor Count. - Harvard Dataverse*. https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/9QDD5J

TensorFlow. (n.d.). Retrieved August 26, 2021, from https://www.tensorflow.org/

tf.data: compila canalizaciones de entrada de TensorFlow. (n.d.). Retrieved August 26, 2021, from https://www.tensorflow.org/guide/data

Vasco, R. (2019). *Optimización Automática de Hiperparámetros en Modelos de Aprendizaje Automático mediante PBIL*. 1–83. http://darkmatter.ciemat.es/documents/585242/809389/AndresVasco-TFM.pdf/b53043be-2860-4dbc-b861-4aabf573dd95

Waskom, M. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/JOSS.03021

Welcome to Flask — Flask Documentation (2.0.x). (n.d.). Retrieved August 26, 2021, from https://flask.palletsprojects.com/en/2.0.x/

Welcome to PyJWT — PyJWT 2.1.0 documentation. (n.d.). Retrieved August 26, 2021, from https://pyjwt.readthedocs.io/en/stable/

Wu, H., & Shapiro, J. L. (2006). Does over-fitting affect performance in estimation of distribution algorithms. *GECCO 2006 - Genetic and Evolutionary Computation Conference*, *1*, 433–434. https://doi.org/10.1145/1143997.1144078

Yang, J. B., Nguyen, M. N., San, P. P., Li, X. L., & Krishnaswamy, S. (2015). Deep convolutional neural networks on multichannel time series for human activity recognition. *IJCAI International Joint Conference on Artificial Intelligence*, *2015-Janua*(Ijcai), 3995–4001.

Yurtman, A., & Barshan, B. (2012). Inter- and intra-subject variations in activity recognition using inertial sensors and magnetometers. *The 5th International Conference on Cognitive Systems*, 8. http://kilyos.ee.bilkent.edu.tr/~billur/publ_list/cogsys12.pdf

Zhang, M., & Sawchuk, A. A. (2012). USC-HAD: A daily activity dataset for ubiquitous activity recognition using wearable sensors. *UbiComp'12 - Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 1036–1043.

Zmitri, M., Fourati, H., & Vuillerme, N. (2019). Human activities and postures recognition: From inertial measurements to quaternion-based approaches. *Sensors (Switzerland)*, *19*(19), 1–18. https://doi.org/10.3390/s19194058

ANEXOS

ANEXO I
Presupuesto

En este **Anexo II** se ha realizado una aproximación del presupuesto necesario para realizar un proyecto similar al presente trabajo de fin de grado.

En primer lugar, es preciso conocer el perfil del ingeniero que va a desarrollar el proyecto para calcular el coste de personal asociado. Entre los requisitos se distingue:

- Conocimiento fluido de lenguaje de programación Python.
- Dominio de herramientas transversales como Make, Docker o Shell scripting.
- Conocimientos específicos en el ámbito de la inteligencia artificial, familiarizado con el uso de librerías como **Tensorflow** y **Keras**.

Por tanto, nos encontramos ante un profesional categorizado tradicionalmente como "Junior" -experiencia laboral entre 2 y 4 años- con cierto grado de especialización en la materia. En consecuencia, la estimación salarial para el presente proyecto es:

Horas	Coste/Hora	Total
863	18.59 €	16043.17 €

Por otra parte, se debe cuantificar el coste material de cada uno de los recursos utilizados.

Recurso	Coste
Ordenador sobremesa	2345 €
Luz	327 €
Formación	186 €

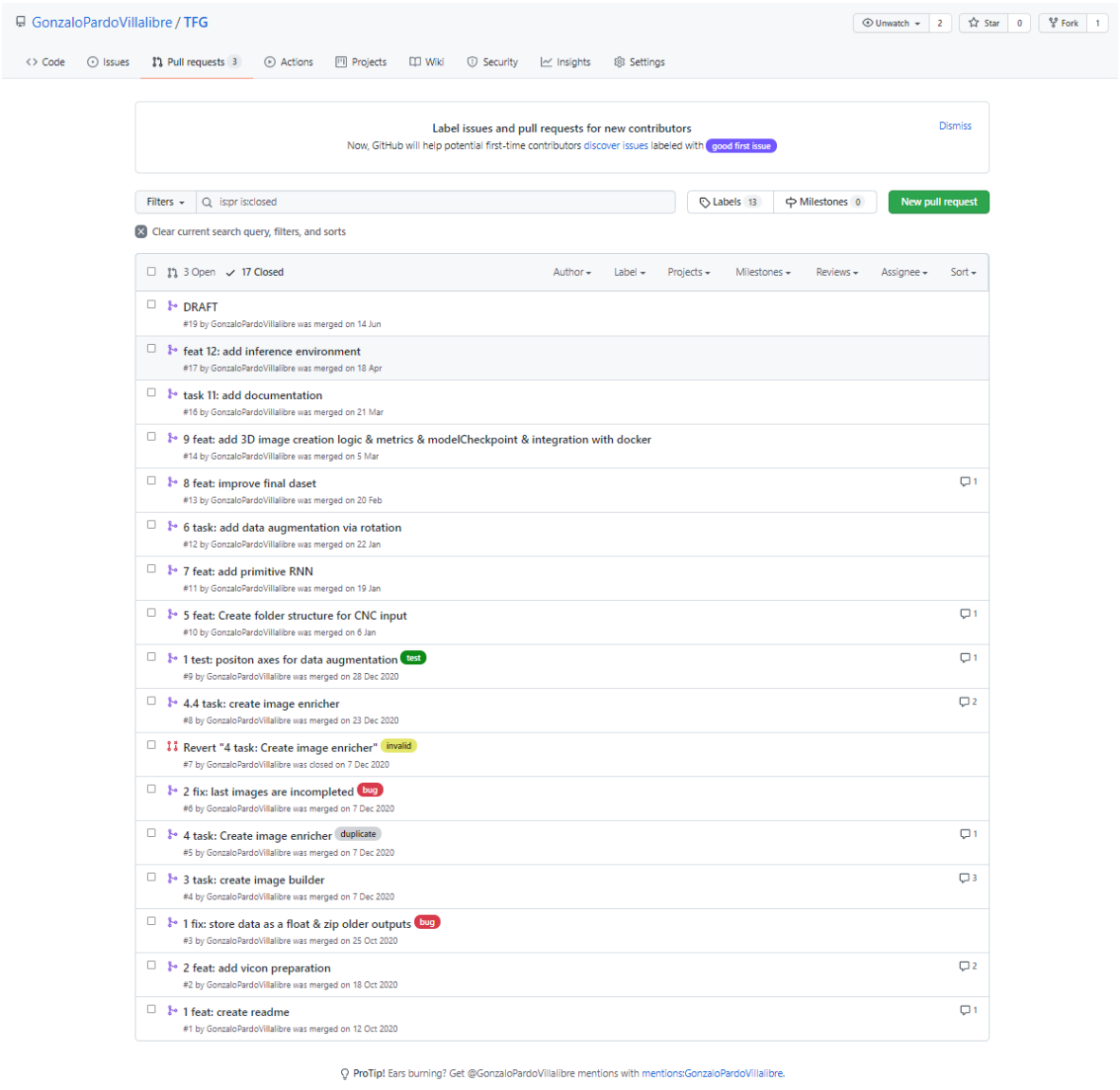
El presupuesto final del proyecto asciende, por tanto, a:

Coste total
18901.17 €

ANEXO II
Guía de uso

Se presenta un extracto de primera página de la guía de uso del framework publicada en GitHub (https://github.com/GonzaloPardoVillalibre/TFG). Está escrita en inglés y usa formato **Markdown**, permitiendo así realizar referencias dinámicas al resto de ficheros del proyecto sobre el propio texto-razón por la cual es complicado realizar una copia estática de la propia guía-.

En este mismo repositorio público se ha alojado la totalidad del código de la herramienta, de naturaleza ***open source*** y disponible para uso no comercial. El uso GitHub permite no solo consultar todas las tareas que se han ido desarrollando a lo largo de la creación del framework, sino también hacer más accesible el desarrollo de futuras funcionalidades.



Captura web 1. *Captura del proceso de desarrollo de la herramienta vía GitHub*

Deep learning networks for human activity recognition

Pre-processing and training framework in tensorflow

This project provides a framework based on docker and aims to expedite the ***human activity classification*** training process. Thus, three separate environments are provided:

- Pre-processing environment.
- Training environment.
- Inference environment

While the ***training environment*** has a more general use, providing a generic tool to solve a vast amount of problems, the ***pre-processing environment*** has its focus on pre-processing human activity datasets (measured in a **quaternion** form) to solve the already mentioned "***human activity classification problem***".

Meanwhile, the ***inference environment*** serves a development framework to deploy a flask rest API. This API loads the desired neural network model and is able to answer prediction requests. This API is also focused on the human activity classification problem, but can be easily tuned for a more generic purpose. To know more about flask you can visit the official flask webpage.

Pre-requirements:

- Docker v17
- GNU Make

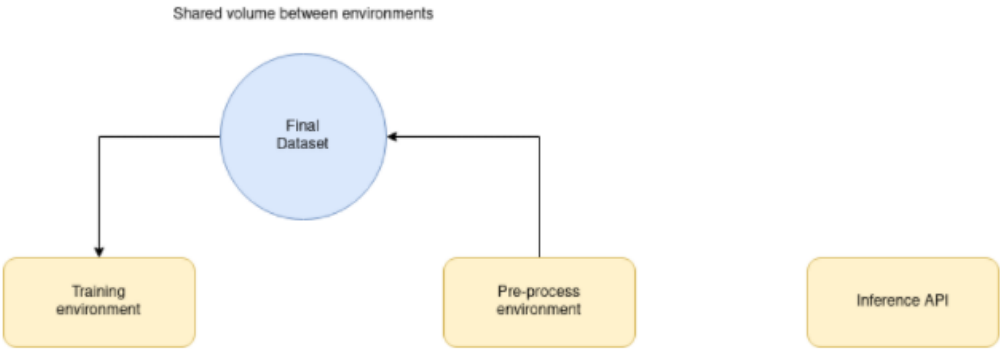
The following instruction launches both environments:

```
# Launch the development environment.
make develenv-up
```

Also a `make help` utility is available to the developer.

Docker architecture

For the reference there is a generic view of the architecture:



As you'd have noticed, the ***inference environment*** has little to do with the previous architecture and can be treated as an individual component. Ideally, this component will be the only one deployed in a production environment.

More information can be found here: data structure documentation

Pre-processing environmnent

```
# Enter the pre-processing environment
make preprocess-sh
```

The guide for this environment can be found here: pre-process environment documentation

Training environment

```
# Enter the training environment
make train-sh
```

The guide for this environment can be found here: train environment documentation

Inference environment

```
# Enter the pre-processing environment
make inference-sh
```

Although, the most useful command in this environment may be:

```
# Display container logs
docker logs -f framework_inferencer_1
```

The guide for this environment can be found here: inference environment documentation

What is this project all about?

This project is the final assignment for Gonzalo Pardo Villalibre. The aim will be to detect which activity is a certain subject performing, minimizing the number of sensors needed. Therefore the student will take advantage of the use of NN (neural networks) from different types such as CNN (convolutional networks) or RNN (recurrent networks) such LSTM.

On this journey the developer decided to not only solve the concrete problem, but also to create a reusable framework making the process easier for future investigators.

More info about specific problem can be found here: more info

Contact

You can contact the creator via e-mail at: `gonzalopmb@gmail.com`

Licensing

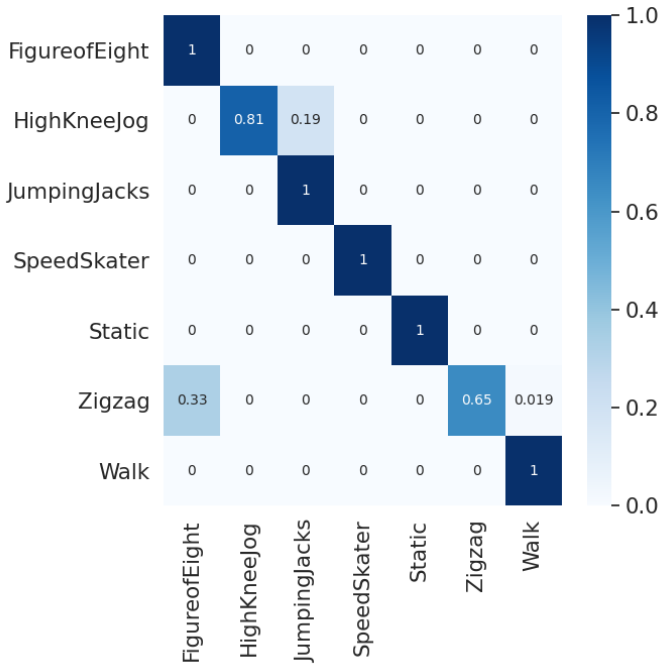
Captura web 2. *Extracto de la guía de uso del framework*

ANEXO III
Matrices de confusión y métricas de rendimiento

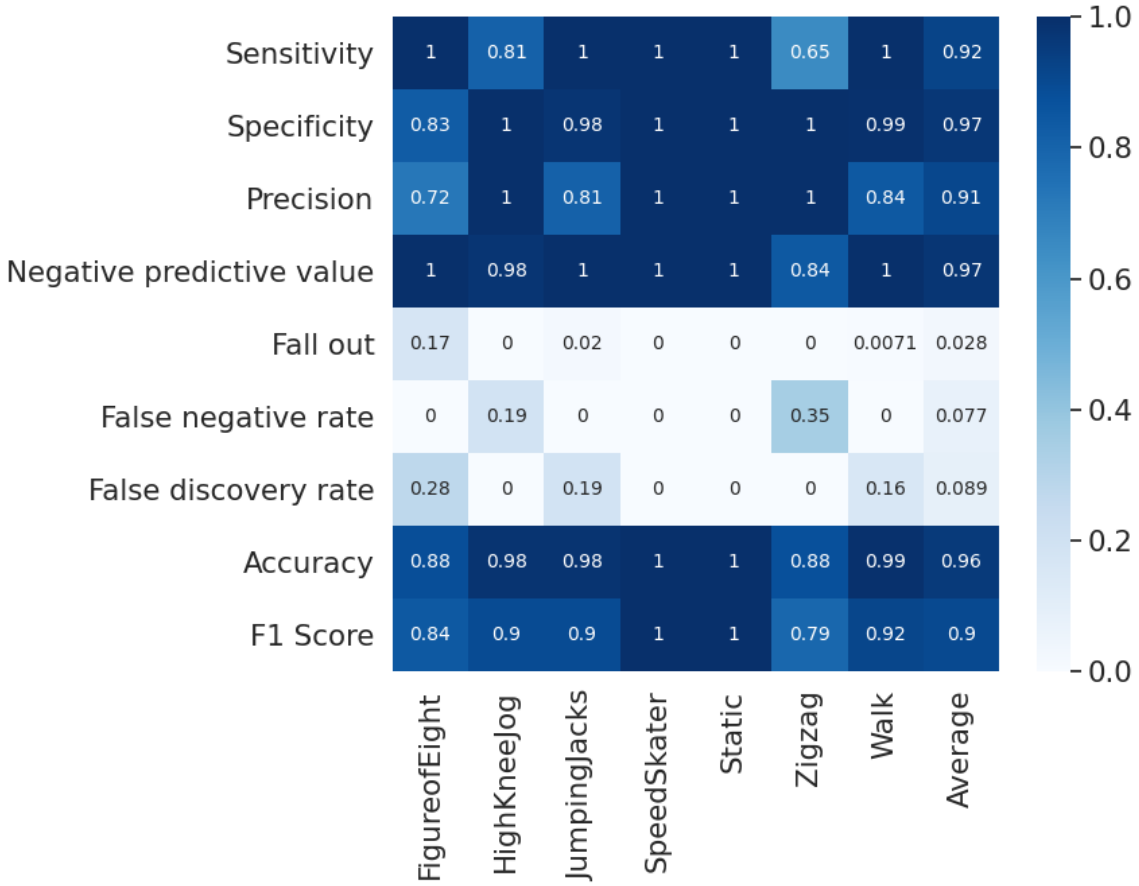
Se presentan en detalle los resultados de los dos entrenamientos 8-fold más relevantes: N5 y CNN+LSTM-4, que han sido analizados en el **Capítulo 4.**

Para cada sujeto del 8-fold se expone la matriz de confusión y su correspondiente matriz de métricas de rendimiento. Finalmente se muestran los resultados agregados.

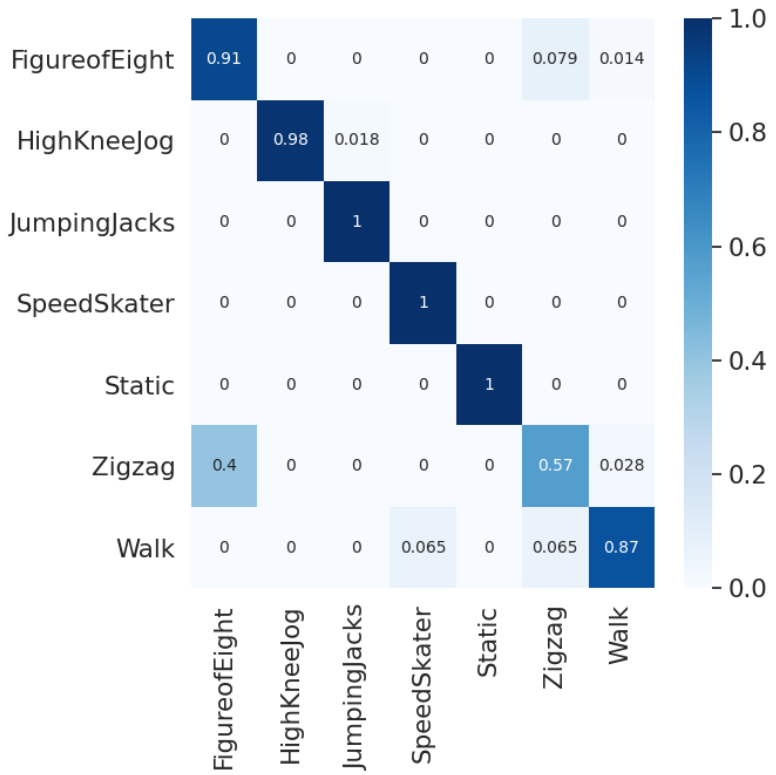
3.1 Modelo N5



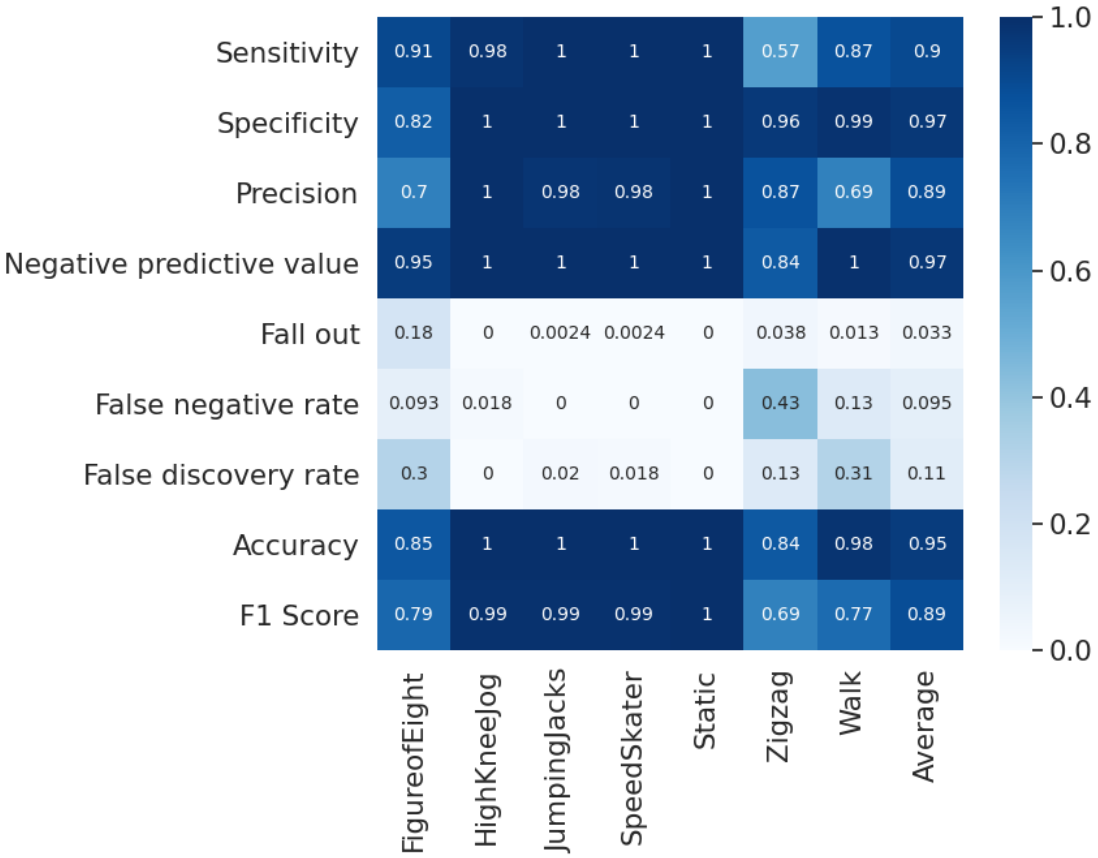
***Matriz de confusión 1**. Sujeto S01*



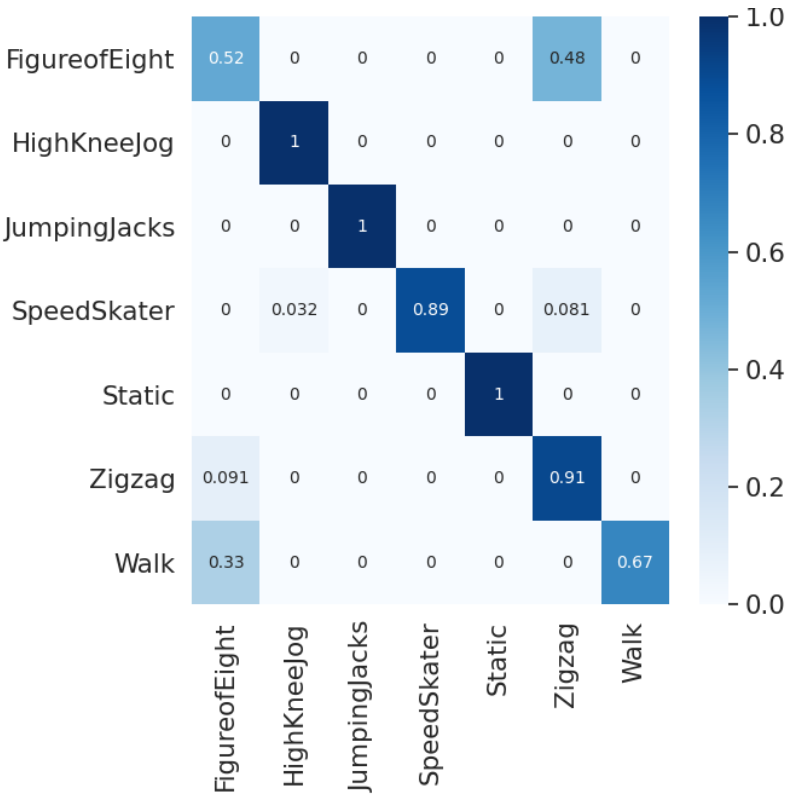
***Métricas de rendimiento 1**. Sujeto S01*



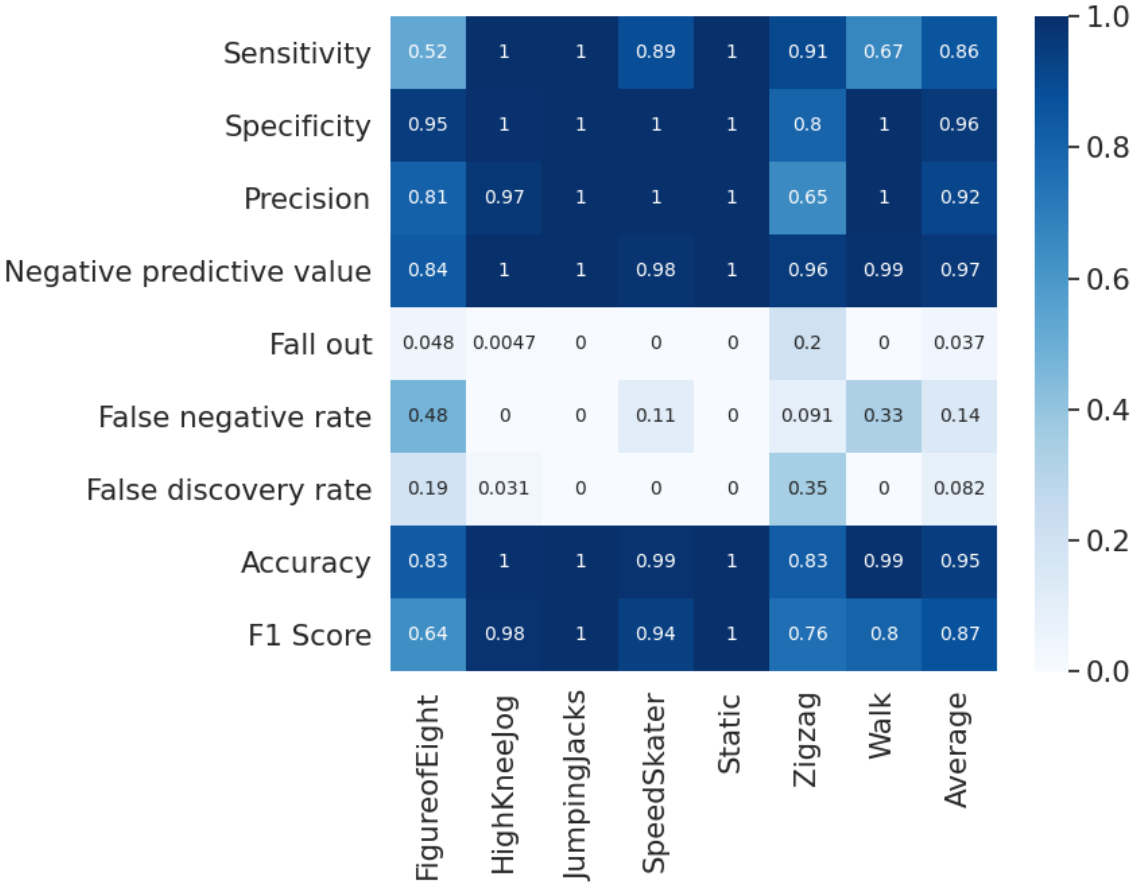
***Matriz de confusión 2**. Sujeto S02*



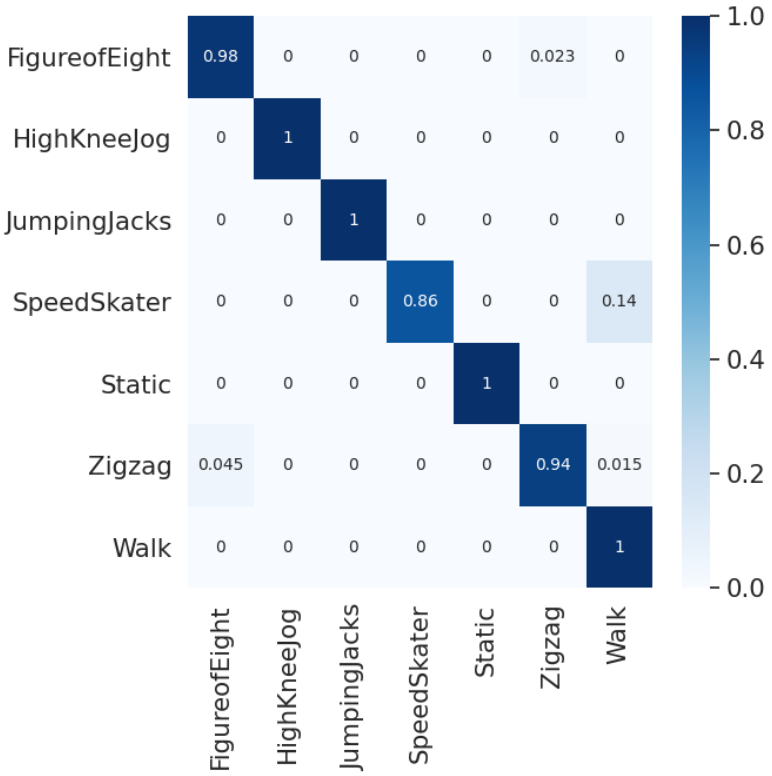
***Métricas de rendimiento 2**. Sujeto S02*



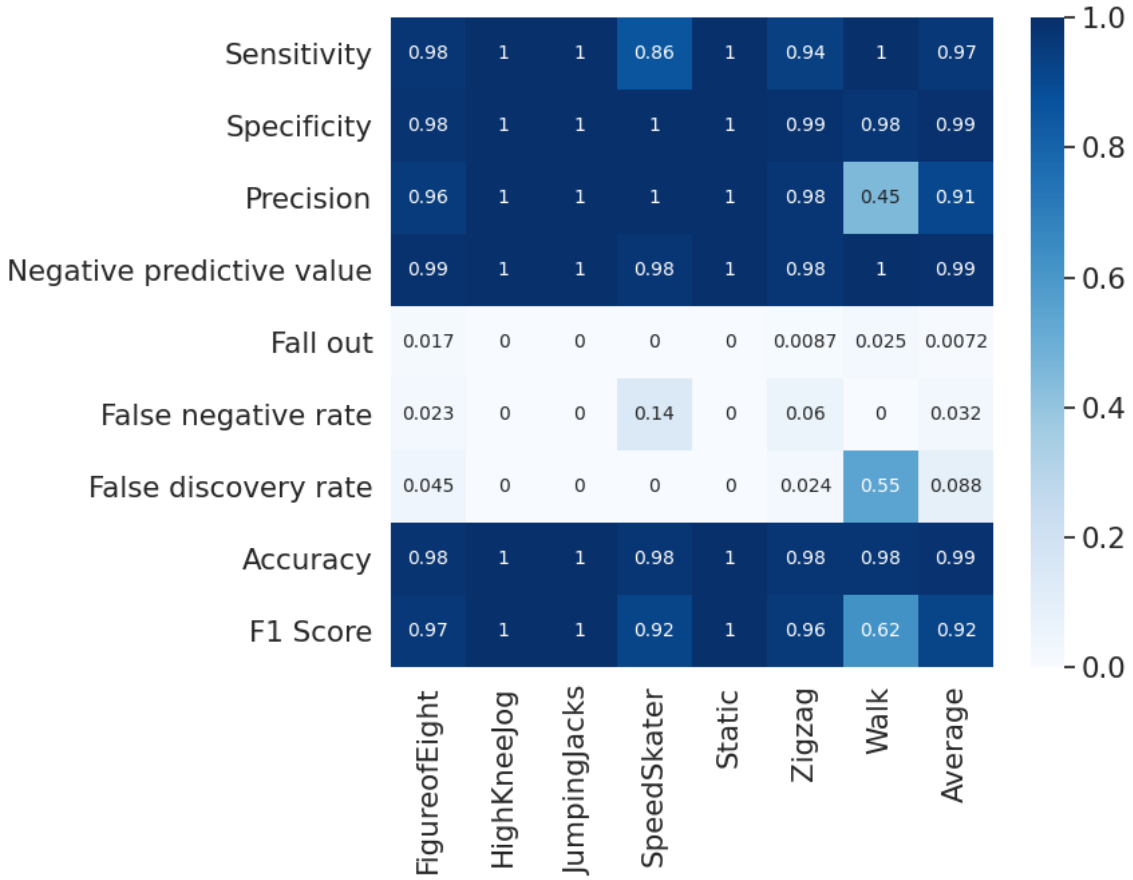
Matriz de confusión 3. *Sujeto S04*



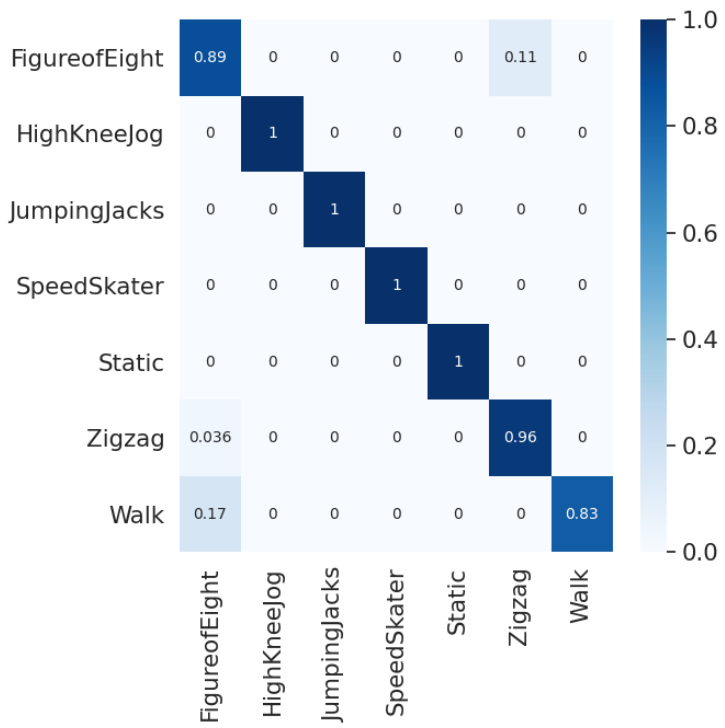
Métricas de rendimiento 3. *Sujeto S04*



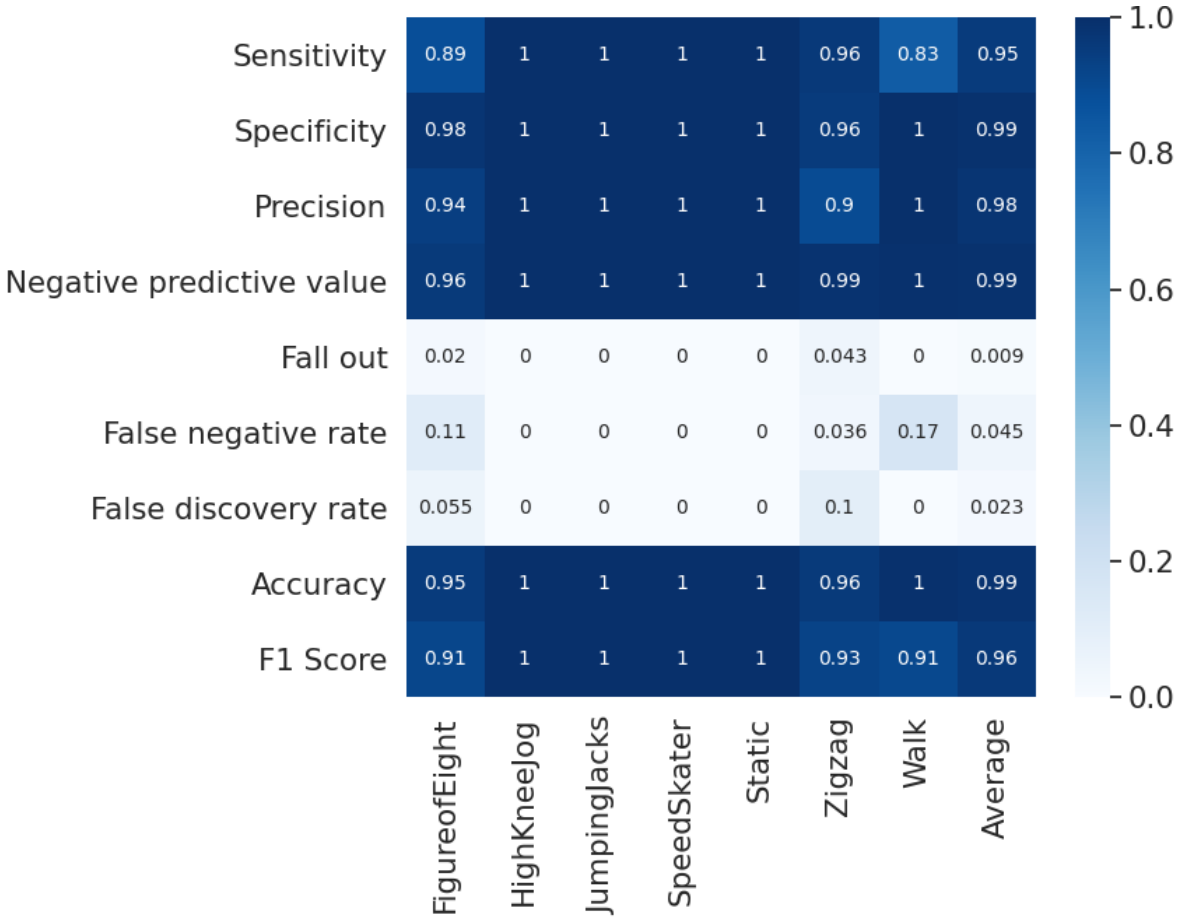
Matriz de confusión 4. *Sujeto S05*



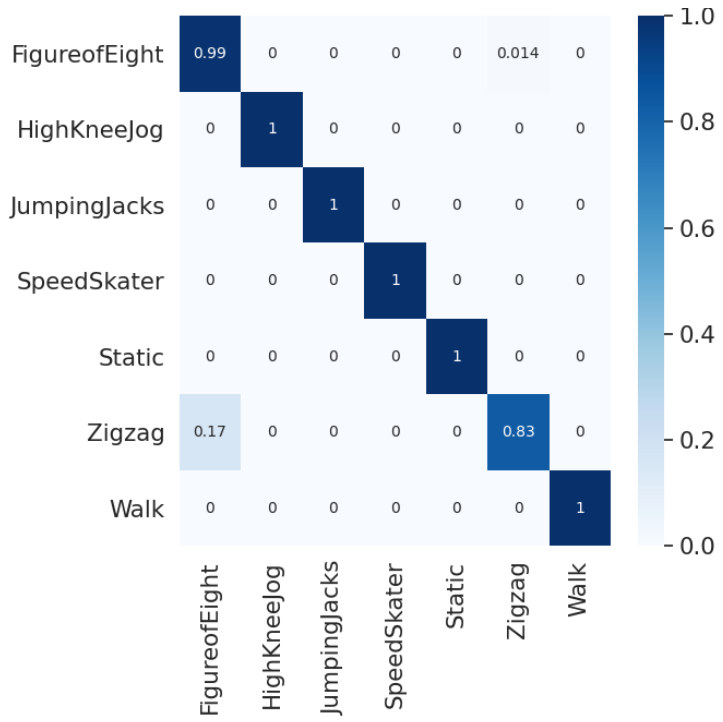
Métricas de rendimiento 4. *S05*



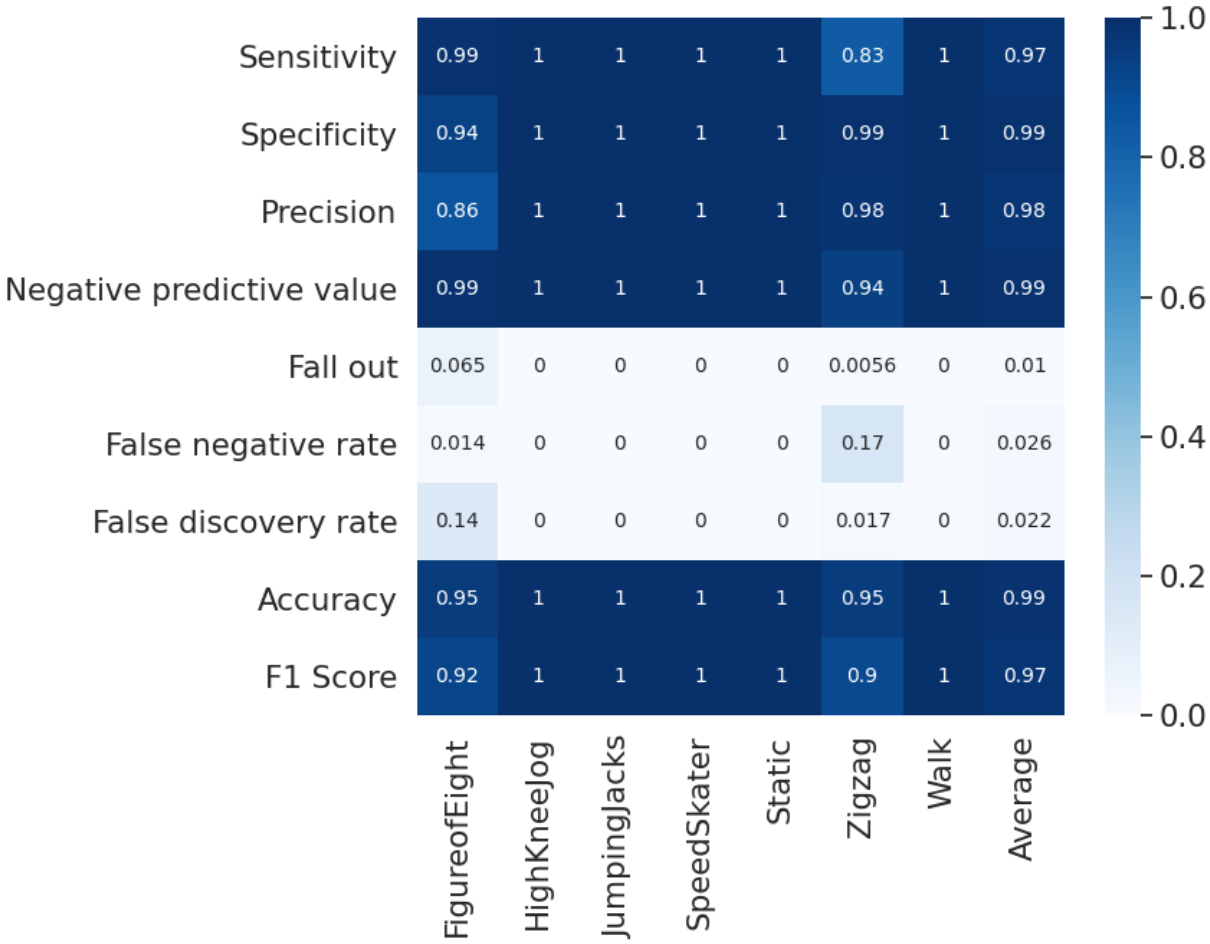
***Matriz de confusión 5**. Sujeto S06*



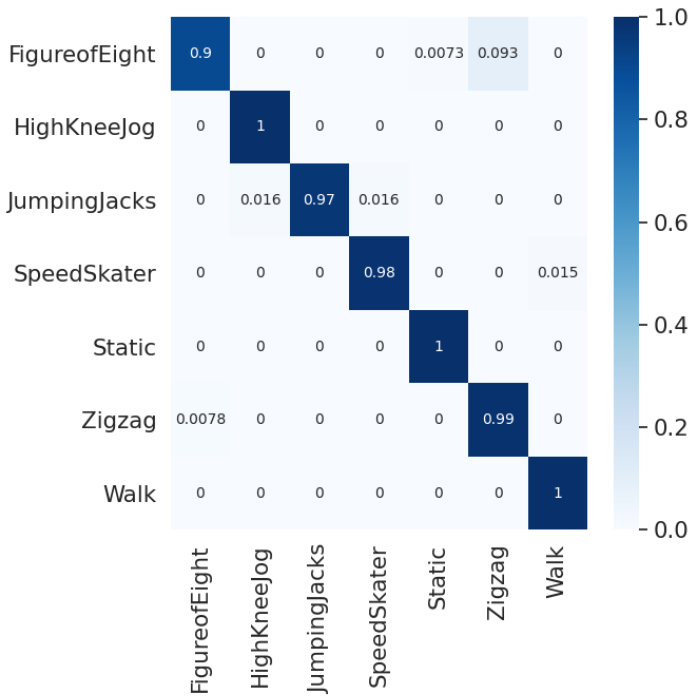
***Métricas de rendimiento 5.** Sujeto S06*



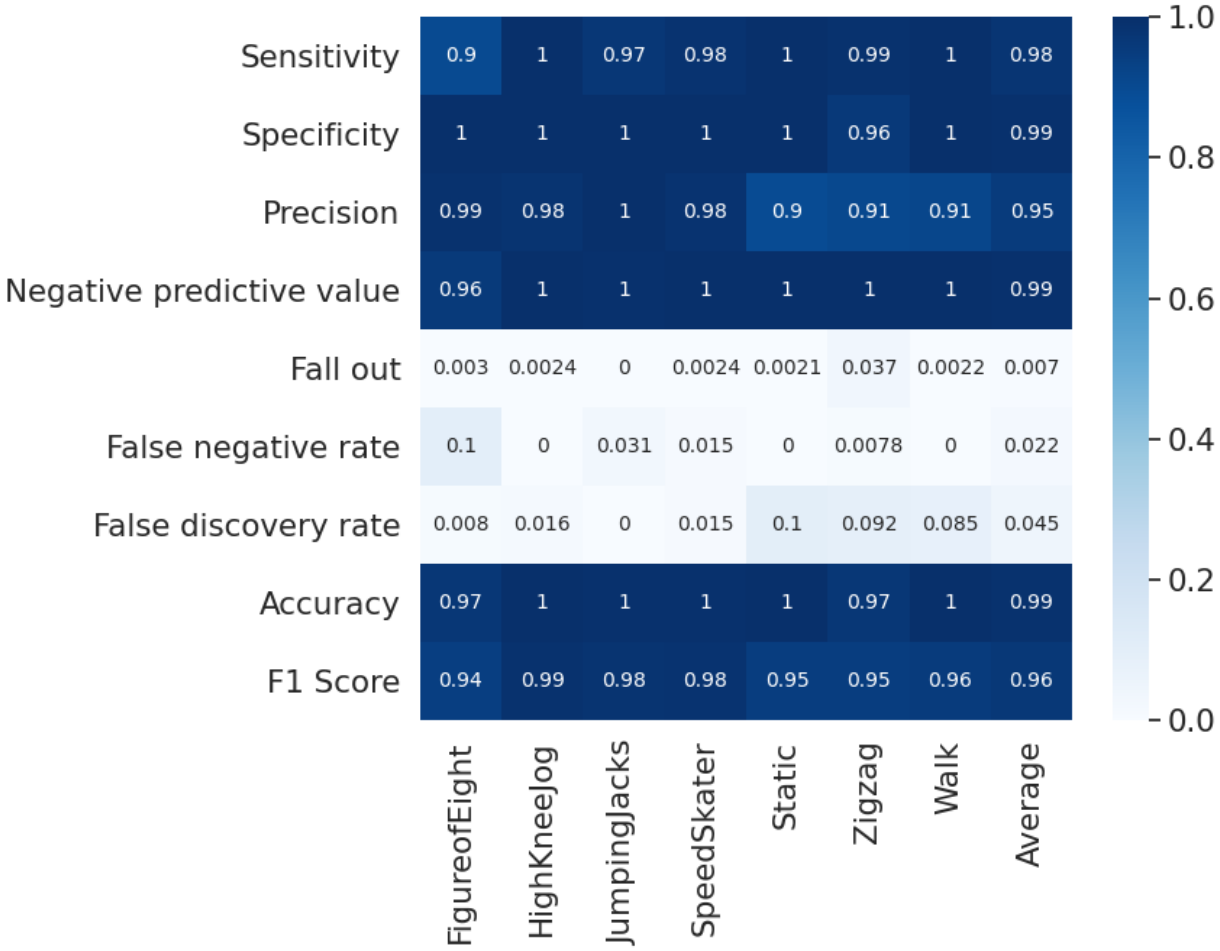
Matriz de confusión 6. *Sujeto S07*



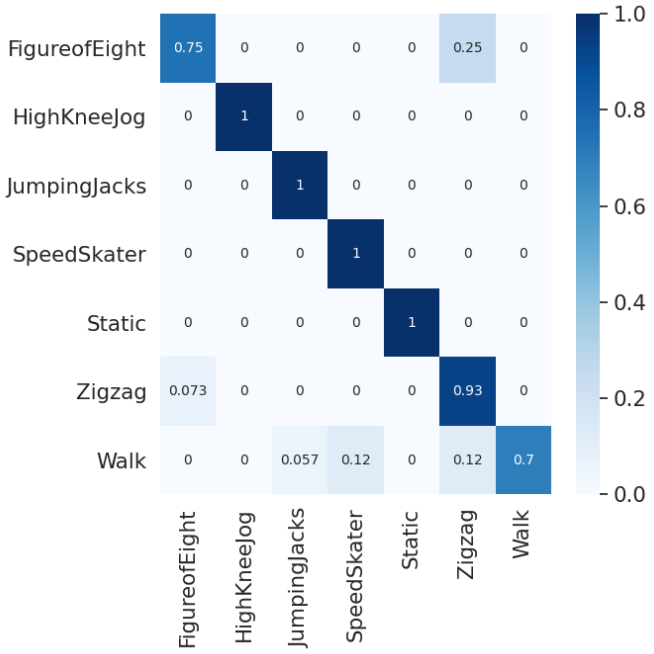
Métricas de rendimiento 6. *Sujeto S07*



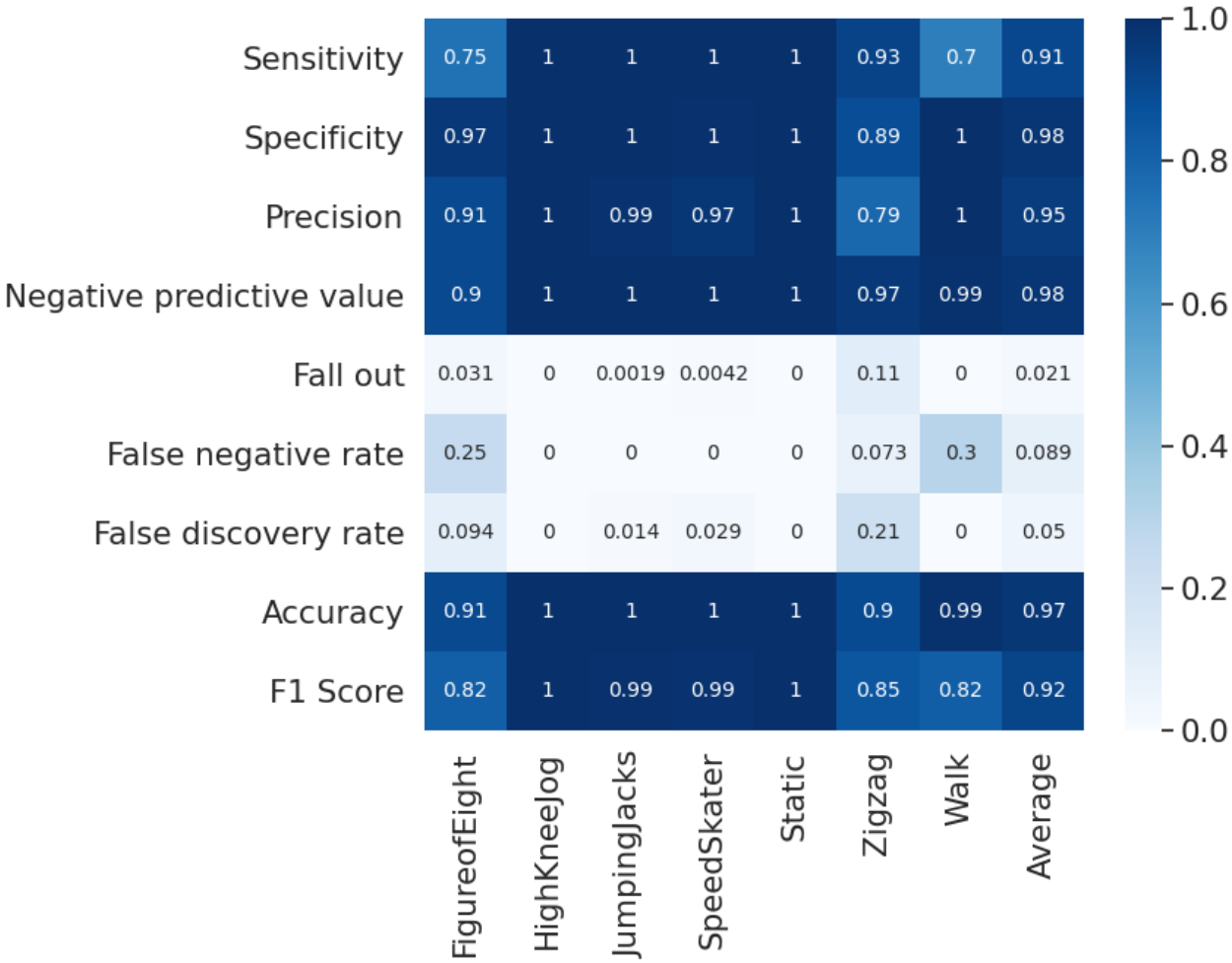
***Matriz de confusión** 7. Sujeto S08*



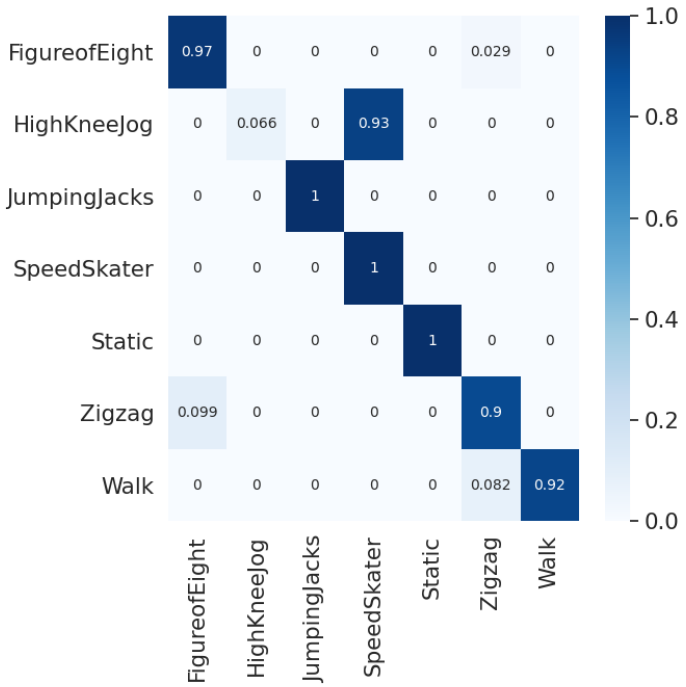
***Métricas de rendimiento** 7. Sujeto S08*



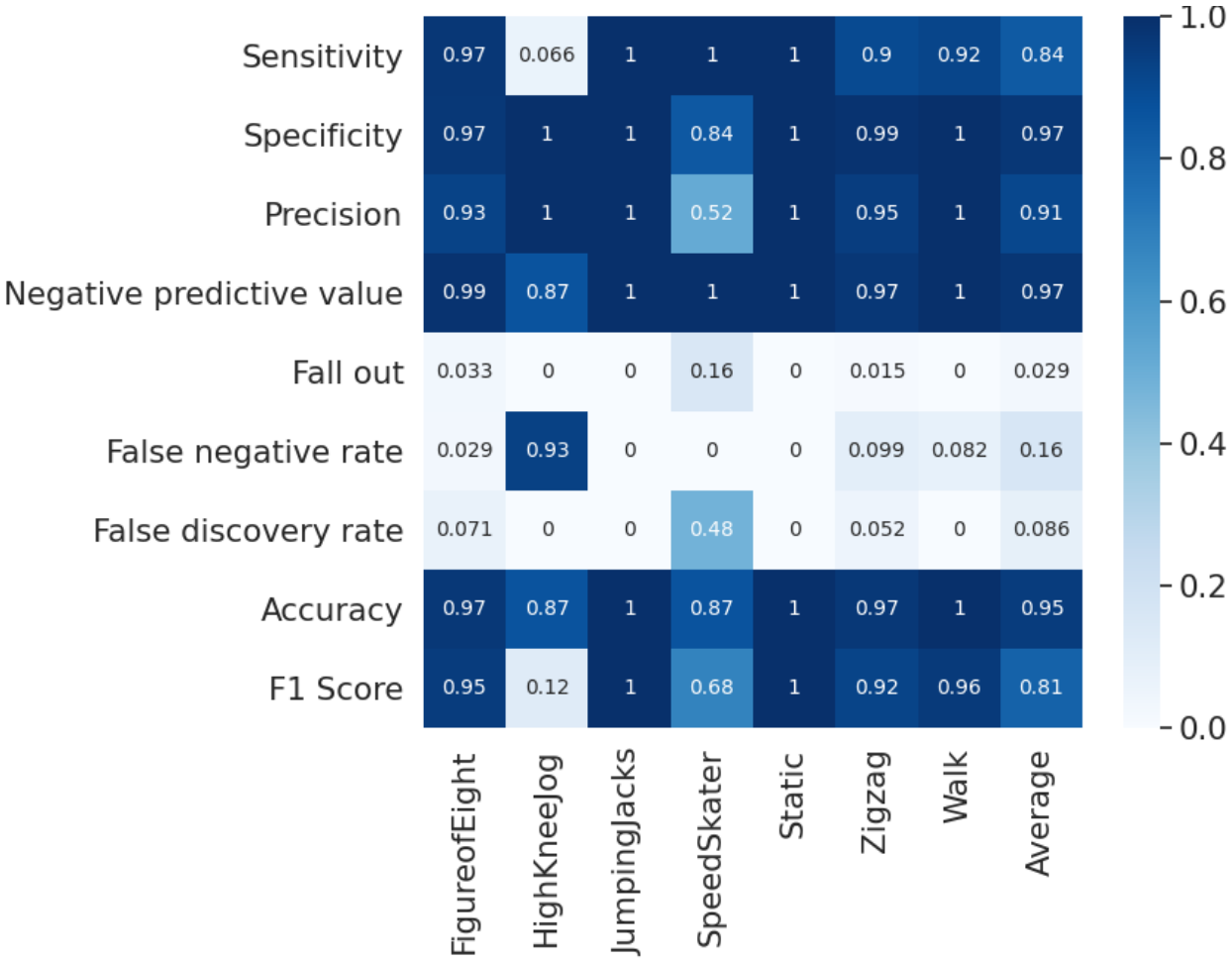
Matriz de confusión 8. *Sujeto S09*



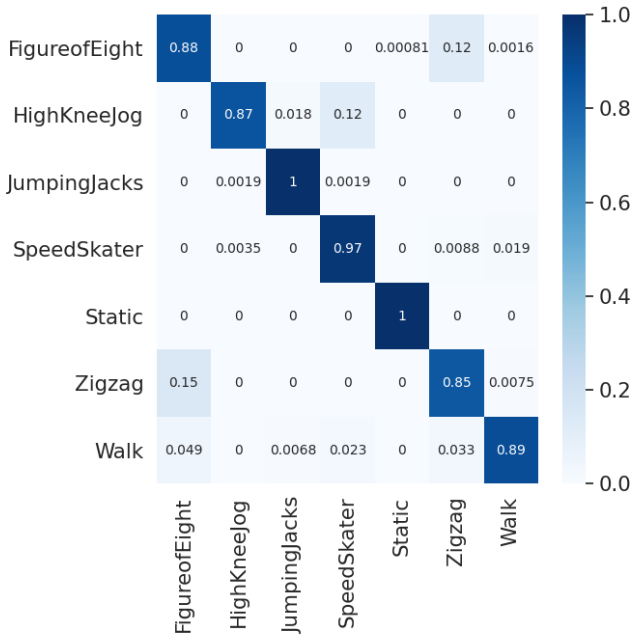
Métricas de rendimiento 8. *S09*



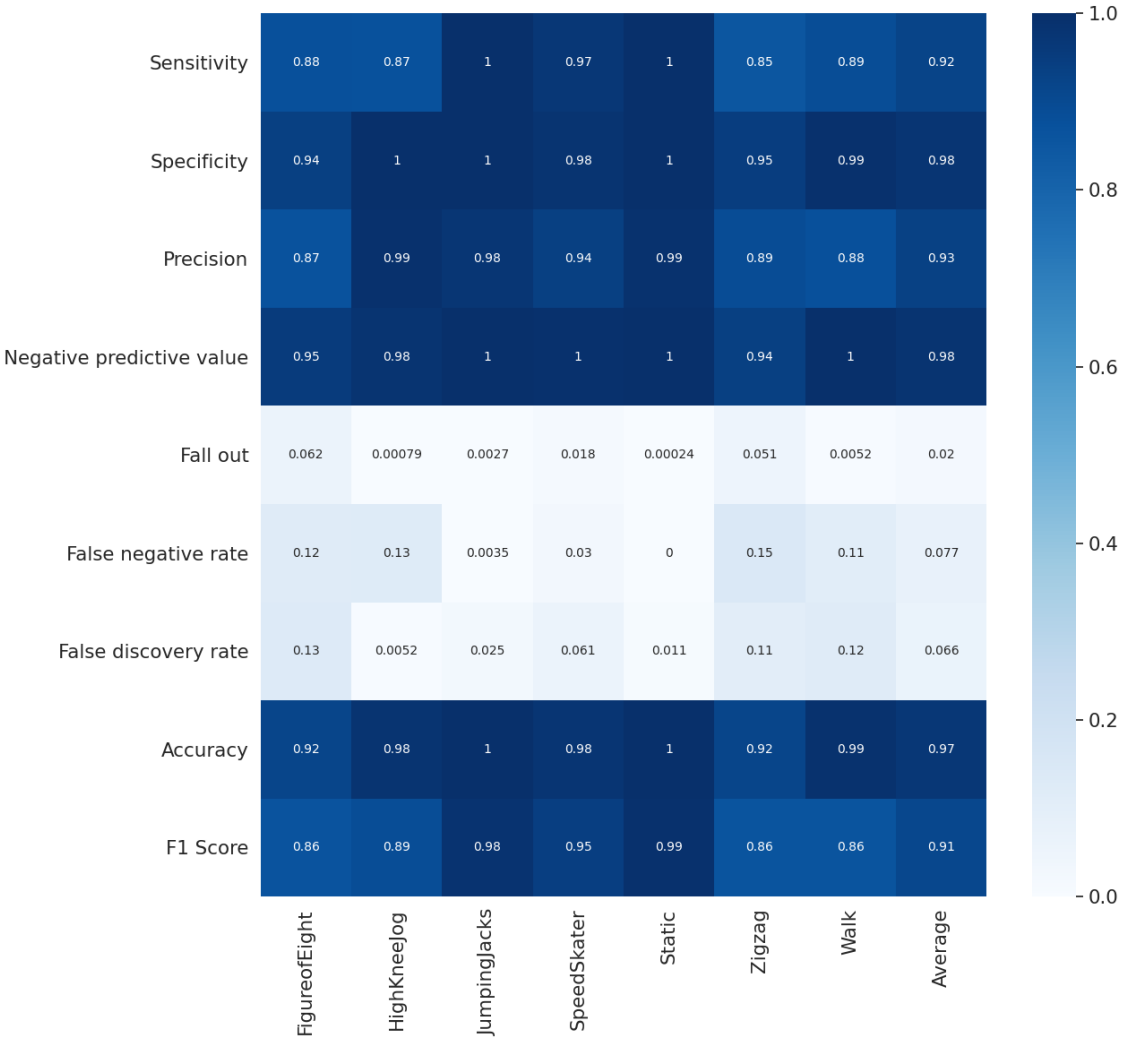
Matriz de confusión 9. *Sujeto S10*



Métricas de rendimiento 9. *Sujeto S10*

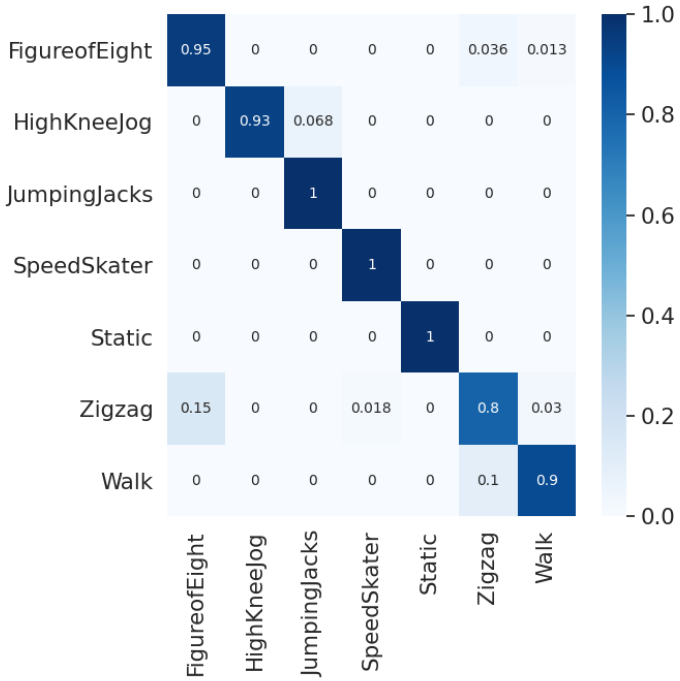


Matriz de confusión 10. *Agregada 8-fold*

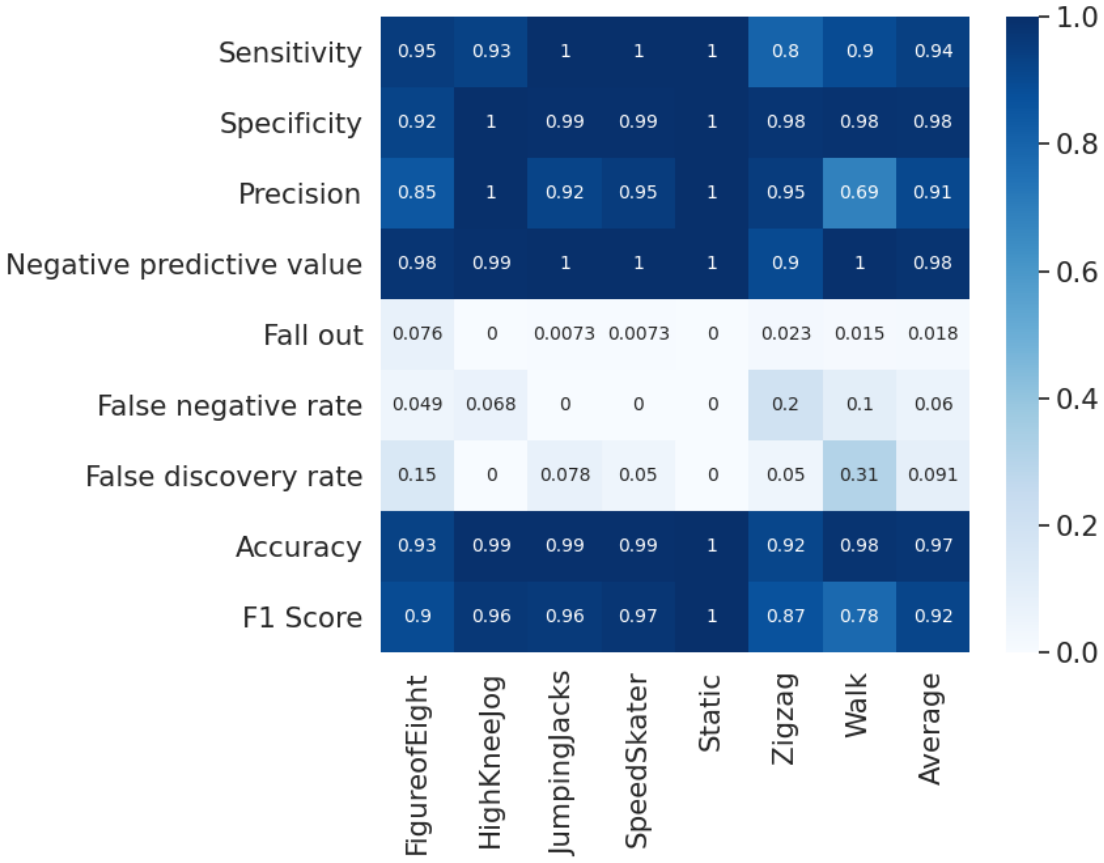


Métricas de rendimiento 10. *Agregada 8-fold*

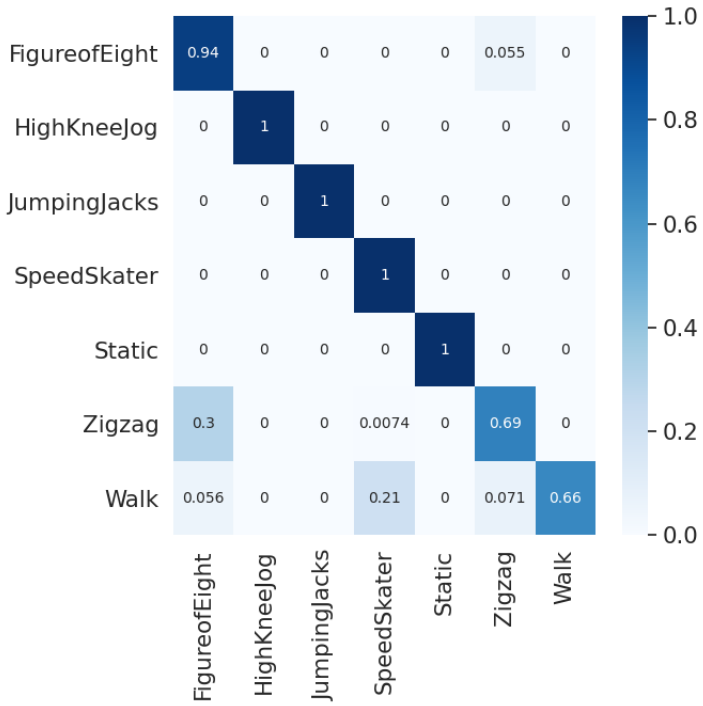
3.2 Modelo CNN+LSTM-4



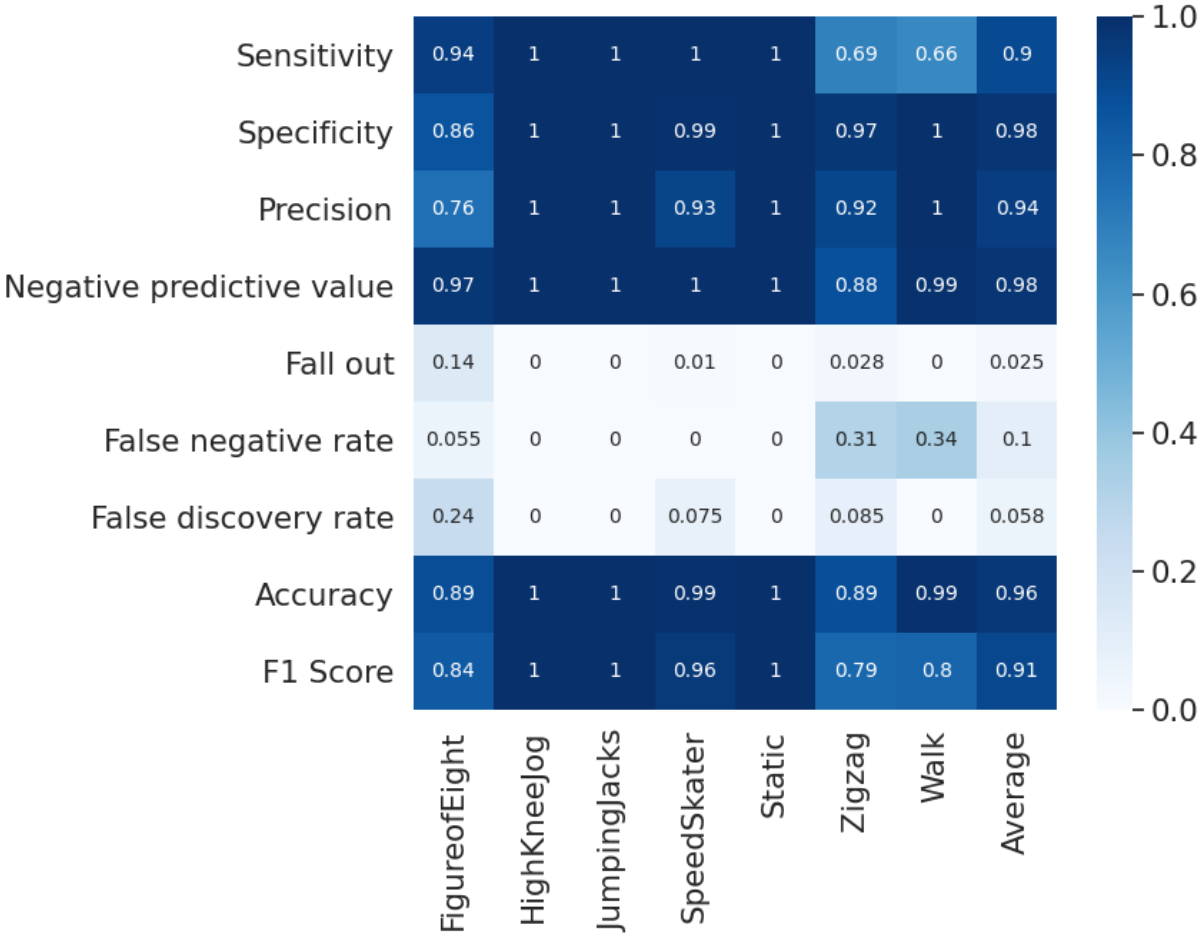
Matriz de confusión 11. *Sujeto S01*



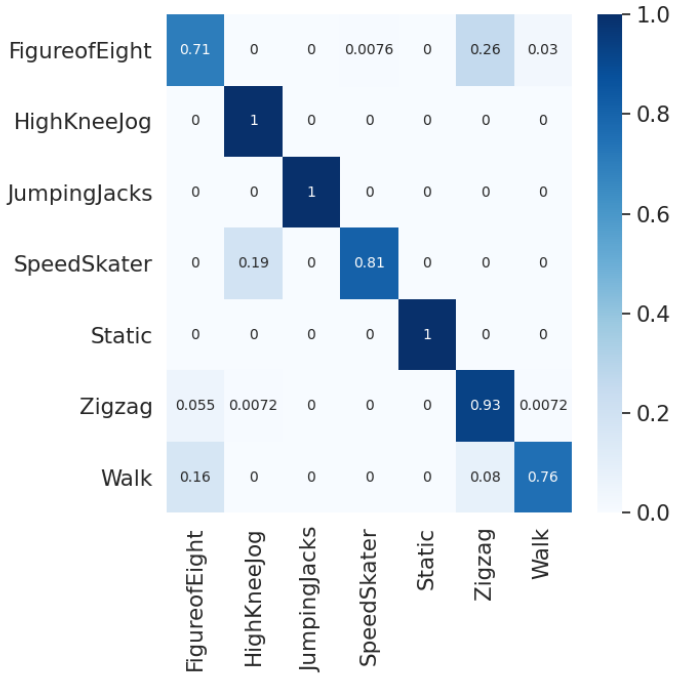
Métricas de rendimiento 11*. Sujeto S01*



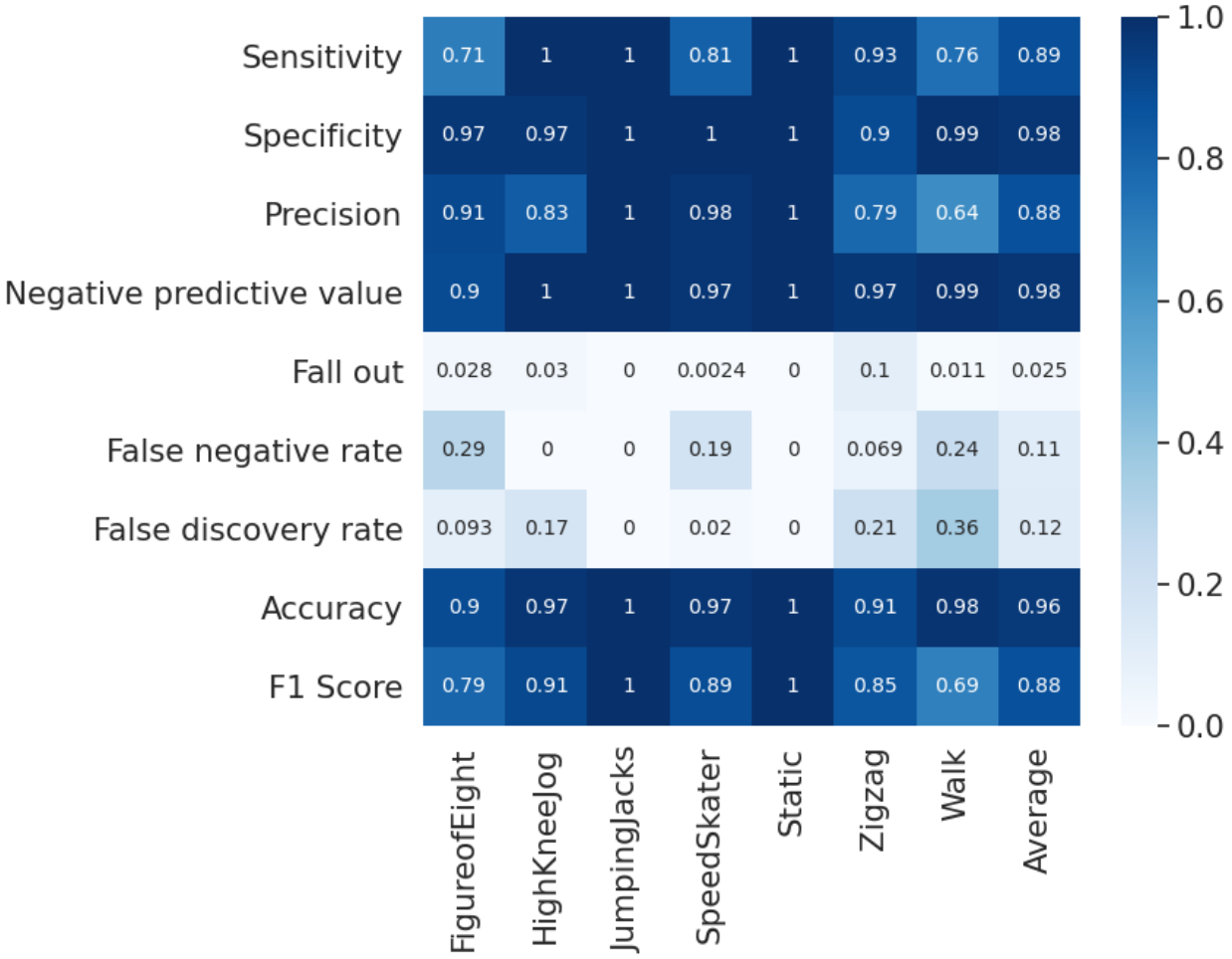
Matriz de confusión 12. *Sujeto S02*



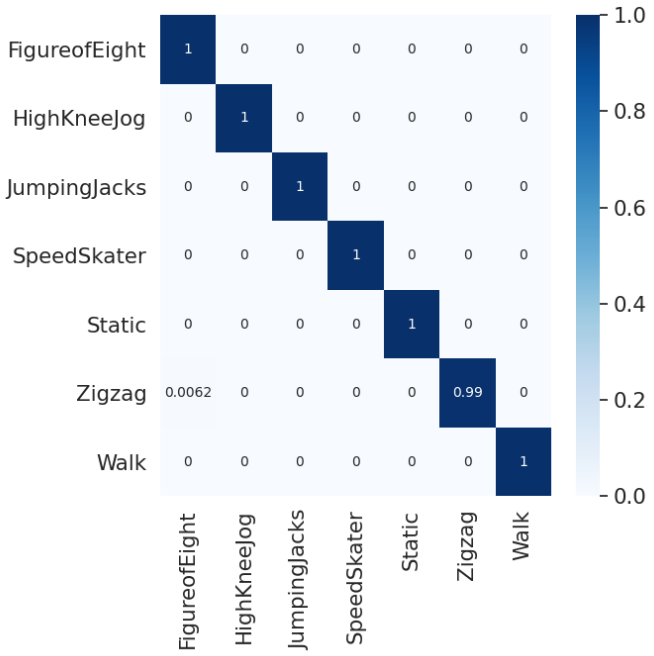
Métricas de rendimiento 12. *Sujeto S02*



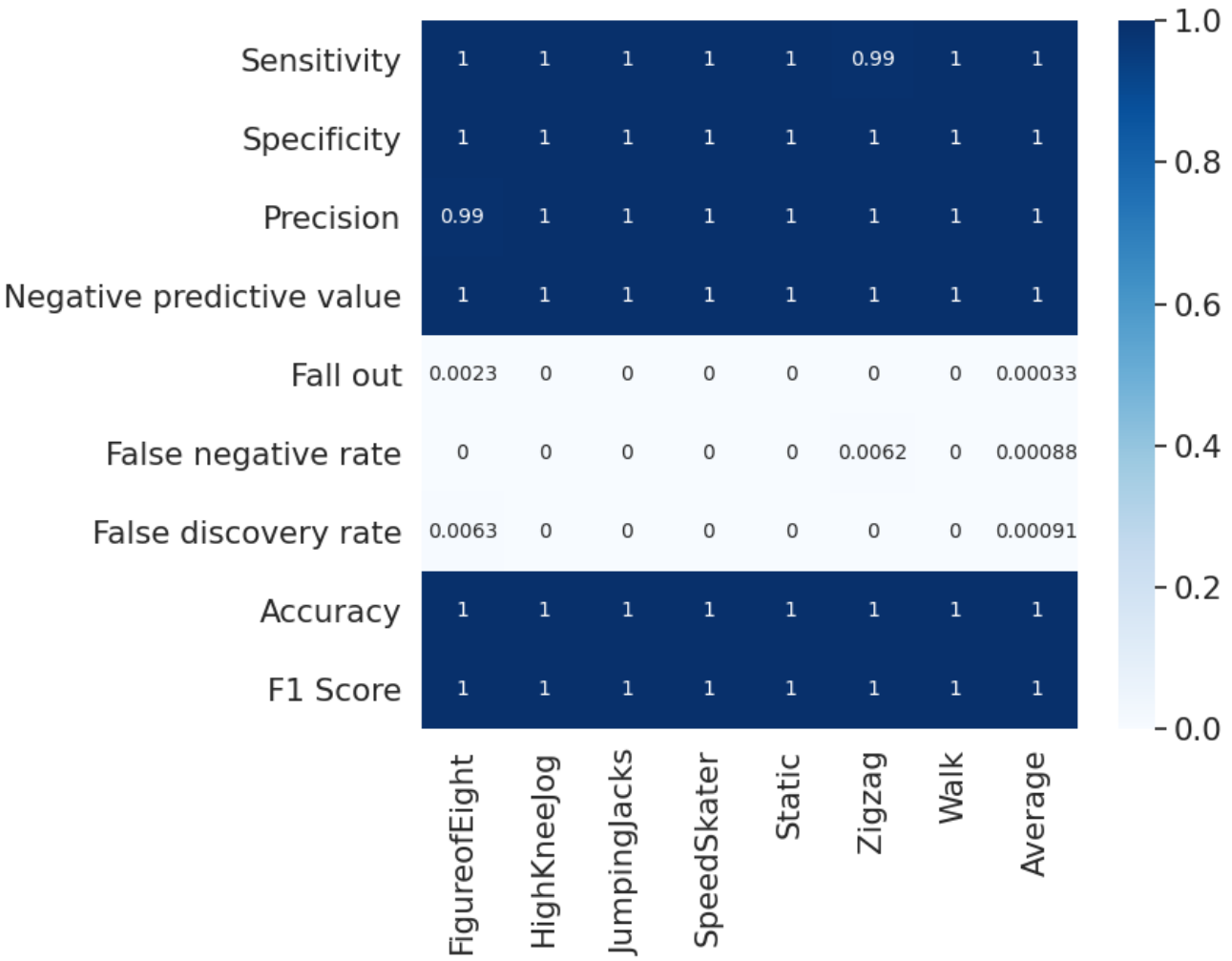
Matriz de confusión 13. *Sujeto S04*



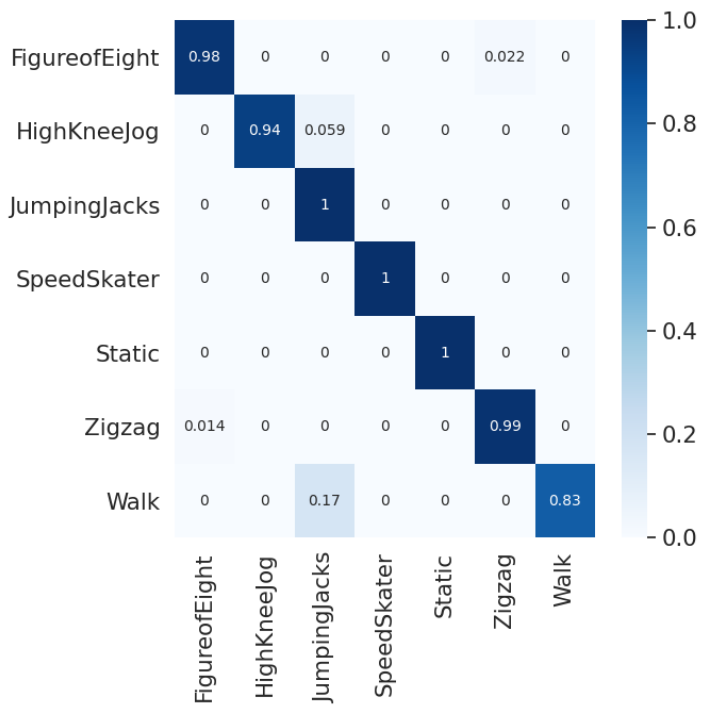
Métricas de rendimiento 13. *Sujeto S04*



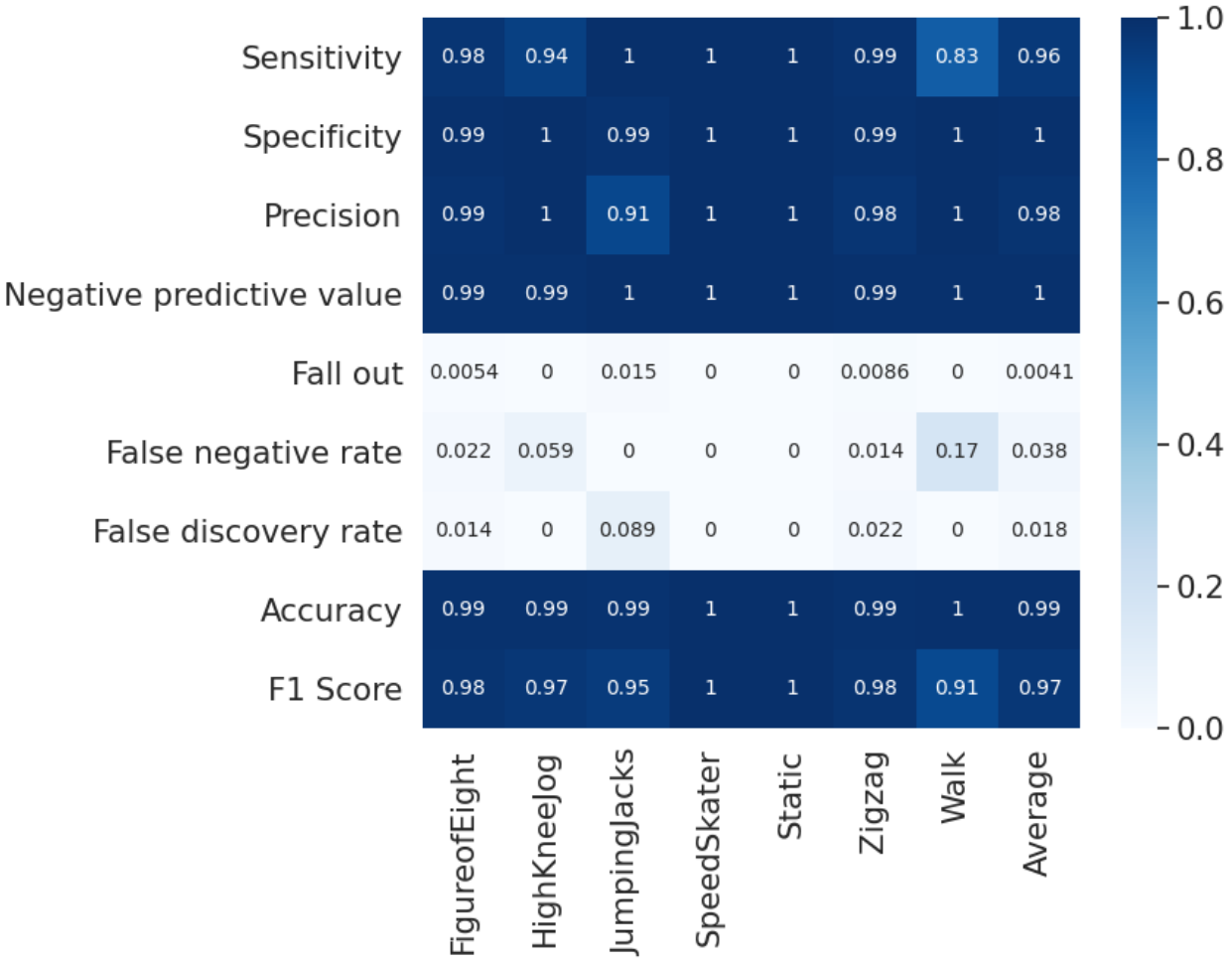
Matriz de confusión 14. *Sujeto S05*



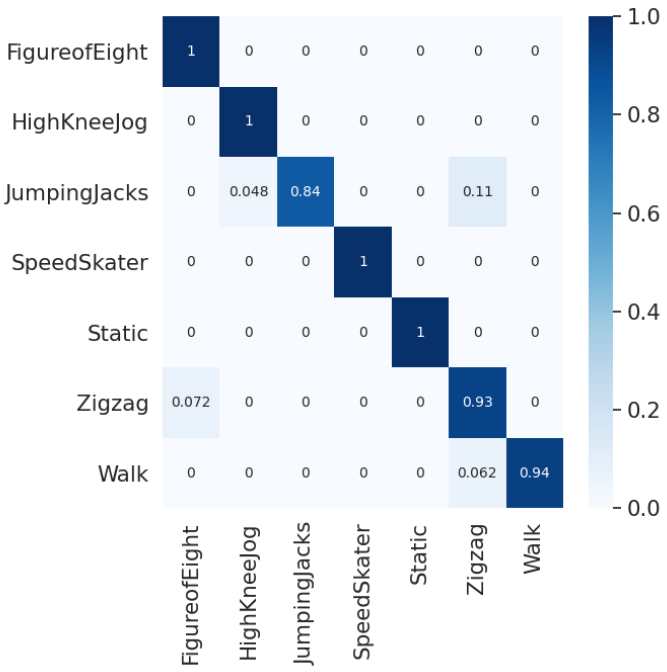
Métricas de rendimiento 14. *Sujeto S05*



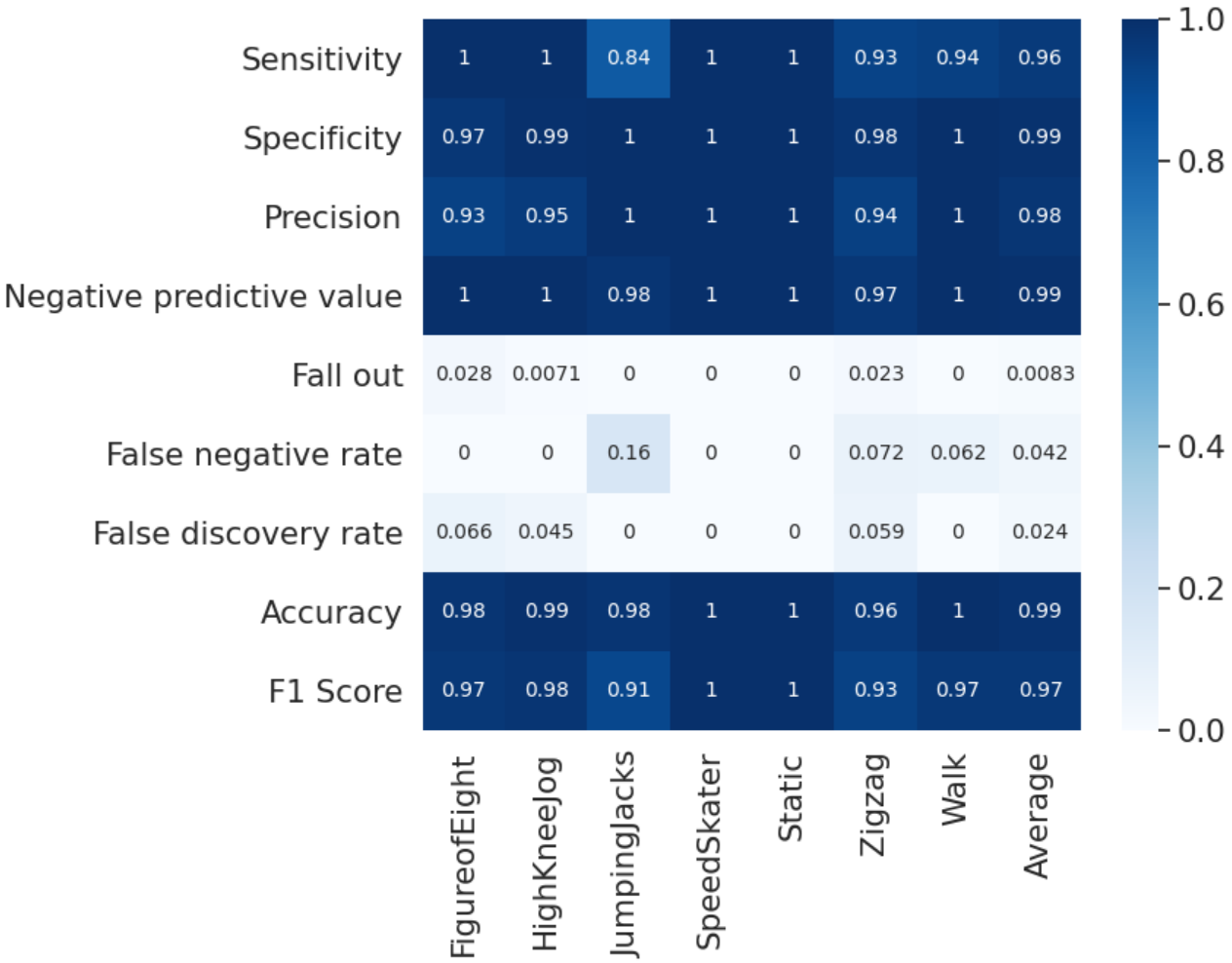
Matriz de confusión 15. *Sujeto S06*



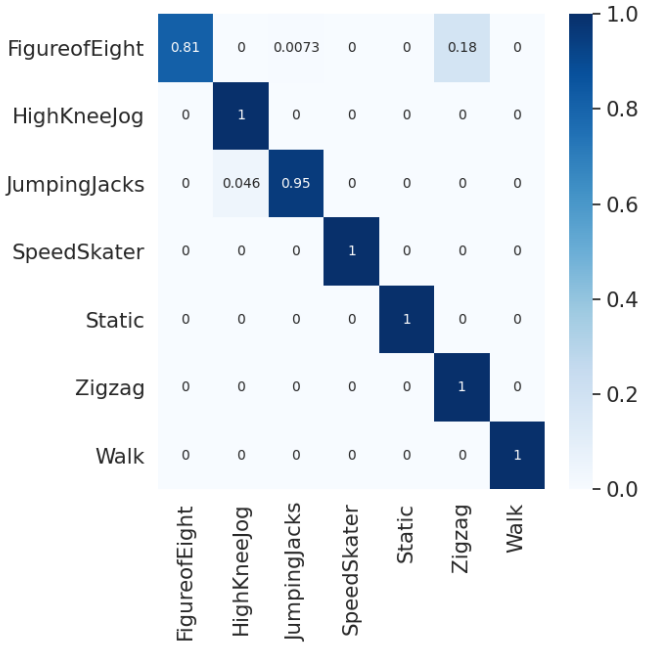
Métricas de rendimiento 15. *Sujeto S06*



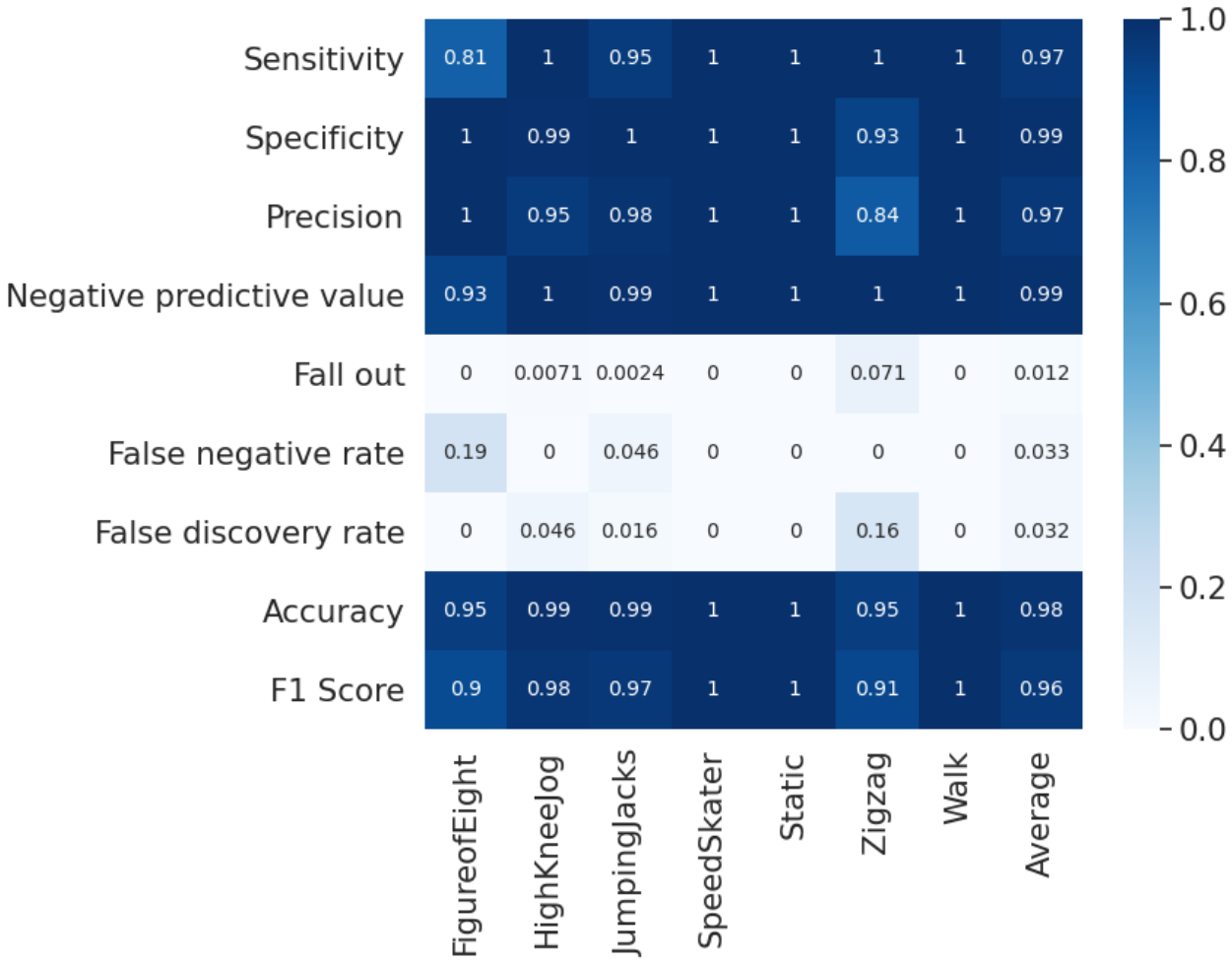
Matriz de confusión 16. *Sujeto S07*



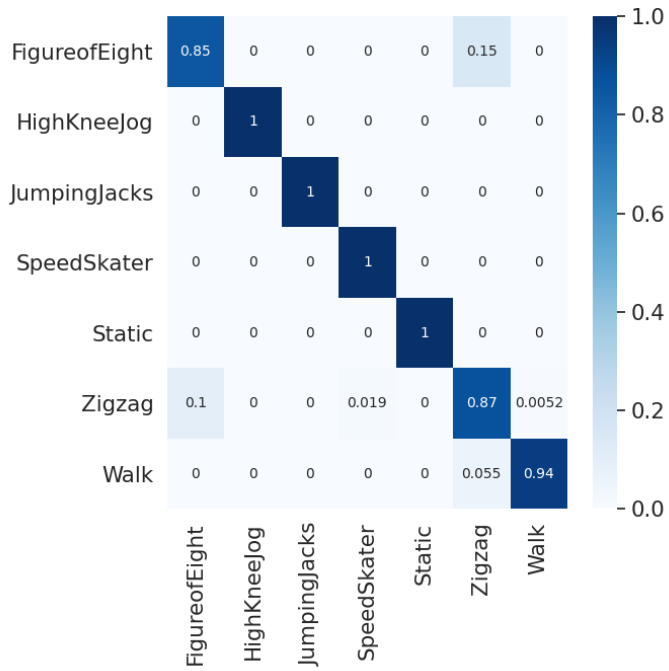
Métricas de rendimiento 16. *Sujeto S07*



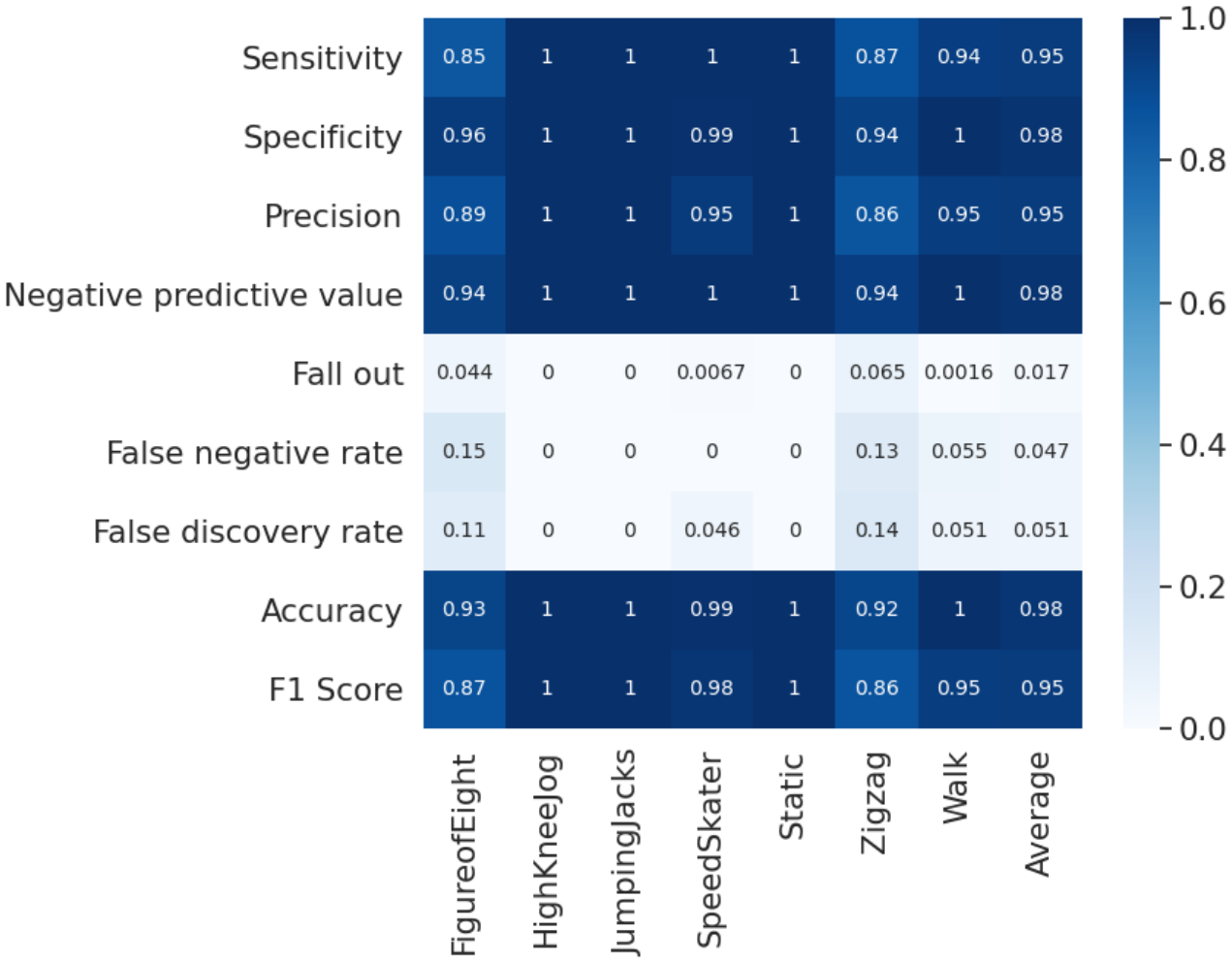
Matriz de confusión 17. *Sujeto S08*



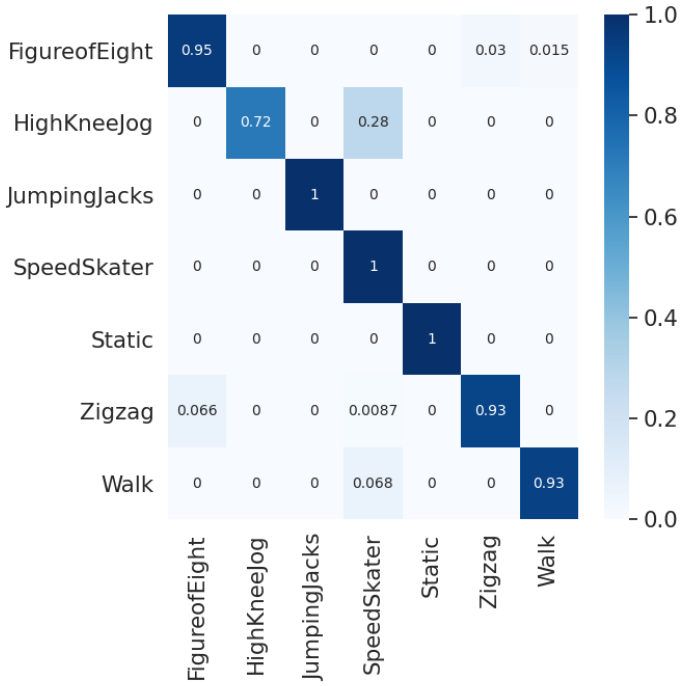
Métricas de rendimiento 17. *Sujeto S08*



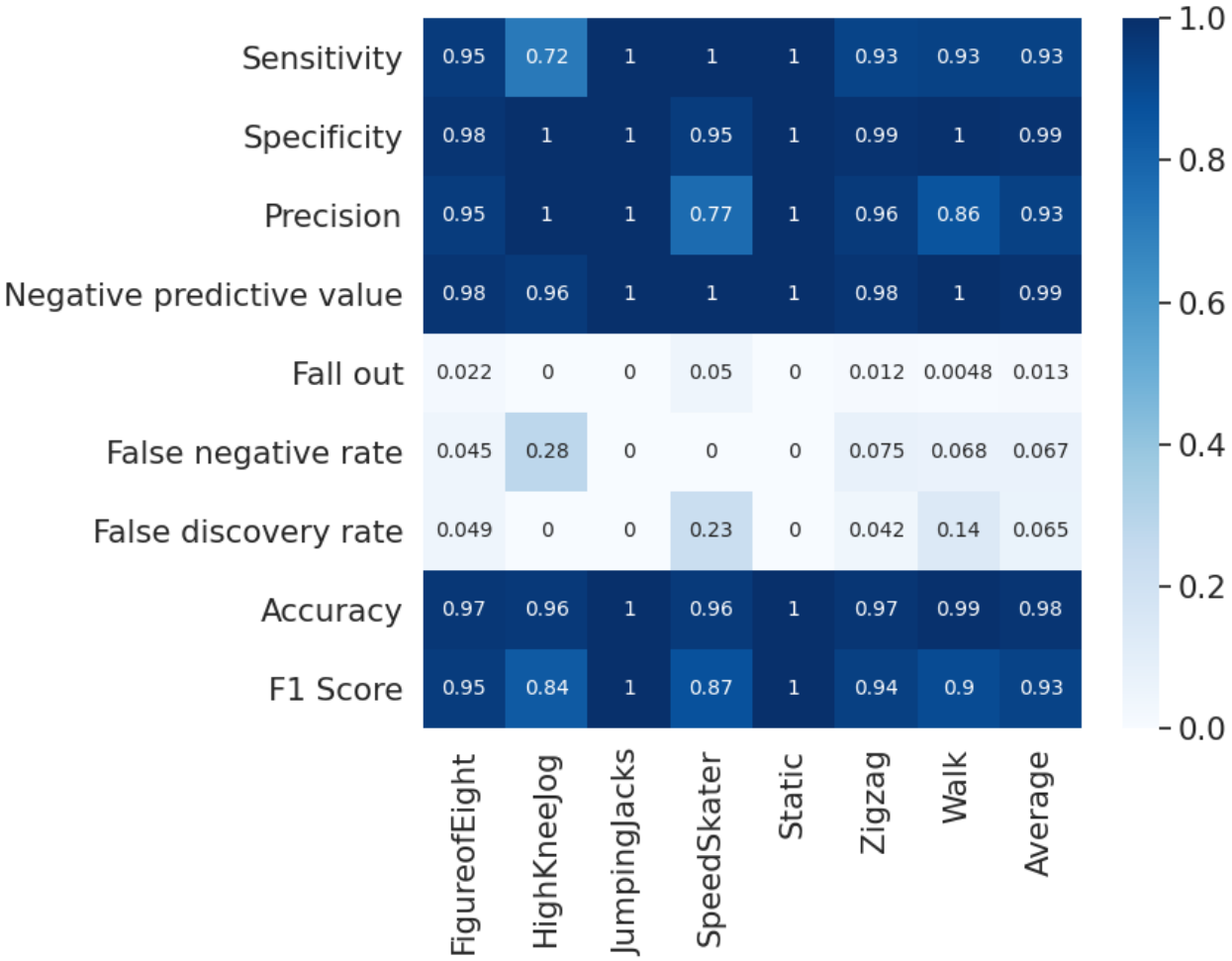
Matriz de confusión 18. *Sujeto S09*



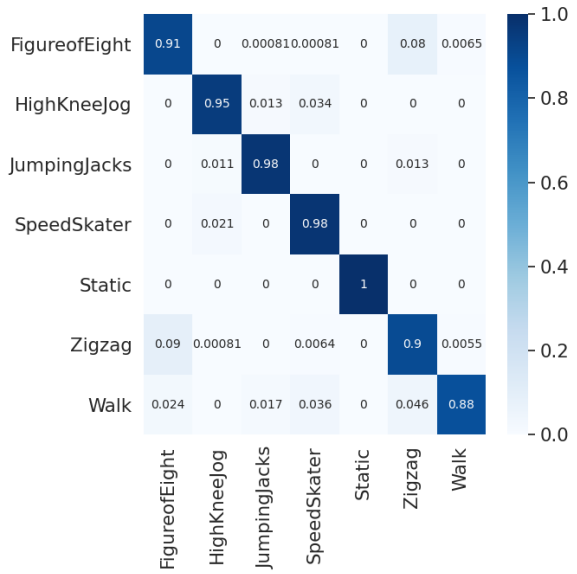
Métricas de rendimiento 18. *Sujeto S09*



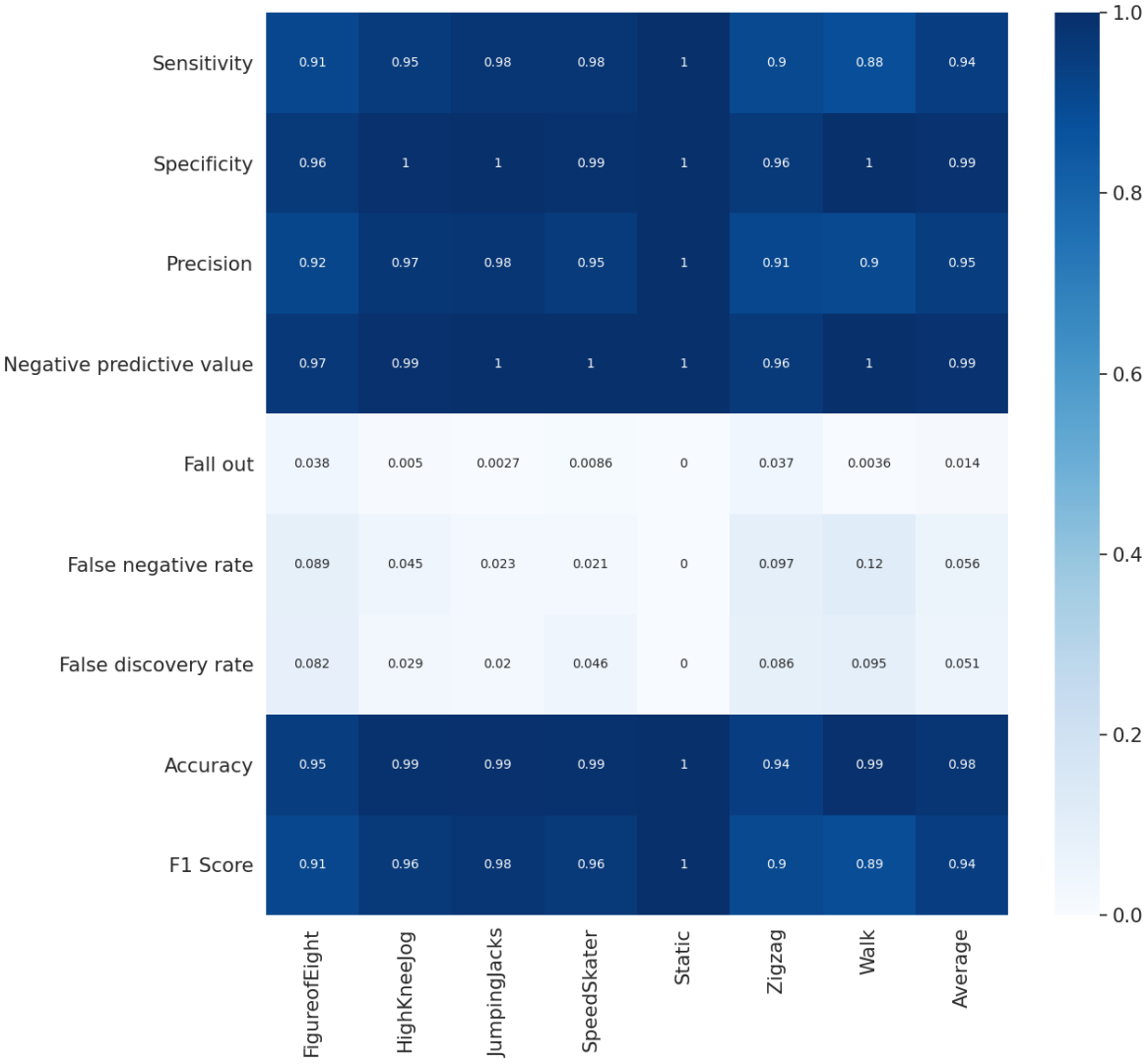
Matriz de confusión 19. *Sujeto S10*



Métricas de rendimiento 19. *Sujeto 10*



Matriz de confusión 20. *Agregada 8-fold*



Métricas de rendimiento 20. *Agregadas 8-fold*