



Universidad  
Rey Juan Carlos

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA  
INFORMÁTICA**

**GRADO EN INGENIERÍA DE SOFTWARE**

**Curso Académico 2020/2021**

**Trabajo Fin de Grado**

**Reconocimiento de enfermedades de plantas con  
deep learning**

**Autor:** Eduardo Sierra Martín

**Director:** Miguel Ángel Rodríguez García

## **Resumen.**

La habilidad de reconocer los problemas que esconden las plantas supone en ocasiones un reto, ya que existe una gran variedad de plagas y enfermedades. Dentro de estos problemas se encuentra, por ejemplo, el color amarillo de las hojas. Este síntoma es muy habitual y puede significar desde que la planta sufre deshidratación, hasta que sufre de alguna enfermedad como la roya. La detección prematura de este tipo de enfermedades en vegetales es vital para mitigar sus efectos nocivos y su propagación, y para mejorar su ciclo de vida.

En este proyecto se utilizarán tecnologías de inteligencia artificial como el *deep learning*, que basa su funcionamiento en redes neuronales para crear un sistema de visión artificial. El modelo a desarrollar se basará fundamentalmente en convoluciones, una técnica que imita el funcionamiento de la vista humana y que permitirá la automatización de la detección de enfermedades y plagas.

El objetivo principal de este proyecto es por tanto el de crear una aplicación a través de iteraciones con intención de perfeccionarla, utilizando para este último fin diversas técnicas de ingeniería del conocimiento.

### **Palabras clave**

Inteligencia Artificial, Aprendizaje Profundo, Redes Convolucionales, Enfermedades de Plantas.

# Índice.

Resumen .....	i
Palabras clave .....	i
Índice de Figuras .....	iv
1. Introducción.....	1
2. Conceptos previos .....	2
2.1. La inteligencia artificial. ....	2
2.2. El perceptrón, la unidad de las redes neuronales.....	4
2.3. Las redes neuronales.....	6
2.4. Las redes convolucionales.....	7
3. Estado del arte. ....	10
4. Objetivos.....	12
5. Metodología.....	13
6. Descripción informática .....	16
6.1. Herramientas.....	16
6.2. Definición del problema .....	17
6.3. Obtención y procesamiento del conjunto de datos .....	17
6.4. Diseño y configuración de parámetros del modelo .....	20
6.5 Alcanzar línea base .....	22
6.5.1 El sobreajuste.....	24
7. Experimentos y resultados.....	26
7.1. El primer modelo: análisis y mejora.....	26
7.2. Aumento de imágenes .....	26
7.3. Desprendimiento.....	28
7.4. Arquitectura más profunda y conexiones residuales .....	30
7.4.1. Capas densas al final del modelo.....	31
7.4.2. Conexiones residuales .....	32
7.5. Experimentos con el modelo actual.....	33
7.5.1. Modelo 3.4.....	33
7.5.2. Modelo 3.5.....	34
7.5.3. Modelo 3.6.....	34
7.5.4. Modelo 3.7.....	35
7.5.4. Modelo 3.8.....	35
7.5.5. Modelo 3.9.....	36
7.5.6. Modelo 3.10.....	36
7.5.7. Modelo 3.11.....	37
7.5.8. Modelo 3.12.....	37

7.6. Normalización de lotes .....	38
7.7. Modelos pre-entrenados: la arquitectura Xception.....	40
7.7.1. Extracción de características .....	41
7.7.2. Modificación y afinamiento.....	42
7.8. Comparación de modelos y resultados. ....	43
8. Conclusiones.....	46
Bibliografía.....	47
Anexo 1. Descripción de la carpeta del TFG .....	49
Anexo 2. Manual de Usuario.....	1
2.1. Requisitos hardware .....	1
2.2. Requisitos software .....	1
2.3. El dataset .....	1
2.4. Guía de uso o ejecución de los modelos.....	2
2.4.1. Inicialización .....	2
2.4.2. Creación un modelo.....	3
2.4.3. Entrenamiento del modelo.....	4
2.4.4. Análisis de resultados .....	5

## Índice de Figuras.

Figura 1 Grados Celsius a Fahrenheit según la programación estándar.....	2
Figura 2 El nuevo paradigma de programación: el aprendizaje automático. ....	3
Figura 3 Representación de una red neuronal que clasifica .....	3
Figura 4 Clasificación de la IA.....	4
Figura 5 El perceptrón simple. ....	4
Figura 6 Función sigmoide.....	5
Figura 7. Red de perceptrones. ....	6
Figura 8. Zona de juegos de TensorFlow. ( <a href="http://playground.tensorflow.org">http://playground.tensorflow.org</a> ) .....	6
Figura 9. Funcionamiento de las convoluciones. ....	7
Figura 10. Ejemplos de la aplicación de un filtro.....	8
Figura 11. Aplicación de las convoluciones a un número escrito a mano.....	8
Figura 12. Funcionamiento de las capas de agrupamiento.....	9
Figura 13. Análisis de éxito de las metodologías ágiles y en cascada según un estudio realizado a más de 10.000 proyectos entre 2011 y 2015 por Standish Group.....	14
Figura 14. Metodología en cascada retroalimentada. ....	15
Figura 15. Esquema de la metodología que se va a utilizar. ....	15
Figura 16. Encuesta de las herramientas software para desarrollar AA más utilizadas en 2021 realizada por Kaggle ( <a href="https://www.kaggle.com/kaggle-survey-2021">https://www.kaggle.com/kaggle-survey-2021</a> ).....	16
Figura 17 Ejemplos de imágenes en el conjunto de datos seleccionado .....	18
Figura 18 Gestión de los datos a partir del conjunto de datos original .....	20
Figura 19. Fórmula de la precisión de clasificación. ....	21
Figura 20. Fórmula de la entropía cruzada. ....	21
Figura 21. La herramienta “validación cruzada de k iteraciones”.....	22
Figura 22. Arquitectura del modelo 0.....	23
Figura 23. Gráficas de pérdida y precisión del modelo 0 .....	23
Figura 24. Ejemplos de ajustes de un modelo .....	25
Figura 25. Ejemplo de aumento de datos. ....	26
Figura 26. Resultado del aumento de datos en el conjunto de datos actual. ....	27
Figura 27. Gráficas de pérdida y de precisión del modelo 0 y del modelo 1. ....	27
Figura 28. Funcionamiento del desprendimiento en modelos.....	28
Figura 29. Arquitectura del modelo 2.....	29
Figura 30. Gráficas de pérdida y de precisión del modelo 1 y del modelo 2. ....	29
Figura 31. Arquitectura del modelo 3.....	30
Figura 32. Gráficas de pérdida y de precisión del modelo 2 y del modelo 3. ....	31
Figura 33. Arquitectura del modelo 3.1.....	31
Figura 34. Gráficas de pérdida y de precisión del modelo 3 y del modelo 3.1. ....	32
Figura 35. Arquitectura del modelo 3.2.....	32
Figura 36. Rendimiento de modelos según el número de conexiones residuales.....	33
Figura 37. Esquema de modelos experimentales.....	33
Figura 38. Modelos 3.3 y 3.4.....	34
Figura 39. Modelos 3.4 y 3.5.....	34
Figura 40. Modelos 3.5 y 3.6.....	35
Figura 41. Modelos 3.6 y 3.7.....	35
Figura 42. Modelos 3.7 y 3.8.....	36

Figura 43. Modelos 3.8 y 3.9.....	36
Figura 44. Modelos 3.8 y 3.10.....	37
Figura 45. Modelos 3.5 y 3.11.....	37
Figura 46. Modelos 3.11 y 3.12.....	38
Figura 47. Gráficas de pérdida y de precisión del modelo 3.10 y del modelo 4 .....	38
Figura 48. Funcionamiento de la capa de aplanamiento y capa de agrupación de promedio global. ....	39
Figura 49. Gráficas de pérdida y de precisión del modelo 4.1 .....	39
Figura 50 Modelos disponibles de Keras ( <a href="https://keras.io/api/applications/">https://keras.io/api/applications/</a> ) .....	40
Figura 51. Extracción de características del modelo Xception. ....	41
Figura 52. Gráficas de pérdida y de precisión de un Xception con extracción de características.....	41
Figura 53. Modificación y afinamiento del Xception.....	42
Figura 54. Gráficas comparativas de precisión y pérdida de los diferentes modelos....	43
Figura 55. Gráficas comparativas del tamaño de los modelos y su relación con la precisión. ....	44
Figura 56. Tiempo de entrenamiento y su relación con la precisión ganada de los distintos modelos.....	45

## 1. Introducción

La inteligencia artificial (IA, en inglés artificial intelligence) es un tema muy concurrido hoy en día. Aplicaciones que usan millones de personas en todo el mundo, como lo son TikTok, Facebook o YouTube basan muchas de sus principales funcionalidades en la IA (Orús Abigail, 2022). Generalmente, en este tipo de aplicaciones se utiliza principalmente con fines estadísticos, así como anuncios personalizados (Oleaga Jon, 2019). Sin embargo, la IA tiene muchas más aplicaciones, algunas de ellas están relacionadas con la imitación de las capacidades humanas, como lo son la vista y el oído. En este ámbito se encuentran, por ejemplo, los famosos asistentes virtuales, Siri, Alexa o el asistente de Google. Los sistemas descritos emplean estas tecnologías para reconocer el habla, lo que les hace capaces de ejecutar acciones como mandar mensajes de texto por voz o encender una bombilla.

A la rama de la informática encargada de aplicar IA para hacer que un computador entienda el contenido de una imagen digital, se le llama visión artificial o visión computarizada (Himanshu Lawaniya, 2020). Para que un sistema de visión artificial funcione hace falta un componente que capte las imágenes y otro que implemente la lógica para que el computador entienda la imagen. El proyecto que se va a desarrollar está enfocado en este último, en el software.

Este proyecto tiene como objetivos principales realizar un estudio en profundidad de las soluciones existentes en la actualidad, estudiar y crear un sistema de visión artificial desde cero y resolver los problemas que se presentan a continuación. El primer problema se centra en diagnosticar una gran cantidad de plantas es una tarea repetitiva que tiende a sufrir del error humano. Otro de los problemas es que, para realizar un correcto diagnóstico, es necesario cierto nivel de especialización (Riley et al., 2002).

Para la consecución de los objetivos y la resolución de los problemas, se utilizarán técnicas de ingeniería del conocimiento y sistemas de redes neuronales. Además, para comprender las modificaciones que se hagan a la red se realizará un estudio en profundidad del impacto y eficacia de las modificaciones, se analizará la arquitectura de la red y se realizarán diversos experimentos en los que se compararán los resultados. Para que el sistema sea capaz de reconocer las enfermedades, se buscará una colección de imágenes bien categorizadas de distintas especies de plantas con distintos síntomas.

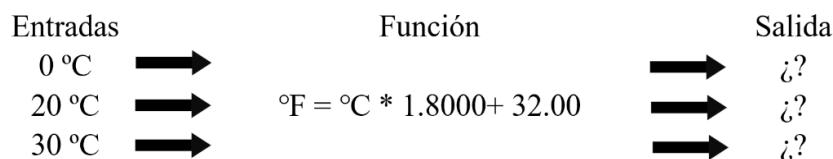
## 2. Conceptos previos

## 2.1. La inteligencia artificial.

En esta sección se va a dar una primera aproximación sobre el concepto de IA. Existen varias definiciones dadas a lo largo de la historia que proporcionan al concepto de IA un enfoque distinto según el punto de vista.

Se tiene pues la IA fuerte, definida como sistemas que piensan como las personas, como señaló John Haugeland en 1985 a lo que señaló como IA simbólica, “*La Inteligencia Artificial solo desea el artículo verdadero: máquinas que tengan mente, en su sentido pleno y literal*” (Haugeland, 2001). Este tipo de IA estuvo en auge desde 1950, cuando Alan Turing crea una prueba apodada con su apellido que examinaba la capacidad que tenía una máquina de mostrar comportamiento inteligente<sup>1</sup>.

Tras el boom de los sistemas expertos<sup>2</sup> en los años 80, la IA fuerte perdió la popularidad debido a que los primeros daban mejores resultados al ser más sencillos y eficaces. Los sistemas expertos abarcan el campo de la IA débil, definida como “*La rama de la Informática que se ocupa de la automatización del comportamiento inteligente*” (F. Luger & A. Stubblefield, 1998). Este enfoque viene dado después de abandonar la idea de poder crear una IA que sea capaz de pensar imitando el ser humano. Hasta la década de los 90 se había seguido el paradigma de la programación estándar donde se tenía un claro objetivo: transformar datos de entrada en datos de salida. Un ejemplo sería el de transformar grados Celsius a grados Fahrenheit (ver Figura 1).

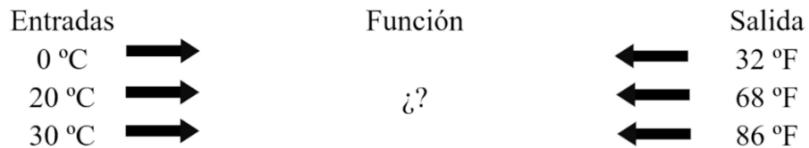


*Figura 1 Grados Celsius a Fahrenheit según la programación estándar.*

Fue entonces cuando se empezó a crear programas que invirtieran el proceso y que fuesen capaces de “aprender”. Se buscaban programas que, en lugar de programas que produjeran una salida como en la Figura 1, fueran capaces de generar una función. De esta forma, serían capaces de generalizar cualquier problema sin tener que escribir un programa para cada uno. En este contexto, aparece un nuevo concepto: el aprendizaje automático (AA, en inglés machine learning).

<sup>1</sup> <https://www.becas-santander.com/es/blog/test-de-turing.html>

<sup>2</sup> <https://www.futurespace.es/machine-learning-los-origenes-y-la-evolucion/>

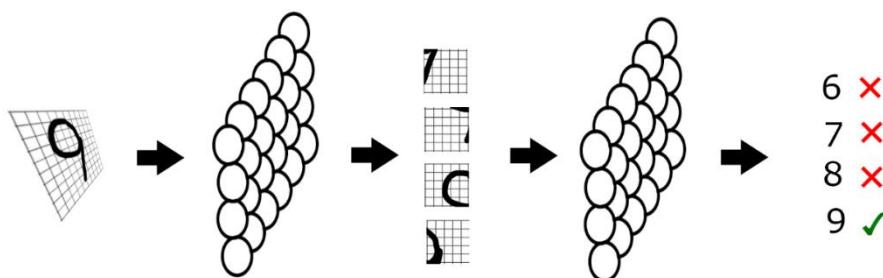


*Figura 2 El nuevo paradigma de programación: el aprendizaje automático.*

En la Figura 2 se puede ver el modelo de entrenamiento con el ejemplo anterior. Este modelo se compone de dos fases: i) fase de entrenamiento, en la que se reciben datos y salidas para crear la función; y ii) fase de uso, en el que el programa ya entrenado, recibe datos de entrada y es capaz de predecir la salida, funcionando de manera similar a la programación estándar.

El AA tiene sentido si se intenta imitar la capacidad de razonamiento de los seres humanos que aprenden del mundo que le rodea a través de ejemplos. Nadie sabría identificar, por ejemplo, un elefante sin haber recibido una descripción o imagen suya. Para imitar este comportamiento no basta con el AA, es necesario el aprendizaje profundo<sup>3</sup> (AP, en inglés deep learning).

El AP es una de las ramas del AA. Esta rama se caracteriza porque, para el problema anterior, no utilizaría un algoritmo de regresión, utilizaría redes neuronales. Las redes neuronales del AP funcionan de forma similar a una cadena de procesamiento. Por ejemplo, una roca con hierro se transforma en lingotes, parte de los lingotes son usados para hacer tornillos y los tornillos se combinan con otras piezas para hacer el producto final, un reloj. Se puede ver que las transformaciones se producen de forma consecutiva, en capas: el producto no pasa a la fase siguiente sin haber sido transformado.



*Figura 3 Representación de una red neuronal que clasifica*

En la Figura 3 se representan las fases de refinamiento para clasificar un número escrito a mano, siguiendo el esquema de capas del ejemplo anterior. En primer lugar, cada uno de los círculos representa la unidad de cada capa. Las unidades son las encargadas de realizar la transformación en cada una de las fases. La primera capa procesa la entrada y la envía a la siguiente y, a su vez, esta procesa los nuevos datos y envía nuevos datos a la siguiente. De forma iterativa, este proceso se realiza capa a capa hasta llegar a una

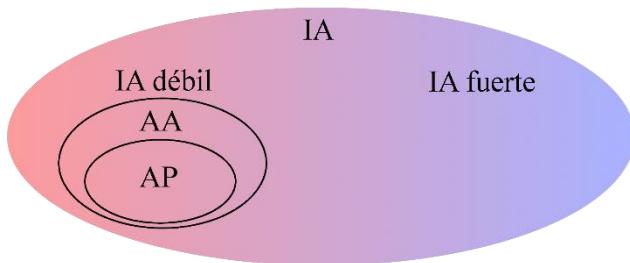
---

<sup>3</sup> <https://iat.es/tecnologias/inteligencia-artificial/deep-learning/>

neurona de salida que finalmente clasifica el resultado, devolviendo que lo que se ha escrito a mano es un 9 y no otro número.

Esta explicación, como introducción, es poco precisa pero útil de cara a dar una primera aproximación de lo que se va a explicar en profundidad a continuación. Realmente el término neurona se refiere a las neuronas físicas que componen un cerebro, en el ámbito del AA su nombre técnico es el de perceptrón (Ligdi González, 2021). Como última aclaración, véase que en el ejemplo anterior se han utilizado dos capas. En realidad, se puede usar un número mucho mayor de capas y perceptrones en cada una de ellas, incluso capas con diferentes números de perceptrones para una misma red.

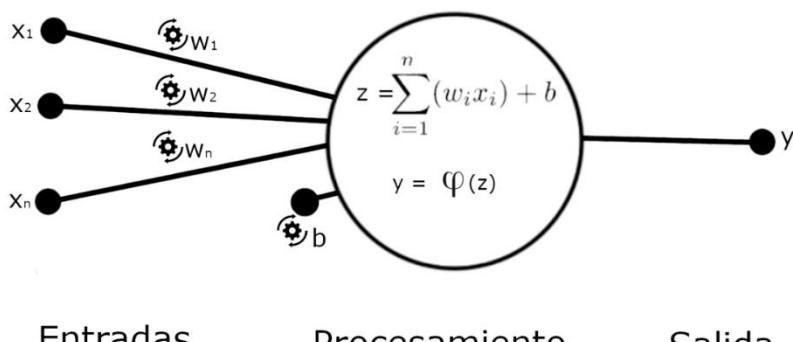
En la Figura 4 se muestra la clasificación de los conceptos vistos hasta ahora<sup>4</sup>. Todos los temas que se abordarán desde este momento estarán en la categoría de AP.



*Figura 4 Clasificación de la IA.*

## 2.2. El perceptrón, la unidad de las redes neuronales

El perceptrón ha sido diseñado para imitar el funcionamiento de una neurona biológica. Para la explicación del funcionamiento de las redes neuronales, se comenzará por la unidad, el perceptrón simple<sup>5</sup>, cuya arquitectura se puede ver en la Figura 5.



*Figura 5 El perceptrón simple.*

<sup>4</sup> <https://blog.bismart.com/diferencia-machine-learning-deep-learning>

<sup>5</sup> <http://avellano.fis.usal.es/~lalonso/RNA/introMLP.htm>

El perceptrón simple está compuesto de una o varias entradas, denominadas como  $x$ , que generalmente son valores numéricos o cadenas de valores numéricos. Por ejemplo, en una imagen en blanco y negro un píxel está representado por un número del 0 a 255, siendo este valor una posible entrada. Estas entradas están acompañadas por un peso denominado como  $w$  (en inglés weight) que, como indica el símbolo que la acompaña, será un valor ajustable en la fase de aprendizaje del perceptrón. Otro valor ajustable será el sesgo  $b$  (en inglés bias) que será utilizado en el núcleo del perceptrón. Se dice que estos dos valores son ajustables porque son los que modifican su valor cuando proporcionan una salida que no coincide con el valor esperado. En otras palabras, son los encargados de generar el conocimiento del perceptrón.

En el núcleo del perceptrón se producen dos fases fundamentalmente matemáticas. Por un lado, se hace el sumatorio del resultado de multiplicar cada entrada por su respectivo peso y sumarle el sesgo, lo que proporciona el valor de la variable  $z$ . Por otro lado, se aplica una función de activación  $\varphi$ . Existen diversas funciones de activación<sup>6</sup>, como la función sigmoide mostrada en la Figura 6.

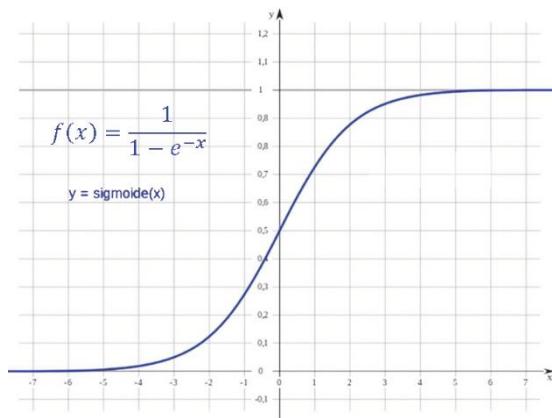


Figura 6 Función sigmoide.

El perceptrón tiene que pasar un proceso de ajuste para funcionar correctamente. En este proceso entran las variables indicadas con el símbolo de engranaje en la imagen del perceptrón: los pesos y el sesgo. Estas variables serán ajustadas en la primera fase, llamada fase de aprendizaje o de entrenamiento, donde los valores de los pesos y el sesgo partirán de valores aleatorios que se modificarán de forma iterativa para tener como salida el valor deseado.

La utilidad principal del perceptrón simple es la de hacer tareas sencillas, como una puerta lógica. Supongamos que se tiene como entrada el valor 0 y 1 y se requiere que el perceptrón tenga la salida de una función *or*<sup>7</sup>. Para que proporcione la salida correcta, se tiene que entrenar de forma que si la salida no es la esperada, se modifiquen los valores de los pesos  $w$  y el sesgo  $b$ . Cuantos más ejemplos sean utilizados para entrenar un perceptrón, más fiable serán sus resultados.

<sup>6</sup> <https://medium.com/ai%C2%B3-theory-practice-business/a-beginners-guide-to-numpy-with-sigmoid-relu-and-softmax-activation-functions-25b840a9a272>

<sup>7</sup> <https://descubrearduino.com/puertas-logicas/>

## 2.3. Las redes neuronales

Cuando el problema a resolver se vuelve más complejo un sólo perceptrón no es de gran ayuda. En su lugar se usan estructuras más complejas compuestas por diversas capas que la constituyen varias decenas de perceptrones conectados, formando una red con una forma similar a la de la Figura 7. La red está compuesta por capas que, a su vez, están compuestas por perceptrones. Cada una de las capas puede conectarse con otras capas. En el caso de la detección de imágenes cada uno de los perceptrones de la capa de entrada recibiría un píxel y la capa de salida sería la encargada de, por ejemplo, dar un resultado de clasificación. Las capas que no son las de entrada o las de salida son las llamadas capas ocultas. La salida de la última capa oculta será la entrada de la última capa, la capa de salida, compuesta de uno a varios perceptrones dependiendo del problema a resolver.

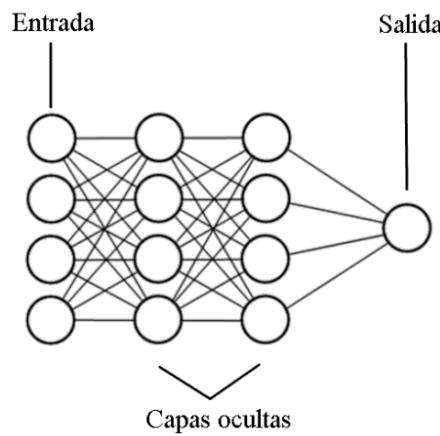


Figura 7. Red de perceptrones.

Para aclarar estos conceptos, TensorFlow tiene una página web dedicada al análisis del impacto de las modificaciones de todos los parámetros. Además, esta página permite analizar el comportamiento de una misma configuración para la resolución de distintos problemas de forma visual e interactiva (ver Figura 8).

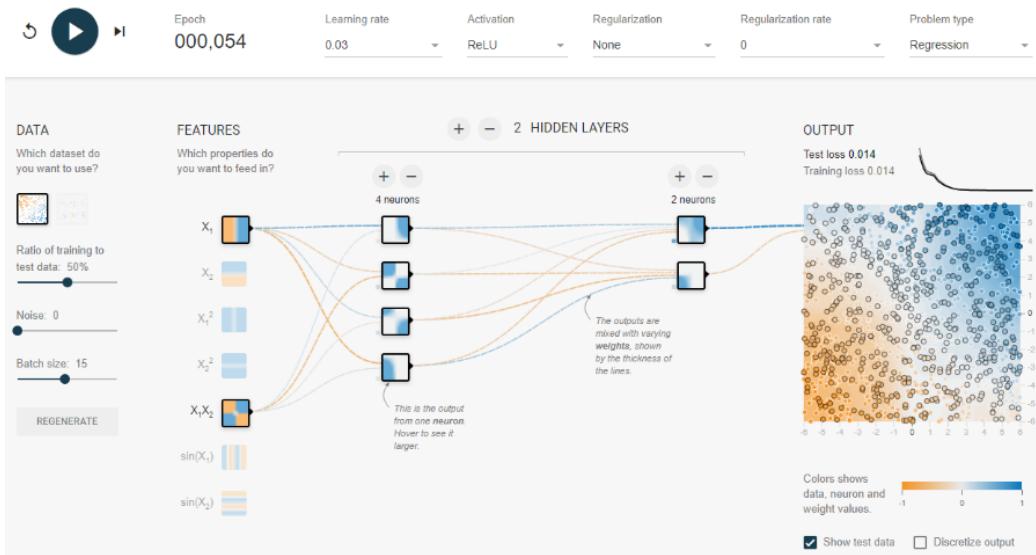


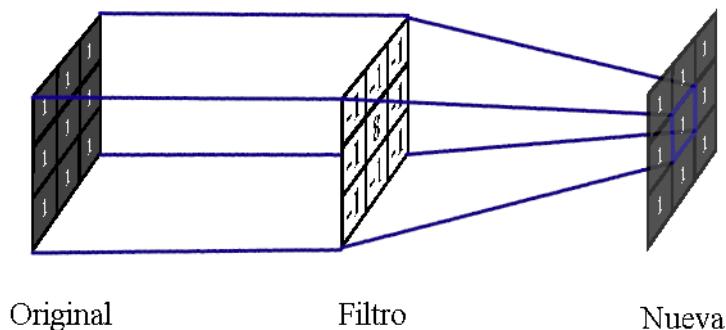
Figura 8. Zona de juegos de TensorFlow. (<http://playground.tensorflow.org>)

Con una red de perceptrones se puede hacer un buen clasificador de imágenes que detecta números escritos a mano. Sin embargo, este tipo de redes memorizan el patrón global de la imagen sin considerar la imagen que tenga dibujada en su interior. De esta forma, si se invierten los colores de la imagen, el modelo entrenado sería incapaz de predecir el resultado correctamente. Para resolver este problema, existe un nuevo tipo de red: las redes convolucionales.

#### 2.4. Las redes convolucionales

Las redes convolucionales son redes de perceptrones especializadas en el tratamiento de imágenes. La principal diferencia con las redes densas es que cuando se transfiere la imagen de capa a capa es procesada a través de convoluciones, gracias a un núcleo convolucional. Un núcleo convolucional es una matriz de tamaño variable, aunque habitualmente es de tamaño  $3 \times 3$  o  $5 \times 5$ . Este núcleo ha sido diseñado para hacer una serie de operaciones a los píxeles de la imagen que recibe. Lo que hace el núcleo convolucional es calcular un único valor numérico a partir de una matriz. La matriz contiene los valores de color de los píxeles, de forma que al final se sustituye el valor central de la matriz con el nuevo valor calculado. El resultado de aplicar el núcleo convolucional a toda la imagen es una nueva imagen a la que se ha aplicado un filtro. En consecuencia, es común decir que cada tipo de núcleo convolucional es un filtro.

En la Figura 9 se puede ver el funcionamiento de un filtro particular que proporciona una nueva imagen únicamente mostrando sus bordes. Este filtro mostrará en blanco los bordes de la imagen y en negro todo lo demás. Para ello multiplica por -1 todos los píxeles circundantes y el resultado se lo suma al producto del píxel central y 8. En este caso, al ser todos los valores unos no se detecta ninguna variación en el patrón y, por tanto, no se realiza ninguna modificación en la imagen.



*Figura 9. Funcionamiento de las convoluciones.*

De forma análoga, se muestran en la Figura 10 más ejemplos en los que sí se producen cambios. Los colores blancos serán cambiados a negro y los negros los dejará como están, mientras que, cuando existan bordes, es decir, detecte blancos y negros a la vez, producirá un color blanco.

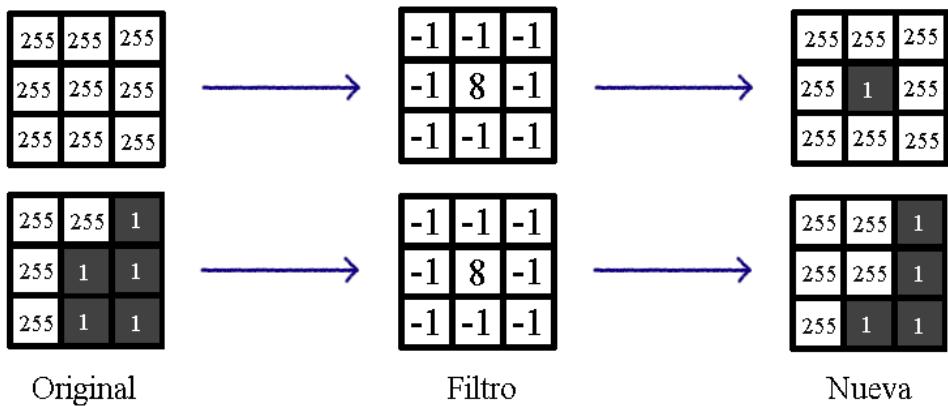


Figura 10. Ejemplos de la aplicación de un filtro.

El resultado de aplicar este filtro a toda la imagen píxel por píxel se puede ver en la Figura 11.



Figura 11. Aplicación de las convoluciones a un número escrito a mano.

Cabe mencionar que las redes convolucionales aplican numerosos filtros a las imágenes. La práctica habitual es la de aplicar unos pocos filtros a la imagen original y luego reducir el tamaño de la imagen a la mitad para luego aplicar, por ejemplo, el doble de filtros. La intención es, por un lado, reducir la complejidad del problema, ya que al reducir el tamaño de la imagen, la aplicación de filtro es más fácil; y, por otro lado, la red será capaz de aprender patrones más abstractos a medida que se apliquen más filtros a imágenes más pequeñas. Esta organización de filtros introduce un nuevo elemento necesario para las redes convolucionales: las capas de agrupamiento que se encargan de escalar la imagen a la mitad de su resolución. Su funcionamiento queda representado en la Figura 12.

1	3	3	1
2	6	8	1
2	3	7	4
2	5	1	2

Imagen original

6	8
5	7

Imagen después de agrupamiento

*Figura 12. Funcionamiento de las capas de agrupamiento.*

Por último, hay que señalar las funciones de activación que serán utilizadas. Las capas intermedias de las redes que se desarrollarán en el proyecto, tanto capas densas como las convolucionales, tendrán una función de activación rectificadora *ReLU*. La salida de los perceptrones que utilicen la función *ReLU* será la misma que la de la función de agregación o 0, si esta proporciona un resultado negativo.

La última capa es la encargada de generar el resultado de clasificación. Esta capa tendrá una función de activación *SoftMax* de un tamaño igual al de posibles resultados de clasificación. Esta configuración se debe a que *SoftMax* proporciona un valor probabilístico de cada una de las opciones de clasificación. Por ejemplo, si se recibe una imagen que contiene un oso y se tiene que clasificar entre perro, oso o gato, esta capa proporcionaría un vector con tres valores numéricos comprendidos entre 0 y 1. Si la red está bien configurada, el valor de oso será el mayor de los tres. Un ejemplo válido de salida para este problema sería entonces: 0.1, 0.9, 0 para la probabilidad de que sea un perro, un oso o un gato, respectivamente. Nótese que los valores del vector tienen que sumar 1.

### 3. Estado del arte.

Los logros de la IA hoy en día es algo que hasta hace algo más de 20 años era impensable (Ignacio Bagnato, 2018). Desde la invención del perceptrón en 1958 por Frank Rosenblatt (B. Loiseau, 2019), pasando por la creación de las redes convolucionales (Y. LeCun et al., 1989), la victoria de una IA llamada Deep Blue sobre el campeón de ajedrez de la época Garry Kasparov en 1997, el AA no ha parado de crecer (Pablo Espeso, 2014). Sin embargo, durante este tiempo se avanzó de forma teórica y no tanto práctica, ya que la velocidad de cómputo, entre otras cosas, era un gran cuello de botella que impedía su desarrollo.

Los tres principales factores que han permitido el desarrollo del AA como lo conocemos hoy en día son: la potencia de cómputo, la disponibilidad de datos y el desarrollo de algoritmos y software. En primer lugar, la potencia de cómputo ha mejorado gracias a la inversión de compañías como Nvidia en crear tarjetas gráficas más potentes con intención de mejorar la calidad gráfica de los videojuegos. Dicha potencia ahora puede ser aprovechada para el entrenamiento de las redes neuronales. En segundo lugar, con la invención y desarrollo de Internet, hoy en día hay disponibles conjuntos de datos de varias decenas de miles de ejemplos para entrenar las redes creadas. Un ejemplo de esto sería la página Kaggle que permite la descarga de casi cualquier tipo de conjunto de datos con miles de imágenes y descripciones. Finalmente, el tercer gran impulsor es el avance de algoritmos, que han mejorado sobre todo entre 2009 y 2016, y el software por ser cada vez más sencillo. La mayor facilidad de desarrollo se debe a que en los principios se utilizaba el lenguaje C++ por su velocidad de ejecución y CUDA por su capacidad de repartir el trabajo de cómputo entre las unidades de procesamiento, mientras que ahora es suficiente con conocimientos básicos de Python.

Un trabajo interesante en el ámbito es el de Guan (Guan et al., 2021), donde se utilizan distintas arquitecturas combinadas entre sí para generar un modelo que tiene mejores resultados. Concretamente, el usuario proporciona los datos en forma de imagen y texto, respectivas a enfermedades de árboles frutales. Después, estos son procesados por dos redes diferentes: CNN-DNN para obtener la información de las imágenes con una red convolucional y, en paralelo, se usa una red BiLSTM para procesar el texto. Los resultados obtenidos de esta nueva red, nombrada CNN-DNN-BiLSTM, concluyen que tiene una mayor precisión prediciendo que por ejemplo la famosa arquitectura VGG o la propia BiLSTM por sí sola.

El artículo de Saleem estudia distintas arquitecturas de redes convolucionales para clasificar enfermedades de plantas (Saleem et al., 2020). En concreto, utiliza un conjunto de unas 54.000 imágenes clasificadas en 38 grupos de plantas sanas y con enfermedades, correspondientes a 14 especies. En la comparativa de arquitecturas se incluyen algunas de las más relevantes de la actualidad, como la VGG-16 o la Xception, así como arquitecturas modificadas, como la GoogLeNet. El estudio concluye con que la Xception tiene uno de los mejores resultados para el propósito específico de clasificar plantas. Las arquitecturas serán comentadas en la sección correspondiente (ver apartado 7.7. Modelos pre-entrenados: la arquitectura Xception.).

En la misma línea, el estudio realizado por Toda y Okura utiliza el dataset anterior para detallar el funcionamiento interno de una red pre-entrenada, capaz de identificar distintos tipos de elementos en una imagen (Toda & Okura, 2019). Los autores realizan distintas

pruebas sobre una red inspirada en la Inception para analizar cómo procesa la red una única imagen. El objetivo de estas pruebas se centra en estudiar la arquitectura de este modelo, analizando la relación entre su profundidad y el nivel de abstracción de la imagen. El resultado de este estudio da lugar a los mapas de atención, que son imágenes similares a la original, sin filtros, pero donde se aprecia con claridad dónde la red presta más atención dependiendo de los colores.

## 4. Objetivos

El objetivo principal de este proyecto se enfoca en diseñar e implementar un sistema que sea capaz de clasificar imágenes de plantas por enfermedades o plagas. Para afrontar el proyecto, este objetivo principal se ha desglosado en los siguientes objetivos que se exponen a continuación.

Obj1: Analizar las aproximaciones existentes en el dominio del reconocimiento de imágenes con IA. Este análisis pretende realizar un estudio del arte que representa los cimientos del proyecto, donde se analizará el estado de las áreas de conocimiento relacionadas para poder establecer la base tecnológica.

Obj2: Analizar un conjunto de datos adecuado para la resolución del problema, ampliándolo si fuera necesario para optimizar los resultados. Este punto es uno de los más importantes, ya que el conjunto de datos es la piedra angular del proyecto y, por ello, es vital seleccionar uno óptimo en tamaño y calidad.

Obj3: Diseñar un sistema basado en AA que permita clasificar imágenes. Para llevarlo a cabo se analizarán los resultados y se optimizará el sistema a lo largo del proyecto.

Obj4: Implementar y validar el funcionamiento del sistema en comparación con las arquitecturas de sistemas existentes. Existen compañías que han desarrollado buenas soluciones gracias a que disponen de mucha potencia de cómputo proveniente de granjas de servidores. Este objetivo plantea la utilización de estas soluciones a priori mejores, por ser sistemas más complejos, y las compara con el sistema creado desde cero.

## 5. Metodología.

Para el desarrollo de proyectos software existen principalmente dos grandes grupos de metodologías: las tradicionales y las ágiles (Espinoza-Meza, 2013). Las metodologías tradicionales se caracterizan por la elaboración exhaustiva de requisitos y por su utilización para procesos de desarrollo largos y bien definidos. Son propias de otras ramas de la ingeniería y fueron aplicadas inicialmente en el desarrollo del software cuando empezaba a ser más complejo, gracias a la aparición de los lenguajes de alto nivel. Un ejemplo es la metodología en cascada, una de las metodologías más populares, que se caracteriza por desarrollar el proyecto en fases secuenciales sin posibilidad de volver a una fase anterior. Su popularidad se debe a la estandarización de su uso en 1985 por el departamento de defensa de Estados Unidos, debido a una mala interpretación de un documento de 1970 escrito por Winston Royce<sup>9</sup> (Javier Garzas, 2014). Sin embargo, lo que Royce indicaba acerca de la metodología en cascada era justo una crítica y no una recomendación. De hecho, lo que proponía en realidad era una variación del modelo en cascada, el modelo en cascada retroalimentada.

Dos años después de la estandarización del modelo en cascada, se dieron cuenta de los altos costes y del poco éxito que tenía esta metodología. Esta situación llevó al propio departamento de defensa a advertir las consecuencias de su uso. A pesar de esto, en 1994 los proyectos software seguían utilizando el modelo en cascada casi en su totalidad (Alaimo, 2018). Un estudio realizado por el Standish Group<sup>10</sup> en ese año publicó el rendimiento de los proyectos software: la mitad de los proyectos se habían excedido en coste o tiempo y sólo el 16% tuvo éxito, con un 31% de índice de fracaso.

Como advirtió Royce, el principal desafío del desarrollo del software son los requisitos cambiantes y por ello abogaba por los modelos incrementales, la base de las metodologías ágiles. Este segundo grupo de metodologías coexistía con los modelos tradicionales a partir de 1990, pero fue en 2001 cuando un grupo de expertos en el tema firmaron el “Manifiesto para el desarrollo ágil del software”<sup>11</sup>. En este manifiesto recoge las características principales de las metodologías ágiles que son: i) ”Individuos e interacciones sobre procesos y herramientas”, ii) “Software funcionando sobre documentación extensiva”, iii) “Colaboración con el cliente sobre negociación contractual” iv) “Respuesta ante el cambio sobre seguir un plan”. “Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda”.

Después de la firma del manifiesto, la popularidad de uso de estas metodologías se incrementó enormemente. Este crecimiento se debe a la gran diferencia de éxito que tienen las metodologías ágiles frente al modelo en cascada. Según un estudio del Standish Group realizado entre 2011 y 2015, los proyectos software desarrollados utilizando metodologías ágiles tienen un 39% de tasa de éxito, alrededor de 4 veces más que los desarrollados con el modelo en cascada que tienen un 11% ( ver Figura 13),

---

<sup>9</sup> <http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf>

<sup>10</sup> [https://www.standishgroup.com/sample\\_research\\_files/chaos\\_report\\_1994.pdf](https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf)

<sup>11</sup> <https://agilemanifesto.org/iso/es/manifesto.html>

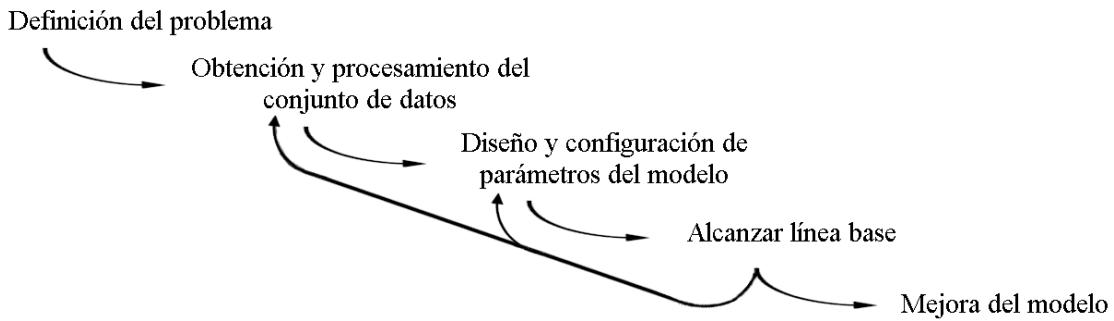


*Figura 13. Análisis de éxito de las metodologías ágiles y en cascada según un estudio realizado a más de 10.000 proyectos entre 2011 y 2015 por Standish Group.*

En este punto queda claro que las metodologías ágiles son las más indicadas para el desarrollo del software, al menos si se tienen en cuenta únicamente los índices de éxito de cada metodología. Sin embargo, hoy en día no existe una norma de usar una metodología sobre otra, de hecho depende mucho de cada proyecto, por lo que se tiene que realizar un estudio que concluya la aplicabilidad de una metodología u otra. En primer lugar, se deben tener en cuenta los objetivos del proyecto. En este caso no es el de desarrollar un software complejo para un cliente, es el de crear un sistema base y, posteriormente, realizar sobre él modificaciones con un exhaustivo análisis de los resultados, con intención de obtener un sistema óptimo. Se tienen por tanto dos fases de desarrollo: creación de un sistema de AA y su mejora iterativa.

Actualmente, para la creación de sistemas de AA se dispone de un guion que seguir, es el caso del llamado “flujo de trabajo universal del AA” (Chollet, 2017). Se trata de una serie de pasos que definen el correcto desarrollo de una IA, cuyos pasos se pueden ver a continuación: i) Definir el tipo de problema a resolver; ii) Obtener y procesar un conjunto de datos; iii) Configurar la red neuronal (a la que se llamará modelo); iv) Alcanzar una línea base; v) Mejora del modelo. Seguir estos pasos uno tras otro sin volver a un paso anterior implica la selección directa de una metodología en cascada sin sufrir una de sus principales deficiencias, invertir una gran cantidad de tiempo en elaborar un plan que seguir, ya que dicho plan ya está creado. No obstante, las fases de “alcanzar una línea base” y “mejora del modelo” son ambiguas e implican algunos problemas, como tener que retroceder a una fase anterior en el caso de no alcanzar una línea base.

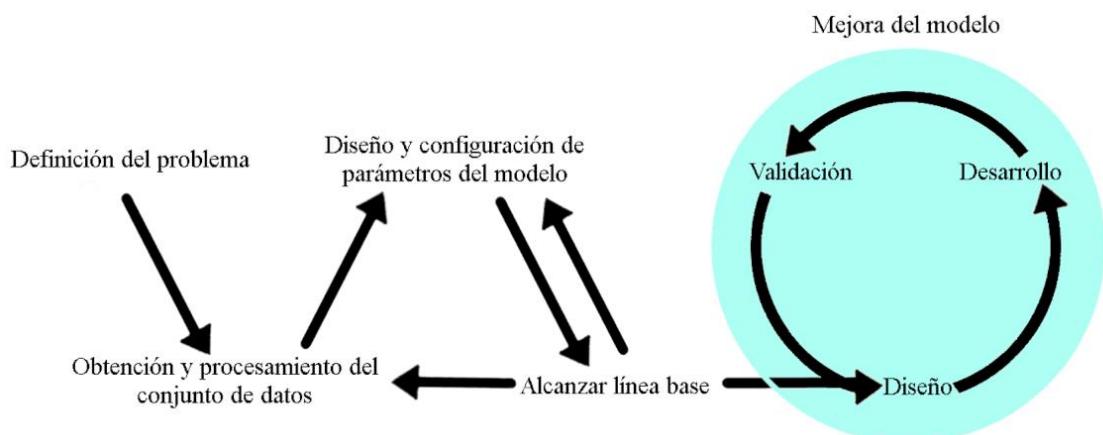
Para la solución al problema de la vuelta a una fase anterior se puede usar la solución propuesta por Royce en 1970: una cascada retroalimentada. En este caso lo único que interesaría es que, sólo en el caso de que no se haya logrado alcanzar una línea base, se pueda volver a la fase directamente anterior o a la siguiente anterior, que son las fases críticas de desarrollo, quedando la cascada como se muestra en la Figura 14.



*Figura 14. Metodología en cascada retroalimentada.*

Una vez se tiene un modelo básico sólo queda el último paso: su mejora. En este paso sí que tendría sentido aplicar una metodología ágil con los ciclos de mejora continua que supone su implantación. Las metodologías ágiles tienen como núcleo, obviando la parte de las personas, una serie de pasos que se hacen en bucle hasta acabar el sistema: diseño, desarrollo y validación, es decir, las iteraciones incrementales.

Si se asocia el esquema a las dos principales fases del proyecto se tiene que: la fase de creación del sistema de AA está cubierta por una metodología en cascada retroalimentada cuyo último paso es el de alcanzar la línea base, es decir, tener un modelo suficientemente funcional; y la fase de mejora iterativa, usando para ello el núcleo de las metodologías ágiles. El esquema resultante mostrado en la Figura 15 encaja a la perfección con las características del proyecto a desarrollar. Cada una de las dos fases corresponde con un apartado del documento: la primera con 6. Descripción informática y la segunda con 7. Experimentos y resultados.



*Figura 15. Esquema de la metodología que se va a utilizar.*

## 6. Descripción informática

En esta sección se utilizará la primera de las dos fases de las que se compone la metodología descrita, donde se sigue un modelo de cascada retroalimentada. Además, se realizará un breve estudio de las herramientas disponibles y de las que se utilizarán.

### 6.1. Herramientas

La forma más efectiva de desarrollar AA hoy en día es mediante el uso de librerías de Python<sup>12</sup>. En la Figura 16 se pueden ver las herramientas software para desarrollar AA más utilizadas según una encuesta de 2021 de Kaggle.

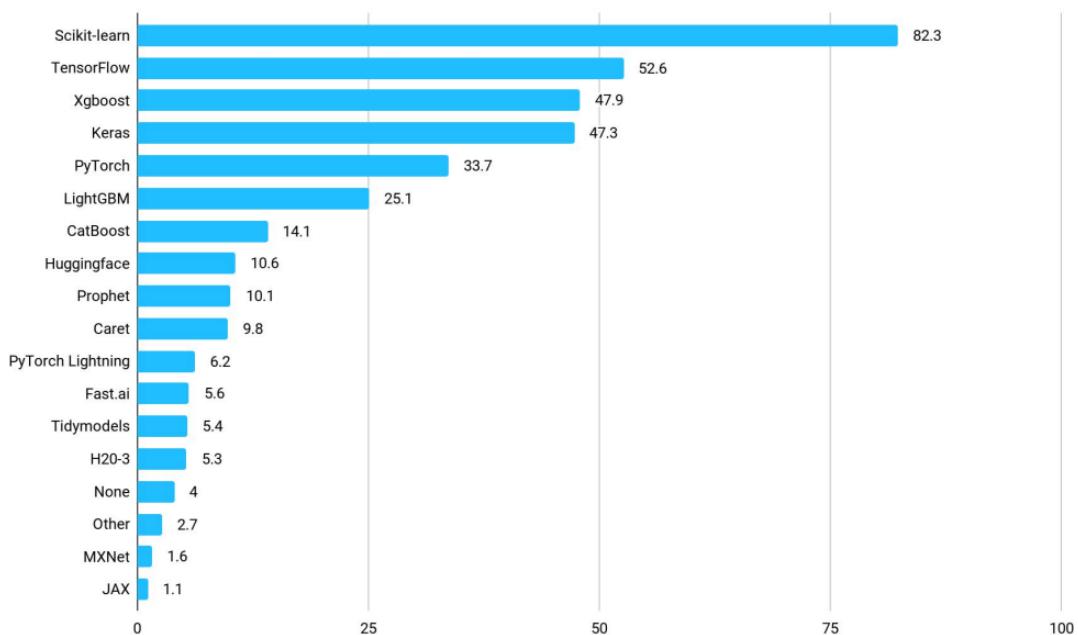


Figura 16. Encuesta de las herramientas software para desarrollar AA más utilizadas en 2021 realizada por Kaggle (<https://www.kaggle.com/kaggle-survey-2021>).

En las primeras posiciones se reparten herramientas para dos tipos de AA: los árboles impulsados por gradiente (en inglés gradient boosted trees) y el AP. Los árboles impulsados por gradiente se utilizan para problemas en los que las características más significativas de los datos se conocen con antelación y a partir de ellas se construyen árboles de decisión (Yıldırım, 2020). Para desarrollarlos se destaca la utilización de las librerías Scikit-learn, Xgboost y LightGBM. Cuando las características de los datos no son conocidas, como es el caso de las imágenes, se utiliza el AP, usándose en este caso TensorFlow, Keras y PyTorch, principalmente.

Por popularidad, apreciable en la Figura 16, se utilizará en el desarrollo del proyecto TensorFlow con Keras, nótese que su uso no es excluyente<sup>13</sup>. La razón del uso en conjunto de estas dos tecnologías es que Keras es el más sencillo de las tres tecnologías y será el utilizado en la totalidad del desarrollo del proyecto, pero a su vez pertenece a TensorFlow.

<sup>12</sup> <https://blog.paperspace.com/15-deep-learning-frameworks/>

<sup>13</sup> <https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article>

Por otro lado, se utilizará Jupyter Notebook para la implementación de código. Así, se podrán compilar los bloques de código que únicamente se requiera en cada momento, sin tener que hacer demasiadas configuraciones. Jupyter permite dejar todo bien documentado y estructurado. Además, ofrece persistencia en el código ejecutado por cada célula o caja de código. Esta característica es muy relevante porque en el entrenamiento de los modelos se puede invertir una gran cantidad de tiempo, varios minutos o incluso horas. Por otro lado, para gestionar las librerías y evitar problemas de conflictos se ha utilizado Conda, un gestor de paquetes de base científica. Por último, para la gestión de versiones del proyecto global, así como su mantenimiento en la nube, se ha utilizado GitHub.

## 6.2. Definición del problema

La tarea consistirá en, a partir de unos datos introducidos, predecir los resultados. Los datos serán imágenes de diferentes plantas que podrán contener enfermedades o plagas. Cada imagen se encuentra catalogadas con una etiqueta y se utilizarán para entrenar un modelo. La intención es que el modelo al recibir una imagen de una planta que nunca haya analizado sea capaz de identificar la especie y la enfermedad o plaga si la tiene. Como aclaración, la salida del sistema no tiene como objetivo proporcionar una especie, sino una clasificación del tipo especie-característica, donde característica puede ser la enfermedad que sufre o *sana* en caso de estarlo. En caso de que no la especie no esté asociada a una característica en la salida, se considerará una especie sana.

Dependiendo de cómo se quiera abarcar el problema, se puede considerar un problema de clasificación múltiple de etiqueta única o un problema de clasificación múltiple de etiqueta múltiple. Este último caso será el de considerar que el modelo reciba una imagen de una planta y lo que proporcionase, sea una lista de características, mientras que el de etiqueta única se enfoca en clasificar la imagen en una categoría. Si por ejemplo se considerase que una planta pudiese tener varias enfermedades al mismo tiempo o hubiera varias especies de plantas en una misma imagen, se utilizaría la clasificación múltiple de etiqueta múltiple. Por simplicidad, se considerará un problema de clasificación múltiple de etiqueta única, donde cada imagen contiene una única planta con una única enfermedad o plaga, en caso de tenerla.

## 6.3. Obtención y procesamiento del conjunto de datos

Una vez que se ha identificado la tarea y se conoce el tipo de problema a tratar, se ha de buscar un conjunto de datos. Esta es la tarea con diferencia en la que más tiempo se invierte al desarrollar AA pues se han de proporcionar miles de ejemplos que deben de tener suficiente calidad. La calidad no se define por la calidad de imagen, sino que el dataset que no debe contener imágenes redundantes, ni ambiguas y han de ser representativas. Por otro lado, la etiqueta de las imágenes o su descripción ha de ser la adecuada y precisa, cualquier error aquí puede ser significativo a la hora de entrenar un modelo. El artículo de Google *The Unreasonable Effectiveness of Data* habla de la importancia de esta sección, donde recalca la importancia de los datos por encima de los algoritmos (A. Halevy et al., 2009).

En este caso se optará por un dataset existente de los numerosos que hay en internet de la página Kaggle. El dataset a utilizar es el proporcionado por el usuario Sadman Sakib

Mahi<sup>14</sup> que se adapta a la perfección con el sistema que se quiere crear, al disponer de varias categorías bien organizadas con imágenes variadas y de buena calidad. Además, la distribución irregular de los datos permite ejemplificar algunas mejoras. En la Figura 17, se pueden ver algunas imágenes que contiene el dataset en cuestión con su etiqueta.



*Figura 17 Ejemplos de imágenes en el conjunto de datos seleccionado*

Para entrenar un modelo, los datos de entrenamiento han de cumplir una serie de características. En primer lugar, tienen que haber suficientes datos que clasificar. En este caso, se ha hecho un pequeño programa para contar la cantidad de imágenes que hay por categoría y el resultado es el siguiente: hay algunas categorías que tienen muy pocas imágenes a pesar de que el dataset escogido tiene más de 66000. Concretamente, de las 58 categorías, 24 tienen menos de 200 ejemplos. La cantidad de imágenes exacta por categoría se puede ver en la Tabla 1, donde se remarcán las categorías con menos de 200 ejemplares.

Categoría	Número de imágenes
Apple Apple scab	2016
Apple Black rot	1988
Apple Cedar apple rust	880
Apple healthy	1316
Bacterial leaf blight in rice leaf	40
Blight in corn Leaf	1146
Blueberry healthy	1202
Brown spot in rice leaf	40
Cercospora leaf spot	63
Cherry powdery mildew	842
Cherry healthy	684
Common Rust in corn Leaf	1306
Corn (maize) healthy	930
Garlic	49
Grape Black rot	3776
Grape Esca Black Measles	4428
Grape Leaf blight Isariopsis Leaf Spot	3444

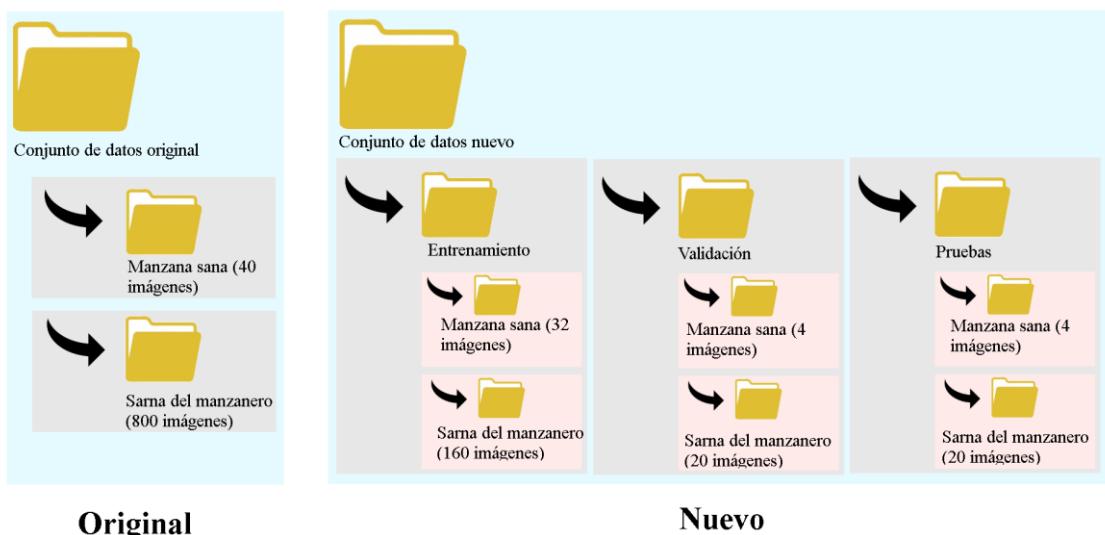
<sup>14</sup> <https://www.kaggle.com/datasets/sadmansakibmahi/plant-disease-expert>

Grape healthy	339
Gray Leaf Spot in corn Leaf	574
Leaf smut in rice leaf	40
Nitrogen deficiency in plant	11
Orange Haunglongbing Citrus greening	17600
Peach healthy	288
Pepper bell Bacterial spot	997
Pepper bell healthy	1183
Potato Early blight	1000
Potato Late blight	1000
Potato healthy	122
Raspberry healthy	297
Sogatella rice	26
Soybean healthy	4072
Strawberry Leaf scorch	888
Strawberry healthy	365
Tomato Bacterial spot	2127
Tomato Early blight	1000
Tomato Late blight	1909
Tomato Leaf Mold	952
Tomato Septoria leaf spot	1771
Tomato Spider mites Two spotted spider mite	1676
Tomato Target Spot	1404
Tomato mosaic virus	373
Tomato healthy	1273
Waterlogging in plant	7
algal leaf in tea	113
anthracnose in tea	100
bird eye spot in tea	100
brown blight in tea	113
cabbage looper	78
corn crop	104
ginger	45
healthy tea leaf	74
lemon canker	61
onion	20
potassium deficiency in plant	18
potato crop	40
potato hollow heart	60
red leaf spot in tea	143
tomato canker	19

Tabla 1. Reparto de imágenes por categoría.

Para el correcto desarrollo del sistema, es necesario dividir los datos en tres conjuntos: pruebas, entrenamiento y validación, que contienen el 10, 80 y 10 por ciento de las

imágenes totales, respectivamente. Además, para agilizar el proceso de entrenamiento que puede durar de varios minutos a horas, se limitará el número de imágenes a 200. Este proceso queda ilustrado en la Figura 18.



*Figura 18 Gestión de los datos a partir del conjunto de datos original*

En este punto se puede utilizar una herramienta propia de Keras que permite crear las etiquetas de las imágenes a partir del nombre de la carpeta que las contiene. Por ejemplo, si una imagen de maíz sano se encuentra en la carpeta llamada maíz sano, el programa lo categorizará automáticamente.

Un parámetro importante aquí es el tamaño de lotes, es decir, el tamaño de los subconjuntos de imágenes con los que se entrena el modelo, ya que la red no se entrena con todas las imágenes de forma continuada. Si por ejemplo se tiene un dataset de 500 imágenes y los lotes son de tamaño 50, la red actualizará los pesos cada vez que reciba 50 imágenes. La creación de lotes de un tamaño menor permite un uso más apropiado de la memoria, mejorando la forma en la que se ajustan los pesos de la red. La mejora se debe a que si la cantidad de imágenes de un lote es menor, la red ajustará más frecuentemente los pesos. Por otro lado, un tamaño de lote demasiado pequeño puede ralentizar el entrenamiento al tener que modificar los pesos de toda la red más veces, en este caso el tamaño de lote elegido es de 64 imágenes.

#### 6.4. Diseño y configuración de parámetros del modelo

En esta fase se tiene que especificar cuál será la medida de éxito del modelo, es decir, la relación entre aciertos y fallos al clasificar. Keras permite la creación de métricas personalizadas, pero también proporciona métricas para todo tipo de problemas: exactitud de clasificación, métricas probabilísticas, métricas de regresión, etc<sup>15</sup>. Para el caso particular de un problema de clasificación, se dispone de métricas como la exactitud binaria, exactitud categórica escasa y exactitud categórica, así como una métrica de

<sup>15</sup> <https://keras.io/api/metrics/>

exactitud genérica (Brownlee Jason, 2017). Todas las métricas de exactitud de clasificación funcionan igual, dividen el número de predicciones correctas entre el número total de predicciones. La diferencia entre todas es cómo son introducidos los datos y que, en caso de poner *binaria*, su uso queda restringido a problemas de clasificación con dos categorías. Por simplicidad en el modelo creado, se utilizará la métrica de exactitud de clasificación genérica (en inglés accuracy). Generalmente, esta métrica es muy útil cuando se tiene una cantidad de ejemplos igual o similar en cada categoría, debido a su funcionamiento (ver Figura 19). El valor deseado que se pretende conseguir es cercano al 1.

El problema de esta métrica es que, si una categoría *A* tiene 98 ejemplos y otra *B* tiene 2, el modelo, en caso de categorizar los 100 ejemplos como *A*, tendrá un valor de exactitud categórica del 98% a pesar de no categorizar correctamente ningún ejemplo de *B* (Mishra, 2018).

$$\text{Exactitud categórica} = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}}$$

*Figura 19. Fórmula de la precisión de clasificación.*

La otra métrica necesaria para configurar el modelo es la de la pérdida. En este caso se utilizará la función de entropía cruzada que funciona penalizando el error a la hora de clasificar<sup>16</sup> (ver la fórmula en la Figura 20).

$$\text{Pérdida} = H(p, q) = - \sum_x p(x) \log q(x)$$

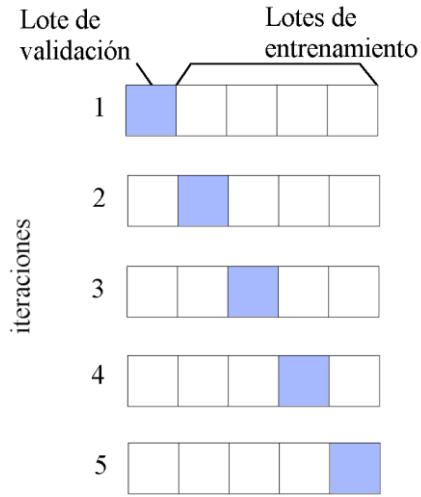
*Figura 20. Fórmula de la entropía cruzada.*

donde  $p(x)$  es la probabilidad que debería tener la clase  $x$  y  $q(x)$  es la probabilidad real que tiene  $x$ . En este caso se buscará que la pérdida tenga un valor cercano a 0.

Otra de las tareas en esta fase es definir un protocolo de evaluación. Principalmente hay tres: mantener un conjunto de datos de validación, usar validación cruzada de  $k$  iteraciones o su variación con recombinación aleatoria. Estas dos últimas son utilizadas cuando los datos disponibles son muy pocos y consisten en que los conjuntos de datos de validación y entrenamiento sean diferentes en cada iteración, como se ve Figura 21.

---

<sup>16</sup> <https://stackoverflow.com/questions/41990250/what-is-cross-entropy/41990932#41990932>



*Figura 21. La herramienta “validación cruzada de k iteraciones”.*

En el caso de recombinación aleatoria, la diferencia es que itera generalmente unas 10 veces, de forma que en cada iteración se baraja el orden de los datos. Estas técnicas de momento no se utilizarán, ya que se considera que la cantidad de datos disponible es suficiente.

## 6.5 Alcanzar línea base

Lo único que queda es utilizar los datos procesados en las fases anteriores para un cometido, clasificar. En el siguiente paso se va a crear un primer modelo que clasifique con cierta eficacia y que tenga un mínimo de funcionalidad. Con esta primera aproximación se espera alcanzar una línea base.

Para esta tarea, se realizará un modelo simple partiendo principalmente dos tipos de capas: las capas convolucionales y las capas de agrupación máxima, *Conv2D* y *MaxPooling2D* respectivamente. El funcionamiento de este tipo de capas se vio en la sección 2. Conceptos previos. Sin embargo, cabe recordar en vista del funcionamiento del modelo que, al hacer una convolución de tamaño 3 en este caso, se reducirá ligeramente el tamaño de la imagen.

Por otro lado, como se puede ver en la Figura 22, las capas de agrupación o *MaxPooling2D* dividirán exactamente en 2 el tamaño de las imágenes. Además, se puede apreciar el aumento de la cantidad de filtros que se aplican a la imagen según se va reduciendo su resolución, alargando su arquitectura con más filtros.

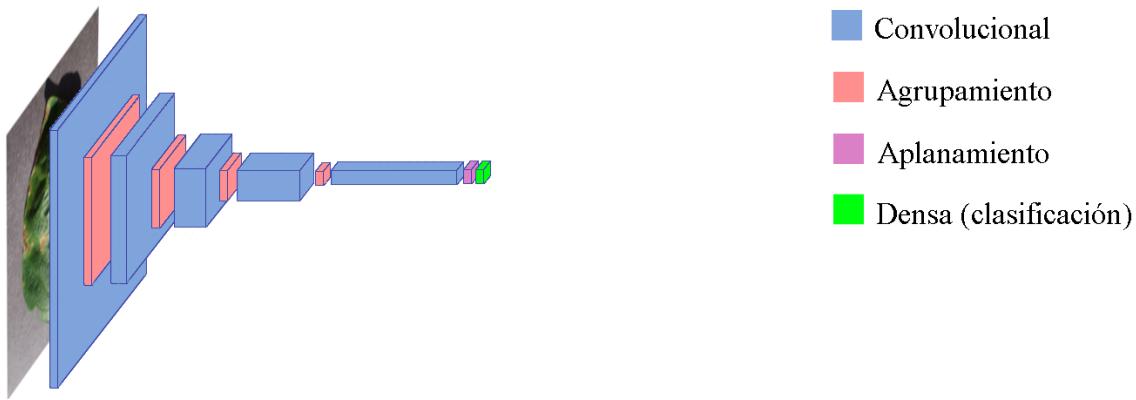


Figura 22. Arquitectura del modelo 0.

De esta forma, se empieza con 32 filtros y una resolución de 256 x 256 que se divide en dos en cada capa de agrupamiento hasta terminar con una resolución de 12 x 12. Los filtros, al contrario, se multiplican por dos cada vez que se pasa por una capa de agrupamiento, lo que da lugar a 64 filtros en la segunda capa, 128 en la tercera y terminar en la última capa con 512. Esto da lugar, gracias a la capa de aplanamiento marcada en rosa, a un vector de 36.864 valores resultante de multiplicar el número de filtros por el número de píxeles de la última capa. El vector es utilizado finalmente por la última capa, una capa densa con función de activación *SoftMax*, que se encarga de clasificar el resultado en una de las 58 categorías.

Por último, en cuanto a capas, hay que señalar la existencia de una capa de redimensionamiento al principio que cambia los valores RGB de 0 a 255, a valores comprendidos entre 0 y 1 para facilitar cálculos.

Con el modelo creado se pasa a la siguiente fase, entrenar el modelo. En esta fase se dan valores a las variables de los perceptrones que forman cada una de las capas para que sean capaces generalizar a través de prueba y error. La Figura 23 representa el proceso de entrenamiento del modelo a lo largo de 30 etapas señaladas como *Epochs*. En cada una de las etapas se entrena el modelo con todas las imágenes pertenecientes al conjunto de entrenamiento, pero los pesos se actualizan por cada lote de 64 imágenes.

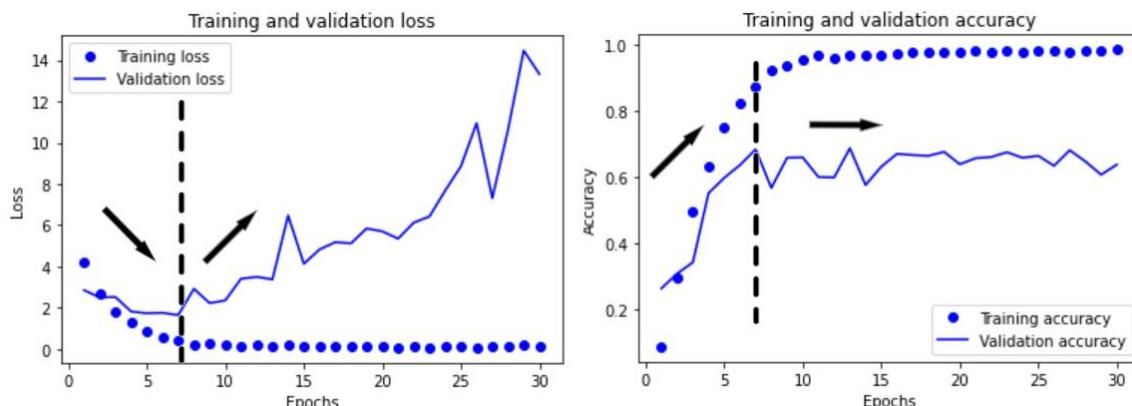


Figura 23. Gráficas de pérdida y precisión del modelo 0

Una vez que el modelo ha usado todas las imágenes de entrenamiento en una etapa, se genera el informe correspondiente con la pérdida y precisión que ha tenido el modelo mientras se entrenaba, generando respectivamente los *Training loss* y *Training accuracy* de las gráficas.

En las gráficas también aparecen los términos *Validation loss* y *Validation accuracy*, que se generan gracias al conjunto de validación. Estos términos sirven para arrojar luz sobre el estado del modelo en la fase de entrenamiento. Para generar dichos valores, al final de cada etapa, se realiza una prueba con el conjunto de datos de validación que contiene imágenes con las que nunca se ha entrenado el modelo antes. El resultado de la prueba de validación es el rendimiento del modelo ante imágenes nuevas.

Si se comparan los datos de validación y los datos de entrenamiento obtenidos, se puede interpretar más información. Como se puede apreciar en el gráfico de la derecha, el modelo poco a poco alcanza una mayor precisión de validación, señalada como *Validation accuracy* hasta llegar a la etapa 7. La precisión de entrenamiento o *Training accuracy*, no obstante, no para de crecer. Este estancamiento de la precisión de validación significa que el modelo no es capaz de mejorar su capacidad predictiva y que, al seguir entrenándolo con el mismo conjunto de imágenes de entrenamiento, lo que sucede es que se empieza a memorizar las imágenes. Debido a ello la precisión de entrenamiento no para de mejorar.

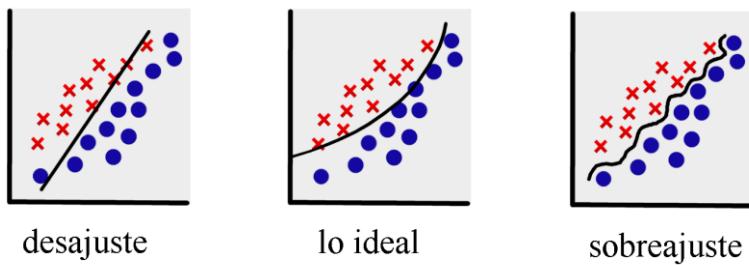
Para contrastar esta información existe la gráfica de la izquierda que muestra el desempeño del modelo a través de la pérdida. Al contrario de la gráfica anterior, los valores se corresponden a un mejor desempeño del modelo cuanto menor sea el valor. Como se puede observar en ella, los valores de pérdida de validación se disparan en la etapa 7, lo que resalta que el modelo está memorizando las imágenes a partir de ese punto, ya que la pérdida de entrenamiento, al contrario, sigue decreciendo.

Los resultados obtenidos concluyen que el modelo sufre de sobreajuste a partir de la etapa 7. Además, se ha realizado una prueba en la etapa 7, donde el modelo tiene su mejor estado y otro después de ser entrenado por 30 etapas. El resultado de las pruebas es que el modelo tiene una precisión con datos que nunca ha visto del 67,8% en la etapa 7, mientras que después de 30 etapas tiene un 66,8%. Este resultado evidencia que entrenar un modelo por demasiadas épocas hace que empeore.

En este punto se puede decir que se ha creado un modelo que alcanza una línea base que tiene una precisión del 68% aproximadamente. Este resultado implica que no se tendrá que volver a pasos anteriores, como se indicó en la metodología.

### 6.5.1 El sobreajuste

El sobreajuste (en inglés *overfitting*) se da cuando el modelo no es capaz de aprender más y memoriza los datos que se le proporcionan en la fase de entrenamiento. El sobreajuste es el caso de la adaptación del modelo ante un conjunto de datos definido, pero sin ser capaz de generalizar. Esta definición se puede entender mejor con el ejemplo de la Figura 24, donde se puede observar, de izquierda a derecha, las tres fases por las que generalmente pasa un modelo cuando es entrenado.



*Figura 24. Ejemplos de ajustes de un modelo*

Sin embargo, la detección de sobreajuste es bastante sencilla, sólo hace falta mirar la gráfica de pérdida. Si la pérdida de validación y entrenamiento bajan, se dice que el modelo está en una fase de desajuste. En este punto es donde la pérdida de validación detiene su decrecimiento y empieza a aumentar es cuando comienza el sobreajuste. Lo ideal es que el modelo pare de entrenar justo antes de sufrirlo.

Se puede comprobar que el modelo de la Figura 23 representa los signos de sobreajuste que se han mencionado. Según el flujo de trabajo universal del AA, una vez que se alcanza la línea base, hay que conseguir que el modelo sufra de sobreajuste para luego combatirlo. De esta forma, si el modelo sufre de sobreajuste, significa que se ha conseguido llegar al límite de su utilidad. Nótese que podría haber ocurrido que el modelo hubiera alcanzado la línea base sin haber sufrido sobreajuste, por ejemplo, si se hubiera entrenado el modelo por 6 etapas en lugar de 30. En ese caso el modelo podría tener una buena precisión, pero no desarrollar todo su potencial. Habría que aplicar una serie de técnicas, como poner capas más grandes o entrenarlo más etapas. Como se ya se ha alcanzado el sobreajuste, lo que queda será combatirlo y alargar la fase de desajuste.

## 7. Experimentos y resultados

A partir de ahora lo que se hará es pasar a la siguiente fase del desarrollo del modelo, reducir el sobreajuste y aumentar la generalización.

### 7.1. El primer modelo: análisis y mejora

Existen varias formas para conseguir que la fase de desajuste dure más y, por ende, aumentar la precisión del modelo. Una de las formas más sencillas de combatir el sobreajuste es con la parada temprana, es decir, dejar de entrenar el modelo justo antes de que sufra el sobreajuste. En Keras existe una función que guarda el modelo siempre que la pérdida de una etapa sea menor que la pérdida mínima existente en el momento. Esta función ha permitido guardar el modelo en la época 7 a pesar de haberlo seguido entrenando, pero como se verá a continuación, hay más formas de mejorar el modelo.

### 7.2. Aumento de imágenes

Una de las razones por las que el modelo no es capaz de alcanzar una mayor precisión es porque esté limitado por la cantidad de datos disponibles. Para resolver esta limitación, Keras ofrece una herramienta<sup>17</sup> de fácil manejo que permite crear imágenes “nuevas” a partir de otras. Para ello, la herramienta toma una imagen del dataset, la procesa con algunas variaciones como ampliación o inclinación y luego, la guarda como una imagen nueva<sup>18</sup>. Su funcionamiento se detalla en la Figura 25, donde se pueden apreciar las variaciones ejercidas sobre una imagen procedente del conjunto de datos original.

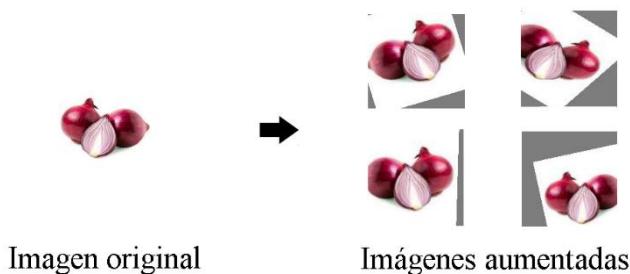


Figura 25. Ejemplo de aumento de datos.

De esta forma, se ha conseguido que el conjunto de datos, por un lado, tenga muchas más imágenes de ejemplo y, por otro lado, que todos los subconjuntos tengan la misma cantidad de imágenes. En la Figura 26 se puede apreciar cómo ha afectado esta adición en el conjunto de datos. Ahora todos los subconjuntos tienen el mismo tamaño, a diferencia de lo que se tenía inicialmente.

<sup>17</sup> [https://stepup.ai/train\\_data\\_augmentation\\_keras/](https://stepup.ai/train_data_augmentation_keras/)

<sup>18</sup> [https://stepup.ai/exploring\\_data\\_augmentation\\_keras/](https://stepup.ai/exploring_data_augmentation_keras/)

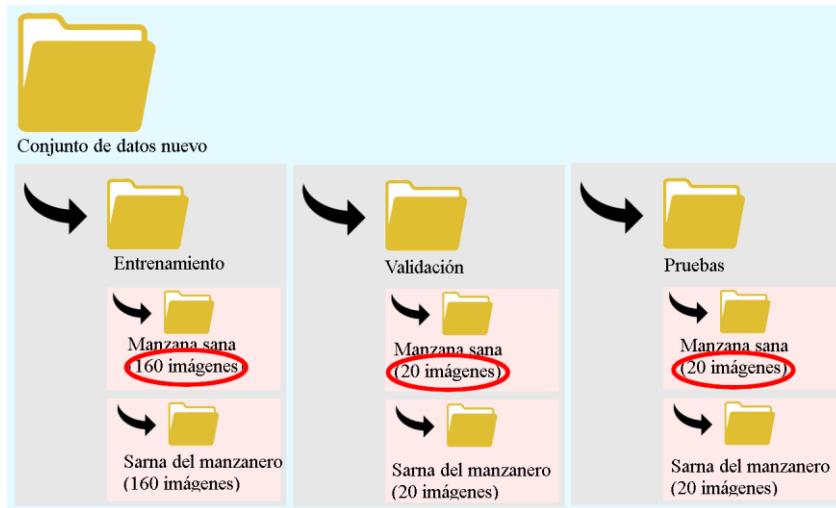


Figura 26. Resultado del aumento de datos en el conjunto de datos actual.

Este cambio produce que el modelo mejore su precisión al disponer de más datos, pero si además los subconjuntos de datos tenían una cantidad de imágenes demasiado dispar, podría exponer un fallo de métricas anterior, como se comentó en 6.4. Diseño y configuración de parámetros del modelo.

En la Figura 27 se pueden apreciar los cambios que han producido el aumento de datos en el modelo. En el nuevo modelo, señalado en rojo, y al que se referirá como modelo 1, el sobreajuste aparece ahora en la etapa 5, en lugar de la 7. La pérdida aumenta de forma ligeramente más suave que en el modelo anterior. Por otro lado, la precisión parece haberse visto reducida, probablemente por haber solucionado el problema de los subconjuntos de tamaños diferentes.

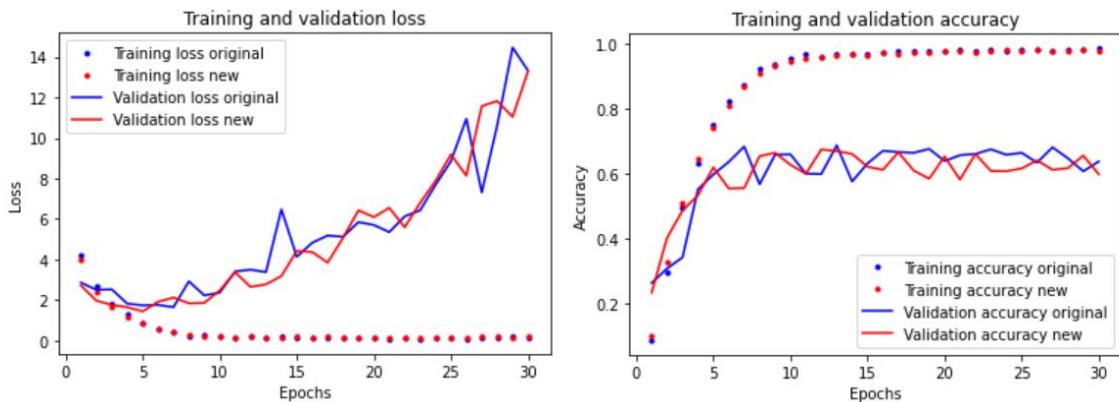


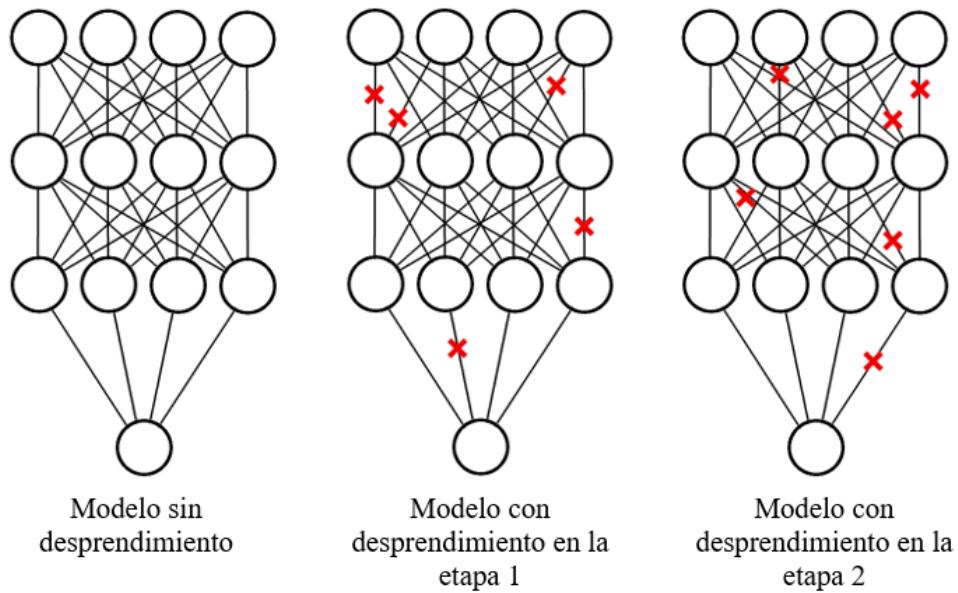
Figura 27. Gráficas de pérdida y de precisión del modelo 0 y del modelo 1.

De hecho, la precisión en este modelo al probarlo es del 59% en la etapa 5 y del 58% tras finalizar las 30 etapas. Este resultado significa una reducción del valor de la precisión de casi un 10% con respecto el modelo anterior, alcanzando 68% en la etapa 7 y 67% en la etapa 30 aproximadamente. No obstante, lejos de ser un resultado peor, significa que la

precisión del modelo anteriormente se mostraba de forma errónea y que ahora es más real, debido a que ahora cada categoría tiene exactamente los mismos ejemplos de entrenamiento.

### 7.3. Desprendimiento

La técnica de desprendimiento (en inglés drop-out) hace que en la fase de entrenamiento y de forma aleatoria, algunos de los valores de salida de cada capa en lugar de tener su valor real, se inicialicen a cero. La puesta en cero de estos valores tiene el efecto de retrasar la aparición del sobreajuste, ya que, aunque el modelo reciba la misma imagen en cada etapa, los valores que proporcionan sus capas no serán las mismas. En otras palabras, se dificulta la capacidad de memorización del modelo. En la Figura 28 se pueden ver un ejemplo del funcionamiento del desprendimiento. La imagen de la izquierda muestra un modelo sin desprendimiento, los siguientes modelos muestran la aplicación del desprendimiento en diversas capas que se muestran mediante cruces rojas para representar los puntos de desconexión entre capas.



*Figura 28. Funcionamiento del desprendimiento en modelos.*

Para aplicar esta técnica, sólo se tiene que añadir al modelo existente una capa nueva al final que inicialice a 0 algunos de los valores aleatorios del vector resultante de la capa de aplanamiento. En la Figura 29 se muestra un ejemplo de aplicación.

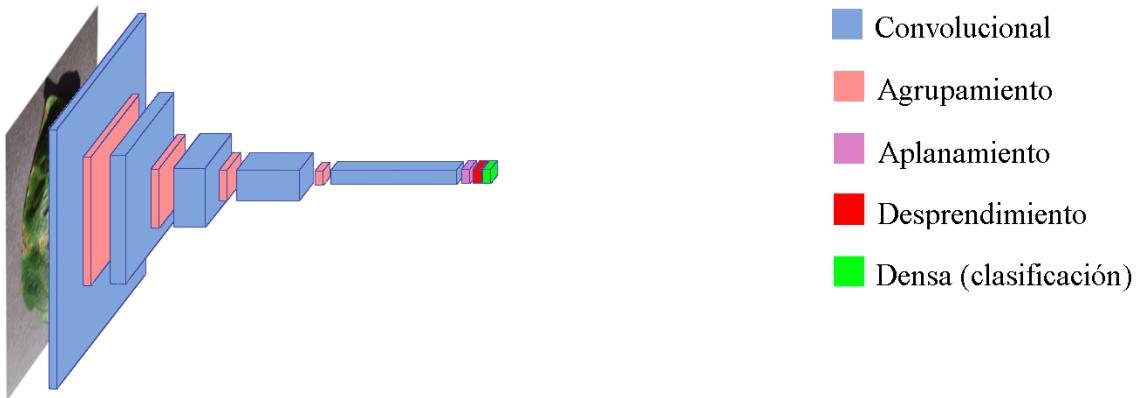


Figura 29. Arquitectura del modelo 2.

Después de añadir la capa de desprendimiento se obtienen los siguientes resultados (ver Figura 30).

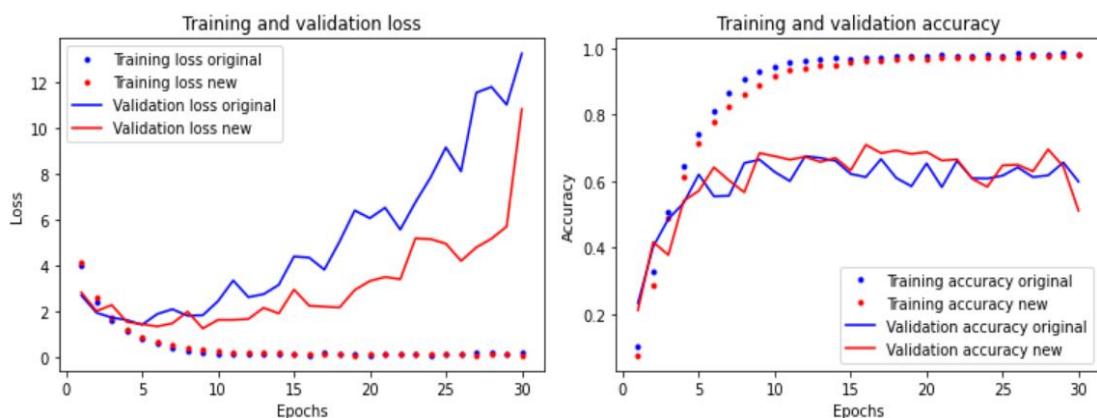


Figura 30. Gráficas de pérdida y de precisión del modelo 1 y del modelo 2.

En color rojo se muestran los resultados de aplicar desprendimiento y en azul los resultados del modelo anterior. Se puede apreciar cómo, en primer lugar, en el modelo con desprendimiento se retarda la aparición del sobreajuste hasta la etapa 9, cuando antes aparecía en la etapa 5. En segundo lugar, este cambio también hace que el sobreajuste aumente de forma mucho más lenta que en el modelo anterior, lo que evidencia que el desprendimiento evita que el modelo memorice.

Al haber retenido la aparición del sobreajuste, el modelo ha podido alcanzar un mejor valor de precisión, ya que se sigue aplicando la configuración de parada temprana que permite guardar el modelo antes de sufrir sobreajuste. En el modelo 1 la parada temprana se ejecutaba en la etapa 5, debido a que es en ella donde la pérdida se empezaba a incrementar, a pesar de que la precisión no tenía su punto más alto. Como el sobreajuste con el nuevo modelo aparece en la etapa 9, se ha podido guardar el modelo con la precisión correspondiente a dicha etapa, entorno al 68%, en lugar del 62% de la etapa 5 del modelo 1. Este modelo con la inclusión de una capa de desprendimiento alcanza en pruebas una precisión del 69%, 10 puntos más que el modelo anterior.

#### 7.4. Arquitectura más profunda y conexiones residuales

En este apartado hace falta aplicar ciertas modificaciones al modelo existente de forma gradual con intención de hacerlo más grande, ya que es necesario para poder aplicar conexiones residuales. El paso de los valores de una capa a otra en un modelo actuará como el juego del teléfono roto, cuantas más tenga, más ruido se generará entre ellas. De esta forma los datos que reciban las capas tendrán cada vez menos sentido, lo que estancará el modelo en tamaño. Es justo ese problema el que intenta solucionar las conexiones residuales.

Inicialmente para notar ese estancamiento de tamaño el modelo ha de ser lo suficientemente grande. A continuación, se muestra cómo afecta la adición de diferentes capas al modelo para luego ver en qué ayuda la aplicación de las conexiones residuales. En la Figura 31 se puede ver la arquitectura del nuevo modelo más profundo, el modelo 3. Este modelo es el resultado de duplicar o triplicar las capas convolucionales del modelo 2, lo que resulta en que el número de capas y filtros, respectivamente, quede de la siguiente forma: 1 x 16, 1 x 32, 2 x 64, 2 x 128 y 3 x 256.

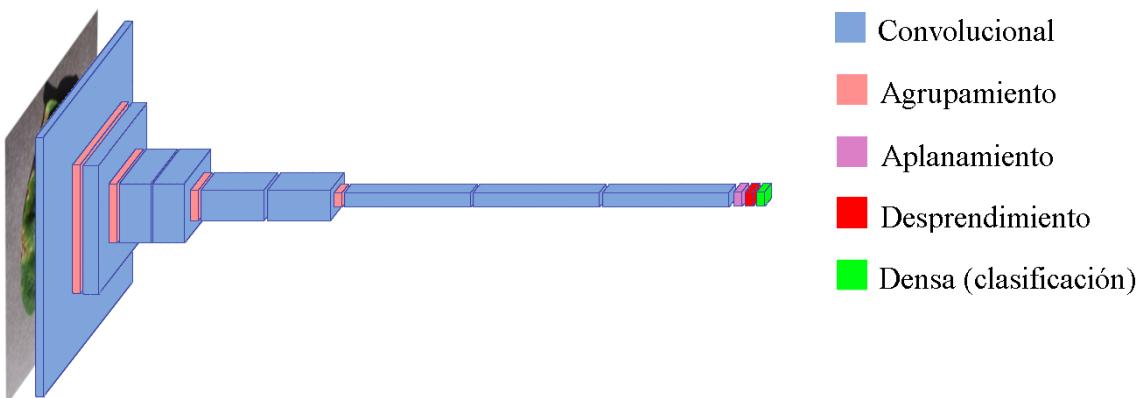


Figura 31. Arquitectura del modelo 3.

La ampliación no ha sido aleatoria, se han añadido capas y filtros siguiendo la norma de que más filtros en capas más profundas es mejor. El motivo de aplicar esa norma es que los filtros de capas más profundas son capaces de percibir patrones más complejos.

Después de la ampliación del modelo, se ha entrenado para obtener los resultados que se muestran en la Figura 32, donde se comparan con los del modelo anterior. Se puede apreciar cómo, al haber hecho el modelo más grande, le cuesta más no solo alcanzar el sobreajuste, sino ajustarse en general. A partir de la etapa 26, la pérdida del modelo 3, representada en rojo, parece empezar a aumentar, aunque no hay una evidencia clara de sobreajuste en la gráfica de pérdida. Sin embargo, la mejora de precisión no se ha visto mejorada, siendo constante a partir de la etapa 17, cuando en el modelo anterior se alcanzaba un sobreajuste claro en la etapa 9. Al ser el modelo más profundo se ha ralentizado el ajuste y no ha mejorado, lo que evidencia que un mayor tamaño del modelo no es siempre mejor. Esto puede deberse, por ejemplo, a la pérdida de información al hacer el modelo más profundo.

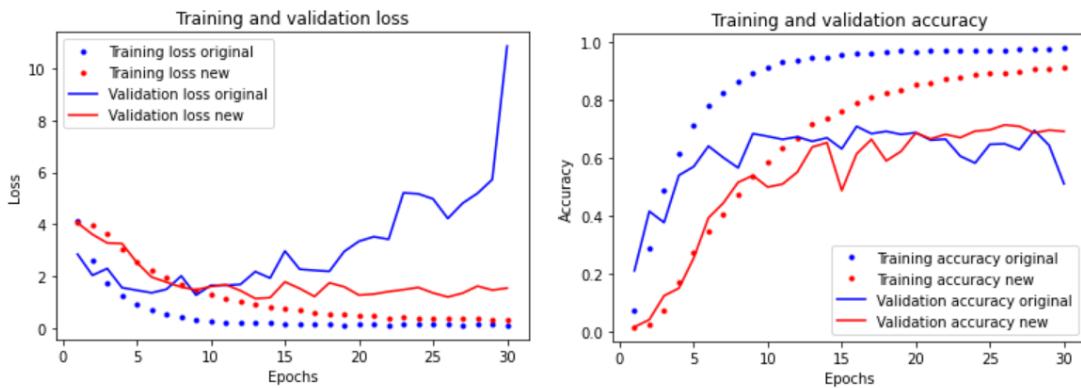


Figura 32. Gráficas de pérdida y de precisión del modelo 2 y del modelo 3.

#### 7.4.1. Capas densas al final del modelo

Una de las prácticas más utilizadas es la de poner una o varias capas densas al final de un modelo. En la Figura 33 se puede ver la arquitectura resultante de haber añadido al modelo anterior una serie de 3 capas densas de 64 perceptrones con activación *ReLU*.

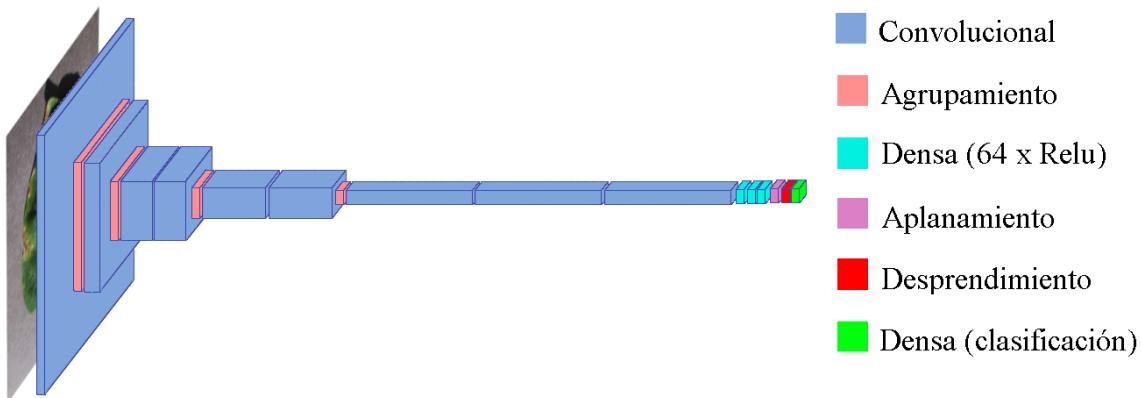


Figura 33. Arquitectura del modelo 3.1.

En la Figura 34 se puede ver la comparación del rendimiento en el entrenamiento del modelo 3.1 y del modelo anterior, el modelo 3. En primer lugar, la curva de aprendizaje del modelo 3.1, mostrada en rojo, es mucho más suave en ambas gráficas, tanto en la fase de entrenamiento como la de validación, lo que indica que de esta forma al modelo le cuesta más ajustarse. En segundo lugar, la precisión y pérdida del nuevo modelo se han visto empeoradas. Ambas consecuencias son, teóricamente, resultado de hacer el modelo más profundo.

La principal utilidad de añadir las capas densas es que la brecha entre los datos de entrenamiento y los datos de validación es mucho más pequeña, permitiendo al modelo entrenar por más etapas sin sufrir una penalización tan elevada por el sobreajuste. Esto es visible en las gráficas de la Figura 34, donde las líneas y los puntos están más próximos entre sí en el caso del nuevo modelo, mostrado en rojo.

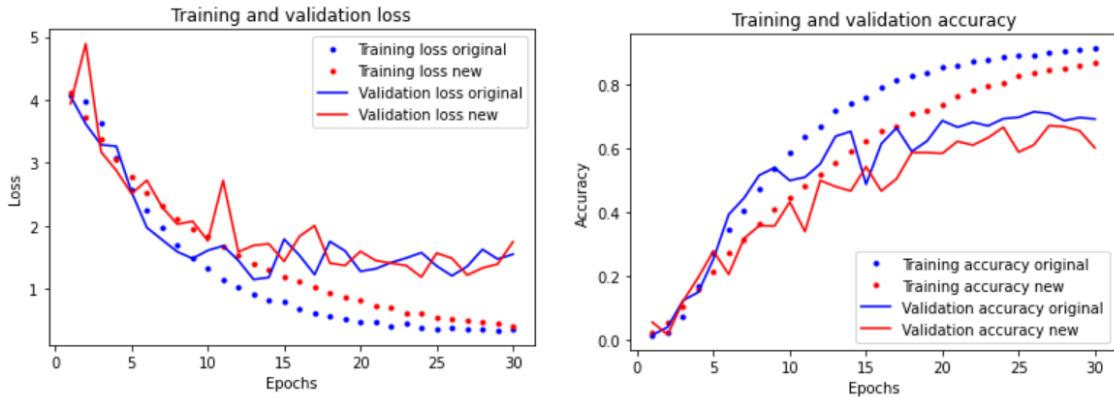


Figura 34. Gráficas de pérdida y de precisión del modelo 3 y del modelo 3.1.

#### 7.4.2. Conexiones residuales

Uno de los motivos de estancamiento del modelo 3 ampliado es, como se mencionó al principio de este punto, el problema del teléfono roto entre las capas. Para solucionarlo, existe un mecanismo llamado conexiones residuales que funciona de la siguiente forma: la información de salida de una capa es mezclada con la de una de las capas siguientes, saltándose algunas intermedias. En la Figura 35 se puede ver el funcionamiento a modo de ejemplo, además de introducir la nueva arquitectura. El inicio de la flecha indica la capa que copia la información y, después de una capa de agrupamiento y otra convolucional, esa información se utiliza para mezclarse con la existente al final de la flecha.

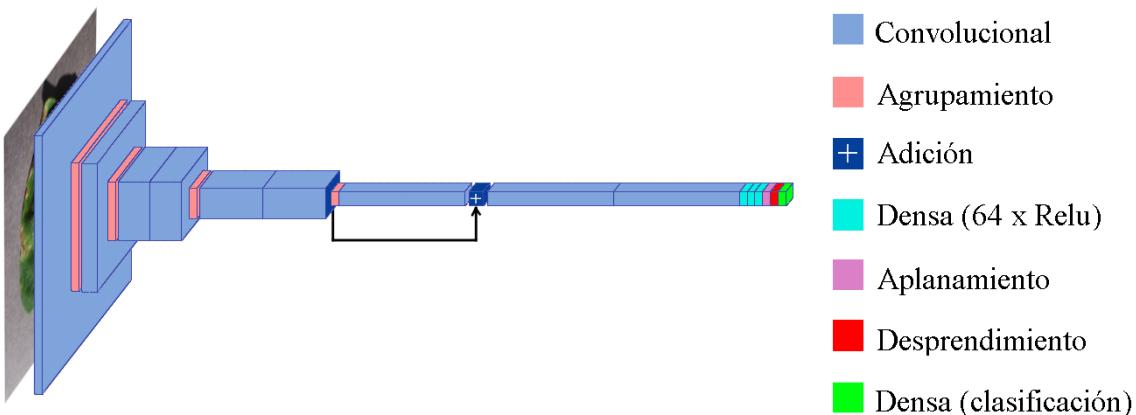
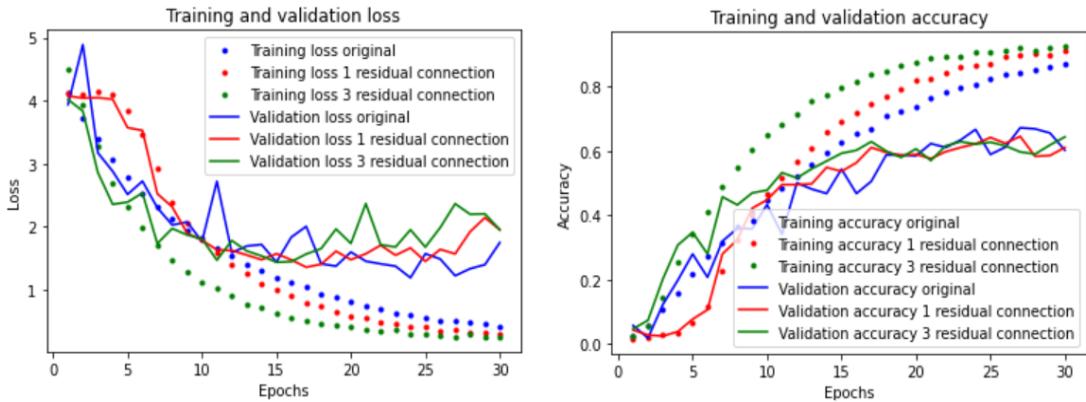


Figura 35. Arquitectura del modelo 3.2.

Además de añadir una conexión residual, se han ejecutado pruebas para comparar el rendimiento dependiente al número de conexiones residuales. Los resultados pueden verse en la Figura 36, donde aparece también el modelo 3.3 que tiene 2 conexiones residuales más que el modelo 3.2. Se aprecia cómo cuantas más conexiones residuales, más pronunciado es el sobreajuste sin acelerar significativamente su aparición, ya que en todos los casos el sobreajuste aparece sobre la etapa 15. Esto es un indicio de que el modelo tiene más capacidad de ajuste. Además, se ve como en la etapa 15, en la gráfica de precisión, tiene más precisión el modelo con más conexiones residuales.



## 7.5. Experimentos con el modelo actual

En el fichero correspondiente al modelo 3, se puede apreciar cómo hay diversos modelos que abarcan desde el modelo 3.4 hasta el modelo 3.12. Estos modelos son el resultado de una serie de experimentos realizados con la intención de arrojar luz a lo que hace que el modelo empeore o mejore usando únicamente las herramientas vistas hasta ahora. El esquema de modificación de los modelos es el mostrado en la Figura 37. Lo que se muestra es un árbol en el que los hijos están hechos a partir de una modificación del modelo padre, dicha modificación está señalada en los paréntesis.

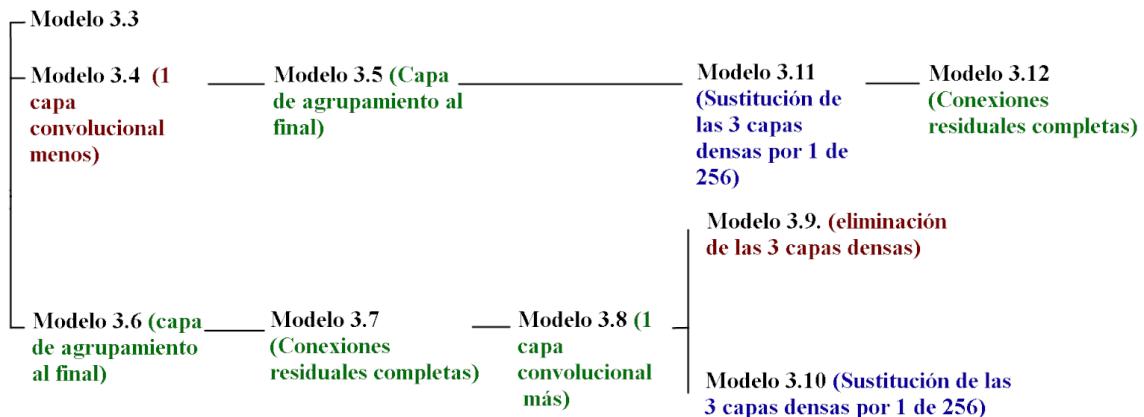


Figura 37. Esquema de modelos experimentales.

### 7.5.1. Modelo 3.4

El modelo 3.3 es el correspondiente al que tiene 3 conexiones residuales. En su momento se mencionó que se ha reducido el ruido generado entre capas, procedente de la profundidad elevada del modelo. Para contrastar la hipótesis surge el modelo 3.4, donde se ha eliminado una de las 3 últimas capas convolucionales del modelo que tienen 256 filtros para ver qué efecto tiene. El resultado se muestra en la Figura 38, donde se puede apreciar que el nuevo modelo tiene mejor desempeño, ya que el sobreajuste aparece después y la pérdida y exactitud mejoran significativamente.

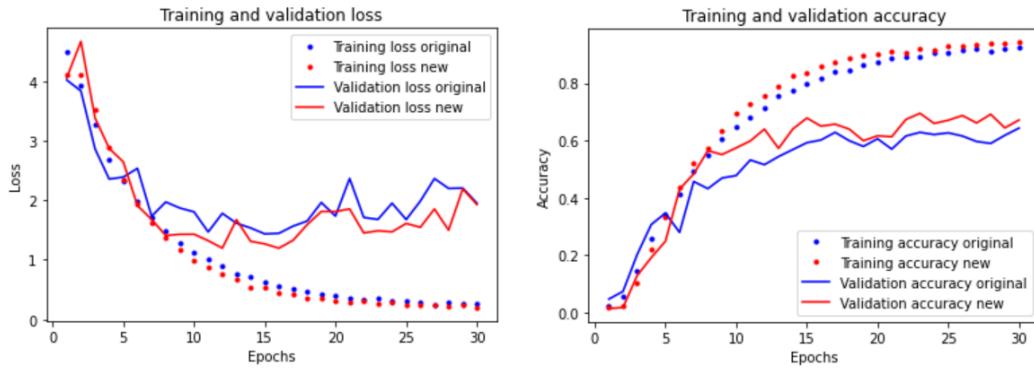


Figura 38. Modelos 3.3 y 3.4.

Si el modelo mejora tanto cuando se reduce su profundidad, es que las conexiones residuales añadidas pueden no ser suficientes, ya que se sigue perdiendo información a medida que crece la profundidad del modelo.

### 7.5.2. Modelo 3.5

Otro de los problemas que tenía el modelo 3.3 era que quizás a las tres capas densas añadidas recientemente le llegan demasiados parámetros. En consecuencia, surge el modelo 3.5, una modificación del modelo 3.4 al que se ha añadido una capa de agrupamiento al final. Este modelo resultante ha alcanzado valores de precisión muy buenos, los mejores hasta el momento, subiendo de un 69% a un 72%, lo que confirmaba la teoría (ver Figura 39).

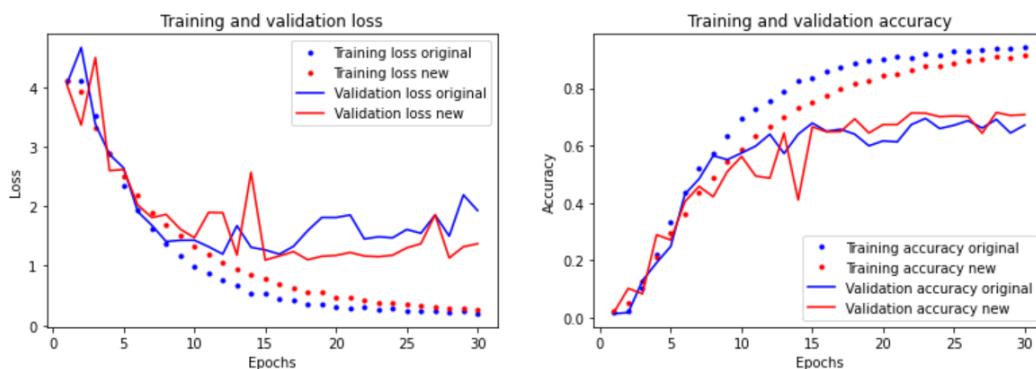


Figura 39. Modelos 3.4 y 3.5.

Gracias a los experimentos anteriores se concluyen dos cosas: que el modelo 3.3 es demasiado profundo; y que la última capa encargada de clasificar recibe demasiados parámetros, ya que mejora la eficacia de clasificación al reducir a la mitad los parámetros con la capa de agrupamiento.

### 7.5.3. Modelo 3.6

El modelo 3.6. es una modificación directa hacia el modelo de partida, el modelo 3.3, donde se incluye la capa de agrupamiento que se le puso al modelo 3.5. La mejora es considerable, pero debido al ruido de la profundidad del modelo, no llega a tener tan buen resultado como el modelo 3.5 que tiene una capa convolucional menos, quedándose en un 67% de precisión (ver Figura 40).

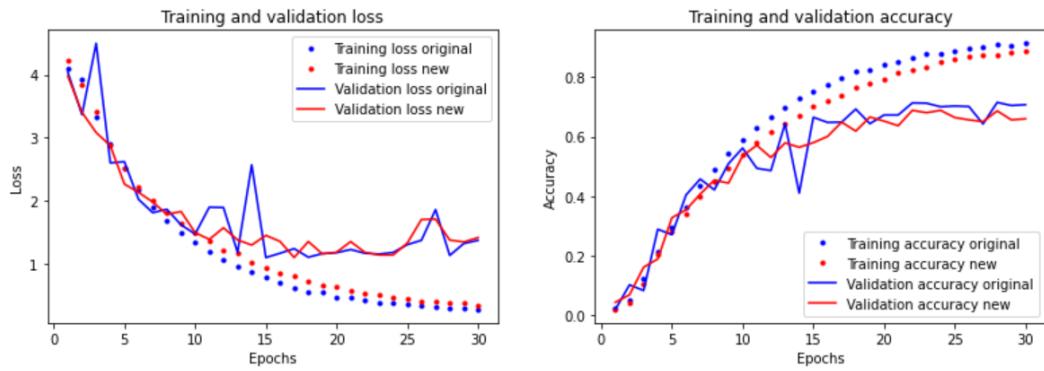


Figura 40. Modelos 3.5 y 3.6.

#### 7.5.4. Modelo 3.7

El modelo 3.7 es el remedio directo al problema de la profundidad del modelo. En este modelo se aplican conexiones residuales en la totalidad del modelo 3.6. Los resultados son satisfactorios, pero no los esperados, alcanzando un 72% de precisión. Esto significa que tan sólo iguala al modelo que tiene una capa menos, el modelo 3.5 (ver Figura 41).

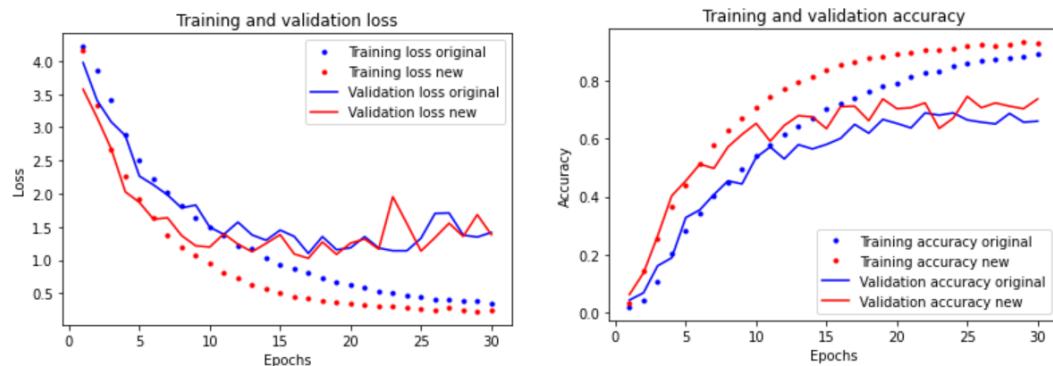
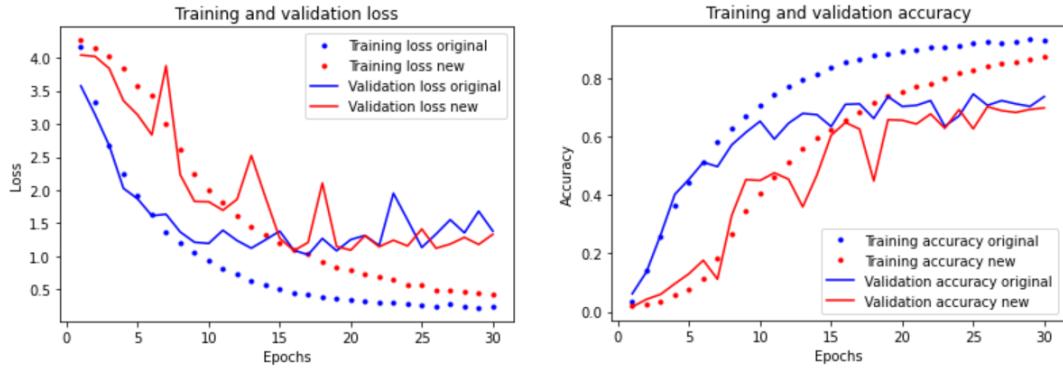


Figura 41. Modelos 3.6 y 3.7.

#### 7.5.4. Modelo 3.8

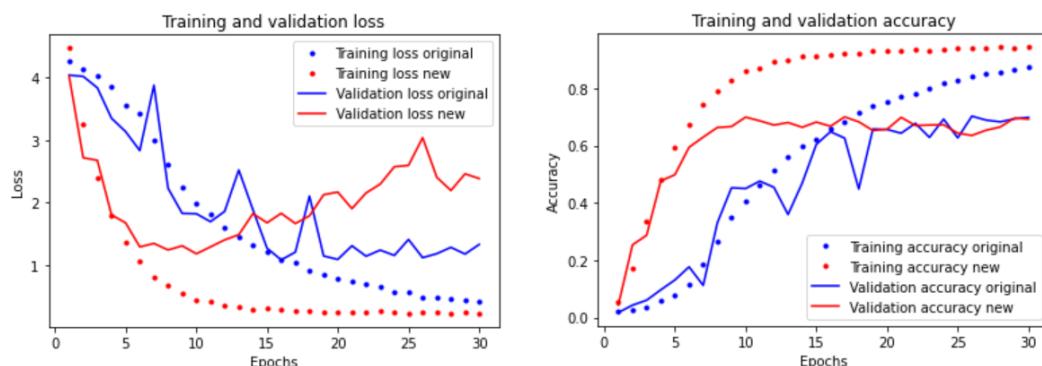
El modelo 3.8 surge de ver hasta qué punto son útiles las conexiones residuales, ya que aun con ellas, se tiene el mismo rendimiento en los modelos 3.7 y 3.5, este último con una capa convolucional menos. Se añade entonces una capa convolucional nueva de 512 filtros y otra capa de agrupamiento al final al modelo 3.7. La razón de poner la capa de agrupamiento es para que las tres últimas capas densas reciban menos parámetros, ya que se sospecha de un posible cuello de botella en ellas. Si el resultado es similar al modelo anterior, podría significar que las capas densas perdían información. El resultado no es el esperado: se ralentiza el ajuste y no mejora el rendimiento (ver Figura 42).



*Figura 42. Modelos 3.7 y 3.8.*

### 7.5.5. Modelo 3.9

Para verificar si las 3 capas densas producen cuello de botella o no, en este modelo se han eliminado, con intención de ver el efecto que tienen sobre los resultados. Como resultado de eliminarlas, en este modelo se aprecia un comportamiento muy diferente: se ajusta más rápido y tiene una curva de sobreajuste mucho más pronunciada. Sin embargo, como se muestra en la Figura 43, el rendimiento general, salvo la cantidad de etapas necesarias para entrenar, no mejora, siendo similar al del modelo anterior con las 3 capas densas.



*Figura 43. Modelos 3.8 y 3.9.*

Este modelo concluye que las capas densas tal y como están configuradas no aportan demasiada mejora.

### 7.5.6. Modelo 3.10

Este modelo se construye a partir del modelo 3.8, sustituyendo las 3 capas densas de 64 perceptrones por una única capa, pero de 256 perceptrones. Este cambio produce una mejora considerable respecto al modelo 3.8 que no tenía capas densas, apreciable en la Figura 44. Cabe mencionar que se han hecho varias pruebas, empezando por añadir una capa de 64 que directamente hacía no funcionar el modelo, hasta terminar con varias capas de 512. El resultado de las pruebas muestra que las capas demasiado pequeñas requieren muchas capas y pierden información, mientras que las capas demasiado grandes funcionan peor al producir sobreajuste. El punto medio se ha encontrado entre las opciones de 2 capas de 128 y 1 capa de 256, optando por esta última por tener un rendimiento ligeramente mejor.

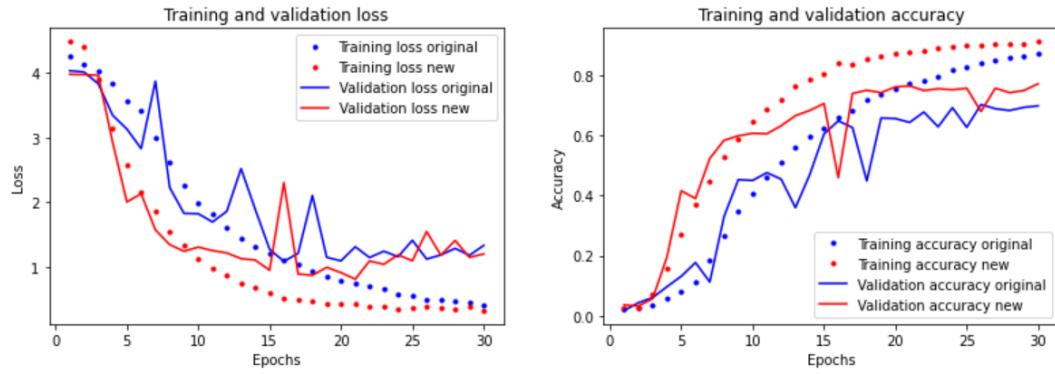


Figura 44. Modelos 3.8 y 3.10.

### 7.5.7. Modelo 3.11

Por último, los modelos 3.11 y 3.12, son el resultado de experimentar añadiendo todo lo visto hasta ahora al modelo 3.5 que tiene dos capas convolucionales menos que el modelo 3.10 con intención de contrastar los resultados. Con el modelo 3.11 se ha hecho una capa densa de 256 perceptrones sustituyendo a las 3 de 64 perceptrones anteriores. Este cambio produce una mejora significativa, evidenciando de nuevo el cuello de botella ejercido anteriormente por las 3 mencionadas capas (ver Figura 45).

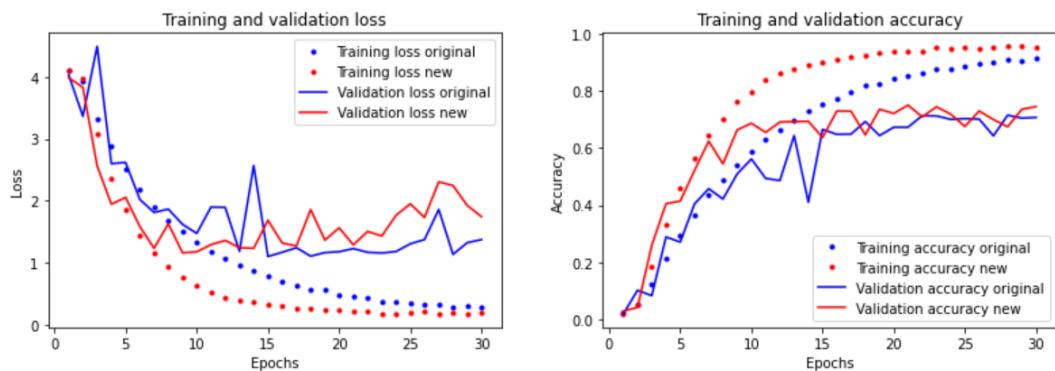


Figura 45. Modelos 3.5 y 3.11.

### 7.5.8. Modelo 3.12

El modelo 3.12 tiene el añadido de conectar todas las capas con conexiones residuales y tiene en torno a un 2% más de precisión que su antecesor (ver Figura 46). Según los resultados obtenidos en este último modelo y con los obtenidos en los modelos 3.7 y 3.10, se clarifica la utilidad de las conexiones residuales para resolver el problema de profundidad del modelo. Concretamente, gracias a las conexiones residuales, el modelo 3.10 alcanza una precisión del 76,5% contra el 72% del modelo 3.12 que tiene dos capas convolucionales menos.

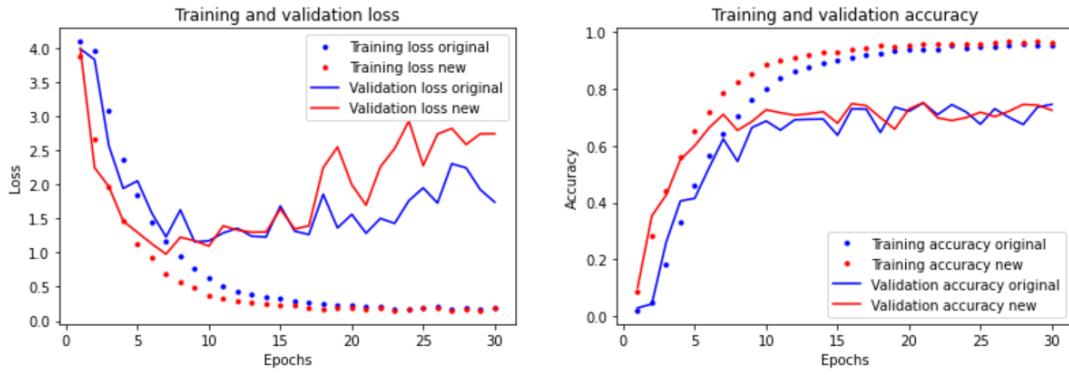


Figura 46. Modelos 3.11 y 3.12.

## 7.6. Normalización de lotes

Los valores de los datos que se introducen en un modelo son numéricos. Por ejemplo, las imágenes que se introducen en los ejemplos vistos son simplemente cadenas de números que simbolizan el color de cada píxel. Esos números suelen transformarse en otros con una escala numérica más fácil de manejar, por ejemplo, de 0 a 1. Este proceso recibe el nombre de normalización de datos. Generalmente, lo más fácil es normalizar los datos antes de introducirlos en el modelo. No obstante, después de la primera capa esa normalización se degrada en las siguientes (Martínez, 2019). En respuesta a este problema existe la normalización de lotes que tiene muy buenos resultados (Ioffe & Szegedy, 2015), ya que aplica la normalización para todas las capas del modelo. Los resultados teóricos de aplicar esta técnica son principalmente una aceleración en el proceso de entrenamiento y una mejora general de los resultados.

En la práctica lo que se ha visto es que el modelo se ajusta mucho más rápido a la vez que se ralentiza la aparición del sobreajuste, permitiéndole estar en la fase de desajuste durante más etapas. La comparación entre el rendimiento del modelo sin normalización de lotes y con ella corresponden a los modelos 3.10 y 4 respectivamente, y se pueden observar en la Figura 47. Se puede apreciar una curva de aprendizaje mucho más pronunciada en las primeras etapas del nuevo modelo, resultado de la aceleración producida por la normalización de lotes. Además, la aparición del sobreajuste se retrasa de la etapa 21 del modelo 3.10 a la etapa 25 en el modelo 4, lo que produce una mejora de precisión en favor de este último.

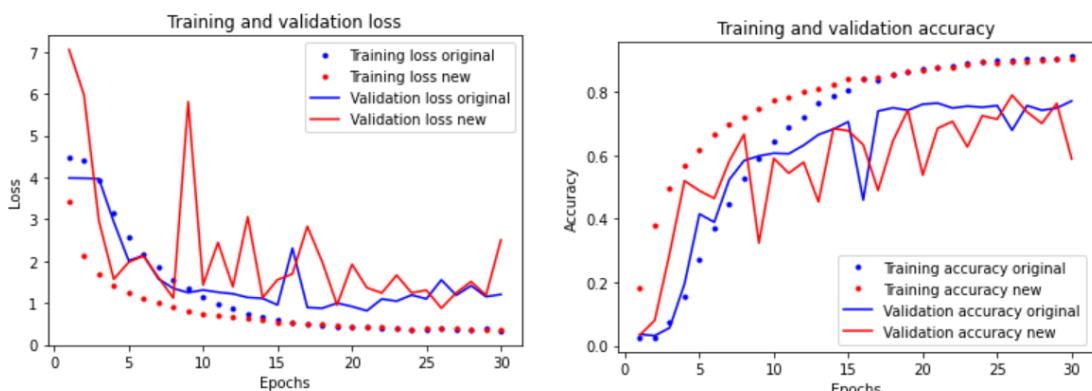
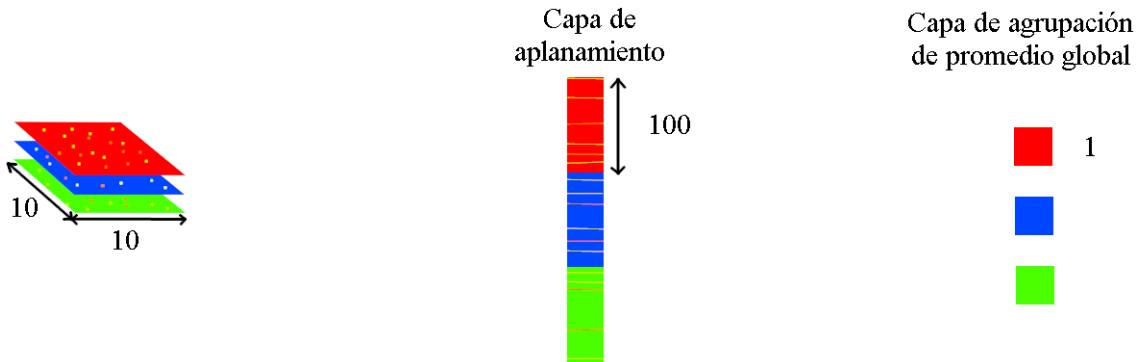


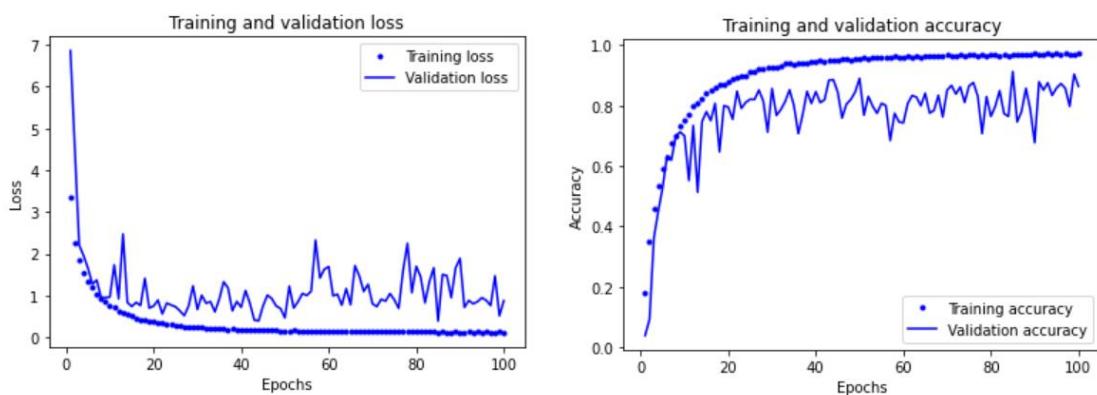
Figura 47. Gráficas de pérdida y de precisión del modelo 3.10 y del modelo 4.

Además, como extensión a este modelo, se ha sustituido la capa de aplanamiento que tenía el modelo 4 por una capa de agrupación de promedio global. Esta nueva capa calculará el promedio de cada uno de los filtros en lugar de simplemente convertir las matrices en un vector. En la Figura 48 se representa un ejemplo que muestra su funcionamiento.



*Figura 48. Funcionamiento de la capa de aplanamiento y capa de agrupación de promedio global.*

Aparentemente es un cambio muy pequeño, pero este cambio ha significado pasar de 30 etapas de entrenamiento para alcanzar el sobreajuste, a entrenar el modelo con 100 etapas y ver todavía potencial de mejora. La Figura 49 muestra el proceso de entrenamiento del modelo 4.1 durante 100 etapas. En ella se puede ver que sobre la etapa 30 se obtienen resultados muy buenos, que varían ligeramente desde entonces, hasta producir sobre la etapa 85 un pico de precisión y de pérdida, siendo en dicha etapa donde el modelo adquiere su mejor estado y es guardado con la parada temprana.



*Figura 49. Gráficas de pérdida y de precisión del modelo 4.1*

No obstante, la aparición del sobreajuste no está del todo clara, lo que indica que el modelo se podría ampliar todavía más. Además, se alcanza un nuevo techo de precisión, cerca del 87%, más de 10 puntos por encima de lo conseguido hasta ahora.

## 7.7. Modelos pre-entrenados: la arquitectura Xception.

Se ha hecho hasta el momento un modelo desde 0, viendo cómo actúan las diferentes partes que lo componen y analizando su rendimiento. Sin embargo, esta forma de desarrollar modelos no es la habitual. La primera razón es que se necesita de una gran capacidad de cómputo para entrenarlos. En este caso se necesitan entre 10 y 50 minutos para entrenar un modelo con el equipo disponible (ver Anexo 2. Manual de Usuario). La segunda razón es que, siguiendo el principio de los patrones de diseño<sup>19</sup>, la solución que se busca ya ha sido creada, perfeccionada y publicada por otras personas. La tercera razón es que para entrenar un modelo desde cero, se ha de disponer de un buen conjunto de datos con una gran cantidad de ejemplares y de una calidad suficiente. Por último, y en relación con la potencia de cómputo, para crear modelos de gran calidad haría falta hacerlos tan grandes que se requeriría utilizar granjas de servidores, en lugar de un ordenador casero.

Por todas estas razones mencionadas anteriormente, la mejor opción es usar los llamados modelos pre-entrenados. En la Figura 50 se muestra una tabla donde se proporciona una comparativa entre los distintos modelos pre-entrenados de los que dispone Keras.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8

Figura 50 Modelos disponibles de Keras (<https://keras.io/api/applications/>)

Debido a la gran similitud con el modelo que se ha estado creando desde cero hasta ahora, el modelo pre-entrenado seleccionado será el de Xception. Este modelo está formado casi íntegramente por componentes que se han explicado como las conexiones residuales, siendo la principal diferencia su profundidad que es mucho mayor. Para su uso, se verán las dos formas principales de adaptación al problema que se está tratando: la extracción de características y la modificación y afinado.

<sup>19</sup> <https://platzi.com/blog/patrones-de-diseno/>

### 7.7.1. Extracción de características

Esta técnica se basa en considerar al modelo pre-entrenado Xception como una caja negra, no se sabe qué capas contiene ni como están configuradas, pero tampoco es relevante, solamente se utiliza. El modelo en la fase de entrenamiento no aprenderá, sólo recibirá como entrada las imágenes que se le proporcione y su salida será la que se utilizará para configurar el modelo que se creará a continuación. El modelo que se entrenará estará compuesto por una capa densa de 256 perceptrones con activación *ReLU* y una capa de 58 perceptrones con activación *SoftMax* encargada de clasificar (ver Figura 51).

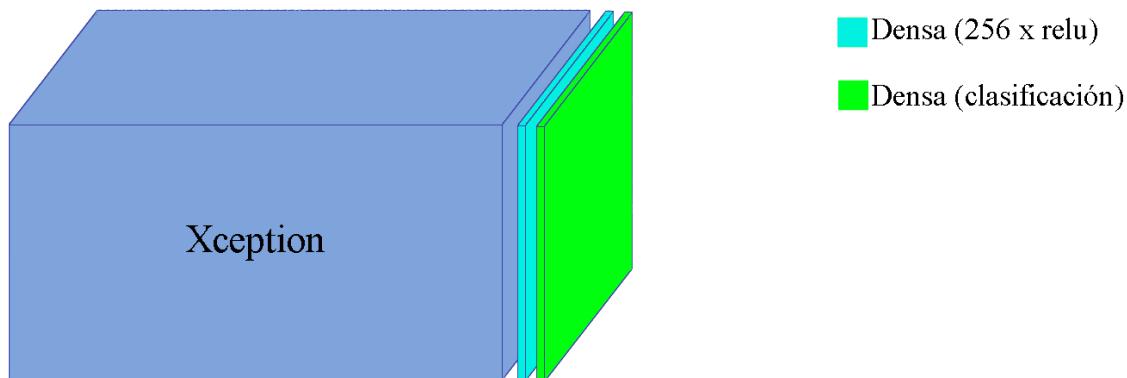


Figura 51. Extracción de características del modelo Xception.

La técnica de extracción de características recibe su nombre debido a que el modelo creado recibe como entrada las salidas de un modelo ya existente, siendo el modelo existente el que recibe las imágenes en su lugar. Como el modelo creado contiene únicamente dos capas modificables, la fase de entrenamiento será muy corta, dando buenos resultados desde el primer ciclo como se puede ver en la Figura 52.

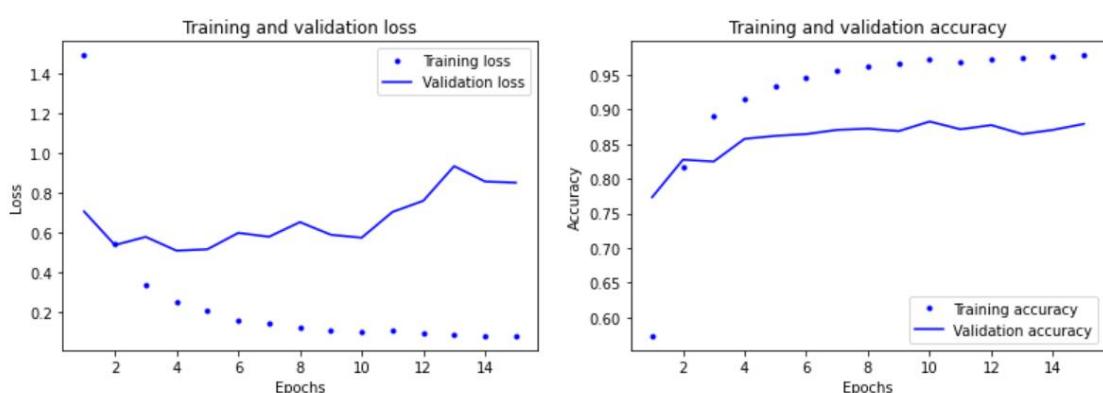


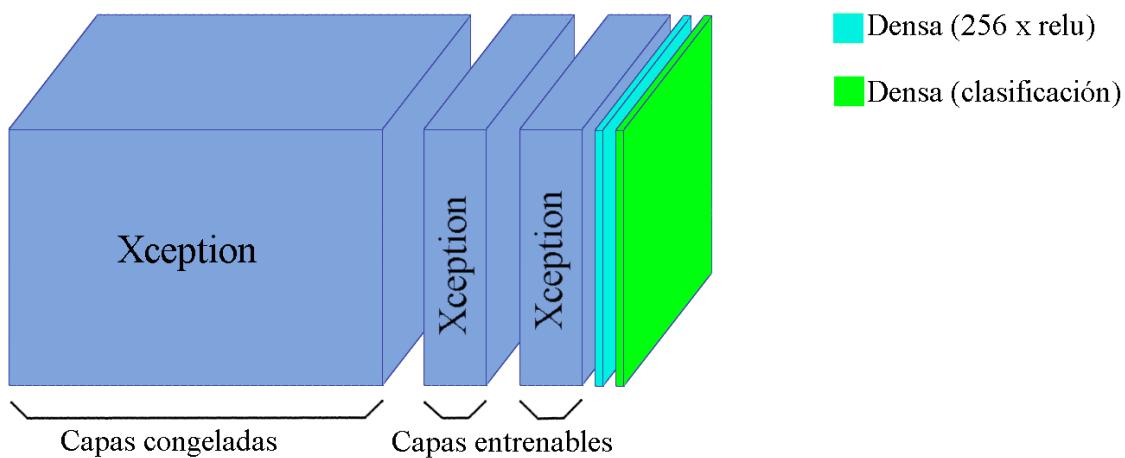
Figura 52. Gráficas de pérdida y de precisión de un Xception con extracción de características.

En los gráficos se puede observar que en la etapa 4 se empieza a producir el sobreajuste. El estado del modelo no obstante, es guardado con la parada temprana en esa etapa para seguir perfeccionándolo en el siguiente punto. Cabe mencionar que con esta técnica se ha

conseguido un 86% de precisión y una pérdida de 0,46, mejorando en pérdida el modelo que se había creado desde cero, pero no en precisión por un punto.

#### 7.7.2. Modificación y afinamiento

La técnica de evitar que el Xception sea modificado durante la etapa de entrenamiento es conocida como congelamiento del modelo. En esta sección lo que se hará es, en lugar de congelar el modelo pre-entrenado completamente, se dejarán como modificables algunas de las últimas capas del Xception. Además, la tasa de aprendizaje se reducirá, provocando que el modelo se ajuste más lentamente con intención de mejorar su rendimiento, sin alcanzar el sobreajuste demasiado rápido. El resultado de esta modificación queda ilustrado en la Figura 53.



*Figura 53. Modificación y afinamiento del Xception*

Por último, cabe mencionar que el Xception tiene capas de normalización de lotes y han sido también congeladas, es decir, no serán modificadas durante la fase de entrenamiento. Esta decisión se debe a que el proceso de normalización es completamente dependiente del conjunto de datos y, en este caso, la normalización se hizo por un conjunto de datos distinto.

Se ha conseguido una precisión del 90,4% con tan sólo un 0,301 de pérdida, aplicando esta técnica al modelo guardado en el punto anterior que tenía un 86% de precisión y un 0,46 de pérdida. A continuación, se ha seguido con el afinamiento reduciendo la cantidad de capas modificables a la mitad y la tasa de aprendizaje, sin conseguir una mejora significativa, 90,5% de precisión y 0,305 de pérdida. Con esta técnica se ha conseguido aumentar en 4 puntos la precisión con respecto a la versión anterior.

Para concluir con los experimentos, se han descongelado las capas de normalización de lotes presentes en las últimas capas que se habían congelado en el primer experimento de esta sección. El resultado es una precisión del 90% contra el 90,4% conseguido con las capas congeladas, evidenciando la desmejora del descongelamiento de dichas capas.

## 7.8. Comparación de modelos y resultados.

En esta sección se realiza una comparativa entre los distintos modelos creados desde 0 y el modelo pre-entrenado seleccionado. Cabe recordar que el modelo 3.3 es el modelo ampliado al que se realizan los cambios que generan los modelos comprendidos del 3.4 al 3.12. En este grupo hay que resaltar que, siguiendo el esquema de la Figura 37, la comparación entre los modelos no sigue el orden dictado directamente por el su número. Por lo tanto, no es justo decir que el modelo 3.6 por ejemplo, es una evolución del 3.5 ya que realmente es una evolución del modelo 3.3. El número indica únicamente el orden en el que fueron creados. Para el resto de los modelos, el orden de creación sí coincide con su nombre.

En la Figura 54 se muestra la primera comparativa de los modelos, respectiva a la precisión y pérdida. Se puede apreciar como a medida que se ha ido perfeccionando el modelo 0, primer modelo creado, la pérdida ha tenido una tendencia a bajar a la vez que la precisión subía. Estas tendencias no son constantes, lo que evidencia la intención del proyecto, que es la mejora a través de prueba y error. Esto se nota especialmente a partir del modelo 3, donde se buscaba crear un modelo más grande para generar ruido debido a la profundidad del modelo.

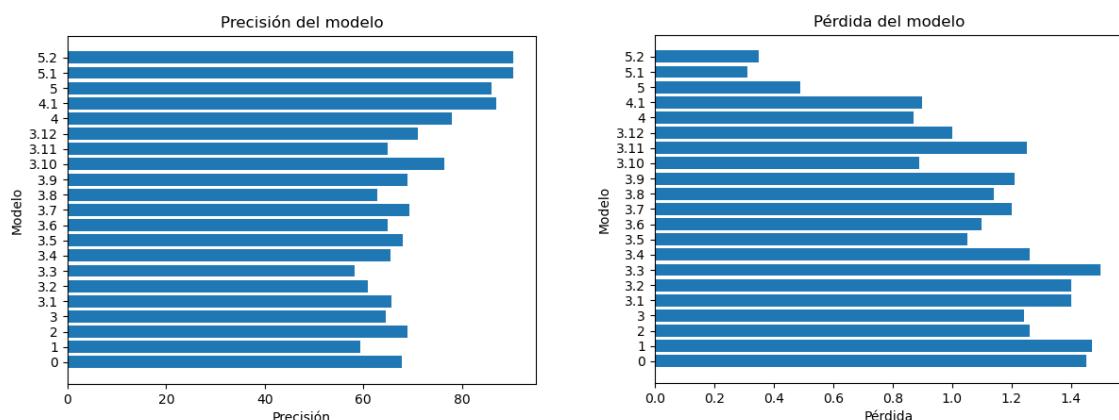


Figura 54. Gráficas comparativas de precisión y pérdida de los diferentes modelos.

Un dato muy interesante de la gráfica de la izquierda es que la precisión de los últimos cuatro modelos es muy similar. De hecho, el modelo 5 correspondiente a la primera versión del modelo pre-entrenado, no es a priori mejor que el modelo 4.1. Sin embargo el modelo 5 es solo una versión de prueba del Xception que no está ajustado del todo. Por otro lado, a pesar de que el modelo 5.1 es la versión ya ajustada y que sí supera al modelo 4.1, la mejora no es demasiado grande. Esto puede indicar un posible estancamiento de aprendizaje debido al conjunto de datos.

Para arrojar más luz se muestra la gráfica de la derecha. Esta gráfica indica que la diferencia entre los modelos pre-entrenados y el modelo 4.1 en realidad no es tan pequeña. A pesar de que el paso del modelo 4.1 al modelo 5 supone una pequeña penalización en precisión, también supone que la pérdida se reduzca a la mitad. Una de las principales razones de la menor pérdida es el tamaño del modelo, ya que cuanto más profundo es y más filtros tengan sus capas, mejor será su desempeño.

Esta cuestión se ve reflejada también en los modelos 3.10 y 3.12. En ellos la única diferencia es que el modelo 3.10 tiene exactamente 2 capas convolucionales más que el 3.12 y la pérdida en este último es notoriamente mayor.

En conclusión, a pesar de que el modelo pre-entrenado tiene los mejores resultados, la cantidad de capas que necesita para lograrlo también es mucho mayor. Concretamente el Xception tiene el doble de capas que el modelo 4.1. La Figura 55 muestra la relación tamaño-precisión que deja seg\xf3n este criterio en \xfaltimo lugar al Xception.

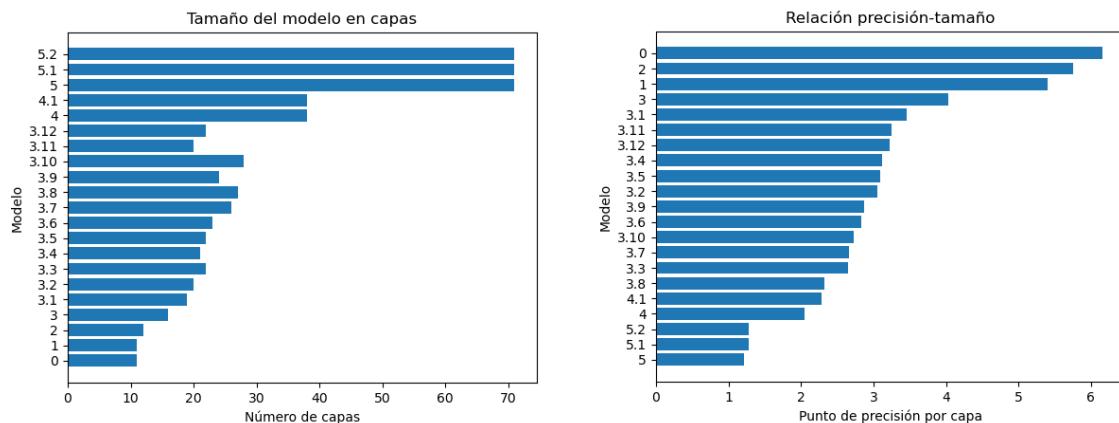


Figura 55. Gráficas comparativas del tamaño de los modelos y su relación con la precisión.

Si se tiene en cuenta únicamente este aspecto, resulta que las mejores opciones residen en los primeros modelos. La razón de ello es que un modelo muy pequeño es capaz de alcanzar fácilmente una precisión del 50%, pero a partir de este punto cada punto extra requiere aumentar el número de capas en gran cantidad. Un ejemplo de esta situación se encuentra en el modelo 5.1 que tiene el doble de capas que el modelo 4.1 y que sólo lo supera por 4 puntos en precisión.

El tamaño de capas es especialmente importante en dispositivos móviles debido a su limitada memoria. Sin embargo, un modelo con un 50% de precisión a la hora de clasificar imágenes entre 58 categorías, carece de utilidad, por lo que en ese caso sería interesante encontrar un equilibrio entre tamaño-precisión. En ese sentido el 3.12 es uno de los más equilibrados en cuanto a pérdida, precisión y tamaño. Cabe recordar una vez más que el modelo 3.12 es una versión con 2 capas convolucionales menos que el 3.10 y tienen, respectivamente, 72% y 76,5% de precisión, lo que acentúa una vez más la relación entre la mejora con la cantidad de capas.

Por último, hay que mencionar la comparativa de tiempo de entrenamiento de cada uno de los modelos. Este aspecto es uno de los más importantes a la hora de elegir un modelo u otro, ya que, como se puede ver en la Figura 56, elegir un modelo pre-entrenado o desplegar uno propio puede marcar la diferencia entre esperar unos segundos u horas.

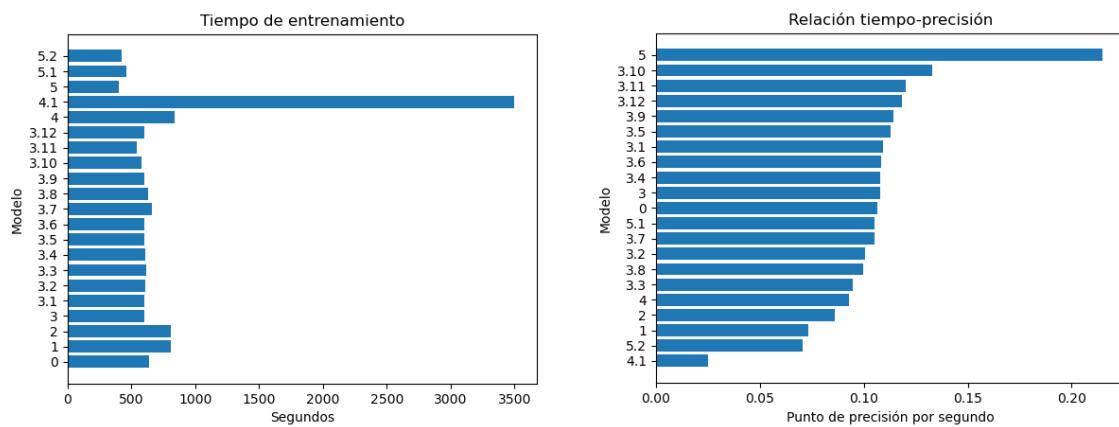


Figura 56. Tiempo de entrenamiento y su relación con la precisión ganada de los distintos modelos.

Si bien es cierto que entrenar el Xception por primera vez da muy buenos resultados en muy poco tiempo de entrenamiento, también es cierto que su perfeccionamiento le suma tiempo de entrenamiento en cada una de las iteraciones, como se puede ver reflejado en la gráfica de la derecha. En concreto, entrenarlo por primera vez supuso 6 minutos y medio, mientras que el modelo 4.1, el más parecido en rendimiento de los creados desde cero, tardó 58 minutos. Para generar el modelo 5.2 hicieron falta 2 iteraciones más, en total 18 minutos, menos de la mitad que el modelo 4.1. De esta forma, para tener el Xception del modelo 5.2 habría que invertir una cantidad de tiempo de entrenamiento, en teoría, mayor que algunos modelos con resultados también muy buenos, como lo es el 3.10 que requirió 6 minutos para su entrenamiento.

La realidad es que, si se tiene en cuenta el tiempo de desarrollo completo además del de entrenamiento, no es factible crear un modelo desde cero cuando se tienen modelos pre-entrenados. De hecho, para llegar al modelo 3.10 han hecho falta muchos ajustes y modelos previos, mientras que para el modelo 5.2. tan sólo haría falta entrenar el Xception y luego ajustarlo.

Como últimos comentarios acerca de los resultados obtenidos, la precisión de los modelos 4.1 al 5.2 es muy similar y no alcanza en ningún caso el 91%. La pérdida tampoco es tan baja como debería ser en los mejores modelos ya que, con un buen conjunto de datos de entrenamiento, se debería tener una precisión y una pérdida más cercanas al 100 y 0 respectivamente. Hay que recordar que en algunos casos el conjunto de datos contenía tan solo 7 ejemplares para un grupo, siendo ampliado hasta llegar a los 200 ejemplares con la técnica de ampliación de imágenes. Esta ampliación tan agresiva puede perjudicar mucho al modelo, por lo que queda pendiente entrenar los modelos con un conjunto de datos diferente y con diferentes técnicas de ampliación de imágenes.

## 8. Conclusiones.

La creación de un sistema de reconocimiento de enfermedades y plagas de plantas supone un reto cada día menor. Entre las soluciones analizadas, la más rápida y sencilla es usando una red neuronal existente y adaptarla al problema a resolver. Este método ha permitido crear un sistema que clasifica imágenes entre 58 categorías de enfermedades y plantas con una efectividad del 91%. Además, gracias a la existencia de herramientas relativamente nuevas como lo son Keras y TensorFlow, desplegadas hace apenas 7 años, se ha podido desarrollar y evolucionar un sistema de *deep learning* complejo con relativa sencillez, permitiendo un seguimiento adecuado del proyecto.

Gracias a estas herramientas no se ha tenido que implementar en detalle algunos conceptos como los perceptrones, sino directamente capas de la red compuestas por ellos, permitiendo invertir el tiempo en un mejor estudio de las tecnologías. Por otro lado, gracias a Google Colab se podrían ejecutar todos los pasos hechos en este proyecto con el único requisito de disponer de un computador con conexión a internet.

El principal inconveniente de la implementación de sistemas de *deep learning* relativamente complejos es el tiempo. Cada uno de los experimentos realizados han tardado entre 10 y 50 minutos y la mayoría de las veces se han hecho varias ejecuciones, ya sea por un error de configuración o para contrastar resultados. Cabe mencionar el tiempo de familiarización con las herramientas descritas ya que, aunque parezca simple por utilizarse pocas líneas, cada una de ellas contiene mucha información.

Gracias al desarrollo de este proyecto, se ha podido plasmar la teoría de la IA en el estado actual de la tecnología, así como su materialización en la práctica. Se ha podido demostrar el impacto de las modificaciones más importantes realizable a un sistema desde cero con éxito y compararlo con un sistema mucho más avanzado. El sistema Xception es, como se vio, un sistema enorme. Su desarrollo desde cero no es factible debido a la potencia de cómputo de los ordenadores caseros. No obstante, sí que he considerado interesante crear un sistema parecido a pequeña escala ya que su conocimiento en profundidad es útil para la creación de modelos personalizados. En el día a día, lo habitual sería la utilización de sistemas como el Xception que se entrenarían en muy pocos minutos o segundos, dependiendo la configuración elegida.

Por último, como se comentó en el análisis de resultados, un 91% de precisión no es un resultado demasiado satisfactorio. Queda pendiente, por tanto, un análisis más exhaustivo del dataset y de lo cómo son procesadas las imágenes. Para este cometido, sería también interesante la creación de matrices de confusión.

## Bibliografía.

- A. Halevy, P. Norvig, & F. Pereira. (2009). The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24(2), 8–12. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4804817&isnumber=4804812>
- Alaimo, M. (2018, October 16). *Waterfall. La historia detrás del error. / by Kleer / Medium.* <https://medium.com/@kleer.la/waterfall-la-historia-detr%C3%A1s-del-error-39f589a12115>
- B. Loiseau, J.-C. (2019). *Rosenblatt's perceptron, the first modern neural network | by Jean-Christophe B. Loiseau / Towards Data Science.* <https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a>
- Brownlee Jason. (2017, August 9). *How to Use Metrics for Deep Learning with Keras in Python.* <https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/>
- Chollet, F. (2017). *Deep Learning with Python.*
- Espinoza-Meza, A. (2013). *MANUAL PARA ELEGIR UNA METODOLOGÍA DE DESARROLLO DE SOFTWARE DENTRO DE UN PROYECTO INFORMÁTICO.* [https://pirhua.udep.edu.pe/bitstream/handle/11042/2747/ING\\_521.pdf?sequ](https://pirhua.udep.edu.pe/bitstream/handle/11042/2747/ING_521.pdf?sequ)
- F. Luger, G., & A. Stubblefield, W. (1998). *Artificial Intelligence.* . Addison-Wesley.
- Guan, L., Zhang, J., & Geng, C. (2021). Diagnosis of Fruit Tree Diseases and Pests Based on Agricultural Knowledge Graph. *Journal of Physics: Conference Series*, 1865(4). <https://doi.org/10.1088/1742-6596/1865/4/042052>
- Haugeland, J. (2001). *La inteligencia artificial. Siglo XXI.*
- Himanshu Lawaniya. (2020). *Computer Vision.* [https://www.researchgate.net/publication/341313012\\_Computer\\_Vision](https://www.researchgate.net/publication/341313012_Computer_Vision)
- Ignacio Bagnato, J. (2018). *Breve Historia de las Redes Neuronales Artificiales / Aprende Machine Learning.* <https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/>
- Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* <http://arxiv.org/abs/1502.03167>
- Javier Garzas. (2014, May 19). *El verdadero origen del ciclo en cascada, sobre caídas de brujas y errores históricos - Javier Garzas.* <https://www.javiergarzas.com/2014/05/el-verdadero-origen-del-ciclo-de-cascada.html>
- Ligdi González. (2021). *¿Qué es el Perceptrón? Perceptrón Simple y Multicapa - Aprende IA.* <https://aprendeia.com/que-es-el-perceptron-simple-y-multicapa/>
- Martínez, J. (2019). *Aprendizaje Veloz con Normalización por Lotes - DataSmarts Español.* <https://datasmarts.net/es/aprendizaje-veloz-con-normalizacion-por-lotes/>
- Mishra, A. (2018). *Metrics to Evaluate your Machine Learning Algorithm / by Aditya Mishra / Towards Data Science.* <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>
- Oleaga Jon. (2019, April 25). *Cómo usa Facebook la inteligencia artificial (y para qué).* [https://www.abc.es/tecnologia/redes/abci-como-facebook-inteligencia-artificial-y-para-201904250140\\_noticia.html](https://www.abc.es/tecnologia/redes/abci-como-facebook-inteligencia-artificial-y-para-201904250140_noticia.html)
- Orús Abigail. (2022, April 27). *Youtube: usuarios a nivel mundial 2012-2021 / Statista.* <https://es.statista.com/previsiones/1289041/usuarios-de-youtube-en-todo-el-mundo>

- Pablo Espeso. (2014, January 24). *Deep Blue, el ordenador con una sola misión: ganar al humano*. <https://www.xataka.com/otros/deep-blue-el-ordenador-con-una-sola-mision-ganar-al-humano>
- Riley, M., Williamson, M., & Maloy, O. (2002). Plant Disease Diagnosis. *Plant Health Instructor*. <https://doi.org/10.1094/PHI-I-2002-1021-01>
- Saleem, M. H., Potgieter, J., & Arif, K. M. (2020). Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers. *Plants* 2020, Vol. 9, Page 1319, 9(10), 1319. <https://doi.org/10.3390/PLANTS9101319>
- Toda, Y., & Okura, F. (2019). How convolutional neural networks diagnose plant disease. *Plant Phenomics*, 2019. <https://doi.org/10.34133/2019/9237136>
- Xiaoxue, L., Xuesong, B., Longhe, W., Bingyuan, R., Shuhan, L., & Lin, L. (2019). Review and Trend Analysis of Knowledge Graphs for Crop Pest and Diseases. In *IEEE Access* (Vol. 7, pp. 62251–62264). Institute of Electrical and Electronics Engineers Inc. <https://doi.org/10.1109/ACCESS.2019.2915987>
- Yıldırım, S. (2020). *Gradient Boosted Decision Trees-Explained | by Soner Yıldırım | Towards Data Science*. <https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af>

## Anexo 1. Descripción de la carpeta del TFG

El TFG se puede descargar desde OneDrive a través del siguiente enlace: [https://urjc-my.sharepoint.com/:f/g/personal/e\\_sierram\\_2018\\_alumnos\\_urjc\\_es/EmgAekSpktNpWSY8ZHF1yYBLbRxrYTe8K33AzDkDBFcFQ?e=FGBVC4](https://urjc-my.sharepoint.com/:f/g/personal/e_sierram_2018_alumnos_urjc_es/EmgAekSpktNpWSY8ZHF1yYBLbRxrYTe8K33AzDkDBFcFQ?e=FGBVC4)

En la carpeta que contiene el TFG se encuentran las siguientes tres subcarpetas:

- Código: esta carpeta contiene un fichero requirements.txt con las librerías de Python necesarias y seis ficheros con la extensión. *ipynb*. En estos ficheros se encuentra todo el código de los sistemas que se han descrito en este documento. Además, contienen los procesos de entrenamiento de los sistemas, las gráficas resultantes de rendimiento en cada etapa de entrenamiento y el dataset con todas las imágenes.
- Memoria: este directorio contiene el documento PDF correspondiente a la memoria del TFG.
- Presentación: esta carpeta contiene el fichero *presentación.pptx* correspondiente a las diapositivas utilizadas para la presentación del TFG

La carpeta .zip tiene la estructura mostrada en la siguiente imagen (ver Figura 1):

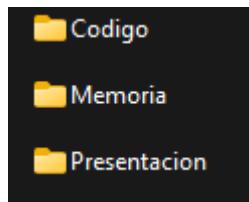


Figura 1. Composición de la carpeta contenedora.

## Anexo 2. Manual de Usuario

En esta sección del anexo se proporciona al usuario un manual de uso. En primer lugar, el código desarrollado en todos los sistemas presentados en este TFG se encuentra en la carpeta “código”. También se ha utilizado un repositorio GitHub para publicar el código. La URL del repositorio es la siguiente: <https://github.com/eduardosoy/TFG> .

### 2.1. Requisitos hardware

Para la ejecución del código, se ha de contar con un equipo con una potencia de cómputo elevada para entrenar los modelos. En mi caso, el entrenamiento de cada modelo ha tardado cerca de 30 minutos con un equipo de las siguientes características:

- Tarjeta gráfica: RTX 3060.
- Procesador: Ryzen 5 2600.
- Memoria RAM: 16GB.

### 2.2. Requisitos software

El código se encuentra en seis ficheros con extensión. *ipynb*. Esta extensión indica que no se trata de ficheros de Python, sino de código en Jupyter. Este formato nos permite generar documentos enriquecidos con bloques de código fuente, que se pueden alternar con texto.

Para la ejecución del código, se tiene que instalar Jupyter y las librerías especificadas en el documento *requirements.txt* con sus versiones correspondientes. Para la instalación de paquetes, se recomienda un gestor de paquetes como anaconda, aunque este paso es opcional.

### 2.3. El dataset

Para poder entrenar los modelos creados, se ha de descargar en la carpeta “código” el dataset utilizado que contiene las imágenes de ejemplo. El dataset se puede descargar desde aquí <https://www.kaggle.com/datasets/sadmansakibmahi/plant-disease-expert>.

Los ficheros “tfg0.ipynb” y “tfg1\_aumento\_de\_datos.ipynb” se diferencian en cómo configuran el dataset. El primero sólo reorganiza los datos y el segundo amplía las imágenes, por lo que se recomienda este último. En cualquier caso, se ha de modificar una variable para que identifique la ruta donde se encuentra la carpeta del dataset, quedando esa línea de código similar a lo siguiente:

```
1. original_dir = pathlib.Path("DatasetOriginal/Image Data Base")
```

En el caso de ejemplo, el directorio es “Image Data Base” y contiene los 58 subdirectorios del dataset.

La configuración del dataset termina en la línea con el título “Modelo creado desde cero”, a partir de entonces se pueden ejecutar todos los experimentos.

## 2.4. Guía de uso o ejecución de los modelos

Esta sección pretende ilustrar el significado del código con intención de que se pueda modificar y probar. Además, muestra los pasos de creación y ejecución de cualquier modelo de los que se han creado. Todo ello en cuatro fases: inicialización, creación del modelo, entrenamiento y análisis.

### 2.4.1. Inicialización

Este paso sirve para inicializar las variables y parámetros que se usarán en cada uno de los modelos que se quiera ejecutar, por lo que se encuentra en las primeras líneas de todos los ficheros, al igual que las importaciones de las librerías. Para ello es necesario el siguiente código (ver Figura 1. Importación de librerías. Figura 1).

```
from tensorflow import keras
from tensorflow.keras import layers
import os, shutil, pathlib
from tensorflow.keras.utils import image_dataset_from_directory
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import load_model
```

*Figura 1. Importación de librerías.*

Para la creación de las variables globales del sistema se ha de ejecutar el código mostrado a continuación (ver Figura 2).

```
new_base_dir = pathlib.Path("ExpandedDataset_organized/")
content = os.listdir(original_dir)
BATCH_SIZE = 64
size_x = 256
size_y = 256
```

*Figura 2. Variables globales.*

Estas variables describen dónde se encuentran las imágenes del dataset, el tamaño del lote y el tamaño al que se escalarán las imágenes. Con los parámetros descritos se crean los subconjuntos de datos con los que se entrenarán los modelos, el conjunto de validación, y el conjunto de pruebas (ver Figura 3).

```

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(size_x, size_y),
    batch_size=BATCH_SIZE,
    label_mode='categorical')
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(size_x, size_y),
    batch_size=BATCH_SIZE,
    label_mode='categorical')
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(size_x, size_y),
    batch_size=BATCH_SIZE,
    label_mode='categorical')

```

*Figura 3. Subconjuntos de datos.*

Gracias a Jupyter, no será necesario volver a ejecutar las líneas de código anterior para cada experimento.

#### 2.4.2. Creación un modelo

Un modelo se crea con un código similar al siguiente, que generalmente se estructura en tres partes: inputs (entradas), capas ocultas y output (salidas), tal y como se ve en la Figura 4.

```

inputs = keras.Input(shape=(size_x, size_y, 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.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(58, activation="softmax")(x)
model2 = keras.Model(inputs=inputs, outputs=outputs)

model2.summary()

```

*Figura 4. Código del modelo 2.*

En la primera línea se indica el tamaño de las imágenes que se le proporcionará al modelo, que será el que se indicó en las variables. A continuación, hay una serie de líneas que empiezan por “x”, son las capas del modelo. Como se puede apreciar en este caso predominan las capas convolucionales (Conv2D) y las de agrupamiento (MaxPooling2d). Entre los paréntesis de las capas convolucionales se indica el número de filtros, el tamaño del núcleo convolucional y el modelo de activación, que en este caso es *ReLU*. Después, se indica la salida (output), que será proporcionada por la capa densa con activación *SoftMax*. Por último, se crea el modelo y se asigna a la variable *model2*.

La última línea, “model2.summary()” es opcional y sirve para mostrar la arquitectura del modelo creado con una estructura similar a la que se muestra a continuación (ver Figura 5). Nótese como en la columna “output shape” se muestra como la imagen se reduce con cada una de las capas del modelo, especialmente en las capas de agrupamiento.

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[ (None, 256, 256, 3) ]	0
rescaling_7 (Rescaling)	(None, 256, 256, 3)	0
conv2d_35 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_28 (MaxPooling)	(None, 127, 127, 32)	0
conv2d_36 (Conv2D)	(None, 125, 125, 64)	18496
max_pooling2d_29 (MaxPooling)	(None, 62, 62, 64)	0

Figura 5. Arquitectura del modelo 2.

#### 2.4.3. Entrenamiento del modelo

Para esta fase en todos los modelos se ha ejecutado un código muy parecido al que se muestra a continuación (ver Figura 6), estructurado en tres bloques. El bloque de “callbacks” indica dónde se guarda el modelo cuando se ejecuta la parada temprana y es opcional. A continuación, para que el modelo pueda ser entrenado, se ha de compilar. En este bloque se indica las medidas de pérdida, de precisión y el optimizador. Por último, el modelo se entrena y se le asigna el resultado de entrenamiento a la variable *history*. Para esta parte es necesario indicar el dataset de entrenamiento, el número de épocas, el dataset de validación y la asignación de callbacks, aunque esta última se puede omitir si no se quiere realizar la parada temprana.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="model2/best_version",
        save_best_only=True,
        monitor="val_loss")
]

model2.compile(loss="categorical_crossentropy",
    optimizer="rmsprop",
    metrics=["accuracy"])

history = model2.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Figura 6. Entrenamiento del modelo.

La ejecución de este código proporciona la siguiente salida (ver Figura 7).

```
Epoch 1/30
145/145 [=====] - 28s 178ms/step - loss: 4.1318 - accuracy: 0.0749 - val_loss: 2.8555 - val_accuracy: 0.2112
INFO:tensorflow:Assets written to: model2\best_version\assets
Epoch 2/30
145/145 [=====] - 26s 169ms/step - loss: 2.6199 - accuracy: 0.2874 - val_loss: 2.0430 - val_accuracy: 0.4164
INFO:tensorflow:Assets written to: model2\best_version\assets
Epoch 3/30
145/145 [=====] - 26s 169ms/step - loss: 1.7571 - accuracy: 0.4872 - val_loss: 2.3057 - val_accuracy: 0.3776
Epoch 4/30
145/145 [=====] - 25s 167ms/step - loss: 1.2638 - accuracy: 0.6136 - val_loss: 1.5679 - val_accuracy: 0.5405
INFO:tensorflow:Assets written to: model2\best_version\assets
```

*Figura 7. Salida del entrenamiento del modelo.*

En primer lugar, cada etapa de entrenamiento se muestra como Epoch. Después se muestra el número de lotes con una barra de carga, seguido del tiempo que se ha tardado en ejecutar esa etapa. En este caso ha tardado aproximadamente 26 segundos por cada etapa durante 30 etapas, en total 13 minutos. Por último, en cada etapa se muestra la pérdida y la precisión de clasificación, tanto la de entrenamiento como la de validación, que se analizará mejor en la fase siguiente. Cuando la pérdida es la menor, se ejecuta la parada temprana y se guarda el modelo según muestra las líneas que empiezan por “INFO”.

#### 2.4.4. Análisis de resultados

Con la parada temprana el modelo es guardado automáticamente, pero para guardar el modelo que se ha entrenado durante todas las etapas y su resultado de entrenamiento se han de ejecutar la siguientes líneas, respectivamente (ver Figura 8)

```
model2.save('model2/model')
np.save('model2/history.npy',history.history)
```

*Figura 8. Código de guardado del modelo y de resultados.*

Para cargar el modelo y el historial de entrenamiento se puede ejecutar lo siguiente (ver Figura 9),

```
my_model2 = load_model('model2/model')
my_history1=np.load('model1/history.npy',allow_pickle='TRUE').item()
```

*Figura 9. Código de carga del modelo y de resultados.*

El guardado manual del modelo es interesante para, por ejemplo, comprobar si el modelo guardado con la parada temprana es mejor que el entrenado por todas las etapas, como se muestra a continuación (ver Figura 10).

```
model = keras.models.load_model('model2/model')
results= model.evaluate(test_dataset)
results
19/19 [=====] - 3s 87ms/step - loss: 11.4377 - accuracy: 0.4914
[11.437688827514648, 0.4913793206214905]

model = keras.models.load_model('model2/best_version')
results= model.evaluate(test_dataset)
results
19/19 [=====] - 3s 77ms/step - loss: 1.2623 - accuracy: 0.6905
[1.2622944116592407, 0.690517246723175]
```

*Figura 10. Evaluación del modelo.*

Como se puede apreciar, el modelo con parada temprana tiene una precisión del 69% y una pérdida de 1.26 en este caso.

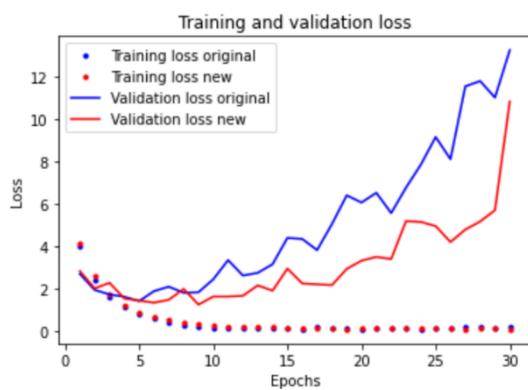
Para mostrar las gráficas correspondientes a los datos de precisión y pérdida durante el entrenamiento, se puede ejecutar un código como el de la Figura 11, gracias a la librería *pyplot*. Nótese como se asignan a las variables de las gráficas las variables de los modelos cargados anteriormente.

```
loss_1 = my_history1["loss"]
val_loss_1 = my_history1["val_loss"]
loss_2 = my_history2["loss"]
val_loss_2 = my_history2["val_loss"]
epochs = range(1, len(loss_1) + 1)

plt.plot(epochs, loss_1, "b.", label="Training loss original")
plt.plot(epochs, loss_2, "r.", label="Training loss new")
plt.plot(epochs, val_loss_1, "b", label="Validation loss original")
plt.plot(epochs, val_loss_2, "r", label="Validation loss new")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

*Figura 11. Código para mostrar las gráficas.*

Este código compara el historial de dos modelos que se habían guardado con anterioridad, lo que proporciona la gráfica de la Figura 12.



*Figura 12. Gráfica de pérdida del modelo durante el entrenamiento.*