Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Procesamiento de imágenes con técnicas de aprendizaje profundo para la detección de hojas en plantas

Autor: Fernando Garrucho Fernández

Tutor: Rubén Martín Clemente

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Ingeniería de Telecomunicaciones

Procesamiento de imágenes con técnicas de aprendizaje profundo para la detección de hojas en plantas

Autor:
Fernando Garrucho Fernández

Tutor:
Rubén Martín Clemente
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Grado: Procesamiento de imágenes con técnicas de aprendizaje profundo para la detección de hojas en plantas

Autor: Fernando Garrucho Fernández

Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

Agradecimientos

Estas líneas son en agradecimiento a todas esas personas que son importantes para mi y que aprecio y respeto, y que sin ellas no hubiese sido posible finalizar esta etapa. A mis padres quienes me han dado todo (y les debo todo) y me han inculcado desde pequeño que hay que trabajar duro para conseguir lo que te propongas. A mi hermano quien es una de las personas más importantes en mi día a día y que seremos compañeros y amigos para toda la vida. A mis abuelas, a mis tios y tias, mis primos y prima, y toda la familia cercana que es importante desde que tengo uso de razón y a todos aquellos que ya no están pero que nunca los olvidaré. A mis amigos los cuáles me sirven de apoyo continuo hasta casi que algunos son considerados familia y que también aportan alegría para afrontar cada día. A mis compañeros de clase y a toda la gente que he conocido por la universidad a través de programas como Erasmus o SICUE los cuales me han hecho mejor persona y he aprendido mucho de ellos. A aquellos profesores que merece la pena escucharlos y que demuestran su vocación por la profesión los cuales han hecho que me apasione esta ingeniería. Y por último, y no por ello menos importante, darme las gracias a mi, gracias por creer en mi y no darme por vencido, por trabajar sin descanso cuando fue necesario y por luchar por todos los objetivos que me propongo sin decaer, porque como aficionado al deporte me acuerdo de las palabras que pronunció el entrenador de tenis Toni Nadal, antiguo entrenador y tío de Rafa Nadal, las cuales sirvieron a Rafa para llegar a ser el mejor tenista de la historia y me sirven a mi para afrontar nuevos retos: “Es fácil poner ilusión en una final de Grand Slam, lo difícil es poner esa ilusión en cada partido, en cada entrenamiento y en cada bola que se juega para ser uno de los pocos privilegiados que llegan hasta allí”.

Fernando Garrucho Fernández

Sevilla, 2022

Resumen

La inteligencia Artificial y más concretamente el Deep Learning o aprendizaje profundo está siendo de gran relevancia hoy en día ya que se implementa en dispositivos que todos usamos diariamente. En este trabajo se conocerá un poco acerca de esta tecnología, viendo su historia, enteniendo sus elementos más básicos, adentrándose en su estructura y diseñando una red neuronal completa, en este caso se diseñará una red neuronal convolucional en Python para clasificación de imágenes más concretamente imágenes de plantas que se clasificarán por su número de hojas y se propondrán varias técnicas de preprocesado de imagen para mejorar el rendimiento de la red. Por último, se compararán los resultados con otro algoritmo que nada tiene que ver con inteligencia artificial y que también sirve para reconocimiento de imágenes.

Este estudio no tiene un fin comercial, sino familiarizarse con esta herramienta tan potente que es el Deep learning, aprender sobre el tratamiento de las imágenes y conocer técnicas para mejorar el aprendizaje de estas redes.

Abstract

Artificial intelligence and more specifically Deep Learning is being of great relevance nowadays as it is implemented in devices that we all use daily. In this work we will learn a little about this technology, seeing its history, understanding its most basic elements, getting into its structure, and designing a complete neural network, in this case we will design a convolutional neural network in Python for image classification, more specifically images of plants that will be classified by their number of leaves and we will propose several image preprocessing techniques to improve the performance of the network. Finally, the results will be compared with another algorithm that has nothing to do with artificial intelligence and is also used for image recognition.

This study does not have a commercial purpose, but rather to become familiar with this powerful tool that is Deep learning, to learn about image processing and to learn techniques to improve learning of this networks.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

LISTA DE ACRÓNIMOS

IA	Inteligencia Artificial
DL	Deep Learning
ML	Machine Learning
CNN	Convolutional Neural Networks
BGR	Blue Green Red
GPU	Graphics Processing Unit
CPU	Central Processing Unit
BBDD	Base de Datos.
HSV	Hue, Saturation, Value

1 INTRODUCCIÓN

"La inteligencia artificial sería la versión definitiva de Google. El motor de búsqueda definitivo que entendería todo en la web. Entendería exactamente lo que quieres, y te daría lo correcto. No estamos cerca de hacer eso ahora. Sin embargo, podemos acercarnos cada vez más a eso, y eso es básicamente en lo que trabajamos"- Larry Page, fundador de Google -

En los siguientes apartados se explican los motivos por los que se decidió realizar este proyecto basado en el campo del 'Deep Learning' o aprendizaje profundo, más concretamente en las redes neuronales y dentro de éstas las convolucionales que son las que mejores prestaciones ofrecen en el tratamiento de imágenes como se verá posteriormente. También se detallan los objetivos de este estudio, así como la estructura u organización de esta memoria.

1.1 Motivación

Cada día se ven numerosas implementaciones de la IA en todos los ámbitos. En la industria del automóvil la conducción autónoma es una realidad, también el reconocimiento de voz se ha implementado de manera exitosa en muchos de los dispositivos usados cotidianamente pudiendo interactuar con ellos con la voz, otro sería el reconocimiento facial o la detección de huellas dactilares los cuales han mejorado tanto que nuestras contraseñas quedan protegidas por estos sistemas, incluso el reconocimiento de imágenes en nuestros móviles puede hacer que detecte las personas que aparecen en cada foto de la galería.

Todas las mejoras enumeradas usan el aprendizaje profundo, es por ello por lo que la motivación de este estudio es familiarizarse con este campo y más concretamente en el campo de visión por ordenador, más conocido como reconocimiento de imágenes por ordenador, el cual se especializa en el reconocimiento de imágenes. Se persigue entender el funcionamiento de estas redes y compararlas con otros métodos alternativos de clasificación para ver si son estrictamente necesarias.

1.2 Objetivos

El principal objetivo de este trabajo es conocer el mundo del Machine Learning y del Deep Learning y aprender sobre preprocesamiento de imágenes en Python. Para ello se elegirá una base de datos sobre la que trabajar, esta base de datos contiene imágenes sobre plantas de diferentes especies clasificadas por su número de hojas. Esas imágenes se preprocesarán para así ayudar a la red neuronal a clasificarlas, luego se diseñará la red neuronal y se optimizará para adecuarla al problema de clasificación presentado. Se harán las modificaciones pertinentes para mejorar lo máximo posible la precisión de la red aprendiendo de las curvas de aprendizaje e interpretando la matriz de confusión, por último, se presentará un algoritmo en Matlab que resuelve problemas de clasificación para compararlo con la red neuronal y ver si esta es realmente necesaria para este problema.

1.3 Estructura

La organización que se ha seguido para el proyecto ha sido la siguiente:

- **Capítulo 1. Introducción:** En este se presenta la motivación de este proyecto, así como el objetivo que

se persigue, se enumera también su estructura.

- **Capítulo 2. Aprendizaje Profundo:** Introducción al Machine Learning y a la Inteligencia artificial, presentación del Deep Learning y profundización en cómo funciona una red neuronal y cuáles son sus partes.
- **Capítulo 3. Redes Neuronales Convolucionales:** Presentación del tipo de redes neuronales conocidas como convolucionales, estructura y partes que la conforman. Descripción de sus principales problemas y métodos usados para solucionarlos, por último, se interpretan sus salidas.
- **Capítulo 4. Preprocesado de los datos:** Herramienta usada para el preprocesamiento de los datos que ayudará a la red a tener mejor rendimiento.
- **Capítulo 5. Implementación y experimentos:** Entorno usado, diseño de la red neuronal, problemas encontrados y soluciones a estos en forma de experimentos, resultados finales. Por último, algoritmo en Matlab para compararlo con el rendimiento de la red neuronal.
- **Capítulo 6. Conclusiones y líneas futuras:** Interpretación de los resultados obtenidos y planteamiento de posibles mejoras futuras

2 APRENDIZAJE PROFUNDO

En el siguiente apartado se detalla todo lo que se necesita conocer sobre aprendizaje profundo para la realización de este proyecto. Se comenzará con un repaso general sobre inteligencia artificial hasta indagar en las redes neuronales y sus diferentes partes y etapas, comprender las curvas de aprendizaje y la posterior mejora de modelos.

2.1. Introducción

2.1.1 Inteligencia Artificial

La Inteligencia Artificial o IA (AI en inglés) es la simulación de procesos de inteligencia humana por parte de sistemas informáticos. Estos procesos incluyen el aprendizaje (la adquisición de información y reglas para el uso de la información), el razonamiento (usando las reglas para llegar a conclusiones aproximadas o definitivas) y la autocorrección. Las aplicaciones particulares de la IA incluyen sistemas expertos, reconocimiento de voz y visión artificial [1].

Se considera que el punto de partida de esta tecnología es el año 1950, precisamente, cuando Turing publica un artículo con el título "*Computing machinery and intelligence*" en la revista Mind, donde se hacía la pregunta: "¿pueden las máquinas pensar?" y proponía un método para determinar si una máquina puede pensar. Los fundamentos teóricos de la IA se encuentran en el experimento que propone en dicho artículo y que pasó a denominarse Test de Turing, en el cual se plantea si una máquina pudiera pasar por un humano en una charla a ciegas. Esta prueba sigue estando vigente en la actualidad y es motivo de estudios e investigaciones continuas. Aunque no fue hasta el año 1956, cuando los padres de la inteligencia artificial moderna, John McCarty, Marvin Misky y Claude Shannon acuñaron formalmente el término IA durante la conferencia de Darmouth. La inteligencia artificial tuvo un período de sequía propiciado por la falta de interés y de optimismo en el tema, este período es conocido como "AI Winter". Hasta el año 1997 cuando el supercomputador Deep Blue de IBM ganó al campeón mundial de ajedrez Gari Kasparov. El año 1997 es considerado por algunos historiadores como el punto de inflexión donde comenzó a oírse la inteligencia artificial fuera de los ámbitos académicos y de investigación. Sin embargo, es en la segunda década del siglo XXI cuando comienzan a aparecer los acontecimientos de impacto para llevar la IA al punto de despegue donde se encuentra en este momento [2] [3].

En febrero de 2011, el supercomputador Watson de IBM gana en el concurso televisivo de Estados Unidos "*Jeopardy*", en el que se realizan preguntas y cuestiones diferentes de todo tipo, cultura y conocimiento, a los dos mejores concursantes del programa, Brad Ruttler y Ken Jennings. Watson es una computadora capaz de aprender a medida que trabaja y acumula información y que puede interactuar con el lenguaje humano en un lenguaje natural. Watson va aprendiendo con las interacciones con el usuario [2] [3].

Otro hito importante fue la presentación de Apple del asistente virtual Siri integrado en el teléfono móvil iPhone 4S en el año 2011 y donde comenzaron las primeras experiencias de aprendizaje automático y los primeros indicios de aprendizaje profundo. Hasta el día de hoy, donde se ha incrementado el uso del IoT y los dispositivos controlados por voz, no sólo por la mayor entrada de los asistentes de voz de Google, Amazon o Apple en el ámbito doméstico sino por el crecimiento esperado de su implantación en oficinas y áreas de trabajo [2] [3].

Aunque al pensar en Inteligencia Artificial se nos viene a la mente robots reemplazando humanos el objetivo real no es ése, sino mejorar la vida de las personas haciendo tareas repetitivas y manejando grandes cantidades de datos mucho mejor y más eficientemente.

La Inteligencia artificial está compuesta de numerosas ramas, en este trabajo nos centraremos en el Machine Learning y más concretamente en el Deep Learning.

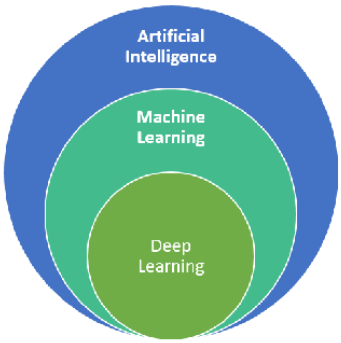


Figura 2-1. Jerarquía de la IA, ML y DL. Fuente: Nadia Berchane (M2 IESCI, 2018)

2.1.2 Aprendizaje Máquina

"En definitiva, el aprendizaje máquina (o machine learning) es un maestro del reconocimiento de patrones, y es capaz de convertir una muestra de datos en un programa informático capaz de extraer inferencias de nuevos conjuntos de datos para los que no ha sido entrenado previamente", explica José Luis Espinoza, científico de datos de BBVA México [4].

"Machine Learning" o Aprendizaje Máquina es una rama dentro de la IA y ésta tomará un conjunto de datos de entrada y emitirá una respuesta basada en los datos dados que determine como correcta. Lo hace mediante algoritmos cuyo rendimiento mejora a medida que se exponen a más datos a lo largo del tiempo. En cuanto a sus algoritmos, estos pueden ser muy potentes, pueden ser entrenados para reconocer patrones en los datos que los humanos no pueden y pueden hacer predicciones futuras basadas en lo que ya saben del pasado. Varias industrias ya lo han implementado en sus operaciones, incluyendo la analítica de negocios, la automatización del marketing, el servicio al cliente, la prevención del fraude y la seguridad [5].

Dada la madurez del área de investigación en Machine Learning, existen muchos enfoques bien establecidos para el aprendizaje automático por parte de máquinas. Cada uno de ellos utiliza una estructura algorítmica diferente para optimizar las predicciones basadas en los datos recibidos. Machine Learning es un amplio campo con una compleja taxonomía de algoritmos que se agrupan, en general, en tres grandes categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [5].

2.1.2.1 Aprendizaje Supervisado

Se refiere a aprendizaje supervisado cuando los datos usados para el entrenamiento incluyen la solución deseada, llamada etiqueta o "*label*". Algunos algoritmos de esta categoría son la regresión lineal, la regresión logística y redes neuronales [6]. Será este último el caso de interés del proyecto.

2.1.2.2 Aprendizaje no Supervisado

Se refiere a aprendizaje no supervisado cuando los datos usados para el entrenamiento no incluyen la solución y será el algoritmo el que intente clasificar la información por sí mismo. Algunos algoritmos de esta categoría son clustering (K-means) o principal component analysis (PCA) [6].

2.1.2.3 Aprendizaje por refuerzo

Se refiere a aprendizaje por refuerzo cuando el modelo se implementa en forma de un agente que deberá explorar un espacio desconocido y determinar las acciones a llevar a cabo mediante prueba y error: aprenderá por sí mismo gracias a las recompensas y penalizaciones que obtiene de sus acciones. Este aprendizaje permite ser combinado con otros tipos, y está ahora mismo muy de moda puesto que el mundo real presenta muchos de estos escenarios [6].

2.1.3 Deep Learning

Tal y cómo se comenta en el punto 2.1.2.1 los modelos supervisados contarán con etiquetas o "labels", que es lo

que se está intentando predecir. A los datos de entrada se les conoce como "features". Un modelo o "model" es la relación entre *features* y *labels* y tiene dos fases claramente diferenciadas. En primer lugar, se establece la fase de entrenamiento donde el modelo aprende mostrándole los datos de entrada que se tienen etiquetados con su solución. Esto hace que se cree una relación entre las *features* y *labels*. Cuando el modelo es entrenado se pasa a la fase de predicción donde basándose en las relaciones creadas en el entrenamiento el modelo intenta predecir nuevos datos sin etiquetar [6].

Si se considera un modelo simple que establezca relación entre *features* y *labels* podría establecerse de la siguiente forma:

$$\vec{y} = \vec{w}\vec{x} + b$$

Ecuación 1

Donde "$\vec{y}$" es el vector de la etiqueta del dato de entrada, "$\vec{x}$" es la característica, "$\vec{w}$" es la pendiente y se llama peso (o *weight* en inglés) y "b" es el punto de intersección de la recta en el eje y se llama sesgo (o *bias* en inglés) [6].

En este contexto, por ahora se puede considerar que la fase de entrenamiento de un modelo consiste básicamente en ajustar los parámetros (los pesos wi y el sesgo b) de tal manera que el resultado de la función de *loss* (discrepancia entre los valores ideales y los reales) retorna el valor mínimo posible.

Sabiendo esto se puede definir Deep Learning o Aprendizaje profundo. Serían modelos, como el de antes mencionado, con múltiples niveles de abstracción que realizan transformaciones lineales y no lineales para obtener una salida próxima a la esperada [6].

Las redes neuronales artificiales no son más que modelos computacionales que imitan estas características arquitecturales, permitiendo que dentro del sistema global haya redes de unidades de proceso que se especialicen en la detección de determinadas características ocultas en los datos [7].

Estos modelos están compuestos por diferentes capas de neuronas, y cada capa sirve para diferente propósito donde cada parámetro importa mucho en el resultado final.

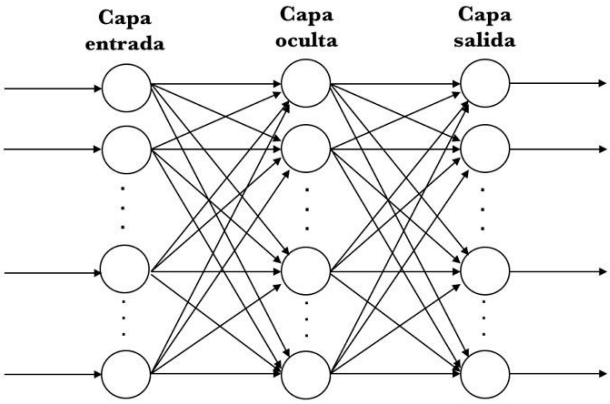


Figura 2-2. Estructura de una red neuronal [6].

En los siguientes apartados se verá cómo funcionan las redes neuronales, las funciones de activación, cómo se conecta entre sí cada neurona y la importancia de los parámetros e hiperparámetros.

2.2. Redes Neuronales

2.2.1 Perceptrón

Para entender las redes neuronales se debe comenzar por comprender el funcionamiento de su unidad mínima, la neurona o el perceptrón. Un perceptrón efectúa cálculos para detectar características o tendencias en los datos de

entrada. Según la "Perceptron Learning Rule" (regla de aprendizaje del perceptrón), el algoritmo enseña automáticamente los coeficientes de peso óptimo. Para determinar si una neurona "se enciende" o no, las características de los datos de entrada se multiplican por esos pesos. El perceptrón recibe múltiples señales de entrada. Si la suma de las señales supera un umbral determinado, se produce una señal, o, si no, no se emite ningún resultado. En el marco del método de aprendizaje supervisado de Machine Learning, es lo que permite predecir la categoría de una muestra de datos. Por tanto, se trata de un elemento esencial [8].

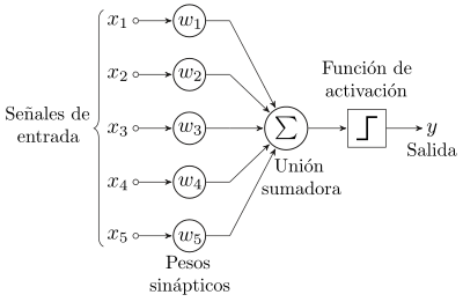


Figura 2-3. Estructura de una neurona [8].

El perceptrón es una función matemática, los datos de entrada (xi) se multiplican por los pesos (wi). El método que decide si una neurona se activa o no se le conoce como función de activación y se verá más en detalle en el siguiente apartado.

2.2.2 Función de activación

Como se comentó en el apartado anterior existen funciones que determinan si una neurona se activa o no, estas funciones son conocidas como funciones de activación. Una función de activación es, por tanto, una función que transmite la información generada por la combinación lineal de los pesos y las entradas, es decir son la manera de transmitir la información por las conexiones de salida. La información puede transmitirse sin modificaciones, función identidad, o bien que no transmita la información. Como el objetivo es que la red neuronal sea capaz de resolver problemas cada vez más complejos, las funciones de activación generalmente harán que los modelos sean no lineales [9].

Existen numerosas funciones de activación, algunas de las más populares y que serán usadas en este proyecto son:

- ReLu: Anula los valores negativos y deja los positivos tal y como están.

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

Ecuación 2

- SoftMax: Transforma las salidas a una representación en forma de probabilidades de forma que el sumatorio de todas las probabilidades es 1. Se utiliza para clasificación.

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Ecuación 3

2.2.3 Datos para alimentar una red neuronal

Tal y como se ha visto estas redes neuronales artificiales necesitan datos para funcionar. Estos datos los dividiremos en tres categorías importantes. La primera sería la de los datos de entrenamiento o training; estos servirán para entrenar el modelo y hacer que aprenda, es decir, modificar sus parámetros e hiperparámetros para predecir ayudándose de la segunda categoría, la de los datos de validación. Estos contienen la salida ideal que debe tener el modelo y hacen que las neuronas vayan cambiando sus parámetros para conseguir la salida deseada.

Por último, hay que reservar una última categoría de datos, llamados datos de predicción; estos no serán usados hasta que el modelo esté terminado y servirán para comprobar cómo de bueno es el modelo prediciendo datos que no ha visto antes.

2.2.4 Entrenamiento

Cómo se ha visto, entrenar una red neuronal no es más que hacer que aprenda los valores de los parámetros pesos (wi) y sesgo (b) de cada neurona. Esto se consigue con un proceso de propagación de los datos hacia delante y hacia atrás en la red, La primera fase en la cual los datos van hacia delante se le conoce como *forwardpropagation*, se da cuando se exponen a la red los datos de entrenamiento y estos cruzan toda la red neuronal y son calculadas las predicciones (*labels*). Es decir, pasan los datos de entrada a través de la red de tal manera que todas las neuronas apliquen su transformación a la información que reciben de las neuronas de la capa anterior y la envíen a las neuronas de la capa siguiente. Cuando los datos hayan cruzado todas las capas, y todas sus neuronas han realizado sus cálculos, se llegará a la capa final con un resultado de predicción de la etiqueta para aquellos ejemplos de entrada. A continuación, se usa una función de pérdida para estimar el error y para comparar y medir cómo fue el resultado de la predicción en relación con el resultado correcto (recordar que en este estudio se usará el aprendizaje supervisado y se dispone de las predicciones correctas). Idealmente, queremos que el error sea cero, es decir, sin divergencia entre valor estimado y el esperado. Por eso a medida que se entrena el modelo se irán ajustando los pesos de las interconexiones de las neuronas de manera automática hasta obtener buenas predicciones. Una vez se tiene calculada la pérdida, se propaga hacia atrás esta información. De ahí su nombre, retropropagación, en inglés *backpropagation*. Partiendo de la capa de salida, esa información de pérdida se propaga hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida [6].

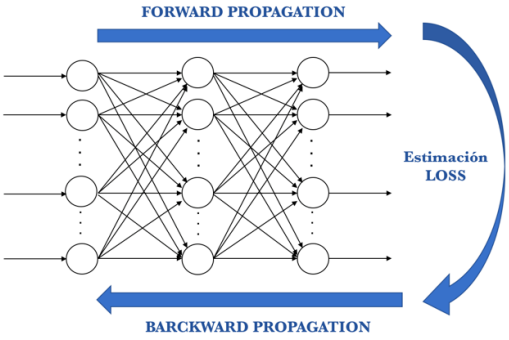


Figura 2-4. Forward Propagation, BackwardPropagation y Loss [6].

Al propagar la información de la loss hacia atrás se ajustan los valores del peso y sesgo, lo que se está haciendo es que el error se aproxime lo máximo a 0 para la próxima predicción, para ello se usa una técnica llamada *gradient descent*, esta técnica va cambiando los pesos en pequeños incrementos; esto lo va haciendo en lotes de datos (*batches*) en las sucesivas iteraciones (*epochs*) del conjunto de todos los datos que le pasamos a la red en cada iteración [6].

Entonces un resumen de como entrena una red neuronal sería: Empieza con unos parámetros aleatorios de peso y sesgo, obtiene la predicción de un conjunto de datos de entrada y compara la predicción obtenida con la etiqueta que contiene la predicción esperada y se obtiene la pérdida. Se realiza la *backpropagation* para propagar la pérdida a todo el conjunto de la red, se hace uso del *gradient descent* y se modifican los parámetros para obtener una mejor predicción y por último se realizan tantas iteraciones hasta que se considere que se ha obtenido un buen modelo.

2.2.5 Parámetros e hiperparámetros

Hasta ahora se ha hablado sobre parámetros e hiperparámetros, pero no se tiene una definición de estos. El objetivo de este apartado es diferenciarlos y ver la importancia de cada uno. Los parámetros son los valores que va

obteniendo el modelo con cada iteración y son independientes del programador, estos se refieren al sesgo y peso de cada neurona, así como el loss. Es por eso por lo que los parámetros son un componente esencial en el modelo ya que serán los que sirvan para obtener las predicciones y mejorar su grado de precisión.

En cambio, los hiperparámetros son aquellos valores que el diseñador de la red debe ajustar antes del entrenamiento y son las configuraciones que se van a seguir durante este. No existen a priori unos valores óptimos para los hiperparámetros por lo que se deberá hacer por prueba y error o tomando otros valores que hayan funcionado para problemas parecidos [10]. A continuación, se verán algunos de los hiperparámetros que se usan en el modelo de este proyecto.

2.2.5.1 Número de epochs

El número de epochs es la cantidad de veces que los datos pasan por la red [6].

2.2.5.2 Batch size

El batch size es una partición de los datos para pasarlos por la red en mini lotes. El tamaño óptimo dependerá de muchos factores, entre ellos de la capacidad de memoria del computador que usemos para hacer los cálculos [6].

2.2.5.3 Learning rate

Los algoritmos de gradient descent multiplican la magnitud del gradiente por un valor conocido, este valor es el *learning rate* y sirve para determinar el siguiente punto (ver Figura 2.5). El valor adecuado de este hiperparámetro es muy dependiente del problema en cuestión, pero en general, si este es demasiado grande, se están dando pasos enormes, lo que podría ser bueno para ir rápido en el proceso de aprendizaje, pero es posible que se salte el mínimo. Y si es demasiado pequeño la red neuronal tardaría muchísimo tiempo en encontrar la solución óptima. [6].

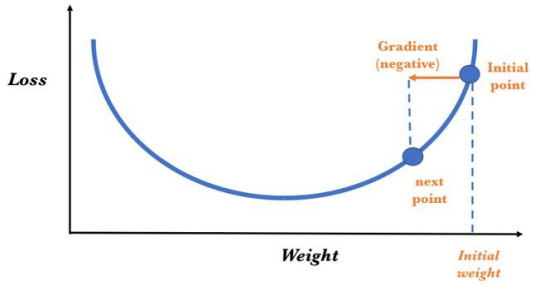


Figura 2-5. Búsqueda del valor mínimo de pérdida ajustando los pesos. [6]

Una vez visto alguno de los hiperparámetros más importantes en el siguiente capítulo se verá un tipo especial de redes neuronales, las convolucionales, las cuales será las que utilizaremos para resolver el problema de este estudio de visión por ordenador.

3 REDES NEURONALES CONVOLUCIONALES

Las redes neuronales convolucionales (en inglés CNN, Convolutional Neuronal Networks, o ConvNets) son un tipo especial de redes neuronales las cuales están preparadas para recibir imágenes como datos de entrada. Esto las hace ideales para nuestro caso de estudio el cual desea clasificar imágenes de plantas. Veremos a continuación sus componentes básicos, como implementar estas redes y las mejoras que suponen respecto a usar redes neuronales densamente conectadas.

3.1. Funcionamiento básico

Las redes neuronales convolucionales son usadas en visión por computador debido a su gran agudeza clasificando imágenes. Cada capa de estas redes aprende un nivel de abstracción, haciendo que identifiquen desde bordes, intersecciones o formas hasta elementos más complejos como la rueda de un coche o el coche entero en sí mismo si se tiene la suficiente profundidad.

La principal diferencia entre una red densamente conectada y una red convolucional es que la red densamente conectada aprenderá patrones locales en las imágenes mientras que una red convolucional aprenderá características generales de cada imagen extrapolables a otra imagen si estas no se encuentran exactamente en el mismo lugar que la imagen anterior. Es por ello por lo que las hace tan potentes para la clasificación de imágenes [11].

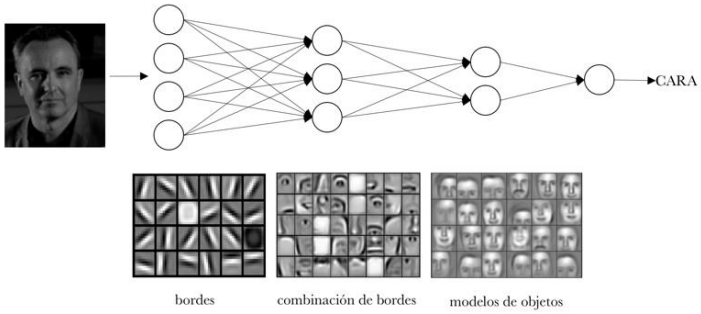


Figura 3-1. Reconocimiento de patrones de una red [6].

Ya que se tiene una visión general de cómo funcionan las redes neuronales convolucionales se pasará a explicar las dos capas esenciales en estas redes: capa de convolución y capa de pooling.

3.1.1 Convolución

En general las capas convolucionaless operan sobre tensores de tres dimensiones, dos ejes para la altura y anchura (*height* y *width*), además de un eje de canal (*channel*) también llamado profundidad (*depth*). Para una imagen de color, la dimensión del eje de profundidad es 3, pues la imagen tiene tres canales: rojo, verde y azul. Para una imagen en blanco y negro, la dimensión este eje es 1, su nivel de gris.

Cada neurona de la capa oculta estará conectada con un conjunto de píxeles de la capa de entrada (ver Figura 3-2) y se usará un filtro o *kernel* multiplicado al valor de los sesgos y pesos de cada puñado de neuronas para determinar el valor de esta neurona en la capa oculta [12].

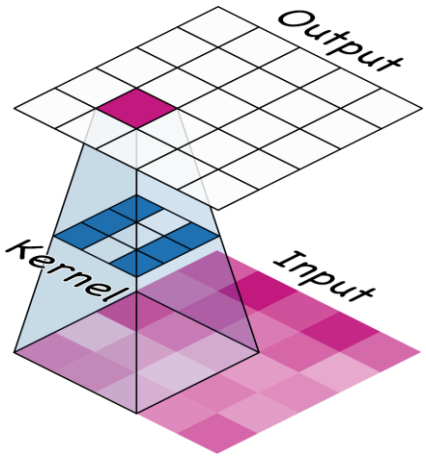


Figura 3-2. Filtro de convolución [12].

Cada neurona de la capa oculta estará calculada por el puñado de neuronas de la capa de entrada multiplicado por un filtro o *kernel*. Los filtros de una capa convolucional determinan el tipo de características que detecta durante el entrenamiento, una red convolucional intenta aprender qué características necesita para resolver el problema de clasificación. Esto significa encontrar los mejores valores para sus filtros [12].

Pero un filtro solo permite detectar una característica concreta en una imagen; por tanto, para poder realizar el reconocimiento de imágenes se propone usar varios filtros a la vez, uno para cada característica que se quiera detectar. Por eso una capa convolucional completa en una red neuronal convolucional incluye varios filtros. En la figura 3-3 se representa un ejemplo usando 32 filtros.

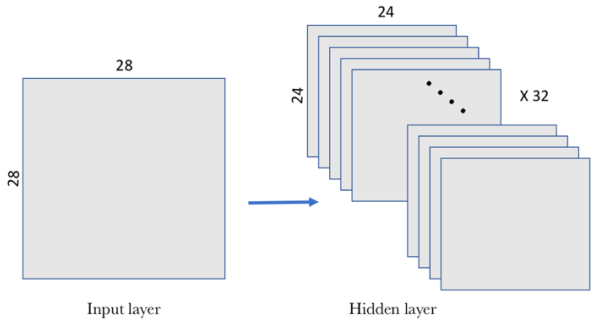


Figura 3-3. Filtros de convolución [6].

Lo más común en este tipo de redes es usar la función de activación ReLu, vista anteriormente, para cada capa de convolución.

3.1.2 Pooling

Además de la capa de convolución las redes neuronales convolucionales cuentan con una capa de *pooling*, estas condensan la información más importante obtenida en las capas de convolución simplificando la información recogida. Hay varias formas de condensar la información, pero la más efectiva y la que se usará en este caso de estudio es *max_pooling*, donde cada grupo de puntos de entrada se transforma en el valor máximo del grupo (ver Figura 3-4). Las ventanas de pooling pueden ser de varios tamaños, el más común es 2x2 [6].

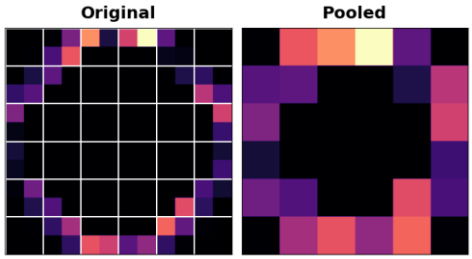


Figura 3-4. Ejemplo de *Pooling* [12].

Tal y como se mencionó anteriormente, la capa convolucional alberga más de un filtro, y, por tanto, como se aplica el *max-pooling* a cada uno de ellos por separado la capa de *pooling* contendrá tantos filtros de *pooling* como de filtros convolucionales. Cada modelo no tendrá solo una capa de convolución y una de *pooling* si no que será un modelo profundo o "deep" con varias capas interconectadas entre sí, haciendo el modelo cada vez más grande pero la imagen cada vez más condensada con la información más importante [13].

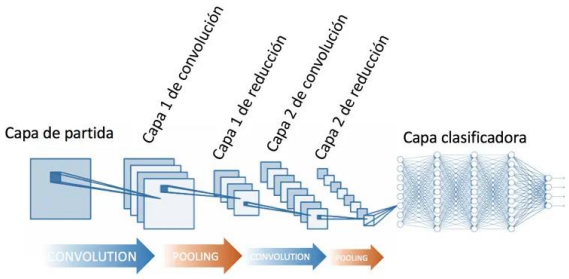


Figura 3-5. Arquitectura de una red neuronal convolucional [13].

Si la primera capa convolucional creada tiene 32 filtros, la segunda puede seguir aumentando el número de filtros o incluso repetir el mismo nivel. Se ha de recordar que la capa de *pooling* debe de ser del mismo tamaño que la capa convolucional.

Para la clasificación de las imágenes se utiliza como se mencionó en apartados anteriores la función de activación softmax, esta función trabaja con tensores en una dimensión, sin embargo, se ha visto que las redes convolucionales funcionan con tensores en tres dimensiones. Por lo tanto, al final del proceso de convolución y *pooling* habrá que "aplanar" los resultados, es decir, transformarlo todo a una dimensión, esto quiere decir que los datos pasan a ser un vector. Esto se hará con una capa denominada *Flatten* [6].

Por último, para el caso de clasificación que es el que ocupa habrá que añadir una capa de red neuronal densamente conectada con el número de neuronas equivalentes a la cantidad de categorías posibles a clasificar, es decir, si se está prediciendo imágenes de 3 tipos de plantas, la última capa deberá tener 3 neuronas a su salida, una para cada categoría con su respectiva función de activación, en este caso se usará Softmax [14].

3.3. Hiperparámetros

Los principales hiperparámetros de las redes neuronales convolucionales son el tamaño de la ventana del filtro, el número de filtros, el *padding* y el *stride*.

3.3.1 Tamaño de ventana y número de filtros

El tamaño de la ventana (height × width) es usualmente de 3×3 o 5×5. El número de filtros que nos indica el número de características que queremos manejar (output_depth) acostumbran a ser de 32 o 64 [12].

3.3.2 Padding

El padding consiste en añadir filas y columnas de 0 al tensor de entrada, se implementa en algunas capas de las

Redes Convolucionales, pues al mantener el tamaño de la imagen entre una capa y otra es posible crear redes más profundas, lo que a su vez permite extraer características más específicas de las imágenes durante el entrenamiento [15] .

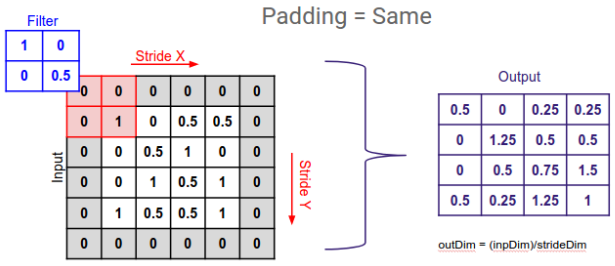


Figura 3-6. Padding [15].

3.3.3 Stride

El stride es el número de posiciones que se mueve la ventana hacia la derecha, en la figura 3-6 se ve como el *Stride* tiene valor 1, esto es lo normal o común ya que para reducir la imagen y simplificar la información se suele utilizar el Pooling. En este caso de estudio no se hará uso del Stride [12].

3.4. Overfitting y Underfitting

A la hora de entrenar las redes neuronales, en este caso las convolucionales, pueden aparecer dos fenómenos llamados sobreajuste o *overfitting* y subajuste o *underfitting*, estos vienen dados cuando la cantidad de imágenes disponibles para entrenar no es demasiado grande. Las redes neuronales convolucionales necesitan varios miles de imágenes para tener resultados sólidos, aunque con varios cientos se pueden obtener resultados decentes como se verá en próximos capítulos.

El sobreajuste ocurre cuando la red neuronal aprende demasiados detalles de unas pocas imágenes dadas, no generalizando las características de la imagen por lo que al intentar predecir una nueva probablemente falle en su predicción. Es decir, el modelo se ajusta demasiado a los datos de entrenamiento y no es capaz de predecir datos que no ha visto antes [16].

También puede ocurrir el fenómeno de subajuste, esto es que los datos son insuficientes para que el modelo aprenda. No se puede intentar clasificar varias categorías con pocas decenas de datos de entrenamiento [16].

Lo ideal sería tener los suficientes datos para que el modelo aprenda y generalice las características de la imagen sin centrarse en los pequeños detalles de los datos de entrenamiento. El overfitting es un problema muy común y se verá en el siguiente apartado varios métodos para intentar mitigarlo.

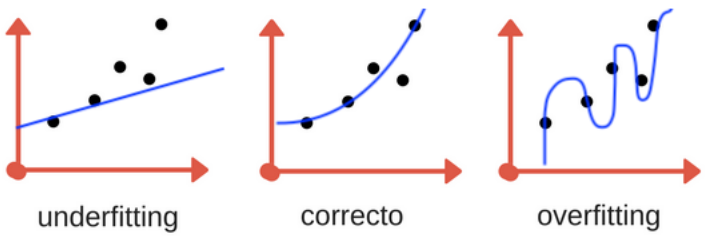


Figura 3-7. Subajuste, correcto y overfitting [17].

3.5. Data Augmentation, Transfer Learning y Dropout

Para evitar el sobreajuste una solución clara ya ha sido mencionada anteriormente y es tener más datos de entrenamiento, pero a veces esto no es posible debido a que la base de datos disponible tiene un número limitado de imágenes y no podemos aumentarlo. Debido a esto nace la función de *Data Augmentation*: ésta permite la creación de más datos a partir de los proporcionados empleando modificaciones como giros, inversiones, cambios de color, aumentar el tamaño, introducir ruido, entre muchos otros.

Esto hace que la base de datos dada aumente considerablemente "engañando" al modelo, que pensará que son imágenes diferentes cuando en realidad han sido modificadas a partir de una sola. Y a mayor número de datos mejor generaliza el modelo.

Es importante resaltar que se realiza la transformación de manera online durante el procesamiento, permitiendo hacer el proceso automático mientras se realiza el entrenamiento sin necesidad de modificar los datos almacenados en disco. Así el modelo ve una imagen generada aleatoriamente una sola vez. Evidentemente, estas transformaciones se podrían realizar previamente e incluirlas en el conjunto de datos de entrenamiento. De esta manera el preprocesado resultaría más rápido, puesto que no se realizarían las transformaciones en tiempo de ejecución, pero en cambio el espacio de almacenamiento y el tiempo de carga de los datos en memoria serían más elevados [18].

Figura 3-8. Ejemplo de data augmentation [12].

El *data augmentation* tiene limitaciones: no se puede llegar a extraer todas las características importantes con unas pocas imágenes ya que realmente las imágenes generadas son repeticiones de otras con pocas modificaciones. Pero existe una forma de poder usar una red ya pre entrenada con una gran cantidad de datos, y poder adaptarla al problema que se requiera. A este procedimiento se le conoce como *Transfer Learning* y existen dos formas de utilizar una red pre entrenada que son *Feature Extraction* y *Fine Tunning* [19].

Feature Extraction no es más que usar una red ya diseñada con los pesos ya optimizados porque ha sido entrenada con millones de imágenes y miles de clases con procesadores muy potentes capaces de soportar tal volumen de entrenamiento. Empresas como *Google* realizan modelos de tal envergadura. Es por ello por lo que el *Feature Extraction* permite hacer uso de estos modelos ya entrenados y haciendo solo pequeñas modificaciones para sus neuronas de salida se pueden adaptar al problema que se requiera [20].

Fine Tunning sigue el mismo concepto que el Feature Extraction. Sin embargo, éste se diferencia en que se modifican algunas capas de la base convolucional del modelo, es decir, se realiza un ajuste más fino para adaptarlo mejor al problema requerido lo que puede ocasionar que mejore su rendimiento [20].

Por último, la técnica conocida como Dropout "apaga" algunas neuronas ya sea de la base convolucional o las neuronas de salida, haciendo que el modelo no mecanice el aprendizaje y así evite el overfitting. Es una técnica ampliamente usada y ha demostrado su buen funcionamiento [21].

3.6. Curvas de aprendizaje, Matriz de confusión y Reporte de clasificación.

Los resultados del entrenamiento por cada época mostrarán tanto el valor de las precisiones para los datos de entrenamiento y para los datos de validación como el error para ambos datos. Estos valores quedarán representados en una gráfica y formarán una curva creciente para el caso de la precisión y una curva decreciente para el caso de la función de pérdida. A estas curvas se les conoce como curvas de aprendizaje.

Las curvas de aprendizaje se utilizan para ver la evolución del modelo, comprobar si aprende correctamente y si va disminuyendo la función de pérdida o si, en cambio, le cuesta aprender debido al subajuste o sobreajuste y la función de pérdida es creciente. Es realmente importante aprender a interpretar las curvas de aprendizaje ya que esto puede ayudar a detectar posibles fallos en el modelo.

Generalmente para evaluar modelos sencillos solo hay que centrarse en su precisión, es decir, la proporción entre las predicciones correctas que ha hecho el modelo y el total de predicciones. Sin embargo, aunque en ocasiones resulta suficiente, otras veces es necesario profundizar más y tener en cuenta los tipos de predicciones correctas e incorrectas que realiza el modelo en cada una de sus categorías.

Una herramienta para evaluar modelos es la matriz de confusión (*confusion matrix*), es una tabla con filas y columnas que contabilizan las predicciones en comparación con los valores reales. Se usa esta tabla para entender mejor cómo el modelo se comporta y es muy útil para mostrar de forma explícita cuando una clase es confundida con otra [6]. Una matriz de confusión tiene la siguiente estructura.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 3-9. Estructura de una matriz de confusión binaria.

En la que VP es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo. VN es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo. FN es la cantidad de positivos que fueron clasificados incorrectamente como negativos. FP es la cantidad de negativos que fueron clasificados incorrectamente como positivos. Con esta matriz de confusión, la precisión se puede calcular sumando los valores de la diagonal y divido por el total:

$$Accuracy = (VP + VN) / (VP + FP + VN + FN)$$

Ecuación 4. Precisión del modelo.

Pero ¿como se muestra una matriz de confusión para modelos de más de una categoría?, se ve en la siguiente figura.

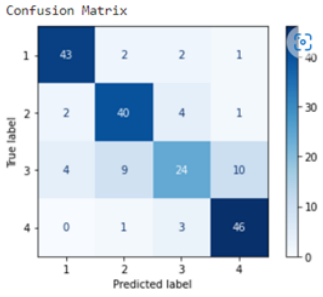


Figura 3-10. Matriz de confusión de un modelo de 4 clases.

En este caso los elementos de la diagonal representan el número de puntos en que la etiqueta que predice el

modelo coincide con el valor real de la etiqueta, mientras que los otros valores nos indican los casos en que el modelo ha clasificado incorrectamente. Por tanto, cuanto más altos son los valores de la diagonal mejor será la predicción.

También existe una función que devolverá un reporte con porcentajes de interés sobre la precisión para cada clase de los datos de predicción. Esta función se denomina *classification_report()* de la librería *Sklearn.* Los parámetros son los siguientes:

Classification Report

	precision	recall	f1-score	support
1	0.96	0.92	0.94	48
2	0.92	0.96	0.94	47
accuracy			0.94	95
macro avg	0.94	0.94	0.94	95
weighted avg	0.94	0.94	0.94	95

Figura 3-11. Ejemplo de Classification Report.

- *Precision*: Porcentaje de predicciones positivas correctas respecto al total de predicciones positivas.
- *Recall*: Porcentaje de predicciones positivas correctas en relación con el total de positivas reales.
- *F1-score*: Media armónica ponderada de la *precision* y el *recall*. Cuanto más se acerque a 1, mejor será el modelo.
- *Support*: Estos valores indican cuantas imágenes pertenecen a cada clase en el conjunto de datos de prueba o test.

Los promedios indicados incluyen la *macro average* (que promedia la media no ponderada por etiqueta) y la *weighted average* (que promedia la media ponderada por soporte por etiqueta). El *micro average* (que promedia el total de verdaderos positivos, falsos negativos y falsos positivos) sólo se muestra para la clasificación multietiqueta o multiclase con un subconjunto de clases, porque de lo contrario corresponde a la *accuracy* (como en la figura 3-11) y sería la misma para todas las métricas.

4 PREPROCESADO DE LOS DATOS

En este capítulo se presentará una herramienta para preprocesar las imágenes de la base de datos antes de ser usadas por el modelo de DL. Se comprobará en capítulos posteriores como éste preprocesado mejora la precisión ya que ayudará al modelo a distinguir mejor el número de hojas de cada imagen. A continuación, se van a presentar las técnicas que serán usadas en los experimentos, así como los programas empleados y el entorno utilizado.

4.1. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto que incluye varios cientos de algoritmos de visión por ordenador. OpenCV tiene una estructura modular, lo que significa que el paquete incluye varias bibliotecas compartidas o estáticas [22].

Figura 4-1. Logo de OpenCV.

Se usará openCV ya que contiene numerosos tutoriales que facilitan la implementación, en este caso se va a realizar en *python* (Ver sección 5.1). OpenCV contiene un apartado para preprocesado de imágenes que es el que interesa para este proyecto, donde se encuentran numerosos métodos de preprocesado, a continuación, se verán los métodos que se emplearán posteriormente.

4.2. Cambiar el espacio del color

También conocido por su traducción al inglés *Changing Color-space* este método permite cambiar de un espacio de color a otro, como puede ser BGR↔ Gray, BGR ↔ HSV, entre otros. Para la conversion de color, se usa la función *cv.cvtColor(input_image, flag)* donde *flag* determina el tipo de conversión. Para la conversion BGR → GRAY se usa el *flag cv.COLOR_BGR2GRAY*. Similarmente para BGR → HSV, se usa el *flag cv.COLOR_BGR2HSV*.

En este caso de estudio se pasará de BGR a HSV, esto se puede usar para extraer un objeto de color. En HSV, es más fácil representar un color que en el espacio BGR. HSV son las siglas en inglés de matiz, saturación y valor respectivamente. Por lo que los colores vendrán representados en una matriz de tres elementos donde el primer valor corresponde al matiz, el segundo a la saturación y el tercero al valor o brillo de éste. Estos valores van desde el 0 al 180 para el matiz, del 0 al 255 para la saturación y del 0 al 255 para el valor. Todo lo explicado se puede ver en la siguiente figura:

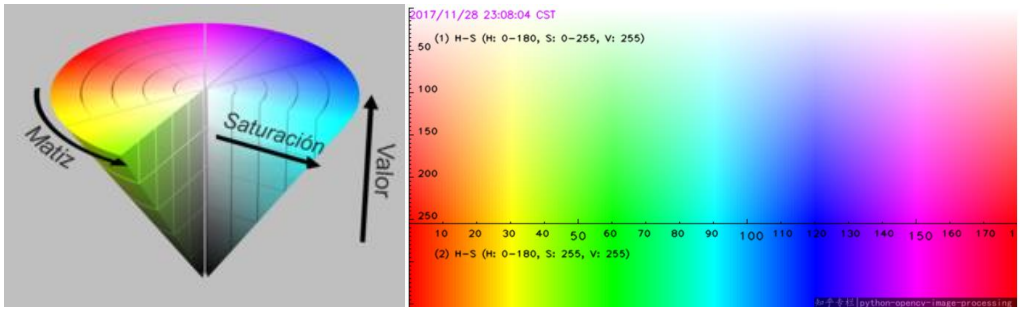


Figura 4-2. Rango de valores para todos los colores en HSV.

Siguiendo la Figura 4-2, en la imagen de la derecha se ve cómo los valores del matiz (*Hue*) están representados en el eje horizontal, los valores de la saturación (*Saturation*) en el eje vertical y los valores del brillo (*Value*) corresponderían a la profundidad representada en la imagen de la izquierda.

Por ejemplo, el color verde aproximadamente comprende el rango desde (30,25,25) hasta (80,255,255) para que sea capaz de detectar todas las tonalidades.

Una vez se tienen los valores que se quieren extraer de la imagen, el método *cv.inRange()* extrae los objetos que están compuestos por el rango de color elegido creando una imagen con dicho objeto de color blanco y el fondo negro, dicha imagen se denomina máscara. Por último, a dicha máscara se le aplicará la operación *and* con la imagen original, esto hará que el objeto extraído tenga su color original mientras el fondo permanecerá negro [23].

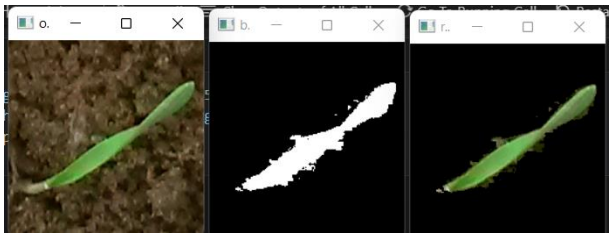


Figura 4-3. Imagen original, máscara y resultado final.

4.3. Transformaciones Morfológicas

Las transformaciones morfológicas son algunas operaciones simples basadas en la forma de la imagen. Normalmente son aplicadas en imágenes binarias. Son necesarias dos entradas, una es la imagen original, la segunda es la estructuración del elemento o kernel que es quien decide la naturaleza de la operación. Existen numerosos métodos, en este caso se va a usar la erosión ya que va a resultar de utilidad, más adelante se verá dicha utilidad.

La idea básica de la erosión es, simplemente, eliminar los bordes de los objetos de la imagen. El kernel se compara con la imagen (como en la convolución). Un píxel en la imagen original (tanto 1 como 0) será considerado 1 solo si todos los píxeles alrededor del kernel son 1, en otro caso es erosionado (se hace 0).

Lo que ocurre es lo siguiente, todos los píxeles cercanos a los límites serán descartados en función del tamaño del kernel. Así, el grosor o el tamaño del objeto en primer plano disminuye o simplemente la región blanca disminuye en la imagen. Es útil para eliminar pequeños ruidos blancos o separar dos objetos conectados que es el caso de este estudio [24].

Figura 4-4. Ejemplo de erosión [24].

5 IMPLEMENTACIÓN Y EXPERIMENTOS

5.1 Entorno

Todo lo explicado anteriormente corresponde a un marco teórico, pero ¿cómo funcionan las redes neuronales en la práctica? ¿Qué hay que hacer para diseñarlas? Para resolver estas preguntas es necesario hablar de Keras. Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Está especialmente diseñada para posibilitar la experimentación rápida con redes de aprendizaje profundo. Sus fuertes son el ser amigable para el usuario, modular y extensible. Su autor principal es el ingeniero de *Google* François Chollet. En 2017, el equipo de TensorFlow de Google decidió ofrecer soporte a Keras en la biblioteca de core de TensorFlow3. Además del soporte para las redes neuronales estándar, Keras ofrece soporte para las Redes Neuronales Convolucionales. Ofrece un conjunto de abstracciones más intuitivas y de alto nivel haciendo más sencillo el desarrollo de modelos de aprendizaje profundo [25].

Figura 5-1. Logo de Keras [25].

Debido a que Keras es básicamente una librería de Python, se requiere hacer un uso completo del intérprete de Python. Es por ello por lo que se va a usar *Jupyter notebook,* puesto que es un entorno de desarrollo muy extendido que permite sacar partido a la interactividad de Python y, a la vez, proporciona una gran versatilidad para compartir parte de códigos con anotaciones a través de la comodidad e independencia de plataforma que ofrece un navegador web [6] [26].

Por este motivo, los *notebooks* son usados a menudo para desarrollar redes neuronales en la comunidad de científicos e ingenieros de datos. Un notebook es un fichero generado por *Jupyter Notebook* que se puede editar desde un navegador web, permitiendo mezclar la ejecución de código Python con anotaciones [6].

Figura 5-2. Logo de jupyter notebook [26].

Un problema conocido y ya mencionado anteriormente sobre las redes neuronales es que entrenar la red con los datos disponibles puede llevar horas. Una forma de hacer que este proceso se aligere es usar la GPU del ordenador a la hora de entrenar, es decir la tarjeta gráfica, en vez de la CPU que es lo que usará por defecto. Usar la GPU incrementa la velocidad de entrenamiento debido a que estos procesadores realizan más rápido los cálculos necesarios que las CPU [6]. Para realizar este cambio será necesario instalar un *software* llamado *cuDNN.*

La biblioteca de redes neuronales profundas de NVIDIA CUDA® (cuDNN) es una biblioteca de primitivas para

redes neuronales profundas acelerada en la GPU. cuDNN proporciona implementaciones altamente ajustadas para rutinas estándar como la convolución hacia delante y hacia atrás, la agrupación, la normalización y las capas de activación. Los investigadores y desarrolladores de marcos de aprendizaje profundo de todo el mundo confían en cuDNN para la aceleración de alto rendimiento en la GPU. cuDNN acelera marcos de aprendizaje profundo ampliamente utilizados, como Keras, MATLAB y TensorFlow [27].

Figura 5-3. Logo de Cuda [27].

La GPU utilizada en este caso de estudio es la siguiente:

```
NVIDIA GeForce GTX 1050 Ti with Max-Q Design
```

Figura 5-4. GPU del pc usado.

Por último, para almacenar todos los códigos que se han ido creando se ha utilizado *github* [28]. *Github* es una herramienta que permite almacenar todo el código creado de manera online lo que hace que esté seguro y que se pueda acceder desde cualquier dispositivo a través del siguiente enlace.

- https://github.com/fgfcad/TFGfgf

En él se pueden encontrar todos los códigos que se hacen referencia durante la implementación del proyecto, así como la salida que generaron.
Para comenzar con el proyecto es necesario importar todas las librerías a usar, las librerías que se van a usar son las siguientes:

```python
import os
import shutil
import zipfile
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from tensorflow.keras.preprocessing import image
from random import sample
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

Código 5-1. Librerías usadas en el proyecto.

- *os*: Se utiliza para trabajar con directorios y ficheros en Python.
- *shutil:* Se utiliza para copiar, mover, eliminar o cualquier otra operación con archivos en Python.
- *zipfile:* Se utilizará para descomprimir la base de datos ya que está en .zip.
- *matplotlib:* Servirá para representar las curvas de aprendizaje.
- *tensorflow:* Incluye Keras. Será esencial para la creación de la red neuronal.
- *imagedatagerator:* Ver sección 5.3.5.
- *numpy:* Da soporte para crear vectores y matrices grandes multidimensionales.
- *random:* Se usará para dividir las imágenes de cada categoría aleatoriamente.
- *sklearn.metrics:* Servirá para representar la matriz de confusión y el reporte de clasificación.

5.2 Base de datos

Para la realización del proyecto se requiere una base de datos que contenga plantas clasificadas por su número de hojas contadas. La base de datos elegida es la siguiente:

- Leaf Counting Dataset [29]

Esta base de datos contiene 9372 imágenes de plantas de diferentes especies con su número de hojas contadas y clasificadas en directorios (ver Figura 5-1). Las imágenes se recogieron en campos de toda Dinamarca utilizando cámaras de teléfonos móviles Nokia y Samsung; cámaras de consumo Samsung, Nikon, Canon y Sony; y una cámara industrial Point Grey [30].

Los directorios que la componen son los siguientes:

|___1
|___2
|___3
|___4
|___5
|___6
|___7
|___8
|___9+

Figura 5-1. Árbol de directorios de la base de datos

En el directorio 1 están las imágenes con una hoja:

Figura 5-2. Imágenes de 1 hoja de la BBDD.

En el directorio 2 están las imágenes de dos hojas:

Figura 5-3. Imágenes de 2 hojas de la BBDD.

En el directorio 3 están las imágenes de tres hojas:

Figura 5-4. Imágenes de 3 hojas de la BBDD.

En el directorio 4 están las imágenes de cuatro hojas:

Figura 5-5. Imágenes de 4 hojas de la BBDD.

En el directorio 5 están las imágenes de cinco hojas:

Figura 5-6. Imágenes de 5 hojas de la BBDD.

En el directorio 6 están las imágenes de seis hojas:

Figura 5-7. Imágenes de 6 hojas de la BBDD.

En el directorio 7 están las imágenes de siete hojas:

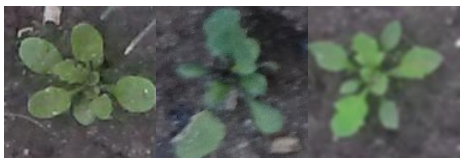


Figura 5-8. Imágenes de 7 hojas de la BBDD

En el directorio 8 están las imágenes de ocho hojas:

Figura 5-10. Imágenes de 8 hojas de la BBDD

En el directorio 9+ están las imágenes de nueve hojas o más:

Figura 5-11. Imágenes de 9 hojas o más de la BBDD

La base de datos viene en extensión *zip* por lo que lo primero será descomponerla y guardarla en un directorio, que se denominó *hojas*.

```
# Descomprime la base de datos

local_zip = "C:/Users/ferga/Documents/Python/hojas/WeedCountImages_v1.zip"

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall("C:/Users/ferga/Documents/Python/hojas")
zip_ref.close()
```

Código 5-2. Descomposición de la BBDD.

El número de imágenes que se tienen de cada categoría viene representado por la Figura 5-12. Donde se ve la gran diferencia en el reparto, las clases de 2 hojas y 4 hojas son las más abundantes mientras que las de 7 y 8 hojas apenas cuentan con algo más de un centenar de imágenes. El código empleado para contarlas es el Código 5-3.

```
dir1 = "C:/Users/ferga/Documents/Python/hojas/1"
dir2 = "C:/Users/ferga/Documents/Python/hojas/2"
dir3 = "C:/Users/ferga/Documents/Python/hojas/3"
dir4 = "C:/Users/ferga/Documents/Python/hojas/4"
dir5 = "C:/Users/ferga/Documents/Python/hojas/5"
dir6 = "C:/Users/ferga/Documents/Python/hojas/6"
dir7 = "C:/Users/ferga/Documents/Python/hojas/7"
dir8 = "C:/Users/ferga/Documents/Python/hojas/8"
dir9 = "C:/Users/ferga/Documents/Python/hojas/9+"
for path in os.listdir(dir1):
    if os.path.isfile(os.path.join(dir1, path)):
        imag_1_hoja += 1
for path in os.listdir(dir2):
    if os.path.isfile(os.path.join(dir2, path)):
        imag_2_hojas += 1
for path in os.listdir(dir3):
    if os.path.isfile(os.path.join(dir3, path)):
        imag_3_hojas += 1
for path in os.listdir(dir4):
    if os.path.isfile(os.path.join(dir4, path)):
        imag_4_hojas += 1
for path in os.listdir(dir5):
    if os.path.isfile(os.path.join(dir5, path)):
        imag_5_hojas += 1
for path in os.listdir(dir6):
    if os.path.isfile(os.path.join(dir6, path)):
        imag_6_hojas += 1
for path in os.listdir(dir7):
    if os.path.isfile(os.path.join(dir7, path)):
        imag_7_hojas += 1
for path in os.listdir(dir8):
    if os.path.isfile(os.path.join(dir8, path)):
        imag_8_hojas += 1
for path in os.listdir(dir9):
    if os.path.isfile(os.path.join(dir9, path)):
        imag_9_hojas += 1
print('Imagenes de 1 hoja: '+ str(imag_1_hoja))
print('Imagenes de 2 hojas: '+ str(imag_2_hojas))
print('Imagenes de 3 hojas: '+ str(imag_3_hojas))
print('Imagenes de 4 hojas: '+ str(imag_4_hojas))
print('Imagenes de 5 hojas: '+ str(imag_5_hojas))
print('Imagenes de 6 hojas: '+ str(imag_6_hojas))
print('Imagenes de 7 hojas: '+ str(imag_7_hojas))
print('Imagenes de 8 hojas: '+ str(imag_8_hojas))
print('Imagenes de 9 hojas: '+ str(imag_9_hojas))
```

Código 5-3. Cuenta el número de imágenes.

```
Imagenes de 1 hoja: 908
Imagenes de 2 hojas: 3292
Imagenes de 3 hojas: 940
Imagenes de 4 hojas: 2082
Imagenes de 5 hojas: 601
Imagenes de 6 hojas: 650
Imagenes de 7 hojas: 169
Imagenes de 8 hojas: 160
Imagenes de 9 hojas: 570
Total:  9372
```

Figura 5-12. Número de imágenes en la BBDD

Se ha elegido esta base de datos principalmente debido a que ya tiene las imágenes de las hojas contadas y clasificadas en sus respectivos directorios, lo que ahorrará una gran cantidad de trabajo ya que usando ImageDataGenerator de keras (se verá en el punto 5.2.5) es conveniente que todas las imágenes se encuentren en sus respectivos directorios para poder asociarle una etiqueta con la clase a la que pertenezcan.

5.3 Implementación de la Red Neuronal

5.3.1 División de imágenes

En el siguiente apartado se van a crear los directorios que contendrán las imágenes de entrenamiento, validación y predicción. Es por ello por lo que se harán tres directorios llamados *entrenamiento, validación y test*. Cada uno de ellos deberá contener 9 subdirectorios a su vez para albergar las imágenes de cada categoría anteriormente vista.

Una vez creados los directorios se debe elegir qué porcentaje de las imágenes servirá para el entrenamiento, cuanto para validación y cuanto para predicción. Los datos de entrenamiento, cómo se ha visto en secciones anteriores (ver sección 2.2.3), serán los que hagan que la red aprenda. Los datos de validación son aquellos que la red neuronal no verá como parte del entrenamiento, pero serán usados para comprobar qué tan bien o qué tan mal se

ha entrenado el modelo en cada *epoch*. Con las métricas, como la *Accuracy*, que se pueden obtener de este conjunto de datos de validación, servirán de guía para decidir como sintonizar los *hiperparámetros* del algoritmo antes de repetir el proceso de entrenamiento para otra *epoch* (ver sección 2.2.3*)*. Por último, los datos de predicción o *test* son aquellos datos que el modelo no ha visto nunca anteriormente durante la etapa de entrenamiento (ni como datos de entrenamiento ni como datos de validación), permitiendo obtener una medida de comportamiento del algoritmo más objetiva y evaluar si nuestro modelo generalizado correctamente [20].

Leyendo varios artículos y haciendo varios experimentos se llegó a la conclusión que un buen reparto en este caso sería de 80-10-10 (%) para entrenamiento, validación y predicción respectivamente [31]. Se ha cogido este reparto mayormente porque se dispone de pocos datos de entrenamiento así que la mejor opción es proporcionar una gran cantidad de imágenes para entrenar dejando lo mínimo recomendado para validación y predicción.

```
n_imag_train_1 = round(imag_1_hoja*0.8)
n_imag_validation_1 = imag_1_hoja//10
n_imag_test_1 = imag_1_hoja-n_imag_train_1-n_imag_validation_1
print('1 hoja: Train=' + str(n_imag_train_1) + ' Validation=' + str(n_imag_validation_1) + ' Test=' + str(n_imag_test_1))
```

Código 5-4. Número de imágenes aplicando el 80-10-10 % para la primera categoría.

```
1 hoja: Train=726 Validation=90 Test=92
2 hojas: Train=2634 Validation=329 Test=329
3 hojas: Train=752 Validation=94 Test=94
4 hojas: Train=1666 Validation=208 Test=208
5 hojas: Train=481 Validation=60 Test=60
6 hojas: Train=520 Validation=65 Test=65
7 hojas: Train=135 Validation=16 Test=18
8 hojas: Train=128 Validation=16 Test=16
9 hojas: Train=456 Validation=57 Test=57
```

Figura 5-13. Reparto de las imágenes de cada categoría.

Para ordenar las fotos en sus respectivas carpetas siguiendo los porcentajes anteriores se ha usado el siguiente código disponible en el fichero *Contador_de_hojas_vfinal.ipynb.*

```
# Ordena cada foto en su respectiva carpeta de train, test o validation

# 1 hoja
contenido_1 = os.listdir(dir1)
contenido_1_random = sample(contenido_1, len(contenido_1))
for i in range(len(contenido_1_random)):
    if i < n_imag_train_1:
        shutil.copy(os.path.join(dir1, contenido_1_random[i]), train_weed1_dir)
    elif i >= n_imag_train_1 and i < n_imag_train_1+n_imag_validation_1:
        shutil.copy(os.path.join(dir1, contenido_1_random[i]), validation_weed1_dir)
    else:
        shutil.copy(os.path.join(dir1, contenido_1_random[i]), test_weed1_dir)
```

Código 5-5. Reparte las imágenes en sus respectivos directorios de train, validation y test.

Se comprueba que cada foto está bien ordenada, para ello se puede ver en el fichero del programa las salidas de dichas comprobaciones. Ahora se pasará al diseño de la red neuronal.

5.3.2 Creación de la red neuronal convolucional

Tras probar diversas estructuras e investigar su funcionamiento la red neuronal con mejor rendimiento para este problema y por tanto la elegida se puede ver en el siguiente código.

```
#Creación de la CNN

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32,(3,3), activation='relu', input_shape=(150,150,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64,(3,3), activation='relu' ),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(9, activation='softmax')
])
```

Código 5-6. Red neuronal Convolucional

Donde la capa de entrada (o "input layer" en inglés) cuenta con 32 filtros o kernel de 3x3 y el tensor usado es de 150x150x3, el 3 es debido a que las imágenes son a color (RGB). La función de activación elegida es ReLu debido a su buen rendimiento en problemas de clasificación (ver sección 3.1.1). Se realiza la operación de maxpooling después de cada capa convolucional con una ventana de 2x2 (ver sección 3.1.2). Se realizan 4 niveles de abstracción hasta obtener 256 filtros kernel, este nivel de abstracción se repite y se procede al aplanado. Después del aplanado, vienen las capas densamente conectadas, estas últimas serán las encargadas de la clasificación, es por eso por lo que la última capa deben de ser estrictamente del mismo número que las categorías con las que se cuentan. Sabiendo esto se añade una capa de neuronas densamente conectadas, exactamente 256 neuronas, para buscar características que quizás la red neuronal convolucional no haya encontrado. Y por último se añaden 9 neuronas, que son las encargadas de clasificar las imágenes en las 9 categorías que se disponen, la función de activación de estas últimas será softmax debido a que es la necesaria para clasificación de imágenes (ver sección 3.3.1). Se ha elegido esta estructura tras probar otras estructuras similares y realizar pequeñas modificaciones a partir de modelos que han demostrado su buen comportamiento en clasificación de imágenes.

Se puede ver un resumen de la red neuronal creada en la siguiente figura:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 256)	295168
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 256)	0
conv2d_4 (Conv2D)	(None, 5, 5, 256)	590080
max_pooling2d_4 (MaxPooling2	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 256)	262400
dense_1 (Dense)	(None, 9)	2313

Total params: 1,243,209
Trainable params: 1,243,209
Non-trainable params: 0

Figura 5-5. Resumen de la red diseñada.

La red neuronal creada contiene 1.2 millones de parámetros. Es lo suficientemente profunda para resolver el problema complejo que se presenta y lo suficientemente extensa para entrenar el gran número de imágenes con el que se cuenta.

5.3.3 Hiperparámetros

Como se vio en secciones anteriores la elección de los hiperparámetros es crucial para el buen funcionamiento de la red. En este caso, los hiperparámetros elegidos han sido los siguientes.

```
model.compile(loss='categorical_crossentropy',
              optimizer= 'sgd',
              metrics=['accuracy'])
```

Código 5-7. Hiperparámetros.

Como función de pérdida se ha usado "categorical_crossentropy". Ésta se utiliza en tareas de clasificación multiclase. Se trata de tareas en las que un ejemplo sólo puede pertenecer a una de las muchas categorías posibles, y el modelo debe decidir cuál (cómo en el caso de estudio). Softmax es la única función de activación que se recomienda utilizar con la función de pérdida "categorical_crossentropy" [32]. Para el optimizador se usa 'sgd' con el learning rate por defecto (0.01), otra opción muy común en problemas de clasificación es usar el optimizador 'Adams' aunque en este caso se ha preferido usar 'sgd' porque se ha demostrado un mejor rendimiento para problemas de clasificación de más de dos clases [33]. Y por último se usa como métrica 'accuracy' debido a que calcula cuantas veces la predicción es igual a la etiqueta, es decir, va devolviendo la precisión que es justamente lo buscado.

5.3.4 Data augmentation

Se hace uso de la técnica conocida como *Data Augmentation* (véase apartado 3.5) a través de *Image Data Generator* (véase apartado 5.2.5) para así poder aumentar la base de datos.

```
train_datagen = ImageDataGenerator(
      rescale=1.0/255.,
      rotation_range=60,
      horizontal_flip=True,
      vertical_flip=True,
      brightness_range=[0.5,1.5],
      fill_mode='nearest'
      )
validation_datagen  = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )
```

Código 5-8. Data augmentation con ImageDataGenerator.

Los cambios que se realizan a las imágenes son: rotación desde 0 hasta 60º, giros tanto verticalmente sobre su eje como horizontalmente y por último cambios en el brillo de la imagen. Estos cambios no afectan negativamente a la hora de extraer características de las imágenes ya que todas se mantienen prácticamente iguales y el único cambio es que estarán giradas o con su brillo modificado. Es por ello por lo que se han elegido estos cambios.

5.3.5 Image Data Generator

Una vez se tiene el modelo definido y configurado, se pasará a ver cómo cargar las imágenes que están en los directorios indicados, convirtiéndolas en tensores float32 y pasarlas a la red neuronal, junto con sus respectivas etiquetas. Para ello se usa un objeto generador para las imágenes de entrenamiento, las imágenes de validación y para las imágenes de prueba. Siguiendo el código 5-6 se ve como la primera capa de neuronas contiene 32 elementos y acepta imágenes del tamaño 150x150x3 (el 3 como se explicó antes hace referencia que son a color), es por ello por lo que el generador deberá generar imágenes de tamaño 150×150 y sus respectivas etiquetas. [6]

Pero, además, los datos deben ser normalizados antes de pasarlos a la red neuronal. En este caso, se preprocesarán las imágenes normalizando los valores de píxeles para que estén en el rango {0,1} (originalmente todos los valores están en el rango {0,255}). En Keras esto se consigue con el parámetro *rescale* de la clase *ImageDataGenerator* del paquete *preprocessing* de Keras (véase código 5-5).

ImageDataGenerator permite instanciar generadores de lotes de imágenes (y sus etiquetas) a través de los métodos *flow(data, labels)* o *flowfromdirectory(directory)* cuando se trata de un directorio, como es este caso. Estos generadores se pueden usar con los métodos del modelo de Keras que aceptan instancias de generadores de datos como argumento como son *fit_generator( ), evaluate_generator( ) y predict_generator( ).*

El código del caso de estudio que crea tres objetos que instancian a cada uno de los generadores respectivamente es:

```
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    target_size=(150, 150))

validation_generator =  validation_datagen.flow_from_directory(validation_dir,
                                                               batch_size=32,
                                                               class_mode  = 'categorical',
                                                               target_size = (150, 150))


test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   shuffle=False,
                                                   batch_size=32,
                                                   class_mode  = 'categorical',
                                                   target_size = (150, 150))
```

```
Found 7498 images belonging to 9 classes.
Found 935 images belonging to 9 classes.
Found 939 images belonging to 9 classes.
```

Código 5-9. Image Data Generator.

Donde el tipo de *class_mode* es categorical debido a que la clasificación a realizar es de más de 2 clases (en el caso de 2 sería '*binary*').

Resaltar que en el generador para los datos de test se ha puesto la variable shuffle (que por defecto está a *true*) a *false.* Dejar este valor a *true* hará que se barajen los datos de prueba lo que puede ocasionar errores en la matriz de confusión mostrando una precisión para los datos de prueba que no coincide a la del modelo. La razón de la diferencia es que el generador produce lotes que comienzan en una posición diferente, por lo que las etiquetas y las predicciones no coincidirán, ya que se refieren a objetos diferentes.

Cómo se vio en la sección 5.3.1 los datos de entrenamiento serán los que hagan que la red aprenda las características de cada categoría para poder clasificar nuevas imágenes correctamente. Los datos de validación son aquellos que la red neuronal no verá como parte del entrenamiento, pero serán usados para comprobar qué tan bien o qué tan mal se ha entrenado el modelo en cada *epoch*. Con las métricas, como la *Accuracy*, que se pueden obtener de este conjunto de datos de validación, servirán de guía para decidir como sintonizar los hiperparámetros del algoritmo antes de repetir el proceso de entrenamiento para otra *epoch*. Por último, los datos de predicción o test son aquellos datos que el modelo no ha visto nunca anteriormente durante la etapa de entrenamiento (ni como datos de entrenamiento ni como datos de validación), permitiendo obtener una medida de comportamiento del algoritmo más objetiva y evaluar si nuestro modelo generalizado correctamente [20].

Los datos de entrenamiento y validación serán ambos usados durante la etapa de entrenamiento (ver código 5-11) mientras que los datos de predicción serán usados en la etapa de testeo o predicción para representar la precisión del modelo y la matriz de confusión (ver código 5-12 y 5-14).

5.3.6 Entrenamiento

Debido a que los datos se generan de manera indefinida, el modelo de Keras necesita saber cuántas muestras extraer del generador antes de decidir que ha finalizado una época (epoch): este es el papel del argumento *steps_per_epoch*. Cuando se usa el método *fit_generator( )* también se le puede pasar el argumento *validation_data* con el generador de las imágenes que usaremos para validar. En este caso se requiere indicar con el argumento *validation_steps* cuántas muestras deben extraerse del generador en cada epoch.

Para calcular sendos valores automáticamente se sigue el siguiente código.

```
batch_size = 64
steps_per_epoch = train_generator.n // batch_size
validation_steps = validation_generator.n // batch_size

print (steps_per_epoch)
print (validation_steps)
```

```
117
14
```

Código 5-10. Obtención de valores de steps_per_epoch y validation_steps.

Donde el *batch size* se ha elegido de 64 debido a la gran cantidad de datos totales que se tienen para que el entrenamiento no dure demasiado. Los valores de los argumentos *steps_per_epoch* y de *validation_steps* se puede

ver en el Código 5-10. Ahora solo falta definir el número de *epochs* y entrenar el modelo.

```
history = model.fit(
    train_generator,
    steps_per_epoch= steps_per_epoch,
    epochs=175,
    validation_data=validation_generator,
    validation_steps= validation_steps,
    verbose=1)
```

Código 5-11. Entrenamiento del modelo

5.3.7 Resultados y Matriz de Confusión

Para mostrar los resultados se han seguido los siguientes códigos: para la precisión de los datos de prueba el código 5-12, para las curvas de aprendizaje y error el código 5-13 y para la matriz de confusión y un resumen de los resultados el código 5-14.

```
test_lost, test_acc= model.evaluate(test_generator)
print ("Test Accuracy:", test_acc)
```

Código 5-12. Precisión para los datos de prueba.

```
acc      = history.history[     'accuracy' ]
val_acc  = history.history[ 'val_accuracy' ]
loss     = history.history[    'loss' ]
val_loss = history.history['val_loss' ]

epochs    = range(1,len(acc)+1,1) # obtener número de epochs del eje X

plt.plot  ( epochs,     acc, 'r--', label='Training acc'  )
plt.plot  ( epochs, val_acc,  'b', label='Validation acc')
plt.title ('Training and Validation Accuracy')
plt.ylabel('acc')
plt.xlabel('epochs')

plt.legend()
plt.figure()


plt.plot  ( epochs,     loss, 'r--' )
plt.plot  ( epochs, val_loss ,  'b' )
plt.title ('Training and Validation Loss'   )
plt.ylabel('loss')
plt.xlabel('epochs')

plt.legend()
plt.figure()
```

Código 5-13. Muestra las curvas de aprendizaje y error.

```
# Confusion Matrix
train_generator.reset()
Y_pred = model.predict(test_generator)
y_pred = np.argmax(Y_pred, axis=1)
target_names = ['1', '2','3','4','5','6','7','8','9+']
print('Confusion Matrix')
cm = confusion_matrix(test_generator.classes, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=target_names)
disp.plot(cmap=plt.cm.Blues)
plt.show()
print('Classification Report')
print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

Código 5-14. Muestra la matriz de confusión y reporte de los resultados.

Una vez ejecutado estos códigos se obtiene lo mostrado en la figura 5-6.

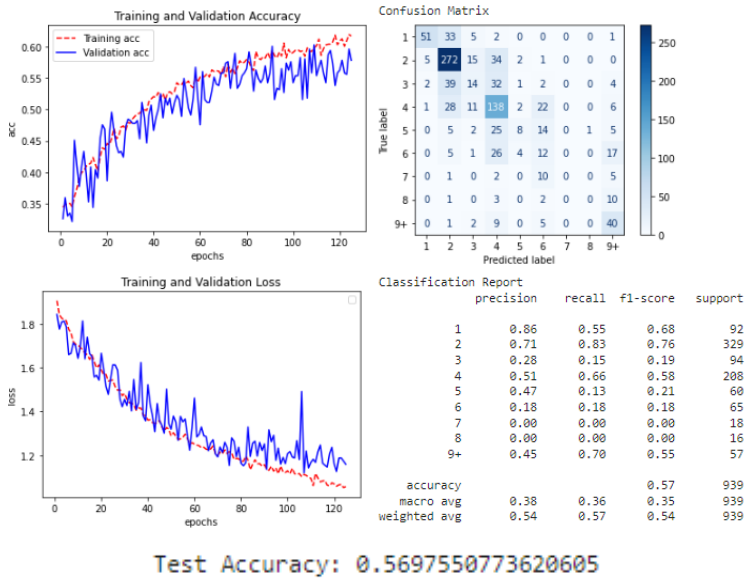


Figura 5-6. Resultados del entrenamiento

Viendo los resultados se pueden sacar varias conclusiones (como interpretar estos resultados y el significado de los parámetros ya se vio en el apartado 3.6): primero se ve que los datos de entrenamiento (curva roja de la matriz de confusión) se van adaptando a la red por cada *epoch* haciendo que la precisión de esta suba hasta más de un 60 %. La precisión de los datos de validación (curva azul de la matriz de confusión) crece hasta alrededor de un 58%. Y por último los datos de prueba dan una precisión de un 57%, esto quiere decir que el 57% de los datos que nunca ha visto antes los clasifica correctamente. Se puede ver en la matriz de confusión cómo el modelo no está funcionando como se espera, el mejor porcentaje de precisión se tiene para las imágenes de 2 y 4 hojas (ambas son las que cuentan con más imágenes para el entrenamiento) mientras que a partir de 5 hojas el acierto es casi nulo. En el reporte de clasificación como se vio en el apartado 3.6 muestra la precisión individual de las distintas categorías para los datos de prueba, se ve como las categorías con más precisión son las que cuentan con más datos como ya se ve en la matriz de confusión.

Se va a crear otro modelo el cual cuente con las mismas imágenes de entrenamiento para cada categoría para ver si mejora su rendimiento.

5.3.8 Segundo Modelo

Para el siguiente modelo se propone igualar el número de imágenes a usar en cada categoría. En el modelo anterior se tiene un número muy distante de imágenes de cada categoría lo que puede estar provocando que se entrene mucho más una clase que otra y la clasificación no sea óptima (ver fichero *Contador_de_hojas_vequeal.ipynb*).

Para este modelo se contará con el siguiente número de imágenes. El cuál es el mayor posible ya que de 9 imágenes no hay más de 570.

```
imag_1_hoja = 570
imag_2_hojas = 570
imag_3_hojas = 570
imag_4_hojas = 570
imag_5_hojas = 570
imag_6_hojas = 570
imag_9_hojas = 570
```

Código 5-15. Número de imágenes elegido.

Descartando las imágenes de 7 y 8 hojas ya que son muy escasas (menos de 200 imágenes). Es esencial modificar el número de neuronas de salida que en este caso será 7 debido a que se han eliminado dos clases. Después de hacer el reparto equitativo, se entrena la misma red neuronal que antes, obteniendo los siguientes resultados.

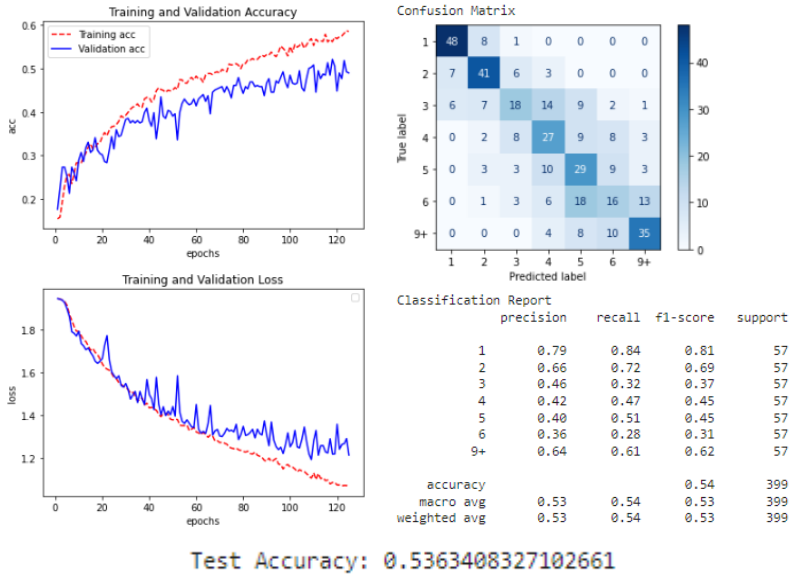


Figura 5-7. Resultados del entrenamiento.

A pesar de igualar el número de imágenes y evitar que se entrene más una categoría a otra se ve como el modelo no ha mejorado en la precisión para los datos de prueba. La precisión sigue estando alrededor de un 54 %. Cabe recordar que para que un modelo de red neuronal tenga una precisión considerada alta, este debe contar con muchísimas imágenes para entrenamiento, en este caso se cuenta con algunos centenares lo que hace que sea difícil obtener un buen porcentaje. Sin embargo, se van a probar diversos métodos para conseguir la solución más óptima.

En el siguiente apartado se probará el método de transfer learning, visto en secciones anteriores, para ver si está siendo problema del diseño del modelo o si en cambio es problema de cómo se están tratando los datos.

5.3.9 Tercer Modelo

Para este modelo se realiza la técnica de transfer learning (ver sección 3.5). Este fichero usa la red neuronal *inceptionv3* preentrenada con las imágenes de imagenet (ver fichero *Contador_de_hojas_vtg.ipynb*). El código para usar para implentar la red con los pesos ya entrenados y definidas las capas que no se requiere volver a entrenarlas es el siguiente:

```
from tensorflow.keras.applications import inception_v3

pre_trained_model = tf.keras.applications.InceptionV3(
                                include_top = False,
                                weights = 'imagenet',
                                input_shape=(150,150,3))

for layer in pre_trained_model.layers:
  layer.trainable = False


pre_trained_model.summary()
```

Código 5-16. Implementación de la red Inception_v3.

El modelo cuenta con 21 millones de parámetros ya entrenados, la salida de esta red neuronal quedará definida de la siguiente forma.

```
modelFE = tf.keras.models.Sequential([
    pre_trained_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(7, activation='softmax')
])
```

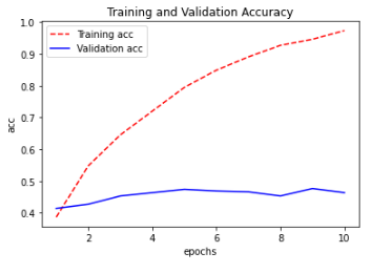
Código 5-17. Creación de las últimas capas de la red.

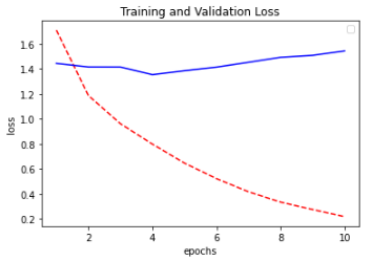
Se puede ver en el código 5-17 que se ha usado la técnica Dropout (ver sección 3.5) para reducir el overfitting. Esta salida genera 9 millones de parámetros entrenables junto a los 21 millones no entrenables de antes suman un total de 31 millones de parámetros.

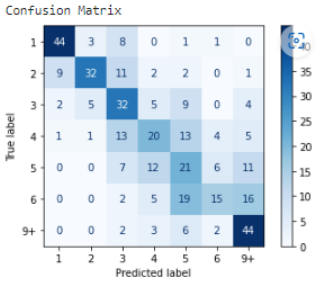
```
Total params: 31,244,071
Trainable params: 9,441,287
Non-trainable params: 21,802,784
```

Figura 5-8. Número de parámetros de la red.

Esta red será entrenada solo con 10 *epochs* ya que la mayoría de sus parámetros ya están pre entrenados. El resultado obtenido es el siguiente.







Classification Report

	precision	recall	f1-score	support
1	0.79	0.77	0.78	57
2	0.78	0.56	0.65	57
3	0.43	0.56	0.48	57
4	0.43	0.35	0.38	57
5	0.30	0.37	0.33	57
6	0.54	0.26	0.35	57
9+	0.54	0.77	0.64	57
accuracy			0.52	399
macro avg	0.54	0.52	0.52	399
weighted avg	0.54	0.52	0.52	399

```
Test Accuracy: 0.5213032364845276
```

Figura 5-9. Resultados del entrenamiento

Esta vez los datos de entrenamiento alcanzan el 100 por cien de precisión mientras que se ve como la precisión de los datos de validación no mejora respecto a los casos anteriores, es decir, se mantiene alrededor de un 50%. Lo mismo ocurre para los datos de prueba donde la precisión está alrededor de un 53 % igual que en los anteriores casos. El modelo no consigue aprender a clasificar correctamente los datos, aun contando con pocos, y al probar con una red pre entrenada no parece que sea problema del modelo, el problema parece estar en las imágenes que son realmente difíciles de clasificar, por lo que en los siguientes apartados se probarán unos experimentos para conseguir una solución que se denomine aceptable procesando las imágenes.

5.4 Experimentos

Esta base de datos ya contaba con un estudio previo en el cual habían conseguido algo menos de un 70% de precisión [29]. El estudio consistía en el uso de InceptionV3 con las últimas capas modificadas, a este proceso se le conoce como Fine Tunning, se usaron las imágenes de ImageNet y la red se entrenó 20 veces para mejorar la precisión, con todo este proceso no se logró superar un 70%.

En la sección anterior se han visto los resultados del mejor modelo conseguido para resolver este problema. Sin embargo, el porcentaje de acierto conseguido no es muy elevado y en la matriz de confusión se puede ver como el modelo falla en sus predicciones y cuáles son sus puntos más débiles. Es por ello por lo que en este apartado se pretende mostrar algunos experimentos realizados para mejorar ese porcentaje y estudiar más a fondo este problema para ver cómo solucionarlo.

5.4.1 Purgar base de datos y preprocesar las imágenes

Uno de los problemas más llamativos de esta base de datos es que tiene muchas imágenes realmente difíciles de clasificar, incluso para el ojo humano (veáse figura 5-10), es por ello por lo que se eliminarán manualmente todas estas fotos que pueden estar confundiendo más que ayudando a entrenar a la red.

Figura 5-10. Imágenes difíciles de clasificar para las categorías 1,2,3,4,5,6,7,8,9+ .

Tras esta reducción, el número resultante de hojas obtenido es el siguiente.

```
Imagenes de 1 hoja: 617
Imagenes de 2 hojas: 3064
Imagenes de 3 hojas: 615
Imagenes de 4 hojas: 786
Imagenes de 5 hojas: 467
Imagenes de 6 hojas: 502
Imagenes de 7 hojas: 169
Imagenes de 8 hojas: 160
Imagenes de 9 hojas: 494
Total:  6874
```

Figura 5-11. Número de imágenes en la base de datos.

La reducción ha sido de 2498 elementos, imágenes que no estaba nada claro a que categoría pertenecen y más que ayudar pueden estar confundiendo a la red. Cómo se ve en la Figura 5-11 se ha decidido no reducir el número de 7 y 8 hojas ya que es extremadamente pequeño para entrenar una red neuronal y no se utilizarán estas categorías ya que sus imágenes son insuficientes para entrenar la red.

También se va a realizar un preprocesado de los datos para así ayudar al modelo como se vio en el capítulo 4. El preprocesado que se va a realizar será el siguiente:

En primer lugar, se eliminará el fondo, es decir, todo aquello que no son hojas y que muestra información irrelevante para el proceso. Para ello se ha creado el siguiente código donde como se vio en el capítulo 4 se elige un rango de ‘verdes’, estos colores elegidos no serán eliminados de la imagen, luego se crea una máscara donde se extrae la zona donde aparece el rango de color elegido, esta máscara es en blanco y negro, para volver a ponerlo de su color original se utiliza el método *bitwise_and()* el cual mezcla la imagen original con la máscara, añadiendo

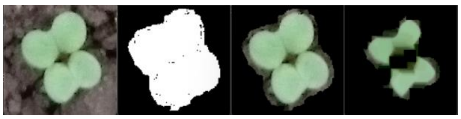
el color solo en la zona extraída. Este procedimiento se puede entender mejor viendo el código 5-18. Para mostrar las imágenes en pantalla se ha utilizado *cv.imshow()*.

```
img = cv.imread('hojas/2/7.png')

# Convertir BGR a HSV
hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
# Selección de los colores
lower_green = np.array([30,25,25])
upper_green = np.array([85,255,255])
# Threshold
mask = cv.inRange(hsv, lower_green, upper_green)
# Bitwise-AND mask e imagen original
res = cv.bitwise_and(img,img, mask= mask)
```

Código 5-18. Extracción del color verde.

Para las imágenes que tienen más de una hoja se va a hacer otra modificación, esta modificación será la erosión (Veáse apartado 4) para ello se ha empleado el código 5-19.

```
# Erosion
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(res,kernel,iterations = 1)
```



Código 5-19. Erosión.

Se ve como en la anterior imagen perteneciente a la categoría de dos hojas tiene estas muy pegadas pudiendo confundir a la red y clasificándola como una hoja, pero con la erosión estas dos hojas quedan aisladas.

Eliminación del fondo y erosión serán los dos cambios aplicados a las fotos, para aplicarlos a toda la base de datos se utiliza el código 5-20.

```
# Bucle para las imágenes de 2 hojas
i=0
while len(os.listdir('hojas/2_mod') ) < 617:
    if os.path.isfile('hojas/2/'+str(i)+'.png'):
        img = cv.imread(hoja2 + '/' + str(i) + '.png')
        hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
        # Selección de los colores
        lower_green = np.array([30,25,25])
        upper_green = np.array([85,255,255])
        # Threshold
        mask = cv.inRange(hsv, lower_green, upper_green)
        # Bitwise-AND mask and original image
        res = cv.bitwise_and(img,img, mask= mask)
        # Erosion
        kernel = np.ones((5,5),np.uint8)
        erosion = cv.erode(res,kernel,iterations = 1)
        # Guardar imagen
        cv.imwrite('hojas/2_mod/' + str(i) + '.png', erosion)
        i+=1
    else:
        i+=1
```

Código 5-20. Extracción de hojas de la imagen.

Una vez realizada la erosión se ve que algunas imágenes no han sido detectadas por el filtro y el resultado final es una imagen en negro, esto es debido a que las hojas de esas imágenes no tienen un tono verdoso y se mezclan con el color de la tierra o son negras o grisáceas. Estas imágenes han sido eliminadas ya que se escapan del objeto de este estudio.

Todo el procesado se hará en el fichero llamado *OpenCV.ipynb*.

Se va a realizar una clasificación del modelo progresiva, es decir, primero se va a clasificar entre 2 categorías, luego entre 3, después entre 4 y así hasta finalizar. Este procedimiento podrá mostrar la evolución del modelo al ir añadiendo categorías y así comprender un poco mejor su funcionamiento.

5.4.2 Clasificación entre 2 categorías

Se va a realizar una clasificación entre plantas con 1 hoja y plantas con 2 hojas (ver '*Fichero CH_V_2hojas.ipynb*'). Se seguirá un modelo entrenado desde cero con la misma estructura e hiperparámetros de los modelos anteriores definidos en el punto 5.3.2. Para ello se tomarán las nuevas rutas de las imágenes que se encuentran en diferentes directorios ya que vamos a utilizarlas preprocesadas y en las carpetas de train, validation y test se crearán a su vez 2 subdirectorios como en el apartado anterior donde irán las carpetas 1 y 2 en este caso, pertenecientes a las clases usadas. El reparto será el mismo 80% imágenes para entrenamiento, 10% para validación y 10% para test.

```
Imagenes de 1 hoja: 473
Imagenes de 2 hojas: 466
Total:  939

1 hoja: Train=378 Validation=47 Test=48
2 hojas: Train=373 Validation=46 Test=47
```

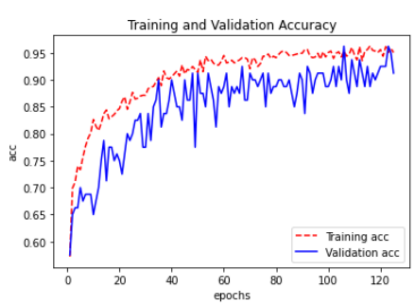
Figura 5-12. Imágenes para la realización de la clasificación y cuantas de ellas servirá para cada cometido.

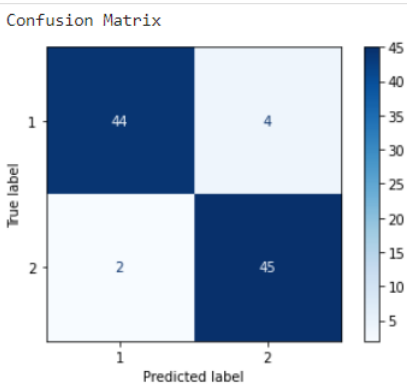
La red neuronal será entrenada desde cero siguiendo el modelo ya usado para los apartados 5.3.7 y 5.3.8, con ligeras diferencias: las neuronas de salida en este caso deben de ser 2 ya que tenemos un problema de clasificación entre 2 categorías. Las imágenes a usar serán las imágenes procesadas del apartado 5.4.1 para ver si mejora al procesar las imágenes respecto a los modelos con las imágenes sin procesar.

```
tf.keras.layers.Dense(2, activation='softmax')
```

Código 5-21. Capa final de la red.

Aplicando los mismos hiperparámetros y la técnica de data augmentation, tras entrenar la red con 125 épocas y un batch de 20, se obtienen los siguientes resultados:





Classification Report

	precision	recall	f1-score	support
1	0.96	0.92	0.94	48
2	0.92	0.96	0.94	47
accuracy			0.94	95
macro avg	0.94	0.94	0.94	95
weighted avg	0.94	0.94	0.94	95

```
Test Accuracy: 0.9578947424888611
```

Figura 5-13. Curvas de aprendizaje y error del modelo, matriz de confusión y precisión con las imágenes de prueba.

Se ve como el modelo alcanza una precisión de casi el 96 por ciento. Si se compara este mismo modelo con otro exactamente igual, pero con las imágenes sin procesar (ver fichero *CH_V_2hojas.ipynb*), se obtienen los siguientes resultados para la misma cantidad de imágenes.

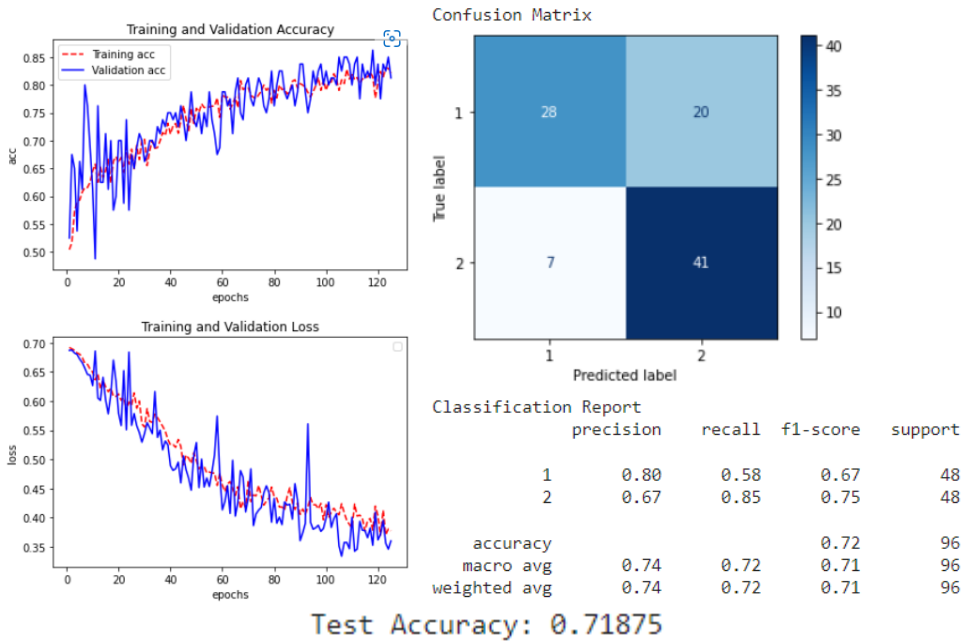


Figura 5-14. Respuesta al entrenamiento. Imágenes sin procesar

Para este modelo se ha usado el conjunto reducido de imágenes, pero sin preprocesar, es decir las imágenes serán las originales (ver fichero *CH_V_2hojas.ipynb*).

Se ve como para las mismas épocas el proceso de aprendizaje es más lento, mientras que el otro modelo llego al 90 por ciento en los datos de validación este llega al 80 por ciento. Para los datos de predicción el modelo que utiliza las imágenes preprocesadas alcanza el 95% mientras que este se mantiene en un 72%. Viendo la matriz de confusión se puede apreciar cómo predice como categoría 2 a 20 imágenes que pertenecen a la categoría de 1 hoja. Este problema claramente se arregla con la erosión debido a que aprende mejor la categoría de 2 hojas al verlas más separadas entre sí y con el fondo de la imagen en negro.

Se ha comprobado como el preprocesado ayuda a la red neuronal. Ahora se añadirá una categoría más para comprobar que le ocurre a la precisión si añadimos la categoría de 3 hojas.

5.4.3 Clasificación entre 3 categorías

Para la clasificación entre 1 hoja, 2 hojas y 3 hojas se cuentan con las siguientes imágenes.

```
Imagenes de 1 hoja: 473
Imagenes de 2 hojas: 466
Imagenes de 3 hojas: 466
Total:  1405

1 hoja: Train=378 Validation=47 Test=48
2 hojas: Train=373 Validation=46 Test=47
3 hojas: Train=373 Validation=46 Test=47
```

Figura 5-15. Número de imágenes

Al igual que en el apartado anterior se debe cambiar el número de neuronas de la última capa a 3, lo demás todo permanecerá igual para ver el efecto de añadir una clase nueva.

Tras el entrenamiento con todos los parámetros iguales y mismo número de épocas (véase el fichero '*CH_V_3hojas_cv.ipynb'*).

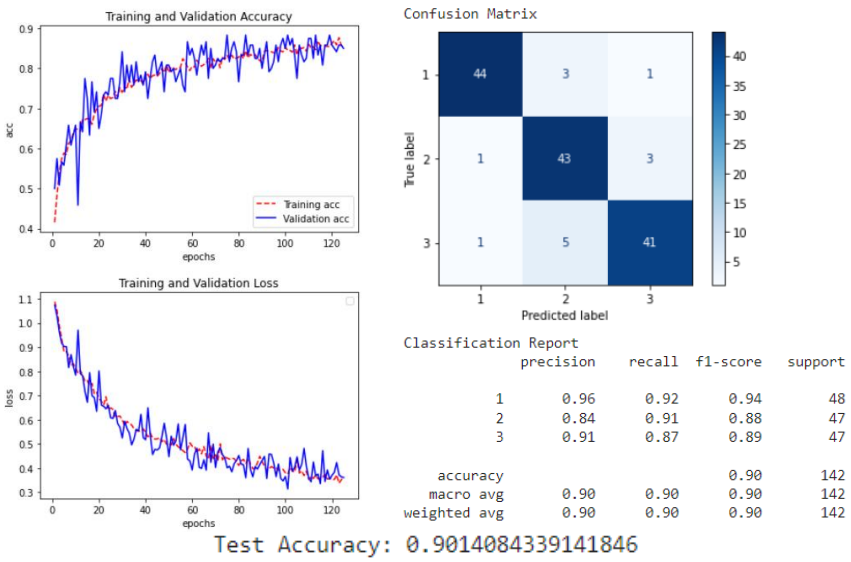


Figura 5-16. Respuesta al entrenamiento del modelo. Imágenes procesadas

La red mantiene un aprendizaje bastante bueno llegando al 90 por ciento en la precisión para los datos de prueba.

Viendo ahora el resultado con las imágenes sin modificar (véase fichero *'CH_V_3hojas.ipynb'*).

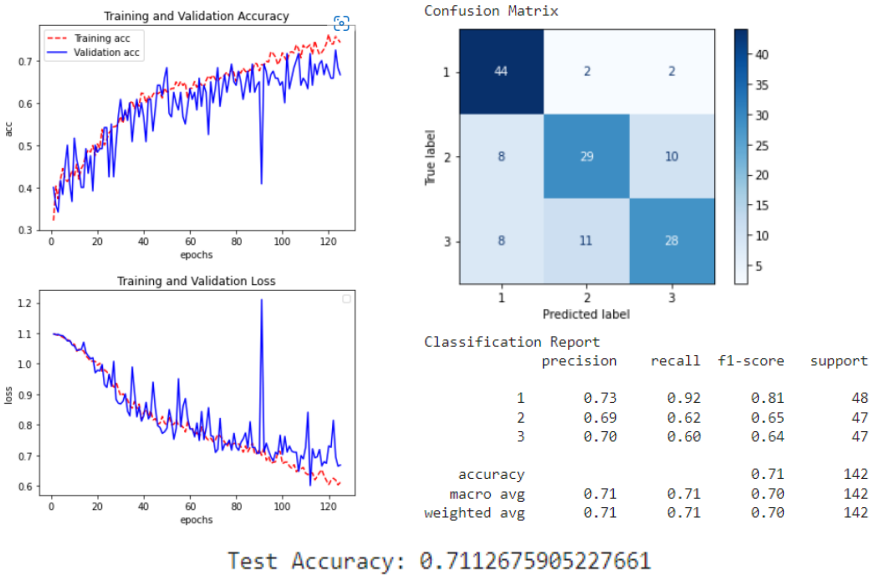


Figura 5-17. Respuesta al entrenamiento del modelo. Imágenes sin procesar.

Se aprecia como al igual que antes a la red le cuesta mucho más aprender si no se preprocesan las imágenes. Con el mismo número de *epoch* llega hasta una precisión del 70% para los datos de prueba, mientras que usando los datos preprocesados se llega a un 90%.

Se ve cómo el modificar los datos ha servido y se vuelve a demostrar para el caso de añadir una categoría más. Se añadirá a continuación la categoría de 4 hojas para observar si se sigue el mismo patrón.

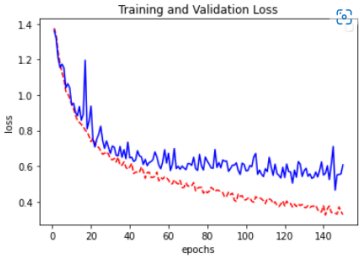
5.4.4 Clasificación entre 4 categorías.

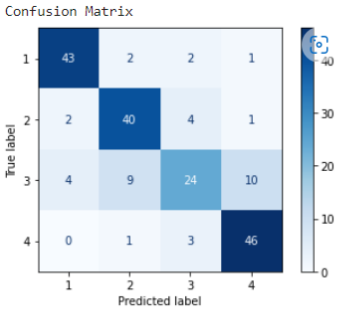
Se cuentan con las siguientes imágenes.

```
Imagenes de 1 hoja: 473
Imagenes de 2 hojas: 466
Imagenes de 3 hojas: 466
Imagenes de 4 hojas: 501
Total:  1906

1 hoja: Train=378 Validation=47 Test=48
2 hojas: Train=373 Validation=46 Test=47
3 hojas: Train=373 Validation=46 Test=47
4 hojas: Train=401 Validation=50 Test=50
```

Figura 5-18. Número de imágenes

Siguiendo con los mismos hiperparámetros y mismo número de épocas y de *batch* el resultado del entrenamiento es el siguiente (ver fichero *CH_V_4hojas_cv.ipynb*):



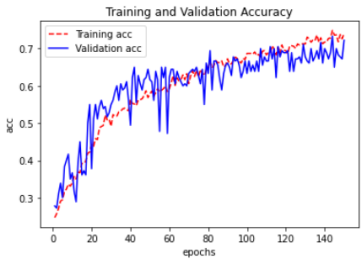


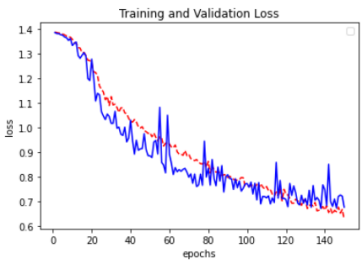

```
Classification Report
              precision    recall  f1-score   support

           1       0.88      0.90      0.89        48
           2       0.77      0.85      0.81        47
           3       0.73      0.51      0.60        47
           4       0.79      0.92      0.85        50

    accuracy                           0.80       192
   macro avg       0.79      0.79      0.79       192
weighted avg       0.79      0.80      0.79       192
```

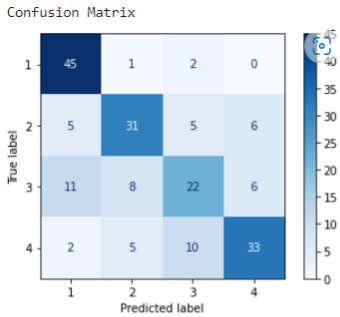
```
Test Accuracy: 0.796875
```

Figura 5-19. Respuesta al entrenamiento del modelo. Imágenes Procesadas.

Alcanza un 80 por ciento de precisión, lo cual sigue siendo bastante bueno, comparándolo con los datos sin preprocesar (ver fichero *CH_V_4hojas.ipynb*):








```
Classification Report
              precision    recall  f1-score   support

           1       0.71      0.94      0.81        48
           2       0.69      0.66      0.67        47
           3       0.56      0.47      0.51        47
           4       0.73      0.66      0.69        50

    accuracy                           0.68       192
   macro avg       0.68      0.68      0.67       192
weighted avg       0.68      0.68      0.67       192
```

```
Test Accuracy: 0.6822916865348816
```

Figura 5-20. Respuesta al entrenamiento del modelo. Imágenes sin procesar.

Se comprueba como sigue el mismo patrón, la precisión para los datos de prueba para el modelo con los datos procesados es alrededor de un 10% superior al modelo con los datos sin procesar. Se va a ver ahora que problema surge para las siguientes categorías.

También resaltar cómo al ir añadiendo más categorías la precisión va decreciendo. Esto es debido a que cómo se comentó en apartados anteriores no se tiene un gran número de imágenes y el entrenamiento no es el deseado, entonces, cuanta más categorías haya más le cuesta al modelo decidir a cuál pertenece la imagen que debe de predecir.

5.4.5 Clasificación entre 5 categorías

Las imágenes disponibles son las siguientes.

```
Imagenes de 1 hoja: 473
Imagenes de 2 hojas: 466
Imagenes de 3 hojas: 466
Imagenes de 4 hojas: 501
Imagenes de 5 hojas: 94
Total:  2000

1 hoja: Train=378 Validation=47 Test=48
2 hojas: Train=373 Validation=46 Test=47
3 hojas: Train=373 Validation=46 Test=47
4 hojas: Train=401 Validation=50 Test=50
5 hojas: Train=75 Validation=9 Test=10
```

Figura 5-21. Número de imágenes.

Con lo cual a partir de este punto existe un problema, son muy pocas las imágenes disponibles para entrenar a la red neuronal ya que la categoría de 5 hojas solo dispone de 94 hojas realmente aprovechables. Esto hace que la red no pueda aprender los parámetros que la hagan decidir que pertenecen a la categoría 5, si miramos el resultado tras haber entrenado a la red (fichero *CH_V_5hojas_cv.ipynb*).

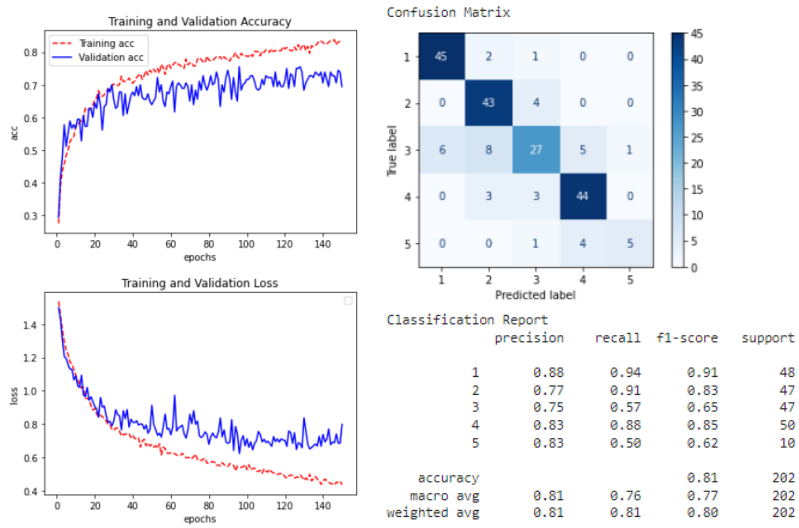


Figura 5-22. Respuesta al entrenamiento del modelo. Imágenes procesadas.

Viendo el parámetro f1-score, que como se vio en apartados anteriores define como de bueno es clasificando para una categoría en concreto, se ve que es el más distante a 1. Esto quiere decir que es la categoría que peor clasifica de las 5, claramente debido al poco volumen de imágenes con las que se cuenta, aun así, la precisión conseguida no es tan inferior pero esto es debido a que son solo 10 imágenes las que debe de clasificar mostrando una precisión de clasificación "engañosa". En este caso el parámetro f1-score es el que da la información correcta.

Para los siguientes casos se cuenta con el siguiente número de imágenes.

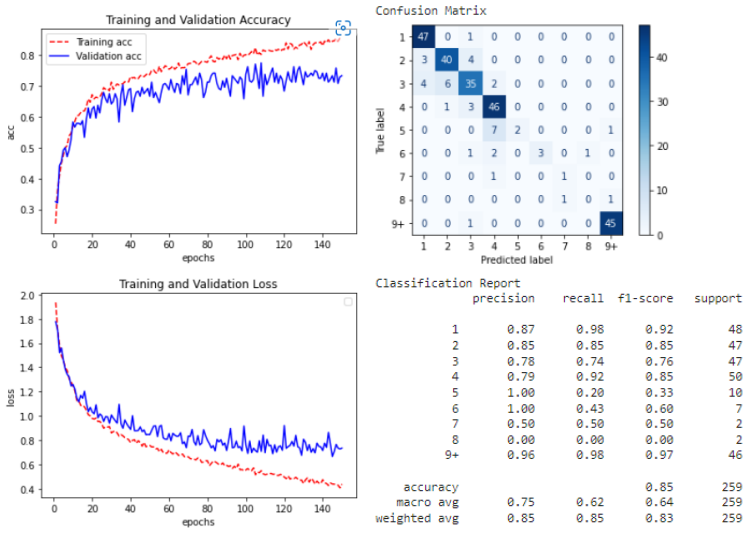
```
Imagenes de 1 hoja: 473
Imagenes de 2 hojas: 466
Imagenes de 3 hojas: 466
Imagenes de 4 hojas: 501
Imagenes de 5 hojas: 97
Imagenes de 6 hojas: 58
Imagenes de 7 hojas: 20
Imagenes de 8 hojas: 9
Imagenes de 9 hojas: 455
Total:  2545

1 hoja: Train=378 Validation=47 Test=48
2 hojas: Train=373 Validation=46 Test=47
3 hojas: Train=373 Validation=46 Test=47
4 hojas: Train=401 Validation=50 Test=50
5 hojas: Train=78 Validation=9 Test=10
6 hojas: Train=46 Validation=5 Test=7
7 hojas: Train=16 Validation=2 Test=2
8 hojas: Train=7 Validation=0 Test=2
9 hojas: Train=364 Validation=45 Test=46
```

Figura 5-23. Número de imágenes

Se ve como para los casos 5,6,7 y 8 se tendrá un comportamiento similar al visto anteriormente, sin embargo, se espera un comportamiento bueno para 9 hojas ya que si se tienen imágenes suficientes. En el siguiente apartado se hará la clasificación de las 9 categorías y se comparará con el resultado que se obtenía originalmente con las imágenes sin preprocesar.

5.4.6 Clasificiación entre 9 categorías

Tras entrenar la red con los datos dados en la figura 5-21 el resultado obtenido es el siguiente (fichero *CH_V_9hojas_cv.ipynb*).




```
Classification Report
              precision    recall  f1-score   support

           1       0.87      0.98      0.92        48
           2       0.85      0.85      0.85        47
           3       0.78      0.74      0.76        47
           4       0.79      0.92      0.85        50
           5       1.00      0.20      0.33        10
           6       1.00      0.43      0.60         7
           7       0.50      0.50      0.50         2
           8       0.00      0.00      0.00         2
          9+       0.96      0.98      0.97        46

    accuracy                           0.85       259
   macro avg       0.75      0.62      0.64       259
weighted avg       0.85      0.85      0.83       259

Test Accuracy: 0.84555983543396
```

Figura 5-24. Respuesta al entrenamiento. Imágenes procesadas.

Donde se ve que la precisión para los datos de prueba llega en general a un 85 por ciento, un 25 por ciento superior a la red con los datos sin preprocesar, visto en la sección 5.3.7, se aprecia como para los casos 5,6,7 y 8 los resultados no son muy buenos pues no se disponen de muchos datos de entrenamiento. Se puede observar también como el parámetro "precision" para las clases 5 y 6 es 1. Esto no quiere decir que la precisión sea de un cien por cien en estas clases si no que cómo se vio en el apartado 3.6 es el porcentaje de predicciones positivas correctas respecto al total de predicciones positivas (Se puede observar en la matriz de confusión). Debido a que estos datos son muy pequeños en comparación con los otros se van a eliminar y se procederá a una clasificación entre 1 hoja, 2 hojas, 3 hojas, 4 hojas y 9 hojas (fichero *CH_V_5hojasmix_cv.ipynb*). Obteniendo el siguiente resultado.

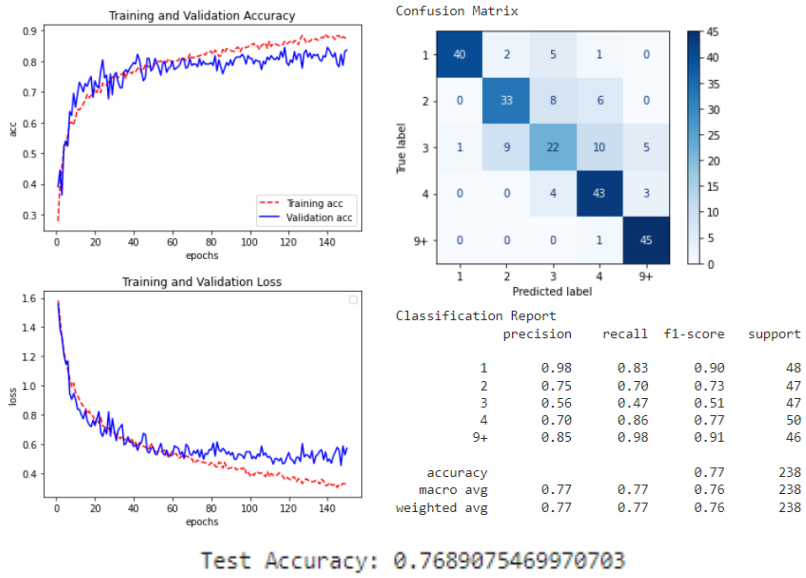


Figura 5-25. Respuesta al entrenamiento del modelo. Imágenes procesadas.

Obteniendo una precisión cercana al 80 por ciento. Se ha obtenido menor precisión que en la anterior a pesar de eliminar las categorías con menor número de hojas, pero esto es debido a que muestran una precisión engañosa estas categorías, ya que, al haber menos de diez hojas para las imágenes de test acertando solo unas pocas ya eleva la precisión muchísimo, sin embargo si nos fijamos en el parámetro *f1_score* se ve como para las clases eliminadas es muy alejado de 1. Se ha comprobado como para los mismos modelos al usar las imágenes preprocesadas el modelo aprende mucho mejor y clasifica con una precisión más elevada. Y también como es necesario una gran cantidad de imágenes para poder entrenar correctamente un modelo de red neuronal convolucional.

5.4.7 Resultados en Matlab

Para concluir los experimentos se va a contar el número de hojas usando un algoritmo en Matlab, las hojas a contar serán las preprocesadas con openCV (véase apartado 4). El algoritmo de Matlab consta de los siguientes pasos. En primer lugar se reescalan todas las imágenes a 100X150, más tarde se lee la imagen con el método *imread()* y se pasa a escala de grises y luego a binario con el método *imbinarize(),* es decir, el fondo permanecerá negro y la hoja de color blanco, acto seguido se elimina el ruido que haya podido generarse rellenando los huecos o *holes* con el *método imfill(img1,'holes').* El siguiente paso será etiquetar y contar los elementos conectados con *bwlabel(),* para finalizar, se elegirá una propiedad de los elementos a medir para detectar cuantos elementos que cumplen dichas propiedades hay en la imagen (método *regionprops()*), en este caso se ha elegido el área y se han buscado aquellos elementos con un área mayor de 100 pixeles para así evitar píxeles que representen ruido en la imagen.

```
% Cuenta las hojas de una planta

% Convierte la imagen a BMW
K = imresize(imread(strcat('C:\Users\ferga\Documents\MATLAB\TFG\2_mod\',archivo)),[100 150]);
img0=rgb2gray(imread('C:\Users\ferga\Documents\MATLAB\TFG\2_mod\1.png'));
img1 = imbinarize(img0);
imshow(img1)

% Rellenar regiones y agujeros de la imagen
img2 = imfill(img1,'holes');
imshow(img2)
% Etiquetar y contar los componentes conectados
[L,Ne] = bwlabel(double(img2));
% Medir las propiedades de la región de la imagen
prop = regionprops(L, 'Area');
% Contador
total = 0;
% Mostrar imagen
imshow(imread('C:\Users\ferga\Documents\MATLAB\TFG\2_mod\1.png'));hold on
% Cuenta
for n=1:size(prop,1)
    if prop(n).Area > 10
        total=total+1;
    else
        total=total+0;
    end
end
hold on
title(['Number of leaves: ',num2str(total)])
```

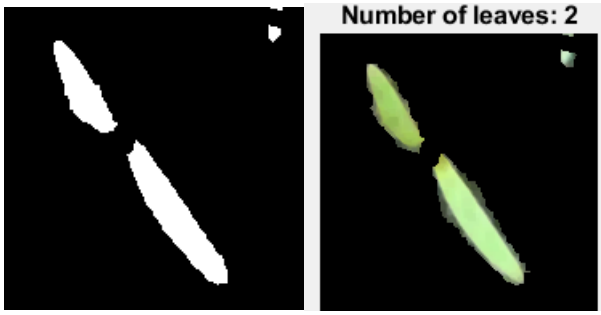


Figura 5-26. Respuesta al algoritmo en matlab.

Se ve como cuenta correctamente el número de hojas en la figura 5-23, pero ahora se va a calcular el porcentaje de acierto para todas las hojas de cada directorio. Primero se comenzará a contar las imágenes de 1 hoja y ver que precisión tiene. Para este cometido se ha seguido el fichero de Matlab *count1leave.m*.

```
>> Count1leaf
   86.1053
```

Figura 5-27. Precisión del fichero Count1leaf.m

Se ha obtenido una precisión del 86%, para entender las imágenes que no está detectando se va a a ver el siguiente ejemplo.

Figura 5-28. Caso de error del algoritmo y su corrección.

Donde se ve como no detecta que hay una hoja. Esto es debido al área mínima que se está utilizando para detectar hojas, es decir, la hoja de la figura 5-30 tiene menos de 200 píxeles de área y aunque se haya hecho el rescalado no la está detectando pero no es problema del algoritmo que se está siguiendo ya que si se reajusta el área a detectar para esta imagen en concreto se ve en la segunda imagen de la figura 5-30 como si la detecta. Es decir, el algoritmo funciona pero la base de datos con la que se cuenta contiene hojas de todos los tamaños y formas que hace difícil ajustar el algoritmo para que detecte todos esos detalles.

Al probar con las imágenes de 2 hojas (Fichero *count2leaves.m*).

```
>> Count2leaves
   46.5665
```

Figura 5-29. Precisión del fichero count2leaves.m.

Se ve como la precisión es mucho menor que la obtenida con las redes neuronales, esto es debido a que no detecta que hay dos elementos, debido a que la erosión no es perfecta y hay bastantes hojas que no están separadas claramente, si se aumenta la erosión muchas imágenes quedarían demasiado erosionadas y desaparecerían hojas de la imagen, habría que hacer una erosión individual por cada imagen de la base de datos para dejar todas las hojas de todas las imágenes claramente separadas, esto se escapa del objetivo de este proyecto. Nuevamente se ve la dificultad que tiene esta base de datos para su clasificación.

Figura 5-30. Muestra una planta con dos hojas clasificada como si fuese 1 hoja ya que no se encuentran sus hojas separadas.

Se ha visto como el problema no es el algoritmo, sino la dificultad de la clasificación para toda la base de datos. Para 3 hojas se tiene la siguiente salida.

```
>> Count3leaves
   29.8283
```

Figura 5-31. Precisión del fichero count3leaves.m

Para 4 hojas.

```
>> Count4leaves
   20.5589
```

Figura 5-32. Precisión del fichero count4leaves.m

El porcentaje de acierto baja por cada categoría añadida, viéndolo para las imágenes de 5 hojas.

```
>> Count5leaves
    4.2918
```

Figura 5-33. Precisión del fichero count5leaves.m

El porcentaje de acierto se reduce a menos del 5%, y el área buscada es menos de 10 píxeles, por lo que al aumentar de categoría es más difícil es para el algoritmo detectar las hojas y al haber más ruido en las imágenes el algoritmo lo confunde con hojas.

El algoritmo visto funciona a la perfección, pero para imágenes bien definidas donde se conoce el Área o las propiedades de los elementos que se desean contar, sin embargo, la base de datos contiene imágenes muy diversas de plantas de todos los tamaños y formas lo que hace realmente difícil ajustar los parámetros del modelo para obtener una buena precisión.

La red neuronal sin embargo puede aprender más características generales y consigue un porcentaje de acierto muy superior para esta base de datos.

6 CONCLUSIONES Y LÍNEAS FUTURAS

En el experimento que se ha seguido se ha creado una red neuronal convolucional para clasificación de imágenes, más concretamente, para imágenes de plantas con su número de hojas contadas. La red neuronal convolucional creada demostró la dificultad que tenía esta tarea ya que las imágenes eran realmente difíciles de clasificar. Primero se vio cómo el modelo creado era incapaz de clasificar bien las imágenes obteniendo un 55% de precisión para los datos de prueba y viendo la matriz de confusión se vio como solo clasificaba correctamente aquellas categorías que contaban con miles de imágenes (sección 5.2.7). Después se creó un segundo modelo el cual contaba con el mismo número de imágenes para cada clase, viendo los resultados se vio como seguía siendo bastante mejorable la precisión obteniendo de nuevo un 55% para los datos de prueba (sección 5.2.8). Es por ello por lo que se creó un tercer modelo, usando la técnica de transfer learning, el cual usaba la compleja red neuronal *inceptionV3* ya preentrenada con las imágenes de *imagenet,* pero este modelo tampoco mejoraba los resultados a la hora de clasificar, obteniendo de nuevo un porcentaje de acierto menor a un 60% (sección 5.2.9).

Es por ello por lo que se opta en este proyecto por hacer experimentos, empezando por purgar la base de datos que contiene imágenes realmente difíciles de clasificar incluso para el ojo humano, ya sea por que aparecen malas hierbas o porque no está nada claro su número de hojas al ser cada una de diferente tamaño y forma (sección 5.3.1). Acto seguido se preprocesaron las imágenes, eliminando todo aquello que no era información importante, es decir, las hojas y dejando las hojas aisladas sobre un fondo negro. La otra modificación fue la erosión para dejar separadas las hojas de aquellas plantas que contaban con sus hojas muy pegadas entre si dificultando también la clasificación (sección 5.3.1). Se propuso generar modelos progresivos empezando por clasificar 2 clases (1 hoja y 2 hojas) y comparándolo con el modelo que usa los datos sin preprocesar. Se ve como la precisión alcanzada por el modelo que usa los datos preprocesados es superior a el que usa los datos sin preprocesar y se demuestra como sigue el mismo patrón al añadir las siguientes categorías (secciones 5.3.2 a 5.3.4). Se tuvieron que eliminar las categorías 5,6,7 y 8 ya que cuentan con demasiadas pocas imágenes realmente aprovechables para hacer que el modelo aprenda, sin embargo, si se añadió la categoría de 9 hojas o más y se sigue comprobando como los experimentos funcionan obteniendo una precisión aceptable y en la matriz de confusión se puede apreciar como la clasificación está siendo correcta (secciones 5.3.5 a 5.3.6).

Por último, se propuso un algoritmo de Matlab para comprobar si era necesario crear una red neuronal con la complejidad que lleva el proceso en vez de clasificarlas por un algoritmo de Matlab más sencillo. El agoritmo demuestra su buen funcionamiento detectando el número de hojas de cualquier imagen si se ajustan bien sus parámetros. La dificultad llega cuando se intenta ver la precisión de acierto para toda una categoría donde cada imagen es diferente a la otra con diferentes tamaños de hojas y formas lo que hace realmente difícil hacer un buen ajuste de los parámetros para que detecte las hojas en todas las imágenes. Esto hace que mientras más hojas haya en la imagen la erosión es peor y hay más ruido por lo que el algoritmo se vuelve prácticamente inservible para tan complejas fotos. Es por ello por lo que en este caso parece esencial el uso de la red neuronal convolucional con el preprocesado de las imágenes que obtiene un más que aceptable porcentaje de acierto para tan pocas imágenes de entrenamiento con las que se cuenta.

Como línea futura de investigación se propone buscar métodos más complejos sobre clasificación o preprocesado de imágenes que puedan ayudar a subir la precisión, sobre todo para aquellas clases que cuentan con un centenar o incluso menos de un centenar de imágenes. O incluso buscar modelos o métodos innovadores de Deep learning que puedan tratar mejor esta base de datos que contiene imágenes nada triviales para su clasificación.

REFERENCIAS

[1] L. Rouhiainen, Inteligencia Artifical, Alienta.

[2] F. Berzal, Inteligencia Artificial, Universidad de Granada.

[3] M. A. Boden, Inteligencia artificial., Turner, 2017.

[4] J. L. Espinoza, "BBVA. Machine Learning y como funciona." [En línea]. Available: https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/.

[5] T. M. &. M. T. M. Mitchell, Machine learning, New York: McGraw-hill, 1997.

[6] J. Torres, Deep Learning: Introduccion a la practica con Keras, Independiente, 2018.

[7] R. Arrabales, "Xataka" Marzo 2016. [En línea]. Available: https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial. [Último acceso: Junio 2022].

[8] F. Ramírez, Historia de la IA: Frank Rosenblatt y el Mark I Perceptrón, el primer ordenador fabricado específicamente para crear redes neuronales en 1957, 2018.

[9] S. Haykin, Neural Networks: A Comprehensive Foundation (2 edición), Prentice Hall, 1998.

[10] L. Yang y A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, 2020.

[11] Y. Cruz, M. Rivas, R. Quiza, A. Villalonga, R. Haber y G. Beruvides, Ensemble of convolutional neural networks based on an evolutionary algorithm applied to an industrial welding process, 2021.

[12] R. Holbrook, "Kaggle" [En línea]. Available: https://www.kaggle.com/learn/computer-vision.

[13] D. Calvo, "DiegoCalvo" 2017. [En línea]. Available: https://www.diegocalvo.es/red-neuronal-convolucional/.

[14] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella y J. Schmidhuber, Flexible, High Performance Convolutional Neural Networks for Image Classification, 2011.

[15] A. Perera, "ayeshmanthaperera" Septiembre 2018. [En línea]. Available: https://ayeshmanthaperera.medium.com/what-is-padding-in-cnns-71b21fb0dd7. [Último acceso: Junio 2022].

[16] W. Koehrsen, Overfitting vs. underfitting: A complete example, Towards Data Science, 2018.

[17] "Aprende Machine Learning" 2017. [En línea]. Available: https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/. [Último acceso: Junio 2022].

[18] C. Shorten y T. M. Khoshgoftaar, A survey on Image Data Augmentation for Deep Learning, 2019.

[19] L. &. S. J. Torrey, Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, 2010.

[20] J. Torres, Introducción a la práctica con Keras 2 parte, Independiente, 2020.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, Dropout: Una forma fácil de prever el sobreajuste en redes neuronales, University of Toronto, 2015.

[22] "OpenCV" [En línea]. Available: https://opencv.org/.

[23] "Open CV. Changing Colorspace" [En línea]. Available: https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html.

[24] "OpenCV. Morphological Transformations" [En línea]. Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

[25] "Keras" [En línea]. Available: https://keras.io/. [Último acceso: 2022].

[26] "Jupyter Notebook" [En línea]. Available: https://jupyter.org/. [Último acceso: 2022].

[27] "cuDNN" [En línea]. Available: https://developer.nvidia.com/cudnn. [Último acceso: 2022].

[28] "Github" [En línea]. Available: https://github.com/.

[29] M. Dyrmann, "Leaf Counting Dataset" 2018. [En línea]. Available: https://vision.eng.au.dk/leaf-counting-dataset/.

[30] N. D. M. N. P. M. S. S. G. &. J. R. Teimouri, Weed Growth Stage Estimator Using Deep Convolutional Neural Networks., 2018.

[31] M. P, "Data Scientist" 13 Mayo 2022. [En línea]. Available: https://datascientest.com/es/cross-validation-definicion-e-importancia#:~:text=La%20t%C3%A9cnica%20del%20Train%2DTest%20Split&text=Una%20parte%20servir%C3%A1%20para%20el,explotar%C3%A1%20en%20la%20cross%2Dvalidation..

[32] J. Brownlee, Probability for Machine Learning, Machine Learning Mastery, 2019.

[33] N. S. &. S. R. Keskar, Improving generalization performance by switching from adam to sgd, 2017.

ANEXO

Como se comentó en la sección 5.1 todos los códigos y ficheros mencionados en esta memoria se pueden ver online a través del siguiente enlace, también se pueden dejar dudas, sugerencias de mejora o si hay algún error para acceder a ellos.

- https://github.com/fgfcad/TFGfgf