Proyecto Fin de Grado
Ingeniería electrónica, robótica y mecatrónica

Sistema bio-inspirado para la estimación de velocidades de rotación

Autor: Alfonso Garijo Campos

Tutor: Fernando Caballero Benítez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Ingeniería de Sistemas y Automática

Sistema bio-inspirado para la estimación de velocidades de rotación

Autor:

Alfonso Garijo Campos

Tutor:

Fernando Caballero Benítez

Profesor Titular de Universidad

Jesús Capitán Fernández

Profesor Contratado Doctor

Dpto. Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado: Sistema bio-inspirado para la estimación de velocidades de rotación

Autor: Alfonso Garijo Campos

Tutor: Fernando Caballero Benítez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El secretario del Tribunal

A mi familia

A mis maestros

Desde el primer momento en el que entré en la universidad he visto como uno tras otro, grandísimos profesionales de cada área de la ingeniería han aportado su granito de arena para poner a mi disposición una enorme cantidad de conocimiento. Y sin embargo no es ese conocimiento lo más valioso que he adquirido durante estos cuatro años. También me han enseñado a desmenuzar cada problema, a ver más allá de lo evidente, a encontrar la lógica detrás de cada concepto, teoría o tecnología, es decir, a enfrentarme a cualquier desafío tecnológico que encuentre en mi vida laboral.

Más allá de toda enseñanza, he visto como esos profesionales formados y en muchos casos brillantes, han hecho grandes esfuerzos para transmitir su pasión por la ingeniería en cada una de sus ramas, tanto a nivel profesional como personal. Es por todo esto que solo puedo agradecer ese afán por investigar, por difundir conocimiento, por inspirar e instruir a los ingenieros de las generaciones que van emergiendo a lo largo de los años. Agradezco enormemente a todo el profesorado de la Escuela Técnica Superior de ingeniería toda su labor docente.

Sin embargo, tampoco puedo olvidar a todos esos maestros que me educaron y guiaron entre los 3 y los 18. Sin ellos sería una persona radicalmente diferente, no podría haber llegado hasta aquí sin su paciencia, cariño y motivación.

Y por último debo agradecer a mi familia toda la ayuda que me ha prestado. Todo el amor, el apoyo, la felicidad, la educación y por supuesto la inversión tanto temporal como económica que han hecho para llevarme hasta aquí. Por enseñarme a hacer todo lo posible por estar orgulloso de mi trabajo y tener la confianza y la fuerza para superar cada obstáculo en mi camino.

Y no solo a mi familia, también a todos aquellos amigos y parejas que, hasta el día de hoy, no solo han hecho mas feliz mi vida, sino que me han enseñado a vivirla a todos los niveles y han estado dispuestos a ayudarme siempre que lo he necesitado.

RESUMEN

Actualmente el sensor más utilizado para conocer la inclinación y orientación de un robot o cuerpo de cualquier índole es una IMU, una combinación de acelerómetros, giróscopos y magnetómetros. Sin embargo, tiene un importante defecto y es que periódicamente es necesario calibrarlo debido a que el error que afecta a su medida es de carácter acumulativo.

A partir de una serie de estudios en el campo de biología sobre los órganos sensoriales de los insectos, se ha descubierto un mecanismo de percepción de la orientación angular en el espacio sensible de ser reproducido de forma artificial. Dicho mecanismo de percepción consiste en la traducción de intensidades lumínicas y longitudes de onda captados por los ocelos, pequeños ojos compuestos ubicados y orientados de forma estratégica en la parte superior de la cabeza de determinados insectos.

La adaptación de este método de forma artificial consistirá en la conversión de imágenes tomadas desde una posición y orientación similar a la observada, en incrementos en los ángulos de Euler del objeto que las toma. Lo cual se llevaría a cabo mediante una red neuronal debido al escaso cocimiento del modelo físico que describiría el fenómeno natural en el que se basa el proyecto, así como al, de nuevo desconocido, funcionamiento interno del sistema cognitivo de un insecto.

Es decir, el proyecto consistirá en la programación de una red neuronal capaz de traducir la información recopilada, a la velocidad angular en los ángulos de roll, pitch y yaw. Para lo cual será necesario fabricar una estructura para posicionar tres cámaras imitando la orientación de los ocelos de un insecto y añadirle una IMU para conocer los ángulos que corresponden a cada conjunto de imágenes para poder entrenar y validar la red.

Y el objetivo final será, al añadir la red neuronal diseñada, lograr un sistema capaz de sustituir a la IMU, o, al menos servir de complemento al mismo para evitar la mencionada calibración periódica que este necesita.

ABSTRACT

Currently, the most accepted sensor to know a body's angular rate is an IMU, an instrument that combines the use of accelerometers, gyroscopes and magnetometers. However, it has a remarkable inconvenient, and it is that it needs a periodical calibration because of the cumulative nature of its error.

From a study in the field of biology about insects' sensory organs, it was discovered an angular rate acquisition mechanism which could be imitated artificially. This mechanism consists on the interpretation of luminous intensities variations sensed by ocelli, a type of simple eyes located on the top of some insects' head in a particular way.

The imitation of this mechanism would consist on the conversion of images taken by cameras located and orientated as similar to the insects' ocelli as possible, to angular rates. Which would be fulfilled by a neural network due to our incapability to build a mathematical model to describe this phenomenon, as well as to our insufficient knowledge about the internal behavior of an insect's brain.

Summarizing, the Project consists on the programming of a neural network able to translate images into angular rates. Which also means that it will be necessary to build a structure to hold the three cameras in the correct positions, imitating ocelli's, as well as to hold an IMU, to log the angular rates associated to each image. This last part is essential to train and validate the network designed.

The goal would be to, using the network designed, be able to substitute an IMU or at least use it as a complement to avoid the mentioned periodical calibration.

ÍNDICES

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

INTRODUCCIÓN

"Mira profundamente en la naturaleza y entonces comprenderás todo mejor."

Albert Einstein

Desde que el primer ser humano comenzó a desarrollar tecnología, ya fuera una piedra afilada, la rueda, la red…, su principal motivación es lograr construir, poseer, herramientas o métodos con el objetivo de hacer más sencillas y eficientes las tareas que necesitaba llevar a cabo para sobrevivir, o progresar.

Cada vez que surge una necesidad, se idean una serie de mecanismos para cubrirla, mecanismos que a lo largo del tiempo se van perfeccionando para, como se ha mencionado antes, prosperar, ya sea logrando cubrirla mejor, o comerciando con un excedente.

Este fenómeno se retroalimenta, es decir a medida que se van logrando satisfacer las distintas necesidades, van surgiendo otras nuevas. Una vez una sociedad ha logrado estar a salvo de las inclemencias del tiempo, de los depredadores y el hambre, entonces comienza a hacer dos tareas en paralelo. Por un lado, desarrollar más y más las tecnologías conocidas que le permiten vivir cómodamente, como la edificación de hogares, la producción y distribución de comida y los sistemas de transporte entre muchas otras. Así como, por otro lado, la investigación de nuevas invenciones capaces de cubrir las nuevas necesidades que van apareciendo, a veces de forma espontánea, debido a la aparición de nuevas actividades que necesitan un apoyo tecnológico. Y otras artificial, es decir, los lujos creados por el ser humano en sí mismo, muchas veces con fines lucrativos.

En cualquier caso, el quid de la cuestión, a donde pretendía llegar desde el principio de la introducción, es que desde el principio la respuesta se ha encontrado en la naturaleza. No por supuesto, la respuesta a todos y cada uno de los problemas que van surgiendo, pero sí una proporción sorprendente de ellos. Eso ocurre porque, aunque también creativo, el ser humano es un gran imitador por naturaleza, algo que se puede observar en cualquier bebé o niño pequeño.

Cuando se afiló la primera piedra para lograr un objeto cortante, probablemente fue para imitar la garra de un depredador, o quizás porque el homínido en cuestión iba paseando y se cortó, deduciendo que imitando una determinada forma podría lograr cosas que de forma exclusivamente manual serían muy difíciles. Las redes, antes que las personas para pescar, las usaban las arañas, una rueda era una roca redonda que quizás cayó en un desprendimiento, o quizás era un canto rodado que se movía debido a la corriente de un río. Como estos, hay miles de ejemplos, todos dignos de mención en una clase de historia o en un documento sobre el origen de la ingeniería.

Continuando con la reflexión, hay que comprender que a medida que la tecnología es más sofisticada, los seres humanos más complejos, ese componente natural que se comenzó a replicar se pierde. Poco a poco es cada vez más difícil ver nuevas ideas en la naturaleza para resolver los problemas actuales. De poco sirve observarla para aumentar la capacidad de procesamiento de un dispositivo electrónico, por ejemplo. Sin embargo, nunca se debe perder

de vista, que absolutamente cualquier comportamiento existente en la naturaleza es un mecanismo tecnológico que se ha desarrollado durante millones de años y ha permitido sobrevivir a los seres vivos que conviven con el ser humano en la tierra. Es más, ni siquiera hay que limitarse a los seres vivos, también procesos naturales como el ciclo del agua, se imitan para, por ejemplo, la recolección de sal.

La observación, abstracción y generación de una solución a partir de elementos ya existentes en la naturaleza me parece algo fascinante, ya que desde mi punto de vista la complejidad e inteligencia existente en los seres vivos y en los procesos naturales de nuestro entorno es muy superior a lo que hasta ahora se ha conseguido alcanzar mediante la tecnología. Y no solo despierta una gran curiosidad en mí, sino que me hace pensar que el verdadero futuro se encuentra en la capacidad de la ingeniería para acercarse al máximo posible a comprender cómo utilizar toda esa información para seguir desarrollando el mundo en el que vivimos. A parte, por supuesto, de la gran belleza que a mi parecer tiene, el mismo proceso de investigación, infructuoso o no.

Por esta razón me sentí atraído por este proyecto. Que, de poderse llevar a cabo hasta el final, resultaría ser un nuevo ejemplo de la abstracción e imitación de un sistema ya existente en la naturaleza, para superar un reto que hasta ahora no tiene una solución óptima. El cual es lograr conocer de forma precisa y sin la necesidad de llevar a cabo recalibraciones, los ángulos de Euler de un objeto en el espacio.

EL GIRÓSCOPO DE LA NATURALEZA

Desde que comenzaron a idearse los primeros robots móviles autónomos, se han ido desarrollando paralelamente todo tipo de sistemas de localización y posicionamiento en el espacio. Sobre todo, para los voladores, conocer su inclinación en los ejes del espacio es crucial para poder diseñar un sistema de control capaz de sustituir a un piloto humano. Como solución a este problema, se idearon múltiples soluciones, siendo principalmente el giróscopo, complementado por el resto de sensores presentes en una IMU, el más utilizado. De hecho, la IMU es una gran solución, cuyos pros, son muy superiores a sus contras. Sin embargo, ya existen seres voladores en la naturaleza, y no la necesitan para poder moverse por el mundo. Como determinados insectos.

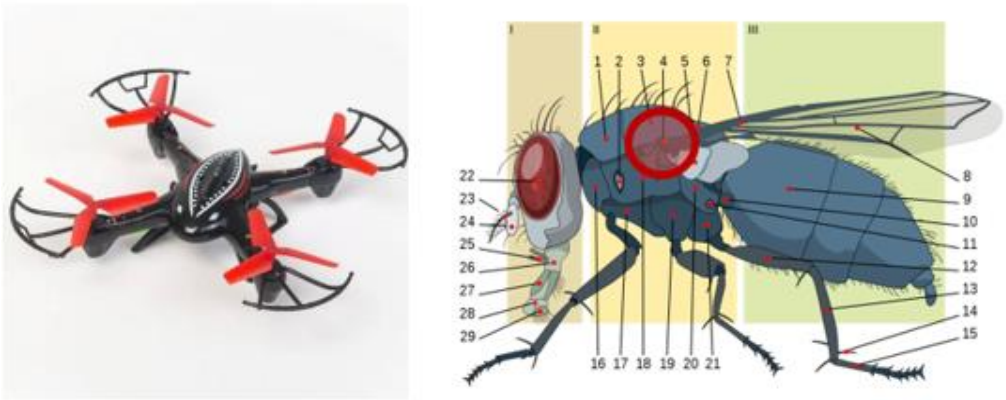

Ilustración 1: Dron vs insecto

Los robots móviles que más necesitan conocer sus ángulos de Euler son sin duda los drones, debido a su maniobrabilidad, y los seres vivos voladores cuyo vuelo es más similar al de un dron, son las moscas, mosquitos, abejas… Insectos pequeños que disponen de alas articuladas en su base, lo que les permite llevar a cabo cambios bruscos de inclinación y dirección durante el vuelo.

En el campo de la biología han estado largo tiempo investigando el método que dicho tipo de insecto volador tiene para lograr tal control y capacidad de maniobra desde el punto de vista del equilibrio y el conocimiento de su posición relativa en los ejes del espacio. Y fue entonces cuando se descubrió el papel crucial que tienen los ocelos.

Ilustración 2: Ocelos dorsales

Los insectos tienen tres tipos de órganos sensoriales:

- Mecanorreceptores:

  Detectan movimientos, vibraciones y otras perturbaciones mecánicas

- Quimiorreceptores:

  Detectan la presencia de sustancias químicas en el aire (olor) o en sustratos (gusto)

- Fotorreceptores:

  Detectan la presencia y calidad de la luz incidente (radiación electromagnética)

Se podría pensar que, en este caso, los órganos sensoriales que son más interesantes para el caso que se trata, son los mecanorreceptores, ya que el giróscopo tal y como lo conocemos, en muchas ocasiones es un dispositivo mecánico. Sin embargo, se descubrió que la clave se encuentra en los fotorreceptores. No es correcto llamar simplemente "ojos" a los órganos fotorreceptores de

los insectos, ya que el concepto es completamente diferente, aunque su principal propósito sea el mismo.

Los "ojos" de los insectos, u "ojos compuestos", están, como su nombre lo indica, formados por muchas facetas similares entre ellas llamadas omatidios, estos son las unidades estructurales y funcionales de la visión de un insecto.

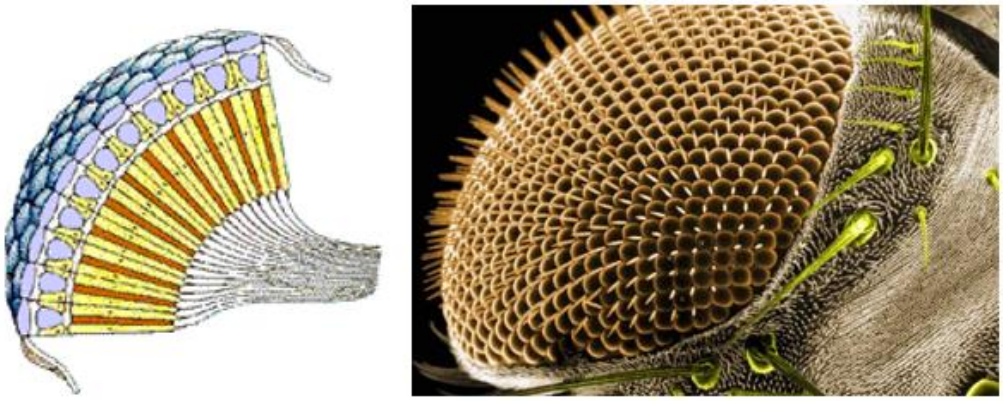

Ilustración 3: Ojo compuesto

Cada faceta apunta hacia una parte diferente del campo visual del insecto, lo que este, tiene que traducir como una imagen compuesta por cada imagen capturada en cada omatidio. Esto implica que no pueden construir una verdadera imagen del entorno, es decir, su agudeza visual es pobre comparada con la de los seres vivos provistos de ojos convencionales. Sin embargo, su capacidad para detectar es superior a la mayoría de animales. En comparación con los humanos, los insectos con ojos compuestos tienen un rango de sensibilidad espectral que abarca muchas más longitudes de onda, sobre todo, más cortas. Por ejemplo, los insectos pueden percibir la luz ultravioleta.

Además de los ojos compuestos frontales, los que utilizan para crear imágenes e interaccionar con el entorno, los insectos tienen ojos simples. Los ojos simples son como los frontales, pero a mucha menor escala, y su nombre técnico es "ocelos".

Los ocelos pueden ser dorsales o laterales, en nuestro caso, nos interesan los ocelos dorsales:

Los ocelos dorsales no forman una imagen o perciben los objetos en el medio ambiente, pero son sensibles a una amplia gama de longitudes de onda de luz y responden rápidamente a los cambios en la intensidad de la luz o sobra. Lo cual es crucial para nuestros intereses. Es mediante a estos órganos, como los insectos son capaces de saber los incrementos en los ángulos espaciales que experimentan durante el vuelo, gracias a la mencionada rapidísima capacidad para detectar cambios en la intensidad de luz, sombra, o en una enorme variedad de longitudes de onda diferentes.

También es muy importante su disposición. Los ocelos son tres, uno por cada eje del espacio, y están orientados como si se tratase de los ejes de un sistema axonométrico.

Es mediante a la información percibida mediante estos pequeños ojos compuestos como los insectos son capaces de conocer su roll, pitch y yaw. Los ocelos, junto con la traducción llevada a cabo por el cerebro de la mosca, son el giróscopo de la naturaleza.

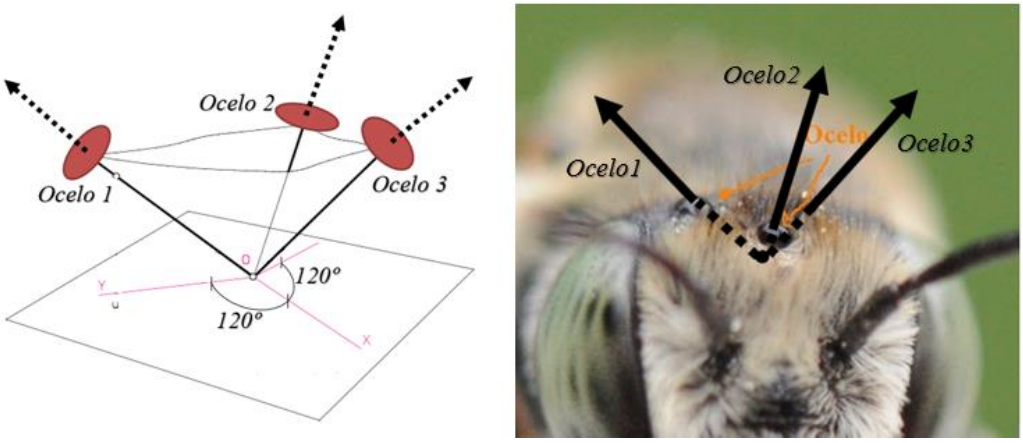

Ilustración 4: Orientación de los ocelos (ejes axonométricos)

ABSTRACCIÓN Y PLANTEAMIENTO DEL SISTEMA

No es posible dotar de ocelos a un dron, pero una aproximación sería sustituirlos por cámaras, y no tratar las imágenes capturadas como imágenes, sino como conjuntos de pixeles (omatidios) cuyo valor va cambiando con el movimiento.

Así se estaría llevando a cabo una imitación de lo que, por ejemplo, una mosca, recibe a través de sus ocelos. Sin embargo, quedaría la parte más compleja, lograr un modelo que sea capaz de convertir esa información en el incremento angular en los ejes del espacio experimentado por el dron, lo cual, el insecto lo hace de forma natural gracias a su propio sistema cognitivo.

Dado que no se conoce ningún modelo matemático que se pueda construir para ser capaces de interpretar los datos recogidos por las cámaras, se intentará llevar a cabo una red neuronal (Deep learning), que a través de un exhaustivo entrenamiento sea capaz de imitar de forma aproximada el proceso que utiliza un insecto para tratar la información que recibe desde los ocelos. Para llevar a cabo el mencionado entrenamiento será necesario conocer la salida correcta para cada conjunto de imágenes, lo cual será posible porque durante los experimentos se recogerán las respuestas proporcionadas por una IMU anexa a la estructura que sujeta las cámaras.

Una red neuronal, aunque por su nombre lo parezca, no imita el comportamiento de un conjunto de neuronas biológicas, ya que sería demasiado complejo y aún no se han logrado buenos resultados con esa tecnología. Una red neuronal, es un modelo matemático, conformado por un montón de pequeños núcleos que contienen una función matemática. Dichos núcleos, o neuronas, se organizan por capas, de manera que, en la primera capa, cada neurona recibe la información que se le asigna como entrada a la red, en la segunda, la información que sale de las neuronas de la primera capa, y así sucesivamente hasta llegar a la última capa, cuya dimensión debe ser igual al número de salidas de la red, y de la cual, deberían salir las entradas traducidas.

Todas las neuronas de cada capa suelen tener la misma función matemática, la cual se aplica a los elementos que entran en ella, dando lugar a la salida. Tener muchas neuronas implica tener una gran variedad de respuestas. Para encontrar el mejor modelo posible, se lleva a cabo un entrenamiento, el cual consiste en asignar un valor en tanto por uno a cada neurona, llamado

peso, que determina la importancia, la contribución, de la salida generada por esa neurona en particular sobre el resultado.

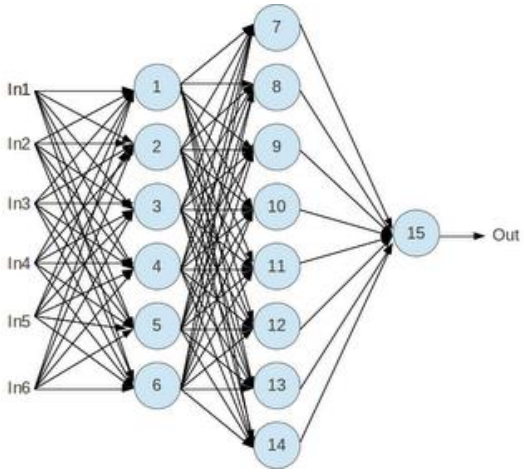

Ilustración 5: Ejemplo de estructura red neuronal 6 entradas 1 salida

Hay una gran variedad de posibilidades a la hora de programar una red neuronal, por lo que, en este proyecto trabajará más de un investigador, y cada uno se preocupará de buscar soluciones con una serie de restricciones, con el fin de no destinar más de un recurso humano a investigar en el mismo entorno. Por ello, se me ha asignado un tipo concreto de red neuronal, que será la secuencial simple, intentando comprobar la viabilidad sobre todo de redes con una función de activación lineal en las neuronas tratadas. Otro tipo de redes como por ejemplo las convolucionales se han asignado a otros compañeros.

El entorno de programación asignado para trabajar sobre las redes ha sido Matlab, aunque debido a la escasez de posibilidades ofrecidas por el Toolbox y el deficiente funcionamiento ante redes del tamaño que se necesita en este proyecto, se terminó utilizando Keras, una librería propia de Python.

LISTA DE ABREVIATURAS Y SÍMBOLOS

DESARROLLO DEL PROYECTO

Para poder comenzar a trabajar sobre redes neuronales el primer paso es tener una base de datos amplia. Para ello hay que definir un entorno de experimentación con una serie de condiciones a tener en cuenta y luego proceder a realizar los experimentos.

Una vez se tenga una base de datos de un tamaño considerable se procederá a programar redes de dimensiones y características adecuadas para el formato de los datos recogidos. Tras lo cual se entrenarán y validarán las redes. Para la validación, es decir, la comparación entre las salidas del sistema recogidas en la fase de experimentación, con las generadas por la red neuronal, es necesario guardar una parte de los datos experimentales, ya que, si se probara el funcionamiento de la red, con señales de entrada y salida que han participado en el entrenamiento de la misma, podría ocurrir que la red diera un resultado correcto por conocer ese caso específico, mientras que nuestra intención no es comprobar la memoria de la red, sino su capacidad de evaluar casos desconocidos para ella.

Cuando se dispone de una serie de redes ya entrenadas y validadas, hay que comparar los diferentes resultados de la validación para elegir la mejor. Esta, pasará entonces un proceso de reentrenamiento y revalidación con diferentes grupos de datos. Es decir, si en la base de datos existen 35357 observaciones de las distintas entradas y salidas, la red se habrá entrenado con 30000 observaciones concretas y validado con las 5357 restantes, y en este paso se pretende volver a entrenar la red y validarla con datos diferentes a los que se utilizaron en la primera fase, con el fin de comprobar si tiene un buen comportamiento independientemente de datos de entrenamiento o validación concretos.

ADQUISICIÓN Y FORMATO DE LOS DATOS

ESTRUCTURA

Físicamente, el instrumento utilizado para la adquisición de datos es un soporte capaz de integrar tres cámaras ubicadas y orientadas correctamente, las cuales captan los datos de entrada de la red neuronal, y en el esquema del cuerpo del insecto equivaldrían a los ocelos. Además, también es capaz de soportar una IMU, cuya información recogida serían los ángulos de roll, pitch y yaw correspondientes a cada trío de imágenes.

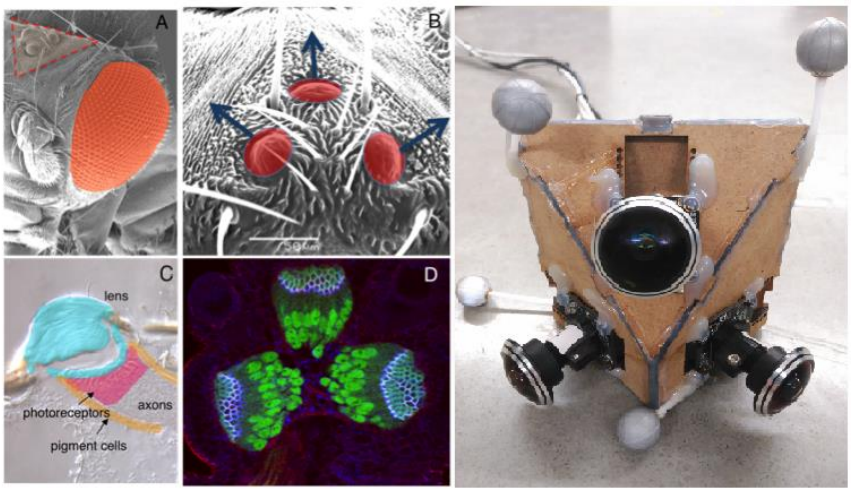

Ilustración 6: Adaptación física de los ocelos

SENSORES

Cada cámara tiene una resolución de 320 x240 pixeles, que, tras ser tratados digitalmente para bajar su resolución, generan imágenes de 8x10 pixeles cada una. En total, sumando todas las imágenes se lograrían 80 pixeles por foto, 240 por trío. Y cada pixel da como información un número entre 0 y 1, que representa la escala de gris captada.

Mientras que la IMU genera mediciones con una precisión de 0.01 rad y registra los ángulos de roll, pitch y yaw.

Ambos instrumentos trabajaron coordinadamente para realizar capturas con una frecuencia de 30Hz.

REALIZACIÓN DE LOS EXPERIMENTOS

Cada experimento se realizó en el interior de una habitación, moviendo la estructura física manualmente durante un periodo de entre 1 y 2 minutos. Se realizaron 14, alternando dos habitaciones diferentes, y cada uno generó un fichero de entre 2200 y 3600 observaciones, con sus correspondientes 240 variables destinadas a guardar el valor asociado a cada pixel, y 3 variables que representan los ángulos registrados por la IMU.

FORMATO DE LOS DATOS

Los datos fueron proporcionados en forma de matrices de 243 columnas. Una primera matriz con las muestras absolutas, es decir, 240 pixeles, con valor entre 0 y 1, y 3 ángulos, en radianes. Y una segunda conteniendo la misma información, pero derivada, es decir, con los incrementos en el valor de cada pixel frente a los incrementos en los ángulos obtenidos mediante la IMU. Dichas matrices se encontraban codificadas en un archivo ". mat", accesible desde Matlab.

En este caso la información relevante se encuentra en la segunda matriz, ya que una imagen en sí misma no aporta información sobre la inclinación del objeto. Mientras que la segunda, relaciona cambios en los datos percibidos por las cámaras con cambios en los ángulos

percibidos por la IMU, que es lo que queremos que la red neuronal sea capaz de hacer.

En total la base de datos está formada por 243 variables, como acabamos de mencionar, y 35.357 observaciones, distribuidos como menciono anteriormente en 14 experimentos, cada uno dentro de un archivo ". mat".

Como entrada para las redes se utilizarán las primeras 240 variables, y como salida y objetivo, se utilizarán las 3 restantes convertidas a revoluciones, para lograr que sus valores oscilen entre números menores, casi ninguno mayor en valor absoluto que 1. Lo cual favorece a las redes, debido a que las entradas rara vez superan la unidad y a que las funciones de activación suelen ser funciones que generan a su salida valores entre -1 y 1 o 0 y 1.

Según avanzó el proyecto se hizo evidente que Matlab no era la mejor herramienta para programar la red neuronal, y se comenzó a trabajar en Keras, a través del entorno de programación R. R es muy parecido a Matlab, pero tiene dos ventajas fundamentales de cara a la realización de este proyecto:

- Es software libre, por lo que no es necesario lidiar con periodos de prueba y licencias para estudiantes. Además, hay muchísima más información, desarrolladores activos aportando ideas e información en la red. Casi cualquier necesidad que pueda surgir a un usuario, no solo ha surgido anteriormente, sino que probablemente su solución ha sido dispuesta de forma gratuita para que cualquiera pueda acceder a ella.
- Está orientado al análisis estadístico, por lo que ofrece muchas posibilidades a la hora de llevar a cabo operaciones relacionadas con él y está mejor optimizado que Matlab para llevarlo. Todo esto es un pro, debido a que no podemos olvidar que una red neuronal no deja de ser un modelo matemático construido a través de análisis y procesos estadísticos iterativos.

Debido a que los datos estaban en un formato inaccesible desde R, fue necesario llevar a cabo un paso intermedio que consistió en la extracción de los datos utilizando Matlab y la generación de ficheros equivalentes en formato de texto separado por comas (".csv"), en los que solo se adjuntaron las matrices que contenían las derivadas de las imágenes y ángulos. Así como un pequeño ejercicio de lectura y conversión de los archivos de texto en matrices compatibles con la librería Keras, ya que las funciones de R, al leer tablas construyen "dataframes", que son matrices capaces de alojar diferentes tipos de variables en cada columna y cuyo direccionamiento es diferente al de las matrices convencionales, por lo que Keras no sabe utilizarlos.

DESARROLLO EN MATLAB

INTRODUCCIÓN

Con la herramienta Matlab, hay dos posibilidades a la hora de programar redes neuronales, el Toolbox "nntools" manejado a través de su interfaz gráfica, o a través de su librería de funciones realizando un script.

El sencillo Toolbox de Matlab para la programación de redes neuronales ofrece una buena oportunidad para poder probar múltiples combinaciones de densidades de capas y funciones de activación, teniendo un muy fácil acceso a una gran variedad de datos estadísticos y gráficas sobre el funcionamiento de la red y la calidad del entrenamiento.

- La interfaz gráfica pone al servicio del desarrollador las herramientas básicas para crear y entrenar rápidamente una red neuronal sin tener siquiera la necesidad de conocer el entorno de programación de Matlab. Pero omite varias herramientas útiles que son de utilidad para este proyecto, como la asignación de drop-outs a las diferentes capas de la red.
- Realizar la red a través de un script utilizando las funciones de la librería ofrece muchas más posibilidades a la hora de configurar una red. Pero el principio de funcionamiento es el mismo, ya que son dichas funciones las que sostienen a la interfaz gráfica, y se espera que los resultados sean también similares.

Ambas herramientas se utilizarán simultáneamente para comprobar si existen diferencias de rendimiento realizando la misma red, y paso a paso se irán mostrando los resultados obtenidos para diferentes configuraciones.

PROBLEMAS E INCONVENIENTES

Antes de comenzar es importante comentar una serie de problemas que surgieron antes de poder programar la primera red:

- Muchas de las funciones de entrenamiento que utiliza Matlab están orientadas para utilizarlas contra redes de dimensiones bastante menores a las que se manejan en este proyecto. Y muchas de ellas coinciden con las más comunes o reconocidas por la propia documentación del programa, por ejemplo, "trainLM". Por lo que al intentar utilizarlas la memoria disponible quedó muy sobrepasada. El error era el siguiente:

  Requested 62400x100011 (46.5GB) array exceeds maximum array size preference. Creation of arrays greater than this limit may take a long time and cause MATLAB to become unresponsive.

  Para solucionar el problema se probó a aumentar la cantidad de memoria disponible para la pila de java (Java Heap Space), pero ni mucho menos fue la solución que se esperaba. Por lo que automáticamente hubo que descartar parte funciones de

entrenamiento que ofrece el Toolbox.

- Entre las demás funciones de entrenamiento, las hay más y menos adecuadas para el entrenamiento de redes como la que se trata en el proyecto, por lo que se continuaron descartando funciones como por ejemplo "TrainR", cuyo funcionamiento consiste en asignar valores aleatorios a la matriz de pesos y comprobar en cada epoch su funcionamiento. Dicha función no solo era extremadamente lenta, sino que en ocasiones el error nunca llegaba a decrecer y el propio programa paraba el entrenamiento. Finalmente, gracias a la ayuda recibida por el tutor del proyecto, me decanté por "TrainGD". Dicha función consiste en modificar los valores de la bias y la matriz de pesos en la dirección en la que desciende el gradiente de la función de rendimiento de la red.

PAUTAS SEGUIDAS PARA GENERAR LA SOLUCIÓN

A partir de entonces comenzó la fase de programación y análisis de distintas redes. Que para ser más eficiente debía seguir una serie de pautas:

- En primer lugar, probar distintas funciones de activación para las distintas capas, centrándose sobre todo en las funciones "tanh" y "linear", sobre todo esta última, ya que así se recomendaba en la asignación del trabajo y porque conceptualmente parece tener sentido que haya una relación lineal, o lineal suavizada en los extremos (tanh)
- En segundo lugar, averiguar hasta cuántas capas ocultas merecía la pena incluir en la red, con el fin de utilizar el menor número posible de capas, para que la red fuera menos compleja.
- En tercer lugar, y ya habiendo decidido el número de capas ocultas de la red, probar varias combinaciones de densidades para cada capa, con la intención de tener cierta orientación sobre aproximadamente qué proporción de densidades ofrecía una mejor respuesta.

Una vez culminados todos estos pasos, se esperaba conseguir un desenlace esperanzador. Los resultados paso a paso, así como las redes creadas para lograrlos, se mostrarán en los próximos apartados.

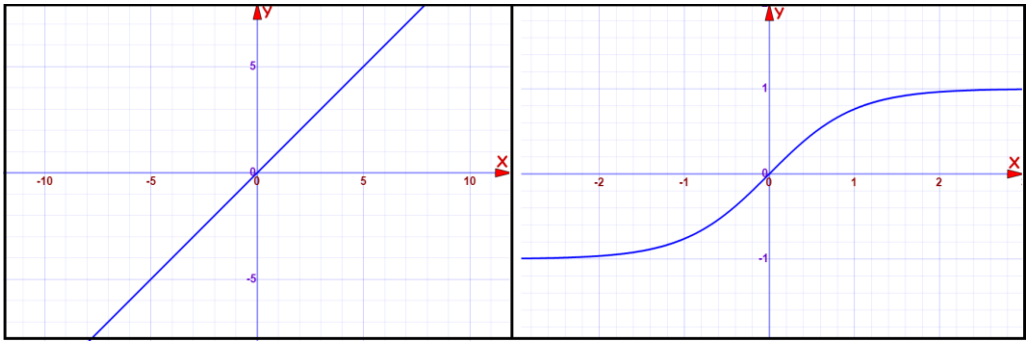

Ilustración 7: Funciones de activación "linear" y "tanh"

PASO 1: FUNCIONES ACTIVACIÓN

En un principio se utilizaron funciones de activación lineales para todas las capas, y más tarde se fueron cambiando por tanh en las capas ocultas, nunca en la capa de salida, debido a que influye directamente en el rango de valores alcanzables en la misma. Siendo siempre mayor el rango asociado a la función lineal ya que esta no se encuentra "saturada" en sus extremos.

En ambos casos se cosecharon pobres resultados y no se podía asegurar que una función ofreciera mejores resultados que cualquiera de las otras, por lo que se optó por seguir adelante e ir probando las diferentes posibilidades cuando se tuviera una estructura asentada para la red.

PASO 2: CAPAS OCULTAS

Como se ha explicado anteriormente, el objetivo de este punto es ver hasta qué número de capas ocultas dejan de mejorar los resultados, analizando durante el test de la red, el error medio al comparar la salida generada por la red neuronal, con la salida recogida durante la fase experimental. Para así tener una red lo más sencilla en cuanto a número de neuronas y capas. Para comparar resultados se programaron redes con las siguientes características:

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Training function	Adaption Learning Function	Performance Function	Net Type	Training Epochs
3	300 Purelin	300 Purelin	170 Purelin	3 Purelin	TrainGD	LearnGDM	MSE	FFBackProp	1000
3	300 Purelin	200 Purelin	120 Purelin	3 Purelin	TrainGD	LearnGDM	MSE	FFBackProp	1000
3	240 Purelin	300 Purelin	170 Purelin	3 Purelin	TrainGD	LearnGDM	MSE	FFBackProp	1000
2	260 Purelin	200 Purelin	3 Purelin	-	TrainGD	LearnGDM	MSE	FFBackProp	1000
2	300Purelin	220 Purelin	3 Purelin	-	TrainGD	LearnGDM	MSE	FFBackProp	1000
2	320 Purelin	220 Purelin	3 Purelin	-	TrainGD	LearnGDM	MSE	FFBackProp	1000

Tabla 1. Set de experimentos para elegir el número de capas ocultas

TRES CAPAS OCULTAS, PRIMER INTENTO

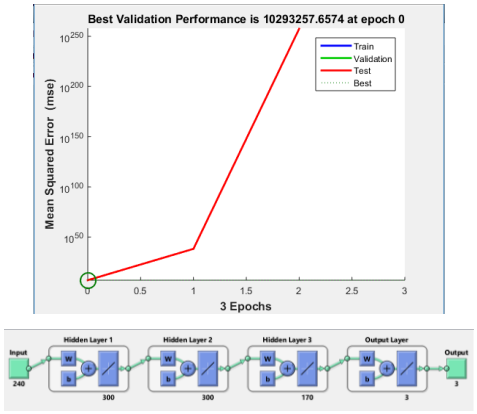

Ilustración 8: 3 capas ocultas, primer intento

TRES CAPAS OCULTAS, SEGUNDO INTENTO

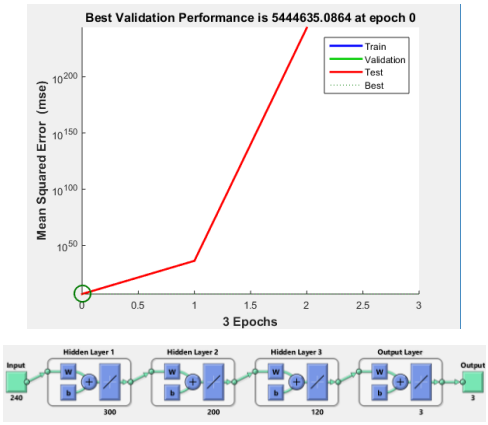

Ilustración 9: 3 capas ocultas, segundo intento

TRES CAPAS OCULTAS, TERCER INTENTO

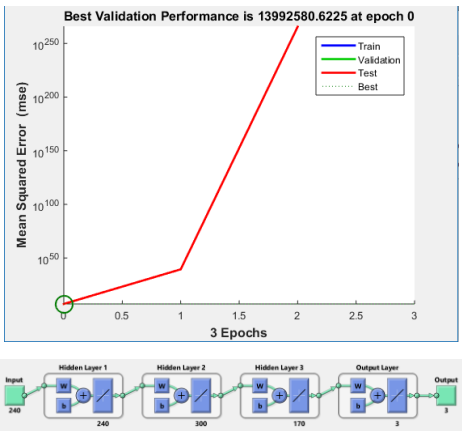

Ilustración 10: 3 capas ocultas, tercer intento

En todos los casos en los que se crearon redes con 3 capas ocultas el resultado del entrenamiento fue de absoluta divergencia.

DOS CAPAS OCULTAS, PRIMER INTENTO

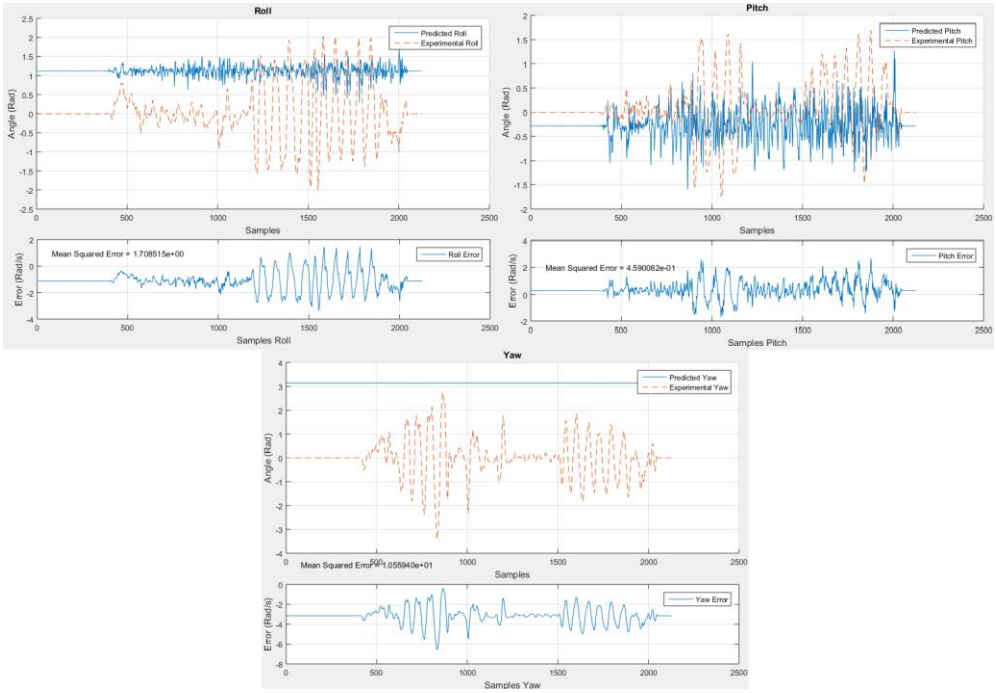

Ilustración 11: 2 capas ocultas, primer intento

DOS CAPAS OCULTAS, SEGUNDO INTENTO

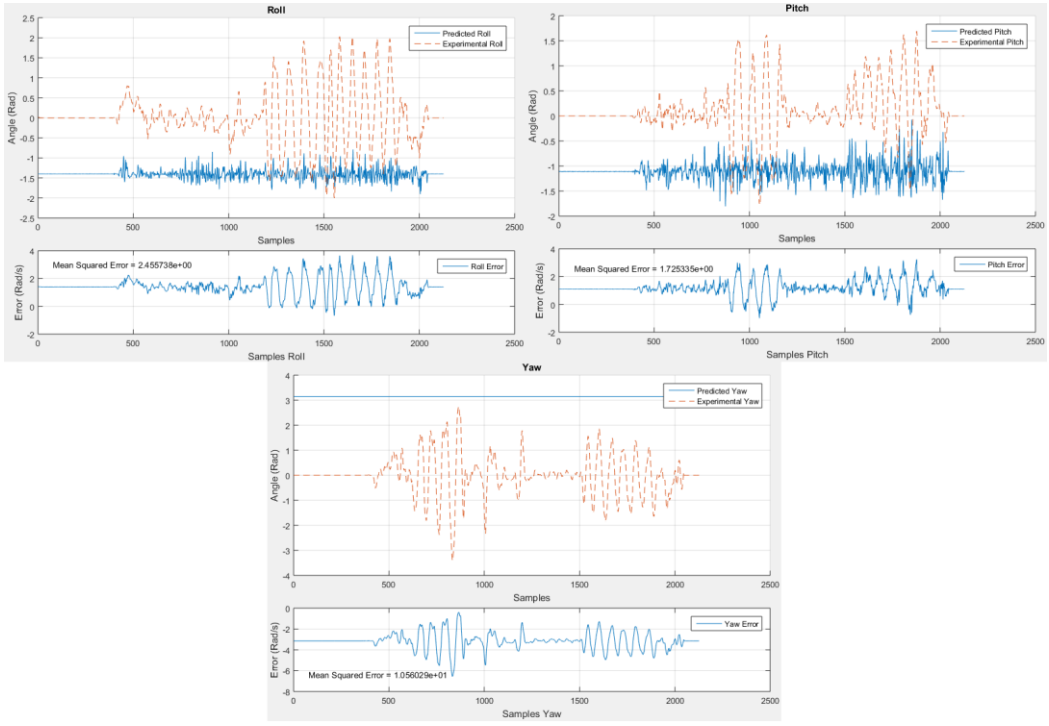

Ilustración 12: 2 capas ocultas, segundo intento

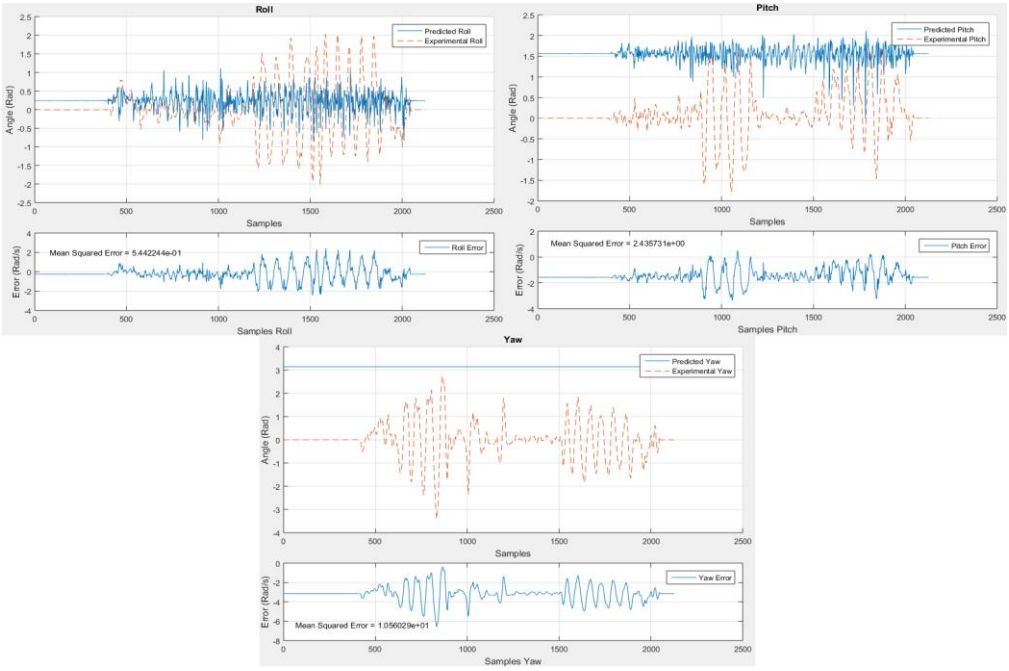

Ilustración 13: 3 capas ocultas, segundo intento

Tras probarlas se llegó rápidamente a la conclusión de que los resultados eran decepcionantes tanto para como para unas como para otras, en todos los casos ni siquiera se llegaban a completar todos los epochs programados para el entrenamiento de la red, ya que se alcanzaba el número máximo de los "validation checks" del Toolbox. Cuyo cometido es detectar si tras un epoch la red no ha convergido. Al alcanzar un determinado número de "validation checks", el entrenamiento concluye, este número es configurable, pero no se modificó el valor por defecto, 6.

Dado que los resultados eran pobres, se probó con otras funciones de entrenamiento, pero no dio resultado. La única capaz de entrenar la red durante más de un epoch fue TrainOSS, que consiste en sustituir el método BFGS para optimización, por una aproximación mediante secante, que es a su vez una versión del método de newton. Así se logra un menor coste computacional y de memoria, y se disfrutan de las ventajas de un método de optimización más completo. Es importante recordar que la mayor limitación de Matlab para trabajar con las redes que precisa este proyecto, es la memoria dedicada.

Encontrar otra función de entrenamiento a parte de TrainGD significó una inyección de moral, aunque muy lenta de entrenar, logró resultados mucho mejores que los obtenidos hasta entonces. Seguían sin ser suficientemente buenos, pero ya comenzó a apreciarse como la red era sensible a los cambios en la entrada. El resultado obtenido fue el siguiente:

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Training function	Adaption Learning Function	Performance Function	NetType	Training Epochs
2	280 Purelin	100 Purelin	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000

Tabla 2. Primera red TrainOSS

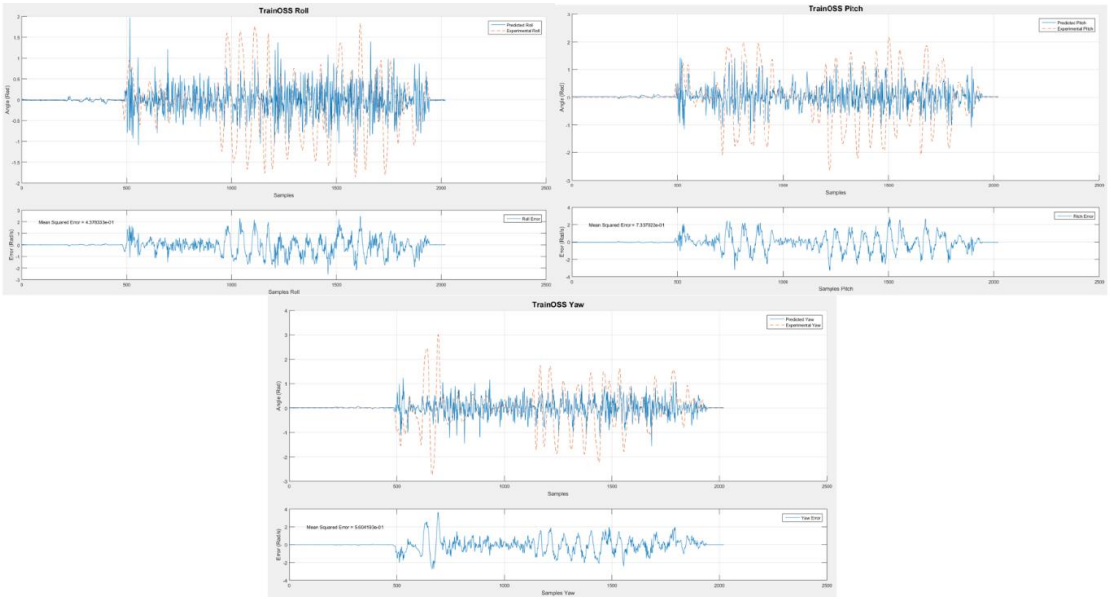

Ilustración 14: TrainOss first simulation

Efectivamente aquí se puede observar como la red por primera vez parece sentir las variaciones. El error sigue siendo alto y la predicción deficiente, pero parece que se ha encontrado un camino a seguir. Aprovechando este resultado se probará para la misma red la función de activación "tansig", en las capas ocultas de la red, pero manteniendo "purelin" en la última:

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Training function	Adaption Learning Function	Performance Function	Net Type	Training Epochs
2	280 Tansig	100 Tansig	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000

Tabla 3. Red TrainOSS con función de activación tanh

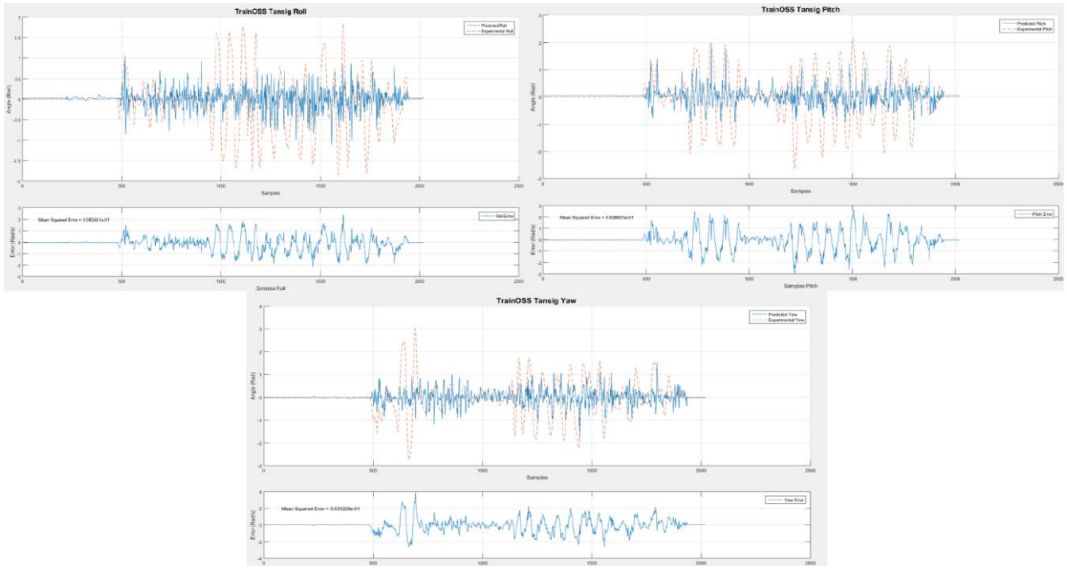

Ilustración 15: TrainOss tansig simulation

De este tercer paso se pueden sacar varias conclusiones:

- La primera, que los resultados ofrecidos al crear y entrenar redes mediante la interfaz gráfica del Toolbox y mediante las funciones pertenecientes al mismo a través de un script son idénticas. La adición de un drop-out en las diferentes capas ocultas en los scripts no afectan al rendimiento de la red, o si lo hace, no es apreciable en las gráficas obtenidas, por lo que, por comodidad, se programaron las siguientes redes siempre mediante un script.
- La segunda, que la función de entrenamiento TrainOSS ofrece mejores resultados que TrainGD y sería la utilizada a partir de ese momento para desarrollar las redes
- Por último, la tercera, que, a la espera de comprobar resultados para diferentes densidades de capas, los resultados eran decepcionantes. Aunque con la nueva función de entrenamiento encontrada, TrainOSS, se había logrado un avance, no se tenían muchas esperanzas de conseguir buenos resultados, ya fuera por el método empleado para diseñar y entrenar las redes, o por la capacidad del tipo de red asignado en el proyecto.

PASO 3: DENSIDAD DE CAPAS

Tras hacer una serie de intentos orientativos para saber entre qué valores concretos realizar una mayor cantidad de pruebas, se pretendía programar un bucle que fuera probando diferentes combinaciones entre densidades de la primera y la segunda capa oculta y fuera guardando el error cuadrático medio de cada red probada. De esta manera se comprobarían muchísimas opciones sin necesidad de supervisión más allá del posterior análisis de los resultados.

Sin embargo, el proceso de entrenamiento resultaba tan lento y los resultados eran tan desalentadores durante los intentos orientativos, que se renunció a llevar este último paso para buscar métodos alternativos para gestionar redes neuronales.

RESULTADOS INTENTOS ORIENTATIVOS

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Quarta capa	Training function	Adaption Learning Function	Performance Function	Net Type	Training Epochs
2	300 Purelin	220 Purelin	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000
2	260 Purelin	200 Purelin	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000
2	280 Purelin	160 Purelin	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000

Tabla 4. Redes para exploración de resultados

PRIMER INTENTO

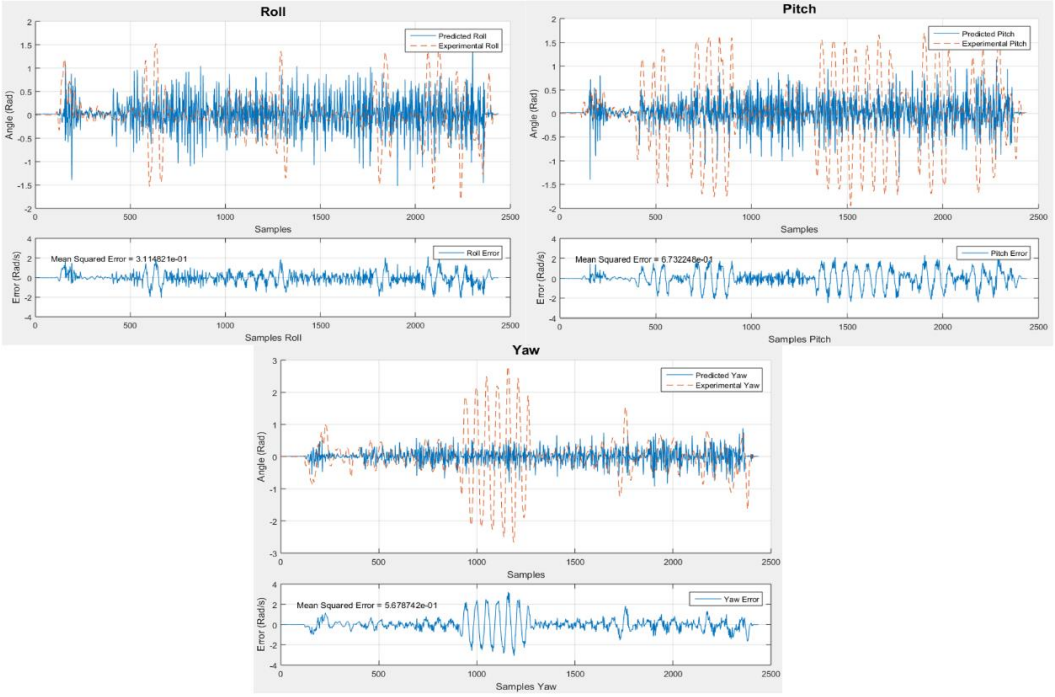

Ilustración 16: Resultados orientativos red 1

SEGUNDO INTENTO

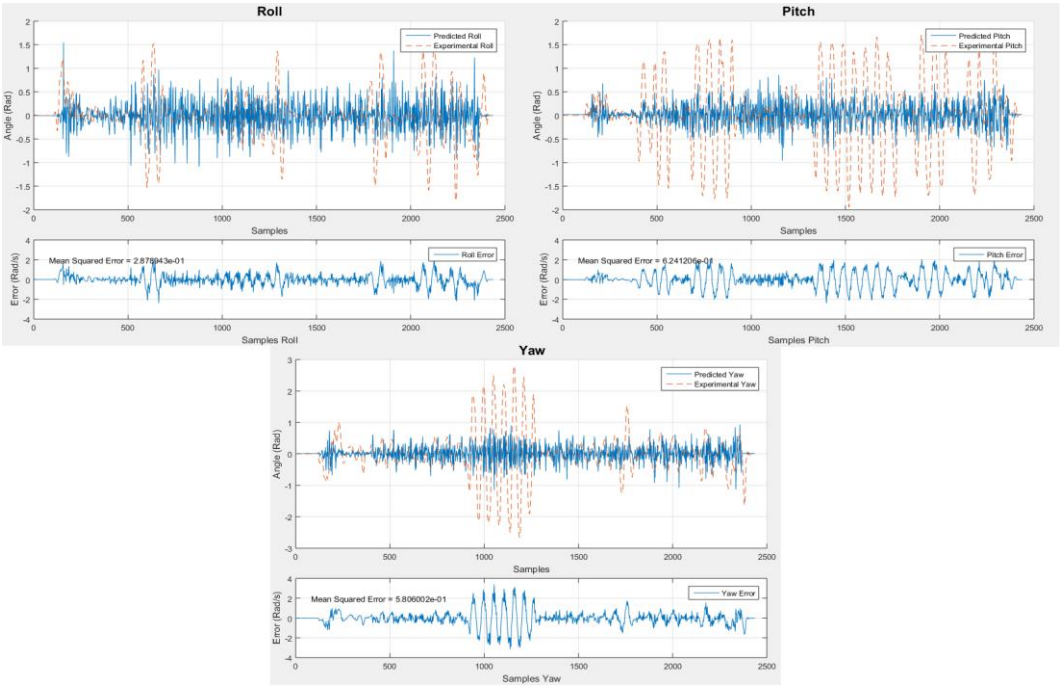

Ilustración 17: Resultados orientativos red 2

TERCER INTENTO

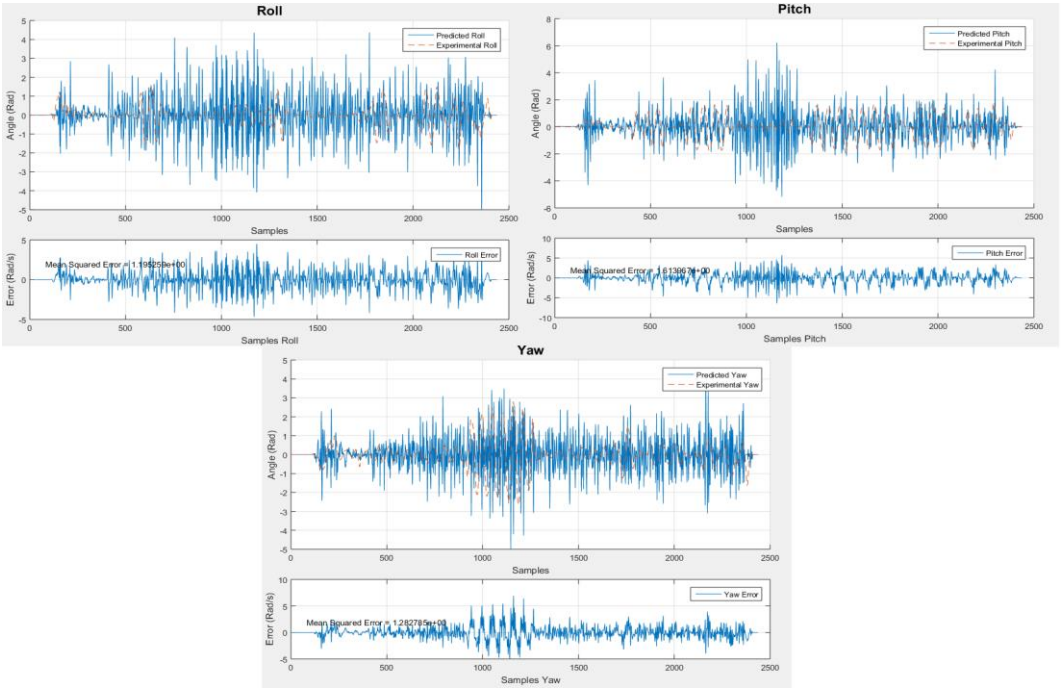

Ilustración 18: Resultados orientativos red 3

TEORÍA DEL YAW

Reflexionando sobre los resultados, se pensó que el ángulo con diferencia más difícil de modelar sería el yaw. La lógica seguida es que probablemente la manera más sencilla de orientarse angularmente es:

- A partir de la línea del horizonte, si se realizaran los experimentos al aire libre
- Con las líneas que conforman las aristas interiores de una habitación rectangular, debido a las sombras que siempre se generan en estos lugares

Ilustración 19: Teoría del horizonte/Aristas

Esta teoría se basa en el conocimiento de que tanto las esquinas, como la línea que separa la silueta del terreno con el cielo, tienen un gran contraste debido a color o a sombras, lo cual genera un gradiente alto de intensidad lumínica.

Como consecuencia de esta reflexión, se quiso comprobar si quizás el intentar modelar el yaw a la misma vez que roll y pitch con las mismas imágenes, la red crecía en complejidad y perdía capacidad de calcular roll y pitch con mayor precisión. Para ello se comprobaron las redes que "mejor" han funcionado hasta ahora, pero esta vez con tan solo roll y pitch como salidas:

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Training function	Adaption Learning Function	Performance Function	Net Type	Training Epochs
2	300 Purelin	220 Purelin	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000
2	260 Purelin	200 Purelin	3 Purelin	-	TrainOSS	LearnGDM	MSE	FFBackProp	1000

Tabla 5. Redes sin yaw

PRIMERA RED SIN YAW

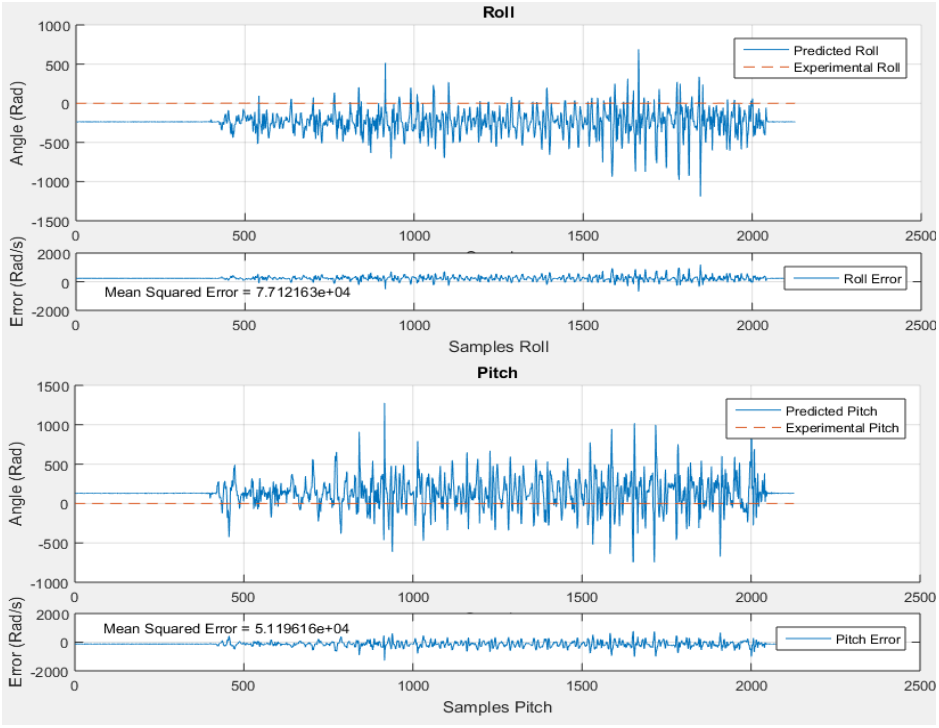

Ilustración 20: Teoría del horizonte resultado 1

SEGUNDA RED SIN YAW

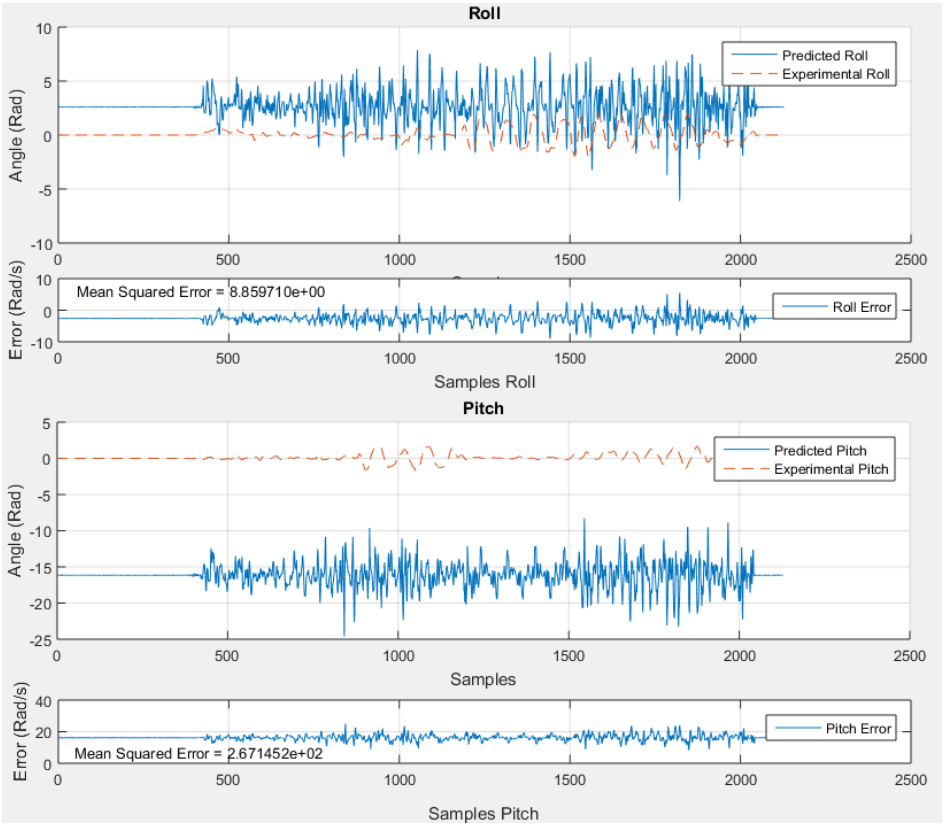

Ilustración 21: Teoría del horizonte resultado 2

CONCLUSIÓN

El Toolbox de Matlab para la programación de redes neuronales es sencillo y completo. La documentación de MathWorks es muy extensa, y ofrece facilidades para que alguien sin tener nociones básicas de programación ni experiencia con redes neuronales pueda utilizarlo e ir aprendiendo poco a poco. Sirvió como punto de partida para comenzar a entender el proyecto e ir adquiriendo conocimiento, pero a la hora de la verdad no resultó ser la mejor herramienta para tratar redes como la propia del proyecto.

Uno de los problemas más destacables y que pueden haber sido determinantes es que algunas de las funciones de entrenamiento más comunes y recomendadas, incluso en la propia documentación del Toolbox, no podían ser utilizadas debido a que requerían que Matlab aprovechara una cantidad de espacio en memoria superior al soportado por el programa. Esto dejó malas sensaciones, principalmente porque dio la sensación de que no está preparado para manipular redes del tamaño y características que es necesario para el proyecto, y también porque quedará la duda de si habrían podido lograrse buenos resultados utilizándolas.

DESARROLLO CON KERASR

INTRODUCCIÓN

KERAS

Keras es una librería para diseño de redes neuronales originalmente escrita en Python, es de código abierto y tiene una gran ventaja y es que permite utilizar el mismo código en CPU y en GPU indistintamente, lo cual aumenta muchísimo la velocidad de entrenamiento de las redes.

Tiene además una sintaxis muy sencilla y ofrece incluso más datos estadísticos referentes al entrenamiento y resultados de las redes que el Toolbox de Matlab. Keras fue creado por François Chollet, ingeniero de Google, y está preparado para diseñar y gestionar redes y conjuntos de datos de grandes dimensiones al no tener una memoria dedicada de un tamaño concreto, como era el caso de Matlab.

Aprovechando que R es también de código abierto y que su lenguaje es similar al de Python, en septiembre de 2017 se anunció que se había traducido la librería de Keras para posibilitar su uso en R y se había añadido al repositorio oficial del entorno de programación. La sintaxis tanto de los lenguajes de programación como de la configuración de la librería son tan parecidos que un individuo que haya experimentado con Keras en cualquiera de sus dos formas es capaz de comprender y utilizar fácilmente la librería en ambos entornos. Esto tiene otra ventaja, y es que las soluciones compartidas en internet son útiles da igual el idioma en el que estén escritas para todos los usuarios.

Además, la página oficial de Keras pone al servicio de los desarrolladores toda la documentación necesaria muy bien estructurada, además de múltiples ejemplos para creación desde cero de redes de cualquier tipo o para la utilización de las diferentes funciones.

¿POR QUÉ R Y NO PYTHON?

Según la página web "Adictec", dedicada a la búsqueda y publicación de las últimas noticias, aplicaciones, artículos y consejos sobre tecnología e informática, los lenguajes más utilizados y mejor preparados para aplicaciones en el campo de la inteligencia artificial son:

- En primer lugar, Python, debido a su simpleza, velocidad de ejecución y disponibilidad de aplicaciones y librerías que facilitan el desarrollo.

- En segundo lugar, R, debido a que es muy cómodo y efectivo a la hora de analizar y manipular datos con fines estadísticos, a su simpleza y a la disponibilidad de paquetes que facilitan la implementación de algoritmos de aprendizaje automático.

- En tercer lugar, Lisp, uno de los pioneros, escrito por John McCarthy, padre de la inteligencia artificial, en 1958. Es muy completo y tiene muchas ventajas, pero no es un lenguaje de propósito general como lo son Python y R, por lo que es necesario complementarlo con otros programas para llevar a cabo el proceso desde cero y documentarlo. Además, al llevar tantos años en funcionamiento, la mayoría de sus características más interesantes, han migrado a otros lenguajes, por lo que ha perdido singularidad.

- En cuarto lugar, Prolog, al nivel de Lisp, especialmente acondicionado para trabajo en proyectos médicos y diseño de sistemas de inteligencia artificial expertos. Es un lenguaje algo más complejo, lo cual la condena verse perjudicada en el ranking de la página web.

- Por último, java. Se encuentra en el ranking debido a sus widgets, que facilitan mucho la interacción del lenguaje y el proceso de diseño al usuario en parte gracias a lo atractivos y sofisticados que se aprecian sus gráficos e interfaces.

Realmente es muy difícil de evaluar la diferencia de funcionamiento de los distintos lenguajes, por lo que este ranking ayuda sobre todo a saber, de cara al desarrollador, cuáles son las características principales de cada lenguaje. Concretamente entre Python y R no parece haber grandes diferencias más allá de la velocidad de ejecución, por lo que la elección entre uno y otro dependería de las facilidades que ofrece el entorno.

Se eligió utilizar Keras en R en vez de en Python debido a mi experiencia personal con ambos entornos. Mientras que en Python tengo poca o ninguna experiencia, y, en cualquier caso, orientada al procesamiento de imágenes y gestión de archivos, llevo realizando análisis de datos de manera profesional en R desde principios del presente curso. Además, R dispone de una interfaz muy completa, Rstudio, muy parecida a la de Matlab, que permite tener una visión global de las variables, el directorio de trabajo y la documentación de las funciones de los distintos paquetes o librerías que no ofrece Python y que simplifican mucho las tareas de análisis de datos.

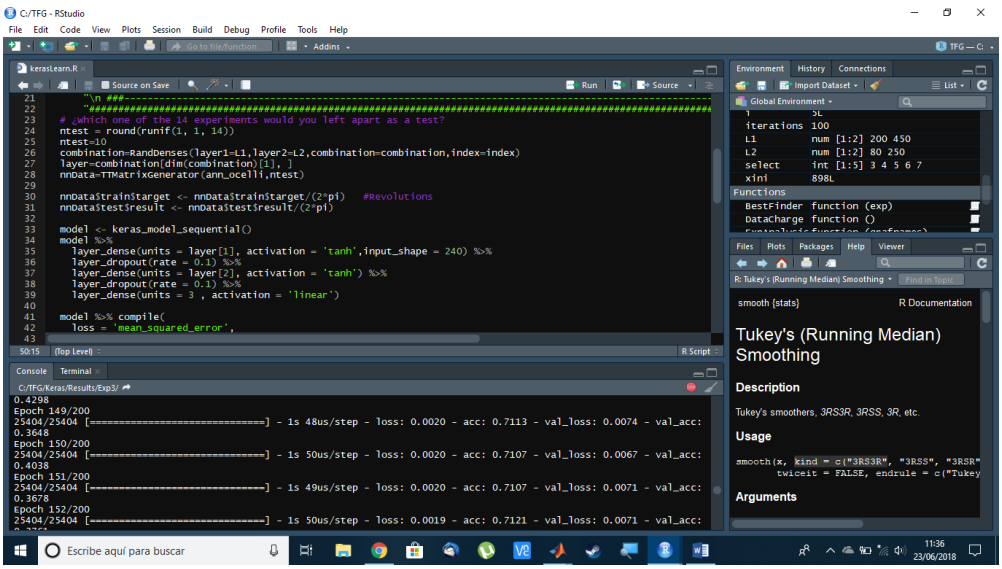

Ilustración 22: Interfaz Rstudio

PAUTAS SEGUIDAS PARA GENERAR LA SOLUCIÓN

Con el fin de ser lo más eficiente posible a la hora de analizar el comportamiento del mayor número posible de redes con la mayor cantidad de variantes posibles, se creó un protocolo a seguir:

- En primer lugar, se realizaron una serie de experimentos para comprobar si el funcionamiento del paquete ofrecía mejores resultados que el Toolbox de Matlab, así como para ver si estos eran lo suficientemente esperanzadores como para continuar con la investigación.

- En segundo lugar, se automatizó todo el proceso basándose en bucles. Cada nueva red se crearía automáticamente sin necesidad de supervisión, se entrenaría, y se almacenaría junto con un análisis de su respuesta dentro del directorio de trabajo. Posteriormente, una función analizaría los resultados en función del índice de precisión adquirido al testar cada red, y solo graficaría los mejores almacenando las imágenes de nuevo dentro del directorio. Y, por último, otra función analizaría el conjunto completo de resultados de todos los experimentos hechos ofreciendo las medias de las precisiones obtenidas para cada tipo de experimento.

- En tercer lugar, se generaron experimentos a partir del sistema de creación y análisis de redes automático creado. Cada experimento constaría de un número variable de redes diferentes y tendría como características a modificar las funciones de activación de las distintas capas, la cantidad de epochs de entrenamiento, el BS también del entrenamiento, y la función de optimización. Generando cada experimento a partir del análisis del anterior.

- En cuarto lugar, se realizó un análisis de los distintos experimentos en función de la precisión obtenida al testar cada red.
- En quinto lugar, se llevó a cabo una "vuelta atrás" para explorar más profundamente la configuración más exitosa hasta el momento, añadiendo una capa oculta más a la red para comprobar si se obtienen así mejores resultados
- En sexto lugar, se realizó un análisis de la viabilidad de la aplicación de un post-procesamiento mediante un filtro de suavización
- En séptimo y último lugar, se realizó un análisis final de los resultados y se expuso la mejor red encontrada.

Una vez culminados todos estos pasos, se obtuvo una red neuronal suficientemente precisa para presentarla como resultado del proyecto, así como un análisis bastante completo de más de un millar de redes diferentes que explica de manera sencilla la capacidad de cada tipo de red frente a las especificaciones del proyecto y la evolución del mismo.

Los resultados paso a paso, así como las redes creadas para lograrlos, se mostrarán en los próximos apartados.

Este paso fue largo debido a que todos o casi todos los ejemplos encontrados sobre programación de redes neuronales en Keras con el modelo secuencial estaban hechos sobre objetivos categóricos, mientras que en este proyecto los objetivos son continuos. Por lo que fue necesario un estudio más profundo de la documentación para encontrar qué funciones y qué argumentos para las mismas era necesario modificar para construir redes como la que se precisaba.

Una vez se aprendió a utilizar el paquete para programar las redes del tipo deseado se probaron dos para comparar los resultados que se obtienen. Estas fueron las siguientes:

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Optimization function	Performance Function	Batch size	Net Type	Training Epochs
2	350 linear	240 linear	3 linear	RMSprop	MSE	128	Sequential	100
2	300 tanh	200 tanh	3 linear	RMSprop	MSE	128	Sequential	100

Tabla 6. Redes sin yaw

TEST FUNCIÓN LINEAL

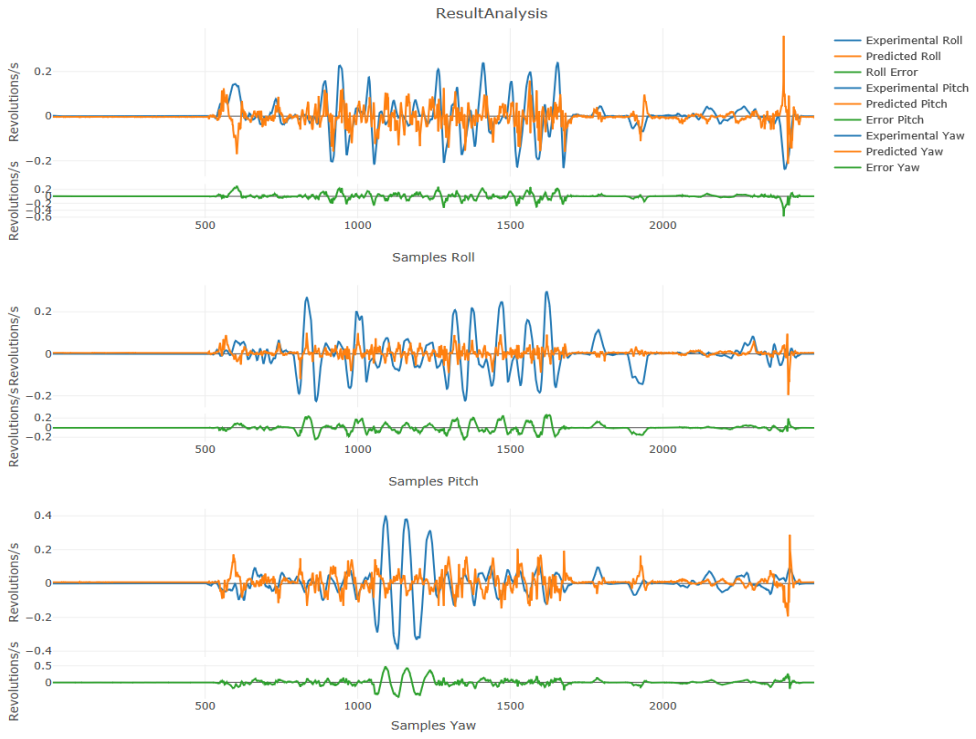

Ilustración 23: Resultados orientativos Keras lineal

TEST FUNCIÓN TANGENCIAL HIPERBÓLICA

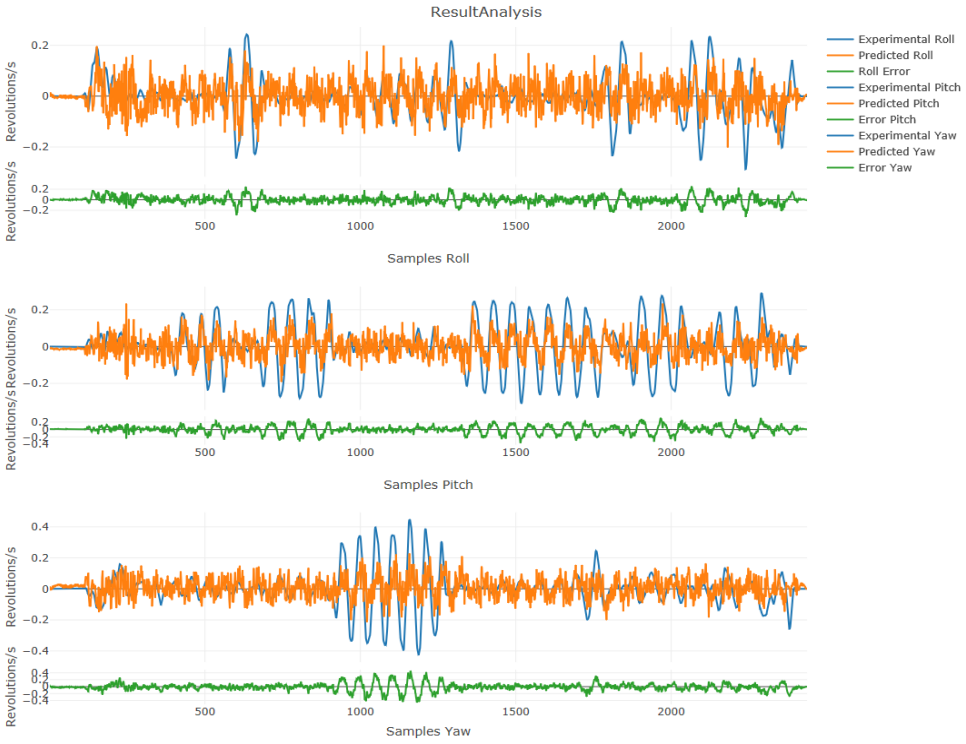

Ilustración 24: Resultados orientativos Keras tangencial hiperbólico

CONCLUSIÓN

Keras respondió mucho mejor que Matlab. En las gráficas se ve como la red es capaz de interpretar las imágenes para saber cuándo hay incrementos y descensos. En estas gráficas se ve como con dos redes casi cualesquiera, se logran resultados que generan optimismo. Es por esta razón que se decidió avanzar hacia el siguiente paso.

PASO 2: AUTOMATIZACIÓN DEL PROCESO

Testar redes una a una es un proceso tedioso, da lugar a tiempos muertos entre un experimento y otro, ya que necesitan unos minutos para entrenarse y graficarse. Además, pueden aparecer muchos criterios diferentes a la hora de elegir las densidades de cada capa de una red, y gran parte del trabajo consiste en elegirlas a ciegas y ver el resultado.

Por estos motivos, se decidió crear un bucle capaz de, con una serie de parámetros introducidos de forma manual, testara redes y guardara los resultados automáticamente para su posterior análisis, que también sería automático.

En total se escribieron 7 funciones y un script maestro que las utiliza todas para llevar a cabo el proceso de experimentación de forma automática.

DATACHARGE ()

En primer lugar, hacía falta una función capaz de extraer y almacenar en un formato adecuado para R toda la información de los experimentos con la estructura física explicada en la introducción. Dichos datos estaban en formato ".m", luego, Matlab los tradujo a ".csv", tal y como se explicó en el apartado de *Formato de los datos*. Y ahora es necesario que R los traduzca a ". RData".

Para ello se realizó la función *DataCharge ()* cuyo cometido es abrir uno a uno los archivos y almacenar los datos en forma de lista. El primer elemento de la lista será otra lista, que almacenará todos los datos separados entre entradas y objetivos para el entrenamiento. Y el resto de elementos serán listas de todos los experimentos por separado, que a su vez también estarán divididos en entradas y objetivos.

TTMATRIXGENERATOR (ANN_OCELLI, NTEST)

En segundo lugar, se hizo necesaria una función para generar matrices aptas para Keras a partir de las listas creadas con *DataCharge* ya que estos están en forma de "dataframe". Como se explicó en *Formato de los datos*, un "dataframe" no es una matriz, es en realidad una lista de vectores de la misma longitud, pero que pueden ser de diferentes tipos, cuyo direccionamiento no es igual al de una matriz, aunque en apariencia parezcan iguales. Este es el formato adoptado por los datos que se recogen mediante las funciones de lectura de ficheros, de ahí que necesitemos traducirlos a matrices, lo cual solo se puede llevar a cabo porque en este caso, todos los vectores que conforman el "dataframe" y que representan a cada columna de la matriz, son del mismo tipo, double.

La misma función que cambia el formato de las matrices, también recibe como argumento un

número, este se corresponderá con el del experimento que se quiere dejar a parte para posteriormente testar la red, una vez entrenada con los demás experimentos. Y esta será la otra utilidad de la función, devolver una lista con los datos destinados para entrenamiento separados de los datos destinados para examinar la red en función del número de experimento especificado en el argumento.

RANDDENSES (LAYER1, LAYER2, COMBINATION, INDEX)

Tras esto, fue necesario crear una función capaz de generar valores aleatorios para la densidad de las capas ocultas de la red. Esto parece tarea sencilla, utilizando una función predeterminada que devuelve números al azar. Nuestra intención es que a cada iteración del bucle maestro se genere una combinación nueva de densidades de capas. Para lograr esto, la función recibe 4 argumentos:

Los dos primeros, son dos vectores que representan entre qué dos valores debe estar el número de neuronas asignados a la primera y a la segunda capa. El tercero es una matriz con las combinaciones utilizadas hasta el momento, las cuales se contemplan para no repetirlas. Y el cuarto es el índice del bucle maestro. En caso de ser la primera iteración, se creará la matriz "combination" con dos valores cualesquiera, si no, se buscará una nueva combinación diferente de las existentes. La función devolverá una matriz con todas las combinaciones hasta el momento, más la nueva generada. Dicha salida será argumento de la función en la siguiente iteración.

RPYDRAW (NNDATA, RESULTS, FILTER)

A continuación, surgió la necesidad de una función que a partir de los datos experimentales y los resultados obtenidos de testar una red fuera capaz de realizar una gráfica mediante el paquete "*Plotly*". Dichos datos y resultados se le pasarán como argumentos, así como uno cuyo valor es booleano, y que determinará si se quiere que se aplique un filtro que suavice la forma de la gráfica de los resultados, esto se utilizará a la hora de comprobar si merecería la pena aplicar un filtro de suavización como post-procesamiento.

GRAPHDIR (MODEL, EXP)

Con la intención de graficar los resultados obtenidos se realizó una nueva función. Ésta crea una carpeta dentro del directorio dedicado a cada conjunto de experimentos. Para saber dónde crearla, necesita como argumento el número del conjunto. A continuación, a partir de un modelo, también recibido como argumento, lleva a cabo un test para cada set de datos experimentales cargándolos mediante la función *TTMatrixGenerator (ann_ocelli, ntest),* y grafica cada resultado mediante *RPYDraw (nnData, results, filter)* guardándolo como PNG en la carpeta antes creada.

BESTFINDER (EXP)

Esta función busca dentro el conjunto de experimentos cuyo número se especifica en el argumento de la función cuál ha sido la red cuya precisión en el test de resultados es mayor. Luego carga su modelo y llama a la función *GraphDir (model, exp)* antes de devolver en forma de lista las características principales de la red buscada.

EXPANALYSIS (GRAFNAMES)

Esta función busca en el directorio cuántos conjuntos de experimentos se han realizado y va entrando en ellos uno por uno para extraer vectores con el índice de precisión adquirido al testar cada red del conjunto. A continuación, dibuja un gráfico que concatena todos los vectores en una nube de puntos en la que se aprecia el rendimiento de cada conjunto de experimentos. Los cuales se diferencian por colores. También se añade al gráfico una línea que indica la media de precisión de cada experimento. Ver *Ilustración 51*.

SCRIPT MAESTRO

El programa principal tiene como objetivo, cambiando una serie de parámetros, diseñar, entrenar, testar y analizar un número determinado de redes con una serie de características concretas.

Para explicar su funcionamiento, será necesario describir el directorio del proyecto, así como el código. Para ambas cosas se utilizarán dos sencillos diagramas:

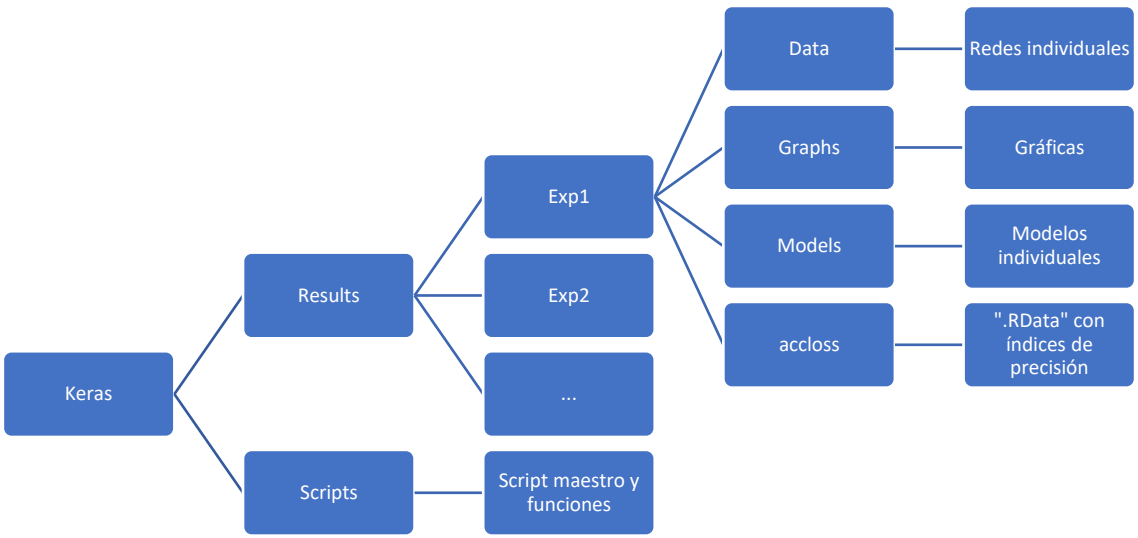

Tabla 7. Diagrama directorio de trabajo

Tabla 8. Diagrama script maestro

PASO 3: EXPERIMENTACIÓN

INTRODUCCIÓN

Una vez construido y depurado el script y las funciones comentadas en el apartado anterior se comenzaron a programar redes de forma automática. Cada conjunto de experimentos tendría una serie de características en común, las únicas que cambiarían a cada iteración serían las densidades de cada capa oculta, cuyos valores serían generados de forma aleatoria.

De entre todas las funciones de activación posibles las más válidas en teoría para el proyecto serían la lineal y la tanh, al igual que en Matlab, debido a que la mayoría de las otras se encuentran más focalizadas al diseño de redes cuyas salidas fueran categóricas.

Durante los primeros dos conjuntos de experimentos, para elegir la densidad de cada capa oculta se exigió al programa que solo tomara valores entre 200 y 350 para la primera, y entre 150 y 250 para la segunda. A partir de entonces, se aumentaron dichas cotas, con la intención de barrer un área mayor de combinaciones, teniendo la primera capa una densidad neuronal de entre 22 y 450, y la segunda de entre 80 y 250. Se eligieron estos valores teniendo en cuenta los tamaños de los vectores de entrada y salida:

- A la entrada habrá siempre un vector de longitud 240, 80 por cada una de las tres imágenes, por lo que, se creyó conveniente tener un número próximo o mayor a 240 para dedicar suficientes neuronas al tratamiento de cada pixel de información.
- Por otro lado, a la salida, solo debería haber 3 neuronas, por lo que parece lógico escoger un número de neuronas preferentemente menor al de la primera capa para tener un valor intermedio de neuronas entre la capa de entrada y la de salida.

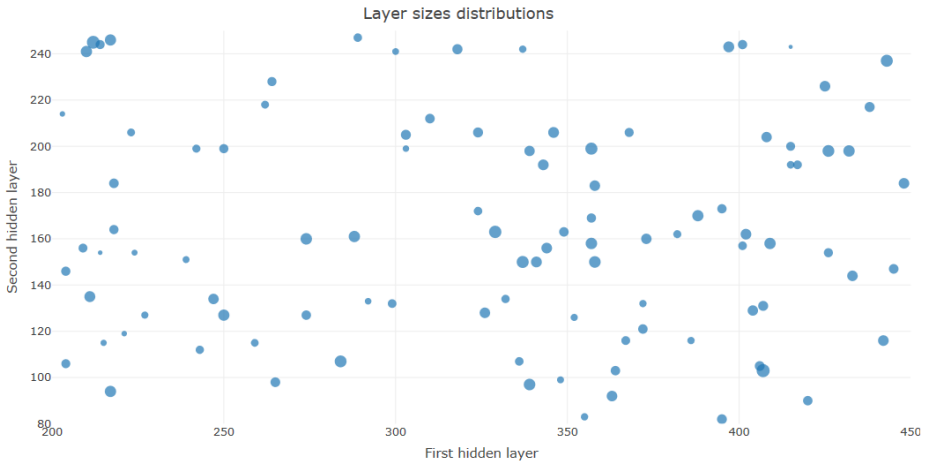

Ilustración 25: Distribución de tamaños de capa de un conjunto de 100 experimentos

Esto daría lugar a 42500 combinaciones posibles. Explorarlas todas sería un proceso excesivamente lento, además de poco eficiente, ya que no sería útil comparar redes de 240 y 340 neuronas en sus capas, con, por ejemplo, redes de 241 y 340. Por lo que el número de iteraciones llevadas a cabo por el bucle, normalmente estará entre 100 y 300, se considera que se está barriendo correctamente toda el área de experimentos pese a tener un número de redes muy inferior a la cantidad de combinaciones posibles, debido a la probabilidad al generar un número aleatorio. Por si acaso, se graficaron todas las combinaciones, utilizando el eje X de la función como número de neuronas en la primera capa oculta, y el Y para la segunda. Tal y como se ve en la *Ilustración 25*.

Para elegir el número de epochs se comenzó a entrenar una red con un número y muy alto y se observó a partir de qué epoch dejaba definitivamente de mejorar la precisión y bajar el error, se sumaron 50 epochs más para dejar margen a redes cuyo entrenamiento pudiera ser más largo. El número final de epochs escogido fue 200.

Las variables restantes son la función del error y el BS. La función del error debía de nuevo ser una que no estuviera orientada a la búsqueda de salidas categóricas para la red, por lo que se escogieron MSE y MAE como opciones. Y el BS, se fijó en 128 en un principio y se fue variando. Según la documentación de Keras, un BS mayor, exigirá más memoria y hará más costoso el entrenamiento, pero deberá lograr una precisión mayor.

TABLA EXPERIMENTOS

Conjunto	Iteraciones	Función activación	Función Error	Epochs	Batch Size
1	300	Linear	MSE	100	128
2	200	Linear	MSE	100	256
3	100	Tanh	MSE	200	256
4	100	Tanh	MAE	200	128
5	200	Linear	MSE	200	128

Tabla 9. Set de experimentos para elegir el número de capas ocultas

PRIMER CONJUNTO

El primer paso fue experimentar con funciones de activación lineales. Se programaron 300 redes neuronales diferentes con un BS de 128, valor escogido como punto de partida. Dado que la red dejaba de converger en torno a los 60 epochs, se programó que utilizara siempre 100 para dar un margen. La mejor red alcanzada logró una precisión del 51.2%.

Capas ocultas	Primera capa	Segunda capa	Tercer a capa	Optimization function	Performance Function	Batch size	Net Type	Training Epochs
2	331 linear	247 linear	3 linear	RMSprop	MSE	128	Sequential	100

Tabla 10. Mejor red del conjunto 1

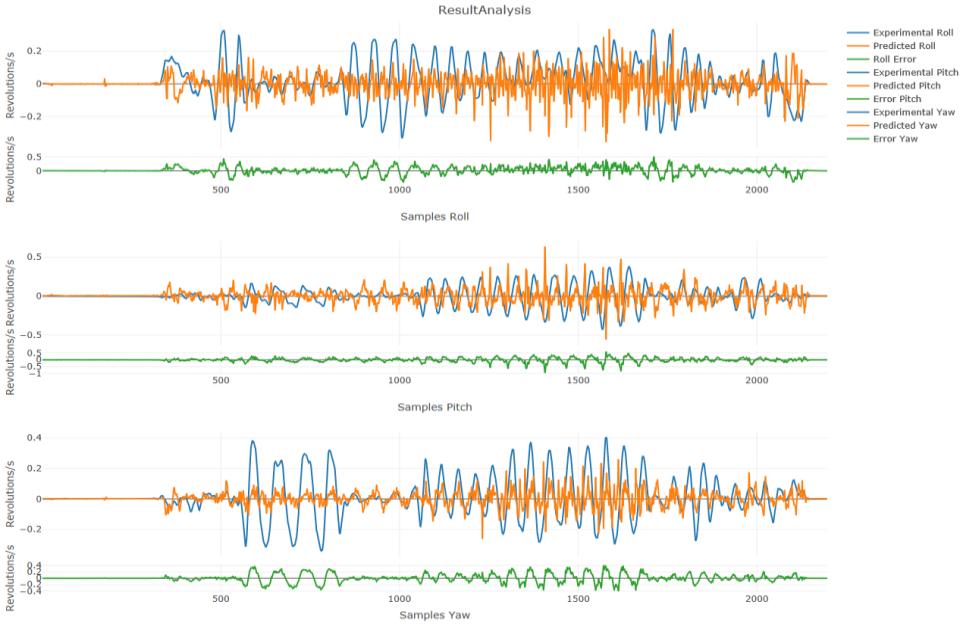

Ilustración 26: Resultados de la mejor red del conjunto 1

Sin embargo, es un valor curioso la precisión calculada a partir del test de la red, ya que siendo un 51.2% es considerablemente mayor que la precisión calculada durante el entrenamiento, cuyo valor es 38.18%. Parece evidente que al testar la red la precisión obtenida debe ser menor, o como máximo igual a la del entrenamiento. Sin embargo, esto no es así, probablemente porque los resultados del entrenamiento son tan pobres, que una mala aproximación, por queda más cerca de la realidad de lo que ha quedado el entrenamiento.

A continuación, se expone una gráfica que indica qué valores de precisión corresponden a cada red del conjunto, así como una línea horizontal que representa la media de la precisión que realmente tiene relevancia, la de los test. Es llamativo como la precisión obtenida por el

entrenamiento para cada red es similar al resto, mientras que la precisión obtenida de los test oscila frecuentemente los mejores y los peores valores. Esto parece indicar que en muchos casos dicha precisión o bien depende de la casualidad, de que coincidan más puntos cercanos a los experimentales, o bien del experimento apartado para realizar los test. Es decir, quizás si al iniciar el bucle, el experimento aislado para testar la red es el 11, que quizás tener registrados movimientos más suaves y la red podría, incluso generando un valor constante, estar más cerca de la realidad que por ejemplo si el experimento apartado para el test fuera otro con movimientos más bruscos.

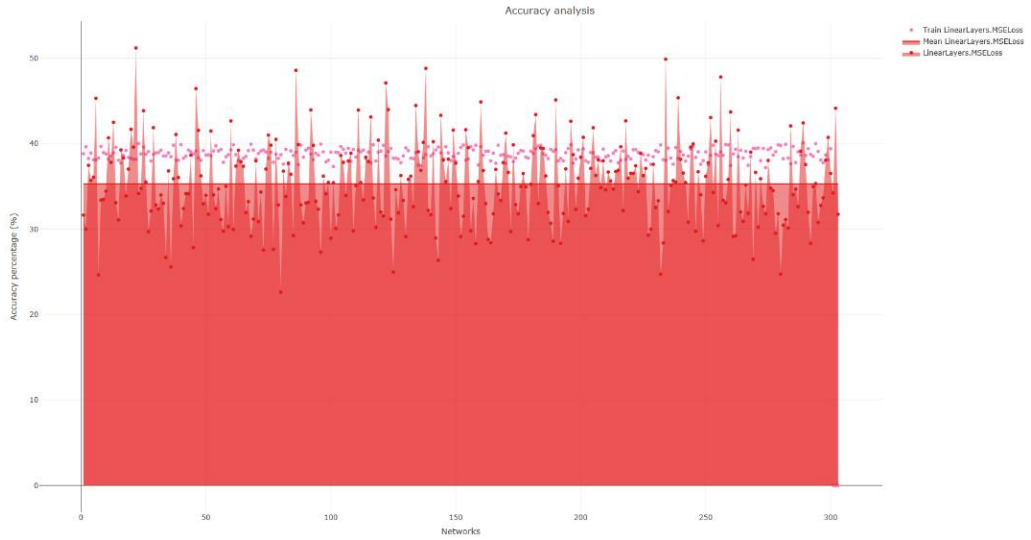

Ilustración 27: Análisis del conjunto 1

En cualquier caso, los resultados eran pobres, por lo que se continuó con otro conjunto diferente de experimentos.

Antes de pasar al siguiente, se expone una gráfica que muestra la distribución de combinaciones de densidades de capas entrenadas durante el experimento, con el fin de mostrar que se está barriendo en cierta manera todas las posibilidades. El tamaño del punto que representa a cada red es proporcional a su precisión calculada a partir del test.

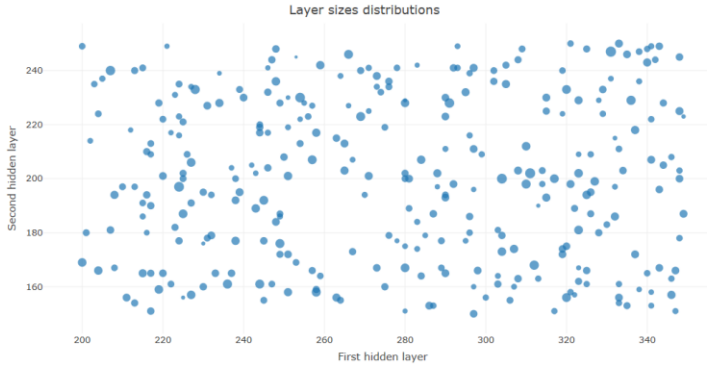

Ilustración 28: distribución tamaños de capa del conjunto 1

SEGUNDO CONJUNTO

Para comprobar si el problema era tener un BS demasiado pequeño, se preparó un experimento exactamente igual, pero duplicando el valor de este parámetro. Esta vez, la mejor red entrenada, logró una precisión en el test del 53.8%.

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Optimization function	Performance Function	Batch size	Net Type	Training Epochs
2	326 linear	200 linear	3 linear	RMSprop	MSE	256	Sequential	100

Tabla 11. Mejor red del conjunto 2

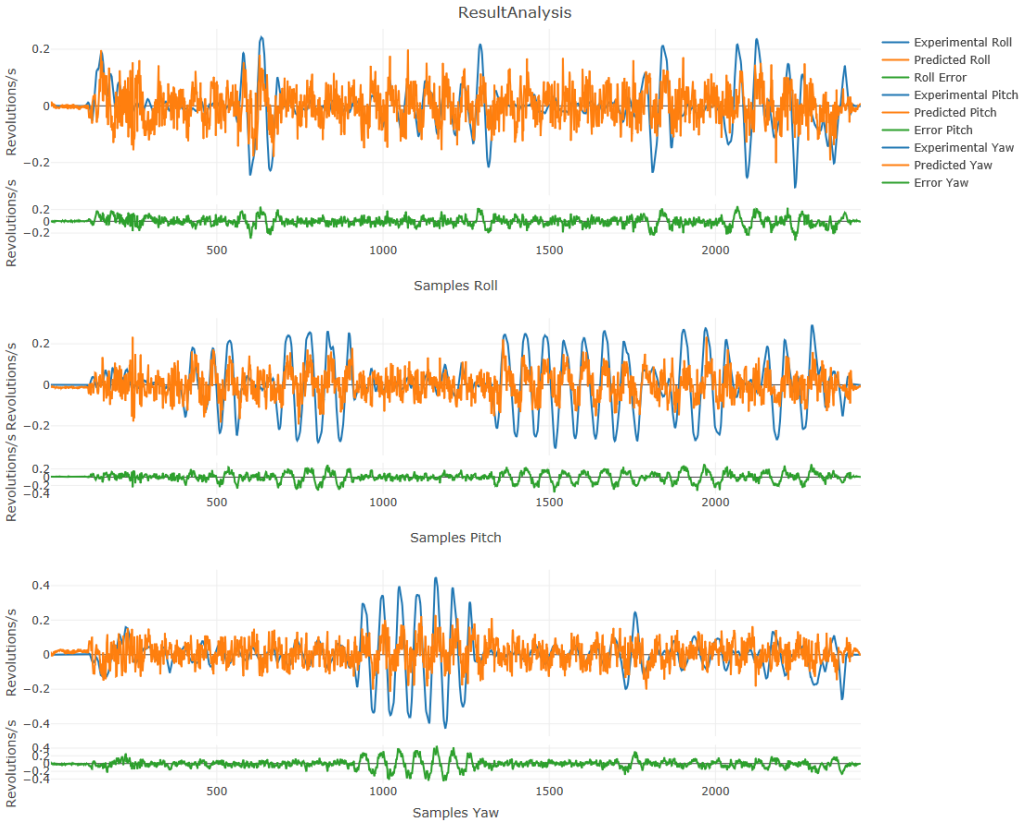

Ilustración 29: Resultados de la mejor red del conjunto 2

Aunque una precisión del 53.8 seguía estando por debajo de las expectativas, si se obtuvo una media mayor de la misma, así como cierta coherencia en las precisiones calculadas durante el entrenamiento de la red, que esta vez eran mucho más altas en comparación con las calculadas a partir del test. Por lo que se puede concluir que los resultados eran mejores y más coherentes. Esto se puede observar en la gráfica que analiza los dos conjuntos de experimentos realizados, la cual se encuentra a continuación.

Además, esta vez sí se vio como la salida generada por la red sigue todas y cada una de las oscilaciones observadas experimentalmente. Existe mucho ruido, y las oscilaciones más altas no son seguidas todavía, sin embargo, se había dado un paso muy grande en comparación con otras simulaciones.

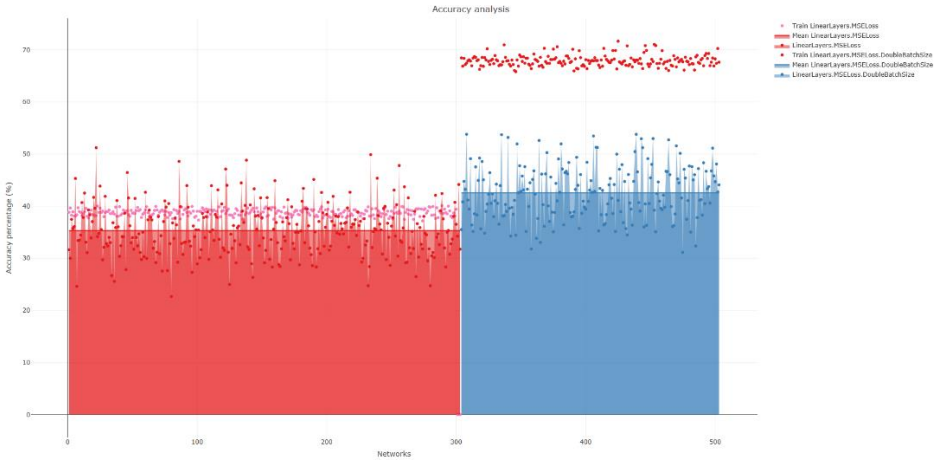

Ilustración 30: Análisis hasta el conjunto 2

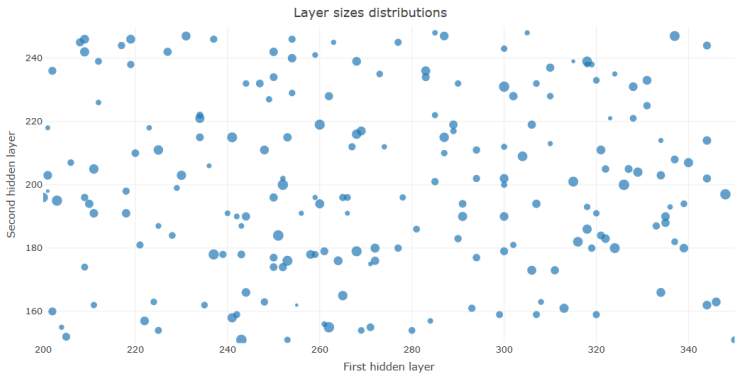

Ilustración 31: distribución tamaños de capa del conjunto 2

TERCER CONJUNTO

Una vez realizados dos conjuntos de experimentos con redes cuya función de activación es lineal, se pasó a realizar uno cuyas funciones sean tanh. Se modificaron el número de epochs, dado que convergía más y durante más epochs que los experimentos anteriores, y el BS, que daba mejores resultados para un valor de 128.

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Optimization function	Performance Function	Batch size	Net Type	Training Epochs
2	407 tanh	103 tanh	3 linear	RMSprop	MSE	128	Sequential	200

Tabla 12. Mejor red del conjunto 3

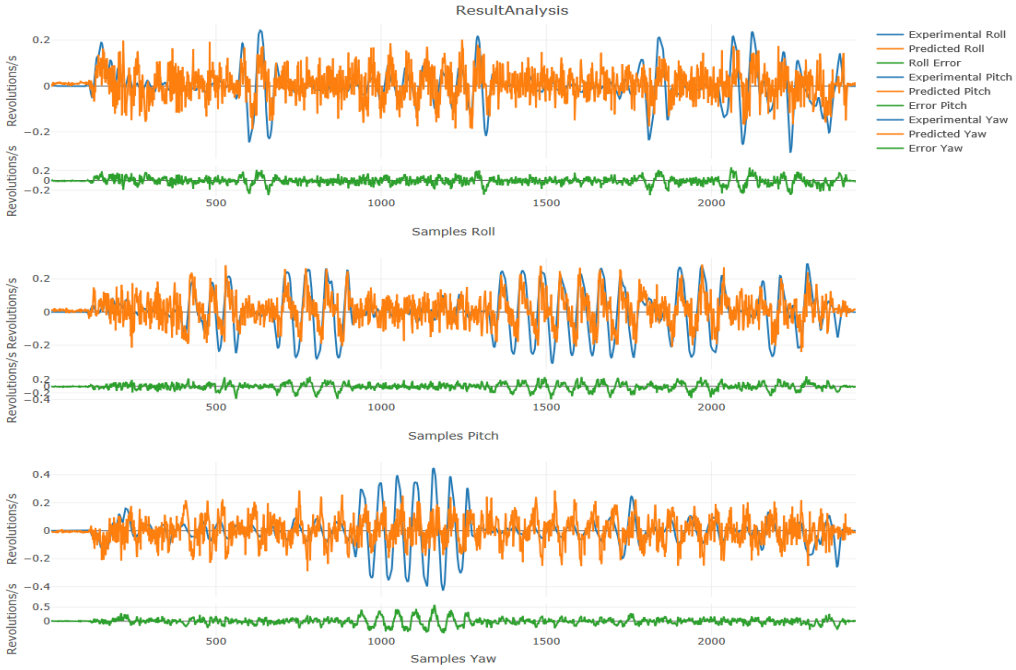

Ilustración 32: Resultados de la mejor red del conjunto 3

Este conjunto resultó ser mejor que el anterior. La mejor red encontrada obtuvo un índice de precisión en el test del 57.12%.

Cada vez parecía que la red era más capaz de seguir las oscilaciones. De nuevo la precisión había aumentado y de nuevo la media de la misma también lo ha hecho. Aunque esta vez, la precisión alcanzada mediante el entrenamiento de la red no era superior a la del experimento anterior. Lo cual era extraño, viendo los resultados sobre el test. Todo esto se puede apreciar en la gráfica adjunta a continuación.

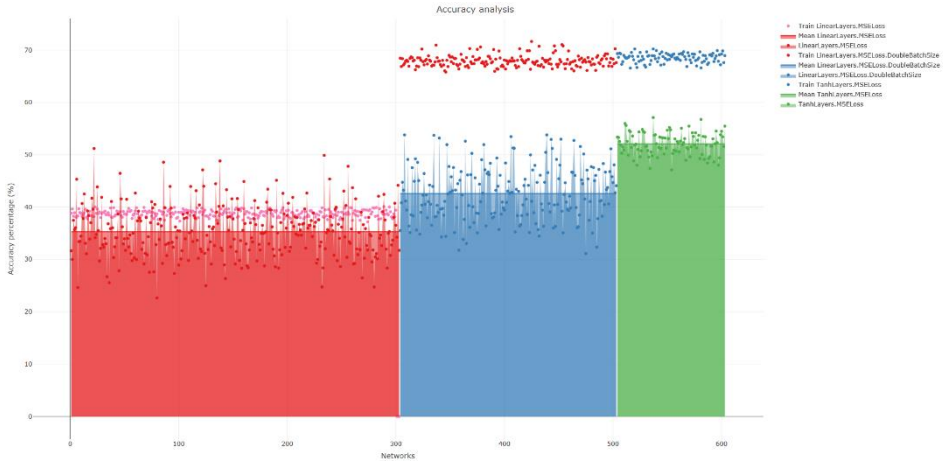

Ilustración 33: Análisis hasta el conjunto 3

En cualquier caso, se volvió a lograr un avance. Cada vez se estaba utilizando un número menor de iteraciones del bucle, lo cual da lugar a cada vez menos redes testadas. Esto se hizo así debido

a que no se detectaban cambios muy significativos en los resultados dependiendo del número de neuronas en cada capa, hasta aquel momento, los factores más importantes para lograr buenos resultados eran el BS, el número de "epochs" y las funciones de activación.

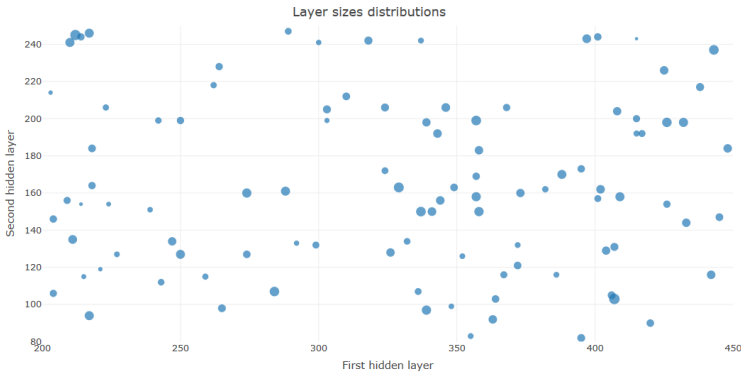

Ilustración 34: distribución tamaños de capa del conjunto 3

CUARTO CONJUNTO

Una vez probadas las redes con funciones de activación lineal y tanh con la función de cálculo del error MSE, se precisaba probar las mismas redes calculando el error mediante la función MAE para comprobar si así se podían obtener mejores resultados. Por lo que se comenzó por configurar un conjunto de redes igual que el anterior.

La mejor red encontrada en este conjunto obtuvo un índice de precisión del 56.5%.

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Optimization function	Performance Function	Batch size	Net Type	Training Epochs
2	223 tanh	206 tanh	3 linear	RMSprop	MAE	128	Sequential	200

Tabla 13. Mejor red del conjunto 4

Pese a que los resultados volvieron a ser buenos, y a que la precisión calculada a partir del entrenamiento seguía estando entre los mismos valores que en los experimentos anteriores, se detectó un descenso en la media de las precisiones calculadas a partir del test. Lo cual hizo pensar que, aunque experimentos aislados tuvieran un buen rendimiento, de media, esta configuración para las redes ofrece prestaciones más bajas que la probada anteriormente.

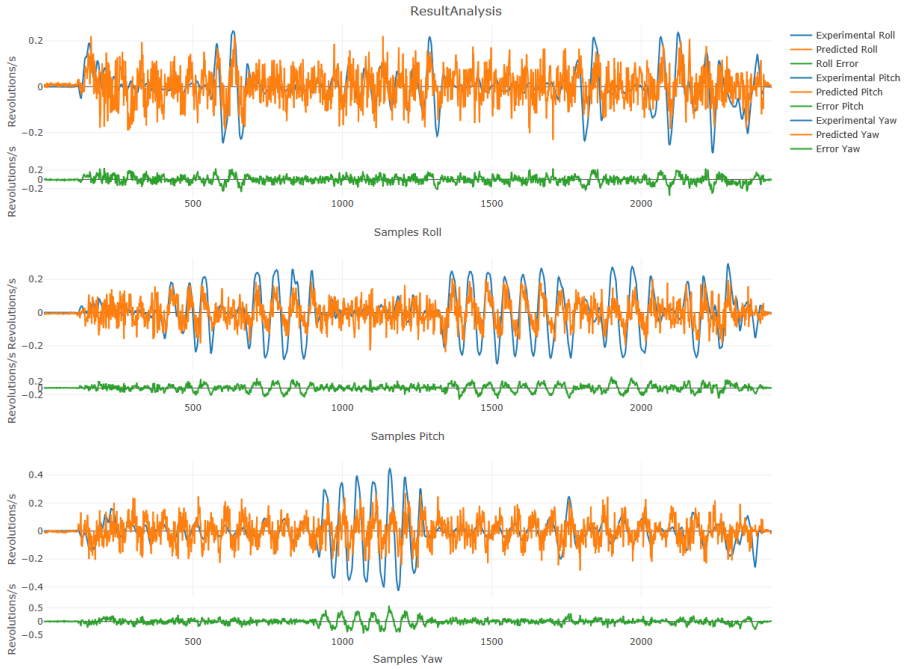

Ilustración 35: Resultados de la mejor red del conjunto 4

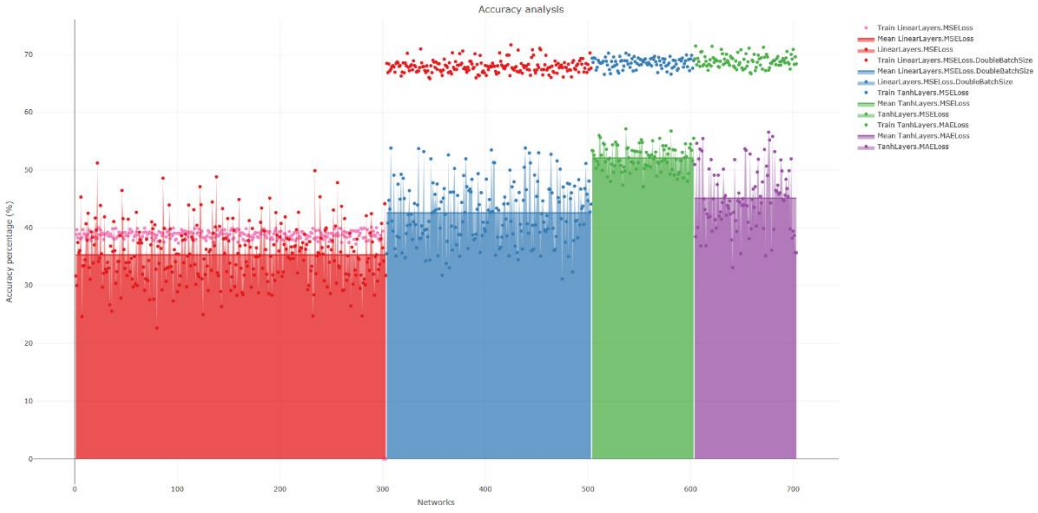

Ilustración 36: Análisis hasta el conjunto 4

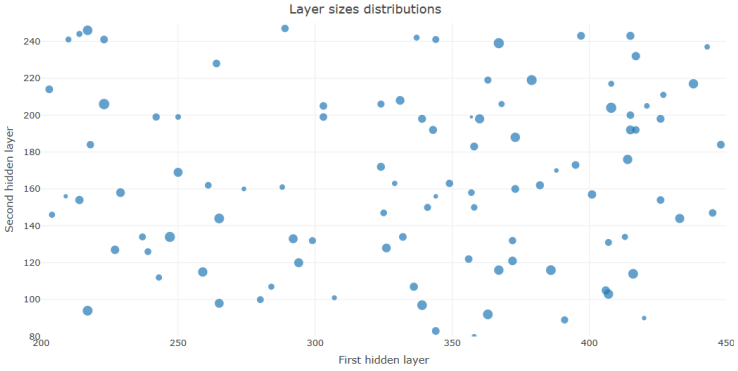

Ilustración 37: distribución tamaños de capa del conjunto 4

QUINTO CONJUNTO

En este paso se repetiría el experimento anterior, pero utilizando funciones de activación lineales para las capas ocultas.

La mejor red encontrada en este conjunto obtuvo un índice de precisión del 44.1%

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Optimization function	Performance Function	Batch size	Net Type	Training Epochs
2	401 linear	157 linear	3 linear	RMSprop	MAE	128	Sequential	200

Tabla 14. Mejor red del conjunto 5

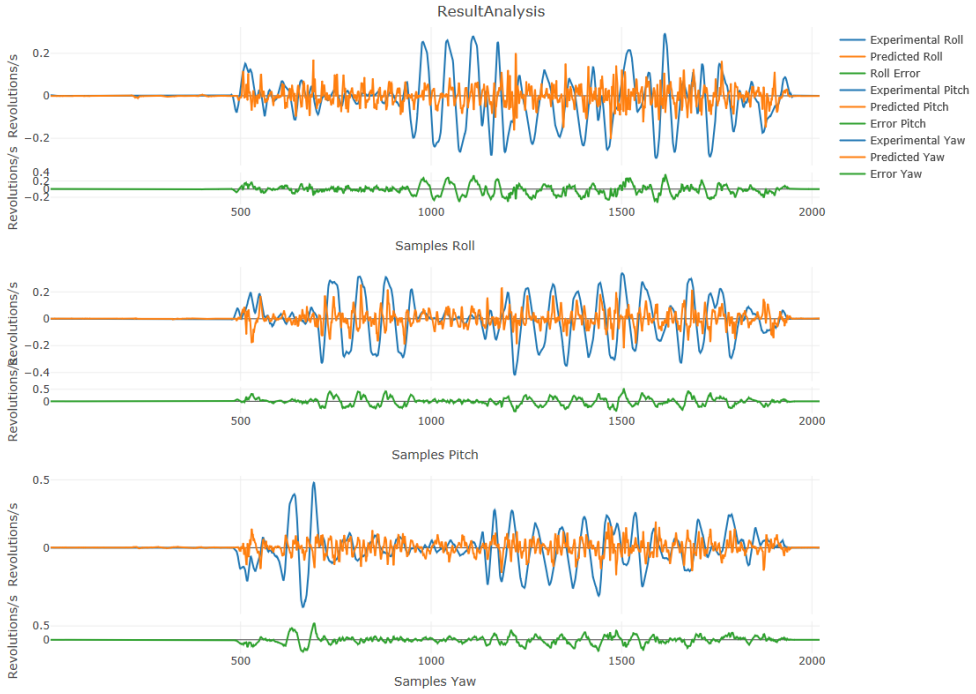

Ilustración 38: Resultados de la mejor red del conjunto 5

En este caso se volvió a dar un paso atrás. Con la función MAE se había detectado un descenso importante de la precisión con redes que utilizaban tanh como función de activación, por lo que, en redes con función lineal, se predecían peores resultados. Sin embargo, no se esperaba cosechar un rendimiento aún más deficiente que el encontrado en la primera tanda de experimentos. De esta forma, se pudo concluir que el cálculo absoluto del error había bajado drásticamente las prestaciones de las redes hasta ese momento testadas.

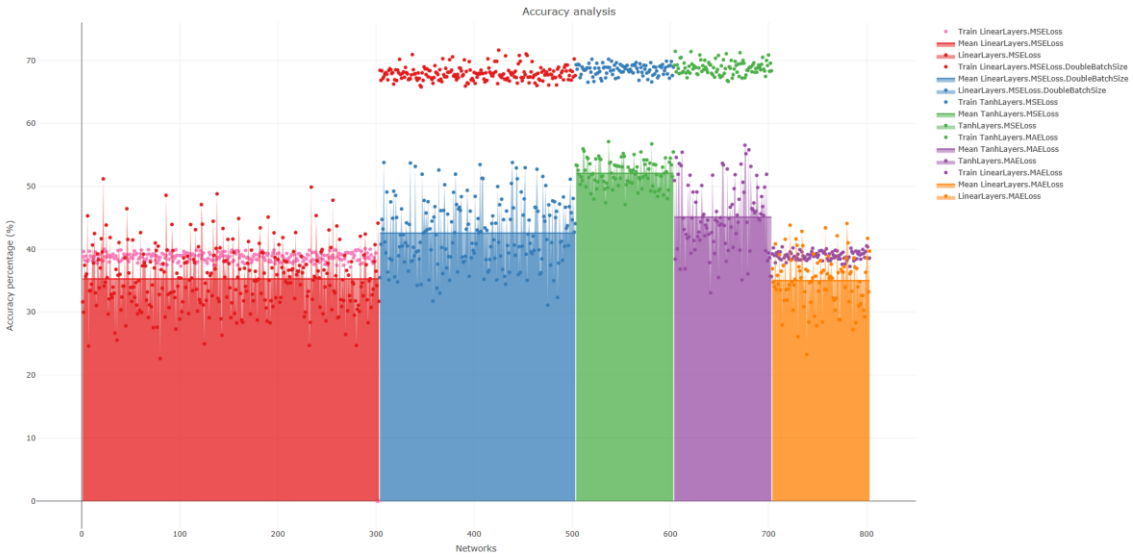

Ilustración 39: Análisis hasta el conjunto 5

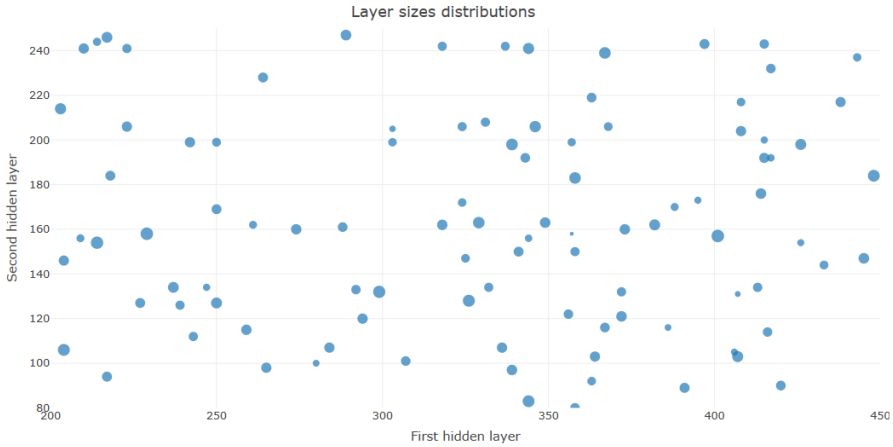

Ilustración 40: distribución tamaños de capa del conjunto 5

Hasta ese momento se habían llevado a cabo 5 conjuntos de experimentos, en los cuales se han probado redes para dos diferentes funciones de activación, para dos funciones de cálculo del error, y para diferentes valores de BS y número de epochs.

El número total de redes entrenadas era de 800. Y a partir de ellas se había concluido que la función de activación con un mejor funcionamiento es la tanh, y que el cálculo del error debía calcularse con la función MSE. Tal y como se hizo en el tercer conjunto de experimentos, ver *Ilustración 39.*

Al dibujar gráficas con la distribución de tamaños de capa dando a cada punto un tamaño proporcional a la precisión obtenida por la red a la que representa, no solo se pretendía demostrar que se estaba haciendo un buen barrido de las diferentes combinaciones posibles de densidad neuronal por capa. También se pretendía poder, observando la gráfica, ver en qué cuadrante se encontraba la mayor densidad de buenas precisiones, para encontrar alguna correlación entre los buenos resultados y las densidades de capas o la proporción de las mismas. Sin embargo, tal y como se puede apreciar en la gráfica a continuación, correspondiente al tercer conjunto de experimentos realizados, el mejor del tramo analizado en este apartado, dicha correlación no parece existir. Las mejores precisiones están muy distribuidas por todos los cuadrantes del gráfico. Es más, la mejor red. Se encuentra alejada del único área que parece contener una densidad mayor de buenos resultados.

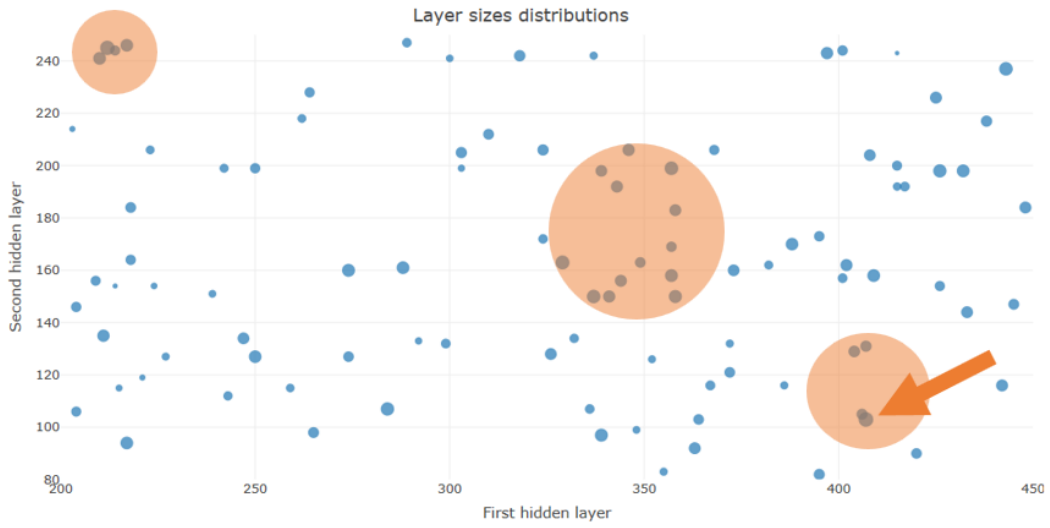

Ilustración 41: distribución tamaños de capa del conjunto 3

La idea era repetir el tercer conjunto de experimentos aumentando el BS, dado que, en teoría al aumentarlo, también lo haría la precisión. Se utilizaría un valor muy alto y se analizarán los resultados.

Con este experimento acabarían los experimentos con redes de dos capas. Desde que se abandonó Matlab como herramienta, no se había probado ninguna red de tres capas, por lo que, con la información ya obtenida, el conocimiento de las mejores configuraciones para montar redes, se creyó conveniente probar a construir redes de 3 capas. Se planeó hacer dos conjuntos de experimentos, uno con función de activación lineal, otro con tanh, pero siempre con un BS y un número de epochs muy sobredimensionado, dado que ya no son experimentos orientativos, sino que se desea exprimir cada entrenamiento al máximo.

Sin embargo, al ver los resultados del conjunto de experimentos con dos capas ocultas y función de activación tanh, se añadió un último conjunto de experimentos, con 3 capas ocultas y la misma función de activación, pero con un BS menor, de 128, al igual que en el tercer conjunto de experimentos. Las razones que llevaron a hacer esto se explicarán en el análisis del sexto conjunto de experimentos.

Para realizar los conjuntos de experimentos con tres capas ocultas se reciclará el código del script maestro añadiendo una tercera capa cuyo número de neuronas será la mitad que el número de neuronas aleatorio que se genere para la segunda capa.

Conjunto	Iteraciones	Función activación	Función Error	Epochs	Batch Size
6	100	Tanh	MSE	300	512
7	100	Linear	MSE	300	512
8	100	Tanh	MSE	300	512
8	100	Tanh	MSE	300	128

Tabla 15. Set de experimentos finales

EXPERIMENTACIÓN Y ANÁLISIS TANH

La mejor red encontrada en este conjunto obtuvo un índice de precisión del 41.8%

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Optimization function	Performance Function
2	324 tanh	249 tanh	3 linear	RMSprop	MSE

Tabla 16. Mejor red del conjunto 6

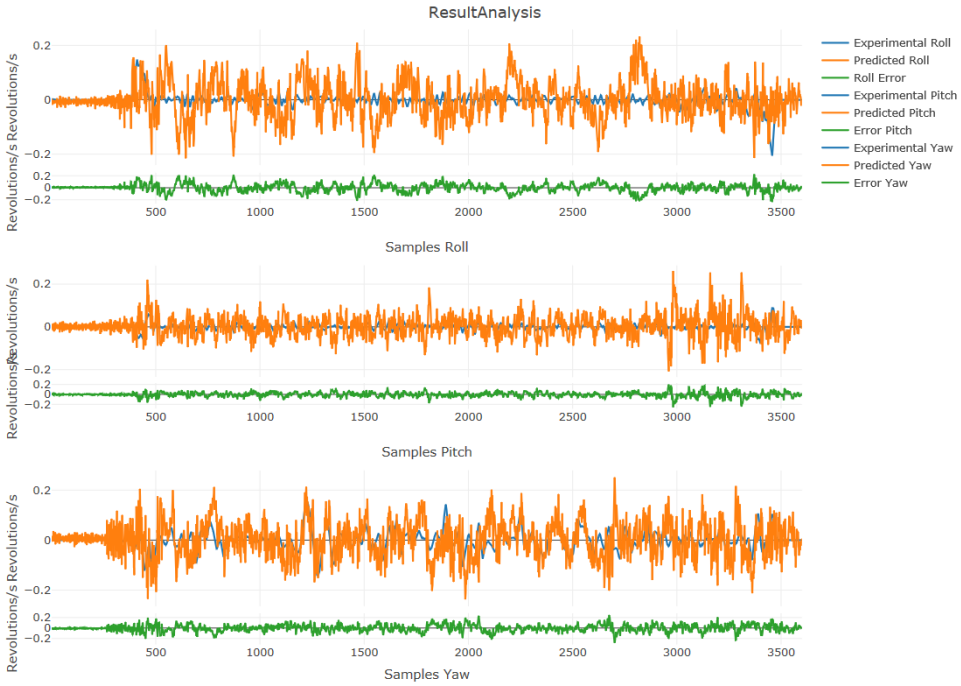

Ilustración 42: Resultados de la mejor red del conjunto 6

Los resultados de este conjunto de redes fueron de lo más inesperados. Salvo por el BS, que esta vez era mucho mayor, todos los demás parámetros eran iguales a los del tercer conjunto. Se esperaban precisiones mucho mejores, sobre todo viendo como las precisiones calculadas durante el entrenamiento sobrepasaban siempre el 70%. Esto parecía indicar que se había dado un claro caso de sobre-ajuste. La red se ha ajustado tanto a los datos del entrenamiento que no es capaz de predecir correctamente valores diferentes a los conocidos.

Esto provocó que se dudara de los resultados que saldrían en los dos próximos conjuntos, cuyo BS es también muy alto. Fue aquí cuando se decidió hacer un noveno experimento, el tercero de tres capas, con el BS de nuevo pequeño, para evitar el sobre-ajuste.

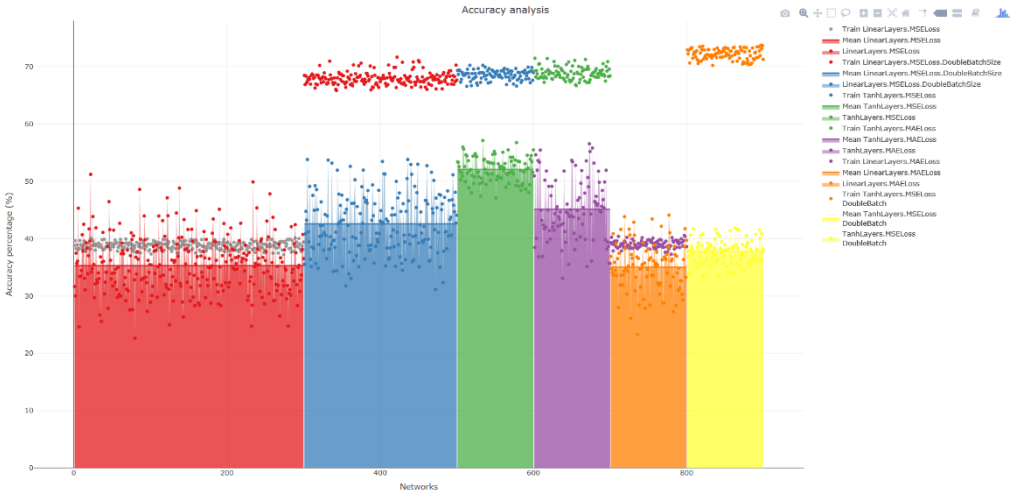

Ilustración 43: Análisis hasta el conjunto 6

EXPERIMENTACIÓN CON REDES DE 3 CAPAS OCULTAS Y ANÁLISIS

La mejor red encontrada en este conjunto obtuvo un índice de precisión del 57.3%

FUNCIÓN ACTIVACIÓN LINEAL

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Optimization function	Performance Function
3	420 linear	141 linear	71 linear	3 linear	RMSprop	MSE

Tabla 17. Mejor red del conjunto 7

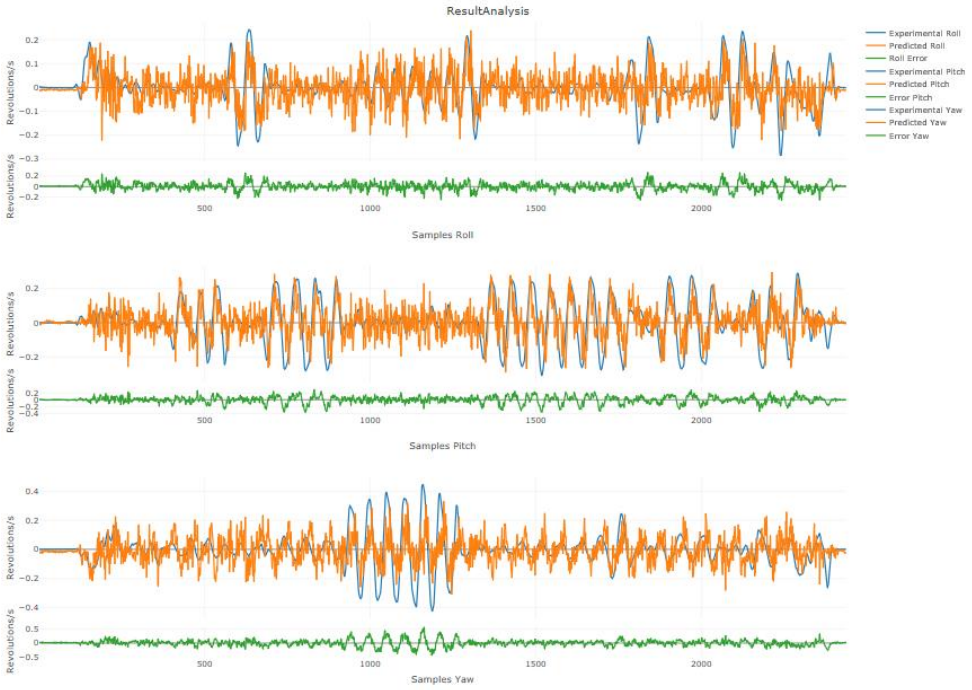

Ilustración 44: Resultados de la mejor red del conjunto 7

Los resultados de este conjunto de experimentos fueron bastante buenos, algo mejores pero muy parecidos a los obtenidos con su red homóloga de dos capas, el segundo conjunto. Esta mejoría es tan pequeña que habría que valorar a nivel computacional si merece la pena tener una red con una capa más y por tanto muchas más neuronas, pero no es ese el objetivo actualmente.

Tras haber analizado el conjunto anterior, las expectativas para el actual eran muy bajas. Fue una sorpresa haber logrado precisiones de más del 50%.

Ilustración 45: Análisis hasta el conjunto 7

FUNCIÓN ACTIVACIÓN TANGENCIAL HIPERBÓLICA

La mejor red encontrada en este conjunto obtuvo un índice de precisión del 57'9%, la mejor hasta ese momento.

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Optimization function	Performance Function
3	433 tanh	239 tanh	120 tanh	3 tanh	RMSprop	MSE

Tabla 18. Mejor red del conjunto 8

Hasta el momento solo se habían obtenido buenos resultados utilizando un BS alto con redes cuya función de activación era lineal. Esta vez, y de nuevo de forma inesperada, se obtuvieron precisiones altas en comparación con el resto teniendo una configuración para la red exacta a la del sexto conjunto, pero con una capa más y pese a tener un BS mayor.

Como no era posible saber si el hecho de que este parámetro fuera mayor estuviera haciendo daño a la precisión calculada durante el test de la red o no, se continuó con la idea de llevar a cabo un conjunto más de experimentos, replicando a este, con el BS menor.

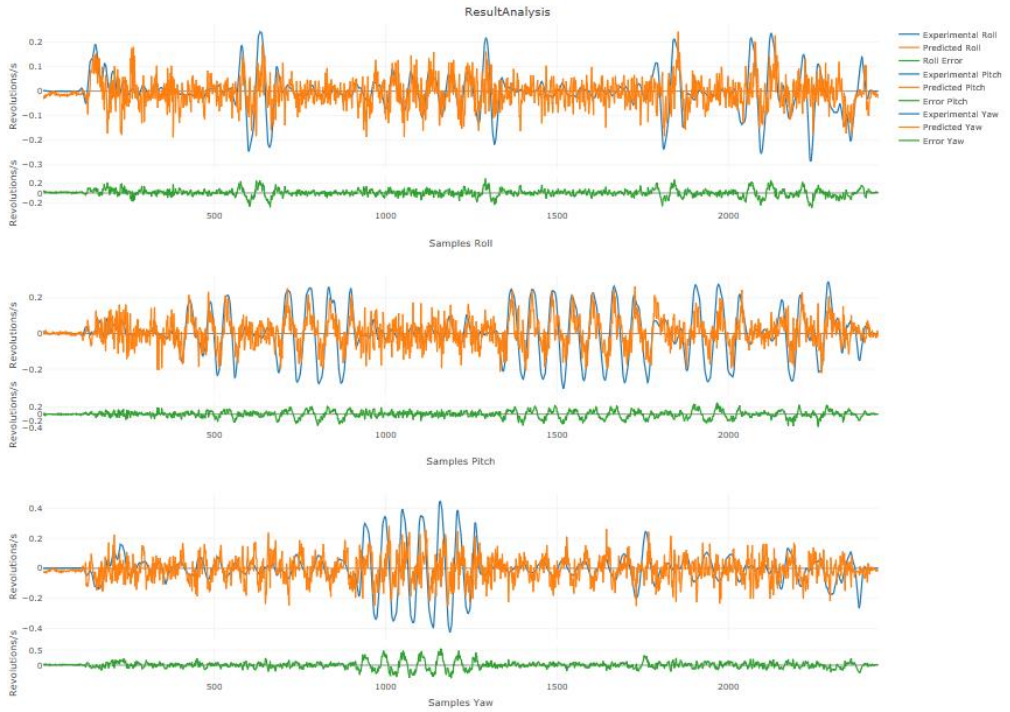

Ilustración 46: Resultados de la mejor red del conjunto 8

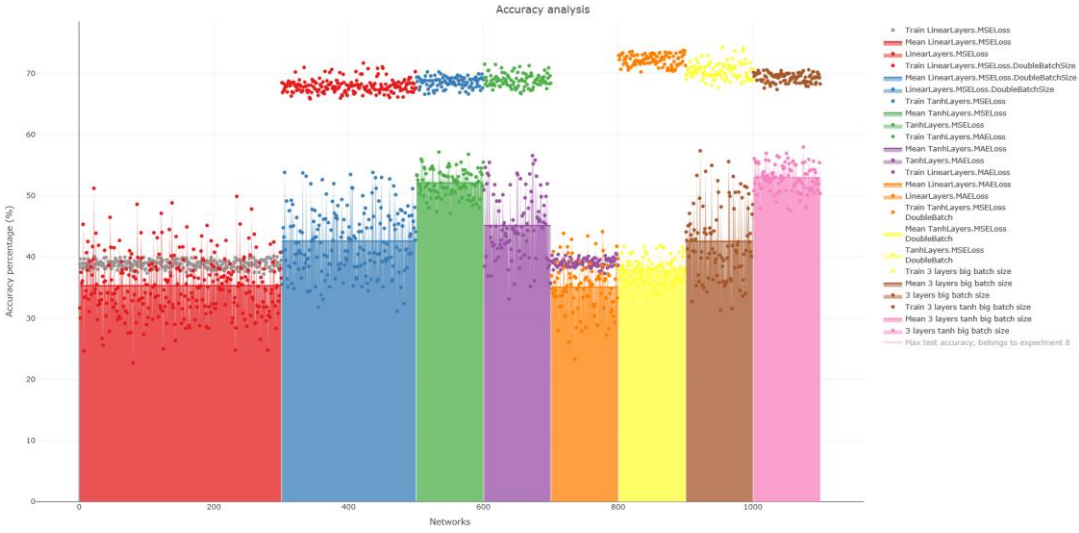

Ilustración 47: Análisis hasta el conjunto 8

FUNCIÓN ACTIVACIÓN TANGENCIAL HIPERBÓLICA 2

La mejor red encontrada en este conjunto obtuvo un índice de precisión del 59'99%, la mejor hasta el momento.

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Optimization function	Performance Function
3	432 tanh	198 tanh	99 tanh	3 tanh	RMSprop	MSE

Tabla 19. Mejor red del conjunto 9

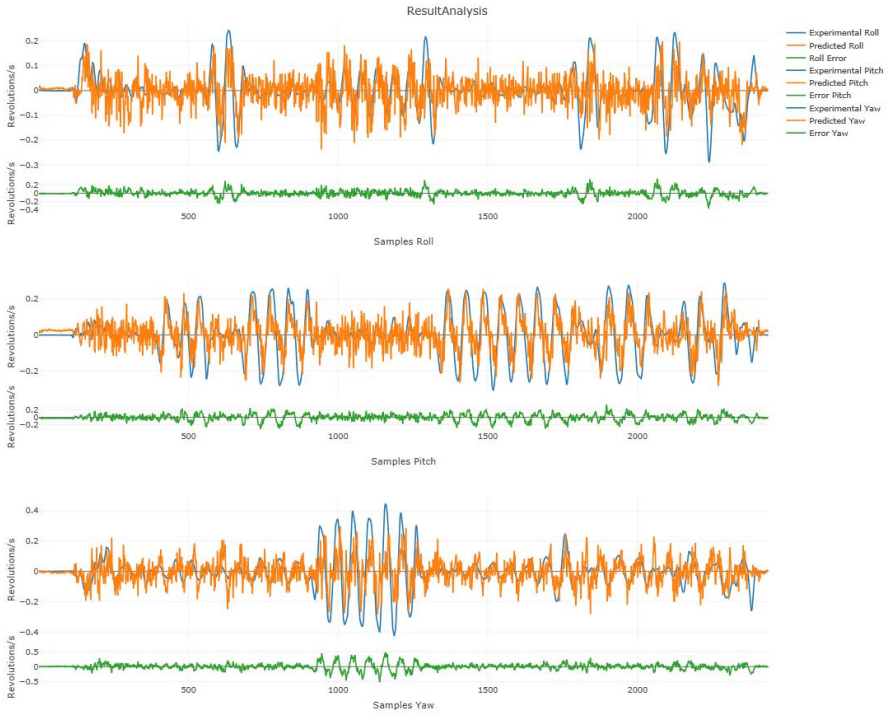

Ilustración 48: Resultados de la mejor red del conjunto 9

Ilustración 49: Análisis hasta el conjunto 9

Los mejores resultados hasta ese momento fueron los de este conjunto, fueron muy parecidos a los del octavo y tercero, pero lograba varias redes cuya precisión superaba en varios puntos a las mejores del resto de conjuntos.

PASO 6: POST PROCESAMIENTO

Como post procesamiento se creyó oportuno añadir un filtro de suavización. Dicho filtro se

aplicará mediante la función “smooth” de R.

La idea del filtro de suavización vino tras ver las salidas de las redes. Los incrementos entre punto y punto son muy agresivos, pero la silueta que se intuye tras ellos es igual o muy parecida a la forma de la señal que se pretendía imitar. Por lo que quizás, aplicando dicho filtro, aparecería limpia la forma que estamos buscando.

COMPARACIÓN

Como se puede apreciar en la *Ilustración 50*, al aplicar el filtro queda una forma mucho más reconocible y comparable a simple vista con la versión experimental de los resultados del experimento. Concretamente este experimento es el número 11 de los 14 disponibles para desarrollar el proyecto.

En la imagen, puede verse el conjunto de gráficas sin post procesamiento a la izquierda, enmarcada en azul, mientras que la gráfica en la que se aplica el filtro de suavización se encuentra a la derecha.

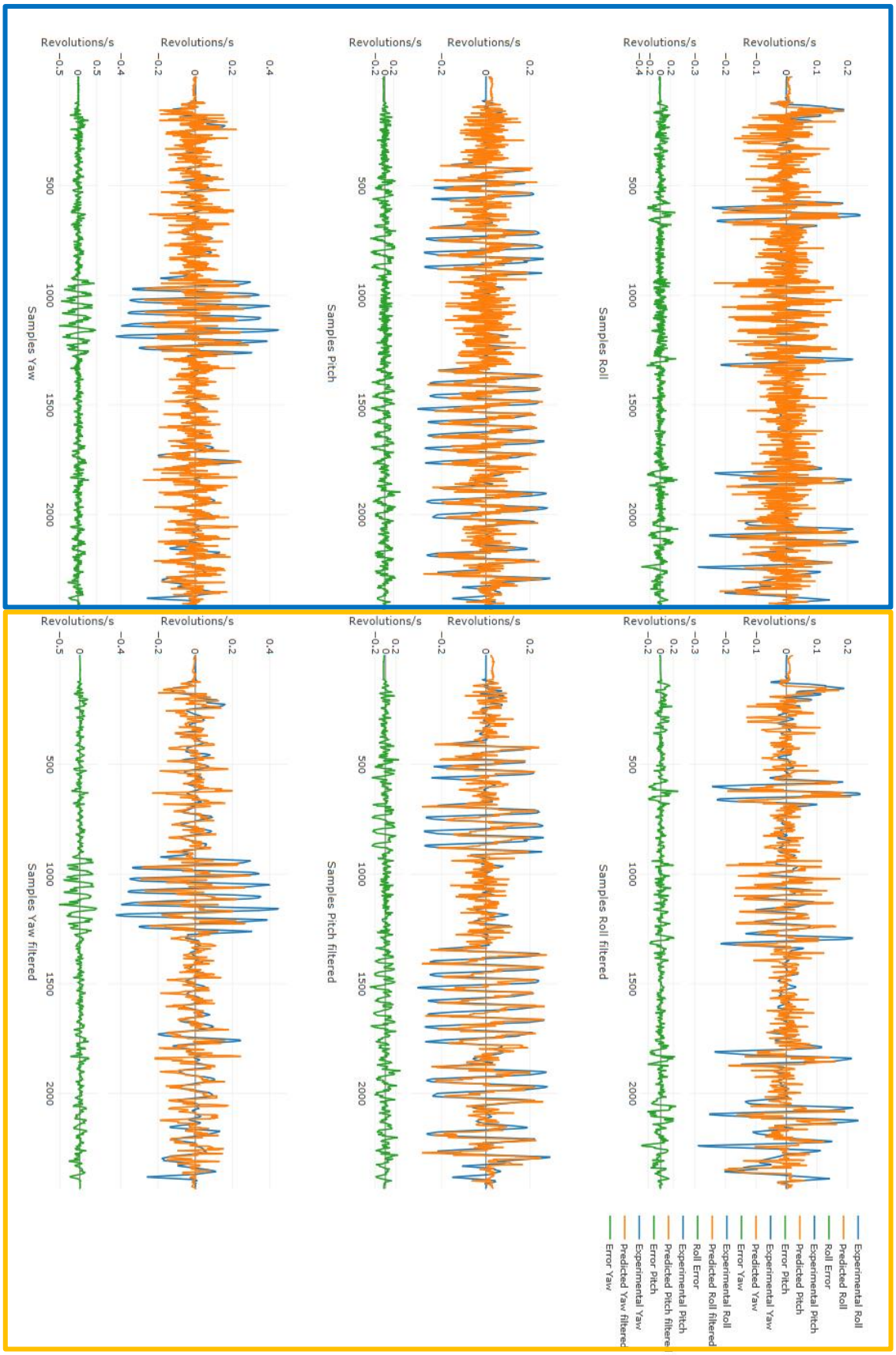

Ilustración 50: Comparación filtro post procesamiento

PASO 7: CONCLUSIÓN

En el presente capítulo se han llevado a cabo 9 conjuntos diferentes de experimentos con características diferentes, así como un total de 1200 redes neuronales con el cometido de hacer un estudio que ayude a encontrar el tipo de red más conveniente para realizar la red necesaria para lograr cumplir el objetivo del proyecto.

Para ello se ha realizado un paquete de funciones para R para facilitar la automatización y estudio de todo el proceso. Así como un entorno de trabajo dentro del proyecto que se amplía de forma automática a medida que se van llevando a cabo nuevos análisis y conjuntos de experimentos.

Como resultado se han obtenido una serie de resultados que, siendo infinitamente mejores que los hallados en Matlab, siguen sin ser tan buenos como se pretendía en un principio.

La conclusión del estudio del conjunto global indica que la mejor manera de alcanzar las más altas prestaciones posibles utilizando redes neuronales secuenciales es mediante el uso de 3 capas ocultas cuyas funciones de activación fueran tanh. Seguidas de cerca por las redes con 3 capas ocultas con funciones de activación lineales, y las redes con 2 capas ocultas con funciones de activación tanh.

El mejor resultado hallado ha sido el analizado en el apartado anterior, el cual ha podido ser mejorado con un post procesamiento basado en un filtro de suavización, ver *Ilustración 50*, y cuyas características son las siguientes:

Capas ocultas	Primera capa	Segunda capa	Tercera capa	Cuarta capa	Optimization function	Performance Function
3	432 tanh	198 tanh	99 tanh	3 tanh	RMSprop	MSE

Tabla 20. Mejor red del análisis global

El gran protagonista del estudio ha sido el BS. Según la teoría siempre que aumenta, aumentará la precisión de la red entrenada a costa de un uso mayor de la memoria disponible. Y aunque pueda parecer lo contrario, ha sido así.

Pese a que los resultados dando a este parámetro un valor mayor han sido en muchos casos peores, sobre todo para las redes que utilizaban funciones de activación tanh, ha cumplido con su cometido, que es lograr mayores valores de precisión durante el entrenamiento de la red. Tal como se ve en la gráfica adjunta a continuación:

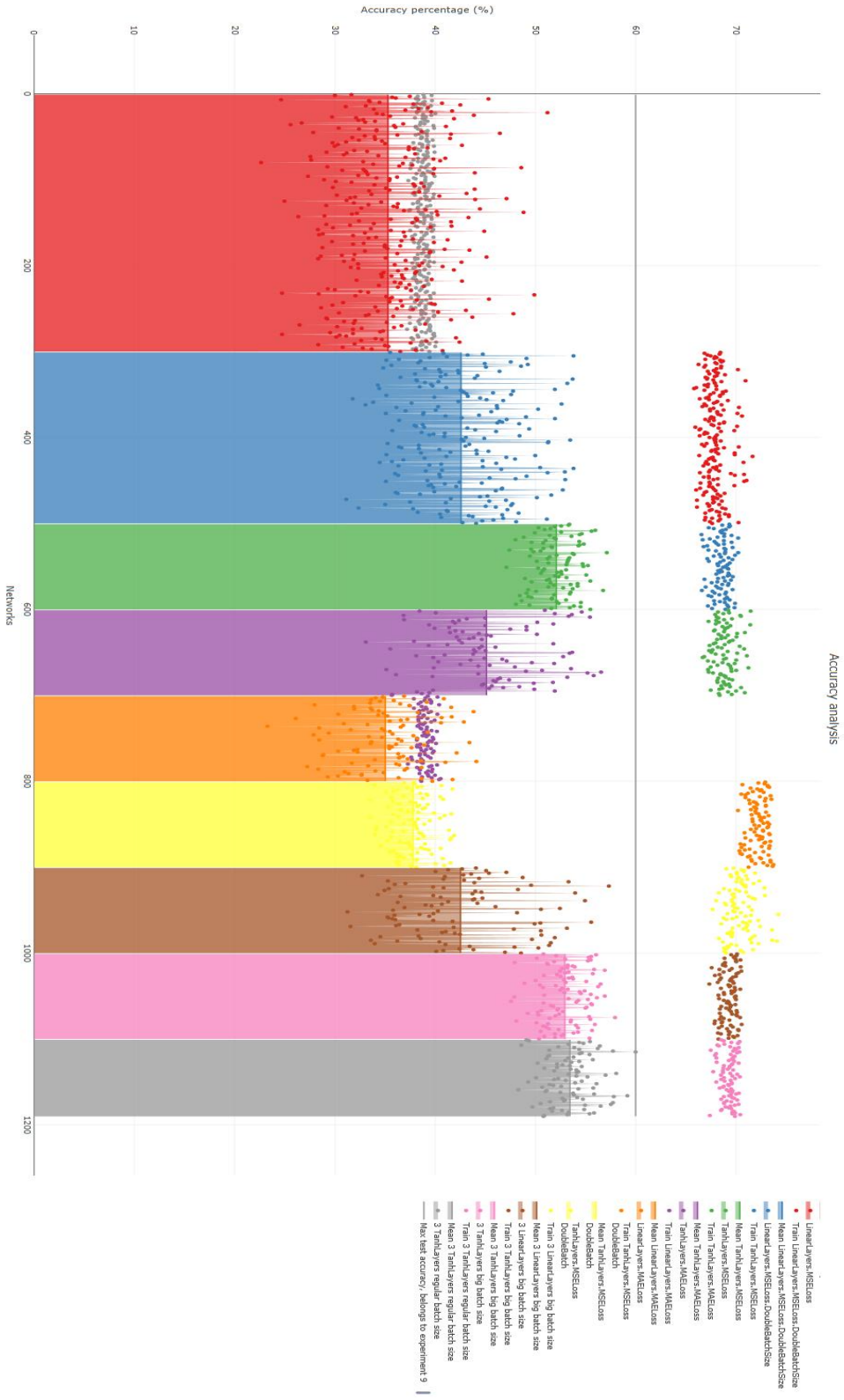

Ilustración 51: Análisis final del estudio

Si se introdujera como entrada de las redes entrenadas en esos conjuntos, un grupo de imágenes utilizado para entrenar la red, el resultado sería muy bueno. Esto ocurre porque, la red se ha entrenado de manera que es capaz de lograr el máximo de precisión posible sobre los datos con

los que ha sido entrenada, lo cual la hace "torpe" ante experimentos que no sean tan parecidos a los conocidos por la red.

Este fenómeno siempre ocurre en mayor o menor medida, por eso para realizar un test de cada red, se ha dejado a parte una parte del conjunto de datos. Para ilustrar esta circunstancia, se adjunta la gráfica a continuación:

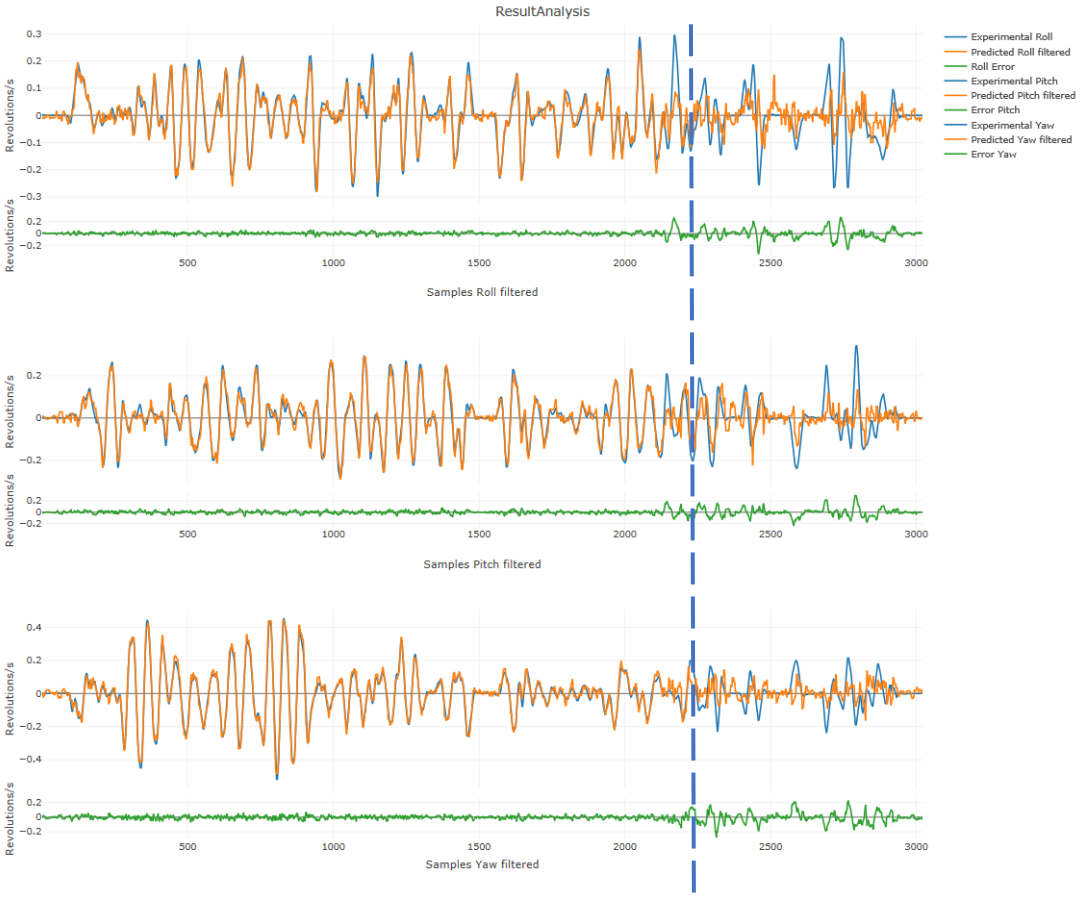

Ilustración 52: Comparación datos entrenados vs datos validación

En esta imagen se puede ver el experimento 12, el cual está al límite entre los datos utilizados para el entrenamiento y la validación de una red del conjunto 6, que tiene las precisiones calculadas a partir del test más bajas, así como las precisiones calculadas a partir del entrenamiento más altas. Ni siquiera habría falta señalar dónde está dicho límite.

Quizás, con un conjunto mucho mayor de datos, se lograría entrenar la red de manera que fuera capaz de lograr resultados como el que se ve en la primera parte de la gráfica, pero esto carecería de sentido, ya que el propósito de una red neuronal es fabricar un modelo, no memorizar una tabla con todas las combinaciones posibles.

Y este es el fin del capítulo dedicado al desarrollo con KerasR, gracias al cual se han logrado alcanzar conclusiones y resultados lo suficientemente completos para reflejar todas las horas de trabajo empleadas durante el transcurso del proyecto.

CONCLUSIÓN DEL PROYECTO

A lo largo del proyecto se ha intentado llevar a cabo una red neuronal capaz de interpretar imágenes de tres cámaras de baja resolución orientadas de una manera determinada, en los ángulos de Euler de la estructura que los sujeta.

Para lograrlo, se han utilizado dos herramientas diferentes, Matlab y KerasR.

MATLAB

En Matlab se han probado:

- Multitud de funciones de entrenamiento
- Varias combinaciones de densidad de capas
- Dos funciones de activación diferentes
- Redes de 2 y 3 capas ocultas
- Redes en las que no se intenta predecir el yaw, para comprobar si es cierta una teoría ideada sobre la capacidad de la red para estimar el mismo

Pero el resultado ha sido deficiente pese al esfuerzo realizado. Los resultados dejaban mucho que desear y toda la intención y trabajo aplicado iban cayendo en un saco sin fondo.

La única conclusión que se pudo sacar fue que o bien la red que se estaba intentando realizar no era posible de hallar, o que no se estaba siendo capaz de comprender y aprovechar toda la potencia de la herramienta, o que el mismo Toolbox no tenía la capacidad suficiente para afrontar una red como la que precisaba el proyecto.

KERASR

Por otro lado, en KerasR se han realizado 9 conjuntos diferentes de experimentos, en los cuales se han probado:

- Dos funciones de activación distintas
- Dos funciones de cálculo del error

- Varios valores para el BS
- Redes de 2 y 3 capas ocultas
- Cientos de combinaciones de densidades neuronales para las distintas capas

Como resultado, se ha podido llevar a cabo un estudio bastante amplio del funcionamiento de las distintas configuraciones con el objetivo de construir una red lo más útil posible para lograr los resultados deseados en el presente proyecto. Así como un paquete de funciones creado desde cero específicamente para este proyecto.

CONCLUSIÓN GLOBAL

No es posible saberlo ya que no se dispone de semejante set de datos, pero no hay nada que indique que, con un entrenamiento más intensivo, la red no sea capaz de lograr mejores resultados, resultados como los que se pueden observar cuando se testa la red con datos que se han utilizado para el entrenamiento. Sin embargo, observando los resultados se intuye que quizás ese 70% de precisión adquirida durante el entrenamiento, es un límite propio de el tipo de red utilizado. Es decir, da lugar a pensar que más allá de los datos utilizados, quizás una red de este tipo es incapaz de lograr mejores resultados que los obtenidos.

En cualquier caso, aunque la red encontrada en sí misma no sea una solución lo suficientemente buena para calcular los ángulos de Euler a partir de imágenes, quizás si podría valer para integrarla en un sistema mayor de sensores, como lo es la IMU a la que se pretendía sustituir, para contribuir a la disminución de los errores de la misma.

Más allá de la utilidad del estudio realizado y de la red encontrada para el proyecto, todo el proceso de investigación, programación de funciones y scripts, y la utilización de NNTool y KerasR para la programación de redes neuronales, ha resultado ser un viaje fascinante y una gran oportunidad para comenzar a adentrarse en el mundo de la inteligencia artificial.

Tanto el conocimiento, como la experiencia adquirida a través de la realización del presente proyecto, me ha llevado a continuar estudiando las distintas posibilidades que ofrece el mundo de la inteligencia artificial para la solución de todo tipo de problemas. Y ha propiciado la continuación de mis estudios académicos dentro de esta rama de la ingeniería.

BIBLIOGRAFÍA

- HOWARD DEMUTH, MARK BEALE. *Neural Network Toolbox For Use with Matlab*. 4. Detroit: 2002
- DIEGO FERNANDO SOCHA GARZÓN, GILBERTO ANTONIO ORTIZ HERRADA. *Aplicación de Redes Neuronales MLP a la Predicción de un Paso en Series de Tiempo.* Bogotá: Pervis Rengifo Rengifo 2005
- Las Redes Neuronales, qué son y por qué están volviendo. Xataka. 21 enero 2016, 12:00 [consulta: 5 mayo 2018]. Disponible en: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo
- Neural Networks, Manifolds and Topology. Github. 6 abril 2014. [consulta: 20 agosto 2017]. Disponible en: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
- Broyden–Fletcher–Goldfarb–Shanno_algorithm, Wikipedia: la enciclopedia libre. 3 abril 2018, 13:56 [consulta: 30 abril 2018]. Disponible en: https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm
- Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python, Elitedatascience. 9 febrero 2017 [consulta: 1 mayo 2018] Disponible en: https://elitedatascience.com/keras-tutorial-deep-learning-in-python
- Múltiples consultas: Mathworks. 3 julio 2018, 12:43 [consulta: 1 septiembre 2016 - 1 junio 2018]. Disponible en: https://es.mathworks.com
- Múltiples consultas: KerasR. 1 julio 2018, 10:22 [consulta: 30 abril 2018 - 1 junio 2018]. Disponible en: https://keras.rstudio.com
- ¿Qué son los ocelos u ojos simples?: Curiosoando. 19 febrero 2018, 13:45 [consulta: 30 septiembre 2016]. Disponible en: https://curiosoando.com/que-son-los-ocelos-u-ojos-simples
- Órganos de los sentidos en insectos: Insectevc. 27 marzo 2013, 13:12 [consulta: 5 noviembre 2016]. Disponible en: http://insectsevc.blogspot.com/2013/03/organos-de-los-sentidos-en-insectos.html

- Top Lenguajes Programación para Inteligencia Artificial: Adictec. 24 noviembre 2017, 11:17 [consulta: 1 mayo 2018]. Disponible en: https://adictec.com/lenguajes-programacion-para-inteligencia-artificial/