Escuela Politécnica Superior

Reconocimiento de aves analizando su canto con Deep Learning

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Dariel Bravo Fuentes

Tutores:
José García

Esther Sebastián González

Universitat d'Alacant
Universidad de Alicante

Septiembre 2022

Universidad de Alicante

Trabajo de Fin de Grado

Reconocimiento de aves analizando su canto con Deep Learning

Autor

Dariel Bravo Fuentes

Tutores

José García

Esther Sebastián González

Grado en Sonido e Imagen en Telecomunicación

Escuela Politécnica Superior

Universitat d'Alacant
Universidad de Alicante

‘Lo que conocemos es una gota, lo que no conocemos es un océano’

Isaac Newton

Resumen

Evaluar la presencia y abundancia de aves es importante para monitorear las especies, así como la salud general del ecosistema. Muchas aves se detectan más fácilmente por sus sonidos y, por lo tanto, el monitoreo acústico es muy apropiado. Convencionalmente, tales datos fueron recolectados por observadores expertos, pero los datos acústicos recolectados pasivamente están emergiendo como una técnica de encuesta alternativa.

En un momento en que el monitoreo automático de poblaciones y ecosistemas genera una gran cantidad de datos que los humanos ya no pueden procesar de manera efectiva, el aprendizaje profundo podría convertirse en una poderosa herramienta de referencia para ecólogos. Sin embargo, la extracción eficiente de datos precisos sobre la riqueza de especies de grandes conjuntos de datos de audio ha demostrado ser un desafío.

Los avances recientes en redes neuronales artificiales profundas (DNN) han transformado el campo del aprendizaje automático, superando con frecuencia las técnicas tradicionales de procesamiento de señales en el dominio de la detección y clasificación de eventos acústicos.

En este proyecto analizaremos varias técnicas de reconocimiento de sonido enfocado al canto de aves, teniendo en cuenta la metodología empleada en otros proyectos relacionados. Hemos desarrollado un programa capaz de entrenar un dataset con cantos de ave, para posteriormente identificar en audios de cualquier duración la presencia del ave entrenada. Se han comparado los diferentes resultados que brinda el código al cambiar ciertos parámetros que afectan tanto al entrenamiento como a la evaluación, para conseguir un análisis más preciso.

Agradecimientos

Es inevitable recordar a las personas que han me han educado, las que han aportado su grano de arena en mis momentos personales, y en las que me han apoyado directamente para poder realizar este proyecto.

En primer lugar, quiero agradecer a mis tutores Jose García y Esther, que me han estado asesorando todas las dudas que me han surgido durante el desarrollo, y se han preocupado por conocer mis avances semanales, ya que sin ellos se me habría dificultado bastante el camino del aprendizaje en el Deep Learning.

A mi madre Tania Fuentes, mi abuela Emilia, y mi familia que siempre me han apoyado para que pueda llegar hasta donde estoy y poder estudiar y dedicarme a lo que me gusta, muchísimas gracias por todo lo que me habéis dado, ya que sin vosotros nada de esto sería posible.

A mis amigos: Laura Lloret, Jethel Hernández, Jon, Helena, Vilma, Celia, Rubén, Alejandro Quesada, Mario y Arturo, que han estado en todo momento apoyándome cuando ha hecho falta.

Y por supuesto, a mis compañeros de carrera Daniel Owen, Luis Oliva, Sebastian Escamilla y Mario Aravid ya que gracias a ellos este camino de esfuerzo y aprendizaje se ha hecho mucho más ameno.

Justificación y Objetivos

La principal motivación para llevar a cabo este proyecto es la curiosidad por saber cómo entrenar a una máquina para el reconocimiento de imagen y sonido. El campo de Machine Learning representa el presente y sobre todo el futuro, ya que al ser una técnica que aporta beneficios a la seguridad, el entretenimiento, la enseñanza, el arte, entre otros, cada vez es más demandado y hoy en día es casi imprescindible.

Agrupar todos los conocimientos aprendidos para realizar este trabajo, más nuevos conceptos que he ido aprendiendo durante el desarrollo, supone para mí una todo un reto de superación. Aunque a lo largo del grado no he aprendido mucho a cerca de Machine Learning, creo que, si se me ha capacitado para saber buscar y aprender nuevos conceptos en el ámbito de la programación, por lo tanto, en este proyecto he contado con dicha capacidad más las técnicas de procesamiento de audio digital que se imparten en la carrera.

El objetivo de este proyecto es reconocer y clasificar diferentes tipos de aves a partir de grabaciones tomadas en el entorno de los humedales utilizando la técnica de Deep Learning, con el fin de establecer un censo de estas aves.

A mi familia y amigos,

todo esto es posible gracias a ellos.

ÍNDICE DE CONTENIDO

1 INTRODUCCIÓN

1.1 Contexto

Dado que el aprendizaje profundo se utiliza para detectar, identificar y clasificar a las personas en los datos de monitoreo automático, estas herramientas se pueden ampliar para ayudar a monitorear poblaciones. Por ejemplo, el tamaño de la población se puede estimar contando individuos [1] [2]. A partir de estos datos también se puede calcular la distribución o la densidad de la población como ya se ha hecho con los métodos tradicionales.

La detección de síntomas de enfermedades es un gran potencial proporcionado por el aprendizaje profundo, lo que refleja las aplicaciones existentes en disciplinas como la medicina (por ejemplo, [3]). Por ejemplo, las CNN se han utilizado para detectar la defoliación de árboles o enfermedades en los cultivos [4]. Esta tecnología podría aplicarse ampliamente a las poblaciones de plantas y animales silvestres para ayudar a encontrar indicios de cicatrices, desnutrición o la presencia de enfermedades visibles.

Dado que las actividades humanas afectan a todos los ecosistemas, una tarea importante para los ecólogos ha sido monitorearlos y comprenderlos, así como sus cambios con fines de manejo y conservación [5]. El aprendizaje profundo (DL) es un método apropiado para cumplir tales objetivos. Por ejemplo, la biodiversidad en un sitio determinado se puede estimar mediante la identificación de especies muestreadas en registros automáticos [6]. El funcionamiento y la estabilidad de los ecosistemas se pueden monitorear convirtiendo todos estos datos e interacciones de especies en modelos y/o centrándose en especies indicadoras como los murciélagos, que son muy sensibles a los cambios climáticos y de hábitat [7].

1.2 Motivación Personal

Cuando era adolescente, descubrí una aplicación para dispositivos móviles, que reconocía el nombre y artista de una canción que estaba sonando, tan solo activando el micrófono del móvil por unos segundos. En aquel momento empecé a preguntarme cómo llegaba a detectar la canción incluso habiendo ruido u otra persona hablando, fue entonces cuando descubrí la inteligencia artificial aplicada al sonido, pues en aquella época no escuchaba mucho hablar del tema.

Al acabar Bachiller, sabía que mi pasión por la tecnología era superior a otros campos, por ello decidí estudiar *Ingeniería en Telecomunicaciones* en la rama de *Imagen y Sonido* en la *Universidad de Alicante*, pues me parecía increíble el mero hecho de crear una comunicación a través de dispositivos.

Durante la carrera pude ampliar mis conocimientos sobre el funcionamiento de diversos sistemas de comunicación, pero a la hora de plantearme el tema para mi tesis, recordé que echaba algo en falta, y es la inteligencia artificial. Me pareció una idea arriesgada, ya que nunca había programado con Deep Learning, pero al ver la propuesta de mi tutor, que además trataba sobre reconocimiento del sonido, supe que era un buen reto de aprendizaje en un campo que no podía dejar atrás al finalizar el grado.

1.3 Objetivos

El principal objetivo de esta tesis es crear un programa con Python para el desarrollo de Deep Learning enfocado al reconocimiento del canto de algunas especies de aves que habitan en los humedales de Alicante, para ello, se han de cumplir los siguientes objetivos:

- Realizar un curso online de Deep Learning, para entender los conceptos teóricos.
- Explorar las técnicas más comunes de reconocimiento de sonido.
- Buscar los datasets de canto de aves más completos.
- Realizar el programa con procesado de audio.
- Entrenar el programa con los datasets encontrados, y analizar los resultados.

2 CONCEPTOS BÁSICOS

2.1 Introducción

En este capítulo se aporta la teoría de aprendizaje automatizado para el reconocimiento de audio, en concreto, de canto de aves. Se abordará el campo de Machine Learning (ML) y Deep Learning (DL), pues su comprensión es clave para llevar a cabo el proyecto.

2.2 Machine Learning

El Aprendizaje Automático es una rama de la Inteligencia Artificial que busca dotar a las máquinas de la capacidad de aprendizaje. Esta subdisciplina no programa un sistema para que se mueva, programa el sistema para que aprenda a moverse. Cuando escuchamos Machine Learning estamos ante un problema en el cual el sistema utilizado va a aprender a realizar la tarea. Existen varias subramas, en las que se divide el Aprendizaje Automático.

- Aprendizaje Supervisado: En este tipo de aprendizaje se le proporciona al sistema un conjunto de datos de entradas y sus correspondientes categorías. Estas categorías son las salidas que debería dar nuestro sistema y que las ha puesto un ser humano, de ahí el nombre de aprendizaje supervisado. El sistema ajustará sus parámetros para que su salida se parezca mucho a las etiquetas proporcionadas y así, cuando se le proporcione un nuevo dato de entrada sea capaz de clasificarlo con una probabilidad alta de acierto. Algunos de los sistemas empleados para aprendizaje supervisado son los sistemas de regresión lineal, las máquinas de soporte de vectores o las redes neuronales.
- Aprendizaje No Supervisado: En este caso solo se le introduce al sistema datos de entrada sin etiquetar. Es el propio sistema que se encarga de aprender similitudes entre ellos y alcanza abstracciones interesantes. Todavía es un campo que está en desarrollo, pero todo indica que la verdadera evolución del aprendizaje automático pasa por esta disciplina, ya que evitaría la acción humana al estar etiquetando datos de entrada.
- Aprendizaje Reforzado: Por último, tenemos un tercer paradigma dentro del aprendizaje automático que se llama el aprendizaje reforzado. En él se persigue que el sistema tome acciones dentro de un ambiente que maximice la señal de recompensa. Es un sistema muy extendido en el campo de los videojuegos, creando sistemas que son capaces de resolver problemas muy diversos.

2.3 Deep Learning

El Deep Learning lleva a cabo el proceso de ML usando una red neuronal artificial que se compone de un número de niveles jerárquicos. En el nivel inicial de la jerarquía, la red aprende algo simple y luego envía esta información al siguiente nivel. El siguiente nivel toma esta información sencilla, la combina, compone una información algo un poco más compleja, y se lo pasa al tercer nivel, y así sucesivamente. En cada capa, excepto en la primera, se calcula la entrada a cada unidad como la suma ponderada de las unidades de la capa anterior, entonces se le aplica una transformación no lineal o función de activación a la entrada para obtener una nueva representación.

2.3.1 Perceptrón

Para entender las redes neuronales vamos a adentrarnos en el sistema más sencillo y que fue el primero en desarrollarse: el perceptrón

El perceptrón fue inventado por Frank Rosenblatt[1] en el laboratorio aeronáutico de Cornell. Basándose en los primeros conceptos de neuronas artificiales, propuso la "regla de aprendizaje del perceptrón".

Un perceptrón es una neurona artificial, y, por tanto, una unidad de red neuronal. El perceptrón efectúa cálculos para detectar características o tendencias en los datos de entrada. Se trata de un algoritmo para el aprendizaje supervisado de clasificadores binarios. Ese algoritmo es el que permite que las neuronas artificiales aprendan y traten los elementos de una serie de datos.

El perceptrón desempeña un papel esencial en los proyectos de ML. Se utiliza en gran medida para clasificar datos, o como algoritmo que permite simplificar o supervisar las capacidades de aprendizaje de los clasificadores binarios.

Sin embargo, se evidenció sus limitaciones a la hora de lidiar con problemas de clasificación más complejos, como el problema de la puerta XOR. La solución fue añadir más neuronas y más capas de neuronas, creando así el perceptrón multicapa o también conocido como las redes neuronales artificiales (ANN).

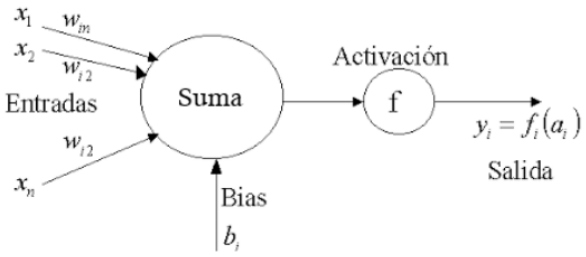

Figura 2-1. Esquema de una neurona.

En cambio, una red de neuronas que solo realiza sumas ponderadas colapsaría en una única neurona capaz de realizar todas las sumas ponderadas en una. Así, para evitar este colapso de la red y seguir manteniendo la estructura de múltiples neuronas y múltiples capas, cada neurona realiza dos tipos de operaciones. La primera es la suma ponderada que hemos hablado, una función lineal ya que es un sumatorio, y la segunda es pasar dicha suma ponderada por una función no lineal, la función de activación. Algunos ejemplos de funciones de activación las mostramos en la Figura 2-1.

[1] https://web.archive.org/web/20180722124753/https:/data-speaks.luca-d3.com/2018/07/historia-de-la-ia-frank-rosenblatt-y-el.html

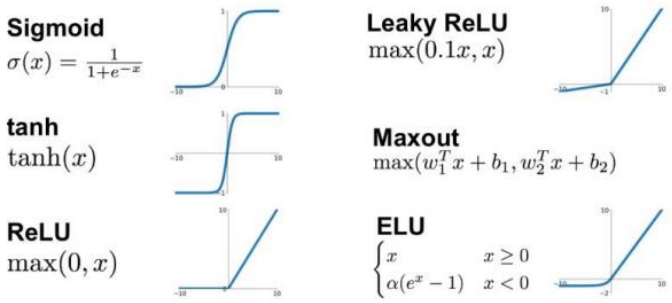

Figura 2-2. Funciones de activación

La estructura de un perceptrón tiene varias entradas:

1. Entradas: son los valores numéricos que recibe el perceptrón para predecir la salida; puede haber un gran número de entradas, pero debe ser siempre el mismo número durante la ejecución.
2. Pesos: tienen un valor aleatorio inicializado al comienzo que se multiplica por la entrada que recibe. En algunas ocasiones se le suma un parámetro fijo, denominado bias, para tener mayor control sobre la salida.
3. El sumatorio de los resultados obtenidos en el paso anterior: que es considerado la salida de este. Aunque este valor se puede ver modificado por la función de activación, que se encarga de dar una salida en un rango. Algunos tipos de funciones de activación son: sigmoide que transforma los valores en un rango de (0,1), la tangente

2.3.2 Redes Neuronales

Para explicar el funcionamiento de las redes neuronales se comenzará describiendo la red neuronal más simple que existe, una red formada por una única neurona y entrada. El esquema de esta red es el representado en la Figura 2-3.

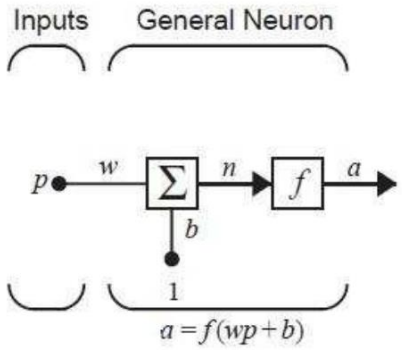

Figura 2-3: Red neuronal de única neurona y una entrada

El proceso que se sigue para obtener la salida de la red es simple. A la entrada llega un escalar *p* que es multiplicado por su peso *w*, quedando w_p, que es una de las entradas del sumatorio. La otra entrada siempre tiene el valor 1 y es multiplicado por una bias *b*. Una vez ambos términos se han sumado se obtiene que la entrada *n* a la función de transferencia es $w_p + b$. Finalmente, a la salida de la función de transferencia se tiene la salida *a* de la neurona.

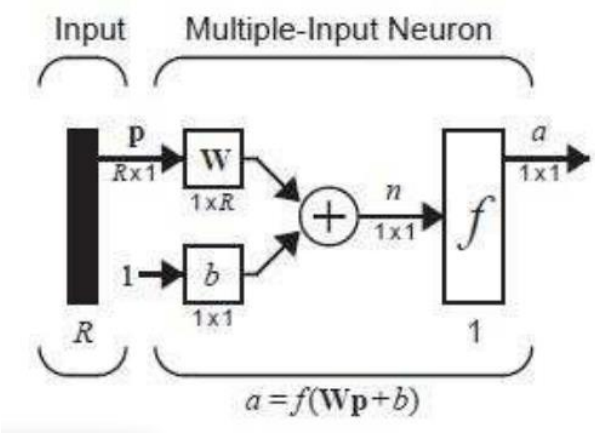

Figura 2-4. Red neuronal de una única neurona y varias entradas

Haciendo un procedimiento análogo al de la neurona con una sola entrada y teniendo en cuenta que por simplicidad de cálculo se trabajará matricialmente, se obtiene que la salida del sistema es:

$$a = f(Wp + b).$$

Ecuación 2-1

Usualmente con una única neurona no será suficiente para resolver la mayoría de los problemas prácticos. Para resolver problemas más complejos se tendrá que hacer un uso conjunto de muchas de estas neuronas simples, dando lugar a una verdadera red neuronal, en la que se tendrán cientos o incluso miles de neuronas. Es ahí donde aparece el concepto de capa, que es la agrupación de todas estas neuronas en varios conjuntos dentro de la red neuronal completa. En la Figura 2-5 se puede observar una red neuronal con múltiples capas.

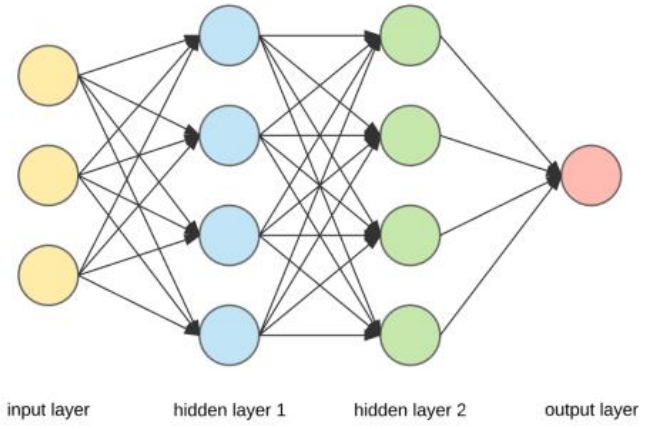

Figura 2-5. Esquema general de una red neuronal de múltiples capas

Si se considera una red neuronal con varias capas y se tiene en cuenta que cada capa tiene sus propias matrices de peso, sus propios vectores de bias y sus respectivas salidas es necesario introducir una nueva notación para poder trabajar correctamente.

En primer lugar, se tiene que existirán n_1 capas en la red neuronal. La *l*-ésima capa se denominará L_l, así la capa de entrada a la red será L_1 y la capa de salida de la red será L_{nl}. También se tendrá que $w_{ij}^{(l)}$ será el peso asociado a la conexión entre el elemento *j* de la capa *l* y el elemento *i* de la capa *l+1*. Además, $b_i^{(l)}$es la bias asociada con el elemento *i* de la capa *l+1*. Por último, se usará $a_i^{(l)}$ para denotar la activación (es decir, el valor de salida) del elemento *i* en la capa *l*. En los elementos propios de cada capa se usa un superíndice que indica en cuál de ellas se está, de esta manera si el superíndice es uno el elemento pertenece a la primera capa, si es dos pertenece a la segunda y así sucesivamente. Es interesante destacar que las entradas a las capas posteriores son las salidas de las capas que les preceden.

2.3.3 Redes Neuronales Convolucionales

Las redes neuronales convolucionales o CNN (Convolutional Neural Network) son un tipo de redes neuronales que reemplazan las multiplicaciones de matrices con convoluciones en, al menos, una capa.

Cuando se trabaja con imágenes, se suele hacer con imágenes con resoluciones muy altas, esto conlleva un grave problema en las redes neuronales convencionales, que se han visto anteriormente. Éstas, al ser full-connected, tienen cada una de las neuronas que conforman la primera capa oculta conectada con todos y cada uno de los píxeles de la imagen de entrada a la vez. Si estas imágenes de entrada son de una resolución asumible, por ejemplo 28 x 28, el número de elementos que habrá que entrenar será lo suficientemente comedido como para que el sistema pueda funcionar correctamente, aunque todo el proceso será significativamente más lento. Sin embargo, si la resolución aumenta ligeramente (y no digamos ya a resoluciones actuales) los tiempos de entrenamiento y testeo se vuelven enormes.

Una CNN es un tipo de red multicapa que consta de diversas capas convolucionales y de pooling (submuestreo) alternadas, y al final tiene una serie de capas full-connected como una red perceptrón multicapa. La entrada de una red capa convolucional suele ser, generalmente, una imagen m x m x r, donde m es tanto la altura como el ancho de la imagen y r es el número de canales (como se verá después, en este proyecto se trabaja con escala de grises por lo que r = 1). Las capas convolucionales tienen *k* filtros (o kernels) cuyas dimensiones son *n x n x q*, donde *n* y *q* son elegidas por el diseñador. Cada filtro genera mediante convolución un mapa de rasgos o características de tamaño $(m - n + 1) \times (m - n + 1) \times p$, siendo *p* el número de filtros que se desean usar. Después cada mapa es sub-muestreado en la capa de pooling con la operación "mean pooling" o "max pooling" sobre regiones contiguas de tamaño *p x p* donde p puede tomar valores desde 2 para imágenes pequeñas hasta, comúnmente, no más de 5 para imágenes grandes. Antes o después del submuestreo, se aplica una función de activación sigmoidal más un sesgo para cada mapa de rasgos.

La composición de bloques anterior se puede ver en la Figura 2-6, en la que está representada una disposición genérica de las capas.

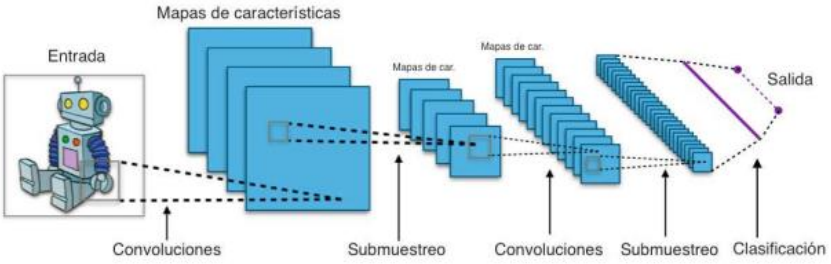

Figura 2-6. Esquema básico de una red neuronal convolucional.

Además, las redes convolucionales están diseñadas suponiendo que la entrada a la red es una imagen, lo cual permite codificar ciertas propiedades en la arquitectura, permitiendo ganar eficiencia y reducir la cantidad de parámetros usados en la red.

2.3.4 Prueba y Error

Función de coste

Una vez tenemos nuestra red es hora de entrenarla con nuestros datos y que sea capaz de resolver nuestro problema. Como el título indica este entrenamiento se hace a base de prueba y error. Al iniciar nuestra red los pesos y sesgos tendrán un valor aleatorio. Al pasar un dato de entrenamiento por nuestra red y al llegar a la capa de salida, lo más seguro es que el resultado sea erróneo. Para cuantificar este error existen numerosas funciones de pérdida (lossfunction), por ejemplo, el error cuadrático medio. Ahora queremos mejorar nuestra red para que el error disminuya. Entonces entra en juego el algoritmo de Backpropagation.

Backpropagation

El algoritmo de Backpropagation [8] calcula todas las derivadas parciales de la función de pérdidas con respecto a todos los pesos y sesgos del modelo. A priori se antoja un cálculo eterno, ya que existen miles de pesos y sesgos en una red. Sin embargo, este algoritmo se vuelve eficiente ya que va calculando las derivadas parciales de atrás hacia delante. Es decir, empieza calculando las derivadas de la última capa oculta, después con estos datos calcula los de la capa anterior, y así sucesivamente. Antes de que el algoritmo de Backpropagation se crease el cálculo de las derivadas se hacía por fuerza bruta, calculando todos los posibles caminos desde la capa de entrada hasta la capa de salida, lo que se convertía en una tarea ineficiente.

Función de reducción del coste

Una vez tenemos todas las derivadas parciales necesitamos un algoritmo de reducción de error. Existen varios, pero el más conocido es el algoritmo del descenso del gradiente. Estos tipos de algoritmos se nutren de las derivadas parciales calculadas en el paso anterior, es decir, el gradiente, y calcula los valores que tienen que poseer los pesos y sesgos para que en la próxima iteración el error disminuya. Mostramos un ejemplo en la Figura 2-7.

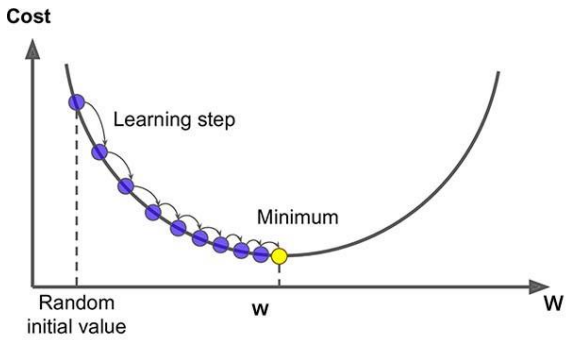

Figura 2-7. Algoritmo de descenso del gradiente

Así se habría completado una iteración. Sin embargo, los datos de entrenamiento no se constituyen de una sola muestra. Consiste en un gran set de datos, que se van introduciendo en lotes de datos más pequeños (batches) a lo largo de más de una iteración (epoch). Así el algoritmo de reducción del error irá mejorando hasta alcanzar el mínimo global, aquel conjunto de valores de pesos y sesgos que hayan reducido al mínimo el error. Para llegar a él es necesario ajustar un valor llamado learning rate o step size. Cuanto mayor es este valor, más grande van a ser los pasos que se dé y más rápido iterará, pero puede que nunca llegue al mínimo global. Cuanto menor es este valor, más pequeños serán los pasos y con mayor seguridad se alcanzará el mínimo global, pero se necesitaría de mucho tiempo para alcanzarlo. Hay que alcanzar un compromiso entre un valor mayor o menor. Ambos casos los podemos apreciar en la Figura 2-8.

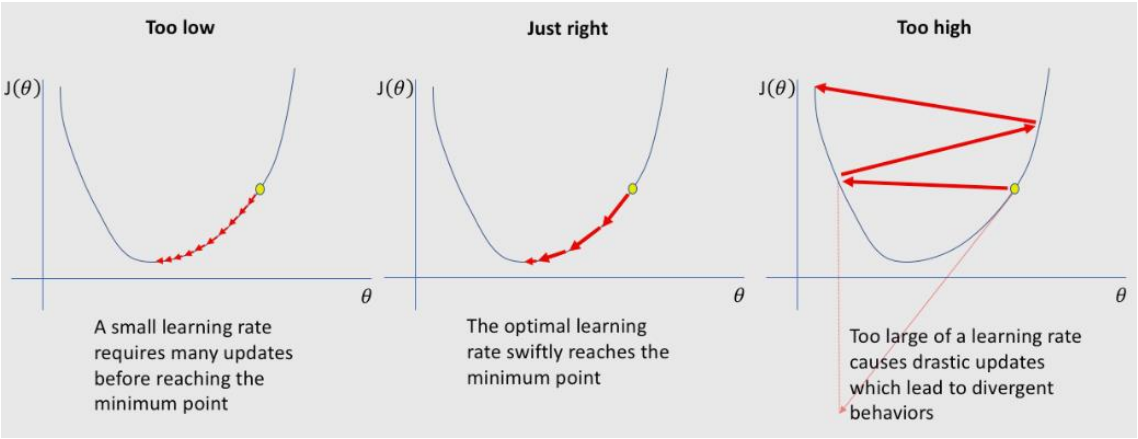

Figura 2-8. Ejemplos de learning rates.

2.3.5 Resultado

Una vez entrenada la red con todos nuestros datos podemos tener varios resultados. El primero de ellos es que la red no converge y no sea capaz de proporcionarnos un resultado. El siguiente caso es que la red sí converge y nos proporcione un resultado. Sin embargo, el resultado puede ser satisfactorio o no según tres tipos de situaciones, que pasamos a explicarlas a continuación (Figura 2-9).

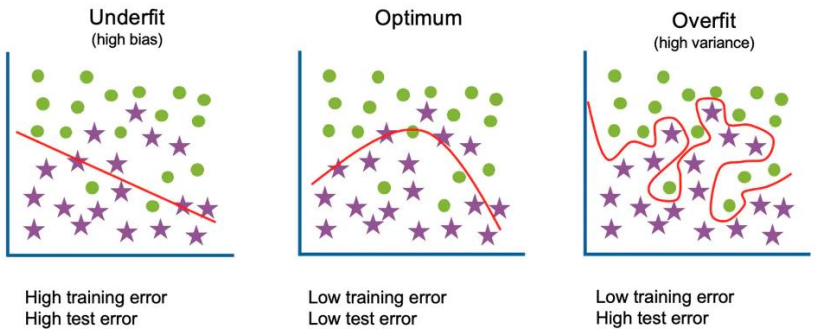

Figura 2-9. Ejemplos de resultados de una red.

Subajuste (Underfit)

El subajuste es el fenómeno por el cual la red no ha sido capaz de ajustarse a los datos de entrenamiento y no ejerce la clasificación de la manera adecuada. Uno de los principales motivos del subajuste se debe a la rigidez del modelo. Es decir, el modelo es poco complejo para los tipos de datos que queremos clasificar y una solución sería incluir más capas de neuronas para que el modelo gane en complejidad y abstracción.

Buen ajuste (Optimum)

Un buen ajuste se produce cuando el modelo ha sido capaz de adaptarse a los datos de entrenamiento y realiza una buena clasificación. Además, el modelo ha conseguido la capacidad de generalizar, por lo tanto, cuando se le introduzca un nuevo dato nunca visto la red lo clasificará con una alta probabilidad de acierto.

Sobreajuste (Overfit)

El sobreajuste se produce cuando entrenamos la red con muchos datos. Cuando esto ocurre, el modelo comienza a aprender del ruido y las entradas de datos inexactas en nuestro conjunto de datos. Entonces, el modelo no categoriza los datos correctamente, debido a demasiados detalles y ruido.

Entrenamiento, validación y test

Para evitar el problema del sobreajuste se aplican distintas técnicas a los datos. Una de las más comunes es dividir el dataset completo de entrenamiento en dos subconjuntos: entrenamiento y validación. Normalmente se emplea las proporciones aproximadas de 80% de los datos para el conjunto de entrenamiento y el 20% restante para el conjunto de validación. En una iteración, el conjunto de entrenamiento servirá para calcular el error de la red y modificar los pesos y sesgos de nuestro modelo. Después se le pasará el conjunto de validación para calcular el error del modelo, pero a diferencia con el error de entrenamiento, el error de validación no servirá para modificar pesos y sesgos. A lo largo del entrenamiento veremos como el error de entrenamiento y el de validación irán disminuyendo paulatinamente. Sin embargo, llegará un punto en el que el error de entrenamiento seguirá disminuyendo, pero el error de validación comenzará a crecer. Es en este punto cuando la red está empezando a memorizar los datos de entrenamiento y está

perdiendo capacidad de generalización. Por lo tanto, el punto para que la red esté bien ajustada es aquel con el mínimo error de validación (Figura 2-10).

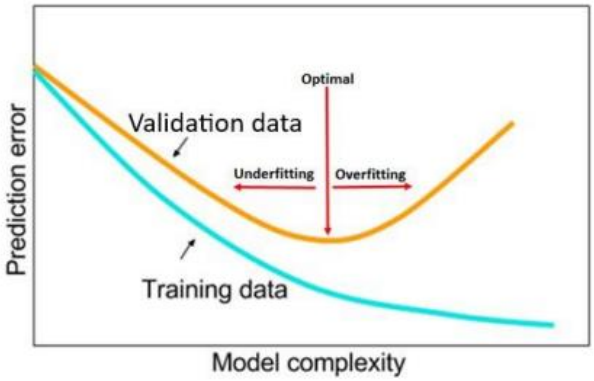

Figura 2-10. Error de entrenamiento y validación

Normalmente existe otro conjunto de datos que no se emplean ni en el entrenamiento ni en la validación. Una vez que el modelo ha sido entrenado se le pasa este conjunto de datos para hacer pruebas, llamado conjunto de test, y calcular parámetros como la precisión del modelo o el error. Este conjunto debe poseer igual número de datos para todas las clases y así al realizar las medidas de la forma más equitativa posible. A veces no existen datasets con un número elevado de muestras para cada clase, o tienen un número desigual de datos para cada clase. En este caso se emplean técnicas como la de Data Augmentation [9] para incrementar el número de datos y así generar un buen dataset. En el caso de un conjunto de datos de imágenes, aplicar Data Augmentation consiste en aplicar rotaciones, incrementos o decrementos de tamaños para así poseer más variedad de datos con los que trabajar. Habríamos completado así el entrenamiento de una red neuronal, explicando todos los elementos.

2.4 Deep Learning para el análisis de audio

El sonido es una onda de presión longitudinal formada por las compresiones y rarefacciones de las moléculas de aire, que se propaga en dirección paralela a la aplicación de energía. La representación de esta onda en el dominio del tiempo no es fácil de interpretar directamente. En muchas ocasiones resulta imposible localizar sonidos en una forma de onda e identificar a simple vista de qué clase de sonido se trata. Es por ello por lo que las representaciones en el dominio de la frecuencia o las representaciones tiempo-frecuencia cobran mayor importancia. En esta sección vamos a explicar cómo se calculan estas representaciones y qué utilidad tiene para este proyecto.

2.4.1 Conversión analógico digital

Podemos obtener el sonido grabándolo con un sistema analógico, lo que obtendríamos una señal dependiente en el tiempo *x(t)*. Esta señal, si la representamos en el tiempo, veríamos su forma de onda. Esta señal no puede ser almacenada en un ordenador para su análisis, ya que es una

señal continua y posee infinitos valores. Es por eso por lo que aplicamos una conversión analógico-digital (Figura 2-11), que consta de las siguientes fases:

- Filtrado paso bajo: la señal x(t) se hace pasar por un filtro cuya frecuencia máxima la llamaremos B. Normalmente B se encuentra en torno a los 20 kHz, ya que esa es la frecuencia máxima de audición del ser humano. Los valores de la señal por encima de esa frecuencia no los vamos a tener en cuenta, ya que serían inaudibles para nosotros y una carga computacional innecesaria. •
- Muestreo temporal: de la señal x(t) obtenemos muestras en el dominio del tiempo con una frecuencia de 2*B. Es decir, cada 1/2B segundos guardamos el valor de la señal x(t) en ese momento. Se elige este valor de frecuencia para evitar el fenómeno de aliasing. La frecuencia de muestreo en un CD convencional es de 44.1 kHz, algo más del doble de la frecuencia máxima de audición humana.
- Cuantificación: los valores de amplitud de cada muestra también pueden tener valores infinitos. Es por ello por lo que se limitan los valores que pueden tener las muestras a unos predefinidos para preservar la capacidad de memoria del ordenador.

Figura 2-11. Esquema de conversión analógico-digital

Tras esta conversión analógica-digital ya tenemos nuestro audio almacenado en el ordenador y podemos trabajar con él. El siguiente paso para la obtención de una representación tiempo-frecuencia es convertir la señal de audio en el domino de la frecuencia.

2.4.2 Dominio Frecuencial

Para convertir la señal temporal al dominio de la frecuencia se emplea la transformada discreta de Fourier (DFT) cuya fórmula se muestra en la Ecuación 2-2.

$$X(f) = \sum_{n=-\infty}^{+\infty} x[n] * e^{-i2\pi fn}$$

Ecuación 2-2. Transformada Discreta de Fourier.

Para cada muestra de la señal temporal se convierte al dominio frecuencial lineal. Normalmente el espectro de X[f] se aproxima al cálculo de la DFT aplicando una ventana de tamaño N a la señal x[n]. Así resulta la transformada de Fourier de tiempo reducido (STFT). La fórmula para calcular la componente f en la muestra t de la señal x[n] se muestra en la Ecuación 2-3.

$$X[t,f] = \sum_{k=0}^{N-1} w[k] * x[tN+k] * E^{-\frac{i2\pi kf}{N}}$$

Ecuación 2-3. Transformada Rápida de Fourier.

En la Ecuación 2-3 w[k] es una ventana que puede adquirir diversas formas: rectangular, Hamming, Blackman, etcétera. El cómputo de la STFT se realiza mediante un algoritmo llamado transformada rápida de Fourier (FFT). Tras calcularlo tenemos entonces un conjunto de grupos de muestras de x[n] enventanados y pasados al dominio de la frecuencia. De cada uno de estos grupos podemos adquirir una representación frecuencial, poniendo en el eje de abscisas las frecuencias y en el eje de ordenadas los niveles que alcanzan dichas frecuencias en la ventana. Se suele pasar a unidades logarítmicas, por lo que el eje de ordenadas estaría en decibelios.

2.4.3 Espectrograma

Una transformada de Fourier convierte una señal a las frecuencias de sus componentes, pero pierde toda la información de tiempo. En comparación, la transformada de Fourier en tiempo reducido (STFT) divide la señal en ventanas de tiempo y ejecuta una transformada de Fourier en cada ventana, preservando parte de la información de tiempo y devolviendo un tensor 2D en el que puede ejecutar convoluciones estándar[2].

En el eje de abscisas se representa el tiempo transcurrido y en el eje de ordenadas las frecuencias. El nivel de intensidad de cada frecuencia se representa en el espectrograma, donde las zonas más oscuras indican mayor nivel de decibelios y las zonas más claras menor nivel. En la Figura 2-12 tenemos un ejemplo de espectrograma de una pista de canto de ave.

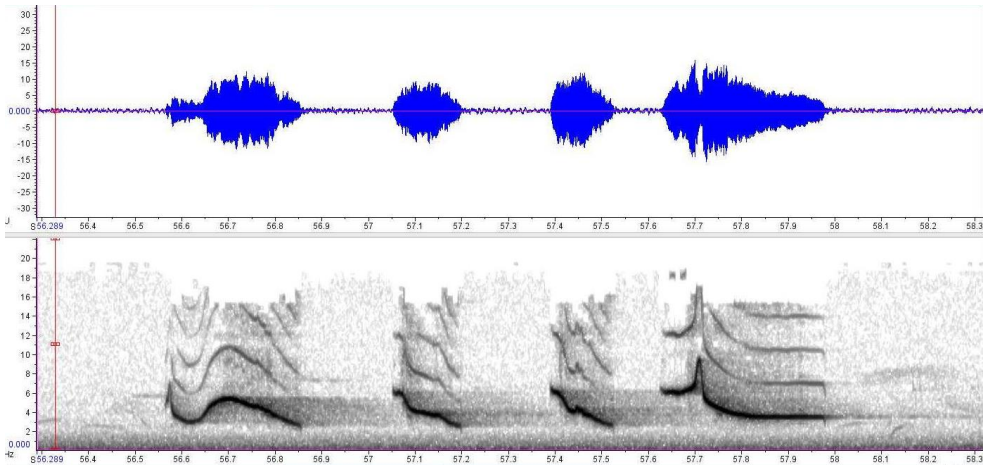

Figura 2-12. Amplitud de onda frente al espectrograma del canto de un ave

De esta manera podremos trabajar con el audio con técnicas de Deep learning aplicadas a imagen, ya que resulta mucho más práctico y sencillo entrenar una red con los datos del espectrograma, a identificar patrones en el dominio del tiempo.

[2] https://www.tensorflow.org/tutorials/audio/simple_audio

3 ESTADO DEL ARTE

3.1 Introducción

A continuación, haremos el estado del arte del reconocimiento de canto de aves. En este capítulo veremos las distintas arquitecturas existentes, la cantidad óptima de datos para el entrenamiento, y los diferentes datasets encontrados.

3.2 Arquitectura de clasificación y detección

En el capítulo anterior, ya vimos cómo funciona el Deep Learning, y que en su aplicación para audio ese necesario emplear espectrogramas. Sin embargo, también tendremos que considerar la clasificación y detección, puesto que son elementos fundamentales en muchos flujos de trabajo.

Existen varios métodos de clasificación [10], entre los más comunes, distinguimos los siguientes:

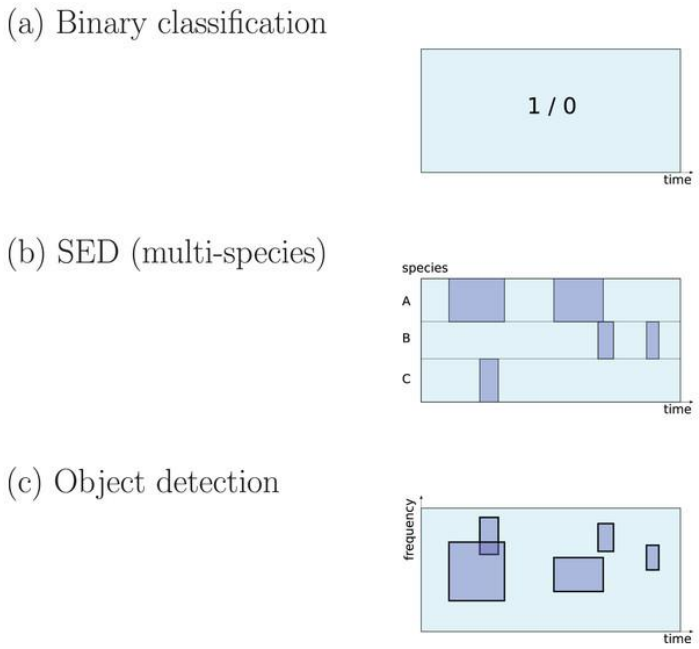

Figura 3-1. Los tres enfoques más comunes para la detección de sonido.

- Clasificación binaria: para un clip de audio dado, devuelve una decisión binaria de sí/no dependiendo de si detecta o no la señal de interés [11]. Es fácil de implementar, ya que la clasificación binaria es una tarea fundamental en DL y no requiere que los datos se etiqueten con detalles de alta resolución.

- Detección como transcripción: en la serie de desafíos y talleres DCASE (Detección y clasificación de escenas y eventos acústicos), la tarea de transcribir eventos de sonido,

potencialmente para múltiples clases en paralelo, se denomina detección de eventos de sonido (SED). Por lo general, se ha abordado mediante el entrenamiento de DL para etiquetar cada pequeño paso de tiempo (por ejemplo, un segmento de 10 ms o 1 s) como positivo o negativo, y las secuencias de positivos se fusionan luego en regiones de eventos predichos [12] [13].

- Detección de objetos: consiste en estimar múltiples cuadros delimitadores que indican ubicaciones de objetos dentro de una imagen. Llevado a los datos del espectrograma, cada cuadro delimitador representaría los límites de tiempo y frecuencia para un "objeto" (un evento de sonido). Esto no se ha utilizado con frecuencia en bioacústica, pero puede estar aumentando su interés [14] [15] [16].

Para estas tres configuraciones de tareas, se encontró que las redes basadas en CNN tienen un rendimiento sólido, superando otras técnicas de ML [17] [18] [19] Dado que el formato de datos en cada una de las tres configuraciones de tareas es diferente, las capas finales (de salida) de una red toman una forma ligeramente diferente, al igual que la función de pérdida utilizada para optimizarlas [20].

3.3 Cantidad de datos necesarios

Quizás el mayor desafío para el DL supervisado radica en la necesidad de un gran conjunto de datos de entrenamiento para lograr una alta precisión. Cuando los algoritmos se entrenan con ejemplos, solo pueden detectar dichos ejemplos. Por lo tanto, los conjuntos de datos de entrenamiento a menudo contienen miles de millones de ejemplos, dependiendo de la tarea, el número de ítems a detectar y el desempeño deseado [21]. No obstante, se han obtenido buenos resultados con conjuntos de datos de entrenamiento más pequeños, con solo varios cientos de ejemplos por clase [22], facilitando el trabajo de recopilación de datos para la mayoría de los campos de ecología. Sin embargo, en general, cuanto mayor sea el conjunto de datos de entrenamiento, mejor será la precisión de la clasificación.

También debemos tener en cuenta que el conjunto de datos de entrenamiento generalmente se divide en dos subconjuntos, uno para entrenar efectivamente el modelo, y el otro para evaluar su rendimiento. Por lo tanto, es esencial que cada subconjunto contenga suficientes ejemplos que sean representativos de todas las categorías de clasificación para permitir un entrenamiento eficiente.

Esta necesidad de datos también implica que el conjunto de datos que queremos analizar se debe tener un tamaño suficiente y que encontrar el umbral adecuado de tamaño es crítico. Por ejemplo, en el procesamiento acústico, al menos 36 horas de grabación fueron necesarios para que un algoritmo de aprendizaje profundo se convirtiera más eficiente que la escucha humana [23].

De acuerdo con la red entrenada que se ha tomado como base para este proyecto[3], alrededor de 200 grabaciones son suficientes para obtener buenos resultados ante grabaciones con poco ruido.

[3] https://github.com/nicknochnack/DeepAudioClassification

3.4 Datasets

Los repositorios de audios han sido de utilidad para este proyecto, pues encontrar datasets de audios específicos para canto de aves, concretamente las especies que habitan en Alicante, no ha sido tarea fácil. En su lugar, se ha tomado grabaciones de dos páginas webs y se ha ido recortando las grabaciones que en ellas se proporciona, en pistas de tamaño similar en las que se escucha al ave a entrenar.

3.4.1 Xeno-Canto

Xeno-canto.org es un sitio web dedicado a compartir sonidos de aves de todo el mundo. Cuenta con más de setecientas mil grabaciones subidas ya sea por un científico investigador, un observador de aves o un amateur.

3.4.2 Macaulay Library

Macaulaylibrary.org es un repositorio de imágenes y grabaciones de aves similar a xeno-canto. Forma parte de Cornell Lab, y en este caso nos encontramos con más de un millón y medio de grabaciones, pero a diferencia de xeno-canto que los audios se pueden descargar de manera inmediata, en Macaulay hay que enviar una petición de los audios que se quieren descargar por correo poniendo enlace por enlace cada grabación.

3.5 Especies

En la provincia de Alicante se han datados 351 especies de aves[4], de las cuales 6 son introducidas y 20 globalmente amenazadas. En este proyecto se busca entrenar la red con las aves cuyo canto presente un espectrograma distintivo al resto, con la finalidad de reducir el error. Dichas aves son: Zorzal Común (Thordus Philomenus), Codorniz Común (*Coturnix Coturnix*), Mosquitero Musical (*Phylloscopus trochilus*), Curruca Zarcerilla (*Curruca curruca*), Zorzal Alirrojo (*Turdus iliacus*).

Además, otro factor a tener en cuenta son los datasets, al no disponer de una cantidad razonable de grabaciones para todas las especies, se ha procurado encontrar aves con más de 1000 grabaciones en Xeno Canto.

[4] https://avibase.bsc-eoc.org/checklist.jsp?lang=ES&p2=1&list=clements&synlang=ES&region=ESvaal&version=text&lifelist=&highlight=0

4 MATERIALES Y METODOLOGÍA

En este capítulo vamos a hablar de todas las herramientas utilizadas para el desarrollo del proyecto, tanto a nivel de software como de hardware.

4.1 Software

4.1.1 Lenguaje de programación

Con la llegada del DL, se desarrollaron y lanzaron muchos marcos de software para aumentar la accesibilidad y facilitar el desarrollo de modelos de DL. Cada una de ellas (bibliotecas y frameworks) ofrecen diferentes niveles de abstracción, flexibilidad y rendimiento.

En la actualidad, Python es uno de los lenguajes de programación más usados en todo el mundo, pues en el campo de la Inteligencia Artificial, el Machine Learning y el Deep Learning la mayoría de código desarrollado es en este lenguaje. Si bien es cierto que durante el grado no hemos dado una asignatura de programación en Python como tal, es un lenguaje de programación muy fácil de comprender y el aprendizaje se puede realizar de manera autodidacta gracias al conocimiento en otros lenguajes como C y Matlab que sí hemos aprendido en la carrera. Es por ello que el código para entrenar las redes en este proyecto se ha desarrollado en Python.

4.1.2 Entorno de programación y librerías

El motor principal de trabajo ha sido Google Colab Pro, debido a la baja capacidad de la GPU de mi portátil personal. La versión gratuita que ofrece Colab fue suficiente para desarrollar el tratamiento de audio (trabajar la frecuencia, crear el espectrograma) pero no para entrenar a la red, es por ello que he optado por la versión Pro que ofrece una mejor GPU, con suficiente potencia para las tareas que requiere este proyecto.

Las librerías utilizadas son:

- Numpy: principal librería para la computación científica. Posee funciones para trabajar en el campo del álgebra, transformadas de Fourier, matrices N-dimensionales y el campo de la estadística.
- Pandas: librería que posee herramientas para el análisis de datos de manera rápida, potente y fácil de usar.
- Matplotlib: librería para la representación y visualización de datos de distintas formas, ya sea estática, animada, interactiva, en 2D, en 3D, etcétera

4.1.3 Framework

Tensorflow es una biblioteca de código abierto usada para aprendizaje automático. Es sin ninguna duda el framework más popular en este momento, ya que muchas marcas importantes como Uber, Airbnb, Nvidia y muchas otras lo utilizan, además, se utiliza en varios lenguajes de

programación [24]. Utiliza una estructura de grafo de flujo de datos, en el cual los nodos del grafo representan operaciones matemáticas y las conexiones del grafo son los conjuntos de datos multidimensionales, llamados tensores. Esta aproximación tiene varias ventajas para el desarrollo de modelos de DL, ya que permite calcular la derivada de cada operación de forma individual y aplicar la regla de la cadena, lo cual es útil para calcular la función de pérdida basada en gradientes descendientes, que es muy comúnmente usada en este entorno. Esto también permite hacer de forma paralela operaciones entre Graphics Processing Unit (GPU) y Central Processing Unit (CPU). Está escrita principalmente en C++ y proporciona una interfaz de alto nivel en Python, aunque también soporta otros lenguajes como JavaScript, C++, Go, C#, Julia y Java. Además, se pueden desarrollar grafos tanto en la CPU como en la GPU.

4.2 Hardware

Todo el proyecto lo he llevado a cabo desde mir ordenador personal, que como he comentado, no posee las características necesarias para trabajar con DL, sus características son:

Ordenador Personal	
GPU	Gráficos Intel® UHD 600
CPU	Intel(R) Celeron(R) N4000 CPU @ 1.10GHz 1.10 GHz
RAM	8 GB DDR4 SDRAM
Disco Duro	SATA de 500 GB, 5400 rpm

Tabla 4-1. Especificaciones del Hadware de Ordenador personal.

Es por ello por lo que he optado por utilizar el servicio de Google Collab Pro, cuyas características[5] son las siguientes:

Ordenador Colab	
GPU	Acelerador Tesla K80 12 GB de VRAM GDDR5
CPU	Intel Xeon a 2.20 GHz
RAM	13 GB
Disco Duro	Almacenamiento en Google Drive

Tabla 4-2. Especificaciones del Hardware de Colab

[5] https://geekflare.com/es/google-colab/

5 DESARROLLO

5.1 Introducción

En este capítulo, veremos las técnicas utilizadas para llevar a cabo este proyecto. En él, nos encontramos con las distintas fases del código, y los pasos a seguir en cada una de ellas. Comenzaremos analizando como se trata el audio para adaptarlo al entrenamiento, así como generar los datasets a nivel de código. Además, veremos como generar el tamaño de las ventanas necesarias para la predicción, así como el histograma la fase de entrenamiento.

5.2 Fase de tratamiento de audio

5.2.1 Función de Cargar Audio

Primero debemos crear la función que nos permita subir un audio y convertir su frecuencia de muestreo en un estándar, para ello hemos utilizado la función `load_wav_16k_mono(filename)` que nos brinda la página de Tensorflow[6]. De esta manera, siempre estaremos trabajando con un audio de 16 kHz, y también, con un solo canal (mono), permitiéndonos introducir cualquier tipo de audio en formato wav.

5.2.2 Crear un Dataset

Para crear el dataset con el que entrenaremos la red, debemos definir en una variable la ruta de la carpeta que contiene los audios a entrenar en cuestión, y en otra variable, la ruta de un conjunto de audios que representen el ruido ambiente (sonido de agua, grillos, ratas, etc.).

Una vez tenemos la ruta definida, creamos dos vectores que contienen los audios la carpeta que representan, con la función `tf.data.Dataset.list_files()`, que enlista los audios de la manera que requiere Tensorflow en la fase de entrenamiento[7].

Por último, necesitamos unir ambos vectores de manera que se diferencien los datos de uno con los del otro, para ello, utilizamos el siguiente código:

```
positives=tf.data.Dataset.zip((pos,tf.data.Dataset.from_tensor_slices(tf.ones(
len(pos)))))
negatives=tf.data.Dataset.zip((neg,tf.data.Dataset.from_tensor_slices(tf.zeros
(len(neg)))))
data=positives.concatenate(negatives)
```

Con estas funciones estaríamos asignando un 1 al nombre de cada pista del canto que vamos a entrenar (pos) y un 0 al de audios del ruido ambiente (neg). Hasta ahora solo estamos trabajando con rutas, es decir, aún no hemos cargado los audios, solo estamos preparando el dataset para que una vez sea cargado, el programa distinga un conjunto de grabaciones del otro.

[6] https://www.tensorflow.org/tutorials/audio/transfer_learning_audio
[7] https://www.tensorflow.org/api_docs/python/tf/data/Dataset

5.2.3 Determinar el tamaño de las ventanas

A la hora de analizar un audio en la fase de experimentación, debemos tener en cuenta que la longitud de los audios introducidos probablemente sea superior a los de entrenamiento, por lo que dividiremos el archivo wav a analizar en varias ventanas. La longitud de dichas ventanas vendrá dada por la media de la duración de las pistas de nuestro dataset, para obtener primero todas las duraciones utilizaremos el siguiente código:

```
lengths=[]
for file in os.listdir(os.path.join('data', 'Carpeta de Grabaciones')):
    tensor_wave=load_wav_16k_mono(os.path.join('data', 'Carpeta de Grabaciones
,file))
    lengths.append(len(tensor_wave))
```

Estas líneas de código muestran cómo guardar en el vector lenghts las longitudes de todos los audios, mediante un bucle que carga cada uno de ellos. Posteriormente usando la función tf.math.reduce_mean() sobre el vector lenght, obtendremos la longitud media de los audios. Dicha longitud viene dada en muestras, pero si queremos conocer su valor en segundos (aunque no es necesario para su posterior uso) dividiremos el valor entre la frecuencia de muestreo (16000).

5.2.4 Obtener Espectrograma

Como bien comentamos en el capítulo Concepto Básicos (Deep Learning para análisis de audio – Espectrograma), trabajaremos con la STFT para la representación del espectrograma, cuya estructura, basándonos en el modelo que brinda Tenwsorflow[8] es la siguiente:

```
def preprocess(file_path,label):
    wav=load_wav_16k_mono(file_path)
    wav=wav[:48000]
    zero_padding=tf.zeros([48000]-tf.shape(wav),dtype=tf.float32)
    wav=tf.concat([zero_padding,wav],0)
    spectrogram=tf.signal.stft(wav,frame_length=320,frame_step=32)
    spectrogram=tf.abs(spectrogram)
    spectrogram=tf.expand_dims(spectrogram,axis=2)
    return spectrogram,label
```

5.3 Fase de entrenamiento

Nuestro modelo cuenta con dos capas convolucionales 2D, todas con la función de activación ReLu. Los filtros de estas capas son de tamaño 3x3, y las capas convolucionales cuentan con 16 filtros. Tras esta primera mitad de arquitectura le sigue una capa Flatten seguida de dos capas Dense, la primera está conectada con activación ReLu, y la última con activación Sigmoid.

[8] https://www.tensorflow.org/tutorials/audio/simple_audio

Como función de reducción del coste, también llamado optimizador, elegimos Adam, que según Diederik P. Kingma [25], el método es " computacionalmente eficiente, tiene pocos requisitos de memoria, invariante al reajuste diagonal de gradientes y es muy adecuado para problemas que son grandes en términos de datos / parámetros ".

Una vez tenemos el modelo, ya solo falta entrenarlo, pero para ello necesitamos conocer el número de epochs necesarios. Veremos en el próximo capítulo cuántos son necesarios.

6 EXPERIMENTACIÓN

6.1 Introducción

En este capítulo pondremos a prueba nuestro programa, y analizaremos los errores encontrados, así como sus posibles soluciones. Primero nos enfocaremos en probar un número elevado de epochs para obtener el número óptimo observando las gráficas de precisión y pérdidas. Por último, veremos los resultados de la fase de prueba y comentaremos el porcentaje de acierto.

6.2 Optimización del número de epochs

Recordemos que estamos trabajando sobre la herramienta Colab que nos brinda Google, la cual nos agiliza los tiempos de ejecución respecto a mi ordenador personal, pero tiene la desventaja de que los tiempos de ejecución son limitados, y todo lo que hayamos entrenado se borrará al pasar un tiempo. Lo ideal sería poder entrenar todas las aves y posteriormente analizar un audio, para determinar que ave ha tenido mayor peso, pero dadas las condiciones, probaremos cada especie por separado, indicando el número de epochs necesarios y el porcentaje de acierto en las predicciones.

Las predicciones vienen dadas por valores entre 0 y 1, eligiendo en cada audio, el valor máximo de todas las ventanas. Dicho valor, será filtrado por un umbral de 0.9 (valor que da bastante precisión) para todas las especies, convirtiéndose en 1 si lo supera, o en 0 si es inferior. Esto trae consigo un problema, y es que es probable que en audios que no corresponda a la especie, obtengamos un uno, significando esto que en dicha grabación se percibe a la especie. La forma de evitarlo es creando el código unificado, pues la especie que realmente represente ese audio tendrá unos valores de predicción superior. Por ejemplo: hemos definido un umbral de 0.7 para el Zorzal Común, pero un audio de la Codorniz presenta un máximo de 0.742, por lo que su valor torna a 1 y tenemos una falsa predicción, en cambio, si analizamos este mismo audio con la red entrenada para la Codorniz obtenemos un máximo de 0.953, por lo que, al comparar ambos resultados, el audio se clasificaría correctamente.

Comencemos con el test del número de epochs, según el código que ha sido de referencia[9] para un conjunto de aproximadamente 200 audios, 4 epochs debería estar ajustado, no obstante, veamos que sucede cuando elevamos a 8:

[9] https://github.com/nicknochnack/DeepAudioClassification

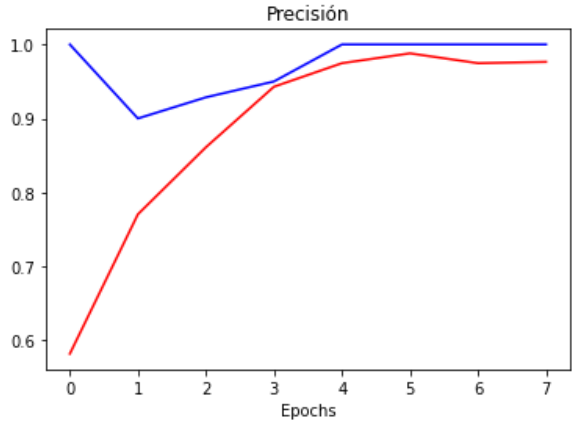

Figura 6-1. Precisión del entrenamiento de Zorzal Común con 8 epochs. Curva Roja: Test, Azul: Validación.

En la Figura 6-1 podemos observar la precisión que va adquiriendo la red por cada número de epoch, siendo 0 el primer epoch y 7 el octavo. Al parecer, 5 epochs son suficientes para obtener unos resultados precisos sin crear un sobreajuste al sistema, ya que en el sexto epoch (valor 5 en el eje de las abscisas) no mejora demasiado, y a partir de este, la precisión se mantiene lineal.

Por ello, tomaremos 5 epochs como referencia para cada especie, pues hemos dotado al dataset de la misma cantidad de audios para cada una de las especies (200) cuyas duraciones no superan los 5 segundos. Debemos de tener en cuenta que 200 audios para un dataset es poco, así como el número de epochs seleccionados.

Además, desde el segundo epoch dejamos de tener pérdidas relevantes como se observa en la Figura 6-2.

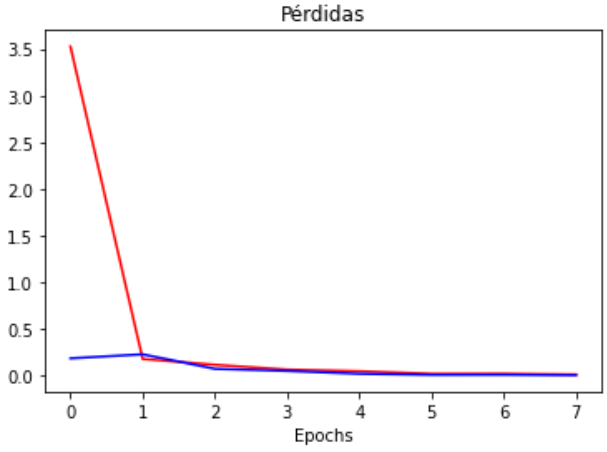

Figura 6-2. Pérdidas del entrenamiento de Zorzal Común con 8 epochs. Curva Roja: Test, Azul: Validación.

6.3 Resultados

6.3.1 Zorzal Común (*Turdus philomelos*)

El canto del Zorzal Común se caracteriza no solo por su corta duración, sino también por su estrecho contenido en frecuencia como se observa en la Figura 6-3. Esto lo hace difícil de detectar cuando el ruido de fondo es considerable, por ello, es necesario tener bastantes muestras en el dataset y así disminuir la influencia de ruido.

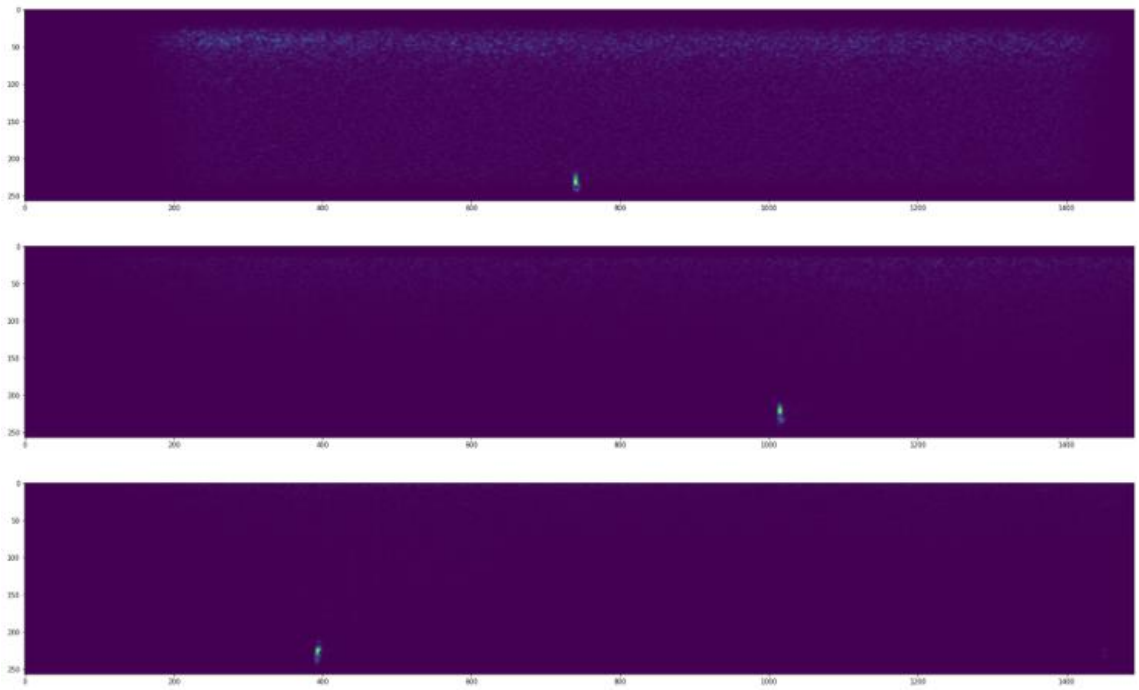

Figura 6-3. Ejemplos de espectrograma de Zorzal Común.

Zorzal Común		
Audios de evaluación	Aciertos	% Acierto
50	32	64%

Tabla 6-1. Resultados de experimentación con 200 audios de Zorzal Común.

El porcentaje de acierto obtenido es el más bajo de las aves entrenadas en este proyecto, pues todos los audios encontrados para la fase de evaluación contenían ruido de fondo, en su mayoría ruido rosa, probablemente generado por alguna cascada o río cercano.

6.3.2 Codorniz Común (*Coturnix coturnix*)

En el caso de la Codorniz común, nos encontramos con un canto más distinguido, pues presenta tres repeticiones que abarcan más espacio que el Zorzal en el dominio de la frecuencia como se puede ver en la Figura 6-3.

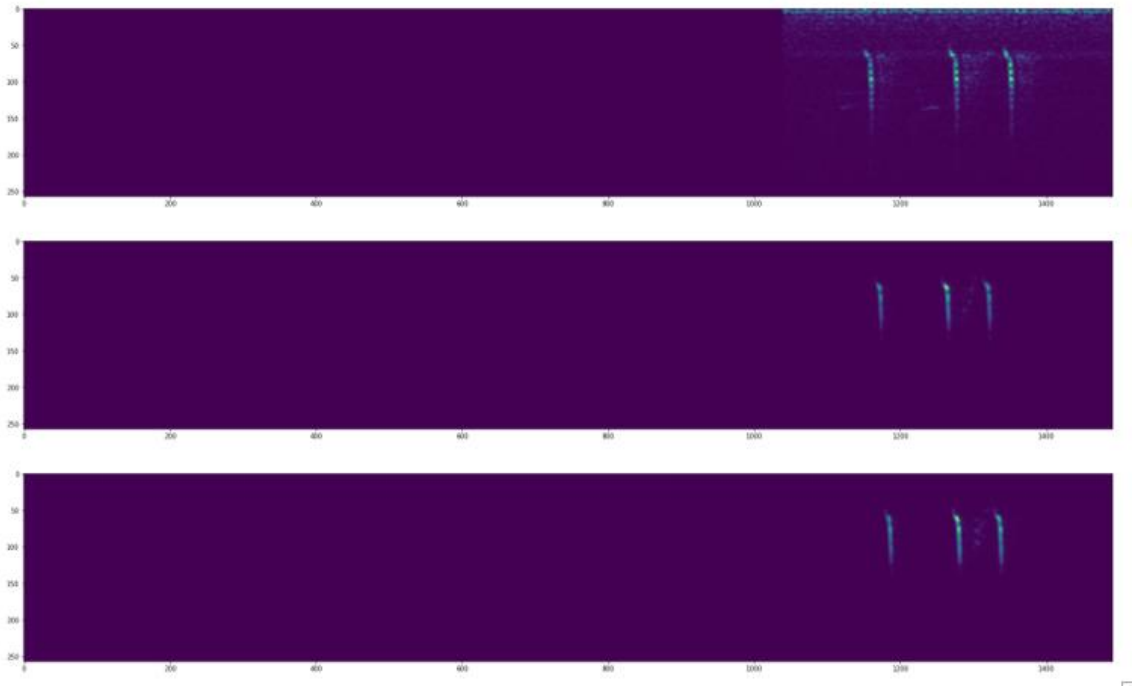

Figura 6-4. Ejemplos de espectrograma de Codorniz común

Codorniz Común		
Audios de evaluación	Aciertos	% Acierto
50	43	86%

Tabla 6-2. Resultados de experimentación con 200 audios de Codorniz Común.

En este caso obtenemos un resultado más preciso, pues como ya hemos comentado, el canto de la Codorniz es más distintivo y, además, hemos podido encontrar audios más limpios para esta especie, aunque también se ha evaluado con ejemplos que contienen ruido.

6.3.3 Zorzal Alirrojo (*Turdus iliacus*)

El canto del Zoral Alirrojo es similar al del común, pero abarca más tiempo, por lo que es ligeramente más fácil de detectar ante situaciones de ruido.

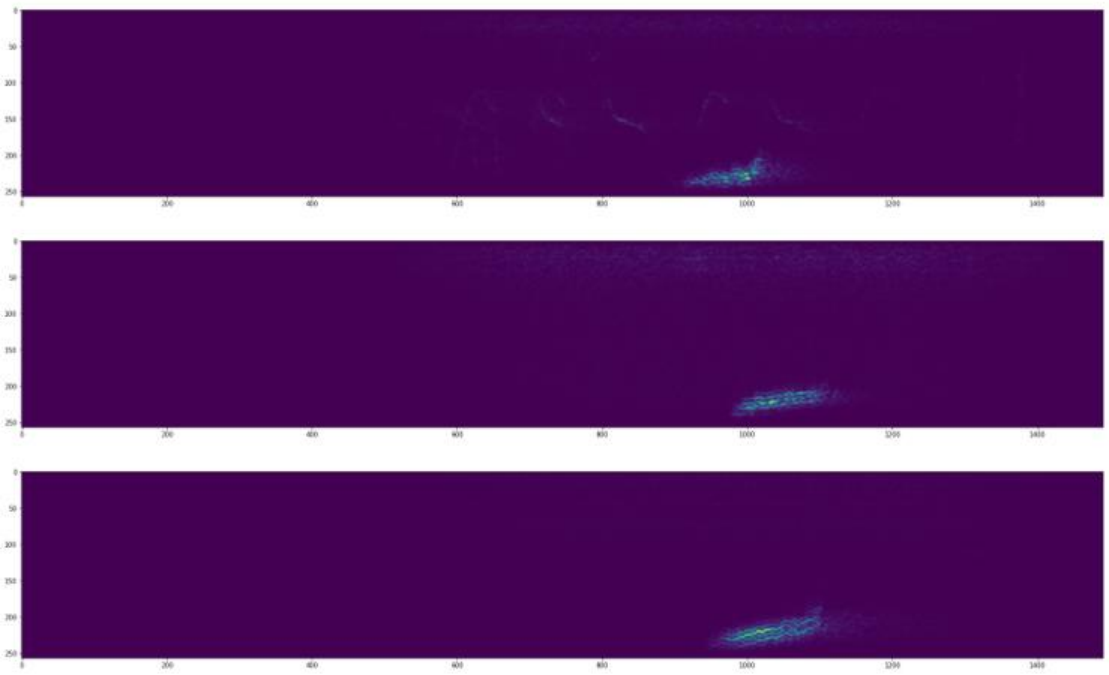

Figura 6-5. Ejemplos de espectrograma de Zorzal Alirrojo.

Zorzal Alirrojo		
Audios de evaluación	Aciertos	% Acierto
50	38	76%

Tabla 6-3. Resultados de experimentación con 200 audios de Zorzal Alirrojo.

6.3.4 Curruca Zarcerilla (*Curruca curruca*)

El canto de esta especie tiene un patrón más repetitivo que las anteriores, lo que la hace más fácilmente reconocible en la fase de evaluación, brindando una mejor precisión.

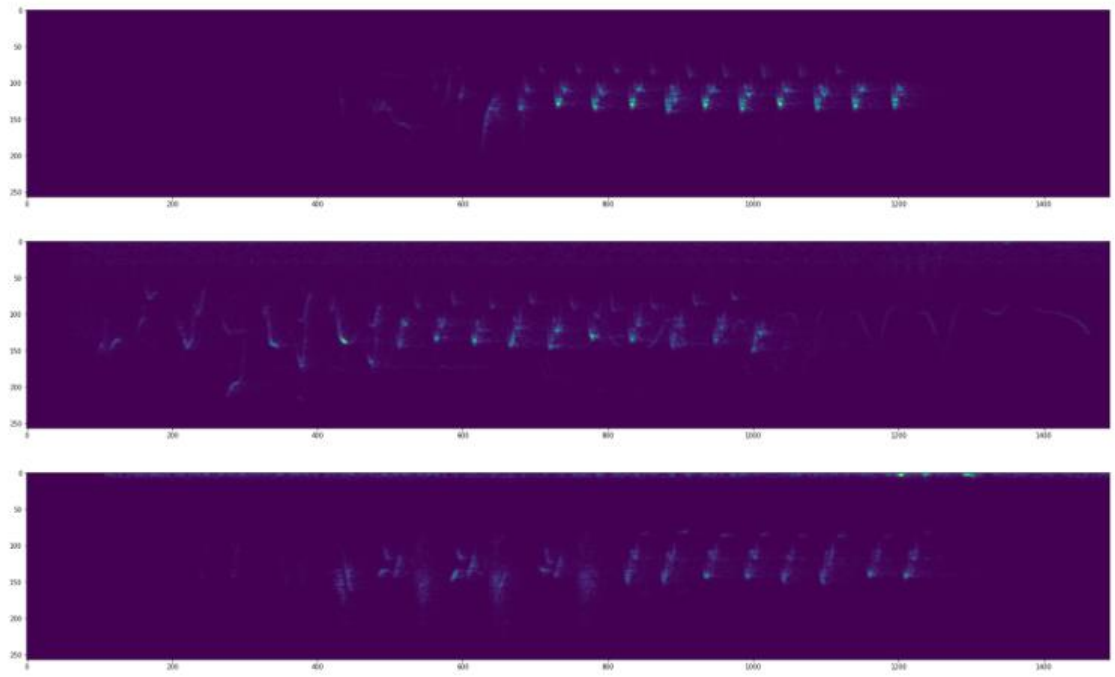

Figura 6-6. Ejemplos de espectrograma de Curruca Zarcerilla.

Curruca Zarcerilla		
Audios de evaluación	Aciertos	% Acierto
50	44	88%

Tabla 6-4. Resultados de experimentación con 200 audios de Zorzal Alirrojo.

6.3.5 Mosquitero Musical (*Phylloscopus trochilus*)

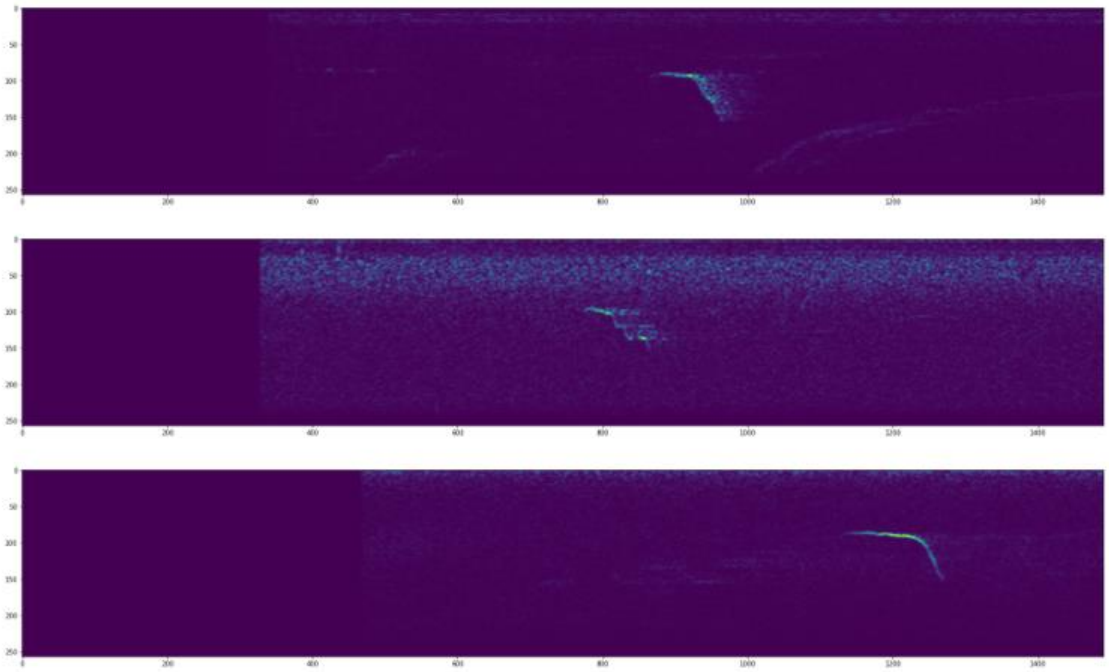

Figura 6-7. Ejemplos de espectrograma de Mosquitero musical.

Curruca Zarcerilla		
Audios de evaluación	Aciertos	% Acierto
50	40	80%

Tabla 6-5. Resultados de experimentación con 200 audios de la Curruca Zarcerilla.

7 CONCLUSIONES

7.1 Conclusión

En bioacústica computacional, como en otros campos, DL ha permitido un salto en el rendimiento de los sistemas automáticos. La bioacústica continuará beneficiándose de desarrollos más amplios en DL, incluidos los métodos adaptados del reconocimiento de imágenes, el habla y el audio en general. Continuará siendo impulsado por la disponibilidad de datos, desarrollos en hardware de audio y potencia de procesamiento, y las demandas de la contabilidad de la biodiversidad nacional e internacional.

En este trabajo hemos revisado la base teórica de la Inteligencia Artificial (ML y DL), investigado las metodologías más populares para reconocimiento de audio, investigado los distintos datasets públicos existentes de cantos de aves y una búsqueda de las especies que habitan en los humedales de Alicante. Por último, hemos puesto en práctica esta información, para desarrollar un algoritmo de Deep Learning capaz de detectar distintas especies de aves a través del canto.

La utilidad de esta herramienta resulta de gran ayuda para los expertos en cantos de aves ya que la mejor manera de preservar las especies en peligro es tener mapas de calidad de presencia y abundancia de las especies, lo que se puede conseguir detectando los diferentes cantos en un área específica. Además, es también útil para uso turístico, pues con el desarrollo de una aplicación móvil la gente podría encontrar la especie de ave que está escuchando en el momento, así como lo hacemos hoy en día con las canciones y Shazam.

El objetivo de este proyecto era conseguir un algoritmo que fuese capaz de diferenciar varias especies en un audio de entrada, pero debido a la limitación de almacenamiento y entrenamiento que ofrece Google Colab, se ha optado por demostrar el funcionamiento de Deep Learning con la clasificación binaria en cada especie por separado. Lo ideal sería entrenar al sistema con todas las especies y determinar qué especie se escucha en la grabación, de esta forma, podríamos representar una matriz de confusión, que representa con más detalles la precisión de la red, disminuyendo probablemente de una manera considerable el porcentaje de acierto que hemos obtenido por clasificación binaria.

Durante este proyecto hemos podido aprovechar los conocimientos adquiridos a lo largo del Grado, desde la parte de gestión de proyecto, hasta el procesamiento de digital de audio. La inteligencia artificial es un ámbito que hemos visto de manera teórica, pero no de forma experimental, por lo que este proyecto supone para mí la primera toma de contacto con Deep Learning.

7.2 Trabajos Futuros

Ya hemos mencionado la utilidad de este proyecto para la ecología y turismo, de modo que una de sus posibles aplicaciones podría ser el funcionamiento en una aplicación para dispositivos móviles. Con el programa creado, se ha podido demostrar que la red reconoce con alta precisión sus respectivas especies, por lo que el siguiente paso sería unificar las redes entrenadas de cada especie, y evaluar un audio para cada una de ellas, de modo que las especie cuyo valor máximo sea el mayor, será la especie de ave que está cantando. Es aquí cuando el porcentaje de precisión disminuiría, ya que es probable que confunda varias especies.

Además, otra meta sería incorporar muchas más grabaciones a la fase de entrenamiento, para así conseguir una precisión mucho mayor, pues 200 audios por especie no resultan suficientes para identificar las especies ante situaciones con demasiado ruido u otras aves cantando a la misma vez. Además, deberíamos tener en cuenta el número de epochs, ya que, si aumentamos el tamaño del dataset, la fase de entrenamiento debe de procurar una mayor precisión.

Por otro lado, también introducir nuevas especies, procurando abarcar aquellas cuyos cantos son similares para evitar ambigüedades. Dependiendo del uso principal que se contemple, se darán prioridades a aquellas en peligro de extinción o con un papel vital en la ecología, o a las más comunes si nuestro objetivo es el público turístico.

8 ANEXO

El código empleado para llevar a cabo este proyecto se encuentra en el siguiente enlace:

https://github.com/daricuba99/Reconocimiento-de-cantos-de-ave

9 BIBLIOGRAFÍA

[1] Emilio Guirado, Siham Tabik, Marga L, Alaraz-Segura D y Herrera F, «Automatic whale counting in satellite images with deep learning,» 2018. [En línea]. Disponible en: https://doi.org/10.1101/443671.

[2] Norouzzadeh M. S, Nguyen a, Kosmala M, Swanson A, Palmer M. S, Packer C y Clune J, «Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning,» 2018. [En línea]. Disponible en: https://doi.org/10.1073/pnas.1719367115.

[3] Shen D, Wu G y Suk H, «Deep Learning in Medical Image Analysis,» 2017. [En línea]. Disponible en: https://doi.org/10.1146/annurev-bioeng-071516-044442.

[4] Kaling U, Lang N, Hug C, Glessler A y Wegner J. D, «Defoliation estimation of forest trees from ground-level images,» [En línea]. Disponible en: https://doi.org/10.1101/441733.

[5] Ellis E. C, «Ecology in an anthropogenic biosphere,» 2015. [En línea]. Disponible en: https://doi.org/10.1890/14-2274.1.

[6] Salamon J, Bello J. P, Farnsworth A y Kelling S, «https://doi.org/10.1109/ICASSP.2017.7952134,» 2017. [En línea]. Disponible en: https://doi.org/10.1109/ICASSP.2017.7952134.

[7] Mac Aodha D, Gibb R, Barlow K. E, Browning E, Firman M, Freeman R, Harder B, Kinsey L, Mead G. R, Newson S. E, Pandourski I, Parsons S y Russ J, «Deep learning tools for bat acoustic signal detection,» 2018. [En línea]. Disponible en: https://doi.org/10.1371/journal.pcbi.1005995.

[8] David R, Geoffrey H y Ronald J, «Learning representations by back-propagating errors.,» Disponible en: https://doi.org/10.1038/323533a0, 1986.

[9] Luke T y Nitschke G, 20 Agosto 2017. [En línea]. Disponible en: https://arxiv.org/abs/1708.06020.

[10] Stowell D, 21 Marzo 2022. [En línea]. Disponible en: https://peerj.com/articles/13152/.

[11] Stowell D, «Automatic acoustic identification of individuals in multiple species: improving identification across recording conditions,» 2018. [En línea]. Disponible en: https://doi.org/10.1098/rsif.2018.0940.

[12] Shiu Y, Klinck H, Fleishman E, Liu X, Nosal E-M, Helbe T, Cholewiak D y Gillespie D, «Improve automatic detection of animal call sequences with temporal context,» 2021. [En línea]. Disponible en: https://doi.org/10.1098/rsif.2021.0297.

[13] Marchall J, Fabianek F y Aubry Y, «Software performance for the automated identification of bird vocalisations: the case of two closely related species,» 2021. [En línea]. Disponible en: https://doi.org/10.1080/09524622.2021.1945952.

[14] Romero-Mujalli D, Bergmann T, Zimmermann A y Scheumann M, «Utilizing DeepSqueak for automatic detection and classification of mammalian vocalizations: a case study on primate vocalizations,» 2021. [En línea]. Disponible en:

https://www.nature.com/articles/s41598-021-03941-1.

[15] S. Zsebök, «Automatic bird song and syllable segmentation with an open-source deep-learning object detection method – a case study in the Collared Flycatcher (Ficedula albicollis),» 2019. [En línea]. Disponible en: https://doi.org/10.2478/orhu-2019-0015.

[16] Coffey KR, Marx RG y Neumaier JF, «DeepSqueak: a deep learning-based system for detection and analysis of ultrasonic vocalizations,» 2019. [En línea]. Disponible en: http://dx.doi.org/10.1038/s41386-018-0303-6.

[17] Marchal J, Fabianek F y Aubry Y, «Software performance for the automated identification of bird vocalisations: the case of two closely related species,» 2021. [En línea]. Disponible en: https://doi.org/10.1080/09524622.2021.1945952.

[18] Knight EC, Hannah K, Foley G, Scott C, Brigham R y Bayne E, «Recommendations for acoustic recognizer performance assessment with application to five common automated signal recognition programs,» 2017. [En línea]. Disponible en: http://dx.doi.org/10.5751/ACE-01114-120214.

[19] Prince P, Hill A y Piña Covarrubias E, «Deploying Acoustic Detection Algorithms on Low-Cost, Open-Source Acoustic Sensors for Environmental Monitoring,» 2019. [En línea]. Disponible en: https://doi.org/10.3390/s19030553.

[20] A. Mesaros, A. Diment, B. Elizalde, T. Heittola, E. Vicent, B. Raj y T. Virtanen, «http://dx.doi.org/10.1109/TASLP.2019.2907016,» 2019. [En línea]. Disponible en: http://dx.doi.org/10.1109/TASLP.2019.2907016.

[21] Marcus G, 2 Junio 2018. [En línea]. Disponible en: http://arxiv.org/abs/1801.00631.

[22] Jesse F, Arnand T, Seth T, Nguyen A, Mohamed A, Sebastian W, Arjan K, Andreas W y Arniban M, «Habitat-Net: Segmentation of habitat images using deep learning,» 2018. [En línea]. Disponible en: https://doi. org/10.1101/483222.

[23] Knight EC, Hannah C, Foley J, Scott D, Brigham R y Bayne E, «Recommendations for acoustic recognizer performance as- sessment with application to five common automated signal recogni- tion programs. Avian Conservation and Ecology,» 2017. [En línea]. Disponible en: https://doi.org/10.5751/ACE-01114-120214.

[24] Sylvain C, Eric H y Nicolas L, «Applications for deep learning in ecology,» [En línea]. Disponible en: https://doi.org/10.1111/2041-210X.13256.

[25] Duedrik P y Jimmy Ba, «A Method for Stochastic Optimization,» 2014. [En línea]. Disponible en: https://doi.org/10.48550/arXiv.1412.6980.