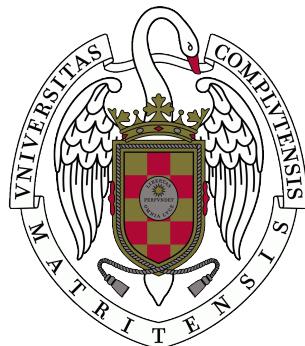

**Análisis de captura de movimiento para
inferencia de la comunicación no verbal**
**Motion capture analysis for inference of
non-verbal communication**



**Trabajo de Fin de Grado
Curso 2024–2025**

Autor

Alejandro Barrachina Argudo, Pablo Sánchez Martín

Director

Alejandro Romero Hernández, Ismael Sagredo Olivenza

Colaborador

**Grado en Ingeniería Informática y Grado en Desarrollo
de Videojuegos**

Facultad de Informática

Universidad Complutense de Madrid

Análisis de captura de movimiento para inferencia de la comunicación no verbal Motion capture analysis for inference of non-verbal communication

Trabajo de Fin de Grado en Ingeniería Informática y Grado en Desarrollo de Videojuegos

Autor

Alejandro Barrachina Argudo, Pablo Sánchez Martín

Director

Alejandro Romero Hernández, Ismael Sagredo Olivenza

Colaborador

Convocatoria: Junio 2025

Grado en Ingeniería Informática y Grado en Desarrollo de Videojuegos

Facultad de Informática

Universidad Complutense de Madrid

15 de mayo de 2025

Dedicatoria

*A Pedro Pablo y Marco Antonio, por crear
TeXiS e iluminar nuestro camino*

Agradecimientos

A mis padres y mi pareja, por estar siempre a mi lado y apoyarme en todo momento. A mis juntas anteriores de LAG, por acompañarme estos años. A la gente que conocí por LAG, por haberme apoyado en los proyectos de la asociación. A Poletti y Pascal, por haberme dado la oportunidad de hacer charlas en la facultad. A la gente de cafetería, por cuidarnos tan bien a todos. A mi gato Umbreon, por ser mi compañero de programación, y por haber escrito casi más líneas de código que yo. Y a todos los voluntarios que vinieron a hacer las pruebas para generar datos, sin ellos este trabajo no hubiera obtenido los resultados que se han conseguido.

Alejandro Barrachina Argudo

Resumen

Análisis de captura de movimiento para inferencia de la comunicación no verbal

Un resumen en castellano de media página, incluyendo el título en castellano. A continuación, se escribirá una lista de no más de 10 palabras clave.

Palabras clave

Máximo 10 palabras clave separadas por comas

Abstract

Motion capture analysis for inference of non-verbal communication

An abstract in English, half a page long, including the title in English. Below, a list with no more than 10 keywords.

Keywords

10 keywords max., separated by commas.

Índice

1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	2
1.3. Plan de trabajo	2
2. Estado de la Cuestión	3
2.1. Entornos virtuales	3
2.2. Comunicación no verbal en entornos virtuales	4
2.3. Captura de movimiento	4
2.4. Motores de videojuegos	6
2.5. Modelos de Inteligencia Artificial (IA)	6
2.5.1. Tecnologías de IA	7
2.5.2. Librerías de IA	7
2.5.2.1. Tensorflow	7
2.5.2.2. PyTorch	7
2.5.2.3. Scikit-learn	8
2.5.2.4. YDF	8
2.5.3. Modelos de reconocimiento de gestos y poses	8
2.5.3.1. YoLoV11	8
2.5.3.2. Modelos para la detección del balance de personas andando	9
2.5.3.3. Otros modelos de detección de gestos y poses	9
3. Descripción del Trabajo	11
3.1. Traje	11
3.1.1. Axis Studio y conexión del traje	11
3.1.2. Unity y la conexión del traje	13
3.1.3. Problemas encontrados con el traje	14
3.2. Busqueda de un dataset	14
3.2.1. Dataset de la Universidad Carnegie Mellon	14
3.2.2. Kaggle	16
3.3. Dataset Artificial	17

3.3.1.	Herramienta para descargarse animaciones de Mixamo de forma automática	17
3.3.2.	Carga mediante asset bundles	18
3.3.3.	Carga mediante Asset Database	19
3.3.4.	Tratamiento de CSV resultantes	21
3.4.	Dataset real	22
3.4.1.	Funcionamiento de la herramienta	22
3.4.2.	Recogida de datos	22
3.5.	Modelos de IA	25
3.5.1.	Interfaz de entrenamiento y seguimiento	26
3.5.2.	Carga de datos	28
3.5.3.	Modelos usados	29
3.5.3.1.	Long Short-Term Memory (LSTM)	29
3.5.3.2.	Convolutional Neural Network (CNN)	32
3.5.3.3.	Recurrent Neural Network (RNN)	34
3.5.3.4.	Random Forest	36
3.5.4.	TensorFlow Serving	37
3.5.5.	Hardware, complejidad de datos y otras limitaciones	38
3.6.	Aplicación final	38
3.6.1.	Funcionamiento de la aplicación	39
4.	Conclusiones y Trabajo Futuro	43
4.1.	Conclusiones	43
4.2.	Trabajo Futuro	43
Introduction		45
4.3.	Motivation	46
4.4.	Objectives	46
4.5.	Work Plan	46
Conclusions and Future Work		47
4.6.	Conclusions	47
4.7.	Future Work	47
Contribuciones Personales		49
Bibliografía		53
A. Cabecera de los Comma Separated Valuess (CSVs) del dataset		57
B. Carteles para la generación del dataset		61
C. Gráficos de la generación del dataset		65
D. Resultados de los distintos modelos LSTM		69

D.1. Intervalo 0.2s	69
D.2. Intervalo 0.4s	70
D.3. Intervalo 0.6s	71
D.4. Intervalo 0.8s	72
D.5. Intervalo 1.0s	73
E. Resultados de los distintos modelos RNN	75
E.1. Intervalo 0.2s	75
E.2. Intervalo 0.4s	76
E.3. Intervalo 0.6s	77
E.4. Intervalo 0.8s	78
E.5. Intervalo 1.0s	79
F. Resultados de los distintos modelos CNN	81
F.1. Intervalo 0.2s	81
F.2. Intervalo 0.4s	83
F.3. Intervalo 0.6s	84
F.4. Intervalo 0.8s	86
F.5. Intervalo 1.0s	87
G. Resultados de los distintos modelos Random Forest	89
G.1. Intervalo 0.2s	89
G.2. Intervalo 0.4s	90
G.3. Intervalo 0.6s	90
G.4. Intervalo 0.8s	91
G.5. Intervalo 1.0s	91
Glosario	93

Índice de figuras

2.1. Imágen del traje de captura de movimiento XSens, Fuente: https://www.researchgate.net/figure/The-Xsens-MVN-full-body-motion-capture-suit-Picfig1_233926874	5
3.1. Captura de la aplicación Axis Studio antes de conectar un traje	12
3.2. Captura del botón para conectar el traje a Axis Studio	12
3.3. Captura del botón para calibrar el traje	13
3.4. Captura del panel de configuración de Axis Studio	13
3.5. Imágen del traje de captura de movimiento usado para la base de datos de la Universidad Carnegie Mellon, Fuente: https://mocap.cs.cmu.edu/info.php	15
3.6. Imágen del esqueleto resultante del traje de captura de movimiento usado por la Universidad Carnegie Mellon, Fuente: https://mocap.cs.cmu.edu/info.php	15
3.7. Herramienta Mixamo Downloader Arreglada	18
3.8. Frame de un <i>timelapse</i> de la ejecución de la herramienta	19
3.9. Diagrama de flujo de la herramienta que carga y ejecuta las animaciones de Mixamo en tiempo de ejecución, llamada Mixamo Dumper .	20
3.10. Diagrama de flujo del proceso de limpieza de datos (funciones data_cleaner y data_cleaner_aux de la clase DataLoader)	21
3.11. Diagrama de la clase DataLoader	21
3.12. Diagrama de la conexión entre los distintos componentes de la herramienta, llamada Mocap Dumper	22
3.13. Fotografía de un usuario en las pruebas	24
3.14. Distribución de los datos brutos	25
3.15. Distribución de los datos estandarizados	25
3.16. Interfaz de entrenamiento	27
3.17. Interfaz de TensorBoard	28
3.18. Estructura del archivo de la interfaz	29
3.19. Estructura de la clase LSTMTrainer	30
3.20. Esquema del modelo LSTM	31
3.21. Matriz de confusión del modelo LSTM	31
3.22. Gráfico de entrenamiento del modelo LSTM	32

3.23. Estructura de la clase CNNTrainer	32
3.24. Esquema del modelo CNN	33
3.25. Matriz de confusión del modelo CNN	33
3.26. Gráfico de entrenamiento del modelo CNN	34
3.27. Estructura de la clase RNNTrainer	35
3.28. Esquema del modelo RNN	35
3.29. Matriz de confusión del modelo RNN	35
3.30. Gráfico de entrenamiento del modelo RNN	36
3.31. Estructura de la clase RFTrainer	36
3.32. Matriz de confusión del modelo RandomForest	37
3.33. Ejemplo de petición a TensorFlow Serving	37
3.34. Ejemplo de respuesta de TensorFlow Serving	38
3.35. Captura de la aplicación final desde el editor de Unity	39
3.36. Diagrama de flujo de la aplicación de demostración	41
B.1. Cartel colgado en la facultad de informática	61
B.2. Cartel colgado en redes sociales	62
B.3. Cartel colgado en pantallas de la facultad	63
C.1. Distribución de género de los encuestados	65
C.2. Distribución de edad de los encuestados	66
C.3. Distribución de nacionalidad de los encuestados	66
C.4. Distribución de idiomas hablados de los encuestados	67
C.5. Distribución de mano dominante de los encuestados	67
D.1. Esquema del modelo LSTM con 0.2s de intervalo	69
D.2. Matriz de confusión del modelo LSTM con 0.2s de intervalo	69
D.3. Gráfico de entrenamiento del modelo LSTM con 0.2s de intervalo (mejor val_accuracy = 0.5655)	70
D.4. Esquema del modelo LSTM con 0.4s de intervalo	70
D.5. Matriz de confusión del modelo LSTM con 0.4s de intervalo	70
D.6. Gráfico de entrenamiento del modelo LSTM con 0.4s de intervalo (mejor val_accuracy = 0.5462)	70
D.7. Esquema del modelo LSTM con 0.6s de intervalo	71
D.8. Matriz de confusión del modelo LSTM con 0.6s de intervalo	71
D.9. Gráfico de entrenamiento del modelo LSTM con 0.6s de intervalo (mejor val_accuracy = 0.5301)	71
D.10. Esquema del modelo LSTM con 0.8s de intervalo	72
D.11. Matriz de confusión del modelo LSTM con 0.8s de intervalo	72
D.12. Gráfico de entrenamiento del modelo LSTM con 0.8s de intervalo (mejor val_accuracy = 0.5912)	72
D.13. Esquema del modelo LSTM con 1.0s de intervalo	73
D.14. Matriz de confusión del modelo LSTM con 1.0s de intervalo	73
D.15. Gráfico de entrenamiento del modelo LSTM con 1.0s de intervalo (mejor val_accuracy = 0.5318)	73

E.1. Esquema del modelo RNN con 0.2s de intervalo	75
E.2. Matriz de confusión del modelo RNN con 0.2s de intervalo	75
E.3. Gráfico de entrenamiento del modelo RNN con 0.2s de intervalo (mejor val_accuracy = 0.3486)	76
E.4. Esquema del modelo RNN con 0.4s de intervalo	76
E.5. Matriz de confusión del modelo RNN con 0.4s de intervalo	76
E.6. Gráfico de entrenamiento del modelo RNN con 0.4s de intervalo (mejor val_accuracy = 0.4535)	76
E.7. Esquema del modelo RNN con 0.6s de intervalo	77
E.8. Matriz de confusión del modelo RNN con 0.6s de intervalo	77
E.9. Gráfico de entrenamiento del modelo RNN con 0.6s de intervalo (mejor val_accuracy = 0.41185)	77
E.10. Esquema del modelo RNN con 0.8s de intervalo	78
E.11. Matriz de confusión del modelo RNN con 0.8s de intervalo	78
E.12. Gráfico de entrenamiento del modelo RNN con 0.8s de intervalo (mejor val_accuracy = 0.47865)	78
E.13. Esquema del modelo RNN con 1.0s de intervalo	79
E.14. Matriz de confusión del modelo RNN con 1.0s de intervalo	79
E.15. Gráfico de entrenamiento del modelo RNN con 1.0s de intervalo (mejor val_accuracy = 0.39377)	79
F.1. Esquema del modelo CNN con 0.2s de intervalo	81
F.2. Matriz de confusión del modelo CNN con 0.2s de intervalo	82
F.3. Gráfico de entrenamiento del modelo CNN con 0.2s de intervalo (mejor val_accuracy = 0.28777)	82
F.4. Esquema del modelo CNN con 0.4s de intervalo	83
F.5. Matriz de confusión del modelo CNN con 0.4s de intervalo	83
F.6. Gráfico de entrenamiento del modelo CNN con 0.4s de intervalo (mejor val_accuracy = 0.49473)	84
F.7. Esquema del modelo CNN con 0.6s de intervalo	84
F.8. Matriz de confusión del modelo CNN con 0.6s de intervalo	85
F.9. Gráfico de entrenamiento del modelo CNN con 0.6s de intervalo (mejor val_accuracy = 0.44952)	85
F.10. Esquema del modelo CNN con 0.8s de intervalo	86
F.11. Matriz de confusión del modelo CNN con 0.8s de intervalo	86
F.12. Gráfico de entrenamiento del modelo CNN con 0.8s de intervalo (mejor val_accuracy = 0.40583)	87
F.13. Esquema del modelo CNN con 1.0s de intervalo	87
F.14. Matriz de confusión del modelo CNN con 1.0s de intervalo	88
F.15. Gráfico de entrenamiento del modelo CNN con 1.0s de intervalo (mejor val_accuracy = 0.56002)	88
G.1. Matriz de confusión del modelo Random Forest con 0.2s de intervalo (mejor val_accuracy = 0.85735)	89

G.2. Matriz de confusión del modelo Random Forest con 0.4s de intervalo (mejor val_accuracy = 0.85735)	90
G.3. Matriz de confusión del modelo Random Forest con 0.6s de intervalo (mejor val_accuracy = 0.85233)	90
G.4. Matriz de confusión del modelo Random Forest con 0.8s de intervalo (mejor val_accuracy = 0.85936)	91
G.5. Matriz de confusión del modelo Random Forest con 1.0s de intervalo (mejor val_accuracy = 0.84681)	91

Índice de tablas

2.1. Puntos de referencia del modelo YOLOv11-pose	9
3.1. Cabecera del CSV de cada animación, en orden descendente y de izquierda a derecha (incompleta).	18
3.2. Reacción del Non Playable Character (NPC) al gesto predicho del jugador	40
A.1. Cabecera del CSV de cada animación, en orden descendente y de izquierda a derecha (completa)	57

Capítulo 1

Introducción

“La revolución industrial y sus consecuencias han sido un desastre para la raza humana”
— Theodore Kaczynski

En los últimos años se ha visto un gran interés y evolución de las tecnologías de realidad virtual a mano de empresas como Apple con su lanzamiento de las Apple Glasses o Meta con el lanzamiento de las Meta Quest 3 o su interés por el **metaverso** con su aplicación de Meta Horizon Worlds.

Es por ello que es interesante investigar la comunicación no verbal en entornos virtuales, ya no solo para aplicaciones con varios usuarios si no también para poder mejorar interacciones con **NPCs** en este tipo de entornos.

Nuestra propuesta en este Trabajo de Fin de Grado es una primera aproximación de cómo se puede usar la **IA** y la tecnología de captura de movimiento para lograr esa mejora en las interacciones en los mundos virtuales mediante la comunicación no verbal, siendo este caso la exploración de la capacidad de clasificar distintos gestos. Los gestos seleccionados han sido seis gestos que se han útiles a la hora de que un NPC pueda reconocerlo en un videojuego. Estos gestos han sido:

1. Bailar
2. Saludar
3. Señalar
4. Sentarse
5. Pelear
6. Correr

Para poder realizar este trabajo se ha requerido usar las gafas de realidad virtual Meta (antes conocidas como Oculus) Quest 2 y 3 y el traje de captura de movimiento Perception Neuron 3, de la empresa Noitom, como hardware y Python C# y Unity como los requisitos de software.

1.1. Motivación

Estudio del reconocimiento de gestos mediante IA para posibles implementaciones en el estudio y mejora de la comunicación no verbal en entornos virtuales.

1.2. Objetivos

Implementación de un modelo de IA que, con baja latencia, permita identificar el gesto que se esté realizando con un traje de captura de movimiento.

1.3. Plan de trabajo

Nuestro plan de trabajo consiste en varios pasos:

1. Búsqueda de un dataset: generar un dataset lo suficientemente grande con varios ejemplos de gestos como para poder entrenar de forma adecuada diferentes modelos.
2. Implementación de modelos de IA: implementación de varios modelos de IA para poder hacer una comparativa entre ellos y decidir cuál es el más adecuado teniendo en cuenta su velocidad de predicción y su precisión.
3. Desarrollo de una aplicación final: desarrollo de una aplicación para las Oculus Quest a forma de demo en la que se conecte al modelo elegido y se pueda ver en tiempo real su uso.

Capítulo 2

Estado de la Cuestión

En este capítulo se presenta el estado del arte de los distintos elementos que componen el trabajo. Estos campos son la comunicación no verbal en entornos virtuales, la captura de movimiento, distintos modelos de IA, motores de videojuegos (en concreto Unity) y el estado actual de los entornos virtuales.

2.1. Entornos virtuales

Como se puede ver en el artículo [Ellis \(1991\)](#) se empezó a hablar en la década de los 90 de entornos virtuales y el [metaverso](#) como nuevas formas de comunicarse con las personas. Estos entornos virtuales al principio eran muy sencillos, con pocas actividades que realizar y el chat escrito como única forma de comunicarse. Algún ejemplo de estos entornos virtuales eran el Second Life o el Habbo Hotel.

Pero con la aparición de la [Realidad Virtual \(VR\)](#) se hicieron populares entornos virtuales focalizados en esta tecnología, ya que ofrecía un grado de interacción superior a los anteriores mencionados. Junto a esto se empieza a hablar de “inmersión” y “presencia”: dos términos que, a menudo se usan como sinónimos, pero son ligeramente distintos aunque están muy relacionados. Como lo define el artículo [Wilkinson et al. \(2021\)](#), la presencia es la calidad experimental en entornos virtuales mientras que la inmersión está asociado con los aspectos técnicos de un sistema virtual que ayudan al usuario a potenciar la sensación de presencia. Estudios recientes como [Van Brakel et al. \(2023\)](#) demuestran como la presencia, tanto la nuestra propia como social, son potenciadores del apoyo social percibido, lo que se asocia con el bienestar de los usuarios; mientras que el estudio [Pallavicini et al. \(2019\)](#) muestra que, aunque no haya mejoras performáticas entre jugar en entornos inmersivos ([VR](#)) y no inmersivos (un ordenador), los jugadores encontraron más emocionantes y con mayor presencia los entornos interactivos.

Con todo esto es natural que las empresas inviertan en tecnologías que aumenten la sensación de inmersión para lograr una mayor presencia. Empresas como Apple o Meta están apostando en la tecnología [VR](#), sacando recientemente las gafas Apple Vision Pro (2024) y Meta Quest 3 (2023); mejorando aspectos como la calidad de las cámaras, el sonido, el reconocimiento de gestos y su comodidad que pueden

amplificar la sensación de inmersión cuando se utilizan. Además, Meta está centrada en la idea de [metaverso](#) como un espacio digital en el que se puede socializar.

Con la idea de mejorar la inmersión en entornos virtuales se ideó este proyecto en el que, mediante la captura de movimiento, se puede usar la comunicación no verbal para expresar ciertas acciones y el entorno sea capaz de reconocer lo que estamos haciendo. Todo esto pensado para que se pueda usar en entornos de [VR](#) para llevar la inmersión al máximo

2.2. Comunicación no verbal en entornos virtuales

En los últimos años se han llevado a cabo distintos estudios sobre la comunicación no verbal en entornos virtuales. El estudio [Xenakis et al. \(2023\)](#) se centra en el concepto de presencia en entornos virtuales, enfatizando los roles de inmersión, contenido emocional y fidelidad de avatares en la mejora de la experiencia del usuario. El estudio resalta como el realismo de los avatares y sus comportamientos (movimiento y lenguaje corporal) ayudan a conseguir un mayor sentimiento de presencia social entre los usuarios. También se discute como estos elementos influyen en las dinámicas interpersonales y la percepción del usuario en entornos virtuales.

Estos roles discutidos en el estudio pueden extrapolarse a la experiencia de jugar a videojuegos. En el contexto de los videojuegos tradicionales, las opciones de comunicación no verbal son las dadas por los desarrolladores en forma de gestos, acciones dentro del juego (agacharse, girar su avatar o mover la cabeza al mover la cámara por ejemplo). Dentro de cada videojuego la comunidad va a intentar usar las opciones que se les proporcione para comunicarse entre ellos mismos de la mejor manera posible.

Un ejemplo claro de evolución de la comunicación no verbal en videojuegos y su importancia para los jugadores es el caso de *VR chat*. *VR chat* es un videojuego [Massively Multiplayer Online Role-Playing Game \(MMORPG\)](#) de [VR](#) donde los jugadores pueden unirse a distintos mundos e interactuar con otros usuarios. Este juego no solo permite el uso de gafas de [VR](#), sino que también permite (de manera experimental) usar distintas formas de captura de movimiento de cuerpo completo ^{[1](#)}.

Los usuarios de esta aplicación han buscado distintas formas de conseguir una mayor expresividad en sus avatares para así poder entablar mejor conversaciones entre ellos o para poder expresarse como les gustaría. Algunos de estos métodos se describen en el siguiente apartado junto a métodos de captura de movimiento no orientados a videojuegos.

2.3. Captura de movimiento

La captura de movimiento es una técnica que permite digitalizar el movimiento de una persona. Esta técnica se usa sobre todo en el ámbito de la animación, el cine y

¹Enlace a la documentación de *VR Chat* para su [Software Development Kit \(SDK\)](#) de captura de movimiento de cuerpo completo [VRChat \(2024\)](#)

los videojuegos. Para lograr esto se han desarrollado diferentes tipos de tecnologías.

Los sistemas basados en cámaras 3D utilizan los **Rayos Infrarrojos (IRs)** para calcular la profundidad. Según el artículo [Zhang \(2012\)](#) la Kinect (Microsoft, 2010) utilizaba esta tecnología para ello. Consiste en un proyector de **IRs** cuyos rayos atraviesan una rejilla de difracción, convirtiéndose en un conjunto de puntos que, como se sabe la geometría relativa entre este proyector y una cámara de **IRs** que venía incorporada, se podía triangular la posición de un objeto haciendo coincidir un punto observado con uno de los puntos del proyector.

Otra tecnología usada es mediante sensores iniciales. Según se puede ver en [Roeftenberg et al. \(2009\)](#), el traje Xsens MVN es un ejemplo de ello. No utiliza cámaras o marcadores si no la combinación de giroscopios, acelerómetros y magnetómetros. Con los datos recibidos de estos sensores se puede estimar la rotación de los diferentes puntos mientras que la posición se puede estimar mediante supuestos de una cadena cinemática. En la figura 2.1 se puede ver el traje visto desde la espalda.

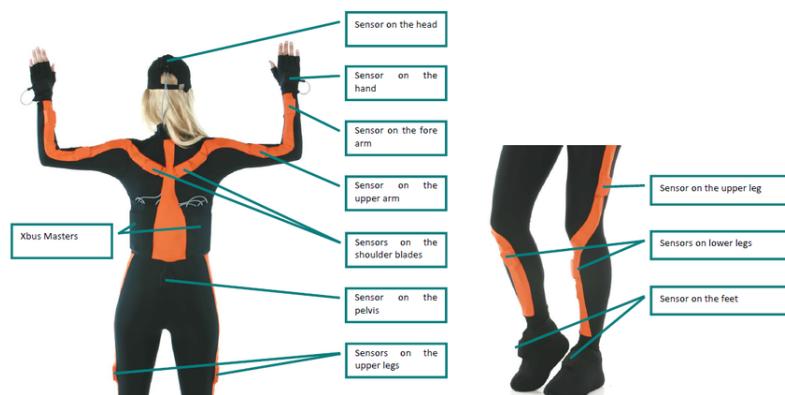


Figura 2.1: Imagen del traje de captura de movimiento XSens, Fuente: https://www.researchgate.net/figure/The-Xsens-MVN-full-body-motion-capture-suit-Picture-source-hthttp-wwwxsenscom_fig1_233926874

La tecnología que se ha usado en este trabajo consiste en la captura mediante sensores electromagnéticos. El artículo [Raab et al. \(1979\)](#) explica que la generación de valores en un campo electromagnético tridimensional es suficiente para determinar la posición y rotación de un objeto emitidor frente a un sensor. Para ello se aplican transformaciones lineales a la excitación de la fuente y los vectores recibidos por el receptor para percibir pequeños cambios en los valores a determinar, los cuales se separan mediante combinaciones lineales de los vectores de salida del sensor y se utilizan para actualizar las coordenadas previas.

En esta última tecnología se basa el traje Perception Neuron 3, el traje de captura de movimiento utilizado en este trabajo. Este traje fue creado por Noitom, empresa fundada en el 2012 enfocada en el desarrollo de tecnología de captación de movimiento. El traje previamente mencionado se encuentra dentro de la serie de trajes "Perception Neuron", los cuales son bandas de velcro a las que se pueden acoplar sensores electromagnéticos.

2.4. Motores de videojuegos

Un motor de videojuegos es un software que proporciona herramientas para desarrollar y ejecutar un videojuego. Estos motores están compuestos a su vez de otros motores más específicos que se encargan de funcionalidades básicas para un videojuego para que el desarrollador del juego no tenga que encargarse de ello. Algunos ejemplos de esto último son el motor de render, el de físicas o el de sonido.

Actualmente los motores de videojuegos más usados en el mercado son Unity, Unreal y Godot. Aunque el artículo [Holfeld \(2024\)](#) no se centre en un análisis del mercado en general, si contiene varias estadísticas sobre el uso de los principales motores de videojuegos en el mercado en 2023. Este artículo menciona que, de los juegos publicados en Steam ese año, un 60.04 % fueron hechos en Unity, un 16.42 % fueron hechos en Unreal y el 23.54 % restante fue hecho en motores propios de empresas (Source o CryEngine, por ejemplo).

Debido a su gran integración con otras plataformas el motor elegido para este estudio ha sido Unity.

Unity es un motor de videojuegos creado por Unity Technologies en 2005. A continuación se listan las principales herramientas que usa el motor:

1. Motor gráfico: Propio de Unity basado mayoritariamente en OpenGL (para depender de qué plataformas puede usar otro, como Direct3D para Windows).
2. Motor físico: PhysX, creado por la empresa Nvidia Corporation.
3. Motor de audio: Propio de Unity.
4. Lenguaje de scripting: C#.
5. Lenguaje de scripting de shaders: ShaderLab.

El modelo clásico de Unity se ha basado en la arquitectura Entidad-Componente (EC). Esta arquitectura deja a todo lo que hay en el juego (las entidades, llamadas en Unity "GameObjects") como contenedores de componentes, que son los que le aportan funcionalidad a éstas. Este modelo supuso una mejoría ante la programación de videojuegos con objetos, ya que conseguía independizar comportamientos.

Desde 2022 mediante el paquete Unity DOTS (Data-Oriented Technology Stack)² se pueden crear videojuegos siguiendo la arquitectura Entidad-Componente-Sistema (ECS).

2.5. Modelos de IA

La inteligencia artificial ha sido un tópico de desarrollo e investigación en los últimos años. Aunque el desarrollo en los últimos años se ha centrado ha centrado sobre todo en el ámbito de las IAs generativas, este estudio se va a centrar sobre todo en el uso de redes neuronales e IAs de clasificación más tradicionales.

²Enlace a la documentación del paquete DOTS: <https://docs.unity3d.com/Packages/com.unity.entities@1.3/manual/index.html>

2.5.1. Tecnologías de IA

En la actualidad, el mundo de la IA está centrado en el desarrollo en python mediante el uso de librerías programadas en C++. Las librerías más comunes son Tensorflow, Pytorch y scikit-learn.

Tensorflow tiene un enfoque más orientado a la producción y despliegue de modelos de IA a un nivel empresarial, pytorch es más usado en el ámbito académico y de investigación por su capacidad de rápido prototipado. Scikit-learn se usa más para enseñar los principios de la IA y desarrollar modelos más ligeros y más tradicionales.

Estas librerías se usan en conjunto con Numpy([Harris et al. \(2020\)](#)) y Pandas([pandas development team \(2020\)](#), [Wes McKinney \(2010\)](#)) para conseguir unos cálculos más efectivos sobre la gran cantidad de datos que se necesitan en el ciclo de vida de un modelo de IA.

2.5.2. Librerías de IA

En el apartado anterior ya se han nombrado distintas librerías de IA que se usan en la actualidad. En este apartado se van a describir las más relevantes para el desarrollo de este trabajo.

2.5.2.1. Tensorflow

Tensorflow([Abadi et al. \(2015\)](#)) es una librería de IA desarrollada y mantenida por el equipo de Google Brain desde 2015. Esta librería permite el desarrollo de modelos de IA usando Central Processing Units (CPUs), Graphics Processing Units (GPUs) y Tensor Processing Units (TPUs). Tensorflow es una plataforma de código abierto que tiene un ecosistema de herramientas y librerías que permite el desarrollo e investigación de modelos de IA. Tensorflow tiene un enfoque más orientado a la producción y despliegue de modelos de IA a un nivel empresarial, por lo que es más común ver su uso en empresas que en el ámbito académico. Un ejemplo de este enfoque es la presencia de la herramienta Tensorflow Serving [Olston et al. \(2017\)](#), que permite el desplegado de modelos de Tensorflow (o compatibles con el ecosistema) de una forma sencilla.

2.5.2.2. PyTorch

PyTorch([Ansel et al. \(2024\)](#)) fue desarrollado por Facebook AI Research en 2026, se hizo de código abierto en 2017 y en 2022 se unió a la Linux Fundation. Esta librería se centra en la idea de que la búsqueda de usabilidad y velocidad son compatibles. Pytorch permite el desarrollo de modelos de IA al igual que Tensorflow, pero su modelo de código y desarrollo ha hecho que se use más para prototipado rápido y desarrollo de proyectos dinámicos. Pytorch permite la integración con otras librerías como NumPy, SciPy y Cython, lo que permite un desarrollo más rápido y sencillo de modelos de IA.

2.5.2.3. Scikit-learn

Scikit-learn([Pedregosa et al. \(2011\)](#)) es una librería de IA desarrollada en Python que se centra en el aprendizaje automático. Esta librería se basa en NumPy, SciPy y matplotlib, lo que permite un desarrollo más sencillo de modelos de IA. Scikit-learn es una librería más ligera y más tradicional que Tensorflow o Pytorch, por lo que es más común ver su uso en el ámbito académico y de investigación. Scikit-learn se centra más en el análisis de datos predictivo y también es de código abierto como las anteriores.

2.5.2.4. YDF

YDF ([Guillame-Bert et al. \(2023\)](#)) es una librería para entrenar, evaluar, interpretar y desplegar Random Forest, gradient boosted decision trees, CART y isolation forest. Esta librería esta integrada con Tensorflow y keras, centra su desarrollo en la optimización de árboles de decisión y en hacer una Application Programming Interface (API) concisa y moderna.

2.5.3. Modelos de reconocimiento de gestos y poses

Muchos modelos de IA se han desarrollado para la detección de gestos y poses. Estos modelos pueden usar imágenes, videos, datos de sensores o esqueletos formados por puntos de referencia. En este apartado se van a destacar los más relevantes para el resto del trabajo.

2.5.3.1. YoLoV11

YoLoV11([Jocher et al. \(2023\)](#)) es un conjunto de modelos You Only Look Once (YOLO), entre los que se encuentra un modelo de detección de poses. Por defecto este modelo usa 17 puntos de referencia (presentados en la tabla 2.1) y usa imágenes de 640x640 píxeles como entrada. En la página del modelo de pose³, Ultralytics incluye los distintos modelos de pose y su rendimiento y requerimientos. Este modelo se ofrece preentrenado en distintos tamaños con la opción de reentrenarse usando un dataset con el formato especificado por Ultralytics⁴

³Página de YOLOv11-pose: <https://docs.ultralytics.com/es/tasks/pose/>

⁴Página de especificación del formato del dataset: <https://docs.ultralytics.com/es/datasets/pose/>

Tabla 2.1: Puntos de referencia del modelo YOLOv11-pose

Nariz
Ojo izquierdo
Ojo derecho
Oreja izquierda
Oreja derecha
Hombro izquierdo
Hombro derecho
Codo izquierdo
Codo derecho
Muñeca izquierda
Muñeca derecha
Cadera izquierda
Cadera derecha
Rodilla izquierda
Rodilla derecha
Tobillo izquierdo
Tobillo derecho

2.5.3.2. Modelos para la detección del balance de personas andando

En el artículo [Ma et al. \(2023\)](#) se presenta una comparativa de modelos para la detección del balance de personas andando en distintos niveles de valance y con distintas restricciones con un traje usado. Este artículo presenta una generación de esqueletos en 3D a partir de una kinect y realiza una comparativa de distintos modelos tradicionales y neuronales para ver su efectividad. En las conclusiones podemos ver como, de los distintos modelos utilizados, el que mejor resultados da es un [Deep Convolutional Neural Network \(DCNN\)](#) que los investigadores proponen. Otros modelos de la comparativa incluyen el [Random Forest](#), el [Support Vector Machine \(SVM\)](#), [LSTM](#) y Naive Bayes en términos de modelos tradicionales. Otra comparativa se realiza con otros modelos basados en [CNNs](#) como AlexNet, VGGNet y ResNet-18.

2.5.3.3. Otros modelos de detección de gestos y poses

En los últimos años se han desarrollado modelos generales de detección de gestos y poses. Estos modelos han tenido distintos enfoques, desde el uso de gafas de [VR](#) para hacer detección de gestos simples con las manos para utilizarlos como método de entrada (Meta Quest 2 y 3 por ejemplo), a modelos de seguimiento de manos y distintos modelos de reconocimiento de Google con Gemini.

Capítulo 3

Descripción del Trabajo

*“In the distant future.
In a land abandoned by man.
Machines wage a continuous war.
Unaware of the futility of their actions.”*
— Nier Automata

3.1. Traje

El primer paso del proyecto consistía en configurar el traje y poder usarlo dentro de Unity. El traje usado, Perception Neuron 3, viene con:

- a. 17 sensores identificados con cada punto.
- b. 15 Bandas de velcro que se ponen en el cuerpo para colocar los sensores.
- c. 3 puertos de carga para los sensores junto a 3 cables USB-C para conectarlos.
- d. 1 USB que contiene la licencia para poder conectar el traje al ordenador.
- e. 1 USB-C que funciona de receptor de las señales del traje.

Para poder usar el traje se necesita la aplicación Axis Studio.¹

3.1.1. Axis Studio y conexión del traje

Axis Studio es la aplicación creada por Noitom para poder capturar y grabar el movimiento en tiempo real y transmitirlo a aplicaciones de terceros (Unity, Blender o Unreal por ejemplo).

Mientras la aplicación está abierta es necesario tener el USB de la licencia introducido en el equipo. Al abrir la aplicación y crear un proyecto aparece un espacio vacío (como se puede ver en la figura 3.1) en la parte izquierda de la pantalla,

¹Enlace a la página de descarga de Axis Studio: <https://www.noitom.com/perception-neuron-downloads>

que es donde se verá en tiempo real la captación de movimiento; y un panel de la información del traje en la parte derecha.

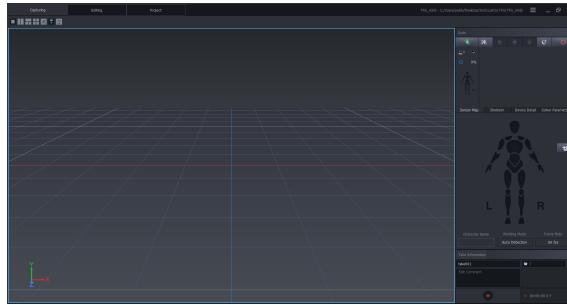


Figura 3.1: Captura de la aplicación Axis Studio antes de conectar un traje

Para conectar el traje se debe tener enchufado en el mismo equipo el USB receptor y los puertos de carga con los sensores en ellos. Posteriormente se deben desenchufar los puertos para que el traje se conecte con el receptor, que se muestra con un parpadeo azul en los sensores.

Una vez los sensores puestos en sus huecos de las bandas se debe pulsar al botón mostrado en la figura 3.2 se debe calibrar el traje en el botón mostrado en la figura 3.3. Esta calibración consiste en tres pasos:

1. Posición de T: cabeza recta, piernas rectas paralelas a los hombros y brazos extendidos hacia los lados.
2. Posición de S: cabeza recta, piernas ligeramente flexionadas, espalda recta y brazos extendidos hacia el frente.
3. Posición de A: cabeza recta, piernas rectas paralelas a los hombros y brazos relajados paralelos al tronco.



Figura 3.2: Captura del botón para conectar el traje a Axis Studio

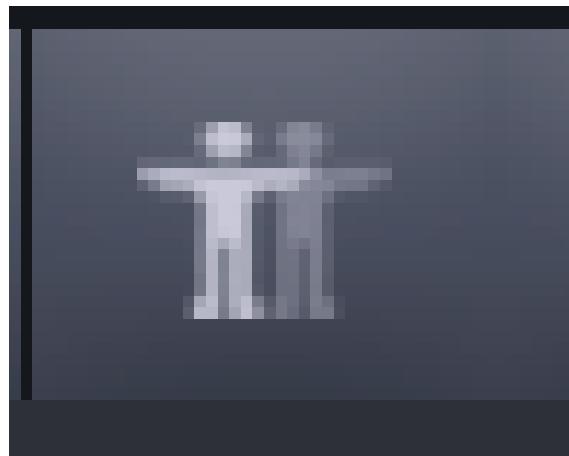


Figura 3.3: Captura del botón para calibrar el traje

Una vez el traje conectado y calibrado ya puede capturar el movimiento en tiempo real y transmitirlo a Unity.

3.1.2. Unity y la conexión del traje

Para poder usar la captura en Unity es necesario el plugin Neuron Mocap Live.²

Una vez metido el plugin en el proyecto se necesita un objeto vacío que contenga el componente “Neuron Source Manager”. Este componente se conecta a la aplicación mediante la dirección IP de la máquina que ejecuta Axis Studio y sockets TCP a un puerto que puedes configurar en la aplicación de Axis Studio mediante el panel que se muestra en la figura 3.4

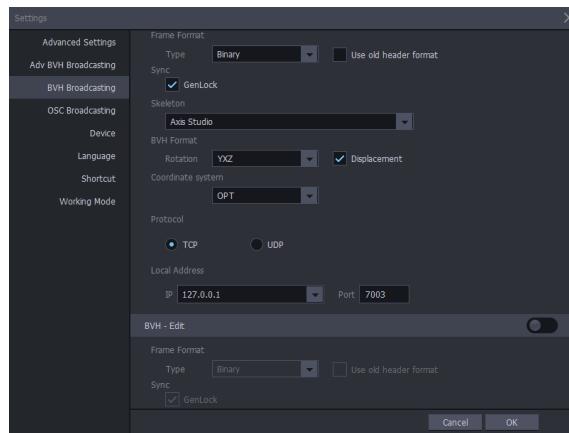


Figura 3.4: Captura del panel de configuración de Axis Studio

Posteriormente todos los objetos que vayan a usar la captura de movimiento deben ser hijos de este objeto vacío y tener el componente “Neuron Transforms

²Enlace a la página de documentación de Neuron Mocap Live: <https://support.neuronmocap.com/hc/en-us/sections/206474008-UNITY-SDK>

Instance”. Este componente tiene una lista de Transforms que usa para mapear los huesos del traje con los del objeto de Unity.

3.1.3. Problemas encontrados con el traje

Durante el desarrollo del proyecto han habido dos posibles problemas con respecto al traje.

El primero de ellos es la ausencia de guantes para capturar los dedos, ya que no vienen incluídos con el traje. A pesar de que esto causa que los dedos estén siempre en la misma posición con respecto a la mano no ha causado ningún problema en la identificación de gestos.

El segundo problema es la cantidad de ruido electromagnético presente en nuestro espacio de trabajo: la Facultad de Informática de la Universidad Complutense de Madrid. Este ruido causaba interferencias con el traje, haciendo que la señal con el ordenador fuese pobre constantemente y no se capturasen los movimientos correctamente.

Para solventarlo se requirió de un cable alargador de USB 2.0 de 10 metros que se conectaba al ordenador en el que se enchufaba el receptor y se acercaba lo más posible al traje, ampliando la señal al máximo.

Una vez conectado el traje, solventados los problemas y sabiendo los puntos que detecta era necesario un dataset que se pudiera acoplar al esqueleto que proporciona el traje.

3.2. Busqueda de un dataset

Para poder entrenar modelos de IA se requiere una gran cantidad de datos, en este caso de animaciones de los gestos escogidos que se han mencionado en la introducción. Además, estos datos tienen que estar en un formato en el que se pueda extraer la información de los huesos (posición y rotación) para que pueda ser usado por los modelos, y tenían que ser compatibles con los huesos del traje de captura de movimiento.

3.2.1. Dataset de la Universidad Carnegie Mellon

La Universidad privada Carnegie Mellon, ubicada en Pittsburgh, Pensilvania, tiene disponible de forma gratuita un dataset de movimientos recogidos mediante la captura de movimiento.³ Sin embargo este dataset no ha sido posible utilizarlo por tres motivos.

El primer motivo es la incompatibilidad del esqueleto utilizado por ellos con el utilizado por el traje de captura de movimiento Perception Neuron 3. Como se puede ver en Carnegie Mellon (2003) su traje contiene 41 puntos usados para la captura, mostrados en la figura 3.5, dando a lugar al esqueleto mostrado en la figura 3.6.

³Enlace a la página del dataset de la Universidad Carnegie Mellon: <https://mocap.cs.cmu.edu/>



Figura 3.5: Imagen del traje de captura de movimiento usado para la base de datos de la Universidad Carnegie Mellon, Fuente: <https://mocap.cs.cmu.edu/info.php>

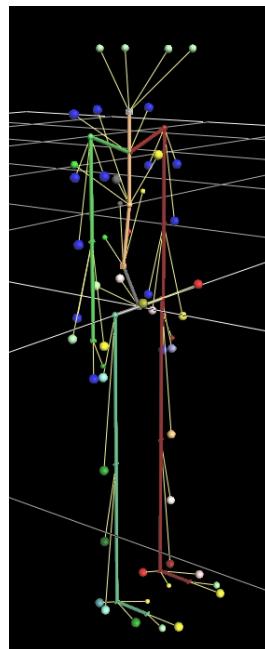


Figura 3.6: Imagen del esqueleto resultante del traje de captura de movimiento usado por la Universidad Carnegie Mellon, Fuente: <https://mocap.cs.cmu.edu/info.php>

Por otro lado, como ya mencionamos en el capítulo 3.1, el traje Perception Neuron 3 contiene 17 puntos, haciendo incompatibles los esqueletos.

El segundo motivo por el que no se ha usado el dataset de la Universidad Carnegie

Mellan es el formato de las animaciones disponibles. Las animaciones presentes en el dataset vienen en tres formatos distintos: c3d, asf (o amc) y vsk (o v). Los formatos c3d, vsk y v son unos formatos en binario usados para objetos en 3D, por lo que no se puede usar de forma sencilla para extraer la información del propio archivo. Por otro lado los archivos asf y amc son archivos en formato ASCII con la información de los huesos. Sin embargo la documentación aportada para poder usar la información es incompleta.

El último motivo por el que no se ha usado este dataset es por la escasez de animaciones de los gestos buscados. Animaciones como “correr” sí están presentes (aunque no en abundancia), pero otras como “pelear” o “saludar” no.

Este último problema también ha estado presente en páginas específicas de bancos de datasets como Kaggle.

3.2.2. Kaggle

Kaggle es una página web dedicada a la ciencia de datos y el aprendizaje automático y la IA. En Kaggle se pueden encontrar datasets, modelos, código y competiciones de IA. Kaggle tiene una gran comunidad de usuarios y una gran selección de datasets para distintas finalidades.

En un primer momento, se pensó en usar datasets de imágenes/vídeo, así que se buscó conjuntos de datos de este formato. La búsqueda no obtuvo los resultados esperados, ya que no se encontró datos de poses generales como las requeridas, y en concreto no se consiguió mucha cantidad de datos.

Tras analizar las desventajas de usar imágenes y videos, se pensó que era mejor usar un conjunto de datos basado en animaciones (o coordenadas de las mismas). Algunas de estas desventajas consideradas fueron:

- Complejidad de renderizado: las imágenes y/o videos requerirían de renderizar una cámara externa dentro del motor para capturar la presencia del usuario. Esto ya requeriría más potencia de computo, sin contar lo que requeriría el modelo.
- Transferencias de datos más grandes: al plantear un servidor para el modelo, se pensó en el tamaño de las imágenes que habría que mandar para hacer las inferencias. En el caso de tener una red de velocidad limitada sería mejor mandar los puntos concretos del traje que mandar imágenes completas. Las imágenes, incluso al ser de baja calidad, requerirían una cantidad muy grande de píxeles para hacer el modelo del usuario fácilmente reconocible.
- Perdida de contexto: el traje de captura de movimiento te permite obtener datos tridimensionales de cada punto por defecto, sin necesidad de tener que recrear un esqueleto 3D con capas extra en el modelo.

Tras este cambio de enfoque, se buscó datasets de animaciones y modelos 3D. Resultó ser una búsqueda incluso peor que la anterior, ya que por la limitación de gente que tiene acceso a trajes de captura de movimiento, no había datasets públicos en dataset con este tipo de datos.

Ya que no se encontró un gran dataset que cumpliese con nuestros requerimientos se tomó la decisión de buscar en un banco de animaciones los gestos requeridos y transformar esas animaciones en un formato que pudiesen ser procesados.

3.3. Dataset Artificial

El banco de animaciones en el que fueron buscados los gestos necesarios fue Mixamo.⁴

Mixamo es una página web creada por Adobe en la que se encuentran de manera gratuita modelos con un *rig* humanoide y animaciones para estos rigs, además de una funcionalidad que permite crear el *rig* de un personaje que se suba a la página. En la página hay un buscador en el que puedes buscar tipos de animaciones, pero como no es viable descargarse todas las animaciones de todos los gestos buscados se usó una herramienta para automatizarlo.

3.3.1. Herramienta para descargarse animaciones de Mixamo de forma automática

Esta herramienta se encontró en GitHub hecha por el usuario *juanjo4martinez*⁵. La herramienta (*mixamo-downloader*⁶) permite buscar y descargar animaciones de Mixamo de manera automática y con el uso de distintos filtros para acomodarse a lo requerido por el usuario.

Al empezar a usarse la herramienta se detectó que al empezar a bajar animaciones, dependiendo del nombre que tuvieran las mismas, el programa era incapaz de guardar dichas animaciones. Esto se debía a la incompatibilidad de distintos sistemas de archivos para usar determinados caracteres en sus nombres de archivos (interrogaciones, barras o dobles barras, dos puntos, etc). Se hizo un *fork* de la herramienta⁷ que efectúa estos arreglos y añade la opción de reescribir animaciones anteriores que tuvieran el mismo nombre. La interfaz completa de la herramienta arreglada se puede ver en la figura 3.7.

⁴Página web de Mixamo: <https://www.mixamo.com>

⁵Enlace al perfil del creador de la herramienta <https://github.com/juanjo4martinez>

⁶Enlace al repositorio de la herramienta <https://github.com/juanjo4martinez/mixamo-downloader>

⁷Enlace al repositorio con los arreglos <https://github.com/ALK222/mixamo-downloader>

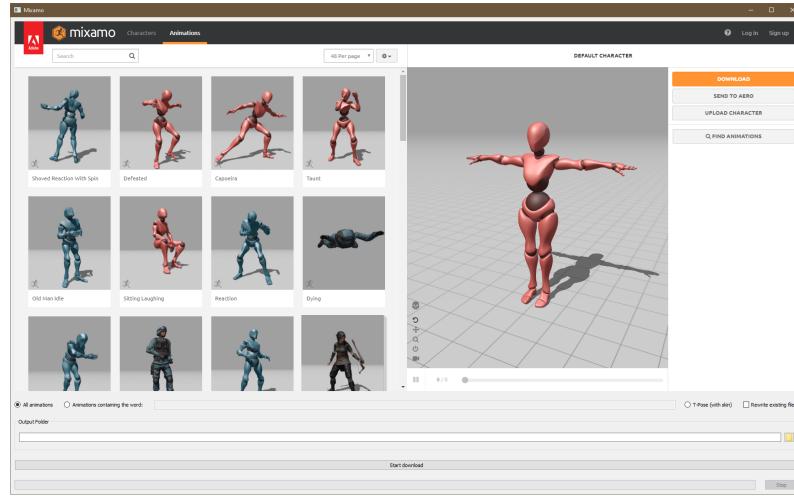


Figura 3.7: Herramienta Mixamo Downloader Arreglada

Una vez realizados los cambios, se descargaron las animaciones disponibles en Mixamo para los gestos que se querían usar. El resultado de estas descargas se puede ver en el dataset [Barrachina Argudo y Sánchez Martín \(2025b\)](#).

Las animaciones descargadas tienen un formato [FBX](#), por lo que era necesario pasárselas a un formato que puedan usar los modelos de [IA](#). Para ello se decidió pasar la información relevante de los huesos en la toda la animación (su posición y rotación en cada frame) a un archivo CSV, donde cada columna iba a ser esta información y cada fila un frame de la animación. En la tabla A.1 se puede ver la cabecera para los CSV. En la siguiente tabla (3.1) se puede ver un ejemplo de los valores que se guardan de un punto concreto del traje.

Tabla 3.1: Cabecera del [CSV](#) de cada animación, en orden descendente y de izquierda a derecha (incompleta).

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_Hips_rotx	Robot_Hips_roty	Robot_Hips_rotz
...

Como no es viable ni escalable meter todas las animaciones a un proyecto de Unity ni crear un [Animator](#) con todas estas se ideó una herramienta en Unity que se metiese todas las animaciones y contruía el Animator en tiempo de ejecución.

3.3.2. Carga mediante asset bundles

En un principio se pensó en usar los assets bundles para cargar las animaciones. Los assets bundles son archivos de Unity que contienen archivos serializados (cualquier asset para un videojuego menos código) y pueden cargarse en tiempo de ejecución. Finalmente se descartó la idea de usar este tipo de archivos, ya que necesitan construirse en un proyecto de Unity, por lo que no solucionaba el problema

de meter las animaciones a mano a un proyecto de Unity

3.3.3. Carga mediante Asset Database

Finalmente se decidió usar Asset Database para la carga automática de las animaciones. Asset Database es una API de Unity que permiten trabajar con assets, siempre y cuando estén incluidos en el proyecto aunque no necesariamente cargados. Para suplir la condición de que estuviesen incluidos en el proyecto se hizo un script que descargaba las animaciones, separadas por carpetas cuyo nombre era el tipo de gesto, desde la base de datos de Kaggle hacia la carpeta “Resources” del proyecto y ejecutaba desde línea de comandos el editor del proyecto.

Una vez las animaciones estuviesen descargadas en la carpeta de “Resources” del proyecto se busca iterativamente por la carpeta para cargar las animaciones y guardarse la relación entre el tipo de gesto y la animación. Una vez completado empieza la ejecución de las animaciones.

Esta ejecución consiste en cargar la animación en la máquina de estados del [Animator](#) gracias a un componente llamado “Animator Override Controller”, que permite cambiar el clip de animación de una instancia del [Animator](#) sin cambiar la lógica de su máquina de estados. Cuando empieza la animación se crea un [CSV](#) con el formato “Animación_Número de animación.csv” y empieza su ejecución. Una vez finalizada la ejecución de una animación se cierra el [CSV](#) y se busca la siguiente animación. Se puede ver un ejemplo de todo este proceso en un timelapse de la ejecución de la herramienta⁸, también se deja una captura del mismo para ver el funcionamiento de la herramienta en el editor de Unity.

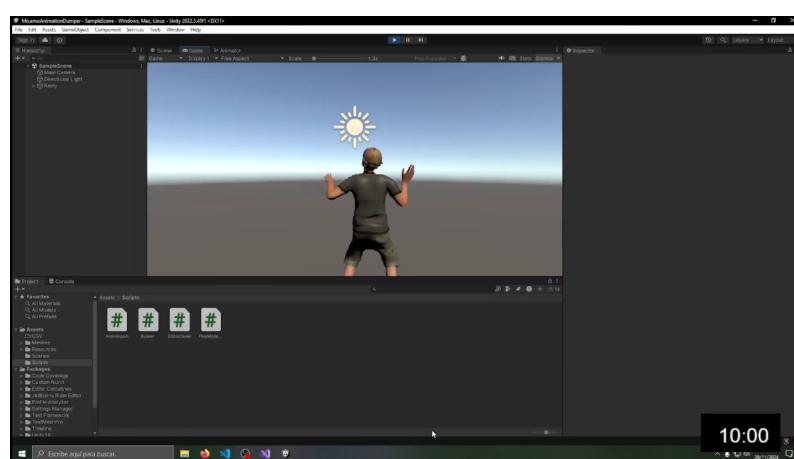


Figura 3.8: Frame de un *timelapse* de la ejecución de la herramienta

En la figura 3.9 se puede ver un diagrama de flujo del funcionamiento de esta herramienta.

⁸Enlace al timelapse completo a x4 de tiempo <https://youtu.be/NOawMoaG98M>

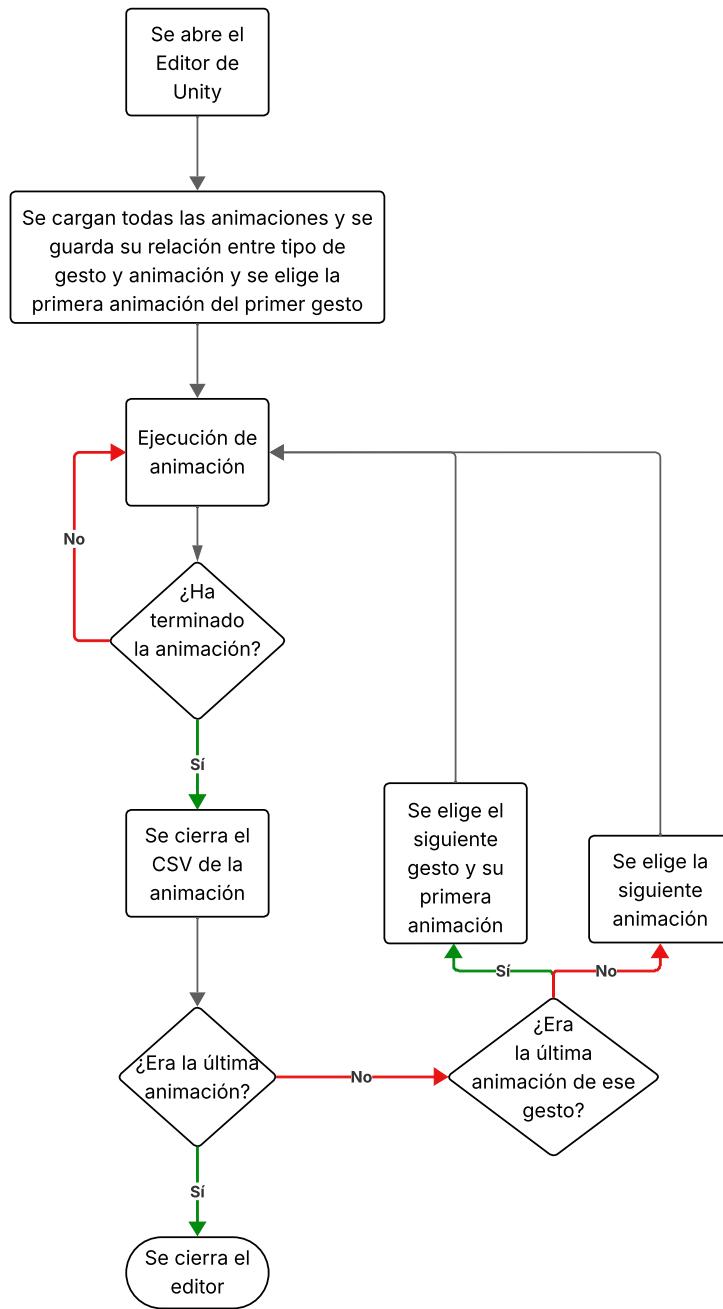


Figura 3.9: Diagrama de flujo de la herramienta que carga y ejecuta las animaciones de Mixamo en tiempo de ejecución, llamada Mixamo Dumper

Al finalizar todas las animaciones se cierra el editor de Unity y se procesan los CSV resultantes.

La ventaja de hacer esta herramienta es que es independiente del número de gestos y de animaciones por cada gesto, lo que hace que esta herramienta haga escalable el caso de que se quieran introducir o quitar gestos.

3.3.4. Tratamiento de CSV resultantes

Los [CSVs](#) restantes se copiaron en un nuevo *dataset* de Kaggle en una carpeta llamada “full_animations”. Una vez subidos estos datos, se creó una clase en Python para estandarizar estos [CSVs](#). Los datos se estandarizaron con el siguiente criterio:

- Todas las animaciones deben de tener el mismo número de frames. Para ello se estandará la duración a 90 frames que es el número de [Frames Per Second \(FPS\)](#) que usan las gafas de [VR](#) de Meta Quest 2 y la herramienta Mixamo Animation Dumper.
- Las animaciones que duraran más de 90 frames se dividen en varias animaciones de 90 frames o menos.
- Las animaciones que duraran menos de 90 frames se repiten desde el inicio hasta completar los 90 frames.
- Las animaciones que no lleguen a 10 frames se consideran inválidas y se eliminan.

Este proceso se puede ver de manera más visual en la figura 3.10 y se puede ver el diagrama de la clase `DataLoader` en la figura 3.11, algunas de las funciones de esta clase se verán en los próximos apartados.

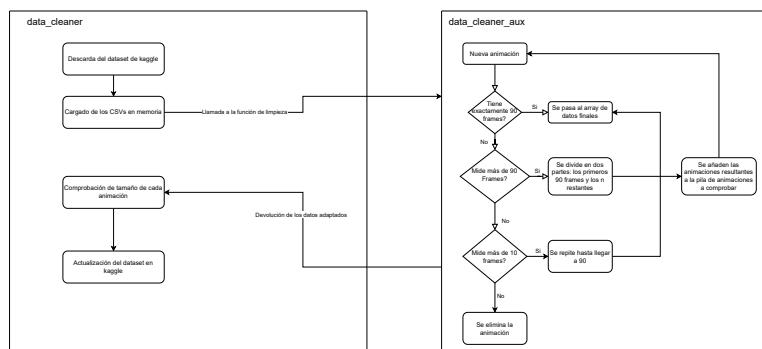


Figura 3.10: Diagrama de flujo del proceso de limpieza de datos (funciones `data_cleaner` y `data_cleaner_aux` de la clase `DataLoader`)

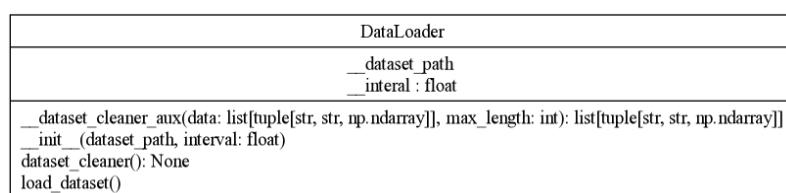


Figura 3.11: Diagrama de la clase `DataLoader`

3.4. Dataset real

Debido a que las animaciones encontradas en el banco de animaciones no eran las suficientes y estaban descompensadas se decidió crear una herramienta para poder recoger los datos de usuarios.

3.4.1. Funcionamiento de la herramienta

La herramienta se debe conectar con el traje como hemos mencionado anteriormente en el apartado 3.1.2 Una vez conectado se le debe pasar al componente MocapDumper el nombre de la animación que se va a grabar, el número de toma de esa animación y un número que sirva como identificador de usuario.

Finalmente cuando la herramienta está ejecutándose y se le da a la tecla “espacio” genera un CSV del formato “animación_User_Número de ususario_Take_Número de toma” (si el fichero ya existía lo sobreescrive), escribe la misma cabecera del CSV que se menciona en la tabla A.1 y pone la aplicación en estado de grabación. En este estado la aplicación escribe en cada frame todos los componentes de la posición y rotación (en forma de vector de ángulos de Euler) de cada hueso. Cuando se le vuelve a dar al espacio la aplicación cierra el fichero y vuelve a un estado de no grabación.

En la figura 3.12 se muestra un diagrama de cómo se conectan los diferentes componentes necesarios en esta herramienta

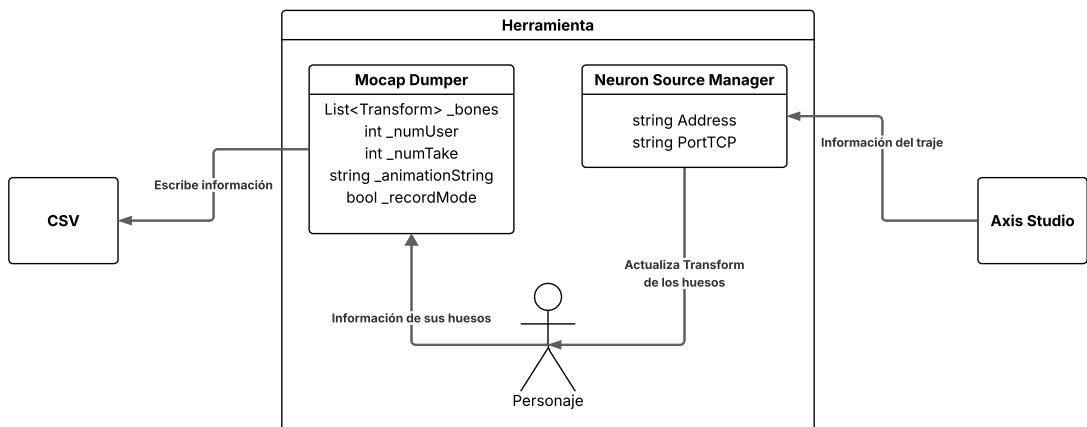


Figura 3.12: Diagrama de la conexión entre los distintos componentes de la herramienta, llamada Mocap Dumper

3.4.2. Recogida de datos

La recogida de datos consistió en ponerle el traje de captura de movimiento a los usuarios y pedirles que realizaran tres tomas de cada uno de los gestos, a excepción del gesto de baile, ya que de ese gesto había bastantes más datos que el resto.

Para ello se creó un formulario para que los usuarios interesados en ello se apuntasen. En el formulario se explicaba el objetivo de la prueba y se numeraban los

datos que iban a ser pedidos en la prueba. Los campos a llenar eran:

1. Correo
2. Aceptar recogida de datos
3. Posible hueco en un horario, en huecos de una hora de lunes a viernes y de 9:00 a 20:00

Una vez creado el formulario se creó una selección de carteles (figuras B.1, B.2, B.3) el cual se colgó en redes y se colgó en distintas facultades, teniendo como resultado que se apuntasen 75 personas en el formulario.

Lo siguiente era tener un sitio en el que hacer las pruebas. Debido a que las pruebas requerían movimiento era preciso un lugar en el que los usuarios se pudiesen mover sin dificultades y que no estuviese a la vista para preservar la intimidad de los mismos.

Una vez se cerró el formulario, se procedió a hacer un algoritmo que asignara la mayor cantidad de citas posibles dadas las restricciones de tiempo de los usuarios y los días disponibles. El algoritmo⁹ usa un esquema de ramificación y poda para conseguir la combinación con mayor cantidad de citas disponibles sin tardar más de 10 segundos en procesar las combinaciones válidas.

Finalmente desde el día 14/04/2025 a las 9:00 hasta el día 17/04/2025 a las 19:30 se pudo grabar en el despacho 216 (Sala de grabaciones) de la Facultad de Informática de la Universidad Complutense de Madrid. De los 75 usuarios apuntados finalmente se presentaron 65, consiguiendo así 975 animaciones en formato CSV, 195 de cada gesto. En la figura 3.13 se muestra una fotografía de un usuario durante el proceso de calibración.

⁹Enlace al algoritmo: https://github.com/FratosVR/Models/blob/main/citas_generacion_dataset/Scheduler.py



Figura 3.13: Fotografía de un usuario en las pruebas

A los usuarios no solo se les pidió que realizaran gestos, sino que también se les pidió una serie de información demográfica de manera anónima para saber como de representativa era la muestra. La información pedida fue:

1. Género
2. Edad
3. Nacionalidad
4. Idiomas hablados
5. Mano dominante

Estos datos se recogieron con la cuenta de Google de las personas que llevaban el formulario, por lo que no se guardó ningún dato identificable de ningún usuario. En la categoría de género (figura C.1) podemos ver que los datos son un 43.1 % femenino, 50.8 % masculino y 6.2 % no binario. La edad (figura C.2) más común es de 21, con un recuento de un 26.2 % de los encuestados. La nacionalidad (figura C.3) más común es la española, con un 89.23 % de los encuestados. En la figura C.4 se puede ver que el idioma más hablado es el español, con un 100 % de los encuestados, seguido de inglés y francés. En la figura C.5 se puede ver que el 93.85 % de los encuestados son diestros.

Tras la recogida de datos, se siguió el mismo proceso de estandarización de datos que se usó en el dataset artificial. Antes de estandarizar los datos parecía que

se había conseguido eliminar el gran desbalance del dataset hacia los gestos de baile, tras la estandarización de los datos, se vió que las animaciones de baile seguían ganando en cantidad por mucho respecto a las otras. La comparativa se puede ver en las figuras 3.14 y 3.15.

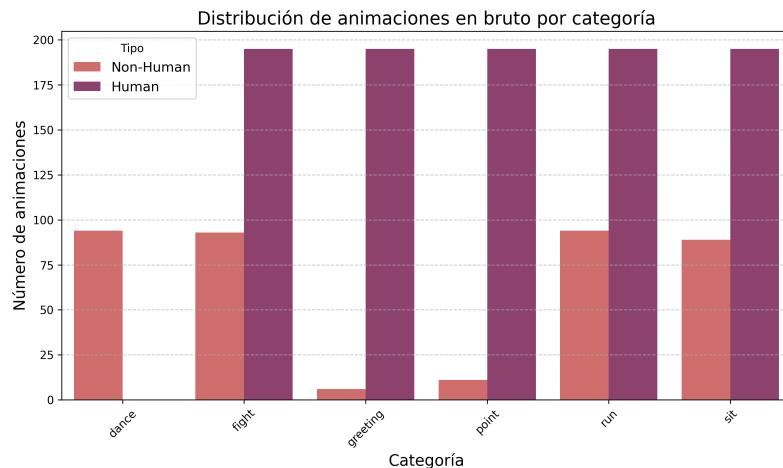


Figura 3.14: Distribución de los datos brutos

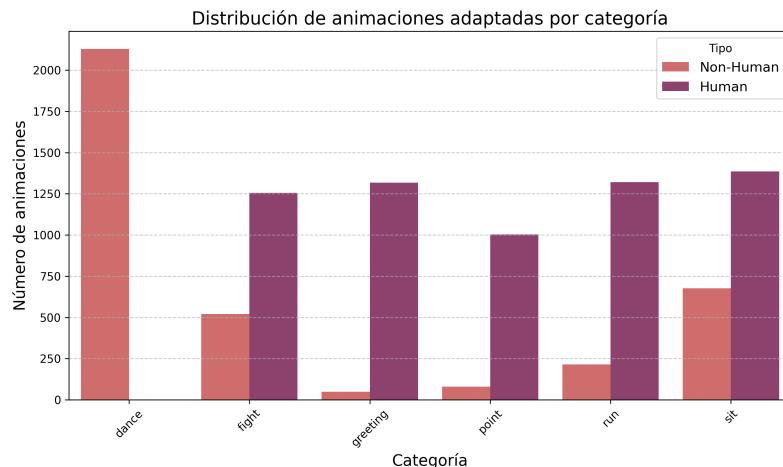


Figura 3.15: Distribución de los datos estandarizados

El resultado final de todos los datos se puede ver en [Barrachina Argudo y Sánchez Martín \(2025a\)](#). Una vez recogidos y procesados los datos recogidos en las pruebas de usuario se procede al entrenamiento de los distintos modelos para su posterior comparativa.

3.5. Modelos de IA

Tras la adaptación y guardado de datos, se procede al desarrollo de los modelos de IA. Este desarrollo no implica solo el modelo, sino también la creación de una interfaz

de entrenamiento y visualizado de resultados(para los modelos que lo permiten). Durante todo el proceso se han usado las siguientes tecnologías:

- **Python**
- **Tensorflow**
- **TensorBoard**
- **Tensorflow Serving**
- **Keras**
- **Pandas**
- **Numpy**
- **Matplotlib**
- **Scikit-learn**
- **YDF**
- **Gradio**
- **Kaggle**
- **Jupyter Notebook**

En el repositorio¹⁰ se puede encontrar el código de todos los modelos y su entrenamiento, así como algunos modelos ya preentrenados. Durante esta sección se explicará el proceso de desarrollo de cada parte del proyecto.

3.5.1. Interfaz de entrenamiento y seguimiento

Al principio del desarrollo, surgió la necesidad de hacer que el entrenamiento de los modelos fuera accesible para gente que nunca hubiera trabajado con IA o Python. Esto ocurre debido a que se ideó una comparativa de modelos, y los ordenadores disponibles no daban la talla para entrenar los modelos en el tiempo necesario. Por esto se pidió como favor a la asociación LAG (asociación de estudiantes de la Facultad de informática de la UCM) el poder usar sus equipos con mejores gráficas para llevar a cabo estos entrenamientos.

Se usó Gradio([Abid et al. \(2019\)](#)) para crear una interfaz web accesible para cualquier persona. En un primer momento esta interfaz solo contaba con una pantalla en la que había:

- Un selector de tipo de modelo
- Un área de texto para introducir la ruta al conjunto de datos

¹⁰Enlace al repositorio de GitHub <https://github.com/FratosVR/Models>

- Un *slider* para seleccionar el intervalo de tiempo entre los frames de animación
- Un botón para seleccionar todos los posibles intervalos de tiempo
- Un botón para iniciar el entrenamiento
- Un área de archivo para descargar el modelo entrenado
- Un área de imagen para mostrar la matriz de confusión tras el entrenamiento
- Un área de texto para mostrar el resultado del entrenamiento

Esta interfaz (figura 3.16) se hizo lo mas simple posible para que un usuario estándar pudiera poner a entrenar un modelo sin tener que tocar nada de código. Sólo se tienen que seleccionar el modelo y el intervalo deseado, introducir la ruta al *dataset* y esperar a que termine.

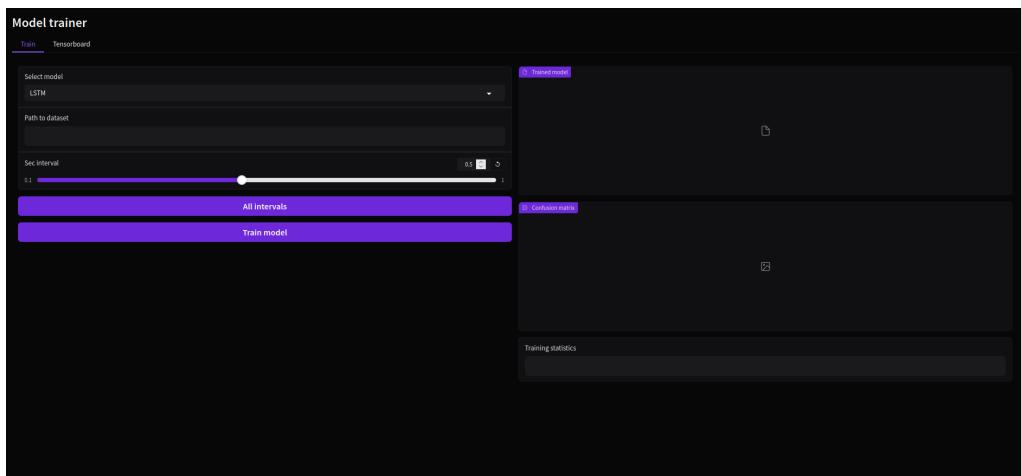


Figura 3.16: Interfaz de entrenamiento

La función de entrenamiento que se ejecuta al pulsar el botón “Train model” hace lo siguiente:

1. Se crea un entrenador del modelo seleccionado
2. Se crea un objeto *DataLoader* que se encargará de cargar los datos con el intervalo de tiempo de separación seleccionado.
3. Se separan los datos cargados en 60 % para entrenamiento, 20 % para validación y 20 % para test.
4. Se pasan las categorías con *LabelEncoder* para que el modelo pueda entenderlas.
5. Se entrena el modelo con los datos de entrenamiento y validación.
6. Se guarda el modelo.

7. Se comprueba cual es el mejor modelo entrenado entre todos los intervalos de tiempo seleccionados.
8. Se muestra la matriz de confusión en la interfaz, el archivo del modelo y sus estadísticas.

En la parte superior izquierda se puede ver dos secciones: “Train” y “TensorBoard”. La sección “Train” es la explicada anteriormente, la sección “TensorBoard” es una sección que permite ver el progreso del entrenamiento en tiempo real. No solo eso, sino que también nos permite ver el rendimiento de modelos ya entrenados y ver como los hiperparámetros han afectado a su tasa de acierto. TensorBoard permite añadir funciones al código de entrenamiento para modificar el comportamiento del método de entrenamiento y dejar constancia de las distintas partes del mismo. Esta interfaz se puede ver en la figura 3.17.

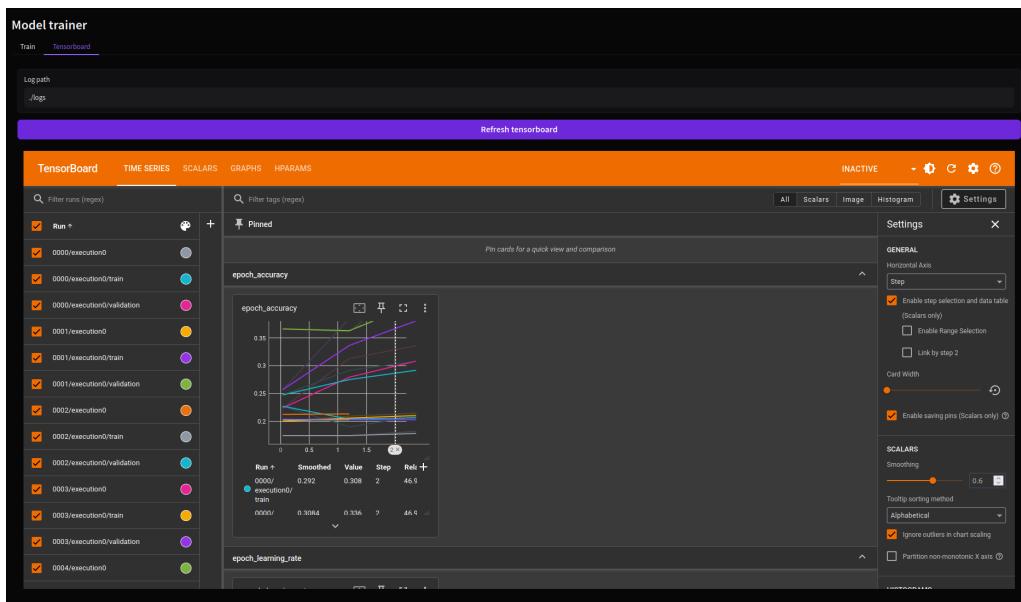


Figura 3.17: Interfaz de TensorBoard

Para todo lo descrito, el archivo “Interface.py” tiene la estructura presentada en la figura 3.18. Las funciones `__change_interval` y `__change_interval_all` cambian los intervalos de tiempo usados dentro del entrenamiento, `__setup_ui` crea el *layout* de la interfaz, `__refresh_tensorboard` cambia la ruta de los *logs* de TensorBoard y recarga el *iframe*. El método `__train` ejecuta todo el proceso de entrenamiento previamente descrito.

3.5.2. Carga de datos

La carga de datos se gestiona con la clase *DataLoader*. Esta clase se mencionó en el apartado del dataset artificial ya que es la misma que se encarga de estandarizar los datos. Cuando se llama a la función de inicio de la clase se le pasa un intervalo de tiempo, este intervalo marca la distancia entre frames que se va a cargar. Si le indicamos una distancia de 0.8, cargará el frame 0 y el 72. Esto nos permite entrenar

Interface
<code>_blocks : Blocks</code> <code>_models : list</code> <code>_models_map : dict</code> <code>_sec_interval : list</code> <code>log_path : NoneType</code> <code>_change_interval(value)</code> <code>_change_interval_all()</code> <code>_init_()</code> <code>_refresh_tensorboard(log_path)</code> <code>_setup_ui()</code> <code>_train(model, path)</code> <code>launch()</code>

Figura 3.18: Estructura del archivo de la interfaz

distintos modelos con distintas ventanas de envío de datos para comprobar si hay alguna diferencia en el rendimiento.

Estos datos una vez cargados se envían como una lista de tuplas `<tipo_animacion, lista_de_frames>`. Tras esto, el objeto que reciba estos datos ya puede transformarlos como sea necesario y separarlos en entrenamiento, test y validación.

3.5.3. Modelos usados

A la hora de escoger modelos para la comparativa, se tuvo en cuenta el estudio [Ma et al. \(2023\)](#) como un referente. Se tuvo siempre en cuenta que se intentaba buscar un modelo con el cómputo más simple posible para que fuera lo más accesible para cualquier tipo de máquina. Valorando también las dificultades de tiempo que conlleva el entrenamiento de modelos al final se consideró probar tres redes neuronales usadas para series temporales y un modelo clásico de clasificación.

En un primer momento se pensó también en modelos como [YOLO](#), pero al ser un modelo basado en imágenes y el estudio plantearse con un traje de captura de movimiento, se decidió que sería una carga muy grande para las aplicaciones que usarán el modelo resultante por tener que renderizar una segunda cámara constantemente para el usuario. Otros modelos como [SVM](#) o [K-Nearest Neighbors \(KNN\)](#) fueron descartados por falta de tiempo de entrenamiento.

3.5.3.1. LSTM

Este primer modelo va a servir para ilustrar como funcionan el resto de modelos basados en redes neuronales en este estudio. La estructura de la clase *LSTMTrainer* se puede ver en la figura 3.19. La mayoría de los atributos privados de la clase son los hiperparámetros del modelo, los otros atributos se usan para guardar información o para guardar el mecanismo usado para buscar hiperparámetros óptimos.

La función `train_with_hparams` se encarga de generar modelos con distintos hiperparámetros y entrenarlos para buscar el mejor. Para que este mecanismo fuera lo más eficiente posible, se ha hecho uso de la biblioteca keras-tuner ([O'Malley et al. \(2019\)](#)), un *framework* de búsqueda de hiperparámetros con distintos algoritmos de

búsqueda ya implementados. Tras leer el artículo [Li et al. \(2018\)](#) se optó por usar el algoritmo Hyperband, dado que los inconvenientes que presenta no representan un problema comparado con las ventajas de velocidad y eficiencia que otorga.

Esta búsqueda de hiperparámetros se centra en los siguientes:

- **activation**: Función de activación de la capa oculta.
- **dropout**: Porcentaje de *dropout* de la capa oculta.
- **recurrent_activation**: Función de activación de la capa recurrente.
- **recurrent_dropout**: Porcentaje de *dropout* de la capa recurrente.
- **unroll**: Si se usa el *unroll* de la capa LSTM.
- **use_bias**: Si se usa el *bias* de la capa LSTM.

El entrenamiento crea un modelo con los hiperparámetros seleccionados dentro de un rango, se entrena durante unas épocas y se comprueban resultados. Se pasa a otro conjunto de hiperparámetros y así hasta completarlos todos. Tras esto, se aumenta el número de épocas y se siguen entrenando los anteriores modelos. Tras acabar de entrenar todos los modelos, se selecciona el que mejor precisión en validación tenga.

De este “mejor modelo” se crea una matriz de confusión, se guarda el modelo en formato keras y se envían las estadísticas a la interfaz de gradio. Todo este proceso queda registrado en TensorBoard gracias a la librería *HParams* ([Petrochuk \(2019\)](#)).

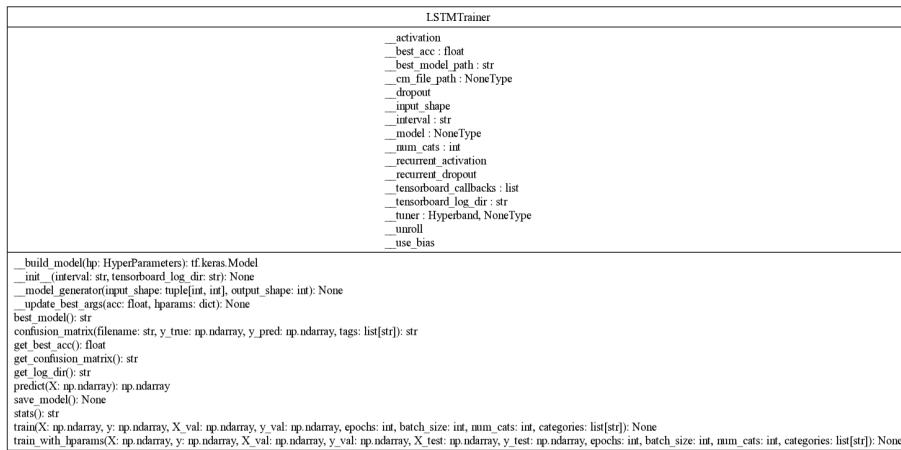


Figura 3.19: Estructura de la clase LSTMTrainer

El modelo [LSTM](#) usado es un *Sequential* de keras con una capa LSTM y una capa densa. La capa LSTM con los hiperparámetros antes descritos y la capa densa tiene tamaño el número de categorías de animaciones y una activación softmax. El modelo se compila utilizando el optimizador *Adam*, la perdida se calcula con *categorical_crossentropy* y la métrica de evaluación es la *accuracy*. Durante el proceso de entrenamiento se usan los *callbacks* de *EarlyStopping* para ahorrar recursos en entrenamientos que no parecen mejorar y los *callbacks* de *TensorBoard* para guardar los logs de entrenamiento y poder verlos en la interfaz de TensorBoard.

Entre los modelos de **LSTM** entrenados, el del intervalo de 0.8 fue el que mejor rendimiento tuvo, con una precisión de 0.5912 en validación. A continuación se puede ver el esquema del modelo en la figura 3.20, la matriz de confusión en la figura 3.21 y el gráfico de entrenamiento en la figura 3.22. Podemos ver en la matriz de confusión que los gestos menos reconocidos son baile y saludo. Los resultados del resto de intervalos se pueden ver en el apéndice D.

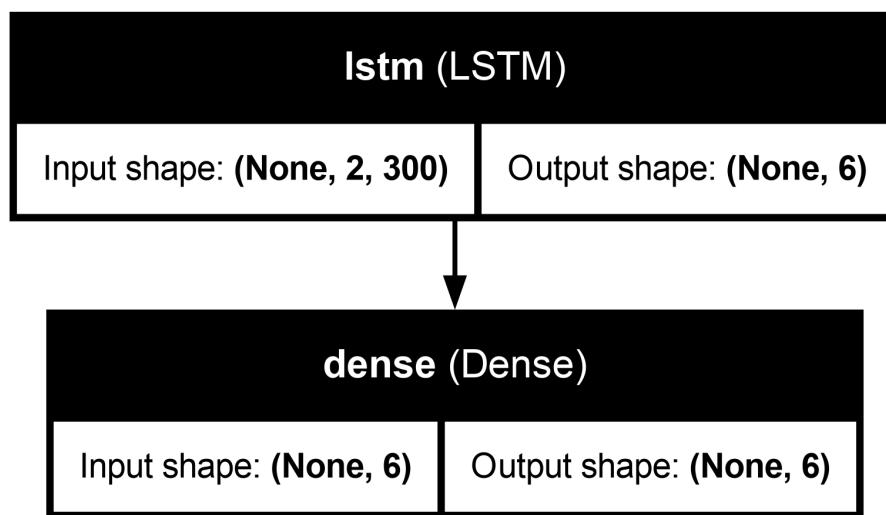


Figura 3.20: Esquema del modelo LSTM

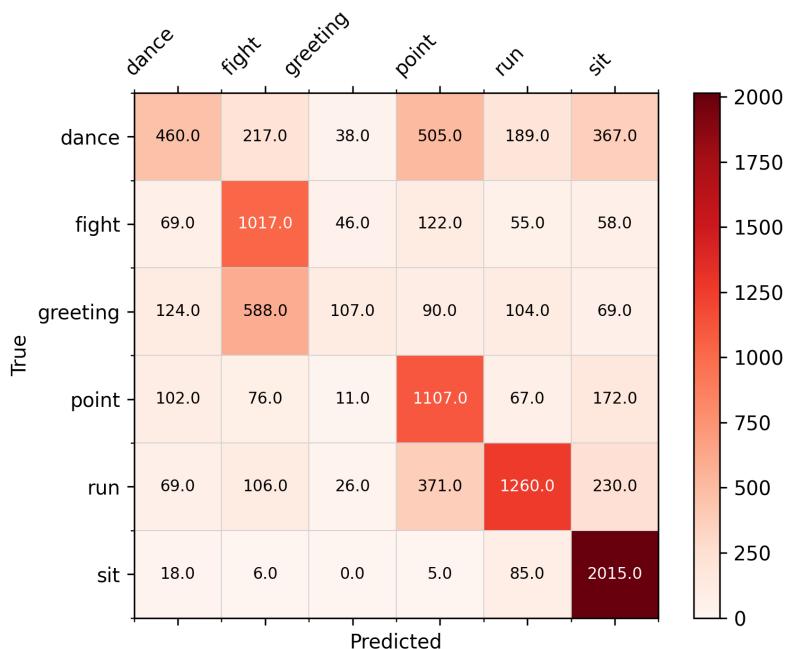


Figura 3.21: Matriz de confusión del modelo LSTM

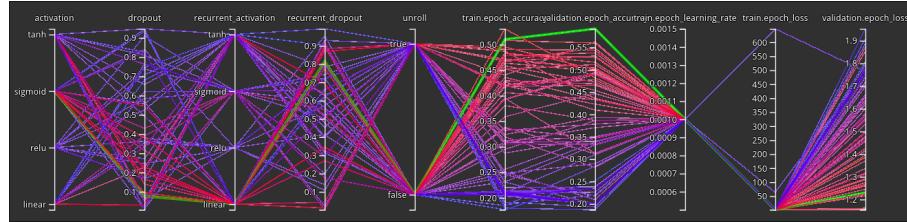


Figura 3.22: Gráfico de entrenamiento del modelo LSTM

3.5.3.2. CNN

El modelo de **CNN** se entrenó de una manera muy similar al de **LSTM**. En el uml de la figura 3.23 se puede ver la estructura de la clase *CNNTrainer*, la cual es muy similar a la de *LSTMTrainer*. La diferencia está en los hiperparámetros buscados. Estos son:

- **activation:** Función de activación de la capa oculta.
- **conv_filters:** Número de filtros de la capa convolucional.
- **dense_units:** Número de neuronas de la capa densa.
- **dropout:** Porcentaje de *dropout* de la capa oculta.
- **kernel_size:** Tamaño del kernel de la capa convolucional.
- **pool_size:** Tamaño del *pooling* de la capa convolucional.

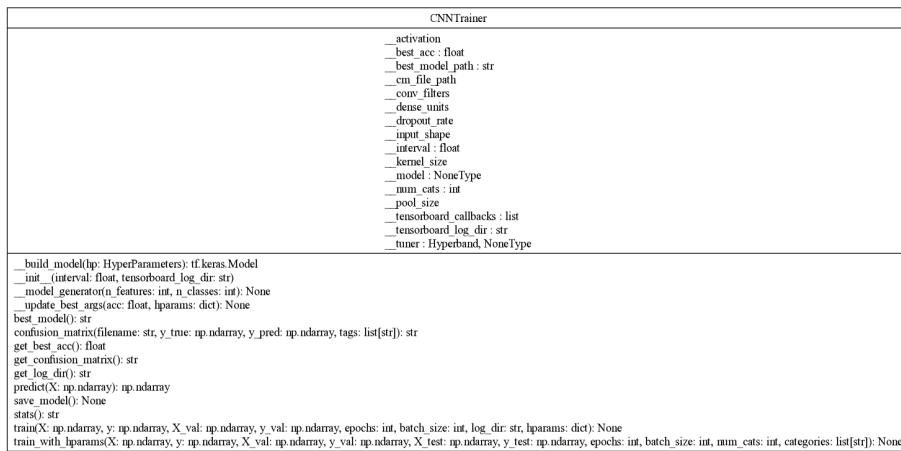


Figura 3.23: Estructura de la clase CNNTrainer

El método de entrenamiento es el mismo que el descrito en el apartado anterior, haciendo uso de keras-tuner para encontrar los mejores hiperparámetros. En el caso de este modelo, el mejor resultado se consigue con el intervalo de 1 segundo, haciendo este modelo más un reconocedor de poses que de gestos, con una precisión de 0.56002 en validación. La precisión en validación fue de 0.6342, lo que lo convierte en el mejor modelo entrenado. En la figura 3.24 se puede ver el esquema del modelo, en la figura

3.25 la matriz de confusión y en la figura 3.26 el gráfico de entrenamiento. En este caso podemos ver que los gestos que peor detecta son baile y saludo (siendo saludo muy confundido con pelea). Los resultados del resto de intervalos se pueden ver en el apéndice F.

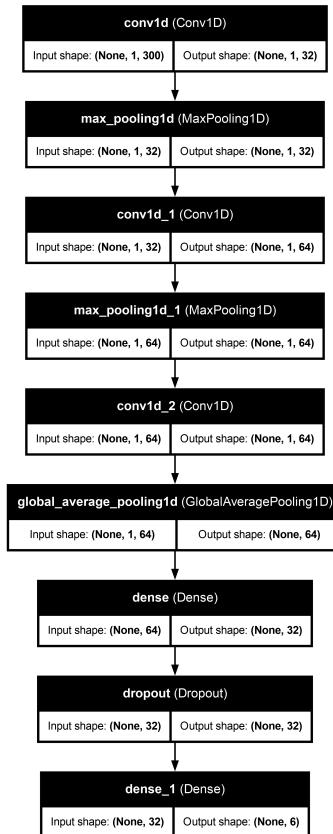


Figura 3.24: Esquema del modelo CNN

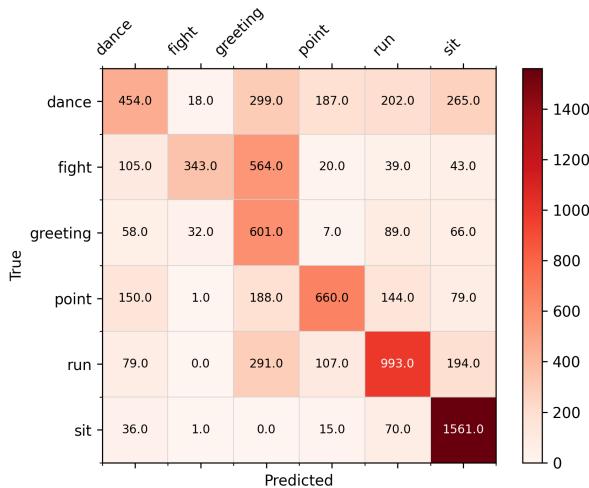


Figura 3.25: Matriz de confusión del modelo CNN

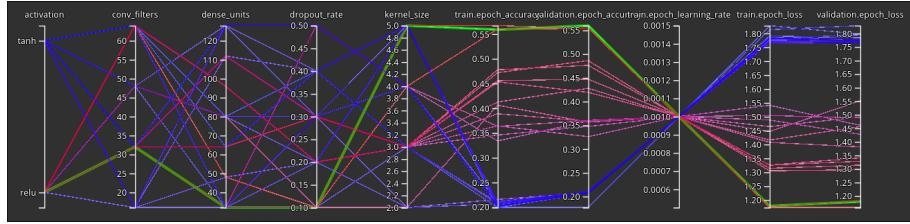


Figura 3.26: Gráfico de entrenamiento del modelo CNN

3.5.3.3. RNN

Con el **RNN** se sigue la misma base que para los dos modelos anteriores. La clase *RNNTrainer* (estructura en la figura 3.27) es muy similar a las anteriores, cambiando una vez más los hiperparámetros a buscar. En este caso son:

- **activation**: Función de activación de la capa oculta.
- **activity regularizer**: Regularizador de la capa oculta.
- **bias constraint**: Restricción del *bias* de la capa oculta.
- **bias initializer**: Inicializador del *bias* de la capa oculta.
- **bias regularizer**: Regularizador del *bias* de la capa oculta.
- **dropout**: Porcentaje de *dropout* de la capa oculta.
- **kernel constraint**: Restricción del kernel de la capa oculta.
- **kernel initializer**: Inicializador del kernel de la capa oculta.
- **kernel regularizer**: Regularizador del kernel de la capa oculta.
- **recurrent constraint**: Restricción de la capa oculta.
- **recurrent dropout**: Porcentaje de *dropout* de la capa oculta.
- **recurrent initializer**: Inicializador de la capa oculta.
- **recurrent regularizer**: Regularizador de la capa oculta.
- **unroll**: Si se usa el *unroll* de la capa oculta.
- **use bias**: Si se usa el *bias* de la capa oculta.

A pesar de ser el modelo con mayor búsqueda de hiperparámetros, está en una reñida competición con el modelo **CNN** para ver cual de los dos detecta peor los gestos. Su mejor intervalo ha sido el de 0.8 segundos con una precisión de 0.47865 en validación. En la figura 3.28 se puede ver el esquema del modelo, en la figura 3.29 la matriz de confusión y en la figura 3.30 el gráfico de entrenamiento. En este caso podemos ver que los gestos que peor detecta son baile y saludo (siendo pelea muy

```

RNNTrainer
    activation
    activity_regularizer: NoneType
    best_acc: float
    best_model_path: str
    bias_constraint: NoneType
    bias_initializer
    bias_regularizer: NoneType
    cm_file_path
    dropout
    input_shape
    interval: str
    kernel_constraint: NoneType
    kernel_initializer
    kernel_regularizer: NoneType
    model: NoneType
    num_cats: int
    recurrent_constraint: NoneType
    recurrent_dropout
    recurrent_initializer
    recurrent_regularizer: NoneType
    tensorboard_callbacks: list
    tensorboard_log_dir: str
    tuner: Hyperband, NoneType
    unroll
    use_bias

build_model(lp: HyperParameters)
int (interval: str, tensorboard_log_dir: str)
model_generator(input_shape: tuple[int, int], output_shape: int): None
update_best_arg(acc: float, res: dict): None
best_model(): str
confusion_matrix(filename: str, y_true: np.ndarray, y_pred: np.ndarray, tags: list[str]): str
get_best_acc(): float
get_confusion_matrix(): str
get_log_dir(): str
predict(X: np.ndarray): np.ndarray
save_model(): None
stats(): str
train(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, epochs: int, batch_size: int, log_dir: str, lparams: dict): None
train_with_lparams(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, X_test: np.ndarray, y_test: np.ndarray, epochs: int, batch_size: int, num_cats: int, categories: list[str]): None

```

Figura 3.27: Estructura de la clase RNNTrainer

confundida con saludo). Los resultados del resto de intervalos se pueden ver en el apéndice E.

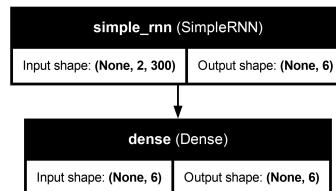


Figura 3.28: Esquema del modelo RNN

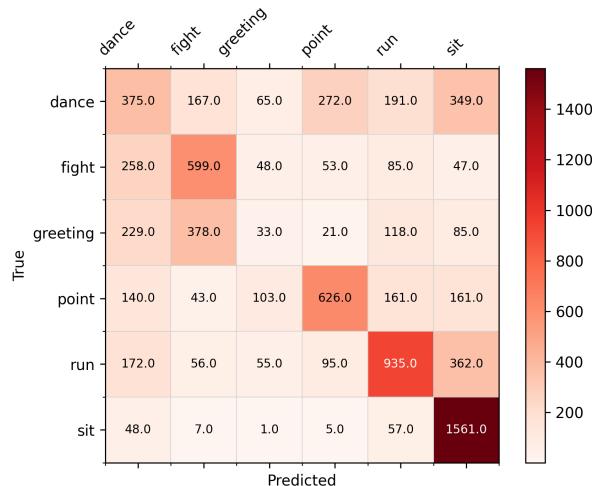


Figura 3.29: Matriz de confusión del modelo RNN

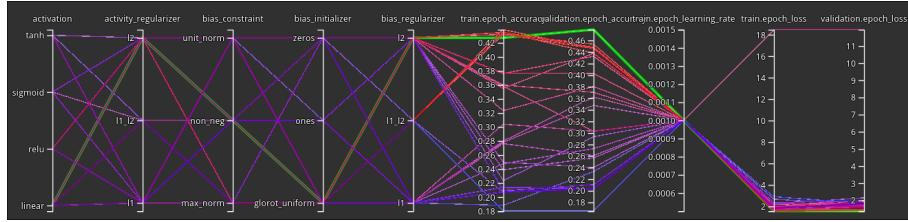


Figura 3.30: Gráfico de entrenamiento del modelo RNN

3.5.3.4. Random Forest

El entrenador del [Random Forest](#) es el más sencillo y el más distinto de todos. Este modelo no es una red neuronal, sino un modelo clásico de clasificación. La clase *RandomForestTrainer* (estructura en la figura 3.31) sigue utilizando la función *train_with_hparams* como las anteriores aún no habiendo *hparams* como tal, haciendo esto se consigue mantener la igualdad de todas las clases de cara a la interfaz de gradio.

Este modelo no usa tensorflow como tal, dado que YDF es un desarrollo que deja atrás el uso de TensorFlow Decision Forest para implementar algoritmos más eficientes tanto en entrenamiento como en inferencia. A pesar de ser un desarrollo paralelo, el equipo de YDF decide hacer el modelo compatible con Tensorflow para poder integrarlo en un mayor ecosistema. Estos datos de mayor eficiencia se pueden comprobar en el estudio [Guillame-Bert et al. \(2023\)](#).

A diferencia de los modelos anteriores, este modelo usa DataFrames de pandas para el entrenamiento y la predicción en vez de tensores. En este entrenamiento usamos *RandomSearchTuner* para buscar la mejor combinación de número de árboles, profundidad máxima y mínimo de ejemplos.

RandomForestTrainer
__best_accuracy : float
__best_model_path : str
__cm_file_path : NoneType
__interval : str
__log_dir : str
__model : NoneType
__tensorboard_callbacks : list

__init__(interval: str, log_dir: str): None
__best_model(): str
get_best_acc(): float
get_confusion_matrix(): str
get_log_dir(): str
plot_confusion_matrix(filename: str, y_true: np.ndarray, y_pred: np.ndarray, tags: list[str]): str
predict(X: pd.DataFrame): np.ndarray
save(model): None
stats(): str
train(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, categories: list[str]): None
train_with_hparams(X: np.ndarray, y: np.ndarray, X_val: np.ndarray, y_val: np.ndarray, X_test: np.ndarray, y_test: int, batch_size: int, num_cats: int, categories: list[str]): None

Figura 3.31: Estructura de la clase RFTrainer

Por desgracia, este modelo no deja una constancia tan rigurosa como los anteriores dado que no usa TensorBoard ni tiene un equivalente. A pesar de esto, este es el modelo con mayor rendimiento de lejos. Todos los intervalos tienen un rendimiento más o menos equivalente, ganando el de 0.8 segundos por muy poco.

En la figura 3.32 se puede ver la matriz de confusión del modelo y en el apéndice G se pueden ver los resultados de todos los intervalos. En este caso podemos ver que los gestos que peor detecta son pelea y saludo. La precisión en validación fue de 0.85936, lo que lo convierte en el mejor modelo entrenado.



Figura 3.32: Matriz de confusión del modelo RandomForest

Este modelo finalmente es exportable a TensorFlow, siendo esto muy útil para su despliegado y compatibilización con otras herramientas. Dentro del repo, en la carpeta de “notebooks” se puede ver un resumen de las características más decisivas del modelo y su importancia en la diferenciación de dos gestos cuales sean.

3.5.4. TensorFlow Serving

Que todos los modelos sean exportables a TensorFlow permite el uso de TensorFlow Serving ([Olston et al. \(2017\)](#)) para desplegar modelos. Serving nos proporciona una interfaz sencilla mediante [API REST](#) para re entrenar los modelos y poder hacer inferencia sobre ellos desde cualquier máquina conectada a la red.

Este método de despliegado es especialmente útil a la hora de integrar los modelos con otras tecnologías sin necesidad de hacer traducciones a otros lenguajes o formatos. El servidor se despliega usando docker y cogiendo el modelo de la última *release* del repositorio. Para hacer una inferencia solo hay que hacer una petición *POST* con el formato visto en la figura 3.33 y el servidor devuelve la respuesta vista en la figura 3.34.

```

1 curl <model_server>/v1/models/<model_name>:predict \
2 -X POST \
3 -d '{
4   "instances": [
5     {
6       "feature_0":0.24,
7       "feature_1":0.12,
8       ...
9       "feature_599":0.56
10    }
11  ]}'
```

Figura 3.33: Ejemplo de petición a TensorFlow Serving

```

1 {
2     "prediction": {
3         "scores": [0.981395662, 0.0186043456, 0.1920000, 0.0112349,
4             0.0001234, 0.0001234],
5         "classes": ["dance", "fight", "greeting", "point", "run",
6             "sit"]
7     }

```

Figura 3.34: Ejemplo de respuesta de TensorFlow Serving

En el repositorio se pueden encontrar dos scripts, uno para windows y otro para linux, que permiten desplegar el servidor de manera automática siempre que docker ya esté instalado en el sistema.

3.5.5. Hardware, complejidad de datos y otras limitaciones

Este estudio ha estado muy limitado por el hardware disponible para el mismo. El equipo usado para el modelo tenía gráficas de consumidor de hace más de 5 años, haciendo que los modelos con redes neuronales tardaran bastante en entrenarse. Esto no fue un problema en el caso del [Random Forest](#), ya que se entrena en [CPU](#).

Estas limitaciones de hardware también han afectado a la visión de los modelos a desarrollar, ya que en un principio se pensó en hacer modelos lo más simples posibles para poder utilizarlos incluso en equipos más limitados (gafas de [VR](#) o equipos más antiguos).

Así mismo, la falta de espacios y de tiempo han resultado en que, junto a la falta de datasets ya formados, el número de datos fuera limitado para lo que requeriría el entrenamiento de redes neuronales.

3.6. Aplicación final

Una vez los modelos estaban preparados era necesaria una aplicación a modo de demostración que se pueda ejecutar en las gafas de [VR](#) y con el traje de caputra de movimiento. La aplicación consta de un panel en el que introduces la IP de la máquina que tiene el servidor de Tensor ejecutándose con el modelo y la aplicación de Axis Studio para poder utilizar el traje. Además, tiene un panel de texto en el que aparecen los resultados de la predicción y un NPC que reacciona a tus gestos. En la figura 3.35 se puede ver una captura de esa aplicación.



Figura 3.35: Captura de la aplicación final desde el editor de Unity

3.6.1. Funcionamiento de la aplicación

Al introducir la IP en el cuadro de texto la aplicación se conecta con el traje y el servidor de Tensor. La conexión con el traje es la misma que la del apartado 3.1.2, mientras que para el servidor se utiliza una [API REST](#) mediante UnityWebRequest,¹¹ una API de Unity para comunicarse con servidores web. Esta ejecución se hace mediante una corutina, la cual al principio espera la cantidad de tiempo óptima entre frame y frame que se determinó en el entrenamiento del modelo usado (en nuestro caso el Random Forest con 0.8 segundos).

La información de los huesos se guarda en una cola, donde se guarda la información actual y, una vez enviado al servidor, se elimina la más antigua, siendo siempre el tamaño de la cola de máximo dos elementos. Al pasar estos segundos la aplicación manda al servidor la información de los huesos en el frame actual y en el frame de hace 0.8 segundos (previamente guardado) en formato JSON mediante POST con protocolo HTTP, a lo que el servidor le responde con dos listas en formato JSON: una con los gestos y otra con el score que ha conseguido cada gesto en la predicción, coincidiendo el índice del score de un gesto con el índice del gesto de la anterior lista. Para hacer todas las conversiones entre cadenas de texto y el formato JSON se ha usado la clase JsonUtility nativa en Unity.

Una vez se tenga la respuesta del servidor empieza de nuevo la corrutina, se ordena de mayor a menor score las predicciones mediante bubble sort para escribirlas en el panel correspondiente y se ejecuta la animación de respuesta del [NPC](#). Las animaciones de respuesta que tiene a los gestos se puede ver en la tabla 3.2

¹¹Página de la documentación de Unity: <https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html>

Gesto	Animación de reacción
Bailar	Aplaudir
Saludar	Saludar
Señalar	Mirar
Sentarse	Sentarse
Pelear	Pose defensiva
Correr	Correr

Tabla 3.2: Reacción del **NPC** al gesto predicho del jugador

Si hay un error de conexión en mitad de la ejecución se deberá ingresar de nuevo la IP de la máquina host en el campo de texto para reintentarlo.

En la figura 3.36 se puede ver el diagrama de flujo de esta aplicación.

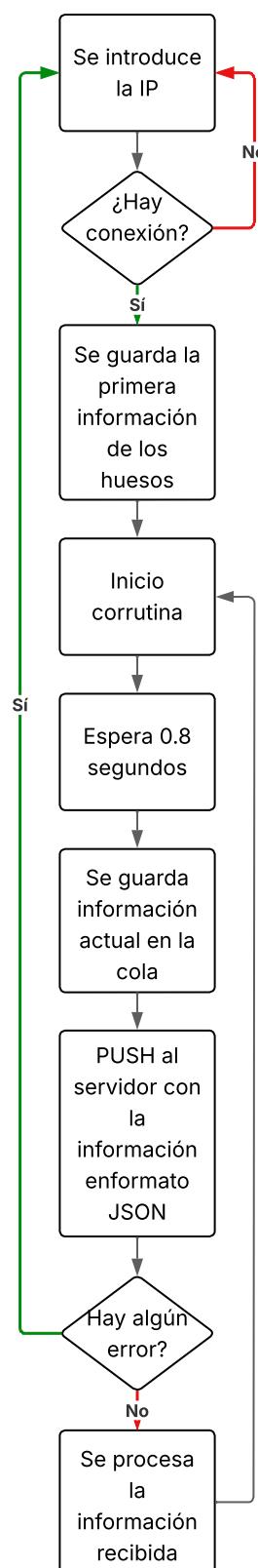


Figura 3.36: Diagrama de flujo de la aplicación de demostración

Conclusiones y Trabajo Futuro

4.1. Conclusiones

Este estudio presenta la comparativa de tres redes neuronales básicas y un [Random Forest](#) para la clasificación de poses realizadas por un humano llevando un traje de captura de movimiento. Teniendo en cuenta las limitaciones del traje (las personas con algún tipo de discapacidad del aparato motor, por ejemplo la falta de alguna extremidad o miembro, no pueden usar correctamente el traje), se ha conseguido un modelo relativamente ligero que detecta las poses de todo el dataset con un gran nivel de acierto.

Este estudio también ha tenido sus limitaciones, como la falta de datos de entrenamiento, la falta de precedentes en el uso de trajes de captura de movimiento para entrenar modelos de [IA](#) y el uso de hardware de consumidor en vez de uno más profesional. Todos estos problemas se han ido subsanando con la ayuda de los voluntarios de los experimentos, el conocimiento adquirido durante la carrera y la ayuda de la asociación LAG al ceder equipos para el entrenamiento.

Este estudio puede servir para la investigación de otros modelos que se centren más en las diferencias emocionales entre mismos tipos de gesto, diferencias culturales entre distintos tipos de gestos o otros temas relacionados con movimientos humanos.

4.2. Trabajo Futuro

El trabajo a futuro de este estudio se puede centrar en varios esfuerzos:

- Mejorar el dataset, incluyendo gente con diversidad funcional, aumentando el número de muestras por gesto y añadiendo nuevas categorías.
- Investigar otros modelos de [IA](#) que puedan mejorar la precisión de la clasificación y hacer clasificación no solo de gestos, sino también de emociones.
- Estandarizar el servidor para que pueda ser usado con otras tecnologías de [IA](#) y no solo con TensorFlow.

Introduction

“La revolución industrial y sus consecuencias han sido un desastre para la raza humana”
— Theodore Kaczynski

In recent years, there has been a great interest and evolution of virtual reality technologies led by companies such as Apple with the launch of Apple Glasses or Meta with the launch of Meta Quest 3 or its interest in the [metaverse](#) with its application “Meta Horizon Worlds”.

It is for this reason that it is interesting to investigate non-verbal communication in virtual environments, not only for applications with multiple users but also to improve interactions with [NPCs](#) in these types of environments.

The proposal in this Bachelor’s Thesis is a first approach to how we can use Artificial Intelligence and motion capture technology to achieve this improvement in interactions in virtual worlds through non-verbal communication, being our case the exploration of the ability to classify different gestures. The gestures we have focused on are six gestures that we have considered useful for an NPC to recognize in a video game. These gestures are:

1. Dance
2. Greet
3. Point
4. Sit down
5. Fight
6. Run

To carry out this work, we have used the Meta (formerly Oculus) Quest 2 and 3 virtual reality glasses and the Perception Neuron 3 motion capture suit, from the company Noitom, as hardware and Python, C# and Unity as the software requirements.

4.3. Motivation

Research on gesture recognition using [Artificial Intelligence \(AI\)](#) for possible implementations in the study and improvement of non-verbal communication in virtual environments.

4.4. Objectives

Implementation of an [AI](#) model that, with low latency, allows identifying the gesture being performed with a motion capture suit.

4.5. Work Plan

The Work Plan consists of several steps:

1. Search for a dataset: generate a dataset large enough with several examples of gestures to adequately train different models.
2. Implementation of [AI](#) models: implementation of several [AI](#) models to make a comparison between them and decide which one is the most suitable considering its prediction speed and accuracy.
3. Development of a final application: development of an application for the Meta Quest as a demo that connects to the chosen model and allows real-time usage.

Conclusions and Future Work

4.6. Conclusions

This study presents a comparison of three basic neural networks and a [Random Forest](#) for the classification of poses performed by a human wearing a motion capture suit. Taking into account the limitations of the suit (people with some type of motor disability, such as the absence of a limb, cannot use the suit properly), a relatively lightweight model has been achieved that detects the poses in the entire dataset with a high level of accuracy.

This study also had its limitations, such as the lack of training data, the lack of precedents in the use of motion capture suits to train [AI](#) models, and the use of consumer-grade hardware instead of more professional equipment. All these issues were mitigated with the help of the experiment volunteers, the knowledge acquired during the degree, and the support of the LAG association by providing equipment for training.

This study can serve as a basis for the research of other models that focus more on emotional differences between the same types of gestures, cultural differences among different types of gestures, or other topics related to human movement.

4.7. Future Work

The future work of this study can focus on several efforts:

- Improve the dataset by including people with functional diversity, increasing the number of samples per gesture, and adding new categories.
- Investigate other [AI](#) models that could improve classification accuracy and enable the classification not only of gestures but also of emotions.
- Standardize the server so it can be used with other [AI](#) technologies and not just TensorFlow.

Contribuciones Personales

Alejandro Barrachina Argudo

Alejandro Barrachina, estudiante del Grado de Ingeniería Informática, se ha involucrado en el desarrollo del estudio desde el momento en que se le propuso la idea. Desde el principio, ha estado en comunicación constante con su compañero y sus tutores para comunicar los avances y dudas del proyecto. A lo largo del proyecto, estas son las tareas que ha desempeñado:

En un primer momento, se dedicó a estudiar el funcionamiento de Unity en gafas de VR y su posible integración con el traje de captura de movimiento y los modelos de IA pensados. Esto implicó aprender sobre Unity, C#, y servidores en Unity.

Una vez tuvo claro el comportamiento y funcionamiento de Unity, junto a su compañero, se dedicó a buscar posibles datasets para entrenar los modelos de IA. Tras varias búsquedas, cuando se optó por crear un dataset a mano, se dedicó a buscar herramientas para automatizar el proceso.

Al encontrar la herramienta Mixamo Downloader, la analizó hasta encontrar los errores de funcionamiento de la misma. Tras arreglarlos y añadir nuevas funcionalidades, se dedicó a crear la primera versión del dataset con los datos descargados, decidiendo la estructura de carpetas y su método de guardado.

Para este método de guardado, se decidió usar Kaggle para evitar problemas con Github y su límite de tamaño por archivo. Alejandro se ha encargado también de la estructura de repositorios y herramientas en la organización creada para este TFG, *FratosVR*.

Mientras Pablo Sánchez desarrollaba la herramienta de conversión de datos, Alejandro investigó como hacer un *pipeline* de descarga del dataset, conversión al formato deseado usando Unity y su posterior actualización en el nuevo conjunto de datos de Kaggle. Para ello tuvo que investigar las funcionalidades de línea de comando de Unity.

Una vez creados estos datos, Alejandro discutió con sus tutores cual sería la mejor manera de estandarizar los datos para su uso en los entrenamientos de los modelos. Cuando se llegó a un formato concreto, Alejandro se dedicó a programar la clase DataLoader para procesar todos los datos, estandarizarlos y actualizarlos en la nube para su posterior uso.

Tras esto, Alejandro modeló los cuatro modelos a probar y comparar. Esto im-

plicó el uso de Gradio y TensorBoard para facilitar el entrenamiento para usuarios ajenos al proyecto. Esta idea se desarrolló por la ayuda cedida por la asociación LAG, que consistió en dejarnos equipos con gráficas más potentes que las que teníamos disponibles para así acelerar el entrenamiento de los modelos.

Mientras que Pablo se dedicaba a la adaptación de la herramienta para su uso con personas reales, Alejandro hizo dos formularios para las pruebas con usuarios. Uno de ellos para que las personas interesadas dejaran su correo y las fechas que tenían disponibles, y otro para la posterior recogida de datos demográficos. Estos formularios se pasaron a María del Carmen Fernández Villalba, Responsable de Protección de datos en Vicepresidencia Primera, perteneciente a Presidencia de La Junta de Comunidades de Castilla La Mancha, para su revisión y aprobación. Esto se hizo para asegurar que ninguno de los formularios incumplía la ley de protección de datos y así asegurar el anonimato de los datos recogidos y de los usuarios participantes. Tras sus sugerencias, se realizaron los cambios pertinentes y se abrieron para respuesta.

De manera adicional, empezó a plantear los diferentes modelos a entrenar y su representación gráfica en la web de entrenamiento. Para ello, investigó YDF, ya que era una herramienta nueva que no había usado, y desarrolló los modelos con TensorFlow de manera que fueran todos iguales de cara a la interfaz.

Una vez Pablo adaptó la herramienta para las pruebas con usuarios, Alejandro creó parte de los carteles para la difusión de las pruebas. Junto a Pablo hicieron una ruta por todas las facultades de la UCM de Ciudad Universitaria para colgarlos. También inició una campaña por sus redes sociales junto a las asociaciones de estudiantes de la facultad de informática para atraer a más gente.

Durante las pruebas, tanto Alejandro como Pablo estuvieron presentes en todas para ayudar a los usuarios a ponerse el traje, realizar el calibrado del mismo y recoger las preguntas del cuestionario. Durante este proceso de pruebas, tanto Alejandro como Pablo se encargaron de atraer usuarios de prueba nuevos en caso de que fallaran los que tenían prueba asignada, resultando en que solo 5 plazas no fueran completadas. También se encargaron de gestionar horarios para maximizar el número de pruebas realizadas y gestionar la espera de los usuarios que se solaparan.

Tras esto, Alejandro se encargó de los entrenamientos de los modelos y los problemas que surgieron durante el proceso (Falta de VRAM, problemas de ingestión de datos, etc). También se encargó de desplegar los mecanismos de entrenamiento en distintos sistemas para no ocasionar inconvenientes a los usuarios de la asociación. Esto consistió en montar [Windows Subsystem for Linux \(WSL\)](#) en los ordenadores de la asociación para compatibilizarlos con las últimas versiones de TensorFlow.

Una vez se acabaron los entrenamientos y se decidió el mejor modelo, Alejandro investigó una manera de hacer que estos modelos fueran multiplataforma y pudieran ser usados en cualquier dispositivo. Esto resultó en el descubrimiento e investigación de TensorFlow Serving. Alejandro posteriormente hizo scripts para Windows y Linux para facilitar el despliegado del modelo en docker usando las *releases* de Github.

Tras finalizar todo esto, Alejandro se dedicó a la redacción de este documento junto a su compañero, garantizando así cohesión durante todo el texto y ayuda con L^AT_EXen los momentos necesarios. También se dedicó a hacer las gráficas explicativas de los modelos y los datos demográficos para asegurar un entendimiento más sencillo

de todos los resultados.

Pablo Sánchez Martín

Pablo Sánchez Martín, estudiante del Grado de Desarrollo de Videojuegos, se ha involucrado en el proyecto desde el momento en el que salió la idea manteniendo una comunicación constante con su compañero y tutores.

Desde el año 2023 mantuvo contacto con uno de los tutores, Alejandro Romero, para hacer un proyecto con un traje de captura de movimiento. Tras varias reuniones debatiendo lo que podría ser un trabajo interesante los dos decidieron el estudio actual. Para ello Pablo contactó al otro tutor del proyecto, Ismael Sagredo, e involucró a su compañero Alejandro Barrachina.

Durante el desarrollo estas son las tareas que ha realizado:

La primera tarea que tuvo que realizar junto a su compañero Alejandro era establecer qué gestos se querían detectar, qué modelos de [IA](#) se iban a utilizar y si se iba a detectar el movimiento de los dedos mediante las gafas de [VR](#) (handtracking). Finalmente los dos acordaron los cinco gestos en los que se basa el proyecto, los cuatro modelos con los que se hace la comparación y la decisión de no tener el handtracking por posibles problemas a la hora de no estar viendo las manos en todo momento.

Lo siguiente que hizo fue investigar el funcionamiento del traje de captura de movimiento junto al software necesario, tanto Axis Studio como el plugin de Unity. Para ello estuvo investigando las distintas funcionalidades dentro de un proyecto de Axis Studio que podría interesar para el estudio y se descargó un proyecto cedido por el tutor Alejandro Romero en el que se usaba el plugin para poder analizarlo y saber cómo usarlo.

Una vez entendió el funcionamiento del software necesario para el funcionamiento del traje empezó a buscar distintos datasets de animaciones para entrenar los modelos junto a su compañero. Al ver la escasez de estos datasets se optó por sacarlos de un banco de animaciones, por lo que empezó el desarrollo de la herramienta Mixamo Dumper.

En un primer momento investigó sobre el uso de los Assets Bundles de Unity para la carga automática de recursos externos a la aplicación, pero al final decidió no usar ese método. Finalmente para esa aplicación decidió usar el Asset Database de Unity, por lo que se dedicó a la investigación de esta [API](#) y finalmente a la implementación de la herramienta. Una vez funcionaba la implementación de la carga de animaciones en tiempo de ejecución investigó como poder ejecutar todas estas animaciones seguidas en tiempo de ejecución, hasta que finalmente encontró el componente "Animator Override Controller", lo que le permitió cumplir el objetivo de la herramienta.

Debido a los pocos datos encontrados en Mixamo se decidió grabar a usuarios con el traje de captura de movimiento, por lo que hizo en Unity una herramienta sencilla para crear [CSVs](#) en tiempo real con el traje de captura de movimiento.

Para las pruebas con usuarios se encargó de la comunicación con sus tutores y personal de la Facultad de Informática para conseguir un lugar en el que poder

grabar. Con el fin de capar usuarios junto a su compañero hizo carteles y ayudó en la campaña en redes sociales y difusión por el resto de facultades de Ciudad Universitaria.

En la prueba explicó junto a su compañero a todos los usuarios el propósito de las pruebas, les ayudó a ponerse el traje y a explicarles el proceso de calibración. También le fue dando indicaciones a los usuarios sobre los gestos a realizar, grabó las animaciones y verificó que no hubiese ningún problema con estas. Adicionalmente se encargó junto a su compañero de buscar nuevos usuarios para cubrir huecos que habían quedado libres con el fin de maximizar el número de datos recogidos y gestionar a las personas que solapaban con otros usuarios.

Una vez recogidos los datos investigó diferentes formas de conectarse con el servidor que hostee el modelo elegido. Investigó el uso de "Unity NetCode", un paquete propio de Unity para la comunicación en red en videojuegos y también investigó la posibilidad de realizar la comunicación mediante las llamadas nativas de C#. Finalmente se decidió usar la arquitectura [API REST](#), por lo que investigó su funcionamiento mediante el paquete nativo "UnityWebRequest" y empezó la aplicación final con la implementación de ésta en Unity.

Una vez estaba hecha la comunicación con el servidor diseñó e implementó una aplicación final con soporte para las gafas de [VR](#) y el traje de captura de movimiento en el que se conectase con el servidor y se pueda ver el resultado del modelo elegido en tiempo real y como un [NPC](#) reacciona a los gestos predichos por éste.

Finalmente ha contribuído a la redacción de este documento junto a su compañero, así como la constante comunicación con los tutores para comprobar que el formato de éste era adecuado y a la realización de distintos diagramas para que las herramientas creadas durante el proceso del proyecto sean más sencillas de entender.

Bibliografía

The sciences, each straining in its own direction, have hitherto harmed us little; but some day the piecing together of dissociated knowledge will open up such terrifying vistas of reality, and of our frightful position therein, that we shall either go mad from the revelation or flee from the deadly light into the peace and safety of a new dark age.

H.P. Lovecraft, *The Call of Cthulhu*

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y. y ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

ABID, A., ABDALLA, A., ABID, A., KHAN, D., ALFOZAN, A. y ZOU, J. Gradio: Hassle-free sharing and testing of ML models in the wild. 2019.

ANSEL, J., YANG, E., HE, H., GIMELSHEIN, N., JAIN, A., VOZNESENSKY, M., BAO, B., BELL, P., BERARD, D., BUROVSKI, E., CHAUHAN, G., CHOUDRIA, A., CONSTABLE, W., DESMAISON, A., DEVITO, Z., ELLISON, E., FENG, W., GONG, J., GSCHWIND, M., HIRSH, B., HUANG, S., KALAMBARKAR, K., KIRSCH, L., LAZOS, M., LEZCANO, M., LIANG, Y., LIANG, J., LU, Y., LUK, C., MAHER, B., PAN, Y., PUHRSCH, C., RESO, M., SAROUFIM, M., SIRACHI, M. Y., SUK, H., SUO, M., TILLET, P., WANG, E., WANG, X., WEN, W., ZHANG, S., ZHAO, X., ZHOU, K., ZOU, R., MATHEWS, A., CHANAN, G., WU, P. y CHINTALA, S. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. En *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 2024.

- BARRACHINA ARGUDO, A. y SÁNCHEZ MARTÍN, P. Csv-pose-animations. 2025a.
- BARRACHINA ARGUDO, A. y SÁNCHEZ MARTÍN, P. Raw mixamo animations. 2025b.
- CARNEGIE MELLON, U. Cmu graphics lab motion capture database. <https://mocap.cs.cmu.edu/info.php>, 2003. Accessed: (04/05/2025).
- ELLIS, S. R. Nature and origins of virtual environments: a bibliographical essay. *Computing Systems in Engineering*, vol. 2, páginas 321–347, 1991.
- GUILLAME-BERT, M., BRUCH, S., STOTZ, R. y PFEIFER, J. Yggdrasil decision forests: A fast and extensible decision forests library. En *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, páginas 4068–4077. 2023.
- HARRIS, C. R., MILLMAN, K. J., VAN DER WALT, S. J., GOMMERS, R., VIRTANEN, P., COURNAPEAU, D., WIESER, E., TAYLOR, J., BERG, S., SMITH, N. J., KERN, R., PICUS, M., HOYER, S., VAN KERWIJK, M. H., BRETT, M., HALDANE, A., DEL RÍO, J. F., WIEBE, M., PETERSON, P., GÉRARD-MARCHANT, P., SHEPPARD, K., REDDY, T., WECKESSER, W., ABBASI, H., GOHLKE, C. y OLIPHANT, T. E. Array programming with NumPy. *Nature*, vol. 585(7825), páginas 357–362, 2020.
- HOLFELD, J. On the relevance of the godot engine in the indie game development industry. 2024.
- JOCHER, G., QIU, J. y CHAURASIA, A. Ultralytics YOLO. 2023.
- LI, L., JAMIESON, K., DESALVO, G., ROSTAMIZADEH, A. y TALWALKAR, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. 2018.
- MA, X., ZENG, B. y XING, Y. Combining 3d skeleton data and deep convolutional neural network for balance assessment during walking. *Frontiers in Bioengineering and Biotechnology*, vol. 11, 2023.
- WES MCKINNEY. Data Structures for Statistical Computing in Python. En *Proceedings of the 9th Python in Science Conference* (editado por Stéfan van der Walt y Jarrod Millman), páginas 56 – 61. 2010.
- OLSTON, C., FIEDEL, N., GOROVY, K., HARMSEN, J., LAO, L., LI, F., RAJASHEKHAR, V., RAMESH, S. y SOYKE, J. Tensorflow-serving: Flexible, high-performance ml serving. 2017.
- O'MALLEY, T., BURSZTEIN, E., LONG, J., CHOLLET, F., JIN, H., INVERNIZZI, L. ET AL. Keras Tuner. <https://github.com/keras-team/keras-tuner>, 2019.
- PALLAVICINI, F., PEPE, A. y MINISSI, M. E. Gaming in virtual reality: What changes in terms of usability, emotional response and sense of presence compared to non-immersive video games? *SIMULATION AND GAMING*, vol. 50, páginas 136–159, 2019.

- PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M. y DUCESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, vol. 12, páginas 2825–2830, 2011.
- PETROCHUK, M. Hparams: Hyperparameter management solution. <https://github.com/PetrochukM/HParams>, 2019.
- RAAB, F. H., BLOOD, E. B., STEINER, T. O. y JONES, H. R. Magnetic position and orientation tracking system. *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-15(5), páginas 709–718, 1979.
- ROETENBERG, D., LUINGE, H., SLYCKE, P. ET AL. Xsens mvn: Full 6dof human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep*, vol. 1(2009), páginas 1–7, 2009.
- PANDAS DEVELOPMENT TEAM, T. pandas-dev/pandas: Pandas. 2020.
- VAN BRAKEL, V., BARREDA-ÁNGELES, M. y HARTMANN, T. Feelings of presence and perceived social support in social virtual reality platforms. *Computers in Human Behavior*, vol. 139, 2023.
- VRCHAT, I. Vr chat full-body tracking. <https://docs.vrchat.com/docs/full-body-tracking>, 2024. Accessed: (03/05/2025).
- WILKINSON, M., BRANTLEY, S. y FENG, J. A mini review of presence and immersion in virtual reality. *Human Factors and Ergonomics Society*, 2021.
- XENAKIS, I., GAVALAS, D., KASAPAKIS, V., DZARDANOVA, E. y VOSINAKIS, S. Non-verbal communication in immersive virtual reality through the lens of presence: A critical review. *PRESSENCE: Virtual and Augmented Reality*, vol. 31, páginas 1–71, 2023.
- ZHANG, Z. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, vol. 19(2), páginas 4–10, 2012.

Apéndice **A**

Cabecera de los [CSVs](#) del dataset

En esta apéndice se presenta la cabecera completa de los [CSVs](#) del dataset.

Tabla A.1: Cabecera del [CSV](#) de cada animación, en orden descendente y de izquierda a derecha (completa)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_Hips_rotx	Robot_Hips_roty	Robot_Hips_rotz
Robot_LeftUpLeg_posx	Robot_LeftUpLeg_posy	Robot_LeftUpLeg_posz
Robot_LeftUpLeg_rotx	Robot_LeftUpLeg_roty	Robot_LeftUpLeg_rotz
Robot_LeftLeg_posx	Robot_LeftLeg_posy	Robot_LeftLeg_posz
Robot_LeftLeg_rotx	Robot_LeftLeg_roty	Robot_LeftLeg_rotz
Robot_LeftFoot_posx	Robot_LeftFoot_posy	Robot_LeftFoot_posz
Robot_LeftFoot_rotx	Robot_LeftFoot_roty	Robot_LeftFoot_rotz
Robot_RightUpLeg_posx	Robot_RightUpLeg_posy	Robot_RightUpLeg_posz
Robot_RightUpLeg_rotx	Robot_RightUpLeg_roty	Robot_RightUpLeg_rotz
Robot_RightLeg_posx	Robot_RightLeg_posy	Robot_RightLeg_posz
Robot_RightLeg_rotx	Robot_RightLeg_roty	Robot_RightLeg_rotz
Robot_RightFoot_posx	Robot_RightFoot_posy	Robot_RightFoot_posz
Robot_RightFoot_rotx	Robot_RightFoot_roty	Robot_RightFoot_rotz
Robot_Spine_posx	Robot_Spine_posy	Robot_Spine_posz
Robot_Spine_rotx	Robot_Spine_roty	Robot_Spine_rotz
Robot_Spine1_posx	Robot_Spine1_posy	Robot_Spine1_posz
Robot_Spine1_rotx	Robot_Spine1_roty	Robot_Spine1_rotz
Robot_Spine2_posx	Robot_Spine2_posy	Robot_Spine2_posz
Robot_Spine2_rotx	Robot_Spine2_roty	Robot_Spine2_rotz

Continua en la siguiente página

Tabla A.1: Cabecera del CSV de cada animación, en orden descendente y de izquierda a derecha (completa) (Continuado)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_LeftShoulder_posx	Robot_LeftShoulder_posy	Robot_LeftShoulder_posz
Robot_LeftShoulder_rotx	Robot_LeftShoulder_roty	Robot_LeftShoulder_rotz
Robot_LeftArm_posx	Robot_LeftArm_posy	Robot_LeftArm_posz
Robot_LeftArm_rotx	Robot_LeftArm_roty	Robot_LeftArm_rotz
Robot_LeftForeArm_posx	Robot_LeftForeArm_posy	Robot_LeftForeArm_posz
Robot_LeftForeArm_rotx	Robot_LeftForeArm_roty	Robot_LeftForeArm_rotz
Robot_LeftHand_posx	Robot_LeftHand_posy	Robot_LeftHand_posz
Robot_LeftHand_rotx	Robot_LeftHand_roty	Robot_LeftHand_rotz
Robot_LeftHandIndex1_posx	Robot_LeftHandIndex1_posy	Robot_LeftHandIndex1_posz
Robot_LeftHandIndex1_rotx	Robot_LeftHandIndex1_roty	Robot_LeftHandIndex1_rotz
Robot_LeftHandIndex2_posx	Robot_LeftHandIndex2_posy	Robot_LeftHandIndex2_posz
Robot_LeftHandIndex2_rotx	Robot_LeftHandIndex2_roty	Robot_LeftHandIndex2_rotz
Robot_LeftHandIndex3_posx	Robot_LeftHandIndex3_posy	Robot_LeftHandIndex3_posz
Robot_LeftHandIndex3_rotx	Robot_LeftHandIndex3_roty	Robot_LeftHandIndex3_rotz
Robot_LeftHandMiddle1_posx	Robot_LeftHandMiddle1_posy	Robot_LeftHandMiddle1_posz
Robot_LeftHandMiddle1_rotx	Robot_LeftHandMiddle1_roty	Robot_LeftHandMiddle1_rotz
Robot_LeftHandMiddle2_posx	Robot_LeftHandMiddle2_posy	Robot_LeftHandMiddle2_posz
Robot_LeftHandMiddle2_rotx	Robot_LeftHandMiddle2_roty	Robot_LeftHandMiddle2_rotz
Robot_LeftHandMiddle3_posx	Robot_LeftHandMiddle3_posy	Robot_LeftHandMiddle3_posz
Robot_LeftHandMiddle3_rotx	Robot_LeftHandMiddle3_roty	Robot_LeftHandMiddle3_rotz
Robot_LeftHandPinky1_posx	Robot_LeftHandPinky1_posy	Robot_LeftHandPinky1_posz
Robot_LeftHandPinky1_rotx	Robot_LeftHandPinky1_roty	Robot_LeftHandPinky1_rotz
Robot_LeftHandPinky2_posx	Robot_LeftHandPinky2_posy	Robot_LeftHandPinky2_posz
Robot_LeftHandPinky2_rotx	Robot_LeftHandPinky2_roty	Robot_LeftHandPinky2_rotz
Robot_LeftHandPinky3_posx	Robot_LeftHandPinky3_posy	Robot_LeftHandPinky3_posz
Robot_LeftHandPinky3_rotx	Robot_LeftHandPinky3_roty	Robot_LeftHandPinky3_rotz
Robot_LeftHandRing1_posx	Robot_LeftHandRing1_posy	Robot_LeftHandRing1_posz
Robot_LeftHandRing1_rotx	Robot_LeftHandRing1_roty	Robot_LeftHandRing1_rotz
Robot_LeftHandRing2_posx	Robot_LeftHandRing2_posy	Robot_LeftHandRing2_posz
Robot_LeftHandRing2_rotx	Robot_LeftHandRing2_roty	Robot_LeftHandRing2_rotz
Robot_LeftHandRing3_posx	Robot_LeftHandRing3_posy	Robot_LeftHandRing3_posz
Robot_LeftHandRing3_rotx	Robot_LeftHandRing3_roty	Robot_LeftHandRing3_rotz

Continua en la siguiente página

Tabla A.1: Cabecera del CSV de cada animación, en orden descendente y de izquierda a derecha (completa) (Continuado)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_LeftHandThumb1_posx	Robot_LeftHandThumb1_posy	Robot_LeftHandThumb1_posz
Robot_LeftHandThumb1_rotx	Robot_LeftHandThumb1_roty	Robot_LeftHandThumb1_rotz
Robot_LeftHandThumb2_posx	Robot_LeftHandThumb2_posy	Robot_LeftHandThumb2_posz
Robot_LeftHandThumb2_rotx	Robot_LeftHandThumb2_roty	Robot_LeftHandThumb2_rotz
Robot_LeftHandThumb3_posx	Robot_LeftHandThumb3_posy	Robot_LeftHandThumb3_posz
Robot_LeftHandThumb3_rotx	Robot_LeftHandThumb3_roty	Robot_LeftHandThumb3_rotz
Robot_Neck_posx	Robot_Neck_posy	Robot_Neck_posz
Robot_Neck_rotx	Robot_Neck_roty	Robot_Neck_rotz
Robot_Head_posx	Robot_Head_posy	Robot_Head_posz
Robot_Head_rotx	Robot_Head_roty	Robot_Head_rotz
Robot_RightShoulder_posx	Robot_RightShoulder_posy	Robot_RightShoulder_posz
Robot_RightShoulder_rotx	Robot_RightShoulder_roty	Robot_RightShoulder_rotz
Robot_RightArm_posx	Robot_RightArm_posy	Robot_RightArm_posz
Robot_RightArm_rotx	Robot_RightArm_roty	Robot_RightArm_rotz
Robot_RightForeArm_posx	Robot_RightForeArm_posy	Robot_RightForeArm_posz
Robot_RightForeArm_rotx	Robot_RightForeArm_roty	Robot_RightForeArm_rotz
Robot_RightHand_posx	Robot_RightHand_posy	Robot_RightHand_posz
Robot_RightHand_rotx	Robot_RightHand_roty	Robot_RightHand_rotz
Robot_RightHandIndex1_posx	Robot_RightHandIndex1_posy	Robot_RightHandIndex1_posz
Robot_RightHandIndex1_rotx	Robot_RightHandIndex1_roty	Robot_RightHandIndex1_rotz
Robot_RightHandIndex2_posx	Robot_RightHandIndex2_posy	Robot_RightHandIndex2_posz
Robot_RightHandIndex2_rotx	Robot_RightHandIndex2_roty	Robot_RightHandIndex2_rotz
Robot_RightHandIndex3_posx	Robot_RightHandIndex3_posy	Robot_RightHandIndex3_posz
Robot_RightHandIndex3_rotx	Robot_RightHandIndex3_roty	Robot_RightHandIndex3_rotz
Robot_RightHandMiddle1_posx	Robot_RightHandMiddle1_posy	Robot_RightHandMiddle1_posz
Robot_RightHandMiddle1_rotx	Robot_RightHandMiddle1_roty	Robot_RightHandMiddle1_rotz
Robot_RightHandMiddle2_posx	Robot_RightHandMiddle2_posy	Robot_RightHandMiddle2_posz
Robot_RightHandMiddle2_rotx	Robot_RightHandMiddle2_roty	Robot_RightHandMiddle2_rotz
Robot_RightHandMiddle3_posx	Robot_RightHandMiddle3_posy	Robot_RightHandMiddle3_posz
Robot_RightHandMiddle3_rotx	Robot_RightHandMiddle3_roty	Robot_RightHandMiddle3_rotz
Robot_RightHandPinky1_posx	Robot_RightHandPinky1_posy	Robot_RightHandPinky1_posz
Robot_RightHandPinky1_rotx	Robot_RightHandPinky1_roty	Robot_RightHandPinky1_rotz

Continua en la siguiente página

Tabla A.1: Cabecera del [CSV](#) de cada animación, en órden descendente y de izquierda a derecha (completa) (Continuado)

Robot_Hips_posx	Robot_Hips_posy	Robot_Hips_posz
Robot_RightHandPinky2_posx	Robot_RightHandPinky2_posy	Robot_RightHandPinky2_posz
Robot_RightHandPinky2_rotx	Robot_RightHandPinky2_roty	Robot_RightHandPinky2_rotz
Robot_RightHandPinky3_posx	Robot_RightHandPinky3_posy	Robot_RightHandPinky3_posz
Robot_RightHandPinky3_rotx	Robot_RightHandPinky3_roty	Robot_RightHandPinky3_rotz
Robot_RightHandRing1_posx	Robot_RightHandRing1_posy	Robot_RightHandRing1_posz
Robot_RightHandRing1_rotx	Robot_RightHandRing1_roty	Robot_RightHandRing1_rotz
Robot_RightHandRing2_posx	Robot_RightHandRing2_posy	Robot_RightHandRing2_posz
Robot_RightHandRing2_rotx	Robot_RightHandRing2_roty	Robot_RightHandRing2_rotz
Robot_RightHandRing3_posx	Robot_RightHandRing3_posy	Robot_RightHandRing3_posz
Robot_RightHandRing3_rotx	Robot_RightHandRing3_roty	Robot_RightHandRing3_rotz
Robot_RightHandThumb1_posx	Robot_RightHandThumb1_posy	Robot_RightHandThumb1_posz
Robot_RightHandThumb1_rotx	Robot_RightHandThumb1_roty	Robot_RightHandThumb1_rotz
Robot_RightHandThumb2_posx	Robot_RightHandThumb2_posy	Robot_RightHandThumb2_posz
Robot_RightHandThumb2_rotx	Robot_RightHandThumb2_roty	Robot_RightHandThumb2_rotz
Robot_RightHandThumb3_posx	Robot_RightHandThumb3_posy	Robot_RightHandThumb3_posz
Robot_RightHandThumb3_rotx	Robot_RightHandThumb3_roty	Robot_RightHandThumb3_rotz

Apéndice B

Carteles para la generación del dataset



Figura B.1: Cartel colgado en la facultad de informática

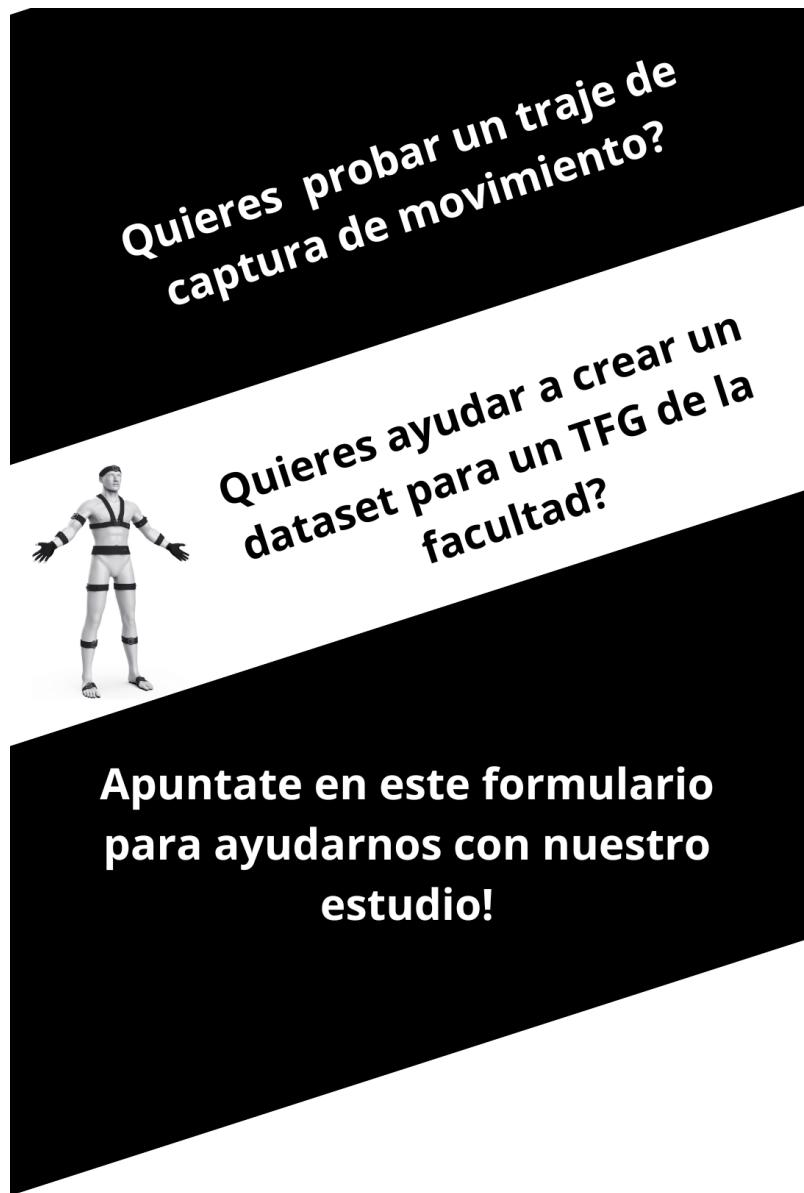


Figura B.2: Cartel colgado en redes sociales



Figura B.3: Cartel colgado en pantallas de la facultad

Apéndice C

Gráficos de la generación del dataset

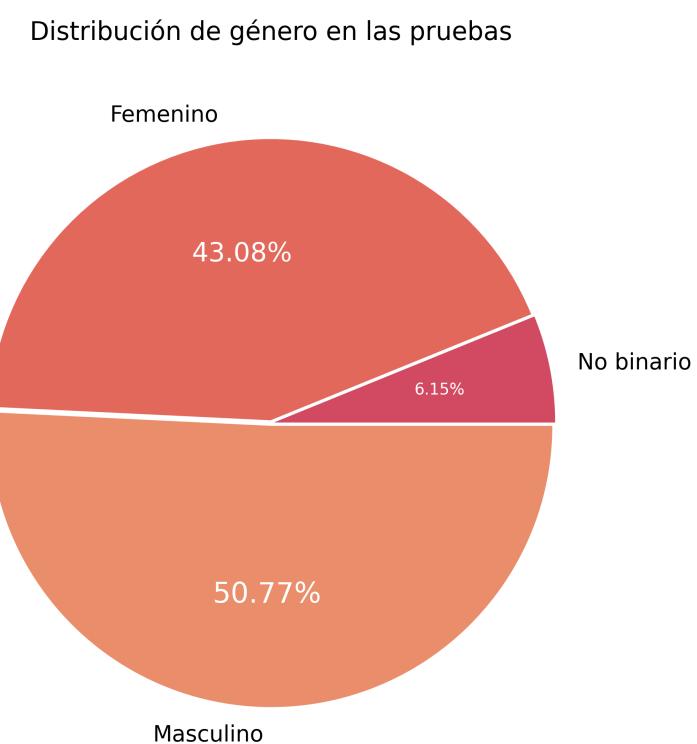


Figura C.1: Distribución de género de los encuestados

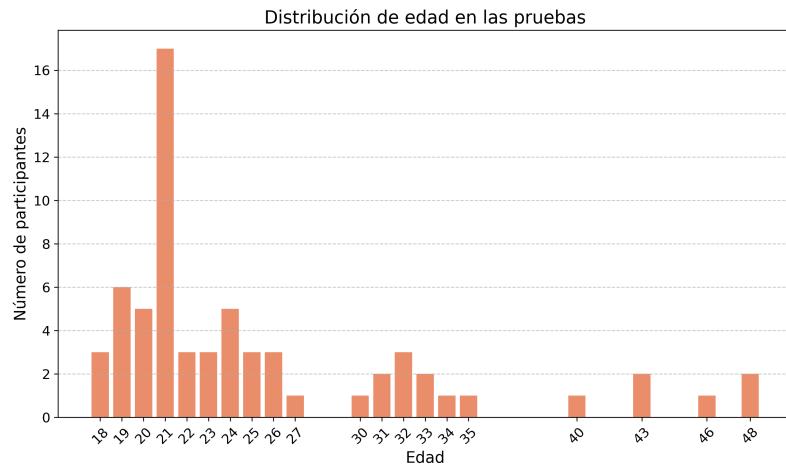


Figura C.2: Distribución de edad de los encuestados

Distribución de nacionalidades en las pruebas

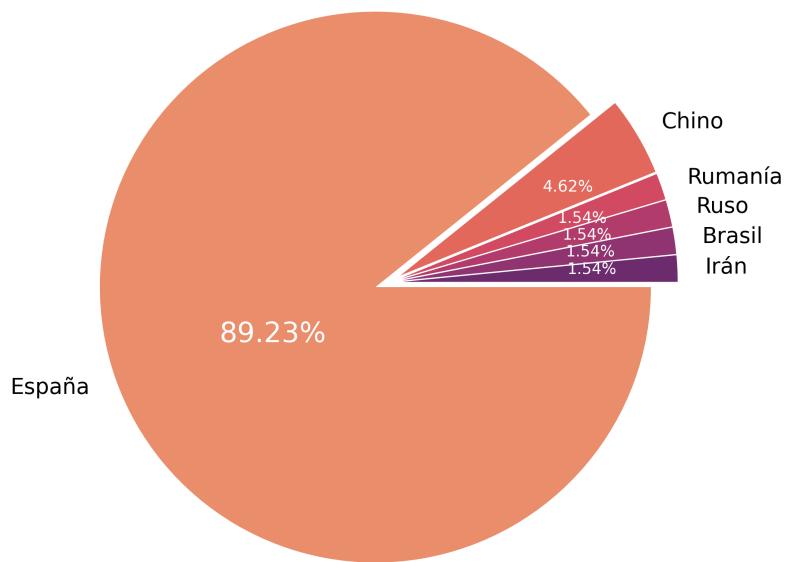


Figura C.3: Distribución de nacionalidad de los encuestados

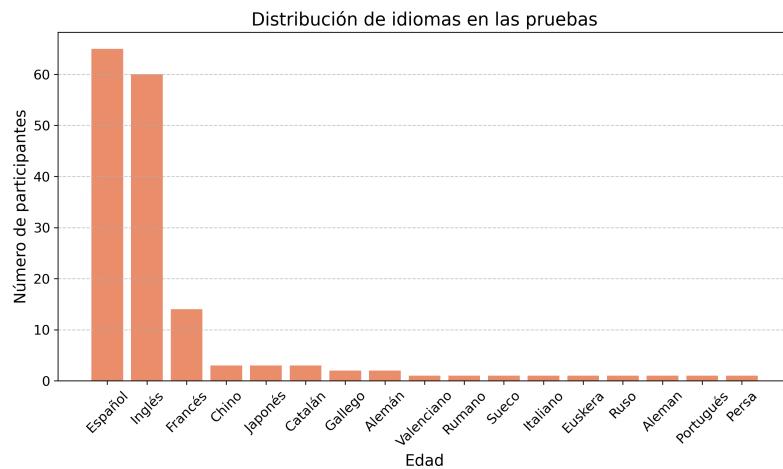


Figura C.4: Distribución de idiomas hablados de los encuestados

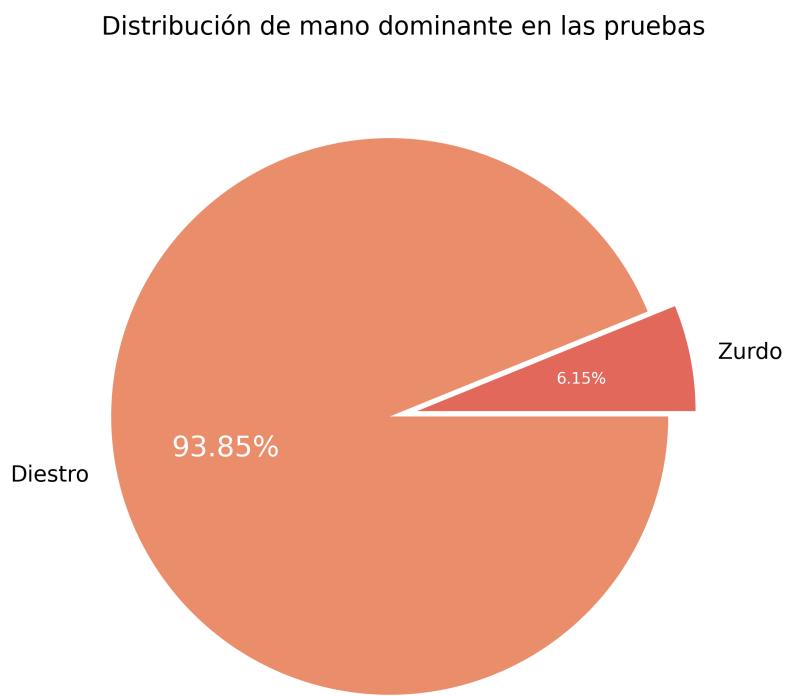


Figura C.5: Distribución de mano dominante de los encuestados

Apéndice D

Resultados de los distintos modelos LSTM

La línea verde en los gráficos de tensorboard indican la mejor combinación de hiperparámetros encontrada en cada caso

D.1. Intervalo 0.2s

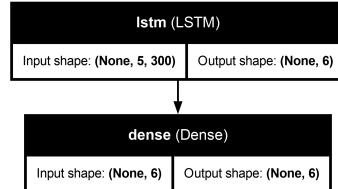


Figura D.1: Esquema del modelo LSTM con 0.2s de intervalo

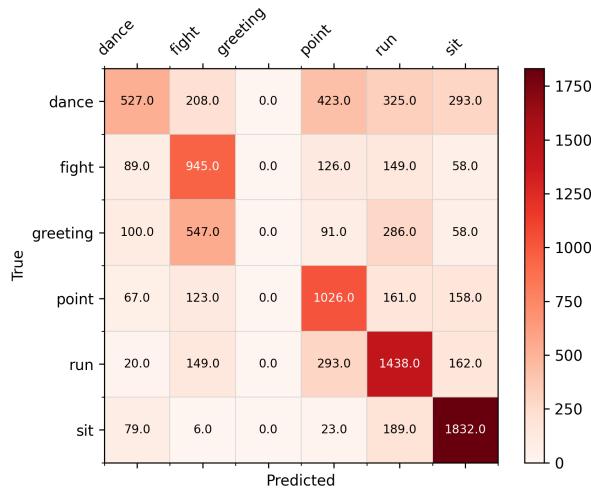


Figura D.2: Matriz de confusión del modelo LSTM con 0.2s de intervalo

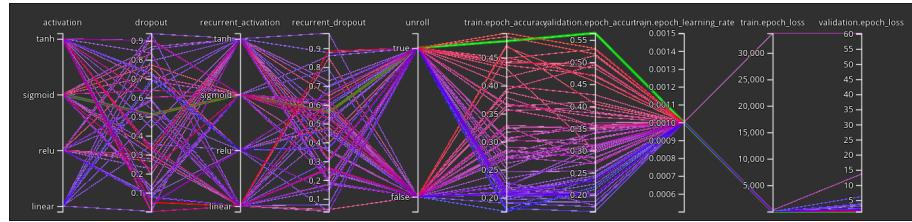


Figura D.3: Gráfico de entrenamiento del modelo LSTM con 0.2s de intervalo (mejor val_accuracy = 0.5655)

D.2. Intervalo 0.4s

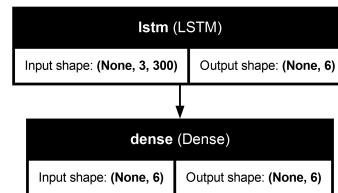


Figura D.4: Esquema del modelo LSTM con 0.4s de intervalo

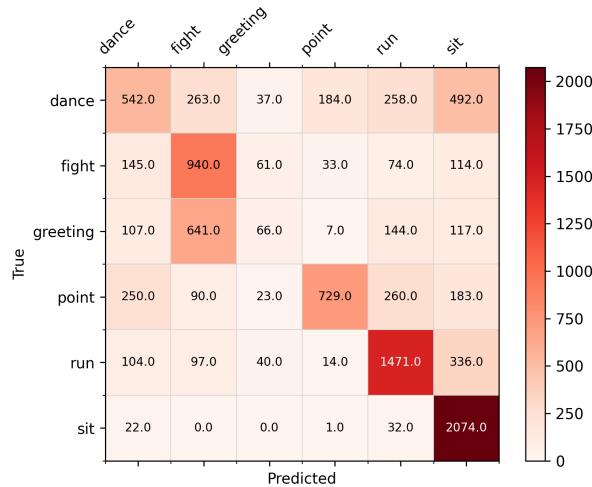


Figura D.5: Matriz de confusión del modelo LSTM con 0.4s de intervalo

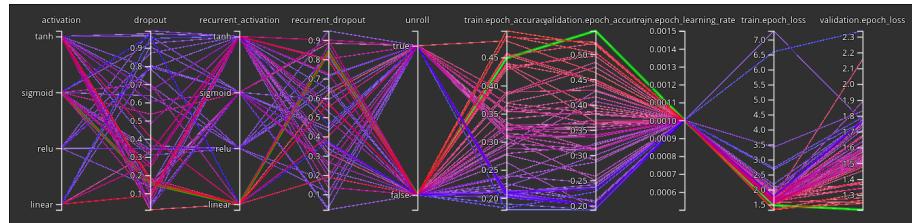


Figura D.6: Gráfico de entrenamiento del modelo LSTM con 0.4s de intervalo (mejor val_accuracy = 0.5462)

D.3. Intervalo 0.6s

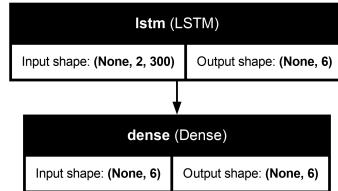


Figura D.7: Esquema del modelo LSTM con 0.6s de intervalo

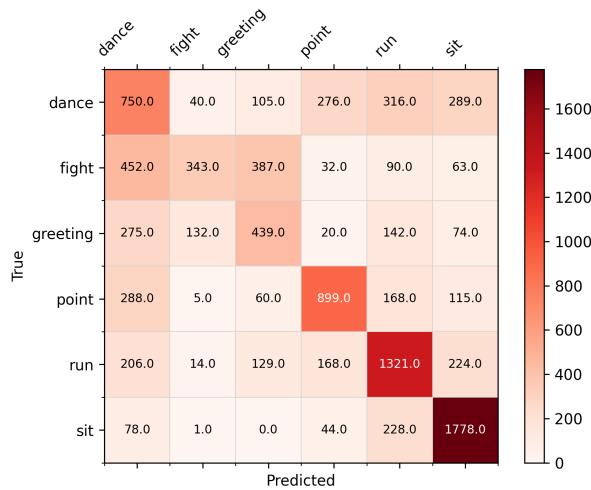


Figura D.8: Matriz de confusión del modelo LSTM con 0.6s de intervalo

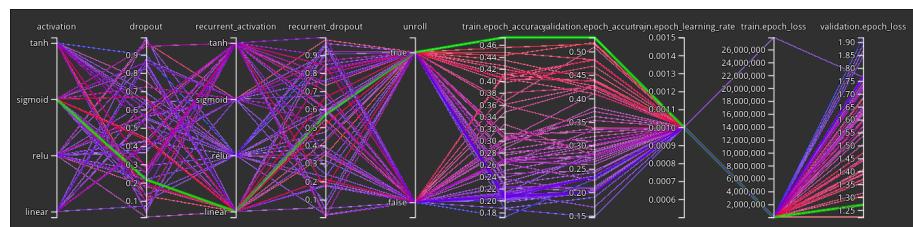


Figura D.9: Gráfico de entrenamiento del modelo LSTM con 0.6s de intervalo (mejor val_accuracy = 0.5301)

D.4. Intervalo 0.8s

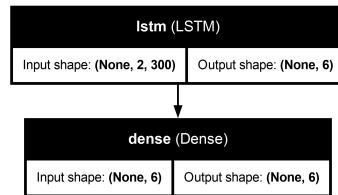


Figura D.10: Esquema del modelo LSTM con 0.8s de intervalo

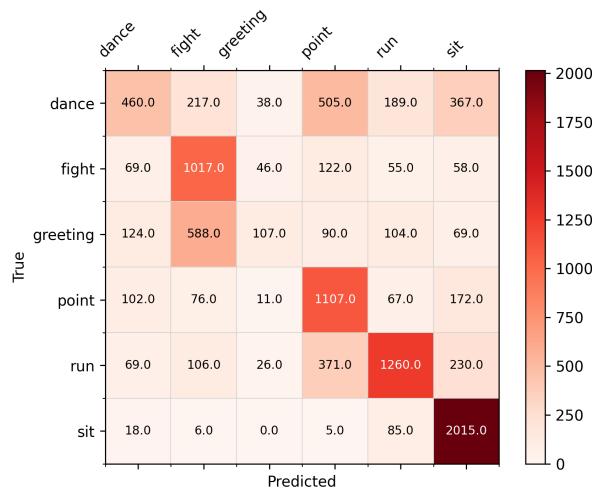


Figura D.11: Matriz de confusión del modelo LSTM con 0.8s de intervalo

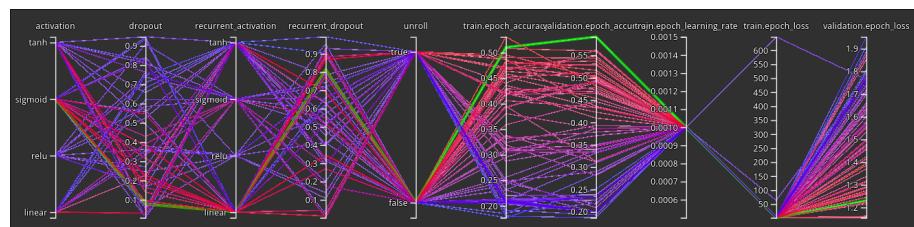


Figura D.12: Gráfico de entrenamiento del modelo LSTM con 0.8s de intervalo (mejor val_accuracy = 0.5912)

D.5. Intervalo 1.0s

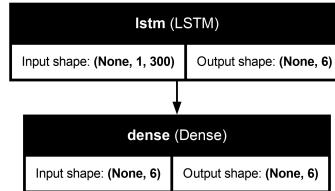


Figura D.13: Esquema del modelo LSTM con 1.0s de intervalo

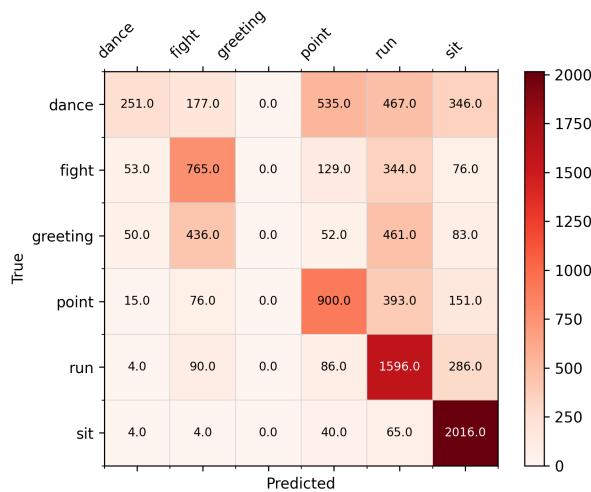


Figura D.14: Matriz de confusión del modelo LSTM con 1.0s de intervalo

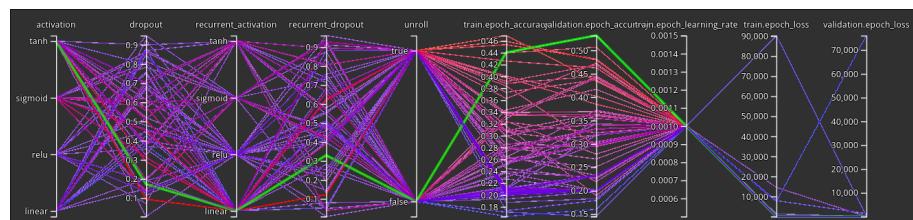


Figura D.15: Gráfico de entrenamiento del modelo LSTM con 1.0s de intervalo (mejor val_accuracy = 0.5318)

Apéndice E

Resultados de los distintos modelos RNN

La línea verde en los gráficos de tensorboard indican la mejor combinación de hiperparámetros encontrada en cada caso

E.1. Intervalo 0.2s

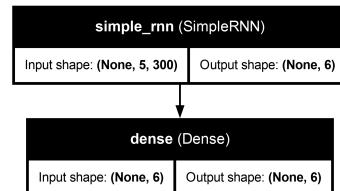


Figura E.1: Esquema del modelo RNN con 0.2s de intervalo

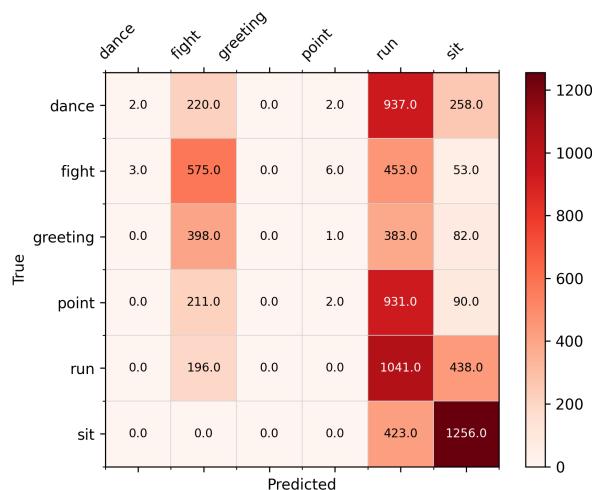


Figura E.2: Matriz de confusión del modelo RNN con 0.2s de intervalo

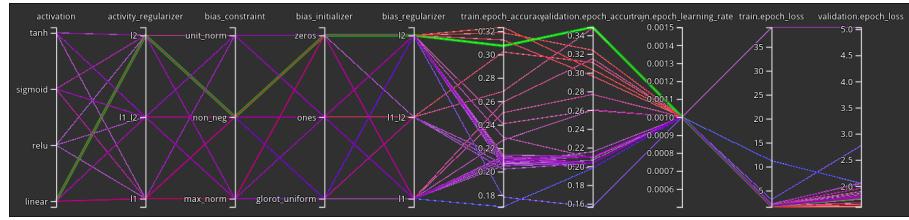


Figura E.3: Gráfico de entrenamiento del modelo RNN con 0.2s de intervalo (mejor val_accuracy = 0.3486)

E.2. Intervalo 0.4s

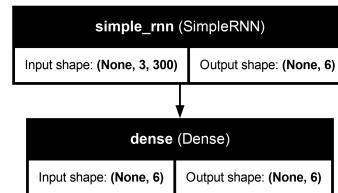


Figura E.4: Esquema del modelo RNN con 0.4s de intervalo

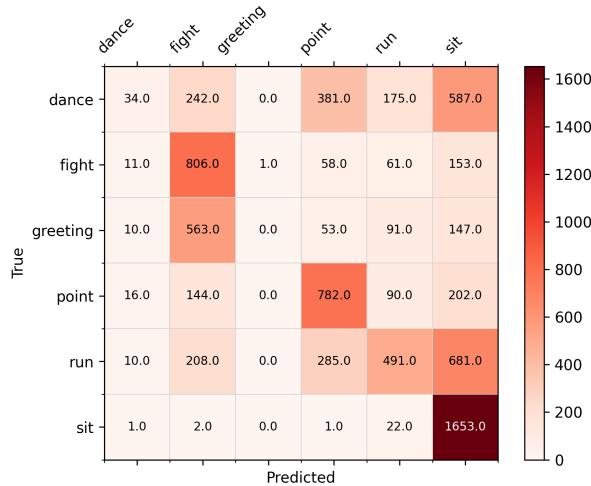


Figura E.5: Matriz de confusión del modelo RNN con 0.4s de intervalo

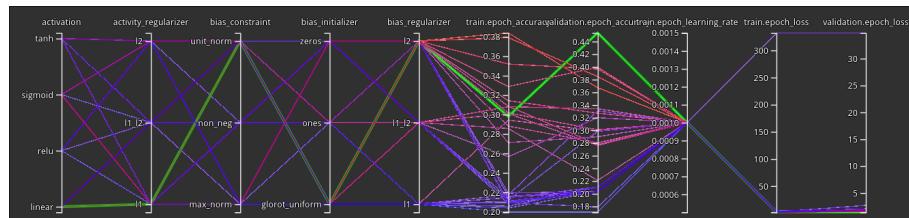


Figura E.6: Gráfico de entrenamiento del modelo RNN con 0.4s de intervalo (mejor val_accuracy = 0.4535)

E.3. Intervalo 0.6s

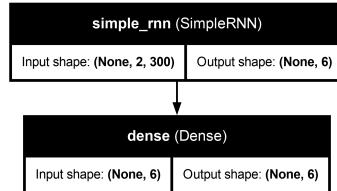


Figura E.7: Esquema del modelo RNN con 0.6s de intervalo

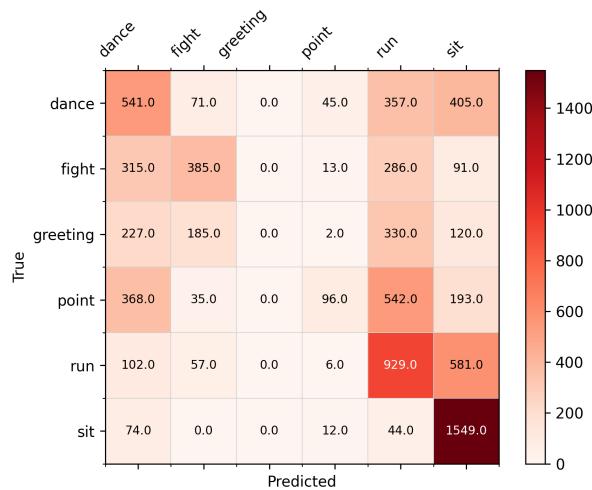


Figura E.8: Matriz de confusión del modelo RNN con 0.6s de intervalo

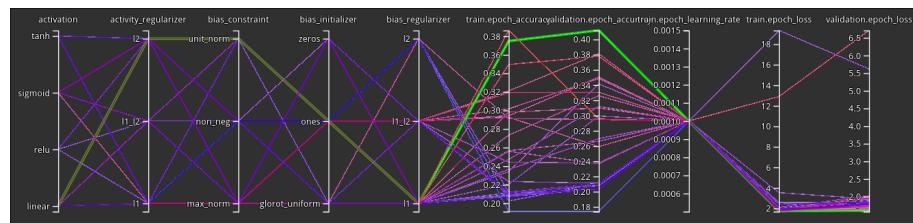


Figura E.9: Gráfico de entrenamiento del modelo RNN con 0.6s de intervalo (mejor val_accuracy = 0.41185)

E.4. Intervalo 0.8s

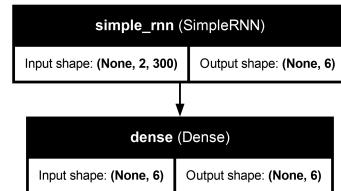


Figura E.10: Esquema del modelo RNN con 0.8s de intervalo

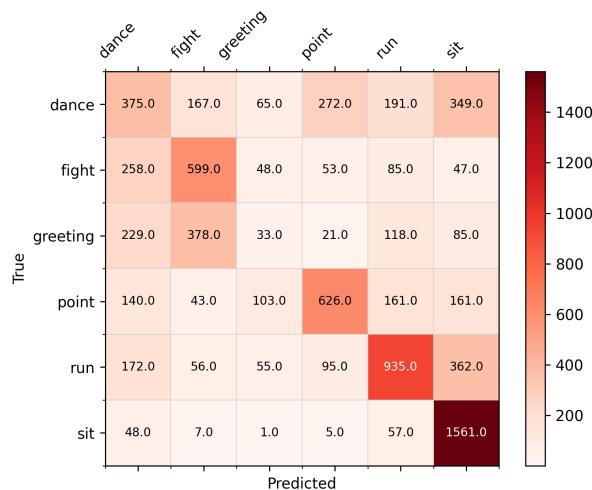


Figura E.11: Matriz de confusión del modelo RNN con 0.8s de intervalo

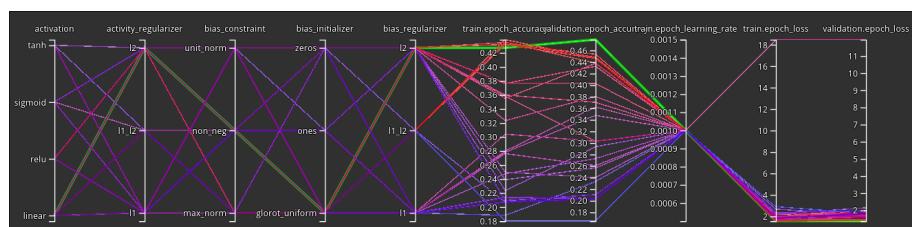


Figura E.12: Gráfico de entrenamiento del modelo RNN con 0.8s de intervalo (mejor val_accuracy = 0.47865)

E.5. Intervalo 1.0s

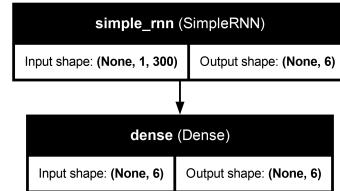


Figura E.13: Esquema del modelo RNN con 1.0s de intervalo

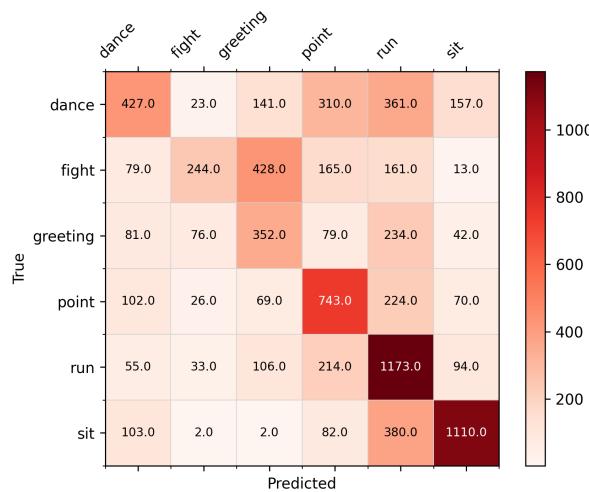


Figura E.14: Matriz de confusión del modelo RNN con 1.0s de intervalo

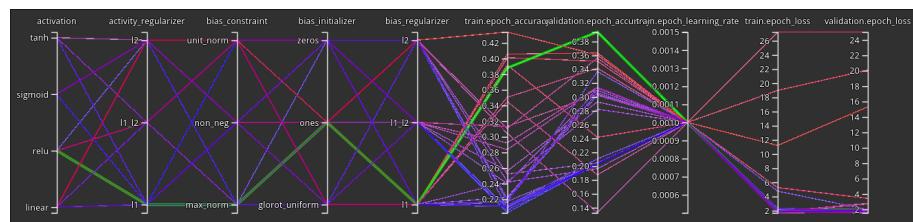


Figura E.15: Gráfico de entrenamiento del modelo RNN con 1.0s de intervalo (mejor val_accuracy = 0.39377)

Apéndice F

Resultados de los distintos modelos CNN

La línea verde en los gráficos de tensorboard indican la mejor combinación de hiperparámetros encontrada en cada caso

F.1. Intervalo 0.2s

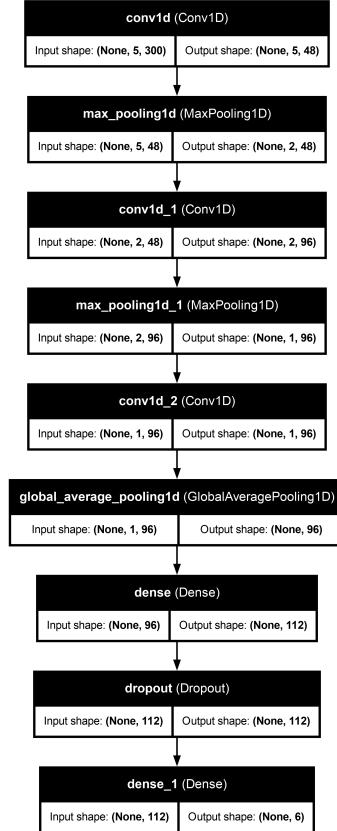


Figura F.1: Esquema del modelo CNN con 0.2s de intervalo

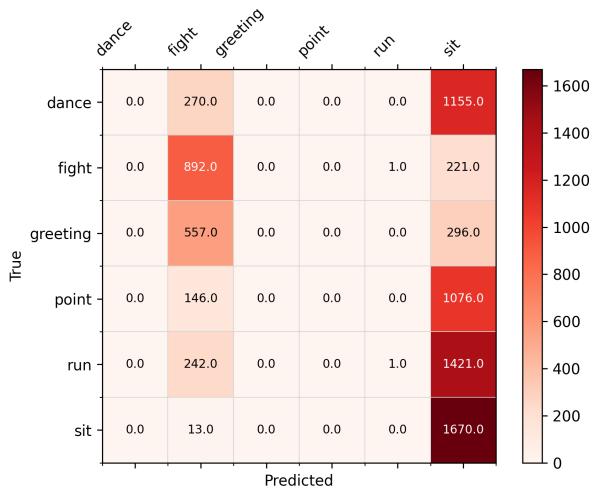


Figura F.2: Matriz de confusión del modelo CNN con 0.2s de intervalo

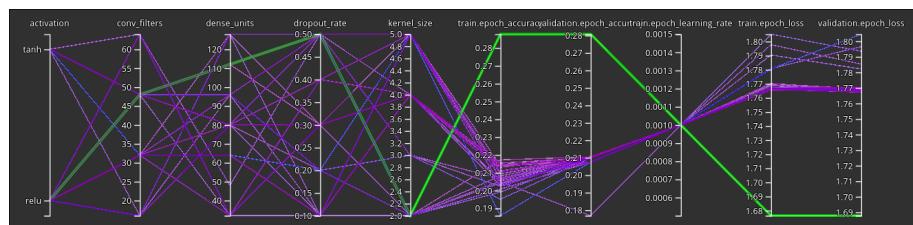


Figura F.3: Gráfico de entrenamiento del modelo CNN con 0.2s de intervalo (mejor val_accuracy = 0.28777)

F.2. Intervalo 0.4s

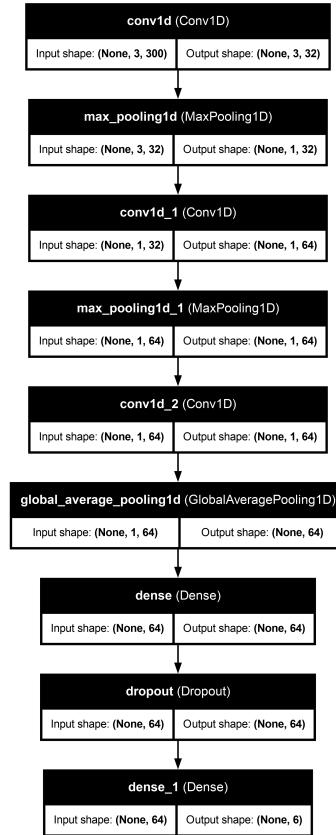


Figura F.4: Esquema del modelo CNN con 0.4s de intervalo

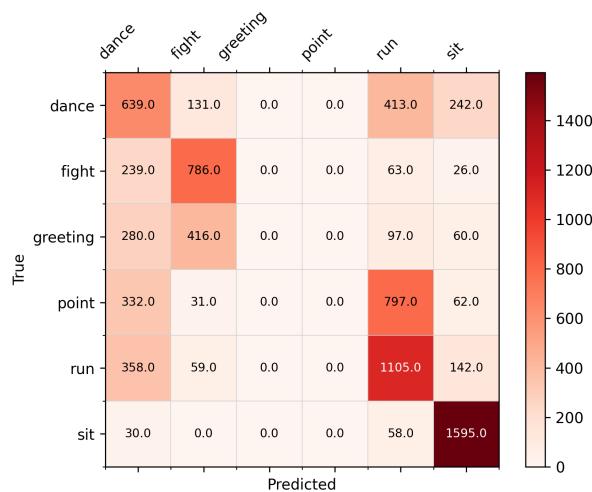


Figura F.5: Matriz de confusión del modelo CNN con 0.4s de intervalo

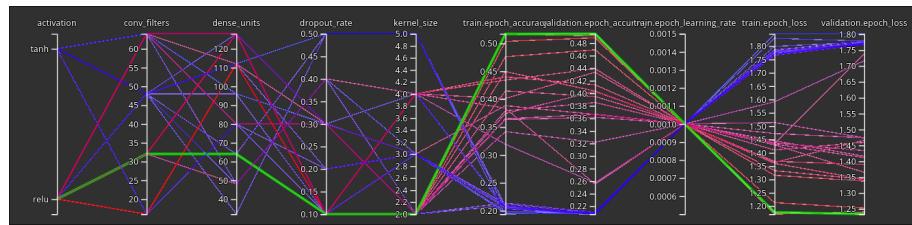


Figura F.6: Gráfico de entrenamiento del modelo CNN con 0.4s de intervalo (mejor val_accuracy = 0.49473)

F.3. Intervalo 0.6s

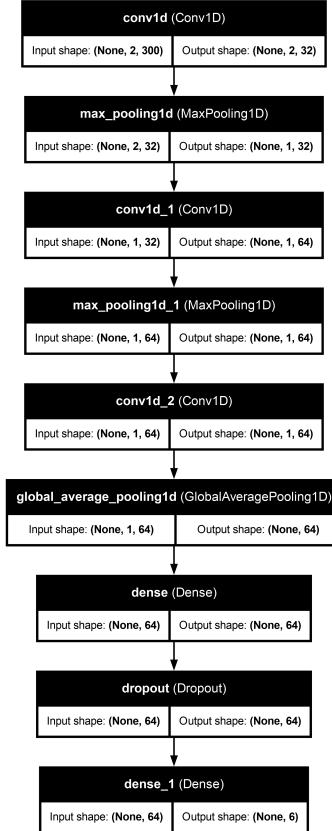


Figura F.7: Esquema del modelo CNN con 0.6s de intervalo

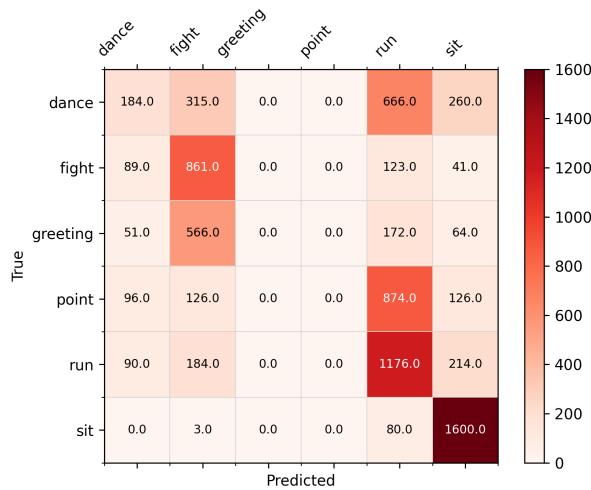


Figura F.8: Matriz de confusión del modelo CNN con 0.6s de intervalo

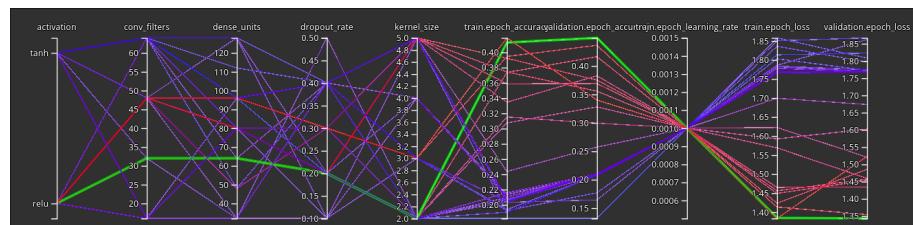


Figura F.9: Gráfico de entrenamiento del modelo CNN con 0.6s de intervalo (mejor val_accuracy = 0.44952)

F.4. Intervalo 0.8s

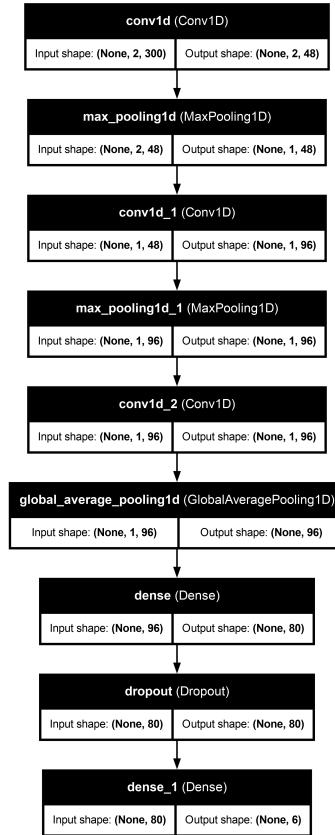


Figura F.10: Esquema del modelo CNN con 0.8s de intervalo

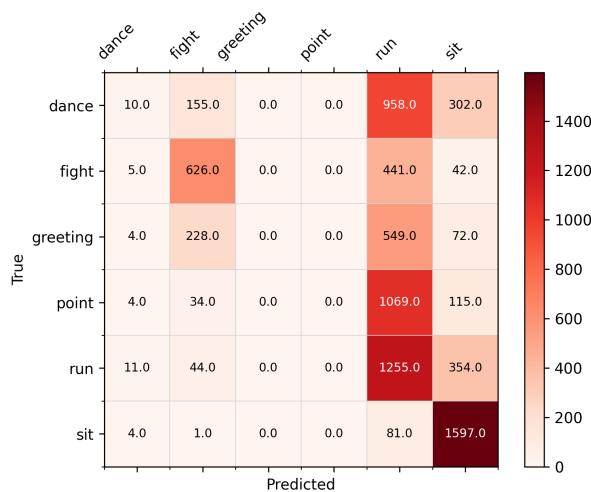


Figura F.11: Matriz de confusión del modelo CNN con 0.8s de intervalo

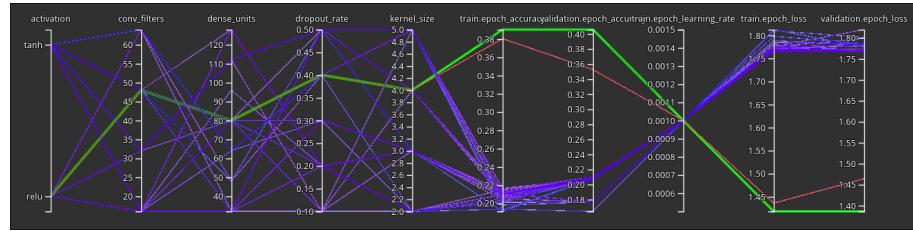


Figura F.12: Gráfico de entrenamiento del modelo CNN con 0.8s de intervalo (mejor val_accuracy = 0.40583)

F.5. Intervalo 1.0s

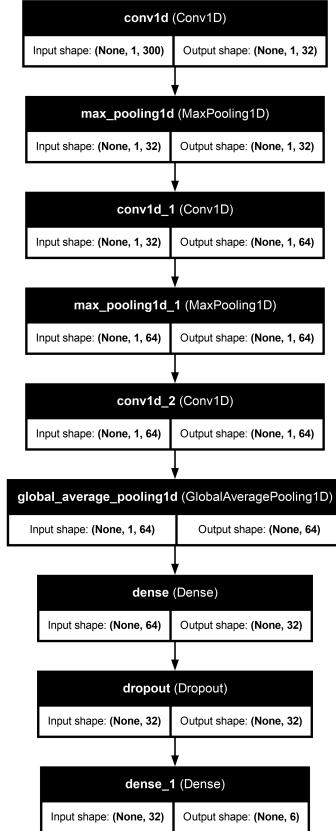


Figura F.13: Esquema del modelo CNN con 1.0s de intervalo

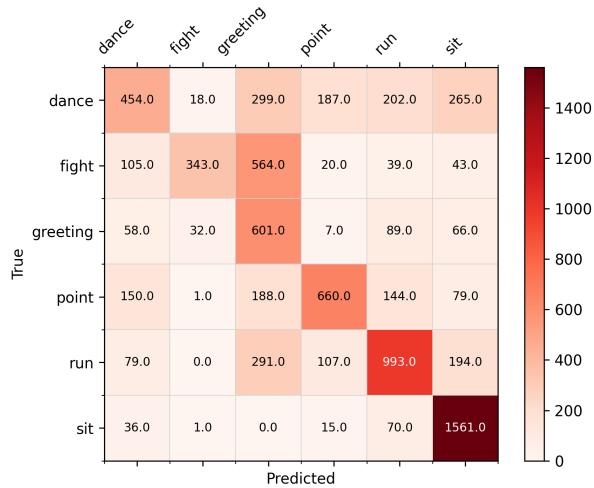


Figura F.14: Matriz de confusión del modelo CNN con 1.0s de intervalo

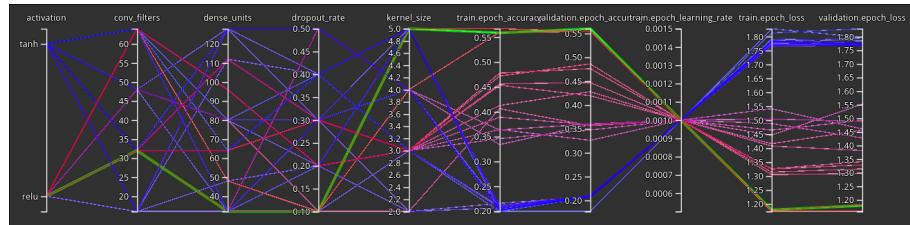


Figura F.15: Gráfico de entrenamiento del modelo CNN con 1.0s de intervalo (mejor val_accuracy = 0.56002)

Resultados de los distintos modelos Random Forest

G.1. Intervalo 0.2s

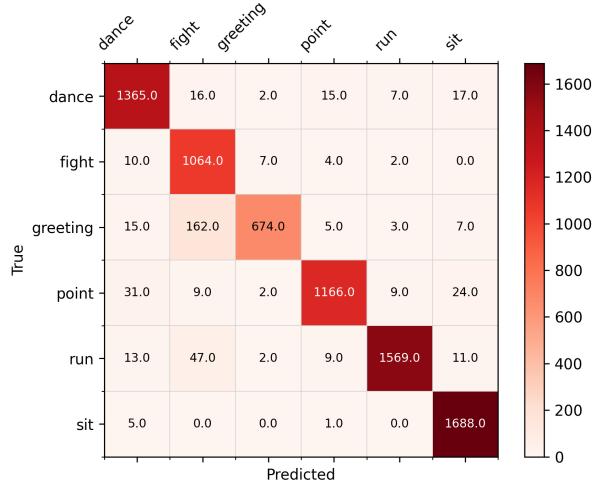


Figura G.1: Matriz de confusión del modelo Random Forest con 0.2s de intervalo (mejor val_accuracy = 0.85735)

G.2. Intervalo 0.4s

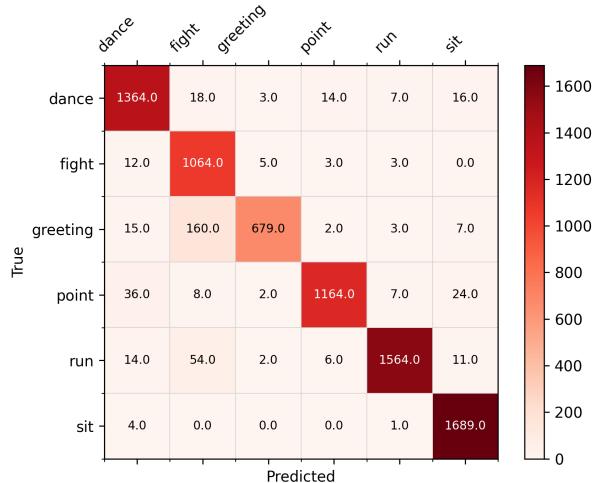


Figura G.2: Matriz de confusión del modelo Random Forest con 0.4s de intervalo (mejor val_accuracy = 0.85735)

G.3. Intervalo 0.6s

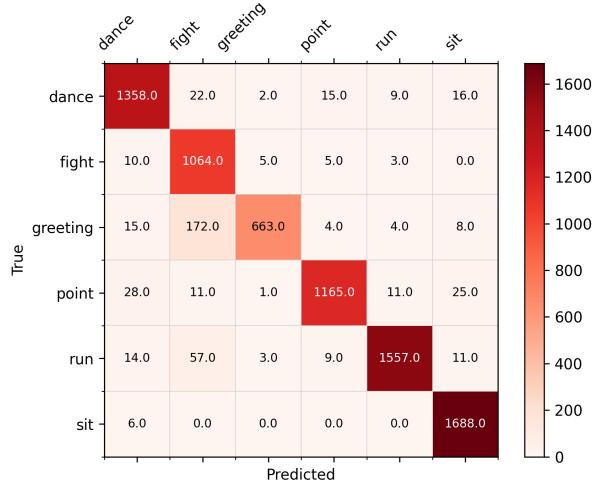


Figura G.3: Matriz de confusión del modelo Random Forest con 0.6s de intervalo (mejor val_accuracy = 0.85233)

G.4. Intervalo 0.8s

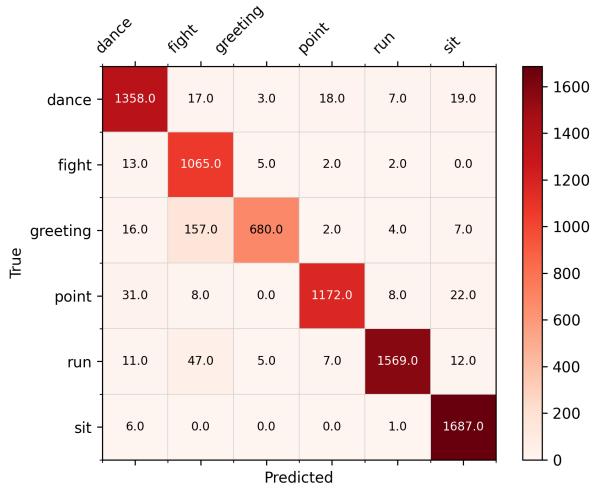


Figura G.4: Matriz de confusión del modelo Random Forest con 0.8s de intervalo (mejor val_accuracy = 0.85936)

G.5. Intervalo 1.0s

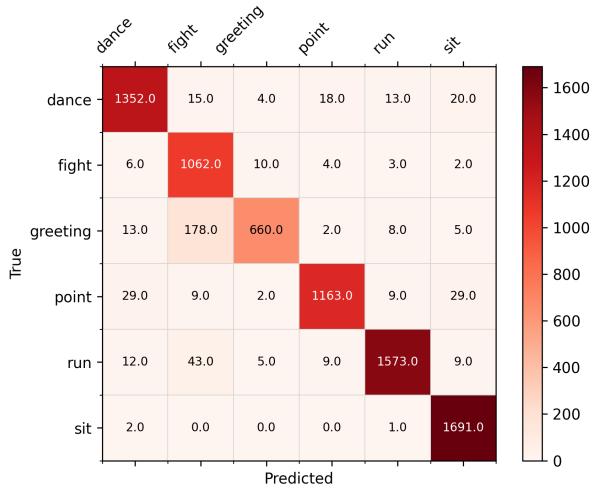


Figura G.5: Matriz de confusión del modelo Random Forest con 1.0s de intervalo (mejor val_accuracy = 0.84681)

Glosario

AI	Artificial Intelligence
Animator	Componente de Unity que funciona como una máquina de estados para la ejecución de animaciones
API	Application Programming Interface
API REST	Arquitectura de diseño utilizada para crear servicios web y comunicar diferentes sistemas
CART	Un árbol de decisión CART (Classification and Regression Trees) es un algoritmo de aprendizaje automático que utiliza árboles binarios para clasificar o predecir valores continuos
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
DCNN	Deep Convolutional Neural Network
FBX	El formato FBX es un tipo de archivo que contiene datos de objetos en 3D así como de animaciones
fork	Copia de un proyecto, generalmente software, que se hace con la intención de ser modificado y desarrollado, con posibilidad de independencia del proyecto original o con afán de mejorar el original
FPS	Frames Per Second
GPU	Graphics Processing Unit

gradient boosted decision tree	Un gradient boosted decision tree es un algoritmo de aprendizaje automático que utiliza múltiples árboles de decisión y optimización por gradiente para mejorar la precisión de las predicciones
IA	Inteligencia Artificial
iframe	Un iframe es un elemento HTML que permite incrustar otro documento HTML dentro de una página web
isolation forest	Un isolation forest es un algoritmo de aprendizaje automático que utiliza árboles de decisión para detectar anomalías en los datos
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
metaverse	The metaverse is a set of digital spaces to socialize, learn, play and do other activities. (Definition by the company Meta)
metaverso	El metaverso es un conjunto de espacios digitales para socializar, aprender, jugar y realizar otras actividades. (Definición de la empresa Meta)
MMORPG	Massively Multiplayer Online Role-Playing Game
NPC	Non Playable Character
Random Forest	Un random forest es un algoritmo de aprendizaje automático que utiliza múltiples árboles de decisión para mejorar la precisión y la robustez de las predicciones
rig	Un rig es el resultado del rigging, una técnica usada en animación digital que permite crear estructuras alrededor de una malla para poder animar de forma más sencilla
RNN	Recurrent Neural Network
SDK	Software Development Kit
SVM	Support Vector Machine
TPU	Tensor Processing Unit
VR	Realidad Virtual

YOLO

You Only Look Once