Universidad Europea

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN

ANÁLISIS DE DATOS MASIVOS (BIG DATA)

TRABAJO FIN DE MÁSTER

CLASIFICACIÓN DE IMÁGENES DE HOJAS DE MAÍZ CON ENFERMEMDADES PROVOCADAS POR HONGOS

NOMBRE:

ASIER ORTE NIETO

CURSO 2021-2022

TÍTULO: CLASIFICACIÓN DE IMÁGENES DE HOJAS DE MAÍZ CON ENFERMEMDADES PROVOCADAS POR HONGOS

AUTOR: ASIER ORTE NIETO

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

DIRECTOR DEL PROYECTO: ALFONSO ANTOLINEZ

FECHA: OCTUBRE de 2022

RESUMEN

El objeto del presente proyecto es crear un *dataset* de imágenes con algunos de los hongos que afectan a hojas del maíz, y mediante una técnica de *Machine Learning* conocida como *Trasfer Learning* y Redes Neuronales Convolucionales ser capaz de clasificar correctamente las imágenes con su clase correspondiente.

Para la construcción del *dataset* de este proyecto se han descargado imágenes a través de páginas web con licencia de libre uso para crear y etiquetar cada una de las clases. Para poder comparar los diferentes modelos de redes implementados se han propuesto interfaces visuales muy simples en formato de tabla.

Al no tener un *dataset* muy amplio las redes neuronales que mejor funcionan son las que tienen una arquitectura más simple, gracias al *Trasfer Learning* y aprovecharse de redes entrenadas como muchas imágenes, se observa que los resultados mejoran considerablemente sobre todo después de realizar el *fine-tuning*.

Palabras clave: *Machine Learning*, Redes Neuronales Convolucionales, *Transfer Learning*, Fine-tuning

ABSTRACT

The purpose of this project is to create a *dataset* of images with some of the fungi that affect corn leaves, and by means of a *Machine Learning* technique known as *Transfer Learning* and Convolutional Neural Networks to be able to correctly classify the images with their corresponding class.

For the construction of the *dataset* of this project, images have been downloaded through web pages with free use license to create and label each of the classes. In order to compare the different network models implemented, very simple visual interfaces in table format have been proposed.

Not having a very large *dataset*, the neural networks that work best are those with a simpler architecture, thanks to *Transfer Learning* and taking advantage of *train*ed networks as many images, it is observed that the results improve considerably, especially after performing the fine-tuning.

Key words: *Machine Learning*, Redes Neuronales Convolucionales, *Transfer Learning*, Fine-tuning

AGRADECIMIENTOS

Agradecer en primer lugar a mi familia por apoyarme en todos los pasos que he dado siempre en mi vida tanto a nivel personal como profesional, a todos mis amigos y a todos mis compañeros de máster por este año compartiendo conocimiento.

Índice

RESUMEN 5
ABSTRACT 6
Capítulo 1. INTRODUCCIÓN 11
1.1 Planteamiento del problema 11
1.2 Objetivos del proyecto 11
1.3 Estructura del proyecto 12
Capítulo 2. ANTECEDENTES 15
Capítulo 3. FUNDAMENTOS TEORICOS 17
3.1 Redes Neuronales Artificiales 18
3.2 Redes Neuronales Convolucionales 21
3.3 *Transfer Learning* 23
3.4 Algoritmos 25
Capítulo 4. TECNOLOGIAS 31
4.1 Entorno de trabajo 31
4.2 Clasificación 31
4.3 Creación de tablas 32
Capítulo 5. DESARROLLO DEL TRABAJO 33
5.1 Obtención y creación del *dataset* 33
5.2 Construcción de las redes neuronales 33
5.2.1 Preparación de los datos y Data augmentation 34
5.2.2 Creación del modelo base 35
5.2.3 Compilar y entrenar el modelo 36
5.2.4 Fine Tuning 37
5.3 Resultados obtenidos 38
Capítulo 6. CONCLUSIONES 41
Capítulo 7. FUTURAS LÍNEAS DE TRABAJO 43
ANEXOS ..... **¡Error! Marcador no definido.**
BIBLIOGRAFÍA 44

Índice de Figuras

Imagen 1. Esquema de una Red Neuronal Simple 18
Imagen 2. Esquema de una Red Neuronal con explicación de los pesos sobre una neurona 18
Imagen 3. Función sigmoide 19
Imagen 4. Función tangente 19
Imagen 5. Función ReLU 20
Imagen 6. Esquema básico del perceptrón multicapa 20
Imagen 7. Esquema de una CNN 22
Imagen 8. Funcionamiento del *Transfer Learning* 24
Imagen 9. Comparación de MobileNet, GoogleNEt y VGG16 26
Imagen 10. Arquitectura de la red Inception 27
Imagen 11. Calculo convolucional de las redes Xception 28
Imagen 12. Arquitectura VGG16 29
Imagen 13. Arquitectura y funcionamiento de ResNet50 30

Índice de Tablas

Tabla 1. Resultados de loss y accuracy de Mobilenet 38
Tabla 2.Resultados loss y accuracy de ResNet50 38
Tabla 3. Resultados loss y accuracy de VGG16 38
Tabla 4. Resultados loss y accuracy de VGG19 39
Tabla 5. Resultados loss y accuracy de Inception 39
Tabla 6. Resultados loss y accuracy de Xception 39
Tabla 7. Matriz de confusión de MOBILENET 39
Tabla 8. Matriz de confusión de RESNET50 40
Tabla 9. Matriz de confusión de VGG16 40
Tabla 10. Matriz de confusión de VGG19 40
Tabla 11. Matriz de confusión de INCEPTION 40
Tabla 12. Matriz de confusión de XCPETION 40
Tabla 13. Tabla de resultados de la función de perdida Sparse Categorical Cross Entropy y la metrica accuracy, medias de todas las ejecuciones realizadas para cada uno de los modelos 42

Capítulo 1. INTRODUCCIÓN

En este primer capítulo se realizará una introducción al problema planteado, asi como los objetivos definidos y la estructura de este.

1.1 Planteamiento del problema

En pleno siglo XXI se generan datos continuamente, sobre todo, gracias al auge de las redes sociales, contenido multimedia como pueden ser imágenes y videos. Esto a priori no parece que pueda aportar mucha información, pero para un ordenador, que es capaz de analizar y extraer una gran cantidad de datos, es una tarea complicada que requiere de muchos recursos, nos permite ampliar nuestras fronteras del conocimiento y convertir a cualquier usuario en “experto” de cualquier ámbito.

Con el crecimiento de las nuevas tecnologías y la inteligencia artificial, en concreto en el campo del *Machine Learning*, se encuentra el procesamiento de imágenes tanto a tiempo real, que requiere de una respuesta rápida, como por ejemplo podría ser detectar objetos o la conducción autónoma, como la clasificación de imágenes, que requieren otro tipo de algoritmos que no necesitan una respuesta rápida y que le permiten realizar cálculos más avanzados y costos con el uso de GPU.

Para este segundo grupo de algoritmos, como se ha comentado anteriormente, es necesario el uso de mucha potencia de GPU del que disponen muy pocas empresas en el mundo, al intentar aterrizar este tipo de algoritmos para trabajos o experimentos del día a día, lo más común es no entrenar el algoritmo desde 0, sino utilizar algoritmos pre-entrenados a los que se le indican las clases en las que clasificar cada una de las imágenes y de esta manera poder hacer uso de los algoritmos a un coste mucho menor.

Para ello este trabajo se centra en esta funcionalidad de algoritmos pre-entrenados denominado como *Transfer Learning.* Se realizará un estudio de estos algoritmos y analizar no solo como funcionan, sino que resultados dan comparando la *accuracy* o precisión de cada uno de ellos.

1.2 Objetivos del proyecto

En este apartado se mencionan los objetivos principales de este proyecto:

- Generar un *dataset* etiquetando distintas imágenes de maíces con enfermedades generadas por hongos.

- Para la construcción del *dataset* se emplearán algoritmos para extracción de imágenes a partir de videos y aumento de *dataset*.

- Implementar un sistema de clasificación modular basado en técnicas de aprendizaje automático, cuyo objetivo será clasificar cualquier imagen de hojas de maíz con alguna de las enfermedades etiquetadas o si está sano.

Para que los algoritmos de *Machine Learning* sean lo más precisos posibles se ha de tener una colección de imágenes amplia, lo cual es complicado al tratarse de un tema muy concreto y que no esta tan extendido en el día a día, como podría ser otro tipo de clasificación, está es la mayor dificultad que se ha encontrado a lo largo del desarrollo del proyecto.

1.3 Estructura del proyecto

En este apartado se hablará sobre la estructura que se ha seguido para llevarlo a cabo.

Lo primero que se realizó tras hablar con el tutor sobre el tema y definir los objetivos de este, se realizó un trabajo de investigación para conocer el estado del arte, trabajos similares y adquirir conocimiento teórico-práctico, acerca de las redes neuronales convolucionales y el *Transfer Learning* aplicado a clasificación de imágenes.

Tras el trabajo de investigación, se llevó a cabo la tarea de crear un *dataset* lo suficientemente robusto como para que los distintos modelos que se iban a construir no tuvieran problemas ni de *underfitting* ni de *overfitting*, en esta tarea se empleó mucho tiempo ya que las imágenes de hojas de maíz con enfermedades no es un tema muy común lo que implica que no sea sencillo localizar una gran cantidad de imágenes de libre uso, finalmente se encontró un *dataset* construido con aproximadamente unas 4000 imágenes en la web más conocida sobre el *Machine Learning* y la comunidad de Científicos de datos, *Kaggle*.

Una vez obtenido el *dataset*, se comenzó a trabajar con las redes neuronales, empezando por un ejemplo base muy simple que *TensorFlow* junto con *Keras* facilita a los usuarios que quieren iniciarse en el este mundo, con los conocimientos adquiridos en el ejemplo base se comienza a programar las distintas redes neuronales (*MobileNet, ResNet, VGG, Inception y Xception*).

Se obtienen resultados de distintas ejecuciones y se calculan las medias de cada una de ellas para estudiar y comparar cuál de las redes ha sido la que mejor ha funcionado,

dibujando a su vez una matriz de confusión para cada uno de los modelos con el fin de ayudar en el estudio de la red más precisa para este problema en concreto.

Las siguientes páginas reflejan con mayor detalle los conocimientos y el proceso de aprendizaje realizado a lo largo de estos meses.

Capítulo 2. ANTECEDENTES

Para empezar a investigar el trabajo se comenzó a estudiar el problema leyendo proyectos similares.

En el mismo contexto de este proyecto de clasificación de imágenes con algoritmos de ML, se han encontrado algunos trabajos con un objetivo similar al de este mismo:

1. **Maize Leaf Disease Detection and Classification Using *Machine Learning* Algorithms** [1]. Trabajo que utiliza técnicas de aprendizaje supervisado para detectar y clasificar distintas enfermedades sobre imágenes, técnicas como *Naive Bayes, Decisión Tree, KNN, Support Vector Machine y Random Forest*. Para realizar una comparación de las precisiones obtenidas por los distintos modelos, siendo el *Random Forest* el modelo con el mejor resultado, obteniendo un 79% de *accuracy*.

2. **Deep Learning and *Transfer Learning* Approaches for Image Classification** [2]**.** Explica distintas arquitecturas de redes neuronales como pueden ser *GoogleNet, VGG* y *Resnet* entre otras, y como utilizar el *Transfer Learning* mediante el uso de Redes Neuronales Convolucionales pre-entrenadas, asi como un *test*eo mediante *dataset*s de *Image Net*[1].

3. ***Machine Learning*-based for Automatic Detection of Plant Diseases Using Image Processing** [3]. Este trabajo se centra en explicar cuál es la característica más importante a la hora de realizar este tipo de clasificaciones de procesamiento de imágenes con *Machine Learning*, para evaluar el rendimiento de las distintitas características mediante técnicas de aprendizaje automático como: *SVM, Decision Tree, Random Forest y Naive Bayes*. El trabajo concluye que el color es la característica con la precisión más alta para la mayoría de los clasificadores.

4. ***Machine Learning* approach for the classification of corn seed using hybrid features** [4]. Este estudio trata de examinar la facilidad del Aprendizaje Automático de clasificar diferentes semillas del maíz, mediante técnicas de clasificación como: *Random Forest, BayesNeet, LogitBoodt y MLP*. Resultando este último como mejor clasificador, obteniendo una precisión del 98,9%.

5. **You Only Look Once: Unified, Real-Time Object Detection** [5]**.** En otro ámbito no tan similar como los proyectos anteriores como los anteriores nos encontramos proyectos como la detección de objetos en tiempo real, que mediante el uso de redes convolucionales permite predecir probabilidades de cada una de las clases a tiempo real evaluando hasta 45 fotogramas por segundo (más fotogramas en modelos más avanzados). El estudio determina que tiene más errores de localización, pero tiende menos a falsos positivos.

[1] Conjunto de datos que se utiliza habitualmente para evaluar algoritmos de clasificación.

6. **Simple convolutional neural network on image classification** [6]**.** Este estudio analiza como clasifica imágenes una red neuronal artificial, para confirmar que este tipo de redes funcionan realmente bien con la clasificación imágenes, va un paso más allá y analiza tambien distintas *Learning Rate*[2] y tratando de obtener los parámetros óptimos.

7. **Medical image classification with convolutional neural network** [7]**.** En este trabajo diseñan y crean su propia red neuronal para clasificar imágenes pulmonares con ILD[3]. La misma arquitectura que presentan es posible utilizarse para llevar a cabo clasificaciones de otras imágenes médicas.

8. **Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, *Dataset* Characteristics and *Transfer Learning*** [8]**.** En este texto estudian tres factores para emplear el Deep CNN para problemas de detección asistida por ordenador. Primero exploran diferentes arquitecturas CNN. Para después evaluar cómo afecta la escala del *dataset* y el contexto espacial de la imagen al rendimiento, para finalmente examinar cuando y como es útil el *Transfer Learning*.

Por último, se realizó la búsqueda de algún trabajo que de alguna manera combinase todas las propiedades de los trabajos anteriormente mencionados; clasificar mediante redes neuronales imágenes de enfermedades de las hojas del maíz, que además hiciera uso del *Transfer Learning* para tener redes neuronales pre-entrenadas que nos puedan dar una precisión mayor, sin necesidad de invertir mucho tiempo en entrenar la red, teniendo en cuenta tambien que el tamaño del *dataset* no es muy amplio. Sin embargo, no se ha tenido éxito en la búsqueda de un proyecto que englobase todas las características que se van a presentar en las siguientes páginas.

[2] Valor constante del algoritmo Descenso de Gradiente Estocástico, que permite minimizar la pérdida de un modelo.
[3] Enfermedad pulmonar intersticial difusa.

Capítulo 3. FUNDAMENTOS TEÓRICOS

Antes de explicar nada se debe tener claro que este trabajo se sitúa en el campo del *Machine Learning*. El *Machine Learning* o Aprendizaje Automático es la capacidad de los ordenadores de aprender tareas en base a ejemplos y casos. Esto viene muy bien cuando no se sabe muy bien cómo explicar algo, como por ejemplo podría ser explicar que es una cara, podría decirse que una cara se compone por dos ojos, una nariz, una boca y un par de orejas. Pero que es un ojo exactamente, hay muchas formas y colores, tamaños, formas y posiciones de ojo. Esto matemáticamente sería muy difícil y costoso de definir, para eso existe el *Machine Learning*, que tiene muchísimos algoritmos distintos en el que cada uno sirve para una tarea distinta.

Principalmente se encuentran dos tipos de problemas: los que se basan en palabras, tambien conocidos como **NLP** (*Natural Lenguaje Proccessing*) o PLN (Procesamiento del Lenguaje Natural) en castellano, dentro de este grupo de problemas encontrar algunos como análisis de sentimiento, por ejemplo, en base a comentarios de redes sociales o análisis morfosintáctico para poder realizar un resumen o inducir el tema sobre el que está hablando en un texto.

Por otro lado, existe otro tipo de problema muy distinto como es el **tratamiento de imágenes**, aquí ya no se hace uso de palabras sino de imágenes, que se componen de pixeles, a los que a su vez son números que representan un color, en caso de que este en color y sea RGB[4], por lo que en este caso se compondrá de 3 números entre el 0 y 255 para representar los 3 colores primarios, rojo, verde y azul, con los que con componer cada uno de los pixeles de la imagen. A priori parece imposible que un ordenador con el simple hecho de conocer estos números se capaz de detectar objetos en una imagen, por ejemplo, pero existen técnicas que funcionan muy bien, para lo que comúnmente se utilizan las redes neuronales convolucionales.

[4] Composición de color en términos de intensidad de la luz de los colores primarios

3.1 Redes Neuronales Artificiales

Las redes neuronales artificiales son un tipo de **red neuronal** que, simplificando mucho y no entrando en detalles de la neurobiología, está inspirado en el funcionamiento de las neuronas del cerebro humano. El modelo matemático que utilizan estas redes artificiales están compuestas por una neurona "y" que recibe valores numéricos, cada una de estas entradas numéricas tendrá un valor "w" que representa el peso dentro de la neurona "y". Tal y como se puede ver en la siguiente imagen, la neurona "y" recibe valores de las neuronas "x_i" del paso inmediatamente anterior, cada una con su peso "w" correspondiente.

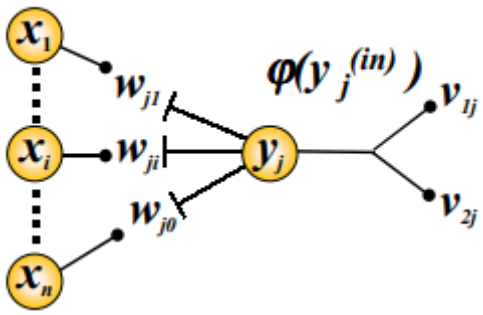

Imagen 1. Esquema de una Red Neuronal Simple

Lo que hace cada uno de los pesos es multiplicar a la entrada con la que se corresponde, para dar un valor más o menos importante a cada una de las entradas. Por lo que la arquitectura final completa se parecería más a la siguiente imagen.

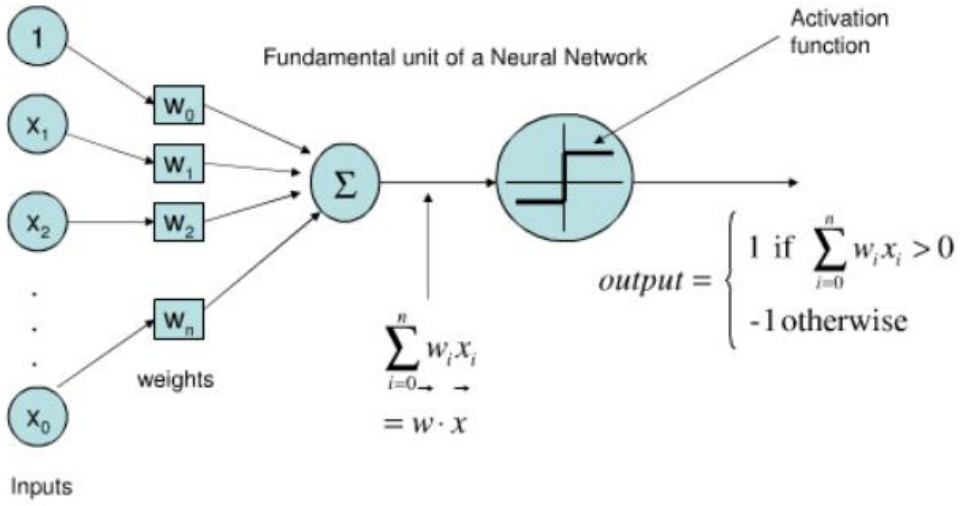

Imagen 2. Esquema de una Red Neuronal con explicación de los pesos sobre una neurona

Como se puede ver el peso "w" se multiplica con el valor de cada una de las entradas "x" y se suma para generar un resultado, en la red neuronal artificial más sencilla,

conocida como perceptrón simple, el resultado es lineal (aunque se pueden usar algunos métodos para añadir la no-linealidad), lo que significa que se puede representar el modelo como otro vector de longitud n, con los pesos w.

Como se puede ver en la imagen (Imagen 2.) se calcula la función de activación, pues como ya se ha indicado se puede conseguir esa no linealidad, de hecho, es importante evitar que la red sea solamente una transformación lineal, a continuación, se comentan algunas de las funciones de activación no lineales más comunes:

- Sigmoide: es la función que se utiliza en la regresión logística, que suaviza el paso por la función, de forma que se obtiene un gradiente en el área cercana a x=0, ya que esta función tiende a una asíntota horizontal.

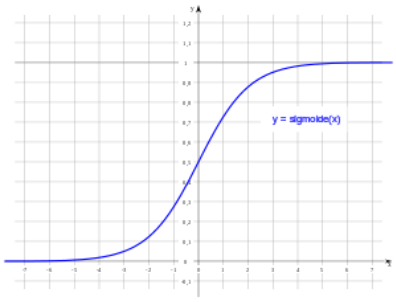

Imagen 3. Función sigmoide

- Tanh: Esta función a diferencia de la sigmoide tiene su media en 0 y no en 0,5 ya que se mueve en el intervalo [-1,1], gracias a que la media es 0 se obtienen mejores resultados.

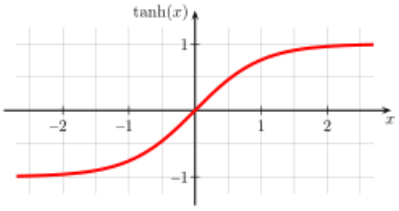

Imagen 4. Función tangente

- ReLU: Rectified Linear Unit. Es la función más común y la que hasta ahora está demostrada que funciona mejor. Su función es: $f(x)=\ln(1+e^x)$

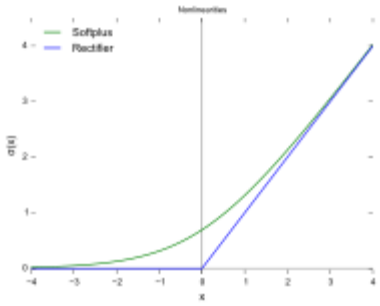

Imagen 5. Función ReLU

Existen modelos neuronales más complejos que el **perceptrón simple**, como el **perceptrón multicapa**. Se fundamenta en la idea del perceptrón simple, pero con una estructura más compleja combinando neuronas artificiales. Se introduce el concepto de capas ocultas, donde no sabremos qué está pasando exactamente, por lo que no podremos dar una explicación del porqué del resultado del modelo entrenado, sino que solamente es posible dar un resultado gracias a una neurona final de activación, el esquema es similar a la siguiente imagen.

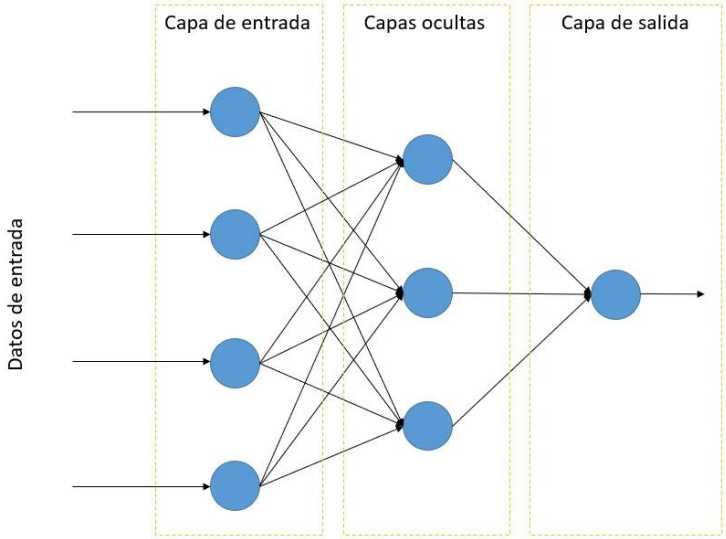

Imagen 6. Esquema básico del perceptrón multicapa

Al igual que en el perceptrón simple cada una de las neuronas tendrá un peso asignado, todos los pesos se combinan para construir una matriz de pesos “W”, en este caso la matriz será de 4x3, 4 por las capas de entrada y 3 por la capa oculta.

Cada una de las neuronas no tiene solamente un peso asignado, así como un *bias*, de modo que es capaz de resolver la operación:

$$\sum_i x_i w_i + b$$

Ecuación 1. Operación que realiza cada neurona para calcular su salida.

Para ajustar los pesos de cada una de las neuronas se utiliza el algoritmo de *backpropagation* o regla de la cadena. De modo que utilizando el descenso por gradiente se optimizan todos los parámetros que hay que utilizar.

3.2 Redes Neuronales Convolucionales

Como ya se ha mencionado anteriormente las redes convolucionales se utilizan para problemas de *Computer Visión*, principalmente reconocimiento de objetos y clasificación de imágenes.

En la clasificación de imágenes existen dificultades varias con las que lidiar para conseguir un buen resultado:

- Brillos
- Variación geométrica (cambios de escala, posición…)
- Deformaciones
- Alta variación *intra-clase*

Una red neuronal convolucional o **CNN** (*Convolutional Neural Network*, en inglés) es una red específicamente adaptada para problemas de *Computer Visión*, que son capaces de manejar entradas de muy altas dimensiones, soportan imágenes multicanal y video entre otras. El esquema general de las CNN es el siguiente:

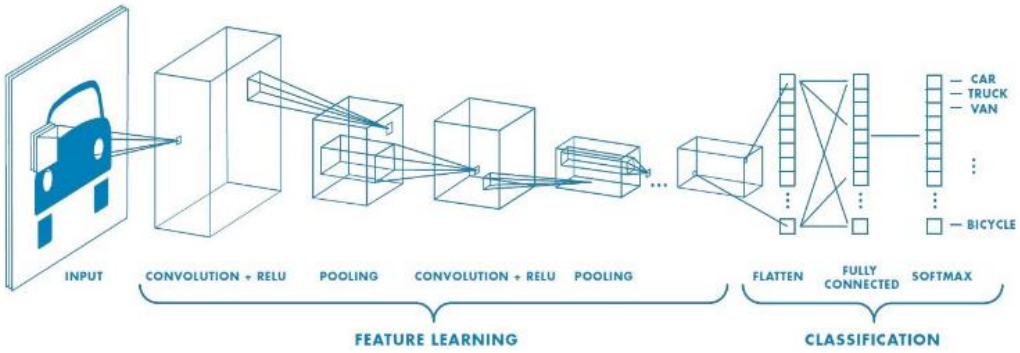

Imagen 7. Esquema de una CNN

Fundamentos de las CNN:

- **Distribución espacial**: cada neurona solamente mira a su campo receptor, es decir, solamente tiene en cuenta una pequeña parte de la imagen y no toda al completo.
- **Capa convolucional**: se trata de un conjunto de mapas de activación, que contiene un vector de entrada como una imagen por ejemplo que utiliza para detectar características, y vectores de salida que se utilizan para crear un mapa de características, es decir, es capaz de extraer características y patrones de la imagen, como se ha indicado solamente extrae información de su receptive field.
- **Convolución**: la convolución es una operación matemática entre matrices, que consiste en multiplicar matrices para dar una nueva como salida, su uso más común es aplicar filtros a una imagen como cambiar el color a blanco y negro, por ejemplo.
- **Pooling**: el *pooling* sirve para reducir las dimensiones de una imagen, de alguna manera resume las estadísticas por grupos. Existen distintos métodos de pooling como: máximo, media y media ponderada. Cada uno de estos métodos hace un cálculo de una pequeña parte de la matriz para reducir el tamaño de esta.

Construcción de una CNN

La estructura que se ha enseñado anteriormente es la típica de una Red Convolucional y suele ser la siguiente:

- **Input layer**: da la imagen de entrada, que a nivel muy bajo son pixeles constituidos por números ya sea en formato RGB o escala de grises entre otros
- **Convolutional layer**: como se ha comentado anteriormente es la capa donde entran en juego los mapas de activación para dar una salida con las características que tenga la imagen. Su entrada puede ser bien una imagen, bien la salida de otra capa convolucional.
- **ReLU**: Aplica la no-linealidad a las funciones de activación de las neuronas de las capas convolucionales.

- **Pooling** (n-veces): se reduce la dimensionalidad de la matriz resultado de las capas convolucionales anteriores. Se aplica tantas veces como sea necesario, este número de repeticiones suele ser un parámetro que explorar para mejorar el resultado del modelo.
- **Fully Connected Layers**: "aplana" el vector para poder realizar una clasificación con redes neuronales estándares.
- **SoftMax**: por último, se encuentra la capa final para calcular la salida, en la que se calcula la perdida.
- **Hiperparámetros**: concepto extra que forma parte de la optimización del modelo, para que este de unos resultados lo más precisos posibles con respecto al problema que se está resolviendo. Nos encontramos atributos tanto en la capa convolucional:
  - **Stride**: paso de la convolución.
  - **Padding**: añadir bordes de pixeles vacíos a la imagen.
  - Tamaño del **kernel**: tamaño de los filtros a aplicar.
  - **Filtros**: número de filtros.

  Como en la capa de pooling:
  - Tamaño del pool: tamaño para reducir las dimensiones.
  - Stride.

3.3 *Transfer Learning*

Cuando no se disponen de *datasets* amplios es posible acudir al *Transfer Learning*, ya que de otra manera lo único que se conseguiría seria que la CNN haga *overfitting*[5] (el modelo se aprenda los datos de entrada de memoria y no pueda dar unos resultados fiables).

Estos modelos suelen necesitar una capacidad de recursos hardware muy potente para entrenarlos, por lo que en el contexto de este trabajo acudiremos a esta técnica para poder obtener unos resultados fiables sin necesidad de tener un *dataset* muy amplio.

¿Qué es el *Transfer Learning*?

El *Transfer Learning* consiste en entrenar una red con un conjunto de datos muy grande, una vez entrenado se transfiere el aprendizaje y se vuelve a entrenar la misma red, pero con el *dataset* reducido del que se dispone, tal y como se puede observar en la siguiente imagen.

[5] Sobre ajuste que realiza el modelo, haciéndolo un modelo malo para predecir.

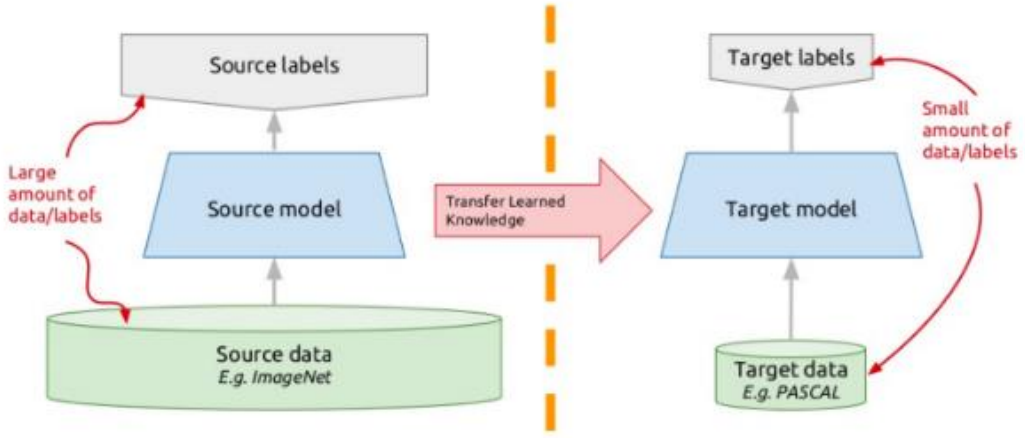

Imagen 8. Funcionamiento del Transfer Learning

Principalmente se utilizan redes pre-entrenadas para que se puedan usar en problemas más pequeños, justo lo que se va a realizar en el TFM. Para hacer uso de estas redes ya entrenadas se pueden utilizar *frameworks* como: *Keras, Tensor Flow o PyTorch* entre otros.

Tipos de *Transfer Learning*

Dentro del *Transfer Learning* nos encontramos 3 tipos distintos:

1. **Aprendizaje por transferencia inductiva:**
   La idea es utilizar los modelos existentes para reducir el campo de aplicación de los modelos, donde el campo de la fuente y el objetivo es el mismo, pero las tareas de fuente y objetivo son parecidas, pero no iguales. Como por ejemplo podría ser utilizar un modelo entrenado para la detección de vehículos para detectar coches.

2. **Aprendizaje por transferencia no supervisada**:
   Al igual que en el caso anterior, el campo de la fuente y el objetivo son el mismo y las tareas diferentes, la única diferencia es que los datos no están etiquetados. Es muy sencillo obtener datos no etiquetados, por lo que generalmente este tipo de aprendizaje genera un interés muy grande.

   Por ejemplo, existe un método llamado *self-taught clustering* que a partir de un conjunto de datos fuente no etiquetados es capaz de realizar un *clustering*[6] de pequeñas colecciones de datos no etiquetados. Se ha demostrado que este método es más efectivo que otros métodos.

[6] Agrupación en grupos con características similares.

3. **Aprendizaje por transferencia transductiva**:

   Con este método, en diferencia a los anteriores, las tareas fuente y objetivo son similares pero los campos correspondientes son diferentes en términos de datos.

   Por ejemplo, los modelos de Procesamiento de Lenguaje Natural, como podrían ser etiquetado morfosintáctico o POS Tagger (*Part-Of-Speech*), que son entrenados y *test*eados con datos de actualidad, se podrían extrapolar a datos extraídos de redes sociales.

3.4 Algoritmos

En este trabajo de fin de máster se trabajará un problema de visión por computación realizando una clasificación de imágenes, en concreto clasificar hongos que pueden hallarse en las hojas de maíz. Para lo cual es muy importante tener un buen conjunto de imágenes que permita etiquetar los casos y que los modelos sean capaces de distinguir los patrones que diferencian a cada una de las enfermedades.

Se han utilizado distintos modelos disponibles en *Keras*, a continuación, se van a mencionar y explicar sus características principales, para finalmente poder comparar los resultados obtenidos y determinar cuál de todos es el que mejor se ajusta al problema a resolver.

Los algoritmos son los siguientes: *Mobilenet, Xception, VGG16, VGG19, ResNet50, InceptionV3*

- MOBILENET

  Es una red convolucional que se explica en un *paper* de Google, cuyo atractivo principal es que es una red tan rápida como un modelo pequeño, debido a su pequeño tamaño se implementa habitualmente en móviles y sistemas embebidos. Tal y como se puede observar en la siguiente imagen tiene una precisión tan alta como *Googlenet* o *VGG*, pero con una arquitectura mucho más pequeña y simple.

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Imagen 9. Comparación de MobileNet, Googlenet y VGG16

- INCEPTIONV3

Al igual que *MobileNet* es una red neuronal definida en un paper de Google en la que se revisa la arquitectura original de Inception y en el que su modelo de arquitectura se forma por bloques de complicación tanto simétricos como asimétricos, incluyendo: convoluciones, reducción promedio, reducción máxima, concatenaciones, retirados y capas *fully conected*. Estando todas las entradas de activación normalizadas. La función de perdida se calcula con *softmax*. En la siguiente imagen se muestra el diagrama del modelo a alto nivel.

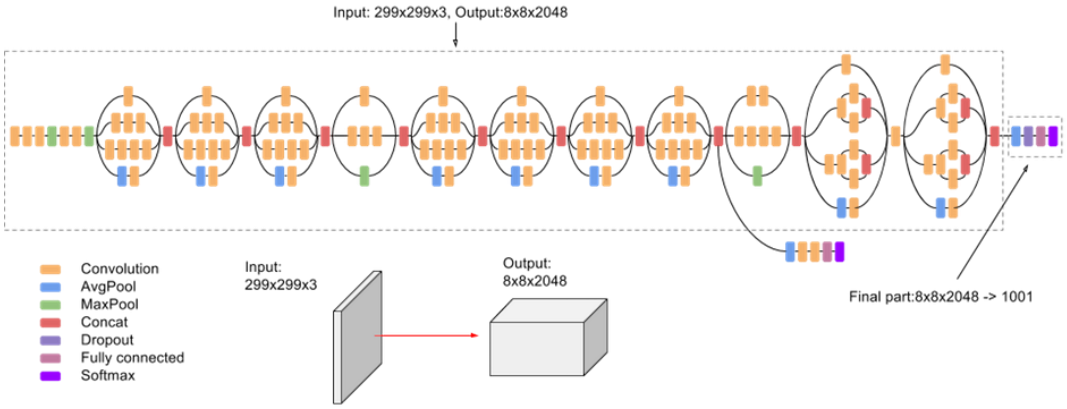

Imagen 10. Arquitectura de la red Inception

- # XCEPTION

Es otra de las mejoras del algoritmo Inception v3 propuesta por Google, hace uso principalmente de convolución separable en profundidad para reemplazar la operación de convolución original de Inception v3. La convolución separable en profundidad es básicamente Mobilenet, que divide la operación convolución en dos pasos: aplicar la convolución solamente a un canal a la vez, de esta manera ahorras realizar la multiplicación N veces donde N es el número de filtros que se aplican. El segundo paso es aplicar una convolución 1x1 a todos los canales, por lo que otra vez estamos ahorrándonos el coste de multiplicar para cada matriz por el número de filtros y el número de canales. En la siguiente imagen se observan ambos pasos de una manera más gráfica siendo a la convolución estándar y b y c el primer y segundo paso respectivamente.

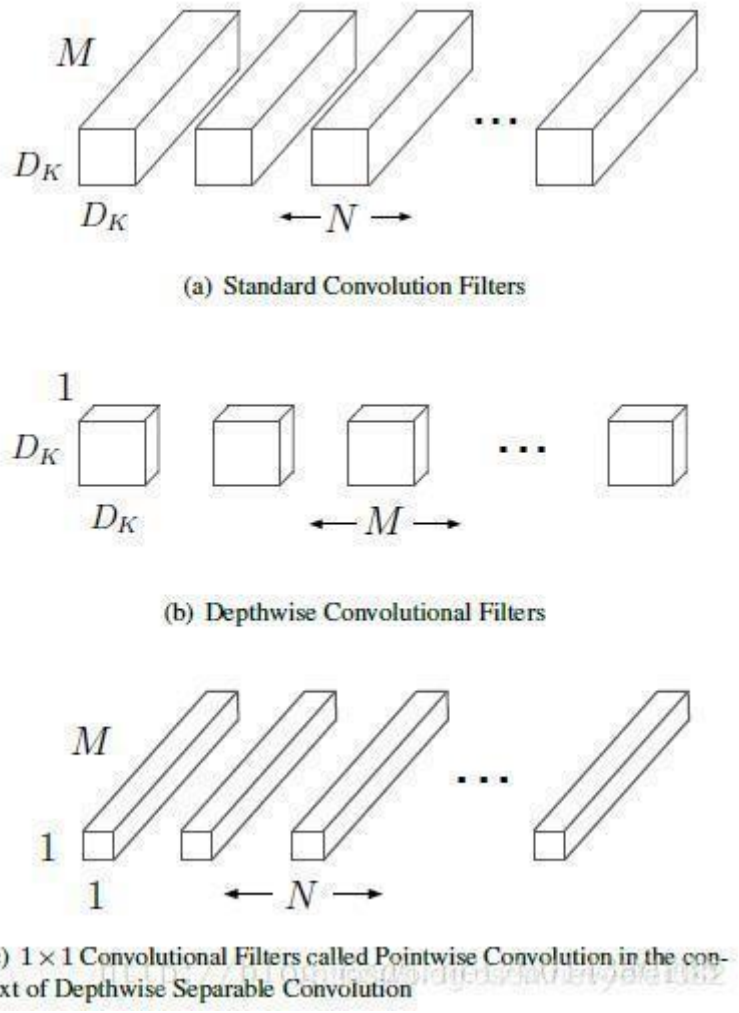

Imagen 11. Calculo convolucional de las redes Xception

.

- VGG

Es una red convolucional de la Universidad de Oxford que consiguió una de las puntuaciones más altas en *ImageNet*. Este modelo propone utilizar núcleos de convolución más pequeños, concretamente de 3x3. Hay dos algoritmos disponibles VGG16 y VGG19, la diferencia entre estos dos algoritmos es el número de capas ocultas que utilizan siendo este 16 y 19 respectivamente. Teniendo ambas una arquitectura similar la de VGG16 es tal y como se observa en la siguiente imagen.

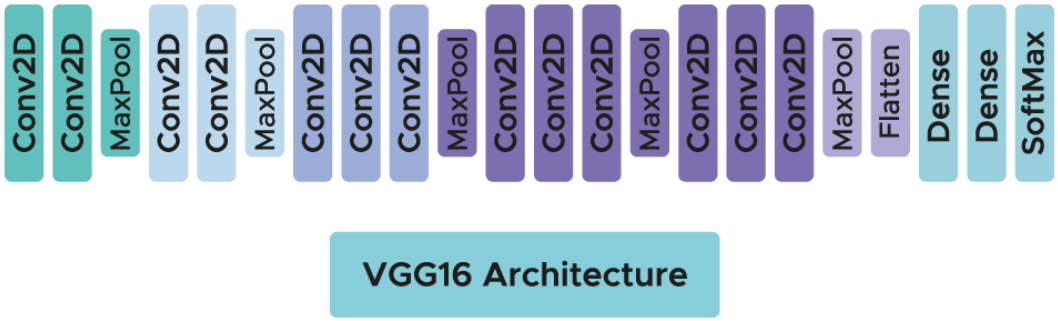

Imagen 12. Arquitectura VGG16

El algoritmo requiere un pre-procesamiento especifico: restar a cada píxel el valor RGB medio del conjunto de entrenamiento.

Durante el entrenamiento del modelo, el input a la primera capa de convolución es una imagen RGB de tamaño 224 x 224. Para todas las capas de convolución, el núcleo de convolución es de tamaño 3×3: la dimensión más pequeña para capturar las nociones de arriba, abajo, izquierda/derecha y centro. Esta era una especificidad del modelo en el momento de su publicación. Hasta el VGG16, muchos modelos estaban orientados a núcleos de convolución de mayor dimensión (tamaño 11 o tamaño 5, por ejemplo). Recordemos que el objetivo de estas capas es filtrar la imagen manteniendo sólo la información discriminante, como las formas geométricas atípicas.

Estas capas de convolución van acompañadas de capas Max-Pooling, cada una de ellas de tamaño 2×2, para reducir el tamaño de los filtros durante el entrenamiento.

A la salida de las capas de convolución y agrupación, tenemos 3 capas de neuronas totalmente conectadas. Las dos primeras están compuestas por 4096 neuronas y la última por 1000 neuronas con una función de activación softmax para determinar la clase de imagen.

Como se puede ver, la arquitectura es clara y sencilla de entender, lo que también es un punto fuerte de este modelo.

- # RESNET50

ResNet es una red neuronal residual donde se permiten conexiones para moverse sobre varias capas, tal y como se puede observar en la siguiente imagen.

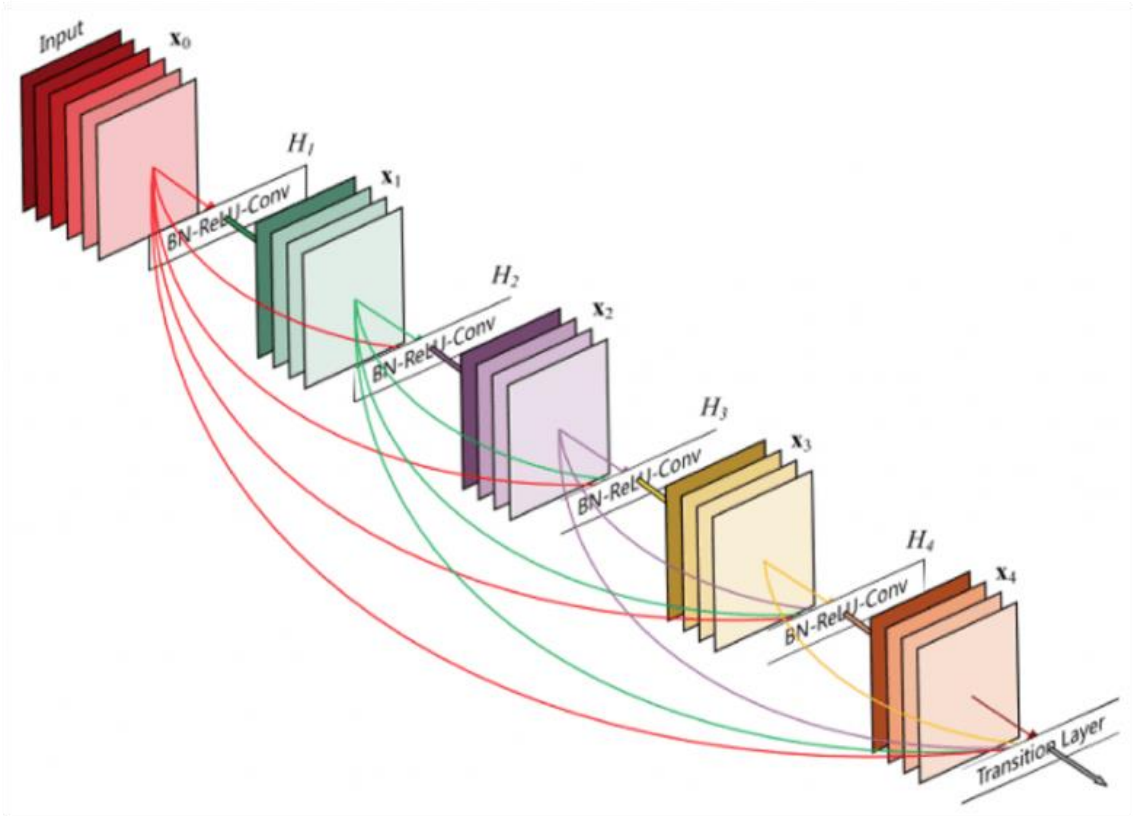

Imagen 13. Arquitectura y funcionamiento de ResNet50

Saltar capas evita a los gradientes que se desvanezcan, ya que al "retro propagarse" a capas anteriores y repetir muchas veces el proceso llegara a ser extremadamente pequeño, de forma que se utilizan las capas anteriores para que las capas adyacentes aprendan los pesos particulares. Mientras se entrena, los pesos se ajustan a las capas anteriores y amplían la capa saltada anteriormente.

Esto no funciona en todos los casos, solamente cuando todas las capas intermedias son lineales o superponen a la capa no lineal. Los saltos permiten que la red sea más simple y utilizando pocas capas durante el entrenamiento.

Capítulo 4. TECNOLOGÍAS

En este apartado se van a exponer cuales son las tecnologías utilizadas y porque se han decidido utilizarlas, todas ellas basadas en Python, que es el lenguaje más común a la hora de programar en el mundo del *Machine Learning*.

4.1 Entorno de trabajo

Como entorno de trabajo se ha utilizado *Visual Studio Code* [9] y *Anaconda* [10], ambas se integran perfectamente ya que se combinan una sencilla interfaz para programar y trabajar con directorios de carpetas, junto con la versatilidad que proporciona *Anaconda* para instalar los paquetes necesarios en diferentes entornos. Se prefirió tener este sistema montado en local, antes que usar herramientas online como por ejemplo puede ser *Google Colab,* debido a las limitaciones computacionales que ofrece la herramienta en su formato de suscripción gratuita.

La máquina física sobre la que se ha trabajado es una estación de trabajo con las siguientes características:

- Procesador Ryzen 5 3600
- Memoria RAM de 16 GB
- GPU RTX 2060 en su modelo de 6GB de memoria virtual
- Disco duro NVMe de 500GB y altas velocidades de lectura y escritura.

4.2 Clasificación

Para realizar la construcción del clasificador de imágenes se ha utilizado *Keras* [11] que viene integrado en el paquete *TensorFlow* [12]. Es la herramienta más conocida a la hora de programar redes neuronales, sobre todo cuando se va a trabajar con redes convolucionales y el *Transfer Learning*.

Existen otras librerías que permiten trabajar con redes neuronales convolucionales como por ejemplo PyTorch [13], pero se prefirió trabajar con Tensor Flow gracias a la gran cantidad de ejemplos disponibles en internet asi como una muy buena documentación, buenos tutoriales y guías para todos los módulos disponibles.

4.3 Creación de tablas

Los resultados obtenidos de las clasificaciones realizadas se han almacenado en un fichero con formato *csv* para poder generar las medias correspondientes a cada una de las ejecuciones y poder crear las matrices de confusión, estas últimas mediante la herramienta Excel, de una forma ágil y sencilla. Se decidió utilizar Excel debido a su facilidad para crear tablas y poder exportarlas fácilmente a Word en el proceso de redacción de este documento.

Capítulo 5. DESARROLLO DEL TRABAJO

A lo largo de este capítulo se va a describir el desarrollo técnico del proyecto, es decir, la obtención y construcción del dataset y la programación realizada para la elaboración de las diferentes redes neuronales convolucionales pre-entrenadas, tratando siempre de realizar el procesamiento más eficiente posible mediante el uso de batches. El ejemplo y conocimiento mencionado a lo largo de este capítulo se ha extraído de uno de los ejemplos [13] que ofrece TensorFlow.

5.1 Obtención y creación del *dataset*

En primera instancia se trató de construir el *dataset* desde cero realizando búsquedas en diferentes webs de imágenes de libre uso como: *Pinterest* [13] o *Pxhere* [14]. Sin embargo, tras obtener un *dataset* de 150 imágenes y realizar las primeras pruebas, los resultados obtenidos no eran lo suficientemente buenos, de hecho, se comprobó que el dataset no era lo suficientemente amplio como para poder hacer una buena partición de *train*, *validation* y *test*. Lo que llevo a plantear un cambio de tema no tan especifico y que la obtención de imágenes no fuera tan costosa, pero tras una búsqueda en *Kaggle* se encontró un *dataset* [15] con aproximadamente 4000 imágenes con distintas enfermedades y maíz sano, que sí que permite poder realizar una clasificación mucho más realista y completa.

Una vez obtenido el *dataset,* se procede a dividir las imágenes en dos conjuntos: de entrenamiento (*train*) y de validación (*validation*), para poder evaluar los modelos neuronales correctamente. El reparto de imágenes entre ambos conjuntos se realizó de la siguiente manera:

- 70% de las imágenes se destinaron al conjunto de *train.*
- 30% restante se utiliza para validar los modelos.

La división de imágenes se realizó para cada una de las clases individualmente, para intentar mantener el número de imágenes lo más proporcionado posible con respecto al *dataset* original.

5.2 Construcción de las redes neuronales

El primer paso dado para llevar a cabo el proyecto fue practicar con este tipo de redes y el *Transfer Learning*, para lo cual se empleó uno de los ejemplos que facilita *TensorFlow-Keras* en su documentación y que explica paso a paso como:

1. Utilizar y preparar el dataset, tanto realizando *data augmentation* como creando *batches* para un procesamiento óptimo de la red neuronal convolucional.
2. Realizar el preprocesamiento de imágenes específico para cada red y el tamaño que estas deben tener para que la red convolucional funcione correctamente.
3. Configurar la red neuronal añadiendo los pasos anteriores (data augmentation y preprocesamiento) al modelo base a utilizar (MobileNet, ResNet …) junto con una capa de salida (*Dense*) que tenga tantas neuronas como clases el conjunto de datos, de este modo se habrá adaptado la red neuronal al problema en concreto.
4. Por último, entrenar el modelo y realizar el Fine Tuning para comparar los resultados obtenidos.

La estructura de este ejemplo ha servido para el estudio que se ha querido realizar, a continuación, se explican los pasos mencionados anteriormente con un detalle descriptivo mayor.

5.2.1 Preparación de los datos y Data augmentation

Tal y como se ha mencionado anteriormente tras crear los *dataset*s de *train* y *validation*, se reserva un 20% del conjunto de validación para la creación del conjunto de *test*, que nos permitirá testear los modelos con datos que no haya visto nunca, lo cual aportará conocimiento extra sobre el comportamiento de las distintas redes.

Al no disponer de un conjunto de datos demasiado amplio, por recomendación del tutor se decide realizar un proceso de aumento de datos o *data augmentation*, para lo cual se utilizan dos funciones nativas de *Keras*, una de ellas para voltear (*RandomFlip*) y la otra para rotar imágenes (*Random Rotation*), por lo que por cada imagen de entra se obtienen 9 imágenes “nuevas”. Tal y como se ilustra en la siguiente figura:

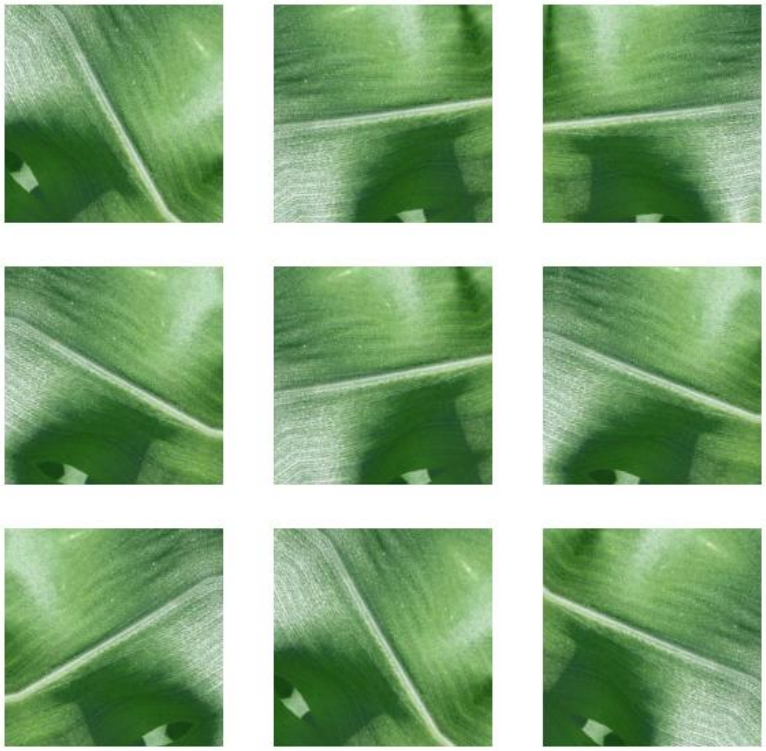

Imagen 14. Resultado de rotar y voltear una imagen

Una vez aumentado el *dataset* se realiza el preprocesamiento especifico que necesita la red neuronal para reescalar cada uno de los pixeles de todas las imágenes, para que el valor del píxel pase de un valor de entre [0,255] a [-1,1] o [0,1] dependiendo de la red convolucional.

5.2.2 Creación del modelo base

Para la creación del modelo base, es necesario hacer uso de la función de *Keras* correspondiente, el objetivo de este apartado es utilizar la red neuronal como un extractor de **características**.

Tensor Flow recomienda que a la hora de realizar la extracción característica la red neuronal base tiene que estar “congelada”, es decir, indicar al modelo que no es “entrenable”, de esta manera se evita modificar los valores de los pesos, al ser una red pre-entrenada se le pueden dar unos pesos iniciales, en este caso se han cargado los pesos de *ImageNet*, esto significa que se emplea el conocimiento obtenido por la red tras entrenarse con el conjunto de imágenes de *ImageNet*.

A la hora de trabajar con muchos modelos en el ámbito del *Fine Tuning* hay que tener especial cuidado con una de sus capas: *BatchNormalization*, esta se encarga de

normalizar las entradas mediante el uso de la media y la varianza del modelo, a la hora de emplear la red como un extractor de características estos estadísticos no se deben modificar, por lo que al congelar el modelo no se actualizan ninguno de estos los valores. Al descongelar un modelo con capas de *BatchNormalization* para realizar un *Fine Tuning*, se debe mantener las capas de *BatchNormalization* en modo de inferencia pasando `*train*ing = False`. De lo contrario, las actualizaciones aplicadas a los pesos no “entrenables” destruirán lo aprendido por el modelo.

Por último, para poder generar predicciones del bloque de características, es necesario añadir una última capa al modelo, que permita convertir las características en un solo vector de elementos por cada imagen, por tanto, una vez se han realizado los pasos anteriores, se añade una nueva capa, la capa *Dense*, que consiste en tener tantas neuronas como clases para clasificar se dispongan, de manera que cada una de las neuronas de una probabilidad de pertenecer a una clase. En el caso de este proyecto en concreto se disponen de cuatro clases diferentes, por ende, la capa Dense dispondrá de 4 neuronas cada una de ellas “especializada” en una clase especifica.

5.2.3 Compilar y entrenar el modelo

Tras preparar los datos y el modelo al completo, hace falta compilar el modelo, en esta etapa se definen la función de perdida y la métrica a utilizar.

Existen muchas funciones de perdida diferentes, la función de perdida establecida para todos los modelos es *Sparse Categorical Cross Entropy,* al tratarse de un problema de clasificación múltiple se decidió emplear *Categorical Cross Entropy,* pero al no haber realizado *one-hot-encode* en el preprocesamiento de los datos se ha tenido que utilizar la función *Sparse.* En cualquier caso, cualquiera de las dos funciones habría servido, siempre dependiendo del preprocesamiento inicial empleado.

Por otro lado, se encuentra la métrica, dentro del contexto de enfrentarse a un problema de clasificación multiclase pueden encajar diferentes métricas como, por ejemplo: ***Accuracy****, Precision, Recall, F1 score, AUC o Categorical Cross Entropy*.

Como aproximación inicial la obtención de la *accuracy* es de gran valor, ya que da una primera impresión de todos los elementos clasificados correctamente, claro que esta métrica tiene una carencia muy importante, solamente es válido al tener las clases lo suficientemente balanceadas.

Por lo que además de medir la *accuracy*, se tienen en cuenta tambien la *precision* y el *recall* (no resulta demasiado relevante para este problema saber cómo de preciso es el modelo o cual es la proporción de elementos clasificados correctamente sino de la distribución de las clasificaciones), para lo que se decide calcular la matriz de confusión correspondiente y poder observar clase por clase el número de clasificaciones correctas e incorrectas (al ser multiclase el ratio de True Positive, True Negative, False Positive y False Negative no es tan interesante como poder analizarlo uno a uno en la matriz de confusión directamente).

Se deciden no utilizar el resto de las métricas ya que con las mencionadas se puede realizar una buena comparación de clases entre cada una de las redes neuronales.

Para la fase de entrenamiento, se marcan 10 *epochs* [7]y se almacenan los distintos valores tanto de la función de perdida como de la métrica, de forma que permita realizar representaciones gráficas de las evoluciones de ambos valores y las distintas ejecuciones.

5.2.4 Fine Tuning

El último paso del proceso de clasificación de imágenes llevado a cabo es el ***Fine Tuning***, tal y como se define y explica en la guía de *Tensor Flow* seguida a lo largo de todo el proceso de clasificación, consiste en aumentar el rendimiento mediante el entrenamiento de los pesos de las capas superiores pre-entrenadas, forzando a utilizar los pesos ajustados desde características genéricas a las características del propio *dataset*.

Es importante realizar este paso después de haber entrenado las capas superiores del clasificador, de otra manera si se agregase un clasificador inicializado aleatoriamente encima de un modelo previamente entrenado y se intentara entrenar todas las capas a la vez, la magnitud de las actualizaciones de gradiente sería demasiado grande (debido a los pesos aleatorios del clasificador) y el modelo previamente entrenado olvidaría lo aprendido.

Además, se debe intentar ajustar una pequeña cantidad de capas superiores en lugar de todo el modelo. En la mayoría de las redes convolucionales, cuanto más arriba está una capa, más especializada es. Las primeras capas aprenden características muy simples y genéricas que se generalizan a casi todos los tipos de imágenes. A medida que avanza, las funciones son cada vez más específicas para el conjunto de datos en el que se entrenó el modelo. El objetivo del *Fine Tuning* es adaptar estas

[7] Instante de tiempo que se considera como punto de inicio para un evento.

características especializadas para que funcionen con el nuevo conjunto de datos, en lugar de sobrescribir el aprendizaje genérico.

En este paso se deben desbloquear las capas superiores del modelo, pero manteniendo los pesos de la capa *BatchNormalization* por lo que `*train*ing = False` haciendo que la capa se ejecute en modo de inferencia, además con el objetivo de que el modelo no haga *overfitting* se modifica la *learning rate* a un valor más pequeño. De modo que al entrenar las métricas calculadas aumentaran con respecto al modelo sin realizar el *Fine Tuning*.

5.3 Resultados obtenidos

A continuación, se van a representar los resultados de la función de perdida y la *accuracy*, tras iterar 3 veces cada una de las redes neuronales, cada iteración se compone por 10 *epochs*, las columnas de la izquierda representan el estado del entrenamiento antes del *Fine Tuning* y las de la derecha después del *Fine Tuning*.

	*Train*ing			*Fine Tuning*	
	Loss	Accuracy		Loss	Accuracy
1º	0,4568	0,8193		0,2389	0,9073
2º	0,4804	0,8106		0,2412	0,9056
3º	0,4974	0,8041		0,2490	0,9024

Tabla 1. Resultados de loss y accuracy de Mobilenet

	*Train*ing			*Fine Tuning*	
	Loss	Accuracy		Loss	Accuracy
1º	0,427	0,8189		0,1831	0,9369
2º	0,4174	0,822		0,1878	0,9349
3º	0,41	0,8265		0,1934	0,9329

Tabla 2.Resultados loss y accuracy de ResNet50

	*Train*ing			*Fine Tuning*	
	Loss	Accuracy		Loss	Accuracy
1º	0,90333	0,70525		0,63083	0,77415
2º	0,9298	0,70394		0,64109	0,77389
3º	0,96872	0,6983		0,65687	0,77271

Tabla 3. Resultados loss y accuracy de VGG16

	*Train*ing			*Fine Tuning*	
	Loss	Accuracy		Loss	Accuracy
1º	1,1288	0,6497		0,6843	0,7816
2º	1,0881	0,665		0,6811	0,7833
3º	1,0735	0,6764		0,6771	0,7852

Tabla 4. Resultados loss y accuracy de VGG19

	*Train*ing			*Fine Tuning*	
	Loss	Accuracy		Loss	Accuracy
1º	0,5869	0,7598		0,2636	0,9066
2º	0,578	0,7626		0,2468	0,9142
3º	0,5711	0,7655		0,24	0,9181

Tabla 5. Resultados loss y accuracy de Inception

	*Train*ing			*Fine Tuning*	
	Loss	Accuracy		Loss	Accuracy
1º	0,6272	0,7482		0,2933	0,8759
2º	0,6346	0,7486		0,2993	0,8722
3º	0,6385	0,748		0,3008	0,8705

Tabla 6. Resultados loss y accuracy de Xception

Como se puede ver en los resultados de *accuracy* de las tablas anteriores, los modelos más simples obtienen un mejor resultado, seguramente achacado a que al tener un *dataset* reducido se centran en las características más evidentes dejando de lado otro tipo de características más rebuscadas, de forma que, si hay enfermedades con características similares, podría llegar a aprender alguna característica demasiado genérica que no le permita distinguir correctamente entre esas clases.

Al estudiar las matrices de confusión representadas en las siguientes tablas, resulta evidente que tanto los modelos VGG como el Inception están aprendiendo algo de la enfermedad *Gray Leaf Spot* que lo confunde con *Blight*, y que el resto de los modelos no se equivocan tanto.

MOBILENET	Blight	Common rust	Gray Leaf Spot	Healthy
Blight	6	0	1	1
Common rust	1	11	0	0
Gray Leaf Spot	0	0	4	0
Healthy	0	0	0	8

Tabla 7 Matriz de confusión de MOBILENET

RESNET50	Blight	Common rust	Gray Leaf Spot	Healthy
Blight	7	0	0	0
Common rust	0	11	0	0
Gray Leaf Spot	1	0	5	0
Healthy	0	0	0	8

Tabla 8Matriz de confusión de RESNET50

VGG16	Blight	Common rust	Gray Leaf Spot	Healthy
Blight	5	0	1	0
Common rust	1	8	1	0
Gray Leaf Spot	7	0	1	0
Healthy	0	0	0	8

Tabla 9Matriz de confusión de VGG16

VGG19	Blight	Common rust	Gray Leaf Spot	Healthy
Blight	6	0	1	0
Common rust	2	8	0	0
Gray Leaf Spot	3	1	3	1
Healthy	0	0	0	7

Tabla 10Matriz de confusión de VGG19

INCEPTION	Blight	Common rust	Gray Leaf Spot	Healthy
Blight	7	0	0	0
Common rust	1	7	0	0
Gray Leaf Spot	5	0	6	0
Healthy	0	0	0	8

Tabla 11Matriz de confusión de INCEPTION

XCEPTION	Blight	Common rust	Gray Leaf Spot	Healthy
Blight	10	0	1	0
Common rust	0	7	0	0
Gray Leaf Spot	1	0	4	0
Healthy	0	0	0	9

Tabla 12Matriz de confusión de XCPETION

De hecho, se observa que las imágenes sanas, que deberían ser muy diferentes a las que contengan hongos, las clasifica prácticamente de forma perfecta con respecto al resto de clases, lo cual nos indica que los modelos no hacen *underfitting*, pero que tampoco hacen *overfitting* porque para algunas clases no realizan una clasificación perfecta.

Capítulo 6. CONCLUSIONES

Tras recopilar las imágenes de hojas de maíz descargadas a través del repositorio de *Kaggle*, junto con las pautas indicadas por el tutor de este trabajo y las distintas guías de aprendizaje de *Tensor Flow*, se ha logrado el objetivo final de este trabajo, clasificando satisfactoriamente las imágenes obteniendo unas métricas bastante aceptables.

No ha sido tarea fácil debido a la dificultad que ha supuesto obtener las imágenes y al trabajo manual que se ha debido realizar para revisar que todas las imágenes fuesen validas y realizar una división del dataset tanto para entrenar como para validar cada una de las redes neuronales. A pesar de no disponer de un dataset muy amplio, esto no ha supuesto una contra ya que el *transfer learning* es una de las prácticas más comunes al encontrarnos con esta casuística y que ya se planteó desde un principio como objetivo, de forma que jugaba a favor y no en contra.

Cabe destacar la importancia que tiene tener un conjunto de datos bien clasificado para que el posterior trabajo tenga consistencia y sea preciso.

Para terminar, se incluye un pequeño resumen de los diferentes modelos en el que los modelos *MobileNet*, *ResNet, Inception* y *Xception* realizan una clasificación más precisa, siendo ***ResNet*** la red neuronal seleccionada como modelo para clasificar imágenes en este problema en concreto. En cambio, las redes *VGG16 y VGG19* realizan una clasificación peor que los anteriores modelos mencionados, pero esto no significa que sean malos, sino que al observar las matrices de confusión se aprecia que confunden *Gray Leaf Spot,* pero para el resto de las clases realizan una clasificación bastante buena.

MODELO	SPARSE CATEGORICAL CROSS ENTROPY	ACCURACY
MOBILENET	0,24303	0,90510
RESNET50	0,18810	0,93490
VGG16	0,68755	0,76482
VGG19	0,68083	0,78337
INCEPTION	0,25013	0,91297
XCEPTION	0,29780	0,87287

Tabla 13 Tabla de resultados de la función de perdida Sparse Categorical Cross Entropy y la metrica accuracy, medias de todas las ejecuciones realizadas para cada uno de los modelos

Capítulo 7. FUTURAS LÍNEAS DE TRABAJO

Como futuras líneas de trabajo, ha quedado pendiente un preprocesamiento extra sobre las imágenes, básicamente consiste en realizar un proceso de *data-augmentation* mucho más específico que simplemente rotando y volteando las imágenes con las funciones de *Keras*. La idea principal es la misma, pero adicionalmente se crean "mini parches" de la imagen de forma que habría que seleccionar cada uno de esos parches a mano identificando cuales tienen parte de imagen que interesa (hoja de maíz) y cuales no interesan (cielo, mesa, otras plantas, etc.) al modelo y que podrían hacer equivocarse al modelo a la hora de aprender características.

Por otro lado, habría sido interesante probar diferentes tamaños de imágenes de entrada, asi como probar diferentes *learning rates* para comparar el comportamiento de todos los modelos en diferentes situaciones.

Para terminar de completar el proyecto sería muy positivo para los resultados realizar un análisis de optimización de parámetros con los que hacer disminuir la función de perdida y mejorar aún más los resultados de las métricas medidas, así como de sus respectivas matrices de confusión.

BIBLIOGRAFÍA

[1] K. P. Panigrahi, H. Das, A. K. Sahoo y S. C. Moharana, «Maize Leaf Disease Detection and Classification Using *Machine Learning* Algorithms,» de *Progress in Computing, Analytics and Networking*, Singapore, 2020.

[2] S. Tulasi Krishna y H. k. Kalluri, «Deep Learning and *Transfer Learning* Approaches for Image Classification,» June 2019.

[3] B. Kusumo, A. Heryana, O. Mahendra y H. Pardede, «*Machine Learning*-based for Automatic Detection of Corn-Plant Diseases Using Image Processing,» 2018.

[4] A. Ali, S. Qadri, W. K. Mashwani, S. B. Belhaouari, S. Naeem, S. Rafique, F. Jamal, C. Chesneau y S. Anam, «*Machine Learning* approach for the classification of corn seed using hybrid features,» *International Journal of Food Properties,* vol. 23, pp. 1110-1124, 2020.

[5] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection,» de *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] T. Guo, J. Dong, H. Li y Y. Gao, «Simple convolutional neural network on image classification,» de *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, 2017.

[7] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng y M. Chen, «Medical image classification with convolutional neural network,» de *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, 2014.

[8] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura y R. M. Summers, «Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, *Dataset* Characteristics and *Transfer Learning*,» *IEEE Transactions on Medical Imaging,* vol. 35, pp. 1285-1298, 2016.

[9] Microsoft, «Visual Studio Code Documentation,» 2022. [En línea]. Available: https://code.visualstudio.com/docs.

[10] Anaconda, «Anaconda Documentation,» 2022. [En línea]. Available: https://docs.anaconda.com/.

[11] *Keras*, «*Keras* Documentation,» [En línea]. Available: https://*Keras*.io/.

[12] TensorFlow, «TensorFlow API Documentation,» 2022. [En línea]. Available: https://www.tensorflow.org/api_docs.

[13] Pinterest, «Pinterest API,» [En línea]. Available: https://developers.pinterest.com/.

[14] «Pxhere,» [En línea]. Available: https://pxhere.com/en/license.

[15] S. GHOSE, «Kaggle *Dataset*s,» 11 11 2020. [En línea]. Available: https://www.kaggle.com/*dataset*s/smaranjitghose/corn-or-maize-leaf-disease-*dataset*.