Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros Informáticos
Departamento de Inteligencia Artificial

Artificial Neural Networks and Deep Learning

Curso 2022/23

Hands-On Project
Building an Artificial Neural Network

Autores	
Markel Tyan Tyan	markel.tyan@alumnos.upm.es
Íñigo Rodríguez Sánchez	i.rsanchez@alumnos.upm.es
Iago Zamorano Chouciño	iago.zamorano@alumnos.upm.es

Tabla de contenido

Figuras

Tablas

Introducción

A nivel general, tal y como se indica en el enunciado de la práctica, esta actividad pretende seguir el proceso de construcción de una red neuronal artificial profunda para un problema de clasificación utilizando el conjunto de datos Fifa19. El objetivo es estimar la calidad general de los jugadores de fútbol en función de sus habilidades como tarea de clasificación.

Material y entorno de trabajo empleado

El material de trabajo que se proporciona para la realización de la práctica es el siguiente:

- *FootballPlayerRawDataset.csv*: Fichero que contiene el conjunto de datos de los futbolistas a clasificar según sus atributos. Se trata de la información en bruto de la que se extrae el conjunto de datos (limpiados y codificados) a procesar por la red neuronal artificial a implementar.

- *PreparingFootballPlayerDataset.ipynb*: Cuaderno de Google Colab® que se emplea para el preprocesamiento del *dataset* (fichero *FootballPlayerRawDataset.csv*). De esta forma, dicho cuaderno se ejecuta para limpiar y preparar los datos y obtener los archivos resultantes para alimentar los modelos neuronales.

Asimismo, se ha empleado Google Colab® a través de los "cuadernos" (ficheros *.ipynb*), con los que esta herramienta trabaja, para la implementación y ejecución de la red neuronal artificial. Para ello, se ha hecho uso del material de trabajo para la obtención de los ficheros que nutren a la red neuronal artificial que se ha implementado y, posteriormente, se ha llevado a cabo una etapa de ajuste y prueba de los hiperparámetros de la red hasta obtener los resultados más precisos.

Primeros pasos

El primer paso que se ha llevado a cabo para el proceso de diseño de la red neuronal artificial, ha sido el de limpiar y preparar los datos y obtener los archivos resultantes para alimentar los modelos neuronales. Así, se ha utilizado el cuaderno *PreparingFootballPlayerDataset.ipynb* para procesar el *dataset FootballPlayerRawDataset.csv*. El resultado de este proceso es la obtención de los ficheros:

- *FootballPlayerOneHotEncodedClasses.csv*: Incluye las correspondientes clases codificadas mediante *one-hot encoding* (*one-hot encoded classes*).

- *FootballPlayerPreparedCleanAttributes.csv*: Contiene instancias preparadas y limpias para los atributos (predictores).

De forma resumida, el conjunto de datos contiene inicialmente 18.207 instancias con 89 atributos cada una relacionados con jugadores de fútbol. Tras el proceso de preparación y limpieza que se ha mencionado, se acaba con 16.122 instancias con 17 atributos relacionados con las habilidades de los jugadores de fútbol. Estos son: *Crossing, Heading Accuracy, Short Passing, Volleys, Dribbling, Curve, Free Kick Accuracy, Long Passing, Ball Control, Reactions, Shot Power, Stamina, Long Shots, Aggression, Positioning, Vision,* y *Composure.* Estos atributos se han elegido porque sus valores se correlacionan en más de 0,4 o menos de -0,4 con la puntuación global (resultado), que se ha discretizado por cuantiles en cuatro clases o categorías:

1. Malos futbolistas con puntuaciones globales en [46, 62]
2. Intermedios para valores en [63, 66]
3. Buenos futbolistas en [67, 71]
4. Excelentes futbolistas para valores globales en [72, 94].

Con todo ello, durante la etapa de limpieza y preparación de los datos, se han definido los siguientes pasos:

1. Se elimina información irrelevante de los jugadores, como algunas columnas, por ejemplo, sus nombres, o algunas filas como, por ejemplo, los porteros, que tienen atributos diferentes al resto de jugadores.

2. Se eliminan todos los jugadores que tienen atributos en blanco. Se puede realizar esta acción dado que se determina que se dispone de suficientes datos como para no depender de las líneas de jugadores eliminados.

3. Se utiliza una matriz de correlación para observar qué atributos tienen una mayor correlación con la calidad global de un jugador. Se eliminan aquellos

que tienen una correlación menor a 0.4 (en términos absolutos).

4. Se prepara el problema para su resolución: se discretiza la calidad de los jugadores en las cuatro clases que se han definido previamente y se divide el conjunto de datos en los atributos del jugador y la clase a la que pertenece.

5. Se aplica la codificación *one-hot encoding* al conjunto de etiquetas de los jugadores.

6. Se normalizan los atributos de los jugadores.

7. Se exportan los atributos y las clases en los dos archivos que se han indicado previamente.

Con todo ello, y como se ha indicado previamente, el objetivo es construir una red neuronal profunda que clasifique a los jugadores de fútbol por sus puntuaciones totales basándose en las puntuaciones de sus habilidades. Para ello, antes de empezar a construir la red se ha necesitado dividir los datos en 3 conjuntos:

- Conjunto de datos de entrenamiento: Este conjunto se ha usado para entrenar la red neuronal artificial y constituye un 80% de los datos totales.

- Conjunto de datos de validación: Este conjunto se ha empleado para comprobar el rendimiento de la red neuronal artificial que se implementa a lo largo del proceso de entrenamiento de la misma. Este proceso, se realiza con datos diferentes al conjunto de datos de entrenamiento, haciendo uso de un 10% de los datos totales que se ha reservado para ello.

- Conjunto de datos de test: Este conjunto se utiliza una vez se construye la red neuronal artificial y se refinan sus hiperparámetros. Así, de acuerdo a la arquitectura final que se le otorga a la red neuronal, se evalúa de forma definitiva el rendimiento de la misma a partir de este conjunto de datos. Constituye el 10% restante de los datos totales.

Proceso de diseño

El problema que se pretende solucionar con el diseño de la Red Neuronal Artificial es el de la clasificación de los jugadores de fútbol en 4 categorías que se han discretizado de forma manual en función de diferentes características relativas a su rendimiento.

La metodología que se ha seguido para cada iteración ha sido la siguiente:

- Se diseña una nueva arquitectura de red neuronal, incluyendo el número de capas y el número de neuronas en cada capa, así como se definen el resto de hiperpárametros.

- Seguidamente, haciendo uso del porcentaje de acierto respecto al porcentaje de acierto obtenido para el conjunto de test se realiza un ajuste de los hiperparámetros para mejorar los resultados. El objetivo es reducir tanto el sesgo como la varianza en los resultados. El error humano a considerar es del 10%.

- Si se observa un alto sesgo, se siguen algunas de las técnicas que permiten ajustarse más al conjunto de entrenamiento. Por ejemplo:
    - Aumentar la complejidad del modelo (añadir más neuronas o capas).
    - Aumentar el número de iteraciones de entrenamiento.
    - Usar mejores algoritmos de optimización (*Momentum*, *RSMProp*, etc.)
    - Probar con otra arquitectura.

- Si se observa una alta varianza, se aplican métodos como el de regularización o el de *drop out* para reducirla y hacer que el modelo se ajuste menos al conjunto de entrenamiento o se prueba con otra arquitectura.

Finalmente, una vez entrenada la red y ajustados los hiperparámetros, se prueba el modelo con el conjunto de test final y se obtienen los resultados finales.

Primeras pruebas

Para empezar, se ha generado una red profunda de 4 capas ocultas de 750-1000-750-500 neuronas en cada una respectivamente. Se utiliza (DOZAT, 2016) *Nadam* como optimizador dado que es el que mejores resultados suele dar en general. La función de activación del modelo es ReLU, porque se está trabajando con redes profundas y no se quiere acumular el error de *Back-Propagation*. Para los demás parámetros simplemente se usan unos valores arbitrarios, parecidos a los de los ejemplos que se han visto durante el curso. Como se observa en la Tabla 1, el *accuracy* que se ha obtenido en test es del 79,6%, lo que no es un gran resultado, pero significa un buen punto de partida. El sesgo obtenido es del 2% y la varianza es del 8%.

Para ver si se podía bajar la varianza obtenida, que es muy alta, se ha aumentado la regularización de 0.001 a 0.003. Lo cual ha dado buen resultado bajando la varianza a 0%. En busca de una mejor *accuracy* se ha intentado aumentar el tamaño del *batch* sacrificando un poco de velocidad de ejecución. También se quería probar el optimizador RMSDrop para ver si influiría en el sesgo. El resultado de todos estos cambios ha sido que la *accuracy* en validación apenas ha mejorado, sin embargo, ha aumentado mucho el sesgo subiendo a un 10%.

Tabla 1. Ejecución n.º 1 y n.º 2

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
1	600	0.01	128	[750-1000-750-500]	nadam	0.001	0.2	relu	0.8798	**0.7959**
2	600	0.01	256	[750-1000-750-500]	rmsprop	0.003	0.2	relu	0.8029	**0.8027**

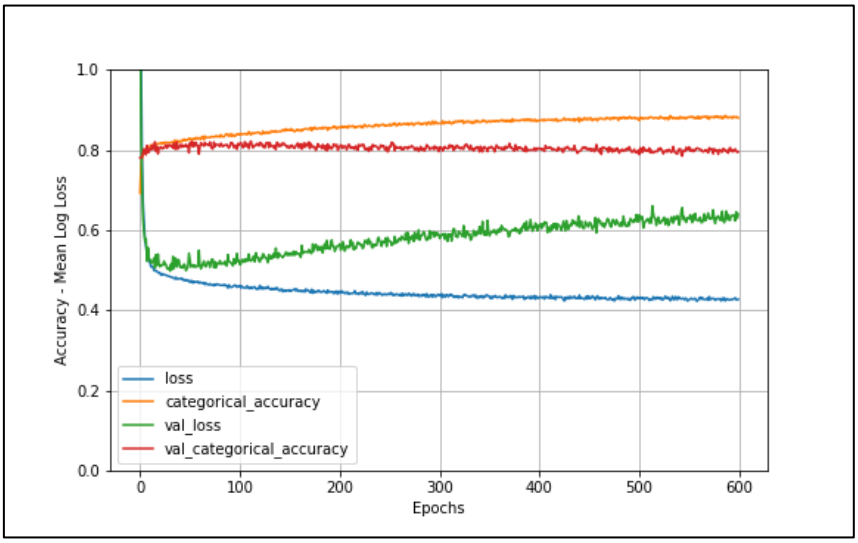

Figura 1. Resultados – Ejecución n.º 1

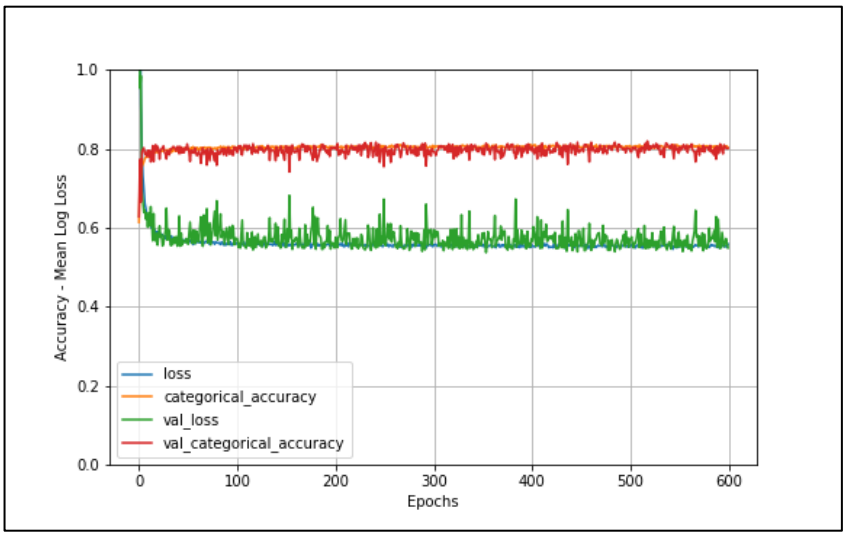

Figura 2. Resultados – Ejecución n.º 2

Para la siguiente iteración se ha intentado reducir ese sesgo tan alto aumentando la complejidad de la red al aumentar el número de capas y, además, aumentar el número de *epochs*. Para que el tiempo de ejecución no sea extremadamente excesivo debido a este cambio, se reduce las neuronas por capa y se disminuye el tamaño del *batch*. También, se ha vuelto a emplear *Nadam* como optimizador ya que el que mejor resultado ha proporcionado. Lamentablemente, como se ve en la Tabla 2, el cambio tanto en la puntuación como en el sesgo es mínimo que sigue siendo de un 10%.

Así se sigue con el objetivo de bajar ese sesgo. En esta ejecución se ha aumentado de forma considerable el número de *epochs*, pero al no obtener cambios significativos, se observa que la mejoría de la red se estanca en las últimas iteraciones y que entrenar durante más tiempo (*epochs*) no aporta grandes mejorías, por lo que se mantiene ese hiperparámetro y se focaliza el trabajo sobre el resto de parámetros.

Tabla 2. Ejecución n.º 5 y n.º 6

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
5	700	0.01	64	[50-75-100-75-50]	nadam	0.002	0.2	relu	0.7979	**0.8033**
6	1000	0.01	64	[50-75-100-75-50]	nadam	0.002	0.2	relu	0.8018	**0.8021**

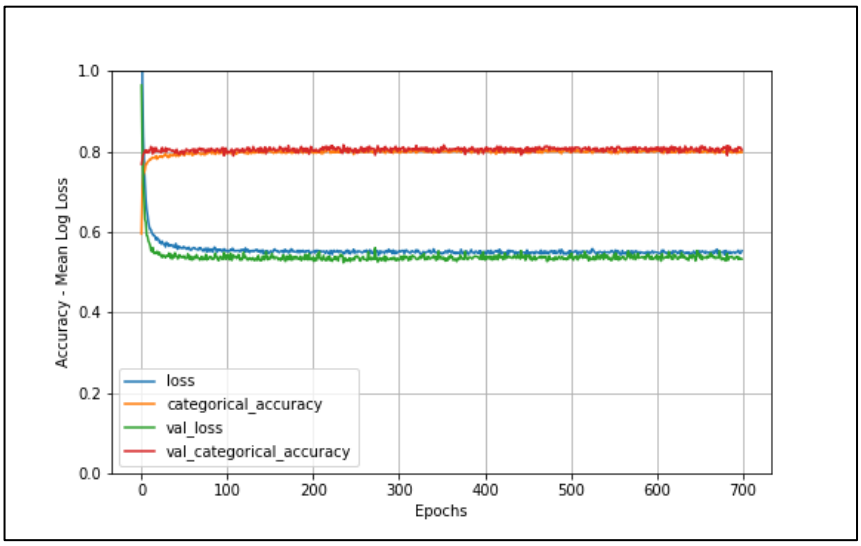

Figura 3. Resultados – Ejecución n.º 5

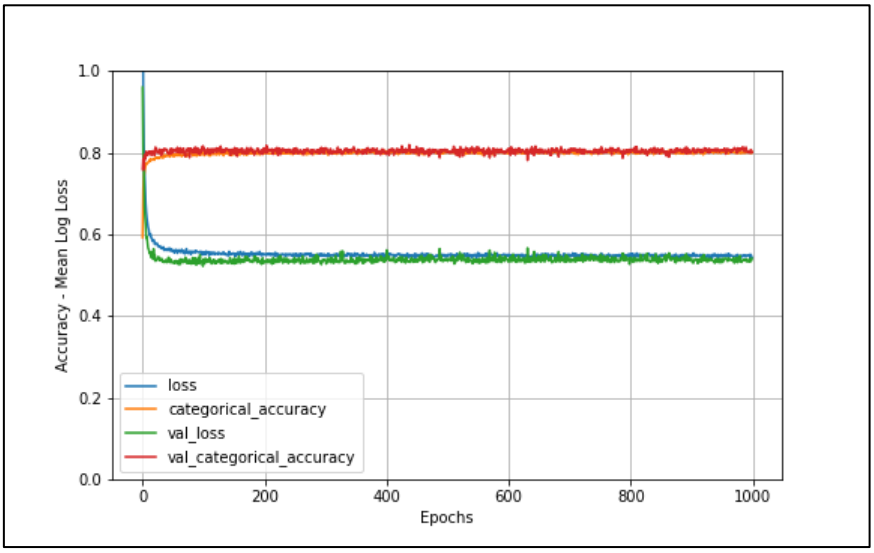

Figura 4. Resultados – Ejecución n.º 6

Al no obtener resultados relevantes, se decide cambiar drásticamente la arquitectura de la red, pasando a 3 capas ocultas con 100 neuronas en cada una, además de disminuir a la mitad el tamaño del *batch*. Esto permite realizar ejecuciones mucho más rápidas al haber disminuido bastante la complejidad, con la ventaja de que no afecta casi al resultado. Se realiza la ejecución para comprobar si se produce algún cambio en los resultados. Como se observa en la Tabla 3, los cambios son mínimos respecto a ejecuciones previas.

Así, se pretende ver cuánto afecta el tamaño del *batch* a la *accuracy*, se aumenta a 128. Adicionalmente, se reduce el número de neuronas en la capa 2 para probar una arquitectura diferente de nuevo. Se obtiene una mejora en el resultado, ya que la *accuracy* a 81,7% por lo que el sesgo se reduce ligeramente.

Tabla 3. Ejecución n.º 7 y n.º 8

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
7	1000	0.01	32	[100-100-100]	nadam	0.002	0.2	relu	0.8044	**0.8028**
8	1000	0.01	128	[100-50-100]	nadam	0.002	0.2	relu	0.8073	**0.817**

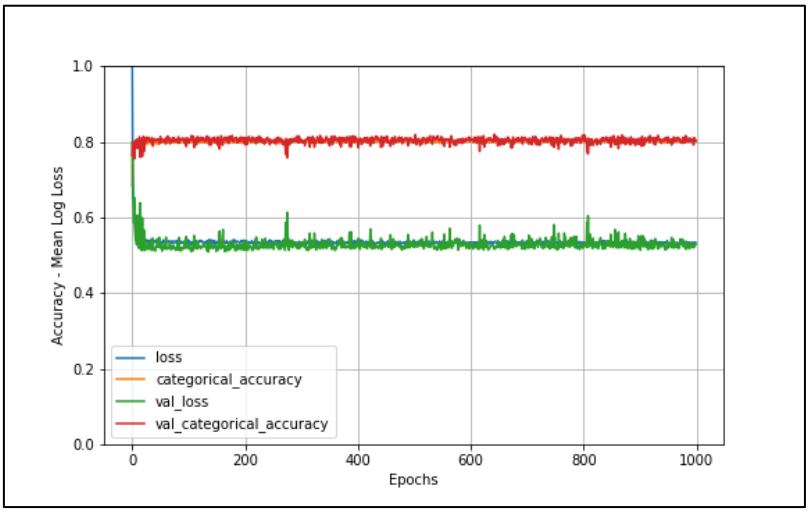

Figura 5. Resultados – Ejecución n.º 7

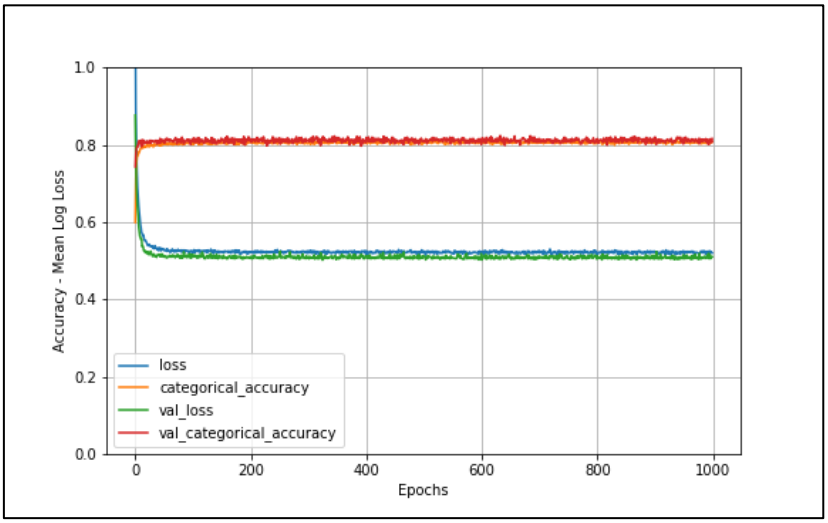

Figura 6. Resultados – Ejecución n.º 8

A partir de lo que se ha visto antes, el aumento de *epochs* no ha proporcionado mejora alguna así que se vuelve a bajar para no entrenar el modelo más tiempo del necesario. Por otro lado, se pretende probar qué optimizador es más adecuado para el modelo, por tanto, se prueba con el descenso de gradiente estocástico. En la Tabla 4 se observa que el cambio a pesar de ser a peor es mínimo, por ello se considerarán ambos optimizadores en un futuro.

Tabla 4. Ejecución n.º 10

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
10	700	0.01	128	[100-50-100]	gradient_descent	0.002	0.1	relu	0.8163	**0.8089**

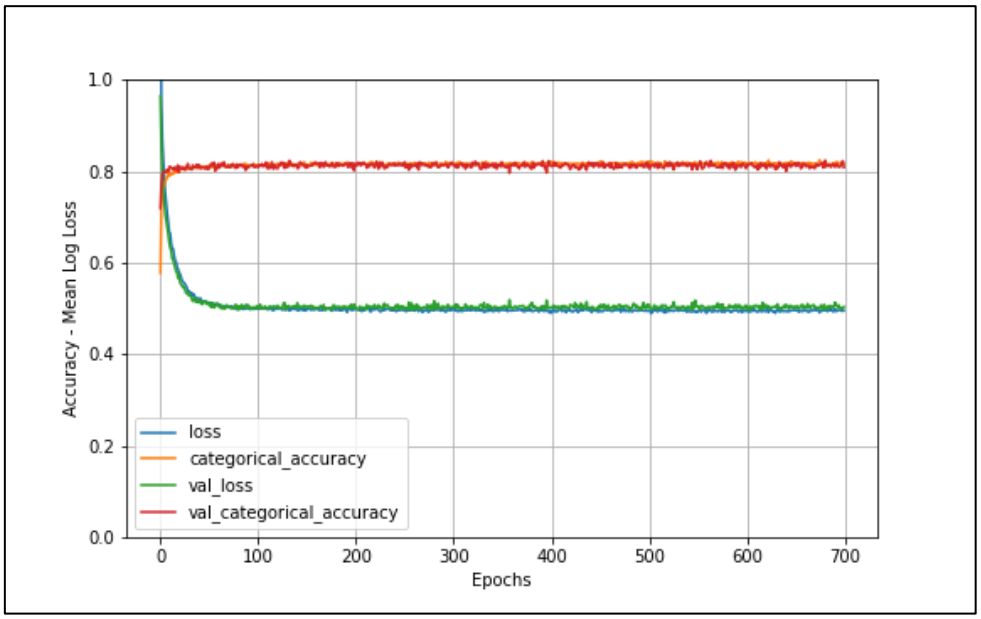

Figura 7. Resultados – Ejecución n.º 10

Continuando con el ajuste de la arquitectura de la red neuronal, se identifica que, para arquitecturas más pequeñas de la red, el resultado que se obtiene es similar, por lo que no es necesario utilizar redes con un número de neuronas tan elevado como se han empleado en ejecuciones previas. Por ello, en las siguientes ejecuciones se trata de limitar el número máximo de neuronas por capa de la red a 50 neuronas, por cuestiones de eficiencia y carga computacional. Asimismo, se mantiene *SGD* como optimizador. De esta forma, se identifica una pequeña mejora en la *accuracy*, obteniendo así el mejor resultado en esta primera fase de pruebas (ver Tabla 5).

Tabla 5. Ejecución n.º 11

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
11	700	0.01	256	[25-50-25]	gradient_descent	0.002	0.1	relu	0.8032	**0.8188**

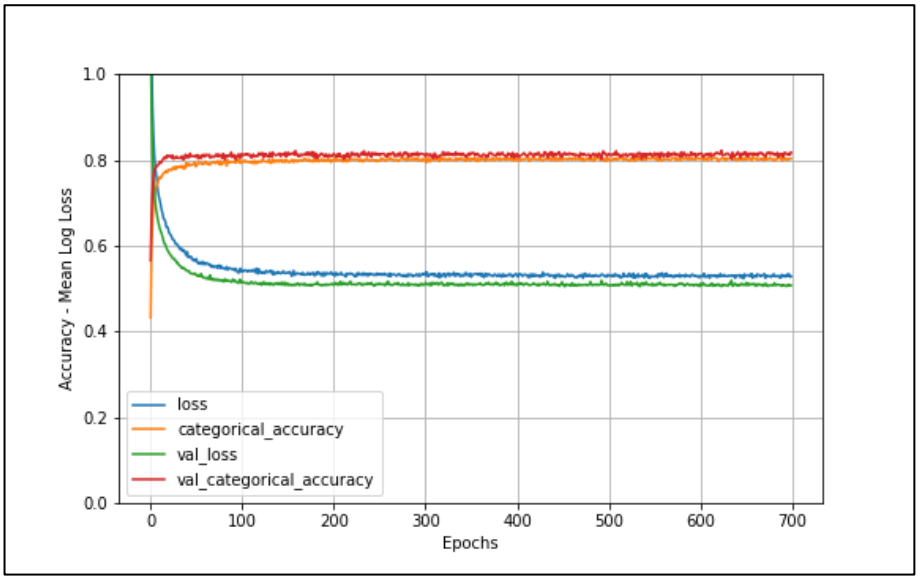

Figura 8. Resultados – Ejecución n.º 11

Tras haber obtenido el merjo resultado hasta el momento, se trata de aumentar la complejidad de la red neuronal para seguir reduciendo el sesgo. Por ello, se introduce una capa más y se disminuye ligeramente el número de neuronas en cada capa, con la intención de no aumentar en exceso la complejidad. Asimismo, se vuelve a emplear *Nadam* como optimizador (ver Tabla 6).

Tabla 6. Ejecución n.º 14 y n.º 15

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
14	1000	0.01	128	[15-30-30-15]	nadam	0.002	0.1	relu	0.7930	**0.7977**
15	1000	0.01	256	[15-30-30-15]	nadam	0.002	0.1	relu	0.7963	**0.7866**

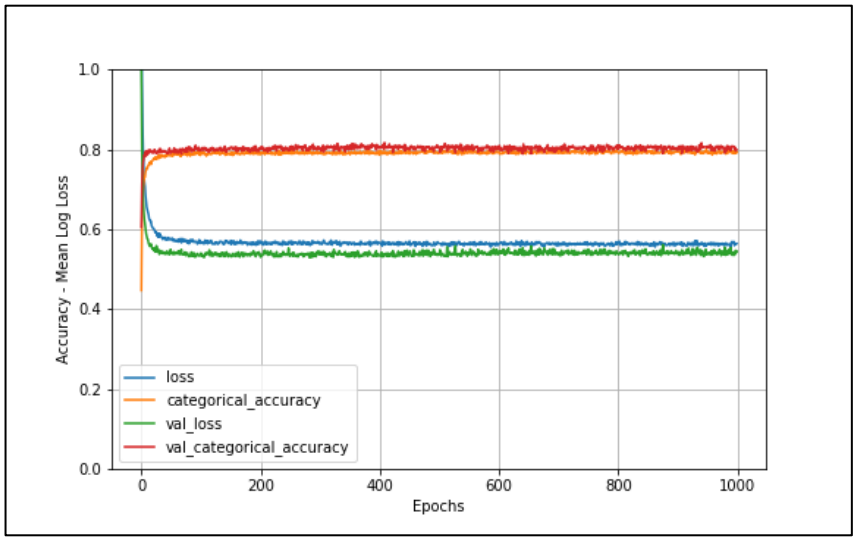

Figura 9. Resultados - Ejecución n.º 14

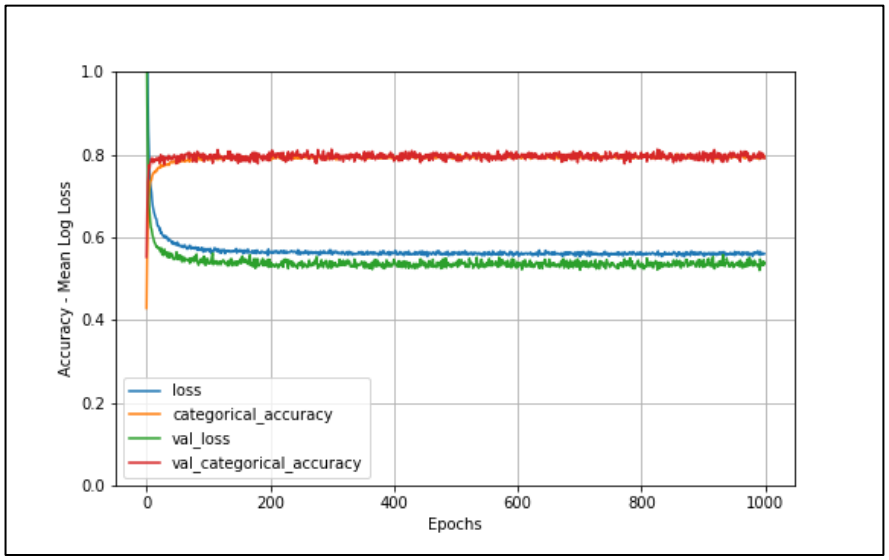

Figura 10. Resultados – Ejecución n.º 15

Proceso de limpieza y preparación de datos - Variaciones

Tal y como se ha visto en el apartado previo, tras un proceso de realización de, todo tipo de pruebas con distintos hiperparámetros para la red sin obtener grandes mejoras, se decide hacer algún cambio en el proceso de limpieza para ver si así podíamos obtener alguna mejora.

Primeramente, se modifica el proceso relativo a la normalización de los datos utilizando la estandarización en lugar de la normalización *min-max*. Este cambio no se tradujo en ninguna mejora de la red ya que el *dataset* con que se cuenta no tiene apenas *outliers* y por tanto es indiferente utilizar una u otra técnica ya que los resultados son muy similares.

A continuación, se pone el foco en la parte de la limpieza en la que se lleva a cabo un estudio de las variables más correlacionadas con la variable objetivo, tras el cual se descarta una parte de ellas por no superar un valor de 0 '4 de correlación. Así, se decide probar a entrenar la red con todas las columnas numéricas disponibles para ver si esto contribuye con algún tipo de mejora en su rendimiento.

Tras probar la red con el dataset completo se observa una mejora general del orden del 10% en la precisión de la red, rondando así el 90% de precisión sobre el conjunto de entrenamiento.

Pruebas con el dataset completo

En la Tabla 7 se puede observar como con los datos nuevos se obtiene una *accuracy* muchísimo mayor que antes, un 89,45%. Se considera este punto, como un punto de inflexión en el que se reinicia el proceso de búsqueda de la mejor combinación de hiperparámetros para maximizar el rendimiento de la red.

Tabla 7. Ejecución n.º 16

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
16	1000	0.1	256	[15-30-30-15]	nadam	0.001	0.1	relu	0.9039	**0.8945**

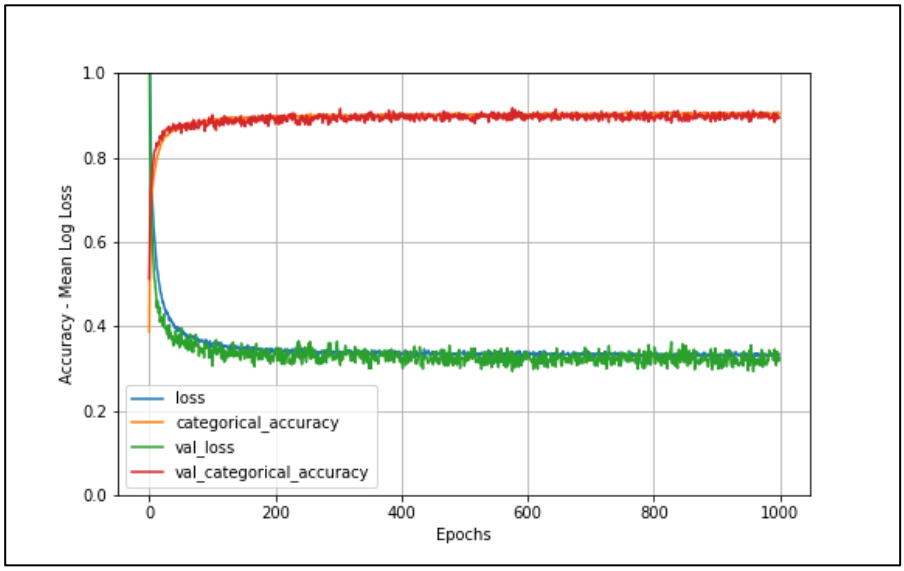

Figura 11. Resultados – Ejecución n.º 16

De esta forma, el primer planteamiento ha sido el de utilizar la arquitectura que mejor resultado proporciona previo al paso de variación en la limpieza y preparación de los datos. Así, tras unas primeras pruebas de ejecución, en las que se observa inestabilidad en la precisión tanto de validación como de entrenamiento según avanzan las *epochs*. Por ello, se comienza a emplear un tamño de *batch* mucho mayor.

En las ejecuciones que se muestran en la Tabla 8 se observa que se han probado distintos optimizadores además del *SGD*, utilizando de nuevo *Nadam* y *RMSProp,* aplicando para estos dos últimos una reducción en la tasa de aprendizaje (*learning rate*) al observar en las primeras pruebas con cada uno de ellos (ejecuciones 27 y 29 que no se muestran en la tabla) un entrenamiento muy inestable.

Finalmente, se supera por primera vez una *accuracy* del 90%, siendo hasta el momento el mejor resultado.

Tabla 8. Ejecución n.º 24-26-28

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
24	700	0.01	1024	[25-50-25]	gradient_descent	0.002	0.2	relu	0.8817	**0.8734**
26	700	0.001	1024	[25-50-25]	nadam	0.002	0.2	relu	0.8779	**0.8548**
28	700	0.001	512	[25-50-25]	rmsprop	0.003	0.15	relu	0.8694	**0.9001**

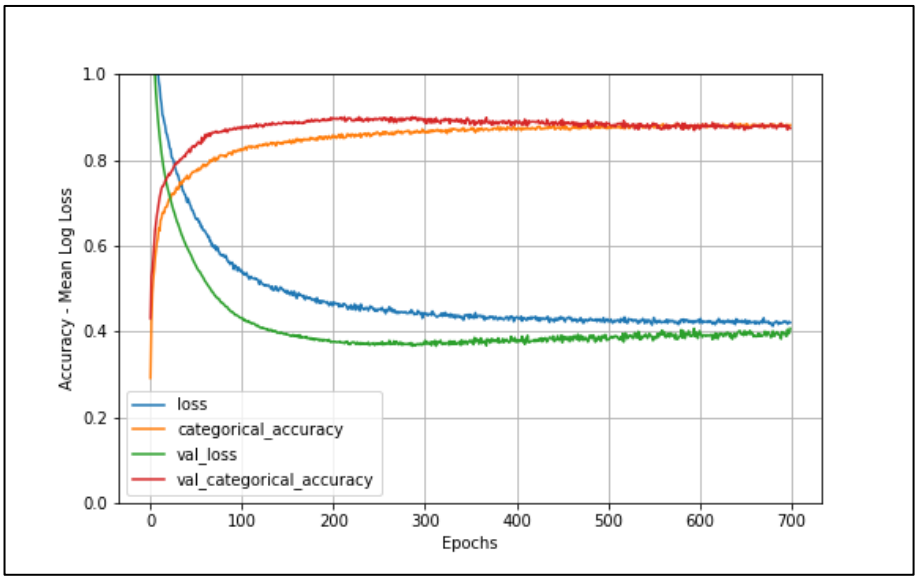

Figura 12. Resultados – Ejecución n.º 24

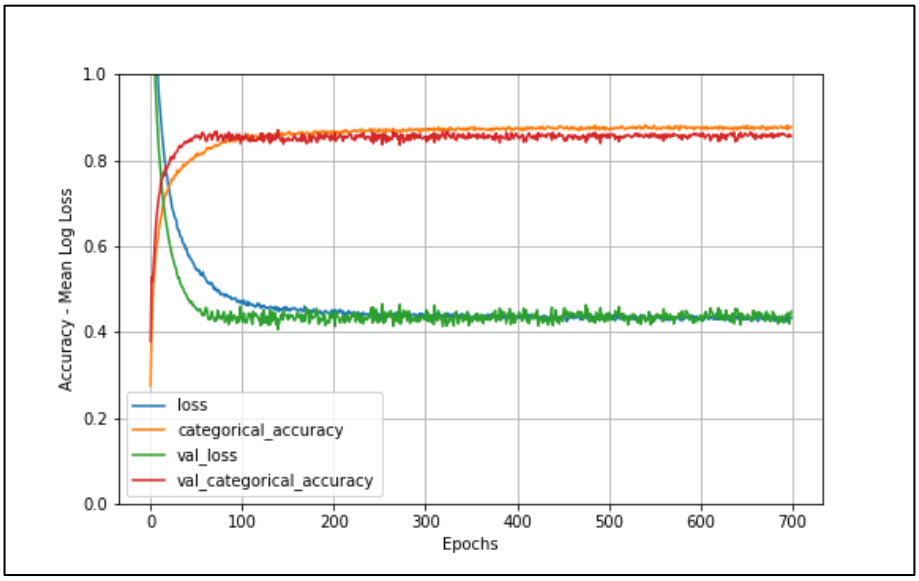

Figura 13. Resultados – Ejecución n.º 26

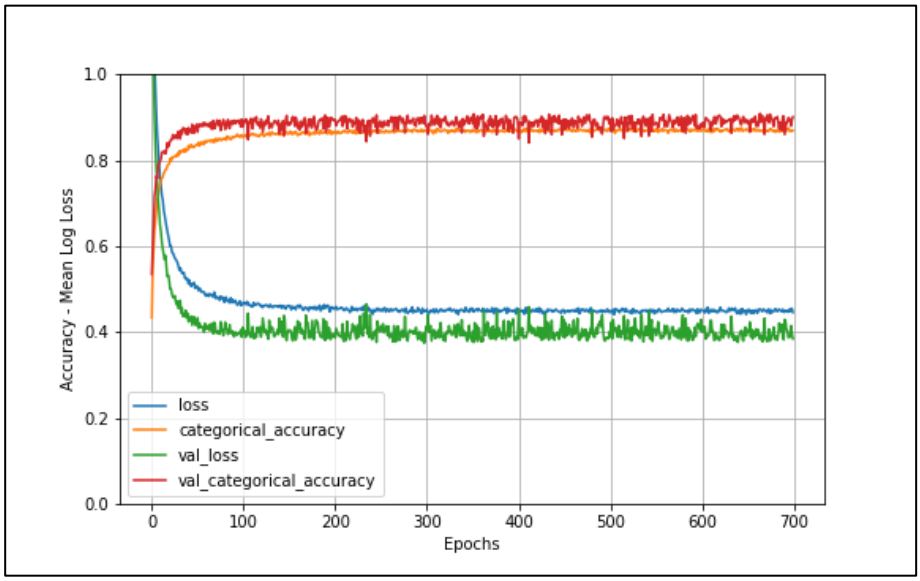

Figura 14. Resultados – Ejecución n.º 28

Tras haber obtenido en la última ejecución el mejor resultado obtenido hasta el momento, se decide probar los mismo hiperparámetros, con la salvedad de aplicar un aumento en el número de *epochs*, que permitan comprobar si con un entrenamiento de mayor duración se obtiene una mejor *accuracy*. Con todo ello, a patir de los resultados que se han obtenido, se identifica que la mejora en la *accuracy* es mínima y se observa que una mayor duración apenas aporta mejorías en dichos resultados.

Tabla 9. Ejecución n.º 36

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
36	1000	0.001	512	[25-50-25]	rmsprop	0.001	0.1	relu	0.9041	**0.9081**

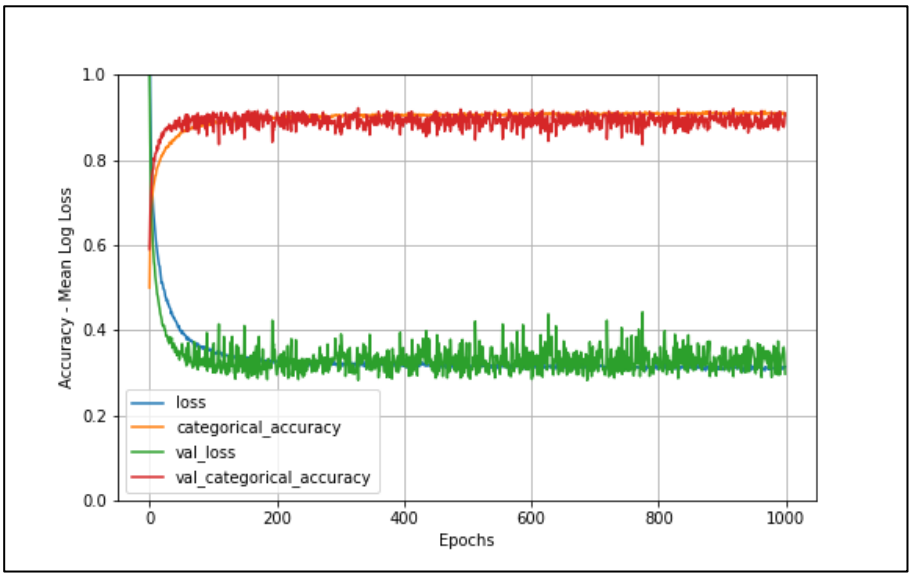

Figura 15. Resultados – Ejecución n-º 36

El siguiente paso, ha sido aumentar la complejidad de la red neuronal, con el objetivo de disminuir el sesgo. Para ello, se plantea una arquitectura diferente a la anterior, en la que se añade una capa más, con un número de neuronas en consonancia con las otras. Además, se sigue empleando *Nadam* como optimizador ya que en base a la experiencia obtenida es que mejores resultados ha proporcionado.

A partir de este punto, con la arquitectura y el optimizador ya fijados, se empiezan a probar diferentes valores para los hiperparámetros *learning rate*, *batch size, regularization rate* y *dropout rate* con el objetivo de encontrar el balance adecuado entre sesgo y varianza, así como la mayor *accuracy* posible en test.

Tabla 10. Ejecución n.º 41-46

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
41	1000	0.001	1024	[50-25-25-50]	nadam	0.002	0.1	relu	0.9063	**0.9137**
42	1000	0.001	1024	[50-25-25-50]	nadam	0.003	0.15	relu	0.8811	**0.9094**
43	1000	0.001	1024	[50-25-25-50]	nadam	0.003	0.05	relu	0.9064	**0.9137**
44	1000	0.001	1024	[50-25-25-50]	nadam	0.002	0.05	relu	0.9209	**0.912**
45	1000	0.001	2048	[50-25-25-50]	nadam	0.002	0.05	relu	0.9142	**0.9143**
46	1000	0.0005	2048	[50-25-25-50]	nadam	0.002	0.05	relu	0.9122	**0.9174**

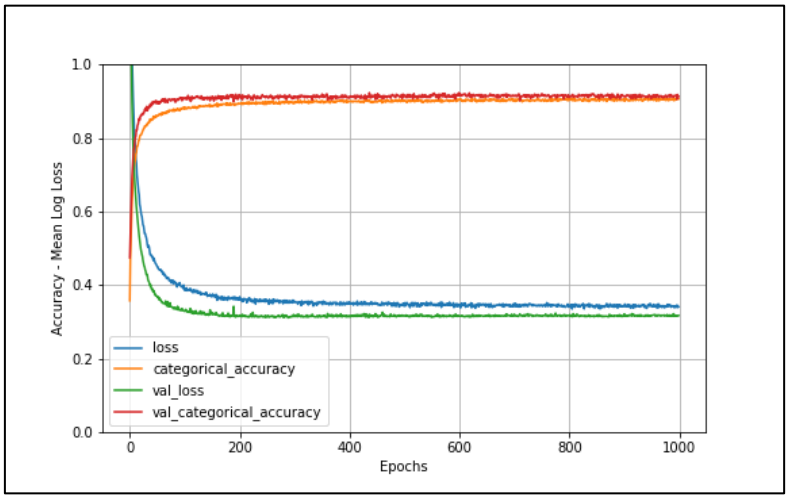

Figura 16. Resultados – Ejecución n.º 41

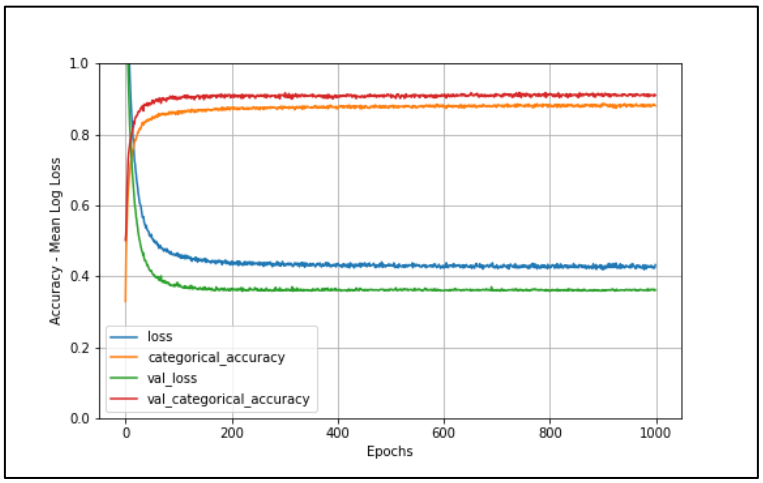

Figura 17. Resultados - Ejecución n.º 42

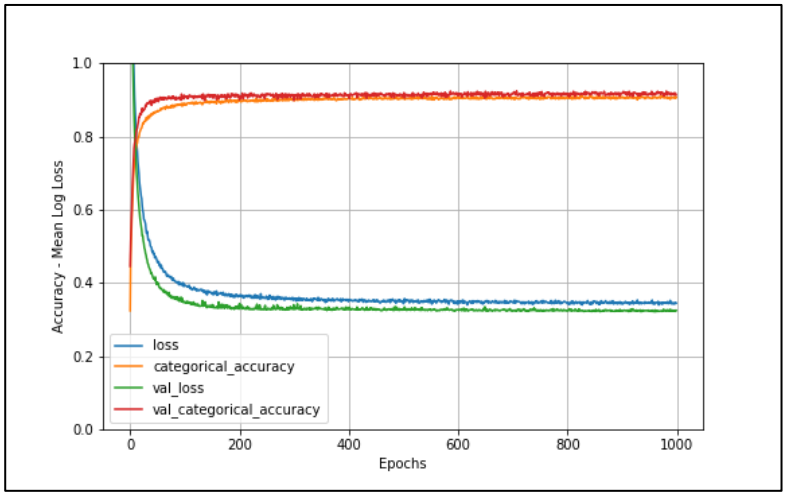

Figura 18. Resultados - Ejecución n.º 43

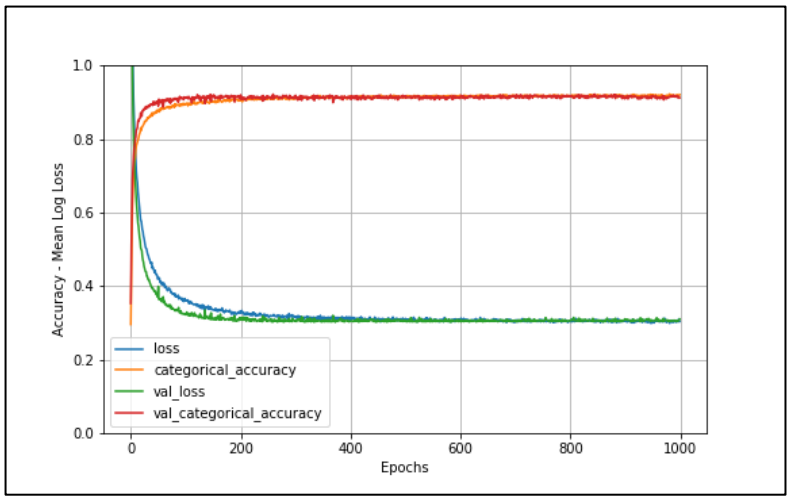

Figura 19. Resultados - Ejecución n.º 44

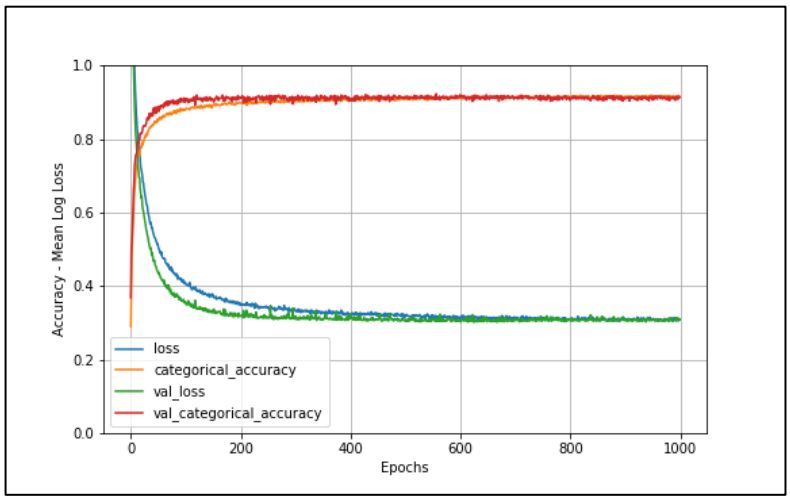

Figura 20. Resultados – Ejecución n.º 45

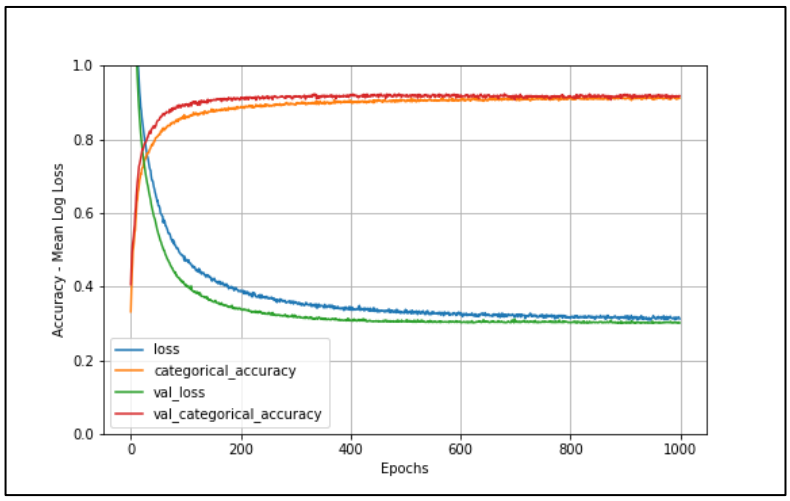

Figura 21. Resultados - Ejecución n.º 46

Resultados finales

Tras el proceso de diseño que se ha llevado a cabo y que se ha descrito en el apartado anterior, se obtiene tiene el mejor resultado en la ejecución n.º 46. Las principales características que definen la red que se ha utlizado son las siguientes:

- Arquitectura: 4 capas con 50-25-25-50 neuronas por capa respectivamente.
- Número de *epochs*: 1000
- *Learning rate*: 0,0005
- *Batch size*: 2048
- Algoritmo de optimización: *Nesterov Adam*
- *Regularization rate*: 0,002
- *Dropout rate*: 5%
- Función de activación: ReLU

Sobre esta red se puede destacar un *learning rate* tan bajo como el definido, así como un número de *epochs* alto contribuyen con una convergencia más progresiva. Asimismo, el empleo de un tamaño elevado del *batch* guarda relación con la estabilidad de la precisión tanto en la validación como en la de entrenamiento. Con todo ello se alcanza una precisión (*accuracy*) sobre el conjunto de test del 91,74%.

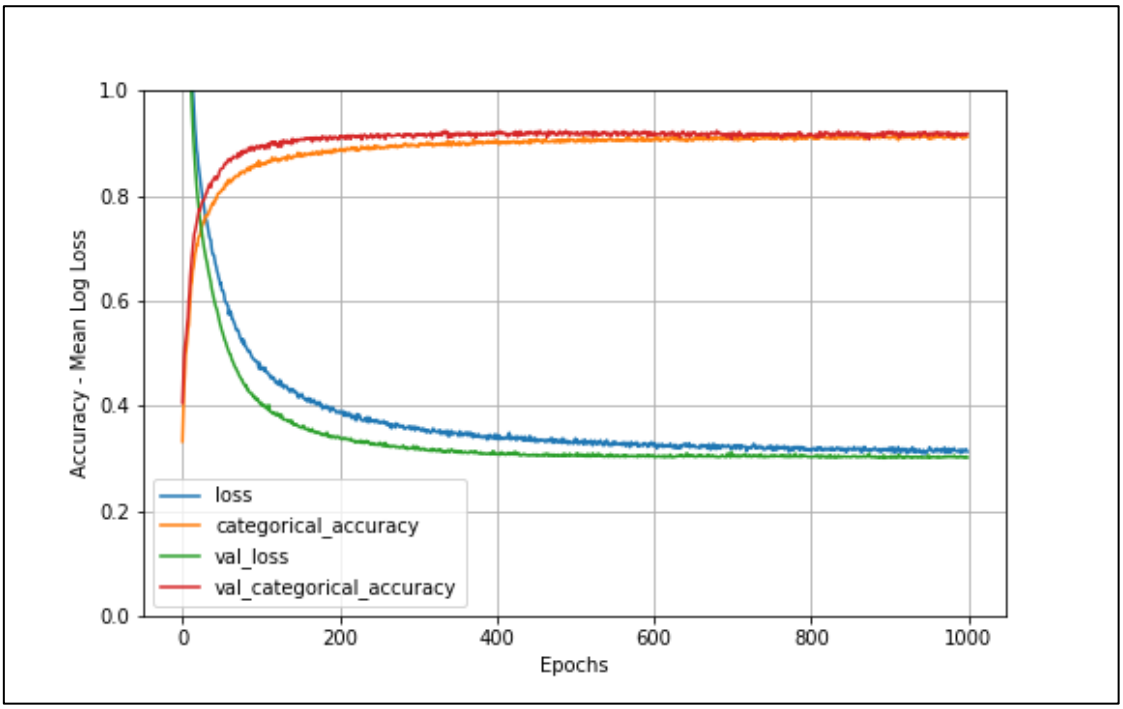

Figura 22. Resultados - Ejecución n.º 46

Conclusiones

El objetivo de la práctica que se ha desarrollado es el diseño de una Red Neuronal Artificial que permite la clasificación de jugadores de fútbol en 4 categorías que se han discretizado de forma manual en función de diferentes características relativas a su rendimiento.

Así, la realización de la práctica ha consistido en un ejercicio de prueba de combinaciones de diferentes hiperparámetros de la red neuronal con el objetivo de obtener la mayor *accuracy* posible sobre el conjunto de test.

El desarrollo de la práctica se divide en dos etapas claramente diferenciadas, siendo el punto de inflexión el incremento en la calidad de los resultados que se han obtenido, determinado por la decisión de emplear el *dataset* completo, volviendo a incluir una serie de columnas de datos descartas en la fase inicial de limpieza y preparación de los datos.

1. En la que se podría considerar como la primera etapa, se alcanza una precisión máxima sobre el conjunto de test de 81,88% (ver Tabla 5 y Figura 8).

2. Seguidamente, en la segunda etapa, los resultados rondan el 90% de precisión sobre el conjunto de test, constituyendo una mejora considerable, siendo el más alto alcanzado un 91,74% (ver Tabla 10 y Figura 22).

Se identifica como principal conclusión la gran relevancia que supone para el resultado final la fase previa de procesamiento, limpieza y preparación de los datos que con los que se entrena y prueba la red.

Finalmente, es importe recalcar que se ha requerido de la realización de una batería considerable de ejecuciones a partir de probar diferentes combinaciones de los hiperparámetros de la red, lo que implica que alcanzar los mejores resultados requiere de dedicación y tiempo.

Anexo I

Tabla 11. Ejecuciones totales realizadas

N.º Ejecución	Epochs	Learning rate	Batch Size	Architecture	Optimizer	Regularization	Dropout	Activation	Train score	TEST SCORE
1	600	0.01	128	[750-1000-750-500]	nadam	0.001	0.2	relu	0.8798	**0.7959**
2	600	0.01	256	[750-1000-750-500]	rmsprop	0.003	0.2	relu	0.8029	**0.8027**
3	700	0.01	128	[500-750-1000-750-500]	rmsprop	0.003	0.2	relu	0.8083	**0.8083**
4	700	0.01	128	[500-750-1000-750-500]	rmsprop	0.003	0.2	elu	0.7695	**0.7785**
5	700	0.01	64	[50-75-100-75-50]	nadam	0.002	0.2	relu	0.7979	**0.8033**
6	1000	0.01	64	[50-75-100-75-50]	nadam	0.002	0.2	relu	0.8018	**0.8021**
7	1000	0.01	32	[100-100-100]	nadam	0.002	0.2	relu	0.8044	**0.8027**
8	1000	0.01	128	[100-50-100]	nadam	0.002	0.2	relu	0.8073	**0.817**
9	1000	0.01	128	[100-50-100]	rmsprop	0.002	0.2	relu	0.8076	**0.8071**
10	700	0.01	128	[100-50-100]	gradient_descent	0.002	0.1	relu	0.8163	**0.8089**
11	700	0.01	256	[25-50-25]	gradient_descent	0.002	0.1	relu	0.8032	**0.8189**
12	700	0.01	256	[20-40-20]	gradient_descent	0.002	0.1	relu	0.7986	**0.8102**
13	700	0.01	256	[20-30-20-30]	adam	0.002	0.1	relu	0.7955	**0.8102**

14	1000	0.01	128	[15-30-30-15]	nadam	0.002	0.1	relu	0.793	**0.7978**
15	1000	0.01	256	[15-30-30-15]	nadam	0.002	0.1	relu	0.7963	**0.7866**
16	1000	0.1	256	[15-30-30-15]	nadam	0.001	0.1	relu	0.9039	**0.8945**
17	1000	0.01	256	[20-40-20]	adam	0.001	0.1	relu	0.8966	**0.8902**
18	700	0.01	1024	[20-40-20]	adam	0.001	0.1	relu	0.9002	**0.8722**
19	700	0.01	1024	[20-40-20]	rmsprop	0.001	0.1	relu	0.9021	**0.9076**
20	700	0.001	1024	[20-40-20]	rmsprop	0.001	0.1	relu	0.8966	**0.879**
21	700	0.001	2048	[20-40-20]	rmsprop	0.001	0.15	relu	0.887	**0.8561**
22	700	0.001	1024	[15-30-15]	rmsprop	0.001	0.15	relu	0.8681	**0.848**
23	700	0.01	1024	[25-50-25]	gradient_descent	0.001	0.15	relu	0.8987	**0.884**
24	700	0.01	1024	[25-50-25]	gradient_descent	0.002	0.2	relu	0.8817	**0.8734**
25	700	0.01	1024	[25-50-25]	nadam	0.002	0.2	relu	0.8758	**0.8238**
26	700	0.001	1024	[25-50-25]	nadam	0.002	0.2	relu	0.8779	**0.8548**
27	700	0.001	512	[25-50-25]	nadam	0.002	0.15	relu	0.8852	**0.889**
28	700	0.001	512	[25-50-25]	rmsprop	0.003	0.15	relu	0.8694	**0.9001**
29	700	0.001	512	[25-50-50-25]	rmsprop	0.003	0.15	relu	0.8723	**0.8592**
30	700	0.001	1024	[25-50-50-25]	rmsprop	0.003	0.15	relu	0.8749	**0.8809**

31	700	0.0001	1024	[25-50-50-25]	rmsprop	0.003	0.15	relu	0.8503	**0.8536**
32	1000	0.0001	1024	[25-50-50-25]	rmsprop	0.001	0.1	relu	0.8904	**0.8703**
33	5000	0.0001	1024	[25-50-50-25]	rmsprop	0.003	0.15	relu	0.8776	**0.8964**
34	1000	0.001	1024	[25-50-25]	rmsprop	0.003	0.15	relu	0.8665	**0.8964**
35	1000	0.001	1024	[25-50-25]	rmsprop	0.001	0.1	relu	0.9072	**0.8964**
36	1000	0.001	512	[25-50-25]	rmsprop	0.001	0.1	relu	0.9041	**0.9082**
37	1000	0.001	1024	[50-25-50]	rmsprop	0.001	0.1	relu	0.91	**0.9063**
38	1000	0.001	1024	[50-25-50-25]	rmsprop	0.001	0.1	relu	0.9199	**0.8976**
39	1000	0.001	1024	[50-25-50]	nadam	0.001	0.1	relu	0.9153	**0.9132**
40	1000	0.001	1024	[50-25-50]	adam	0.001	0.1	relu	0.917	**0.91**
41	1000	0.001	1024	[50-25-25-50]	nadam	0.002	0.1	relu	0.9063	**0.9138**
42	1000	0.001	1024	[50-25-25-50]	nadam	0.003	0.15	relu	0.8811	**0.9094**
43	1000	0.001	1024	[50-25-25-50]	nadam	0.003	0.05	relu	0.9064	**0.9138**
44	1000	0.001	1024	[50-25-25-50]	nadam	0.002	0.05	relu	0.9209	**0.9119**
45	1000	0.001	2048	[50-25-25-50]	nadam	0.002	0.05	relu	0.9142	**0.9144**
46	1000	0.0005	2048	[50-25-25-50]	nadam	0.002	0.05	relu	0.9122	**0.9175**