

# Reconocimiento visual de aves con Deep Learning



Grado en Ingeniería Informática

## Trabajo Fin de Grado

Autor: Juan García Martínez

Tutor/es:

José García Rodríguez  
Esther Sebastián González

02/2024

## Resumen

La inteligencia artificial está revolucionando rápidamente diversos aspectos de nuestras vidas, mejorando la calidad de las tecnologías, y como resultado, permitiendo crear tecnologías capaces de ayudar a las personas en su vida diaria de distintas maneras.

El reconocimiento visual de imágenes es una rama clave de la inteligencia artificial que persigue el objetivo principal de capacitar a las máquinas para interpretar y reconocer objetos de manera análoga a cómo el cerebro humano procesa la información visual captada por el ojo.

En este contexto, el reconocimiento visual de aves mediante imágenes se ha convertido en un campo de interés en el mundo de la investigación y conservación de la biodiversidad, permitiendo identificar y clasificar grandes cantidades de especies de aves a partir de imágenes.

En este proyecto, se ha desarrollado un clasificador de aves mediante el uso de tecnologías de aprendizaje automático y redes neuronales profundas, herramientas para recopilación de imágenes para la creación de un dataset propio y la evaluación de nuestro clasificador. Esta tecnología no solo ayuda con la labor de identificar aves para los investigadores, científicos y aficionados, sino también juega un papel muy importante en la conservación de la biodiversidad. Esto hace que tengamos la capacidad de monitorizar de forma precisa la población de aves para evaluar los riesgos de amenaza y conservación de las especies, sus hábitats y sus comportamientos.

## Palabras clave

Inteligencia artificial, Machine Learning, Deep Learning, Transfer Learning, Redes neuronales Convolucionales, dataset, aves, arquitectura, época, precisión y pérdida.

# Índice general

1. Introducción.....	10
1.1 Descripción general .....	10
1.2 Motivación.....	10
1.3 Estructura del TFG.....	10
2. Objetivos.....	12
3. Marco teórico.....	13
3.1 Machine Learning .....	13
3.2 Deep Learning.....	14
3.3 Redes Neuronales Artificiales.....	15
3.3.1 Redes Neuronales Convolucionales .....	15
3.3.2 Redes Neuronales Recurrentes .....	19
3.3.3 Redes Generativas Adversarias.....	22
3.4 Transfer Learning.....	23
3.5 Arquitecturas .....	24
3.5.1 VGGNet.....	24
3.5.2 ResNet152V2.....	25
3.5.3 EfficientNetV2.....	26
3.5.4 InceptionV3.....	27
3.6 Aplicaciones .....	27
3.6.1 Cornell Lab of Ornithology.....	27
3.6.2 iNaturalist.....	30
4. Metodología.....	32
4.1 Hardware .....	32
4.2 Software.....	32
4.2.1 Python.....	32
4.2.2 Google Colab con Python.....	33
4.2.3 Anaconda Distribution con Python.....	33
4.2.4 CUDA ToolKit y cuDNN.....	34
4.2.5 Librerías.....	35
4.3 Dataset .....	39
5. Desarrollo.....	40
5.1 Obtención de datos para el dataset.....	40
5.1.1 Dataset propio .....	40
5.1.2 Recopilación de datos.....	41
5.1.3 Filtrado de datos.....	43
5.2 Análisis de datos .....	46
5.2.1 Versiones del dataset .....	46
5.2.3 División del conjunto.....	48
5.3 Modelos .....	49
5.3.1 Aumento de datos .....	49

5.3.2 Arquitecturas .....	51
5.3.3 Técnicas empleadas .....	51
6. Experimentación.....	54
6.1 Pruebas con el dataset NABirds .....	54
6.2 Pruebas con el dataset propio .....	57
6.2.1 Prueba añadiendo capas densas y dropout .....	58
6.2.2 Prueba añadiendo capas densas y dropout con regularización de pesos .....	60
6.2.3 Fine-tuning con capas densas y dropout.....	63
6.2.4 Fine-tuning con Capas densas y dropout con regularización de pesos.....	65
6.2.5 Prueba con EfficientNetV2B3 preentrenado.....	68
6.3 Selección del mejor modelo y pruebas con imágenes .....	71
7. Conclusión .....	74
Referencias.....	75
Lista de Acrónimos y Abreviaturas.....	79

# Índice de figuras

Figura 3.1: Inteligencia Artificial, Machine Learning y Deep Learning .....	13
Figura 3.2: Esquema del proceso de Machine Learning.....	14
Figura 3.3: Esquema del proceso de Deep Learning.....	15
Figura 3.4: Esquema de una arquitectura CNN.....	16
Figura 3.5: Operación de convolución.....	16
Figura 3.6: Función sigmoide.....	17
Figura 3.7: Función de activación softmax.....	17
Figura 3.8: Función de activación ReLU.....	18
Figura 3.9: Función de activación Tanh. ....	18
Figura 3.10: Operación pooling para una matriz 4x4 con filtro de pooling con tamaño 2x2.....	19
Figura 3.11: Arquitectura RNN.....	20
Figura 3.12: Leyenda para arquitecturas de la figura 3.13 y figura 3.14.....	20
Figura 3.13: Arquitectura LSTM.....	20
Figura 3.14: Arquitectura GRU.....	21
Figura 3.15: Arquitectura de una GAN.....	22
Figura 3.16: Esquema TL en DL.....	23
Figura 3.17: Arquitectura VGGNet internamente.....	24
Figura 3.18: Arquitecturas VGG16 y VGG19.....	25
Figura 3.19: Arquitectura ResNet152V2.....	25
Figura 3.20: Arquitectura de EfficientNetV2B3.....	26
Figura 3.21: Arquitectura InceptionV3.....	27
Figura 3.22: Inicio de la página web de eBird. ( <a href="https://ebird.org/explore">https://ebird.org/explore</a> ) .....	28
Figura 3.23: Logo de Merlin Bird ID.....	29
Figura 3.24: Proceso para identificar un ave mediante la funcionalidad Paso a paso. ....	29
Figura 3.25: Resultado de Paso a paso.....	29
Figura 3.26: Proceso para identificar un ave mediante audio.....	30
Figura 3.27: Proceso de reconocimiento de aves mediante fotos.....	30
Figura 3.28: Funcionalidad de creación de un proyecto en la web de iNaturalist. ....	31
Figura 4.1: Python para IA. ....	33
Figura 4.2: Estructura de los arrays con distintas dimensiones.....	37
Figura 4.3: Ejemplo de la salida de una matriz de confusión.....	38
Figura 5.1: Nombre de las especies de las aves.....	40
Figura 5.2: Apartado de consulta de observaciones iNaturalist. ....	42
Figura 5.3: Apartado selección de columnas para exportar a CSV.....	42
Figura 5.4: Ejemplo de consulta con filtros de eBird.....	43
Figura 5.5: Ejemplo de mala clasificación de un Serín Verdecillo extraída de eBird. ....	43
Figura 5.6: Ejemplo de multitud de flamencos común en los arrozales de l'Albufera.....	44

Figura 5.7: Ejemplo de imagen borrosa de un Jilguero europeo.....	44
Figura 5.8: Diferencia entre macho y hembra de Pato cucharón norteño.....	45
Figura 5.9: Pato cucharón norteño secándose y regulando su temperatura.....	45
Figura 5.10: Diagrama de barras de la cantidad de imágenes por clase.....	46
Figura 5.11: Similitud entre Gaviota picofina, gaviota patiamarilla y gaviota reidora. ....	47
Figura 5.12: Similitud entre Vencejo común (izquierda) y Avión roquero (derecha). ....	48
Figura 5.13: Diagrama de barras de la cantidad de imágenes por clase del dataset final.....	48
Figura 5.14: Ejemplo de aplicación de la técnica shear range.....	50
Figura 6.1: Entrenamiento conjunto NABirds con arquitectura VGG16. ....	55
Figura 6.2: Entrenamiento conjunto NABirds con arquitectura VGG19. ....	55
Figura 6.3: Entrenamiento conjunto NABirds con arquitectura ResNet152V2. ....	55
Figura 6.4: Entrenamiento conjunto NABirds con arquitectura InceptionV3.....	55
Figura 6.5: Entrenamiento conjunto NABirds con arquitectura EfficientNetV2B3.....	55
Figura 6.6: Matriz de confusión conjunto de test sobre el modelo InceptionV3. ....	57
Figura 6.7: Entrenamiento de ResNet152V2 mediante TL. ....	58
Figura 6.8: Entrenamiento de InceptionV3 mediante TL. ....	58
Figura 6.9: Entrenamiento de EfficientNetV2B3 mediante TL. ....	59
Figura 6.10: Mejor época de entrenamiento en ResNet152V2 con TL antes de sobreentrenar .....	59
Figura 6.11: Entrenamiento de ResNet152V2 mediante TL con regulador de pesos.....	61
Figura 6.12: Entrenamiento de InceptionV3 mediante TL con regulador. ....	61
Figura 6.13: Entrenamiento de EfficientNetV2B3 mediante TL con regulador. ....	62
Figura 6.14: Entrenamiento de ResNet152V2 mediante TL con fine-tuning.....	64
Figura 6.15: Entrenamiento de InceptionV3 mediante TL con fine-tuning. ....	64
Figura 6.16: Entrenamiento de EfficientNetV2B3 mediante TL con fine-tuning.....	64
Figura 6.17: Entrenamiento de ResNet152V2 mediante fine-tuning con regulador de pesos. ....	66
Figura 6.18: Entrenamiento de InceptionV3 mediante fine-tuning con regulador de pesos. ....	66
Figura 6.19: Entrenamiento de EfficientNetV2B3 mediante fine-tuning con regulador de pesos.....	66
Figura 6.20: Entrenamiento de EfficientNetV2B3-21k mediante TL con capa densa. ....	68
Figura 6.21: Entrenamiento EfficientNetV2B3-21k mediante TL con capas densas y dropout.....	69
Figura 6.22: Matriz de confusión empleando EfficientNetV2B3 mediante TL con capas densas y dropout .....	70
Figura 6.23: Prueba del mejor modelo (EfficientNetV2B3) con imágenes distintas al dataset.....	71
Figura 6.24: Predicción correcta de Avoceta común. ....	72
Figura 6.25: Predicción errónea de Jilguero europeo en un peñasco.....	72
Figura 6.26: Predicción correcta de Jilguero europeo en un peñasco. ....	72
Figura 6.27: Predicción de Gaviota reidora en imagen con una Gaviota reidora y una Paloma Torcaz. ....	73
Figura 6.28: Porcentajes de confianza de cada una de las clases respecto a la figura anterior. ....	73

# Índice de tablas

Tabla 3.1: Información de las arquitecturas.	24
Tabla 4.1: Especificaciones del ordenador sobremesa empleado.	32
Tabla 4.2: GPUs CPUs y TPUs que ofrece Google Colab Pro.	32
Tabla 5.1: Cantidad de parámetros de las diferentes arquitecturas.	51
Tabla 6.1: Salida del entrenamiento del conjunto de NABirds con las diferentes arquitecturas.	54
Tabla 6.2: Entrenamiento hasta checkpoint del conjunto NABirds con las diferentes arquitecturas.	56
Tabla 6.3: Resultados conjunto de test NABirds con las diferentes arquitecturas.	56
Tabla 6.4: Resultados del entrenamiento mediante TL.	58
Tabla 6.5: Resultados con los mejores pesos de entrenamiento mediante TL.	59
Tabla 6.6: Resultados de ResNet152V2 con TL antes de sobreentrenar.	60
Tabla 6.7: Resultados conjunto de test con el dataset propio mediante TL.	60
Tabla 6.8: Resultados del entrenamiento con el dataset propio mediante TL con regulador.	60
Tabla 6.9: Resultados con los mejores pesos de entrenamiento mediante TL con regulador.	62
Tabla 6.10: Resultado de conjunto de test mediante TL con regulador.	62
Tabla 6.11: Cantidad total de parámetros a entrenar mediante TL con fine-tuning.	63
Tabla 6.12: Resultados del entrenamiento con dataset propio mediante TL con fine-tuning.	63
Tabla 6.13: Resultados con los mejores pesos de entrenamiento mediante TL con fine-tuning.	65
Tabla 6.14: Resultado de conjunto de test mediante TL con fine-tuning.	65
Tabla 6.15: Resultados del entrenamiento con dataset propio mediante fine-tuning y regulador de pesos.	65
Tabla 6.16: Resultados con los mejores pesos de entrenamiento mediante fine-tuning con regulador de pesos.	67
Tabla 6.17: Resultado de conjunto de test mediante fine-tuning.	67
Tabla 6.18: Resultados del entrenamiento de EfficientNetV2B3 mediante TL con capa densa.	68
Tabla 6.19: Resultados del entrenamiento de EfficientNetV2B3 mediante TL con capas densas y dropout.	69
Tabla 6.20: Resultado con mejores pesos de entrenamiento de EfficientNetV2B3 mediante TL con capas densas y dropout.	69



# 1. Introducción

En este primer capítulo se introduce el tema principal de la investigación. La estructura se organiza de la siguiente manera: en el subapartado 1.1 se presenta un breve resumen del desarrollo abordado en este trabajo, en la Sección 1.2 se explica la motivación que ha impulsado su realización. Finalmente, en la Sección 1.3 se detalla la estructura que seguirá este trabajo.

## 1.1 Descripción general

En este proyecto se ha investigado el uso del Deep Learning para el desarrollo de un clasificador de aves mediante el uso de técnicas avanzadas, como Redes Neuronales Convolucionales. El objetivo principal es crear un sistema capaz de identificar automáticamente diferentes especies de aves a partir de imágenes. Este enfoque tiene como motivación mejorar la precisión en la identificación de aves en diversos contextos, como la conservación de especies, estudios científicos y monitoreo ambiental.

## 1.2 Motivación

La elección de esta propuesta es mayoritariamente una motivación personal, ya que me fascinan las aves, conocer su hábitat, su comportamiento y la conservación de las especies.

Considero que los parques naturales representan un escenario fundamental en la gestión para la sostenibilidad de la biodiversidad a largo plazo. Son espacios donde se concentra la protección de ecosistemas frágiles, la investigación científica y la educación ambiental, desempeñando un papel fundamental en la preservación de la diversidad biológica y en el desarrollo de prácticas sostenibles.

Las aves son fundamentales en los ecosistemas, dispersando semillas, controlando plagas y polinizando plantas. Según BirdLife International, son clave para la salud ambiental, indicando efectos del cambio climático y la pérdida de hábitat. Conservarlas es crucial para mantener la estabilidad y salud de los ecosistemas frente a desafíos ambientales.

La decisión para contemplar este tema en el Trabajo Fin de Grado no se limita a satisfacer mis intereses personales, sino que también para ampliar mis conocimientos sobre un área que me interesa y en la que todavía tengo mucho que aprender.

## 1.3 Estructura del TFG

A continuación, se detallará la estructura del TFG y las tareas que se deben de realizar para su desarrollo:

- **Objetivos:** Se definirán los objetivos que se pretende alcanzar mediante el desarrollo del TFG.
- **Marco teórico:** Se explicarán las bases y el contexto de la rama a la que pertenece el TFG, como la Inteligencia Artificial, Machine Learning, Deep Learning, Redes Neuronales Artificiales y las aplicaciones.
- **Metodología:** Se mostrarán las herramientas empleadas en este TFG.
- **Desarrollo:** Se detallará el proceso paso a paso en la realización y modificación del clasificador de aves.

- **Resultados:** Se analizarán los resultados con las diferentes pruebas sobre sus arquitecturas y parámetros.
- **Conclusión:** Se realizará un análisis de los resultados obtenidos para extraer conclusiones y determinar si se han alcanzado los objetivos propuestos.

## 2. Objetivos

El objetivo principal de este TFG es realizar un sistema capaz de reconocer y clasificar visualmente aves utilizando técnicas de Deep Learning. Este sistema podrá identificar automáticamente las aves mediante imágenes y para ello se empleará un conjunto de imágenes.

El primer objetivo necesario para cumplir el propósito del trabajo consiste en la comparación de los diferentes modelos de Redes Neuronales Convolucionales escogidos para el reconocimiento visual de aves. Este análisis permitirá identificar cuál es el modelo más adecuado en términos de precisión y eficiencia para el contexto específico de implementación en dispositivos como Raspberry Pi o Jetson Nano.

El segundo objetivo se centra en evaluar la idoneidad de los modelos seleccionados para ser implementados en plataforma, como la Raspberry Pi y el Jetson Nano. Esta evaluación incluirá la medición de recursos computacionales requeridos (como CPU y memoria) y la optimización de los modelos para garantizar un desempeño eficiente.

Por último, el objetivo necesario para cumplir con el propósito del TFG implica crear y filtrar un dataset de imágenes de aves, correctamente etiquetado y organizado. Este dataset será fundamental para el entrenamiento y validación de los modelos de Redes Neuronales Convolucionales seleccionados, asegurando así la precisión y generalización del modelo.

### 3. Marco teórico

En la actualidad, gracias al aumento de potencia del cómputo y el almacenamiento, se han desarrollado nuevas técnicas y algoritmos que tienen la capacidad de procesar información derivada de datos y tomar decisiones para lograr un objetivo dado, dicho de otra manera, que puedan pensar y actuar como humanos e incluso superarlo en ciertos ámbitos.

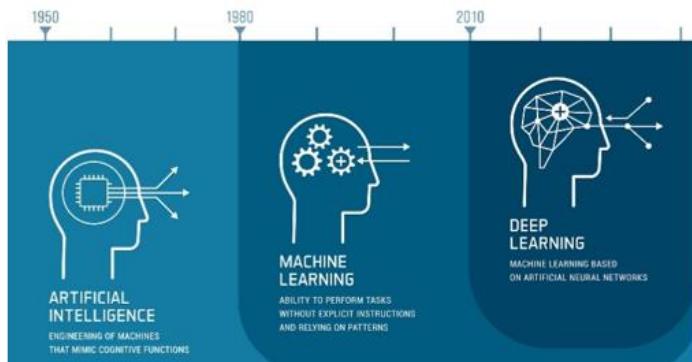


Figura 3.1: Inteligencia Artificial, Machine Learning y Deep Learning.

En este capítulo se introducirán las bases teóricas para el posterior entendimiento del proyecto. En el subapartado 3.1 se introducirá el concepto de Machine Learning (aprendizaje automático), en el subapartado 3.2 se introduce el Deep Learning (aprendizaje profundo), destacando su capacidad para modelar representaciones complejas a partir de datos no estructurados, en el subapartado 3.3 se examinan los diferentes tipos de Redes Neuronales Artificiales utilizadas en este contexto, enfatizando sus estructuras y funciones específicas que las hacen adecuadas para la clasificación de especies aviares basadas en imágenes.

Se aborda también en el subapartado 3.4 el Transfer Learning (aprendizaje por transferencia), una técnica esencial que permite aprovechar conocimientos previamente adquiridos en grandes conjuntos de datos para mejorar el rendimiento de modelos en tareas específicas de reconocimiento de aves. Además, en el subapartado 3.5 se analizan diversas Arquitecturas de redes neuronales convolucionales, las cuales son especialmente diseñadas para extraer características relevantes de las imágenes de aves y lograr una clasificación precisa. Finalmente, en el subapartado 3.6 se exploran Aplicaciones prácticas de estas tecnologías en la conservación de especies, investigación biológica y monitoreo ambiental

#### 3.1 Machine Learning

Machine Learning (ML) (IEEE, 2018), también conocido como aprendizaje automático, es una subdisciplina de la IA que permite aprender de los datos utilizando algoritmos para identificar patrones, realizar predicciones y mejorar el rendimiento sin estar programadas para ello.

El proceso de ML se constituye principalmente por 3 componentes: el modelo, la entrada y la salida de datos. El modelo es un algoritmo que aprende patrones a partir de los datos, la entrada de datos es el conjunto de datos que se emplea para entrenar el modelo y los datos de salida son los resultados generados por la predicción del modelo tras el procesamiento de los datos de entrada.

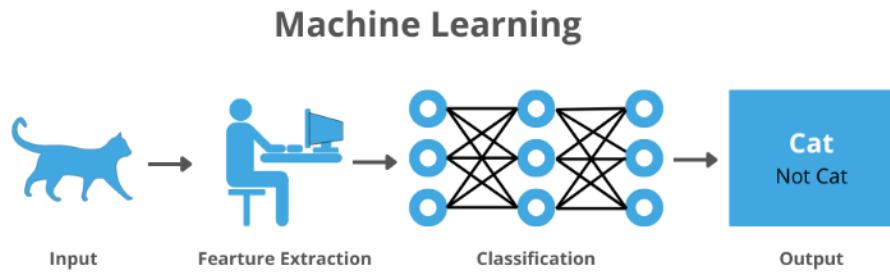


Figura 3.2: Esquema del proceso de Machine Learning.

Existen diversos tipos de ML, dependiendo de la forma en la que aprenden: El aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

El aprendizaje supervisado entrena con una base de datos con información correctamente etiquetada que se utiliza posteriormente para predecir nuevos datos sin etiquetar. El aprendizaje no supervisado entrena con datos sin etiquetar detectando patrones o estructuras ocultas en los datos. Por último, el aprendizaje por refuerzo tiene la finalidad de construir modelos que aumenten el rendimiento y realicen predicciones tomando como base los resultados, ya sean recompensas o castigos, de cada iteración realizada en el modelo.

ML se emplea en diversos ámbitos, desde la detección de spam, completar texto y recomendación de contenido hasta servicios en línea, predicción del clima y diagnósticos médicos.

Existen gran cantidad y diversidad de algoritmos de ML, algunos de ellos incluyen Redes Neuronales, los Árboles de decisión (supervisado), K means (no supervisado) y Deep Q-Networks (por refuerzo).

### 3.2 Deep Learning

Deep Learning (DL) (Goodfellow, Bengio, & Courville, 2016) es una subrama del ML que se enfoca en el uso de Redes Neuronales Artificiales para modelar y aprender patrones complejos en gran cantidad de datos de entrada, que a diferencia de ML emplea capas de neuronas artificiales extrayendo automáticamente características de alto nivel directamente de los datos.

A las capas de entrada y salida se les denominan capas visibles, en la capa de entrada es donde se introducen los datos originales para el procesado y la capa de salida es donde se realiza la predicción final. Entre estas capas se encuentran las neuronas que se componen las capas ocultas, de ahí el término “Deep” (profundo), donde capa tras capa se aprenden características jerárquicamente, donde las capas iniciales aprenden características simples y las más profundas características más abstractas y complejas dando lugar a representaciones complejas que mejoran la precisión de las predicciones en la capa de salida.

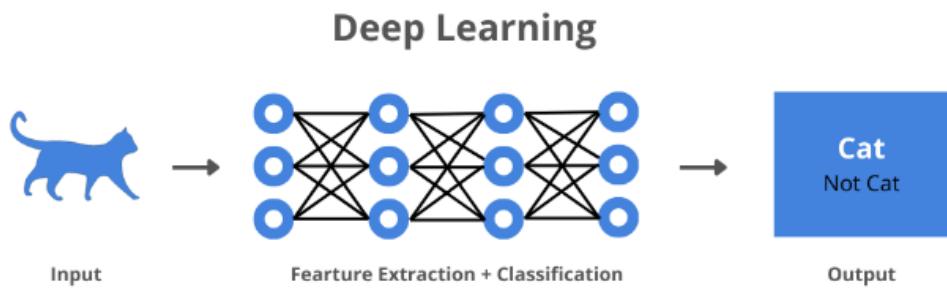


Figura 3.3: Esquema del proceso de Deep Learning.

Respecto al uso del DL, es muy útil en áreas como Video Captioning, Procesamiento del Lenguaje Natural (NLP), ciencias de la salud y reconocimiento de voz y audio, gracias a su capacidad para manejar y extraer información de grandes volúmenes de datos. Con el avance y mejoras sustanciales en los algoritmos, el DL se emplea en diversas áreas desde el marketing personalizado hasta la conducción autónoma y el diagnóstico médico.

En resumen, actualmente el DL es una herramienta de vital importancia gracias a la capacidad de trabajar de manera autónoma. Sin embargo, presenta ciertas limitaciones como los altos requerimientos computacionales, necesidad de grandes conjuntos de datos y el riesgo de sobreajuste, más conocido como overfitting.

### 3.3 Redes Neuronales Artificiales

Las Redes Neuronales Artificiales ([ANN](#)) (UNIR, 2021) son modelos computacionales que permiten dotar a los sistemas de la capacidad de aprender, interpretar y predecir de una manera muy similar a como lo hace el cerebro humano.

Las ANN están formadas por neuronas artificiales conectadas entre sí, las cuales reciben información de los datos de entrada o de otras neuronas, de manera muy similar a las neuronas del cerebro, que reciben impulsos nerviosos. Estas ANN generan un valor de salida que se transmite a otras neuronas o a la salida final de la red. En los siguientes apartados se introducirán los tipos más comunes de ANN, cada uno de ellos diseñados para resolver distintos tipos de desafíos.

#### 3.3.1 Redes Neuronales Convolucionales

En este subapartado se profundizará sobre una familia comúnmente empleada en el procesamiento de imágenes y reconocimiento visual.

Las Redes Neuronales Convolucionales ([CNN](#)) (DataScientest, 2021) son una subcategoría de las ANN implementadas para el procesamiento y clasificación de imágenes, es decir, para aplicaciones de VC. La estructura y procesamiento están inspiradas en el sistema visual de los animales.

La arquitectura de una CNN enfocadas a la clasificación de imágenes es prácticamente igual, aplicando filtrado de imágenes, extracciones de características de cada imagen y reducción de parámetros.

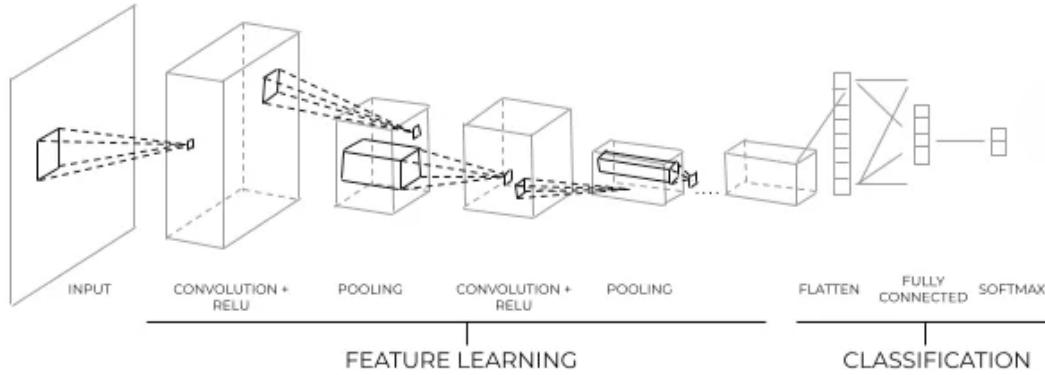


Figura 3.4: Esquema de una arquitectura CNN.

En la *figura 3.4* se observan las diferentes capas fundamentales. Cada capa realiza un propósito concreto:

- La capa convolucional es la capa encargada de analizar los datos de entrada detectando patrones, texturas, bordes y otras características mediante un filtro *kernel* elegido y que recorrerá todos los píxeles de la imagen y se obtendrá una nueva matriz de salida.

Se realiza entre una matriz bidimensional de entrada y otra llamada filtro, máscara o *kernel* que se desliza por la matriz bidimensional realizando una operación matemática en cada sección de esta y almacenando el resultado en otra matriz. Esta operación realizada, consiste en una suma de los resultados de la multiplicación de los valores de la matriz filtro por los valores correspondientes de la matriz bidimensional. Esta operación se realizará iterativamente hasta desplazar el filtro por toda la matriz de entrada y completar la matriz de salida.

La *figura 3.5* ilustra la operación de convolución de una matriz de entrada  $7 \times 7$  con un filtro de tamaño  $3 \times 3$ .

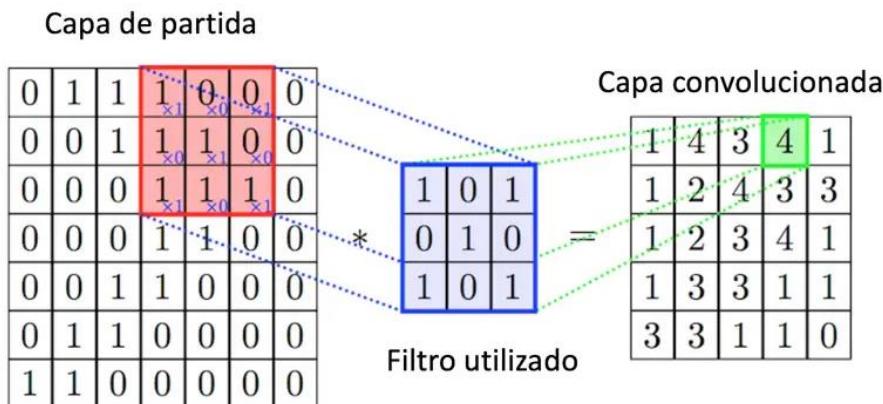


Figura 3.5: Operación de convolución.

- La capa de activación y sesgo implementa una función de activación transformando cada valor en la salida de la convolución. Si el valor de salida es negativo, lo cambia a cero, si es positivo lo deja igual.

Esta capa juega un papel importante ya que sin ellas no se podría modelar las complejidades de los datos ya que la red sería una combinación lineal de entradas.

Las funciones más comunes de activación son las siguientes:

### Sigmoide

La función sigmoide es una función matemática con forma de S donde su salida está definida en el rango de 0 y 1, siendo muy útil en problemas de clasificación binaria y frecuentemente empleada en la última capa de la red neuronal.

$$f(x) = \frac{1}{1 + e^{-x}}$$

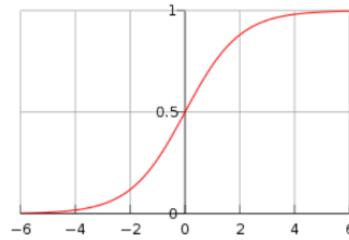


Figura 3.6: Función sigmoide.

### Softmax

La función softmax, es conocida como función exponencial normalizada siendo muy empleada en la clasificación multiclase. Esta función opera sobre un vector convirtiéndolo en una distribución de probabilidades cuya suma de todo el vector sería uno. Este conjunto de probabilidades se podría definir como la cantidad de confianza para cada clase, proporcionando información sobre la certeza del modelo respecto a sus previsiones.

### Softmax Function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

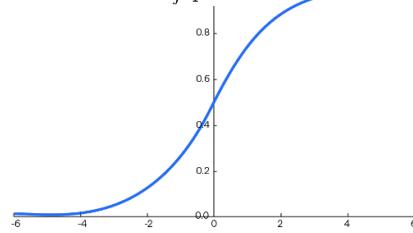


Figura 3.7: Función de activación softmax.

Esta función se usa principalmente en la última capa de las redes neuronales para tareas de clasificación con múltiples clases.

## ReLU

ReLU es una de las funciones de activación más empleadas dentro de las Redes Neuronales Profundas (DNN). Para las entradas mayores que cero, actúa como una función lineal con un gradiente de 1, es decir, no modifica las entradas positivas y evita a mitigar el desvanecimiento de gradiente (pérdida de parámetros en la red neuronal).

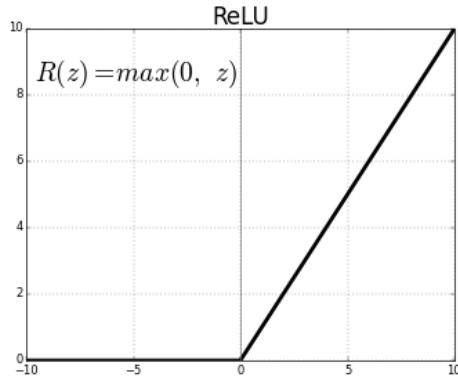


Figura 3.8: Función de activación ReLU.

Como ReLU produce cero para todas las entradas negativas, realiza activaciones de neuronas de manera dispersa y en cualquier momento, activando solo un subconjunto de neuronas, lo que hace que computacionalmente sea más eficiente.

Esta función es utilizada principalmente en las capas ocultas de la ANN.

## Tanh

Esta función es similar a sigmoide, pero centrada en cero produciendo valores en el rango de -1 hasta 1, pudiendo tratar los valores negativos con mejor eficacia que sigmoide. Esta permite aprendizaje y convergencias más rápidas durante el entrenamiento siendo más resistentes al desvanecimiento de gradiente que la función sigmoide.

A pesar de estas ventajas, la función Tanh también sufre el problema de desvanecimiento de gradiente. Esto puede ralentizar drásticamente el proceso de entrenamiento y provocar malas propiedades de convergencia.

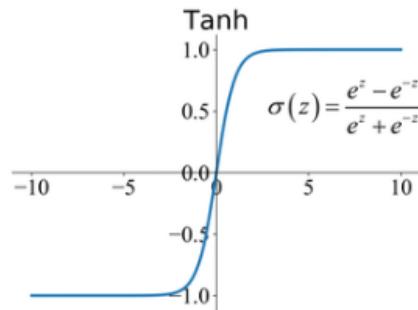


Figura 3.9: Función de activación Tanh.

Tanh es empleada en las capas ocultas de las redes neuronales.

- La capa pooling reduce las dimensiones de la matriz de las características detectadas en las capas convolucionales, conservando así las características más esenciales. Consiste en dividir la matriz de entrada en regiones no solapadas, desplazando el filtro de tamaño  $p_1 \times p_2$ , siendo  $p_1$  el número de filas de la matriz y  $p_2$  el número de columnas de la matriz, sobre la matriz de entrada y aplicar una función de agregación sobre cada una de las regiones. Esto genera una red más robusta y disminuye la cantidad de parámetros y el coste computacional. Existen varios tipos de funciones de agregación en pooling:

**Avg-Pooling:** Calcula la media de los valores en cada región del mapa de características.

**Min-Pooling:** Calcula el valor mínimo en cada región del mapa de características.

**Max-pooling:** Calcula el valor máximo en cada región del mapa de características, siendo una de las más empleadas debido a su efectividad para extraer las características más destacadas. En la siguiente *figura 3.10* se muestra el resultado de aplicar *maximum pooling* y *average pooling* sobre la misma matriz de entrada.

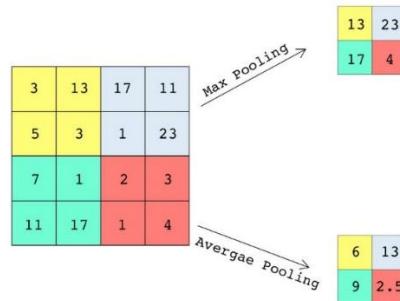


Figura 3.10: Operación pooling para una matriz 4x4 con filtro de pooling con tamaño 2x2.

- La capa Fully Connected o Dense están ubicadas al final de la red, donde cada neurona en una capadense, está conectada a todas las neuronas de la capa anterior, obteniendo como parámetro de entrada las características extraídas, realizando la clasificación y determinando la categoría o clase mediante una función de activación convirtiendo las salidas en probabilidades o a la clase que pertenece la entrada original.

Actualmente, este tipo de redes se emplean en aplicaciones cotidianas como sistemas de vigilancia (Sánchez, Diez, & Heredia, 2020), de análisis de imagen médica (Badillo, Hernández, Narváez, & Trillos, 2021) o de detección de objetos.

### 3.3.2 Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (RNN) (Gamco, s.f.) están diseñadas principalmente para analizar secuencias de datos, donde la ordenación y dependencia tanto espacial como temporal son importantes. La información en las RNN fluye en bucle, es decir, la salida en un momento dado se procesa como entrada en el siguiente momento.

Una característica clave en las RNN es su capacidad de almacenar la información de los pasos anteriores mediante el denominado, estado oculto. Existen dos tipos de RNN, de Corto y Largo plazo.

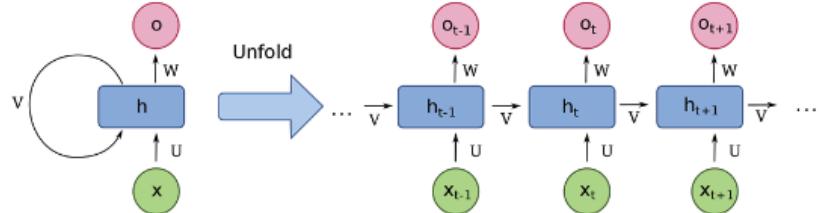


Figura 3.11: Arquitectura RNN.

Las Redes Neuronales de Memoria a Corto Plazo permite capturar dependencias en los datos, donde la influencia del estado oculto decae a medida que aumenta el tiempo, por lo que los pasos más próximos generarán un mayor impacto en la salida.

Las Redes Neuronales de Memoria a Largo Plazo aparecen para eliminar la limitación de capturar dependencias en los datos en las RNN estándar debido al desvanecimiento del gradiente, que se vuelven pequeños y dificulta el aprendizaje a medida que transcurre el tiempo. Aparecen dos soluciones: LSTM y GRU.

Para entender las figuras 3.13 y 3.14, debemos de tener en cuenta la siguiente figura:



Figura 3.12: Leyenda para arquitecturas de la figura 3.13 y figura 3.14.

- **LSTM:** Diseñadas para manejar dependencias a largo plazo mediante celdas de memoria, que almacenan información durante un periodo prolongado. Esta celda está controlada por 3 puertas: Puerta de entrada, puerta de olvido y puerta de salida, que informan agregan, eliminan o generan de la celda de memoria.

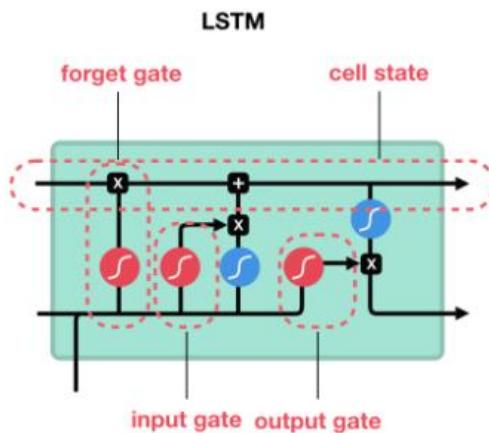


Figura 3.13: Arquitectura LSTM.

- **GRU:** Diseñada para resolver el problema del desvanecimiento de gradiente en las RNN, considerándose una simplificación de LSTM. Emplean dos puertas, la puerta de actualización que decide cuánta información pasada debe de transmitir en el siguiente paso y la puerta de reinicio que se emplea para decidir cuánta información pasada se debe olvidar.

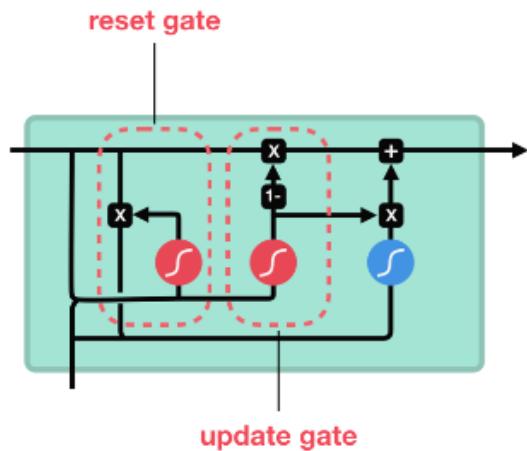


Figura 3.14: Arquitectura GRU.

Esta capacidad de procesamiento de datos secuenciales hace que este tipo de red sea útil para aplicaciones como reconocimiento de voz, traducción de texto y generación de texto. Algunos ejemplos de aplicaciones son Google Translate, Asistentes virtuales y OpenAI GPT-2.

### 3.3.3 Redes Generativas Adversarias

Las Redes Generativas Adversarias (GAN) (Goodfellow, y otros, 2014) son un tipo de redes introducidas en 2014 y se emplean para que un cierto dato se adopte para otro conjunto de datos, es decir, busca que dada una entrada se asemeje demasiado a la distribución deseada y que el dato generado sea indistinguible comparado con un dato real de dicha distribución.

Las GAN están compuestas por dos ANN que compiten entre sí:

- **Red generativa:** Emplea el ruido aleatorio como entrada y genera datos falsos muy semejantes a los reales, es la que se encarga del trabajo creativo y se ve obligada a mejorar para conseguir engañar a la segunda ANN (red discriminadora) o que le apruebe la tarea. Esta red es capaz de realizar millones de test hasta que la Red discriminadora acepta el resultado.
- **Red discriminadora:** Toma datos reales como dato de entrada y trata de distinguir entre ellos, revisando las creaciones de la primera red y moderando. Esta es mucho más precisa ya que los modelos actuales tienen más capacidad para reconocer y refinar imágenes que para crear.

En la figura 3.15 se muestra la arquitectura de una GAN:

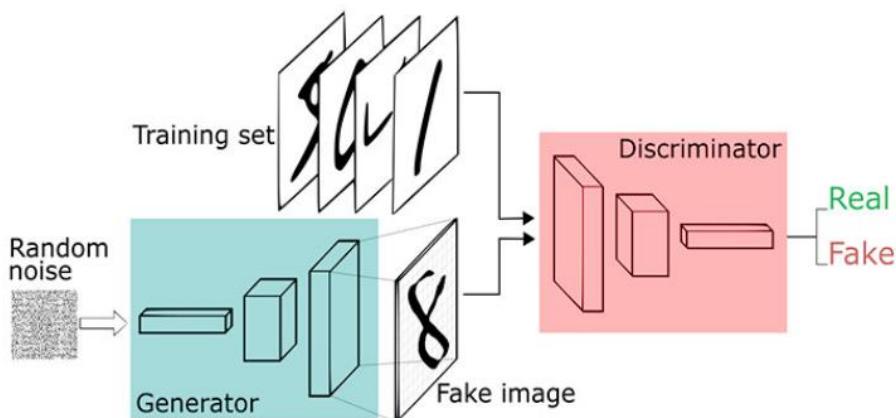


Figura 3.15: Arquitectura de una GAN.

Al inicio del entrenamiento los datos generados por la Red generativa serán significativamente distintos a los datos reales, entonces no será complicado para la Red discriminadora, distinguir los datos reales de los generados y el error en ambos casos será diminuto.

Sin embargo, en la etapa final del entrenamiento, la Red generativa puede ser capaz de generar datos muy realistas y la Red discriminadora se vuelve extremadamente eficiente, aunque ya no puede distinguir claramente entre datos reales y generados, debido a la alta capacidad creativa de la Red generativa. Este equilibrio entre la generación y discriminación hace que este tipo de redes sean muy útiles en aplicaciones para generar datos realistas, como imágenes, videos, texto, sonido y otros varios.

### 3.4 Transfer Learning

El Transfer Learning (TL) (Geeksforgeeks, 2024) es un método de ML que aprovecha el entrenamiento de una red adquirida en una tarea determinada para emplear como punto de partida en el desarrollo del modelo con otra tarea diferente pero relacionada. En el contexto de DL, su uso es muy popular debido a su efectividad y capacidad de mejorar el rendimiento de los modelos y acelerar los procesos de entrenamiento.

En DL, el uso de TL es especialmente útil cuando se parte de un conjunto de datos limitado para la nueva tarea, ya que el modelo empleado como punto de partida es entrenado con un conjunto de datos grandes y diversos.

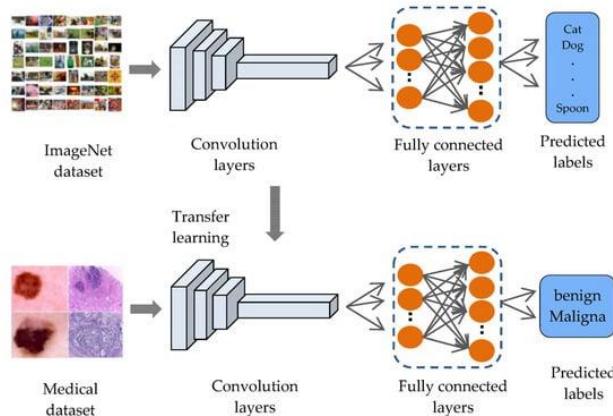


Figura 3.16: Esquema TL en DL.

Algunas aplicaciones de TL dentro de DL son las siguientes:

- **Modelo Preentrenado:** Emplear una CNN preentrenada con un conjunto de datos grande, como *ImageNet*. Esto permite aprovechar las características y patrones que la red ha aprendido de ese conjunto de datos amplio y diverso.
- **Fine-Tuning:** Reemplazar y entrenar las capas finales del modelo con el nuevo conjunto de datos específico para la nueva tarea.
- **Congelación de capas:** Se congela las primeras capas del modelo ya que aprenden características muy genéricas, mientras que las últimas se ajustan a la tarea específica.

Aunque TL es más común emplearlo con DL gracias a la capacidad de aprendizaje en modelos complejos, también se emplea en ML mejorando modelos de NLP ajustándose a otras tareas de NLP.

Algunas técnicas de TL en ML son las siguientes:

- **NLP:** En modelos preentrenados de grandes textos, pudiendo ser entrenados para tareas más específicas como la clasificación de texto y la traducción automática.
- **Reconocimiento de Voz y Audio:** Empleado en modelos con grandes datos de audios para ajustarse a tareas específicas como la detección de sonidos específicos o reconocimiento de voz en distintos idiomas.

### 3.5 Arquitecturas

En este apartado se introducirán las arquitecturas que se emplearán en este proyecto, como VGGNet, ResNet, EfficientNet y Inception.

Antes de introducir las arquitecturas, en la siguiente tabla 3.1 se muestran las arquitecturas con su porcentaje de precisión y la cantidad de parámetros.

Nombre	Precisión (%)	Cantidad de parámetros (millones)
<b>VGG16</b>	71.3	138.4
<b>VGG19</b>	71.3	143.7
<b>ResNet152V2</b>	78.0	60.4
<b>InceptionV3</b>	77.9	23.9
<b>EfficientNetV2B3</b>	82.0	14.5M

Tabla 3.1: Información de las arquitecturas.

#### 3.5.1 VGGNet

Este tipo de arquitectura en DL fue de las primeras en aparecer, introducida por Simonyan y Zisserman en 2014, mediante el artículo titulado *Very Deep Convolutional Networks for Large-Scale Image Recognition* (Simonyan & Zisserman, 2014).

En la siguiente figura 3.17, se muestra cómo se compone la arquitectura internamente:

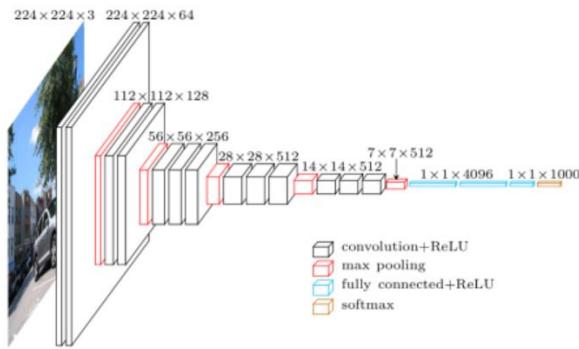


Figura 3.17: Arquitectura VGGNet internamente.

Esta arquitectura emplea bloques con un número progresivo de capas convolucionales con filtros de tamaño 3x3. Además, para reducir el tamaño de los mapas de activación se intercalan bloques de *max pooling*. Finalmente, se emplean dos capas densas de 4096 neuronas cada una y una última de salida de 1000 neuronas.

Respecto a la arquitectura VGG16 y VGG19, el número 16 y 19 representan el número de capas con pesos que tiene la red, las convolucionales y densas. En la siguiente figura 3.18, se muestran las diferencias de capas entre la arquitectura VGG16 y VGG19.

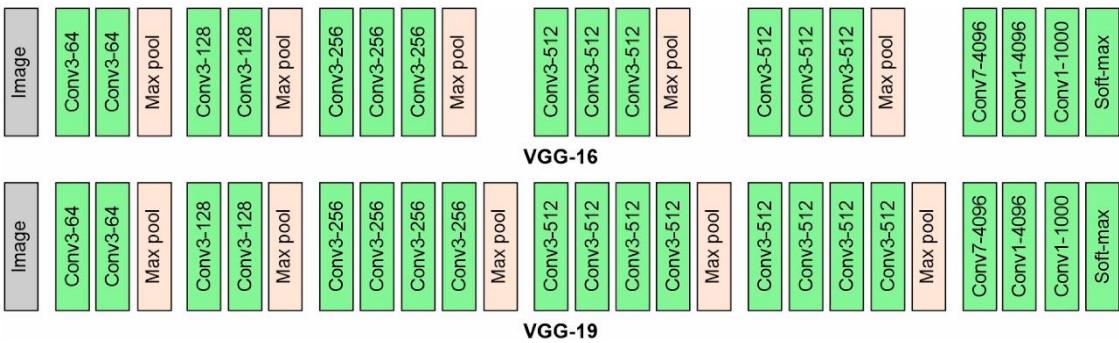


Figura 3.18: Arquitecturas VGG16 y VGG19.

### 3.5.2 ResNet152V2

ResNet152V2 es una variante de la arquitectura ResNet o Residual Net que fue desarrollada por un equipo de Microsoft para abordar el problema del desvanecimiento del gradiente en redes profundas, lo que dificultaba el entrenamiento efectivo de redes neuronales muy profundas. Por ello, para permitir que una red pudiera variar su cantidad efectiva de capas, los autores del trabajo introdujeron el concepto de bloque residual.

ResNet152V2 consta de 152 capas que emplean bloques residuales para construir una red muy profunda con la capacidad de evitar los problemas del desvanecimiento de gradiente. La primera capa de convolución inicial que aplica un filtro de 7x7 con 64 canales a las imágenes de entrada, siendo la responsable de extraer las características principales como las texturas y bordes. A continuación, despliega una serie de bloques residuales, cada una con múltiples capas de convolución 3x3. Las primeras etapas contienen bloques con 64 filtros, seguidas de bloques con 128 filtros, y así sucesivamente, aumentando hasta bloques con 512 filtros en las últimas etapas.

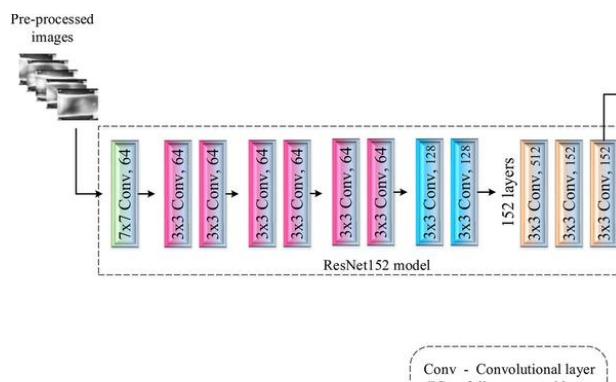


Figura 3.19: Arquitectura ResNet152V2.

### 3.5.3 EfficientNetV2

EfficientNetV2, desarrollada por un equipo de Google, es una arquitectura avanzada de la arquitectura EfficientNet, diseñada para mejorar la velocidad de entrenamiento y la eficiencia computacional.

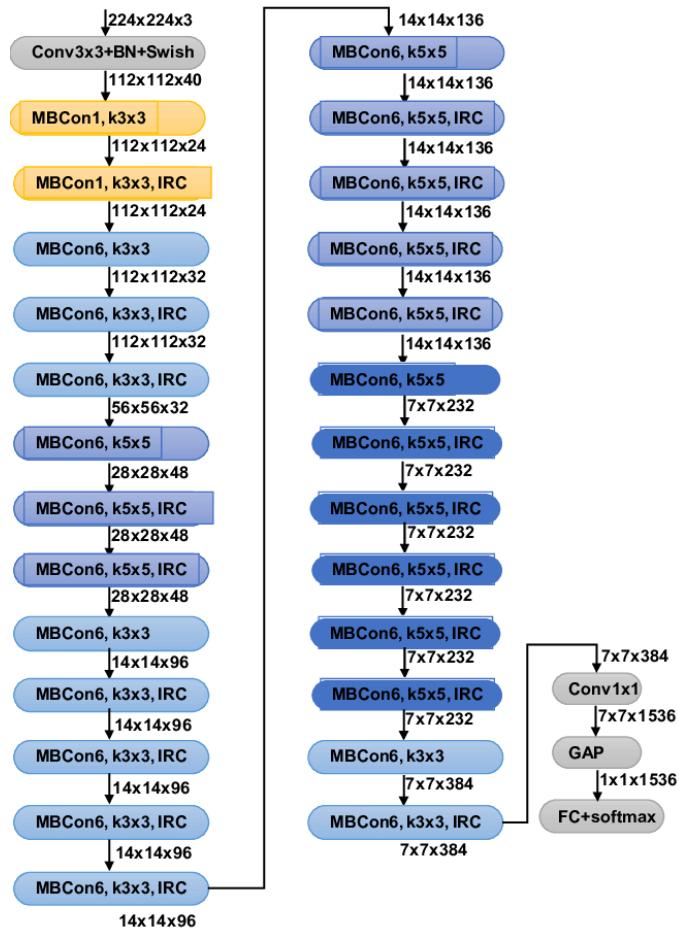


Figura 3.20: Arquitectura de EfficientNetV2B3.

Como se puede observar en la figura 3.20, esta arquitectura tiene una primera capa de convolución  $224 \times 224 \times 3$  seguida de una normalización por lotes y activación Swish convirtiendo la imagen a una dimensión de  $112 \times 112 \times 40$ . A continuación de la primera capa, los bloques MBConv1 (Mobile Inverted Bottleneck Convolution) van reduciendo la cantidad de canales hasta 24, manteniendo las dimensiones de anchura y altura y los bloques MBConv6 aplican una serie de filtros y expansiones, aumentando los canales y reduciendo la anchura y altura.

Algunos bloques MBConv incluyen una reducción de dimensión, conocida como IRC (Inverted Residual and Convolution). Además, emplean filtros kernel 5x5 y 3x3, manteniendo la resolución de 14x14 y reduciéndose más adelante a 7x7, e incrementando los canales hasta 384, que son los que se encargarán de obtener las características más profundas de las imágenes.

Finalmente, se aplica una última capa convolucional 1x1 con 1536 canales, seguida de una capa GAP, siglas de Global Average Pooling y la capa fully connected con función de activación softmax.

### 3.5.4 InceptionV3

InceptionV3 es una de las versiones más avanzadas de CNN de la familia Inception, desarrollada por Google. Uno de los aspectos más importantes en esta versión de la arquitectura son los bloques de Inception que combinan múltiples bloques convolucionales de dimensiones 1x1, 3x3 y 5x5 y operaciones de pooling, tanto *max-pooling* como *avg-pooling*, ejecutadas en paralelo, formando una CNN con 48 capas de profundidad. Esto permite detectar características a diferentes escalas.

El primer bloque Inception, tiene una entrada con un tamaño de 229x229x3. Después de los bloques Inception, la arquitectura incluye capas de reducción y capas densas con función de activación softmax, que terminan en una capa de salida para la clasificación con tamaño de 8x8x2048. Estas capas finales toman las características extraídas y las utilizan para obtener la predicción del modelo.

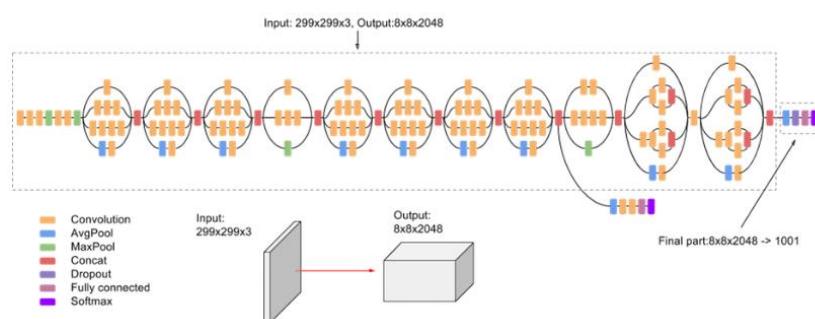


Figura 3.21: Arquitectura InceptionV3.

En la figura 3.21, se puede observar cómo están distribuidas en paralelo las capas convolucionales y las operaciones de pooling que convergen entre sí después de cada bloque hasta la salida final que incluye una desactivación de neuronas, capa *fully connected* con función de activación softmax.

## 3.6 Aplicaciones

En la actualidad el reconocimiento visual de aves utilizando DL ha avanzado notablemente gracias a los desarrollos de las CNN, aplicaciones y conjuntos de imágenes etiquetados. En este subapartado se abordarán los métodos, tecnologías y aplicaciones que se emplean en la actualidad para el reconocimiento visual de aves mediante imágenes.

### 3.6.1 Cornell Lab of Ornithology

El laboratorio de ornitología de Cornell (Cornell University, s.f.) se fundó en 1910 en Ithaca, Nueva York, Estados Unidos por unos miembros de la universidad que se centran en el estudio de aves y otros animales salvajes, así como de la naturaleza.

El principal objetivo de Birds Cornell es apoyar iniciativas que ayuden a la conservación de las aves y la educación en las ciencias y acciones por el medio ambiente para que impacten positivamente en la biodiversidad e inspirar a las personas, sin importar las edades y el sector que desempeñen, a cuidar la naturaleza y su entorno. Para ello, ofrece una gran cantidad variada de recursos públicos con los que se podrán obtener gran cantidad de datos, desde imágenes, audios, videos y seminarios, hasta aplicaciones para identificar aves mediante imágenes y sonido.

Unos ejemplos de aplicaciones sobre el reconocimiento de aves mediante imágenes son los siguientes:

- **eBird:** Base de datos que contiene millones de datos sobre aves.
- **Merlin Bird ID:** Herramienta diseñada para identificar aves.
- **Bird Academy:** Imparte seminarios creados para poder entender y conectar con las aves.
- **Living Bird:** Es una revista trimestral que publica artículos sobre investigaciones y conservación de las aves.

En los siguientes apartados se profundizará en eBird y Merlin Bird ID.

### *eBird*

eBird empezó como una simple idea en el laboratorio de Ornitología de Cornell hasta convertirse en uno de los proyectos científicos de biodiversidad más grande del mundo, gracias a su expansión en 2010.

Actualmente con el Gobierno de España, específicamente con el Centro Nacional de Educación Ambiental (CENEAM) donde el objetivo principal es la conservación y educación de las especies.



Figura 3.22: Inicio de la página web de eBird. (<https://ebird.org/explore>)

eBird tiene una base de datos de observaciones sobre las aves, con un conteo de más de 100 millones de imágenes aportadas por científicos, ornitólogos y aficionados de todo el mundo. Estos datos no son solo imágenes, audios y videos, sino también la distribución, abundancia, hábitat, comportamientos, localización y muchos más.

Esta base de datos es abierta y gratuita a aficionados, ebird cuenta con expertos en aves que realizan una rigurosa tarea de chequeo para garantizar la precisión, relevancia y la veracidad de los datos proporcionada por observadores.

En resumen, gracias a este proyecto se ha revolucionado la forma en la que se obtienen y emplean los datos sobre las aves y ha generado una comunidad global de observadores y científicos que contribuyen a la conservación de las especies y la biodiversidad.

## Merlin Bird ID

Merlin Bird ID es una aplicación móvil gratuita implementada por el Laboratorio de Ornitología de Cornell que permite reconocer aves mediante imágenes, audios y descripciones de su aspecto, tamaño y comportamiento.



Figura 3.23: Logo de Merlin Bird ID.

En las figuras 3.24, 3.25, 3.26 y 3.27 mostramos el funcionamiento de la aplicación y los pasos que sigue:

- **Paso a paso:** reconoce aves mediante tamaño, color y comportamiento y genera un listado de las posibles aves que tienen esas características.



Figura 3.24: Proceso para identificar un ave mediante la funcionalidad Paso a paso.



Eurasian Blackbird

Común y generalizado en hábitats boscosos, parques, jardines y tierras de cultivo con setos que a menudo se alimenta en los campos y en el césped. Tiene canto afautado y melódico que a menudo se escucha en los b...

Figura 3.25: Resultado de Paso a paso.

- **Audio:** reconoce aves mediante un audio en tiempo real y aparece una alerta cuando lo ha detectado.

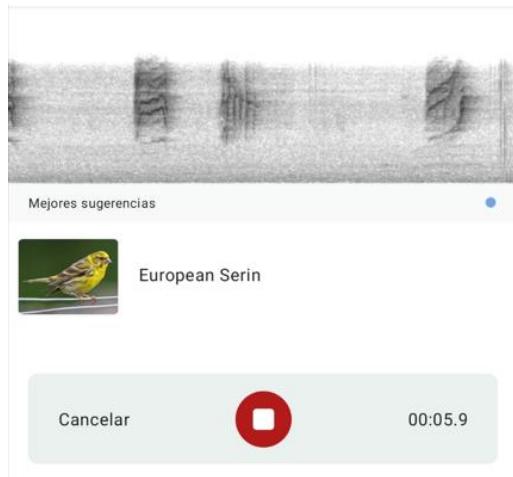


Figura 3.26: Proceso para identificar un ave mediante audio.

- **Foto:** reconoce aves mediante imágenes en tu dispositivo o en tiempo real, puedes indicar la ubicación y la fecha o que no la sabes.

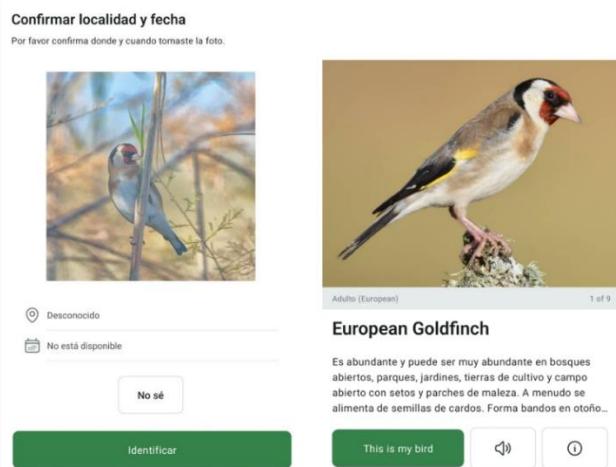


Figura 3.27: Proceso de reconocimiento de aves mediante fotos.

Actualmente, Merlin Bird ID es una de las mejores aplicaciones para el reconocimiento de aves, tanto visualmente como por audios. Es muy completa y precisa en sus análisis, gracias a su extensa base de datos y tecnología avanzada de IA.

### 3.6.2 iNaturalist

iNaturalist es una plataforma tanto web como móvil que fue desarrollada en el año 2008 como proyecto final de postgrado de Nate Agrin, Jessica Kline y Ken-ichi para la Academia de Ciencias de California en Berkeley. Permite a los usuarios llevar un registro de la biodiversidad y compartirlos a nivel mundial.

Los objetivos de esta plataforma es promover la ciencia ciudadana, recopilar datos para crear una base de datos global para proteger la biodiversidad y a su vez emplearse para conservacionistas, educadores e investigadores, aumentando el conocimiento público sobre la concienciación ambiental.

El reconocimiento automático de aves no es su única funcionalidad, ofrece una amplia cantidad de funcionalidades que no están limitadas a aves, pero es la que interesa en este TFG. En el Apartado 5.1.2 se profundizará en las funcionalidades específicas de reconocimiento visual de aves y a continuación, mencionaré algunas de las funcionalidades que tiene esta plataforma:

- **Registrar observaciones:** permite que los usuarios suban imágenes, audios y videos desde plantas y hongos hasta insectos o animales, con una serie de características como podría ser la ubicación precisa.
- **Comentarios:** los usuarios tienen la posibilidad de comentar y discutir sobre las observaciones registradas, proporcionando correcciones o ampliando la información sobre la especie observada.
- **Proyectos:** los usuarios tienen la posibilidad de crear y unirse a proyectos sobre cualquier especie en específico.

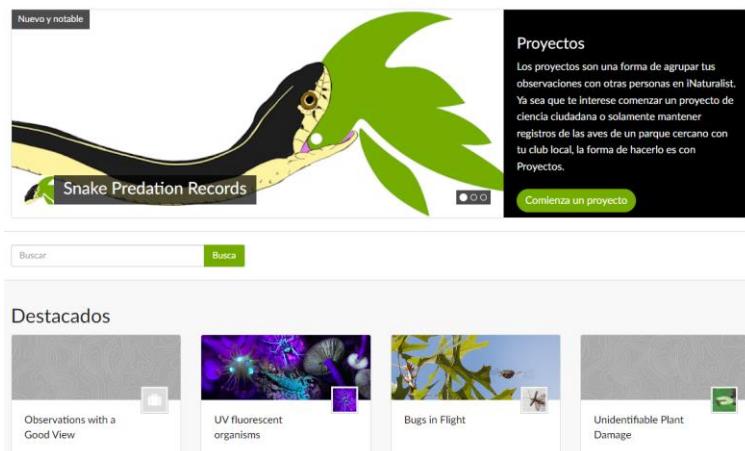


Figura 3.28: Funcionalidad de creación de un proyecto en la web de iNaturalist.

- **Herramienta de exportación y análisis:** los usuarios pueden exportar datos en formato CSV con gran cantidad de información pudiendo filtrar para limitar los datos extraídos y se puede visualizar un mapa de distribución geográfica de las observaciones registradas por los usuarios.

En resumen, iNaturalist es una plataforma gratuita muy poderosa para cualquier persona interesada en la conservación y el estudio de la biodiversidad, abarcando gran variedad de especies, no solo de aves.

## 4. Metodología

En este apartado se describen los componentes clave necesarios para el desarrollo y ejecución del proyecto. En el subapartado 4.1 se detallan las especificaciones del hardware utilizado, destacando su capacidad de procesamiento y memoria RAM relevantes para el rendimiento del sistema. Posteriormente, en el subapartado 4.2 se aborda el software empleado, donde se destaca el uso de Python con la distribución Anaconda para la gestión eficiente de entornos y paquetes. Además, se menciona la optimización en GPU utilizando CUDA Toolkit y cuDNN, junto con diversas librerías específicas que contribuyen al desarrollo del proyecto. Finalmente, en el subapartado 4.3 se presentan los conjuntos de datos utilizados, incluyendo detalles sobre su origen, cantidad de imágenes y características relevantes que afectan al entrenamiento y validación del modelo.

### 4.1 Hardware

Para la realización de este TFG, se ha empleado un ordenador de sobremesa y los recursos proporcionados por Google Colab.

Las especificaciones del dispositivo más importantes del ordenador de sobremesa se muestran en la siguiente tabla.

Componente	Especificación
CPU	Intel(R) Core (TM) i5-10400 CPU @ 2.90GHz 2.90 GHz
RAM instalada	15.91 GB
Tipo de sistema	Sistema operativo de 64 bits, procesador basado en x64
GPU	NVIDIA GeForce RTX 2060
Versión del controlador	555.85
Edición de Windows	Windows 10 Home
Versión de Windows	22H2
Compilación del sistema operativo	19045.4474

Tabla 4.1: Especificaciones del ordenador sobremesa empleado.

En la siguiente Tabla 4.1 se muestran las GPUs, CPUs y TPU que ofrece Google Colab Pro para la ejecución en entornos con Python:

Nombre	Tipo	RAM del sistema (GB)	RAM (GB)	GPU	Disco (GB)
	Acelerador				
CPU	CPU	Depende del modelo	Depende del modelo	No	Depende del modelo
TPU	TPU v4-32	32	32	TPU v4-32	200
A100	A100	32	16	NVIDIA Tesla A100	120
V100	V100	32	16	NVIDIA Tesla V100	100
T4	T4	32	16	NVIDIA Tesla T4	100

Tabla 4.2: GPUs CPUs y TPUs que ofrece Google Colab Pro.

### 4.2 Software

#### 4.2.1 Python

Python es un lenguaje de programación de alto nivel en el que se le da mucha importancia a la legibilidad del código. Se trata de un lenguaje interpretado, es decir, que el código escrito no se traduce realmente a un formato legible en tiempo de ejecución.



Figura 4.1: Python para IA.

Python es uno de los lenguajes más empleados para programar IA de reconocimiento de imágenes y el más empleado actualmente y con bastante diferencia, gracias a sus robustas bibliotecas y frameworks existentes, como TensorFlow, OpenCV, Keras, PyTorch, entre otros, la comunidad activa de desarrolladores que comparten sus conocimientos y soluciones y la gran flexibilidad que tiene para diferentes métodos de desarrollo y tipos de proyectos.

Actualmente, empresas líderes como *Google*, *Amazon* y *Netflix* están empleando la combinación de Python e IA para impulsar la mejora e innovación de la eficiencia. Incluso una de las aplicaciones más revolucionarias de estos últimos años, ChatGPT, está mayormente programada en Python junto a otros lenguajes.

La versión empleada es la 3.8.18.

#### 4.2.2 Google Colab con Python

Google Colab es un servicio alojado de Jupyter Notebook que no requiere de ningún tipo de configuración y el acceso a recursos como CPU y GPU es completamente gratuito hasta un cierto límite de horas diario.

Google Colab ofrece suscripciones de pago en la que obtienes unidades computacionales que se consumen dependiendo del tiempo que se ejecutan entornos. Existen diversos planes de pago, con diferentes precios, dependiendo de lo que requiera cada usuario, así como obtener unidades informáticas sueltas, uso de TPU, gráficas más rápidas, mejor almacenamiento y aumento en la duración de las sesiones.

La versión de Python instalada en Google Colab es la 3.10.12

Google Colab es una herramienta que ha tomado mucha relevancia para investigadores y desarrolladores en el campo de la IA, específicamente en modelos de aprendizaje profundo, el procesamiento de imágenes y el análisis de datos.

#### 4.2.3 Anaconda Distribution con Python

Anaconda fue creado especialmente para facilitar el uso de Python al análisis de datos empresariales. Desde entonces, el uso del lenguaje de Python ha incrementado con creces siendo el lenguaje más popular hasta el momento. Es una distribución de Python basada mayormente en código abierto que funciona como un gestor de entorno, de paquetes y de los cuales, 720 de ellos son de código abierto.

Anaconda Distribution se distribuye en cuatro sectores que se instalan automáticamente y de manera sencilla, para obtener así una aplicación multiplataforma, Anaconda Navigator, Anaconda Project, Librerías de Ciencia de datos y Conda.

Las ventajas que tiene son múltiples. Permite instalar y administrar paquetes, dependencias y entornos de manera muy sencilla en Python. Además, permite la creación de proyectos empleando múltiples entornos de desarrollo como JupyterLab, Jupyter Notebook, Spyder, VSCode, RStudio, entre otros.

Además de las ventajas comentadas anteriormente, la que interesa para este TFG es que permite compilar Python para una ejecución rápida, facilita la escritura de algoritmos complejos, cuenta con soporte para computación de alto rendimiento y elimina los problemas de dependencia entre los paquetes y el control de versiones. No obstante, siempre es recomendable asegurar las versiones correspondientes a sus dependencias revisando la documentación e instalándolas manualmente.

La versión de Anaconda instalada es la 2024.02-1 y la versión correspondiente de Python en ese entorno es la 3.8.

#### 4.2.4 CUDA ToolKit y cuDNN

El kit de herramientas NVIDIA CUDA permite aprovechar la potencia de las GPU NVIDIA proporcionando un entorno de desarrollo para crear aplicaciones de alto rendimiento aceleradas por GPU que permite desarrollar, optimizar e implementar aplicaciones de aprendizaje automático, supercomputadoras, y otras más.

Las GPUs de NVIDIA están diseñadas específicamente para ejecutar tareas de manera simultánea, es decir, en paralelo que consiste en miles de núcleos de procesamiento pequeños pero lo suficientemente eficientes estando optimizados para manejar tareas en paralelo, lo que las hace ideales para sistemas que requieren de gran número de cálculos.

Respecto a las herramientas y librerías que ofrece NVIDIA CUDA Toolkit son muy numerosas y fundamentales para el procesamiento de alto rendimiento y baja latencia, específicamente diseñadas para el uso con CUDA. Una de las más importantes en el reconocimiento visual de imágenes es cuDNN (NVIDIA Deep Neural Network Library) que es una biblioteca primitiva para las DNN, proporcionando implementaciones altamente optimizadas para desarrollos de forward y backward convolucionales, pooling y normalización.

Esta plataforma (NVIDIA CUDA) y librería (cuDNN) se pueden emplear junto a la aplicación Anaconda Navigator configurando un entorno de Python e instalando las librerías necesarias para aprovechar el aceleramiento por GPU. Esto es realmente efectivo en este TFG ya que el código emplea gran cantidad de cálculos, lo que te permite desarrollar y ejecutar código que se beneficie de la potencia de las GPU NVIDIA de una manera más integrada y sencilla.

Hay que tener en cuenta, mencionado anteriormente en el Apartado 5.1.3 Anaconda Distribution con Python, que las versiones del entorno creado en Anaconda Navigator con Python, de NVIDIA CUDA Toolkit y cuDNN tienen que ser compatibles. En las referencias [3] y [4] se muestran la compatibilidad de las versiones.

La versión de NVIDIA CUDA Toolkit instalada es la 11.2 y la versión correspondiente de Python en ese entorno es la 8.1.0.

#### 4.2.5 Librerías

Todos los scripts empleados para este TFG han sido programados con Python. Por tanto, las siguientes librerías pertenecen a este lenguaje, alguna más empleada que otra, pero todas prescindibles.

##### **Concurrent**

Concurrent se emplea para realizar lanzamiento de tareas en paralelo, pudiendo mejorar el rendimiento de nuestro script o programa. El submódulo empleado es *concurrent.futures* el cual provee una interfaz de alto nivel para realizar invocaciones de manera asíncrona. Estas invocaciones se pueden realizar mediante hilos o procesos independientes, dependiendo de las funciones o necesidades que se requieran.

Esta librería se emplea para realizar descargas de imágenes de manera concurrente mediante el uso de hilos empleando la función *ThreadPoolExecutor* la cual unirá a todos los hilos creados antes de que el intérprete termine la ejecución del programa.

##### **CSV**

CSV (Comma Separated Value) es uno de los formatos más importantes para la importación y exportación de datos ya sea para hojas de cálculo o bases de datos. El módulo csv implementa clases permitiendo la lectura y escritura en formato CSV de manera sencilla y eficiente.

##### **Cv2**

Cv2 hace referencia a OpenCV, una librería de VC de código abierto ofreciendo una alta gama de funciones para el procesamiento de imágenes, de video y muchos más. Profundizando más en el reconocimiento de imágenes, incluyen operaciones para la manipulación de píxeles, modificando el tamaño de las imágenes, convirtiendo los colores y una gran integración con bibliotecas de aprendizaje profundo, como Tensorflow y PyTorch.

##### **Pillow**

Pillow es una biblioteca empleada para la manipulación de imágenes, es decir, agrega capacidades de procesamiento de imágenes proporcionando una amplia compatibilidad de formatos de archivo y capacidades de procesamiento de imágenes bastante potentes. Es una actualización de la librería PIL.

Principalmente se emplea para abrir y cargar imágenes, aplicar filtros, detección de bordes, corrección de colores, procesar imágenes e incluso generar imágenes simples, como gráficos o diagramas y muchas funciones más.

##### **Keras**

Keras es una biblioteca de redes neuronales de alto nivel de código abierto escrita en Python que se ejecuta sobre frameworks como Theano y TensorFlow.

Está diseñada principalmente para que sea modular, rápida y sencilla de emplear, siendo una de las más empleadas en la actualidad ya que facilita enormemente la creación de capas en ANN o la configuración de distintas arquitecturas y cuenta con una gran comunidad de desarrolladores y usuarios que contribuyen en mejoras y documentación.

Una de sus mejores características es que está diseñada para aprovechar el poder de las unidades de GPU, pudiendo acelerar los procesos de modelos de ANN con gran cantidad de datos.

## Math

Math es una biblioteca de las más sencillas y útiles ya que permite el acceso a funciones matemáticas y constantes que son empleadas en operaciones. Estas funciones van desde lo más básico como una suma o resta hasta funciones más complejas como redondeos, logaritmos, trigonometría, exponentes y muchas más.

## Matplotlib

Matplotlib es una de las librerías más exhaustivas y populares para la visualización de datos en Python ya sean estáticos, animados o interactivos. Puede llegar a ser muy potente si se combinan con otras librerías como NumPy, Pandas, SciPy, Seaborn y otras más.

Permite la generación de una gran variedad de gráficos, desde una simple línea o de diagramas hasta modelos volumétricos y 3D. También tiene una gran flexibilidad permitiendo a los usuarios personalizar y combinar diferentes tipos de gráficos, así como añadiendo etiquetas, leyendas, títulos, cambiando colores, estilos y formatos e incluso con el uso de bibliotecas como Ploty o Bokeh generar visualizaciones interactivas.

En resumen, es una de las bibliotecas más empleadas gracias a que tiene una gran flexibilidad y variedad de gráficos y visualizaciones, permitiendo mostrar los datos de manera sencilla y atractiva.

## Mlxtend

Mlxtend (Machine learning extensions) es una librería muy útil para las tareas de data science o ML. Contiene gran variedad de funciones que permiten la evaluación de modelos, preprocesamiento de los datos y visualización de datos que complementan algunas funcionalidades de otras librerías como scikit-learn y matplotlib.

Se ha empleado para la generación de matrices de confusión para evaluar el rendimiento de los modelos de clasificación, que muestra una tabla con las comparaciones de las etiquetas originales con las etiquetas predicha, siendo de importancia para entender el rendimiento y la precisión del modelo entrenado.

## Numpy

Numpy, abreviatura de *Numerical Python*, es una biblioteca creada en 2005 por Travis Oliphant para extender la biblioteca *Numeric* y que da soporte para la creación vectores y matrices multidimensionales junto a una gran colección de funciones matemáticas de alto nivel para poder operar con estas.

Incorpora una nueva clase de objetos llamados **arrays** que permite realizar colecciones de datos del mismo tipo con varias dimensiones y funciones muy eficientes para su manipulación. En la siguiente figura 4.2 se muestra un ejemplo de las estructuras de los **arrays**, siendo axis las dimensiones del array o también conocidos como ejes.

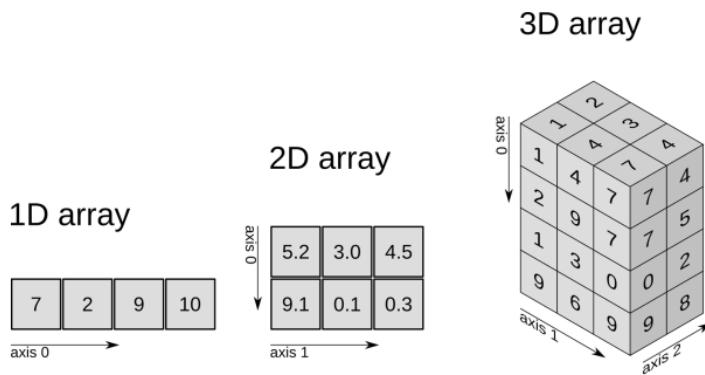


Figura 4.2: Estructura de los arrays con distintas dimensiones.

La ventaja de esta biblioteca es el procesamiento de los arrays que la hacen una buena elección para el procesamiento de vectores y matrices ya que realiza las operaciones hasta cincuenta veces más rápido que las listas.

Actualmente, se ha vuelto una de las bibliotecas fundamentales en la IA, por su capacidad de procesamiento de matrices, en la ciencia de datos, análisis o cualquier área que requiera de cálculos complejos en Python y sus usos van desde extraer datos de un simple array, como el valor máximo o mínimo de un array, hasta operaciones complejas de álgebra lineal, estadística y otras más.

## Os

Os es una de las librerías más empleadas en Python. Permite interactuar con el Sistema Operativo, permitiendo funcionalidades dependientes de este, como manejar directorios y contar la cantidad de elementos en la carpeta.

Es posible realizar operaciones complejas de manera eficiente y portable, lo que la convierte en una herramienta con mucha relevancia para cualquier desarrollador que necesite interactuar con el sistema operativo desde sus scripts de Python.

## Pandas

Pandas es una biblioteca desarrollada por AQR Capital Management. Se convirtió en una librería de código abierto en 2009 y desde entonces se ha convertido en una herramienta fundamental para los científicos, investigadores, analistas y desarrolladores, que ofrece estructuras de datos rápidas, flexibles y expresivas que facilitan el trabajo.

Las estructuras más empleadas son Series y DataFrame, siendo una estructura dimensional como un array en Python y la otra bidimensional siendo similar a una base de datos o una hoja Excel, respectivamente. Además, incluye herramientas de limpieza, fusión y manipulación de datos restantes.

## Scikit-learn

Sklearn es una biblioteca de software libre muy popular de Python que se emplea para el ML, siendo accesible para todos. Está construido sobre las bibliotecas NumPy, SciPy y matplotlib y ofrece múltiples herramientas simples y eficientes para el análisis predictivo de datos.

En la siguiente figura 4.3 se muestra la función que crea una matriz de confusión que evalúa el rendimiento de un modelo de clasificación, mostrando la cantidad de predicciones correctas e incorrectas desglosadas por clases.

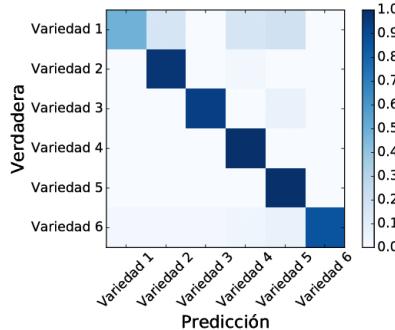


Figura 4.3: Ejemplo de la salida de una matriz de confusión.

Actualmente, es una biblioteca fundamental para la IA y ciencia de datos debido a la facilidad de uso y la amplia gama de algoritmos, desde algoritmos de regresión lineal y algoritmos de clasificación como K-nearest hasta algoritmos de reducción de dimensionalidad.

## TensorFlow

TensorFlow es una biblioteca de código abierto creada en 2015 por Google Brain para ML y DL. Empleando una arquitectura permite crear y entrenar redes ANN para detectar patrones que detectan los humanos.

Es la herramienta que facilita la gestión e implementación de los procesos de ML y DL de forma que desarrollar una ANN sea de forma fácil, sin importar el lenguaje o la plataforma que utilices. Además de trabajar con ANN, TensorFlow es multiplataforma lo que hace posible trabajar con GPUs y CPUs e incluso con las unidades de procesamiento de tensores (TPUs).

Las utilidades de TensorFlow son extensas gracias a su robustez y capacidad de integridad, teniendo un repositorio para modelos preentrenados, llamado TensorFlow Hub, donde los desarrolladores o investigadores pueden reutilizar y adaptar dependiendo de las necesidades específicas que se requieran. Tiene una gran capacidad de manejar grandes volúmenes de datos y cálculos, siendo una de las métricas más importantes en el desarrollo de la IA la cual continuará creciendo debido a la gran demanda de soluciones para diversos sectores.

Actualmente, TensorFlow se emplea en una amplia gama de aplicaciones, desde el reconocimiento de imágenes y NPL hasta el análisis de sentimientos y diagnósticos médicos.

### 4.3 Dataset

El dataset seleccionado para las pruebas iniciales sobre los modelos es NABirds (North American Birds) que contiene una colección de imágenes de 48000 imágenes etiquetadas de 400 especies distintas que son muy comunes en Norte América, incluyendo entre ellas la diferenciación entre machos, hembras y crías, componiendo así la cantidad aproximada de 700 categorías distintas.

Las imágenes seleccionadas para el dataset se obtuvieron mediante la colaboración entre el Laboratorio de Ornitológia de Cornell (Cornell University, s.f.), Cornell Tech, Caltech, la Universidad de Brigham Young y miles de contribuyentes. Las imágenes seleccionadas fueron verificadas para que fuesen de alta calidad y donde las aves fuesen claramente visibles e identificables, eliminando imágenes duplicadas y aquellas que no cumplían los requisitos de calidad necesarios. Una vez filtradas se realizaron anotaciones detalladas, incluyendo la especie del ave, la ubicación de la foto y más aspectos relevantes para el reconocimiento visual de imágenes. Todas las clases seleccionadas tienen la misma cantidad de imágenes, 150 imágenes por clase y tras filtrarlas y realizar las anotaciones se procesaron para estandarizar el formato, resolución y el tamaño. El dataset de NABirds se empleará durante las pruebas iniciales para evaluar el rendimiento de diversos algoritmos de reconocimiento visual, con el objetivo de mejorar la precisión y eficiencia de cada uno de los modelos para detectar aves.

Respecto al dataset para la detección de imágenes en la provincia de Alicante, se ha creado un dataset propio que refleja un considerable esfuerzo y dedicación. Este dataset es la pieza central de este Trabajo de Fin de Grado (TFG) y consta de más de 12,000 imágenes meticulosamente etiquetadas en 24 clases diferentes. Cada clase contiene un número variable de imágenes, garantizando una representación diversa y detallada de las aves locales, mediante herramientas y aplicaciones gratuitas o con licencias gracias a la Universidad de Alicante. En el próximo apartado [5.1](#), se detallarán los métodos específicos de recolección, anotación y procesamiento utilizados para la creación de este dataset, subrayando su calidad y utilidad para futuros estudios.

## 5. Desarrollo

En este apartado se describe el proceso para llegar a la fase de experimentación y pruebas mostrados en el apartado 6. Para ello, se mostrarán los pasos seguidos en cada uno de los apartados enfocados en el reconocimiento visual de aves.

En el apartado 5.1 se detalla la fase de obtención de datos para el dataset propio, donde se analizan fuentes como eBird, iNaturalist y estudios sobre humedales alicantinos para la selección de aves y se detalla cómo se asegura la calidad y precisión del dataset mediante filtros que incluyen la clasificación taxonómica, cantidad de aves por imagen, calidad de imagen y características específicas como sexo, etapa de desarrollo y comportamiento, en el apartado 5.2 se analiza el dataset resultante en términos de cantidad de imágenes y distribución de clases y en el apartado 5.3 se abordan las técnicas y estrategias empleadas para el entrenamiento de modelos de reconocimiento visual de aves.

### 5.1 Obtención de datos para el dataset

En esta fase se profundizará sobre la obtención y análisis de los datos para la creación de un dataset propio. Se explicará el origen de los datos, diferentes formas de recopilación y las aplicaciones o herramientas empleadas, así como *scripts* de ayuda.

#### 5.1.1 Dataset propio

Para la selección de las especies de aves se hizo un análisis a los listados de 2024 de eBird<sup>1</sup>, iNaturalist<sup>2</sup> y artículos sobre humedales alicantinos. Finalmente, tras analizar los 3 listados, se obtuvieron veinticuatro especies de aves para realizar el TFG.

	Nombre
1	AGUA COLINERA
2	ALCA COMÚN
3	AVEFRÍA EUROPEA
4	AVIÓN ROQUERO
5	AVOCETA COMÚN
6	CHARRÁN PATINEGRO
7	CHORLITO DORADO EUROPEO
8	ESTORNINO PINTO
9	FLAMENCO COMÚN
10	FOCHA COMÚN
11	GARCILLA VUYERA OCCIDENTAL
12	GAVIOTA PATIAMARILLA
13	GAVIOTA PICOFINA
14	GAVIOTA REIDORA
15	GOLONDRINA COMÚN
16	GORRIÓN COMÚN
17	GRAJILLA OCCIDENTAL
18	JILGUERO EUROPEO
19	MORITO COMÚN
20	PALOMA TORCAZ
21	PARDELA BALEAR
22	PATO CUCHARÓN NORTEÑO
23	SERÍN VERDECILLO
24	VENCEJO COMÚN

Figura 5.1: Nombre de las especies de las aves.

<sup>1</sup> Listado eBirds 2024: [https://ebird.org/region/ES-VC-AN/bird-list?yr=cur&rank=hc&hs\\_sortBy=count](https://ebird.org/region/ES-VC-AN/bird-list?yr=cur&rank=hc&hs_sortBy=count)

<sup>2</sup> Listado iNaturalist 2024: [https://www.inaturalist.org/observations?place\\_id=30030&verifiable=any&view=species&iconic\\_taxa=Aves](https://www.inaturalist.org/observations?place_id=30030&verifiable=any&view=species&iconic_taxa=Aves)

En la figura anterior 5.1 se muestran las clases de aves escogidas para la creación de un dataset propio.

En el siguiente apartado 5.2, se profundizará en el estudio del dataset, su división de los conjuntos, así como las ventajas y desventajas que asociadas.

Con el objetivo de proporcionar el acceso público al dataset, se ha creado un GitHub<sup>3</sup> dónde este puede consultarse.

### 5.1.2 Recopilación de datos

La recopilación de datos es una parte fundamental para el desarrollo de cualquier proyecto de análisis y modelado de datos, ya que la calidad de los datos influye directamente en la precisión y efectividad del modelo a desarrollar. En el contexto de este TFG enfocado al reconocimiento visual de imágenes hay que prestar atención en aspectos clave como el etiquetado, calidad y la veracidad de la fuente de los datos.

#### Origen de los datos

Existen diversas herramientas de acceso público enfocadas a la recopilación de imágenes de aves, entre otras, que ofrecen colecciones extensas de imágenes etiquetadas por expertos. La mayoría de las imágenes obtenidas provienen de la plataforma de iNaturalist y una minoría de eBird.

iNaturalist contiene una herramienta para exportar observaciones de aves, entre muchas otras, que solo son accesibles si te registras como usuario. Esta herramienta contiene un apartado de consultas para filtrar las observaciones ajustándose a la necesidad del usuario. Entre ellos se encuentran los más importantes para este proyecto, que solo incluya fotos, que estén etiquetadas como ave, el taxón o nombre de la especie y a ser posible que esté revisada, y si queremos profundizar en un lugar específico se puede indicar el continente.

---

<sup>3</sup> Repositorio en: <https://github.com/Juangm14/Dataset-Aves-Reconocimiento-Visual/tree/main>

Figura 5.2: Apartado de consulta de observaciones iNaturalist.

Una vez realizada la búsqueda, nos mostrará un listado de vista previa del ave consultada y otro apartado más para exportar los datos a un CSV.

**3 Seleccionar columnas**

Elija las columnas que desea exportar

Básico ([Todos](#) | [Ninguno](#))

<input type="checkbox"/> id	<input type="checkbox"/> observed_on_string	<input type="checkbox"/> observed_on	<input type="checkbox"/> time_observed_at
<input type="checkbox"/> time_zone	<input type="checkbox"/> user_id	<input type="checkbox"/> user_login	<input type="checkbox"/> user_name
<input type="checkbox"/> created_at	<input type="checkbox"/> updated_at	<input type="checkbox"/> quality_grade	<input type="checkbox"/> license
<input type="checkbox"/> url	<input checked="" type="checkbox"/> image_url	<input type="checkbox"/> sound_url	<input type="checkbox"/> tag_list
<input type="checkbox"/> description	<input type="checkbox"/> num_identification_agreements	<input type="checkbox"/> num_identification_disagreements	<input type="checkbox"/> captive_cultivated
<input type="checkbox"/> oauth_application_id			

Figura 5.3: Apartado selección de columnas para exportar a CSV.

Como se muestra en la figura 5.3, para obtener las imágenes solo necesitamos el enlace de la imagen. Existen muchas más columnas aparte de las mostradas, pero para este proyecto son irrelevantes.

Una vez obtenemos el CSV, hay que descargar las imágenes de la columna exportada `image_url`, para ello se procede a la creación de un script en Python. Se podría realizar manualmente, pero teniendo en cuenta que se exportan más de 900 imágenes en algunas clases de aves, es mejor descartar esta idea y programar un script que nos ayude con la descarga de estas imágenes.

Este script se ha realizado en Python empleando librerías de paralelismo, para que el tiempo del proceso de disminuyese. La librería empleada en este caso ha sido concurrent.futures nombrada en el apartado 5.2.5 *Librerías*. Este script creaba la carpeta con el nombre de la especie del ave en caso de no existir y almacenaba las imágenes descargadas del fichero CSV.

Respecto a la extracción de imágenes mediante eBird, se hizo manualmente para completar algunas clases añadiendo características o puntos de vista diferentes que no nos proporcionaba

iNaturalist por falta de datos en algunas especies, como por ejemplo crías, fotografías al vuelo y perspectivas de frente.

The screenshot shows the eBird search interface with the following details:

- Species:** d. European Goldfinch - *Carduelis carduelis*
- Location:** 41.805, 1824, 406
- Contributor:** More filters
- Recently uploaded:** Done
- Filters (left sidebar):**
  - Age:** Adult, Immature, Juvenile, Unknown
  - Sex:** Male, Female, Unknown
  - Behaviors:** Foraging or eating, Flying, Vocalizing, Feeding young, Carrying food, Molting, Nest-building, Courtship, display, or copulation, Carrying fecal sac, Preening
  - Sounds:** Song, Call, Flight song, Flight call, Non-vocal, Dawn song, Duet
  - Tags:** Nest, Egg(s), Habitat, Hand, Field notes/sketch, Multiple species, Watermark, Back of camera, Dead, No bird
  - Categories:** Environmental, Egg(s), Habitat, People
  - Exotic status:** Native, Naturalized, Provisional, Escaped
  - Rating:** Native, No rating
  - Request:** Commercially requestable media
- Review status:** Confirmed
- Captive:** Not captive
- Collection:** All
- Ictio project:** Ictio - fish of the Amazon Basin
- eBird Checklist ID:** Catalog number(s)
- Specimen:** Specimen collected, Specimen ID

Figura 5.4: Ejemplo de consulta con filtros de eBird.

Para descargar las imágenes, se ha realizado la opción de Google de “*Guardar como*” en la carpeta correspondiente y renombrando la imagen, ya que eran una proporción muy pequeña de imágenes.

### 5.1.3 Filtrado de datos

Una vez descargadas todas las imágenes de iNaturalist, es necesario realizar una revisión de las imágenes para ver que realmente la especie descargada coincide con la esperada y que cumpla una serie de requisitos para que el modelo no tenga problemas en la fase de entrenamiento.

A continuación, se detallarán los filtros que se han aplicado para obtener el dataset definitivo.

### Clasificación taxonómica

La clasificación taxonómica es una tarea compleja y propensa a errores, algunas imágenes están mal clasificadas debido a que no están revisadas y etiquetadas por un experto o no tienen la calidad suficiente para clasificarla. Este tipo de errores suele ocurrir cuando ciertas aves tienen características muy similares.

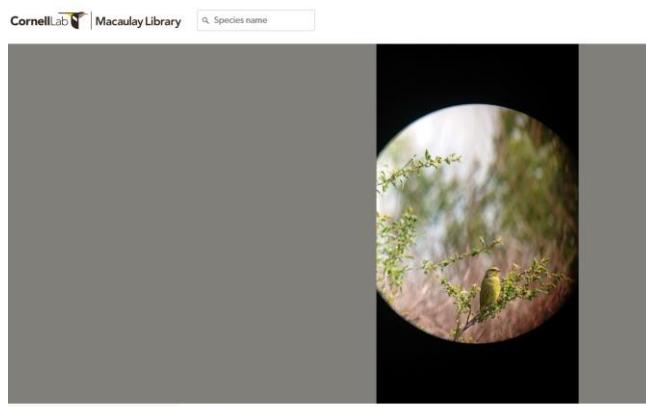


Figura 5.5: Ejemplo de mala clasificación de un Serín Verdecillo extraída de eBird.

### Cantidad de aves

La cantidad de aves que aparecen en una imagen es un aspecto muy importante en el entrenamiento del modelo, puede ser beneficioso si contamos con una gran cantidad de datos sobre esa especie, ya que al entrenar aumenta la variabilidad de los datos de entrenamiento y

permite identificar mejor a la especie, ya que algunas especies es muy complicado ver a una sola ave en la imagen.

Por el contrario, si se trata de un modelo sencillo como este TFG, incluir múltiples aves o multitudes, puede llevar al modelo a aprender características no relacionadas con el ave, datos redundantes o incluso llevar a un sobreajuste en lugar de mejorar y que dificultará la clasificación del ave.



Figura 5.6: Ejemplo de multitud de flamencos común en los arrozales de l'Albufera.

Finalmente, para asegurar la calidad y precisión del modelo se toma la decisión de no incluir aquellas imágenes que incluyan a más de un ave en la imagen, ya sean múltiples aves o multitudes.

### Calidad de imagen

La calidad de las imágenes es fundamental en la detección de imágenes, ya que cuanta mejor calidad tenga las imágenes mayor precisión tendrá el modelo, ya que podrán aprender mejores detalles y características de estas aves, sobre todo en aquellas aves que tengan mucha similitud.

En cambio, si no descartamos imágenes borrosas, el ave muy al fondo de la imagen o de baja resolución pueden hacer que el modelo tenga dificultades para identificar patrones relevantes que identifican claramente al ave.



Figura 5.7: Ejemplo de imagen borrosa de un Jilguero europeo.

En resumen, las imágenes de mala calidad se descargan del modelo ya que es un aspecto que afecta a todas las etapas del entrenamiento del modelo.

## **Sexo y etapa de desarrollo**

En la mayoría de las especies obtenidas en la recopilación de datos, contienen imágenes de diferentes sexos y etapas de desarrollo. Esto es muy importante ya que la mayoría de las aves cambian considerablemente entre las etapas de cría y adulteza, y en algunas de ellas hay diferencias muy notables entre los sexos.

Por ello, se decide descartar aquellas imágenes de cría, machos o hembras que no tengan una gran proporción del conjunto de datos de esa clase de ave ya que habría un desbalance y no identificaría bien las características de esas imágenes.



Figura 5.8: Diferencia entre macho y hembra de Pato cucharón norteño  
(arriba macho y abajo hembra).

Aunque se han descartado, existe una solución para no descartarlas, sería separar los sexos o crías en otra clase y realizar aumento de datos creando diferentes versiones modificadas de las imágenes existentes.

## **Comportamiento**

Una de las características más importantes dentro del reconocimiento de aves, es el comportamiento específico de ellas. La mayoría de las aves tienen comportamientos muy diferentes entre ellas, lo que las hace más distinguibles de sí mismas.

Estos comportamientos no solo se basan en acciones, sino que también puede ser en el entorno en el que se rodean, forma de alimentarse, canto, etc. Este tipo de comportamientos pueden ser captados en cámara y empleados como herramientas efectivas para la clasificación del tipo de ave.



Figura 5.9: Pato cucharón norteño secándose y regulando su temperatura.

En nuestro conjunto de imágenes se han incluido imágenes de comportamientos de las aves, ya sean aleteos, forma de las alas en el vuelo, de alimentación o la manera de beber agua, postura en reposo o incluso el cortejo antes del apareamiento.

### Aves enfermas o muertas

El motivo de incluir imágenes de aves enfermas o muertas es entrenar al modelo para que sea capaz de distinguir no sólo aves sanas, sino también aquellas que presentan enfermedades que comprometen su integridad física, cambiando la forma visual, o de aquellas que han fallecido, pero que aún pueden ser clasificadas sin problema. No todas las clases incluyen este tipo de imágenes, ya que en algunas de ellas no existen o no hay suficientes imágenes para que suponga una mejora en el modelo.

Además, hay que destacar que no se han incluido imágenes de huesos o aves aplastadas.

## 5.2 Análisis de datos

Una vez terminada la extracción y el filtrado de las imágenes, en este apartado se analizará el dataset propio y se abordarán aspectos como la cantidad de imágenes, la distribución de las clases y la división en conjuntos de entrenamiento, validación y de test.

### 5.2.1 Versiones del dataset

En este apartado se mostrará las versiones del dataset, distribución de las clases y la cantidad de imágenes para cada una de las clases.

#### Versión 1.0

En la figura 5.10 se muestra un diagrama de barras con la cantidad de imágenes inicial, eje vertical, por cada clase ordenadas alfabéticamente, eje horizontal.

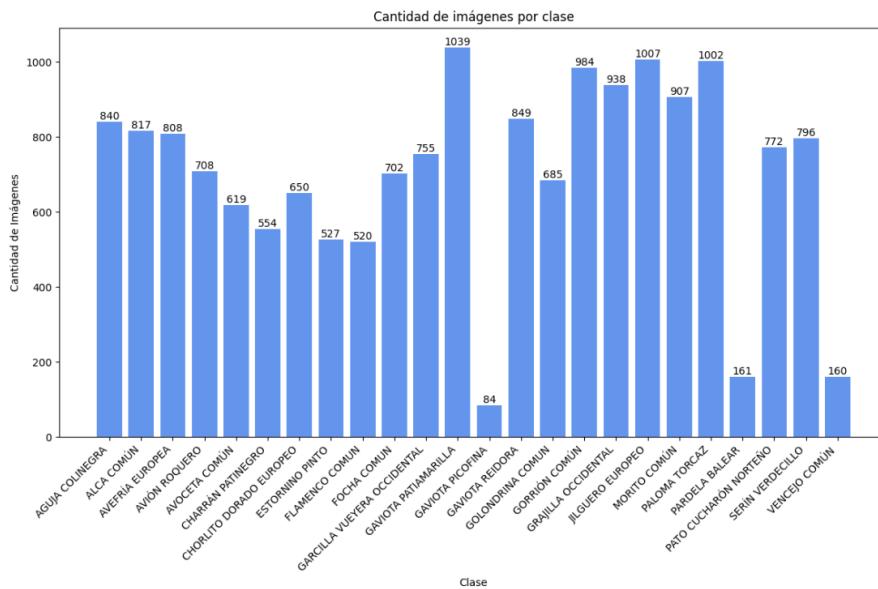


Figura 5.10: Diagrama de barras de la cantidad de imágenes por clase.

Como se puede observar en la figura anterior, todas las tienen una gran cantidad de imágenes, predominando la *Gaviota Patiamarilla*, pero claramente hay tres clases que destacan por la baja

cantidad de imágenes. Estas son la *Gaviota picofina* con ochenta y cuatro imágenes, seguida del *Vencejo Común* con ciento sesenta y la *Pardela Balear* con ciento sesenta y una imágenes.

Estas clases con una cantidad de imágenes muy diferente a las otras puede presentar un inconveniente. Este problema se conoce como desbalanceamiento de imágenes, pudiendo afectar negativamente al rendimiento del modelo, pudiendo disminuir la precisión en estas clases debido a la falta de ejemplos de entrenamiento.

Además, el modelo puede sufrir sobreajustes por la gran diferencia de cantidad de imágenes, generando una falta de generalización, no pudiendo adquirir el correcto aprendizaje de las características de esa clase.

Respecto a las clases restantes, hay que destacar que tienen una cantidad considerable de imágenes lo que hace que el aprendizaje del modelo sea más robusto y generalizable. Con estas cantidades de imágenes el modelo tiene menos probabilidad de sufrir un sobreajuste o más conocido como overfitting, aprender y discriminar características de cada clase y mejorar la precisión de la clasificación.

El dataset o conjunto de imágenes puede considerarse bueno en términos generales debido a la cantidad total de imágenes y a la cantidad de clases, pero analizándolo detalladamente las tres clases que tienen menor cantidad de imágenes tienen una gran similitud con otras clases del dataset. Esto podría afectar de manera que no pueda generalizar bien ambas clases, dando lugar a una mala predicción.



Figura 5.11: Similitud entre Gaviota picofina, gaviota patiamarilla y gaviota reidora.

Como se puede observar en la figura 5.11 estas tres especies de gaviotas son muy similares, por lo que tener muy poca cantidad de imágenes en la clase *Gaviota picofina* no mejoraría el entrenamiento, sino que confundiría más y el modelo no sería capaz de aprender las características de esa clase. Por lo tanto, se procede a eliminar esta clase ya que iNaturalist no tiene más datos y en eBird no están etiquetadas.

## Versión 2.0

En esta nueva versión del dataset solo se cuenta con veintitrés clases, después de excluir la clase *Gaviota picofina*.

En esta versión sigue habiendo un desbalance entre las clases, puntualizando mejor en la clase *Pardela balear* y *Vencejo común*. Al realizar un análisis de la clase de *Pardela Balear*, se concluye que habrá pocos problemas, aunque la cantidad de imágenes sea mucho menor que en las demás, ya que el ave es muy distingible entre las demás y las imágenes son de buena calidad. Por tanto, la clase *Pardela balear* se mantiene en el dataset.

Respecto a la otra clase desbalanceada, el *Vencejo común*, tiene gran similitud con el *Avión Roquero*, ya que son de la misma especie. En la [figura 5.12](#) se muestra la similitud entre ambas aves.



Figura 5.12: Similitud entre Vencejo común (izquierda) y Avión roquero (derecha).

Respecto al inconveniente en el proceso de entrenamiento, presentaría la misma situación que en la *versión 1.0* con la *Gaviota picofina*. Por tanto, se realiza una búsqueda en eBird y se concluye, que como se pueden adjuntar más imágenes de *Vencejo común* al dataset, que se aumentará la cantidad de imágenes para la clase *Vencejo común*.

## Versión 2.1

Esta versión, solo ha variado la cantidad de imágenes en la clase Vencejo común, duplicando las imágenes de la anterior versión, para bajar el porcentaje de desbalanceamiento de clases. En la figura 5.13 se muestra la nueva cantidad de la clase Vencejo común y la cantidad de imágenes en el dataset.

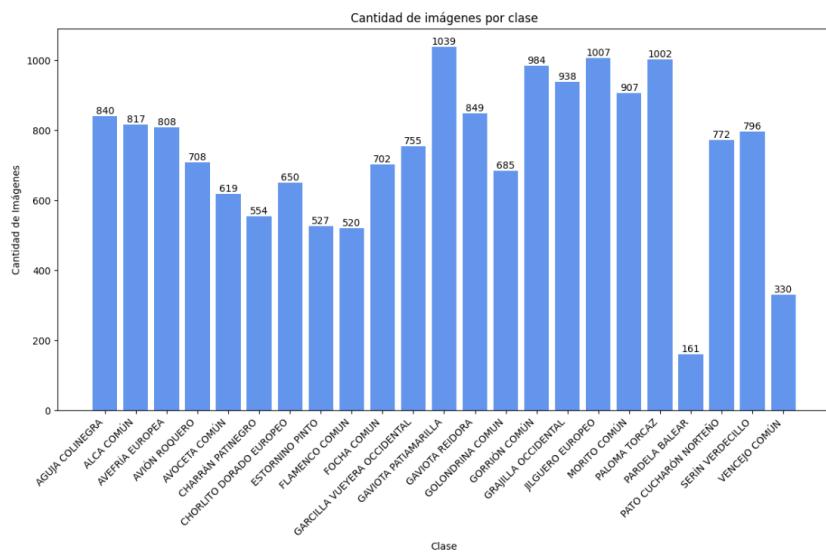


Figura 5.13: Diagrama de barras de la cantidad de imágenes por clase del dataset final.

Este va a ser el dataset final para realizar las pruebas, con una cantidad de 23 clases y 16970 imágenes.

### 5.2.3 División del conjunto

En este apartado se explicará la división del conjunto de datos para la fase de experimentación y de resultados.

Se parte de un conjunto de datos razonable para entrenar, validar y probar modelos, lo que permitirá equilibrar el modelo. Se dispone a dividir el modelo en las siguientes porciones:

- **Entrenamiento:** 70% de los datos. Esta cantidad de datos es suficientemente grande para que el modelo pueda entrenar y aprender características.

- **Validación:** 20% de los datos. Este conjunto permitirá comprobar que el modelo está aprendiendo correctamente durante el entrenamiento, previniendo el sobreajuste del modelo.
- **Test:** 10% de los datos. Se empleará para evaluar el rendimiento final del modelo, siendo una porción de datos totalmente independiente a los otros y no ayudan a mejorar el modelo activamente, pero sí a proporcionar una estimación imparcial del rendimiento.

Esta división del modelo es una de las estrategias más adecuadas en la mayoría de los casos en ML y DL. No obstante, la proporción ideal puede ajustarse dependiendo de las necesidades y características del problema a solventar.

## 5.3 Modelos

En este apartado se detallará el aumento de datos, las arquitecturas empleadas, introducidas en el apartado anterior, y finalmente las técnicas que se emplearán tanto en el dataset de *NABirds* como en el dataset propio.

### 5.3.1 Aumento de datos

El aumento de datos es una técnica en el campo de reconocimiento de imágenes empleando CNN que consiste en generar imágenes nuevas a partir de las imágenes de entrenamiento mediante transformaciones aleatorias. Este tipo de técnica se emplea para situaciones en las que tienes clases desbalanceadas o hay poca cantidad de imágenes en el dataset.

El aumento de datos que se introducen en nuestro proyecto son las siguientes:

#### *Rotación*

La rotación permite generar rotaciones aleatoriamente en las imágenes en un rango. Esto puede ayudar a que el modelo generalice mejor ante diferentes orientaciones de las aves.

Se ha empleado una rotación del 10%, que al ser rango puede rotar valores entre -10 grados o +10° grados.

#### *Zoom*

El zoom realiza un aumento o reducción de la imagen en un rango de forma aleatoria.

Se ha empleado el valor 0.1, que equivale a aumentar o reducir hasta un 10% la imagen.

#### *Width shift y Height shift*

Permite realizar un desplazamiento horizontal o vertical en un valor aleatorio dentro de un rango del ancho de la imagen y en el caso de ser vertical, del alto de la imagen.

Se ha empleado el valor 0.1, que es desplazar vertical y horizontalmente la imagen hasta un 10%.

### *Shear range*

Shear range aplica un cizallamiento aleatorio en las imágenes, es decir, que toma un grado de inclinación distorsionando la imagen, moviendo un conjunto de puntos en una dirección. Es muy parecido a la rotación, pero en este caso fijamos un eje y se estira la imagen en un ángulo.

En la figura 5.14 se muestra cómo afecta shear range tanto al eje x como al eje y.

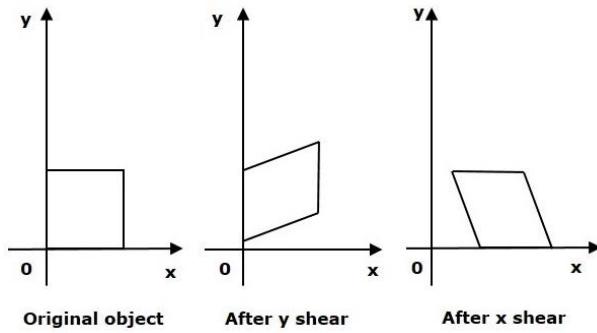


Figura 5.14: Ejemplo de aplicación de la técnica shear range.

Se ha aplicado una inclinación de 10 grados.

### *Horizontal flip y Vertical flip*

Estas técnicas voltean una imagen tanto horizontal como verticalmente. Por defecto, no se voltean, pero si se indica las voltará de forma aleatoria pudiendo voltearse o no la imagen.

Se ha empleado el volteado horizontal, pero no se ha empleado el vertical ya que sería complejo ver a un ave del revés.

### *Fill mode*

Fill mode determina como se quiere que se rellenen los nuevos píxeles obtenidos tras una transformación, como un desplazamiento o una rotación.

Se ha empleado la opción *nearest*, que empleará el valor del píxel más cercano al píxel que se ha transformado

### 5.3.2 Arquitecturas

En el apartado anterior 3.6. se introdujo a las arquitecturas que se emplearán en este proyecto. En este subapartado se detallarán las diferencias de entrenamiento respecto a la arquitectura a emplear.

Todas las arquitecturas empleadas han sido entrenadas con el uso de *ImageNet*, una gran base de datos visual diseñada para su uso en la investigación de software de reconocimiento de objetos visuales. En la siguiente Tabla 5.1 se muestran las diferentes arquitecturas, donde cada una de ellas está formada de forma muy diferente y con una cantidad de parámetros muy variada.

Nombre	Parámetros totales	Parámetros entrenables	Parámetros no entrenables
<b>VGG16</b>	15,316,824	602,136	14,714,688
<b>VGG19</b>	20,626,520	602,136	20,024,384
<b>ResNet152V2</b>	60,740,120	2,408,472	58,331,648
<b>InceptionV3</b>	23,031,608	1,228,824	21,802,784
<b>EfficientNetV2B3</b>	14,736,982	2,162,712	12,574,270

Tabla 5.1: Cantidad de parámetros de las diferentes arquitecturas.

Respecto a los parámetros que se muestran en la Tabla 5.1, son con los que se ha entrenado con el Dataset de NA, congelando las capas de la arquitectura original y realizando un fine-tunning de la última capa en caso de no ser una capa Flatten.

### 5.3.3 Técnicas empleadas

En este subapartado, se explicarán las técnicas empleadas para entrenar el modelo, pudiendo mejorar la precisión, efectividad, rendimiento y lo más importante, evitar el sobreentrenamiento.

#### *Transfer Learning*

Se emplea un modelo preentrenado con un conjunto de datos, en este proyecto se emplea ImageNet. De esta forma se aprovecha el entrenamiento de un modelo para transferir el conocimiento aprendido de una gran cantidad de datos y aplicarlo a un problema en específico, el reconocimiento visual de aves mediante imágenes, lo que mejorará la precisión y eficiencia del modelo.

#### *Fine-tuning*

El fine-tuning es una técnica de entrenamiento que implica la congelación de algunas capas del modelo preentrenado. Se emplea para adaptar el modelo preentrenado para un problema en específico, en este caso, a imágenes de aves.

- **Congelación selectiva de capas:** En el modelo preentrenado se congela para evitar que sus pesos sean actualizados en el proceso de entrenamiento con el nuevo conjunto de datos. Esto evita modificar el aprendizaje obtenido del modelo y ayuda a prevenir el sobreajuste en el modelo con los nuevos datos, las imágenes de aves.

- **Descongelación selectiva de capas:** En el modelo preentrenado se descongelan capas de forma selectiva, frecuentemente las capas superiores, permitiendo un mejor ajuste de pesos en estas capas para que el modelo con los nuevos datos pueda aprender mayor cantidad de características de los datos específicos, las imágenes de aves.

### Cálculo de pesos de las clases

El cálculo de los pesos de las clases se emplea para contrarrestar el desbalance en las clases que tienen las distintas clases del dataset, lo que implica asignar diferentes pesos a las clases. De esta forma, los pesos influyen en la función de pérdida durante el entrenamiento, obligando al modelo para tener en cuenta las clases minoritarias en el entrenamiento.

En las CNN no puede funcionar muy bien y la eficacia puede variar según el nuevo conjunto de datos y el modelo empleado. Se recomienda emplear cuando se tienen clases desequilibradas, pero que no sea extremo.

### Optimizadores

Un optimizador optimiza los valores de los parámetros pudiendo reducir el error cometido por la red neuronal, más conocido como el proceso de *backpropagation*.

En este proyecto se ha empleado el optimizador *Adam* que es una combinación de los algoritmos *AdaGrad* y *RMSProp*

### Técnicas complementarias

Las técnicas complementarias, como las funciones callback son una herramienta para personalizar el comportamiento de un modelo durante el entrenamiento. En este proyecto se han empleado tres técnicas de callback.

### ModelCheckpoint

ModelCheckpoint guarda el modelo o los pesos en algún intervalo, de modo que el modelo guardado se pueda emplear para continuar más tarde con el entrenamiento o para conservar únicamente el modelo que logra el mejor rendimiento. Tiene la posibilidad de monitorear la pérdida o la precisión y si se debe de maximizar o minimizar para el guardado automático.

En este proyecto se ha empleado la monitorización de la precisión del conjunto de validación (val\_accuracy).

### EarlyStopping

Esta herramienta hace que cuando la métrica monitoreada, que se le pasa como parámetro, se detenga cuando haya dejado de mejorar o empeorar durante una indicada espera, cantidad de épocas. Dependiendo la métrica que se esté monitorizando tendremos que emplear una maximización o minimización para indicar cuando debe de detenerse.

Otro parámetro, es la restauración de los pesos de época con el mejor valor de la cantidad monitoreada.

En este proyecto se ha empleado EarlyStopping, con una espera entre 5 y 10 épocas dependiendo la arquitectura empleada, monitorizando val\_loss e indicando que se restauren los mejores pesos.

### **ReduceLROnPlateau**

ReduceLROnPlateau, abreviatura de Reduce Learning Rate On Plateau, monitorea una métrica especificada durante la fase de entrenamiento. Si la métrica deja de mejorar o se estanca durante una cierta espera (cantidad de épocas de entrenamiento indicada por parámetro) entonces reduce la tasa de aprendizaje en un factor por defecto o uno específico para intentar mejorar la tasa de aprendizaje.

Se ha empleado esta técnica para monitorizar el valor de pérdida (val\_loss) con una espera de 5 épocas, un factor de 0.2 y un límite mínimo de 0.00001.

## 6. Experimentación

Una vez puesto en contexto, en este apartado se explicarán las pruebas realizadas para cada una de las arquitecturas. Primeramente, se analizarán las pruebas realizadas con el dataset de *NABirds* para cada una de las cinco arquitecturas elegidas con el fin de obtener las tres mejores arquitecturas y, por último, se analizará la mejor arquitectura con el dataset propio.

### 6.1 Pruebas con el dataset NABirds

En este subapartado se van a analizar las pruebas realizadas a las cinco arquitecturas seleccionadas con el dataset de NABirds. Se han seleccionado 24 clases distintas para la clasificación del modelo, ya que se demoraría mucho en el entrenamiento.

Todas las pruebas se han realizado con la misma cantidad de aumento de datos (escalado, rotación, zoom, etc.), mismo optimizador (Adam por defecto), mismos callbacks y mismas dimensiones de las imágenes 224x224x3.

Respecto a las épocas de entrenamiento, se ha utilizado un límite de 100 épocas, pero eso no implica que llegue a realizarlas todas, ya que se han empleado técnicas de callback, como almacenar un punto de guardado, reducción de aprendizaje y detección estratégica anticipada, para evitar el sobreentrenamiento y mejorar el entrenamiento. Lo que realmente se busca es comprobar cuál de ellas ofrece mejores resultados con los mismos datos, asegurando que el modelo aprenda a generalizar correctamente sin memorizar las imágenes en lugar de las características.

En la siguiente tabla 6.1, se muestran los resultados del entrenamiento obtenidos para cada una de las arquitecturas, empleando TL, aplanando el modelo con una capa Flatten y añadiendo una capa fully connected para adaptar a la cantidad de clases que tiene el dataset de NABirds, 24 clases.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>VGG16</b>	12	0.1521	0.9523	<b>0.4923</b>	0.9167
<b>VGG19</b>	13	0.1937	0.9368	0.5511	0.8917
<b>ResNet152V2</b>	11	0.1414	0.9915	0.6201	<b>0.9667</b>
<b>InceptionV3</b>	8	0.7802	0.9573	1.4921	0.9333
<b>EfficientNetV2B3</b>	14	<b>0.0127</b>	<b>0.9972</b>	1.0023	0.9417

Tabla 6.1: Salida del entrenamiento del conjunto de NABirds con las diferentes arquitecturas.

A continuación, se mostrarán las figuras pertenecientes a los gráficos de entrenamiento de cada una de las arquitecturas escogidas para la fase de experimentación.

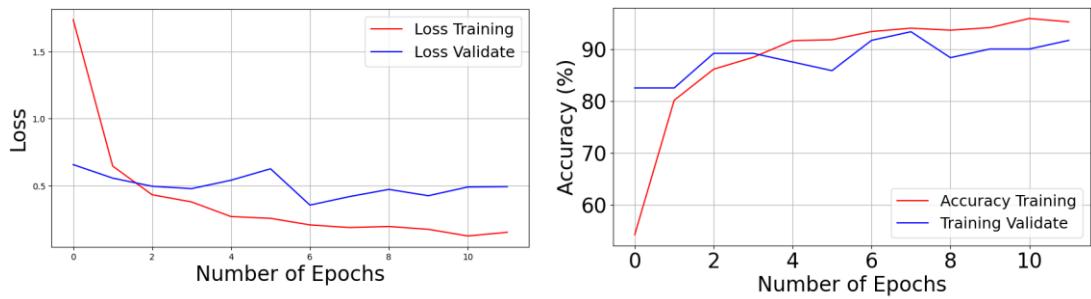


Figura 6.1: Entrenamiento conjunto NABirds con arquitectura VGG16.

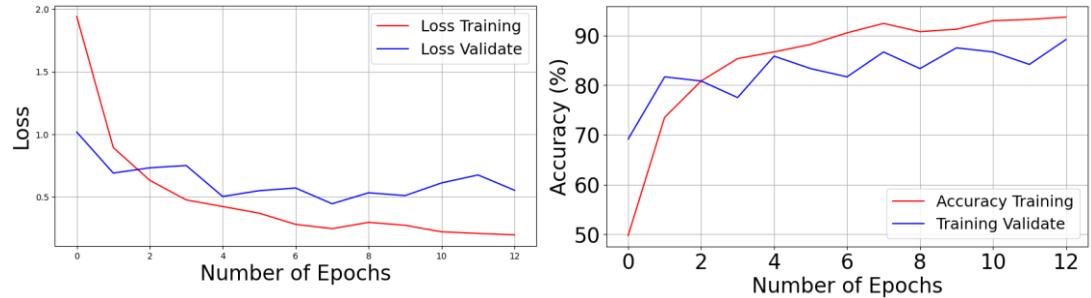


Figura 6.2: Entrenamiento conjunto NABirds con arquitectura VGG19.

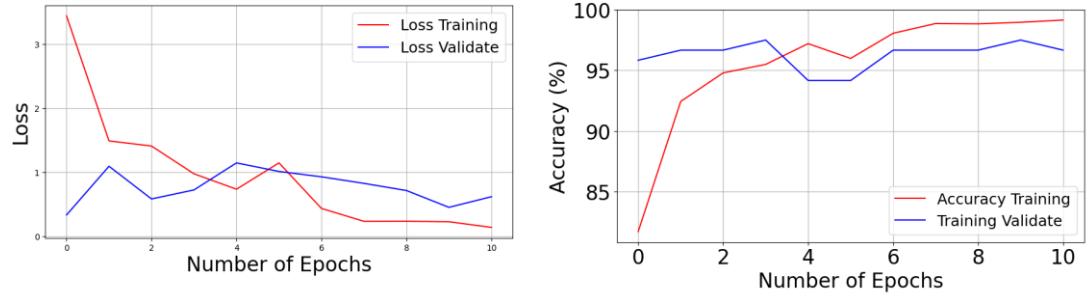


Figura 6.3: Entrenamiento conjunto NABirds con arquitectura ResNet152V2.

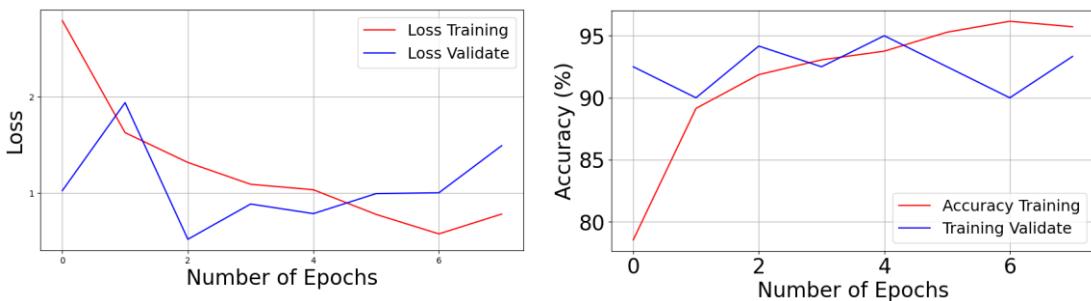


Figura 6.4: Entrenamiento conjunto NABirds con arquitectura InceptionV3.

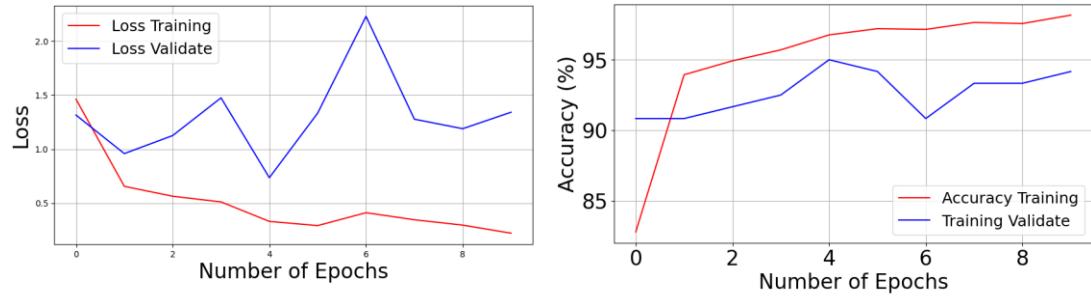


Figura 6.5: Entrenamiento conjunto NABirds con arquitectura EfficientNetV2B3.

Como se puede observar en las figuras anteriores, las arquitecturas que nos dan mejores resultados de entrenamiento y validación, en orden decreciente, son Resnet152V2, EfficientNetV2 e InceptionV3.

El valor de pérdida del conjunto de validación no es nada estable y eso se puede deber a varios motivos, uno de ellos es la cantidad de imágenes de validación respecto a la de entrenamiento es muy escasa y no es capaz de obtener gran cantidad de características, y si se entrenase el modelo por 50 épocas tiene una alta probabilidad a que sobreentrene muy tempranamente. Respecto a la pérdida del conjunto de entrenamiento, se reduce correctamente fluctuando un poco su valor, pero consiguiendo muy buena disminución de la pérdida.

No solo hay que tener en cuenta el entrenamiento final, ya que se ha empleado checkpoint. En la siguiente tabla 6.2 se mostrarán los resultados del entrenamiento y validación hasta el checkpoint y los resultados del conjunto de test que se obtienen empleando el mejor modelo almacenado.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>VGG16</b>	8	<b>0.1874</b>	0.9402	<b>0.4182</b>	0.9333
<b>VGG19</b>	13	0.1937	0.9368	0.5511	0.8917
<b>ResNet152V2</b>	4	0.9779	0.9549	0.7260	<b>0.9750</b>
<b>InceptionV3</b>	5	1.0336	0.9376	0.7850	0.9500
<b>EfficientNetV2B3</b>	5	0.3307	<b>0.9676</b>	0.7353	0.9500

Tabla 6.2: Entrenamiento hasta checkpoint del conjunto NABirds con las diferentes arquitecturas.

Como se puede observar en la Tabla 6.2, las mejores arquitecturas siguen siendo las mismas que en la tabla 6.1, la única métrica que ha cambiado ha sido la pérdida de entrenamiento. A continuación, se mostrarán los resultados que se obtienen a partir de imágenes totalmente distintas a las de entrenamiento y validación, para comprobar la precisión de cada modelo ante nuevas fuentes de datos.

Nombre	Precisión test (%)
<b>VGG16</b>	91,74
<b>VGG19</b>	93.39
<b>ResNet152V2</b>	95,04
<b>InceptionV3</b>	<b>96,69</b>
<b>EfficientNetV2B3</b>	95,87

Tabla 6.3: Resultados conjunto de test NABirds con las diferentes arquitecturas.

Como se marca en la Tabla 6.3 el modelo con mejor precisión en el conjunto de test es InceptionV3 y teniendo en cuenta los porcentajes del conjunto de entrenamiento y validación, se puede determinar que las mejores arquitecturas son InceptionV3, EfficientNetV2B3 y ResNet152V2.

En la figura 6.6, se muestran los resultados del conjunto de test para la arquitectura InceptionV3, ya que es la que ha obtenido mejor porcentaje de precisión en el conjunto de test, mediante una matriz de confusión que muestra las predicciones del modelo sobre cada clase.

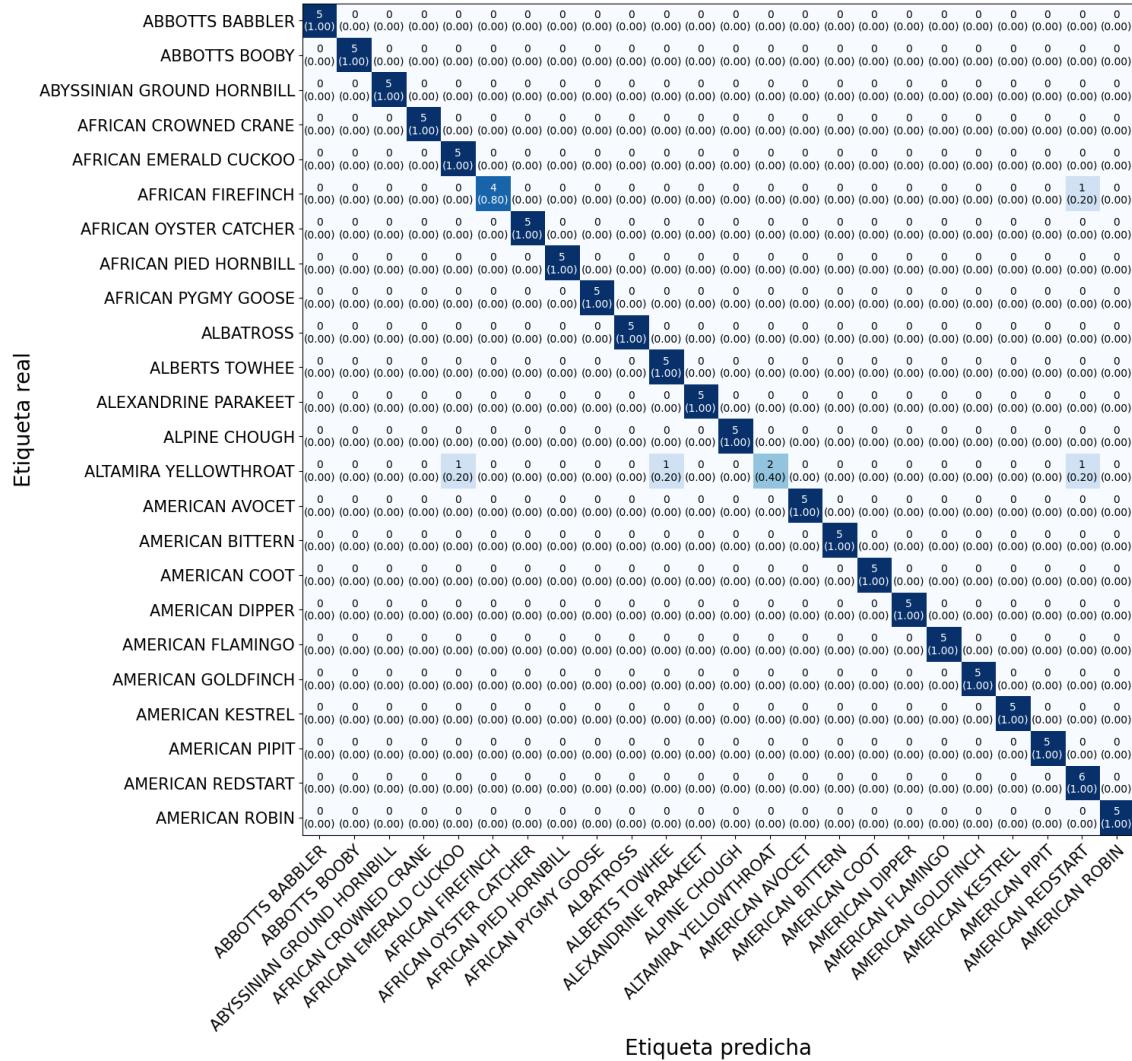


Figura 6.6: Matriz de confusión conjunto de test sobre el modelo InceptionV3.

## 6.2 Pruebas con el dataset propio

En este apartado se realizará la experimentación con el dataset propio y las mejores arquitecturas seleccionadas en el apartado anterior 6.1, todas las pruebas que se llevarán a cabo serán con el mismo aumento de datos, mismas épocas, mismos callbacks, donde solo variarán los parámetros dependiendo de la prueba que estemos realizando. Se experimentará con TL y TL junto con fine-tuning de las últimas capas del modelo.

Respecto a las funciones callback empleadas, serán las mismas para cada una de las pruebas de experimentación, la cantidad de épocas para que se pare el entrenamiento será entre 7 y 10, dependiendo de la arquitectura y si sobreentrena más fácilmente o no.

La aumentación de datos será la misma que se ha empleado en el apartado 6.1. Las épocas de entrenamiento serán 100 como máximo y se empleará el optimizador Adam por defecto.

### 6.2.1 Prueba añadiendo capas densas y dropout

En este caso, se experimentará con TL, realizando un aplanamiento de la salida del modelo base, mediante una capa Flatten, añadiendo una capa fully connected con la cantidad de neuronas de 1024, con función de activación ReLU y con un dropout de las neuronas del 50% que hará que en las diferentes épocas estén activas ciertas neuronas, con la posibilidad de aprender características distintas en cada época y evitar así el sobreentrenamiento y finalmente la capa final fully connected con una cantidad de neuronas equivalente al número de clases que tiene el dataset y con función de activación *softmax*.

En la siguiente tabla 6.4, se muestran los resultados del entrenamiento con las características comentadas.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	15	<b>0.2199</b>	<b>0.9156</b>	<b>0.7131</b>	<b>0.8336</b>
<b>InceptionV3</b>	50	0.7191	0.7465	0.7239	0.7945
<b>EfficientNetV2B3</b>	32	0.5341	0.8166	0.9512	0.7931

Tabla 6.4: Resultados del entrenamiento mediante TL.

A continuación, se presentarán las distintas gráficas obtenidas por cada una de las arquitecturas, teniendo una gráfica con el valor de pérdida y otra con la precisión.

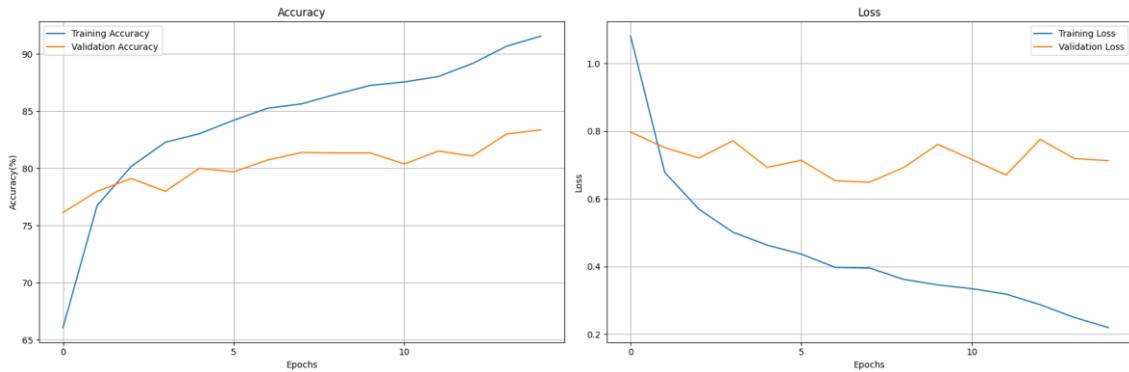


Figura 6.7: Entrenamiento de ResNet152V2 mediante TL.

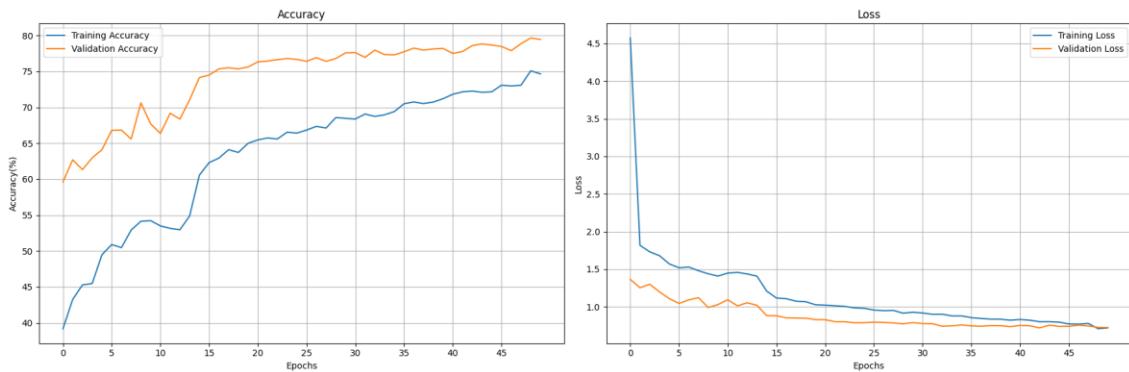


Figura 6.8: Entrenamiento de InceptionV3 mediante TL.

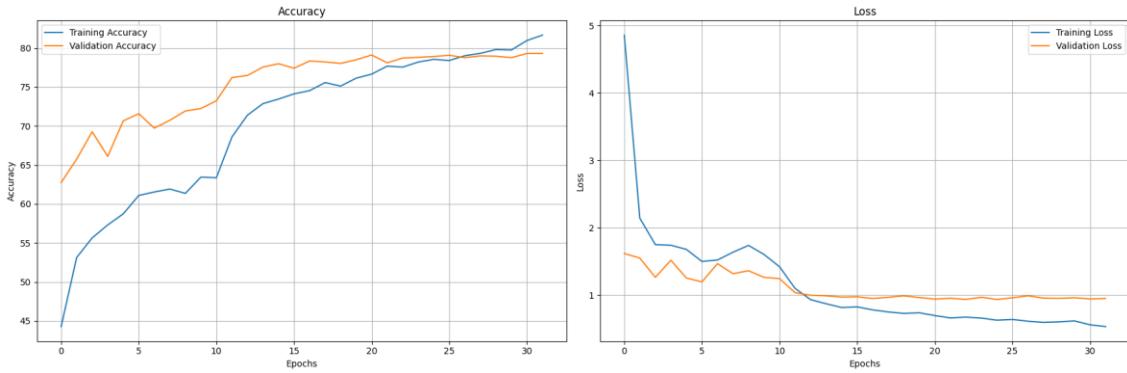


Figura 6.9: Entrenamiento de EfficientNetV2B3 mediante TL.

En las figuras 6.7, 6.8 y 6.9, se puede observar que tanto la precisión de entrenamiento como de validación mejoran notablemente al inicio, pero tras unas cuantas épocas se empieza a estabilizar y los incrementos en la precisión son algo más constantes. Respecto a las gráficas de la pérdida de entrenamiento y validación, inicialmente presentan una disminución brusca y luego se estabiliza hasta el punto en el que la precisión de validación baja muy lenta y finalmente para tras no mejorar en 7 épocas.

ResNet152V2 está sufriendo un sobreentrenamiento temprano, ya que no se ha trabajado con las mejores métricas para esta arquitectura, se puede observar perfectamente en las curvas de accuracy que en vez de juntarse comienzan a separar aumentando la diferencia de precisión entre los conjuntos. Respecto a InceptionV3 y EfficientNetV2B3 parece que puedan estar sufriendo sobreentrenamiento, pero la pérdida de validación disminuye lentamente, y el resto tiene un comportamiento normal.

En la siguiente tabla 6.5 se muestran los valores del entrenamiento en el último checkpoint.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	15	<b>0.2199</b>	<b>0.9156</b>	<b>0.7131</b>	<b>0.8336</b>
<b>InceptionV3</b>	49	0.7080	0.7508	0.7254	0.7965
<b>EfficientNetV2B3</b>	31	0.5601	0.8097	0.9440	0.7931

Tabla 6.5: Resultados con los mejores pesos de entrenamiento mediante TL.

La mejor arquitectura en esta prueba, con mejores resultados de entrenamiento es ResNet152V2, no solo teniendo en cuenta la tabla 6.5 sino también la última época antes de empezar a sobreentrenar como se muestra en la siguiente figura 6.10.

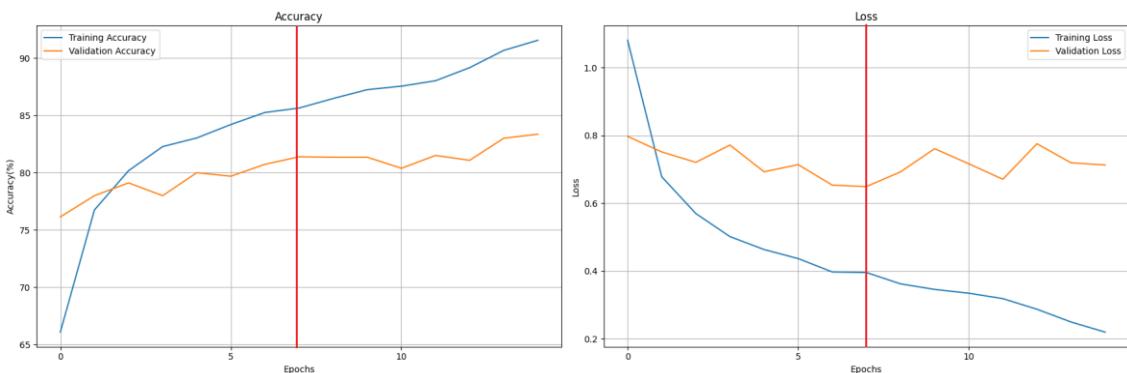


Figura 6.10: Mejor época de entrenamiento en ResNet152V2 con TL antes de sobreentrenar

En la línea marcada en la figura anterior, muestra la época, número 8, donde el modelo comienza a sobreentrenar. En la siguiente tabla 6.6 se muestra resultado de entrenamiento en esa época.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	8	0.3958	0.8564	0.6494	0.8139

Tabla 6.6: Resultados de ResNet152V2 con TL antes de sobreentrenar.

La tabla 6.7 muestra los resultados del conjunto de test, con los mejores pesos entrenados en cada arquitectura.

Nombre	Precisión test (%)
<b>ResNet152V2</b>	<b>86.44</b>
<b>InceptionV3</b>	84.37
<b>EfficientNetV2B3</b>	84.16

Tabla 6.7: Resultados conjunto de test con el dataset propio mediante TL.

Como se puede observar, la mejor arquitectura con este método de entrenamiento es Resnet152V2.

### 6.2.2 Prueba añadiendo capas densas y dropout con regularización de pesos

Esta prueba se ha realizado empleando TL, congelando todas las capas del modelo preentrenado, añadiendo una capa densa con 1024 neuronas con activación ReLU y con un regulador de pesos, penalizando los pesos durante el proceso de entrenamiento, con un factor de penalización de 0.001, ayuda a evitar que los modelos se vuelvan demasiado complejos al mitigar fuertemente los coeficientes grandes en la función de pérdida, ayudando a prevenir el sobreentrenamiento. Se añade otra capa densa con el número de clases y dropout para la desactivación del 50% de las neuronas.

En la tabla 6.8 se muestran los resultados del entrenamiento.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	39	0.6601	0.7906	0.8755	0.8039
<b>InceptionV3</b>	96	<b>0.3188</b>	0.9418	<b>0.7534</b>	<b>0.8494</b>
<b>EfficientNetV2B3</b>	100	0.3711	<b>0.9638</b>	0.8046	0.8459

Tabla 6.8: Resultados del entrenamiento con el dataset propio mediante TL con regulador.

Como se puede observar en la tabla anterior, la introducción de un regulador de pesos con una tasa de 0.001 afecta negativamente a la arquitectura ResNet152V2, a pesar de haberse entrenado durante más épocas respecto al entrenamiento anterior, sin regulador.

En contraste, las arquitecturas InceptionV3 y EfficientNetV2B3 experimentaron una mejora significativa al incorporar el regulador. A pesar de haber prolongado el entrenamiento durante más épocas, en EfficientNetV2B3 llegando al límite de épocas, estas redes lograron mejorar las métricas de precisión, tanto de entrenamiento como de validación y la pérdida de entrenamiento, pero la pérdida de validación solo disminuye en EfficientNetV2B3. Esta prueba se alinea con el

propósito del regulador, el cual es penalizar los pesos para fomentar un aprendizaje más gradual de las características del modelo, y así prevenir el sobreentrenamiento.

A continuación, la siguiente figura muestra los gráficos de precisión y pérdida de los diferentes modelos.

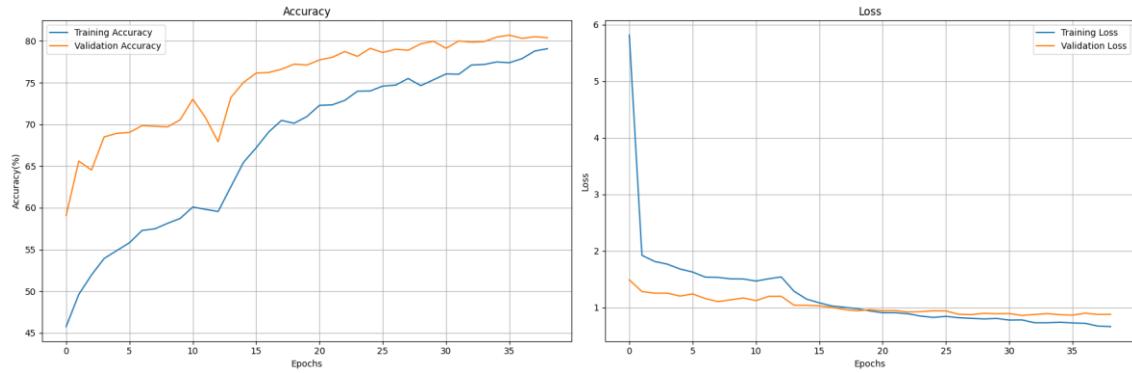


Figura 6.11: Entrenamiento de ResNet152V2 mediante TL con regulador de pesos.

En la figura 6.11 se observa que tanto la precisión como la pérdida de entrenamiento es mucho menor que el de validación, esto se debe a que estamos aplicando técnicas de regulación como dropout, regulación de pesos y pueden influir de forma distinta en el conjunto de entrenamiento y validación.

Finalmente, la pérdida de entrenamiento converge con la de validación y acaba mejorándola y si se hubiera seguido entrenando es posible que la precisión de entrenamiento convergiese con la de validación.

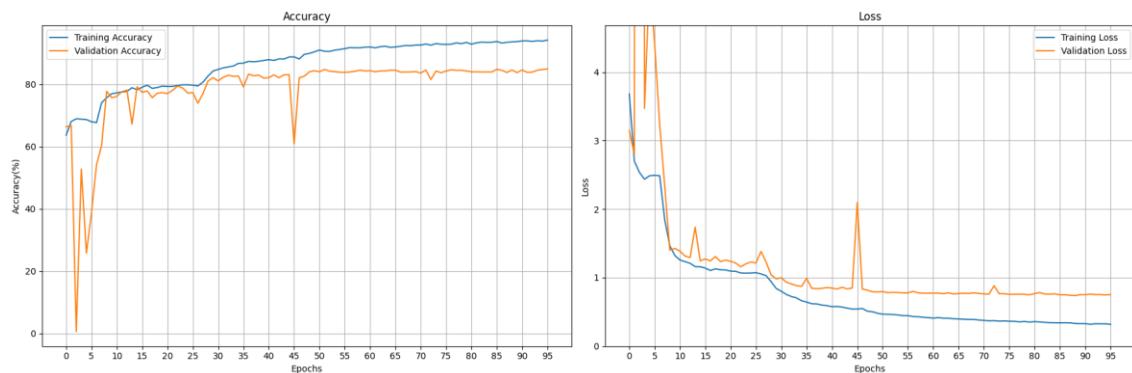


Figura 6.12: Entrenamiento de InceptionV3 mediante TL con regulador.

En la figura 6.12, se muestra que la penalización de pesos en este entrenamiento ha afectado en bajones repentinos de precisión y un aumento muy notable en la pérdida de validación, entre las épocas 3 y 7. A partir de la época 8, comienza a tener resultados más propios de una red neuronal y esto sugiere que el modelo está aprendiendo y generalizando correctamente, y las fluctuaciones en el modelo es algo normal, como en la época 45, pero no es muy frecuente encontrarse con picos tan elevados.

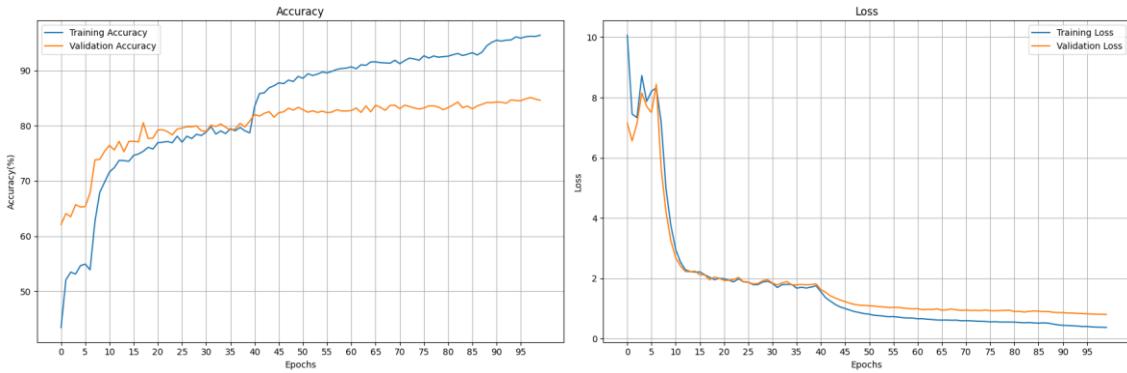


Figura 6.13: Entrenamiento de EfficientNetV2B3 mediante TL con regulador.

En la figura 6.13, se muestra el entrenamiento de la arquitectura EfficientNetV2B3 con regulador de pesos. Inicialmente comienza con bajos valores de precisión y pérdidas bastante altas y desde la época 7 hasta la 12 bajan drásticamente las pérdidas de los conjuntos y aumenta un 10% la precisión tanto de entrenamiento como de validación. Finalmente, tiene otra bajada de pérdida y aumento de precisión en ambos conjuntos, desde la época 38 a la 48 y a partir de esta época comienza a mejorar muy lentamente, sin llegar a sobreentrenar.

En la época 98, comienza a bajar la precisión del conjunto de validación y subir la pérdida, lo que indica que esa época podría ser un punto de inflexión para que ocurra sobreentrenamiento.

En la siguiente tabla 6.9, se muestra el checkpoint de cada arquitectura con los mejores pesos del proceso de entrenamiento.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	36	0.7243	0.7739	0.8619	0.8046
<b>InceptionV3</b>	96	<b>0.3188</b>	0.9418	<b>0.7534</b>	0.8494
<b>EfficientNetV2B3</b>	98	0.3802	<b>0.9619</b>	0.9619	<b>0.8510</b>

Tabla 6.9: Resultados con los mejores pesos de entrenamiento mediante TL con regulador.

Como se puede observar en la tabla 6.10, la arquitectura con mejores resultados empleando los modelos con mejores pesos en esta prueba es EfficientNetV2B3, con una precisión del 87,29.

Nombre	Precisión test (%)
<b>ResNet152V2</b>	84.85
<b>InceptionV3</b>	86.98
<b>EfficientNetV2B3</b>	<b>87.29</b>

Tabla 6.10: Resultado de conjunto de test mediante TL con regulador.

### 6.2.3 Fine-tuning con capas densas y dropout

En este caso, se experimentará empleando TL, pero con fine-tunning de las últimas capas convolucionales, permitiendo una mejor adaptación al dataset propio, incluiremos dos capas fully connected, una con 1024 redes neuronales y otra con el número de clases del dataset para que el modelo tenga la estructura adecuada para poder representar la salida, y una desactivación del 50% de las neuronas.

Al tratarse de un fine-tunning la cantidad de parámetros a entrenar varía respecto a los anteriores entrenamientos, ya que no solo se tiene en cuenta las nuevas capas añadidas, sino las que se han descongelado. A continuación, en la Tabla 6.11 se muestran la cantidad de parámetros descongelados y los parámetros de las nuevas capas de cada arquitectura a entrenar.

Nombre	Parámetros entrenados	Parámetros de la arquitectura	Parámetros de las nuevas capas
<b>ResNet152V2</b>	103,842,839	1,053,696	102,789,143
<b>InceptionV3</b>	52,849,175	391,680	52,457,495
<b>EfficientNetV2B3</b>	77,456,407	357,376	77,099,031

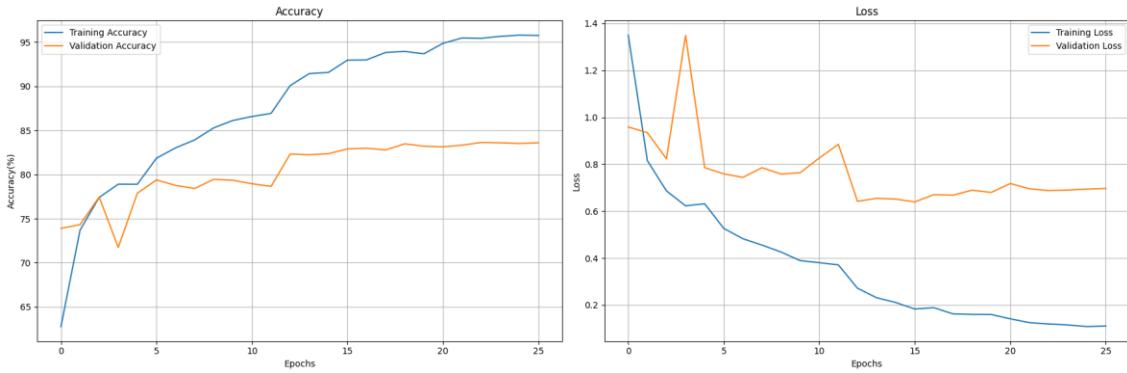
Tabla 6.11: Cantidad total de parámetros a entrenar mediante TL con fine-tuning.

El fine-tune realizado en cada una de las arquitecturas, ha sido descongelar las dos últimas capas que fuesen fully connected ya que descongelar una capa que no tenga parámetros no va a actualizar los pesos, sino que puede mejorar el rendimiento, pero lo que interesa es descongelar capas densas para que el modelo pueda ajustarse mejor a los nuevos datos específicos del dataset propio.

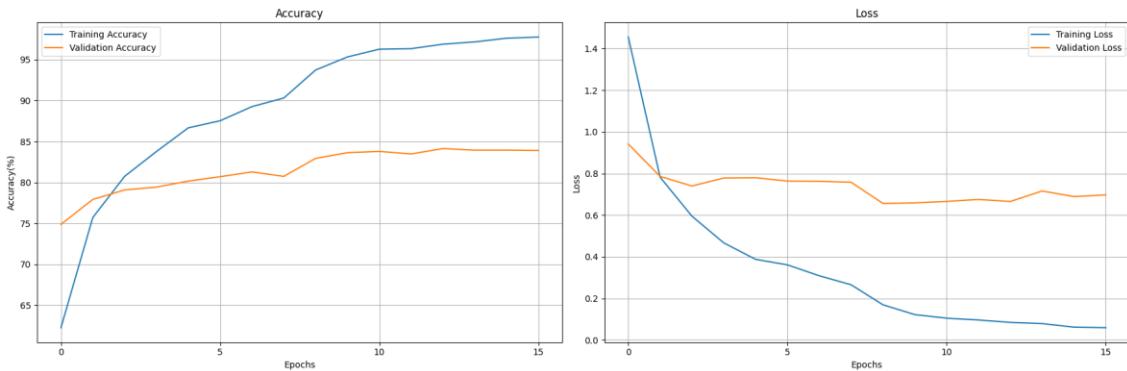
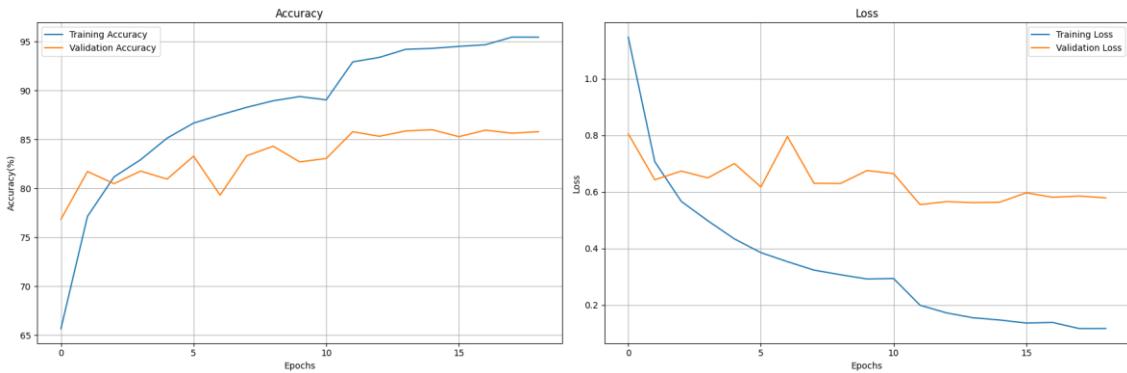
Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	26	0.1105	0.9574	0.6968	0.8359
<b>InceptionV3</b>	19	0.1169	0.9544	<b>0.5787</b>	<b>0.8578</b>
<b>EfficientNetV2B3</b>	16	<b>0.0595</b>	<b>0.9777</b>	0.6972	0.8390

Tabla 6.12: Resultados del entrenamiento con dataset propio mediante TL con fine-tuning.

La tabla 6.12 se muestra como EfficientNet es mejor en la parte de entrenamiento, en perdida y precisión, pero inceptionV3 es mejor en el conjunto de validación, tanto pérdida como precisión. A continuación, se analizarán las gráficas, para dar aclarar la tabla de resultados anterior.



En la figura anterior se puede observar como a partir de la época número 12, como la precisión aumenta lentamente y la pérdida aumenta muy poco, esto se puede deber a diversos factores como el sobreentrenamiento o que el conjunto de validación no tiene más características para que pueda aprender el modelo y se genera un estancamiento, más conocido en DL como **meseta**.



mejora la precisión con casi el mismo valor de pérdida en el conjunto de validación, como se mostrará en la siguiente tabla.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	23	0.1192	0.9542	0.6878	0.8363
<b>InceptionV3</b>	15	0.1470	0.9430	<b>0.5695</b>	<b>0.8598</b>
<b>EfficientNetV2B3</b>	13	<b>0.0850</b>	<b>0.9689</b>	0.6655	0.8413

Tabla 6.13: Resultados con los mejores pesos de entrenamiento mediante TL con fine-tuning.

Con los resultados obtenidos con la tabla 6.13 y 6.14, la arquitectura InceptionV3 es algo mejor que EfficientNetV2B3, siendo superior en el conjunto de validación y de test.

Nombre	Precisión test (%)
<b>ResNet152V2</b>	86.60
<b>InceptionV3</b>	<b>87.61</b>
<b>EfficientNetV2B3</b>	87.51

Tabla 6.14: Resultado de conjunto de test mediante TL con fine-tuning.

Concluidos los resultados para el conjunto de test, la mejor arquitectura es InceptionV3, pero no descartaría EfficientNetV2B3, ya que la curva de aprendizaje no fluctúa tanto en EfficientNetV2B3.

#### 6.2.4 Fine-tuning con Capas densas y dropout con regularización de pesos

En esta prueba se procede a realizar un modelo mediante TL con fine-tuning descongelando las últimas capas densas del modelo base, las mismas que en el apartado anterior 6.2.3 Se ha añadido una capa fully connected de 1024 neuronas con activación ReLU y regulador de pesos. Esta capa incluye un dropout aleatorio del 50% de las neuronas. Finalmente, se añade una capa fully connected adaptada al número de clases a entrenar.

En la tabla 6.15, se muestran los resultados obtenidos tras entrenar empleando TL con fine-tuning y un regulador de pesos.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	65	0.2581	0.9626	0.9079	0.8243
<b>InceptionV3</b>	45	0.1271	0.9788	<b>0.7393</b>	<b>0.8645</b>
<b>EfficientNetV2B3</b>	75	<b>0.1194</b>	<b>0.9954</b>	0.9220	0.8475

Tabla 6.15: Resultados del entrenamiento con dataset propio mediante fine-tuning y regulador de pesos.

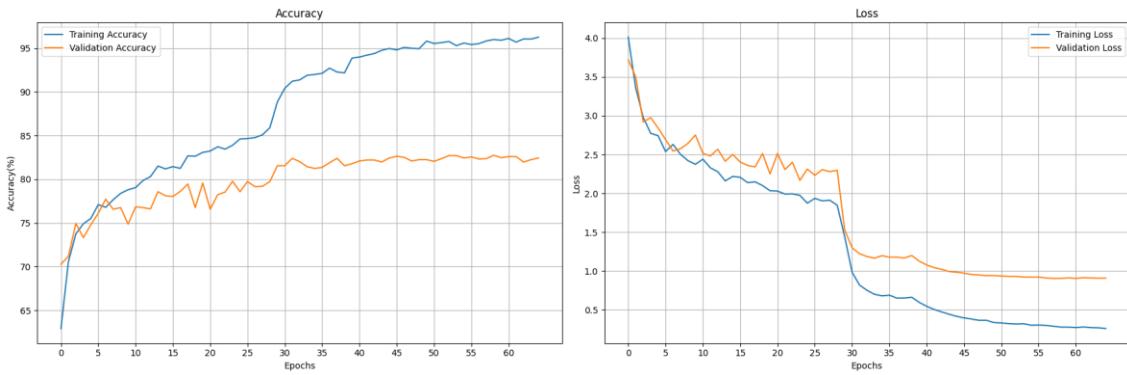


Figura 6.17: Entrenamiento de ResNet152V2 mediante fine-tuning con regulador de pesos.

En la figura 6.17 se observa como el modelo tiene una curva de aprendizaje gradual. Sin embargo, a partir de la época 30, la pérdida mejora muy lentamente y la precisión de entrenamiento aumenta gradualmente. Por otro lado, la precisión de validación aumenta lentamente hasta estancarse y lo que provoca una parada anticipada mediante el callback EarlyStopping con una espera de las 7 épocas.

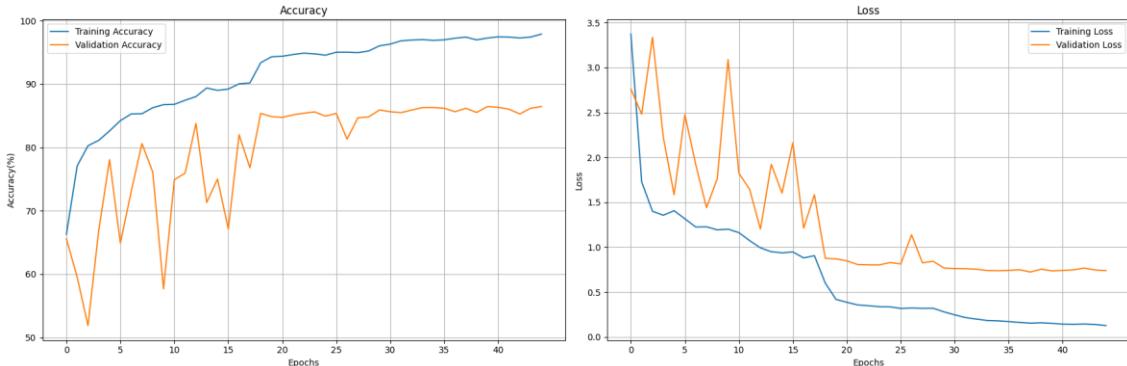


Figura 6.18: Entrenamiento de InceptionV3 mediante fine-tuning con regulador de pesos.

En la figura 6.18, se aprecia que el modelo presenta fluctuaciones muy variables, lo que podría indicar que la configuración de hiperparámetros o la integración del modelo preentrenado con las nuevas capas no está funcionando adecuadamente. Esto impide que el modelo generalice bien en el conjunto de validación, resultando en sobreentrenamiento.

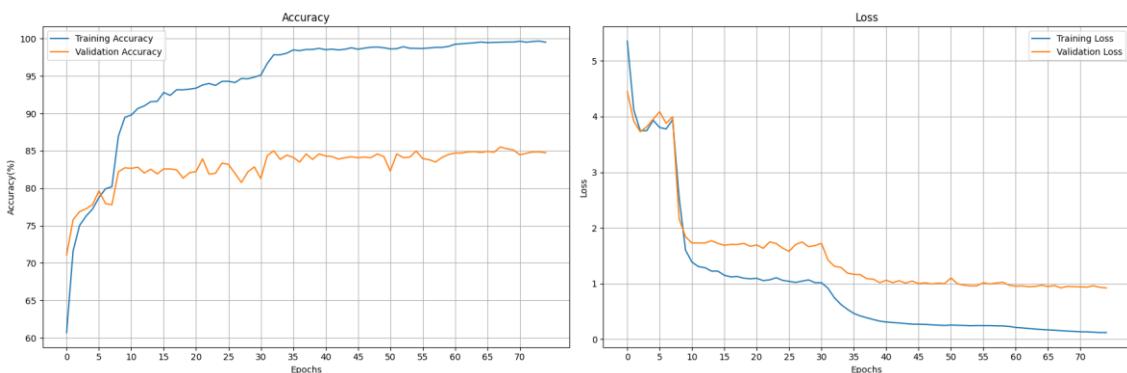


Figura 6.19: Entrenamiento de EfficientNetV2B3 mediante fine-tuning con regulador de pesos.

En la figura 6.19, se observa que la precisión de entrenamiento aumenta rápidamente, estabilizándose luego en una meseta. Esto sugiere que el modelo está aprendiendo bien los datos de entrenamiento. No obstante, la diferencia entre la precisión de entrenamiento y validación

sugiere dos posibles situaciones: sobreentrenamiento o que los datos de validación no son suficientemente representativos para que el modelo aprenda más características. La segunda opción parece más probable, ya que en ninguna de las pruebas se ha superado el 0.86 de precisión en el conjunto de validación.

Respecto al gráfico de pérdida, ambos casos muestran una pérdida inicial alta que luego disminuye. La similitud entre las curvas de pérdida de entrenamiento y validación sugiere un sobreentrenamiento moderado. Esto indica que el modelo está aprendiendo bien los patrones específicos del conjunto de entrenamiento, pero no está generalizando bien al conjunto de validación, o podría haber ruido en los datos de validación.

La siguiente tabla muestra los resultados del entrenamiento almacenados en el último checkpoint.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>ResNet152V2</b>	59	0.2773	0.9598	0.9044	0.8274
<b>InceptionV3</b>	40	<b>0.1514</b>	0.9726	<b>0.7550</b>	<b>0.8552</b>
<b>EfficientNetV2B3</b>	68	0.1527	<b>0.9953</b>	0.9219	<b>0.8552</b>

Tabla 6.16: Resultados con los mejores pesos de entrenamiento mediante fine-tuning con regulador de pesos.

El modelo que ha mostrado el mejor desempeño es EfficientNetV2B3, con una diferencia mínima respecto a InceptionV3.

Nombre	Precisión test (%)
<b>ResNet152V2</b>	85.96
<b>InceptionV3</b>	88.46
<b>EfficientNetV2B3</b>	<b>88.68</b>

Tabla 6.17: Resultado de conjunto de test mediante fine-tuning.

Como se puede observar en las tablas 6.16 y 6.17 respectivamente el mejor modelo, a pesar de presentar sobreentrenamiento en todas las pruebas de entrenamiento en cada arquitectura, es EfficientNetV2B3.

### 6.2.5 Prueba con EfficientNetV2B3 preentrenado

Tras haber realizado pruebas de las arquitecturas con TL y Fine-tuning, se observa que para nuestro dataset emplear un regulador de pesos, mejora los porcentajes de precisión, pero finalmente acaba sobreentrenando y fluctuando demasiado en el entrenamiento.

Respecto a la elección de TL con Fine-tuning o sin Fine-tuning, se puede elegir ambas ya que los resultados tanto de entrenamiento como de test son muy similares, lo único diferente es que varía la precisión y la pérdida, pero no es muy notable. En este proyecto, para la prueba con la mejor arquitectura encontrada, EfficientNetV2B3 con 21 mil clases de Google, se realizará mediante TL.

En este apartado hay que mencionar que se va a añadir una mejora de la arquitectura de EfficientNetV2B3 creada por Google, llamada imagenet21k-b3-feature-vector (Google, 2019), que emplean la arquitectura EfficientNetV2B3 empleada en este proyecto, pero en vez de estar creada con 1000 clases, Google entrena el modelo (como bien dice en su nombre) con 21000 clases y genera vectores de características de imágenes, es decir, que el modelo toma imágenes y las transforma en vectores numéricos que representan las características más importantes de esas imágenes.

#### Transfer learning con fully connected y dropout

La primera prueba que se va a realizar será con una capa fully connected con 1024 neuronas con un dropout del 50% de las neuronas y finalmente otra capa fully connected con el número de clases de nuestro dataset propio. En la siguiente tabla 6.18 se muestran los resultados del entrenamiento con las características anteriores.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>EfficientNetV2B3-21k</b>	21	0.0917	0.9657	0.3087	0.9220

Tabla 6.18: Resultados del entrenamiento de EfficientNetV2B3 mediante TL con capa densa.

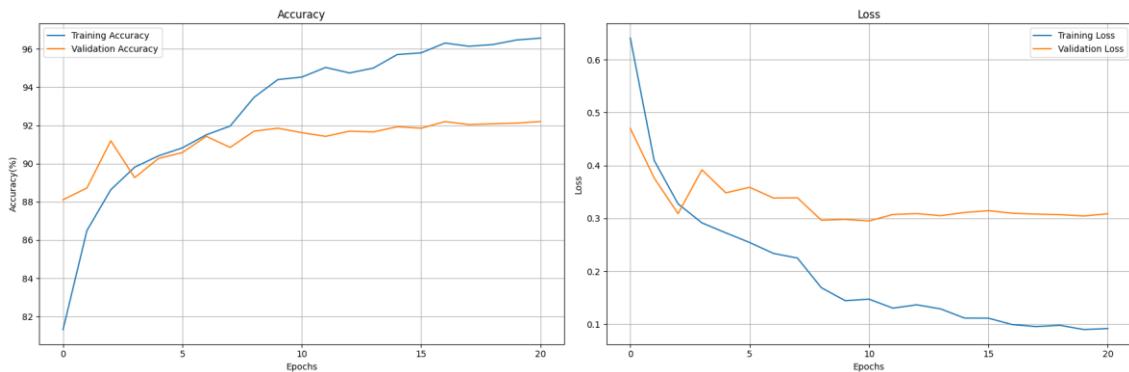


Figura 6.20: Entrenamiento de EfficientNetV2B3-21k mediante TL con capa densa.

En la figura 6.20 se observa la diferencia entre esta arquitectura y las anteriores, empieza con una pérdida y precisión que en algunas pruebas con las arquitecturas anteriores serían el resultado de su última época. Se puede observar que tiene un buen comienzo de entrenamiento y como el conjunto de validación mejora, pero más lentamente que el de entrenamiento. Finalmente, tras una espera de 10 épocas para la mejora del entrenamiento se detiene obteniendo como resultado en el conjunto de test un **93.57%** de precisión.

## Transfer learning con tres capas fully connected con dropout

En esta prueba se realizará con 3 capas fully connected con función de activación ReLU de 512, 256 y 128 neuronas con 50%, 30%, 30% de desactivación de neuronas aleatorias respectivamente. Finalmente, se añade otra capa fully connected con la misma cantidad de neuronas al número de clases del dataset propio.

Los resultados del entrenamiento en esta prueba se muestran en la siguiente tabla 6.19.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>EfficientNetV2B3-21k</b>	27	0.1668	0.9418	0.2754	0.9251

Tabla 6.19: Resultados del entrenamiento de EfficientNetV2B3 mediante TL con capas densas y dropout.

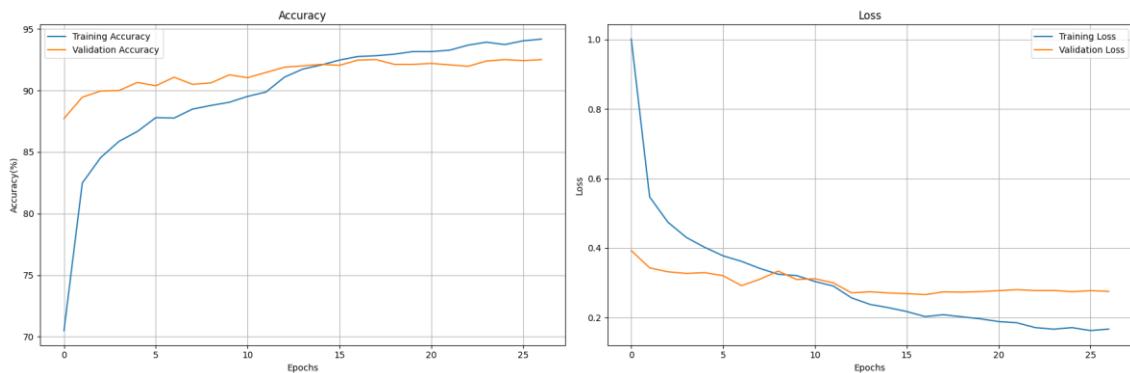


Figura 6.21: Entrenamiento EfficientNetV2B3-21k mediante TL con capas densas y dropout.

Como se observa en la figura 6.21, el modelo deja de mejorar la precisión a partir de la época 18, a partir de ahí baja lentamente la pérdida en el modelo de entrenamiento y la de validación fluctúa muy poco pero no mejora, produciendo lo que se conoce como meseta.

El checkpoint de autoguardado con los mejores pesos está en la época 18, con los datos que se muestran en la siguiente tabla 6.20.

Nombre	Épocas	Pérdida	Precisión	Pérdida de validación	Precisión de validación
<b>EfficientNetV2B3-21k</b>	18	0.2081	0.9283	0.2739	0.9251

Tabla 6.20: Resultado con mejores pesos de entrenamiento de EfficientNetV2B3 mediante TL con capas densas y dropout.

Finalmente, tras lanzar el modelo con el conjunto de test se obtiene una precisión del 94.10% de precisión del modelo. A continuación, se mostrará una matriz de confusión para analizar con qué clases se ha confundido más el modelo.

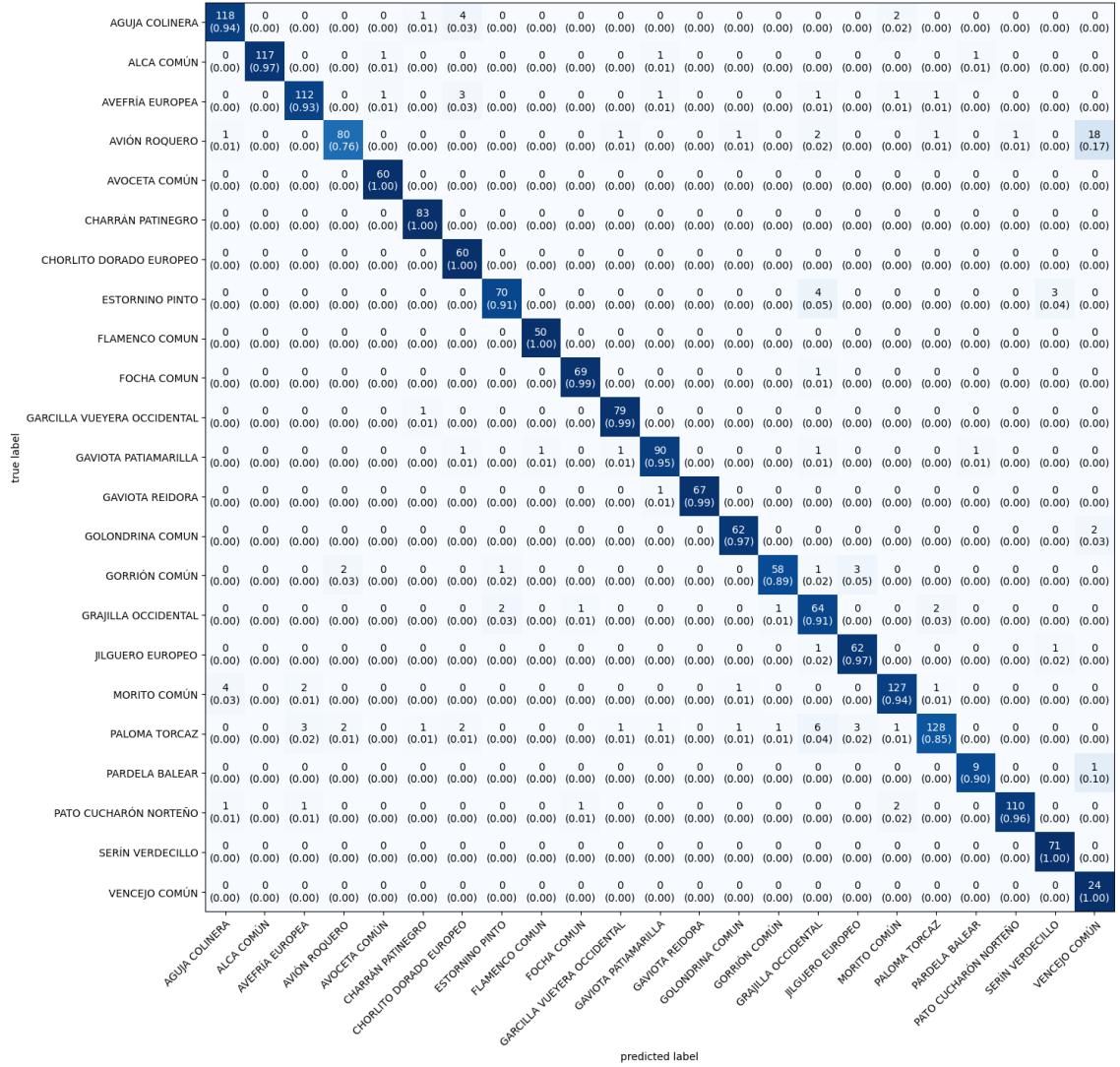


Figura 6.22: Matriz de confusión empleando EfficientNetV2B3 mediante TL con capas densas y dropout

Como se puede observar en la figura 6.22, se muestra una matriz de confusión donde en el eje y se encuentra la clase real y en el eje x se encuentra la predicción del modelo. De esta manera, es posible visualizar con cuáles clases se ha confundido una clase en concreto y el porcentaje de predicción por clase y la global.

El modelo con la clase que peor predicción es Avión roquero confundiéndolo con Vencejo común, siendo el porcentaje de precisión de 76%. Esto se debe a que el avión roquero y el vencejo común son especies muy similares y no solo se han escogido imágenes de alta calidad, ya que se buscan resultados realistas. Además, hay que tener en cuenta que existe un desbalance de imágenes entre estas clases, por tanto, es normal que prediga mal algunas imágenes.

Respecto al porcentaje de acierto que tiene el modelo, es bastante bueno, tratándose de un modelo medianamente simple y considerando que se están evaluando 1881 imágenes entre 23 clases.

## 6.3 Selección del mejor modelo y pruebas con imágenes

En este apartado se realizarán pruebas, con el mejor modelo para cada una de las clases. El mejor modelo es EfficientNetV2B3 ya que alcanza una precisión de validación del 92.51%, significativamente superior a las otras opciones que no superaron el 86%. Se le pasará una imagen al modelo, que no esté incluida en el dataset, e imprimirá la etiqueta real, la etiqueta predicha y la imagen real. Además, se pondrán algunos fallos por similitud de aves, por la lejanía del ave y con más de un ave en la imagen para probar el comportamiento del modelo.

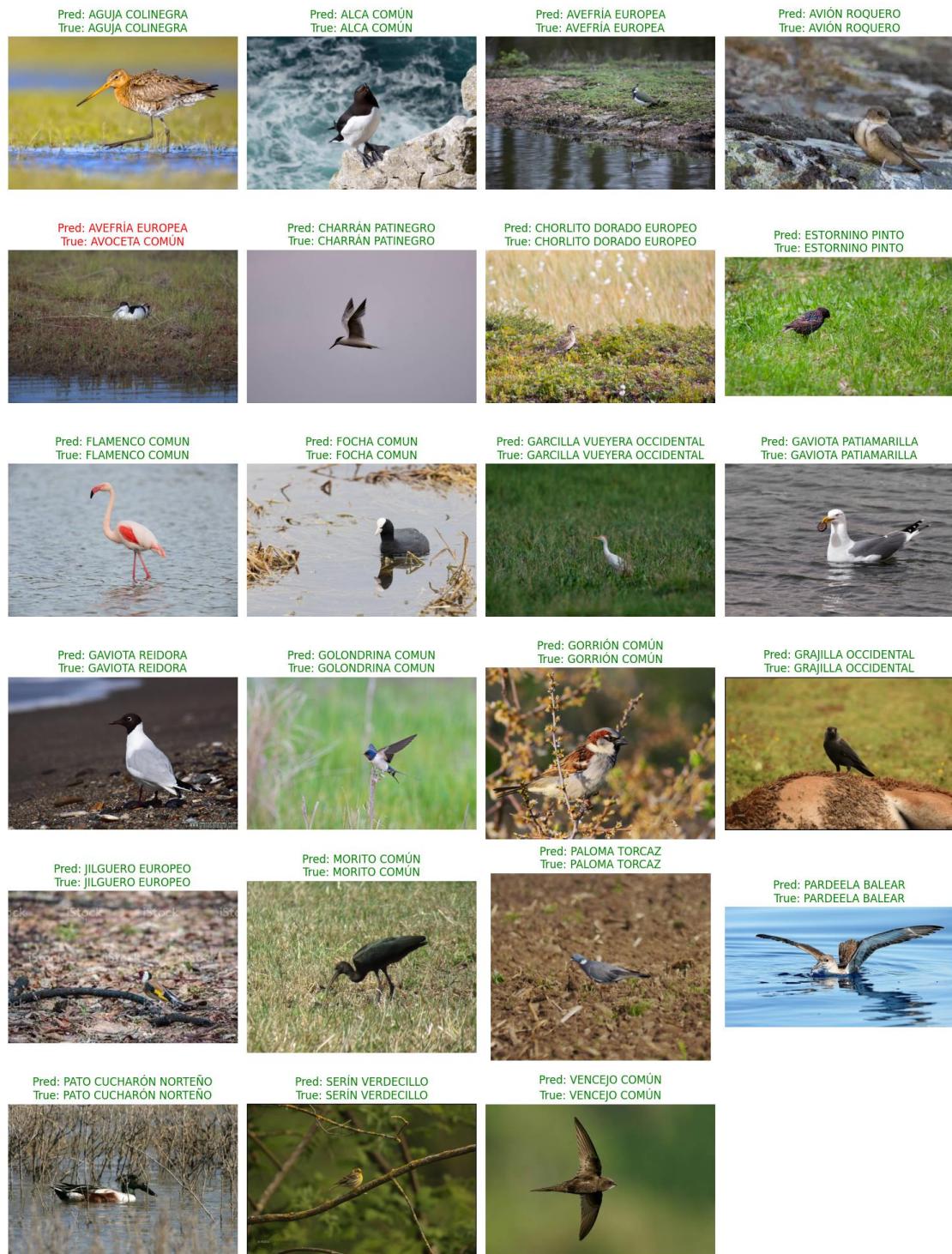


Figura 6.23: Prueba del mejor modelo (EfficientNetV2B3) con imágenes distintas al dataset.

Como se puede observar en la figura 6.23, de las 23 imágenes para predecir solo hay una predicción mal, está en la primera imagen de la segunda fila con el título en color rojo. La clase real de la imagen se trata de una Avoceta común pero la clase predicha es Avefría europea, esto se debe a que en la imagen no se detectan bien las características de la Avoceta común ya que está lejos y la figura es muy parecida a un Avefría europea.

Como se puede observar en la figura 6.24, si se cambia la imagen de la Avoceta común a una con mejor calidad y donde el ave esté más cerca, el modelo es capaz de predecir el ave correctamente.



Figura 6.24: Predicción correcta de Avoceta común.

Una prueba errónea respecto a la postura y lejanía del ave se encuentra una imagen de la clase Jilguero común, que está posada en un peñasco teniendo una postura muy común en aves (incluidas en nuestro dataset) como Golondrina común, Vencejo común o Avión roquero.



Figura 6.25: Predicción errónea de Jilguero europeo en un peñasco.

Se procede a recortar la imagen, para que el ave esté más cerca y el modelo pueda predecir mejor, ya que lo más probable es que la imagen de la figura 6.25 tenga demasiado ruido por el ambiente. Se vuelve a realizar la predicción y se comprueba en la figura 6.26 que la nueva predicción del modelo es correcta.

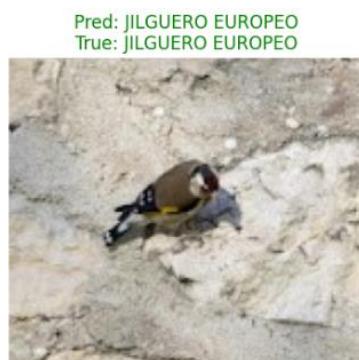


Figura 6.26: Predicción correcta de Jilguero europeo en un peñasco.

Por último, se va a realizar una prueba con una imagen donde aparece más de una especie, en este caso dos, siendo una de ellas es una Gaviota reidora y la otra, es una paloma torcaz.



Figura 6.27: Predicción de Gaviota reidora en imagen con una Gaviota reidora y una Paloma Torcaz.

A continuación, adjuntamos en la figura 6.28 los porcentajes de confianza para cada una de las clases para la figura anterior.

AGUJA COLINEGRA:	0.01%
ALCA COMÚN:	0.00%
AVEFRÍA EUROPEA:	0.82%
AVIÓN ROQUERO:	0.00%
AVOCETA COMÚN:	0.00%
CHARRÁN PATINEGRO:	0.01%
CHORRITO DORADO EUROPEO:	0.00%
ESTORNINO PINTO:	0.02%
FLAMENCO COMUN:	0.07%
FOCHA COMUN:	0.00%
GARCILLA VUEYERA OCCIDENTAL:	0.25%
GAVIOTA PATIAMARILLA:	8.29%
GAVIOTA REIDORA:	89.54%
GOLONDRINA COMUN:	0.00%
GORRÓN COMÚN:	0.00%
GRAJILLA OCCIDENTAL:	0.03%
JILGUERO EUROPEO:	0.05%
MORITO COMÚN:	0.00%
PALOMA TORCAZ:	0.88%
PARDELA BALEAR:	0.01%
PATO CUCHARON NORTEÑO:	0.00%
SERÍN VERDECILLO:	0.01%
VENCEJO COMÚN:	0.00%

Figura 6.28: Porcentajes de confianza de cada una de las clases respecto a la figura anterior.

Como se puede observar en la figura 6.28, detecta por mucha diferencia, con el porcentaje de 89.54, las características de la *Gaviota Reidora*, frente a un porcentaje de 0.88, de la *Paloma Torcaz*.

## 7. Conclusión

En este Trabajo de Fin de Grado (TFG), se ha diseñado un clasificador de aves mediante el uso de técnicas de Deep Learning. Se emplearon redes neuronales convolucionales y arquitecturas avanzadas como EfficientNetV2B3. El modelo entrenado ha logrado una alta precisión en la identificación de aves a partir de imágenes. A lo largo del proyecto, se implementaron múltiples modelos utilizando técnicas como Transfer Learning y fine-tuning, donde EfficientNetV2B3 destacó por su precisión en el reconocimiento visual. Para optimizar el modelo, se exploraron y aplicaron diversas técnicas y funciones, como callbacks, aumento de datos, desactivación de neuronas, ajuste de pesos y reducción de la tasa de aprendizaje.

Se creó un dataset propio que permitió conocer todas las características que debe tener una imagen para que el modelo sea capaz de generalizar y los inconvenientes en ciertas imágenes, que deben de ser eliminados. Este dataset también facilitó la identificación de aves locales en los humedales de la provincia de Alicante. Los experimentos realizados resaltan la importancia de la calidad y el preprocesamiento de las imágenes para mejorar el modelo. Se demostró que al recortar una imagen y reduciendo el ruido ambiental incrementa la exactitud en las predicciones del modelo.

Para futuros desarrollos, se propone aumentar el número de especies y el volumen de imágenes, lo que podría mejorar la precisión del modelo. Además, se sugiere optimizar el modelo e implementar una aplicación multiplataforma para el reconocimiento de imágenes en tiempo real, incluyendo la predicción de comportamientos y posibles amenazas para diferentes especies de aves. Desde un punto de vista académico, este proyecto es significativo para la conservación de la biodiversidad. El sistema desarrollado puede implementarse en dispositivos como Raspberry Pi o Jetson Nano, utilizando YOLO (You Only Look Once) para la detección en tiempo real y de múltiples especies simultáneamente. Esto permitirá un seguimiento automatizado y continuo de las aves y su comportamiento, facilitando la evaluación de posibles amenazas y la implementación de estrategias para su conservación.

En resumen, con este TFG se demuestra el uso efectivo del Deep Learning para el reconocimiento visual de aves, aportando un clasificador tanto para la investigación científica como para la conservación de la biodiversidad.

## Referencias

- Abbas , M. A., Belal, A.-K., & Mazin , M. A. (4 de 1 de 2023). *Incorporating a Novel Dual Transfer Learning Approach for Medical Images*. Obtenido de Incorporating a Novel Dual Transfer Learning Approach for Medical Images: Incorporating a Novel Dual Transfer Learning Approach for Medical Images
- Alfaro, M., Calvo, J., Gallego, A. J., Garrido, C., Ríos, A., & Valero, J. J. (2023). *Deep Learning con Keras y Pytorch*. Madrid: Anaya.
- Anaconda. (26 de 8 de 2019). *Digital Education Resources - Vanderbilt Libraries Digital Lab*. Obtenido de <https://heardlibrary.github.io/digital-scholarship/script/anaconda/>
- Anaconda. (s.f.). *Working with GPU packages*. Obtenido de <https://docs.anaconda.com/free/working-with-conda/packages/gpu-packages/>
- Anna. (24 de 9 de 2023). *EfficientNet V2 Image Classification: 93% accuracy*. Obtenido de <https://www.kaggle.com/code/annafabris/efficientnet-v2-image-classification-93-accuracy#Introduction>
- Aprendeconalf. (12 de 5 de 2022). *La librería Numpy*. Obtenido de <https://aprendeconalf.es/docencia/python/manual/numpy/>
- BBVA. (8 de 11 de 2019). Obtenido de Machine learning ¿qué es y cómo funciona?: <https://www.bbva.com/es/innovacion/machine-learning-que-es-y-como-funciona/>
- CENEAM. (s.f.). *Centro Nacional de Educación Ambiental - CENEAM*. Obtenido de <https://www.miteco.gob.es/es/ceneam/quienes-somos.html>
- Comparini, N. (29 de 7 de 2022). *Redes Neuronales Generativas Adversarias (GANs)*. Obtenido de <https://blog.damavis.com/redes-neuronales-generativas-adversarias-gans/>
- Cornell University. (4 de 9 de 2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Obtenido de <https://arxiv.org/abs/1409.1556>
- Cornell University. (s.f.). *CornellLab Merlin*. Obtenido de <https://merlin.allaboutbirds.org/es/pagina-de-inicio/>
- Cornell University. (s.f.). *eBird*. Obtenido de <https://ebird.org/spain/home>
- Cornell University. (s.f.). *Recursos del Cornell Lab of Ornithology*. Obtenido de <https://www.solucionescosteras.org/recursos-del-cornell-lab-of-ornithology/>
- DataScientest. (16 de 12 de 2021). *Convolutional Neural Network : definición y funcionamiento*. Obtenido de <https://datascientest.com/es/convolutional-neural-network-es>
- DataScientest. (14 de 12 de 2021). *Recurrent Neural Network (RNN): ¿de qué se trata?* Obtenido de <https://datascientest.com/es/recurrent-neural-network-rnn-de-que-se-trata>
- DataScientest. (10 de 08 de 2023). *Inteligencia artificial : definición, historia, usos, peligros*. Obtenido de Inteligencia artificial : definición, historia, usos, peligros:

<https://datascientest.com/es/inteligencia-artificial-definicion#:~:text=La%20historia%20de%20la%20inteligencia%20artificial%20comenz%C3%A9n%20en%201943%20con,creaci%C3%B3n%20de%20una%20red%20neuronal>

DataSmarts. (7 de 1 de 2010). *¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?* Obtenido de <https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/>

Gamco. (s.f.). *Red Neuronal Recurrente*. Obtenido de <https://gamco.es/glosario/red-neuronal-recurrente/>

Geeksforgeeks. (18 de 2 de 2024). *Transfer Learning with Fine-tuning*. Obtenido de <https://www.geeksforgeeks.org/transfer-learning-with-fine-tuning-in-nlp/>

Geeksforgeeks. (10 de 6 de 2024). *What is LSTM – Long Short Term Memory?* Obtenido de <https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/>

Gobierno de España. (4 de 2018). *eBird*. Obtenido de <https://www.miteco.gob.es/es/ceneam/recursos/pag-web/ebird.html>

Gobierno de España. (19 de 04 de 2023). *Plan de Recuperación, Transformación y Resiliencia del Gobierno de España*. Obtenido de <https://planderecuperacion.gob.es/noticias/que-es-inteligencia-artificial-ia-prtr>

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., D. W.-F., Ozair, S., . . . Bengio, Y. (10 de 6 de 2014). *Generative Adversarial Nets*. Obtenido de <https://arxiv.org/pdf/1406.2661>

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Google. (2019). *efficientnet\_v2*. Obtenido de <https://www.kaggle.com/models/google/efficientnet-v2/TensorFlow2/imagenet21k-b3-feature-vector/1>

Google. (25 de 1 de 2024). *Guía avanzada de Inception v3*. Obtenido de <https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=es-419>

Gustavo. (13 de 7 de 2021). *Redes neuronales recurrentes*. Obtenido de <https://kwfoundation.org/blog/2021/07/13/redes-neuronales-recurrentes/>

ICML Conference. (2021). *EfficientNetV2 B0 to B3 and S, M, L*. Obtenido de [https://keras.io/api/applications/efficientnet\\_v2/](https://keras.io/api/applications/efficientnet_v2/)

International Business School. (3 de 4 de 2023). *¿Por qué Python es el mejor lenguaje para programar IA?* Obtenido de <https://eiposgrados.com/blog-python/por-que-python-mejor-lenguaje-para-programar-ia/>

INTERSOG. (18 de 8 de 2023). *AI'S POWERFUL DUO: DIFFERENCE BETWEEN MACHINE LEARNING AND DEEP LEARNING*. Obtenido de <https://intersog.co.il/blog/ais-powerful-duo-difference-between-machine-learning-and-deep-learning/>

- KeepCoding. (s.f.). *Arquitectura VGG16 y VGG19 en Deep Learning*. Obtenido de <https://keepcoding.io/blog/arquitectura-vgg16-vgg19-deep-learning/>
- Keras. (s.f.). *Callbacks API*. Obtenido de <https://keras.io/api/callbacks/>
- Kota, S. D. (17 de 4 de 2020). *Understanding Image Augmentation Using Keras(Tensorflow)*. Obtenido de <https://medium.com/analytics-vidhya/understanding-image-augmentation-using-keras-tensorflow-a6341669d9ca>
- Laguna, N. (11 de 10 de 2023). *Python y la Inteligencia Artificial: La Combinación Perfecta para Impulsar la Innovación Empresarial*. Obtenido de <https://www.linkedin.com/pulse/python-y-la-inteligencia-artificial-combinaci%C3%B3n-perfecta-laguna/>
- Lozada, S. (27 de 4 de 2023). *Redes Neuronales Convolucionales(CNN)*. Obtenido de <https://medium.com/latinixinai/redes-neuronales-convolucionales-cnn-dd4b50ceb24>
- Matplotlib. (s.f.). *Plot types*. Obtenido de [https://matplotlib.org/stable/plot\\_types/index.html#d-and-volumetric-data](https://matplotlib.org/stable/plot_types/index.html#d-and-volumetric-data)
- McCarthy, J. (1962). *LISP 1.5 programmer's manual*. Cambridge.
- NumPy. (s.f.). *NumPy*. Obtenido de <https://numpy.org/>
- NVIDIA. (s.f.). *NVIDIA cuDNN*. Obtenido de <https://developer.nvidia.com/cudnn>
- Oppermann, A. (2022). *Optimization in Deep Learning: AdaGrad, RMSProp, ADAM*. Obtenido de <https://artemoppermann.com/optimization-in-deep-learning-adagrad-rmsprop-adam/>
- Pandas. (2008). *Pandas*. Obtenido de <https://pandas.pydata.org/about/>
- Paperswithcode. (20 de 6 de 2024). *Max Pooling*. Obtenido de <https://paperswithcode.com/method/max-pooling>
- Ruiz, J. (14 de 2 de 2024). *Humedales alicantinos*. Obtenido de <https://vadeaves434152954.wordpress.com/2024/02/14/humedales-alicantinos/>
- scikit-learn. (s.f.). *scikit-learn Machine Learning in Python*. Obtenido de <https://scikit-learn.org/stable/>
- Tan, M., & Le, Q. V. (23 de 6 de 2021). *EfficientNetV2: Smaller Models and Faster Training*. Obtenido de <https://arxiv.org/pdf/2104.00298.pdf>
- TokioSchool. (13 de 3 de 2023). *Tipos de Deep Learning: una guía completa*. Obtenido de <https://www.tokioschool.com/noticias/tipos-deep-learning/>
- Torre, J. d. (18 de 2 de 2023). *REDES GENERATIVAS ADVERSARIAS (GAN)*. Obtenido de <https://arxiv.org/pdf/2302.09346.pdf>
- Torres, J. (22 de 9 de 2019). *Redes Neuronales Recurrentes*. Obtenido de <https://torres.ai/redes-neuronales-recurrentes/>
- Ueda, K.-i., Agrin, N., & Kline, J. (2008). *iNaturalist*. Obtenido de <https://www.inaturalist.org/>

UNIR. (3 de 8 de 2021). *¿Qué son las redes neuronales? Concepto y usos principales*. Obtenido de *¿Qué son las redes neuronales? Concepto y usos principales*: <https://www.unir.net/ingenieria/revista/redes-neuronales-artificiales/>

UNIR. (29 de 3 de 2022). *¿Qué es un sistema experto? Usos y aplicaciones en Inteligencia Artificial*. Obtenido de <https://www.unir.net/ingenieria/revista/sistema-experto/#:~:text=Los%20sistemas%20expertos%20son,un%20profesional%20en%20la%20materia>

Viñals, J. T. (2020). *PYTHON DEEP LEARNING: INTRODUCCIÓN PRÁCTICA CON KERAS Y TENSORFLOW 2*. Barcelona: Marcombo75.

## **Lista de Acrónimos y Abreviaturas**

ANN	Redes Neuronales Artificiales
CNN	Redes Neuronales Convolucionales
CPU	Central Processing Unit
CSV	Valores Separados Por Comas
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network
DL	Deep Learning
DNN	Redes Neuronales Profundas
GAN	Redes Generativas Adversarias
GPU	Graphic Processing Unit
GRU	Unidades Recurrentes Cerradas
IA	Inteligencia Artificial
LSTM	Memoria a largo plazo
ML	Machine Learning
NLP	Procesamiento de Lenguaje Natural
ReLU	Unidad Lineal Rectificada
RNN	Redes Neuronales Recurrentes
TFG	Trabajo de Fin de Grado
TL	Transfer Learning
TPU	Tensor Processing Unit
VC	Visión por computadora