Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Inteligencia Artificial aplicada a la Monitorización y Control de fauna

Alumno: Jaime Álvarez Urueña

Tutora de universidad: Mª Aránzazu Simón Hurtado

Tutor de empresa: Javier Curto Hernández

"No hay nada más gratificante
que ver el fruto de
tu esfuerzo y dedicación."
Fernando Alonso.

Agradecimientos

A mis tutores, Mª Aránzazu Simón Hurtado y Javier Curto Hernández, por vuestra ayuda y atención.

Al Air Institute por brindarme la oportunidad de aprender y trabajar en este proyecto.

A mis padres, mi hermano y mi abuelo por su cariño.

Resumen

La energía eólica es una de las fuentes de energía renovables y limpias que más se están usando en la actualidad para luchar contra el cambio climático, la reducción de gases de efecto invernadero y conseguir la autonomía energética de los distintos países. No obstante, los parques eólicos constituyen un gran peligro para las aves autóctonas, pues corren el riesgo de chocar contra los molinos, así como la pérdida de su hábitat y el desplazamiento del mismo.

Es por ello que es necesario poder identificar y clasificar las aves existentes en un espacio geográfico para cuantificar el riesgo que corren en caso de crear un parque eólico en esa localización. Para ello, se han creado una serie de modelos de inteligencia artificial para poder identificar y clasificar las aves mediante sus imágenes y los sonidos que emiten.

En este Trabajo de Fin de Grado se han desarrallodo 3 algoritmos: uno mediante visión artificial con un modelo de aprendizaje automático supervisado para poder segmentar las aves en una imagen. El segundo algoritmo usa modelos (supervisados) para clasificar las imágenes de las aves detectadas, y el último, también supervisado, se encarga de clasificar las aves según el sonido que emiten.

Abstract

Nowadays, wind energy is one of the most widely used source of renewable and clean energy to combat climate change, helping to reduce greenhouse gases, and achieve energy autonomy for various countries. However, wind farms may pose a significant threat to native birds, as they risk collision with windfarm turbines, losing their habitats, and being displaced.

Therefore, it is necessary to identify and classify the birds present in a geographic area to assess the risk they may face if a wind farm is installed in that location. For this purpose, a solution of artificial intelligence models has been developed to identify and classify birds through the recognition of images and sounds they emit.

In this Bachelor's Thesis, three algorithms have been developed: one using computer vision techniques through a supervised machine learning model to identify and segment the birds within an image. The second algorithm uses models to classify the images of the detected birds' species; and the last one, also supervised, is responsible for classifying the birds according to the sounds they emit.

Índice general

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

Capítulo 1

Introducción

1.1. Contexto y motivación

Los avances tecnológicos realizados en los últimos años requieren de una gran cantidad de energía eléctrica para llevar a cabo sus distintas funciones. Esta gran demanda de energía eléctrica, sumada a las limitaciones de las centrales eléctricas convencionales por su impacto medioambiental, su emisión de gases de efecto invernadero, y el uso de fuentes de energía finitas no renovables, hacen que sea necesario el uso de otras fuentes de energía renovables y respetuosas con el medioambiente. La energía eólica es una de esas fuentes renovables y sostenibles. No obstante, los parques eólicos constituyen un gran peligro para las aves autóctonas de las inmediaciones, pues pueden verse afectadas por los molinos.

Con el objetivo de proteger tanto la vida como el hábitat de las aves, es necesario hacer un estudio previo a la instalación de un parque eólico en una localización, identificando y clasificando las aves de las inmediaciones para evaluar el potencial riesgo que constituye el parque eólico para las aves.

Estas tareas serían muy complicadas llevarlas a cabo con las aproximaciones algorítmicas tradicionales. No obstante, con la revolución que están produciendo las distintas técnicas de aprendizaje automático (especialmente el aprendizaje profundo) que se están desarrollando en los últimos años, junto con las altas capacidades de cómputo, es posible entrenar una serie de modelos de inteligencia artificial para llevar a cabo estas tareas con una mayor tasa de acierto y con mayor eficiencia.

Este proyecto es la continuación de las prácticas curriculares realizadas anteriormente. En dichas prácticas, se entrenó un modelo YOLOv5 y otro YOLOv8 y se hicieron pruebas de distintos modelos de visión artificial aplicados a la clasificación de aves. Los modelos y algoritmos desarrollados en este proyecto son distintos a los desarrollados en las prácticas.

En este Trabajo de fin de Grado (TFG) se van a diseñar e implementar 3 algoritmos distintos para poder identificar y clasificar las aves, dos de ellos trabajarán con las imágenes

y vídeos de las aves, y el otro será el encargado de clasificar las aves por su sonido. Se explicará el preprocesamiento realizado a los datos, así como la justificación de los modelos usados y entrenados. También se incluirán los resultados y las conclusiones obtenidas.

Con el fin de adquirir datos para entrenar el modelo de identificación de aves, el proyecto ha adquirido una cámara de alta calidad (AXIS Q6225-LE PTZ Camera [1]). Esta cámara permite la adquisición de vídeos a una resolución de 1920x1080, con zoom óptico x31. De la misma manera, esta cámara también permite la obtención y grabación del sonido de las aves para su posterior clasificación.

1.2. Objetivos

1.2.1. Objetivo General

Este TFG tiene como objetivo general la implementación de 3 algoritmos, todos basados en aprendizaje automático supervisado, uno para la segmentación de aves en imágenes, otro para la clasificación de las aves mediante su imagen y otro para la clasificación de aves mediante el sonido que producen. Este proyecto y los modelos desarrollados en el mismo tienen como fin ser utilizados en la evaluación del riesgo que constituye para las aves la instalación de un parque eólico.

1.2.2. Objetivos Específicos

Para alcanzar el objetivo general, se abordan los siguientes objetivos específicos:

1. **Obtención y preprocesamiento de datos:** Para entrenar todos los modelos será necesario obtener datos de buena calidad que sean útiles y aporten información relevante para describir el concepto objetivo.

2. **Aprendizaje de tecnologías y frameworks de aprendizaje automático:** Se utilizarán distintas librerías de Python especializadas en el aprendizaje automático, entre las que se encuentran Tensorflow, Keras, Opencv y Pytorch.

3. **Investigación, implementación y elección de modelos para la segmentación de imágenes:** La segmentación de objetos en imágenes es una funcionalidad básica del algoritmo de visión artificial a implementar. Se hará una búsqueda de los modelos más usados y se elegirá e implementará el que mejor satisfaga las necesidades del proyecto.

4. **Investigación, implementación y elección de modelos para la clasificación de imágenes:** Al igual que con la segmentación de objetos, existen muchos modelos que pueden resultar útiles para la clasificación de imágenes. Se estudiarán e implementarán los que mejores resultados ofrezcan.

5. **Investigación de preprocesamiento de ondas de sonido y algoritmos de aprendizaje automático para sonidos:** Será necesario hacer un preprocesamiento muy específico a las ondas de los sonidos de las aves para poder entrenar algoritmos de inteligencia artificial.

6. **Uso conjunto de los modelos desarrollados:** Será necesario hacer un uso conjunto de los modelos y algoritmos desarrollados para obtener resultados óptimos.

1.3. Organización y estructura de la memoria

La memoria de este TFG presenta la siguiente estructura:

Capítulo 1. Introducción: Definición del problema a resolver, su contexto y los objetivos del proyecto.

Capítulo 2. Planificación: Explicación de la planificación realizada en este proyecto, así como la descomposición, duración y definición de las tareas realizadas. También se define la metodología de planificación, y se realiza un análisis de riesgos del proyecto.

Capítulo 3. Estado del arte: Breve estado del arte de la identificación y clasificación de aves.

Capítulo 4. Obtención de datos: Explicación del origen de los datos utilizados en este TFG para el entrenamiento de los distintos modelos entrenados, junto con un resumen ejecutivo de todos los conjuntos de datos.

Capítulo 5. Identificación de aves por imagen: Base teórica de la segmentación de objetos con el modelo YOLOv8, seguido de la explicación de la implementación realizada en este proyecto.

Capítulo 6. Clasificación de aves por imagen: Explicación de los 2 modelos utilizados para la clasificación de imágenes (EfficientNet y Visual Transformers), así como su aplicación en este proyecto.

Capítulo 7. Clasificación de aves por sonido: Explicación teórica de la extracción de características de los sonidos, junto con la implementación desarrollada en este TFG.

Capítulo 8. Conclusiones: Conclusiones obtenidas en el proyecto.

Capítulo 9. Trabajo futuro: Futuro trabajo a realizar para mejorar los resultados del proyecto.

Capítulo 2

Planificación

2.1. Método de planificación

Debido a las características del proyecto, así como los objetivos generales y específicos del mismo, se ha elegido como método de planificación una aproximación basada en prototipos incrementales. Esta aproximación se basa en la implementación de sucesivos prototipos de modo que la implementación de un prototipo esté basado en todos los anteriores. El intervalo de tiempo escogido para desarrollar un nuevo prototipo mejorando e implementando nuevas características será aproximadamente de 2-3 semanas. El tiempo aproximado que se dedicará al proyecto será de 40 horas semanales empezando la semana del 25/03/2024.

Debido a que el proyecto es extenso con 3 dominios distintos de trabajo bien diferenciados (segmentación de objetos en imágenes, clasificación de imágenes y clasificación de sonido) en cada uno de esos períodos de aproximadamente 2-3 semanas se implementará un prototipo perteneciente a uno de esos dominios. Como el desarrollo del proyecto empieza el 25/03/2024, inicialmente se estima que se podrán desarrollar 5-6 prototipos incrementales. Los primeros estarán relacionados con la segmentación de las aves en las imágenes, a continuación se desarrollarán los relacionados con la clasificación de imágenes, y por último los de clasificación de sonido.

2.2. Descripción y duración de las tareas

Para garantizar que se consiguen todos los objetivos del proyecto, dentro de cada período de desarrollo de cada prototipo se ha descompuesto y definido cada una de las tareas y actividades. Las actividades desarrolladas son tareas relacionadas con el estudio y documentación (funcionamiento de modelos, entornos y bibliotecas de programación, etc.), desarrollo de tareas específicas de implementación del prototipo, y redacción de TFG.

Al inicio del proyecto se ha hecho una predicción del tiempo requerido por cada tarea dependiendo de su complejidad, conocimientos previos y relevancia en el proyecto. El tiempo predicho de estas actividades es una aproximación inicial. No obstante, el tiempo empleado en cada una de ellas podrá ser modificado según las necesidades del proyecto. Dichas descripciones, así como sus duraciones predichas y sus duraciones reales se pueden ver en las Tablas 2.1 - 2.6.

Tabla 2.1: Resumen tareas del primer prototipo.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Estudio modelo YOLOv5	25/03/24	26/03/24	25/03/24	01/04/24	16	40
Estudio modelo YOLOv8	28/03/24	29/03/24	02/04/24	02/04/24	16	8
Segmentación de imágenes	01/04/24	01/04/24	03/04/24	05/04/24	8	24
Creación de fichero .yaml	02/04/24	02/04/24	08/04/24	08/04/24	8	8
Selección de parámetros y entrenamiento	03/04/24	04/04/24	09/04/24	10/04/24	16	10
Total de horas					64	90

Tabla 2.2: Resumen tareas del segundo prototipo.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Procesamiento de vídeos	05/04/24	08/04/24	11/04/24	11/04/24	16	2
Representación de predicciones	09/04/24	10/04/24	12/04/24	15/04/24	16	16
Obtención de trayectorias	11/04/24	12/04/24	16/04/24	19/04/24	16	32
Estudio de parámetros óptimos	15/04/24	15/04/24	22/04/24	22/04/24	8	4
Recorte de predicciones	16/04/24	16/04/24	23/04/24	23/04/24	8	2
Total de horas					64	56

Tabla 2.3: Resumen tareas del tercer prototipo.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Obtención de las direcciones	17/04/24	18/04/24	24/04/24	26/04/24	16	24
Actualización heurística IOU	19/04/24	22/04/24	29/04/24	02/05/24	16	32
Imagen con trayectorias	23/04/24	23/04/24	03/05/24	03/05/24	8	8
Optimización de parámetros	24/04/24	25/04/24	06/04/24	07/05/24	8	12
Total de horas					48	76

Tabla 2.4: Resumen tareas del cuarto prototipo.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Estudio de las EfficientNet	26/04/24	29/04/24	08/05/24	08/05/24	16	6
Estudio redes densas	30/04/24	30/04/24	08/05/24	08/05/24	2	1
Estudio redes residuales	30/04/24	30/04/24	08/05/24	08/05/24	2	1
Creación entorno python	02/05/24	02/05/24	09/05/24	10/05/24	4	16
Preprocesamiento de datos	03/05/24	06/05/24	13/05/24	15/05/24	16	24
Entrenamiento	07/04/24	07/04/24	16/04/24	17/05/24	8	16
Discusión de resultados	08/05/24	08/05/24	20/05/24	20/05/24	8	8
Total de horas					56	72

Tabla 2.5: Resumen tareas del quinto prototipo.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Estudio de los Transformer	09/05/24	09/05/24	21/05/24	21/05/24	8	8
Estudio de los ViT	10/05/24	10/05/24	22/05/24	22/05/24	8	8
Implementación de los ViT	13/05/24	14/05/24	23/04/24	27/04/24	16	24
Entrenamiento	15/04/24	15/04/24	28/04/24	28/05/24	8	8
Discusión de resultados	16/05/24	16/05/24	29/05/24	29/05/24	8	8
Total de horas					48	56

Tabla 2.6: Resumen tareas del sexto prototipo.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Estudio de features de sonido	17/05/24	20/05/24	30/05/24	03/06/24	16	24
Preprocesamiento de datos	21/05/24	22/05/24	04/06/24	06/06/24	8	16
Entrenamiento de modelos	23/05/24	24/05/24	07/06/24	07/06/24	16	8
Discusión de resultados	27/05/24	27/05/24	10/06/24	10/06/24	8	8
Total de horas					48	56

Aparte de las tareas mencionadas anteriormente, se han realizado otras tareas correspondientes al desarrollo de la memoria del TFG. Estas han sido desarrolladas a lo largo de todo el proyecto, desde su inicio hasta su fin. En la Tabla 2.7 se ven los tiempos específicos de cada tarea relacionada con la redacción de la memoria del TFG.

El número total de horas predichas fue de 448, y el número total de horas de trabajo ha sido de 542. La diferencia de horas predichas de las reales se debe a las dificultades encontradas a lo largo del desarrollo del proyecto que se comentarán en los capítulos posteriores. La gran cantidad de horas de trabajo se debe a la compleja naturaleza del proyecto, así como su extenso dominio (segmentación de objetos en imágenes, clasificación de imágenes y clasificación de sonidos). Además, se utilizan una gran cantidad de modelos de inteligencia artificial (YOLO, EfficientNet, Visual Transformers) en los que se ha tenido que invertir muchas horas de estudio para optimizar sus resultados.

Tabla 2.7: Resumen tareas de la redacción del TFG.

Tarea	Inicio Estimado	Fin Estimado	Inicio Real	Fin Real	Horas Estimadas	Horas Reales
Redacción de la introducción	27/03/24	28/03/24	27/03/24	29/03/24	16	24
Redacción de la planificación	1/04/24	2/04/24	10/04/24	12/04/24	16	24
Redacción del estado del arte	7/04/24	8/04/24	20/04/24	20/04/24	16	8
Redacción de la segmentación de imágenes	26/04/24	27/04/24	11/05/24	13/05/24	16	24
Redacción de la clasificación de imágenes	28/05/24	29/05/24	31/05/24	31/05/24	16	16
Redacción de la clasificación de sonidos	28/05/24	28/05/24	11/06/24	12/06/24	8	16
Redacción de las conclusiones	11/04/24	12/04/24	13/06/24	13/06/24	8	4
Redacción del trabajo futuro	29/05/24	29/05/24	13/06/24	13/06/24	8	4
Correcciones finales	30/05/24	31/05/24	14/06/24	15/06/24	16	16
Total de horas					120	136

2.3. Descripción de prototipos

En esta sección se va a realizar una descripción breve de cada uno de los prototipos desarrollados incrementalmente, así como las dificultades encontradas en cada una de las tareas y las posibles causas de algún retraso en las mismas.

2.3.1. Primer prototipo

El primer prototipo tiene como finalidad entrenar el modelo YOLOv8 para, en siguientes prototipos, poder segmentar aves en imágenes. En las prácticas se entrenó un modelo YOLOv5 y otro YOLOv8, no obstante, se decidió reentrenar un modelo YOLOv8 con nuevos hiperparámetros y nuevas imágenes para obtener mejores resultados. El modelo YOLOv8 ofrece una mayor versatilidad y menos restricciones que la versión 5, y de ahí su elección para este proyecto.

Para poder entrenar el algoritmo fue necesaria la selección de imágenes que contuvieran un número arbitrario de aves. Los algoritmos de la familia YOLO para ser entrenados requieren, además de las imágenes, los cuadros delimitadores de cada ave en cada una de esas imágenes. Para ello se utilizó la herramienta Make Sense [2] que permite hacer esa segmentación de objetos en las imágenes de una manera rápida y sencilla, obteniendo los resultados en ficheros con extensión *.txt*.

Para entrenar el modelo (se utilizó el modelo desarrollado por Ultralytics [3]), es necesario crear un fichero con extensión *.yaml* donde se definen los hiperparámetros del modelo, (rutas con las imágenes de entrenamiento-prueba-validación y número de clases).

La última tarea para completar este prototipo fue entrenar el algoritmo y hacer una

selección óptima de los parámetros del mismo.

El tiempo estimado es inferior al tiempo real ya que el estudio exhaustivo de los modelos fue una tarea compleja. Además, la segmentación de las aves fue una tarea que llevó mucho tiempo a pesar de su aparente sencillez.

2.3.2. Segundo prototipo

Con el segundo prototipo se consigue poder utilizar el modelo entrenado en el prototipo 1. También se implementa la funcionalidad para obtener los recortes de las aves detectadas en vídeos.

Se estudiaron bibliotecas de Python para el procesamiento de vídeo como Opencv y Numpy. A continuación se ideó un algoritmo para detectar las trayectorias de las aves en sucesivos frames de un vídeo, así como para poder mostrar esas trayectorias como salida. También se ajustaron los parámetros del algoritmo creado para obtener unos resultados óptimos.

El tiempo estimado es aproximadamente el mismo que el tiempo real, aunque la tarea de la obtención de trayectorias requirió más tiempo del predicho inicialmente, principalmente por su complejidad. Por otro lado, el resto de tareas resultaron más sencillas de lo estimado inicialmente.

2.3.3. Tercer prototipo

En este prototipo se creó un nuevo algoritmo para solucionar ciertos problemas encontrados en el prototipo anterior sobre la detección de las trayectorias seguidas por las aves. Este nuevo algoritmo hace uso de los ángulos de las trayectorias, aunque como no se obtuvieron resultados mejores que en el prototipo anterior, se descartó el algoritmo. Por ello, se trabajó de nuevo en el algoritmo del prototipo anterior modificando las heurísticas establecidas en este, obteniendo mejores resultados.

También se implementó la funcionalidad de obtener una imagen con todas las trayectorias de las aves presentes en la imagen, y de nuevo hubo que hacer un ajuste de los parámetros del algoritmo.

De nuevo hay una pequeña discrepancia entre los tiempos estimados y el tiempo real, principalmente por la modificación del algoritmo que no funcionó.

2.3.4. Cuarto prototipo

Con respecto a los objetivos de clasificación de aves, inicialmente se propusieron tanto las redes EfficientNet como los ViT (Visual Transformers) como algoritmos de clasificación.

En este prototipo se hizo uso de las EfficientNet, estudiando y documentando tanto este tipo de redes como otros tipos de redes utilizadas por este modelo (redes densas y residuales, también usadas en YOLOv8). En las prácticas se hizo uso de este modelo también. No obstante se utilizó un dataset distinto. Además, para la realización de este TFG se modificaron las últimas capas de las EfficientNet para adaptarlas a los datos con los que se han trabajado en este TFG.

Para la implementación, se pudo entrenar el modelo en un dispositivo con GPU. Para ello fue necesario crear y configurar un entorno de ejecución. El preprocesamiento realizado en los datos para realizar un entrenamiento óptimo se comentará en posteriores capítulos.

La tarea más complicada de este prototipo fue el preprocesamiento de los datos para que el modelo los pudiera procesar correctamente, y es la principal razón del pequeño retraso.

2.3.5. Quinto prototipo

Este prototipo tiene como fin implementar los ViT, discutir sus resultados, y posteriormente elegir cuál de los dos modelos usados para clasificación de aves obtiene mejores resultados, los ViT de este prototipo o las EfficientNet del prototipo anterior. Cabe destacar que aunque en las prácticas ya se trabajó sobre este modelo, se han realizado una serie de modificaciones sustanciales en la capa del encoder y en la salida del mismo, siendo evidentemente distinto al ViT de las prácticas. Además, en este TFG se ha utilizado un conjunto de datos distinto al de las prácticas.

Para conseguir estos objetivos, hubo una etapa de estudio tanto de los Transformers como de su modificación para el procesado de imágenes. Posteriormente se implementaron en Keras y se entrenaron con el mismo dataset utilizado en el prototipo anterior. Todo el trabajo realizado en la parte de preprocesamiento de datos y la creación del entorno python en el prototipo anterior se pudo reutilizar en este prototipo.

Los tiempos reales en este prototipo son aproximadamente los predichos, aunque la implementación de este modelo requirió más tiempo del pensado inicialmente.

2.3.6. Sexto prototipo

El sonido es el último de los objetivos restantes. Por ello, este sexto prototipo trata de la obtención de datasets con sonidos de aves, su preprocesamiento y su posterior uso en algoritmos de aprendizaje automático supervisado.

Hubo de nuevo una etapa muy extensa que trató del estudio del sonido, su correcto tratamiento y la obtención de características del mismo. Esta fue la tarea más compleja y que más tiempo requirió debido a los escasos conocimientos iniciales de este dominio. Por otra parte, como los modelos usados son los mismos que en los del prototipo 4 (Sección 2.3.4) y prototipo 5 (Sección 2.3.5) se mitigaron los retrasos de las otras tareas.

2.4. Riesgos

Para la realización del análisis de riesgos se ha seguido uno de los estándares más comunes para el análisis de riesgos de proyectos, el PMBOK [4]. Formalmente, un riesgo se define como un suceso con ciertas probabilidades de ocurrir con posibles consecuencias negativas o positivas para el proyecto. Debido a esto, para hacer un correcto análisis de riesgos, es necesario por cada uno de ellos tener en cuenta la probabilidad de que se materialice, así como la severidad de las consecuencias. Es por ello, que se definen 3 posibles probabilidades de que ocurra un riesgo (alta, media, baja) y 3 posibles impactos del riesgo (alto, medio, bajo). La combinación de probabilidad-impacto da lugar al grado del riesgo. En la Tabla 2.8 se definen las 9 posibles combinaciones probabilidad-impacto con su grado de riesgo correspondiente.

Tabla 2.8: Nivel de riesgo en función de la tupla probabilidad-impacto.

Probab \ **Impacto**	Bajo	Medio	Alto
Baja	Bajo	Bajo	Medio
Media	Medio	Medio	Alto
Alto	Medio	Alto	Alto

2.4.1. Riesgos del proyecto

Dependiendo del grado de riesgo de cada riesgo será necesario implementar contramedidas para bien reducir la probabilidad de materialización del riesgo, o reducir su impacto. Los riesgos también se pueden agrupar en distintas categorías dependiendo del dominio en el que afecten. Estos dominios dependen del proyecto, en este caso se considerarán los riesgos que afecten a los dominios: hardware, software, personal, datos y seguridad. En las posteriores Tablas 2.9 - 2.16 se definen los riesgos del proyecto, junto con una pequeña descripción y sus contramedidas.

Tabla 2.9: Riesgo HW.1. Rotura de GPU.

Código de Riesgo	HW.1
Nombre del Riesgo	Fallo en dispositivos hardware.
Categoría	Hardware
Probabilidad	Baja
Impacto	Medio
Mitigación	Monitorizar los dispositivos, siguiendo todas las medidas de seguridad definidas por el fabricante, haciendo un uso correcto de los dispositivos.
Grado de riesgo	Bajo

Tabla 2.10: Riesgo HW.2. Rotura de Cámara.

Código de Riesgo	HW.2
Nombre del Riesgo	Rotura de Cámara.
Categoría	Hardware
Probabilidad	Baja
Impacto	Alto
Mitigación	Colocar la cámara en lugares seguros y seguir todas las indicaciones del fabricante.
Grado de riesgo	Medio

Tabla 2.11: Riesgo D.1. Datos erróneos/insuficientes.

Código de Riesgo	D.1
Nombre del Riesgo	Datos erróneos/insuficientes.
Categoría	Datos
Probabilidad	Baja
Impacto	Alto
Mitigación	Utilizar técnicas de minería de datos para limpiar los datos y hacer el preprocesamiento óptimo.
Grado de riesgo	Medio

Tabla 2.12: Riesgo D.2. Sesgo en los datos.

Código de Riesgo	D.2
Nombre del Riesgo	Bias y sesgo en los datos.
Categoría	Datos
Probabilidad	Baja
Impacto	Alto
Mitigación	Utilizar todas las herramientas disponibles para evitar el sesgo, y hacer disjuntos los conjuntos de datos para test/entrenar/validar.
Grado de riesgo	Medio

Tabla 2.13: Riesgo SE.1. Seguridad de los sistemas.

Código de Riesgo	SE.1
Nombre del Riesgo	Seguridad de los sistemas.
Categoría	Seguridad
Probabilidad	Baja
Impacto	Alto
Mitigación	Hacer uso de cortafuegos, tener actualizados todos los sistemas y evitar direcciones web sospechosas.
Grado de riesgo	Medio

Tabla 2.14: Riesgo SO.1. Uso incorrecto de herramientas Software.

Código de Riesgo	SO.1
Nombre del Riesgo	Utilización incorrecta de las herramientas Software.
Categoría	Software
Probabilidad	Baja
Impacto	Medio
Mitigación	Usar la documentación oficial de los productos usados y estudio previo de los mismos para su uso óptimo.
Grado de riesgo	Bajo

Tabla 2.15: Riesgo P.1. Enfermedad del personal.

Código de Riesgo	P.1
Nombre del Riesgo	Enfermedad del personal.
Categoría	Personal
Probabilidad	Baja
Impacto	Medio
Mitigación	Evitar situaciones peligrosas para el personal que desarrolla el proyecto para evitar retrasos.
Grado de riesgo	Bajo

Tabla 2.16: Riesgo D.3. Datos no relevantes.

Código de Riesgo	D.3
Nombre del Riesgo	Datos no relevantes.
Categoría	Datos
Probabilidad	Media
Impacto	Alto
Mitigación	Utilizar los mejores modelos para la extracción de información de los datos.
Grado de riesgo	Alto

El riesgo que se ha materializado en este proyecto ha sido el D.3, en el dominio de clasificación de imágenes, debido a la baja resolución de estas. Se ha mitigado al máximo posible usando técnicas de aumentación de datos, así como uso de distintos modelos y algoritmos. La materialización de este riesgo no ha supuesto ningún retraso, únicamente ha afectado a los resultados de los modelos.

Capítulo 3

Estado del arte

En este capítulo se procede a presentar las principales técnicas y métodos existentes hasta Junio de 2024 para identificar y clasificar aves. Principalmente hay 3 aproximaciones: radares, imágenes y sonidos. Las dos últimas han experimentado cambios importantes debido a las técnicas de aprendizaje automático desarrolladas en los últimos años.

3.1. Clasificación de aves

3.1.1. Sistemas de radares

El funcionamiento base de este método consiste en la instalación de radares metereológicos y radares de vigilancia en aviones que vuelen a bajas altitudes (no más de 2 km) para detectar las aves. En el artículo [5] se explica que la razón de volar a baja altitud se debe a que la probabilidad de avistar aves por encima de los 2 km de altitud es realmente baja. Aquí se encuentra la principal desventaja de este método, y es que se necesitaría una gran cantidad de aviones volando a bajas altitudes para poder detectar aves, siendo un gasto económico enorme. Además, los aviones comerciales no vuelan a esas altitudes (únicamente al despegar y aterrizar), así que habría que usar aviones de carácter militar. Un artículo que explica más explícitamente el uso del radar para la clasificación de aves es [6].

3.1.2. Funcionamiento del radar

Otro artículo en el que también se definen los radares para la clasificación de aves haciendo una explicación del funcionamiento del radar es [7]. El radar es una tecnología que se desarrolló en los años 30 del siglo pasado, y desde entonces se ha ido optimizando y actualizando. Su funcionamiento base consiste en emitir pulsos de ondas electromagnéticas en una dirección. Las ondas al llegar al objeto (aves en este caso) rebotan y se detectan con

una antena. Dependiendo de distintas condiciones físicas del objeto en el que la onda ha rebotado (forma, tamaño, velocidad, etc.) la onda rebotada tendrá unas características u otras. Aunque con esta tecnología teóricamente pueda ser posible reconocer y clasificar aves, las restricciones para llevar a cabo los experimentos son demasiado exigentes.

3.1.3. Redes convolucionales

Las redes convolucionales (CNN) son un subconjunto de las redes neuronales artificiales (ANN). Este tipo de redes realizan una serie de operaciones (convoluciones) que resultan de gran utilidad para el procesamiento de imágenes, pues son capaces de procesar la información espacial que tiene cada uno de los píxeles. El artículo [8] hace una explicación detallada del funcionamiento de estas redes a nivel teórico, sin aplicarlo a las aves. Otros autores también han escrito artículos con conclusiones y métodos similares [9]. Otras operaciones que realizan estas redes son operaciones de pooling, explicadas en el artículo [10], y funciones de activación, definidas en el artículo [11].

La idea principal subyacente tras estas redes consiste en crear una tubería de procesamiento de capas convolucionales en serie, de forma que las salidas (llamadas características) de la capa *n* sean la entrada de la capa *n+1*. Esta tubería de procesamiento se encarga de hacer el proceso llamado *extracción de características*, que es el proceso en el cual se extraen características específicas de la imagen. Posteriormente esas características se hacen pasar por una red *fully connected* para clasificar la imagen. En la Figura 3.1 se describe un ejemplo de una red convolucional con arquitectura VGG16, la arquitectura CNN más sencilla.

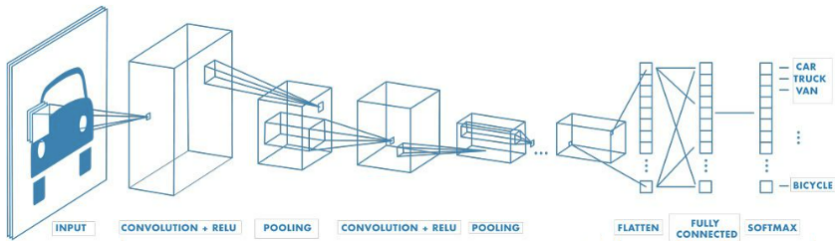


Figura 3.1: Ejemplo de funcionamiento de una CNN. [12]

No obstante, a pesar de que las CNN son capaces de extraer la información espacial de cada píxel, presentan algunas destacables desventajas, como la alta capacidad de cómputo necesaria para entrenarlas y la gran cantidad de imágenes necesarias para poder generalizar el concepto objetivo. Con fin de paliar estos problemas se desarrollaron otros tipos de redes como las redes residuales (eliminan el problema de la degradación del gradiente y facilita el aprendizaje) [13] o redes densas (combinan características de distintas capas facilitando el aprendizaje [14]). Técnicas de aumentación de datos para obtener un mayor número de imágenes significativas que aporten información para generalizar el concepto también son técnicas muy comúnmente utilizadas en este ámbito [15]. En el artículo [16] se hace un resumen de las desventajas de las CNN, así como de posibles soluciones. La CNN que a fecha de Junio de 2024 mejores resultados está ofreciendo en dominios de distinta índole,

incluyendo la clasificación de aves, son las EfficientNet [17]. Es un tipo de redes en las que se ha conseguido disminuir drásticamente el número de parámetros necesarios a aprender, siendo muy eficientes. También resuelve problemas muy recurrentes en todo el campo del aprendizaje profundo, como el problema de la degradación, presente en todos los modelos definidos hasta la fecha. El artículo en el que se presentó por primera vez las EfficientNet es [17], definiendo los problemas que resuelve y las ventajas que presenta. Este tipo de redes, sus ventajas y situaciones bajo las que se usan se definirán en el Capítulo 6.

3.1.4. Visual Transformers

Los Visual Transformers son una modificación de los transformers desarrollados en 2017 [18]. Los transformers inicialmente fueron desarrollados para ser utilizados en el ámbito de las series temporales. Las series temporales son un tipo de datos en el que el orden de los datos de la secuencia es relevante, y cada dato en esa secuencia se procesa uno tras otro (secuencialmente). Este tipo de redes fueron una gran revolución y son las más usadas en campos como el procesamiento de lenguaje natural o IA generativa. La definición de los transformers, la revolución de los mecanismos de Auto-Atención, su estructura e implementación se encuentran en el artículo [18]. La característica principal de las redes recurrentes es que la salida del dato n de una secuencia es la entrada de la red junto con el dato *n+1*. En la Figura 3.2 se ve una retropropagación de una neurona recurrente en el tiempo, es decir, la neurona es la misma en distintos instantes de tiempo. Las entradas son $X_{t-3}, X_{t-2}, X_{t-1}$ y X_t, mientras que las salidas son $y_{t-3}, y_{t-2}, y_{t-1}$, y_t. Se puede apreciar que la salida de un instante cualquiera es la entrada del instante siguiente.

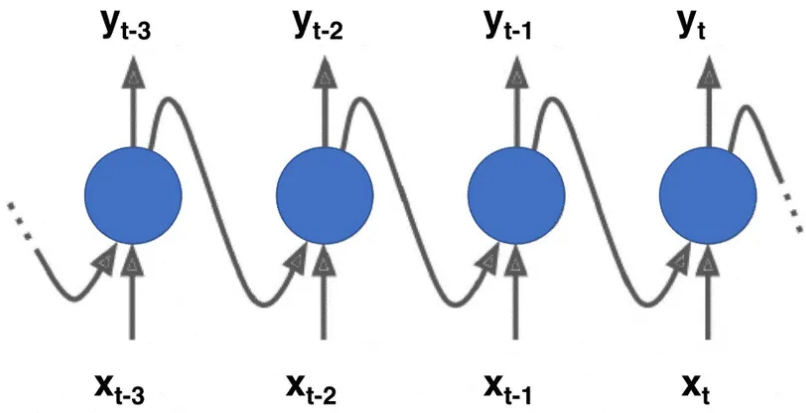


Figura 3.2: Ejemplo de una red neuronal recurrente procesando una secuencia. [19]

Los transformers se caracterizan por el uso de mecanimos de Auto-atención. Este mecanismo se basa en codificar el dato teniendo en cuenta 3 características del dato. En el ámbito del lenguaje natural, estas 3 características son: la palabra, el orden de palabras y el contexto. Esto se puede extrapolar a otros dominios donde los datos no sean palabras, como

el dominio de las imágenes. Toda esta información está definida en el artículo [20].

La trasformación que hay que realizar para que los transformers puedan clasificar imágenes es significativa, teniendo que dividir las imágenes en una serie de *parches* y hacer esas operaciones de Auto-atención en los píxeles de cada parche. El artículo en el que se mencionaron por primera vez los ViT y su implementación es [21]. En el capítulo 6 se tratará más a fondo tanto la base teórica de este modelo como su implementación.

3.1.5. Clasificación de sonidos

El sonido que las aves emiten, junto con su forma, tamaño y colores, es una de las características más distintivas de estos seres vivos. No obstante, primero es necesario hacer un preprocesamiento muy exhaustivo y específico de dichos sonidos para obtener características que puedan servir para entrenar un modelo de aprendizaje automático que pueda clasificar los sonidos.

El sonido es una onda física producida por las diferencias de presión en el aire. Hay muchas maneras distintas de procesar los sonidos para entrenar modelos de aprendizaje automático. No obstante, dependiendo del dominio del problema, unas características aportarán más información que otras, mejorando el aprendizaje del modelo. De nuevo, este proceso de extracción de características de sonidos es un proceso de prueba y error.

Las características que más comúnmente se utilizan son los *espectrogramas* y los *cepstrums*, como por ejemplo los *Coeficientes Cepstrales de la frecuencia de Mel* (MFCC), *Coeficientes Cepstrales de la Frecuencia Gammatone* (GTCC) o los *Coeficientes Cepstrales de Predicción Lineal* (LPCC). En los artículos [22], [23] se hace un resumen de los cepstrum más usados hasta la fecha para la clasificación de aves. Dependiendo de las características que se extraigan, así como del procesamiento posterior que se les realicen, se entrenarán unos modelos u otros. Las características extraídas son matrices de 2 dimensiones, es decir que se pueden interpretar como imágenes con un solo canal de color.

En el caso de que tras obtener las características, se haga un aplanado de estas (se transforma la matriz de 2 dimensiones en un vector de 1 dimensión apilando horizontalmente las filas de la matriz), se pueden entrenar modelos como máquinas de vector soporte, árboles de decisión como random forest o redes fully connected para generalizar el concepto. En el artículo [22] se hace uso de este método, usando una máquina de vector soporte para hacer la clasificación. Si por el contrario no se hace el aplanado, se utilizan técnicas de visión artificial, normalmente basadas en CNN como las EfficientNet o basadas en transformers como los Visual Transformers.

Para seleccionar un método u otro, es crucial tener en cuenta primero el número de características extraídas de los sonidos, y segundo el número de datos disponibles para entrenar los algoritmos. En caso de tener pocos datos con pocas características se recomienda el uso del primer método, ya que se usan modelos más simples con menos parámetros a aprender, mientras que, si se disponen de muchos datos con muchas características, los modelos de visión artificial suelen presentar mejores resultados.

3.2. Segmentación de objetos en imágenes

3.2.1. Segmentación de imágenes con CNN

En el apartado anterior, se han definido las redes convolucionales para resolver problemas de clasificación, es decir, dar una imagen como entrada a la red para que esta la clasifique entre 1 de las n clases para las que fue entrenada.

Sin embargo, estas redes ofrecen más funcionalidades en otros dominios, como la segmentación de objetos en imágenes. La segmentación de objetos en imágenes consiste en, dada una imagen con 1 o más objetos, identificar los n objetos existentes en dicha imagen. Los objetos en la imagen pueden pertenecer a distintas clases, aunque en este caso serán solo aves. En la Figura 3.3 se puede ver un ejemplo de una imagen habiendo segmentado todas las aves.

Los modelos de segmentación de imágenes son capaces tanto de segmentar los objetos en una imagen, como de clasificarlos. Otra opción es en un primer paso segmentar todos los objetos, y luego otro modelo es entrenado para clasificarlos. En este proyecto se seguirá la segunda aproximación.

Figura 3.3: Ejemplo de segmentación de aves en una imagen usando CNN. Elaboración propia.

Para hacer segmentación de objetos en imágenes, el modelo más utilizado y el que mejores resultados presenta es el modelo YOLO, que está basado en CNN. En el Capítulo 5 se detallará el funcionamiento de este modelo, así como su entrenamiento para la segmentación de aves como en la Figura 3.3. En el artículo [24] se hace uso del modelo YOLO para la segmentación de aves en imágenes.

3.3. Segmentación por colores

Con fin de segmentar los objetos contenidos en una imagen, otro método extensamente utilizado antes de la llegada de las redes neuronales fue la segmentación por colores.

El procedimiento de esta técnica empieza por eliminar el fondo de la imagen (se puede interpretar como ruido). Los colores de los bordes de la imagen se escanean y se hace un ranking según la frecuencia de aparación de cada color en un histograma. Se establece un umbral y una serie de heurísticas para definir qué colores son fondo de la imagen y cuáles no. A continuación se recorren todos los píxeles de la imagen comparando su color con la información ofrecida por el histograma, siendo considerado fondo u objeto en cada caso.

En caso de que la imagen esté en blanco y negro, habrá un solo histograma. Si la imagen tiene 3 canales de color (RGB), tendrá un histograma por cada canal. El paso de seleccionar el umbral a partir del cual unos colores se consideran fondo y otros no, es un proceso iterativo de prueba y error, que depende del dominio y características de las imágenes. En el artículo [25] se detalla el proceso seguido para hacer la segmentación por colores explicada anteriormente.

Aunque la segmentación por colores es un método que requiere poca capacidad de cómputo, es una técnica que presenta numerosas desventajas. Entre ellas cabe destacar que dependiendo de las caracterísiticas de la imagen, puede que el fondo no sea muy diferenciable de los objetos. En el caso específico de las aves, en caso de que el fondo de la imagen no presente un color uniforme (si hay nubes, distinta luminosidad u otros factores que afecten a la diferencia de colores entre las aves y el fondo) la tasa de acierto de este modelo para segmentar aves se reduce drásticamente. Como las redes neuronales resuelven este problema aprendiendo este tipo de peculiaridades (las CNN aprenden a reconocer los objetos sin importar los colores del fondo), las redes neuronales han sustituido a esta técnica en los últimos años.

Capítulo 4

Obtención de datos

Los datos son la base de todos los algoritmos de aprendizaje automático supervisado. Si no se dispone de unos datos de calidad, sin sesgo y relevantes para el concepto, no importa los modelos que se usen ni las características del hardware con el que se entrenen los algoritmos. Los datos y el tratamiento de ellos es una fase clave en cualquier proyecto donde se use aprendizaje automático. En este capítulo se procede a indicar la fuente de los datos, sus características y forma de extracción. Se hará también un resumen ejecutivo de cada dataset, y se dejará para posteriores capítulos el preprocesamiento realizado, así como su uso.

4.1. Dataset para segmentación de imágenes

El primero de los dataset necesarios para entrenar los modelos que conforman este proyecto es el dataset con imágenes donde se han segmentado todas las aves. Dado que el objetivo final del proyecto es poder utilizar todos estos modelos con los vídeos, imágenes y audios extraídos con la cámara AXIS Q6225-LE PTZ Camera [1], se han obtenido imágenes con dicha cámara, y a continuación se han segmentado las aves en esos fotogramas. El control, puesta a punto y funcionamiento de la cámara no concierne a este proyecto, así que simplemente se va a hacer uso de la información que ofrece. En la Figura 4.1 se puede ver un ejemplo de una imagen extraída por la cámara.

Figura 4.1: Ejemplo de una imagen extraída con la cámara AXIS Q6225-LE PTZ Camera. Elaboración propia.

Se ha hecho uso de la herramienta online Make Sense [2] para segmentar las aves en las imágenes. En la Figura 4.2 se muestra una imagen segmentada con esta herramienta.

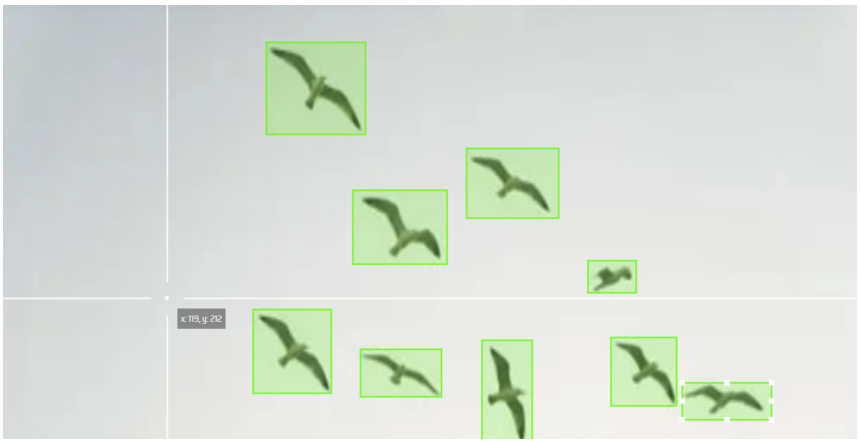


Figura 4.2: Ejemplo de una imagen segmentada con Make Sense [2]. Elaboración propia.

La salida que ofrece la herramienta es un fichero con extensión *.txt* por cada imagen que se ha segmentado. En cada uno de esos ficheros, cada línea hace referencia a cada uno de los objetos pertenecientes a la imagen. Cada línea tiene 5 números. El primero de ellos hace referencia a la clase, y los 4 siguientes sirven para identificar el cuadro delimitador de ese objeto. Los 2 primeros son la (x,y) del punto inferior izquierdo del cuadro delimitador, y los 2 siguientes son la (x,y) del punto superior derecho del cuadro delimitador. Con esos dos puntos, ya se puede obtener el cuadro delimitador completo.

$$0 - 0{,}021985 - 0{,}279523 - 0{,}028894 - 0{,}033920$$

La expresión anterior es un ejemplo de un cuadro delimitador. Cabe destacar que las (x,y) del cuadro delimitador están contenidas en el intervalo [0,1]; esto se debe a que han sido escaladas en dicho intervalo.

4.1.1. Resumen ejecutivo

En el conjunto de datos extraído con la cámara AXIS Q6225-LE PTZ Camera [1], hay un total de 89 imágenes con aves. Solo hay una clase (nominal), ya que la finalidad de este modelo será únicamente identificar las aves, sin importar su especie. Con respecto a los atributos, a todas las imágenes se les hará una modificación de sus dimensiones a 3x640x640 píxeles, de forma que hay 640 * 640 * 3 = 1.228.800 atributos por cada imagen, todos ellos numéricos. Se usarán directamente estos atributos, sin hacer ninguna operación de reducción de dimensiones.

4.2. Dataset para la clasificación de imágenes

Para la clasificación de imágenes, se quiere poder reconocer el mayor número posible de tipos de aves distintos. No obstante, como para ello se necesitaría obtener mucha información con la cámara que llevaría mucho tiempo, primero se van a entrenar los modelos con un dataset que contiene 4 clases: no-aves, halcones, cuervos y otros tipos de aves. Es un dataset de código abierto, perteneciente a Naemura-Lab [26]. Con él se llevó a cabo un proyecto en el cual se paran los molinos de viento en el momento en el que un ave se acerca a uno de ellos.

4.2.1. Resumen ejecutivo

El conjunto de datos se compone de un total de 28.987 imágenes. El tamaño de cada una de ellas es variable, ya que son recortes de imágenes más grandes (se segmentaron las aves). Estas variaciones van aproximadamente desde los 46*46 píxeles hasta los 600*600 píxeles. Todas ellas tienen 3 canales de color (RGB).

Con respecto a las clases, son 4 (todas nominales) y sus distribuciones de probabilidad son las siguientes:

- Halcones: 3031 ejemplos con distribución 0,104.
- Cuervos: 998 ejemplos con distribución 0,0344.
- No-aves: 19.999 ejemplos con distribución 0,689.
- Otras aves: 4.959 ejemplos con distribución 0,171.

4.3. Dataset para clasificación de audio

Uno de los requisitos iniciales del proyecto fue utilizar los datos del dataset público *Xeno-canto* [27]. Es un dataset que consta de una gran cantidad de audios con cantos de aves de todo el mundo. Para obtenerlos, se pueden descargar directamente de la página web o se pueden extraer a través de su propia API.

Para la realización de este proyecto, se descargaron los audios a través de la API. El código para obtener estos audios es tan sencillo que no merece mayor comentario. En la página web [28] se hace una explicación de cómo hacer la descarga, así como el contenido de los datos y opciones de descarga. Los audios son de longitud variable, de modo que en un mismo audio hay varios cantos del ave. La segmentación de estos audios y su preprocesamiento se discutirá en el Capítulo 7.

4.3.1. Resumen ejecutivo

El conjunto de datos descargados del dataset Xeno-canto [27] son solo de la Comunidad Autónoma de Castilla y León, pues el objetivo del proyecto es estudiar únicamente las aves de dicha Comunidad Autónoma. Se han obtenido un total de 1.143 archivos, de 131 clases diferentes (todas nominales). No obstante, se va a utilizar el dataset segmentado.

Tras la segmentación, se obtienen un total de 17.942 ficheros. Al hacer la descarga de los audios originales se obtienen muchos datos y atributos de ese audio, pero por ahora solo interesa la onda, que es un dato numérico. A partir de esa onda, se hará una extracción de características que se explicará en el Capítulo 7, y posteriormente se entrenarán algoritmos de aprendizaje automático.

Capítulo 5

Identificación de aves por imagen

La segmentación de objetos en imágenes consiste en, dada una imagen, ser capaz de identificar los distintos objetos en dicha imagen. Hay muchos modelos que permiten realizar esta tarea, pero en este proyecto se ha elegido usar un modelo de la familia YOLO (acrónimo del ingés 'You Only Look Once'), que es YOLOv8. Estos modelos están basados en redes neuronales convolucionales (CNN) con una serie de modificaciones. En los siguientes apartados se va a hacer una explicación detallada de los modelos y redes que usan estos modelos.

5.1. Base teórica

5.1.1. Funciones de pérdida y descenso del gradiente

Las redes neuronales son modelos de aprendizaje automático que aprenden sus parámetros a través del descenso del gradiente el cual permite optimizar las funciones de pérdida del modelo.

Las funciones de pérdida consisten en funciones que cuantifican el error que ha cometido la red al clasificar un ejemplo. Esto lo realizan comparando los valores predichos por la red y los valores reales del ejemplo (aprendizaje supervisado). Los valores predichos dependen de los parámetros que la red tiene que ir aprendiendo en el entrenamiento. Como el objetivo es que la pérdida sea 0, la red tiene que ir actualizando esos parámetros para que el valor de la función de pérdida sea lo más pequeño posible (idealmente 0). Existen muchos tipos de funciones de pérdida con distintas características, como el error cuadrático medio o la entropía cruzada.

Para hacer esta actualización de parámetros (también llamados pesos), se usan distintos métodos de optimización de funciones (adam, adaDelta, adaGrad, etc) aunque todos están

basados en el descenso del gradiente. El gradiente de una función se define como la dirección en la que más pendiente hay. Su cálculo consiste en un vector con tantas componentes como parámetros tenga la función, y cada componente es la derivada parcial de la función de pérdida con respecto a ese parámetro, es decir $\frac{\partial}{\partial\theta_j}\mathrm{J}(\theta)$. Como el objetivo es minimizar la función de pérdida, el gradiente se pone con signo negativo (Ecuación 5.1). En caso de que fuera positivo se estaría maximizando la función.

La fórmula de la actualización del parámetro θ_j usando el descenso del gradiente se representa en la Ecuacion 5.1, donde θ_j es el j-ésimo parámetro del modelo, $\frac{\partial}{\partial\theta_j}J(\theta)$ es la derivada parcial de la función de pérdida $\mathrm{J}(\theta)$ respecto al parámetro θ_j y α es la tasa de aprendizaje (sirve para hacer actualizaciones de pesos más grandes o más pequeños).

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}\mathrm{J}(\theta) \tag{5.1}$$

5.1.2. Redes convolucionales

Las redes convolucionales son las redes que mejores resultados presentan en el dominio de los datos en forma de imagen. La principal ventaja que presentan este tipo de redes, es que son capaces de captar la información espacial que tiene cada píxel de la imagen. Esto significa que una red convolucional es capaz de aprender e identificar formas y patrones independientemente de su localización en la imagen. Esto se consigue a través de una serie de operaciones que se discutirán en las siguientes secciones.

Convolución

La convolución consiste en la aplicación de un filtro o kernel de tamaño fijo de 2 dimensiones a una porción de la imagen. Se define como F al filtro, i a la fila i-ésima del filtro y j a la columna j-ésima, de forma que se reprensentará como F_{ij} a la celda ij del filtro F. De la misma forma, se representa a la imagen como I, k a la fila k-ésima de la imagen y l a la columna l-ésima de la imagen.

La operación de la convolución se representa en la Ecuacion 5.2, donde I es la imagen, l es el tamaño del filtro, F es el filtro, y m, n son los offset de la imagen vertical y horizontal (inicialmente ambos valen 0).

$$\mathrm{Conv}(I,F)(m,n) = \sum_{i=0}^{l}\sum_{j=0}^{l} I_{m+i,n+j} * F_{ij} \tag{5.2}$$

Los filtros suelen tener tamaños 3x3, 5x5 o 7x7. De esta manera, al aplicar una convolución con filtros de esos tamaños, se están teniendo en cuenta 9, 25 y 49 píxeles respectivamente.

De esta forma no solo se tiene en cuenta el valor de cada uno de ellos individualmente, sino también los valores de los píxeles contiguos, siendo posible aprender formas y patrones independientemente de sus localizaciones en la imagen. Tras la aplicación del filtro a la imagen para *m, n = 0*, se actualizan esos offset para aplicar el filtro a toda la imagen. Las fórmulas que definen las actualizaciones de estos offset son las siguientes:

$$m := m + sv \tag{5.3}$$

$$n := n + sh \tag{5.4}$$

En las fórmulas anteriores, *sv* y *sh* son los *stride* vertical y horizontal. El stride marca el 'salto' que hace el filtro tras aplicar la Ecuación 5.2. Normalmente los strides tienen valor 1, aunque es posible que tengan valores más grandes. La razón principal para que el valor del stride sea mayor que 1 es realizar menos operaciones, y por tanto usar menos cómputo. Evidentemente el valor del stride no puede ser mayor que el tamaño del filtro, pues en ese caso quedarían celdas de la imagen sin procesar y se estaría perdiendo información. De la misma manera, su valor no puede ser 0, pues entonces el filtro no estaría cambiando de posición. Primero se actualiza la n (las columnas) y una vez que se ha llegado al final de la imagen, se vuelve a poner a 0 y se actualiza la m (filas).

El filtro que se ha definido en la Ecuación 5.2 tiene 2 dimensiones, es decir, que la entrada tiene únicamente 1 solo canal. En caso de que la imagen tenga 3 canales (RGB) la expresión se puede generalizar a la Ecuación 5.5, donde c es el número de canales de la imagen.

$$\text{Conv}(I, F)(m, n) = \sum_{k=0}^{c} \sum_{i=0}^{l} \sum_{j=0}^{l} I_{k,m+i,n+j} * F_{kij} \tag{5.5}$$

La salida de la aplicación de un filtro a toda la imagen se llama *característica*. En cada capa convolucional se aplican una gran cantidad de filtros, ya que cuantos más filtros se apliquen, más características se obtienen y mayor capacidad de generalización tiene la red. Como por cada filtro aplicado en una capa se obtiene una característica de salida, el número de características de salida será igual al número de filtros aplicados. Las entradas de la capa convolucional n serán las características extraídas en la capa *n-1*, y se aplicará la Ecuación 5.5 sustituyendo la c por el número de características provenientes de la capa *n-1*.

Los valores de los filtros son los valores que la red va a ir aprendiendo en todo el proceso de entrenamiento mediante la función de pérdida y el descenso del gradiente.

Pooling

La operación de pooling se aplica tras la operación de convolución. De nuevo tiene un tamaño fijo (normalmente es 3x3) y se aplica de la misma forma que la convolución, va

iterando por la imagen cambiando de posición. Ese cambio de posición viene definido por su stride (distinto al de la operación de convolución).

La idea que hay detrás de esta operación es reducir la dimensionalidad de las características, eligiendo el valor de un píxel de los 9 (3x3) de una región de la característica, pero perdiendo la menor cantidad de información posible. Hay distintas maneras de hacer esta elección, aunque la más común es *Max-pooling*, que consiste en elegir el valor máximo. La operación de Max-pooling viene definida por la Ecuación 5.6, donde S_{ij} es el conjunto de valores de la imagen centrados en los índices *ij*.

$$\text{MaxPooling}(I)(i,j) = \max_{(p,q)\in S_{i,j}} I(p,q) \tag{5.6}$$

Padding

Si se estudia el número de veces que cada píxel de la imagen es procesado por un filtro, el resultado es que los píxeles del centro de la imagen son procesados más veces que los de los bordes. De esta manera, los píxeles del centro de la imagen tienen más peso a la hora de clasificar imágenes.

Para paliar este problema, se creó el padding, que consiste en aumentar de tamaño la imagen añadiendo más píxeles en el borde de la imagen. Hay distintas maneras de hacer esta operación. Por ejemplo, se puede hacer una interpolación en base a los píxeles cercanos del borde, pero lo más común es añadir esos píxeles con valor 0. Esta operación también se usa cuando es necesario ampliar el tamaño de la imagen, ya que siempre que se aplica una operación de convolución o de pooling se disminuye el tamaño de la característica.

Capa de clasificación

Al proceso de aplicar a una imagen todas las capas convolucionales de una red convolucional se le llama *extracción de características*. Estas características son las características en las cuales se basará el modelo para hacer la clasificación. Para realizar la clasificación hay 2 formas de realizar la inferencia.

La primera de ellas consiste en acoplar una red fully connected (funcionan con entradas de 1 dimensión) tras hacer un aplanado de las características. Aplanar las características consiste en convertirlas de 2 dimensiones a 1. La red fully connected en su capa de entrada tendrá tantas neuronas como valores aplanados tengan las características, y en la capa de salida tendrá tantas neuronas como número de clases. Se suele usar una función *softmax* como función de activación de la última capa, ya que el recorrido de dicha función es [0,1] y se pueden interpretar sus salidas como una distribución de probabilidad.

La segunda forma es incluir una última capa convolucional con tantos filtros como clases. Cada uno de esos filtros tendrá el tamaño de las características de entrada a esa capa. De

esta manera se obtendrán tantas características de salida como clases, y cada característica tendrá un solo valor (al aplicar a una imagen de tamaño NxN un filtro de tamaño NxN se obtiene un solo valor), que tras aplicarles de nuevo la función softmax, se interpretan como probabilidades de pertenencia a las clases.

Funciones de activación

Los conceptos objetivos o funciones que las redes neuronales tratan de aproximar en la mayoría de casos son funciones no lineales. Dado que las redes neuronales aplican operaciones lineales, teóricamente serían incapaces de aproximar a la perfección esas funciones no lineales.

Por eso se aplican una serie de funciones no lineales a las funciones lineales de las redes neuronales para poder aproximar mejor el concepto. Esas funciones no lineales se llaman funciones de activación, y se aplican siempre a la salida de cualquier capa de una red neuronal. Existen muchas funciones de activación (softmax, tangente hiperbólica, sigmoide, etc.) aunque las funciones de activación más usadas en la actualidad son las de la familia *relu*. La fórmula de la función de activación relu es la de la Ecuacion 5.7. La función de activación relu presenta una serie de desventajas, y es que cuando el valor es negativo, lo cambia por valor 0. El problema es que al realizar la propagación del gradiente hacia las primeras capas, al ser valor 0, el gradiente se va haciendo cada vez más pequeño hasta que es 0 y por tanto las primeras capas de la red no aprenden. A este problema se le conoce como evanescencia del gradiente.

$$\text{Relu}(x) = \text{máx}(0, x) \tag{5.7}$$

Para resolver ese problema, se usa la función de activación *leaky relu*, cuya fórmula es la representada en la Ecuación 5.8, donde el valor α es un valor cercano a 0 como 0,01 para evitar el problema de la evanescencia del gradiente.

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases} \tag{5.8}$$

Entre otras funciones de activación también muy utilizadas están la tanh y la softplus, o también llamada softmax, cuyas ecuaciones se representan en la Ecuación 5.9 y 5.10 respectivamente. La salida de la función tanh está contenida en el intervalo [-1,1], mientras que la salida de la función softmax está en el intervalo [0,1]. Debido a que la función softmax tiene su salida en ese intervalo, esta se interpreta como una distribución de probabilidad. Esto es muy común en las salidas de las redes fully connected, donde se ponen tantas neuronas en la capa de salida como clases. Las salidas se consideran probabilidades de pertenencia a las clases, aunque no sean formalmente una distribución de probabilidad, pues la suma de todas ellas no siempre es 1.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.9}$$

$$\text{Softplus}(x) = \log(1 + e^x) \tag{5.10}$$

5.1.3. Capas de transición

Después de la aplicación de una operación de convolución, se hace siempre un pooling (en cualquiera de sus variantes aunque el más común es Max-pooling) para reducir la dimensionalidad (menor capacidad de cómputo requerida) perdiendo la mínima cantidad de información posible. A continuación se aplica una función de activación, que suele ser alguna de la familia relu (relu, leaky relu, mish, etc). A ese conjunto de 3 operaciones (convolución + pooling + función de activación) se les agrupa como una sola capa y se llama *capa de transición*. Es posible que existan otras operaciones en estas capas, como el padding, la normalización (escalar los valores de los píxeles) o el downsampling/upsampling. El downsampling/upsampling consiste en disminuir/aumentar la resolución de las características. Esto se debe a que todas las operaciones de convolución, padding y pooling modifican la dimensionalidad de todas las características. Debido a que existen ocasiones en las que es necesario concatenar o sumar características, como en las capas CSP (Sección 5.1.6) o redes densas (Sección 5.1.5), es necesario que todos los elementos que se concatenen o sumen tengan las mismas dimensiones.

5.1.4. Redes residuales

Problema de la degradación

Existe un problema en las redes neuronales conocido como *problema de la degradación*, el cual consiste en que se ha comprobado experimentalmente que modelos más sencillos con menos capas y menos neuronas por capa pueden generalizar mejor que modelos más complejos. Para explicar este problema se expone el siguiente ejemplo.

Se define la función f(x) como el concepto, la función que la red va a intentar aproximar. Como ejemplo, definimos esta función de la siguiente forma (cabe destacar que en ningún caso esta función es conocida, solo se conocen pares (x_n, y_n) / f(x_n) = y_n):

$$\text{f}(x) = ax^2 + bx + c \tag{5.11}$$

La función en la Ecuación 5.11 tiene grado 2, por tanto puede ser aproximada por funciones que tengan grado mayor o igual que dos. Por ejemplo la función de la Ecuación 5.12 que tiene grado 3 puede aproximar la función de la Ecuación 5.11 haciendo que: a=e, b=k, h=c y d=0.

De esta manera en realidad lo que se está haciendo es anular el término que hace que la ecuación 5.12 tenga grado 3 (el coeficiente del término cúbico es 0) e igualar el resto de coeficientes. Esto significa que una función g(x) de grado m puede aproximar otra función f(x) de grado n, siendo $m >= n$, es decir que una función de mayor grado puede aproximar una función de menor grado.

$$\mathrm{g}(x) = dx^3 + ex^2 + kx + h \tag{5.12}$$

El tamaño de una red es equivalente al grado de las funciones anteriores, no obstante el resultado anterior no es extrapolable a las redes. Se podría pensar que una red más grande (más capas y más neuronas) va a funcionar al menos igual de bien que otra red más pequeña, pues se podrían poner a 0 las primeras capas y transformarse en una red más simple (igual que se hizo con la función del ejemplo anterior). No obstante, esto no es posible, ya que existen múltiples operaciones no lineales que se producen en la red que no se pueden anular. Aunque los pesos de las primeras capas se pongan a 0, las funciones no lineales convertirán esos valores iguales a 0 en otros valores distintos, dando un significado y un sentido a cada capa y a cada neurona. Tras comprobaciones experimentales, se ha comprobado que redes excesivamente complejas y grandes en relación al concepto y a los datos de entrada se comportan peor que redes más pequeñas y sencillas.

Además, si una red tiene más parámetros que optimizar, serán necesarios muchos más ejemplos para entrenar la red, y en caso de que esos datos no existan, habrá un infraentrenamiento que se traducirá en un comportamiento peor.

Redes residuales

Para solucionar el problema de la degradación se crearon las redes residuales. Estas redes crean nuevas conexiones entre capas de neuronas pero no introducen nuevos parámetros ni mayor coste computacional. La idea principal de la red residual es conectar la entrada de una capa a la salida de esa misma capa. La función de una red residual se muestra en la Ecuación 5.13.

$$\mathrm{H}(x) = \mathrm{F}(x) + x \tag{5.13}$$

Donde H(x) es la salida de la capa residual, F(x) es el llamado *residuo* ('identity' en inglés) y x es la entrada a la capa. Esto se puede extrapolar a la creación de bloques residuales, formados por distintas capas donde la entrada del bloque está conectado a la salida del bloque, y entre medias hay capas de transición (Sección5.1.3). El término *F(x)* se le denomina residual porque las capas que son 'saltadas' tratan de aprender el residuo, que despejado de la fórmula 5.13 se obtiene:

$$\mathrm{F}(x) = \mathrm{H}(x) - x \tag{5.14}$$

En la Figura 5.1 se observa un bloque residual de dos capas con sus conexiones.

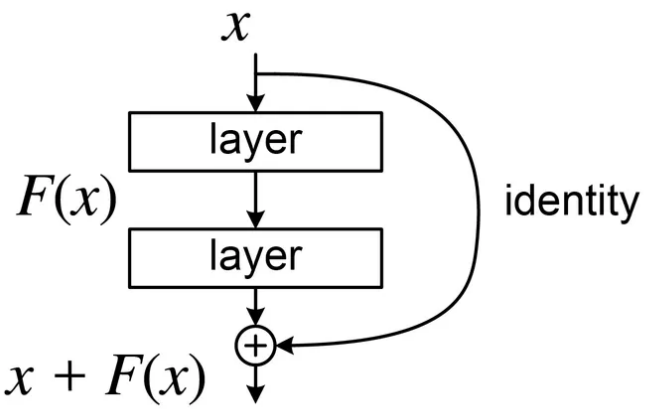


Figura 5.1: Ejemplo de un bloque residual. [29]

El uso de estas redes no solo soluciona el problema de la degradación, sino que también ayuda a paliar el problema de la evanescencia del gradiente ya que crea mayores conexiones que pueden propagar más fácilmente el gradiente hacia las primeras capas de la red.

5.1.5. Redes densas

Las redes densas son otras redes muy utilizadas en las CNN. De nuevo, ayudan a paliar el problema de la evanescencia del gradiente, pues crean más conexiones entre capas secuenciales, haciendo más directa la propagación del gradiente hacia las primeras capas. No obstante, su objetivo principal es ser capaz de disminuir el número de parámetros a aprender, pues hasta entonces, todas las redes convolucionales requerían de una gran cantidad de parámetros para aprender, además de mucha capacidad de cómputo. La idea es que todas las capas estén conectadas con las capas posteriores. Para ello, la entrada de la capa n será la concatenación de todas las características provenientes de las *n-1* capas anteriores, es decir:

$$\mathrm{y_n} = \mathrm{F}[y_0, y_1, \ldots, y_{n-1}] \tag{5.15}$$

Donde $\mathrm{y_n}$ es la salida de la capa n, F es la aplicación de las operaciones de la capa n y [a,b] representa la concatenación de la tupla (a,b). Las salidas de las capas *0, 1,* y *n-1* son y_0, y_1 y y_{n-1} respectivamente.

5.1.6. Cross Space Partial Network

Las Cross Space Partial Network (CSP) van en la misma línea que las redes residuales y las redes densas, su objetivo final es hacer una combinación más rica del gradiente facilitando el aprendizaje, a la vez que reducir el coste computacional reduciendo el número de parámetros a aprender.

El funcionamiento de este tipo de redes se basa en dividir en 2 todas las características de entrada a un bloque, es decir:

$$x_n = [x'_n, x''_n] \tag{5.16}$$

Donde x_n es la entrada de la capa n, x'_n es la primera partición que va a ser procesada por capas de transición contenidas en el bloque CSP, y x''_n es la segunda partición que se concatenará a la salida de las capas de transición del bloque CSP. Posteriormente se volverá a pasar ese resultado de la concatenación por una capa de transición, obteniendo la salida del bloque CSP. La salida de un bloque CSP se define en la Ecuación 5.17, donde y_n es la salida de la capa n, T(x) es el resultado de aplicar una capa de transición a las características contenidas en x, y [a,b] representa la concatenación de la tupla (a,b). Las capas de transición aplicadas antes y después de la concatenación son distintas.

$$\mathrm{y_n} = \mathrm{T}([x''_n, \mathrm{T}'(x'_n)]) \tag{5.17}$$

En cuanto a las funciones de activación que usan este tipo de redes, la más comúnmente utilizada es la función *mish*. La fórmula de esta función se define en la Ecuación 5.18, donde tanh y softplus son funciones de activación explicadas en la Sección 5.1.2.

$$\mathrm{mish}(x) = x * \tanh(\mathrm{softplus}(x)) \tag{5.18}$$

La ecuación en 5.18 es equivalente a:

$$\mathrm{mish}(x) = x * \tanh(\ln(1 + e^x)) \tag{5.19}$$

La principal ventaja que tiene esta función de activación es que es una función no monótona que es más suave que las otras de su familia (relu o leaky relu), propiciando un mejor aprendizaje.

5.1.7. YOLOv8

El modelo YOLOv8 fue uno de los primeros modelos de segmentación de objetos que presentó unos buenos resultados. El modelo YOLOv8 es un modelo complejo, donde se pueden diferenciar 3 grandes módulos:

1. **Backbone:** Se encarga de hacer la extracción de características de las imágenes de entrada.
2. **Neck:** Se encarga de hacer una combinación de las características extraídas en el backbone para mejorar los resultados de la segmentación.
3. **Head:** Hace las predicciones tanto de las clases como de los cuadros delimitadores de cada objeto.

Backbone

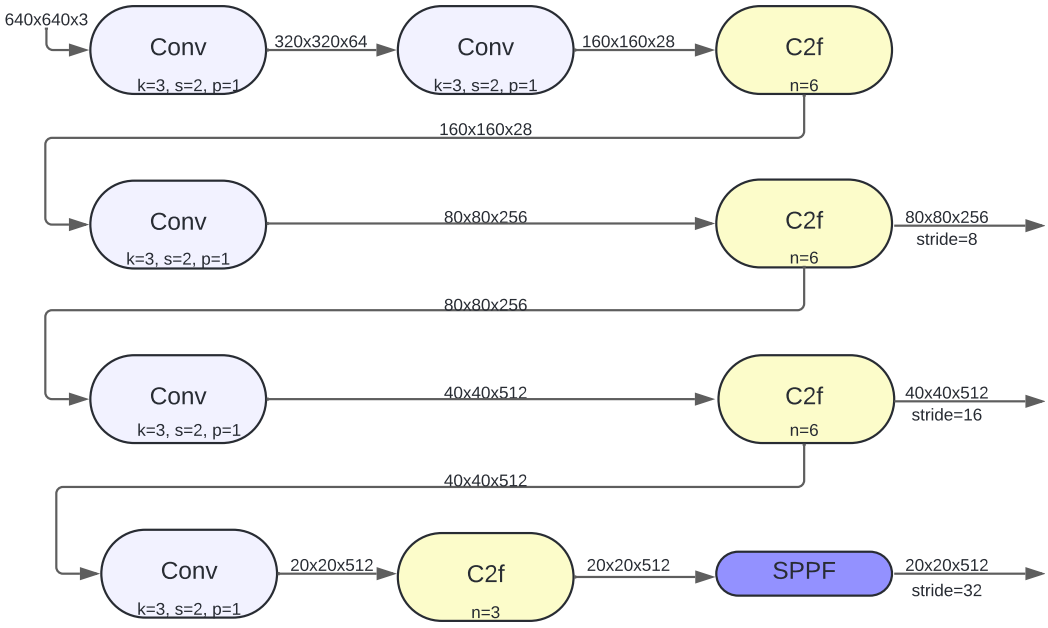


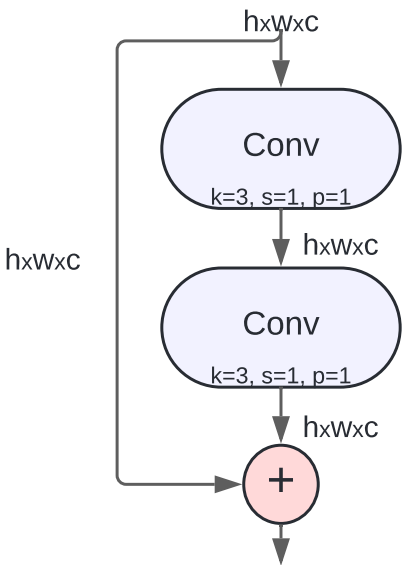
Figura 5.2: Backbone de YOLOv8. Elaboración propia.

El backbone que utiliza YOLOv8 es conocido como *CSP-Darknet53*. Este módulo está basado en la conjunción de capas densas (Sección 5.1.5), capas CSP (Sección 5.1.6) y redes residuales (Sección 5.1.4).

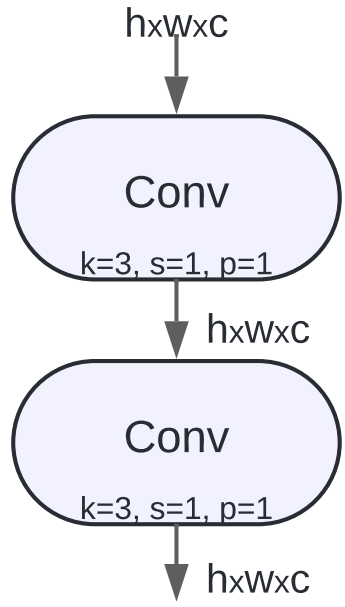
En la Figura 5.2 se define el backbone del modelo YOLOv8. Cabe destacar que en las Figuras 5.3, 5.4a y 5.4b, la k hace referencia al tamaño del kernel, p al padding, s al stride,

w,h a la anchura y altura de las características y c al número de canales. Los módulos que usa YOLOv8 en el backbone son:

- **Conv:** Consiste en la aplicación de una capa convolucional y una operación de Max-pooling. Después hay una normalización de todos los valores y por último se aplica la función de activación *silu* (de la familia relu).
- **Cuello de botella:** Consiste en 2 capas convolucionales, de forma que la primera capa convolucional usa pocos filtros, produciendo pocas características. De esa forma la computación es menos costosa en la siguiente capa, pues tiene menos parámetros que aprender. La siguiente capa del módulo usa muchos más filtros, volviendo a usar muchas características. Se puede usar este cuello de botella con una capa residual (Figura 5.3a) o sin ella (Figura 5.3b).
- **C2f:** Es una actualización del módulo C3 de versiones anteriores de YOLO. Presenta mejores resultados que su homólogo C3, y su arquitectura se presenta en la figura 5.4b. Cabe destacar que utiliza un móludo Conv, a continuación se añade un módulo denso (Sección 5.1.5) con capas de cuello de botella (Sección 5.3), y esa salida se pasa de nuevo por otro módulo Conv.
- **SPPF:** Es el último módulo del backbone, y es una actualización del SPP aplicado en otros modelos YOLO. Aplica una serie de capas de Max-pooling con intención de crear una salida con tamaños fijos. Usa la técnica de CSP, es decir que antes de entrar en una capa de pooling hace una división de características, de forma que algunas no entran en la capa de pooling. Tras hacer todas las operaciones de pooling, se concatenan los resultados del pooling y las características reservadas anteriormente, y se pasan todas por una capa Conv. En la Figura 5.4a se ve la estructura de este módulo.



(a) Con conexión residual. Elaboración propia

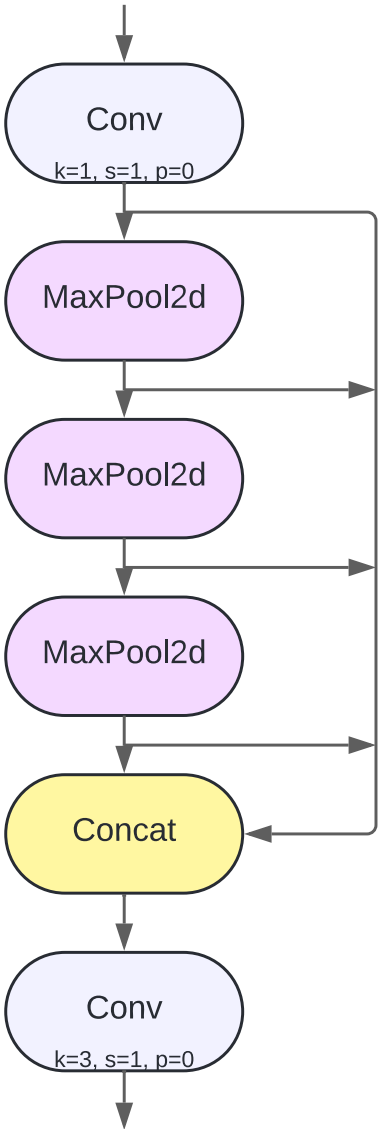


(b) Sin conexión residual.

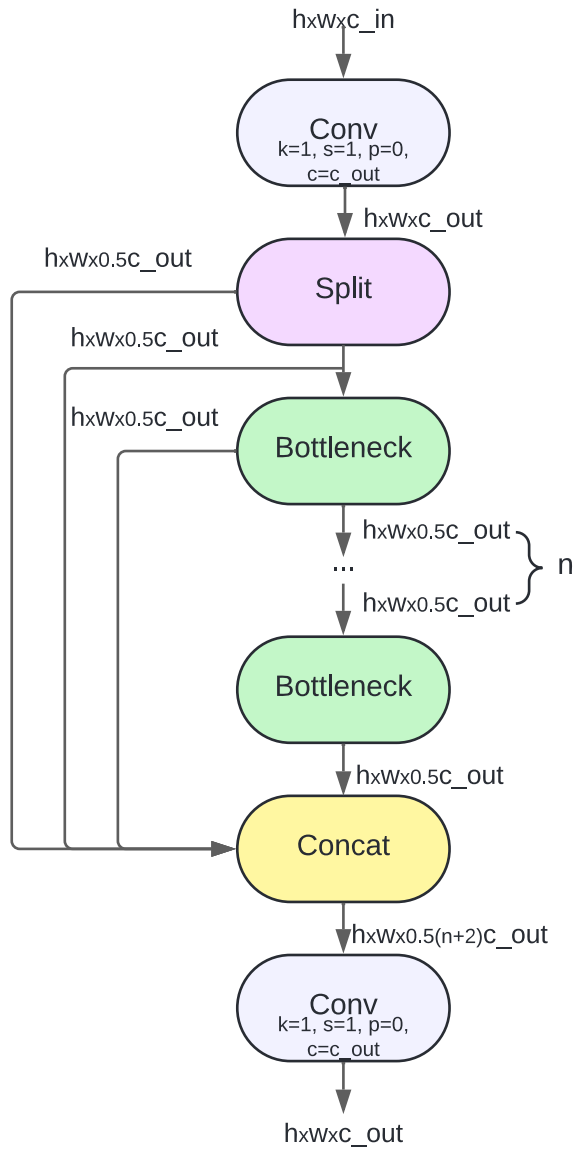
Figura 5.3: Cuellos de botella en YOLOv8.

En la Figura 5.2 se aprecia que en 3 de las capas del backbone, sus salidas se pasan al siguiente módulo de YOLOv8, el neck. Esto se debe a que dependiendo del número de convoluciones hechas en las características, estas tendrán un tipo de información u otro.

Hay 2 tipos de información en las imágenes. Una es la información semántica, que es el objeto o el significado de la imagen, y el otro tipo es la información espacial, que es donde está situado dicho objeto. En las primeras capas, las características contienen mucha información espacial y poca información semántica, pues todavía no se ha perdido mucha información en las capas de pooling y se han llevado a cabo pocas operaciones convolucionales como para haber detectado formas y patrones (información semántica). De la misma manera, en capas profundas con baja resolución, muchos de los patrones presentes se han podido detectar, pero a costa de haber perdido información espacial al haberse reducido en gran medida las dimensiones de las características. Como YOLOv8 quiere detectar ambos tipos de información, espacial y semántica, va a hacer una serie de combinaciones de las características de salida de distintas capas para captar la información semántica de las capas profundas y la información espacial de las primeras capas. De esto se encargará el neck.



(a) Módulo SPPF de YOLOv8. Elaboración propia.



(b) Módulo C2f de YOLOv8. Elaboración propia.

Figura 5.4: Módulos de YOLOv8.

Neck

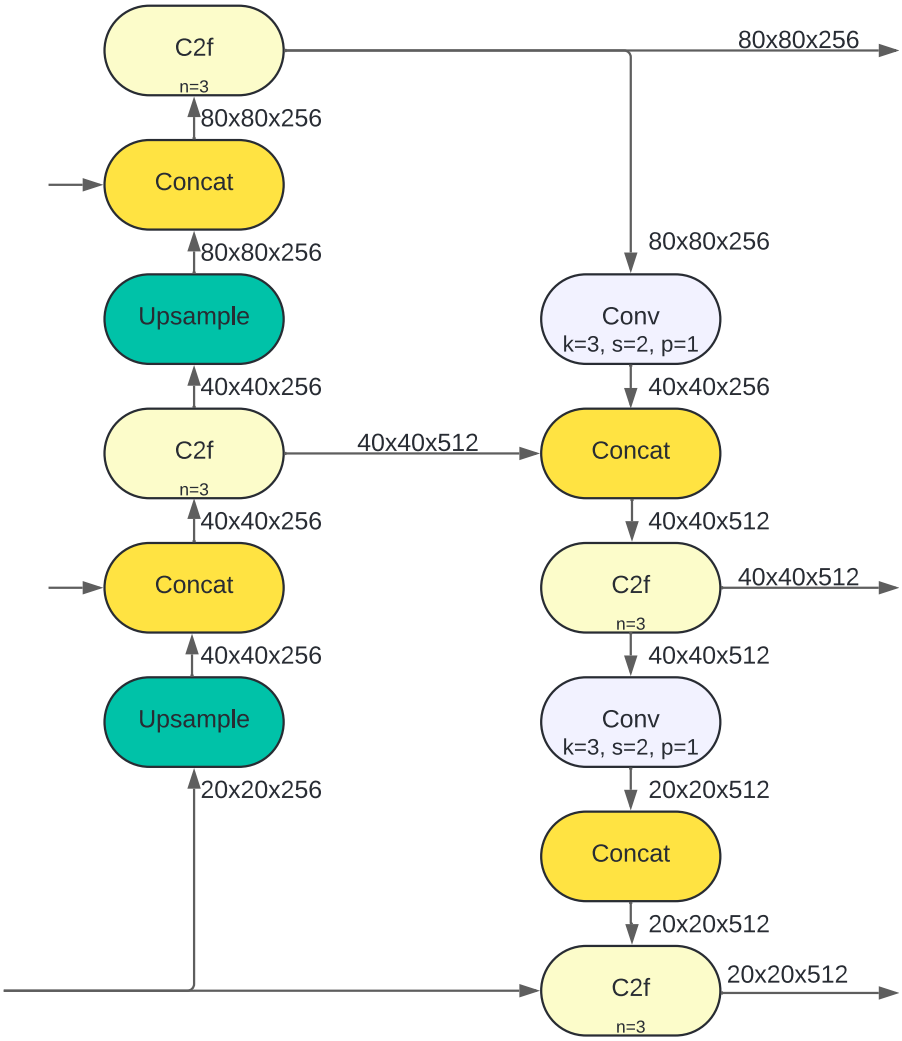


Figura 5.5: Neck de YOLOv8. Elaboración propia.

Como se ha mencionado anteriormente, el neck combina la información de distintos niveles provenientes del backbone para obtener la información espacial de las primeras capas y la información semántica de las capas profundas. A este proceso se le conoce como *agregación de características*. El neck que utiliza YOLOv8 es el llamado *Path Aggregation Netwok* (PAN), también usado en otros modelos de la familia YOLO. El modelo PAN está formado por los mismos módulos que conforman el backbone. Cabe destacar que se utilizan capas de upsampling, que como se ha explicado anteriormente, consiste en aumentar las dimensiones de las características para poder ser concatenadas unas con otras.

Con respecto a las salidas de este módulo, hay 3, cada una de ellas irá a una *head* distinta. Las head son el último módulo de YOLOv8 y se encargan de hacer las predicciones.

Head

En el modelo YOLOv8 estándar, existen 3 head, aunque en modelos más grandes de YOLO se pueden usar más. En la Figura 5.6 se ve la estructura de este módulo. La razón de que haya más o menos módulos de predicción es que las head que tienen como entrada características con mucha resolución están más especializadas en la detección de objetos pequeños, mientras que las que tienen como entrada características pequeñas detectan objetos grandes.

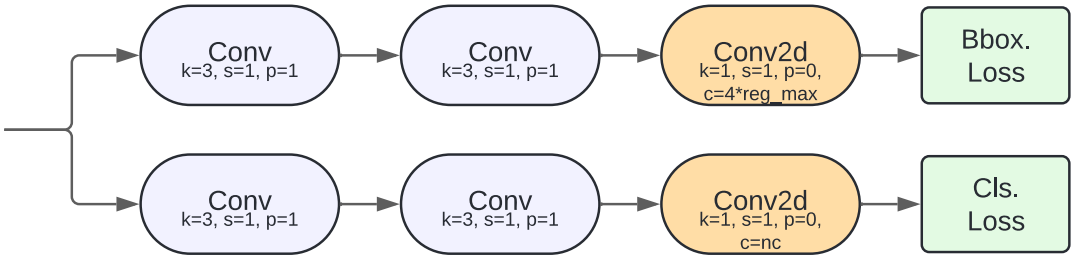


Figura 5.6: Head de YOLOv8. Elaboración propia.

La idea es que en las head se dividen las características en una matriz con un número de celdas fijo. Por cada una de esas celdas de esa matriz se hacen 3 predicciones (este parámetro puede variar de un modelo a otro). Cada head hace esas celdas de distinto tamaño, y así pueden detectar cada una objetos de distinto tamaño.

Cada una de esas predicciones es un vector. El tamaño de ese vector depende del número de clases que se estén detectando. Siendo nc el número de clases a detectar, el número de componentes del vector de la predicción será 5 + nc. El 5 se debe a que se predice la probabilidad o la confianza de pertenencia a la clase predicha, y los 4 restantes son el punto central del cuadro delimitador (x,y), su anchura y su altura (w,h). El número de clases nc se computa en formato OneHot. Como por cada celda se hace más de 1 predicción, el número total de salidas por cada celda es el de la Ecuación 5.20.

$$\text{Pred} = \text{p} * (5 + nc) \tag{5.20}$$

El número de predicciones que se realizan por cada celda es p. En anteriores versiones de YOLO se utilizaban las llamadas *'cajas de anclaje'*, que son cuadros delimitadores de tamaño predefinido que ayudan a ajustar las salidas de los cuadros delimitadores predichos por el modelo al cuadro delimitador real. No obstante el modelo YOLOv8 es capaz de predecir los cuadros delimitadores de los objetos sin necesidad de cajas de anclaje, obteniendo mejores resultados que versiones anteriores que usan cajas de anclaje.

Como por cada celda en cada head se hacen 3 predicciones (este parámetro puede variar de un modelo a otro), es muy probable que en muchas ocasiones se estén haciendo muchas predicciones sobre el mismo objeto. Para ello, se creó el algoritmo *non max-supression* (Sección 5.1.7), que es capaz de desechar predicciones sobre el mismo objeto.

Con respecto a las funciones de pérdida, hay 2: una función de pérdida que se usa para calcular el error cometido en la predicción de los cuadros delimitadores y otra función de pérdida que computa el error cometido al predecir la clase. Normalmente ambos errores tienen el mismo peso a la hora de actualizar los pesos de la red con el algoritmo del descenso del gradiente, aunque es posible que tengan distinta importancia dependiendo del problema y de los datos.

IOU

La medida *Intersection Over Union* (IOU), consiste en una medida de similitud entre 2 cuadros delimitadores, y su fórmula es la presentada en la Ecuación 5.21. En caso de que los cuadros delimitadores sean coincidentes, el resultado de IOU será 1, y si son disjuntas, será 0. Es decir, que la salida de la medida IOU está contenida en el intervalo [0,1], siendo 0 disjuntas y 1 coincidentes.

$$\text{IOU}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.21}$$

Non max-supression

El algoritmo *non max-supression* consiste en descartar aquellas predicciones que se refieran a los mismos objetos. Es un algoritmo iterativo, donde inicialmente en el conjunto B están todas los cuadros delimitadores predichos de todas las head, y el conjunto D está vacío. En todo momento los conjuntos B y D serán conjuntos disjuntos.

Iterativamente, se selecciona el cuadro delimitador que tenga mayor probabilidad de pertenenecia a una clase, incluyéndolo en el conjunto D y eliminándolo del conjunto B. A continuación se calcula la medida IOU (Sección 5.1.7) de ese cuadro delimitador con los restantes del conjunto B. Si el resultado de la medida IOU es mayor que un umbral predefinido, se considera que ambos cuadros delimitadores hacen referencia al mismo objeto, y por tanto se desecha el cuadro delimitador de menor probabilidad, eliminándolo del conjunto B. Una vez se ha calculado el IOU con todos los cuadros delimitadores restantes en B, se vuelve a empezar el proceso seleccionando la siguiente predicción de mayor probabilidad en B. El proceso termina cuando el conjunto B está vacío. En el conjunto D estarán todas las predicciones reales y sin repeticiones.

5.2. Implementación

En las prácticas curriculares previamente realizadas a este TFG se entrenó un modelo YOLOv5 y un modelo YOLOv8. No obstante, se entrenaron con imágenes de muy baja calidad y no se obtuvieron muy buenos resultados. En este TFG se ha hecho un reentrenamiento completo del modelo YOLOv8 con imágenes de mayor calidad. Además, se ha hecho una optimización de hiperparámetros del modelo a la hora de entrenar, así como una optimización de parámetros a la hora de inferir. Por otro lado se ha trabajado sobre el algoritmo de detección de trayectorias iniciado en las prácticas, modificando la búsqueda que realiza (búsqueda primero en anchura), así como las heurísticas utilizadas. También se implementa un módulo de obtención de longitud de trayectorias para filtrar falsos positivos.

En esta sección se va a explicar el proceso seguido para hacer la detección de aves con el modelo YOLOv8 explicado en la Sección 5.1.7. Cabe destacar que en este caso no hay ningún interés en que el modelo sea capaz de clasificar cada ave presente en una imagen. El objetivo del modelo es que identifique el mayor número de aves en la imagen, sin importar que el número de falsos positivos (predicciones como aves que en realidad no lo son) sea alto, ya que existe otro algoritmo que se encargará de hacer esa separación entre imágenes con aves y sin aves. Todas las implementaciones en este proyecto se han realizado en Python.

5.2.1. Preprocesamiento de las imágenes

Para el entrenamiento del modelo, lo primero es disponer de un dataset con imágenes significativas y que aporten información en relación al concepto objetivo. No solo es importante tener imágenes con aves para que el modelo sea capaz de identificarlas, sino también es necesario que haya imágenes sin aves para evitar que el modelo prediga un excesivo número de falsos positivos. Como la intención es identificar aves sin importar las condiciones ambientales, con la cámara del proyecto se han capturado imágenes en distintas condiciones lumínicas y metereológicas. En la Sección 4.1 se explica el proceso de adquisición y segmentación de los datos.

El modelo de YOLOv8 que se va a usar es el modelo *'n'* de Ultralytics [3], que es el más pequeño de todos los disponibles. Esto se debe a que el proyecto se encuentra en las fases iniciales y no se dispone de suficientes imágenes como para ser capaces de entrenar modelos más grandes. Además, si se escoge un modelo excesivamente grande, se corre el riesgo de sufrir el problema de la degradación, explicado en la Sección 5.1.4. En la Tabla 5.1 se ve un resumen de los modelos YOLOv8 disponibles hasta la fecha. 'FLOPs' se refiere a el número de operaciones en punto flotante realizadas para cada procesamiento de 1 imagen. 'Parámetros' se refiere al número de parámetros que el modelo tiene que aprender.

Tabla 5.1: Distintos modelos YOLOv8.

Modelo YOLOv8	Tamaño (píxeles)	Parámetros (millones)	FLOPs (Billones)
YOLOv8n	640	3.2	8.7
YOLOv8s	640	11.2	28.6
YOLOv8m	640	25.9	78.9
YOLOv8l	640	43.7	165.2
YOLOv8x	640	68.2	257.8

Como se aprecia en la Tabla 5.1, todos los modelos requieren que las imágenes tengan 640x640 píxeles, por tanto, en el preprocesamiento de las imágenes, se realizó esa modificación. No fue necesario realizar más preprocesamiento como la normalización de las imágenes, ya que el modelo lo hace en cada uno de los módulos Conv (Sección 5.1.7) en las capas de normalización. Tampoco fue necesario utilizar técnicas de aumentación de datos ya que el propio modelo de Ultralytics tiene implementada esa funcionalidad.

La separación de los datos en conjuntos de entrenamiento-prueba-evaluación se hizo en este punto. Se decidió que los porcentajes serían 80 % ,10 % y 10 % respectivamente. Evidentemente, todos estos conjuntos son disjuntos, para asegurarse de utilizar un método honesto. Se utiliza el método de selección de conjuntos *hold-out* por una serie de razones, pero la principal es que se entrena un solo modelo. En caso de utilizar métodos donde se entrene más de un modelo, el coste temporal y computacional sería enorme, ya que los modelos YOLOv8 son modelos muy grandes con muchos parámetros a aprender.

5.2.2. Creación del fichero *.yaml*

En la documentación de Ultralytics del modelo YOLOv8, se explica que a la hora de entrenar, es necesario dar como parámetro un fichero con extensión *.yaml* al modelo para su correcto entrenamiento. En dicho fichero, se incluyen las rutas donde se encuentran las imágenes de entrenamiento-validación-test, así como el número de clases y su nombre. En la Figura 5.7 se muestra el contenido del fichero *.yaml* utilizado en este proyecto.

```
train: ..\\train_data_birds\\images\\train
val: ..\\train_data_birds\\images\\val
test: ..\\train_data_birds\\images\\test

nc: 1

names: ['bird']
```

Figura 5.7: Fichero *.yaml* usado para entrenar el modelo YOLOv8. Elaboración propia.

En las rutas mostradas en la Figura 5.7 se encuentran las imágenes, no obstante los

ficheros .txt resultantes de la segmentación de las imágenes explicadas en la Sección 4.1 se encuentran en las rutas relativas:

- ..\train_data_brids\labels\train
- ..\train_data_brids\labels\val
- ..\train_data_brids\labels\test

5.2.3. Creación del entorno python

Para crear el entorno de ejecución Python se utilizó la herramienta *venv* de Python. Entre las bibliotecas y librerías instaladas, cabe destacar la instalación de *cudnn*, que es una biblioteca para hacer uso de tarjetas gráficas de NVIDIA (GPU) para acelerar el entrenamiento. Hubo que instalar también la biblioteca de Ultralytics para tener acceso a su modelo. De la misma manera, el modelo de Ultralytics funciona por debajo con la librería *pytorch*, que también hubo que instalarla.

5.2.4. Entrenamiento del modelo

Para entrenar el modelo hay una serie de hiperparámetros a tener en cuenta:

- **Patience:** Número de épocas a esperar sin que mejoren las métricas de resultados sobre el conjunto de validación antes de acabar el entrenamiento. Por defecto este valor es 100, aunque debido al bajo cómputo disponible en este proyecto, se puso a 60.
- **Epochs:** Es el número de épocas que hará como máximo el modelo. Como se le ha dado un conjunto de validación y un valor en patience, se puede poner en este parámetro un valor muy alto, ya que en caso de empezar a sobreajustar, finaliza el entrenamiento. Se entrenó el modelo con este parámetro con un valor de 2.000.
- **Batch:** Es el tamaño del lote, que en este caso se eligió el valor 4.
- **Freeze:** Este parámetro se refiere al número de capas empezando por las primeras que no van a ser entrenadas. Como inicialmente al modelo se le cargan los pesos de un modelo YOLOv8 entrenado con el dataset COCO [30], los pesos de las primeras capas puede ser que no cambien mucho tras el entrenamiento. Al fin y al cabo, las primeras capas se encargan de hacer la extracción de características, y dependiendo del dataset, esa extracción de características puede ser igual en ambos dominios. El dataset COCO está formado por clases muy distintas entre sí, y como la finalidad del modelo es identificar solo aves, se decidió que lo mejor era poner este parámetro a 0 para entrenar todas las capas y hacer un entrenamiento completo.
- **Close_mosaic:** El modelo crea un mosaico a partir de las imágenes de entrenamiento. Esto se debe a que se ha comprobado experimentalmente que esta técnica puede llevar

a una mejor generalización ya que en muchas ocasiones estos mosaicos introducen ruido, evitando el sobreajuste. No obstante, en las últimas épocas no se debe usar esta técnica, y es por eso que este parámetro se asegura que las n últimas épocas no se use esta técnica. Ultralytics recomienda que este parámetro tenga valor 10, así que no se modifica.

El proceso de entrenamiento fue un proceso que gracias al uso de una GPU NVIDIA GeFORCE GTX 1660 TI tardó poco tiempo, alrededor de 2,5 horas. Se completaron un total de 464 épocas, donde cada época tardó alrededor de 19 segundos. Se hizo una prueba que consistió en entrenar el mismo modelo con los mismos hiperparámetros en una CPU Intel i7-1165G7. Los resultados fueron que la CPU tardaría 54,1 horas (7 minutos por época), casi 22 veces más que usando la GPU. Este es un claro ejemplo de la gran utilidad del uso de GPUs en el dominio del aprendizaje automático.

Estas fueron todas las funcionalidades y tareas realizadas en el primer prototipo. A partir de aquí se usa en posteriores prototipos el modelo YOLOv8 entrenado para la identificación de aves.

5.2.5. Detección de trayectorias

Como la tubería de procesamiento de un vídeo es procesarlo fotograma a fotograma, es necesario poder identificar la misma ave en fotogramas consecutivos, y saber que se trata de la misma ave. Esto es importante porque a la hora de obtener el recorte de cada ave, como es posible que se visualicen muchas aves, se obtengan muchos recortes de aves y haya una sobrecarga de imágenes. Además, no se quieren capturar todos los recortes de un ave en todos los fotogramas que aparece, sino solo unos pocos para evitar la sobrecarga.

El primer algoritmo que se ideó para implementar esta funcionalidad es uno basado en la medida IOU. La idea fundamental es comparar la medida IOU del cuadro delimitador obtenido en el procesamiento del fotograma n con los cuadros delimitadores obtenidos en el procesamiento del fotograma *n-1*. Se considera que el cuadro delimitador del fotograma n pertenece a la trayectoria del cuadro delimitador del fotograma *n-1* que mayor medida IOU tenga. En la Figura 5.8 se presenta la salida del anterior algoritmo con los vídeos capturados por la cámara. A cada trayectoria se le asigna un color distinto para facilitar la visualización.

Tal y como se muestra en la Figura 5.8, ahora es posible distinguir unas aves de otras bajo circunstancias normales. No obstante, hay circunstancias para las que este algoritmo no funciona bien:

- Dos aves se cruzan en el mismo instante de tiempo.
- No se detecta un ave en un fotograma.

Figura 5.8: Salida del algoritmo inicial de detección de trayectorias. Elaboración propia.

Para solucionar los problemas anteriores se hacen una serie de modificaciones en el algoritmo.

5.2.6. Primera mejora del algoritmo de detección de trayectorias.

La primera mejora que se implementa es la modificación de parámetros en el uso del modelo explicado en la Sección 5.2.4.

- El primer parámetro a modificar es la **confianza**. Este parámetro se refiere a la confianza que tiene el modelo en que la predicción sea correcta. Como ya se ha explicado en secciones anteriores, a la hora de clasificar la especie de ave, habrá un modelo que discernirá entre aves y no-aves. Por eso no importa que haya muchos falsos positivos (predicciones como aves que en realidad no lo son). Debido a esto, la confianza se puede poner a un valor muy bajo. En este caso se puso a 0.01. La consecuencia de esto es que se van a producir muchas predicciones (es lo que se está buscando). No obstante se van a producir muchas predicciones sobre las mismas aves, de forma que va a dificultar la identificación de trayectorias y puede empeorar el funcionamiento del algoritmo drásticamente. Por ello se modifica el siguiente parámetro.

- El segundo parámetro que se modifica es **iou**. Este parámetro es el umbral que se va a utilizar en el algoritmo non max-supression (Sección 5.1.7) del modelo YOLOv8. Se pone su valor cercano a 0, y se asegura que no se hagan muchas predicciones sobre el mismo objeto.

Con la modificación de los parámetros anteriores se consigue que el modelo haga muchas predicciones, pero de aves distintas. Esto soluciona parcialmente ambos problemas, pero los problemas persisten, ya que el modelo no es capaz de identificar en todos los fotogramas todas las aves.

5.2.7. Segunda modificación del algoritmo de detección de trayectorias.

Para solucionar el problema de que un ave no sea detectado en un fotograma, se crea una heurística. Inicialmente, en caso de detectar un ave en el fotograma *n* y su IOU con los cuadros delimitadores del fotograma *n-1* sea con todos 0 (significa que no se ha detectado el ave en el fotograma *n-1*), se asigna esa ave a la trayectoria más cercana. Aunque esto a veces solucionaba el problema, es una heurística muy genérica que falla mucho. Por eso, partiendo de esta idea, se implementaron una serie de modificaciones. Hasta aquí llega la implementación del segundo prototipo. Las siguientes modificaciones se realizan sobre este prototipo, dando lugar al tercer prototipo.

Se calculó la dirección (vectorialmente) que seguía la trayectoria del ave en el fotograma *n-1*, y se asignaba al cuadro delimitador obtenido en el fotograma *n* que mejor siguiera esa trayectoria. No obstante, esta modificación no mejoró los resultados anteriores, principalmente porque las trayectorias seguidas por las aves no son perfectas y pueden hacer cambios de direcciones, de forma que la dirección en el instante *n-1* no sea la misma que en el instante *n*.

Dado que la anterior modificación no funcionó, se implementó otra. Esta consiste en obtener todas las medidas IOU con todas las predicciones de todos los fotogramas (empezando por el fotograma *n-1* y acabando en el fotograma 0), de forma que si se obtiene un IOU mayor que 0, se considera que ese cuadro delimitador pertenece a esa trayectoria. En la Figura 5.9a se ve un ejemplo de una imagen donde las aves no son detectadas en todos los fotogramas, y en los siguientes fotogramas que se detectan esas mismas ave, se consideran que pertenecen a nuevas trayectorias y, por lo tanto, son aves distintas (a cada trayectoria se le asigna un color distinto). En la Figura 5.9b se muestra esa misma imagen, pero usando la modificación explicada. Se puede pensar que este algoritmo puede presentar el problema de que cuando dos aves se cruzan en distinto instante de tiempo no funcione bien y los colores de las trayectorias no se mantengan. No obstante esto no ocurre, ya que se realiza una búsqueda primero en anchura empezando por los fotogramas del final. En la Figura 5.10 se ve un ejemplo de este caso, con resultados positivos. El ave con trayectoria rosa cruza las trayectorias de color amarillo y azul. De la misma manera, la trayectoria azul se cruza con la trayectoria verde sin presentar el problema comentado.

(a) Salida del algoritmo cuando hay aves no detectadas en un fotograma sin la modificación. Elaboración propia.

(b) Salida del algoritmo cuando hay aves no detectadas en un fotograma con la modificación. Elaboración propia.

Figura 5.9: Aves no detectadas en un fotograma.

Figura 5.10: Cruce de aves en distinto instante de tiempo. Elaboración propia.

No obstante, esto solo ocurre si las aves se cruzan en instantes de tiempo distintos. Las fases que ocurren cuando 2 o más aves se cruzan en el mismo instante de tiempo son las siguientes:

1. Antes de que se crucen, hay tantas trayectorias como aves, es decir, si hay n aves hay n trayectorias.
2. Las aves se cruzan en el mismo instante. En este punto el algoritmo solo detecta 1 ave ya que estas se solapan, por tanto solo hay 1 trayectoria.
3. Las aves se vuelven a separar. Como en el punto anterior solo había una trayectoria, el algoritmo anterior asigna estas *n-1* nuevas trayectorias a la única trayectoria cercana, la del fotograma anterior.

Como conclusión, cuando 2 o más aves se cruzan en el mismo instante de tiempo, asigna todas ellas a la misma trayectoria. No obstante, esto no es problema del algoritmo desarrollado, sino del modelo que no es capaz de identificar a todas las aves solapadas como aves distintas. Por ahora no se ha encontrado solución a este problema, y permanece como trabajo futuro el poder resolverlo. Una posible solución puede ser utilizar las trayectorias de las aves como series temporales e intentar predecir su siguiente posición. En la Figura 5.11a se visualiza el momento en el que dos aves se solapan (trayectoria verde y trayectoria azul), y en la Figura 5.11b se visualiza el momento en el que ambas aves se dejan de solapar y ambas tienen el mismo color de trayectoria (verde).

(a) Aves en el instante donde se solapan. Elaboración propia.

(b) Aves tras cruzarse en el mismo instante. Elaboración propia.

Figura 5.11: Aves cruzándose en el mismo instante de tiempo.

5.2.8. Longitud de trayectorias

La última de las mejoras realizadas con respecto al algoritmo de detección de trayectorias fue establecer un mínimo de longitud de una trayectoria para considerarla correcta.

Se ha hecho un experimento donde incialmente se eligieron 3 posibles valores: 5, 8 y 16. Tras procesar los vídeos y observar los resultados, se ha determinado que el umbral que mejor funciona es 16. Como los parámetros **iou** y **conf** (Sección 5.2.6) del modelo se han configurado para realizar muchas predicciones sobre objetos distintos, es muy probable que se obtengan muchas trayectorias de poca longitud que sean falsos positivos. Al establecer el umbral en 16, se consigue eliminar gran cantidad de esas trayectorias que son falsos positivos. Aunque es cierto que puede que se pierdan trayectorias de aves reales, esas trayectorias que se pierden son de escasa longitud que se encuentran en los bordes de la imagen y, por tanto, la que menos información ofrece. Por tanto, la información que se descarta es la menos relevante.

5.2.9. Obtención de imágenes con trayectorias

Uno de los objetivos iniciales del proyecto era obtener imágenes con las trayectorias de las aves obtenidas. Se implementó dicha funcionalidad, y en la Figura 5.12 se muestra un

ejemplo. De la misma manera, se ha implementado la funcionalidad de realizar los recortes de las aves detectadas.

Figura 5.12: Imagen con todas las trayectorias de las aves. Elaboración propia.

Estas son todas las funcionalidades desarrolladas en el dominio de segmentación de imágenes. Se ha requerido de 3 prototipos incrementales para implementar el módulo final, y alrededor de 222 horas de desarrollo.

Capítulo 6

Clasificación de aves por imagen

En este capítulo se van a definir los dos modelos usados para clasificar a las aves. Primero se desarrollará una base teórica y a continuación se definirán las implementaciones. Los datos usados para implementar los modelos se explican en la Sección 4.2.

6.1. Base teórica

6.1.1. EfficientNet

Se ha demostrado que las EfficientNet son hasta 7 veces más pequeñas y 6 veces más rápidas que otros modelos CNN, teniendo métricas mejores que redes más grandes y complejas. Este es un ejemplo claro del problema de la degradación (explicado en la Sección 5.1.4).

Aparte de ser más eficientes y rápidas, solucionan el problema del escalado de redes (aumentar el tamño de una red), traduciéndose en mejores resultados. Para entender cómo estas redes solucionan dicho problema, es importante entender cuáles son las dimensiones de una CNN:

- **Resolución:** Resolución de entrada de las imágenes (número de píxeles de alto y de ancho).
- **Profundidad:** El número de capas que presenta la red.
- **Anchura:** El número de características de salida que tienen las capas de la red.

Tradicionalmente, cuando se quería aumentar el tamaño de una red, solo se modificaba 1 de esas dimensiones (resolución, profundidad o anchura). El desbalance de las dimensiones presentan una serie de problemas:

- Aumentar solo la profundidad da lugar a la aparición del problema de la degradación, como se ha explicado en la Sección 5.1.4.
- Si se aumenta la anchura, solo se obtienen mejores resultados para modelos pequeños, pero estos buenos resultados no escalan a modelos de mayor tamaño y complejidad.
- Si solo se aumenta la resolución de las imágenes de entrada, el concepto puede ser demasiado complejo como para que la red lo aprenda [17].

Por eso se tienen que escalar las 3 dimensiones y, además, tienen que estar balanceadas entre sí. Tal y como se explica en el artículo donde se presentan las EfficientNet [17], se definen unos parámetros α, β, γ (profundidad, anchura y resolución respectivamente) que se usan para balancear las 3 dimensiones. Para ello, dichos parámetros tienen que cumplir la condición de la Ecuación 6.1, y además todos ellos tienen que ser mayores o iguales que 1. Estos parámetros se obtienen haciendo una búsqueda en un modelo inicialmente pequeño.

$$\alpha * \beta^2 * \gamma^2 \approx 2 \tag{6.1}$$

En el artículo [17] se define que los tamaños ideales son:

$$\begin{aligned} \alpha &= 1,2 \\ \beta &= 1{,}1 \\ \gamma &= 1{,}15 \end{aligned} \tag{6.2}$$

φ es un coeficiente definido por el usuario que controla el escalado de la red. Si se quieren usar 2^N recursos computacionales más, entonces $\varphi = N$, y el escalado quedaría:

$$\begin{aligned} \mathrm{d} &= \alpha^{\varphi} \\ \mathrm{w} &= \beta^{\varphi} \\ \mathrm{r} &= \gamma^{\varphi} \end{aligned} \tag{6.3}$$

donde *d, w* y *r* son las proporciones reales de reescalado de la profundidad, anchura y resolución respectivamente.

En la biblioteca *Keras*, se pueden usar los modelos EfficientNetB0 - EfficientNetB7, donde se han seguido las fórmulas definidas en 6.2 y 6.3. En la Tabla 6.1 se muestra la resolución de entrada de los distintos modelos, resultado de las ecuaciones 6.3 y 6.2. Todos los modelos EfficientNet tienen la misma estructura, lo que les diferencia es el número de filtros, los tamaños de las características y el número de capas. En la Tabla 6.2 se muestra la estructura del modelo EfficientNetB0. Cabe destacar que en la Tabla 6.2, la columna 'Operador' no varía para los distintos modelos EfficientNet, lo que cambia son las columnas 'Resolución', 'Canales' y 'Número de capas', según los parámetros *d, w* y *r*.

Tabla 6.1: Tamaño de entrada de las imágenes en los modelos EfficientNet.

Modelo EfficientNet	Tamaño de entrada (píxeles)
EfficientNetB0	224
EfficientNetB1	240
EfficientNetB2	260
EfficientNetB3	300
EfficientNetB4	380
EfficientNetB5	456
EfficientNetB6	528
EfficientNetB7	600

Tabla 6.2: Estructura del modelo EfficientNetB0.

Orden	Operador	Resolución	Canales	Número de capas
1	Conv3x3	224x224	32	1
2	MBConv1, k3x3	112x112	16	1
3	MBConv1, k3x3	112x112	24	2
4	MBConv1, k5x5	56x56	40	2
5	MBConv1, k3x3	28x28	80	3
6	MBConv1, k5x5	14x14	112	3
7	MBConv1, k5x5	14x14	192	4
8	MBConv1, k3x3	7x7	320	1
9	Conv1x1 & Pooling & FC	7x7	1280	1

En la Tabla 6.3 se muestra el número de canales de entrada de cada capa de cada versión

Tabla 6.3: Anchura de los modelos EfficientNet.

Orden	B1	B2	B3	B4	B5	B6	B7
1	32	32	40	48	48	56	64
2	16	16	24	24	24	32	32
3	24	24	32	32	40	40	48
4	40	48	48	56	64	72	80
5	80	88	96	112	128	144	160
6	112	120	136	160	176	200	224
7	192	208	232	272	304	344	384
8	320	352	384	448	512	576	640
9	1280	1408	1536	1792	2048	2304	2560

El número de FLOPs requeridos por la red es proporcional a d, w^2 y r^2. Esto significa que multiplicar por 2 la profundidad de la red dobla el número de FLOPs requeridos, pero multiplicar por 2 la resolución o la anchura requiere 4 veces más FLOPs. Otra ventaja que tiene hacer el escalado proporcional de todas las dimensiones de la red es que la red se puede enfocar en las regiones más relevantes donde más objetos hay (y por tanto más información) y realizar una mejor clasificación.

6.1.2. MobileNet

El modelo EfficientNet usa una red llamada MobileNet [31], tal y como se observa en la Tabla 6.2 (operador MBConv1). Como su nombre indica, es una red que es ampliamente usada en dispositivos móviles donde la capacidad de cómputo es muy limitada. Su objetivo principal es disminuir drásticamente el número de parámetros y el número de FLOPs necesarios para realizar predicciones. Se ha demostrado que son redes muy pequeñas con bajo gasto computacional, pero con unas métricas buenas [32]. En la Tabla 6.4 se define la estructura de las MobileNets.

Tabla 6.4: Estructura del modelo MobileNet.

Operador	Resolución	Expansión	Número de Canales	Número de capas	Stride
Conv2D k3x3	224x224	1	32	1	2
Bottleneck	112x112	6	16	1	1
Bottleneck	56x56	6	32	3	2
Bottleneck	28x28	6	64	4	2
Bottleneck	14x14	6	96	3	1
Bottleneck	14x14	6	160	3	2
Bottleneck	7x7	6	320	1	1
Conv2D k1x1	7x7	-	1280	1	1
AvgPool k7x7	7x7	-	-	1	-
Conv2D k1x1	1x1	-	k	-	-

Cabe destacar que en la estructura de las MobileNet presentada en la Tabla 6.4, se usa mayoritariamente un bloque llamado *bottleneck*. Es en estas capas donde las MobileNet son capaces de reducir los parámetros de aprendizaje.

A estos bloques se les llama *cuellos de botella lineales*, y funcionan como un cuello de botella explicado en la Sección 5.3 pero de forma inversa [33]. En este caso, primero hay una capa que reduce el número de características haciendo convoluciones con kernels de tamaño 1x1 (pocas convoluciones pero con muchas operaciones). A continuación hay una capa que hace una convolución 3x3 que aumenta el número de características, y por último una capa de nuevo con kernels 1x1 que reduce el número de características a las que había al inicio del bloque. Además este bloque es residual (se suma la entrada a la salida). En la Tabla 6.5 se muestra la estructura de este bloque.

Tabla 6.5: Estructura del bloque bottleneck de las MobileNets.

Orden	Operador
1	Conv2D k1x1
2	BatchNormalization
3	Activation (Relu6)
4	DepthWiseConv2D k3x3
5	BatchNormalization
6	Activation (Relu6)
7	Conv2D k1x1
8	BatchNormalization
9	Residual Connection

Cabe destacar que en la estructura presentada en la Tabla 6.5 se usa la función de activación *relu6*, que pertenece a la familia relu. En la Ecuación 6 se representa su función. Es la misma fórmula que la relu definida en la Ecuación 5.7, solo que en caso de que el valor sea mayor que 6, el resultado es el propio valor.

$$\text{Relu6}(x) = \text{mín}(\text{máx}(0, x), 6) \tag{6.4}$$

Por otra parte, en la Tabla 6.5 se presenta una operación desconocida hasta el momento, que es la *convolución en profundidad*. La idea de esta convolución es tener un filtro por cada canal de entrada. Se hace la convolución entre cada filtro y su correspondiente canal y se concatenan los resultados. En la Figura 6.1 se muestra un ejemplo de este tipo de convolución. Inicialmente la imagen de entrada con 3 canales (rojo, verde y azul) se separan, y cada uno de ellos se convoluciona con un filtro distinto. Finalmente, se concatenan todos los resultados de esas convoluciones.

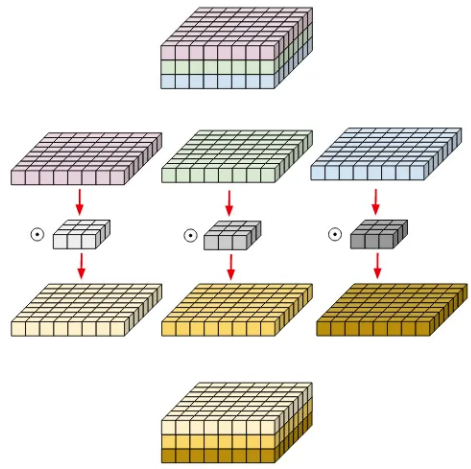


Figura 6.1: Convolución en profundidad. [34]

Otra posible operación de convolución que pueden usar las MobileNets son las *convoluciones separables*. La idea fundamental de esta convolución es hacer 2 convoluciones de 1 dimensión cada una, en vez de hacer 1 convolución de 2 dimensiones. Computacionalmente se realizan menos FLOPs, lo cual era la finalidad principal de las MobileNets, tener un bajo coste computacional.

Se puede unir la idea de *convolución en profundidad* con *convoluciones separables* para crear las *convoluciones separables en profundidad* [35]. La idea es hacer la convolución de cada característica de la imagen de entrada con sus correspondientes filtros de tamaño 3x3, y posteriormente, antes de concatenarlos, se hacen pasar por unos filtros de tamaño 1x1. De esta forma, si por ejemplo se tiene una entrada con 16 características de entrada y 32 de salida, primero se haría una convolución de 16 kernels de 3x3 y posteriormente 32 de tamaño 1x1. En ese caso, el número de operaciones de convolución sería (16*3*3) + (32*16*1*1) = 656. En caso de no utilizar este tipo de convolución y se usara la convolución tradicional, el número de convoluciones sería: 16*32*3*3=4608. Casi 8 veces más convoluciones que con la convolución separable en profundidad.

6.1.3. Visual Transformers

Otro modelo que a fecha de Junio de 2024 es muy utilizado para la clasificación en imágenes es Visual Transformer. Consiste en una modificación de los transformers [18] (usados inicialmente para series temporales y procesamiento de lenguaje natural). En este TFG no se va a explicar el funcionamiento de los transformer ya que se escapan del dominio de aplicación de este proyecto. No obstante, se procede a explicar el funcionamiento de las capas de Auto-Atención en el dominio de las imágenes, ya que es la operación que se usa en este modelo en sustitución de las convoluciones.

Auto-Atención

El mecanismo de Auto-Atención consiste en codificar cada píxel en función de su valor, su posición respecto al resto de los píxeles y el valor de los píxeles que le rodean. Realmente, este mecanismo es parecido a la operación de convolucion (con los filtros se tenían en cuenta los valores de los píxeles y sus posiciones espaciales). La mayor diferencia es que el mecanismo de Auto-Atención tiene más parámetros usados para la codificación de los píxeles, haciendo posible una mejor clasificación.

Inicialmente, la imagen presenta 3 valores por cada píxel (en caso de que la imagen se represente en RGB). El mecanismo de Auto-Atención puede hacer una expansión de dimensiones para poder tener un mayor dominio de representación. Esta expansión de dimensiones depende de la complejidad del concepto, tamaño de imágenes etc. Es un parámetro de la red.

Una vez que cada píxel está representado en n dimensiones, cada uno de ellos se hace pasar por 3 redes fully connected (la red tiene que aprender sus parámetros). Cada una de estas redes va a dar unas salidas que representan al píxel. La dimensionalidad de estas salidas

es la misma que la de las entradas, es decir, si cada píxel está representado por n valores, cada una de las 3 salidas tendrá n valores. Esas 3 salidas son:

- **Vector consulta:** Es un conjunto de valores que determinan la importancia que da un píxel a otros píxeles para que le representen.
- **Vector clave:** Conjunto de valores que determinan la importancia que puede ofrecer un píxel para representar a otros píxeles.
- **Vector valor:** Valores que representan el valor del propio píxel.

Una vez que se han calculado todos los vectores consulta, clave y valor de cada píxel, se calcula la similitud de un píxel i con el resto. La similitud consiste en usar el producto punto entre el vector consulta del píxel i con los vectores clave del resto de píxeles. La operación de la Ecuación 6.5 representa esta operación. El término K_{jk} representa el valor k del vector clave del píxel j. De la misma manera ocurre con Q_{jk}, donde Q_j es el vector consulta del píxel j.

$$\text{Similitud}(Q, K)(i) = \sum_{j=0}^{n} \sum_{k=0}^{m} Q_{ik} * K_{jk} \tag{6.5}$$

Tras la aplicación de la Ecuación 6.5 del píxel i con respecto a todos los píxeles (incluyéndose a sí mismo), se obtienen las similitudes. S_{ij} representa la similitud que hay entre el píxel i con el píxel j. Hay que destacar que los resultados no son simétricos, es decir, que S_{ij} es distinto de S_{ji}. Cada uno de esos valores S_{ij} se hace pasar por la función softmax, de forma que ahora esas similitudes se interpretan como valores de pertenencia a una distribución de probabilidad (todas están en el intervalo [0,1]). De la misma manera, se divide la similitud por la raíz de el número de valores totales que tienen los vectores [18]. Dicha fórmula se representa en la Ecuación 6.6. Es una manera de normalizar los resultados para tener en cuenta el número de píxeles sobre los que se ha calculado la atención.

Los resultados previos significan la importancia o la influencia que tiene cada uno de los píxeles j en el píxel i. Es decir, si $S_{ij} = 0$, significa que el píxel j no tiene ninguna importancia ni guarda ninguna relación con respecto a la codificación del píxel i. Sin embargo, si $S_{ij} = 1$, ocurre lo contrario, el píxel j tiene una gran influencia en la codificación real del píxel i.

La codificación final del píxel i será la presentada en la Ecuación 6.6. Cada una de esas similitudes entre los píxeles j con el píxel i se multiplican a todas las componentes de los vectores valor de los píxeles j, y posteriormente se suman todos.

$$\text{Atencion}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6.6}$$

La salida de la Ecuación 6.6 es la codificación final del píxel i, teniendo en cuenta su valor y los valores de los píxeles que lo rodean. Habría que hacer el proceso anterior para calcular todas las codificaciones de todos los píxeles.

Entre las muchas ventajas que ofrece este mecanismo, una de los principales es la paralelización. La parelización consiste en la capacidad de computar operaciones a la vez, sin que unas tengan que esperar al resultado de otras. Como los pesos de los perceptrones multicapa que obtienen los valores de los vectores clave, valor y consulta son los mismos para todos los píxeles, se pueden calcular todos los vectores a la vez. Incluso se puede calcular la codificación final de cada uno de los píxeles a la vez, pues no es necesario esperar a los resultados de otros píxeles. Esto supone una ventaja enorme, pues el mecanismo de atención requiere una gran cantidad de operaciones, pero gracias a la paralelización, todas esas operaciones se pueden computar a la vez. De nuevo, aquí encontramos la gran utilidad de las GPUs en el ámbito del aprendizaje automático, pues dichos dispositivos son capaces de paralelizar todas esas operaciones. Incluso es posible usar más de una GPU para acelerar todo el proceso de entrenamiento [36].

El mecanismo de atención tiene una complejidad computacional y temporal de $O(n^2)$, y esto en imágenes puede traer muchas complicaciones. Por ejemplo, dada una imagen de 224x224 píxeles, habría que computar el mecanismo de atención todos los píxeles con todos. El número de operaciones será $(224 * 224)^2 = 2.517.630.976$. Este valor es el número de veces que se computa todo el proceso explicado anteriormente. Además habría que computar todas esas iteraciones de atención por cada imagen que se procesa, y aunque es cierto que se puede paralelizar y acelerar ese proceso con GPUs, es demasiado caro temporal y computacionalmente como para poder utilizar este método.

Por eso, se divide la imagen en trozos denominados *parches*, de forma que los mecanismos de atención solo se aplicarán entre píxeles del mismo parche. Siguiendo con el ejemplo de la imagen de 224x224 píxeles, si esa imagen se divide en 8*8 parches, se obtendrán un total de 64 parches, cada uno con 28x28 píxeles. Aplicando el mecanismo de atención entre píxeles del mismo parche, el número de iteraciones de atención necesarias será $(28 * 28)^2 * 64 = 39.337.984$. El número de iteraciones de atención necesarias es 64 veces menos, son 2.483 millones de iteraciones menos.

Al hacer este proceso se ahorra una gran cantidad de iteraciones, pero se crea un nuevo problema. El mecanismo de atención no tiene en cuenta la posición espacial de los píxeles, simplemente tiene en cuenta los valores de los píxeles de alrededor (los del mismo parche en este caso) pero no se tiene en cuenta la posición de esos parches. Por ello, a cada parche se le va a añadir un valor llamado *incrustación espacial*. En el artículo donde se define el modelo Visual Transformer [37] se explica que la mejor opción a la hora de realizar esta incrustación espacial es sumar al parche la posición que ocupa. Para ello, lo que se hace es crear una matriz cuyas filas se suman a sus correspondientes parches antes de entrar al encoder (donde se encuentran los mecanismos de atención). De esta forma se tiene en cuenta el valor del propio píxel, los valores de los píxeles que le rodean y la posición que ocupa el parche.

Hay un parámetro más del Visual Transformer a tener en cuenta, y es que inicialmente los transformers se crearon con intención de aplicarse al procesamiento de lenguaje natural y la generación de texto, no para clasificación. Por ello, se añade un parámetro nuevo al

transformer, que es un parámetro aprendible, que es el CLS. Son siglas que significan ('Classification') en inglés, y sirve para que el transformer clasifique en vez de realizar regresiones (un transformer cuando hace problemas de procesamiento de lenguaje natural predice números que posteriormente se transforman en palabras). Este parámetro fue incialmente usado por el modelo BERT [38].

La entrada del encoder del Visual Transformer será el CLS, y un vector por cada parche, es decir, en el ejemplo anterior donde la imagen tiene 224x224 píxeles y se dividía en 8*8 parches, su entrada tendrá una dimensión de (1+8*8) = 65. Por cada una de esas entradas, se obtendrá una salida. No obstante, como es un problema de clasificación, solo interesa la salida de la entrada CLS, pues actúa como una especie de resumen de la imagen. La salida de la entrada CLS se pasa por un perceptrón multicapa que finalmente hace la clasificación.

En la Figura 6.2 se muestra la estructura de los Visual Transformer. Cabe destacar que antes de la capa de atención se hace una operación de normalización, y además hay una conexión residual a la salida de la capa de atención. A continuación se vuelve a normalizar la salida y un perceptrón multicapa con conexión residual clasifica las imágenes.

Existen posibles modificaciones que se pueden realizar a este Visual Transformer, especialmente en la parte de la incrustación posicional. En sustitución del aplanado de los parches para sumarles la fila correspondiente de la matriz de la incrustación posicional, se pueden usar operaciones convolucionales (intentando aumentar la expresividad del modelo). No obstante, a fecha de Junio de 2024 esto todavía no ha producido resultados mejores.

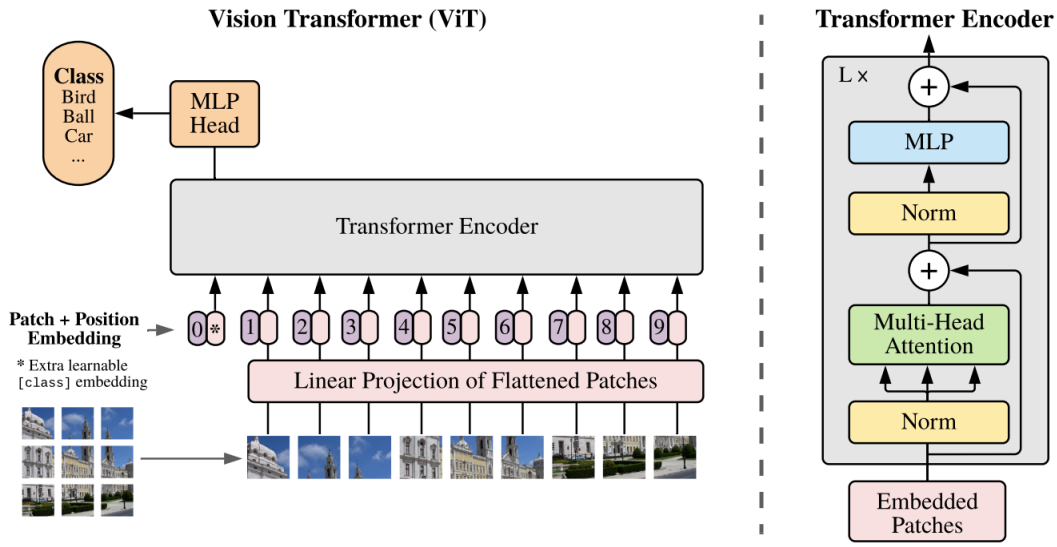


Figura 6.2: Estructura de un Visual Transformer. [37]

6.1.4. Aumentación de datos

Aumentación de datos es un proceso de preprocesamiento de imágenes que sirve para obtener nuevas imágenes a partir de las ya existentes. No obstante es necesario que estén modificadas de forma que sean útiles para el aprendizaje del modelo. Lo que hacen estas imágenes es introducir ruido en los datos, de forma que ayudan a paliar el problema del sobreajuste. Además ofrecen nueva información útil para generalizar el concepto [39].

Entre las modificaciones que se pueden hacer a las imágenes se encuentran [15]:

- **Transformaciones geométricas:** Consisten en modificar las imágenes modificando las características espaciales de las imágenes. Cabe destacar:
  - **Rotación:** Se rota la imagen unos grados definidos.
  - **Voltear:** Se trata de voltear la imagen (eje vertical u horizontal).
  - **Zoom:** Consiste en eliminar las zonas exteriores de la imagen ampliando las del centro.
  - **Traslación:** Consiste en trasladar la imagen para enseñar al modelo que no importa la posición del objeto, sino el objeto en sí (aprender patrones y no memorizar posiciones).
- **Transformaciones de color:** Consiste en modificar características de los colores de la imagen.
  - **Brillo:** Modifica el brillo para simular distintas condiciones lumínicas.
  - **Contraste:** Modificar el contraste ayuda a aprender el concepto bajo distintas condiciones de claridad.
  - **Saturación:** Permite al modelo clasificar las imágenes bajo distintas intensidades de color.

(a) Imagen original. [40]

(b) Imagen rotada. Elaboración propia.

Figura 6.3: Rotación de imágenes.

(a) Imagen original. [40]

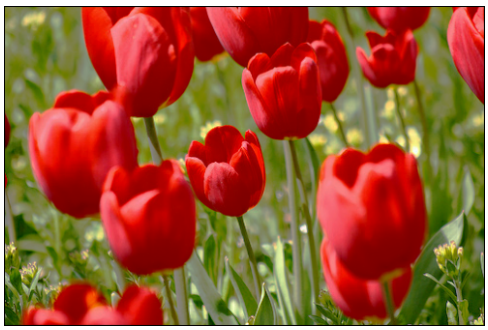
(b) Imagen con zoom. Elaboración propia.

Figura 6.4: Zoom de imágenes.

(a) Imagen con brillo disminuido. Elaboración propia.

(b) Imagen con brillo aumentado. Elaboración propia.

Figura 6.5: Modificación de brillo en imágenes.



(a) Imagen con contraste disminuido. Elaboración propia.

(b) Imagen con contraste aumentado. Elaboración propia.

Figura 6.6: Modificación de contraste en imágenes.

(a) Imagen con saturación disminuida. Elaboración propia.

(b) Imagen con saturación aumentada. Elaboración propia.

Figura 6.7: Modificación de saturación en imágenes.

(a) Imagen original. [40]

(b) Imagen volteada en eje y. Elaboración propia.

Figura 6.8: Volteo de imágenes.

Existe la posibilidad de hacer este tipo de modificaciones solapadas, es decir en una misma imagen se puede modificar más de una característica. De esta manera se obtienen imágenes (a partir de las ya existentes) significativamente distintas que aportan nueva información respecto al concepto.

6.2. Implementación

6.2.1. Preprocesamiento

El preprocesamiento realizado en los datos fue hacer aumentación de datos sobre las clases minoritarias y posteriormente normalizarlos. De esta manera el desbalance entre las clases es menor, evitando que siempre se prediga la clase mayoritaria. Hay varias formas de hacer

aumentación de datos. Una manera es hacerlo en tiempo de ejecución, de forma que a un lote antes de ser dado como entrada a la red se le hace transformaciones como las explicadas en la Sección 6.1.4. La desventaja de este método es que el experimento no es reproducible, ya que las modificaciones que se le hacen a cada lote son pseudoaleatorias. La otra técnica es hacer esas modificaciones a las imágenes, y guardarlas en almacenamiento persistente. La ventaja que tiene este método es que el experimento es reproducible, y además se pueden normalizar las imágenes. Como en este proyecto se tiene intención de probar dos modelos distintos, se implementó el segundo método.

Para hacer la carga de imágenes de almacenamiento persistente a memoria RAM, se implementaron Data Loaders que cargan las imágenes en tiempo de ejecución. Es decir se tiene un array con todas las direcciones de memoria donde están guardadas esas imágenes, y solo se cargan los datos de las imágenes que van a procesarse en ese lote. Esto se hace ya que cuando se tiene un dataset demasiado grande, no hay memoria suficiente para cargar en memoria RAM todas las imágenes. En este caso, a pesar de que el dataset tuviera 28.987 + las imágenes obtenidas por la aplicación de aumentación de datos, como estas tenían poco tamaño (4 KB cada imágen) se podían cargar todas en memoria, pues se tiene 32 GB de memoria RAM. No obstante, hacer la carga de imágenes en tiempo de ejecución es una buena práctica que puede acelerar el proceso de entrenamiento y, además, es código que se puede reutilizar en caso de modificar el dataset.

Por otro lado el Data Loader tiene otra función, y es que antes de empezar el procesamiento de una nueva época, cambia el orden de las imágenes en los lotes. Esto se hace para que el orden de las imágenes no sea el mismo y, por tanto, se evita el aprendizaje memorístico, en el que el modelo no aprende el concepto, sino el orden de los datos.

Además, tal y como se detalla en la Tabla 6.1, dependiendo del modelo EfficientNet usado, las dimensiones de las entradas deben ser unas u otras. En este caso, como se usa el modelo más pequeño, EfficientNetB0, se escalan las imágenes a tamaño 224x224.

6.2.2. Clasificador de 4 clases

Inicialmente, se implementó un modelo basándose en las EfficientNet de Keras para que clasificara las imágenes del dataset explicado en la Sección 4.2. Se hicieron modificaciones en las últimas capas del modelo para adapatarlo a los datos de entrada. La matriz de confusión obtenida por este modelo es la presentada en la Figura 6.9, en la cual la clase 0 corresponde a otros tipos de aves, 1 a cuervo, 2 a halcón y 3 a no ave. En dicha figura, se observa que la red es capaz de diferenciar muy bien las imágenes que son aves de las que no. No obstante, no es capaz de discernir perfectamente entre las imágenes que muestran algún tipo de ave. En la Figura 6.10 se ve tanto la tasa de acierto como la función de pérdida del modelo. En dichas figuras se puede ver el gran sobreajuste que se produce, a pesar de obtener una tasa de acierto de 0.9051 sobre el conjunto de test.

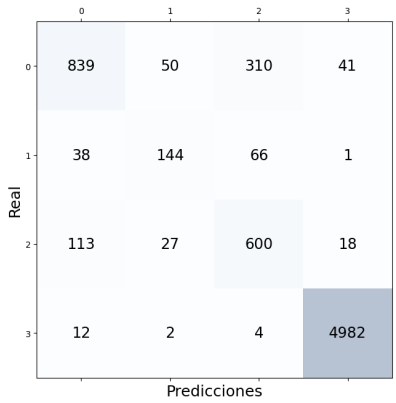
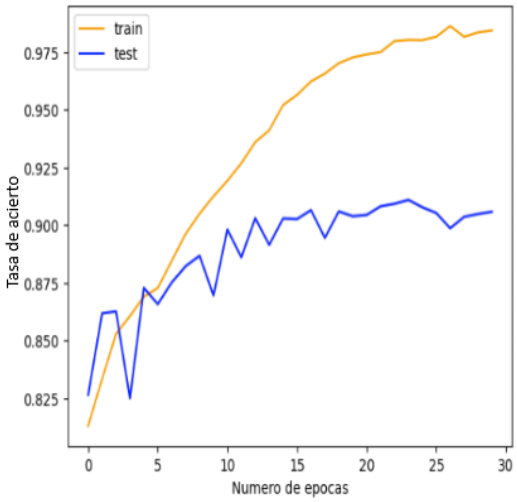
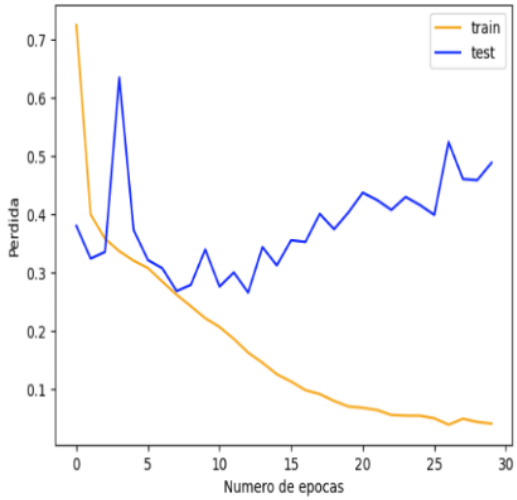


Figura 6.9: Matriz de confusión del modelo EfficientNet. Elaboración propia.



(a) Tasa de acierto de EfficientNetB0. Elaboración propia.



(b) Función de pérdida de EfficientNetB0. Elaboración propia.

Figura 6.10: Salida del modelo EfficientNetB0.

6.2.3. Clasificador ensemble

A la vista de los resultados anteriores, se sacan las siguientes conclusiones.

- Las clases siguen estando muy descompensadas (a pesar de la aumentación de datos) y puede ser una de las razones de que en muchas ocasiones se prediga la clase mayoritaria (no-ave).

- Las imágenes pueden tener una resolución demasiado pobre como para diferenciar unas aves de otras.

Por ello, se propuso hacer dos modelos: el primero de ellos se encargará únicamente de realizar clasificación binaria (discernir si en la imagen hay un ave o no). Las imágenes que ese clasificador identifique como aves, se procesaran en otro clasificador que tratará de discernir el tipo de ave contenido en la imagen. Ambos clasificadores son modelos EfficientNetB0.

Dado que se van a hacer 2 clasificadores, hay que definir cuidadosamente los conjuntos de entrenamiento y test de cada uno de ellos para que no se solapen (sean disjuntos) y sea un método honesto. Inicialmente, las imágenes se separan en 2 conjuntos donde el 80 % se utilizará para entrenar y el 20 % para test. Esa división se hace estratificada por clases, es decir que la distribución de clases es aproximadamente la misma en ambos conjuntos.

A continuación, de esos conjuntos, se reserva un 35 % para el clasificador binario y el restante 65 % para el clasificador no binario.

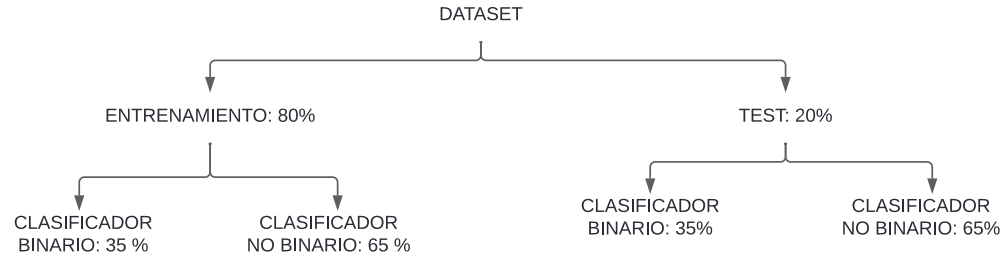


Figura 6.11: Separación de los conjuntos de datos para entrenar/test de cada modelo. Elaboración propia.

Con respecto al clasificador binario (se utiliza finalmente un 28 % de todas las instancias para entrenar y un 7 % para test), los resultados son muy buenos. A pesar de las pocas instancias tanto para entrenar como para test, es capaz de obtener muy buenos resultados. En la Figura 6.12 se muestra la matriz de confusión del modelo, donde la clase 0 hace referencia a no-aves y la clase 1 a aves. El modelo es capaz de discernir muy bien entre las imágenes que muestran aves y las que no. En la Figura 6.13 se muestra la tasa de acierto y la función de pérdida del clasificador.

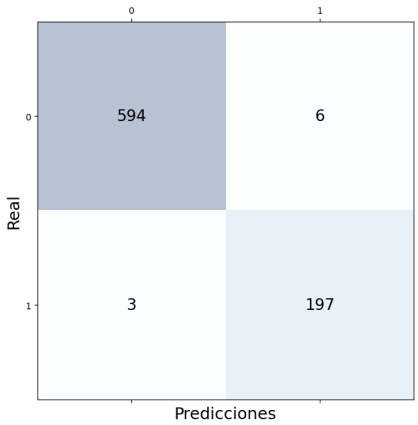
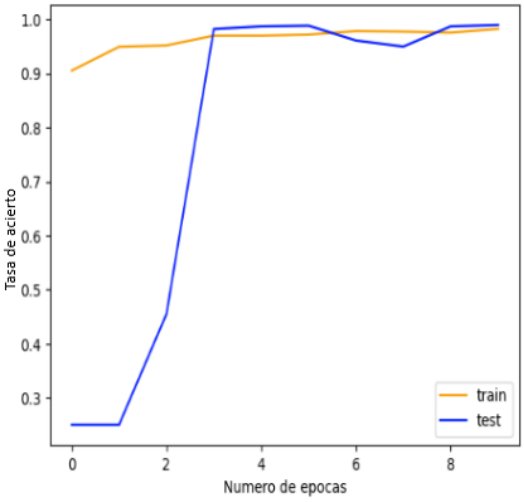
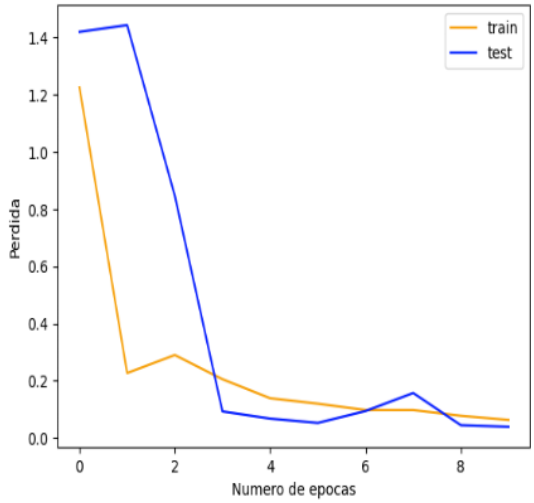


Figura 6.12: Matriz de confusión del modelo binario EfficientNet. Elaboración propia.



(a) Tasa de acierto de EfficientNetB0 binario. Elaboración propia.



(b) Pérdida de EfficientNetB0 binario. Elaboración propia.

Figura 6.13: Salida del modelo EfficientNetB0 binario.

El resto de datos disponibles son para entrenar (52 %) y probar (13 %) el segundo clasificador. Lo primero, será dar al clasificador binario todos estos datos como entrada. Las imágenes que se predigan como aves se darán como entrada al segundo clasificador. Cabe destacar que, aunque el primer clasificador filtre casi al completo todas las imágenes sin aves, tiene una pequeña tasa de error, y es por eso que al segundo clasificador le llega alguna imagen que no tiene aves. En la Figura 6.14 se ve la matriz de confusión de este clasificador. De nuevo, la clase 0 corresponde a otros tipos de aves, 1 a cuervo, 2 a halcón y 3 a no-ave.

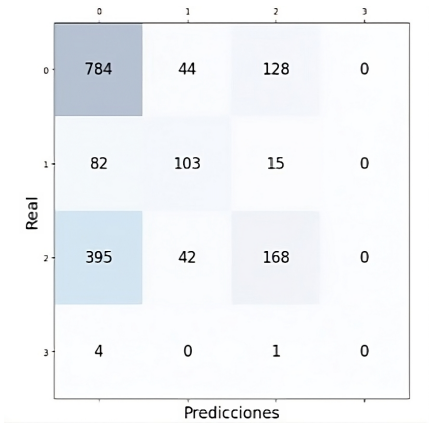
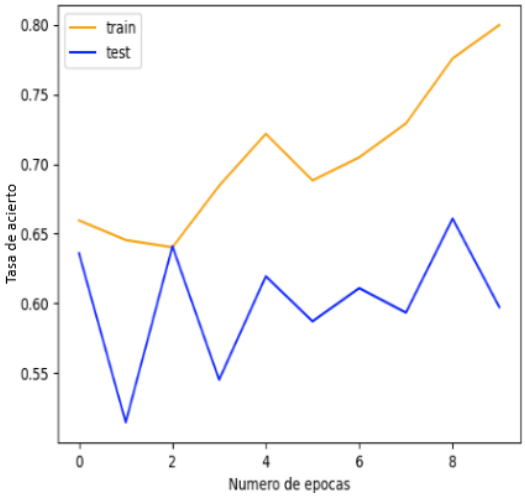
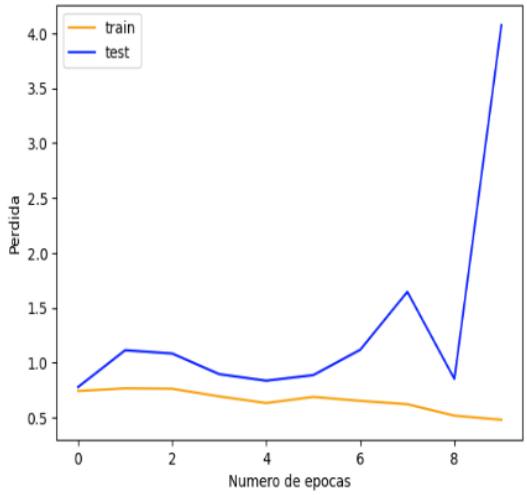


Figura 6.14: Matriz de confusión del modelo no binario EfficientNetB0. Elaboración propia.



(a) Acierto de EfficientNetB0 no binario. Elaboración propia.



(b) Pérdida de EfficientNetB0 binario. Elaboración propia.

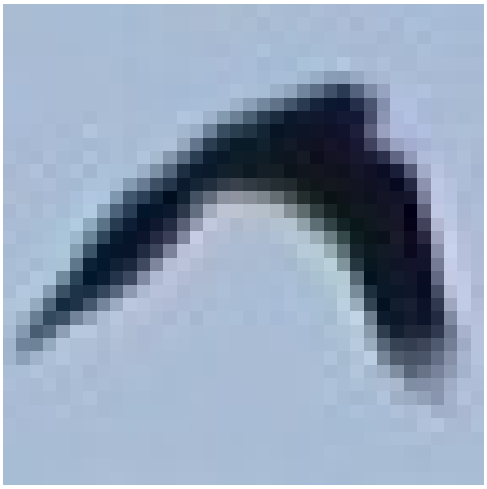
Figura 6.15: Resultados del modelo EfficientNetB0 no binario.

Se puede destacar que muy pocas imágenes que no tienen aves pasan el filtro del primer clasificador, aunque luego esas no las clasifica bien el segundo. No obstante, ocurre el mismo problema que antes, el segundo clasificador no es capaz de diferenciar entre un halcón, un cuervo u otro tipo de ave. La tasa de acierto del clasificador respecto a los tipos de aves es de alrededor de 0.6.

La principal conclusión a la que se ha llegado es que la calidad de las imágenes es demasiado pobre como para poder clasificarlas. En la Figura 6.16 se muestra un ejemplo de una imagen de un halcón y otra de un cuervo, donde se aprecia la poca resolución y calidad de

las imágenes.

(a) Imagen de ejemplo de un halcón. [26]



(b) Imagen de ejemplo de un cuervo. [26]

Figura 6.16: Ejemplos de imágenes del dataset.

6.2.4. Implementación del Visual Transformer

Con respecto a la implementación del Visual Transformer, no hay un modelo ya desarrollado en la librería keras ni Tensorflow, así que hubo que desarrollarlo de cero. En las prácticas curriculares se hizo una primera aproximación. No obstante, en este TFG, se ha modificado esa primera implementación añadiendo el token CLS y modificando la entrada del perceptrón (es la salida del encoder) que se encarga de hacer la clasificación. La parte más compleja fue seleccionar únicamente la salida del token CLS del encoder para realizar la predicción final.

En cuanto al preprocesamiento, se seleccionó el dataset en el cual se habían aplicado técnicas de aumentación de datos en las clases minoritarias, con fin de evitar que siempre se seleccione la clase mayoritaria y, por tanto, no generalizando correctamente. Se normalizaron las instancias y se realizó la separación en conjuntos de entrenamiento-test. Esta se realiza de tal manera que estos sean disjuntos, estratificados por clases y con distribución entrenamiento-test del 80 %-20 %.

En la Figura 6.17 se muestra la matriz de confusión del modelo ViT entrenado. Cabe destacar los malos resultados que obtiene con las imágenes de cuervos y de halcones. No hay ningún ejemplo que prediga como halcón o cuervo. Es muy posible que se deba a que las imágenes tengan demasiada poca resolución como para poder extraer información. En la Figura 6.18 se muestran las funciones de pérdida y la tasa de acierto de los conjuntos de entrenamiento y de test.

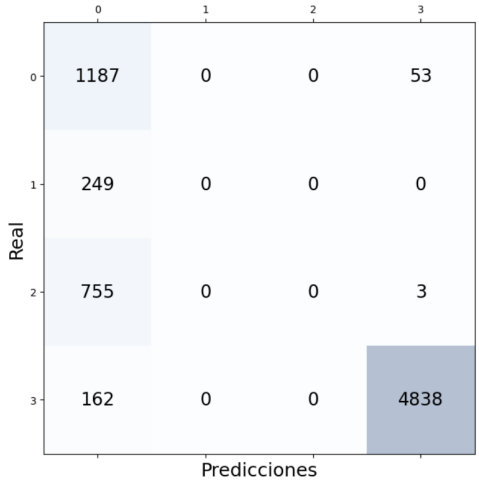
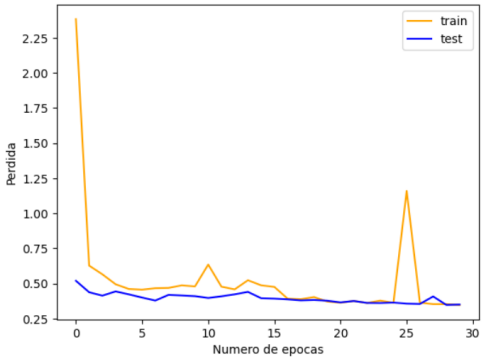
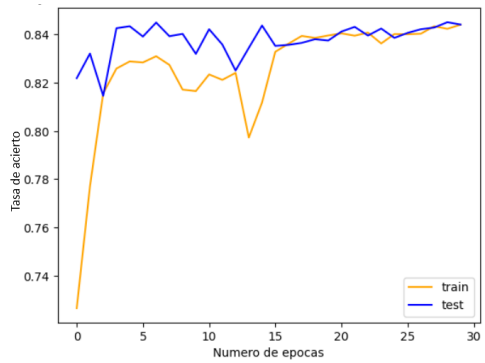


Figura 6.17: Matriz de confusión del modelo ViT. Elaboración propia.



(a) Función de pérdida modelo ViT. Elaboración propia.



(b) Tasa de acierto del modelo ViT. Elaboración propia.

Figura 6.18: Resultados del modelo ViT.

Cabe destacar que este modelo no solo ha tenido una tasa de acierto peor que los modelos de las EfficientNet (0.9051 EfficientNet frente a 0.84 ViT) sino que además hay clases que nunca predice. No obstante, todo esto no es suficiente para afirmar que las EfficientNet funcionan mejor para este conjunto de datos que los ViT, sería necesario hacer tests estadísticos para verificarlo. Entre los test que se pueden usar son el test de McNemar [41] o los test de Student [42] (todos se usan para comparar dos modelos sobre el mismo conjunto de datos). Como este conjunto de datos no es el conjunto de datos final, no se realizarán los test. Cuando se obtenga el dataset real con el que se entrenen los modelos finales, sí se realizarán dichos test para seleccionar el mejor modelo para ese conjunto de datos.

Capítulo 7

Clasificación de aves por sonido

7.1. Base teórica

El sonido se define formalmente como una onda física producida por las diferencias de presión en el aire. Todas las técnicas de aprendizaje automático usadas para procesar datos en forma de sonido van en la misma línea; se hace una extracción de características de los sonidos y después se aplican técnicas de visión artificial [43]. Puede parecer contradictorio el hecho de usar técnicas de visión artificial (CNN, Visual Transformers, etc.), pero las características obtenidas a partir de los sonidos tienen siempre 2 dimensiones, y se tratan como una imagen con un solo canal de color.

Otros métodos seguidos para obtener conocimiento a partir de las características extraídas del sonido es realizar un aplanado de ellas (pasar de 2 dimensiones a 1) y posteriormente entrenar algoritmos de aprendizaje automático tradicionales (árboles de decisión, MVS, Naive Bayes, etc.).

7.1.1. Espectrograma

Los espectrogramas son muy utilizados en la extracción de características de los sonidos. La finalidad de un espectrograma es representar las características principales del sonido (frecuencia, amplitud y tiempo) en una misma imagen [44]. La representación de una onda de sonido se hace en la dimensión del tiempo, con la amplitud en el eje y y el tiempo en el eje x, tal y como se observa en la Figura 7.1. El espectrograma representa la onda en 3 dimensiones: en el eje x representa el tiempo, en el eje y la frecuencia en Hz, y el color del espectrograma es la amplitud. En la Figura 7.2 se muestra la imagen de un espectrograma. Como el espectrograma tiene un solo canal de color, se suele representar en escala de grises. No obstante, se puede modificar la escala para que muestre otros colores.

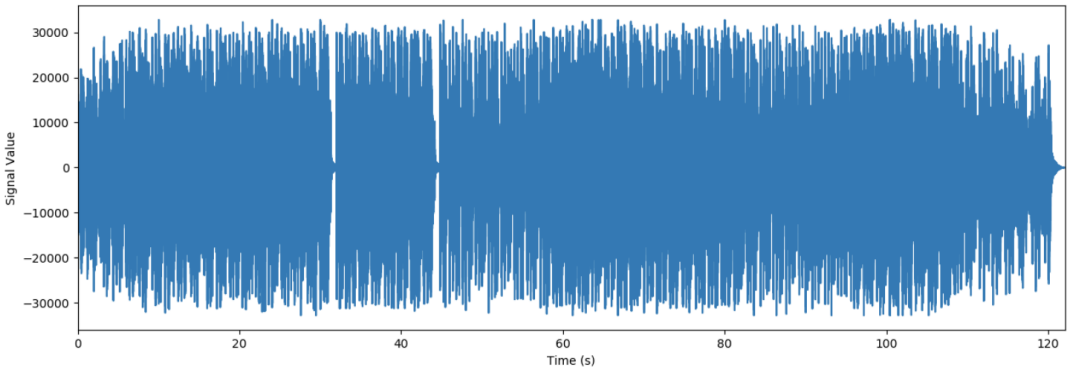


Figura 7.1: Representación de un sonido en la dimensión del tiempo. [45]

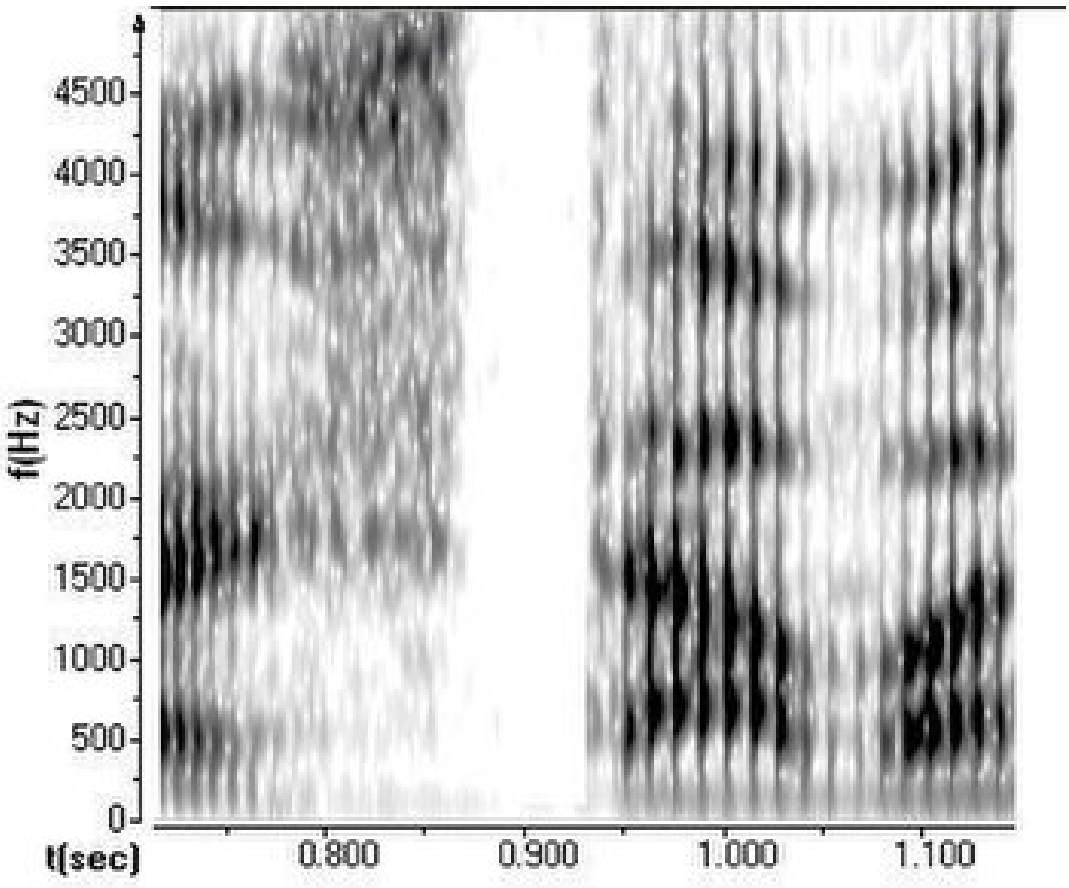


Figura 7.2: Espectrograma de un sonido. [46]

Un espectrograma aporta mucha información con respecto al sonido original, ya que para cada instante de tiempo se conocen todas sus frecuencias y sus respectivas amplitudes. Aplicando técnicas propias de visión artificial se pueden aprender patrones y formas [47], sin importar su localización espacial, tal y como se ha explicado en la Sección 5.1.2. A continuación, se especifica el proceso para obtener un espectrograma a partir de la onda de un sonido.

Tasa de muestreo

Lo primero que es necesario conocer es la forma en la que se representa un sonido digitalmente. Los sonidos, tal y como se producen en la naturaleza, son ondas físicas continuas. No obstante, para poder ser representadas en una máquina y poder ser tratadas, es necesario discretizarlas [48]. Es por ello que cuando se graba un sonido, se hacen medidas de ese sonido cada ciertos períodos de tiempo. Esos períodos de tiempo es la tasa de muestreo. Por ejemplo, si un sonido tiene una tasa de muestreo de 22.050 Hz, significa que se capturan 22.050 muestras de ese sonido por segundo.

Transformada de Fourier

La Transformada de Fourier se define como la operación que permite transformar señales (sonidos en este caso) del dominio del tiempo al dominio de la frecuencia [49]. Tal y como se ha comentado en la Sección 7.1.1, el sonido se almacena y procesa de forma discreta, por tanto, hay que aplicar la Transformada Discreta de Fourier (DFT), que se usa para transformar ondas discretas del dominio del tiempo al dominio de la frecuencia. La fórmula de esta operación se muestra en la Ecuación 7.1.Sin embargo, el algoritmo usado para computar esta operación es la *Transformada Rápida de Fourier* (FFT), que sirve para aplicar tanto la Transformada Discreta de Fourier como su inversa [50]. La Transformada Inversa Discreta de Fourier (IDTF) transforma una señal de la dimensión de la frecuencia a la dimensión del tiempo. La salida de la DFT ofrece información sobre cómo la energía de una onda varía en el tiempo [46].

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-\frac{2\pi i}{N} kn} \tag{7.1}$$

La DFT es simétrica y periódica, es decir, devolverá tanto frecuencias positivas como negativas. Las frecuencias negativas son una consecuencia matemática de la Ecuación 7.1, sin ningún significado físico [50]. Por ello, en el preprocesamiento posterior se eliminan.

El sonido es una onda no estacionaria, es decir, que el rango de frecuencias que contiene varía en el tiempo. En ondas no estacionarias la aplicación de la DFT no puede realizarse pues esta supone que la onda es periódica e infinita, es decir, que la onda debe repetirse infinitamente para que la DFT pueda representarla correctamente. No obstante, se considera que en intervalos diferenciales de tiempo, las frecuencias de la onda no varían. En ese caso,

sí se puede aplicar la DFT a esos intervalos pequeños de tiempo. Un espectrograma es el resultado de computar la DFT en pequeños intervalos de tiempo (25ms) en la onda original, y posteriormente hacer una apilación de todos ellos.

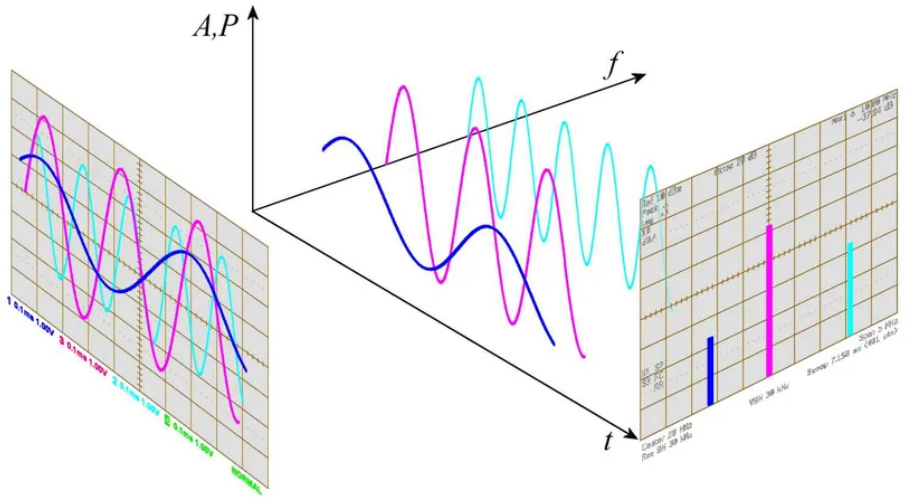


Figura 7.3: Dominio del tiempo vs dominio de la frecuencia en una onda. [51]

En la Figura 7.3 se ve la relación que hay entre el dominio del tiempo (izquierda) y el dominio de la frecuencia (derecha) de una onda. La DFT y la IDFT sirven para pasar de un dominio a otro.

7.1.2. Espectrograma de Mel

La escala de Mel es una escala logarítmica que trata de imitar la escala de audición del ser humano. El ser humano oye logarítmicamente, es decir, es capaz de diferenciar muy bien sonidos a bajas frecuencias, pero no es capaz de diferenciar sonidos con frecuencias muy altas. La escala de Mel trata de imitar ese comportamiento, de forma que a bajas frecuencias tiene un gran espectro de representación, a costa de no poder representar tan ricamente las altas frecuencias [52]. En la Ecuación 7.2 se define la fórmula para transformar Hercios (Hz) a Mel. En la Ecuación 7.3 se muestra la fórmula para realizar la operación inversa, de Mel a Hz.

$$M = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right) \tag{7.2}$$

$$f = 700 \cdot \left(10^{\frac{M}{2595}} - 1\right) \tag{7.3}$$

La aplicación de la fórmula 7.2 a un espectrograma da lugar a un espectrograma de Mel. En el dominio del aprendizaje automático aplicado a sonidos, estos son muy usados, especialmente en el reconocimiento de sentimientos en el habla. En la Figura 7.4 se muestra la representación gráfica de la Ecuación 7.2.

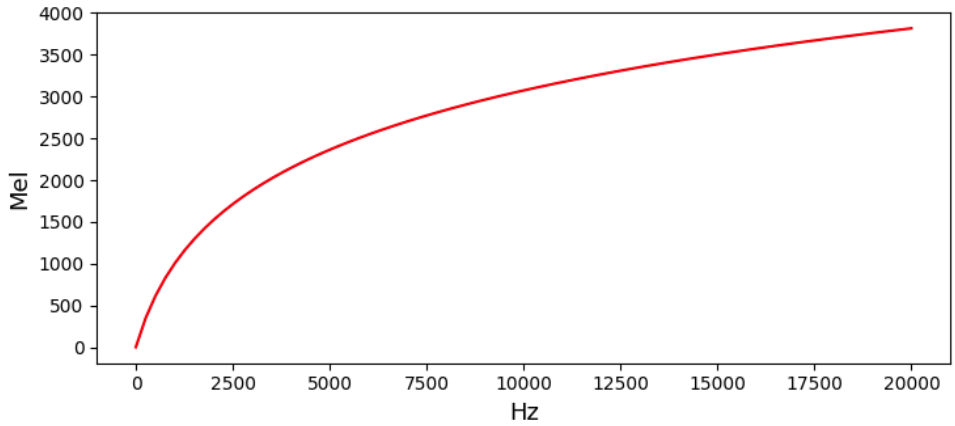


Figura 7.4: Escala de Mel. Elaboración propia.

Para la aplicación de la escala de Mel se crean una serie de filtros, de manera que cada uno de ellos solo capta unas frecuencias y filtra otras [53]. Para computarlo, se realiza una multiplicación de matrices donde la primera de ellas es el espectrograma y la segunda los filtros de Mel. En la Figura 7.5 se muestran los filtros de la escala de Mel, donde cada triángulo en la imagen hace referencia a un filtro. Por ejemplo, el filtro morado (el de más a la derecha de la imagen) es un filtro que solo acepta las frecuencias en el intervalo [3600,4000], filtrando las frecuencias restantes. Normalmente se aplican 26 filtros, aunque es posible usar una cantidad distinta. La separación de esos filtros se define en la Ecuación 7.4.

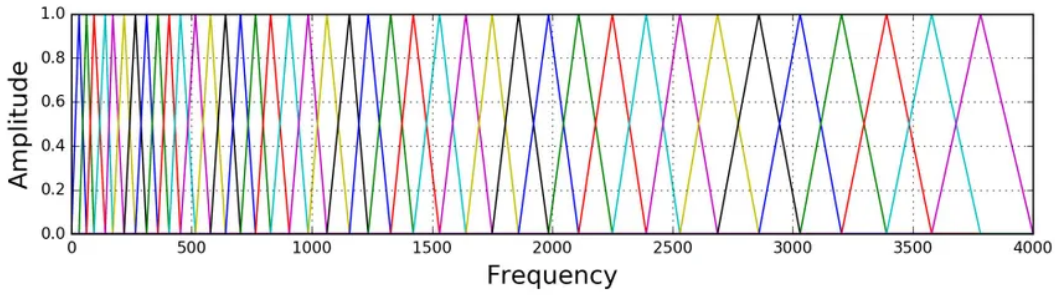


Figura 7.5: Filtros de la escala de Mel. [54]

$$M(f) = 1125 * \ln(1 + \frac{f}{700}) \tag{7.4}$$

7.1.3. Coeficientes cepstrales en frecuencia Mel

Para obtener los coeficientes cepstrales en frecuencia Mel (MFCC) se parte de un espectrograma de Mel. La primera operación que se hace es aplicar el logaritmo al espectrograma de Mel, y a continuación se calcula la IDFT [55]. La salida serán los MFCC, que es una característica muy comúnmente extraída para el procesamiento y extracción de conocimiento de señales. Su principal ventaja es la eliminación de información no relevante y la focalización en información relevante para el concepto. Es una característica muy ampliamente utilizada junto a los espectrogramas de Mel en el reconocimiento de sentimientos en el habla [56]. En este caso, en el eje x se tiene el tiempo y en el eje y los coeficientes de cada uno de los instantes de tiempo. Normalmente se calculan 13 coeficientes, aunque es posible que ese número varíe dependiendo de la naturaleza de los datos y del dominio.

Otras características que se pueden obtener son los deltas de los MFCC [57], que aportan información de cómo varían en el tiempo los MFCC (se obtienen haciendo operaciones sobre MFCCs contiguos).

7.1.4. Procesamiento de características

Tal y como se ha explicado en la Sección 3.1.5, hay principalmente 2 técnicas de procesamiento de las características extraídas de los sonidos. Una es apilar todas las características y procesarlas como si fueran imágenes (se puede interpretar que hay por cada sonido una sola imagen con tantos canales como características extraídas), y la otra es hacer un aplanado de esas características (convertirlas de 2 dimensiones a 1) y procesarlas con algoritmos de aprendizaje automático como árboles de decisión o máquinas de vector soporte [58].

7.2. Implementación

7.2.1. Segmentación de los sonidos

El objetivo de este módulo será, dado un audio con el canto de un ave, saber a qué ave corresponde. Por ello, la primera tarea que se realizó en este módulo fue el preprocesamiento de los datos. Tal y como se ha explicado en la Sección 4.3, se han obtenido los datos a partir de la API de la página web *Xeno-canto* [27]. Como por ahora solo hay intención de trabajar con las aves de la Comunidad Autónoma de Castilla y León, se seleccionaron únicamente aves procedentes de esa región. Esos audios con sonidos de aves están sin procesar, es decir

tienen ruido de fondo que pueden dificultar el aprendizaje. Por eso, lo primero que se hizo, fue usar la librería de Python *noisereduce* para eliminar el ruido de todos los audios [59].

Como cada uno de los audios tenía una cantidad arbitraria de cantos del ave, por cada uno de esos audios se obtuvieron los decibelios medios y se recortaron aquellas porciones del audio donde los decibelios fueran mayores o iguales que la media más un umbral. Se hizo una búsqueda para determinar el valor óptimo de este umbral y se determinó que era 80 dB. De esta manera se pudo segmentar el audio para que en cada porción hubiera un solo sonido del ave, además de eliminar el ruido entre sucesivos cantos del ave.

7.2.2. Extracción de características y preprocesamiento

A continuación se creó un módulo para realizar la extracción de características de todos los sonidos. Las características que se extrayeron son:

- **ZCR:** Significa 'Zero Crossing Rate' en inglés y es una medida de que hace referencia al número de veces que la energía de una señal cruza el eje x [60]. Digitalmente, la energía de las señales se escalan en el intervalo [-1,1]. Esta característica aporta información sobre el nivel de actividad de la onda.
- **Chroma:** Es similar a un espectrograma, con la diferencia de que no analiza todas las frecuencias presentes en el sonido como haría un espectrograma, sino que únicamente se centra en las frecuencias de las 12 notas musicales [61]. Esta característica ayuda a clasificar las tonalidades, necesario para discernir entre cada una de las especies de aves.
- **Espectrograma de Mel:** Explicado en la Sección 7.1.2.
- **Valor cuadrático medio:** Esta característica hace referencia a la amplitud o potencia de la señal. Ayuda a obtener información sobre la intensidad del sonido [62].
- **MFCC:** Definido y explicado en la Sección 7.1.3.

Existen 2 aproximaciones para realizar el aplanado de estas características. Se puede concatenar la fila *n+1* con la fila *n* recursivamente de todas las características extraídas, o se puede hacer una media de los valores por columnas. En este caso, se decidió usar la segunda opción, ya que la concatenación de filas da lugar a muchos valores por cada dato, haciendo el gasto computacional mucho mayor. La desventaja del método usado es que al hacer la media se pierde información.

El último, paso de la extracción de características es concatenarlas todas, de forma que finalmente por cada sonido se obtiene un vector de 1 dimensión, donde se ha aplanado y concatenado cada característica.

Por último se normalizan las características, en este caso se estandarizó. Se prefiere en este caso estandarizar a escalar ya que no se ha hecho un análisis de outliers. En caso de existir outliers, no es recomendable escalar los datos.

7.2.3. Modelos y resultados

Con respecto a los modelos usados, dada la gran cantidad de instancias que hay, se ha buscado modelos que sean baratos computacionalmente. Se utilizó tanto el modelo Random Forest como redes fully connected. Se utilizó la herramienta Optuna [63] para realizar una optimización de hiperparámetros. Optuna es una herramienta que entrena muchos modelos del mismo tipo (RandomForest y redes fully connected) con distintas condiciones iniciales, dando como resultado el mejor modelo. En este caso, el modelo que mejor resultados obtuvo fue el Random Forest con una tasa de acierto de 0.9923, con los siguientes hiperparámetros:

- Número de estimadores: 106.
- Profundidad máxima: 16.
- Número mínimo de instancias por partición: 4.
- Número mínimo de instancias por hoja: 9.

Por otra parte, las redes fully connected obtuvieron resultados peores aunque siguen siendo muy buenos (tasa de acierto de 0.9058). De nuevo, se usó la herramienta Optuna para seleccionar la estructura óptima de la red. Esta consta de un total de 5 capas ocultas con 1000,600,300,150,75 neuronas respectivamente. Ante resultados similares, se prefiere el método que menos gasto computacional requiere, en este caso, los árboles.

Los expcepcionalmente buenos resultados con distintos algoritmos hace pensar que no se debe a ningún tipo de aleatoriedad. Además, estas tasas de error para distintos algoritmos demuestra que las características obtenidas de los sonidos explican casi por completo a los sonidos de las aves. Por otra parte, también demuestra que el preprocesamiento realizado (segmentación de los audios) ha sido el correcto, favoreciendo la generalización.

Capítulo 8

Conclusiones

Con respecto a las conclusiones del proyecto, se va a desarrollar una sección por cada dominio en el que se ha trabajado.

8.1. Segmentación de imágenes

El modelo de segmentación de imágenes (YOLOv8) obtiene unos resultados buenos con respecto a la detección de aves. Se ha trabajado mucho en el algoritmo de detección de trayectorias y es muy notable la evolución de los resultados. Tal y como se encuentra ahora el algoritmo, es capaz de detectar todas las aves, sin importar su tamaño, siempre y cuando las condiciones lumínicas y ambientales no sean muy extremas. Además, en caso de que en algún fotograma no se detecte algún ave, el algoritmo de trayectorias es capaz de paliar ese problema.

El único problema notable con respecto a este algoritmo es si dos aves se cruzan en el mismo instante de tiempo, que el modelo YOLOV8 no es capaz de detectar dos aves distintas. Como trabajo futuro, se puede intentar entrenar al modelo con imágenes donde haya solapamiento de aves.

8.2. Clasificación de imágenes

En la clasificación de imágenes de aves se han hecho varios modelos y varias ejecuciones. Se puede destacar que el uso de técnicas de aumentación de datos pueden mejorar mucho los resultados, especialmente cuando se realizan sobre clases minoritarias para evitar que los modelos predigan siempre las clases mayoritarias. Dado que el dataset usado no tiene todas las clases relativamente bien balanceadas, se obtienen mejores resultados al hacer aumentación de datos sobre aquellas clases que son minoritarias. Se hicieron pruebas únicamente para

comprobar si el uso de esas técnicas aportan nueva información, y los resultados fueron que se obtenía una tasa de acierto de entre un 0.1 y un 0.15 mayor.

Por otra parte, tanto el modelo EfficientNet como los ViT son capaces de diferenciar muy bien si en una imagen hay un ave o no (clasificación booleana), incluso a pesar de la poca resolución de las imágenes (tasa de acierto de 0.9887 y 0.9629). No obstante, no son capaces de distinguir suficientemente bien unas aves de otras (tasa de acierto de 0.6 y 0.5417 respectivamente), seguramente por la baja resolución de las imágenes. No se ha hecho ningún test estadístico para comprobar cuál de los dos funciona mejor para el dataset usado, ya que este no es el dataset final. No obstante, cuando se disponga de los datos finales, sí se harán y se podrá tomar la decisión sobre cuál de los dos modelos funcionan mejor para ese dataset.

8.3. Clasificación de audio

Los buenos resultados que se obtienen no solo con un modelo (tasa de acierto de 0.9923 con Random Forest), sino con modelos distintos (redes fully connected con tasa de acierto de 0.9058) hace pensar que el algoritmo encargado de realizar la segmentación inicial de los audios ha sido capaz de diferenciar bien cuándo empieza y acaba un canto del ave.

Por otra parte, cada una de las características extraídas (ZCR, chroma, espectrograma de Mel, valor cuadrático medio y MFCC) representan información relevante y específica de cada dato. Queda por investigar si la inclusión de más características distintas pueden mejorar los resultados en unos datos más amplios (solo se han seleccionado especímenes grabados en Castilla y León).

Capítulo 9

Trabajo futuro

Con respecto al trabajo futuro a realizar, hay unas líneas claras.

- La primera es la obtención del dataset final para la clasificación de aves a través de su imagen. Será necesario seguir los pasos explicados en el Capítulo 6. La calidad de las imágenes debe ser la mayor posible, ya que como se ha visto en el Capítulo 6, con imágenes con baja resolución no se podrá extraer la suficiente información como para clasificar las imágenes de las aves.
- Realizar test estadísticos para comprobar si hay alguna diferencia significativa entre las EfficientNet y los ViT. Como se comparan 2 algoritmos respecto al mismo conjunto de datos, se debe usar el test de McNemar y los test de Student. En caso de tener capacidad de cómputo como para realizar varios entrenamientos distintos por cada algoritmo, se pueden usar los test de Student en sus distintas variantes. Si por el contrario no se tiene recursos computacionales suficientes, se utilizará el test de McNemar, que requiere de un solo entrenamiento por cada modelo.
- Obtener más imágenes y vídeos con condiciones lumínicas pobres con la cámara del proyecto (AXIS Q6225-LE PTZ Camera), segmentarlos y entrenar el algoritmo YOLOv8. De esta manera, se podrán detectar aves nocturnas.
- Integración de los modelos de clasificación (imagen y sonido) para que trabajen conjuntamente y se puedan usar a la vez.
- Optimización de los parámetros y heurísticas usadas en la detección de trayectorias, para evitar que si 2 aves se cruzan en el mismo instante, el algoritmo no detecte solo 1 ave sino a las 2.

Universidad de Valladolid

INSTITUTE

ANEXO

DOCUMENTO PARA LA COMPARTICIÓN DE TITULARIDAD DE DERECHOS DE PROPIEDAD INTELECTUAL DE TFG/TFM EN CONVENIO

D. Jaime Álvarez Urueña alumno de la Universidad de Valladolid y autor del TFG/TFM en convenio con la Fundación instituto internacional de investigación en inteligencia artificial y ciencias de la computación titulado "Inteligencia Artificial aplicada a la Monitorización y Control de fauna", expresa su decisión de compartir la titularidad de los derechos de propiedad intelectual de su TFG/TFM con la Fundación instituto internacional de investigación en inteligencia artificial y ciencias de la computación para realizar las siguientes acciones:

- Realizar publicaciones de los resultados del TFG/TFM siempre y cuando el alumno esté informado y, además, figure su nombre en dichas publicaciones.
- Realizar nuevos proyectos que impliquen modificaciones o ampliaciones del TFG/TFM.
- Reutilizar partes del TFG/TFM en otros proyectos que puedan tener fines comerciales, siempre y cuando lo permitan las licencias bajo las que se ha desarrollado el TFG/TFM.

En Valladolid, a 03 de Julio de 2024

Fdo.: Jaime Álvarez Urueña

Bibliografía

[1] «AXIS Q6225-LE PTZ Camera — Axis Communications.» (), dirección: https://www.axis.com/es-es/products/axis-q6225-le (visitado 15-05-2024).

[2] «Make Sense.» (), dirección: https://www.makesense.ai/ (visitado 16-05-2024).

[3] Ultralytics. «Inicio.» (), dirección: https://docs.ultralytics.com/es/ (visitado 17-05-2024).

[4] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 7th. Newtown Square, PA: Project Management Institute, 2021, ISBN: 978-1-62825-664-2.

[5] J. Alves, J. Shamoun-Baranes, P. Desmet y col., *Monitoring continent-wide aerial patterns of bird movements using weather radars*. 31 de mar. de 2015.

[6] F. Liechti y H. van Gasteren, «CURRENT STAGE OF BIRD RADAR SYSTEMS,» 1 de jun. de 2010.

[7] P. Gemmar, «Detection of bird activity in radar images,» 1 de ene. de 2012. dirección: https://www.academia.edu/71442826/Detection_of_Bird_Activity_in_Radar_Images (visitado 17-05-2024).

[8] S. Albawi, T. A. Mohammed y S. Al-Zawi, «Understanding of a convolutional neural network,» en *2017 International Conference on Engineering and Technology (ICET)*, ago. de 2017, págs. 1-6. DOI: 10.1109/ICEngTechnol.2017.8308186. dirección: https://ieeexplore.ieee.org/document/8308186 (visitado 17-05-2024).

[9] S. Indolia, A. K. Goswami, S. P. Mishra y P. Asopa, «Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach,» *Procedia Computer Science*, International Conference on Computational Intelligence and Data Science, vol. 132, págs. 679-688, 1 de ene. de 2018, ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.05.069. dirección: https://www.sciencedirect.com/science/article/pii/S1877050918308019 (visitado 17-05-2024).

[10] J. Nagi, F. Ducatelle, G. A. Di Caro y col., «Max-pooling convolutional neural networks for vision-based hand gesture recognition,» en *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, nov. de 2011, págs. 342-347. DOI: 10.1109/ICSIPA.2011.6144164. dirección: https://ieeexplore.ieee.org/document/6144164 (visitado 17-05-2024).

[11] S. R. Dubey, S. K. Singh y B. Chaudhuri, «Activation Functions in Deep Learning: A comprehensive Survey and Benchmark,» *Neurocomputing*, vol. 503, 1 de jul. de 2022. DOI: 10.1016/j.neucom.2022.06.111.

[12] J. M. B. Genes. «Breve historia de las redes neuronales,» Medium. (22 de oct. de 2021), dirección: https://medium.com/@joselobenitezg/breve-historia-de-las-redes-neuronales-357e60ea443c (visitado 17-05-2024).

[13] K. He, X. Zhang, S. Ren y J. Sun, *Deep Residual Learning for Image Recognition*, 10 de dic. de 2015. DOI: 10.48550/arXiv.1512.03385. arXiv: 1512.03385[cs]. dirección: http://arxiv.org/abs/1512.03385 (visitado 20-05-2024).

[14] G. Huang, Z. Liu, L. van der Maaten y K. Weinberger, *Densely Connected Convolutional Networks*. 24 de jul. de 2017. DOI: 10.1109/CVPR.2017.243.

[15] C. Shorten y T. M. Khoshgoftaar, «A survey on Image Data Augmentation for Deep Learning,» *Journal of Big Data*, vol. 6, n.º 1, pág. 60, 6 de jul. de 2019, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. dirección: https://doi.org/10.1186/s40537-019-0197-0 (visitado 20-05-2024).

[16] A. Yang y D. Silver, «The Disadvantage of CNN versus DBN Image Classification Under Adversarial Conditions,» 18 de mayo de 2021. DOI: 10.21428/594757db.b65acd40.

[17] M. Tan y Q. V. Le, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, 11 de sep. de 2020. DOI: 10.48550/arXiv.1905.11946. arXiv: 1905.11946[cs,stat]. dirección: http://arxiv.org/abs/1905.11946 (visitado 17-05-2024).

[18] A. Vaswani, N. Shazeer, N. Parmar y col., *Attention is all you need*, 1 de ago. de 2023. arXiv: 1706.03762[cs]. dirección: http://arxiv.org/abs/1706.03762 (visitado 15-05-2024).

[19] «Archivo:Redes-neuronales-recurrentes-deep-learning-jordi-torres-1024x535.png - Wikipedia, la enciclopedia libre.» (23 de abr. de 2023), dirección: https://commons.wikimedia.org/wiki/File:Redes-neuronales-recurrentes-deep-learning-jordi-torres-1024x535.png (visitado 17-05-2024).

[20] A. Gillioz, J. Casas, E. Mugellini y O. Abou Khaled, *Overview of the Transformer-based Models for NLP Tasks*. 26 de sep. de 2020, 179 págs., Pages: 183. DOI: 10.15439/2020F20.

[21] A. Dosovitskiy, L. Beyer, A. Kolesnikov y col., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 3 de jun. de 2021. DOI: 10.48550/arXiv.2010.11929. arXiv: 2010.11929[cs]. dirección: http://arxiv.org/abs/2010.11929 (visitado 17-05-2024).

[22] M. Ramashini, P. E. Abas, K. Mohanchandra y L. C. D. Silva, «Robust cepstral feature for bird sound classification,» *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, n.º 2, págs. 1477-1487, 1 de abr. de 2022, Number: 2, ISSN: 2722-2578. DOI: 10.11591/ijece.v12i2.pp1477-1487. dirección: https://ijece.iaescore.com/index.php/IJECE/article/view/25893 (visitado 17-05-2024).

[23] S. Carvalho, «Automatic classification of bird sounds: Using MFCC and mel spectrogram features with deep learning,» *Vietnam Journal of Computer Science*, dirección: https://www.academia.edu/114461733/Automatic_Classification_of_Bird_Sounds_Using_MFCC_and_Mel_Spectrogram_Features_with_Deep_Learning (visitado 17-05-2024).

[24] M. Ning, Y. Lu, W. Hou y M. Matskin, «YOLOv4-object: an Efficient Model and Method for Object Discovery,» en *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, Madrid, Spain: IEEE, jul. de 2021, págs. 31-36, ISBN: 978-1-66542-463-9. DOI: 10.1109/COMPSAC51774.2021.00016. dirección: https://ieeexplore.ieee.org/document/9529473/ (visitado 17-05-2024).

[25] L. Lucchese y S. Mitra, «Color Image Segmentation: A State-of-the-Art Survey,» *Proceedings of Indian National Science Academy*, vol. 2, 1 de ene. de 2001.

[26] nae-lab. «Naemura Lab,» The University of Tokyo, JAPAN. (), dirección: https://nae-lab.tumblr.com/ (visitado 17-05-2024).

[27] «xeno-canto :: Sharing wildlife sounds from around the world.» (), dirección: https://xeno-canto.org/ (visitado 16-05-2024).

[28] «API :: xeno-canto.» (), dirección: https://xeno-canto.org/explore/api (visitado 17-05-2024).

[29] «Archivo:ResBlock.png - Wikipedia, la enciclopedia libre.» (1 de nov. de 2015), dirección: https://commons.wikimedia.org/wiki/File:ResBlock.png (visitado 20-05-2024).

[30] T.-Y. Lin, M. Maire, S. Belongie y col., *Microsoft COCO: Common Objects in Context*, 20 de feb. de 2015. DOI: 10.48550/arXiv.1405.0312. arXiv: 1405.0312[cs]. dirección: http://arxiv.org/abs/1405.0312 (visitado 21-05-2024).

[31] A. Howard, M. Zhu, B. Chen y col., «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,» 16 de abr. de 2017.

[32] B. Khasoggi, E. Ermatita y S. Samsuryadi, «Efficient mobilenet architecture as image recognition on mobile and embedded devices,» *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, pág. 389, 1 de oct. de 2019. DOI: 10.11591/ijeecs.v16.i1.pp389-394.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, 21 de mar. de 2019. DOI: 10.48550/arXiv.1801.04381. arXiv: 1801.04381[cs]. dirección: http://arxiv.org/abs/1801.04381 (visitado 20-05-2024).

[34] Y. Yuldashev, M. Mukhiddinov, A. Abdusalomov, R. Nasimov y J. Cho, «Parking Lot Occupancy Detection with Improved MobileNetV3,» *Sensors*, vol. 23, pág. 7642, 3 de sep. de 2023. DOI: 10.3390/s23177642.

[35] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, 4 de abr. de 2017. DOI: 10.48550/arXiv.1610.02357. arXiv: 1610.02357[cs]. dirección: http://arxiv.org/abs/1610.02357 (visitado 20-05-2024).

[36] X. Miao, Y. Wang, Y. Jiang y col., «Galvatron: Efficient Transformer Training over Multiple GPUs Using Automatic Parallelism,» *Proceedings of the VLDB Endowment*, vol. 16, n.º 3, págs. 470-479, nov. de 2022, ISSN: 2150-8097. DOI: 10.14778/3570690.3570697. arXiv: 2211.13878[cs]. dirección: http://arxiv.org/abs/2211.13878 (visitado 20-05-2024).

[37] A. Dosovitskiy, L. Beyer, A. Kolesnikov y col., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 3 de jun. de 2021. DOI: 10.48550/arXiv.2010.11929. arXiv: 2010.11929[cs]. dirección: http://arxiv.org/abs/2010.11929 (visitado 20-05-2024).

[38] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 24 de mayo de 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805[cs]. dirección: http://arxiv.org/abs/1810.04805 (visitado 20-05-2024).

[39] A. Mikołajczyk-Bareła y M. Grochowski, *Data augmentation for improving deep learning in image classification problem*. 1 de mayo de 2018, 117 págs., Pages: 122. DOI: 10.1109/IIPHDW.2018.8388338.

[40] «Aumento de datos — TensorFlow Core.» (), dirección: https://www.tensorflow.org/tutorials/images/data_augmentation?hl=es-419 (visitado 20-05-2024).

[41] M. Smith y G. Ruxton, «Effective use of the McNemar test,» *Behavioral Ecology and Sociobiology*, vol. 74, pág. 133, 10 de oct. de 2020. DOI: 10.1007/s00265-020-02916-y.

[42] A. Al-Achi, «The Student's t-Test: A Brief Description,» vol. 5, págs. 1-3, 9 de feb. de 2019.

[43] F. Roche, «Music sound synthesis using machine learning : Towards a perceptually relevant control space,» Tesis doct., 29 de sep. de 2020.

[44] S. Fulop y K. Fitz, «A Spectrogram for the Twenty-First Century,» *Acoustics Today*, vol. 2, 1 de ene. de 2006. DOI: 10.1121/1.2961138.

[45] «How to visualize sound in python,» LearnPython.com. Section: blog. (24 de feb. de 2022), dirección: https://learnpython.com/blog/plot-waveform-in-python/ (visitado 16-05-2024).

[46] «Wikiwand - Espectrograma,» Wikiwand. (), dirección: https://www.wikiwand.com/es/Espectrograma (visitado 16-05-2024).

[47] V. Zue y R. Cole, *Experiments on spectrogram reading*. 1 de mayo de 1979, vol. 4, 116 págs., Pages: 119. DOI: 10.1109/ICASSP.1979.1170735.

[48] D. Lavry y L. Engineering, «Sampling theory for digital audio,»

[49] M. Müller, «The Fourier Transform in a Nutshell,» en 1 de ago. de 2015, págs. 39-57, ISBN: 978-3-319-21944-8.

[50] U. Oberst, «The Fast Fourier Transform,» *SIAM J. Control and Optimization*, vol. 46, págs. 496-540, 1 de ene. de 2007. DOI: 10.1137/060658242.

[51] D.-I. (C. Wolff. «Fundamentos de radar - Analizador de espectro.» Publisher: Dipl.-Ing. (FH) Christian Wolff. (), dirección: https://www.radartutorial.eu/22.messpraxis/mp06.es.html (visitado 16-05-2024).

[52] S. Umesh, L. Cohen y D. Nelson, *Fitting the Mel scale*. 15 de abr. de 1999, vol. 1, 217 págs., Pages: 220 vol.1, ISBN: 978-0-7803-5041-0. DOI: 10.1109/ICASSP.1999.758101.

[53] S. R. Madikeri y H. A. Murthy, «Mel Filter Bank energy-based Slope feature and its application to speaker recognition,» *2011 National Conference on Communications (NCC)*, págs. 1-4, ene. de 2011, Conference Name: 2011 National Conference on Communications (NCC) ISBN: 9781612840901 Place: Bangalore, India Publisher: IEEE. DOI: 10.1109/NCC.2011.5734713. dirección: http://ieeexplore.ieee.org/document/5734713/ (visitado 16-05-2024).

[54] M. Montanaro, A. M. Rinaldi, C. Russo y C. Tommasino, «A rule-based obfuscating focused crawler in the audio retrieval domain,» *Multimedia Tools and Applications*, vol. 83, n.º 9, págs. 25 231-25 260, 1 de mar. de 2024, ISSN: 1573-7721. DOI: 10.1007/s11042-023-16155-6. dirección: https://doi.org/10.1007/s11042-023-16155-6 (visitado 16-05-2024).

[55] M. A. Hossan, S. Memon y M. A. Gregory, «A novel approach for MFCC feature extraction,» en *2010 4th International Conference on Signal Processing and Communication Systems*, dic. de 2010, págs. 1-5. DOI: 10.1109/ICSPCS.2010.5709752. dirección: https://ieeexplore.ieee.org/document/5709752 (visitado 16-05-2024).

[56] L. Muda, M. Begam e I. Elamvazuthi, «Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques,» vol. 2, n.º 3, 2010.

[57] K. Kumar, C. Kim y R. M. Stern, «Delta-spectral cepstral coefficients for robust speech recognition,» en *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, Czech Republic: IEEE, mayo de 2011, págs. 4784-4787, ISBN: 978-1-4577-0538-0. DOI: 10.1109/ICASSP.2011.5947425. dirección: http://ieeexplore.ieee.org/document/5947425/ (visitado 16-05-2024).

[58] J.-C. Wang, J.-F. Wang, C.-B. Lin, K.-T. Jian y W.-H. Kuok, *Content-Based Audio Classification Using Support Vector Machines and Independent Component Analysis*. 1 de ene. de 2006, vol. 4, 157 págs., Pages: 160. DOI: 10.1109/ICPR.2006.407.

[59] *noisereduce: Noise reduction using Spectral Gating in Python*, ver. 3.0.2. dirección: https://github.com/timsainb/noisereduce (visitado 16-05-2024).

[60] A. Sharma, «Implementation of ZCR and STE techniques for the detection of the voiced and unvoiced signals in Continuous Punjabi Speech,» *International journal of Emerging Trends in Science and Technology*, 26 de jun. de 2016. DOI: 10.18535/ijetst/v3i06.14.

[61] A. Shah, M. Kattel, A. Nepal y D. Shrestha, *Chroma Feature Extraction*. 20 de ene. de 2019.

[62] «Valor Cuadrático Medio o RMS — PDF — Media cuadrática — Media,» Scribd. (), dirección: https://es.scribd.com/document/368294501/Valor-Cuadratico-Medio-o-RMS (visitado 16-05-2024).

[63] «Optuna - a hyperparameter optimization framework,» Optuna. (), dirección: https://optuna.org/ (visitado 17-06-2024).