UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Estudio de la biodiversidad en el Parc de lAlbufera mediante técnicas de aprendizaje profundo

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Nuño Pérez, David

Tutor/a: Piñero Sipán, María Gemma

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

PORTADA GENERADA AUTOMÁTICAMENTE EN EBRON

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

La monitorización medioambiental para el análisis de hábitats de especies ha requerido tradicionalmente de un alto grado de participación humana tanto en la adquisición de información como en la supervisión del procedimiento. En este Trabajo de Fin de Grado se detalla como el *Deep Learning* puede ayudar a automatizar el proceso, reduciendo así el número de horas dedicadas a esta tarea por los investigadores. Primero, se hará una introducción para tratar brevemente aspectos teóricos del *Deep Learning*, así como el contexto del Parque Natural de la Albufera. Posteriormente, se hablará de todo el preprocesado (bases de datos, recorte de audios, extracción de características…) hasta llegar a la Red Neuronal Convolucional que se ha usado para clasificar especies de pájaros. Para concluir, se mostrarán los resultados obtenidos y algunas reflexiones.

Resum

La monitorització mediambiental per a l'anàlisi d'hàbitats d'espècies tradicionalment ha requerit un alt grau de participació humana tant en l'adquisició d'informació com en la supervisió del procediment. En aquest Treball de Fi de Grau es detalla com el Deep Learning pot ajudar a automatitzar el procés i reduir així el nombre d'hores dedicades a aquesta tasca pels investigadors. En primer lloc, es farà una introducció per tractar breument aspectes teòrics del Deep Learning, així com el context del Parc Natural de l'Albufera. Posteriorment, es parlarà de tot el preprocessament (bases de dades, retallada d'àudios, extracció de característiques…) fins a arribar a la Xarxa Neuronal Convolucional que s'ha fet servir per classificar espècies d'ocells. Per concloure, es mostraran els resultats obtinguts i algunes reflexions.

Abstract

Environmental monitoring for the analysis of species habitats has traditionally required a high degree of human participation both in the acquisition of information and in the supervision of the procedure. This Final Degree Project details how Deep Learning can help automate the process, thus reducing the number of hours dedicated to this task by researchers. First, an introduction will be made to briefly discuss theoretical aspects of Deep Learning, as well as the context of the Albufera Natural Park. Subsequently, all the preprocessing (databases, audio trimming, feature extraction...) will be discussed until reaching the Convolutional Neural Network that has been used to classify bird species. To conclude, the results obtained and some reflections will be shown.

Índice

I

Capítulo 1. Introducción 1
1.1 Estado del arte: Parc de l'Albufera 2
1.2 Clasificadores basados en Machine Learning / Deep Learning 2
1.2.1 AlexNet - 2012 3
1.2.2 GoogLeNet – 2014 3
1.2.3 VGG – 2014 4
1.2.4 ResNet – 2015 4
1.3 Clasificadores y datasets previos usados en la clasificación de aves 5
1.3.1 DCASE 5
1.3.2 Kaggle 5
Capítulo 2. Base de datos de *Kaggle* 6
2.1 Clasificador de *Kaggle* 6
Capítulo 3. Base de datos de l'Albufera 8
Capítulo 4. Clasificador para especies de pájaros 9
4.1 Preprocesado de los audios 9
4.2 Extracción de características 9
4.2.1 Espectrograma Mel 10
4.2.2 MFCC 10
4.3 Red neuronal convolucional (CNN) 10
4.3.1 Arquitectura 10
4.3.2 Hiperparámetros 11
4.3.3 Baseline 12
4.3.4 Modelo 1 12
4.3.5 Modelo 2 12
4.3.6 Modelo 3 13
4.3.7 Modelo 4 13
Capítulo 5. Resultados 16
5.1 Base de datos de *Kaggle* 16
5.1.1 Baseline 16
5.1.2 Modelo 1 17
5.1.3 Modelo 2 18
5.1.4 Modelo 3 18

5.1.5 Modelo 4 19
5.2 Base de datos de l'Albufera 20
Capítulo 6. Conclusiones 24
Capítulo 7. Anexos 25
7.1 Listado de la base de datos de *Kaggle* 25
7.2 Listado de la base de datos de la Albufera 28
7.3 Código utilizado para el preprocesado 33
7.4 Código del clasificador 37
Capítulo 8. Bibliografía 41

Capítulo 1. Introducción

Antes de la llegada de la inteligencia artificial (IA) a nuestras vidas resultaba complicado dejar en manos de una máquina ciertas tareas o, sin ir más lejos, pensar que podría ser una ayuda tan grande. Hoy en día, intentar defender que la ayuda de la IA es prescindible resulta bastante complicado. En cuanto a las aplicaciones y soluciones que nos brinda la IA, se pueden destacar numerosos ejemplos:

- Asistentes de voz, como los muy populares Google Home o Amazon Echo.
- Publicidad en redes sociales, personalizada en base a un estudio del usuario por parte de una máquina.
- Información en tiempo real del tráfico por carretera, con el objetivo de optimizar la ruta que proponen las principales apps de navegación.
- Domótica.
- Ayuda al diagnóstico médico.
- Gestión de las redes móviles.

Evidentemente, la IA está muy presente en nuestras vidas, ya sea silenciosamente o de forma rotunda. Sin embargo, sus aplicaciones no son para nada limitadas. Entre muchos beneficios, uno de los más interesantes es la automatización del proceso. Y es que, aunque muchos estudios concluyen en que el cerebro es mucho más potente que cualquier superordenador, la capacidad de éstos para ejecutar procesos, comandos, rutinas, etc., de manera incansable, es mucho mayor.

Existen diversas técnicas para aprovechar toda esta potencia en términos de IA. En el caso de este TFG es el *Deep Learning* aplicado a la clasificación de imágenes. Esta técnica trata de emular la forma en la que aprende un ser humano. Su estructura se basa en la manera que tienen de conectarse las neuronas en nuestro cerebro, de ahí que reciba el nombre de redes neuronales. El siguiente dibujo corresponde a un ejemplo de una estructura de red neuronal simplificada:

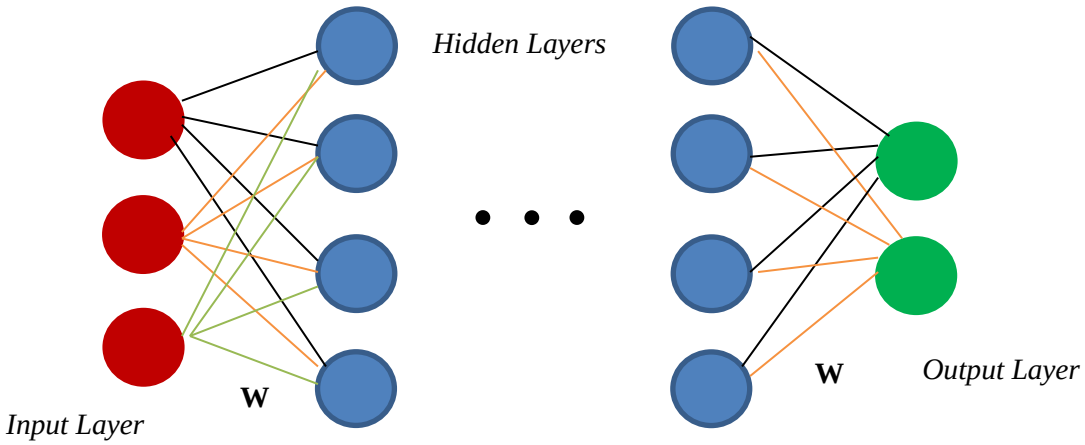

Figura 1. Ejemplo de red neuronal básica.

Como se puede observar, cada capa consta de un número determinado de “neuronas”, las cuales se conectan entre ellas. La capa de entrada recibe la información que queremos que aprenda a clasificar la red. En este caso, serían imágenes de un determinado tamaño. Tras pasar todas las capas ocultas, se llega a la capa de salida, donde se clasifica la entrada entre todos los posibles resultados. El mecanismo básico sería ese, sin embargo, para que haya aprendizaje falta algo. De forma iterativa, el mismo set de imágenes entra a la red durante un número de épocas. Tras cada época, se evalúa el resultado, y en función de ello se actualizan los pesos en base a una función

de pérdidas, de modo que el aprendizaje se basa en minimizar dicha función mediante el vector de pesos. A medida que esta función se haga menor, el acierto será mayor. Por ejemplo, una función de pérdidas muy conocida es el error cuadrático medio. Se trata de restar la predicción y el valor a predecir, elevando el resultado al cuadrado y promediando por todas las muestras. Su fórmula es:

$$\boldsymbol{ECM = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2} \quad \textbf{(1.1)}$$

La forma en que se relacionan los pesos con dicha función da lugar a un algoritmo de optimización. Por completar el ejemplo, este algoritmo podría ser el de descenso por gradiente, que tiene la siguiente forma:

$$\boldsymbol{w \leftarrow w - \eta \frac{\partial ECM}{\partial w}} \quad \textbf{(1.2)}$$

Donde η es el *learning rate*, uno de los hiperparámetros de la red de los cuales se hablará más adelante. De esta manera se conseguiría actualizar los pesos, dando lugar a un aprendizaje y emulando la estructura neuronal de los seres humanos.

Haciendo hincapié en el caso concreto de este TFG, la idea es clasificar audios de pájaros, permitiendo monitorizar y analizar la biodiversidad del *Parc de l'Albufera* de manera automatizada. Para ello, los audios han de convertirse a imágenes, llevando a cabo un preprocesado de los datos. Todo ello se tratará más adelante.

1.1 Estado del arte: Parc de l'Albufera

El Parque Natural de la Albufera (https://parquesnaturales.gva.es/es/web/pn-l-albufera), situado en Valencia, es un espacio natural protegido con una gran biodiversidad, en particular avícola. Su superficie es igual a 21.120 hectáreas. Se podrían relacionar más de 150 especies de pájaros con este parque (*Porzana, Acrocephalus, Locustella, Curruca…*), sin embargo, las más destacables son las siguientes:

- Pato colorado
- Cuchara común
- Ánade azulón
- Colonias de garzas
- Charrán común

Aparte de estas aves, también son destacables las aves acuáticas. Más de 350 especies habitan en l'Albufera, bien sea de manera habitual o para reproducirse. Sin embargo, el momento más importante de este parque es cuando comienza la época de cría, cuando determinadas especies pueden alcanzar hasta las 6000 parejas. Fuera de esta época, una de las especies que más parejas acumula de forma habitual es el charrán común, con unas 5000.

La popularidad del Parc de l'Albufera también tiene que ver con las especies en peligro de extinción que alberga, posibilitando así su vida. Entre ellas, se puede mencionar la cerceta pardilla (2 – 4 parejas/año), el chorlitejo patinegro o la canastera común.

Teniendo en cuenta la gran cantidad de especies y los ejemplares asociados, el uso de IA para el estudio de la biodiversidad facilita enormemente la tarea de especialistas, cumpliendo con un pilar fundamental de la misma mencionado anteriormente: automatizar el proceso.

1.2 Clasificadores basados en Machine Learning / Deep Learning

Es muy común las competiciones donde se evalúan diferentes modelos de redes neuronales. En concreto, los ganadores de ImageNet han ido sentando precedentes en cuanto a modelos muy

eficientes de redes neuronales convolucionales (CNN). Dado que la red que se utilizará también será una CNN, resulta interesante hacer este ejercicio de análisis de modelos exitosos.

1.2.1 AlexNet - 2012

Una de las novedades que incluyó fue el entrenamiento sobre GPU's, permitiendo entrenar modelos con imágenes de alta resolución. Su arquitectura se compone de 8 capas, donde aparte del uso de GPU's destacan dos novedades: ReLU como función de activación a la salida de las capas convolucionales y *dropout*, técnica que consiste en "apagar" neuronas en cada época, usando diferentes neuronas por iteración y forzándolas a ser más robustas a la hora de aprender. Por el contrario, aumenta el tiempo de entrenamiento del modelo. La función ReLU se define como 0 para valores negativos y el mismo valor de entrada cuando éste es mayor o igual a 0. Su mayor ventaja es que siempre es lineal, no hay zonas de saturación, evitando que el entrenamiento pueda estancarse.

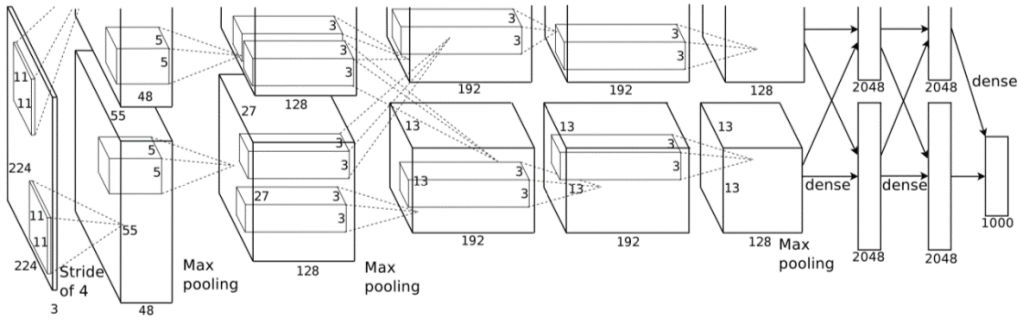

Figura 2. Arquitectura de AlexNet.

1.2.2 GoogLeNet – 2014

La arquitectura de GoogLeNet consta de 27 capas incluyendo las capas de *pooling*. Inicialmente, uno de los objetivos de esta red era ser lo más eficiente posible sin perder potencia en comparación con los modelos que le precedían. Una de las maneras en las que esto se consigue es reduciendo la imagen de entrada y quedándose con la información más relevante. Esto conlleva un preprocesado de los datos, en el que todos los píxeles no son igual de importantes, estableciendo un criterio selectivo.

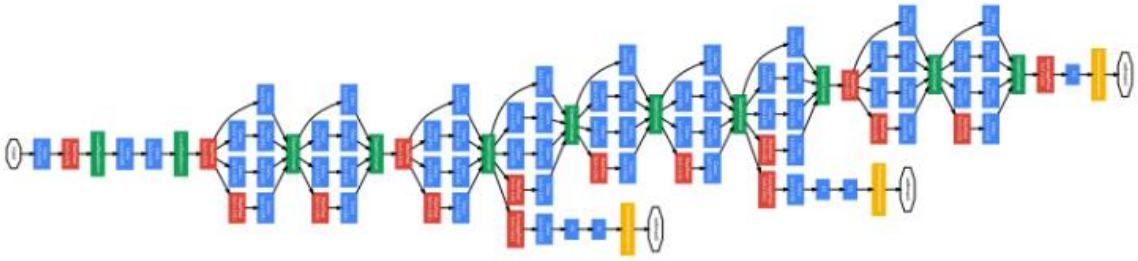

Figura 3. Arquitectura de GoogLeNet.

Otra curiosidad que incluye GoogLeNet es que no se calcula la predicción únicamente a la salida, sino también en dos puntos previos, como se observa en la imagen. Si solo se realizara a la salida, al actualizar los pesos mediante la función de pérdidas los valores que llegarían a las primeras capas serían muy pequeños, convirtiendo a esta red en una red imposible de entrenar.

1.2.3 VGG – 2014

Aunque esta red no fue ganadora del concurso ImageNet (GoogLeNet obtuvo mejores resultados ese año), ha sido una red con mucho éxito. Sentó un precedente en cuanto a patrones de diseño de las redes convolucionales. Las capas convolucionales no modifican el tamaño de las imágenes, solo lo hacen las capas de *pooling*, doblando el número de filtros cada vez que se usa una de estas.

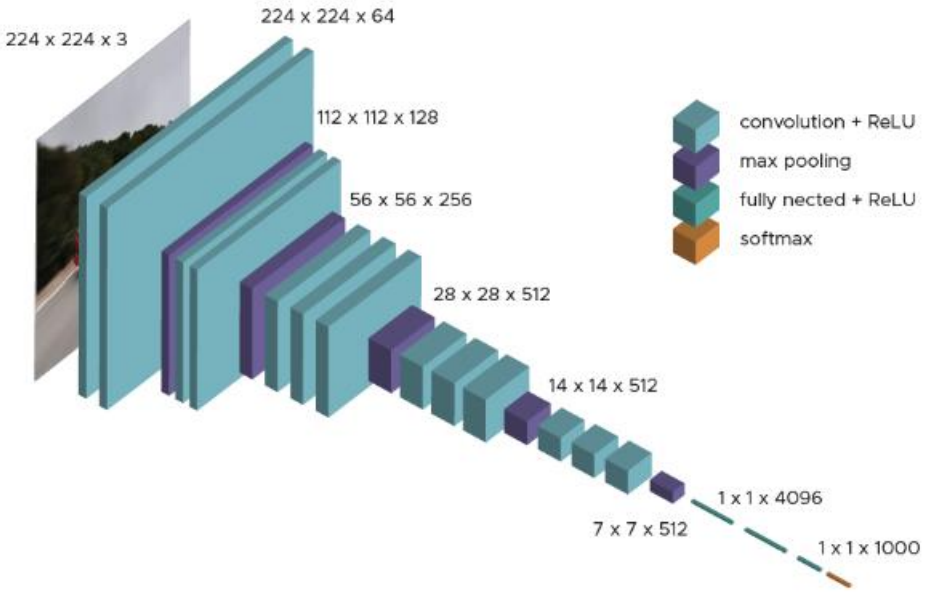

Figura 4. Arquitectura de VGG.

1.2.4 ResNet – 2015

No es una red como tal, sino una familia de redes. Uno de los detalles más destacables es el uso de *skip connections*. En GoogLeNet se ha visto como se tuvo que predecir en etapas intermedias para actualizar los pesos de manera correcta. Estas conexiones permiten dos cosas: por una parte, predecir solo en la última capa y actualizar los primeros pesos sin necesidad de pasar por todas las capas que hay entre medias, y por otra parte, dotar a la red de autonomía para decidir si saltarse alguna capa en el proceso de aprendizaje, si es que su uso no da buenos resultados.

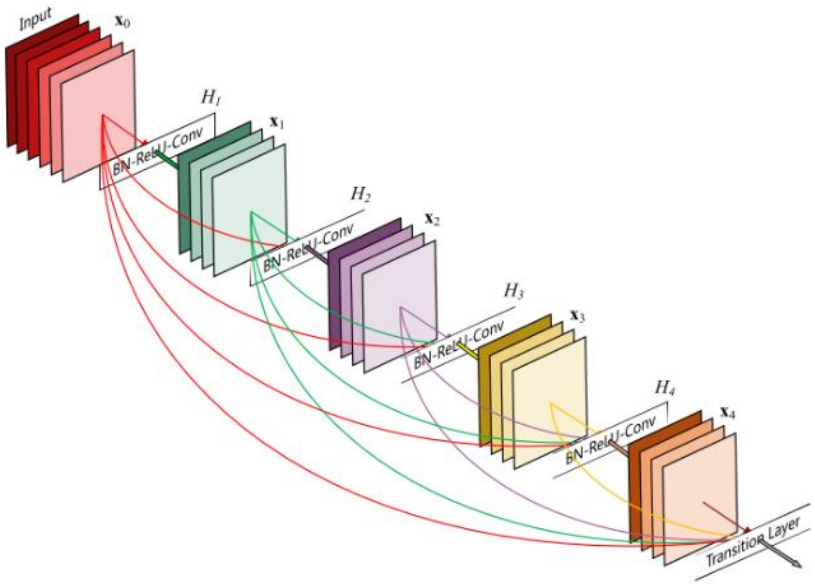

Figura 5. Arquitectura de ResNet.

1.3 Clasificadores y datasets previos usados en la clasificación de aves

1.3.1 DCASE

En cuanto a clasificadores cuya función sea detectar audio, un evento anual muy importante es el DCASE (*Detection and Classification of Acoustic Scenes and Events*). En esta competición se proporciona un *dataset* tanto de entrenamiento como de test a todos los participantes, y se estructura el desafío en varias etapas. Finalmente, el que mejores resultados obtenga gana.

Más concretamente, el que tuvo lugar en 2018 tenía como objetivo la detección de sonidos de pájaros. Se proporcionaron tres *datasets* para entrenar y tres para evaluar. El objetivo de este apartado era, dado un pequeño audio, determinar con un 1 o un 0 la presencia o ausencia de cualquier sonido de pájaro. El código de todos los participantes no siempre esta accesible, sin embargo, algunos sí. El tercer clasificado estructuró su clasificador en 27 capas. Por desgracia, esta tarea no se asemeja al 100% al caso que se trata en este TFG, que es determinar en base a un audio de entrada de qué especie se trata.

1.3.2 Kaggle

Kaggle es una página muy interesante para descargar *datasets* previamente creados por otras personas. De aquí se ha obtenido la base de datos con la cual se han hecho las pruebas, dado que era necesario tener audios con los que trabajar a la espera de tener audios de l'Albufera. Junto a dicha base de datos se encontraba accesible el código de un clasificador que sí pretendía clasificar entre todas las especies posibles. Por tanto, ha sido de gran utilidad a la hora de basarse en un ejemplo funcional. Esta red convolucional se construye a partir de bloques de capas convolucional-*pooling-dropout*, otra capa de *pooling* y una capa densa. En el punto 4.3 se tratará con más detalle.

Capítulo 2. Base de datos de *Kaggle*

Antes de pensar en una base de datos grabada en la Albufera, es recomendable comenzar con antelación con alguna de prueba. Por eso, encontrar una en *kaggle* es un buen punto de partida, ya que a partir de ella se puede avanzar mucho el trabajo y posteriormente extrapolar el código y todo el preprocesado.

En este sentido, se ha escogido la base de datos del repositorio de *kaggle* llamada "*Avian Vocalizations from CA & NV, USA*" (accesible en la URL: https://www.kaggle.com/datasets/samhiatt/xenocanto-avian-vocalizations-canv-usa), la cual se compone de 2730 archivos *mp3* entre los que se encuentran grabadas un total de 91 especies. Todos los audios provienen de *xeno-canto*, y en concreto fueron grabados en Califonia y en Nevada, EEUU. Hay un total de 30 muestras por especie, aunque todas ellas no tienen la misma duración, volumen, claridad o regularidad, refiriéndose con regularidad a que la especie en cuestión se escuche bien durante todo el tiempo grabado. Todo ello hace que sea necesario un preprocesado, el cual puede ser muy sencillo o complejo, pero hay que procurar que la entrada a la red convolucional sea similar. Como mínimo, es necesario que la entrada sea del mismo tamaño, lo que conlleva audios de igual duración y mismas características espectrales.

2.1 Clasificador de *Kaggle*

La manera de tratar esta información en una de las soluciones asociada a este *dataset* no era muy compleja, aun así, resultada efectiva. Dado que el objetivo era el mismo que tiene este TFG, la solución no es tan compleja como pueden ser las del evento *DCASE* y los resultados son aceptables, invita a pensar que con algún pequeño cambio se puede obtener un resultado satisfactorio. En este caso la *accuracy* de la que estamos hablando es de 0.42. No llega al 50% de precisión como podemos ver en la imagen siguiente donde se muestra la curva de aprendizaje:

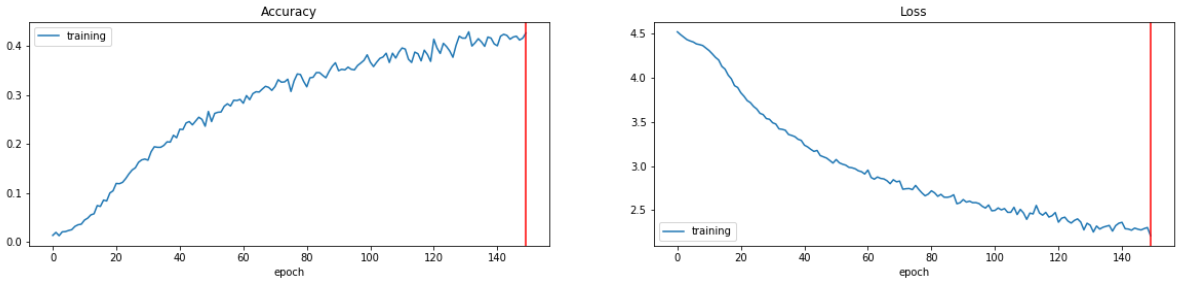

Figura 6. Curva de aprendizaje del clasificador de *Kaggle*.

A simple vista, el número de épocas es insuficiente. La curva aun tiene mucho margen de mejora y se puede pensar que un simple aumento en las épocas de entrenamiento conseguiría elevar la *accuracy* por encima del 0.5 sin problemas. En la siguiente tabla se recogen los hiperparámetros de la red convolucional que pueden modificarse para mejorar la *accuracy*:

Número de épocas	*Learning rate*	*Batch size*	*Dropout*
150	0.001	64	0.2

Tabla 1. Hiperparámetros del clasificador de *Kaggle*.

Para saber cómo se deben modificar, es de vital importancia conocer el papel que desempeñan, por eso se van a analizar a continuación.

El *learning rate* ya se ha visto en la ecuación (1.2). Su valor permite modificar el ritmo al que los pesos se actualizan. Si suponemos el valor inicial del *learning rate* demasiado alto, los pesos se actualizarán con valores demasiado dispares, y nunca llegará al mínimo global, si es que existe. Es decir, se quedaría estancado. Sin embargo, si le damos al *learning rate* un valor pequeño, sí que podrá llegar al mínimo global. Como conclusión, tenemos 2 opciones: utilizar un *learning rate* bajo que garantice llegar al mínimo global (sin estancarse tampoco en un mínimo local, no debe ser excesivamente bajo) o utilizar un *learning rate* variable. Esto es, empezar con un valor alto que nos acerque rápidamente al mínimo para posteriormente continuar con un valor pequeño, que nos sirva de ajuste fino. En cualquier caso, 0.001 es un buen punto de partida que, si da buenos resultados, puede mantenerse.

El tamaño del *batch* es el número de ejemplos que se introducen en la red para que entrene de una. Se plantean dos escenarios opuestos: si el número es pequeño, la red entrenará a mayor velocidad puesto que almacena pocos datos en memoria, pero aumenta la posibilidad de que no aprenda características/detalles significativos en la predicción. Si, por el contrario, el número es grande, es más probable que tenga en cuenta los detalles más importantes a la hora de aprender, pero entrenará más lento. Por lo tanto, es un compromiso, y en este sentido 64 es un buen valor de partida (suelen ser potencias de 2).

El concepto de *dropout* ya se ha tratado en el punto 1.2.1, a la hora de comentar la arquitectura de AlexNet. “Apagar” diferentes neuronas por cada época las obligaba a ser más robustas, ahora bien, apagar más de la cuenta puede hacer que éstas no aprendan correctamente. De nuevo se trata de un compromiso, en el cual llegar hasta 0.4 – 0.5 (la mitad de las neuronas “apagadas”) sería razonable.

Capítulo 3. Base de datos de l'Albufera

El objetivo de este trabajo es clasificar audios en base a las especies que se han entrenado, concretamente especies grabadas en el Parc de l'Albufera. En particular, este TFG se engloba en el proyecto de investigación DAPHNE - *Deep-learning Analysis and cyber-PHysical systems applied to biodiversity in urban and Natural Environments* de la Universidad de Sevilla en colaboración con la UPV y cuyas actuaciones se desarrollan en el Parque Nacional de Doñana y el Parc de l'Albufera. Dicho esto, para llevar a cabo las grabaciones son necesarios unos micrófonos, llamados nodos, proporcionados por los científicos de Sevilla. La forma de proceder, resumiendo, sería la siguiente: colocar los nodos correctamente distribuidos alrededor del parque, dejarlos grabando y como último paso montar la base de datos extrayendo los sonidos de pájaros de las grabaciones.

A día de hoy, todavía quedan algunos nodos por colocar en l'Albufera, por lo tanto, no es posible realizar el estudio completo, ya que no estaría disponible la base de datos. En su defecto, se ha construido una a mano, recopilando audios de la página web de *xeno-canto* siguiendo la lista de especies que los biólogos han considerado de interés. Esta base de datos se compone de 174 especies. Para cada especie se ha seleccionado aproximadamente 1 minuto y medio de duración, intentando elegir varios audios distintos aportando así diversidad. Tras este ejercicio, el tamaño total de la base de datos es de 523 MB repartidos en 924 audios *mp3*. Aunque es preferible entrenar sobre la base de datos encontrada en *kaggle*, la cual se sabe que es útil, y dejar esta para encontrar patrones de cada especie, también se entrenará. Para ello, se escogerán los hiperparámetros del modelo que mejor *accuracy* haya logrado y se creará un nuevo modelo, específico, que reconozca especies de interés que habitan en l'Albufera. No se podría usar el mejor modelo entrenado de la base de datos de kaggle ya que no coinciden las especies, por lo que no sabría reconocer especies con las que no ha sido entrenada. En cuanto a la base de datos con audios grabados en el propio parque, cuando se definan los patrones previamente mencionados, se simplificará la tarea de determinar qué especies se están escuchando en las grabaciones de los nodos, y así poder construir una base de datos de interés real, una base de datos de l'Albufera.

Un detalle que puede ser interesante destacar es que, a medida que se han ido recopilando los audios de *xeno-canto*, llegó un punto en el que resultaba difícil distinguir las especies ya que muchas sonaban casi iguales. Suponiendo que esto sucede también en la base de datos de *kaggle* ya que, al fin y al cabo, son pájaros, podrían esperarse resultados algo peores con este base de datos que prácticamente dobla el número de especies. Un *input* pasará de tener 91 opciones de ser clasificado a 174, lo cual complica el proceso, sobre todo cuando se tratan de audios parecidos.

Capítulo 4. Clasificador para especies de pájaros

Como ya se ha mencionado en el capítulo 3, el clasificador que se usará para este estudio será el mismo que se encontró como solución asociada a la base de datos de *kaggle*, introduciendo algún que otro detalle a la hora de preprocesar la información en crudo (los audios de pájaros extraídos de *xeno-canto*) y modificando los hiperparámetros hasta encontrar la mejor solución. Así mismo, la base de datos que se usará también será la de *kaggle*. La arquitectura de la red convolucional no cambiará, puesto que se entiende que da buenos resultados teniendo en cuenta que se prevé margen de mejora tanto en el preprocesado como en la elección de los hiperparámetros. Además, cabe mencionar que la red debería ser considerablemente compleja para clasificar con alta *accuracy* un número de especies tan grande (91 especies) y con tanta variabilidad. Dicho esto, se espera alcanzar un mínimo de 0.65 de *accuracy*. A continuación, se explicará los cambios introducidos en las dos líneas de trabajo que se viene comentando en este capítulo.

4.1 Preprocesado de los audios

Lo primero que llama la atención cuando se analiza el clasificador que se encontró como solución es el hecho de que los audios no están recortados. Es decir, entran al clasificador sin ningún tipo de criterio y es dentro de él cuando se decide el tamaño que tomarán. En este caso, se tenían tanto los espectrogramas Mel como los coeficientes MFCC precargados. Una vez se llamaba a la clase para generar los datos de entrada a la red convolucional era cuando se determinaba el tamaño de los mismos. El tamaño de entrada deseado era de (128,128) para los espectrogramas Mel y (20,128) para los MFCC. Esto se corresponde con aproximadamente 3.15 segundos de audio. Para conseguirlo, los audios que fueran más cortos se rellenaban con ceros, y de los más largos se elegía un trozo aleatorio de 3.15 segundos contenido en la totalidad de la grabación. La solución es válida, pero se pierden muchas muestras por el camino, además de que no se asegura que los trozos elegidos dentro de los audios más largos sean buenos: pueden tener mucho silencio o simplemente no ser un claro ejemplo de cómo suena esa especie.

La solución que se ha llevado a cabo es recortar todos los audios previamente al proceso de extracción de características. Así se consigue aumentar considerablemente el volumen de muestras del *dataset* y explotar de manera más óptima todo su potencial. Se pasa de 2730 muestras a 22409 muestras, es decir, un aumento del 820%, multiplicando por 8 la cantidad de *inputs*.

4.2 Extracción de características

Al tratarse de una red neuronal convolucional (CNN), es preciso que la entrada sean imágenes. La posibilidad, en cuanto a características elegibles, es bastante amplia. Sin embargo, se ha seguido con la misma idea que el ejemplo dado, es decir, combinar espectrograma Mel y MFCC. Solo en caso de no obtener la mejora en los resultados esperados se pensaría modificar el *input* de la CNN.

La manera de combinar espectrograma Mel y MFCC es situando los coeficientes MFCC en las frecuencias más bajas. Para ello, en la clase que genera los datos para la CNN se especifica que la imagen que devuelve sea de tamaño (128,128), donde el primer valor corresponde a las características y el segundo a las muestras temporales. Por tanto, las 20 primeras características se sustituyen por los MFCC. Con el propósito de aclarar esta idea, se muestra un ejemplo de lo que sería una imagen combinando espectrograma Mel y MFCC, la cual entraría como *input* a la CNN:

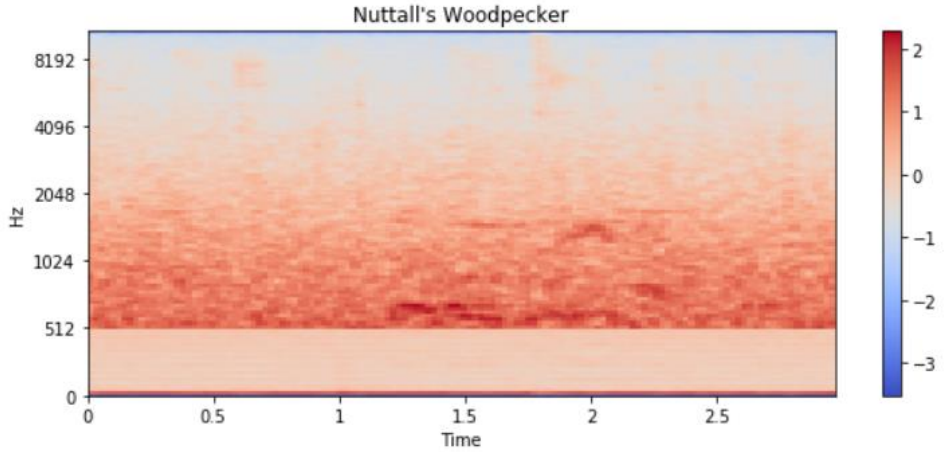

Figura 7. Ejemplo de *input* a la CNN (espectrograma Mel + MFCC)

Como se puede observar, las frecuencias más bajas corresponden con los MFCC y las altas con el espectrograma Mel de la trama en cuestión.

4.2.1 Espectrograma Mel

Partiendo de los audios recortados a 3.15 segundos aproximadamente, se generan los respectivos 22409 espectrogramas Mel, los cuales se almacenan en una carpeta aparte. Para llevar a cabo este proceso, se ha utilizado la librería *librosa* de Python, en concreto *librosa.feature.melspectrogram*. Mediante un bucle *for* que leyera todos los archivos de la carpeta donde se almacenaron previamente los audios recortados, se obtenía el nombre del archivo, y con ello se creaba el espectrograma Mel. Posteriormente se exportaba como *.dat* a la carpeta deseada.

4.2.2 MFCC

El procedimiento de extracción de los MFCC es básicamente el mismo que para los espectrogramas Mel, obteniéndolos en el mismo bucle *for*. La librería usada es también *librosa*, concretamente *librosa.feature.mfcc*. Tras la extracción, de nuevo se exporta como *.dat* a la carpeta deseada, en este caso una diferente. En este punto se tienen ya los datos que usará la clase para generar los *inputs*, a falta de un detalle: escalar.

Para escalar (o normalizar) las características extraídas se usa la clase *StandardScaler* de la librería *sklearn*. Se crean dos objetos, *mfcc_scaler* y *mel_scaler*, que se van configurando a medida que reciben espectrogramas Mel o MFCC, todo ello en un bucle *for* y haciendo uso del método *partial_fit*. Estos objetos se usan en la clase generadora de *inputs* para la CNN: cuando se carga una *feature* se escala, restando le media y dividiendo por la desviación típica.

4.3 Red neuronal convolucional (CNN)

4.3.1 Arquitectura

La arquitectura de la CNN se ha introducido en el apartado 1.3.2. Consta de tres bloques de capas convolucional-*max pooling*-*dropout*, otra capa de *average pooling* y una capa densa. En total, 11 capas con las que se obtiene una *accuracy* de aproximadamente 0.6. En el siguiente dibujo se muestra la arquitectura de la CNN:

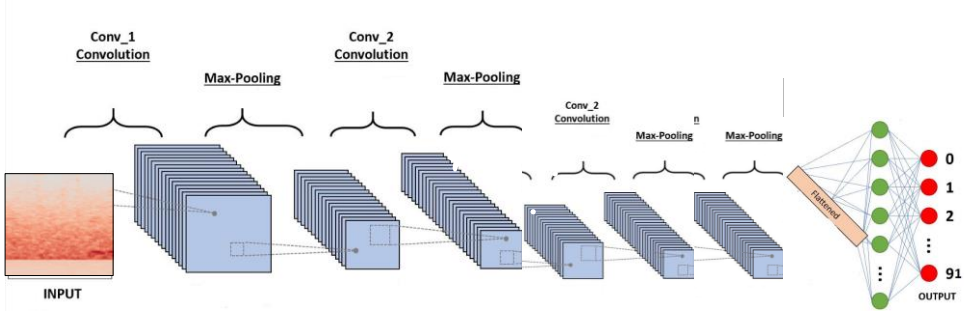

Figura 8. Arquitectura CNN *kaggle*

El número de filtros que aparece en la figura no coincide con los que hay en realidad. Habría 64 filtros de 3x3 en cada capa, tanto convolucional como de *pooling*. Además, la capa de *dropout* que hay tras cada *max pooling* se entiende implícita en esta última para simplificar el esquema.

La imagen de entrada tiene siempre un tamaño de 128x128, y las únicas capas que reducen el tamaño son las de *pooling*. Según la documentación de Keras, tras cada *max pooling* el tamaño reduce de la siguiente manera:

$$\boldsymbol{output\ shape} = \frac{\boldsymbol{input\ shape - pool\ size}}{\boldsymbol{strides}} + \mathbf{1} \qquad \textbf{(4.1)}$$

Al tener *pool_size = 3* y *strides = 1*, las dimensiones son: 126x126 tras el primer *max pooling*, 124x124 tras el segundo y 122x122 tras el tercero. La última capa de *pooling* se llama *global average pooling*, y actúa como una capa *flatten*, la diferencia es que aplica *average pooling* en las dimensiones espaciales hasta que cada dimensión espacial es igual a uno. En el caso de la típica capa *flatten*, toma un tensor de cualquier tamaño y lo transforma en un tensor de una dimensión. Aunque el resultado es el mismo, no hacen lo mismo para llegar a él.

4.3.2 Hiperparámetros

Como se ha visto, variar ciertos hiperparámetros con sentido puede suponer grandes cambios. En los siguientes puntos se tratará el estudio realizado con los diferentes hiperparámetros y el resultado obtenido en el proceso de entrenamiento, así como el porqué de haber modificado unos u otros.

A continuación, se muestra la tabla que recoge los hiperparámetros asignados a cada modelo:

	BASELINE	MODELO 1	MODELO 2	MODELO 3	MODELO 4
Épocas	1000	1000	1000	1000	1000
Batch size	64	128	64	128	256
Learning rate	0.001	0.001	0.001	0.001	0.001
Dropout	0.2	0.2	0.4	0.4	0.2

Tabla 2. Hiperparámetros de cada modelo.

4.3.3 Baseline

Como aparece en la tabla 2, el modelo base o inicial parte de 1000 épocas, 64 de *batch size*, 0.001 de *learning rate* y 0.2 de *dropout*. El hecho de elegir un número de épocas significativamente más alto que las 150 que usaba este clasificador es para asegurar que la curva de aprendizaje se estanca y que no se conseguiría mayor *accuracy* con unas pocas épocas más.

Tras entrenar con los 22409 audios de 3.15 segundos aproximadamente, se obtiene una *accuracy* de **0.652.** Es decir, los resultados que se esperaban ya se cumplen con el primer modelo que se prueba: superar una *accuracy* de 0.6 mediante el troceado de los audios más grandes de 3.15 segundos y el aumento de épocas.

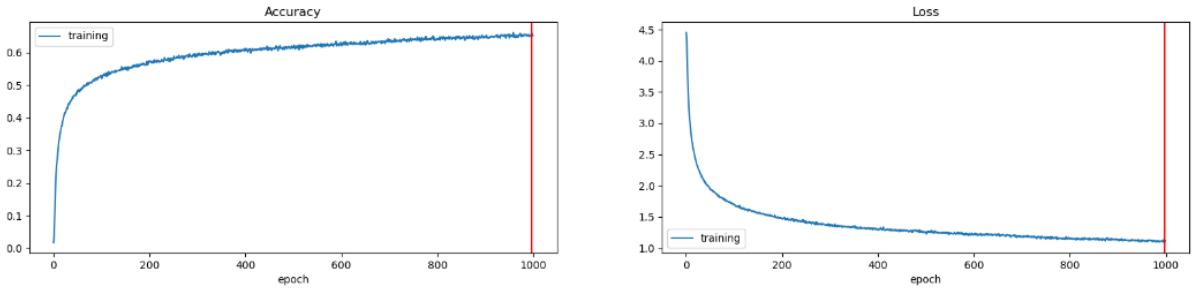

Figura 9. Curva de aprendizaje del modelo *baseline*

4.3.4 Modelo 1

Como aparece en la tabla 2, el modelo 1 parte de 1000 épocas, 128 de *batch size*, 0.001 de *learning rate* y 0.2 de *dropout*. Ya que los resultados han sido buenos con el modelo base, la idea es introducir un pequeño cambio. Doblar el tamaño del *batch* se sabe que hará que la red tenga en cuenta más características y/o detalles de las imágenes recibidas.

Tras el proceso de entrenamiento, la *accuracy* obtenida es de **0.7166**. De nuevo, los resultados son satisfactorios y ya se alcanza un valor bastante alto teniendo en cuenta que el número de especies no es pequeño, y que seguramente sea difícil distinguir algunos pájaros que suenen parecido.

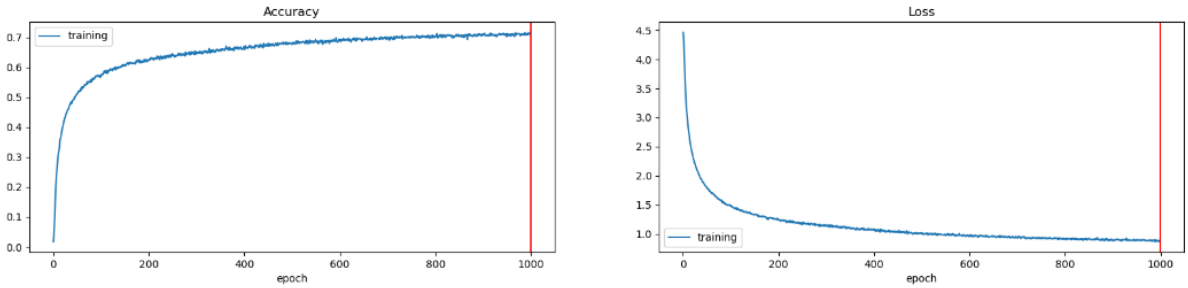

Figura 10. Curva de aprendizaje del modelo 1

4.3.5 Modelo 2

Como aparece en la tabla 2, el modelo 2 parte de 1000 épocas, 64 de *batch size*, 0.001 de *learning rate* y 0.4 de *dropout*. La idea en este modelo y el siguiente es probar a aumentar el *dropout*. Para ello el modelo 2 replicará al *baseline* y el modelo 3 al modelo 1, ambos modificando el *dropout.*

Tras el entrenamiento, la *accuracy* obtenida es de **0.523**. Esta modificación no ha supuesto ninguna mejora, sino al contrario.

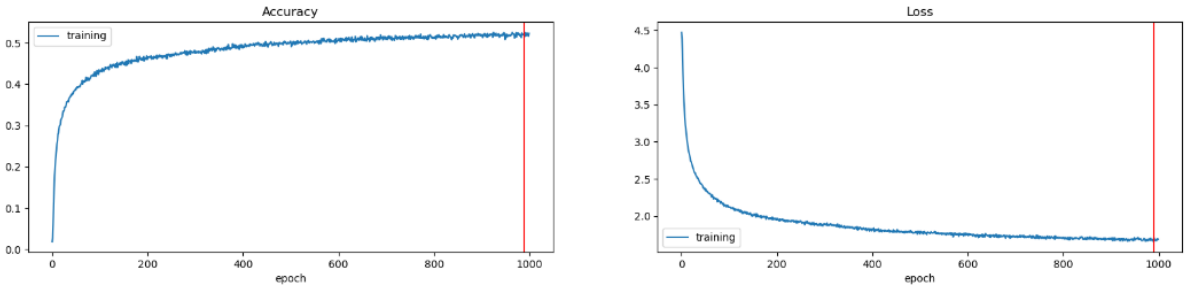

Figura 11. Curva de aprendizaje del modelo 2

4.3.6 Modelo 3

Como aparece en la tabla 2, el modelo 3 parte de 1000 épocas, 128 de *batch size*, 0.001 de *learning rate* y 0.4 de *dropout*. Aunque la prueba anterior sobre el *dropout* no ha salido bien, es interesante comprobar si con un tamaño de *batch* diferente sigue ocurriendo el mismo efecto.

Una vez entrenado el modelo, se obtiene una *accuracy* de **0.53**. Se confirma que en esta CNN los aumentos de *dropout* no suponen ninguna mejora.

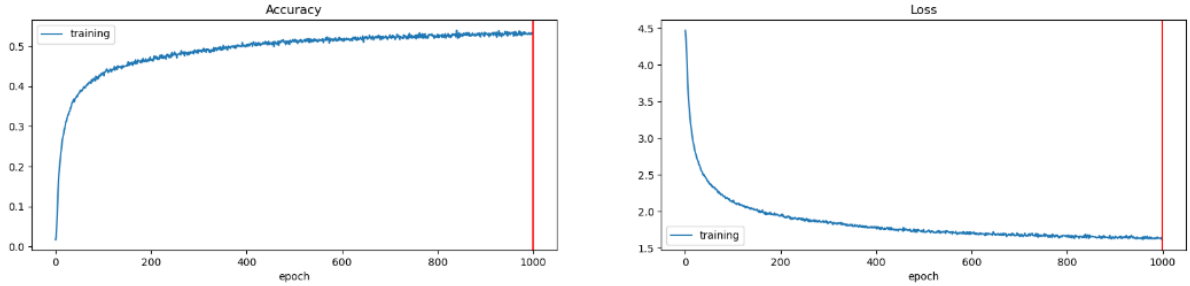

Figura 12. Curva de aprendizaje del modelo 3

4.3.7 Modelo 4

Vista la experiencia con el resto de los modelos, las conclusiones son claras: aumentar el *batch* mejora y aumentar el *dropout* empeora. Siguiendo esta regla, se ha querido hacer un último intento con estos hiperparámetros: 1000 épocas, 256 de *batch size*, 0.001 de *learning rate* y 0.2 de *dropout*.

Finalmente, el resultado del último modelo probado es de **0.7124**, muy similar al modelo 1. En resumen, tenemos dos modelos con bastante buena *accuracy*, el 1 y el 4, los cuales se han conseguido gracias a un aumento significativo del *set* de entrenamiento, otro aumento considerable de las épocas y un tamaño del *batch* mayor.

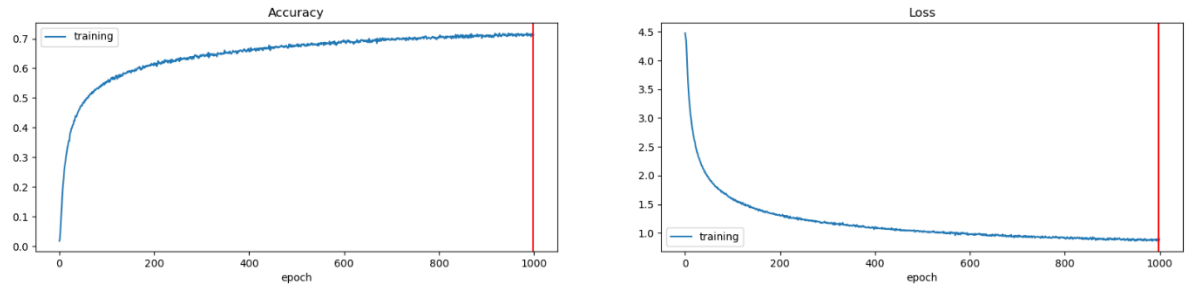

Figura 13. Curva de aprendizaje del modelo 4

En el punto 2.1, se trataba el concepto de *learning rate* variable. Aunque no se haya usado en ningún modelo, sí que se ha probado. El motivo que lleva a no usarlo es que no hay convergencia, el optimizador se pierde en mínimos locales y nunca llega a converger. Se puede observar gráficamente prestando atención al *log* durante el entrenamiento:

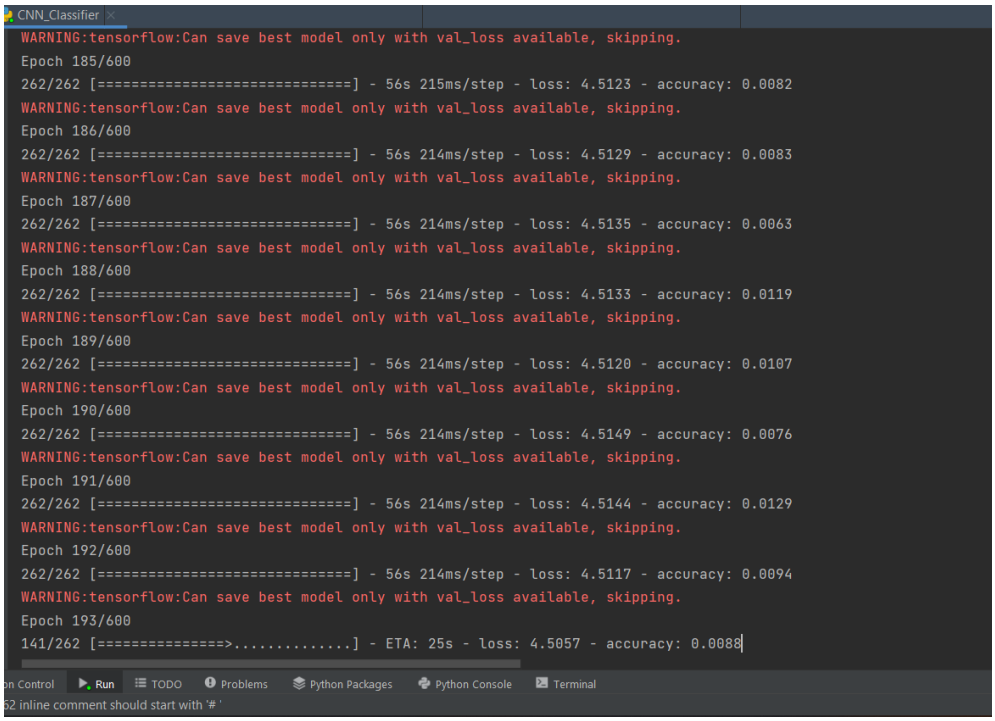

Figura 14. No convergencia de un modelo con *learning rate* variable

Es fácil ver que no hay convergencia, ya que tras casi 200 épocas ni siquiera se alcanzó un 0.1 de *accuracy*, mientras otros modelos a esas alturas pasaban el 0.5. A continuación se puede ver la curva de aprendizaje del modelo 1 con *learning rate* variable y configurado a 600 épocas:

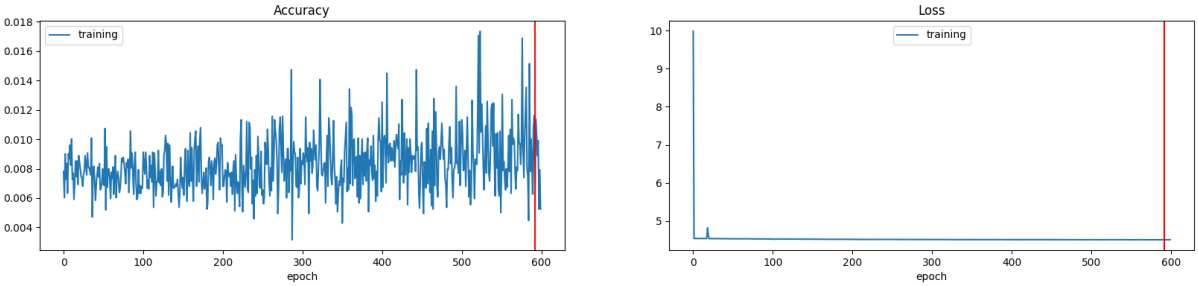

Figura 15. No convergencia del modelo 1 con *learning rate* variable

Este resultado puede deberse a que el *learning rate* inicial es demasiado alto. Sin embargo, se probó a empezar desde 0.1, 0.01 y 0.001 obteniendo siempre este resultado. Por algún motivo, la CNN en cuestión tiene mejor rendimiento cuando este hiperparámetro es fijo.

A modo de resumen, se recogen los resultados del entrenamiento en la siguiente tabla:

	BASELINE	MODELO 1	MODELO 2	MODELO 3	MODELO 4
Accuracy	0,652	0,7166	0,523	0,53	0,7124

Tabla 3. Resultados del entrenamiento.

Importante: todos los modelos han sido entrenados con la base de datos de *kaggle*. La idea es replicar la estructura que mejor resultado ha dado (modelo 1) y entrenar con la base de datos de la Albufera, obteniendo un modelo que clasifique pájaros de interés en la Albufera. Esto se tratará en el punto 5.2.

Capítulo 5. Resultados

Tras el entrenamiento, hay que pasarle a nuestro clasificador el *set* de prueba para que lleve a cabo su propósito, clasificar los audios en una especie u otra. Este *set* contiene un tercio del total de audios de la base de datos en cuestión: 5602 para el clasificador de *kaggle* y 1408 para el clasificador de la Albufera. De todos los modelos se ha obtenido la matriz de confusión, así como la *accuracy*.

5.1 Base de datos de *Kaggle*

5.1.1 Baseline

El modelo base tenia una *accuracy* de 0.652, por tanto, no se espera un resultado mejor a la hora de realizar el test, ya que son ejemplos que no ha visto nunca. El resultado obtenido es una *accuracy* de **0.61**, bastante bueno. A continuación, se muestra la matriz de confusión:

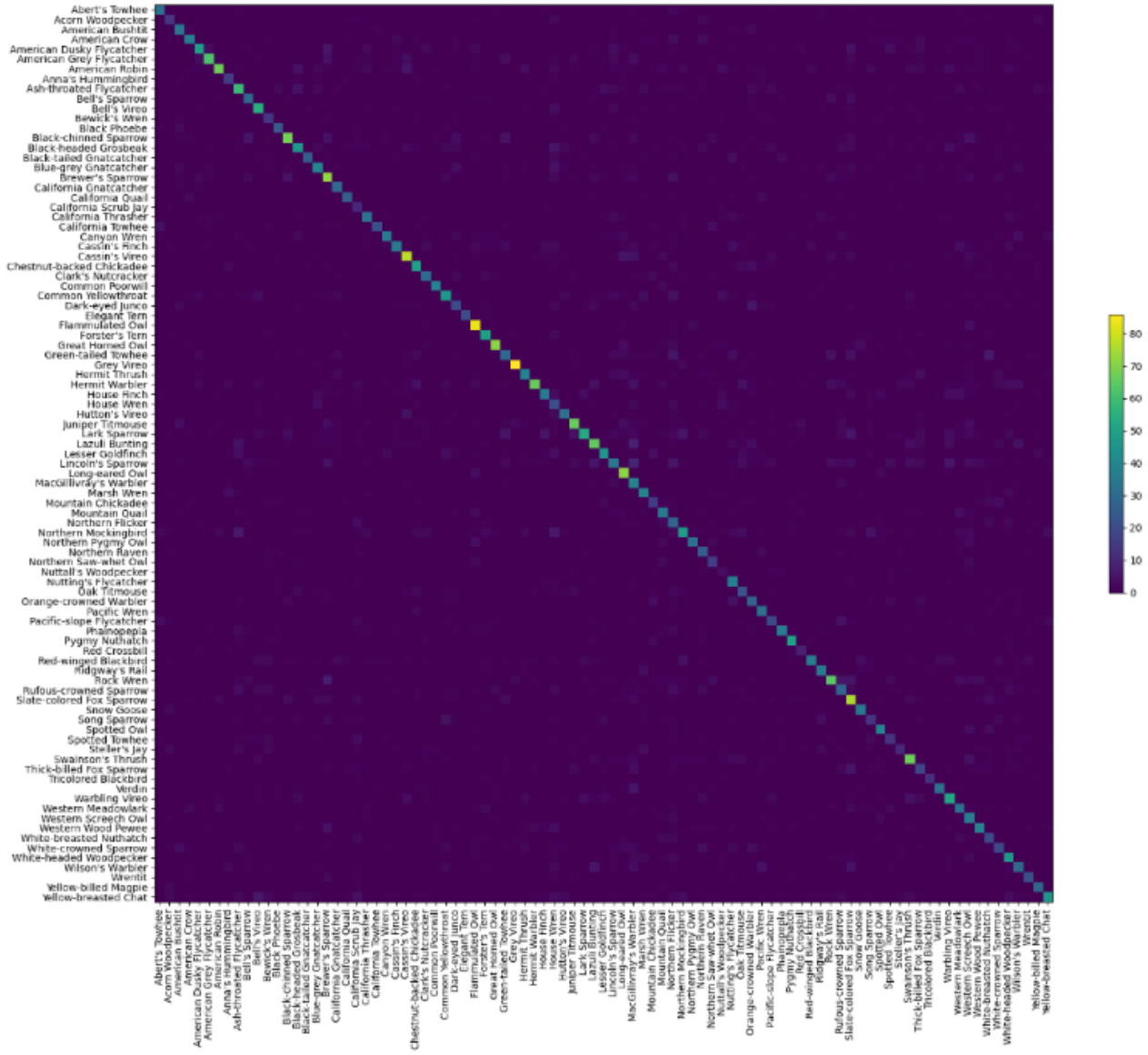

Figura 16. Matriz de confusión Baseline.

Tal y como se puede apreciar por la matriz y la leyenda, el clasificador cumple en general bastante bien con su función. No hay un error demasiado acentuado, siendo la diagonal principal de la matriz bastante clara y sin colores resaltados ni en la mitad inferior ni en la superior.

5.1.2 Modelo 1

El modelo 1 era el que mejores resultados había dado, concretamente 0.7124 de *accuracy*. Los resultados con el *set* de test son de **0.65**, lo cual sigue siendo un resultado bastante bueno teniendo en cuenta que no ha visto esos ejemplos nunca. Su matriz de confusión es la siguiente:

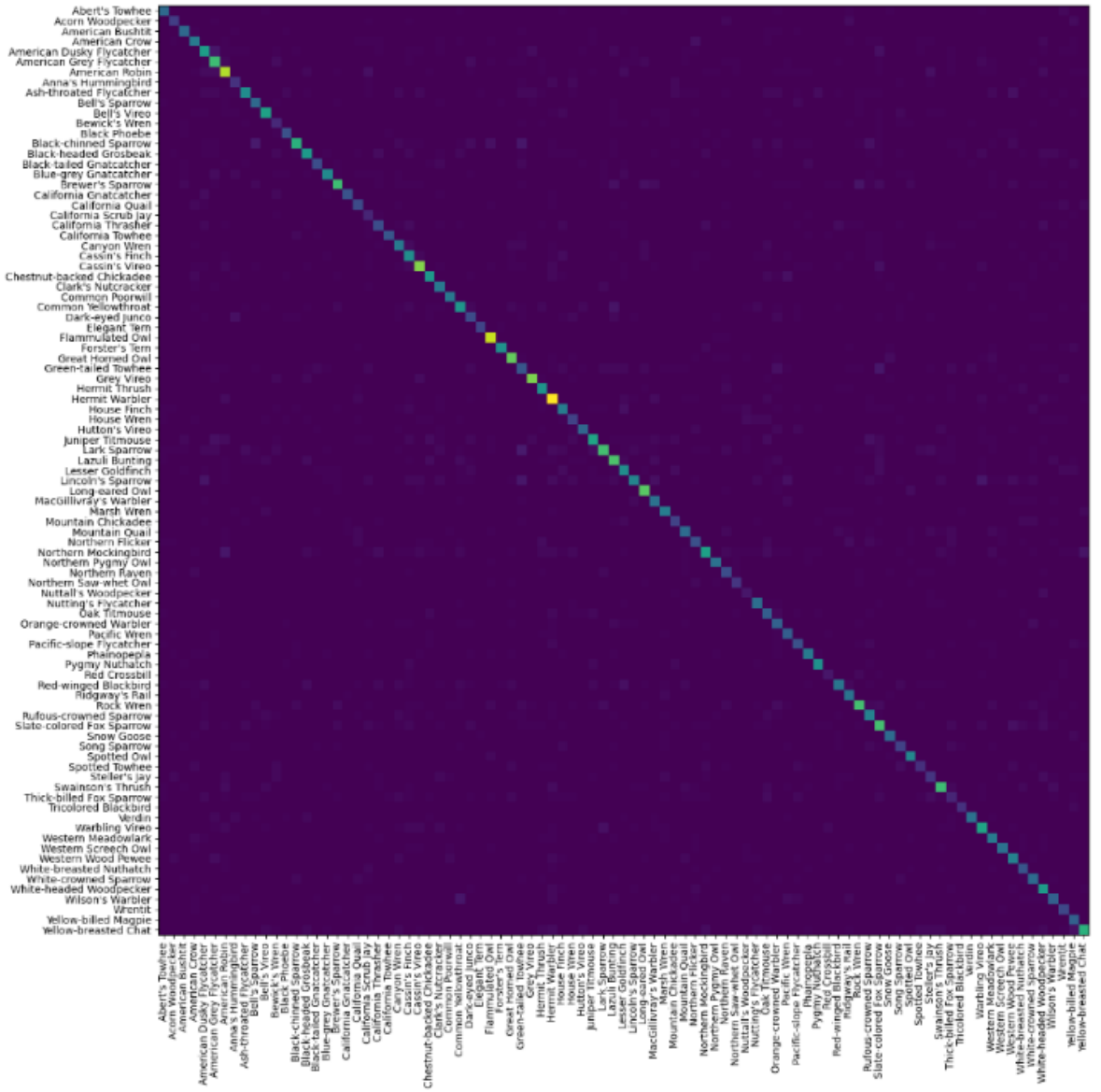

Figura 17. Matriz de confusión del modelo 1.

De nuevo no se aprecian colores fuertes en ninguna mitad, consiguiendo una diagonal principal muy clara. Se observa más fallos en la matriz anterior, como es lógico. Teniendo en cuenta que este es el mejor modelo, esta es la mejor matriz de confusión que se mostrará.

5.1.3 Modelo 2

El modelo 2 tenía una *accuracy* de 0.523, mientras que al pasar el *set* de prueba obtiene **0.573**, superando así el resultado esperado. A continuación, se muestra la matriz de confusión:

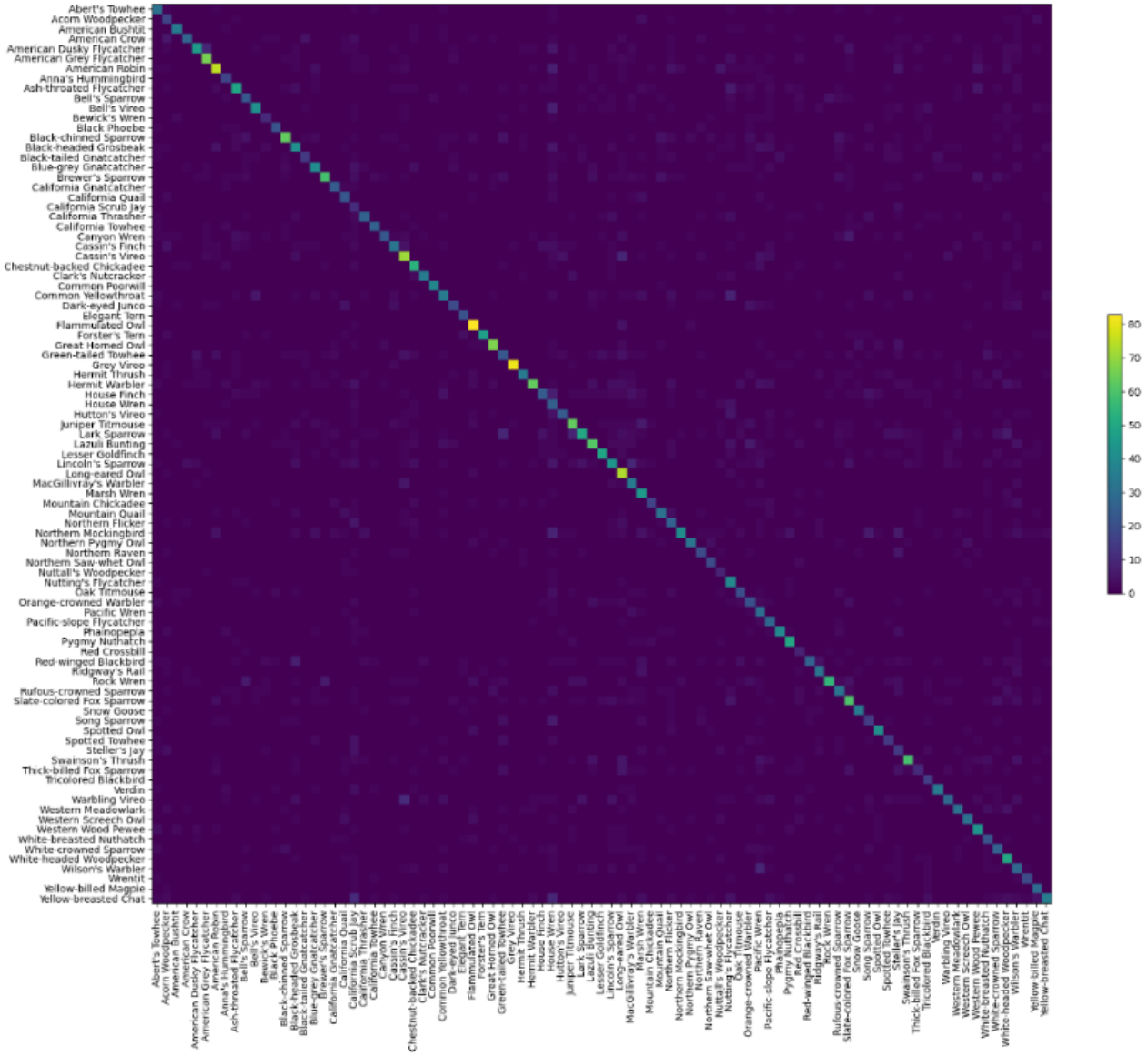

Figura 18. Matriz de confusión del modelo 2.

5.1.4 Modelo 3

El modelo 2 tenía una *accuracy* de 0.53, mientras que al pasar el *set* de prueba obtiene **0.59**, mejorando también el resultado esperado al igual que el modelo 2. A continuación se muestra la matriz de confusión:

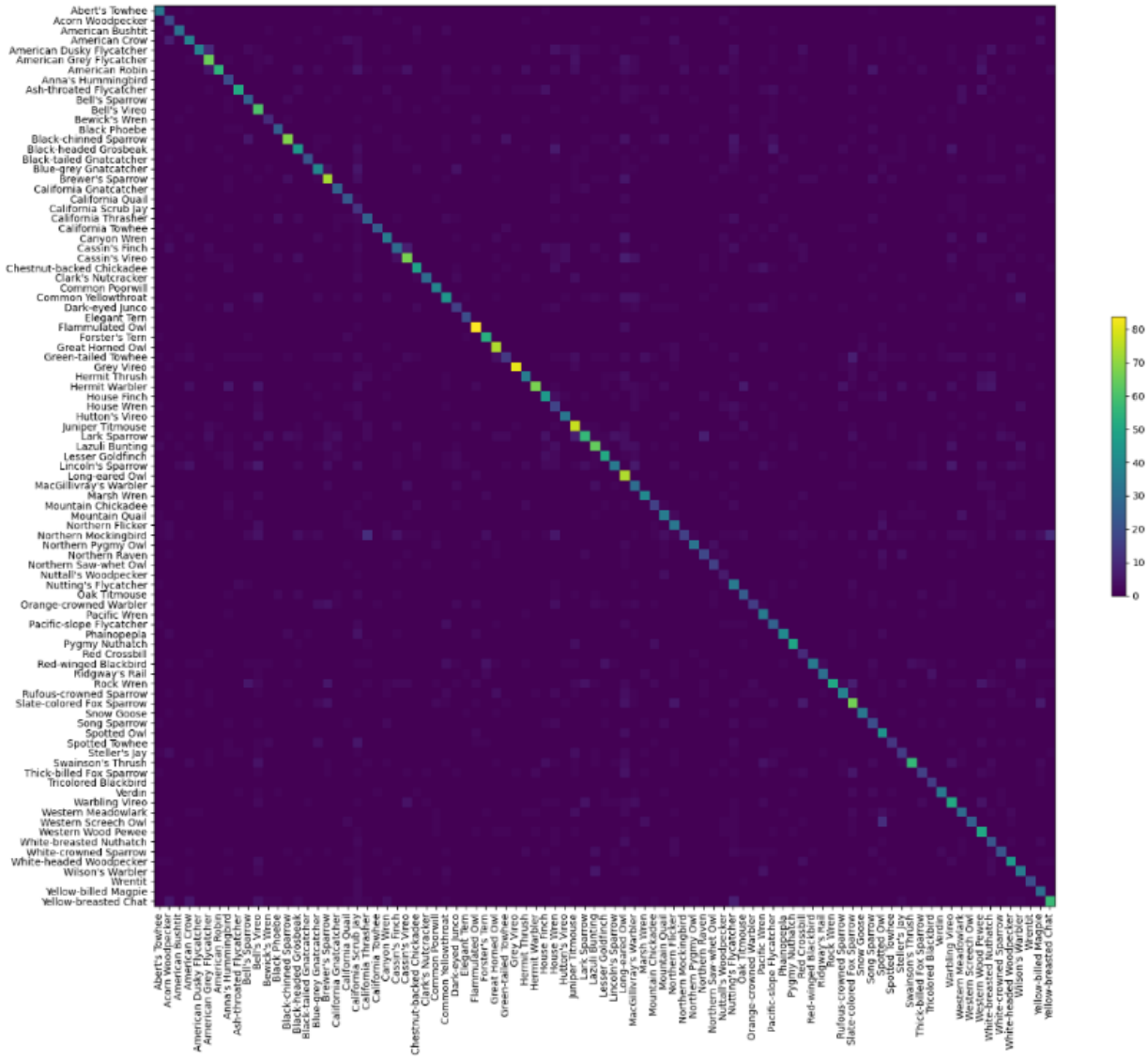

Figura 19. Matriz de confusión del modelo 3.

5.1.5 Modelo 4

El modelo 2 tenía una *accuracy* de 0.7124, mientras que al pasar el *set* de prueba obtiene **0.654**. Tras este resultado se puede confirmar que los modelos en los que el *dropout* se ha aumentado consiguen mejor resultado en el test, a pesar de tener peores resultados con el *set* de entrenamiento. A continuación, se muestra la matriz de confusión:

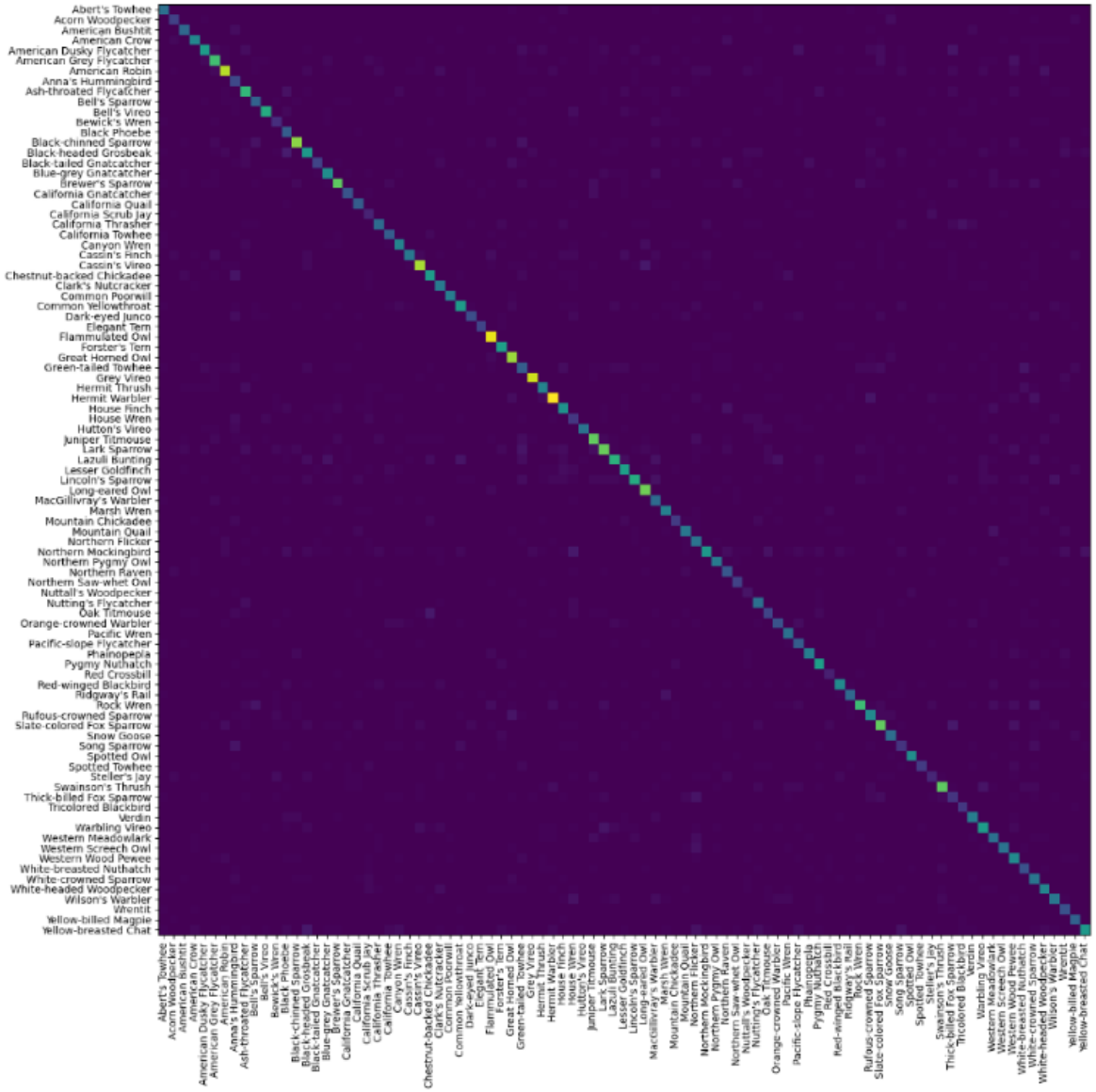

Figura 20. Matriz de confusión del modelo 4.

5.2 Base de datos de l'Albufera

Tal como se comentó en el punto 3.5.2, para clasificar la base de datos de la Albufera se usaría el mejor conjunto de hiperparámetros que resultara del análisis con la base de datos de prueba. Todo ello tras el preprocesado de estos audios: recortándolos en trozos de 3.15 segundos, obteniendo sus espectrogramas Mel y coeficientes MFCC, y finalmente pasando los *sets* de entrenamiento y de test al clasificador.

A la hora de entrenar se han usado tres cuartos del total de audios, es decir, 4225 audios. Por tanto, se entrena con menos muestras de las que se "testea" con la base de datos de *kaggle*. Sin embargo, los resultados de este entrenamiento son realmente buenos:

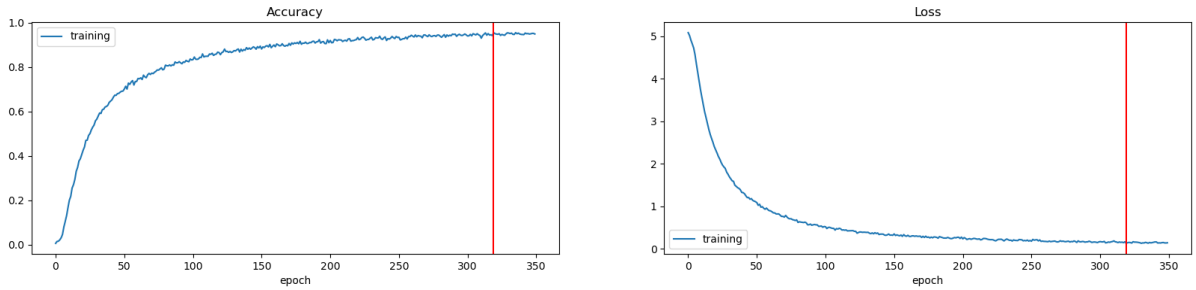

Figura 21. Curva de aprendizaje del modelo de la Albufera.

Se alcanza un **0.9483** de *accuracy* sin necesidad de emplear 1000 épocas, tan solo con 350. Teniendo en cuenta que con la base de datos de *kaggle*, mismo preprocesado, misma CNN y casi 100 especies menos que clasificar se obtiene alrededor de 0.7, se puede concluir en que es un resultado no esperado y no tan confiable como el anterior estudio. El principal problema que puede aparecer es *overfitting*, es decir, el modelo se estaría ajustando demasiado bien a las muestras en el entrenamiento. A su vez, esto ocurriría por tener menos grabaciones de las que sería ideal, de modo que, aunque se troceen y se tengan ejemplos de sobra, estos ejemplos son muy parecidos entre sí y la red estaría aprendiendo otras características propias de la grabación y no tanto del sonido de los pájaros como tal. Es decir, se aprende muy bien las imágenes, pero no hay certeza de que haya aprendido correctamente las características del canto de cada pájaro.

A la hora de pasar el *set* de prueba, el cual no ha visto nunca este modelo, la *accuracy* resultante es de **0.9431**, prácticamente la misma que la de entrenamiento. En consecuencia, parece claro que se ha aprendido características de las propias imágenes más que de los cantos y, además, que hay pocas muestras, faltando diversidad y favoreciendo que la CNN haga “trampas” para acertar con el entrenamiento. La manera de comprobar fácilmente que faltan muestras/diversidad es que el resultado es el mismo con un *set* que no conoce, lo que quiere decir que son muy parecidas todas las muestras. Esto hace que la base de datos no sea de gran utilidad para entrenar un modelo, y que deba usarse para la detección de patrones en las especies como uno de los pasos previos a la grabación de la base de datos definitiva en la Albufera.

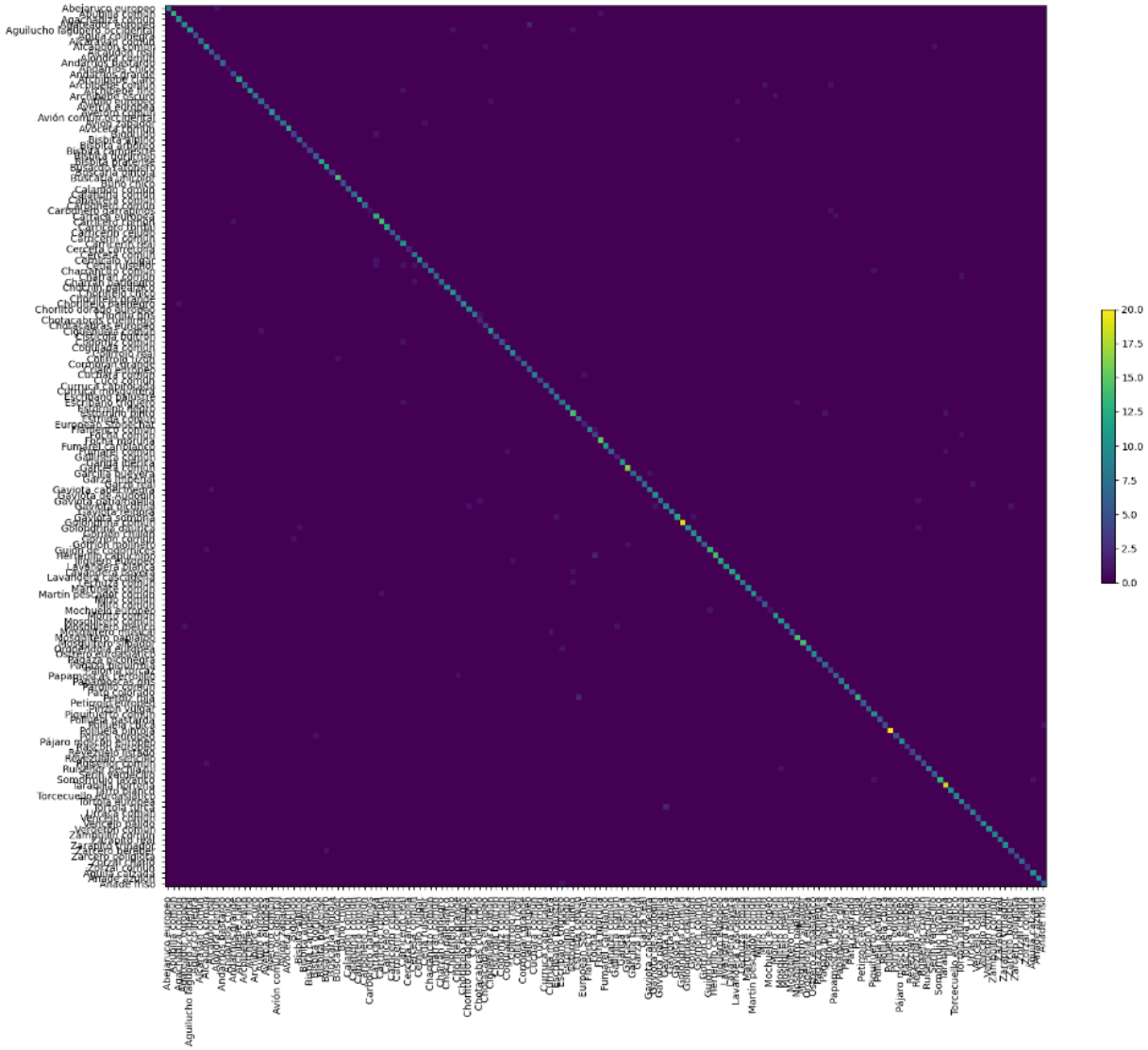

Figura 22. Matriz de confusión del modelo de la Albufera.

Idealmente, la mejor solución es añadir muestras que además correspondan a grabaciones distintas. Sin embargo, existe otra solución. Consiste en dividir el conjunto entero (entrenamiento + test) en varias carpetas, diez, por ejemplo. De estas diez, nueve se usarían para entrenar y darían lugar a nueve modelos distintos. La carpeta sobrante sería el conjunto de test. La *accuracy* finalmente sería el promedio de las obtenidas con los 9 modelos. Este recurso contra el *overfitting* se conoce como *K-fold Cross Validation*, y se detalla en la siguiente imagen:

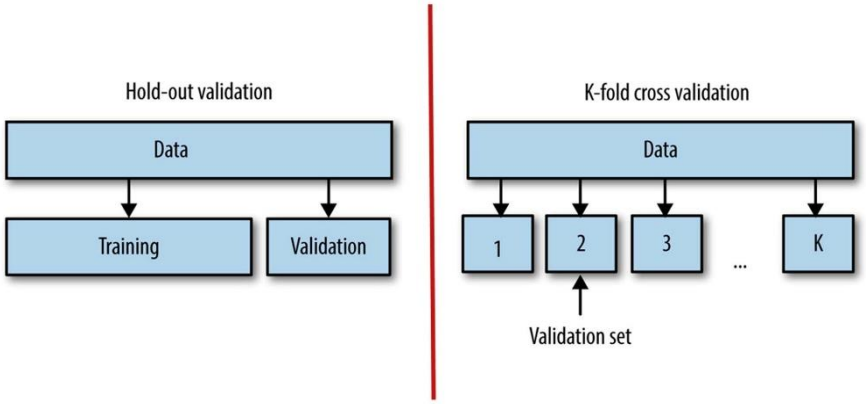

Figura 23. Validación clásica vs *K-fold Cross Validation*.

Al no tratarse de una base de datos definitiva, sino más bien una prueba, esta solución no se llevará a cabo y se tendrá en cuenta el problema que puede ocasionar la falta de audios a la hora de grabar la base de datos final. Se sabe que la CNN funciona bien, que el preprocesado es correcto y que los resultados son satisfactorios para un número de especies grande. Si la nueva base de datos es buena cualitativa y cuantitativamente, el rendimiento será bueno. Si además se establece como objetivo clasificar menos especies, se facilitará la tarea de la CNN y los resultados seguramente sean algo mejores que 0.7. Por lo tanto, el resultado de la prueba con la base de datos de la Albufera construida a partir de audios de *xeno-canto* no es problemático.

Capítulo 6. Conclusiones

Primera.

Uno de los aspectos más importantes a la hora de desarrollar un clasificador no tiene que ver con el clasificador en sí, sino con todo el trabajo previo. El preprocesado de los datos en crudo (audios con extensión mp3 en este caso) es de vital importancia, pues es de donde va a nutrirse el clasificador, tanto para el proceso de aprendizaje como en el de decisión. De hecho, este trabajo puede ser el más laborioso de todos.

Segunda.

La base de datos contiene los datos en crudo, siendo el origen de todo. De nuevo, es un punto clave como ya se ha podido comprobar y sigue siendo un punto ajeno al clasificador, incluso ajeno a cualquier conocimiento técnico como puede ser el lenguaje de programación, por ejemplo, que es bastante transversal a todo el trabajo y una condición necesaria para poder llevarlo a cabo. Tener una base de datos en la cual abunden las muestras y además haya diversidad es primordial. Es el primer paso de todos y de él depende en gran medida que los resultados sean satisfactorios y tengan sentido.

Tercera.

El rendimiento de una red neuronal convolucional depende tanto de su arquitectura como del valor que se define para los hiperparámetros. Una arquitectura acertada garantizará un resultado correcto, sin embargo, variar sus hiperparámetros en busca de la mejor combinación es lo que la llevará a ser óptima. Su implementación consta de dos etapas: definición de la arquitectura (número de capas, tipo, características de la entrada y de la salida…) y optimización del resultado mediante los hiperparámetros.

Capítulo 7. Anexos

7.1 Listado de la base de datos de *Kaggle*

Especies	Número de audios	Duración total (s)
Abert's Towhee	30	554
Acorn Woodpecker	30	354
American Bushtit	30	652
American Crow	30	846
American Dusky Flycatcher	30	1220
American Grey Flycatcher	30	1228
American Robin	30	1666
Anna's Hummingbird	30	416
Ash-throated Flycatcher	30	1136
Bell's Sparrow	30	482
Bell's Vireo	30	764
Bewick's Wren	30	379
Black Phoebe	30	524
Black-chinned Sparrow	30	1584
Black-headed Grosbeak	30	1073
Black-tailed Gnatcatcher	30	572
Blue-grey Gnatcatcher	30	943
Brewer's Sparrow	30	1558
California Gnatcatcher	30	476
California Quail	30	624
California Scrub Jay	30	307
California Thrasher	30	727
California Towhee	30	466
Canyon Wren	30	859
Cassin's Finch	30	861
Cassin's Vireo	30	1344
Chestnut-backed Chickadee	30	963
Clark's Nutcracker	30	672
Common Poorwill	30	605
Common Yellowthroat	30	1071

Dark-eyed Junco	30	557
Elegant Tern	30	354
Flammulated Owl	30	1247
Forster's Tern	30	701
Great Horned Owl	30	1077
Green-tailed Towhee	30	865
Grey Vireo	30	1041
Hermit Thrush	30	900
Hermit Warbler	30	1684
House Finch	30	909
House Wren	30	525
Hutton's Vireo	30	604
Juniper Titmouse	30	1252
Lark Sparrow	30	1877
Lazuli Bunting	30	1399
Lesser Goldfinch	30	888
Lincoln's Sparrow	30	1337
Long-eared Owl	30	1193
MacGillivray's Warbler	30	743
Marsh Wren	30	622
Mountain Chickadee	30	372
Mountain Quail	30	764
Northern Flicker	30	681
Northern Mockingbird	30	1554
Northern Pygmy Owl	30	687
Northern Raven	30	467
Northern Saw-whet Owl	30	305
Nuttall's Woodpecker	30	242
Nutting's Flycatcher	30	660
Oak Titmouse	30	468
Orange-crowned Warbler	30	673
Pacific Wren	30	671
Pacific-slope Flycatcher	30	467
Phainopepla	30	972

Pygmy Nuthatch	30	786
Red Crossbill	30	185
Red-winged Blackbird	30	1133
Ridgway's Rail	30	576
Rock Wren	30	1535
Rufous-crowned Sparrow	30	831
Slate-colored Fox Sparrow	30	1396
Snow Goose	30	572
Song Sparrow	30	404
Spotted Owl	30	529
Spotted Towhee	30	365
Steller's Jay	30	373
Swainson's Thrush	30	1417
Thick-billed Fox Sparrow	30	626
Tricolored Blackbird	30	234
Verdin	30	569
Warbling Vireo	30	1055
Western Meadowlark	30	725
Western Screech Owl	30	727
Western Wood Pewee	30	828
White-breasted Nuthatch	30	453
White-crowned Sparrow	30	768
White-headed Woodpecker	30	940
Wilson's Warbler	30	687
Wrentit	30	423
Yellow-billed Magpie	30	481
Yellow-breasted Chat	30	1206
(en blanco)		
Total general	**2730**	**73508**

7.2 Listado de la base de datos de la Albufera

Especies	Número de audios	Duración total (s)
Abejaruco europeo	6	117,0954828
Abubilla común	4	139,6149563
Agachadiza común	5	91,97040547
Agateador europeo	5	170,3783168
Águila calzada	6	149,0695483
Aguilucho lagunero occidental	7	81,80712032
Aguja colinegra	7	116,1009861
Alcaraván común	13	125,0594161
Alcaudón común	4	144,599321
Alcaudón real	6	105,7368245
Alondra común	3	104,5514861
Ánade azulón	6	71,88938931
Ánade friso	4	165,6058571
Andarríos bastardo	10	129,7327065
Andarríos chico	11	85,63859623
Andarríos grande	6	120,8896835
Archibebe claro	6	126,5173113
Archibebe común	7	140,9362717
Archibebe fino	7	113,2515833
Archibebe oscuro	10	186,2036872
Autillo europeo	4	164,3544742
Avefría europea	7	107,8500984
Avetoro común	5	117,5902649
Avión común occidental	5	78,24364612
Avión zapador	6	126,8281617
Avoceta común	5	94,19010417
Bigotudo	3	61,08075368
Bisbita alpino	7	108,1271609
Bisbita arbóreo	5	124,1556281
Bisbita campestre	6	127,9021327
Bisbita gorjirrojo	10	99,88431944
Bisbita pratense	5	123,8355686

Búho chico	3	101,1284141
Busardo ratonero	5	132,6349806
Buscarla pintoja	3	78,7576352
Buscarla unicolor	3	109,4881859
Calamón común	7	80,95493155
Calandria común	5	120,7982708
Canastera común	9	171,5667601
Carbonero común	4	151,4623129
Carbonero garrapinos	5	144,6180601
Carraca europea	5	116,6148508
Carricerín cejudo	3	99,13172917
Carricerín común	5	114,2601355
Carricerín real	3	71,12063818
Carricero común	4	91,46299972
Carricero tordal	3	94,34209838
Cerceta carretona	7	135,922362
Cerceta común	6	105,0407344
Cernícalo vulgar	6	134,4173525
Cetia ruiseñor	9	89,49030145
Charrán común	5	114,90525
Charrán patinegro	9	155,122381
Charrancito común	10	116,0698427
Chochín paleártico	4	140,5052608
Chorlitejo chico	5	118,7868132
Chorlitejo grande	8	119,6521256
Chorlitejo patinegro	8	82,9170566
Chorlito dorado europeo	7	71,81242775
Chorlito gris	10	127,5443676
Chotacabras cuellirrojo	5	120,1365561
Chotacabras europeo	5	113,9649852
Cigüeñuela común	4	103,7445017
Cistícola buitrón	5	120,9871542
Codorniz común	9	93,72808404
Cogujada común	4	140,1707917

Colirrojo real	5	126,1495332
Colirrojo tizón	5	164,9702466
Cormorán grande	6	115,907953
Críalo europeo	8	73,82051545
Cuchara común	5	115,627875
Cuco común	5	119,1907327
Curruca capirotada	7	168,143531
Curruca mosquitera	5	150,3435558
Escribano palustre	6	228,4909059
Escribano triguero	6	138,7520272
Estornino negro	4	83,30279167
Estornino pinto	4	111,7307174
Estrilda común	6	99,31561066
European Stonechat	5	161,4571837
Flamenco común	7	143,3252973
Focha común	5	93,27459807
Focha moruna	4	62,41491811
Fumarel cariblanco	6	73,49583333
Fumarel común	8	134,5577756
Gallineta común	7	86,05990595
Ganga ibérica	3	97,09600907
Garceta común	16	162,2717072
Garcilla bueyera	7	110,2633181
Garza imperial	8	144,9338793
Garza real	4	113,0357215
Gaviota cabecinegra	11	159,8337943
Gaviota de Audouin	6	124,2820259
Gaviota patiamarilla	7	186,6070115
Gaviota picofina	8	217,4586814
Gaviota reidora	9	133,1332252
Gaviota sombría	5	123,0913872
Golondrina común	4	100,6863778
Golondrina dáurica	6	103,2251625
Gorrión chillón	3	136,090697

Gorrión común	7	166,5311893
Gorrión molinero	4	116,185717
Guión de codornices	3	158,6580434
Herrerillo capuchino	6	130,0826713
Jilguero europeo	6	148,1899786
Lavandera blanca	6	97,0058108
Lavandera boyera	6	117,6802999
Lavandera cascadeña	7	128,3107114
Lechuza común	10	91,33346287
Martín pescador común	4	76,27008588
Martinete común	7	38,08072985
Mirlo común	4	152,8564056
Mito común	5	126,8531014
Mochuelo europeo	3	117,6817911
Morito común	7	98,7038375
Mosquitero común	7	162,5805125
Mosquitero ibérico	5	131,5745437
Mosquitero musical	4	138,195624
Mosquitero papialbo	4	121,4006781
Mosquitero silbador	6	102,738684
Oropéndola europea	6	114,5921131
Ostrero euroasiático	5	108,7373491
Pagaza piconegra	4	102,7525821
Pagaza piquirroja	8	91,79160828
Pájaro moscón europeo	6	138,2370865
Paloma torcaz	5	94,1292824
Papamoscas cerrojillo	8	176,7359875
Papamoscas gris	4	100,2155708
Pardillo común	4	100,9609135
Pato colorado	6	122,9495896
Perdiz roja	6	102,0485803
Petirrojo europeo	2	87,01053189
Pinzón vulgar	5	211,8882388
Piquituerto común	6	94,85175772
Polluela bastarda	3	95,35288535

Polluela chica	3	80,72191667
Polluela pintoja	3	108,8448073
Porrón europeo	9	157,1957
Rascón europeo	5	70,94705215
Reyezuelo listado	4	119,4221889
Reyezuelo sencillo	3	103,8850208
Ruiseñor común	4	185,1663038
Ruiseñor pechiazul	4	160,0596958
Serín verdecillo	5	134,4351392
Somormujo lavanco	7	100,1519004
Tarabilla norteña	5	134,3984053
Tarro blanco	5	80,40943183
Torcecuello euroasiático	2	115,3444891
Tórtola europea	5	83,81479762
Tórtola turca	5	81,59866851
Urraca común	4	115,3785805
Vencejo común	5	98,58759963
Vencejo pálido	12	127,8054482
Verderón común	4	145,5604675
Zampullín común	9	101,0216026
Zarapito real	4	71,5438845
Zarapito trinador	5	103,1193659
Zarcero bereber	4	65,60877012
Zarcero políglota	4	114,2029583
Zorzal charlo	4	167,2273539
Zorzal común	4	175,5093288
(en blanco)		
Total general	**922**	**19231,42095**

7.3 Código utilizado para el preprocesado

Disponible en:
https://drive.google.com/drive/folders/1LdcOBbT2EFnNlVSa9Cntb9tLoJo9DdgQ?usp=sharing

```
import os
import math
import numpy as np
import pandas as pd
import librosa as lib
from librosa import display as display
import warnings
import matplotlib.pyplot as plt
from pydub.utils import make_chunks
from pydub import AudioSegment
from sklearn.preprocessing import StandardScaler
import sklearn
import csv
```

```
# To avoid a warning everytime it reads an mp3 file
warnings.filterwarnings('ignore', message='PySoundFile failed. Trying audioread in
                        category=Warning, module='librosa',
                        lineno=0, append=False)

# 3.15 seconds file in miliseconds
size = 3150

# The length of a 8 seconds duration file
len_8secs = 44100*8
```

```
files_path = r"C:\Users\David Nuño\Desktop\TFG\dataset_albufera(XC)"
output_path = "C:\\Users\\David Nuño\\Desktop\\TFG\\chunked_dataset_albufera\\"

# pad_range = len_8secs - len(y)

list_ID = pd.read_excel(r"C:\Users\David Nuño\Desktop\TFG\metadata_albufera.xlsx",
list_ID = list_ID.values.tolist()
```

```
#Export the chunked files to a new folder

for ID in list_ID:

    file = os.path.join(files_path, ID[0])
    #y, fs = lib.load(file, sr=None)

    try:
        audio = AudioSegment.from_file(file, "mp3")
    except:
        audio = AudioSegment.from_file(file, "mp4")

    if len(audio)/1000 <= 3.15:
    #     y = np.pad(y, (pad_range,0), 'constant', constant_values=0)
        silent = AudioSegment.silent(duration=3150-len(audio))
        chunk = audio.append(silent, crossfade=0)
        chunk_name = str(ID[0]).split('.')[0].split('-')[0].strip() + '0.' + str(I
        chunk_name = output_path + chunk_name
        chunk.export(chunk_name, format="mp3")

    else:
    #     parts = math.floor(len(y)/len_8secs)
```

```
    #     for part in range(0, parts*len_8secs, len_8secs):

        chunks = make_chunks (audio, size) ## Corta el archivo en trozos de 10 segu

        for i, chunk in enumerate(chunks[:-1]):
                ## Enumeración, i es el índice, chunk es el archivo cortado
            chunk_name = (str(ID[0]).split('.')[0].split('-')[0].strip() + '{0}.' +
            chunk_name = output_path + chunk_name
            #print(chunk_name)
                ## guardar documento
            chunk.export(chunk_name, format="mp3")
```

```
# print(y.shape)
# print(len(y))

#Sorting the ID and species list including the chunked elements

list_species = pd.read_excel(r"C:\Users\David Nuño\Desktop\TFG\metadata_albufera.x
list_species = list_species.values.tolist()

chunked_list_ID = os.listdir(r"C:\Users\David Nuño\Desktop\TFG\chunked_dataset")

new_list_ID = []
new_list_species = []

for ID in list_ID:
    index = list_ID.index(ID)
    for i in range(70):
        filename = (str(ID[0]).split('.')[0] + '{0}.' + str(ID[0]).split('.')[1]).
        if filename in chunked_list_ID:
            new_list_ID.append(filename)
            specie = list_species[index]
            new_list_species.append(specie[0])


# for ID in chunked_list_ID:

#     ID_aux = ID
#     ID_aux = ID_aux.split('.')[0]
#     ID_aux = ID_aux[:-1]
#     ID_aux = ID_aux + '.mp3'

#     ID_aux2 = ID
#     ID_aux2 = ID_aux2.split('.')[0]
#     ID_aux2 = ID_aux2[:-2]
#     ID_aux2 = ID_aux2 + '.mp3'

#     if [ID_aux] in list_ID:
#         index = list_ID.index([ID_aux])

#         if ID.split('.')[0][-1:] == '0':
#             list_ID[index] = [ID]
#         else:
#             list_ID.insert(index+1, [ID])
#             list_species.insert(index+1, list_species[index])

#     elif [ID_aux2] in list_ID:
#         index = list_ID.index([ID_aux2])
#         list_ID.insert(index+1, [ID])
#         list_species.insert(index+1, list_species[index])
```

```
# Definición de prámetros para el análisis. No dependden de fs
B_mel = 128 # Número de bandas Mel

# reference values
ref_a2dB = 5e-2
amin_a2dB = 1e-08
ref_p2dB = 20e-6
amin_p2dB = 1e-16
```

```
filenames_path = r"C:\Users\David Nuño\Desktop\TFG\chunked_dataset_albufera"
mfcc_shapes = []
mel_shapes =[]
file_ids = []

new_list_ID = os.listdir(filenames_path)


for ID in new_list_ID:

    fn_mp3 = os.path.join(filenames_path, ID)
    y, fs = lib.load(fn_mp3, sr=None)

    T_frame = 0.05 # 50 ms        3.15  s de frame
    L_frame = round(T_frame*fs)
    L_hop = int(L_frame/2)


    p = np.ceil(np.log2(L_frame))
    nfft = int(2**p)

    #Mel spectrogram
    specM = lib.feature.melspectrogram(y = y, sr = fs, n_fft = nfft, n_mels = B_me
                                       hop_length = L_hop, win_length = L_frame)
    specM_dB = lib.power_to_db(specM, ref = ref_p2dB, amin=amin_p2dB)

    specM_dB = specM_dB[:,:127]


    #MFCC
    mfcc = lib.feature.mfcc(y=y, sr=fs, n_fft=nfft,
                                       n_mfcc=20, n_mels=B_mel,
                                       hop_length=L_hop, win_length=L_frame)

    mfcc = mfcc[:,:127]

    #Export as .dat
    specM_dB.tofile(r'C:\Users\David Nuño\Desktop\TFG\mel_spec_albufera\XC%s_melsp
    mfcc.tofile(r'C:\Users\David Nuño\Desktop\TFG\mfcc_albufera\XC%s_mfcc.dat'%ID.

    #Export shapes as .csv
    file_id = str(ID).split('.')[0].split('C')[1]

    file_ids.append(file_id)
    mfcc_shapes.append(mfcc.shape)
    mel_shapes.append(specM_dB.shape)


header = ["file_id", "melspectrogram_shapes", "mfcc_shapes"]

numpy_array = np.array([file_ids, mel_shapes, mfcc_shapes])
transpose = numpy_array.T
data = transpose.tolist()
```

```
df = pd.DataFrame(data, columns=header)

print(df)

df.to_csv(r'C:\Users\David Nuño\Desktop\TFG\feature_shapes_albufera.csv')
```

7.4 Código del clasificador

Disponible en:
https://drive.google.com/drive/folders/1LdcOBbT2EFnNlVSa9Cntb9tLoJo9DdgQ?usp=sharing

```
import pandas as pd
import numpy as np
import librosa as lr
from librosa.display import specshow
from glob import glob
import os
from IPython.display import Audio
from matplotlib import pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import StratifiedShuffleSplit, train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import tensorflow
import keras
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dropout
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras.models import Sequential
from itertools import islice
from sklearn.utils import class_weight
import re
import math
import joblib
import pickle


config = tensorflow.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
# config.gpu_options.per_process_gpu_memory_fraction = 1.0
sess = tensorflow.compat.v1.Session(config=config)


def parse_shape(shape_str):
    """Shape was saved in feature_shapes as a string. Woops. Parse out the values. """
    a, b = re.search('\((\d+), (\d+)\)', shape_str).groups()
    return int(a), int(b)


def log_clipped(a):
    """Convenience function to clip the input to positive values then return the log."""
    return np.log(np.clip(a, .0000001, a.max()))


# sounds_dir = r"/home/danupe/chunked_dataset_albufera"
# # melspec_dir = r"C:\Users\David Nuño\Desktop\TFG\mel_spec"
#
# files_list = glob(os.path.join(sounds_dir,"*.mp3"))
# # print("%i mp3 files in %s"%(len(files_list), sounds_dir))
#
shapes_df = pd.read_csv(r"/home/danupe/feature_shapes_albufera.csv",index_col=0 )
# display(shapes_df.head(2))


list_species = pd.read_excel(r"/home/danupe/metadata_albufera.xlsx", usecols="C")
list_species = list_species.values.tolist()

list_ID = pd.read_excel(r"/home/danupe/metadata_albufera.xlsx", usecols="D")
list_ID = list_ID.values.tolist()

chunked_list_ID = os.listdir(r"/home/danupe/chunked_dataset_albufera")

new_list_ID = []
new_list_species = []

for ID in list_ID:
    index = list_ID.index(ID)
    for i in range(50):
        filename = ('XC' + str(ID[0]) + '{0}.' + 'mp3').format(i)
        if filename in chunked_list_ID:
            new_list_ID.append(filename)
            specie = list_species[index]
            new_list_species.append(specie[0])


list_label = []
names = []
i = 0
j = 0

for name in new_list_species:
    names.append(name)
    if name == new_list_species[0]:
        list_label.append(0)
        j+=1
    elif name != names[j-1]:
        i+=1
        list_label.append(i)
        j+=1
```

```
    else:
        list_label.append(i)
        j+=1

df = pd.DataFrame(columns = ['file_id','label','species'])
df['file_id'] = new_list_ID
df['species'] = new_list_species
df['label'] = list_label
# display(df.head(10))

df_shuffled=df.sample(frac=1).reset_index(drop=True)
# display(df_shuffled.head(50))

num_classes = len(np.unique(df_shuffled.label))
# print(num_classes)

# print(list_label[-100:])
unique_vals = list(dict.fromkeys(list_label))
# print(unique_vals)
# print(len(list_label))
# print(list_label[0])
# print(list_label[1])

ids = df_shuffled['file_id'].values.tolist()
id_list = []

for ID in ids:
    ident = ID.split('.')[0].split('C')[1]
    id_list.append(ident)

df_shuffled['file_id'] = id_list
#
df_shuffled.to_csv("full_ds_albufera.csv")
# df_shuffled = pd.read_csv(r'/home/danupe/full_ds_albufera.csv')

train_df = df_shuffled.iloc[:int(round(3*5633/4))]
# print(len(train_df))
# display("Training data:",train_df.head(20))

test_df = df_shuffled.iloc[-int(math.floor(1*5633/4)):]
# print(len(test_df))
# display("Test data:",test_df.head(20))

# print(len(test_df)+len(train_df))

y_english_labels_entire_dataset = [s['species'] for i, s in df_shuffled.iterrows()]
label_encoder = LabelEncoder().fit(y_english_labels_entire_dataset)
n_classes = len(label_encoder.classes_)
# print(n_classes)

X_train = list(train_df['file_id'])
y_train = list(train_df['label'])
print(len(X_train))
# print("Training data len:",len(X_train), len(y_train))
# print(X_train[10:20])

class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train)

d_class_weights = dict(enumerate(class_weights))

X_test = list(test_df['file_id'])
y_test = list(test_df['label'])


mfcc_scaler = StandardScaler()
mel_scaler = StandardScaler()

for file_id in shapes_df.file_id:
    mfcc = np.memmap(r'/home/danupe/mfcc_albufera/XCXC%s_mfcc.dat'%file_id,
        shape=parse_shape(shapes_df[shapes_df.file_id==file_id]['mfcc_shapes'].values[0]),  dtype='float32', mode='readonly')
    mfcc_scaler.partial_fit(np.transpose(mfcc))
    mel = np.memmap(r'/home/danupe/mel_spec_albufera/XCXC%s_melspectrogram.dat'%file_id,
        shape=parse_shape(shapes_df[shapes_df.file_id==file_id]['melspectrogram_shapes'].values[0]),  dtype='float32', mode='readonly')
    mel_scaler.partial_fit(np.transpose(mel))


class AudioFeatureGenerator(tensorflow.keras.utils.Sequence):
    'Generates data for Keras'

    def __init__(self, list_IDs, labels, batch_size, n_frames=127, n_channels=1,
                 n_classes=10, shuffle=False, seed=37):
        'Initialization'
        self.n_frames = n_frames
        self.dim = (128, self.n_frames)
        self.batch_size = batch_size
        self.labels = {list_IDs[i]: l for i, l in enumerate(labels)}
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.seed = seed
```

```
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        indexes = self.indexes[index * self.batch_size:(index + 1) * self.batch_size]
        list_IDs_temp = [self.list_IDs[k] for k in indexes]
        X, y = self.__data_generation(list_IDs_temp)
        return X, y

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.seed(self.seed)
            self.seed = self.seed + 1  # increment the seed so we get a different batch.
            np.random.shuffle(self.indexes)

    def __data_generation(self, list_IDs_temp):
        'Generates data containing batch_size samples'  # X : (n_samples, *dim, n_channels)
        X = np.empty((self.batch_size, *self.dim, self.n_channels))
        y = np.empty((self.batch_size, self.n_classes), dtype=int)
        # y = list()#np.empty((self.batch_size, self.n_classes), dtype=int) # one-hot encoded labels

        for i, ID in enumerate(list_IDs_temp):
            mel = np.memmap(r'/home/danupe/mel_spec_albufera/XCXC%s_melspectrogram.dat' % ID,
                            shape=(128, 127), dtype='float32', mode='r+')

            mfcc = np.memmap(r'/home/danupe/mfcc_albufera/XCXC%s_mfcc.dat' % ID,
                             shape=(20, 127), dtype='float32', mode='r+'

            # Normalize MFCCs and Mel Spectrograms
            mel = mel_scaler.transform(np.transpose(mel))
            mfcc = mfcc_scaler.transform(np.transpose(mfcc))

            mel = np.transpose(mel)
            mfcc = np.transpose(mfcc)


            X[i,] = mel.reshape(1, 128, self.dim[1], 1)
            # Overwrite the bottom of X with MFCCs (we don't need the low frequency bands anyway)
            X[i, :20] = mfcc.reshape(1, 20, self.dim[1], 1)
            y[i,] = to_categorical(self.labels[ID], num_classes=self.n_classes)
            # y.append(to_categorical(self.labels[ID], num_classes=self.n_classes))

        # y = np.array(y)

        return X, y


# In[ ]:


# generator = AudioFeatureGenerator(X_test, y_test, batch_size=1, shuffle=True, seed=37, n_frames=127, n_classes=n_classes)
# for g in islice(generator,0,4): # show a few examples
#     for i,spec in enumerate(g[0]):
#         plt.figure(figsize=(10,4))
#         spec_ax = specshow(spec.squeeze(), x_axis='time', y_axis='mel')
#         plt.title(label_encoder.classes_[np.argmax(g[1][i])])
#         plt.colorbar()
#         plt.show()
#         print(spec.squeeze())


# In[16]:


def vis_learning_curve(learning):
    train_loss = learning.history['loss']
    train_acc = learning.history['accuracy']
    val_loss = learning.history['val_loss'] if hasattr(learning.history, 'val_loss') else None
    val_acc = learning.history['val_acc'] if hasattr(learning.history, 'val_acc') else None

    fig, axes = plt.subplots(1, 2, figsize=(20, 4), subplot_kw={'xlabel': 'epoch'})
    axes[0].set_title("Accuracy")
    axes[0].plot(train_acc, label='training')
    if val_acc is not None: axes[0].plot(val_acc, label='validation')
    axes[0].legend()
    axes[1].set_title("Loss")
    axes[1].plot(train_loss, label='training')
    if val_acc is not None: axes[1].plot(val_loss, label='validation')
    axes[1].legend()

    # Plot a line to indicate the best epoch
    best_training_epoc = np.argmin(val_loss) if val_acc is not None else np.argmin(train_loss)
    axes[0].axvline(x=best_training_epoc, color="red")
    axes[1].axvline(x=best_training_epoc, color="red")
```

```
n_epochs = 350
params = {  # 'dim': (128,256),
    'n_frames': 127,
    'n_classes': n_classes,
    'n_channels': 1}
training_generator = AudioFeatureGenerator(X_train, y_train, batch_size=128, shuffle=True, seed=37, **params)
# validation_generator = AudioFeatureGenerator(cv_val_index, y_cv_val, batch_size=len(cv_val_index), **params)
dim = training_generator.dim

snapshot_filename = r"weights.h5"

checkpointer = ModelCheckpoint(filepath=snapshot_filename, verbose=1, save_best_only=True)

model = Sequential()
# model.add(Conv2D(16,(1,4),input_shape=(20+dim[0], dim[1], 1),padding='valid',activation="relu"))
model.add(Conv2D(64, 3, input_shape=(dim[0], dim[1], 1), padding='valid', activation="relu"))
model.add(MaxPooling2D(pool_size=3))
model.add(Dropout(rate=.2))
model.add(Conv2D(64, 3, padding='valid', activation="relu"))
model.add(MaxPooling2D(pool_size=3))
model.add(Dropout(rate=.2))
model.add(Conv2D(64, 3, padding='valid', activation="relu"))
model.add(MaxPooling2D(pool_size=3))
model.add(Dropout(rate=.2))
model.add(GlobalAveragePooling2D())
model.add(Dense(n_classes, activation="softmax"))
model.summary()

#initial_learning_rate = 0.1
# lr_schedule = tensorflow.keras.optimizers.schedules.ExponentialDecay(
#     initial_learning_rate=0.01,
#     decay_steps=2250,
#     decay_rate=0.96,
#     staircase=True)
# optimizer=tensorflow.keras.optimizers.Adam(learning_rate=lr_schedule)   #de 0.01 a 0.00009775

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
learning = model.fit(
    training_generator,
    #             validation_data=validation_generator,
    epochs=n_epochs,
    callbacks=[checkpointer],
    class_weight=d_class_weights,
    #             use_multiprocessing=True, workers=4,
    verbose=1)
pd.DataFrame(learning.history).to_csv('training_history_albufera.csv', index_label='epoch')
vis_learning_curve(learning)
plt.savefig("learning_curve_albufera.png")
#plt.show()
acc_at_min_loss = learning.history['accuracy'][np.argmin(learning.history['loss'])]
# print("Min training loss: %.5f, Training accuracy at min loss: %.5f"%(np.min(learning.history['loss']), acc_at_min_loss ))


model.save("modelo_albufera.h5")

params = {  # 'dim': (128,256),
     'n_frames': 127,
     'n_classes': n_classes,
     'n_channels': 1}

model = keras.models.load_model('/home/danupe/modelos/modelo_albufera.h5')
X_batch, y_batch = AudioFeatureGenerator(X_test, y_test, batch_size=len(X_test), **params)[0]
predictions = model.predict(X_batch)
y_predicted = [np.argmax(p) for p in predictions]
y_true = [np.argmax(y) for y in y_batch]
test_score = accuracy_score(y_true, y_predicted)
print("Test accuracy score: "+str(test_score))

conf_matrix = confusion_matrix(y_true, y_predicted, labels=range(n_classes))
pd.DataFrame(conf_matrix).to_csv('conf_matrix_albufera.csv')
plt.figure(figsize=(20,20))
plt.imshow(conf_matrix)
plt.xticks(range(n_classes), label_encoder.classes_, rotation='vertical')
plt.yticks(range(n_classes), label_encoder.classes_)
plt.colorbar(shrink=.25)
plt.savefig("conf_matrix_albufera.png")

plt.figure(figsize=(15,4))
plt.title("Percent Correctly Classified, by Species (Baseline)")
pct_correct_by_class = np.zeros(n_classes)
counts = np.sum(conf_matrix,axis=1)
np.divide(np.array([conf_matrix[i,i] for i in range(n_classes)]), counts,
          out=pct_correct_by_class, where=counts!=0)*100
plt.bar(range(n_classes), pct_correct_by_class, .75)
plt.xlim(-1,91)
plt.xticks(range(n_classes), label_encoder.classes_, rotation='vertical')
plt.savefig("correct_classified_by_species_albufera.png")

plt.figure(figsize=(15,4))
plt.title("Percent Correct, by Predicted Class (Baseline)")
pct_correct_by_predicted_class = np.zeros(n_classes)
counts = np.sum(conf_matrix,axis=0)
np.divide(np.array([conf_matrix[i,i] for i in range(n_classes)]), counts,
```

Capítulo 8. Bibliografía

[1] Parque Natural de la Albufera, Wikipedia https://es.wikipedia.org/wiki/Parque_natural_de_la_Albufera

[2] Ejemplos de Redes Neuronales Convolucionales, YouTube https://www.youtube.com/watch?v=IKB8rzGU8Wo&list=PLkgbkukKg_NpdJPhhHbemaWHNQCY-lmh1&index=22

[3] Arquitectura AlexNet https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

[4] Explicación de los hiperparámetros https://www.linkedin.com/pulse/tuneando-los-hiperpar%C3%A1metros-de-una-red-neuronal-lstm-casas-gonzalez/?originalSubdomain=es

[5] Estudio de aves protegidas en la Albufera https://www.espaisimbiosi.com/proyecto/estudio-aves-protegidas-albufera/#:~:text=En%20el%20Cat%C3%A1logo%20Valenciano%20de,charrancito%20com%C3%BAn%20(Sternula%20albifrons).