



**UNIVERSIDAD
DE LA RIOJA**

UNIVERSIDAD DE LA RIOJA
FACULTAD DE CIENCIA Y TECNOLOGÍA
TRABAJO FIN DE GRADO

**Aplicación para la clasificación de
grupos fónicos de quirópteros
mediante deep learning**

Grado en Ingeniería Informática

Autor

Ramón Sieira Martínez

Tutores

Javier Martínez de Pisón Ascacíbar
Félix González Álvarez

Logroño, 9 de Julio de 2021

Resumen

Hoy en día, se puede observar un impulso constante de innovaciones técnicas en inteligencia artificial (IA). La visión por computador es una de las ramas más estudiadas. En este TFG se aborda la clasificación de archivos de audio correspondientes a señales de ultrasonidos de quirópteros, conocidos comúnmente como murciélagos, mediante modelos de *deep learning* capaces de identificar patrones a partir de los espectrogramas generados.

La identificación y clasificación de las especies es una base fundamental de cualquier estudio sobre fauna y, por tanto, es esencial que este reconocimiento sea fiable. Con esta finalidad hemos desarrollado una herramienta especializada en determinar el grupo fónico al que pertenece una determinada grabación de los ultrasonidos emitidos por quirópteros locales. Para uno de estos grupos (especies pipistreloides) se ha precisado su pertenencia a diferentes subgrupos.

Abstract

Nowadays, we can observe a constant momentum of technical innovations in artificial intelligence (AI). Computer vision is one of the most studied branches. This dissertation deals with the classification of audio files corresponding to ultrasound signals from chiroptera, commonly known as bats, by means of *deep learning* models capable of identifying patterns from the spectrograms generated.

The identification and classification of species is a fundamental basis of any study on fauna and, therefore, it is essential that this recognition is reliable. To this end, we have developed a specialised tool to determine the phonic group to which a given recording of ultrasounds emitted by local chiropterans belongs. For one of these groups (pipistrelleid species) we have determined their membership of different subgroups.

Agradecimientos

Quiero agradecer a mi madre por el apoyo incondicional y por confiar en mí durante toda mi vida. Agradezco a Javier y Félix la formación que me han procurado durante estos meses, además de la ayuda que me han otorgado para facilitarme la elaboración de este TFG. Quiero además dar las gracias a Pablo y Romé, unos buenos amigos en los que siempre he confiado.

“No dejes que termine el día sin haber crecido un poco, sin haber sido feliz, sin haber aumentado tus sueños.” W.Whitman.

Índice general

1	Introducción	1
2	Características del Proyecto	3
2.1	Objetivos del Proyecto	3
2.2	Requisitos funcionales	3
2.3	Requisitos no funcionales	4
2.4	Alcance	4
2.5	Hipótesis y Restricciones	5
3	Estado del arte	6
4	Descripción de la solución	8
5	Resultados y Discusión	28
6	Conclusiones y vías de continuación del TFG	33
7	Organización y gestión del proyecto	35
7.1	Organigrama y matriz de responsabilidades	35
7.2	Directrices para la gestión de los cambios en el alcance	35
7.3	Directrices de comunicación entre cliente y proveedor	36
7.4	Planificación temporal	36
A	Augmentation: Filtros para Audio	41
A.1	Explicación Básica de los Filtros	41
A.2	Espectrogramas	43
B	Entregable 1: Tabla Data Augmentation	47

Capítulo 1

Introducción

Este documento corresponde con la memoria del Trabajo de Fin de Grado (TFG) del Grado en Ingeniería Informática de la Universidad de La Rioja titulado “Aplicación para la clasificación de grupos fónicos de quirópteros mediante *deep learning*”. En este proyecto se han desarrollado modelos de *deep learning* para clasificar sonidos ultrasónicos de murciélagos, con la finalidad de facilitar la labor de los biólogos a la hora de identificar especies en la realización de inventarios ambientales.

Durante esta última década, el desarrollo de modelos de clasificación de imágenes y audios; ya sea en nuestro caso en el mundo de la biología o en otros ámbitos como en medicina, medio ambiente, industria, etc.; ha permitido abordar con éxito numerosos problemas como por ejemplo: la clasificación de imágenes médicas o de biología, la conducción autónoma, la detección de plagas mediante análisis de imágenes multiespectrales, el control de calidad en productos, el reconocimiento de voz, la identificación de especies de pájaros según su canto para control poblacional, etc.

Para el desarrollo de modelos es necesario trabajar juntamente con los expertos y “extraer conocimiento” de su experiencia para generar herramientas útiles en la toma de decisiones.

Este proyecto se ha basado en la metodología KDD (*Knowledge Discovery Databases*) para el desarrollo de las actividades de éste, desde la definición de los objetivos y, la captura y preprocesado de la información, hasta el desarrollo y validación de los modelos.

Para el entrenamiento de los modelos se partió de una fuente formada por centenares de miles de grabaciones de diferentes especies de murciélagos y grabadas en diferentes localidades. Entre esas grabaciones también existían audios clasificados como ruido. Estos últimos fueron de especial interés ya que el ruido de los diferentes hábitats utilizados por las especies variaba y, para realizar modelos precisos, era importante tenerlos en cuenta.

Todos estos datos debieron ser preprocesados, es decir, clasificados de la forma más precisa posible, una de las tareas más laboriosas del *KDD*, y que fue realizada por el biólogo que suministró los archivos ya debidamente etiquetados.

A partir de estos datos, se aplicaron técnicas de minería de datos con las que obtener diferentes resultados. Los modelos desarrollados se contrastaron con otras muestras de audio, se seleccionaron los modelos de mejor calidad basándose en diferentes técnicas y métricas, y finalmente, se creó una interfaz para así facilitar el uso por el usuario.

Félix González, especializado en el estudio de los murciélagos, trabaja diferenciando las especies mediante el análisis de los pulsos de ultrasonidos que emiten durante sus desplazamientos y en la captura de presas. El proceso de análisis es muy laborioso y el objetivo final del mismo es la identificación a nivel de especie, aunque esto no es siempre posible y requiere experiencia y mucho tiempo de análisis. Francisco Javier Martínez de Pisón Ascacíbar, catedrático por la Universidad de La Rioja, comenzó a desarrollar una solución para automatizar el proceso de análisis proponiendo a Ramón Sieira, alumno de la Universidad de la Rioja, para desarrollarlo durante un Trabajo de Fin de Grado en este sentido.

La idea principal del proyecto fue clasificar en seis grupos fónicos y los cuatro subgrupos que componen uno de ellos, calculando la probabilidad que fuera de un grupo/subgrupo fónico.

Un grupo fónico es una agrupación formada por una o varias especies, independientemente de su género, que muestran características acústicas similares que pueden ser diferenciadas de las que presentan otras especies.

En la actualidad, muchos biólogos usan diferentes herramientas de análisis, e incluso ya hay disponibles programas comerciales con esa finalidad [1], pero sus resultados generan cierta controversia [2, 3] llegando a despertar cierta desconfianza. Por lo tanto, la herramienta que se pretendió desarrollar debía de ser al menos igual de precisa que estos programas.

El elevado número de grabaciones y el tiempo necesario para su análisis con varios programas diferentes, justifican el interés en que todo el proceso pudiera estar automatizado. Por ello, el obtener una herramienta precisa, funcional y fácil de usar (ya que estaría orientada a personal no técnico) podría suponer un avance relevante que facilitara el estudio acústico de quirópteros.

Capítulo 2

Características del Proyecto

2.1 Objetivos del Proyecto

El objetivo de este trabajo fue el desarrollo de una aplicación que permitiera clasificar las grabaciones de sonido obtenidas en los muestreos acústicos de quirópteros en el campo, de manera que fuera posible diferenciar los diferentes grupos fónicos en los que se pueden agrupar las especies identificadas en el noroeste de la península ibérica.

Además de las agrupaciones de especies según sus características acústicas, se prestaría especial atención a las secuencias correspondientes al grupo de los pipistreloides concretamente la especie *Pipistrellus pipistrellus* (*Ppip*). Este murciélagos es la especie más frecuente en el área y que suele representar más del 85 % de las grabaciones realizadas en todos los trabajos de muestreo acústico. A nivel técnico, el desarrollo de la aplicación se ha enfocado como un script en *voila*, es decir un cuaderno de trabajo de *Python* que funciona a través de una interfaz sin mostrar el código interno. La aplicación permitiría extraer resultados de interés para el procesado de las grabaciones, filtrando el ruido para seleccionar solo las muestras que realmente tengan señales de ultrasonidos emitidas por murciélagos y facilitando una primera clasificación de las muestras.

2.2 Requisitos funcionales

El sistema se ha dividido en tres herramientas unidas en un cuaderno de trabajo:

- La herramienta A se utiliza para clasificar los grupos y subgrupos fónicos (clases), que a partir de una carpeta de entrada, genera una carpeta de salida con tantas carpetas como clases. Además, dentro de cada carpeta de cada clase se clasifica la calidad de la muestra (archivos de sonido o “audios”) de 0 (calificación más baja) a 5 (calificación más alta) a partir del resultado de cada modelo de *deep learning* creado con diferentes arquitecturas. Para establecer esta clasificación se utilizó el criterio de “voto por modelo”, de manera que cuando un determinado modelo indica que se ha superado la probabilidad fijada para una clase determinada se considera un voto. Por ejemplo, si en un audio hay una determinada clase, por ejemplo *Ppip*, y los cinco modelos calculan para esa clase su probabilidad, y solo tres modelos superan la probabilidad de 0.83 (probabilidad por defecto que; se obtiene a partir de las pruebas

realizadas sobre los modelos), el número de votos sería 3. También se clasifican los diferentes archivos de sonido según contengan dos o más clases considerando estos archivos como múltiples y, si solo contienen una clase, considerándolos como “simple”. También se generan varios archivos en formato CSV (archivo delimitado por comas) que contendrán información de utilidad, como los votos emitidos por cada arquitectura para cada audio y también la predicción y probabilidades dentro de cada intervalo de tiempo dentro de una muestra de audio. Para clasificar las clases se utilizan cinco arquitecturas de modelos compuestos por cinco folds o pliegues. Por tanto, para un intervalo de una muestra de audio aportan información mediante su voto (“votan”) los 25 modelos.

- La herramienta B se utiliza para separar audio de murciélagos, tiene un funcionamiento similar al anterior, en este caso los modelos están especializados en clasificación binaria, es decir, para separar entre señales emitidas por murciélagos y ruido. Se genera el mismo formato de salida, además de los CSV anteriormente citados.
- La última herramienta, denominada herramienta C, permite guardar en formato PNG los espectrogramas ¹ de los archivos de una carpeta de entrada con una calidad de la imagen superior a la de entrenamiento.

2.3 Requisitos no funcionales

La aplicación ha de ser sencilla de usar, no tener una interfaz complicada y ser portable a cualquier sistema operativo con *Python* y con unas dependencias que se encontraran en el repositorio de *GitHub* [4]. Dado que la aplicación ha de realizar análisis de gran cantidad de muestras es preciso que resulte eficiente, además los resultados han de ser claros e intuitivos.

2.4 Alcance

El alcance del proyecto consistió en crear un cuaderno de *jupyter notebook* con una extensión llamada “*Voila*” para mostrar una interfaz, la cual fuera capaz de predecir las clases de murciélagos consideradas además de las dos clases que indican si el archivo de sonido contiene señales de murciélagos o solo ruido. También debería transformar una carpeta con archivos de sonido a una carpeta con los espectrogramas de cada uno de los archivos.

Las predicciones de las dos primeras herramientas se visualizarían por pantalla en la aplicación, aunque lo interesante es generar archivos CSV con la información, además de clasificar las muestras de sonido según las clases y su calidad.

La aplicación se apoyará en varios modelos: cinco modelos con cinco *folds* en cada una por cada herramienta de clasificación, en total 50 modelos para las dos primeras herramientas (A y B).

¹es el resultado de calcular el espectro de una señal por ventanas de tiempo de la misma

2.5 Hipótesis y Restricciones

Las hipótesis se ha dividido en dos grupos: generación de modelos y desarrollo de la aplicación.

GENERACIÓN DE MODELOS: Para generar modelos de calidad se deben establecer varios componentes:

- Los audios etiquetados suministrados/facilitados por un especialista en quirópteros.
- Establecer los filtros que se van a usar para realizar modelos robustos. Los filtros nos servirán para añadir diferentes variaciones en los audios de entrenamiento para que pueda realizar inferencia para cualquier audio y condición. Esto se denomina *Data Augmentation*
- Establecer la métrica que se utiliza para evaluar los modelos.
- Establecer la arquitectura para realizar los cinco *folds* por cada modelo.

DESARROLLO DE LA APLICACIÓN: Para desarrollar la aplicación se deben realizar varias tareas:

- Configurar un *notebook* de *Jupyter* con un entorno válido para hacer inferencia.
- Establecer la forma de recoger la información necesaria para poder hacer inferencia.
- Establecer la forma de ejecutar los distintos modelos, además de si el uso de las herramientas se puede realizar paralelamente o debe ejecutarse secuencialmente.

El tiempo deberá ser gestionado mediante una planificación lo más objetiva posible, llevando un seguimiento semanal de las tareas que se han ido cumpliendo, para ir comprobando que se van logrando los objetivos.

El equipo utilizado para el proyecto no va a tener la *GPU* (*Graphics Processing Unit* o Unidad de Procesamiento Gráfico) necesaria para poder elaborar algunos modelos o simplemente se necesitaría de bastante tiempo para entrenarlos, por lo que tendrán que ser ejecutados en otros equipos.

El entrenamiento de este tipo de modelos es complejo y lento, por ello, fueron desarrollados conjuntamente con Javier Martínez de Pisón. Se han utilizado gráficas del grupo EDMANS [5] concretamente un servidor HP con una GPU RTX-5000.

Capítulo 3

Estado del arte

Durante los últimos años se ha producido una revolución dentro del *deep learning*. Uno de los avances más significativos ha sido la visión por computador y en concreto al desarrollo de un tipo de red neuronal: las redes neuronales convolucionales (RNC) [6] que están especializadas en trabajar con imágenes y sonido. La importancia de estas redes viene por su capacidad de poder descifrar patrones complejos dentro de millones de imágenes. El funcionamiento de estas redes neuronales es similar a las neuronas de la corteza visual primaria de un cerebro biológico [7]. Esta sección cerebral está asociada al movimiento y la representación de objetos, además de almacenar información visual de nuestro entorno. Este tipo de redes neuronales están basadas en el perceptrón multicapa de Minsky y Papert [8] de 1969 aunque las RNC están especializadas en matrices bidimensionales y, por lo tanto, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes.

Una de las tecnologías más utilizadas para identificar eventos de sonido es el reconocimiento de sonido inteligente (ISR). Esta técnica se basa principalmente en analizar las características del audio e incorporar dicho conocimiento de percepción en máquinas o robots.

La clasificación de sonido ambiental , también conocida como reconocimiento de eventos de sonido, sirve como un paso fundamental y esencial del ISR. El objetivo principal es clasificar con precisión la clase de un sonido detectado. Los desarrollos recientes han aportado grandes mejoras en el reconocimiento automático de voz y el reconocimiento de información musical. Sin embargo, debido a las características de los sonidos ambientales, este tipo de señales no pueden describirse del mismo modo que el habla o la música, por lo tanto, para el reconocimiento del sonido ambiental es esencial desarrollar un sistema de reconocimiento de sonido inteligente eficiente. Para crear clasificadores de sonido ambiental se necesitan dos componentes fundamentales: los extractores de características y los clasificadores.

Los primeros trabajos de reconocimiento de audio comenzaron extrayendo características del audio a partir del histograma: se reconoció que las representaciones visuales de las muestras de audio contienen una valiosa información por lo que se emprendió la búsqueda de nuevas técnicas basadas en clasificación de imágenes. Los espectrogramas de Mel y los espectrogramas clásicos muestran bidimensionalmente la frecuencia respecto al

tiempo por lo que se empezaron a entrenar Maquinas Vector Soporte (SVM) basadas en la generación de espectrogramas a partir de pistas de música, de esta manera se concluyó que los modelos basados en imágenes ofrecían mejores resultados que los sistemas basados exclusivamente en características acústicas.[\[9\]](#)

Tras la aparición de equipo técnico más complejo y económico para entrenar modelos como las GPU (*Graphics Processing Unit*) y el desarrollo de las redes neuronales convolucionales, muchos investigadores han comenzado a aplicar técnicas de aprendizaje profundo a las representaciones acústicas.

Este tipo de redes neuronales también permiten el análisis de sonidos, ya que un sonido se puede representar como una imagen mediante la generación del espectrograma. Una aplicación (*app*) más utilizada en dispositivos móviles ha sido *Shazam* [\[10\]](#), una herramienta que permite identificar la canción que está grabando nuestro dispositivo móvil. Inicialmente *Shazam* no usaba redes convolucionales sino que se trabajaba con *Audio Fingerprinting* [\[11\]](#). Esta técnica se basa en extraer determinados puntos de un espectrograma y almacenarlos en una base de datos. Cuando se realizaba la grabación del audio se extraían los puntos y se comparaba con los datos almacenados, de esta manera se podría identificar la canción. Tras el desarrollo de redes neuronales convolucionales esta técnica quedó en desuso, ya que este tipo de redes neuronales permite encontrar patrones más complejos.

Un ejemplo basado en la idea de *Shazam* ha sido *BirdNet* [\[12\]](#) enfocado en el mundo de la biología. Esta aplicación permite identificar multitud de especies de pájaros y está basada en este tipo de redes neuronales.

La idea del reconocimiento de sonido inteligente (ISR) permitió establecer las bases del reconocimiento automático de voz (ASR). ASR es el proceso de convertir las señales vocales de voz en texto mediante transcripciones. El ASR juega un papel importante en la mejora de la experiencia de usuario, ya que el usuario puede realizar una variedad de aplicaciones como controlar dispositivos o la interacción con la atención al cliente. Uno de los últimos avances en ASR lo ha desarrollado *Google* con su asistente de voz que permite reconocer varios idiomas a la vez; ya sea para realizar labores de traducción o para hablar con dispositivos IoT (*Internet of Things*)

Un nuevo avance dentro del mundo de la telefonía ha sido la eliminación de ruidos ambientales para dispositivos de grabación dentro de los *smartphones*. Estos sistemas de cancelación de ruido se basan en eliminar el ruido separando el audio del interlocutor del ruido de fondo. Para ello es necesario un ISR para detectar el sonido ambiental y el habla del interlocutor.

Observando la evolución en el reconocimiento de sonido durante los últimos años y su indudable interés para la bioacústica, se desarrolló este trabajo de fin de grado (TFG), especializado en el reconocimiento de grupos fónicos de quirópteros. En este campo de estudio se han producido ya algunas iniciativas de reconocimiento basadas en aprendizaje profundo [\[13\]](#), con la intención de mejorar el reconocimiento acústico de las especies basado únicamente en redes neuronales [\[14\]](#) y árboles de decisión, y cuyos resultados generan dudas entre los especialistas [\[2, 3\]](#).

Capítulo 4

Descripción de la solución

Durante la FASE 1 del desarrollo del proyecto, uno de los objetivos principales fue asimilar información durante las reuniones con el biólogo para entender con qué grupo/subgrupo fónico de quirópteros se iban a trabajar y que tipos de audio existían. En España se han identificado 34 especies de murciélagos pertenecientes a 4 familias y 12 géneros.

Se dividen de la siguiente manera:

- Familia Rhinolophidae:

- Rhinolophus (4)

- Familia Vespertilionidae:

- Myotis (11)

- Pipistrellus (5)

- Nyctalus (3)

- Hypsugo (1)

- Eptesicus (2)

- Vespertilio (1)

- Barbastella (1)

- Plecotus (4)

- Familia Mnipteridae:

- Miniopterus (1)

- Familia Molossidae:

- Tadarida (1)

Entre paréntesis se muestra el número de especies dentro de cada género. El objetivo del proyecto fue separar grupos fónicos además del subgénero de los *PPIT*. La composición de los grupos fónicos y el subgrupo *PPIT*:

1. *RHIN*: Familia Rhinolophidae, Género *Rhinolophus*
2. *MYOT*: Familia Vespertilionidae, Género *Myotis*
3. *PPIT*:
 - (a) *PPIP*: Familia Vespertilionidae, Género *Pipistrellus*
 - (b) *Pyg-Msch*: Familia Vespertilionidae, Género *Pipistrellus*
 - (c) *Pnat-Pkuh*: Familia Vespertilionidae, Género *Pipistrellus*
 - (d) *Hsav*: Familia Vespertilionidae, Género *Pipistrellus*
4. *NYCT*: Familia Vespertilionidae, Género *Nyctalus*
5. *BARB*: Familia Vespertilionidae, Género *Barbastella*
6. *PLEC*: Familia Vespertilionidae, Género *Plecotus*

Se tenía, por tanto, las clases necesarias para clasificar cada audio de ortóptero o también denominado: “*BATS*”. Uno de los objetivos del TFG fue diferenciar correctamente las muestras de audio de un ortóptero frente a una grabación de ruido. De este modo, se generaron dos tipos de modelos: Un modelo binario para diferenciar ruido (“*NOISE*”) respecto a los quirópteros (“*BATS*”) y un segundo grupo de modelos para diferenciar las etiquetas de los grupos fónicos y el subgrupo de los *PPIT*.

Respecto a los archivos de audio se definieron 2 tipos de ruido: ruido de fondo o *backnoise* (“*BACK*”) que se refiere a ruido de fondo pero que puede contener ruido de insectos y ruido detectado como murciélagos por otros programas (“*NOISE*”) en el que no se encuentra ningún murciélago. En este momento se tenían las etiquetas para diferenciar ruido respecto a los murciélagos y las etiquetas para diferenciar los grupos fónicos y el subgrupo de los *PPIT*.

Una vez definidas las etiquetas de clasificación se generó un *dataset* con audios de cada clase. No se dispuso de muchas muestras de audio para todas las etiquetas, por lo que el *dataset* estaba desbalanceado. En la siguiente tabla se muestra el porcentaje de cada clase:

Etiqueta	Porcentaje
<i>PPIP</i>	24,85 %
<i>MYOT</i>	20,53 %
Rhin	12,49 %
Nyct	9,77 %
<i>NOISE</i>	8,36 %
<i>BACK</i>	7,77 %
Pnat-Pkuh	5,45 %
Ppyg-Mcsh	4,46 %
Hsav	2,99 %
Barb	2,35 %
Plec	0,99 %

Cuadro 4.1: Porcentaje de cada clase en el *dataset*.

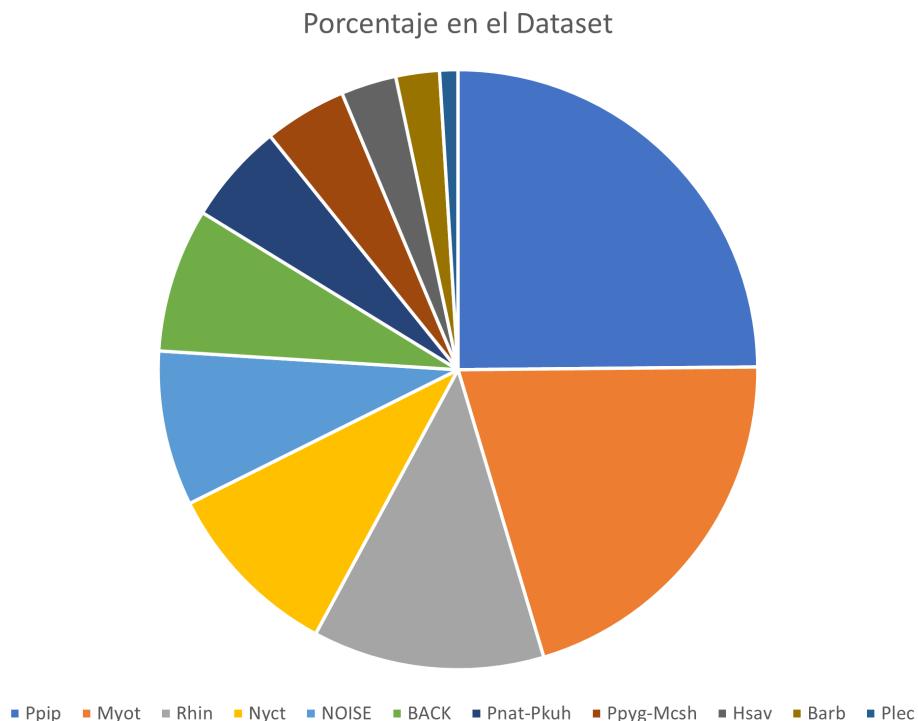


Figura 4.1: Porcentaje de cada clase en el *dataset*.

No se disponía de grabaciones de todos los murciélagos puesto que se necesitaba capturar al animal para determinar su clase y no servía simplemente con visualizar el espectrograma. Las muestras que se suministraron variaban en condiciones ambientales y temporales por lo que condicionaba la calidad de la muestra. Para el proyecto se utilizaron muestras correctamente identificadas para entrenar la red neuronal.

Una cuestión de relevancia era cómo se debían validar los modelos. Los modelos de DL pueden memorizar muy fácilmente o aprender de las imágenes algo que no se deseó y

haga que éstos no sean capaces de generalizar correctamente el problema. Un ejemplo que explica claramente este problema es una experiencia conocida de desarrollo de modelos de DL para el reconocimiento de calidad de mármoles. Durante el entrenamiento, la red neuronal se entrenaba correctamente (el error de validación era bueno), pero durante la inferencia el error de testeo era muy malo. Esto era debido a que la red neuronal había aprendido del código de barras que identificaba a la cantera, es decir asignaba la calidad del mármol al código de barras y no a la estructura del material. Al realizar inferencia con los modelos entrenados sobre mármoles que no tenían código de barras el modelo no clasificaba bien la calidad y por tanto los modelos no eran correctos. Similar a este caso, podría ocurrir que durante el desarrollo se identificará la clase a partir del ruido y no a partir de las señales emitidas por los murciélagos. Para evitar este posible error se optó por separar las grabaciones usadas en el entrenamiento y la validación según las localizaciones. Si se utilizaban muestras de una misma localización el modelo basado en redes neuronales podría identificar elementos del espectrograma por asociación, clasificando el tipo de señal según las condiciones de esa localización (ruido de fondo, ruido del micrófono usado, etc.) y no por el tipo de patrones existentes en las señales. Esto, por ejemplo, supondría que si al entrenar un modelo con audios de una clase que se pudiera identificar por estar vinculada a las condiciones ambientales (especies que utilizan preferentemente un cierto tipo de hábitat), y durante la inferencia se le suministra otra clase diferente bajo las mismas condiciones ambientales, la clase predicha podría ser incorrecta.

Dentro de esta primera fase, también se seleccionó un entorno, en este caso, para el desarrollo de *notebooks* de *Python*. Para ello se utilizó *Jupyter Notebook* alojado en un sistema operativo *Linux* y para la instalación de librerías se utilizó *Anaconda*, generando un entorno (*environment*) propio. Este entorno fue suministrado por el director del proyecto (el fichero *yaml* para simular el mismo entorno se encuentra en el repositorio *GitHub* [4]).

El proceso de entrenamiento y validación de los modelos se muestra en la figura 4.2.

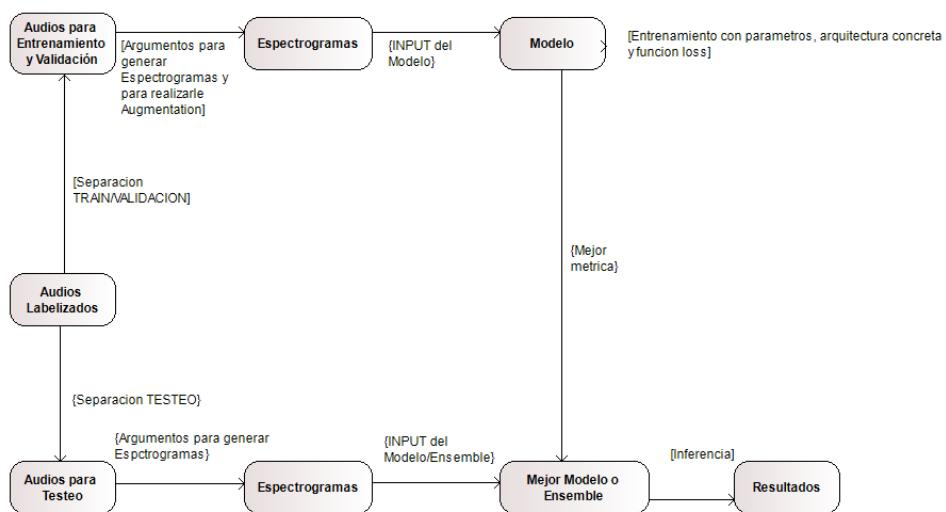


Figura 4.2: Proceso de Entrenamiento y Validación.

El proceso de creación de modelos fue lento y complejo. En primer lugar fue necesario establecer el *dataframe* para entrenar y testear los modelos. El *dataframe* contenía las localizaciones de las muestras de audio y como *target* la clase o etiqueta a la que pertenecía cada audio.

Una vez desarrollado el *dataframe*, se separaron las muestras para realizar entrenamiento/validación mediante validación cruzada estratificada con 5 folds. Para entrenar y validar se desarrolló un *DataLoader*, que permitiera realizar tres operaciones:

1. Convertir el audio a spectrograma.
2. Realizar Data Augmentation.
3. Controlar la entrada de imágenes a la red neuronal.

Para convertir el audio a spectrograma se adjuntaron diversos parámetros de configuración y la librería *librosa*. Después de numerosas pruebas, se decidió fijar la ventana de tiempo del audio en 1.5 segundos. Algunos audios no duraban exactamente 1.5 segundos, para ello se completaba el resto del audio hasta 1.5 segundos con el comienzo del audio.

Para determinar los filtros de augmentation se realizaron diversas pruebas con diversos filtros registrando los resultados y seleccionando aquellos que obtuvieron mejores resultados de validación cruzada.

El entrenamiento de la red neuronal se realizó para cada uno de los folds. Una vez ajustados los filtros de augmentation, se realizaron más de 150 modelos con más de 25 arquitecturas diferentes (DENSENET, RESNET, EFFICIENTNET, REXNET, RESNEXT, etc.) ya preentrenadas.

El uso de modelos preentrenados con la base de datos *Imagenet*, que está formada por millones de imágenes, permitió acelerar el proceso de entrenamiento. De este modo, únicamente se reentrenó el modelo con muestras de sonogramas. Se optó por los modelos preentrenados de la librería *timm*.

Para poder evaluar el entrenamiento del modelo se debe establecer la función de pérdida. Esta función de pérdida también se conoce como *loss function*.

Los modelos debían llegar al mínimo de la función de pérdida, es decir, la pérdida durante el entrenamiento debe ser mínima, para ello hay que establecer un *learning rate* o tasa de aprendizaje: el *learning rate* es un parámetro de ajuste que determina el paso en cada iteración que se realiza sobre la función de pérdida. Si se realizan pasos muy largos el modelo no converge correctamente al mínimo, si los pasos son muy cortos tarda demasiado en converger. Existen dos técnicas: fijar el *learning rate* o recalcular el *learning rate* durante el entrenamiento. Durante el desarrollo se han usado ambas técnicas aunque finalmente se optó por utilizar un *scheduler* cosine de un solo ciclo con 15 epochs de subida lineal y 45 de bajada tipo coseno.

El *scheduler* permite actualizar el *learning rate* del modelo, si no se utiliza, el modelo siempre tendrá el mismo *learning rate* durante todo el entrenamiento. Durante la finalización del entrenamiento del primer *fold*, el modelo será testeado con el conjunto de testeo del *fold* y calculando diferentes métricas. El proceso se realiza *fold* a *fold* y se evalúa el el entrenamiento con todos las predicciones de los folds de salida (*out of folds, oof*).

Para seleccionar los mejores modelos y realizar la inferencia lo más precisa posible se realizó Cross Validation estratificado. Cross Validation permite desarrollar modelos más robustos ya que es una técnica utilizada para garantizar que la partición de entrenamiento y prueba son independientes.

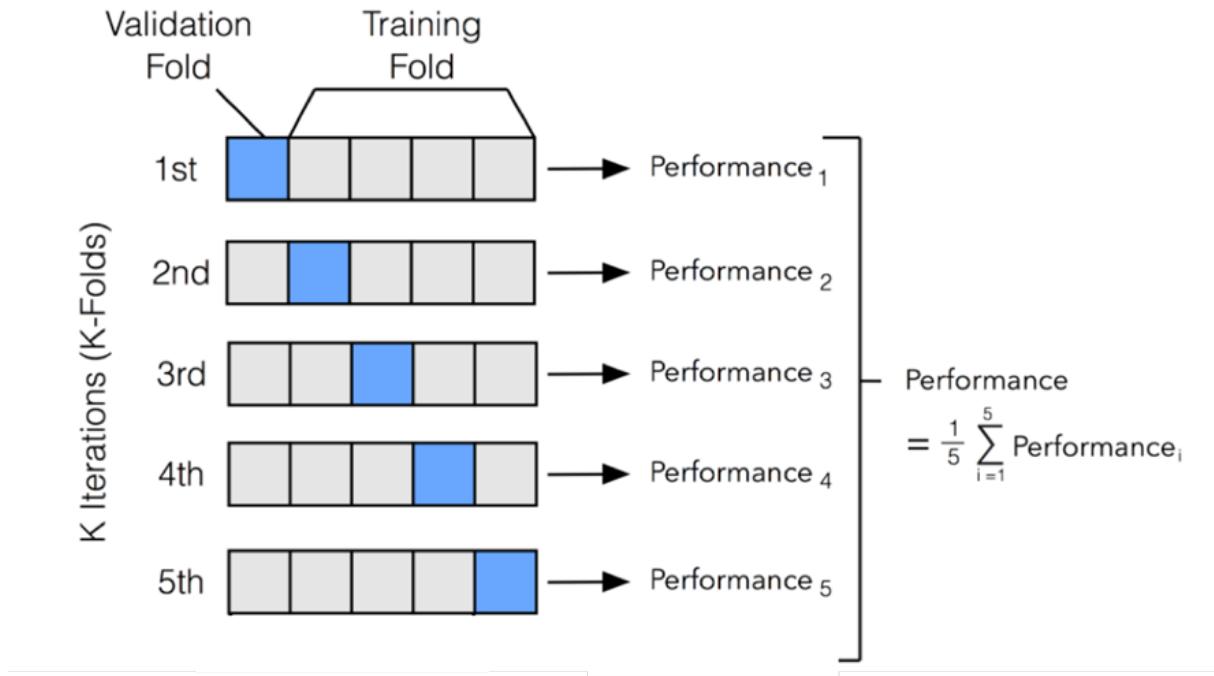


Figura 4.3: Cross Validation.

Se dividió la base de datos de entrenamiento/validación en tantos pliegues (*folds*), en este caso se realizó en cinco *folds*. Para cada fold se entrenó un modelo entrenado con cuatro partes del *dataset*, la quinta parte se usó como validación, obteniendo un entrenamiento y validación diferente para cada *fold*. Al variar las partes de entrenamiento y validación se obtuvieron los diferentes pliegues. Finalmente con los cinco *folds* se determinó la calidad del modelo. Los *folds* generados debían estar estratificados, lo que significa que debían tener el mismo porcentaje de clases en cada uno de ellos y el mismo porcentaje de validación y entrenamiento. Ver [4.3](#)

El primer paso consistió en determinar los parámetros para convertir el sonido en imagen, es decir en espectrogramas, ya que los modelos de *deep learning* aprenden de imágenes y no sonido en sí. La librería que se utilizó para generar los sonogramas fue *librosa* [15]. Para convertir una señal de sonido en un sonograma es necesario transformar una amplitud (medida en decibelios) respecto el tiempo, en una frecuencia (medida en Hertzios) respecto al tiempo [4.4](#).

Convertir audio a sonograma consiste en transformar una amplitud (medida en decibelios) respecto el tiempo a frecuencia (medida en Hertzios) respecto al tiempo. Como se observa en la figura 4.4 el audio se encuentra en parte inferior y el sonograma generado con *librosa* en la parte superior.

Un spectrograma es el resultado de calcular el espectro de una señal por ventanas de tiempo. En cada ventana se realiza el cálculo del contenido frecuencial mediante transformadas de Fourier (FFT). El proceso se realiza a lo largo de todo el audio. La suma de las transformadas de Fourier de las distintas ventanas consecutivas aporta información de la variación de la frecuencia sobre el tiempo, de esta forma se puede convertir amplitud en variación de energía o frecuencia.

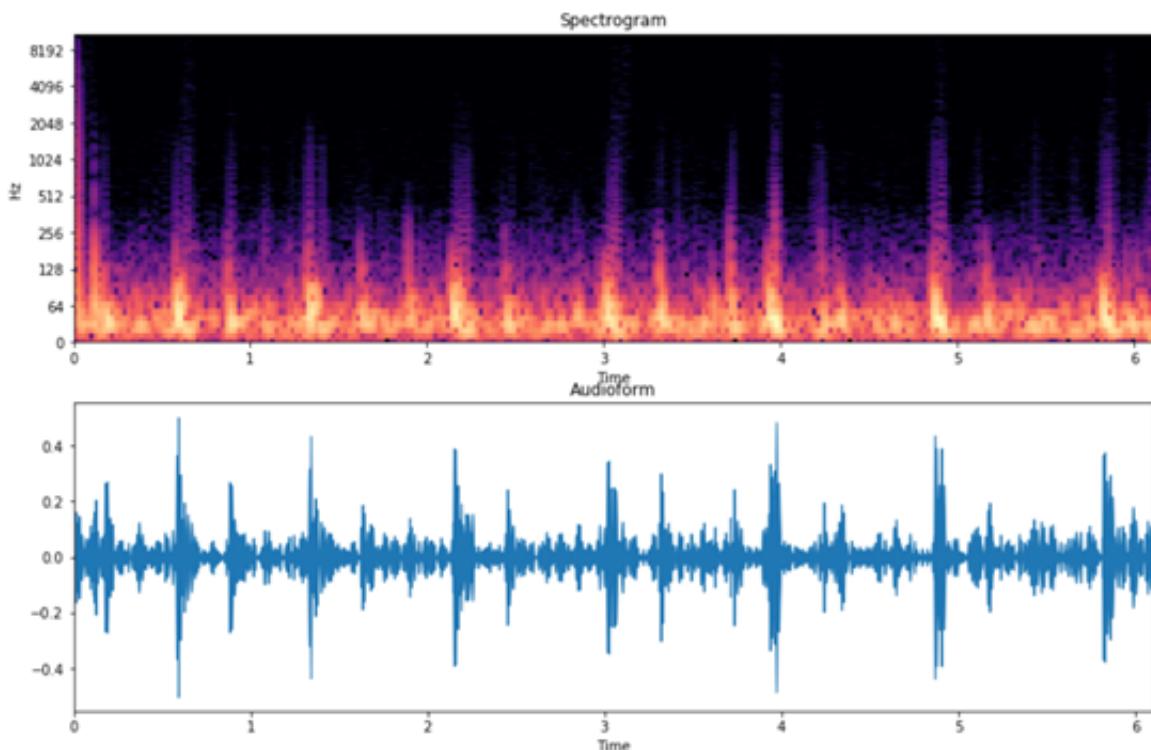


Figura 4.4: Audio (abajo) y Espectrograma (Arriba).

Para realizar la conversión fue necesario definir 4 parámetros:

1. N_FFT: Tamaño de la ventana para realizar la transformada rápida de Fourier (FFT) . Este parámetro permite definir la resolución en número de bandas de frecuencias del sonograma, correspondiente a la altura de la imagen, cuentas más series de Fourier se calculen la resolución aumentara y también el coste computacional.
2. HOP_LENGTH: este parámetro permite determinar el paso en muestras de audio de la ventana que calcula la FFT. Es decir, cuanto se mueve la ventana de FFT. Si se aumenta el paso, se reduce la anchura de la imagen, pero se pierde resolución temporal. Si se reduce, se mejora la resolución temporal pero aumenta el ancho de la imagen así como el coste computacional para tratarla.
3. F_MIN: frecuencia mínima, indica la frecuencia mínima del spectrograma.

4. F_MAX: frecuencia máxima, indica la frecuencia máxima del espectrograma.

La fijación de los parámetros se estableció para poder seleccionar los rangos de frecuencias (F_MAX y F_MIN)

En primer lugar se tuvo que debe determinar el intervalo:

- El grupo fónico que incluye especies que emiten señales a menor frecuencia es el de los *nyctaloides* (Nyct). Dentro de este grupo la especie que emite es *Tadarida teniotis* y emite en torno a los 12 kHz (Barataud, 2015), pero para generar el espectrograma se seleccionó un límite inferior algo más bajo, de 8 kHz.
- El grupo fónico que incluye especies que alcanzan mayor frecuencia son los *Myotis* (Myot) con una frecuencia de hasta 140 kHz (Barataud, 2015). La frecuencia máxima en el espectrograma está determinada por la frecuencia de muestreo utilizada para la de grabación, de manera que si las muestras tienen un *sample rate* de 192000 por el teorema de Nyquist-Shannon la frecuencia máxima es la mitad; dividir el menor *sample rate* de los audios entre dos $192000 / 2 = 96000 \text{ Hz} = 96 \text{ kHz}$ de frecuencia máxima.

Hay dos tipos de espectrogramas a elegir:

1. Espectrograma Mel [16]: es un espectrograma enfocado en representar los audios basándose en la percepción auditiva humana. Estos tipos de espectrogramas surgen de la necesidad de obviar el ruido de fondo, el volumen, el tono y otras características. Es adecuado para el reconocimiento de voz humana.
2. Espectrograma Clásico: es un espectrograma enfocado en representar los audios manteniendo las características del entorno de grabación.

En este caso, se eligió el uso de espectrogramas de tipo Clásico. Para determinar el número de transformadas de Fourier habitualmente hay que tener en cuenta dos factores: la sensibilidad y la frecuencia máxima de emisión. Una vez se definieron los parámetros para generar los espectrogramas se entrenaron los primeros modelos de *deep learning* con arquitecturas **RESNET34**. Para comprobar la calidad de los modelos se utilizaron dos métricas: *CrossEntropyLoss* y *F1-Macro*:

1. *CrossEntropyLoss* [17]: es la función de pérdida de los modelos. Medía el rendimiento de un clasificador donde el resultado pronosticado es un valor de probabilidad entre 0 y 1 que indica la distancia de cada predicción con respecto a la etiqueta real. Servía para verificar que el entrenamiento de los modelos se estaba realizando correctamente. También es conocido como *log loss*. Se calcula para cada predicción y para todo el conjunto de prueba. Durante el entrenamiento se realizarán “épocas”, es decir, cada vez que se pasen todos los datos de entrenamiento la red neuronal se actualizará según la función de pérdida y comenzará de nuevo una nueva época. También se fijaba un criterio de parada, por ejemplo si durante 20 épocas el *log loss* no se reducía la red neuronal dejaba de actualizarse y se verifica su calidad.
2. La métrica *F1-Macro* se utilizó como métrica de validación, es decir sería la métrica principal cuando se validaban los modelos, aunque también se calculaban otras métricas. Esta métrica se utiliza principalmente en problemas de clasificación múltiple

y no está afectada por el desbalanceo de clases. Su valor se encuentra entre 0 y 1, siendo 1 la máxima calidad del modelo.

Debido al limitado número de casos existentes en algunas clases, el siguiente paso consistió en añadir *Data Augmentation* en la fase de entrenamiento. Es una técnica que se utiliza para aumentar la cantidad de muestras agregando copias ligeramente modificadas de datos ya existentes. También ayuda a reducir el sobreajuste, siendo este uno de los problemas más habituales de las redes neuronales. Se denomina sobreajuste al hecho de generar un modelo ajustado a los datos de entrenamiento y que no generalice bien a los datos de testeo. El Data augmentation en muestras de audio se puede realizar en partes diferentes del proceso: se puede realizar esta técnica en el sonograma ya generado o directamente sobre el audio. En este caso se efectuó en ambas partes del proceso, utilizando diferentes filtros.

Para poder elegir los filtros se realizaron varias pruebas con un modelo de red neuronal básico RESNET34: si las métricas de evaluación mejoraban significaba que los filtros eran adecuados. En la figura 4.5 se representa el resultado de la evolución de la F1 Macro para las 49 pruebas que se realizaron.

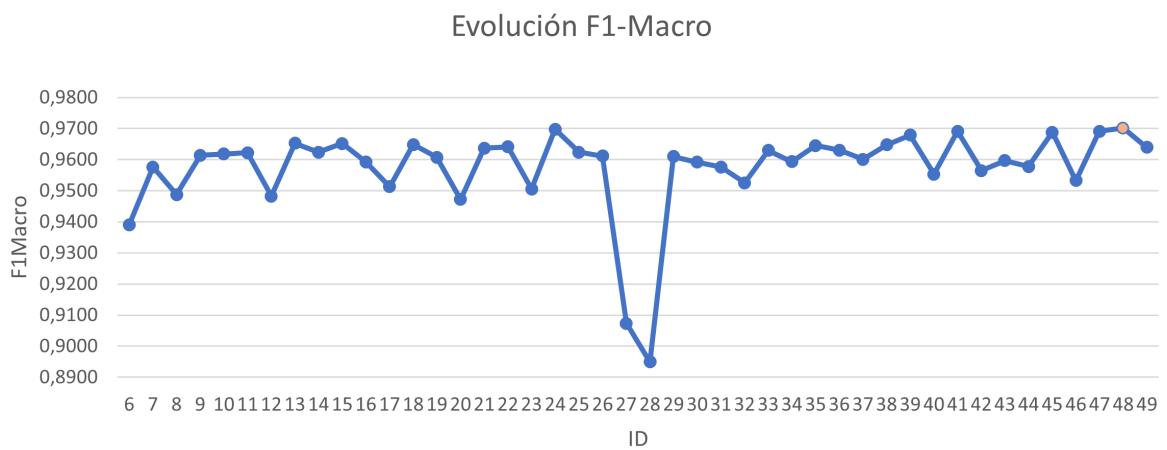


Figura 4.5: Evolución del F1-Macro durante el Augmentation.

El mejor modelo obtenido (id=48) obtuvo un valor de F1_Macro de 0.9701 aplicando 4 filtros de *data augmentation* (los resultados de todos los *data augmentation* pueden consultarse al final de la memoria en el “Entregable 1”).

En la siguiente gráfica se observa que es necesario mantener fijo el resto de los parámetros fijos para comprobar qué filtros mejoran el *F1-Score*.

Audios	Bloqueado	Bloqueado						F1-Macro
	Conversión a Sonogramas	Augmentation	Sonogramas	Entrenamiento	Modelo	Validación	Argumentos	

Cuadro 4.2: Bloqueos durante el ajuste del data augmentation.

Teniendo en cuenta los resultados obtenidos con los filtros anteriores, la conversión de audio a imagen y la elección de la métrica; se el siguiente paso consistió en seleccionar la arquitectura de la red neuronal, siendo éste el ajuste 3 dentro del desarrollo de los modelos. Sin embargo, fue necesario mantener fijo el resto de los parámetros para comprobar que modelos mejoran el *F1-score*.

Para realizar el ultimo ajuste durante el desarrollo de los mejores modelos fue necesario probar diferentes arquitecturas preentrenadas y evaluar las métricas. Algunas de las arquitecturas utilizadas han sido las siguientes:

1. RESNET34
2. RESNET50
3. DENSENET121
4. DENSENET161
5. RESNET101
6. RESNET152
7. RESNEXT50
8. EFFICIENTNETB0
9. EFFICIENTNETB1
10. EFFICIENTNETB2

Se generaron también dos métricas de validación a parte de la *F1-Macro*: AUC (Área bajo la curva curva) y ACC (*accuracy*). La métrica AUC es muy utilizada como se observa en la figura 4.6: sobre el eje *x* se calcula el porcentaje de falsos positivos, es decir, que han sido clasificados como una clase de manera incorrecta. Por ejemplo, si se fija como positivo que un archivo contenga señales de murciélagos (BATS) y negativo que se trate de ruido (NOISE; no contiene señales de murciélagos), un falso positivo sería si la muestra ha sido clasificada como murciélago cuando en realidad se trata de un ruido.

En el eje *y* se establece el número de muestras que tanto la predicción como la clase real coinciden, volviendo al ejemplo de BATS-NOISE, cuando una muestra está clasificada con predicción real BATS y el modelo la predice como BATS se dice que es un positivo verdadero. De esta manera se está representando sensibilidad respecto especificidad. Si se calcula el área bajo la curva, será un parámetro entre 0 y 1 siendo 1 el mejor valor para esta métrica. En 4.6 el modelo C que se encuentra en la esquina superior izquierda es el mejor modelo.

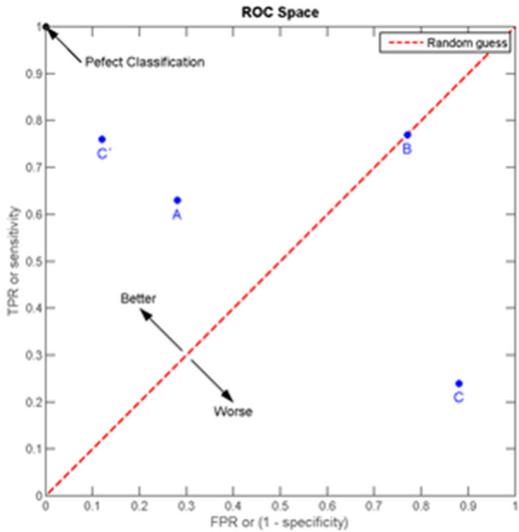


Figura 4.6: Área ROC.

El *Accuracy* es el tanto por uno de muestras que se han clasificado correctamente, es decir, se dividen las muestras clasificadas correctamente entre el número de muestras totales.

Una vez fijados los parámetros de calidad del modelo se obtuvo la siguiente tabla de resultados:

La columna Num hace referencia al identificador de la prueba. La columna denominada FP/FN hace referencia al número de falsos positivos y falsos negativos, la columna F1S hace referencia a la métrica F1-macro, cuando se le añade el término MAX y MEAN al F1 Score o a los falsos positivos y falsos negativos se hace referencia al realizar inferencia con la media de las probabilidades de los modelos o con la probabilidad máxima de los modelos, si a la métrica F1-Macro se le añade el término Pipis se refiere a la métrica F1-Macro en el grupo de los Pipistreloides. La columna ValLoss hace referencia a la métrica LogLoss anteriormente citada. La columna ACC hace referencia a la métrica Accuracy, es decir, el porcentaje de aciertos que tiene el modelo dividido entre las muestras totales evaluadas. La columna AUC hace referencia a la curva ROC anteriormente citada.

Num	FP/FN 1000	ValLoss	ACC	AUC	F1S	Descripción
87	10/67 (0.99)	0.4689	0.9611	0.9868	0.9627	Resnet50 (Murciélagos vs Noise) BCEWithLogitLoss (augmentation 0081)
Num	F1S- 5FOLDS	F1S-1K MEAN	FP/FN MEAN	F1S-1K MAX	FP/FN MAX	Descripción
88	0.9987	0.929 (0.7837)	20/30	0.942 (0.9997)	16/25	Como el 87 pero guardando mejor F1-SCORE clase 'Murciélagos'

89	0.9957	0.934 (0.7949)	13/33	0.924 (0.9981)	17/36	Como el 88 pero con Resnet101 y 20 de batch
90	0.9989	0.925 (0.8258)	24/29	0.930 (0.9997)	22/28	Como el 88 pero con Efficientnet_B0 y 32 batch
91	0.9994	0.950 (0.7784)	16/17	0.896 (0.9997)	34/40	Como el 88 pero con Efficientnet_B1 y 22 batch
92	0.9991	0.912 (0.8133)	13/47	0.826 (0.9999)	67/59	Como el 88 pero con Efficientnet_B2 y 20 batch
93	0.9994	0.931 (0.8938)	14/34	0.937 (0.9985)	22/23	Como el 88 pero con Densenet121 y 21 batch
94	0.9991	0.923 (0.7720)	20/34	0.938 (0.9998)	09/34	Como el 88 pero con Densenet161 y 10 batch
96	0.9991	0.923 (0.9213)	13/40	0.942 (0.9999)	17/24	Como el 88 pero un ResNext50 con 23 de batch
Num	F1S- 5FOLDS	F1S-1K MEAN	FP/FN MEAN	F1S-1K MAX	FP/FN MAX	Descripción
98	0.9946	0.894 (0.8889)	28/46	0.928 (0.9983)	23/28	Como el 88 pero con nueva BD incluyendo NYCT (10 clases)
99	0.9963	0.906 (0.8287)	20/45	0.929 (0.9995)	17/33	Como el 98 pero con Efficientnet_B0 y 31 batch
100	0.9951	0.853 (0.9423)	32/68	0.842 (0.9998)	46/64	Como el 98 pero con Densenet161 y 10 de batch
101	0.9955	0.891 (0.9430)	22/53	0.890 (0.99998)	44/36	Como el 98 pero con Densenet121 y 21 de batch
102	0.9959	0.881 (0.8911)	33/50		37/25	Como el 98 pero con Resnext50 y 21 de batch
103	0.9926	0.897 (0.9550)	23/48	0.918 (0.9989)	19/38	Como el 98 pero con Resnet101 y 20 de batch
104	0.9966	0.926 (0.8875)	18/34	0.867 (0.9998)	51/45	Como el 98 pero con Efficientnet_B1 y 22 batch
Num	F1 Pipis	ACC	AUC	F1S		Descripción
109	0.9762	0.9140	0.9842	0.9059		Como el 88 con RESNET34 batch=79 y 11 clases
109	0.9747	0.9191	0.9848	0.9162		Como el 88 con RESNET34 batch=79 y 11 classes 75epochs
110	0.9720	0.9170	0.9850	0.9118		Como el 0109 RESNET50 batch=30 epochs=75
111	0.9784	0.9199	0.9899	0.9167		Como el 0109 DEN-SENET121 batch=21 epochs=75

112	0.9754	0.9196	0.9910	0.9165	Como el 0109 DEN-SENET161 batch=10 epochs=75	
113	0.9786	0.9229	0.9888	0.9266	Como el 0109 RESNET101 batch=19 epochs=75	
114	0.9788	0.9231	0.9901	0.9166	Como el 0109 RESNET152 batch=13 epochs=75	
115	0.9790	0.9269	0.9872	0.9200	Como el 0109 RESNEXT50 batch=23 epochs=75	
116	0.9827	0.9335	0.9912	0.9334	Como el 0109 EFFICIENTNET_B0 batch=31 epochs=75	
117	0.9817	0.9378	0.9894	0.9355	Como el 0109 EFFICIENTNET_B1 batch=22 epochs=75	
118	0.9808	0.9341	0.9886	0.9295	Como el 0109 EFFICIENTNET_B2 batch=20 epochs=75	
119	0.9761	0.9245	0.9886	0.9272	Como el 0109 EFFICIENTNET_B0_NS batch=31 epochs=75	
120	0.9818	0.9365	0.9906	0.9312	Como el 0109 EFFICIENTNET_B1_NS batch=22 epochs=75	
121	0.9818	0.9346	0.9899	0.9343	Como el 0109 EFFICIENTNET_B2_NS batch=20 epochs=75	
122	0.9858	0.9365	0.9899	0.9338	Como el 0109 EFFICIENTNET_B3_NS batch=15 epochs=75	
123	0.9809	0.9322	0.9878	0.9311	Como el 0109 EFFICIENTNET_B4_NS batch=11 epochs=75	
Num	F1 Pipis	ValLoss	ACC	AUC	F1S	Descripción
124	0.9845	18.906	0.9346	0.9914	0.9378	Como el 0109 EFFICIENTNET_B3_NS batch=15 epochs=104
125	0.9816	18.754	0.9367	0.9898	0.9385	Como el 0109 EFFICIENTNET_B2_NS batch=20 epochs=104
126	0.9819	18.960	0.9351	0.9898	0.9383	Como el 0109 EFFICIENTNET_B1_NS batch=22 epochs=104

128	0.9830	16.516	0.9349	0.9897	0.9313	Como el 0109 SERES-NEXT50 batch=19 epochs=104
129	0.9830	16.452	0.9338	0.9886	0.9307	Como el 0109 SERES-NEXT101 batch=12 epochs=104
130	0.9837	16.243	0.9327	0.9931	0.9315	Como el 0109 DEN-SENET161 batch=10 epochs=104
131	0.9804	16.659	0.9226	0.9868	0.9199	Como el 0109 DEN-SENET121 batch=21 epochs=104
132	0.9801	16.601	0.9282	0.9882	0.9297	Como el 0109 RES-NET101D batch=19 epochs=104
133	0.9797	16.277	0.9290	0.9908	0.9285	Como el 0109 RES-NET152D batch=13 epochs=104
Num	F1S-5FOLDS	F1S-1K MEAN	FP/FN MEAN	F1S-1K MAX	FP/FN MAX	Descripción
140	0.9917	0.859 (0.9843)	14/78	0.891 (0.9998)	28/48	Como el 98 incluyendo RHIN
141	0.9951	0.925 (0.7611)	23/30	0.938 (0.9988)	21/23	Como el 98 incluyendo RHIN y fmax=120KHz batch=23
Num	F1S-5FOLDS	F1S-1K MEAN	FP/FN MEAN	F1S-1K MAX	FP/FN MAX	Descripción
142	0.9915	0.950 (0.6083)	23/13	0.919 (0.9910)	44/16	Como el 141 más DATA (NOISE/BATS) (fmax=120KHz batch=23)
Num	F1S 5FOLDS	F1S-1K MEAN	FP/FN MEAN	F1S-1K MAX	FP/FN MAX	Descripción
143	0.9922	0.953 (0.6550)	19/15	0.963 (0.9916)	17/10	Como el 142 pero con RES-NET50D batch=22
144	0.9928	0.963 (0.2853)	19/08	0.962 (0.9614)	13/14	Como el 142 pero con TF_EFFICIENTNET_B0 y batch=24
145	0.9931	0.968 (0.3759)	15/08	0.956 (0.9342)	19/13	Como el 142 pero con TF_EFFICIENTNET_B1 y batch=17
146	0.9925	0.958 (0.5603)	12/18	0.927 (0.9384)	44/11	Como el 142 pero con TF_EFFICIENTNET_B2 y batch=15

147	0.9919	0.942 (0.7382)	15/26	0.935 (0.9997)	19/27	Como el 142 pero con RESNEXT50_32X4D y batch=18
148	0.9914	0.938 (0.7243)	13/30	0.952 (0.9984)	12/22	Como el 142 pero con DEN-SENET121 y batch=16
149	0.9913	0.947 (0.6168)	18/20	0.927 (0.9988)	26/26	Como el 142 pero con DEN-SENET161 y batch=8
150	0.9922	0.971 (0.3820)	12/10	0.955 (0.9705)	22/11	Como el 142 pero con REX-NET_100 y batch=21
151	0.9920	0.971 (0.4982)	12/10	0.964 (0.9924)	15/11	Como el 142 pero con REX-NET_130 y batch=17
152	0.9921	0.967 (0.2853)	19/05	0.958 (0.9051)	26/05	Como el 142 pero con REX-NET_150 y batch=14
153	0.9922	0.971 (0.5237)	15/06	0.930 (0.9981)	18/31	Como el 142 pero con TF_EFFICIENTNET_B0 (AdvProp) batch=24
154	0.9931	0.970 (0.3193)	14/08	0.963 (0.7751)	20/07	Como el 142 pero con TF_EFFICIENTNET_B1 (AdvProp) batch=17
155	0.9922	0.955 (0.3877)	13/19	0.925 (0.9753)	21/32	Como el 142 pero con TF_EFFICIENTNET_B2 (AdvProp) batch=15
156	0.9917	0.944 (0.4775)	19/21	0.934 (0.9694)	23/24	Como el 142 pero con GLUON_SERESNEXT50 batch=14
157	0.9916	0.967 (0.4183)	14/10	0.954 (0.9802)	16/17	Como el 142 pero con GLUON_SERESNEXT101 batch=10

Cuadro 4.3: Resultados del Desarrollo de Modelos.

Para seleccionar los modelos se eligieron 5 de las mejores arquitecturas con las clases con las que se iba a trabajar en la interfaz, los modelos fueron desarrollados por el director del proyecto mientras se desarrollaban las siguientes fases.

Para realizar inferencia, se modificó el *DataLoader* para introducir las imágenes en el modelo sin realizarles augmentation. Las probabilidades calculadas por el modelo se almacenaron en un dataframe teniendo como columnas el archivo y sus correspondientes etiquetas: BATS y NOISE para el discriminador binario de ruido-murciélagos y PPIP, MYOT, RHIN, NYCT, NOISE, BACK, PNAT-PKUH, PPYG-MCSH, HSAV, BARB y PLEC para el clasificador de grupos fónicos, siendo PPIP, PNAT-PKUH, PPYG-MCSH y HSAV los subgrupos del subgrupo PPIT.

La fase 3 comenzó con el desarrollo de la primera versión basándose en el *DataLoader* anteriormente citado, realizándose un prototipo de la aplicación. El prototipo de la interfaz fue el siguiente:



Figura 4.7: Prototipo Interfaz v1.

Se valoraron dos herramientas para realizar la interfaz: *QT Creator*, una herramienta utilizada para realizar aplicaciones de escritorio en *Python*, y *Widgets de Fastai* [18]. Esta última permitía integrar pequeñas aplicaciones que están ya desarrolladas, además son sencillas de usar, aunque la mayor ventaja es su capacidad para integrarlas con cuadernos de trabajo de *jupyter* ya desarrollados. Estos *widgets* se agregan directamente sobre el cuaderno pero sin una extensión conocida como “*Voilá*” se vería el código interno del proyecto. Esta extensión se agregó directamente sobre el servidor de *Jupyter*.

El objetivo fue que se pudiera seleccionar un INPUT de audios grabados en el campo y mediante los modelos seleccionados realizar una inferencia que generara dos *dataframes*, uno para clasificar ruido y murciélagos; y otro para los grupos/subgrupos fónicos. Estos *dataframe* se guardarían como CSV en una carpeta OUTPUT.

Observando los resultados se detectaron varios problemas o aspectos que no eran del todo correctos:

- **Problema 1:** se realizaba inferencia con 25 modelos para BATS-NOISE y 25 modelos para los grupos/subgrupos fónicos sobre todos los audios, la solución fue quitar las muestras de audio de ruido antes de realizarles la inferencia sobre los modelos de los grupos fónicos, para no tener que realizar inferencia sobre archivos que no fueran murciélagos.
- **Problema 2:** Solamente se generaban CSV pero no clasificaba directamente los audios, es decir, no se clasificaba en carpetas según su etiqueta.

- **Problema 3:** Las probabilidades eran por modelo por lo que se tendría que determinar cómo calcular su probabilidad general y no por modelo.
- **Problema 4:** Los archivos que se utilizaban eran de 1.5 segundos y habría que modificarlos para usar directamente muestras de cualquier duración.

Para resolver el problema 1 se separó la funcionalidad, una sección realizaba la separación de murciélagos y ruido y , otra sección, se utilizaba para separar los grupos fónicos, de esta manera la interfaz era modular y se podría utilizar cada herramienta independientemente. Para resolver los otros tres problemas se realizaron cambios sobre el *DataLoader*: en vez de trabajar con audios de 1.5 segundos se sustituyó para trabajar con audios de cualquier duración. Esto se consiguió definiendo un paso o ventana(*window*) deslizante de 0.75 segundos:

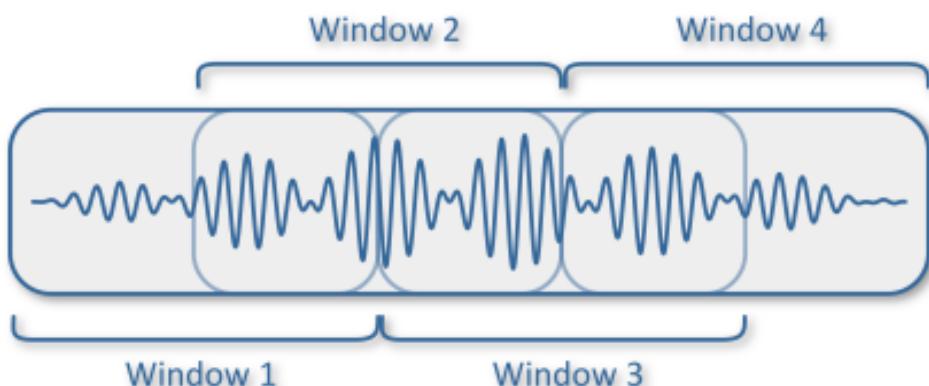


Figura 4.8: Ventana (*window*) recorriendo un audio.

De este modo, para cada ventana de 1.5 segundos se realizaba la inferencia y luego se avanzaba la misma con un paso de 0.75 segundos (ver Figura 4.8) realizando inferencia sobre las secciones de audio. El *DataLoader* cargaba los datos en el modelo para calcular las probabilidades de forma que a partir de un archivo de duración variable (duración distinta a 1,5s) recorriera esa sección de audio convirtiéndola en sonograma y, sin realizarle *augmentation*, pasárselo al modelo y obtener las probabilidades y la predicción de ese segmento. Cada sección del audio tenía una probabilidad por cada modelo.

Para calcular una probabilidad general u otro mecanismo que nos permitiera establecer a que clase pertenecía el audio, se optó por clasificar un audio dentro de una determinada categoría si su probabilidad cumplía una serie de condiciones.

Las condiciones que debía de cumplir el audio para ser clasificado en una determinada clase debían de ser las siguientes:

- La clase debía superar una determinada frecuencia de aparición, si una de las clases aparecía más de un 10 % en todas las predicciones se indicaba que superaba el umbral y el audio podía contener determinada clase. Este parámetro se debería configurar en la versión 2 de la interfaz.

- La probabilidad para esa clase debía superar un umbral, es decir, que las probabilidades para esa clase debían superar un determinado valor establecido por el usuario. Esta parámetro se debería configurar en la versión 2 de la interfaz y debería ser configurado para todas las clases.
- La calidad del audio se determinaría por los votos de los modelos, si se cumplían las condiciones anteriores se ponderaría cuántos modelos clasifican ese audio con esa clase, de forma que si 5 modelos clasificaban el audio completo como “clase 2” se señalaba que la muestra de audio contiene un “clase 2” con una calidad de 5 (máxima calidad).

Con estas condiciones podría darse el caso que en una misma muestra de audio se detectara una misma clase (“Archivo Simple”) o también diferentes clases (“Archivo Múltiple”). Para poder reconocer un audio como múltiple se añadió una condición: si se detectaban clases diferentes debían de contener el mismo número de votos. Por ejemplo: si en determinada muestra de audio se detectaban dos clases: “clase 1” con 5 votos y “clase 2” con 4 votos, se determinaba que el audio es simple con la clase 1 de calidad 5, si por el contrario se tenía la “clase 1” con 4 votos y “clase 2” con 4 votos, se determinaba que el audio era múltiple con calidad 5 para ambas clases. Si no se hubiera tenido en cuenta esta condición, el número de múltiples aumentaría y por tanto sería preciso revisar las muestras para estimar qué porcentaje real de cada clase tiene un determinado conjunto de muestras de audio.

Teniendo en cuenta la predicción final de cada muestra de audio se distribuyeron en diferentes carpetas. La estructura del OUTPUT finales tiene la siguiente estructura:

- OUTPUT: archivos CSV referentes a BATS y NOISE
 - NOISE: archivos WAV de NOISE
- BATS: archivos CSV referentes a SIMPLES y MULTIPLES
 - SIMPLES: archivos WAV simples
 - MULTIPLES: archivos WAV múltiples
- Y en cada uno de ellos (SIMPLES y MULTIPLES), los grupos fónicos
 - RHIN: archivos WAV de RHIN
 - MYOT: archivos WAV de MYOT
 - PPIT: archivos WAV de PPIT
 - * Ppip: archivos WAV con Ppip
 - * Ppyg-Msch: archivos WAV con Ppyg-Msch
 - * Pnat-Pkuh: archivos WAV con Pnat-Pkuh
 - * Hsav: archivos WAV con Hsav.
 - NYCT: archivos WAV de NYCT
 - BARB: archivos WAV de BARB

- PLEC: archivos WAV de PLEC

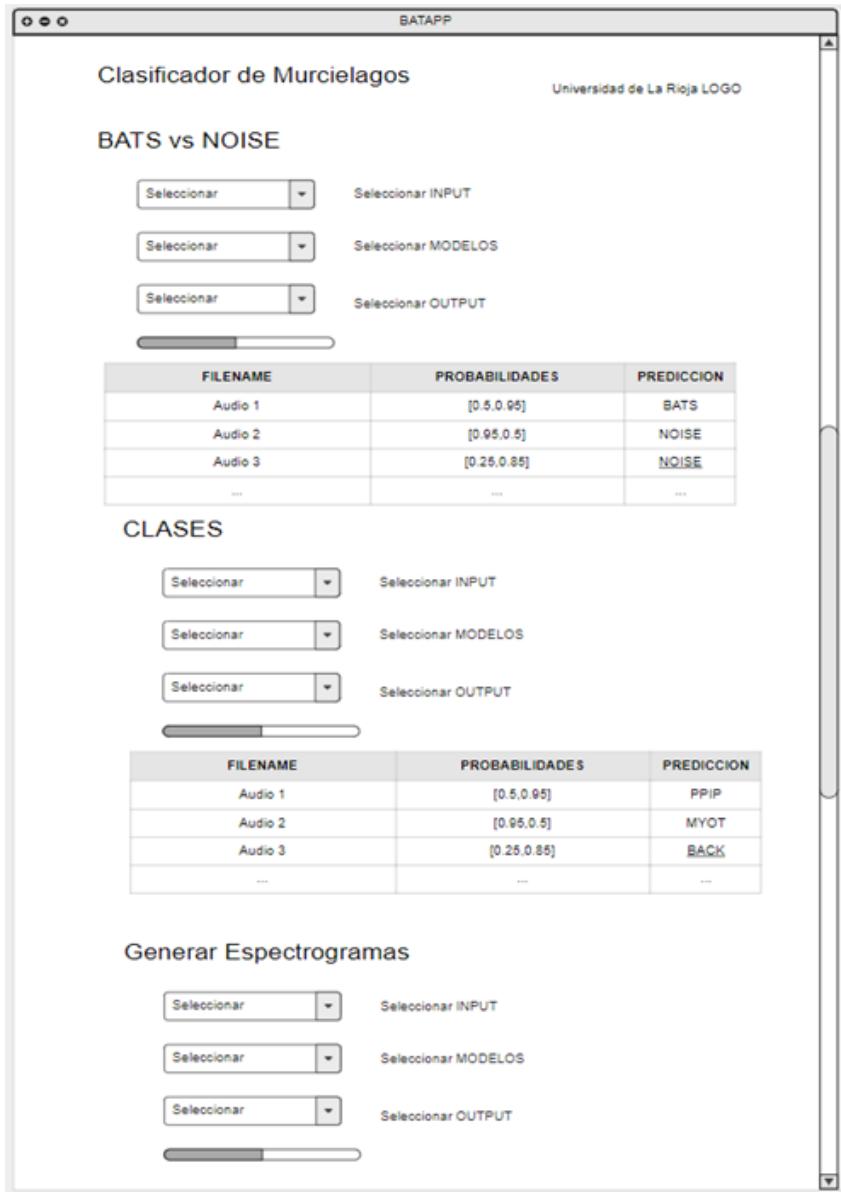


Figura 4.9: Prototipo Interfaz v2.

La FASE 4 actualizó los errores de la FASE 3 y utilizó el prototipo anterior para su desarrollo. Su desarrollo permitió detectar algunos problemas que pudieron ser corregidos.

La primera herramienta permite eliminar de un conjunto de audios de quirópteros las muestras que sean ruido de las muestras que contienen señales de ortópteros; y generar un CSV para poder realizar analíticas de datos posteriores. Permite seleccionar las muestras, los modelos que se van a usar y el output donde se encontrará la información generada y dos carpetas, una con el ruido o NOISE y otra con los archivos que contienen murciélagos (BATS). En el apartado 5 se hablará sobre la importancia de esta función.

La segunda herramienta permite clasificar entre los diferentes grupos fónicos y los subgrupos de grupo fónico PPIT. Se genera de nuevo información para realizar analíticas, además de separar las clases en carpetas, se añaden otras subcarpetas separando por calidad.

En esta fase se incorporó además una herramienta para generar sonogramas para los archivos múltiples de manera que pueda facilitar más rápidamente su revisión siendo la tercera herramienta de esta aplicación.

Capítulo 5

Resultados y Discusión

En este apartado se observan algunas secciones de los CSV generados por la aplicación para entender cómo votan los modelos y cómo finalmente se establece la predicción final para un audio. La columna **NAME_FILE** marca la muestra de audio que estamos analizando. Las columnas **BATS** y **NOISE** muestran el número de votos obtenidos por los modelos para dichas clases. Con los votos de cada modelo se puede establecer en la columna **PREDICTION** la clase final, calculada a partir del máximo de los votos de **BATS** y **NOISE**. En caso de empate clasificaremos la muestra como **BATS**. Un extracto de un CSV real se observa en la figura 5.1

NAME_FILE	BATS	NOISE	PREDICTION
LOC02_20180808_194957_711_000.wav	1	5	NOISE
LOC02_20180808_195145_324_000.wav	2	5	NOISE
LOC02_20180808_195249_386_000.wav	1	5	NOISE
LOC02_20180808_195310_386_000.wav	2	5	NOISE
LOC02_20180808_195551_386_000.wav	5	4	BATS
LOC02_20180808_195617_386_000.wav	4	2	BATS
LOC02_20180808_195704_386_000.wav	5	0	BATS

Cuadro 5.1: Extracto BATS-NOISE "Votos por modelo".

En la tabla 5.2 se observa una sección de un CSV con las probabilidades para BATS-NOISE para un intervalo de la muestra de audio calculado a partir de los 5 modelos. La columna **NAME_FILE** marca la muestra de audio que estamos analizando. La columna **INF** y **SUP** marcan el intervalo de tiempo del audio analizado. La columna **MODEL** señala el modelo con el que se ha realizado inferencia. Las columnas **BATS** y **NOISE** contienen la probabilidad calculada por el modelo para cada una de las clases. Finalmente, la columna **PREDICTION** muestra la clase cuya probabilidad sea máxima.

NAME_FILE	INF	SUP	MODEL	BATS	NOISE	PREDICTION
LOC01_20170814_194719_009_000.wav	0.0	1.5	densenet121	0.99987257	0.32474926	BATS
LOC01_20170814_194719_009_000.wav	0.0	1.5	efficientnet_b0	0.99996114	0.30473435	BATS
LOC01_20170814_194719_009_000.wav	0.0	1.5	resnext50_32x4d	0.9999995	0.105623	BATS
LOC01_20170814_194719_009_000.wav	0.0	1.5	resnet101	1.0	0.6975991	BATS
LOC01_20170814_194719_009_000.wav	0.0	1.5	resnet50	0.9999999	0.24583341	BATS

Cuadro 5.2: Extracto BATS-NOISE Probabilidades.

El apartado de grupos/subgrupos fónicos funciona del mismo modo que las salidas de los CSV de BATS-NOISE, la diferencia fundamental es que en vez de ser modelos binarios (2 clases: BATS y NOISE) son 10 clases (las clases citadas en la FASE 1 del desarrollo).

En este apartado, se desarrolló además un breve análisis de los resultados obtenidos por los modelos de *deep learning* (**DL5** **83**) desarrollados en este proyecto frente a otras 3 herramientas comerciales (**KCP**, **SBT** y **SCH**). Se tomó como predicciones reales un breve estudio de 500 muestras desarrollado por el experto en biología (**MANUAL**).

En las tablas **5.3**, **5.4**, **5.5** y **5.6** se muestran tres métricas para evaluar las herramientas analizadas y los modelos de DL, además se calcula la matriz de confusión. La primera métrica es la **precisión**: la precisión intenta responder a la pregunta ¿Qué proporción de identificaciones positivas fue correcta?. Esta métrica se calcula dividiendo los verdaderos positivos (TP) entre la suma de los verdaderos positivos (TP) más los falsos positivos (FP). Para calcularlo, para la clase negativa, se toman los verdaderos negativos (TN) y se dividen entre la suma de los verdaderos negativos (TN) más los falsos negativos (FN). La segunda métrica es el **recall, cobertura o exhaustividad**: se intenta responder a la pregunta ¿Qué proporción de positivos reales se identificó correctamente?. Esta métrica se calcula tomando los verdaderos positivos (TP) y dividiéndolos entre la suma de los verdaderos positivos (TP) más los falsos negativos (FN). Para calcularlos sobre la clase negativa, en este caso NOISE, se dividen los verdaderos negativos (TN) entre la suma de verdaderos negativos (TN) más los falsos positivos (FP). La tercera métrica es el **F1-Score**, se emplea en la determinación de un valor único ponderado de la precisión y la exhaustividad utilizando la siguiente formula:

$$F_1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

Figura 5.1: Fórmula F1-Score.

Otro elemento calculado es la **matriz de confusión**: es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases.

En la siguiente tabla observamos los resultados que nos ofrece la herramienta **KCP** comparados con los resultados del estudio del experto (**MANUAL**). Se muestran las métricas anteriormente citadas para cada clase además de la matriz de confusión.

MANUAL vs KCP		
	BATS	NOISE
precision	0,9442	0,8755
recall	0,8843	0,9397
f1score	0,9133	0,9064
	PRED_NOISE	PRED_BATS
TRUE_NOISE	218	14
TRUE_BATS	31	237

Cuadro 5.3: Resultados MANUAL vs KCP.

En la siguiente tabla observamos los resultados que nos ofrece la herramienta **SCH** comparados con los resultados del estudio del experto (**MANUAL**). Se muestran las métricas anteriormente citadas para cada clase además de la matriz de confusión.

MANUAL vs SCH		
	BATS	NOISE
precision	0,9922	0,9504
recall	0,9552	0,9914
f1score	0,9734	0,9705
	PRED_NOISE	PRED_BATS
TRUE_NOISE	230	2
TRUE_BATS	12	256

Cuadro 5.4: Resultados MANUAL vs SCH.

En la siguiente tabla observamos los resultados que nos ofrece la herramienta **SBT** comparados con los resultados del estudio del experto (**MANUAL**). Se muestran las métricas anteriormente citadas para cada clase además de la matriz de confusión.

MANUAL vs SBT		
	BATS	NOISE
precision	0,596	0,524
recall	0,556	0,5647
f1score	0,5753	0,5436
	PRED_NOISE	PRED_BATS
TRUE_NOISE	131	101
TRUE_BATS	119	149

Cuadro 5.5: Resultados MANUAL vs SBT.

En la siguiente tabla observamos los resultados que nos ofrece la herramienta **DL5_83** comparados con los resultados del estudio del experto (**MANUAL**). Se muestran las métricas anteriormente citadas para cada clase además de la matriz de confusión.

MANUAL vs DL5_83		
	BATS	NOISE
precision	1	0,9393
recall	0,944	1
f1score	0,9712	0,9687
	PRED_NOISE	PRED_BATS
TRUE_NOISE	232	0
TRUE_BATS	15	253

Cuadro 5.6: Resultados MANUAL vs DL5_83.

Como se observa en las tablas anteriores, la herramienta comercial que obtuvo los mejores resultados fue **SCH** por lo que se eligió para comparar directamente con la herramienta desarrollada en este TFG. La herramienta **SCH** falla en 14 muestras y los modelos (**DL5_83**) en 15, por lo que se obtiene un resultado similar para ambas herramientas. Uno de los objetivos de este TFG fue minimizar el número de falsos positivos, es decir, había que asegurar que BATS era realmente BATS, para ello comparamos las precisiones de ambas herramientas para la clase BATS. Como se puede observar, para esta muestra los modelos de *deep learning* no tienen falsos positivos, es decir, la precisión es 100 % para la clase BATS. Tras realizar un análisis de tiempo de computación en la misma máquina se obtiene que SCH tarda en realizar un análisis completo 59 minutos y los modelos de deep learning 27 minutos, de modo que se reduce a la mitad el tiempo de inferencia. En el caso de los modelos de *deep learning* el número de clases es inferior al número de clases de SCH, aunque teniendo en cuenta que el número de clases no interfiere a la hora de hacer inferencia, se observa que los tiempos mejoran considerablemente. Además, es importante destacar que la licencia de SCH ronda entre los 700€ y 1200€ anuales.

En la tabla 5.7 se muestran los resultados de la herramienta comercial **DL5_83** frente a los resultados del etiquetado manual del experto (**MANUAL**) para la clasificación de grupos fónicos. Se calculan de nuevo la matriz de confusión y las tres métricas anteriormente utilizadas (*precision*, *recall* y *f1 score*). Existen 3 estrategias de promediado: *Micro*, *Macro* y *Weighted*, para calcular las 3 métricas anteriores.

- *Micro* o Micropromediado: todas las muestras contribuyen por igual a la métrica promediada final.
- *Macro* o Macropromediado : todas las clases contribuyen por igual a la métrica promediada final.
- *Weighted* o Ponderado promedio : la contribución de cada clase con el promedio es ponderado por su tamaño.

Dado que el *dataset* de murciélagos esta desbalanceado¹ se optó por utilizar las métricas *macro* para la *precisión*, el *recall* y el *f1-score*.

Para realizar la tabla 5.7 no se tuvieron en cuenta los resultados de las tablas de BATS-NOISE anteriores, puesto que los resultados de partida eran diferentes. No se muestran

¹existe gran diferencia entre el número de instancias en cada clase.

los resultados de los modelos de grupos fónicos para NOISE, únicamente se muestran los datos para grupos fónicos, ya que los modelos de clasificación de grupos fónicos permiten clasificar también NOISE, aunque son menos precisos que los específicos para diferenciar BATS de NOISE. Las herramientas SCH y DL5_83 detectaron 256 y 232 murciélagos respectivamente de la muestra de 500 audios.

		MANUAL vs DL5_83				
precision		0,940291				
recall		0,871014				
f1score		0,885414				
TRUE		PRED				
		MYOT	NYCT	PLEC	PPIT	RHIN
		MYOT	5	0	0	1
		NYCT	0	12	2	9
		PLEC	0	0	6	0
		PPIT	0	0	0	196
		RHIN	0	0	0	1

Cuadro 5.7: Resultados MANUAL vs DL5_83 para Grupos Fónicos.

En la tabla 5.8 se muestran los resultados de la herramienta comercial **SCH** frente a los resultados del etiquetado manual del experto (**MANUAL**) para la clasificación de grupos fónicos.

		MANUAL vs SCH				
precision		0,709773				
recall		0,652035				
f1score		0,667546				
TRUE		PRED				
		MYOT	NYCT	PLEC	PPIT	RHIN
		MYOT	12	0	1	1
		NYCT	0	15	2	10
		PLEC	0	1	6	0
		PPIT	0	0	0	205
		RHIN	0	0	0	1

Cuadro 5.8: Resultados MANUAL vs SCH para Grupos Fónicos.

A pesar de que SCH detecta un mayor número de murciélagos que los modelos de grupos fónicos de DL5_83, los resultados son mejores para DL5_83: la *precisión* es un 14 % superior, la cobertura o *recall* es un 22 % superior y finalmente el f1-score es casi un 22 % mas alto. Se observa que los modelos de *deep learning* ofrecen una ventaja significativa respecto a las herramientas comerciales al analizar *datasets* desbalanceados.

Capítulo 6

Conclusiones y vías de continuación del TFG

El desarrollo del Trabajo de Fin de Grado ha permitido mejorar la discriminación de las muestras de audio mediante modelos de aprendizaje profundo (*deep learning*). El primer hito completado ha sido la separación de ruido frente a las señales de quirópteros. El ruido en las grabaciones realizadas habitualmente en el campo representa por término medio más del 85 % del total de archivos que es preciso analizar (calculado sobre grabaciones realizadas en 10 localidades), pudiendo llegar a alcanzar volúmenes de varios centenares de miles de archivos. Teniendo en cuenta los resultados obtenidos en este trabajo parece posible reducir considerablemente los tiempos de procesado e incrementar la fiabilidad del reconocimiento de las señales que realmente pertenecen a quirópteros. Un segundo hito es el reconocimiento de los grupos fónicos en los que se pueden agrupar las emisiones ultrasónicas de las especies existentes en el área y, además, en el de los subgrupos que es posible reconocer en uno de ellos (pipistreloides; PPIT). Considerando que entre el 90 y 95 % de las señales estudiadas corresponden a este grupo, el desarrollo de los modelos permite agilizar considerablemente el proceso de análisis de las muestras de audio, facilitando centrar la atención de los especialistas en el examen de muestras de especies menos frecuentes. No obstante, el desarrollo de esta herramientas aún debe avanzar en el reconocimiento de otros subgrupos fónicos y uno de los problemas principales: el reconocimiento de archivos múltiples, es decir, con señales de más de un grupo fónico o especie. Durante el desarrollo del Trabajo de Fin de Grado se ha intentado abordar este problema, aunque para solucionarlo es preciso establecer criterios de clasificación más exhaustivos y procesar nuevas muestras, una labor que se tiene pensado abordar de cara a la presentación de un póster sobre el tema en un congreso sobre estudio y conservación de murciélagos que se celebrará a finales de año en Murcia.

Un campo de aplicación en el que mayor utilidad podría tener este tipo de herramienta basada en el aprendizaje profundo es en el de la elaboración de estudios ambientales en el sector de las energías renovables, concretamente de parques eólicos. En la actualidad, el 21.9 % de la energía producida en España tiene su origen en la producción de los algo más de 1200 parques instalados [19] y las previsiones de crecimiento estiman que España ha de duplicar la potencia total de estas instalaciones energéticas durante los próximos 9 años (Plan Nacional Integrado de Energía y Clima, PNIEC 2021-2030). Sin embargo, uno de los mayores impactos ocasionados por este tipo de instalaciones energéticas es

debido a la colisión de aves y de quirópteros con los aerogeneradores en funcionamiento [20]. La mortalidad registrada, mayor en quirópteros que en aves [21], es la primera causa de mortalidad no natural de este grupo de mamíferos en el mundo [22] y está provocando descensos significativos en las poblaciones de numerosas especies. Los protocolos establecidos para el estudio de los quirópteros en los parques eólicos [23] [24] requieren el procesado de grandes volúmenes de grabaciones de ultrasonidos (en algunos casos miles de horas de grabaciones), por lo que agilizar los procesos de análisis y la fiabilidad en el reconocimiento de las señales de las especies puede contribuir de forma notable a incrementar la fiabilidad de las evaluaciones.

Capítulo 7

Organización y gestión del proyecto

Este apartado servirá para controlar y gestionar el proyecto:

7.1 Organigrama y matriz de responsabilidades

En el proyecto intervienen tres personas con funciones bien diferenciadas: director del trabajo de fin de grado (TFG), Francisco Javier Martínez de Pisón, el biólogo especialista en quirópteros y codirector, Félix González, que actúa también en calidad de experto y proyectista y el desarrollador, Ramón Sieira Martínez.

El director tiene como finalidad ser el encargado de controlar y gestionar el proyecto por lo que será consultado a lo largo de su desarrollo debiendo ser informado de los avances y posibles dificultades que pudieran surgir durante el mismo al igual de los posibles problemas. El experto suministra datos correctamente clasificados para comenzar la puesta en marcha del proyecto y facilita indicaciones tanto al director como al proyectista para que la herramienta cumpla con las necesidades requerida. El proyectista es el encargado de llevar a cabo el desarrollo, elaborando las tareas necesarias de acuerdo con las indicaciones y propuestas tanto del director del proyecto como del experto.

7.2 Directrices para la gestión de los cambios en el alcance

Durante la realización del proyecto se pueden llevar cambios en el alcance, por lo que estableció un protocolo:

1. Tanto el proyectista como el director podrían proponer cambios al proyecto.
2. Se evaluaría si dichos cambios fueran viables o no, según los plazos previstos en la planificación, el trabajo realizado y el trabajo por realizar.
3. Si el cambio fuera aceptado por parte del director y el proyectista se realizaría la redefinición del proyecto.

7.3 Directrices de comunicación entre cliente y proveedor

En este caso, el biólogo especialista en quirópteros actuó como experto y cliente al mismo tiempo, por lo que tanto el director como el projectista pudieron ponerse en contacto directamente. Las tres personas estaban comunicadas sin intermediarios.

7.4 Planificación temporal

El proyecto se estructuró siguiendo 4 fases de desarrollo:

FASE 1: Estudio previo, selección de software y entorno. En esta fase se realiza un estudio previo del problema durante 25 horas, recopilando información del problema, se efectúan algunas reuniones y se establecen nomenclaturas para las diferentes clases. Una vez descrito el problema, se busca el mejor software para el desarrollo de modelos y para el desarrollo de la interfaz. Finalmente también se tiene en cuenta el trabajo previo realizado y las diferentes herramientas ya desarrolladas. Esta primera fase está compuesta por las dos tareas:

1.1 Estudio previo

1.2 Selección de Software y Entorno

FASE 2: Con un desarrollo de 105 horas, esta fase es el núcleo del proyecto. En ella tiene lugar la creación de la base del notebook de desarrollo, el proceso de Augmentation, la generación de modelos y la inferencia. A partir de los entornos y el *feedback* de los expertos se generan los modelos predictivos que ayudaran a clasificar rápidamente los grupos fónicos descritos. Esta segunda fase está compuesta por cuatro tareas:

2.1 Creación de la Base del *notebook*

2.2 *Augmentation*

2.3 Generación de Modelos

2.4 Inferencia

FASE 3: Generación de la primera versión de la interfaz. En esta fase, en la que se invirtieron unas 30 horas de duración, se diseñó un *front end* para usar la inferencia a partir de los mejores modelos generados. Para su desarrollo se tuvieron en cuenta directrices de usabilidad. Solo se hace inferencia para diferenciar archivos con señales de sonido de murciélagos de los que han de ser considerados simplemente como ruido. Solo está compuesta por una tarea:

3.1 Interfaz v1

FASE 4: Generación de la segunda versión de la interfaz. En esta fase, de unas 70 horas, se rediseñó una nueva versión para usar la inferencia a partir de los mejores modelos generados, se agregaron nuevas funcionalidades, como es la clasificación de las diferentes clases (los grupos y subgrupos fónicos y las clases de ruido), y además una herramienta para generar espectrogramas de los audios de quirópteros que tengan varias clases. Esta fase está compuesta por una tarea:

4.1 Interfaz v2

Identificador	Tarea	Inicio	Fin	Duración
1.1	Estudio Previo	22/02/2021	01/03/2021	15
1.2	Selección de Software y Entorno	02/03/2021	14/03/2021	10
2.1	Creación Base del <i>notebook</i>	16/03/2021	29/03/2021	15
2.2	Augmentation	30/03/2021	14/04/2021	25
2.3	Generación de Modelos	30/03/2021	06/07/2021	50
2.4	Inferencia	15/04/2021	22/04/2021	15
3.1	Interfaz v1	23/04/2021	10/05/2021	30
4.1	Interfaz v2	11/05/2021	11/06/2021	65
5.1	Reuniones			20
6.1	Memoria TFG	22/02/2021	07/07/2021	55
HORAS TOTALES				300

Cuadro 7.1: Tareas del Proyecto.

El diagrama de Gantt:

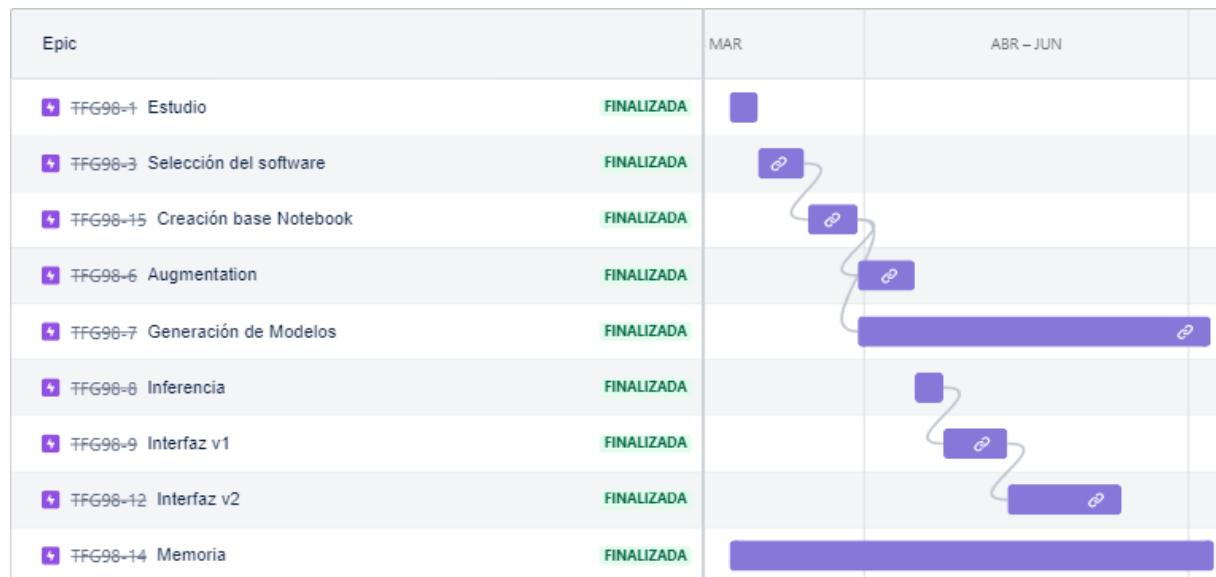


Figura 7.1: Diagrama de Gantt.

Los entregables del proyecto son los siguientes:

Id	Entregable	Fecha de Entrega
1	Tabla Filtros para Augmentation	14/04/2021
2	<i>notebook</i> con la Interfaz v1	10/05/2021
3	<i>notebook</i> con la Interfaz v2	11/06/2021
4	Tabla Modelos Generados	06/07/2021
5	Memoria TFG	07/07/2021

Cuadro 7.2: Entregables del proyecto.

El calendario del proyecto con los hitos y los días de trabajo es el siguiente:

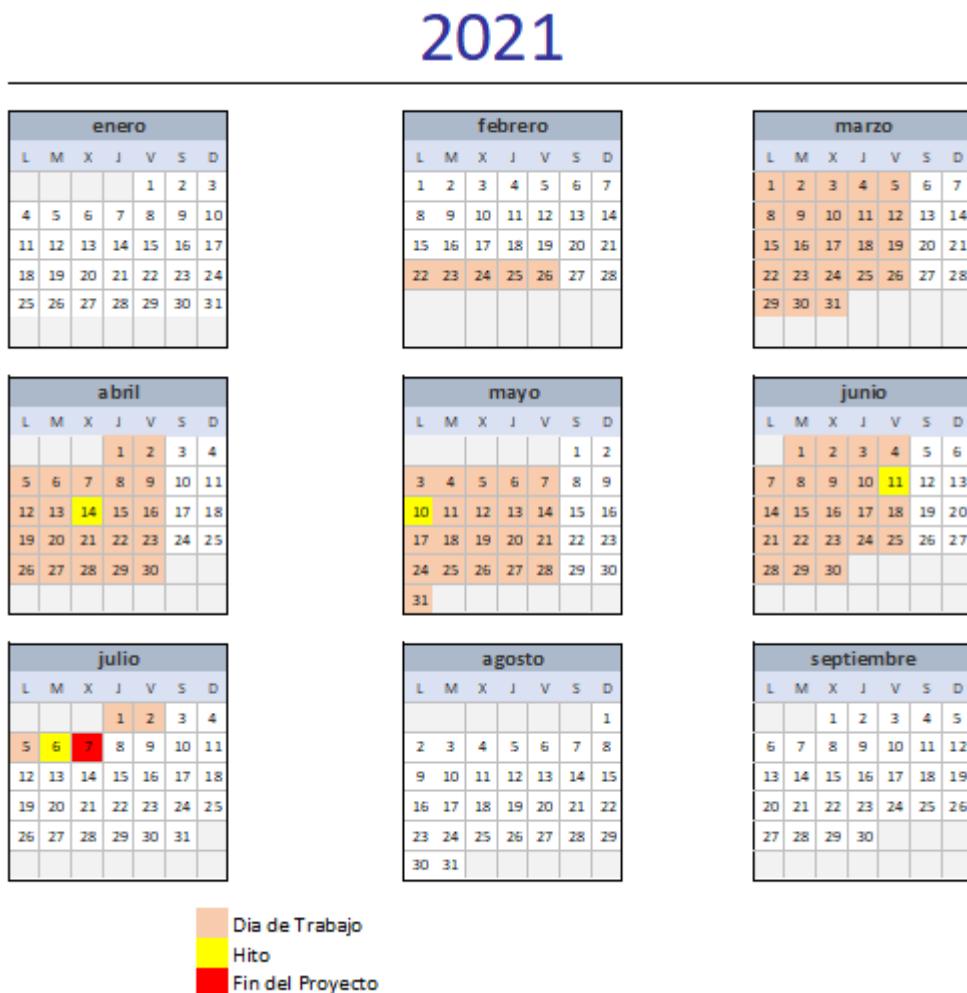


Figura 7.2: Calendario del Proyecto.

Bibliografía

- [1] *Kaleidoscope*. URL: <https://www.wildlifeacoustics.com/products/kaleidoscope-pro> (vid. pág. 2).
- [2] Cliff Lemen, Patricia Freeman, Jeremy White y Brett Andersen. “The Problem of Low Agreement among Automated Identification Programs for Acoustical Surveys of Bats”. En: *Western North American Naturalist* 75 (ago. de 2015). DOI: [10.3398/064.075.0210](https://doi.org/10.3398/064.075.0210) (vid. págs. 2, 7).
- [3] Danilo Russo, Leonardo Ancillotto y Gareth Jones. “Bats are still not birds in the digital era: Echolocation call variation and why it matters for bat species identification”. En: *Canadian Journal of Zoology* 96 (sep. de 2017). DOI: [10.1139/cjz-2017-0089](https://doi.org/10.1139/cjz-2017-0089) (vid. págs. 2, 7).
- [4] *Repositorio del TFG de Ramón Sieira*. URL: <https://github.com/rasieira/tfg-rasieira> (vid. págs. 4, 11).
- [5] *Grupo EDMANS*. URL: <https://edmans.webs.com/> (vid. pág. 5).
- [6] *Red Neuronal Convolucional*. URL: https://es.wikipedia.org/wiki/Red_neuronal_convolucional (vid. pág. 6).
- [7] *Corteza Visual*. URL: https://es.wikipedia.org/wiki/Corteza_visual (vid. pág. 6).
- [8] *Perceptrón multicapa*. URL: https://en.wikipedia.org/wiki/Multilayer_perceptron (vid. pág. 6).
- [9] Loris Nanni, Gianluca Maguolo, Sheryl Brahnam y Michelangelo Paci. “An Ensemble of Convolutional Neural Networks for Audio Classification”. En: (jul. de 2020) (vid. pág. 7).
- [10] *Shazam*. URL: <https://www.shazam.com/es> (vid. pág. 7).
- [11] *Huella digital acústica*. URL: https://en.wikipedia.org/wiki/Acoustic_fingerprint (vid. pág. 7).
- [12] *BirdNet*. URL: <https://birdnet.cornell.edu/> (vid. pág. 7).
- [13] Oisin Mac Aodha, Rory Gibb, Kate E. Barlow, Ella Browning, Michael Firman, Robin Freeman, Briana Harder, Libby Kinsey, Gary R. Mead, Stuart E. Newson, Ivan Pandourski, Stuart Parsons, Jon Russ, Abigel Szodoray-Paradi, Farkas Szodoray-Paradi, Elena Tilova, Mark Girolami, Gabriel Brostow y Kate E. Jones. “Bat detective—Deep learning tools for bat acoustic signal detection”. En: *PLOS Computational Biology* 14.3 (mar. de 2018), págs. 1-19. DOI: [10.1371/journal.pcbi.1005995](https://doi.org/10.1371/journal.pcbi.1005995). URL: <https://doi.org/10.1371/journal.pcbi.1005995> (vid. pág. 7).
- [14] Charlotte L. Walters, Robin Freeman, Alanna Collen, Christian Dietz, M. Brock Fenton, Gareth Jones, Martin K. Obrist, Sébastien J. Puechmaille, Thomas Sattler, Björn M. Siemers, Stuart Parsons y Kate E. Jones. “A continental-scale tool for acoustic identification of European bats”. En: *Journal of Applied Ecology* 49.5 (2012), págs. 1064-1074. DOI: <https://doi.org/10.1111/j.1365-2664.2012.02182.x>.

- eprint: <https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2664.2012.02182.x>. URL: <https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2664.2012.02182.x> (vid. pág. 7).
- [15] *Libreria para Python Librosa*. URL: <https://librosa.org/doc/latest/index.html> (vid. pág. 13).
- [16] *Espectrograma Mel*. URL: <https://es.wikipedia.org/wiki/MFCC> (vid. pág. 15).
- [17] *Log Loss*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html (vid. pág. 15).
- [18] *FastAI en GitHub*. URL: <https://github.com/fastai/fastai> (vid. pág. 23).
- [19] Asociacion Empresarial Eolica. *Anuario Eólico 2020*. Jul. de 2020. URL: <https://www.aeeolica.org/comunicacion/publicaciones-aee/anuarios/4264-anuario-eolico-20-toda-la-informacion-del-sector-en-el-ano-2019> (vid. pág. 33).
- [20] Eva Schuster, Lea Bulling y Johann Köppel. “Consolidating the State of Knowledge: A Synoptical Review of Wind Energy’s Wildlife Effects”. En: *Environmental Management* 56.2 (abr. de 2015), págs. 300-331. DOI: [10.1007/s00267-015-0501-5](https://doi.org/10.1007/s00267-015-0501-5) (vid. pág. 34).
- [21] Taber Allison, Jay Diffendorfer, Erin Baerwald, Julie Beston, David Drake, Amanda Hale, Cris Hein, Manuela Huso, Scott Loss, Jeff Lovich, M Strickland, Kathryn Williams y Virginia Winder. “Impacts to wildlife of wind energy siting and operation in the United States”. En: *Issues in Ecology* 21 (sep. de 2019), págs. 1-24 (vid. pág. 34).
- [22] Thomas J. Oshea, Paul M. Cryan, David T.s. Hayman, Raina K. Plowright y Daniel G. Streicker. “Multiple mortality events in bats: a global review”. En: *Mammal Review* 46.3 (ene. de 2016), págs. 175-190. DOI: [10.1111/mam.12064](https://doi.org/10.1111/mam.12064) (vid. pág. 34).
- [23] Felix Gonzalez, Juan Tomas Alcalde y Carlos Ibanez. “Directrices básicas para el estudio del impacto de instalaciones eólicas sobre poblaciones de murciélagos en España”. En: *Barbastella* 6.Especial (mar. de 2013), págs. 1-31. URL: http://secemu.org/wp-content/uploads/2016/12/barbastella_6_num_esp_2013_red.pdf (vid. pág. 34).
- [24] Luisa Rodrigues, Lothar Bach, Marie-Jo Dubourg-Savage, Branko Karapandža, Diana Rnjak, Thierry Kervyn, Jasja Dekker, Andrzej Kepel, Petra Bach, J. Collins, C. Harbusch, Kirsty Park, Branko Micevski y J. Minderman. *Guidelines for consideration of bats in wind farm projects Revision 2014*. Mar. de 2015. ISBN: 978-92-95058-30-9 (vid. pág. 34).

Apéndice A

Augmentation: Filtros para Audio

A.1 Explicación Básica de los Filtros

Para entender que hacen los filtros utilizados, se exponen cada uno de ellos comparado con un audio original sin ruido, con muy buena calidad e identificado sin ambigüedad como *Pipistrellus Pipistrellus (Ppip)*.

Esta es la representación del audio, en el eje X se representa el tiempo y el en el eje Y el valor del audio.

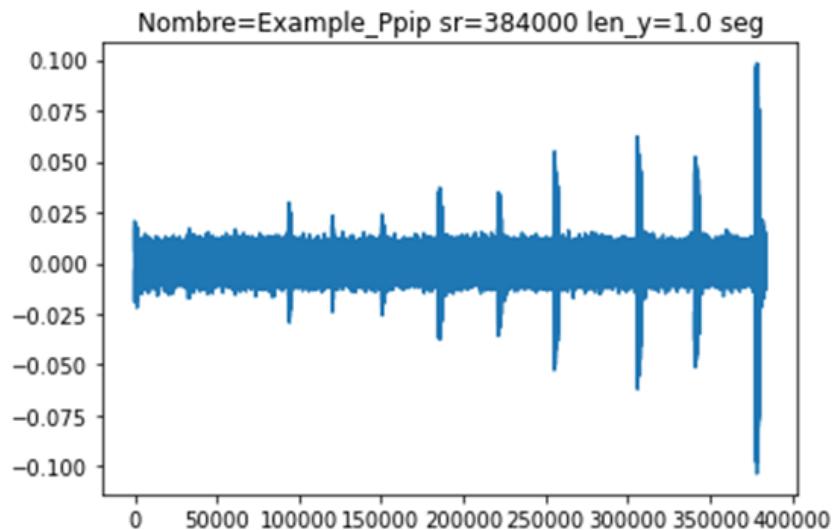


Figura A.1: Ejemplo de muestra.

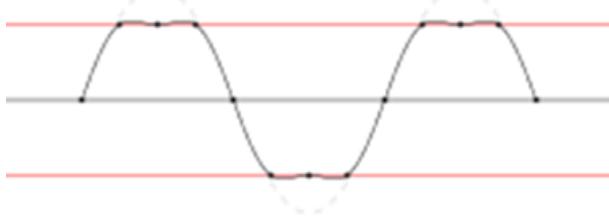


Figura A.2: Filtro ClippingDistortion.

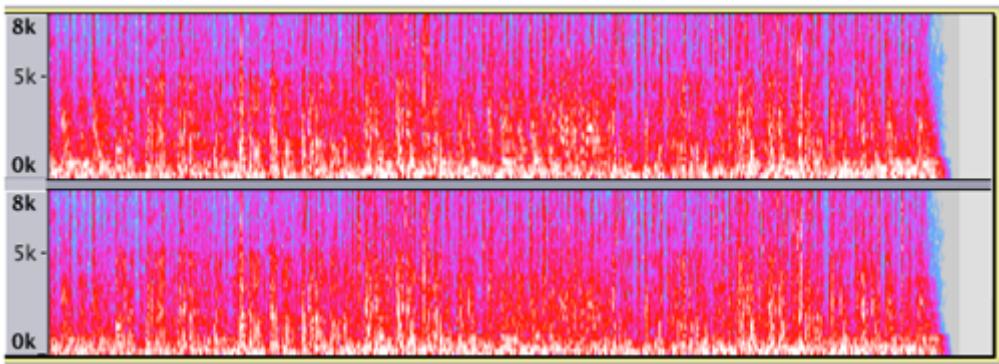


Figura A.3: Filtro Gain.

Si se aumenta el ruido mediante SNR se obtiene que las ráfagas tienen distorsión, en este caso se ha aumentado a 2 para ver el efecto. Esto se consigue mediante la ampliación de la relación señal-ruido, que consiste en distorsionar el audio mediante un factor.

El factor permite cambiar entre audio sin ruido a audio con ruido, cuanto mayor es el factor (expresado en decibelios) más ruido contiene la muestra. Teóricamente se define como la proporción existente entre la potencia de la señal que se transmite y la potencia de ruido que se distorsiona. Añadir este tipo de ruido permite que el modelo aprenda mejor ya que en las muestras de validación se utilizan muestras que contienen ruido de diferentes localizaciones.

Con el filtro *PitchShift* se obtienen dos resultados: se modifica el tono, esto significa que las ráfagas pueden ser más aplanadas. La segunda parte que permite añadirle un *Shift*, es decir cortarle fragmento del espectrograma final y añadirlo al principio, de esa manera la muestra es diferente a como se grabó

El filtro ClippingDistortion permite eliminar parte de la onda y, por tanto, eliminar picos dentro de cada ráfaga.

La ganancia (*Gain*) es la magnitud que expresa la relación entre la amplitud del audio final con respecto a la amplitud del audio de entrada. Esto quiere decir que se podrá aumentar la amplitud final del audio aumentando la ganancia. El filtro aumenta por tanto la amplitud de la señal del audio con respecto el audio original. En la figura A.3 se observa un ejemplo de ello.

Los parámetros considerados para las imágenes de muestra son:

- **AddGaussianSNR** ($p=pvalue$, $\text{max_SNR}=2$)
- **PitchShift** ($\text{min_semitones}=-8$, $\text{max_semitones}=8$, $p=pvalue$)
- **PitchShift** ($\text{min_semitones}=-8$, $\text{max_semitones}=8$, $p=pvalue$)
- **Shift** ($p=pvalue$, $\text{min_fraction}=-0.25$, $\text{max_fraction}=0.25$)
- **ClippingDistortion** ($\text{max_percentile_threshold}=100$, $p=pvalue$)
- **Gain** ($p=pvalue$, $\text{min_gain_in_db}=-100$, $\text{max_gain_in_db}=1$)

A.2 Espectrogramas

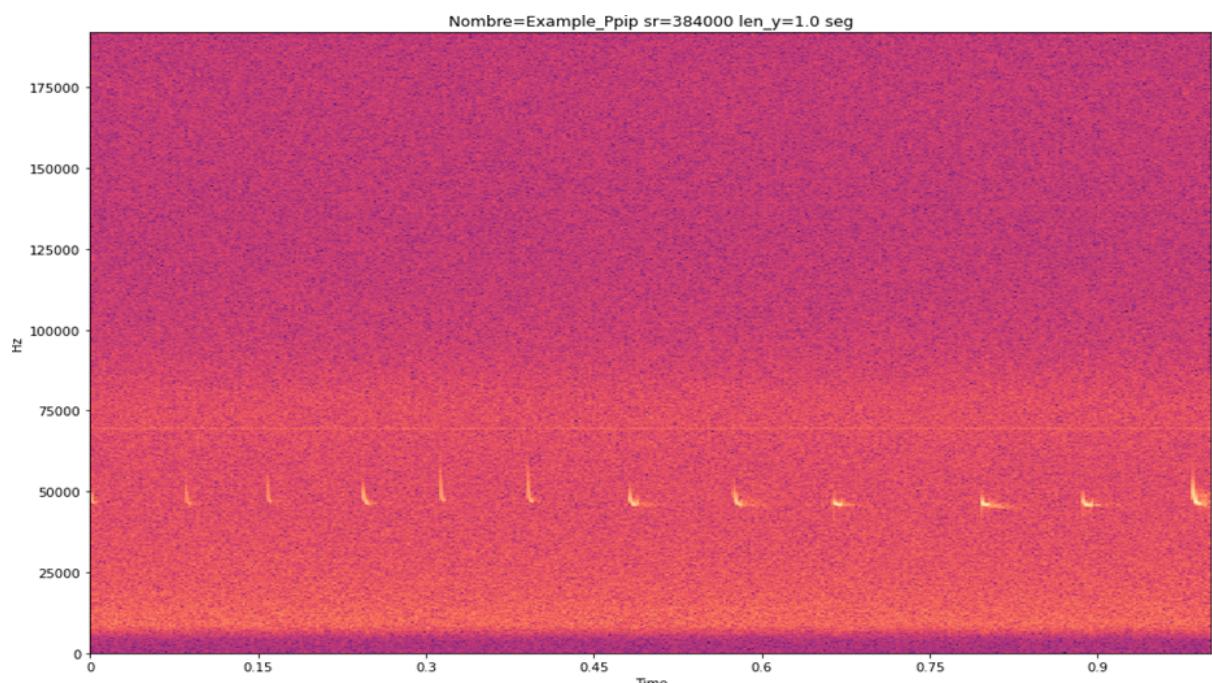


Figura A.4: Espectrograma Sin Filtro.

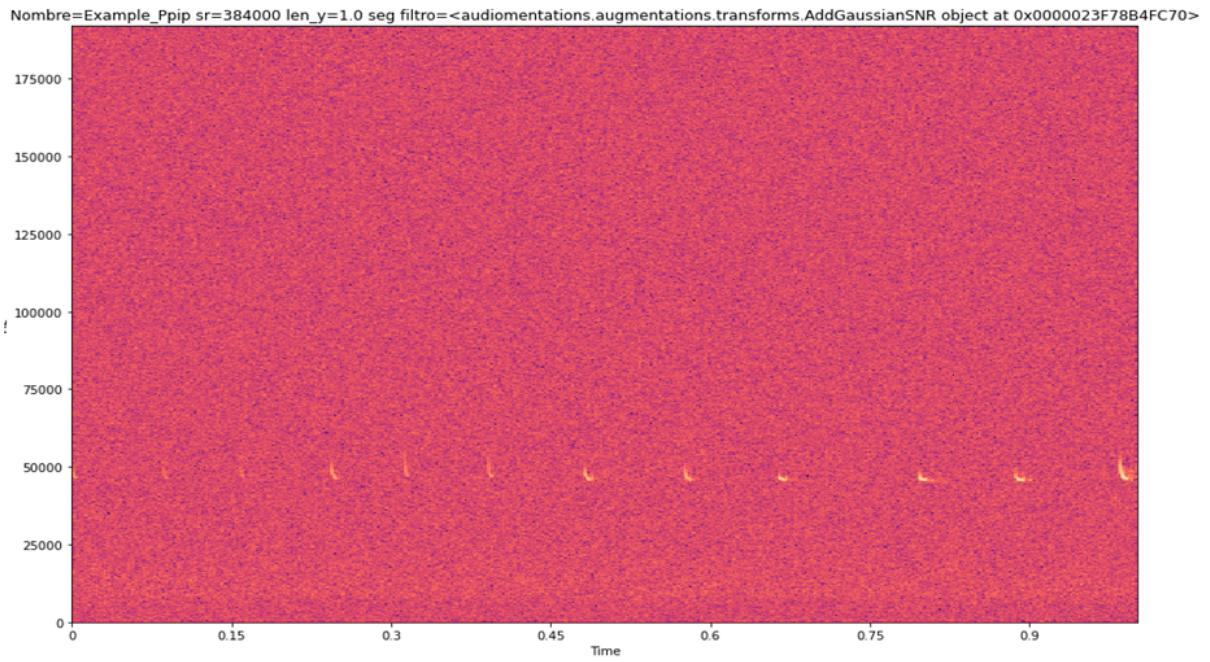


Figura A.5: Espectrograma con GaussianSNR.

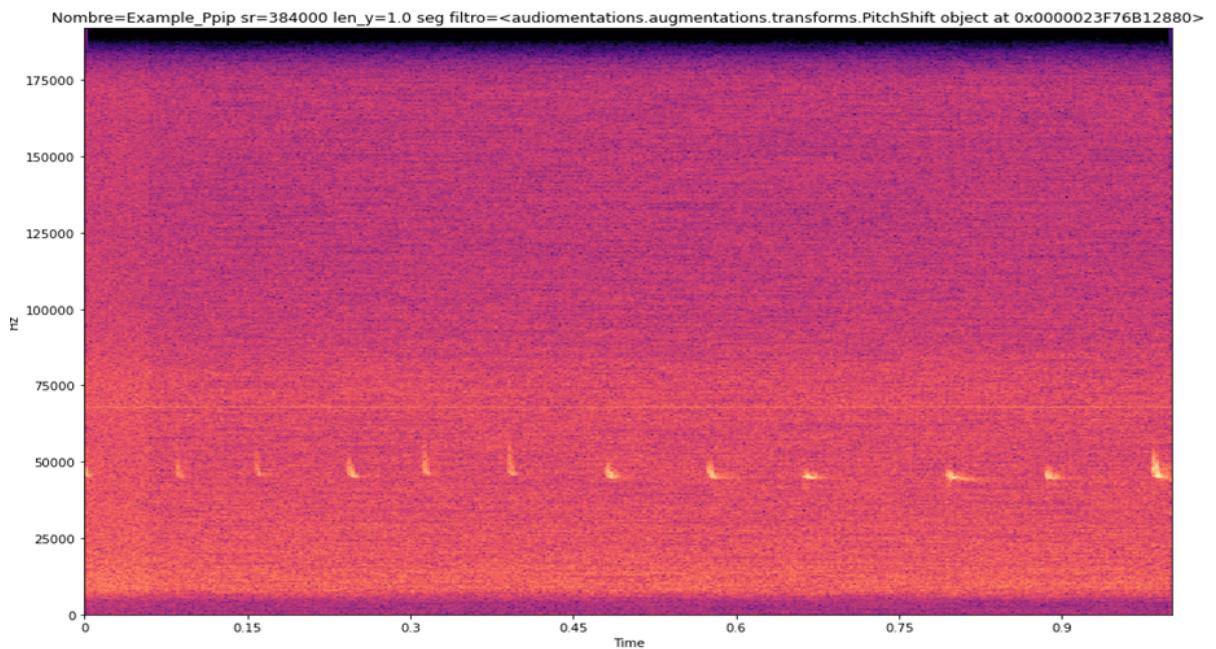


Figura A.6: Espectrograma PitchShift.

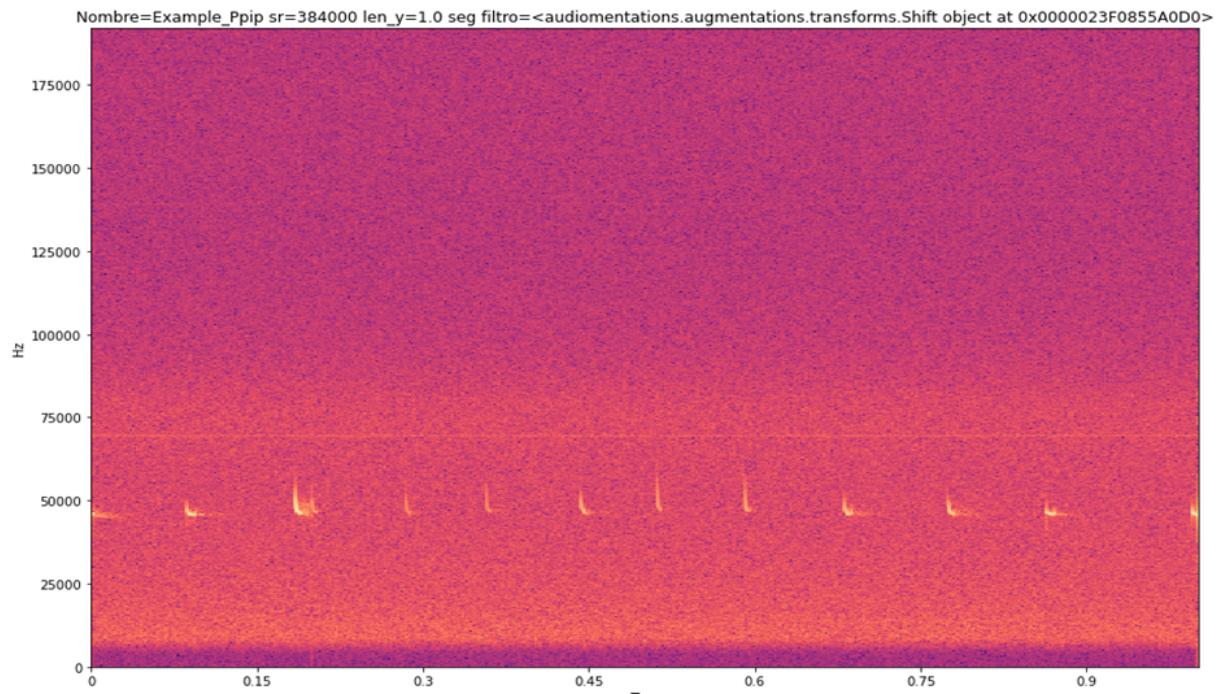


Figura A.7: Espectrograma Shift.

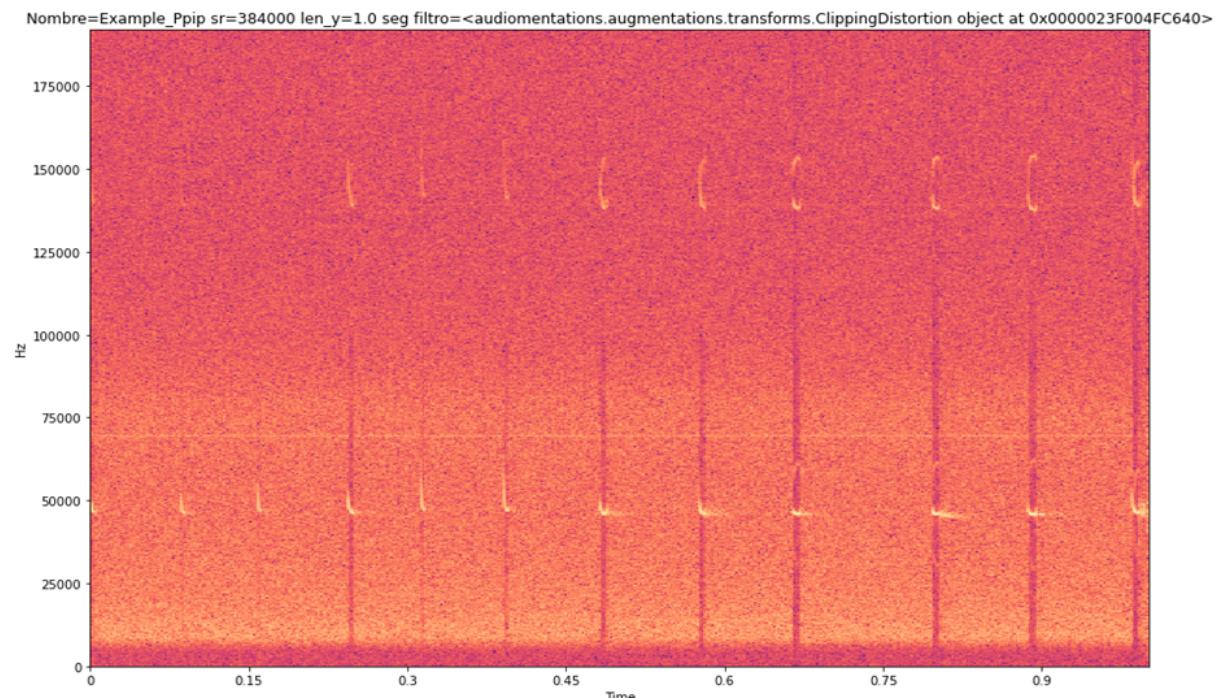


Figura A.8: Espectrograma ClippingDistortion.

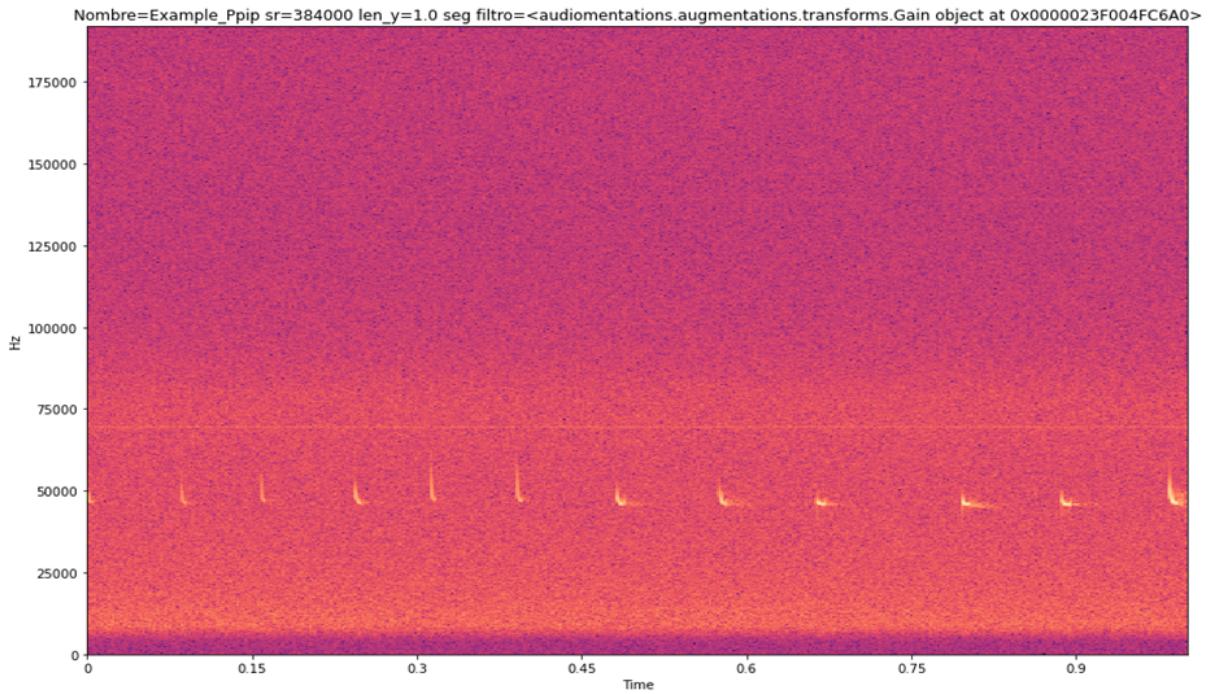


Figura A.9: Espectrograma Gain.

Apéndice B

Entregable 1: Tabla Data Augmentation

ID	ValLoss	ACC	AUC	F1	Descripción
1	1,3087	0,9747	0,9973	0	5 Folds con nuevas clases (6+noise), ventana Hamming, incluida matriz de resultados por clase
2	1,2609	0,9673	0,9963	0	Igual que 20 pero con 1 solo canal (10 sg más rápido por epoch)
3	1,2768	0,9642	0,9952	0	Un solo canal con la base de datos fijada 'df_orig_10marzo.csv'
4	1,2822	0,9741	0,9966	0	Igual que 22 pero con Ramon (gaussianNoise=0.1, FreqMask(min=0,max=0.2,p=0.1),Shift(p=0.01)
5	1,2776	0,9723	0,9971	0	Shift(p=0.1)
6	1,289	0,9655	0,994	0,939	Igual que 22 sacando métric F1-Score
7	1,2719	0,9753	0,9969	0,9576	Igual que 23 sacando métrica F1-Score (casa)
8	1,2567	0,9655	0,9956	0,9488	Igual que 22 sin filtros pero guardando modelo con mejor F1-Macro (BASE)
9	1,2541	0,9735	0,9971	0,9614	Shift(min_fraction=-0.5, max_fraction=0.5) con mejor F1-Macro
10	1,2574	0,9772	0,9971	0,9618	AddGaussianSNR(max_SNR=1.0)
11	1,2663	0,9753	0,9962	0,9622	Gain(min_gain_in_db=-12, max_gain_in_db=12)
12	1,2514	0,9655	0,9956	0,9483	ClippingDistorsion(max_percentile_threshold=1)
13	1,2737	0,9766	0,9955	0,9653	Shift(min_fraction=-0.25, max_fraction=0.25) con mejor F1-Macro
14	1,2664	0,9741	0,997	0,9624	Shift(min_fraction=-0.75, max_fraction=0.75) con mejor F1-Macro
15	1,2558	0,9778	0,9987	0,9652	ClippingDistorsion(max_percentile_threshold=25)
16	1,2621	0,9753	0,9971	0,9593	AddGaussianSNR(max_SNR=0.50)
17	1,254	0,9686	0,9966	0,9513	Gain(min_gain_in_db=-24, max_gain_in_db=24)
18	1,2491	0,979	0,9961	0,9649	ClippingDistorsion(max_percentile_threshold=10)

19	1,2704	0,9741	0,9954	0,9608	AddGaussianSNR(max_SNR=0.75)
20	1,2651	0,9636	0,995	0,9472	Gain(min_gain_in_db=-8, max_gain_in_db=8)
21	1,2655	0,9778	0,9973	0,9637	ClippingDistorsion(max_percentile_threshold=40)
22	1,271	0,9766	0,9956	0,9641	AddGaussianSNR(max_SNR=1.25)
23	1,2748	0,9649	0,9966	0,9506	Gain(min_gain_in_db=-18, max_gain_in_db=18)
24	1,2503	0,9784	0,9977	0,9698	ClippingDistorsion(max_percentile_threshold=32)
25	1,268	0,9747	0,9965	0,9624	ClippingDistorsion(max_percentile_threshold=28)
26	1,2628	0,9803	0,9959	0,9612	AddGaussianSNR(max_SNR=1.50)
27	1,3099	0,9242	0,9794	0,9074	PitchShift(min_semitones=-4, max_semitones=4)
28	1,3335	0,9069	0,9705	0,895	PitchShift(min_semitones=-8, max_semitones=8)
29	1,2659	0,976	0,9966	0,961	Gain(min_gain_in_db=-15, max_gain_in_db=15)
30	1,2752	0,9747	0,9955	0,9592	AddGaussianSNR(max_SNR=1.35)
31	1,2697	0,9716	0,9966	0,9576	Gain(min_gain_in_db=-10, max_gain_in_db=10)
32	1,3045	0,9655	0,9904	0,9525	PitchShift(min_semitones=-1, max_semitones=1) con 52 epochs
33	1,2459	0,9784	0,9981	0,963	Shift(min_fraction=-0.25, max_fraction=0.25) + Gain(min_gain_in_db=-12, max_gain_in_db=12) con mejor F1-Macro
34	1,2769	0,9766	0,9951	0,9594	PitchShift(min_semitones=-0.25, max_semitones=0.25) con 32 epochs
35	1,2681	0,976	0,9968	0,9645	Gain(min_gain_in_db=-12, max_gain_in_db=12) + ClippingDistorsion(max_percentile_threshold=32) con 52 EPOCHS
36	1,2788	0,9766	0,9912	0,9631	40 % de ClippingDistorsion(max_percentile_threshold=32) y 20 % de Gain(min_gain_in_db=-12, max_gain_in_db=12),
37	1,2683	0,976	0,9961	0,9601	AddGaussianSNR(max_SNR=1.25) + ClippingDistorsion(max_percentile_threshold=32)
38	1,2624	0,979	0,9971	0,9648	4 filtros (shift, gaussian, distorision, gain) al 75 %
39	1,273	0,9784	0,9958	0,9679	4 filtros (shift, gaussian, distorision, gain) al 100 %
40	1,2915	0,9679	0,9954	0,9553	4 filtros (shift, gaussian, distorision, gain) al 85 %
41	1,281	0,9809	0,9967	0,9691	4 filtros (shift, gaussian, distorision, gain) al 90 %
42	1,2812	0,9735	0,9958	0,9565	4 filtros (shift, gaussian, distorision, gain) al 95 %
43	1,3054	0,9741	0,9948	0,9597	4 filtros (90 %) + SpecAugment (timewarp=5, ti-memask=16, frqmask=16) num masks = 2, 2
44	1,4181	0,9739	0,9948	0,9578	4 filtros (90 %) con clase BACK incluida

45	1,4143	0,9786	0,9962	0,9688	4 filtros (90 %) + SpecAugment (timewarp=5, timemask=32, frqmask=32) num masks = 2, 2
46	1,4431	0,9702	0,9946	0,9533	4 filtros (90 %) con clase BACK incluida + BATCH=50 y 32 epochs
47	1,4306	0,9791	0,9958	0,9691	FP16 Batch=50 Epochs=60 4 filtros (90 %) + SpecAugment (timewarp=5, timemask=32, frqmask=32) num masks = 2, 2
48	1,4257	0,9796	0,9962	0,9701	Batch=50 Epochs=60 4 filtros (90 %) + SpecAugment (timewarp=5, timemask=40, frqmask=40) num masks = 2, 2
49	1,4171	0,9754	0,9962	0,964	4 filtros (90 %) + SpecAugment (timewarp=5, timemask=24, frqmask=24) num masks = 2, 2