Reconocimiento de imágenes mediante aprendizaje profundo

Autor: Martín Vilasánchez Freijomil

Curso: 2º Técnico Superior en Desarrollo de Aplicaciones Multiplataforma

Tutor: Rubén González Martín

Junio 2023

ABSTRACT

El 15 de junio de 2022 la Fundación Princesa de Asturias concedió por unanimidad de los miembros del jurado el Premio Princesa de Asturias de Investigación Científica y Técnica a los que son considerados los padres de una técnica esencial de la inteligencia artificial, el deep learning o aprendizaje profundo. Según el jurado, el impacto actual y futuro del aprendizaje profundo en el progreso de la sociedad puede ser calificado de extraordinario.

Una de las aplicaciones de naturaleza más populares del mundo, iNaturalist nos ayuda a identificar las plantas y los animales que nos rodean. Es una iniciativa conjunta de la Academia de Ciencias de California y la National Geographic Society, en la que, al registrarnos y compartir nuestras observaciones, se crearán datos de calidad de investigación para los científicos que trabajan para comprender y proteger mejor la naturaleza.

Estas dos inquietudes nos han llevado a plantearnos un proyecto en el que pueda adquirir conocimiento sobre las tecnologías relacionadas con el aprendizaje profundo y que nos permita el automatizar la catalogación de la biodiversidad natural.

Abstract

On June 15, 2002, the Princess of Asturias Foundation unanimously awarded the Princess of Asturias Award for Technical and Scientific Research to those who are considered the fathers of an essential technique in artificial intelligence, deep learning. According to the jury, the current and future impact of deep learning on the progress of society can be described as extraordinary.

One of the world's most popular nature apps, iNaturalist helps us identify the plants and animals that surround us. It is a joint initiative of the California Academy of Sciences and the National Geographic Society, whereby logging and sharing our observations, research-quality data will be created for scientists working to better understand and protect nature.

These two questions have led us to consider a project in which we can gain knowledge about deep learning technologies and automate the cataloguing of natural biodiversity.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

1. JUSTIFICACIÓN DEL PROYECTO

Este trabajo de fin de grado (TFG) representa un nuevo nivel de crecimiento académico y personal. Salir de la zona de confort ha sido la esencia de lo que he aprendido en el Instituto Nebrija a lo largo de estos 2 años. Es un TFG sobre la investigación de redes neuronales que me brinda la oportunidad de desafiar mis propias limitaciones y trascender lo que consideraba posible. Familiarizarme con nuevas técnicas y mantenerme actualizado en un campo en constante evolución. Aunque pueda parecer intimidante, ha sido una de las mejores decisiones que podría haber tomado, ya que mi experiencia en el Instituto Nebrija me ha preparado para abrazar estos desafíos. Estoy emocionado por la oportunidad de aplicar lo aprendido y expandir aún más mis conocimientos en un área tan puntera:

Cabe recordar que el 15 de junio de 2022 la Fundación Princesa de Asturias concedió por unanimidad de los miembros del jurado el Premio Princesa de Asturias de Investigación Científica y Técnica a los que son considerados los padres de una técnica esencial de la inteligencia artificial, el deep learning o aprendizaje profundo. Según el jurado, el impacto actual y futuro del aprendizaje profundo en el progreso de la sociedad puede ser calificado de extraordinario.

2. INTRODUCCIÓN

En este trabajo exploraremos el reconocimiento automático de imágenes, cuya finalidad es construir un clasificador que sea capaz de distinguir entre diez mil clases de seres vivos que en buena parte de las ocasiones compartirán características similares. Haremos uso de redes neuronales convolucionales (CNN) para la construcción de nuestro modelo. En una clase de arquitectura ampliamente utilizada en los últimos años para tareas de clasificación de imágenes.

Aplicaremos la premisa básica del aprendizaje por transferencia: tomaremos un modelo entrenado en un gran conjunto de datos y transferiremos su conocimiento a nuestro conjunto de datos. Así, podemos utilizar una red entrenada en categorías no relacionadas en un conjunto de datos masivo (normalmente ImageNet[1]) y aplicarla a nuestro propio problema porque hay características universales de bajo nivel compartidas entre las imágenes.

El conjunto de datos que utilizaremos para entrenar y validar nuestro modelo ha sido descargado de “iNat Challenge 2021 - FGVC8” que contiene seiscientas mil imágenes de diez mil categorías diferentes de animales, hongos y plantas. Las imágenes han sido obtenidas en localizaciones de todo el planeta y en circunstancias muy diferentes.

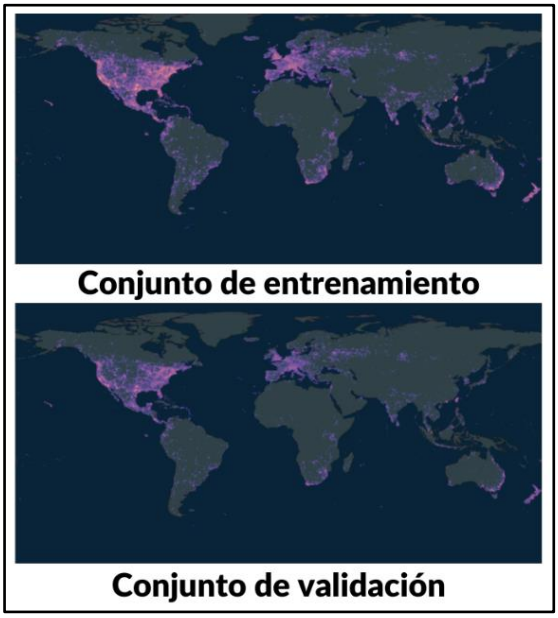

Figura 1 Distribución geográfica de las imágenes

[1] **ImageNet** es el conjunto de datos por excelencia en la actualidad para evaluar algoritmos de clasificación, localización y reconocimiento de imágenes. Es un proyecto a gran escala que involucra a diversas instituciones educativas, como Stanford y Princeton. Su objetivo es ser un banco de datos visual enorme para la investigación y desarrollo de software especializado en reconocimiento de imágenes

Las imágenes han sido obtenidas en infinidad de escenarios diferentes, donde las condiciones de luz, la distancia, el fondo, entre otros factores, pueden cambiar de forma significativa de una imagen a otra. Unido a la gran variedad de especies a clasificar lo convierten en el mayor reto que debe afrontar nuestro modelo.

Desarrollaremos un clasificador que recibirá una única imagen que contendrá un elemento de las especies incluidas en nuestro modelo de datos, y deberá de predecir correctamente de que especie se trata. Nos devolverá como resultado el nombre común de esa especie y su clasificación taxonómica.

2.1 Enfoque y método seguido

Expondremos a continuación, cual ha sido el proceso que hemos seguido para la realización del proyecto:

1. *Estudio de las redes CNN y sus diferentes arquitecturas*. Hemos realizado un estudio de las técnicas que se utilizan actualmente en la clasificación de imágenes.

2. *Análisis del conjunto de datos*. Un punto muy importante a la hora de realizar un aprendizaje supervisado es analizar previamente el conjunto de datos con el que vamos a trabajar. Identificar las categorías que están incluidas en el conjunto de datos y los potenciales problemas que puedan estar ocultos en los datos y que puedan ser subsanados o atenuados antes de su utilización.

3. *Pre-procesado del conjunto de datos e implementación de los modelos*. Una vez hecho el análisis del conjunto de datos procederemos a cargar los datos y realizar las transformaciones necesarias previas para un uso lo más eficiente posible. Implementaremos los diferentes modelos de CNN que vamos a analizar.

4. *Entrenamiento y selección del modelo a implementar*. Realizaremos entrenamientos de los diferentes modelos y seleccionaremos el que consideremos más adecuado para la realización de predicciones.

5. *Prueba de concepto: desarrollo de una aplicación web*. Por último, implementaremos una aplicación web que nos permita subir una imagen y a partir de la misma realizar la predicción con nuestro modelo de CNN, entregando el nombre común y la clasificación taxonómica del elemento predominante en la imagen.

2.2 Planificación del trabajo

A continuación, incorporamos el cronograma:

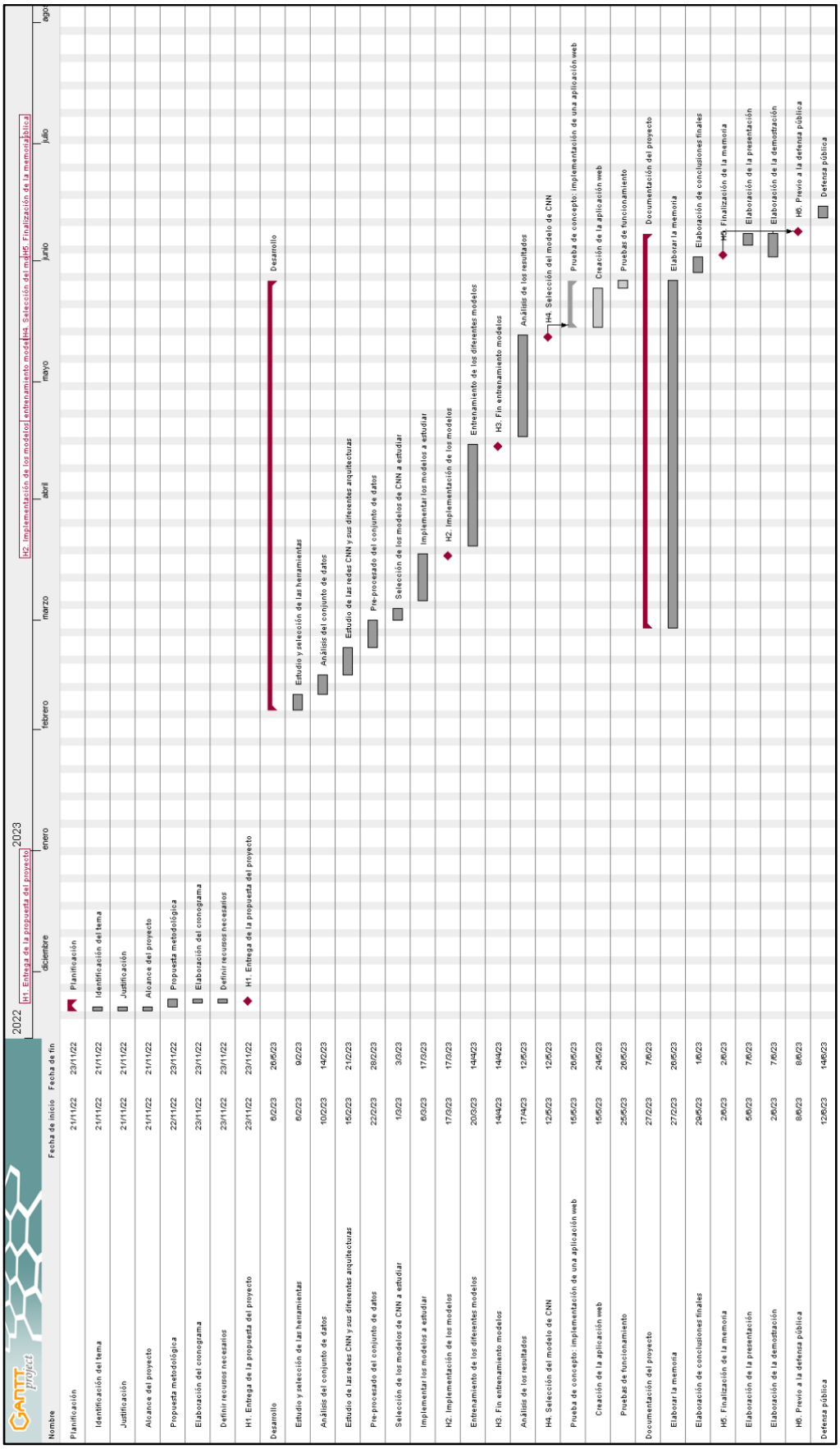

Figura 2 Diagrama de Gantt

Las fechas más destacadas en la elaboración de nuestro proyecto son las siguientes:

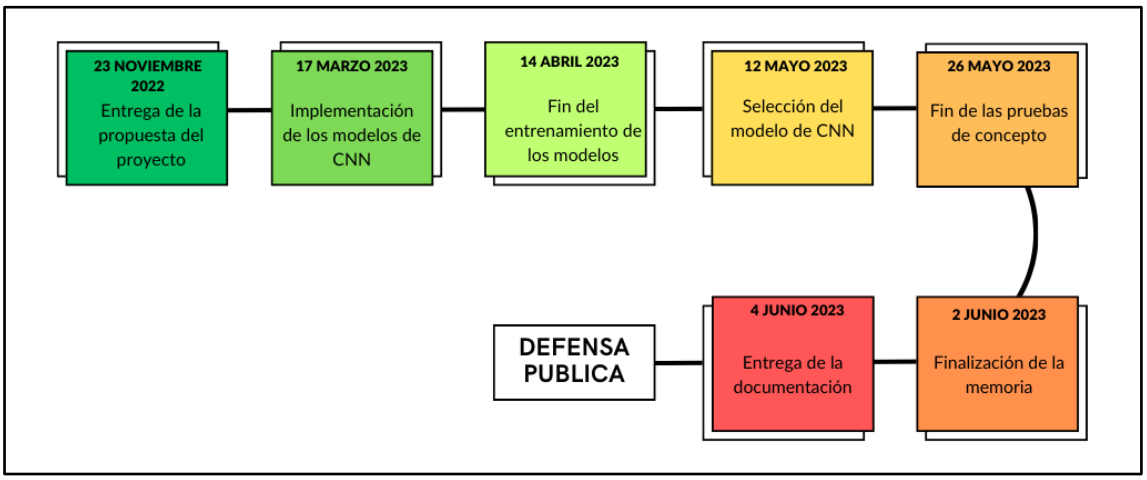

Figura 3 Hitos del proyecto

2.3 Entregables

Detallamos a continuación los entregables obtenidos como resultado final de la elaboración de este trabajo de fin de grado:

- **Memoria**. Es el presente documento, en el que reunimos toda la información relacionada con el desarrollo del trabajo. En él se detalla desde la justificación del mismo, pasando por los objetivos que pretendemos alcanzar, la planificación, metodología utilizada, una descripción detallada del trabajo realizado, hasta las conclusiones que hemos obtenido al finalizar.
- **Repositorio de GitHub**, donde hemos depositado todo el código que se ha utilizado en la elaboración del trabajo. Los cuadernos de Jupyter empleados para la investigación y ejecución de todo el código relacionado con el aprendizaje profundo. El código fuente de la aplicación desarrollada.
- **Prueba de concepto.** Basada en el desarrollo de una aplicación web que permite realizar una predicción a partir de una imagen subida por el usuario.
- **Demostración,** del funcionamiento de la aplicación desarrollada durante la prueba de concepto.

3. OBJETIVOS

El objetivo general de este trabajo es implementar un clasificador de imágenes, capaz de reconocer y clasificar imágenes de seres vivos: animales, hongos y plantas.

Este objetivo, a su vez, lo dividiremos en los siguientes objetivos específicos a conseguir durante el desarrollo del proyecto:

- Implementar un clasificador de imágenes basado en redes neuronales convoluciones.
- Hacer uso de librerías y frameworks que se utilizan actualmente en la implementación de sistemas de aprendizaje profundo.
- Estudiar diferentes modelos de redes neuronales convolucionales y comprender el funcionamiento de estos modelos.
- Seleccionar el modelo que tenga mayor probabilidad de éxito a la hora de realizar una predicción.
- Hacer una prueba de concepto: desarrollar una aplicación web que permita importar imágenes, realizar una predicción y mostrar el resultado obtenido.

4. PRUEBA DE CONCEPTO. IMPLEMENTACIÓN DE UNA APLICACIÓN WEB.

4.1 Arquitectura de la solución.

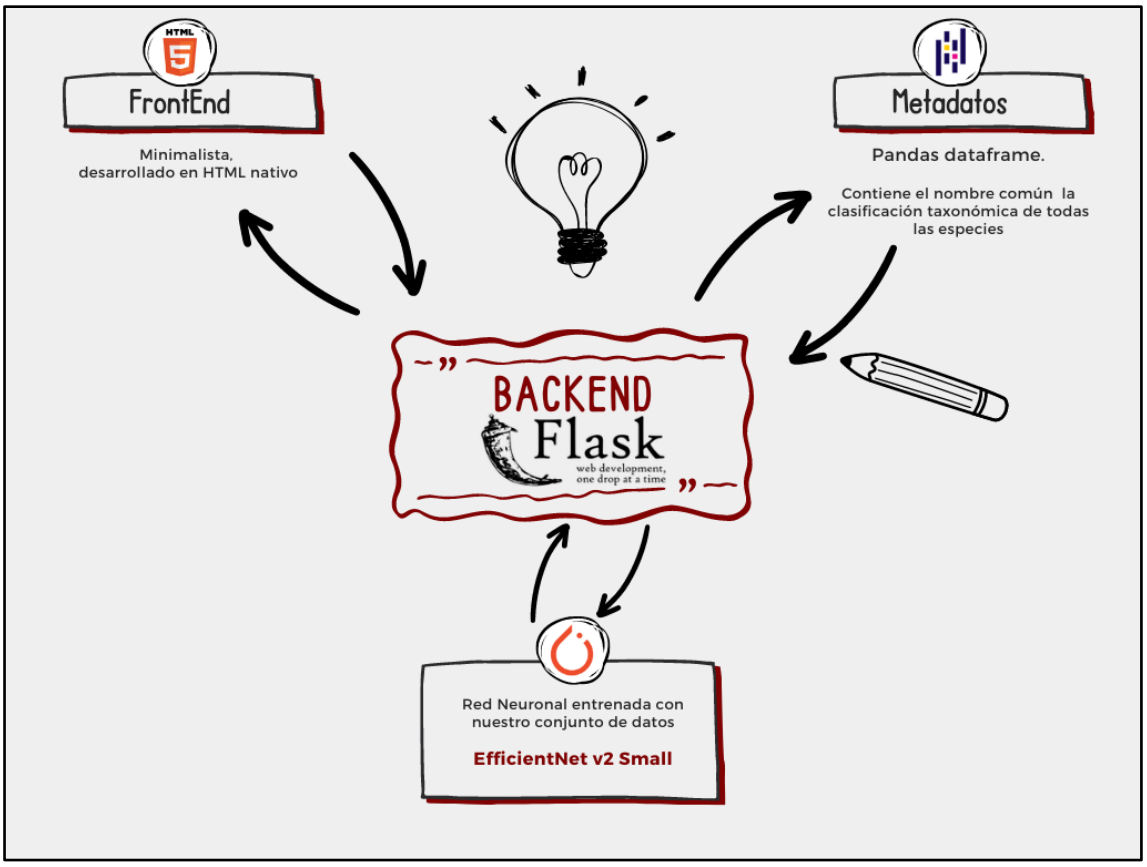

Figura 4 Prueba de Concepto. Arquitectura

FrontEnd

Nuestra aplicación consta de un pequeño portal web desarrollado en tecnologías HTML5 nativas, en el cual el usuario puede cargar el fichero .JPG que contiene la imagen que desea evaluar. El formato .JPG es el único formato que admitiremos de imagen. En este mismo portal se visualizarán los resultados del análisis de la imagen. Gracias al motor de plantillas Jinja2, se actualiza dinámicamente una vez obtenido el resultado.

Módulo metadata

Este módulo carga en un dataframe de pandas, a partir de un .csv generado con anterioridad, como explicaremos más adelante. Este dataframe contiene el nombre común y todas las categorías taxonómicas de las 10.000 especies que contiene nuestro modelo, además de un identificador de cada una de ellas.

Como todo nuestro conjunto de datos es una única tabla de 10.000 filas por 8 columnas desestimamos la implementación de una base de datos.

	common_name	kingdom	phylum	class	order	family	genus	specific_epithet
0	Common Earthworm	Animalia	Annelida	Clitellata	Haplotaxida	Lumbricidae	Lumbricus	terrestris
1	Mediterranean Fanworm	Animalia	Annelida	Polychaeta	Sabellida	Sabellidae	Sabella	spallanzanii
2	Serpula columbiana	Animalia	Annelida	Polychaeta	Sabellida	Serpulidae	Serpula	columbiana
3	Blue Tube Worm	Animalia	Annelida	Polychaeta	Sabellida	Serpulidae	Spirobranchus	cariniferus
4	Giant House Spider	Animalia	Arthropoda	Arachnida	Araneae	Agelenidae	Eratigena	duellica
5	California Turret Spider	Animalia	Arthropoda	Arachnida	Araneae	Antrodiaetidae	Atypoides	riversi
6	Oak Spider	Animalia	Arthropoda	Arachnida	Araneae	Araneidae	Aculepeira	ceropegia
7	Gorse Orbweaver	Animalia	Arthropoda	Arachnida	Araneae	Araneidae	Agalenatea	redii
8	Giant Lichen Orbweaver	Animalia	Arthropoda	Arachnida	Araneae	Araneidae	Araneus	bicentenarius
9	Cross Orbweaver	Animalia	Arthropoda	Arachnida	Araneae	Araneidae	Araneus	diadematus

Figura 5 Las 10 primeras filas del dataframe

En este módulo únicamente implementamos una función que recibe como entrada el id de la especie y devuelve un JSON con el nombre común y la clasificación taxonómica


```
{
"common_name":"Black-bellied Whistling-Duck",
"kingdom":"Animalia",
"phylum":"Chordata",
"class":"Aves",
"order":"Anseriformes",
"family":"Anatidae",
"genus":"Dendrocygna",
"specific_epithet":"autumnalis"
}
```


Figura 6 JSON devuelto por la función

Módulo CNN

La siguiente pieza de nuestra arquitectura contiene el modelo de red neuronal que previamente hemos entrenado con el conjunto de datos de iNaturalist y que se encargará de predecir una especie con un nivel de confianza para cada imagen que se le facilite.

Hemos implantado dos funciones. La primera de ellas recibe el fichero que contiene la imagen en formato .JPG, esta imagen se transforma en un tensor, que a su vez se incluye dentro de un nuevo tensor con una única posición y los mismos datos subyacentes del primer tensor. El segundo tensor es el que devuelve la función como resultado.

En la parte izquierda de la siguiente figura se muestra una imagen .JPG y en la parte derecha el tensor resultante de procesar dicha imagen con la función anteriormente descrita.

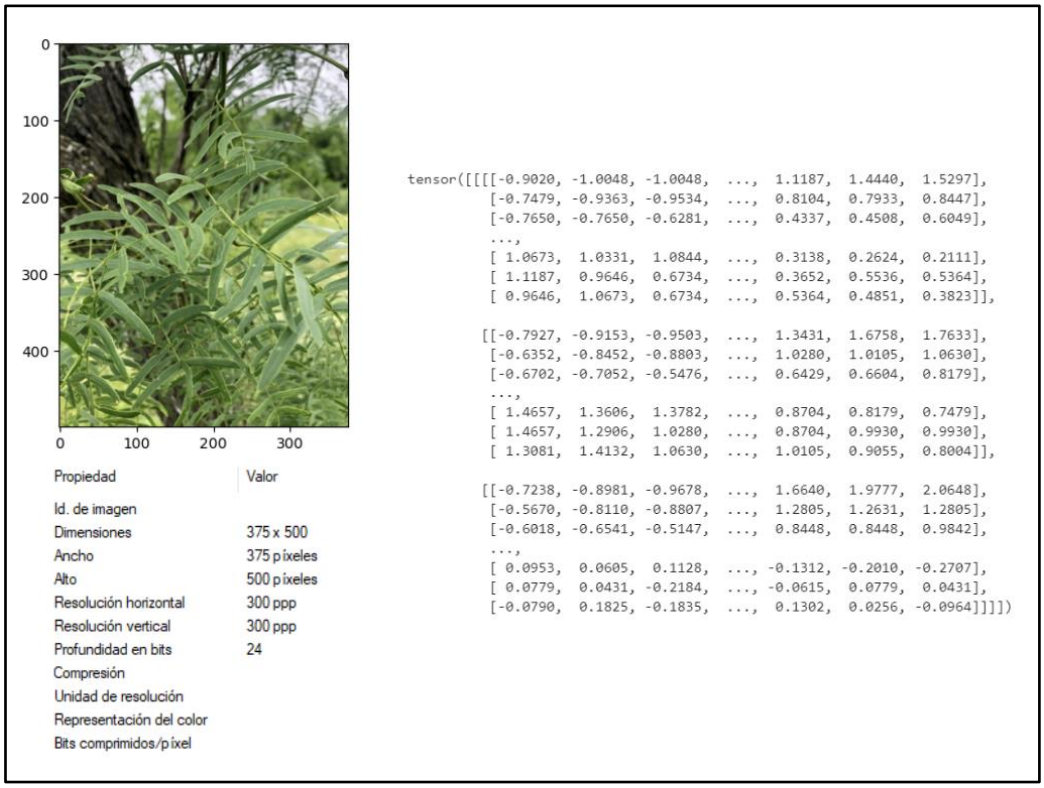

Figura 7 Tensor que a su vez contiene un único tensor

La segunda función recibe un tensor que a su vez contiene un único segundo tensor. Lo lanza contra el modelo de nuestra red neuronal, que hemos puesto en modo evaluación y obtiene el identificador de la especie cuyo valor de confiabilidad es el mayor de todos; también devuelve el valor de confiabilidad obtenido.

BackEnd

Y finalmente para el backend hemos optado por utilizar Flask. Y que se comporta de la siguiente forma:

- En el momento que recibe un fichero .JPG que proviene del FrontEnd, lo envía al módulo CNN que se encarga de transformar el fichero en un tensor, que a su vez contiene un segundo tensor con la imagen transformada.
- El siguiente paso es enviar el tensor de nuevo al módulo CNN donde ahora nuestro modelo de red neuronal calcula cual es la especie que tiene mayor nivel de confiabilidad. Devuelve el nivel de confiabilidad y el identificador de la especie seleccionada.

- Con el identificador de la especie se invoca a la función del módulo de metadatos. Este módulo nos devolverá un JSON con el nombre común y la clasificación taxonómica de la especie.
- El propio backend interpreta el fichero JSON y junto con el valor de confiabilidad y la imagen proporcionada por el usuario monta las plantillas Jinja2 y se las retorna al front.

4.2 Implementación

Una vez que hemos explicado cómo se deberá de comportar nuestra arquitectura, vamos a detallar como hemos logrado implementar esta arquitectura en nuestra aplicación.

Módulo Metadata (metadata.py)

```python
import pandas as pd

path_data = 'models/categorias_iNat2021.txt'

# cargamos el fichero con las categorias
df = pd.read_csv(path_data, sep = '\t')

# devuelve el nombre común y la clasificación taxonómica
def render_prediction(prediction_idx):
    result = df.iloc[prediction_idx].to_json(orient="index")
    return result
```

Figura 8 metadata.py

Módulo CNN (cnn.py)

```python
import torch
import torchvision.models as models
import torchvision.transforms as transforms
import PIL.Image as Image
import numpy as np

img_size = (224, 224)
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
path_cnn = 'models/EfficientNetV2S_iNat.pt'

device = torch.device('cpu')
model = torch.load(path_cnn, map_location=device)
model.eval()

# Función que transforma la imagen en un tensor
def transform_image(infile):
    data_transform = transforms.Compose([
        transforms.Resize(img_size),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)])
```

```
    # abre el fichero imagen
    image = Image.open(infile)
    # transforma la imagen en un tensor
    tensor_image = data_transform(image)
    # Crea un modelo de entrad por lotes con un solo elemento
    return tensor_image.unsqueeze(0)

# Función que realiza la prediccion
def get_prediction(input_tensor):
    # obtiene todas las probabilidades
    outputs = model.forward(input_tensor)
    # selecciona la clase con mayor probabilidad
    prob, classes = outputs.max(1)
    return prob.item(), classes.item()
```

Figura 9 cnn.py

Módulo Main (app.py)

```
from flask import Flask, request , render_template
from cnn import transform_image, get_prediction
from metadata import render_prediction
import json
from PIL import Image

app = Flask(__name__)

#archivos permitidos para subir al servidor
ALLOWED_EXTENSIONS = {'jpg'}
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

#funcion para renderizar la pagina web
@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')


#funcion para predecir la imagen subida por el usuario y devolver la prediccion en la pagina web
#y que devuelve un error si no es un archivo permitido
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        file = request.files['fileToUpload']
        if file is not None and allowed_file(file.filename):

            #guardamos la imagen en la carpeta static/images
            file.save("static/images/" + file.filename)
            image = "static/images/" + file.filename

            #llamamos a la funcion transform_image para transformar la imagen en un tensor
            input_tensor = transform_image(file)
            predict_prob, prediction_idx = get_prediction(input_tensor)
            class_name = render_prediction(prediction_idx)

            #pasamos el predict prob a 2 digitos decimales
            predict_prob = round(predict_prob, 2)
```

```
            #parseamos el json class_name para devolver las categorias de la imagen almacenadas
            #en el json pasandolos a strings
            details = json.loads(class_name)
            common_name = details["common_name"]
            kingdom = details["kingdom"]
            phylum = details["phylum"]
            class_ = details["class"]
            order = details["order"]
            family = details["family"]
            genus = details["genus"]
            specific_epithet = details["specific_epithet"]

            #creamos un diccionario con los datos de la prediccion
            response = {'prediction': {'accuracy': predict_prob, 'common_name': common_name, 'kingdom': kingdom,
                                       'phylum': phylum, 'class': class_, 'order': order, 'family': family,
                                       'genus': genus, 'specific_epithet': specific_epithet}}

            #devolvemos la prediccion en a la pagina web
            return render_template('index.html',
                                   user_image = image,
                                   prediction_text = "Con una precisión del {0} '%' ".format(response["prediction"]["accuracy"]),
                                   prediction_text2 =  " El Nombre comun es: {0}".format(response["prediction"]["common_name"]),
                                   prediction_text3 =  " El reino es: {0}".format(response["prediction"]["kingdom"]),
                                   prediction_text4 =  " El phylum es: {0}".format(response["prediction"]["phylum"]),
                                   prediction_text5 =  " La clase es: {0}".format(response["prediction"]["class"]),
                                   prediction_text6 =  " El orden es: {0}".format(response["prediction"]["order"]),
                                   prediction_text7 =  " La familia es: {0}".format(response["prediction"]["family"]),
                                   prediction_text8 =  " El genero es: {0}".format(response["prediction"]["genus"]),
                                   prediction_text9 =  " La especie es: {0}".format(response["prediction"]["specific_epithet"])
                                   )

        return render_template('index.html', prediction_text="No se ha podido realizar la predicción,
                               por favor seleccione un archivo valido e inténtelo de nuevo.")
if __name__ == '__main__':
    app.run(debug=True)
```

Figura 10 app.py

5. DESARROLLO

5.1 Marco teórico

5.1.1 Organización taxonómica

Según la R.A.E. la taxonomía es la ciencia que trata de los principios, métodos y fines de la clasificación. Se aplica en particular, dentro de la biología, para la ordenación jerarquizada y sistemática, con sus nombres, de los grupos de animales y de vegetales.

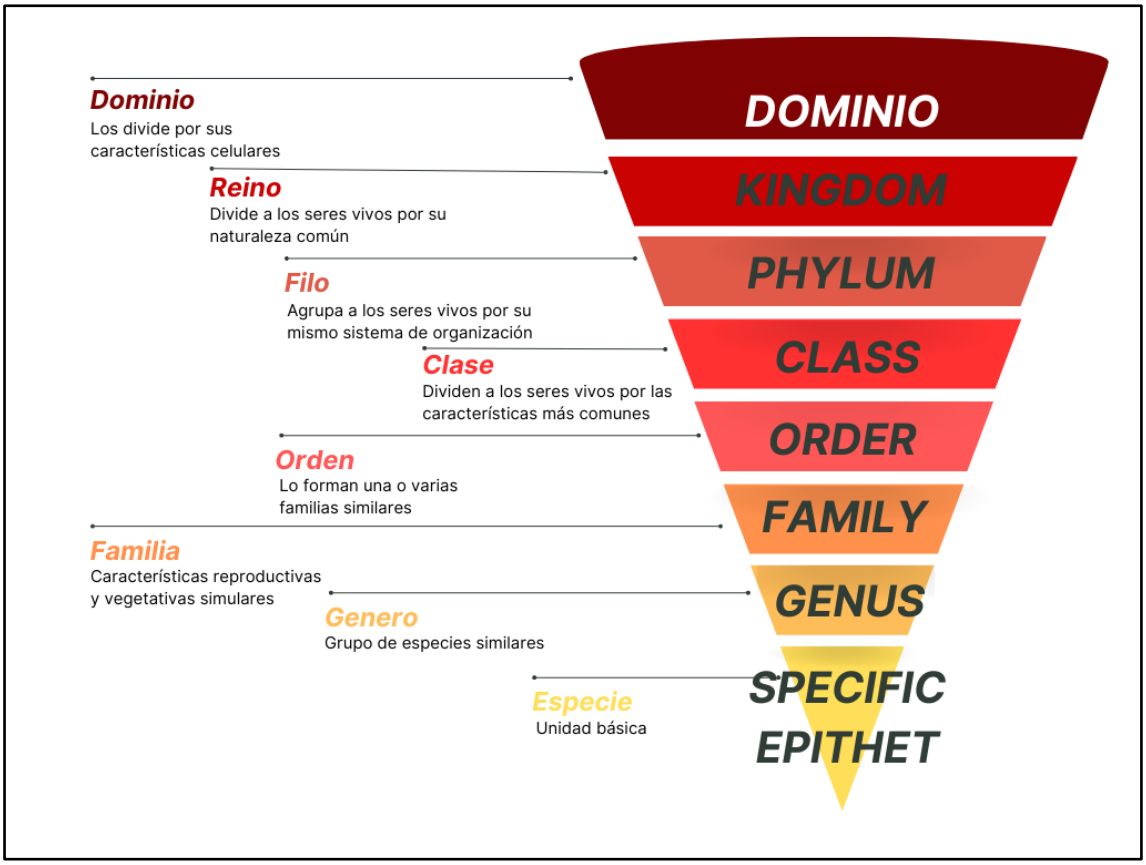

Figura 11 Clasificación taxonómica

Cada grupo que aparece en la figura anterior es un taxón. La taxonomía es una ciencia en constante cambio, en la actualidad la organización taxonómica es la siguiente (microbiologia.net, 2023) (Area de Ciencias, 2023):

- **Dominio**. Es la categoría que agrupa más seres vivos; los divide por sus características celulares. Así, existen 3 dominios en la actualidad: Archaea, Bacteria y Eukarya.
- **Reino**. Divide a los seres vivos por su naturaleza común. Archaea y Bacteria son reinos que coinciden además con dominios, son unicelulares, procariontes y tienen otras características bioquímicas y biofísicas que las hacen diferentes. El dominio Eukarya se divide en varios reinos:

  - *Fungi*. Organismos heterótrofos de pared celular compuesta de quinita. Son setas, mohos y levaduras.
  - *Plantae*. Organismos autótrofos, con pared celular compuesta mayoritariamente de celulosa y sin capacidad de locomoción.
  - *Animalia*. Organismos heterótrofos, sin pared celular y con capacidad de locomoción.
- **Filo**. Agrupa a los seres vivos por su mismo sistema de organización. Ejemplo: en el reino animal, los bivalvos, los gasterópodos y los cefalópodos tienen el mismo tipo de tejidos, reproducción, órganos y sistemas, por lo tanto, se agrupan en el filo Mollusca (moluscos).
- **Clase**. Dividen a los seres vivos por las características más comunes que hay entre miembros de un filo. Por ejemplo: la clase de los mamíferos incluye todos los mamíferos que son los murciélagos, roedores, canguros, ballenas, grandes simios y el hombre.
- **Orden**. Lo forman una o varias familias similares. El orden al que pertenece el ser humano, por ejemplo, es el orden de los primates, que comparte con los monos y los lemures. Todos los felinos están incluidos en el orden Carnívoros.
- **Familia**. Se puede agrupar varios géneros por características reproductivas y vegetativas simulares. Por ejemplo, los gatos y el leopardo se incluyen en la familia de los felinos.
- **Género**. Se define como grupo de especies similares. Los perros no pueden reproducirse con los chacales porque no son de la misma especie, pero son lo suficientemente parecidos como para formar parte de un mismo género: canis.
- **Especie**. Es la unidad básica, el grupo de individuos que tiene capacidad de reproducción con descendencia fértil.

Dentro de nuestro trabajo visualizaremos únicamente los 7 últimos niveles de la organización taxonómica. Nuestro conjunto de datos contiene únicamente imágenes de especies que pertenecen al dominio Eukarya, que está formado por los reinos Fungi, Plantae y Animalia.

5.1.2 Inteligencia Artificial

Históricamente la Inteligencia Artificial se ha planteado de diferentes puntos de vista: la capacidad de pensar o la habilidad de actuar de forma inteligente (Russell & Norvig, 2016). Un primer planteamiento más próximo a una idea humana de inteligencia, en el que las máquinas pienses y sean racionales. Una segunda aproximación se basa en el resultado obtenido en lugar de basarse en el proceso, considerando inteligencia artificial a la capacidad de actuar y emular el resultado de una acción estrictamente racional.

Parte fundamental de la inteligencia reside en el aprendizaje, proceso en el cual, a través de la información, estudio y experiencia se adquiere una formación determinada. El procesamiento de datos para lograr la identificación de patrones que permitan el elaborar predicciones cada vez más perfeccionadas, basándose estos algoritmos en un conocimiento estadístico complejo es lo que conocemos como aprendizaje automático o machine learning.

Arthur Samuel (1959, IBM): acuñó el término machine learning o aprendizaje automático como el campo de estudio que confiere a los ordenadores la capacidad de aprender sin ser programados explícitamente.

En los últimos años el aprendizaje automático ha evolucionado a lo que conocemos como aprendizaje profundo o deep learning. El aprendizaje profundo entiende el mundo como una jerarquía de conceptos (Goodfellow, 2016), diluyendo la información en diferentes capas mediante el uso de módulos, que transforman su representación en un nivel más alto y abstracto. Esto permite la amplificación de la información relevante y eliminar la superflua (LeCun, Bengio, & Hinton, 2015). Históricamente, el concepto de aprendizaje profundo se originó en la investigación sobre redes neuronales artificiales (Deng, 2014).

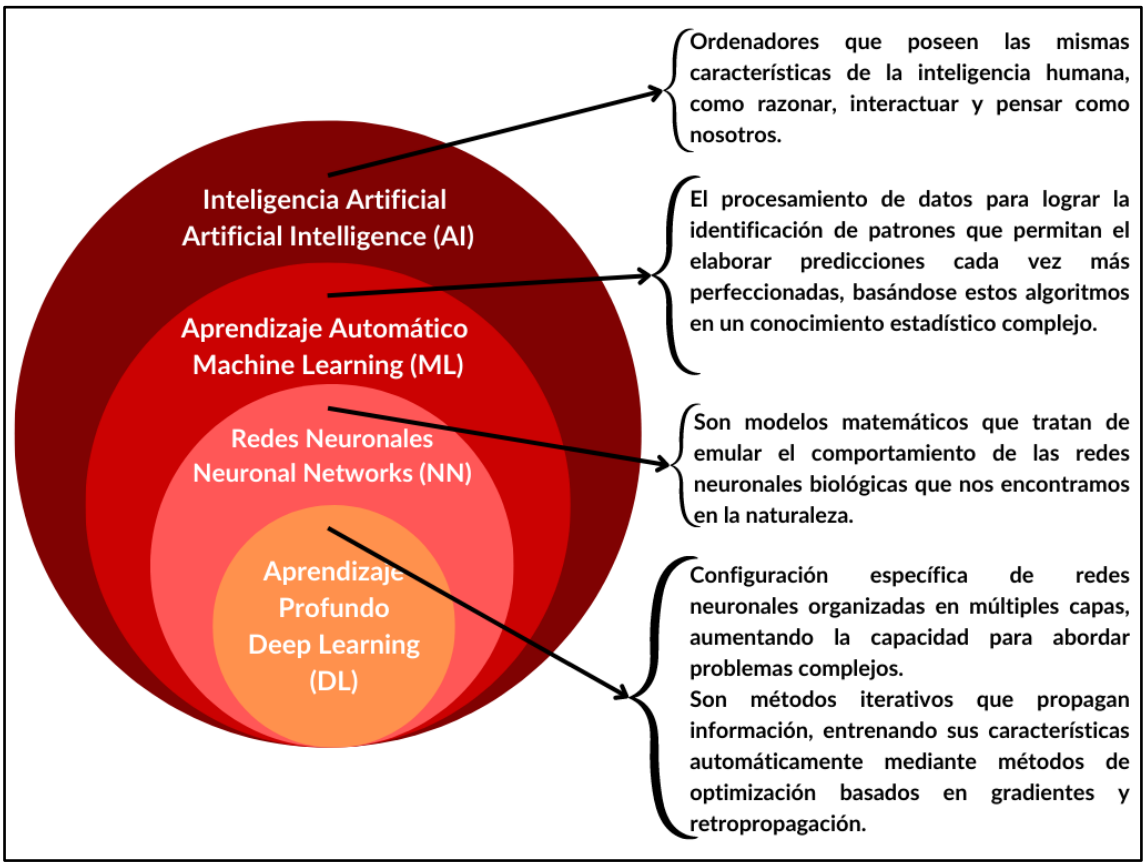

Figura 12 Inteligencia Artificial y Aprendizaje Profundo

5.1.3 Aprendizaje Automático

La premisa principal del Aprendizaje automático es introducir algoritmos que ingieran datos de entrada, apliquen análisis informáticos para predecir valores de salida dentro de un rango aceptable de precisión, identifiquen patrones y tendencias en los datos y, por último, aprendan de la experiencia previa (Handelman, y otros, 2018).

Uno de los conceptos importantes del aprendizaje automático es el aprendizaje supervisado y no supervisado.

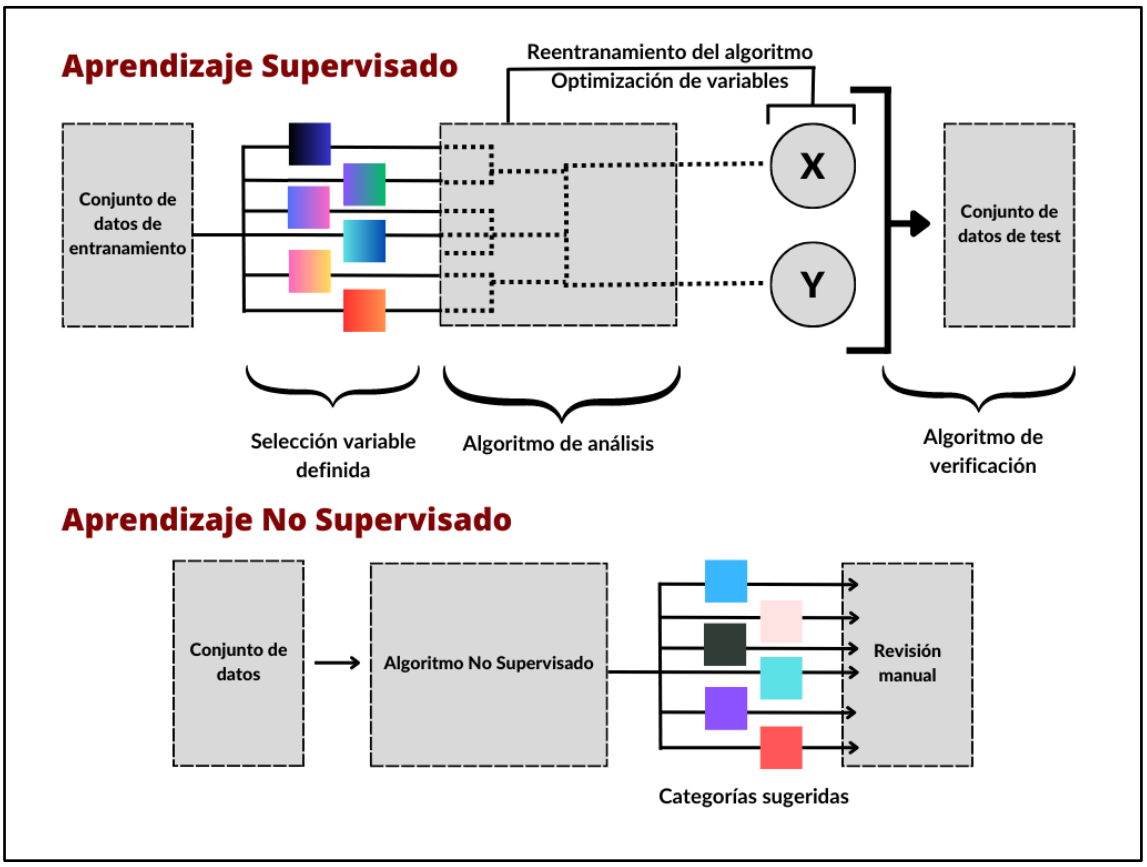

Figura 13 Aprendizaje supervisado vs No supervisado

Aprendizaje supervisado

En el aprendizaje supervisado, se proporcionan características relacionadas con el objetivo de aprendizaje y las medidas de resultado deseadas que deben alcanzarse con el objetivo de identificar vínculos entre ambos en el conjunto de datos.

Las técnicas de aprendizaje supervisado se centran principalmente en la clasificación, para identificar categorías (subpoblaciones) de una nueva observación, basándose en un conjunto de datos de entrenamiento que comprende observaciones cuya pertenencia a una categoría es conocida. Una vez creado y optimizado el modelo, se prueba con datos no incluidos en los datos de entrenamiento para determinar su validez externa y,

por tanto, su aplicabilidad. Es por ello, que el aprendizaje supervisado requiere la intervención humana.

Aprendizaje No supervisado

En el aprendizaje no supervisado, se proporcionan registros de datos sin clasificar para que reconozca y determine si existen patrones latentes, lo que a veces produce tanto respuestas como preguntas que pueden no haber sido concebidas por los investigadores. Desde un punto de vista técnico, mientras que el aprendizaje supervisado se ocupa principalmente de los problemas de clasificación y regresión, el aprendizaje no supervisado se ocupa más de la agrupación y la reducción de la dimensionalidad. Los patrones identificados en el aprendizaje no supervisado suelen tener que ser evaluados para determinar su utilidad. Por tanto, los algoritmos no supervisados no precisan la supervisión humana.

El clustering se refiere a la identificación de grupos dentro de los datos, es decir, al algoritmo se le proporcionan los datos, los analiza y determina cualquier similitud latente dentro de los datos que permita agrupar a los sujetos en subsecciones y descubrir patrones dentro de los datos.

5.1.4 Red Neuronal Artificial

Las redes neuronales artificiales son modelos matemáticos que tratan de emular el comportamiento de las redes neuronales biológicas que nos encontramos en la naturaleza.

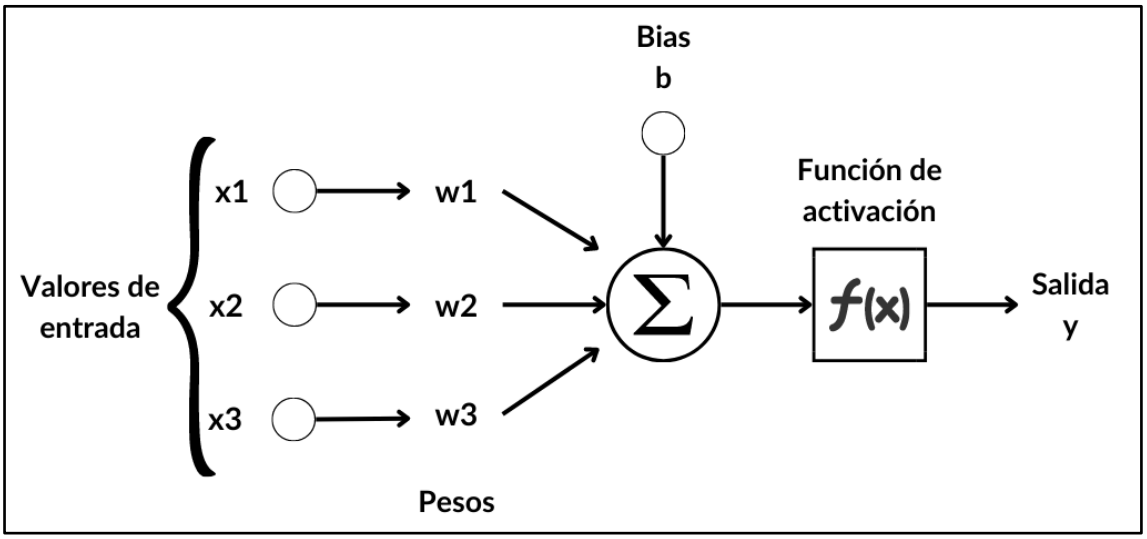

Figura 14 Esquema neurona artificial

Una neurona artificial es una unidad lógica que recibe varios inputs o valores de entrada, con sus correspondientes pesos, aplica una función de activación y emite un resultado que se propaga a la siguiente capa.

Durante la fase de entrenamiento de una red neuronal los parámetros pesos y bias son reajustados con el fin de adaptar el modelo y mejorar las predicciones. La función de activación será seleccionada de acuerdo al problema que se pretende resolver.

Estas neuronas artificiales a su vez, agrupan en diferentes niveles o capas. Cuentan con una capa de entrada, que está formada por las neuronas que se encargan de introducir la información a la red; varias capas ocultas, cuyo número de capas puede variar y, por último, tenemos la capa de salida que es la que se encarga de entregar el resultado.

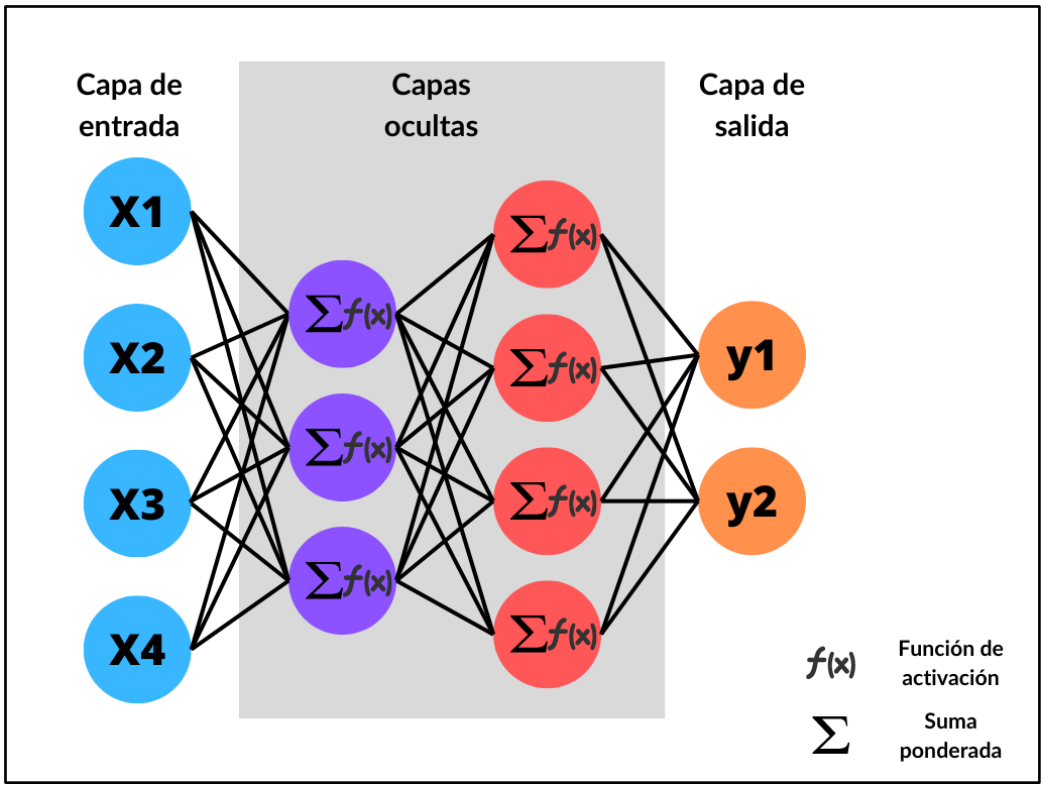

Figura 15 Esquema red neuronal multicapa

5.1.5 Red Neuronal Convolucional (CNN)

Las redes neuronales convolucionales o convolutional neural networks (CNN) es una arquitectura que ha estado presente en toda la revolución que se ha vivido en el campo del deep learning.

Son un tipo de red neuronal especialmente diseñada para aprender a analizar los patrones más complejos de una imagen. Son especialmente útiles en problemas de visión por computador, y en particular, en el reconocimiento de objetos.

Reciben este nombre porque su actividad se modeliza a partir de la operación matemática de convolución, utilizada ampliamente en la edición y procesamiento de imágenes. Ésta consiste en la aplicación de un filtro, denominado kernel, al conjunto de datos de entrada, multiplicaremos y sumaremos los valores de cada pixel vecino, obteniendo así el nuevo valor del pixel.

Los valores del filtro serán los que la red neuronal irá aprendiendo poco a poco para hacer cada vez mejor su tarea. A cada uno de las imágenes generadas se le conoce como mapa de características. Esta operación se va a realizar secuencialmente, donde el output de una capa se va a convertir en el input de la siguiente.

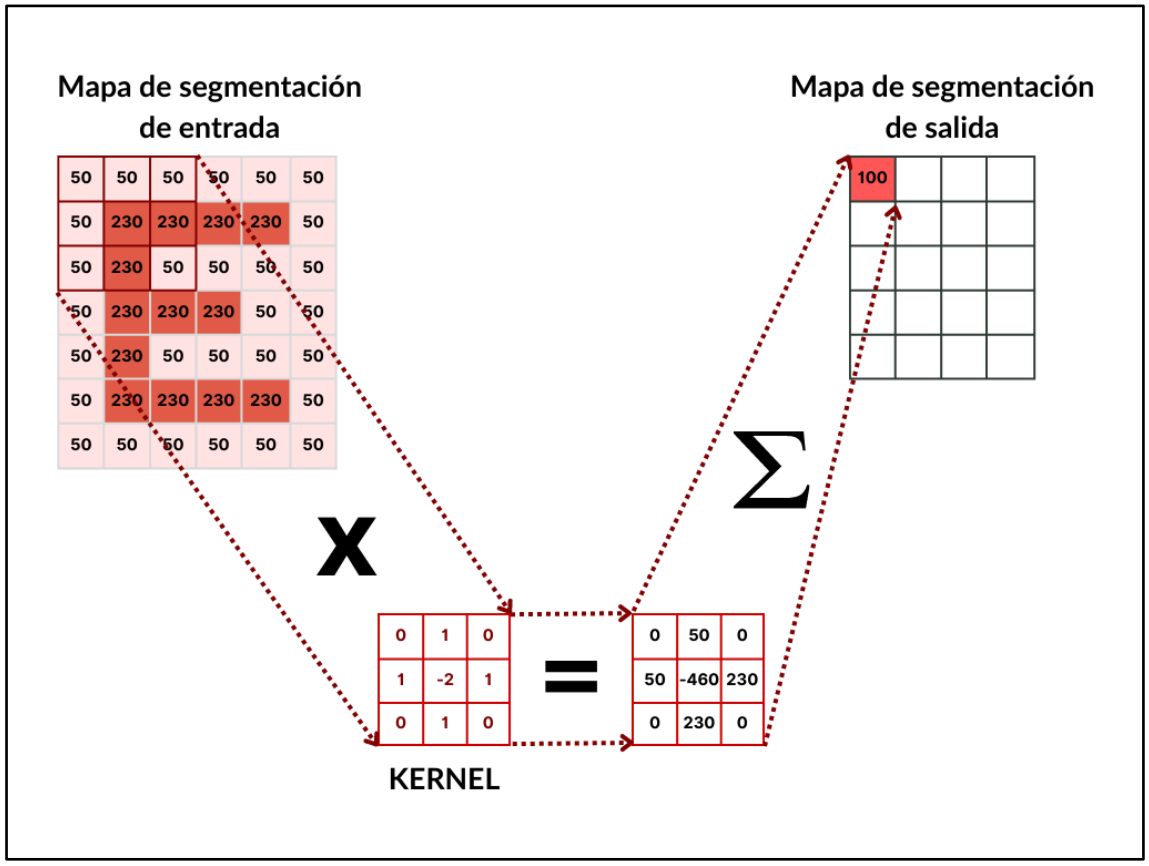

Figura 16 Operación de convolución

La convolución se va a ir haciendo cada vez más potente, si tomamos el ejemplo de la figura anterior, en la que se aplica un filtro de 3x3, ahora donde teníamos una región de 9 píxeles nuestro filtro lo ha convertido en un único pixel de información, si ahora volvemos a aplicar una nueva convolución a estos mapas de características, estaremos accediendo cada vez a más información espacial de la imagen original. Lo que nos permite componer cada vez patrones más complejos, el número de mapas de características va en aumento. El único entrenamiento que se realiza es el de aprender los filtros; la red aprende automáticamente esta idea jerárquica, donde los primeros

filtros son muy básicos y luego por composición se van encontrando cada vez filtros más abstractos.

La arquitectura básica de una CNN consta de dos partes: Un proceso de extracción de características y un proceso de clasificación que predice la clase de la imagen basándose en las características extraídas en las etapas anteriores (Gurucharan, 2022).

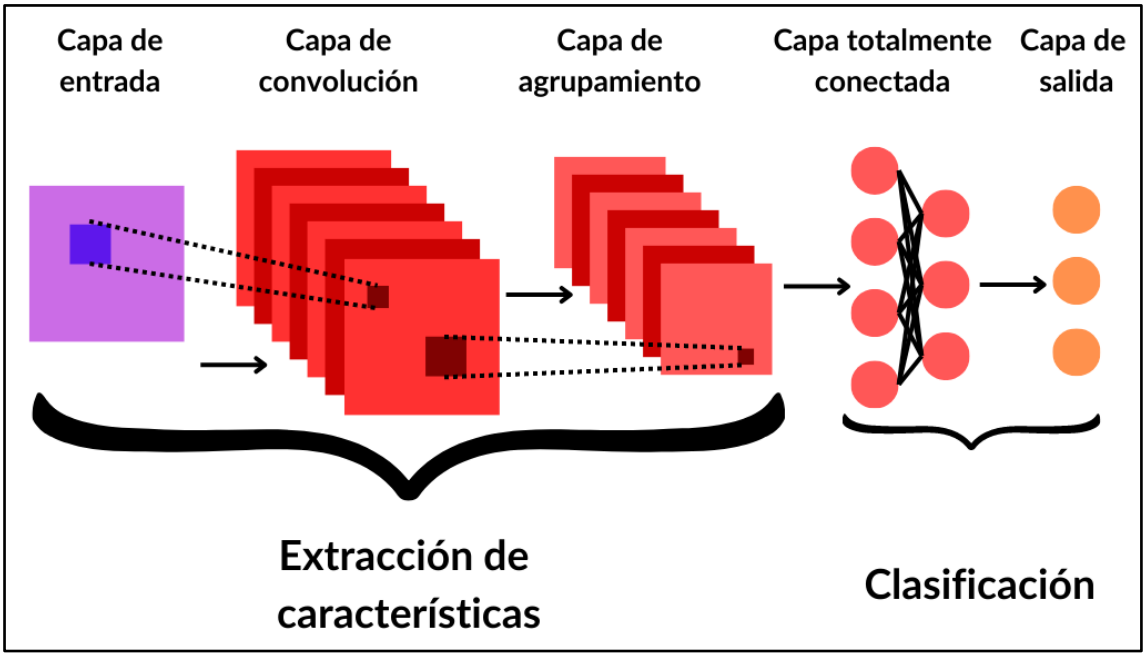

Figura 17 Arquitectura CNN

A continuación, definimos las distintas capas que aparecen en la arquitectura anterior (Kumar, 2023):

- **Capa convolucional**: Las capas convolucionales funcionan deslizando un conjunto de "filtros" o "núcleos" por los datos de entrada. Cada filtro está diseñado para detectar una característica o patrón específico, como bordes, esquinas o formas más complejas en el caso de capas más profundas. A medida que estos filtros se desplazan por la imagen, generan un mapa que indica las zonas en las que se han encontrado esas características. La salida de la capa convolucional es un mapa de características, que es una representación de la imagen de entrada con los filtros aplicados. Las capas convolucionales pueden apilarse para crear modelos más complejos, que pueden aprender características más intrincadas de las imágenes.
- **Capa de agrupamiento**: Las capas de agrupamiento siguen a las capas convolucionales y se utilizan para reducir el tamaño espacial de la entrada, lo que facilita su procesamiento y requiere menos memoria. Al reducir las dimensiones, las capas de agrupamiento ayudan a reducir el número de

parámetros o pesos de la red. Esto ayuda a combatir el sobreajuste y a entrenar el modelo de forma rápida. Hay dos tipos principales de agrupamiento: agrupamiento máximo y agrupamiento medio. El agrupamiento máximo toma el valor máximo de cada mapa de características. Por ejemplo, si el tamaño de la ventana de agrupación es de 2×2, elegirá el píxel con el valor más alto en esa región de 2×2. La agrupación máxima captura el rasgo o característica más prominente dentro de la ventana de agrupación. El agrupamiento medio calcula la media de todos los valores de la ventana de agrupamiento. Proporciona una representación suave y media de las características.

- **Capa totalmente conectada**: Las capas totalmente conectadas son uno de los tipos más básicos de capas en una red neuronal convolucional (CNN). Como su nombre indica, cada neurona de una capa totalmente conectada está totalmente conectada a todas las neuronas de la capa anterior. Las capas totalmente conectadas se suelen utilizar hacia el final de una CNN, cuando el objetivo es tomar las características aprendidas por las capas convolucionales y de agrupación y utilizarlas para hacer predicciones, como clasificar la entrada en una etiqueta.

A continuación, se muestra una cronología aproximada de cómo han mejorado las arquitecturas CNN de código abierto (Kumar, 2023):

Arquitectura	Año	Características principales	Casos de uso
LeNet	1988	Primeras aplicaciones con éxito de CNN, 5 capas (alternando entre convolucional y agrupamiento), utiliza funciones de activación tangente hiperbólica / sigmoide.	Reconocimiento de caracteres manuscritos e impresos a máquina.
AlexNet	2012	Más profunda y amplia que LeNet, utiliza la función de activación ReLU, implementa capas de abandono, utiliza GPU para el entrenamiento.	Tareas de reconocimiento de imágenes a gran escala.
ZFNet	2013	Arquitectura similar a AlexNet, pero con diferentes tamaños y número de filtros, Técnicas de visualización para comprender la red.	Clasificación ImageNet
VGGNet	2014	Redes más profundas con filtros más pequeños (3×3), Todas las capas convolucionales tienen la misma profundidad. Múltiples configuraciones (VGG16, VGG19)	Reconocimiento de imágenes a gran escala.
GoogleLeNet	2014	Se introduce el módulo Inception, que permite un cálculo más eficiente y redes más profundas, múltiples versiones (Inception v1, v2, v3, v4).	Reconocimiento de imágenes a gran escala, obtuvo el 1er puesto en el ILSVRC 2014.

Arquitectura	Año	Características principales	Casos de uso
ResNet	2015	Introducción de "conexiones de salto" o "atajos" para permitir el entrenamiento de redes más profundas, Múltiples configuraciones (ResNet-50, ResNet-101, ResNet-152).	Reconocimiento de imágenes a gran escala, obtuvo el 1er puesto en el ILSVRC[2] 2015.
ResNeXt	2016	Transformaciones residuales agregadas para redes neuronales profundas. Se construye repitiendo un bloque de construcción que agrega un conjunto de transformaciones con la misma topología.	Obtuvo el segundo puesto en el ILSVRC 2016.
DenseNet	2017	La red convolucional densa (DenseNet), que conecta cada capa con todas las demás de forma feed-forward. Similar precisión con la mitad de parámetros.	Premio a la mejor ponencia-CVPR2017
MobileNets	2017	Diseñado para aplicaciones de visión móviles e integradas. Utiliza convoluciones separables en profundidad para reducir el tamaño y la complejidad del modelo.	Aplicaciones de visión móviles e integradas, detección de objetos en tiempo real.
EfficientNets	2019	Nuevo método de escalado que escala uniformemente todas las dimensiones de profundidad/anchura/resolución utilizando un coeficiente compuesto sencillo pero muy eficaz.	Reconocimiento de imágenes a gran escala.

Tabla 1 Resumen arquitecturas CNN de código abierto

5.2 Herramientas utilizadas

5.2.1 Herramientas Software

El lenguaje de programación que hemos elegido para realizar el desarrollo ha sido Python.

Python es un lenguaje de programación muy versátil, utilizado en una amplia variedad de campos. Cuenta con una gran biblioteca estándar que contiene códigos reutilizables. Sus principales características son las siguientes (Amazon):

- Es un lenguaje interpretado, lo que significa que ejecuta directamente el código, línea por línea. Si existen errores en el código del programa, su ejecución se detiene.
- Un lenguaje fácil de utilizar. Python utiliza palabras similares a las del inglés. A diferencia de otros lenguajes de programación, Python no utiliza llaves. En su lugar, utiliza sangría.
- Un lenguaje tipeado dinámicamente. Los programadores no tienen que anunciar tipos de variables cuando escriben código porque Python los determina en el tiempo de ejecución.

[2] **ILSVRC**: ImageNet Large Scale Visual Recognition Challenge

- Un lenguaje orientado a los objetos. Python considera todo como un objeto, pero también admite otros tipos de programación, como la programación estructurada y la funcional.

Entre los casos de uso en el desarrollo de aplicaciones en Python, nos encontramos:

Desarrollo web del lado del servidor

El desarrollo web del lado del servidor incluye las funciones complejas de backend que los sitios web llevan a cabo para mostrar información al usuario. Por ejemplo, los sitios web deben interactuar con las bases de datos, comunicarse con otros sitios web y proteger los datos cuando se los envía a través de la red.

Python es útil para escribir código del lado del servidor debido a que ofrece muchas bibliotecas que constan de código prescrito para crear funciones de backend complejas. Los desarrolladores también utilizan un amplio rango de frameworks de Python que proporcionan todas las herramientas necesarias para crear aplicaciones web con mayor rapidez y facilidad. Por ejemplo, los desarrolladores pueden crear la aplicación web esqueleto en segundos porque no deben escribirla desde cero. Pueden probarla por medio de las herramientas de prueba del framework, sin depender de herramientas de prueba externas.

Realizar tareas de data science y machine learning

Python tiene potentes librerías que permiten realizar tareas como las que se indican a continuación:

- Limpieza de datos. Corregir y eliminar datos incorrectos.
- Extraer y seleccionar varias características significativas de los datos.
- Visualización de los datos mediante tablas y gráficos.
- Entrenamiento de modelos de Deep Learning para efectuar tareas de clasificación de imágenes, reconocimiento del habla, reconocimiento facial, clasificación de textos entre otros.

5.2.2 Python frameworks

En el campo de la programación, cuando hablamos de qué es un framework, lo posicionamos en un marco de trabajo que tiene como objetivo facilitar y solventar problemas que pueden aparecer en la programación de forma frecuente.

Pytorch

Los dos frameworks más extendidos a día de hoy en el campo del aprendizaje profundo son Tensorflow y Pytorch. Las razones que nos han llevado a seleccionar Pytorch como el farmework a utilizar son las siguientes:

- *La facilidad de uso*. Lo podemos entender desde dos aspectos:
    1. La forma de crear las redes neuronales y la lógica dentro de las mismas es mucho más parecida a Python que en Tensorflow.
    2. La propia API es mucho más intuitiva en Pytorch que en Tersorflow.
- *Pytorch es mucho más fácil de depurar el código que en Tensorflow*. En pytorch el gráfico computacional es dinámico, por lo que puede ir cambiando, mientras que en Tensorflow es estático. Unido al hecho de que es más parecido a Python hace que sea más fácil el depurar el código.
- *Pytorch es mucho más utilizado hay en dia que Tensorflow*. Pytorch ha ganado mucha popularidad en los últimos años.

El framework más popular para machine learning se ha creado sobre la biblioteca Torch, que es a su vez una biblioteca de machine learning de código abierto. Permite escribir de forma ágil código en Python de machine learning (capaz de ejecutarse en una o varias GPUs). Capaz de acceder a muchos modelos de aprendizaje profundo preconstruidos (Torch HUB / torchvision.models). Abarca toda la pila: preprocesamiento de los datos, datos del modelo, despliegue del modelo en la aplicación/nube. Originalmente diseñado y utilizado internamente por Meta/Facebook, actualmente es código opensource y utilizado por compañías como Testa, Microsoft, OpenAI, …

Cuadernos de Jupyter

Los ficheros cuya extensión es *.ipynb* son ficheros de un proyecto de código abierto que permite combinar fácilmente anotaciones de texto y código fuente Python ejecutable dentro del propio cuaderno. Para la ejecución de todo el código relacionado con la parte de aprendizaje profundo y CNN hemos utilizado Visual Studio Code, que permite trabajar con Jupyter Notebooks de forma nativa.

Flask

(Python Tutorial, 2021) Flask es un framework escrito en Python que permite desarrollar aplicaciones web fácilmente. Tiene un núcleo pequeño y fácil de extender: es un microframework que no incluye un ORM (Object Relational Manager) o características similares. Se basa en el conjunto de herramientas WSGI Werkzeg y el motor de plantillas

Jinja2, ambos proyectos de Pocco (grupo internacional de entusiastas de Python, liderado por Armin Ronacher).

Werkzeug es un conjunto de herramientas WSGI[3] que implementa peticiones, objetos de respuesta y funciones de utilidad. Esto permite construir un marco web sobre él.

Jinja2 es un motor popular de plantillas para Python. Un sistema de plantillas web combina una plantilla con una fuente de datos específica para renderizar una página web dinámica.

5.2.3 Entorno de trabajo

El proyecto lo hemos desarrollado en un ordenador personal, con las siguientes características:

- Procesador: Intel Core i7 8700
- Memoria RAM: 16 GB DDR4
- GPU: nVidia RTX 3060 (12GB VRAM)
- Sistema Operativo: Windows 10 Pro 64 bits
- Python vers. 3.11.1
- Visual Studio Code

5.3 Análisis del conjunto de datos

Hemos decidido utilizar el conjunto de datos del iNat Challenge 2.021 para entrenar y validar nuestro modelo. Contiene imágenes de 10.000 especies diferentes. El conjunto completo de datos de entrenamiento contiene casi 2,7 millones de imágenes. Para que el conjunto de datos sea más accesible, existe un subconjunto de datos de entrenamiento mini, formado por 500.000 imágenes, que contiene 50 imágenes de cada una de las 10.000 especies catalogadas. El conjunto de datos de validación contiene para cada especie 10 imágenes, sumando un total de 100.000 imágenes.

Si utilizamos el conjunto de entrenamiento mini, tendremos que el 80% de los datos forman parte del conjunto de datos de entrenamiento y el 20% de los datos son el conjunto de validación. En ambos casos todas las especies se encuentran representadas de forma proporcional, por lo que podemos decir que nuestros conjuntos de entrenamiento y validación están balanceados. Evitamos así uno de los problemas más habituales en el entrenamiento de redes neuronales.

[3] **WSGI**: Web Server Gateway Interface

A su vez las 10.000 especies se encuentran agrupadas de la siguiente forma:

Reino	Filo	Nº Especies	Nº imágenes entrenamiento	Nº imágenes validación
Animalia	Annelida	4	200	40
	Arthropoda	2.752	137.600	27.520
	Chordata	2.416	120.800	24.160
	Cnidaria	26	1.300	260
	Echinodermata	21	1.050	210
	Mollusca	169	8.450	1.690
Fungi	Ascomycota	84	4.200	840
	Basidiomycota	257	12.850	2.570
Plantae	Bryophyta	35	1.750	350
	Chlorophyta	4	200	40
	Marchantiophyta	7	350	70
	Rhodophyta	7	350	70
	Tracheophyta	4.218	210.900	42.180
Total		**10.000**	**500.000**	**100.000**

Tabla 2 Distribución del conjunto de datos

Todas las fotografías se han almacenado en formato JPEG, las dimensiones de las mismas varían dentro de los siguientes rangos, el alto entre un mínimo de 63 píxeles y máximo 500 píxeles; mientras que el ancho varía entre los 61 y los 500 píxeles. Por lo tanto, no es un conjunto homogéneo de imágenes, en lo que a tamaño se refiere.

Uno de los problemas de identificar al mismo tiempo individuos de diferentes reinos es que en algunas imágenes aparecen al mismo tiempo diferentes elementos pertenecientes a diferentes reinos. También podemos encontrar imágenes en las que el individuo es apenas perceptible en la imagen y predominan otros elementos.

A continuación, mostramos un ejemplo de 9 imágenes que forman parte del conjunto de datos de entrenamiento, en la parte superior de la imagen se muestra el nombre común del individuo. En estas imágenes podemos observar algunos de los problemas comentados. El tamaño de las imágenes ha sido previamente ajustado, para que el ejemplo se pueda visualizar de forma correcta.

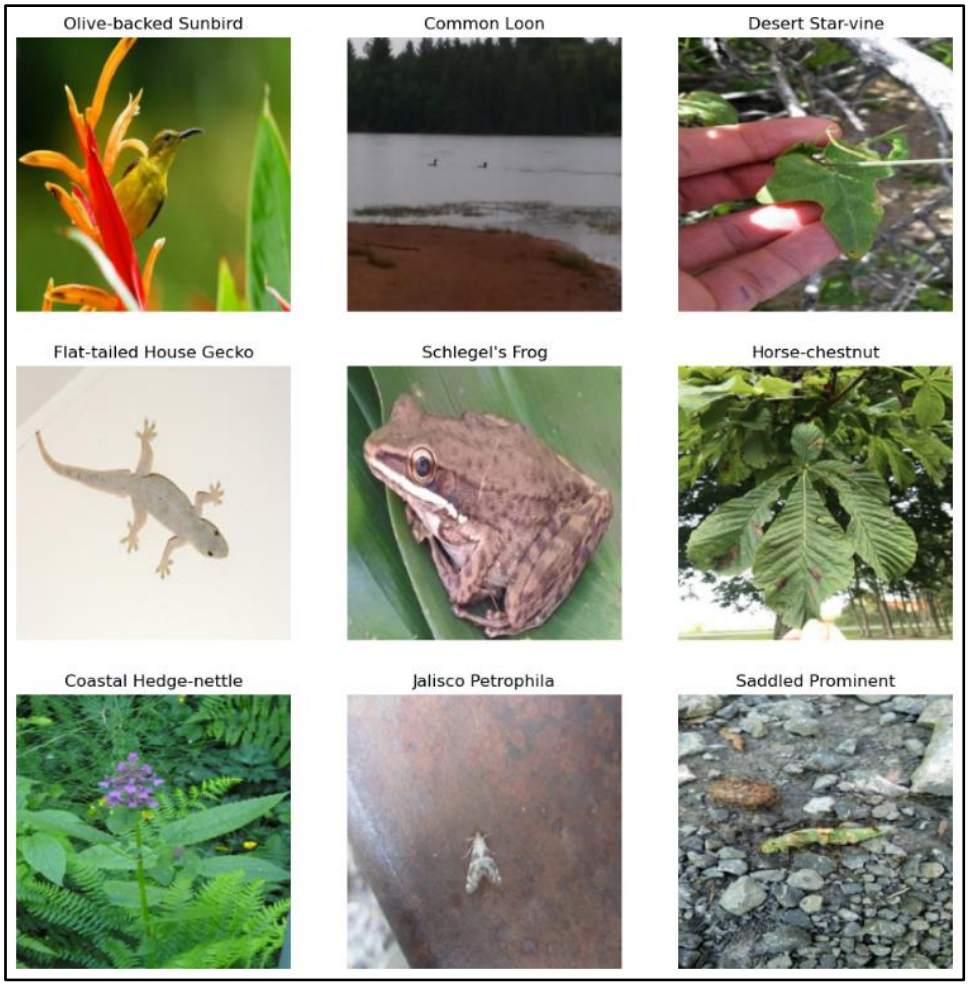

Figura 18 Ejemplo de imágenes del conjunto de datos

Acompañando a cada uno de los conjuntos de datos tenemos un fichero de anotaciones almacenadas en formato JSON. Hemos respetado el conjunto de datos original y por lo tanto hemos mantenido los nombres que aparecen en este fichero. Concretamente nos hemos centrado en el diccionario de categorías que aparece en este fichero y es el que nos ha servido para establecer la clasificación taxonómica en 7 niveles de cada una de las diferentes especies. Además de los siete niveles de la clasificación taxonómica nos aportará también el nombre común de cada una de las especies.

En uno de nuestros cuadernos de Jupyter hemos analizado el fichero JSON y extraído la información que acabamos de comentar a un dataframe de pandas. El contenido del dataframe se almacena en un fichero CSV, separado por tabuladores; posteriormente utilizaremos este fichero CSV en la función del módulo metadata. A continuación, incluimos la ejecución del código Python en el cuaderno junto con el resultado obtenido a la salida:

```
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
[7]: with open('./data/train_mini.json') as data_file:
         data = json.load(data_file)

     data.keys()
```

```
[7]: dict_keys(['info', 'images', 'categories', 'annotations', 'licenses'])
```

```
[8]: df = pd.json_normalize(data['categories'])
     df.dtypes
```

```
[8]: id                 int64
     name              object
     common_name       object
     supercategory     object
     kingdom           object
     phylum            object
     class             object
     order             object
     family            object
     genus             object
     specific_epithet  object
     image_dir_name    object
     dtype: object
```

```
[9]: df = df[['common_name', 'kingdom', 'phylum', 'class', 'order', 'family', 'genus', 'specific_epithet']]
```

```
[10]: df.to_csv('categoriasiNat.txt', sep='\t', index=False)
```

Figura 19 Cuaderno Jupyter. Tratamiento fichero JSON

5.4 Preprocesamiento de los datos

Antes de entrenar nuestro modelo aplicaremos una serie de transformaciones a los dos conjuntos de datos, el de entrenamiento y el de validación.

En primer lugar, vamos a redimensionar todas las imágenes para que todas sean del mismo tamaño. Todas las imágenes de nuestro conjunto de datos se transforman en imágenes de 224 píxeles de alto por 224 píxeles de ancho.

Hemos aplicado dos transformaciones más, la primera de ellas es el voltear la imagen horizontalmente y la segunda el girar la imagen 10 grados, o bien hacia la derecha, o hacia la izquierda. Estas dos trasformaciones se aplican de forma aleatoria, es decir, no todas las imágenes se volean y se giran, conforme se va cargando el conjunto de datos se aplica la transformación o no de forma aleatoria.

La siguiente transformación es convertir nuestro conjunto de datos en un tensor. Los tensores son una estructura de datos especializada muy similar a las matrices y vectores. En PyTorch, usamos tensores para codificar las entradas y salidas de un modelo, así como los parámetros del modelo. Los tensores pueden ejecutarse en GPUs u otros aceleradores de hardware (pytorch.org, 2023).

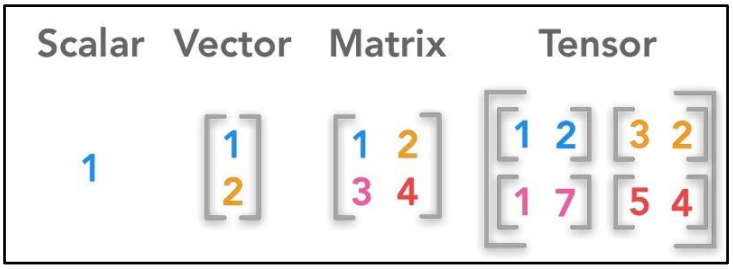

Figura 20 Tensor

Por último, normalizaremos cada canal de color por separado. Para las medias, es [0,485, 0,456, 0,406] y para las desviaciones estándar [0,229, 0,224, 0,225], estos valores desplazarán cada canal de color para que esté centrado en 0 y oscile entre -1 y 1. Estos parámetros de normalización se han establecido en base a las imágenes del conjunto de datos ImageNet.

```
[3]: data_transforms = {
        'train': transforms.Compose([
            transforms.Resize(img_size),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(degrees=10),
            transforms.ToTensor(),
            transforms.Normalize(mean, std)
        ]),
        'val': transforms.Compose([
            transforms.Resize(img_size),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(degrees=10),
            transforms.ToTensor(),
            transforms.Normalize(mean, std)
        ]),
    }
```

Figura 21 Cuaderno Jupyter. Código para la transformación de las imágenes

Cuando construimos y entrenamos un modelo de aprendizaje profundo en Pytorch, este modelo se comporta como una función que recibe un tensor Pytorch y devuelve otro tensor a la salida. Lo más fácil es preparar un gran tensor de todo el conjunto de datos que almacene la imagen y su etiqueta correspondiente.

```
image_datasets = {
    x: datasets.INaturalist(
        root = data_dir,
        version = '2021_train_mini' if x == 'train' else '2021_valid',
        transform = data_transforms[x],
        download = False
    )
    for x in ['train', 'val']
}
```

Figura 22. Cuaderno Jupyter construcción del conjunto de datos

Este código lo que hace es descargar del repositorio de TorchVision el conjunto de imágenes y almacenarlas en el directorio especificado por la variable data_dir. Al conjunto de datos le aplica las transformaciones que hemos definido en el paso anterior. Recordemos que una de las transformaciones que hace es convertir la imagen en un tensor. El parámetro download, nos permite indicar si queremos que descargue siempre el fichero del repositorio, True. Si el parámetro lo igualamos a False, previamente comprueba si el conjunto de imágenes está guardado en el directorio data_dir, en cuyo caso comprueba la integridad de los datos y no los volvería a descargar.

La habitual es que el modelo procese un conjunto de varias muestras de forma simultánea. Podemos realizar un corte en el tensor y almacenarlo con la misma sintaxis en un lote con un tamaño mucho mas pequeño. Estos lotes podemos generarlo barajando las imágenes de forma que nos garantice que el orden de los datos no va a ser el mismo en cada etapa de entrenamiento. El código que lo implementa es el siguiente:

```
dataloaders = {
    x: torch.utils.data.DataLoader(
        image_datasets[x],
        batch_size = bs,
        shuffle = True,
        num_workers = 4
    )
    for x in ['train', 'val']
}
```

Figura 23 Cuaderno Jupyter generación del DataLoader

5.5 Construcción del modelo

Utilizaremos varios de los modelos incluidos en *TorchVision.models*, que son modelos preconstruidos de las redes convolucionales más extendidas. Esta librería nos ofrece la posibilidad de utilizar el modelo limpio, sin inicializar los parámetros de la red neuronal, o el mismo modelo cuyos parámetros han sido inicializados con el resultado del entrenamiento con el conjunto de datos ImageNet.

Utilizaremos también una técnica de aprendizaje por transferencia, que es una de las técnicas más utilizadas en el aprendizaje profundo. En el aprendizaje por transferencia, se toma un modelo de aprendizaje automático o profundo preentrenado con un conjunto de datos anterior, como ya hemos comentado cogeremos el modelo preentrenado con el conjunto de datos de ImageNet y se utiliza para resolver un problema diferente, en nuestro caso el conjunto de datos de iNaturalist. Esto significa que el uso de métodos de aprendizaje por transferencia puede reducir en gran medida la demanda de datos, ya que los pesos y los sesgos están preajustados y son capaces de funcionar mejor con

sólo una pequeña cantidad de datos ajustando un poco los pesos y los sesgos (Martinez, 2021). Las primeras capas del modelo que son las que se encargan de la clasificación de las imágenes, analizando las características más comunes de las imágenes se mantiene el valor de los parámetros del modelo preentrenado. Únicamente se entrenarán las capas de clasificación, que son las que analizan los detalles más específicos.

5.6 Entrenamiento y validación del modelo

En este punto vamos a mostrar y analizar los resultados de los diferentes experimentos y modelos que hemos realizado durante el proyecto. En todas las pruebas realizadas, tal y como ya hemos indicado se ha utilizado el conjunto de datos de iNaturalist 2021, así el conjunto de entrenamiento está formado por el 80% de los datos y el conjunto de validación, por el 20% restante.

La función de entrenamiento que hemos utilizado ha sido la misma en todos los entrenamientos y para todos los modelos utilizados. Su código se puede consultar en el Anexo A, apartado 9.1.

5.6.1 Primer modelo probado: AlexNet

La arquitectura consta de ocho capas: cinco capas convolucionales y tres capas totalmente conectadas, como se puede observar en la siguiente figura. Tiene un total de 94.639.696 parámetros, todos ellos modificables a lo largo del entrenamiento.

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
AlexNet                                  --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       23,296
│    └─ReLU: 2-2                         --
│    └─MaxPool2d: 2-3                    --
│    └─Conv2d: 2-4                       307,392
│    └─ReLU: 2-5                         --
│    └─MaxPool2d: 2-6                    --
│    └─Conv2d: 2-7                       663,936
│    └─ReLU: 2-8                         --
│    └─Conv2d: 2-9                       884,992
│    └─ReLU: 2-10                        --
│    └─Conv2d: 2-11                      590,080
│    └─ReLU: 2-12                        --
│    └─MaxPool2d: 2-13                   --
├─AdaptiveAvgPool2d: 1-2                 --
├─Linear: 1-3                            92,170,000
==========================================================================
Total params: 94,639,696
Trainable params: 94,639,696
Non-trainable params: 0
==========================================================================
```

Figura 24 Arquitectura de la red AlexNet

Una función de pérdida, es una función que evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje. Cuanto menor es el resultado de esta función, más eficiente es la red neuronal. Mientras que la precisión es el porcentaje de elementos clasificados

correctamente. A nuestra red le pediremos que conforme vamos realizando más etapas de entrenamiento la función de pérdida vaya disminuyendo y que la precisión vaya aumentando.

En la misma gráfica visualizamos el resultado que obtenemos para el conjunto de datos de entrenamiento, la línea de color gris y el conjunto de datos de validación, la línea de color rojo. A la hora de evaluar la bondad de nuestro modelo nos centraremos en la precisión del conjunto de validación.

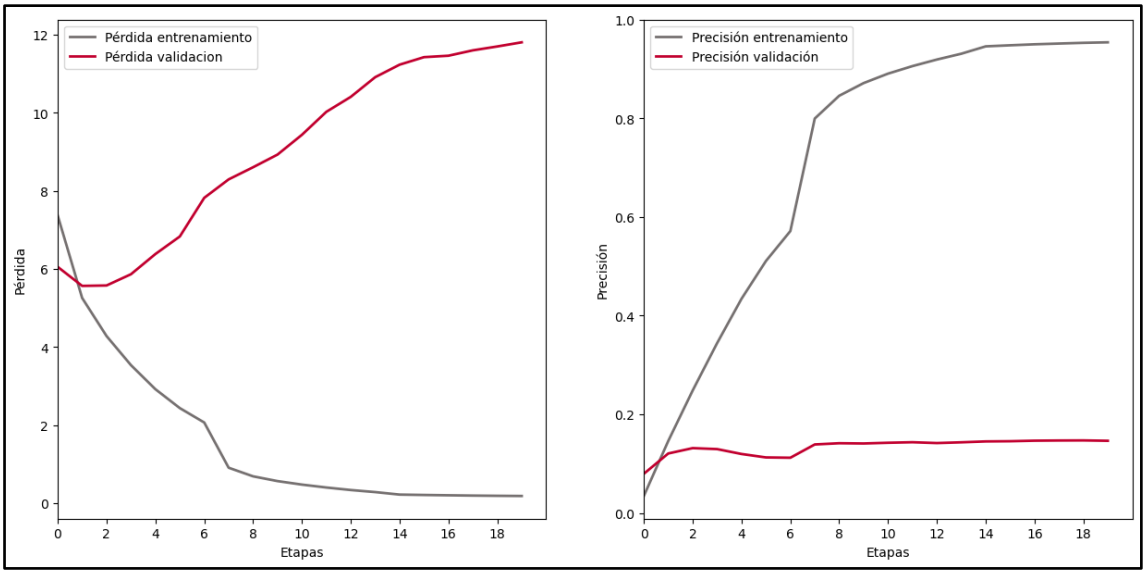

Figura 25 AlexNet, pérdida y precisión

El código con el que hemos generado esta representación gráfica se encuentra en el apartado 9.2 del Anexo A.

La figura anterior muestra las gráficas de perdida y precisión después de realizar 20 etapas o ciclos de entrenamiento, invirtiendo un tiempo total de 5 horas y 56 minutos.

Y el mayor porcentaje de aciertos alcanzado con el conjunto de validación ha sido de 14.71% en la etapa 18 de entrenamiento. A parte de ser un resultado muy pobre, en la gráfica de precisión se observa que apenas mejora su valor desde la octava etapa y que cada vez se distancia más de la precisión del entrenamiento.

5.6.2 ResNet-50

El siguiente modelo de red que hemos probado ha sido el ResNet-50. Debe su nombre a que está formada por 50 capas (48 capas convolucionales, una capa MaxPool y una capa average pool). ResNet significa Red Residual y es un tipo específico de CNN que forma redes apilando bloques residuales, que lo que hacen es presentar un camino

alternativo que se puede tomar o no y que permite saltar unas cuantas capas, como se puede observar en la siguiente figura (Deshpande, 2016):

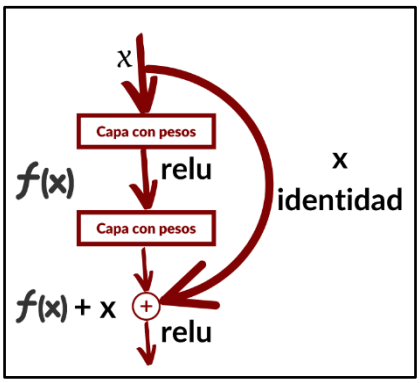

Figura 26 Bloque residual

En esta ocasión TorchVision nos permite descargar el modelo con los parámetros de preetrenamiento de ImageNet v2.

En las gráficas de pérdida y precisión obtenidas en el entrenamiento de este modelo, se puede observar un mejor comportamiento de ambos valores, para el conjunto de validación que cuando utilizamos la red AlexNet.

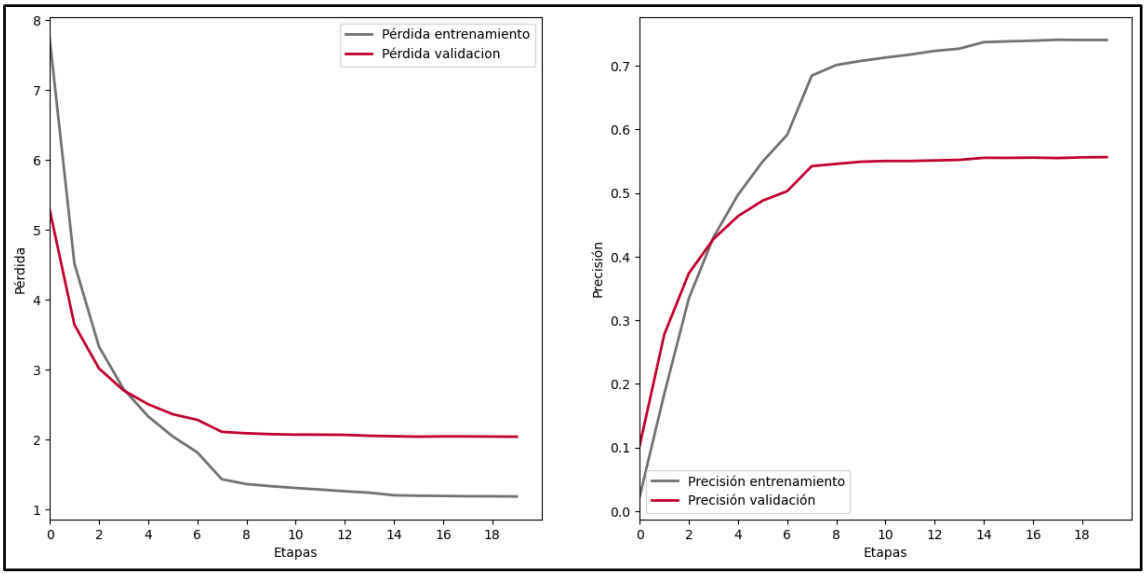

Figura 27 Pérdida y precisión ResNet-50

Al igual que en el entrenamiento del anterior modelo, hemos realizado un entrenamiento que consta de 20 etapas, y en la última etapa de entrenamiento hemos logrado una mayor precisión con el conjunto de validación cuyo valor más alto ha sido el 55,66% de aciertos, en la última etapa del entrenamiento.

5.6.3 EfficientNet-B0

En el año 2019 Google desarrolla una serie de modelos de redes neuronales de código abierto llamados EfficientNet. El modelo básico es EfficientNet-B0 que utiliza la convolución de cuello de botella invertida móvil (MBConv).

EfficientNet utiliza una técnica denominada coeficiente compuesto para escalar modelos de forma sencilla pero eficaz. En lugar de escalar aleatoriamente la anchura, la profundidad o la resolución, el escalado compuesto escala uniformemente cada dimensión con un determinado conjunto fijo de coeficientes de escalado. Utilizando este modelo de escalado han desarrollado 7 modelos.

La siguiente imagen es la arquitectura del modelo base: EfficentNet-B0

```
==========================================================================
Layer (type:depth-idx)                                   Param #
==========================================================================
EfficientNet                                             --
├─Sequential: 1-1                                        --
│    └─Conv2dNormActivation: 2-1                         --
│    │    └─Conv2d: 3-1                                  864
│    │    └─BatchNorm2d: 3-2                             64
│    │    └─SiLU: 3-3                                    --
│    └─Sequential: 2-2                                   --
│    │    └─MBConv: 3-4                                  1,448
│    └─Sequential: 2-3                                   --
│    │    └─MBConv: 3-5                                  6,004
│    │    └─MBConv: 3-6                                  10,710
│    └─Sequential: 2-4                                   --
│    │    └─MBConv: 3-7                                  15,350
│    │    └─MBConv: 3-8                                  31,290
│    └─Sequential: 2-5                                   --
│    │    └─MBConv: 3-9                                  37,130
│    │    └─MBConv: 3-10                                 102,900
│    │    └─MBConv: 3-11                                 102,900
│    └─Sequential: 2-6                                   --
│    │    └─MBConv: 3-12                                 126,004
│    │    └─MBConv: 3-13                                 208,572
│    │    └─MBConv: 3-14                                 208,572
│    └─Sequential: 2-7                                   --
│    │    └─MBConv: 3-15                                 262,492
│    │    └─MBConv: 3-16                                 587,952
│    │    └─MBConv: 3-17                                 587,952
│    │    └─MBConv: 3-18                                 587,952
│    └─Sequential: 2-8                                   --
│    │    └─MBConv: 3-19                                 717,232
│    └─Conv2dNormActivation: 2-9                         --
│    │    └─Conv2d: 3-20                                 409,600
│    │    └─BatchNorm2d: 3-21                            2,560
│    │    └─SiLU: 3-22                                   --
├─AdaptiveAvgPool2d: 1-2                                 --
├─Linear: 1-3                                            12,810,000
==========================================================================
Total params: 16,817,548
Trainable params: 16,817,548
Non-trainable params: 0
==========================================================================
```

Figura 28 Arquitectura EfficentNet B0

Sobre este modelo de CNN hemos realizado tres entrenamientos diferentes:

En primer lugar, hemos utilizado el modelo el modelo sin inicializar los parámetros de la red neuronal, es decir, hemos cargado el modelo sin preentrenamiento. Realizamos un

entrenamiento de 25 etapas y el mayor porcentaje de precisión en con el conjunto de datos de validación lo hemos obtenido en la etapa 23, con un 13,61%.

Para el segundo entrenamiento, cargamos el modelo con los parámetros del preentrenamiento y realizamos un entrenamiento de 20 etapas. En la etapa 18, para el conjunto de validación, obtenemos una precisión del 42,92%.

En nuestro tercer entrenamiento hemos aplicado el aprendizaje por transferencia. Consiste en coger un modelo preentrenado, mantener fijos los valores de las primeras capas, la idea es que las capas convolucionales extraigan características generales de bajo nivel aplicables a todas las imágenes como pueden ser bordes, patrones, gradientes, etc. y que las últimas capas identifiquen características específicas dentro de una imagen. En esta ocasión, realizamos un entrenamiento de 25 etapas, el mejor resultado de precisión para el conjunto de datos de validación fue 10,9%. Como se puede observar el peor resultado de los tres entrenamientos realizados. A continuación, mostramos las gráficas de pérdida y precisión obtenidas al utilizar el modelo preentrenado.

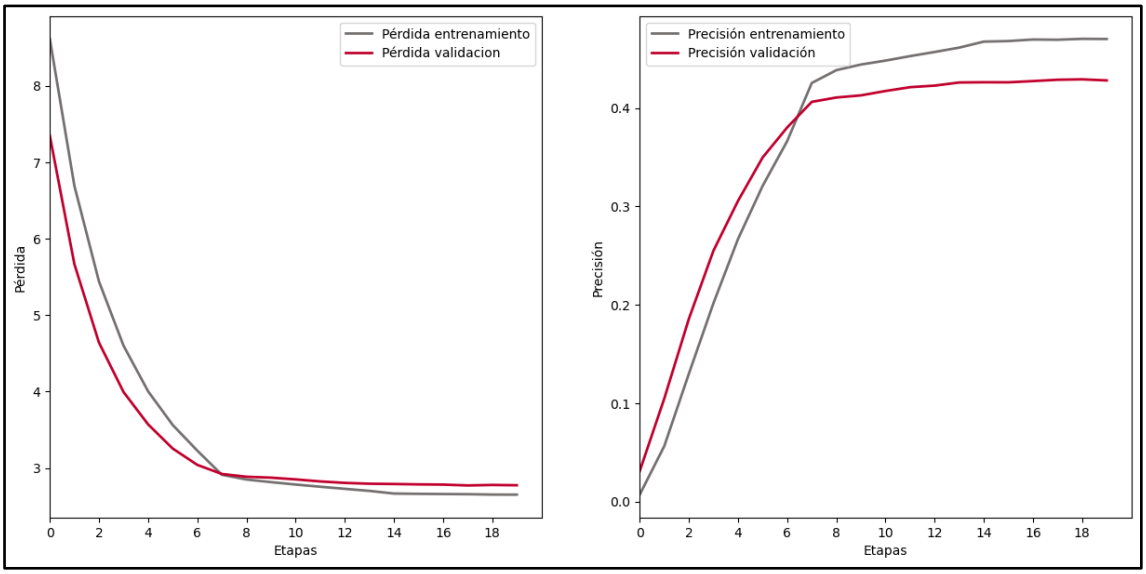

Figura 29 Pérdida y precisión EfficientNet B0

5.6.4 EfficientNet-B3

El siguiente modelo que hemos entrenado ha sido EfficientNet-B3 preentrenado.

El mejor resultado de precisión del conjunto de validación es de 53,31%, como era de esperar mejora el resultado de EfficienteNet-B0 y se aproxima al valor que hemos obtenido con el modelo ResNet-50.

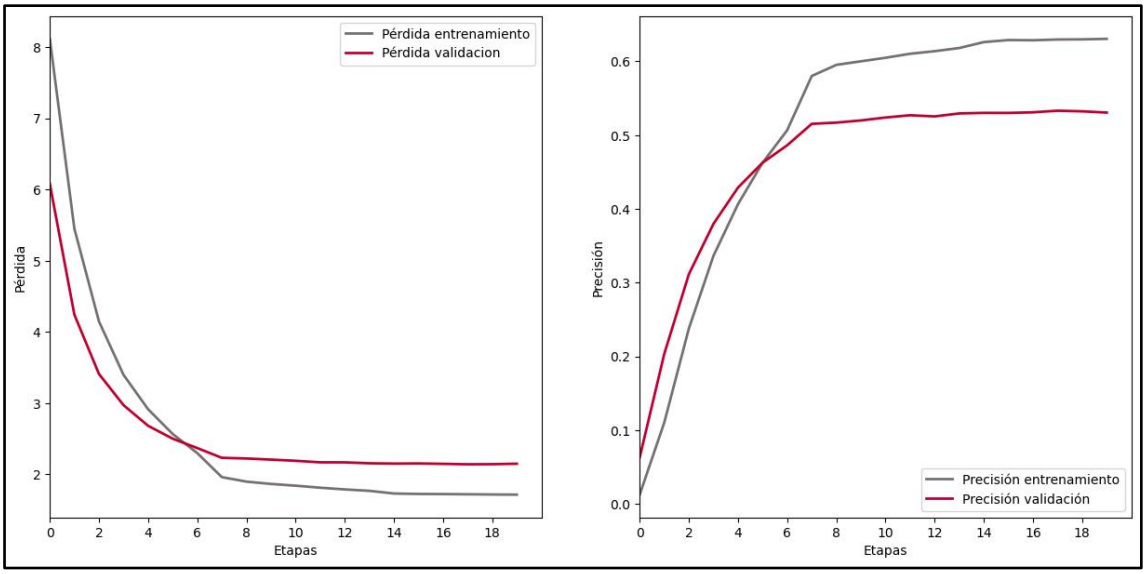

Figura 30 Pérdida y precisión EfficientNet B3

5.6.7 EfficientNet v2 Small

EfficientNetV2 va un paso más allá que EfficientNet para aumentar la velocidad de entrenamiento y la eficiencia de los parámetros. Esta red se genera utilizando una combinación de escalado (anchura, profundidad, resolución) y búsqueda de arquitectura neuronal. El objetivo principal es optimizar la velocidad de entrenamiento y la eficiencia de los parámetros. Además, esta vez el espacio de búsqueda también incluía nuevos bloques convolucionales como Fused-MBConv.

Hemos realizado un entrenamiento con el modelo más pequeño de EfficientNet v2 de los tres modelos que hay disponibles en TorchVision. Hemos cargado el modelo preentrenado con ImageNetv1 y lo hemos entrenado durante 25 etapas, durante 28 horas y 50 minutos. El porcentaje máximo alcanzado por el conjunto de validación es del 59,82% en la última etapa de entrenamiento.

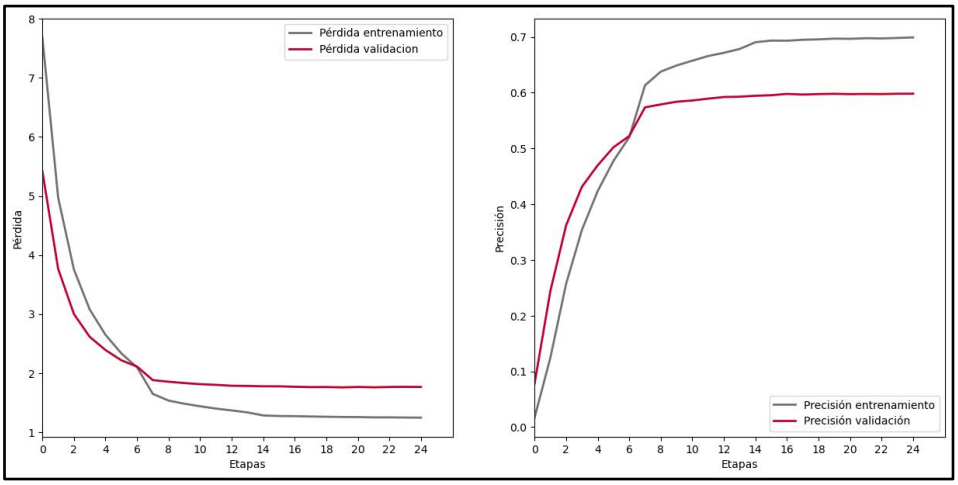

Figura 31 Pérdida y precisión EfficientNet v2 Small

5.7 Selección del modelo a implementar

Vamos a comparar los resultados obtenidos para los tres modelos que han obtenido los mejores valores precisión para el conjunto de validación.

Modelo	Menor valor de pérdida	% precisión
ResNet-50	2,0433	55,66%
EfficientNet B3	2,1402	53,31%
EfficientNet v2 Small	1,7577	59,82%

Tabla 3 Resultados tres mejores modelos

A continuación, presentamos en la misma gráfica los resultados de los tres modelos, tanto para la pérdida, como para la precisión. Se corresponden en ambos casos con los valores obtenidos para el conjunto de validación. En el gráfico de precisión se incluye también los resultados obtenidos para el conjunto de entrenamiento, son los valores representados por una línea discontinua.

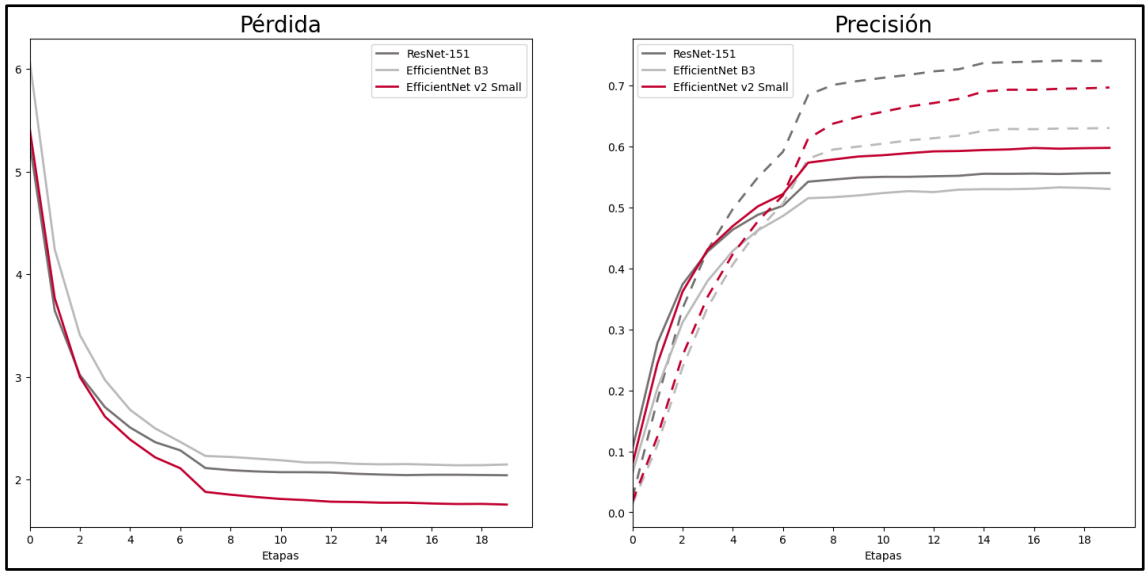

Figura 32 Pérdida y precisión. Tres mejores modelos

Si observamos en detalle la gráfica anterior y los resultados mostrados en la tabla 3, sin ningún tipo de dudas, la elección del modelo para realizar la implementación en la aplicación web es el modelo *EfficientNet v2 Small*.

5.8 Prueba de concepto. Ejemplo de uso de la aplicación web.

Desde un navegador accederemos a la URL donde se encuentra alojada nuestra aplicación, y nos encontraremos con que se nos mostrará la siguiente página:

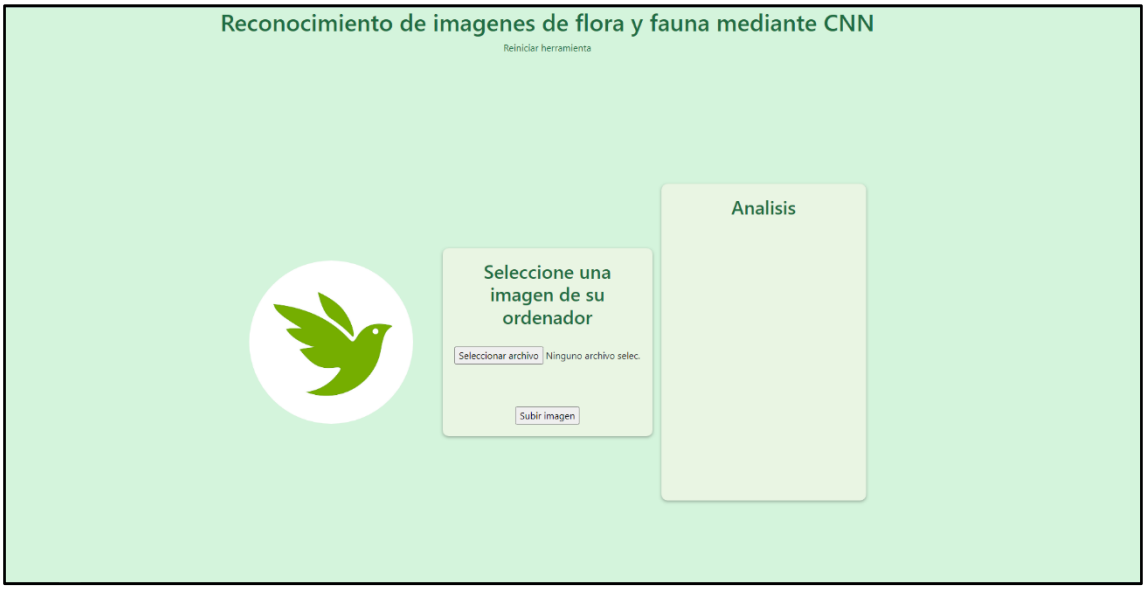

Figura 33 Pantalla principal aplicación

En la parte central de la pantalla tenemos un formulario para seleccionar la imagen. En primer lugar, pulsaremos el botón “Seleccionar archivo” y abrirá una ventana que nos permitirá explorar el contenido de nuestro ordenador y seleccionar una imagen.

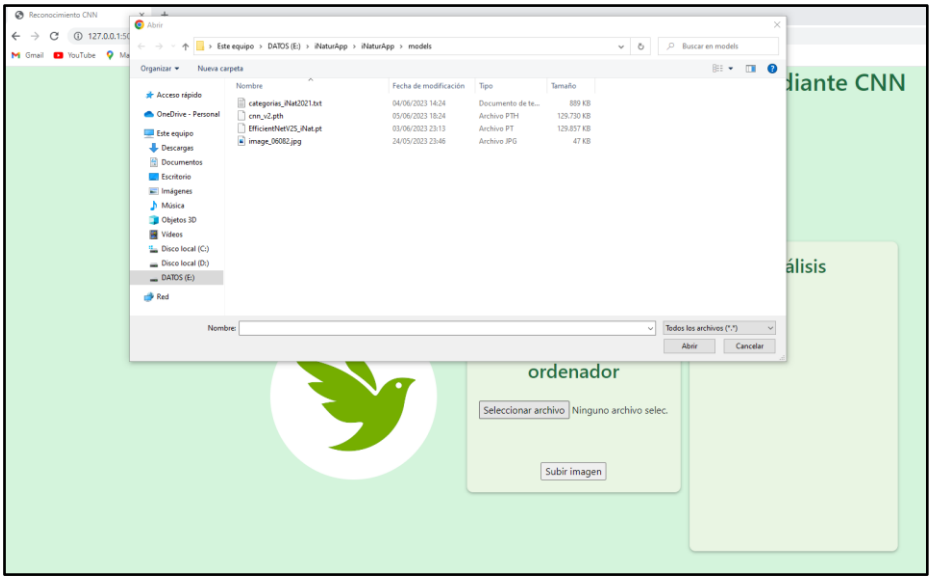

Figura 34 Aplicación. Ventana selección archivo

Una vez que hemos seleccionado el archivo, pulsaremos el botón subir imagen. Y aparecerá la imagen que hemos seleccionado en el recuadro del centro de la pantalla. En el recuadro situado en el lado derecho aparecerá en la parte superior el nivel de confianza de nuestro resultado expresado en porcentaje. A continuación, según vamos descendiendo por el cuadro se muestra el nombre común y a continuación la clasificación taxonómica desde el reino hasta la especie.

Figura 35 Aplicación. Resultado del análisis de la imagen

Si queremos realizar una nueva consulta, en la parte superior de la página, debajo del título, haremos un click sobre “Reiniciar herramienta” y nos devolverá a la situación inicial.

6. CONCLUSIONES

6.1 Selección de la herramienta

Cuando comenzamos este trabajo, después de realizar varias indagaciones, nos propusimos utilizar el framework de código abierto, desarrollado y mantenido por Google Tensorflow. La librería Keras, también de código abierto, nos parecía el complemento ideal ya que se puede ejecutar sobre TensorFlow y nos facilita el uso de modelos preentrenados. Desde un principio observamos que la depuración del código era tediosa y el tiempo empleado en subsanar errores era elevado. Lo que nos llevó a plantearnos el utilizar Pytorch como framework y la librería TorchVision como fuente de los diferentes modelos de red neuronal.

6.2 Conjunto de datos

El mejor valor que hemos obtenido para la precisión del conjunto de entrenamiento es de un 74,11% en el caso del modelo ResNet-50 y para el modelo EfficientNet v2 Small la mejor precisión es 69,91%. Ambos valores están muy por debajo del 81,9% y 87,3% que alcanzan estos modelos con el conjunto de datos ImageNet. Creemos que esta pérdida de precisión se debe a que el conjunto de datos ImageNet está formado por 1.000 categorías diferentes, frente a las 10.000 categorías de nuestro modelo.

Otro de los retos que se plantean con el conjunto de datos es la similitud visual entre categorías diferentes. A continuación, se muestra un ejemplo de dos especies de mariposas similares visualmente.

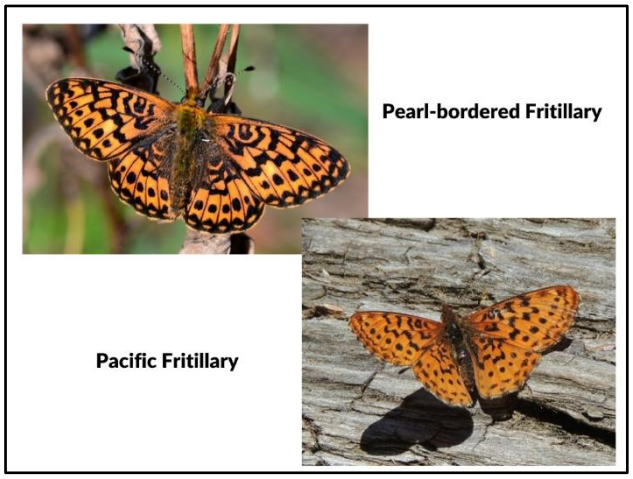

Figura 36 Especies con similitud visual

El hecho de que existan pequeñas diferencias entre una especie y otra para clasificar correctamente una imagen es un problema ya que estas características pueden no

apreciarse bien, por la calidad de la imagen, o por la disposición del individuo dentro de la imagen.

Figura 37 Imagen perteneciente al conjunto de validación

La imagen anterior se corresponde con una de las diez imágenes del conjunto de validación de una de las dos especies de mariposas anteriores, como se puede observar identificarla correctamente entraña una gran complejidad.

6.3 Técnicas de entrenamiento

Además de los modelos descritos en el aparatado *"Entrenamiento y validación del modelo"*, hemos realizado entrenamientos con los modelos DenseNet-121 y VGG19. Sobre cada uno de los modelos hemos aplicado diferentes técnicas de entrenamiento:

1. Utilizar el modelo sin inicializar los parámetros de la red neuronal. Por lo tanto, todo el conocimiento lo adquiere a través de la retroalimentación proporcionada por las diferentes etapas de entrenamiento.
2. Inicializar el modelo con los parámetros resultantes del entrenamiento del conjunto de imágenes ImageNet. En este caso, los parámetros tienen ya un valor inicial que se ve corregido durante las sucesivas etapas de entrenamiento.
3. Aplicamos la técnica de aprendizaje por transferencia.

Todos los modelos en los que aplicamos el aprendizaje por transferencia el resultado ha sido el peor resultado obtenido. En contra de nuestra idea inicial hemos descartado el aprendizaje por transferencia.

6.4 Entorno de trabajo

Los tiempos de entrenamiento de los diferentes modelos a excepción del modelo AlexNet se han ido siempre por encima de las 20 horas, lo ha provocado el invertir muchas horas en la parte de entrenamiento, alargando el tiempo previsto inicialmente para la realización del trabajo.

Los tiempos de aprendizaje por transferencia se reducían a la mitad de tiempo aproximadamente, pero como hemos comentado en el punto anterior los resultados obtenidos fueron muy pobres en relación a las otras técnicas de entrenamiento.

6.5 Puntos de mejora

Una vez que hemos descartado el aprendizaje por transferencia, deberíamos de buscar otro tipo de planteamientos que se ajusten mejor a conjuntos de datos donde el número de salidas posibles es muy elevado, en nuestro caso 10.000.

Mejorar el entorno de trabajo, aumentando la memoria RAM del equipo, solamente teníamos 16 GB, y valorar la posibilidad de utilizar dos GPUs de forma simultánea para acortar los tiempos de entrenamiento. Esto nos permitiría el realizar entrenamientos con un mayor número de etapas.

7. BIBLIOGRAFÍA

Amazon. (s.f.). *AWS.* Obtenido de ¿Qué es Python?: https://aws.amazon.com/es/what-is/python/

Area de Ciencias. (26 de Febrero de 2023). Obtenido de https://www.areaciencias.com/biologia/taxonomia-clasificacion-de-los-seres-vivos/

Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing Volume 3*.

Deshpande, A. (24 de agosto de 2016). *ADIT DESHPANDE.* Obtenido de The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3): https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

Goodfellow, I. (2016). *Deep learning Vol. 1.* Cambrige: MIT press.

Gurucharan, M. (28 de Julio de 2022). *UpGrad.* Obtenido de Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network: https://www.upgrad.com/blog/basic-cnn-architecture/

Handelman, G. S., Kok, H. K., Chandra, R. V., Razavi, A. H., Lee, M. J., & Asasi, H. (13 de Agosto de 2018). eDoctor: machine learning and the future of medicine. *Journal of Internal Medicine*, 603-619.

Kumar, A. (12 de Mayo de 2023). *Data Analytics.* Obtenido de Different Types of CNN Architectures Explained: Examples: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature 521.7553*, 436.

Martinez, J. C. (3 de Noviembre de 2021). *freeCodeCamp.* Obtenido de Deep Learning Tutorial – How to Use PyTorch and Transfer Learning to Diagnose COVID-19 Patients: https://www.freecodecamp.org/news/deep-learning-with-pytorch/

microbiologia.net. (26 de Febrero de 2023). Obtenido de https://microbiologia.net/biologia/clasificacion-de-los-seres-vivos-taxonomia/

pytorch.org. (2023). Obtenido de Tensors: https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html

Russell, S. J., & Norvig, P. (2016). *Artificial Intelligence a modern approach.* Malaysia: Pearson Education Limited.

9. ANEXO A.

En este anexo se incluye código más significativo escrito en Python y empleado en los cuadernos de jupyter.

9.1 Función de entrenamiento de la red neuronal

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
  since = time.time()

  best_model_wts = copy.deepcopy(model.state_dict())
  best_acc = 0.0

  for epoch in range(num_epochs):
    print(f'Entrenando etapa {epoch + 1} de {num_epochs} ...')
    epoch_init = time.time()

    # Cada iteración tiene una fase de validación y una fase de entrenamiento
    for phase in ['train', 'val']:
      if phase == 'train':
        model.train()  # Establece el modelo en modo entrenamiento
      else:
        model.eval()   # Establece el modelo en modo validación

      running_loss = 0.0
      running_corrects = 0

      # Itera sobre los datos
      for inputs, labels in dataloaders[phase]:
        inputs = inputs.to(device)
        labels = labels.to(device)

        # pone a cero los gradientes de los tensores optimizados
        optimizer.zero_grad()

        # track historial adelante solo en la fase de entrenamiento
        with torch.set_grad_enabled(phase == 'train'):
          outputs = model(inputs)
          _, preds = torch.max(outputs, 1)
          loss = criterion(outputs, labels)

          # Solo si está en la fase de entrenamiento retroceder + optimizar
          if phase == 'train':
            loss.backward()
            optimizer.step()

        # estatisticas
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
      if phase == 'train':
        scheduler.step()

      epoch_loss = running_loss / dataset_sizes[phase]
      epoch_acc = running_corrects.double() / dataset_sizes[phase]

      if phase == 'train':
        results["train_loss"].append(epoch_loss)
        results["train_acc"].append(epoch_acc.item())
        print(f'Entrenamiento\tPerdida: {epoch_loss:.4f}\tPrecisión: {epoch_acc:.4f}')
      else:
        tittle = 'Validación'
```

```
            results["val_loss"].append(epoch_loss)
            results["val_acc"].append(epoch_acc.item())
            print(f'Validación\tPerdida: {epoch_loss:.4f}\tPrecisión: {epoch_acc:.4f}'
                f'\t{(time.time() - epoch_init)//60:.0f}min {(time.time() - epoch_init)%60:.0f}seg')

        # copia del modelo con mejor porcentaje de acierto en la validación
        if phase == 'val' and epoch_acc > best_acc:
            print('Guardando el modelo ...')
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print()

time_elapsed = time.time() - since
if time_elapsed < 3600:
    print(f'Entrenamiento completado en: {time_elapsed // 60:.0f}min. {time_elapsed % 60:.0f}seg.')
else:
    rest_elapsed = time_elapsed % 3600
    print(f'Entrenamiento completado en: {time_elapsed // 3600:.0f}horas {rest_elapsed // 60:.0f}min. {rest_elapsed
% 60:.0f}seg.')
print(f'Mejor precisión validación: {best_acc:.4f}')

# guarda los mejores pesos del modelo
model.load_state_dict(best_model_wts)
return model
```

9.2 Función que visualiza la gráfica de pérdida y precisión

```
def plot_curves(results):

  # Recuperamos los valores de pérdida de entrenamiento y validación
  train_loss = results['train_loss']
  valid_loss = results['val_loss']

  # Recuperamos los valores de precisión de entrenamiento y validación
  train_accuracy = results['train_acc']
  valid_accuracy = results['val_acc']

  # Número de iteraciones que tenemos
  epochs = range(len(results['train_loss']))

  # Definimos el gráfico
  plt.figure(figsize=(15, 7))

  # Pérdida
  plt.subplot(1, 2, 1)
  plt.plot(epochs, train_loss, label='Pérdida entrenamiento', color='#767171', linewidth=2)
  plt.plot(epochs, valid_loss, label='Pérdida validacion', color='#C2002F', linewidth=2)
  plt.ylabel('Pérdida')
  plt.xlabel('Etapas')
  plt.xlim(0,20)
  plt.xticks(np.arange(0, 20, 2))
  plt.legend()

  # Precisión
  plt.subplot(1, 2, 2)
  plt.plot(epochs, train_accuracy, label='Precisión entrenamiento', color='#767171', linewidth=2)
  plt.plot(epochs, valid_accuracy, label='Precisión validación', color='#C2002F', linewidth=2)
  plt.ylabel('Precisión')
  plt.xlabel('Etapas')
  plt.xlim(0,20)
  plt.xticks(np.arange(0, 20, 2))
  plt.legend()
```