Universidad de Castilla-La Mancha

Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Clasificación Automática de Pájaros en Función de su Canto

Javier Senabre Urrea

8 de Julio de 2024

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Clasificación Automática de Pájaros en Función de su Canto

Autor: Javier Senabre Urrea

Tutores: José Antonio Gámez Martín

Juan Carlos Alfaro Jiménez

Declaración de Autoría

Yo, Javier Senabre Urrea, con DNI 49432922B, declaro que soy el único autor del trabajo fin de grado titulado “Clasificación Automática de Pájaros en Función de su Canto” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 8 de Julio de 2024

Fdo: Javier Senabre Urrea

SENABRE URREA JAVIER - 49432922B

Firmado digitalmente por SENABRE URREA JAVIER - 49432922B
Fecha: 2024.07.08 00:03:17 +02'00'

Resumen

Hoy en día, la conservación de animales como las aves es un tema muy importante debido a que nuestra actividad como humanos ha causado que numerosas especies se encuentren en peligro de extinción. Por este motivo, facilitar herramientas que permitan la detección de estas especies ayudará a conservarlas y monitorearlas.

Gracias al aprendizaje automático, es posible entrenar modelos para identificar pájaros a partir de su imagen. Sin embargo, nos centraremos en identificarlos con su canto porque permite reconocer pájaros sin necesidad de verlos, ya que algunos ecosistemas donde estos se encuentran tienen poca visibilidad.

Durante este proyecto, crearemos distintos modelos siguiendo ciertas arquitecturas concretas de redes neuronales convolucionales para comprobar su éxito. Además, aplicaremos varios procesos de preprocesamiento para mejorar el resultado de los modelos. Algunos de estos procesos consistirán en transformar los audios en espectrogramas, aplicar ciertos filtros, reducir el ruido de los audios, etc.

Como contribución final, se desarrollará una aplicación que incluirá el mejor modelo obtenido mediante aprendizaje automático, que permitirá llevar a cabo el proceso de clasificación a partir del canto de un pájaro de forma sencilla y amigable.

Agradecimientos

Primero, quiero agradecer a mis padres por apoyarme incondicionalmente durante toda mi vida. No estaría donde estoy sin ellos. Gracias Reme y Celestino por ser unos padres maravillosos.

También, quiere agradecer a mi hermana Lucía por ser un pilar fundamental de mi vida. Siempre alegrándome y haciéndome reír con su forma de ser y sus palabras.

Quiero agradecer a mis amigos de la Universidad que han ayudado a lo largo de estos 4 años. Pablo, Álvaro, Adrián y Pedro gracias por acompañarme durante estos años.

A mis tutores José Antonio Gámez Martín y Juan Carlos Alfaro Jiménez, por ayudarme a realizar este trabajo y siempre estar ahí para resolverme cualquier duda.

Quiero agradecer a mis amigos de mi pueblo Juan Carlos, Cristián y Fernando por apoyarme y darme esa motivación para continuar. Nunca olvidaré los momentos inolvidables que hemos vivido.

Finalmente, quiero agradecer a todas esas personas que han trabajado en crear documentales sobre la vida salvaje que me han inspirado para realizar este trabajo.

Muchas gracias a todos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Introducción

La naturaleza produce distintos sonidos que cautivan al ser humano, sin embargo, el canto de los pájaros siempre ha sido un sonido distintivo que ha llamado bastante la atención. Este canto nos ha acompañado desde tiempo lejanos, ayudándonos a reconocer a estos animales y en cierta medida, influenciándonos.

El canto de los pájaros tiene como objetivo comunicar un mensaje a otros animales, es decir, los pájaros utilizan sus cantos para aparearse, alertar sobre peligros, amenazar a otros rivales, etc. Con este hecho, podemos notar que el canto de cada ave es diferente a otra, permitiendo que sea posible identificarlas.

La motivación para clasificar pájaros según su canto son las numerosas aplicaciones que se pueden desarrollar con los nuevos avances tecnológicos sobre todo en el ámbito de la inteligencia artificial, facilitando la conservación de estos pájaros al tener herramientas que permitan a cualquiera reconocerlos sin ningún conocimiento previo sobre estos. Cualquiera que pueda grabar el canto de un pájaro, podría conocer que ave es sin necesidad de ver al animal, ayudando a personas en áreas boscosas o con poca visibilidad a monitorear estas especies.

1.2 Objetivos

El objetivo principal de este trabajo es estudiar y clasificar las diferentes muestras de sonido de canto de pájaros que hemos obtenido para ayudar a la conservación y monitoreo de estas especies. Además, se realizará una aplicación con el modelo que mejor resultado haya dado. Para alcanzar este objetivo principal, planteamos los siguientes objetivos específicos:

- **Seleccionar el conjunto de algoritmos de aprendizaje automático** *a priori* **más eficaces** para abordar la tarea propuesta, entre ellos las redes neuronales profundas.
- **Diseñar distintos *pipelines*** que permitan llevar a cabo un **preprocesamiento** que limpie y enriquezca los datos disponibles.
- **Comparar la eficacia de diferentes algoritmos y sus parametrizaciones** (selección de modelos) mediante las medidas apropiadas (tasa de acierto, sensibilidad, precisión, etc.).
- **Diseñar una pequeña aplicación** que permita ejecutar el sistema desarrollado.

1.3 Competencias abordadas

Las competencias abordadas estarán relacionadas con la intensificación de Computación, ya que este trabajo se encuentra en este ámbito. Las competencias abordadas son:

- **Capacidad para evaluar la complejidad computacional de un problema**, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.
- **Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes** y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.
- **Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano** en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes entornos inteligentes.
- **Capacidad para conocer y desarrollar técnicas de aprendizaje computacional** y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

1.4 Estructura del documento

El trabajo seguirá la siguiente estructura:

1. **Contexto y análisis**: Explicaremos conceptos esenciales para comprender el problema y analizaremos las grabaciones.
2. **Creación del modelo**: Desarrollaremos el modelo según la metodología CRISP-DM, es decir, realizaremos un ciclo de vida.

3. **Creación de la aplicación**: Crearemos una aplicación con el mejor modelo obtenido.
4. **Conclusiones**: Explicaremos las conclusiones obtenidas.
5. **Anexos**: Se nombrarán los distintos recursos y librerías utilizadas.

Capítulo 2

Estado del Arte

2.1 Canto de los pájaros

Cada pájaro tiene un canto distinto, aunque algunas aves puedan sonar de forma parecida. Este canto es distinto porque cada pájaro aprende los cantos y sonidos de sus padres, sin embargo, la morfología del animal también afecta a los sonidos emitidos.

No obstante, los pájaros son capaces de emitir estos cantos gracias a un órgano llamado siringe. Similar a los seres humanos con sus cuerdas vocales, los pájaros tienen ese órgano que les permite modular el sonido en una amplia escala temporal, por esto mismo, algunas aves son capaces de imitar sonidos como los loros. Este órgano es una estructura cartilaginosa ubicada en la unión de los bronquios y la tráquea que, al desplazarse el aire por esos tejidos, vibran provocando la fonación de estas aves [1].

En la Figura 2.1 se puede ver una imagen con este órgano. Como se puede apreciar, la siringe tiene dos ramificaciones que conducen a cada pulmón. Esta ramificación permite a los pájaros controlar de forma óptima el aire que es expulsado por cada pulmón, permitiendo que puedan cerrar una ramificación para producir un sonido específico y viceversa. También, se puede observar la similitud con nuestras cuerdas vocales, pero con la diferencia antes mencionada y la ubicación de su órgano fonador al encontrase en la tráquea.

No todas las aves tienen siringe tan desarrolladas, algunas aves como los avestruces o cigüeñas blancas no tienen este órgano o no esta tan desarrollado, comunicándose de otra forma con sus semejantes. Es curioso porque muchas de las aves que tienen desarrollado este órgano, utilizan otros sonidos para comunicarse, como en el caso de las cigüeñas, que hacen ruido golpeando sus picos (crotorar) e incluso, algunas que lo tienen, utilizan sonidos del pico o su canto como el picozapato.

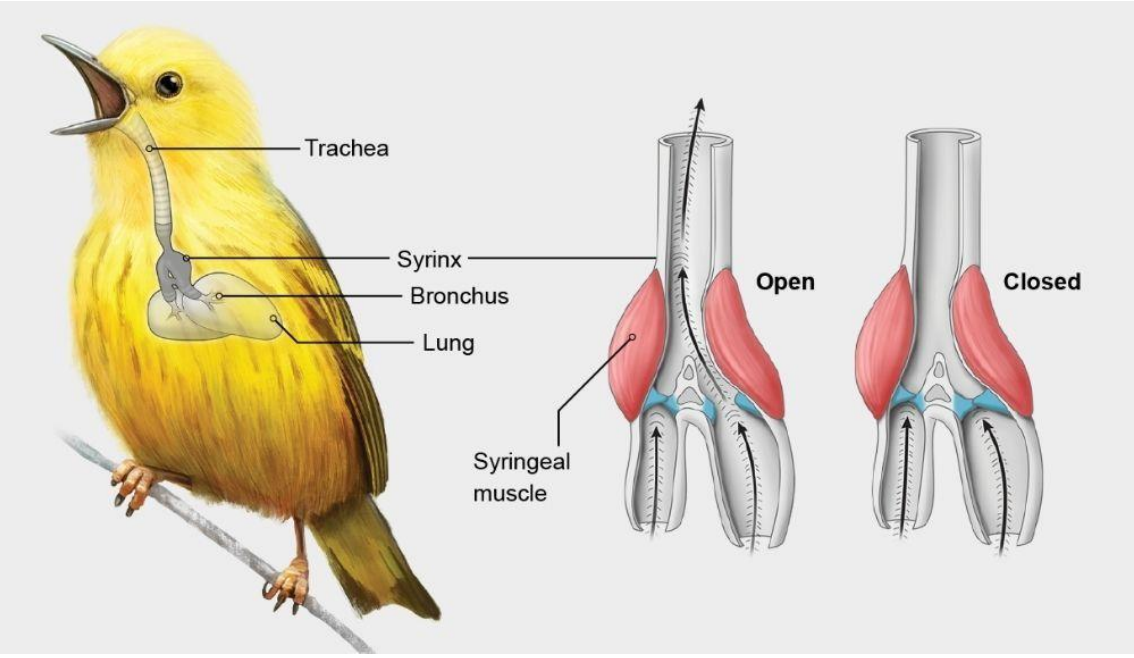

Figura 2.1 Siringe de un ave cantora [37]

2.2 Grabación y obtención de los sonidos

El entorno y la calidad del equipo de grabación influyen mucho en la grabación de los audios. Existen diversos contextos posibles para grabar cantos de pájaros, dado que no es lo mismo grabar un audio en un estudio que grabar un audio en la naturaleza. Normalmente, las grabaciones de cantos de pájaros son realizadas al aire libre, porque los pájaros se comportan de manera natural al encontrarse en su hábitat. Esta práctica es conocida como grabación de campo [2].

Numerosos usuarios realizan grabaciones de campo y suben esas grabaciones a la nube. Algunas de las páginas web más importantes para obtener grabaciones de cantos de pájaros son *kaggle*[1] y *Xeno-canto*[2].

Kaggle tiene varias competiciones relacionadas el problema a resolver. Esas competiciones tienen un conjunto de datos con el sonido de varios pájaros que los participantes deben intentar clasificar. Por ejemplo, la competición *BirdCLEF* [3] ofrece

[1]*Kaggle*: Una página web de Google que permite subir conjuntos de datos y código. Además, se realizan competiciones sobre estos conjuntos de datos [40].

[2] Xeno-canto: Una página web para compartir grabaciones de sonidos de la vida salvaje [20].

un conjunto de datos de sonidos de pájaros cada año cuyo ganador será quien consiga clasificar el mayor número de sonidos correctamente.

En el momento de la realización de este trabajo fin de grado, esta competición está activa con incluso un premio de 15000$ para el primer puesto. El conjunto de datos de esta competición consiste en 32971 ficheros de audio de 397 pájaros de la zona de las Ghats occidentales en la India.

No obstante, existen más competiciones que se han realizado como *MLSP 2013 Bird Classification Challenge* [4] en el año 2013. Esta competición dispone de un conjunto de datos de 645 ficheros de audio de 10 segundos que se repartían entre 19 pájaros de las cordilleras de las Cascadas en Estados Unidos.

Xeno-canto contiene grabaciones de sonido de aves de todo el mundo. Estos audios contienen diferentes metadatos que más adelante utilizaremos. *Xeno-canto* tiene 738144 ficheros de audios que corresponde a 11083 especies de seres vivos [5].

Queremos obtener grabaciones de especies de Albacete, pero no hemos encontrado ningún conjunto de datos que cumpla con este requisito. No obstante, si existen grabaciones de aves en *Xeno-canto* de especies que se encuentran en esta provincia. Por ello, hemos decidido consultar las aves de interés en la página web de la sociedad albacetense de ornitología (S.A.O) [6]. En el Capítulo Capítulo 3, utilizaremos esta información para buscar grabaciones de esas especies y construir nuestro propio conjunto de datos.

2.3 Aprendizaje automático

Es una rama de la inteligencia artificial que busca que un algoritmo aprenda un modelo a partir de un conjunto de datos con los cuáles se entrena, y buscará identificar un patrón o conjunto de patrones con el que puede realizar predicciones sobre nuevos datos. Principalmente, utilizamos datos y sus salidas o etiquetas, si las conocemos, para descubrir las reglas o patrones, a diferencia del paradigma habitual donde a partir de los datos y reglas se obtienen las salidas [7].

Existen diversos tipos de aprendizaje automático, pero los tipos más importantes son:

- Aprendizaje supervisado: El conjunto de datos viene etiquetado con las salidas, siendo esta etiqueta formalmente denominada variable clase u objetivo. Por ejemplo, nuestro conjunto de datos tiene la especie del ave como etiqueta (variable clase) en los ficheros de audio [8].

- Aprendizaje no supervisado: El conjunto de datos no está etiquetado, es decir, solo tienes los datos de entrada. Algunos problemas que resuelve son el agrupamiento y la reducción de dimensionalidad [8].

- Aprendizaje por refuerzo: En este caso, el algoritmo aprende a base de ensayo y error mediante recompensas y castigos. En este escenario, los algoritmos aprenden estrategias para minimizar estas penalizaciones y maximizar las recompensas, con el objetivo de encontrar el mejor ajuste para resolver la tarea solicitada [7].

Nosotros tenemos unos conjuntos de datos etiquetados, es decir, nos centraremos en el paradigma del aprendizaje supervisado.

También, existe otro factor que influye en la elección de los algoritmos:

- Datos estructurados: Siguen una secuencia regular que corresponde a un modelo de datos. Por ejemplo, pueden tratarse como tablas u otros formatos. Algunas ventajas que ofrecen este tipo de datos son el fácil análisis de los datos como el tratamiento de estos.

- Datos no estructurados: No siguen ninguna secuencia, son datos que se encuentran en su formato original, por tanto, no se puede tratar ni analizar de forma sencilla. Este tipo de datos son el 80% de los datos existentes y unos ejemplos de este tipo de datos son los vídeos, imágenes, audios, textos, etc [9].

En nuestro caso, el conjunto de datos son audios etiquetados con la especie a la que pertenecen, por tanto, se trata de un conjunto de datos no estructurados.

El problema de utilizar audios para entrenar nuestros modelos es la extracción de características de estos ficheros de audio. Un algoritmo de aprendizaje supervisado necesita características de los audios para aprender y reconocer sus patrones, pero algoritmos como *random Forest* [10] u otros, necesitan que otros algoritmos extraigan características para aprenderlas.

Por este motivo, utilizaremos redes neuronales, en particular, redes convolucionales, para resolver este problema de clasificación porque son excelentes para datos no estructurados al aprender la extracción de características sin necesidad de otros algoritmos.

2.4 Redes neuronales

Las redes neuronales son un tipo de algoritmo que intenta imitar el funcionamiento de las neuronas de nuestro cerebro. Están conformadas por diversas capas de nodos

(neuronas artificiales) y cada neurona se conecta a otra con un peso y umbral, salvo en la capa de entrada que no tiene umbral. Si la salida de una neurona está por encima del valor umbral especificado, dicha neurona se activa y envía datos a la siguiente capa.

La estructura de una red neuronal está conformada por una capa de entrada, varias capas ocultas y una capa de salida. La capa de entrada tiene un número de neuronas equivalente al número de parámetros de entrada. Las capas ocultas se encargan del aprendizaje de los patrones. Por último, la capa de salida tiene tantas neuronas como salidas necesitamos para resolver el problema [11]. La Figura 2.2 Capas de una red neuronal Figura 2.2 muestra una representación de estas redes neuronales.

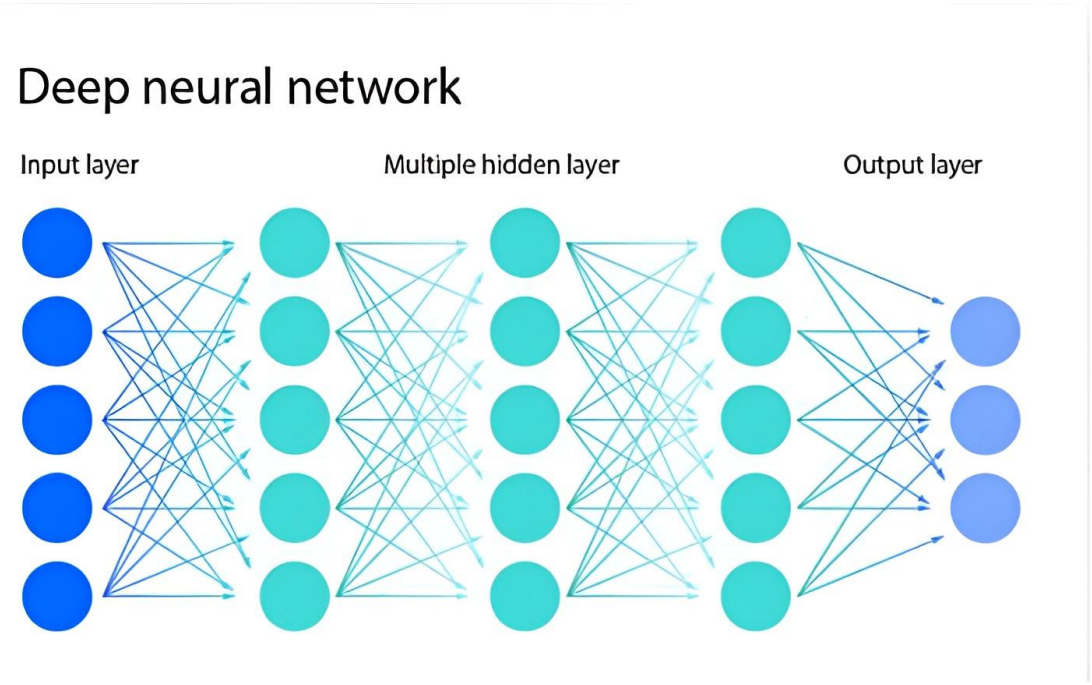

Figura 2.2 Capas de una red neuronal [11]

La primera red neuronal artificial creada fue el perceptrón por Frank Rosenblatt en 1958, sin embargo, la idea de las redes neuronales surgió en 1943 por parte de Warren S. McCulloch y Walter Pits [11]. La Figura 2.3 muestra la arquitectura del perceptrón.

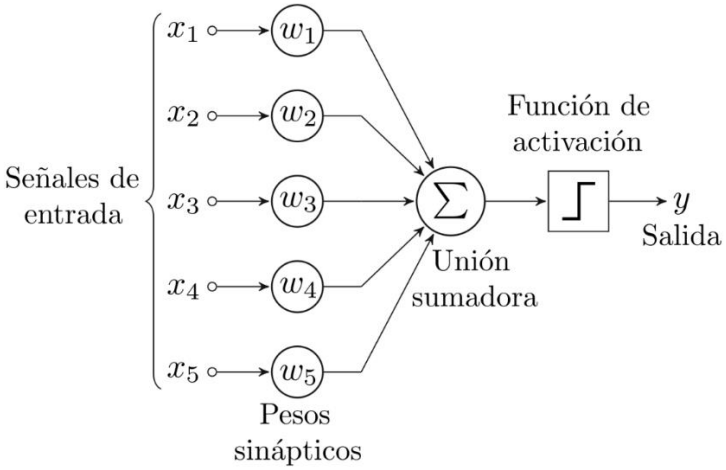

Figura 2.3 Perceptrón

Las redes neuronales no avanzaron hasta 1974 debido a la necesidad de utilizar funciones de activación no lineales para capturar patrones más complejos. No obstante, Paul Werbos y otros investigadores diseñan el algoritmo de retropropagación que permitiría entrenar las redes neuronales con estas funciones de activación no lineales mediante el algoritmo de gradiente-descendiente. Algunas funciones de activación no lineales más destacables son las funciones *ReLU*, *sigmoide* y *tangente hiperbólica*. La Figura 2.4 muestra algunas funciones antes mencionadas y algunas variantes.

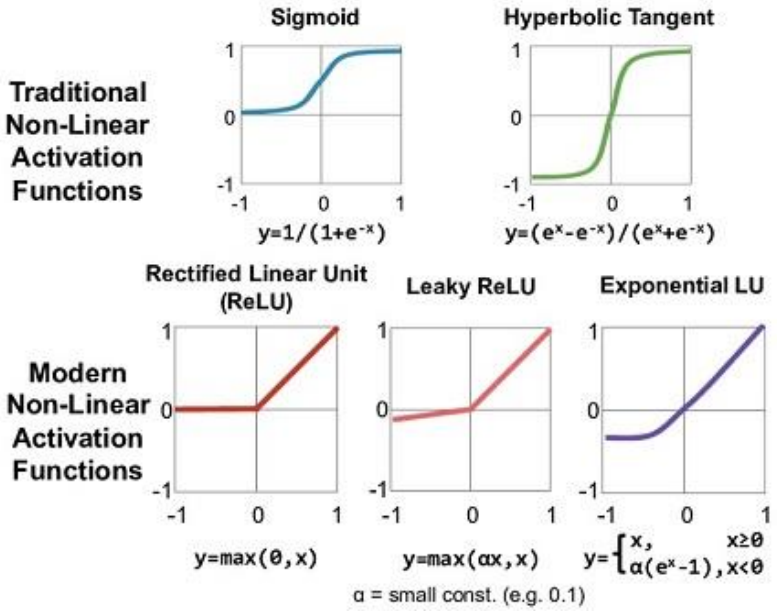

Figura 2.4 Funciones de activación no lineales [12]

Yann LeCun publicó un estudio que muestra el uso de restricciones en la retropropagación y su integración para el entramiento de redes neuronales en 1989 [11].

2.4.1 Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de red especializada en procesar datos que tienen una topología similar a una cuadrícula. En este caso, nosotros utilizaremos estas redes para este proyecto.

La red convolucional recibirá una imagen u otro formato de dato como entrada y mediante unas capas, se extraen las características de los datos para permitir aprender patrones a las redes. Estas capas son llamadas capas de convolución y como su propio nombre indica, realizan una serie de operaciones llamadas convoluciones. Primero, debemos suponer que la imagen que se pasa como entrada es una matriz de 3 dimensiones (RGB), donde cada posición de la matriz representa el color del píxel (0 a 255) de la imagen en una de las dimensiones del RGB. Esto no significa que las redes no sean capaces de trabajar con imágenes de menos o más dimensiones. Las convoluciones consisten en aplicar un filtro (*kernel*), normalmente una matriz de 3x3 con un peso en cada posición, que se aplica en un área de la imagen. Se calcula el producto escalar de esa matriz según los pesos del filtro y los valores de los pixeles del área, asignando ese nuevo valor al píxel de la imagen. Al final, este filtro se va desplazando a otras áreas de la imagen para seguir aplicando las operaciones hasta que el filtro recorre todos los pixeles de la imagen [13].

En resumen, las capas de convolución extraen características de los datos de entrada, actuando como un filtro que reduce los parámetros (pesos) a estimar porque solo dependen del tamaño del *kernel* y no del tamaño de la matriz de los datos de entrada. La Figura 2.5 enseña de forma gráfica un ejemplo de operación de convolución.

Luego, las capas de agrupación se encargan de reducir la dimensionalidad de la entrada (imagen). Las capas de agrupación realizan convoluciones, pero el filtro no tiene pesos porque se realiza una operación como una suma, media, etc. Esto provoca una pérdida de información, sin embargo, reduce la complejidad, el riesgo de sobreajuste y mejora la eficiencia. Esto permite que la imagen se transforme en características de más alto nivel que son aplanadas por las capas de aplanamiento para ser tratadas por las capas totalmente conectadas. La Figura 2.6 muestra la estructura general que las redes convolucionales utilizan.

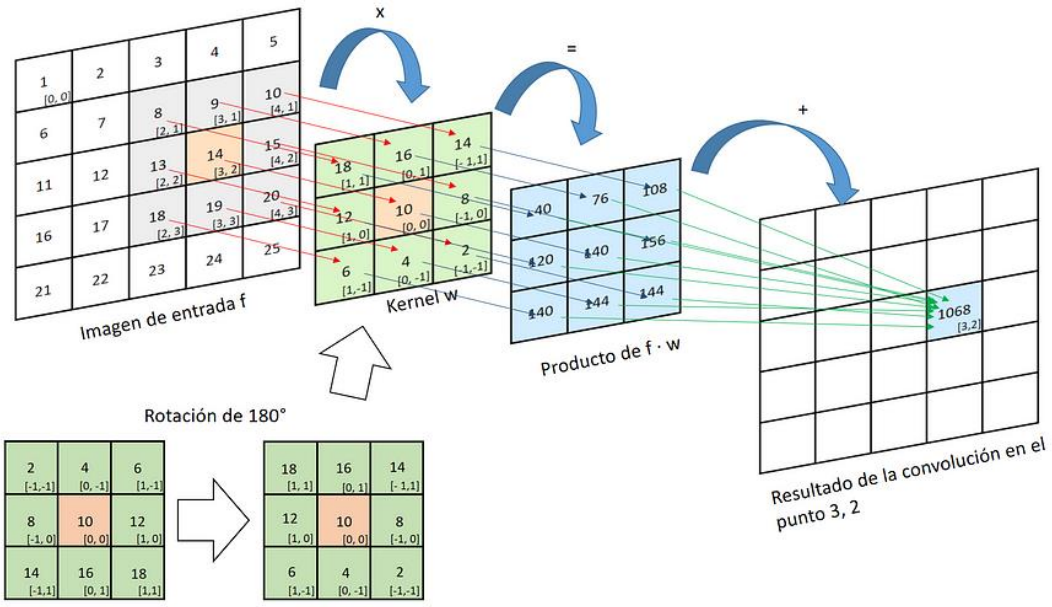

Figura 2.5 Operación de convolución [14]

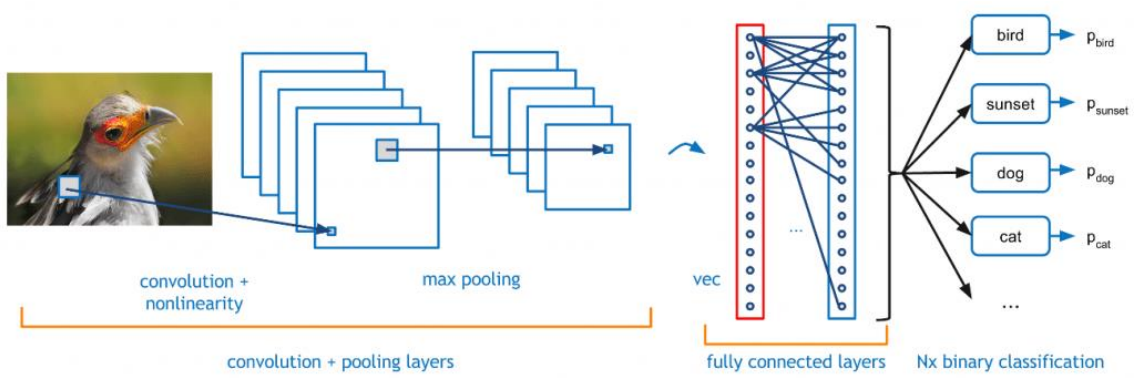

Figura 2.6 Red convolucional completa [15]

2.4.2 Transfer learning

Un modelo pre-entrenado es una red neuronal que ha sido entrenada con un gran conjunto de datos. Normalmente, estos modelos han sido entrenados para realizar tareas de clasificación de gran escala [16].

El *transfer learning* se basa en utilizar un modelo pre-entrenado para extraer características [16]. La extracción de características será mucho mejor debido a que se aprovechan los conocimientos (pesos) que aprendió con más datos. Así, se extraen características de patrones sencillos para entrenar las capas encargadas de la clasificación permitiendo que esas capas reduzcan su tiempo de entrenamiento. Las capas del modelo pre-entrenado se congelan (no aprenden de los datos nuevos) para no perder ese conocimiento adquirido.

Además, se puede realizar una técnica llamada *fine-tuning* para adaptar mejor el modelo pre-entrenado. Esta técnica consiste en permitir el aprendizaje de algunas capas finales del modelo pre-entrenado para que adapten su conocimiento previamente adquirido al problema en particular.

Existen modelos pre-entrenados en librerías como *Tensorflow* [17]. En nuestro caso, utilizaremos estas técnicas más adelante para mejorar el rendimiento de los modelos.

2.5 Evaluación

Para determinar el éxito de la clasificación de los pájaros, necesitamos conocer métricas para analizar de forma correcta el resultado y su rendimiento. A continuación, explicaremos el concepto de matriz de confusión y luego, algunas métricas que se pueden calcular a partir de ella.

La matriz de confusión es una herramienta que nos permite visualizar los aciertos y fallos de nuestro clasificador de forma sencilla [18]. Por ejemplo, una matriz con 2 clases posibles se muestra en la Figura 2.7.

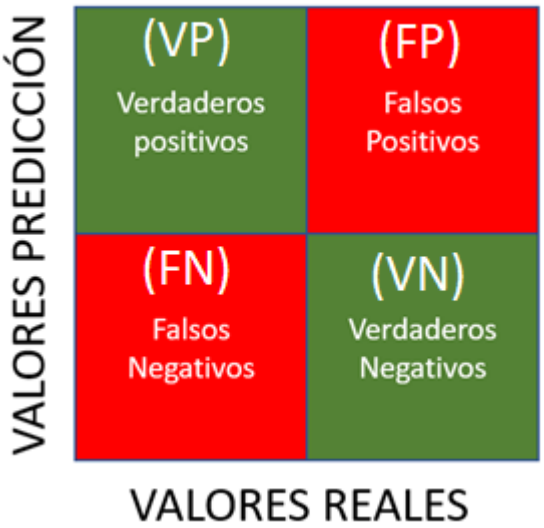

Figura 2.7 Matriz de confusión [18]

Los valores en la diagonal verde son los valores que han sido correctamente clasificados, pero la diagonal roja representa los valores mal clasificados. A partir de este concepto, podemos explicar las métricas que se pueden obtener de él:

- Tasa de acierto: Se refiere a predicciones que han sido clasificadas correctamente, es decir, se calcula con los valores totales de los verdaderos positivos y negativos entre el número total de predicciones.

- Precisión: Se refiere al valor de verdaderos positivos entre los valores positivos totales, es decir, el porcentaje de éxito de los valores de la clase positiva ($\frac{VP}{VP+FP}$).

- Sensibilidad: Es la proporción de casos positivos clasificados correctamente, es decir, es el valor de los verdaderos positivos entre los verdaderos positivos y los falsos negativos ($\frac{VP}{VP+FN}$).

- *F1-Score*: Es una métrica que combina la precisión y sensibilidad. Se calcula como

$$F1 = \frac{2 * Precisión * Sensibilidad}{Precisión + Sensibilidad}$$

2.6 CRISP-DM

CRISP-DM es una metodología para desarrollar un proyecto de minería de datos. Como hemos mencionado anteriormente, nosotros aplicaremos esta metodología a este proyecto [19].

Comprende estas fases:

- Comprensión del problema: Esta fase se centra en establecer los objetivos del proyecto, entendiendo los objetivos básicos del proyecto y el contexto de estos.

- Comprensión de datos: Esta fase consiste en explorar los datos, sus estructuras y su recolección. En nuestro proyecto, esta fase estará muy relacionada con la recolección de datos.

- Preparación de datos: Los datos serán procesados y analizados para prepararlos para el modelo en esta fase. Por ejemplo, nosotros desarrollaremos en esta fase el preprocesamiento de los datos.

- Modelado: En esta fase, se estudian y eligen las técnicas o modelos que se utilizan para resolver el problema. En nuestro caso, será la elección de los distintos modelos y algoritmos, y en algún caso, la construcción de alguno de ellos.

- Evaluación: Los modelos deben ser evaluados para validar su resultado y decidir como continuar. Esta fase es muy importante porque marcará el éxito de nuestros modelos o no. En nuestro proyecto, realizaremos distintas pruebas sobre los modelos para validar su éxito y resultados.

- Despliegue: Los modelos anteriormente seleccionados son desplegados para que el usuario final pueda utilizarlos. Esta fase es el final del proyecto si todos los objetivos se han cumplido correctamente. En nuestro caso, incorporaremos el mejor modelo a nuestra aplicación.

La metodología CRISP-DM es un proceso iterativo que puede ir retrocediendo o avanzando entre fases. El proceso se puede observar en la Figura 2.8.

Figura 2.8 Metodología CRISP-MD

Capítulo 3

Recolección de los datos

3.1 Introducción

La recolección de los datos es una tarea muy importante en cualquier proyecto de ciencia de datos puesto que son necesarios para analizar el problema y entrenar los modelos que solucionarán el problema. En este caso, nosotros recolectaremos nuestros datos mediante la API de *Xeno-canto* [20], porque nos permitirá descargarnos cualquier audio de un pájaro. Además, nos permitirá elegir cuántas especies distintas queremos considerar en la variable clase.

3.2 Peticiones a la API de *Xeno-canto*

La API de *Xeno-canto* tiene distintos parámetros para realizar una petición a su base de datos de registros de audios. Nosotros utilizaremos el nombre científico de las aves para realizar la petición, sin embargo, existen otros parámetros como la región.

Utilizando la librería *request* de Python, realizaremos una petición "*https get*" para obtener los registros de audio disponibles de esa ave. Introducciones una *url* con los parámetros del ave (nombre científico) y obtenemos en el mensaje de respuesta de todos los identificadores de los audios disponibles.

Un ejemplo de *url* para obtener los registros del gorrión común es "https://www.xeno-canto.org/api/2/recordings?query=Passer%20domesticus". Sin embargo, aún no tenemos esos ficheros descargados, por tanto, utilizamos los identificadores de estos para descargarnos esos audios correspondientes al pájaro. La *url* de la nueva petición "*get*" quedaría así "https://www.xeno-canto.org/id_fichero/download", donde *id_fichero* es el identificador del fichero que se va a descargar.

3.3 Filtrado de los registros de audio

No queremos descargarnos todos los audios de pájaros sin antes hacer un filtrado para asegurarnos de que cumplan ciertas condiciones. La API nos responde con un JSON de todos los audios con sus respectivos metadatos. Algunos de esos metadatos más relevantes son:

- Id: Identificador del audio que utilizamos para descargarlo.
- Lic: Licencia del audio puesta por el usuario.
- Length: Duración del audio.
- Also: Otros pájaros (distinta especie o subespecie) en el audio.
- Quality (q): Calidad del audio, puntuada por los usuarios.
- Auto: Usuario que grabó el audio.

La respuesta JSON lo transformamos en varios diccionarios (uno por cada audio) que posteriormente filtraremos para seleccionar los audios que nos interesen. Ahora, mencionaremos las condiciones que hemos utilizado para filtrar la descarga de los audios:

- El primer filtro que realizamos se basa en seleccionar solo los audios que tengan la calidad máxima, es decir, la calidad “A”.

- El segundo filtro que aplicamos consiste en solo descargar los audios que no contengan ningún otro pájaro (distinta especie o subespecie).

- El último filtro consiste en confirmar que la licencia Creative Commons del audio no sea BY-NC-ND ni BY-ND, ya que no permiten hacer transformaciones ni derivaciones de sus audios y no podríamos aplicar el preprocesamiento a esos audios.

Por ejemplo, si queremos descargar ficheros de audios del ave *Alauda arvensis* (Alondra común), realizamos una petición a la API de *Xeno-canto* con el nombre científico del ave con la *url* “https://xeno-canto.org/api/2/recordings?query=Alauda%20arvensis”. Como respuesta, obtenemos la Figura 3.1.

Cuando hemos obtenido la respuesta, filtramos la respuesta utilizando las condiciones antes mencionadas. Los atributos que se utilizan en las condiciones se pueden observar en la leyenda de la Figura 3.1. En resumen, se utiliza la calidad del audio para la primera condición, el parámetro “*also*” para la segunda condición y la licencia del audio para la tercera condición.

Al final, utilizamos el enlace de descarga para descargarnos ese archivo de audio y empezamos con el siguiente archivo de audio.


```
"numRecordings": "3049",
"numSpecies": "1",
"page": 1,
"numPages": 7,
"recordings": [
  {
    "id": "911897",
    "gen": "Alauda",
    "sp": "arvensis",
    "ssp": "",
    "group": "birds",
    "en": "Eurasian Skylark",
    "rec": "Lars Edenius",
    "cnt": "Sweden",
    "loc": "Drakamöllan, Kristianstad Municipality, Skåne County",
    "lat": "55.7589",
    "lng": "14.1338",
    "alt": "50",
    "type": "reclamo, canto",
    "sex": "",
    "stage": "",
    "method": "field recording",
    "url": "//xeno-canto.org/911897",
    "file": "https://xeno-canto.org/911897/download",
    "file-name": "XC911897-Sånglärka_14.mp3",
    "sono": {
      "small": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/ffts/XC911897-small.png",
      "med": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/ffts/XC911897-med.png",
      "large": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/ffts/XC911897-large.png",
      "full": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/ffts/XC911897-full.png"
    },
    "osci": {
      "small": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/wave/XC911897-small.png",
      "med": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/wave/XC911897-med.png",
      "large": "//xeno-canto.org/sounds/uploaded/MMEJYLOPDO/wave/XC911897-large.png"
    },
    "lic": "//creativecommons.org/licenses/by-nc-sa/4.0/",
    "q": "A",
    "length": "0:41",
    "time": "07:00",
    "date": "2024-06-07",
    "uploaded": "2024-06-10",
    "also": [],
    "rmk": "SNR 55dB (ISO/ITU)",
    "bird-seen": "sí",
    "animal-seen": "sí",
    "playback-used": "no",
    "temp": "",
    "regnr": "",
    "auto": "no",
    "dvc": "",
    "mic": "",
    "smp": "48000"
  },
  {
    "id": "911894",
```


Leyenda

- Id del archivo
- Enlace de descarga
- Licencia del archivo de audio
- Calidad del audio
- Duración del audio
- Otras especies (distinta especie o subespecie) en el audio

Figura 3.1 Respuesta de https://xeno-canto.org/api/2/recordings?query=Alauda%20arvensis

3.4 Generación de los conjuntos de datos

El conjunto de datos tendrá 50 clases. Las clases seleccionadas serán distintas aves que se pueden encontrar por Albacete o sus alrededores. Utilizando la página web de la Sociedad Albacetense de Ornitología podemos conocer que especies de aves se encuentran en Albacete [6].

El conjunto de datos será generado mediante el proceso de las Secciones 3.2 y 3.3. Descargaremos 125 ficheros de audios para cada clase cuya distribución será 80% para el conjunto de entrenamiento, 10% para el conjunto de validación y 10% para el conjunto de prueba (*test*). Las aves de Albacete que han sido seleccionadas para su clasificación se muestran en la Figura 3.2.

Figura 3.2 Matriz de aves seleccionadas

Capítulo 4

Exploración de los datos

4.1 Introducción

En este capítulo exploraremos el conjunto de datos generado en la Sección 3.4, observaremos sus distribuciones y la naturaleza de los ficheros de audio descargados.

4.2 Exploración de los conjuntos de datos

Hemos decidido descargar 125 archivos de audios por cada clase, es decir, se deberían descargar 6250 archivos de audio. Sin embargo, no todas aves tienen disponibles 125 audios en *Xeno-canto*, por tanto, algunas clases solo tendrán todos los audios disponibles, provocando desbalanceo entre clases.

Las distribuciones de las clases en los distintos conjuntos de datos generados para el entrenamiento, validación y prueba se muestran en la Figura 4.1, Figura 4.2 y Figura 4.3. Con esta configuración, el conjunto de entrenamiento tiene 4236 audios, el conjunto de validación tiene 508 audios y el conjunto de prueba tiene 508 audios.

Los audios se encuentran en formato *.mp3* y tienen distinta duración. El formato *.mp3* representa el audio mediante una serie temporal de números reales pertenecientes a un intervalo de -32000 a 32000. No obstante, existen algunos audios en formato *.wav* que representa el audio mediante una serie temporal de números reales pertenecientes a un intervalo de -16000 a 16000. En nuestro caso, estableceremos todos los ficheros de audio en el formato *.wav* porque es un formato más simple para entrenar los modelos.

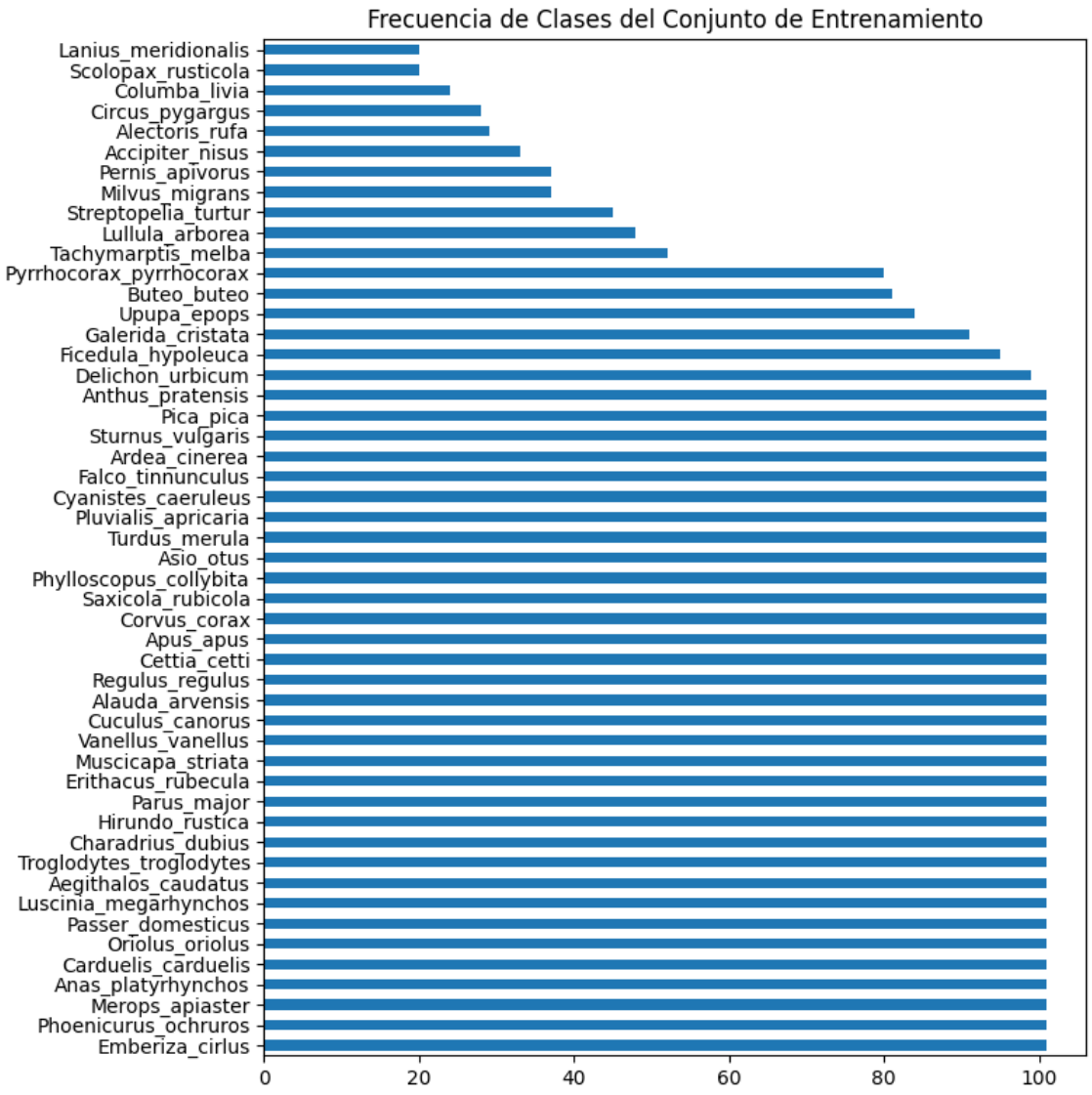

Figura 4.1 Histograma del conjunto de entrenamiento

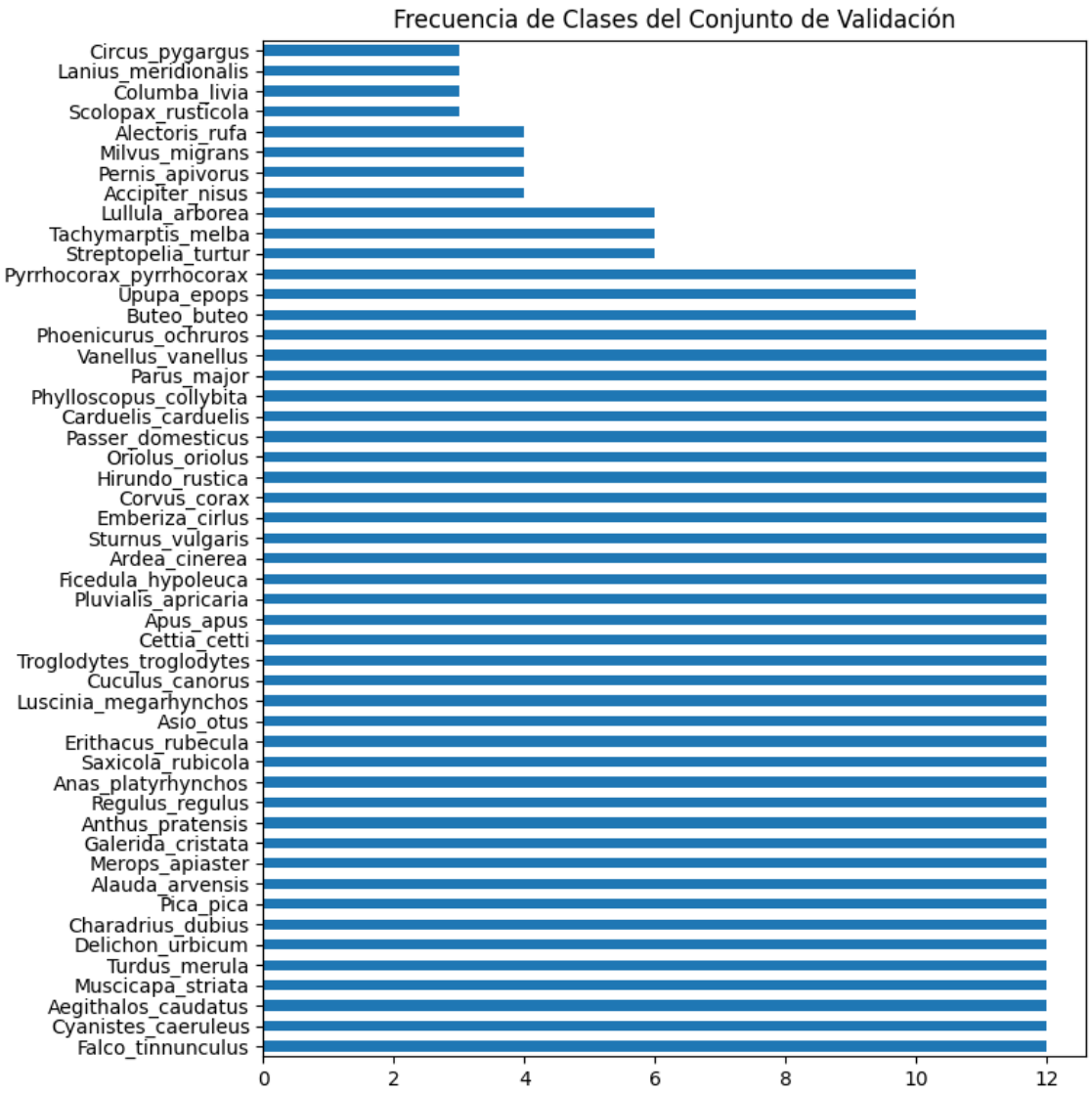

Figura 4.2 Histograma del conjunto de validación

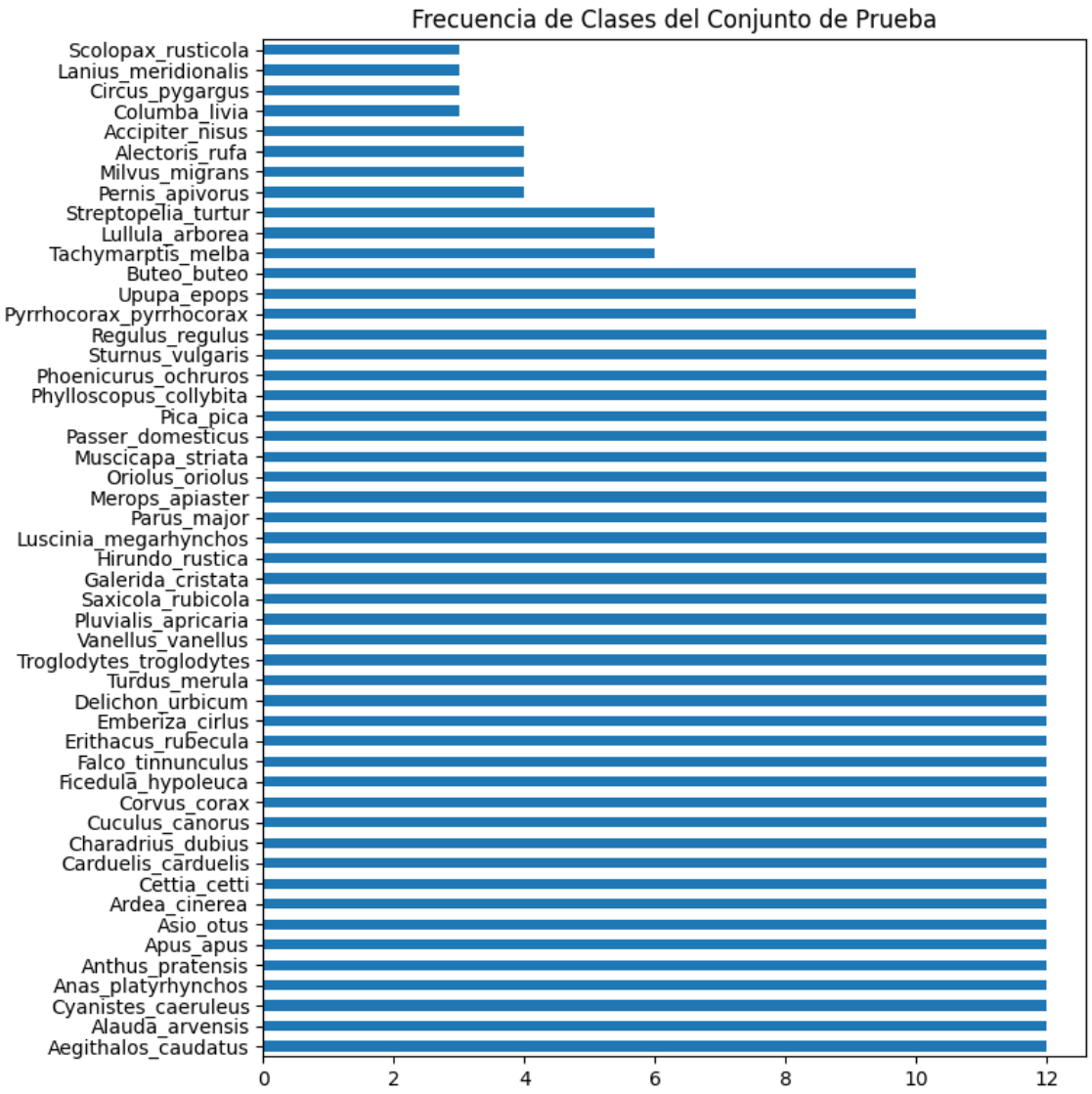

Figura 4.3 Histograma del conjunto de prueba

Capítulo 5

Preprocesamiento

5.1 Introducción

Los audios deben ser preprocesados para que el modelo sea capaz de manejar esos datos y muestre un mejor resultado. Nombraremos dos formas de preparar los ficheros de audios para que nuestro modelo sea capaz de utilizarlos y aprenda de ellos.

La primera forma consiste en decodificar los audios para convertirlos en X números reales, es decir, se utilizan series temporales de estos valores para entrenar el modelo. La segunda forma consiste en decodificar los audios y, con su serie temporal, se obtiene el espectrograma para luego clasificar esa imagen del espectrograma. Nosotros utilizaremos la segunda forma para entrenar nuestros modelos. No obstante, también debemos aplicar otros procesos que más adelante explicaremos.

5.2 Normalización de los audios

Es importante normalizar y realizar ciertos procesos de tratamiento de sonido para facilitar el trabajo de otros procesos más importantes de preprocesamiento. El primer proceso que llevaremos a cabo, se llama normalización.

La normalización del audio consiste en establecer los valores de la serie temporal en forma de onda entre -1 y 1. En la Figura 5.1 se muestra un ejemplo del gorrión común.

Como se puede apreciar en la Figura 5.1, no hay mucha diferencia en la forma de la onda, pero este proceso mejora la generación de los espectrogramas. Además, siempre es una buena práctica normalizar los datos para el aprendizaje automático.

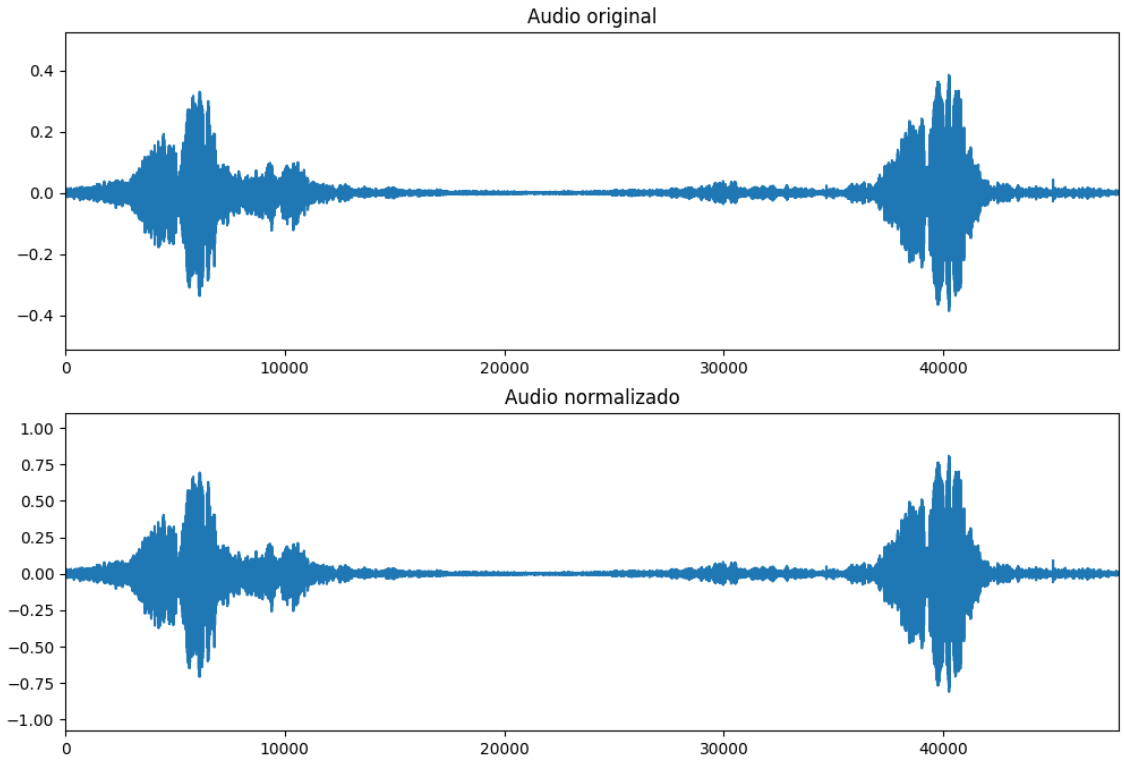

Figura 5.1 Diferencia entre un audio sin normalizar y normalizado

No obstante, esta normalización puede causar problemas si la onda no es simétrica respecto a 0 por ello, antes de normalizar la onda, nos aseguramos de centrar la onda respecto 0 mediante la resta del promedio de la onda.

Por último, cambiaremos la tasa de muestreo de los audios para reducir la complejidad de los cálculos de los siguientes procesos. Además, estandarizamos la tasa de muestreo en todos los audios. La tasa de muestreo nos indica la cantidad de muestras que se toman por segundo [21] y si se reduce, la calidad del audio disminuye, sin embargo, es algo necesario para procesar posteriormente los audios. Nosotros estableceremos la tasa de muestreo a 16000 kHz porque otros modelos como *YAMNET* de Google [22] establecen esa tasa de muestreo. Realizamos estas transformaciones mediante la librería *Librosa* [23].

5.3 Reducción del ruido

La reducción del audio es algo necesario porque muchos audios tienen ruido ambiental que impide que los espectrogramas muestren de forma clara el canto de los pájaros. Utilizaremos una librería llamada *noisereduce* [23] que nos permite utilizar un algoritmo estacionario de reducción de ruido. En la Figura 5.2 se muestran los resultados de la reducción del ruido del gorrión común.

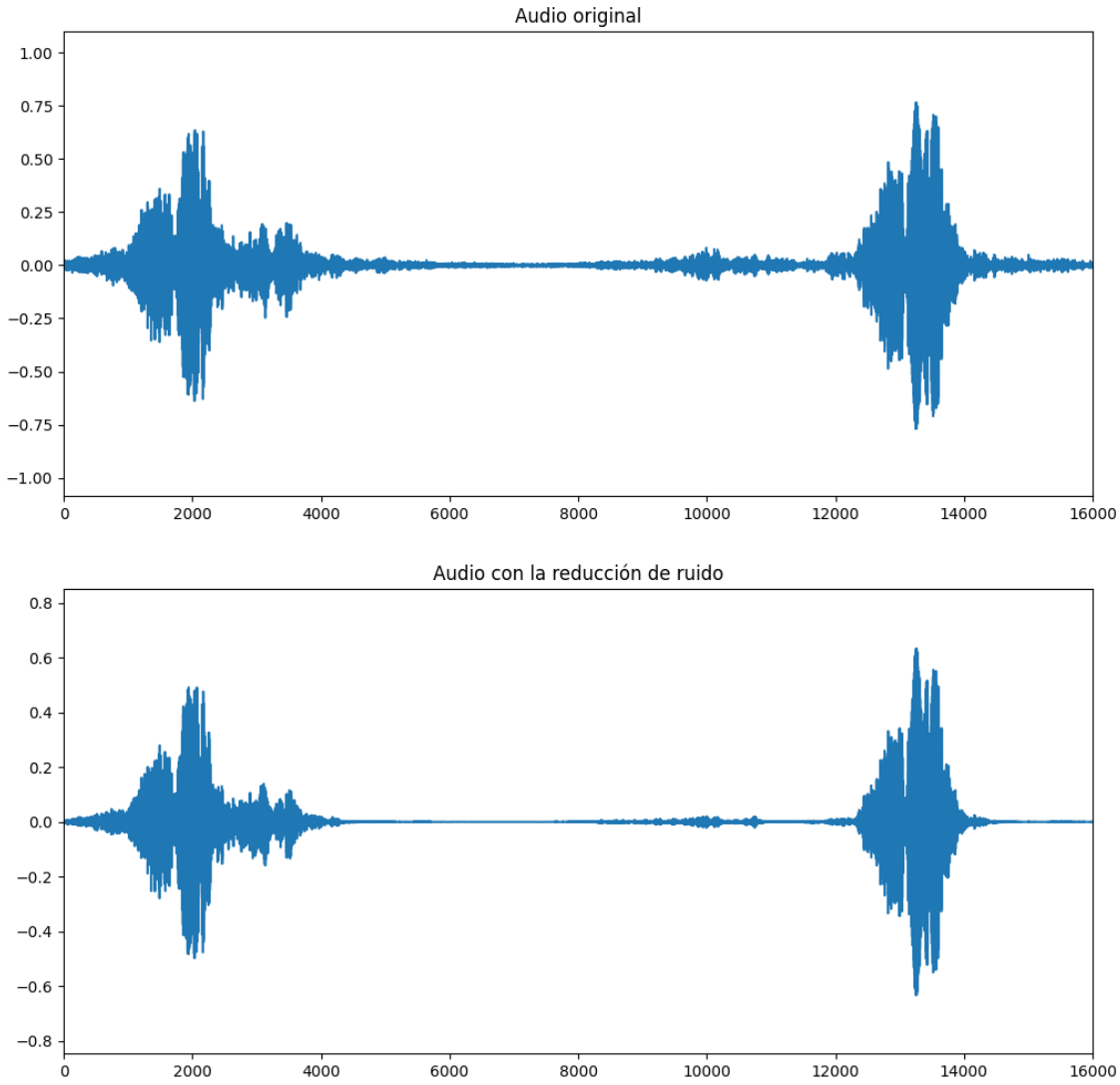

Figura 5.2 Forma de onda con ruido reducido

5.4 Generación de los espectrogramas

En este punto, ya tenemos los audios codificados como números reales, es decir, tenemos el sonido en forma de onda, pero como hemos mencionado anteriormente necesitamos el espectrograma. Transformaremos esa onda para guardar su imagen.

Un espectrograma es una representación visual que permite identificar las diferentes variaciones de la frecuencia y la intensidad del sonido a lo largo de un periodo de tiempo [24]. Esto ayuda a la red neuronal a extraer características de forma más efectiva del sonido, es decir, pasamos de un problema de clasificación de audio a un problema de clasificación de imágenes.

No obstante, para generar el espectrograma, primero debemos separar la onda del sonido en distintas ventanas. Las ventanas tendrán dos parámetros, su tamaño y el índice de solapamiento. Dependiendo de estos valores, los espectrogramas tendrán

mayor calidad o no. Luego, se aplica una ventana de *Hamming*[3] o *Blackman*[4] a cada ventana obtenida y posteriormente, se aplica la transformada Rápida de Fourier (FTT) para convertir la señal del dominio del tiempo al dominio de la frecuencia.

Por último, se calcula la magnitud del espectro con el resultado de la FTT para conseguir el espectrograma. Por suerte, existen librerías de Python que ya realizan esta labor. Las librerías que nosotros utilizamos son *Librosa* [25] y *Matplotlib* [26]. En la Figura 5.3, se muestra un ejemplo del sonido del gorrión común con su forma de onda y espectrograma.

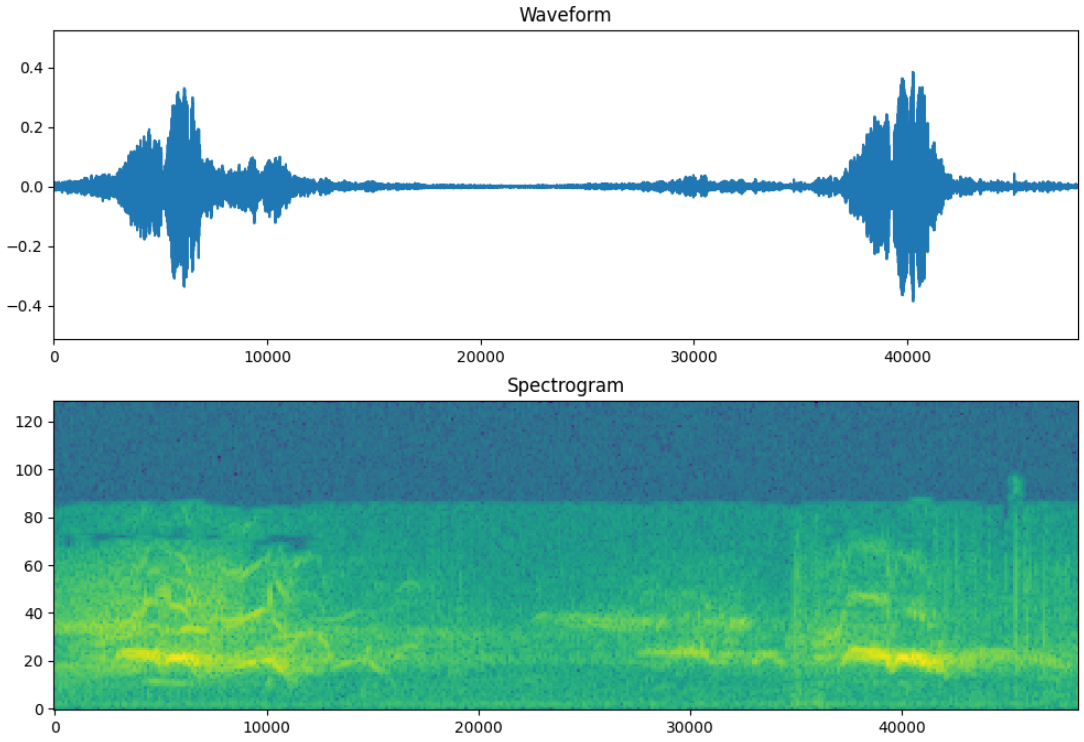

Figura 5.3 Ejemplo de la transición de forma de onda a espectrograma

Con esto, ya podemos transformar los espectrogramas a la *escala de mel* que, para aprendizaje automático de sonidos, mejora el resultado [27]. La *escala de mel* consiste en aplicar una serie de filtros para llevar el audio a una escala de frecuencias que son muy parecidas a nuestra percepción auditiva. Podemos ver el resultado de aplicarlo al sonido anterior en la Figura 5.4.

[3] *Haming:* Esta función de ventana tiene una forma sinusoidal, provocando que esta función sea mejor para medidas de ruido donde se desea una mejor resolución de frecuencia [41].

[4] *Blackman*: Esta función de ventana es muy similar a la ventana de *Hamming*, pero su espectro tiene un pico más amplio [41].

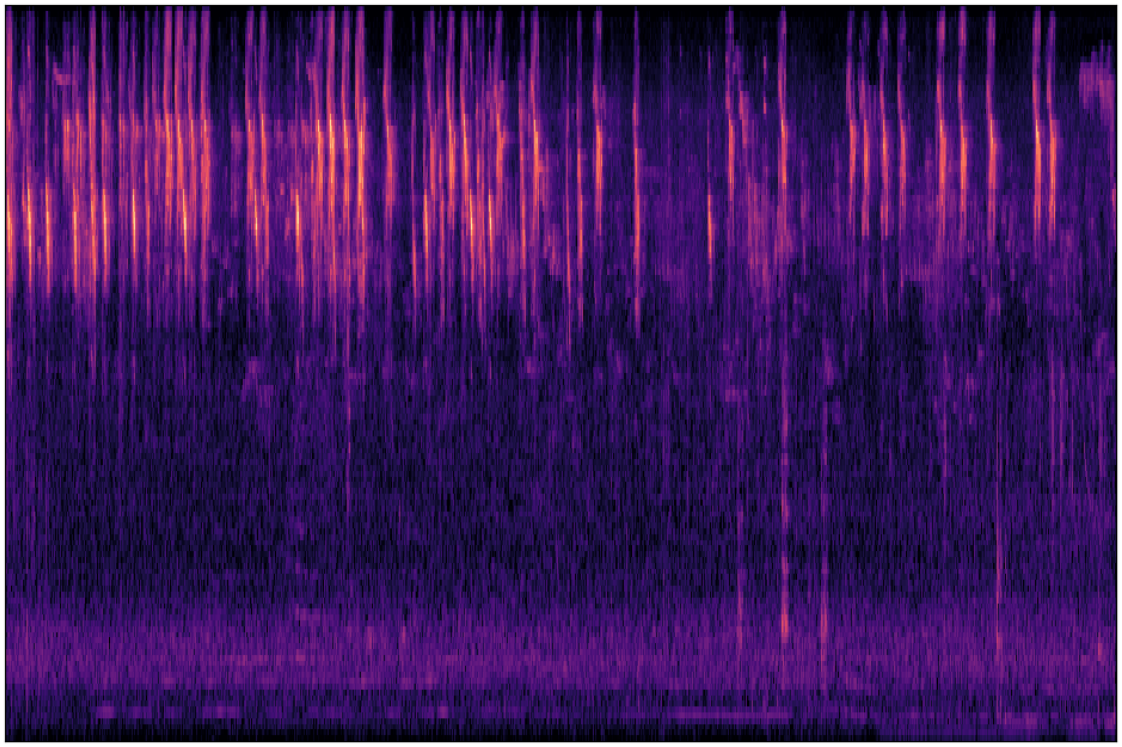

Figura 5.4 Espectograma en la escala mel

Finalmente, ya tenemos las imágenes de los espectrogramas que utilizaremos para empezar el entrenamiento de los modelos, pero aún se pueden mejorar estos audios para obtener mejores espectrogramas (de cara al aprendizaje/clasificación). Las imágenes creadas tienen un tamaño de 255 x 255 x 3 porque es un tamaño estándar y fácil de manejar. Las imágenes de los espectrogramas son imágenes RGB debido a que el color es muy importante para permitir aprender patrones a la red convolucional.

La Figura 5.5 muestra el resultado del preprocesamiento descrito en las Secciones 5.2 y 5.3 sobre tres audios de pájaros distintos, elegidos aleatoriamente. Las aves seleccionadas son la golondrina común, busardo ratonero y paloma doméstica. Enseñamos la forma de onda, el espectrograma normal y en *escala de mel* de los tres audios para evidenciar las diferencias entre las especies.

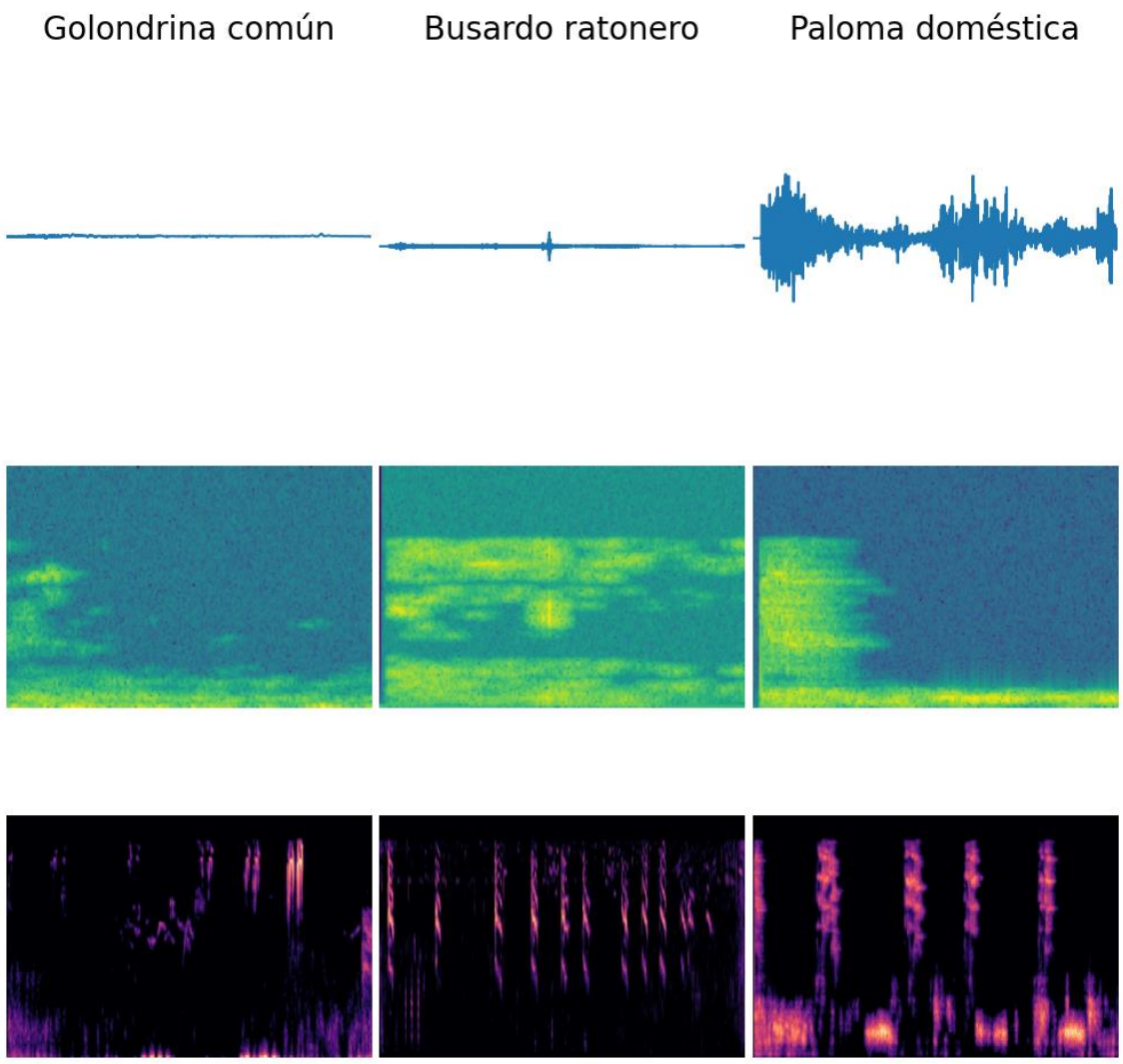

Figura 5.5 Ondas y espectrogramas de tres aves

5.5 División del audio

Los audios tienen distinta duración, haciendo que los espectrogramas se distorsionen entre los audios con una larga duración y los audios con una corta duración. Esto empeora el aprendizaje de las redes neuronales y reduce el éxito en la validación. Por ejemplo, la Figura 5.6 muestra dos audios con distinta duración en forma de espectrograma en *escala de mel*.

Los espectrogramas de la Figura 5.6 son muy diferentes debido a la duración del audio, provocando que la imagen del espectrograma se distorsione y empeore el resultado de la clasificación. Sin embargo, podemos apreciar algo llamativo en los espectrogramas, se observan picos o manchas en algunas zonas. Los picos o manchas señalan los momentos donde el pájaro ha cantado, es decir, solo hay 2 picos o manchas en el espectrograma porque el gorrión solo canta dos veces. Hemos marcado esos picos con unos círculos verdes en la Figura 5.7. También, se puede apreciar un pico en medio del

espectrograma que se trata de ruido del ambiente detectado como sonido de pájaros. Lo hemos marcado con un círculo rojo.

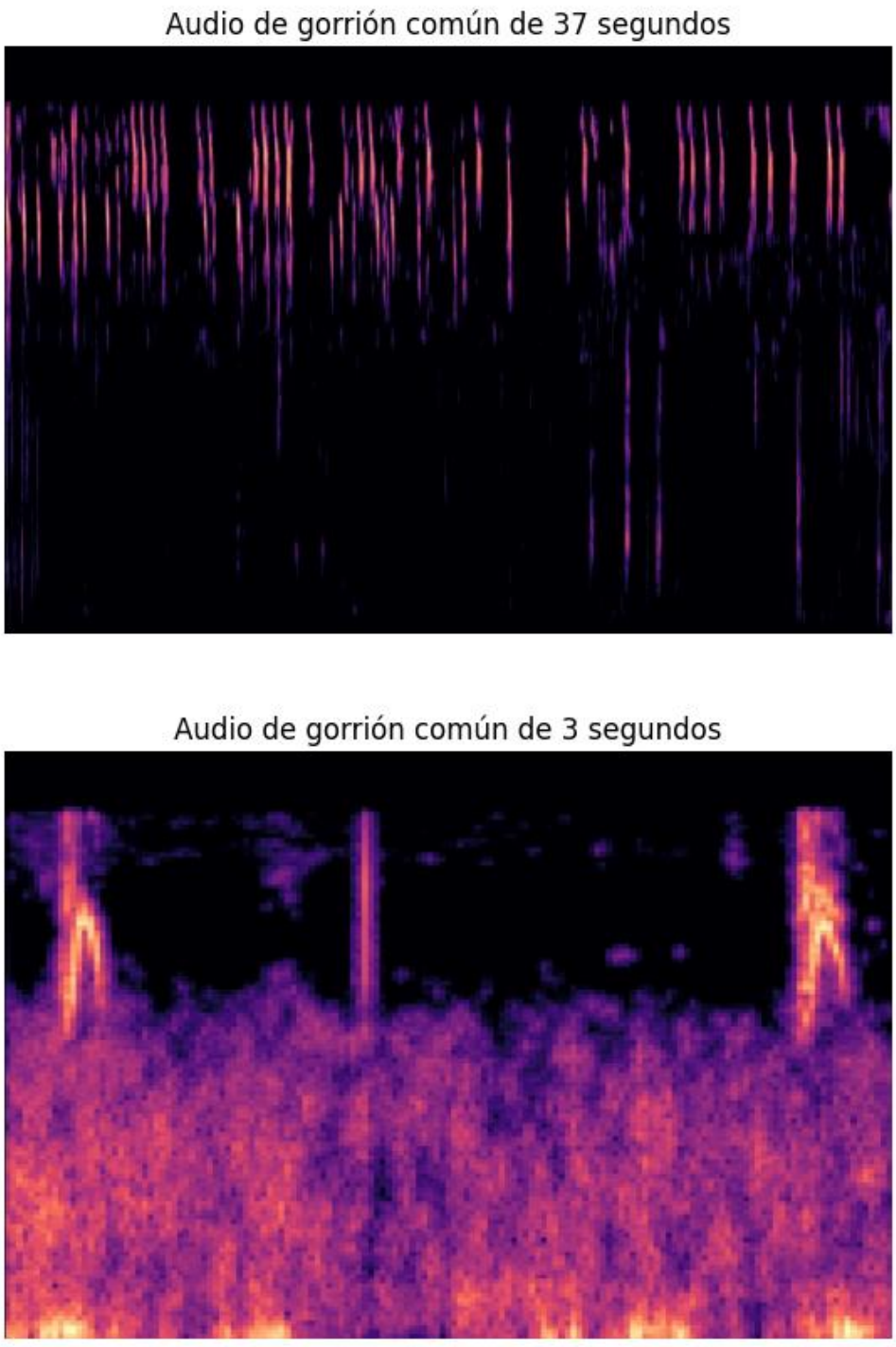

Figura 5.6 Espectrogramas en escala Mel con distintas duraciones

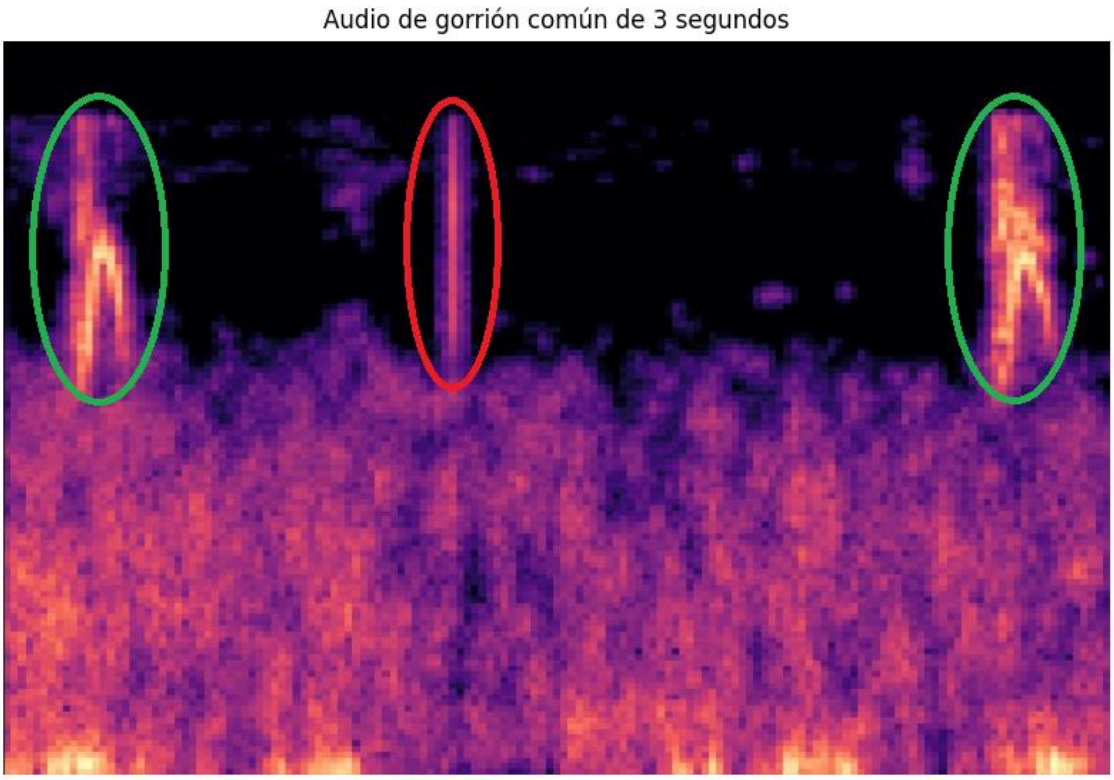

Figura 5.7 Picos del espectrograma en la escala Mel

Sabiendo esto podemos identificar esos picos y procesarlos para obtener muchas más imágenes de un solo espectrograma. Además, la duración ya no distorsionaría las imágenes porque solo extraemos los picos encontrados, por tanto, cada audio procesado será dividido en X partes que detectemos como picos. Utilizaremos la librería *Librosa* para extraer estos fragmentos.

No obstante, no es perfecto porque algunos fragmentos obtenidos se tratan de ruido detectado como picos. Esto sucede porque determinar un valor de decibelios para identificar el silencio es muy complicado porque la intensidad de los sonidos varía según los audios. Además, el algoritmo de reducción de ruido no borra totalmente el ruido y algunos fragmentos se ven algo distorsionado por el ruido. La Figura 5.8 muestra los fragmentos obtenidos en el audio de 3 segundos anterior.

Se puede apreciar que el fragmento 2 se trata de ruido de la Figura 5.8, pero el resto de los fragmentos ha sido extraído correctamente con algo de ruido por debajo. A continuación, mostraremos los fragmentos obtenidos por el audio de 37 segundos en la Figura 5.9.

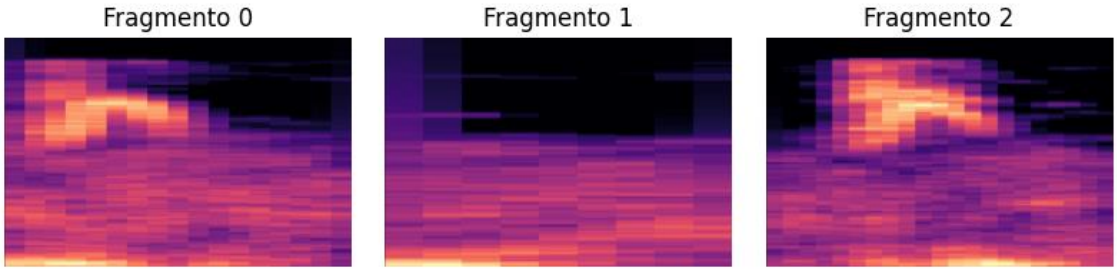

Figura 5.8 Fragmentos obtenidos de la Figura 5.7

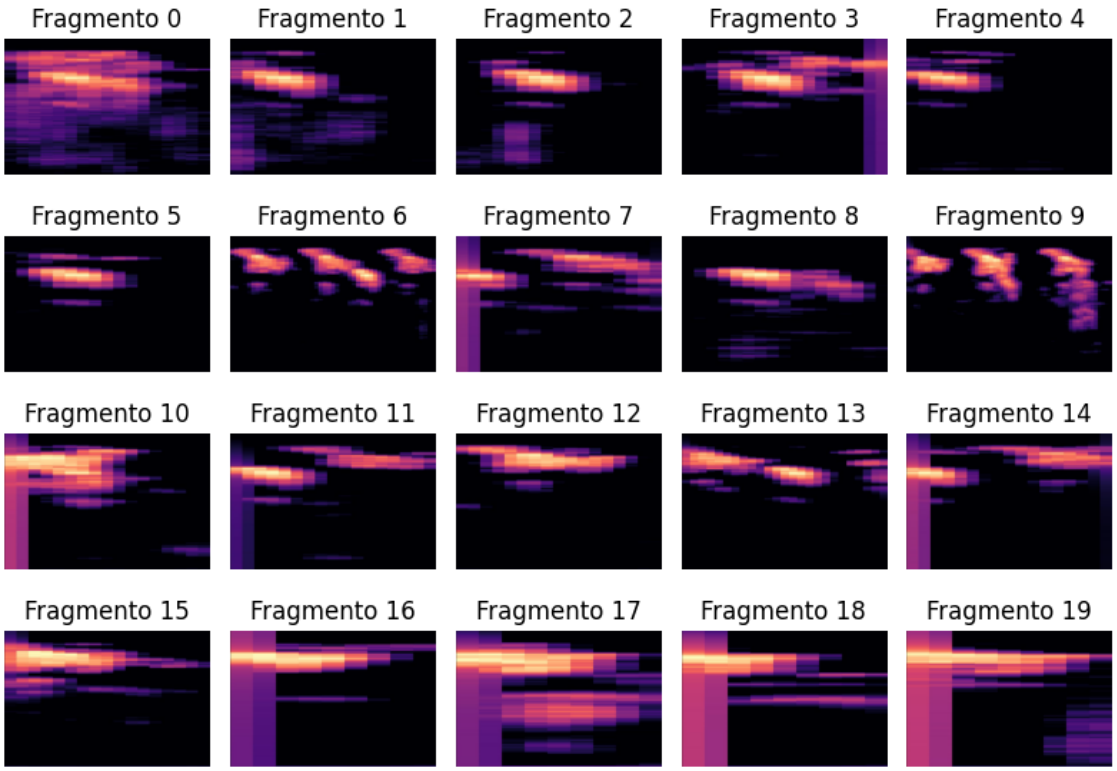

Figura 5.9 Fragmentos obtenidos del audio de 37 segundos

Algunos fragmentos están algo distorsionados, pero se puede ver que tienen cierta semejanza. También, podemos detectar otro problema que consiste en que otros pájaros (misma especie) pueden cantar mientras el principal canta, por tanto, algunos fragmentos tienen más manchas, haciendo que sea más complejo para la red neuronal. Esto sucede porque las respuestas de la API no tienen un parámetro para indicar que solo es un único pájaro de esa especie, es decir, somos capaces de filtrar los audios si contiene sonidos de pájaros de distinta especie (gracias al parámetro "*also*"), pero si contiene sonidos de pájaros de la misma especie y subespecie, no existe ningún parámetro que lo indique.

Nos gustaría mencionar que también, podríamos haber limitado la duración del audio para obtener solo audios con una duración homogénea, pero de esta forma la cantidad de audios disponibles se reduce notablemente y por eso, hemos optado por esta decisión.

Ahora, mostraremos las distribuciones resultantes al aplicar la división de los audios en la Figura 5.10, Figura 5.11 y Figura 5.12. El conjunto de entrenamiento con esta configuración tiene 50941 imágenes, el conjunto de validación tiene 5624 imágenes y el conjunto de prueba tiene 5582 imágenes.

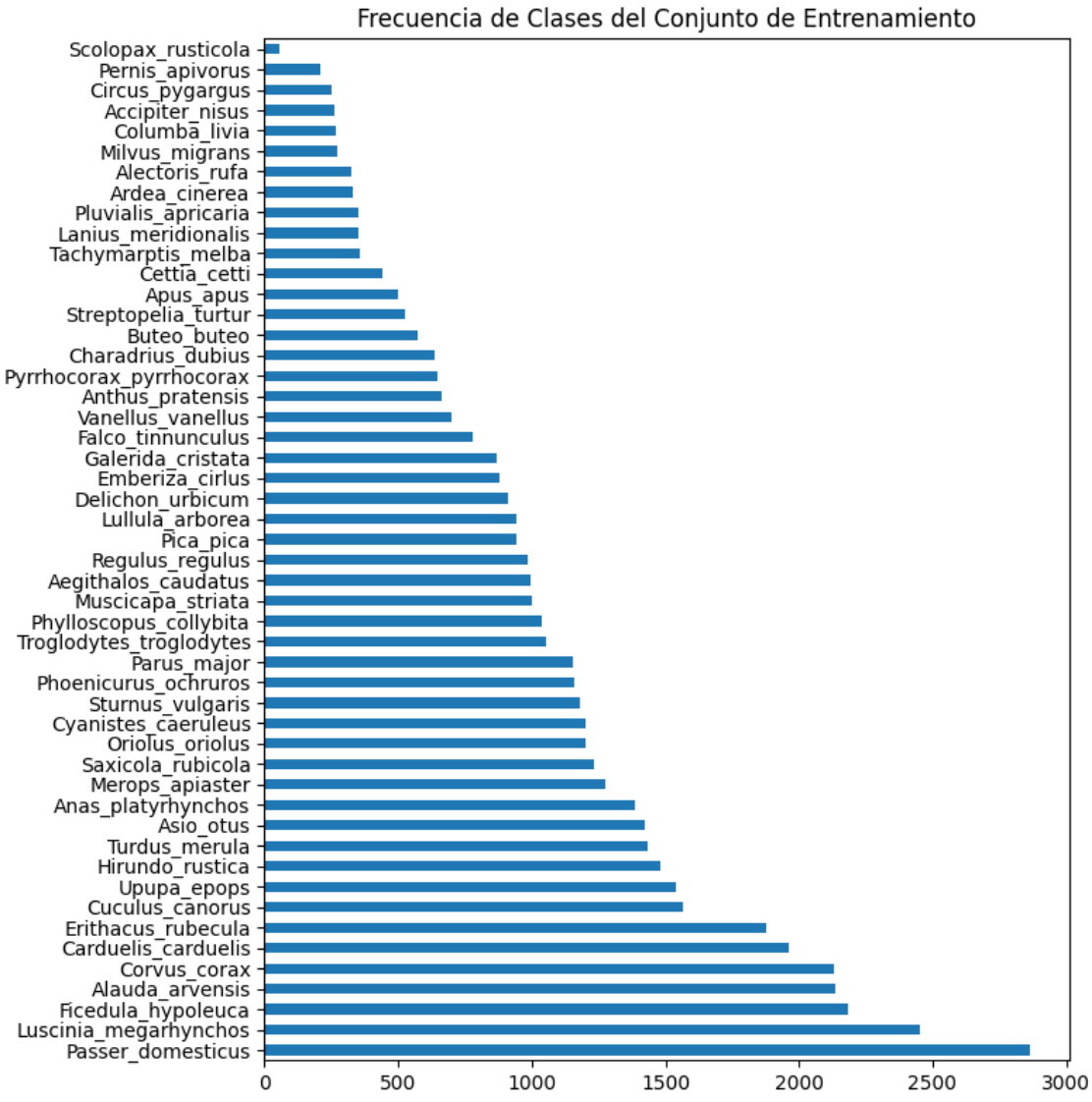

Figura 5.10 Histograma del conjunto de entrenamiento con división y sin aumento de datos

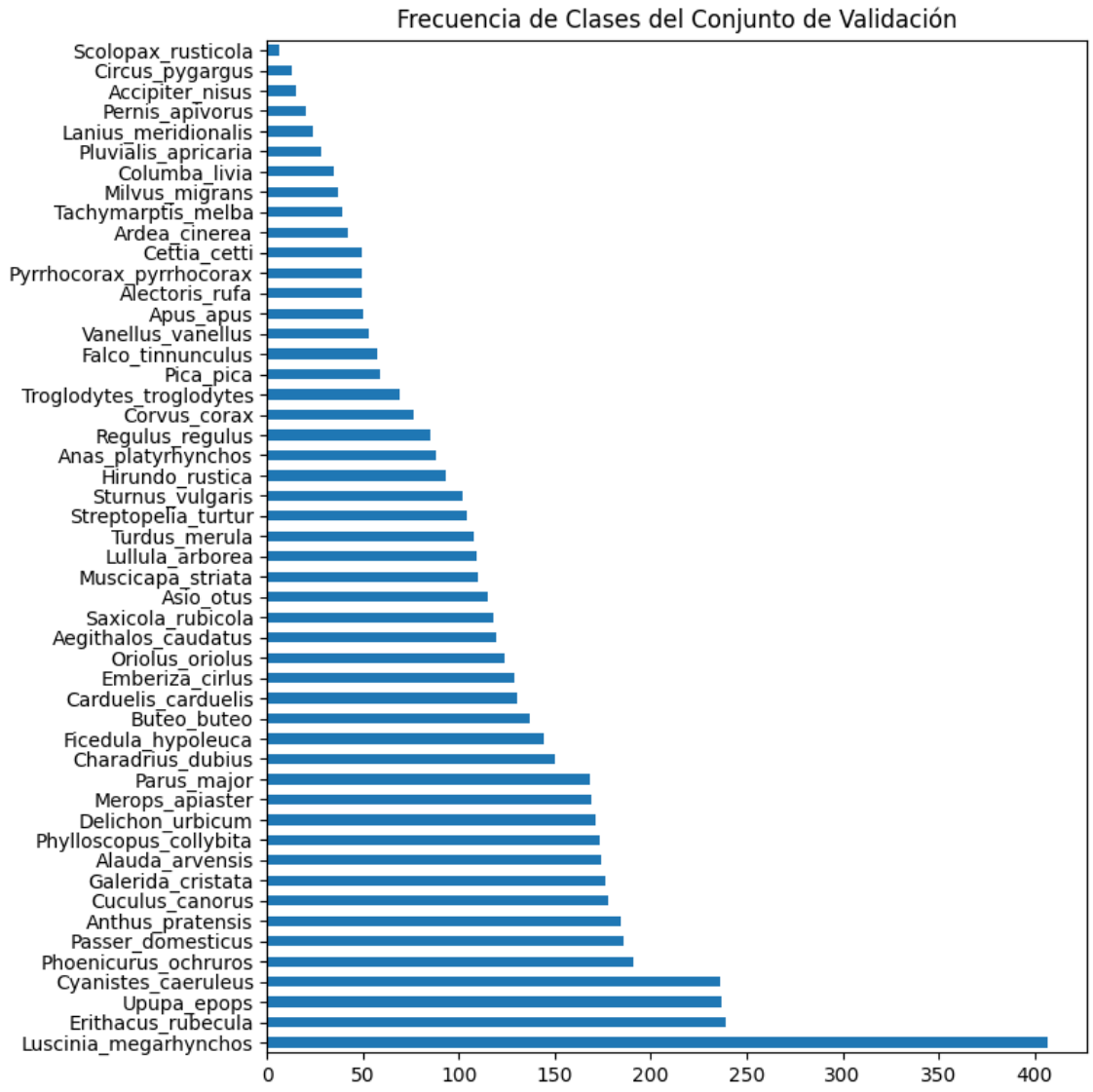

Figura 5.11 Histograma del conjunto de validación con división y sin aumento de datos

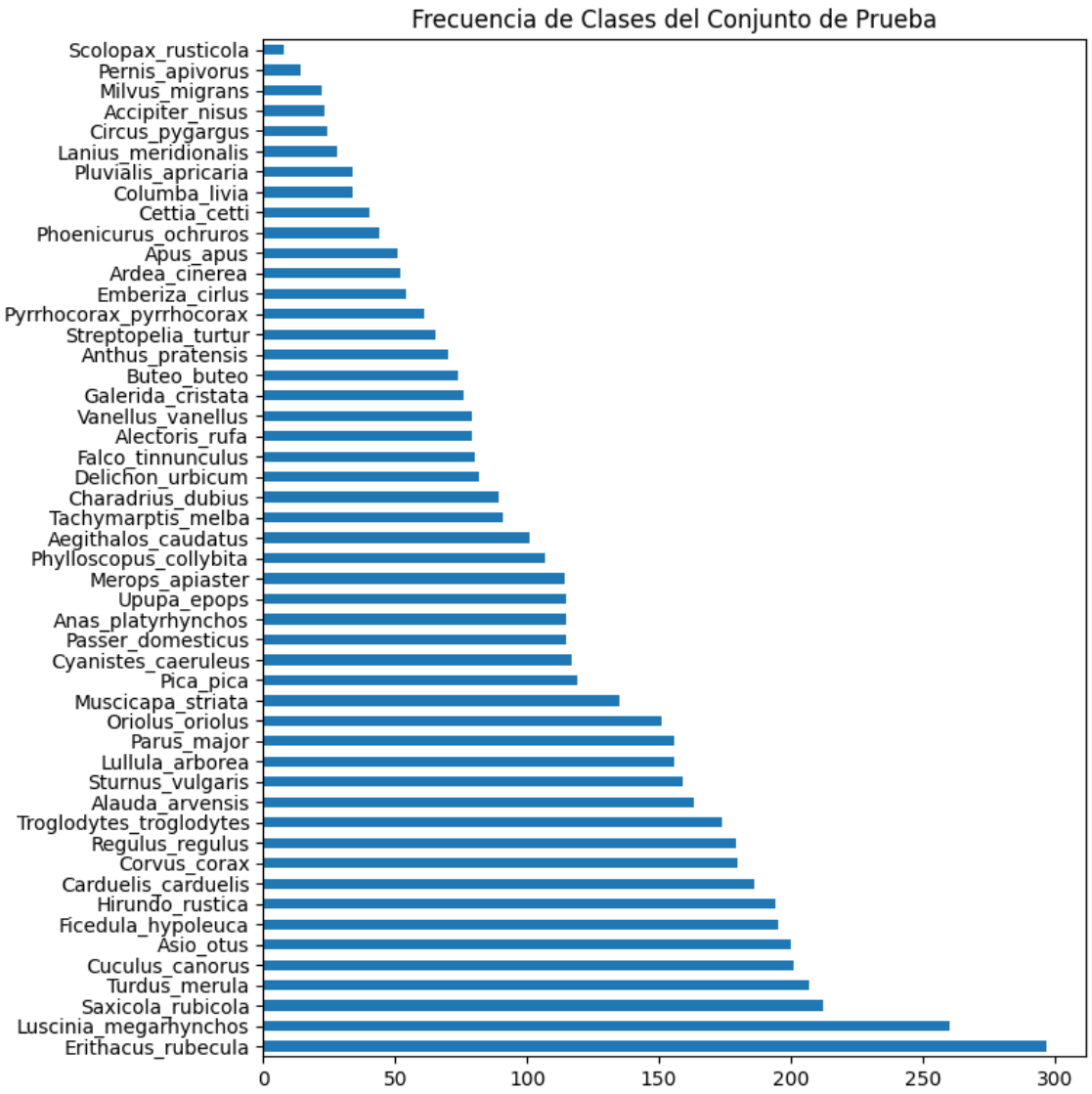

Figura 5.12 Histograma del conjunto de prueba con división y sin aumento de datos

Se puede observar que las distribuciones están desbalanceadas debido a que algunas clases contienen archivos de audio con mayor duración y la división de audios de mayor duración genera un mayor número de fragmentos.

5.6 Aumento de datos

El aumento de datos es una técnica que consiste en aumentar artificialmente el conjunto de entrenamiento creando copias modificadas de un conjunto de datos utilizando los datos existentes [28].

Nosotros utilizaremos dos técnicas de aumento de datos para aumentar el conjunto de entrenamiento. La primera técnica consiste en desplazar aleatoriamente el audio hacia la derecha o izquierda, cambiando el espectrograma del audio. La técnica es llamada "*shifting*". La segunda técnica consiste en modificar la frecuencia del audio para alterar

el tono del audio que percibimos del audio original [28]. La Figura 5.13 muestra los cambios producidos en el espectrograma de un audio de gavilán común al aplicar estas dos técnicas.

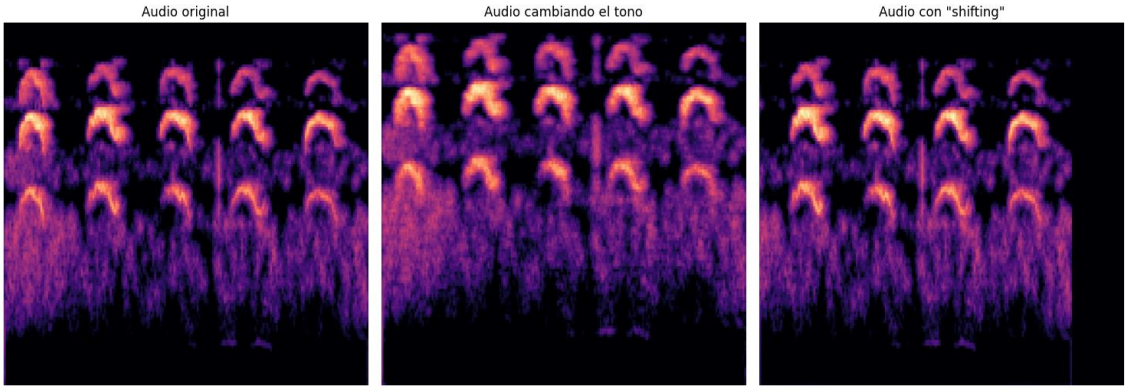

Figura 5.13 Aumento de datos en un audio de gavilán común

También, la división del audio puede ser considerada como un aumento de datos sobre el conjunto de entrenamiento. Más adelante, estudiaremos su relevancia para mejorar la clasificación.

Finalmente, la distribución del conjunto de entrenamiento con la división de los audios y con aumento de datos se muestra Figura 5.14. El conjunto de validación y prueba no son modificados porque no tiene sentido alterar las imágenes que se utilizan para la validación o prueba. Entonces, el conjunto de entrenamiento tiene 151355 imágenes con la división de audios y el aumento de datos.

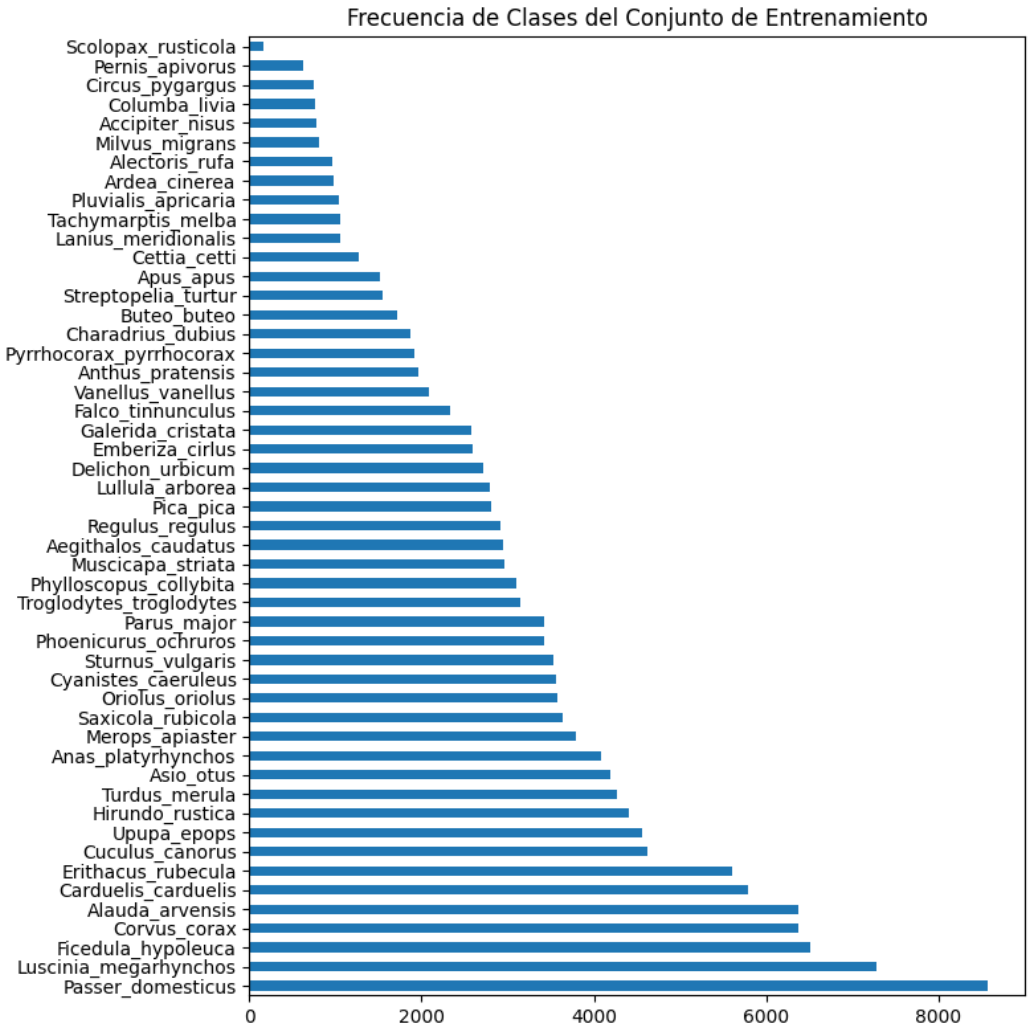

Figura 5.14 Histograma del conjunto de entrenamiento con división y aumento de datos

Capítulo 6

Creación de los modelos

6.1 Introducción

Desarrollaremos diversas redes neuronales convolucionales para clasificar los sonidos de los pájaros y realizar distintas pruebas. Más específicamente en este capítulo, mostraremos las arquitecturas de las redes neuronales que utilizaremos más adelante en las pruebas para elegir un modelo para la aplicación.

Hemos decidido que probaremos 3 arquitecturas de redes neuronales convolucionales y un ensemble de redes convoluciones "*one class vs all*" que consiste en entrenar una serie de redes neuronales con solo una clase para especializarla en detectarla, cada una se especializa en una clase diferente. Posteriormente, detallaremos el motivo del diseño de las arquitecturas y mencionaremos ciertos problemas que pueden tener las distintas redes explicadas. Además, describiremos ciertos procesos que se deben realizar antes para poder entrenar el ensemble y realizar sus predicciones.

6.2 Red convolucional profunda propia

Hemos decidido diseñar una red convolucional profunda debido a que queremos probar el éxito de una red convolucional entrenada por nosotros. Además, esta arquitectura de red será utilizada por el ensemble para cada red que se especializará en un pájaro.

La red neuronal consiste en 21 capas. Tiene una primera capa de entrada y una capa de rescalado que se encarga de normalizar las imágenes, posteriormente empiezan las capas de convolución y *maxpooling*. La primera capa de convolución consiste en 8 filtros, la segunda capa tiene 16 filtros, la tercera y la cuarta capa de convolución tiene 32, la quinta capa de convolución tiene 64 filtros y la última capa convolucional tiene 128 filtros. Todos los filtros son 3x3.

Luego, colocamos varias capas de *dropout* para intentar evitar el sobreajuste y, por último, dos capas densas de 128 neuronas y 256 neuronas para finalmente tener una capa densa con el mismo número de neuronas que clases. Todas las capas tienen *ReLU* como función de activación, salvo la capa de salida que utiliza la función *softmax*.

En resumen, la red se puede dividir en una parte que se encarga de extraer características y otra parte que se encarga de aprender la clasificación de los pájaros. Las capas de la 3 a la 13 se encargan de la extracción de características y el resto del aprendizaje de la clasificación. La Figura 6.3 muestra la estructura de nuestra red convolucional.

Diseñamos esta estructura con tantas capas porque creemos que utilizar muchas capas para extraer características puede ayudar en conjuntos de datos con tantas clases como es nuestro caso.

6.3 Redes convolucionales con *transfer learning*

Añadiremos partes de modelos ya entrenados por muchos más datos para intentar mejorar nuestros modelos. Los modelos seleccionados serán *VGG19* y *Xception*. Realizaremos *fine-tuning* para adaptar mejor nuestras redes convolucionales al problema a resolver.

6.3.1 VGG19

El modelo preentrenado *VGG19* es una red convolucional con 19 capas creado por Karen Simonyan y Andrew Zisserman. La arquitectura de la red convolucional se muestra en la Figura 6.1 y se puede observar que se trata de una serie de bloques de convoluciones [29].

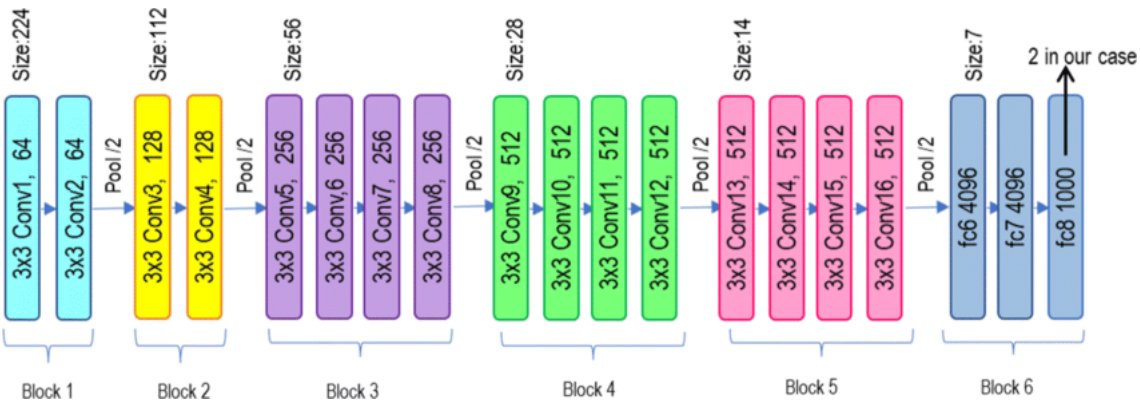

Figura 6.1 Arquitectura de *VGG19* [30]

Este modelo tiene otro modelo muy similar llamado *VGG16* que está compuesta por 16 capas. Estos dos modelos fueron diseñados para ser entrenadas por el conjunto de datos *ImageNet* [31].

Decidimos utilizar *VGG19* para realizar *transfer learning* porque queremos comprobar si utilizar una red con menos capas (22 capas) que el siguiente modelo pre-entrenado que mencionaremos obtiene un buen resultado.

La extracción de características será realizada por el modelo pre-entrenado y el resto es igual que el modelo anterior. Sin embargo, antes aplicaremos el preprocesamiento que necesita en la segunda capa y tercera capa. El preprocesamiento del modelo consiste en convertir las imágenes RGB a BGR y normalizarlas [32].

La capa 7 tiene el modelo pre-entrenado y hemos decidido descongelar las últimas 4 capas del modelo pre-entrenado para adaptar mejor el modelo a nuestro problema. Por último, las capas de la 8 a la 14 son iguales a las capas de la 14 a la 20 de la Sección 6.2. La Figura 6.4 presenta de forma gráfica la arquitectura de nuestra red convolucional con *VGG19*.

6.3.2 Xception

Esta vez, utilizaremos el modelo pre-entrenado *Xception* para extraer características. *Xception* es una arquitectura de red convolucional que se inspira en los módulos *Inception* de la red neuronal *Inception V3* [33]. Fue creada por François Chollet en 2016 [34]. Esta arquitectura de red modifica estos módulos *Inception* para utilizar convoluciones separables que consiste en dividir el proceso de convolución en dos fases:

1. La primera fase consiste en realizar una convolución espacial lo que significa que se aplica una convolución de dos dimensiones (convoluciones 2D).

2. La segunda fase utiliza una convolución de punto, es decir, realiza una convolución de una dimensión.

Las convoluciones separables permiten reducir el número de hiperparámetros entrenables y la carga computacional. Esto provoca una mejora en el modelo y reduce el sobreajuste. Además, este modelo mejora ligeramente el resultado en el conjunto de datos *ImageNet* [31] frente a su inspiración *Inception* V3. La Figura 6.2 muestra la arquitectura de *Xception*.

Hemos elegido este modelo para realizar *transfer learning* porque este modelo tiene muchas más capas que el modelo *VGG19* y queremos estudiar cómo afecta el número de capas en la extracción de características. Nosotros utilizaremos este modelo pre-entrenado igual que en la Sección 6.3.1, es decir, será utilizado para extraer características. Posteriormente, utilizaremos capas totalmente conectadas para aprender la clasificación. Toda la arquitectura de nuestra red convolucional con *Xception* se corresponde con la Figura 6.5.

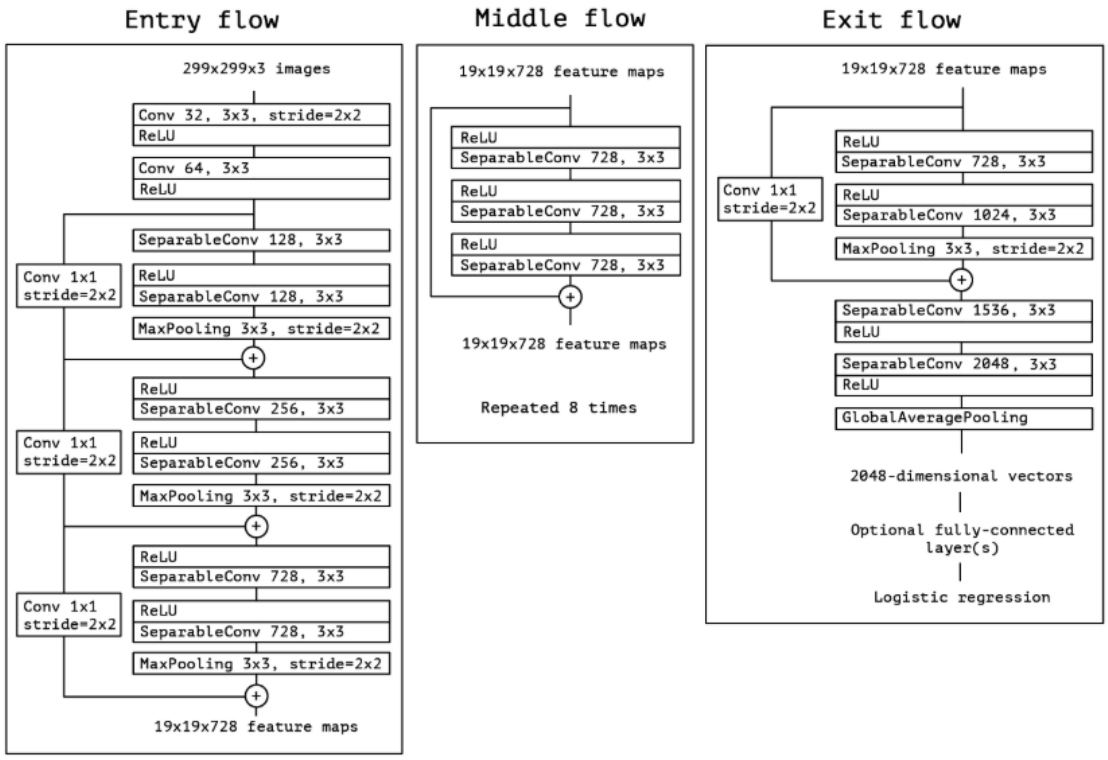

Figura 6.2 Arquitectura de *Xception* [34]

Antes de empezar con la extracción de características, debemos aplicar el preprocesamiento para garantizar el correcto funcionamiento del modelo pre-entrenado. Las capas 2 y 3 se encargan de normalizar la imagen entre -1 y 1, aplicando el preprocesamiento para *Xception*.

La capa 4 es el modelo pre-entrenado *Xception* y hemos aplicado *fine-tuning* para adaptar mejor el modelo a nuestro problema. Hemos optado por descongelar las últimas 22 capas de 132 capas porque el modelo *Xception* no está entrenado con espectrogramas y nos facilita el aprendizaje de los espectrogramas de forma más sencilla al ser una cantidad grande de capas descongeladas, sin perjudicar en gran medida la extracción de características que ya tenía el modelo pre-entrenado.

Por último, utilizamos la capa 6 para aplanar el output del modelo *Xception* y las capas 7 y 9 se encargan de aprender la clasificación al ser capas totalmente conectadas. Entre estas capas, las capas de *Dropout* reducen el riesgo de sobreajuste. La última capa calcula la clase predicha aplicando la función *softmax*.

6.4 Ensemble de redes convolucionales “one class vs all”

Realizaremos un ensemble de redes convolucionales, pero cada red convolucional estará especializa en solo aprender un solo tipo de pájaro. Existirán tantas redes convolucionales como tipos de pájaros tengamos que clasificar. Para hacer esto,

entrenaremos cada red con los espectrogramas del pájaro que tiene que especializarse en clasificar como una clase y el resto de las imágenes como otra clase.

Para ello, realizamos un paso más en el preprocesamiento que consistirá en preparar las bases de datos para asegurar el aprendizaje de cada red neuronal especializada. Creamos un conjunto de datos con 2 clases, la clase principal que será el ave que tiene que especializarse el pájaro y otra clase llamada “Otros”. La clase principal contendrá las imágenes del ave principal o a especializar y “Otros” contendrá el resto.

Sin embargo, esto provocará un desbalanceo porque la clase principal tendrá menos imágenes que la clase “Otros” al ser la mayoría del conjunto. Para solucionar este problema, la clase “Otros” tendrá el mismo número de imágenes que la clase principal, asegurándonos que la clase “Otros” sea estratificada para que todas las demás clases estén presentes en la clase “Otros”. Las imágenes de las clases que formarán la clase “Otros” serán seleccionados aleatoriamente respetando las distribuciones para cumplir la condición de no generar clases desbalanceadas, es decir, seguimos un muestreo aleatorio.

Cuando el ensemble recibe una imagen de un espectrograma, cada red convolucional especializada en un ave realiza inferencia con esta imagen. Las redes convolucionales producirán dos probabilidades cada una que corresponden a las probabilidades de pertenecer a una de las dos clases. El ensemble reúne todas las probabilidades y realiza el siguiente cálculo:

$$\hat{y} = \underset{y \in Aves}{\arg max}\, \mathbb{P}(y; M_i)$$

Entendiendo que M es el modelo utilizado para clasificar esa clase.

En resumen, el ensemble compara todas las probabilidades obtenidas de la clase principal (la clase contraria a “Otros”) y el ave principal de la red convolucional que obtiene la mejor probabilidad es seleccionado como la clase predicha.

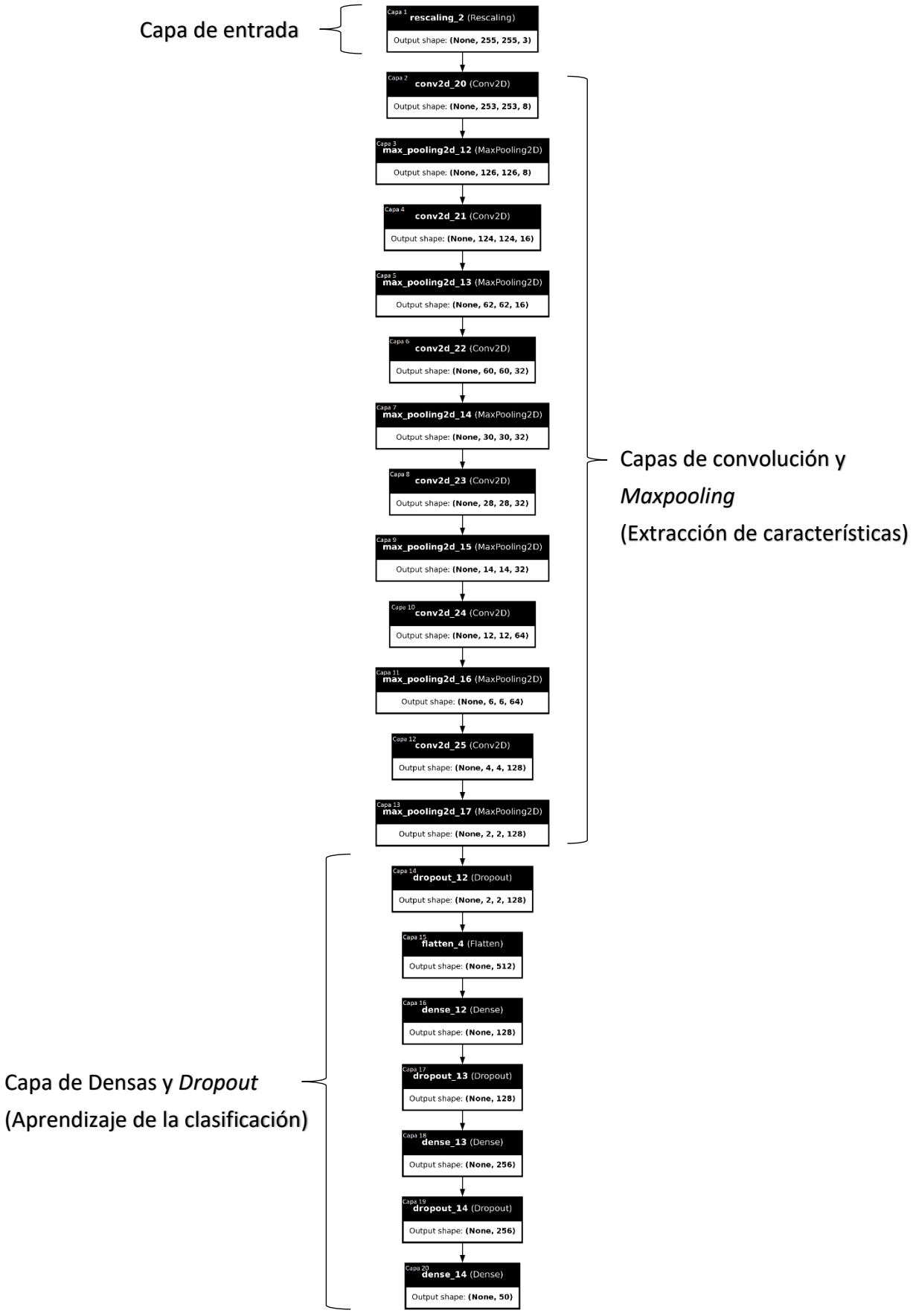

Figura 6.3 Arquitectura de Red Convolucional propia

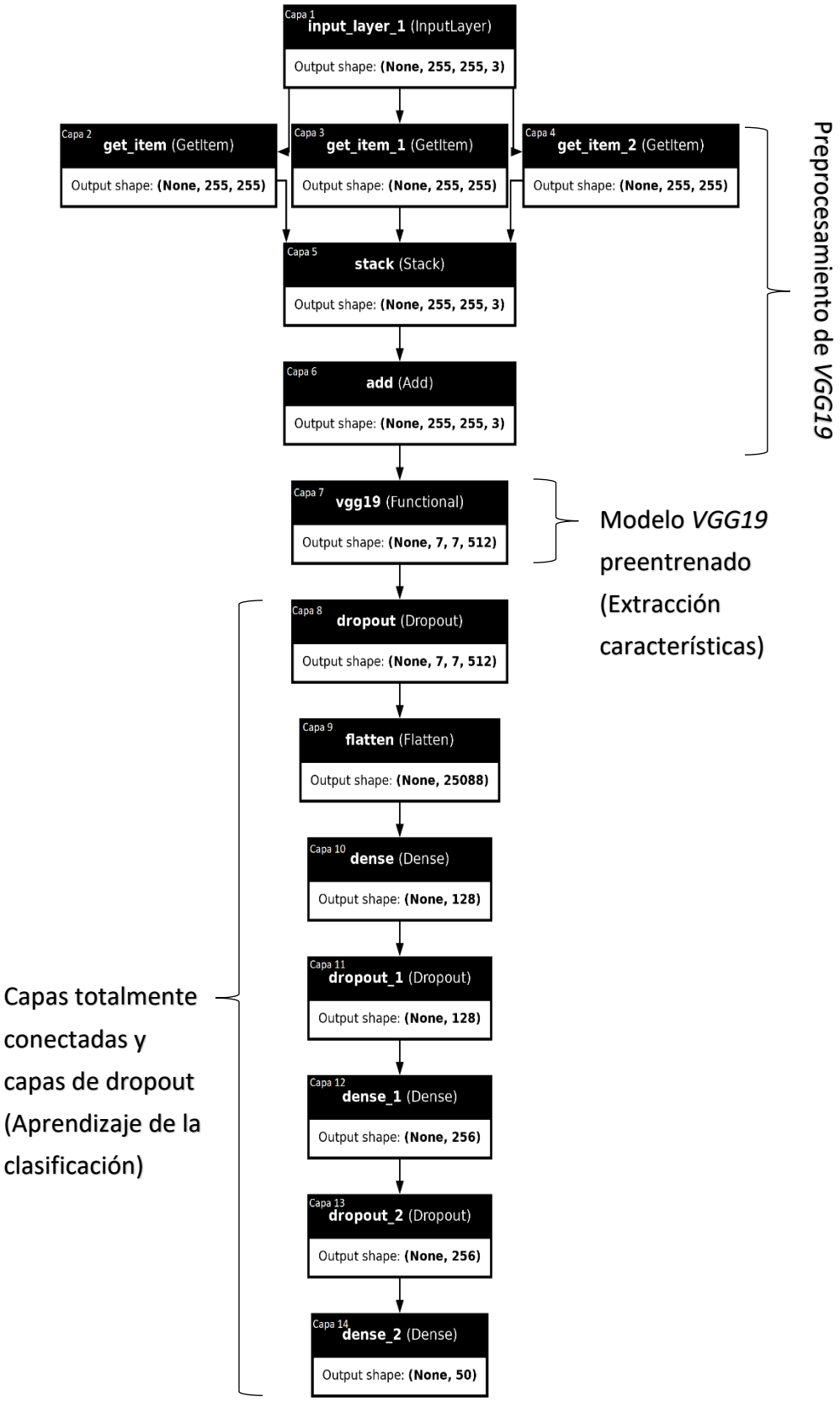

Figura 6.4 Arquitectura de nuestra red con *VGG19*

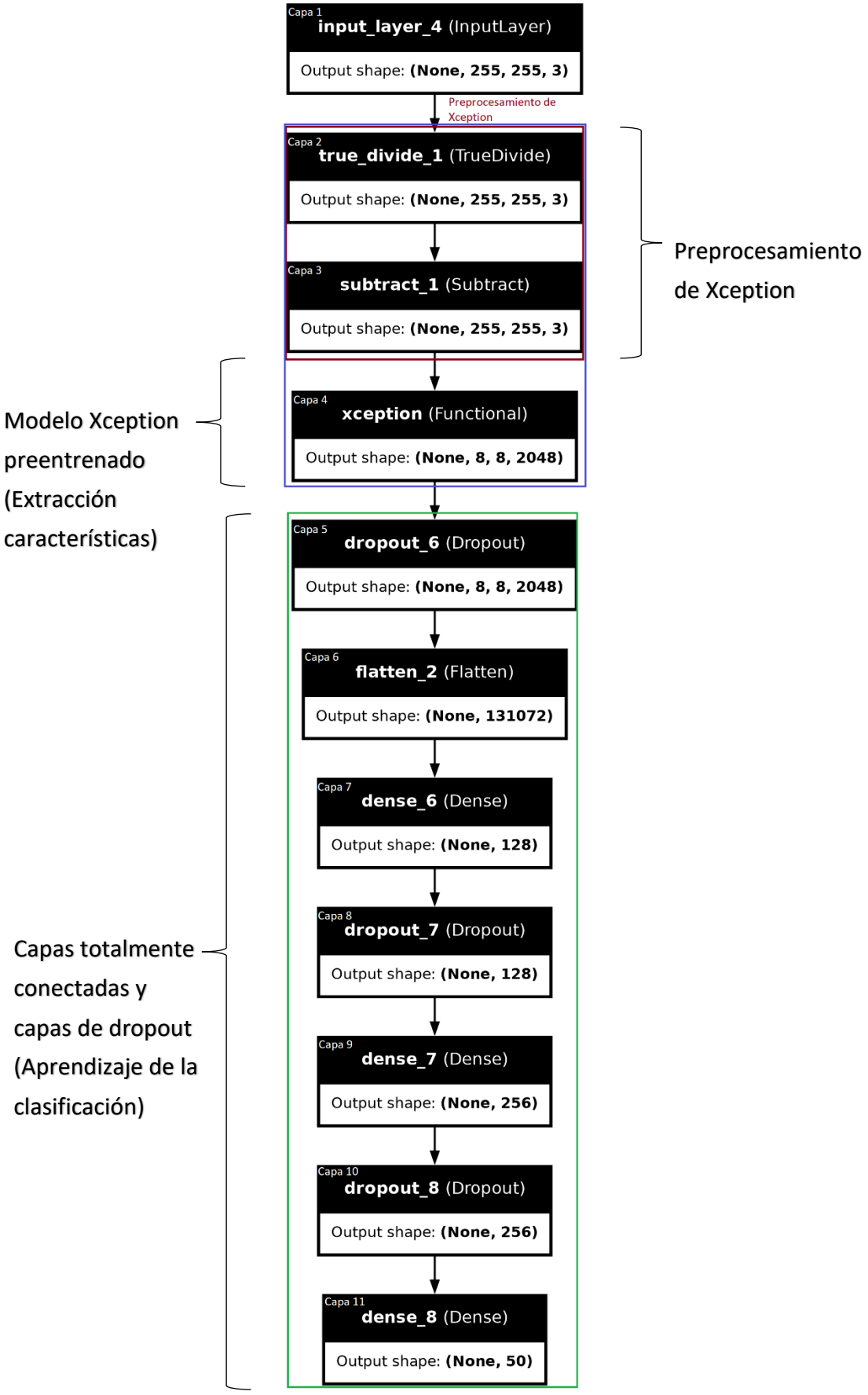

Figura 6.5 Arquitectura de nuestra red con *Xception*

Capítulo 7

Pruebas de los modelos

7.1 Introducción

Evaluaremos y validaremos los modelos construidos en el capítulo anterior. Para ello, realizaremos diversas pruebas a todos los modelos y aplicaremos diversos procesos de preprocesamiento.

7.2 Pruebas con un conjunto de datos sin división de audios y sin aumento de datos

Primero, probaremos a entrenar los modelos con los espectrogramas en *escala de mel* sin ningún aumento de datos ni división de audios, por tanto, utilizaremos los conjuntos de datos obtenidos en la Sección 3.4 sin ninguna modificación. El ensemble solo será evaluado con el conjunto de prueba porque no tiene sentido evaluarlo con los otros conjuntos de datos al ser entrenado y validado por otros conjuntos de datos como hemos explicado en la Sección 6.4.

Mostraremos los resultados de los modelos tras 10 *epochs* de entrenamiento en la Tabla 1, Tabla 2 y Tabla 3. Es importante mencionar que las métricas de precisión, sensibilidad y *F1-Score* son *macro* que consiste en aplicar los siguientes cálculos:

$$Precisión_{Macro} = \frac{\sum_{i=0}^{n} Precisión_i}{n}$$

$$Sensibilidad_{Macro} = \frac{\sum_{i=0}^{n} Sensibilidad_i}{n}$$

$$F1Score_{Macro} = \frac{\sum_{i=0}^{n} F1Score_i}{n}$$

Siendo n el número de clases del conjunto. Aplicamos *macro* porque tenemos múltiples clases y necesitamos un solo valor para mostrar el resultado de nuestros modelos de forma global.

Tabla 1 Resultados con el conjunto de entrenamiento sin división de datos ni aumento de datos

Entrenamiento	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.432	0.398	0.410	0.354
VGG19	0.024	0.02	0	0
Xception	0.379	0.340	0.334	0.309

Tabla 2 Resultados con el conjunto de validación sin división de datos ni aumento de datos

Validación	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.360	0.328	0.335	0.288
VGG19	0.024	0.02	0	0
Xception	0.303	0.273	0.265	0.239

Tabla 3 Resultados con el conjunto de prueba sin división de datos ni aumento de datos

Prueba (*test*)	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.394	0.356	0.394	0.324
VGG19	0.0236	0.02	0	0
Xception	0.270	0.246	0.221	0.210
Ensemble	0.173	0.166	0.094	0.102

Se puede observar que los resultados son algo sorprendentes porque las redes convolucionales que utilizan modelos pre-entrenados (*VGG19* y *Xception*) dan peor resultado que la red convolucional que hicimos desde cero. El ensemble "*one class vs all*" tampoco da un buen resultado, pero quizás con más imágenes el resultado sea mejor. También, el modelo VGG19 muestra un resultado pésimo frente al resto de modelos, pero lo mantendremos en las pruebas posteriores porque con la división de datos de la Sección 5.5 puede mejorar el resultado.

7.3 Pruebas con un conjunto de datos con división de audios y sin aumento de datos

Ahora, probaremos a aplicar la división de audios de la Sección 5.5, pero no aplicaremos el aumento de datos descrito en la Sección 5.6. Los resultados se muestran en la Tabla 4, Tabla 5 y Tabla 6.

Tabla 4 Resultados con el conjunto de entrenamiento con división de datos y sin aumento de datos

Entrenamiento	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.547	0.461	0.561	0.458
VGG19	0.056	0.02	0.001	0.002
Xception	0.869	0.826	0.836	0.825

Tabla 5 Resultados con el conjunto de validación con división de datos y sin aumento de datos

Validación	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.458	0.408	0.433	0.381
VGG19	0.033	0.02	0	0.001
Xception	0.637	0.599	0.614	0.588

Tabla 6 Resultados con el conjunto de prueba con división de datos y sin aumento de datos

Prueba (*test*)	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.492	0.412	0.439	0.389
VGG19	0.021	0.02	0	0
Xception	0.650	0.595	0.606	0.588
Ensemble	0.425	0.384	0.425	0.353

Se puede observar que los resultados han mejorado bastante sobre todo con el modelo *Xception* que ha mejorado de un 27% de tasa de acierto a un 65% de tasa de acierto en el conjunto de prueba. También, mejora en el resto de las métricas y conjuntos de datos restantes.

La red convolucional entrenada mejora bastante, pero no tanto como el modelo *Xception*. En general, la división de audios mejora el entrenamiento de los modelos, por tal razón, aplicaremos esta división de audios en la siguiente sección junto al aumento de datos de la Sección 5.6. No obstante, el modelo *VGG19* sigue sin mejorar, por tanto, no seguiremos evaluando este modelo. El ensemble sigue estando por detrás, por tanto, las próximas pruebas no contendrán este ensemble.

7.4 Pruebas con un conjunto de datos con división de audios y con aumento de datos

Para la última evaluación, aplicaremos división de datos y aumento de datos sobre el entrenamiento de los modelos. Los resultados son mostrados en la Tabla 7, Tabla 8 y Tabla 9.

Tabla 7 Resultados con el conjunto de entrenamiento con división de datos y aumento de datos

Entrenamiento	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.489	0.400	0.470	0.392
Xception	0.916	0.894	0.902	0.895

Tabla 8 Resultados con el conjunto de validación con división de datos y aumento de datos

Validación	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.434	0.370	0.403	0.344
Xception	0.682	0.650	0.669	0.644

Tabla 9 Resultados con el conjunto de prueba con división de datos y aumento de datos

Prueba (*test*)	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
Red propia	0.465	0.388	0.436	0.370
Xception	0.691	0.651	0.665	0.646

Los resultados muestran que el aumento de datos ha sido beneficioso para el entrenamiento del modelo *Xception*, pero nuestra propia red convolucional ha empeorado.

Tras las pruebas, hemos llegado a una posible hipótesis sobre el desempeño de las redes con los modelos pre-entrenados. Puede que los modelos no extraigan las características de forma correcta debido a que las imágenes proporcionadas al modelo pre-entrenado son muy diferentes a un espectrograma y su extracción de características no ayuda a reconocer patrones en estas.

Para comprobar esta hipótesis, entrenaremos los modelos que utilizan *Xception* y *VGG19* con todas capas descongeladas, es decir, reajustaremos los pesos que ya tienen.

Realizaremos 10 *epochs* para cada nuevo modelo. Los resultados de los nuevos modelos se encuentran en la Tabla 10, Tabla 11 y Tabla 12.

Tabla 10 Resultados de los modelos pre-entrenados con todas las capas descongeladas con el conjunto de entrenamiento

Entrenamiento	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
VGG19	0.056	0.02	0.001	0.002
Xception	0.853	0.828	0.843	0.830

Tabla 11 Resultados de los modelos pre-entrenados con todas las capas descongeladas con el conjunto de validación

Validación	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
VGG19	0.033	0.02	0	0.001
Xception	0.725	0.707	0.715	0.696

Tabla 12 Resultados de los modelos pre-entrenados con todas las capas descongeladas con el conjunto de prueba

Prueba(*test*)	**Tasa de acierto**	**Precisión**	**Sensibilidad**	***F1-Score***
VGG19	0.021	0.002	0	0
Xception	0.735	0.696	0.705	0.686

Finalmente, el modelo *Xception* ha mejorado y el modelo *VGG19* no ha mostrado ninguna mejora. No podemos asegurar con certeza que descongelar todas las capas del modelo pre-entrenado mejore el modelo porque no ha mejorado mucho y en el entrenamiento, ha empeorado. Entonces, elegiremos el modelo *Xception* con todas las capas descongeladas para construir nuestra aplicación y aplicaremos la división de audio como el aumento de datos descritos en las Secciones 5.5 y 5.6 porque ha sido él que mejor resultados a dado en el conjunto de validación y prueba.

Capítulo 8

Creación de la aplicación

8.1 Introducción

Desarrollaremos una aplicación que permita utilizar nuestro modelo de forma cómoda y sencilla para clasificar un audio de pájaro mediante una interfaz. Relataremos las tecnologías que hemos utilizado y el motivo del diseño de la interfaz de la aplicación. Utilizará el mejor modelo obtenido en el Capítulo 7 y a continuación, se realizará la fase de despliegue mencionado en la metodología CRISP-DM.

8.2 Flask

Flask es un *framework* web escrito en Python que permite crear aplicaciones web de forma sencilla. Este *framework* contiene muchas herramientas que permiten desarrollar tanto el *Back-End* como el *Front-End* de la aplicación web. El *Back-End* será creado utilizando Python, por tanto, un archivo con formato .py contendrá toda la lógica del servidor para atender a los clientes. No obstante, el *Front-End* será programado y diseñado mediante archivos HTML con algo de *Javascript* y CSS [35].

También, volvemos a utilizar la librería *Tensorflow* para realizar la inferencia al archivo de audio subido. El modelo que nosotros hemos elegido se encuentra como un archivo con extensión .h5 que hemos guardado con la misma librería.

Por último, utilizamos *Swiper* que es una biblioteca de *Javascript* que permite crear carruseles y deslizadores táctiles para mostrar imágenes u otros elementos de forma sencilla y estética [36].

8.3 Aplicación web

Hemos decidido que la aplicación será desarrollada como una aplicación web porque así, cualquier usuario con un navegador web puede acceder fácilmente a nuestra aplicación. La interfaz de aplicación web tiene un estilo minimalista con un fondo gris. Los usuarios pueden subir el audio, pulsando el botón subir archivos o arrastrando el audio que quieren subir.

El audio debe subirse con un formato .wav o .mp3 porque estos dos formatos son los más comunes utilizados por los usuarios. Por ejemplo, la página web *Xeno-canto* tiene sus ficheros de audio con formato .mp3. Sin embargo, no significa que no sea compatible con otros formatos de audio, estos dos formatos han sido probados extensamente y no podemos asegurar que otros formatos vayan a funcionar correctamente. En la parte superior, se dan los reconocimientos necesarios sobre los recursos como fotos y audios utilizados para crear la aplicación. La Figura 8.1 muestra la página inicial de la aplicación.

Al pulsar el botón de clasificar y tener un fichero de audio seleccionado, la aplicación comenzará el preprocesamiento para realizar la inferencia para clasificar el audio. Mientras se realiza este proceso, se muestra una pantalla de carga para mostrar al usuario que el proceso sigue. Esta página de carga se muestra en la Figura 8.2.

Cuando termina la inferencia, se procesa el resultado y los posibles pájaros son mostrados en una lista. La lista muestra primero los pájaros que tienen mayor probabilidad con su imagen y porcentaje, y pulsando o deslizando la lista se mostrarán el resto de los pájaros como se puede ver en la Figura 8.3 y Figura 8.4.

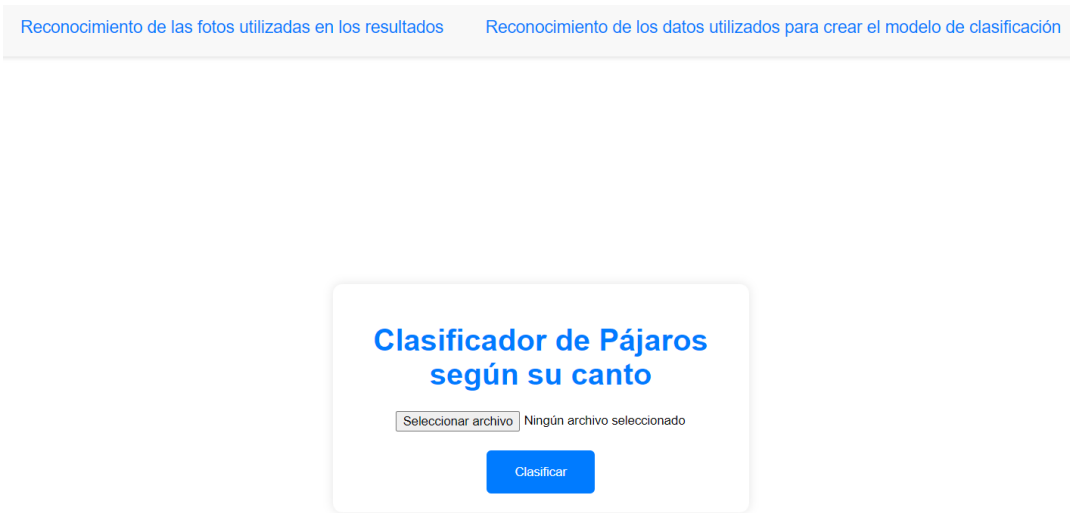

Figura 8.1 Página inicial de la aplicación

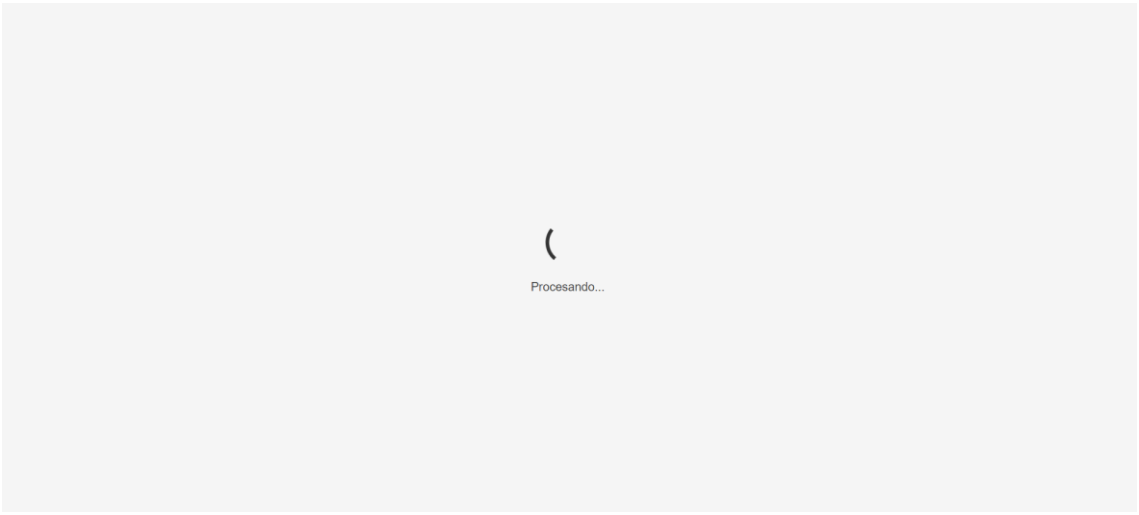

Figura 8.2 Página de carga mientras el modelo clasifica

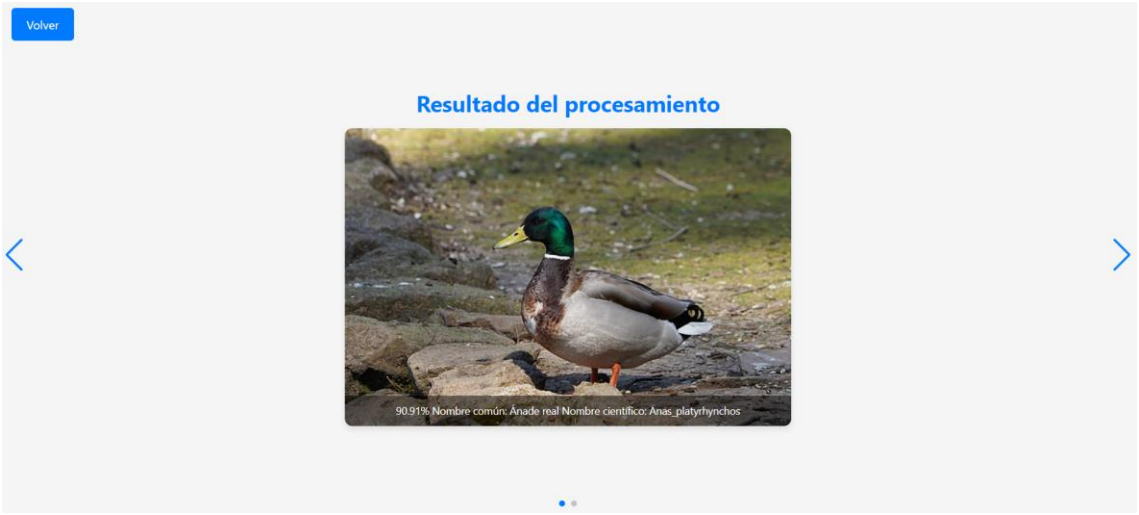

Figura 8.3 Página del resultado

Figura 8.4 Página del segundo resultado

Capítulo 9

Conclusión

9.1 Conclusiones y trabajo a futuro

Comenzamos el trabajo de fin de grado con el objetivo de estudiar como clasificar el canto de pájaros de forma automática para clasificar pájaros sin necesidad de verlos y ayudar a su conservación. Gracias a las tecnologías modernas de aprendizaje automático, sobre todo *deep learning*, hemos conseguido crear modelos capaces de reconocer esos pájaros y clasificarlos.

Nos hemos limitado a realizarlo con 50 aves de Albacete para probar si era posible resolver el problema. No obstante, hemos desarrollado un flujo de trabajo para generar de forma automática conjuntos de datos que permitan entrenar modelos para clasificar cualquier ave que exista en *Xeno-canto*. Por tanto, podríamos mejorar nuestro modelo para permitir que pueda clasificar más aves sin necesidad de buscar conjuntos de datos en internet.

También, hemos descubierto formas de preprocesar audios para mejorar el resultado de los modelos, pero aún se puede mejorar reduciendo mejor el ruido, utilizando mejores algoritmos de división de audios, utilizando modelos pre-entrenados enfocados en obtener el espectrograma en *escala de mel* y optimizando el número de capas descongeladas.

Los resultados obtenidos son buenos al conseguir una tasa de acierto elevada con tantas clases. No obstante, los resultados pueden mejorar aún más si utilizásemos más datos como hemos demostrado con la arquitectura de *Xception*, reduciendo el riesgo de sobreajuste.

Hemos creado una aplicación web con el mejor modelo obtenido para facilitar el uso de este de una forma visual y sencilla. Sin embargo, nuestra aplicación aún tiene margen

de mejora porque puede ser más bonita visualmente, añadir más funcionalidades o realizar despliegue continuo para subir la aplicación a la nube.

Nos gustaría mencionar que creemos que esta aplicación puede ayudar mucho a concienciar a los ciudadanos de Albacete sobre la biodiversidad de aves que existen. Esto provocará que más personas sean más respetuosas con estas aves y su ecosistema, haciendo que su conservación sea más sencilla.

Bibliografía

[1] R. Laje, «La Física del Canto de Aves,» Agosto 2005. https://bibliotecadigital.exactas.uba.ar/download/tesis/tesis_n3860_Laje.pdf.

[2] J. Smolicki, «TUTORIAL del MANUAL DE ECOLOGÍA ACÚSTICA,» https://www.sfu.ca/sonic-studio-webdav/cmns/Handbook%20Tutorial/FieldRecording.html.

[3] CORNELL LAB OF ORNITHOLOGY, «BirdCLEF 2024,» https://www.kaggle.com/competitions/birdclef-2024.

[4] Catherine Huang, fb, Will Cukierski, «MLSP 2013 Bird Classification Challenge,» https://www.kaggle.com/competitions/mlsp-2013-birds/data.

[5] Xeno-canto Foundation for Nature Sounds, «Xeno-canto - Bird sounds from around the world,» https://www.gbif.org/dataset/b1047888-ae52-4179-9dd5-5448ea342a24.

[6] Sociedad Albacetense de Ornitología (S.A.O.), «Anuario Ornitológico de Albacete online (AOA),» https://anuario.albacete.org/especie/lista_ab/.

[7] C. M. P. Pertuz, Aprendizaje automatico y profundo en python, Ra-Ma.

[8] J. L. Gonzalez, «Tipos de aprendizaje automático,» https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2.

[9] DG, «Datos Estructurados y no Estructurados: 5 Diferencias Principales,» https://dgcloud.com.br/es/datos-estructurados-y-no-estructurados-5-diferencias-principales/.

[10] T. G. E, «Introducción al algoritmo Random Forest,» https://tutegomeze.medium.com/introducci%C3%B3n-al-algoritmo-random-forest-54577e93458b.

[11] IBM, «¿Qué son las redes neuronales?,» https://www.ibm.com/es-es/topics/neural-networks.

[12] I. G.R, «Catálogo de componentes de redes neuronales (II): funciones de activación,» http://bluechip.ignaciogavilan.com/2020/05/catalogo-de-componentes-de-redes_20.html.

[13] IBM, «¿Qué son las redes neuronales convolucionales?,» https://www.ibm.com/es-es/topics/convolutional-neural-networks.

[14] J. cuartas, «El concepto de la convolución en gráficos, para comprender las Convolutional Neural Networks (CNN) o redes convolucionadas,» https://josecuartas.medium.com/el-concepto-de-la-convoluci%C3%B3n-en-gr%C3%A1ficos-para-comprender-las-convolutional-neural-networks-cnn-519d2eee009c.

[15] lisdatasolutions, «Deep Learning: clasificando imágenes con redes neuronales,» https://www.lisdatasolutions.com/es/blog/deep-learning-clasificando-imagenes-con-redes-neuronales/.

[16] Google, «Transferencia de aprendizaje y ajuste,» https://www.tensorflow.org/tutorials/images/transfer_learning?hl=es-419.

[17] Google,«tf.keras.applications,» https://www.tensorflow.org/api_docs/python/tf/keras/applications.

[18] Juan Ignacio Barrios Arce, «La matriz de confusión y sus métricas,» https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/.

[19] J. F. V. Rueda, «CRISP-DM: una metodología para minería de datos en salud,» https://healthdataminer.com/data-mining/crisp-dm-una-metodologia-para-mineria-de-datos-en-salud/.

[20] Xenocanto, «Xenocanto,» https://xeno-canto.org/.

[21] Hoygrabo, «¿Qué es frecuencia de muestreo y profundidad de bits?,» https://hoygrabo.com/que-es-frecuencia-de-muestreo-y-profundidad-de-bits/.

[22] Google, «Transferir el aprendizaje con YAMNet para la clasificación de sonido ambiental,» https://www.tensorflow.org/tutorials/audio/transfer_learning_audio?hl=es-419.

[23] T. Sainburg, «noisereduce,» https://pypi.org/project/noisereduce/.

[24] Unir, «¿Qué es el espectrograma y cuáles son sus usos en el análisis musical?,» https://ecuador.unir.net/actualidad-unir/espectrograma/#:~:text=El%20espectrograma%20es%20una%20representaci%C3%B3n,de%20un%20periodo%20de%20tiempo..

[25] Librosa, «Librosa,» https://librosa.org/doc/main/index.html.

[26] «Matplotlib,» https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html.

[27] K. Doshi, «Audio Deep Learning Made Simple - Why Mel Spectrograms perform better,» https://ketanhdoshi.github.io/Audio-Mel/.

[28] A. A. Awan, «Guía completa para el aumento de datos,» https://www.datacamp.com/es/tutorial/complete-guide-data-augmentation.

[29] K. S. y. A. Zisserman, «Redes convolucionales muy profundas para el reconocimiento de imágenes a gran escala,» https://arxiv.org/pdf/1409.1556.

[30] A. K. y. S. M. K. Quadri, «Generalization of convolutional network to domain adaptation network for classification of disaster images on twitter,»

https://www.researchgate.net/publication/359771670_Generalization_of_convolutional_network_to_domain_adaptation_network_for_classification_of_disaster_images_on_twitter.

[31] Stanford Vision Lab, Stanford University, Princeton University , «ImageNet,» Available: https://www.image-net.org/.

[32] Tensorflow,«VGG19,» https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg19/preprocess_input.

[33] V. V. S. I. J. S. Z. W. Christian Szegedy, «Rethinking the Inception Architecture for Computer Vision,» 2 12 2015. https://arxiv.org/abs/1512.00567.

[34] F. Chollet, «Xception: Deep Learning with Depthwise Separable Convolutions,» 7 10 2016. https://arxiv.org/pdf/1610.02357.

[35] J. J. L. Gómez, «Tutorial de Flask en español: Desarrollando una aplicación web en Python,» https://j2logo.com/tutorial-flask-espanol/.

[36] nolimits4web, «Swiper,» https://swiperjs.com/.

[37] autopsia, «¿Cuándo comenzaron los animales a hacer ruido?,» 14 Enero 2022. https://www.autopista.es/planeta2030/cuando-comenzaron-animales-hacer-ruido_249378_102.html.

[38] Gobierno de España Ministerio de tranformación digital, «¿Cómo aprenden las máquinas? Machine Learning y sus diferentes tipos,» https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos.

[39] Xeno-Canto, «Xeno-Canto API,» https://xeno-canto.org/explore/api.

[40] Google, «Kaggle,» [En línea]. https://www.kaggle.com/.

[41] Emerson, «Comprender las FFT y las ventanas,» https://www.ni.com/es/shop/data-acquisition/measurement-fundamentals/analog-fundamentals/understanding-ffts-and-windowing.html.

Anexo I. Recursos utilizados

I.1 Recursos hardware

Los recursos hardware utilizados para la realización de este proyecto son:

- Servidor de *Kaggle* (Entrenamiento y evaluación de los modelos)
  - **GPU** P100 de los servidores de *Kaggle*
  - **RAM** 29GB
- Ordenador personal (Preprocesamiento, generación de conjuntos de datos y despliegue de la aplicación web)
  - **CPU** Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
  - **RAM** 16GB
  - **GPU** Intel Iris Xe

I.2 Recursos software

Los recursos software utilizados para la realización de este proyecto son:

- Tensorflow 2.16.1: Librería utilizada para crear, evaluar y entrenar los modelos.

- Jupyter notebooks: Aplicación web que permite escribir fragmentos de código de Python mezclados con texto normal (Markdown).

- Python 3.11.1: Lenguaje de programación utilizado para programar libretas de Jupyter y realizar la generación de los conjuntos de datos y preprocesamiento.

- Python 3.10.1: Lenguaje de programación utilizado para crear, evaluar y entrenar los modelos.

- Librosa 0.10.1: Librería de Python utilizada para cargar, modificar y separar los audios.

- Noisereduce 3.0.2: Librería utilizada para reducir el ruido de los audios.

- Flask 3.0.3: *Framework* para programar la aplicación web.

- Request 2.31.0: Librería de Python para recibir y enviar peticiones.
- Numpy 1.24.1: Librería de Python para realizar operaciones matemáticas de forma eficiente.

- Pandas 1.5.2: Librería de Python para gestionar cantidades grandes de datos.

- Pydub 0.25.1: Librería de Python utilizada para transformar los archivos de audio de un formato de .mp3 a .wav.

- Matplotlib 3.7.0: Librería de Python utilizada para dibujar las gráficas y espectrogramas.

- Soundfile 0.12.1: Librería de Python utilizada para escribir los audios cuando han sido modificados.

Anexo II. Planificación del proyecto

II.1 Planificación

Como expusimos en el Sección 2.6, hemos seguido la metodología CRISP-DM. A continuación, vamos a ver cómo hemos desarrollado esta metodología durante la realización del trabajo fin de grado.

La Figura I.1 muestra el diagrama de Gantt de nuestro proyecto y a continuación, describiremos la duración de las fases.

- Comprensión del problema: Se ha desarrollado esta fase durante el Capítulo 1 y ha tenido una duración de las tres primeras semanas de marzo.

- Comprensión de datos: Se ha desarrollado esta fase durante el Capítulo 2 y Capítulo 5, durando desde finales de marzo a inicios de abril.

- Preparación de datos: Se ha desarrollado durante el Capítulo 3, Capítulo 5 y Capítulo 4, durando desde inicios de abril a finales de mayo.

- Modelado: Esta fase se ha desarrollado durante el Capítulo 6 y su duración fue de todo el mes de mayo.

- Evaluación: Esta fase se ha desarrollado durante el Capítulo 7 y su duración fue desde finales de mayo a inicio de junio.

- Despliegue: Esta fase se ha desarrollado durante el Capítulo 8 y su duración fue desde finales de mayo a finales de junio.

Por último, expondremos el tiempo invertido para realizar este documento. El documento comenzó a redactarse en la semana 2 y desde entonces, ha sido redactado mientras se realizaban el resto de las fases.

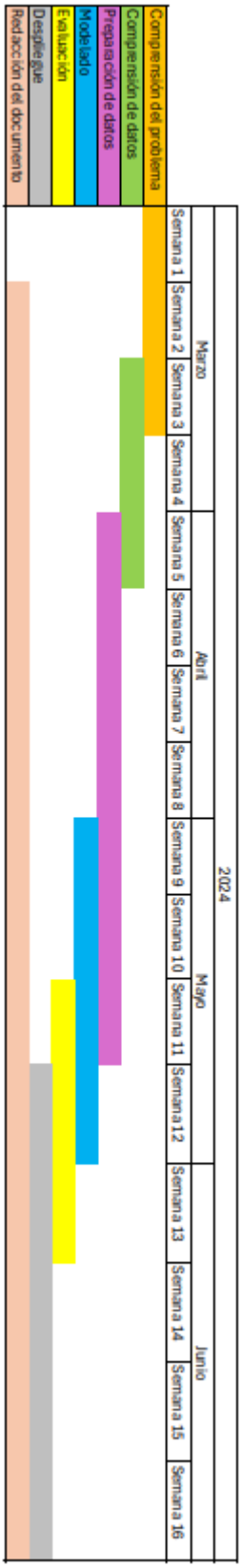

Figura I.1 Diagrama de Gantt del proyecto

Anexo III.

Información sobre el software creado

III.1 Generación de los conjuntos de datos

Todo el software creado se encuentra en un repositorio de *GitHub* cuyo enlace es https://github.com/Chispa4013/TFG-Software.

El fichero "pre-procesamiento" con extensión *.ipynb* es una libreta de *jupyter* que contiene todos los métodos para crear conjuntos de datos. El método principal se llama "crear_conjuntos_datos" que genera tres carpetas llamadas *train*, *test* y *validation*. Los conjuntos de datos se guardan en esas carpetas, contiendo los audios y los espectrogramas generados en diferentes subcarpetas.

Las subcarpetas generadas más importantes son:

- "data_set_birds": Contiene los audios sin preprocesar
- "melspectogram_reduce_noise_set_birds": Contiene los espectrogramas en *escala de mel* sin dividirlos.
- "melspectogram_reduce_noise_set_birds_split": Contiene los espectogramas en escala de mel divididos. Las imágenes generadas tienen el nombre del fichero original y el número del fragmento en su nombre, separados por "_".
- "reduce_noise_set_birds": Contiene los audios con el ruido reducido.

Las subcarpetas tienen varias carpetas con los archivos de cada clase. El nombre de esas carpetas es el nombre científico del ave. El primer parámetro llamado "nombres_cientificos" es la lista con los nombres científicos de las aves. El segundo

parámetro consiste en el número de fichero que se quiere descargar por clase. Los dos últimos parámetros representan las distribuciones de *test* y *validation* que se desea.

III.2 Ejecución de la aplicación web

La ejecución del servidor para la aplicación web consiste en ejecutar el comando "Python app.py" en el directorio donde se encuentra ese archivo. La aplicación contiene diversas carpetas con los recursos que utiliza el servidor.

III.3 Creación de los modelos

El software para crear los modelos que hemos usado se encuentra en la libreta *jupyter* llamada "tfg-trabajo". Esta libreta contiene los distintos modelos utilizados en las Secciones 6.2, 6.3.1, 6.3.2 y 6.4. También, contiene diversos métodos para evaluar esos modelos.

Por último, también hemos escrito en un fichero txt que se incluye con el código y el TFG, todos los autores de los audios para garantizar el reconocimiento de los autores.