Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Clasificación de Audio para Detección de Intrusiones en Sensores Frontera

Autor(a): Juan Eizaguerri Serrano
Tutor(a): Esteban García Cuesta

Madrid, Julio 2025

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Clasificación de Audio para Detección de Intrusiones en Sensores Frontera

Julio 2025

Autor(a): Juan Eizaguerri Serrano
Tutor(a): Esteban García Cuesta
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Agradecimientos

Quiero dar las gracias a mi tutor Esteban García por brindarme la oportunidad de trabajar en este proyecto. También a Diego Machuca por prestarme apoyo, conocimientos y perspectivas que han sido invaluables en el desarrollo de este trabajo, y a Jon Noble por la supervisión del proyecto. Gracias también a todo el equipo de Verisure, que me ha recibido abiertamente y ha hecho más amenas las horas de trabajo durante los últimos meses.

Por último, no puedo dejar de mencionar a mis amigos, familia y pareja por el apoyo que me han brindado en todo momento no sólo a lo largo de este proyecto, sino durante toda mi etapa universitaria. Este logro también es vuestro.

Resumen

En el panorama actual de los sistemas de alarma inteligentes, los avances en dispositivos IoT han propiciado la integración de la inteligencia artificial para interpretar datos sensoriales en tiempo real. En este contexto, el análisis de audio emerge como una herramienta fundamental y complementaria a los sistemas visuales, permitiendo la detección temprana de sonidos asociados a intrusiones, incluso en condiciones donde otras modalidades sensoriales podrían ser ineficaces. Sin embargo, la implementación de estos sistemas en sensores frontera o dispositivos embebidos presenta desafíos considerables, tales como la variabilidad del entorno acústico, la presencia de ruido, la escasez de datos etiquetados y la necesidad de modelos eficientes, robustos y de bajo consumo que operen en tiempo real con recursos computacionales y energéticos limitados.

Como respuesta a estos retos, el trabajo se propone desarrollar y evaluar un sistema de clasificación de audio optimizado para la detección de intrusiones cumpliendo con las restricciones recién planteadas.

El estudio se ha enfocado en varios componentes clave: la investigación exhaustiva del estado del arte en clasificación de audio, con especial atención a las técnicas adaptadas a sistemas con recursos limitados, el diseño y la implementación de una metodología de evaluación para distintas arquitecturas de aprendizaje profundo, la aplicación de técnicas de optimización de modelos para su despliegue eficiente, y la integración y validación del sistema completo en un microcontrolador real.

En síntesis, este trabajo demuestra que el análisis de audio mediante modelos de *deep learning* optimizados, constituye una solución eficaz y viable para la detección de intrusiones en sensores frontera.

Abstract

In the current landscape of smart alarm systems, advancements in IoT devices have fostered the integration of artificial intelligence for real-time sensory data interpretation. In this context, audio analysis emerges as a fundamental tool, complementary to visual systems, enabling the early detection of sounds associated with intrusions, even in conditions where other sensory modalities might be ineffective. However, implementing these systems in edge sensors or embedded devices presents considerable challenges, such as acoustic environment variability, noise presence, scarcity of labeled data, and the need for efficient, robust, and low-power models that operate in real-time with limited computational and energy resources.

In response to these challenges, this work proposes the development and evaluation of an audio classification system optimized for intrusion detection, adhering to the mentioned constraints.

The study has focused on several key components: exhaustive research into the state of the art in audio classification, with particular attention to techniques adapted for resource-constrained systems, the design and implementation of an evaluation methodology for various deep learning architectures, the application of model optimization techniques for efficient deployment, and the integration and validation of the complete system on a real microcontroller.

In summary, this work demonstrates that audio analysis using optimized deep learning models constitutes an effective and viable solution for intrusion detection in edge sensors.

Tabla de contenidos

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Sistemas de alarma inteligentes

En el contexto actual de creciente preocupación por la seguridad tanto en hogares como negocios y comercios, los sistemas de alarma han evolucionado significativamente gracias al auge de los dispositivos IoT y las tecnologías inteligentes. Los sistemas tradicionales de seguridad, basados en sensores de movimiento o cámaras de vigilancia, presentan limitaciones importantes: alta tasa de falsas alarmas, dependencia de supervisión humana constante, y escasa capacidad para adaptarse a nuevos patrones de comportamiento anómalos. En contraste, en los últimos tiempos han surgido diversos sistemas de alarma inteligentes que integran tecnologías de inteligencia artificial (IA) y aprendizaje automático para interpretar datos sensoriales en tiempo real, identificar eventos sospechosos y reducir la intervención humana en la medida de lo posible.

Los sistemas de alarma desarrollados hoy en día están habitualmente conformados por una red de sensores conectados con unidad central que procesa los eventos recibidos y determina cómo actuar en base a ellos, activando contramedidas y avisando al usuario o a una central receptora de alarmas si es necesario.

Entre los dispositivos y sensores más utilizados se encuentran los detectores de movimiento, cámaras, perimetrales y de interior, detectores de inhibición, detectores de humo y gases, sensores magnéticos de puertas y ventanas, cerraduras inteligentes, sirenas y paneles de comunicación. En conjunto, todos estos dispositivos son capaces de detectar y actuar ante amenazas humanas y ambientales. La integración con tecnologías de comunicación como Wi-Fi, Zigbee [3] o LoRaWAN [4] permite que estos sensores trabajen de forma coordinada mejorando significativamente la eficacia y reduciendo la cantidad de falsas alarmas.

1.2. Análisis de audio en sensores borde

Uno de los avances más prometedores en la línea de los sistemas de alarma inteligentes es la integración de procesamiento de señales de audio. A diferencia de sensores visuales que pueden verse afectados por condiciones de iluminación o campo de visión, el audio proporciona información contextual rica que puede ser capturada incluso en condiciones adversas. En escenarios de seguridad perimetral, los sensores

acústicos permiten detectar y clasificar sonidos indicativos de intrusión, por ejemplo, golpes, pisadas, voces humanas, antes de que el intruso llegue a zonas críticas. Los sistemas de alarma basados en audio no solo representan una fuente complementaria de información, sino que también pueden ser desplegados en lugares donde otros sensores no son viables por coste, mantenimiento o cobertura. En concreto, en sistemas de alarma para hogares y negocios, se pueden situar en puertas y ventanas para detectar sonidos comúnmente asociados con intrusiones como golpes, pasos, voces, etc...

Sin embargo, el desarrollo de estos sistemas implica importantes desafíos técnicos, entre ellos: la variabilidad del entorno acústico, la presencia de ruido de fondo constante, la escasez de datos etiquetados y la necesidad de modelos capaces de generalizar ante nuevos escenarios y ejecutables en sistemas embebidos.

Los sensores frontera deben operar en condiciones altamente dinámicas, expuestos a ruidos meteorológicos y sonidos ambiente característicos de entornos urbanos. Por ello, los modelos de clasificación de audio en este contexto deben ser robustos, eficientes y capaces de operar en tiempo real. El uso de técnicas de *deep learning* ha permitido avances notables en este campo al permitir el desarrollo de modelos con mayor capacidad de generalización ante condiciones inesperadas.

Un aspecto clave en esta línea de investigación es el diseño de conjuntos de datos realistas y representativos del entorno operativo. A menudo, los *datasets* disponibles son limitados o poco adecuados para aplicaciones de seguridad perimetral, lo que obliga a recurrir a estrategias como la recopilación de audio en campo, el uso de técnicas de aumento de datos o el aprendizaje autosupervisado.

Además, se deben considerar criterios de eficiencia energética y computacional ya que los modelos se implementan en dispositivos con recursos limitados. En el contexto de la seguridad el tiempo de reacción ante cualquier posible peligro es crítico por lo que los sistemas de clasificación de audio deben funcionar en tiempo real. Además, los sensores frontera rara vez están conectados a la red eléctrica, sino que son alimentados por pilas o baterías. La comunicación con la unidad central es un proceso energéticamente exigente por lo que el envío de las señales de audio para su procesamiento queda descartado. Esto significa que todo el procesamiento debe realizarse en el propio dispositivo.

1.3. Objetivos

En este contexto, el presente trabajo tiene como objetivo principal el diseño e implementación de un sistema de clasificación de audio orientado a la detección de intrusiones en sensores frontera, utilizando técnicas modernas de procesamiento de señales con modelos de aprendizaje profundo, optimizados para entornos con alta variabilidad acústica y teniendo en cuenta restricciones operativas de los sistemas embebidos.

Así pues, los objetivos específicos planteados para este proyecto son los siguientes:

- Estudiar las tecnologías más utilizadas para el problema de clasificación de audio profundizando en el problema de clasificación de sonidos ambientales, e identificar las características y limitaciones generales en sistemas embebidos y sensores frontera (Capítulo 2).

- Diseñar una metodología para evaluar distintos modelos de clasificación de audio (Capítulo 3) para su posterior evaluación y comparación (Capítulo 4).
- Implementar e integrar un sistema de alarma mediante clasificación de audio en un sistema embebido real, y evaluar su coste energético (Capítulo 5).
- Recoger conclusiones del proyecto y discutir líneas futuras de trabajo (Capítulo 6).

Capítulo 2

Estado del arte

2.1. Clasificación de audio

La clasificación de audio consiste en el análisis y la asignación de fragmentos de audio a distintas categorías establecidas, siendo en esencia un problema de procesamiento y clasificación de señales. Habitualmente la clasificación va precedida de un preprocesado que se aplica a la señal de audio con el objetivo de adaptarla a la entrada del modelo de clasificación o extraer características relevantes (ver Figura 2.1).

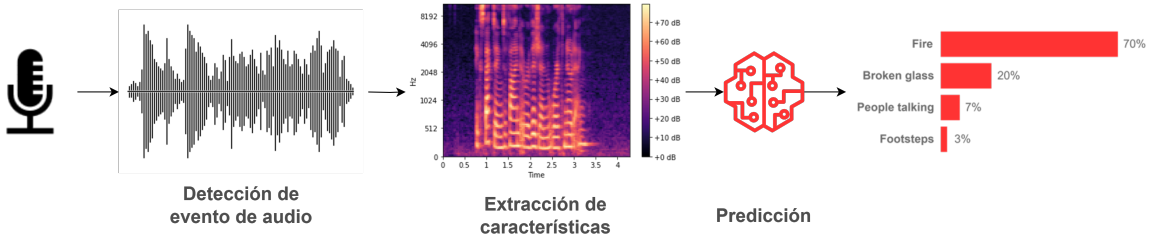


Figura 2.1: Proceso de clasificación de audio

Se trata de un campo amplio con múltiples ramas de estudio, que habitualmente son agrupadas de la siguiente forma [5, 1]:

- **Clasificación de información acústica:** Consiste en la identificación del entorno en el que se ha un audio mediante los sonidos que se escuchan en él. Habitualmente este problema trata de diferenciar ambientes muy diferentes entre sí por lo que son pocas las propuestas que utilizan *deep Learning*.
- **Clasificación de música:** Se basa en el procesamiento de piezas musicales para determinar alguna de sus características. Este campo agrupa diversos problemas como la clasificación de género musical, la segmentación y transcripción de instrumentos.
- **Clasificación del lenguaje natural:** Este campo se basa en el análisis del lenguaje hablado en distintos contextos. Incluye desde problemas sencillos como la detección de idioma hasta otros más complejos como el análisis de sentimiento o la transcripción.

- **Clasificación de sonidos ambientales:** Se enfoca en los sonidos del entorno, ya sean naturales o artificiales, y su clasificación para usos como la monitorización ambiental, análisis de sonidos en espacios urbanos o detección de anomalías.

En cuanto a las tecnologías utilizadas para resolver este problema, Zaman et al. [1] proponen la clasificación que se observa en la Figura 2.2.

Históricamente, la clasificación de audio se ha basado en técnicas tradicionales de procesamiento de señales mediante extracción de características combinadas con algoritmos de aprendizaje clásico como SVMs [6] o *Random Forest* [7]. Estas técnicas, aunque sencillas, han demostrado ser efectivas en una amplia variedad de aplicaciones [1, 7, 8].

En la actualidad, las técnicas de *deep Learning* están permitiendo resolver problemas más complejos. Las Redes Neuronales Convolucionales (CNNs) se emplean ampliamente para capturar características espaciales y frecuenciales de los espectrogramas [9]. Por otro lado, las Redes Neuronales Recurrentes (RNNs) permiten modelar la naturaleza temporal de los sonidos, lo que resulta de gran utilidad en el procesamiento de sonidos con cierta estructura incluyendo la música y el habla [10]. Más recientemente, modelos basados en *Transformers* han demostrado excelentes resultados gracias a su capacidad para procesar relaciones globales entre distintos fragmentos del audio mediante sus mecanismos de atención [11].

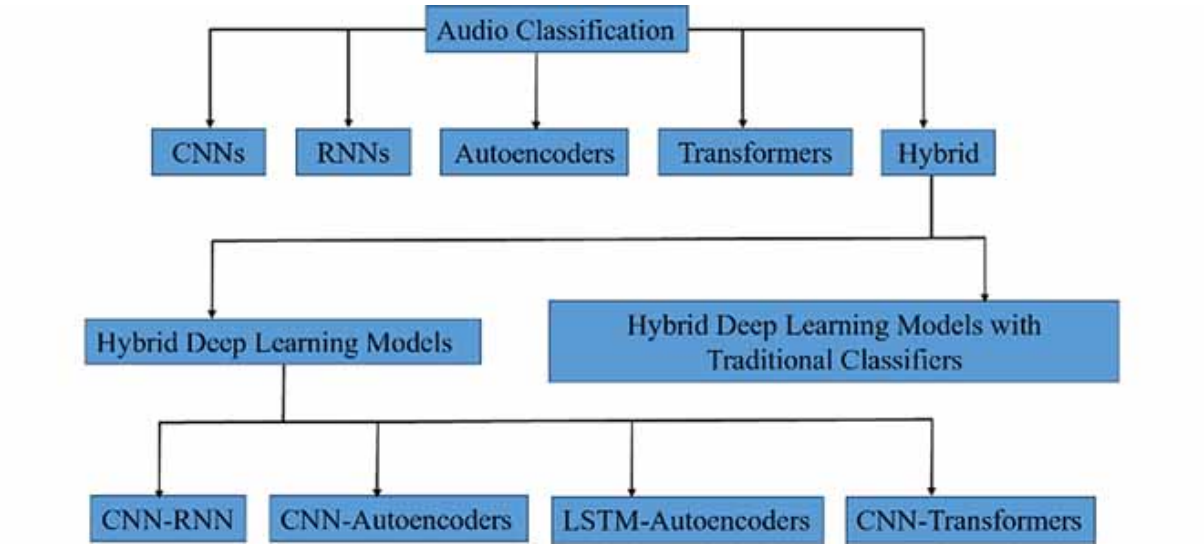


Figura 2.2: Clasificación de arquitecturas para clasificación de audio [1]

2.1.1. Métodos tradicionales

Los enfoques tradicionales para la clasificación de audio se basan en una serie de pasos bien definidos. En primer lugar, se extraen características representativas de la señal de audio [12, 13]. La Tabla 2.1 muestra un resumen de las características más utilizadas en este tipo de problemas.

Las características de audio se agrupan comúnmente en diferentes categorías según el tipo de información que capturan de la señal. Las características temporales describen propiedades relacionadas con la forma de onda en el dominio del tiempo, como la tasa de cruces por cero (*zero-crossing rate*), que puede indicar la presencia

de ruido, y la energía cuadrática media (*RMS energy*), que refleja la intensidad del sonido. Estas se emplean frecuentemente en detección de eventos y segmentación de señales. Las características espectrales se extraen del espectro de frecuencia y capturan información sobre la distribución de energía espectral, como el *spectral centroid* (percepción de brillo del sonido) o el *spectral rolloff* (frecuencia por debajo de la cual se encuentra una fracción de la energía total), siendo especialmente útiles en tareas de clasificación de instrumentos y timbre. Las características cepstrales, como los coeficientes MFCC o LPC, representan la envolvente espectral de manera compacta y son probablemente las características más utilizadas para procesamiento de sonido no relacionado con el habla [14, 15]. Las características armónicas, como el tono y el *harmonic ratio*, describen propiedades tonales de la señal y son relevantes en análisis de melodía, detección de tono y clasificación de música. Finalmente, las características rítmicas, como el tempo y el ritmo, capturan aspectos relacionados con la estructura temporal periódica de la señal y se aplican comúnmente en tareas de análisis de ritmo y sincronización musical.

En la práctica, las características extraídas directamente de la señal de audio suelen ser utilizadas para calcular descriptores estadísticos que resumen su comportamiento a lo largo del tiempo. Por ejemplo, es común obtener medidas como la media, la varianza, los valores máximos y mínimos, o los percentiles de cada característica. Por último, existen características basadas en modelos que provienen de representaciones más abstractas obtenidas mediante modelos estadísticos o de aprendizaje automático.

Categoría	**Nombre**
Temporales	Zero-Crossing Rate
	RMS Energy
Espectrales	Spectral Centroid
	Spectral Rolloff
	Spectral Bandwidth
	Spectral Contrast
Cepstrales	MFCC (Mel-Frequency Cepstral Coefficients)
	Linear Predictive Coding (LPC)
Harnónicas	Harmonic Ratio
	Tono
Rítmicos	Tempo
	Ritmo

Tabla 2.1: Características comunes de audio

En cuanto a los modelos de aprendizaje automático utilizados para la clasificación de audio, históricamente se han adoptado diversos enfoques probabilísticos y no probabilísticos que permiten capturar la estructura y variabilidad de las señales acústicas. Los Modelos de Mezcla Gaussiana (GMM) [16] representan la distribución de las características mediante una combinación ponderada de varias gaussianas y se han utilizado con éxito en reconocimiento de ambientes sonoros y detección de fuentes acústicas [17, 18]. Los Modelos Ocultos de Markov (HMM) [19], por su parte, capturan la dinámica temporal de la señal mediante estados ocultos y han sido ampliamente implementados en reconocimiento de voz y clasificación de eventos acústicos [18]. El método *k-Nearest Neighbors* (k-NN) resulta eficaz en contextos con clases bien

separadas y baja complejidad computacional, aunque carece de modelado explícito de la distribución de los datos [14].

Más recientemente, las Support Vector Machines (SVM) se han consolidado como una alternativa robusta, basada en el principio de margen máximo para construir fronteras de decisión óptimas en el espacio de características. En estudios comparativos, las SVM han superado tanto a k-NN como a métodos basados en árboles de decisión o modelos por prototipos, mostrando mejor capacidad de generalización y mayor precisión en tareas de clasificación y recuperación de audio [20, 21, 22]. En concreto, Guo y Li [23] demostraron que las SVM aplicadas a combinaciones de características perceptuales y cepstrales alcanzan tasas de error significativamente menores que otros sistemas que utilizan k-NN, especialmente cuando se emplean kernels no lineales. Aunque el entrenamiento puede resultar más costoso y requiere selección cuidadosa del kernel y sus parámetros [6, 24], las SVM siguen siendo una referencia de primer orden en clasificación de audio, especialmente en escenarios con alta dimensionalidad y complejidad de datos.

2.1.2. Métodos basados en *deep Learning*

En los últimos años, el *deep learning* ha revolucionado el procesamiento de audio al permitir la extracción automática de características directamente de los datos, sin necesidad de diseñarlas manualmente. Esto ha llevado a una mejora significativa tanto en tareas tradicionales (como reconocimiento de voz o detección de eventos sonoros) como en aplicaciones multimedia y de análisis musical.

Un enfoque ampliamente adoptado traslada técnicas de visión computacional al dominio del audio, aplicando redes convolucionales (CNN) sobre representaciones espectrales como el espectrograma o el espectrograma de Mel [1, 5, 15]. Esta estrategia aprovecha la capacidad de las CNN para detectar patrones locales en frecuencia y tiempo, lo que permite reconocer texturas, transiciones y timbres. Esto elimina la necesidad de seleccionar características específicas para cada problema y abre las puertas al uso modelos mucho más generalistas y reutilizables. Purwins et al. [25] destacan este enfoque como la columna vertebral de muchas arquitecturas de audio clasificadoras.

Sin embargo, no todos los modelos de redes neuronales para clasificación de audio operan sobre espectrogramas. Varios autores han utilizado redes convolucionales 1D en diversos problemas de clasificación de audio [26, 27]. Estas redes toman la onda de audio en bruto sin una conversión o extracción de características previa. Han sido además combinadas con otros métodos como mecanismos de atención [26] o modelos autoregresivos [28], logrando muy buenos resultados en problemas como clasificación de música y sonidos ambientales o reconocimiento del habla. Mientras que en modelos convolucionales 2D es común la compresión, expansión y corte del espectrograma para asegurar un tamaño fijo de entrada, las redes convolucionales 1D necesitan otro mecanismo. Habitualmente se alimenta a la red distintos fragmentos del audio mediante una ventana deslizante con superposición para después agregar los resultados (ver Figura 2.3)

En el campo de la visión por computador, se ha popularizado el uso de arquitecturas predefinidas, bien establecidas y fácilmente reproducibles. Es el caso de las ResNet [29], redes convolucionales. Estas redes contienen bloques residuales, donde la salida

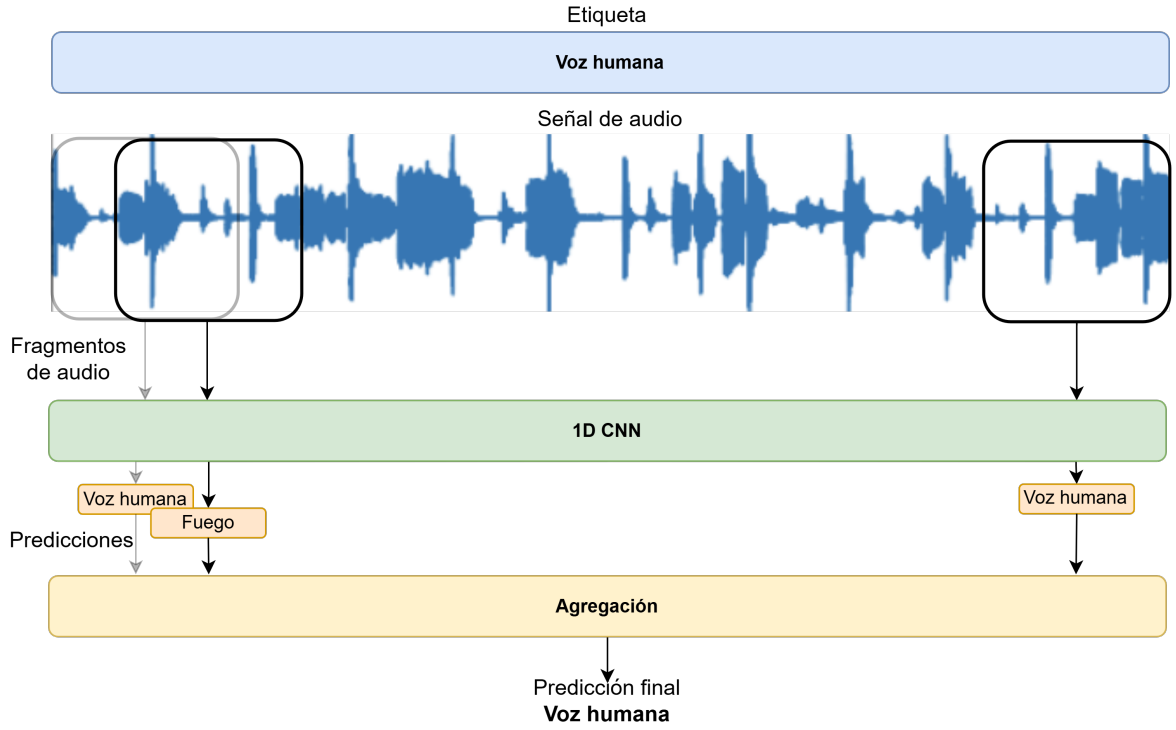


Figura 2.3: Proceso predicción mediante ventana deslizante en CNNs 1D

y del bloque se compone de la suma de la propia entrada x del bloque y el resultado $H(x)$ de las capas subyacentes (ver Ecuación 2.2). De esta forma, la red no aprende la transformación directa que hay que realizar sobre la entrada, sino que aprende una función residual $F(x)$ que expresa qué cambia en la entrada (ver Ecuación 2.1). En la práctica, como se muestra en la Figura 2.4, esto añade conexiones de salto, aliviando el problema de desvanecimiento del gradiente [30] en redes profundas.

$$F(x) = H(x) - x \tag{2.1}$$

$$y = F(x) + x \tag{2.2}$$

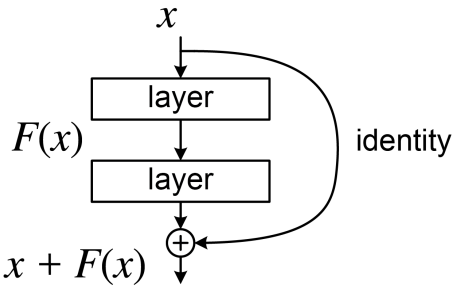


Figura 2.4: Diagrama de bloque residual

Adicionalmente, las Redes Neuronales Recurrentes (RNN) y sus variantes como LSTM [31] y GRU [32] han sido utilizadas para modelar secuencias continuas de audio

en tareas que incluyen audios con dependencias temporales importantes como la clasificación de géneros musicales, demostrado una gran efectividad en este tipo de problemas gracias a su capacidad inherente de procesamiento de datos temporales.

Por otro lado, recientemente los modelos basados en *transformers* han ganado popularidad, conocidos por su capacidad de atención global y modelado de dependencias largas. Han sido aplicados en una amplia variedad de problemas de clasificación de audio con resultados notables, destacando su uso en sistemas de recomendación personalizada [33, 34]. Un trabajo destacable es el Audio Spectrogram Transformer (AST) [35], que aplica un *transformer* puro al espectrograma y ha logrado resultados estado del arte en *benchmarks* como AudioSet o ESC-50. Otras arquitecturas como el Causal Audio Transformer (CAT) [36] combinan atención multiresolución y módulos causales para mejorar la clasificación. Además, existen variantes multicapa como Perceiver [37], que adaptan la atención para datos multimodales, incluyendo audio, lo que permite procesar formas de onda e imágenes mediante una única codificación generalista. Por último, modelos preentrenados como Wav2Vec2 [38, 39] han sido reutilizados con gran éxito en clasificación de audio al añadirles una capa de clasificación. Estos modelos, entrenados en tareas de reconocimiento de voz, pueden extraer representaciones ricas de la señal y posteriormente afinados (*fine-tuning*) para tareas específicas, alcanzando niveles de *accuracy* muy altos con relativamente poco entrenamiento.

2.1.3. Clasificación de sonidos ambientales

La clasificación de sonidos ambientales o *Environmental Sound Classification* (ESC) se centra en la identificación automática de eventos acústicos en entornos naturales o urbanos, tales como lluvia, sirenas, ladridos o tráfico. A diferencia de dominios estructurados como la voz o la música, los sonidos ambientales son altamente variados, presentan gran variabilidad espectral y temporal, y habitualmente están contaminados con ruido de fondo y superposiciones sonoras, lo cual plantea desafíos sustanciales en la extracción de patrones discriminativos [40, 41].

Este problema trae consigo sus propios retos. Uno de ellos es la alta diversidad intraclase, donde eventos acústicamente similares dificultan al modelo distinguir entre categorías cercanas. Además, las grabaciones a menudo contienen, ruido ambiente, del propio dispositivo, tramos silenciosos o semánticamente irrelevantes que diluyen las información significativa de las señales.

Otro desafío importante es la escasez de datos etiquetados para el entrenamiento de los modelos de clasificación. Para lograr sistemas eficaces y robustos mediante técnicas de aprendizaje automático se requiere un conjunto de datos lo más similar posible al que se encontrará el sistema en un contexto de operación real, en este caso, en grabaciones de campo, es decir, fuera de un entorno controlado, con ruido ambiente y micrófonos no profesionales que pueden presentar artefactos en las grabaciones o ruido estático. La creación de conjuntos de datos etiquetados para esta tarea es por tanto un proceso extremadamente costoso, por lo que los *datasets* más utilizados como ESC-50 [42] o UrbanSound8K [43] tienen un tamaño muy limitado, dificultando el entrenamiento de redes profundas, lo que en varias ocasiones ha promovido el uso de técnicas como *data augmenting* o *transfer learning*.

Pese a las dificultades del problema, gran cantidad de autores han logrado desa-

rrollar sistemas de ESC con muy buenos resultados utilizando distintas técnicas, destacando aquellas que utilizan CNNs [44, 45, 46, 47]

2.2. Sistemas embebidos y sensores frontera

En los últimos años, el uso de dispositivos IoT ha experimentado un aumento sin precedentes, con una estimación de más de 50 billones de dispositivos conectados en 2025 [48]. Junto a este fenómeno, aparece el concepto de la computación de borde o *edge computing*, respaldado por un conjunto de tecnologías que permiten la recogida, almacenamiento y procesamiento de datos en el dispositivo. Se trata de una arquitectura de microservicios integrados en los propios dispositivos de borde. La implantación de estas tecnologías ha sido un éxito en los últimos años, dando lugar a dispositivos rápidos, robustos e inteligentes.

Por otro lado, el campo de la inteligencia artificial ha vivido un auge sin precedentes propulsado por los avances en potencia computacional y el impulso de las redes neuronales. La aplicación de *transformers* para el procesamiento del lenguaje natural y otras tareas generativas ha extendido las aplicaciones de la inteligencia artificial a multitud de nuevos campos.

Generalmente, este gran crecimiento ha ido de la mano de modelos cada vez más grandes y computacionalmente demandantes [49]. Sin embargo, múltiples estudios recientes han ido en dirección opuesta a esta tendencia con el objetivo de obtener modelos pequeños y eficientes, permitiendo dejar de depender de la computación en la nube para utilizar modelos potentes de forma local en dispositivos móviles y embebidos. Es así como nace el concepto de *Tiny ML* [50, 51] o *Edge AI* [52].

En los últimos años, se ha trabajado en la integración de inteligencia artificial en dispositivos embebidos con distintas aplicaciones como la salud [53], la videovigilancia [54] o los vehículos autónomos [55].

En general, estos dispositivos suelen contar con menos de 100kB de RAM y 1MB de memoria flash [56], son mayoritariamente programados en lenguaje C por cuestiones de eficiencia [57]. En el contexto de la seguridad, los dispositivos de borde incorporan sensores cuya información es procesada por el microcontrolador para distinguir eventos de riesgo y habitualmente son alimentados por pilas para facilitar su instalación.

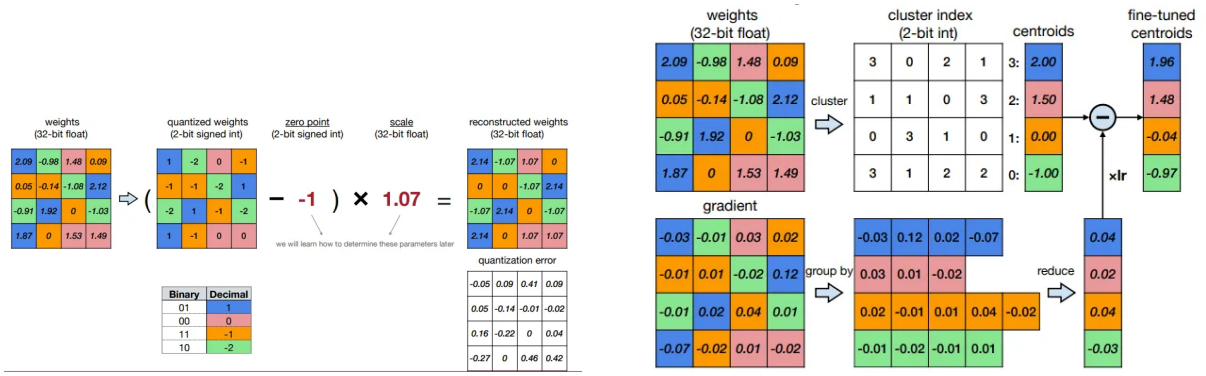
2.3. Métodos de optimización de redes neuronales

Debido a las características habituales del hardware en el que habitualmente se ejecutan este tipo de modelos, el coste en tiempo y potencia computacional, así como el tamaño, tanto en memoria temporal como persistente, se convierten en factores críticos en su desarrollo. Por ello, diversas técnicas de optimización se han convertido en un pilar fundamental de esta área.

2.3.1. Cuantización

La cuantización de modelos es una estrategia esencial para adaptar redes neuronales a entornos con recursos limitados, ya que permite reducir notablemente el tamaño

del modelo y el coste computacional. En la forma más básica, la cuantización lineal uniforme transforma los pesos y activaciones de punto flotante, habitualmente representados en 32 bits, a valores discretos (por ejemplo, enteros de 8 bits) mediante pasos equidistantes, lo que facilita una inferencia más eficiente (ver Figura 2.5a) . Sin embargo, cuando los parámetros presentan distribuciones no uniformes, este enfoque puede inducir pérdidas de precisión apreciables. Una de las técnicas que aparecen para solventar este problema es la cuantización por k-means, también conocida como *vector quantization* [58]. Consiste en la agrupación de los valores de los pesos mediante *clustering*, asignándoles el centroide más cercano (ver Figura 2.5b), adaptándose mejor a distribuciones multimodales y ofreciendo tasas de compresión superiores con pérdida mínima de precisión. Este método es efectivo en la compresión del modelo aunque todas las operaciones se siguen realizando en punto flotante, por lo que no ofrece mejoras en cuanto a coste computacional.



(a) Esquema de cuantización lineal

(b) Esquema de cuantización K-means

Figura 2.5: Esquema del proceso de cuantización: (a) lineal, (b) K-means [2]

Como alternativa a estos métodos de cuantización post-entrenamiento, y con el objetivo de recuperar la exactitud perdida, surge el *quantization-aware training (QAT)* [59], una técnica que simula la cuantización durante el entrenamiento, ajustando los pesos y parámetros (como clipping o escalas) para mejorar la robustez del modelo. Es importante destacar que la QAT requiere acceso al conjunto de datos original, incluyendo muestras y etiquetas, para poder simular y compensar eficazmente los efectos de la cuantización durante el entrenamiento. En la práctica, ambos métodos de cuantización son muy utilizados y su elección depende primariamente de las necesidades de tamaño, velocidad y exactitud.

En el campo de los grandes modelos de lenguaje, aparecen otras técnicas de compresión como el Knowledge Distillation (KD), consistente en el entrenamiento de un modelo más pequeño (student) para imitar el comportamiento de uno mayor (teacher), conservando una buena precisión mientras se reduce tamaño y costo de inferencia [60, 61].

2.3.2. Pruning y clustering

El *pruning* o poda de modelos consiste en ajustar gradualmente a cero los pesos de una red neuronal dando lugar a un modelo lo más disperso posible. Esto tiene beneficios en el tiempo de inferencia al poder saltar las operaciones con ceros, sin embargo, su principal atractivo se encuentra en la mejora de la compresión del modelo. Los

modelos dispersos resultantes contienen una gran cantidad de ceros, resultando en secuencias frecuentes de ceros. Los algoritmos de compresión habituales como zip o gzip son más efectivos cuando hay mucha repetición de patrones en los datos, por lo que esta técnica mejora el proceso de compresión. Por ejemplo, en [62] se emplea esta técnica para obtener una razón de compresión de 9x sobre AlexNet [63].

El *clustering* o agrupamiento de pesos en redes neuronales tiene un propósito similar. Este algoritmo agrupa los pesos de la red en N clusters y les asigna el valor del centroide de cada uno, reduciendo a N la cantidad de valores únicos de pesos. Esto también aumenta considerablemente la cantidad de patrones repetidos permitiendo una mejor compresión.

Capítulo 3

Diseño de experimentación

3.1. Conjuntos de datos

La clasificación de sonidos ambientales es un problema que requiere de procesos de entrenamiento supervisados. Es necesario por tanto un conjunto de datos etiquetado para el entrenamiento y la evaluación de los modelos que se van a desarrollar.

Como se ha explicado en la Sección 1.2, los modelos deben presentar un alto nivel de robustez ante distintos tipos de ruido para lograr un buen rendimiento. Por lo tanto, es preferible utilizar conjuntos de datos extraídos de grabaciones de campo. ESC-50 [42] y UrbanSound8K [43] son dos conjuntos de datos populares de este tipo. Ambos utilizan audios recogidos por el proyecto Freesound [64] etiquetados como grabaciones de campo. ESC-50 se compone de 2000 audios de 5 segundos distribuidos equitativamente en 50 clases. Esto resulta en tan solo 40 muestras por clase, que se considera insuficiente para entrenar correctamente varios de los modelos que se quieren probar. Su derivado ESC-10 presenta el mismo problema, ya que se trata de un subconjunto de 10 clases seleccionadas. UrbanSoun8K soluciona este problema aunque plantea sus propios retos: Se incluyen audios de distintas duraciones, aunque siempre menor a 4s, además, el conjunto presenta dos clases considerablemente desbalanceadas. Otros conjuntos que no se limitan a grabaciones de campo contienen un número de muestras considerablemente mayor, véanse FSD50K [65] y Google Audioset [66]. segundo tiene un tamaño mucho mayor, hay que tener en cuenta que ha sido extraído de vídeos de YouTube y la calidad general del etiquetado es inferior. La Tabla 3.1 recoge la información general de cada uno de estos conjuntos.

Se plantea el uso uso de dos conjuntos distintos para el entrenamiento y evaluación

Dataset	**N. Audios**	**N. Clases**	**Duración de audios (s)**	**Balanceado (S/N)**
ESC-50	2000	50	5	S
FSD50K	51197	200	NS	N
Google AudioSet	>2M	527	NS	N
UrbanSound8K	8732	10	<4 (no fijado)	S*

* Balanceado a excepción de dos clases.

Tabla 3.1: Resumen de características de los datasets

de los modelos desarrollados. En primer lugar, se va a utilizar UrbanSound8K para comparar los modelos desarrollados con los de otros trabajos en el problema general de ESC. Por otro lado se quiere comprobar el funcionamiento del modelo para el caso de uso específico estudiado en este trabajo, por lo que se va a utilizar un segundo conjunto para evaluar el modelo seleccionado ante sonidos relacionados con intrusiones. Dado que no se tiene constancia de un conjunto de datos de estas características se requiere la creación de uno propio.

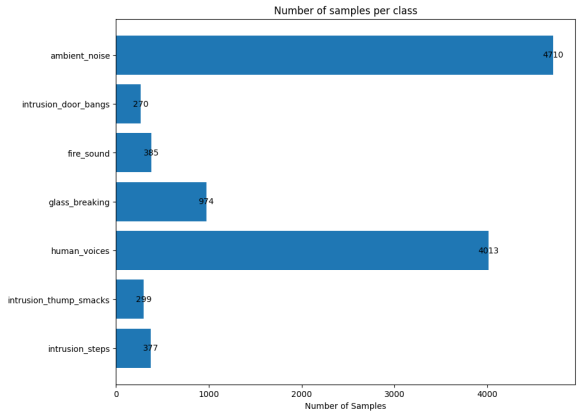
Se ha decidido la siguiente clasificación para el conjunto de datos desarrollado: *Rotura de cristal, Ruido ambiente, Pasos, Golpes, Golpes a puertas, Voces humanas, Fuego*. Se descarta la posibilidad de grabar los sonidos para el conjunto de datos por limitación de recursos. Una solución a este problema que se ha visto en estudios recientes es la generación de sonidos sintéticos mediante IA generativa, sin embargo, esto habitualmente resulta en modelos que no son capaces de generalizar o no son robustos ante muestras reales. En su lugar, se crea un subconjunto de FSD50K buscando equivalencias y extrayendo las clases de interés. Del subconjunto extraído, se eliminan las muestras de más de 30 segundos debido al tamaño que supone el almacenamiento de secuencias tan largas, en ocasiones de varios minutos de duración.

El conjunto de datos resultante cuenta con aproximadamente 11k muestras, aunque con un desequilibrio muy grande entre sus clases, como se puede ver en la Figura 3.1a. A la hora de su utilización se utilizará por tanto una versión balanceada a la clase más pequeña, por lo que se cuenta con un total 270 muestras por clase. La duración de los audios del conjunto de datos y muy variada. Aproximadamente el 50 % de las muestras duran menos de 5 segundos, llegando hasta los 30 segundos de duración. Además, la distribución de la duración de las muestras varía mucho entre las distintas clases del *dataset* (ver Figuras 3.1b y 3.1c).

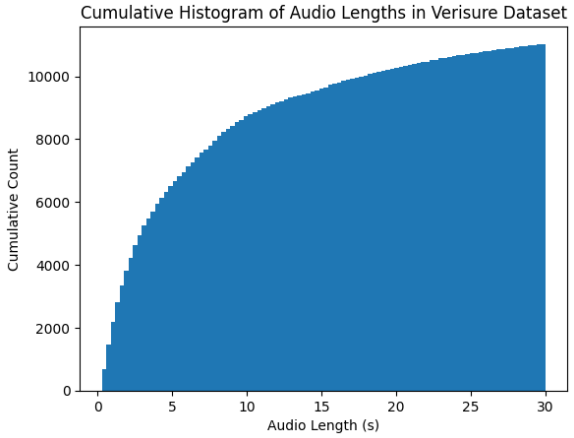
A la hora de la utilización del conjunto de datos recopilado, ha de tenerse en cuenta que el dataset *FSD50K* que se ha utilizado para realizar el subconjunto contiene audios de todo tipo, no sólo grabaciones de campo. Un análisis del conjunto resultante demuestra que efectivamente, la calidad del etiquetado es generalmente menor a la que se encuentra en *datasets* de ESC como UrbanSound8K o ESC-50. En especial, se observa que la clase *Ruido Ambiente* es demasiado generalista, a menudo conteniendo sonidos que se podrían incluir dentro de otras de las categorías definidas. Será importante por tanto estudiar el efecto de la exclusión de esta categoría a la hora de evaluar modelos de clasificación.

3.2. Arquitecturas evaluadas

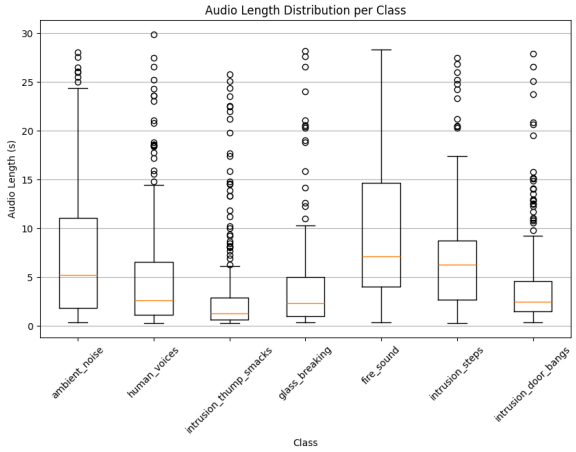
Este apartado describe las arquitecturas implementadas y evaluadas con el objetivo de seleccionar las tecnologías más adecuadas. Generalmente se han implementado modelos basados en el estado del arte actual diseñados específicamente para sistemas con recursos limitados. Estos modelos han sido adaptados para ajustarse al tamaño del conjunto de datos de evaluación, sin perder de vista las restricciones de hardware. Además, algunas implementaciones se realizan de forma consciente de que no cumplirán con las limitaciones establecidas. Sin embargo, se llevan a cabo de igual manera con propósitos específicos, como analizar la diferencia de rendimiento entre un modelo adaptado a las restricciones y otro de mayor tamaño. Esto permite una comprensión más profunda de las posibilidades y limitaciones de cada arquitectura.



(a) Distribución de clases



(b) Duración de los audios



(c) Duración de los audios por clase

Figura 3.1: Resumen visual del dataset elaborado

3.2.1. SVM

Entre los métodos tradicionales estudiados, se realiza una evaluación de un modelo de SVM al demostrar los mejores resultados dentro de esta categoría en múltiples estudios.

Como se ha visto en la Sección 2.1.1, distintas características extraídas de la onda de audio son frecuentemente utilizadas como entrada del modelo. Por ello, se han entrenado modelos con varias de ellas para comparar su rendimiento. Para cada una de ellas se entrena el modelo utilizando kernel lineal, RBF, polinomial y sigmoide.

3.2.2. Arquitecturas convolucionales

Las Redes Neuronales Convolucionales (CNN) incluyen capas convolucionales, reduciendo considerablemente la cantidad de parámetros respecto a redes densas y permitiendo la extracción de automática de características. Su uso principal se encuentra en tareas de visión por computador para procesamiento de imágenes, sin embargo, es sencillo transladar sus aplicaciones al campo del audio a través de la representación de estos sonidos en forma de espectrograma.

La Figura 3.2 detalla el proceso seguido para la generación de los espectrogramas, el cual se ha seguido para transformar la entrada de los datos de todas las redes convolucionales 2D evaluadas. Partiendo de la onda en bruto del audio, que muestra la amplitud en función del tiempo, se aplica la transformada rápida de Fourier [67] para representarlo en el dominio de frecuencias. Este proceso se puede repetir para fragmentos pequeños de audio en forma de ventana deslizante con solapamiento, dando lugar a un espectrograma, que refleja cómo la amplitud varía con el tiempo para distintas frecuencias. Este espectrograma se pasa además a escala logarítmica para representar la intensidad en decibelios. Por último, se realiza una transformación a escala de Mel [68], que representa el sonido de forma más fiel a como es escuchado por los humanos, de forma que distancias iguales en la magnitud del tono resultan en cambios iguales para el oyente. En las pruebas realizadas, el espectrograma se realiza con un tamaño de ventana de 2048 muestras, que con una frecuencia de muestreo de 16kHz se traduce en una ventana de 128ms. El tamaño de salto utilizado es de 512 muestras o 32ms.

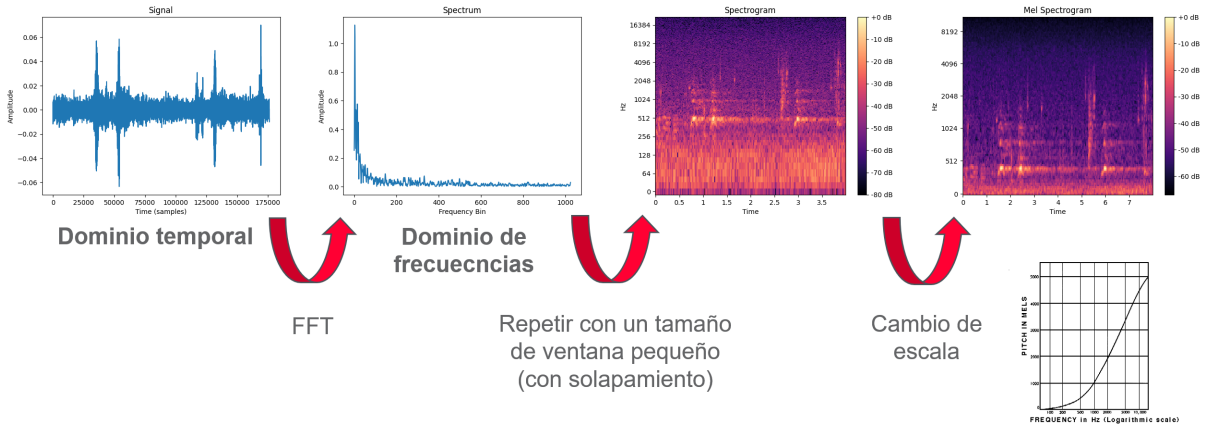


Figura 3.2: Proceso de generación del espectrograma de Mel

Todos los modelos se entrenan con optimizador Adam [69] y una tasa de aprendizaje de 0.001, utilizando *categorical cross-entropy* como medida de error.

Se han implementado y evaluado 5 arquitecturas distintas de CNNs agrupadas en tres categorías principales: Dos CNN básicas, una CNN 1D que utiliza la onda de audio en bruto, y dos ResNet. Cada una de estas arquitecturas se encuentra detallada a continuación.

a. *CNN básicas*

En primer lugar, se evalúan dos redes convolucionales básicas cuya arquitectura se puede ver en la Figura 3.3. Estas redes incluyen tres bloques de convolución compuestos por una capa convolucional 2D con kernel 3x3 y activación ReLU seguida de una de *max pooling* con tamaño de *pooling* 2x2 para reducir la dimensionalidad. Estos bloques convolucionales van seguidos de una capa densa con activación ReLU, y por último, la capa de salida, con el número de neuronas correspondiente al de las categorías distintas del conjunto de datos a las que se les aplica una función *softmax* de manera que la salida de cada una de estas neuronas dada una entrada refleja la probabilidad de pertenencia a la categoría.

Aunque arquitecturalmente similares, las dos redes evaluadas difieren en gran medida en su complejidad a través del número de parámetros. La más sencilla, a la que llamaremos CNN-Mel (small) utiliza 16, 32 y 64 filtros en las capas convolucionales y 64 neuronas en la capa densa. Mientras tanto, la de mayor tamaño (CNN-Mel (big)) utiliza 24, 48 y 96 filtros en su lugar, junto con 100 neuronas en la capa densa.

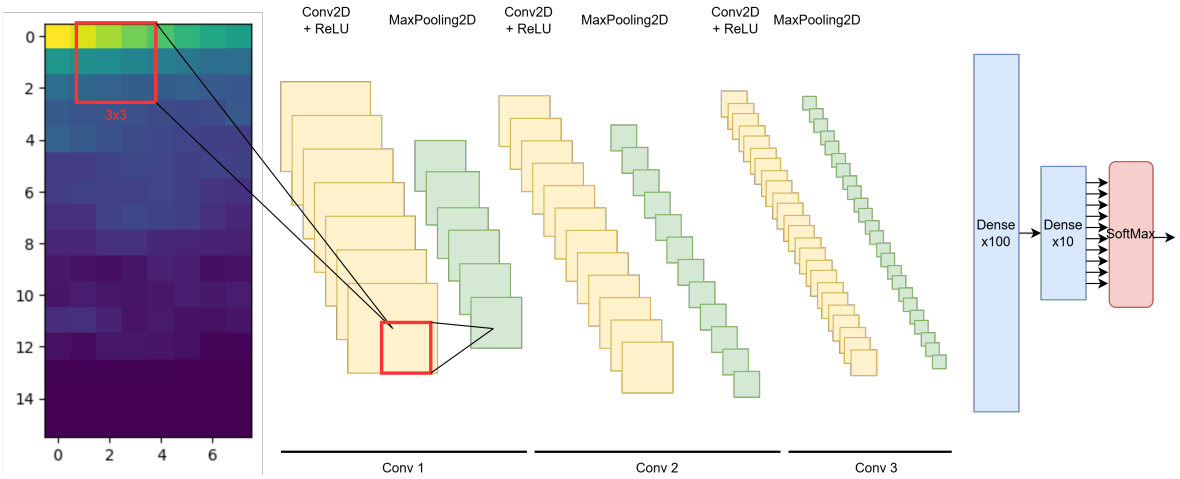


Figura 3.3: Arquitectura de la CNN

b. *CNN 1D*

A raíz de los buenos resultados obtenidos mediante redes convolucionales 1D en [26], se plantea su uso en este proyecto. Como se ha cubierto en la Sección 2.1.2, su uso en clasificación de audio permite la entrada de la onda de audio en crudo en lugar de en forma de espectrograma.

La arquitectura de la red está basada en la diseñada por Abdoli et al. La red cuenta con 5 capas convolucionales 1D, con 16, 32, 64, 128 y 256 filtros seguidas de activaciones ReLU y de capas de *max pooling 1D* en los tres primeros bloques. Además, se incorporan capas de regularización como *batch normalization* y *dropout*. La red termina en dos capas densas de 128 y 64 neuronas, otra capa densa de salida y una función de activación *softmax*.

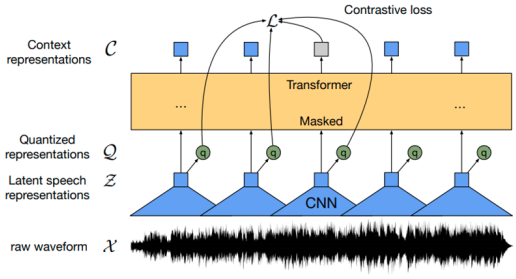
La red realiza inferencias sobre fragmentos de un segundo del audio, se aplica el método de ventana deslizante explicado en la Sección 2.1.2 para hacer predicciones sobre audios de mayor longitud.

c. *ResNet*

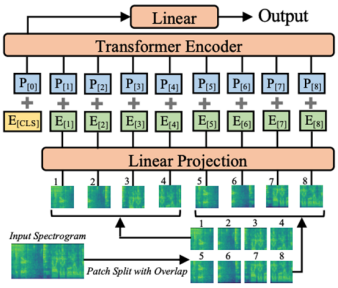
Por último, es interesante analizar el rendimiento de las arquitecturas ResNet para el problema que se plantea. En lo que respecta a este tipo de redes, es común tanto el uso de arquitecturas predefinidas para su entrenamiento desde 0, como de modelos preentrenados para tareas específicas. Se quiere explorar ambos métodos, por lo que se implementa y evalúa una ResNet-56 entrenada desde cero y un modelo preentrenado de ResNet-50 para reconocimiento de imagen al que se le aplica un *fine-tuning* con el conjunto de datos de entrenamiento.

3.2.3. Arquitecturas basadas en *transformers*

De entre los modelos basados en *transformers* analizados en la Sección 2.1.2, se evalúa el rendimiento de dos modelos preentrenados de Wav2Vec y Audio Spectrogram Transformer (AST). En concreto, los modelos *facebook/wav2vec2-xls-r-300m* y *MIT/ast-finetuned-audioset-10-10-0.4593*, respectivamente, disponibles en sus correspondientes repositorios de Huggingface Hub [70].



(a) Wav2Vec [39]



(b) Audio Spectrogram Transformers (AST) [35]

Figura 3.4: Esquema de arquitecturas de modelos de *transformers* para audio

Aunque como se puede ver en las Figuras 3.4a y 3.4b ambas arquitecturas son similares, tienen las suficientes diferencias para que valga la pena evaluar ambas. Wav2Vec opera sobre la onda de audio en bruto, se entrena mediante tareas auto-supervisadas y presenta un modelo híbrido que incluye capas convolucionales. Mientras tanto, AST utiliza una entrada en forma de espectrograma, tiene un entrenamiento supervisado y es un transformer puro, sin incluir capas convolucionales como Wav2Vec.

Ambos modelos son ajustados mediante un proceso de *fine-tuning* para la tarea específica de clasificación de sonidos ambientales con el conjunto de datos seleccionado. Cabe destacar además que los dos modelos, han tenido que ser modificados en gran medida para disminuir su tamaño y coste de entrenamiento debido a limitaciones en cuanto al hardware disponible para el entrenamiento y el gran tamaño por defecto de estos modelos (87M y 317M de parámetros).

3.3. Metodología de evaluación

Cada una de las arquitecturas propuestas se entrena con el conjunto de datos UrbanSound8K para evaluar su rendimiento. Se utiliza una partición de los datos del 20 % como conjunto de test, y el 80 % restante como conjunto de entrenamiento.

Las redes neuronales son entrenadas durante las épocas necesarias para que se estabilice el *accuracy* en el conjunto de validación. Además, se aplican las extracciones de características necesarias para adaptar los audios a la entrada de cada modelo.

Se está tratando con un problema de clasificación multiclase, por lo que para medir la calidad de las predicciones se utilizan métricas de confusión. como se puede ver en la Figura 3.5, se calcula el número de *True Positives* (TP), *True Negatives* (TN), *False Positives* (FP) y *False Negatives* (FN), que permite el cálculo de diversas métricas. Al tratarse de un conjunto de datos casi completamente balanceado, se considera la

medida de *accuracy* (ver Ecuación 3.3) suficiente para la comparación de los modelos, y se reservan las métricas de precisión (ver Ecuación 3.1) y *recall* (ver Ecuación 3.2) para un análisis más detallado del rendimiento del modelo seleccionado con el conjunto de datos desarrollado. Adicionalmente, se hará uso de las gráficas y figuras que sean necesarias para una mejor visualización de los resultados.

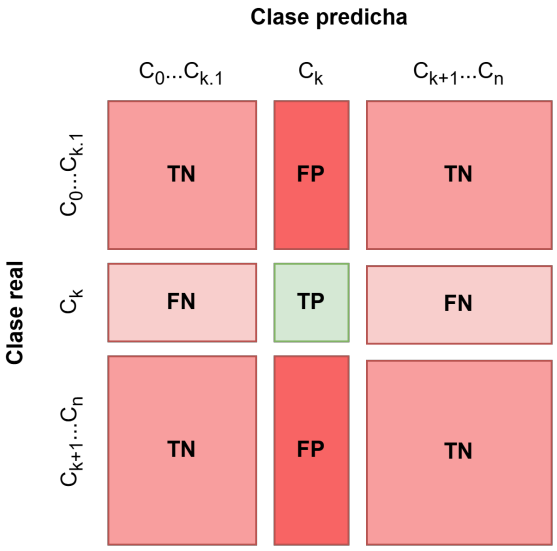


Figura 3.5: Matriz de confusión multiclase

$$\text{Precisión (P)} = \frac{TP}{TP + FP} \tag{3.1}$$

$$\text{Recall (R)} = \frac{TP}{TP + FN} \tag{3.2}$$

$$\text{Accuracy (ACC)} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Por otro lado, en el problema que se está tratando en este proyecto es crítico reducir lo máximo posible el tamaño del modelo, tanto en memoria persistente *flash*, como temporal (RAM). Se quieren tener en cuenta estos factores para la comparación de tecnologías, por lo que además del *accuracy*, para cada modelo evaluado se registran también estos valores.

En primer lugar, el número de parámetros de los modelos ofrece una visión general de su complejidad, más concretamente con su tamaño en memoria persistente. Los parámetros de las redes neuronales se almacenan habitualmente en coma flotante de 32 bits, por lo que el tamaño es de aproximadamente $\#params * 4$ kiloBytes.

Los recursos de RAM necesarios para la ejecución del modelo sin embargo dependen de la arquitectura de la red. Para medir este valor, se genera un resumen del tamaño de buffer para cada una de las capas de la red, siendo el valor máximo entre ellos el que establece la memoria mínima para la ejecución del modelo.

Los entrenamientos son realizados en un equipo con procesador Intel Core ultra 7 165H a 3.8GhZ, GPU Nvidia RTX 1000 Ada Generation y 16GB de memoria RAM.

Capítulo 4

Resultados

Este apartado recoge los resultados de las distintas pruebas propuestas en el Capítulo 3 para la comparación de las distintas tecnologías de clasificación de sonidos ambientales.

La Tabla 4.1 recoge os resultados obtenidos para cada uno de los modelos evaluados siguiendo la metodología planteada en la Sección 3.3.

Tipo	**Modelo**	**#Param**	**Test acc.**	**T. entrenamiento**
SVM	-	–	0.63	3s
CNN	CNN-Mel (big)	1.1M	0.88	1h 6min
	CNN-Mel (small)	**17K**	**0.84**	**4min 19s**
	1D-CNN	291K	0.66	33min
	ResNet-56	274K	0.71	1h 17min
	ResNet-50 (pretrained)	23M	0.91	16min 49s
Transformers	AST	25K	0.19	28min
	Wav2Vec	423K	0.33	4h 6min

Tabla 4.1: Comparativa de modelos en precisión, tamaño y tiempo de entrenamiento

Respecto al modelo de SVM, se han entrenado y evaluado modelos con distintas características de entrada, todos ellos entrenados utilizando un parámetro de regularización $C = 1$ y utilizando kernel lineal, RBF, polinomial grado 3 y sigmoide. Como se observa en la Tabla 4.2. Utilizando un kernel lineal, la característica a través de la que se obtiene los mejores resultados es MFCC, lo que explica su uso extendido en otros trabajos de este tipo. Sin embargo, al extender la prueba a otros kernels, se observa una mejora en el resultado utilizando características cromáticas con kernel de función de base radial (RBF) o polinomial llegando a un *accuracy* del 63 % en el conjunto de evaluación. El kernel sigmoidal no parece ofrecer buen rendimiento con ninguna de las características, apenas ofreciendo mejoras ante el clasificador aleatorio en algunos casos. El entrenamiento de un modelo con la concatenación de todas las características como entrada no aporta mejora respecto al mejor modelo mencionado.

En cuanto a los modelos ResNet, La ResNet-50 preentrenada para reconocimiento en imagen logra el mejor *accuracy* de todas las arquitecturas evaluadas, aunque también

Nombre	**Accuracy**			
	Lineal	**RBF**	**Polinomial**	**Sigmoidal**
Zero-Crossing Rate	0.20	0.26	0.17	0.07
RMS	0.15	0.22	0.16	0.11
MFCC	**0.57**	0.59	0.50	**0.21**
Chroma Features	0.33	**0.63**	**0.63**	0.09
Spectral Contrast	0.41	0.43	0.46	0.11
Tono	0.22	0.38	0.26	0.10
Spectral Flatness	0.17	0.24	0.17	0.14

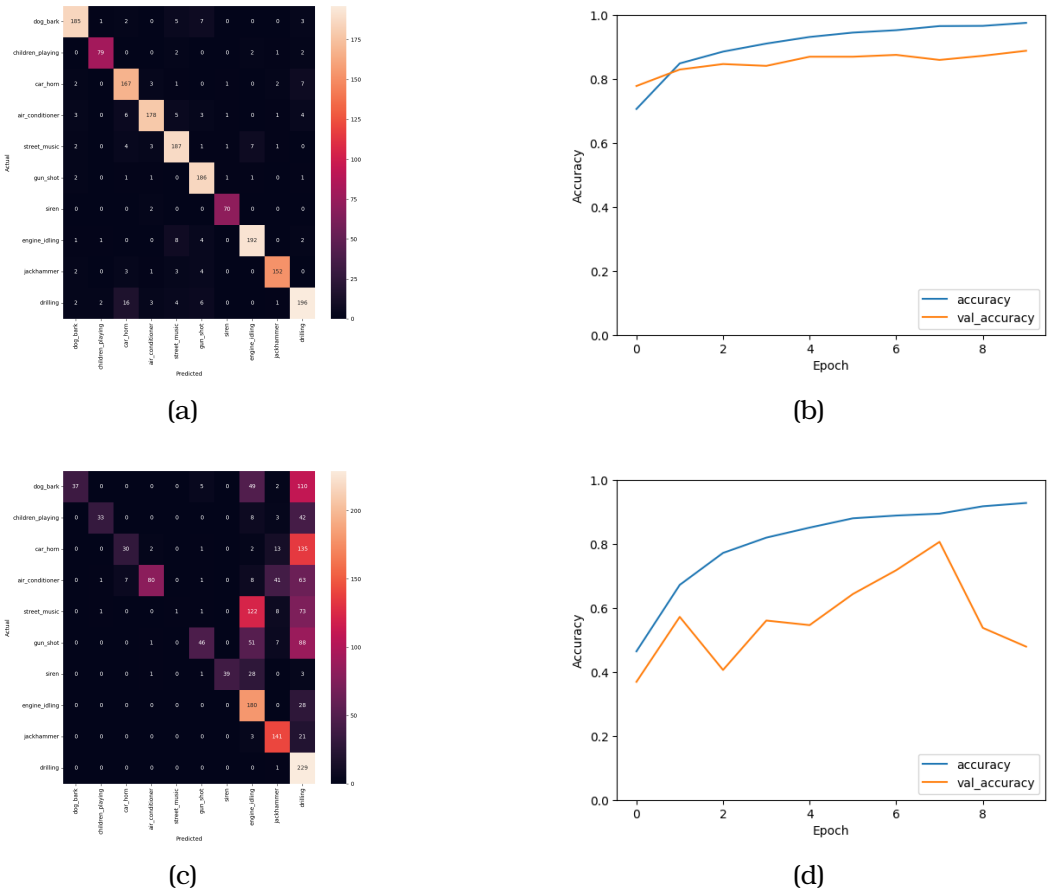
Tabla 4.2: Evaluación de características de entrada para SVM

es con diferencia la de mayor tamaño con un total de 23M de parámetros. Aunque debido a este tamaño el modelo no es apto para su uso en dispositivos embebidos, sirve como un buen punto de referencia para evaluar el desempeño del resto de modelos. Como se puede ver en los resultados de la ResNet-56, la reducción de este modelo resulta en una pérdida considerable de exactitud. Como se observa en las matrices de confusión y los historiales de entrenamiento de la Figura 4.1, mientras que ResNet-50 muestra un buen rendimiento para todas las clases y no presenta signos de sobreajuste, ResNet-56 no generaliza bien y presenta un sobreajuste notable, dando lugar a una fuerte tendencia a la predicción de la clase *drilling*. Cabe destacar que el historial mostrado en las Figuras 4.1b y 4.1d es de tan solo las primeras 10 épocas, por lo que es esperable que el sobreajuste observado en la ResNet-56 aumente todavía más durante su entrenamiento. En cuanto al tiempo de entrenamiento, es interesante observar que a pesar del gran número de parámetros de la ResNet-50, tan solo una pequeña porción de ellos (131K) son modificados durante el proceso de *fine-tuning*. Esto resulta en un tiempo de entrenamiento excepcionalmente corto en relación a los resultados alcanzados, especialmente cuando se compara con el de ResNet-56.

En resumen, aunque los modelos ResNet tienen la capacidad de alcanzar los mejores resultados de entre aquellos evaluados, necesitan un tamaño demasiado grande para que su uso sea viable en el caso de uso que se trata en este problema, en el que se trabaja con altas restricciones de memoria. Algo similar sucede con la red convolucional 1D. Obtiene un *accuracy* ligeramente peor que el alcanzado por ResNet-56 con incluso más parámetros.

Pasando a los modelos basados en *transformers*. El modelo AST alcanza un *accuracy* de tan solo 19 %, y ni siquiera el modelo Wav2Vec con un número de parámetros mucho más alto llega a resultados aceptables. Los *transformers* son modelos que mejoran su rendimiento con el número de parámetros y se benefician de altos volúmenes de datos de entrenamiento, dos recursos limitados en el contexto de este proyecto. Otros trabajos que afrontan problemas de clasificación audio a mediante el uso *transformers* habitualmente usan un número de parámetros y una cantidad de datos varias órdenes de magnitud mayor para llegar a buenos resultados, por lo tanto, su uso parece poco viable dadas las restricciones del proyecto.

Quedan por analizar son los modelos convolucionales básicos. Atendiendo a la relación entre el número de parámetros y el *accuracy* de ambos modelos, se observa



(a) (b)

(c) (d)

Figura 4.1: Matriz de confusión sobre el conjunto de test e historial de entrenamiento de los modelos ResNet-50 (a y b) y ResNet-56 (c y d)

que CNN-Mel (small) llega a un buen equilibrio entre tamaño y eficacia del modelo, logrando un *accuracy* de 84 % con únicamente 17K parámetros. La versión más grande de esta arquitectura sirve para demostrar que un aumento del tamaño de la red no supone una mejora significativa en los resultados a partir de este punto, por lo que es conveniente utilizar el modelo más pequeño posible obtenible sin una pérdida notable en la calidad de los resultados.

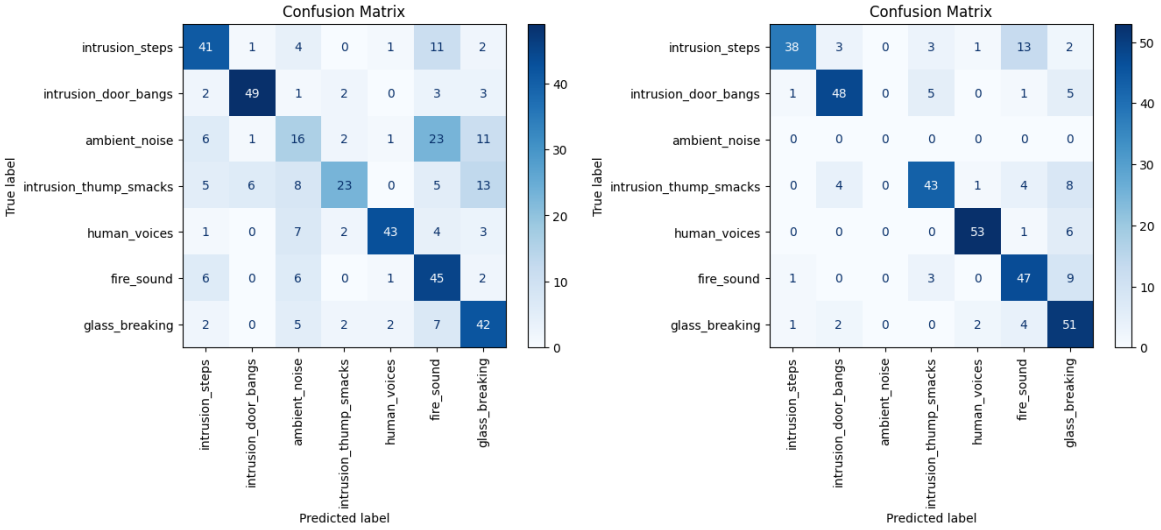
Dado el anterior análisis de los resultados, se decide utilizar el modelo CNN-Mel (small) en las sucesivas pruebas e implementaciones, modificando ligeramente su arquitectura para adaptarlo a distintos tamaños de entrada y salida según se crea conveniente.

4.1. Rendimiento del modelo en detección de intrusiones

Una vez comparadas las distintas tecnologías para clasificación de audio para un problema general de ESC mediante el *dataset* UrbanSound8K, determinando que las arquitecturas CNN básicas son las que mejor rendimiento ofrecen dadas las limitaciones temporales y de memoria de este proyecto, se quiere ver su desempeño en el caso de uso de interés en este trabajo, la detección de intrusiones, a través del conjunto de datos propio recogido tal y como se ha explicado en la Sección 3.3. Se

utiliza una versión ligeramente modificada de la arquitectura CNN-Mel (small) de 28K parámetros, con 7 neuronas de salida en lugar de 10 y un tamaño de entrada mayor, para adaptar la red a la mayor dificultad y el número de categorías de este conjunto de datos. Se entrena el modelo con el conjunto de datos balanceado al la clase con menos muestras durante 100 épocas y se comprueban los resultados.

Como se ha planteado en la Sección 3.1, se evalúa el rendimiento con el conjunto de datos completo, y excluyendo la clase *Ruido Ambiente*. En el primer caso se obtiene una exactitud de tan solo 61 % en el conjunto de test, un resultado por debajo de lo deseable, tal y como se esperaba por la mala calidad de los datos incluidos en esta categoría, que acumula la mayoría de los errores como se puede observar en la Figura 4.2a. Mientras tanto, el entrenamiento del modelo excluyendo esta categoría se estabiliza con un *accuracy* en el conjunto de test del 79 % (ver Figura 4.3a), y su matriz de confusión, visible en la Figura 4.2b indica que no parece haber ninguna categoría con desempeño considerablemente inferior al resto.



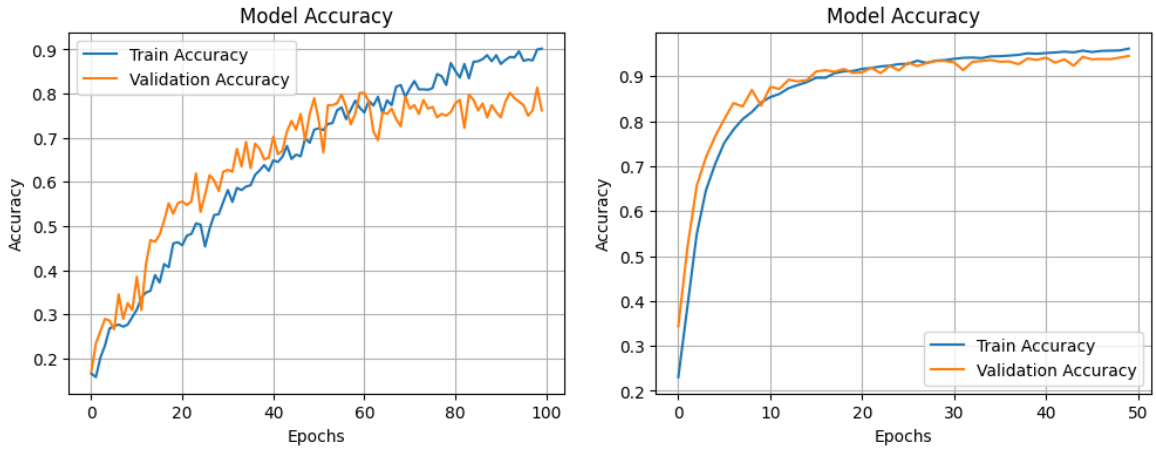
(a) Incluyendo la clase *ambient noise*

(b) Sin incluir la clase *ambient noise*

Figura 4.2: Estudio del impacto de la cantidad de datos en la precisión del modelo

Por otro lado, se quiere probar el efecto de la aplicación de técnicas de aumento de datos en el proceso de entrenamiento en los resultados obtenidos. Se implementan las siguientes técnicas de aumento:

- **Escalado de amplitud**: Se modifica el valor de la amplitud escalando su valor por un valor aleatorio entre 0.5 y 1.5, constante a lo largo de todo el audio. A efectos prácticos, haciendo el sonido más fuerte o más tenue.
- **Desplazamiento de tono**: Se modifica el tono del audio en un valor de hasta media octava en cualquiera de los sentidos.
- **Adición de ruido**: Se añade al audio ruido aleatorio uniforme, lo que permite simular de cierto modo ruido ambiente o estático que opaca el sonido de interés.
- **Reverberación**: Se añade en cada muestra del audio el valor anterior multiplicado por un factor de escala entre 0.9 y 1.
- **Estiramiento temporal**: Consiste en encoger o estirar la onda de audio, cam-



(a) Entrenamiento normal

(b) Con aumento de datos

Figura 4.3: Historial de entrenamiento del modelo con el conjunto de datos personalizado, con y sin aumento de datos

biando la velocidad a la que se reproduce el sonido en un ratio entre 0.5 (mitad de velocidad) y 2 (doble de velocidad).

Del conjunto de datos original, se reservan 100 muestras de cada clase para ser utilizadas como test, y del resto se extraen con repetición 1000 muestras por clase, que se añaden al conjunto de entrenamiento junto a sus copias aumentadas por cada uno de los métodos utilizados. Esto da lugar a un conjunto de entrenamiento final con 6000 muestras por clase, es decir, 42000 muestras totales.

Los resultados del entrenamiento se pueden ver en la Figura 4.3b. A primera vista, se logra un entrenamiento más estable, que se estabiliza en menos épocas, y no presenta ninguna muestra de sobreajuste. Sin embargo, hay varios factores a tener en cuenta a la hora de interpretar esta gráfica. En primer lugar, se debe tener en cuenta que el número de muestras con las que se entrena el modelo en cada época es aproximadamente 30 veces más grande que sin *data augmenting*. Al seguir el entrenamiento en tiempo de ejecución, ambos modelos alcanzan resultados parecidos en tiempos similares. En cuanto a la eliminación del sobreajuste, es importante tener en cuenta que el conjunto de validación contiene datos aumentados, y es posible que el modelo esté aprendiendo de los patrones del aumento de datos en lugar de la información característica de los audios.

Al comprobar el rendimiento del modelo sobre el conjunto reservado de test, que no contiene audios aumentados, se obtiene un *accuracy* del 81.5 %, muy por debajo del observado en la gráfica, lo que indica por un lado que efectivamente el modelo se está ajustando al aumento de datos, y por otro, que la mejora en rendimiento real es de aproximadamente un 3 %.

4.2. Comprobación de varianza e impacto de nuevos datos

Por último, se tiene interés en conocer el comportamiento del modelo ante nuevos conjuntos de datos, observando la varianza de los resultados. Para esto, se entrena el modelo con *k-fold cross validation* [71]. Se divide el conjunto de datos en 10 par-

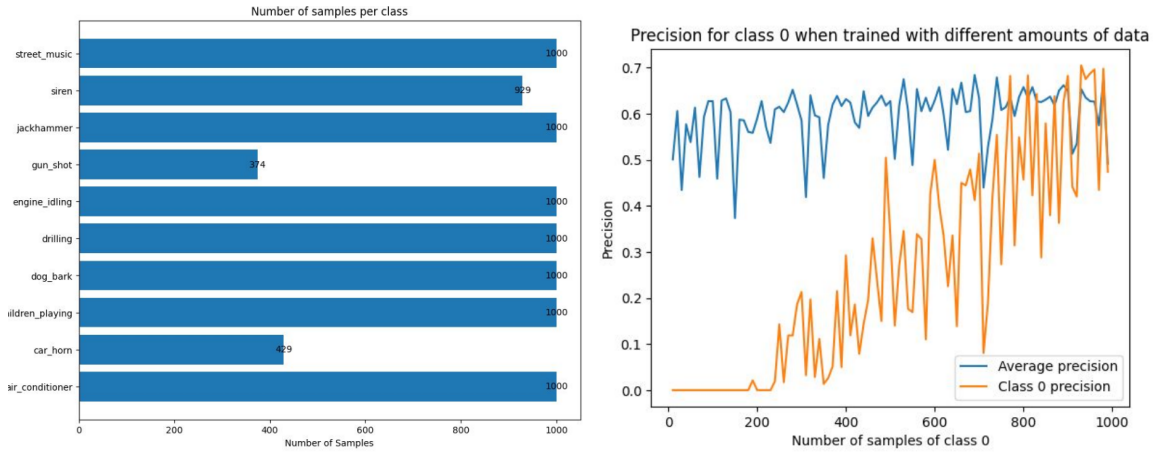
ticiones distintas. Se entrena el modelo con 9 de ellos y se utiliza la restante para evaluar su funcionamiento. Se repite este proceso hasta evaluar con cada una de las particiones y se extraen métricas sobre los datos recogidos.

Clase	Desviación estándar
1	0.09
2	0.13
3	0.12
4	0.19
5	0.17
6	0.17
7	0.16
8	0.14
9	0.24
10	0.17

Tabla 4.3: Desviación estándar por clase de predicción

La desviación típica en el *accuracy* obtenido para cada partición es de tan solo 0.02, lo que significa que el modelo tiene una varianza baja, es decir, no tiene una dependencia notable de la partición de datos utilizada para entrenar y parece ser capaz de generalizar ante conjuntos de datos de entrenamiento distintos. Se calcula también la desviación típica en la precisión de cada una de las categorías, la cual se puede ver en la Tabla 4.3.

Al observar estos datos, llama la atención que las clases 4 y 9, que son aquellas desbalanceadas con menos muestras que el resto en el conjunto de datos, son las que mayor varianza presentan en su precisión. Al ver esto, surge la idea de estudiar el impacto de la cantidad de datos de entrenamiento utilizados en la calidad de los resultados. Para ello, se entrena el modelo con diferentes cantidades de muestras en una clase entre 0 y 1000 dejando intacto el resto del *dataset* y se estudia la precisión en dicha clase en cada caso. Los resultados de dicho estudio se encuentran en la Figura 4.4b. De ella, se puede extraer que la cantidad de datos en una clase mantiene una relación casi lineal con la precisión obtenida para dicha clase. Por lo tanto, el tamaño del conjunto de datos y su balance resulta un factor imprescindible a la hora de la elección o recogida de conjuntos de datos para este modelo.



(a) Número de muestras por categoría en UrbanSound8K

(b) Precisión en clase 0 en función de la cantidad de muestras

Figura 4.4: Estudio del impacto de la cantidad de datos en la precisión del modelo

Capítulo 5

Implementación y prueba de concepto

5.1. Características del microcontrolador

Una vez seleccionado el modelo de clasificación para el problema de detección de intrusiones que se trata en este trabajo, se quiere comprobar la viabilidad de su integración en un sistema embebido real a modo de prueba de concepto.

El microcontrolador utilizado para esta prueba dispone de un procesador Cortex-M33, 64KB de memoria RAM y 512kB de memoria flash. El procesador incluye componentes espécificos que son útiles en el problema abordado, como una unidad de punto flotante (FPU) o un procesador de señales digitales (DSP) que facilitan las operaciones sobre las señales previas a la inferencia de forma eficiente.

5.2. Optimización del modelo

Como se ha cubierto en la Sección 2.3, es habitual la aplicación de diversas técnicas de optimización de redes neuronales, especialmente en aquellas diseñadas para su ejecución en sistemas embebidos. Su uso puede aportar grandes ventajas en el despliegue de un sistema real, por lo que estudian varias de estas técnicas, concretamente, se han evaluado las técnicas de cuantizacón, poda (*pruning*) y agrupamiento (*clustering*) de modelos.

5.2.1. Cuantización

Por defecto, los parámetros de los modelos utilizan el formato de coma flotante de 32 bits (float32), lo cual implica un consumo de memoria considerable. La cuantización del modelo permite una reducción de su tamaño a través del uso de tipos de dato más pequeños. En el caso de este trabajo, esto permite la uso de modelos en el microcontrolador que de otra forma no podrían utilizarse, o lo que es lo mismo, permite el uso de modelo más complejos y potentes con el mismo coste en memoria.

Se evalúa el rendimiento del modelo entrenado en el conjunto de datos UrbanSound8K al aplicar cuantización post-entrenamiento lineal uniforme convirtiendo los pesos y activaciones a 16 bits. Al utilizarlo con tamaños de dato más pequeños el modelo

pierde demasiado *accuracy*. Para solucionar esto, al tener acceso a los datos de entrenamiento, se aplica entrenamiento consciente de cuantización para convertir los parámetros a uint8. Los resultados de estas evaluaciones se pueden ver en la Tabla 5.1.

Modelo	Accuracy	Memoria flash (kB)	RAM (kB)
Baseline	0.8889	833	–
Convertido a TFLite	0.8889	264	162
Post-entrenamiento	0.8900	135	81
Con entrenamiento consciente	0.8614	75	40.5

Tabla 5.1: Comparativa de métodos de cuantización de modelos

Como es de esperar, la cuantización post-entrenamiento a float16 supone una reducción en un 50 % del tamaño del modelo, tanto en memoria persistente como temporal, aparentemente sin ninguna impacto negativo en la calidad de los resultados. Por su parte, la aplicación de entrenamiento consciente de cuantización logra una razón de compresión del 75 % con una pérdida de *accuracy* de tan solo un 2 %.

5.2.2. *Pruning* y *Clustering*

Las técnicas de *Pruning* y *Clustering* apenas ofrecen beneficios en cuanto a la eficiencia del modelo, pero tienen utilidad en el problema tratado en este trabajo. En el caso de un despliegue real de un dispositivo que incluya el sistema de detección desarrollado en este trabajo, la actualización del modelo para su mejora o para añadir funcionalidades supone un reto por la cantidad de datos que el dispositivo tiene que recibir. Estas técnicas de optimización permiten un menor tamaño del modelo cuando se codifica con algoritmos de compresión, lo que reduce los recursos de computacionales y de red necesarios para su envío.

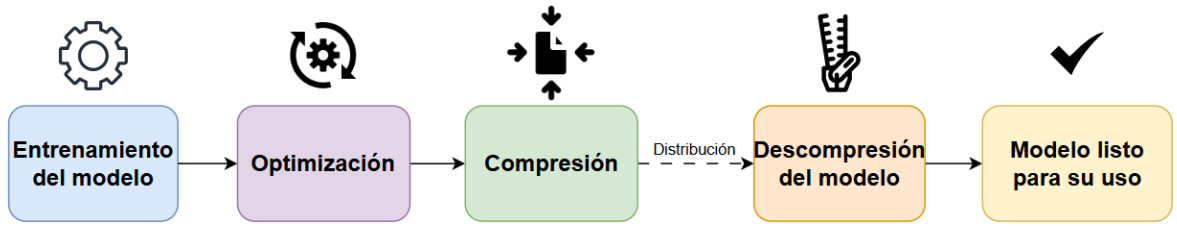


Figura 5.1: Diagrama de distribución del modelo a los dispositivos conectados

La Figura 5.1 muestra el posible proceso a seguir para la distribución de una versión del modelo a los dispositivos conectados: Una vez entrenado el nuevo modelo, se le aplican los distintos métodos de optimización para después convertirlo en un fichero comprimido, el cual es enviado a cada uno de los dispositivos instalados que sólo deben descomprimirlo para actualizar su imagen local del modelo.

La Tabla 5.2 muestra los resultados de la aplicación de ambos métodos al modelo utilizado, entrenado con el conjunto de datos *UrbanSound8K*. Además de evaluar cada una de las técnicas de optimización con respecto al modelo base, se combinan ambos métodos junto con cuantización por entrenamiento consciente mediante *Sparsity and cluster preserving quantization aware training* (PCQAT) [72], que mantiene la

dispersión y las agrupaciones durante el proceso de cuantización. Se utiliza un 0.8 de dispersión final en *pruning* y 16 *clusters* en *clustering*.

Modelo	Accuracy	Tamaño (kB)	Tamaño gzip (kB)
Baseline	0.8889	833	491
TFLite	0.8889	264	156
Pruning	0.8454	518	92
Clustering	0.8763	286	32
PCQAT	0.8603	75	28

Tabla 5.2: Comparativa de métodos de compresión de modelos

Ambas técnicas de optimización reducen en gran medida el tamaño del modelo al comprimirlo mediante gzip [73], basado en el algoritmo de compresión de Lempel-Ziv [74]. La aplicación de *clustering* reduce el tamaño del fichero comprimido a tan solo 32kB con una pérdida de *accuracy* de apenas un 1 %. Los resultados mediante *pruning* también son buenos, aunque consigue una peor compresión y presenta una pérdida de *accuracy* del 5 %.

Adicionalmente, se observa que aplicando PCQAT presenta los beneficios de todas las técnicas que combina, no solo reduciendo todavía más el el tamaño del fichero comprimido hasta los 28kB, sino que además mantiene el tamaño en memoria persistente de 75kB obtenido mediante cuantización con entrenamiento consciente una vez descomprimido el modelo.

5.3. Funcionamiento en el microcontrolador

Se quiere comprobar el funcionamiento del modelo desarrollado en el microcontrolador definido en la Sección 5.1. Para ello, se debe realizar una implementación específica.

Se utiliza la librería Tensorflow Lite (LiteRT) para Microcontroladores [75] como herramienta para realizar las inferencias. El modelo ha sido desarrollado con Keras por lo que el primer paso es almacenarlo como modelo *.tflite*. Además, el microcontrolador no cuenta con sistema operativo, y por tanto no dispone de sistema de ficheros, por lo que no es posible cargar el modelo desde el fichero como se haría habitualmente. En su lugar, se convierte a un array de C que puede ser incluido dentro del programa que se va a ejecutar. Se asigna una sección de memoria destinada a la inferencia de acuerdo al tamaño de buffer máximo del modelo y se implementa el código necesario para la realización de las evaluaciones realizadas en la siguiente sección.

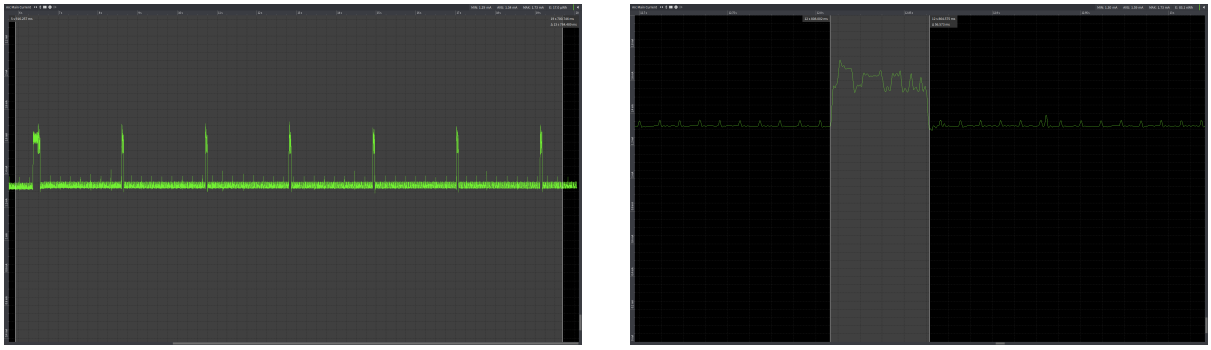
El sistema implementado no incluye las fases de captura de sonido mediante micrófono ni su posterior extracción de características. En su lugar, se utilizan los espectrogramas de Mel ya cargados en memoria como entrada del clasificador.

5.4. Evaluación

Una vez funcionando el modelo en el microcontrolador, con el objetivo de comprobar si el sistema desarrollado cumple con las restricciones planteadas en los objetivos del

proyecto, se realiza una evaluación del tiempo de ejecución y consumo energético del proceso de clasificación.

Para la realización de esta validación se dispone del dispositivo de medición Otii Ace [76], que permite un análisis detallado de la energía consumida por el sistema. La Figura 5.2a muestra los picos de energía consumida fruto de varias inferencias sucesivas a través de la visualización de la corriente del sistema en función del tiempo. La Figura 5.2b permite ver en detalle una de estas inferencias.



(a) (b)

Figura 5.2: Análisis de corriente del dispositivo en (a) múltiples inferencias y (b) una inferencia

En cuanto al tiempo de ejecución, se observa un intervalo de 56.57ms desde el inicio hasta el fin de la clasificación, incluyendo las operaciones previas y salidas de *log*. Contando los ciclos de reloj necesarios para la inferencia del modelo se calcula el tiempo específico asociado a esta tarea, siendo de 27.42ms de media. Para un análisis más profundo del proceso se mide el tiempo de ejecución de cada una de las capas de la red, los cuales se encuentran reflejados en la Tabla 5.3. Se observa que casi la totalidad del coste temporal de la inferencia se encuentra en las capas convolucionales, mientras que las de *max pooling*, *reshape* y *fully connected* suponen un gasto casi despreciable en comparación. Además, estos datos confirman que las operaciones de cuantización y descuantización tienen un costo total de tan solo 0.25ms, lo que incentiva todavía más su uso.

Capa	**Tipo**	**Tiempo ejecución (ms)**	**#Ciclos CPU**
0	QUANTIZE	0.211	8,243
1	CONV_2D	8.210	320,222
2	MAX_POOL_2D	0.759	29,621
3	CONV_2D	11.580	451,491
4	MAX_POOL_2D	0.197	7,691
5	CONV_2D	5.310	207,000
6	RESHAPE	0.035	1,399
7	FULLY_CONNECTED	1.050	41,124
8	FULLY_CONNECTED	0.092	3,604
9	SOFTMAX	0.183	7,145
9	DEQUANTIZE	0.042	1,676

Tabla 5.3: Tiempo de ejecución por capa de la red neuronal

Para calcular el gasto energético de la inferencia, en primer lugar se calcula la diferencia de corriente durante la misma respecto al estado de reposo. La corriente media en reposo I_{idle} es 1.3 mA, mientras que durante la inferencia I_{inf} es de 1.59mA, por lo tanto, la diferencia ΔI es de 0.29mA (ver Ecuación 5.1). Con esto, y sabiendo que el sistema funciona a 3V se puede calcular una energía total consumida (ver Ecuación 5.2) de 83.91nWh por inferencia.

$$\Delta I = I_{inf} - I_{idle} \tag{5.1}$$

$$E = P \cdot t = V \cdot I \cdot t \tag{5.2}$$

A continuación se presenta un caso práctico que permite hacer una estimación del tiempo de vida de un dispositivo ejecutando el sistema de clasificación de audio. Asumiendo que se utiliza una alimentación de dos pilas de Litio AAA de 1200mAh de capacidad cada una, se dispondría de una capacidad total de 2400mAh. A través de la corriente y la capacidad, se puede estimar un tiempo de vida en reposo de 76.92 días (ver Ecuación 5.3). En el otro extremo, si el sistema se encontrase en todo momento realizando inferencias, el tiempo de batería se reduciría a 62.89 días, un impacto de aproximadamente un 18 %.

Para reducir lo máximo posible el gasto energético del sistema de clasificación de audio, se plantea la posibilidad de que se realice tan solo cierto número de inferencias por minuto, Como se observa en la Ecuación 5.4, conociendo el delta de corriente y el tiempo de la inferencia se calcula la capacidad pérdida por cada inferencia, que resulta en un coste de 4.557×10^{-6} mAh, restando aproximadamente 12.61ms de vida al sistema respecto a si estuviera en reposo. Suponiendo una inferencia por segundo, el gasto adicional por hora respecto al sistema en reposo sería de 0.0164mAh. Calculando el tiempo de batería con este método se obtendrían 75.97 días de vida útil, es decir, la incorporación del sistema supondría una pérdida de tan solo un 1.62 % de la vida del dispositivo. La Tabla 5.4 recoge estos resultados, mostrando el consumo por hora, el tiempo de batería y la pérdida de batería respecto al sistema en reposo.

$$t\ (\text{h}) = \frac{Q\ (\text{mAh})}{I\ (\text{mA})} \tag{5.3}$$

$$Q\ (\text{mAh}) = I\ (\text{mA}) \times t\ \left(\frac{\text{ms}}{3.6 \times 10^6}\right) \tag{5.4}$$

Inferencias	**Consumo/hora (mWh)**	**Tiempo de batería (días)**	**Pérdida (%)**
Nunca	1.3	76.92	0
Cada 1s	1.316	75.96	1.62
Constante	1.59	62.89	18.24

Tabla 5.4: Tiempo de vida del sistema para distintos intervalos de inferencia

Es importante tener en cuenta que la corriente en reposo de la placa de evaluación con la que se han realizado estas pruebas es considerablemente mayor el que tendría

un sistema real, por lo que cabría esperar un tiempo de vida todavía mayor en un sistema final.

Capítulo 6

Conclusiones

Puede afirmarse que se ha cumplido con los objetivos establecidos en la Sección 1.3.

Se han evaluado distintas tecnologías para clasificación de sonidos ambientales respetando unas limitaciones restringidas y se ha recopilado un conjunto de datos para evaluar el rendimiento del modelo seleccionado en el caso de uso de la detección de intrusiones. Además, se han realizado las modificaciones necesarias para integrar este sistema de detección en un microcontrolador, estudiando y aplicando diversas técnicas de optimización, para después medir el rendimiento en este microcontrolador en términos de tiempo de ejecución y consumo de energía.

En cuanto al estudio preliminar de tecnologías, queda claro que métodos tradicionales como SVMs se quedan atrás en rendimiento con respecto a técnicas basadas en *deep learning*. Las ResNet ofrecen un muy buen rendimiento a cambio de un número de parámetros demasiado elevado y queda claro que los modelos que hacen uso de *transformers* requieren más datos de entrenamiento de los que se dispone. En cuanto a las redes convolucionales básicas, se ha podido ver que se puede obtener un muy buen resultado con redes pequeñas, con mejoras menos significativas a medida que se aumenta la complejidad de la red a partir de cierto umbral.

El modelo CNN seleccionado es capaz de clasificar correctamente sonidos relacionados con intrusiones con alrededor de un 80 % de *accuracy* a través de su espectrograma de Mel, con un tamaño de tan solo 75kB en memoria flash y 40 kB de RAM, y es capaz de generalizar y adaptarse a nuevos conjuntos de entrenamiento sin necesidad de realizar aumento de datos.

Respecto a los métodos de optimización estudiados, la cuantización ha permitido comprimir el modelo en una razón del 75 %, y se ha visto cómo los métodos de *clustering* y *pruning* pueden ser de gran utilidad en la distribución de actualizaciones en el modelo a dispositivos conectados.

Por último, la integración en el microcontrolador ha demostrado que el modelo de clasificación desarrollado se adecúa a las restricciones establecidas con un tiempo de ejecución suficientemente corto para realizar detecciones en tiempo real y un gasto energético bajo que permita el la alimentación del microcontrolador por pilas, con un impacto en en el tiempo de vida del dispositivo de menos del 2 % respecto a su funcionamiento en reposo.

6.1. Líneas de trabajo futuras

Analizando el trabajo realizado y los resultados obtenidos surgen varias posibles líneas de trabajo.

En primer lugar, aunque el conjunto de datos con sonidos de intrusiones recogido es suficiente para comprobar la viabilidad general de la utilización del modelo seleccionado para su uso en el campo de la seguridad, no se ha planteado su desarrollo con el objetivo de utilizarlo en una aplicación en producción. En su lugar, sería conveniente la grabación y el etiquetado de un conjunto de datos desde el hardware específico utilizado, lo que permitiría entrenar el modelo con datos que se asemejen en mayor medida a los que aparecerán durante la inferencia, con la misma sensibilidad a frecuencias, ruido propio etc...

En siguiente lugar, se plantea la posibilidad de iterar sobre el modelo de clasificación modificando su arquitectura con el objetivo de reducir todavía más su tamaño en memoria persistente y RAM. Esto facilitaría que el sistema de detección de intrusiones implementado se ejecute al mismo tiempo que otros procesos, incluso, almacenando y utilizando varios modelos de inteligencia artificial en el mismo microcontrolador.

Por otro lado, para la implementación realizada en el microcontrolador en este trabajo sólo se ha adaptado la etapa del proceso de detección relacionada con la clasificación realizada por la red neuronal. Un siguiente paso en esta línea de trabajo podría ser adaptar también la recogida de fragmentos de audio por un micrófono conectado al dispositivo y la extracción de características de dichos audios. Esto permitiría una evaluación del sistema completo y una estimación más certera del consumo total del sistema desarrollado.

Para terminar, el trabajo se ha desarrollado con las especificaciones de un único modelo de microcontrolador en mente, que cuenta con hardware específico para procesamiento de señales y manejo de datos en punto flotante. Sería interesante explorar el rendimiento del sistema sin estos componentes para estudiar la viabilidad de su uso en dispositivos que no dispongan de estos componentes.

Bibliografía

[1] K. Zaman, M. Sah, C. Direkoglu, and M. Unoki, "A survey of audio classification using deep learning," *IEEE Access*, vol. 11, pp. 106 620–106 649, 2023.

[2] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han, "Enable deep learning on mobile devices: Methods, systems, and applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 3, pp. 1–50, 2022.

[3] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on zigbee technology," in *2011 3rd international conference on electronics computer technology*, vol. 6. IEEE, 2011, pp. 297–301.

[4] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke, "A survey of lorawan for iot: From technology to application," *Sensors*, vol. 18, no. 11, p. 3995, 2018.

[5] B. Kaur and J. Singh, "Audio classification: Environmental sounds classification," https://hal.science/hal-03501143/, 2021, preprint available on HAL.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[7] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[8] E. Garcia-Cuesta, A. B. Salvador, and D. G. Pãez, "Emomatchspanishdb: study of speech emotion recognition machine learning models in a new spanish elicited database," *Multimedia Tools and Applications*, vol. 83, no. 5, pp. 13 093–13 112, 2024.

[9] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[10] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning internal representations by error propagation," 1985.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] L. Lu, H.-J. Zhang, and H. Jiang, "Content analysis for audio classification and segmentation," *IEEE Transactions on speech and audio processing*, vol. 10, no. 7, pp. 504–516, 2002.

[13] Z. Fu, G. Lu, K. M. Ting, and D. Zhang, "A survey of audio-based music classification and annotation," *IEEE transactions on multimedia*, vol. 13, no. 2, pp. 303–319, 2010.

[14] V. Peltonen, J. Tuomi, A. Klapuri, J. Huopaniemi, and T. Sorsa, "Computational auditory scene recognition," in *2002 IEEE international conference on acoustics, speech, and signal processing*, vol. 2. IEEE, 2002, pp. II–1941.

[15] S. Chu, S. Narayanan, and C.-C. J. Kuo, "Environmental sound recognition with time–frequency audio features," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 6, pp. 1142–1158, 2009.

[16] G. J. McLachlan and D. Peel, *Finite mixture models*. John Wiley & Sons, 2000.

[17] D. Barchiesi, D. Giannoulis, D. Stowell, and M. D. Plumbley, "Acoustic scene classification: Classifying environments from the sounds they produce," *IEEE Signal Processing Magazine*, vol. 32, no. 3, pp. 16–34, 2015.

[18] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Soundsense: scalable sound sensing for people-centric applications on mobile phones," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, 2009, pp. 165–178.

[19] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[20] S. Z. Li and G.-d. Guo, "Content-based audio classification and retrieval using svm learning," in *First IEEE Pacific-Rim Conference on Multimedia, Invited Talk, Australia*. Citeseer, 2000.

[21] L. Yang and H. Zhao, "Sound classification based on multihead attention and support vector machine," *Mathematical Problems in Engineering*, vol. 2021, no. 1, p. 9937383, 2021.

[22] K. Subashini, S. Palanivel, and V. Ramaligam, "Audio-video based segmentation and classification using svm," in *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*. IEEE, 2012, pp. 1–6.

[23] G. Guo and S. Z. Li, "Content-based audio classification and retrieval by support vector machines," *IEEE transactions on Neural Networks*, vol. 14, no. 1, pp. 209–215, 2003.

[24] D. Boswell, "Introduction to support vector machines," *Departement of Computer Science and Engineering University of California San Diego*, vol. 11, pp. 16–17, 2002.

[25] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath, "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.

[26] S. Abdoli, P. Cardinal, and A. L. Koerich, "End-to-end environmental sound classification using a 1d convolutional neural network," *Expert Systems with Applications*, vol. 136, pp. 252–263, 2019.

[27] S. Allamy and A. L. Koerich, "1d cnn architectures for music genre classification," in *2021 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2021, pp. 01–07.

[28] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, vol. 12, 2016.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[33] W. Bendada, T. Bontempelli, M. Morlon, B. Chapus, T. Cador, T. Bouabça, and G. Salha-Galvan, "Track mix generation on music streaming services using transformers," in *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023, pp. 112–115.

[34] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1441–1450.

[35] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," *arXiv preprint arXiv:2104.01778*, 2021.

[36] X. Liu, H. Lu, J. Yuan, and X. Li, "Cat: Causal audio transformer for audio classification," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.

[37] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira, "Perceiver: General perception with iterative attention," in *International conference on machine learning*. PMLR, 2021, pp. 4651–4664.

[38] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *arXiv preprint arXiv:1904.05862*, 2019.

[39] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020.

[40] A. Bansal and N. K. Garg, "Environmental sound classification: A descriptive review of the literature," *Intelligent systems with applications*, vol. 16, p. 200115, 2022.

[41] S. Chachada and C.-C. J. Kuo, "Environmental sound recognition: A survey," *APSIPA Transactions on Signal and Information Processing*, vol. 3, p. e14, 2014.

[42] K. J. Piczak, "Esc: Dataset for environmental sound classification," in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 1015–1018.

[43] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 1041–1044.

[44] M. R. Ahmed, T. I. Robin, and A. A. Shafin, "Automatic environmental sound recognition (aesr) using convolutional neural network." *International Journal of Modern Education & Computer Science*, vol. 12, no. 5, 2020.

[45] Z. Mushtaq, S.-F. Su, and Q.-V. Tran, "Spectral images based environmental sound classification using cnn with meaningful data augmentation," *Applied Acoustics*, vol. 172, p. 107581, 2021.

[46] V. Boddapati, A. Petef, J. Rasmusson, and L. Lundberg, "Classifying environmental sounds using image recognition networks," *Procedia computer science*, vol. 112, pp. 2048–2056, 2017.

[47] Z. Chi, Y. Li, and C. Chen, "Deep convolutional neural network combined with concatenated spectrogram for environmental sound classification," in *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*. IEEE, 2019, pp. 251–254.

[48] M. Burhan, R. A. Rehman, B. Khan, and B.-S. Kim, "Iot elements, layered architectures and security issues: A comprehensive survey," *sensors*, vol. 18, no. 9, p. 2796, 2018.

[49] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine learning model sizes and the parameter gap," *arXiv preprint arXiv:2207.02852*, 2022.

[50] R. Kallimani, K. Pai, P. Raghuwanshi, S. Iyer, and O. L. López, "Tinyml: Tools, applications, challenges, and future research directions," *Multimedia Tools and Applications*, vol. 83, no. 10, pp. 29 015–29 045, 2024.

[51] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, vol. 11, pp. 96 892–96 922, 2023.

[52] R. Singh and S. S. Gill, "Edge ai: a survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023.

[53] V. Tsoukas, E. Boumpa, G. Giannakas, and A. Kakarountas, "A review of machine learning and tinyml in healthcare," in *Proceedings of the 25th Pan-Hellenic Conference on Informatics*, 2021, pp. 69–73.

[54] R. Ke, Y. Zhuang, Z. Pu, and Y. Wang, "A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on iot devices," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 4962–4974, 2020.

[55] R. Marculescu, D. Marculescu, and U. Ogras, "Edge ai: Systems design and ml for iot data analytics," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3565–3566.

[56] A.-M. Comeagă and I. Marin, "Memory management strategies for an internet of things system," in *2023 International Symposium on Fundamentals of Electrical Engineering (ISFEE)*. IEEE, 2023, pp. 1–6.

[57] J. A. Stankovic, "Real-time and embedded systems," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 205–208, 1996.

[58] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[59] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[60] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[61] X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao, and T. Zhou, "A survey on knowledge distillation of large language models," *arXiv preprint arXiv:2402.13116*, 2024.

[62] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[64] F. Font, G. Roma, and X. Serra, "Freesound technical demo," in *Proceedings of the 21st ACM international conference on Multimedia*, 2013, pp. 411–412.

[65] E. Fonseca, X. Favory, J. Pons, F. Font, and X. Serra, "Fsd50k: an open dataset of human-labeled sound events," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 829–852, 2021.

[66] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. 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, 2017, pp. 776–780.

[67] H. J. Nussbaumer and H. J. Nussbaumer, *The fast Fourier transform*. Springer, 1982.

[68] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The journal of the acoustical society of america*, vol. 8, no. 3, pp. 185–190, 1937.

[69] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[70] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[71] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[72] B. Predić, U. Vukić, M. Saračević, D. Karabašević, and D. Stanujkić, "The possibility of combining and implementing deep neural network compression methods," *Axioms*, vol. 11, no. 5, p. 229, 2022.

[73] J.-l. Gailly and M. Adler, "Gnu gzip," *GNU Operating System*, pp. 8–18, 1992.

[74] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[75] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[76] "Otii Ace Pro | Qoitech User Manual — docs.qoitech.com," https://docs.qoitech.com/user-manual/otii/hardware/otii-ace-pro#datasheet, [Accessed 27-06-2025].