UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Software de ayuda para el análisis de las características angulares en diferentes articulaciones en la especie canina durante la locomoción.

TRABAJO DE FÍN DE TÍTULO

GRADO EN INGENIERÍA INFORMÁTICA – TECNOLOGÍAS DE LA INFORMACIÓN

NOVIEMBRE 2020 – LAS PALMAS DE GRAN CANARIA

Autor

Adrián García Sánchez

Tutores

Dr. Alexis Quesada Arencibia – Ciencias de la Computación e Inteligencia Artificial

Dr. José Manuel Vilar Guereño – Medicina y Cirugía Animal

AGRADECIMIENTOS

En primera instancia agradecer tanto a Alexis como a José Manuel por haberme dado la oportunidad de realizar este proyecto, así como su ayuda durante su elaboración.

Por otra parte, a Pedro Figueirinhas, por haber realizado el *dataset* necesario para la elaboración del proyecto.

Y, por último lugar, a mis compañeros y amigos Javier Sintes, Pablo Hernández y Aarón Alemán, por su apoyo y ayuda durante la totalidad de la carrera, así como durante la realización de este proyecto.

RESUMEN

El trabajo consiste en el análisis y tratamiento de datos provenientes de un conjunto de sensores inerciales situados en perros con el fin de detectar las características angulares en diferentes articulaciones durante la locomoción. Dichas características (rango angular de movimiento, velocidad/aceleración angular, etc.) permitirían discernir estados de la articulación tanto fisiológicos como patológicos (cojeras que tienen como origen una osteoartritis, por ejemplo). El tratamiento y gestión de los datos se realiza a través de una aplicación web, y la identificación del estado de las articulaciones a través de un módulo basado en el uso de redes neuronales.

ABSTRACT

This project consist in the analysis and treatment of data from inertial sensors place in dogs, with the aim of detect angular characteristics in differents joints during the walk. These characteristics (angle range of movement, speed/angular acceleration, etc..) would allow differentiate joint's states, like physiology or pathology (lameness from a osteoarthritis, for example). The treatment and manage of data is done with a web app and the identification of joint's states with a neuronal network module.

CONTENIDO

Agradecimientos 2
Resumen 3
Abstract 3
Tabla de ilustracions 7
Introducción 9
Estructura del documento 11
1. Estado actual y objetivos iniciales. 14
1.1 Estado Actual 14
1.1.1 Redes neuronales 16
1.2 Objetivos 22
2. Competencias cubiertas 23
2.1 IS01 23
2.2 IS02 23
2.3 CP04 23
2.4 TI06 24
3. Aportaciones 25
3.1 Entorno personal 25
4. Normativa y Legislación 26
4.1 Licencias Software 26
4.1.1 Licencia GNU 26
4.1.2 Licencia PSF 27
4.1.3 Licencia MIT 27
4.1.4 Licencia BSD 27
4.1.5 Licencia JetBrains 27
5. Metodología de trabajo y planificación 28
5.1 Planificación inicial 28
5.2 Metodología de trabajo. 30
6. Tecnologías, herramientas e infraestructuras hardware. 33
6.1 Técnologías y herramientas empleadas en la aplicación web 33
6.1.1 Servidor 34
6.1.2 Cliente 36

6.2 Tecnologías y herramientas de la red neuronal 37
6.3 Infraestructuras hardware 39
7. Análisis. 40
7.1 Elementos de la aplicación 40
7.2 Roles de usuario en la herramienta 41
7.2.1 Administrador 41
7.2.2 Veterinario 41
7.3 Análisis del problema 42
7.3.1 Características tomas 42
7.3.1 Problema cojera 43
7.3.2 Características angulares 49
8. Diseño. 51
8.1 Arquitectura del sistema 51
8.2 Diseño de la base de datos 52
8.3 Almacenamiento de archivos en el servidor 54
8.4 Diseño interfaz de usuario 55
8.5 Diseño red neuronal 58
8.5.1 RNN y LSTM 59
9. Desarrollo 63
9.1 Desarrollo del cliente 63
9.1.1 Estructura del contenido 63
9.1.2 Servicios 64
9.1.3 Control de acceso 65
9.2 Desarrollo del servidor 66
9.2.1 Endpoints 67
9.2.2 Control de Acceso 71
9.3 Desarrollo de la red neuronal 72
9.3.1 Lectura de los datos 72
9.3.2 Modelo 74
10. Pruebas 83
10.1 Aplicación web 83
10.2 Red neuronal 84

10.2.1 Mejoras 86
11. Conclusiones y Trabajo futuro 90
11.1 Conclusiones 90
11.2 Trabajo futuro 91
Referencias 92
Manual de usuario 95
Inicio de sesión 95
Gestión de usuarios 96
Vista de perros 98
Gestión de sesiones 101
Vista de tomas 103

TABLA DE ILUSTRACIONS

Ilustración 1: Comparativa capacidad de procesamiento en GB/S entre tarjetas gráficas y procesadores. 16

Ilustración 2: Comparativa resultados en Imagenet. 17

Ilustración 3: Comparación estructura interna de una red neuronal artificial, y una humana. 18

Ilustración 4: Ejemplo clasificación lineal. 19

Ilustración 5: Red neuronal multicapa. 20

Ilustración 6: Imagen del sensor utilizado. 39

Ilustración 7: Localización de sensores en un perro. 43

Ilustración 8: Grado de amplitud al caminar de un perro sano. 44

Ilustración 9: Grado de amplitud al caminar de un perro con cojera. 45

Ilustración 10: Comparativa de datos entre perros sanos. 46

Ilustración 11: Comparativa de datos entre perros sanos eliminando "Outliers". 47

Ilustración 12: Visualización datos perro sano. 48

Ilustración 13: Grado de amplitud al caminar de un perro con cojera. 48

Ilustración 14: Localización sensores, para obtención ángulo relativo. 49

Ilustración 15: Mostrar ángulo relativo, ... 50

Ilustración 16: Comunicación entre los componentes de la aplicación. 52

Ilustración 17: Estructura archivos en la aplicación. 54

Ilustración 18: Mockup vista general perros. 55

Ilustración 19: Vista general de perros en la aplicación. 56

Ilustración 20: Vista de videos en la aplicación final. 57

Ilustración 21: Visualización de gráfica de cojera en la aplicación final. 57

Ilustración 22: Comparativa RNN (imagen superior) y LSTM (imagen inferior). 59

Ilustración 23: Implementación de la red neuronal. 62

Ilustración 24: Estructura del proyecto en Angular. 63

Ilustración 25: Categoría de los endpoints. 67

Ilustración 26: Obtención de las tomas en Python. 73

Ilustración 27: Métodos auxiliares para la lectura de los datos. 73

Ilustración 28: Feedfoward red neuronal. 74

Ilustración 29: Diferencias learning rate. 76
Ilustración 30: Diferencias learning rate. 78
Ilustración 31: Fórmula error cuadrático medio. 79
Ilustración 32: Entrenamiento de la red. 82
Ilustración 33: Prueba de la red. 85
Ilustración 34: Representación Early stopping. 86
Ilustración 35: Añadida regularización al entrenamiento. 88
Ilustración 36: Precisión del modelo tras añadir regularización. 89
Ilustración 37: Página de acceso a la aplicación. 95
Ilustración 38: Página principal "Vista de perros". 96
Ilustración 39: Gestión de usuarios. 96
Ilustración 40: Creación de usuarios. 97
Ilustración 41: Actualización de usuarios. 97
Ilustración 42: Confirmación de borrado de usuarios. 98
Ilustración 43: Creación de perro. 99
Ilustración 44: Vista detallada de los perros. 99
Ilustración 45: Creación de sesión. 100
Ilustración 46: Vista de sesiones. 101
Ilustración 47: Gestión de una toma. 102
Ilustración 48: Vista de una toma. 103
Ilustración 49: Vista gráfica de los datos de los sensores. 104
Ilustración 50: Observación de la toma. 104

Introducción

Cada año, una gran cantidad de perros sufren de algún tipo de dolencia en alguno de sus miembros, lo que puede llevarlos a cojear durante la locomoción. Este tipo de dolencias provoca en los animales que la padecen, que pasen una gran cantidad de tiempo inmóvil al día, pues su movilidad va a estar limitada a causa del dolor haciendo que su salud y calidad de vida empeoren drásticamente.

Dentro del sector veterinario [1] se han desarrollado múltiples métodos para analizar el estado locomotor de los perros, con la finalidad de medir parámetros cinemáticos con cierta precisión y fiabilidad que permitan detectar cojera. Sin embargo, todo el análisis posterior requiere de un experto en la materia para determinar un diagnóstico.

Por otro lado, en los últimos años se ha producido un importante auge de la Inteligencia Artificial (IA), especialmente en sectores relacionados con el ámbito de la medicina. La aplicación de la IA en el ámbito médico ha permitido el desarrollo de sistemas de apoyo al diagnóstico de diferentes enfermedades, existiendo numerosos ejemplos donde estos sistemas han demostrado ser capaces de ofrecer una fiabilidad incluso superior a los humanos y de forma casi instantánea. Un ejemplo de este tipo de sistemas es el propuesto por Bobby Buka, en [2], donde se propone un sistema para la detección precoz de cáncer de piel donde a través del análisis de imágenes es posible detectar melanomas malignos en la piel con una alta tasa de acierto, pudiendo salvar una inmensa cantidad de vidas humanas al año, así como mejorar su calidad de vida.

Romiti et al [3] tratan de la detección de enfermedades cardiovasculares. En 2015, 85 millones de europeos sufrieron algún tipo de enfermedad cardiovascular, ocasionando una gran cantidad de fallecidos, y un elevado coste económico. Sin embargo, si se realiza una detección precoz de la enfermedad, se le podrá ofrecer un tratamiento al paciente para prevenir su evolución, mejorando así su calidad de vida. Por ello, se han aplicado técnicas

basadas en inteligencia artificial, como la detección de dicha enfermedad, a través de ecocardiografías, donde son capaces de ayudar al médico encargado de su evaluación, puesto que es estas ecocardiografías están sujetas a varios factores, como de la variabilidad del interoperador o de la experiencia del propio médico encargado, mientras que la inteligencia artificial puede detectar un amplio abanico de patrones relacionados con la enfermedad.

Siguiendo esta línea de trabajo, en este TFT se propone la aplicación de una red neuronal para la detección de cojera en perros, tomando como fuente de datos un conjunto de sensores inerciales colocados en el animal, con el fin de medir ciertos parámetros claves que permitan a esta red realizar una propuesta de diagnóstico, es decir, indicar si el perro padece o no de cojera, con una cierta precisión.

Se trata de una propuesta novedosa e innovadora, pues hasta el momento de la escritura de esta memoria no se conocen trabajos en esta línea, siendo por tanto un primer paso en este campo que podría permitir la elaboración de aplicaciones de apoyo a los especialistas que permitan diagnósticos más complejos y donde resulte fundamental la información que este tipo de sistemas inteligentes puedan aportar.

ESTRUCTURA DEL DOCUMENTO

El documento se encuentra estructurado en múltiples apartados, donde cada uno de estos trata un aspecto concreto del trabajo realizado. A continuación, se explica el contenido de cada uno de ellos:

- Apartado 1: Estado actual y objetivos iniciales.
    - Se explica la situación actual sobre la cojera en perros en múltiples ámbitos, también se realiza una breve introducción a las redes neuronales, y se finaliza planteando los objetivos iniciales para el proyecto.
- Apartado 2: Competencias cubiertas.
    - Se justifican y enumeran las distintas competencias atribuidas al grado de ingeniería informática que se cumplen con la realización de este proyecto.
- Apartado 3: Aportaciones
    - En este apartado se describen las aportaciones del proyecto a nivel personal.
- Apartado 4: Normativa y legislación.
    - Se enumeran y describen las licencias software utilizadas para realizar el proyecto.
- Apartado 5: Metodología de trabajo y planificación.
    - Se explica la metodología de trabajo aplicada para el proyecto, así como la planificación planteada en primera instancia, y las deviaciones del proyecto, debido a factores externos.

- Apartado 6: Tecnologías, herramientas e infraestructuras hardware.
    - Se indican las distintas tecnologías y herramientas utilizadas para el desarrollo del proyecto, y se justifica su elección, así como la infraestructura hardware utilizada.
- Apartado 7: Análisis.
    - Se procede a explicar las fases de análisis del proyecto, así como los distintos actores involucrados en la aplicación.
- Apartado 8: Diseño.
    - Se explica el diseño de los distintos componentes de la aplicación web, y su justificación, así como el tipo de red utilizada para el proyecto y el motivo de su utilización.
- Apartado 9: Desarrollo.
    - Se justifica el trabajo realizado, explicando cada uno de los distintos elementos que conforman el proyecto, y su integración en la aplicación.
- Apartado 10: Pruebas.
    - En este apartado se comentan las distintas pruebas realizadas a lo largo del proyecto para comprobar el correcto funcionamiento del sistema. Además, también se indican las pruebas y mejoras realizadas sobre la red neuronal.

- Apartado 11: Conclusiones y trabajo futuro.
    - En este apartado se comentan las conclusiones obtenidas tras la finalización del proyecto, y se comentan posibles mejoras y ampliaciones.

- Apartado 12: Bibliografía.
    - Aquí se enumeran todas las referencias utilizadas para llevar a cabo este proyecto.

1. ESTADO ACTUAL Y OBJETIVOS INICIALES.

1.1 ESTADO ACTUAL

Actualmente no existe ningún software comercial para la detección de cojera o la obtención de características angulares en perros. Sin embargo, existen herramientas para otros tipos de animales como los caballos, por ejemplo *equinosis"* [4]. En esta aplicación, a través del uso de diversos sensores inerciales situados en la cabeza, grupa y una extremidad torácica, el sistema es capaz de captar ciertas anomalías en el andar del caballo, con el objetivo de conocer si cuenta con algún tipo de cojera que pueda ser diagnosticado en sus primeros estados y de esta forma poder prevenir posibles dolencias futuras, debido a la fuerte carga y desgaste que sufren por la gran cantidad de trabajo que realizan.

Por otra parte, se ha observado un trabajo de investigación donde se propone una metodología parecida a la empleada por el software *equinosis,* pero en este caso aplicada a perros [5]. En este estudio se hace uso de cintas de correr con la idea de establecer una velocidad determinada, y así poder estandarizar todas las tomas realizadas.

En el estudio indicado [5], todos los perros se encuentran en un rango de peso determinado y se limita a unas ciertas razas, puesto que como veremos más adelante no todos los perros son aptos para este tipo de análisis, ya que deben reunir unas ciertas características. Para el análisis de las tomas realizadas, los investigadores han realizado un estudio estadístico, haciendo uso de gráficas que permitan ver las diferencias entre las distintas tomas, con el objetivo de buscar y detectar posibles patrones en ellos.

Finalmente, se concluyó que la utilización de sensores inerciales es una buena manera de detectar y ver las diferencias entre distintos tipos de cojera, pero a diferencia de este proyecto, no se hace uso de redes neuronales.

Por lo tanto, actualmente no existe ninguna alternativa en el mercado específico para los perros donde pueda realizarse un análisis de posibles cojeras o bien, observar los distintos ángulos que va formando alguna extremidad a la hora de caminar con el fin de conocer su rango de actuación de forma precisa.

1.1.1 REDES NEURONALES

Hoy en día, las redes neuronales tienen una gran relevancia en el mundo actual y existen numerosos ejemplos de su de su aplicación en diferentes ámbitos [referencias]. Por tanto, es muy previsible que en los próximos años tengan aún más relevancia.

Su inicio se remonta a los años 60 [3], con la elaboración de las primeras redes neuronales artificiales para problemas de clasificación binaria (distinguir entre dos categorías), en los años posteriores se desarrollaron múltiples algoritmos para mejorar su desempeño. Sin embargo, se requieren de millones de operaciones matemáticas para llevar a cabo las predicciones, y el *hardware* con el que se contaba antiguamente no ofrecía la capacidad suficiente para esta labor.

Con la aparición de las tarjetas gráficas, se ha conseguido ofrecer la capacidad de procesamiento necesaria para esta labor. En la ilustración 1 se observa una comparación en la capacidad de procesamiento en procesadores y tarjetas gráficas, y se puede observar la gran mejoría que ofrecen sobre los procesadores.

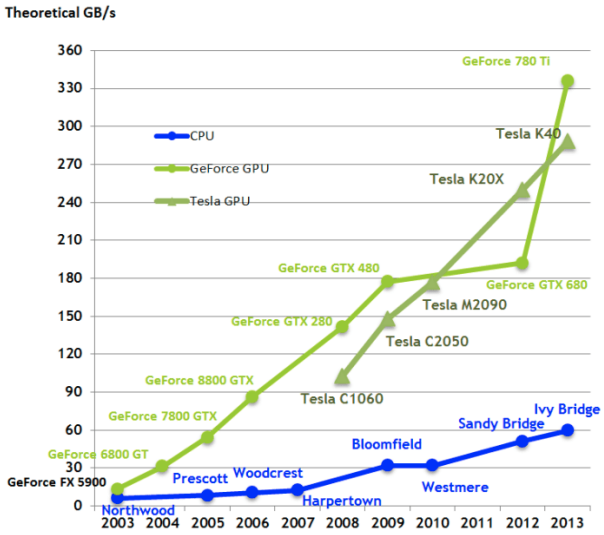

Ilustración 1: Comparativa capacidad de procesamiento en GB/S entre tarjetas gráficas y procesadores.

Para observar la mejoría que ofrecen las redes neuronales sobre métodos tradicionales, se pone el ejemplo de "*Imagenet Challenge*" [6], una competición que ofrece a los participantes una gran variedad de imágenes de distintas categorías (aviones, perros o coches), donde el objetivo es clasificar la mayor cantidad de imágenes en la categoría correcta. En la ilustración 2, se observan los porcentajes de error de los ganadores.

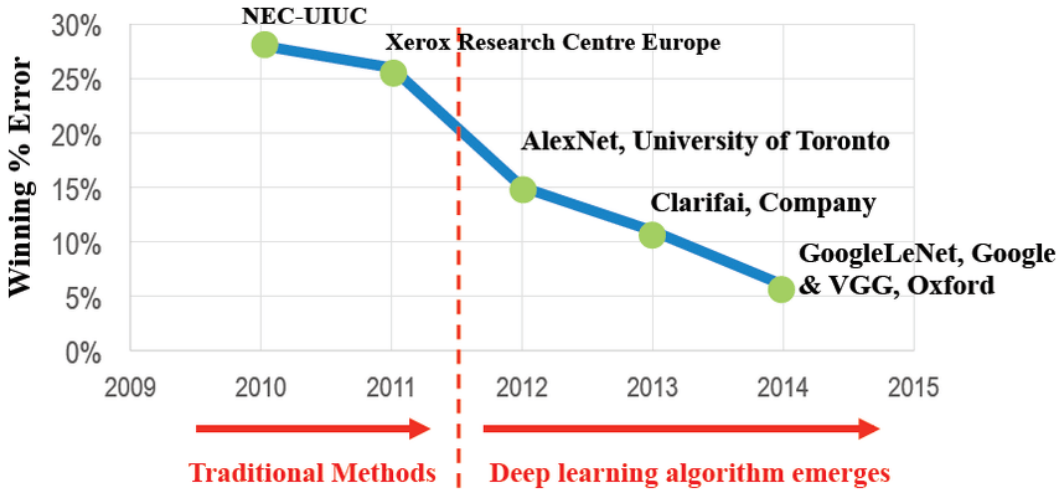

Ilustración 2: Comparativa resultados en Imagenet.

Si se observan los años 2011 y 2012, no deja lugar a dudas que las redes neuronales ofrecen un desempeño ampliamente superior a los métodos tradiciones. Esto se debe precisamente a la irrupción de las tarjetas gráficas dentro de la competición, algo que ocurrió en el año 2012, y que viene a justificar su uso cuando existan restricciones importantes de tiempo.

Desde un punto de vista teórico, las redes neuronales artificiales [7] tratan de emular el comportamiento de las neuronas humanas, pudiéndose ver las similitudes observando la "estructura" interna de ambas. En la ilustración 3 se puede apreciar una neurona humana y una neurona artificial.

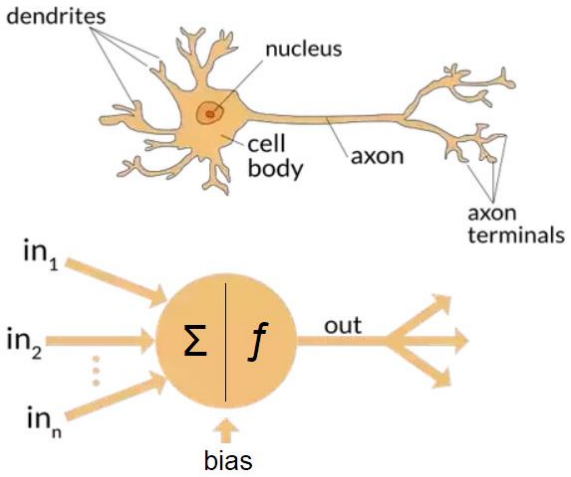

Ilustración 3: Comparación estructura interna de una red neuronal artificial, y una humana.

Hay que resaltar las dendritas en la neurona humana, estas se encargan de recibir las señales de otras neuronas para su posterior procesado en el soma o cuerpo celular, para su posterior propagación mediante señales a través de los axones al resto de neuronas.

Si se observa la neurona artificial, cuenta con una estructura similar, la entrada sería la encargada de recibir un conjunto de datos, un ejemplo puede ser un vector de valores que represente una imagen, con cada uno de sus píxeles representados en un rango de 0 a 255.

En el "cuerpo" de la red neuronal artificial se aplican funciones matemáticas para producir una salida. Estas funciones se apoyan en el uso de regresión lineal [8], un modelo matemático que trata de realizar clasificaciones lineales como la representada en la ilustración 4, siendo el objetivo "agrupar" las distintas categorías según sus características.

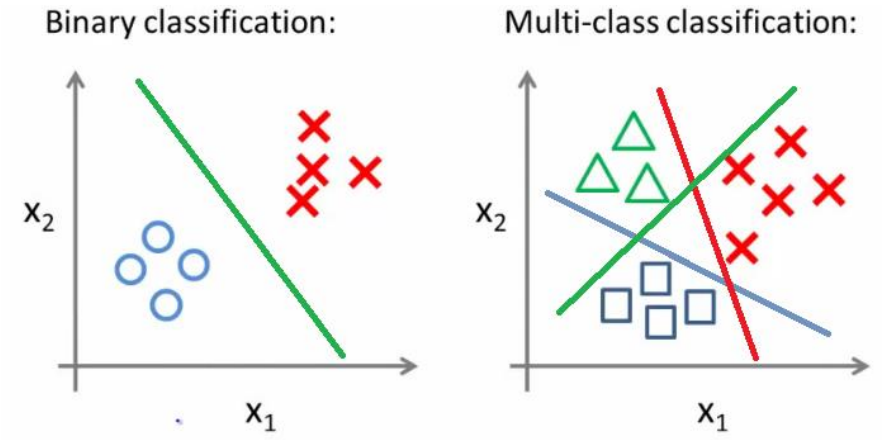

Ilustración 4: Ejemplo clasificación lineal.

Conviene señalar que pueden existir tantas categorías como se necesiten, al igual que ocurre con las entradas. Si se observa una imagen RGB (tres canales de colores) de 256x256 pixeles, se cuenta con 196.608 entradas. Para el procesado se hace uso de la siguiente formula:

$$y = wa + b$$

a es el conjunto de datos de entrada, w es el "peso" o importancia que se le da a cada entrada, por ejemplo, si se estuviese tratando de predecir el precio de una vivienda en base al número de habitaciones del lugar, el número de baños y la distancia al colegio más cercano, estos pesos serían distintos dependiendo de su importancia de cara al precio final de la vivienda (los pesos los ajusta la red neuronal).

Por otra parte, se observa b, se trata del "bias", una constante añadida al producto de la entrada y peso, que tiene la finalidad de ajustar la salida de la neurona, ya que a la salida de la neurona, se le aplica una función de activación que tiene el propósito de situar la salida de la neurona en un rango determinado. En posteriores apartados se observarán distintas funciones de activación y el objetivo de cada una de ellas.

Una vez realizado el procesado de los datos de entrada, se genera la salida de la neurona, la cual puede resultar la salida final de un problema de clasificación, prediciendo si una imagen proporcionada se trata de un gato o no, por ejemplo.

Sin embargo, no se hace uso de una única neurona en entornos reales, puesto que resulta insuficiente para poder abordar la mayoría de problemas reales. Si se observa la ilustración 4, en problemas de clasificación binaria se puede utilizar una neurona, aunque cuando existen múltiples categorías, como la gráfica que representa una clasificación multiclase, esta necesita de varias neuronas para clasificar correctamente las distintas categorías.

Cuando se agrupan múltiples neuronas, nos encontramos con una red formada por varias capas como la que se observa en la ilustración 5.

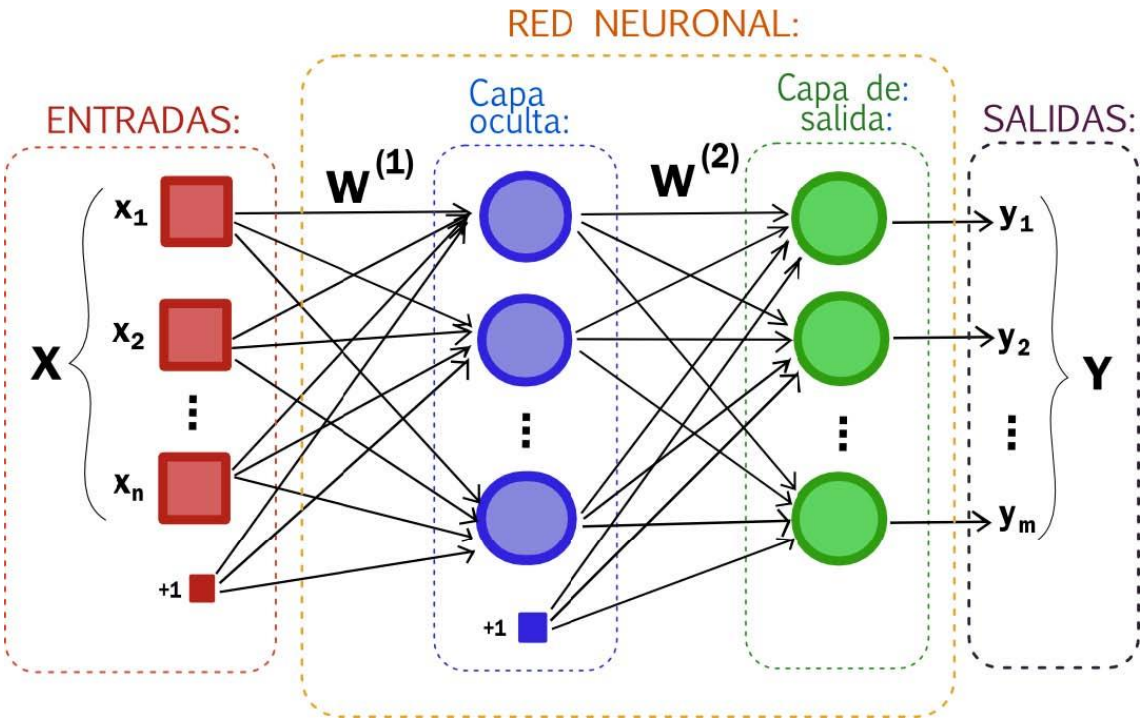

Ilustración 5: Red neuronal multicapa.

Se observan 3 capas, en primer lugar, la capa de entrada, la encargada de obtener los datos de entrada; la siguiente capa, se denomina capa oculta, pudiendo existir tantas capas ocultas como se requieran, son las encargadas de realizar el procesado de los datos obtenidos de la capa de entrada. Una vez procesados, se obtiene la salida de la red a través de la capa de salida. Existen tantas entradas como parámetros a analizar, y tantas salidas como categorías a clasificar.

Para la correcta predicción de las diversas categorías con una cierta precisión, se procede a "entrenar" la red. Para ello se debe contar con un amplio conjunto de entrenamiento, la red irá "aprendiendo" de estos datos de entrada, labor para la cual se aplica una función de perdida, la cual compara la predicción de la red y la esperada. Esta función se utiliza en conjunto con múltiples algoritmos que podremos ver a lo largo del documento.

1.2 OBJETIVOS

En primera instancia, al completar este proyecto se espera desarrollar una red neuronal, capaz de diagnosticar si un perro puede tener cojera o no. Sin embargo, debido a la poca cantidad de datos recolectados, no se espera contar una alta tasa de acierto, ya que se necesita de una extensa cantidad de datos para poder entrenar a la red.

A partir de los datos obtenidos, se realizará un estudio estadístico de los datos con el objetivo de detectar patrones en estos, así como conocer su estructura, para determinar qué tipo de red neuronal se ajusta mejor a este problema.

Por otra parte, con el fin de ayudar al experto, se desarrollará una vista capaz de mostrar las distintas características angulares del perro, para conocer el rango de movimiento de sus extremidades, al mismo tiempo que se visualiza el video.

Por último, para agrupar todas las funcionalidades, se realizará el desarrollo de un portal web, con la capacidad de gestionar los datos de perros, permitiendo llevar un historial de las tomas realizadas y su evolución en el tiempo.

2. COMPETENCIAS CUBIERTAS

2.1 IS01

"Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software."

Durante este proyecto se ha desarrollado un portal web, analizando los requisitos de usuario requeridos, y aplicado los principios de ingeniería del software para asegurar el correcto funcionamiento de la aplicación.

2.2 IS02

"Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones."

Se han realizado varias reuniones, para especificar el alcance del trabajo, y acordar los requisitos esperados de la aplicación final, con el objetivo de conocer las necesidades del usuario final de la aplicación.

2.3 CP04

"Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación."

Se ha desarrollado un servicio que hace uso de redes neuronales, con la capacidad de ofrecer un diagnóstico a partir de los datos proporcionados, para que el usuario final pueda visualizarlos.

2.4 TI06

"Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil."

El proyecto ha sido desarrollado empleando diversas tecnologías de red, haciendo uso de distintos protocolos para la comunicación entre cliente y servidor, para poder transferir la información y recursos necesarios como videos o ficheros de datos.

3. APORTACIONES

3.1 ENTORNO PERSONAL

A nivel personal este proyecto me ha brindado la oportunidad de conocer el sector veterinario, un sector desconocido hasta entonces, permitiéndome trabajar en el desarrollo de un problema de principio a fin, realizando tareas como el análisis del problema o la elaboración del *dataset* requerido para el entrenamiento de la red neuronal, conociendo así los procedimientos a seguir para su correcta elaboración.

Por otra parte, me ha permitido ponerme en la situación de un proyecto de investigación, desenvolviendo un rol de analista de datos, un puesto no experimentado hasta entonces.

Es preciso señalar que he podido seguir aprendiendo inteligencia artificial, un campo apasionante y de suma importancia en nuestros días, y de hecho, contará con mayor relevancia e importancia en los próximos años.

Del mismo modo, se han hecho uso de estructuras de redes neuronales utilizadas actualmente en multitud de proyectos reales, como el reconocimiento de voz, o la traducción de texto. Consecuentemente, se han utilizado librerías y estructuras de datos desconocidas hasta el momento.

Para concluir, con la elaboración de la aplicación web, se han utilizado tecnologías ampliamente utilizadas en el mercado laboral, con el objetivo de prepararme así para ello. Para concluir, al tener que establecer reuniones con el usuario final, me ha permitido desarrollar mis conocimientos y habilidades como analista, trabajar aspectos relacionados con el trato con clientes, así como entender sus requisitos de cara a la aplicación final.

4. NORMATIVA Y LEGISLACIÓN

Otro de los aspectos claves a observar durante el desarrollo del proyecto, ha sido prestar atención a las licencias del software utilizado, puesto que pueden existir algunas limitaciones que impidan desarrollar ciertos aspectos debido a temas legales.

En los siguientes apartados se observan las distintas licencias involucradas en el desarrollo del proyecto, debido a la utilización de ciertas tecnologías en este.

4.1 LICENCIAS SOFTWARE

En primera instancia hay que entender las licencias software, *"Estas básicamente son un contrato entre el autor del programa y el usuario, y comprenden una serie de términos y cláusulas que el usuario deberá cumplir para usar el mismo."* [9] .

Por lo tanto, se deberán seguir las cláusulas especificadas por el software de terceros utilizado para desarrollar el proyecto.

4.1.1 LICENCIA GNU

Esta es mayoritariamente utilizada por todo aquel software libre [10], ya que sus propias siglas lo indica "*General Public License*" o Licencia Pública General. Todo aquel software publicado bajo esta licencia debe contar con documentación libre con el fin de entender cada una de las funcionalidades de este.

Este tipo de licencia es el utilizado por la herramienta *MtManager*, la cual es utilizada para extraer los datos de los sensores inerciales para su posterior procesado, y también por el gestor de versiones *git,* requerido para llevar un control de las versiones del proyecto, así como contar con una copia de este.

4.1.2 LICENCIA PSF

Este tipo de licencia es el utilizado por el lenguaje *Python* [11],y no establece ninguna limitación para el desarrollo del proyecto, puesto que permite la libre distribución de contenido sin tener que realizar ningún pago, publicidad o derivados.

4.1.3 LICENCIA MIT

Se trata de una de las licencias más famosas en el ámbito informático. Ha sido desarrollada por el Instituto Tecnológico de Massachusetts, y es bastante similar a la licencia GNU, otorgando al usuario bastante permisividad para el uso del software asociado a alguna de ellas, además de ser gratuita.

Dentro del proyecto, múltiples componentes hacen uso de esta licencia, por ejemplo, *Angular, ChartJs* o las librerías utilizadas para el desarrollo de la lógica en el servidor.

4.1.4 LICENCIA BSD

Así mismo, la licencia BSD o *Berkeley Software Distribution* [12], se trata de otra de las licencias más famosas y utilizadas actualmente. Al igual que la licencia MIT, se trata de una licencia libre permisiva y gratuita.

Esta licencia es utilizada por una gran cantidad de librerías incluidas en el proyecto, como *numpy, pandas o matplotlib,* así como por *Flask*. También hace uso de esta licencia el entorno de desarrollo *Jupyter Notebook*.

4.1.5 LICENCIA JETBRAINS

Se ha hecho uso de los entornos de desarrollo proporcionados por *JetBrains* para el desarrollo del proyecto. Esta empresa proporciona licencias a cualquier estudiante para que pueda hacer uso libremente de ellos.

Las limitaciones con las que cuenta, es que únicamente pueden utilizarse para desarrollar proyectos con fines educativos, y no podremos realizar proyectos que vayan a comercializarse.

5. METODOLOGÍA DE TRABAJO Y PLANIFICACIÓN.

5.1 PLANIFICACIÓN INICIAL

La planificación inicial del proyecto viene referenciada en la tabla 1. Se puede ver a continuación y se contemplan las distintas etapas seguidas para el desarrollo del proyecto, así como la duración estimada inicialmente de cada una de ellas. Sin embargo, debido a distintas circunstancias que se han presentado durante el desarrollo del proyecto, la duración real del proyecto ha sido ligeramente superior.

Fases	**Duración Estimada**	**Duración Real**	**Tareas**
Estudio previo	80	100	Tarea 1.1: Análisis del problema. Tarea 1.2: Requisitos de la aplicación Tarea 1.3: Estudio de distintas redes neuronales
Diseño/ Desarrollo/ Implementación	120	130	Tarea 2.1: Diseño de la aplicación Tarea 2.2: Desarrollo de la aplicación Tarea 2.3: Implementación de la red neuronal Tarea 2.4: Entrenamiento de la red neuronal
Pruebas	60	40	Tarea 3.1: Pruebas de funcionamiento con datos reales Tarea 3.2: Evaluación y ajuste de los parámetros de la red neuronal
Documentación	40	50	Tarea 4.1: Realización de la memoria Tarea 4.2: Preparación de la presentación

Tabla 1: Planificación Inicial del proyecto.

En primera instancia, se procedió a analizar como detectar las características angulares en diferentes articulaciones durante la locomoción, con el objetivo de discernir estados de la articulación tanto fisiológicos como patológicos (cojeras de origen osteoartrítico).

A consecuencia de esto, se procedió a conocer cómo obtener los datos provenientes de lo sensores, para ello se decidió utilizar el software *"MT Manager"*, puesto que se trata de software libre, que permite exportar los datos en formato *"TXT",* y con una estructura agradable para su posterior procesado. Por consiguiente, se procedieron a establecer los requisitos de usuario necesarios en la aplicación, con el usuario final de la aplicación.

Posteriormente, una vez familiarizado con el problema a resolver y los datos con los que se contarían, se procedió a estudiar las distintas alternativas a utilizar para implementar la arquitectura de red neuronal.

Puesto que se requería de un *dataset* para el desarrollo de la red neuronal, se procedió con el diseño e implementación de la herramienta, en primera instancia se hicieron varios *"mockups"* para tener una vista preliminar de la aplicación final, así como su posterior validación con las personas involucradas en el proyecto.

Por consiguiente, se ha procedido con el desarrollo de la red neuronal, elaborando los distintos métodos encargados de la extracción y procesado de datos, para el posterior entrenamiento de la red con estos datos, aunque estos métodos no pudieron ser probados y utilizados en el tiempo esperado, debido a las dificultades para obtener tomas, a causa del estado de emergencia declarado el pasado mes de marzo. Además, debido a dificultades durante el desarrollo de la red neuronal, el tiempo requerido para su desarrollo ha resultado superior al esperado.

Una vez realizado el *dataset*, se procedió con el desarrollo y validación de la red neuronal, con el objetivo de añadirla en la aplicación web.

En cuanto a la pruebas, se han realizado pruebas de la implementación de la web, con la finalidad de asegurar que funciona correctamente. Por otra parte, una vez finalizada la red neuronal, se procedió a comprobar la precisión de la red neuronal, con el objetivo de comprobar que puede realizar diagnósticos fiables a partir de tomas no conocidas por la red anteriormente. Una vez revisados los resultados, se procedió a la mejorar la precisión de la red, mediante distintas técnicas descritas en posteriores apartados.

Por último lugar, se ha procedido a realizar la documentación del proyecto, con la realización de un pequeño manual de usuario explicado el funcionamiento de la aplicación y por otro lugar, con la realización de la memoria del proyecto.

5.2 METODOLOGÍA DE TRABAJO.

Se ha seguido una metodología de trabajo incremental, una técnica de trabajo donde se planifican bloques temporales de un tiempo determinado (de 2 a 4 semanas), buscando implementar en cada bloque una serie de tareas definidas según la prioridad que se le ha dado a cada una de estas tareas.

Al final de cada bloque, se puede observar una evolución de la aplicación, para así organizar reuniones con el cliente o usuario final de la aplicación, con el objetivo de que muestren su opinión acerca del producto, así como priorizar una serie de tareas a realizar en el siguiente bloque temporal.

Por ello, se realizaban pequeñas iteraciones de 2-3 semanas con la intención de realizar un producto viable mínimo, de manera que los tutores tuviesen una representación de la futura aplicación final, para así dar su opinión sobre ella, por si se tuviese que realizar algún tipo de cambio.

Con el fin de organizar las tareas de trabajo, se ha utilizado un tablón "*Trello*". Este permite la creación de tablones para conocer las tareas en sus distintos estados:

1. Pendientes.
2. En Progreso.
3. Finalizada.

Este tablón tiene el objetivo de conocer las próximas funcionalidades a implementar, así como la importancia de cada una de ellas, pudiendo observar qué tareas son más importantes que otras. De esta manera se puede conocer qué se encuentra finalizado, y qué tareas se están realizando actualmente.

Con el fin de elaborar la red neuronal, se siguió el siguiente proceso de análisis de datos:

1. Entender el problema: En primer lugar, se entendió el problema a resolver, y los distintos escenarios que pueden darse, así como la estructuración de tareas a seguir durante el desarrollo de la red.

2. Obtención de los datos: Se buscaron distintas alternativas para conocer como extraer los datos de los sensores, y su posterior procesado.

3. Ver los datos: A través de distintas gráficas, se procedieron a observar patrones en los datos, de modo que se pueda visualizar qué diferencia a un perro con cojera de uno sano.

4. Limpieza de los datos: Una vez obtenidos los datos y visualizados, se procedió a eliminar datos duplicados, no válidos (lecturas de los sensores no correcta) y estandarizar los datos.

5. Diseño red neuronal: Se realizo un análisis exhaustivo con el fin de conocer qué tipo de red neuronal se ajustaba mejor al problema, para así proceder posteriormente a su diseño y desarrollo.

6. Entrenamiento de la red: Se desarrolló un método por el cual entrenar los distintos parámetros de la red, de modo que pudiese aprender de los datos de entrada y realizar diagnósticos fiables.

7. Ajustar red: Una vez entrenada, se procedió a ajustar de los hiperparámetros y estructura de la red, con el fin de buscar la máxima precisión posible.

8. Validar con datos: Se procedió a comprobar la precisión de la red, con datos distintos a los de entrenamiento.

6. TECNOLOGÍAS, HERRAMIENTAS E INFRAESTRUCTURAS HARDWARE.

6.1 TÉCNOLOGÍAS Y HERRAMIENTAS EMPLEADAS EN LA APLIACIÓN WEB

Para el desarrollo de la aplicación web, se ha optado por hacer uso de un *framework,* tanto para el desarrollo del cliente como para la elaboración de la web en el servidor, puesto que en su mayoría estos *frameworks* hacen uso de patrones de diseño, como puede ser el patrón *MVC* (se observa en detalle en posteriores apartados), permitiendo así distribuir la estructura del proyecto de una manera clara y sencilla.

Del mismo modo, de cara al futuro, permite llevar un mejor mantenimiento de la aplicación, puesto que la estructura sigue un estándar, permitiendo a otras personas entender cómo se estructura el proyecto.

Estos *frameworks* cuentan con abundante documentación en la red, de modo que si surgen problemas durante el desarrollo, se puede buscar su solución en la red de manera sencilla.

Por otra parte, otro de los motivos ha sido el desarrollo personal, pues así se pueden emplear tecnologías no antes utilizadas, además de reforzar y mejorar los conocimientos en algunos lenguajes y tecnologías.

Como entorno de desarrollo se ha hecho uso de *PyCharm* para la parte del servidor, mientras que para el cliente se ha hecho uso de *WebStorm*. Su elección ha sido debido a que ofrecen una serie de módulos que permiten funciones de autocompletado de código que agilizan el desarrollo de software. Por otra parte, se ofrecen utilidades que permiten buscar errores en la totalidad del proyecto, y posibles soluciones a algunos de ellos.

6.1.1 SERVIDOR

Para la elaboración de la aplicación web, se ha optado por desarrollar una *"API REST",* debido a que permitir separar toda la lógica del servidor de la interfaz de usuario mejora el mantenimiento de la aplicación. De esta forma, por una parte tenemos un proyecto con una serie de funcionalidades para la comunicación con la base de datos (creación, actualización, borrado y vista), y por otra parte tenemos la interfaz web, donde se observan las distintas vistas realizadas.

Acerca de las tecnologías, se ha decidido utilizar *"Python"* como lenguaje, puesto que al hacer uso de inteligencia artificial, este es uno de los mejores lenguajes actualmente para ello, debido a que permite instalar diversas librerías específicas como *"Keras", "Numpy" o "Pytorch"* entre otras. Para su desarrollo como aplicación web, se ha utilizado *"Flask"*, debido a los siguientes motivos:

1. Fácil de utilizar.
2. Compatibilidad con las librerías de inteligencia artificial a utilizar.
3. Ligero.
4. Código abierto.
5. Gran cantidad de documentación.

Si se buscan alternativas a este, se observa *Django,* que aunque cuenta con una mayor cantidad de librerías compatibles, mejor sistema de autenticación y una gran documentación, no suele ser utilizado para proyectos "pequeños" como este, puesto que requiere de una mayor cantidad de recursos para ello, además de ser más complejo de aprender y utilizar en primera instancia.

6.1.1.2 LIBRERÍAS

En la siguiente lista se describen y justifican la diversas librearías empleadas en el proyecto:

- *Flask-Restful*: Implementa muchas funcionalidades para el desarrollo de la aplicación como *"API REST"*.

- *Flask-JWT-Extended*: Ofrece una gran variedad de métodos para la gestión de tokens de tipo *"JSON WEB TOKENS",* los cuales han sido utilizados para la autenticación de usuarios.

- *Flask-SQLAlchemy*: Permite hacer uso de bases de datos relacionales haciendo uso de un *"ORMs (Object Relational Mappings)",* con el fin de abstraerse de la base de datos y tratar al modelo como un objeto.

- *Flask-Cors*: Puesto que se cuenta con una *"API REST",* el servidor y el cliente puede que no se sitúen en el mismo dominio. Por ello se necesita configurar el *"CORS (Cross-Origin Resource Sharing)",* y es por este motivo por el que se ha utilizado esta librería, ya que permite configurarlo correctamente, puesto que de lo contrario, el navegador web no permitiría la comunicación entre ambos por motivos de seguridad.

Para la comprobación del correcto funcionamiento de los *endpoints (*ruta en el servidor encargada de realizar una tarea, como el borrado de un elemento*)*, se ha hecho uso de la herramienta *Postman*, una herramienta que permite enviar peticiones con los distintos métodos *HTTP (GET, POST; PUT o DELETE)* a la dirección requerida, así como la posibilidad de elaborar una batería de tests para así evaluar la aplicación de manera automática.

6.1.2 CLIENTE

Para el desarrollo de la interfaz web se optó por hacer uso de *"Angular"*. *Angular* permite desarrollar software de calidad y bien estructurado, además de contar con muchas facilidades para su desarrollo, de lo contrario, el desarrollo de la interfaz hubiese requerido mucho más trabajo del original.

Debido a que se trata de un *framework* de tipo *SPA (Single Page Application),* renderiza todo el contenido en una única página, por ello no se requiere recargar el navegador al acceder a otras rutas, ofreciendo una mayor fluidez al usuario final.

Otra de las ventajas de su uso es la organización del código, ya que lo estructura siguiendo el patrón de diseño *MVC o Model View Controller*, separando cada uno de estos en distintos archivos para su fácil mantenimiento y desarrollo. En este patrón, el modelo se refiere al contenido de la página (perros, usuarios o tomas). Todas estas entidades se muestran a través de la vista. La interacción entre vista y modelo se realiza mediante el controlador, que se encarga de convertir el modelo para mostrarlo correctamente en la vista. Además, ante cualquier cambio del modelo en la vista, el controlador se encarga de actualizarlo en la base de datos.

Si se compara con sus alternativas, como *VueJs o ReactJs,* este cuenta con una mayor documentación, debido a la gran cantidad de usuarios que lo utilizan, pudiendo encontrar soluciones a múltiples problemas comunes de una manera sencilla . Otra de las ventajas percibidas es su organización del contenido, ya que sus alternativas no distribuyen el contenido de los componentes en múltiples archivos, por lo que se encuentra tanto la interfaz de usuario como su lógica en un único archivo.

6.1.2.1 LIBRERIAS

Por la parte de librerías, se podría destacar *Chart.js*, una librería utilizada para generar múltiples gráficas. Se requiere su uso para mostrar el contenido de los sensores.

Por otra parte, para la correcta autenticación de usuarios en la parte web, se hace uso de *"Angular JWT"*. Se requiere su uso para manejar los tokens de autenticación utilizados, ya que permite comprobar si se trata de tokens válidos, así como de su gestión en la aplicación, permitiendo así la identificación de usuarios en cada petición realizada al servidor.

Para la elaboración de ventanas emergentes, se ha hecho uso de *SweetAlert2*, puesto que cuenta con una gran cantidad de opciones de personalización y por su gran relevancia en la mayoría de los portales actuales.

6.2 TECNOLOGÍAS Y HERRAMIENTAS DE LA RED NEURONAL

Como gestor de dependencias se ha hecho uso de *anaconda*, el cual se trata de un gestor de paquetes para *Python*, que permite la creación de entornos virtuales con distintas versiones del lenguaje y las librerías requeridas. Esto permite tener varios entornos separados, y según el proyecto a realizar, contar con las librerías necesarias y en su versión correcta, evitando que se actualice una cierta librería o la versión del lenguaje, y por temas de incompatibilidades el proyecto no funcione correctamente.

Para el análisis y recogida de datos, se ha hecho uso de las siguientes librerías:

- *Pandas*: Se trata de la librería más utilizada para ello, puesto que permite la creación de estructuras de datos organizadas en tablas para tratar los datos. Así mismo, ofrece una gran cantidad de utilidades para analizar los datos, posibilitando obtener medias, medianas o varianza de los datos obtenidos.

- *Matplotlib*: Se ha utilizado esta librería para poder visualizar los datos, y observar patrones en ellos, ya que ofrece una gran cantidad de utilidades gráficas para ello, como diagramas de caja o dispersión entre otros.

- *sklearn*: Ofrece multitud de herramientas para el procesado de datos.

Por otra parte, para el tratamiento de los datos, se ha hecho uso de la librería *numpy*, ya que permite tratar grandes cantidades de datos de una manera sencilla y con un muy buen rendimiento, debido a que se encuentra implementada sobre el lenguaje de programación *C,* el cual es mucho más eficiente que *Python*.

Por consiguiente, una vez tratados los datos, para la elaboración de la red neuronal, se ha hecho uso de *Pytorch*, una de las librerías más utilizadas en el campo de la inteligencia artificial actualmente, y que cumple con todos los requisitos necesarios para la elaboración del proyecto.

Por la parte de herramientas, se ha hecho uso de *Jupyter Notebook* para el proceso de análisis de datos y diseño de la red. Ofrece un entorno basado en celdas, con el fin de que se pueda ejecutar la sección que necesitas, evitando así tener que ejecutar la totalidad del programa. Por otra parte, permite contar con celdas con formato *"Markdown",* para mostrar conclusiones o secciones de texto relevantes sobre una porción del código, así como exportar el trabajo realizado en múltiples formatos (*HTML* o *PDF* entre otros).

Por último, se ha hecho uso del entorno *Colab*, el cual es idéntico a *Jupyter*, pero este se encuentra desarrollado por *Google*. Permite tener un proyecto en la nube y trabajar con él desde cualquier ordenador. Se utilizó durante el entrenamiento de la red, ya que proporciona el uso gratuito de una tarjeta gráfica, lo que posibilita que el entrenamiento se realice en mucho menor tiempo que si se realiza con el procesador del computador.

6.3 Infraestructuras hardware

Para la toma de datos se ha hecho uso de un par de sensores inerciales de la empresa *Xsens*. En concreto se hace uso del modelo *"MTw Awinda Wireless 3DOF Motion Tracker*". Este tipo de sensores son capaces de captar una gran cantidad de datos, tales como la velocidad angular, rotación o el grado de inclinación en cada momento. En la ilustración 6 se puede observar una representación del sensor.

Ilustración 6: Imagen del sensor utilizado.

Todos estos datos son recabados por el software de la misma empresa *MT Manager*, y posteriormente son exportados a un archivo con extensión *txt*, separando sus valores por coma, con el objetivo de su posterior procesado en la aplicación.

7. ANÁLISIS.

7.1 ELEMENTOS DE LA APLICACIÓN

Dentro de la aplicación de distinguen dos elementos claves para su desarrollo, por una parte se visualizan las sesiones, y por otra parte las tomas.

Las sesiones son las encargadas de seguir la evolución de un perro cuando se le aplique algún tipo de tratamiento, con la finalidad de observar si dicho tratamiento ha resultado favorable o no.

Cada una de ellas, se compone de al menos una toma, estas se tratan de evaluaciones realizadas al perro, con el par de sensores colocados en la extremidad a analizar, o si se desea analizar la cojera, se sitúan en la grupa y cabeza del animal.

Para su evaluación, es necesario aportar los datos obtenidos del sensor, así como de los videos tomados durante su elaboración, con el objetivo de visualizar los datos en gráficas, al mismo tiempo que se visualiza el video. Además, cada toma ofrece la posibilidad de añadir un mensaje elaborado por el experto veterinario, con sus observaciones acerca de dicha toma.

7.2 ROLES DE USUARIO EN LA HERRAMIENTA

Se han observado dos roles de usuarios distintos en la aplicación por su naturaleza, ya que en este caso será una herramienta personal para el veterinario, y únicamente tendrán acceso los usuarios creados por el administrador.

7.2.1 ADMINISTRADOR

Por una parte, se encuentra el administrador del portal, que tendrá acceso a la totalidad de funcionalidades de la web. Sin embargo, su principal funcionalidad será gestionar usuarios, debido a que es el único rol que puede realizarlo.

7.2.2 VETERINARIO

Por otra parte, se observa el rol principal del portal, los "veterinarios". Este rol podrá realizar las siguientes funciones:

- Gestión de perros: altas de caninos en el sistema, actualización de sus datos, visualización de imágenes, etc…
- Visualizar las sesiones de los perros, teniendo la posibilidad de crear nuevas o editarlas.
- Visualizar las tomas, con el fin de observar las características angulares de los perros analizados, y observar la conclusión dada por la red neuronal.

7.3 ANÁLISIS PROBLEMA

En relación con el análisis del problema, se distinguen dos problemas principales a resolver. Por una parte, se tendrá que hallar el ángulo relativo formado por los dos sensores colocados en alguna de las extremidades del perro, con el fin de que el experto pueda observar el rango de movimiento de la articulación a analizar. Por otra parte, se tendrán que observar patrones en los datos obtenidos por los sensores, con la finalidad de discernir que elementos juegan un papel relevante a la hora de observar si un perro tiene cojera o no.

7.3.1 CARACTERÍSTICAS TOMAS

Acerca de las tomas, en primer lugar cabe destacar que estas tomas no pueden realizarse en cualquier tipo de perro, puesto que no pueden realizarse en perros pequeños, como podrían ser chihuahuas, carlinos o bulldogs, debido a que al situarle el par de sensores en cabeza y grupa, se sentirían incomodos, ocasionando movimientos bruscos durante la toma que lleven a obtener datos inconsistentes.

La cantidad de restricciones que existen para la toma de datos ha dificultado enormemente la toma de datos para el proyecto. Esta toma de datos está sujeta a importantes limitaciones: la llegada de caninos a la clínica veterinaria de la ULPGC que cumplan las características anteriores, que el profesional observe que sea un perro adecuado para la toma de datos, que sus dueños acepten participar en el estudio, etc… Por este motivo, nos hemos encontrado con serias limitaciones a la hora de conseguir un *dataset* con suficientes datos y una amplia variedad de perros.

En cuanto al conjunto de datos para el proyecto, se cuenta con un total de 15 tomas, de las que 12 se utilizarán para el entrenamiento de la red, y 3 tomas para la posterior comprobación del entrenamiento. De estas 15 tomas, 7 corresponden a perros sanos y 5 a perros con cojera, por lo que se cuenta con un *dataset* algo escaso, debido a los motivos anteriormente comentados. Por

este motivo, hay que tener en cuenta que el resultado de la red no será tan fiable como cabe esperar.

7.3.1 PROBLEMA COJERA

El principal indicio de cojera en un perro tiene que observarse con la amplitud al andar en la parte trasera del perro, en concreto en la parte de la grupa, ya que en el caso del sensor delantero, los datos obtenidos de ellos son muy inconsistentes por el constante movimiento del perro al andar, y resulta muy complicado estandarizarlos. En la ilustración 7 se puede observar la localización de los sensores en un perro.

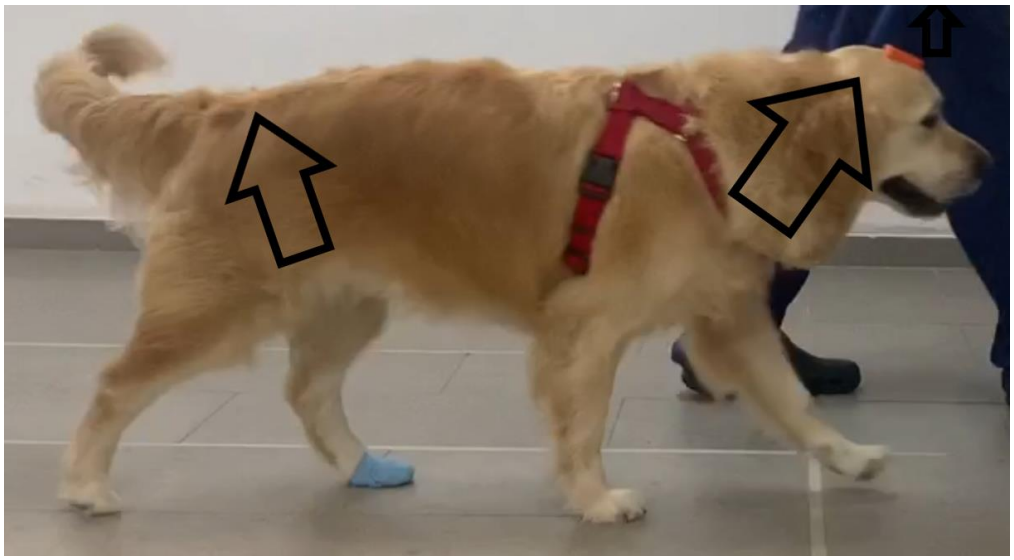

Ilustración 7: Localización de sensores en un perro.

El par de sensores permite captar la amplitud del perro al andar, donde en el caso de estar sano, el rango de movimiento debería de ser bastante amplio en ambas extremidades y simétrico, mientras que en perros con dolencias, este debería de ser asimétrico.

Tras haber obtenido una serie de muestras de datos reales, se procedió a analizar los datos obtenidos por los sensores. Estos nos ofrecen un parámetro conocido como *"Roll",* el cual nos indica el grado de inclinación del sensor en cada momento. En la ilustración 8 se puede observar una muestra de un perro sano y su rango de movimiento.

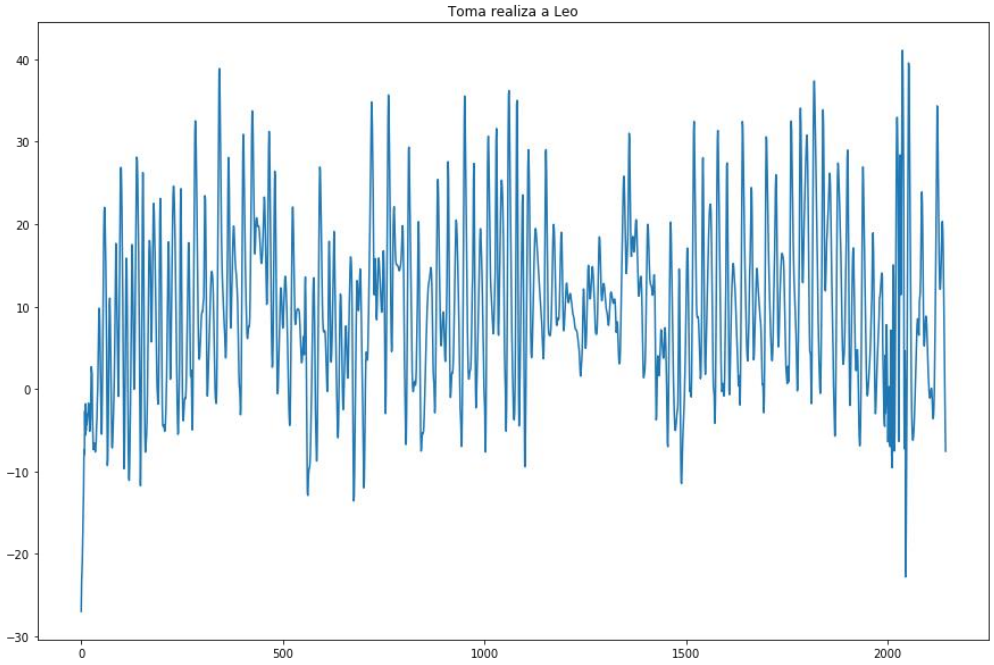

Ilustración 8: Grado de amplitud al caminar de un perro sano.

Se puede observar que sigue un patrón ascendente y descendente bastante homogéneo, teniendo unos máximos y mínimos siguiendo un patrón coherente, a excepción de algunos datos que exceden, que pueden ser debidos a un movimiento brusco del perro a la hora de realizar la tomas.

Por otra parte, en la ilustración 9, se puede observar una gráfica que representa una toma realizada a un perro con cojera. Como se puede apreciar, presenta un rango de movimiento mucho menos homogéneo, ya que al apoyar la pata en la que presenta el dolor o molestia, necesita hacer una mayor fuerza con la otra, para evitar así tener dolor al andar.

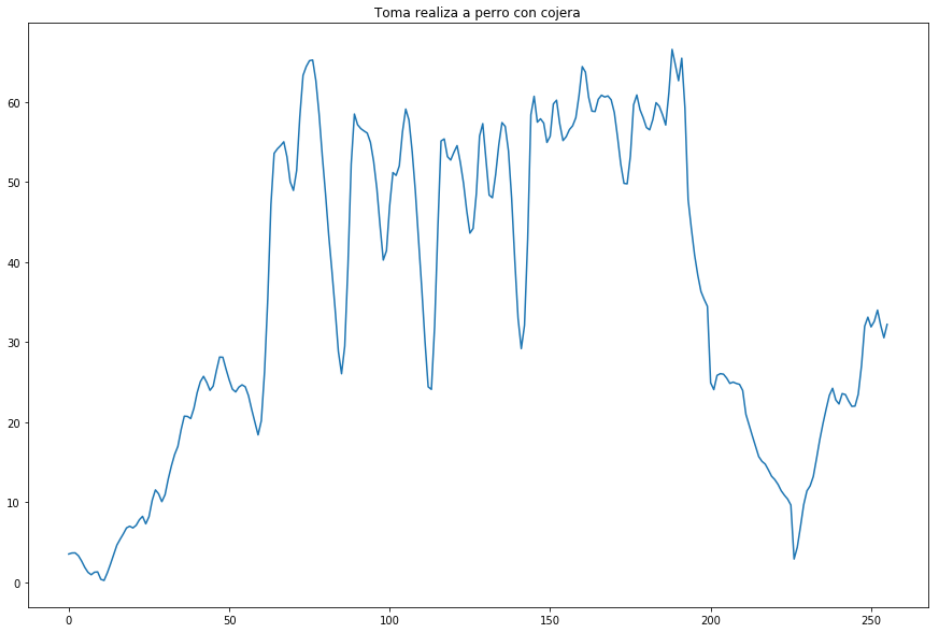

Ilustración 9: Grado de amplitud al caminar de un perro con cojera.

A simple vista una persona podría discernir un perro cojo de uno sano, por lo que una red neuronal podría aprender de estos datos y ser capaz de ello.

Para aplicar inteligencia artificial en este problema, no se puede aplicar una red neuronal tradicional, puesto que los datos son secuenciales y no son independientes unos de otros. Es necesario observarlos como una serie temporal para llevar a cabo una conclusión. En el apartado de diseño se podrá observar el tipo de red utilizada, y las distintas observaciones que se han tenido en cuenta para su desarrollo.

7.3.1.1 LIMPIEZA DE DATOS

En las ilustraciones anteriores se pudo observar el patrón de comportamiento de perros sanos o con algún tipo de dolencia, pero al observar los datos más detenidamente existen bastante datos "fuera de serie". Es decir, durante el inicio de la toma de datos el sensor ofrece una serie de datos que no caracterizan el andar del perro, ya que puede ser que se encuentre quieto en primera instancia o al finalizar la toma puede que se produzcan movimientos bruscos que causen que los datos tengan una serie de imperfecciones. Por ejemplo, en la ilustración 10 se pueden apreciar diagramas de caja representando algunas de las tomas realizadas.

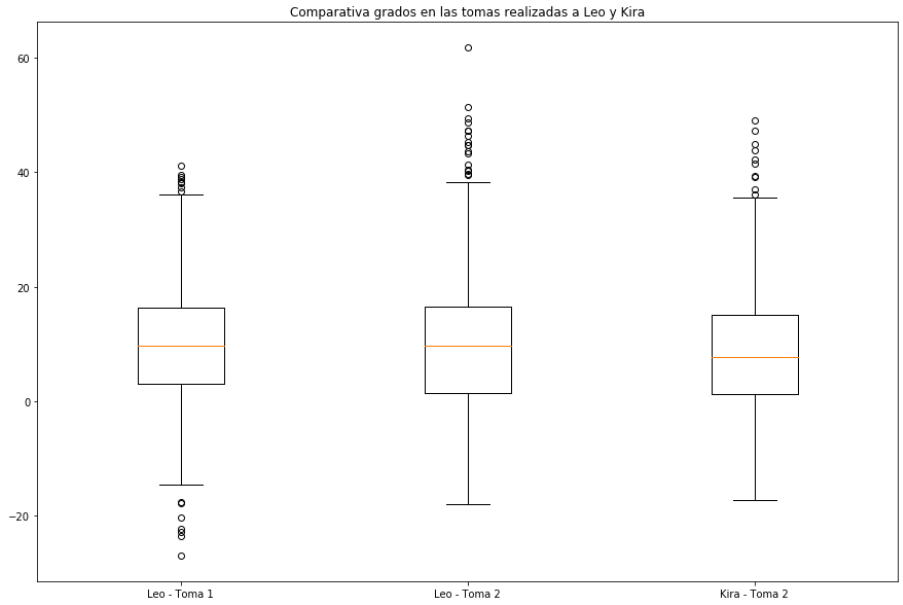

Ilustración **10**: Comparativa de datos entre perros sanos.

En la ilustración se puede observar que el conjunto de datos en su mayoría se concentra en el rango de 0 a 20 grados de inclinación, pero se pueden observar bastante datos fuera de serie, y esto podría ocasionar una dificultad para la red a la hora de poder generalizar los datos de entrada.

Por lo tanto, hay que limpiar los datos de entrada. En primera instancia se ha procedido a eliminar una serie de datos, por una parte se eliminarán todos los datos que se encuentren por debajo del 10% del total, mientras que por

otra parte, se eliminarán todos los que se encuentren por encima del 90% del total. Estos datos son los denominados generalmente *outliers*. En la ilustración 11 se puede observar cómo han cambiado estos datos.

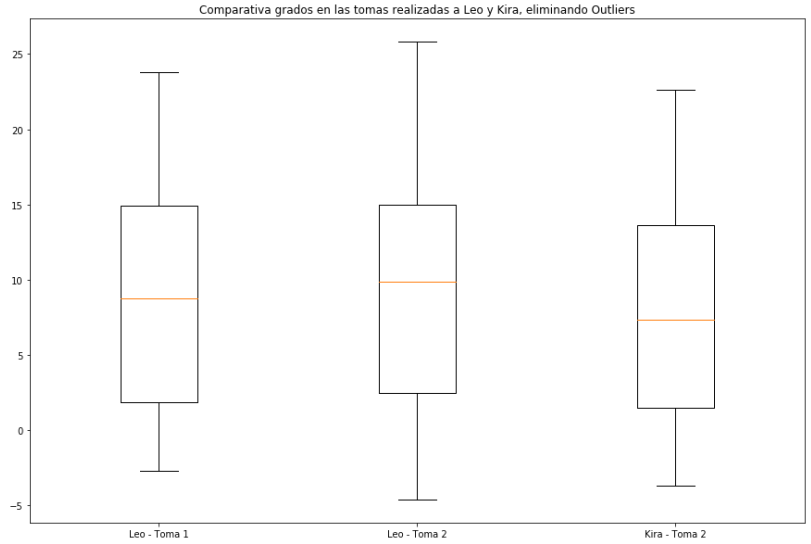

Ilustración 11: Comparativa de datos entre perros sanos eliminando "Outliers".

De esta forma los datos se encuentran más homogéneos, situándolos en un rango bastante similar. En las ilustraciones 12 y 13 se pueden observar cómo se representan actualmente los datos y sus diferencias respectos a las ilustraciones 8 y 9.

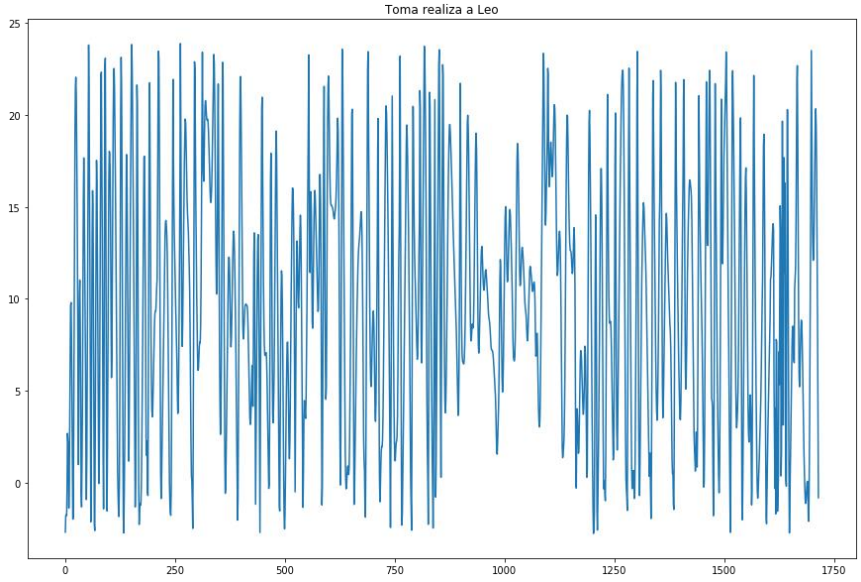

Ilustración 12: Visualización datos perro sano.

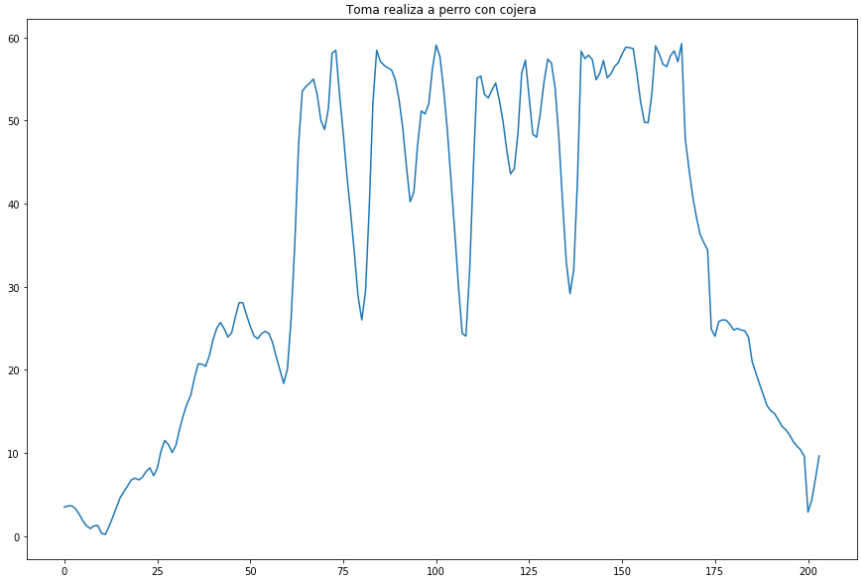

Ilustración 13: Grado de amplitud al caminar de un perro con cojera.

7.3.2 CARACTERÍSTICAS ANGULARES

Para el cálculo del ángulo relativo entre los dos sensores, se realizaron múltiples tomas específicas para ello. Sin embargo, no se ha podido confirmar la veracidad de los datos, puesto que los videos ofrecidos para estas tomas no permiten observar el ángulo formado en la extremidad.

En la ilustración 14 se puede observar donde se deberían de colocar los sensores, con el objetivo de conocer el ángulo formado por la posición de los sensores. Para así conocer, el rango de amplitud de la extremidad.

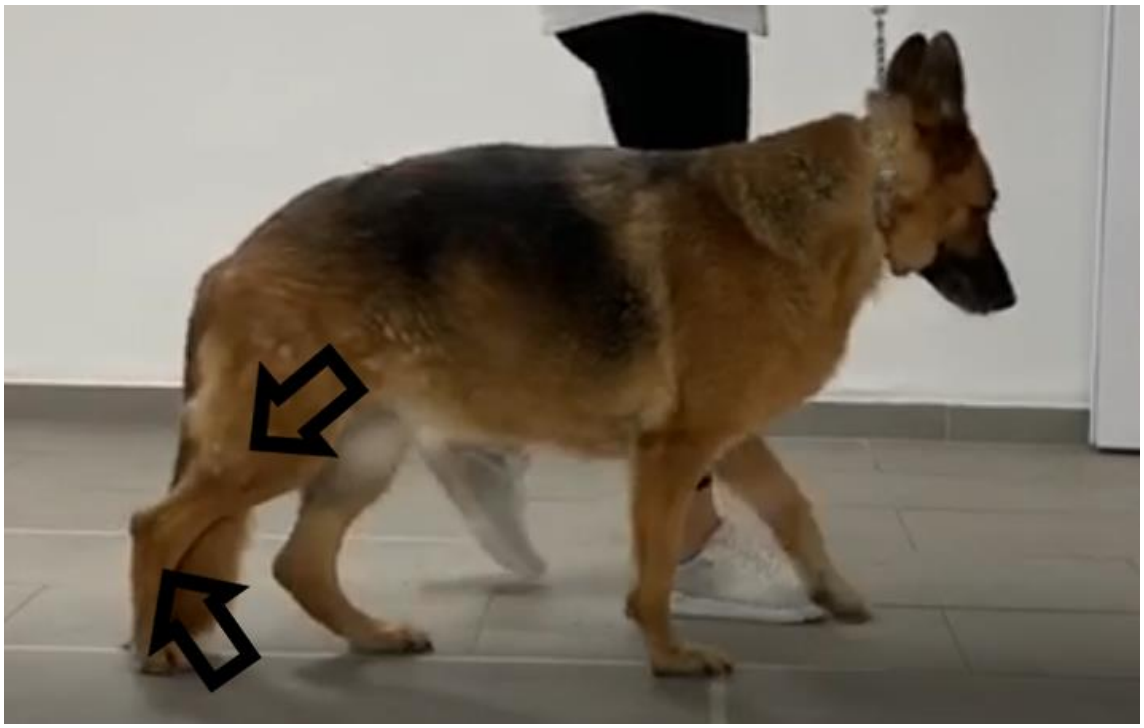

Ilustración 14: Localización sensores, para obtención ángulo relativo.

Con el objetivo de conocer, si existe alguna relación trigonométrica para la obtención del ángulo relativo, se observaron los distintos parámetros que ofrecen los sensores.

Entre los distintos datos que ofrece, se puede conocer la orientación en cualquiera de los tres ejes en cada momento, siendo este parámetro necesario para ello, puesto que si se coloca el sensor de manera vertical en cualquier de la extremidad, en el instante inicial el sensor deberá ofrecer en el eje "x" un valor de 0º, durante la locomoción. De esta forma, si se realiza la diferencia entre la orientación del sensor situado en la parte superior y la del sensor situado en la parte inferior de la extremidad, se podrá obtener el ángulo relativo de los sensores.

En la ilustración 15, se puede observar los datos obtenidos de una de las tomas realizadas, donde se puede apreciar que el ángulo formado por los sensores oscila entre los 15 y 100 grados.

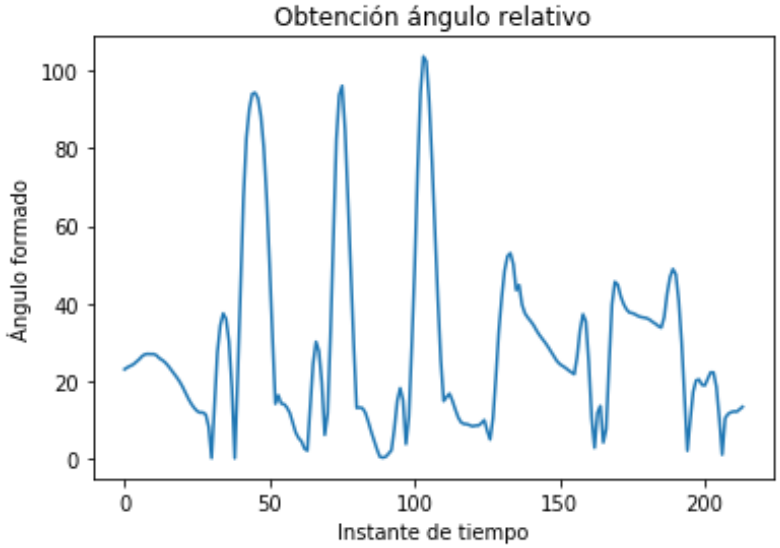

Ilustración 15: Mostrar ángulo relativo,

8. DISEÑO.

8.1 ARQUITECTURA DEL SISTEMA

La aplicación en su conjunto sigue la estructura cliente-servidor, siendo el servidor una *"API REST"* como se comentaba anteriormente, mientras que el cliente hace uso de *Angular*. En la ilustración 16 se puede ver como interaccionan las distintas partes.

En esta estructura, el cliente y el servidor se comunican a través del protocolo *HTTP*, haciendo uso de los distintos métodos de petición enumerados a continuación:

- *GET*: Se trata del método de obtención de recursos del servidor, como la obtención de perros, sesiones o tomas.
- *POST*: Se hace uso para la creación de nuevo contenido en la página, se envía todo el contenido a través del cuerpo de la petición, suministrando los distintos parámetros requeridos por el servidor.
- *PUT:* Es bastante similar al método *POST*, la diferencia radica en que esta se utiliza para la actualización de contenido.
- *DELETE:* Se utiliza para el borrado de contenido en el portal.

Para el correcto aseguramiento del servidor, y que únicamente realicen las operaciones los usuarios autorizados, se ha uso de tokens, que se generan cada vez que un usuario se autentica en el servidor, El tipo de token utilizado se denomina *"JSON WEB TOKEN"* [13], uno de los más utilizados actualmente para la autenticación de usuarios. En posteriores apartados se observará en detalle este tipo de token.

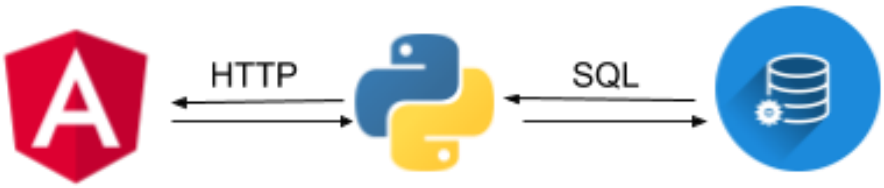

Ilustración 16: Comunicación entre los componentes de la aplicación.

8.2 DISEÑO DE LA BASE DE DATOS

Para la persistencia de los datos, debido a la naturaleza y sencillez de la aplicación, se hace uso de una base de datos tipo *SQLITE3*, puesto que si se observan alternativas de bases de datos relaciones como *MySQL*, *MariaDB* o *PostgreSQL*, estas últimas requieren de un servidor de base de datos exclusivo para ellas, mientras que *SQLITE3*, hace uso de un archivo para el almacenado de archivos, evitando así la configuración de un servidor.

Tal y como se comentaba con anterioridad, se ha usado una librería que se encarga de la gestión de la base de datos, como la creación de las distintas tablas necesarias y de todas las relaciones entre ellas. Además, también permite manejar los datos sin hacer uso de sentencias *SQL,* permitiendo tratar a cada una de las entidades como objetos, de forma que mediante funciones se pueden realizar las consultas con el mismo nivel de detalle que haciendo uso del lenguaje tradicional.

Dentro de la página, se observan varias entidades, en primera instancia se observa la entidad “Perro”, y como su propio nombre indica, se encarga de representar a los perros creados en el portal. Cuenta con algunos datos descriptivos del mismo, como puede ser el nombre, fecha de nacimiento, raza, peso o altura, ya que algunos de estos datos pueden resultar relevantes a la hora diagnosticar algún tipo de cojera.

Por otra parte, los perros tienen asociadas múltiples sesiones. Esta entidad se utiliza para representar las sesiones de trabajo de un veterinario con un perro, con el fin de almacenar múltiples tomas de datos. Las tomas son las encargadas de albergar los datos obtenidos por los sensores, así como su posterior procesado con el fin de mostrar las distintas características angulares, así como procesar los datos en la red neuronal para ofrecer diagnósticos sobre el grado de cojera del perro.

Por último, se observa el usuario del portal, mostrando algunos datos descriptivos como el nombre o el apellido. Así mismo, cuenta con un campo para conocer el rol del usuario (Administrador o Veterinario). Con el fin de autenticarse, se observa el email y la contraseña. La contraseña por motivos de confidencialidad y seguridad se cifra haciendo uso de un algoritmo *SHA-256*.

8.3 ALMACENAMIENTO DE ARCHIVOS EN EL SERVIDOR

Como se ha comentado anteriormente, el portal permite la subida de archivos, como son los videos de las distintas tomas, y por otra parte, los datos obtenidos por los sensores. Para ello se ha ideado una estructura jerárquica para almacenarlos. En la ilustración 17 se muestra un esquema de la estructura definida.

En primera instancia, cada perro cuenta con su propia carpeta, el nombre se compone del prefijo *"dogFolder",* y se le añade el identificador del perro, con el fin de garantizar de que no existan dos carpetas con el mismo nombre.

Por consiguiente, por cada sesión creada, existe una carpeta, donde el nombre sigue la misma estructura, pero en este caso, el prefijo es *"Sesion"*. En el siguiente nivel, se observan las tomas, que siguen la misma estructura comentada anteriormente, y por último se observan los archivos relativos a los sensores y los videos.

Tanto videos como datos se encuentran restringidos a una serie de extensiones permitidas. Por la parte de lo sensores, se permiten los archivos con extensión *"txt" o "csv"*. En cuanto a los vídeos, se permiten videos con extensión *"mp4" o "mp3"*. Se ha establecido un límite en el tamaño de los archivos que es *2 MB*. Una vez eliminado un perro se procede a eliminar la carpeta entera y con ello todos los datos de este.

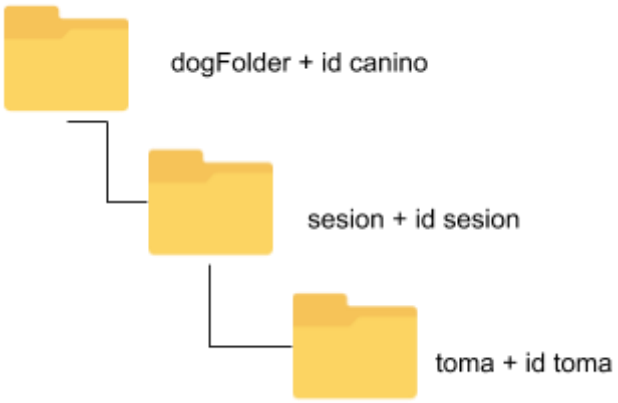

Ilustración 17: Estructura archivos en la aplicación.

8.4 DISEÑO INTERFAZ DE USUARIO

Con relación al diseño de la interfaz de usuario, en primera instancia se realizaron diversos *mockups* haciendo uso de la herramienta *Figma*. Esta herramienta permite realizar bocetos y compartirlos con múltiples usuarios de manera sencilla. Los bocetos se realizaron con el fin de tener una visión de la aplicación final, así como verificar que la interfaz cumpliese con los requisitos deseados por el usuario final.

Por una parte, se realizaron dos vistas distintas en relación con los perros, en primer lugar se realizó una vista para la visualización general de perros en el portal, con el fin de mostrar los perros registrados en el portal, dejando la posibilidad al usuario de filtrarlos por el nombre del perro a mostrar, u ordenarlos por distintos parámetros como la fecha de creación o el mismo nombre. En la ilustración 18 se puede observar el boceto inicial.

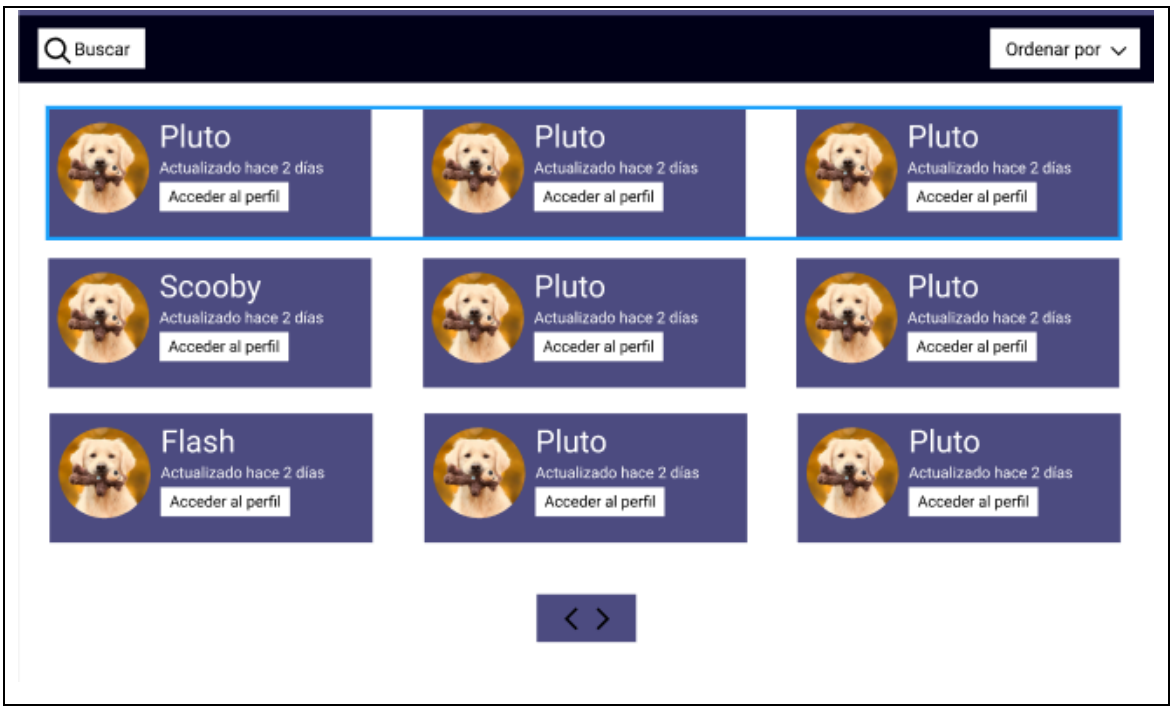

Ilustración 18: Mockup vista general perros.

De cara a la aplicación final, se ha modificado el formato elegido para mostrar la información de los caninos, aunque se sigue mostrando la imagen del perro en cuestión, (si no cuenta con una, se establece una por defecto), así como la

fecha de actualización de alguno de sus campos. En la ilustración 19 se puede observar la vista en la aplicación final.

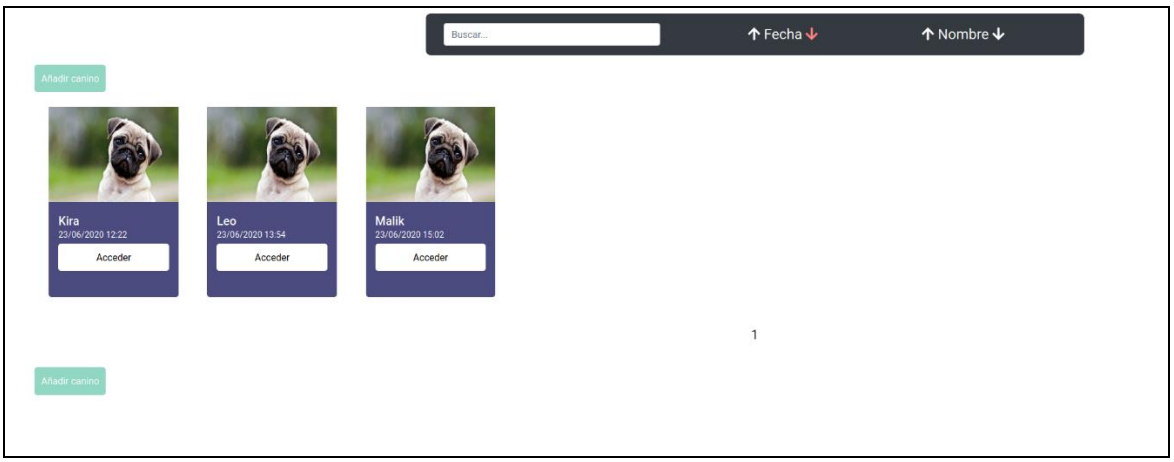

Ilustración 19: Vista general de perros en la aplicación.

En la siguiente vista, se puede acceder a un perro en concreto, mostrando información de este, así como las sesiones con las que cuenta el perro. Tal y como hemos explicado, generalmente para cada sesión tendremos un conjunto de tomas realizadas en un mismo día para visualizar el comportamiento del perro. El sistema permite dar de alta sucesivas sesiones y tomas dentro de ellas, con la finalidad de poder observar la evolución que ha tenido el perro ante un cierto tratamiento.

Asimismo, accediendo a cualquiera de las sesiones, nos muestra las distintas tomas que lo componen. Al acceder a una toma, se muestra una vista para el análisis de los datos obtenidos de los sensores. En esta vista, se muestran los videos vinculados a la toma en cuestión, admitiendo 3 videos como máximo. En la ilustración 20 se observa la visualización de los videos en la aplicación final.

Ilustración 20: Vista de videos en la aplicación final.

Por otra parte, se ofrece la posibilidad de visualizar dos gráficas, en una de ellas se puede observar los datos sobre la cojera en el perro, mientras que la otra gráfica, ofrece el ángulo relativo entre los dos sensores colocados en alguna extremidad. La visualización de datos se sincroniza con la reproducción del video. En la ilustración 21 se observa la visualización de gráficas en la aplicación.

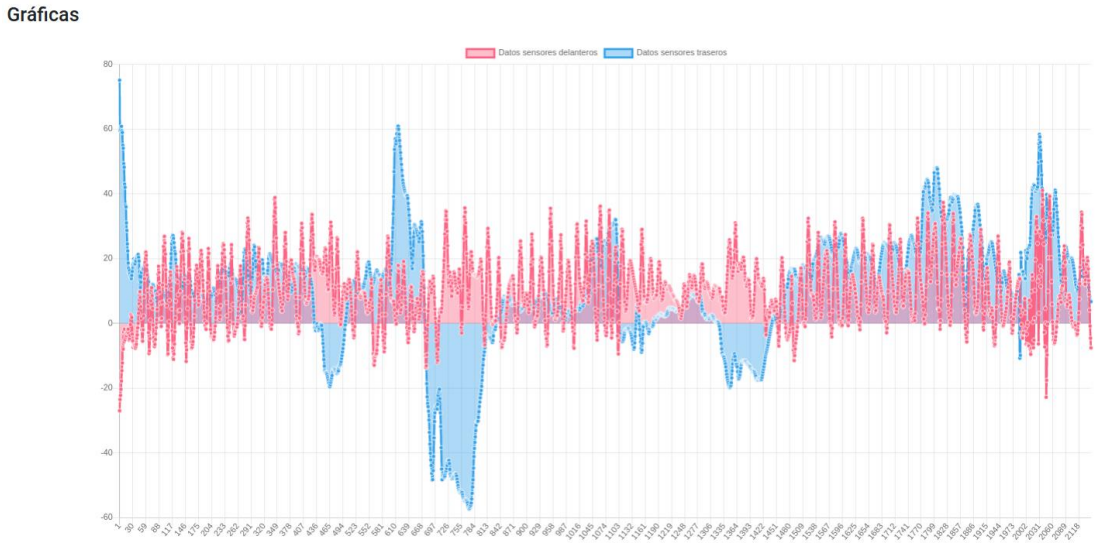

Ilustración 21: Visualización de gráfica de cojera en la aplicación final.

8.5 DISEÑO RED NEURONAL

Como se comentaba en el apartado de análisis, no se puede hacer uso de una red neuronal tradicional. Los datos obtenidos de los sensores son series temporales, donde un dato en un momento determinado depende de los datos anteriores. Se pueden observar casos similares en otros campos, como puede ser en el procesamiento de lenguaje natural, donde se procede a implementar redes que sean capaces de entender el vocabulario humano, y dependiendo de la frase, ciertas palabras pueden tener un cierto significado u otro.

Para ello se hace uso de redes recurrentes (*RNN*), ya que son capaces de recordar los eventos anteriores, siendo en este caso las palabras anteriores, para entender el contexto de cada una de las palabras siguientes, y si el objetivo es traducir una frase por ejemplo, ciertas palabras tendrían un significado dependiendo de las palabras que la preceden, permitiendo así elaborar mejores traducciones. Este tipo de redes también se utilizan en el precio de acciones, que aunque se trata de un sector muy volátil, debido a que es difícil predecir qué ocurrirá en el futuro, suelen desempeñar un papel bastante bueno. Pero el campo donde mayoritariamente se utilizan este tipo de redes es en el de reconocimiento de voz, como se comentaba con anterioridad. La mayoría de los gestores virtuales como *Alexa, Google o Siri* [14] llevan una gran cantidad de años utilizando este tipo de redes, ya que deben de ser capaces de escuchar y entender cada una de las peticiones del usuario, como “poner un recordatorio” o “¿Qué tiempo hace?”.

8.5.1 RNN Y LSTM

Las *Recurrent Neural Networks* [15] hacen uso de neuronas donde la entrada está compuesta por los datos en ese instante de tiempo, pero también toman en cuenta los datos obtenidos en instante pasados, con el objetivo de conocer los estados anteriores para tomar la decisión actual. Así mismo, existen variantes bidireccionales, donde también tienen que conocer los datos posteriores.

Pero estas redes sufren de lo que se conoce como *The Vanishing Gradient Problem* [16], puesto que al ir propagando en el tiempo las entradas, si la secuencia es muy larga, a la hora de ir aprendiendo los patrones, las neuronas que se encuentran "más alejadas" de la salida, apenas podrán aprender nada nuevo, y por ello no serán capaces de generalizar los datos de entrenamiento.

Es por ello por lo que se ha hecho uso de una de sus variantes, conocida como *LSTM (Long Short Term Memory)* [17] [18], que se diferencia por la gran cantidad de operaciones que se realizan dentro de cada una de ellas. En la ilustración 22 se observa la diferencia entre las dos arquitecturas.

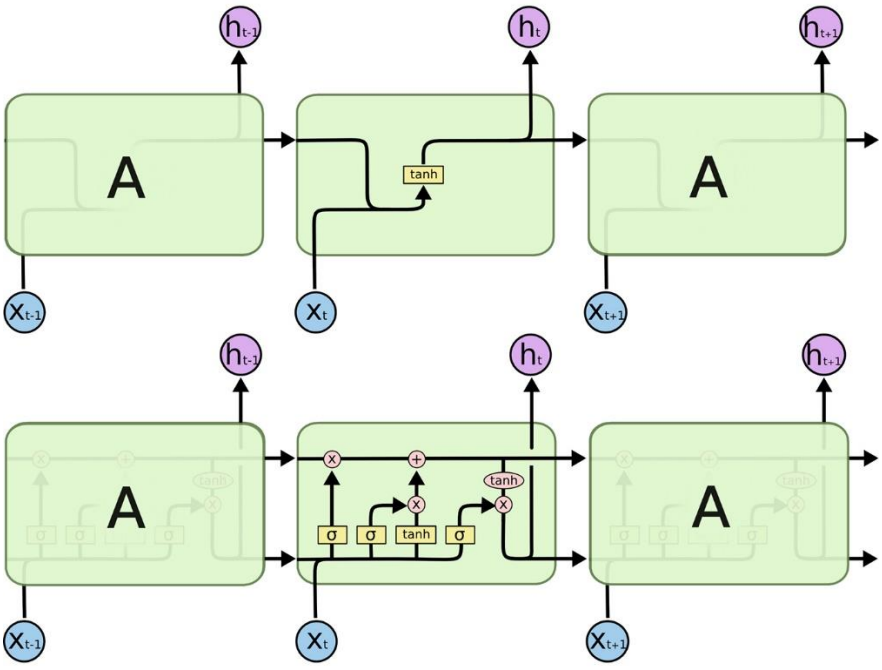

Ilustración 22: Comparativa RNN (imagen superior) y LSTM (imagen inferior).

En la ilustración 22 se puede observar las distintas operaciones que realiza cada una ellas. Mientras que la *RNN* se limita al uso de una función de activación, la red *LSTM* hace uso de múltiples operaciones como sumas, restas y productos, combinando todas estas operaciones con funciones de activación.

Estas operaciones se realizan ya que este tipo de redes no proporcionan un único estado a las siguientes neuronas, debido a que hace uso de dos estados distintos. Por una parte, se observa el estado a largo plazo (traza superior de la ilustración 22) de los estados anteriores, que contiene un espectro de información bastante amplio. Por otra parte, se le proporciona los datos previos (traza inferior de la ilustración 22), pero a corto plazo.

Cada neurona tiene la capacidad de "recordar" o eliminar parte de la información "más antigua" en función de la entrada, y de los datos "más recientes", dependiendo del conjunto de entrada suministrado a la neurona en un cierto instante.

Aunque evitan bastante bien el *The Vanishing Gradient Problem,* se deben limitar la cantidad de datos de entrada [19] a la red durante el entrenamiento, puesto que ante secuencias muy largas (+500) pueden tener dificultades para el aprendizaje. Es por ello por lo que se ha limitado a 250 el tope máximo de datos provenientes de los sensores, puesto que la mayoría de las tomas realizadas cuentan con 400 valores aproximadamente. Sin embargo, muchos de ellos son eliminados tal y como se observó en la fase de limpieza de datos, por lo que la mayoría de las tomas suelen oscilar entre 200-300 valores. En caso de que la toma tenga menos datos, los datos serán rellenados con 0, empezando por la posición 0, para así representar que el perro está quieto y no influya en el aprendizaje. Hemos tomado esta decisión ya que si procedemos a añadir estos 0 al final, la red no será capaz de generalizar correctamente los datos de entrada y nos puede llevar a equivocaciones. En el caso de que la toma cuente con un mayor número de datos, se procederá a

trocear dicha toma en varias tomas de 250 datos cada una, obteniendo así un mayor número de tomas para el entrenamiento.

A continuación se indican algunos parámetros establecidos inicialmente para la red neuronal:

- Número de entradas: 1.
- Capas *LSTM*: 2.
- *Dropout*: 50%.
- Número de salidas: 2.
- Número de neuronas en cada celda: 256.

Anteriormente se comentó que se contaría con una longitud de secuencia de 250 datos, pero cada uno de ellos cuenta con el valor del sensor en ese instante, motivo por el cual se cuenta con una única entrada.

También se ha decidido hacer uso de *Dropout* [20] para la fase de entrenamiento. Esta técnica se basa en "congelar" aleatoriamente un número determinado de neuronas, siendo en este caso del 50%, de forma que estas neuronas son ignoradas durante la fase de actualización de parámetros, produciendo así que las neuronas que se encuentren funcionando cuenten con una mayor relevancia, produciendo así ligeros cambios en los datos de entrenamiento, y que la red sea capaz de generalizar mejor los datos de entrada. Esta técnica se utiliza exclusivamente durante el entrenamiento.

Por otro lado, el sistema cuenta con 2 salidas, representando 0 para perros sanos y 1 para perros con dolencia. Sin embargo, esta salida no se refiere a la red *LSTM* en sí. Este tipo de redes se suele complementar con una red neuronal tradicional (*Fully Connected Layer*), que será la encargada de procesar los datos de entrada y concluir si un perro puede contar con cojera o no. A esta red se le proporciona un vector de 256 valores y realiza el diagnostico final.

Por último, al tratarse de un problema de clasificación binaria, se le aplica la función sigmoide, una función de activación que ante cualquier entrada, ofrece una salida entre 0 y 1, representando en este caso a un perro sano de

uno que no. En la ilustración 23 se puede observar una representación de este primer diseño de la red neuronal representada en *Pytorch*.

```
class LSTM(nn.Module):
    def __init__(self, hidden_size = 256, input_size = 1, output_size = 1, dropout = 0.5, num_layers = 2):
        super(LSTM, self).__init__()

        self.input = input_size
        self.hidden = hidden_size
        self.output = output_size
        self.n_layers = num_layers


        # Definición LSTM
        self.lstm = nn.LSTM(
            input_size = input_size,
            hidden_size = hidden_size,
            num_layers = num_layers,
            batch_first = True,
            dropout = dropout
        )

        # Definición Fully-Connected
        self.fc = nn.Linear(
            hidden_size,
            output_size
        )

        self.sigmoid = nn.Sigmoid()
```

Ilustración 23: Implementación de la red neuronal.

En posteriores apartados se observarán otros diseños llevados a cabo, modificando ciertos parámetros, como el número de neuronas o la longitud de secuencia utilizada. Además se explicará cómo se ha entrenado la red y los distintos hiperparámetros utilizados para ello.

9. DESARROLLO

A continuación, se mostrará el desarrollo de los distintos componentes de la aplicación. En primer lugar se explicará el desarrollo de la parte del cliente, comentando los elementos relevantes. A continuación, se tratará el desarrollo del servidor y se finalizará tratando los aspectos del desarrollo de la red neuronal, comentando las distintas arquitecturas probadas, y los diferentes hiperparámetros escogidos entre otros tópicos.

9.1 DESARROLLO DEL CLIENTE

9.1.1 ESTRUCTURA DEL CONTENIDO

En la ilustración 24 se puede observar la estructura utilizada para organizar el código.

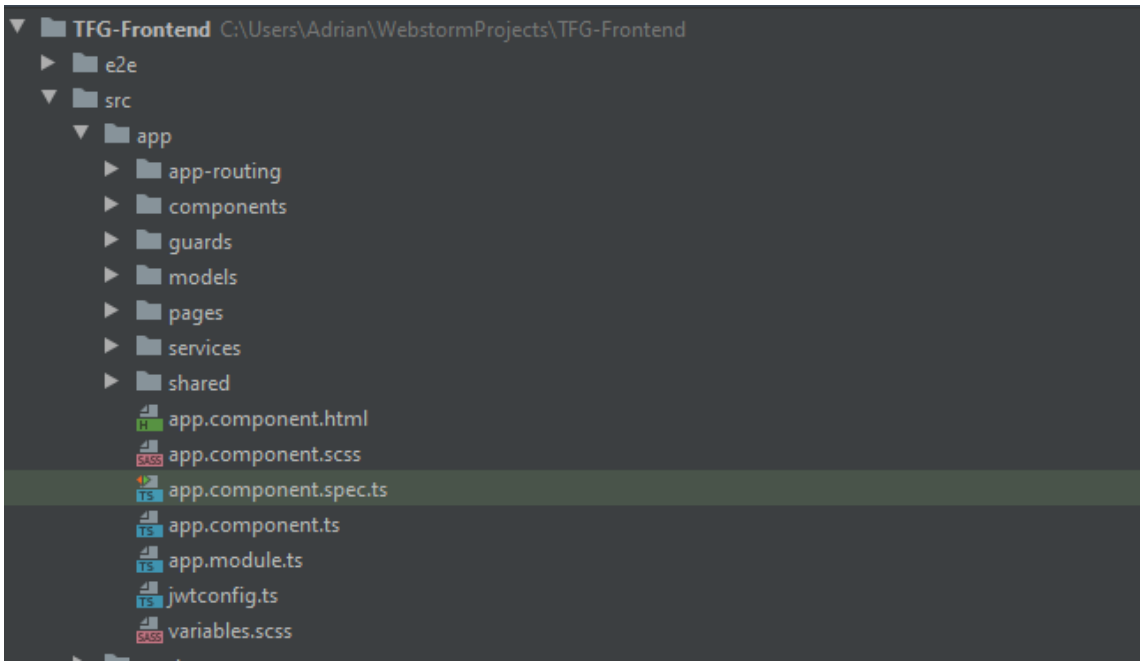

Ilustración 24: Estructura del proyecto en Angular.

En primera instancia se observa el módulo principal *"app",* que contiene la página de inicio, siendo la página de inicio de sesión en este caso. También se encarga de gestionar la totalidad de rutas en la página, haciendo uso de *"lazy-loading"* , una técnica que carga los módulos requeridos en cada momento, evitando así tener módulos cargados que no van a ser utilizados y por lo tanto, mejorando los tiempos de carga.

A continuación se muestran los distintos modelos utilizados en la página, que se encuentran en el directorio *"models"*. Por otra parte, se observa el directorio *"services"*. La aplicación cuenta con varios componentes que son utilizados en diferentes módulos, y con el fin de reutilizar algunos de ellos, se ha creado un módulo específico en el directorio *"shared"*.

El contenido de cada página se encuentra situado en el directorio *pages*, organizando así mejor el contenido de cada una de ellas para su correcto desarrollo y mantenimiento.

Para controlar el acceso a los distintos módulos, se hace uso de los denominados *"guards"* en *Angular*, que se encargan de controlar el acceso de los usuarios a ciertas rutas en la página. A continuación se observarán los distintos *guards* creados, que se encuentran situados en el directorio con ese mismo nombre.

9.1.2 SERVICIOS

Tal y como se comentaba con anterioridad, existe un directorio específico para los servicios utilizados. Estos se dividen en archivos según el modelo o funcionalidad a tratar, existiendo por ejemplo uno para las tomas, otro para la autenticación de usuarios, etc…

Cada uno de estos servicios se comunica con los métodos implementados en el servidor mediante los distintos métodos *HTTP (POST, GET, etc…).* Para verificar la autenticidad del usuario que realiza la petición, se hacen uso de tokens *JWT*, con el objetivo de que usuarios sin permisos realicen ciertas acciones, como eliminar usuarios o perros del portal.

Acerca del servicio de autenticación de usuarios, este utiliza el almacenamiento del navegador (*LocalStorage*), para la persistencia del token, con el fin de que mantenga la sesión iniciada pese a cerrar la pestaña actual.

Si el usuario no cuenta con el token almacenado, o en caso de que este se encuentre, pero no sea válido (puesto que los tokens cuentan con un tiempo de validez predeterminado, en el caso de la aplicación actual es de 2 días), la sesión no se encontrará iniciada y se le solicitará que inicie sesión.

Al contrario, si el token se encuentra, y además es válido, se procede a crear un objeto *"reactivo",* conocido en *Angular* como *Observable*. Estos se caracterizan por seguir el patrón software *"Observer",* donde en caso de "subscribirnos" a uno de ellos, en el momento de que su valor cambie, nos devolverá el nuevo valor. Sin embargo, se optó por utilizar una variante, denominada *BehaviourSubject* [enlace]. Estos se caracterizan por obtener su valor en cada momento y ofrecen igualmente la opción de suscripción, con el fin de conocer posibles cambios en su valor. Su finalidad es mantener actualizado el panel de navegación, para mostrar los elementos requeridos en función del rol del usuario logueado.

Siguiendo el mismo esquema se han desarrollado el resto de los servicios, como puede ser el de perros, tomas, sesiones o usuarios, de tal forma que al igual que ocurre en los casos anteriores, se envían las correspondientes peticiones al servidor para las operaciones *CRUD (Create, read, update y delete)*, así como para la subida de archivos.

9.1.3 CONTROL DE ACCESO

Como se ha comentado anteriormente, el acceso al portal se encuentra restringido a los usuarios, es por ello por lo que en la parte del cliente hay que restringir el acceso a las distintas rutas de la web, a excepción de la página de login.

Se ha hecho uso de los *"Route Guards"* de *Angular,* que se utilizan para proteger las vistas. Creando dos variantes, por una parte un *guard* para la autorización de usuarios autenticados, donde en caso de que el usuario no se encuentre autenticado se le redirigirá a la página principal, pero en caso contrario, se le permitirá acceso a la ruta.

También existe una vista específica para administradores, por lo que se ha desarrollado un *guard* adicional para esta comprobación. Cada usuario tiene un rol especificado, por ello este *guard*, observa si el usuario es administrador o no, según el valor en este campo.

9.2 DESARROLLO DEL SERVIDOR

Por la parte de desarrollo del servidor se ha hecho uso de las librerías comentadas con anterioridad en el apartado 6. Se podría destacar la librería *Flask-Restful*, que añade múltiples funciones para el desarrollo de una *API REST* en *Flask*, como puede ser la utilización de *"Resources"*, que permiten crear controladores añadiendo únicamente los métodos *HTTP* necesarios para cada uno de ellos. A continuación se observarán los distintos controladores elaborados para cada uno de los *endpoints* implementados.

9.2.1 ENDPOINTS

El portal cuenta con una gran variedad de *endpoints* para la comunicación en ambos sentidos entre cliente y servidor, diversificando cada uno de ellos según el modelo o funcionalidad a utilizar.

En la ilustración 25, se puede observar las categorías de los *endpoints* creados, y en cada uno de los subapartados se entrará en detalle en las distintas variantes de cada uno de ellos y el motivo por el que ha sido desarrollado y su interacción con los *tokens* para verificar la autenticación de cada petición.

Ilustración 25: Categoría de los endpoints.

Por otro lado, los códigos de respuesta [21] siguen el estándar oficial, dependiendo de si se ha ejecutado con éxito o no la petición. A continuación se pueden observar los códigos de respuestas utilizados en el servidor y el motivo por el cual se hace uso de ellos.

Códigos de repuesta:

- 200 (Éxito): Se utiliza para peticiones que han sido resueltas correctamente, como puede ser la obtención de un recurso, o la actualización de este.
- 201 (Creación): Se hace uso de este código cuándo se crea un nuevo recurso en el portal.
- 400 (Petición errónea): Cuando la petición no es correcta, por ejemplo a la hora de crear un perro, no se envían los campos requeridos, se le devuelve este código junto con un mensaje mostrando los campos requeridos.
- 401 (No autorizado): Se emplea cuando no se envía la cabecera de autorización junto con el token, o este no sea válido.
- 403 (Prohibido): Usuario autenticado, pero no cuenta con los permisos necesarios para llevar a cabo la acción.
- 404 (No encontrado): Cuando el usuario solicita un recurso, pero este no existe.
- 409 (Conflicto): Utilizado cuando se intenta crear un usuario con un email existente.
- 500 (Problema en el servidor): Cuando ocurre algún problema al obtener un recurso de la base de datos, y este no puede procesarlo correctamente.

9.2.1.1 USUARIO

En primera instancia, se observan los *endpoints* de usuarios, que abarca las utilidades de autenticación de usuarios mediante la implementación de un método de login, y en caso de que sea exitosa, se procede a la creación de un *JWT*, con la información del usuario logueado. Estos *tokens*, tienen una validez predeterminada, en este caso, se ha optado por darles una validez de 2 días, puesto que el tiempo predeterminado que ofrecen es de 15 minutos, que consideramos insuficiente y que obligaría al usuario a estar autenticándose continuamente a la hora de utilizar la aplicación.

Por otra parte, se debe hacer uso de una clave privada, que nadie más debe conocer para cifrar el *token* con ella, con la finalidad de verificar la integridad de este cuando se haga una petición al servidor. Para ello se hace uso de las variables del sistema y se requiere que el sistema anfitrión cuente con la variable *"SECRET_KEY"*, con una clave compleja de adivinar para su cifrado.

También se encuentran las funcionalidades relativas al manejo de usuarios por parte del administrador. En estos *endpoints* se pueden crear, listar, eliminar o actualizar usuarios, además de contar con la posibilidad de cambiar la contraseña. Todos estos se encuentran asegurados, de forma que únicamente el administrador puede hacer uso de ellos. Posteriormente se observará como se ha implementado esta funcionalidad.

Todos los *endpoints* siguen los principios *HTTP* [22], haciendo uso de los métodos *GET* para la obtención de datos. Por otra parte, se hace uso del método *POST* para la creación de nuevo contenido en el portal, mientras que el método *PUT* se emplea ara la actualización de contenido y *DELETE* para su borrado.

9.2.1.2 PERROS

Por otro lado, se observan los *endpoints* implementados para los perros, con el fin de permitir su obtención, creación, actualización o borrado de los mismos. Para la creación de estos se hacen una serie de comprobaciones en la solicitud, verificando que los campos obligatorios se encuentran presentes y el tipado de estos. En caso de que no sean válidos, se procede a enviar el código de respuesta 400 enumerando los campos incorrectos.

También se observan algunos *endpoints* para el manejo de las fotos de perros, se permite la subida de archivos, especificando el perro destino. Además se permite su borrado y por su puesto la obtención de este.

9.2.1.2.1 SESIONES

Al igual que los perros, se han implementado *endpoints* para las operaciones *CRUD* sobre las sesiones. Sin embargo, en este caso hay que especificar un perro al cual asignar la sesión, puesto que de lo contrario, el servidor no encontrará a dicho perro, y generará un código de respuesta 404 especificando que el canino no existe

9.2.1.2.2 TOMAS

En este apartado se pueden observar algunos *endpoints* de mayor complejidad, puesto que algunos de ellos tratan los archivos de los sensores, lo que hace necesario su lectura mediante el uso de la librería *pandas*, con el fin de ofrecerle al cliente una representación de los datos necesarios para mostrar en las distintas gráficas.

9.2.2 CONTROL DE ACCESO

Todas las rutas en el servidor necesitan estar aseguradas para prevenir que usuarios no autorizados hagan uso de ciertas funcionalidades. Por ejemplo, para evitar crear perros, o incluso eliminarlos sin permiso,

Ya se ha comentado que se hace uso de tokens *JWT*. La librería utilizada *Flask-JWT-Extended*, ofrece una serie de decoradores de *Python* para la verificación de estos. En primera instancia, ofrecen el decorador *jwt-required*, que al situarlo sobre una función requiere que la petición contenga la cabecera *Authorization,* y que su contenido contenga el prefijo *Bearer* y a continuación, tras un espacio, el *token* del usuario, además de comprobar que sea válido.

Pero como se ha comentado, existen funcionalidades que los usuarios comunes no pueden realizar, como puede ser el manejo de usuarios, el cual queda reservado a los administradores. Es por ello por lo que se han creado *"claims"*. Mediante su uso, se permite añadir características extras al token, como el nombre del usuario, o en este caso su rol. Para ello, a partir del identificador que se encuentra en el *token*, se obtiene el usuario de la base de datos, y comprueba si el usuario es un administrador o no. En caso de que el usuario no se trate de un administrador, el sistema avisa con un mensaje de "No está autorizado" y el código de respuesta 403.

9.3 DESARROLLO DE LA RED NEURONAL

Este apartado se dividirá en varias secciones, explicando cada uno de los pasos que se han seguido para el desarrollo del modelo.

9.3.1 LECTURA DE LOS DATOS

Para poder entrenar la red, se requiere procesar los datos previamente de forma adecuada. Como primer paso se ha procedido a clasificar los datos en perros sanos y con cojera, representando a los primeros con un 0, mientras que los perros sanos se representan con un 1.

Se ha procedido a leer cada uno de los directorios, obteniendo la toma en cuestión y añadiendo la etiqueta correspondiente (si se trata de un perro con cojera o no), además de proceder a prepararlos como se comentaba en el apartado 7.3.3, con la eliminación de los *outliers* y la posterior división en tomas con el tamaño de secuencia máximo y el relleno con 0 en caso de ser inferior a este tamaño. Para finalizar se procede a barajar los datos leídos, con el fin de darle cierta aleatoriedad a la red. Además, si se leen los datos de forma secuencial y no se realiza este paso, ,la red solo obtendría datos de perros sanos, y no de perros con cojera.

Posteriormente, es necesario dividir los datos entre el conjunto de entrenamiento y de test. El primero se empleará para entrenar a la red de forma que pueda "aprender" cuáles son las características de los perros sanos y los perros con cojera.

Para la distribución del conjunto de datos, se utilizará el 80% de los datos para el entrenamiento, puesto que contamos con muy pocos datos, y se le debe dar prioridad al entrenamiento sobre las pruebas a realizar en la red. Por otra parte, se utilizará un tamaño de lote de 2, esto significa que por cada iteración de entrenamiento, 2 tomas serán proporcionadas a la red, con el objetivo de que pueda generalizar los datos de entrada. Por último, se procede a crear estos lotes haciendo uso de *Pytorch,* que ofrece utilidades para este fin.

En la ilustración 26 se puede observar la metodología seguida para la obtención de los datos, mientras que en la ilustración 27, se observan los métodos auxiliares elaborados para etiquetar estos datos.

```
In [86]: path = './data/'
         X = []
         Y = []
         sequence_length = 250


         for filepath in listdir(path):
             label = filepath
             for file in listdir(path + label):
                 for x in splitArray(readCsv(join(path,label,file)), sequence_length):
                     X.append(pad_features(x, sequence_length))
                     Y.append(int(label))


         perrosSanos = sum(np.array(Y) == 0)
         print('Cantidad de tomas: {}\nPerros sanos: {}\nPerros con dolencia: {}'.format(len(X), perrosSanos, len(Y)-perrosSanos))
         X, Y = shuffle(X, Y)

         Cantidad de tomas: 25
         Perros sanos: 20
         Perros con dolencia: 5
```

Ilustración 26: Obtención de las tomas en Python.

```
In [69]: def cleanOutliers(df):
             df_new = df[df["Roll"].between(df["Roll"].quantile(.1), df["Roll"].quantile(.9))]
             return df_new.reset_index()

         def readCsv(path):
             df = pd.read_csv(path, skiprows=4, sep="\t")
             return cleanOutliers(df)['Roll'].tolist()

         def readCsvRaw(path):
             df = pd.read_csv(path, skiprows=4, sep="\t")
             return np.abs(df['Roll'][200:].tolist())

         def splitArray(array, seq_length):
             data = []

             for i in range((len(array) // seq_length) + 1):
                 data.append(array[i*seq_length:(i+1)*seq_length])

             return data

         def pad_features(inputData, seq_length):
             features = np.zeros(seq_length, dtype=int)
             features[-len(inputData):] = inputData[0:seq_length]

             return features
```

Ilustración 27: Métodos auxiliares para la lectura de los datos.

9.3.2 MODELO

En esta parte, se procede a implementar el diseño comentado en el apartado 8.5.1 y que se puede observar en la ilustración 23 donde se visualiza la implementación de la red. Una vez desarrollado e implementado el modelo, hay que elaborar los pasos a seguir por la red para procesar los datos de entrenamiento y realizar las predicciones. En la ilustración 28 se observan los pasos a seguir.

```
def forward(self, state, hidden):

    batch_size = state.size(0)

    # Convertir datos a enteros
    state = state.float()

    # Obtener valores de LSTM
    output_lstm, hidden = self.lstm(state, hidden)

    # Agrupar salidas LSTM
    output_lstm = output_lstm.contiguous().view(-1, self.hidden)

    # Pasar salida por FC
    output = self.fc(output_lstm)

    # Convertir salida para obtener el último valor
    output = output.view(batch_size, -1, self.output)
    output = output[:, -1]

    output = self.sigmoid(output)

    return output, hidden
```

Ilustración 28: Feedfoward red neuronal.

En primera instancia se procede a tomar el tamaño de lote del conjunto de entrada para posteriores conversiones. A continuación se convierten los datos a número flotante, y se procede a proporcionarle estos datos a la red.

Por consiguiente, se obtiene la salida de la red, junto con los distintos estados, que representan los eventos sucedidos a lo largo de la toma, con el objetivo de que en cada instante de tiempo, se conozcan los datos ocurridos en el pasado.

Debido a que se cuentan con las salidas de las distintas celdas "*LSTM*" que componen la red, se procede a agruparlas, para posteriormente, hacer uso de

una red neuronal "*Fully-Connected*", la cual será la encargada de procesar cada una de las salidas de las celdas, con el objetivo de dar el diagnóstico final de la red. Además, únicamente resulta relevante obtener la salida de la última celda, puesto que esta es la encargada de diagnosticar si el perro cuenta con cojera o no.

Por último, la salida de la red puede contener cualquier valor numérico, lo que dificultad identificar la categoría identificada por la red. Por ello se ha utilizado la función sigmoide, que se encarga de normalizar la entrada a un valor entre 0 y 1, siendo 0 un perro sano, y 1 un perro con cojera.

9.3.2.1 HIPERPARÁMETROS

Otro de los aspectos claves a destacar del modelo son los hiperparámetros, cuya finalidad sirve para mejorar ciertos aspectos del aprendizaje de la red. Estos pueden ser determinantes de cara al resultado final de la red, ya que unos valores no optimizados pueden llevar a altas tasas de fallos y una precisión algo baja.

Para la optimización de estos no existe un protocolo o fórmula que pueda determinarlos, ya que hay muchos factores que alteran el comportamiento de la red.

Cuando se entrena una red, existe lo que se denominan pesos, estos determinan la importancia que se le da a cada una de las entradas de la red y son inicializados en valores aleatorios. Durante el entrenamiento, la red ofrece un resultado sobre los datos obtenidos, en este caso determina si el perro se encuentra sano o no.

Posteriormente se procede a observar la salida dada con la esperada, y se calcula que tan distante ha sido. A este valor se le denomina perdida o *loss*. Por consiguiente se procede a actualizar cada uno de los pesos según esta "distancia".

Es preciso señalar el *learning rate*, este se encarga de establecer la frecuencia de actualización de los pesos. Hay que resaltar que valores altos ocasionan grandes cambios en los pesos, mientras que si el valor es muy pequeño se producirán pocas variaciones y el aprendizaje será más lento.

Aunque tenga sentido darle un valor muy alto, esto puede ocasionar que los pesos se sitúen lejos de los mínimos esperados, dificultando en consecuencia el entrenamiento. Sin embargo, el objetivo es configurar unos valores mínimos en los pesos, para así tener una pérdida prácticamente nula.

Por otra parte, si situamos un valor muy pequeño, la actualización será muy lenta y puede llevar mucho tiempo. También existe la posibilidad de que los valores se sitúen lejos de los mínimos, ya que pueden caer en mínimos locales, ocasionando que la red no aprenda. En la ilustración 29 se puede observar una mejor representación de este valor.

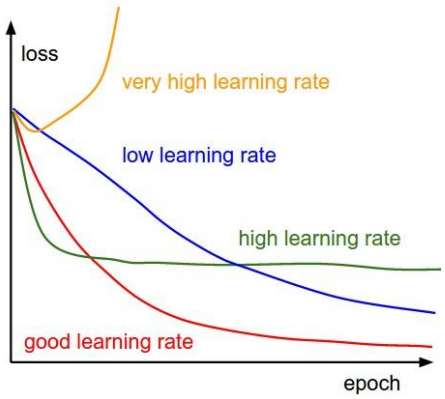

Ilustración 29: Diferencias learning rate.

En el caso del proyecto, se ha utilizado uno de los valores estándar para ello (0.1).

Otro de los hiperparámetros conocidos tiene que ver con el número de épocas. Se encuentra relacionado con el entrenamiento, y puede definirse como el número de iteraciones que la red será entrenada con el conjunto de datos. Cada vez que se le suministra a la red el conjunto de datos, sucede una época. Por lo tanto, si establecemos 25 épocas, se le suministran a la red 25 veces el

conjunto de entrenamiento, con el objetivo de que sea capaz de generalizar estos datos y aprender de ellos.

Con este parámetro se encuentra ligado otro parámetro que es el tamaño de lote, un parámetro que como se comentaba, establece el número de datos de entrenamiento a proporcionar a la red al mismo tiempo. Cabe destacar que si no se hiciese uso de este parámetro, se estaría utilizando una mayor cantidad de memoria, al tener cargado en memoria la totalidad de datos. Para este parámetro se suele utilizar una potencia de 2, debido a que las memorias también tienen valores con esta misma potencia.

Al igual que el *learning rate*, no existe una fórmula para establecer el número de épocas. Es necesario ir probando distintos valores e ir observando el rendimiento de la red, ya que pueden darse varios casos.

Por una parte, si se establece un número de épocas muy pequeño, la red no puede ser capaz de generalizar los datos de entrada, ocasionando que no aprenda y dando lugar a predicciones erróneas y ocasionando *"Underfitting"*.

Por otra parte, si se establece un valor muy grande, en primer lugar el entrenamiento puede demorarse mucho tiempo, sobre todo en conjuntos de datos muy amplios. Sin embargo, el mayor inconveniente es todo lo contrario al "*Underfitting*". La red terminará memorizando el conjunto de datos de entrada, y aunque se observe durante el entrenamiento una precisión y errores excelentes, a la hora de ponerla en producción, cuando se intente clasificar datos nunca vistos, dará lugar a predicciones erróneas. A este término se le conoce como "*Overfitting*" o sobreajuste.

En la ilustración 30 se puede observar una diferencia entre los términos anteriormente comentados. La línea azul representa el ajuste de parámetros a los datos de entrada, siendo el gráfico central el ideal.

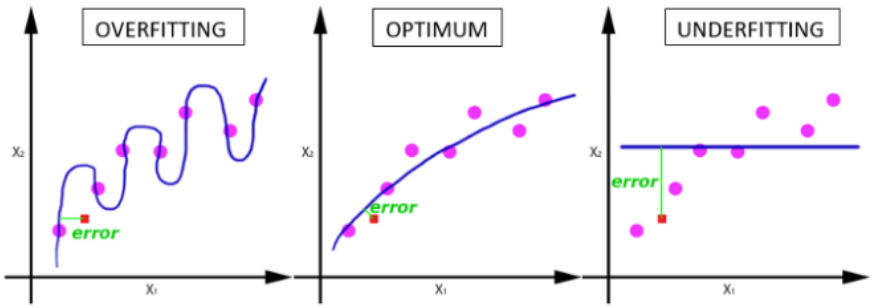

Ilustración 30: Diferencias learning rate.

Cabe destacar que el número de épocas no va a causar por si solo uno de los problemas anteriormente comentados. Existen múltiples parámetros involucrados en ello, además de múltiples técnicas para optimizar la red. En el caso de este proyecto, puesto que se cuenta con un conjunto de datos algo escaso, se ha optado por utilizar un número alto de épocas, 25, para intentar que la red "aprenda" la mayor cantidad de patrones posibles.

9.3.3 ENTRENAMIENTO

Respecto al entrenamiento, en primer lugar comentaremos las funciones de optimización y perdidas utilizadas, explicando el rol de cada una de ellas. A continuación, se observará el procedimiento seguido para entrenar a la red, y el posterior testeo de la aplicación observando su porcentaje de acierto con datos nunca vistos.

9.3.3.1 FUNCIÓN DE COSTE

La función de coste se encarga de comparar la salida de la red y lo valores esperados que esta debería obtener. Es un buen indicativo de cuanto de bien lo está haciendo la red con el conjunto de entrenamiento, puesto que si obtenemos un valor cercano a 0 significa que la red apenas está cometiendo errores durante el entrenamiento.

Sin embargo, puede darse el caso de que durante el entrenamiento se obtenga una perdida bastante baja, y una vez se le proporcionan datos no antes vistos, la precisión no sea la esperada, ocasionando *Overfitting*.

Existen múltiples funciones de coste, cada una de ellas mide el error de una manera distinta. Dependiendo del problema a tratar, habrá que seleccionar la más indicada para ello, puesto que esta será la encargada de decidir cuanto de bien lo está haciendo la red con el conjunto de entrenamiento.

Se ha hecho uso se la función de error cuadrático medio, que sigue la formula descrita en la ilustración 31. Esta función calcula la suma cuadrática de la diferencia entre el resultado real y el generado por la red en términos medios.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Ilustración 31: Fórmula error cuadrático medio.

9.3.3.2 FUNCIÓN DE OPTIMIZACIÓN

Una vez obtenido el error, hay que ajustar los pesos de la red para que se adapten lo mejor posible al conjunto de datos, ya que así la red producirá mejores predicciones. Para ello habrá que seleccionar una función que actualice los pesos, para lo cual se aplica el algoritmo de descenso por el gradiente [23]. A través de diversas operaciones matemáticas como derivadas es capaz de ir optimizando los parámetros de la red.

Se trata de un proceso iterativo que se ejecuta tras suministrar un conjunto de datos a la red. En esta función el *learning rate* juega un papel muy importante, puesto que es el encargado de decidir cuanta importancia darle a la actualización de los pesos: un valor muy alto causa que varíen bastante, mientras que un valor muy pequeño causa que apenas varíen respecto a su valor anterior, con las ventajas y desventajas comentadas en el apartado 9.3.2.1.

Este método tradicional no es el ideal para redes tan complejas, ya que no cuenta con ningún de algoritmo adicional que permita acelerar el entrenamiento e incluso mejorarlo. Al igual que la función de coste, existen múltiples algoritmos para optimizar esta etapa. En este caso se ha optado por utilizar el algoritmo *Adam* [24]. Este se trata de uno los algoritmos de optimización más utilizados por sus ventajas, como por su gran eficiencia o su reducido consumo de recursos.

Este se compone de las ventajas de otros dos algoritmos como son:

- AdaGrad [25]: Se encarga de adaptar el *learning rate* a los distintos parámetros, produciendo actualizaciones pequeñas a parámetros muy utilizados, y actualizaciones más grandes en parámetros poco utilizados.

- RMSProp: También se encarga de adaptar el *learning rate*, pero únicamente se fija en las actualizaciones más recientes, mientras que *AdaGrad* contiene un acumulado de todas las actualizaciones.

9.3.3.3 ENTRENAMIENTO

Tras haber descrito los distintos componentes que forman parte de la red en su conjunto, se procede a entrenar la red. En este caso se ha optado por mostrar únicamente la perdida proporcionada por la red, y observar la precisión en la etapa de test, para dar así una tasa de precisión lo más cercana a la realidad. En la ilustración 32 se puede observar los pasos que se han seguido para el entrenamiento de la red, así como un desglose del coste durante su entrenamiento.

```
counter = 0
print_every = 25

model.train()

for e in range(epochs):
    h = model.init_hidden(batch_size)
    for inputs, labels in train_loader:
        counter += 1

        if(train_on_gpu):
            inputs, labels = inputs.cuda(), labels.cuda()

        h = model.init_hidden(inputs.size(0))
        model.zero_grad()
        output, h = model(inputs.view(inputs.size(0), -1, 1), h)

        loss = criterion(output.squeeze(), labels.float())
        loss.backward()

        optimizer.step()

        if counter % print_every == 0:
            print("Epoch: {}/{}...".format(e+1, epochs),
                  "Step: {}...".format(counter),
                  "Loss: {:.6f}...".format(loss.item()))
```

```
Epoch: 4/25... Step: 25... Loss: 0.000043...
Epoch: 7/25... Step: 50... Loss: 0.013998...
Epoch: 10/25... Step: 75... Loss: 0.170345...
Epoch: 13/25... Step: 100... Loss: 0.002957...
Epoch: 16/25... Step: 125... Loss: 0.014182...
Epoch: 19/25... Step: 150... Loss: 0.000574...
Epoch: 22/25... Step: 175... Loss: 0.025430...
Epoch: 25/25... Step: 200... Loss: 0.078772...
```

Ilustración 32: Entrenamiento de la red.

En primera instancia se procede a establecer la red en modo entrenamiento, para permitir la actualización de sus pesos. Tras ello se procede a obtener los datos de entrenamiento en lotes, y se hace uso de la tarjeta gráfica en caso de contar con una, para así acelerar el entrenamiento.

A continuación, se procede a suministrar los datos de entrenamiento a la red para observar las predicciones generadas por la red, y su posterior comprobación con el resultado real.

Se finaliza actualizando los pesos con el optimizador. Si se observan los datos ofrecidos en la ilustración 32, se puede apreciar que se obtiene un coste bastante bajo, en torno a 0, aunque a veces es algo superior debido a algún lote de datos que a la red le cuesta clasificar correctamente. Como se comentaba con anterioridad, puede resultar un buen indicativo, ya que ha conseguido identificar una gran cantidad de datos correctamente, pero habrá que observar la precisión con datos no antes vistos.

10. PRUEBAS

En relación con las pruebas, en primera instancia se evaluarán los distintos *endpoints* de la aplicación web. Para ello se ha elaborado una batería de tests automáticos.

Por otro lado, se verificará que la red neuronal es capaz de ofrecer diagnósticos fiables, empleando para ello varias tomas realizadas que la red neuronal no ha conocido previamente.

10.1 APLICACIÓN WEB

Para las pruebas en la aplicación web se ha hecho uso de *Postman*, la cual permite simular peticiones a la dirección deseada mediante cualquier método *HTTP*, así como realizar una batería de pruebas unitarias que se ejecutarán de forma secuencial una tras otra.

Estas pruebas nos permite añadir ciertas cabeceras, además de tener la posibilidad de programar el contenido que será enviado en cada petición, con el objetivo de evitar duplicidades en algunos campos, como el correo electrónico al crear un nuevo usuario.

Una de las primeras pruebas realizadas ha sido la de verificar la seguridad de la totalidad de rutas en el servidor, con el fin de que ningún usuario pueda realizar acciones no autorizadas. Para ello se verificó que cada una de las peticiones recibía un código de respuesta 401, que especifica “No autorizado”.

A continuación se realizó pruebas sobre las distintas entidades representadas en la web, como perros, sesiones o tomas, verificando que un usuario autenticado puede crear, actualizar, visualizar y eliminar cada una de ellas. Además, se validó que un usuario con rol "administrador" tenía la posibilidad de hacerlo sobre los usuarios. De lo contrario el servidor nos brinda un código de respuesta 403 o "Prohibido".

Por último, se verificó la correcta subida de archivos al servidor, con varios tamaños de archivo para comprobar que archivos de gran tamaño no pudieran ser enviados, y su posterior visualización.

10.2 RED NEURONAL

En apartados anteriores, se visualizó que la red contaba con un coste bastante cercano a 0 en el conjunto de datos de entrenamiento. Sin embargo, como ya hemos comentado, esto no indica la fiabilidad de la red, puesto que puede haber "memorizado" estos datos, y al observar datos nunca vistos puede ofrecer conclusiones no fiables, al no haber ha sido capaz de generalizar los datos de entrada, dando lugar a predicciones erróneas.

Es por ello, que se suele hacer uso de un conjunto de datos adicional al conjunto de entrenamiento. Por este motivo se ha creado un pequeño conjunto de tomas para las pruebas. Este conjunto es un tipo de datos no conocidos por la red, pero de la misma distribución que el resto. Nos permite conocer que tan bien la red ha generalizado los datos y que tan fiable es.

Este conjunto suele representar un pequeño porcentaje del total, en este caso se ha optado por un 20%, puesto que al contar con tan pocas muestras, se le ha dado prioridad al entrenamiento, con el objetivo de que la red pueda aprender patrones de estos de una mejor manera. También habrá que tener en cuenta que se tendrá que desactivar la regularización empleada, siendo en este caso *Dropout*, puesto que la red ya se encuentra entrenada. Por lo tanto, se activa la red en modo evaluación. En la ilustración 33 se puede observar el procedimiento llevado a cabo para testar la red con este conjunto de datos.

```
test_losses = []
num_correct = 0

h = model.init_hidden(batch_size)

model.eval()
for inputs, labels in test_loader:
    h = model.init_hidden(inputs.size(0))

    if(train_on_gpu):
        inputs, labels = inputs.cuda(), labels.cuda()

    output, h = model(inputs.view(inputs.size(0), -1, 1), h)


    test_loss = criterion(output.squeeze(), labels.float())
    test_losses.append(test_loss.item())

    pred = torch.round(output.squeeze())

    correct_tensor = pred.eq(labels.float().view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    num_correct += np.sum(correct)


print("Test loss: {:.3f}".format(np.mean(test_losses)))
test_acc = num_correct/len(test_loader.dataset)
print("Test accuracy: {:.3f}".format(test_acc))
```

```
Test loss: 0.368
Test accuracy: 0.600
```

Ilustración 33: Prueba de la red.

Se puede observar en la ilustración 33 que la pérdida en el conjunto de test es ampliamente superior al de entrenamiento, siendo en este caso bastante superior al obtenido en el conjunto de entrenamiento.

Por otra parte, se puede observar que la precisión ha sido del 60%, una precisión algo escasa para un problema de clasificación binaria, puesto existe un 50% de posibilidades de que acierte la predicción al tener dos clases únicamente, pero al contar con tan pocas muestras, es difícil contar con una precisión superior.

10.2.1 MEJORAS

Por lo tanto, se puede afirmar que la red cuenta con sobreajuste o *Overfitting*. Para solventar este problema se pueden emplear diversas técnicas, entre las cuales destacamos las siguientes:

- Aumentar la cantidad de datos.
- *Early Stopping.*
- Añadir regularización.

Dado que estamos limitados en la cantidad de datos por la dificultad que conlleva la obtención de estos, este no ha sido el camino escogido para buscar mejoras en los resultados, Además, la generación de conjuntos de datos de forma artificial es compleja en nuestro caso, ,ya que contamos con datos secuenciales difíciles de replicar.

Sucede algo similar con la segunda estrategia, ya que la parada temprana requiere de la elaboración de un nuevo conjunto de datos conocido como conjunto de validación, que se utiliza junto con el conjunto de entrenamiento. Cuando se observa que la precisión de la red mejora con los datos de validación, se procede a almacenar el modelo, dejando así el modelo con la mejor precisión. En la ilustración 34 se puede observar una gráfica donde se observa el funcionamiento de este conjunto.

Ilustración 34: Representación Early stopping.

Pero ante la poca cantidad de datos, el conjunto de validación no sería superior a 4 tomas, una cantidad algo escasa para llevarlo a cabo.

Y por último tenemos la estrategia consistente en hacer uso de regularización, que consiste en añadir capas como la comentada *Dropout* o en este caso, hacer uso de *Gradiente clipping* [26]. Esta técnica se utiliza en redes tipo *LSTM*. Consiste en evitar que al realizar el descenso por el gradiente, los valores sean excesivamente grandes o pequeños llevando a actualizaciones erróneas de los distintos pesos. Para ello trata de establecer el vector obtenido del descenso por el gradiente a un valor mínimo o máximo, siempre y cuando se exceda un cierto rango. Por este motivo, hay que aplicarlo durante la fase de entrenamiento al hallar la función de coste.

Otro de los cambios realizados ha sido aplicar más *Dropout,* estableciendo un 60%. Por último, se ha establecido un tamaño de lote de 3 tomas. En la ilustración 35 se puede observar el añadido de esta regularización, así como la nueva pérdida obtenida durante el entrenamiento.

```
counter = 0
print_every = 25
clip=3

model.train()

for e in range(epochs):
    h = model.init_hidden(batch_size)
    for inputs, labels in train_loader:
        counter += 1

        if(train_on_gpu):
            inputs, labels = inputs.cuda(), labels.cuda()

        h = model.init_hidden(inputs.size(0))
        model.zero_grad()
        output, h = model(inputs.view(inputs.size(0), -1, 1), h)

        loss = criterion(output.squeeze(), labels.float())
        loss.backward()

        nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        if counter % print_every == 0:
            print("Epoch: {}/{}...".format(e+1, epochs),
                  "Step: {}...".format(counter),
                  "Loss: {:.6f}...".format(loss.item()))
```

```
Epoch: 4/25... Step: 25... Loss: 0.400359...
Epoch: 8/25... Step: 50... Loss: 0.307301...
Epoch: 11/25... Step: 75... Loss: 0.006455...
Epoch: 15/25... Step: 100... Loss: 0.023495...
Epoch: 18/25... Step: 125... Loss: 0.031325...
Epoch: 22/25... Step: 150... Loss: 0.537307...
Epoch: 25/25... Step: 175... Loss: 0.014279...
```

Ilustración 35: Añadida regularización al entrenamiento.

Por la parte de la perdida, es bastante similar al obtenido anteriormente, aunque si se observa la ilustración 36, la precisión con el conjunto de test es ampliamente superior al anterior, situándose en un 85.7%, mejorando así las capacidades del modelo.

```
test_losses = []
num_correct = 0

h = model.init_hidden(batch_size)

model.eval()
for inputs, labels in test_loader:
    h = model.init_hidden(inputs.size(0))

    if(train_on_gpu):
        inputs, labels = inputs.cuda(), labels.cuda()

    output, h = model(inputs.view(inputs.size(0), -1, 1), h)


    test_loss = criterion(output.squeeze(), labels.float())
    test_losses.append(test_loss.item())

    pred = torch.round(output.squeeze())

    correct_tensor = pred.eq(labels.float().view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    num_correct += np.sum(correct)


print("Test loss: {:.3f}".format(np.mean(test_losses)))
test_acc = num_correct/len(test_loader.dataset)
print("Test accuracy: {:.3f}".format(test_acc))
```

```
Test loss: 0.110
Test accuracy: 0.857
```

Ilustración 36: Precisión del modelo tras añadir regularización.

11. CONCLUSIONES Y TRABAJO FUTURO

11.1 CONCLUSIONES

Tras la finalización del proyecto, se puede concluir que todos los objetivos propuestos al comienzo de este han sido cumplidos.

En primera instancia, se ha implementado una red neuronal con una precisión cercana al 90% para la detección de cojera en perros, aunque teniendo un conjunto de datos algo escaso, sería muy prematuro ponerla en explotación en un entorno real.

Por otra parte, se ha diseñado y desarrollado una aplicación web que permita llevar el registro de perros creando distintos perfiles para cada uno, con información básica sobre cada uno de ellos. Se permite crear varias sesiones, lo que permite ver la evolución del canino cada vez que se hacen ajustes en su tratamiento. Del mismo modo se permite la creación de tomas para así ver la efectividad de un cierto tratamiento, y la posible mejoría de una de sus extremidades. De esta manera, se permite a un veterinario la gestión de una gran cantidad de perros en una única plataforma y con varias funcionalidades que no pueden adquirirse en el mercado.

Por otra parte, a nivel personal, al haber sido un proyecto en el que no había ningún trabajo previo, y en un ámbito bastante interesante y desconocido hasta el momento, me ha permitido seguir aprendiendo distintas tecnologías punteras hoy en día como *Pytorch, Matplotlib o Pandas*. Así como, *Angular y Flask* para la elaboración de la aplicación web.

11.2 TRABAJO FUTURO

Pese a haber cumplido los objetivos iniciales y las distintas funcionalidades hayan sido implementada, este proyecto puede ser mejorado.

Por una parte, se podría desarrollar un programa encargado de la recogida y tratamiento de los sensores, evitando así utilizar el software proporcionado por la empresa propietaria de los sensores, con el objetivo de que el usuario no requiera utilizar varios programas para poder llevar a cabo la grabación de tomas, así como registrar tanto la toma de datos como el video en tiempo real, además de obtener el diagnóstico de la red neuronal al instante, evitando así tener que generar los archivos correspondientes, y posteriormente subirlos a la plataforma para obtener el diagnóstico.

Por otra parte, la red ha sido entrenada con un conjunto de datos algo escaso, aunque se haya obtenido una muy buena precisión, una mayor cantidad de datos resultaría beneficiosa para que pueda llevar a cabo mejores predicciones.

También se podría comparar los resultados obtenidos empleando otro tipo de red neuronal, empleando por ejemplo directamente el vídeo en lugar de sensores para la detección de la cojera, permitiendo así utilizar incluso el teléfono móvil para ofrecer diagnósticos en tiempo real.

REFERENCIAS

[1] S. López, J. M. Vilar, M. Rubio, J. Sopena, E. Damiá, D. Chicharro, Á. Santana y J. M. Carrillo, «Center of pressure limb path differences for the detection of lameness in dogs: a preliminary study.,» *BMC Vet Res,* 2019.

[2] K. M. Robinson, «webmd,» [En línea]. Available: https://www.webmd.com/melanoma-skin-cancer/features/ai-skin-cancer#1. [Último acceso: Abril 2020].

[3] S. Romiti, M. Vinciguerra, W. Saade, I. Anso Cortajarena y E. Greco, «Hindawi,» [En línea]. Available: https://www.hindawi.com/journals/crp/2020/4972346/. [Último acceso: 9 Noviembre 2020].

[4] Equinosis, «Equinosis,» [En línea]. Available: https://equinosis.com/. [Último acceso: 2020].

[5] P. G. A. B. M. R. C.B. Gómez Álvarez, «Inertial sensor-based system for lameness detection in trotting dogs,» 2017.

[6] Imagenet, «Imagenet,» [En línea]. Available: http://www.image-net.org/challenges/LSVRC/. [Último acceso: Octubre 2020].

[7] C. Guerra Artal, «Github,» [En línea]. Available: https://nbviewer.jupyter.org/url/cayetanoguerra.github.io/ia/nbpy/redneuronal1.ipynb. [Último acceso: 2020].

[8] S. Swaminathan, «Towards Data Science,» [En línea]. Available: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc?gi=3abc251a1b8e. [Último acceso: 2020].

[9] «Tecnología Informática,» [En línea]. Available: https://www.tecnologia-informatica.com/tipos-licencias-software-libre-comercial/. [Último acceso: 2020].

[10] «GNU,» [En línea]. Available: https://www.gnu.org/licenses/licenses.es.html. [Último acceso: 2020].

[11] Python, «Python,» [En línea]. Available: https://docs.python.org/3/license.html. [Último acceso: 2020].

[12] «Desde Linux,» [En línea]. Available: https://blog.desdelinux.net/hablemos-de-la-licencia-bsd/. [Último acceso: 2020].

[13] «JWT,» [En línea]. Available: https://jwt.io/introduction/. [Último acceso: 2020].

[14] D. Coimbra de Andrade, «Towards Data Science,» [En línea]. Available: https://towardsdatascience.com/recognizing-speech-commands-using-recurrent-neural-networks-with-attention-c2b2ba17c837. [Último acceso: 2020].

[15] P. Liu, X. Qiu y X. Huang, «Recurrent Neural Network for Text,» 2016.

[16] S. Team, «SuperDataScience,» 23 Agosto 2018. [En línea]. Available: https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem. [Último acceso: 2020].

[17] Colah, «Github,» 27 Agosto 2015. [En línea]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Último acceso: 2020].

[18] R. Staudemeyer y E. Rothstein Morris, «Recurrent Neural Network for Text,» 2017.

[19] J. Brownlee , «Machine Learning Mastery,» 26 Junio 2017. [En línea]. Available: https://machinelearningmastery.com/handle-long-sequences-long-short-term-memory-recurrent-neural-networks/. [Último acceso: 2020].

[20] J. Brownlee, «Machine Learning Mastery,» 3 Diciembre 2018. [En línea]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Último acceso: 2020].

[21] «Developer Mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Status. [Último acceso: 2020].

[22] Gelito, «Kubide,» [En línea]. Available: https://kubide.es/principios-de-un-webservice-restful/. [Último acceso: 2020].

[23] C. Santana, «Youtube,» 4 Febrero 2018. [En línea]. Available: https://kubide.es/principios-de-un-webservice-restful/. [Último acceso: 2020].

[24] J. Brownlee , «Machine Learning Mastery,» 3 Julio 2017. [En línea]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Último acceso: 2020].

[25] S. Ruder, «Ruder,» 19 Enero 2016. [En línea]. Available: https://ruder.io/optimizing-gradient-descent/index.html#adagrad. [Último acceso: 2020].

[26] J. Brownlee, «Machine Learning Mastery,» 6 Febrero 2019. [En línea]. Available: https://machinelearningmastery.com/how-to-avoid-exploding-gradients-in-neural-networks-with-gradient-clipping/. [Último acceso: 2020].

MANUAL DE USUARIO

A continuación se describen las funcionalidades implementadas en la aplicación web.

INICIO DE SESIÓN

Para acceder a la herramienta, es necesario contar con una cuenta, no existe la posibilidad de registrarse manualmente, un administrador del sitio se encarga de esta labor. La autenticación se realiza a partir del formulario mostrado en la ilustración 37.

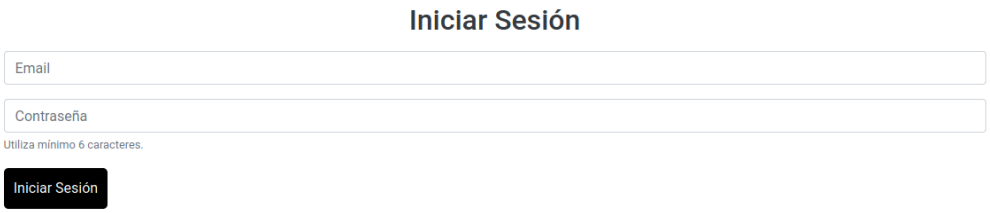

Ilustración 37: Página de acceso a la aplicación.

Una se tiene acceso, se muestra una vista general de los caninos creados en el portal, con información descriptiva de cada uno de ellos, como su nombre y una imagen representativa. Además, se cuentan con un filtro para la búsqueda por nombre, y con opciones de ordenación por su nombre o fecha de actualización (ilustración 38).

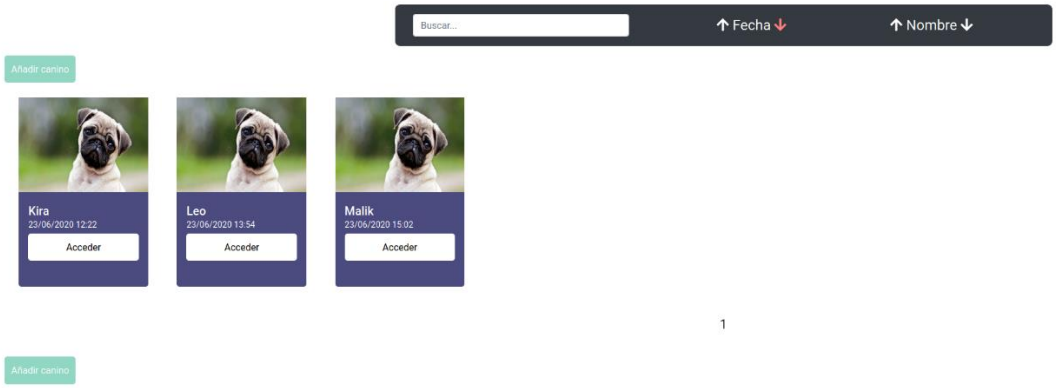

Ilustración 38: Página principal "Vista de perros".

GESTIÓN DE USUARIOS

Si el usuario autenticado se trata de un administrador del sitio, en la parte superior, se muestra la opción de acceder a la gestión de usuarios del portal, a partir de esta pestaña, el administrador tendrá la opción de gestionar los usuarios del portal (Ilustración 39) pudiendo buscar usuarios por nombre o email.

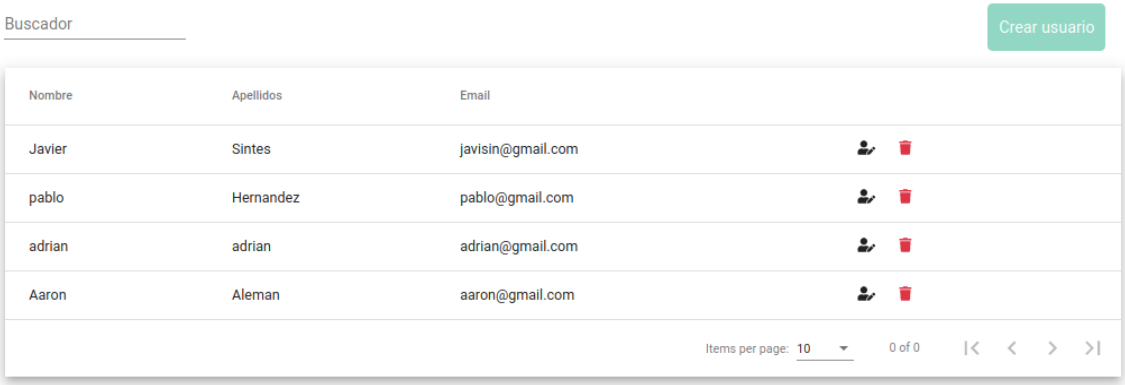

Ilustración 39: Gestión de usuarios.

A partir del botón de “Crear usuario”, se accede a la vista de un formulario donde el administrador pueda crear un nuevo usuario, para ello le asignará un nombre, email, rol y contraseña (ilustración 40).

Ilustración 40: Creación de usuarios.

Así mismo, también se permitirá la edición de estos, para editar un usuario concreto, se deberá acceder mediante la tabla de usuario descrita en la ilustración 39, donde se mostrarán los datos del usuario en cuestión, permitiendo cambiar cualquiera de sus datos, así como actualizar su contraseña (ilustración 41).

Ilustración 41: Actualización de usuarios.

Si se desea eliminar un usuario, se podrá realizar a través de la tabla descrita en la ilustración 39, con la finalidad de evitar borrados accidentales, se deberá confirmar su borrado a través de una ventana emergente (ilustración 42).

Ilustración 42: Confirmación de borrado de usuarios.

VISTA DE PERROS

Como se muestra en la ilustración 38, se pueden ver los perros creados en el portal, así como buscarlos según su nombre u ordenarlos según el mismo nombre o la fecha de actualización de este. Si se desea crear un nuevo perro, haciendo clic sobre el botón "Crear perro" mostrado en la ilustración 38.

En esta nueva vista, se permite crear un nuevo perro en el portal, para ello habrá que añadir su nombre, raza, fecha de nacimiento, género, así como su peso y altura. También se permite añadir una foto, en caso de no desearlo se muestra una imagen por defecto (ilustración 43).

Ilustración 43: Creación de perro.

Al crear al perro, se puede acceder a su perfil, donde se muestran los distintos datos establecidos, así como las distintas sesiones del perro, pudiendo acceder a cada una de ellas, como crear nuevas (ilustración 44).

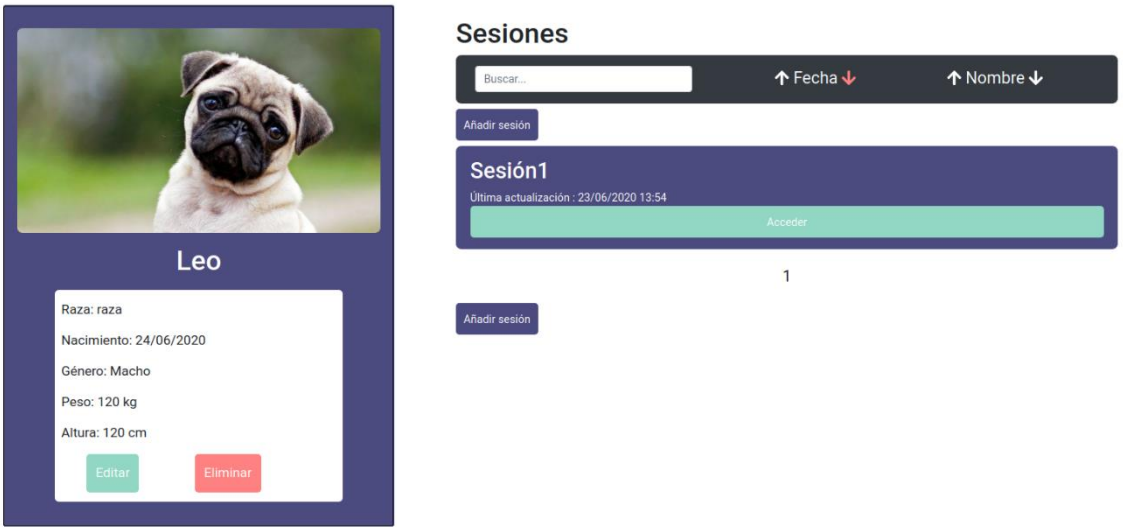

Ilustración 44: Vista detallada de los perros.

Del mismo modo que el borrado de usuarios, si se desea eliminar el perro, para evitar borrados accidentales, se muestra una ventana emergente como la vista en la ilustración 42. Si se desea editar algún dado de un perro, se puede acceder haciendo clic sobre el botón "editar" mostrado en la ilustración 44.

El formulario de edición es idéntico al mostrado en la ilustración 43 pero en este caso se editará el perro, y no se creará uno nuevo. En la vista descrita en la ilustración 44 se permite la opción de crear sesiones, al hacer clic en ella, nos muestra una ventana emergente para introducir el nombre de la sesión a crear (ilustración 45).

Ilustración 45: Creación de sesión.

GESTIÓN DE SESIONES

Al acceder a una de las sesiones, nos muestran las distintas tomas que componen dicha sesión, pudiendo buscarlas por nombre, así como ordenarlas según distintos parámetros (ilustración 46).

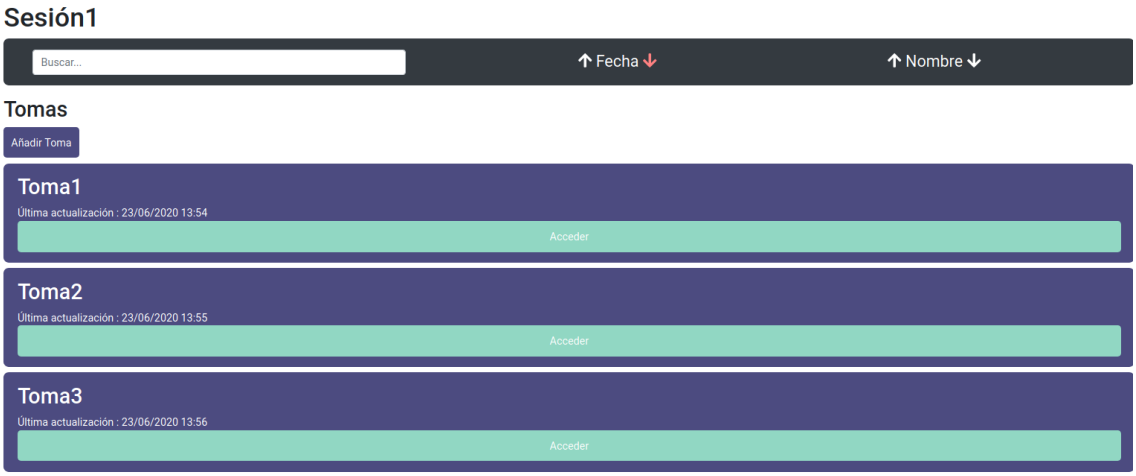

Ilustración 46: Vista de sesiones.

Si se hace clic en el botón "Añadir toma", se muestra una ventana emergente como la vista en la ilustración 45, aunque en este caso, al crear la toma, se muestra una nueva vista, donde se pueden subir los archivos de la toma, como los distintos videos y datos de los sensores (ilustración 47).

Ilustración 47: Gestión de una toma.

VISTA DE TOMAS

Al finalizar la creación de una toma, o accediendo a través de la vista de sesiones, se puede acceder a la vista de una toma en concreto, en esta se ofrece la posibilidad de editarla, así como las distintas opciones para reproducir la toma o mostrar alguna gráfica en concreto (ilustración 48).

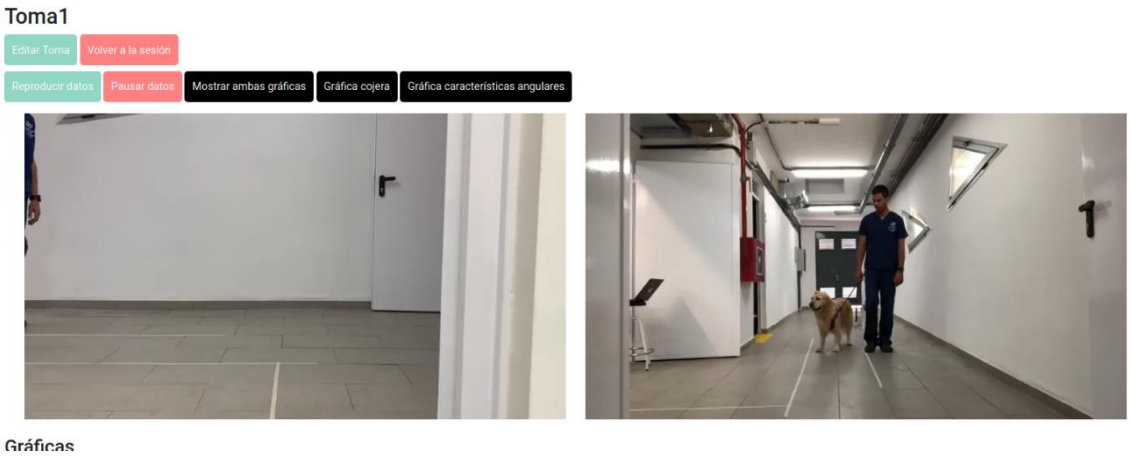

Ilustración 48: Vista de una toma.

En la parte inferior de los videos, se muestran las gráficas en caso de activarlas, por una parte se muestra una gráfica que ofrece los datos relativos a la cojera en perros, mientras que la otra gráfica muestra los ángulos relativos obtenidos de los sensores (ilustración 49).

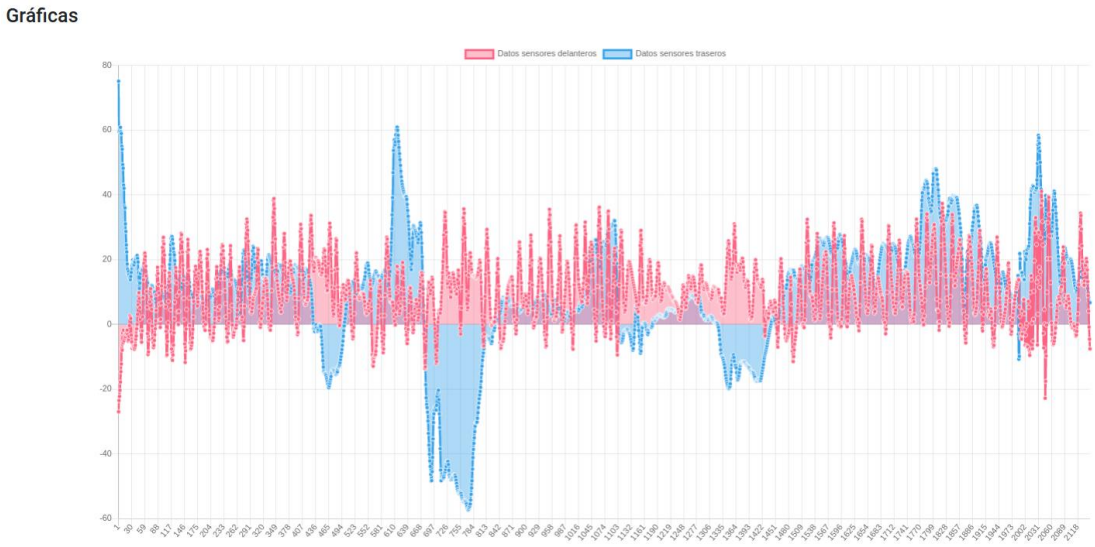

Ilustración 49: Vista gráfica de los datos de los sensores.

Por último lugar, en la parte inferior de esta vista, existen dos observaciones, por una parte una observación general que le permitirá al usuario dar su diagnóstico subjetivo sobre la toma, mientras que por la otra parte, se muestra el diagnostico ofrecido por la red (ilustración 50), este diagnóstico se realiza al subir los archivos de los sensores en el formulario de edición o creación de una toma (ilustración 47).

Ilustración 50: Observación de la toma.