Universidad Zaragoza
1542

Trabajo Fin de Máster

Desarrollo y evaluación de un sistema de identificación de audios no musicales ni habla, mediante el uso de Modelos de Aprendizaje Profundo

Development and Evaluation of a System for Non-Musical and Non-Speech Audio Identification, utilizing Deep Learning Models

Autor

Paula Royo Emperador

Director

José Ramón Beltrán Blázquez

Máster Universitario en Ingeniería de Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2024

Desarrollo y evaluación de un sistema de identificación de audios no musicales ni habla, mediante el uso de Modelos de Aprendizaje Profundo

RESUMEN

La identificación y clasificación de sonidos de habla y música ha avanzado y evolucionado mucho en los últimos años. Sin embargo, los eventos de sonido, que no se incluyen en esas dos categorías, siguen suponiendo un gran desafío. Lo cual se debe tanto a la variabilidad impredecible de los sonidos, como a la complejidad del etiquetado de los datos. Para abordar el problema se plantea un sistema compuesto de dos redes neuronales, un codificador y un clasificador, que en conjunto deben ser capaces de caracterizar y clasificar los audios de entrada. La investigación se divide en tres partes: tratamiento de bases de datos, adaptación del modelo de codificación y desarrollo de la clasificación de audio.

Para la extracción de características, se propone emplear un modelo de codificación y decodificación preentrenado, empleando el codificador para la caraterización de los audios. Para ello se plantea el uso de modelos basados en '*SoundStream*', que busca realizar representaciones eficientes de audio, mejorando así tanto la calidad del sonido como la capacidad de clasificación.

El codificador se conectará a un clasificador neuronal, que es el encargado de analizar las características extraídas y asignarles etiquetas correspondientes a eventos específicos.

En conjunto, se busca una arquitectura de codificación y clasificación que proporcione una solución innovadora al desafío planteado en este contexto específico.

Development and Evaluation of a System for Non-Musical and Non-Speech Audio Identification, utilizing Deep Learning Models

ABSTRACT

The identification and classification of speech and music sounds has made significant progress in recent years. However, the classification of sound events outside of these two categories remains a significant challenge, primarily due to the unpredictable variability of these sounds and the complexity of data labeling. To address this issue, a system consisting of two neural networks, an encoder and a classifier, is proposed to effectively characterize and categorize input audio. This research is divided into three main stages: database processing, adaptation of the encoding model, and development of an audio classification framework.

For feature extraction, a pre-trained encoding-decoding model is proposed that uses the encoder to the encoder to characterize the audio data. In particular, models based on *SoundStream* are considered, since this model is designed to learn efficient audio representations, thus improving both sound quality and classification performance.

The encoder is connected to a neural classifier, which is responsible for analyzing the extracted features and assign labels corresponding to specific events.

Overall, the goal is to develop an encoding and classification architecture that that provides a robust solution to the challenge posed in this specific context.

Índice

Capítulo 1

Introducción y objetivos

1.1. Motivación

El uso de herramientas de inteligencia artificial con señales de audio se lleva explotando desde hace un tiempo, estos desarrollos se han enfocado especialmente en el contexto de habla y música. Sin embargo, hay otros tipos de audio, que no encajan dentro de estas dos categorías, que son menos predecibles y cuyo estudio está en continuo desarrollo, el estudio de estos se clasifica como *SED* (Sound Event Detection) [1].

La detección de eventos de sonido (*SED*), persigue saber, a partir de los sonidos que se producen, qué está pasando y cuándo, con lo que se obtendrá información sobre el o los objetos que provocan el sonido y en qué instante temporal. Se incluye en *SED* todo aquel sonido que no pertenezca al habla ni a la música, lo que se denomina '*nonspeech, nonmusic*'. Esto se debe a que la detección de habla o música puede limitarse, debido a la definición una base finita, como pueden ser los fonemas en el habla, o las notas musicales en la música, que se usarán como diccionario para la identificación de los sonidos, en el caso de *SED* no existe la posibilidad de diseñar este diccionario.

Con el fin de resolver este problema se ha tratado de adaptar las soluciones que funcionan para otros tipos de audios, imágenes o incluso texto, lo cual se conoce como aprendizaje por transferencia. Sin embargo, la diferente naturaleza de sonidos que se incluyen en esta categoría presenta un gran reto, al cual se suma la dificultad de la forma en la que aparecen en entornos naturales, con duración y frecuencia de aparición inciertas. Esto provoca que las características de los sonidos sean muy diversas, se pueden encontrar sonidos breves y transitorios, como un golpe, o largos y armónicos, como el sonido de un aire acondicionado.

Por todo lo anterior, el SED se estudia en grupos limitados, como puede ser 'sonidos urbanos', o 'sonidos del hogar', para que la cantidad de fuentes de sonido sea finita, y puedan encapsularse en un número de categorías finito. De esta forma se podrán

detectar un número de tipos de evento, que se podrán clasificar en distintas categorías previamente acordadas, dado que actualmente no se plantea la posibilidad de realizar un detector universal, pero sí que se busca como objetivo lograr una buena generalización.

Se plantea resolver el problema aplicando las redes neuronales a SED, de forma que para una señal de audio de entrada se pueda clasificar qué sonido (dentro de las categorías especificadas) y en qué instante temporal está sucediendo. Para ello se debe entrenar el sistema con una base de datos ya etiquetada, con el fin de que el sistema pueda obtener una predicción de las etiquetas de forma automática. En este punto aparece un gran reto, dado que tanto la recogida de datos, como el etiquetado, resulta un trabajo difícil. El etiquetado inicial de una grabación debe ser manual, lo cual añade un grado de subjetividad al resultado, puesto que cada persona puede considerar que un sonido corresponde a una categoría u otra, y sobre todo variar la duración del sonido. Para evitar esta subjetividad se emplean varias técnicas, y se busca realizar una validación intensiva de los datos, para eliminar la subjetividad y ser lo más fiel a la realidad posible.

Tradicionalmente se ha buscado extraer las características de audio mediante representaciones directas de la señal correspondiente, como espectrograma, espectrograma MEL o transformando al dominio de la frecuencia con STFT (short-time Fourier transform). Y, posteriormente, empleando los valores obtenidos para clasificarlos, pero como se ha comentado hay ruidos que son continuos, y ruidos que son muy cortos, prácticamente instantáneos, por lo que se necesita un gran ancho de banda para poder representarlos todos. Por ello, se propone emplear otros métodos para obtener las características de las señales de audio. En este trabajo se propone como alternativa a estos métodos de extracción de características el uso de un codificador. De esta forma el objetivo final es el estudio de la clasificación de audios, que no sean música ni habla, mediante el uso de codificadores.

1.2. Objetivos

Se propone emplear un modelo de codificador-decodificador preentrenado en conjunto con un clasificador neuronal para resolver el problema de SED en entornos urbanos. Tomando la salida del codificador se puede caracterizar audios del tipo 'no música y no habla', mediante los 'tokens' que proporciona. Esta salida será la entrada al modelo clasificador, que será el encargado de identificar la clase a la que pertenece el audio inicial. Con esta arquitectura se busca aportar una solución innovadora al problema actual de la clasificación en *SED*.

Esta arquitectura se puede dividir en tres estudios diferenciados. En primer lugar, el

tratamiento de las distintas bases de datos, incluyendo el estudio, procesado, y aumento de datos cuando se necesite. En segundo lugar, el estudio y uso de una red preentrenada de codificación para adaptarla a los intereses que se proponen. Y finalmente, el diseño y resolución de la clasificación de audio.

1.3. Estructura de la Memoria

En el capítulo 2, se realiza un estudio del arte con los distintos conocimientos que son la base para toda la investigación posterior. Primero se explican los codificadores de audio en redes neuronales, que serán la base para la extracción de características, incluyendo el modelo *SoundStream* sobre el que posteriormente se trabajará. En segundo lugar se exponen las distintas bases de datos disponibles y las ventajas e inconvenientes de las mismas. Y finalmente se exponen las redes neuronales para clasificadores.

El desarrollo se divide en dos partes, y cada una de ellas se explica en su propio apartado, en el capítulo 3 se explica toda la investigación y trabajo codificando los audios de la base de datos escogida. Mientras que en el capítulo 4 se realiza el trabajo de clasificación en base la salida de la red neuronal anterior y se presentan los resultados obtenidos.

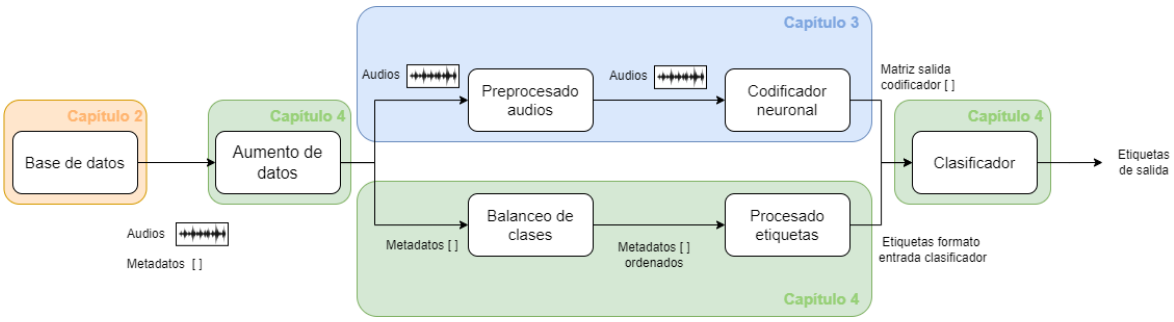

Figura 1.1: Esquema general del trabajo

Finalmente se cierra el trabajo con las conclusiones y posibles trabajos a futuro en base a los resultados obtenidos.

Capítulo 2

Estado del arte

En este apartado se incluyen las bases teóricas necesarias para el estudio e investigación posterior. Este capítulo se divide en dos mitades claramente diferenciadas, el codificador de audio y la clasificación.

El codificador de audio neuronal, apartado 2.2, es el encargado de extraer las características de los audios, para ello se debe emplear un modelo completo que codifique y decodifique el audio, entrenado como un conjunto para después emplear las características intermedias que se haya obtenido como representación del sonido de entrada. Además, este debe complementarse con un cuantificador, apartado 2.1, como se puede ver en la imagen 2.1. Finalizando esta primera parte del capítulo con la explicación de un modelo concreto *SoundStream* 2.3, y una evolución de este, *Encodec* 2.4.

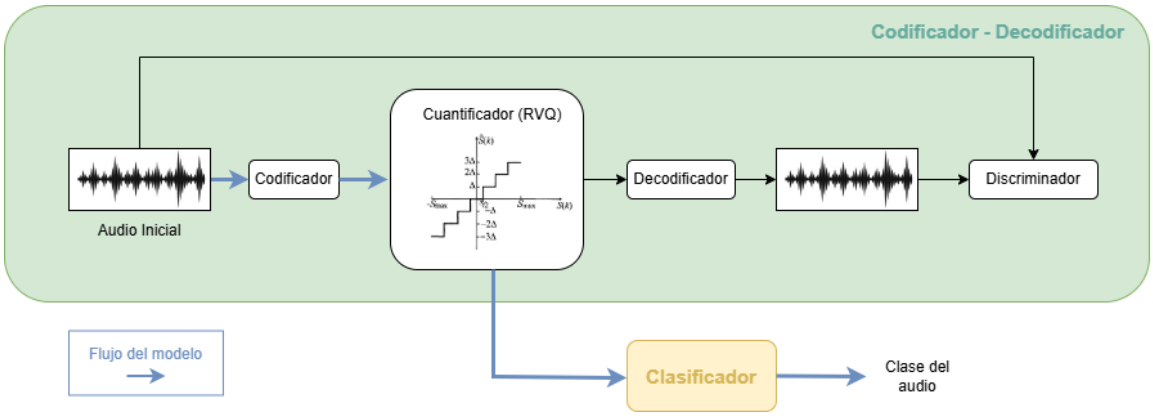

Figura 2.1: Arquitectura general del trabajo

En la segunda parte, se realiza un estudio sobre las bases de datos disponibles y cómo implementar un clasificador neuronal. Conocer estos temas es crucial para la comprensión del trabajo realizado.

2.1. Cuantificadores de audio

La cuantificación es el proceso de convertir una señal de entrada en valores discretos para poder ser tratados con mayor facilidad, a costa de perder precisión. Esta se puede realizar tomando un solo elemento de la secuencia de datos de entrada a la vez, lo que es cuantificación escalar, o tomando un grupo de elementos a la vez, lo que se conoce como cuantificación vectorial, VQ ('*Vector Quantization*'). [2]

Un cuantificador escalar se encarga de comparar la entrada con un conjunto finito predeterminado, llamado diccionario, y entregar una salida de forma que la diferencia sea mínima, es decir, la salida es el valor más cercano a la entrada. Generalizando este proceso, de forma que la entrada es un vector, y el diccionario se forma por N vectores del mismo tamaño que la entrada, tendremos un *VQ*. A los vectores del diccionario se les denomina 'vectores código' o 'palabras código'.

Al aplicar *VQ* a cada vector de la señal a codificar, no se tiene en cuenta la dependencia entre ellos, es decir, *VQ* carece de memoria. Con el fin de tener en cuenta esta dependencia se va a incluir la predicción, obteniendo un cuantificador vectorial predictivo. Para un predictor de orden m, cuando se codifica X_n tenemos disponibles en memoria los vectores aproximados $\hat{X}_{n-1}, \hat{X}_{n-2}, ...\hat{X}_{n-m}$. En función de estos se puede obtener $\tilde{X}_n$, que es la predicción de X_n, la diferencia de estos dos se conoce como error de predicción vectorial, E. La suma de la predicción y el error de predicción produce el vector aproximado, el cual se almacena en la memoria para emplearse al momento de codificar los próximos vectores de la secuencia de entrada. En el destino, a partir del índice se obtiene el error de predicción y este se suma con el vector de predicción obtenido por el predictor en función de los m vectores aproximados previos.

2.2. Codificadores de audio

Los codificadores de audio se encargan de convertir la señal digital en un formato específico dependiendo de su finalidad. Se pueden clasificar en dos categorías, codificadores de forma de onda o paramétricos. Los primeros buscan la posibilidad de reconstrucción de la entrada, suelen procesar la información en dominio tiempo-frecuencia y procesarse con cuantificación perceptiva, para poder ser completamente reconstruida, esto hace que puedan funcionar para todo tipo de audio. Como consecuencia, producen audio de muy alta calidad a velocidades de bits medias o altas, pero tienden a introducir artefactos de codificación cuando funcionan a tasas de bits bajas [2].

Los paramétricos pretenden solucionar este problema introduciendo un modelo que

describe el proceso de síntesis de audio. En el codificador se estiman los parámetros del modelo para después ser cuantificados. Por su parte, el decodificador genera una forma de onda que usa un modelo de síntesis con los parámetros cuantificados. A diferencia de los los códecs de forma de onda, no busca la reconstrucción basada en muestra a muestra, pero el resultado es perceptiblemente igual al original. Para mejorar su resultado se pueden añadir modelos de aprendizaje automático como postprocesado, lo cual hace que se logre una muy buena resolución del audio. Algunas soluciones adoptan modelos basados en '*Matching Learning*' (ML) dentro de la arquitectura del códec, como puede ser *'WaveNet'* [3], un modelo generativo de voz a partir de texto, que se emplea como decodificador en un códec neuronal.

Otra opción es *SoundStream* [4], un códec de audio que puede comprimir voz, música y audio general de forma eficiente. *SoundStream* aprovecha las soluciones más avanzadas en el campo de la síntesis basada en ML de audio e introduce un nuevo módulo de cuantificación entrenable, para ofrecer audio de alta calidad perceptiva, con una velocidad de bits de baja a media. A continuación se explican dos modelos distintos desarrollados a partir de éste, los cuales se emplearán en el desarrollo experimental, se trata de *SoundStream - Pytorch* [5] y *Encodec* [6].

2.3. *SoundStream*

SoundStream [4] es un códec de audio basado en redes neuronales diseñado para comprimir voz, música y audio de forma eficaz. Su arquitectura de alto nivel, mostrada en la figura 2.2, incluye un codificador completamente convolucional que toma como entrada una forma de onda en el dominio temporal y produce una secuencia de incrustaciones (*embeddings)* a una tasa de muestreo más baja. Estas incrustaciones se cuantifican mediante un cuantificador vectorial residual.

El modelo se entrena de extremo a extremo para distinguir entre el audio decodificado y el original, logrando como resultado un espacio donde se puede calcular una pérdida de reconstrucción basada en características. Tanto el codificador como el decodificador sólo utilizan convoluciones causales, por lo que la latencia arquitectónica global del modelo viene determinada la relación de muestreo temporal entre la forma de onda original y las incrustaciones.

2.3.1. Bases teóricas

Códecs de audio tradicionales, de este tipo podemos encontrar modelos como *Opus* [7] o *EVS* [8] que combinados con herramientas de codificación tradicionales que son eficientes para distintos tipos de sonido, velocidades y muestreo. *Opus*

es el codificador de audio que emplea *Youtube* para el 'streaming', mientras que EVS (*Enhanced Voice Services*) se trata del codificador estandarizado por 3GPP, especialmente diseñado para '*Voice over LTE*' (VoLTE).

Modelos generativos de audio, como *WaveNet* [9] o *SampleRNN* [10], generan audio de alta calidad a costa de una elevada complejidad computacional, dado que las muestras se generan una a una. Una evolución de estos son los modelos adversiales generativos, que dan una gran calidad de audio con poca complejidad computacional, como es *MelGAN* [11] que está entrenado para producir formas de onda de audio condicionadas por los espectrogramas-mel. El diseño del decodificador en *SoundStream* está basado en este tipo de clases de modelos generativos de audio.

Mejora del audio. El sistema aplica mejoras de audio entre las que se incluye la eliminación del ruido y de la reverberación, la codificación con pérdidas y la ampliación del ancho de banda de frecuencias.

Vector de cuantización. Emplea el aprendizaje del cuantificador vectorial paramétrico (PVQ) para lograr una alta codificación. Este modelo se emplea en el contexto de las redes neuronales para comprimir las variables del espacio latente de las características de entrada. Sin embargo, este modelo crece de tamaño exponencialmente al aumentar la tasa de bit de los audios de entrada, por lo que se proponen soluciones como los cuantificadores vectoriales estructurados para balancear entre la complejidad computacional y la eficiencia de codificación tradicional. *'Vector Quantized Variational Autoencoder'* (VQ-VAE) [12] es un cuantificador vectorial con capacidad de aprendizaje que se emplea en *SoundStream* en conjunto con un cuantificador vectorial residual (conocido como multietapa), entrenándolos de forma conjunta, siendo el primer modelo que lo planteó así, según los propios autores [4].

Cuantificador vectorial residual. También conocido como RVQ, se trata de una cuantificación vectorial propuesta en [4], basada en la implementación de *Tensorflow* de *Deepmind*. Se basa en el uso de medias móviles exponenciales para la actualización del diccionario.

Códecs de audio neuronales. Los códecs de audio neuronales de extremo a extremo emplean métodos basados en datos para aprender representaciones de audio eficientes, en lugar de depender de componentes de procesamiento de señales hechos a mano. Se ha comprobado en varios trabajos que una compresión eficiente de audio mediante el uso de redes neuronales es posible, en la mayoría se emplea la codificación del habla a tasas de bits bajas. En estos casos se especifica un tipo de audio, a diferencia de *Soundstream* que no realiza suposiciones acerca de la naturaleza de la señal que codifica, gracias a ello funciona para diversos tipos de audio, debido a su entrenamiento extremo a extremo. Además, *Soundstream* logra la escalabilidad de la tasa de bits, es

decir, un modelo único funciona para distintas entradas sin coste adicional, mediante el uso del cuantificador vectorial residual, y del modelo de entrenamiento que emplean. Esto hace que fuera el primer códec de audio neuronal para una gran gama de tasas de bits, a diferencia de otros como *Opus* [7] o *EVS* [8] .

Compresión y mejora de las articulaciones. Integra una capa de acondicionamiento que permite la eliminación del ruido controlable en tiempo real, esto permite el manejo de sonidos naturales que de otro modo se eliminarían.

2.3.2. Arquitectura de SoundStream

La arquitectura de *SoundStream* se puede dividir en tres fases, codificador, cuantificador residual vectorial y decodificador. Las cuales son entrenadas de extremo a extremo junto con un discriminador. Además, puede elegirse si se aplica la eliminación de ruido en el lado del codificador o del decodificador.

- ***Codificador:*** convierte el audio de entrada a parámetros listos para cuantificar.
- ***Vector de cuantificador residual:*** reemplaza cada elemento por una suma de vectores codificados, comprimiendo así la representación con un número de bits.
- ***Decodificador:*** realiza una reconstrucción con pérdidas a partir de la salida anterior.
- ***Discriminador:*** puede ser un discriminador basado en la señal temporal que recibe una sola entrada, o un discriminador basado en STFT, que recibe un valor complejo STFT de la señal de entrada. Originalmente se plantea el uso de Espectrograma-MEL ya que es más similar a la percepción humana, aunque se ha evolucionado hacia el uso de STFT como se indica en [13].

A continuación se va a explicar en profundidad el codificador en conjunto con el cuantificador, ya que en este trabajo no se va a emplear el decodificador.

La arquitectura del codificador (ver figura 2.2) consta de una capa de convolución 1D (con canales C_{enc}), seguida de B_{enc} bloques de convolución. Cada bloque de convolución consta de tres unidades residuales que contienen convoluciones dilatadas con tasas de dilatación de 1, 3 y 9, respectivamente, seguidas de una capa de muestreo descendente en forma de convolución estriada. El número de canales se duplica cada vez que se realiza un submuestreo, a partir de C_{enc}. Finalizando con una capa de convolución 1D con un núcleo de longitud 3 y un paso de 1 para ajustar a las dimensiones de los datos embebidos.

El modelo es capaz de realizar predicciones causales, para mantenerla se aplica técnicas de relleno sólo en entradas anteriores, pero no en posteriores. Se emplea activación ELU ('Exponential Linear Unit'), que es capaz de manejar valores negativos suavizando la salida, sin normalización.

El tamaño de la salida depende del número de bloques de convolución, B_{enc}, y los saltos que tengan. En este caso para 4 bloques, con saltos de (2, 4, 5, 8), el tamaño de la entrada es $M = 2458 = 320$. Resultando la salida del codificador $\text{enc}(x) \in \mathbb{R}^{S \times D}$, donde $S = \frac{T}{M}$. Siendo D el número de características por cada vector de salida del codificador y T la longitud temporal del audio de entrada.

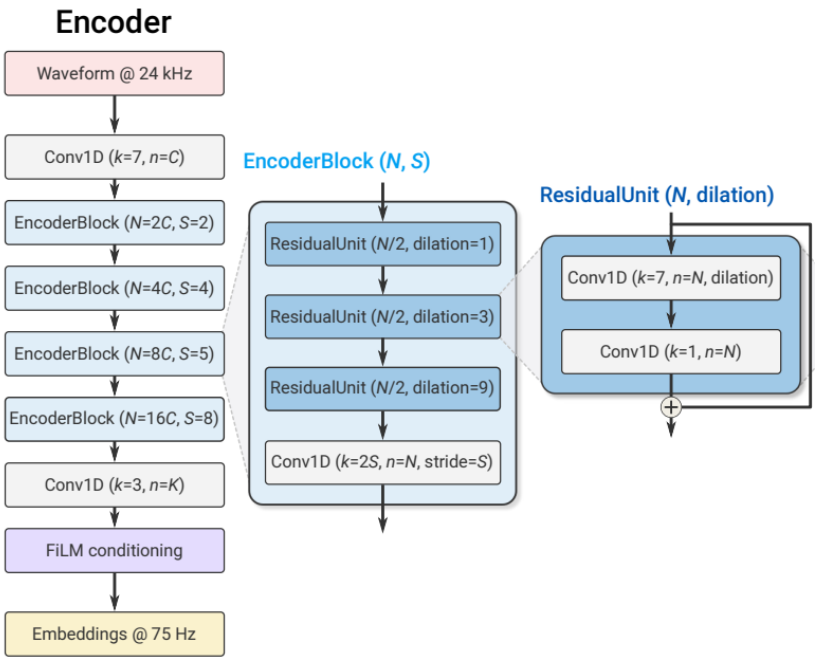

Figura 2.2: Codificador SoundStream, [4]

La salida del codificador es la entrada del cuantificador, encargado de comprimirlo para lograr la tasa de bits deseada, 'R'.

En cuanto al entrenamiento de los diccionarios, 'codebooks', se actualizan usando el promedio móvil exponencial, 'EMA', que permite ajustas los vectores del diccionario de manera suave y gradual basándose en el promedio ponderado de las observaciones anteriores y actuales. A lo cual se añaden las siguientes mejoras:

- ***Inicialización mediante K-means,*** se asegura que los vectores iniciales del diccionario estén más cerca de la distribución de entradas desde el comienzo, mejorando así la eficiencia del modelo.

- ***Reemplazo de vectores no asignados,*** si un vector del diccionario no ha sido asignado a ningún cuadro de entrada durante varios lotes de entrenamiento,

se reemplaza por una muestra aleatoria de un cuadro de entrada del lote actual.

– ***Seguimiento de asignaciones con promedio móvil exponencial,*** que usa el promedio móvil exponencial de las asignaciones, con un factor de 0.99. De forma que si el promedio cae por debajo de 2, se reemplaza.

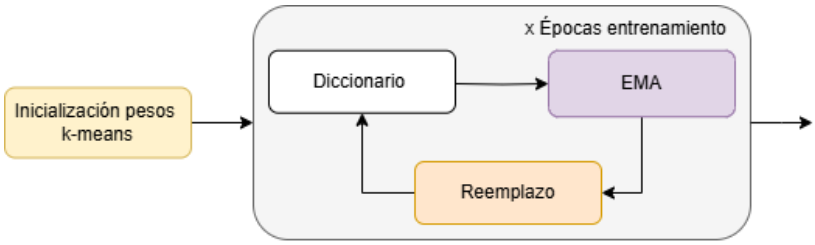

Figura 2.3: Estructura cuantificador, RVQ

El cuantificador utiliza un estimador directo (straight-through-estimator) [14] para calcular el gradiente del codificador, de forma que el paso de cuatización se emplea al realimentar el sistema.

Se emplea una combinación de pérdidas, por un lado respecto a la fidelidad de la reconstrucción de la señal y por otro lado a la calidad perceptual, esta segunda es la extraída de la combinación de los errores de los discriminadores. [4].

2.3.3. Objetivo de entrenamiento, mejoras

El entrenamiento de *SoundStream* se realiza en el sistema completo $G(x) = dec(Q(enc(x))$. Para lograr fidelidad en la reconstrucción de la señal y la suficiente calidad perceptiva se emplea una mezcla de pérdidas, siguiendo los principios del equilibrio percepción-distorsión. Además, se emplea la pérdida adversarial para promover la calidad perceptiva y se define como una pérdida sobre el discriminador, ponderando sobre varios discriminadores en tiempo.

$$L_G = \lambda_{adv} L_G^{adv} + \lambda_{feat} \cdot L_G^{feat} + \lambda_{rec} \cdot L_G^{rec}.$$

Siendo L_G las pérdidas globales, L_G^{adv} las pérdidas adversariales, L_G^{feat} las pérdidas por las características y L_G^{rec} las pérdidas de reconstrucción. Los valores de λ son constantes que varían en función de la implementación.

Las distintas pérdidas se calculan como:

$$L_G^{adv} = \mathbb{E}_x \left[\frac{1}{K} \sum_{k,t} \frac{1}{T_k} \text{máx}\left(0, 1 - D_{k,t}(G(x))\right) \right]$$

Donde $\mathbb{E}_x$ es el valor esperado respecto a x, K el número de discriminadores, T_k la cantidad de valores de salida por instante temporal para cada discriminador y $D_{k,t}$ la salida del discriminador.

$$L_G^{feat} = \mathbb{E}_x \left[\frac{1}{KL} \sum_{k,l} \frac{1}{T_{k,l}} \sum_t \left| D_{k,t}^{(l)}(x) - D_{k,t}^{(l)}(G(x)) \right| \right]$$

Donde L es el número de capas internas.

$$L_G^{rec} = \sum_{s \in \{2^6, \ldots, 2^{11}\}} \sum_t \| S_t^s(x) - S_t^s(G(x)) \|_1 + \alpha_s \sum_t |\log S_t^s(x) - \log S_t^s(G(x))|_2$$

Donde $S_s^t(x$ es el fotograma número t del espectrograma mel y α_s es una variable que se puede elegir según la implementación.

De esta forma se computan las distintas pérdidas del sistema buscando la solución que disminuya todas ellas teniendo en cuenta su peso dentro del cómputo global.

Una innovación respecto a los canales tradicionales es la cohesión de la compresión y la optimización en el mismo módulo, disminuyendo la latencia de extremo a extremo. Además, incluye una señal de acondicionamiento, la cual se encarga de tratar el ruido en el codificador o en el decodificador según su estado. De esta forma la red se entrena para reconstruir el habla ruidosa si la señal de acondicionamiento está desactivada, y para producir una versión limpia de la entrada ruidosa si está activada. Teniendo en cuenta que cuando las entradas consisten en audio limpio (voz o música), la eliminación de ruido se desactiva. Esto se hace para evitar que *SoundStream* afecte negativamente al audio limpio cuando se activa la eliminación de ruido.

2.3.4. Dataset de entrenamiento

Este trabajo emplea *SoundStream* preentrado [5], por lo que es de interés conocer la forma en la que se ha entrenado para poder conocer cómo va a afectar a los resultados.

Se puede clasificar la entrada en tres tipos de audios: habla limpia, habla ruidosa y música, para una frecuencia de muestreo 24kHz. Donde se emplean las bases de datos de la tabla 2.1 como conjunto de entrenamiento. Mientras que las evaluaciones objetivas y subjetivas se realizaron utilizando 200 clips de audio, 50 de cada base de datos, de entre 2 y 4 segundos de duración.

Tipo de dato	Base de datos
Voz limpia	LibriTTS
Voz ruidosa	LibriTTS con ruido de Freesound
Música	MagnaTagATune
Voz de campo cercano y lejano	Datos propios recopilados

Tabla 2.1: Bases de datos

Además, generan una base de datos propia formada por datos reales y lejanos (en espacios reverberante), con ruido de fondo en algunos casos. En definitiva, se entrena el modelo sobre un conjunto de 200 clips de audio de 2 a 4 segundos de duración, con 50 muestras de cada uno de los cuatro de cada uno de los cuatro conjuntos de datos anteriores.

2.4. *Encodec*

Encodec [15] es un modelo de codificador y decodificador propiedad de *Meta*, que parte de *SoundStream* y evoluciona a partir de él. Este modelo surge de la necesidad de aumentar la compresión de los audios, y además reducir el tráfico generado cuando se transmiten, prestando más atención sobre la calidad de salida que en otros trabajos. Se recalca que una de las motivaciones principales de su desarrollo es minimizar las pérdidas de acuerdo a una métrica que está directamente correlada con la percepción humana.

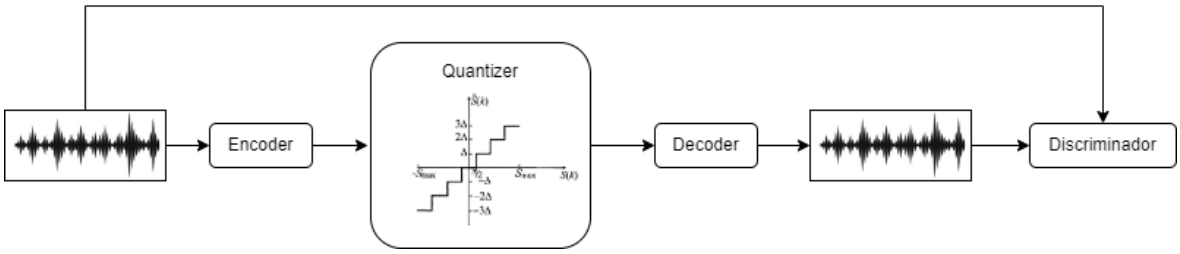

Figura 2.4: Codificador Encodec, [15]

Los principales problemas de estos sistemas, y que no quedan resueltos en SoundStream son dos. Por un lado, el modelo debe ser capaz de abarcar un amplio rango de señales de entrada, sin llegar a un sobreajuste. En este punto en definitiva se busca que el modelo tenga una buena generalización. Para solucionarlo, se plantea una mejora en el '*dataset*' que se emplea para el entrenamiento, haciéndola más amplia y diversa, e implementa grandes mejoras en los discriminadores. El otro problema es la eficiencia de compresión en tiempo y tamaño, el cual se logra mejorar respecto a [4].

2.4.1. Arquitectura de Encodec

Este modelo se compone de una arquitectura simple de codificador-decodificador basada en convoluciones y en '*streaming*', con un componente de modelado secuencial aplicado sobre la representación latente, tanto en el lado del codificador como en el del decodificador.

Codificación y decodificación

Se puede ver la estructura del codificador en la figura 2.5, y el decodificador refleja al codificador, utilizando convoluciones transpuestas en lugar de convoluciones con salto, y con los pasos en orden inverso al del codificador, generando el audio mono o estéreo final.

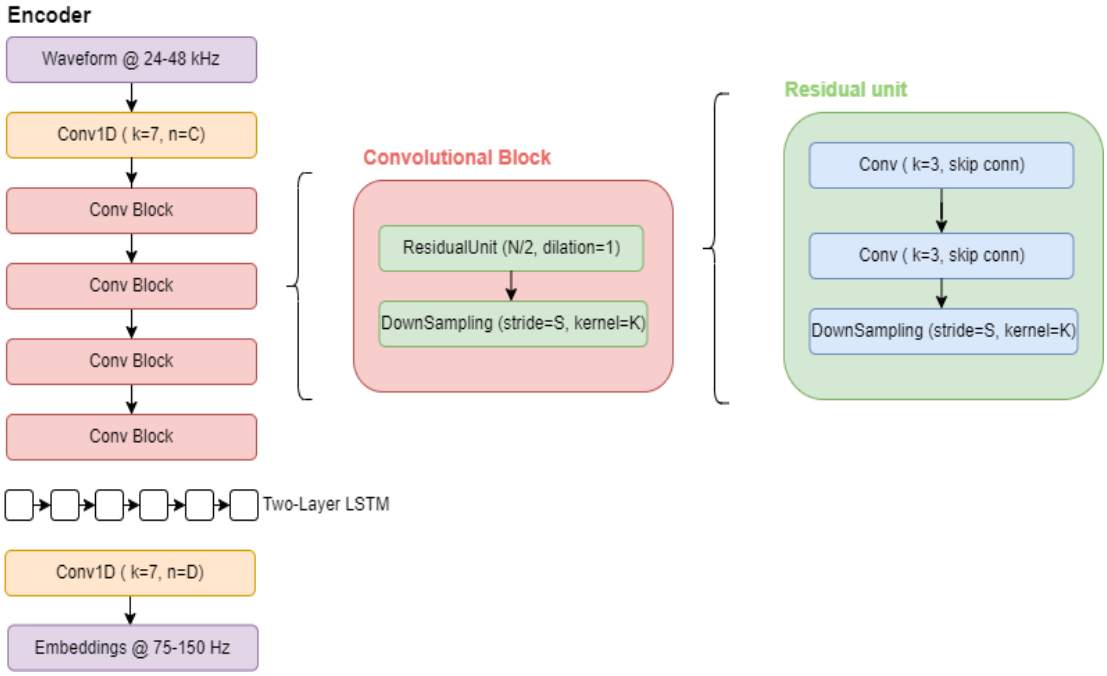

Figura 2.5: Codificador Encodec, [6]

Cuantificador residual de vectores (RVQ)

Se basa en la versión de *SoundStream*, de forma que realiza una doble cuantificación, la propiamente dicha y una a partir del residuo resultante de la primera. Para esta estructura se emplean 32 libros de códigos con 1024 entradas cada uno, lo que resulta en 10 bits por libro de códigos. Como se ha comentado, se busca variabilidad, con este fin se entrena con un ancho de banda variable, seleccionando aleatoriamente un número de libros de códigos con un múltiplo de 4. La salida del codificador es de forma $[B, D, T]$, siendo B el número de bloques convolucionales, D el número de canales de

salida y T el número de canales de audio, y tras salir del cuantificador se convierte en el conjunto discreto $[B, N_q, T]$, N_q es el número de libros de códigos seleccionados.

Modelado del lenguaje y entropia de los códigos

Como novedad este modelo introduce una parte de modelo del lenguaje basado en '*Transformer*', según se describe en [16]. Cuyo objeto final es acelerar la inferencia a costa de entropía cruzada final, aunque se produzca algo de pérdida en términos de entropía, es una mejora a nivel global.

Pérdidas

Las pérdidas del conjunto se deben a varias fuentes, por lo que es interesante medir cada una de ellas. Las pérdidas totales son el cómputo ponderado de las mismas, los pesos se denominan λ.

$$L_G = \lambda_t \cdot \ell_t(x, \hat{x}) + \lambda_f \cdot \ell_f(x, \hat{x}) + \lambda_g \cdot \ell_g(\hat{x}) + \lambda_{\text{feat}} \cdot \ell_{\text{feat}}(x, \hat{x}) + \lambda_w \cdot \ell_w(w)$$

A continuación se explica cada una de las pérdidas por separado.

Función de pérdida adversaria. Encargada de medir la diferencia entre las muestras reales y las generadas.

$$l_g(\hat{x}) = \frac{1}{K} \sum_k max\left(0, 1 - D_k(\hat{x})\right)$$

Teniendo K discriminadores, denominando D_k a los discriminadores de L capas.

Pérdidas de reconstrucción. Se busca minimizar la distancia entre el objetivo y el audio de salida en dominio del tiempo. Para disminuir la distancia en el dominio temporal se emplea:

$$l_t(x, \hat{x}) = \|x - \hat{x}\|_1$$

Mientras que en el dominio de la frecuencia se emplea una combinación lineal entre las pérdidas sobre el espectrograma de Mel para distintas escalas de tiempo.

$$l_f(x, \hat{x}) = \frac{1}{|\alpha| \cdot |s|} \sum_{\alpha_i \in \alpha} \sum_{i \in e} \left(\|S_i(x) - S_i(\hat{x})\|_1 + \alpha_i \|S_i(x) - S_i(\hat{x})\|_2\right)$$

Siendo S_i un espectrograma-MEL de tamaño 64 usando STFT normalizado con ventana de tamaño 2^i y tamaño del salto $2^i/4$, $e = 5, ..., 11$ es la escala a emplear, y α coeficientes escalares.

Pérdidas del discriminador. Pérdidas perceptuales basadas en un discriminador MS-STFT (transformada de Fourier de tiempo corto multi-escala), capaces de capturar

distintas estructuras en señales de audio. Este discriminador consiste en redes con la misma estructura que operan en STFT de valores complejos en múltiples escalas con las partes real e imaginaria concatenadas.

$$l_{\text{feat}}(x, \hat{x}) = \frac{1}{KL} \sum_{k=1}^{K} \sum_{l=1}^{L} \frac{\|D_k^l(x) - D_k^l(\hat{x})\|_1}{\text{mean}(\|D_k^l(x)\|_1)}$$

Los discriminadores son entrenados para minimizar la función de pérdida $L_d(x, \hat{x}) = \frac{1}{K} \sum_{k=1}^{K} (\text{máx}(0, 1 - D_k(x)) + \text{máx}(0, 1 + D_k(\hat{x})))$ donde K es el número de discriminadores.

Pérdida de compromiso VQ. Se añaden unas pérdidas a la salida del codificador y antes de su valor cuantificado, sin que se calcule el gradiente para el valor cuantizado. En general, el generador se entrena para optimizar la siguiente pérdida.

$$l_w = \sum_{c=1}^{C} \|z_c - q_c(z_c)\|_2^2$$

Balanceador. Añade estabilidad al entrenamiento, en particular la escala variable de los gradientes provenientes de los discriminadores.

$$\tilde{g}_i = \frac{RP}{\lambda_i} \cdot \frac{g_i}{\sum_j \lambda_j \|g_i\|_2^2}$$

Donde toman $R = 1, \beta = 0{,}999$.

Dataset de entrenamiento

Este modelo se ha entrenado con audios de 24kHz monofónicos de diversos dominios: habla, habla ruidosa, música y audios generales. Para el habla se emplea segmentos de habla limpios extraídos del '*DNS Challenge 4*' [17] y del conjunto de datos '*Common voice*' [18]. Para audio general se emplea '*AudioSet*' [19] y '*FSD50K*' [20]. Y la música se entrena con '*Jamendo*' [21] para entrenamiento y evaluación, además de con un conjunto de música privado. Para el entrenamiento y validación emplean una estrategia mixta, empleando entre una y tres fuentes de datos para poder realizar un resultado totalmente generalizable.

Método de evaluación

Se han empleado tanto métricas objetivas como subjetivas. Estas últimas se implementan mediante el modelo MUSHRA (*Multriple Stimuli with Hidden Reference and Anchor*) [22], es un método subjetivo para cuantificar la calidad de audio percibida tras el proceso de codificación y decodificación. Se deben filtrar las anotaciones ruidosas y los valores atípicos se eliminan de las anotaciones que se requieran. Mientras que para

la métrica objetiva se emplea ViSQOL [23], métrica que emplea modelos de percepción auditiva para calcular cuánto de similar es la calidad de la voz transmitida a la calidad de la voz original. Para medir la relación señal ruido se emplea la métrica invariante en escala, SI-SNR (Scale-Invariant Signal-to-Noise) [24].

2.5. Ajuste fino

El ajuste fino, o *fine tuning*, es una técnica de entrenamiento que busca poder emplear una red convolucional predefinida y pre-entrenada. Se trata de un proceso en el que se realiza el ajuste fino de algunas capas de la red para obtener las salidas deseadas. Es decir, se ajustan ligeramente ciertas representaciones del modelo preentrenado para hacerlo más pertinente al problema en cuestión. Esto evita tener que definir la estructura de la red neuronal y entrenarla desde cero. Cabe destacar que no es '*transfer learning*'. En el aprendizaje por transferencia, un modelo entrenado para una tarea se reutiliza para resolver otra congelando los parámetros del modelo existente. Mientras que en el ajuste fino, se toman los parámetros de red existentes y se siguen entrenando para realizar la segunda tarea. Básicamente, se adapta y entrena la estructura del modelo.

2.5.1. Técnicas de ajuste fino

- ***Con muchos datos de entrenamiento.*** Se basa en tener acceso a un modelo preentrenado para una tarea similar y disponer de muchos conjuntos de datos para el entrenamiento. De tal forma se parte de la base y se continúa el entrenamiento con los nuevos datos.

- ***Cantidad limitada de datos de entrenamiento.*** En el caso en el que los datos de entrenamiento son escasos, se debe evitar el sobreajuste, por lo que se plantea entrenar solo las últimas capas de la estructura de la red, congelando el resto.

Cabe al posibilidad de que la red no esté preentrenada pero sí predefinida, en ese caso se proponen las dos opciones siguientes:

- ***Con muchos datos de entrenamiento.*** La mejor opción es entrenar el modelo desde cero.

- ***Cantidad limitada de datos de entrenamiento.*** Se debe optar por congelar las primeras capas de un modelo con un fin similar, usando las características

universales que sí es capaz de extraer este modelo y entrenar las últimas capas únicamente.

2.6. Bases de datos

Debido a la imposibilidad de realizar con los medios actuales una base de datos que sirva para una clasificación universal, se han desarrollado distintas bases de datos dependiendo del objetivo.

Además, al tratarse de un campo muy amplio, las bases de datos se pueden agrupar según las fuentes, según el estado de las fuentes o según factores ambientales. Cada base de datos debe ser etiquetada para poder ser usada, este etiquetado puede ser de todo el audio completo, sin especificación temporal, lo que se denomina etiquetado débil, o etiquetado cada sonido en su instante temporal dentro del audio, lo que es etiquetado fuerte.

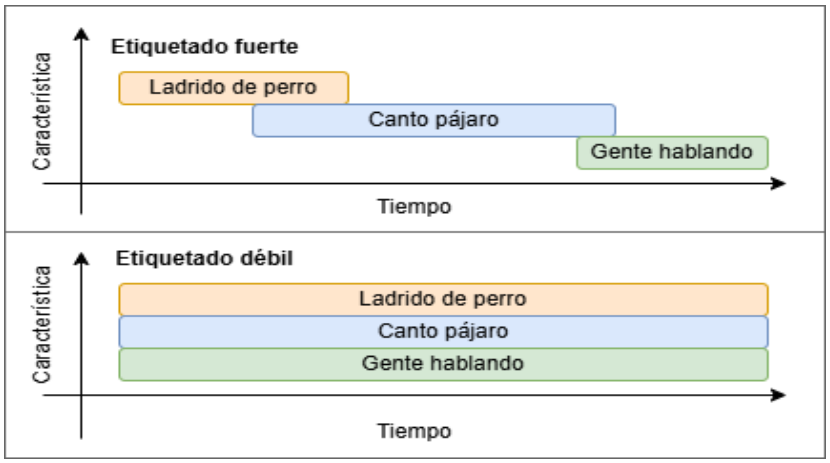

Figura 2.6: Etiquetado débil frente a fuerte, imagen original [1]

La mejor opción es tener un etiquetado fuerte, aunque para obtenerlo en un entorno real es necesario invertir mucho tiempo y recursos en realizarlo de forma correcta, tratando de eliminar la subjetividad. Las anotaciones que contienen información temporal para cada clase de forma binaria crean un mapa de bits donde para cada instante se indica si esa clase está presente o no. A esto se le conoce como clasificación 'Multicalse', puesto que tenemos varias clases, 'Multietiqueta', puesto que puede haber varias clases presentes en el mismo instante. En una aproximación al problema, se puede trabajar con una única etiqueta, de forma que para un mismo instante, solo se marca como presente una clase en cada instante, a pesar de la posibilidad de haber varios. Otros trabajos de única etiqueta se han entrenado con grabaciones de muy corta duración donde solo aparezca una clase.

Debido a la dificultad del etiquetado, algunos trabajos proponen generar un audio sintético, de forma que se realice automáticamente. Crear un audio sintético no se trata de mezclar los sonidos referencia, sino de obtener un audio que suene natural, de forma que la aparición de sonidos, su superposición, frecuencia de aparición y relación entre ellos sea similar a la realidad. Además, se pueden utilizar numerosas técnicas de aumento para tener un conjunto de datos mayor. El aumento de datos puede ser mediante la adición de ruido, otros sonidos no clasificables, distintas mezclas, o realizar ensanchado entre otros muchos.

En la literatura podemos encontrar varias bases de datos desarrolladas para SED de uso público, de forma que se pueden usar como entrada al sistema. Sin embargo, debido a la inmensidad de tipos de sonidos en esta categoría, no todos son válidos para lograr nuestro objetivo. En esta sección se va a ver un breve estudio de las bases de datos más conocidas y sus ventajas e inconvenientes.

Base de datos	**Anotación**	**Etiq.**	**Clases**	**Datos**	**Notas**
FSDnoisy18k[25]	Manual y autom.	Débil	20	42.5h	Fuerte verificación de etiq. manual
AudioSet[26]	Automático	Débil	527	5800h	Solo parcialmente verificada
URBAN-SED	Sintético	Fuerte	10	30h	Todo material sintético
TUT Sound Events 2016	Manual	Fuerte	18	2h	-
CHime-Home	Manual	Fuerte	7	6.8h	Sonidos de entorno doméstico
Urban Sound 8k	Manual	Fuerte	10	27h	Sonidos de entorno urbano
ESC-50	Manual	Fuerte	50	6.95h	Sonidos ambientales
DESED	Manual y Sintético	Fuerte	10	2h	Dataset de detección de eventos sonoros en entorno doméstico

Tabla 2.2: Bases de datos

En esta primera fase de investigación se necesita un etiquetado fuerte, de forma que permita realizar unas pruebas más certeras y con mayor seguridad. Con etiquetado fuerte se puede encontrar, URBAN-SED [27], TUT Sound Events 2016 [28], CHime-Home [29], Urban Sound 8k [30], ESC-50 [31] y DESED [32], se va a realizar un pequeño análisis de ellas a continuación.

URBBAN-SED

Esta base de datos se ha creado con el objetivo de usarse para la detección en ambientes urbanos. De forma que se pueda emplear para el desarrollo de nuevas soluciones SED, sobretodo en aplicaciones de monitorización de ciudades, coches,

vigilancia y control bioacústico. Debido al tiempo necesario para realizar un etiquetado fuerte de gran calidad de forma manual, se ha decidido realizar una base de datos sintética, ayudada del método de aumento, para asegurar el etiquetado correcto, no sujeto a la subjetividad del etiquetado manual. Para su creación se ha empleado el método '*opensource*' *Scaper* sobre *Python* [27], que actúa como un secuenciador de audio de alto nivel, controlado probabilísticamente que puede generar nuevos paisajes sonoros o añadir nuevos eventos sonoros a los existentes, controlando las características que se le especifique. Esta base de datos tiene 10000 paisajes sonoros de 10 segundos, empleando como entrada los clips de *UrbanSound8K*, y ampliándolos con *Scaper*. Introduciendo un fondo normalizado a -50LUFS, de ruido marrón igual para todos los clips, esto evita añadir espurios a las anotaciones. Sobre este, se incluye un sonido de las 10 clases que especifica la base, distribuyendo los sonidos según un modelo gaussiano, de forma que hay zonas temporales donde son más probables que aparezca el sonido que otras. Para crear más variedad, cada evento se desplaza un semitono según una distribución normal y se alarga un factor según una distribución uniforme entre 0.8 y 1.2. Sin embargo, esta base de datos contiene polifonía, de hasta 7 sonidos superpuestos, por lo que no es válido para el objetivo que buscamos.

TUT Sound Events 2016

Esta base de datos pretende cumplir con la tarea de '*in-home sound source recognition*' (SSR) asignando etiquetas semánticas a una grabación de audio dada. Las fuentes de audio corresponden a '*acoustic scene*' y tienen una única etiqueta por audio. Esta clasificación se centra en aportar seguridad y protección en el hogar, siendo sus aplicaciones objetivo tales como la detección de intrusos, el cuidado de personas mayores, la seguridad del hogar, etc.

Contiene grabaciones de entorno residencial y doméstico. Se trata de grabaciones reales ya que defiende que la creación sintética no permite replicar la variabilidad de la vida real, a pesar de hacer que el trabajo sea menos costoso y da una solución mucho más amplia. Las grabaciones se han realizado con una frecuencia de muestreo de 44.1kHz, que han sido anotados y postprocesados. Se plantean dos entornos, uno al aire libre (zona residencial) y otro interior (hogar), con la siguiente clasificación de sonidos:

Num apariciones	Descripción
23	Golpe
271	Canto pájaro
108	Coche
31	Niños
52	Gente hablando
44	Gente andando
30	Aire

Tabla 2.3: Sonidos de área residencial

Num apariciones	Descripción
60	Crujidos de objetos
57	Chasquidos de objetos
40	Armario
76	Cubiertos
151	Platos
51	Cajón
36	Cristales
250	Choque de objetos
54	Gente andando
84	Lavado de platos
47	Grifo abierto

Tabla 2.4: Sonidos de interior de un hogar

Cabe destacar que se descartó la base de datos *TUT Sound Events 2017*, debido a que se trata de un entorno urbano, y tiene menos similitud con el tema del trabajo, siendo una evolución de la versión de 2016 la escogida.

CHime-Home

Cuenta con 6.8h de audio doméstico, etiquetado manualmente por 3 personas distintas todo el sonido para al final sacar las métricas más exactas posibles, dentro de la subjetividad del etiquetado manual.

Las grabaciones se han extraído cuidadosamente de un trabajo previo, de 19.5h se han obtenido 6.8h, de sonidos de salón y cocina. Principalmente, se trata de sonidos de dos adultos, dos niños, televisión, electrodomésticos, y algunos sonidos del exterior.

En cuanto a las anotaciones, se etiquetan fragmentos de 4 segundos no solapados. Este tamaño viene dado por investigaciones previas [33] que cumple con el compromiso resolución temporal de anotaciones y coste temporal de obtenerlas. En total se han obtenido 6138 segmentos de 4 segundos, de los cuales se emplean 4378 segmentos, de

forma que para cada segmentos hay un conjunto de etiquetas. Las posibles etiquetas son:

Etiqueta	Descripción
c	Habla de niños
m	Habla de adulto masculina
f	Habla de adulto femenina
v	Televisión / Videojuego
p	Sonidos percusivos (crash, bang, golpe, pisadas...)
b	'Broadband noise', como pueden ser los electrodomésticos
o	Otros (identificables pero no incluios en los anteriores)
S	Silencio
U	No identificables

Tabla 2.5: Clases CHime-Home

Todos los datos se han recogido a 48kHz, y a esta frecuencia han sido etiquetados, tal y como podemos ver en el *dataset* original [34]. En este caso tienen asignada una única etiqueta semántica para cada grabación de audio, es decir, es monofónico.

Urban Sound 8k

Este conjunto de datos se centra en resolver dos obstáculos típicos para la detección de sonido urbano, la escasez de datos anotados de gran tamaño y la falta de una taxonomía común. Contiene 8732 extractos de audio reales etiquetados, cuya duración es menor a 4 segundos, divididos en 10 clases derivadas de un estudio de taxonomía de sonidos, que se puede encontrar explicada en [30]. Esta base de datos ha sido extensamente tratada para facilitar su uso, de forma que se divide en 10 subgrupos que se deben emplear de forma específica para el estudio, este uso se verá aplicado en el apartado 4.1.3.

La división de clases se muestra en la tabla 2.6, cabe destacar que a excepción de 'niños jugando' y 'disparos de pistolas' que han sido añadidos para aportar variedad, el resto de clases se han seleccionado de entre toda la taxonomía debido a su alta frecuencia de aparición en los ruidos urbanos según los estudios realizados. Se ha limitado a estas 10 clases porque conllevan un etiquetado manual, y se decidió abarcar los campos más frecuentes con aspiraciones de aumentar la base de datos en un futuro.

Num. apariciones	Descripción
1000	Aire acondicionado
571	Motor coche
1000	Niños jugando
1000	Perro ladrando
1000	Taladro
1000	Ralentí del motor
373	Disparos de pistolas
1000	Martillo neumático
1000	Sirena
1000	Música callejera

Tabla 2.6: Clases de Urban Sound 8k

ESC-50

Este *dataset* [31] cuenta con 2000 grabaciones etiquetadas con 50 clases balanceadas, 40 audios por clase. Estas clases se pueden agrupar en 5 categorías que incluyen 10 clases por categoría, como se puede ver en su distribución:

Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Dog	Rain	Crying baby	Door knock	Helicopter
Rooster	Sea waves	Sneezing	Mouse click	Chainsaw
Pig	Crackling fire	Clapping	Keyboard typing	Siren
Cow	Crickets	Breathing	Door, wood creaks	Car horn
Frog	Chirping birds	Coughing	Can opening	Engine
Cat	Water drops	Footsteps	Washing machine	Train
Hen	Wind	Laughing	Vacuum cleaner	Church bells
Insects (flying)	Pouring water	Brushing teeth	Clock alarm	Airplane
Sheep	Toilet flush	Snoring	Clock tick	Fireworks
Crow	Thunderstorm	Drinking, sipping	Glass breaking	Hand saw

Figura 2.7: Distribución clases y categorías, fuente [35]

Los audios se procesan con el objetivo de mantener los eventos sonoros de primer plano limitando el ruido de fondo cuando sea posible. Sin embargo, las grabaciones de campo están lejos de estar libres de ruido, por lo que algunos *clips* aún pueden mostrar superposición auditiva en el fondo.

Este conjunto de datos contiene un número limitado de *clips* disponibles por clase. Esto se debe al alto coste de la anotación y extracción manual, y la decisión de

mantener un equilibrio estricto entre las clases a pesar de la disponibilidad limitada de grabaciones para tipos más exóticos de eventos sonoros. No obstante, se espera que sea útil en su forma actual y es un concepto que podría expandirse si resulta de interés.

2.6.1. Aumento de datos

Por lo general, los conjuntos de audio disponibles para SED suelen ser reducidos, debido a la complejidad de obtención, como se ha explicado previamente, es por ello que son necesarias técnicas de aumento de datos para evitar un sobreajuste del sistema, es decir, que aprenda demasiado de los datos de entrada y no sea generalizable a otras entradas. [36] Las técnicas de aumento de audio más extendidas son:

- ***Estiramiento temporal.*** Aumenta o disminuye la velocidad de reproducción del audio sin afectar al tono o duración.
- ***Cambio de tono.*** Altera el tono o la frecuencia del audio, manteniendo su duración.
- ***Escalado temporal.*** Este produce el cambio simultáneo de la duración de la pista y su tono. Para ello combina dos técnicas: el estiramiento temporal y el cambio de tono.
- ***Escalado de amplitud.*** Ajusta el volumen o la intensidad de la señal de audio, esto simula distintos niveles de sonido o distancias desde la fuente al micrófono.
- ***Inversión temporal.*** La inversión de una pista a lo largo de su eje temporal se relaciona con el volteo aleatorio de una imagen, que es una técnica de aumento muy utilizada en el dominio visual.
- ***Recorte aleatorio y relleno.*** Debido al requisito de alinear la duración de la pista antes de procesarla a través del modelo, se aplica recortes aleatorios o relleno a las muestras que eran más largas o más cortas que la pista más larga en un conjunto de datos no aumentado, respectivamente.
- ***Ruido aleatorio.*** La adición de ruido aleatorio es útil para superar el sobreajuste en tareas con realismo visual, por lo que se ha extrapolado su uso a otros contextos, como los audios.
- ***Reverberación.*** Se aplican efectos al clip de audio para simular distintos entornos acústicos. Esto puede ayudar a que el modelo se generalice mejor en diferentes entornos de grabación.

2.7. Redes neuronales para clasificadores de audio

Para la tarea de clasificar se emplea un enfoque basado en aprendizaje supervisado, es decir, los datos de entrada están etiquetados, y estas etiquetas corresponden a la salida esperada. Durante el aprendizaje el sistema aprende la correspondencia entre las características extraídas de la señal de audio y la anotación que representa la actividad de cada clase. Finalmente en la fase de test el sistema recibe las características extraídas del audio, mediante un codificador en este caso, y proporciona una ertiqueta de salida.

2.7.1. Método de evaluación de resultados

Idealmente, el rendimiento de un sistema SED debería medirse por la satisfacción del usuario en la aplicación en la que se utiliza para responder las expectativas de calidad del usuario, como se explica en [37]. Durante el desarrollo, se utilizan métricas computacionales más sencillas para evaluar el rendimiento comparando el resultado del sistema con la anotación de referencia. La comparación proporciona estadísticas sobre la detección correcta y errónea y calcula métricas comunes utilizadas en la clasificación de patrones, lo cual permite una comparativa entre distintos trabajos. El sistema emite una decisión para cada segmento de audio, y este resultado se compara con la anotación de referencia para determinar el rendimiento del sistema.

La comparación entre la anotación de referencia y el resultado del sistema utiliza los indicadores binarios de actividad a nivel de segmento para contar los eventos detectados correcta y erróneamente en términos de verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN), que pueden utilizarse para calcular las distintas medidas de rendimiento. Las métricas utilizadas habitualmente en la clasificación de patrones incluyen precisión (P), tasa de verdaderos positivos '*recall*' (R), puntuación F, índice de error (ER) o tasa de falsos positivos (FPR). Se definen como:

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$
$$F = \frac{2PR}{P + R}$$
$$FPR = \frac{FP}{FP + TN}$$

Para ver la totalidad de errores en conjunto se emplea el índice de error, que tiene en cuenta el error de sustitución (S), las inserciones restantes (I), que son los falsos positivos no contabilizados en S, y las supresiones restantes (D), que son los falsos

negativos no contabilizados en S. Este error se calcula en base al número de eventos de referencia (N).

$$ER = \frac{S + D + I}{N}$$

La curva '*receiver operating characteristic* ' (ROC) caracteriza el rendimiento del modelo mediante una representación gráfica, mediante la relación entre '*recall*' y FPR a medida que varía el umbral de decisión del clasificador. Un modelo perfecto tendría una curva que pasa por el punto superior izquierdo de la gráfica (TPR = 1 y FPR = 0), lo que indica que clasifica correctamente todos los positivos y no comete errores con los negativos. Además, el área bajo la curva, AUC ('*Area under the curve*') se usa como un resumen numérico del rendimiento del clasificador. Un valor de AUC de 1 indica un modelo perfecto, mientras que un AUC de 0.5 sugiere un modelo que no es mejor que una clasificación aleatoria.

Esta métrica indica la probabilidad de que se clasifique un negativo como positivo de entre todos los negativos. Con el fin de que la subjetividad del etiquetado no perjudique la salida, se añade cierto grado de desalineación, conocido como '*collar*'. De forma que para considerarse como TP debe tener la misma etiqueta que la referencia y sus límites temporales deben estar dentro de los límites temporales permitidos respecto a la referencia. Esta métrica se ha desarrollado hacia un nuevo punto de vista, donde se redefinen los verdaderos y falsos positivos en función de la intersección entre la salida del sistema y la referencia, de forma que la evaluación tolere la segmentación de los eventos. Este método se conoce como '*polyphonic sound detection score*' (PSDS), que define el área normalizada bajo la curva ROC.

Las definiciones anteriores se han adoptado para todos los sistemas SED, de forma que se puedan evaluar de forma común aunque tengan áreas distintas. En este reto cabe destacar el '*Detection and Classification of Acoustic Scenes and Events*' (DCASE), que se trata de un importante reto comunitario que ofrece tareas para la evaluación pública de los métodos.

Capítulo 3

Codificación de audio

En este capítulo se comienza con el estudio práctico de la caracterización a partir de los audios. Para ello se debe escoger un conjunto de datos, preprocesarlos y tratarlos para introducirlos en la red neuronal. Como se ha visto en el capítulo anterior hay estudios previos donde se trata la extracción las características empleando espectrogramas o derivados. En este caso se busca extraer las características directamente de la onda sonora y discretizarlo en *tokens*, de forma que se empleen los *tokens* no como representaciones intermedias del sistema codificador/decodificador, sino como objetivo para definir las características.

Los tokens pueden ser de tipo semánticos o acústicos [38], los primeros tienen limitaciones cuando se busca reconstruir la señal, mientras que los segundos son más fieles a la onda, lo que permite una síntesis de gran calidad. En este trabajo se busca la mejor caracterización posible, por lo que estos segundos son más convenientes.

El modelo preentrenado de *SoundStream* permite la tokenización que se busca, mediante el uso de RVQ y un vocabulario de N símbolos. Con este objetivo se va a seleccionar una base de datos y un modelo preentrenado basado en *SoundStream* como punto de partida.

3.1. Consideraciones iniciales

3.1.1. Elección de base de datos

En base al estudio de las bases de datos visto en el apartado anterior, se van a emplear varias bases de datos distintas para poder comparar resultados y realizar un estudio completo mediante el uso de las herramientas disponibles.

Además, hay una característica de las bases de datos que no se ha tenido en cuenta en la clasificación anterior, es si son monofónicas o polifónicas. Un escenario real se trata de un escenario polifónico, pero para que todo el sistema funcione en esas condiciones, debemos asegurar en un primer paso que funcione para fuentes monofónicas. En algunos

estudios tratan como fuentes monofónicas a fuentes polifónicas con un método de multiclase y etiqueta única, por lo que en cada instante de elección se selecciona solo un sonido. Dado que nuestro objetivo es caracterizar cada uno de los sonidos de forma independiente, si se superponen y no lo tenemos en cuenta puede dar lugar a resultados erróneos, por lo que esta metodología no es válida [39]. En primer lugar, se emplea la librería *CHime-Home*, donde aparecen varios tipos de audio dentro de cada archivo, pero nunca superpuestos, es decir, es multilabel y monofónica. Los datos de la librería están ofertados para uso público en [34].

En segundo lugar, se va a emplear *Urban Sound 8k*, este *dataset* ha sido extensamente empleado en distintos estudios, lo cual va a permitir una comparativa directa con otros trabajos. Y finalmente se va a completar con el uso de *ESC-50*, siguiendo el criterio anterior.

3.1.2. Elección de SoundStream preentrenado

En el estado del arte se expone la versión de *SoundStream* original, el cual se ha empleado en distintos trabajos con diferentes fines, a continuación se explican algunos de ellos y la motivación para el uso de uno en concreto.

Existen distintos modelos preentrenados que pueden ser de utilidad '*SoundStream for Pytorch*' [5], '*Lyra SoundStream*' [40], '*AudioLM - Pytorch*' [38] y '*Encodec*' [15].

SoundStream for Pytorch

Es una implementación no oficial disponible preentrenada a 16kHz, y con código abierto para poder realizar un ajuste fino. El modelo se ha preentrenado usando LibriSpeech con NVIDIA T4 durante un total de 50 horas aproximadamente, siendo un modelo no causal.

AudioLM

Emplea un modelo mixto de SoundStream, del cual extrae tokens acústicos, y w2v-BERT [41] del que se extraen tokens semánticos. La primera parte puede ser de utilidad, pero la segunda es un extra, que va a consumir capacidad de cálculo y tiempo, y no es útil para este trabajo, por lo que se descarta como modelo a emplear.

Lyra

Se trata de modelo creado por *Google* que tiene como base *SoundStream*, y está enfocado al uso en móviles *Android*, aunque también tiene disponible una API para sistema operativo *macOS*, *Linux* y *Windows*. Este modelo es dependiente de *Bazel*, un

constructor propio de *Google*, por lo que se necesita esta herramienta extra para poder usarlo. Además, viendo los resultados expuestos en [4] a mismo bitrate, tiene peores prestaciones que *SoundStream*, según la métrica *MUSHRA*.

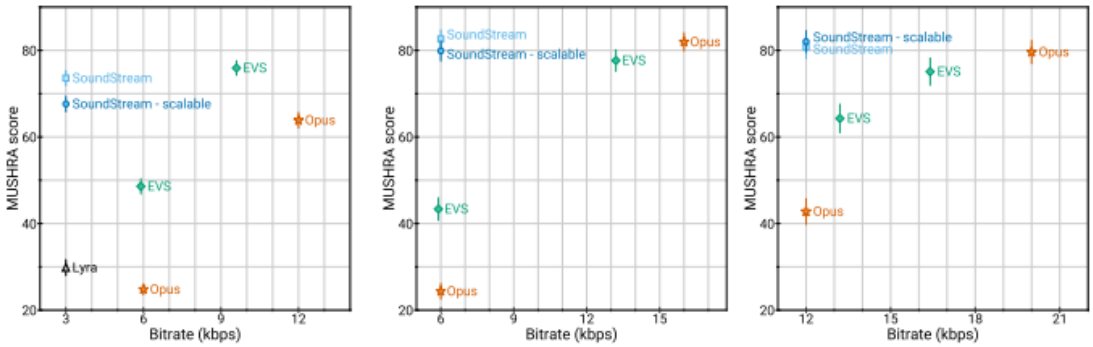

Figura 3.1: Comparativa de distintos modelos, [4]

En esta comparativa se nombra a *Opus* [7] y EVS (Enhanced Voice Services) [8], en ambos casos, se trata de codificadores con un propósito inicial concreto, distinto al de este trabajo, además se ve que tienen resultados similares o peores al de *SoundStream*, en función del bitrate empleado.

Encodec - Meta

Se trata de modelo creado por *Facebook* (actualmente *Meta*) que tiene como base *SoundStream*. Este modelo se centra en aumentar la compresión del audio manteniendo alta fidelidad. Ha sido entrenado con varias librerías de distintos tipos de audio, para realizar un trabajo lo más general posible. El objetivo del proyecto no es una alta fidelidad y este modelo conlleva una mayor complejidad respecto a los anteriores.

Por todo ello, se va a continuar con SoundStream for Pytorch.

3.2. Modelo encoder-decoder: SoundSream

En esta sección se describe cómo se ha preparado la base de datos y se ha empleado el modelo preentrenado escogido, además de mostrar resultados del uso conjunto.

3.2.1. Procesado de la base de datos

La base de datos contiene 3 tipos de archivos, el audio original con frecuencia 48kHz, el audio a 16kHz submuestreado y las etiquetas en formato `csv`. Dado que el modelo emplea como entrada audio a 16kHz, el primer paso es eliminar el audio de 48kHz, ya que son los mismos pero con distinta frecuencia de muestreo, y normalizar los audios

que se van a emplear. Por otro lado, los archivos de etiqueta llevan un mayor procesado, el cual queda descrito en el siguiente capítulo junto al uso de las mismas para el bloque clasificador.

3.2.2. Uso del modelo preentrenado. Ajuste fino.

En primer lugar se debe testear el modelo ofrecido, para ello es necesario preparar los requisitos. A nivel de python son necesarias las librerías `torchaudio` y `torch`, además se necesita instalar en el dispositivo los programas `sox` y `ffmpeg`. Una vez se cumplan los requisitos, se puede emplear el modelo SoundStream for Pytorch[5]. Tras pasar un audio de entrada por el modelo, vemos el resultado de la figura 3.2 a la salida del decodificador. Como entrada se ha tomado el audio `7061-6-0-0.wav` del conjunto de '*Urban Sound 8k*'.

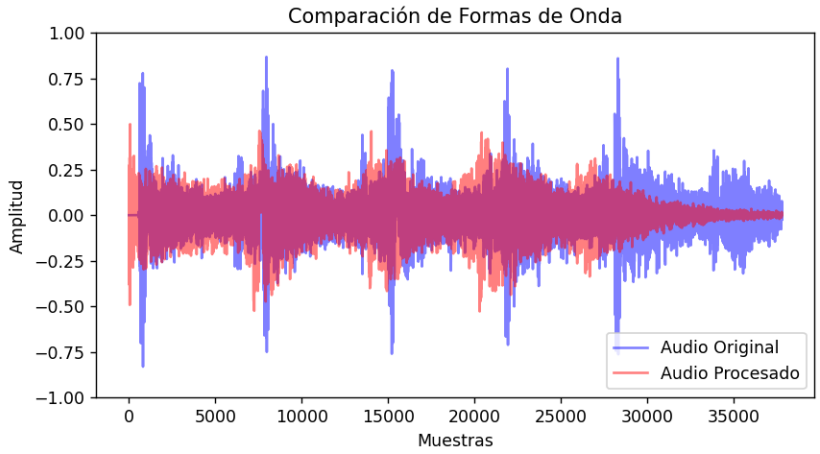

Figura 3.2: Comparativa forma de onda a la entrada y salida, *SoundStream pytorch*

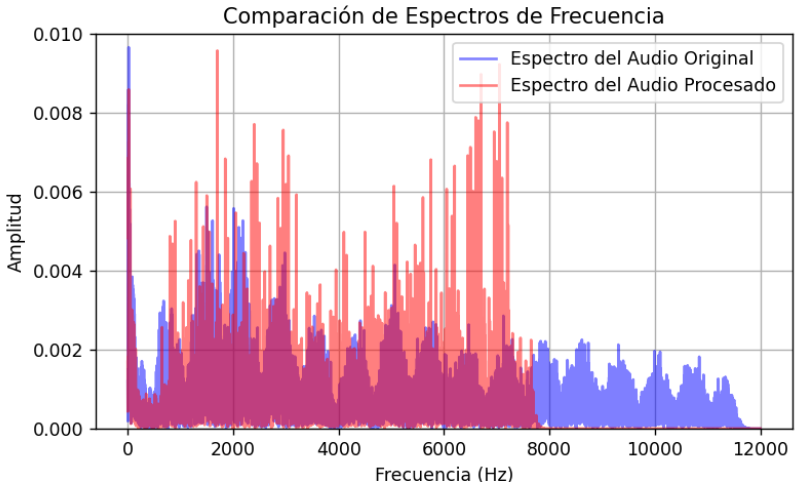

Figura 3.3: Comparativa espectro del audio a la entrada y salida, *SoundStream pytorch*

A partir de un exhaustivo estudio del modelo inicial, al cual se hace referencia en

la literatura, se ha extraído la arquitectura del sistema 3.4.

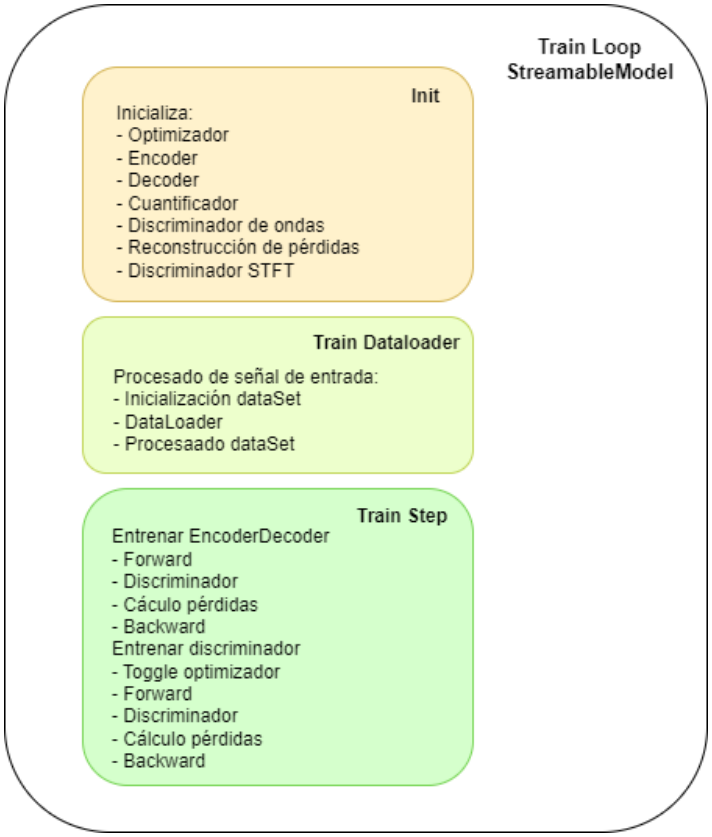

Figura 3.4: Estructura código, [4]

Empleando el código abierto como base, se va a comenzar con el diseño del ajuste fino. Se busca cambiar el modelo lo menos posible, así que se comienza con un enfoque conservador y se va incrementando gradualmente las capas descongeladas, esto ayuda a evitar el sobreajuste y a encontrar el equilibrio adecuado entre el rendimiento del modelo y el tiempo de entrenamiento. En el entrenamiento se debe entrenar tanto el modelo '*Encoder-Decoder*' como los discriminadores. En primer lugar, se va a testear congelando todas las capas y descongelando las dos últimas del '*encoder*', de forma que el modelo ajusta las características de nivel superior mientras mantiene las características de nivel más bajo aprendidas durante el pre-entrenamiento, después se puede variar el número de capas decongeladas en el '*encoder*', en el '*decoder*', y en los distintos discriminadores. A partir de aquí se hacen pruebas, variando el número de capas congeladas, la tasa de aprendizaje, el tamaño del lote y diversas variables del modelo.

Las pruebas iniciales comienzan con un ajuste del codificador, para ello se ha ajustado la variable '*num_embeddings*', con esto se busca modificar la secuencia embebida de salida del codificador, este número define de igual forma el tamaño de la entrada del decodificador. Esta prueba busca un ajuste mejor a las características de

los audios de entrada, de naturaleza muy distinta a los audios de entrenamiento del modelo. Con la misma idea, y de forma complementaria se modifica '*num_quantizers*', del vector de cuantificador residual, ya que la salida del codificador es la entrada del RVQ.

El modelo busca reducir el tiempo de computación al mínimo posible, para ello en lugar de emplear el audio de entrada completo, lo divide en segmentos más pequeños y los trata de forma independiente. La siguiente prueba va a ser mejorar la precisión del modelo para un mejor ajuste a los audios de entrada, a costa de aumentar el tiempo computacional, esto se logra mediante la modificación de '*segment_length*'. Sin embargo, no se han logrado mejoras con este cambio.

Para el entrenamiento se emplea la librería `lighning trainer` [42], que permite ajustar la cantidad de bits que emplea para representar los números de los cálculos en coma flotante, este parámetro se debe modificar para lograr un balance entre rendimiento y uso de la memoria. Puede seleccionarse la precisión de 64 ('*double precision*'), 32 ('*full precision*') o 16 ('*half precision*') bits. La precisión de 16 bits puede ser mixta, que combina 16 o 32 bits para optimizar el rendimiento sin perder mucha precisión, o verdadera, que emplea únicamente 16 bits. Aunque no todas las opciones se pueden emplear con todos los hardware, en este caso sí ha sido posible gracias al equipo disponible en el laboratorio de Audio y Vídeo del Departamento.

Con el fin de valorar los distintos cambios expuestos, se pueden comprobar varios parámetros:

- ***'num_replaced'***: RVQ reemplaza los códigos cuantificados buscando la mejor opción. En el estudio original se cambian por una de las entradas de forma aleatoria, mientras que en este desarrollo se emplea con la media de los valores obtenidos como resultado. Según explica el propietario de este código, los valores óptimos de los números reemplazados debería ser de 0.3.

- **Entropía**: RVQ calcula la entropía de sus cuantificadores, su objetivo es reducirla al máximo, pero serían valores aceptables por debajo de 6.8.

- **Pérdidas**: Las pérdidas del modelo se calculan como una media de las pérdidas de reconstrucción de la señal en el decodificador y de las pérdidas del discriminador.

Conociendo estas métricas, se puede evaluar el modelo, al tratarse de un modelo de audio, una de las métricas más importantes es la percepción humana, por lo que es el primer parámetro a evaluar es este, se escucha el audio de salida y se detecta una gran cantidad de ruido. A partir de aquí se comprueban las pérdidas, se puede observar que aunque tienen una tendencia decreciente, no llegan a estabilizarse de forma consistente, y en ninguna de las partes se obtiene un valor de pérdida satisfactorio.

Para las siguientes gráficas se emplea una tasa de aprendizaje de 10^{-4}, tamaño del lote de 32, descongelado total de las capas (codificador, decodificador y discriminadores), precisión de 64 bits, tamaño de segmento de 32270 y tamaño de la SFTF de 1024. Se emplean como ejemplo representativo del modelo, ya que para otras pruebas el resultado era similar. Se pueden ver todas las pruebas en A.

Figura 3.5: Pérdidas discriminadores

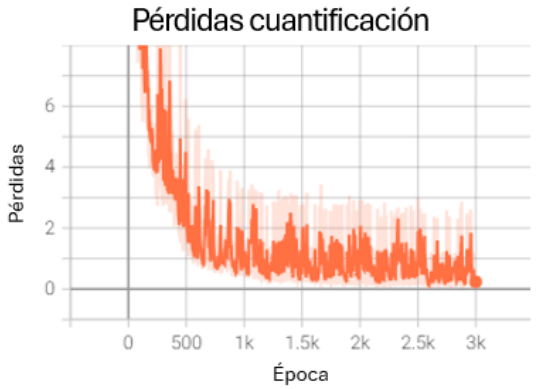

Figura 3.6: Pérdidas cuantificación

Figura 3.7: Pérdidas totales del modelo

Por ello, se decide indagar en el error, encontrando que la entropía tiene valores aceptables y funciona de la forma esperada. Sin embargo, el reemplazo de los códigos no converge, y tiene picos enormes, donde prácticamente está reemplazando todos los códigos, que son 1024 en esta prueba.

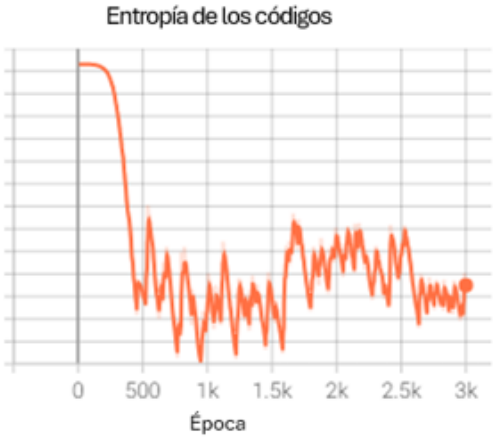

Figura 3.8: N. códigos reemplazados

Figura 3.9: Pérdidas cuantificación

Mediante este estudio se ha comprobado que el modelo '*SoundStream for Pytorch*' no es un buen generalizador, por lo que es necesario avanzar un paso más, mediante el uso de modelos más complejos, como puede ser *Encodec*.

3.3. Encodec. Modelo SoundStream desarrollado por Meta

Encodec ha sido entrenado con un amplio abanico de tipos de audios de entrada, por lo que es mucho más generalizable que el modelo anterior, y soluciona este problema. Con el fin de emplear este modelo se deben preparar los audios previamente, que en esta caso consiste en convertir los audios a monocanal, cambiarles la frecuencia de muestreo a 24kH y normalizarlos.

Para poder comprobar su eficacia se escogen varios audios aleatoriamente de entrada, y se pueden ver los resultados de las figuras 3.10 y 3.11. Además se puede realizar una evaluación subjetiva escuchando los audios y comparando a nivel humano su similitud, con lo que se concluye que este modelo es suficientemente bueno. Como entrada se ha tomado el audio `7061-6-0-0.wav` del conjunto de '*Urban Sound 8k*', con el fin de compararlo con el modelo anterior.

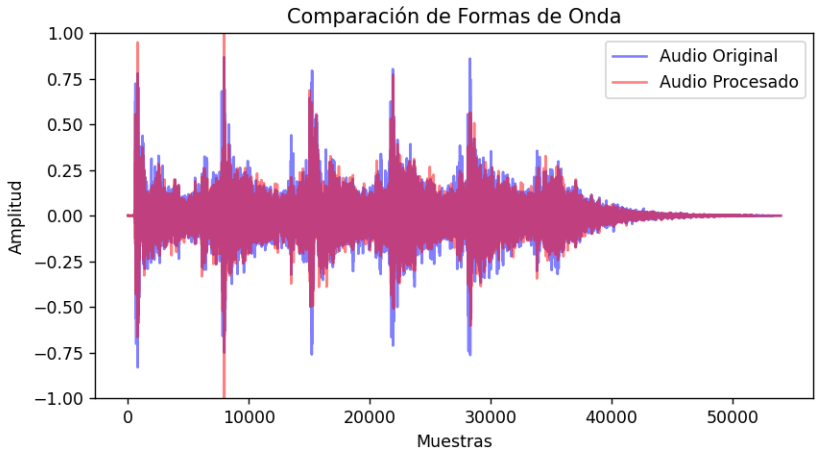

Figura 3.10: Comparativa forma de onda a la entrada y salida, *Encodec*

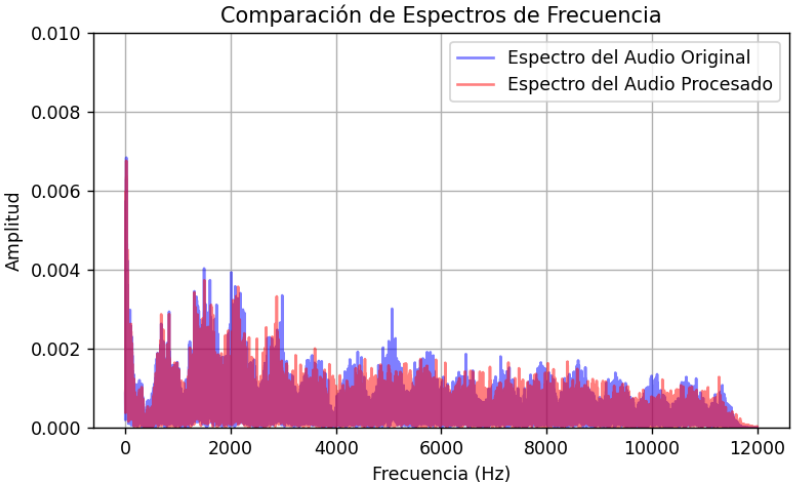

Figura 3.11: Comparativa espectro audio a la entrada y salida, *Encodec*

En este punto se ha comprobado que el modelo es generalizable para todo tipo de eventos de sonido, cabe destacar que este modelo se ha entrenado con dos bases de datos de este estilo *AudioSet* y *FSD50K*, es por ello que se esperaba que funcionase correctamente para otras bases de datos con el mismo tipo de sonidos.

3.3.1. Modelo preentrenado

El modelo preentrenado se encuentra alojado en [6], este contiene varios archivos python con el código completo, con la estructura que se puede ver en la figura 3.12.

Por un lado, están los archivos representados en verde, que contienen todas las herramientas necesarias para el desarrollo del modelo. Por otro lado, se encuentran los archivos `main.py`, `model.py` y `compress.py`, a los cuales se debe prestar más atención. Estos tres se encargan del entrenamiento del codificador y decodificador, siguiendo el flujo de flechas en el orden que se indica en la figura 3.12.

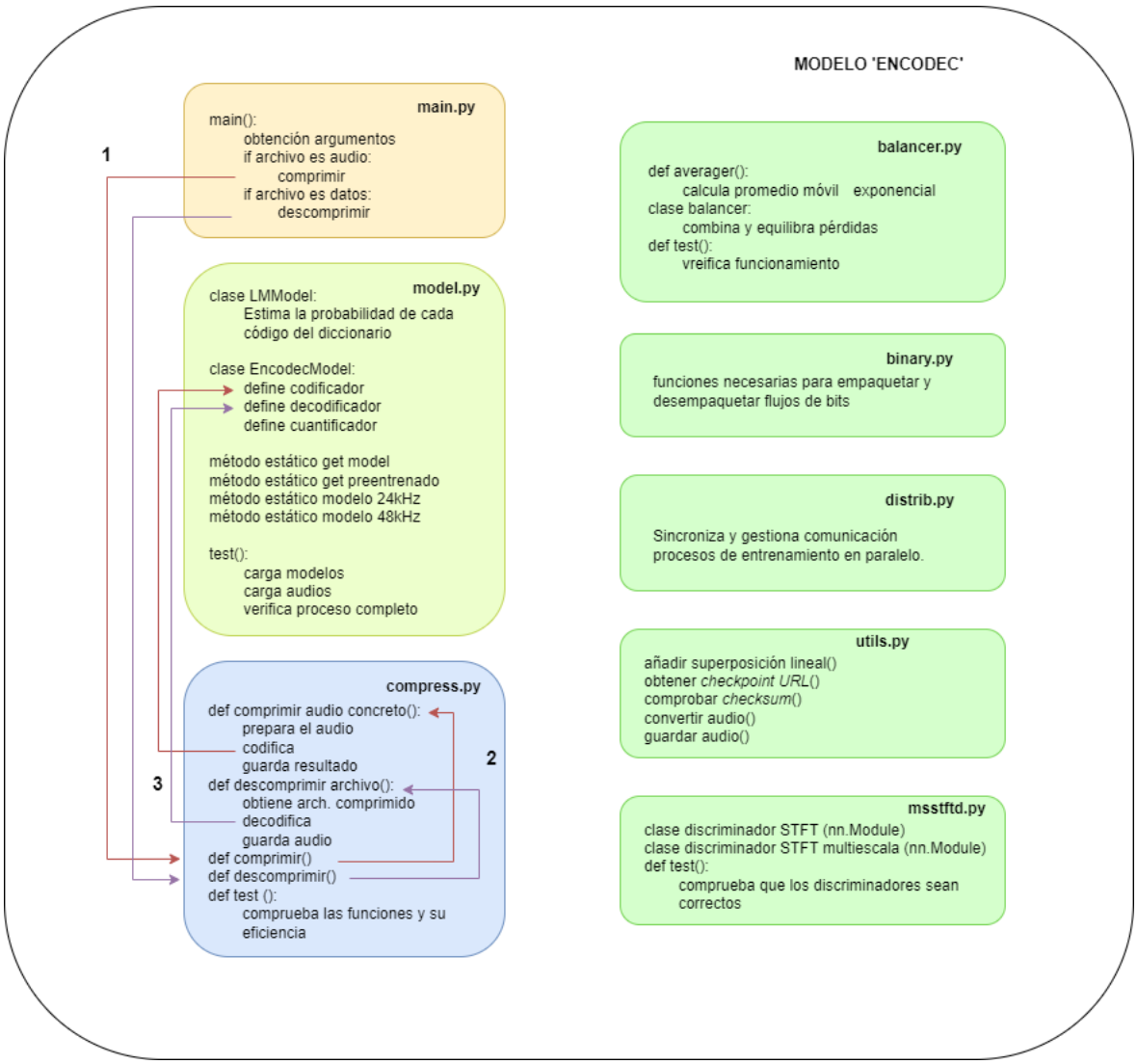

Figura 3.12: Estructura código fuente *Encodec*

Además, cabe destacar el archivo `msstftd.py`, que incluye los discriminadores, los cuales son entrenados en conjunto con el codificador y decodificador, tal y como se ha explicado en el estado del arte.

Por otro lado, el modelo cuenta con un archivo `benchmark.py`, el cual se emplea para evaluar el rendimiento de codificación y decodificación en función del tiempo empleado en cada tarea. Los pasos que evalúa son: la carga del audio, realización de la codificación y decodificación, y finalmente evaluación de los tiempos requeridos para cada parte. Este código permite evaluar el modelo, por lo que se puede ver para los audios de ejemplo que proporcionan, y se extiende al testeo de un audio de la base de datos '*Urban Sound 8k*', concretamente con el audio `7061-6-0-0.wav`. El código se ha modificado levemente para adaptar a la longitud máxima de 4 segundos, duración de los audios del *dataset Urban Sound 8k*.

Proceso	Tiempo (s)
Codificación	1.414
Evaluación LM	1.503
AC codificación	0.244
AC decodificación	0.167
Decodificación	1.680

Tabla 3.1: Resultados modelo Encodec 24kHz, entrada de ejemplo

Proceso	Tiempo (s)
Codificación	0.931
Evaluación LM	4.606
AC codificación	0.332
AC decodificación	0.366
Decodificación	5.643

Tabla 3.2: Resultados modelo Encodec 48kHz, entrada de ejemplo

Proceso	Tiempo (s)
Codificación	1.198
Evaluación LM	1.578
AC codificación	0.23
AC decodificación	0.182
Decodificación	1.788

Tabla 3.3: Resultados modelo Encodec 24kHz, entrada audio de *Urban Sound 8k*

Donde AC se refiere a la clase *ArithmeticDecoder*, tal y como se define en el código, se utiliza para decodificar datos comprimidos utilizando un método de codificación aritmética. Esta decodificación queda definida internamente en el modelo, por lo que no se ha reflejado en el esquema anterior. Decodifica una secuencia de bits en símbolos utilizando la misma serie de CDF cuantizados (funciones de distribución acumulativa) que se usaron en la codificación, asegurando que se mantenga la integridad de los datos originales.

Comparando ambos modelos, el de mayor tasa de muestreo tiene tiempos mayores, es por ello que finalmente se decide emplear el de tasa de 24kHz. Además, al comparar los resultados de la base de datos empleada en este trabajo, con los ejemplos dados por *Meta*, se ven resultados muy similares.

3.3.2. Salida del modelo

Dado que no necesita ajuste fino, a diferencia de *SoundStream pytorch*, se va a proceder a extraer los tokens directamente de la salida del codificador. Este token se trata de una tupla (codebook, scale), donde 'codebook' es un tensor que representa los códigos discretos codificados para una parte del audio. Este es de tamaño [1, 32, n], donde el primer valor corresponde al tamaño del lote, el segundo al número de 'codeblocks' y el tercero a la longitud del marco temporal, que variará en función de la longitud original de los audios.

El tensor salida del codificador es almacenado en archivos que serán la entrada al modelo clasificador. Se ha optado por esta división para reducir el tiempo de compilación, ya que este modelo ya está entrenado, mientras que el clasificador no. De esta forma se logra una mayor eficiencia al reducir el tiempo de codificación, que como hemos visto es entorno a un segundo por audio, lo cual son dos horas y tres cuartos para una base de datos de 10000 audios.

En definitiva, este modelo no solo garantiza una codificación de alta calidad, sino que también sienta las bases para un análisis de audio más profundo en etapas posteriores del estudio.

Capítulo 4

Validación experimental

Este capítulo se centra en la implementación y evaluación del clasificador neuronal, diseñado para identificar los distintos eventos de sonido. Se analiza el comportamiento de este modelo en las diferentes bases de datos expuestas, lo que proporciona una visión detallada de su rendimiento y eficacia. Se exploran aspectos fundamentales del diseño del clasificador, incluidos los parámetros de la red neuronal, las técnicas de entrenamiento utilizadas y los métodos de validación. Además, se discuten los resultados obtenidos según la entrada, destacando las fortalezas y limitaciones del clasificador en diversos contextos.

4.1. Datos de entrada al modelo

A continuación, se detalla cómo se preparan las etiquetas en función de su fuente para introducirlas en el modelo. Estas etiquetas procesadas, junto con la salida del codificador, serán serán la entrada para el modelo que se entrenará, validará y probará.

4.1.1. Aumento de datos

Las bases de datos seleccionadas presentan un desbalanaceo entre las distintas clases que las componen, para solucionarlo se ha aplicado aumento de datos para las distintas entradas.

Este proceso se ha realizado usando la librería '*audiomentation*' [43], que proporciona funciones para las distintas formas de aumento para audios. Gracias a ella se implementan las técnicas de ruido aleatorio, estiramiento temporal, cambio de tono y escalado temporal, de las expuestas en 2.6.1. Para ello se debe seleccionar la carpeta a aumentar y el número de audios que se quiere obtener, y el código implementado realiza el aumento aplicando una o varias de las técnicas con parámetros aleatorios dentro de los márgenes preestablecidos.

4.1.2. Procesado de *CHime-Home*

Procesado de las etiquetas

Este '*dataset*' proporciona un archivo de extensión '`.csv`' por cada audio de entrada, el cual contiene el proceso de etiquetado que se ha seguido, según el formato descrito en 4.1.2.

Nombre	**Descripción**
Segmentname	Nombre del segmento de audio del que se extrae el 'chunk'
ChunkNumber	El desplazamiento dentro del segmento de audio del que se obtuvo el 'chunk', en unidades de 4 segundos.
FrameStart	El desplazamiento dentro del segmento de audio del que se obtuvo el fragmento, en fotogramas
Annotation_a1	La cadena de anotación del 'chunk', definido por el anotador A1
Session_a1	La sesión de anotación del 'chunk', definido por el anotador A1
Annotation_a2	La cadena de anotación del 'chunk', definido por el anotador A2
Session_a2	La sesión de anotación del 'chunk', definido por el anotador A2
Annotation_a3	La cadena de anotación del 'chunk', definido por el anotador A3
Session_a3	La sesión de anotación del 'chunk', definido por el anotador A3
Majorityvote	La cadena de anotación del chunk, obtenida aplicando un voto mayoritario sobre la presencia de cada etiqueta.
Chunkname	Identificador único del fragmento, obtenido a partir de los campos segmentname y chunknumber

Tabla 4.1: Anotaciones sobre cada 'chunk'

Cabe destacar que algunos de los archivos de etiquetas estaban incompletos, o incluso eran erróneos, por ejemplo, pueden estar vacíos o con caracteres que no corresponden a ninguna categoría, por ello se ha realizado una criba eliminando los que no proporcionan información, resultando 6020 audios con etiquetas sin error.

La información aportada por el proveedor es mayor que la necesaria para el análisis y, además, la presentan en un formato legible para las personas, tabla 4.1.2, pero que necesita una interpretación mayor a nivel de código. Por ello se procesan los archivos para transformarlos en un modelo *one-hot* y tener como entrada solo valores '1' o '0', de esta forma será más sencillo trabajar con ellos posteriormente.

segmentname	CR_lounge_200110_1601.s0
chunknumber	0
framestart	0
annotation_a1	cv
session_a1	2.0
annotation_a2	cv
session_a2	4.0
annotation_a3	cv
session_a3	5.0
majorityvote	cv
chunkname	CR_lounge_200110_1601.s0_chunk0r

Tabla 4.2: Ejemplo de tabla de anotación

Esta tabla se traduce a un vector, en este caso su equivalencia sería:

$$[1, 0, 0, 1, 0, 0, 0, 0, 0]$$

Donde las el orden de las clases es el siguiente, respetando el código de clases descrito en 2.6.

$$[c, m, f, v, p, b, o, S, U]$$

En este punto las etiquetas están listas para ser tratadas por el clasificador.

Estudio de la distribución del *dataset*

Con los datos procesados es interesante realizar un estudio sobre su distribución para conocer las limitaciones del sistema. La distribución de los datos en las distintas clases no es equitativa, ya que algunas clases aparecen pocas veces en los audios y otras clases son positivas en un mayor porcentaje de los audios. Por otro lado, para el entrenamiento se van a separar los audios en tres grupos: de entrenamiento, de validación y de testeo, con un porcentaje de audios empleados para categoría de cada grupo del 60 %, 20 % y 20 %, respectivamente. Por ello se puede clasificar los datos según dos agrupaciones, por un lado la clase inicial a la que pertenecen, y por otro lado la fase del entrenamiento en la que se van a emplear.

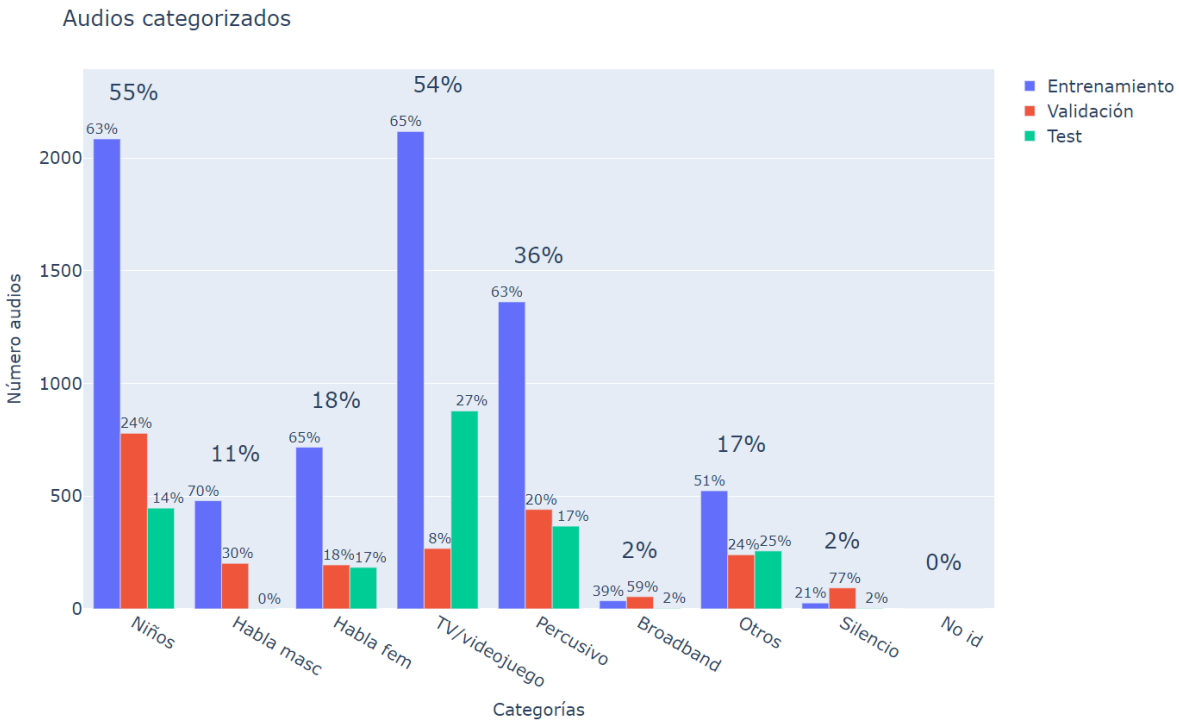

Figura 4.1: Histograma distribución audios por categorías y uso

En esta gráfica podemos ver los porcentajes de aparición de cada una de las categorías, por ejemplo, para los 6020 audios, habría habla de niños en 3311 de ellos, para estos porcentajes se debe tener en mente que los audios son '*multilabel*'. Y en cuanto a la clasificación del uso en el entrenamiento, los porcentajes indican para qué se usan esos positivos, por ejemplo de los 3311 positivos, 2085 se emplearán para entrenar, 779 para validar y 447 para test. Además, en el eje 'y' se puede ver el número de audios que incluye cada subcategoría.

La distribución no es equitativa para ninguna de las dos clasificaciones. Por un lado, la cantidad de apariciones para cada una de las clases, es muy distinta, destacando especialmente 'Broadband' y 'Silencio' con porcentaje del 2 %, que corresponde a 90 y 103 positivos, la falta de datos para estas clases va a dificultar el aprendizaje posteriormente.

Por otro lado, la división de audios para cada uno de sus usos se ha realizado por orden alfabético de los datos originales, lo cual descompensa las clases, siendo más visible en 'Habla masculina'. Para mejorar esta distribución, se plantea reordenar el uso de los datos de entrada para que dentro de cada clase quede una clasificación lo más similar a 60-20-20. Finalmente, se logra como mejor distribución la presentada en la figura 4.2, donde se ha ordenado para las clases mayoritarias de forma exacta, pero de las que se tienen menos datos se ha logrado un peor resultado, como son 'Habla masculina' y 'Broadband'.

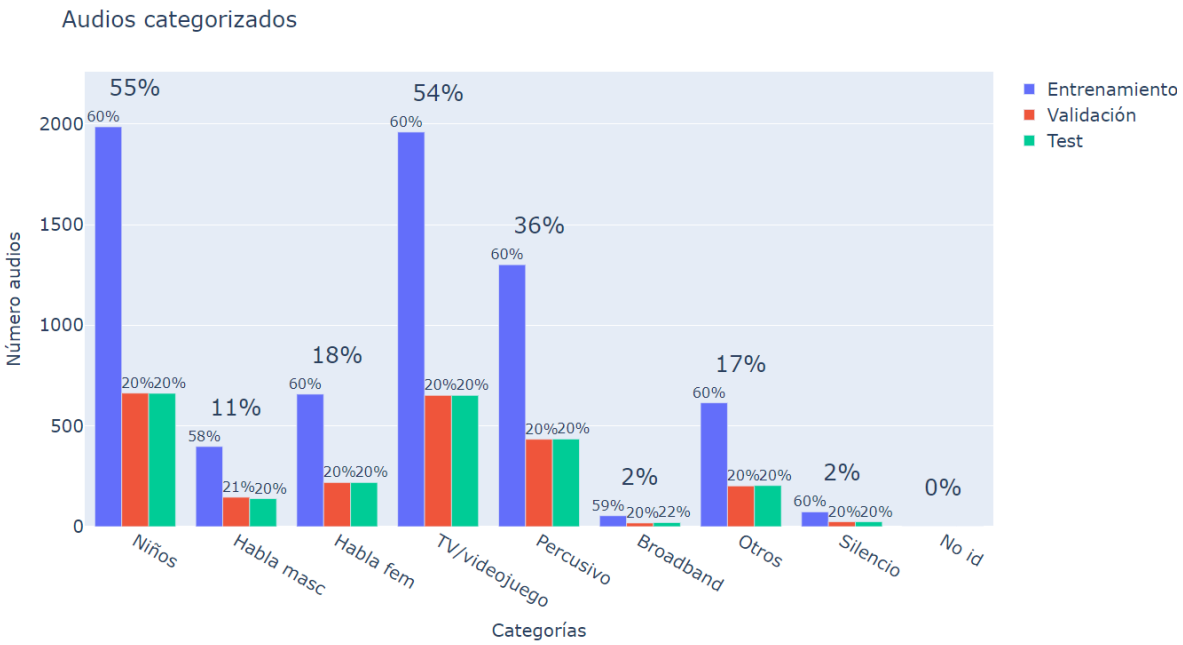

Figura 4.2: Histograma distribución audios por categorías y uso

4.1.3. Procesado de *Urban Sound 8k*

Procesado de los datos de etiquetado

Este *dataset* facilita una única tabla con la clasificación de todos los audios, que incluye las columnas según la tabla 4.3.

slice_file_name	Nombre del archivo con extensión *.wav*
start	Segundo en el que comienza (precisión centenas de milisegundos)
end	Segundo en el que finaliza (precisión centenas de milisegundos)
salience	Valoración (subjetiva) de la procedencia del sonido (primer o segundo plano)
fold	Carpeta que contiene el archivo
classID	Número identificativo de la clase
class	Nombre de la clase que aparece

Tabla 4.3: Columnas descriptivas de los audios del '*dataset*'

Todos los audios se han dispuesto de forma que tienen una duración de 4 segundos, mientras que en las columnas '*start*' y '*end*', se pueden encontrar valores superiores. Esto se debe a que pertenecen a audios de mayor longitud, los cuales han sido segmentados previamente en estos audios, de forma que cada audio de entrada, es en realidad un fragmento de un audio de duración superior. Dado que la fuente de datos facilita la segmentación de audios, no es necesario realizarla por el modelo clasificador.

Este archivo se procesa hasta crear vectores 'one-hot', de igual forma que en el apartado anterior, se relaciona cada uno con un audio. Con la diferencia de que en cada vector solo puede contener una clase, por lo que serán todas nulas excepto una.

Estudio de la distribución de la base de datos

Como se ha comentado anteriormente, la distribución de este *dataset* está ampliamente estudiada, de forma que la misma base de datos se distribuye en carpetas para emplearlas con el método de validación cruzada, con el fin de que distintos trabajos puedan compararse fácilmente. La validación cruzada consiste en emplear 9 de las 10 carpetas que facilitan como datos de entrenamiento y validación, y la restante para el testeo, rotándolas todas y realizando 10 entrenamientos distintos, empleando como resultado la media de los 10 entrenamientos.

En este trabajo se propone emplear la distribución de dos formas distintas: una imitando al *dataset* anterior para que sea comparable, y una segunda forma siguiendo las indicaciones para poder compararlo con otros trabajos. De esta forma se logra una doble comparación en función del uso de los datos.

Primera distribución, finalidad comparativa con otros dataset. El primer estudio de los datos nos muestra que hay dos clases con muchos menos datos (ver figura 4.3), por lo que se deben balancear los datos para tener la misma cantidad en todas las clases. Por ello se realiza aumento de datos para 'car_horn', 'gun_shot' y 'siren', obteniendo como resultado unos datos totalmente balanceados (figura 4.4), esta distribución se empleará para la comparativa con el dataset 'CHime-Home'.

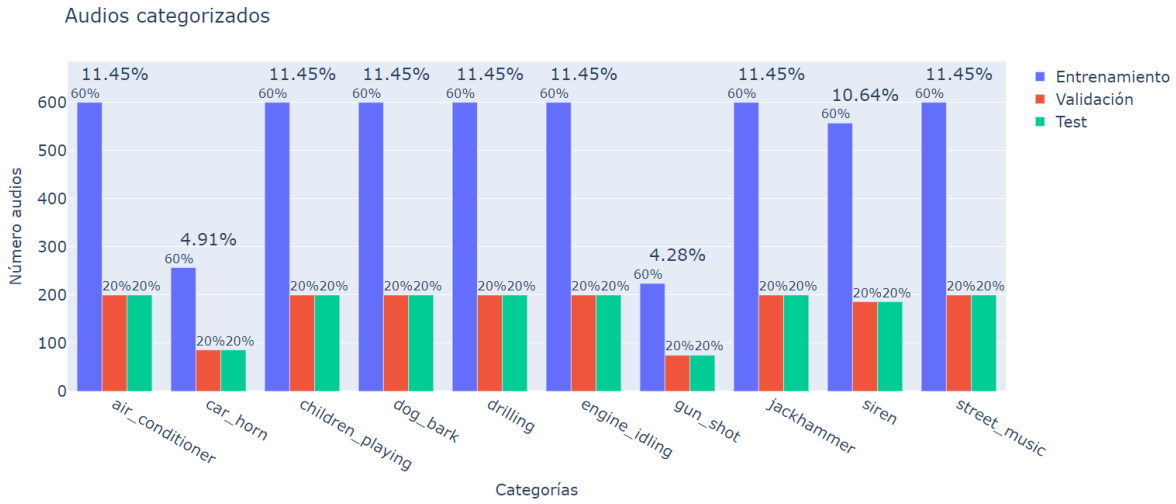

Figura 4.3: Distribución audios por categorías y carpetas

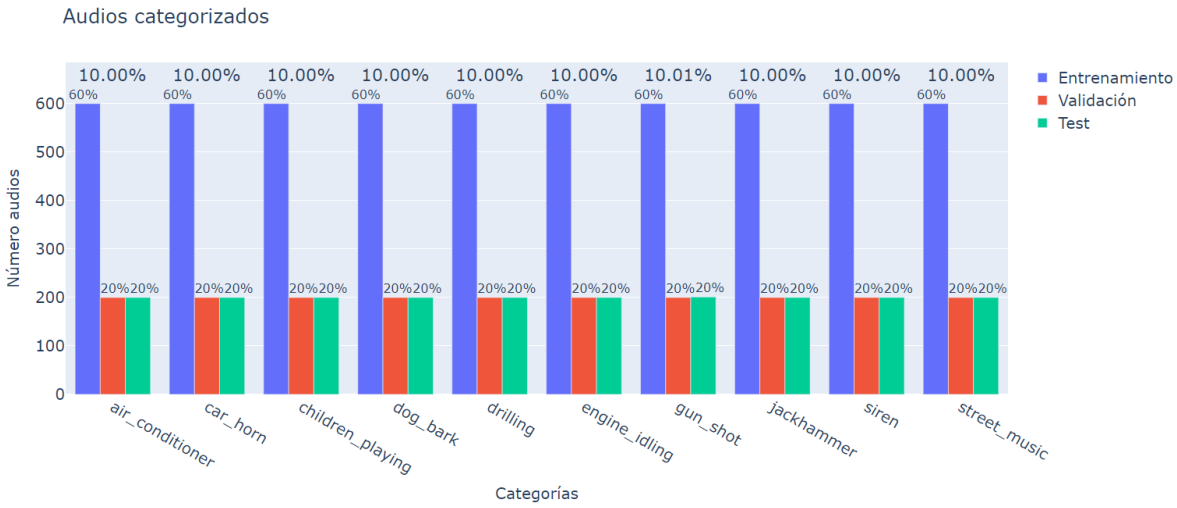

Figura 4.4: Distribución audios por categorías y carpetas, datos balanceados

Segunda distribución, finalidad comparativa con otros trabajos. Esta segunda evaluación consiste en ver la distribución de las clases dentro de las carpetas facilitadas, en ella se observa el mismo desbalanceo de datos. Con el fin de balancear los datos se incluyen los datos aumentados, mediante los métodos explicados en 4.1.1, y se distribuyen equitativamente en las carpetas, dando como resultado clases balanceadas, aunque no distribuidas uniformemente a nivel de las carpetas, siguiendo los consejos del autor [30] se mantiene esta distribución.

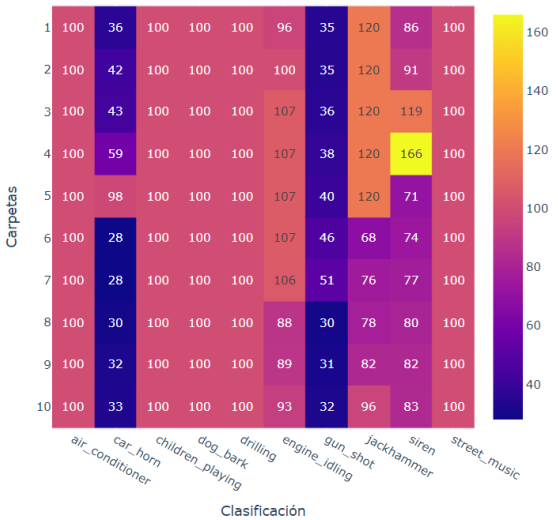

Figura 4.5: Distribución audios por categorías y carpetas

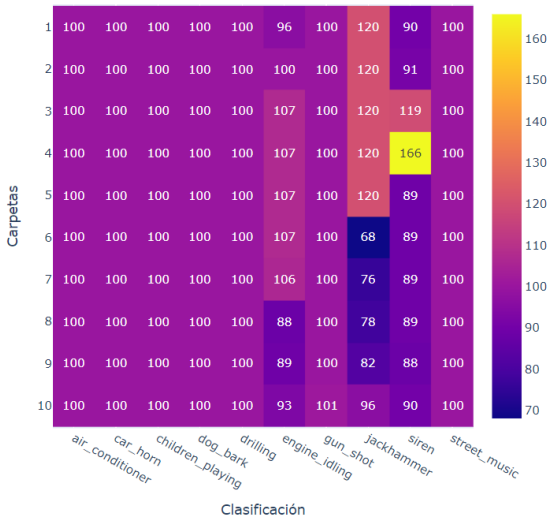

Figura 4.6: Distribución audios balanceados por categorías y carpetas

4.1.4. Procesado de *ESC-50*

Procesado de los datos de etiquetado

Los audios de este *dataset* poseen los metadatos que se muestran en la tabla 4.4. Este archivo se procesa hasta lograr un vector compuesto de vectores '*one-hot*', para poder emplearlo en el clasificador neuronal.

file_name	Nombre del archivo con extensión `.wav`
fold	Carpeta que contiene el archivo
target	Número identificativo de la clase
category	Nombre de la categoría que aparece
esc10	Indica si pertenece o no al dataset *esc10*
scr_file	Archivo original del que extrajo
take	Versión de la original escogida

Tabla 4.4: Columnas descriptivas de los audios del 'dataset'

Estudio de la distribución del '*dataset*'

Esta base de datos, al igual que la anterior, cuenta con un tratamiento previo extenso que lo clasifica en carpetas, además de tener validación cruzada. Por ello se van a emplear dos distribuciones, una con finalidad comparativa dentro de este trabajo, y otra para compararlo con trabajos externos.

Los audios se reparten de forma equitativa en 5 carpetas, y a su vez de forma balanceada en 50 categorías, por lo que no es necesario realizar un balanceo y corrección de la distribución a diferencia de los casos anteriores. Sin embargo, cuenta con tan solo 2000 audios, lo cual no es suficiente para un entreno de 50 clases, es por ello que en este caso se aplica una técnica de aumento de datos manteniendo las características de los datos iniciales, se aplican las técnicas indicadas en 4.1.1.

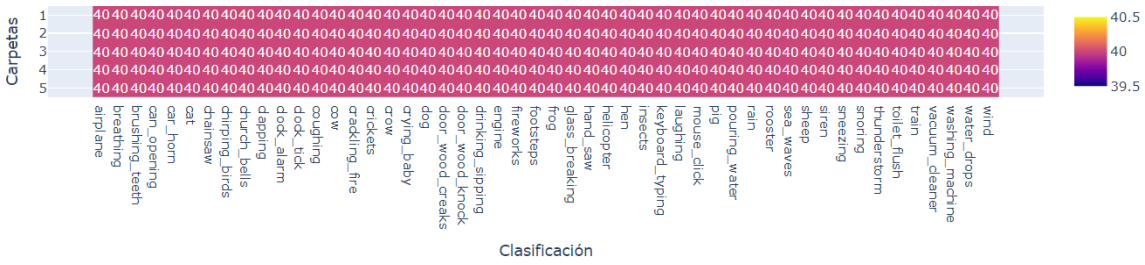

Figura 4.7: Distribución audios por categorías y carpetas, tras aumento de datos

4.2. Desarrollo del modelo clasificador

El primer paso para desarrollar el modelo es definir su estructura. Los requisitos iniciales son que la entrada debe estar totalmente conectada con la salida de la red anterior, y el tamaño de la salida viene determinado por el número de etiquetas de los datos. Esta salida contiene las probabilidades de aparecer o no cada una de las clases, que será necesario pasar por un umbral de decisión para ver el resultado final. Las capas intermedias se deben establecer para el modelo que dé mejores resultados.

Las capas ocultas del modelo están compuestas por una capa lineal, a definir el número de neuronas, con una de activación ReLu, la decisión de cuántas hace falta y el tamaño de las mismas se toma mediante el uso de la herramienta '*Ray Tune*' [44] encargada de buscar el mejor modelo con los parámetros de entrada que se le asignan. A partir de la siguiente estructura se realizan numerosas pruebas para determinar cuántas capas lineales y qué tamaño es lo óptimo para el objetivo.

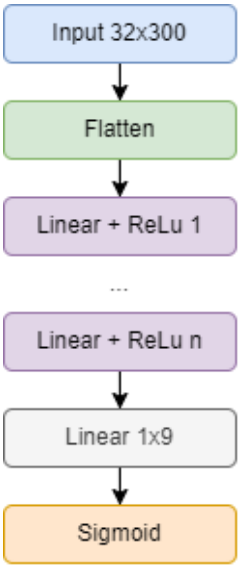

Figura 4.8: Estructura del modelo base, con los tamaños de entrada y salida adaptados a CHime-Home

En cada prueba se busca establecer los parámetros de la tabla 4.5, para lo cual se inicia con estos valores y se van acotando hasta encontrar el mejor modelo.

N. Capas	1 - 6
Tamaño cada capa	32 - 356
'Learning rate'	$10^{-6} - 10^{-2}$
N. épocas	200
N. Entren. paralelos	20
Umbral de decisión	0.5

Tabla 4.5: Punto de partida para entrenamientos

Partiendo del escenario anterior, la librería se encarga de buscar el mejor modelo de entre los 20 entrenamientos con hiperparámetros aleatorios escogidos dentro de los márgenes establecidos. De esta primera prueba se comprueba que los resultados son demasiado pesados, y se decide reducir a 10 el número de entrenamientos paralelos, con lo que los datos del resultado de la librería se reducen a 24.2 GB, tamaño considerable pero asumible. Esto implica que se han dividido las pruebas, ejecutando múltiples tandas de 10, todas las pruebas realizadas se pueden ver en el anexo A.

Primero se decide reducir el número de capas, dejando como máximo 4 capas, ya que como podemos ver en la figura 4.10, los resultados con 5 capas son bastante peores, con grandes valores de pérdida y baja precisión en comparación con los resultados de dos capas.

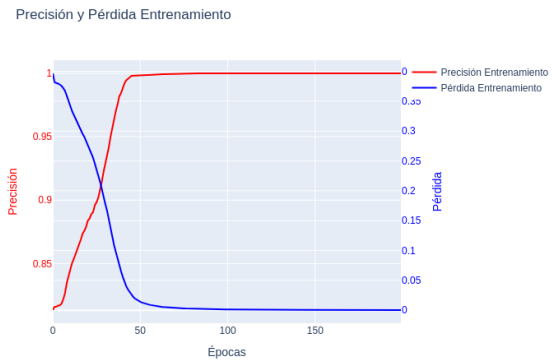

Figura 4.9: Ejemplo 2 capas

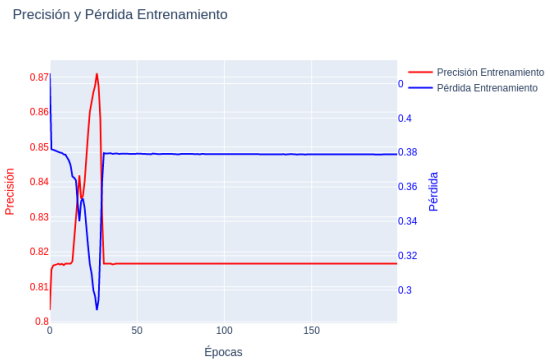

Figura 4.10: Ejemplo 5 capas

Por otro lado, se puede observar sobreajuste, ya que las líneas de evolución de entrenamiento y validación tienen una tendencia totalmente distinta (figura 4.11), para solucionarlo se implementa un sistema de '*early stopping*', encargado de parar el entrenamiento cuando se detecte esta desigualdad en el desarrollo de ambas curvas. De esta forma el número de épocas que se emplea para el entrenamiento varía para cada entrenamiento, pudiendo ser menor al máximo establecido si se detecta posible sobreajuste.

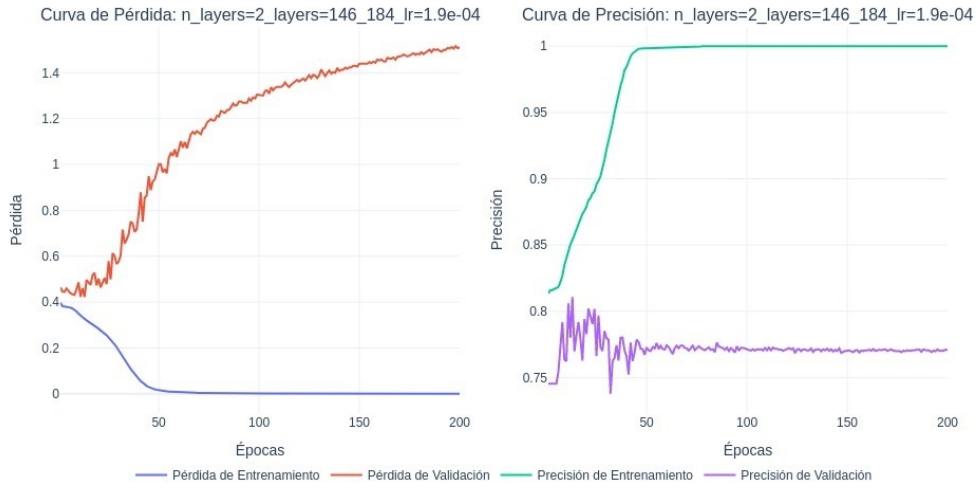

Figura 4.11: Ejemplo gráfica con sobreajuste

El resultado de los entrenamientos se puede comparar mediante las curvas ROC y '*precision-recall*', teniendo en cuenta los valores de AUC (*area under the curve*), se pueden ver ejemplos de ello en las gráficas 4.12 y 4.13. Además de estas gráficas, hay que tener en cuenta la precisión y la pérdida que se obtiene de cada modelo.

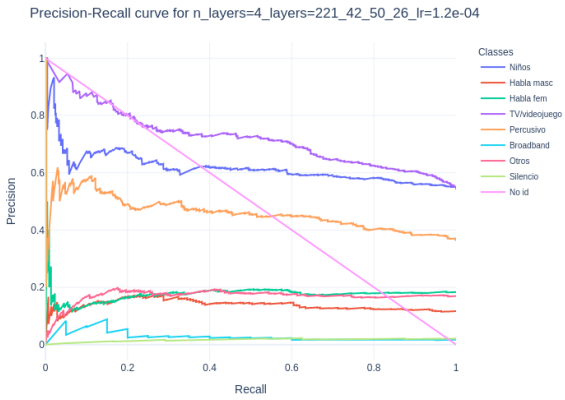

Figura 4.12: Relación 'precision-recall'

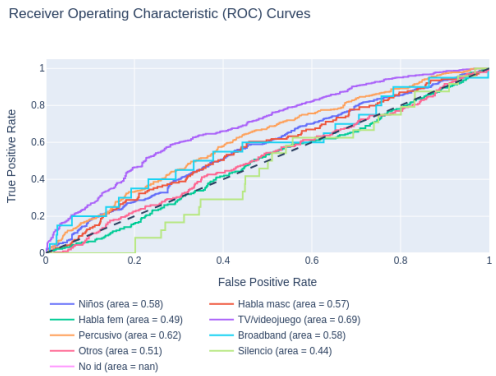

Figura 4.13: Curva ROC

Finalmente, se tiene un sistema que es capaz de encontrar la mejor arquitectura del modelo con un abanico de posibilidades delimitado. Además, se ha probado distintos criterios de cálculo de pérdidas, optimizadores y planificadores, y elegido la mejor opción para cada base de datos. Sobre este modelo se va a trabajar con los tres *datasets* explicados, para ver los resultados de cada uno de ellos.

Cabe destacar que se han realizado pruebas para otras arquitecturas, modificando las capas lineales por capas densas o incluso a arquitectura RNN, obteniendo resultados muy similares. Además de aplicar técnicas de pesos iniciados aleatoriamente, y de reducción de dimensionalidad con PCA (Análisis de Componentes Principales)

buscando simplificar los datos. Por ello se mantiene una arquitectura más simple, excluyendo de ella todo lo nombrado anteriormente, y se han puesto los esfuerzos en mejorar el resultado de la arquitectura totalmente conectada.

4.2.1. Resultados modelo escogido

Una vez se han realizado todas las pruebas se decide cuál es la mejor solución para cada base de datos, dando lugar a tres configuraciones que se reflejan en la tabla 4.6.

	CHime-Home	**US8k**	**ESC-50**
N. capas	4	2	2
Capa 1	221	211	339
Capa 2	42	103	198
Capa 3	50	-	-
Capa 4	26	-	-
Tasa aprend.	$1{,}2 * 10^{-4}$	$3{,}8 * 10^{-4}$	$1{,}3 * 10^{-4}$
Umbral	0.52	-	-
Criterio	`BCEWithLogitsLoss()`	`CrossEntropyLoss()`	`CrossEntropyLoss()`
Optimizador	SGD	Adam	Adam
Planificador	-	`ReduceLROnPlateau()`	`ReduceLROnPlateau()`

Tabla 4.6: Arquitectura modelo para distintos *dataset*

Comparativa entre las distintas bases de datos

Se puede ver las gráficas finales de los resultados para cada una de las bases de datos en las siguientes figuras. En primer lugar se muestran los resultados para la base de datos 'CHime-Home' (figuras 4.14, 4.15). Los valores de AUC están entorno a 0.5, y la precisión es de 81.83 %. Cabe destacar que para este caso no se muestran matrices de confusión, ya que en un mismo audio puede haber varias clases (aunque no en el mismo instante temporal) lo que dificulta la representación en una gráfica bidimensional del resultado.

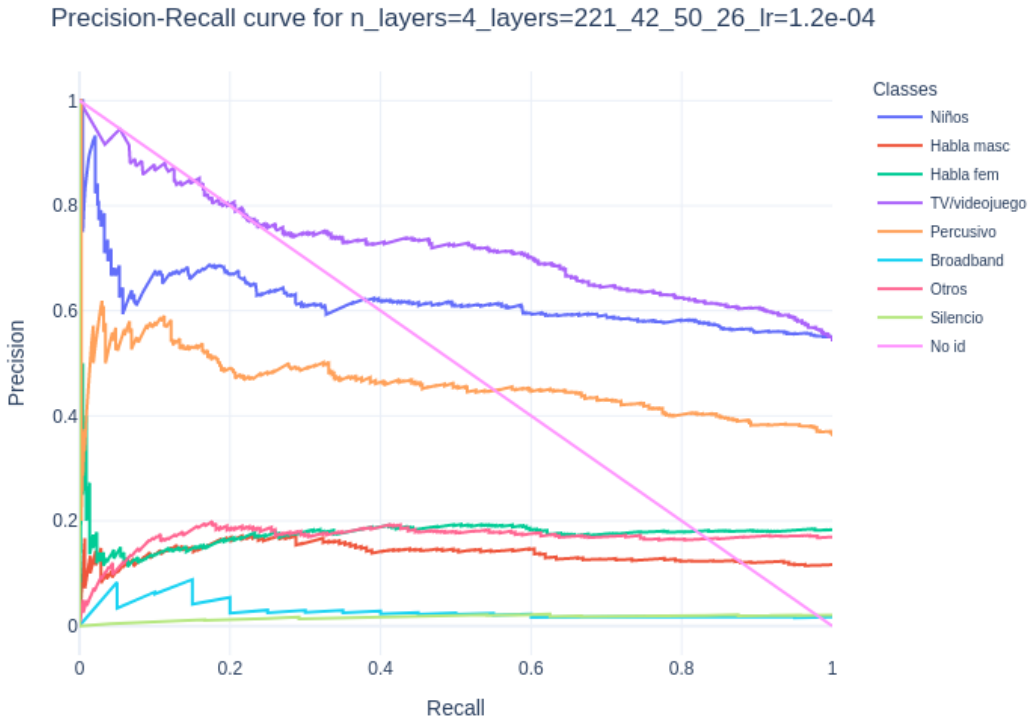

Figura 4.14: Relación 'precision-recall', Chime-Home

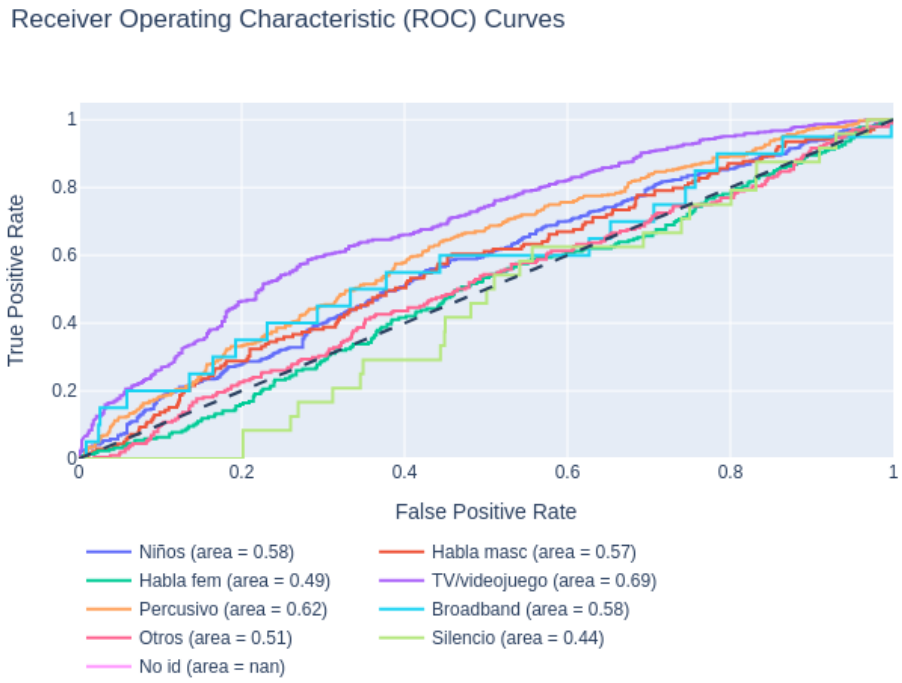

Figura 4.15: Curva ROC, Chime-Home

La siguiente base de datos es UrbanSound-8k (figuras 4.16, 4.17 y 4.18). En este caso se puede ver que la curva '*precision-recall*' tiene una tendencia mejor que la anterior, idea que se refuerza con los altos valores de AUC. Además, la matriz de confusión tiene una clara diagonal marcada. En cuanto a precisión, se logra un 84.4 %.

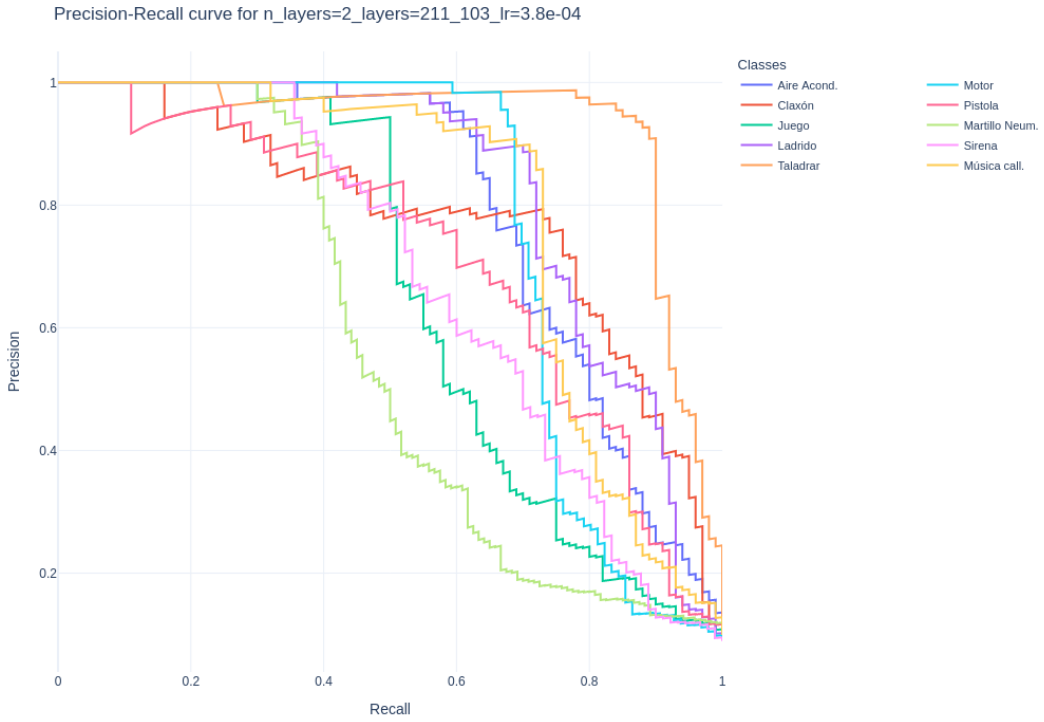

Figura 4.16: Relación 'precision-recall'

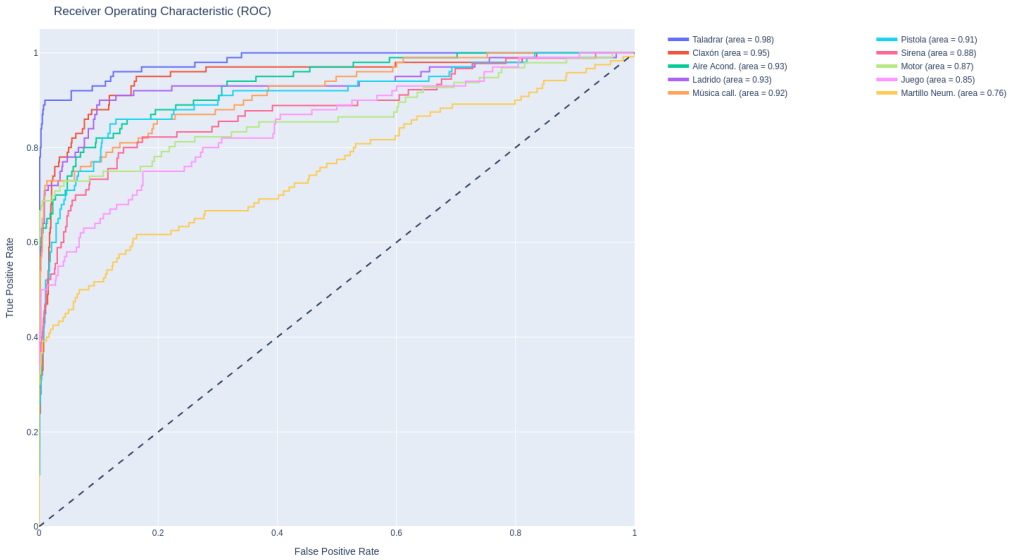

Figura 4.17: Curva ROC

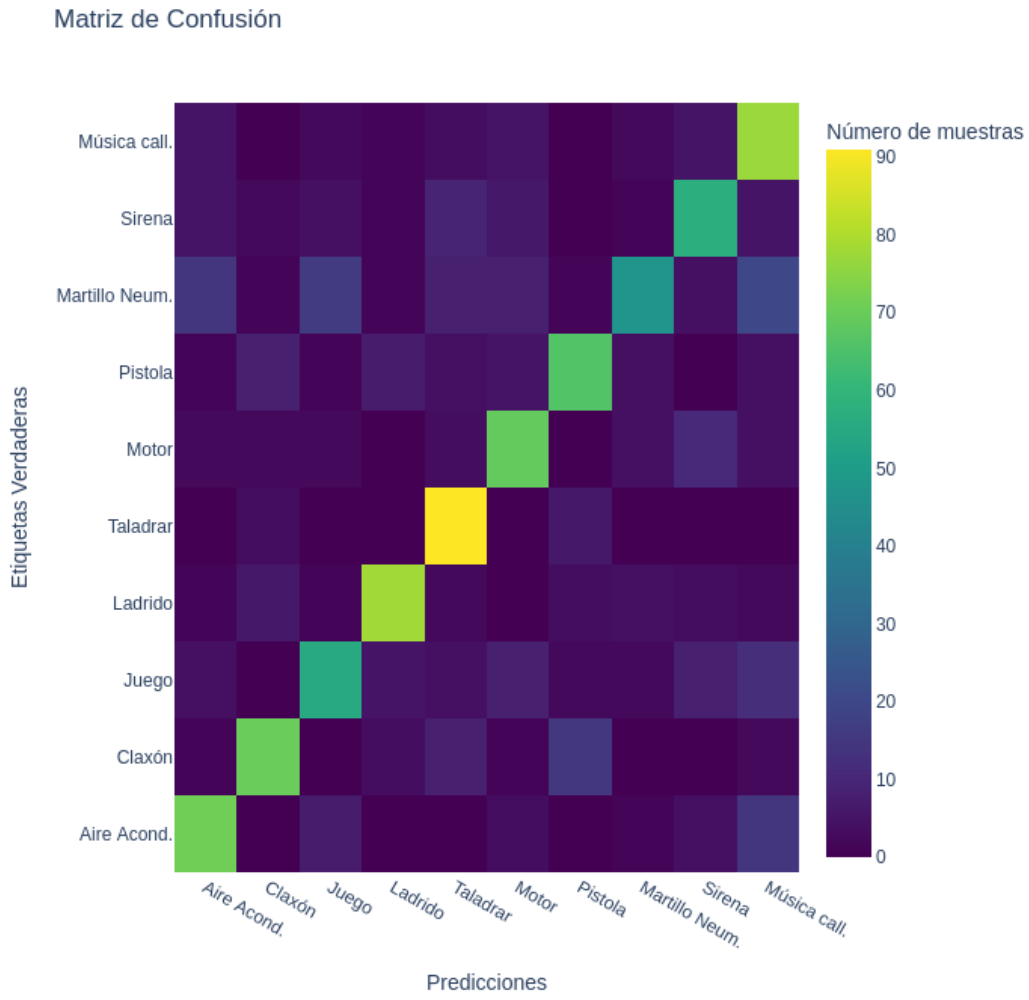

Figura 4.18: Matriz de confusión

Y finalmente se exponen los resultados para la última base de datos, ESC-50. En este caso la forma de la primera gráfica 4.19 es mucho peor que para los datos anteriores. Los valores de AUC en la curva ROC (figura 4.20) son variopintos, y mientras que algunos sonidos como 'Estornudo' alcanza valores altos, otros se mantienen en 0, creando una desigualdad muy grande entre las distintas clases. Esta idea se refuerza con la matriz de confusión, donde se aprecia una nube de puntos sin marcar una tendencia clara. En este caso se ha obtenido una precisión de 56.87 %, mucho inferior a las anteriores. Esto se debe a que, a pesar de haber realizado aumento de datos hasta 20k audios, son pocos audios para representar las 50 clases de las que se compone, por lo que se concluye que no son audios suficientes para caracterizar esta base de datos. Siendo la capacidad de aumento de datos una limitación en este trabajo, que se podría mejorar para trabajos futuros.

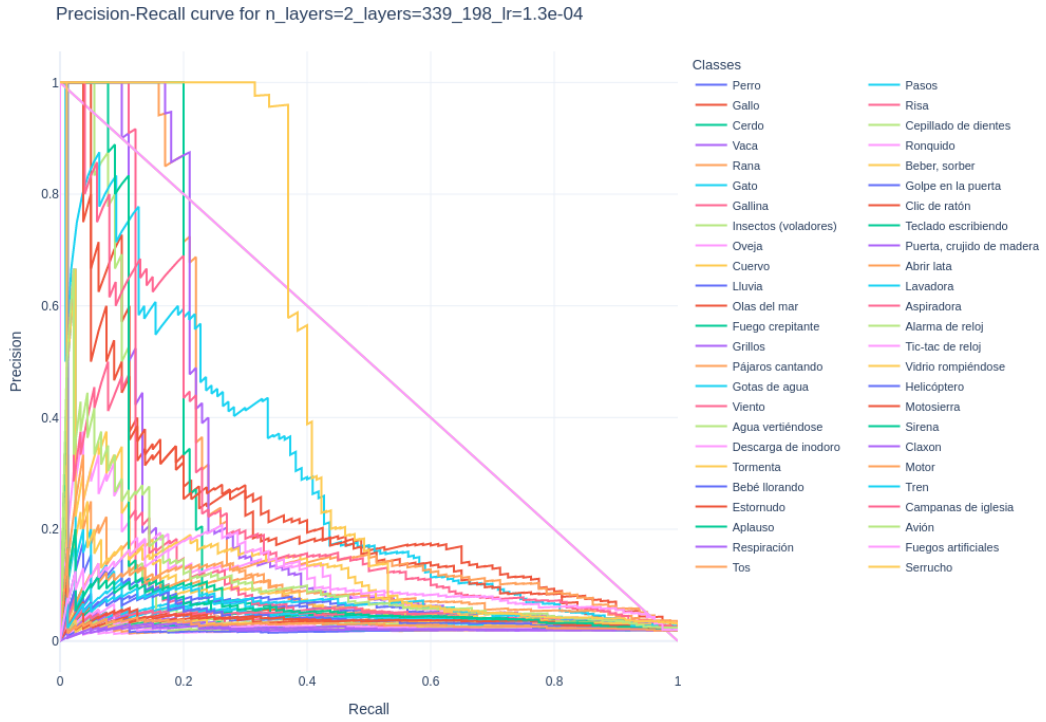

Figura 4.19: Relación 'precision-recall'

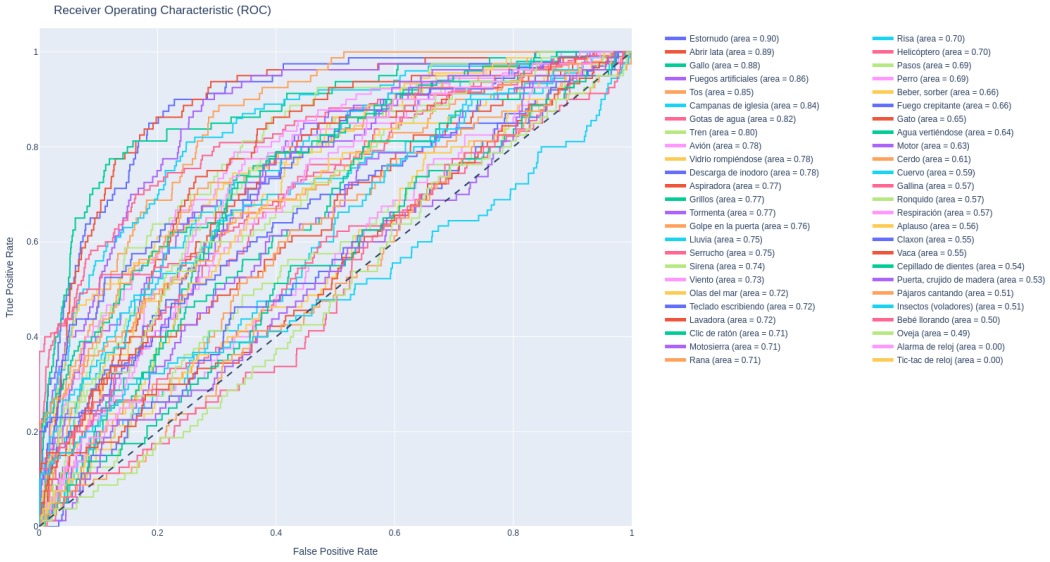

Figura 4.20: Curva ROC

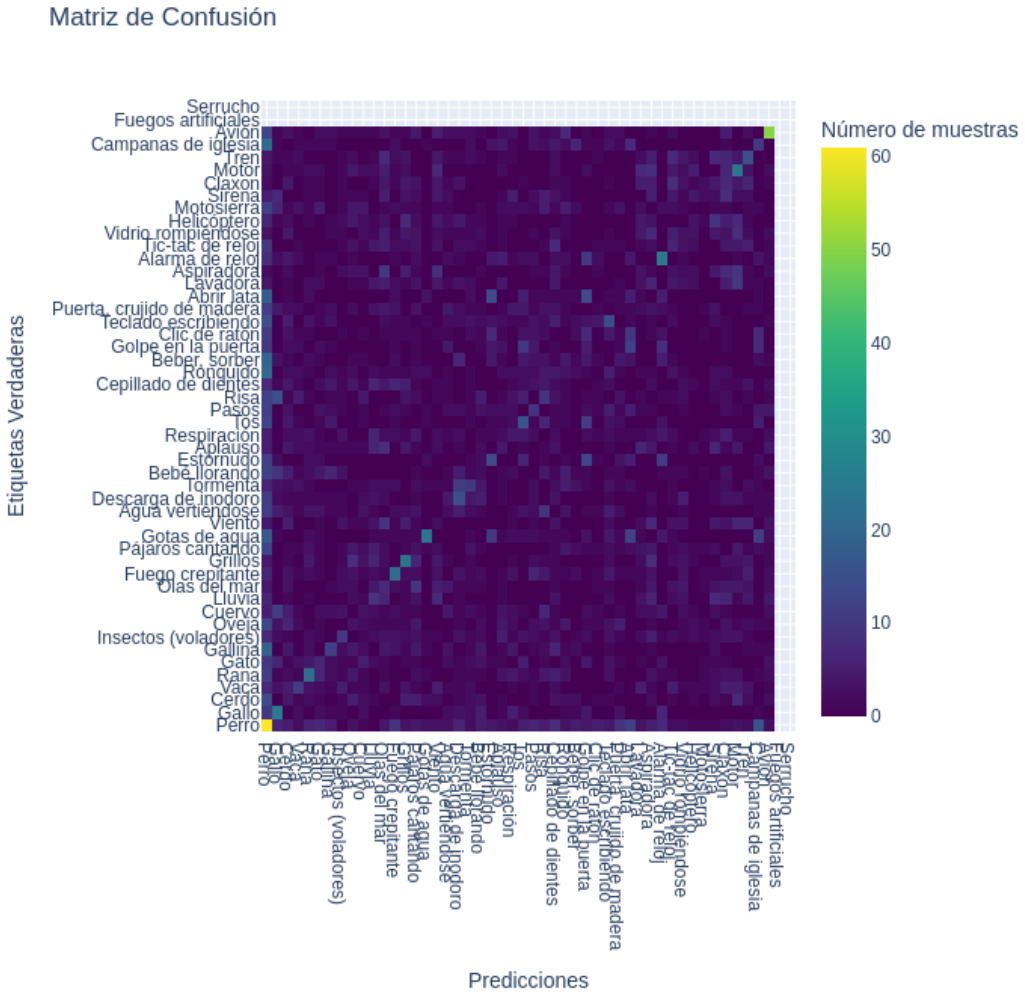

Figura 4.21: Matriz de confusión

La precisión final de cada una de las bases de datos se puede ver en la tabla 4.7, este valor nos permite compararlas entre ellas, denotando claramente un peor resultado para la base de datos *ESC-50*. Además, resulta de interés comparar estos resultados con trabajos previos, lo cual se aborda en la siguiente sección 4.2.1.

	CHime-Home	**US8k**	**ESC-50**
Precisión	81.84 %	84.4 %	56.87 %

Tabla 4.7: Resultados de la clasificación para distintos *dataset*

Comparativa con los resultados de otros trabajos

En este apartado se va a exponer brevemente distintos trabajos, y sus resultados, representados en la tabla 4.8, comparando con las bases de datos *Urban Sound 8k* y *ESC-50*, ya que se utilizan ampliamente.

Wav2CLIP[45]. En este estudio emplea el método CLIP 'Contrastive Language-Image Pre-trained', este modelo proyecta el audio en un espacio compartido con imágenes y texto, de esta forma se pueden empelar menos datos de audio, gracias a la información adquirida de otros tipos de datos.

Transfer learn. for music[46]. Este estudio se basa en aprendizaje por transferencia para tareas de clasificación y regresión musical, a partir de un modelo preentrando se generaliza para todo tipos de audio.

ES-ResNet[47]. Este trabajo propone un modelo basado en espectrogramas generados por STFT (ShortTime Fourier Transform) combinado con varios enfoques propios de los estudios de imágenes, de forma que resulta útil tanto para entradas mono como estéreo.

DenseNet[48]. En este artículo se parte de los modelos 'CNN' de aprendizaje profundo preentrenados por *ImageNet*, a los cuales añade cierta aleatorización para obtener mejores resultados.

Base de datos	**US8k**	**ESC-50**	**Ref.**
Wav2CLIP	81.01 %	85.98 %	[45]
Transfer learn. for music	79 %	-	[46]
ES-ResNet	85.4 %	91.5 %	[47]
DenseNet (Random)	76 %	72.5 %	[48]
DenseNet (Pretrained)	85 %	91 %	[48]
DenseNet (Pretrained Ensemble)	**87 %**	**92.9 %**	[48]
Codec + Class.	84.4 %	56.87 %	-

Tabla 4.8: Comparativa de la precisión de los modelos actuales

Además, se han realizado estudios relacionados con la precisión en caso de clasificación llevada a cabo por humanos [31], donde se dan un resultado de precisión del 95.7 % para '*Urban Sound 8k*' y un 81.3 % para '*ESC-50*'. Donde se destaca que los porcentajes de precisión son muy distintos para las clases, siendo de un 34.1 % para el lavavajillas, y alcanzando el 100 % para el llanto de bebe y los ladridos.

En definitiva, el modelo completo es competente respecto a otros trabajos para el caso de la base de datos *US8k*. Sin embargo, no logra el resultado esperado para *ESC-50*, a pesar de ser una base de datos ampliamente empleada se nombra en varios estudios la problemática de la poca cantidad de datos que aporta, para solucionarlo el primer paso es aplicar aumento de datos, pero solo con ello no es suficiente. En [47] se propone combinarlo con la inicialización de los pesos mediante un método específico basado en el clasificador de *ImageNet* preentrenado. Mientras que en [48] se estudia la inicialización de pesos preentrenados de distintos modelos, y destaca que el impacto de aplicarlo es de entorno a un 20 % para el caso de *ESC-50*, mucho mayor que en otras bases de datos. En el presente trabajo se ha estudiado la inicialización de pesos aleatorios, y por el método Xavier, en ninguno de los casos se veía una mejora respecto a los resultados obtenidos.

Capítulo 5

Conclusiones y trabajo futuro

Se ha desarrollado un nuevo paradigma en el que se utiliza un códec y luego se clasifica a diferencia de los modelos habituales. Este modelo innovador obtiene resultados comparables a otros trabajos actuales, como se refleja en la tabla 4.8. Esto se debe al buen funcionamiento de ambas partes del sistema: por un lado, al codificador que como se ha comprobado da muy buenos resultados y por otro lado al clasificador neuronal.

Respecto a este segundo bloque, se ha decidido emplear un clasificador neuronal totalmente conectado, ya que los resultados obtenidos son buenos, y se han centrado los esfuerzos en modificar sus hiperparámetros, la capa de activación, el optimizador o el planificador, para buscar el mejor resultado con una red neuronal simple. Conociendo estos resultados, se puede modificar el clasificador para emplear modelos más complejos, y ver si al aplicar un ajuste sobre ellos se pueden obtener mejores resultados. Aunque como se ha explicado en el apartado 4.2, en una primera aproximación no se obtienen mejores resultados.

A lo largo del trabajo, una gran limitación encontrada, como ya se preveía en el estado del arte, es la cantidad de datos disponibles para este estudio. A pesar del esfuerzo dedicado al desarrollo de bases de audio específicas para SED, continúa siendo insuficiente. Esto se hace patente con la base de datos ESC-50, que a pesar de estar completa y bien trabajada, necesita de técnicas de aumento de datos para ser empleada, dado que solo proporciona 40 audios para cada clase. Es por ello que se propone un desarrollo de un sistema con capacidad de aumento de datos fiable, donde no se apliquen las técnicas típicas de aumento, sino que se centren los esfuerzos en un estudio y desarrollo de nuevos audios sintéticos que puedan ser similares a los reales y diferenciables entre ellos. De esta forma se debe elevar el número de audios por clase para igualarlos a las otras bases de datos que tienen entorno a 1000 audios por clase, en este caso supone un aumento de 2000 audios a 50000.

Otro punto importante es la diferencia de resultados entre bases de datos,

concretamente para el caso de *ESC-50* donde se ha visto que el uso de pesos preentrenados es de gran importancia para obtener una precisión competitiva. Por ello, se propone un estudio exclusivo de esta base de datos con un pesos preentrenados para lograr equiparar los resultados tanto a otros trabajos, como a las otras bases de datos estudiadas.

Durante este trabajo se ha trabajado con cada base de datos por separado, buscando resultados concretos y desarrollando el modelo en un caso simple. Un clasificador debe ser generalizable, es decir, que para cualquier audio de entrada se le pudiera dar una salida. Aunque este objetivo es utópico, se puede buscar aumentar las clases de forma que sea generalizable dentro de un tipo de audios. Por ello, un trabajo futuro debe ser el diseño de un único clasificador entrenado con las distintas bases de datos, de forma que sea capaz de clasificar los distintos audios con un único modelo.

Capítulo 6

Bibliografía

[1] Sharath Adavanne and Tuomas Virtanen. A report on sound event detection with different binaural features. *CoRR*, abs/1710.02997, 2017.

[2] Agustín Ramírez Agundis. *Diseño y experimentación de un cuantizador vectorial hardware basado en redes neuronales para un sistema de codificación de video.* PhD thesis, Universidad politéctica de Valencia, 2008.

[3] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[4] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec, 2021.

[5] kaiidams. Soundstream for pytorch. https://github.com/kaiidams/soundstream-pytorch?tab=readme-ov-file. Accedido el 9 de marzo de 2024.

[6] lucidrains. Encodec, soundstream. https://github.com/lucidrains/audiolm-pytorch. Accedido el 9 de marzo de 2024.

[7] Jean-Marc Valin, Koen Vos, and Timothy B. Terriberry. Definition of the Opus Audio Codec. RFC 6716, September 2012.

[8] Martin Dietz, Markus Multrus, Vaclav Eksler, Vladimir Malenovsky, Erik Norvell, Harald Pobloth, Lei Miao, Zhe Wang, Lasse Laaksonen, Adriana Vasilache, Yutaka Kamamoto, Kei Kikuiri, Stephane Ragot, Julien Faure, Hiroyuki Ehara, Vivek Rajendran, Venkatraman Atti, Hosang Sung, Eunmi Oh, Hao Yuan, and Changbao Zhu. Overview of the evs codec architecture. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5698–5702, 2015.

[9] Dario Rethage, Jordi Pons, and Xavier Serra. A wavenet for speech denoising, 2018.

[10] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model, 2017.

[11] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville. Melgan: Generative adversarial networks for conditional waveform synthesis, 2019.

[12] lucidrains. Vector quantize, pytoch. https://github.com/lucidrains/vector-quantize-pytorch. Accedido el 9 de marzo de 2024.

[13] Alexey A. Gritsenko, Tim Salimans, Rianne van den Berg, Jasper Snoek, and Nal Kalchbrenner. A spectral energy distance for parallel speech synthesis, 2020.

[14] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

[15] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *arXiv preprint arXiv:2210.13438*, 2022.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[17] Harishchandra Dubey, Vishak Gopal, Ross Cutler, Ashkan Aazami, Sergiy Matusevych, Sebastian Braun, Sefik Emre Eskimez, Manthan Thakker, Takuya Yoshioka, Hannes Gamper, and Robert Aichner. Icassp 2022 deep noise suppression challenge, 2022.

[18] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M. Tyers, and Gregor Weber. Common voice: A massively-multilingual speech corpus, 2020.

[19] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 776–780, 2017.

[20] Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra. Fsd50k: An open dataset of human-labeled sound events. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:829–852, 2022.

[21] Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The mtg-jamendo dataset for automatic music tagging. In *International Conference on Machine Learning*, 2019.

[22] Recommendation itu-r bs.1534-1 - method for the subjective assessment of intermediate quality level of coding systems. 2003.

[23] Andrew Hines, Jan Skoglund, Anil Kokaram, and Naomi Harte. Visqol: an objective speech quality model. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015 (13):1–18, 2015.

[24] Yi Luo and Nima Mesgarani. Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(8):1256–1266, 2019.

[25] Eduardo Fonseca, Manoj Plakal, Daniel P. W. Ellis, Frederic Font, Xavier Favory, and Xavier Serra. Learning sound event classifiers from web audio with noisy labels, 2019.

[26] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, March 2017.

[27] Justin Salamon, Duncan MacConnell, Mark Cartwright, Peter Li, and Juan Pablo Bello. Scaper: A library for soundscape synthesis and augmentation. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, October 2017.

[28] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. Tut database for acoustic scene classification and sound event detection. In *2016 24th European Signal Processing Conference (EUSIPCO)*. IEEE, August 2016.

[29] Peter Foster, Siddharth Sigtia, Sacha Krstulovic, Jon Barker, and Mark D. Plumbley. Chime-home: A dataset for sound source recognition in a domestic environment. In *2015 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, October 2015.

[30] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, pages 1041–1044, New York, NY, USA, 2014. ACM.

[31] Karol J. Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 1015–1018, New York, NY, USA, 2015. Association for Computing Machinery.

[32] Nicolas Turpault, Romain Serizel, Ankit Parag Shah, and Justin Salamon. Sound event detection in domestic environments with weakly labeled data and soundscape synthesis. In *Workshop on Detection and Classification of Acoustic Scenes and Events*, New York City, United States, October 2019.

[33] Selina Chu, Shrikanth Narayanan, and C.-C. Jay Kuo. Environmental sound recognition with time–frequency audio features. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(6):1142–1158, August 2009.

[34] Chime home dataset. https://archive.org/details/chime-home. Accedido el 9 de marzo de 2024.

[35] Karol J. Piczak. Esc-50: Dataset for environmental sound classification. https://github.com/karolpiczak/ESC-50, 2015.

[36] Lucas Ferreira-Paiva, Elizabeth Alfaro-Espinoza, Vinicius M. Almeida, Leonardo B. Felix, and Rodolpho V. A. Neves. A survey of data augmentation for audio classification. In *Procedings do XXII Congresso Brasileiro de Automática*, CBA2022. SBA Sociedade Brasileira de Automática, October 2022.

[37] Sacha Krstulović. *Audio Event Recognition in the Smart Home*, page 335–371. Springer International Publishing, September 2017.

[38] Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. Audiolm: a language modeling approach to audio generation, 2023.

[39] Maxime Baelde, Christophe Biernacki, and Raphaël Greff. Real-time monophonic and polyphonic audio classification from power spectra. *Pattern Recognition*, 92:82–92, 2019.

[40] google. Lyra, soundstream. https://github.com/google/lyra. Accedido el 9 de marzo de 2024.

[41] Yu-An Chung, Yu Zhang, Wei Han, Chung-Cheng Chiu, James Qin, Ruoming Pang, and Yonghui Wu. W2v-bert: Combining contrastive learning and masked language modeling for self-supervised speech pre-training, 2021.

[42] PyTorch Lightning Team. Trainer - pytorch lightning, 2024. Accedido: 2024-08-04.

[43] Ivan Dubovy and Contributors. Audiomentations: A library for audio data augmentation in machine learning experiments. https://pypi.org/project/audiomentations/0.20.0/, 2023. Version 0.20.0.

[44] Ray. Ray tune: Scalable hyperparameter tuning. https://docs.ray.io/en/latest/tune/index.html, 2024. Accessed: 2024-10-16.

[45] Ho-Hsiang Wu, Prem Seetharaman, Kundan Kumar, and Juan Pablo Bello. Wav2clip: Learning robust audio representations from clip, 2022.

[46] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Transfer learning for music classification and regression tasks, 2017.

[47] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. Esresnet: Environmental sound classification based on visual domain models, 2020.

[48] Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. Rethinking cnn models for audio classification, 2020.

[49] H. Farnsworth. What-if machine analysis and design. *IEEE Transactions on quantum neuroscience electronics*, 3031.

[50] Justin Salamon and Juan Pablo Bello. Feature learning with deep scattering for urban sound analysis. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, New York, USA, 2015. Center for Urban Science and Progress, New York University and Music and Audio Research Laboratory, New York University.

[51] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. In *IEEE Signal Processing Letters*, New York, USA, 2016. Center for Urban Science and Progress, New York University and Music and Audio Research Laboratory, New York University.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Resultados distintos modelos clasificador neuronal

En este anexo se presentan detalladamente las pruebas y experimentos realizados durante el proceso de desarrollo y entrenamiento del clasificador neuronal. Cada prueba se diseñó para ajustar un conjunto de parámetros concreto.

A.1. Pruebas para la base de datos *CHime-Home*

La tabla A.1 muestra las primeras pruebas, que se emplearon para acotar los hiperparámetros de la red.

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**C. 5**	**C. 6**
74.48 %	$7{,}996 \times 10^{-3}$	3	85	44	36	-	-	-
75.21 %	$4{,}63 \times 10^{-3}$	3	193	75	69	-	-	-
74.51 %	$1{,}92 \times 10^{-3}$	3	229	207	95	-	-	-
74.02 %	$5{,}86 \times 10^{-3}$	1	167	158	119	-	-	-
73.52 %	$6{,}29 \times 10^{-3}$	1	146	-	-	-	-	-
73.82 %	$4{,}57 \times 10^{-3}$	3	144	100	100	-	-	-
74.05 %	$4{,}81 \times 10^{-3}$	3	243	231	170	-	-	-
73.96 %	$7{,}38 \times 10^{-3}$	1	230	-	-	-	-	-
73.85 %	$3{,}28 \times 10^{-3}$	3	150	60	48	-	-	-
74.12 %	$3{,}33 \times 10^{-3}$	2	54	39	-	-	-	-
75.15 %	$4{,}597 \times 10^{-4}$	3	204	192	69	-	-	-
77.33 %	$6{,}23 \times 10^{-4}$	3	206	122	88	-	-	-
77.00 %	$1{,}86 \times 10^{-4}$	2	54	39	-	-	-	-
77.56 %	$4{,}43 \times 10^{-4}$	3	164	102	33	-	-	-
75.11 %	$4{,}15 \times 10^{-4}$	3	205	113	70	-	-	-
76.89 %	$9{,}71 \times 10^{-4}$	3	137	128	66	-	-	-
77.56 %	$3{,}34 \times 10^{-4}$	1	166	-	-	-	-	-

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**C. 5**	**C. 6**
72.14 %	$2{,}715 \times 10^{-4}$	6	221	159	150	127	91	60
73.86 %	$1{,}58 \times 10^{-4}$	6	229	209	199	187	177	123
74.48 %	$5{,}69 \times 10^{-4}$	5	207	119	86	76	40	-
78.60 %	$2{,}27 \times 10^{-5}$	1	154	-	-	-	-	-
78.07 %	$6{,}919 \times 10^{-5}$	5	180	119	118	97	59	-
74.48 %	$1{,}62 \times 10^{-5}$	4	255	166	128	83	-	-
77.94 %	$6{,}91 \times 10^{-5}$	6	210	192	170	130	90	72
77.13 %	$4{,}63 \times 10^{-5}$	6	234	198	158	130	36	34
74.48 %	$1{,}65 \times 10^{-5}$	4	249	195	162	93	-	-
78.69 %	$1{,}15 \times 10^{-5}$	1	229	-	-	-	-	-
79.05 %	$1{,}04 \times 10^{-5}$	2	184	73	-	-	-	-
78.31 %	$2{,}91 \times 10^{-5}$	3	137	127	116	-	-	-
74.48 %	$1{,}47 \times 10^{-5}$	5	223	192	145	130	112	-
75.27 %	$2{,}28 \times 10^{-6}$	1	126	-	-	-	-	-
76.24 %	$2{,}98 \times 10^{-6}$	1	123	-	-	-	-	-
75.92 %	$2{,}82 \times 10^{-6}$	1	60	-	-	-	-	-
74.48 %	$4{,}34 \times 10^{-6}$	4	233	198	65	49	-	-
74.68 %	$1{,}59 \times 10^{-6}$	1	167	-	-	-	-	-
74.48 %	$1{,}35 \times 10^{-6}$	6	183	179	174	109	107	95
74.48 %	$1{,}53 \times 10^{-6}$	6	250	236	228	210	147	111
74.48 %	$1{,}33 \times 10^{-6}$	5	152	136	125	107	71	-
74.48 %	$7{,}22 \times 10^{-6}$	5	248	245	206	145	72	-
74.48 %	$2{,}70 \times 10^{-6}$	3	182	115	80	-	-	-

Tabla A.1: Comparativa modelos, CHime-Home

En la tabla A.2 se muestran las pruebas realizadas para acotar el número de épocas y el umbral a partir del cual se decide que está presente la clase. Además se implementó y ajustó el sistema de '*early stopping*' durante las mismas.

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
80.97 %	$1{,}19 \times 10^{-4}$	1	115	-	-	-	0.478
80.91 %	$1{,}48 \times 10^{-4}$	2	81	78	-	-	0.477
80.82 %	$1{,}19 \times 10^{-4}$	1	92	-	-	-	0.460
80.29 %	$9{,}99 \times 10^{-4}$	2	140	115	-	-	0.517
81.08 %	$1{,}07 \times 10^{-4}$	2	158	57	-	-	0.542
80.99 %	$4{,}5 \times 10^{-4}$	1	81	-	-	-	0.484
81.04 %	$2{,}62 \times 10^{-4}$	2	139	118	-	-	0.546
80.92 %	$6{,}36 \times 10^{-4}$	1	54	-	-	-	0.544
80.88 %	$3{,}88 \times 10^{-4}$	1	143	-	-	-	0.466
80.77 %	$1{,}55 \times 10^{-4}$	2	192	184	-	-	0.453

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
80.29 %	$2{,}06 \times 10^{-3}$	1	80	-	-	-	0.458
80.29 %	$2{,}16 \times 10^{-3}$	3	252	118	74	-	0.499
79.37 %	$2{,}46 \times 10^{-3}$	2	218	217	-	-	0.536
79.35 %	$1{,}90 \times 10^{-3}$	1	190	184	-	-	0.547
78.24 %	$4{,}09 \times 10^{-3}$	3	251	160	141	-	0.541
80.29 %	$1{,}36 \times 10^{-3}$	1	164	-	-	-	0.493
80.29 %	$2{,}22 \times 10^{-3}$	1	109	-	-	-	0.476
80.29 %	$1{,}91 \times 10^{-3}$	3	181	114	111	-	0.525
79.37 %	$8{,}00 \times 10^{-3}$	2	148	113	-	-	0.540
79.37 %	$3{,}56 \times 10^{-3}$	2	76	48	-	-	0.547
80.69 %	$2{,}65 \times 10^{-4}$	2	168	123	-	-	0.5
82.60 %	$9{,}14 \times 10^{-5}$	3	137	66	39	-	0.5
80.28 %	$5{,}02 \times 10^{-5}$	3	248	53	49	-	0.5
81.20 %	$6{,}79 \times 10^{-5}$	3	106	31	26	-	0.5
82.75 %	$6{,}22 \times 10^{-5}$	2	169	107	-	-	0.5
82.72 %	$6{,}52 \times 10^{-5}$	2	164	97	-	-	0.5
82.44 %	$9{,}16 \times 10^{-5}$	3	101	68	47	-	0.5
83.01 %	$8{,}04 \times 10^{-5}$	3	225	47	33	-	0.5
83.10 %	$5{,}28 \times 10^{-5}$	2	98	63	-	-	0.5
81.79 %	$9{,}77 \times 10^{-5}$	3	112	78	24	-	0.5
82.39 %	$8{,}77 \times 10^{-5}$	2	113	47	-	-	0.5
80.92 %	$2{,}16 \times 10^{-4}$	2	234	37	-	-	0.5
81.51 %	$1{,}26 \times 10^{-4}$	3	118	70	37	-	0.5
79.72 %	$1{,}71 \times 10^{-4}$	3	227	77	31	-	0.5
80.95 %	$2{,}07 \times 10^{-4}$	2	116	49	-	-	0.5
79.70 %	$1{,}69 \times 10^{-4}$	3	150	49	25	-	0.5
81.28 %	$1{,}07 \times 10^{-4}$	2	236	90	-	-	0.5
80.08 %	$2{,}60 \times 10^{-4}$	3	114	78	55	-	0.5
81.17 %	$1{,}07 \times 10^{-4}$	2	100	34	-	-	0.5
80.38 %	$3{,}42 \times 10^{-4}$	3	213	83	56	-	0.5
80.60 %	$1{,}82 \times 10^{-4}$	2	153	33	-	-	0.5
80.29 %	$4{,}61 \times 10^{-3}$	4	134	87	48	28	0.5
80.29 %	$3{,}15 \times 10^{-3}$	3	57	36	24	-	0.5
80.29 %	$1{,}77 \times 10^{-3}$	2	118	31	-	-	0.5
80.29 %	$3{,}14 \times 10^{-3}$	2	242	32	-	-	0.5
80.29 %	$1{,}88 \times 10^{-3}$	2	223	110	-	-	0.5
81.94 %	$1{,}93 \times 10^{-3}$	2	241	126	-	-	0.5
80.29 %	$2{,}05 \times 10^{-3}$	2	135	20	-	-	0.5
80.29 %	$2{,}50 \times 10^{-3}$	4	124	123	35	27	0.5
80.29 %	$4{,}69 \times 10^{-3}$	3	233	62	25	-	0.5
80.29 %	$4{,}86 \times 10^{-3}$	3	121	46	45	-	0.5
80.29 %	$7{,}12 \times 10^{-3}$	2	206	90	-	-	0.5
80.29 %	$6{,}00 \times 10^{-3}$	4	174	86	52	18	0.5
79.16 %	$7{,}75 \times 10^{-3}$	2	247	85	-	-	0.5
80.29 %	$6{,}39 \times 10^{-3}$	2	237	26	-	-	0.5
80.29 %	$6{,}25 \times 10^{-3}$	2	118	94	-	-	0.5

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
80.29 %	$7{,}61 \times 10^{-3}$	2	71	120	-	-	0.5
80.29 %	$7{,}69 \times 10^{-3}$	4	229	116	57	23	0.5
79.37 %	$6{,}67 \times 10^{-3}$	3	129	123	30	36	0.5
80.29 %	$8{,}21 \times 10^{-3}$	4	147	105	53	20	0.5
80.29 %	$6{,}89 \times 10^{-3}$	3	42	105	49	59	0.5
80.29 %	$6{,}34 \times 10^{-3}$	4	114	110	53	18	0.5
80.29 %	$7{,}17 \times 10^{-3}$	3	81	73	50	36	0.5
80.29 %	$8{,}72 \times 10^{-3}$	3	159	77	61	27	0.5
80.29 %	$5{,}33 \times 10^{-3}$	2	217	56	-	-	0.5
80.29 %	$7{,}34 \times 10^{-3}$	2	188	37	-	-	0.5
80.29 %	$6{,}58 \times 10^{-3}$	2	100	85	-	-	0.5
80.29 %	$5{,}04 \times 10^{-3}$	3	186	57	42	55	0.5
80.29 %	$9{,}38 \times 10^{-3}$	4	36	28	48	40	0.5
80.29 %	$5{,}61 \times 10^{-3}$	4	148	38	47	16	0.5
80.29 %	$5{,}16 \times 10^{-3}$	2	60	66	-	-	0.5
80.29 %	$8{,}01 \times 10^{-3}$	3	138	80	48	35	0.5
80.29 %	$5{,}40 \times 10^{-3}$	4	197	89	60	61	0.5
80.29 %	$7{,}96 \times 10^{-3}$	4	248	115	58	37	0.5
80.29 %	$5{,}04 \times 10^{-3}$	2	93	34	-	-	0.5
80.29 %	$7{,}96 \times 10^{-3}$	2	141	97	-	-	0.5
80.29 %	$8{,}87 \times 10^{-3}$	4	172	62	31	55	0.5
80.29 %	$5{,}50 \times 10^{-3}$	3	101	91	32	50	0.5
80.29 %	$5{,}66 \times 10^{-3}$	2	113	95	-	-	0.5
80.29 %	$5{,}14 \times 10^{-3}$	2	200	37	-	-	0.5
80.29 %	$8{,}94 \times 10^{-3}$	3	176	85	43	30	0.5
80.45 %	$2{,}41 \times 10^{-4}$	4	223	75	16	43	0.5
80.93 %	$2{,}30 \times 10^{-4}$	2	206	59	-	-	0.5
78.29 %	$3{,}49 \times 10^{-4}$	3	118	51	53	23	0.5
80.35 %	$1{,}86 \times 10^{-4}$	3	59	87	49	45	0.5
81.03 %	$3{,}98 \times 10^{-4}$	2	72	59	-	-	0.5
81.44 %	$1{,}06 \times 10^{-4}$	2	208	109	-	-	0.5
80.68 %	$2{,}10 \times 10^{-4}$	4	214	111	36	52	0.5
80.36 %	$2{,}40 \times 10^{-4}$	4	35	60	25	44	0.5
81.84 %	$\mathbf{1{,}17 \times 10^{-4}}$	**4**	**221**	**42**	**50**	**26**	**0.5**
81.48 %	$1{,}41 \times 10^{-4}$	2	104	65	-	-	0.5

Tabla A.2: Comparativa modelos, cambio n.épocas y umbrales, CHime-Home

A.2. Pruebas para la base de datos *Urban Sound 8k*

Acotados los valores para los distintos parámetros, en este apartado se muestran las pruebas realizadas para obtener el mejor modelo para la base de datos *Urban Sound 8k*.

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
80.00 %	$1{,}25 \times 10^{-4}$	4	232	99	61	49	0.5
80.00 %	$1{,}95 \times 10^{-4}$	2	107	50	-	-	0.5
80.00 %	$1{,}84 \times 10^{-4}$	3	80	120	60	18	0.5
80.00 %	$3{,}46 \times 10^{-4}$	2	250	80	-	-	0.5
80.00 %	$1{,}84 \times 10^{-4}$	3	44	123	55	58	0.5
80.00 %	$1{,}87 \times 10^{-4}$	2	78	123	-	-	0.5
80.00 %	$2{,}84 \times 10^{-4}$	2	32	80	-	-	0.5
80.00 %	$2{,}40 \times 10^{-4}$	3	93	118	41	22	0.5
80.00 %	$2{,}41 \times 10^{-4}$	4	222	60	30	39	0.5
80.00 %	$3{,}32 \times 10^{-4}$	2	188	43	-	-	0.5
72.69 %	$2{,}56 \times 10^{-4}$	2	84	46	-	-	0.5
74.01 %	$3{,}44 \times 10^{-4}$	3	238	118	57	63	0.5
73.30 %	$1{,}56 \times 10^{-4}$	4	188	74	49	20	0.5
73.21 %	$1{,}07 \times 10^{-4}$	3	166	26	22	27	0.5
74.85 %	$2{,}98 \times 10^{-4}$	2	103	100	-	-	0.5
73.34 %	$1{,}40 \times 10^{-4}$	3	54	110	31	25	0.5
74.85 %	$3{,}98 \times 10^{-4}$	2	222	31	-	-	0.5
74.14 %	$2{,}25 \times 10^{-4}$	4	99	60	53	24	0.5
74.05 %	$2{,}76 \times 10^{-4}$	3	79	40	21	23	0.5
74.01 %	$1{,}40 \times 10^{-4}$	2	163	60	-	-	0.5
74.05 %	$1{,}66 \times 10^{-4}$	4	233	78	19	34	0.5
80.54 %	$4{,}12 \times 10^{-4}$	3	125	43	24	60	0.5
79.91 %	$4{,}64 \times 10^{-4}$	3	208	17	55	20	0.5
80.09 %	$1{,}74 \times 10^{-4}$	4	207	64	47	57	0.5
80.91 %	$1{,}69 \times 10^{-4}$	2	145	124	-	-	0.5
80.01 %	$3{,}09 \times 10^{-4}$	4	142	126	48	56	0.5
79.99 %	$2{,}31 \times 10^{-4}$	4	91	29	47	21	0.5
79.87 %	$1{,}56 \times 10^{-4}$	3	135	115	50	24	0.5
80.88 %	$3{,}26 \times 10^{-4}$	2	51	84	-	-	0.5
79.86 %	$4{,}37 \times 10^{-4}$	3	229	73	38	60	0.5
79.97 %	$1{,}33 \times 10^{-4}$	4	208	60	23	46	0.5
80.51 %	$3{,}81 \times 10^{-4}$	3	135	124	53	26	0.5
80.05 %	$1{,}93 \times 10^{-4}$	3	105	37	37	53	0.5
80.50 %	$3{,}28 \times 10^{-4}$	2	200	57	-	-	0.5
79.94 %	$1{,}75 \times 10^{-4}$	3	162	18	18	46	0.5

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
79.90 %	$2{,}84 \times 10^{-4}$	4	157	36	49	35	0.5
80.31 %	$3{,}28 \times 10^{-4}$	3	40	103	58	62	0.5
80.65 %	$4{,}63 \times 10^{-4}$	2	73	81	-	-	0.5
80.15 %	$4{,}29 \times 10^{-4}$	4	244	86	29	43	0.5
80.40 %	$1{,}82 \times 10^{-4}$	3	33	73	59	20	0.5
80.02 %	$1{,}36 \times 10^{-4}$	4	174	61	35	51	0.5
80.43 %	$4{,}45 \times 10^{-5}$	2	91	31	-	-	0.5
80.32 %	$1{,}47 \times 10^{-5}$	2	206	103	-	-	0.5
80.12 %	$3{,}60 \times 10^{-4}$	2	219	94	-	-	0.5
80.60 %	$8{,}49 \times 10^{-5}$	2	245	89	-	-	0.5
79.10 %	$2{,}00 \times 10^{-3}$	2	96	81	-	-	0.5
79.78 %	$2{,}06 \times 10^{-3}$	2	185	66	-	-	0.5
79.34 %	$1{,}19 \times 10^{-3}$	2	70	124	-	-	0.5
80.81 %	$2{,}56 \times 10^{-5}$	2	33	116	-	-	0.5
80.48 %	$9{,}48 \times 10^{-5}$	2	62	118	-	-	0.5
79.56 %	$1{,}11 \times 10^{-3}$	2	42	55	-	-	0.5
79.86 %	$3{,}16 \times 10^{-6}$	3	191	115	53	21	0.5
84.19 %	$2{,}25 \times 10^{-4}$	3	218	75	44	26	0.5
76.43 %	$1{,}78 \times 10^{-6}$	3	158	80	48	27	0.5
79.88 %	$1{,}01 \times 10^{-5}$	2	214	73	-	-	0.5
84.40 %	$\mathbf{3{,}82 \times 10^{-4}}$	**2**	**211**	**103**	-	-	**0.5**

Tabla A.3: Comparativa modelos, Urban Sound 8k

Además, se pueden ver los resultados de las pruebas cruzadas, cabe destacar que estas pruebas no se han realizado para el mejor modelo.

Carp. Entr.	Carp. Valid.	Carp. Test	Precisión
1 - 8	9	10	82.14 %
2 - 9	10	1	82.30 %
3 - 10	1	2	81.06 %
4 - 10, 1	2	3	80.49 %
5 - 10, 1 - 2	3	4	81.64 %
6 - 10, 1 - 3	4	5	83.18 %
7 - 10, 1 - 4	5	6	82.64 %
8 - 10, 1 - 5	6	7	81.60 %
9 - 10, 1 - 6	7	8	82.14 %
1 - 7, 10	8	9	82.14 %

Tabla A.4: Resultados pruebas cruzadas, Urban Soun 8k

A.3. Pruebas para la base de datos *ESC-50*

Esta sección muestra las combinaciones y el resultado de las pruebas realizadas con la base de datos *ESC-50*.

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
56.35 %	$1{,}26 \times 10^{-3}$	2	99	28	-	-	0.5
56.41 %	$1{,}54 \times 10^{-3}$	2	101	48	-	-	0.5
56.24 %	$1{,}08 \times 10^{-5}$	2	72	89	-	-	0.5
56.29 %	$1{,}54 \times 10^{-5}$	2	97	53	-	-	0.5
56.26 %	$9{,}74 \times 10^{-5}$	2	38	87	-	-	0.5
56.38 %	$7{,}72 \times 10^{-3}$	2	226	71	-	-	0.5
56.30 %	$7{,}40 \times 10^{-3}$	2	135	26	-	-	0.5
56.37 %	$1{,}43 \times 10^{-4}$	2	198	96	-	-	0.5
56.33 %	$5{,}20 \times 10^{-5}$	2	157	44	-	-	0.5
56.36 %	$1{,}32 \times 10^{-4}$	2	227	87	-	-	0.5
51.79 %	$7{,}04 \times 10^{-4}$	2	197	49	-	-	0.5
55.03 %	$9{,}01 \times 10^{-3}$	2	107	25	-	-	0.5
56.61 %	$9{,}60 \times 10^{-5}$	2	164	127	-	-	0.5
56.29 %	$1{,}39 \times 10^{-5}$	2	77	69	-	-	0.5
56.32 %	$1{,}31 \times 10^{-5}$	2	145	56	-	-	0.5
56.33 %	$2{,}54 \times 10^{-5}$	2	105	35	-	-	0.5
54.15 %	$4{,}62 \times 10^{-3}$	2	115	109	-	-	0.5
53.72 %	$9{,}51 \times 10^{-4}$	2	197	36	-	-	0.5
56.49 %	$9{,}24 \times 10^{-5}$	2	92	89	-	-	0.5
45.25 %	$2{,}07 \times 10^{-3}$	2	104	17	-	-	0.5
45.50 %	$3{,}15 \times 10^{-5}$	2	172	101	-	-	0.5
61.60 %	$1{,}27 \times 10^{-3}$	2	45	29	-	-	0.5
49.85 %	$1{,}05 \times 10^{-5}$	2	195	96	-	-	0.5
50.60 %	$4{,}80 \times 10^{-4}$	2	139	67	-	-	0.5
51.35 %	$3{,}54 \times 10^{-5}$	2	185	112	-	-	0.5
52.55 %	$4{,}93 \times 10^{-4}$	2	49	117	-	-	0.5
50.85 %	$3{,}90 \times 10^{-5}$	2	226	25	-	-	0.5
49.10 %	$3{,}50 \times 10^{-5}$	2	88	109	-	-	0.5
39.75 %	$2{,}49 \times 10^{-4}$	2	135	82	-	-	0.5
50.80 %	$2{,}07 \times 10^{-5}$	2	140	34	-	-	0.5
52.95 %	$3{,}59 \times 10^{-4}$	2	209	91	-	-	0.5
50.40 %	$1{,}48 \times 10^{-3}$	2	41	125	-	-	0.5
53.85 %	$2{,}13 \times 10^{-5}$	2	182	118	-	-	0.5
53.85 %	$1{,}58 \times 10^{-5}$	2	151	28	-	-	0.5
51.55 %	$8{,}16 \times 10^{-4}$	2	186	109	-	-	0.5
47.20 %	$3{,}64 \times 10^{-5}$	2	152	32	-	-	0.5
49.50 %	$1{,}36 \times 10^{-5}$	2	251	42	-	-	0.5
46.40 %	$6{,}12 \times 10^{-5}$	2	80	121	-	-	0.5
52.55 %	$4{,}48 \times 10^{-4}$	2	95	96	-	-	0.5

Precisión	**Learn. Rate**	**Num. Capas**	**C. 1**	**C. 2**	**C. 3**	**C. 4**	**Umbral**
50.80 %	$2{,}43 \times 10^{-4}$	2	295	233	-	-	0.5
49.98 %	$1{,}41 \times 10^{-5}$	2	413	217	-	-	0.5
51.15 %	$4{,}97 \times 10^{-4}$	2	352	192	-	-	0.5
49.65 %	$3{,}54 \times 10^{-3}$	2	409	197	-	-	0.5
51.00 %	$1{,}08 \times 10^{-4}$	2	377	186	-	-	0.5
49.55 %	$5{,}28 \times 10^{-3}$	2	352	165	-	-	0.5
49.85 %	$9{,}59 \times 10^{-4}$	2	378	151	-	-	0.5
49.92 %	$3{,}14 \times 10^{-3}$	2	417	137	-	-	0.5
52.33 %	$1{,}61 \times 10^{-5}$	2	333	206	-	-	0.5
52.10 %	$4{,}59 \times 10^{-3}$	2	460	203	-	-	0.5
49.40 %	$3{,}05 \times 10^{-5}$	2	256	153	-	-	0.5
51.46 %	$1{,}04 \times 10^{-3}$	2	262	153	-	-	0.5
51.87 %	$1{,}14 \times 10^{-5}$	2	270	248	-	-	0.5
49.96 %	$9{,}38 \times 10^{-4}$	3	418	203	99	-	0.5
52.23 %	$4{,}26 \times 10^{-3}$	3	343	146	85	-	0.5
51.93 %	$6{,}92 \times 10^{-4}$	3	489	147	126	-	0.5
56.87 %	$\mathbf{1{,}30 \times 10^{-4}}$	**2**	**339**	**198**	-	-	**0.5**

Tabla A.5: Comparativa modelos, ESC50

Además, se pueden ver los resultados de las pruebas cruzadas, cabe destacar que estas pruebas no se han realizado para el mejor modelo.

Carp. Entr.	Carp. Valid.	Carp. Test	Precisión
1 - 3	4	5	52.32 %
2 - 4	5	1	54.81 %
3 - 5	1	2	53.27 %
4 - 5, 1	2	3	53.41 %
5, 1 - 2	3	4	53.76 %

Tabla A.6: Resultados pruebas cruzadas, ESC50

Anexos B

Hardware empleado para la realización del trabajo

El trabajo se ha desarrollado en un equipo propiedad de la Universidad de Zaragoza, con las siguientes características.

Componente	**Especificaciones**
Sistema Operativo	CentOS Linux 7
CPU	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Memoria RAM	64 GB
Tarjeta Gráfica	NVIDIA GeForce RTX 3090
Disco duro 1: SSD Kingston	KINGSTON SHSS37A 240GB
Disco duro 2: HDD Seagate	ST1000DM003-1SB1. Seagate Barracuda, 1 TB
Disco duro 3: SSD Crucial	CT2000MX500SSD1. Crucial MX500 2TB

Tabla B.1: Anotaciones sobre cada 'chunk'

Además se han empleado distintos recursos Software nombrados a lo largo de la memoria en su apartado específico.