Universidad Politécnica de Madrid

Máster Universitario en Inteligencia Artificial

Práctica 1: FIFA Players Classification

Asignatura: Artificial neural networks and deep learning

Autores:
Juan José Flores Arellano
Borja Reinoso Hidalgo
Daniel Carmona Pedrajas

Correos:
jj.flores.arellano@alumnos.upm.es
borja.reinoso@alumnos.ump.es
daniel.carmonap@alumnos.upm.es

16 Diciembre 2022

Índice

1. Introducción

Este proyecto tiene como objetivo diseñar una red neuronal artificial para resolver un problema de clasificación supervisada de jugadores del videojuego FIFA 19. Para ello se hará uso de las técnicas y herramientas aprendidas en la clase de "Redes de neuronas artificiales y deep learning".

Se hará uso del lenguaje de programación Python 3 junto a las librerías Tensorflow 2 y Keras. Los entornos utilizados serán: Jupyter notebook en local y Google Colab.

El dataset a utilizar se encuentra aquí. Este dataset contiene 18.207 instancias con 89 atributos cada una. Tras un preprocesado se han reducido tanto las instancias como los atributos a utilizar, 16.122 instancias con 17 atributos cada una. Los atributos están relacionados con la habilidad de cada jugador en ciertas facetas relacionadas con el fútbol: Crossing, Heading Accuracy, Pase Corto, Voleas, Regate, Curva, Precisión de Tiro Libre, Pase Largo, Control del Balón, Reacción, Potencia de Tiro, Resistencia, Tiros lejanos, Agresividad, Posicionamiento, Visión y Compostura.

Para etiquetar las clases se ha hecho una media de los atributos de los jugadores y se ha repartido de la siguiente manera:

1. Poor para medias entre [46,62]
2. Intermediate para medias entre [63,66]
3. Good para medias entre [67,71]
4. Excellent para medias entre [72,94]

1.1. Estructura

La investigación se divide en dos fases:

- Familiarización con hiperparámetros y técnicas de regularización.
- Optimización sistemática de hiperparámetros.

En ambas partes realizaremos cambios atómicos para comprobar el efecto de los mismos en el modelo, pero en la primera fase trabajamos de una forma reactiva, solucionando los problemas que nos surgen para comprender el impacto de cada cambio que hacemos. Una vez interiorizados estos efectos pasamos a comprobar iterar sobre distintos valores para cada hiperparámetro y nos quedamos con el mejor, fijándolo en la configuración y pasando al siguiente. Se pueden encontrar modelos y scripts en este repositorio de GitHub, en él se pueden encontrar los archivos de entrenamiento de modelos, procesado de datos, algunos modelos guardados y parte de los resultados obtenidos durante la práctica.

Por último, presentaremos nuestras conclusiones.

2. Fase 1: Cambios reactivos

Como hemos comentado anteriormente, en esta fase iremos realizando pequeños cambios, observando cómo afectan a los resultados del modelo y al proceso de entrenamiento.

2.1. Arquitectura 1: Feed Forward Neural Network

La primera arquitectura que usaremos es tan simple como:

1. Capa densa con 512 neuronas.

2.1.1. Experimento 1: Configuración base arbitraria

La configuración inicial se puede ver en la tabla 1 y los resultados en la tabla 2.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
100	0.1	512	ReLU	Categorical Crossentropy	SGD	None

Tabla 1: Hiperparámetros para el Experimento 1 de la Arquitectura 1

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	79.38	77.94	15.61	1.44	14
Std	0.05	0.14	0.05	0.19	0

Tabla 2: Resultados del Experimento 1 de la Arquitectura 1 tras 5 entrenamientos

Tener un *bias* alto y una *variance* baja significa que hay margen de mejora antes de llegar al *overfitting* y hay varias posibilidades para conseguir una mejor *accuracy*: añadir más neuronas, entrenar con más *epochs*, ...

2.1.2. Experimento 2: Aumentamos *epochs*

Tras el experimento anterior, nos decantamos por entrenar el modelo durante más *epochs* para reducir el *bias* usando la misma configuración.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	512	ReLU	Categorical Crossentropy	SGD	None

Tabla 3: Hiperparámetros para el Experimento 2 de la Arquitectura 1 tras 5 entrenamientos

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	84.02	81.5	10.97	2.47	199
Std	0.03	0.17	0.03	0.18	8.8

Tabla 4: Resultados del Experimento 2 de la Arquitectura 1 tras 5 entrenamientos

Con respecto al experimento anterior hemos aumentado el *accuracy* tanto en el entrenamiento como en validación, reduciendo así el *bias* del modelo en un 5 % aunque ha aumentado ligeramente el *variance*. Como es lógico el tiempo de entrenamiento ha crecido, aunque no de forma lineal.
En la figura 1 vemos que a partir del epoch 400 no hay una mejora en *accuracy* para el conjunto de validación

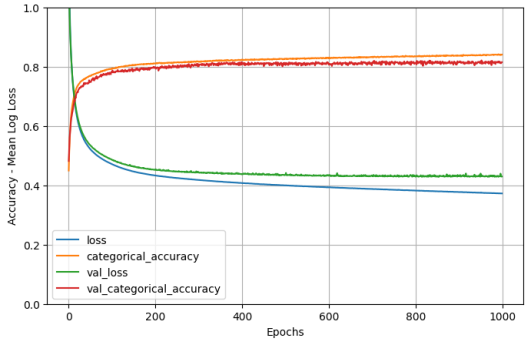


Figura 1: Entrenamiento durante el Experimento 2 de la Arquitectura 1

aunque sí para el conjunto de entrenamiento lo que nos indica que un número de *epochs* tan elevado como el que hemos usado en este experimento con esta arquitectura y configuración conduce a un *overfitting* del modelo.

2.1.3. Experimento 3: Cambiamos a *tanh* y reducimos *epochs*

Para este experimento decidimos reducir las *epochs* ya que como hemos visto en el experimento anterior, no hay una mejora significativa en validación con más epochs. Además de esto, cambiaremos la función de activación a *tanh*. Hasta el momento hemos usado *ReLU* pero no hay razón para usarla para esta arquitectura porque se usa para mitigar el problema del *vanishing gradient* que se da en arquitecturas profundas.
La configuración que usamos para el experimento 3 es:

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	0.1	512	**tanh**	Categorical Crossentropy	SGD	None

Tabla 5: Hiperparámetros para el Experimento 3 de la Arquitectura 1 tras 5 entrenamientos

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	79.58	78.3	15.41	1.28	72
Std	0.2	0.51	0.22	0.33	3.27

Tabla 6: Resultados del Experimento 3 de la Arquitectura 1

Con esta configuración el modelo ha vuelto a aumentar el *bias* y obtenemos unos resultados prácticamente idénticos al experimento uno con la función de activación *ReLU* aunque con más epochs. Parece que con la función *tanh*, el modelo se queda atrapado en mínimos globales como podemos apreciar en la figura 2 y necesita más epochs para escapar de ellos.

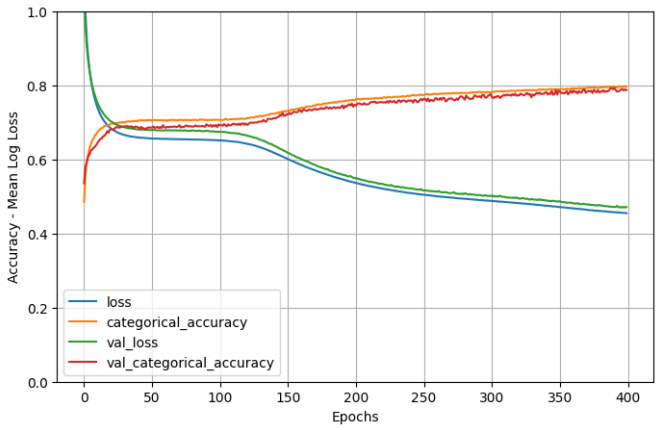


Figura 2: Entrenamiento durante el Experimento 3 de la Arquitectura 1

También observamos que no ha habido overfitting hasta la epoch 400, al contrario de lo que habíamos supuesto al inicio de este experimento.

2.1.4. Experimento 4: Aumentamos *epochs*

El objetivo de este experimento es comprobar cuántas *epochs* podemos realizar antes de que el modelo comience a dirigirse hacia un *overfitting* por lo que la configuración es la misma que en la ejecución anterior, con la diferencia de que incrementamos las epochs a 1000.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	512	tanh	Categorical Crossentropy	SGD	None

Tabla 7: Hiperparámetros para el Experimento 4 de la Arquitectura 1

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	81.17	80.23	13.82	0.94	189
Std	0.14	0.62	0.14	0.53	2.3

Tabla 8: Resultados del Experimento 4 de la Arquitectura 1 tras 5 entrenamientos

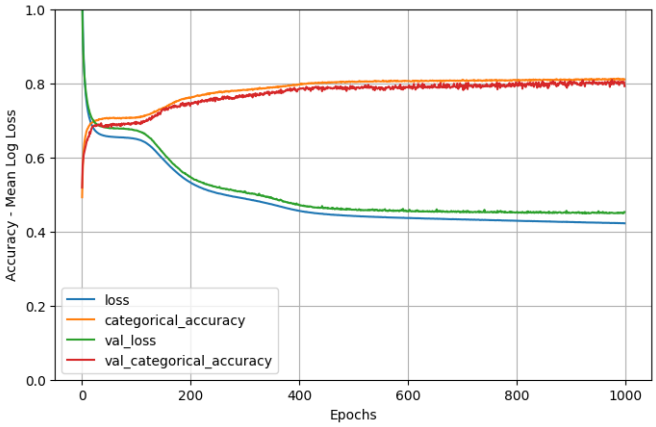


Figura 3: Entrenamiento durante el Experimento 4 de la Arquitectura 1

Aunque hemos doblado las *epochs* con respecto al experimento 3, la mejora ha sido de apenas un 2 % en *accuracy*. Por otra parte, como se muestra en la figura 3, el modelo no ha llegado al punto de *overfitting* aun habiendo usado un número tan alto de epochs. Esto nos indica que todavía hay margen de mejora si seguimos entrenando durante más epochs aunque llevaría mucho tiempo porque el aprendizaje es lento y el modelo parece estancarse en un mínimo local.

2.1.5. Experimento 5: Reducimos *batch size*

Como se necesitaría aumentar exponencialmente el número de *epochs* para conseguir una mejora en el *accuracy*, decidimos reducir el *batch size* y comprobar si de esta forma el modelo consigue mejores resultados.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	**128**	tanh	Categorical Crossentropy	SGD	None

Tabla 9: Hiperparámetros para el Experimento 5 de la Arquitectura 1

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	82.92	80.84	12.07	2.08	517.2
Std	0.19	0.29	0.19	0.28	25.61

Tabla 10: Resultados del Experimento 4 de la Arquitectura 1 tras 5 entrenamientos

Como vemos en la tabla 10 no conseguimos mejorar de forma significativa el *accuracy*. Aumentar el *batch size* ha incrementado exponencialmente el tiempo de entrenamiento por lo que no merece la pena reducir el *batch size* en este caso. Lo que hemos conseguido reduciendo el *batch size* ha sido que el modelo escape rápidamente del primer mínimo local con el que se topa como vemos en la figura 4 así que concluimos que con esta técnica podemos obtener un resultado aceptable en menor tiempo.

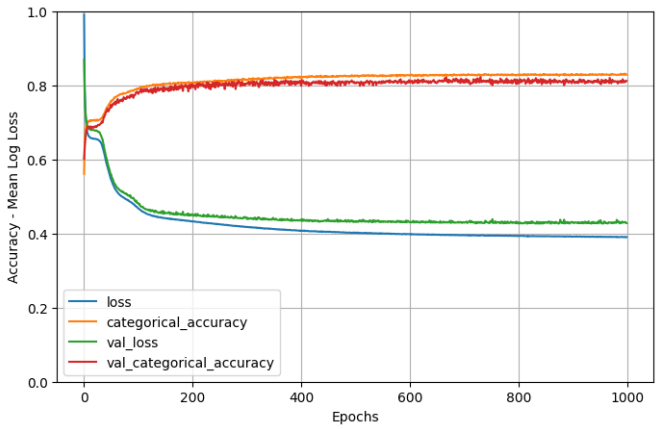


Figura 4: Entrenamiento durante el Experimento 5 de la Arquitectura 1

2.1.6. Conclusiones de la Arquitectura 1

- *ReLU* llega a un estado de *overfitting* con menos *epochs* que *tanh*
- *tanh* tiene un proceso de aprendizaje más lento que *ReLU*
- Reducir el *batch size* implica llegar a un óptimo de forma más rápida con *tanh*.

2.2. Arquitectura 2: Deep Feed Forward Neural Network

Hemos visto que con la arquitectura anterior obtenemos un *accuracy* máximo de un 84 % a partir de donde el aprendizaje es lento y llegamos a un *overfitting* con las técnicas utilizadas. Es por esto que decidimos usar una Deep Feed Forward Neural Network para intentar reducir el *bias* del modelo.
La arquitectura que utilizaremos para la nueva serie de experimentos será:

1. Capa densa de 128 neuronas
2. Capa densa de 64 neuronas
3. Capa densa de 32 neuronas

2.2.1. Experimento 1: Comparación con arquitectura 1

En este experimento utilizaremos la misma configuración que en el experimento anterior 2.1.5:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	128	tanh	Categorical Crossentropy	SGD	None

Tabla 11: Hiperparámetros para el Experimento 1 de la Arquitectura 2

Hemos obtenido un modelo con un *overfitting* muy alto, tan alto que ha cometido menos error que un humano y por eso el *bias* es negativo. Podemos ver el *overfitting* reflejado en el *variance* de un 25 % de la tabla 12 y la evolución del *validation categorical accuracy* en la figura 5 que empeora a lo largo del entrenamiento.

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	99.89	74.97	-4.89	24.91	578
Std	0.04	0.26	0.04	0.26	4.54

Tabla 12: Resultados del Experimento 1 de la Arquitectura 2 tras 5 entrenamientos

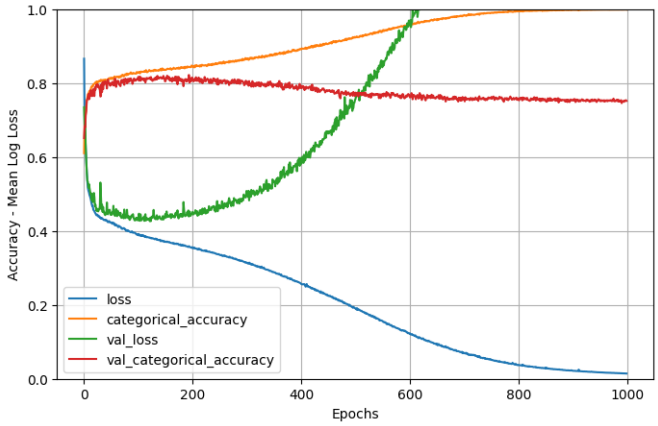


Figura 5: Entrenamiento durante el Experimento 1 de la Arquitectura 2

2.2.2. Experimento 2: Reducimos *epochs* y utilizamos regularización

Una de las técnicas que podemos utilizar para reducir el *variance* es usar regularización para controlar que los pesos de las neuronas no crezcan demasiado. Además para ahorrar algo de tiempo también reduciremos las epochs. La configuración para este experimento queda reflejada en la tabla 13.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.1	128	tanh	Categorical Crossentropy	SGD	**l2 0.001**

Tabla 13: Hiperparámetros para el Experimento 2 de la Arquitectura 2

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	81.19	79.91	13.8	1.28	249
Std	0.2	0.95	0.2	0.92	3.29

Tabla 14: Resultados del Experimento 2 de la Arquitectura 2 tras 5 entrenamientos

Como se observa en la tabla 14, hemos conseguido reducir el *variance* un 24 % simplemente usando regularización, ahora bien, en cuanto a *accuracy* seguimos sin conseguir una mejora significativa.

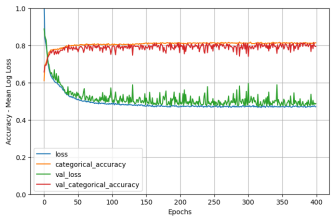


Figura 6: Entrenamiento durante el Experimento 2 de la Arquitectura 2

En la figura 6 se aprecia ruido en las métricas del conjunto de validación, esto es una consecuencia de la regularización, sin ella ese ruido se descontrolaría y llegaríamos al *overfitting* como en el experimento anterior.

2.2.3. Experimento 3: Cambiamos el optimizador

En este experimento vamos a comprobar si con un optimizador distinto podemos obtener mejor *accuracy* sin llegar al *overfitting*, para ello utilizaremos ADAM. La configuración se muestra en la tabla 15.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.1	128	tanh	Categorical Crossentropy	**ADAM**	l2 0.001

Tabla 15: Hiperparámetros para el Experimento 3 de la Arquitectura 2

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	25.34	26.25	69.65	-0.91	266.8
Std	0.29	2.61	0.29	2.44	4.81

Tabla 16: Resultados del Experimento 3 de la Arquitectura 2 tras 5 entrenamientos

Como vemos en la tabla resultados son los peores obtenidos hasta el momento y con mucha diferencia, apenas hemos conseguido un 25 % de *accuracy*. Analizando la figura 7, el modelo se queda estancado en un mínimo local muy temprano en el entrenamiento.

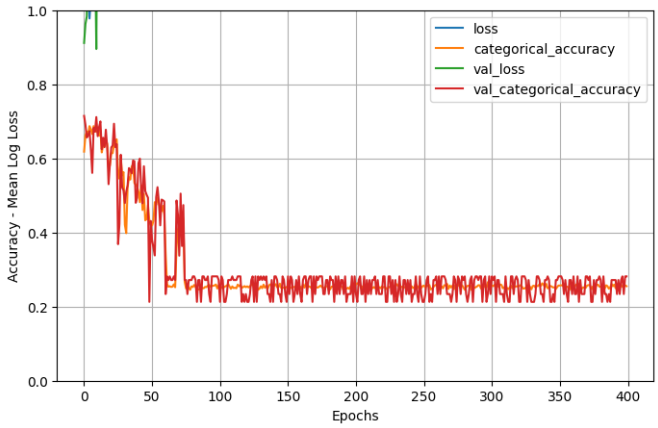


Figura 7: Entrenamiento durante el Experimento 4 de la Arquitectura 2

Observando la matriz de confusión en la figura 8 confirmamos nuestra teoría, clasifica todos los jugadores en la misma clase. Esto puede deberse a que no estamos usando correctamente el optimizador ADAM.

2.2.4. Experimento 4: Ajustamos el optimizador ADAM

Consultando el artículo en el que se presentó el optimizador ADAM (*Adam: A Method for Stochastic Optimization*, (2015), D. Kingma y J. Ba), vemos que este optimizador es una combinación entre AdaGrad y RMSProp y que tiene unos parámetros que controlan el ratio de decrecimiento de los momentos (beta1=0.9 y beta2=0.999) que no hemos definido en el experimento anterior. Además de esto, los autores sugieren usar un *learning rate* más bajo del que estamos usando.

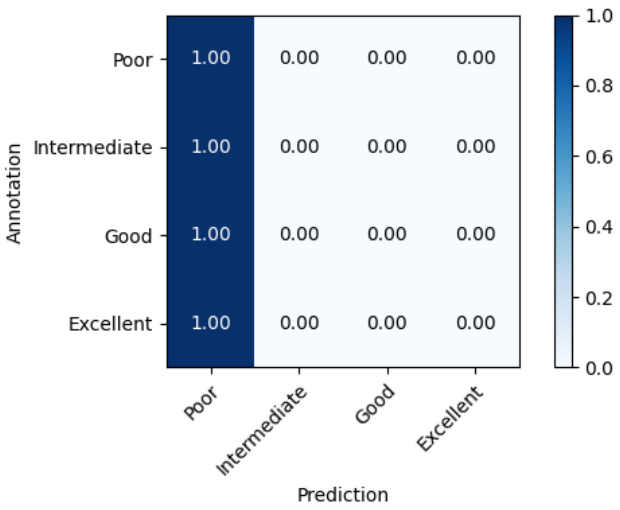


Figura 8: Matriz de confusión en el Experimento 3 de la Arquitectura 2

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	**0.001**	128	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 17: Hiperparámetros para el Experimento 4 de la Arquitectura 2

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	81.97	80.48	13.02	1.49	250.2
Std	0.15	0.15	0.15	0.19	19.35

Tabla 18: Resultados del Experimento 4 de la Arquitectura 2 tras 5 entrenamientos

Al haber configurado correctamente el optimizador volvemos a conseguir un *accuracy* similar al que hemos estado obteniendo. Observando la figura 9 vemos que podríamos seguir entrenando el modelo por más *epochs* antes de llegar al *overfitting*. Aun con todo esto, no hemos obtenido mejores resultados que con el optimizador *SGD*.

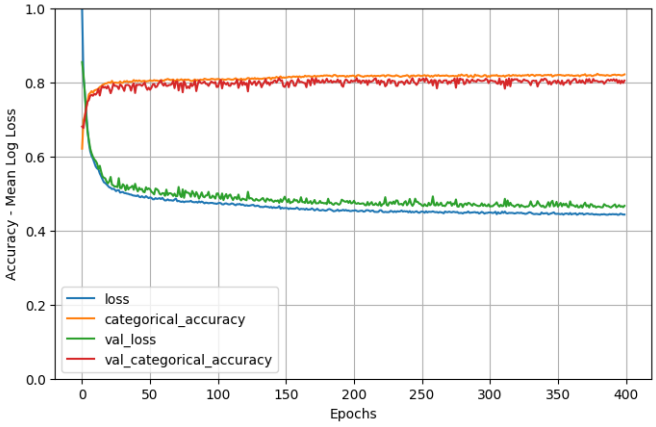


Figura 9: Entrenamiento durante el Experimento 4 de la Arquitectura 2

Analizando la matriz de confusión en la figura 10 vemos que ahora hemos solucionado el problema del modelo

anterior que clasificaba todos los jugadores en la misma clase

Figura 10: Matriz de confusión en el Experimento 4 de la Arquitectura 2

2.2.5. Experimento 5: Reducimos *batch-size*

Decidimos reducir el *batch size* para intentar reducir el *bias* como en el experimento 2.1.5. La configuración para este experimento es:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	**64**	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 19: Hiperparámetros para el Experimento 5 de la Arquitectura 2

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	81.87	80.03	13.12	1.83	506.4
Std	0.22	0.77	0.22	0.89	38.48

Tabla 20: Resultados del Experimento 5 de la Arquitectura 2 tras 5 entrenamientos

No hemos obtenido mejora, este cambio no merece la pena porque aumenta mucho el tiempo de entrenamiento. Como vemos en la figura 11, parece que el modelo deja de aprender, quizás por el problema del *vanishing gradient* que intentaremos resolver con la siguiente arquitectura.

2.2.6. Conclusiones de la Arquitectura 2

A lo largo de estos experimentos hemos concluido que:

- La regularización evita el *overfitting*.
- El optimizador ADAM necesita un *learning rate* bajo.
- El optimizador ADAM no ha proporcionado mejoras significativas con respecto al SGD para esta arquitectura.

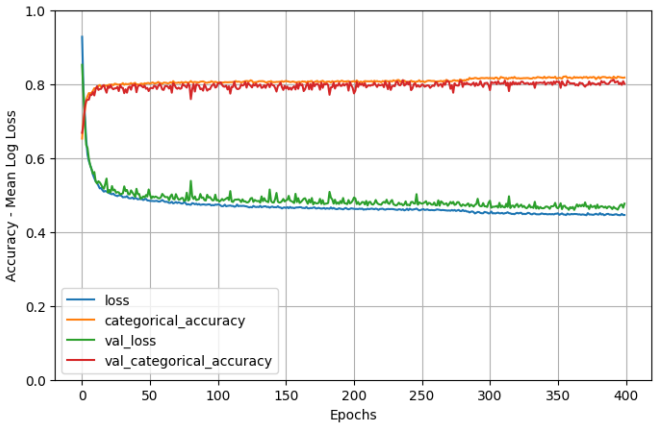


Figura 11: Entrenamiento durante el Experimento 5 de la Arquitectura 2

2.3. Arquitectura 3: Batch normalization

Después de no conseguir reducir el *bias* de nuestro modelo con más capas ocultas suponemos que estamos ante un problema de *vanishing gradient*, una de las soluciones que existe es utilizar capas de *batch normalization* para estandarizar los pesos de las neuronas. La arquitectura que utilizaremos durante los siguientes experimentos es:

1. Capa densa de 128 neuronas
2. Capa de *Batch Normalization* antes de la activación.
3. Capa densa de 64 neuronas
4. Capa de *Batch Normalization* antes de la activación.
5. Capa densa de 32 neuronas
6. Capa de *Batch Normalization* antes de la activación.

2.3.1. Experimento 1: Probamos la configuración del experimento 2.2.5

Para comprobar como afecta la introducción del *batch normalization*, utilizamos la configuración del último experimento realizado como vemos en la tabla 21.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	64	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 21: Hiperparámetros para el Experimento 1 de la Arquitectura 3

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	89.23	76.23	5.76	15.59	950.4
Std	0.46	1.54	0.46	1.75	55.11

Tabla 22: Resultados del Experimento 1 de la Arquitectura 3 tras 5 entrenamientos

La tabla 22 nos muestra que con *batch normalization* hemos obtenido una buena *accuracy* en el conjunto de entrenamiento pero a cambio de obtener un modelo con *overfitting* como nos indica el *variance* de 15 %. También ha sido la ejecución más larga dado el tamaño del batch y que hemos usado *batch normalization* que ralentiza el proceso de entrenamiento.

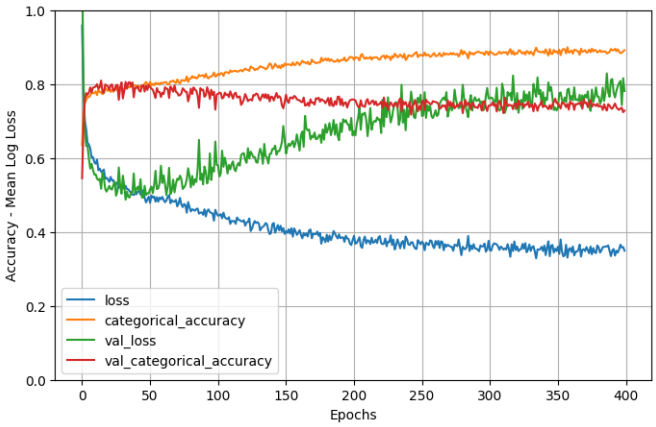


Figura 12: Entrenamiento durante el Experimento 1 de la Arquitectura 3

En la figura 12 vemos como el modelo llega al punto de *overfit* alrededor de la *epoch* 50 donde el *accuracy* de validación se separa del de entrenamiento.

2.3.2. Experimento 2: Aumentamos *batch size*

Aún no hemos conseguido mejorar el *accuracy* en el conjunto de validación, pero en este experimento vamos a intentar retrasar el *overfitting* del modelo a la vez que el tiempo de entrenamiento aumentando el *batch size*. La configuración para este experimento es la siguiente:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	**256**	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 23: Hiperparámetros para el Experimento 2 de la Arquitectura 3

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	97.02	73.89	-2.02	23.17	254
Std	0.53	0.48	0.53	0.61	14.04

Tabla 24: Resultados del Experimento 2 de la Arquitectura 3 tras 5 entrenamientos

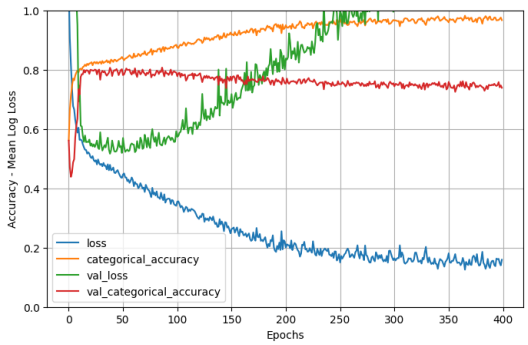


Figura 13: Entrenamiento durante el Experimento 2 de la Arquitectura 3

Lo único que hemos conseguido reducir con este experimento ha sido el tiempo de entrenamiento, el *variance* ha aumentado hasta un 23 %. En la figura 13 vemos reflejado el *overfitting*, lo que explica el *bias* negativo.

2.3.3. Experimento 3: Aumentamos la penalización del regularizador

El regularizador L2 añade una penalización en base al tamaño de los pesos, como no queremos que se descontrolen como ha estado ocurriendo, aumentamos esta penalización para evitar el overfitting. La configuración para este experimento es:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	tanh	Categorical Crossentropy	ADAM	**l2 0.1**

Tabla 25: Hiperparámetros para el Experimento 3 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	80.92	75.14	14.08	5.72	255.2
Std	0.51	2.41	0.51	2.27	9.25

Tabla 26: Resultados del Experimento 3 de la Arquitectura 3 tras 5 entrenamientos

En este experimento hemos conseguido reducir considerablemente el *variance* así que podemos concluir que el aumento de la penalización del regularizador sirve para evitar el *overfitting*. En la figura 14 observamos una evo-

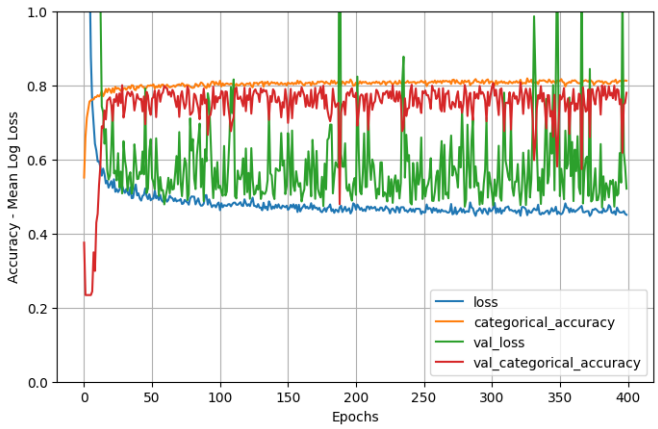


Figura 14: Entrenamiento durante el Experimento 3 de la Arquitectura 3

lución errática en el *loss* en validación lo que puede indicar la persistencia del problema del *vanishing/exploding gradient*.

2.3.4. Experimento 4: Volvemos a ReLU

Cambiamos la función de activación a ReLU para comprobar si ayuda con el problema del *vanishing/exploding gradient* como se aprecia en la tabla de configuración 27.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	**ReLU**	Categorical Crossentropy	ADAM	l2 0.1

Tabla 27: Hiperparámetros para el Experimento 4 de la Arquitectura 3

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	80.92	77.03	14.08	4.07	258.4
Std	0.46	1.8	0.46	1.56	7.46

Tabla 28: Resultados del Experimento 4 de la Arquitectura 3 tras 5 entrenamientos

Si nos fijamos en la tabla 28, vemos que hemos obtenido unos resultados similares al experimento anterior, además en la figura 15 se aprecia que no hay una fase de entrenamiento tan errática.

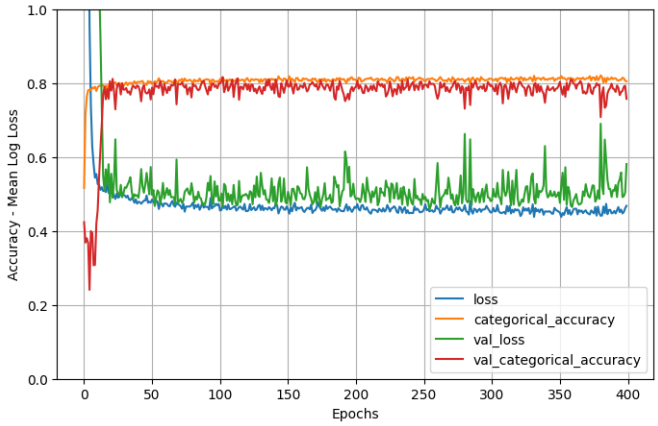


Figura 15: Entrenamiento durante el Experimento 4 de la Arquitectura 3

2.3.5. Experimento 5: Eliminamos la regularización

Nos preguntamos si tantos mecanismos contra el *vanishing gradient* y *overfitting* (*batch normalization*, ReLU, regularización L2) están interfiriendo entre ellos así que para comprobarlo retiramos la regularización L2 de las capas ocultas.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	**No**

Tabla 29: Hiperparámetros para el Experimento 5 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	97.31	74.55	-2.31	22.76	248
Std	0.44	0.99	0.44	0.68	6.12

Tabla 30: Resultados del Experimento 5 de la Arquitectura 3 tras 5 entrenamientos

El variance del 22 % que nos muestra la tabla 30 nos convence de que definitivamente la regularización es necesaria para evitar el *overfitting* que se observa en la figura 16.

2.3.6. Conclusiones de la Arquitectura 3

- La regularización L2 es necesaria para evitar el *overfitting*.
- A mayor penalización en L2 menor *overfitting*.
- ReLU ayuda a mitigar el *vanishing gradient*.
- Batch Normalization no ayuda a reducir el *bias*.

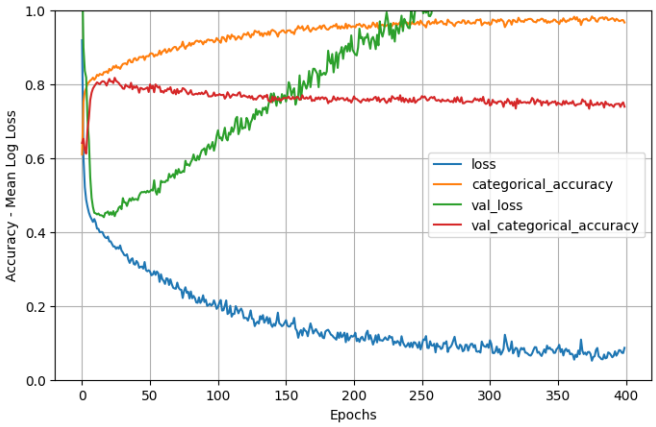


Figura 16: Entrenamiento durante el Experimento 5 de la Arquitectura 3

2.4. Arquitectura 4: Añadimos profundidad

Hasta el momento el máximo *accuracy* en validación que hemos conseguido ha sido en el experimento 2.1.2 con un 81.5 %, nos parece que es un resultado malo para un problema de juguete como es este. Tras haber intentado distintos métodos para reducir el *bias* sin aumentar el *variance* sin éxito decidimos aumentar la profundidad de la red a 7 capas con mayor complejidad:

1. Capa densa de 2048 con Batch Normalization.
2. Capa densa de 1048 con BN.
3. Capa densa de 512 con BN.
4. Capa densa de 256 con BN.
5. Capa densa de 128 con BN.
6. Capa densa de 64 con BN.
7. Capa densa de 32 con BN.

Cabe destacar que hemos estado repitiendo cada experimento 5 veces y no hemos encontrado diferencias significativas entre ellos. El objetivo de esta fase es comprender los hiperparámetros y analizar el impacto que tiene la variación de sus valores en el modelo y analizar su efecto en el modelo así que a partir de este punto solo realizaremos cada experimento una vez para probar más combinaciones.

2.4.1. Experimento 1: Repetimos la última configuración

Vamos a usar la configuración del experimento 2.3.5 para comprobar como afecta el cambio de arquitectura.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	No

Tabla 31: Hiperparámetros para el Experimento 1 de la Arquitectura 4

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
99.88	74.13	-4.88	25.87	426

Tabla 32: Resultados del Experimento 1 de la Arquitectura 4

Como vemos, hemos vuelto a obtener *overfitting*, dado que no hemos reintroducido la regularización desde el experimento 2.3.5.

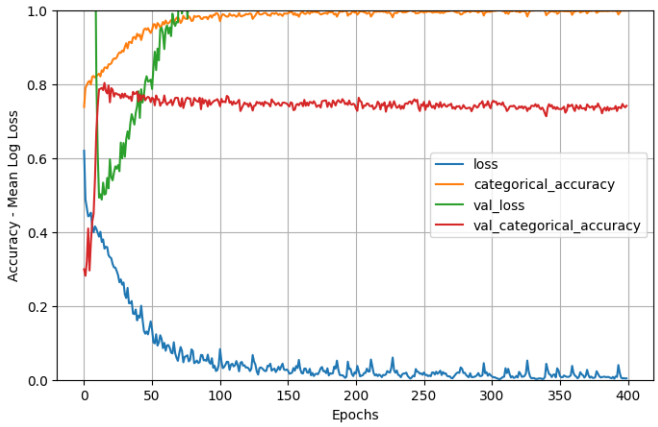


Figura 17: Entrenamiento durante el Experimento 1 de la Arquitectura 4

2.4.2. Experimento 2: Reintroducimos la regularización L2

Para evitar el *overfitting* volvemos a usar la regularización L2 con penalización 0.1

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	**l2 0.1**

Tabla 33: Hiperparámetros para el Experimento 2 de la Arquitectura 4

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.99	74.75	14.01	6.24	457

Tabla 34: Resultados del Experimento 2 de la Arquitectura 4

El *variance* de la tabla 34 nos confirma que hemos mitigado el *overfitting* pero en la figura 18 se ve un entrenamiento demasiado errático en cuanto al conjunto de validación, el *loss* varía enormemente en cada iteración, puede ser porque estemos estancados en un mínimo local del que el modelo está intentando salir. En

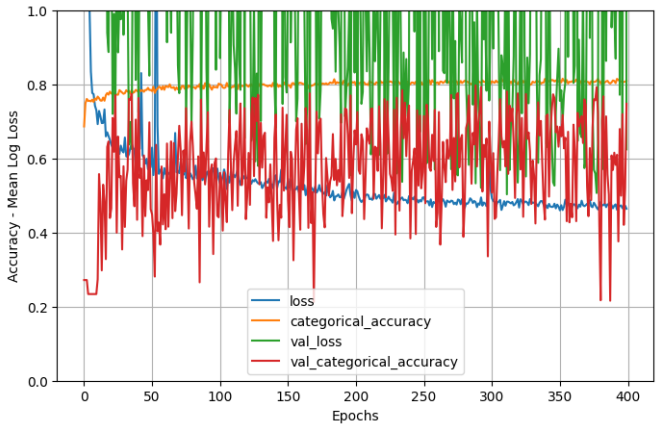


Figura 18: Entrenamiento durante el Experimento 2 de la Arquitectura 4

la matriz de confusión de la figura 19, hemos detectado que por primera vez las clases *Intermediate* y *Good* son las que mayor *recall* tienen lo que refuerza nuestra teoría del mínimo local.

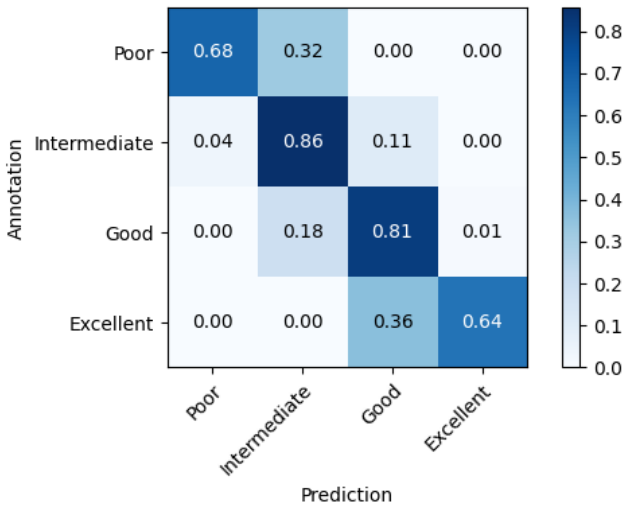


Figura 19: Matriz de confusión en el Experimento 2 de la Arquitectura 4

2.4.3. Experimento 3: Desactivamos el bias de las capas ocultas

En este momento reparamos en que estamos usando el vector *bias* de las capas ocultas, pero no es necesario tenerlo en cuenta porque al estar usando *Batch Normalization* ya estamos incluyendo un parámetro de *offset* β. Vamos a comprobar si mejora el proceso de entrenamiento.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	l2 0.1

Tabla 35: Hiperparámetros para el Experimento 3 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
81.16	75.81	13.84	5.35	422

Tabla 36: Resultados del Experimento 3 de la Arquitectura 4

Como vemos en la tabla 36, el cambio introducido no ha afectado al proceso de entrenamiento, simplemente hemos reducido el tiempo en 30 segundos.

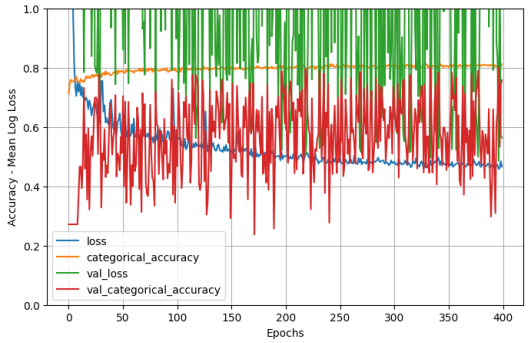


Figura 20: Entrenamiento durante el Experimento 3 de la Arquitectura 4

En la matriz de confusión comprobamos que la diagonal ha vuelto a los valores usuales.

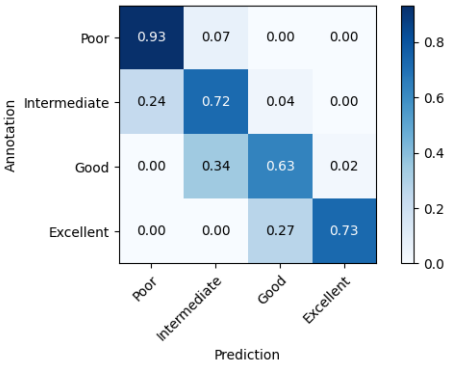


Figura 21: Matriz de confusión en el Experimento 3 de la Arquitectura 4

2.4.4. Experimento 4: Usamos inicializadores

Al estar usando una arquitectura profunda es importante que los pesos no se inicializen con valores demasiado altos ya que estos valores pueden crecer descontroladamente en las últimas capas, así que utilizaremos el initicializador *He Normal*. La configuración queda así:

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**	**Initializer**
400	0.001	256	ReLU	C.C.	ADAM	l2 0.1	**He Normal**

Tabla 37: Hiperparámetros para el Experimento 4 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
80.47	45.29	14.53	35.18	464

Tabla 38: Resultados del Experimento 4 de la Arquitectura 4

Los resultados han empeorado considerablemente como se observa en la tabla 38 y en la figura 22 donde se ve que la erraticidad sigue presente.

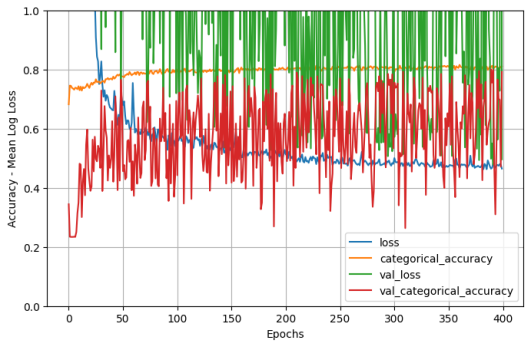


Figura 22: Entrenamiento durante el Experimento 4 de la Arquitectura 4

2.4.5. Experimento 5: Cambiamos el regularizador

Creemos que el problema del entrenamient errático está producido por el regularizador, hemos aumentado mucho la penalización y además de esto L2 añade penalización proporcional a la media del cuadrado de los pesos, estamos siendo demasiado agresivos. Pensamos que utilizando L1 y reduciendo λ conseguiremos solucionar nuestro problema.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	256	ReLU	C.C.	ADAM	**l1 0.0001**	He Normal

Tabla 39: Hiperparámetros para el Experimento 5 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
95.86	74.01	-0.86	21.85	455

Tabla 40: Resultados del Experimento 5 de la Arquitectura 4

Relajando la penalización en la regularización hemos vuelto al *overfitting* como se ve en la tabla 40, ya que tenemos un *variance* de 22 %. Lo bueno es que hemos reducido la variación en el entrenamiento como vemos en la figura 23.

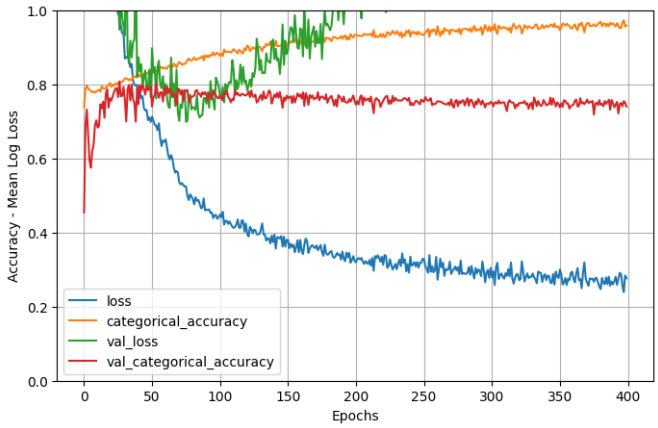


Figura 23: Entrenamiento durante el Experimento 5 de la Arquitectura 4

2.4.6. Experimento 6: Aumentamos *batch size*

Antes de pasar a la siguiente arquitectura decidimos aumentar el tamaño del *batch size* para probar si así reducimos algo el *overfitting*.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	**1024**	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 41: Hiperparámetros para el Experimento 6 de la Arquitectura 4

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
99.8	74.88	-4.8	24.92	455

Tabla 42: Resultados del Experimento 6 de la Arquitectura 4

Ha aumentado el *ovefitting* como nos indica el *variance* de la tabla 42, de hecho el *loss* del conjunto de validación no aparece en el gráfico 24 de evolución durante el entrenamiento.

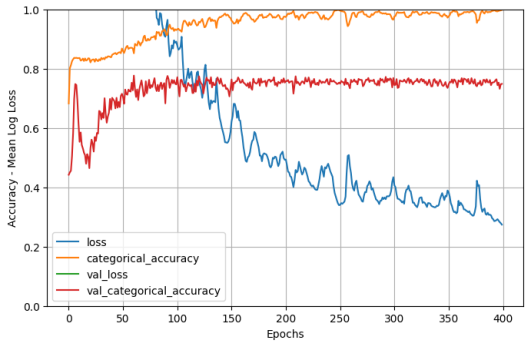


Figura 24: Entrenamiento durante el Experimento 6 de la Arquitectura 4

2.4.7. Experimento 7: Reducimos *batch size*

No estamos satisfechos con el último experimento que además nos sugiere que a mayor *batch size* mayor *bias* por lo que lo reducimos a un tamaño de 32.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	**32**	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 43: Hiperparámetros para el Experimento 7 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
78.14	77.79	16.85	0.35	3239

Tabla 44: Resultados del Experimento 7 de la Arquitectura 4

Como vemos, reducir drásticamente el *batch size* ha eliminado por completo el *overfitting* y de hecho el *accuracy* del conjunto de entrenamiento y validación han convergido.

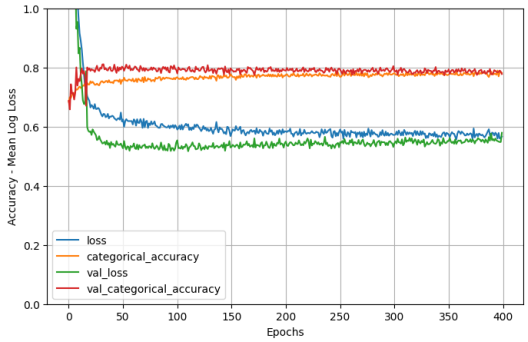


Figura 25: Entrenamiento durante el Experimento 7 de la Arquitectura 4

2.4.8. Conclusiones Arquitectura 4

- Sin regularización el *overfitting* persiste.

- A menor tamaño del *batch size* menor *overfitting*.
- La regularización L2 con un λ grande hace que el entrenamiento sea errático.
- Una arquitectura con mayor profundidad no ha producido una mejora en el *bias*.

2.5. Arquitectura 5: Arquitectura reloj de arena

Aunque no llega a ser un encoder-decoder, decidimos usar una arquitectura que imite este concepto. La arquitectura es la siguiente:

1. Capa densa de 512 neuronas con *Batch Normalization*
2. Capa densa de 128 neuronas con *BN*
3. Capa densa de 32 neuronas con *BN*
4. Capa densa de 128 neuronas con *BN*
5. Capa densa de 512 neuronas con *BN*

2.5.1. Experimento 1: Probamos con la misma configuración del experimento 2.4.7

Probamos con la configuración del experimento anterior

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	32	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 45: Hiperparámetros para el Experimento 1 de la Arquitectura 5

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.81	76.74	14.19	4.07	3414

Tabla 46: Resultados del Experimento 1 de la Arquitectura 5

Como vemos en 46, el *variance* ha aumentado un 4 % y en cuanto a *accuracy* hemos mejorado un 2 % en el conjunto de entrenamiento aunque empeorado mínimamente en cuanto al de validación. Es interesante el efecto que ha tenido esta arquitectura en el proceso de entrenamiento como se ve en la figura 26, los *accuracys* de los dos conjuntos se cruzan siendo el de validación el que comienza con un mejor valor.

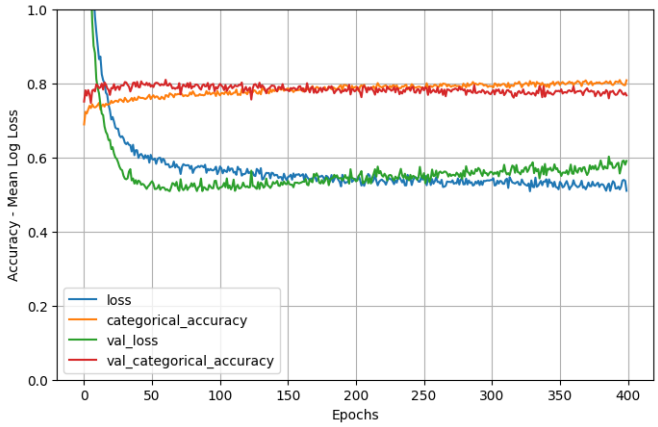


Figura 26: Entrenamiento durante el Experimento 1 de la Arquitectura 5

Esta arquitectura no ha mejorado el *bias* y en la matriz de confusión mostrada en la figura 27 tampoco apreciamos cambios, las clases *Intermediate* y *Good* siguen siendo las clases con peor *recall*.

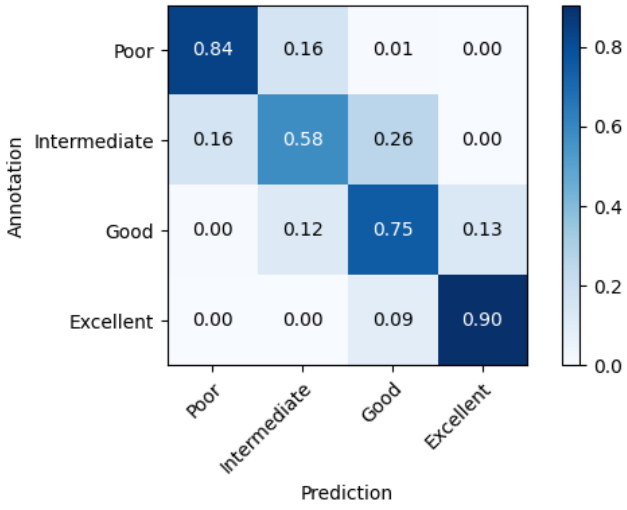


Figura 27: Matriz de confusión en el Experimento 1 de la Arquitectura 5

2.5.2. Experimento 2: Aumentamos *batch size* y *learning rate*

El problema de tener un *batch size* pequeño es que el tiempo de entrenamiento aumenta de forma drástica. En el paper *Understanding Batch Normalization (2018)* de J. Bjorck et al. se menciona que el optimizador está afectado por un ruido acotado superiormente por $\frac{lr}{bs}$, es decir por el *learning rate* dividido entre el *batch size*. Esto implica que aumentar el *batch size* tiene el mismo efecto que reducir el *learning rate*. Como actualmente tenemos un buen equilibrio entre estos dos hiperparámetros que no conduce al *overfitting* vamos a aumentar ambos para mantener esta proporción y además reducir el tiempo de entrenamiento.

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**	**Initializer**
400	**0.01**	**256**	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 47: Hiperparámetros para el Experimento 2 de la Arquitectura 5

Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
84.43	80.21	10.57	4.22	379

Tabla 48: Resultados del Experimento 2 de la Arquitectura 5

El efecto que hemos conseguido con este cambio ha sido muy satisfactorio, el tiempo de entrenamiento se ha reducido considerablemente y además hemos obtenido un mejor *accuracy*. La pega es que con más *epochs* llegaríamos a un *overfitting* como se aprecia en la figura 28.

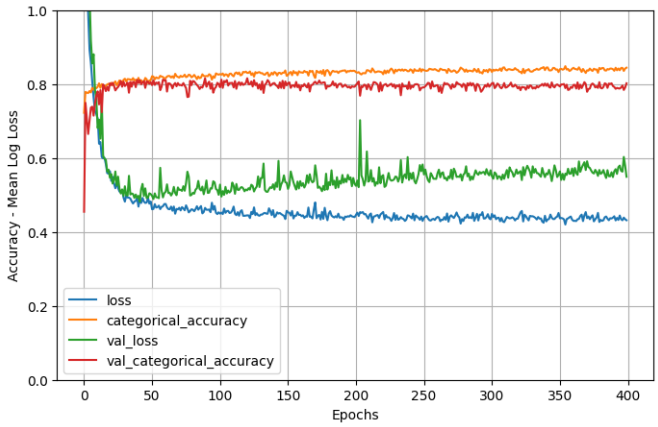


Figura 28: Entrenamiento durante el Experimento 2 de la Arquitectura 5

2.5.3. Experimento 3: Aumentamos *batch size* y *learning rate*

Para evitar el problema del *overfitting* a largo plazo que hemos detectado en el experimento 2.5.2 introducimos una técnica que hasta ahora no habíamos tenido en cuenta, el *dropout*.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer	Dropout
400	0.01	256	ReLU	C.C.	ADAM	l1 0.0001	He Normal	**0.1**

Tabla 49: Hiperparámetros para el Experimento 3 de la Arquitectura 5

Y obtenemos los siguientes resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.13	75.37	14.87	4.76	382

Tabla 50: Resultados del Experimento 3 de la Arquitectura 5

Hemos empeorado el *bias* aunque hemos mantenido el *variance* y como se ve en la figura 29 el entrenamiento se ha vuelto más errático aunque la divergencia que veíamos entre los *accuracy* de los conjuntos de entrenamiento y validación han desaparecido.

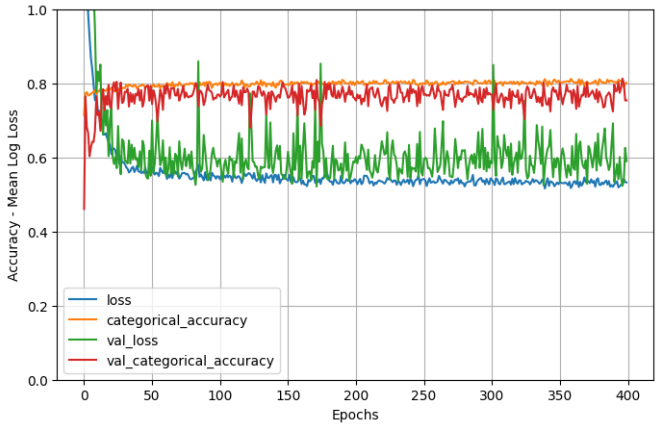


Figura 29: Entrenamiento durante el Experimento 3 de la Arquitectura 5

2.5.4. Experimento 4: Cambiamos la función de activación

Otra opción que no hemos probado hasta ahora es usar la función de activación *eLU*, así que en este experimento la usaremos con la esperanza de mejorar el *bias* del modelo.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer	Dropout
400	0.01	256	**eLU**	C.C.	ADAM	l1 0.0001	He Normal	0.1

Tabla 51: Hiperparámetros para el Experimento 4 de la Arquitectura 5

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.25	77.67	14.75	2.58	388

Tabla 52: Resultados del Experimento 4 de la Arquitectura 5

No hemos conseguido reducir el *bias* pero si el *variance* un 2 %. Como consecuencia de usar esta función de activación, el proceso de entrenamiento se ha visto afectado, la variación tanto en el *loss* como en el *accuracy* del conjunto de validación tiene una variación muy grande a lo largo del entrenamiento.

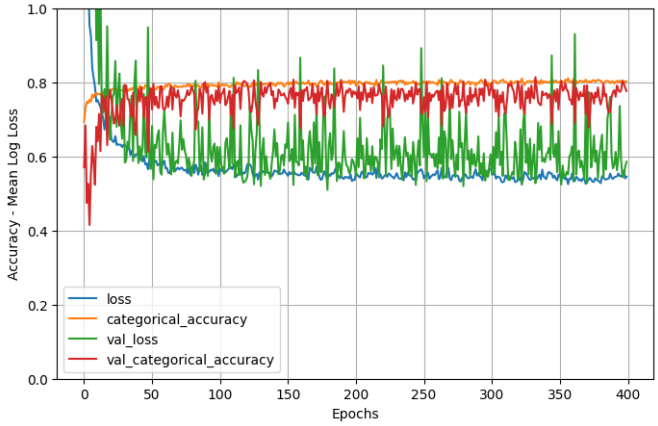


Figura 30: Entrenamiento durante el Experimento 4 de la Arquitectura 5

2.5.5. Conclusiones Arquitectura 5

- El *dropout* consigue mitigar el *overfitting*.
- La función de activación *eLU* consigue un menor *variance* en esta arquitectura.

3. Procesamiento de datos

En esta sección mostraremos cómo hemos manipulado los datos ya que en la siguiente fase introduciremos un nuevo conjunto de datos procesado para mejorar nuestros modelos.

3.1. Data imputation

Tomando como base el notebook de *PreparingFootballPlayerDataset* proprocinado, lo modificamos para seleccionar distintas *features* e imputar datos mediante **regresión lineal** para el campo de *Release Clause* e *imputación de medias* para 48 jugadores con una puntuación global de 82. Para más detalle comprobar el notebook *Data Imputation* que se ha proporcionado o se puede consultar en nuestro repositorio.
A continuación haremos dos pruebas para ver cómo afecta usar este nuevo conjunto en algunas de las arquitecturas anteriores.

3.2. Comparación de Arquitectura 2.1 con y sin data imputation

Usaremos la configuración del experimento 2.1.2 y comparamos resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Sin DI	84.02	81.5	10.97	2.47	199
Con DI	85.52	83.31	9.48	2.21	180

Tabla 53: Comparación de resultados del Experimento 2 de la Arquitectura 1

(a) Entrenamiento sin *data imputation* (b) Entrenamiento con *data imputation*

Figura 31: Comparación del entrenamiento

La figura 31 nos muestra la diferencia entre usar distintos datos en el entrenamiento. Con *data imputation* obtenemos un entrenamiento en el que la red aprende de manera más constante, además obtenemos mejores valores en todas las métricas.

3.3. Comparación de Arquitectura 2.2 con y sin data imputation

Usaremos la configuración del experimento 2.3.4 y comparamos resultados: La figura 32 nos muestra la

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Sin DI	80.92	77.03	14.08	4.07	258
Con DI	98.86	87.52	-3.86	11.34	272

Tabla 54: Comparación de resultados del Experimento 4 de la Arquitectura 3

(a) Entrenamiento sin *data imputation* (b) Entrenamiento con *data imputation*

Figura 32: Comparación del entrenamiento

diferencia entre usar distintos datos en el entrenamiento. Con *data imputation* obtenemos *overfitting*, aún así, el porcentaje de *accuracy* en validación es el mayor obtenido hasta el momento y podemos asegurar que el procesamiento de los datos tiene un gran peso a la hora de mejorar resultados.

4. Fase 2: Optimización sistemática

En esta fase iremos realizando cambios atómicos, es decir, para cada experimento nos centraremos en un hiperparámetro a optimizar, quedándonos con el modelo que tenga mejor *accuracy* en el conjunto de validación. Todos los experimentos a continuación han sido realizados 10 veces para obtener una representación robusta de los modelos usados.

4.1. Arquitectura 1: MLP4

Probaremos una arquitectura de complejidad media para buscar el overfitting y aplicaremos técnicas de regularización para reducir la varianza.

- Capa densa de 256 con Batch Normalization y Dropout = 0
- Capa densa de 124 con BN y Dropout = 0
- Capa densa de 64 con BN y Dropout = 0
- Capa densa de 32 con BN y Dropout = 0

4.1.1. Experimento 1: Primera configuración

Como se puede observar en la figura 33, tenemos un overfitting bastante grande y debemos aplicar técnicas de regularización para mejorar la generalización de la red.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
400	0.001	64	ReLU	C.C.	ADAM	No	0

Tabla 55: Hiperparámetros para el Experimento 1 de la Arquitectura 1

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	98.7	70.47	-3.7	28.23	817
Std	0.83	0.28	0.83	0.86	8.4

Tabla 56: Resultados del Experimento 1 de la Arquitectura 1 tras 10 entrenamientos

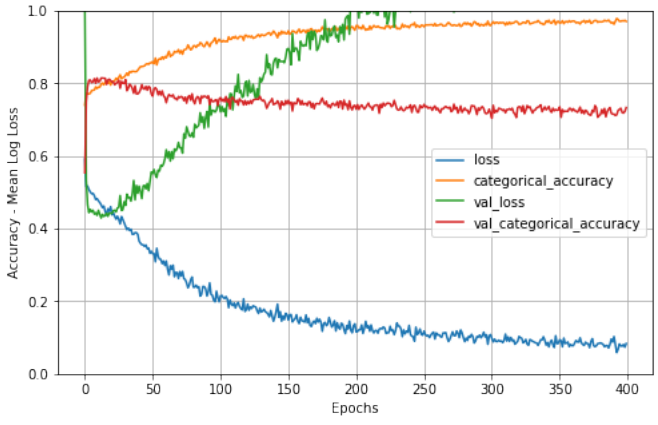


Figura 33: Entrenamiento durante el Experimento 1 de la Arquitectura 1

4.1.2. Experimento 2: Añadimos Dropout 0.2

Se añade un Dropout de 0.2 para reducir la varianza y mejorar la generalización. La configuración es la siguiente

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
400	0.001	64	ReLU	C.C.	ADAM	No	0.2

Tabla 57: Hiperparámetros para el Experimento 2 de la Arquitectura 1

Los resultados obtenidos son los siguientes:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	88.28	76.67	6.72	11.61	743
Std	1.15	0.79	1.15	1.89	8.9

Tabla 58: Resultados del Experimento 2 de la Arquitectura 1 tras 10 entrenamientos

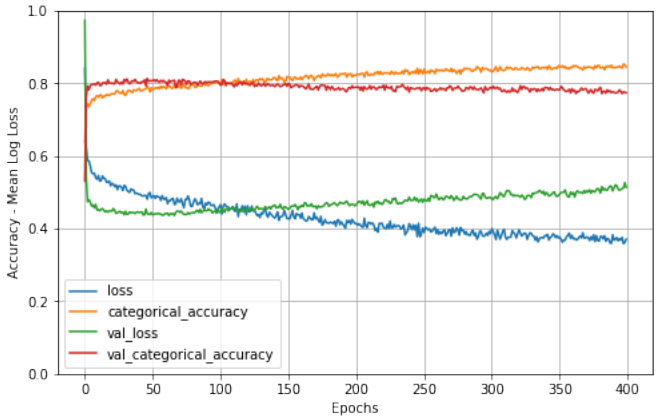


Figura 34: Entrenamiento durante el Experimento 2 de la Arquitectura 1

Se consigue una reducción del *variance* a cambio de un empeoramiento del *bias*.

4.1.3. Experimento 3: Regularizacón L1, L2 y L1-L2

Seguimos buscando la reducción de la varianza. Para ello probaremos los regularizadores L1, L2 y L1-L2.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
400	0.001	64	ReLU	C.C.	ADAM	?	0.2

Tabla 59: Hiperparámetros para el Experimento 3 de la Arquitectura 1

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
L1 0.1	63.6	68.98	31.4	-5.38	765
L2 0.1	73.06	76.86	21.94	-3.8	864
L1-L2 0.1	63.7	72.02	31.3	-8.32	804

Tabla 60: Resultados del Experimento 3 de la Arquitectura 1 tras 10 entrenamientos

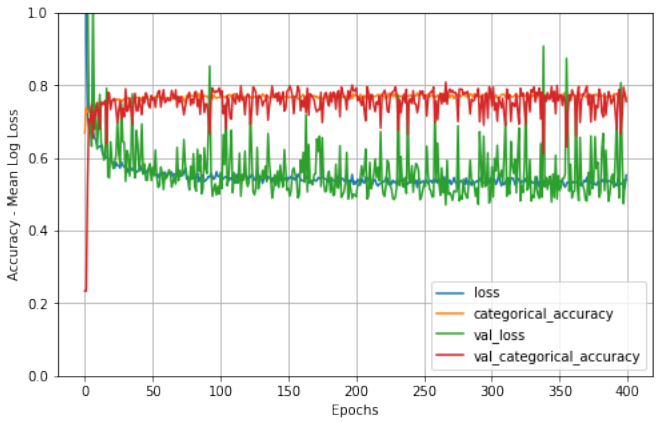


Figura 35: Entrenamiento durante el Experimento 3 (L2) de la Arquitectura 1

Como se puede observar en la tabla 57 la regularizacón L1 ha mejorado drasticamente la varianza y se ha conseguido una configuración que generaliza mejor. Sin embargo se ha aumentado demasiado el *bias*, es por ello que en los siguientes experimentos habrá que intentar reducirlo.

La regularizacón L2 tambien ha conseguido reducir considerablemente la varianza, pero tiene la ventaja de que hemos conseguido menor empeoramiento que con L1. Se concluye que en este caso es conveniente usar L2 frente a L1. En los siguientes experimentos habrá que intentar reducir el *bias*, las soluciones partirían de una reducción del tamaño del *batch* hasta una aumento de *epochs*.

La regularización L1-L2 también mejora drásticamente la varianza, pero empeora en exceso el Bias.

Se concluye que el mejor regularizador para nuestra configuración actual es el L1.

4.1.4. Experimento 4: Aumentar Epochs

En el experimento anterior se ha conseguido una mejora de la generalización de la red a cambio de un alto *bias*, es por ello que en este experimento se busca la reducción del *bias* utilizando más *epochs* de entrenamiento.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
?	0.001	64	ReLU	C.C.	ADAM	L2 0.1	0.2

Tabla 61: Hiperparámetros para el Experimento 4 de la Arquitectura 1

epochs	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
400	73.06	76.86	21.94	-3.8	864
800	73.52	77.98	21.48	-4.46	1585
1200	72.64	75.37	22.36	-2.73	1944

Tabla 62: Resultados del Experimento 4 de la Arquitectura 1 tras 10 entrenamientos

Duplicando o triplicando el número de epochs no se está logrando una mejora significativa de la varianza. Esto puede ser debido a que la red se está quedando atrapada en un mínimo local y hay que aplicar más reconfiguraciones para solventar este problema.

4.1.5. Experimento 5: Optimizadores

Se van a probar los diferentes optimizadores con los hiper parámetros por defecto para buscar una mejor aproximación e intentar salir en el mínimo local en que se encuentra la red.

Como se puede observar con el optimizador SGD se consigue una ligera mejora en los resultados. Sin embargo, creemos que se puede conseguir mejores resultados ya que el *bias* sigue siendo demasiado alto y la curva de aprendizaje es plana, lo que nos puede indicar que seguimos atrapados en un mínimo local.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
800	0.001	64	ReLU	C.C.	?	L2 0.1	0.2

Tabla 63: Hiperparámetros para el Experimento 5 de la Arquitectura 1

Batch Size	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Adam	73.52	77.98	21.48	-4.46	1585
RMSprop	71.39	79.53	23.61	-8.14	865
SGD	77.42	78.35	17.58	-0.93	683

Tabla 64: Resultados del Experimento 5 de la Arquitectura 1

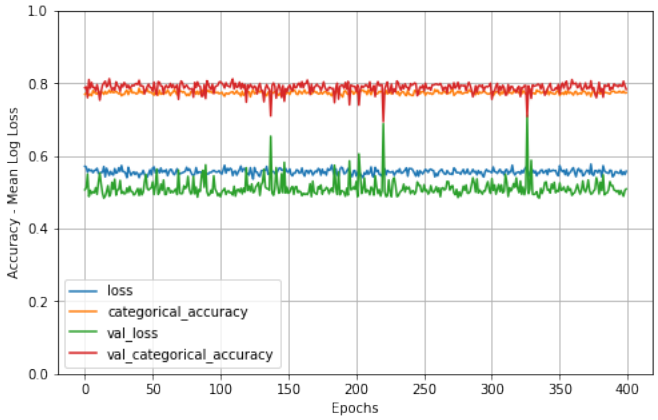


Figura 36: Entrenamiento durante el Experimento 5 (SGD) de la Arquitectura 1

4.1.6. Experimento 6: Variando el Batch size

Se sigue buscando una reducción del *bias*, para ello en este experimento se probarán diferentes tamaños de *batch* para comprobar si se consigue una mejora.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
800	0.001	?	ReLU	C.C.	SGD	L2 0.1	0.2

Tabla 65: Hiperparámetros para el Experimento 6 de la Arquitectura 1

Optimizer	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
64	77.42	78.35	17.58	-0.93	683
32	73.9	79.16	21.1	-5.26	863
128	81.24	79.53	13.76	1.71	443
256	84.74	78.47	10.26	6.27	383
1024	84.33	80.65	10.67	3.68	263
2048	75.45	79.34	19.55	-3.89	323

Tabla 66: Resultados del Experimento 6 de la Arquitectura 1 tras 10 entrenamientos

El tamaño del *batch* óptimo para la configuración actual es 1024. Obtenemos el mejor *validation accuracy* y *bias* de los últimos. Se concluye que al aumentar el tamaño del *batch* se consigue explorar más y salir del mínimo local en el que nos encontrábamos antes.

4.1.7. Experimento 7: Usando los nuevos datos

En este experimento se probará la mejor configuración con el nuevo dataset. Se observa que los resultados mejoran ligeramente, con lo cual se probará a aumentar los epochs para comprobar si mejora el rendimiento.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
800	1024	64	ReLU	C.C.	SGD	L2 0.1	0.2

Tabla 67: Hiperparámetros para el Experimento 7 de la Arquitectura 1

Epochs	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
800	81.51	82.45	13.49	-0.94	383
1600	85.88	85.85	9.12	0.03	785
3000	88.61	85.72	6.39	2.89	1643

Tabla 68: Resultados del Experimento 7 de la Arquitectura 1 tras 10 entrenamientos

El mejor modelo se obtiene con 1600 epochs, a partir de esos epochs sufrimos estancamiento en el validation accuracy, aunque se ha logrado una ligera mejora en el train accuracy.

4.2. Arquitectura 2: Buscando la mejor configuración

En esta sección buscaremos encontrar el mejor modelo, es decir, el que mejor *accuracy* obtiene, es por tanto que, todos los experimentos a continuación también han sido realizados 10 veces y calculando la media de estos. Además, para verificar que el cambio realizado es el responsable de la mejora del resultado los cambios han sido atómicos, es decir, uno a uno.

4.2.1. Ronda de experimentos 0

En este primer experimentos se ha usado la arquitectura base, únicamente el número de neuronas de cada capa ha sido puesto en múltiplos de dos. Por tanto la arquitectura usada ha sido un MLP-4 con [512,256,64,32].

Epochs	Learning rate	Batch size	Activation	Optimizer	Regularization	Initializer	Dropout
1000	0.1	512	ReLu	SGD	None	None	None

Tabla 69: Hiperparámetros para el Experimento 0 y de la Arquitectura 2

Tras realizar 10 veces el experimento con la anterior configuración, obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)
Mean	1.00	83.00
Best	1.00	83.31

Tabla 70: Resultados del Experimento 0 y de la Arquitectura 2

Como podemos comprobar, el primer problema con el que nos encontramos es el *overfitting*. Por lo tanto será esto lo primero que intentaremos solucionar en los siguientes experimentos.

4.2.2. Ronda de experimentos 1 - Añadimos Batch Normalization

Primer experimento buscando solucionar el *overfitting*, para ello en este introduciremos *Batch-Normalization* antes de la función de activación.

Tras realizar 10 veces el experimento con la anterior configuración, obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)
Mean	99.31	78.40
Best	98.99	79.71

Tabla 71: Resultados del Experimento 1 y de la Arquitectura 2

A la vista de los resultados, el *overfitting* persiste en el modelo por lo que habrá que seguir buscando la manera de solucionarlo.

4.2.3. Ronda de experimentos 2 - Añadimos Regularización L1-L2

En este segundo experimento buscando solucionar el *overfitting* introducimos la regularización L1-L2 con el objetivo de regularizar los pesos de nuestro modelo y que de esta forma el modelo no sobre aprenda. Es por tanto que en cada capa densa de nuestro modelo añadimos lo siguiente:

kernel_regularizer=regularizers.L1L2(l1=0.001,l2=0.1)

Tras realizar 10 veces el experimento con la anterior configuración, obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)
Mean	75.20	76.18
Best	78.10	75.40

Tabla 72: Resultados del Experimento 2 y de la Arquitectura 2

Como podemos ver, el overfitting se ha solucionado por lo que es el momento de mejorar ese accuracy.

Experimento 2.1 - Regularización L1-L2 sin *Batch-Normalization* Este experimento lo realizamos con el objetivo de comprobar si lo que ha solucionado el overfitting ha sido el *Batch-Normalization* junto a la regularización L1-L2 o la regularización por si sola sería capaz de solucionarlo.

Tras realizar 10 veces el experimento con la anterior configuración, obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)
Mean	47.64	49.97
Best	53.30	56.38

Tabla 73: Resultados del Experimento 2.1 y de la Arquitectura 2

A partir de los resultados de la Tabla 73 el modelo no aprende si introducimos la regularización sin usar el *Batch-Normalization*.

4.2.4. Ronda de experimentos 3 - Batch size

En esta ronda de experimentos el objetivo es encontrar el Batch size con el que mejor resultado obtenemos.

Epochs	Learning rate	Activation	Optimizer	Regularization	Initializer
1000	0.1	ReLu	SGD	L1L2	None

Tabla 74: Hiperparámetros para esta ronda de experimentos y usando la Arquitectura 2

Tras realizar 10 veces el experimento para cada Batch size con la anterior configuración, obtenemos los siguientes resultados:

A raíz de los anteriores resultados se ha tomado la decisión de utilizar 4096 como valor para el *batch size*, ya que este es el que obtiene los mejores resultados en el conjunto de validación obteniendo un 79.91 (%)

Batch size	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
64	70.81	75.27	70.94	76.55
128	72.78	77.01	72.81	78.66
256	74.17	77.51	74.37	79.21
512	75.20	76.18	75.40	78.10
1024	77.00	79.85	77.29	81.01
2048	79.71	78.13	78.33	81.27
4096	**79.73**	**79.91**	**80.45**	**81.14**
8192	79.97	76.40	79.93	79.71

Tabla 75: Resultados de la ronda de experimentos 3 y usando la Arquitectura 2

4.2.5. Ronda de experimentos 4 - Arquitectura

Una vez definido el tamaño de *batch*, el siguiente paso será encontrar la mejor arquitectura. En la Tabla 76 veremos los hiperparámetros que serán utilizados para todas las arquitecturas evaluadas.

Epochs	Learning rate	Batch size	Activation	Optimizer	Regularization	Initializer
1000	0.1	4096	ReLu	SGD	L1L2	None

Tabla 76: Hiperparámetros para esta ronda de experimentos y usando la arquitectura 2

Tras realizar 10 veces el experimento para cada arquitectura con la anterior configuración, obtenemos los siguientes resultados:

Arquitectura	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
[8,4]	79.21	80.59	79.81	81.38
[32,16]	80.39	80.78	81.12	81.38
[256,128]	80.04	80.73	79.93	81.69
[512,256]	79.96	80.57	80.09	81.07
[16,8,4]	79.51	81.19	79.73	82.19
[32,16,8]	**80.05**	**81.27**	**79.67**	**82.32**
[64,32,16]	79.99	80.76	79.98	81.38
[256,64,32]	79.84	80.62	80.25	81.51
[512,256,128]	80.08	78.66	79.81	80.70
[64,32,16,8]	79.45	79.91	79.83	80.95
[256,64,32,16]	79.60	80.13	79.02	81.01
[1024,512,256,64]	80.15	79.80	81.17	80.83
[512, 256, 64, 32]	79.73	79.91	80.45	81.14
[2048,1024,512,256]	80.02	79.72	79.80	80.83
[32,16,16,8,8]	79.33	80.08	78.75	80.89
[32,32,16,16,8]	79.36	79.98	78.78	80.83
[1024,512,256,128,64]	80.52	78.61	81.78	79.83

Tabla 77: Resultados de la ronda de experimentos 4 usando la arquitectura de la primera columna

Como podemos observar, en este caso estructura con la que mejor *accuracy* en el conjunto de validación obtenemos es una MLP3 con [32,16,8] neuronas por capa respectivamente.

4.2.6. Ronda de experimentos 5 - Learning Rate

En esta ronda de experimentos nuestro objetivo es encontrar el mejor valor para el *learning rate*.

Arquitectura	Epochs	Batch size	Activation	Optimizer	Regularization	Initializer
[32,16,8]	1000	4096	ReLu	SGD	L1L2	None

Tabla 78: Hiperparámetros para esta ronda de experimentos y usando la Arquitectura 2

Tras realizar 10 veces el experimento para cada valor de *learning rate* con la anterior configuración, obtenemos los siguientes resultados:

Learning Rate Fijo	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
0.1	80.32	81.13	79.96	81.63
0.01	83.17	83.07	83.12	83.43
0.001	69.33	69.03	72.94	73.51
0.0001	58.81	56.79	61.14	59.11

Tabla 79: Resultados de la ronda de experimentos 5

En este caso, nos hemos dado cuenta que cuando el *learning rate* = 0.001 el mejor resultado es obtenido en la epoch 1000, lo que nos indica que si continuaramos entrenando este podría mejorar. Por lo que el próximo experimento lo haremos con la configuración siguiente:

Arquitectura	Epochs	Batch size	Activation	Optimizer	Regularization	Initializer
[32,16,8]	5000	4096	ReLu	SGD	L1L2	None

Tabla 80: Hiperparámetros para esta ronda de experimentos.

A partir de los hiperparámetros anteriores obtenemos el siguiente resultado:

Learning Rate Fijo	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
0.001	85.10	83.53	85.05	84.18

Tabla 81: Resultado para un learning rate = 0.001 con 5000 epochs.

Como podemos comprobar en la Tabla 81, este valor de *learning rate* es el mejor que hemos encontrado, y por lo tanto, el que utilizaremos para los siguientes experimentos.

4.2.7. Ronda de experimentos 5.1 - Reducing Learning Rate

Antes de continuar con otros hiperparámetros nos planteamos el uso del callback ReduceLROnPlateau en nuestro modelo para comprobar si con este obtenemos un mejor resultado que usando un learning rate fijo tal y como se ha empleando en el experimento anterior. Por tanto el callback utilizado es el siguiente: ReduceLROnPlateau('val_categorical_accuracy', factor=0.1, patience=300, verbose=1) Los settings para este experimento se pueden ver en la Tabla 80.

Como podemos ver en la Tabla 82 no conseguimos superar los resultados obtenidos con el learning rate fijo a 0.001 durante todo el entrenamiento.

4.2.8. Ronda de experimentos 6 - Función de Activación

En esta ronda de experimentos nuestro objetivo es el de encontrar la mejor función de activación para nuestro modelo. En la Tabla 83 podemos ver los hiperparámetros utilizados para estos experimentos.

En la Tabla 84 podemos ver los resutados de los experimentos realizados donde como observamos, el mejor resultado es obtenido con ReLu, seguido por la Elu.

Learning Rate Inicial	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
1	63.71	59.46	67.32	68.73
0.1	80.82	81.64	81.17	82.28
0.01	62.81	62.63	78.64	79.21
0.001	84.48	83.38	84.22	83.31

Tabla 82: Resultado para un learning rate = 0.001 con 5000 epochs.

Arquitectura	**Epochs**	**Batch size**	**Learning Rate**	**Optimizer**	**Regularization**	**Initializer**
[32,16,8]	5000	4096	0.001	SGD	L1L2	None

Tabla 83: Hiperparámetros para esta ronda de experimentos.

Función de activación	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
ReLu	**85.10**	**83.53**	**85.05**	**84.18**
Elu	82.45	83.36	82.52	83.68
Tanh	82.46	83.01	82.67	83.68

Tabla 84: Resultados para cada función de activación.

Debido a los resultados de estos experimentos, nos quedaremos con la ReLu como función de activación para el resto de experimentos.

4.2.9. Ronda de experimentos 7 - Inicializador

En esta nueva ronda de experimentos, exploraremos el campo de los inicializadores de pesos. En la Tabla 85 podemos ver los hiperparámetros para estos experimentos.

Arquitechture	**Epochs**	**Batch size**	**LR**	**Opt.**	**Activation**	**Regularization**	**Initializer**
[32,16,8]	5000	4096	0.001	SGD	ReLu	L1L2	None

Tabla 85: Hiperparámetros para esta ronda de experimentos.

En la Tabla 86 vemos como el mejor inicializador es el He Uniform, ya que recordamos que el resultado que nos interesa en el *accuracy* en el conjunto de validación y este es el que mayor porcentaje de acierto obtiene.

Inicializador	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
Ninguno	85.10	83.53	85.05	84.18
He Normal	84.87	83.66	84.70	84.36
He Uniform	**84.83**	**83.73**	**84.94**	**84.24**
Uniform (-0.1,0.1)	84.11	83.34	84.78	83.93

Tabla 86: Resultados para cada inicializador.

Al igual que en el resto de experimentos, nos quedamos con el mejor inicializador para los siguientes, y por lo tanto nuestro inicializador a partir de ahora será el He Uniform.

4.2.10. Ronda de experimentos 8 - Optimizador

En esta ronda de experimentos buscaremos el mejor optimizador para nuestro modelo. En la Tabla 87 podemos ver los hiperparámetros que emplearemos para los experimentos.

Tras haber realizado las diez inicializaciones para cada optimizador, en la Tabla 88 podemos comprobar que el mejor es el SGD por delante de otros como Adam o RMSProp.

Arquitechture	Epochs	Batch size	LR	Activation	Regularization	Initializer
[32,16,8]	5000	4096	0.001	ReLu	L1L2	He Uniform

Tabla 87: Hiperparámetros para esta ronda de experimentos.

Optimizador	Train accuracy (%) Mean	Validation accuracy (%) Mean	Train accuracy (%) Best	Validation accuracy (%) Best
SGD	**84.83**	**83.73**	**84.94**	**84.24**
SGD + Momentum	86.79	83.64	86.80	84.36
RMSProp	86.16	83.07	86.63	83.56
Adam	88.33	81.76	88.57	82.38
Nadam	88.37	82.33	88.50	83.00

Tabla 88: Resultados para cada Optimizador

4.3. Conclusiones

Como conclusiones podriamos decir, que sin realizar data imputation en los datos de entrenamiento, la mejor configuración es la que podemos ver en la Tabla 90.

Arquitechture	Epochs	Batch size	Optimizer	LR	Activation	Regularization	Initializer
[32,16,8]	5000	4096	SGD	0.001	ReLu	L1L2	He Uniform

Tabla 89: Hiperparámetros definitivos.

La cual obtiene un accuracy en el conjunto entrenamiento del 84.83 % y en el de validación de 83.73 %.

5. Resultados finales

Utilizaremos la mejor configuración encontrada en la Fase 2 que determinamos en la sección 4.3.
Además de usar ReduceOnPlateau usaremos el *callback* de ModelCheckpoint para guardar el mejor modelo

Arquitechture	Epochs	Batch size	Optimizer	LR	Activation	Regularization	Initializer
[32,16,8]	5000	4096	SGD	0.001	ReLu	L1L2	He Uniform

Tabla 90: Hiperparámetros definitivos.

encontrado durante el entrenamiento.
En primer lugar, presentamos en la tabla 91 los resultados obtenidos. Se ve que el *dataset* con datos introducidos

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Sin DI	83.11	81.33	11.89	1.78	463
Con DI	92.17	91.53	2.73	0.74	484

Tabla 91: Comparación de resultados de la mejor arquitectura encontrada

tiene un gran impacto en los resultados, ambos *accuracys* han incrementado un 10 % y hemos obtenido al mismo tiempo unos *bias* y *variance* bajos, cosa que hasta ahora no había ocurrido.

Podemos ver la comparativa del proceso de entrenamiento en la figura 37. Se aprecia que con *data imputation*,

(a) Entrenamiento sin *data imputation*

(b) Entrenamiento con *data imputation*

Figura 37: Comparación del entrenamiento

el modelo comienza con un *accuracy* muy bajo y tarda unas 1500 *epochs* en alcanzar su punto óptimo mientras que en el caso (a) llega a ese punto antes de las 1000 *epochs*. Además de esto podemos comprobar que el modelo generaliza bien mirando las matrices de confusión en la figura 38 donde vemos que el recall del modelo con *data imputation* es mucho mejor.

(a) Entrenamiento sin *data imputation*

(b) Entrenamiento con *data imputation*

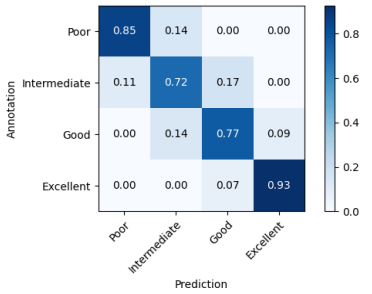
Figura 38: Comparación de las matrices de confusión

Finalmente usamos el conjunto de test para evaluar nuestro modelo. Claramente la tabla 92 nos muestra

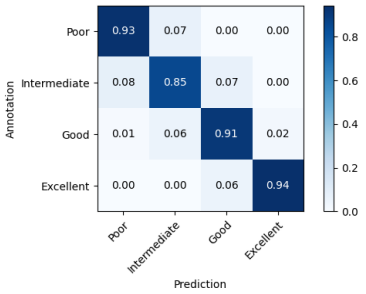
	Train accuracy (%)
Sin DI	81.32
Con DI	90.85

Tabla 92: Comparación de resultados de la mejor arquitectura encontrada en el conjunto de test

de nuevo la gran diferencia entre usar datos imputados y no hacerlo. Lo bueno es que es ningún caso hay una diferencia extrema entre los resultados obtenidos en el conjunto de validación. En la figura 39 podemos ver que tampoco hay gran diferencia entre el conjunto de test y el de validación en cuanto a *recall*.



(a) Entrenamiento sin *data imputation*



(b) Entrenamiento con *data imputation*

Figura 39: Comparación de las matrices de confusión para el conjunto de test

6. Conclusiones

Finalizando esta práctica hemos concluido que:

- Encontrar el equilibrio entre los hiperparámetros es un proceso empírico que requiere conocer los problemas y soluciones que pueden surgir con las redes neuronales.
- El procesamiento de datos es tan o más importante que la optimización de los hiperparámetros. Tener unos buenos datos de entrenamiento implica tener unos mejores resultados y modelos como hemos podido comprobar.