UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Estudio comparativo de diversas redes convoluciones de baja complejidad para la clasificación de escenas sonoras

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Zaragoza Paredes, Josep Manel

Tutor/a: Naranjo Ornedo, Valeriana

Cotutor/a externo: Naranjo Alcázar, Javier

CURSO ACADÉMICO: 2024/2025

Resumen

El presente TFG se basa en el desarrollo y evaluación de modelos convolucionales para la clasificación entre escenas acústicas en tiempo real. La audición por computador es una de las ramas de inteligencia artificial que se basa en emular la capacidad humana para interpretar y entender el sonido. La extracción de la información del audio es importante, ya que contiene características importantes para ser aprovechadas en diferentes funciones como la detección de anomalías en máquinas o, como en el presente TFG, clasificar entre diferentes escenas acústicas.

Por otra parte, la implementación de modelos en tiempo real es muy relevante para el uso en dispositivos como smartphones, sistemas embedidos (*embedding systems* en inglés) o microcomputadores. Gracias a su bajo coste y portabilidad son excelentes para aplicaciones prácticas. Por lo tanto, es importante implementar modelos eficientes tanto a coste computacional como en precisión para que no suponga un problema en el rendimiento del dispositivo. Así pues, definitivamente ser capaz de procesar y clasificar entre diferentes escenas acústicas en tiempo real abre oportunidades para aplicaciones móviles, mejorando la adaptabilidad de los sistemas basados en audio.

Resum

El present TFG es basa en el desenvolupament i avaluació de models convolucionals per a la classificació entre escenes acústiques en temps real. L'audició per computador és una de les branques d'intel·ligència artificial que es basa a emular la capacitat humana per a interpretar i entendre el so. L'extracció de la informació de l'àudio és important, ja que conté característiques importants per a ser aprofitades en diferents funcions com la detecció d'anomalies en màquines o, com en el present TFG, classificar entre diferents escenes acústiques.

D'altra banda, la implementació de models en temps real és molt rellevant per a l'ús en dispositius

com a telèfons intel·ligents, sistemes embedits (*embedding systems* en anglés) o microcomputadors. Gràcies al seu baix cost i portabilitat són excel·lents per a aplicacions pràctiques. Per tant, és important implementar models eficients tant a cost computacional com en precisió perquè no supose un problema en el rendiment del dispositiu. Així doncs, definitivament ser capaç de processar i classificar entre diferents escenes acústiques en temps real obri oportunitats per a aplicacions mòbils, millorant l'adaptabilitat dels sistemes basats en àudio

Abstract

The present TFG is based on the development and evaluation of convolutional models for the classification between acoustic scenes in real time. Computer-aided listening is one of the branches of artificial intelligence that is based on emulating the human ability to interpret and understand sound. The extraction of information from audio is important, since it contains important features to be exploited in different functions such as anomaly detection in machines or, as in the present TFG, classifying between different acoustic scenes.

On the other hand, the implementation of real-time models is very relevant for use in devices such as smartphones, embedded systems or microcomputers. Thanks to their low cost and portability they are excellent for practical applications. Therefore, it is important to implement efficient models both in computational cost and accuracy so that it does not pose a problem in the performance of the device. Thus, definitely being able to process and classify between different acoustic scenes in real time opens opportunities for mobile applications, improving the adaptability of audio-based systems.

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la Inteligencia Artificial.

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY: 1.1. Problem statement and opportunity 1.2. Constraints (standards, codes, needs, requirements & specifications) 1.3. Setting of goals	**1. IDENTIFICAR:** 1.1. Planteamiento del problema y oportunidad 1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones) 1.3. Establecimiento de objetivos	S	2, 24-25
2. FORMULATE: 2.1. Creative solution generation (analysis) 2.2. Evaluation of multiple solutions and decision-making (synthesis)	**2. FORMULAR:** 2.1. Generación de soluciones creativas (análisis) 2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	16-21, 26-24
3. SOLVE: 3.1. Fulfilment of goals 3.2. Overall impact and significance (contributions and practical recommendations)	**3. RESOLVER:** 3.1. Evaluación del cumplimiento de objetivos 3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	46-58

A mi familia y amigos.

Índice general

II Anexos

Índice de figuras

Índice de tablas

Parte I

Memoria

Capítulo 1

Introducción

En los pasados años, la Inteligencia Artificial ha avanzado de manera significativa en diferentes hábitos como la visión por computador y el procesamiento natural del lenguaje. Recientemente, una de las áreas por las que se tiene especial interés es la audición por computador, en donde se busca emular la capacidad humana para entender e interpretar los sonidos.

Uno de los problemas en este campo es la clasificación de escenas acústicas que se centra en la identificación y categorización de diferentes entornos sonoros a partir de diversas grabaciones de audio. Para esta clasificación, es realmente importante la capacidad de extraer la información relevante que se encuentra en el audio, debido a que cada sonido contiene características que pueden ser utilizadas para mejorar la funcionalidad de los sistemas automatizados.

Uno de los objetivos principales es la implementación de modelos que además de ser precisos, también deben ser eficientes a nivel de coste computacional. Esto es crucial ya que la finalidad es poder utilizar esos modelos en dispositivos pequeños como *smartphones* o microcomputadoras debido a que abre un abanico de soluciones portátiles y económicas que pueden integrarse día a día a un nivel usuario. Además de suponer una gran mejora en términos de privacidad y seguridad ya que el audio es procesado en el mismo dispositivo sin necesidad de procesamiento en la nube.

1.1. Planteamiento del problema

La audición por computador y en concreto la clasificación de escenas acústicas en tiempo real presenta una serie de desafíos, ya que se intenta implementar modelos en dispositivos con recursos limitados. Algunos de los problemas son los siguientes:

- **Coste computacional**: los modelos que se implementan deben ser eficientes para poder ejecutarse en tiempo real con dispositivos con una capacidad de procesamiento limitada.
- **Precisión**: este tipo de modelos requieren que su precisión y robustez sea alta, debido a que los entornos acústicos pueden variar de manera significativa y pueden aparecer interferencias o ruidos no deseados.
- **Portabilidad**: se deben diseñar los modelos de manera que el hardware donde se implemente pueda ser portable y de un bajo costo.

1.2. Audición por computador

La audición por computador o percepción auditiva artificial es el campo del conocimiento que tiene como objetivo la extracción de información relevante a partir de señales de audio. Al igual que en otros dominios, las soluciones de este campo del estado del arte se basan en técnicas de Inteligencia Artificial (IA). En resumen, la audición por computador es el campo que estudia como emular la capacidad humana de entender e interpretar los sonidos. En esta rama, la audición por computador procesa y analiza las señales de audio para la extracción relevante de características y tomar decisiones en base a ellas. Los procesos clave en la audición por computador son:

1. **Obtención de los datos de audio**: el proceso empieza por la captura de sonidos mediante micrófonos.

2. **Procesamiento del audio**: es un paso importante ya que se puede eliminar las interferencias que se hayan podido dar del paso anterior.

3. **Obtención de las características**: extracción de características relevantes presentes en las señales acústicas. Un ejemplo puede ser la obtención del espectrograma para la conversión bidimensional del audio.

4. **Modelado**: el uso de modelos de aprendizaje automático para la aplicación que se quiera realizar

5. **Evaluación**: para comprobar la precisión de los modelos y si es necesario realizar algunos cambios para optimizarlos.

Capítulo 2

Estado del arte (SoTA)

En este capítulo se repasarán los orígenes de la Inteligencia Artificial (IA), se explicarán los conceptos básicos sobre Aprendizaje Máquina o *Machine Learning* (ML) en inglés y Aprendizaje Profundo o *Deep Learning* (DL) en inglés. También, se expondrá el contexto del concurso DCASE (*Detection and Classification of Acoustic Scenes and Events*) y se explicarán todos los modelos implementados en este trabajo

2.1. Introducción al Aprendizaje Profundo

Las soluciones mayoritariamente presentes en el estado del arte se basan en técnicas de *Deep Learning*. No obstante, tal y como se ve en la Figura 2.1 [1], el campo de la Inteligencia Artificial (IA) ha sufrido enormes aportaciones durante su corta realidad. En este apartado se hará un repaso general de este campo de conocimiento además de explicar una serie de conceptos básicos.

[1] Link imagen Figura 2.1

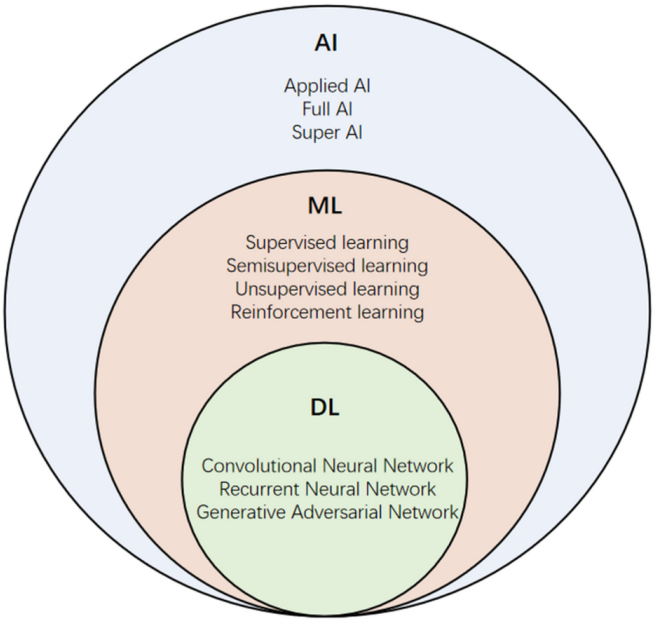


Figura 2.1: Relaciones entre IA, ML y DL

2.1.1. Historia de la Inteligencia Artificial

Actualmente, las soluciones basadas en Inteligencia Artificial (IA) se pueden encontrar en prácticamente todos los entornos tecnológicos: como la salud [1], ayudando en los diagnósticos médicos, o en transporte [2], con la conducción autónoma. Aunque pueda parecer que esta tecnología es muy reciente, la base teórica viene de varios años atrás.

En 1943 se publicó el primer artículo científico [3] dónde se propuso un modelo matemático que trataba de imitar una neurona humana. El trabajo presentado en [4] introdujo la regla de Hebb cuya idea principal es que las conexiones neuronales se refuerzan cada vez que se utilizan, un concepto fundamentalmente esencial para la forma en que los humanos aprenden. En 1950 aparecieron los primeros programas informáticos que aplican técnicas de IA [5]. En 1956 tuvo lugar la Conferencia de Dartmouth dirigida por John McCarthy donde se congregaron los investigadores más influyentes del momento y se definió por primera vez el término Inteligencia Artificial. En 1958 se diseñó el perceptrón [6]. En 1965 se diseñó la primera red neuronal multicapa, lo

que provocó el nacimiento del Aprendizaje Profundo o *Deep Learning* en inglés [7]. No obstante, esta tecnología no despuntó en esos años debido a la poca potencia de los computadores y las escasas bases de datos públicas [8]. A pesar de las complicaciones, durante este tiempo se realizaron varias contribuciones [9] donde se propuso un algoritmo de retropropagación o *back-propagation* en inglés para redes neuronales de varias capas. A partir de 1980 las computadoras fueron más potentes, dando lugar a los métodos modernos en IA.

2.1.2. Aprendizaje Máquina y Aprendizaje profundo

La importancia de la Inteligencia Artificial (IA) ha sido muy notable los últimos años. Esto ha conducido a una gran propuesta de algoritmos y técnicas (validación, partición del conjunto de datos, etc.). Respecto a los algoritmos, estos se pueden clasificar acorde a su diseño. En primer lugar, tenemos los algoritmos de Aprendizaje de Máquina o *Machine Learning* (ML) que son aquellos modelos que tienen ciertos parámetros ajustables cuyos valores se calculan a partir de datos de entrenamiento y distintas técnicas de optimización. Algunos ejemplos de estos algoritmos pueden ser las Máquinas de Vectores de Soporte o *Support Vector Machines* en inglés [10] o los Árboles de Decisión o *Decision Trees* [11] en inglés. Por otra parte, encontramos los algoritmos de *Deep Learning* (DL) que cumplen los requisitos de estar formados por Redes Neuronales y tener al menos una capa intermedia de estas redes. Los algoritmos de *Deep Learning* están basados en diferentes tipos de Redes Neuronales como Redes de Aprendizaje Profundo o *Deep Neural Networks* (DNNs) [7] en inglés, Redes Neuronales de Convolución o *Convolutional Neural Networks* (CNNs) [12] en inglés entre muchas otras y Redes Neuronales Recurrentes o *Recurrent Neural Networks* (RNNS) [13] en inglés.

Dependiendo de la información disponible en el conjunto de datos, ambos algoritmos pueden ser entrenados de forma supervisada o de forma no supervisada, a continuación se muestra en mayor detalle cada caso:

- **Aprendizaje supervisado**: cada muestra del conjunto de datos o *Dataset* en inglés tiene asignada una etiqueta o *label* en inglés. El objetivo de estos algoritmos es crear una función que prediga (mapee) a partir de un dato de entrada la correspondiente etiqueta de salida. Algunos ejemplos son problemas de clasificación (asignando un dato de entrada a una clase específica) o problemas de regresión (prediciendo un valor real a partir de un dato de entrada.

- **Aprendizaje no supervisado**: las muestras del conjunto de datos no tienen ninguna etiqueta asignada. El objetivo principal de estos algoritmos es encontrar similitud entre los diferentes datos del dataset. El principal uso es el *clustering* (agrupación de los datos debido a su similitud).

Dependiendo del número de etiquetas presentes en el conjunto de datos, los aprendizajes supervisados se pueden subdividir en dos categorías.

- **Clasificación binaria**: es el problema de clasificación más simple, incluye únicamente dos clases.

- **Clasificación multiclase**: este problema presenta más de dos clases y cada dato pertenece únicamente a una clase.

En este TFG se proponen soluciones de DL basadas en arquitecturas de CNNs sobre un conjunto de datos supervisado multiclase.

2.1.3. Conceptos básicos del Aprendizaje Profundo

En este apartado se introducirán los conceptos básicos de las redes neuronales del estado del arte. También se explicarán los dos procesos principales que realizan. Estos son, el entrenamiento (ajuste de parámetros) y la predicción o inferencia (parámetros fijos y obtención de resultados sobre un conjunto de muestras).

Perceptrón

El perceptrón [6] o también llamado neurona consiste en una estructura que presenta varios parámetros de entrada, los cuales procesan los datos para producir un resultado a la salida. Este resultado puede ser la predicción (si es la última capa de la red) o la entrada a otra neurona. Las operaciones que realiza una neurona son las siguientes (ver Figura 2.2 para una explicación visual)

1. Multiplicación de los valores de entrada ($X_n = \{x_1, x_2, \ldots, x_n\}$) con los valores ajustables de las neuronas (o pesos, *weights* en inglés) ($W_n = \{w_1, w_2, \ldots, w_n\}$). Cada valor se multiplica elemento a elemento y por último se suma un factor de *bias* (b).

2. Sumatorio de todos los elementos resultantes del punto anterior.

3. Aplicación de la función de activación al resultado del paso anterior. Con este proceso, se consigue un fenómeno de no linealidad, lo que permite el aprendizaje de funciones complejas.

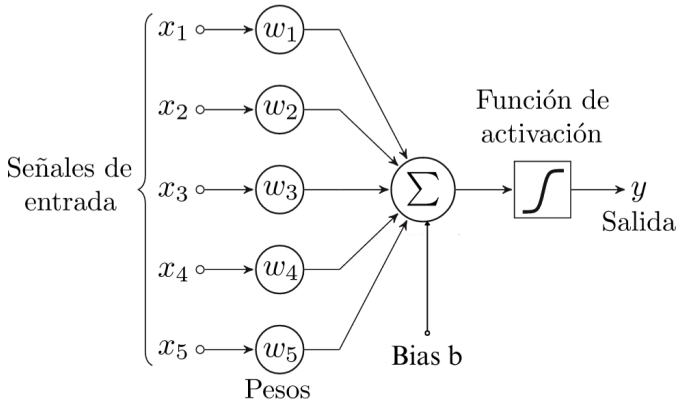


Figura 2.2: Estructura de un perceptrón simple

Sintetizando la explicación anterior así como la Figura 2.2, la fórmula matemática del perceptrón sería:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

Perceptrón multicapa

La combinación en serie de varios perceptrones es conocido en el estado del arte como red neuronal profunda o *Deep Neural Network* (DNN) [14]. La DNN viene definida por una serie de capas donde en cada una hay un número determinado de neuronas tal y como aparece en la Figura 2.3

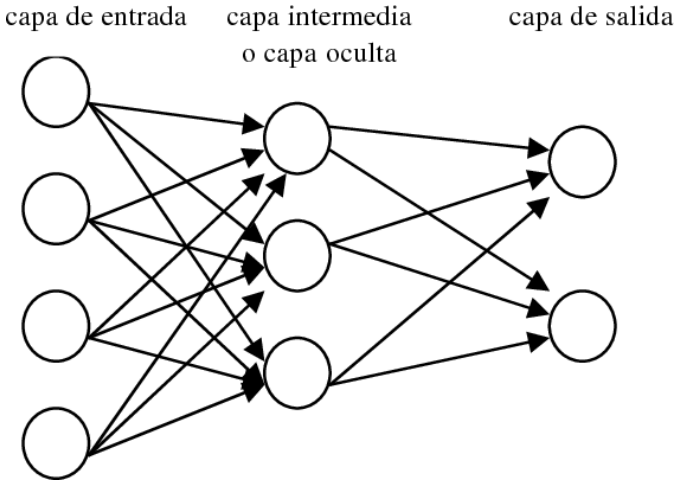
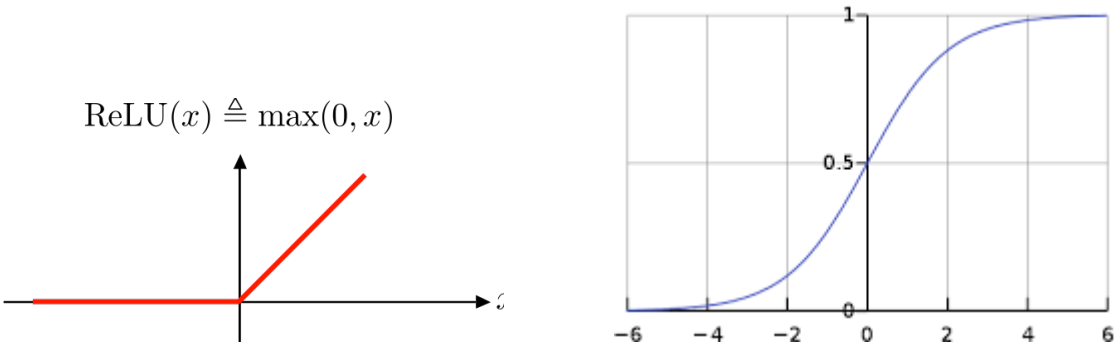


Figura 2.3: Estructura de un perceptrón multicapa con una capa intermedia

Para que la red sea considerada como profunda debe estar compuesta como mínimo por tres capas: una de entrada, una o más capas intermedias (capas ocultas o *hidden layers* en inglés) y una última capa de salida. Cada neurona de una capa concreta está conectada a todas las neuronas de la capa anterior y posterior a través de conexiones ponderadas (ver Ecuación 2.1). La capa de entrada tendrá tantas neuronas como características queramos introducir (representación de la muestra de entrada). La capa de salida estará diseñada con un número de neuronas igual al número de clases que se desean clasificar.

Funciones de activación

Una función de activación es una función matemática utilizada en cada neurona de la red para introducir no linealidad y permitir que el modelo aprenda relaciones no lineales entre los datos de entrada y salida. Existen diferentes tipos de funciones de activación (ver Figura 2.4), pero las más utilizadas en el estado del arte son la unidad lineal rectificada ReLu [15] (que se usa mayoritariamente en las capas ocultas) y Softmax [16] que se usa en la última capa para producir una distribución de probabilidad sobre varias clases. Esta activación se utiliza en los problemas de clasificación donde la suma de probabilidades de todas las neuronas es igual a 1 y la red mostrará un mayor grado de certeza sobre la neurona que tenga una probabilidad mayor.



(a) Función de activación ReLU **(b) Función de activación Soft Max**

Figura 2.4: Funciones de activación usadas en el proyecto

Redes convolucionales

Este tipo de redes han obtenido una gran popularidad en los últimos años debido a que se han utilizado con éxito en determinados problemas que conllevan procesamiento de imágenes. En nuestro caso audio, ya que podemos representarlo como una matriz de 2 dimensiones mediante el espectrograma.

Este tipo de arquitectura (ver Figura 2.5) está basada en la organización del sistema visual del cerebro humano y utilizan capas convolucionales [15] que aplican operaciones de convolución.

Esta operación implica deslizar una ventana o *kernel* en inglés sobre los datos de entrada y multiplicar los valores que se encuentran en la ventana por los valores de los pesos correspondientes a la capa convolucional. Posteriormente, se realiza la suma de las ponderaciones para tener un único valor que representa la activación de la neurona correspondiente a las siguientes capas.

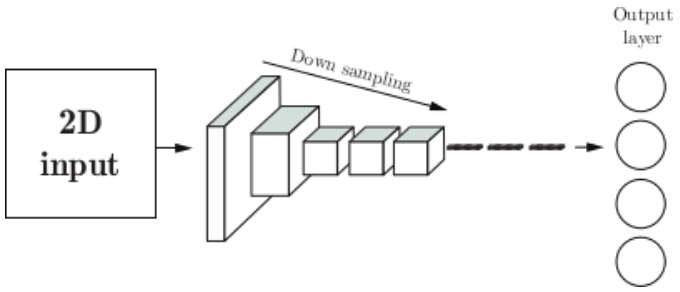


Figura 2.5: Estructura de una red básica convolucional [17]

Las capas convolucionales presentan los siguientes parámetros:

1. Tamaño de la ventana (*kernel size* en inglés): los valores más usuales para CNN que trabajan con entradas en 2D son 3x3 o 5x5.

2. Paso (*stride* en inglés): controla el desplazamiento del filtro (*kernel*) que se mueve sobre los datos de entrada. Suele ser de 1x1

3. Número de canales: número de filtros que conforman la capa convolucional

4. Relleno (*padding* en inglés): determina el tamaño de la salida de una capa convolucional. Se refiere a la adición de ceros alrededor de la entrada justo antes de aplicar la operación de convolución. Esto se realiza porque si no el tamaño de los datos de salida se reduciría en comparación con el de la entrada.

Donde la ecuación de una capa convolucional sería la siguiente:

$$z(x,y) = f\left(\sum_{i=1}^{k_H}\sum_{j=1}^{k_W} K_{i,j}a_{x+i-1,y+j-1} + b_{x,y}\right) \tag{2.2}$$

Por otra parte, este tipo de capas suelen ser seguidas por capas de submuestreo o *pooling* en inglés. Lo que hacen es reducir el tamaño de los datos al tomar el valor máximo o promedio de un conjunto de datos vecinos. Además, este tipo de arquitecturas se suelen combinar con DNNs al final de las capas convolucionales para producir la predicción final de la red como un clasificador

2.1.4. Fases de una solución de IA

Conjunto de datos

Para desarrollar soluciones de IA es necesario, en primer lugar, recopilar los datos correspondientes según nuestra aplicación. Lo siguiente será separar el conjunto de datos en dos subconjuntos entrenamiento y test. El subconjunto de entrenamiento o *training dataset* en inglés, es el encargado para entrenar el modelo. Por otro lado, el subconjunto de test o *test dataset* en inglés, se utiliza para evaluar el rendimiento del modelo. Este conjunto de datos representa los datos que el modelo no ha visto durante la fase de entrenamiento y suele ser entre un 20 % y 30 % del conjunto total de datos.

Aprendizaje

El diseño de una red conlleva una serie de parámetros que se han inicializado de forma aleatoria. El primer proceso al que se enfrenta una red neuronal (ya sea DNN o CNN) es el de ajuste de parámetros sobre el problema que se desea resolver. Para ello, se utiliza el conjunto de datos de entrenamiento. Los pasos durante este proceso de entrenamiento son:

1. Cálculo del error (*loss* en inglés) mediante una función de pérdida. Una vez los datos pasan por la red (proceso llamado *forward propagation* en inglés) y se obtiene la salida,

esta se compara con la salida esperada. La diferencia entre la salida de la red neuronal y la salida esperada se mide usando la función de pérdida. El principal objetivo es minimizar la función de pérdida para tener una buena predicción.

2. Retropropagación del error (*Backward propagation* en inglés): en este paso la función de pérdida se utiliza para calcular el gradiente de los parámetros de la red neuronal. El gradiente se propaga hacia atrás a través de la red y se usa para actualizar los parámetros, utilizando algoritmos de optimización como Adam [18] o RMSprop[19] entre otros.

3. Los pasos 1 y 2 se van realizando de forma continua hasta que finaliza el proceso de entrenamiento. La finalización puede venir definida porque se establece el número de iteraciones que se deben realizar o porque se ha establecido una regla lógica de mejora de la pérdida. Por ejemplo, se para el proceso de entrenamiento si no mejora la pérdida durante un número concreto de iteraciones.

Predicción

La inferencia o predicción (*forward propagation*) es el proceso por el que una red neuronal predice una salida. Para ello, la entrada se propaga hacia delante a todas las capas de la red. Cada capa procesa la entrada para que sea compatible con la siguiente capa. Mientras que cada neurona realiza la suma ponderada de las entradas y a partir de la función de activación no lineal produce la predicción.

2.1.4.1. Métricas

Para poder evaluar la eficacia de los modelos de IA se establecen una serie de métricas tanto *training* como en *test*. Las de training nos darán a entender que el modelo está aprendiendo de manera correcta, mientras que las métricas obtenidas de test (cuando el modelo ha finalizado el entrenamiento) nos permite analizar el rendimiento.

Las principales métricas son:

1. Precisión (*accuracy* en inglés): corresponde al cociente entre el número de muestras clasificadas correctamente y el número total de muestras. Se puede formular como:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

   - **TP:** Verdaderos positivos
   - **TN:** Verdaderos negativos
   - **FP:** Falsos positivos
   - **FN:** Falsos negativos

2. *Loss*: se ha usado en concreto la *categorical crossentropy* porque es una de las más utilizadas para problemas de clasificación. La característica principal es que penaliza más fuerte a las predicciones incorrectas con probabilidades altas. Se formula como:

$$L(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i) \tag{2.4}$$

   - L: función de pérdida.
   - y: vector de etiquetas codificado en *one-hot* (representación binaria de la clase real).
   - $\hat{y}$: vector de las probabilidades predichas por el modelo para cada clase.

3. Matrices de confusión: este método de evaluación consiste en representar las filas como las clases reales y las columnas representan las clases predichas por el modelo y compararlas para obtener la *accuuracy* de la clase.

2.1.4.2. Flujo de una solución de Audición por Computador

La audición por computador o *machine listening* en inglés se podría definir como el campo que estudia diferente tipos de algoritmos que sean capaces de extraer información relevante a partir de datos de audio mediante una computadora. Este campo multidisciplinar se compone de diferentes ámbitos como el tratado de señales digitales, reconocimiento de patrones o técnicas de Inteligencia Artificial. Algunas tareas en el campo del *machine listening* son la detección de eventos ecústicos o *Acoustic Event Classification* [20] en inglés o la clasificación de escenas acústicas *Acoustic Scene Classification* [21] en inglés.

La mayoría de flujo o *pipelines* en inglés de *machine listening* se pueden dividir en 5 pasos:

- Audio *streaming*: consiste en capturar el audio a través de micrófonos. La frecuencia de muestreo (*sampling rate* en inglés) y la profundidad de bit determinan el nivel de resolución.
- Segmentación: cuando se ha capturado el audio se debe segmentar para poder ser procesado (es necesario que la longitud del audio sea constante para un procesado eficiente). Una opción podría ser usar algoritmos de *triggering* [22].
- Extracción de características (espectrograma): debido a que las CNN han demostrado una muy buena eficacia clasificando imágenes [23], las soluciones del estado del arte proponen convertir el audio 1D a una representación espectro-temporal donde un eje corresponde a la frecuencia y el otro al tiempo. Actualmente encontramos muchas opciones como espectrogramas basados en filtros Mel [24, 25, 26], filtros Gammatone [27] o *Leaf* [28].

- Clasificación: este paso es el responsable de dar un resultado a partir de los datos de entrada. Para ello se usan diferentes técnicas de DL como el uso de CNNs.

Para este TFG se ha programado una interfaz de usuario *graphical user interface* en inglés (GUI). Su funcionamiento consiste en el cargado de audio y la realización de una predicción. Añadiendo así un paso extra al flujo anterior.

2.2. Marco de investigación

Debido a que el interés de la comunidad científica (académica y empresarial) en la audición por computador ha aumentado estos últimos años, se ha creado el evento Detección y Clasificación de Escenas y Eventos Acústicos o *Detection and Classification of Acoustic Scenes and Events* en inglés (DCASE) para abordar los problemas derivados en la capacidad de las máquinas para detectar, clasificar y entender sonidos en entornos diversos.

La primera edición ocurrió en el año 2013 y fue organizado por el Centro para Música Digital de la universidad Queen Mary de Londres y por el IRCAM (*Institut de Recherche et Coordination Acoustique/Musique*) de París. La siguiente edición no ocurrió hasta 2016, pero posteriormente se han ido celebrando todos los años a partir de 2020.

El evento se basa en el desafío (*challenge* en inglés) y el taller (*workshop* en inglés):

- Desafío: consiste en una competición donde equipos internacionales proponen y desarrollan soluciones para diferentes tareas propuestas por el evento.
- Taller: consiste en un encuentro donde diferentes perfiles de investigadores y académicos presentan sus trabajos y discuten los resultados del desafío.

En el caso del presente TFG se realizó la tarea 1 del año 2022: Clasificación de escenas acústicas de baja complejidad (*Low-Complexity Acoustic Scene Classification* en inglés). La finalidad es

poder clasificar escenas acústicas a partir de una grabación de prueba en una de las 10 clases de escenas acústicas. La tarea se enfoca en modelos cuyo coste computacional y memoria sea bajo. Además, el modelo debe generalizar bien entre diferentes dispositivos, ya que los datos de audio han sido grabados por una variedad de dispositivos.

2.2.1. Ediciones previas

En la anterior edición del DCASE (2020) aparte de cambiar el dataset por "TAU Urban Acoustic Scenes 2020 Mobile, development dataset" se establecieron una serie de requisitos respecto al diseño del modelo de IA. El requisito más importante era que no se podían usar modelos con un tamaño mayor a 128kB en los parámetros que no fuesen cero (*non-zero* en inglés). Esto se traduce en un límite de 32768 parámetros cuando se usan de tipo *float* de 32 bits. Tal y como se muestra en la Ecuación 2.5.

$$\frac{32768 \text{ parámetros} \times 32 \text{ bits por parámetro}}{8 \text{ bits por byte}} = 131072 \text{ bytes} = 128 \text{ kB} \tag{2.5}$$

Dicho esto, el modelo que ganó en el año 2020 [29] combina diferentes estrategias como aumento de datos (*data augmentation* en inglés) [30], transferencia de conocimiento (*knowledge transfer* en inglés), prunado (*pruning* en inglés) [31] y cuantificación. El proceso de prunado se realizó gracias a un sistema llamado *Lottery Ticket Hypothesis* (LTH) para encontrar una subred asociada con una cantidad pequeña de parámetros *non-zero*. Por ello, realizaron un *framework* llamado *Acoustic Lottery* (ver Figura 2.6) [2] que podía comprimir un modelo a $1/10^4$. Los resultados fueron los mejores, obteniendo un 79,4 % de *accuracy* y un 0.64 de *Log Loss*.

[2] Link imagen Figura 2.6

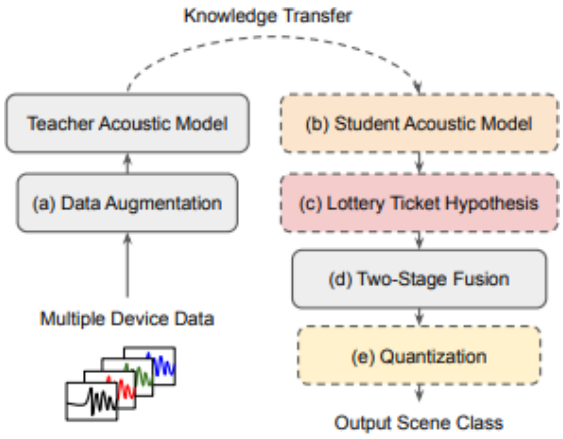


Figura 2.6: Pipeline del modelo que usa la transferencia de conocimiento con técnicas de *data augmentation*

Por otra parte, el trabajo presentado en [17] se basó en utilizar audio como una representación en dos dimensiones llamadas *Gammatone Filter Bank* y una red convolucional utilizando técnicas de compresión y excitación o *squeeze-excitation* en inglés. Este trabajo no propone ninguna otra estrategia de aumentado de datos o de pruning como el trabajo anterior. Se centra más en el análisis de este banco de filtros y en aplicar técnicas avanzadas en la propia CNN (ver Tabla 2.1). Con estas estrategias se consiguió un *accuracy* del 64,18 % y un tamaño de 95,96kB.

Gammatone representation
Conv-StandardPost
Max pooling
Dropout
Conv-SantardPost
Max Pooling
Dropout
Global average pooling
Classification

Tabla 2.1: Arquitectura del modelo para clasificación basada en la representación Gammatone

2.3. Arquitecturas CNNs analizadas

Debido a la restricción de la tarea de que no se podía exceder de un determinado número de parámetros (entre otros requisitos) se usaron diferentes arquitecturas que supusiera un coste computacional bajo. En las siguientes subsecciones se introducen las distintas CNNs estudiadas en el marco de este TFG.

2.3.1. Conv-Sep

Este modelo se basa en la arquitectura VGG [32]. Se apilan dos capas convolucionales seguidas de capas *pooling* (reducción de la dimensionalidad) y capas de *dropout* (evitan el sobreajuste apagando aleatoriamente un porcentaje de las neuronas. Por otra parte, se ha aplicado entre la operación de convolución y la de activación una capa *batch normalization* que se usa para normalizar las salidas de las operaciones de la convolución para mantener la distribución controlada de las activaciones a lo largo de la red. No obstante, se han introducido una serie de modificaciones respecto a una red VGG convencional. Las capas de activación ReLU se sustituyen por capas ELU [33]. Para cumplir los requisitos de complejidad impuestos en el *Challenge*, la segunda capa convolucional se sustituye por una capa de convolución separable [23]. Así, el módulo convolucional puede verse en la siguiente tabla 2.2:

Conv-Sep
Conv Layer (number of filters)
Batch Normalization
ELU
Separable Conv Layer (number of filters)
Batch Normalization
ELU

Tabla 2.2: Arquitectura del módulo convolucional de Conv-Sep

2.3.2. ConvMixer

La arquitectura ConvMixer [34] (ver Figura 2.7) [3] utilizada tiene tres partes. La primera parte consiste en un *patch embedding* que no es más que una capa convolucional con el tamaño de *kernel* y *stride* equivalentes al *patch size* y con c canales de entrada y h canales de salida. Todo esto seguido de una activación GELU [35] y un *batch normalization*. La segunda parte está compuesta por el bloque ConvMixer (o capa ConvMixer) que es repetido según el valor de la variable *depth* veces. Esta capa consiste en un bloque residual el cual toma la salida de la capa de convolución Depthwise, la concatena a las entradas originales y pasa por la función de activación GELU y una *batch normalization*. Seguidamente, la salida del anterior bloque residual pasa por una convolución Pointwise, la función de activación GELU y otro *batch normalization*. Finalmente, la tercera parte está formada por una capa *global pooling* y el clasificador Softmax con el número de clases.

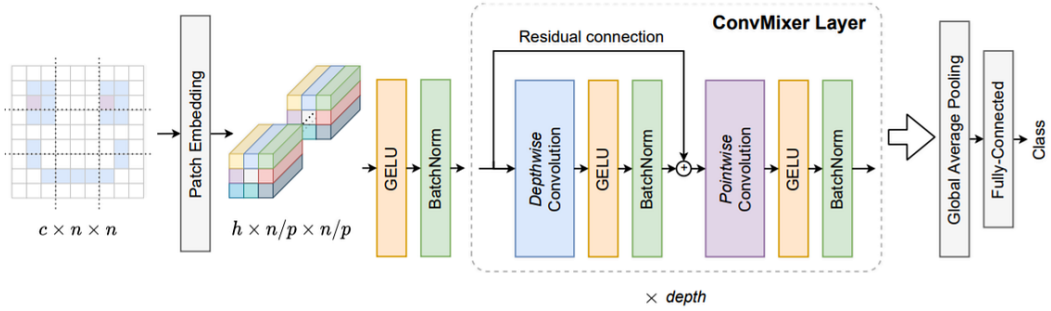


Figura 2.7: Estructura del modelo ConvMixer

2.3.3. Red Lambda *(LambdaNet)*

Para este modelo se usó una arquitectura con capas Lambda [36] proporcionada por el repositorio de Lucidrains [4]. Estas redes son una alternativa a las CNN y redes de atención (como Transformers[37]). La principal característica, es que puede buscar relaciones a largo plazo en los datos

[3] Link imagen Figura 2.7
[4] Repositorio lucidrains

mediante mecanismos de autoatención, teniendo además una eficiencia computacional más alta que las CNN o Transformers. La estructura general (ver Tabla 2.3) del modelo se compone en primer lugar en una capa Lambda que se repite según el número de filtros *kernel* que se le asigne, seguida de una capa de normalización batch y una función de activación ELU. A continuación, se aplica una capa de *pooling* para reducir la dimensionalidad y *Dropout* para prevenir el sobreajuste. Finalmente, se usa una capa *global average pooling* (sirve para calcular el promedio de todas las características en cada canal) y una capa densa para obtener la salida (clasificación) del modelo

Lambda Network
Lambda layer
Batch Normalization
ELU
MaxPool2D
Dropout
GlobaleAveragePooling2D
Dense Layer

Tabla 2.3: Arquitectura del modelo Lambda

2.3.4. Squeeze Excitation

Este tipo de arqutectura [38] introduce un mecanismo de atención a través de dos operaciones: *Squeeze* y *Excitation* [39]. El objetivo principal de esta red es mejorar la representación de características en las CNNs, usando una recalibración de las características más relevantes según la tarea específica. Por ejemplo, en el caso de este proyecto es relevante poder detectar características como ciertas frecuencias específicas, patrones temporales, etc. Por ello, este tipo de red podría recalibrar los canales que mejor capturan esas frecuencias y ciertos patrones temporales que sean relevantes.

Las operaciones mencionadas anteriormente consisten en:

- Squeeze: esta operación reduce las dimensiones espaciales de las características en una

más compacta.

- Excitation: esta operación se encarga de generar los pesos correspondientes para cada canal con las características comprimidas. Además, estos pesos se aplican a los canales originales, ajustando su importancia relativa para mejorar la capacidad de la red centrándose en las características más importantes.

La estructura del modelo consiste principalmente en el bloque llamado *convStandardPost* (ver Figura 2.8)[5]. Este bloque implementa una serie de redes convolucionales estándar (red residual) con los mecanismos de atención Squeeze Excitation. En más detalle, se realizan dos operaciones convolucionales estándar (con su normalización y función de activación). Posteriormente, se realizan las operaciones mediante los mecanismos de atención Squeeze Excitation que recalibran la importancia de las características de cada canal y un ajuste para garantizar la compatibilidad dimensional en su salida (*Scale*). Finalmente, se combinan las características recalibradas con la de los canales originales.

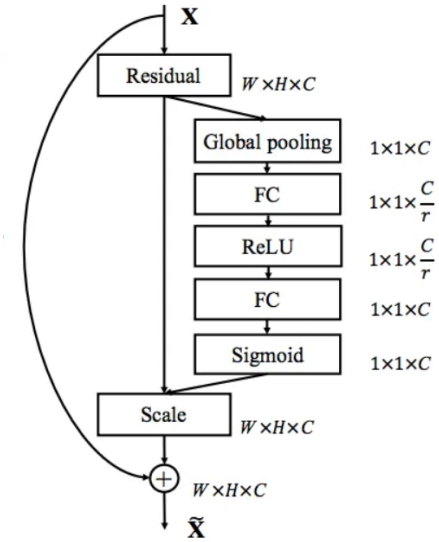


Figura 2.8: Bloque convStandardPost *Squeeze* y *Excitation*

[5]Link imagen Figura 2.8

El segundo bloque, consiste en una capa de *pooling*, seguida de una capa *dropout* y una capa *global average pooling* para terminar con una capa Densa y así obtener la salida del modelo (ver tabla 2.4).

Squeeze Excitation
convStandardPost
MaxPooling2D
Dropout
GlobalAveragePooling2D
Dense

Tabla 2.4: Arquitectura del modelo Squeeze Excitation

Capítulo 3

Objetivos

En el capítulo presente se van a explicar los objetivos (tanto principales como específicos) que son fundamentales para guiar a la investigación. Además, se contextualizará el problema que se está tratando y la motivación para proponer soluciones.

3.1. Motivación

La motivación principal fue crear una aplicación que fuese capaz de extraer información a partir del audio para poder clasificar escenas acústicas. Otras motivaciones fueron el poder contribuir al avance de este campo de la Inteligencia Artificial y explorar diferentes técnicas de clasificación de escenas acústicas para mejorar la precisión y rendimiento de modelos existentes. Además, a un nivel personal desarrollar mis habilidades en el uso de herramientas enfocadas a la IA y al procesamiento de señales. Finalmente, realizar este proyecto para que pueda servir como base para futuras investigaciones.

Para abordar este problema se ha utilizado la tarea de clasificación de escenas acústicas presentada en el concurso del DCASE.

3.2. Objetivos generales y específicos

Los objetivos principales fueron los siguientes:

- Crear una aplicación de Inteligencia Artificial que fuese capaz de clasificar entre diferentes escenas acústicas.
- Obtener las habilidades básicas en herramientas de IA y procesado de señales.

Los objetivos específicos fueron:

- Participar en el concurso y superar el modelo base que proponen.
- Realizar un flujo de entrenamiento y validación parecidos a los que se pueden encontrar en un entorno más laboral.
- Crear un uso práctico y funcional de la aplicación que ha sido una interfaz gráfica.
- Presentar el proyecto asociado al concurso en congresos específicos de audición por computador.

Capítulo 4

Metodología

En este Capítulo se describirán en detalle todos los procedimientos que se han seguido a la hora de desarrollar la solución de audición por computador para esta tarea en concreto del concurso DCASE.

4.1. Preprocesado del audio

El preprocesado de audio es una parte bastante relevante en este tipo de soluciones, ya que será el dato de entrada para los algoritmos de *Deep Learning*. En nuestro caso, se han usado diferentes técnicas pasando los audios a representaciones bidimensionales. Esto es debido a que este tipo de representación presenta ciertas ventajas a la hora de emplear soluciones basadas en redes CNN:

- Los espectrogramas representan contenido espectral del audio de manera muy eficiente, lo que facilita el entrenamiento de los modelos.
- Presentan una reducción de la dimensionalidad respecto a las formas de onda, lo que reduce la complejidad computacional.

Por ello, para este TFG se han usado espectrogramas basados en banco de filtros Mel, espectrogramas basados en bancos de filtros Gammatone y una representación conocida como *Leaf Audio* [1]. Posteriormente, se aplica un proceso de normalización sobre las representaciones para evitar comportamientos de sobreajuste durante el entrenamiento.

4.1.1. Base de datos de escenas sonoras

Para los experimentos se han utilizado los datasets de la tarea del *DCASE Low-Complexity Acoustic Scene Classification* de los años 2022 y 2019.

El dataset del año 2022 está formado por grabaciones realizadas en 12 ciudades europeas compuesto por 10 escenas acústicas diferentes usando 4 tipos de dispositivos de grabación. Además, se han usado datos sintéticos a partir de las grabaciones originales.

Las 10 escenas acústicas son las siguientes:

Escena Acústica	**Etiqueta**
Aeropuerto	airport
Interior de un centro comercial	shopping_mall
Estación de metro	metro_station
Calle peatonal	street_pedestrian
Plaza pública	public_square
Calle con un nível medio de tráfico	street_traffic
Viajando en un tram	tram
Viajando en un bus	bus
Viajando en el metro	metro
Parque urbano	park

Tabla 4.1: Escenas acústicas y etiquetas asociadas del conjunto de datos de la edición 2022.

Las grabaciones fueron realizadas usando los 4 dispositivos a la vez. El principal dispositivo de grabación fue un micrófono Soundman OKM II Klassik/Studio A3 conectado a una grabadora Zoom F8 usando un *sample rate* de 48kHz y 24 bits de resolución. Este micrófono tiene la peculiaridad de ser binaural por lo que las grabaciones se asemejan mucho a las de un oído humano.

[1] Repositorio Leaf Audio

Los dispositivos de grabación secundarios fueron un Samsung Galaxy s7, un iPhone SE y una GoPro Hero 5 Session.

Los datos sintéticos proceden del dispositivo de grabación principal a los que se han aplicado una serie de convoluciones con respuestas de impulso y una compresión del rango dinámico.

El conjunto de datos tiene una duración total de 40 horas dividido en audios de 1 segundo. El audio tiene un único canal y una frecuencia de muestro de 44.1kHz a 24 bits de resolución. Además está dividido según la finalidad de uso: entrenamiento, evaluación/validación y test.

Por otra parte, el dataset de 2019 está formado por los mismos datos que el de 2022 a excepción de los datos sintéticos. Además, los audios fueron segmentados en duraciones de 10 segundos.

4.1.2. Espectrograma con banco de filtros Mel

La primera técnica de preprocesado que se ha utilizado es convertir los audios a espectrogramas empleando un banco de filtros Mel [24, 25, 26] debido a que es una de las representaciones más utilizadas y estudiadas para abordar este tipo de problemas.

El espectrograma de Mel (ver Figura 4.1) es una representación bidimensional de una señal de audio, donde el eje horizontal representa normalmente el tiempo y el eje vertical la frecuencia. Tiene la particularidad de que las frecuencias no se representan de manera lineal, sino que se usa una escala de frecuencias no lineales que tienen como finalidad simular la percepción auditiva humana.

Para obtener el espectrograma de Mel se deben realizar los siguientes pasos:

1. Se divide la señal de audio en pequeños segmentos llamados frames acorde a un tamaño de ventana y un salto entre ventanas. Un ejemplo puede ser un tamaño de ventana de $40ms$ y un salto de $20ms$.

2. Para cada frame se calcula la STFT [40] que obtiene la distribución de energía en el do-

minio de la frecuencia.

3. Aplica un banco de filtros Mel a la salida de la STFT. Estos filtros simulan la respuesta del sonido humano superponiéndose para calcular la energía de las bandas de frecuencia Mel.

4. Por último, se calcula el logaritmo de la energía en cada banda Mel expresado en dBs.

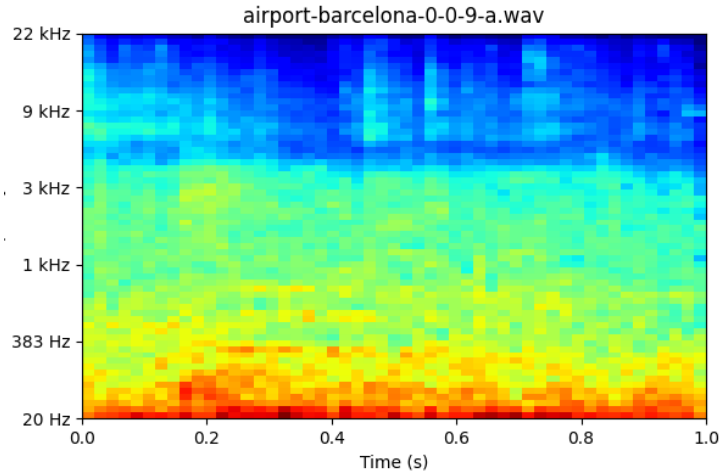


Figura 4.1: Espectrograma de Mel usando un segundo y 64 canales Mel

.

En nuestro caso usamos la librería de Python Librosa [2] para obtener los espectrogramas Mel. Siguiendo las líneas del estado del arte, los parámetros utilizados fueron los siguientes:

[2]Librosa link

Parámetros	Valor
Window Length	4 ms
Hope Length	2 ms
número de filtros	64
Sample Rate	44,1 kHz
Frecuencia mínima	80 Hz
Mono	1 canal
Número de FFTs	1024

Tabla 4.2: Parámetros de los espectrogramas

4.1.3. Espectrogramas con banco de filtros *Gammatone*

La segunda técnica ha sido utilizar un espectrograma Gammatone [27] debido a que modela de manera aproximada a como un oído humano percibe, enfatiza y separa las frecuencias del sonido. Este espectrograma usa una serie de filtros Gammatone cuya anchura aumenta con el incremento de la frecuencia central que son aplicados a una señal en el dominio del tiempo.

Sigue la siguiente fórmula:

$$g(t) = at^{n-1}e^{-2\pi bt}\cos(2\pi ft + \phi) \tag{4.1}$$

Donde f (Hz) es el centro de la frecuencia, ϕ (en radianes) es la fase de la portadora, a es la amplitud, b (Hz) es el ancho de banda de los filtros y t (en segundos) es el tiempo.

Para calcular el espectrograma de Gammatone (ver Figura 4.2) hemos usado la librería Gammatone Filterbank Toolkit [3] con los parámetros usados en el anterior espectrograma (Mel).

Este filtro presenta una respuesta al impulso que es una onda sinusoidal multiplicada por una función de distribución gamma.

[3] Gammatone Filter Bank link

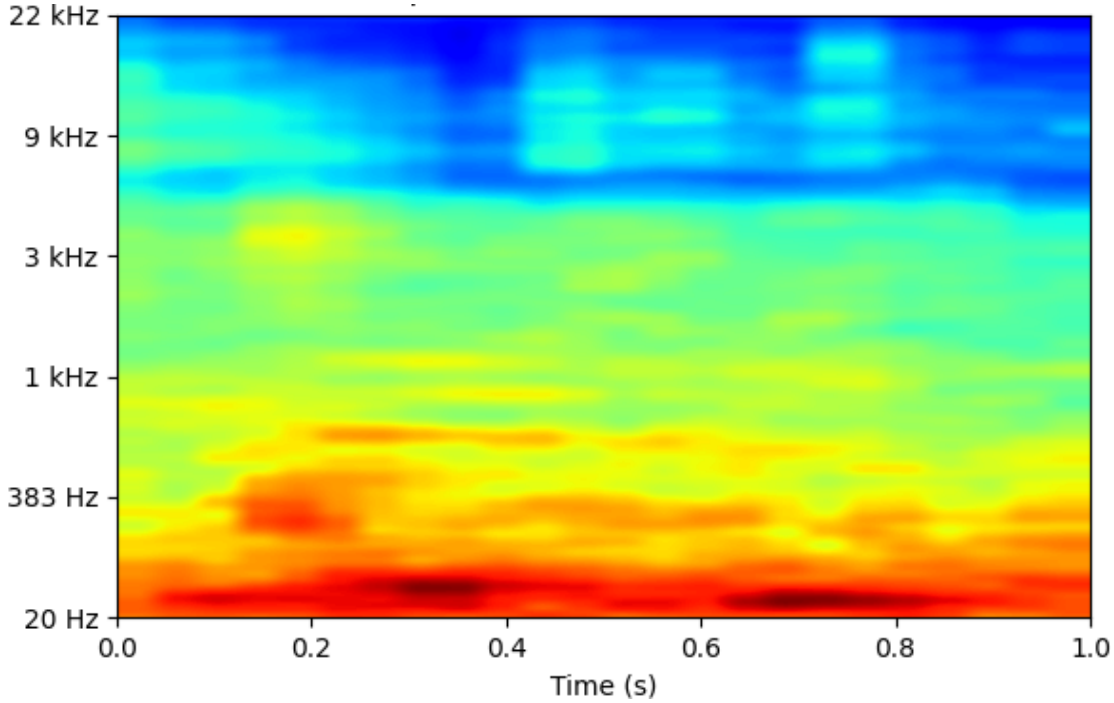


Figura 4.2: Espectrograma de Gammatone usando los parámetros descritos anteriormente.

4.1.4. Representación *Leaf*

La tercera opción fue utilizar la arquitectura Leaf [28] (*Learnable Frontend for Audio Classification*) con la que podemos extraer características de audio. El modelo está diseñado para aprender todas las operaciones de extracción de características del audio, desde el filtrado hasta *pooling*, compresión y normalización. Además, puede ser integrado en cualquier modelo, ya que tiene un coste computacional prácticamente insignificante.

La arquitectura consta en tres componentes: un banco de filtros aprendible, una capa *pooling* y una capa de normalización. El banco de filtros aprendible es un conjunto de capas convolucionales y recurrentes que se encarga de aprender a extraer las características del audio. Está diseñado para captar patrones locales y temporales de la señal de audio. La capa de *pooling* captura las características más destacadas y descarta información redundante. Además, reduce la dimensionalidad de la salida del banco de filtros. La capa de normalización sirve para garantizar que la

salida del modelo sea invariable a los cambios de la señal.

La salida del modelo Leaf es un conjunto de características representadas de manera bidimensional en el que el número de frames y el número de canales vienen determinados por los parámetros del modelo.

Cabe destacar, que este modelo difiere de los bancos de filtros Mel, ya que estos usan características matemáticas realizadas a mano, no aprendidas como Leaf.

4.1.5. Normalización del espectrograma

Los espectrogramas pueden ser tratados de una manera similar a las imágenes, es por ello que normalizarlos es una buena práctica, ya que nos ofrecen ciertas ventajas. En primer lugar, permite una mejor estabilidad del entrenamiento, porque si los datos de entrada no están normalizados las diferencias de amplitudes pueden llevar a que los gradientes en el proceso de *backpropagation* sean muy grandes o muy pequeños, empeorando el proceso de optimización. Por otra parte, la normalización acelera la convergencia, disminuyendo la variabilidad en las características de entrada. En este proyecto se han normalizados los audios de dos formas: por un valor global y por canal.

El algoritmo para normalizar por un valor global tiene los siguientes pasos:

Observamos un espectrograma $X \in \mathbb{R}^{n \times m}$ donde n es el número de muestras y m es el número de características por muestra. El objetivo principal es normalizar cada muestra usando una media acumulativa μ y una desviación estándar acumulativa σ.

Cálculo Acumulativo de la Media y Desviación Estándar

Para calcular la media acumulativa μ y la desviación estándar σ se realiza de la siguiente forma:

Inicialización para $i = 1$:

Para la primera muestra ($i = 1$), la media y desviación estándar iniciales se definen como:

$$\mu_1 = \frac{1}{m} \sum_{j=1}^{m} X_{1,j}, \tag{4.2}$$

$$\sigma_1 = \sqrt{\frac{1}{m} \sum_{j=1}^{m} (X_{1,j} - \mu_1)^2}. \tag{4.3}$$

Donde $X_{1,j}$ representa el valor de la característica j para la muestra 1.

Actualización para $i \geq 2$:

Para cada nueva muestra X_i, calculamos la media μ_i y la desviación estándar σ_i:

$$\mu_i = \frac{1}{m} \sum_{j=i}^{m} X_{i,j}, \tag{4.4}$$

$$\sigma_i = \sqrt{\frac{1}{m} \sum_{j=i}^{m} (X_{i,j} - \mu_i)^2}. \tag{4.5}$$

La media acumulativa $\mu_{\text{acumulativa}}$ y la desviación estándar acumulativa $\sigma_{\text{acumulativa}}$ se actualizan con:

$$\mu_{\text{acumulativa}} = \frac{n}{n+1} \mu_{\text{acumulativa}} + \frac{1}{n+1} \mu_i, \tag{4.6}$$

$$\sigma^2_{\text{acumulativa}} = \frac{n}{n+1} \sigma^2_{\text{acumulativa}} + \frac{1}{n+1} \sigma_i^2 + \frac{n}{(n+1)^2} (\mu_{\text{acumulativa}} - \mu_i)^2, \tag{4.7}$$

$$\sigma_{\text{acumulativa}} = \sqrt{\sigma^2_{\text{acumulativa}}}. \tag{4.8}$$

Finalmente, incrementamos el contador de muestras:

$$n = n + 1. \tag{4.9}$$

Normalización del Espectrograma

Una vez calculadas la media acumulativa $\mu_{\text{acumulativa}}$ y la desviación estándar acumulativa $\sigma_{\text{acumulativa}}$, normalizamos cada muestra X_i del espectrograma X de la siguiente manera:

$$\hat{X}_i = \frac{X_i - \mu_{\text{acumulativa}}}{\sigma_{\text{acumulativa}}}, \quad \forall i \in \{1, 2, \ldots, n\}. \tag{4.10}$$

Este proceso también se aplica a un conjunto de validación X_{val} de forma similar:

$$\hat{X}_{\text{val},i} = \frac{X_{\text{val},i} - \mu_{\text{acumulativa}}}{\sigma_{\text{acumulativa}}}, \quad \forall i. \tag{4.11}$$

El algoritmo para normalizar por canal tiene los siguientes pasos:

Consideremos un espectrograma apilado $X \in \mathbb{R}^{n \times m \times c}$ donde n es el número de muestras, m es el número de características por muestra, y c es el número de canales. El objetivo es normalizar cada canal de forma independiente utilizando una media acumulativa μ_c y una desviación estándar acumulativa σ_c.

Cálculo Acumulativo de la Media y Desviación Estándar por Canal

Para cada canal c, la media acumulativa μ_c y la desviación estándar acumulativa σ_c se calculan como sigue:

Inicialización para $i = 1$:

Para la primera muestra ($i = 1$), se calculan la media inicial $\mu_{1,c}$ y la desviación estándar inicial $\sigma_{1,c}$ de cada canal c:

$$\mu_{1,c} = \frac{1}{m} \sum_{j=1}^{m} X_{1,j,c}, \quad \forall c, \tag{4.12}$$

$$\sigma_{1,c} = \sqrt{\frac{1}{m} \sum_{j=1}^{m} (X_{1,j,c} - \mu_{1,c})^2}, \quad \forall c. \tag{4.13}$$

Donde $X_{1,j,c}$ representa el valor de la característica j para el canal c de la muestra 1.

Actualización para $i \geq 2$:

Para cada nueva muestra X_i, se calculan la media $\mu_{i,c}$ y la desviación estándar $\sigma_{i,c}$ de cada canal c:

$$\mu_{i,c} = \frac{1}{m}\sum_{j=i}^{m} X_{i,j,c}, \quad \forall c, \tag{4.14}$$

$$\sigma_{i,c} = \sqrt{\frac{1}{m}\sum_{j=i}^{m}(X_{i,j,c} - \mu_{i,c})^2}, \quad \forall c. \tag{4.15}$$

La media acumulativa $\mu_{c,\text{acumulativa}}$ y la desviación estándar acumulativa $\sigma_{c,\text{acumulativa}}$ para cada canal c se actualizan con:

$$\mu_{c,\text{acumulativa}} = \frac{m_{\text{acumuladas}}}{m_{\text{acumuladas}} + n}\mu_{c,\text{acumulativa}} + \frac{n}{m_{\text{acumuladas}} + n}\mu_{i,c}, \quad \forall c, \tag{4.16}$$

$$\begin{aligned}\sigma^2_{c,\text{acumulativa}} = {} & \frac{m_{\text{acumuladas}}}{m_{\text{acumuladas}} + n}\sigma^2_{c,\text{acumulativa}} + \frac{n}{m_{\text{acumuladas}} + n}\sigma^2_{i,c} \\ & + \frac{m_{\text{acumuladas}} \cdot n}{(m_{\text{acumuladas}} + n)^2}(\mu_{c,\text{acumulativa}} - \mu_{i,c})^2, \quad \forall c,\end{aligned} \tag{4.17}$$

$$\sigma_{c,\text{acumulativa}} = \sqrt{\sigma^2_{c,\text{acumulativa}}}, \quad \forall c. \tag{4.18}$$

Donde $m_{\text{acumuladas}}$ se inicializa en 501 (el número de características por canal en cada muestra) y se incrementa en 501 después de cada iteración:

$$m_{\text{acumuladas}} = m_{\text{acumuladas}} + n. \tag{4.19}$$

Normalización del Espectrograma por Canal

Una vez calculadas la media acumulativa $\mu_{c,\text{acumulativa}}$ y la desviación estándar acumulativa $\sigma_{c,\text{acumulativa}}$ para cada canal c, cada muestra X_i del espectrograma X se normaliza de la siguiente manera:

$$\hat{X}_{i,j,c} = \frac{X_{i,j,c} - \mu_{c,\text{acumulativa}}}{\sigma_{c,\text{acumulativa}}}, \quad \forall i, j, c. \tag{4.20}$$

Este proceso también se aplica a un conjunto de validación X_{val} de manera similar:

$$\hat{X}_{\text{val},i,j,c} = \frac{X_{\text{val},i,j,c} - \mu_{c,\text{acumulativa}}}{\sigma_{c,\text{acumulativa}}}, \quad \forall i, j, c. \tag{4.21}$$

4.2. Flujo de entrenamiento

El flujo de entrenamiento se puede dividir en 3 fases (ver Figura 4.3). En primer lugar, la obtención de los ficheros *pickle* (tipo de fichero de Python para almacenar datos de manera persistente) donde almacenamos las etiquetas de los audios codificadas usando un *one hot encoder* y los ficheros en formato h5 (tipo de fichero que sirve para almacenar grandes volúmenes de datos de una manera eficiente) donde se almacenan los espectrogramas (Mel, Gammatone y Leaf). El conjunto de estos archivos son los datos de entrada para los modelos. En segundo lugar, la fase de entrenamiento y posteriormente la conversión del modelo entrenado a un archivo más ligero con extensión TFLITE. Finalmente, la realización de inferencias para poder obtener evaluaciones como *accuracy* o matrices de confusión. Para un mejor entendimiento del proceso, explicaremos los pasos más detalladamente.

4.2.1. Obtención de archivos Pickle

Con el archivo CSV (*Comma-Separated Values*) que se nos ha otorgado por el DCASE para esta tarea se ha extraído las etiquetas que contienen la clase de cada dato. Posteriormente, se realizó una codificación *one-hot* para categorizar los datos en una representación binaria. Esta

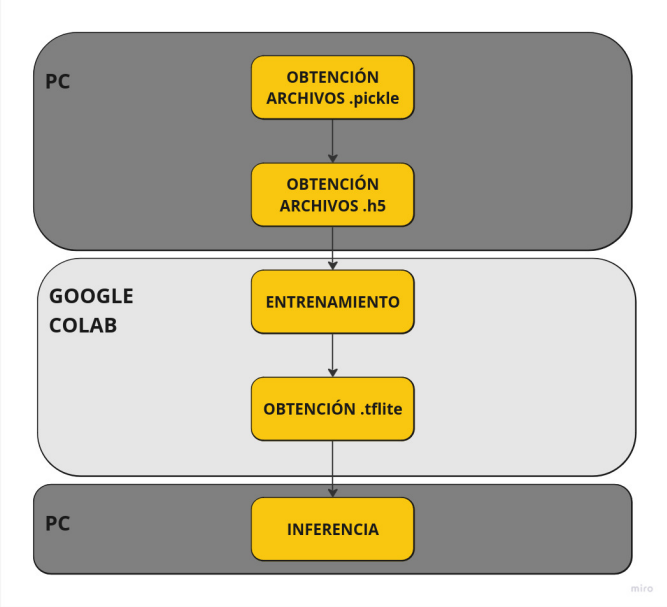


Figura 4.3: Diagrama general del *pipeline*

representación son columnas binarias (0, 1) donde cada columna representa una clase y se usa un valor de 1 en la columna correspondiente a la clase y 0 en el resto de columnas. Esto asegura que cada categoría esté representada de una única manera. Finalmente, se guardaron estos datos en un archivo *pickle* que serializan estos objetos a bytes.

4.2.2. Obtención ficheros h5

A partir de los audios se realizó un preprocesado donde se aseguró que el *sample rate* estuviese a 44,1kHz y si no se resampleaba a esa frecuencia. Después, se obtuvieron los espectrogramas de Mel, Leaf y Gammatone. Para algunos entrenamientos se ha aplicado una normalización global o por canal a los espectrogramas Mel. Finalmente, se almacenaron estos espectrogramas en un archivo h5 que es un formato estándar en el almacenamiento de datos en *machine learning* con la ventaja de organizar los datos de una manera eficiente. Además, se guardaron en el archivo h5 el *label* y *one hot encoder* asociado a cada espectrograma (ver Figura 4.5) Los ficheros h5 están divididos según el tipo de espectrograma y finalidad (entrenamiento, validación y test).

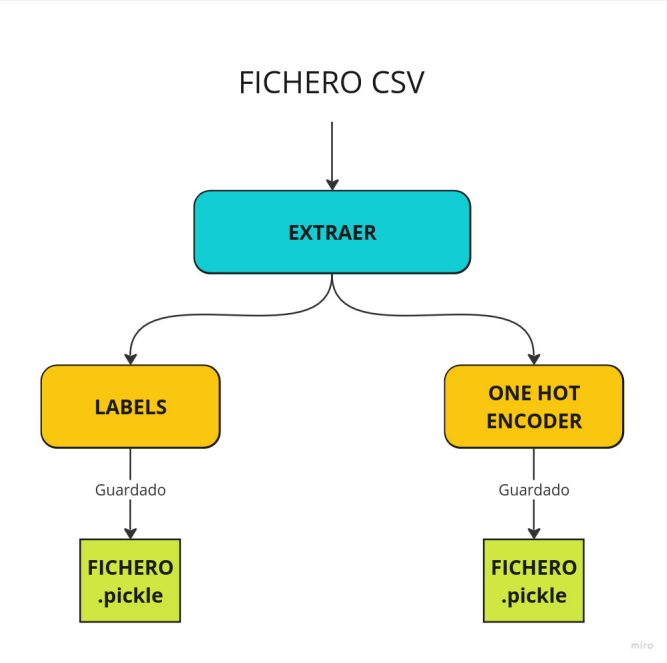


Figura 4.4: Obtención ficheros pickle

4.2.3. Entrenamiento

Para realizar los entrenamientos (ver Figura 4.6) se ha usado *google colab pro*[4] que es un servicio de máquinas virtuales donde se pueden desarrollar y entrenar modelos. Se utilizó la versión pro de pago, ya que permitía utilizar GPUs potentes de entrenamiento. Según la demanda de usuarios se asignaban las NVIDIA A100 y las NVIDIA T4 (menos capacidad de cómputo que las A100).

Por otra parte, se realizó un archivo de configuración (*config* en inglés) universal en el que se definían todos los parámetros (también llamados hiperparámetros en el contexto de *Deep Learning*) de los modelos y el modelo que se iba a usar en el entrenamiento. Se hizo de una manera flexible en la que se podía cambiar cualquier parámetro. En nuestro caso solo se cambió el número de filtros y la dimensión del espectrograma para probar las configuraciones más eficientes y eficaces.

Para el entrenamiento se utilizó varias llamadas de retorno (*callbacks* en inglés) de la librería de

[4]Google Colab Pro link

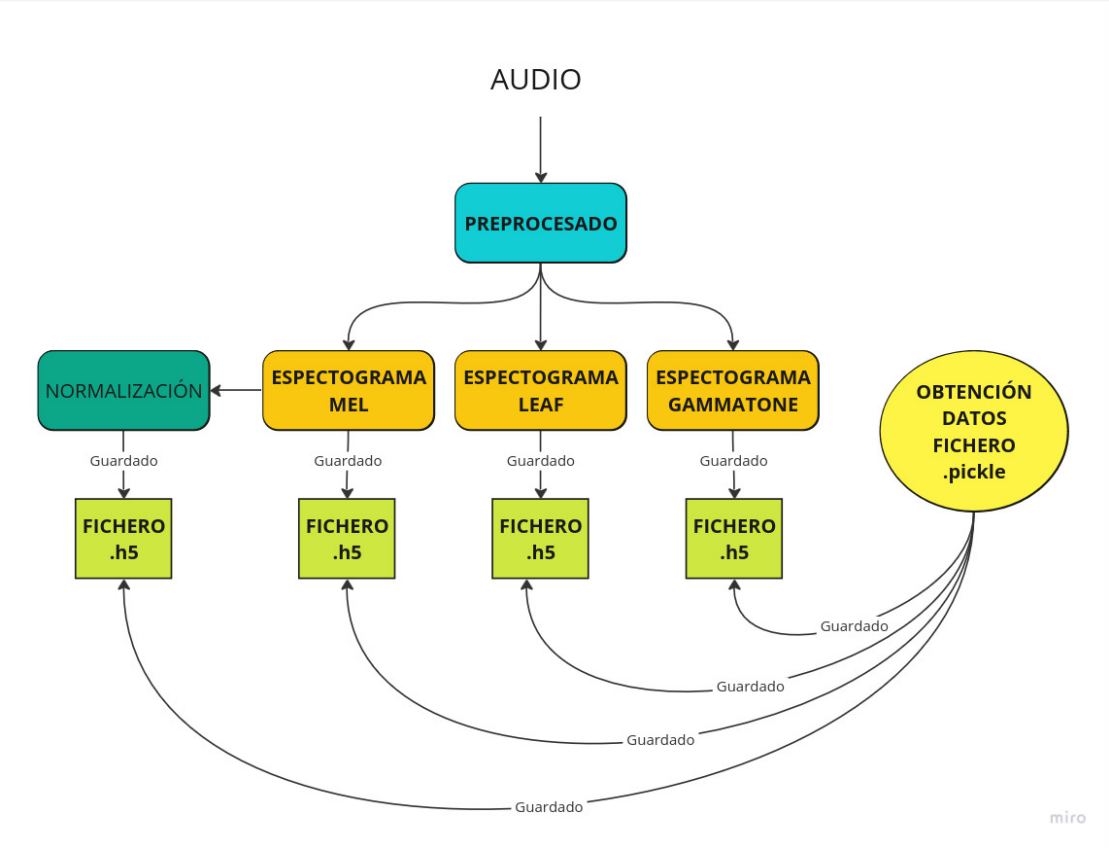


Figura 4.5: Proceso en el que se extraen las características del audio y se guardan en ficheros h5. Estos ficheros también contienen la *label* en formato *one hot encoder*

Keras [5]. Las *callbacks* son funciones que se ejecutan en ciertos puntos durante el entrenamiento del modelo para realizar funciones específicas que se explicarán a continuación:

- *Early Stop*: sirve para parar el entrenamiento si la *loss* no se reduce. En nuestro caso se usó para parar el modelo si la *loss* no descendía en 30 épocas
- Guardado del modelo según la *accuracy*: se guardaba el modelo en cada época si la *accuracy* era mayor respecto la época anterior.
- CSV *Logger*: para guardar las métricas (*accuracy* y *loss*) en un archivo de texto.

[5] keras link

- *ReduceLROnPLateu*: se encargaba de bajar la tasa de aprendizaje (*Learning Rate* en inglés) cada 15 épocas si la *accuracy* no descendía. La tasa de aprendizaje es un hiperparámetro que ajusta los pesos del modelo respecto al gradiente de la función de pérdida.

Por otra parte, el optimizador utilizado fue el Adam, la *loss* fue entropía cruzada categórica (*categorical crossentropy* en inglés) especializada para aplicaciones de clasificación multiclase y la métrica usada fue categorical *accuracy*, también especializada en esta aplicación.

El orden con el que se entrenaron los modelos fue el siguiente. Se empezó por entrenar todos los modelos con los espectrogramas de Mel, Leaf y Gammatone. Una vez realizado estos entrenamientos, se evaluaron los resultados y se escogieron los dos mejores modelos con el espectrograma Mel como entrada (los detalles y las evaluaciones se explicarán detenidamente más adelante en el presente documento). Seguidamente, se experimentó cambiando el tamaño de los filtros en los modelos para reducir el número de parámetros entrenables y hacer disminuir al máximo el tamaño del modelo sin alterar prácticamente el rendimiento. Finalmente, después de ajustar estos modelos se entrenaron con espectrogramas normalizados de manera global y por canal para mejorar la eficiencia.

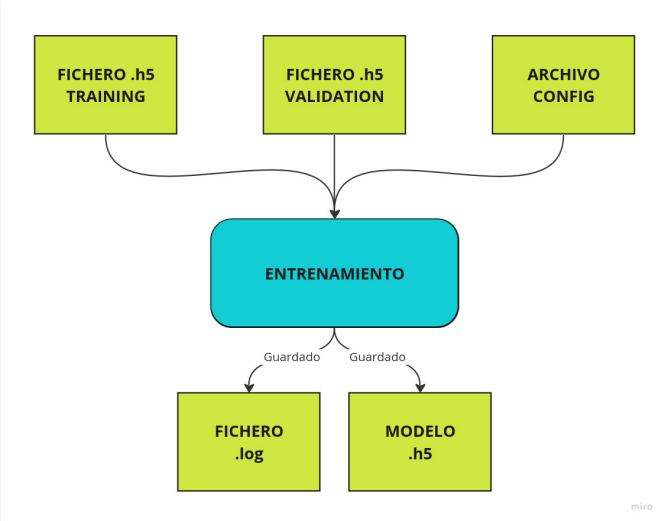


Figura 4.6: Proceso de entrenamiento

4.2.4. Conversión a TFLITE

Como ya se ha comentado anteriormente, la finalidad de este proyecto era tener modelos de TinyML computacionalmente poco costoso) [41], por ello se decidió convertir los modelos a TFLITE [42] (ver Figura 4.7).

En primer lugar, se aplicó una conversión *uint8* para cambiar la precisión de los números en el modelo de coma flotante de 32 bits a enteros sin signo de 8 bits, permitiendo una reducción considerable del tamaño del modelo. Al ser el modelo más pequeño la inferencia será más rápida (siempre que el hardware utilizado esté especializado para operaciones con enteros). Seguidamente, se convirtió el modelo a TFLITE para que pueda ser usado por el compilador de Tensor Flow Lite en dispositivos móviles o dispositivos con un hardware muy limitado.

Para dispositivos móviles, este formato es ideal debido a que permite una eficiencia en los recursos reduciendo la memoria y potencia de cómputo. Además, TFLITE está optimizado para realizar inferencias en tiempo real.

La principal desventaja al usar este formato es que se pierde algo de precisión y rendimiento en el modelo.

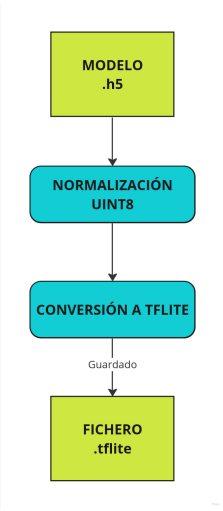


Figura 4.7: Conversión a TFLITE

4.2.5. Inferencia

Para realizar las inferencias (ver Figura 4.8) se usó el dataset de test proporcionado por el DCASE. Se utilizó tanto los modelos h5 como los modelos TFLITE. El resultado de las predicciones se almacenaron en un csv conteniendo la *accuracy* de cada clase y también en una matriz de confusión.

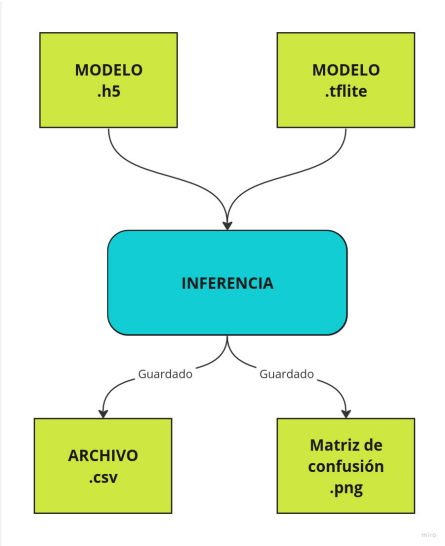


Figura 4.8: Proceso de inferencia

4.3. Sistema de evaluación

Para decidir la calidad y eficiencia del modelo se tuvieron en cuenta tres métricas: *accuracy*, *loss* y matrices de confusión (explicadas previamente). Por otra parte, al enfocar el proyecto como una aplicación para TinyMLse se tuvo mucho en cuenta el número de parámetros del modelo ya que no podía pasar de 128K. Además, una norma impuesta por el concurso de DCASE es que el máximo de MACCs que es el número total de multiplicaciones y acumulaciones realizadas por el modelo, no podía superar los 30 millones. Para obtener este último cálculo se usó la herramienta NeSsi proporcionada por la organización del concurso

4.4. Aplicación práctica (GUI)

Se realizó una pequeña GUI con la finalidad de poder realizar predicciones de una manera visual y más amigable. Para ello, se usó la librería de Python PySimpleGUI [6] ya que aporta diferentes herramientas para la creación de interfaces. El uso se quiso hacer bastante intuitivo, únicamente seleccionas el modelo (ya sea la versión TFLITE o la versión normal h5), el audio y el archivo pickle con las labels (ver Figura 4.9). Se selecciona a predecir y salta una ventana en modo de *pop up* enseñando la predicción y la *accuracy* (ver Figura 4.10).

Figura 4.9: Ventana de inicio

[6] PySimpleGUI link

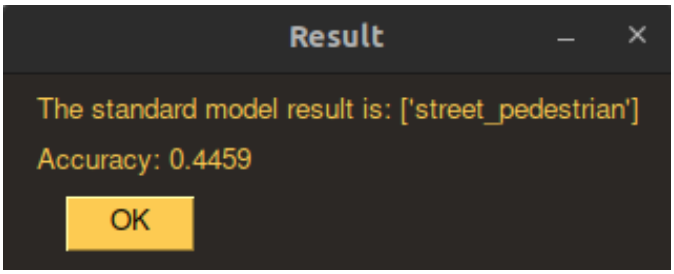


Figura 4.10: Ventana de resultados

Capítulo 5

Resultados

En el presente capítulo se expondrán los resultados obtenidos en las inferencias para poder evaluar cómo funcionan los modelos y saber cuál es su rendimiento.

5.1. *Accuracy*, *log loss* y número de parámetros

En este apartado se mostrará mediante tablas los diferentes resultados obtenidos para cada modelo. Tanto con el dataset del 2022 (año del concurso) y la edición de 2019. Además, se expondrá de manera clara los resultados de cada modelo seleccionando la mejor variante de cada uno de ellos (ver tabla 5.1).

Conv-Sep

Modelo	Accuracy	Loss	Parámetros (Tamaño)	Filtros	Espectrograma
convsep_mel	0.514	1.405	44,770 (174.88 kB)	64-48	mel
convsep_gamma	0.502	1.449	44,770 (174.88 kB)	64-48	gamma
convsep_leaf	0.449	1.548	44,770 (174.88 kB)	64-48	leaf
convsep_mel_40_40	0.467	1.494	7,139 (27.89 kB)	40-40	mel normalizado por canal
convsep_64_64	0.511	1.494	48,714 (190.29 kB)	64-64	mel normalizado por canal
convsep_32_64	0.5	1.452	26,314 (102.79 kB)	32-64	mel normalizado por canal
convsep_48_48	0.505	1.439	28,090 (109.73 kB)	48-48	mel normalizado por canal
convsep_48_48_2019	0.646	0.906	28,090 (109.73 kB)	48-48	mel normalizado por canal
convsep_48_48_spliced_2019	0.606	1.077	28,090 (109.73 kB)	48-48	mel normalizado por canal

Tabla 5.1: Métricas de rendimiento en el modelo basado en Conv-Sep

Como se ha descrito en el apartado de entrenamiento, se probaron diferentes combinaciones del modelo cambiando el número de filtros y el tipo de espectrogramas. Al cambiar el número de filtros el tamaño del modelo incrementa, es por ello que se tuvo especial cuidado a la hora de probar con diferentes valores, ya que no se podía pasar el umbral de 125kB. Por otra parte, con el espectrograma Mel se obtuvieron mejores resultados que con Leaf o Gamma, es por ello que

se decidió normalizar este tipo de espectrograma. Por lo tanto, variando el número de filtros sin pasar el umbral y usando el Mel normalizado por canal se obtuvo que el mejor modelo fue convsep_48_48 con una *accuracy* de 0.505, una *loss* de 1.439 y 28090 parámetros (109.73kB).

Lambda Netowrk

Los resultados de Lambda Network (ver tabla 5.2) fueron bastante mediocres, ya que el *accuracy* apenas llega al 0.428. Además, la cantidad de parámetros son relativamente grandes 26526 (102,56 kB).

Modelo	Accuracy	Loss	Parámetros (Tamaño)	Espectrograma
lambda_mel	0,428	1,645	26256 (102.56 kB)	mel
lambda_mel_gamma	0,412	1.601	26256 (102.56 kB)	gamma
lambda_48_48_2019	0,582	1.727	26256 (102.56 kB)	mel normalizado por canal

Tabla 5.2: Métricas de rendimiento en el modelo basado en Lambda Network

Squeeze Excitation

En este modelo la *accuracy* 0.511 conseguida es relativamente más alta que en los otros. Sin embargo, el número de parámetros 32384 (126,54 kB) hace el modelo demasiado grande para nuestro caso (ver tabla 5.3).

Modelo	Accuracy	Loss	Parámetros (Tamaño)	Filtros	Espectrograma
squeeze_mel	0.511	1.59	32394 (126.54 kB)	32-32	mel
squeeze_gamma	0.506	1.596	32394 (126.54 kB)	32-32	gamma
squeeze_leaf	0.469	1.727	32394 (126.54 kB)	32-32	leaf
squeeze_48_48	0.509	1.447	42953 (192.39 kB)	48-48	mel normalizado por canal
squeeze_48_48_2019	0.621	1.145	42953 (192.39 kB)	48-64	mel normalizado por canal

Tabla 5.3: Métricas de rendimiento en el modelo basado en Squeeze Excitation

Conv Mixer

Al igual que con los anteriores modelos se realizaron diferentes versiones cambiando el número de filtros y el tipo de espectrograma (ver tabla 5.4). Al realizar la primera ronda de entrenamientos con los espectrogramas Mel, Leaf y Gammatone, se comprobó una *accuracy* relativamente buena para un modelo cuyo tamaño es muy pequeño. Es pero ello por lo que se realizaron algunos experimentos más como la utilización de la normalización de Mel global y por canal. Por lo tanto, el modelo que mejor resultado ha dado ha sido el conv_mixer_48_48_fixed con una *accuracy* de 0.472, una *loss* de 1.639 y tan solo 13642 parámetros (53,29kB) siendo el modelo más liviano con mejor *accuracy* de todos.

Modelo	Accuracy	Loss	Parámetros (Tamaño)	Filtros	Espectrograma
conv_mixer_mel	0.466	1.59	10,210 (39.88 kB)	32-32	mel
conv_mixer_gamma	0.468	1.601	10,210 (39.88 kB)	32-32	gamma
conv_mixer_leaf_large	0.42	1.727	149,194 (582.79 kB)	64-48	leaf
conv_mixer_mel_large	0.437	1.941	149,194 (582.79 kB)	64-48	mel normalizado global
conv_mixer_32_64	0.412	1.8	12,490 (48.79 kB)	32-64	mel normalizado global
conv_mixer_48_48	0.441	1.648	14,026 (54.79 kB)	48-48	mel normalizado global
conv_mixer_32_64_fixed	0.467	1.641	12,234 (47.79 kB)	32	mel normalizado global
conv_mixer_64_64	0.477	1.696	22,794 (89.04 kB)	64-64	mel normalizado global
conv_mixer_128_128	0.455	2.194	78,346 (306.04 kB)	128-128	mel normalizado global
conv_mixer_4040_fixed	0.463	1.495	10,090 (39.41 kB)	40-40	mel normalizado por canal
conv_mixer_48_48_fixed	0.472	1.638	13,642 (53.29 kB)	48-48	mel normalizado por canal

Tabla 5.4: Métricas de rendimiento en el modelo basado en Conv Mixer

5.2. Modelos descartados

Los modelos descartados son los siguientes:

- Lambda Network: el *accuracy* obtenido de 0.428 no supera el umbral del modelo *baseline* impuesto por el DCASE de 0.429. Además, la cantidad de parámetros es relativamente alta haciendo que sea un modelo con un rendimiento bastante bajo para la *accuracy* obtenida.
- Squeeze Excitation: a pesar de que conseguimos una *accuracy* relativamente mejor en comparación a los otros modelos, la cantidad de parámetros es muy alta, excediendo la norma del DCASE de no sobrepasar los 125 kB de tamaño.

5.3. Matrices de confusión

Para una mejor visualización de los resultados se han escogido los 8 mejores modelos que fueron los que se presentaron al DCASE y se ha hecho su matriz de confusión.

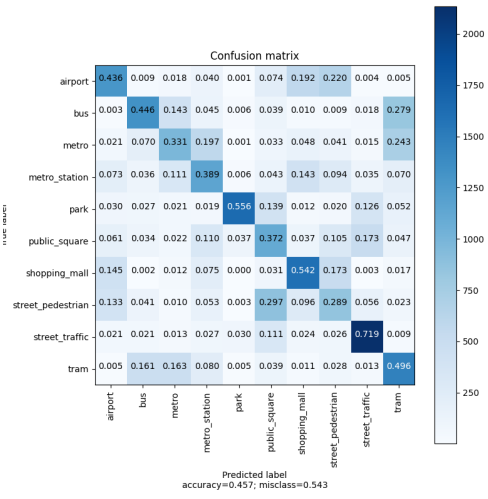


Figura 5.1: Conv Sep usando filtros (40, 40) y espectrogramas de Mel

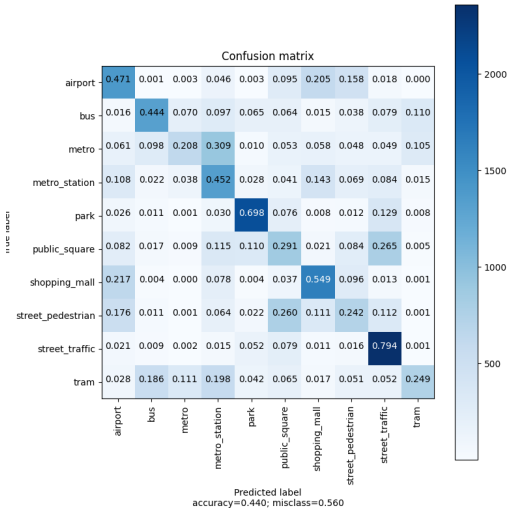


Figura 5.2: Conv Mixer usando filtros (40, 40) y espectrogramas de Mel

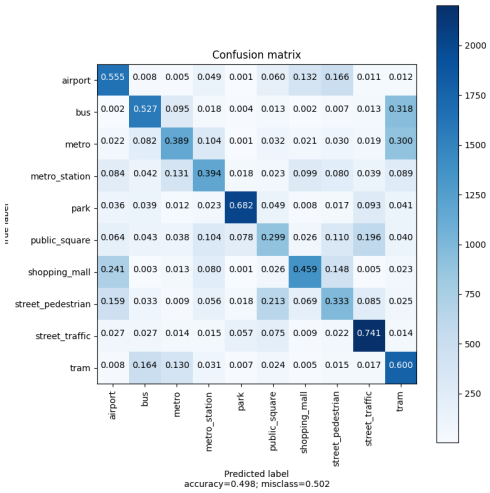


Figura 5.3: Conv Sep usando filtros (48, 48) y espectrogramas de Mel

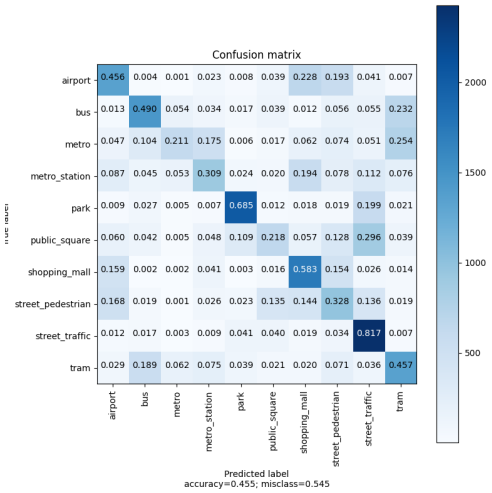


Figura 5.4: Conv Mixer usando filtros (48, 48) y espectrogramas de Mel

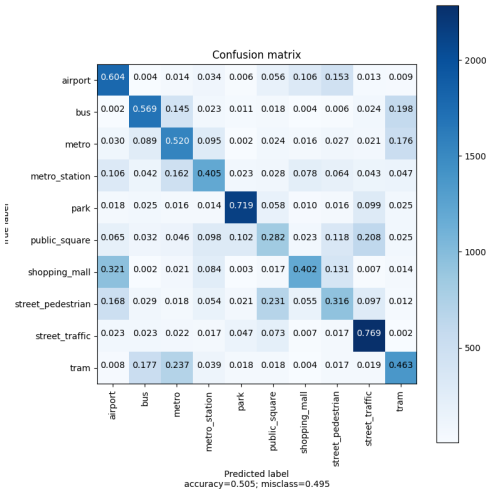


Figura 5.5: Conv Sep usando filtros (32, 64) y espectrogramas de Mel

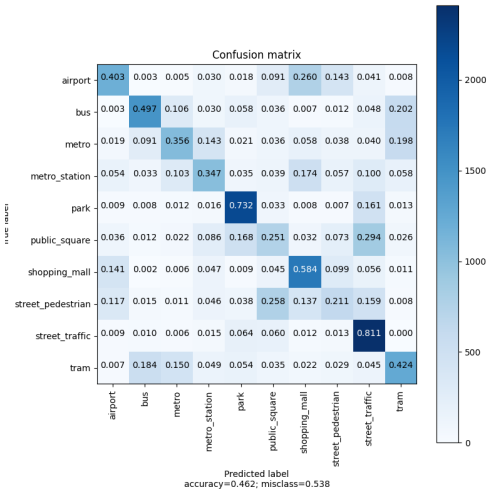


Figura 5.6: Conv Mixer usando filtros (32, 64) y espectrogramas de Mel

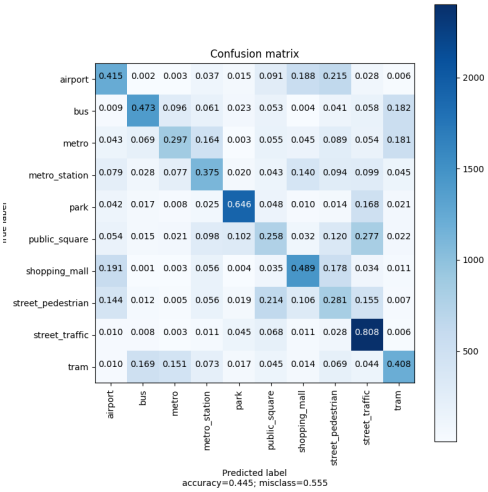


Figura 5.7: Conv Sep usando filtros (64, 64) y espectrogramas de Mel

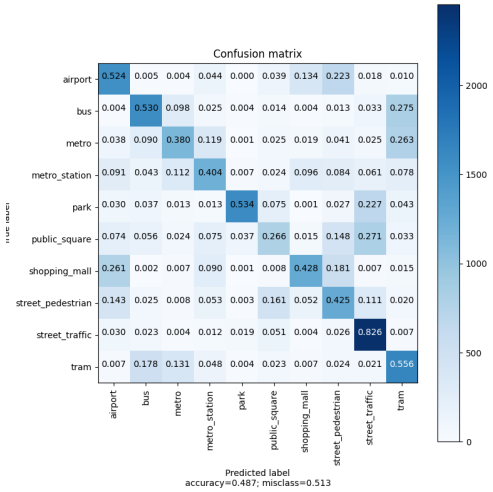


Figura 5.8: Conv Mixer usando filtros (64, 64) y espectrogramas de Mel

Como se puede apreciar, los resultados son bastante similares a los obtenidos durante el entrenamiento. Por otra parte, las categorías con mejor *accuracy* destacan sobre todo *park* y *street traffic*. Sin embargo, los modelos no predicen bien la clase *public square*. En el apartado de conclusiones se explicará más en detalle.

Capítulo 6

Conclusiones

En este capítulo se detalla cómo han sido los resultados, las limitaciones encontradas, el cumplimiento de los objetivos propuestos y las futuras líneas de trabajo.

Análisis integral de los resultados obtenidos

Respecto al rendimiento obtenido en los modelos se puede concluir que han sido relativamente acordes a lo esperado. Es decir, al ser modelos tan pequeños, el nivel de *accuracy* obtenido no es muy alto para lo que el estándar en modelos de clasificación de audio exige. Pero si nos focalizamos únicamente en modelos pequeños enfocados en Tiny ML (que ha sido el propósito de este proyecto) los resultados obtenidos son parejos a otros modelos similares. Para ello compararemos el modelo con otros que han sido presentados en la misma edición del DCASE (ver Figura 6.1).

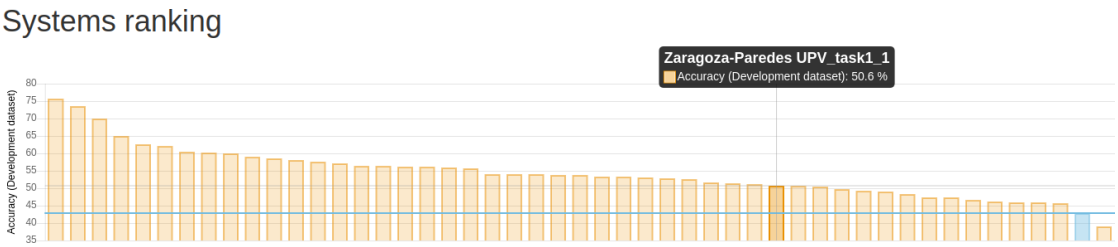


Figura 6.1: Resultado de nuestro modelo en comparación con el resto que se presentaron al concurso. La barra azul era el *baseline* que había que superar.

Como se puede comprobar, los resultados obtenidos son relativamente parecidos al resto de competidores, ya que la mayoría se sitúa en torno al 50-60 % de *accuracy* (exceptuando los 8 primeros).

Limitaciones

Han habido una serie de limitaciones que han interferido en los resultados obtenidos. Las más simbólicas han sido el número de MACs y el de parámetros, ya que limitaba bastante las alteraciones estructurales en los modelos. Esto ha provocado que la mejor forma de mejorar y experimentar con los modelos fuese cambiando el número de filtros de las capas convolucionales. Por otra parte, no se ha podido contar con un entorno de entrenamiento profesional y se ha tenido que usar *google collab pro*, haciendo que el máximo tiempo de entrenamiento por sesión fuese de 8h. Además, al ser un sistema que funciona bajo demanda los días que era alta, el stock de GPUs era muy limitado, haciendo que no se asignase ninguna GPU o tener que esperar en una cola.

Objetivos superados

Respecto a los objetivos propuestos se han superado varios de ellos:

1. Se ha realizado una aplicación completa de IA para el audio por computador

2. Se han adquirido las habilidades necesarias para usar herramientas IA y procesado de señales.

3. Se ha superado el *baseline* del concurso de 42,9 % de *accuracy* con dos de los tres modelos. Por lo que, teniendo en cuenta que es un concurso internacional en el que se presentan expertos de diferentes universidades y empresas privadas es un logro bastante importante.

4. Se ha realizado un *pipeline* de entrenamiento y evaluación completo parecido a los que se pueden encontrar en un entorno más laboral.

5. Se ha realizado una aplicación funcional como es la GUI.

No obstante, no se superó el objetivo de publicar el *paper* en diferentes congresos o revistas. Esto se debe a que aunque los resultados fueron buenos, no se llegó a tener una aplicación lo suficientemente novedosa como para estar en el SoTA actual debido a la gran competencia en el sector.

Futuras líneas de investigación

Algunas posibilidades para continuar con este proyecto podrían ser las siguientes:

1. Implementación en un microcontrolador. Si los modelos se implementasen en un pequeño microcontrolador como una RasperryPy junto con un micrófono podría usarse en el exterior para detectar este tipo de escenas acústicas. Haciendo más maniobrable la obtención de los resultados.

2. Entrenar con otro tipo de audios. En este caso se han usado los datos que proporcionaban el DCASE pero si se tiene una base de datos lo suficientemente grande, estos modelos podrían usarse para clasificar ruidos domésticos (timbres, golpes a la puerta, alarmas...), detección de ruidos anómalos en maquinaria o detección de disparos entre muchos otros.

3. Buscar arquitecturas más ligeras. En nuestro caso hemos usado arquitecturas basadas en redes convolucionales, pero en un futuro es posible que se encuentren algoritmos más eficientes.

Bibliografía

[1] Shujian Deng et al. “Deep learning in digital pathology image analysis: a survey”. En: *Frontiers of Medicine* 14 (jul. de 2020). DOI: 10.1007/s11684-020-0782-9.

[2] Pranav Chib y Pravendra Singh. “Recent Advancements in End-to-End Autonomous Driving using Deep Learning: A Survey”. En: (jul. de 2023).

[3] Warren S McCulloch y Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. En: *The bulletin of mathematical biophysics* 5.4 (1943), págs. 115-133.

[4] R. G. Morris. “D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949”. En: *Brain Research Bulletin* 50.5-6 (nov. de 1999), pág. 437. DOI: 10.1016/s0361-9230(99)00182-3.

[5] Arthur Samuel. “Some Studies in Machine Learning. Using the Game of Checkers. II-Recent Progress”. En: *IBM Journal of Research and Development* 11.6 (1967), págs. 601-617.

[6] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain”. En: *Psychological Review* 65.6 (1958), págs. 386-408.

[7] Sourav Dutta. “An overview on the evolution and adoption of deep learning applications used in the industry”. En: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (2018), e1257.

[8] James Lighthill. “Artificial intelligence: A general survey”. En: *Artificial Intelligence: a paper symposium*. Science Research Council London, 1973, págs. 1-21.

[9] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. En: *Proceedings of the IEEE* 78.10 (1990), págs. 1550-1560.

[10] Corinna Cortes y Vladimir Vapnik. “Support-vector networks”. En: *Machine Learning* 20.3 (1995), págs. 273-297.

[11] S. Rasoul Safavian y David Landgrebe. “A survey of decision tree classifier methodology”. En: *IEEE Transactions on Systems, Man, and Cybernetics* 21.3 (1991), págs. 660-674.

[12] Yann LeCun et al. “Object recognition with gradient-based learning”. En: *Shape, Contour and Grouping in Computer Vision*. Springer, 1999, págs. 319-345.

[13] Wei Wei et al. “A-CRNN: A Domain Adaptation Model for Sound Event Detection”. En: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, págs. 276-280. DOI: 10.1109/ICASSP40776.2020.9054248.

[14] David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams. “Learning representations by back-propagating errors”. En: *Nature* 323 (1986), págs. 533-536. DOI: 10.1038/323533a0.

[15] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. En: *Advances in Neural Information Processing Systems*. Ed. por F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[16] Pedro Domingos. “A Few Useful Things to Know About Machine Learning”. En: *Communications of the ACM* 55.10 (2012), págs. 78-87. DOI: 10.1145/2347736.2347755.

[17] Javier Naranjo-Alcazar et al. “Task 1A DCASE 2021: Acoustic Scene Classification with mismatch-devices using squeeze-excitation technique and low-complexity constraint”. En: (jul. de 2021). DOI: 10.31219/osf.io/gjxa3.

[18] D.P. Kingma y J. Ba. “Adam: A Method for Stochastic Optimization”. En: *arXiv preprint arXiv:1412.6980* (2014). URL: https://arxiv.org/abs/1412.6980.

[19] Tijmen Tieleman y Geoffrey Hinton. *Lecture 6.5 - RMSprop: Divide the Gradient by a Running Average of Its Recent Magnitude*. Inf. téc. 4. COURSERA: Neural Networks for Machine Learning, 2012, págs. 26-31. URL: https://www.coursera.org/learn/neural-networks-deep-learning.

[20] Karol J. Piczak. "Environmental Sound Classification with Convolutional Neural Networks". En: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2015, págs. 1-6. DOI: 10.1109/MLSP.2015.7327852.

[21] Daniele Barchiesi et al. "Acoustic Scene Classification: Classifying Environments from the Sounds They Produce". En: *IEEE Signal Processing Magazine* 32.3 (2015), págs. 16-34. DOI: 10.1109/MSP.2015.2404067.

[22] Yibin Zhang y Jie Zhou. "Audio Segmentation Based on Multi-Scale Audio Classification". En: *Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 4. IEEE, 2004, págs. iv-iv. DOI: 10.1109/ICASSP.2004.1326141.

[23] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, págs. 1251-1258. DOI: 10.1109/CVPR.2017.140.

[24] Geoffrey E. Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition". En: *IEEE Signal Processing Magazine* 29.6 (nov. de 2012), págs. 82-97. DOI: 10.1109/MSP.2012.2205597.

[25] Tara N. Sainath et al. "Deep Convolutional Neural Networks for Large-Scale Speech Tasks". En: *Neural Networks* 64 (2015), págs. 39-48. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.08.005. URL: https://www.sciencedirect.com/science/article/pii/S0893608014002007.

[26] Yuxuan Wang et al. "Tacotron: Towards End-to-End Speech Synthesis". En: *arXiv preprint arXiv:1703.10135* (2017). URL: https://arxiv.org/abs/1703.10135.

[27] Malcolm Stanley. *Auditory Toolbox Version 2*. Inf. téc. Technical Report 1998-010. Interval Research Corporation, 1998.

[28] Neil Zeghidour, Olivier Teboul y de Chaumont Quitry. "LEAF: A Learnable Frontend for Audio Classification". En: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021. URL: https://openreview.net/forum?id=3bW76I9cCnv.

[29] Hao Yen et al. "A Lottery Ticket Hypothesis Framework for Low-Complexity Device-Robust Neural Acoustic Scene Classification". En: *Journal of Acoustic Scene Classification* (2024).

[30] Menglong Wu et al. "Multi-Feature Convergence Network for Acoustic Scene Classification". En: *Proceedings of the 2022 6th International Conference on Electronic Information Technology and Computer Engineering*. EITCE '22. Xiamen, China: Association for Computing Machinery, 2023, págs. 1142-1145. ISBN: 9781450397148. DOI: 10.1145/3573428.3573633. URL: https://doi.org/10.1145/3573428.3573633.

[31] Jonathan Frankle y Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". En: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJl-b3RcF7.

[32] Irene Martin-Morato et al. *Low-complexity acoustic scene classification in DCASE 2022 Challenge*. Jun. de 2022. DOI: 10.48550/arXiv.2206.03835.

[33] Djork-Arné Clevert, Thomas Unterthiner y Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (ELUs)". En: *arXiv preprint arXiv:1511.07289* (2015).

[34] Asher Trockman y J Zico Kolter. "Patches Are All You Need?" En: *Transactions on Machine Learning Research* ().

[35] Dan Hendrycks y Kevin Gimpel. "Gaussian Error Linear Units (GELUs)". En: *arXiv preprint arXiv:1606.08415* (2016).

[36] Irwan Bello. “LambdaNetworks: Modeling long-range Interactions without Attention”. En: *International Conference on Learning Representations*. 2021.

[37] Ashish Vaswani et al. “Attention is All You Need”. En: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, págs. 5998-6008.

[38] Jie Hu, Li Shen y Gang Sun. “Squeeze-and-Excitation Networks”. En: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, págs. 7132-7141. DOI: 10.1109/CVPR.2018.00745.

[39] Abhijit Guha Roy, Nassir Navab y Christian Wachinger. “Concurrent Spatial and Channel ‘Squeeze & Excitation’ in Fully Convolutional Networks”. En: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*. Ed. por Alejandro F. Frangi et al. Cham: Springer International Publishing, 2018, págs. 421-429. ISBN: 978-3-030-00928-1.

[40] Lawrence R. Rabiner y Bernard Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall, 1975.

[41] Pete Warden y Daniel Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O’Reilly Media, 2019.

[42] TensorFlow Authors. *TensorFlow Lite: An Embedded Machine Learning Framework for On-Device Inference*. https://www.tensorflow.org/lite. Accessed: 2024-08-27. 2017.

Parte II

Anexos

Apéndice A

Relación del proyecto respecto ODS

Objetivo de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1 - Fin de la pobreza				X
ODS 2 - Hambre cero				X
ODS 3 - Salud y bienestar				X
ODS 4 - Educación de calidad				X
ODS 5 - Igualdad de género				X
ODS 6 - Agua limpia y saneamiento				X
ODS 7 - Energía asequible y no contaminante				X
ODS 8 - Trabajo decente y crecimiento económico				X
ODS 9 - Industria, innovación e infraestructura	X			
ODS 10 - Reducción de las desigualdades				X
ODS 11 - Ciudades y comunidades sostenibles				X
ODS 12 - Producción y consumo responsables				X
ODS 13 - Acción por el clima				X
ODS 14 - Vida submarina				X
ODS 15 - Vida de ecosistemas terrestres				X
ODS 16 - Paz, justicia e instituciones sólidas				X
ODS 17 - Alianzas para lograr los objetivos				X

Tabla A.1: Tabla de Objetivos de Desarrollo Sostenible

En el año 2015, los diferentes líderes mundiales propusieron un conjunto de objetivos y medidas a nivel global con tal de proteger el planeta, eliminar la pobreza y asegurar la prosperidad como civilización. Todos estos retos forman parte de una nueva agenda que tiene ciertas metas que

deben cumplirse en los próximos 15 años.

En la Tabla A.1 se muestra el impacto que tiene el presente TFG respecto los Objetivos de Desarrollo Sostenible (ODS). A continuación, se detallará más en profundidad que ODS se cumplen en este proyecto.

- **ODS 9 - Industria, innovación e infraestructura**

  Este objetivo se basa en la construcción de infraestructuras resilientes, promover la industrialización inclusiva y sostenible, además de fomentar la innovación. El presente TFG está relacionado con este objetivo debido a que se fomenta la innovación en el campo de los clasificadores acústicos usando IA. Por otra parte, también se relaciona con el impulso a la industria tecnológica ya que incentiva a las industrias a usar métodos de IA para optimizar sus procesos y ser más sostenibles. Finalmente, al participar en el concurso internacional DCASE, se contribuye a la colaboración internacional en investigación que es primordial para el avance en innovación.

Como conclusión, este proyecto ha demostrado implementar técnicas de *Deep Learning* e Inteligencia Artificial avanzadas haciendo que tenga un impacto significativo en el ODS 9 de los objetivos de desarrollo sostenible.