UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"ANÁLISIS COMPARATIVO DE ESTRATEGIAS PARA LA REDUCCIÓN DEL SOBREAJUSTE EN APLICACIONES BASADAS EN DEEP LEARNING."

TRABAJO FIN DE GRADO

Septiembre

2023

AUTOR: Alejandro Llopis Llorens

DIRECTOR: Ramon Pedro Ñeco García

Índice:

CAPÍTULO 1.

INTRODUCCIÓN

1. DESCRIPCIÓN DEL PROYECTO

Este proyecto pretende ser un análisis comparativo de las diferentes técnicas y estratégias más utilizadas para evitar el "overfitting" o sobreajuste en aplicaciones fundamentadas en "deep learning" o aprendizaje profundo. Por lo tanto, a lo largo del documento se tratarán temas tales como el aprendizaje profundo, las redes neuronales convolucionales, el sobreajuste, los conjuntos de datos o las diferentes técnicas para evitar el sobreajuste.

El sobreajuste se define como un fenómeno donde la red neuronal convolucional se ajusta excesivamente a los datos de entrenamiento, aprendiendo también del ruido y otras variaciones de forma que no se puede generalizar correctamente a nuevos datos.

Quizás el párrafo anterior no resulte completamente claro, así que quisiera explicar lo más importante mediante un ejemplo. Imaginemos que la red neuronal es como un estudiante que aprende a partir de ejemplos. Le damos ejemplos de datos para que los estudie y aprenda cómo funcionan. Sin embargo, cuando hay pocos ejemplos o la red es muy complicada, la situación se complica. En lugar de entender realmente los ejemplos, la red parece memorizarlos. Esto se convierte en un problema cuando enfrenta nuevos ejemplos que no ha visto antes. La red no sabe cómo lidiar con ellos y comete errores. Esto ocurre porque la red convolucional memoriza los ejemplos de entrenamiento y no es capaz de "entender" o generalizar lo aprendido.

Este desafío, el sobreajuste, es recurrente en las aplicaciones de inteligencia artificial que emplean el aprendizaje profundo. Por esta razón, se han desarrollado a lo largo del tiempo multitud de técnicas para abordar este problema. En este proyecto, se han seleccionado seis de las técnicas más comunes y recomendadas, las cuales se han aplicado en dos redes neuronales convolucionales representativas. La primera red, de pequeña dimensión, se utiliza con el conjunto de datos MNIST. La segunda, de tamaño moderado, se emplea con el conjunto de datos "dogs-vs-cats". Además, se suele aplicar más de una técnica a la vez, por ello también se analizará y comparará las principales uniones de técnicas.

2. OBJETIVOS

1. Entender y describir los diferentes conceptos relativos a la inteligencia artificial y el "deep learning".
2. Comprender y explicar el concepto de entrenamiento y sobreajuste.
3. Identificar y comprender las principales técnicas contra el sobreajuste.
4. Analizar y comparar las técnicas mencionadas en el punto anterior
5. Diseñar y analizar diferentes estrategias con las que combatir el sobreajuste.
6. Conocer e implementar las redes neuronales convolucionales.
7. Identificar la mejor técnica, estrategia o variante para redes sencillas y redes medianas.

3. DESCRIPCIÓN DE LA MEMORIA

Capítulo dos. Conceptos previos. Redes neuronales y sobreajuste

El segundo capítulo tiene como propósito introducir el tema del proyecto mientras explica los diversos conceptos que resultan indispensables para entender el contenido del documento. Entre estos conceptos se incluyen, por ejemplo, "deep learning", aprendizaje supervisado, red neuronal convolucional, capa convolucional y entrenamiento.

Capítulo tres. Técnicas y estrategias para evitar y reducir el sobreajuste

En este capítulo se pretende identificar y explicar las diferentes técnicas y estrategias empleadas contra el sobreajuste. Las técnicas son las siguientes:

1. "Early stopping" o parada temprana.
2. "Dropout" o abandono.
3. Tamaño de la red.
4. Activación del clasificador.
5. Aumento de datos
6. Regularización L1 y L2

Capítulo cuatro. *Red neuronal de "MNIST".*

En este capítulo se aborda la implementación de las diferentes técnicas y estrategias en las

redes neuronales de pequeñas dimensiones y su análisis. El capítulo se divide en cuatro grandes bloques.

El primero de los bloques se enfoca en la creación de la red neuronal convolucional, la descarga del conjunto de datos "MNIST" y su procesamiento. El segundo bloque aborda la implementación y el análisis de las diferentes técnicas, requiriendo la recopilación y comprensión de los resultados obtenidos tras la aplicación de estas técnicas. El tercer bloque, similar al segundo, se centra en la implementación y análisis de las estrategias. Finalmente, el cuarto bloque compara los resultados logrados en los dos bloques previos.

Capítulo cinco. Red neuronal de "dogs-vs-cats".

La estructura de este capítulo es la misma que la del anterior. Este capítulo pretende la implementación y el análisis de las diferentes técnicas y estrategias en las redes neuronales de tamaño mediano.

CAPÍTULO 2.

CONCEPTOS PREVIOS. REDES NEURONALES Y SOBREAJUSTE

Antes de combatir el sobreajuste es necesario conocer diferentes conceptos previos, como por ejemplo la propia definición de este, qué es la inteligencia artificial o qué componen las redes neuronales.

1. INTELIGENCIA ARTIFICIAL, MACHINE LEARNING, DEEP LEARNING Y REDES NEURONALES ARTIFICIALES

Quisiera empezar por el principio, en la introducción se ha mencionado diferentes ejemplos de inteligencia artificial y se ha insinuado qué es la inteligencia artificial.

1.1. DEFINICIÓN DE INTELIGENCIA ARTIFICIAL

La Real Academia Española de la Lengua define inteligencia artificial como “disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico” (ASALE & RAE, s. f.-c). Obviamente, esta no es una definición válida para este documento, pues trata la inteligencia artificial como una ciencia y no como el programa en sí.

Podemos encontrar en el libro Deep Learning : Introducción práctica con Kerás del profesor Jordi Torres una definición simple pero más próxima a nuestro propósito, “aquella inteligencia que muestran las máquinas, en contraste con la inteligencia natural de los humanos” complementada con la definición “el esfuerzo para automatizar tareas intelectuales normalmente realizadas por humanos” («Deep Learning – Introducción Práctica Con Keras», s. f.). Así pues, una inteligencia artificial es un sistema o programa informático capaz de realizar tareas intelectuales, tales como el aprendizaje, la percepción, el razonamiento, el reconocimiento de patrones, la planificación o la resolución de problemas.

Los sistemas de inteligencia artificial se suele clasificar en dos categorías principales:

- Inteligencia artificial estrecha (narrow AI): Sistemas diseñados para llevar a cabo tareas específicas y limitadas. Estos sistemas pueden ser altamente especializados y eficientes

en su área de enfoque, pero carecen de la capacidad de generalizar o aplicar su conocimiento en otras áreas. Este es el tipo de inteligencia artificial que se analizará en este proyecto.

- Inteligencia artificial general (artificial general intelligence, AGI): Sistemas que pretenden poseer una capacidad cognitiva similar o incluso superior a la humana, lo que le permitiría comprender y resolver una amplia gama de tareas, adaptarse a nuevas situaciones y aprender de manera autónoma. Un ejemplo actual es Chat GPT.

1.2. DEFINICIÓN DE "MACHINE LEARNING"

En la definición de inteligencia artificial he indicado algunos ejemplos de las tareas que pretenden realizar, entre ellos el aprendizaje. Existen muchas formas de que una inteligencia artificial aprenda, pero estos métodos se pueden dividir en dos categorías: automáticos y no automáticos. "Machine learning" es el nombre que recibe el aprendizaje automático y la rama de la inteligencia artificial que lo estudia en inglés.

"Machine learning" se define como la rama de la inteligencia artificial que se enfoca en el desarrollo de algoritmos y técnicas que permiten a máquinas aprender de los datos y mejorar su rendimiento en tareas específicas sin ser programadas exclusivamente para ello. Esto significa que los diferentes algoritmos, en lugar de seguir instrucciones directas, utilizan datos históricos para identificar patrones, tendencias y relaciones en los datos para, tras generalizar, tomar decisiones o hacer predicciones sobre nuevos datos nunca antes vistos.

Hay diferentes enfoques que se utilizan en el aprendizaje automático que se agrupan, normalmente, en tres conjuntos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

El primero, aprendizaje supervisado, se refiere a la utilización de conjuntos de datos que incluyen las soluciones deseadas, denominadas "etiquetas" o "labels". Este proceso sirve para entrenar algoritmos como por ejemplo la regresión lineal, la regresión logística, los árboles de decisión o las redes neuronales. El término "aprendizaje no supervisado" se refiere al uso de conjuntos que no incluyen las etiquetas siendo el propio algoritmo quién intentará clasificar la información. Los principales algoritmos que utilizan este tipo de aprendizaje son: Principal Component Analysis (PCA) y clustering (K-means). Por último, el aprendizaje por refuerzo requiere que el modelo aprenda mediante prueba y error, recibiendo recompensas y

penalizaciones tras sus acciones. Este aprendizaje permite la combinación tanto con los dos anteriores como con aprendizajes no automáticos.

1.3. DEFINICIÓN DE DEEP LEARNING

El deep learning o aprendizaje profundo es una subclase del aprendizaje automático con características únicas y grandes resultados, "un caso especial de algoritmos de Machine Learning" («Deep Learning – Introducción Práctica Con Keras», s. f.). IBM, una de las principales empresas del sector define así el aprendizaje profundo: "Deep learning es un subconjunto de machine learning, que es básicamente una red neuronal con tres o más capas." (*¿Qué es Deep Learning?*, s. f.). Sin embargo, estas definiciones están un poco cojas pues no explican el modo de aprendizaje el cual podemos entender cómo: "En lugar de organizar datos para que se ejecuten a través de ecuaciones predefinidas, el deep learning configura parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento." (*¿Qué es deep learning?*, s. f.). El punto más importante de la cita anterior es el reconocimiento de patrones. El aprendizaje profundo se basa en reconocer y aplicar patrones a muy diferentes niveles.

1.4. DEFINICIÓN DE RED NEURONAL ARTIFICIAL

Normalmente no se diferencian las redes neuronales del aprendizaje profundo ya que son inseparables. Mientras que el deep learning es un método de aprendizaje, las redes neuronales son quienes usan el método. Se define red neuronal artificial como un modelo matemático y computacional inspirado en la estructura y funcionamiento del cerebro humano.

La red neuronal está compuesta por un conjunto de unidades interconectadas, también conocidas como neuronas artificiales o nodos. Cada neurona recibe una información de entrada, la procesa realizando una operación matemática determinada y envía una salida, normalmente el resultado de la operación. Otro aspecto importante de la red neuronal artificial es su estructura, los nodos se organizan en capas.

Pero ¿qué sentido tiene separar en capas las neuronas? Por un lado, para repartir las tareas y funciones. Hay tres tipos de capas: capa de entrada, capas ocultas y capa o capas de salida. La capa de entrada o "input layer" recibe los datos de entrada y los transmite a la siguiente capa. Las capas ocultas o "hidden layers", también conocidas como capas intermedias son las encargadas de procesar y transformar la información de entrada para extraer características y

patrones complejos de los datos. La capa o capas de salida o "output layer" produce el resultado final de la red neuronal, produce la tarea específica que el modelo debe realizar.

Por otro lado, se separan las neuronas en capas para poder transmitir correctamente la información. Las conexiones entre neuronas son siempre hacia la siguiente capa, no hay comunicación entre neuronas de la misma capa. Además, estas uniones están asociadas a pesos que determinan la influencia que cada neurona tiene sobre las demás. Durante el entrenamiento de la red, estos pesos se ajustan y actualizan iterativamente para minimizar el error entre las predicciones del modelo y las salidas reales.

2. LAS REDES NEURONALES CONVOLUCIONALES

Este proyecto se centra principalmente en las redes neuronales convolucionales o "ConvNet" y su principal problema, el sobreajuste, sobreentrenamiento u "overfitting".

2.1. RED NEURONAL CONVOLUCIONAL

2.1.1. DEFINICIÓN

Una red neuronal convolucional es una clase de red neuronal artificial especializada en procesar datos que tienen una estructura cuadriculada, como imágenes y videos. Indicamos anteriormente que las redes neuronales procesan la información en las capas aplicando una operación matemática, la operación principal realizada en estas redes concretas se denomina convolución. Así pues, red neuronal convolucional se define como categoría de red neuronal cuya operación matemática característica es la convolución.

2.1.2. FUNCIONAMIENTO GENERAL

Con objeto de mostrar el funcionamiento general de las redes neuronales, se puede equiparar este a el reconocimiento de una cara. Sabemos que las inteligencias artificiales pretenden imitar a los seres humanos, pero ¿cómo lo hacen los humanos? ¿Cómo reconocemos los humanos las caras? Según el profesor Jordi Torres:

> "... si vemos una cara, la reconocemos porque tiene orejas, ojos, una nariz, cabello, etc. Entonces, para decidir si algo es una cara, lo hacemos como si tuviéramos unas casillas mentales de verificación de las características que vamos marcando. Algunas veces una cara puede no tener una oreja por estar tapada por el pelo, pero igualmente lo

> clasificamos con una cierta probabilidad como cara porque vemos los ojos, la nariz y la boca. ” («Deep Learning – Introducción Práctica Con Keras», s. f.)

Pero cada una de esas partes de la cara las identificamos a su vez desde secciones más pequeñas, como líneas, texturas o formas. Así funcionan las redes convolucionales, las primeras capas aprenden y encuentran los diferentes bordes, las siguientes reconocen ya combinaciones de bordes y las últimas ya reconocen modelos de objetos.

2.1.3. VENTAJAS Y FUNCIONES DE LAS REDES CONVOLUCIONALES

Dado que las capas tienen funciones diferentes, las redes neuronales incluyen en su aprendizaje representaciones jerárquicas y abstractas de las imágenes. Además, estas redes neuronales son capaces de manejar eficientemente grandes conjuntos de datos en relativo poco tiempo a cambio de cierta potencia computacional. Esta gran adaptabilidad les permite abordar tareas de visión por computadora de alta complejidad, como el reconocimiento de objetos, la detección de objetos o la segmentación de imágenes. Estas tareas son fundamentales para una amplia gama de sistemas y aplicaciones de tecnología de vanguardia, como sistemas de reconocimiento facial para la autenticación biométrica, vehículos autónomos para la percepción del entorno y toma de decisiones, y diagnósticos médicos asistidos por imagen para la detección temprana y precisa de enfermedades.

2.2. COMPONENTES BÁSICOS DE UNA RED NEURONAL CONVOLUCIONAL

Pese a que las redes neuronales convolucionales se describen como redes neuronales cuya operación principal es la convolución, estas redes no están compuestas únicamente por capas convolucionales, también hay capas de pooling, una capa flatten y capas densamente conectadas clasificadoras.

2.2.1. CAPAS CONVOLUCIONALES

Las capas convolucionales son aquellas capas que permiten a la red aprender de forma local, pequeños patrones en ventanas de dos dimensiones. El propósito principal de estas capas es detectar características, patrones o rasgos visuales en las imágenes que se encuentren en pequeñas ventanas. Por tanto, las capas convolucionales tienen las siguientes características principales:

1. Los patrones que aprenden son invariantes a traslación. Una vez aprendido un patrón concreto en una posición concreta, la ConvNet podrá reconocerlo en cualquier otro lugar.
2. Aprenden jerarquías especiales de patrones. La primera capa aprenderá pequeños patrones locales (bordes, esquinas). La segunda capa convolucional aprenderá patrones más grandes que están formados por los patrones y características de las primeras capas. Finalmente aprenderá patrones globales.

La entrada y salida de las capas convoluciones están compuestas por un vector de rango 3 (altura, anchura, canales). El número de canales se determina con un argumento (filters) que suelen ser 32, 64, 128 o 256, excepto en la primera entrada que viene determinada por el número de canales. Las imágenes de color RGB tienen una dimensión del eje de canal tres, dado que estas imágenes tienen tres colores (red, green and blue). Al contrario, en una imagen en blanco y negro la dimensión es 1.

Operación de convolución:

Siempre estamos hablando de que las neuronas realizan una operación, en concreto la convolución. Pero ¿qué es la convolución? ¿Cuáles son los operandos de la operación?

La convolución es una operación que extrae pequeñas ventanas de la entrada y le aplica la misma máscara de transformaciones a todos estos conjuntos. El tamaño de la máscara, y por tanto de las ventanas de datos suelen ser 3x3 o 5x5, sobre todo 3x3. La convolución es la multiplicación de la máscara por la submatriz (ventana) de la entrada sobre la que hace efecto. La máscara se desplaza por toda la entrada de forma que el resultado tendrá la forma del tensor de entrada, pero con los ejes espaciales ligeramente más pequeños. Se puede visualizar correctamente en la ilustración 1.

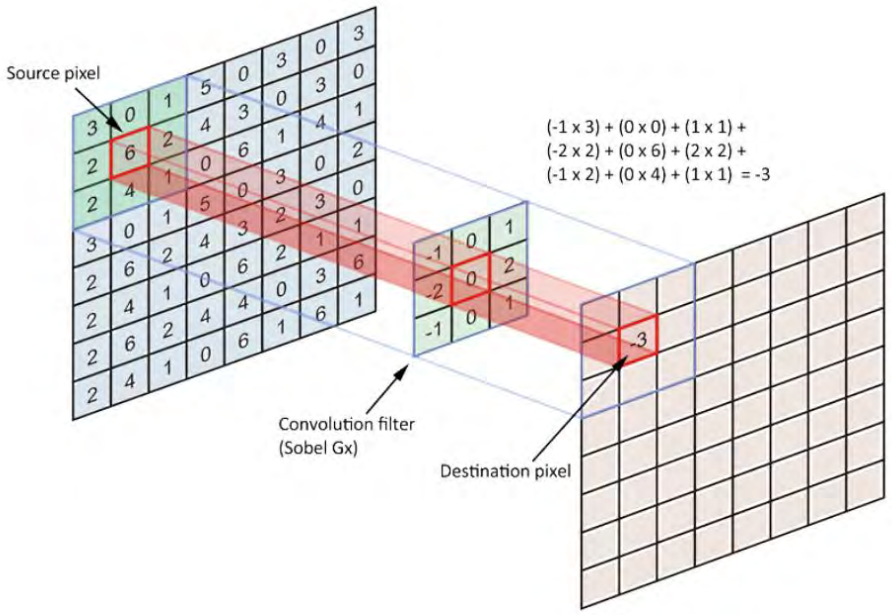

Ilustración 1. (https://keepcoding.io/blog/que-es-la-convolucion/, 2022)

2.2.2. CAPAS POOLING. OPERACIÓN DE POOLING

Una de las capas más habituales son aquellas que se ocupan de realizar la operación pooling, las cuales suelen situarse inmediatamente detrás de una capa convolucional. El objetivo principal de la capa es reducir el tamaño del mapa de características, simplificando y condensando la información.

La operación de pooling o agrupación consiste en aplicar una función de reducción en regiones locales, es decir escoger un valor representativo de la región escogida. Las regiones se definen generalmente mediante ventanas de tamaño fijo y pequeño. En estas capas no tiene sentido que las ventanas se solapen de forma que el salto entre ellas será igual a su tamaño, normalmente 2x2. Las dos funciones de reducción más utilizadas son:

- Max Pooling: Para cada ventana, se toma el valor máximo de los elementos incluidos. Esto resalta la característica más relevante en la región y descarta el resto.
- Average Pooling: Para cada ventana, se calcula el promedio de los elementos incluidos. Esto suaviza la información y ayuda a reducir el ruido en los datos.

La operación de agrupación tiene varias ventajas: Reduce la cantidad de datos, mejora la tolerancia a las traslaciones y ayuda a evitar el sobreajuste. Al seleccionar solo un valor representativo para cada ventana, se reduce la cantidad de datos en las capas subsiguientes, lo

que conduce a una reducción en la carga computacional, en el sobreajuste y en la cantidad de parámetros en la red. Además, al reducir la resolución de los mapas de características la red se vuelve más resistente a las variaciones de la posición de los rasgos y patrones.

Se podría pensar que hay una gran pérdida de información, pero hay que tener en cuenta que la mayoría de ésta es intrascendente, por ejemplo, al disminuir el tamaño de bordes, el tamaño del fondo, etc. Por otro lado, es necesario reducir el tamaño de imágenes para conseguir la jerarquía de patrones.

2.2.3. CAPA FLATTEN

Esta capa tiene una única función, aplanar el vector que recibe como entrada. Esto significa convertir el rango de los datos, de 3 a 1. Esto significa finalizar la red y proceder de manera secuencial.

2.2.4. PILAS DE CAPAS DENSAMENTE POBLADAS

Este tipo de capa tiene como objetivo funcionar como clasificador final. Así pues, estas capas cumplen un papel esencial en la última etapa del procesamiento de la información en una red neuronal convolucional. Se diferencia de las capas anteriores principalmente en dos características, la conexión entre neuronas y la operación que realiza la capa.

En una capa densamente poblada, cada neurona está conectada a todas las neuronas de la capa anterior y posterior. Es decir, cada neurona recibe como entrada la salida de todas las neuronas de la capa anterior. Esto significa que todas las características extraídas en las capas anteriores son combinadas y ponderadas por cada neurona en la pila de capas densamente pobladas.

La operación que realiza una capa densamente poblada es una transformación lineal, seguida de una función de activación no lineal. Matemáticamente, la operación se puede describir como:

$$y_1 = f_{activación}(y_0 * m + n)$$

Donde:

- y_0: Es la entrada a la neurona. Viene dado como características de la capa anterior.
- y_1: Es la salida de la neurona.
- $f_{activación}$:Es una función de activación no lineal.

- m: Es el peso específico de esa conexión. Estos pesos son los parámetros que se aprenden durante el entrenamiento y determinan la influencia de cada entrada.
- n: Denominado habitualmente bias, es una constante que se suma completando la transformación lineal.

La función de activación se escoge adecuándose a la tarea específica que la red debe resolver.

2.3. EL ENTRENAMIENTO Y EL SOBREAJUSTE

2.3.1. CONCEPTO DE ENTRENAMIENTO

La Real Academia Española de la Lengua define entrenamiento como "acción y efecto de entrenar o entrenarse" (ASALE & RAE, s. f.-a) y entrenar como "Preparar o adiestrar personas o animales, especialmente para la práctica de un deporte" (ASALE & RAE, s. f.-b). Uniendo ambas definiciones podemos inferir que el entrenamiento es la acción y efecto de preparar o adiestrar personas o animales, especialmente para la práctica de un deporte. La Real Academia Española de la Lengua no menciona en ningún momento la inteligencia artificial o las máquinas. Por tanto, no nos sirve su definición. En este sentido una definición mejor sería "el procedimiento utilizado para llevar a cabo el proceso de aprendizaje en una red neuronal" (Entrenamiento de Redes Neuronales: mejorando el Gradiente Descendente - Fernando Sancho Caparrini, s. f.).

2.3.2 ENTRENAMIENTO DE UNA RED NEURONAL CONVOLUCIONAL

A lo largo de este capítulo se ha comentado varias veces el aprendizaje, pero ¿qué es lo que aprende una red convolucional? La red aprende los valores de los pesos (m) y los sesgos (n) con intención de optimizarlos para la resolución de la tarea asignada. Este proceso implementa un algoritmo iterativo de ida y vuelta de la información. Dado que las redes convoluciones realizan un aprendizaje supervisado el entrenamiento se aprovechará tanto las imágenes en la ida como las etiquetas en la vuelta.

La ida es la primera fase que se aplica. Esta también se conoce como "forward propagation", propagación hacia delante en inglés, ocurre cuando los datos de entrenamiento se introducen en la red. En esencia, los datos de entrada se canalizan a través de la red de manera que cada neurona aplique su transformación a la información recibida de las neuronas de la capa anterior. Finalmente, la capa de salida proporciona una predicción de las etiquetas asociadas a cada

imagen de entrada. La segunda fase consiste en medir cuán mala es la estimación, comparándola con el resultado correcto. Esta comparación permite estimar una función de error, o "loss" en inglés. La tercera y última fase es la vuelta, denominada retropropagación, en inglés "back propagation". Consiste en propagar la información de error partiendo de la capa de salida. Sin embargo, las neuronas sólo reciben de la señal la parte proporcional de cada neurona a la salida original.

Idealmente la diferencia entre la predicción y la etiqueta sea cero. Por ello, a medida que el entrenamiento progresa se ajustan los pesos y sesgos. Existen diferentes técnicas como Descenso de gradiente, Momentum, Adagrad u Optimización Bayesana. La primera, el descenso de gradiente es el método fundamental y más utilizado. Esta técnica varía los pesos en pequeños incrementos gracias al cálculo del gradiente (o derivada) de la función de error. El gradiente permite seguir la dirección correcta hacia el mínimo global.

El algoritmo iterativo se puede resumir en:

- Paso 0: Asignar valores aleatorios a los diferentes parámetros de la red.
- Paso 1: Aplicar forward propagation, es decir, introducir el conjunto de ejemplos como datos de entrenamiento para obtener una predicción.
- Paso 2: Calcular la función de error.
- Paso 3: Realizar el back propagation, propagando la función a los parámetros del modelo.
- Paso 4: Aplicar la técnica del descenso de gradiente para ajustar los parámetros.
- Paso 5: Continuar con el paso 1, hasta conseguir un buen modelo o las veces que se haya determinado.

2.3.3. EL SOBREAJUSTE

En el punto anterior se ha comentado que la función de error se reduce a cero. Pero eso es lo ideal, existen muchos problemas y dificultades que afectan al aprendizaje y al entrenamiento. Una de las situaciones más habituales al entrenar un modelo de clasificación de imágenes es que la cantidad de datos sea escasa. Esa limitación produce el problema denominado sobreajuste.

"El 'overfitting' o sobreajuste es un concepto en la ciencia de datos qué ocurre cuando un modelo estadístico se ajusta exactamente a sus datos de entrenamiento." (*¿Qué es el*

sobreajuste? | *IBM*, s. f.) explica IBM. En lo relativo al aprendizaje automático, el sobreajuste se puede definir como un fenómeno donde el modelo se ajusta excesivamente a los datos de entrenamiento, aprendiendo también del ruido y otras variaciones de forma que no se puede generalizar correctamente a nuevos datos.

Existen diferentes motivos por los que se puede producir el sobreajuste. Los dos principales son la falta de datos de entrada y una excesiva complejidad del modelo. Sin embargo, ¿cómo saber si se produce el sobreentrenamiento?

Si sólo se realiza el entrenamiento, iterando una y otra vez los mismos datos, no se puede saber si se produce o no sobreentrenamiento hasta utilizarlo pudiendo ser erróneo. Para comprobar la aptitud del modelo se aplica la llamada validación cruzada. La validación cruzada consiste en dividir el conjunto de entrenamiento en al menos dos partes, la parte de entrenamiento más grande, aproximadamente un 80% de los datos originales, y la parte de validación más pequeña, aproximadamente un 20%. Se utilizan los datos de entrenamiento primero y se evalúa este entrenamiento al final de cada iteración o época aplicando el conjunto de validación. Finalmente, se comparan los datos recogidos del entrenamiento con los recogidos de la validación tras múltiples épocas. Las métricas a analizar en la validación cruzada son la precisión y el error del modelo, los cuales varían según el momento del entrenamiento. Ambos valores se basan en la respuesta de la red; mientras que la precisión indica el porcentaje de conjeturas correctas, el error mide cuán errónea es la predicción.

Con objeto de facilitar el entendimiento del análisis, en este proyecto se utilizarán gráficas representativas y tablas demostrativas. Un ejemplo de tabla demostrativa sería:

Modelo	**Precisión de entrenamiento**	**Precisión de validación**	**Época de sobreajuste**	**Sobreajuste de precisión**
Original	100%	90%	< 5	10%
Técnica a evaluar	97.62%	95%	>25	2.62%

Tabla ejemplo.

Para este ejemplo se ha elegido una tabla de precisión, siendo las de error muy similares. Tanto el valor de entrenamiento como de validación seleccionados son los óptimos; en el caso de la

precisión, el máximo y en el caso del error, el mínimo. La "época de sobreajuste" indica a partir de qué momento se produce el sobreentrenamiento, dado que cada vez que se realiza el entrenamiento los datos varían ligeramente el valor indicado es orientativo. Finalmente, el sobreajuste indicado no es más que la diferencia entre los valores indicados en la tabla de entrenamiento y validación.

Un ejemplo de gráficas representativas sería:

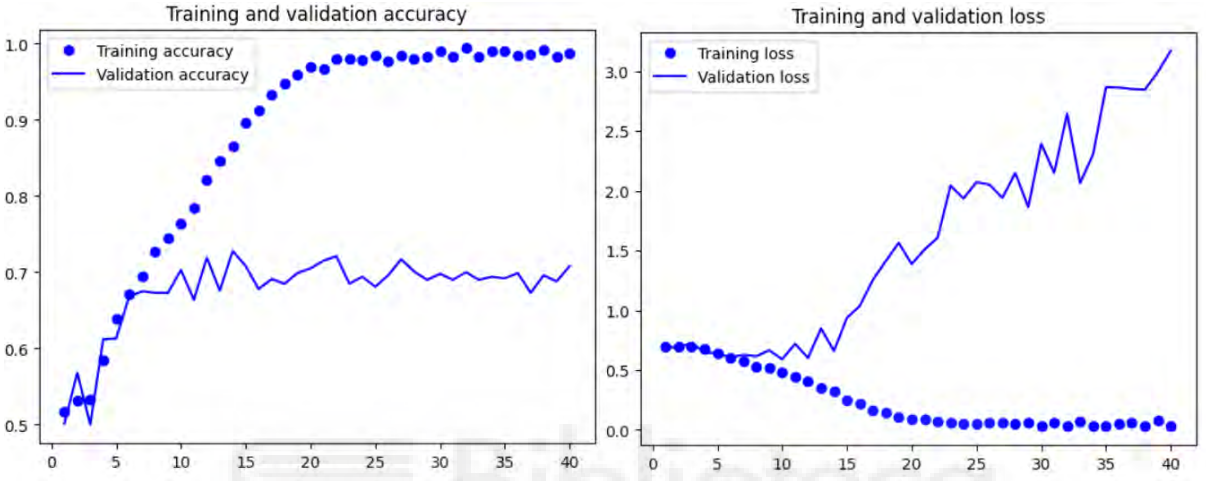

Imagen x. División en figuras x1 y x2.

La imagen x es representativa del sobreajuste. En la figura x1 de la izquierda se representan los resultados de precisión tanto de entrenamiento (puntos) como de validación (línea). Todos los datos se dibujan respecto a las épocas. A partir de la sexta época se produce el sobreajuste, fácilmente observable pues ambas series divergen apasionadamente. De manera similar se representa en la figura x2 las funciones de error, tanto la de validación (línea) como la de entrenamiento (puntos).

CAPÍTULO 3.

TÉCNICAS Y ESTRATEGIAS PARA EVITAR Y REDUCIR EL SOBREAJUSTE

Una vez presentados los conceptos fundamentales como son el aprendizaje profundo, las redes neuronales convolucionales y el problema del sobreajuste, procedemos a exponer las técnicas para resolver este inconveniente. Estas técnicas se basan fundamentalmente en aumentar el número de datos de entrada y reduciendo la complejidad o el tamaño de la red, entre otras. Las técnicas que se expondrán no tienen por qué funcionar en todas las redes neuronales, por ello en próximos capítulos se aplicarán en redes de diferentes tamaños.

1. TÉCNICAS PARA EVITAR EL SOBREAJUSTE

En este apartado se desarrollarán y comentarán las principales técnicas que permiten evitar o, al menos, reducir el sobreajuste.

1.1. PRIMERA TÉCNICA. EARLY STOPPING

El método “Early stopping” o parada temprana en castellano consiste en pausar el entrenamiento antes de que el modelo empiece a producir el sobreentrenamiento. La idea intuitiva tras esta técnica es simple, si el problema radica en que una red se ajusta en demasía a los datos de entrada una posible solución sería detener el entrenamiento antes de que se produzca. Un riesgo de aplicar esta técnica es interrumpir el entrenamiento excesivamente pronto, no alcanzando el entrenamiento máximo previo al sobreajuste.

IBM explica así la parada temprana: “este método consiste en pausar el entrenamiento antes de que el modelo empiece a aprender el ruido dentro del modelo. Este enfoque corre el riesgo de detener el proceso de entrenamiento demasiado pronto, lo que puede generar el problema opuesto de subajuste. Encontrar el "punto óptimo" entre subajuste y sobreajuste es el objetivo final aquí.” (*¿Qué es el sobreajuste? | IBM*, s. f.).

1.2. SEGUNDA TÉCNICA. “DROPOUT”

Anteriormente se ha comentado que una de las principales formas de evitar o reducir el sobreentrenamiento es reducir la complejidad o el tamaño de la red convolucional. El abandono o “dropout” es una de las que pretenden reducir el tamaño efectivo de la red.

La técnica consiste en desactivar unas pocas neuronas seleccionadas aleatoriamente en cada iteración. Estas neuronas son abandonadas durante esa iteración de forma que cada época de entrenamiento la red es diferente, además de ser más simple y de menor tamaño que la original.

Hay varias consecuencias de bloquear estas neuronas. Las dos primeras consecuencias se dan en la misma iteración, la neurona no participa ni en la “forward propagation” ni en la “back propagation” afectando tanto a los resultados del entrenamiento como a la actualización de los pesos.

> “El abandono es una técnica en la que las neuronas seleccionadas al azar son ignoradas durante el entrenamiento. Ellas se abandonan al azar. Esto significa que su contribución a la activación de las neuronas aguas abajo se elimina temporalmente en el pase directo y las actualizaciones de peso no se aplican a la neurona en el pase hacia atrás.” (Fernandez, 2018)

Dado que en cada época del entrenamiento la red convolucional que es entrenada es diferente, las neuronas abandonadas son diferentes y por tanto la red es única cada vez, existen múltiples modelos muy diversos cuyas representaciones internas de los datos de entrada son diferentes, de forma que se crea la posibilidad de generar azarosamente un modelo especialmente óptimo al generalizar los resultados. Además, otra consecuencia del “dropout” y de las diversas representaciones es que la ConvNet consigue una menor sensibilidad a los pesos específicos de los nodos.

> “ … si las neuronas son abandonadas aleatoriamente de la red durante el entrenamiento, otras neuronas tendrán que intervenir y manejar la representación requerida para hacer predicciones de las neuronas perdidas. Se cree que esto resulta en múltiples representaciones internas independientes aprendidas por la red. El efecto es que la red se vuelve menos sensible a los pesos específicos de las neuronas.” (Fernandez, 2018)

Finalmente se puede observar que el modelo generaliza mejor y es muy probable que evite o reduzca el “overfitting”.

1.3. TERCERA TÉCNICA. TAMAÑO DE LA RED

Esta técnica no tiene complicación alguna. Simplemente es añadir o eliminar capas para variar el tamaño de red con objeto comprobar si la red mejora tras el entrenamiento. Si el tamaño de la red es demasiado grande se produce sobreajuste debido a que la complejidad del modelo provoca que los pesos se ajusten en exceso a los datos de entrada y que las neuronas se especializan en las estructuras de las imágenes de forma demasiado concreta, recreando incluso el ruido.

Por otro lado, si el tamaño de la red es escaso es probable que se solucione el sobreajuste. Sin embargo, aparecerán otros problemas relativos a la capacidad y complejidad deficientes para analizar patrones complejos presentes en los datos. Dado que no puede aprender patrones complejos esta pequeña red es incapaz de realizar algunas tareas o distinguir características sutiles.

1.4. CUARTA TÉCNICA. ACTIVACIÓN DEL CLASIFICADOR

En el capítulo anterior, en el subapartado “2.2.2.2 Pilas de capas densamente pobladas” se explica el funcionamiento de la capa densamente pobladas que funcionan como clasificador. La operación varía según la función de activación escogida, es posible que otra operación sea más efectiva.

Algunas funciones de activación muy utilizadas que son aplicables a un clasificador son las siguientes:

- Sigmoidea: definida como $f(x) = \frac{1}{1 + e^{-x}}$

  Se aplica principalmente en problemas de clasificación binaria y con objeto de reducir el peso de los valores extremos, limitando el rango entre o y 1. Su problema principal es el “desvanecimiento del gradiente”, el cual ralentiza el aprendizaje. Se usa como “activation=‘sigmoid’ ”.

- Softmax: definida como $f(x) = \frac{e^{x_i}}{\Sigma\, e^{x_j}}$

  Para cada elemento x_i del vector de entrada x. La función transforma un vector de entrada en uno con la misma dimensión que representa la probabilidad de pertenencia a

una clase. Por ello es útil en las capas de salida en redes que resuelven problemas de clasificación multiclase. Se usa como "activation='softmax' ".

- Tanh: definida como $f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

  De forma similar a la fórmula sigmoidea se aplica en problemas de clasificación, especialmente binaria. También reduce los valores extremos, pero como el rango es de -1 a 1 permite manejar correctamente los números negativos.

- Swish: definida como $f(x) = \frac{x}{1 + e^{-x}}$

  Esta es una ecuación relativamente nueva (2017) de alto coste computacional que mejora la función sigmoidea evitando el "desvanecimiento del gradiente".

1.5. QUINTA TÉCNICA. AUMENTO DE DATOS

Anteriormente se ha comentado que una de las principales formas de evitar o reducir el sobreentrenamiento es aumentando el número de datos de entrada. Sucede a nivel práctico que no es posible conseguir datos nuevos, tal vez no tengas acceso a una base de datos mayor o no tienes capacidad para procesar o editar esas nuevas imágenes. Tampoco se pueden reutilizar los datos pues facilita el sobreajuste. Para solventar estas dificultades, pero aumentar la cantidad de datos se utiliza esta técnica. "La técnica de sintetizar datos nuevos a partir de datos existentes se conoce como 'Aumento de datos'." (Jay, 2022)

La técnica sirve para cualquier tipo de información incluso textos. El aumento de datos consiste en, justo al inicio de la red, multiplicar el número de datos de entrada. En el caso de las imágenes, reproduciendo estas mismas con pequeñas variaciones. Existen diferentes tipos de modificaciones aplicables a imágenes de entrada, algunos de los cuales son:

- Filtros kernel o desenfoque: Esta modificación agudiza o desenfoca la imagen.
- Unión de imágenes: Este enfoque permite combinar o mezclar dos o más imágenes.
- Borrado aleatorio: Consiste en suprimir pequeñas porciones de la imagen.
- Transformaciones geométricas: Esta técnica realiza transformaciones de tipo geométrico, como por ejemplo rotar, recortar o girar las imágenes.
- Voltear una imagen: Este enfoque cambia la orientación de las imágenes, de vertical a horizontal y viceversa.
- Flip o espejo: Esta modificación consiste en cambiar la posición de las imágenes de forma similar a colocar un espejo, tanto horizontal como vertical.

- Alteración del espacio color: Esta modificación permite transformar los canales de color RGB o cambiar el color seleccionado.
- Re-Scaling o zoom: Se trata de editar la escala visual. Se puede hacer que la imagen sea más grande o pequeña.

Las modificaciones que utilizaremos durante este proyecto serán la aplicación de un espejo, una rotación y un zoom o reescalado.

1.6. SEXTA TÉCNICA. REGULARIZACIÓN

La sexta y última técnica que se utilizará en este proyecto es la regularización. Esta técnica pretende reducir la complejidad del modelo. Una forma de entenderlo de forma simplificada sería " … penaliza la complejidad innecesaria en nuestro modelo" (*Regularización. ¿Qué, por qué, cuándo y cómo?*, s. f.). Además de limitar la complejidad del modelo y, obviamente, reducir el sobreajuste; la regularización pretende mejorar el rendimiento y aumentar la generalización del modelo. Para conseguir estos objetivos se debe reducir el tamaño de los valores extremos de los pesos y sesgos. Conocemos que esos pesos y sesgos se producen al entrenar, así pues, se limitan añadiendo restricciones adicionales. Dado que se pueden generar todo tipo de restricciones existen múltiples posibles regularizaciones, las que se utilizarán en este proyecto serán L1 y L2.

Mencionar que la segunda técnica, .Dropout(), es considerada por muchos un tipo de regularización. Debido a que se pueden aplicar a la vez el abandono y una regularización en este proyecto se mencionan siempre como técnicas diferentes.

Regularización L1:

La regularización L1, también conocida como regularización de LASSO (Least Absolute Shrinkage and Selection Operator), penaliza los valores absolutos de los pesos hasta el punto de conseguir que algunos pesos valgan cero. Se define como la adición a la función de pérdida de la suma de los valores absolutos de los pesos del modelo.

En cuanto al efecto sobre el Overfitting, los rasgos mencionados a continuación ayudan a controlarlo. La penalización incluida en esta regularización favorece las soluciones dispersas, reduce la complejidad del modelo, mejora la selección automática de características y controla el tamaño de los pesos.

El efecto sobre la selección de características es indudable, descartando los pesos de atributos de poca importancia y, por tanto, reduciendo la complejidad del modelo. Como el modelo se vuelve más robusto la regularización L1 suele resultar útil en aplicaciones que incluyen seleccionar rápidamente características como por ejemplo el reconocimiento facial.
Matemáticamente se expresa como:

$$L1 = \lambda * \Sigma|w|$$

Donde:

- λ es el hiperparámetro que controla la fuerza de la regularización.
- Σ|w| es la suma de los valores absolutos de los pesos del modelo.

Regularización L2:
La regularización L2, también conocida como regularización de peso o Ridge regression, penaliza el cuadrado de la norma euclídea de los pesos, empequeñeciendo los pesos evitando que se alcancen el cero. Se define como la adición a la función de pérdida del cuadrado de la norma euclidiana (o L2) de los pesos del modelo.

Este tipo regularización permite controlar los mayores pesos, simplifica el modelo, aporta estabilidad y permite una regularización global. Todos estos atributos evitan que el modelo sea demasiado sensible a los cambios y no rechazan pequeñas características de los datos, de manera que el sobreajuste se reduce y aumenta la generalización.

Debido a las características recientemente especificadas anteriormente la regularización L2 es muy utilizada en aplicaciones de regresión lineal, regresión logística, procesamiento de lenguaje natural y clasificación, detección y reconocimiento de objetos.

Matemáticamente se expresa como:

$$L2 = \lambda * ||w||^2$$

Donde:

- λ es el hiperparámetro que controla la fuerza de la regularización.
- $||w||$es el cuadrado de la norma euclídea de los pesos del modelo.

2. ESTRATEGIAS PARA EVITAR EL SOBREAJUSTE

En este proyecto se denomina estrategia a la combinación de al menos dos técnicas. Dado que hay seis técnicas la cantidad de estrategias posibles es excesivamente elevada. Con objeto de poder analizar correctamente los resultados sólo se aplicarán las variantes más efectivas de las principales estrategias. Además, la primera de las técnicas, "early stopping" se puede aplicar siempre de forma que en caso de que su inclusión sea beneficiosa se añadirá a la estrategia.

2.1. PRIMERA ESTRATEGIA. AUMENTO DE DATOS Y "DROPOUT"

A lo largo del capítulo se ha mencionado varias veces que, simplificando mucho, hay dos formas de evitar el sobreajuste, aumentar el número de datos de entrada y simplificar o reducir el modelo. Así pues, aplicar ambas formas de evitar el sobreajuste debe reducirlo. Esta estratégia aprovecha la técnica del aumento de datos, para obviamente aumentar el número de datos de entrada y aprovecha la técnica de abandono para simplificar y reducir el modelo.

2.2. SEGUNDA ESTRATEGIA. AUMENTO DE DATOS Y REGULARIZACIÓN

Al igual que la primera, la segunda estrategia pretende tanto aumentar el número de datos de entrada como reducir y simplificar el modelo. Sin embargo, la técnica aplicada para la segunda tarea es la denominada regularización.

2.3. TERCERA ESTRATEGIA. AUMENTO DE DATOS, REGULARIZACIÓN Y "DROPOUT"

Finalmente, se combinan las dos estrategias anteriores. Quisiera dejar claro que no tiene por qué ser la estrategia más efectiva pese a combinar un mayor número de técnicas, es posible que se reduzca o simplifique en exceso la red o simplemente que alguna técnica diferente consiga un resultado superior.

CAPÍTULO 4.

RED NEURONAL DE “MNIST”

El primer caso de estudio trata una pequeña aplicación de clasificación, específicamente la clasificación de dígitos escritos manualmente. Existe un conjunto de datos llamado “MNIST” que contiene fotografías de dígitos escritos a mano. En concreto, el conjunto contiene 70000 imágenes, de las cuales 60000 son para entrenar y 10000 para probar. Los dibujos del conjunto se mueven en una escala de grises y han sido normalizadas a tamaño 28x28 píxeles. Por último, este conjunto de datos asocia a cada una de las imágenes una etiqueta, label, que indica el dígito representado (0 a 9).

1. DESCRIPCIÓN Y EXPLICACIÓN DE TODOS LOS ASPECTOS REQUERIDOS EN LA CREACIÓN Y ENTRENAMIENTO DE LA RED NEURONAL DE “MNIST”

Esta aplicación consiste en la clasificación de dígitos escritos manualmente. En la ilustración 2 se muestran dos ejemplos de las imágenes de entrada.

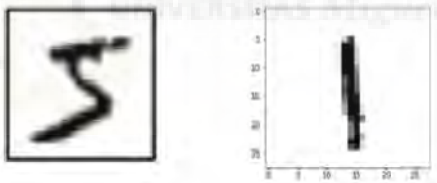

Ilustración 2.

1.1. DESCARGA DE “MNIST” Y PREPARACIÓN DE LOS DATOS

Bibliotecas:

El paso previo a todo programa escrito en python es importar las bibliotecas necesarias:

```
import keras
import numpy as np
from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical
```

Descarga de “MNIST”:

El conjunto "MNIST" se puede descargar en la página *The MNIST database*[1], pero en nuestro caso no será necesario pues está precargado en Keras en forma de 4 arrays Numpy.

```
(train_i, train_l), (test_i, test_l) = mnist.load_data()
```

Siendo *train_i* y *train_l* el conjunto de entrenamiento. *test_i* y *test _l* conforman el conjunto de test. Obviamente, los datos integrados en *train_i* y en *test_i* representan imágenes, por tanto, contienen un tensor 3D compuesto por el número de imágenes de tamaño 28 x 28 píxeles en escala de grises. *train_l* y *test_l* son vectores de dos dimensiones, la primera es el enlace a las imágenes, la segunda es un vector de tamaño 10.

Disminución del número de imágenes:

Como ya hemos indicado, "MNIST" contiene 60000 imágenes, datos más que suficientes para que una pequeña red convolucional no produzca sobreajuste. El objeto de este apartado es reducir el número de datos para permitir estudiar la mejora contra el sobreentrenamiento. Para ello se utiliza el siguiente código:

```
i=0
indice1=[]
indice2=[]
while i < 1000 :
  if i<100: indice2.append(i)
  indice1.append(i)
  i = i+1
x_train = train_images[indice1]
y_train = train_labels[indice1]
x_test = test_images[indice2]
y_test = test_labels[indice2]
```

No es más que dos arrays con los números enteros de 0 a 1000 y de 0 a 100 que hacen de índices en los tensores anteriores de forma que el número total de imágenes que se le introducirá son 1100.

Normalización:

Las redes neuronales no pueden aceptar datos de cualquier tipo, por ello siempre es necesario normalizar los datos de entrada. En este caso se requieren tres cambios, el primero es añadir

una dimensión más que introduzca la escala de grises indicando el número de canales (1), el segundo cambio es cambiar el tipo de dato y, por último, escalar los datos a [0,1].

```
x_train =x_train.reshape((1000, 28, 28, 1))
x_train = x_train.astype("float32") /255
x_test = x_test.reshape((100, 28, 28, 1))
test_images = test_images.astype("float32") /255
```

El método ".reshape" es una función muy utilizada en procesamiento de datos que permite cambiar el formato de un tensor sin alterar sus elementos. Es decir, facilita la reorganización de los datos en distintas dimensiones y tamaños manteniendo los componentes. El argumento que requiere ".reshape" es la nueva forma que toma el tensor, en nuestro caso: (1000, 28, 28, 1). Esta debe ser compatible con la anterior. Reshape, en definitiva, cambia la representación del tensor.

Otra función muy utilizada en el procesamiento de datos es el método ".astype()". Este método se aplica un tensor y permite convertir los elementos de estos a un tipo de datos diferentes. Hay que tener en cuenta que este método crea un nuevo objeto y por tanto es necesario asignar el resultado. El argumento que requiere ".astype()" es el tipo de datos al que se desea convertir los elementos del tensor, en este caso "float32". "Float" es el nombre del tipo de dato, número de punto flotante. "32" indica el número de bits. Otros argumentos más utilizados son: float64, int32 y uint8.

Las matrices que componen las imágenes suelen medir la intensidad de sus canales con valores entre 0 y 255. Por otro lado, las redes convolucionales requieren que estos datos tengan el rango de valores: [0, 1]. Para igualarlos simplemente hay que dividir los valores de intensidad entre 255.

1.2. CREACIÓN DEL MODELO DE UNA RED NEURONAL O CONVOLUCIONAL

Una vez descargados y normalizados los datos de entrenamiento, ya se puede diseñar el modelo. A grandes rasgos está compuesto por capas convolucionales con activación "relu" alternadas con capas "MaxPooling" que disminuyen el tamaño de la representación, permitiendo por una parte la disminución de parámetros (aumentarían mucho más), por otra la eliminación de información sobrante (bordes gruesos, etc) y, por último, el entendimiento de

los mapas de características. Tras estas capas se incluye una capa densa que actúa como clasificador. Finalmente, se crea un modelo que recoge este conjunto.

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=5, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=5, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

A continuación, se explican los principales métodos y las principales funciones utilizadas en el código anterior.

keras.Input:

Se define como una función de keras que crea una capa de entrada en una red neuronal. Permite definir la entrada de datos y especificar sus características; como, por ejemplo, el formato y tipo de datos. Sus principales argumentos son:

1. Shape: Especifica la forma de los datos de entrada. Este argumento es obligatorio. En este caso se aplica "`shape=(28, 28, 1)`" indicando que las imágenes de entrada deben tener un tamaño 28x28 píxeles con un solo canal de color, en escala de grises.
2. batch_size: Permite especificar el tamaño del lote (batch size) de los datos de entrada. El valor predeterminado es "None", lo que significa que el tamaño del conjunto de entrada puede variar. Si deseas fijar un tamaño específico, puedes proporcionar un número entero. En este caso se aplica el valor por defecto.
3. dtype: Indica el tipo de datos de los elementos de entrada. Posibles argumentos son algunos tipos de datos como float32, int32, etc. El valor predeterminado es "None", lo que significa que se utilizará el tipo de datos predeterminado de Tensor Flow, siendo este float32. En este caso se aplica el valor por defecto.
4. name: Permite asignar un nombre a la capa de entrada. Esto es útil cuando deseas acceder o referenciar la capa más adelante en el código. El valor por defecto es "None", el cual significa una asignación automática con un contador interno de Keras con formato "input_1" variando el número según el contador.

layers.Conv2D:

Es una función de keras que crea una capa convolucional. Tanto la entrada como la salida de estas capas están compuesta por un vector de rango 3 (altura, anchura, canales). El número de canales se determina con el primer argumento (filters) de las capas Conv2D que suelen ser 32, 64, 128 o 256, excepto en la primera entrada que viene determinada por el número de canales, en este caso 1. Las capas Conv2D de keras tienen los siguientes argumentos de entrada:

1. filters: Número entero que indica la cantidad de filtros o kernels de convolución que se aplicarán a los datos de entrada. Cada filtro extraerá una característica diferente de los datos de entrada. Por defecto se aplica "filters=1".
2. kernel_size: Número entero o tupla de dos números enteros que especifica la altura y el ancho de la ventana. Ejemplo: "kernel_size=3" o "kernel_size=(3,3)". Si el argumento introducido es un solo int se asume que es un cuadrado. Por defecto se aplica "kernel_size=(3,3)".
3. strides: Entero o tupla de dos números enteros que especifica los pasos que hay entre ventanas. El valor por defecto es "strides= (1, 1)".
4. padding: Cadena de strings que puede ser "valid" o "same". "valid" produce que la convolución solo se aplique en regiones donde los datos de entrada y los filtros se superponen por completo, es decir no se produce en los bordes de la entrada. Este es el valor predeterminado. Contrariamente, "same" aplica la convolución de forma que la salida tenga la misma forma que la entrada.
5. activation: Función de activación que se aplica a la salida de la capa convolucional. Algunos de los métodos de activación son:
    a. ReLU o Rectified Linear Unit: Definida como $f(x) = max(0, x)$

       Es una de las funciones más utilizadas y se aplica para eliminar entradas negativas y activar linealmente las entradas positivas. Esto implica una no linealidad y el entrenamiento de capas muy profundas y eficientes. Se usa como "activation='relu' ".

    b. Leaky ReLU: variante de la función ReLU que se define como $f(x) = max(zx, x)$ Donde z es un hiperparámetro menor que 1. Esta función hace que los valores negativos se propaguen a través de la convnet con una pequeña pendiente. Se usa como "activation='LeakyReLU' " tras haberla importado.

c. ELU o Exponential Linear Unit: variante de la función ReLU que se define como $f(x) = x$ si $x > 0$, y $f(x) = z * (e^x - 1)$ si $x < 0$. Esta función elimina el "deadReLU" (eliminación de los valores negativos por parte de ReLU) y varía la escala. Se usa como "activation=" 'elu' ".

Existen otras muchas funciones de activación, esto es solo una representación de las más importantes.

6. input_shape: Tupla de 3 ints que especifica la forma de los datos de entrada. El orden viene determinado por el argumento "data_format".
7. data_format: cadena de strings que puede ser "channels_last" o "channels_first". Señala dónde se posiciona la indicación de canales. Por defecto se aplica "data_format = 'None' ", lo que significa que se basará en la configuración de "image_data_format" en el archivo de configuración de Keras.

layer.MaxPooling2D:

Es una función de keras que crea una capa pooling El objetivo principal de la capa es reducir el tamaño del mapa de características. Consiste extraer ventanas de los mapas de características de entrada y obtener el valor máximo de cada canal. Normalmente las ventanas de esta función son 2x2 con saltos de valor 2 con el fin de reducir el mapa a la mitad. Es necesario comentar que una excesiva cantidad de parámetros previos a la capa densa genera un exceso de argumentos en esta, provocando un sobreajuste.

Las capas MaxPooling2D de keras tienen los siguientes argumentos de entrada:

1. pool_size: Número entero o tupla de dos enteros que dan tamaño a la ventana mencionada anteriormente. Cuando se introduce un solo entero, se asume que el alto y la anchura son iguales. El valor por defecto es: "pool_size=(2,2)".
2. strides: Entero o tupla de dos números enteros que representa el desplazamiento de la ventana. Por defecto, el valor es el mismo que el introducido en el argumento "pool_size".
3. padding: Cadena de strings que puede ser "valid" o "same". "valid" produce que la convolución solo se aplique en regiones donde los datos de entrada y los filtros se superponen por completo, es decir no se produce en los bordes de la entrada. Por lo contrario, "same" aplica la convolución de forma que la salida tenga la

misma forma que la entrada agregando el relleno que sea necesario. El valor por defecto es “valid”.

4. data_format: cadena de strings que puede ser “channels_last” o “channels_first”. Señala dónde se posiciona la indicación de canales. Por defecto se aplica “channels_last”.

Hay muchos otros argumentos opcionales como “name” que asigna un nombre único a la capa.

layers.Flatten:

Esta capa aplana el vector que recibe como entrada. Esto significa convertir el rango de los datos, de 3 a 1. Esto significa finalizar la red y proceder de manera secuencial.

layers.Dense:

Este comando genera un clasificador densamente conectado: una pila de capas densas. Los principales argumentos de la clase son:

1. units: Número entero que representa la dimensionalidad de la capa Dense. Especifica el número de salidas. Es un argumento obligatorio.
2. activation: Igual que en las capas “Conv2D” este argumento indica la función de activación a utilizar. En el capítulo 3, en el subapartado “3.1.4 Cuarta técnica. Activación del clasificador” se explican las principales funciones de activación que se utilizan para este comando. Su argumento por defecto es “None”, esto significa que no se aplica ninguna función de activación generando una salida lineal.
3. use_bias: Booleano que indica si hay agregar un sesgo a la capa. Por defecto su valor es “True” aplicando el sesgo.
4. kernel_initializer: Inicializa los pesos de la capa. Puede ser una cadena que simboliza un inicializador predefinido o un inicializador importado por el usuario. Su valor por defecto es “glorot_uniform”.
5. kernel_regularizer: Regulación opcional de los pesos de la capa. Es una instancia de una clase regularizadora, pero puede ser personalizada o no. Su valor por defecto es “None”, es decir no aplicar la regulación.
6. bias_regularizer: Regulación opcional de los sesgos de la capa. Es una instancia de una clase regularizadora pero puede ser personalizada o no.Su valor por defecto es “None”, es decir no aplicar la regulación.

7. activity_regularizer: Regulación opcional a la salida de la capa. Es una instancia de una clase regularizadora, pero puede ser personalizada o no. Su valor por defecto es “None”, es decir no aplicar la regulación.

keras.Model:

Función que permite crear un modelo de red neuronal. Este modelo estructura las diferentes capas de la red coherentemente y define el flujo de datos interior. Esta función puede crear tanto modelos simples (como el de este ejemplo) como modelos complejos, pasando por modelos funcionales y modelos de subclases. Algunos de los argumentos habituales son:

1. inputs: Indica las capas (o capa) de entrada del modelo. Este parámetro es obligatorio y no tiene valor predefinido.
2. outputs: Muestra las capas (o capa) de salida del modelo. Este argumento es obligatorio y no tiene valor por defecto.
3. name: Permite asignar un nombre al modelo para referenciar más adelante. En este caso no tiene sentido pues el modelo es asignado a una variable.
4. compile: Booleano que indica si se debe compilar el modelo tras ser creado. Su valor por defecto es “False”. En caso de indicar “True” hay que proporcionar la configuración de compilación adicional.
5. Existen varios argumentos que especifican datos requeridos para el entrenamiento como son “optimizer”, “loss”, “metrics”, etc. Se comentará sobre ellos en la etapa de entrenamiento.

Hay otros parámetros que permiten una mayor personalización del modelo, por ejemplo: “loss_weights”, “sample_weight_mode” y “weighted_metrics”.

1.3. COMPILACIÓN Y ENTRENAMIENTO DEL MODELO

1.3.1. COMPILACIÓN DEL MODELO

Tras crear el modelo y antes de realizar el entrenamiento hay que compilar esta red convolucional. Compilar se refiere a configurar las opciones de entrenamiento. Se especificarán detalles importantes como el optimizador, la función de pérdida y las métricas de evaluación.

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

El método ".compile()" es, como el nombre indica, permite compilar el modelo. Los argumentos más comunes son:

1. loss: Permite seleccionar la función que determina el error del modelo. La seleccionada en este caso, "binary_crossentropy" mide la discrepancia entre la distribución de probabilidad predicha y la distribución de probabilidad real de las clases objetivo. Cuanto menor sea el valor de la función de pérdida, mejor será el ajuste del modelo a los datos de entrenamiento. Otras funciones de error son: "categorical_crossentropy", "sparse_categorical_crossentropy", "MeanSquaredError" y "mean_absolute_error".
2. optimizer: Especifica el optimizador que se utiliza durante el entrenamiento. El optimizador escogido en este caso es RMSprop o "rmsprop", el cual es un algoritmo que cambia cada uno de los pesos dependiendo del aprendizaje. Esto permite un entrenamiento más eficiente de la red neuronal. Existen muy diferentes optimizadores, algunos de ellos son: Adam, SGD,Adagrad, Adadelta, y Adamax.
3. metrics: Tupla de strings que se utilizarán para evaluar el rendimiento del modelo. En este caso se usa la precisión.
4. loss_weights: Pesos opcionales y adicionales que se pueden asignar a las salidas si son múltiples
5. sample_weight_mode: El modo en que los pesos de muestra se tendrán en cuenta durante el entrenamiento.

1.3.2. ENTRENAMIENTO DEL MODELO

Procedemos a entrenar modelo. Previendo que puede ser necesario reutilizar los pesos de un determinado punto del entrenamiento se utiliza la función "ModelCheckpoint()" de la clase callbacks de keras. Además, la función será asignada a la variable callbacks.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
```

Esta función tiene varios argumentos, los utilizados son:

1. filepath: Cadena de texto que representa el nombre del archivo que guarda el modelo. Este archivo tiene que acabar con ".keras".
2. monitor: Cadena de texto que indica la métrica a monitorizar.
3. save_best_only: Booleano que indica cuando hay que guardar el modelo (como es en un único archivo, el modelo se sobreescribe). Si el valor es "True" sólo se guardan los mejores modelos según la métrica monitorizada.
4. save_weights_only: Un booleano que indica si se deben guardar solo los pesos del modelo en lugar de todo el modelo. Si se establece en "True", solo se guardarán los pesos. Por defecto, es "False".
5. mode: Indica si se debe maximizar (max) o minimizar (min) la métrica monitoreada. Por ejemplo, "mode='min' " se usa cuando se quiere minimizar la pérdida. Por defecto se utiliza "mode='min' ".
6. save_freq: Especifica con qué frecuencia se deben guardar los pesos del modelo. Puede ser un número entero (representando el número de épocas) o una cadena en formato de época ("epoch") o paso ("batch"). Por ejemplo, "save_freq=10" guardará los pesos cada 10 épocas. El valor predefinido es "save_freq=1".

El entrenamiento en sí mismo se produce a continuación mediante el método ".fit". El resultado se guarda en la variable "history" con objeto de poder representar los resultados.

```
history = model.fit(
    x_train,
    y_train,
    batch_size=64,
    epochs=60,
    callbacks= [callbacks],
    validation_split= 0.2)
```

1. x_train e y_train: Son los datos de entrenamiento, donde "x_train" son las características de entrada e "y_train" son las etiquetas de salida correspondiente. Estos dos argumentos son sustituibles por una variable que guarde los lotes de datos de entrenamiento. Son argumentos obligatorios.
2. epochs: Número entero que representa la cantidad de veces que el modelo recorrerá los datos de entrenamiento, si se inserta un número excesivamente alto en este argumento se producirá un sobreajuste. Este es un parámetro obligatorio.

3. batch_size: Dato tipo int que determina el número de muestras que se utilizan en cada paso de entrenamiento para actualizar los pesos del modelo.
4. validation_split: número decimal entre 0 y 1, que representa la fracción de datos que se reservarán para la validación. El conjunto de validación se extrae automáticamente del final del conjunto de datos de entrenamiento. Este argumento puede ser sustituido por el próximo.
5. validation_data: Variable que proporciona datos de validación para evaluar el rendimiento del modelo durante el entrenamiento.
6. callbacks: Permite especificar una o varias listas de objetos de tipo Callback que se ejecutarán durante el entrenamiento.

1.4. REPRESENTACIÓN DE LAS CURVAS DE PRECISIÓN Y ERROR

Código de representación:

Con objeto de analizar los resultados del entrenamiento se van a representar. Lo primero es almacenar en unas tuplas la precisión, la precisión de los datos de validación, el error, el error de los datos de validación y las épocas.

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
```

La función “plot” dibuja la curva deseada. Los parámetros necesarios a indicar son el “eje x” (epochs), el “eje y” (tupla a representar), información de representación y la leyenda. El código no está bien ordenado para unificar aquellas líneas de código que tratan la misma función.

```
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
```

```
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
```

Las funciones “.title()” y “.legend()” generan los títulos y las leyendas de las gráficas respectivamente.

```
plt.title("Training and validation accuracy")
plt.legend()
```

```
plt.title("Training and validation loss")
plt.legend()
```

Así mismo, ".figure()" permite cambiar dos gráficas.

```
plt.figure()
```

Finalmente, se utiliza ".show" para mostrar la representación.

```
plt.show()
```

El código entero y organizado:

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

Representación del entrenamiento de una red MNIST:

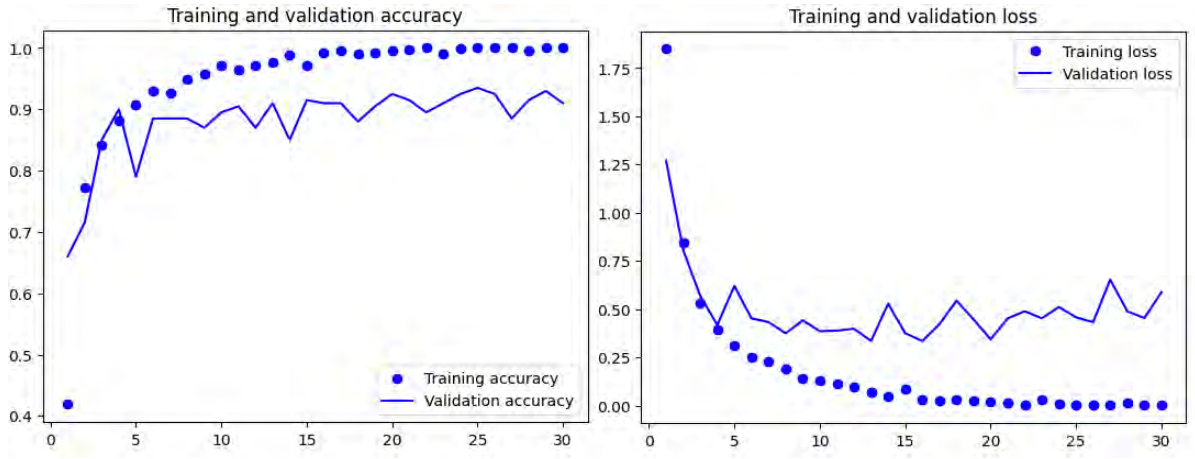

Imagen 1. División en Figuras 1 y 2.

En la figura 1 se representa la precisión (accuracy) en el eje Y. Se puede observar que en el caso de las imágenes train, se alcanza una precisión cercana al 100%. Por el contrario, los datos de validación solo alcanzan una precisión aproximada del 90%. Esta diferencia es el llamado sobreajuste. Además, se advierte que ambas líneas tienen la misma tendencia en un principio, hasta que la precisión se acerca al 90%. En esta red neuronal, con estos datos, se produce antes de la época 5.

MNIST	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%

Tabla 1.

En la figura 2 se representa el error (loss) en el eje Y. Se advierte que el error en los datos train se aproxima a 0. Opuestamente, el error en los datos de validación se mantiene en torno a 0.5 puntos. Igual que con la precisión, en esta diferencia se entiende el sobreajuste. Los datos de entrenamiento y de validación tienen al principio unos valores similares, pero a partir de la época 5 el error de validación se mantiene.

MNIST	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5

Tabla 2

2. APLICACIÓN DE TÉCNICAS CONTRA EL SOBREAJUSTE EN LA RED NEURONAL CONVOLUCIONAL DE MNIST

Entendida ya la red convolucional MNIST y su proceso de entrenamiento con limitación de datos se aplican sobre ella las diferentes técnicas y estrategias diseñadas para mitigar el sobreajuste. Pese a que ya se ha explicado en profundidad los diferentes métodos no sólo se expondrán los efectos sobre el sobreajuste de la red, sino también una pequeña descripción.

2.1. PRIMERA TÉCNICA: EARLY STOPPING

El sobreajuste se produce por un exceso de entrenamiento, para evitarlo se implementa un callbacks llamado EarlyStopping. De esta forma el entrenamiento se detiene en el momento correcto. El código editado es el entrenamiento del modelo, en concreto:

```
from tensorflow.keras.callbacks import EarlyStopping
callbacks = [keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
early_stopping = EarlyStopping(
    monitor= "val_loss",
    patience = 3,
    restore_best_weights = True
)
history = model.fit(
    x_train,
    y_train,
    batch_size=64,
    epochs=30,
    callbacks= [callbacks, early_stopping],
    validation_split= 0.2)
```

La llamada "early_stopping" requiere los argumentos "monitor", "patience" y "restore_best_weights".

- Patience: Indica el número de epochs sin mejorar antes de detener el entrenamiento
- Restore_best_weights: Similar a "save_best_only" en cuanto al objetivo, restaura los pesos del modelo al mejor punto durante el entrenamiento.

Se asigna 3 a "patience" pues es un valor suficientemente elevado para evitar falsas tendencias y suficientemente bajo para disminuir el sobreajuste. Se aplica sobre la gráfica señalada con "monitor", normalmente el error de validación. Dando como resultado:

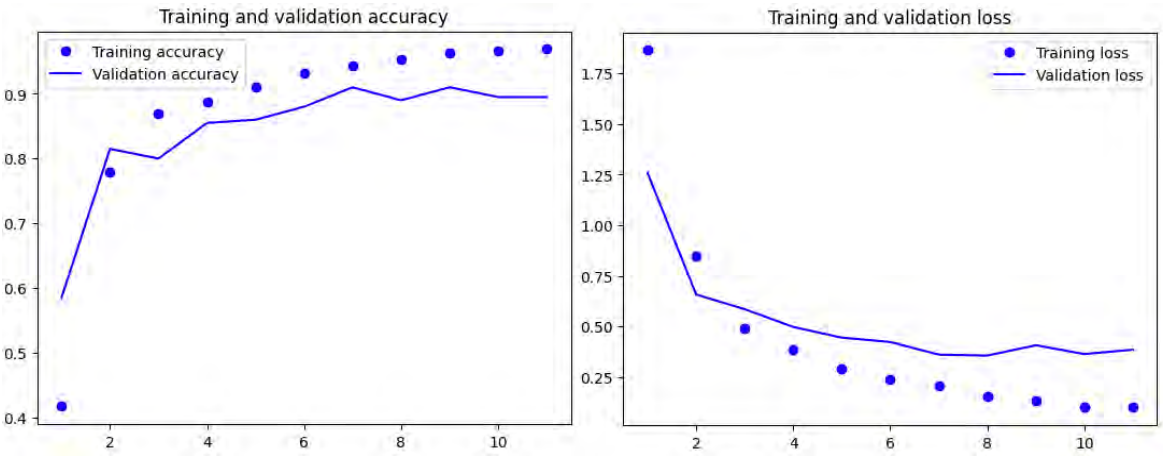

Imagen 2. División en Figuras 3 y 4.

En el caso de una red de pequeño tamaño parece no ser tan efectivo permitiendo cierto sobreentrenamiento. El entrenamiento se finaliza en la época 11, sabiendo que el sobreajuste se produce antes de la 5. También, se puede notar que no se han llegado a estabilizar las tendencias.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
Early Stopping	96.88%	90%	<5	6.88%

Tabla 3.

Claramente mejora el sobreajuste de precisión, pero sin mejorar la precisión de validación. En la gráfica 3, se observa que la curva de entrenamiento tiende a aumentar de forma constante hacia el 100% pero teniendo como valor máximo 96.88%. Contrariamente, la curva de validación se centra en torno al 90%.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
Early Stopping	0.1003	0.3614	<5	0.2611

Tabla 4.

En este caso, se observa una mejora del sobreajuste del error y error de validación pero, ni es una mejora suficiente ni consigue alcanzar el objetivo (val_loss = 0.00). Respecto a la representación 4, el error de entrenamiento tiende a 0.00 pero solo logra un mínimo de 0.1003. Así mismo, el error de validación en torno al 0.38 alcanzando un máximo en 0.4078.

2.2. SEGUNDA TÉCNICA: “DROPOUT”

El método “.dropout()” permite una pequeña simplificación y regulación del modelo.

2.2.1. CAMBIO DEL CÓDIGO PARA AÑADIR LA TÉCNICA “DROPOUT”

Una segunda forma de evitar el sobreajuste es simplificar el modelo, para ello se añade el método .Dropout() a la red convolucional. El resto del código es el original.

```
inputs = keras.Input(shape=(28, 28, 1))
```

```
x = layers.Conv2D(filters=32, kernel_size=5, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=5, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Recuerdar que las próximas comparaciones son exclusivamente respecto al origen.

Resultados Dropout(0.5):

0.5 es el parámetro que se suele añadir por defecto.

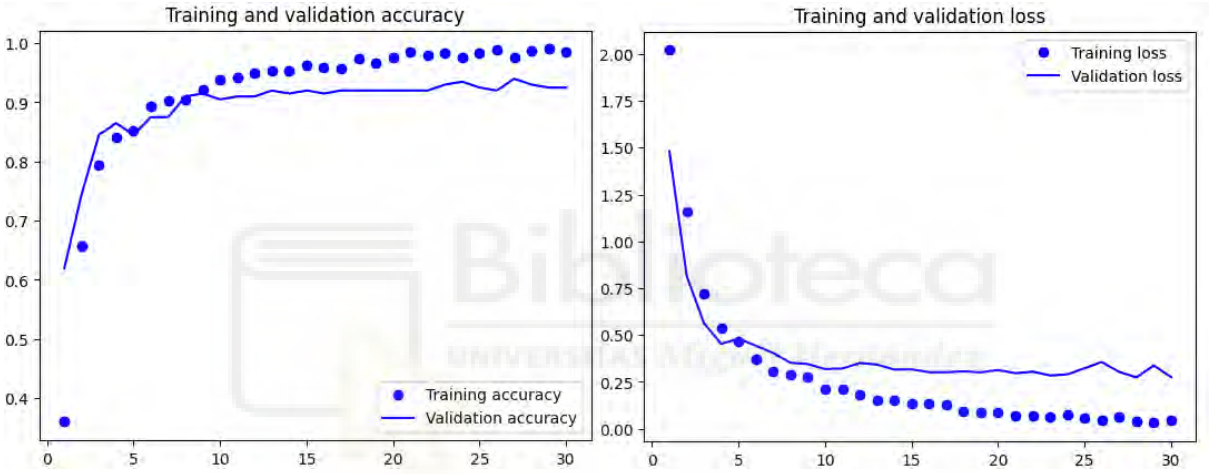

Imagen 3. División en Figuras 5 y 6.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
.Dropout(0.5)	99%	94%	10	5%

Tabla 5.

Existe mejora en el sobreentrenamiento reduciéndose en un 5%, produciéndose al menos 5 épocas después. Esto es en parte consecuencia de la mejora en la precisión de validación. En la figura 5, alcanzando la estabilidad en torno al 92% con un máximo en el 94%. La otra cara, menos amable, es el pequeño empeoramiento de la precisión de entrenamiento que no logra cumplir el objetivo de aproximarse al 100% alcanzando únicamente el 99%.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
.Dropout(0.5)	0.031	0.2743	6	0.2433

Tabla 6.

En relación a la figura 6, el error de entrenamiento no alcanza 0 exacto, pero parece mostrar una tendencia que eventualmente convergerá. Esta curva tiene por valor mínimo 0.031. Por otro lado, la tendencia del error de validación se estabiliza alrededor de 0.3, con un valor mínimo de 0.2761. Pese al minúsculo empeoramiento del error de entrenamiento (se incumple uno de los objetivos), el error de validación y el sobreentrenamiento disminuyen casi en un 50%; retrasando así el inicio del sobreajuste a la época 6 (pequeña pero existente diferencia).

Resultados Dropout(0.25) :

Veamos que sucede al disminuir el parámetro a la mitad.

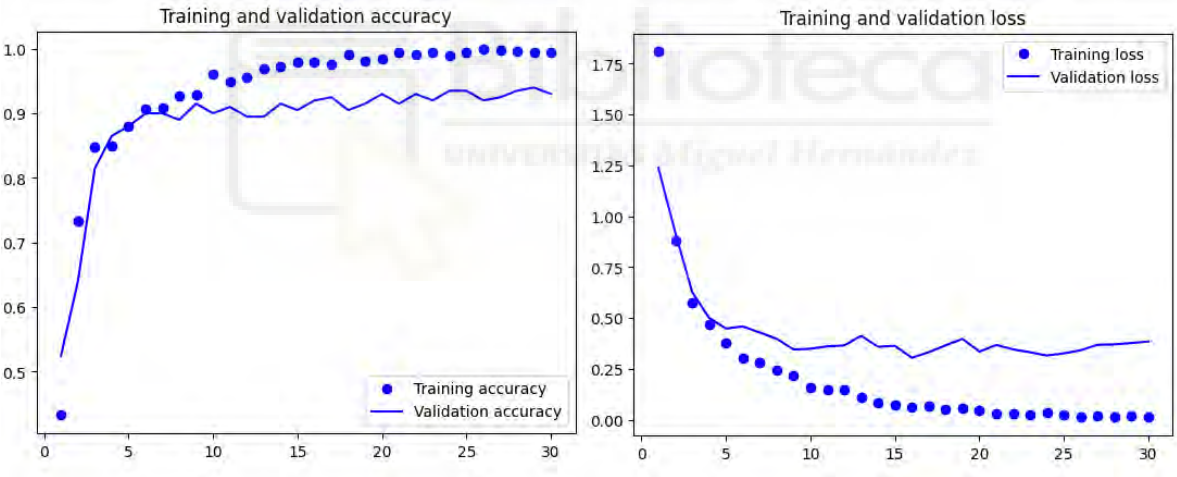

Imagen 4. División en Figuras 7 y 8.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
.Dropout(0.25)	99.87% (aprox 100%)	94%	10	5.87% (aprox 6%)

Tabla 7.

Existe una mejora en la precisión de validación, la cual tiene tendencia ascendente y alcanza un máximo en el 94%. Esta mejora explica, en parte, la disminución del sobreentrenamiento

de precisión en un 4%. Una posible razón para ambas mejoras es el retraso en al menos 5 épocas del sobreajuste. En la gráfica 7 tenemos una aproximación de la precisión de entrenamiento al 100% con un máximo en 99.87%.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
.Dropout(0.25)	0.014	0.3043	5	0.2903

Tabla 8.

Respecto a la gráfica 8, el sobreentrenamiento se observa en la quinta época. Además, la mejora del sobreajuste valorada en poco más de 2 décimas se debe mayoritariamente a la mejora del error de validación; el cual cuenta con un valor 0.3043 producido en la época 16. La gráfica de validación se estabiliza en torno a un valor cercano a 0.33. Apenas varía el error de entrenamiento teniendo el error mínimo en 0.0140 puntos.

Resultados Dropout(0.75):

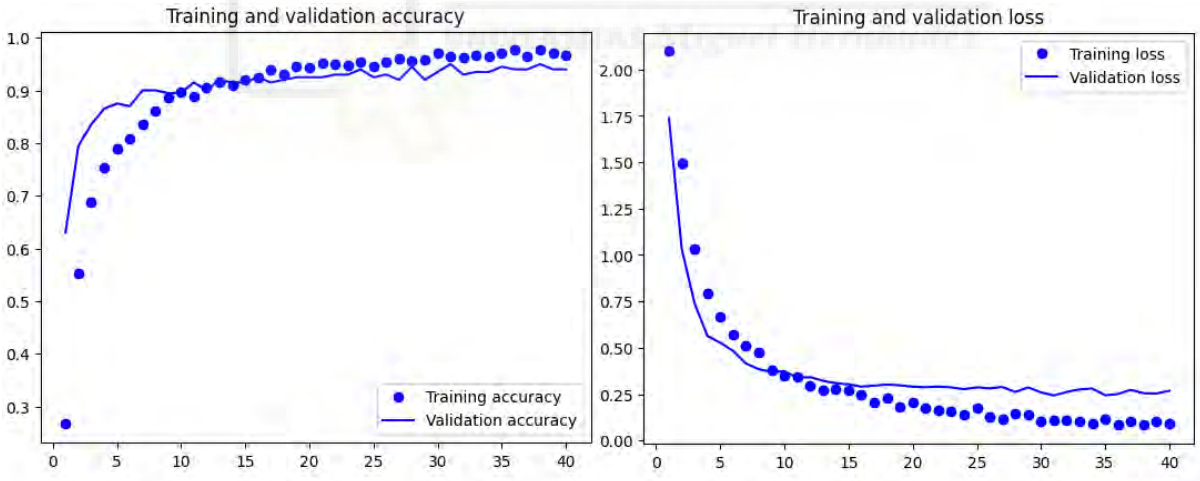

Imagen 5. División en Figuras 9 y 10.

En la gráfica de precisión, figura 9 se observa la tardanza, más allá de la época 25, y el muy reducido valor del sobreentrenamiento. En el otro lado de la imagen 5, la gráfica de error es más similar a las anteriores.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión

Original	100%	90%	< 5	10%
.Dropout(0.75)	97.62%	95%	>25	2.62%

Tabla 9.

El que el sobreajuste sea pequeño se debe a dos motivos, el primero es la gran precisión de validación con propensión al 94% y máximo del 95%. La segunda razón es la disminución de la precisión de entrenamiento, logrando un máximo de 97.62%. En mi opinión, la bajada de la precisión de entrenamiento eclipsa absolutamente el resto de las mejoras pues, ¿de qué sirve disminuir el sobreajuste si no se logra mantener el 100% en entrenamientos? Con esta precisión de entrenamiento, no pasaría nada el reducir a cero el sobreentrenamiento, pues la precisión de validación no alcanzaría el objetivo del 100%. Aun así, se producen mejoras sustanciales, un 5% en la precisión de validación, un gran retraso del sobreajuste, cuyo desfase máximo es un 2.62%

MNIST. Gráfica de error	**Error de entrenamiento**	**Error de validación**	**Época de sobreajuste**	**Sobreajuste de error**
Original	0.00	0.5	< 5	0.5
.Dropout(0.75)	0.0826	0.2433	>15	0.1607

Tabla 10.

Como siempre, el error de entrenamiento empeora respecto al original; pero cada vez más, tanto este como el error de validación se aproximan al objetivo, puntuar 0 justos. Con este "dropout" se consigue reducir en más de la mitad el error de validación, alejar en más de diez épocas el sobreajuste cuyo desfase de mínimos es 0.1607.

Sin embargo, también se produce un retraso del sobreentrenamiento hasta, al menos, la época 15. El error de entrenamiento pretende aproximarse al 0 absoluto, alcanzando un mínimo en 0.0826 y estabilizandose en valores próximos a 0.25. Así mismo, el error de validación tiene un mínimo en 0.2433.

2.2.1. ANÁLISIS COMPARATIVO ENTRE ABANDONOS

Cambiar el parámetro de la función ".Dropout(n)" nos aporta cierta variabilidad, a continuación se realizará un análisis comparativo entre las resoluciones prácticas de los cambios del argumento.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
.Dropout(0.25)	99.87% (aprox 100%)	94%	10	5.87% (aprox 6%)
.Dropout(0.5)	99%	94%	10	5%
.Dropout(0.75)	97.62%	95%	>25	2.62%

Tabla 11.

Precisión de entrenamiento: El objetivo de la red sobre la precisión de entrenamiento es, obviamente, mantenerse muy próximo del 100%. Tanto. Dropout(0.25) y .Dropout(0.5) superan el 99% pero, .Dropout(0.75) disminuye en más de un 2%.

Precisión de validación: Otro de los propósitos de la red es lograr una precisión de entrenamiento lo más cercana posible al 100%. En este sentido, el parámetro más adecuado es 0.75.

Época de sobreajuste: Retrasar el sobreentrenamiento permite alcanzar valores de precisión más elevados y a disminuir su máximo. El argumento 0.75 es claramente la opción óptima.

Sobreajuste de precisión: Ya sabemos que el sobreajuste se puede definir como la diferencia entre las gráficas de entrenamiento y las gráficas de validación, de test o de ejecución. El objetivo final de implementar el método .Dropout() es disminuir al máximo este aproximando los datos de validación al 100%, en este sentido todos mejoran a la red original, pero 0.75 es la opción más sobresaliente.

En conclusión, en cuanto a la precisión, la recomendación es 0.75 en muchos de los casos. Solo no es recomendable cuando la precisión de entrenamiento sea limitante.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
.Dropout(0.25)	0.014	0.3043	5	0.2903
.Dropout(0.5)	0.031	0.2743	6	0.2433
.Dropout(0.75)	0.0826	0.2433	>15	0.1607

Tabla 12.

Respecto a los datos de error, el principal enfoque consiste en reducir el sobreajuste acercando a cero tanto el error de entrenamiento como el error de validación.

Error de entrenamiento: El fin de la red neuronal sobre este punto es aproximar la curva a 0. Bajo esta premisa, disminuir el argumento fortalece la red.

Error de validación: Igual que en el caso anterior la meta de la red es aproximar la gráfica a 0. Desde esta óptica, aumentar el parámetro potencia la convnet.

Época de sobreajuste: Retrasar el inicio del sobreajuste posibilita disminuir tanto el propio sobreentrenamiento, la precisión de entrenamiento y, sobre todo, de validación.

Sobreajuste de error: El objetivo final de implementar el método .Dropout() es disminuir al máximo el sobreajuste aproximando los datos de validación al 100%, en este sentido todos mejoran a la red original, pero 0.75 es la opción óptima.

En conclusión, en cuanto al error, la recomendación es 0.75 en muchos de los casos. Solo no es recomendable cuando el error de entrenamiento es limitante.

2.3. TERCERA TÉCNICA: TAMAÑO DE LA RED

Es bien conocido que tanto una red convolucional excesivamente grande como una demasiado pequeña pueden contribuir al problema de sobreajuste. Por ello, esta técnica trata directamente la cantidad de capas neuronales.

Eliminar capas:

Esta técnica disminuye el número de capas neuronales como podemos observar en el siguiente código:

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=5, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Los resultados de este modelo se representan:

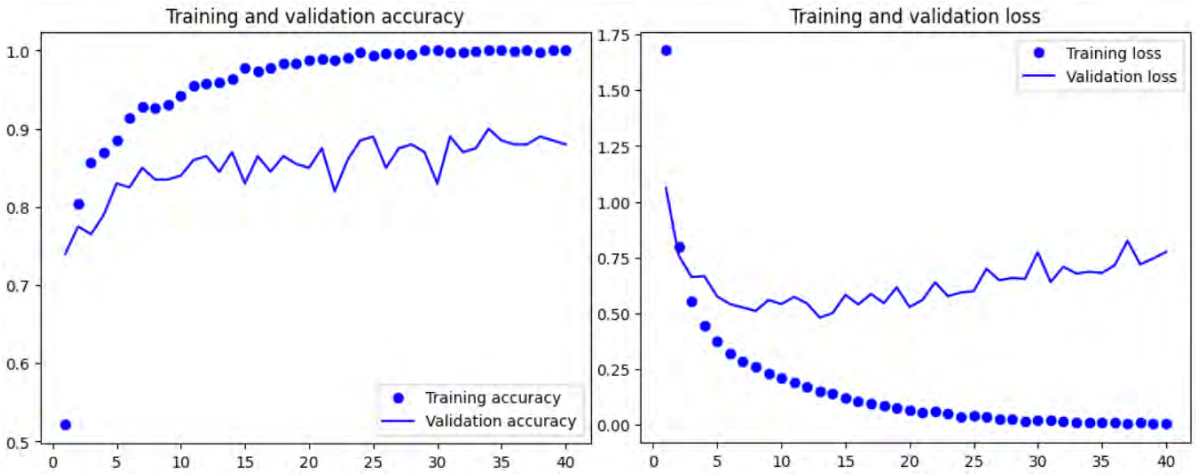

Imagen 6. División en Figura 11 y 12.

Pese a que gráficamente pueda parecer lo contrario no hay una gran variación ni en la precisión ni en el error, salvo, quizás, en la tendencia del error de validación. La discrepancia visual se debe a que el programa nos muestra los resultados integrados en una gráfica ajustada a estos, haciendo parecer que el sobreajuste es mayor al 10%.

MNIST. Gráfica de precisión	**Precisión de entrenamiento**	**Precisión de validación**	**Época de sobreajuste**	**Sobreajuste de precisión**
Original	100%	90%	< 5	10%
Capas eliminadas	100%	90%	3	10%

Tabla 13.

La precisión de entrenamiento, el sobreajuste, la época de inicio de este y el máximo de la precisión de validación no varían más allá de las pequeñas variaciones intrínsecas del entrenamiento. Sin embargo, las tendencias de la precisión si muestran alguna diferencia interesante, sobre todo el centro de estas que disminuye hasta el 88.5% (aproximadamente).

MNIST. Gráfica de error	**Error de entrenamiento**	**Error de validación**	**Época de sobreajuste**	**Sobreajuste de error**
Original	0.00	0.5	< 5	0.5
Capas eliminadas	0.0038	0.4641	3	0.4603

Tabla 14.

Igual que con los datos de precisión, los datos de error no varían excesivamente. El error de validación máxima mejora ligeramente, 0.04 puntos, y por tanto el sobreajuste también decrece. La verdadera diferencia es en la trayectoria de la curva de validación, pues tras superar el mínimo en la época 17 aumenta significativamente finalizando el entrenamiento alrededor de 0.746 puntos con un máximo relativo en 0.8255p.

Añadir capas sin "Dropout".

Esta modificación consiste en aumentar el número de capas neuronales como podemos observar en el siguiente código:

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=5, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=5, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

El modelo tiene como representación:

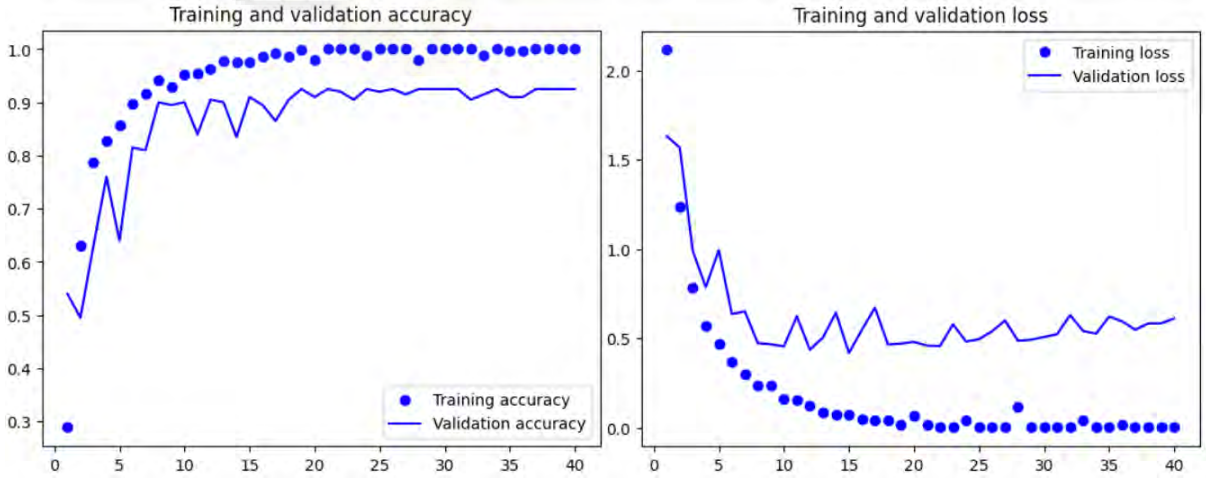

Imagen 7. División en Figuras 13 y 14.

Igual que con el modelo anterior existe una desacople visual. Es innegable que los datos corroboran una mejora en los sobreajustes, pero ni es tan pequeño como se observa en la figura 13 siendo 7.5% ni tan grande como se observa en la figura 14, 0.3135 puntos.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión

Original	100%	90%	< 5	10%
Capas añadidas sin “Dropout”	100%	92.5%	<5	7.5%

Tabla 15.

Dos de los objetivos principales se cumplen, el primero, solo parcialmente, la disminución del sobreajuste. El segundo es mantener la precisión de entrenamiento en un 100%. También se observa un aumento en la curva de la precisión de validación, alcanzando la estabilidad en torno al 92.5% con un máximo a esa misma altura, con algunos mínimos menores.

Gráfica de error	**Error de entrenamiento**	**Error de validación**	**Época de sobreajuste**	**Sobreajuste de error**
Original	0.00	0.5	< 5	0.5
Capas añadidas sin “Dropout”	0.000	0.3135	8	0.3135

Tabla 16.

En el caso del error no sólo se cumplen, al menos parcialmente, dos objetivos sino tres. El error de entrenamiento se mantiene en 0, se retrasa mínimamente el sobreajuste hasta la época 8 y se disminuye este en 0.1865 puntos. Con respecto al error de validación hay que comentar que el valor de la tabla 16 es el mínimo; la curva tiene una tendencia en aumento, no muy pronunciada, con un máximo en 0.6709.

Añadir capas con “dropout”:

El argumento utilizado será 0.5 pues es el estándar.

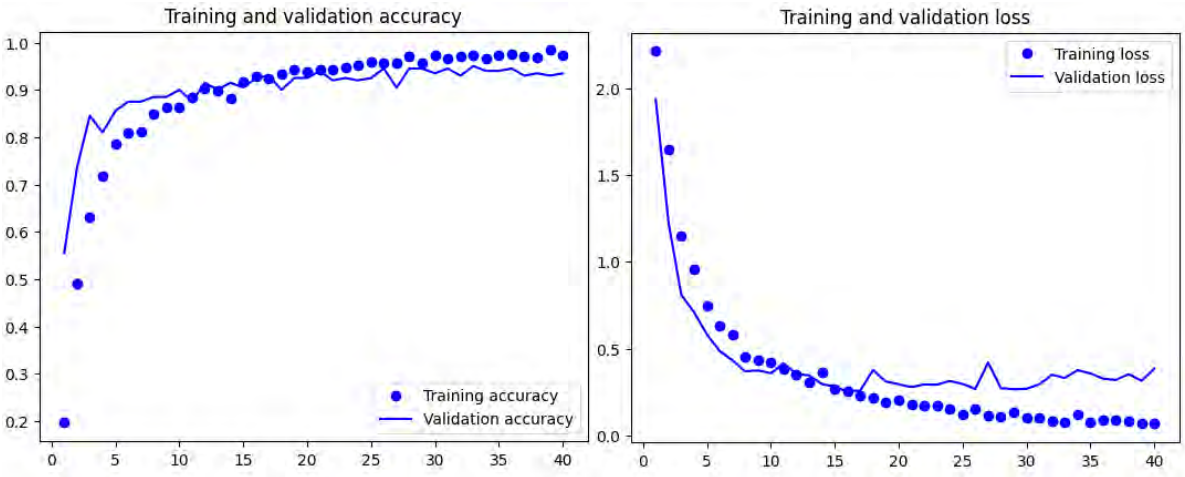

Imagen 8. División en Figura 15 y 16.

Se percibe en la imagen 8 cierto perfeccionamiento de la red convolucional pues el sobreentrenamiento es reducido. Hay que entender, que este rendimiento se debe a una optimización de las curvas de validación y a un empeoramiento de las series de entrenamiento.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
Capas añadidas con “Dropout”	98.37%	95%	>20	3.87%

Tabla 17.

El único punto negativo es el trazado de entrenamiento; ni siquiera su máximo no logra el objetivo (100%) quedando a 1.63%, además, la tendencia de la serie finaliza en el entorno del 97%. El resto de los datos son positivos; el sobreajuste se retrasa en más de 15 épocas y disminuye hasta el 3.87%. La precisión de validación, por su lado, aumenta tanto su máximo (95%) como su trayectoria finalizando alrededor del 94%.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
Capas añadidas con “Dropout”	0.0667	0.2522	>15	0.1855

Tabla 18.

Del mismo modo que en la gráfica 15 se examina que la curva de entrenamiento es el mayor punto deficitario. Pese a que su trayectoria es decreciente, el mínimo comprobado (0.0667) no alcanza el valor 0.00. Otro dato inconveniente es la orientación mínimamente ascendente del error de validación. Su máximo es 0.4202, bastante menor al mínimo original; por ello no es preocupante y sigue siendo un punto positivo. Este refinamiento de la validación reduce el sobreajuste, el cual empieza más de 10 épocas después.

2.4. CUARTA TÉCNICA. ACTIVACIÓN DEL CLASIFICADOR

Otra parte de la ConvNet que se puede modificar es la activación del clasificador. Tras comprobar numerosas opciones como ‘ReLU’ o ‘Leaky ReLU’ las dos únicas funciones útiles

para este servicio son ‘sigmoid’ y ‘ softmax’. Así pues, veamos los resultados de ‘sigmoid’:

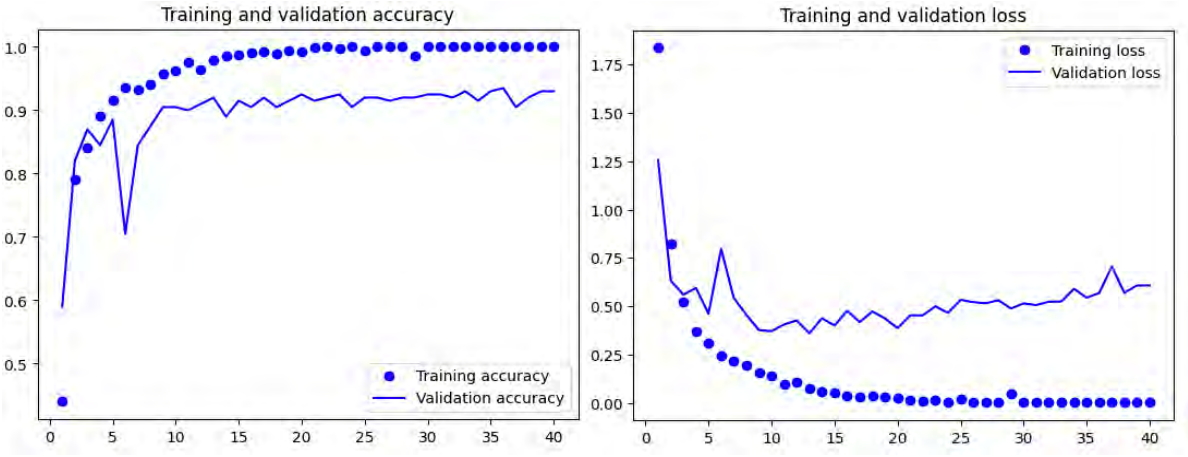

Imagen 9. División en Figura 17 y 18.

De la misma manera que la función ‘softmax’ produce un clasificador correcto, pero con gráficas de aprendizaje que exhiben sobreajuste, la función ‘sigmoid’ presenta una situación muy similar hasta el punto en el cual se puede entender que ambas opciones son iguales.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
‘Softmax’	100%	90%	< 5	10%
‘Sigmoid’	100%	93%	<5	7%

Tabla 19.

La única diferencia en cuanto a la precisión radica en la validación siendo ‘sigmoid’ un 3% superior.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
‘Softmax’	0.00	0.5	< 5	0.5
‘Sigmoid’	0.00	0.6	<5	0.6

Tabla 20.

La única diferencia respecto al error radica en la validación siendo el de ‘sigmoid’ 0.1 puntos más elevado.

2.5. QUINTA TÉCNICA. AUMENTO DE DATOS

El aumento de datos es una técnica habitual para evitar el sobreajuste. La técnica se basa en multiplicar el número de imágenes de entrada sin repetirlas. Para ello se editan de forma aleatoria; primero se le puede aplicar un espejo; segundo se puede rotar; finalmente se le puede aplicar un zoom.

Debido a las características de los datos de entrada esta técnica se ha de suavizar, eliminando la primera función y limitando el rango de las otras dos.

```
data_augmentation = keras.Sequential(
    [
        layers.RandomRotation(0.05),
        layers.RandomZoom(0.5),
    ])
```

El resultado de aplicar esta técnica es:

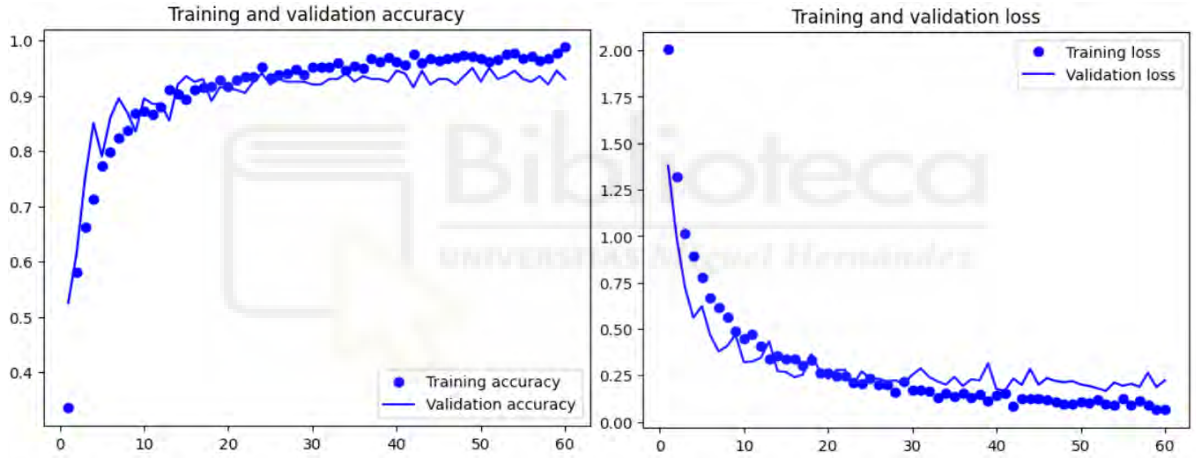

Imagen 10. División en Figura 18 y 19.

El principal y primer objetivo de las técnicas y estrategias aplicadas, reducir el sobreajuste, se cumple en ambas gráficas. En parte por fortalecer las curvas de validación, en parte por degradar las series de entrenamiento. Otro de los objetivos cumplidos es el retraso del sobreentrenamiento conseguido en toda la imagen.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
Aumento de datos	98.75%	95%	>20	3.75%

Tabla 21.

El sobreajuste de precisión se reduce en más de un 6% hasta el 3.75%. Este hecho más el retraso del inicio del sobreajuste más allá de la época 20 se debe a dos razones, el aumento de la precisión máxima de validación y una disminución de la precisión de entrenamiento. La tendencia de la precisión de entrenamiento se mantiene cerca del 97%, a 3% del objetivo 100%. Por otro lado, la trayectoria de la precisión de validación se aproxima al 94.5%, cada vez más cerca de la meta.

MNIST. Gráfica de error	**Error de entrenamiento**	**Error de validación**	**Época de sobreajuste**	**Sobreajuste de error**
Original	0.00	0.5	< 5	0.5
Aumento de datos	0.0645	0.1670	>25	0.1025

Tabla 22.

El sobreajuste se reduce en casi 4 décimas de puntos dejándolo en 0.1025p, también se retrasa el inicio del sobreentrenamiento hasta más allá de la época 25, cumpliendo así dos de las metas. Otro de los propósitos cumplido es el refinamiento del error de validación, cuyo mínimo vale 0.1670 puntos. No obstante, su tendencia finaliza en torno a 0.2 con un máximo en 0.2875 puntos. La única serie que incumple los objetivos es el error de entrenamiento, la cual tiene como mínimo 0.0645 puntos, pero con un trazado decreciente.

Añadir la función .randomflip() al aumento de datos.

A continuación, para asegurar que es un error utilizar la función .RandomFlip(), voy a analizar sus datos, tanto con el argumento “horizontal”, como con el argumento “vertical”.

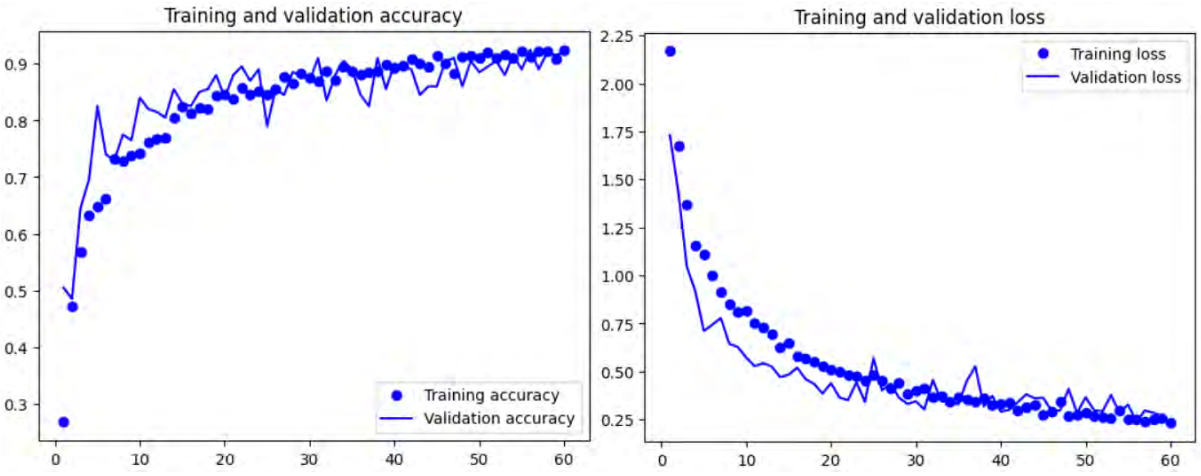

Imagen 11. División en Figuras 21 y 22.

La imagen 11 representa los datos de la red incluyendo el argumento “horizontal” en la función .RandomFlip(). Se advierte varias cosas; por un lado, el sobreentrenamiento se reduce al máximo hasta el punto de no observarse, sin embargo, eso se debe en gran medida al gran deterioro de las curvas de entrenamiento. Por otro lado, las gráficas de validación son menos regulares, con forma de dientes de sierra, debido a que las nuevas imágenes se diferencian en exceso entre sí.

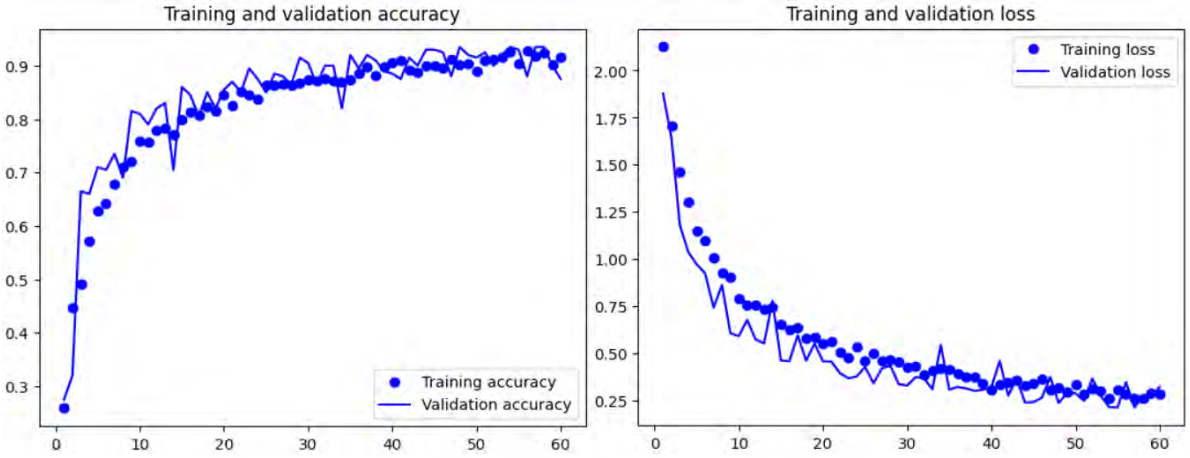

Imagen 12. División en Figuras 23 y 24.

De forma similar a la anterior, la imagen 12 representa los datos de la ConvNet incluyendo el argumento “vertical” en la función .RandomFlip(). Sus principales ventajas e inconvenientes son iguales, el sobreajuste se erradica a cambio de una gran reducción de las series de entrenamiento y las gráficas de validación son sumamente irregulares, con forma de dientes de sierra. Además, en este caso, el perfil de las series de entrenamiento también tiene forma de dientes de sierra.

Alteración del parámetro de la función .RandomRotation() en el aumento de datos.

La segunda función que se puede editar, .RandomRotation() refiere al rango de rotación que se puede aplicar. Así pues, se aumentará y disminuirá el hiperparámetro para variar el rango de rotación. Las comparativas se realizan directamente frente a las tasas del aumento de datos.

Parte 1: Rango de giro menor

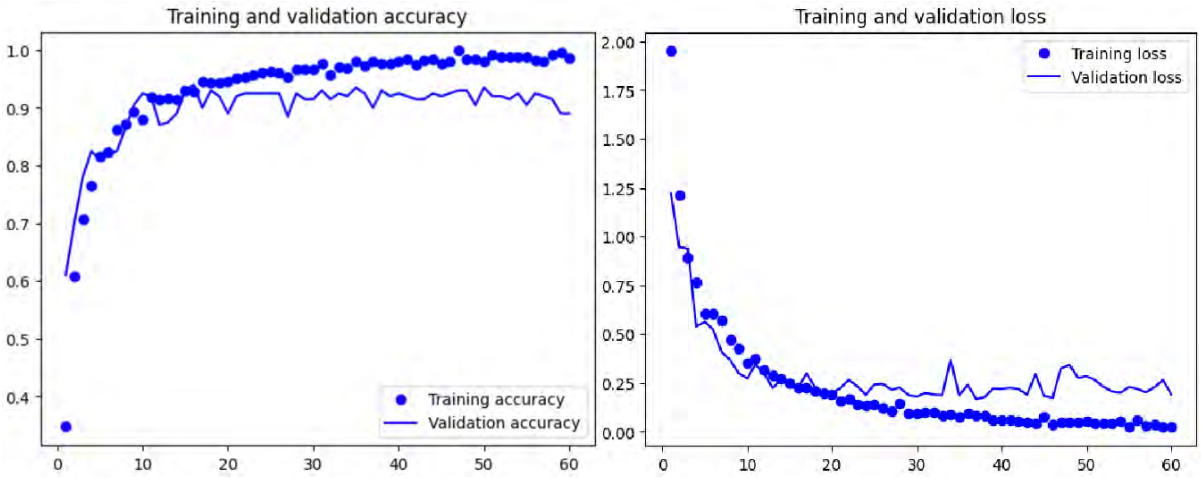

Imagen 13. División en Figuras 25 y 26.

Visualmente aparenta una gran mejora, sobre todo teniendo en cuenta la edición anterior. En ese aspecto, tanto la figura 25 como la 26 muestran una red superior pues sus trayectorias no zigzaguean con tanta intensidad. Respecto a la red original, se produce una gran reducción del sobreajuste con solamente una pequeña pérdida en las curvas de entrenamiento.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Aumento de datos	98.75%	95%	>20	3.75%
Rotación máxima = 0.005	99.5%	93.5%	<20	6.00%

Tabla 23.

La tabla 23 compara los resultados en precisión del aumento de datos original y los resultados de aplicar el aumento de datos con una rotación máxima menor. El sobreajuste comienza antes y aumenta en dimensión, debido a una reducción de la curva de validación en 1.5% y a un aumento de la representación de entrenamiento hasta el 99.5%. Así mismo, mientras que la trayectoria final de la precisión de entrenamiento se mantiene en el entorno del 98.5%, la trayectoria de validación no logra alejarse del ámbito del 92%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Aumento de datos	0.0645	0.1670	>25	0.1025
Rotación	0.0250	0.1672	<20	0.1422

máxima = 0.005

Tabla 24.

En relación con el error, el análisis comparativo entre los resultados de aplicar el aumento de datos original y el aumento de datos con una rotación máxima menor. El sobreajuste aumenta en 0.4 puntos debido mayoritariamente a un refinamiento del error de entrenamiento. Acerca de la trayectoria del error de validación es la de aumentar, pero no a gran velocidad manteniéndose alrededor de 0.22 puntos.

Parte 2: Rango mayor

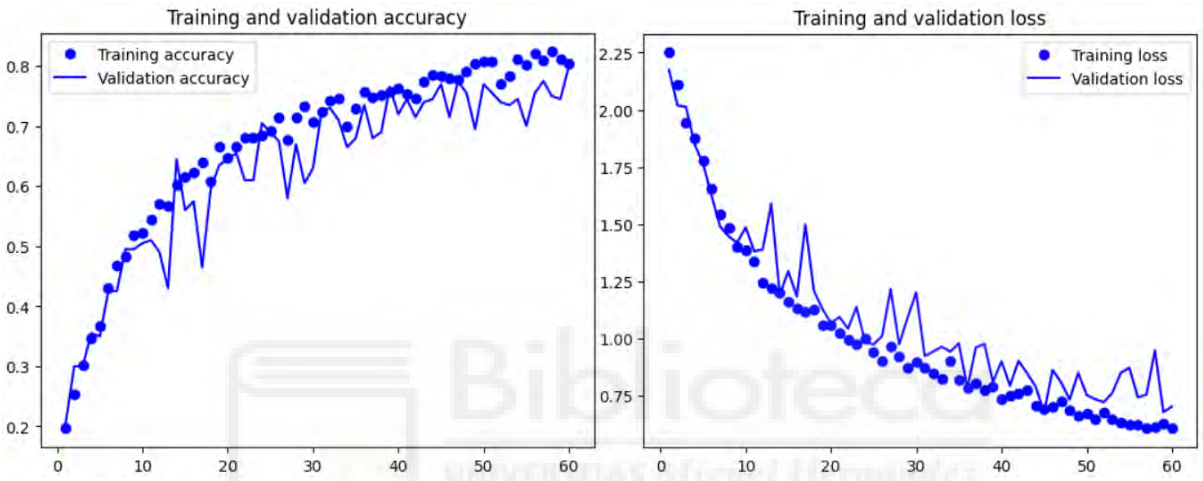

Imagen 14. División en Figura 27 y 28.

Un rango de rotación tan elevado diferencia excesivamente las figuras con una misma etiqueta, esto produce la imposibilidad de la red de encontrar patrones dificultando el entrenamiento. Este problema es el mismo que con la función ".flip" el cual genera una gráfica zigzagueante con forma de dientes de sierra. El escenario reproducido en la imagen 14 no es factible pese a conseguir eliminar el sobreajuste en las primeras 60 épocas debido a la absurda reducción de las curvas en ambas gráficas.

En la gráfica 27 se muestra que la precisión no logra superar el 80% pareciendo que se estabiliza en ese entorno. Así mismo, en la gráfica 28 se declara un mínimo del error en 0.6 puntos en tendencia decreciente.

Para conseguir aproximar los valores de entrenamiento a imágenes anteriores no solo es necesario producir el sobreajuste, sino aumentar desmesuradamente el tiempo de entrenamiento superando las 200 épocas; por tanto, alargando y complicando la red y su aplicación.

Alteración del parámetro de la función .randomzoom() en el aumento de datos.

La última función que se puede editar, .RandomZoom() refiere al rango de zoom aplicable. Así pues, se aumentará y disminuirá el hiperparámetro para variar este rango. Las comparativas se realizan directamente frente a las tasas del aumento de datos.

Parte 1: Rango menor

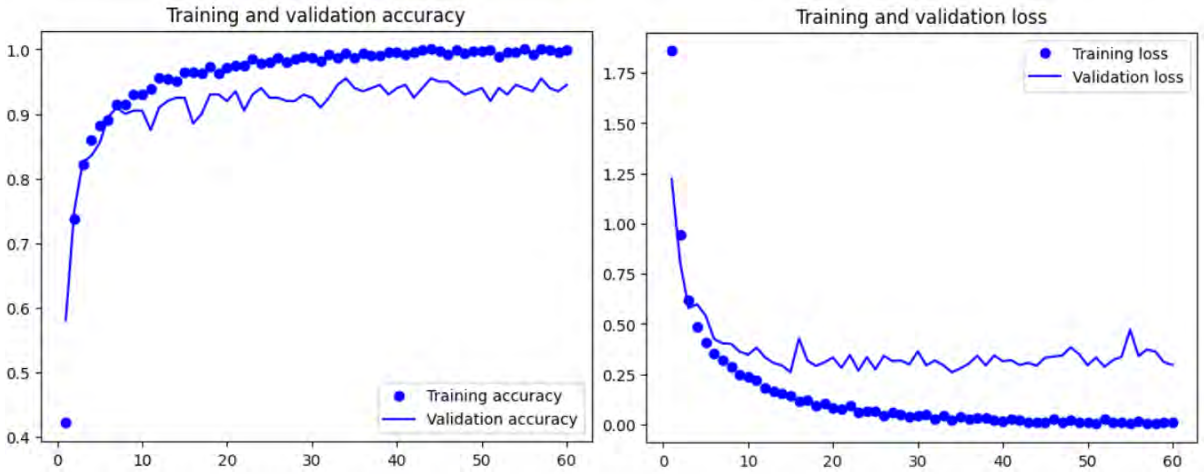

Imagen 15. División en Figura 29 y 30.

Hay dos puntos principales en las figuras. El primero es el gran y temprano sobreajuste, el segundo es que se logra el objetivo de entrenamiento. Además, otro punto positivo, no se observa una forma de dientes de sierra.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Aumento de datos.	98.75%	95%	>20	3.75%
Zoom máximo = 0.05	100%	95.5%	<10	4.5%

Tabla 25.

Como ya he indicado, la precisión de entrenamiento logra alcanzar el objetivo del 100%. Dado que la precisión de validación aumenta en menor medida que la de entrenamiento el sobreajuste aumenta en un 0.75%. Por otro lado, la tendencia de la curva de validación aumenta finalizando en la cercanía del 94.5%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error

Aumento de datos	0.0645	0.1670	>25	0.1025
Zoom máximo = 0.05	0.0021	0.2594	<10	0.2573

Tabla 26

El sobreentrenamiento también se adelanta en la figura 30, también aumenta en más de 0.15 puntos. En este caso no se debe sólo a la mejoría del entrenamiento cuya trayectoria se aproxima a 0 según se entrena, sino también al error de validación cuya tendencia ascendente alcanza el entorno de 0.33 puntos al finalizar el entrenamiento propuesto.

Pese a aumentar el error, si se quiere conseguir la mayor precisión esta configuración es un buen comienzo.

Parte 2: Rango mayor

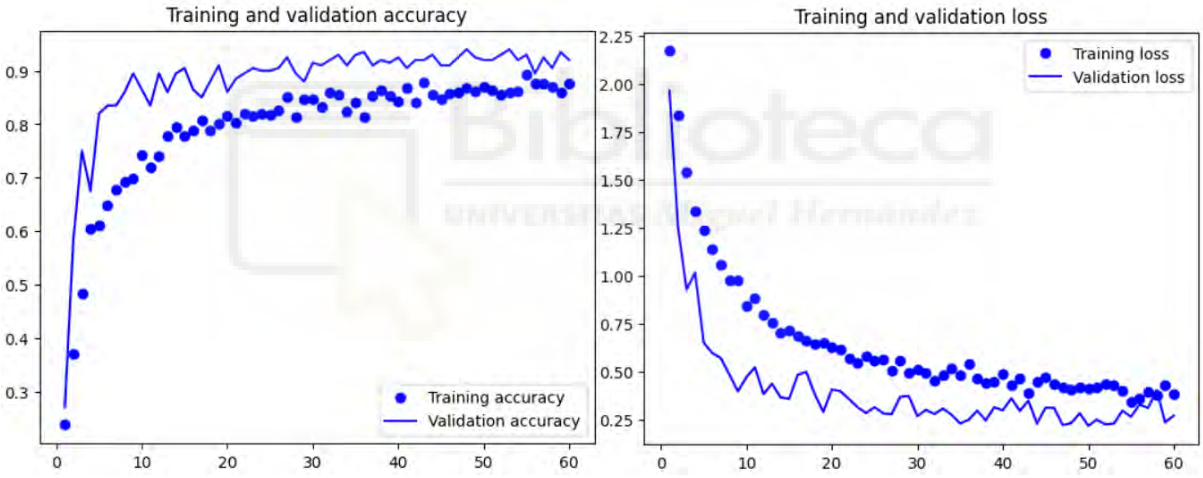

Imagen 16. División en Figuras 31 y 32.

El principal objetivo, evitar el sobreajuste sucede. Los otros objetivos, a saber, que las diferentes curvas de precisión valgan 100% y que la puntuación final de las curvas de error resulte en 0 no se logran. La siguiente tabla no contiene valores máximos, sólo tendencias.

Imagen 16	**Precisión de entrenamiento**	**Precisión de validación**	**Error de entrenamiento**	**Error de validación**
Aumento de datos	98.75%	95%	0.0645	0.1670
Zoom máximo = 1	95.5%	95.5%	0.2	0.2

Tabla 27.

Los datos de precisión declaran una ligera mejora, pero los datos de error un ligero empeoramiento.

2.6. SEXTA TÉCNICA. REGULARIZACIÓN

La regularización es una técnica que agrega restricciones adicionales durante el entrenamiento de la red. Sus objetivos son evitar el sobreentrenamiento, mejorar el rendimiento y generalizar el modelo. Las dos técnicas más comunes (y las utilizadas en este análisis) son:

- Regulación L1: Agrega la suma de los valores absolutos de los pesos del modelo a la función de pérdida.
- Regularización L2: Agrega la suma de los cuadrados de los pesos del modelo a la función de pérdida.

Ambas regularizaciones se agregan a las capas convolucionales (Conv2D) y a la capa densamente conectada (Dense), y pueden ser combinadas entre ellas y con otras técnicas como "dropout" o "data augmentation".

2.6.1. REGULARIZACIÓN L1

A continuación, se aplicará sobre la red convolucional la regularización L1 con diferentes hiperparámetros λ con el fin de analizar las diferentes variantes.

Variante 1: λ = 0.01 o superior

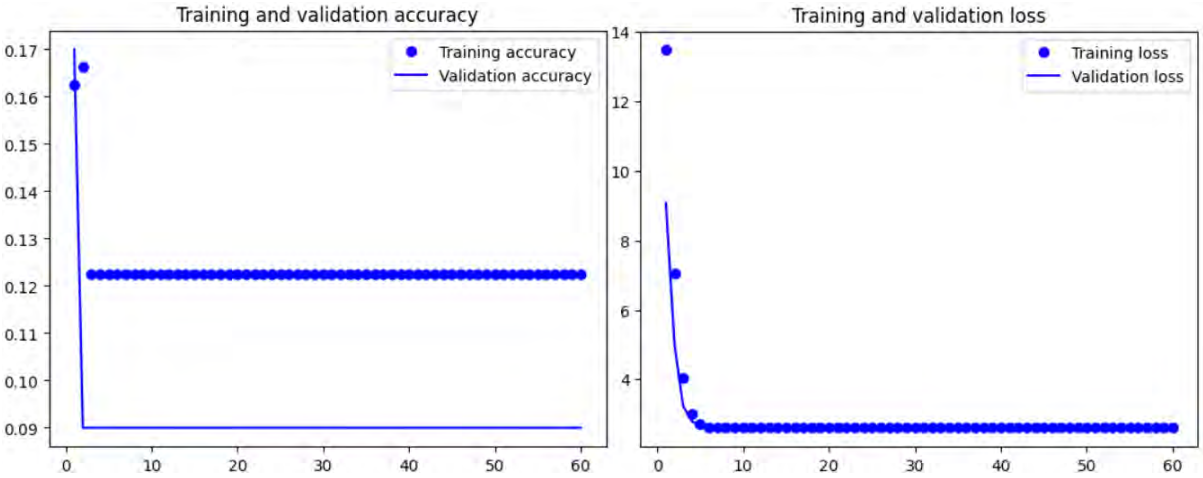

Imagen 17. División en Figuras 33 y 34.

Cuando el hiperparámetro λ es demasiado grande, la restricción impuesta adicionalmente también lo es simplificando la red y obstaculizando el entrenamiento hasta tal punto que no se produce un aprendizaje.

Variante 2: λ = 0.001

Los primeros resultados prácticos de la red con regularización L1 se consiguen al aplicar λ con magnitud 0.001.

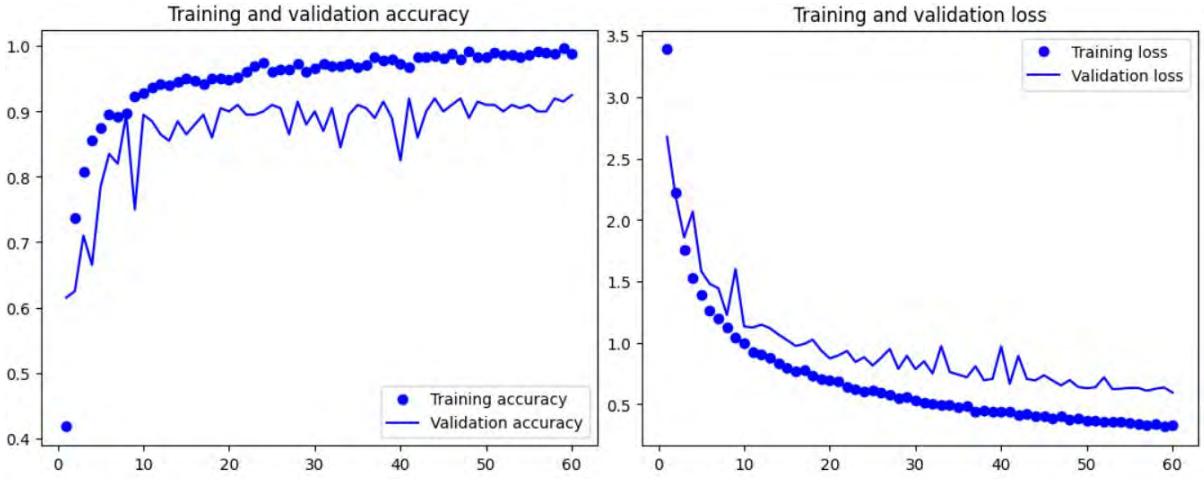

Imagen 18. División en Figuras 35 y 36

Pese a producir resultados, la restricción sigue obstaculizando el entrenamiento produciendo un sobreajuste casi inmediato y un rendimiento semejante al original.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
Regularización L1: λ = 0.001	99.62%	92.5%	<5	7.12%

Tabla 28.

El sobreajuste disminuye en casi un 3% respecto al original debido en prácticamente su totalidad a la curva de precisión. Por otra parte, la tendencia de la precisión de validación se estabiliza en el entorno del 91% con un máximo final del 92.5%. Así mismo la orientación de la precisión de entrenamiento se mantiene cercana a 99%.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
Regularización L1: λ = 0.001	0.3172	0.5954	<5	0.2782

Tabla 29.

Pese a que el sobreajuste mejora, empeora tanto el error de validación, casi 0.1 puntos, como el error de entrenamiento en más de 0.3. Al menos la tendencia en ambas series es hacia 0.

Variante 3: $\lambda = 0.0001$ o inferior

A partir de esta magnitud el efecto de las restricciones se atenúa profundamente, la función de pérdida no varía pues la función que se ha de sumar siempre vale aproximadamente 0.

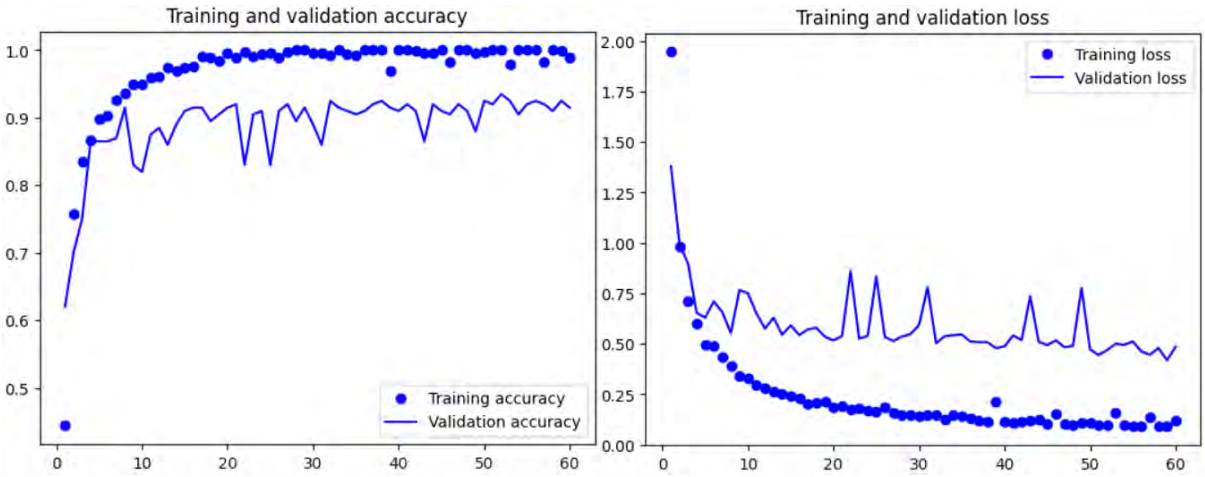

Imagen 19. División en Figuras 37 y 38.

Visualmente puede parecer que hay un gran aumento del sobreajuste en la imagen 19 respecto a la imagen 18 pero no es del todo así. Si se produce un pequeño aumento, pero exageradamente elevado. Las curvas que sí han mejorado encarecidamente, aproximándose a su objetivo, son las de entrenamiento.

MNIST. Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
Regularización L1: $\lambda = 0.0001$	100%	93.5%	<10	6.5%

Tabla 30.

El sobreajuste disminuye un 3.5% respecto al original debido al máximo 93.5% de la curva de precisión. Esta curva se estabiliza en torno al 92% desde la época 10 aproximadamente con algunos mínimos relativos.

MNIST. Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error

Original	0.00	0.5	< 5	0.5
Regularización L1: $\lambda = 0.001$	0.0891	0.4184	<5	0.3293

Tabla 31.

En lo relativo al error, el sobreajuste mejora en casi 0.2 puntos, la mitad por un empeoramiento en el error de entrenamiento y la otra mitad debido a una mejoría en el error de validación. Igualmente, el decrecimiento de la curva de validación no es suficiente.

2.6.2. Regularización L2

A continuación, se aplicará sobre la red convolucional la regularización L2 con diferentes hiperparámetros λ con el fin de analizar las diferentes variantes.

Variante 1: $\lambda = 1$ o superior

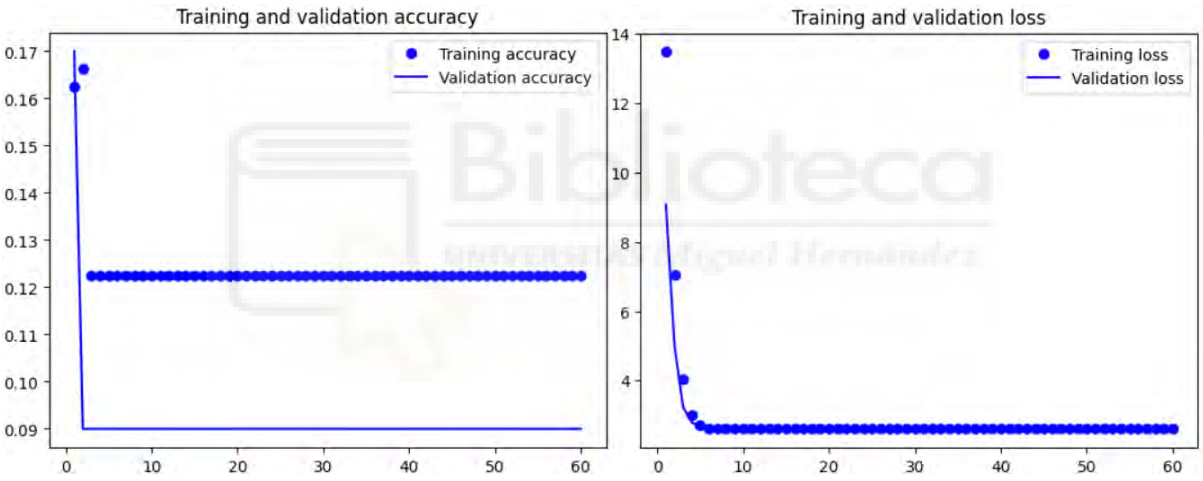

Imagen 20. División en Figuras 39 y 40.

Cuando el hiperparámetro λ es demasiado grande, la restricción impuesta adicionalmente también lo es simplificando la red y obstaculizando el entrenamiento hasta tal punto que no se produce un aprendizaje.

Variante 2: λ = 0.1

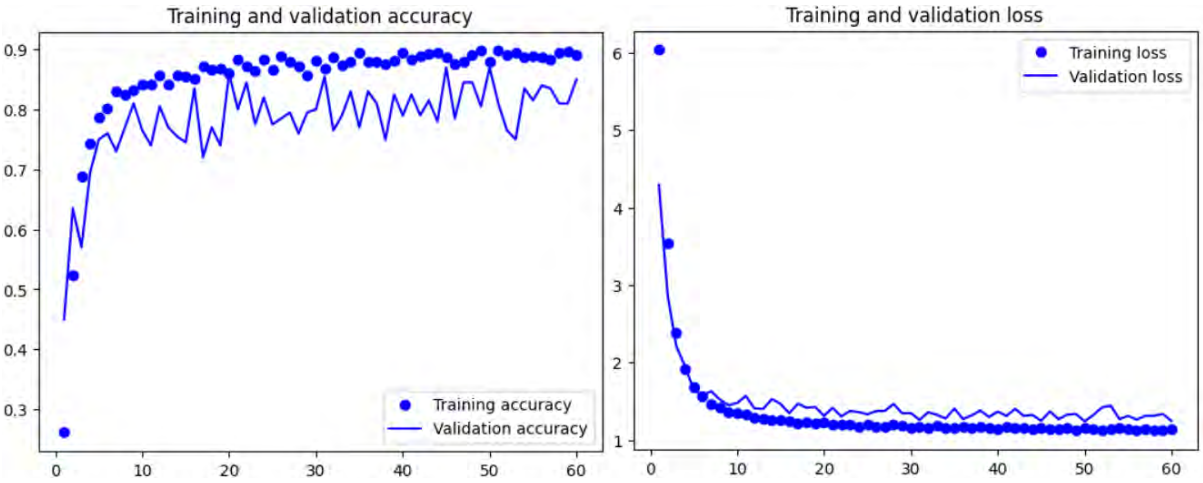

Imagen 21. División en Figuras 41 y 42.

La imagen 21 es sumamente interesante, ambas figuras tienen algo interesante a comentar. La figura 41 tiene forma de sierra. Las curvas de ambas figuras descienden de forma abrumadora. Por estos dos motivos, y pese a reducir el sobreajuste de la figura 42, no se puede utilizar este hiperparámetro.

Variante 3: λ = 0.01

Los primeros resultados prácticos de la red con regularización L2 se consiguen al aplicar λ con magnitud 0.01.

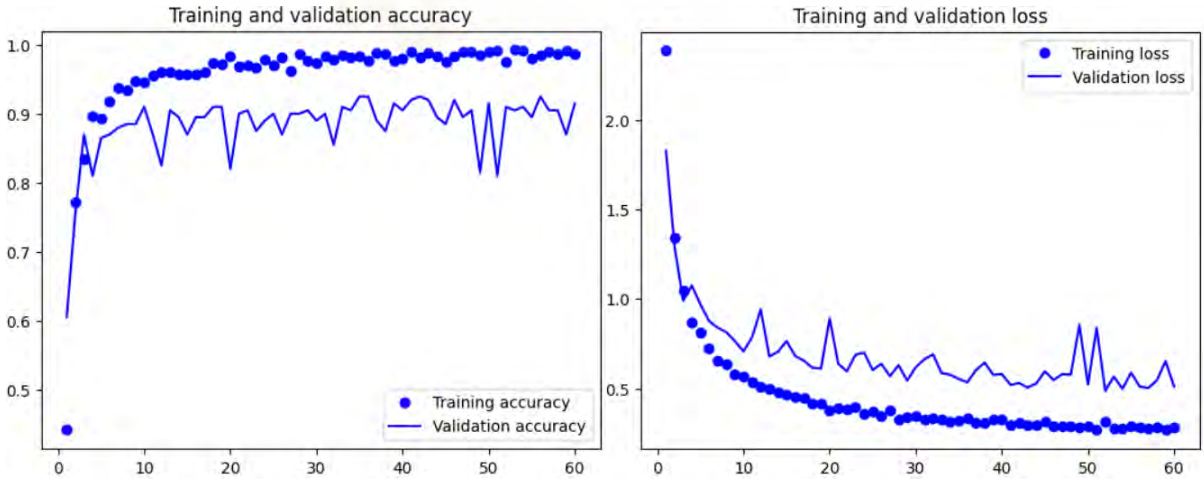

Imagen 22. División en Figuras 43 y 44.

Aunque se obtienen resultados, la restricción sigue dificultando el entrenamiento generando un sobreajuste casi inmediato y un rendimiento ligeramente superior al original.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión

Original	100%	90%	< 5	10%
Regularización L2: $\lambda = 0.01$	99.25%	92.5%	<5	6.75%

Tabla 32.

El sobreajuste disminuye en 3.25% principalmente gracias a la mejora de la curva de validación cuya tendencia finaliza en el ámbito del 91.5%. Por otra parte, la serie de entrenamiento empeora un 0.75% incumpliendo uno de los objetivos, contribuyendo a reducir el sobreentrenamiento en dicho porcentaje. El sobreajuste sigue produciéndose demasiado pronto.

Gráfica de error	**Error de entrenamiento**	**Error de validación**	**Época de sobreajuste**	**Sobreajuste de error**
Original	0.00	0.5	< 5	0.5
Regularización L2: $\lambda = 0.01$	0.2672	0.4867	<5	0.2195

Tabla 33

La reducción del sobreajuste no viene dada por el error de validación, cuya tendencia es similar a la original en la cercanía de 0.5215 puntos; sino a la curva de entrenamiento cuya trayectoria se mantiene en el ámbito de los 0.275 puntos. El sobreentrenamiento sigue produciéndose prematuramente.

Variante 4: $\lambda = 0.001$ o inferior

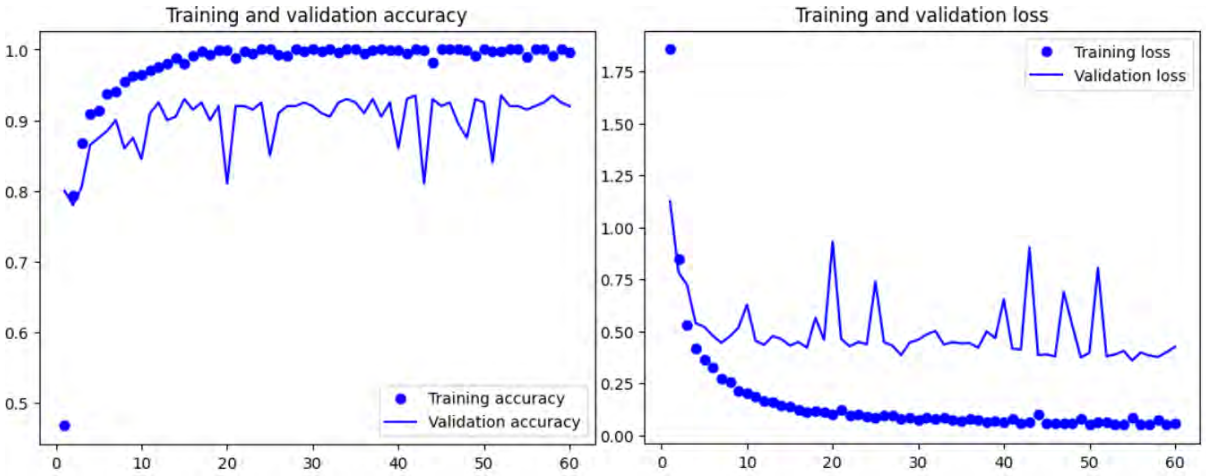

Imagen 23. División en Figuras 45 y 46.

Desde una perspectiva visual, puede parecer que hay un aumento significativo del sobreajuste en la imagen 23 en comparación con la imagen 22, pero esto no es completamente cierto. Si bien se observa un ligero incremento, no es exageradamente elevado. Sin embargo, las curvas

de entrenamiento han mejorado considerablemente, acercándose significativamente a su objetivo.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
Regularización L2: $\lambda = 0.001$	100%	93.5%	<10	6.5%

Tabla 34.

El sobreajuste se reduce en un 3.5% en comparación con el resultado original, gracias al máximo del 93.5% en la curva de precisión. A partir de aproximadamente la época 10, esta curva se estabiliza alrededor del 92%, con algunos mínimos relativos.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
Regularización L1: $\lambda = 0.001$	0.0891	0.4184	<5	0.3293

Tabla 35.

Con respecto al error, el sobreajuste mejora en casi 0.2 puntos, dividido en partes iguales entre un empeoramiento en el error de entrenamiento y una mejora en el error de validación. Sin embargo, el decrecimiento de la curva de validación no es suficiente para evitar completamente el sobreajuste.

3. APLICACIÓN DE ESTRATEGIAS CONTRA EL SOBREAJUSTE EN LA RED NEURONAL DE MNIST

Estas estrategias son combinaciones de dos o más técnicas Las diferentes estrategias pueden tener diferentes variaciones, pero sólo se incluirán las principales o más representativas para esta red.

3.1. PRIMERA ESTRATEGIA. AUMENTO DE DATOS Y “DROPOUT”

La primera estrategia surge de la combinación de dos de las técnicas más provechosas, el aumento de datos y el “dropout”. Además, se le añadirá la técnica “early stopping” con objeto de reducir al máximo la capacidad de cálculo. Los parámetros del aumento de datos serán: sin

espejo, rotación máxima de 0.05 y zoom máximo de 0.5.

Variante 1: Aumento de datos y .dropout(0.5)

Parte 1: Sin "Early stopping".

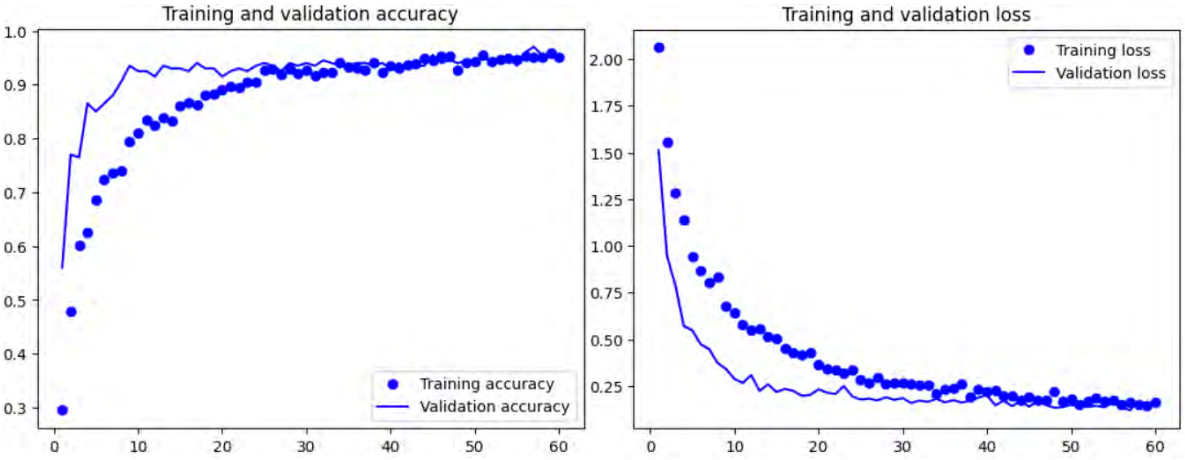

Imagen 24. División en Figuras 47 y 48.

La complejidad del sistema y el aumento de datos dificulta el entrenamiento inicial, pero esto mismo permite una buena generalización consiguiendo (al principio) mejores datos de validación, retrasando o eliminando el sobreajuste.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
A.D. y dropout(0.5)	95.88%	96%	>60	- - -

Tabla 36.

Como ya he indicado, el sobreajuste ha sido suprimido en las primeras 60 épocas. No indico que se ha eliminado totalmente pues la trayectoria ascendente de la curva de entrenamiento indica que más adelante sí se produciría. Así mismo, esta estrategia logra el hasta ahora mejor aproximamiento a la precisión de validación absoluta superando en un 6% a la original.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
A.D. y dropout(0.5)	0.1455	0.1475	>55	0.0020

Tabla 37.

El verdadero mínimo de la serie error de validación es 0.1441, pero se escoge 0.1475 pues es el mínimo tras el inicio del sobreajuste. Como el sobreentrenamiento se retrasa tanto, el producido en esas pequeñas épocas es enano. En cuanto a las tendencias finales de las curvas, las diferencias son muy reducidas. La trayectoria de entrenamiento es ligeramente descendente acabando en el entorno 0.16 puntos. Por otro lado, la tendencia de validación es estable desde la época 25 en el ámbito de 0.17 puntos.

Parte 2: Con "early stopping"

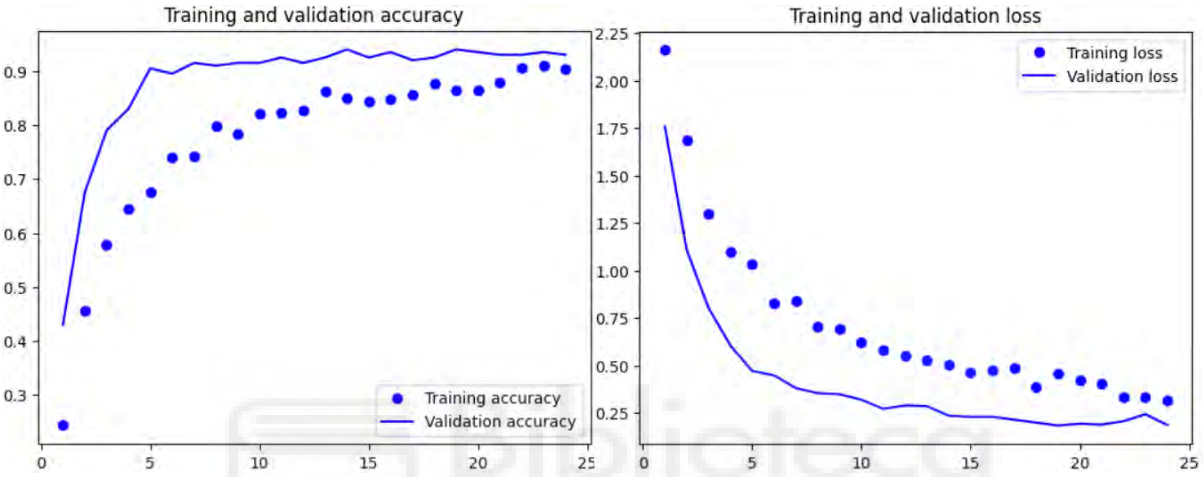

Imagen 25. División en Figura 49 y 50.

No se produce sobreajuste, sin embargo, no se consigue ni tanta precisión de validación, atenuación en un 2% hasta el 94%; ni tan poco error de validación, error mínimo de 0.186 puntos.

Variante 2: Aumento de datos y .dropout(0.75)

Parte 1: Sin "Early stopping"

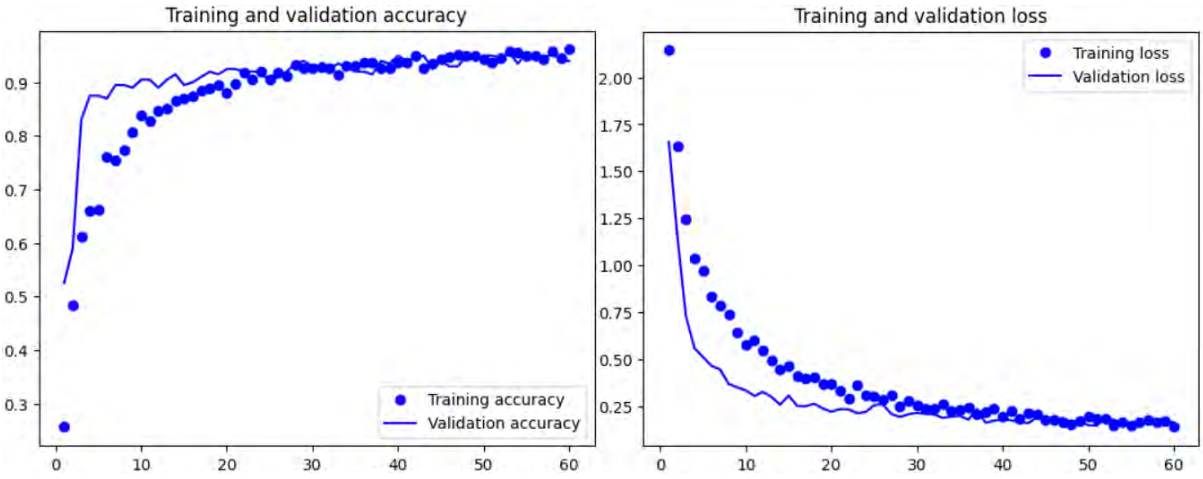

Imagen 26. División en Figuras 51 y 52.

Igual que en la variante anterior la complejidad del sistema y el aumento de datos dificulta el entrenamiento inicial, pero esto mismo permite una buena generalización consiguiendo (al principio) mejores datos de validación, retrasando o eliminando el sobreajuste.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
A.D. y dropout(0.75)	96.13%	95.5%	>60	- - -

Tabla 38.

Puede parecer que sí hay sobreentrenamiento; pero ambos son máximos individuales, teniendo las dos curvas la misma tendencia final, 95%. Cierto es que, de haber seguido entrenando la serie de validación se habría mantenido y la de entrenamiento habría aumentado.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
A.D. y dropout(0.75)	0.1384	0.1424	>60	- - -

Tabla 39.

Siendo el mismo caso que en la precisión, en el error no se produce sobreentrenamiento. Las tendencias se producen en torno a 0.16 puntos, pero con una tendencia del error de entrenamiento decreciente.

Parte 2: Con "Early stopping"

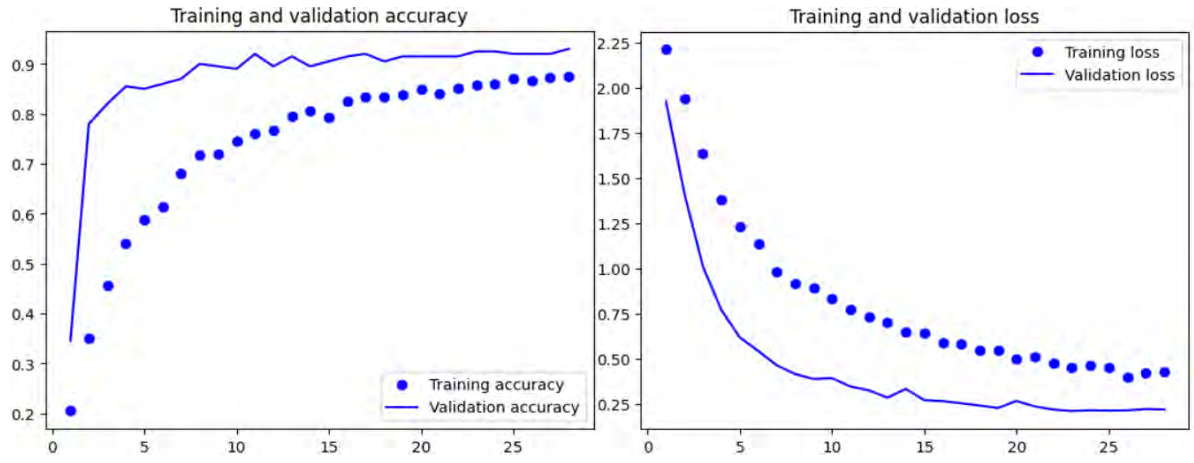

Imagen 27. División en Figuras 53 y 54.

No se produce sobreajuste, sin embargo, no se consigue ni tanta precisión de validación, atenuación en un 2.5% hasta el 93%; ni tan poco error de validación, error mínimo de 0.2093 puntos.

3.2. SEGUNDA ESTRATEGIA. AUMENTO DE DATOS Y REGULARIZACIÓN

La segunda estrategia surge de la combinación de dos de las técnicas más provechosas, el aumento de datos y las técnicas de regularización. Además, se le puede añadir la técnica "early stopping". Los parámetros del aumento de datos serán: espejo horizontal, rotación máxima de 0.1 y reescalado máximo de 0.2.

Variante 1: Regularización L1

Parte 1: Sin "early stopping"

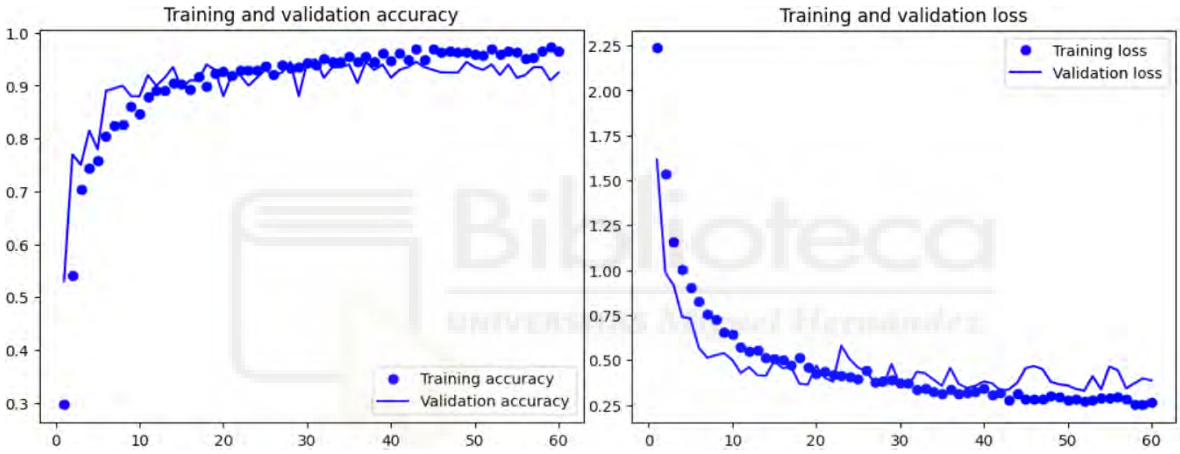

Imagen 28. División en Figuras 55 y 56.

Igual que en la variante anterior la complejidad del sistema y el aumento de datos dificulta el entrenamiento inicial, pero en este caso el sobreajuste se produce poco después de la época 25.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
A.D. y L1	97.25%	94%	>40	3.25%

Tabla 40.

El sobreajuste se reduce radicalmente hasta un 3.25%. En parte debido a la mejora en un 4% de la curva de validación y en parte al empeoramiento de la serie de entrenamiento (2.75%). Indicar que la trayectoria de la curva de validación se estabiliza en un rango próximo a 92.5%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error

Original	0.00	0.5	< 5	0.5
A.D. y L1	0.2503	0.3362	>25	0.0859

Tabla 41.

Prácticamente se cancela el sobreajuste de error, pero en gran parte se debe a la deficiente curva de entrenamiento. En cuanto a tendencias, el sobreajuste es de 0.1 debido a que la curva de entrenamiento se mantiene próximo a 0.27 puntos y la serie de validación en el entorno de 0.37 puntos.

Parte 2: Con "early stopping"

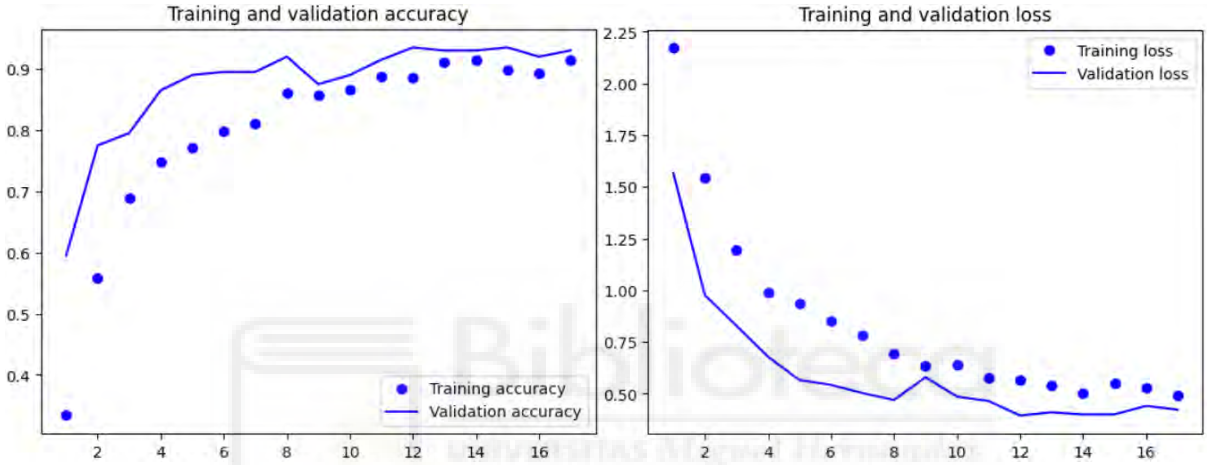

Imagen 29. División en Figuras 57 y 58.

El objetivo al aplicar "early stopping" es eliminar el sobreentrenamiento y reducir el esfuerzo computacional del entrenamiento sin perder competitividad. En este caso; en cuanto a la precisión no hay problema, sin embargo, en el error si se empeora no disminuye de 0.4 puntos.

Variante 2: Regularización L2

Parte 1: Sin "early stopping"

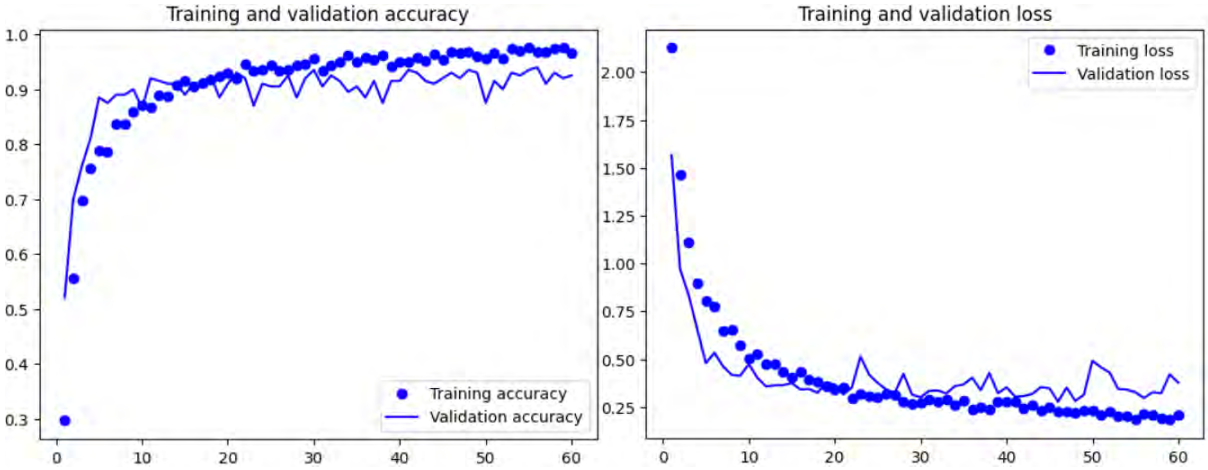

Imagen 30. División en Figuras 59 y 60.

De manera similar a las variantes previas, la complejidad del sistema y el aumento de datos dificultan el entrenamiento inicial. Sin embargo, el cruce de curvas e inicio del sobreentrenamiento se produce mucho antes que en otras variantes sobre todo en la figura z1 (época 20 aproximadamente).

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	90%	< 5	10%
A.D. y L2	97.5%	94%	<25	3.5%

Tabla 42.

Los datos de la tabla 42 y de la tabla 40 son muy similares, salvo por el adelanto del inicio del sobreajuste. Así pues, el análisis es idéntico.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.00	0.5	< 5	0.5
A.D. y L2	0.1821	0.2774	>25	0.0953

Tabla 43.

Respecto a los datos de error, se puede observar una gran similitud entre estos y los representados en la tabla 41 y en la figura 60 con un desfase de -0.1. Esto significa una aceptable mejora, un sobreajuste de error cercano a 0.1 y el inicio del sobreentrenamiento previo a la época 25.

Parte 2: Con "early stopping"

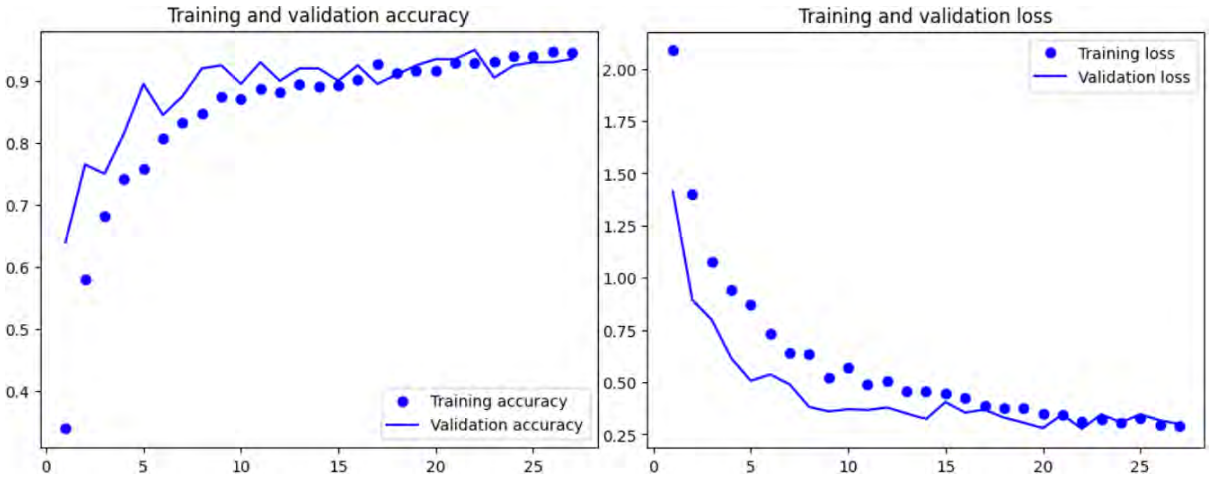

Imagen 31. División en Figuras 61 y 62.

Esta aplicación de “early stopping” sí es provechosa, consigue por una parte evitar el sobreajuste y por otra parte mejorar algunas competencias, especialmente las de validación alcanzando el 95% de precisión y disminuyendo el error hasta 0.2766 puntos (tabla 44).

MNIST. Gráfica mixta	Precisión de entrenamiento	Precisión de validación	Error de entrenamiento	Error de validación
Original	100%	90%	0.00	0.5
A.D. y L2	94.75%	95%	0.2921	0.2766

Tabla 44.

3.3. TERCERA ESTRATEGIA. AUMENTO DE DATOS, REGULARIZACIÓN Y “DROPOUT”

Esta estrategia agrupa las principales técnicas contra el sobreajuste. Conociendo los resultados de las diferentes técnicas que participan en esta estrategia se utilizará una única tabla mixta en cada variante y se comentará la idoneidad de utilizar “early stopping”.

Variante 1: Regularización L1, función .Dropout(0.5)

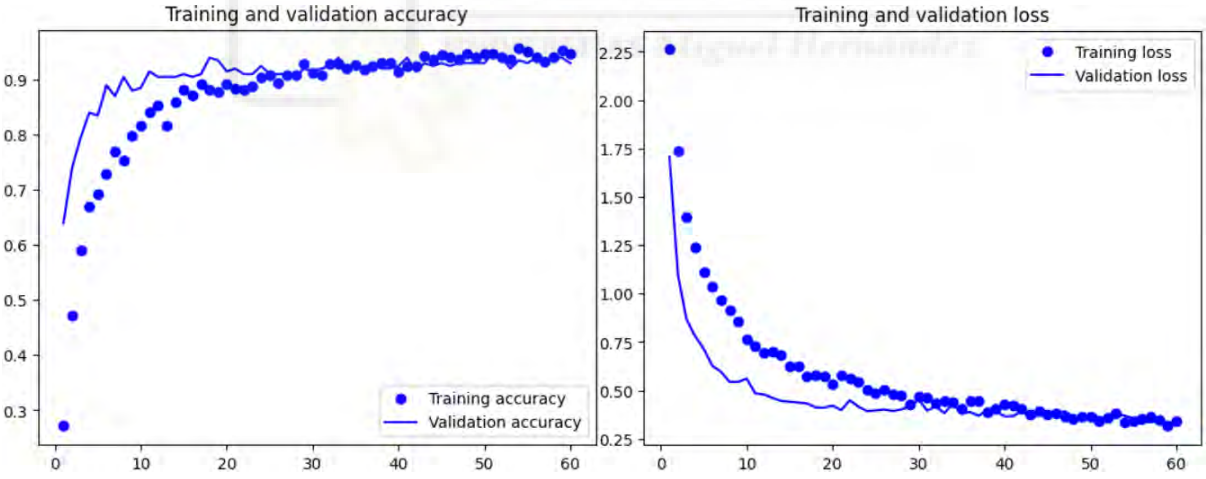

Imagen 32. División en Figuras 63 y 64.

El dato fácilmente extraíble de la imagen 32 es la eliminación absoluta del sobreajuste. Lamentablemente en la tabla 45 se puede observar que, en el caso del error, no se debe precisamente a que la curva de validación se aproxima certeramente a su objetivo, sin embargo, la precisión de validación sí logra un buen 96%.

MNIST. Gráfica mixta	Precisión de entrenamiento	Precisión de validación	Error de entrenamiento	Error de validación

3ª estrategia v1 sin E.S.	95.5%	96%	0.3291	0.3069
3ª estrategia v1 con E.S.	92.87%	95%	0.422	0.3585

Tabla 45.

En este caso sí podría ser interesante aplicar "early stopping" en caso de que sea necesario reducir el tiempo de entrenamiento a costa de medio punto de error y un 1% de precisión.

Variante 2: Regularización L1, función .Dropout(0.75)

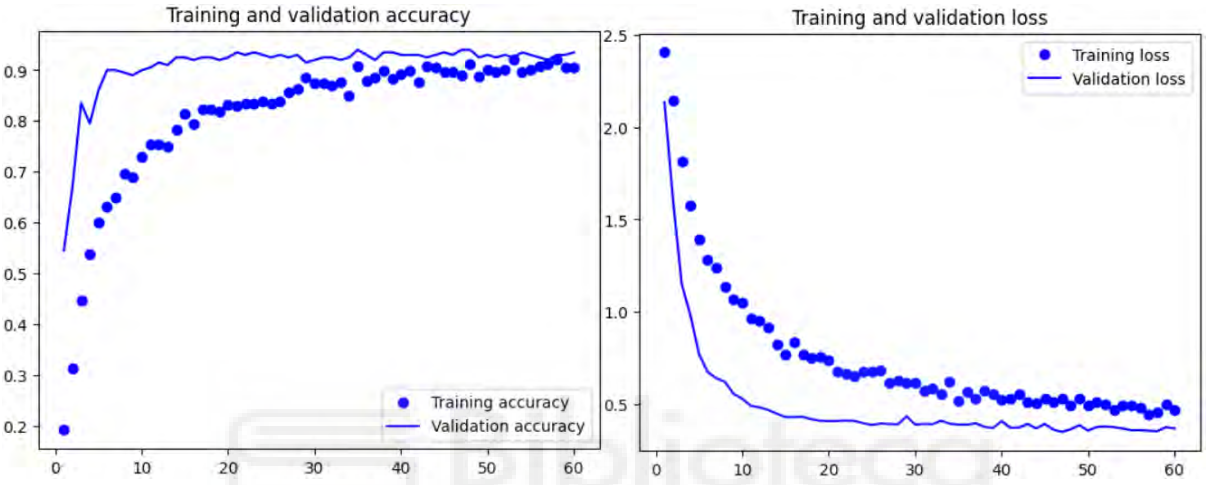

Imagen 33. División en Figuras 65 y 66.

Igual que en la variante anterior, se observa claramente la eliminación absoluta del sobreentrenamiento. Desafortunadamente, en la tabla 46 se puede observar que no se debe específicamente a que las curvas de validación se aproximen adecuadamente a sus objetivos. Comentar, finalmente, la estabilidad de las series de validación.

MNIST. Gráfica mixta	**Precisión de entrenamiento**	**Precisión de validación**	**Error de entrenamiento**	**Error de validación**
3ªestrategia v2 sin E.S.	92.12%	94%	0.446	0.3561
3ªestrategia v2 con E.S.	89.88%	93%	0.516	0.3816

Tabla 46.

No es recomendable utilizar "early stopping" en esta variante, principalmente pues esta variante es peor que la anterior.

Variante 3: Regularización L2, función .Dropout(0.5)

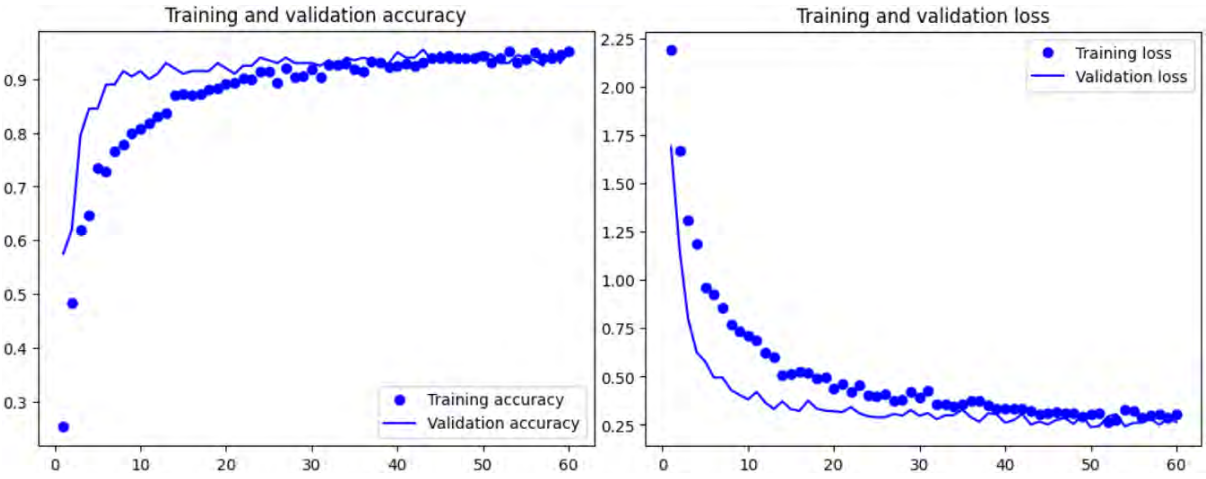

Imagen 34. División en Figuras 67 y 68.

Una vez más el sobreajuste es totalmente eliminado, además esta variante tiene el mejor error de validación. Sin embargo, este error sigue puntuando demasiado alto, 0.238 puntos. Por otro lado, la precisión de validación es bastante buena logrando un 95.5%.

MNIST. Gráfica mixta	Precisión de entrenamiento	Precisión de validación	Error de entrenamiento	Error de validación
3ª estrategia v3 sin E.S.	95.25%	95.5%	0.2642	0.238
3ª estrategia v3 con E.S.	91%	93.5%	0.3847	0.2866

Tabla 47.

Se desaconseja la utilización de "early stopping" en esta variante porque la pérdida de competitividad es extraordinaria.

Variante 4: Regularización L2, función .Dropout(0.75)

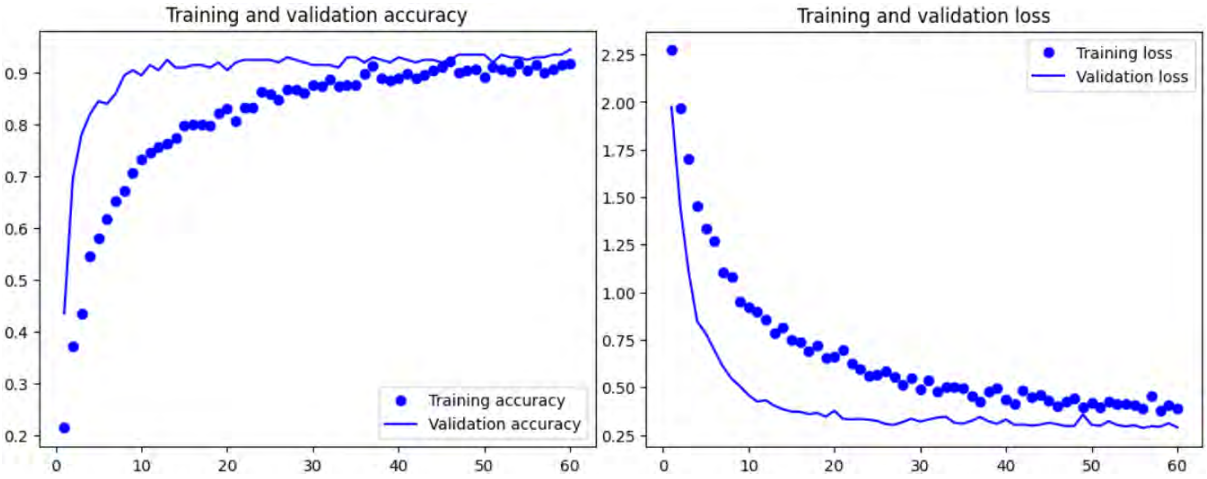

Imagen 35. División en Figuras 69 y 70.

Una vez más el sobreajuste es totalmente eliminado. Sin embargo, esto se debe principalmente a los malos resultados de entrenamiento. Además, las curvas de validación siguen débiles, puntuando la de error demasiado alto, 0.2863 puntos, y la de precisión algo baja, 94.5%.

MNIST. Gráfica mixta	**Precisión de entrenamiento**	**Precisión de validación**	**Error de entrenamiento**	**Error de validación**
3ª estrategia v4 sin E.S.	92.25%	94.5%	0.3792	0.2863
3ª estrategia v4 con E.S.	90%	94%	0.4763	0.2883

Tabla 48.

Se puede añadir la técnica “early stopping” a esta variable, no altera en exceso los resultados prácticos de validación.

4. ANÁLISIS COMPARATIVO DE LAS DIFERENTES TÉCNICAS Y ESTRATEGIAS CONTRA EL SOBREAJUSTE EN LA RED NEURONAL DE MNIST

MNIST. Gráfica mixta	Precisión de entrenamiento	Precisión de validación	Error de entrenamiento	Error de validación
Original	100%	90%	0.0000	0.5
Early Stopping	96.88%	90%	0.1003	0.3614
.Dropout(0.5)	99%	94%	0.031	0.2743
.Dropout(0.25)	99.87%	94%	0.014	0.3043
.Dropout(0.75)	97.62%	95%	0.0826	0.2433
Aumento de datos	98.75%	95%	0.0645	0.1670
A.D. Rotación máxima = 0.005	99.5%	93.5%	0.0250	0.1672
A.D. Zoom máximo = 0.05	100%	95.5%	0.0021	0.2594
A.D. Zoom máximo = 1	95.5%	95.5%	0.2	0.2
Regularización	100%	93.5%	0.0891	0.4184

L1: $\lambda = 0.0001$				
A.D. y dropout(0.5)	95.88%	96%	0.1455	0.1475
A.D. y dropout(0.75)	96.13%	95.5%	0.1384	0.1424
A.D. y L1	97.25%	94%	0.2503	0.3362
A.D. y L2	94.75%	95%	0.2921	0.2766
3ª estrategia v1	95.5%	96%	0.3291	0.3069
3ª estrategia v2	92.12%	94%	0.446	0.3561
3ª estrategia v3	95.25%	95.5%	0.2642	0.238
3ª estrategia v4	92.25%	94.5%	0.3792	0.2863

Tabla 49.

Si tu único objetivo es eliminar el sobreajuste en este tipo de red puedes utilizar cualquier variante de la tercera estrategia, la estrategia que combina el aumento de datos y la función ".dropout(0.5)" y el aumento de datos con un zoom = 1. De estas mi recomendación personal es la primera estrategia, es decir, la combinación de la técnica de aumento de datos y la función ".Dropout(0.5)" pues maximiza la precisión y disminuye a sólo 0.2 puntos el error.

Si tu objetivo es maximizar la precisión de validación tienes dos opciones. La primera es la mencionada anteriormente conjunción de la técnica de aumento de datos y la función ".Dropout(0.5)". La segunda es la primera versión de la tercera estrategia la cual combina las técnica de aumento de datos, la regularización L1 y la función ".Dropout(0.5)".

Si tu objetivo es minimizar el error de validación la mejor elección es la segunda variante de la primera estrategia, es decir, la unión de la técnica de aumento de datos y la función ".Dropout(0.75)". Es necesario comentar que la primera variante de esa estrategia también reduce mucho el error.

CAPÍTULO 5

RED NEURONAL DE “DOGS VS CATS”

El segundo caso de estudio trata también una aplicación de clasificación, concretamente clasificación binaria entre perros y gatos. Existe un conjunto de datos llamado “dogs-vs-cats” que se puede encontrar gracias a Kaggle, y está compuesto por 25000 imágenes de perros y gatos repartidas a partes iguales y sus respectivas etiquetas.

Esta segunda red neuronal es más compleja que la anterior debido principalmente a la complejidad superior de los datos de entrada. Por un lado, se requieren más canales en la capa de entrada pues estas imágenes tienen color. Por otro lado, el tamaño de las imágenes de “MNIST” es concreto y pequeño, 28 x 28 píxeles, a mayor tamaño mayor número de parámetros necesarios. También hay que tener en cuenta que en este caso existe un fondo diverso y la posibilidad de ruido. Finalmente, la ubicación indeterminada y las múltiples características requeridas en la identificación y clasificación.

1. DESCRIPCIÓN Y EXPLICACIÓN DE TODOS LOS ASPECTOS REQUERIDOS EN LA CREACIÓN Y ENTRENAMIENTO DE LA RED NEURONAL DE “DOGS VS CATS”

Esta aplicación consiste en la clasificación binaria en el conjunto de datos “dogs-vs-cats”. En la ilustración 3 se muestran dos ejemplos de las imágenes de entrada, el primero es un gato y el segundo es un perro.

Ilustración 3.

1.1. DESCARGA DE “DOGS VS CATS” Y PREPARACIÓN DE LOS DATOS

Bibliotecas

El paso previo a todo programa escrito en python es importar las bibliotecas necesarias:

```
import os, shutil, pathlib
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from google.colab import files
from tensorflow.keras.utils import image_dataset_from_directory
```

Descarga de "dogs vs cats"

Al contrario que con "MNIST", "dogs vs cats" no se encuentra incluído en keras sino, como ya se ha comentado, en Kaggle. Los siguientes comandos se utilizan para acceder correctamente a la web.

```
files.upload()
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Una vez subido, colocado y protegido el archivo clave, ya se pueden descargar los datos. La línea de código que hay a continuación se puede traducir como: De "Kaggle competitions" descargar el conjunto dogs-vs-cats. Las siguientes se utilizan para extraer los datos de los archivos ".zip".

```
!kaggle competitions download -c dogs-vs-cats
!unzip -qq dogs-vs-cats.zip
!unzip -qq train.zip
```

Reorganización de los datos, disminución del número de imágenes

El pequeño conjunto de datos que se utilizará en este proyecto, con objeto de que se produzca sobreajuste, se compone de un total de 3000 imágenes, 1500 perros y 1500 gatos. Para el entrenamiento 2000 y para la validación 1000. Como la colección de datos descargados es mucho más grande de lo necesario se crea un directorio donde copiar el pequeño conjunto.

```
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
```

```
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)
make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
```

Normalizado de los datos

Los datos no se pueden intoducir sin más como archivo JPEG, deben tener un formato concreto. Los datos deben ser tensores de coma flotante con un tamaño adecuado y correctamente procesados. Los pasos de formateo son:

1. Leer los archivos de imagen.
2. Decodificar el contenido JPEG en cuadrículas RGB de píxeles.
3. Convertirlas en tensores de coma flotante.
4. Redimensionarlos a un mismo tamaño, en nuestro caso 180x180.
5. Agruparlos en lotes (lotes de 32 imágenes).

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
```

“image_dataset_from_directory” es una función de keras que permite formatear rápidamente lotes de imágenes convirtiéndolos en lotes de tensores preprocesados. Los argumentos que debe tener la función son: un nuevo directorio (“new_base_dir”), el nuevo tamaño (“image_size”) y el tamaño de los lotes (“batch_size”).

1.2. CREACIÓN DEL MODELO DE UNA RED NEURONAL O CONVOLUCIONAL

Una vez descargados, almacenados y formateados los datos de entrenamiento se puede diseñar el modelo. La red neuronal convolucional es similar a la anterior, salvo por una cantidad mayor de capas y por una capa de reescalado previa.

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

La primera capa limita la entrada a imágenes con tres canales de tamaño 180 x 180 píxeles. La segunda capa reescala los valores de los inputs al rango [0,1] dividiendo entre 255. En lo relativo a la capa .Dense, solo es necesario una salida así que el valor de units es 1.

1.3. COMPILACIÓN Y ENTRENAMIENTO DEL MODELO

En cuanto a la compilación y el entrenamiento, el proceso y el código empleado es muy similar o idéntico al utilizado en la red de "MNIST". Concretamente, la compilación es la misma.

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Entrenamiento del modelo

Tras la compilación del modelo el paso a seguir es el entrenamiento. De la misma manera que en el caso de "MNIST" se utilizará la función "ModelCheckpoint()" de la clase callbacks de keras con objeto de almacenar el modelo entrenado más hábil.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
```

El entrenamiento en sí mismo se produce aplicando el método ".fit". Con objeto de poder representar los resultados se guardan en la memoria "history".

```
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
```

```
    callbacks=callbacks)
```

Dado que se introducen a priori menos argumentos puede parecer que el entrenamiento se simplifica, pero no es así. “train_dataset” es una variable que permite introducir lotes de datos de entrenamiento sustituyendo a “x_train” e “y_train”. La otra diferencia es que “batch_size” no se incluye, así pues, se utilizará el valor por defecto, 64.

1.4. REPRESENTACIÓN DE LAS CURVAS DE PRECISIÓN Y ERROR

Igual que con los puntos anteriores, el código utilizado es coincidente con el de la red anterior.

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

Representación del entrenamiento de una red “dogs vs cats”

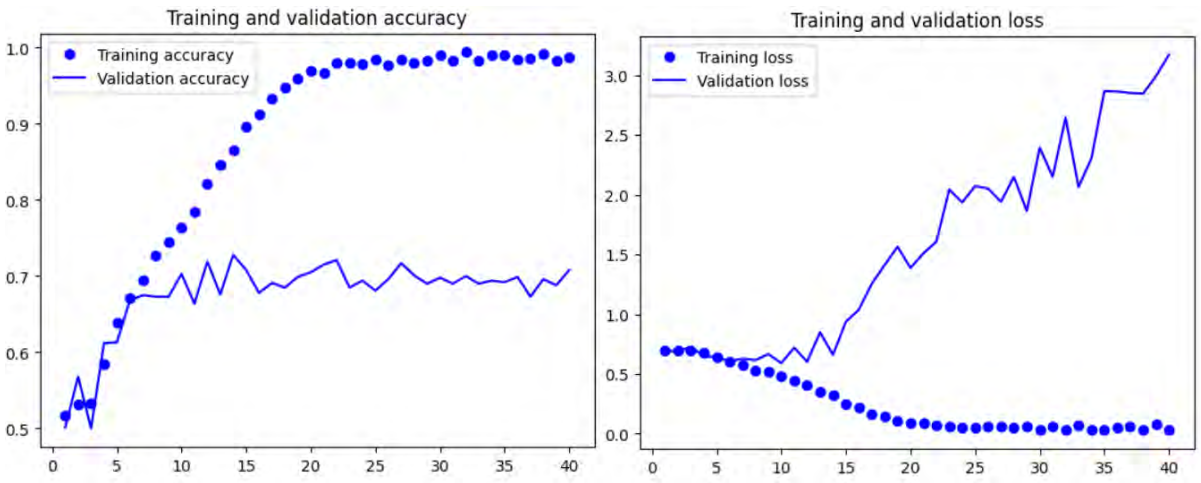

Imagen 36. División en Figuras 71 y 72.

Estos resultados son un ejemplo clásico de “overfitting”. Se produce debido, entre otros

motivos, a la enorme escasez de imágenes de entrada que se impone.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%

Tabla 50.

En la figura 71 se representa la precisión (accuracy) en el eje X. Se puede observar que en el caso de las imágenes de entrenamiento, se alcanza una precisión cercana al 100%. Por el contrario, los datos de validación solo alcanzan una precisión aproximada del 70%. Esta diferencia es el llamado sobreajuste. Además, se advierte que ambas líneas tienen la misma tendencia en un principio, hasta que la precisión se acerca al 70%. En esta red neuronal, con estos datos, se produce entre las épocas 5 y 10.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.0277	0.5899	< 5	0.5622

Tabla 51.

En la figura 72 se representa el error (loss) en el eje Y. Se advierte que el error en los datos train se aproxima a 0. Opuestamente, el error en los datos de validación aumenta significativamente hasta llegar a los dos puntos. Igual que con la precisión, en esta diferencia se entiende el sobreajuste. Los datos de entrenamiento y de validación tienen al principio unos valores similares, pero a partir de las épocas 5-10 el error de validación se incrementa.

2. APLICACIÓN DE TÉCNICAS CONTRA EL SOBREAJUSTE EN LA RED NEURONAL CONVOLUCIONAL DE "DOGS VS CATS"

Después de diseñar y comprender la ConvNet junto con su proceso de entrenamiento bajo restricción de datos, se emplean diversas técnicas contra el sobreajuste. En el tercer capítulo se explican en detalle las diferentes técnicas, no obstante, se realizará una pequeña descripción de cada técnica para facilitar el entendimiento de los resultados.

2.1. PRIMERA TÉCNICA. EARLY STOPPING

Como su nombre indica, el sobreentrenamiento se produce por un exceso de entrenamiento. Con objeto de detener el entrenamiento antes de excederse se introduce una "callback" denominada "early_stopping".

```
from tensorflow.keras.callbacks import EarlyStopping
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
early_stopping = EarlyStopping(
    monitor= "val_loss",
    patience = 3,
    restore_best_weights = True
)
history = model.fit(
    train_dataset,
    epochs=40,
    validation_data=validation_dataset,
    callbacks=[callbacks, early_stopping]
```

Dando como resultado:

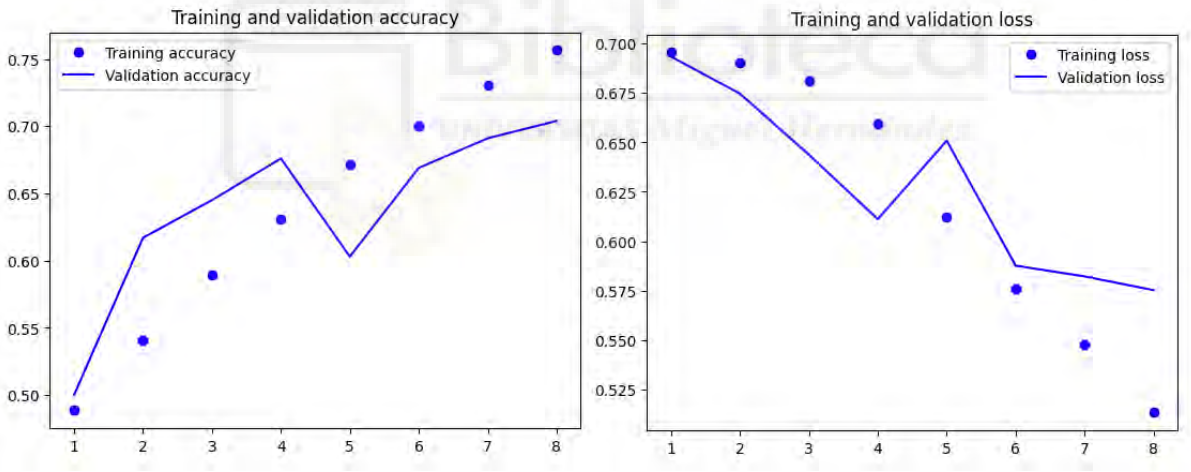

Imagen 37. División en Figuras 73 y 74.

En el caso de una red de tamaño medio sí parece ser efectivo contra el sobreentrenamiento. Con todo, tanto la precisión como el error dan resultados ineficientes.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original.	100%	72.1%	< 5	27.9%
Early Stopping	77.8%	70%	5	7.8%

Tabla 52.

Claramente mejora el sobreajuste de precisión, pero es debido principalmente a una bajada de

la precisión de entrenamiento no a una mejora en la generalización pues no se da.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
Early Stopping	0.4711	0.5919	6	0.1208

Tabla 53.

Se reduce el sobreajuste de error también, pero al igual que con la precisión se debe a unos malos datos de entrenamiento, no existe una mejora en la validación.

Esta técnica es muy probable que pueda ser aplicable en las estrategias contra el sobreajuste de una red convolucional de tamaño mediano, sin embargo, a nivel individual los resultados son deplorables.

2.2. SEGUNDA TÉCNICA. "DROPOUT"

Uno de los motivos por los que se produce el "overfitting" es un excesivo tamaño y una excesiva complejidad de la red, la técnica del abandono ayuda a solucionar ambas vicisitudes. La nueva red convolucional quedaría así:

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

La función ".Dropout()" tiene un argumento, el cual indica la cantidad de neuronas a abandonar. Así pués, se probará la técnica con distintos argumentos, 0.25, 0.5 y 0.75.

Resultados de aplicar Dropout(0.5)

0.5 es el parámetro que se suele añadir por defecto.

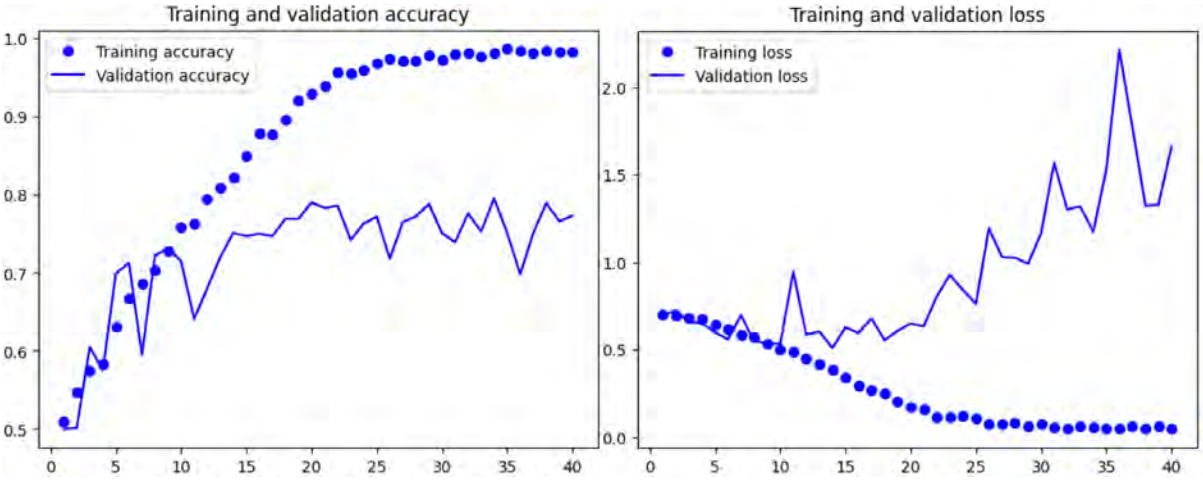

Imagen 38. División en Figuras 75 y 76.

Obviamente sigue habiendo sigue habiendo "overfitting", pero se retrasa y disminuye.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
.Dropout(0.5)	98.65%	79%	<10	19.65%

Tabla 54.

Existe una mejora en el sobreentrenamiento reduciéndose en un 8.25% y produciéndose este al menos 5 épocas después. Esto es en gran parte consecuencia de la mejora en la precisión de validación, observable en la figura 75, alcanzando la estabilidad en torno al 77% con un máximo en 79%. Por otra parte, hay un pequeño empeoramiento de la precisión de entrenamiento que no logra cumplir el objetivo de aproximarse al 100% el 98.65%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.0277	0.5899	< 5	0.5622
.Dropout(0.5)	0.0466	0.5092	>10	0.4626

Tabla 55.

Esta técnica consigue una pequeña mejora, de 0.1 puntos y un retraso del sobreentrenamiento de al menos cinco épocas. Hay una mejora evidente en la validación mínima. A pesar de ello, la tendencia exponencial de su serie en la figura 76 muestra un muy ineficiente entrenamiento.

Resultados Dropout(0.25)

Veamos que sucede al aplicar el abandono en un 25% de las neuronas permitidas.

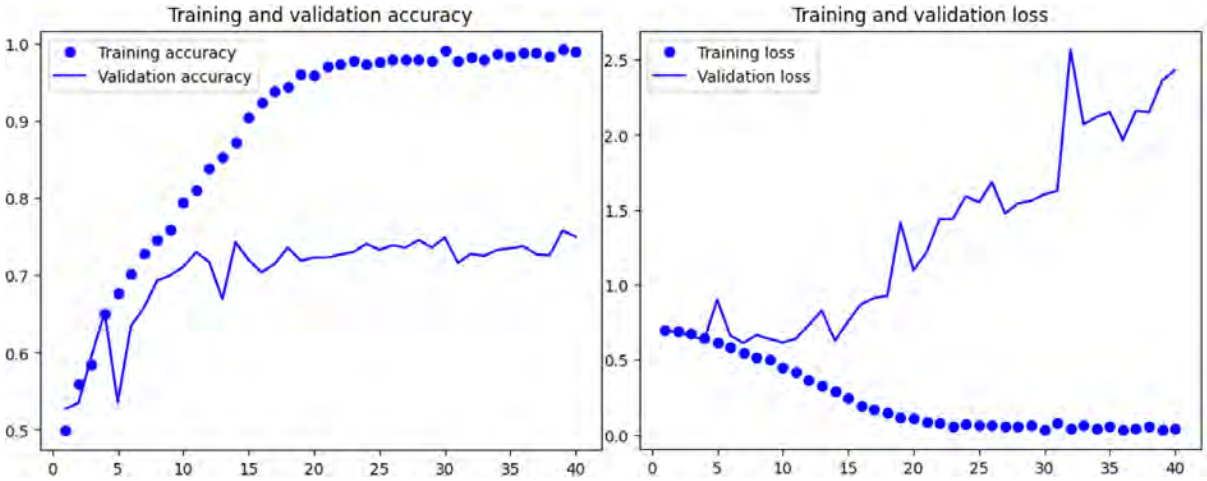

Imagen 39. División en Figuras 77 y 78.

Parece que abandonar tan poca cantidad de neuronas no parece suficiente.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
.Dropout(0.25)	99.2%	75.8%	<5	23.4%

Tabla 56.

En el sobreajuste de precisión hay una ligera mejora de 4.5% debido a una ligera mejora en la tendencia final de la curva de validación, valores cercanos al 75%, con un máximo en 75.8%. El empeoramiento en el entrenamiento apenas es visible en la figura 77.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
.Dropout(0.25)	0.0304	0.6121	<5	0.5817

Tabla 57.

En cuanto al error, figura 78 y tabla 57, no hay ninguna mejora respecto al modelo original. Por ello este argumento en la segunda técnica queda totalmente descartado.

Resultados Dropout(0.75)

Veamos qué sucede al aumentar la proporción de abandono.

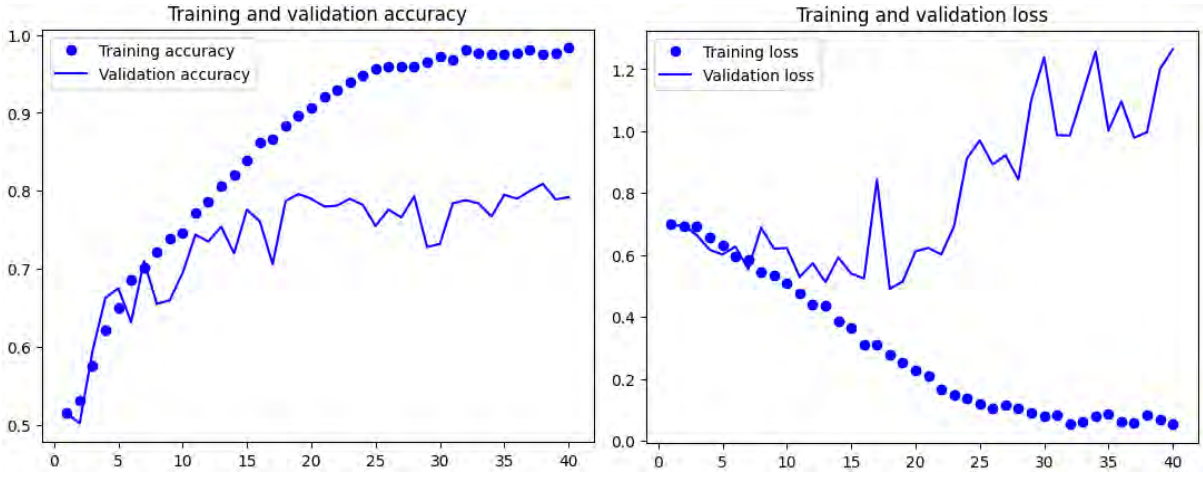

Imagen 40. División en Figuras 79 y 80.

En la figura 79, la de precisión, se observa un tramo de casi diez épocas donde el sobreentrenamiento es muy reducido, a partir de la época 15 el desfase se agranda. En cuanto al error la gráfica es muy similar a la original.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
.Dropout(0.75)	98.45%	81%	<10	17.45%

Tabla 58.

Por primera vez esta red, la precisión de validación logra superar el 80%, en concreto alcanza el 81%. La curva se estabiliza alrededor del 79.5%. El sobreajuste se reduce en 10.45% y se retrasa unas 5 épocas. El único punto negativo es la precisión de entrenamiento, que se reduce ligeramente hasta el 98.45%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.0277	0.5899	< 5	0.5622
.Dropout(0.75)	0.0545	0.5131	<10	0.4586

Tabla 59.

En la gráfica 80, la del error, el error se reduce en poco más de una décima. Obviamente es una mejora, pero insuficiente; sobre todo porque una parte de la reducción se debe a un aumento del error de entrenamiento.

2.3. TERCERA TÉCNICA. TAMAÑO DE LA RED

Es ampliamente reconocido que tanto una red convolucional excesivamente grande como una demasiado pequeña pueden aportar al problema de sobreajuste. Por esta razón, esta técnica aborda directamente la cantidad de capas neuronales.

Eliminar capas

Este enfoque reduce la cantidad de capas convolucionales, como se puede apreciar en el código siguiente:

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Los resultados de este modelo se representan:

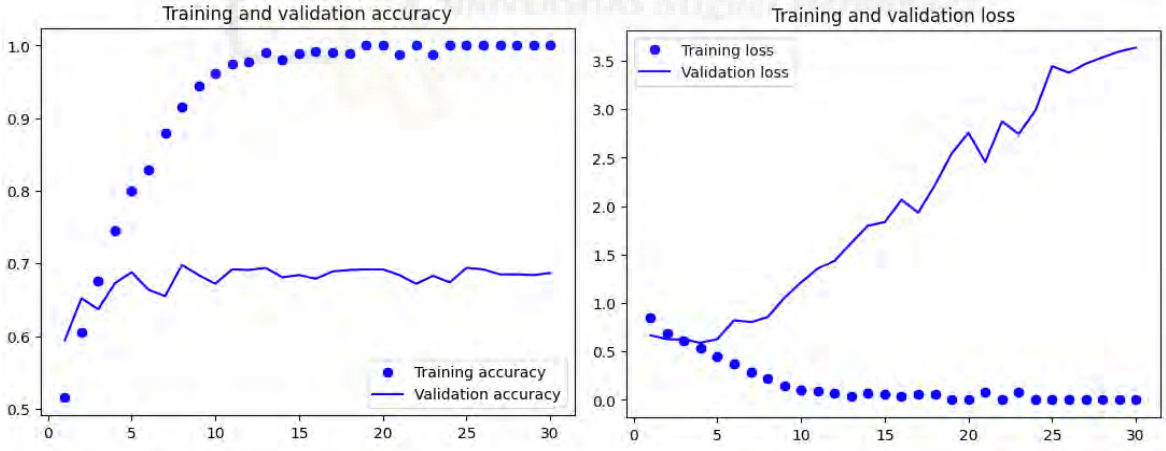

Imagen 41. División en Figuras 81 y 82.

Con estos resultados se observan algunas cosas interesantes: El modelo apenas entrena, casi toda la imagen 41 es sobreentrenamiento. Por esto mismo la precisión no consigue llegar al 70% y el error evoluciona exponencialmente desde el principio.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%

Capas eliminadas	100%	69.6%	<5	30.4%

Tabla 60.

En lo relativo a la figura 81, indicar que ninguno de los parámetros que se encuentran en la tabla 60 superan al original. Aumenta el sobreajuste de precisión y disminuye la precisión de validación.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.0277	0.5899	< 5	0.5622
Capas eliminadas	0.0001	0.5945	< 5	0.5944

Tabla 61.

Igual que con la precisión, el error empeora generalizadamente. El único parámetro que mejora es el error de entrenamiento, este se aproxima mucho más al cero absoluto. El error de validación mínimo es poco mayor al original, sin embargo, su evolución es tal que en las épocas finales se alcanzan los 4 puntos.

Aumentar capas

Esta modificación consiste en aumentar el número de capas neuronales como podemos observar en el siguiente código:

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

El modelo tiene como representación:

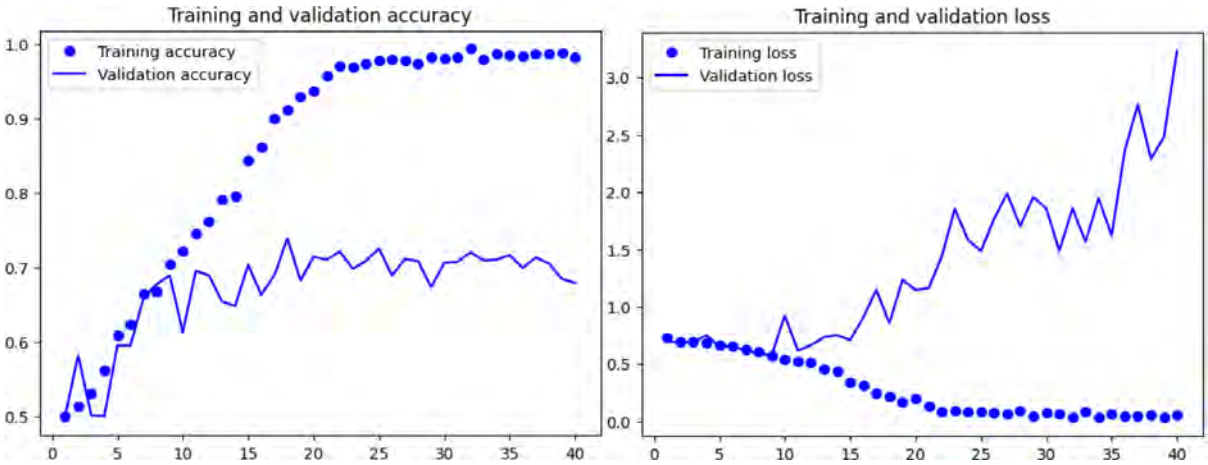

Imagen 42. División en Figuras 83 y 84.

Indicar un retraso en el sobreentrenamiento representado en la imagen 42 de cinco épocas.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
Capas añadidas	99.3%	72.5%	<10	26.8%

Tabla 62.

Los datos no varían en exceso, la pequeña diferencia en la precisión de validación está dentro del rango de variación que se da cada vez que se entrena una red. Por otro lado, el decrecimiento del entrenamiento si es importante.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.0277	0.5899	< 5	0.5622
Capas añadidas	0.0388	0.5749	<10	0.5361

Tabla 63.

Igual que con la precisión, los datos de validación son tan similares respecto al original que se puede entender desde la variabilidad propia del entrenamiento. El único punto no anecdótico es el desfase en el inicio del sobreajuste mencionado anteriormente.

Teniendo en cuenta estos resultados podría entenderse que esta red convolucional podría ser mejor como punto de partida. No es así pues al ser de mayor tamaño y de mayor complejidad surgen diferentes problemas añadidos. Una de estas dificultades es que requiere una mayor fuerza computacional con muy poca mejora. Otro problema significativo es que al aumentar tanto la complejidad algunas técnicas como el abandono, la segunda, pierden efectividad.

2.4. CUARTA TÉCNICA. ACTIVACIÓN DEL CLASIFICADOR

Otra parte de la red convolucional alterable es la activación del clasificador. La teoría expresada en el capítulo 3 nos indica que varias funciones podrían tener sentido utilizar:

Funciones "ReLU" y "Leaky ReLU"

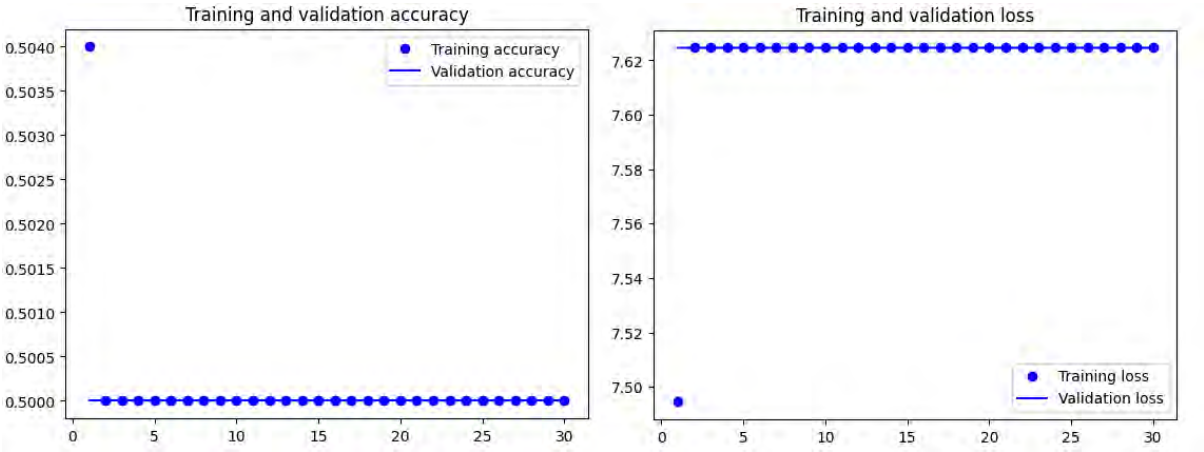

Imagen 43. División en Figuras 85 y 86.

Como se puede observar en la gráfica, las funciones de activación "ReLU" y "Leaky ReLU" no son útiles. La precisión se mantiene en 50%, lo que en el caso de una selección binaria significa que no es capaz de (hay que tener en cuenta que la mitad de la muestra son perros y la otra mitad son gatos). En cuanto al error se mantiene alto y constante.

Función "tanh"

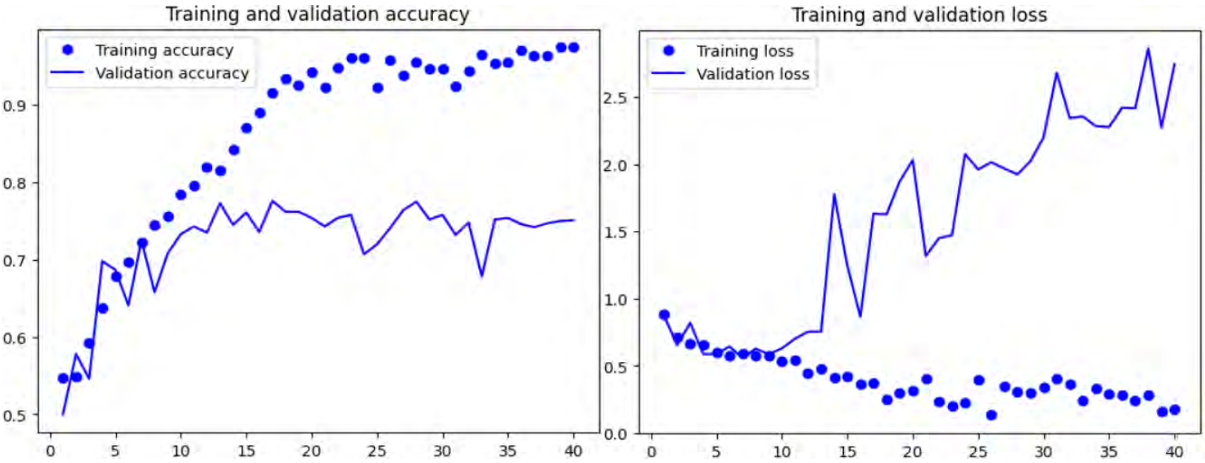

Imagen 44. División en Figuras 87 y 88.

El clasificador con la función "tanh" puede ser una opción interesante, principalmente por el aumento de precisión y por el retraso del inicio del sobreajuste. Sin embargo, tanto las series de validación y entrenamiento tienen una variabilidad muy alta dando a la gráfica una forma de sierra.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
Tanh	97.45%	77.6%	<10	19.85%

Tabla 64.

En lo relativo a la precisión, figura 85, el problema del sobreajuste se reduce a menos del 20%. El mayor punto positivo de la función “tanh” es su alta precisión de validación. El dato negativo, es la limitación impuesta debido a la baja precisión de entrenamiento. En este sentido hay que tener en cuenta que la mayoría de las otras técnicas aplicables reducen todavía más esta variable.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
Tanh	0.1604	0.6296	<10	0.4692

Tabla 65.

Pese a que el sobreajuste de error se reduce en casi una décima, el error de validación aumenta. El sobreentrenamiento disminuye principalmente debido a un muy mal error de entrenamiento.

Esta función no será utilizada debido a que, sin importar la estrategia empleada, demanda un alto número de épocas para aproximarse al objetivo. Además, resulta crucial que las series de entrenamiento sean efectivas.

Función “softmax”

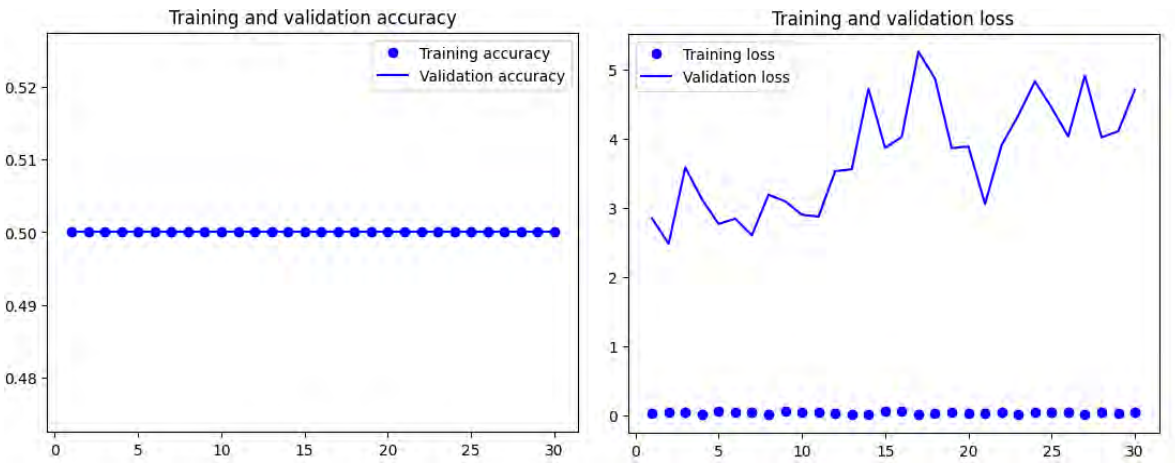

Imagen 45. División en Figuras 89 y 90.

Este resultado es decepcionante pues “softmax” es utilizado en clasificadores multiclases. Igual que las funciones “ReLU” y “Leaky ReLU”, la función activadora “softmax” no es útil para esta red convolucional. La red parece no aprender manteniendo una precisión constante del 50%. En cuanto al error, los datos de validación son desastrosos.

Función “swish”

Igual que las funciones “ReLU”, “Leaky ReLU” y “softmax”, la función activadora “swish” no es útil para esta red convolucional. No solo el error es muy elevado y la precisión se mantiene constante en 50% sino que esta función exige una gran potencia computacional.

2.5. QUINTA TÉCNICA. AUMENTO DE DATOS

El aumento de datos es una estrategia común para mitigar el sobreajuste. Esta técnica implica incrementar la cantidad de imágenes de entrada mediante transformaciones no repetitivas. Estas transformaciones se aplican de manera aleatoria y pueden incluir espejo, rotación y zoom. El código base de esta técnica es el siguiente:

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

El resultado de aplicar esta técnica:

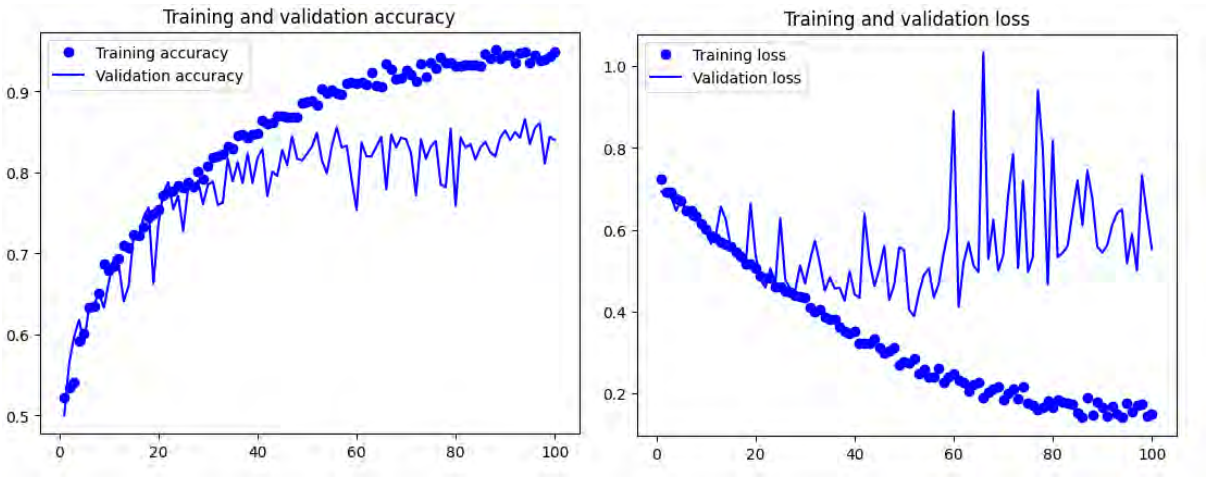

Imagen 46. División en Figuras 91 y 92.

Con objeto de seguir apreciando el sobreajuste se ha aumentado el número de épocas total hasta 100. Este será el número de épocas por defecto al utilizar aumento de datos.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
A.D.	94.36%	85.7%	>30	8.66%

Tabla 66.

En la figura 89, la de precisión, se observa un sobreajuste muy reducido en comparación al original, en gran medida se debe a la gran mejoría de los datos de validación cuya tendencia se mantiene cercana al 83% con un máximo en 85.7%, un 13.6% superior. Pese a ello, también hay un importante decrecimiento de la precisión de entrenamiento, cuya trayectoria final se mantiene en el entorno del 94%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
A.D.	0.1402	0.3975	< 30	0.2573

Tabla 67

En lo relativo al error, figura 90, también hay una gran disminución del sobreentrenamiento. En parte se debe a la atenuación del error mínimo de validación en casi dos décimas, no obstante, la tendencia final de la serie se asienta en puntuaciones cercanas a 0.75. Otro motivo para la mejoría en el sobreajuste es el valor mínimo de entrenamiento, que lejos de aproximarse a cero o al error de entrenamiento original se mantiene alrededor de 0.145 puntos.

Las funciones que componen la técnica del aumento de datos tienen argumentos que pueden variar, con objeto de comprobar que los asignados son los óptimos se variarán a continuación. Por ello las próximas tablas de la técnica no se aplicarán respecto al original, sino a los datos del aumento de datos base.

Alteración del parámetro de la función .RandomFlip() en el aumento de datos

La primera función del aumento de datos cuyo argumento es modificable es ".RandomFlip()". Esta función aplica un espejo en el sentido indicado por el atributo, en el caso original "horizontal", sobre cada imagen de entrada. "Vertical" es el otro argumento que puede recibir esta función.

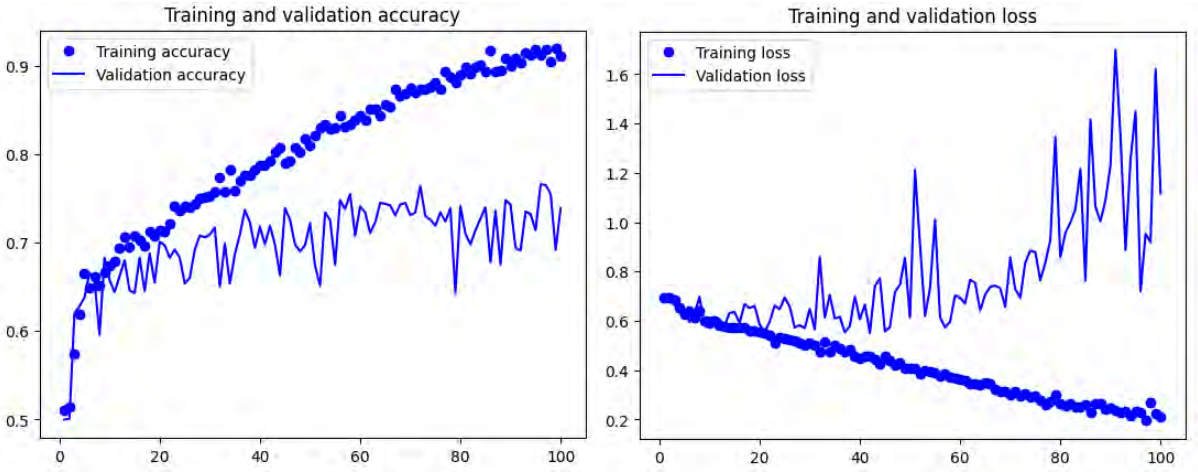

Imagen 47. División en Figuras 93 y 94.

Visualmente se puede observar que esta opción no mejora los datos de validación respecto al aumento de datos básico.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A. D.	94.36%	85.7%	>30	8.66%
Flip Vertical	93.25%	75.63%	< 20	17.62%

Tabla 68.

En cuanto a la precisión, ninguno de los datos representados en la tabla 68 superan a los del aumento de datos base. El sobreentrenamiento casi se duplica y se retrasa al menos diez épocas.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573
Flip Vertical	0.1978	0.583	20	0.3852

Tabla 69.

Al igual que con la precisión, en el error no existe mejora alguna. El sobreajuste aumenta en más de una décima y el error de validación es muy similar al de la gráfica, de la red original.

Alteración del parámetro de la función ".RandomRotation()" en el aumento de datos

Otra función del aumento de datos cuyo argumento es modificable es ".RandomRotation()". Esta función aplica un giro a la imagen de entrada. La rotación máxima que se puede ejecutar es la indicada por el atributo, por tanto, variar este puede implicar muchas alteraciones excesivamente elevadas o muchas variantes demasiado similares entre sí.

Parte 1: Rango de giro menor

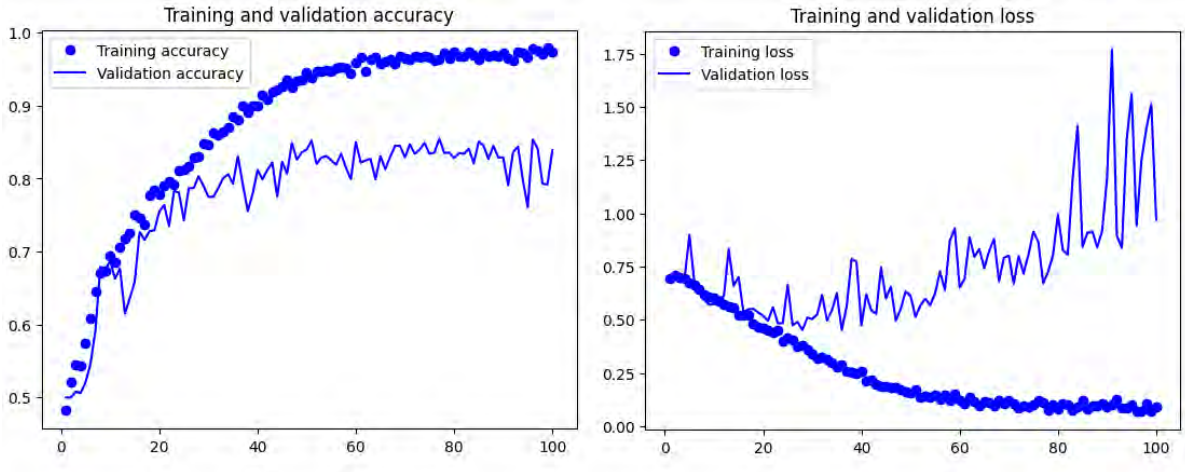

Imagen 48. División en Figuras 95 y 96.

Dado que el rango de rotación es menor, la variabilidad es menor. Por tanto, los datos de entrenamiento deberían mejorar y el sobreentrenamiento aumentar ligeramente. En general, reducir el rango de rotación al orden de las centésimas no es eficaz para esta red.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A. D.	94.36%	85.7%	>30	8.66%
Rotación máxima = 0.01	97.89%	84.21%	< 20	13.68%

Tabla 70.

Como ya he mencionado, la precisión de entrenamiento aumenta hasta casi un 98% en trayectoria ascendente. Por otro lado, la serie de validación de la gráfica 93, cuyo valor máximo es 84.21%, se consolida en porcentajes cercanos al 83.5%. Estos datos provocan un aumento del "overfitting".

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573
Rotación máxima = 0.01	0.0714	0.4286	< 20	0.3572

Tabla 71.

El sobreajuste de error crece en una décima. Por una parte, el error de validación mínimo aumenta tres centésimas. Por otra parte, el mínimo de la serie de entrenamiento se reduce a la

mitad, siete centésimas. Además, el error se produce mucho antes.

Parte 2: Rango de giro mayor

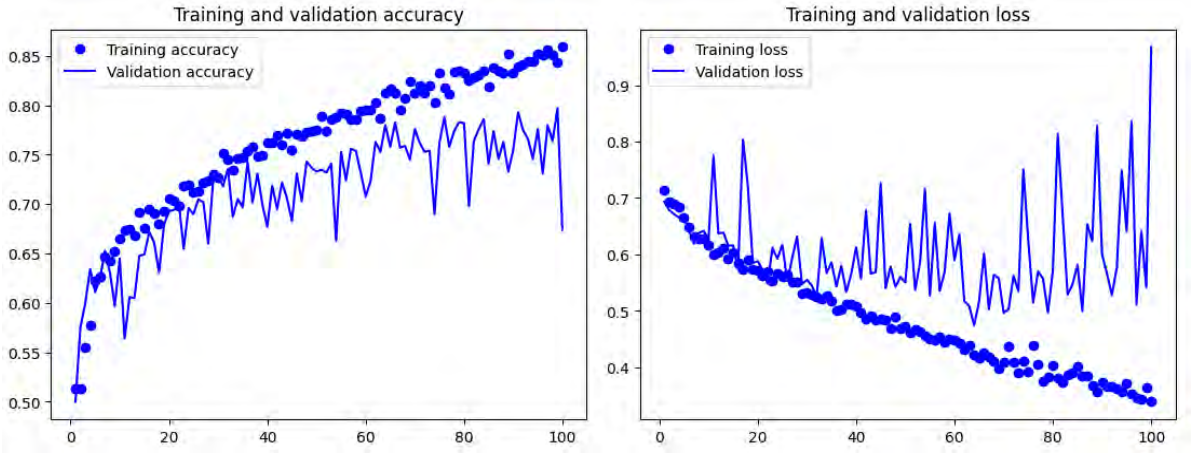

Imagen 49. División en Figuras 97 y 98.

Ya que el rango de rotación es mayor hay más diversidad en las imágenes, pero es posible que algunos giros excesivos distorsionen el aprendizaje y la generalización de algunos conjuntos de píxeles. Por ese motivo los datos de entrenamiento no se estabilizan y existen picos negativos en los datos de validación. En general, aumentar el rango de rotación al mayor orden posible no es eficaz para esta red por el decrecimiento de la precisión.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A. D.	94.36%	85.7%	>30	8.66%
Rotación máxima = 1	85.65%	78.73%	> 35	6.92%

Tabla 72.

El sobreajuste se rebaja en un 1.74% y se produce 5 épocas después, sin embargo, estos muy positivos resultados se empañan por la escasa precisión; tanto en la serie de entrenamiento con trayectoria ascendente y máximo en el 85.65%, como en la secuencia de validación que tiene una tendencia estable y un máximo menor al 80%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573
Rotación máxima = 1	0.2037	0.4813	> 35	0.2776

Tabla 73.

Pese a que el sobreajuste es muy similar este se produce desfasado una décima. Es decir, tanto el valor mínimo del error de entrenamiento como el de validación son aproximadamente una décima superior. Además, en este caso el error de validación se mantiene constante en valores cercanos a 0.5 puntos salvo por numerosos picos. El error de entrenamiento tiene una trayectoria decreciente.

Alteración del parámetro de la función ".RandomZoom()" en el aumento de datos

La última función del aumento de datos cuyo argumento es modificable es ".RandomZoom()". Esta función aplica un escalado a la imagen de entrada. El escalado máximo que se puede ejecutar en ambos sentidos es el indicado por el atributo. Por tanto, variar el rango del zoom puede implicar alteraciones excesivamente elevadas, pudiendo eliminar datos necesarios, o excesivas variantes demasiado similares entre sí. A continuación, se disminuye el rango.

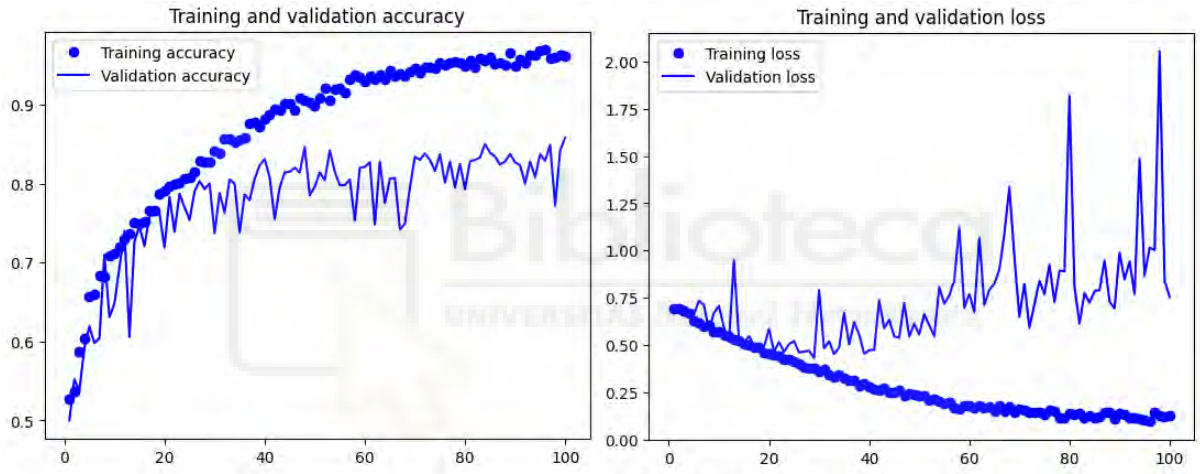

Imagen 50. División en Figuras 99 y 100.

Como las imágenes de entrada son menos diversas el entrenamiento se acelera, produciendo antes el sobreajuste, aproximadamente en la época 20. Además, los datos de entrenamiento deberían mejorar y el sobreentrenamiento aumentar ligeramente. En general, aumentar el rango de rotación al mayor orden posible no es eficaz para esta red por el crecimiento del error.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A. D.	94.36%	85.7%	>30	8.66%
Zoom máximo = 0.02	97%	86.12%	> 20	10.88%

Tabla 74.

Se ha mencionado que el sobreentrenamiento aumenta ligeramente, en este caso en un 2.22%. Sin embargo, tanto los datos de entrenamiento (97%) como los de validación (86.12%) mejoran

al menos ligeramente.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573
Zoom máximo = 0.02	0.085	0.4666	< 25	0.3816

Tabla 75.

Ya que la gráfica de validación es ascendente (promedio final de 1), el sobreentrenamiento aumenta exageradamente con las épocas, siendo el final cercano a 0.915 puntos.

2.6. SEXTA TÉCNICA. REGULARIZACIÓN

La regularización constituye una estrategia que introduce restricciones adicionales mientras se entrena la red neuronal. Su propósito radica en prevenir el sobreajuste, elevar el desempeño y lograr la generalización del modelo. En este análisis, se emplean las dos técnicas más usuales con este fin:

- Regulación L1: Integra la suma de los valores absolutos de los pesos del modelo en la función de pérdida.
- Regularización L2: Incorpora la suma de los cuadrados de los pesos del modelo en la función de pérdida.

Ambas formas de regularización se introducen en las capas convolucionales (Conv2D) y en la capa densamente conectada (Dense). Además, es posible combinarlas entre sí y con otras técnicas como “dropout” o “data augmentation”.

2.6.1. REGULARIZACIÓN L1

A continuación, se aplicará sobre la red convolucional la regularización L1 con diferentes hiperparámetros λ con el fin de analizar las diferentes variantes.

Variante 1: λ = 0.0001 o superior.

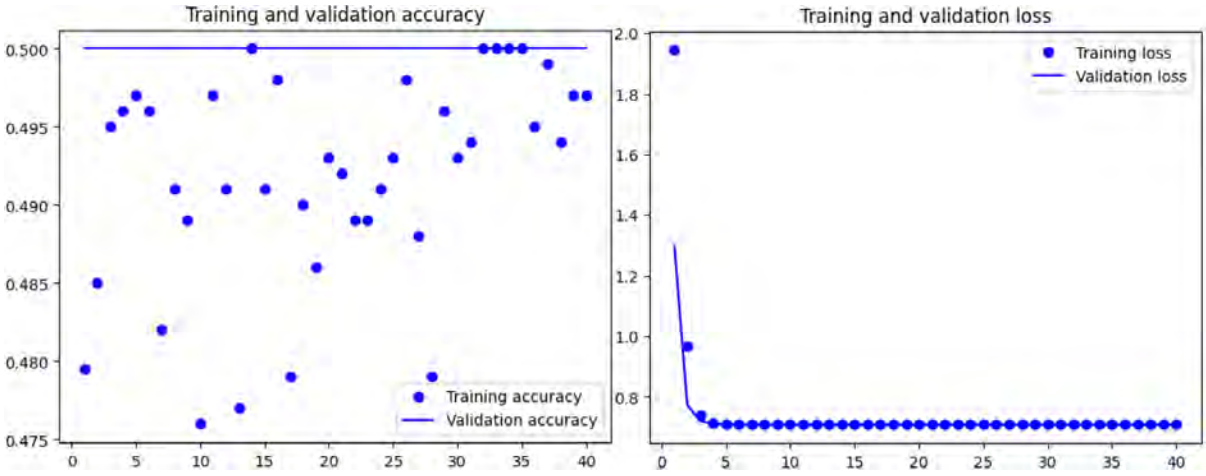

Imagen 51. División en Figuras 101 y 102

Cuando el valor del hiperparámetro λ es excesivamente elevado, la restricción adicional impuesta también se intensifica, lo que simplifica en exceso la red y dificulta el proceso de entrenamiento al punto de evitar un aprendizaje efectivo.

Variante 2: λ = 0.00001

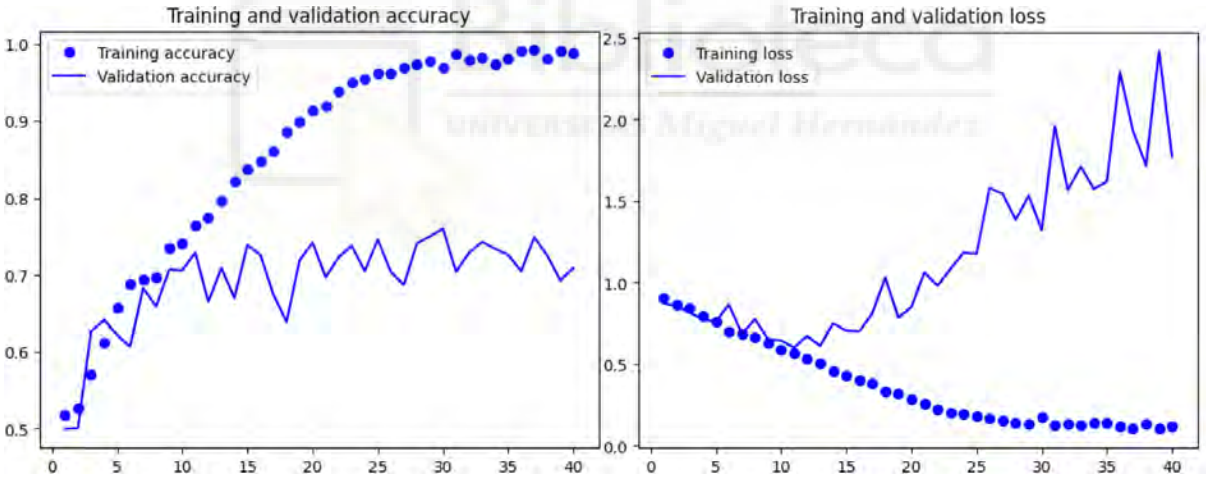

Imagen 51. División en Figuras 101 y 102

Los resultados son relativamente similares al original salvo por un retraso del sobreentrenamiento de 5 épocas y un desfase del 5% en la tendencia de la precisión.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
R. L1: λ = 0.00001	99.2%	76%	< 10	23.2%

Tabla 76.

El sobreajuste se reduce en casi un 5% debido, por una parte de la mejora de 3.9% de la serie de validación y por otra parte una ligera reducción del valor máximo y de la tendencia de la precisión de entrenamiento.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
R. L1: $\lambda = 0.00001$	0.1033	0.6017	< 10	0.4984

Tabla 77.

En el caso del error, el sobreajuste disminuye exclusivamente por la serie de entrenamiento que no consigue aproximarse a los datos finales del original. En cuanto a la secuencia de validación, esta es similar a la de la red original con un pequeño déficit de 0.0118 puntos.

Variante 3: $\lambda = 0.000001$ o inferior

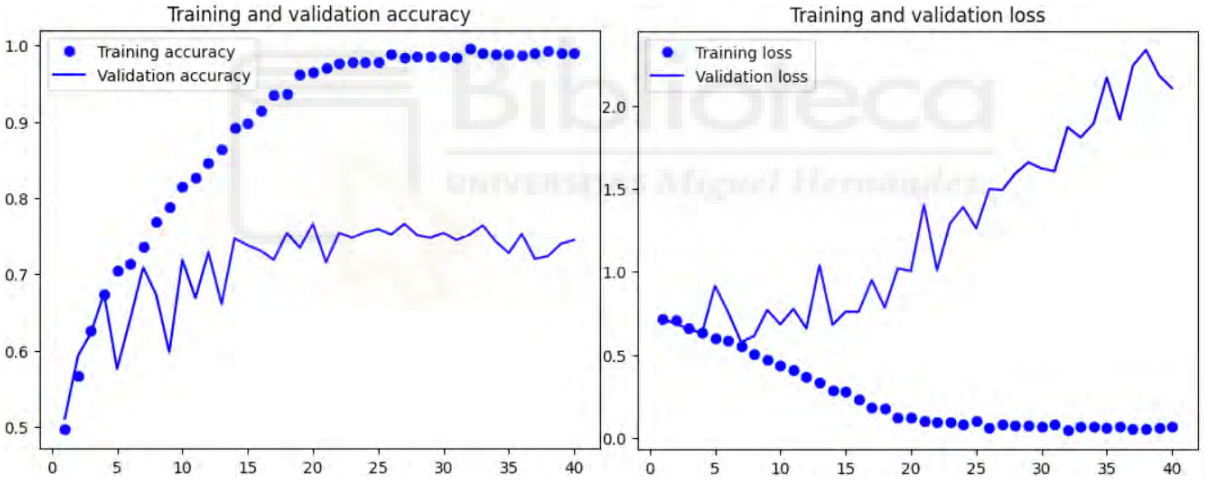

Imagen 52. División en Figuras 103 y 104.

Esta variante añade una menor penalización que la anterior y por ello la curva es más similar a la original que la anterior, los datos de entrenamiento se ajustan más y el sobreentrenamiento se produce en épocas muy cercanas. A partir de esta magnitud el efecto de las restricciones se atenúa profundamente, la función de pérdida no varía pues la función que se ha de sumar siempre vale aproximadamente 0.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%

R. L1: $\lambda = 0.000001$	99.5%	76.6%	< 5	22.9%

Tabla 78.

He mencionado que los datos de entrenamiento se asemejan más a los originales, sólo 0.5% de diferencia. Sin embargo, la serie de validación si aumenta hasta el 76.6%, un 4.5% de divergencia, reduciendo así el sobreajuste de precisión en un 5%. La trayectoria de validación se mantiene en torno al 75%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
R. L1: $\lambda = 0.000001$	0.0534	0.5763	5	0.5229

Tabla 79.

Tanto el error de entrenamiento como el de validación se ajustan al original. Por ello el sobreajuste de error solo difiere en 4 centésimas. Por tanto, en el error apenas hay mejora.

2.6.2. REGULARIZACIÓN L2

A continuación, se aplicará sobre la red convolucional la regularización L2 con diferentes hiperparámetros λ con el fin de analizar las diferentes variantes.

Variante 1: $\lambda = 0.01$ o superior.

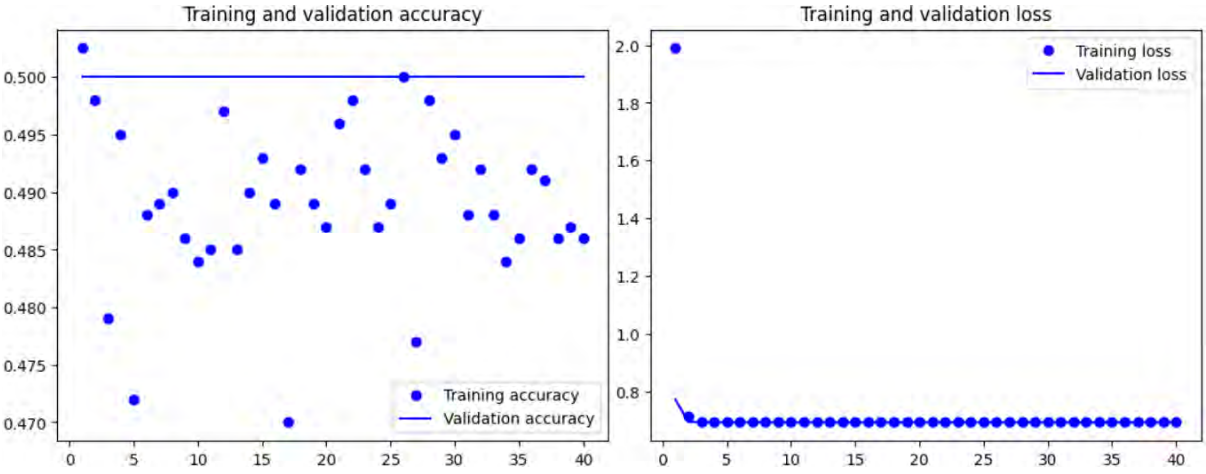

Imagen 53. División en Figuras 105 y 106.

Cuando el valor del hiperparámetro λ es excesivamente elevado, la restricción adicional impuesta también se intensifica, lo que simplifica en exceso la red y dificulta el proceso de entrenamiento al punto de evitar un aprendizaje efectivo.

Variante 2: λ = 0.001

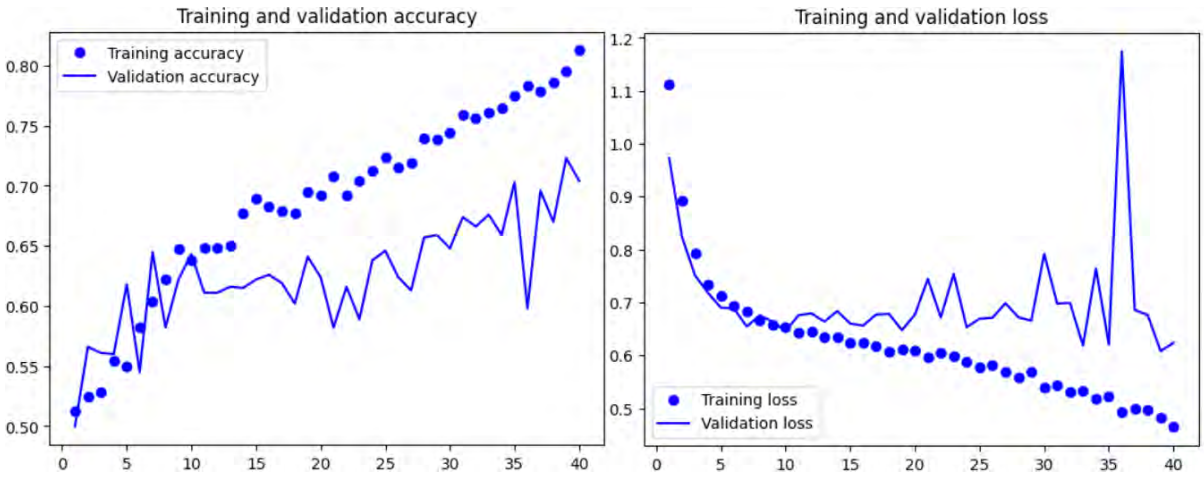

Imagen 54. División en Figuras 107 y 108.

La imagen 54 es sumamente interesante, ambas figuras tienen algo interesante a comentar. En ambas figuras los resultados son muy inferiores al original. Concretamente, el error de entrenamiento es sólo ligeramente inferior a 0.5 puntos, la precisión de entrenamiento apenas supera el 80%, el error de validación es superior a 0.6 puntos y la precisión de validación no alcanza el 70%.

Variante 3: λ = 0.0001 o inferior

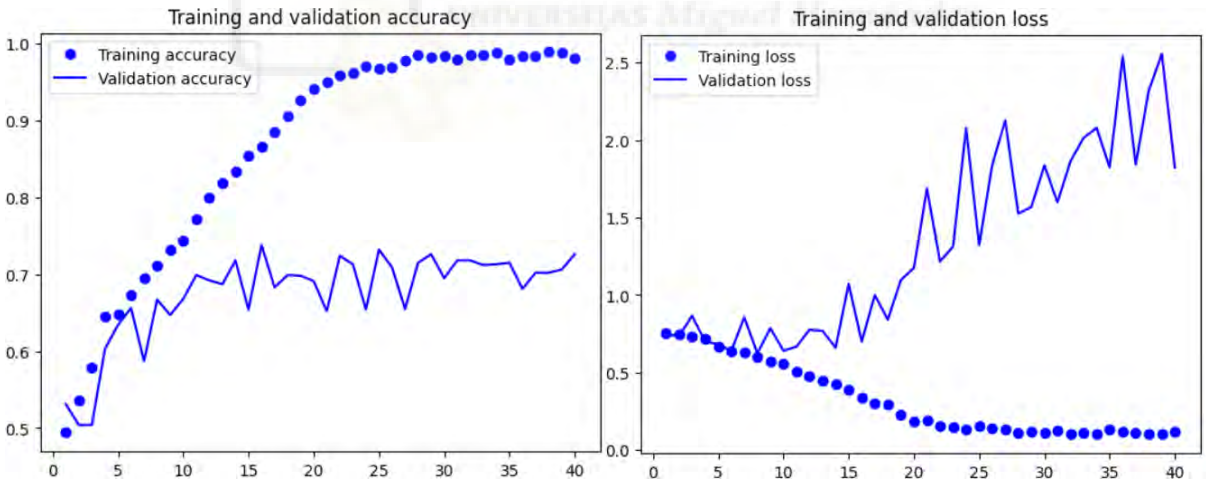

Imagen 55. División en Figuras 109 y 110.

Igual que en la segunda variante, la imagen 55 presenta unos datos inferiores al original, pero dado que la penalización es menor, estos datos se asemejan más a los de la red original.

Conclusión regularización L2

Dado que en todas las variantes la penalización deteriora significativamente el entrenamiento siendo los resultados de validación notablemente peores a los de la ConvNet original, esta técnica no es aplicable de forma exclusiva en este tamaño de red.

3. APLICACIÓN DE ESTRATÉGIAS CONTRA EL SOBREAJUSTE EN LA RED NEURONAL CONVOLUCIONAL DE "DOGS VS CATS"

Estas estrategias son combinaciones de dos o más técnicas Las diferentes estrategias pueden tener diferentes variaciones, pero sólo se incluirán las principales o más representativas.

3.1 PRIMERA ESTRATEGIA. AUMENTO DE DATOS Y "DROPOUT"

La primera estrategia surge de la combinación de dos de las técnicas más provechosas, el aumento de datos y el "dropout". Además, se le puede añadir la técnica "early stopping" o probar con "tanh" como función de activación del clasificador. Los parámetros del aumento de datos serán: espejo horizontal, rotación máxima de 0.1 y reescalado máximo de 0.2. Los diferentes argumentos con los que ejecutar el abandono son 0.25, 0.5 y 0.75. Estos argumentos dan pie a las diferentes variantes.

Variante 1: Dropout (0.5)

Parte 1: Técnicas de aumento de datos y abandono

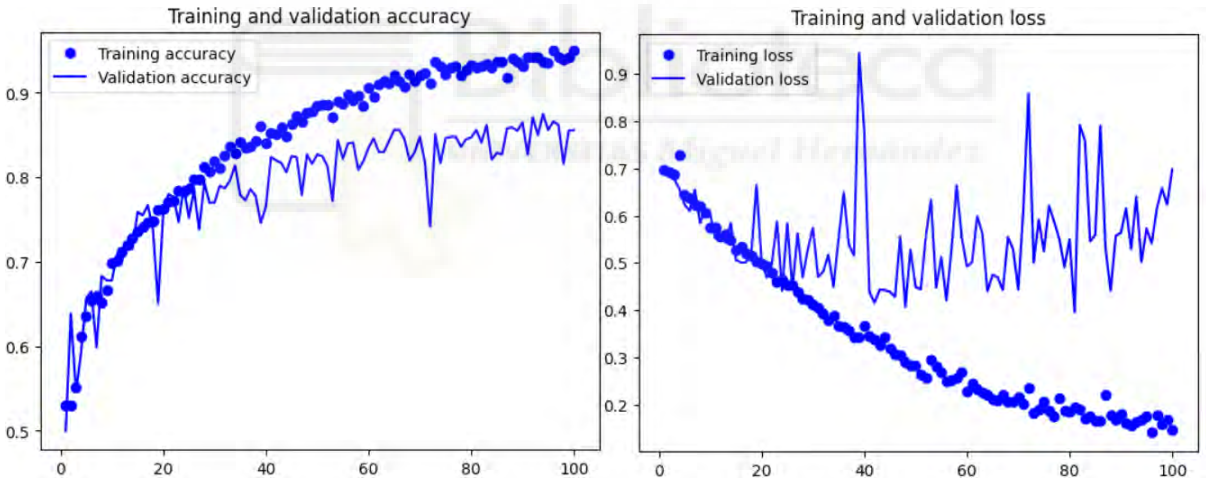

Imagen 56. División en Figuras 111 y 112

Hay una evidente mejora utilizando esta variante de la estrategia respecto al original o al uso de casi cualquier técnica, no solo se produce una disminución del sobreentrenamiento sino también un gran retraso de este y una mejora en los datos de validación.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
A.D. Dropout(0.5)	94.95%	86.6%	< 30	8.35%

Tabla 80.

La precisión de validación es en gran medida el justificante de la reducción del sobreajuste, ya que su máximo aumenta en un 14.5%. Pese a ello, la secuencia de entrenamiento también participa en la merma del sobreentrenamiento pues esta no alcanza el 95%, un 5% menor al original. Así mismo, el inicio del sobreajuste se posterga 25 épocas.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original.	0.0277	0.5899	< 5	0.5622
A.D. Dropout(0.5)	0.1456	0.4067	< 25	0.2611

Tabla 81

El sobreajuste se rebaja en tres décimas, una de ellas debido al error de entrenamiento y las otras dos por causa de la sucesión de validación. Excluyendo los grandes picos, el error de validación tiene una gráfica estable con trayectoria recta en torno a 0.55 puntos, en ese sentido es un caso poco ordinario.

Parte 2: Técnicas de aumento de datos, "tanh" como función de activación y abandono

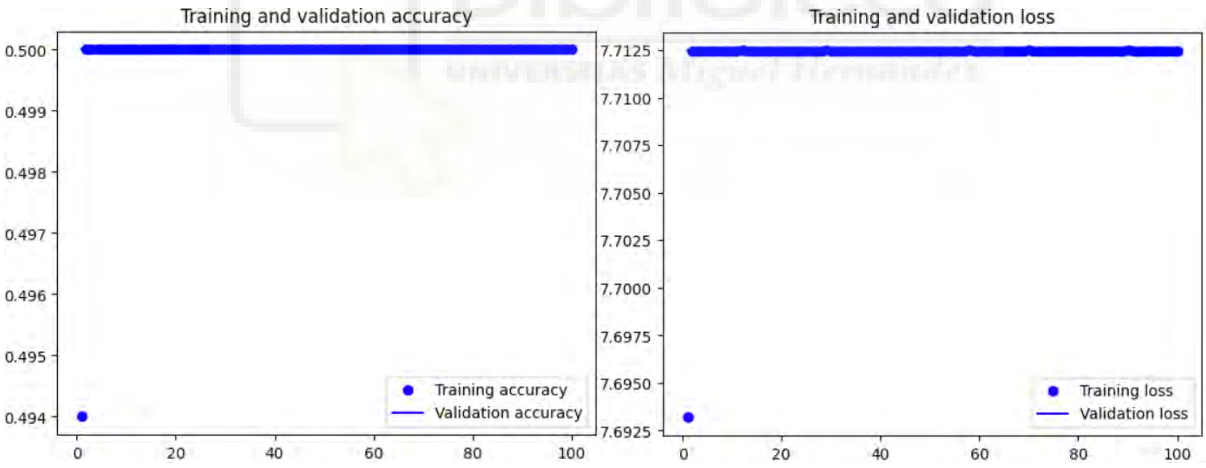

Imagen 57. División en Figuras 113 y 114.

Estas técnicas aplicadas con la función de activación "tanh" no funcionan. No consiguen entrenar siquiera. Dado que no es útil, no se probará en las siguientes variantes.

Parte 3: Técnicas de aumento de datos, "early stopping" y abandono

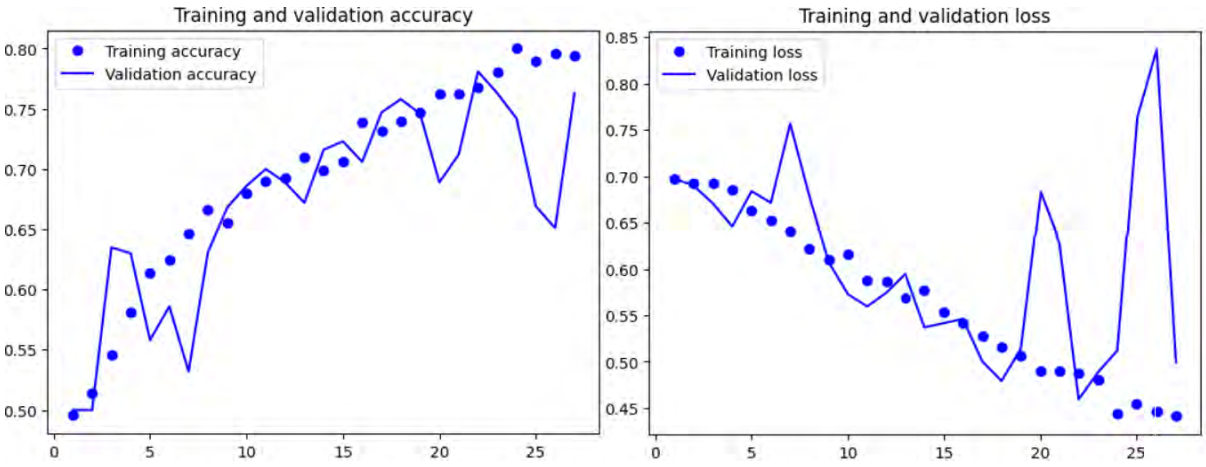

Imagen 58. División en Figuras 113 y 114.

Si es necesario limitar el número de épocas a 25, esta es una buena opción. Cierto es que no se alcanzan tan buenas cotas, pero una precisión de validación de 78.1% y un error de validación de 0.4594 puntos son unas buenas mejoras respecto al original teniendo en cuenta el escaso entrenamiento limitado.

Variante 2: Dropout (0.25)

Parte 1: Técnicas de aumento de datos y abandono.

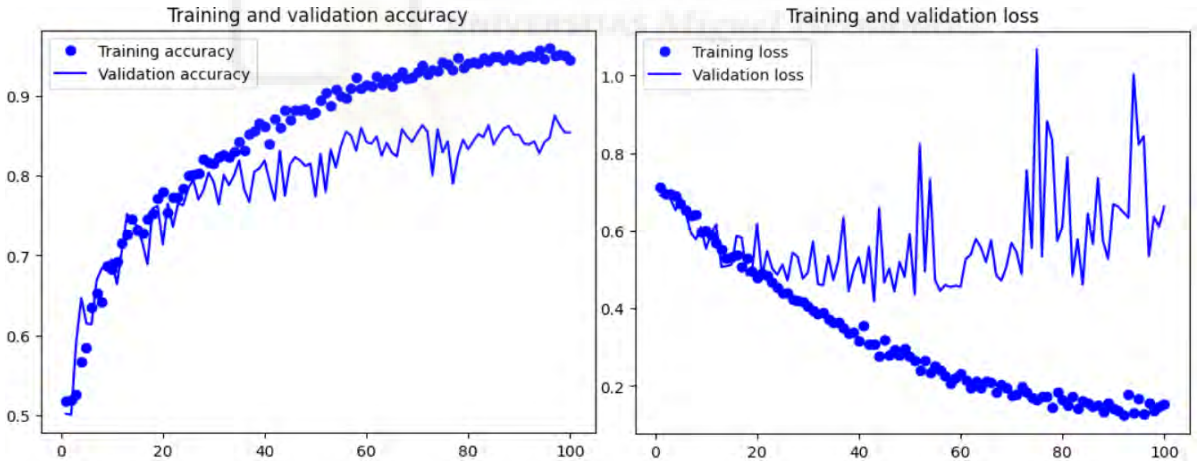

Imagen 57. División en Figuras 113 y 114

Igual que con la primera variante hay una evidente mejora utilizando esta variante de la estrategia respecto al original o al uso de casi cualquier técnica, se reduce el sobreajuste y se linealiza la figura 114.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión

A.D. Dropout(0.5)	94.95%	86.6%	< 30	8.35%
A.D. Dropout(0.25)	95.9%	87.5%	< 30	8.4%

Tabla 81.

El "overfitting" de precisión se mantiene igual respecto a la primera variante, sin embargo, tanto la serie de entrenamiento como la de validación mejoran en un 1%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. Dropout(0.5)	0.1456	0.4067	< 25	0.2611
A.D. Dropout(0.25)	0.1241	0.4361	< 25	0.312

Tabla 82.

En cuanto al error, si hay una cierta discrepancia en el sobreajuste valorada en 0.0509 puntos. No solo se debe a una mejora en el conjunto de entrenamiento, sino a un empeoramiento de aproximadamente 0.03 en la serie de validación.

Parte 2: Técnicas de aumento de datos, "early stopping" y abandono

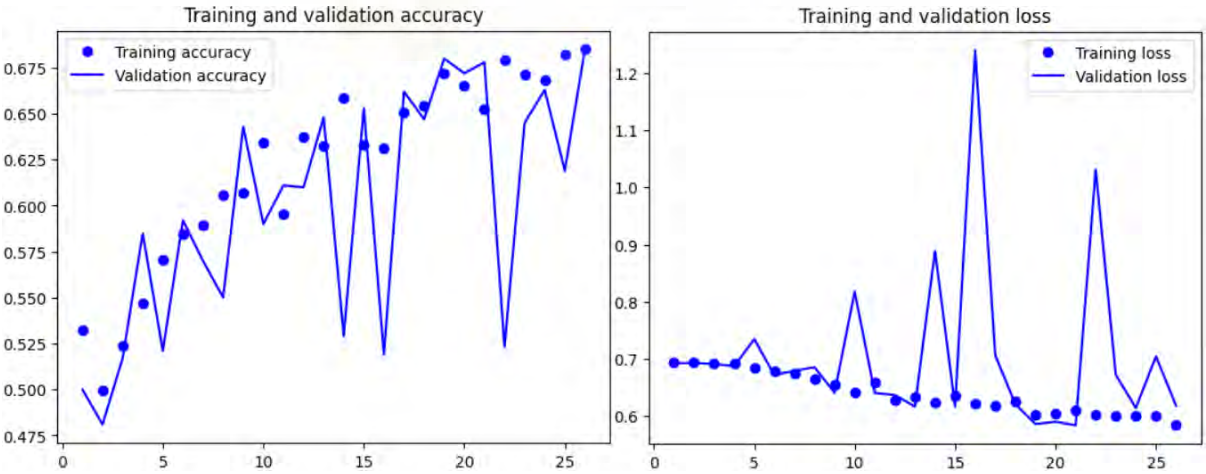

Imagen 58. División en Figuras 115 y 116

Aplicar la técnica de la parada temprana en el contexto de abandono a 0.25 y aumento de datos no es funcional. Pese a que no hay sobreajuste, la precisión de validación no alcanza el 70% y todo el error es superior a 0.6 puntos.

Variante 3: Dropout (0.75)

Parte 1: Técnicas de aumento de datos y abandono.

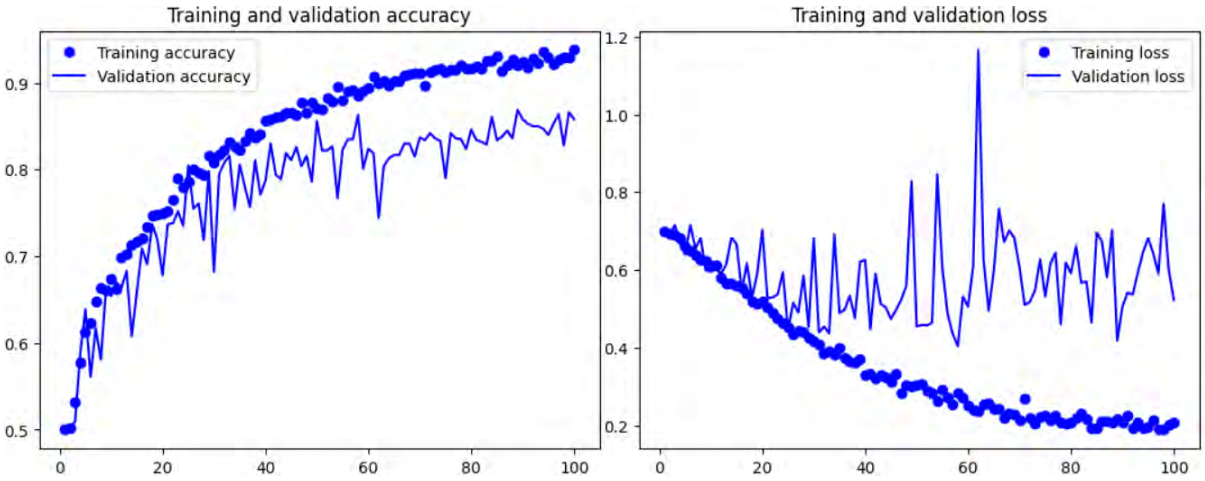

Imagen 59. División en Figuras 117 y 118

Visualmente puede parecer que se ha reducido el "overfitting" exageradamente y así es al compararlo con el resultado de la red original. En cambio, con respecto a los resultados producidos al ejecutar la primera variable de esta técnica, la disminución del sobreajuste no es tan significativa.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. Dropout(0.5)	94.95%	86.6%	< 30	8.35%
A.D. Dropout(0.75)	93.8%	86.95%	< 30	6.85%

Tabla 83

El sobreajuste de precisión se reduce en un 1.5% y la secuencia de validación solo aumenta en un 0.35%. Esto significa que la gran parte del diferencial del sobreajuste se debe a la serie de entrenamiento, la cual disminuye en un 1.15%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. Dropout(0.5)	0.1456	0.4067	< 25	0.2611
A.D. Dropout(0.75)	0.1886	0.4175	< 30	0.2289

Tabla 84

La serie de validación aumenta en poco más de una centésima, no obstante, el sobreentrenamiento disminuye debido al error de entrenamiento en 0.043 puntos.

Parte 2: Técnicas de aumento de datos, "early stopping" y abandono

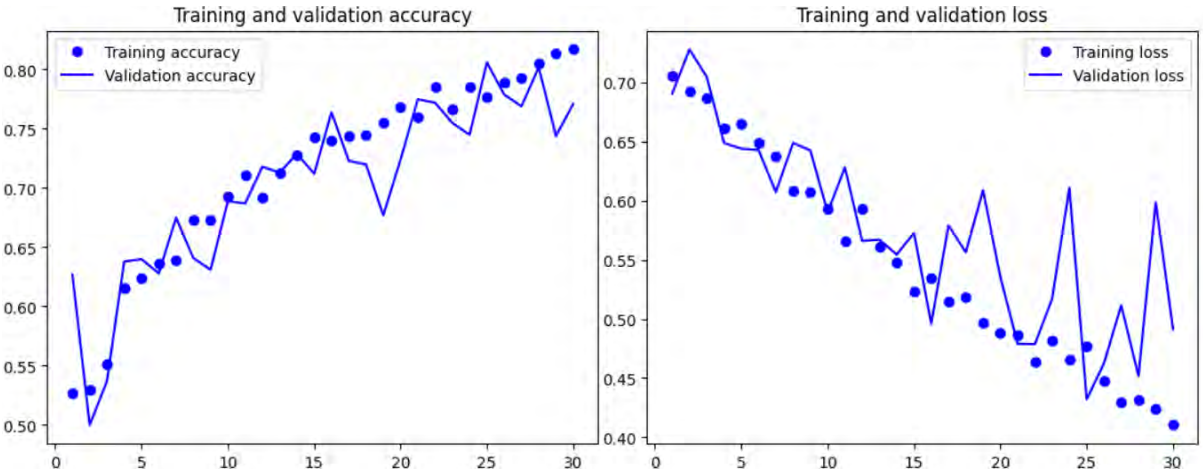

Imagen 60. División en Figuras 119 y 120

Si es necesario limitar el número de épocas a unas 30, esta es una buena opción, cierto es que no se alcanzan tan buenas cotas, pero una precisión de validación de 81.7% y un error de validación de 0.432 puntos son unas buenas mejoras respecto al original teniendo en cuenta el escaso entrenamiento.

3.2 SEGUNDA ESTRATEGIA. AUMENTO DE DATOS Y REGULARIZACIÓN

La segunda estrategia surge de la combinación de dos de las técnicas más provechosas, el aumento de datos y las técnicas de regularización. Además, se le puede añadir la técnica "early stopping". Los parámetros del aumento de datos serán: espejo horizontal, rotación máxima de 0.1 y reescalado máximo de 0.2.

3.2.1. AUMENTO DE DATOS Y REGULARIZACIÓN L1

Variante 1: λ=0.0001 o superior

Cuando el valor del hiperparámetro λ de la regularización es excesivamente elevado, la restricción adicional impuesta a la función de aprendizaje también se intensifica, lo que simplifica en exceso la red y dificulta el proceso de entrenamiento al punto de evitar un aprendizaje efectivo.

Variante 2: λ=0.00001

Parte 1: Técnicas de aumento de datos y regularización L1.

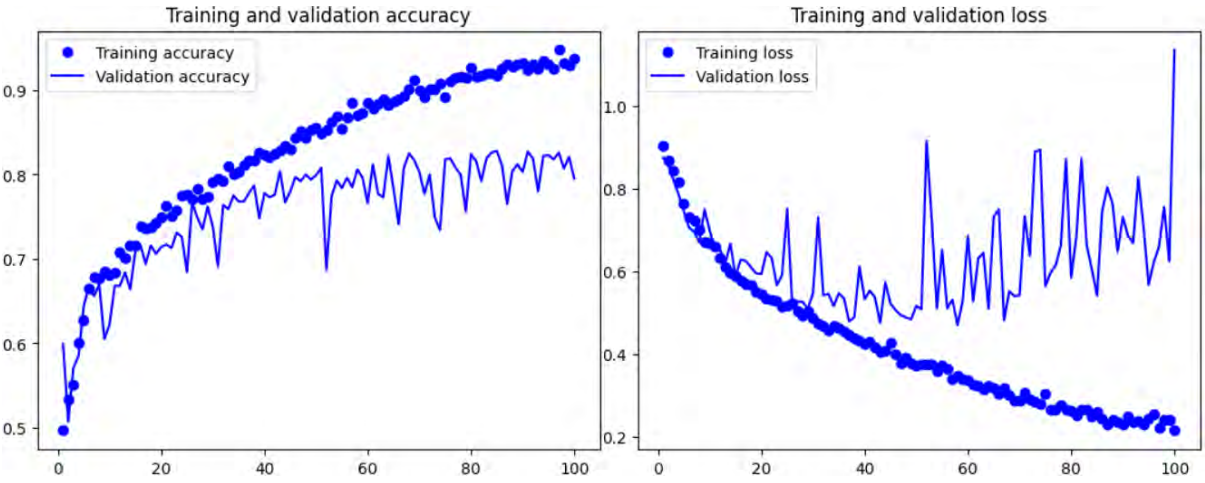

Imagen 60. División en Figuras 119 y 120

Hay una evidente mejora implementando esta variante de la segunda estrategia respecto al original, no solo se produce una disminución del "overfitting" sino también un gran retraso de este y una mejora considerable en los datos de validación. No obstante, respecto a la primera estrategia estos resultados son visiblemente peores.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
Original	100%	72.1%	< 5	27.9%
A.D. R.L1 (*λ=0.00001)*	94.75%	82.8%	< 25	11.95%

Tabla 85

La precisión de entrenamiento logra aproximarse al 95% comenzando a estabilizarse en ese entorno, por otro lado, la precisión de validación se estabiliza en porcentajes próximos al 81.5%. Ambas series participan en la disminución del sobreajuste.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
Original	0.0277	0.5899	< 5	0.5622
A.D. R.L1 (*λ=0.00001)*	0.2154	0.4791	< 30	0.2637

Tabla 86

En cuanto al error de entrenamiento, no se logra atenuar por debajo de 0.2 puntos. La secuencia de validación tiene el mínimo en 0.4791 tras el cual aumenta hasta superar el punto. Así pues,

el sobreajuste mínimo es de 0.2637 y el máximo supera los 0.78 puntos.

Parte 2: Técnicas de aumento de datos, "early stopping" y regularización L1.

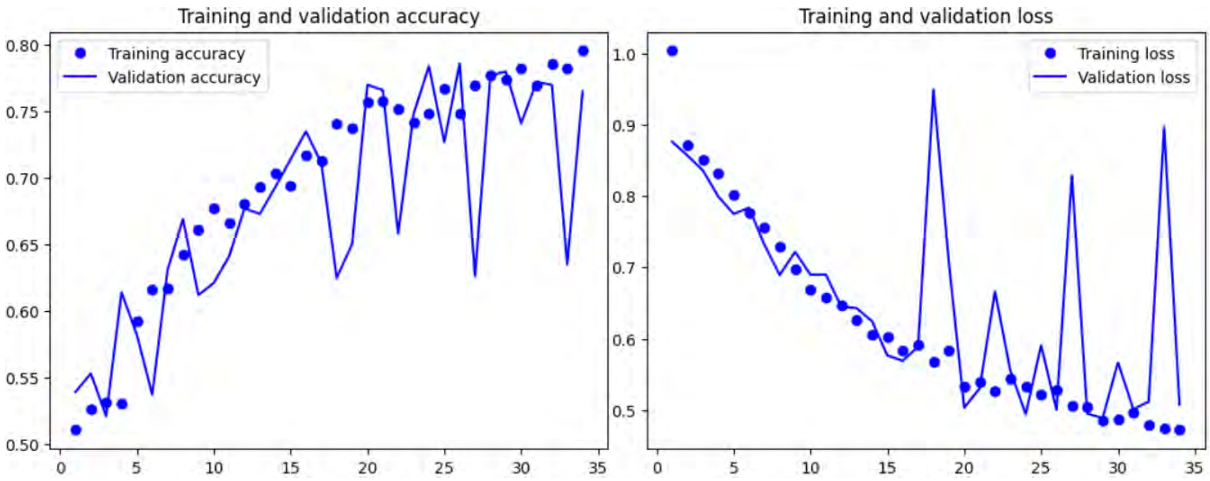

Imagen 61. División en Figuras 121 y 122

Si es necesario limitar el número de épocas a unas 35, esta es una opción con aumento de datos correcta. Cierto es que no se alcanzan tan buenas cotas como otras opciones, pero una precisión de validación de 78.6% y un error de validación de 0.4888 puntos son unas buenas mejoras respecto al original.

Variante 3: λ=0.000001 o inferior

Parte 1: Técnicas de aumento de datos y regularización L1.

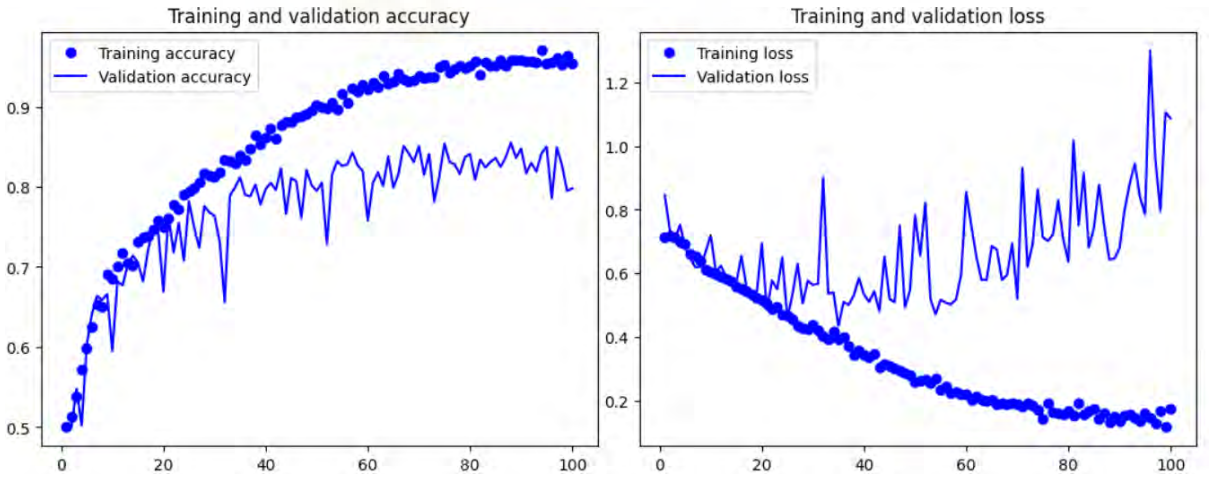

Imagen 62. División en Figuras 123 y 124

Esta es la última variante de esta estrategia pues el parámetro λ ya tiene muy pequeño valor. A medida que el parámetro se reduce, la regularización pierde efecto teniendo siempre resultados de validación muy similares a los obtenidos aplicando únicamente la técnica del aumento de datos.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. R.L1 (*λ=0.00001)*	94.75%	82.8%	< 25	11.95%
A.D. R.L1 (*λ=0.000001)*	97%	85.5%	<25	11.5%

Tabla 87

El sobreentrenamiento difiere en un 0.45%, porcentaje que entra dentro de la variabilidad propia del entrenamiento. Esto significa que los sobreajustes se pueden asumir como iguales. No obstante, un mismo sobreajuste no significa unas mismas series de validación y entrenamiento, sino un mismo diferencial entre ellas. El desfase producido en la figura 123 respecto a la figura 119 es de aproximadamente un 2.5%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. R.L1 (*λ=0.00001)*	0.2154	0.4791	< 30	0.2637
A.D. R.L1 (*λ=0.000001)*	0.1164	0.4355	<30	0.3191

Tabla 88

En relación al error, el "overfitting" si difiere considerablemente, el de esta variable aumenta en 0.0554 puntos. Este mal resultado se debe a que los datos de validación (-0.0436 puntos) no mejoran tanto como los de entrenamiento (-0.099 puntos). Así pues, este es un caso donde el sobreajuste aumenta consiguiendo mejores datos de validación.

Parte 2: Técnicas de aumento de datos, "early stopping" y regularización L1.

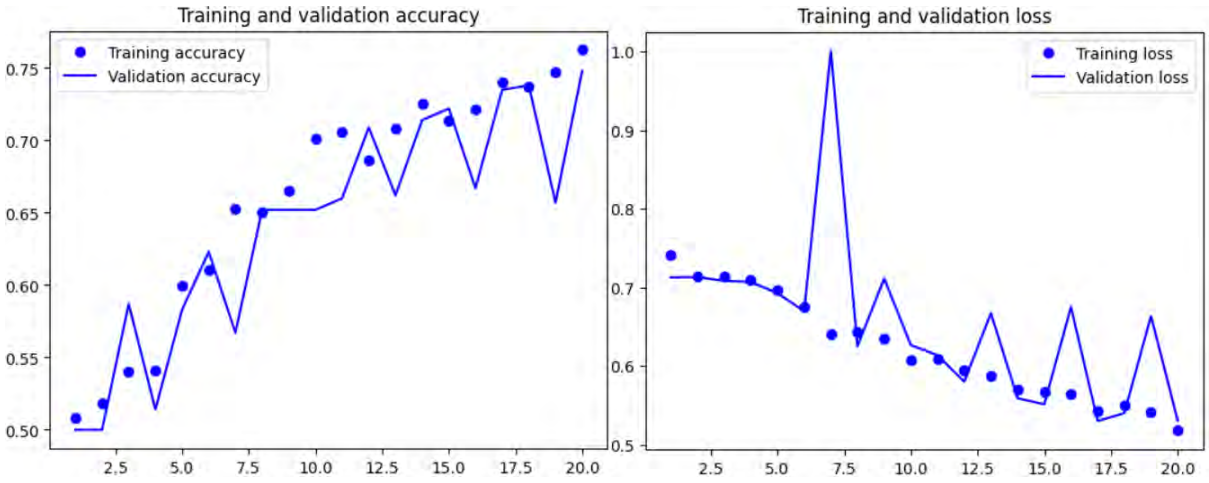

Imagen 63. División en Figuras 125 y 126

Dado que los datos de validación son superiores a los datos de la red convolucional original, el sobreajuste se elimina y se produce en tan solo 20 épocas; esta puede ser una opción a considerar. Sin embargo, un 74.8% de precisión máxima y un error mínimo de 0.5298 son datos a mejorar.

3.2.2. AUMENTO DE DATOS Y REGULARIZACIÓN L2

Variante 1: λ=0.01 o superior

Del mismo modo que la regularización L1, cuando el valor del hiperparámetro λ de la regularización L2 es excesivamente elevado, la restricción adicional impuesta a la función de aprendizaje también se intensifica, lo que simplifica en exceso la red y dificulta el proceso de entrenamiento al punto de evitar un aprendizaje efectivo.

Variante 2: λ=0.001

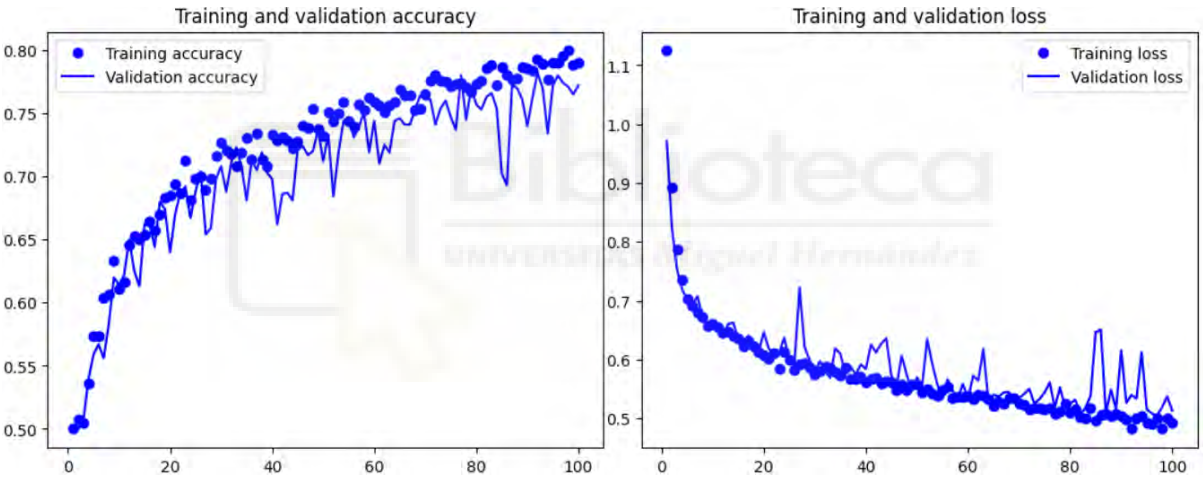

Imagen 64. División en Figuras 127 y 128

Con esta variante se logra el principal objetivo, evitar el sobreajuste. Los otros objetivos, que incluyen alcanzar una precisión del 100% en las diversas curvas y lograr que la puntuación final de las curvas de error resulten en 0, no se consiguen.

Gráfica de error	Precisión de entrenamiento	Precisión de validación	Error de entrenamiento	Error de validación
A.D. R.L2 *(λ=0.001)*	79.95%	78.2%	0.4820	0.5044

Tabla 89

El eludir el sobreentrenamiento se ve ensombrecido por la escasa precisión y el excesivo error de validación. La precisión es tan escasa que no supera el 80%. Por otro lado, el error sólo mejora 8 centésimas el resultado de la red original.

Variante 3: λ=0.0001

Parte 1: Técnicas de aumento de datos y regularización L2.

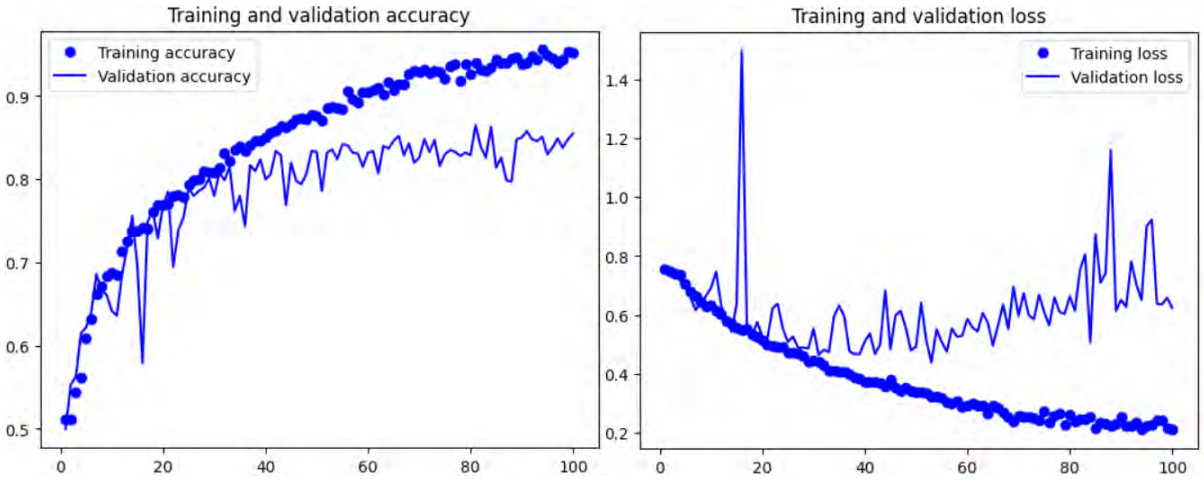

Imagen 65. División en Figuras 129 y 130

Pese a producir "overfitting", en algunos sentidos esta variante mejora a la anterior. Principalmente se potencia los datos de validación, especialmente los de precisión. Además, el sobreajuste no se elimina, pero si se reduce bastante.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. R.L2 (*λ=0.001)*	79.95%	78.2%	- - - -	- - - -
A.D. R.L2 (*λ=0.0001)*	95.55%	86.5%	>30	9.05%

Tabla 90

No solo el máximo de la precisión de validación es bastante bueno, sino que finaliza el entrenamiento en el entorno del 84%. En lo relativo a los datos de entrenamiento; el máximo se mantiene alejado del objetivo, pero no tanto como la variante anterior (+ 15.6 %).

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. R.L2 (*λ=0.001)*	0.4820	0.5044	- - - -	- - - -
A.D. R.L2 (*λ=0.0001)*	0.2104	0.4643	< 30	0.2539

Tabla 91

0.2539 puntos es un sobreentrenamiento relativamente elevado, sin embargo y pese a no ser buenos datos se produce un fortalecimiento tanto en el entrenamiento como en la validación.

Parte 2: Técnicas de aumento de datos, "early stopping" y regularización L1.

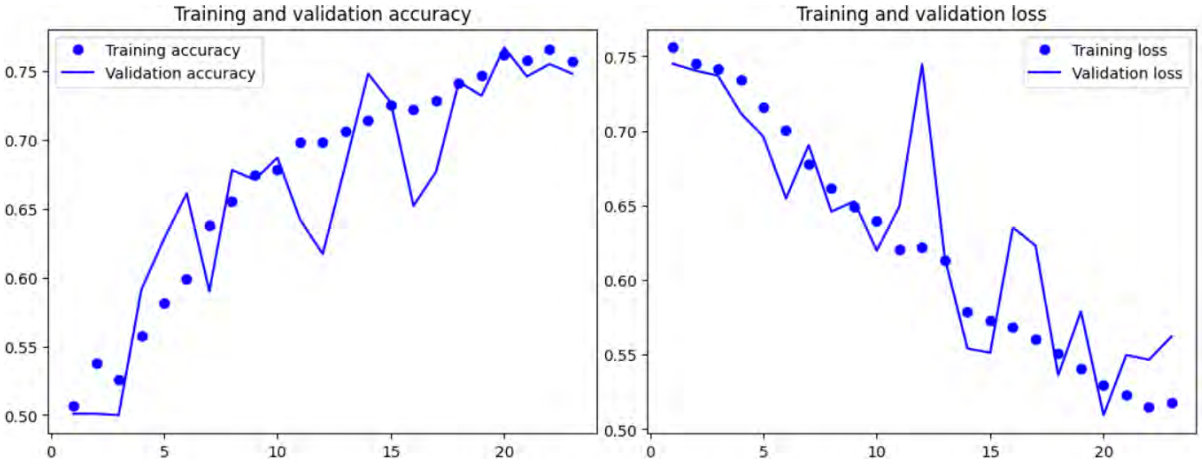

Imagen 66. División en Figuras 131 y 132

El único motivo para aplicar la técnica "early stopping" en esta estrategia es limitar el número de épocas de entrenamiento, pues la segunda variante ya evita totalmente el sobreajuste con valores similares.

Variante 3: λ=0.00001

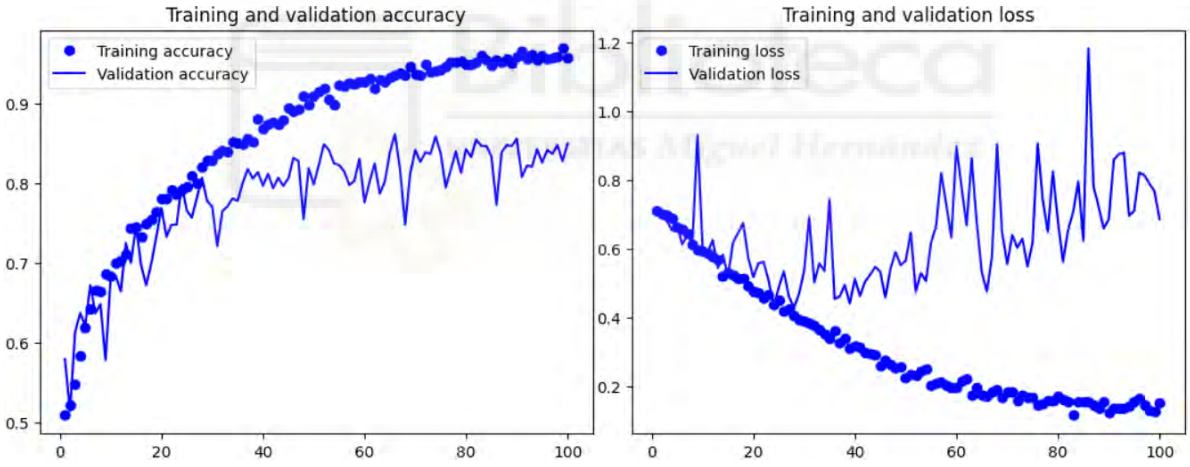

Imagen 67. División en Figuras 133 y 134

Esta es la última variante de esta estrategia pues el parámetro λ ya tiene muy pequeño valor. A medida que el parámetro se reduce, la regularización pierde efecto teniendo siempre resultados muy similares a los obtenidos aplicando únicamente la técnica del aumento de datos.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D.	94.36%	85.7%	>30	8.66%
A.D. R.L2 *(λ=0.00001)*	96.95%	85.6%	< 30	11.35%

Tabla 92

Dado que los datos de validación son muy similares, la diferencia en el sobreajuste de precisión se debe enteramente a la serie de entrenamiento.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573
A.D. R.L2 *(λ=0.0001)*	0.1265	0.4135	< 30	0.287

Tabla 93

Tanto los datos de entrenamiento como los de validación se aproximan a los del aumento de datos, sin embargo, el diferencial sigue siendo relativamente elevado. Debido a estos diferenciales de unas dos centésimas cada uno el sobreajuste se agranda.

3.3. TERCERA ESTRATEGIA. AUMENTO DE DATOS, ABANDONO Y REGULARIZACIÓN

La tercera estrategia surge de la combinación de las tres técnicas más provechosas, el aumento de datos, el "dropout" y la regularización. Adicionalmente, podría considerarse la inclusión de la técnica "early stopping" o experimentar con la función de activación "tanh" para el clasificador. Sin embargo, debido a que en las estrategias anteriores no se encontraron diferencias significativas al implementar estas técnicas, en esta estrategia se optará por no ejecutarlas con objeto de limitar el contenido y la extensión del proyecto.

Al igual que en las estrategias previas los parámetros del aumento de datos serán: espejo horizontal, rotación máxima de 0.1 y reescalado máximo de 0.2. Los diferentes argumentos con los que ejecutar el abandono son 0.25, 0.5 y 0.75. La regularización L1 se utilizará con λ=0.00001 y la regularización L2 con λ=0.0001, pues estos son los parámetros más elevados con resultados óptimos.

3.3.1. AUMENTO DE DATOS, "DROPOUT" Y REGULARIZACIÓN L1

Variante 1: .dropout(0.5)

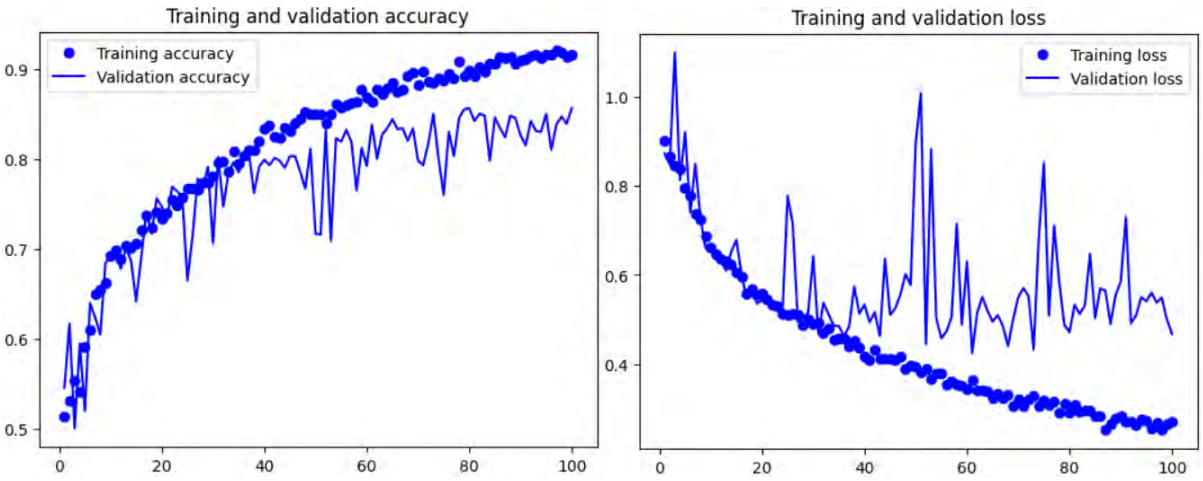

Imagen 68. División en Figuras 135 y 136

El sobreajuste se reduce bastante y su aparición se posterga en comparación con los datos originales. El sobreajuste inicia en la época 30, pero en la gráfica 135 se observa una convergencia de ambas series en la época 50. La comparación en las tablas se lleva a cabo en relación con el conjunto de datos aumentados.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D.	94.36%	85.7%	>30	8.66%
A.D. R.L1 dropout(0.5)	91.95%	87%	> 30	4.95%

Tabla 94

El sobreajuste de precisión se reduce a un porcentaje menor del 5% y los datos de validación se aumentan en casi un 1.5% respecto al aumento de datos. No obstante, estos éxitos se ven diluidos porque gran parte de la disminución del sobreajuste se debe a la serie de entrenamiento.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573
A.D. R.L1 dropout(0.5)	0.2539	0.4248	> 30	0.1709

Tabla 95

En este caso la reducción del “overfitting” se produce únicamente gracias al error de entrenamiento, hasta el punto de que el valor mínimo de la secuencia de validación de la red con la técnica de aumento de datos es menor al valor mínimo dado en esta variante.

Variante 2: .dropout(0.25)

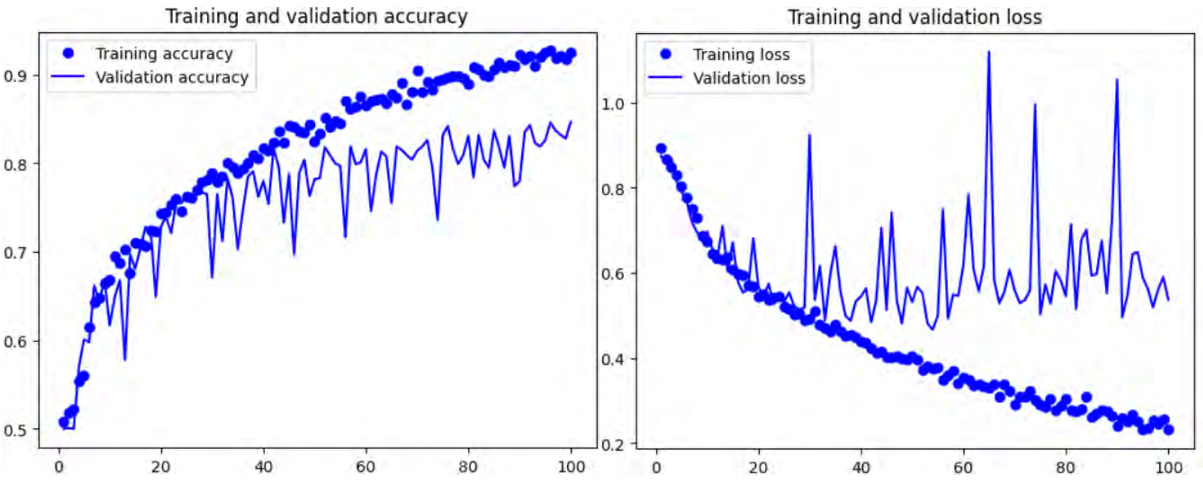

Imagen 69. División en Figuras 137 y 138

El sobreajuste se minimiza y retrasa respecto a los datos originales, dándose a partir de la época 35.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. R.L1 dropout(0.5)	91.95%	87%	> 30	4.95%
A.D. R.L1 dropout(0.25)	92.2%	84.7%	> 35	7.5%

Tabla 96

No sólo aumenta el sobreajuste en un 2.5%, sino que la precisión de validación disminuye más de un 2%. El único punto positivo es un retraso todavía mayor que el de la variante previa.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. R.L1 dropout(0.5)	0.2539	0.4248	> 30	0.1709
A.D. R.L1 dropout(0.25)	0.2315	0.4667	> 35	0.2352

Tabla 97

Pese a que el sobreentrenamiento se retrasa, aumenta en 0.05 puntos. Esto se debe principalmente al aumento del error de validación.

Variante 3: .dropout(0.75)

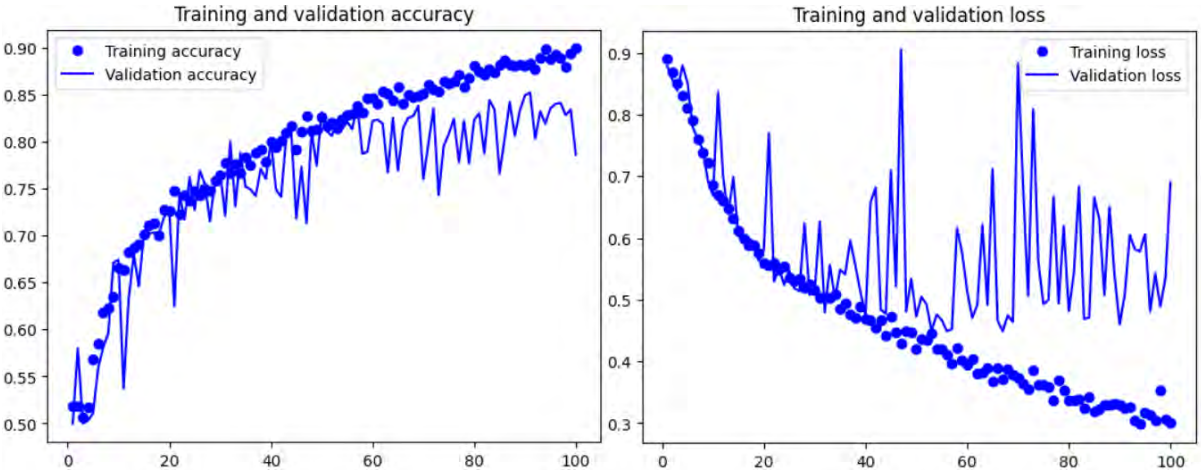

Imagen 70. División en Figuras 139 y 140

De la imagen 70 se pueden extraer dos datos destacados de manera sencilla. En primer lugar, se observa que el "overfitting" no solo disminuye en comparación con el modelo original, sino también en relación con la variante previa. En segundo lugar, es relevante señalar el considerable retraso en la aparición del sobreajuste, que se extiende hasta la época 60.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. R.L1 dropout(0.5)	91.95%	87%	> 30	4.95%
A.D. R.L1 dropout(7.5)	90%	85.2%	> 60	4.5%

Tabla 98

Con excepción del aplazamiento del inicio del sobreentrenamiento, todos los valores presentes en la tabla exhiben una tendencia decreciente. Aunque la disminución en el caso del sobreentrenamiento es un resultado positivo, este ligero descenso se origina debido al mayor desfase existente entre las series de entrenamiento. La última fase de la secuencia de validación se mantiene en torno al 83.5%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. R.L1 dropout(0.5)	0.2539	0.4248	> 30	0.1709
A.D. R.L1 dropout(0.75)	0.298	0.4491	< 60	0.1511

Tabla 99

Mientras que el error mínimo de entrenamiento aumenta en más de 0.04 unidades, el valor mínimo en la secuencia de validación solo se incrementa en 2 centésimas, lo que resulta en una disminución del sobreajuste.

3.3.2. AUMENTO DE DATOS, "DROPOUT" Y REGULARIZACIÓN L2

Variante 1: .dropout(0.5)

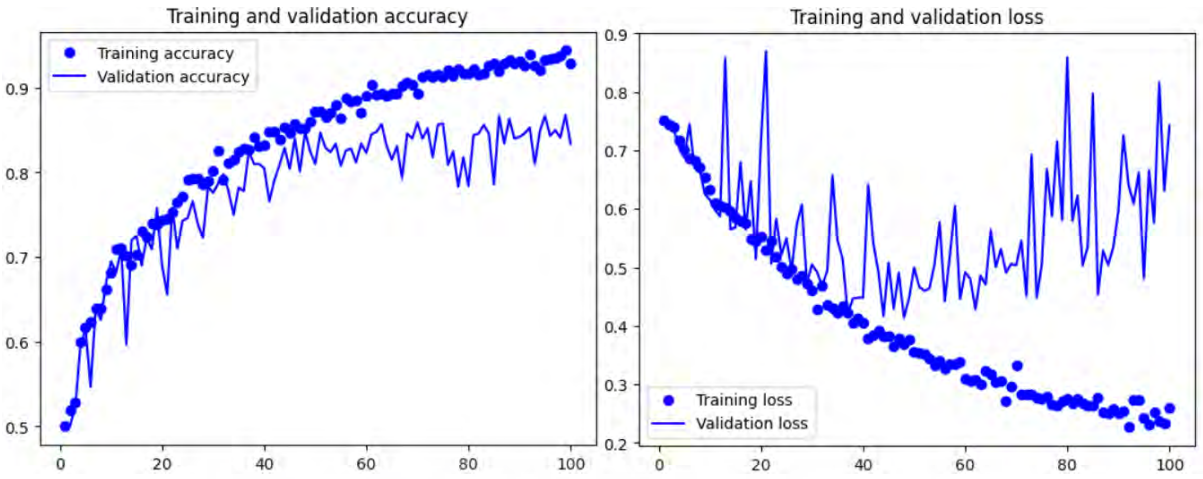

Imagen 71. División en Figuras 141 y 142

El sobreajuste se reduce mucho y su aparición se posterga en comparación con los datos originales. El sobreajuste inicia a partir de la época 35.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D.	94.36%	85.7%	>30	8.66%
A.D. R.L2 dropout(0.5)	94.4%	86.8%	> 45	7.6%

Tabla 100

Extraordinariamente, la precisión de entrenamiento es idéntica en ambas estrategias; por lo tanto, todas las diferencias en el sobreajuste provienen exclusivamente de la serie de validación. El puntaje máximo alcanza el 86.8%, lo que representa un aumento del 1% con respecto al valor de referencia. La evolución final en la serie de validación se mantiene en torno a valores cercanos al 85%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D.	0.1402	0.3975	< 30	0.2573

A.D. R.L2 dropout(0.5)	0.2266	0.414	< 40	0.1874

Tabla 101

El sobreajuste se reduce en 0.07 unidades, sin embargo, dado que el error de validación aumenta ligeramente, esta disminución se atribuye en su totalidad al agravamiento del error de entrenamiento.

Variante 2: .dropout(0.25)

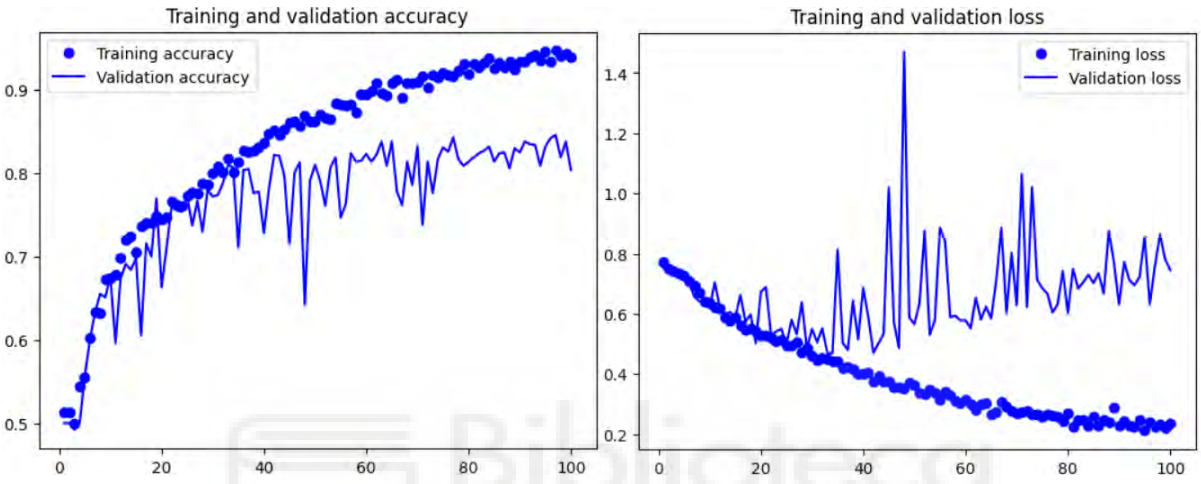

Imagen 72. División en Figuras 143 y 144

Comparado con el modelo original, la reducción en el "overfitting" es evidente, manifestándose aproximadamente 25 épocas más tarde. No obstante, respecto a la variante anterior, se observa un aumento en ambos casos de sobreajuste, y estos se materializan con menos épocas de entrenamiento.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. R.L2 dropout(0.5)	94.4%	86.8%	> 45	7.6%
A.D. R.L2 dropout(0.25)	94.7%	84.6%	> 30	10.1%

Tabla 102

El sobreentrenamiento aumenta en un 2.5% debido principalmente a la rebaja de la precisión de validación, la cual se reduce en un 2.2%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error

A.D. R.L2 dropout(0.5)	0.2266	0.414	< 40	0.1874
A.D. R.L2 dropout(0.25)	0.2117	0.4687	< 35	0.257

Tabla 103

En cuanto al sobreajuste de error, se registra un incremento de casi 7 centésimas. El error de validación aumenta en 0.0547 unidades. Así pues el resto del sobreentrenamiento se atribuye a una ligera mejora de la serie de entrenamiento.

Variante 2: .dropout(0.75)

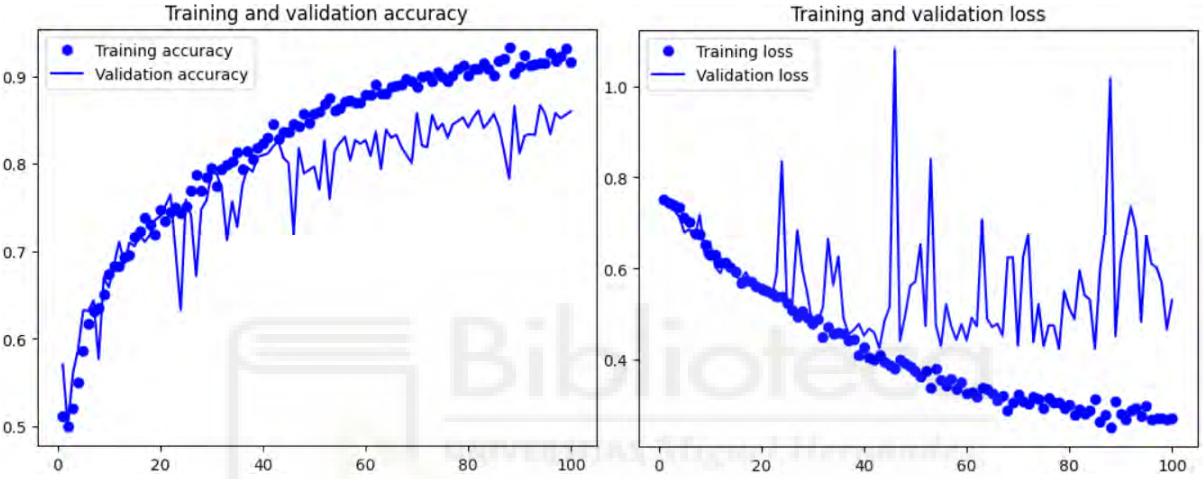

Imagen 73. División en Figuras 145 y 146

En comparación con el modelo original, la disminución del "overfitting" es clara y se hace evidente aproximadamente 35 épocas después. Sin embargo, respecto a la primera variante, se observa una ligera mejora.

Gráfica de precisión	Precisión de entrenamiento	Precisión de validación	Época de sobreajuste	Sobreajuste de precisión
A.D. R.L2 dropout(0.5)	94.4%	86.8%	> 45	7.6%
A.D. R.L2 dropout(0.75)	93.25%	86.8%	> 30	6.45%

Tabla 104

Extraordinariamente, la precisión de validación es idéntica en ambas estrategias; por lo tanto, todas las diferencias en el sobreajuste provienen exclusivamente de la serie de entrenamiento. El puntaje máximo alcanza el 93.25%, lo que representa un decrecimiento del 1.15% con

respecto al valor de referencia, sin embargo, la tendencia sigue aumentando. Respecto a la trayectoria de validación, esta se estabiliza en valores cercanos al 85.5%.

Gráfica de error	Error de entrenamiento	Error de validación	Época de sobreajuste	Sobreajuste de error
A.D. R.L2 dropout(0.5)	0.2266	0.414	< 40	0.1874
A.D. R.L2 dropout(0.75)	0.2511	0.4244	< 35	0.1733

Tabla 105

Tanto la serie de entrenamiento como la de validación aumentan. Sin embargo, el diferencial permite una disminución del sobreajuste de error en 0.0141 puntos.

4. ANÁLISIS COMPARATIVO DE LAS DIFERENTES TÉCNICAS Y ESTRATEGIAS CONTRA EL SOBREAJUSTE EN LA RED NEURONAL DE "DOGS VS CATS"

A continuación, se pretende hacer un pequeño análisis comparativo de las técnicas y estrategias comentadas en este capítulo.

Gráfica mixta	Precisión de entrenamiento	Precisión de validación	Error de entrenamiento	Error de validación
Original	100%	72.1%	0.0277	0.5899
Early Stopping	77.8%	70%	0.4711	0.5919
.Dropout(0.5)	98.65%	79%	0.0466	0.5092
.Dropout(0.25)	99.2%	75.8%	0.0304	0.6121
.Dropout(0.75)	98.45%	81%	0.0545	0.5131
Capas eliminadas	100%	69.6%	0.0001	0.5945
Capas añadidas	99.3%	72.5%	0.0388	0.5749
Tanh	97.45%	77.6%	0.1604	0.6296

A.D.	94.36%	85.7%	0.1402	0.3975
Flip Vertical	93.25%	75.63%	0.1978	0.583
Rotación máxima = 0.01	97.89%	84.21%	0.0714	0.4286
Rotación máxima = 1	85.65%	78.73%	0.2037	0.4813
Zoom máximo = 0.02	97%	86.12%	0.085	0.4666
R. L1: $\lambda = 0.00001$	99.2%	76%	0.1033	0.6017
R. L1: $\lambda = 0.000001$	99.5%	76.6%	0.0534	0.5763
A.D. Dropout(0.5)	94.95%	86.6%	0.1456	0.4067
A.D. Dropout(0.25)	95.9%	87.5%	0.1241	0.4361
A.D. Dropout(0.75)	93.8%	86.95%	0.1886	0.4175
A.D. R.L1 ($\lambda=0.00001$)	94.75%	82.8%	0.2154	0.4791
A.D. R.L1 ($\lambda=0.000001$)	97%	85.5%	0.1164	0.4355
A.D. R.L2 ($\lambda=0.001$)	79.95%	78.2%	0.4820	0.5044
A.D. R.L2 ($\lambda=0.0001$)	95.55%	86.5%	0.2104	0.4643
A.D. R.L2 ($\lambda=0.00001$)	96.95%	85.6%	0.1265	0.4135
A.D. R.L1 dropout(0.5)	91.95%	87%	0.2539	0.4248
A.D. R.L1 dropout(0.25)	92.2%	84.7%	0.2315	0.4667

A.D. R.L1 dropout(7.5)	90%	85.2%	0.298	0.4491
A.D. R.L2 dropout(0.5)	94.4%	86.8%	0.2266	0.414
A.D. R.L2 dropout(0.25)	94.7%	84.6%	0.2511	0.4244
A.D. R.L2 dropout(0.75)	93.25%	86.8%	0.2511	0.4244

Tabla 106

Si el único propósito es eliminar el sobreajuste en este tipo de red, se puede optar por cualquier variante de las tres estrategias combinándolas con la técnica de "early stopping", o elegir la opción de la segunda estrategia que involucra la regularización L2 con λ=0.001. Sin embargo, al seleccionar estas alternativas, es importante considerar que el tiempo de entrenamiento se reduce, lo que puede afectar la consecución de niveles de rendimiento más elevados.

Como eliminar el sobreentrenamiento resulta negativo en cuanto al rendimiento, tal vez es más apropiado simplemente mitigar el “overfitting”. En este contexto, las alternativas más efectivas son las variantes 1 (.Dropout(0.5)) y 3 (.Dropout(0.75)) de la tercera estrategia con regularización L1. La elección más favorable recae en la primera variante, ya que ambas cifras de validación son superiores.

Si el objetivo consiste en maximizar la precisión de validación, la opción óptima es la segunda variante de la primera estrategia (87.5). Otra opción viable es la mencionada anteriormente por el escaso sobreajuste que presenta: la primera variante de la tercera estrategia con regularización L1 (87%). Esta alternativa posee datos de error más favorables.

Si la meta es minimizar el error de validación la elección más adecuada es la técnica de aumento de datos (0.3975 p). Otra alternativa sólida es la primera variante de la primera estrategia (0.4067 p). Es necesario tener en cuenta que, a pesar de que la disparidad en el error es pequeña, la diferencia en la precisión favorece considerablemente a la segunda opción.

CAPÍTULO 6

CONCLUSIONES Y TRABAJOS FUTUROS

1. REFLEXIONES FINALES

El “overfitting” es el principal problema asociado al “deep learning” y a las redes neuronales convolucionales. Existen diversos factores que pueden causar dicho fenómeno. Los dos factores más notables son la carencia de datos de entrada y una complejidad excesiva en el modelo. De esta manera, las técnicas y estrategias utilizadas para prevenir el sobreajuste pretenden solventar al menos uno de estos problemas de forma efectiva.

Si la red convolucional diseñada tiene el tamaño adecuado para la aplicación requerida, las técnicas más efectivas incluyen el aumento de datos, el abandono y la regularización, con la precaución de que la última es especialmente eficaz cuando se incrementa el tamaño de la red. El aumento de datos, tal como su nombre sugiere, es una técnica que apunta a resolver el primer problema, la carencia de datos. Por otra parte, las otras dos técnicas, el “dropout” y la regularización, persiguen afrontar el segundo problema, la excesiva complejidad del modelo.

Puesto que esas son las técnicas más efectivas y hay dos factores que evitar, combinarlas adecuadamente es la mejor estrategia posible. Por consiguiente, en toda estrategia, es fundamental incluir al menos una técnica dirigida a evitar cada uno de estos problemas. La primera estrategia adoptada, incluye el aumento de datos y el abandono, es la mejor opción en redes de menor envergadura, donde la regularización resulta ineficaz. En cambio, la segunda estrategia implementada, compuesta por el aumento de datos y la regularización, no es tan adecuada para redes de pequeño y mediano tamaño. Por último, la tercera estrategia combina las tres técnicas y emerge como una solución acertada para redes de tamaño medio.

El primer escenario de estudio utilizado para examinar las diversas técnicas y estrategias consiste en una red convolucional de pequeñas dimensiones diseñada para la clasificación de dígitos en el conjunto de datos MNIST. En términos generales, la elección más acertada para esta red es la primera variante de la primera estrategia, es decir, aplicar las técnicas del aumento de datos y el “dropout” con un valor de 0.5 como argumento. Esta opción satisface

prácticamente todos los requisitos; elimina los sobreentrenamientos, maximiza la precisión y disminuye notablemente el error.

El segundo y último caso de estudio involucra una red de dimensiones superiores a la anterior diseñada para clasificar binariamente el conjunto “dogs-vs-cats”. En términos generales hay dos buenas alternativas para mejorar la red. Una de ellas es la primera variante de la tercera estrategia con regularización L1. En otras palabras, aplicar en la ConvNet las técnicas de aumento de datos, regularización L1 con λ=0.00001 y “dropout” con un valor de 0.5 como argumento. Esta elección mitiga el sobreajuste en gran medida, el de precisión se limita a 4.95% y el de error a 0.1709 puntos. Pese a maximizar la precisión de validación (87%), los resultados del error de validación no son óptimos (0.4248 p).
La segunda alternativa es la primera variante de la primera estrategia. A saber, aplicar las técnicas del aumento de datos y el “dropout” con un valor de 0.5 como argumento. La característica mejor optimizada de esta alternativa es la serie del error de validación (0.4067 p). Aunque tampoco es despreciable la optimización de la precisión de validación, que alcanza un máximo de 86.6%. Sin embargo, es importante destacar que esta opción experimenta un grado significativo de sobreajuste en comparación con la otra alternativa.

En conclusión, las mejores técnicas para la reducción del sobreajuste en redes neuronales convolucionales de pequeño tamaño son el aumento de datos y el abandono. Según se agranda la red la técnica de la regularización es más eficaz hasta incluirse en el conjunto de las mejores técnicas.

2. TRABAJOS FUTUROS

Existen muchas formas de ampliar el proyecto, pero principalmente hay dos, analizar más técnicas y analizar estas técnicas en otras aplicaciones y redes convolucionales.

Posibles técnicas a añadir.

1. Técnicas de eliminación de ruido. En algunos casos el sobreentrenamiento puede deberse a una mala limpieza de los datos. Si los datos contienen ruido puede dificultar la tarea de aprendizaje de las redes neuronales convolucionales.
2. Técnicas de “Transfer learning”. Utilizar partes de redes ya entrenadas con aplicaciones similares o que incluyan la aplicación objetivo puede reducir el sobreajuste y el tiempo

de entrenamiento. No se ha incluido en este proyecto pues se buscaba técnicas que no alteren tanto la red neuronal.

3. Otra posibilidad es diseñar una técnica propia contra el sobreajuste, analizarla y compararla con las integradas en este proyecto.

Aplicaciones y redes extra.

Las redes neuronales convolucionales utilizadas en este proyecto tienen una dimensión pequeña o mediana, así pues, se puede ampliar con redes de mayor tamaño. Algunas aplicaciones son:

1. Clasificaciones de múltiples objetos como puede ser el conjunto CIFAR-10.
2. Reconocimiento de textos escritos a mano.
3. Reconocimiento de textos digitales, ampliación y corrección de los mismos.
4. Segmentación de imágenes.
5. Segmentación y clasificación de objetos en vídeos.
6. Reconocimiento de voz.

Para todas estas ampliaciones, no sólo sería necesario partir de este proyecto, sino un aumento de los medios a disposición.

BIBLIOGRAFÍA

ASALE, R.-, & RAE. (s. f.-a). *Entrenamiento | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. Recuperado 7 de agosto de 2023, de https://dle.rae.es/entrenamiento

ASALE, R.-, & RAE. (s. f.-b). *Entrenar | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. Recuperado 7 de agosto de 2023, de https://dle.rae.es/entrenar

ASALE, R.-, & RAE. (s. f.-c). *Inteligencia | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. Recuperado 1 de agosto de 2023, de https://dle.rae.es/inteligencia

Chollet, F. (2021). Introduction to deep learning for computer vision. En *Deep Learning with Python,* (Second Edition). Manning.

Deep Learning – Introducción práctica con Keras. (s. f.). *Jordi TORRES.AI*. Recuperado 1 de agosto de 2023, de https://torres.ai/deep-learning-inteligencia-artificial-keras/

Fernandez, R. (2018, julio 12). Reducir el sobreajuste con la regularización de la Deserción. *▷ Cursos de Programación de 0 a Experto © Garantizados*. https://unipython.com/reducir-el-sobreajuste-con-la-regularizacion-de-la-desercion/

https://keepcoding.io/blog/que-es-la-convolucion/. (2022, agosto 22). *¿Qué es la convolución? | KeepCoding Bootcamps*. https://keepcoding.io/blog/que-es-la-convolucion/

Jay. (2022, octubre 19). *Aumento de datos: Esencial para los modelos de aprendizaje automático*. HashDork. https://hashdork.com/es/aumento-de-datos/

¿Qué es deep learning? (s. f.). Recuperado 2 de agosto de 2023, de https://www.sas.com/es_es/insights/analytics/deep-learning.html

¿Qué es Deep Learning? | IBM. (s. f.). Recuperado 2 de agosto de 2023, de

https://www.ibm.com/es-es/topics/deep-learning

¿Qué es el sobreajuste? | IBM. (s. f.). Recuperado 8 de agosto de 2023, de https://www.ibm.com/es-es/topics/overfitting

Regularización. ¿Qué, por qué, cuándo y cómo? (s. f.). ICHI.PRO. Recuperado 13 de agosto de 2023, de https://ichi.pro/es/regularizacion-que-por-que-cuando-y-como-233797244596863