

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS
INFORMÁTICOS



**Predicción de lanzamientos de penalti en LaLiga Santander
mediante el uso de Redes Neuronales**

Trabajo Fin de Grado

Grado en Ingeniería de Computadores

Curso académico 2020-2021

Autor:

Miguel Garrido Carpio

Tutor:

Edgar Talavera Muñoz

1 de julio de 2021

Resumen

Durante estos últimos años se ha podido observar como la tecnología ha llegado al deporte, y más específicamente en el fútbol siendo el deporte rey en España e incluso Europa, gracias a sus recursos económicos y el alcance que tiene en la sociedad actual. Ahora se sabe que los clubes de fútbol analizan todo lo posible por analizar, desde el estado físico de los jugadores, pasando por cómo adaptar los entrenamientos en función de optimizar el rendimiento de cada jugador, hasta ser capaces de analizar los penaltis de los clubes rivales y poder predecir dichos penaltis en función de diferentes características. Esto último de predecir penaltis va a ser lo que se trata en este Trabajo Fin de Grado (TFG).

Desde la creación del *dataset* a partir de recopilar la información de los penaltis, pasando por el análisis de dichos datos, hasta la ampliación de este mediante técnicas de *data augmentation*.

Después, se pasará a la realización de un modelo de *deep learning* usando redes neuronales. El objetivo del modelo será el de predecir si un penalti va a ser gol o no y la dirección donde se lanza, y si un portero va a ser capaz de pararlo o no y hacia qué lado de la portería se va a lanzar buscando parar el penalti. A este modelo se aplicarán diferentes técnicas de reducción de dimensionalidad y técnicas que buscan reducir el sobreajuste en el entrenamiento de la red.

En definitivo, este proyecto consta de dos puntos fundamentales: la creación de un *dataset* y la implementación de una red neuronal funcional.

Abstract

In recent years, it has been observed how technology has arrived in sports, more specifically in football, which is the main sport in Spain and even Europe, thanks to its economic resources and the scope it has in today's society. It is now known that football clubs analyse everything possible to analyse, from the physical condition of the players, to how to adapt the training sessions in order to optimise the performance of each player, to being able to analyse the penalties of rival clubs and being able to predict these penalties based on different characteristics. The latter of predicting penalties will be the subject of this Final Degree Project (TFG).

From the creation of the dataset collecting penalty information, through the analysis of this data, to the extension of the dataset by means of data augmentation techniques.

Then, a deep learning model will be developed using neural networks. The aim of the model will be to predict whether a penalty is going to be a goal or not and the direction in which it is taken, and whether a goalkeeper is going to be able to save it or not and which side of the goal he is going to shoot towards in order to save the penalty. Several dimensionality reduction techniques and techniques that seek to reduce the overfitting in the training of the network will be applied to this model.

In short, this project consists of two fundamental points: the creation of a dataset and the implementation of a functional neural network.

Índice

| | |
|--|-----------|
| 1. Introducción | 1 |
| 1.1 Contexto..... | 1 |
| 1.2 Motivación | 3 |
| 1.3 Objetivos..... | 4 |
| 2. Estado del arte | 5 |
| 2.1 ¿Qué es la Inteligencia Artificial?..... | 5 |
| 2.1.1 Tipos de Inteligencia Artificial | 5 |
| 2.2 Historia de la Inteligencia Artificial..... | 6 |
| 2.2.1 Año 1921 | 6 |
| 2.2.2 Año 1936 | 6 |
| 2.2.3 Año 1941 | 7 |
| 2.2.4 Años 1943 - 1955 | 7 |
| 2.2.5 Año 1956 | 8 |
| 2.2.6 Años 1956 - 1969 | 8 |
| 2.2.7 Años 1970-1980 | 9 |
| 2.2.8 Años 1980-1986 | 9 |
| 2.2.9 Años 1987-2011 | 9 |
| 2.2.10 Desde 2012 hasta la actualidad..... | 10 |
| 2.3 Machine Learning | 10 |
| 2.4 Redes Neuronales Artificiales | 12 |
| 3. Desarrollo del proyecto..... | 16 |
| 3.1 Creación del dataset | 16 |
| 3.1.1 Elección de las características | 17 |
| 3.1.2 Determinación de la dirección del penalti | 18 |
| 3.1.3 Obtención de los datos | 19 |
| 3.1.4 Data augmentation | 19 |
| 3.1.5 Problemática data augmentation | 22 |
| 3.2 Creación de la red neuronal | 22 |
| 3.2.1 Estructura de la red neuronal | 24 |
| 3.2.2 Entrenamiento de la red neuronal | 26 |
| 3.2.3 Evaluación de la red neuronal..... | 27 |
| 3.2.4 Reducción de dimensionalidad de la red neuronal..... | 32 |
| 3.3 Entorno empleado..... | 35 |

| | |
|---|-----------|
| 4. Resultados..... | 36 |
| 4.1 Análisis de los resultados | 36 |
| 4.1.1 Primeros resultados | 36 |
| 4.1.2 Evolución de la red neuronal..... | 41 |
| 4.1.3 Afinando la red neuronal..... | 43 |
| 4.1.4 Modelo con tres capas ocultas | 45 |
| 4.1.5 Modelo con dos capas ocultas..... | 47 |
| 4.1.6 Modelo con una capa oculta..... | 50 |
| 4.1.7 Modelo con dos capas ocultas y dropout..... | 52 |
| 4.2 Modelo definitivo..... | 54 |
| 5. Impacto sociales y medioambientales | 59 |
| 6. Conclusiones..... | 60 |
| 7. Futuros proyectos | 61 |
| Bibliografía | 63 |
| Anexo. Acrónimos y siglas..... | 64 |

Índice de ilustraciones.

| | |
|--|----|
| Ilustración 1. Sistema de cámaras en el Ojo de Halcón [1] | 1 |
| Ilustración 2. Ejemplo de uso del Ojo de Halcón en el tenis [2] | 2 |
| Ilustración 3. Ejemplo del ojo de halcón en el fútbol | 2 |
| Ilustración 4. Konrad Zuse junto a la máquina Z3 | 7 |
| Ilustración 5. Test de Turing | 8 |
| Ilustración 6. Watson ganador de Jeopardy!, venciendo a los dos campeones del juego .. | 10 |
| Ilustración 7. Comparación de una red neuronal biológica y de una red neuronal artificial [11] | 12 |
| Ilustración 8. Representación de las capas de una red neuronal | 13 |
| Ilustración 9. El portero Tim Krul durante un partido entre el Norwitch y el Tottenham | 16 |
| Ilustración 10. Diagrama de flujo de la decisión de scored y saved | 18 |
| Ilustración 11. Portería dividida en nueve zonas. | 19 |
| Ilustración 12. División del dataset en conjunto de entrenamiento y validación..... | 22 |
| Ilustración 13. Resumen del modelo secuencial. | 25 |
| Ilustración 14. Resumen de la red neuronal. | 26 |
| Ilustración 15. Función evaluate..... | 27 |
| Ilustración 16. Función accuracy de las variables scored y saved respectivamente..... | 28 |
| Ilustración 17. Función accuracy de las variables kick_direction y keeper_direction respectivamente. | 28 |
| Ilustración 18. Función loss de las variables scored y saved respectivamente. | 29 |
| Ilustración 19. Función loss de las variables kick_direction y keeper_direction respectivamente. | 29 |
| Ilustración 20. Matriz de confusión y matriz de confusión normalizada de la variable scored..... | 29 |
| Ilustración 21. Matriz de confusión y matriz de confusión normalizada de la variable saved..... | 30 |
| Ilustración 22. Matriz de confusión y matriz de confusión normalizada de la variable kick_direction..... | 30 |
| Ilustración 23. Matriz de confusión y matriz de confusión normalizada de la variable keeper_direction..... | 31 |
| Ilustración 24. Funciones accuracy_score y f1_score..... | 31 |
| Ilustración 25. Varianza de cada característica de entrada y varianza acumulada..... | 32 |
| Ilustración 26. Accuracy y F1_score tras aplicar PCA y utilizar las 9 características principales..... | 33 |
| Ilustración 27. Accuracy y F1_score tras aplicar PCA y utilizar las 2 características principales..... | 33 |
| Ilustración 28. Porcentaje de varianza explicada por cada característica de entrada..... | 33 |
| Ilustración 29. Accuracy y F1_score tras aplicar PCA y utilizar las 8 características principales..... | 34 |
| Ilustración 30. Accuracy y F1_score tras aplicar LDA..... | 34 |
| Ilustración 31. Red neuronal con dos capas de salida. | 37 |
| Ilustración 32. Valores predichos y reales. | 37 |
| Ilustración 33. Accuracy y loss tras 100 epochs de la red neuronal con dos capas de salida. | 39 |
| Ilustración 34. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida. | 39 |
| Ilustración 35. Accuracy y loss tras 100 epochs de la red neuronal con dos capas de salida aplicando PCA y utilizando 9 componentes principales..... | 40 |
| Ilustración 36. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida aplicando PCA y utilizando 2 componentes principales..... | 41 |
| Ilustración 37. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida aplicando LDA. | 41 |

| | |
|--|----|
| Ilustración 38. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida y una tercera capa oculta de 32 neuronas..... | 42 |
| Ilustración 39. Accuracy y loss tras 10000 con dos capas de salida salida y una tercera capa oculta de 32 neuronas..... | 42 |
| Ilustración 40. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida y una tercera capa oculta de 64 neuronas..... | 43 |
| Ilustración 41. Accuracy y loss tras 10000 epochs de la red neuronal con dos capas de salida y una tercera capa oculta de 64 neuronas..... | 43 |
| Ilustración 42. Modelo de la red neuronal con cuatro capas de salida..... | 44 |
| Ilustración 43. Accuracy y F1_score aplicando batch_size=64. | 44 |
| Ilustración 44. Accuracy y F1_score sin aplicar batch_size..... | 45 |
| Ilustración 45. Función accuracy y loss de la primera capa de salida (scored) | 45 |
| Ilustración 46. Función accuracy y loss de la segunda capa de salida (saved) | 45 |
| Ilustración 47. Función accuracy y loss de la tercera capa de salida (kick_direction) | 46 |
| Ilustración 48. Función accuracy y loss de la cuarta capa de salida (keeper_direction) ... | 46 |
| Ilustración 49. Resultados del entrenamiento y evaluación del modelo. | 47 |
| Ilustración 50. Resultados de la precisión y Valor-F de las capas de salida. | 47 |
| Ilustración 51. Función accuracy y loss de la primera capa de salida (scored) | 48 |
| Ilustración 52. Función accuracy y loss de la segunda capa de salida (saved) | 48 |
| Ilustración 53. Función accuracy y loss de la tercera capa de salida (kick_direction) | 48 |
| Ilustración 54. Función accuracy y loss de la cuarta capa de salida (keeper_direction)... | 49 |
| Ilustración 55. Resultados del entrenamiento y evaluación del modelo. | 49 |
| Ilustración 56. Resultados de la precisión y Valor-F de las capas de salida. | 49 |
| Ilustración 57. Función accuracy y loss de la primera capa de salida (scored) | 50 |
| Ilustración 58. Función accuracy y loss de la segunda capa de salida (saved) | 50 |
| Ilustración 59. Función accuracy y loss de la tercera capa de salida (kick_direction) | 51 |
| Ilustración 60. Función accuracy y loss de la cuarta capa de salida (keeper_direction) ... | 51 |
| Ilustración 61. Resultados del entrenamiento y evaluación del modelo. | 51 |
| Ilustración 62. Resultados de la precisión y Valor-F de las capas de salida. | 52 |
| Ilustración 63. Función accuracy y loss de la primera capa de salida (scored) | 52 |
| Ilustración 64. Función accuracy y loss de la segunda capa de salida (saved) | 52 |
| Ilustración 65. Función accuracy y loss de la tercera capa de salida (kick_direction) | 53 |
| Ilustración 66. Función accuracy y loss de la cuarta capa de salida (keeper_direction) ... | 53 |
| Ilustración 67. Resultados del entrenamiento y evaluación del modelo. | 54 |
| Ilustración 68. Resultados de la precisión y Valor-F de las capas de salida. | 54 |
| Ilustración 69. Resumen del modelo definitivo | 55 |
| Ilustración 70. Matriz de confusión y matriz de confusión normalizada de la variable scored..... | 55 |
| Ilustración 71. Matriz de confusión y matriz de confusión normalizada de la variable saved..... | 56 |
| Ilustración 72. Matriz de confusión y matriz de confusión normalizada de la variable kick_direction..... | 56 |
| Ilustración 73. Matriz de confusión y matriz de confusión normalizada de la variable keeper_direction..... | 57 |
| Ilustración 74. Ejemplo del análisis de datos en el fútbol..... | 59 |

Índice de tablas

| | |
|--|----|
| Tabla 1. Funciones de activación | 14 |
| Tabla 2. Zonas de una portería | 18 |
| Tabla 3. Distribución Normal | 21 |
| Tabla 4. Número de penaltis/zona | 21 |
| Tabla 5. Porcentaje penaltis/zona | 21 |
| Tabla 6. Ruido | 21 |
| Tabla 7. Ruido + Porcentaje | 21 |
| Tabla 8. Ejemplo de aplicar nuevos valores al dataset | 23 |
| Tabla 9. Ejemplo de normalización del dataset | 23 |
| Tabla 10. One Hot Encoding en las variables de direcciones | 24 |
| Tabla 11. Correspondencia entre capa y variable de salida | 27 |
| Tabla 12. Determinación del valor predicho en la función de activación sigmoid | 37 |
| Tabla 13. Determinación del valor predicho en la función de activación softmax | 38 |

Índice de ecuaciones

| | |
|--------------------------------------|----|
| Ecuación 1. Mean Squared Error..... | 13 |
| Ecuación 2. Mean Absolute Error..... | 13 |

1. Introducción

1.1 Contexto

Es conocido que el uso de la inteligencia artificial llegó al mundo para quedarse, y que está en nuestras vidas incluso cuando no nos damos cuenta: desde los asistentes de voz que son capaces de reconocer la voz y entender lo que se le está pidiendo; hasta las búsquedas en internet donde simplemente escribiendo las primeras letras de una palabra o las primeras palabras de una sentencia, el buscador es capaz de autocompletar lo que se quería escribir; o las recomendaciones, ya sea de productos en páginas web, como en Amazon o en los banner de publicidad de un periódico por ejemplo; o recomendaciones de contenido digital que podría ser de interés, como en YouTube; o de música que no resulta conocida al usuario, pero pertenece al mismo género musical que suele escuchar, como podría ser en las diferentes plataformas de música.

Se puede observar que el uso de la inteligencia artificial es indispensable hoy en día, y su ámbito de aplicación es inmenso. Es por eso por lo que, el deporte en general, y el fútbol en particular, no se han quedado atrás a la hora de implementar la inteligencia artificial de una u otra manera.

Aunque en los deportes, la inteligencia artificial se ha comenzado a usar hace relativamente poco, ya es inimaginable la práctica de algún deporte profesional sin hacer uso de ella. Un ejemplo de esto anterior es el tenis, que incorporó la tecnología de ojo de halcón en marzo de 2006, cuyo funcionamiento se puede observar en la *Ilustración 1*. Dicha tecnología es capaz de generar una imagen de la trayectoria de la pelota gracias a las imágenes proporcionadas por diferentes cámaras posicionadas estratégicamente en el estadio. De esta manera, ante la duda de algún jugador ante un posible fallo del árbitro, podrían solicitar el uso de esta tecnología para demostrar que el árbitro está equivocado.

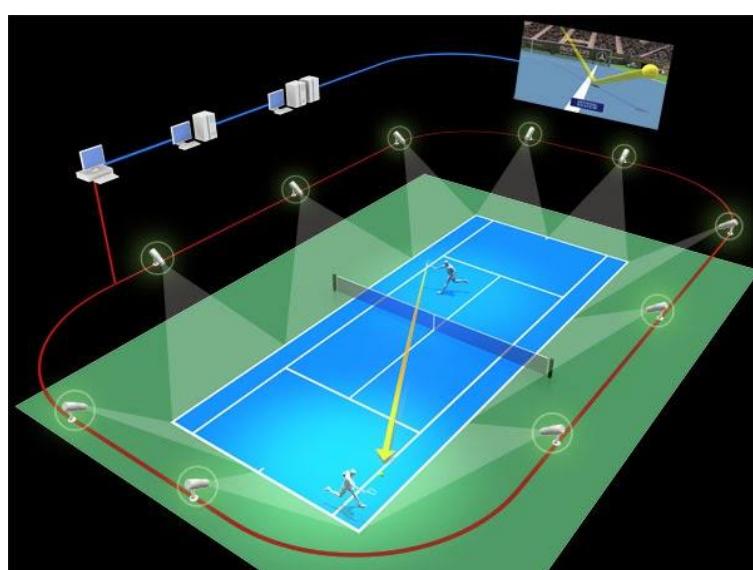


Ilustración 1. Sistema de cámaras en el Ojo de Halcón [1]

Un punto del amplio uso de la inteligencia artificial es el de intentar evitar el error humano; ya que, como en el caso anteriormente mencionado del tenis, el ojo humano, por muy entrenado que esté, hay veces que es imposible determinar si una pelota de tenis a una velocidad de unos 150km/h ha botado, aunque sea mínimamente, en la línea o no, cuyo ejemplo se puede apreciar en la *Ilustración 2*.

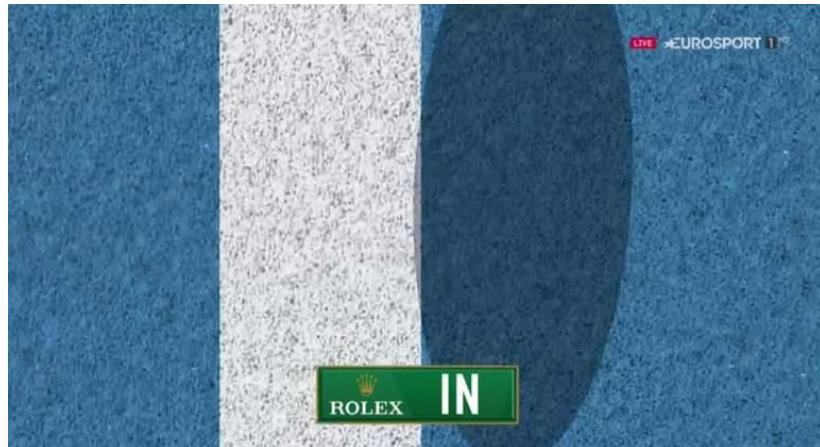


Ilustración 2. Ejemplo de uso del Ojo de Halcón en el tenis [2]

En el fútbol, la revolución tecnológica tardó algo más en llegar, quizá debido a las diferentes interpretaciones dentro del deporte, ya que para una misma jugada a dos árbitros les puede parecer sanciones diferentes. Pero la revolución tecnológica es imparable. Es por eso por lo que en el año 2012 llegó el anteriormente mencionado ojo de halcón, funcionando de manera parecida al tenis, donde no hay interpretabilidad posible. Como se puede observar en la *Ilustración 3*, aunque la gran parte del balón ha sobrepasado la línea de gol, hasta que el balón no supera la línea de gol en su totalidad no es gol. Es lo que sucede en esta ilustración: a vistas de árbitros y jugadores, fue una jugada de gol, pero la tecnología dictaminó que no era gol, por lo que debía seguir la jugada y no subir el gol al marcador.



Ilustración 3. Ejemplo del ojo de halcón en el fútbol

Desde 2012 hasta hoy, el ojo de halcón no ha sido la única implementación de la tecnología en el fútbol, ya que en 2018 se incorporó la tecnología de Árbitro Asistente de Vídeo (VAR), donde unos colegiados pueden revisar jugadas polémicas, como posibles penaltis, fuentes

de juego en jugadas de gol o posibles agresiones donde, una vez más, el ojo humano no alcanza a verlo todo.

Pero donde más se usa la inteligencia artificial en el fútbol, es quizá donde menos presencia tiene públicamente o es menos conocido por los aficionados y no aficionados. Y es el uso que hacen los diferentes clubes e instituciones de ella, ya que prácticamente cualquier club profesional tiene servicios contratados que monitorizan cualquier aspecto del juego, instalaciones, nutrición; de cara a optimizar al máximo los recursos del club y de los jugadores. Gracias a la inteligencia artificial, ahora los clubes pueden saber el riesgo de lesión de sus jugadores [3] gracias a, por ejemplo, la cantidad de minutos jugados, el nivel de exigencia en esos minutos de juego o si ha tenido lesiones previas y dónde las ha tenido.

Y en lo que ciñe a este Trabajo Fin de Grado, también ha sido de gran utilidad la aplicación de la inteligencia artificial en los penaltis. Esto es de gran utilidad en, por ejemplo, las tandas de penaltis, en las cuales un portero debe hacer frente a varios lanzadores. Pero si dicho portero conoce las características con las que lanza cada jugador, como si se para antes rematar, a qué dirección y altura, cuánta potencia suele aplicarle al golpeo, si es un chute seco o con efecto, el portero se encuentra en mayor disponibilidad de poder parar el lanzamiento que si no conociese nada del jugador rival [4]. En estos estudios se ha de tener en cuenta también las habilidades de los porteros, ya que en una tanda de penaltis quizás te interesa cambiar de portero porque tienes uno en el banquillo que simplemente es mejor parador de penaltis que tu portero titular. Para eso, también sirve este estudio.

1.2 Motivación

El principal motivo que me ha llevado a la realización de este TFG es la unión entre dos de mis pasiones: en el ámbito educativo o profesional, como es la informática en general, y la inteligencia artificial que a lo largo de la carrera siempre me había llamado la atención, y el fútbol del cuál soy aficionado desde que tengo uso de razón.

Desde que entré en la carrera, la inteligencia artificial me llamó la atención a pesar de la poca cantidad de asignaturas que hay relacionadas con este ámbito en el grado en ingeniería de computadores. Y, a medida que iba avanzando cursos, me iba fijando más como en las retransmisiones de los partidos y en las redes sociales, algo que acrecentó el deseo de involucrarme más en el ámbito de la inteligencia artificial, y, a ser posible, aplicada al deporte.

Además, es curioso cómo, según vas aprendiendo, vas viendo la infinidad de aplicaciones de la inteligencia artificial en el mundo y aprendes el cómo funcionan cosas que antes pasaban desapercibidas.

Dicho esto, y con mi intención de ver diferentes aplicaciones de la inteligencia artificial, quise realizar un proyecto en el cual fuese capaz de ver de primera mano los problemas que supone obtener por uno mismo los datos, ver la calidad de esos datos, ver la relación que hay entre diferentes datos, y obtener una red neuronal funcional capaz de, a partir de esos datos obtenidos, predecir donde va a lanzar un penalti un futbolista concreto, donde se va a lanzar un portero concreto y predecir si va a ser gol o no, y en caso de que no si va a ser porque el portero lo ha parado, o en su defecto, simplemente ha fallado el lanzador.

1.3 Objetivos

Los objetivos de este TFG se podrían dividir en diferentes tipos de objetivos dependiendo de la importancia que tengan.

Objetivos principales:

- **Obtención de un dataset¹ con una cantidad de penaltis considerable:** en el cual estén agrupadas las características principales que definen a un penalti, como podrían ser: jugador que lanza el penalti, portero que lo intenta parar, club tanto del jugador como del portero, temporada, jornada, resultado final del partido, si fue gol o no, si fue parada o fallo, dirección del lanzamiento, dirección donde va el portero con intención de parar el penalti, pie con el que lanza el jugador, partido en el que sucede el penalti, fecha del partido y tiempo en el que se lanza el penalti.
- **Procesado del dataset:** determinar qué es lo que se desea predecir en la red neuronal, y cuáles de los datos que componen el dataset son los que aportan valor al dataset.
- **Funcionamiento de una red neuronal:** ser capaz de realizar una red neuronal que, a partir de un dataset de entrada y del entrenamiento de la red neuronal, predecir si un penalti es gol o no, y la dirección del penalti.

Objetivos secundarios:

- **Aprender Python:** Python es un lenguaje de programación que es el más usado en informática en 2021. A pesar de que en una asignatura ya había programado en Python, mi conocimiento del lenguaje era muy limitado y gracias a este TFG he podido profundizar y familiarizarme más con Python y ver el potencial que tiene gracias a la gran cantidad de librerías que posee.
- **Aumentar el dataset original:** dado que el dataset de partida tiene un tamaño limitado respecto a las necesidades de una red neuronal con buenos resultados, es necesario aplicar algún algoritmo de aumento de datos de tal manera que no se pierda el valor y las características del dataset original.
- **Familiarizarse con el mundo de la inteligencia artificial:** otro objetivo de la realización de este TFG era conocer cómo funciona una red neuronal, saber qué algoritmos aplicar dependiendo de los datos que posees o de los datos que deseas obtener. Entender los algoritmos que se aplica en cada situación es esencial para saber qué se está haciendo en cada momento.
- **Dar a conocer el uso de la tecnología en el deporte:** muchas personas no están familiarizadas con la informática o con el deporte, desconoce lo que puede hacer el uso de la tecnología y de la inteligencia artificial en el deporte, pudiendo dar un punto de vista diferente del deporte y que las cosas no solo suceden por suerte.

¹ Un *dataset* es un conjunto de datos que contiene los valores para cada variable.

2. Estado del arte

2.1 ¿Qué es la Inteligencia Artificial?

La Inteligencia Artificial (IA), tal y como su nombre indica, es la inteligencia creada artificialmente que consiste en que una máquina sea capaz de simular procesos y comportamientos de la mente humana.

“La Inteligencia Artificial es la capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas a través de la adaptación flexible.” (Andreas Kaplan, Michael Haenlein)

2.1.1 Tipos de Inteligencia Artificial

Aunque no existe una definición exacta de qué es la Inteligencia Artificial, pero todas las definiciones son parecidas entre sí; en cuanto a los tipos de Inteligencia Artificial sí que hay un consenso más amplio, realizado por Stuart J. Russell y Peter Norvig [5].

Esta división de tipos de Inteligencia Artificial se realiza en función del tipo de enfoque que se hace de ella [6], si se centra en un comportamiento humano o racional. Un sistema es racional si hace lo que cree que es correcto, tratándose así de un concepto ideal de inteligencia debido a que solo dispone de un conocimiento limitado para decidir si algo es correcto o no, pudiendo estar equivocado.

- **Sistemas que piensan como humanos.** Dentro de estos sistemas entrarían aquellos que se encargan de toma de decisiones, resoluciones de problemas o aprendizaje. Por ejemplo, las redes neuronales artificiales² [7] pertenecen a este tipo de sistema.
- **Sistemas que actúan como humanos.** Este es el caso de los robots³ que se conocen hoy en día. Tienen como función realizar tareas que podría realizar un humano, pero se busca con ellos que mejoren la eficiencia humana.
- **Sistemas que piensan racionalmente.** Aquellos sistemas capaces de percibir, razonar y actuar, imitando así un pensamiento lógico racional humano. Dentro de este tipo de sistemas tendrían lugar los sistemas expertos⁴.

² Una *red neuronal artificial* es un grupo de neuronas conectadas entre sí tal que a partir de una información de entrada produce unos valores de salida. Simula el funcionamiento de redes neuronales de seres humanos.

³ Un *robot* es una máquina automática programable capaz de realizar movimientos mecánicos.

⁴ Un *sistema experto* emplea conocimiento humano para resolver problemas como lo haría una persona experta en un ámbito.

- **Sistemas que actúan racionalmente.** Son los sistemas que tratan de imitar racionalmente el comportamiento humano. Los agentes inteligentes⁵ son un ejemplo de este tipo de sistemas.

La Inteligencia Artificial no deja de ser una ciencia, por lo que está en constante evolución y es dependiente de diferentes desarrollos tecnológicos, ya que cuanto mejor sean las máquinas, con mejor hardware, mejores algoritmos, más podrá avanzar la Inteligencia Artificial.

2.2 Historia de la Inteligencia Artificial

Aunque popularmente se crea que la Inteligencia Artificial comenzó a existir en el siglo XXI, la realidad es que a mediados del siglo XX.

2.2.1 Año 1921

Incluso antes del propio nacimiento de la Inteligencia Artificial, el autor teatral checo Karel Čapek estrenó una obra llamada *Rossum's Universal Robots* [8], en la cual una empresa construía humanos artificiales, haciéndose pasar por humanos y pudiendo pensar por sí mismos. Además, esta obra propone un dilema, ya que en la obra dichos robots en vez de ayudar a los seres humanos, acaban intentando destruir la humanidad, perdiendo así la empresa el control de dichos robots, siendo éste un dilema que ha rodeado siempre cualquier innovación tecnológica.

Esta obra es notoria importancia porque es donde se incluyó por primera vez la palabra robot⁶.

2.2.2 Año 1936

Este año es de vital importancia para lo que conocemos hoy en día como Inteligencia Artificial, ya que es el año en el cual el matemático Alan Turing publicó un artículo llamado “On Computable Numbers, with an Application to the Entscheidungsproblem”⁷ [9]. En este artículo Alan Turing, considerado uno de los principales fundadores de la computación, introdujo el concepto de algoritmo, además del concepto de máquinas lógicas capaces de resolver problemas matemáticos.

⁵ Un *agente inteligente* es una entidad que, basándose en su propio conocimiento y de percibir su entorno, es capaz de procesar dichas percepciones y actuar de manera racional con el objetivo de maximizar u optimizar un resultado.

⁶ La palabra *robot* fue ideada a partir de la palabra checa <>*robota*<>, que significa <>*esclavo*<>.

⁷ El *Entscheidungsproblem* fue el reto en lógica simbólica de encontrar un algoritmo capaz de decidir si una fórmula de cálculo de primer orden es un teorema.

2.2.3 Año 1941

En 1941 el ingeniero alemán Konrad Zuse creó Z3 [10], siendo así la primera computadora programable y automática de la historia. Se podría decir que fue el primer ordenador de la historia. En la *Ilustración 4* se puede apreciar el gran tamaño que tenía la computadora.

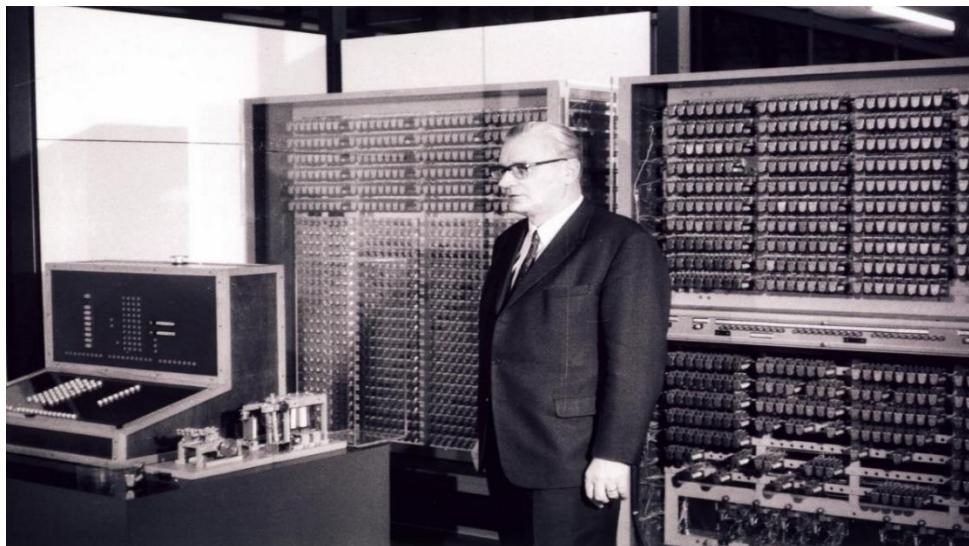


Ilustración 4. Konrad Zuse junto a la máquina Z3

Z3 estaba construida con dos mil trescientos relés, realizando cálculos en coma flotante. Era capaz de realizar bucles a pesar de no disponer saltos condicionales. Fue construida con componentes electromecánicos.

2.2.4 Años 1943 - 1955

En 1943, Warren McCulloch (neurólogo y cibernetico estadounidense) y Walter Pitts (lógico estadounidense que trabajó en el campo de neurociencia computacional) realizaron lo que se considera el primer trabajo de Inteligencia Artificial. Partieron de sus conocimientos en fisiología y funcionamiento de las neuronas en el cerebro, del análisis formal de la lógica proposicional de Russell y Whitehead y la teoría de la computación de Turing. El trabajo consistía en un conjunto de neuronas artificiales que podían tener dos estados: activadas o desactivadas. La activación consistía en una estimulación provocada por neuronas vecinas.

En 1950, Alan Turing propuso una prueba que consiste en determinar si una máquina es capaz de realizar un comportamiento similar al que haría una persona. Si una persona externa, no es capaz de determinar si quien le está respondiendo es una persona o una máquina, es que la máquina ha pasado la prueba, como se puede ver representada en la *Ilustración 5*. Dicha prueba es conocida mundialmente como el *Test de Turing*.

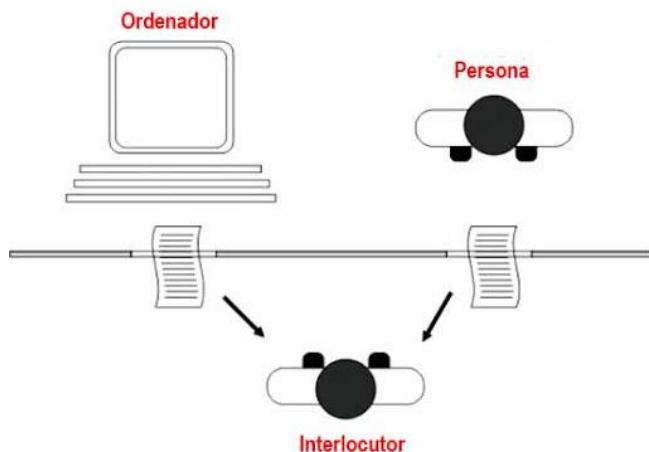


Ilustración 5. Test de Turing

Además, en este mismo año Turing introdujo lo que es el aprendizaje automático, el aprendizaje por refuerzo y los algoritmos genéticos.

En 1951, los matemáticos Marvin Minsky y Dean Edmonds construyeron el primer computador basado en una red neuronal, llamado SNARC.

2.2.5 Año 1956

En este año es donde, por primera vez, se formaliza el término “inteligencia artificial”. Ocurrió en una conferencia de John McCarthy, informático especializado en inteligencia artificial.

2.2.6 Años 1956 - 1969

En esta época, gracias a la incorporación de los transistores a los computadores, las velocidades de ejecución de cálculos se redujeron considerablemente, permitiendo así grandes éxitos en el mundo de la Inteligencia Artificial. Por ejemplo, en 1958 John McCarthy publicó un artículo llamado *Programs with Common Sense*, donde diseñó un programa con la intención de encontrar solución a problemas utilizando el conocimiento. Además, también McCarthy en 1958 desarrolló el lenguaje de alto nivel *Lisp*, uno de los lenguajes de programación más antiguos del mundo.

Fueron unos años donde se crearon múltiples programas capaces de resolver problemas con cierto grado de dificultad.

En 1959, Frank Rosenblatt introdujo el perceptrón, aunque comenzó su desarrollo en 1957.

Entre 1964 y 1966, Joseph Weizenbaum diseñó un programa llamado *ELIZA*, capaz de procesar lenguaje natural, en lugar de requerir una comunicación con las máquinas a través de una programación en código.

2.2.7 Años 1970-1980

A lo largo de los 70s ocurrió lo que se conoce como el primer invierno de la Inteligencia Artificial. Esto fue provocado por las limitaciones hardware de la época.

Aun así, en estos años aumentó el uso de sistemas expertos. Algunas de ellas siguen usándose hoy en día como el *Shell*⁸.

2.2.8 Años 1980-1986

A partir de esta época, la Inteligencia Artificial se convierte en una industria, convirtiéndose hasta la industria que es hoy en día.

En 1981, Kazuhiro Fuchi anunció un proyecto llamado *Quinta generación de computadores*, que consistió en desarrollar una clase de computadores que, usando tecnologías de Inteligencia Artificial, usando el lenguaje *Prolog*⁹, fuese capaz de resolver problemas como, por ejemplo, la traducción del inglés a japonés y viceversa.

2.2.9 Años 1987-2011

Tal y como ocurrió en la década de 1970, esta época sufrió otro invierno en cuanto a investigación se refiere. No fue tanto un problema tecnológico, como anteriormente, pero sí de utilidad y coste, ya que el almacenamiento de datos era extremadamente caro.

Aun así, sucedieron algunos hitos históricos, como la victoria de la supercomputadora desarrollada por IBM¹⁰, *Deep Blue*, al ajedrez en 1996 frente al campeón mundial Garri Kímovich Kaspárov. Este hito quizás no fue nada muy innovador en cuanto a tecnología, pero sí consiguió acercar al público general lo que era la Inteligencia Artificial.

En 1998, la doctora Cynthia Breazeal en el MIT¹¹, desarrolló el primer robot capaz de reconocer y simular emociones, llamado *Kismet*. En 2009 ya había sistemas inteligentes terapéuticos capaces de detectar emociones para poder interactuar con niños autistas.

En 2011, IBM creó una supercomputadora, llamada *Watson*, basada en inteligencia artificial capaz de responder a preguntas realizadas en lenguaje natural. Posee una base de datos almacenada localmente. Dicha supercomputadora fue capaz

⁸ El *Shell* es un intérprete de comandos que provee una interfaz de usuario para acceder a los servicios del sistema operativo.

⁹ *Prolog* proviene del francés *PROgrammation en LOGique*.

¹⁰ IBM: International Business Machines Corporation.

¹¹ MIT: Instituto de Massachusetts de Tecnología.

de ganar un concurso televisivo de responder preguntas, como se puede apreciar en la Ilustración 6 el premio final.



Ilustración 6. Watson ganador de *Jeopardy!*, venciendo a los dos campeones del juego

2.2.10 Desde 2012 hasta la actualidad

Con la tecnología cada vez más avanzada, desde 2012 se han ido produciendo hitos hasta el momento inimaginables en Inteligencia Artificial.

En 2012, Google crea un superordenador capaz de reconocer gatos, caras y cuerpos humanos.

En 2014, una inteligencia artificial fue capaz de engañar a 30 de los 150 jueces de un Test de Turing, superando así la prueba. La inteligencia artificial les hizo creer que estaban hablando con un niño ucraniano de 13 años.

Hitos como los anteriormente mencionados, se han ido sucediendo a lo largo de estos años.

El uso de la Inteligencia Artificial no se basa en la consecución de retos, si no que su uso abarca todos los ámbitos de la sociedad actual. Desde su uso en videojuegos, automatización de máquinas en fábricas, seguridad informática, conducción autónoma o salud, como, por ejemplo, en la detección a tiempo del agravamiento del coronavirus SARS-CoV-2 (conocido como Covid-19) en los pacientes.

2.3 Machine Learning

El *machine learning* o aprendizaje automático es un subconjunto de la Inteligencia Artificial, diferenciándose en el concepto general de la Inteligencia Artificial en que el *machine learning* se busca la automatización del aprendizaje por las propias máquinas mediante el reconocimiento de patrones.

Esa búsqueda de patrones basa su funcionamiento en diferentes algoritmos de aprendizaje que hace esa labor de automatización, según el tipo de tarea que se desea realizar. Básicamente, aprenden a través de la experiencia obtenida de las características de entrada.

Estos algoritmos de aprendizaje pueden ser de diferente tipo:

- **Aprendizaje supervisado.** Es un tipo de aprendizaje que aprende de datos etiquetados o clasificados. Se le proporciona unos datos de entrada y la salida deseada, y a partir de ahí intenta buscar patrones de tal manera que sólo con los datos de entrada, la salida se acerque lo más posible a la salida que también se ha proporcionado.
- **Aprendizaje no supervisado.** A diferencia del aprendizaje supervisado, aquí los datos no están etiquetados, es decir, se le proporciona el conjunto de datos de entrada también, pero, en cambio, no se le proporciona los datos de salida por lo que en este tipo de aprendizaje no se conoce el valor a predecir. Este desconocimiento de los valores de salida es lo que se conoce como “no supervisado”.
- **Aprendizaje por refuerzo.** Este método de aprendizaje se basa en la técnica “prueba y error” hasta que es capaz de alcanzar la mejor forma de realizar lo que se desea, recibiendo una retroalimentación en cada prueba. Esta retroalimentación puede ser una recompensa o un castigo, si la prueba ha sido satisfactoria o no respectivamente.

En cuanto al tipo de algoritmo que se puede usar, depende del tipo de implementación que se desea hacer:

- **Algoritmos de regresión.** Se fija una característica y el objetivo de estos algoritmos es encontrar patrones y relaciones entre el resto de las características y la fijada.
- **Algoritmos Bayesianos.** Son aquellos que están basados en el teorema de Bayes¹². Su función es clasificar los valores de una característica en clases o categorías.
- **Algoritmos de agrupación.** También conocidos como *clustering*. Se usan en aprendizaje no supervisado debido a que su función es encontrar agrupaciones en conjuntos de datos no etiquetados, es decir, grupos de datos similares entre sí.
- **Algoritmos de árboles de decisión.** Son algoritmos con estructura de árbol que se asemejan a un diagrama de flujo, siendo el nodo del árbol la decisión tomada y las ramas hijas del nodo los resultados de haber tomado dicha decisión.
- **Algoritmos de Redes Neuronales.** Son las Redes Neuronales Artificiales, básicamente basan su funcionamiento en imitar el funcionamiento del cerebro a la hora de procesar información. Se ampliará su explicación en el apartado 2.3.
- **Algoritmos de reducción de dimensionalidad.** Estos algoritmos buscan reducir el número de característica de entrada sin perder información del conjunto de datos ni la relación entre los mismos, mejorando así la eficiencia y el tiempo de cómputo. Se

¹² El teorema de Bayes es una teorema utilizado en probabilidad, planteado por Thomas Bayes y publicada pasada su fallecimiento, en 1763.

ampliará la explicación y los tipos de algoritmos de reducción como el *Principal Component Analysis* (PCA) y el *Linear Discriminant Analysis* (LDA) en el apartado 3.2.4.

- **Algoritmos de Deep Learning.** También conocido como aprendizaje profundo. Son aquellos que se ejecutan en varias capas de Redes Neuronales. Aprenden más sobre las características de entrada a medida que van pasando por cada capa.

2.4 Redes Neuronales Artificiales

Las redes neuronales artificiales son un modelo computacional cuyo objetivo es imitar el comportamiento de un cerebro a la hora de procesar información de entrada y generar una o varias salidas en base al tratamiento de dicha información, tal y como lo harían las neuronas de un cerebro.

Las redes neuronales están formadas por neuronas y las conexiones que existen entre ellas. Cada neurona es la encargada de realizar el tratamiento de la información de entrada, que en las redes neuronales artificiales será una función matemática, un cálculo numérico. Esta función matemática es la denominada función de activación. Además, no todas las informaciones de entrada pueden ser de igual importancia a la neurona, o en una vista más general del problema, a la red neuronal entera. Es decir, cada característica de entrada (x en *Ilustración 7*) podría tener una ponderación o peso diferente (w en *Ilustración 7*).

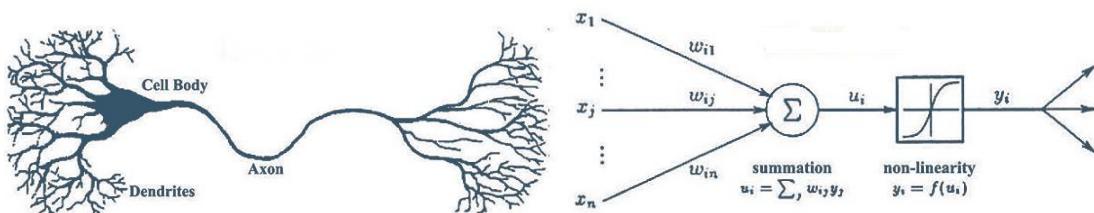


Ilustración 7. Comparación de una red neuronal biológica y de una red neuronal artificial [11]

Una neurona artificial es lo que se conoce como perceptrón.

Una red neuronal está formada por capas. Existen tres posibles tipos de capas:

- **Capa de entrada.** Es la primera capa de una red neuronal. Es la encargada de recibir la información que se quiere procesar. Tiene tantas neuronas como características o variables de entrada tengan los datos. Por ejemplo: edad, peso, altura, sexo, etcétera.
- **Capa oculta.** Las capas ocultas son aquellas capas intermedias que conectan la capa de entrada con la capa de salida. Puede haber tantas capas ocultas como se desee, incluyendo que no haya ninguna.
- **Capa de salida.** Es la última capa del modelo. El valor o los valores de salida son el resultado del trabajo de toda la red neuronal.

Esta constitución de la red neuronal en capas se puede apreciar en la *Ilustración 8*.

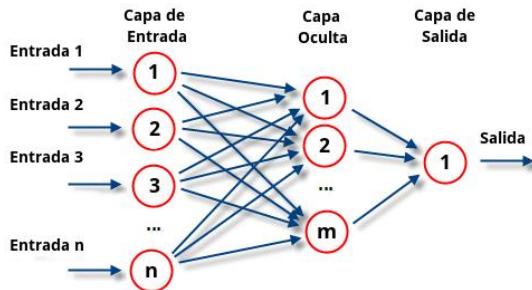


Ilustración 8. Representación de las capas de una red neuronal

La red es capaz de aprender a través del entrenamiento de esta, porque en el entrenamiento la red aprende de resultados conocidos, pero una vez finalizado el entrenamiento la red es capaz de predecir resultados a partir de datos del mismo tipo con los que se realizó el entrenamiento de los cuales no se conoce el resultado previamente.

A la hora de realizar el entrenamiento de la red, existen diferentes tipos de configuraciones que se le puede aplicar a la red. Dependiendo de dicha configuración, se puede obtener unos resultados u otros en la red.

Dicha configuración de la red neuronal se divide en parámetros e hiperparámetros. Los parámetros son aquellos valores que la red neuronal aprende por sí misma; mientras que los hiperparámetros son aquellos que no se pueden aprender, sino que deben configurarse y ajustarse.

Algunos hiperparámetros son:

- **Tasa de aprendizaje.** También conocido como *learning rate*. Determina el tamaño de lo que se avanza en cada iteración de la función de coste. Suele ser un valor muy pequeño, aunque eso lleva más tiempo de ejecución.
- **Elección de función de coste.** La función de coste busca determinar el error entre el valor estimado y el valor real. Algunas funciones de coste son:
 - MSE: *Mean Squared Error*. Mide la diferencia cuadrada entre el resultado esperado y_i y el resultado predicho \hat{y}_i .

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Ecuación 1. Mean Squared Error

- MAE: *Mean Absolute Error*. Calcula la diferencia en valor absoluto entre el resultado esperado y_i y el resultado predicho \hat{y}_i .

$$MAE = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

Ecuación 2. Mean Absolute Error

- *Categorical Cross-Entropy*. Sirve para calcular qué tan buena es nuestra distribución respecto a un número de clases, dando la probabilidad de cada clase.
- *Binary Cross-Entropy*. Tiene el mismo funcionamiento que *Categorical Cross-Entropy*, solo que esta es para valores binarios, 1 ó 0
- **Elección de método de optimización del peso.** Los métodos más usados en redes neuronales son los de tipo gradiente, más en concreto los de descenso del gradiente, que consiste en encontrar los parámetros que minimizan la función de coste. Algunos optimizadores del descenso de gradiente son:
 - RMSProp: Root Mean Squared Propagation.
 - ADAM: *Adaptative moment estimation*. Es el más utilizado gracias a que es el más efectivo respecto a otros optimizadores.
- **Elección función de activación.** Es la función que se le aplica en una neurona a los valores de entrada que le llegan desde la capa anterior y la cual obtiene una salida. Algunas funciones de activación son las mostradas en la *Tabla 1*.

| Función | Fórmula | Representación |
|-----------------------------|---|----------------|
| Lineal | $f(x) = x$ | |
| Escalón | $f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 & \text{si } x \geq 1 \end{cases}$ | |
| Sigmoidal | $f(x) = \frac{1}{1 - e^{-x}}$ | |
| ReLU: Rectified Linear Unit | $f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 1 \end{cases}$ | |
| Tanh: Tangente hiperbólica | $f(x) = \frac{2}{1 + e^{-2x}} - 1$ | |
| Softmax | $f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}}$ | |

Tabla 1. Funciones de activación

- **Elección del tamaño del lote.** Conocido como *batch size*. El lote es el número de datos con el que trabaja la red. El tamaño suele ser 32, 64, 128 ó 256, aunque su rango está entre 1 y el tamaño del conjunto de datos de entrenamiento. No es estrictamente necesario dividir el conjunto de entrenamiento en lotes.
- **Número de epochs.** Es la cantidad de veces que el algoritmo de entrenamiento trabaja sobre el conjunto de datos de entrenamiento.
- **Número de capas ocultas.** Es muy importante una buena elección de la cantidad de capas ocultas que va a tener la red neuronal, aunque no hay una respuesta única acerca de saber cuántas capas ocultas usar.
- **Número de unidades en cada capa oculta.** Las unidades de la capa oculta representan las neuronas. Al igual que en la elección del número de capas, no hay una respuesta única sobre el número de unidades que debe tener cada capa oculta. Si hay demasiadas neuronas en la capa oculta, puede provocar *overfitting*¹³; pero en caso de quedarse corto con el número de neuronas, puede provocar *underfitting*¹⁴.

No obstante, no todo es idílico en las redes neuronales artificiales, ya que las propias redes podrían estar entrenándose mal, obteniendo así unos resultados incorrectos o, mejor dicho, no correctos del todo. Además, no existe una ciencia cierta a la hora de saber qué características de la red funcionan correctamente, por lo que una labor que hay que realizar con una red neuronal artificial es la de probar diferentes hiperparámetros y funciones hasta que la red neuronal funciona correctamente y poder obtener unos resultados coherentes.

¹³ El *overfitting* o sobreajuste es lo que ocurre cuando el modelo solo aprende los casos particulares y es incapaz de reconocer datos nuevos.

¹⁴ El *underfitting* o subajuste es cuando el modelo no es capaz de generalizar el conjunto de datos, es decir, aprende el ruido.

3. Desarrollo del proyecto

En todos los trabajos donde se usan redes neuronales, tan importante es una red neuronal bien entrenada como un buen *dataset*, ya que son la base del entrenamiento de la red neuronal. Dado que este TFG trata acerca de la predicción de penaltis, y los datos de éstos son difícilmente accesibles, se decidió crear el dataset desde cero.

3.1 Creación del dataset

Dado que se decidió crear el *dataset*, se decidió qué características eran importantes conocer a la hora de lanzar un penalti, ya que no es solo llegar al punto de penalti y tirar. La ciencia de los penaltis en el fútbol está mucho más avanzada de lo que creemos, ya que tener una información previa de hacia qué lado suelen lanzar los penaltis los jugadores rivales, e incluso la fuerza que le imprimen, la altura y el tipo de golpeo, puede ayudar mucho al portero. Del mismo modo, para un lanzador de penaltis le es de gran importancia saber si el portero suele lanzarse hacia un lado u otro, o quedarse quieto.

Como se puede ver en la *Ilustración 9* a continuación, el portero Tim Krul del Norwich, durante un partido de copa entre el Norwich y el Tottenham, llevaba apuntado la información de hacia qué lado solían lanzar los jugadores rivales. Gracias a dicha información, Tim Krul paró dos penaltis de una tanda de cinco, clasificándose así el Norwich para la siguiente ronda.



Ilustración 9. El portero Tim Krul durante un partido entre el Norwich y el Tottenham

Ejemplos como el anterior se han dado incontables veces, aunque no siempre te puede ayudar a ganar, ya que en la pasada final de Europa League entre el Villareal y el Manchester United, el portero David De Gea contaba con la misma información de los jugadores del Villareal, y sin embargo no fue capaz de parar ninguno de los once lanzamientos que realizó el Villareal, ganando así el Villareal la tanda de penaltis y, por ende, la competición.

3.1.1 Elección de las características

Es una de las partes más importantes de un *dataset*. Las características son los elementos que definen el *dataset*. Dado que el tema que se trata es el de los penaltis, se han considerado las siguientes características a recoger por cada penalti lanzado:

- **Season.** La temporada en que se ha lanzado el penalti.
- **Match Week.** La jornada de liga en la cual se ha lanzado el penalti. Es importante porque, por ejemplo, en las últimas jornadas de liga hay más presión si un equipo se está jugando el ganar la liga, o descender de liga, afectando tanto a lanzadores como a porteros que puedan fallar más de lo normal. Valor entre 1 y 38.
- **Date.** Fecha en que se lanzó el penalti.
- **Player.** Jugador que lanzó el penalti.
- **Goalkeeper.** Jugador que intenta parar el penalti.
- **Team Player.** Equipo de *Player*.
- **Team Goalkeeper.** Equipo de *Goalkeeper*.
- **Match.** Partido que enfrenta a los jugadores, son *Team Player* y *Team Goalkeeper*, pero identificando qué equipo juega de local y cuál de visitante. Por ejemplo: Real Madrid – Barcelona, siendo el Real Madrid el equipo local y el Barcelona el equipo visitante. Esto puede ser importante de cara a que un penalti a favor del equipo visitante, el jugador puede tener más presión y fallar el penalti provocado por la presión del público, por ejemplo.
- **Time of Penalty Awarded.** El minuto en que fue concedido el penalti. Puede servir para ver si es más normal realizar penaltis en la primera parte o en la segunda. Por ejemplo, con el cansancio a lo largo del partido, podría ser que se realicen más penaltis según pasan en los minutos, es decir, en la segunda parte. Tal y como se ha comentado en otras características, cuanto más tarde se lance el penalti, más incidencia en el resultado podría tener el penalti, por lo cual más presión para el lanzador y el portero.
- **Final Result.** Resultado final del partido.
- **Foot.** Pie con el que lanza el jugador. Es muy importante saber si un lanzador es diestro (R) o zurdo (L).
- **Scored.** Si el penalti ha sido gol (1) o no (0).

- **Saved.** En caso de que el penalti no haya sido gol, es importante saber si el portero lo ha parado (1) o, en cambio, el lanzador lo ha fallado solo lanzándola fuera de la portería o dando al palo y rebotando fuera de la portería (0). En este *dataset* se ha considerado que en el segundo caso de que el balón de en el palo y salga rebotado fuera, será considerado igual que si hubiera sido directamente fuera de portería, como se puede apreciar en la *Ilustración 10*.

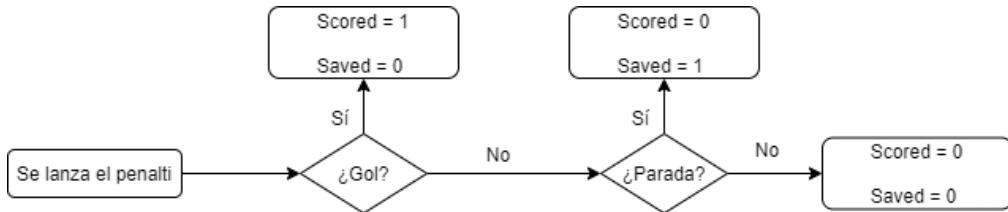


Ilustración 10. Diagrama de flujo de la decisión de *scored* y *saved*

- **Kick Direction.** Dirección en la cual es lanzado el penalti.
- **Keeper Direction.** Dirección donde se tira el portero buscando parar el penalti.

Estas cuatro últimas características, *scored*, *saved*, *kick direction* y *keeper direction*, son los objetivos del *dataset* y serán los valores a predecir por la red neuronal; mientras que las otras once características primeras, *season*, *match week*, *date*, *player*, *goalkeeper*, *team player*, *team goalkeeper*, *match*, *time of penalty awarded*, *final result* y *foot*, serán las características de entrada de la red neuronal.

3.1.2 Determinación de la dirección del penalti

Es necesario hacer alguna especie de división de la portería para ser capaces de determinar por dónde se lanza el penalti y hacia donde se lanza el portero. Esto se puede ver en la *Tabla 2*, donde se ha dividido la portería como si fuese una matriz de 3x3.

| | | |
|-----------------|-------------------|------------------|
| LT: Left Top | CT: Center Top | RT: Right Top |
| LC: Left Center | CC: Center Center | RC: Right Center |
| LD: Left Down | CD: Center Down | RD: Right Down |

Tabla 2. Zonas de una portería

Se ha decidido hacer esta división, ya que refleja lo que se quiere demostrar, y es hacia dónde suelen lanzarse los penaltis. En el caso de los porteros, dado que un cuerpo humano ocupa más que un balón de fútbol y el portero podría cubrir con su cuerpo más de un espacio de los nueve en los que se ha hecho la división, se ha dado el caso en los que se ha recopilado penaltis en los cuales un portero para el penalti con los pies, aunque se había lanzado en otra dirección diferente al balón, gracias a que con su cuerpo puede ocupar dos o incluso tres espacios en la portería.

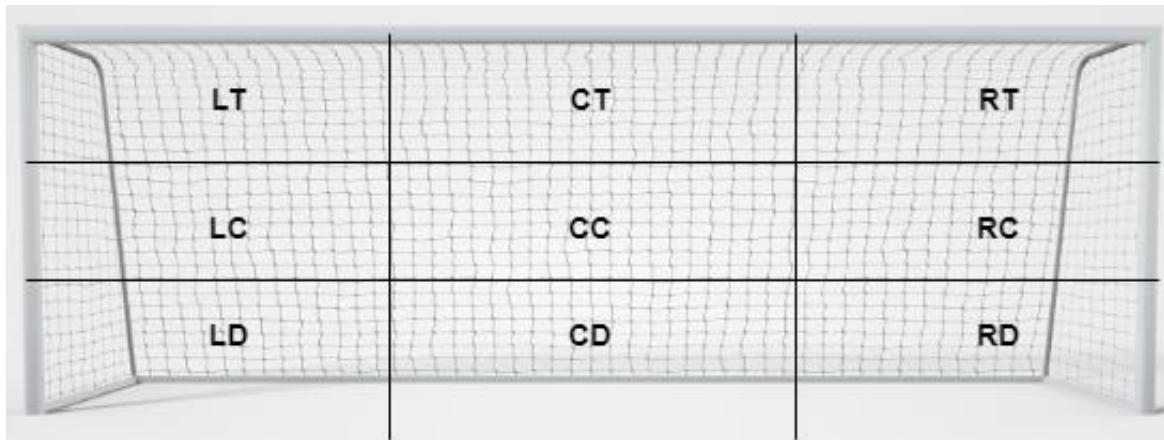


Ilustración 11. Portería dividida en nueve zonas.

Se considera que la dirección del penalti es tomada desde el punto de vista del lanzador, de frente a la portería, tal y como se muestra en la *Ilustración 11*. Es decir, en el caso del portero hace efecto espejo, si un portero se lanza hacia su derecha, como la vista se toma desde el lanzador, en realidad el portero se está lanzando hacia la izquierda.

3.1.3 Obtención de los datos

Una vez decidido qué datos son importantes a la hora de analizar un penalti, y hecha la división de la portería por zonas para determinar dónde se ha lanzado un penalti y donde se ha lanzado el portero, es momento de obtener los datos.

Para ello, se ha realizado una búsqueda en una página web llamada “Resultados Futbol” (<https://www.resultados-futbol.com/>) de todos los partidos de cada jornada, desde el momento en que se había disputado la última jornada de La Liga Santander, liga española de primera división, en la jornada 28 de la temporada 2020/2021, hacia atrás en el tiempo, llegando así hasta la temporada 2012/2013.

Así, se consiguieron 1048 muestras, contando las temporadas desde la 2012/2013 hasta la 2019/2020 completas, y las primeras 28 jornadas de la liga 2020/2021.

Pero en dicha página, sólo se pudieron conseguir las primeras 13 características del *dataset*, pero no venía hacia dónde se había lanzado el penalti ni el portero. Para ello, gracias al canal de YouTube de La Liga Santander [12], que es la propia liga la que tiene los derechos de dichas imágenes, se pudo obtener gracias a vídeos resúmenes de los partidos la dirección de ambos.

3.1.4 Data augmentation

No obstante, las 1048 muestras obtenidas manualmente son una cantidad bastante pequeña de cara a hacer una red neuronal que sea capaz de obtener unos

buenos resultados. Es por ello, que es necesario aplicar alguna técnica para aumentar el tamaño del *dataset*.

Se consideró aumentar el tamaño de datos mediante *Random Noise Method*, pero como este método aumentaba el tamaño de los datos mediante valores aleatorios de cada característica ya existentes en el dataset, se descartó.

Finalmente se decidió usar el ruido gaussiano, *Gaussian Noise*, ya que este método a diferencia de *Random Noise Method*, es que conserva cierta coherencia en los datos que se perderían de la otra manera.

Gaussian Noise agrega ruido muestreando de una distribución normal. El ruido agregado tiene una media de 0, pero tiene una desviación estándar configurable. Básicamente, es un error que se ajusta a la distribución normal gaussiana.

Se decidió aumentar el tamaño del conjunto de datos de 1048 muestras a 4191 muestras, de tal manera que las 1048 muestras originales resultaran el 25% del *dataset* que representara a la parte de validación, y el otro 75% generado representara a la parte de entrenamiento.

Para ello, el procedimiento que se siguió fue el de aplicar ruido gaussiano en ciertas características en las cuáles es esencial conocer la historia de esa característica, como podría ser *scored*, *saved*, *kick direction* o *keeper direction*, y en otras características como la fecha o de qué jugador generar los datos en cada iteración, se elegían al azar.

Dado que no importa de qué jugador se realiza el aumento de datos, no importa cuál de ellos escoger. Por eso, se elige dos valores aleatorios de entre las 1048 muestras originales, una para el lanzador y otra para el portero. Se ha realizado así ya que, si se coge una muestra, estaríamos haciendo *data augmentation* de los mismos penaltis, sin entrelazarse, del mismo lanzador contra el mismo portero todo el rato. Pero, por culpa de esto, hay ciertas características, en principio no tan importantes, que se han escogido aleatoriamente de una u otra muestra o aleatorio, porque si no se estaría haciendo aumento del conjunto de datos con las características de la muestra a cada iteración.

De cada muestra escogida, seleccionamos el nombre del lanzador y del portero respectivamente. También los clubes a los que pertenecen, y en el caso del lanzador del penalti se guarda también el pie con el que lanza, ya que es lo usual que siempre lance con el mismo pie.

Tal y como se ha comentado anteriormente, hay valores que se han generado aleatoriamente como la jornada en que ocurrió el penalti, el minuto, el resultado y la decisión de qué equipo es local o visitante y la fecha en que ocurrió.

Respecto a la dirección del lanzamiento y de dónde se lanza el portero, se recorre todo el *dataset* original buscando los lanzamientos pasados tanto del lanzador como del portero, y se guardan, teniendo así un array con la cantidad de veces que ese lanzador y ese portero han tirado a cada zona. Se hace lo mismo con las características *scored* y *saved*, teniendo así un histórico de las cuatro características a predecir.

Una vez recopilado los datos históricos buscados del lanzador y del penalti, se calcula la media y desviación típica, ya que son necesarios para calcular el ruido

gaussiano. Y con eso, se calcula el ruido con la distribución normal mencionada anteriormente, como se muestra en la *Tabla 3*.

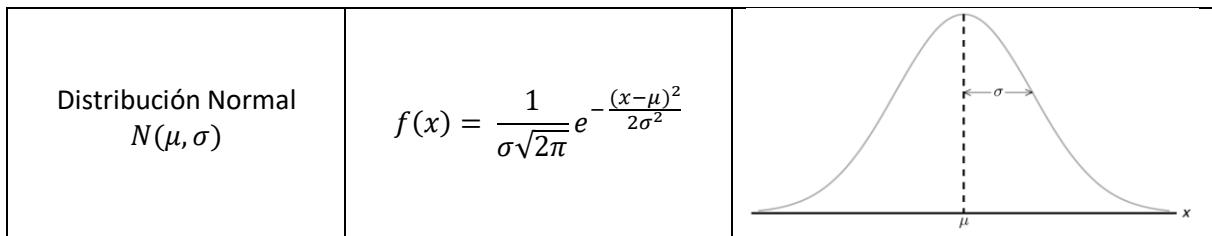


Tabla 3. Distribución Normal.

Dicho ruido generado se ha de añadir a cada variable, el caso del vector de nueve posiciones representando las nueve zonas de la portería debe añadirse a los porcentajes de las nueve posiciones, somo se van a mostrar en las *Tabla 4*, *Tabla 4*, *Tabla 6* y *Tabla 6*.

Como el ruido generado podría ser negativo, podría darse el caso de que exista alguna zona de las nueve que tenga un valor negativo, algo que no tiene sentido ya que no hay probabilidades negativas. En estos casos se barajó dos opciones: el ruido negativo considerarlo como si no hubiese ruido, es decir, ruido de 0.0, o considerar el valor absoluto del ruido. Se decidió seguir lo segundo, ya que podría darse el caso de que un tercio o más de las zonas tuvieran probabilidad cero.

| | | |
|---|---|---|
| 2 | 1 | 1 |
| 2 | 0 | 3 |
| 5 | 1 | 6 |

Tabla 4. Número de penaltis/zona

| | | |
|------|------|------|
| 0.09 | 0.04 | 0.04 |
| 0.09 | 0 | 0.14 |
| 0.23 | 0.04 | 0.28 |

Tabla 5. Porcentaje penaltis/zona

| | | |
|-------|-------|-------|
| -0.08 | 0.06 | -0.10 |
| -0.14 | -0.08 | -0.07 |
| 0.08 | -0.07 | -0.16 |

Tabla 6. Ruido

| | | |
|-------|-------|-------|
| 0.01 | 0.10 | -0.06 |
| -0.05 | -0.08 | 0.07 |
| 0.31 | -0.03 | 0.12 |

Tabla 7. Ruido + Porcentaje

Respecto a *scored* y *saved*, se actúa de manera similar. Se genera ruido gaussiano aplicando la distribución normal mediante la media y la desviación estándar. Se le añade ruido, tal y como se realizó con *kick direction* y *keeper direction*, como muestra el ejemplo de *Tabla 4*, *Tabla 4*, *Tabla 6* y *Tabla 6*. En este caso, al contrario que con las direcciones de los penaltis, en caso de que el ruido sea negativo se toma como si fuese igual a 0.0, en vez de hacer valor absoluto.

En el caso de que tanto *scored* como *saved* la probabilidad sea mayor o igual a 1, se ha decidido comparar cuál tiene más probabilidad y será el valor a tomar, si *scored* es mayor a *saved* se decide que el penalti es gol (*scored* = 1, *saved* = 0), y si no al revés (*scored* = 0, *saved* = 1).

En los otros casos no hace falta tomar ninguna decisión, ya que las otras tres posibilidades son posibles, la única que no era posible era la comentada anteriormente en la cual ocurría gol y parada a la vez, algo que no es posible.

En caso de que *saved* = 1, es decir, el portero para el penalti, se decidió que la dirección de hacia donde se lanza el portero coincidiera con el lanzamiento del penalti,

debido a que no tiene sentido que el portero pare el penalti si el balón hacia el lado izquierdo, y el portero hacia el lado derecho, por ejemplo. Aunque no es del todo correcto, es una intención de darle sentido a que el portero pare el penalti.

Una vez obtenido ya todos los nuevos valores de las características en la muestra generada mediante ruido gaussiano, se añade al *dataset* original. Esto se repite hasta llegar a las 4191 muestras.

3.1.5 Problemática data augmentation

Uno de los problemas de aplicar *data augmentation* al *dataset* es que, al fin y al cabo, se está añadiendo ruido al conjunto de datos originales, pero que es necesario para intentar tener una cantidad de muestras considerables. El meter ruido al conjunto de datos deriva en cierta pérdida de coherencia de los datos.

Otro de los problemas encontrados a la hora de implementar esta técnica, fue la del tiempo de computación, ya que al final se está recorriendo varias veces el *dataset* original solamente para generar una muestra nueva, en el caso de generar 3143 muestras, el tiempo de cómputo aumentó exponencialmente.

Una mala aplicación de *data augmentation* podría llevar al modelo a sufrir *overfitting*.

3.2 Creación de la red neuronal

Una vez definido el *dataset* con el que se va a trabajar, es momento de realizar el estudio de implementar una red neuronal.

Lo primero que hay que hacer, antes de elegir qué modelo usar y qué tipos de capas usar, es la división del *dataset* en un conjunto de entrenamiento y en otro conjunto de validación, distribuyendo el conjunto de datos original tal y como se muestra en la siguiente ilustración más adelante.



Ilustración 12. División del *dataset* en conjunto de entrenamiento y validación.

Para facilitar el aprendizaje de la red neuronal, se hizo una labor de normalización del *dataset*, normalizando así las variables de entrada de la red. De esta manera, tendremos todos los valores de entrada en el mismo rango de valores, en este caso entre los valores 0 y 1, comprimiendo así los valores del *dataset*.

Dado que el *dataset* implementado apenas posee variables de valores numéricos enteros, antes de normalizar es necesario hacer aplicar a cada valor de cada característica de entrada un valor alternativo pero que ese valor alternativo sea único. Es decir, en la característica de entrada *season*, por ejemplo, la temporada ‘2012/2013’ va a ser igual al valor ‘1’, la temporada ‘2013/2014’ va a ser equivalente al valor ‘2’, etcétera. Así con todas las variables de entrada de la red. Para luego saber qué representa ese valor ‘1’ o ese valor ‘2’ en la variable *season*, se guardan en vectores los nombres originales de la variable, y su posición en el vector es el valor nuevo adquirido, tal y como se muestra en la *Tabla 8*.

| Valor original ‘season’ | Valor nuevo ‘season’ |
|-------------------------|----------------------|
| 2012/2013 | 1 |
| 2013/2014 | 2 |
| 2012/2013 | 1 |
| 2020/2021 | 9 |
| 2016/2017 | 5 |
| ... | ... |

Tabla 8. Ejemplo de aplicar nuevos valores al dataset.

Una vez hecho lo mencionado anteriormente, teniendo ya todas las características del dataset con valores enteros, se puede implementar la normalización en el rango de valores entre 0 y 1 como se muestra más adelante, ya que al final estos valores enteros en la variable *season* va a haber nueve valores, mientras que en la variable *date*, por ejemplo, puede haber ochocientos valores, por lo que no tendrían el mismo rango entre las características del *dataset*.

| Valor ‘season’ | Normalización ‘season’ |
|----------------|------------------------|
| 1 | 0.0 |
| 2 | 0.125 |
| 1 | 0.0 |
| 9 | 1.0 |
| 5 | 0.5 |
| ... | ... |

Tabla 9. Ejemplo de normalización del dataset.

Para las variables de las direcciones, tanto del lanzamiento como de dónde va el portero, no basta con normalizar los datos, sino que es necesario implementar el método *one hot encoding*, que consiste en transformar un valor en un vector cuyo tamaño corresponde al total de los posibles valores que puede tener la variable en concreto. Es decir, como en la portería hemos definido que hay nueve zonas, el valor, por ejemplo, LD (Left-Down), se va a transformar en un vector de 3x3 donde todos los valores van a tener 0 salvo el elemento correspondiente a LD. Se puede observar mejor en la *Tabla 10*.

Esto es necesario ya que, como se explicará en el apartado 3.2.1, el objetivo es que la red devuelva un array con nueve elementos representando a las nueve zonas de la portería y el porcentaje de que el jugador lance o el portero se tire a cada zona, siendo el mayor porcentaje de las nueve posiciones la dirección elegida, mediante la implementación de la función *softmax*.

| Dirección | Valor normalizado | Dirección con One Hot Encoding |
|-----------|-------------------|---|
| LT | 0 | $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| CT | 0.125 | $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| RT | 0.25 | $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| LC | 0.375 | $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| CC | 0.5 | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| RC | 0.625 | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$ |
| LD | 0.75 | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ |
| CD | 0.875 | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ |
| RD | 1 | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |

Tabla 10. One Hot Encoding en las variables de direcciones.

3.2.1 Estructura de la red neuronal

En primer lugar, se tuvo en cuenta el hacer una red convolucional de clasificación, ya que al final el decidir si un penalti es gol o no, si va hacia un lado, al lado contrario o al medio, no deja de ser un problema de clasificación; pero, por otro lado, no tenía ningún sentido hacer una red convolucional ya que éstas son usadas, sobre todo, en clasificación de imágenes.

Se ha usado un modelo secuencial de Keras¹⁵, que consiste en la agrupación de manera secuencial de las capas que conforman el modelo.

La red neuronal está conformada por una serie de capas, todas ellas de tipo *Dense*, salvo la primera, que sirve de entrada del modelo que es de tipo *Input*.

La capa *Input* es la que conecta, por así decirlo, el *dataset* con el modelo secuencial. Recibe como entrada las características del *dataset*, por lo que su tamaño será igual al número de características de entrada.

¹⁵ Keras es una librería de redes neuronales en Python. Se ejecuta sobre TensorFlow.

Después, el modelo está conformado por capas de tipo *Dense*, que son capas cuyas neuronas están totalmente conectadas entre sí. Estas capas son las capas ocultas. En este TFG, se han probado varias configuraciones de este modelo, usando entre dos y tres capas ocultas, con diferente número de neuronas por cada capa. La activación de las capas pertenecientes al modelo secuencial ha sido la ReLU, ya que los valores negativos no son importantes aquí.

Por último, la red neuronal finaliza con tres salidas:

- **validity_1**. Es la capa Dense correspondiente a la salida de la variable *scored*. Su función de activación es *sigmoid*, es la activación perfecta para esta salida ya que su rango oscila entre 0 (valor correspondiente a no gol) y 1 (valor correspondiente a gol).
- **validity_2**. Es la capa Dense correspondiente a la salida de la variable *saved*. Su función de activación es *sigmoid*, es la activación perfecta para esta salida ya que su rango oscila entre 0 (valor correspondiente a no parada) y 1 (valor correspondiente a parada).
- **label_1**. Es la capa *Dense* correspondiente a la salida de la variable *kick_direction*. Su función de activación es *softmax*, es la activación perfecta para esta salida ya que devuelve un vector cuyo tamaño es igual al número de clases posibles de la salida, con la probabilidad de que ocurra cada clase. En este caso, las clases son nueve y corresponden al número de direcciones definidos anteriormente en una portería.
- **label_2**. Es la capa *Dense* correspondiente a la salida de la variable *keeper_direction*. Su función de activación es *softmax*, es la activación perfecta para esta salida ya que devuelve un vector cuyo tamaño es igual al número de clases posibles de la salida, con la probabilidad de que ocurra cada clase. En este caso, las clases son nueve y corresponden al número de direcciones definidos anteriormente en una portería.

| Model: "sequential" | | |
|--------------------------|--------------|---------|
| Layer (type) | Output Shape | Param # |
| dense (Dense) | (None, 16) | 192 |
| dense_1 (Dense) | (None, 64) | 1088 |
| dense_2 (Dense) | (None, 256) | 16640 |
| <hr/> | | |
| Total params: 17,920 | | |
| Trainable params: 17,920 | | |
| Non-trainable params: 0 | | |

Ilustración 13. Resumen del modelo secuencial.

En la *Ilustración 13*, se ve el modelo secuencial que lo forman las capas ocultas. En la *Ilustración 14* se muestra un resumen del modelo de la red neuronal realizado por la función *summary*, perteneciente a la librería Keras. Aquí se puede ver que tiene como

entrada la capa *Input*, después el modelo secuencial con las capas ocultas, y por último las cuatro capas de salidas, una por cada salida.

| Model: "model" | | | |
|-------------------------|--------------|---------|------------------|
| Layer (type) | Output Shape | Param # | Connected to |
| input_1 (InputLayer) | [None, 11] | 0 | |
| sequential (Sequential) | (None, 256) | 17920 | input_1[0][0] |
| dense_3 (Dense) | (None, 1) | 257 | sequential[0][0] |
| dense_4 (Dense) | (None, 1) | 257 | sequential[0][0] |
| dense_5 (Dense) | (None, 9) | 2313 | sequential[0][0] |
| dense_6 (Dense) | (None, 9) | 2313 | sequential[0][0] |

Total params: 23,060
Trainable params: 23,060
Non-trainable params: 0

Ilustración 14. Resumen de la red neuronal.

3.2.2 Entrenamiento de la red neuronal

Una vez finalizada la red neuronal, llega el momento de entrenar la red. Para ello, es necesario compilar el modelo para poder ejecutarla. El proceso de compilación implica una caracterización de hiperparámetros que hay que ir ajustando. Es por eso, que en este TFG se han probado diferentes configuraciones buscando el mejor resultado posible, aunque dando todos ellos resultados parecidos.

A la hora de compilar hay dos hiperparámetros principales:

- **Optimizador.** Es necesario indicar el algoritmo de optimización que la red va a utilizar. Se ha utilizado el algoritmo *Adam*, explicado anteriormente en el apartado 2.3, con una tasa de aprendizaje de 0.0001, aunque también se ha probado con una tasa de aprendizaje de 0.0002.
- **Función de coste.** También es necesario indicar la función de coste. Se han usado tanto la función de coste '*mean_squared_error*', como el conjunto de funciones de coste '*binary_crossentropy*' y '*categorical_crossentropy*'. En este último caso, la función de coste '*binary_crossentropy*' se implementa para las dos primeras variables de salida: *scored* y *saved*; y la función '*categorical_crossentropy*' para las dos últimas variables de salida: *kick_direction* y *keeper_direction*.
- **Métricas.** Lista de las métricas a evaluar durante el entrenamiento y validación del modelo. Es opcional. En este TFG se ha implementado la métrica de *accuracy* que computa la frecuencia con que la predicción de la variable acierta el valor real de la variable. También se ha implementado la métrica de *loss*.

A la hora de ajustar el modelo, se pueden configurar dos parámetros:

- **Número de épocas.** También conocido como *epochs*. Es el número de veces que se van a ejecutar los algoritmos de propagación y propagación hacia atrás.
- **Lote.** Se puede ajustar el tamaño del lote con el que se va a realizar la ejecución de cada *epoch*. En este TFG se ha probado con un tamaño de lote de 16 y 64.

3.2.3 Evaluación de la red neuronal

Tal y como se ha comentado en el apartado 3.2.1, la red neuronal va a tener cuatro salidas, una por cada variable a predecir.

La evaluación del modelo se lleva a cabo con la función *evaluate* que devuelve tanto la pérdida como las métricas especificadas durante la compilación, como se puede observar en la *Ilustración 15*.¹⁶

```
33/33 [=====] - 0s 625us/step - loss: 6.6205 -
dense_3_loss: 0.9760 - dense_4_loss: 0.8331 - dense_5_loss: 2.4441 - dense_6_loss: 2.3673
dense_3_accuracy: 0.3139 - dense_4_accuracy: 0.3836 - dense_5_accuracy: 0.1069 - dense_6_accuracy: 0.1164
```

Ilustración 15. Función *evaluate*.

| Capa | Variable |
|---------|------------------|
| Dense_3 | Scored |
| Dense_4 | Saved |
| Dense_5 | Kick_direction |
| Dense_6 | Keeper_direction |

Tabla 11. Correspondencia entre capa y variable de salida.

En la *Tabla 11* se muestra qué capa de salida corresponde a qué variable a predecir.

En la *Ilustración 16* e *Ilustración 17* se va a poder observar cómo evoluciona los valores de precisión en las salidas de la red neuronal.

¹⁶ Todas las pruebas realizadas en este apartado han sido con los siguientes parámetros: *epochs*=1000, *batch_size*=64, *losses* = ['binary_crossentropy', 'binary_crossentropy', 'categorical_crossentropy', 'categorical_crossentropy'], *optimizer* = Adam(*lr*=0.0001), función de activación de las capas de salida: ['sigmoid', 'sigmoid', 'softmax', 'softmax'], tres capas ocultas con 16, 32 y 256 neuronas respectivamente cuya función de activación de las tres es '*relu*'.

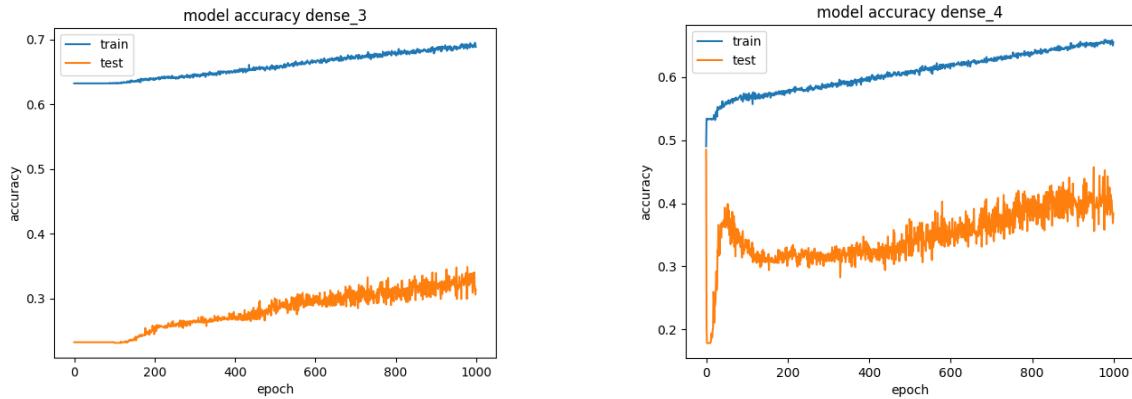


Ilustración 16. Función accuracy de las variables scored y saved respectivamente.

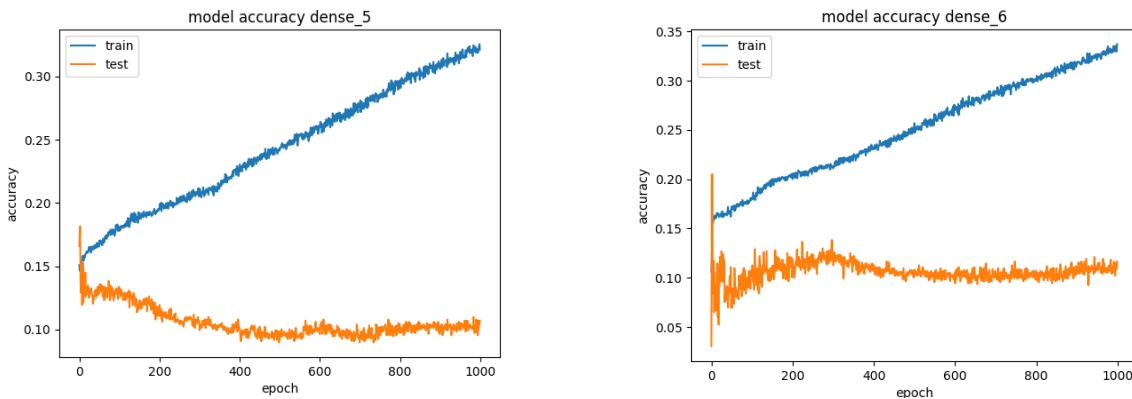


Ilustración 17. Función accuracy de las variables kick_direction y keeper_direction respectivamente.

En la *Ilustración 16* se puede observar cómo aumenta la precisión tanto del conjunto de datos de entrenamiento como del de validación, a medida que el número de *epochs* también aumenta, aunque existe una gran diferencia en la precisión de ambos conjuntos. Por otro lado, en la *Ilustración 17*, se puede apreciar la evolución de la precisión en las dos últimas salidas correspondientes a *kick_direction* y *keeper_direction*, que no evolucionan de la misma manera en este caso, ya que en el conjunto de entrenamiento sí aumenta la precisión según aumenta el número de *epochs*, pero la precisión pasado un número de *epochs*, la precisión comienza a descender.

En la *Ilustración 18* e *Ilustración 19* se observa la función de coste de las cuatro variables de salida, del mismo modo que se ha realizado en el párrafo anterior con la métrica de precisión.

Ahora se puede observar como la pérdida del conjunto de validación es bastante más alta que la del conjunto de entrenamiento y, mientras que la pérdida del conjunto de entrenamiento se reduce a medida que el número de *epochs* avanza, en el conjunto de validación aumenta. Esto es un rasgo claro de *overfitting*, por lo que será un tema para tratar más adelante.

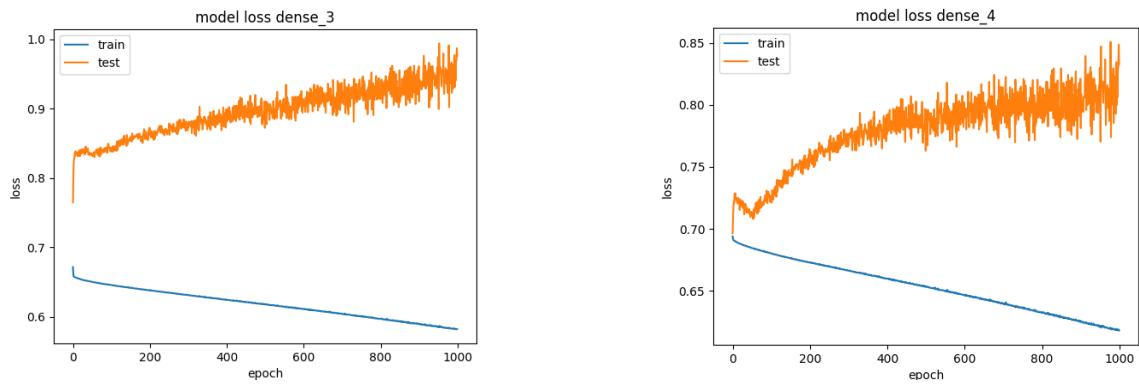


Ilustración 18. Función loss de las variables scored y saved respectivamente.

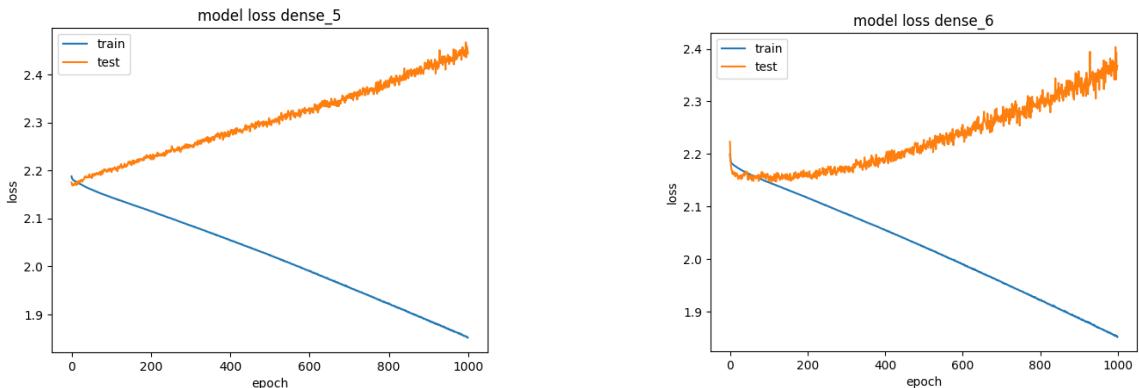


Ilustración 19. Función loss de las variables kick_direction y keeper_direction respectivamente.

Además de estas métricas, también se puede contar con las matrices de confusión. Las matrices de confusión son aquellas matrices que permiten clasificar qué tan bien funciona un modelo de clasificación.

Dado que hay cuatro salidas diferentes, con diferentes métodos de clasificación, las matrices de confusión serán de diferentes tamaños. De hecho, para las dos primeras salidas correspondientes a *scored* y *saved*, como son solo valores binarios, la matriz de confusión será de 2x2 como se puede ver en la *Ilustración 20* e *Ilustración 21*. Como las otras dos salidas tienen nueve posibles valores, las matrices de confusión serán de 9x9.

La implementación de estas matrices de confusión se ha realizado con el conjunto de validación, correspondiente a los datos de prueba.



Ilustración 20. Matriz de confusión y matriz de confusión normalizada de la variable scored.

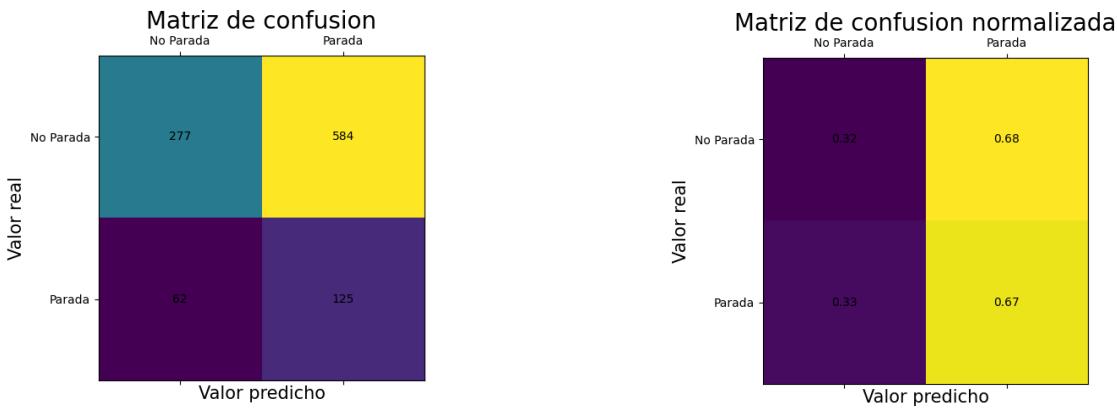


Ilustración 21. Matriz de confusión y matriz de confusión normalizada de la variable saved.

En una matriz de 2x2, los valores correspondientes a la diagonal principal, que va de la parte superior izquierda de la matriz a la inferior derecha, son los valores que el modelo ha acertado, tanto los verdaderos positivos como los falsos negativos. Por el otro lado, la otra diagonal, la diagonal inversa, representan los valores en los que el modelo se ha equivocado, es decir, son los valores que el modelo ha predicho que eran verdaderos o falsos y se ha equivocado, conocidos como falsos negativos y falsos positivos.

Ahora, en el caso de las matrices de confusión correspondientes a las variables de *kick_direction* y *keeper_direction*, el tamaño de ellas va a ser de 9x9 como se puede ver en la *Ilustración 22* e *Ilustración 23*, ya que la clasificación de las variables es de 9 posibles valores. Como en las matrices de 2x2, los valores de la diagonal principal corresponden a los valores que el modelo funciona correctamente, mientras que los valores que están por encima y por debajo de la diagonal principal son valores que el modelo no ha predicho bien.

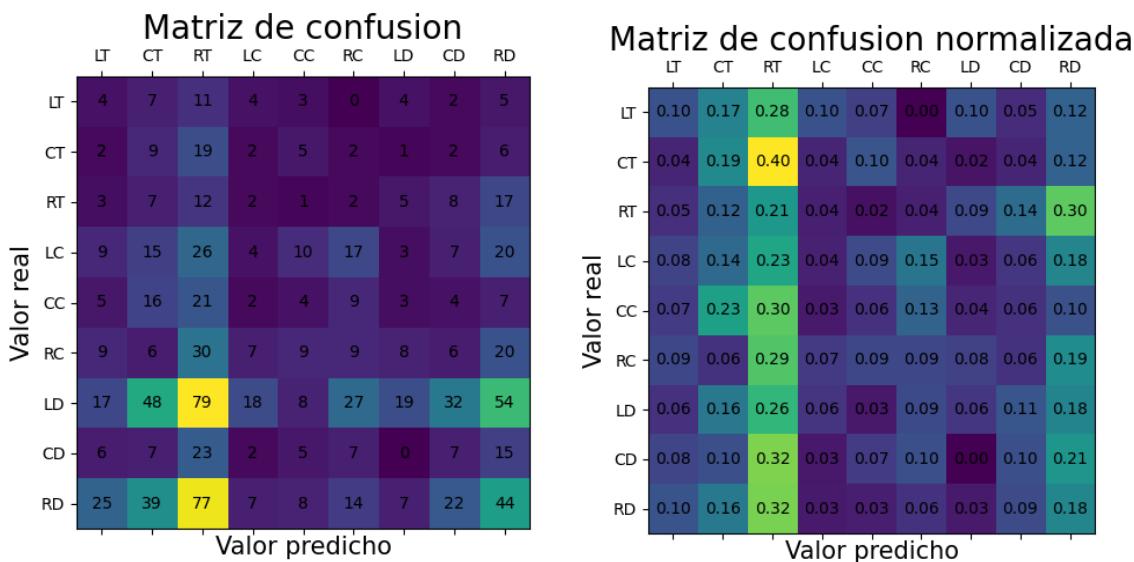


Ilustración 22. Matriz de confusión y matriz de confusión normalizada de la variable kick_direction.

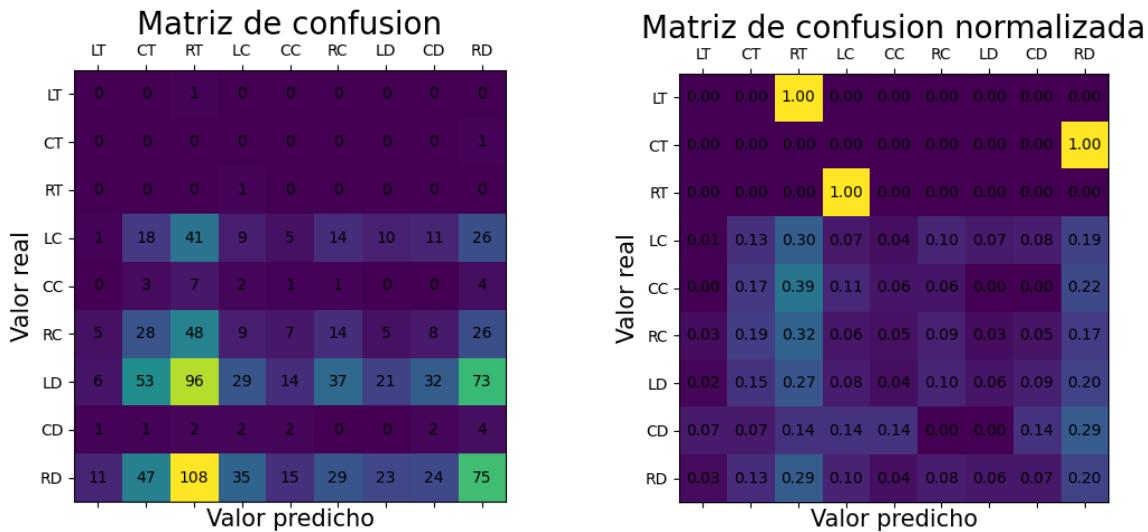


Ilustración 23. Matriz de confusión y matriz de confusión normalizada de la variable keeper_direction.

Cabe destacar que, si el conjunto de datos no está muy bien clasificado o la red neuronal no está bien entrenada o parametrizada, los resultados pueden ser muy dispersos, como es en el caso de las dos ilustraciones anteriores, donde no se ven buenos porcentajes de predicción en la dirección de los penaltis.

Mediante la función *predict* de la librería Keras, se puede realizar predicciones del modelo implementado, compilado y ajustado anteriormente, y con una serie de métricas comprobar su efectividad.

Después, está la métrica *F1_score*, también conocido como Valor-F, es una media ponderada de la precisión. Tiene varios tipos de parámetros: *binary*, *micro*, *macro*, *weighted* y *samples*. En este caso se ha usado *weighted* para *kick_direction* y para *keeper_direction*, ya que devuelve la media considerando la proporción de cada valor de la variable en el dataset. Para *scored* y para *saved* se ha implementado *binary*.

En la Ilustración 24 se puede observar tanto la precisión como el Valor-F, gracias a las funciones *accuracy_score* y *f1_score* para cada salida. Como se puede observar, la precisión obtenido gracias a la función *accuracy_score* es la misma mostrada en la función *evaluate* que evaluaba el modelo.

```
Accuracy columna scored: 0.3139312977099237
F1 Score columna scored: 0.24712041884816754
Accuracy columna saved: 0.383587786259542
F1 Score columna saved: 0.27901785714285715
Accuracy columna kick_direction: 0.10687022900763359
F1 Score columna kick_direction: 0.11374366182797331
Accuracy columna keeper_direction: 0.11641221374045801
F1 Score columna keeper_direction: 0.15358563886655027
```

Ilustración 24. Funciones accuracy_score y f1_score.

Este estudio se ha realizado en todas las pruebas que se han hecho, ya sea modificando hiperparámetros o añadiendo o quitando capas ocultas. La interpretación de resultados se hará en el apartado 4 correspondiente a los resultados.

3.2.4 Reducción de dimensionalidad de la red neuronal

Una de las aplicaciones más implementadas en las redes neuronales es la de reducción de dimensionalidad, debido a que no siempre todas las características de entrada son útiles para el modelo. Además, cuantas más características de entrada, mayor será el tiempo de cómputo, aumento así la probabilidad de *overfitting*.

Es posible que haya casos que, con menos características de entrada de un *dataset*, no se pierde el valor de las características del conjunto de datos. Para ello, hay varios algoritmos de reducción de dimensionalidad, los más usados son:

- **PCA: Principal Component Analysis.** El análisis de componentes principales es la técnica más usada. Este método selecciona las características que definen el conjunto de datos, aquellas cuya influencia en el *dataset* es mayor, conservando esas propiedades originales, reduciendo así el tamaño original del conjunto de datos. Calcula las características principales en base a la varianza de cada característica y la varianza acumulada.

Se ha aplicado esta técnica de dos maneras, dando ambas resultados prácticamente iguales a la hora de definir qué características de entrada son las principales.

En el primer método, lo primero que se va a hacer es mostrar en la *Ilustración 25* la varianza de cada característica y la varianza acumulada.

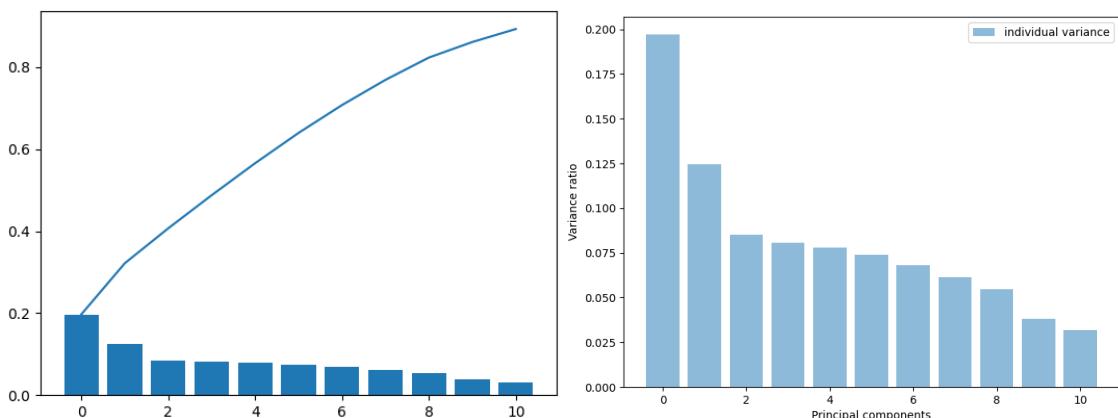


Ilustración 25. Varianza de cada característica de entrada y varianza acumulada.

Como se puede observar, hay una característica de entrada, la primera de todos correspondiente a la variable *season*, que es la mayor característica principal del *dataset*. Va seguida de la segunda característica de entrada, con menor importancia en el *dataset*, y así sucesivamente va teniendo menos influencia la siguiente característica de entrada.

Al haber una gran diferencia entre las dos primeras características de entrada con el resto, y luego una no tan grande diferencia entre las primeras nueve características de entrada y las últimas dos, se han realizado pruebas tanto con dos como con nueve características de entrada.

Respecto al modelo con nueve características de entradas tras aplicar PCA, se observan resultados prácticamente iguales a si se usasen todas las características de entrada, demostrando así que esta técnica funciona

correctamente a la hora de reducir dimensionalidad de 11 a 9 características de entrada.

```
Accuracy columna scored: 0.3320610687022901
F1 Score columna scored: 0.2886178861788618
Accuracy columna saved: 0.41889312977099236
F1 Score columna saved: 0.29595375722543354
Accuracy columna kick_direction: 0.11164122137404581
F1 Score columna kick_direction: 0.1092885368609387
Accuracy columna keeper_direction: 0.13740458015267176
F1 Score columna keeper_direction: 0.16074663020972463
```

Ilustración 26. Accuracy y F1_score tras aplicar PCA y utilizar las 9 características principales.

En comparación con la *Ilustración 26*, se puede observar que los resultados son prácticamente los mismos, incluso mejorando tanto la precisión como el Valor-F del modelo en las dos primeras columnas de salida.

Pero, por otro lado, a la hora de aplicar PCA y utilizar el modelo con las dos primeras características de entrada como se muestra en la *Ilustración 27*, se puede observar que la precisión baja en las dos primeras salidas, llegando incluso a obtener un Valor-F de 0.0 en la columna *scored*. Sin embargo, para las últimas dos columnas los valores siguen siendo prácticamente iguales, y funcionando el modelo de manera similar.

```
Accuracy columna scored: 0.23282442748091603
F1 Score columna scored: 0.0
Accuracy columna saved: 0.26622137404580154
F1 Score columna saved: 0.3078307830783078
Accuracy columna kick_direction: 0.11641221374045801
F1 Score columna kick_direction: 0.09748716453539291
Accuracy columna keeper_direction: 0.1383587786259542
F1 Score columna keeper_direction: 0.15765695284427853
```

Ilustración 27. Accuracy y F1_score tras aplicar PCA y utilizar las 2 características principales.

Aplicando PCA al *dataset* con otro método, se ha obtenido un resultado similar al mencionado anteriormente, se puede comparar esto en la *Ilustración 28*.

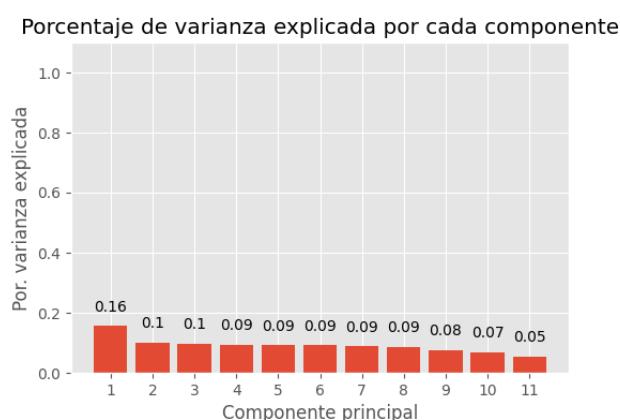


Ilustración 28. Porcentaje de varianza explicada por cada característica de entrada.

De manera similar a la *Ilustración 25*, en este método se puede observar que hay una gran diferencia entre la primera característica y el resto, pero no hay

tanta diferencia a partir de la segunda, ya que hay un rango entre la segunda característica y la octava donde tienen una influencia casi equivalente sobre el conjunto de datos. Con esta forma de implementar PCA, se ha probado con ocho características de entrada en vez de con nueve, tal y como se hacía con el anterior.

```
Accuracy columnna scored: 0.31774809160305345
F1 Score columnna scored: 0.26515930113052416
Accuracy columnna saved: 0.3969465648854962
F1 Score columnna saved: 0.25821596244131456
Accuracy columnna kick_direction: 0.14599236641221375
F1 Score columnna kick_direction: 0.1339101386889027
Accuracy columnna keeper_direction: 0.17270992366412213
F1 Score columnna keeper_direction: 0.17392592311065175
```

Ilustración 29. Accuracy y F1_score tras aplicar PCA y utilizar las 8 características principales.

En comparación con los resultados anteriores, en la Ilustración 29, se puede ver que los resultados son casi los mismos, empeorando un poco los obtenidos en las dos primeras columnas, pero mejorando en las otras dos columnas; y a su vez, casi iguales que si se entrenara el modelo con todas las características de entrada. Así, se puede determinar que aplicar PCA al modelo es posible y, por ende, reducir la dimensionalidad del modelo sin que afecte significativamente al mismo.

- **LDA: Linear Discriminant Analysis.** El análisis discriminante lineal es otra de las principales técnicas usadas en redes neuronales. Su objetivo es el de reconocer patrones que puedan determinar una combinación lineal que represente a todas las características del conjunto de datos.
Respecto a esta técnica de reducción de dimensionalidad, al crear una característica nueva a partir de las originales de una dimensión, no funciona tan correctamente como PCA.

```
Accuracy columnna scored: 0.23282442748091603
F1 Score columnna scored: 0.0
Accuracy columnna saved: 0.22328244274809161
F1 Score columnna saved: 0.3066439522998296
Accuracy columnna kick_direction: 0.06679389312977099
F1 Score columnna kick_direction: 0.031140025971043265
Accuracy columnna keeper_direction: 0.02862595419847328
F1 Score columnna keeper_direction: 0.04406071567995683
```

Ilustración 30. Accuracy y F1_score tras aplicar LDA.

Se puede observar en la Ilustración 30 que los valores son bastante similares a los que mostraba PCA cuando se implementaba con dos o tres características de entrada, ya que se pierde mucha información del *dataset* al intentar agrupar todas las características de entrada en una única.

3.3 Entorno empleado

El ordenador en el que se ha realizado el proyecto, tanto las técnicas de *data augmentation* como el entrenamiento de la red neuronal, tiene las siguientes especificaciones:

- **Sistema operativo:** Microsoft Windows 10 Education N de 64 bits.
- **Procesador:** AMD Ryzen 5 3600 3.6GHz.
- **Memoria RAM:** 2x8GB G. Skill FlareX DDR4 3200.

En cuanto al software utilizado:

- **Visual Studio Code:** es un programa editor de código fuente desarrollado por Microsoft, compatible con múltiples lenguajes de programación. Mediante la instalación de extensiones, se puede usar para ejecutar código en el mismo programa, aunque en este proyecto se ejecutó en el terminal¹⁷ del ordenador.
- **Python:** es un lenguaje de programación multiparadigma, es decir, soporta la orientación a objetos, programación imperativa y programación funcional. Destaca por su característica de código legible. Es ampliamente utilizado en *machine learning*, gracias a algunas de sus librerías como Tensorflow o Keras. La versión de Python utilizada es la 3.8.10.
- **Librerías de Python:** las librerías son un conjunto de módulos y paquetes que permiten acceder a funcionalidades de las mismas, cuyo objetivo es facilitar la programación. Las librerías utilizadas en este proyecto han sido:
 - *random*: es un módulo capaz de generar números pseudoaleatorios.
 - *pandas*: es una librería empleada para el manejo y el análisis de estructuras de datos.
 - *numpy*: es posiblemente la librería más utilizada en Python. Se usa para cálculos numéricos y análisis de datos, empleada principalmente para el manejo de matrices de diferentes dimensiones.
 - *sklearn*: es una librería ampliamente usada en *machine learning* gracias a que poseen gran cantidad de algoritmos de aprendizaje en Python.
 - *tensorflow*: es una librería desarrollada por Google y diseñada para calcular gradientes automáticamente. La librería Keras hace uso de ella.
 - *keras*: es una librería diseñada para la creación y el manejo de redes neuronales.
 - *matplotlib*: es una librería especializada en representar gráficamente diferentes tipos de valores. Sirve para crear diagramas, histogramas o mapas de color, por ejemplo.

¹⁷ El terminal o consola es un programa del sistema operativo cuyo cometido es leer comandos y ejecutar programas.

4. Resultados

En este punto en el que se encuentra ya definido el conjunto de datos y la red neuronal, llega el momento de comprobar su efectividad de cara a predecir los valores deseados. Para ello se han realizado varias pruebas en función del ajuste de los hiperparámetros que se han realizado y de la evolución del proyecto y de la propia red neuronal, ya que las primeras pruebas no corresponden a la fase final del proyecto.

Aunque técnicamente son diferentes redes neuronales debido a que la red de cada prueba tiene diferente ajuste y número de capas, se va a hablar en este apartado como si fuese la misma red neuronal.

Con las métricas mencionadas anteriormente, como las gráficas de *accuracy* y *loss*, las matrices de confusión, el valor de precisión y el Valor-F, se podrá analizar los resultados obtenidos.

4.1 Análisis de los resultados

A lo largo de este análisis, se verá como al principio se obtenían resultados inciertos, como no ser capaz de obtener correctamente la salida de las variables de *kick_direction* y *keeper_direction*, a tener definido una red neuronal funcional, aunque tampoco con buenos resultados.

4.1.1 Primeros resultados

El avance en la obtención de resultados ha sido exponencial, muy costosa y lenta al principio, ya que había gran cantidad de errores en la implementación, pero una vez se implementó una red neuronal funcional, aunque fuera muy simple, se han ido obteniendo más y mejores resultados, y haciendo una red neuronal mejor.

Al principio, se construyó una red neuronal con dos capas de salida de tipo *Dense*, una con función de activación *sigmoid* con dos neuronas de salida, *dense_2* y *dense_3* en la *Ilustración 31*, una correspondiente a la variable *scored* y otra a *saved*; y la otra salida también con dos neuronas la capa con función de activación *softmax* para las variables *kick_direction* y *keeper_direction*.

La función de coste utilizada fue *mean_squared_error* y el optimizador fue *Adam*. Constaba de dos capas ocultas de 16 y 32 neuronas respectivamente dentro del modelo secuencial que hay entre la entrada *Input* y las dos capas de salida descritas previamente.

```

Model: "model"
=====
Layer (type)          Output Shape       Param #     Connected to
=====
input_1 (InputLayer)   [(None, 11)]      0
sequential (Sequential) (None, 32)        736        input_1[0][0]
dense_2 (Dense)        (None, 2)         66         sequential[0][0]
dense_3 (Dense)        (None, 2)         66         sequential[0][0]
=====
Total params: 868
Trainable params: 868
Non-trainable params: 0

```

Ilustración 31. Red neuronal con dos capas de salida.

El problema aquí fue que la función de activación *softmax*, aunque en el conjunto de datos las variables correspondientes a las direcciones estaban clasificadas en 9 clases, al tener una capa de salida de dos neuronas, es decir, dos clases, por la propia definición de *softmax* que devuelve un vector del tamaño de las neuronas de la capa con la probabilidad de que ocurra cada clase, sumando entre todas las probabilidades de cada clase debe sumar 1. Dicho esto, en cada neurona salía un valor, digamos X , y en la otra neurona salía $1 - X$ ya que entre ambos valores debe sumar 1, siendo valores erróneos debido a una mala implementación. Además, los resultados en la capa de salida correspondiente a la función de activación *sigmoid*, tampoco funcionaba correctamente, como se puede ver los valores predichos en la Ilustración a continuación.

```

[[array([[5.2981162e-14, 1.0000000e+00],
       [1.2562023e-13, 1.0000000e+00],
       [9.9328607e-01, 1.0492504e-03],
       [9.9963361e-01, 1.5985966e-04]], dtype=float32), array([[0.545737 , 0.45426297],
       [0.5446049 , 0.45539507],
       [0.63904995, 0.36095002],
       [0.6542644 , 0.34573564]], dtype=float32)],
 [[1.  0.  0.875 1. ],
 [0.  0.  1.  1. ],
 [0.  1.  1.  1. ],
 [1.  0.  0.75 1. ]]]

```

Ilustración 32. Valores predichos y reales.

En la *Ilustración 32* se puede ver que la salida de arriba de la imagen, con dos arrays, un array por cada capa de salida mencionada previamente, y la parte de debajo corresponde a los valores reales. Se puede confirmar lo comentado anteriormente, donde al ser la función de activación *softmax*, si la primera dirección predecía 0.545737, la otra tenía que ser $1 - 0.545737 = 0.454263$.

En cuanto a la salida *sigmoid* correspondiente a si es gol o no, y si es parada o no, como la función *sigmoid* devuelve un valor entre 0 y 1, se ha seguido la siguiente lógica. En la *Tabla 12* se considera que x es el valor real, y z es el valor predicho por la red neuronal.

| Valor real | Valor predicho |
|------------|-------------------------|
| $x = 0$ | $x = 0$ si $z < 0.5$ |
| $x = 1$ | $x = 1$ si $z \geq 0.5$ |

Tabla 12. Determinación del valor predicho en la función de activación sigmoid.

Esta lógica usada en la *Tabla 12*, se aplicará a todos los resultados de este TFG.

Y, tal y como se hizo con la función de activación *sigmoid*, había que hacer una asignación de valores parecidos en la capa de salida *softmax*, definidas dichas asignaciones en la *Tabla 13*.

| Valor real | Valor predicho |
|-------------|---|
| $x = 0$ | $x = 0$ si $0 \leq z < 0.0625$ |
| $x = 0.125$ | $x = 0.125$ si $0.0625 \leq z < 0.1875$ |
| $x = 0.25$ | $x = 0.25$ si $0.1875 \leq z < 0.3125$ |
| $x = 0.375$ | $x = 0.375$ si $0.3125 \leq z < 0.4375$ |
| $x = 0.5$ | $x = 0.5$ si $0.4375 \leq z < 0.5625$ |
| $x = 0.625$ | $x = 0.625$ si $0.5625 \leq z < 0.6875$ |
| $x = 0.75$ | $x = 0.75$ si $0.6875 \leq z < 0.8125$ |
| $x = 0.875$ | $x = 0.875$ si $0.8125 \leq z < 0.9375$ |
| $x = 1$ | $x = 1$ si $z \geq 0.9375$ |

Tabla 13. Determinación del valor predicho en la función de activación softmax.

Aunque la implementación era errónea, los resultados de predicción y de pérdida ya daban signos de *overfitting*, porque, como se puede observar en la Ilustración 35, en el conjunto de entrenamiento hay una precisión de 0.61 y 0.53 para las capas de salida, mientras que en el conjunto de validación hay una precisión de 0.27 y 0.51 respectivamente, tras 100 *epochs*.

De cara a aclarar los resultados que se van a mostrar en este tipo de resultados, como en las siguientes ilustraciones, cabe decir que se van a mostrar las cinco últimas *epochs* de cada entrenamiento, en el cual se ven la *accuracy* y *loss* de cada capa, además de la *loss* del modelo en general. Y, después, se ve la evaluación del modelo, en la cual los resultados corresponden a la evaluación del conjunto de validación del entrenamiento. Los valores que se van a ver en dichas ilustraciones son:

- **loss:** valor de pérdida del modelo en el conjunto de entrenamiento.
- **dense_X_loss:** siendo X el número de capa correspondiente a la salida o una de las salidas, es el valor de pérdida en dicha capa de salida en el conjunto de entrenamiento.
- **dense_X_accuracy:** siendo X el número de capa correspondiente a la salida o una de las salidas, es el valor de precisión de dicha capa de salida en el conjunto de entrenamiento.
- **val_loss:** valor de pérdida del modelo en el conjunto de validación.

- **val_dense_X_loss**: siendo X el número de capa correspondiente a la salida o una de las salidas, es el valor de pérdida en dicha capa de salida en el conjunto de validación.
- **val_dense_X_accuracy**: siendo X el número de capa correspondiente a la salida o una de las salidas, es el valor de precisión en dicha capa de salida en el conjunto de validación.

Como se ha mencionado antes, estos tres últimos valores, *val_loss*, *val_dense_X_loss* y *val_dense_X_accuracy*, serán los mismos que nos mostrará la función *evaluate* mostrada en la *Ilustración 33* que evalúa el modelo sobre el conjunto de validación.

```
Epoch 95/100
99/99 [=====] - 0s 837us/step - loss: 0.3307 - dense_2_loss: 0.2225 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.6128 - dense_3_accuracy: 0.5040
Epoch 96/100
99/99 [=====] - 0s 847us/step - loss: 0.3306 - dense_2_loss: 0.2224 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.6106 - dense_3_accuracy: 0.5269
Epoch 97/100
99/99 [=====] - 0s 817us/step - loss: 0.3306 - dense_2_loss: 0.2224 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.6080 - dense_3_accuracy: 0.5091
Epoch 98/100
99/99 [=====] - 0s 817us/step - loss: 0.3305 - dense_2_loss: 0.2223 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.6093 - dense_3_accuracy: 0.5148
Epoch 99/100
99/99 [=====] - 0s 868us/step - loss: 0.3297 - dense_2_loss: 0.2215 - dense_3_loss: 0.1081 - dense_2_accuracy: 0.6096 - dense_3_accuracy: 0.4970
Epoch 100/100
99/99 [=====] - 0s 908us/step - loss: 0.3299 - dense_2_loss: 0.2218 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.6109 - dense_3_accuracy: 0.5358
val_loss: 0.4301 - val_dense_2_loss: 0.3180 - val_dense_3_loss: 0.1122 - val_dense_2_accuracy: 0.2770 - val_dense_3_accuracy: 0.5033
val_loss: 0.4283 - val_dense_2_loss: 0.3161 - val_dense_3_loss: 0.1122 - val_dense_2_accuracy: 0.2760 - val_dense_3_accuracy: 0.4919
val_loss: 0.4346 - val_dense_2_loss: 0.3226 - val_dense_3_loss: 0.1120 - val_dense_2_accuracy: 0.2684 - val_dense_3_accuracy: 0.4928
val_loss: 0.4400 - val_dense_2_loss: 0.3278 - val_dense_3_loss: 0.1121 - val_dense_2_accuracy: 0.2579 - val_dense_3_accuracy: 0.4947
val_loss: 0.4438 - val_dense_2_loss: 0.3305 - val_dense_3_loss: 0.1134 - val_dense_2_accuracy: 0.2607 - val_dense_3_accuracy: 0.5081
val_loss: 0.4278 - val_dense_2_loss: 0.3151 - val_dense_3_loss: 0.1127 - val_dense_2_accuracy: 0.2751 - val_dense_3_accuracy: 0.5119
Evaluate:
33/33 [=====] - 0s 469us/step - loss: 0.4278 - dense_2_loss: 0.3151 - dense_3_loss: 0.1127 - dense_2_accuracy: 0.2751 - dense_3_accuracy: 0.5119
```

Ilustración 33. Accuracy y loss tras 100 epochs de la red neuronal con dos capas de salida.

Con mayor número de *epochs*, aumentó la precisión de la primera capa de salida, aunque seguía existiendo gran diferencia en la métrica de precisión entre el conjunto de entrenamiento y el de validación. Respecto a la segunda capa de salida, tanto la métrica *loss* como la de *accuracy* se mantenían estables, algo apreciable en la *Ilustración 34*.

```
Epoch 995/1000
99/99 [=====] - 0s 857us/step - loss: 0.2872 - dense_2_loss: 0.1789 - dense_3_loss: 0.1083 - dense_2_accuracy: 0.7267 - dense_3_accuracy: 0.5075
Epoch 996/1000
99/99 [=====] - 0s 857us/step - loss: 0.2869 - dense_2_loss: 0.1786 - dense_3_loss: 0.1083 - dense_2_accuracy: 0.7206 - dense_3_accuracy: 0.5183
Epoch 997/1000
99/99 [=====] - 0s 855us/step - loss: 0.2874 - dense_2_loss: 0.1791 - dense_3_loss: 0.1083 - dense_2_accuracy: 0.7213 - dense_3_accuracy: 0.5021
Epoch 998/1000
99/99 [=====] - 0s 857us/step - loss: 0.2878 - dense_2_loss: 0.1795 - dense_3_loss: 0.1083 - dense_2_accuracy: 0.7226 - dense_3_accuracy: 0.5161
Epoch 999/1000
99/99 [=====] - 0s 1ms/step - loss: 0.2880 - dense_2_loss: 0.1797 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.7229 - dense_3_accuracy: 0.5030 -
Epoch 1000/1000
99/99 [=====] - 0s 888us/step - loss: 0.2877 - dense_2_loss: 0.1793 - dense_3_loss: 0.1083 - dense_2_accuracy: 0.7232 - dense_3_accuracy: 0.5056
- val_loss: 0.5656 - val_dense_2_loss: 0.4524 - val_dense_3_loss: 0.1132 - val_dense_2_accuracy: 0.3095 - val_dense_3_accuracy: 0.5167
- val_loss: 0.5681 - val_dense_2_loss: 0.4544 - val_dense_3_loss: 0.1137 - val_dense_2_accuracy: 0.3142 - val_dense_3_accuracy: 0.5196
- val_loss: 0.5705 - val_dense_2_loss: 0.4562 - val_dense_3_loss: 0.1143 - val_dense_2_accuracy: 0.3181 - val_dense_3_accuracy: 0.5158
- val_loss: 0.5589 - val_dense_2_loss: 0.4459 - val_dense_3_loss: 0.1131 - val_dense_2_accuracy: 0.3286 - val_dense_3_accuracy: 0.5177
val_loss: 0.5787 - val_dense_2_loss: 0.4646 - val_dense_3_loss: 0.1141 - val_dense_2_accuracy: 0.3066 - val_dense_3_accuracy: 0.5177
- val_loss: 0.5460 - val_dense_2_loss: 0.4335 - val_dense_3_loss: 0.1125 - val_dense_2_accuracy: 0.3391 - val_dense_3_accuracy: 0.5205
Evaluate:
33/33 [=====] - 0s 469us/step - loss: 0.5460 - dense_2_loss: 0.4335 - dense_3_loss: 0.1125 - dense_2_accuracy: 0.3391 - dense_3_accuracy: 0.5205
```

Ilustración 34. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida.

También se realizaron pruebas con 10000 epochs, obteniendo el mismo patrón recolectado al aumentar de 100 a 1000 epochs, ya que aumentó un 10% la precisión de la primera capa de salida, llegando incluso a un 0.80 de precisión en el conjunto de entrenamiento, aumentando también la pérdida; pero, la segunda capa de salida manteniéndose prácticamente igual.

Aún con estas dos capas de salida, se probaron las técnicas de reducción de dimensionalidad mencionadas en el apartado 3.2.4, tanto PCA como LDA, y ya se podía observar lo comentado en dicho apartado.

Respecto a la técnica PCA, ya en estas dos capas de salida y sin ser el modelo definitivo, se podía observar que utilizando las nueve características de entrada principales, la precisión y la pérdida eran prácticamente idénticas a que si la red estuviese entrenándose con todas las características de entrada apreciable en la *Ilustración 35*, es decir, utilizando las once características de entrada.

```

Epoch 95/100
99/99 [=====] - 0s 827us/step - loss: 0.3341 - dense_2_loss: 0.2260 - dense_3_loss: 0.1081 - dense_2_accuracy: 0.5982 - dense_3_accuracy: 0.5129
Epoch 96/100
99/99 [=====] - 0s 837us/step - loss: 0.3343 - dense_2_loss: 0.2262 - dense_3_loss: 0.1081 - dense_2_accuracy: 0.5985 - dense_3_accuracy: 0.4798
Epoch 97/100
99/99 [=====] - 0s 827us/step - loss: 0.3336 - dense_2_loss: 0.2256 - dense_3_loss: 0.1080 - dense_2_accuracy: 0.5985 - dense_3_accuracy: 0.4973
Epoch 98/100
99/99 [=====] - 0s 827us/step - loss: 0.3333 - dense_2_loss: 0.2252 - dense_3_loss: 0.1081 - dense_2_accuracy: 0.5918 - dense_3_accuracy: 0.4868
Epoch 99/100
99/99 [=====] - 0s 827us/step - loss: 0.3343 - dense_2_loss: 0.2262 - dense_3_loss: 0.1081 - dense_2_accuracy: 0.5927 - dense_3_accuracy: 0.5154
Epoch 100/100
99/99 [=====] - 0s 827us/step - loss: 0.3339 - dense_2_loss: 0.2257 - dense_3_loss: 0.1082 - dense_2_accuracy: 0.5883 - dense_3_accuracy: 0.4763
val_loss: 0.4220 - val_dense_2_loss: 0.3103 - val_dense_3_loss: 0.1117 - val_dense_2_accuracy: 0.3190 - val_dense_3_accuracy: 0.4823
val_loss: 0.4342 - val_dense_2_loss: 0.3220 - val_dense_3_loss: 0.1122 - val_dense_2_accuracy: 0.2703 - val_dense_3_accuracy: 0.4986
val_loss: 0.4568 - val_dense_2_loss: 0.3446 - val_dense_3_loss: 0.1121 - val_dense_2_accuracy: 0.2426 - val_dense_3_accuracy: 0.4976
val_loss: 0.4069 - val_dense_2_loss: 0.2940 - val_dense_3_loss: 0.1129 - val_dense_2_accuracy: 0.3582 - val_dense_3_accuracy: 0.5253
val_loss: 0.4347 - val_dense_2_loss: 0.3228 - val_dense_3_loss: 0.1120 - val_dense_2_accuracy: 0.2818 - val_dense_3_accuracy: 0.4833
val_loss: 0.4187 - val_dense_2_loss: 0.3056 - val_dense_3_loss: 0.1131 - val_dense_2_accuracy: 0.3152 - val_dense_3_accuracy: 0.5234
Evaluate:
33/33 [=====] - 0s 469us/step - loss: 0.4187 - dense_2_loss: 0.3056 - dense_3_loss: 0.1131 - dense_2_accuracy: 0.3152 - dense_3_accuracy: 0.5234

```

Ilustración 35. Accuracy y loss tras 100 epochs de la red neuronal con dos capas de salida aplicando PCA y utilizando 9 componentes principales.

Como se puede ver, los resultados aplicando PCA prácticamente coinciden a los vistos en la *Ilustración 33*, tanto en el conjunto de entrenamiento como en el conjunto de datos. Al igual que se comentó en el apartado 3.2.4, se probaron dos métodos de PCA y en el primero había una gran diferencia entre las dos primeras características de entrada y el resto (*Ilustración 25*), y también en esta red neuronal inicial baja considerablemente la precisión (*Ilustración 36*).

```

Epoch 995/1000
99/99 [=====] - 0s 827us/step - loss: 0.3463 - dense_2_loss: 0.2376 - dense_3_loss: 0.1088 - dense_2_accuracy: 0.5422 - dense_3_accuracy: 0.5017
Epoch 996/1000
99/99 [=====] - 0s 806us/step - loss: 0.3463 - dense_2_loss: 0.2375 - dense_3_loss: 0.1088 - dense_2_accuracy: 0.5437 - dense_3_accuracy: 0.5187
Epoch 997/1000
99/99 [=====] - 0s 827us/step - loss: 0.3462 - dense_2_loss: 0.2374 - dense_3_loss: 0.1088 - dense_2_accuracy: 0.5434 - dense_3_accuracy: 0.4776
Epoch 998/1000
99/99 [=====] - 0s 817us/step - loss: 0.3463 - dense_2_loss: 0.2375 - dense_3_loss: 0.1088 - dense_2_accuracy: 0.5441 - dense_3_accuracy: 0.5167
Epoch 999/1000
99/99 [=====] - 0s 959us/step - loss: 0.3463 - dense_2_loss: 0.2375 - dense_3_loss: 0.1087 - dense_2_accuracy: 0.5444 - dense_3_accuracy: 0.5482
Epoch 1000/1000
99/99 [=====] - 0s 817us/step - loss: 0.3464 - dense_2_loss: 0.2375 - dense_3_loss: 0.1088 - dense_2_accuracy: 0.5399 - dense_3_accuracy: 0.4681
val_loss: 0.4184 - val_dense_2_loss: 0.3064 - val_dense_3_loss: 0.1120 - val_dense_2_accuracy: 0.2178 - val_dense_3_accuracy: 0.4967
val_loss: 0.4152 - val_dense_2_loss: 0.3031 - val_dense_3_loss: 0.1121 - val_dense_2_accuracy: 0.2283 - val_dense_3_accuracy: 0.5014
val_loss: 0.4205 - val_dense_2_loss: 0.3083 - val_dense_3_loss: 0.1122 - val_dense_2_accuracy: 0.2139 - val_dense_3_accuracy: 0.5043
val_loss: 0.4142 - val_dense_2_loss: 0.3022 - val_dense_3_loss: 0.1119 - val_dense_2_accuracy: 0.2159 - val_dense_3_accuracy: 0.4852
val_loss: 0.4186 - val_dense_2_loss: 0.3073 - val_dense_3_loss: 0.1113 - val_dense_2_accuracy: 0.2159 - val_dense_3_accuracy: 0.4737
val_loss: 0.4171 - val_dense_2_loss: 0.3056 - val_dense_3_loss: 0.1115 - val_dense_2_accuracy: 0.2139 - val_dense_3_accuracy: 0.4919
Evaluate:
33/33 [=====] - 0s 469us/step - loss: 0.4171 - dense_2_loss: 0.3056 - dense_3_loss: 0.1115 - dense_2_accuracy: 0.2139 - dense_3_accuracy: 0.4919

```

Ilustración 36. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida aplicando PCA y utilizando 2 componentes principales.

Con la técnica de reducción de dimensionalidad LDA, ocurre exactamente lo mismo, como con PCA, a lo comentado en el apartado 3.2.4, ya que a la propia naturaleza de este método, que reduce todas las características de entrada a una única característica, se reduce la precisión como se puede ver en la *Ilustración 37*.

```

Epoch 995/1000
99/99 [=====] - 0s 837us/step - loss: 0.3483 - dense_2_loss: 0.2394 - dense_3_loss: 0.1089 - dense_2_accuracy: 0.5332 - dense_3_accuracy: 0.4919
Epoch 996/1000
99/99 [=====] - 0s 857us/step - loss: 0.3483 - dense_2_loss: 0.2394 - dense_3_loss: 0.1089 - dense_2_accuracy: 0.5332 - dense_3_accuracy: 0.4496
Epoch 997/1000
99/99 [=====] - 0s 786us/step - loss: 0.3483 - dense_2_loss: 0.2394 - dense_3_loss: 0.1089 - dense_2_accuracy: 0.5332 - dense_3_accuracy: 0.5840
Epoch 998/1000
99/99 [=====] - 0s 847us/step - loss: 0.3484 - dense_2_loss: 0.2394 - dense_3_loss: 0.1089 - dense_2_accuracy: 0.5332 - dense_3_accuracy: 0.4728
Epoch 999/1000
99/99 [=====] - 0s 837us/step - loss: 0.3482 - dense_2_loss: 0.2393 - dense_3_loss: 0.1089 - dense_2_accuracy: 0.5332 - dense_3_accuracy: 0.4057
Epoch 1000/1000
99/99 [=====] - 0s 847us/step - loss: 0.3483 - dense_2_loss: 0.2394 - dense_3_loss: 0.1089 - dense_2_accuracy: 0.5332 - dense_3_accuracy: 0.5145
val_loss: 0.4023 - val_dense_2_loss: 0.2901 - val_dense_3_loss: 0.1122 - val_dense_2_accuracy: 0.1777 - val_dense_3_accuracy: 0.5549
val_loss: 0.4062 - val_dense_2_loss: 0.2938 - val_dense_3_loss: 0.1124 - val_dense_2_accuracy: 0.1777 - val_dense_3_accuracy: 0.5540
val_loss: 0.4083 - val_dense_2_loss: 0.2961 - val_dense_3_loss: 0.1122 - val_dense_2_accuracy: 0.1777 - val_dense_3_accuracy: 0.5540
val_loss: 0.4037 - val_dense_2_loss: 0.2916 - val_dense_3_loss: 0.1121 - val_dense_2_accuracy: 0.1777 - val_dense_3_accuracy: 0.5186
val_loss: 0.4067 - val_dense_2_loss: 0.2943 - val_dense_3_loss: 0.1124 - val_dense_2_accuracy: 0.1777 - val_dense_3_accuracy: 0.5540
val_loss: 0.4034 - val_dense_2_loss: 0.2913 - val_dense_3_loss: 0.1121 - val_dense_2_accuracy: 0.1777 - val_dense_3_accuracy: 0.4776
Evaluate:
33/33 [=====] - 0s 438us/step - loss: 0.4034 - dense_2_loss: 0.2913 - dense_3_loss: 0.1121 - dense_2_accuracy: 0.1777 - dense_3_accuracy: 0.4776

```

Ilustración 37. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida aplicando LDA.

4.1.2 Evolución de la red neuronal

Después, se probó a añadir una capa oculta más, con diferente número de neuronas buscando una mayor precisión y una menor pérdida en el modelo, aunque sin modificar aún las dos capas de salida.

Como ya había dos capas ocultas de 16 y 32 neuronas respectivamente, la tercera se probó con diferente número de neuronas. En concreto, con 32 y 64 neuronas.

```

Epoch 995/1000
99/99 [=====] - 0s 868us/step - loss: 0.1935 - dense_3_loss: 0.0851 - dense_4_loss: 0.1085 - dense_3_accuracy: 0.8864 - dense_4_accuracy: 0.5119
Epoch 996/1000
99/99 [=====] - 0s 857us/step - loss: 0.1957 - dense_3_loss: 0.0873 - dense_4_loss: 0.1084 - dense_3_accuracy: 0.8801 - dense_4_accuracy: 0.5492
Epoch 997/1000
99/99 [=====] - 0s 888us/step - loss: 0.1957 - dense_3_loss: 0.0871 - dense_4_loss: 0.1085 - dense_3_accuracy: 0.8785 - dense_4_accuracy: 0.5288
Epoch 998/1000
99/99 [=====] - 0s 888us/step - loss: 0.1942 - dense_3_loss: 0.0858 - dense_4_loss: 0.1084 - dense_3_accuracy: 0.8851 - dense_4_accuracy: 0.5326
Epoch 999/1000
99/99 [=====] - 0s 929us/step - loss: 0.1946 - dense_3_loss: 0.0862 - dense_4_loss: 0.1084 - dense_3_accuracy: 0.8772 - dense_4_accuracy: 0.5180
Epoch 1000/1000
99/99 [=====] - 0s 898us/step - loss: 0.1946 - dense_3_loss: 0.0861 - dense_4_loss: 0.1085 - dense_3_accuracy: 0.8804 - dense_4_accuracy: 0.5272
val_loss: 0.5771 - val_dense_3_loss: 0.4644 - val_dense_4_loss: 0.1127 - val_dense_3_accuracy: 0.4718 - val_dense_4_accuracy: 0.5043
val_loss: 0.6221 - val_dense_3_loss: 0.5089 - val_dense_4_loss: 0.1133 - val_dense_3_accuracy: 0.4126 - val_dense_4_accuracy: 0.4881
val_loss: 0.6160 - val_dense_3_loss: 0.5032 - val_dense_4_loss: 0.1129 - val_dense_3_accuracy: 0.4155 - val_dense_4_accuracy: 0.5024
val_loss: 0.6062 - val_dense_3_loss: 0.4943 - val_dense_4_loss: 0.1119 - val_dense_3_accuracy: 0.4336 - val_dense_4_accuracy: 0.4804
val_loss: 0.6014 - val_dense_3_loss: 0.4883 - val_dense_4_loss: 0.1132 - val_dense_3_accuracy: 0.4384 - val_dense_4_accuracy: 0.5014
val_loss: 0.6288 - val_dense_3_loss: 0.5148 - val_dense_4_loss: 0.1140 - val_dense_3_accuracy: 0.4117 - val_dense_4_accuracy: 0.5072
Evaluate:
33/33 [=====] - 0s 500us/step - loss: 0.6288 - dense_3_loss: 0.5148 - dense_4_loss: 0.1140 - dense_3_accuracy: 0.4117 - dense_4_accuracy: 0.5072

```

Ilustración 38. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida y una tercera capa oculta de 32 neuronas.

```

Epoch 9995/10000
99/99 [=====] - 0s 919us/step - loss: 0.1690 - dense_3_loss: 0.0606 - dense_4_loss: 0.1084 - dense_3_accuracy: 0.9135 - dense_4_accuracy: 0.5374
Epoch 9996/10000
99/99 [=====] - 0s 949us/step - loss: 0.1658 - dense_3_loss: 0.0577 - dense_4_loss: 0.1081 - dense_3_accuracy: 0.9150 - dense_4_accuracy: 0.5056
Epoch 9997/10000
99/99 [=====] - 0s 990us/step - loss: 0.1676 - dense_3_loss: 0.0593 - dense_4_loss: 0.1083 - dense_3_accuracy: 0.9125 - dense_4_accuracy: 0.5332
Epoch 9998/10000
99/99 [=====] - 0s 939us/step - loss: 0.1693 - dense_3_loss: 0.0614 - dense_4_loss: 0.1078 - dense_3_accuracy: 0.9131 - dense_4_accuracy: 0.5250
Epoch 9999/10000
99/99 [=====] - 0s 950us/step - loss: 0.1697 - dense_3_loss: 0.0614 - dense_4_loss: 0.1083 - dense_3_accuracy: 0.9138 - dense_4_accuracy: 0.5262
Epoch 10000/10000
99/99 [=====] - 0s 944us/step - loss: 0.1658 - dense_3_loss: 0.0577 - dense_4_loss: 0.1081 - dense_3_accuracy: 0.9166 - dense_4_accuracy: 0.5348
val_loss: 0.6436 - val_dense_3_loss: 0.5301 - val_dense_4_loss: 0.1136 - val_dense_3_accuracy: 0.4556 - val_dense_4_accuracy: 0.5119
val_loss: 0.6582 - val_dense_3_loss: 0.5444 - val_dense_4_loss: 0.1138 - val_dense_3_accuracy: 0.4422 - val_dense_4_accuracy: 0.5005
val_loss: 0.6651 - val_dense_3_loss: 0.5535 - val_dense_4_loss: 0.1116 - val_dense_3_accuracy: 0.4269 - val_dense_4_accuracy: 0.4690
val_loss: 0.6642 - val_dense_3_loss: 0.5531 - val_dense_4_loss: 0.1111 - val_dense_3_accuracy: 0.4327 - val_dense_4_accuracy: 0.4690
val_loss: 0.6518 - val_dense_3_loss: 0.5398 - val_dense_4_loss: 0.1120 - val_dense_3_accuracy: 0.4384 - val_dense_4_accuracy: 0.4728
val_loss: 0.6580 - val_dense_3_loss: 0.5413 - val_dense_4_loss: 0.1167 - val_dense_3_accuracy: 0.4422 - val_dense_4_accuracy: 0.4995
Evaluate:
33/33 [=====] - 0s 500us/step - loss: 0.6580 - dense_3_loss: 0.5413 - dense_4_loss: 0.1167 - dense_3_accuracy: 0.4422 - dense_4_accuracy: 0.4995

```

Ilustración 39. Accuracy y loss tras 10000 con dos capas de salida salida y una tercera capa oculta de 32 neuronas.

Como se puede observar en la *Ilustración 38* y en la *Ilustración 39*, la precisión en el conjunto de entrenamiento llega a aumentar a 0.80 y 0.91, valores en principio muy buenos. En el conjunto de validación también se observa un aumento del 10% de precisión más o menos, pero sigue siendo un valor bajo respecto al conjunto de entrenamiento. Además, según aumenta la precisión de las capas de salida, al menos de la primera capa de salida, también aumenta la pérdida del modelo.

En el caso en que la tercera capa oculta conste de 64 neuronas, en vez de 32 neuronas, apenas se aprecia diferencia en los resultados, aumentando menos de un 5% la precisión en el conjunto de entrenamiento, como se puede ver en la *Ilustración 40* e *Ilustración 41* en las ejecuciones del modelo con 1000 y 10000 epochs respectivamente.

```

Epoch 995/1000
99/99 [=====] - 0s 878us/step - loss: 0.1777 - dense_3_loss: 0.0719 - dense_4_loss: 0.1058 - dense_3_accuracy: 0.8921 - dense_4_accuracy: 0.5539
Epoch 996/1000
99/99 [=====] - 0s 871us/step - loss: 0.1786 - dense_3_loss: 0.0728 - dense_4_loss: 0.1059 - dense_3_accuracy: 0.8861 - dense_4_accuracy: 0.5492
Epoch 997/1000
99/99 [=====] - 0s 878us/step - loss: 0.1780 - dense_3_loss: 0.0722 - dense_4_loss: 0.1057 - dense_3_accuracy: 0.8832 - dense_4_accuracy: 0.5527
Epoch 998/1000
99/99 [=====] - 0s 878us/step - loss: 0.1759 - dense_3_loss: 0.0699 - dense_4_loss: 0.1060 - dense_3_accuracy: 0.8890 - dense_4_accuracy: 0.5466
Epoch 999/1000
99/99 [=====] - 0s 908us/step - loss: 0.1761 - dense_3_loss: 0.0703 - dense_4_loss: 0.1058 - dense_3_accuracy: 0.8906 - dense_4_accuracy: 0.5460
Epoch 1000/1000
99/99 [=====] - 0s 878us/step - loss: 0.1783 - dense_3_loss: 0.0725 - dense_4_loss: 0.1057 - dense_3_accuracy: 0.8918 - dense_4_accuracy: 0.5536
val_loss: 0.6438 - val_dense_3_loss: 0.5270 - val_dense_4_loss: 0.1168 - val_dense_3_accuracy: 0.4126 - val_dense_4_accuracy: 0.5215
val_loss: 0.6571 - val_dense_3_loss: 0.5404 - val_dense_4_loss: 0.1166 - val_dense_3_accuracy: 0.3859 - val_dense_4_accuracy: 0.5272
val_loss: 0.6317 - val_dense_3_loss: 0.5161 - val_dense_4_loss: 0.1156 - val_dense_3_accuracy: 0.4222 - val_dense_4_accuracy: 0.5024
val_loss: 0.6412 - val_dense_3_loss: 0.5253 - val_dense_4_loss: 0.1158 - val_dense_3_accuracy: 0.4050 - val_dense_4_accuracy: 0.4967
val_loss: 0.6462 - val_dense_3_loss: 0.5301 - val_dense_4_loss: 0.1162 - val_dense_3_accuracy: 0.4107 - val_dense_4_accuracy: 0.5033
val_loss: 0.6273 - val_dense_3_loss: 0.5109 - val_dense_4_loss: 0.1164 - val_dense_3_accuracy: 0.4269 - val_dense_4_accuracy: 0.5244
Evaluate:
33/33 [=====] - 0s 469us/step - loss: 0.6273 - dense_3_loss: 0.5109 - dense_4_loss: 0.1164 - dense_3_accuracy: 0.4269 - dense_4_accuracy: 0.5244

```

Ilustración 40. Accuracy y loss tras 1000 epochs de la red neuronal con dos capas de salida y una tercera capa oculta de 64 neuronas.

```

Epoch 9995/10000
99/99 [=====] - 0s 908us/step - loss: 0.1399 - dense_3_loss: 0.0359 - dense_4_loss: 0.1041 - dense_3_accuracy: 0.9262 - dense_4_accuracy: 0.5851
Epoch 9996/10000
99/99 [=====] - 0s 878us/step - loss: 0.1341 - dense_3_loss: 0.0307 - dense_4_loss: 0.1034 - dense_3_accuracy: 0.9294 - dense_4_accuracy: 0.5725
Epoch 9997/10000
99/99 [=====] - 0s 878us/step - loss: 0.1381 - dense_3_loss: 0.0266 - dense_4_loss: 0.1036 - dense_3_accuracy: 0.9325 - dense_4_accuracy: 0.5781
Epoch 9998/10000
99/99 [=====] - 0s 878us/step - loss: 0.1285 - dense_3_loss: 0.0249 - dense_4_loss: 0.1036 - dense_3_accuracy: 0.9348 - dense_4_accuracy: 0.5870
Epoch 9999/10000
99/99 [=====] - 0s 878us/step - loss: 0.1276 - dense_3_loss: 0.0244 - dense_4_loss: 0.1032 - dense_3_accuracy: 0.9332 - dense_4_accuracy: 0.5887
Epoch 10000/10000
99/99 [=====] - 0s 878us/step - loss: 0.1279 - dense_3_loss: 0.0243 - dense_4_loss: 0.1036 - dense_3_accuracy: 0.9354 - dense_4_accuracy: 0.5883
val_loss: 0.6299 - val_dense_3_loss: 0.5101 - val_dense_4_loss: 0.1198 - val_dense_3_accuracy: 0.4670 - val_dense_4_accuracy: 0.5110
val_loss: 0.6557 - val_dense_3_loss: 0.5344 - val_dense_4_loss: 0.1212 - val_dense_3_accuracy: 0.4336 - val_dense_4_accuracy: 0.5234
val_loss: 0.6544 - val_dense_3_loss: 0.5313 - val_dense_4_loss: 0.1230 - val_dense_3_accuracy: 0.4384 - val_dense_4_accuracy: 0.5425
val_loss: 0.6620 - val_dense_3_loss: 0.5407 - val_dense_4_loss: 0.1213 - val_dense_3_accuracy: 0.4288 - val_dense_4_accuracy: 0.5263
val_loss: 0.6498 - val_dense_3_loss: 0.5306 - val_dense_4_loss: 0.1193 - val_dense_3_accuracy: 0.4394 - val_dense_4_accuracy: 0.5110
val_loss: 0.6556 - val_dense_3_loss: 0.5352 - val_dense_4_loss: 0.1204 - val_dense_3_accuracy: 0.4308 - val_dense_4_accuracy: 0.5062
Evaluate:
33/33 [=====] - 0s 500us/step - loss: 0.6556 - dense_3_loss: 0.5352 - dense_4_loss: 0.1204 - dense_3_accuracy: 0.4308 - dense_4_accuracy: 0.5062

```

Ilustración 41. Accuracy y loss tras 10000 epochs de la red neuronal con dos capas de salida y una tercera capa oculta de 64 neuronas.

En todos estos resultados, tanto de este apartado como del anterior, se puede observar lo comentado anteriormente de que no estaba implementado correctamente la capa de salida de las direcciones del penalti, ya que la pérdida y la precisión se mantenía prácticamente igual fuera cual fuera la configuración de la red neuronal.

4.1.3 Afinando la red neuronal

Visto que el modelo funciona mejor con tres capas ocultas que con dos, se decidió implementar en este punto del proyecto tres capas ocultas de 16 y 32 neuronas en las dos primeras capas, y de un número diferente de neuronas en la tercera capa oculta en cada prueba.

Además, se comenzaron a hacer pruebas con cuatro capas de salida, una por cada variable a predecir por el modelo. Las dos primeras capas de salida tienen función

de activación *sigmoid*, para las variables *scored* y *saved*, con un único valor de salida por cada capa. Las dos últimas capas de salida tienen función de activación *softmax*, con nueve neuronas cada capa, correspondiente a las nueve zonas definidas dentro de una portería, pudiéndose observar esto en la *Ilustración 42*.

| Model: "model" | | | |
|-------------------------|--------------|---------|------------------|
| Layer (type) | Output Shape | Param # | Connected to |
| input_1 (InputLayer) | [None, 11] | 0 | |
| sequential (Sequential) | (None, 256) | 17920 | input_1[0][0] |
| dense_3 (Dense) | (None, 1) | 257 | sequential[0][0] |
| dense_4 (Dense) | (None, 1) | 257 | sequential[0][0] |
| dense_5 (Dense) | (None, 9) | 2313 | sequential[0][0] |
| dense_6 (Dense) | (None, 9) | 2313 | sequential[0][0] |

Total params: 23,060
Trainable params: 23,060
Non-trainable params: 0

Ilustración 42. Modelo de la red neuronal con cuatro capas de salida.

También, a las métricas de *accuracy* y *loss* vistas anteriormente, se añadieron métricas como matrices de confusión y gráficos representando los valores de *accuracy* y *loss* según avanza el número de *epochs*, por cada capa de salida, además de la métrica del Valor-F.

Respecto al Valor-F, tal y como se comentó en el apartado 3.2.3, la métrica del Valor-F para las dos primeras capas de salida de tipo *sigmoid* se ha usado la media de tipo *binary* y para las dos últimas capas de tipo *softmax* se ha usado la media de tipo *weighted*.

Adicionalmente, se probaron también las funciones de pérdida *binary_crossentropy* y *categorical_crossentropy*, la primera para las dos primeras capas de salida, y la última para las dos últimas capas de salida. Esto es debido a que en las variables *scored* y *saved*, solo vamos a querer valores binarios, 0 ó 1; y, por el otro lado, en las variables *kick_direction* y *keeper_direction*, se desea que devuelva un vector con nueve clases. También se le añadió una tasa de aprendizaje al ya utilizado anteriormente optimizador *Adam*, de 0.0001 o 0.0002.

Respecto al tamaño de lote, se ha decidido que la red neuronal no entrenara por lotes debido a que las mejoras de dividir el conjunto de entrenamiento en lotes no son palpables y, como el conjunto de datos de entrenamiento no es grande tampoco, no hay grandes problemas de tiempo de cómputo del entrenamiento del modelo. Por ejemplo, para una red neuronal con tres capas ocultas de 16, 64 y 256 neuronas respectivamente, con optimizador *Adam* con tasa de aprendizaje de 0.0001, 1000 *epochs*, función de coste *binary_crossentropy* y *categorical_crossentropy*, veamos la diferencia entre no aplicar el tamaño de lote y usar un tamaño de lote de 64 en la *Ilustración 43* e *Ilustración 44*.

```
Accuracy columna scored: 0.3139312977099237
F1 Score columna scored: 0.24712041884816754
Accuracy columna saved: 0.383587786259542
F1 Score columna saved: 0.27901785714285715
Accuracy columna kick_direction: 0.10687022900763359
F1 Score columna kick_direction: 0.11374366182797331
Accuracy columna keeper_direction: 0.11641221374045801
F1 Score columna keeper_direction: 0.15358563886655027
```

Ilustración 43. Accuracy y F1_score aplicando batch_size=64.

```

Accuracy columna scored: 0.3616412213740458
F1 Score columna scored: 0.3610315186246418
Accuracy columna saved: 0.41889312977099236
F1 Score columna saved: 0.26181818181818184
Accuracy columna kick_direction: 0.09923664122137404
F1 Score columna kick_direction: 0.10371299461618304
Accuracy columna keeper_direction: 0.12022900763358779
F1 Score columna keeper_direction: 0.1566766924478552

```

Ilustración 44. Accuracy y F1_score sin aplicar batch_size.

Como se puede observar, las dos últimas capas de salida tienen valores parecidos, pero en las dos primeras capas de salida aumenta la precisión de ambas, y aumenta también el Valor-F considerablemente en la primera columna de salida.

Aunque hay algo de diferencia, tampoco es muy apreciable, y debido a que dividir el entrenamiento en lotes aumentaba el tiempo de cómputo, se decidió no aplicar este hiperparámetro.

Ahora, toca evaluar el comportamiento del modelo.

4.1.4 Modelo con tres capas ocultas

Modelo con tres capas ocultas de 16, 64 y 256 neuronas respectivamente, cuatro capas de salida con función de coste *binary_crossentropy* para las dos primeras y *categorical_crossentropy* para las dos últimas, 1000 epochs, optimizador *Adam* con tasa de aprendizaje de 0.0001.

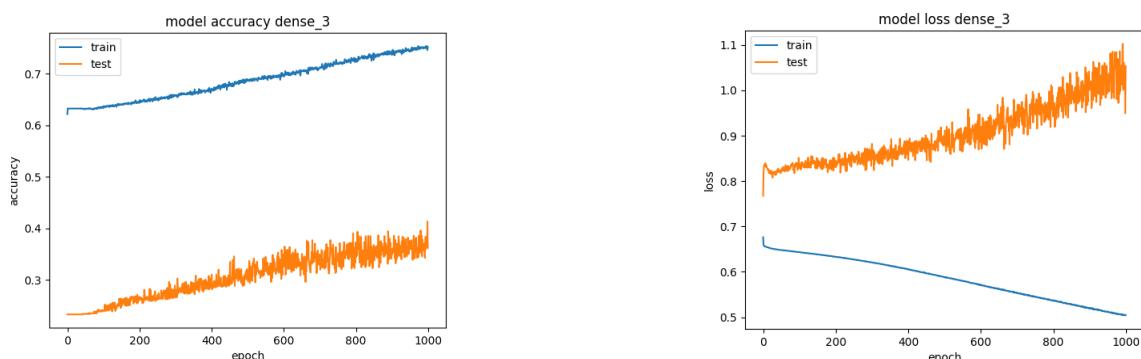


Ilustración 45. Función accuracy y loss de la primera capa de salida (scored)

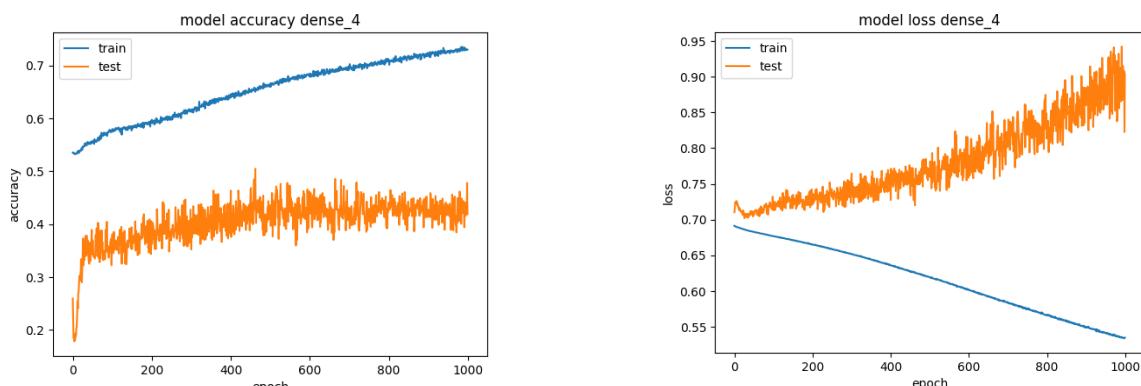


Ilustración 46. Función accuracy y loss de la segunda capa de salida (saved)

En el caso de las dos primeras capas de salida, la *Ilustración 45* y la *Ilustración 46*, se puede ver como la precisión de esa capa el conjunto de entrenamiento tiene tiene un porcentaje alto, llegando a superar el 70% de precisión, pero en el conjunto de validación o de test, apenas llega al 40% de precisión, esto es debido a que el modelo ha sido capaz de entrenar bien los datos que previamente conocía y se le ha suministrado, pero a la hora de predecir nuevos valores no es capaz de reconocer patrones y de ahí la diferencia de precisión. Además, respecto a la métrica de pérdida o *loss*, se puede ver que ambas salidas en el conjunto de validación tienen su valor mínimo de pérdida pasado muy pocos *epochs* y después no para de aumentar, mientras en el conjunto de entrenamiento la pérdida se reduce según avanza el número de *epochs* ajustándose a los datos de entrenamiento, ajuste que no ocurre en el conjunto de validación.

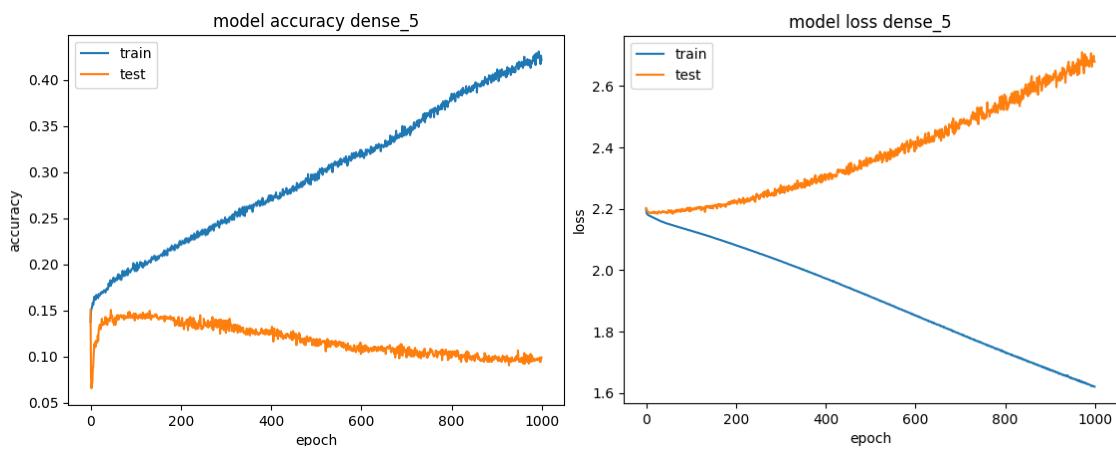


Ilustración 47. Función accuracy y loss de la tercera capa de salida (kick_direction)

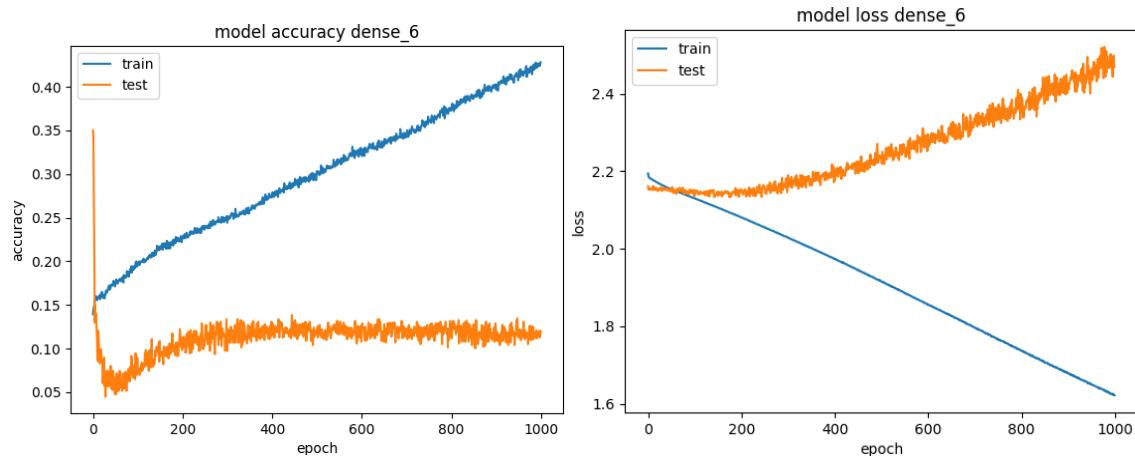


Ilustración 48. Función accuracy y loss de la cuarta capa de salida (keeper_direction)

Respecto a la *Ilustración 47* e *Ilustración 48*, correspondientes a las dos últimas salidas de la red neuronal de las variables *kick_direction* y *keeper_direction*, también se aprecian rasgos de claro *overfitting*. La precisión de ambas variables llegado entorno al 10-15% de precisión, se mantiene constante e incluso baja la precisión en la tercera capa de salida, mientras que en el entrenamiento aumenta linealmente. En la métrica de pérdida ocurre algo parecido a lo comentado en las dos primeras variables: el valor mínimo de pérdida de los conjuntos de validación se alcanzan en los primeros *epochs*, mientras que en los conjuntos de entrenamiento la pérdida se reduce a medida que aumentan el número de *epochs*.

En las cuatro anteriores ilustraciones, se han podido observar problemas que ya se habían podido detectar simplemente viendo los resultados de la red neuronal. Pero ahora, de manera gráfica, se aprecia *overfitting* en el modelo.

En la *Ilustración 49* e *Ilustración 50* se puede apreciar mejor lo que nos muestra los resultados de *accuracy*, *loss* y el Valor-F.

```
Epoch 995/1000
99/99 [=====] - 0s 1ms/step - loss: 4.2896 - dense_3_loss: 0.5053 - dense_4_loss: 0.5352 - dense_5_loss: 1.6237 - dense_6_loss: 1.6254 - dense_3_accuracy: 0.7549 - dense_4_accuracy: 0.7323 - dense_5_accuracy: 0.4243 - dense_6_accuracy: 0.4233 - val_loss: 7.1551 - va
l_dense_3_loss: 1.0609 - val_dense_4_loss: 0.9149 - val_dense_5_loss: 2.6902 - val_dense_6_loss: 2.4891 - val_dense_3_accuracy: 0.3687 - val_dense_4_accuracy: 0.4160 - val_dense_5_accuracy: 0.0973 - val_dense_6_accuracy: 0.1155
Epoch 996/1000
99/99 [=====] - 0s 1ms/step - loss: 4.2861 - dense_3_loss: 0.5051 - dense_4_loss: 0.5349 - dense_5_loss: 1.6226 - dense_6_loss: 1.6234 - dense_3_accuracy: 0.7514 - dense_4_accuracy: 0.7295 - dense_5_accuracy: 0.4236 - dense_6_accuracy: 0.4268 - val
l_dense_3_loss: 1.0400 - val_dense_4_loss: 0.8847 - val_dense_5_loss: 2.6854 - val_dense_6_loss: 2.4437 - val_dense_3_accuracy: 0.3655 - val_dense_4_accuracy: 0.4380 - val_dense_5_accuracy: 0.0992 - val_dense_6_accuracy: 0.1202
Epoch 997/1000
99/99 [=====] - 0s 1ms/step - loss: 4.2858 - dense_3_loss: 0.5045 - dense_4_loss: 0.5344 - dense_5_loss: 1.6224 - dense_6_loss: 1.6245 - dense_3_accuracy: 0.7588 - dense_4_accuracy: 0.7295 - dense_5_accuracy: 0.4262 - dense_6_accuracy: 0.4265 - val
l_dense_3_loss: 1.0105 - val_dense_4_loss: 0.8743 - val_dense_5_loss: 2.6954 - val_dense_6_loss: 2.4947 - val_dense_3_accuracy: 0.3779 - val_dense_4_accuracy: 0.4370 - val_dense_5_accuracy: 0.0945 - val_dense_6_accuracy: 0.1145
Epoch 998/1000
99/99 [=====] - 0s 1ms/step - loss: 4.2830 - dense_3_loss: 0.5039 - dense_4_loss: 0.5340 - dense_5_loss: 1.6215 - dense_6_loss: 1.6236 - dense_3_accuracy: 0.7527 - dense_4_accuracy: 0.7381 - dense_5_accuracy: 0.4173 - dense_6_accuracy: 0.4239 - val
l_dense_3_loss: 1.0446 - val_dense_4_loss: 0.9078 - val_dense_5_loss: 2.6985 - val_dense_6_loss: 2.4981 - val_dense_3_accuracy: 0.3635 - val_dense_4_accuracy: 0.4227 - val_dense_5_accuracy: 0.0945 - val_dense_6_accuracy: 0.1126
Epoch 999/1000
99/99 [=====] - 0s 1ms/step - loss: 4.2837 - dense_3_loss: 0.5051 - dense_4_loss: 0.5350 - dense_5_loss: 1.6210 - dense_6_loss: 1.6227 - dense_3_accuracy: 0.7494 - dense_4_accuracy: 0.7304 - dense_5_accuracy: 0.4265 - dense_6_accuracy: 0.4268 - val
l_dense_3_loss: 0.9492 - val_dense_4_loss: 0.8227 - val_dense_5_loss: 2.6785 - val_dense_6_loss: 2.4680 - val_dense_3_accuracy: 0.4132 - val_dense_4_accuracy: 0.4781 - val_dense_5_accuracy: 0.0973 - val_dense_6_accuracy: 0.1145
Epoch 1000/1000
99/99 [=====] - 0s 1ms/step - loss: 4.2822 - dense_3_loss: 0.5048 - dense_4_loss: 0.5351 - dense_5_loss: 1.6205 - dense_6_loss: 1.6218 - dense_3_accuracy: 0.7521 - dense_4_accuracy: 0.7301 - dense_5_accuracy: 0.4217 - dense_6_accuracy: 0.4281 - val
l_dense_3_loss: 1.0532 - val_dense_4_loss: 0.9034 - val_dense_5_loss: 2.6789 - val_dense_6_loss: 2.4695 - val_dense_3_accuracy: 0.3616 - val_dense_4_accuracy: 0.4189 - val_dense_5_accuracy: 0.0992 - val_dense_6_accuracy: 0.1202
Evaluate:
33/33 [=====] - 0s 424us/step - loss: 7.1850 - dense_3_loss: 1.0532 - dense_4_loss: 0.9034 - dense_5_loss: 2.6789 - dense_6_loss: 2.4695 - dense_3_accuracy: 0.3616 - dense_4_accuracy: 0.4189 - dense_5_accuracy: 0.0992 - dense_6_accuracy: 0.1202
```

Ilustración 49. Resultados del entrenamiento y evaluación del modelo.

| |
|--|
| Accuracy columna scored: 0.3616412213740458 |
| F1 Score columna scored: 0.3610315186246418 |
| Accuracy columna saved: 0.41889312977099236 |
| F1 Score columna saved: 0.261818181818184 |
| Accuracy columna kick_direction: 0.09923664122137404 |
| F1 Score columna kick_direction: 0.10371299461618304 |
| Accuracy columna keeper_direction: 0.12022900763358779 |
| F1 Score columna keeper_direction: 0.1566766924478552 |

Ilustración 50. Resultados de la precisión y Valor-F de las capas de salida.

En la *Ilustración 49*, se puede observar como la precisión y pérdida final de cada capa de salida alcanza los valores mostrados en las gráficas anteriores, y en la *Ilustración 50* se puede apreciar el Valor-F de cada salida.

Visto este primer modelo, hay un claro problema: el *overfitting*. Para frenar el sobreajuste hay varios métodos que se pueden emplear, entre ellos el simplificar el modelo tanto en número de capas ocultas como en número de neuronas por cada capa oculta, también está el *dropout* o abandono, que es una técnica que elimina ciertas neuronas aleatorias durante el entrenamiento de la red neuronal.

Se empezará probando el primer método de simplificación del modelo dado que visualmente va a ser más sencillo de ver si de verdad se reduce el *overfitting*. Dado que antes se tenían tres capas ocultas de 16, 64 y 256 neuronas, se ha decidido eliminar la última capa que es la que más neuronas tenía y se comparará con los resultados vistos en este mismo apartado anteriormente.

4.1.5 Modelo con dos capas ocultas

Modelo con dos capas ocultas de 16 y 64 neuronas respectivamente, cuatro capas de salida con función de coste *binary_crossentropy* para las dos primeras y *categorical_crossentropy* para las dos últimas, 1000 epochs, optimizador *Adam* con tasa de aprendizaje de 0.0002.

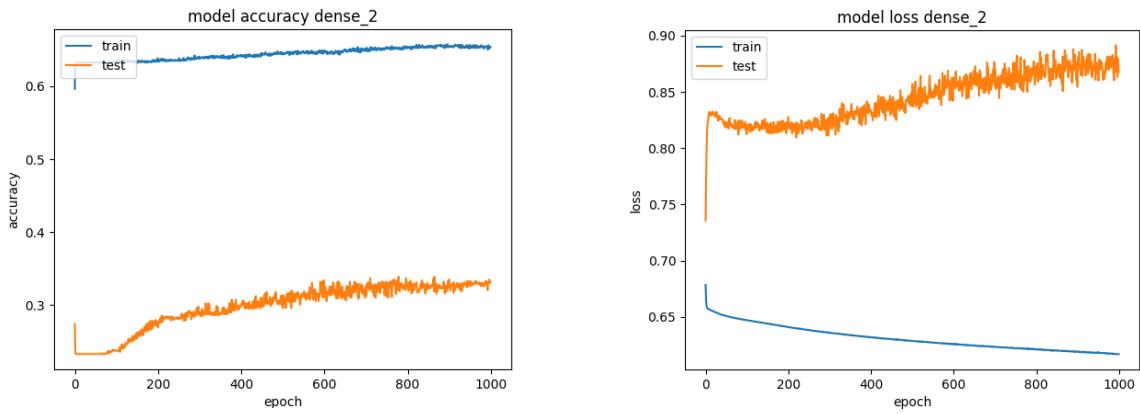


Ilustración 51. Función accuracy y loss de la primera capa de salida (scored)

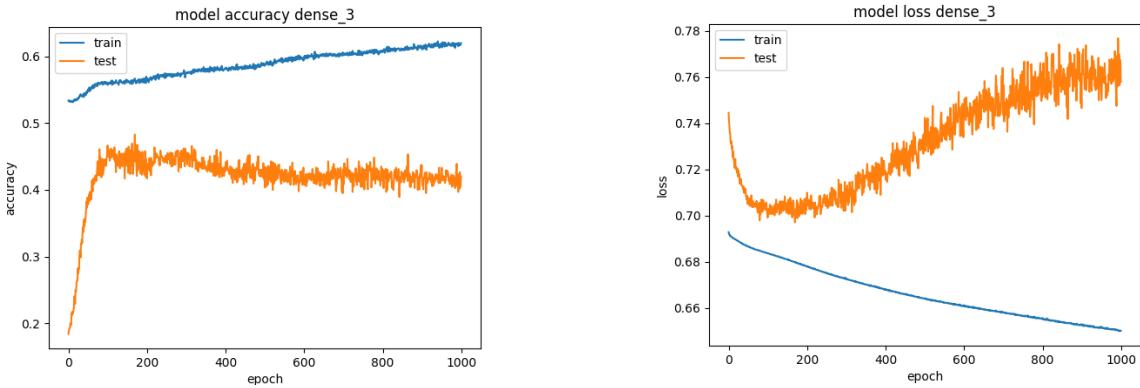


Ilustración 52. Función accuracy y loss de la segunda capa de salida (saved)

Se puede observar en esta nueva configuración de la red neuronal, con dos capas ocultas en vez de tres, que en algunas salidas ha mejorado considerablemente el sobreajuste, aunque también se ve que esta mejora no es suficiente de cara a enfrentar el *overfitting*, como es el caso de la *Ilustración 52* donde se está reduciendo la *loss* hasta en torno a 150 epochs. Sin embargo, es sorprendente que siendo la columna *saved* del mismo estilo en cuanto a valores que la variable *scored*, se ve en la *Ilustración 51* que no hay tanta mejoría.

En la *Ilustración 53*, en concreto en la métrica de precisión es donde más se puede apreciar una mejora del *overfitting*, ya que se ve como aumenta la precisión en los primeros *epochs* de manera similar el conjunto de validación al de entrenamiento, pero luego el de validación deja de mejorar.

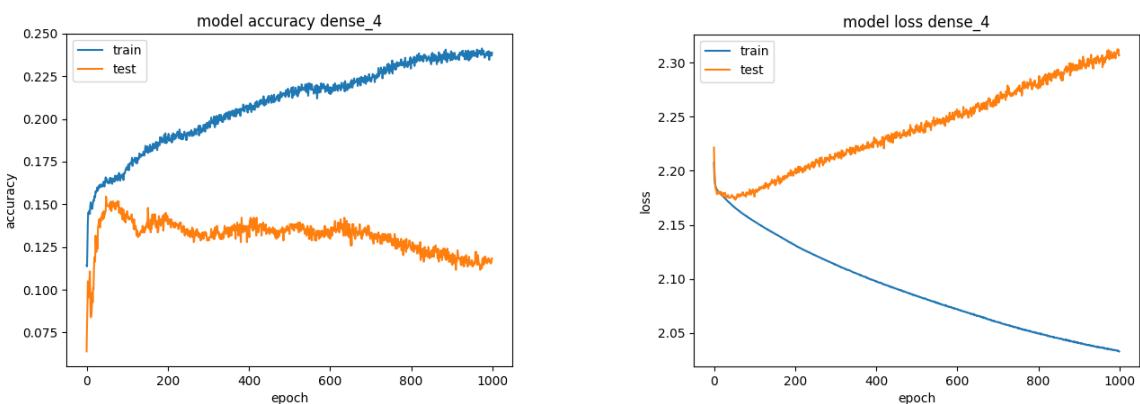


Ilustración 53. Función accuracy y loss de la tercera capa de salida (kick_direction)

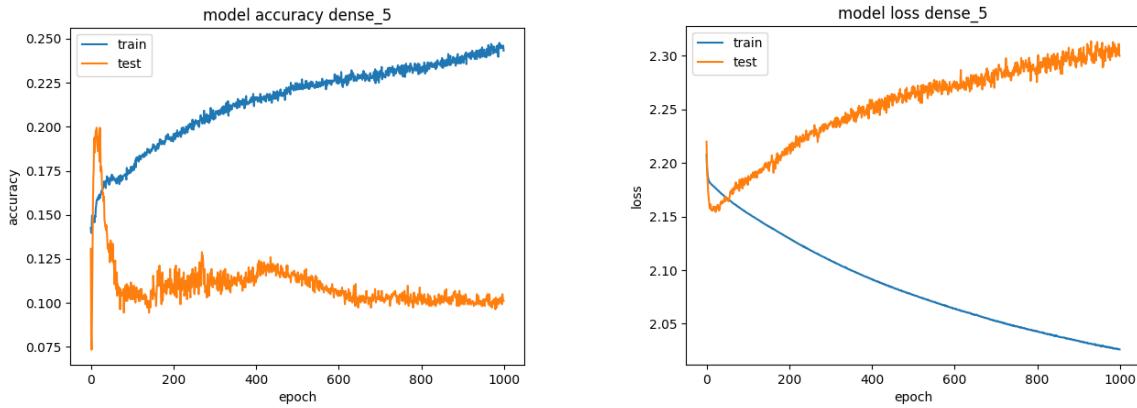


Ilustración 54. Función accuracy y loss de la cuarta capa de salida (keeper_direction)

Finalmente, en la *Ilustración 54* se puede apreciar cierta mejora en el overfitting de la función de pérdida, pero mínima.

En general las funciones de *loss* o pérdida es donde más se puede apreciar ese mejoramiento. Por ejemplo, en las tres últimas ilustraciones ya no se da el caso que en los primeros *epochs* se encuentre el valor más bajo de *loss* en la gráfica, algo que antes sí sucedía con tres capas ocultas. En eso, ha mejorado la red neuronal algo el *overfitting*. Además, se ve también que en la función de *accuracy* se asemeja más los valores correspondientes el conjunto de validación al de entrenamiento que en el anterior modelo. Aun así, se sigue apreciando *overfitting* ya que la precisión sube hasta cierto valor y de ahí se mantiene constante o incluso desciende.

Además, se puede ver que tanto los valores de precisión como del Valor-F son similares al modelo con dos capas ocultas, por lo que este modelo es más eficiente, aunque eso implica que desciende la precisión del conjunto de entrenamiento como se puede ver en la *Ilustración 55* y en la *Ilustración 56*.

```
Epoch 995/1000
99/99 [=====] - 0s 978us/step - loss: 5.3275 - dense_2_loss: 0.6169 - dense_3_loss: 0.6502 - dense_4_loss: 2.8338 - dense_5_loss: 2.8267 - dense_2_accuracy: 0.6563 - dense_3_accuracy: 0.6174 - dense_4_accuracy: 0.2387 - dense_5_accuracy: 0.2457 - val_loss: 6.2498 - val_dense_2_loss: 0.8754 - val_dense_3_loss: 0.7612 - val_dense_4_loss: 2.3089 - val_dense_5_loss: 2.3035 - val_dense_2_accuracy: 0.3292 - val_dense_3_accuracy: 0.4168 - val_dense_4_accuracy: 0.1164 - val_dense_5_accuracy: 0.1011
Epoch 996/1000
99/99 [=====] - 0s 983us/step - loss: 5.3273 - dense_2_loss: 0.6170 - dense_3_loss: 0.6501 - dense_4_loss: 2.8336 - dense_5_loss: 2.0266 - dense_2_accuracy: 0.6521 - dense_3_accuracy: 0.6174 - dense_4_accuracy: 0.2371 - dense_5_accuracy: 0.2460 - val_loss: 6.2708 - val_dense_2_loss: 0.8812 - val_dense_3_loss: 0.7698 - val_dense_4_loss: 2.3123 - val_dense_5_loss: 2.3075 - val_dense_2_accuracy: 0.3302 - val_dense_3_accuracy: 0.4017 - val_dense_4_accuracy: 0.1155 - val_dense_5_accuracy: 0.0992
Epoch 997/1000
99/99 [=====] - 0s 968us/step - loss: 5.3273 - dense_2_loss: 0.6170 - dense_3_loss: 0.6499 - dense_4_loss: 2.8339 - dense_5_loss: 2.0265 - dense_2_accuracy: 0.6512 - dense_3_accuracy: 0.6178 - dense_4_accuracy: 0.2398 - dense_5_accuracy: 0.2447 - val_loss: 6.2306 - val_dense_2_loss: 0.8631 - val_dense_3_loss: 0.7653 - val_dense_4_loss: 2.3102 - val_dense_5_loss: 2.3087 - val_dense_2_accuracy: 0.3349 - val_dense_3_accuracy: 0.4265 - val_dense_4_accuracy: 0.1155 - val_dense_5_accuracy: 0.1050
Epoch 998/1000
99/99 [=====] - 0s 938us/step - loss: 5.3271 - dense_2_loss: 0.6171 - dense_3_loss: 0.6502 - dense_4_loss: 2.8335 - dense_5_loss: 2.8263 - dense_2_accuracy: 0.6544 - dense_3_accuracy: 0.6165 - dense_4_accuracy: 0.2384 - dense_5_accuracy: 0.2454 - val_loss: 6.2698 - val_dense_2_loss: 0.8790 - val_dense_3_loss: 0.7675 - val_dense_4_loss: 2.3122 - val_dense_5_loss: 2.3111 - val_dense_2_accuracy: 0.3302 - val_dense_3_accuracy: 0.4084 - val_dense_4_accuracy: 0.1155 - val_dense_5_accuracy: 0.1031
Epoch 999/1000
99/99 [=====] - 0s 973us/step - loss: 5.3269 - dense_2_loss: 0.6170 - dense_3_loss: 0.6502 - dense_4_loss: 2.8335 - dense_5_loss: 2.0262 - dense_2_accuracy: 0.6537 - dense_3_accuracy: 0.6181 - dense_4_accuracy: 0.2371 - dense_5_accuracy: 0.2460 - val_loss: 6.2520 - val_dense_2_loss: 0.8770 - val_dense_3_loss: 0.7669 - val_dense_4_loss: 2.3068 - val_dense_5_loss: 2.3012 - val_dense_2_accuracy: 0.3321 - val_dense_3_accuracy: 0.4046 - val_dense_4_accuracy: 0.1183 - val_dense_5_accuracy: 0.1011
Epoch 1000/1000
99/99 [=====] - 0s 945us/step - loss: 5.3263 - dense_2_loss: 0.6171 - dense_3_loss: 0.6501 - dense_4_loss: 2.8338 - dense_5_loss: 2.0261 - dense_2_accuracy: 0.6548 - dense_3_accuracy: 0.6200 - dense_4_accuracy: 0.2387 - dense_5_accuracy: 0.2432 - val_loss: 6.2323 - val_dense_2_loss: 0.8677 - val_dense_3_loss: 0.7579 - val_dense_4_loss: 2.3067 - val_dense_5_loss: 2.3008 - val_dense_2_accuracy: 0.3311 - val_dense_3_accuracy: 0.4198 - val_dense_4_accuracy: 0.1183 - val_dense_5_accuracy: 0.1011

Evaluation:
33/33 [=====] - 0s 714us/step - loss: 6.2323 - dense_2_loss: 0.8677 - dense_3_loss: 0.7579 - dense_4_loss: 2.3067 - dense_5_loss: 2.3008 - dense_2_accuracy: 0.3311 - dense_3_accuracy: 0.4198 - dense_4_accuracy: 0.1183 - dense_5_accuracy: 0.1011
```

Ilustración 55. Resultados del entrenamiento y evaluación del modelo.

```
Accuracy columnna scored: 0.33110687022900764
F1 Score columnna scored: 0.2897669706180344
Accuracy columnna saved: 0.4198473282442748
F1 Score columnna saved: 0.2674698795180723
Accuracy columnna kick_direction: 0.1183206106870229
F1 Score columnna kick_direction: 0.12484612673016186
Accuracy columnna keeper_direction: 0.10114503816793893
F1 Score columnna keeper_direction: 0.13703265900848619
```

Ilustración 56. Resultados de la precisión y Valor-F de las capas de salida.

Dado que reduciendo el número de capas ocultas en una unidad ha funcionado, se procede a comprobar si con una única capa oculta en la red neuronal aún se reduce más el *overfitting*.

4.1.6 Modelo con una capa oculta

Modelo con una capa oculta de 16 neuronas, cuatro capas de salida con función de coste *binary_crossentropy* para las dos primeras y *categorical_crossentropy* para las dos últimas, 1000 epochs, optimizador Adam con tasa de aprendizaje de 0.0001.

Viendo que reducir el número de capas de tres a dos ha funcionado satisfactoriamente, podría ser lógico pensar que con una única capa funcionaría mejor.

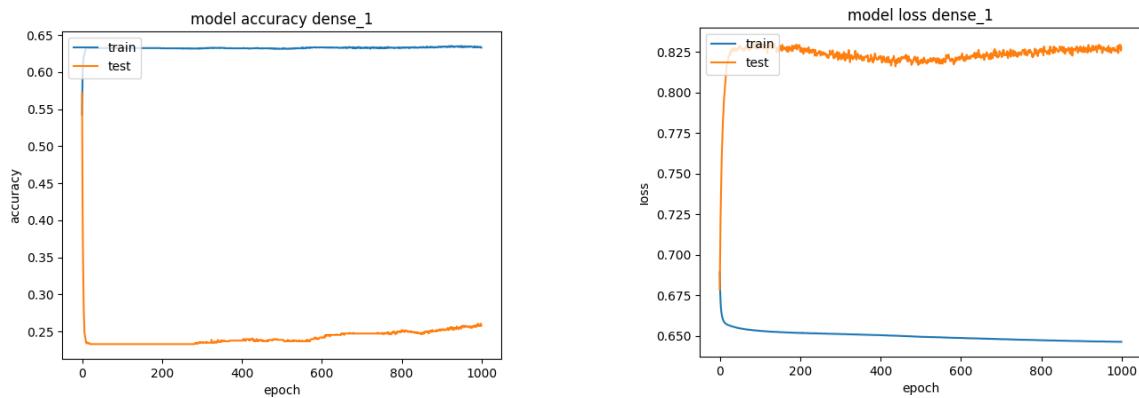


Ilustración 57. Función accuracy y loss de la primera capa de salida (scored)

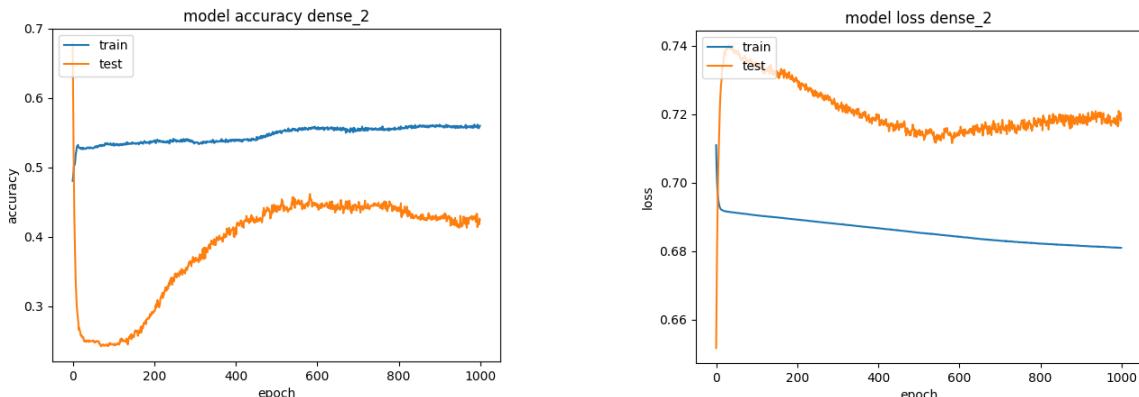


Ilustración 58. Función accuracy y loss de la segunda capa de salida (saved)

En este modelo con una única capa oculta de 16 neuronas, se puede ver en las dos primeras capas de salida (Ilustración 57 e Ilustración 58) que la precisión es algo inferior que con dos capas ocultas y ya no se asemejan las curvas del conjunto de datos de entrenamiento con el de validación, mientras que la pérdida es similar, aunque asciende linealmente y no aumenta en demasiado su valor.

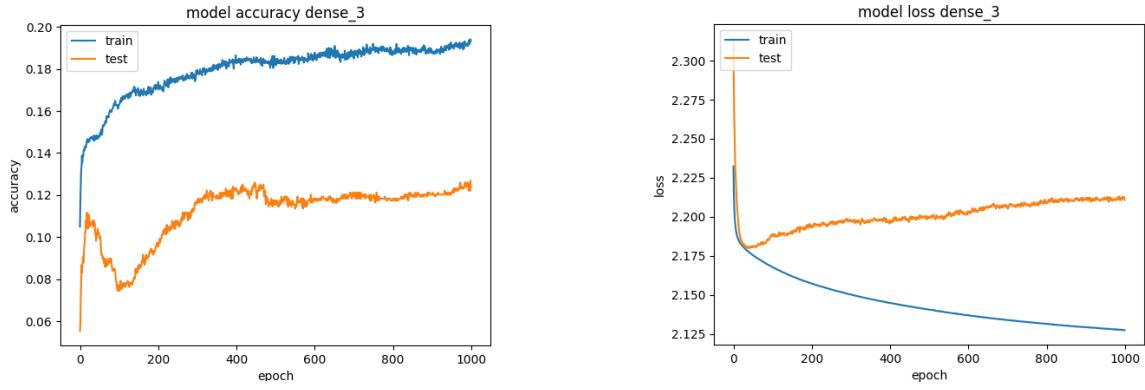


Ilustración 59. Función accuracy y loss de la tercera capa de salida (kick_direction)

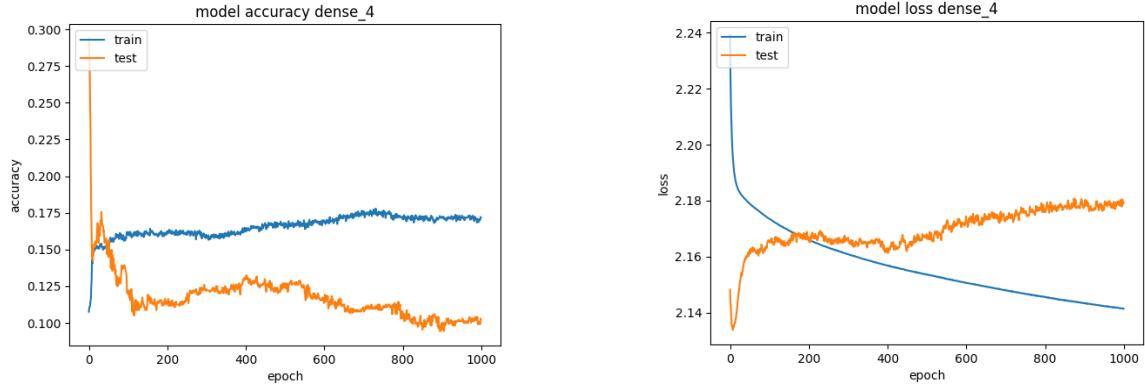


Ilustración 60. Función accuracy y loss de la cuarta capa de salida (keeper_direction)

En la pérdida de las dos últimas capas de salida (*Ilustración 59* e *Ilustración 60*) pasa algo parecido que en las dos primeras salidas en cuanto a la función de pérdida, es decir, a partir de cierto número de *epochs* se mantiene más o menos estable la pérdida. Por otro lado, la función de precisión es similar o incluso un poco inferior al modelo con dos capas.

En la *Ilustración 61* e *Ilustración 62* se aprecia lo comentado respecto a que la precisión en este modelo es incluso inferior, aunque también se reduce el valor de pérdida. Por otro lado, se ve como el Valor-F de la primera columna se ha reducido considerablemente a pesar de que en el resto de las columnas más o menos se mantiene estable.

```

Epoch 995/1000
99/99 [=====] - 0s 941us/step - loss: 5.5963 - dense_1_loss: 0.6464 - dense_2_loss: 0.6809 - dense_3_loss: 2.1275 - dense_4_loss: 2.1415 - dense_1_accuracy: 0.6337 - dense_2_accuracy: 0.5685 - dense_3_accuracy: 0.1932 - dense_4_accuracy: 0.1696 - val_loss: 5.9387 - val_dense_1_loss: 0.8281 - val_dense_2_loss: 0.7190 - val_dense_3_loss: 2.2126 - val_dense_4_loss: 2.1796 - val_dense_1_accuracy: 0.2576 - val_dense_2_accuracy: 0.4206 - val_dense_3_accuracy: 0.1221 - val_dense_4_accuracy: 0.0992
Epoch 996/1000
99/99 [=====] - 0s 945us/step - loss: 5.5964 - dense_1_loss: 0.6464 - dense_2_loss: 0.6810 - dense_3_loss: 2.1275 - dense_4_loss: 2.1415 - dense_1_accuracy: 0.6340 - dense_2_accuracy: 0.5576 - dense_3_accuracy: 0.1938 - dense_4_accuracy: 0.1706 - val_loss: 5.9394 - val_dense_1_loss: 0.8294 - val_dense_2_loss: 0.7199 - val_dense_3_loss: 2.2122 - val_dense_4_loss: 2.1799 - val_dense_1_accuracy: 0.2586 - val_dense_2_accuracy: 0.4198 - val_dense_3_accuracy: 0.1221 - val_dense_4_accuracy: 0.0992
Epoch 997/1000
99/99 [=====] - 0s 980us/step - loss: 5.5962 - dense_1_loss: 0.6464 - dense_2_loss: 0.6810 - dense_3_loss: 2.1275 - dense_4_loss: 2.1414 - dense_1_accuracy: 0.6337 - dense_2_accuracy: 0.5563 - dense_3_accuracy: 0.1919 - dense_4_accuracy: 0.1703 - val_loss: 5.9389 - val_dense_1_loss: 0.8275 - val_dense_2_loss: 0.7188 - val_dense_3_loss: 2.2122 - val_dense_4_loss: 2.1804 - val_dense_1_accuracy: 0.2576 - val_dense_2_accuracy: 0.4237 - val_dense_3_accuracy: 0.1258 - val_dense_4_accuracy: 0.3002
Epoch 998/1000
99/99 [=====] - 0s 958us/step - loss: 5.5964 - dense_1_loss: 0.6464 - dense_2_loss: 0.6809 - dense_3_loss: 2.1275 - dense_4_loss: 2.1415 - dense_1_accuracy: 0.6334 - dense_2_accuracy: 0.5573 - dense_3_accuracy: 0.1926 - dense_4_accuracy: 0.1703 - val_loss: 5.9412 - val_dense_1_loss: 0.8293 - val_dense_2_loss: 0.7204 - val_dense_3_loss: 2.2127 - val_dense_4_loss: 2.1788 - val_dense_1_accuracy: 0.2586 - val_dense_2_accuracy: 0.4179 - val_dense_3_accuracy: 0.1221 - val_dense_4_accuracy: 0.0992
99/99 [=====] - 0s 941us/step - loss: 5.5962 - dense_1_loss: 0.6464 - dense_2_loss: 0.6809 - dense_3_loss: 2.1274 - dense_4_loss: 2.1415 - dense_1_accuracy: 0.6340 - dense_2_accuracy: 0.5589 - dense_3_accuracy: 0.1941 - dense_4_accuracy: 0.1722 - val_loss: 5.9347 - val_dense_1_loss: 0.8259 - val_dense_2_loss: 0.7181 - val_dense_3_loss: 2.2112 - val_dense_4_loss: 2.1796 - val_dense_1_accuracy: 0.2605 - val_dense_2_accuracy: 0.4265 - val_dense_3_accuracy: 0.1269 - val_dense_4_accuracy: 0.1031
Epoch 1000/1000
99/99 [=====] - 0s 942us/step - loss: 5.5961 - dense_1_loss: 0.6464 - dense_2_loss: 0.6809 - dense_3_loss: 2.1274 - dense_4_loss: 2.1413 - dense_1_accuracy: 0.6338 - dense_2_accuracy: 0.5602 - dense_3_accuracy: 0.1935 - dense_4_accuracy: 0.1719 - val_loss: 5.9361 - val_dense_1_loss: 0.8274 - val_dense_2_loss: 0.7187 - val_dense_3_loss: 2.2108 - val_dense_4_loss: 2.1791 - val_dense_1_accuracy: 0.2576 - val_dense_2_accuracy: 0.4246 - val_dense_3_accuracy: 0.1248 - val_dense_4_accuracy: 0.1021
Evaluate:
33/33 [=====] - 0s 346us/step - loss: 5.9361 - dense_1_loss: 0.6274 - dense_2_loss: 0.7187 - dense_3_loss: 2.2108 - dense_4_loss: 2.1791 - dense_1_accuracy: 0.2576 - dense_2_accuracy: 0.4246 - dense_3_accuracy: 0.1240 - dense_4_accuracy: 0.1021

```

Ilustración 61. Resultados del entrenamiento y evaluación del modelo.

```

Accuracy column scored: 0.25763358778625955
F1 Score column scored: 0.07819905213270142
Accuracy column saved: 0.42461832061068705
F1 Score column saved: 0.28299643281807374
Accuracy column kick_direction: 0.12404580152671756
F1 Score column kick_direction: 0.10335139356762944
Accuracy column keeper_direction: 0.10209923664122138
F1 Score column keeper_direction: 0.12074367171351728

```

Ilustración 62. Resultados de la precisión y Valor-F de las capas de salida.

Puesto que reducir el número de capas ocultas de dos a una única capa ya no funciona tanto de cara a reducir el *overfitting* como sí funcionó reducir de tres a dos capas ocultas, se ha decidido dejar dos capas ocultas de 16 y 64 neuronas respectivamente.

Otra de las técnicas más usadas para reducir el sobreajuste es el *dropout*, mencionado anteriormente. Esta técnica, que consiste en eliminar aleatoriamente un porcentaje indicándolo en el modelo, se empleará después de la última capa oculta.

4.1.7 Modelo con dos capas ocultas y dropout

Modelo con dos capas ocultas de 16 y 64 neuronas respectivamente, cuatro capas de salida con función de coste *binary_crossentropy* para las dos primeras y *categorical_crossentropy* para las dos últimas, 1000 epochs, optimizador *Adam* con tasa de aprendizaje de 0.0001, *dropout* de 0.2.

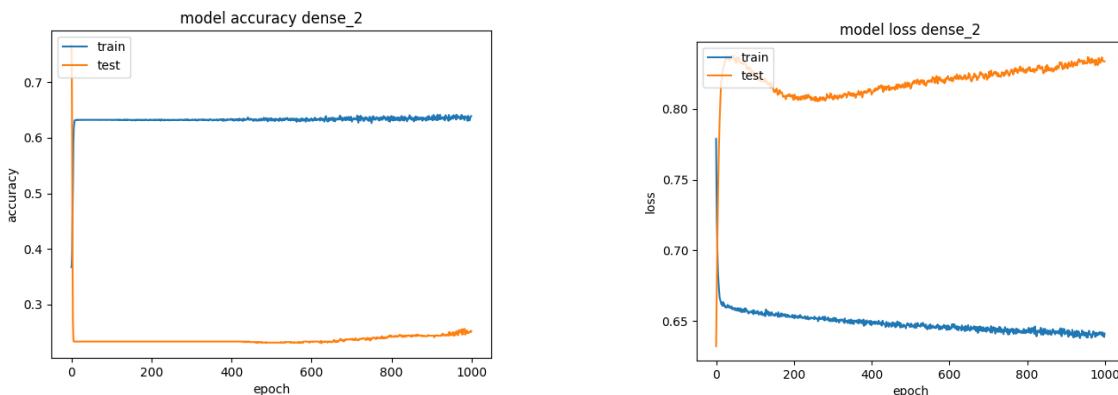


Ilustración 63. Función accuracy y loss de la primera capa de salida (scored)

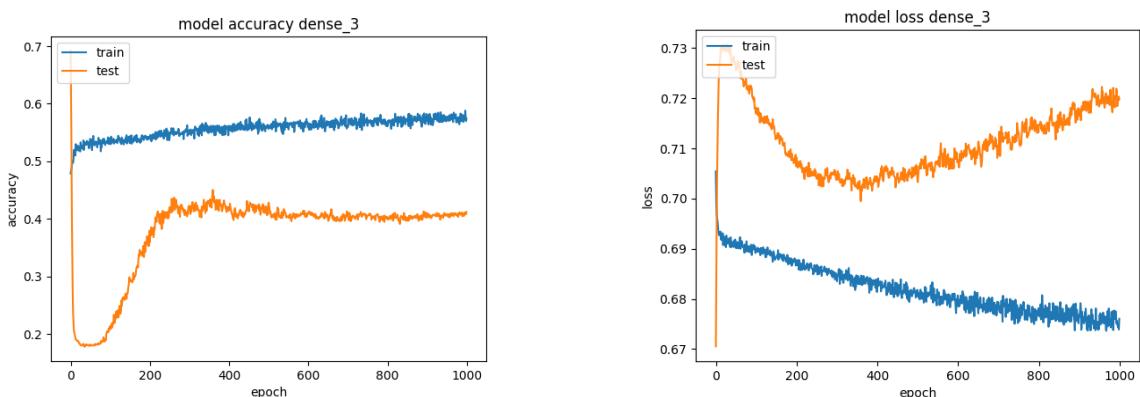


Ilustración 64. Función accuracy y loss de la segunda capa de salida (saved)

En la *Ilustración 63* se puede ver como la precisión del modelo en esa salida de la red se ha reducido considerablemente respecto a los modelos comentados anteriormente, al igual que la pérdida ha aumentado aumentando la diferencia tanto de precisión como de pérdida entre el conjunto de validación y de entrenamiento. En la *Ilustración 64* se aprecia como tanto en la métrica de precisión como de pérdida tiene un comportamiento extraño en el inicio del entrenamiento, pero después la precisión aumenta hasta la *epoch* 200 más o menos y de ahí se mantiene estable, mientras que la función de pérdida se reduce hasta el *epoch* 300 y ya de ahí asciende de nuevo, certificando así que el modelo aun así sigue sufriendo *overfitting*.

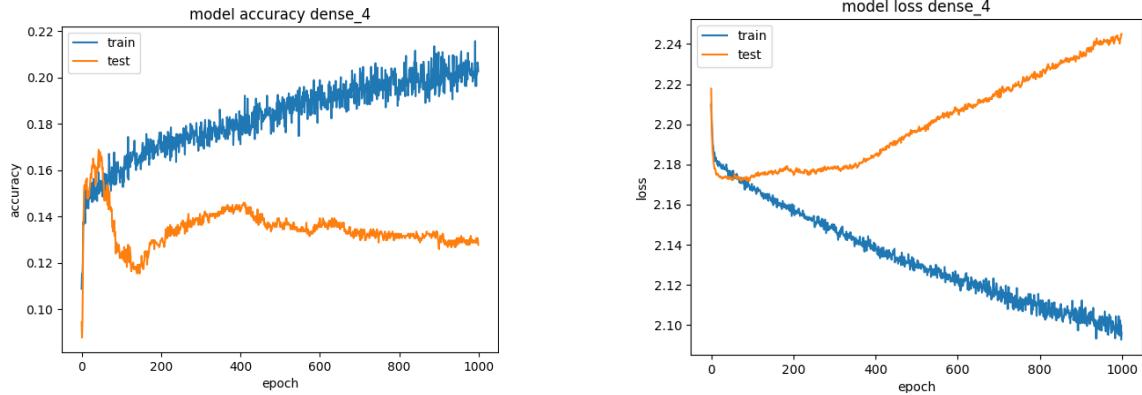


Ilustración 65. Función accuracy y loss de la tercera capa de salida (kick_direction)

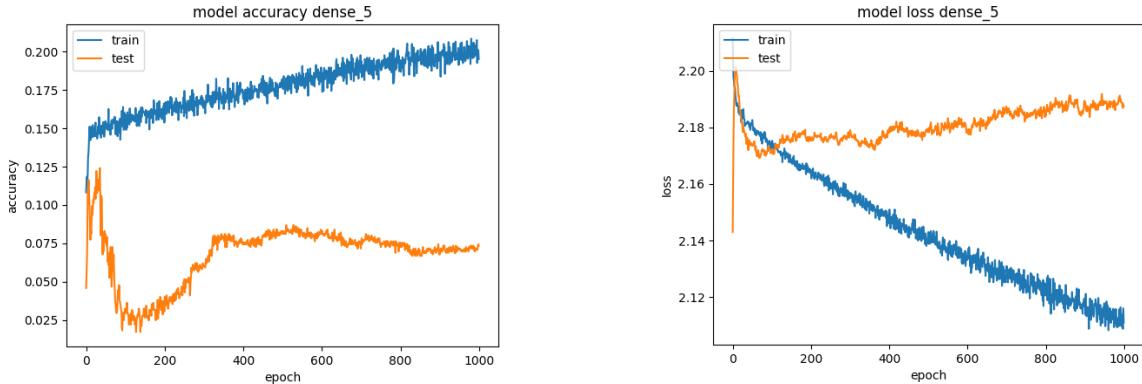


Ilustración 66. Función accuracy y loss de la cuarta capa de salida (keeper_direction)

En la *Ilustración 65* e *Ilustración 66* se observa como en la función de precisión en los primeros *epochs* el conjunto de validación parece que va a seguir la curva del conjunto de entrenamiento, pero en seguida se reduce la precisión. Por otro lado, en la función de pérdida de la *Ilustración 65* ocurre algo parecido, ya que se va reduciendo la pérdida al principio, pero enseguida aumenta en el conjunto de validación.

Tras aplicar una capa de *dropout* al modelo de un 20%, se puede observar que respecto al modelo de dos capas ocultas (*Ilustración 51*, *Ilustración 52*, *Ilustración 53* e *Ilustración 54*), las gráficas tanto de precisión como de pérdida son bastante similares, sin reducirse el *overfitting* en considerablemente. Además, la precisión de tres de las cuatro capas es inferior a la obtenido anteriormente como se puede ver en la *Ilustración 67* e *Ilustración 68*. Aunque la pérdida es algo inferior, no es algo tan considerablemente importante como para que merezca la pena añadir esta capa oculta, debido a que la capa añadida de *dropout* amplia considerablemente el tiempo de cómputo, al tener que eliminar un 20% de las neuronas tras cada *epoch* del entrenamiento.

```
Epoch 995/1000
99/99 [=====] - 0s 1ms/step - loss: 5.5301 - dense_2_loss: 0.6421 - dense_3_loss: 0.6751 - dense_4_loss: 2.0991 - dense_5_loss: 2.1138 - dense_2_accuracy: 0.6321 - dense_3_accuracy: 0.5729 - dense_4_accuracy: 0.1964 - dense_5_accuracy: 0.2075 - val_loss: 5.9810 - va
l_dense_2_loss: 0.8337 - val_dense_3_loss: 0.7184 - val_dense_4_loss: 2.2402 - val_dense_5_loss: 2.1887 - val_dense_2_accuracy: 0.2500 - val_dense_3_accuracy: 0.4103 - val_dense_4_accuracy: 0.1307 - val_dense_5_accuracy: 0.0716
Epoch 996/1000
99/99 [=====] - 0s 1ms/step - loss: 5.5290 - dense_2_loss: 0.6395 - dense_3_loss: 0.6747 - dense_4_loss: 2.1023 - dense_5_loss: 2.1126 - dense_2_accuracy: 0.6314 - dense_3_accuracy: 0.5729 - dense_4_accuracy: 0.2024 - dense_5_accuracy: 0.1926 - val
l_dense_2_loss: 0.8338 - val_dense_3_loss: 0.7188 - val_dense_4_loss: 2.2418 - val_dense_5_loss: 2.1886 - val_dense_2_accuracy: 0.2500 - val_dense_3_accuracy: 0.4113 - val_dense_4_accuracy: 0.1288 - val_dense_5_accuracy: 0.0706
Epoch 997/1000
99/99 [=====] - 0s 1ms/step - loss: 5.5185 - dense_2_loss: 0.6398 - dense_3_loss: 0.6743 - dense_4_loss: 2.0947 - dense_5_loss: 2.1096 - dense_2_accuracy: 0.6349 - dense_3_accuracy: 0.5882 - dense_4_accuracy: 0.2015 - dense_5_accuracy: 0.1988 - val
l_dense_2_loss: 0.8336 - val_dense_3_loss: 0.7200 - val_dense_4_loss: 2.2427 - val_dense_5_loss: 2.1869 - val_dense_2_accuracy: 0.2510 - val_dense_3_accuracy: 0.4084 - val_dense_4_accuracy: 0.1288 - val_dense_5_accuracy: 0.0716
Epoch 998/1000
99/99 [=====] - 0s 1ms/step - loss: 5.5235 - dense_2_loss: 0.6398 - dense_3_loss: 0.6747 - dense_4_loss: 2.1081 - dense_5_loss: 2.1089 - dense_2_accuracy: 0.6372 - dense_3_accuracy: 0.5773 - dense_4_accuracy: 0.2066 - dense_5_accuracy: 0.2011 - val
l_dense_2_loss: 0.8335 - val_dense_3_loss: 0.7196 - val_dense_4_loss: 2.2431 - val_dense_5_loss: 2.1883 - val_dense_2_accuracy: 0.2500 - val_dense_3_accuracy: 0.4094 - val_dense_4_accuracy: 0.1307 - val_dense_5_accuracy: 0.0735
Epoch 999/1000
99/99 [=====] - 0s 1ms/step - loss: 5.5214 - dense_2_loss: 0.6380 - dense_3_loss: 0.6739 - dense_4_loss: 2.0927 - dense_5_loss: 2.1161 - dense_2_accuracy: 0.6384 - dense_3_accuracy: 0.5700 - dense_4_accuracy: 0.2053 - dense_5_accuracy: 0.2005 - val
l_dense_2_loss: 0.8336 - val_dense_3_loss: 0.7203 - val_dense_4_loss: 2.2458 - val_dense_5_loss: 2.1871 - val_dense_2_accuracy: 0.2500 - val_dense_3_accuracy: 0.4084 - val_dense_4_accuracy: 0.1298 - val_dense_5_accuracy: 0.0744
Epoch 1000/1000
99/99 [=====] - 0s 1ms/step - loss: 5.5246 - dense_2_loss: 0.6413 - dense_3_loss: 0.6768 - dense_4_loss: 2.0962 - dense_5_loss: 2.1118 - dense_2_accuracy: 0.6391 - dense_3_accuracy: 0.5742 - dense_4_accuracy: 0.2027 - dense_5_accuracy: 0.1951 - val
l_dense_2_loss: 0.8336 - val_dense_3_loss: 0.7198 - val_dense_4_loss: 2.2469 - val_dense_5_loss: 2.1876 - val_dense_2_accuracy: 0.2519 - val_dense_3_accuracy: 0.4122 - val_dense_4_accuracy: 0.1279 - val_dense_5_accuracy: 0.0735
Evaluate:
33/33 [=====] - 0s 562us/step - loss: 5.9859 - dense_2_loss: 0.8336 - dense_3_loss: 0.7198 - dense_4_loss: 2.2409 - dense_5_loss: 2.1876 - dense_2_accuracy: 0.2519 - dense_3_accuracy: 0.4122 - dense_4_accuracy: 0.1279 - dense_5_accuracy: 0.0735
```

Ilustración 67. Resultados del entrenamiento y evaluación del modelo.

```
Accuracy columna scored: 0.25190839694656486
F1 Score columna scored: 0.07109004739336493
Accuracy columna saved: 0.4122137404580153
F1 Score columna saved: 0.2837209302325581
Accuracy columna kick_direction: 0.12786259541984732
F1 Score columna kick_direction: 0.10664257432071503
Accuracy columna keeper_direction: 0.07347328244274809
F1 Score columna keeper_direction: 0.10160815203711734
```

Ilustración 68. Resultados de la precisión y Valor-F de las capas de salida.

4.2 Modelo definitivo

Tal y como se ha comentado en el apartado anterior, el modelo definitivo será el mencionado en el mismo, es decir, un modelo con los siguientes hiperparámetros:

- **Número de capas ocultas:** 2
- **Número de neuronas en cada capa oculta:** 16 en la primera y 64 en la segunda.
- **Cuatro capas de salida.**
- **Función de activación:** *sigmoid* para las dos primeras, *softmax* para las dos últimas.
- **Función de coste:** *binary_crossentropy* para las dos primeras y *categorical_crossentropy* para las dos últimas.
- **Número de epochs:** 1000.
- **Optimizador:** *Adam* con tasa de aprendizaje 0.0002.

En la *Ilustración 69* se puede ver el resumen del modelo con los hiperparámetros descritos arriba, mediante la ejecución de la función *summary* del modelo.

| Model: "sequential" | | | |
|---|--------------|---------|------------------|
| Layer (type) | Output Shape | Param # | |
| dense (Dense) | (None, 16) | 192 | |
| dense_1 (Dense) | (None, 64) | 1088 | |
| Total params: 1,280 Trainable params: 1,280 Non-trainable params: 0 | | | |
| Model: "model" | | | |
| Layer (type) | Output Shape | Param # | Connected to |
| input_1 (InputLayer) | [(None, 11)] | 0 | |
| sequential (Sequential) | (None, 64) | 1280 | input_1[0][0] |
| dense_2 (Dense) | (None, 1) | 65 | sequential[0][0] |
| dense_3 (Dense) | (None, 1) | 65 | sequential[0][0] |
| dense_4 (Dense) | (None, 9) | 585 | sequential[0][0] |
| dense_5 (Dense) | (None, 9) | 585 | sequential[0][0] |
| Total params: 2,580 Trainable params: 2,580 Non-trainable params: 0 | | | |

Ilustración 69. Resumen del modelo definitivo

Ahora que ya hay un modelo definitivo, aunque había otras opciones con resultados similares, se terminó decidiendo por esta opción por el equilibrio entre minimizar el *overfitting*, obtener el mejor resultado de *accuracy*, *loss* y Valor-F mostrado en la *Ilustración 51*, *Ilustración 52*, *Ilustración 53* e *Ilustración 54*.

Para ver los resultados obtenidos con el entrenamiento de esta red, se han utilizado las matrices de confusión para ver cómo de bien predice o no nuestra red, aunque con los valores obtenidos de precisión mostrados en la *Ilustración 55* e *Ilustración 56*, no se deberían esperar muy buenos resultados.

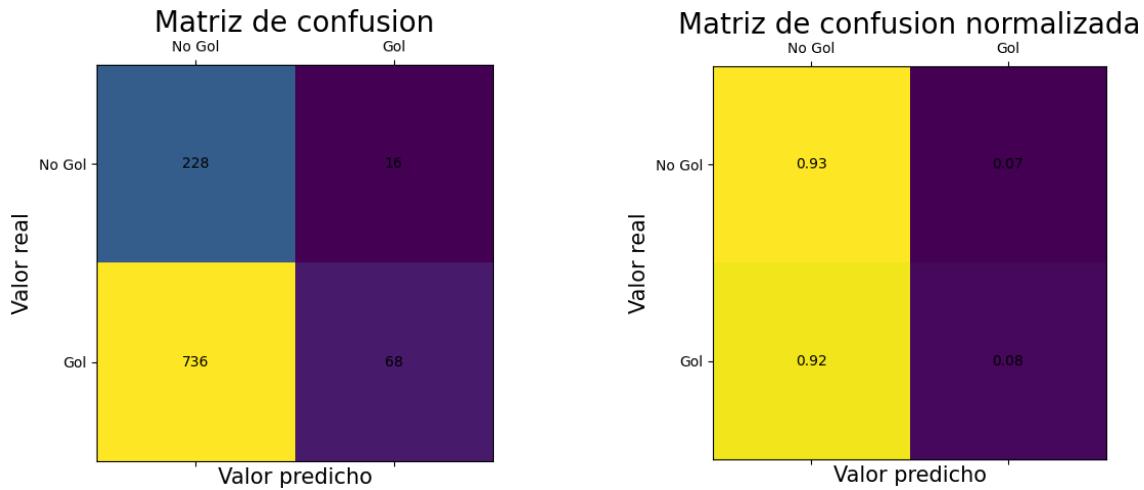


Ilustración 70. Matriz de confusión y matriz de confusión normalizada de la variable *scored*.

En estas dos matrices de confusión de la *Ilustración 70*, correspondientes a los valores predichos de la primera salida, se puede ver que el 93% de las veces que no era gol (*scored* = 0) se ha predicho bien, mientras que sólo el 8% de las veces que era gol (*scored* = 1) el resultado se predijo correctamente.

Gracias a la primera matriz de confusión, se puede determinar que, de las 1048 muestras, hay 804 muestras que son goles, un 76.71% de acierto en el penalti en las muestras recogidas.

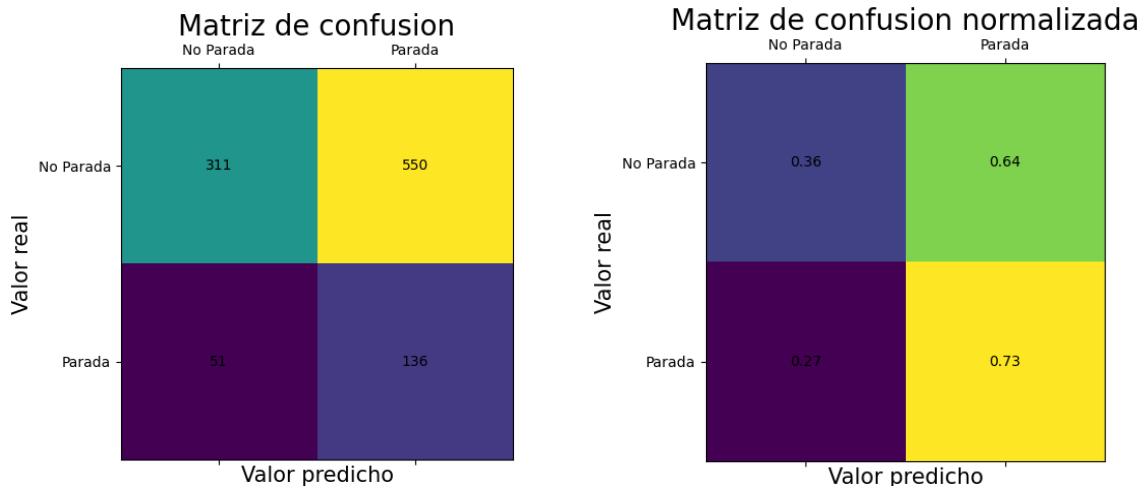


Ilustración 71. Matriz de confusión y matriz de confusión normalizada de la variable *saved*.

Las matrices de confusión correspondientes a la segunda salida del modelo de la *Ilustración 71*, salida de la variable *saved*, muestran un estilo parecido a los de la anterior variable, pero al revés. Es decir, solo el 36% de las veces que el valor real es “No parada” (*saved* = 0) el modelo las ha predicho correctamente; mientras que el 73% de las veces que el portero es capaz de parar el penalti (*saved* = 1), el modelo predice bien esas paradas.

Esta columna tiene sus valores condicionados, debido a que su valor depende de que, en primer lugar, el penalti sea fallado por el lanzador, ya que los penaltis en el fútbol son siempre una ventaja para el lanzador y se puede comprobar en la *Ilustración 70*, donde el 76.71% de los penaltis registrados han sido gol. Por lo que, si el portero ha parado el penalti, que no siempre ocurre cuando no es gol, ocurre dentro de ese 23.29% restante de penaltis que no son goles. Y, como se puede entender, si un penalti es gol, no puede ser parado por el portero.

Vistas las dos primeras columnas de salida, se pasa a analizar las dos últimas capas de salidas correspondientes a la dirección donde se lanza el penalti y donde se lanza el portero.

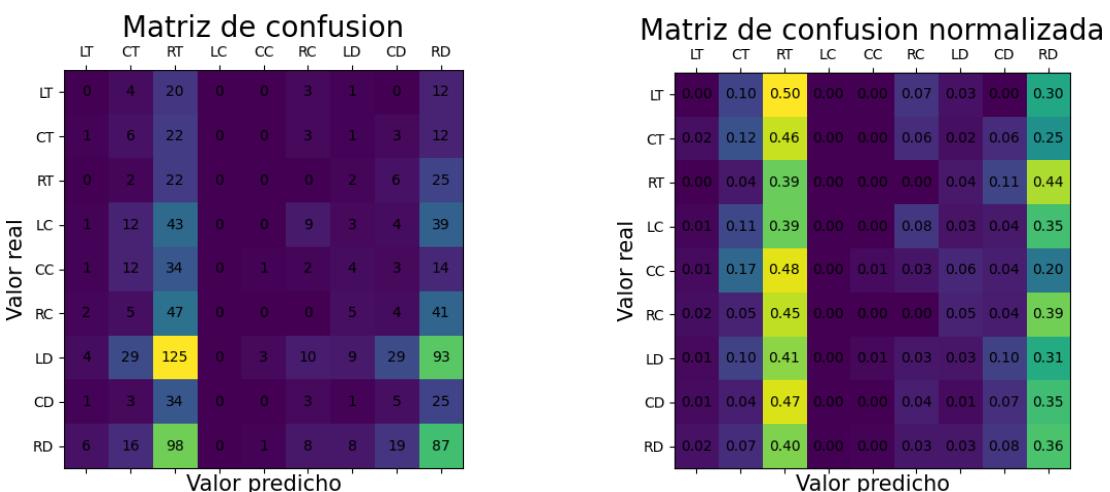


Ilustración 72. Matriz de confusión y matriz de confusión normalizada de la variable *kick_direction*.

En esta salida los valores a interpretar no son tan sencillos como en las dos primeras salidas que eran valores binarios. Aquí, hay nueve posibles valores, por las nueve posiciones definidas de la portería. Los resultados predichos por la red neuronal corresponden a la diagonal principal de la matriz (la diagonal que va desde la parte superior izquierda a la inferior derecha).

Cabe destacar que en la primera matriz de la *Ilustración 72*, hay dos valores, es decir, dos direcciones, que tienen el mayor número de lanzamientos: la zona inferior izquierda (LD) un 28.88% y la zona inferior derecha (RD) un 23.18% de las muestras. No es casualidad, ya que reúnen dos de las mayores seguridad para meter un gol: lanzar a un lado de la portería para que el portero tenga que adivinar el lado, y lanzar el penalti raso al suelo ya que en estos lanzamientos sólo puedes lanzarla fuera si la envías más allá de la extensión de la portería, mientras que si la lanzas alta, es fácil no controlar la fuerza o la dirección y lanzar por encima del larguero. Es por eso por lo que se ven tan pocos penaltis en cualquiera de las tres zonas de la parte superior, solo 145 penaltis de los 1048 penaltis, un 12.83% de los penaltis. También es difícil ver lanzamientos de penalti que vayan al medio en cuanto a dirección, ya que es la zona que en un principio cubre el portero estando de pie y son las zonas donde más cubren con su cuerpo, en concreto solo 191 de los 1048 penaltis, un 18.22% de los penaltis, van al centro de la portería.

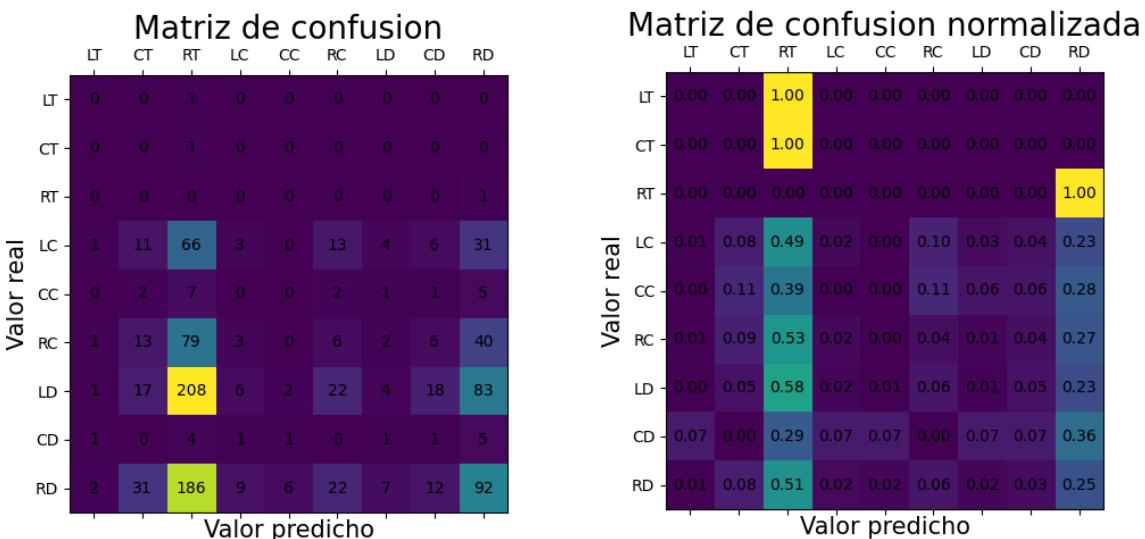


Ilustración 73. Matriz de confusión y matriz de confusión normalizada de la variable `keeper_direction`.

Respecto a la última salida del modelo, la correspondiente a la dirección en la que se lanza el portero, cabe destacar que, como se mencionó en el apartado 3.2.1, aunque el portero se ha lanzado a una única dirección, puede llegar a cubrir varias zonas con su cuerpo e incluso parar un balón con los pies, que correspondería a otra zona de donde inicialmente se había lanzado.

En la matriz de confusión inicial con el número de muestras, se puede observar en la *Ilustración 73* que en las tres zonas superiores solo hay una muestra en cada una de ellas. Aunque en las direcciones del lanzamiento ya eran pocas las muestras en esas tres zonas, en los porteros aún es menor, ¿a qué se debe esto? Pues esto es debido a, ni más ni menos, las limitaciones físicas del portero, ya que una portería mide 7.32 metros de ancho y 2.44 metros de alto, si un balón fuera hacia la zona superior derecha o zona superior izquierda, lo que ocurre es que el portero la suele parar con la mano aun lanzándose en otra dirección, en concreto a

una altura centro. Físicamente el portero tendría que hacer un salto en parado de 3,5 metros de ancho y 1 metro de alto más o menos, algo bastante difícil. De ahí que haya tan pocos valores en esas zonas.

Lo más fácil para un portero es lanzarse a las zonas bajas de los lados, ya que es la zona más preferida por lanzadores y, a su vez, una zona no tan difícil para el portero. De ahí que 361 penaltis se lanzaron a la zona inferior izquierda (LD), un 34.44% de las muestras, y 367 de las 1048 muestras, un 35.01% van hacia la zona inferior derecha (RD).

A simple vista, ya se puede ver que los resultados no son buenos, algo que ya podíamos deducir debido a la precisión del modelo en estas dos últimas salidas, sobre todo en la precisión de las dos últimas salidas que se obtenían valores entorno al 10%-15%.

5. Impacto sociales y medioambientales

Dado que este TFG trata sobre la predicción de penaltis en el fútbol, no se aprecian impactos medioambientales que puedan nombrarse.

Sin embargo, en cuanto a impacto social, quizá no tanto este proyecto, pero sí el ámbito que trata sí tiene un impacto social. Esto es debido a que, en el mundo del fútbol, cada vez está todo más informatizado y analizado, algo que en otros deportes sobre todo de Estados Unidos de América, donde ya analizan todos los posibles parámetros. Esto lleva ocurriendo unos años en Europa y en concreto en el fútbol ya que es el deporte que más recursos económicos tiene en Europa. Gracias a los grandes recursos económicos en comparación con otros deportes, se puede hacer mayor labor de investigación e invertir más en analizar todo, como la distancia corrida por los jugadores, la intensidad con que corren, el descanso de sueño, cómo se encuentran los músculos de los jugadores para poder prevenir lesiones, etcétera. Y el análisis de los penaltis no se queda atrás.

Toda este análisis cada vez le interesa más al público del fútbol, gracias a la accesibilidad que existe hoy en día a múltiple información, cada vez gusta más estar más informado de lo que a uno le gusta y al alcance de la mano, como puede ser un móvil mientras estás viendo el partido en directo. De ahí la incorporación de gráficos durante la transmisión en directo del partido en la temporada 2020/2021 como la de dónde se suelen lanzar los saques de esquina, si en corto, al primer palo o al segundo palo, dividido en zonas y con su porcentaje; y la inclusión de la dirección y los aciertos en los lanzamientos del jugador que lanza el penalti, en la *Ilustración 74* hay dos ejemplos de esta tecnología usada en la temporada 2020/2021.



Ilustración 74. Ejemplo del análisis de datos en el fútbol.

Cabe destacar que este proyecto está hecho a muy pequeña escala, con un proyecto más grande, con mayor número de muestras, mejor análisis de los datos y mayor número de muestras, se podrían alcanzar grandes impactos sociales con una buena estrategia de marketing de cara a intentar llegar con este conocimiento, su importancia y su posible aplicación a cualquier ámbito, a todo el mundo posible.

6. Conclusiones

Este TFG ha supuesto una mezcla de sentimientos encontrados, dado que se han cumplido algunos objetivos del proyecto, pero por otro lado no se han cumplido todos. Como se ha abarcado desde la creación del *dataset*, la ampliación de éste aplicando técnicas de aumento de datos, creación de una red neuronal y ajustarla para su correcto funcionamiento, no solo hay que sacar conclusiones generales del proyecto, sino también conclusiones parciales de las distintas partes que conforman el todo.

En primer lugar, se han aprendido técnicas que se suelen aplicar en modelos de redes neuronales que de no haber realizado este proyecto no habría sido posible. Además, como el conjunto de datos con el que se ha entrenado la red neuronal han sido creados explícitamente para este TFG, sin obtenerlos de ninguna empresa o de internet, ha permitido descubrir lo difícil que es la recopilación de datos desde cero. Desde los datos básicos como quién es el que lanza, con qué pie lanza, en qué equipo juega, etcétera, hasta encontrar documentación videográfica de los penaltis recopilados para determinar la dirección a la que se lanzan. Cabe destacar que ésta no es la mejor manera de determinar hacia dónde se lanza un penalti, es decir, que un ojo humano lo determinó ya que aquí entra el error humano. Y como también se determinó la división de una portería en nueve zonas, había casos en los que el penalti iba lanzado a mitad de camino entre dos zonas, teniendo que decidir así hacia qué lado había ido el balón.

Después, como lo descrito anteriormente supuso una labor costosa, sobre todo en tiempo, y ya se llegó a un punto donde apenas había documentación videográfica de penaltis en la liga de fútbol profesional española, apenas había muestras disponibles para la implementación de una red neuronal. Es por eso por lo que se tuvieron que aplicar técnicas de *data augmentation*, como el ruido gaussiano que permitió aumentar de 1048 hasta 4191 muestras.

En cuanto a la implementación del modelo tampoco ha sido del todo satisfactoria, porque se sufrió *overfitting* desde las primeras pruebas sin encontrar una manera de reducirlo considerablemente. Seguramente este sobreajuste proviene de un mal entrenamiento ya que se hizo con un *dataset* generado a partir de solo 1048 muestras y ampliado mediante ruido gaussiano. Además de este problema, la precisión en el conjunto de validación del *dataset* tampoco ha sido buena, sobre todo en las predicciones de las direcciones.

Respecto a la reducción de dimensionalidad de la red neuronal, la técnica PCA (*Principal Component Analysis*) sí se ha podido demostrar que funcionaba correctamente pudiendo reducir las características de entrada de once a ocho o nueve sólo, obteniendo resultados prácticamente iguales en ambas. Sin embargo, la técnica LDA (*Linear Discriminant Analysis*) a pesar de implementarla, no se pudo demostrar su efectividad.

En definitiva, uno de los mayores problemas encontrados en la red neuronal, no viene en sí de la red neuronal, viene del *dataset*, ya que, al haber sido creado para este TFG, puede tener errores o no estar bien clasificado. Y no es noticia que con un mal conjunto de datos no se pueden obtener resultados de datos.

Y, como conclusión final, este TFG ha supuesta una primera toma de contacto en el mundo de la inteligencia artificial, adquirir conocimientos y experiencia trabajando en el desarrollo del *dataset* y del modelo, y muchas ganas de seguir descubriendo y aprendiendo más.

7. Futuros proyectos

En cuanto a posibles líneas de futuro de este TFG, hay varias a destacar debido a que no se han obtenido buenos resultados, por lo que se podría mejorar en muchos ámbitos.

Primero, con un *dataset* bien clasificado y estructurado, seguramente los resultados serían bastante mejores. Por ejemplo, colaborar con algunas empresas del sector como *Olocip*, *InStat*, *StatsBomb* u *Opta Sports*, que son empresas de análisis deportivo, algunas de ellas especializadas en el fútbol, y que tienen el *dataset* de penaltis perfecto para esta idea.

En cuanto a la mejora del *dataset* actual, en un futuro se podría hacer una labor de actualización constante, es decir, recopilar los nuevos penaltis que van sucediendo a lo largo de las jornadas. También se podrían añadir nuevas características que pueden ser determinantes y está demostrado, como el tiempo que pasa desde que un árbitro señala el penalti hasta que el jugador atacante lanza el penalti, porque cuanto más tiempo pasa más baja la probabilidad de meter gol, quizás por la presión e incertidumbre y tener más tiempo para tomar una decisión de dónde y cómo lanzar el penalti. Otro ejemplo que influye en los lanzamientos de penalti es si detrás de la portería donde lanza está su propia afición o la del equipo rival, ya que, si se tiene a los propios aficionados detrás animando al lanzador o metiendo presión al portero, hay más probabilidad de marcar el penalti.

En adición a esto último, se podrían añadir también penaltis de otras competiciones, aunque eso ampliaría el número de jugadores de los que se obtienen datos. Ya que, si por ejemplo se añade competiciones europeas, se obtendrían datos de un portero de un club italiano que se enfrentó a un club español, pero sólo se tendría una muestra de ese jugador. Con lo cual, sería mejor que si se hace una ampliación en cuanto a las competiciones que se quieren registrar penaltis, lo suyo sería ampliar varias ligas nacionales también que conforman a dichas competiciones europeas. Además, se podrían añadir también tandas de penalti de cara a ampliar el número de muestras recopiladas, pero esto va en concordancia con la recopilación de penaltis en competiciones europeas o internacionales, debido a que en ligas nacionales no existe el concepto de tandas de penalti. En el caso de las tandas de penalti, se podría registrar quién lanza primero, ya que el equipo que lanza primero en tandas de penalti suele ganar un 60% de las veces la tanda, porque si marca el penalti después el rival tiene la presión de marcar, pero si falla hay opción de que el rival también falle.

También se podría considerar el añadir la forma de golpeo, ya que no todos los golpes de penalti son iguales, el golpeo puede ser: con el empeine, con el interior, de panenka¹⁸; y ver qué tan importante es el golpeo o incluso la posición del cuerpo, ya que, si a la hora de golpear se tiene el cuerpo demasiado inclinado hacia atrás, hay muchas posibilidades de lanzar el balón muy por encima de la portería.

Y, por último, respecto al *dataset*, se podría hacer un modelo capaz de determinar la dirección tanto del penalti como de dónde se lanza el portero mediante reconocimiento de imágenes, evitando así el error humano de tener que tomar una decisión acerca de a qué zona

¹⁸ Un penalti a lo panenka es un disparo de penalti que consiste en engañar al portero con que vas a lanzar hacia un lado, pero terminas picando el balón suavemente. Obtiene ese nombre gracias al jugador checoslovaco Antonín Panenka, siendo el primer futbolista en lanzar un penalti de esta manera en la Eurocopa de 1976 durante una tanda de penaltis.

se ha lanzado. Aunque esto sería lo ideal, también supondría una labor de búsqueda de imágenes del lanzamiento de penalti desde el mismo ángulo.

Con estas posibles mejoras, se podría seguir utilizando todo lo aprendido e implementado en cuanto a la red neuronal, pudiendo mejorar los resultados obtenidos hasta ahora.

Por otro lado, se podría realizar una labor de ajuste de los hiperparámetros de la red neuronal combinado con las diferentes técnicas para reducir *overfitting* en busca de resolver este problema a la vez que se mejoran los resultados obtenidos del modelo y también la optimización de este. Esto puede suponer trabajar durante una cantidad de tiempo considerable sólo en la realización de pruebas con diferentes ajustes. Esto, combinado con la posibilidad de determinar la dirección del lanzamiento mediante reconocimiento de imágenes y adición de características potencialmente importantes, podría suponer un gran avance en el proyecto de cara a predecir los lanzamientos de penalti.

Bibliografía

- [1] J. (2018, 20 julio). ¿Por qué no se usa el “Ojo de Halcón” en el fútbol? Naukas. <https://naukas.com/2018/07/20/por-que-no-se-usa-el-ojo-de-halcon-en-el-futbol/>
- [2] EcoDiario.es. (2017, 27 enero). Rafa Nadal y el épico ojo de halcón que le salvó en la semifinal del Open de Australia contra Dimitrov. El Economista. <https://ecodiario.eleconomista.es/tenis/noticias/8113609/01/17/Rafa-Nadal-y-el-epico-ojo-de-halcon-que-casi-le-salvo-en-la-semifinal-del-Open-de-Australia-contra-Dimitrov.html>
- [3] Regalado, I. (2019, 9 mayo). La inteligencia artificial es capaz de predecir el riesgo de lesión. La Razón. <https://www.larazon.es/atusalud/salud/la-inteligencia-artificial-es-capaz-de-predecir-el-riesgo-de-lesion-DF23267114/>
- [4] Díaz, Á. (2019, 4 febrero). 100.000 penales bajo la lupa: Descubre los mejores lanzadores del mundo. Marca. <https://www.marca.com/claro-mx/futbol-internacional/2019/02/04/5c582fdb46163fd1248b45b5.html>
- [5] Russell, S. J., & Norvig, P. (2021). Inteligencia Artificial Un Enfoque Moderno (2.a ed.). PRENTICE HALL/PEARSON.
- [6] Royaltic Group. (s. f.). INTELIGENCIA ARTIFICIAL. Recuperado 29 de junio de 2021, de <http://www.royalticgroup.com/inteligencia-artificial.html>
- [7] Iberdrola. (2020, 30 abril). ¿Qué es la inteligencia artificial? <https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial>
- [8] Wikipedia. (2021, 22 mayo). R.U.R. (Robots Universales Rossum). Wikipedia, la enciclopedia libre. [https://es.wikipedia.org/wiki/R.U.R._\(Robots_Universales_Rossum\)](https://es.wikipedia.org/wiki/R.U.R._(Robots_Universales_Rossum))
- [9] Turing, A. M. (1936, 12 noviembre). On Computable Numbers, with an Application to the Entscheidungsproblem. The Graduate College, Princeton University, New Jersey, USA. https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf
- [10] Llaca, M. (2016, 4 julio). La máquina Z3 de Konrad Zuse. Parcela Digital. <https://parceladigital.com/2016/07/04/la-maquina-z3-de-zuse>
- [11] Magiquo. (2020, 23 enero). Redes neuronales o el arte de imitar el cerebro humano. <https://magiquo.com/redes-neuronales-o-el-arte-de-imitar-el-cerebro-humano/>
- [12] LaLiga Santander [laliga]. (2021). LaLiga Santander. YouTube. <https://www.youtube.com/user/laliga>

Anexo. Acrónimos y siglas

| | |
|-------------------|---|
| TFG | Trabajo Fin de Grado |
| Km/h | Kilómetros por hora |
| VAR | Video Assistant Referee |
| IA | Inteligencia Artificial |
| SNARC | Stochastic Neural-Analog Reinforcement Calculator |
| MIT | Massachusetts Institute of Technology |
| IBM | International Business Machines Corporation |
| Prolog | PROgrammation en LOGique |
| SARS-CoV-2 | Enfermedad infecciosa causada por un coronavirus |
| Covid-19 | Enfermedad infecciosa causada por un coronavirus |
| MSE | Mean Squared Error |
| MAE | Mean Absolute Error |
| RMSProp | Root Mean Square Propagation |
| Adam | Adaptive Moment Estimation |
| ReLU | Rectified Linear Unit |
| TanH | Tangente Hiperbólica |
| R | Right-handed (diestro) |
| L | Left-handed (zurdo) |
| LT | Left-Top (izquierda-arriba) |
| CT | Center-Top (centro-arriba) |
| RT | Right-Top (derecha-arriba) |
| LC | Left-Center (izquierda-medio) |
| CC | Center-Center (centro-medio) |
| RC | Right-Center (derecha-medio) |
| LD | Right-Down (izquierda-abajo) |
| CD | Center-Down (centro-abajo) |
| RD | Right-Down (derecha-abajo) |
| PCA | Principal Component Analysis |
| LDA | Linear Discriminant Analysis |