



POLITÉCNICA

escuela técnica superior de
ingeniería
y diseño
industrial

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO FIN DE GRADO

Análisis de la contaminación acústica mediante Inteligencia Artificial

Autor: Álvaro Vellella Ramos

Tutora:

Raquel Cedazo León
Departamento de Ingeniería
Eléctrica, Electrónica,
Automática y Física Aplicada

Madrid, septiembre 2022

Título del trabajo: Análisis de la contaminación acústica mediante Inteligencia Artificial.

Autor: Álvaro Vellella Ramos

Tutora: Raquel Cedazo León

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día de de en, en la Escuela Técnica Superior de Ingeniería y Diseño Industrial de la Universidad Politécnica de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO

PRESIDENTE

ÍNDICE

ÍNDICE.....	1
ÍNDICE DE FIGURAS.....	4
RESUMEN.....	7
ABSTRACT	8
LISTA DE ACRÓNIMOS.....	9
Capítulo 1. INTRODUCCIÓN	11
1.1. Importancia de la contaminación acústica	11
1.2. Inteligencia Artificial	12
1.3. Motivación	14
1.4. Objetivos	15
1.5. Estructura del documento	16
Capítulo 2. ESTADO DEL ARTE	17
2.1. Clasificación del sonido ambiental	17
2.2. Modelos tradicionales para la ESC.....	18
2.3. Redes Neuronales para la ESC	20
Capítulo 3. FUNDAMENTOS TEÓRICOS.....	29
3.1. Convolución	29
3.2. Redes neuronales convolucionales.....	30
3.2.1. Entrenamiento: Parámetros e Hiperparámetros.....	32
3.3. Sobreajuste	36
3.3.1. Aumento de datos	38
3.3.2. Regularización	38
3.4. Características del sonido	39
3.4.1. Niveles sonoros: el decibelio	41
3.4.2. Extracción de características del sonido.....	42
3.5. Clasificación de micrófonos	44
3.5.1. Por su principio de funcionamiento	44
3.5.2. Por su respuesta en frecuencia.....	46

3.5.3. Por su diagrama polar	46
Capítulo 4. DESARROLLO DEL PROYECTO	47
4.1. Software empleado.....	47
4.1.1. Entornos y herramientas empleadas.....	47
4.1.2. Librerías utilizadas	48
4.2. Hardware empleado	49
4.2.1. Raspberry Pi	50
4.2.2. Micrófono	51
4.2.3. Batería externa	54
4.2.4. Sonómetro	55
4.2.5. Presupuesto	56
4.2.6. Magerit-3	57
4.3. Desarrollo remoto con Magerit-3.....	58
4.4. Conjunto de datos.....	60
4.5. Etapas del proyecto	62
4.5.1. Desarrollo de la CNN.....	63
4.5.1.1. Preprocesamiento del conjunto de datos US8K	63
4.5.1.2. Diseño, entrenamiento y validación de la CNN	67
4.5.1.2.1. Modelo inicial	67
4.5.1.2.2. Resultados del modelo inicial	70
4.5.1.2.3. Modelo 2: Reducción de parámetros	71
4.5.1.2.4. Resultados del modelo 2	72
4.5.1.2.5. Modelo 3: Aumento de datos.....	74
4.5.1.2.6. Resultados del modelo 3	74
4.5.1.2.7. Modelo 4: <i>Batch normalization</i>	76
4.5.1.2.8. Resultados del modelo 4	77
4.5.1.2.9. Modelo 5: Aumento de datos en imagen.....	78
4.5.1.2.10. Resultados del modelo 5	79
4.5.1.2.11. Modelo final: Regularización <i>dropout</i> y L2	82
4.5.1.2.12. Resultados del modelo final	84
4.5.1.2.13. Comparación de los modelos y elección final	86
4.5.1.2.14. Tiempos de entrenamiento	87
4.5.2. Configuración del dispositivo de medición.....	88

4.5.2.1.	Configuración inicial de la RPI	88
4.5.2.2.	Configuración para la toma de audios	89
4.5.2.3.	Calibración del micrófono	92
4.5.3.	Toma de muestras	94
4.5.3.1.	ETSIDI.....	94
4.5.3.2.	Campus de Montegancedo	95
4.5.4.	Clasificación de muestras	96
Capítulo 5. RESULTADOS.....		99
Capítulo 6. CONCLUSIONES Y PERSPECTIVAS FUTURAS.....		109
6.1.	Conclusiones	109
6.2.	Perspectivas futuras.....	110
BIBLIOGRAFÍA		111
Anexo A. CÓDIGO		118
A.1.	<i>Script</i> de inicio de la RPI.....	118
A.2.	<i>Script</i> de grabación	120
A.3.	<i>Script</i> de aumento de datos y preprocesamiento	128
A.4.	<i>Script</i> de entrenamiento del modelo CNN final.....	131
A.5.	<i>Script</i> de preprocesamiento de las muestras	137
A.6.	<i>Script</i> de clasificación de las muestras.....	138
A.7.	<i>Script</i> para transferir archivos desde RPI.....	139

ÍNDICE DE FIGURAS

Figura 1-1. Campos dentro de la Inteligencia Artificial [12].	13
Figura 2-1. Flujo de procesamiento típico de un sistema de ESC [31]......	18
Figura 2-2. Arquitectura CNN con tres capas convolucionales y dos de agrupación [38].	22
Figura 2-3. Conexiones para un nodo de una capa de CLNN [45].	24
Figura 2-4. Un paso de una MCLNN [45].....	25
Figura 2-5. Ventana deslizante sobre señal de sonido con superposición del 50% [46].	26
Figura 2-6. Capa de una RNN [47]......	27
Figura 3-1. Operación de convolución discreta [49].	30
Figura 3-2. Extracción de características de una CNN [50].	31
Figura 3-3. Neurona artificial [51].	32
Figura 3-4. Funciones de activación Sigmoide, Tanh y ReLU [54].....	35
Figura 3-5. Efecto del sobreajuste en las curvas de aprendizaje [55].....	36
Figura 3-6. Subajuste y sobreajuste de un modelo de clasificación [56].	37
Figura 3-7. Presión acústica [60].	40
Figura 3-8. Nivel de presión sonora.	41
Figura 3-9. Espectrograma log-Mel de 128 bandas.	43
Figura 3-10. Espectrograma delta-log-Mel de 128 bandas.....	43
Figura 3-11. Principio de funcionamiento de los micrófonos dinámicos [63].	44
Figura 3-12. Principio de funcionamiento de micrófonos de condensador [63].....	45
Figura 3-13. Tipos de diagramas polares [65].	46
Figura 4-1. Dispositivo de medición.	49
Figura 4-2. Raspberry Pi 4.	50
Figura 4-3. Omnitronic MIC MM-2USB.	52
Figura 4-4. Respuesta en frecuencia de Omnitronic MIC MM-2USB [67].	53
Figura 4-5. Diagrama polar de Omnitronic MIC MM-2USB [67].....	53
Figura 4-6. Batería externa ADDTOP 26800mAh.	54
Figura 4-7. Sonómetro VOLT CRAFT Schallpegel-Messgerät SL-200.	55

Figura 4-8. Presupuesto del proyecto.....	57
Figura 4-9. Configuración de nodos que componen Magerit [69].....	57
Figura 4-10. Trabajo para ejecutar un <i>script</i> de Python.	59
Figura 4-11. Lanzamiento y listado de trabajos en Magerit-3.....	60
Figura 4-12. (a) Duración total por clase. (b) Número total de muestras por clase. Desglose por audio en primer plano (FG) y segundo plano (BG) [33].....	61
Figura 4-13. Metadatos contenidos en UrbanSound8K.csv.....	62
Figura 4-14. Diagrama de flujo de las etapas del proyecto.....	63
Figura 4-15. Diagrama de flujo del <i>script</i> de aumento de datos.....	65
Figura 4-16. Diagrama de flujo del <i>script</i> de preprocesamiento.	66
Figura 4-17. Resumen del modelo inicial.....	68
Figura 4-18. <i>Accuracies</i> de los conjuntos de entrenamiento y validación del modelo inicial.....	70
Figura 4-19. <i>Loss</i> de los conjuntos de entrenamiento y validación del modelo inicial. 71	
Figura 4-20. Resumen del modelo 2.	72
Figura 4-21. <i>Accuracies</i> de los conjuntos de entrenamiento y validación del modelo 2.	73
Figura 4-22. <i>Loss</i> de los conjuntos de entrenamiento y validación del modelo 2.....	73
Figura 4-23. <i>Accuracies</i> de los conjuntos de entrenamiento y validación del modelo 3.	75
Figura 4-24. <i>Loss</i> de los conjuntos de entrenamiento y validación del modelo 3.....	75
Figura 4-25. Resumen del modelo 4.	76
Figura 4-26. <i>Accuracies</i> de los conjuntos de entrenamiento y validación del modelo 4.	77
Figura 4-27. <i>Loss</i> de los conjuntos de entrenamiento y validación del modelo 4.....	78
Figura 4-28. Aumento de datos con ImageGenerator.	79
Figura 4-29. <i>Accuracies</i> de entrenamiento y validación del modelo 5 con relleno <i>constant</i>	80
Figura 4-30. <i>Loss</i> de entrenamiento y validación del modelo 5 con relleno <i>constant</i> ..	80
Figura 4-31. <i>Accuracy</i> de entrenamiento y validación del modelo 5 con relleno <i>nearest</i>	81
Figura 4-32. <i>Loss</i> de entrenamiento y validación del modelo 5 con relleno <i>nearest</i>	81
Figura 4-33. Resumen del modelo final.	83
Figura 4-34. Diagrama de flujo del <i>script</i> de entrenamiento del modelo final.....	84

Figura 4-35. <i>Accuracies</i> de los conjuntos de entrenamiento y validación del modelo final.....	85
Figura 4-36. <i>Loss</i> de los conjuntos de entrenamiento y validación del modelo final....	85
Figura 4-37. a) <i>Accuracies</i> de validación medias y máximas. b) Valores de <i>loss</i> para las <i>accuracies</i> de validación medias y máximas. (MI = Modelo Inicial, M1 = Modelo 1, M2 = Modelo 2, M3 = Modelo 3, M4 = Modelo 4, M5c = Modelo 5 <i>constant</i> , M5n = Modelo 5 <i>nearest</i> , MF = Modelo Final).....	86
Figura 4-38. a) Valores de <i>accuracy</i> máximos por pliegues. b) <i>loss</i> relativos a los valores de <i>accuracy</i> máximos para cada pliegue.....	87
Figura 4-39. Tiempos medios de entrenamiento en Magerit.....	88
Figura 4-40. Flujograma del <i>script</i> de inicio de la RPI.	90
Figura 4-41. Flujograma del <i>script</i> de grabación de muestras.....	91
Figura 4-42. Calibración del micrófono con sonómetro.	93
Figura 4-43. Calibración del micrófono con Smaart v8.....	93
Figura 4-44. Toma de sonido ambiental en la ETSIDI.....	95
Figura 4-45. Toma de sonido ambiental en el campus de Montegancedo.	96
Figura 4-46. Transferencia de muestras entre RPI y Magerit.	97
Figura 4-47. Flujograma del <i>script</i> de clasificación de muestras.	98
Figura 5-1. Porcentaje del número de muestras obtenidas por clase en la ETSIDI.	99
Figura 5-2. Porcentaje de las clasificaciones obtenidas en la ETSIDI.	100
Figura 5-3. Porcentaje del número de muestras obtenidas por clase en Montegancedo.	101
Figura 5-4. Porcentaje de las clasificaciones obtenidas en Montegancedo.	101
Figura 5-5. Número de muestras por clase obtenidas en la ETSIDI.	102
Figura 5-6. Número de muestras por clase obtenidas en Montegancedo.	103
Figura 5-7. Nivel de intensidad sonora por clase para el entorno de la ETSIDI.	104
Figura 5-8. Nivel de intensidad sonora por clase para el entorno del campus de Montegancedo.	104
Figura 5-8. Porcentaje de cada clase por horas en la ETSIDI.	105
Figura 5-9. Porcentaje de cada clase por horas en la ETSIDI (sin clases motor ni <i>other</i>).	106
Figura 5-10. Porcentaje de cada muestra por horas en el campus de Montegancedo.	107
Figura 5-11. Niveles de presión sonora medios y máximos por horas en la ETSIDI. ...	107
Figura 5-12. Niveles de presión sonora medios y máximos por horas en el campus de Montegancedo.....	108

RESUMEN

La contaminación acústica es un problema cada vez más presente en la sociedad actual y que afecta a las universidades. El objetivo de este proyecto es evaluar dicho problema en varios centros universitarios de la Universidad Politécnica de Madrid e identificar las fuentes sonoras del que procede.

Para ello, se utilizarán algoritmos de Inteligencia Artificial, tales como Redes Neuronales Convolucionales, que permitan llevar a cabo la tarea de Clasificación de los Sonidos Ambientales en sus diferentes categorías (motores, sirenas, ladridos de perro, etc.). Tras entrenar y validar el modelo con un conjunto de datos de libre acceso, se implementará en un dispositivo de bajo coste dotado con un micrófono.

Finalmente, se tomarán muestras de sonido ambiental en las distintas localizaciones y se analizarán los resultados obtenidos, haciendo un estudio de la contaminación acústica en los distintos centros.

ABSTRACT

Noise pollution is a problem that is increasingly present in today's society and that affects universities. The goal of this project is to evaluate this problem in some university centers of the Universidad Politécnica de Madrid and to identify the sound sources from which it originates.

For this purpose, Artificial Intelligence algorithms, such as Convolutional Neural Networks, will be used to carry out the task of Environmental Sound Classification in its different categories (engines, sirens, dog barking, etc.). After training and validating the model with a freely available dataset, it will be implemented in a low-cost device equipped with a microphone.

Finally, environmental sound samples will be taken in the different locations and the results obtained will be analyzed, making a study of the noise pollution in the different centers.

LISTA DE ACRÓNIMOS

OMS	Organización Mundial de la Salud
IA	Inteligencia Artificial
NLP	Procesamiento del Lenguaje Natural (<i>Natural Language Processing</i>)
ML	Aprendizaje Automático (<i>Machine Learning</i>)
ANN	Red Neuronal Artificial (<i>Artificial Neural Network</i>)
DL	Aprendizaje Profundo (<i>Deep Learning</i>)
UPM	Universidad Politécnica de Madrid
CNN	Red Neuronal Convolucional (<i>Convolutional Neural Network</i>)
ESC	Clasificación del Sonido Ambiental (<i>Environmental Sound Classification</i>)
ASR	Reconocimiento Automático de voz (<i>Audio Speech Recognition</i>)
MIR	Recuperación de Información Musical (<i>Music Information Retrieval</i>)
MFCC	Coeficientes Cepstrales en las Frecuencias de Mel (<i>Mel Frequency Cepstral Coefficients</i>)
CRP	Gráficos de Referencia Cruzada (<i>Cross Recurrence Plots</i>)
KNN	K-Vecinos más cercanos (<i>K-Nearest Neighbors</i>)
SVM	Máquina de Vectores de Soporte (<i>Support Vector Machine</i>)
GMM	Modelo de Mezcla Gaussiana (<i>Gaussian Mixture Model</i>)
HMM	Modelo Oculto de Markov (<i>Hidden Markov Model</i>)
RBF	<i>Radial Basis Function</i>
ZCR	Tasa de Cruce por Cero (<i>Zero Crossing Rate</i>)
ICA	Análisis de Componentes Independientes (<i>Independent Component Analysis</i>)
PCA	Análisis de Componentes Principales (<i>Principal Component Analysis</i>)

LDA	Análisis Discriminante Lineal (<i>Linear Discriminant Analysis</i>)
CLNN	Red Neuronal Condicional (<i>Conditional Neural Network</i>)
MCLNN	Red Neuronal Condicional Enmascarada (<i>Masked Conditional Neural Network</i>)
RNN	Red Neuronal Recurrente (<i>Recurrent Neural Network</i>)
LSTM	<i>Long-Short Term Memory</i>
GRU	<i>Gated Recurrent Unit</i>
SGD	<i>Stochastic Gradient Descent</i>
AdaGrad	<i>Adaptive Gradient Algorithm</i>
RMSprop	<i>Root Mean Square Propagation</i>
Adam	<i>Adaptive moment estimation</i>
dB SPL	Decibelios de Nivel de Presión Sonora (<i>Decibels Sound Pressure Level</i>)
dBFS	Decibelios a Escala Completa (<i>Decibels Full Scale</i>)
RPI	Raspberry Pi
SSH	<i>Secure Shell</i>
US8K	UrbanSound8K
ETSIDI	Escuela Técnica Superior de Ingeniería y Diseño Industrial
ETSIINF	Escuela Técnica Superior de Ingenieros Informáticos

Capítulo 1

INTRODUCCIÓN

1.1. IMPORTANCIA DE LA CONTAMINACIÓN ACÚSTICA

En la actualidad, debido al aumento de la población y de la actividad humana en las grandes ciudades, cada vez más personas están expuestas a un ambiente de ruido constante. El tráfico, las obras de construcción, las grandes concentraciones de personas o la actividad industrial son algunas de las principales fuentes de esta contaminación acústica, o también llamado, ruido ambiental.

Sin embargo, no todo sonido o ruido se considera contaminación acústica. El Ministerio de Transición Ecológica y el Reto Demográfico [1], define la contaminación acústica como *“la presencia en el ambiente de ruidos o vibraciones, cualquiera que sea el emisor acústico que los origine, que impliquen molestia, riesgo o daño para las personas, para el desarrollo de sus actividades o para los bienes de cualquier naturaleza, o que causen efectos significativos sobre el medio ambiente”*.

El ruido ambiental es un enemigo invisible y un problema real, que muchas veces es ignorado o subestimado debido a sus características. La contaminación acústica, en comparación con otros tipos de contaminación, no deja residuos ni tiene un efecto acumulativo en el medio, aunque sí que lo puede tener sobre el ser humano. Además, tiene un radio de acción mucho menor que otros contaminantes, ya que se localiza en espacios muy concretos. Tampoco se traslada a través de los sistemas naturales, como puede ser el caso del aire contaminado movido por el viento o sustancias químicas nocivas transportadas por el agua, por ejemplo. Es el contaminante más barato de producir y necesita muy poca energía para ser emitido. Además, el ruido se percibe por un único sentido: el oído, lo cual hace subestimar su efecto. Esto no sucede con el agua, por ejemplo, donde la contaminación se puede percibir por su aspecto, olor y sabor [2].

La Organización Mundial de la Salud (OMS) clasifica la contaminación acústica debida al tráfico como la segunda causa más importante de problemas de salud en Europa [3]. En la mayoría de los países europeos, más de un 50% de la población que

vive en áreas urbanas está expuesta a altos niveles de ruido y se estima que hasta 113 millones de personas se ven afectadas por el ruido del tráfico a largo plazo durante el día, la tarde y la noche. Además, con las proyecciones de crecimiento urbano en Europa y una mayor demanda de transporte, se prevé incluso un aumento del número de personas expuestas a la contaminación acústica en los próximos años [4].

La exposición prolongada a altos niveles de ruido, más allá de efectos negativos sobre la audición, puede causar graves daños en la salud de las personas [3]–[7]:

- **Efectos patológicos:** como dolor de cabeza, agitación respiratoria, aumento de la frecuencia cardiaca y de la presión arterial y otros efectos negativos en los sistemas cardiovascular y metabólico.
- **Efectos psicológicos:** el ruido puede provocar episodios de estrés, fatiga, depresión, ansiedad o histeria en las personas expuestas.
- **Alteración del sueño:** la exposición al ruido perturba el sueño en proporción a la cantidad de ruido percibida, aumentando el número de cambios en las etapas del sueño y de despertares. También puede haber efectos secundarios durante el día después de un sueño perturbado, como la calidad del sueño percibida, el estado de ánimo y el tiempo de reacción de la persona, pudiendo incluso influir en la conducta provocando episodios de agresividad o irritabilidad.
- **Memoria y atención:** el ruido puede afectar a la capacidad de concentración de las personas expuestas, lo que puede derivar en un bajo rendimiento académico o profesional a largo plazo. También afecta a la memoria, dificultando por ejemplo tareas como el estudio.

Las respuestas subjetivas al ruido, como la molestia o la alteración del sueño, no sólo dependen de los niveles de exposición, sino también de otros factores contextuales, situacionales y personales. Por ejemplo, los impactos pueden depender de la medida en que el ruido interfiere con lo que uno está tratando de hacer (por ejemplo, dormir, concentrarse o comunicarse) y la expectativa de paz y tranquilidad durante tales actividades [8].

1.2. INTELIGENCIA ARTIFICIAL

En la década de 1940, una serie de científicos sentaron las bases de la programación informática, una tecnología capaz de traducir una serie de instrucciones en acciones ejecutables por un ordenador. Estos precedentes hicieron posible que, en 1950, el matemático Alan Turing plantease por primera vez la pregunta de si es posible que las máquinas puedan pensar [9], plantando así la semilla de la creación de la Inteligencia Artificial (IA) [10].

La Inteligencia Artificial es la disciplina que trata de crear sistemas capaces de simular comportamientos inteligentes. Estos comportamientos pueden ser muy diversos, como analizar patrones, reconocer sonidos o tomar decisiones, y también son muchas las formas en las que una máquina puede simular un comportamiento inteligente. Un sistema puede simular ser inteligente, pero que en realidad su programación sea un conjunto de instrucciones sencillas que no lo dota de una inteligencia propia, como, por ejemplo, la programación secuencial de un brazo robótico que hace siempre el mismo movimiento.

Dentro del área de la Inteligencia Artificial, hay distintos campos que tratan de resolver los diferentes comportamientos inteligentes (Figura 1-1), como la robótica, que estudia la capacidad de moverse y adaptarse el entorno, el procesamiento del lenguaje natural (*Natural Language Processing*, NLP), que estudia la capacidad de entender el lenguaje, o la visión artificial, que intenta dotar a los sistemas de la capacidad de ver y reconocer su entorno. Sin embargo, si hay una capacidad que de verdad hace inteligente a un sistema, es la capacidad de aprender [11].

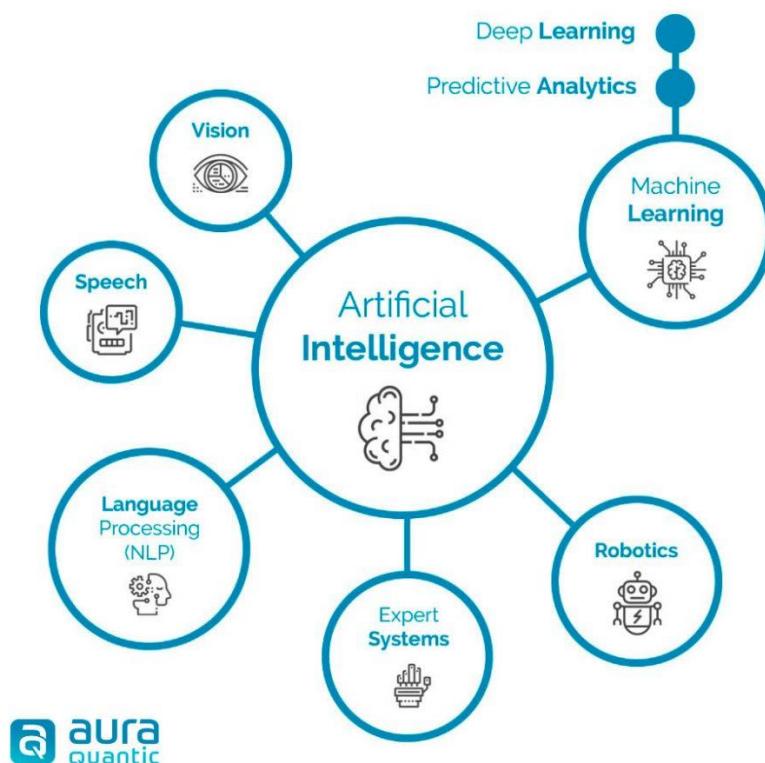


Figura 1-1. Campos dentro de la Inteligencia Artificial [12].

El Aprendizaje Automático o, también llamado, Aprendizaje Máquina (*Machine Learning*, ML), es la rama del campo de la Inteligencia Artificial, que trata de dotar a las máquinas de la capacidad de aprendizaje. Entendiendo éste, como el proceso de adquisición de conocimiento a través de un conjunto de experiencias. Esto permite a las

máquinas, por ejemplo, identificar una cara en una imagen, reconocer emociones en un texto o jugar al ajedrez, sin la necesidad de que ésta haya sido programada expresamente para realizar tales tareas. Esto quiere decir que es el propio sistema el que aprende automáticamente, por ejemplo, qué partes comprenden una cara (ojos, nariz, boca, etc.), qué palabras tienen un significado similar, o qué movimiento tiene la mayor probabilidad de ganar, sin que el programador haya intervenido en ello de forma explícita. Dependiendo de los datos disponibles y la tarea que se quiera abordar, se utilizarán distintos tipos de aprendizaje, como el supervisado, no supervisado o reforzado.

Las Redes Neuronales Artificiales son modelos que se basan en el proceso de cognición humano para abstraer información de forma jerarquizada. En ellas, la información se aprende por niveles, donde las primeras capas aprenden conceptos muy concretos, como qué es un tornillo, un espejo o una rueda, y en las capas posteriores se utiliza la información aprendida previamente para aprender conceptos más abstractos, como, por ejemplo, qué es un coche, un camión o una moto [11].

El aumento del número de capas de estos algoritmos y también de su complejidad, es lo que ha dado lugar al campo del Aprendizaje Profundo (*Deep Learning*, DL), que pertenece a su vez al campo del *Machine Learning*.

Y es que, aunque las Redes Neuronales Artificiales y otros algoritmos de *Machine Learning* datan de mediados del siglo pasado, ha sido en los últimos años que, debido al crecimiento exponencial del rendimiento y la capacidad de computación del hardware informático, se han empezado a aplicar en una gran variedad de campos, entre ellos, la clasificación de audio.

1.3. MOTIVACIÓN

Vivimos en un momento histórico en el que, gracias al impulso tecnológico sucedido en los últimos años, los algoritmos de Inteligencia Artificial nos han ayudado a resolver problemas complejos, mejorando así nuestra calidad de vida. Pero a su vez, la población de las ciudades no deja de aumentar y estamos cada vez más expuestos a problemas de contaminación acústica.

Los campus universitarios son especialmente sensibles al ruido, debido al carácter de las actividades que se desarrollan en ellos. Y es que, el ruido ambiental puede afectar de manera considerable a la capacidad de concentración y rendimiento de los estudiantes. Además, estos lugares pueden ser también una fuente potencial de contaminación acústica, debido a la gran concentración y afluencia de personas, al tráfico, las obras, aires acondicionados u otras instalaciones, sirenas y otras actividades no académicas que se desarrollan en ellos.

Por todos estos motivos, es de vital importancia mantener un entorno tranquilo y libre de contaminación acústica en los centros universitarios, para garantizar la calidad de las actividades, tanto académicas como no académicas, que se desarrollan en ellos.

¿Qué se puede hacer para alcanzar este objetivo? Pese a que pueda no tener una solución simple o eficaz, en cualquier caso, el primer paso para enfrentarse a este problema será evaluar el grado de contaminación acústica e identificar las causas que lo provocan.

Ante el actual contexto cambiante y en crisis, tanto ambiental como social, los campus universitarios están llamados a ser laboratorios donde experimentar ideas, prototipos y acciones que produzcan evidencias para construir nuevos modelos de ciudad [13]. Desde 2018, la Universidad Politécnica de Madrid (UPM) se ha dotado de un Plan Estratégico de Sostenibilidad Ambiental [14] para hacer frente al reto que supone esta crisis.

Este proyecto ha sido enmarcado en una colección de trabajos de fin de grado para la concienciación de la sensibilidad ambiental de la comunidad universitaria y para mejorar la sostenibilidad de los campus universitarios de la UPM [13].

1.4. OBJETIVOS

La meta principal de este proyecto es identificar las distintas fuentes de ruido mediante técnicas de Inteligencia Artificial para hacer un análisis de la contaminación acústica en varios centros de la UPM.

Para ello, se diseñará, entrenará y validará una red neuronal hasta obtener un modelo capaz de clasificar el ruido ambiental en varias clases con la mayor precisión y fiabilidad posible.

Posteriormente, se implementará en un dispositivo de bajo coste y se realizarán mediciones en las distintas localizaciones.

Por último, con los datos recogidos, se elaborará un análisis que refleje y compare las fuentes de contaminación acústica obtenidas para cada centro universitario.

1.5. ESTRUCTURA DEL DOCUMENTO

En los siguientes capítulos se expondrá el trabajo realizado para alcanzar los objetivos mencionados.

Se comenzará en el Capítulo 2 con el desarrollo del estado del arte para la clasificación de sonidos ambientales y la evolución de los modelos utilizados. Se dará una visión general del problema a tratar, se contextualizará la evolución de los métodos y arquitecturas utilizadas y se discutirán los modelos propuestos por la bibliografía.

En el Capítulo 3 se explicarán los conceptos y fundamentos teóricos que intervienen en el proyecto. Estos conceptos están relacionados con las Redes Nueronales Convolucionales (CNN, *Convolutional Neural Networks*) y el sonido. Se desarrollan los fundamentos en los que se basa el entrenamiento y funcionamiento de una CNN, las propiedades del sonido, su extracción de características y la caracterización de los distintos tipos de micrófonos.

En el Capítulo 4 se desarrollará el proceso seguido en cada etapa del proyecto. Se especificarán las herramientas hardware y software empleadas en él y el conjunto de datos utilizado para el entrenamiento y la evaluación de la CNN. Se describirá el proceso de entrenamiento de la CNN, desde el preprocesamiento del conjunto de datos hasta la obtención del modelo final, comentando todas las iteraciones realizadas. Posteriormente, se desarrollará el proceso de obtención de sonido ambiental, que incluye la configuración y calibración del equipo de medida y el procesamiento de las muestras de audio obtenidas, así como su clasificación final utilizando el modelo diseñado.

En el Capítulo 5 se discutirán los resultados obtenidos de la clasificación e intensidad de las muestras recogidas. Se realizará un análisis del ruido ambiental y se compararán los resultados obtenidos en cada entorno de grabación.

Finalmente, en el Capítulo 6 se expondrán las conclusiones extraídas del análisis de los resultados anteriores y se indicarán una serie de propuestas de mejora para los trabajos futuros relacionados con este proyecto.

Capítulo 2

ESTADO DEL ARTE

2.1. CLASIFICACIÓN DEL SONIDO AMBIENTAL

El ser humano, al igual que otros muchos seres vivos, tiene la capacidad de identificar y distinguir los sonidos de su alrededor en todo momento a través del oído. Sin embargo, lo que para nosotros es una habilidad natural, para un ordenador no es una tarea sencilla y abre todo un nuevo problema.

La clasificación automática de sonidos es un tema de investigación que ha cobrado un gran interés en los últimos años debido a la necesidad de los sistemas inteligentes de interpretar adecuadamente su entorno acústico o reconocer sonidos concretos. Existen varios campos de estudio relacionados con el reconocimiento y la clasificación del sonido y audio, como la clasificación de sonidos ambientales (*Environmental Sound Classification*, ESC), el reconocimiento automático de voz (*Audio Speech Recognition*, ASR) o la recuperación de información musical (*Music Information Retrieval*, MIR).

Sin embargo, aunque estos campos pueden parecer muy similares entre sí, son los sonidos ambientales los que, debido a su naturaleza no estructurada, presentan una mayor dificultad a los modelos de IA a la hora detectar patrones útiles que conduzcan a una clasificación precisa [15]. Y es que, el sonido ambiental, en contraste con el habla y la música, no tiene patrones de tiempo estáticos como ritmos, melodías o fonemas, por lo tanto, es difícil encontrar características universales que puedan representar los distintos patrones temporales. Además, contiene mucho ruido y otros sonidos irregulares y aleatorios no relacionados con el evento sonoro que se desea reconocer, lo que dificulta aún más la tarea de clasificación [16].

La ESC está presente en muchas aplicaciones prácticas, entre las que se incluyen el reconocimiento del contexto para distintos sistemas de vigilancia y seguridad [17], el hogar inteligente [18], la audición robótica [19], la monitorización del ruido en las ciudades [20] o su mitigación mediante el uso de sensores inteligentes [21]. La correcta

ESC permite a estos sistemas distinguir, por ejemplo, un sonido que supone un peligro, como una alarma, un cristal roto o un disparo, de otro completamente ordinario como puede ser una conversación entre personas.

2.2. MODELOS TRADICIONALES PARA LA ESC

Durante la última década se ha realizado una cantidad significativa de trabajo hacia el modelado de sistemas para la ESC. En la mayoría de estos trabajos, el procedimiento para resolver este problema de clasificación se separa en dos partes diferenciadas, mostradas en la Figura 2-1.

El primer paso es el procesamiento y la extracción de características de la señal sonora y su apropiada representación, donde se incluyen espectrogramas¹ [22], matrices de coeficientes cepstrales en las frecuencias de Mel (*Mel Frequency Cepstral Coefficients, MFCC*) [23], espectrogramas log-Mel [24], gráficos de recurrencia (*Cross Recurrence Plots, CRP*) [25] o filtros basados en las transformadas de Wavelet [26].

A continuación, se emplean diferentes algoritmos de clasificación de ML como K-vecinos más cercanos (*K-Nearest Neighbors, KNN*) [27], máquinas de vectores de soporte (*Support Vector Machine, SVM*) [28], modelos de mezcla Gaussiana (*Gaussian Mixture Model, GMM*) [29] o modelos ocultos de Markov (*Hidden Markov Model, HMM*) [30], que utilizan las representaciones de las características del sonido obtenidas anteriormente como entrada y que finalmente, clasifican el evento sonoro.

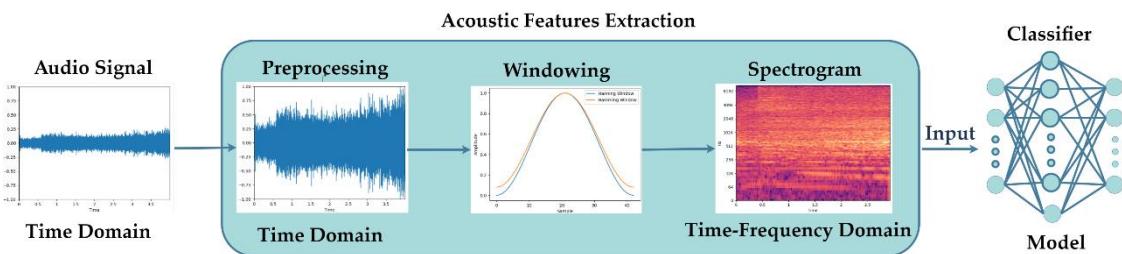


Figura 2-1. Flujo de procesamiento típico de un sistema de ESC [31].

El algoritmo de clasificación KNN calcula la distancia entre los nuevos datos de entrada y los k puntos más cercanos, lo que determina la clase de un nuevo punto de datos de entrada. Es adecuado para problemas de clasificación simples con funciones

¹ El concepto de espectrograma se detalla en el apartado 3.6.

básicas de entrenamiento. A medida que aumenta el número de funciones de entrenamiento, aumenta la complejidad computacional y el tiempo de KNN. SVM es factible cuando existe un claro margen de separación entre clases, y es más efectivo en espacios de alta dimensionalidad. Sin embargo, el rendimiento de SVM disminuye cuando el conjunto de datos tiene ruido o es demasiado grande. HMM es un método estadístico ampliamente utilizado para el reconocimiento de audio. Una de las principales ventajas de los HMM sobre los clasificadores descritos anteriormente es que consiguen modelar las correlaciones secuenciales de los datos con muestras vecinas [31].

Uno de los primeros trabajos en el área de la ESC fue el de los autores Eronen et al. [32] (2006), en el cual investigan el modelado de un sistema de reconocimiento del contexto basado en la clasificación de audio. Para la extracción de características emplearon un gran conjunto de técnicas, entre ellas el MFCC, la tasa de cruce por cero (*Zero Crossing Rate*, ZCR) o la energía media a corto plazo, y algunas transformaciones de características como el análisis de componentes independientes (*Independent Component Analysis*, ICA), análisis de componentes principales (*Principal Component Analysis*, PCA) y análisis discriminante lineal (*Linear Discriminant Analysis*, LDA) para reducir su dimensionalidad. Como métodos de clasificación se estudiaron KNN, GMM y HMM. Finalmente se realizaron distintos experimentos comparando el rendimiento del modelo propuesto con humanos, obteniendo una precisión de 58 % frente a 69 % para contextos y 82 % frente a 88 % para clases de alto nivel para el sistema y humanos, respectivamente.

En [27] (2006), Wang et al. proponen una arquitectura híbrida de SVM y KNN como clasificador. Para la extracción de características emplearon tres descriptores de bajo nivel. Los experimentos realizados sobre un conjunto de datos de sonidos ambientales obtuvieron una precisión del 85,1%, ligeramente superior al 83,2% obtenido con un clasificador HMM.

En [18] (2008), los mismos autores utilizaron un banco de filtros basado en las transformadas de Wavelet para el preprocesamiento de la señal sonora. A continuación, aplicaron un ICA sobre MFCC para alimentar a un clasificador SVM, que obtuvo un 73,6% de precisión en un experimento real para algunos sonidos del hogar.

En [33] (2014), Salomon et al. presentan la taxonomía de sonidos urbanos y un nuevo *dataset* que cobrará una gran importancia en los años futuros. Para estudiar las características de este conjunto de datos, emplean MFCC como método de extracción de características del sonido y cinco algoritmos de clasificación distintos: SVM (*Radial basis function kernel*), bosque aleatorio (500 árboles), KNN ($k = 5$), árbol de decisión (J48) y un clasificador de voto mayoritario de referencia (ZeroR). Pese a que el objetivo de este trabajo no es encontrar el modelo de clasificación óptimo, sólo los dos primeros algoritmos mostraron resultados significativos, con una precisión en torno al 70%.

En [34] (2015), estos mismos autores proponen un nuevo método de extracción de características, con el objetivo de capturar la dinámica temporal de las fuentes de

sonido urbanas. Para ello, extraen el spectrograma de log-Mel y le aplican un PCA para reducir su dimensionalidad. Posteriormente utilizan el algoritmo k-medias esférico para el aprendizaje de características del audio y que sirve como entrada de un clasificador de bosque aleatorio. En [35] (2015), siguiendo la metodología de su trabajo anterior, proponen la transformada de dispersión como una representación de señal alternativa al MFCC. Esta vez, utilizan además un clasificador SVM.

Aunque los algoritmos convencionales de ML han conseguido mejorar el rendimiento del reconocimiento de sonidos hasta cierto punto, también tienen claras deficiencias. Una de ellas, y la más importante, es su incapacidad para extraer características de tiempo y frecuencia, lo que obliga a construir representaciones de éstas a través de la operación manual, lo cual consume mucho tiempo. Además, para encontrar la mejor combinación de funciones de extracción y representación de características, a menudo se requieren muchos experimentos, haciendo el proceso bastante engorroso [16].

Sin embargo, con el desarrollo de la teoría del aprendizaje profundo, se ha demostrado que las redes neuronales tienen una gran capacidad para extraer características automáticamente y obtener resultados precisos. Además, también consiguen capturar características de tiempo-frecuencia, lo que resuelve las grandes limitaciones de los métodos tradicionales [22].

2.3. REDES NEURONALES PARA LA ESC

En los últimos años, las redes neuronales han tenido un gran éxito en las tareas de ESC y han conseguido un mejor desempeño que los algoritmos de ML tradicionales. Éstas son capaces de extraer características directamente de la señal de audio sin procesar o de otras características hechas a mano. Sin embargo, la arquitectura profunda y completamente conectada de las redes neuronales convencionales no es robusta frente a este tipo de datos [36], por lo que se han empleado otras variantes para resolver este problema.

Las **Redes Neuronales Convolucionales** (*Convolutional Neural Networks*, CNN) son unas de las arquitecturas de modelos de aprendizaje profundo más utilizadas para la clasificación y reconocimiento de patrones. Éstas cuentan con una o más capas convolucionales, las cuales están formadas por varios filtros de convolución que aprenden a extraer las características de los datos de entrada y que determinan posteriormente su clasificación.

Hay varios tipos de arquitecturas CNN dependiendo de los datos de entrada, las más utilizadas para la ESC son la CNN 1D y 2D.

La **CNN 1D** se utiliza para el procesamiento de datos de secuencia, es decir, la señal de audio. Esta red utiliza distintos filtros de convolución² 1D que se aplican directamente a la señal de audio de entrada para extraer las características del sonido que permitan su clasificación. A continuación, se añaden unas pocas capas de neuronas que son las encargadas de aprender la relación de dichas características extraídas con la etiqueta de la señal. Como consecuencia, se fusionan las operaciones de clasificación y extracción de características en un proceso que se puede optimizar para maximizar el rendimiento de la clasificación, sin la necesidad de procesar previamente la señal de entrada.

Estas redes son ventajosas para ciertas aplicaciones con respecto a las CNN 2D, debido a la menor complejidad computacional de los filtros de convolución, siendo exponencialmente menor al de las CNN 2D. Normalmente tienen una arquitectura con un número reducido de parámetros³, lo que aumenta la velocidad de la red y de su entrenamiento. También presentan un mejor rendimiento en los casos en los que se dispone de un número reducido de datos etiquetados y altas variaciones de señal, disminuyendo los problemas de sobreajuste⁴ de la red.

Mohamed et al. proponen en [37] (2021) un método de extremo a extremo para la clasificación de sonido ambiental basado en una CNN 1D con optimización bayesiana y aprendizaje de conjunto. Los modelos propuestos aprenden directamente la representación de características de la señal de audio, sin embargo, se agregaron características como espectrogramas de Mel, Log-Mel y MFCC para mejorar su rendimiento. Para recortar los audios de entrada se utiliza una ventana deslizante con una superposición del 50% (Figura 2-5). Los modelos incluyen capas convolucionales 1D, capas de agrupación, de aplanamiento y de neuronas completamente conectadas. La arquitectura del método propuesto está compuesta por un conjunto de 5 modelos diferentes, donde cada uno es entrenado y validado por un pliegue del conjunto de datos UrbanSound8K. Para diferenciar cada modelo y ajustar los hiperparámetros⁵ de las redes se utilizó una optimización bayesiana. Los 5 modelos por separado alcanzaron una precisión media del 89,6%, mientras que el conjunto de los 5 modelos logró una precisión del 94,6%.

Sin embargo, a pesar de las ventajas de utilizar la señal de audio como entrada, la gran eficacia de los modelos CNN 2D en el reconocimiento de patrones y clasificación de imágenes ha hecho que se quieran aplicar también en el campo del reconocimiento de sonido.

Las **CNN 2D** utilizan imágenes como entrada, lo que requiere la transformación de la señal de audio a imagen. Para ello, al igual que con los modelos de clasificación tradicionales, se deben extraer las características del sonido en espectrogramas

² Se explica el concepto de convolución en el apartado 3.1.

^{3, 5} Los parámetros e hiperparámetros se detallan en el apartado 3.2.1.

⁴ En el apartado 3.3. se explica el sobreajuste.

manualmente. Estas representaciones de tiempo-frecuencia son especialmente útiles como características de aprendizaje, debido a la naturaleza no estacionaria y dinámica de los sonidos [36].

Al igual que las CNN 1D, las CNN 2D aprenden los distintos filtros que deben aplicar a las imágenes de entrada para extraer sus características y relacionarlas con su clasificación, con la diferencia de que ahora, estos filtros son matrices de dos dimensiones que se desplazan tanto en el tiempo como en la frecuencia. Estos filtros transforman la imagen de entrada en otra mediante la operación de convolución entre la matriz de la imagen y la del filtro, resaltando ciertas características dependiendo del tipo de filtro. El reconocimiento de patrones se va jerarquizando a medida que se añaden más capas convolucionales. Mientras que los filtros de las primeras capas aprenderán a reconocer patrones simples como bordes o formas, a medida que avanzamos de capa los nuevos filtros aprenderán a reconocer patrones de combinaciones de éstos, siendo capaces de abstraer la información de la imagen de entrada. A medida que se entrena la red, ésta optimiza los valores deben tomar los filtros para reconocer los patrones contenidos en las imágenes.

Normalmente, se añaden capas de *pooling* o agrupación entre las distintas capas convolucionales. Estas capas agrupan los valores de varios píxeles contiguos en un único píxel, reduciendo la dimensionalidad de los datos y, en consecuencia, la velocidad de procesamiento de la red. Una de las más utilizadas son las capas *max-pooling* o de agrupación máxima, las cuales mantienen el valor máximo de los píxeles de alrededor en el nuevo píxel. Al agrupar la información reducir el tamaño de la imagen, también se permite añadir más filtros de convolución en las capas posteriores. La Figura 2-2 muestra como aumenta el número de filtros a medida que se añaden capas convolucionales y de agrupación.

Finalmente, al igual que con las CNN 1D, se añaden capas neuronales completamente conectadas para la clasificación de la imagen de entrada en función de las características extraídas en las capas convolucionales.

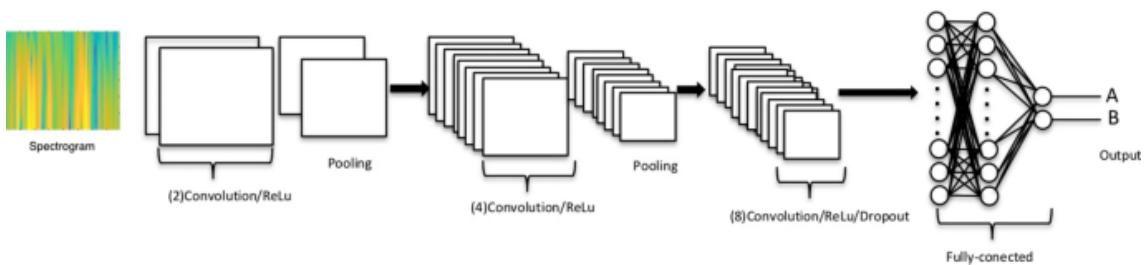


Figura 2-2. Arquitectura CNN con tres capas convolucionales y dos de agrupación [38].

Este modelo fue introducido por primera vez por Piczak et al. en (2015) [39]. Utilizaron el espectrograma log-Mel y delta-log-Mel como entradas a una CNN 2D de

dos canales. La arquitectura de la red se compone de dos capas convolucionales con filtros de 3x3 dimensiones, dos capas de *max-pooling* y dos capas de neuronas completamente conectadas y con un *dropout*⁶ del 50%. Evaluaron la red con una evaluación cruzada de 5 pliegues para los conjuntos de sonidos ambientales ESC-10, ESC-50 y una validación cruzada de 10 pliegues para UrbanSound8K. Obtuvieron una precisión media del 44%, 73% y 68% respectivamente.

Justin Salomón y Juan Pablo Bello proponen en [40] (2017) una arquitectura similar a la anterior, agregando una capa convolucional y de agrupación más, además de una regulación L2⁷ con una tasa de penalización de 0,001 en la capa de neuronas. Representaron las características del sonido en espectrogramas de Mel a escala logarítmica con 128 bandas y un tamaño de ventana de 23ms y evaluaron la red a través de una evaluación cruzada de 10 pliegues, obteniendo una precisión del 73% para el conjunto de sonidos de UrbanSound8K. Tras aplicar un aumento de datos⁸ sobre este conjunto, consiguieron una precisión media del 79%, demostrando la mejora del rendimiento del modelo gracias al conjunto de aumento.

Zhejian Chi et al. [41] (2019) concatenaron dos espectrogramas regulares, el espectrograma log-Mel y el espectrograma Log-Gammatone. La red propuesta se compone de tres bloques, que a su vez se componen de tres capas convolucionales y una capa de agrupación promedio. El método se probó en ESC-50 y UrbanSound8K y logró una precisión de clasificación del 83,8 % y el 80,3 %, respectivamente.

Otros trabajos [42], [43] (2020) han combinado la capacidad de extracción de características de las CNN con otros clasificadores tradicionales. En estos trabajos se entrena una o varias CNN para que aprendan a extraer las características de los espectrogramas de entrada y, una vez entrenadas, se prescinde de las últimas capas totalmente conectadas de la CNN y en su lugar se alimenta a un clasificador como KNN o SVM con los vectores de las características extraídas por las capas convolucionales y de agrupación. Estos modelos obtuvieron unas precisiones comparables a los trabajos anteriores e incluso superiores.

Una de las mayores limitaciones de las CNN es que no preservan la localización espacial de las características aprendidas, lo cual es una consideración importante para espectrogramas o representaciones de tiempo-frecuencia en general. Esto significa que la red, debido a su arquitectura, sólo tiene en cuenta la detección de ciertas características para la clasificación, pero no la localización de dichas características a lo largo de la imagen de entrada. Sin embargo, la ubicación de las características aprendidas es relevante ya que especifica el componente espectral, donde el mismo valor de energía puede referirse a diferentes frecuencias según la posición en la que se detectó [44].

^{6, 7} Estos conceptos se explican en el apartado 3.3.2.

⁸ El aumento de datos se detalla en el apartado 3.3.1.

Las **Redes Neuronales Condicionales** (*Conditional Neural Networks*, CLNN) son un modelo discriminativo diseñado para señales temporales multidimensionales que pretende resolver las limitaciones anteriores. La entrada de esta red es una ventana compuesta por un número determinado de fotogramas, donde la salida para cada fotograma tiene también en cuenta las características de los N fotogramas contiguos (Figura 2-3). La ventana de entrada, por lo tanto, debe tener una dimensión de $2N + 1$ fotogramas. Con cada capa que se añade a la red, el número de fotogramas de salida se reduce en un factor de $2N$. Por lo tanto, el tamaño de la ventana de entrada aumenta de manera considerable a medida que se añaden más capas a la red.

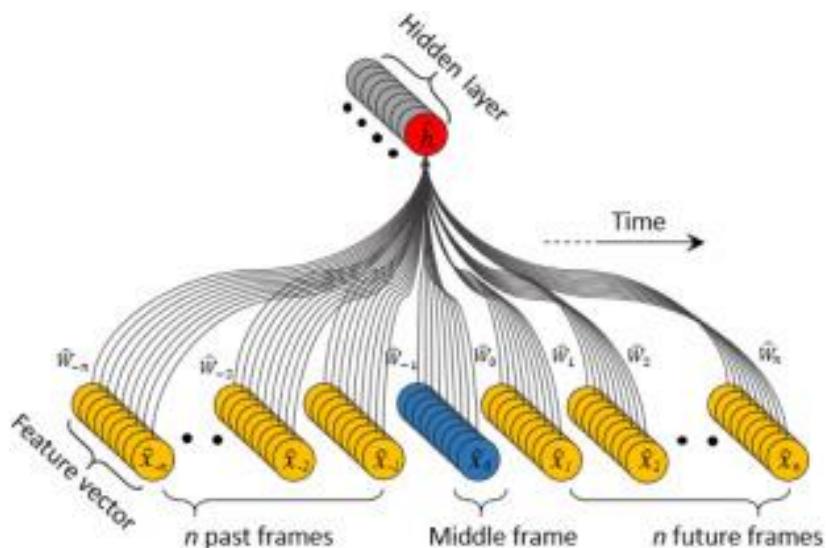


Figura 2-3. Conexiones para un nodo de una capa de CLNN [45].

La **Red Neuronal Condicional Enmascarada** (*Masked Conditional Neural Network*, MCLNN) es una extensión de la CLNN. Estas redes imitan un comportamiento similar al de un banco de filtros a través de una máscara binaria que actúa entre las conexiones de los fotogramas de entrada y las neuronas de la capa oculta (Figura 2-4). Lo que se consigue con esta máscara es discriminar las características de los fotogramas que llega a cada neurona eliminando las conexiones entre ambos o no. De esta manera se dividen las frecuencias del spectrograma de entrada en bandas, similar a lo que haría un filtro paso banda en un banco de filtros.

El diseño de la máscara está controlado por dos hiperparámetros: el ancho de banda, que controla el número de características que se considerarán en la misma banda, y la superposición, la cual controla el número de características que tienen conexión con varias neuronas contiguas.

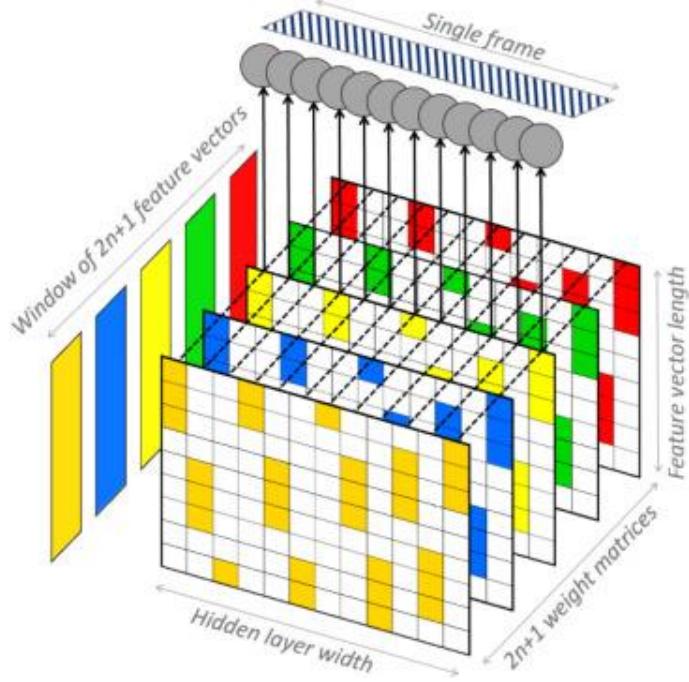


Figura 2-4. Un paso de una MCLNN [45].

En [45] (2020), Fady Methat et al. utilizaron una MCLNN para evaluar su desempeño en la clasificación de géneros musicales y reconocimiento de sonidos ambientales. Para este último, se probaron varias arquitecturas de MCLNN sobre los conjuntos de sonidos ambientales: ESC-10, ESC-50 y UrbanSound8K, obteniendo unas precisiones de hasta el 85,5%, 66,6% y 74,22% respectivamente.

Todas estas redes tienen la limitación común de que requieren de una entrada de dimensiones fijas para su funcionamiento. En el caso de las redes CNN 1D, la longitud de la señal de audio (no la duración) debe ser siempre la misma, al igual que las dimensiones de las representaciones del sonido en forma de espectrogramas para las CNN 2D, CLNN y MCLNN.

Una de las soluciones más empleadas es lo que se conoce como *Windowing* o ventana deslizante. Este proceso consiste en dividir la señal de audio en ventanas o segmentos de un ancho fijo. La longitud de estas ventanas se corresponde con el número de muestras de la señal, que se puede calcular multiplicando la duración del fragmento de audio en segundos por su frecuencia de muestreo en hercios. Además, estas ventanas pueden estar superpuestas unas con otras, compartiendo información de la señal y aumentando el número de fragmentos disponibles.

En los casos en los que se quiere clasificar un audio de mayores dimensiones al de la entrada de la red, éste se fragmenta y se introducen todos los fragmentos por separado al modelo. La clasificación de audio original se determina a partir de las calificaciones obtenidas para los distintos fragmentos evaluados. Para ello, una de las técnicas más utilizadas es la votación por mayoría, que consiste en mantener el

resultado obtenido un mayor número de veces en las clasificaciones de los fragmentos que componen el audio a evaluar.

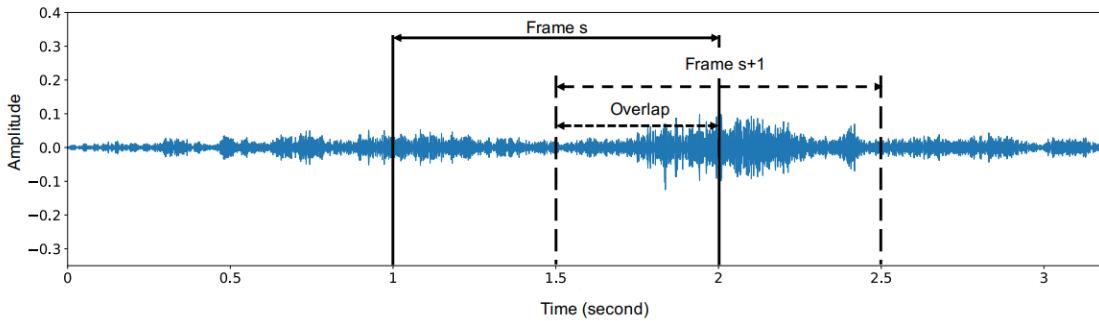


Figura 2-5. Ventana deslizante sobre señal de sonido con superposición del 50% [46].

En las redes vistas hasta ahora, la información viaja de la capa de entrada a la de salida en una única dirección. Las **Redes Neuronales Recurrentes** (*Recurrent Neural Networks*, RNN) rompen con este esquema, incluyendo conexiones en ambas direcciones para analizar datos de series temporales. A diferencia, por ejemplo, de las CNN, cuya clasificación depende exclusivamente de la imagen de entrada y no de las imágenes procesadas anteriormente, las salidas de las RNN dependen también de la información obtenida de las entradas anteriores, capturando así la dependencia temporal entre las distintas entradas.

Las neuronas de las capas de este tipo de red están retroalimentadas, lo que significa que su salida depende de las entradas de la neurona en ese instante y de la salida de los instantes anteriores (Figura 2-6), dotando a la neurona de cierta memoria. Durante el entrenamiento de la red, cada neurona aprenderá en qué medida debe influir cada entrada y salidas anteriores para obtener la salida que permita una clasificación óptima. Sin embargo, un gran inconveniente de este tipo de arquitectura es que el elevado número de parámetros entrenables que requiere cada neurona ralentiza mucho su procesamiento.

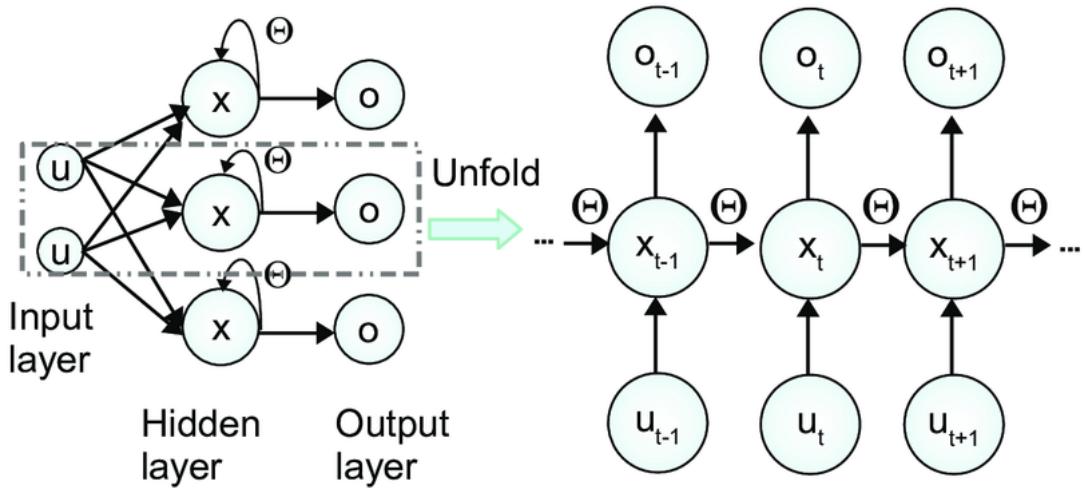


Figura 2-6. Capa de una RNN [47].

Una de las mayores limitaciones de este modelo es que, aunque las RNN tratan de almacenar la información temporal, es bastante difícil propagarla en largos períodos de tiempo. Como las neuronas toman como entradas las salidas anteriores, van arrastrando también sus errores además de los actuales. Cuando una RNN tiene demasiadas capas, esta retroalimentación puede desencadenar dos problemas a la hora de entrenar la red: que los gradientes mediante los cuales se calculan los nuevos pesos para las entradas de las neuronas crezcan excesivamente, lo que se conoce como gradientes explosivos o *Exploding Gradients*, o que estos gradientes se vayan haciendo cada vez más pequeños y la red deje de aprender, llamado gradientes desvanecidos o *Vanishing Gradients*.

La solución para el primer problema es bastante sencilla, basta con truncar los gradientes o reducirlos. Sin embargo, el segundo es más difícil de resolver y ha requerido la elaboración nuevos modelos.

Long-Short Term Memory (LSTM) son una extensión de las RNN en las que se amplía la memoria de la red para recordar sus entradas durante un largo período de tiempo. Cuenta con una serie de “puertas” que se encargan de decidir qué información entra (puerta de entrada) o sale (puerta de olvido) de esta memoria y cuándo se utiliza (puerta de reajuste).

Gated Recurrent Unit (GRU) son otro tipo de RNN más simple que las LSTM y con menos parámetros, consiguiendo un mejor rendimiento en aquellos casos en los que se disponga de un conjunto de datos de entrenamiento limitado. En estos modelos, la puerta de actualización decide la proporción de información que se debe recordar y la puerta de reajuste, la cantidad de información que se debe olvidar.

En (2019) [48] se examina un modelo de RNN para la clasificación de sonidos urbanos. La arquitectura de la red empleada consta de dos capas LSTM con una tasa de abandono del 25%, seguidas de una capa neuronal completamente conectada con

función de activación⁹ *softmax*. Eligieron una función de pérdida¹⁰ de entropía cruzada categórica y el optimizador¹¹ Adam para minimizarla. Como entrada de la red utilizaron el espectrograma de Mel con 128 bandas. La red LSTM propuesta logra un 84,25 % evaluada con una validación cruzada de 5 pliegues del conjunto de datos UrbanSound8K, frente al 81,67% de la CNN con la que la comparan.

Debido a la naturaleza de las muestras de sonido que se quieren clasificar en este proyecto, consideramos que no se necesita de una red especializada en captar dependencias temporales, ya que los fragmentos de audio ambiental son muy cortos y no tienen interrelación temporal entre ellos. Por tanto, la arquitectura empleada en este proyecto para enfrentarnos al problema de la ESC será una CNN 2D, debido a la gran robustez que han demostrado para resolver problemas de clasificación relacionados con el reconocimiento de patrones.

^{9, 10, 11} Estos conceptos se detallan en el apartado 3.2.1.

Capítulo 3

FUNDAMENTOS TEÓRICOS

3.1. CONVOLUCIÓN

La convolución es la operación matemática que permite a las CNN extraer las características de una imagen. Desde el punto de vista digital, una imagen es una matriz de dos dimensiones en la que cada elemento está formado por uno o varios bytes, dependiendo del número de canales por el que esté compuesta la imagen. Por ejemplo, en el caso de una imagen a color el número de canales será de 3: un byte para el color rojo, otro para el verde y otro para el azul, mientras que una imagen en escala de grises tendrá únicamente un canal.

La operación de convolución discreta es una transformación en la que el valor del píxel resultante es una combinación lineal de los valores de los píxeles vecinos en la imagen. Esta transformación se puede definir por dos matrices: la imagen y la matriz de coeficientes, también conocida como máscara de convolución, filtro de convolución o kernel, la cual define los pesos que se aplicarán a cada píxel de la imagen y sus píxeles vecino (Figura 3-1). Los filtros de convolución tendrán unas dimensiones mucho menores a las de la imagen, tomando normalmente valores impares de 3x3, 5x5 o 7x7 para anclar el centro del filtro a cada píxel de la imagen.

Como resultado de este proceso iterativo se obtiene una imagen de menores dimensiones a la original, cuyas características se ven realzadas dependiendo del tipo de filtro utilizado.

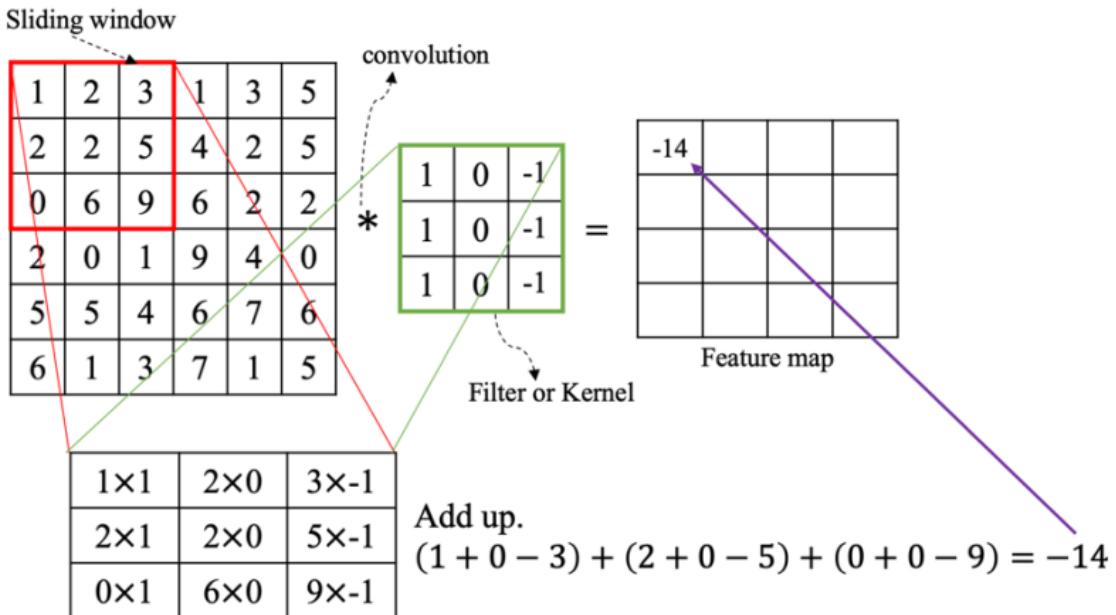


Figura 3-1. Operación de convolución discreta [49].

3.2. REDES NEURONALES CONVOLUCIONALES

Las CNN explotan la operación de convolución en imágenes para extraer las características de las imágenes de entrada. Estas redes tienen una o varias capas convolucionales compuestas por un número determinado de filtros convolucionales. Cada filtro lleva a cabo la operación de convolución sobre la imagen de entrada. Al principio, los pesos de cada filtro tendrán unos valores de inicialización o aleatorios que, a medida que se entrena la red, convergerán en los valores óptimos que permitan detectar las distintas características que permiten clasificar la imagen adecuadamente.

Cada filtro se especializará en detectar una característica concreta y, a medida que se añadan capas convolucionales, las características detectadas por cada capa superior tendrán cada vez una mayor complejidad, ya que utilizan las características más simples extraídas en las capas anteriores.

Normalmente, las capas convolucionales se intercalan con capas de agrupación que reducen la dimensionalidad de las imágenes de salida. Estas capas agrupan los valores de varios píxeles contiguos en un único píxel siguiendo una norma determinada. Una de las más utilizadas son las capas *max-pooling* o de agrupación máxima, las cuales mantienen el valor máximo de un conjunto de píxeles en el nuevo píxel. A medida que se reduce las dimensiones de las matrices de características, se pueden añadir más filtros en la siguiente capa convolucional.

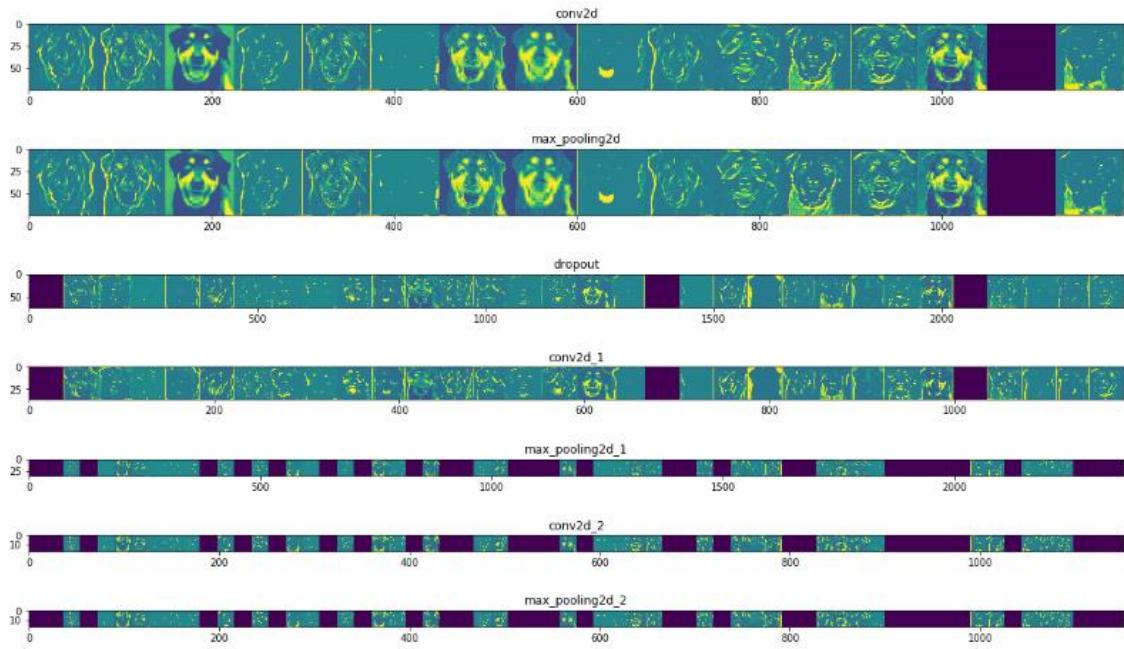


Figura 3-2. Extracción de características de una CNN [50].

En la Figura 3-2 se puede observar el proceso de extracción de características de una CNN a través de las capas de convolución y agrupación. Mientras que algunos filtros se especializan en detectar bordes, otros lo hacen para detectar ciertas texturas o contrastes, obteniendo las representaciones de las distintas características que componen la imagen. A medida que se aplican las capas de agrupación, las dimensiones de la imagen se reducen.

Una vez extraídas todas las matrices de características, éstas se aplanan en un único vector mediante una capa de aplanamiento o *flatten*, o mediante una capa de agrupación global, la cual agrupa todos los píxeles de las imágenes de características en un único valor, como puede ser el valor máximo.

Finalmente, se añaden una o varias capas de neuronas completamente conectadas para llevar a cabo la tarea de clasificación (Figura 2-2). Se llaman así debido a que cada neurona está conectada a todas las salidas de la capa anterior. La salida de cada neurona es el resultado de la suma ponderada de todas sus entradas. La ponderación de cada entrada viene definida por el peso que se le asigna a cada conexión (Figura 3-3). Además de esto, cada neurona tendrá también un término independiente conocido como *bias* o sesgo, actuando como un modelo de regresión lineal. Al resultado de esta operación se le aplica una función no lineal, llamada función de activación, que evita que todo el conjunto de neuronas colapse como si de una única neurona se tratase (debido a que la suma de varias deformaciones lineales es equivalente a una única deformación lineal). A medida que se entrena la red, las neuronas van ajustando estos valores para obtener las salidas que permitan su clasificación final.

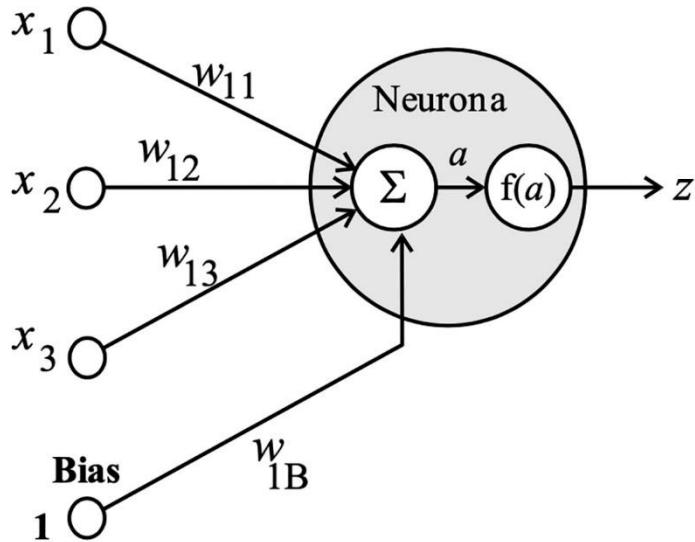


Figura 3-3. Neurona artificial [51].

La última capa neuronal de la red será la capa de salida, la cual debe tener el mismo número de neuronas que clases. La salida de cada neurona será la predicción de la red para su respectiva clase.

3.2.1. ENTRENAMIENTO: PARÁMETROS E HIPERPARÁMETROS

Además de la arquitectura, hay otros dos conjuntos de elementos que definen la configuración de una red neuronal, estos son los parámetros e hiperparámetros.

Los parámetros de una red se corresponden con aquellos valores que definen el comportamiento de ésta. Se pueden diferenciar dos tipos de parámetros: los que son entrenables, como son los pesos y sesgos de las neuronas o los pesos de las máscaras de convolución, y los no entrenables, como algunos valores que se utilizan en capas de normalización¹². Los parámetros entrenables son aquellos cuyo valor se modifica o ajusta durante el entrenamiento, mientras que los no entrenables se mantienen constantes una vez calculados. Estos parámetros se pueden contar en miles para redes simples, millones e incluso miles de millones para los modelos más complejos.

Los hiperparámetros de una red son los parámetros ajustables que permiten controlar el proceso de entrenamiento del modelo. Por ejemplo, en una CNN algunos de estos hiperparámetros pueden ser el número de capas convolucionales de la red, la cantidad o las dimensiones de los kernel, el tipo de agrupación que se aplica, el número de neuronas ocultas, etc. La correcta elección de estos hiperparámetros influirá de

¹² Las capas de normalización se ven en el apartado 3.3.1.

manera significativa en el rendimiento de la red. Desafortunadamente, el proceso de optimización de la configuración de estos hiperparámetros suele ser manual y muy costoso computacionalmente, ya que requiere de prueba y error.

Además de los hiperparámetros que definen la arquitectura de la red, también están los que configuran el proceso de entrenamiento, como son las funciones de pérdida, optimización y activación, la tasa de aprendizaje o el tamaño del lote.

El entrenamiento de una CNN se basa en el aprendizaje supervisado. Es decir, la red necesita un conjunto de datos etiquetados representativo que le permita evaluar sus predicciones para reajustar sus parámetros y conseguir así resultados más precisos. Las funciones de pérdida o de coste son las encargadas de evaluar la desviación entre las predicciones de la red para cada entrada y los valores reales. El error obtenido se tendrá en cuenta para la autocorrección de los parámetros que tuvieron influencia en esa desviación. Algunas de las funciones de pérdida más comunes son el error cuadrático medio, el error absoluto o el error absoluto escalar.

En tareas de clasificación multiclase en las que una entrada sólo puede pertenecer a una de muchas categorías posibles, es común utilizar la función de pérdida de entropía cruzada categórica (*categorical cross-entropy*). Esta función está diseñada para cuantificar la diferencia entre dos funciones de probabilidad, por lo que es adecuada para aquellos casos en los que, aunque la predicción ha sido incorrecta, la probabilidad obtenida se ha quedado cerca del valor real. Además, su derivada es sencilla, lo que facilita los cálculos computacionales y mejora el rendimiento del entrenamiento.

El objetivo del entrenamiento es minimizar la función de coste encontrando los parámetros entrenables adecuados y asegurando, al mismo tiempo, una buena generalización. El reajuste de estos parámetros se lleva a cabo mediante un algoritmo numérico llamado *backpropagation*. La función de optimización es la encargada de generar valores de los parámetros cada vez mejores. Su funcionamiento se basa en calcular el gradiente de la función de coste (derivada parcial) por cada parámetro de la red. Como lo que se quiere es minimizar el error, los parámetros se modificarán en la dirección negativa del gradiente. De cara a agilizar la convergencia de la función de coste hacia su mínimo, el vector de gradiente se multiplica por un factor denominado tasa de aprendizaje (*Learning Rate*) [52].

El descenso de gradiente es un algoritmo iterativo, que comienza desde un punto aleatorio en una función y viaja por su pendiente en pequeños pasos hasta que alcanza un mínimo. Este algoritmo es útil en los casos en los que no se pueden encontrar los puntos óptimos al igualar la pendiente de la función a 0, como es el caso de las funciones de coste en redes neuronales. Sin embargo, tiene la limitación de que puede converger en un mínimo local y no absoluto.

El cálculo de la derivada parcial de la función de coste respecto de cada parámetro de la red para cada entrada del conjunto de datos es inviable debido a la enorme cantidad de ambos. La función Stochastic Gradient Descent (SGD) limita el

cálculo de la derivada a tan solo una observación aleatoria por cada iteración, aunque existen algunas variaciones como mini-batch SGD que seleccionan varias observaciones en vez de una.

La función *Momentum* guarda un registro de la media de los antiguos vectores de descenso del gradiente para acelerar el descenso en aquellas direcciones que son similares a las anteriores.

Las funciones AdaGrad (*Adaptive Gradient Algorithm*) y RMSProp (*Root Mean Square Propagation*) adaptan la tasa de aprendizaje de la red a cada parámetro. En el caso de AdaGrad, las tasas de aprendizaje para cada parámetro se calculan con la raíz cuadrada del sumatorio de los valores anteriores al cuadrado, mientras que RMSProp utiliza una media ponderada exponencial.

Por último, el algoritmo Adam (*Adaptive moment estimation*) es una combinación de las funciones de AdaGrad y RMSProp. Adam mantiene una tasa de aprendizaje para cada parámetro y, además de calcular RMSProp, cada factor de entrenamiento también se ve afectado por la media del *momentum* del gradiente.

El entrenamiento de una red neuronal es un proceso iterativo en el que cada dato del conjunto de entrenamiento pasa varias veces por este proceso de optimización. Se conoce como épocas o *epoch* al número de veces que todo el conjunto de datos pasa por este proceso. Cuando el conjunto de datos es muy grande, se suele dividir en lotes de menor tamaño, también conocido como *batch*. Una iteración sería cada vez que un *batch* pasa por el proceso de optimización. Tanto el número de *epochs* que se entrena la red, como el tamaño del *batch* empleado se consideran también hiperparámetros.

Otro hiperparámetro del que ya se ha hablado en el apartado anterior son las funciones de activación. Estas funciones se corresponden con las deformaciones no lineales que se aplican a las salidas de cada neurona. Su elección tiene un gran impacto en la capacidad de aprendizaje de la red neuronal. En general, una buena función de activación cumple las siguientes características [53]:

- Fuga de gradiente: como se ha visto anteriormente, las redes neuronales se entrenan utilizando el descenso de gradiente y el algoritmo de *backpropagation*, lo que quiere decir que el gradiente de cada capa afecta a la capa anterior. Si el gradiente de la función de activación es cercano a cero, perjudicará el entrenamiento de la red neuronal, pues todos los parámetros de las capas anteriores no se verán casi afectados por la función de optimización.
- Centrado en cero: la función de activación debe ser simétrica en cero, de esta manera, los gradientes no se desplazan hacia una dirección particular.
- Gasto computacional: las funciones de activación se aplican después de cada capa y deben calcularse millones de veces, por lo que una buena función de activación debe ser económica computacionalmente.

- Diferenciable: el cálculo del descenso de gradiente requiere de la derivación de la función de activación, luego necesariamente tienen que ser diferenciables.

Las funciones de activación Sigmoid (*Sigmoid*) y Tangente hiperbólica (*Tanh*) se utilizan para clasificadores binarios. Un inconveniente de estas funciones es que, para valores muy pequeños o demasiado grandes, la derivada converge hacia 0, lo que conlleva un problema de fuga de gradiente. Además, no son muy económicas computacionalmente debido a las operaciones exponenciales.

La diferencia entre estas dos funciones es que Tanh está centrada en cero y tiene un rango de entre -1 y 1, mientras que la función sigmoide está centrada en 0,5 y devuelve valores normalizados entre 0 y 1 (Figura 3-4).

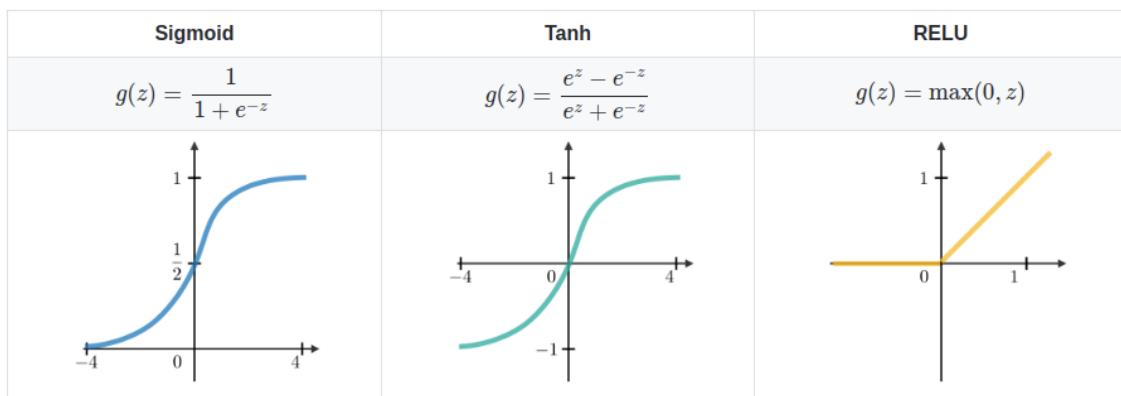


Figura 3-4. Funciones de activación Sigmoid, Tanh y ReLU [54].

La función de activación más popular en el aprendizaje profundo es la ReLU (*Rectified Linear Units*). Esta función descarta cualquier valor negativo y mantiene los positivos. Tiene una mejor propagación del gradiente en comparación con las anteriores. Además, es invariante en escala y es una función muy rápida de calcular. Sin embargo, no está centrada en 0 ni tampoco es diferenciable en 0, aunque sí en el resto de valores. Otro inconveniente que tiene es que, al no tener un límite superior, la salida se puede hacer excesivamente grande, dejando a estos nodos inutilizables.

Finalmente, para clasificaciones multiclase es habitual utilizar la función de activación *Softmax*. Esta función transforma las salidas de una capa neuronal en forma de probabilidades, de manera que el sumatorio de todas las probabilidades de las salidas es 1. Se utiliza en la última capa de clasificación.

3.3. SOBREAJUSTE

Cada vez que se entrena una red neuronal, existe el riesgo de que los parámetros de la red se ajusten demasiado a los datos del entrenamiento. Un buen modelo es aquel que consigue generalizar de manera adecuada la información proporcionada por los datos del conjunto de entrenamiento, de manera que sea capaz de hacer predicciones precisas cuando se introducen nuevos datos que no ha visto antes.

Normalmente, se utilizan dos conjuntos de datos a la hora de entrenar un modelo, estos son el conjunto de entrenamiento y el de validación. El conjunto de entrenamiento contiene los datos con los que se va a entrenar la red, mientras que el conjunto de validación se utiliza únicamente para que el modelo haga predicciones, sin reajustar los parámetros de la red. Los datos de estos conjuntos deben ser independientes y nunca deben mezclarse, ya que validar la red con un dato con el que ha sido entrenada no proporciona una valoración objetiva de su rendimiento.

La manera de evaluar el rendimiento una red neuronal es mediante las curvas de aprendizaje (Figura 3-5). Éstas se componen de la tasa de acierto de las predicciones de la red (*accuracy*) y del valor obtenido de la función de pérdida para cada conjunto (*loss*).

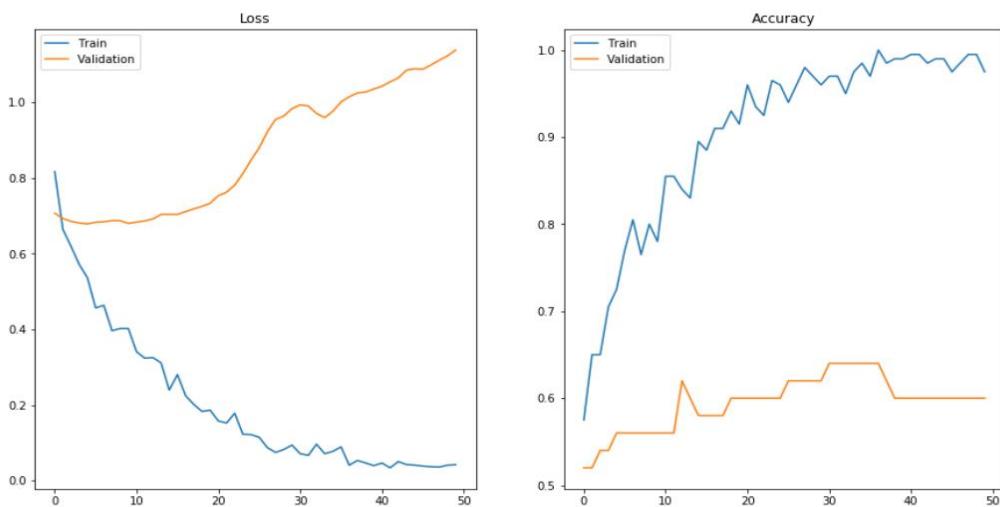


Figura 3-5. Efecto del sobreajuste en las curvas de aprendizaje [55].

Cuando un modelo es demasiado simple, es posible que no tenga los recursos necesarios para ajustarse a los datos de entrenamiento y hacer predicciones precisas. En estos casos, por más que se entrene la red, aunque las curvas de aprendizaje para ambos conjuntos coincidan, ésta no consigue alcanzar una buena precisión para ninguno de los conjuntos, quedándose estancada en un valor intermedio, al igual que la curva de la función de pérdida. Cuando sucede esto, es lo que se conoce como subajuste.

Normalmente, este problema se soluciona aumentando la complejidad de la red. Es decir, añadiendo un mayor número de filtros, neuronas o capas.

No se debe confundir el subajuste con aquellos casos en los que la red no está aprendiendo y obtiene predicciones aleatorias. Cuando hay un problema de subajuste, la red aprende hasta que sus limitaciones lo permiten, y ambas curvas convergen en un valor distinto a la probabilidad de acertar lanzando predicciones aleatorias, mientras que cuando la red no está aprendiendo, los valores de la curva de precisión siempre estarán en torno al valor de dicha probabilidad.

Por el contrario, cuando la tasa de aciertos para el conjunto de entrenamiento es considerablemente mayor que la del conjunto de validación (Figura 3-6), es una señal de que el aprendizaje de la red se está ajustando demasiado a los datos del entrenamiento y no está siendo capaz de generalizar las características de estos datos para hacer predicciones precisas sobre otros nuevos. Este problema es lo que se conoce como sobreajuste. En la Figura 3-6 se muestra un ejemplo de los problemas de subajuste y sobreajuste para un modelo sencillo de clasificación.

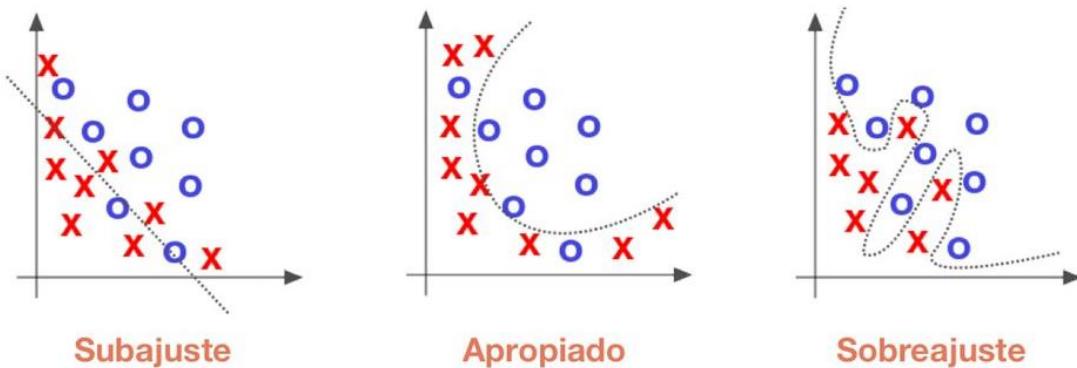


Figura 3-6. Subajuste y sobreajuste de un modelo de clasificación [56].

Los problemas de sobreajuste es algo común a la hora de entrenar una red neuronal. En algunos casos se puede solucionar reduciendo la complejidad de la red, aunque la forma más efectiva de evitarlo es tener un conjunto de entrenamiento lo más amplio y representativo posible. Pero si por lo contrario se cuenta con un conjunto de datos limitado, existen diversas estrategias que permiten generalizar el aprendizaje de la red, como el aumento de datos (*data augmentation*) o las técnicas de regularización.

3.3.1. AUMENTO DE DATOS

Esta técnica consiste en aplicar diversas transformaciones sobre las entradas originales para obtener muestras ligeramente diferentes pero iguales en esencia. De esta manera se puede aumentar en un factor considerable la cantidad de datos disponibles para el entrenamiento.

En este proyecto, las transformaciones se pueden aplicar tanto en el campo del sonido, como en el de la imagen. Algunas de las transformaciones que se pueden aplicar a los audios del conjunto de entrenamiento son:

- Estiramiento de tiempo (*Time Stretching*, TS): ralentiza o acelera las muestras de audio. Esto se consigue aumentando o reduciendo la frecuencia de muestreo de la señal.
- Cambio de tono (*Pitch Shifting*, PS): sube o baja el tono de las muestras de audio.
- Ruido de fondo (*Background Noise*, BG): mezcla la muestra de audio con ruido de fondo. El ruido introducido puede ser tanto una muestra de otras escenas acústicas, como ruido rosa o ruido blanco.

En cuanto a las transformaciones que se pueden aplicar a la representación del sonido en imagen (espectrogramas), estas pueden ser el desplazamiento en los ejes horizontal y vertical de la imagen o el aumento o reducción del *zoom*.

Hay que tener en cuenta que se debe definir un método de relleno de los píxeles que han perdido información, por ejemplo, en los desplazamientos o al reducir el *zoom*. Algunos de los métodos que tienen más sentido utilizar en este campo de aplicación es el relleno por constante, o por el valor del píxel más cercano.

Además, una buena práctica para el correcto entrenamiento de la red es la normalización de los datos de entrada. Esto se hace para evitar que se tengan distancias muy diferentes entre ellos. Por ejemplo, en una imagen a color se pueden tener valores de 0 hasta 255, normalizar estos valores a un rango de 0 a 1 ayudará a la red neuronal a trabajar mejor.

3.3.2. REGULARIZACIÓN

Las técnicas de regularización consisten en limitar el aprendizaje de la red para que los parámetros no se ajusten tanto a los datos de entrenamiento y mejorar su capacidad de generalización. Algunas de ellas son:

- **Regularización L1 y L2:** a medida que el entrenamiento progresaba, algunos pesos se hacen excesivamente grandes para ajustarse cada vez más a los datos de

entrada. Una posible solución es reducir el crecimiento de estos pesos. Esta técnica introduce un término adicional de penalización en la función de coste original que, para el caso de la regularización L1 será proporcional al valor absoluto del peso, y para el caso de L2 será proporcional al cuadrado de este valor.

- **Dropout:** esta técnica consiste en desactivar aleatoriamente un determinado porcentaje de neuronas en cada iteración. Lo que se consigue con esto es que ninguna neurona memorice parte de la entrada y que aprendan todas en conjunto. Una vez que el modelo haya sido entrenado y esté listo para realizar predicciones sobre nuevas muestras, se deben compensar de alguna manera el hecho de que no todas las neuronas permanecieran activas en el entrenamiento, ya que a la hora de predecir sí que estarán todas funcionando y por tanto habrá más activaciones contribuyendo a la salida de la red. Un ejemplo de dicha compensación podría ser multiplicar todos los parámetros por la probabilidad de no descarte [57].
- **Normalización por lotes (*batch normalization*):** cuando se normalizan los datos de entrenamiento, sólo la capa de entrada se beneficia de esto, ya que conforme los datos van pasando por otras capas ocultas, esta normalización se va perdiendo [58]. El método de *batch normalization* normaliza los datos en función de la media y la varianza del lote antes de que pasen por la función de activación en cada capa de la red, de esta manera los datos no toman valores desorbitados.

3.4. CARACTERÍSTICAS DEL SONIDO

El sonido es una onda mecánica longitudinal que se propaga a través de un medio elástico, como es el aire. Se trata de un transporte de energía sin transporte de materia.

Se produce cuando un cuerpo vibra y transmite dichas vibraciones al medio circundante en forma de ondas sonoras. Éstas se desplazan de forma expansiva a una velocidad determinada que depende de las condiciones del medio de propagación (en el aire, en condiciones normales de presión y temperatura, es de aproximadamente 340 m/s.), y pueden ser absorbidas o rebotar en los distintos tipos de superficies que se encuentren a su paso, logrando diferentes efectos de eco o de distorsión [59].

En el aire, el fenómeno de propagación se debe a la puesta en vibración de las moléculas próximas al elemento vibrante, que a su vez transmiten el movimiento a las moléculas vecinas, y así sucesivamente. La vibración de las moléculas de aire provoca una variación de la presión atmosférica, es decir, el paso de una onda sonora por el aire produce una onda de presión. Esta variación de la presión se denomina presión acústica

o presión sonora, y se define como la diferencia entre la presión instantánea y la presión atmosférica en un instante dado [60] (Figura 3-7).

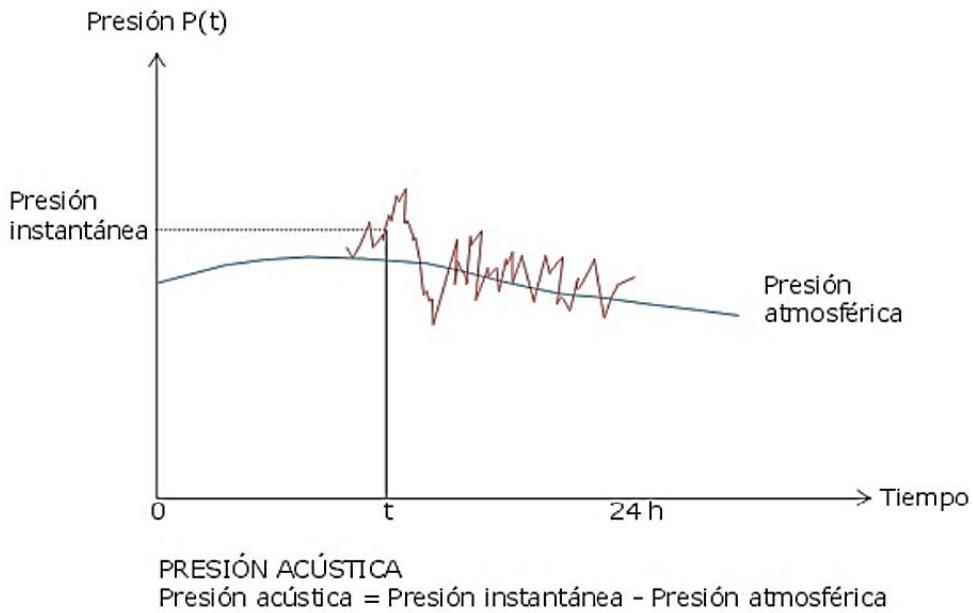


Figura 3-7. Presión acústica [60].

Al igual que cualquier onda en física, el sonido tiene tres características fundamentales: amplitud, frecuencia y composición armónica, lo que en acústica se denomina: intensidad, tono y timbre, respectivamente.

- **Intensidad:** es lo que se conoce como volumen. Hace referencia a la amplitud de la onda sonora y se relaciona con la cantidad de energía transmitida. Se mide en decibelios (dB).
- **Tono:** hace referencia a la frecuencia de la onda sonora. Es el número de oscilaciones por segundo y se mide en hercios (Hz). Los tonos graves se relacionan con las frecuencias bajas y los agudos, con las altas. El ser humano es capaz de escuchar sonidos con tonos comprendidos entre los 20 y 20.000 Hz
- **Timbre:** es la cualidad que permite distinguir dos sonidos de igual frecuencia e intensidad emitidos por distintas fuentes. El sonido, normalmente, no es una única onda de una frecuencia concreta, sino que está compuesto por varias ondas de frecuencias distintas superpuestas. La onda principal se le conoce como fundamental, y el resto de las ondas acopladas tienen el nombre de armónicos [38].

3.4.1. NIVELES SONOROS: EL DECIBELIO

Las presiones acústicas a las cuales es sensible el oído humano varían en un intervalo muy grande. El umbral inferior de la audición humana es de $2 \cdot 10^{-5}$ Pa, mientras que el umbral máximo es de alrededor de 20 Pa. Además, el comportamiento del oído humano se asemeja más a una función logarítmica que a una lineal [60]. Es por estos motivos por lo que resulta más conveniente utilizar una escala logarítmica en lugar de una lineal para medir el nivel de presión sonora.

El nivel de presión sonora L se define por la siguiente expresión:

$$L_p = 10 \cdot \log \frac{p^2}{p_o^2} = 20 \cdot \log \frac{p}{p_o}$$

Figura 3-8. Nivel de presión sonora.

Donde p es la presión acústica de la onda sonora y p_o es el valor de referencia. El nivel de presión sonora L se expresa en decibelios (dB).

Generalmente, se toma el valor de la presión acústica que representa el umbral inferior de la audición humana ($2 \cdot 10^{-5}$ Pa) como valor de referencia. Se habla entonces de decibelios de nivel de presión sonora (Sound Pressure Level, dB SPL). Sin embargo, para sistemas digitales, este valor de referencia es generalmente el valor máximo disponible que puede recoger el dispositivo. Se habla de decibelios a escala completa (Full Scale, dBFS).

En el caso de un micrófono digital, la máxima intensidad sonora que puede medir se corresponderá con los 0 dBFS. Por lo tanto, aquellas intensidades que superen este umbral se registrarán también como 0 dBFS, y las inferiores a él tomarán valores negativos, siendo los más alejados del cero los menos intensos.

En la escala SPL, el sonido audible más bajo se correspondería con un valor de 0 dB SPL, un aumento del doble de la energía de la onda sonora se traduciría en un incremento del nivel de presión sonora de 3 dB, y si ésta aumenta en un factor de 10, sería un incremento de 10 dB en la escala logarítmica.

Para hacerse una idea de los valores de esta escala, la intensidad del sonido percibida en una biblioteca se correspondería a un valor de 40 dB aproximadamente, una conversación normal rondaría los 60 dB, un restaurante ruidoso, en torno a 90 dB, el interior de una discoteca estaría a unos 110 dB y la explosión de un globo, 150 dB.

Sin embargo, la intensidad percibida de las ondas acústicas emitidas por la fuente sonora se va atenuando a medida que aumenta la distancia a ésta. En el caso de una propagación esférica desde una fuente puntual, como puede ser un disparo o el ladrido

de un perro, al doblar la distancia, el nivel de presión sonora disminuye en 6 dB, y en una propagación cilíndrica desde una fuente lineal, como puede ser una carretera, doblar la distancia supone una pérdida de 3 dB [60].

Además de la distancia, hay otros factores que también atenúan la intensidad de la onda sonora, como la absorción del aire. Debido a que el aire no es un gas de densidad homogénea, ni está en absoluto reposo, existe una atenuación debida a la transformación de parte de la energía acústica en calor. Esta atenuación depende de la frecuencia del sonido, de la temperatura y de la humedad del aire. Cuanto mayor es la frecuencia, mayor es la atenuación experimentada [60].

3.4.2. EXTRACCIÓN DE CARACTERÍSTICAS DEL SONIDO

La señal digital de audio se obtiene de la discretización de la onda de presión sonora descrita en los apartados anteriores a una determinada frecuencia de muestreo. El valor de cada muestra se corresponde con la suma de las amplitudes de todos los armónicos que componen la onda en ese instante y, aunque esta representación del sonido tiene algunas ventajas para su tratamiento digital, no aporta demasiada información de las características del sonido.

El conjunto de datos que se empleará para entrenar la red, así como las muestras de sonido que se tomarán posteriormente, tienen un formato de archivo de audio (WAV). Sin embargo, el modelo de CNN que se empleará en este proyecto está diseñado para recibir imágenes como entrada. Esto hace que se requiera de un método que permita extraer las características de la señal de audio y representarlas en una imagen que la red pueda interpretar, esto es, un espectrograma.

Los espectrogramas son una representación visual de las variaciones de frecuencia e intensidad del sonido a lo largo de un periodo de tiempo. Existen diferentes técnicas de extracción y representación las de características del sonido, una de ellas y la que se utilizará en este proyecto es el espectrograma de log-Mel (Figura 3-9).

Para descomponer la señal en cada una de las frecuencias individuales y amplitud que la conforman se debe aplicar la transformada de Fourier [61]. Además, como en un audio hay señales sonoras que varían con el tiempo, se debe aplicar la transformada rápida de Fourier (*Fast Fourier Transform*, FFT), que recoge los espectros de la señal aplicando la transformada de Fourier en segmentos muy cortos del audio superpuestos. De esta manera, se convierte la señal del dominio del tiempo al dominio de la frecuencia, obteniendo lo que se conoce como espectro.

La escala de Mel interpreta las frecuencias de manera similar a como lo hace el oído humano. Tiene como unidad un tono, tal que distancias iguales entre tonos suenan igual de distantes para una persona. Un espectrograma de Mel, por tanto, no es más

que el espectro obtenido de aplicar la FFT a la onda de audio y representado en la escala de Mel.

Para obtener la representación del spectrograma en esta escala, se debe aplicar un determinado número de filtros de Mel. Dependiendo del número de filtros aplicados se obtendrá un determinado número de bandas, lo que está directamente relacionado con la resolución de la matriz obtenida. Si además se representan las intensidades del sonido en escala logarítmica (dB), se obtiene lo que se conoce como spectrograma de Log-Mel. En la Figura 3-9 se muestra un spectrograma de Log-Mel de 128 bandas.

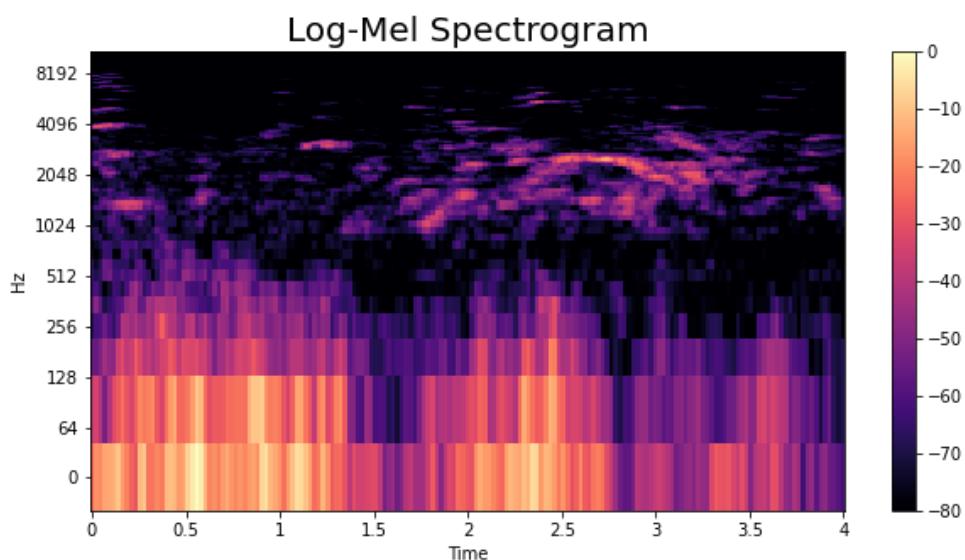


Figura 3-9. Espectrograma log-Mel de 128 bandas.

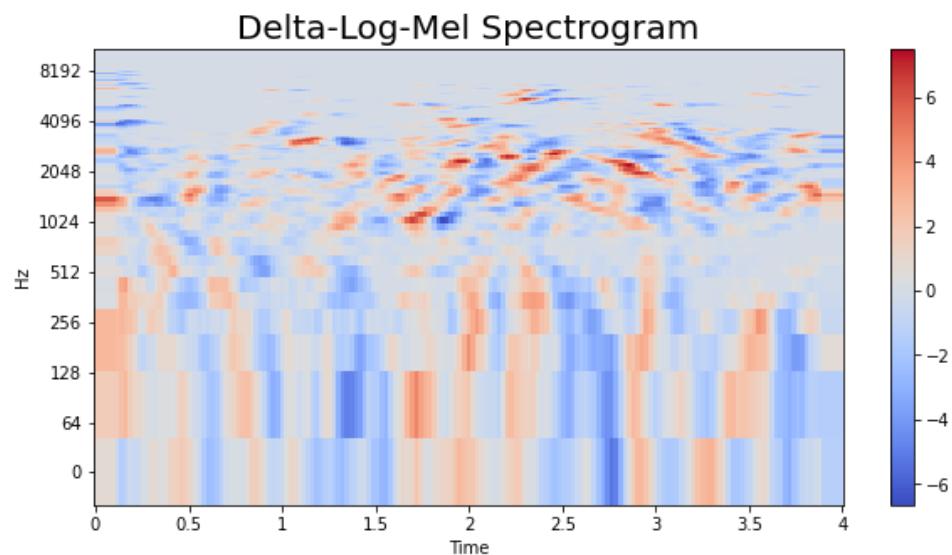


Figura 3-10. Espectrograma delta-log-Mel de 128 bandas.

También existen otras variaciones del spectrograma de Mel, como pueden ser el delta-log-Mel o el delta-delta-log-Mel. Estos spectrogramas representan la dinámica del spectrograma de log-Mel, es decir, son una diferenciación de éste. En la Figura 3-10 se puede ver un spectrograma delta-log-Mel de 128 bandas.

3.5. CLASIFICACIÓN DE MICRÓFONOS

Un micrófono es un transductor acústico-mecánico-eléctrico que transforma las variaciones de presión de una onda acústica en un movimiento mecánico, y éste a su vez genera una señal eléctrica [62]. Se pueden clasificar atendiendo a diferentes criterios:

3.5.1. POR SU PRINCIPIO DE FUNCIONAMIENTO

Según el tipo de transductor del micrófono, éstos se pueden clasificar en micrófonos de carbón, piezoelectrómicos, de cinta, dinámicos o de condensador, siendo estos dos últimos los más utilizados.

- **Micrófonos dinámicos o de bobina móvil:**

Su principio de funcionamiento está basado en la ley de Lenz. Están formados por un diafragma unido a una bobina móvil, la cual se desplaza por las vibraciones de presión a lo largo de un imán fijo (Figura 3-11), produciendo diferencias de voltaje en las puntas de la bobina.

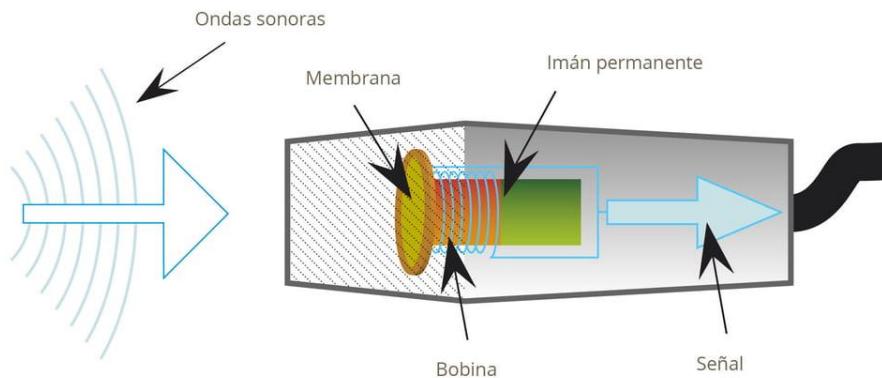


Figura 3-11. Principio de funcionamiento de los micrófonos dinámicos [63].

Los micrófonos dinámicos no necesitan de ninguna alimentación, de hecho, están siempre en funcionamiento, incluso cuando no están conectados, ya que la membrana es sensible a las diferencias de presión en todo momento [64].

Son micrófonos muy duros, con una baja sensibilidad, y se suelen utilizar para grabar sonidos muy potentes.

- **Micrófonos de condensador:**

Se basan en el principio de funcionamiento de los condensadores. Estos micrófonos utilizan dos placas paralelas (Figura 3-12), una de ellas está fija y la otra varía su distancia a ésta en función de los cambios de presión, produciendo fluctuaciones en la capacidad del condensador. La señal eléctrica generada por este método es muy débil y debe ser amplificada. Es por este motivo, que estos micrófonos necesitan para su funcionamiento de una fuente externa, esta fuente es conocida comúnmente como una fuente fantasma (*Phantom Power*), que suele ser de 48 voltios [64].

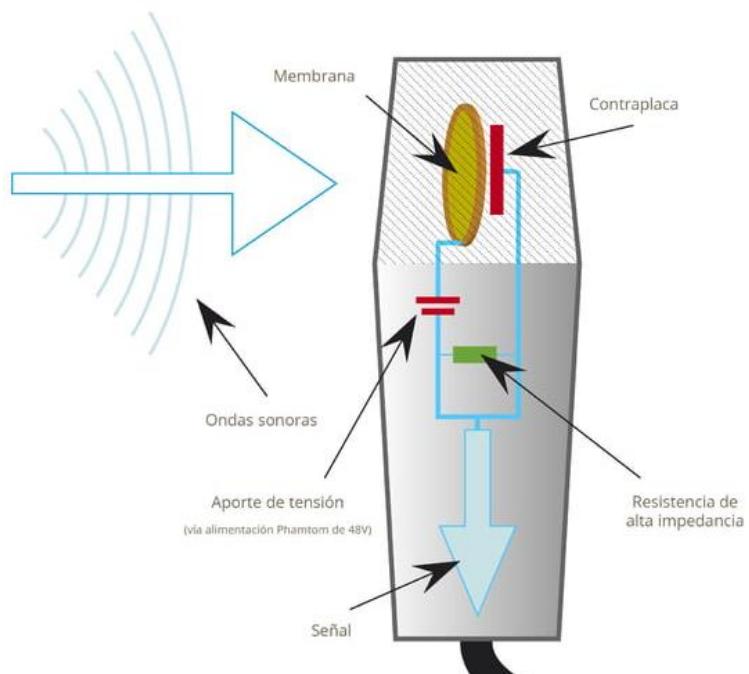


Figura 3-12. Principio de funcionamiento de micrófonos de condensador [63].

Estos micrófonos son mucho más sensibles que los dinámicos, y se utilizan para grabar sonidos muy finos o de ambiente.

3.5.2. POR SU RESPUESTA EN FRECUENCIA

La respuesta en frecuencia de un micrófono representa la variación de la sensibilidad en función del rango frecuencial del sonido. Entendiendo sensibilidad como la relación entre la tensión eléctrica generada por el micrófono y la presión sonora incidente.

Dependiendo de la aplicación para la que se requiera el micrófono, éste tendrá una respuesta en frecuencia ajustada o ajustable, donde se realzan o se atenúan ciertos rangos de frecuencia, o una respuesta en frecuencia plana (Figura 4-4), donde no se atenúa ni se realza ningún rango de frecuencias, obteniendo una señal muy parecida a la original [62].

3.5.3. POR SU DIAGRAMA POLAR

El diagrama polar de un micrófono representa la sensibilidad con la que el micrófono es capaz de captar un sonido según su frecuencia y ángulo con el que incide en él [62]. El diagrama polar de un micrófono puede ser de varios tipos (Figura 3-12 y Figura 4-5).

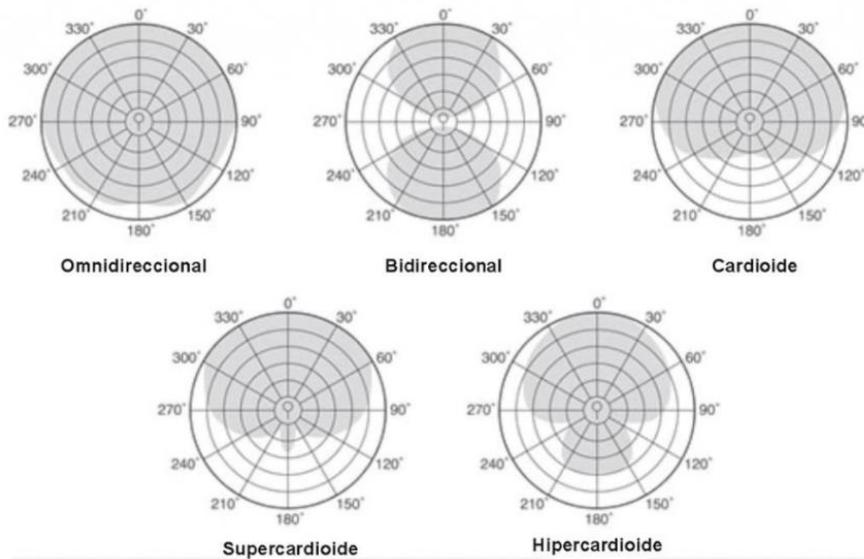


Figura 3-13. Tipos de diagramas polares [65].

Un micrófono con patrón omnidireccional capta por igual el sonido proveniente de todas las direcciones, mientras que uno con patrón cardioide o supercardioide presenta una mayor sensibilidad en su frente, reduciéndose por los lados hasta hacerse nulo o casi nulo en su parte posterior [62].

Capítulo 4

DESARROLLO DEL PROYECTO

4.1. SOFTWARE EMPLEADO

4.1.1. ENTORNOS Y HERRAMIENTAS EMPLEADAS

El desarrollo del proyecto consta de tres partes: 1) implementación de la CNN, 2) implementación del dispositivo de medición y 3) toma de muestras y clasificación. A continuación, se describe el software empleado para las dos primeras partes:

- **Implementación de la CNN**

Para esta parte se ha empleado el Magerit-3, un sistema de supercomputación gestionado por el Centro de Supercomputación y Visualización de Madrid (CeSViMa) [66] y destinado a la ejecución de cargas de trabajo científico que requieran un alto rendimiento. Magerit-3 utiliza el sistema operativo **CentOS**, una distribución basada en Linux.

Todo el código de esta parte se ha desarrollado en el lenguaje de programación **Python**, debido a su sencillez, a su extendido uso en la ciencia de datos y a la gran cantidad de librerías que dispone.

- **Implementación del dispositivo de medición**

Para configurar la recopilación y procesamiento de los audios se ha empleado una Raspberry Pi 4 (RPI) con el sistema operativo **Raspberry Pi OS Full (Raspbian) 32-bit**, que está basado en la distribución de Linux, Debian. El código de esta parte se ha desarrollado en los lenguajes de **Python** y **Bash**.

Para la depuración de código, tanto de Python como de Bash, se ha empleado el entorno **Visual Studio Code** (VSCode) en ambas partes. Además, este programa también se ha utilizado como interfaz para conectarse al Magerit-3.

Para algunas tareas adicionales se ha empleado un ordenador portátil Toshiba Portégé A30-D-C10 con el sistema operativo **Windows 10**.

Para la calibración del micrófono se ha empleado el programa **Rational Acoustics: Smaart v8**.

Para la visualización de la RPI en el portátil, se han empleado los siguientes programas:

Advanced IP Scanner: para escanear la red WiFi e identificar la dirección IP de la RPI.

PuTTY: para realizar la conexión por protocolo SSH (*Secure Shell*) a la RPI.

VNC Viewer: para acceder de forma remota a la interfaz gráfica de la RPI.

4.1.2. LIBRERÍAS UTILIZADAS

Todas las librerías de Python utilizadas este proyecto son de código abierto. Se describen a continuación:

Tensorflow: es una librería diseñada para trabajar con modelos de ML.

Keras: es un marco construido sobre Tensorflow, diseñado para trabajar con redes neuronales de forma sencilla. Se empleará para el diseño, entrenamiento y empleo de la CNN.

Sklearn: es una librería para ML. Se usará para configurar la validación cruzada del entrenamiento de la red.

Librosa: librería para el análisis de audio. Se utilizará para leer archivos de audio y para la extracción de características del sonido.

PyAudio: librería para la reproducción y grabación de audio. Se empleará para grabar las muestras de audio.

Pydub: librería para la manipulación y edición de archivos de audio. Se utilizará para segmentar audios en clips de menor duración.

Numpy: librería para el cálculo de operaciones matemáticas complejas.

PyDrive2: librería que permite la comunicación entre Python y el API de Google Drive. Se utilizará para subir archivos a Google Drive y realizar un seguimiento de la ejecución del código desde cualquier dispositivo.

También se han importado los siguientes módulos de la librería estándar de Python:

Wave: interfaz para archivos con formato WAV. Se utilizará para guardar los audios en este formato.

Os: proporciona funciones para interactuar con el sistema operativo. Se empleará para leer, buscar, crear o eliminar archivos y directorios.

Datetime: permite manejar datos con formato de fecha y hora. Se utilizará para obtener la fecha y hora actuales del sistema y para operaciones de tiempo.

Random: para la generación de números aleatorios.

4.2. HARDWARE EMPLEADO

Para la toma y procesamiento de muestras de audio, se ha implementado un dispositivo electrónico de bajo coste compuesto por: una Raspberry Pi 4, un micrófono y una batería externa opcional. En la Figura 4-1 se muestra su montaje.



Figura 4-1. Dispositivo de medición.

4.2.1. RASPBERRY PI

Como dispositivo programable se ha utilizado una Raspberry Pi 4 Model B Rev1.4 de 4GB de RAM (Figura 4-2). Este dispositivo es un ordenador de bajo coste y formato compacto que permite instalar el software necesario para el desarrollo del proyecto. Requiere de una tarjeta SD para el sistema operativo y almacenamiento y que, para este proyecto, será de 64 GB. Además, se le ha instalado una carcasa protectora, unos disipadores y un ventilador para evitar su sobrecalentamiento.



Figura 4-2. Raspberry Pi 4.

Sus especificaciones son las siguientes:

- **Procesador:** Broadcom BCM2711, quad-core Cortex-A72 (ARM v8), 64-bit SoC @ 1.5GHz
- **Memoria RAM:** 4GB LPDDR4
- **Conectividad:** WiFi Dual Band 2.4 GHz y 5.0 GHz IEEE 802.11b/g/n/ac
Wireless LAN
Bluetooth 5.0, BLE
Gigabit Ethernet

- **GPIO:** Standard 40-pin GPIO header (retrocompatible con las anteriores Raspberry Pi de 40 puertos)
- **Audio y Video:** 2 puertos micro HDMI con soporte 4K 60fps
Puertos MIPI DSI/CSI para cámara y pantalla
Salida jack para audio estéreo y video compuesto
- **Multimedia:** H.265 (4Kp60 decode)
H.264 (1080p60 decode, 1080p30 encode)
Gráficos 3.0 OpenGL ES
- **Soporte SD:** Zócalo Micro SD para almacenamiento y arranque de sistema operativo
- **Alimentación:** 5V DC mediante conector USB-C (recomendado 3A)
5V DC a través de GPIO (mínimo 3A)
Alimentación a través de Ethernet (PoE)-enabled (requiere la instalación del HAT POE)
- **Temperatura de operación:** 0–50°C

4.2.2. MICRÓFONO

Para grabar sonido ambiental se necesita un micrófono que consiga producir una señal lo más parecida al sonido original, que tenga un rango de frecuencia amplio, que no realce ni atenúa algunas frecuencias frente a otras, y que recoja el sonido en todas direcciones.

El micrófono escogido y que cumple todos los requisitos para capturar el sonido ambiental es el Omnitronic MIC MM-2USB (Figura 4-3). Este micrófono de medición tiene respuesta de frecuencia plana (Figura 4-4), patrón polar omnidireccional (Figura 4-5), un rango de frecuencia igual al espectro de audición del ser humano y una gran sensibilidad, permitiendo obtener una señal precisa y fiel a la realidad desde cualquier ángulo.

Este micrófono tiene incorporado un convertidor A/D que permite obtener la señal digital directamente desde el conector USB a la RPI, la cual no cuenta con un

convertidor A/D propio para discretizar por sí misma la señal analógica. Además, se le ha colocado un protector antiviento para reducir el ruido causado por éste y un pequeño trípode para sujetarlo a cierta distancia del suelo.



Figura 4-3. Omnitronic MIC MM-2USB.

Sus características técnicas son las siguientes:

- **Tipo:** Micrófono de medición
- **Tipo de cápsula:** Condensador
- **Patrón polar:** Omnidireccional
- **Rango de frecuencia:** 20-20000 Hz
- **Relación S / N:** > 76 dB
- **Sensibilidad:** -37 dB

- **Max. SPL:** 134 dB
 - **Longitud del cable:** 1,2 m
 - **Impedancia:** 200 ohmios
 - **Puerto USB:** Tipo B
 - **Conexión a PC:** A través de USB
 - **Longitud:** 19,2 cm
 - **Diámetro:** Ø 13 cm
 - **Peso:** 0,18Kg

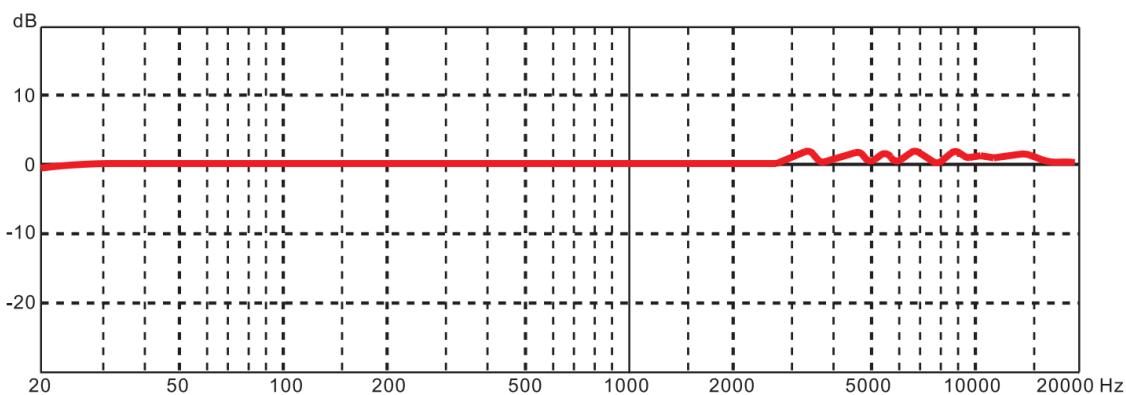


Figura 4-4. Respuesta en frecuencia de Omnitronic MIC MM-2USB [67].

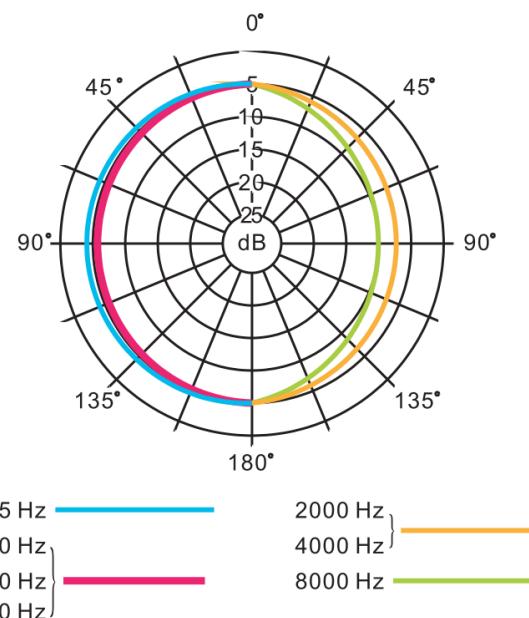


Figura 4-5. Diagrama polar de Omnitronic MIC MM-2USB [67].

4.2.3. BATERÍA EXTERNA

Aunque la RPI puede funcionar conectada a la corriente con su fuente de alimentación oficial, esto limita la utilización del dispositivo a un lugar fijo.

Con el fin de dotar al dispositivo de medición de una mayor portabilidad, se le ha conectado una batería externa como fuente de alimentación (Figura 4-6). Esta batería portátil cumple con los requerimientos de alimentación de la RPI y es totalmente compatible y apta para el trabajo a realizar, dotando de una autonomía de hasta 24 horas al dispositivo de medición.



Figura 4-6. Batería externa ADDTOP 26800mAh.

Sus especificaciones son las siguientes:

- **Batería:** Batería de polímero de litio
- **Puertos:** 3 puertos USB de salida, 1 puerto USB C de entrada/salida
- **Salida:** 5V, 3.0A / 9V, 2.0A / 12V, 1.5A, 18W Máx
- **Entrada:** 5V, 3.0A / 9V, 2.0A, 18W Max

- **Pantalla:** Pantalla LED indicador de batería
- **Dimensión:** 15 x 7.4 x 2.3 cm
- **Peso:** 358.34 g

4.2.4. SONÓMETRO

Aunque no forma parte del dispositivo de medición, se ha empleado un sonómetro VOLTCRAFT Schallpegel-Messgerät SL-200 (Figura 4-7) para la calibración del micrófono, explicado en el apartado 4.5.2.3.



Figura 4-7. Sonómetro VOLTCRAFT Schallpegel-Messgerät SL-200.

Este sonómetro tiene las siguientes características:

- **Cumplimiento de normativa:** EN 60651
- **Batería:** 1 x 9 V (006 P, MN1604)
- **Consumo de corriente:** Aprox. 8 mA
- **Vida de la batería:** aprox. 50 h (batería alcalina)
- **Pantalla:** 3.5-LCD digital
- **Resolución:** 0.1 dB (recarga 0.5 s)
- **Precisión:** + / - 1.5 % (Clase 2)
- **Micrófono:** Micrófono condensador de electreto de media pulgada.
- **Rango de frecuencia:** De 31.5 Hz. a 8 kHz.
- **Rango de nivel de sonido:** De 30 a 130 d B
Lo: De 30 a 100 dB / Hi: De 60 a 130 dB.
- **Valoración de frecuencia:** A y C.
- **Valoración temporal:** RÁPIDA (125 ms) / LENTA (1 s)
- **Condiciones funcionamiento:** Temperatura, de 0°C a 40°C. Humedad relativa del aire, de 10% a 90% (sin condensación).
- **Condiciones almacenamiento:** Temperatura, de -10° C a 60° C. Humedad relativa del aire, de 10% a 75% (sin condensación).
- **Peso (batería incluida):** aprox. 230 g.
- **Dimensiones:** 210 x 55 x 32 (mm)

4.2.5. PRESUPUESTO

A excepción del sonómetro, que fue cedido por el departamento de física de un instituto, el resto de componentes del dispositivo de medición fueron adquiridos

durante el transcurso del proyecto. En la Tabla 4-1 se muestra una tabla con el presupuesto total y desglosado del proyecto:

Componente	Precio/ud	Cantidad	Precio total
Raspberry Pi 4 Model B 4GB RAM	59,50 €	1	59,50 €
Carcasa + ventilador + disipadores	14,19 €	1	14,19 €
Tarjeta SD 64 GB	8,99 €	1	8,99 €
Micrófono USB	45,00 €	1	45,00 €
Trípode para micrófono	9,99 €	1	9,99 €
Batería externa 26800 mAh	36,99 €	1	36,99 €
Total			174,66 €

Figura 4-8. Presupuesto del proyecto.

4.2.6. MAGERIT-3

Tal como se puede leer en su web [68], Magerit-3 consiste en un clúster de propósito general compuesto por 68 nodos *ThinkSystem SD530*, cada uno de ellos equipado con procesadores Intel® Xeon® Gold 6230, 192 GiB de RAM y un disco SSD de 480 GiB. Esta configuración es capaz de proporcionar una potencia pico de 182.78 TFLOPS (DP). Adicionalmente, se dispone de nodos *ThinkSystem SR670* con aceleradores (GPU) específicos. En la Tabla 4-2 se puede ver la configuración de estos nodos.

Cantidad	Procesador	RAM	HD	Aceleradores
68 nodos	2 × Intel® Xeon® Gold 6230 (20 cores @ 2.1 GHz)	192 GiB	480 GiB (SSD M.2)	—
4 nodos	2 × Intel® Xeon® Gold 6240R (24 cores @ 2.4 GHz)	192 GiB	128 GiB (SSD M.2)	4 × NVIDIA A100
2 nodos	2 × Intel® Xeon® Gold 6230 (20 cores @ 2.1 GHz)	192 GiB	128 GiB (SSD M.2)	2 × NVIDIA V100

Figura 4-9. Configuración de nodos que componen Magerit [69].

4.3. DESARROLLO REMOTO CON MAGERIT-3

Como se ha comentado al principio de este capítulo, se ha empleado el sistema de supercomputación Magerit-3 proporcionado por la UPM y gestionado por el CeSViMa para la implementación de la CNN.

Este sistema cuenta con un gran número de nodos, descritos en la Tabla 4-2, que son los que se utilizarán para llevar a cabo esta tarea. Y aunque todos los nodos son idénticos, éstos pueden tener dos funciones muy diferenciadas [69]:

- **Nodos interactivos:** son nodos virtualizados que permiten el acceso a la infraestructura desde cualquier dispositivo y lugar del mundo. Desde ellos se realiza la gestión de trabajos y el intercambio de datos.
- **Nodos de cómputo:** son los nodos en los que se ejecutan los trabajos y se encuentran aislados del exterior. Las ejecuciones en estos nodos se realizan mediante trabajos por lotes *batch* gestionados por el planificador de recursos SLURM.

El acceso a los nodos interactivos se realiza mediante SSH a `magerit.cesvima.upm.es` utilizando las credenciales de usuario que se facilitan con el alta de la cuenta.

Para realizar esta conexión se utilizará el programa de VSCode, aunque antes se deberá tener instalado un cliente SSH compatible con OpenSSH y el paquete de extensión *Remote Development*. Una vez dentro de VSCode, se abrirá la paleta de comandos (Ctrl+Shift+P o Ver>Paleta de Comandos) y se elegirá la opción: “*Remote SSH: Connect to Host*”. Tras añadir el nuevo host mediante el comando “`ssh usuario@magerit.cesvima.upm.es`” y seleccionar Linux como la plataforma del host remoto, la configuración estará lista para poder conectarnos a Magerit-3.

Una vez hayamos iniciado sesión con nuestro usuario y contraseña, tendremos acceso a las carpetas de usuario y proyecto asignado, donde podremos configurar los *environments* con las librerías necesarias para el desarrollo del proyecto, crear y transferir archivos o ejecutar tareas que no requieran un tiempo de ejecución mayor de 10 minutos.

Para simplificar el uso de las aplicaciones instaladas en Magerit, se utiliza el sistema de gestión de aplicaciones Lmod, que es una implementación de *Environment Modules*. Esta utilidad se encarga de preparar el entorno de ejecución para utilizar distintas versiones de las aplicaciones y sus dependencias mediante la carga de módulos de configuración [70].

Para la ejecución de trabajos se deberá crear un archivo con las instrucciones y características que se necesitan. Las directivas de SLURM son comentarios que

empiezan con `#SBATCH` seguido de una serie de parámetros donde se definen cuántos recursos se quieren reservar, están detalladas en [71]. Algunos de ellos son:

- **--job-name**: nombre del trabajo.
- **--time**: define durante cuánto tiempo se ejecutará el proceso.
- **--ntasks**: número de tareas a realizar.
- **--cpus-per-task=<# cpus>**: número de subprocessos de OpenMP.
- **--partition=<nombre>**: aquí especificamos que queremos usar GPU.
- **--gres=gpu:<tipo>:<# gpus>**: aquí especificamos el tipo (a100 o v100) y número de GPU que queremos utilizar.
- **--mem=<#>**: cantidad de RAM necesaria para que se ejecute el proceso.
- **--mail-user**: dirección de correo donde notificar los eventos del trabajo.
- **--mail-type**: tipos de eventos a notificar.

En la Figura 4-8 se muestra un ejemplo de un trabajo en Magerit-3.

```
new > scripts > $ job_train_cnn.sh
 1  #!/bin/bash
 2  ##----- Start job description -----
 3  #SBATCH --job-name=train_cnn
 4  #SBATCH --time=24:00:00
 5  #SBATCH --ntasks=1
 6  #SBATCH --cpus-per-task=2
 7  #SBATCH --partition=standard-gpu
 8  #SBATCH --gres=gpu:v100:1
 9  #SBATCH --mem=8G
10  #SBATCH --mail-user=a.vellella@alumnos.upm.es
11  #SBATCH --mail-type=ALL
12  ##----- End job description -----
13  cd /home/u828/u828568/tfg/us8k_p
14  module purge && module load Python/3.9.5-GCCcore-10.3.0
15  source .virtualenvs/deep_learning/bin/activate
16  srun python /home/u828/u828568/tfg/scripts/train_model1.py
```

Figura 4-10. Trabajo para ejecutar un *script* de Python.

SLURM proporciona una serie de comandos que permiten controlar los trabajos en ejecución [71]:

- **sbatch**: envía un trabajo al sistema devolviendo su identificador.
- **squeue**: lista los trabajos que se están ejecutando o esperando para ejecutar.
- **scancel**: cancela un trabajo encolado. Si el trabajo ya estaba ejecutando se abortará la ejecución en ese momento.

Una vez se ha lanzado el trabajo, el sistema se encargará de reservar los recursos necesarios y ejecutar las tareas en función de su prioridad y los recursos disponibles. En la Figura 4-9 se muestra un ejemplo del lanzamiento y listado de trabajos.

```

● [u828568@login3 outs]$ sbatch ./scripts/job_train_cnn.sh
Submitted batch job 335128
● [u828568@login3 outs]$ squeue
      JOBID PARTITION      NAME      USER ST       TIME  NODES NODELIST(REASON)
      335128 standard- train_cn  u828568 PD      0:00      1 (Resources)
      335001 standard- train_cn  u828568 R   23:55:02      1 r2n6
      335100 standard- train_cn  u828568 R   5:26:56      1 r2n5
      335099 standard- train_cn  u828568 R   9:27:50      1 r2n5
      335127 standard- train_cn  u828568 R    4:19      1 r2n6
○ [u828568@login3 outs]$ █

```

Figura 4-11. Lanzamiento y listado de trabajos en Magerit-3.

4.4. CONJUNTO DE DATOS

El conjunto de datos de sonidos urbanos que se ha utilizado en este proyecto es el de UrbanSound8K (US8K). Este conjunto de sonidos urbanos es de libre acceso y se puede descargar desde su página web¹³. Contiene 8.732 audios en formato WAV, con una duración máxima de 4 segundos etiquetados en 10 categorías:

- Aire acondicionado (1000 audios)
- Claxon (429 audios)
- Niños jugando (1000 audios)
- Ladrido de perro (1000 audios)
- Taladradora (1000 audios)
- Motor (1000 audios)

¹³ <https://urbansounddataset.weebly.com/urbansound8k.html>

- Sirena (929 audios)
- Música callejera (1000 audios)
- Martillo neumático (1000 audios)
- Disparo (374 audios)

La duración total del conjunto de datos es de alrededor de 8,75 horas. En la Figura 4-10 (a) se muestra la duración total por clase con un desglose por prominencia, donde el color rojo se corresponde con sonido en primer plano (*foreground*, FG) y el amarillo con sonido en segundo plano o de fondo (*background*, BG). En la Figura 4-10 (b) se muestra el número total de audios por clase con el mismo desglose anterior.

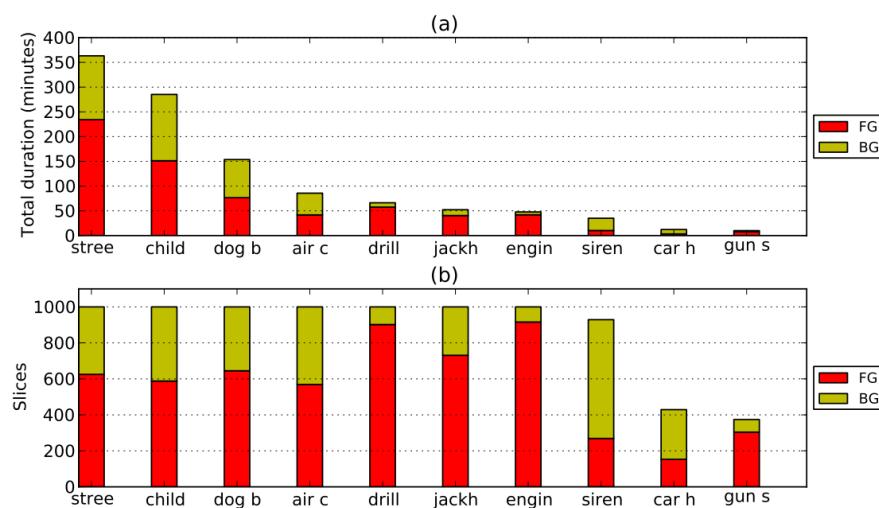


Figura 4-12. (a) Duración total por clase. (b) Número total de muestras por clase. Desglose por audio en primer plano (FG) y segundo plano (BG) [33].

Las clases de este conjunto de datos se extraen de la taxonomía de sonidos urbanos propuesta en [33]. Todos los audios fueron extraídos de Freesound¹⁴, un repositorio de audio online con más de 500.000 audios subidos por millones de usuarios y de acceso libre para uso no comercial. Esto hace que muchos audios tengan distintas frecuencias de muestreo, número de canales y profundidad de bits. El proceso de elaboración del conjunto de datos se describe en [33].

Los audios están agrupados en 10 carpetas para evaluar la arquitectura de la red con una validación cruzada de 10 pliegues. Todos los clips que provienen del mismo audio original están en la misma carpeta. De esta manera se evita que segmentos del mismo audio se utilicen tanto en el entrenamiento como en la validación, lo que daría lugar a resultados inflados que no representan el rendimiento real del modelo.

¹⁴ <https://freesound.org>

Además de los extractos de sonido, US8K también proporciona un archivo CSV que contiene metadatos sobre cada clip de audio (Figura 4-11), donde se incluye:

- ***slice-file-name***: el nombre del archivo de audio, que toma el siguiente formato: [fsID]-[classID]-[occurrenceID]-[sliceID].wav.
- ***fsID***: el ID de Freesound de la grabación de la que se tomó el clip de audio.
- ***start***: el tiempo de inicio del segmento en la grabación original de Freesound.
- ***end***: el tiempo de finalización del corte en la grabación original de Freesound.
- ***salience***: clasificación subjetiva, por escucha, que indica el plano en el que se encuentra la clase identificada dentro del fragmento: 1 = primer plano, 2 = fondo.
- ***fold***: el número de la carpeta a la que pertenece el clip.
- ***classID***: identificador numérico del 1 al 10 que identifica la clase a la que pertenece el clip de audio.
- ***class***: el nombre de la clase a la que pertenece.

slice_file_name	fsID	start	end	salience	fold	classID	class
100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
100263-2-0-143.wav	100263	71.500000	75.500000	1	5	2	children_playing
100263-2-0-161.wav	100263	80.500000	84.500000	1	5	2	children_playing
100263-2-0-3.wav	100263	1.500000	5.500000	1	5	2	children_playing
100263-2-0-36.wav	100263	18.000000	22.000000	1	5	2	children_playing
100648-1-0-0.wav	100648	4.823402	5.471927		2	10	1 car_horn

Figura 4-13. Metadatos contenidos en UrbanSound8K.csv.

4.5. ETAPAS DEL PROYECTO

El desarrollo de este proyecto se compone de tres partes: 1) implementación de la CNN, 2) implementación del dispositivo de medición y 3) toma de muestras y clasificación.

Para llevar a cabo la implementación de la CNN, primero será necesario hacer un preprocesamiento del conjunto de datos que se utilizará para entrenar la red. Seguidamente, se procederá al diseño de la arquitectura de la red neuronal, se definirán sus parámetros e hiperparámetros y se entrenará la red hasta obtener el modelo con la mejor precisión posible.

Para la implementación del dispositivo de medición, lo primero será configurar la RPI desde cero y programarla después para la toma automática y procesamiento de muestras de sonido. También será necesario calibrar el micrófono de medición.

Finalmente, una vez completadas ambas primeras partes, se llevará a cabo un experimento real en el que se tomarán muestras reales de sonido ambiental en dos centros universitarios de la UPM para analizar la contaminación acústica.

Las etapas seguidas en el desarrollo del proyecto se muestran en el diagrama de flujo representado en la Figura 4-12.

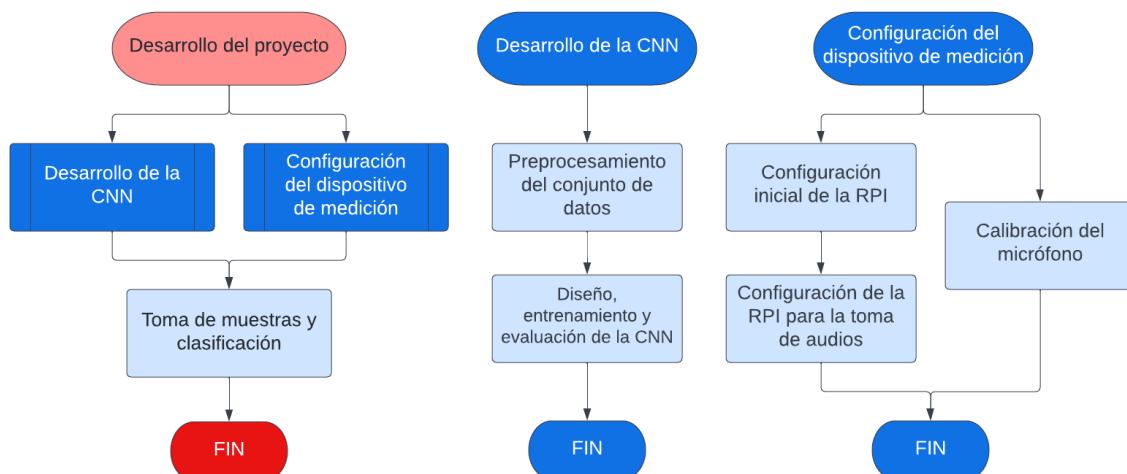


Figura 4-14. Diagrama de flujo de las etapas del proyecto.

4.5.1. DESARROLLO DE LA CNN

4.5.1.1. PREPROCESAMIENTO DEL CONJUNTO DE DATOS US8K

El preprocesamiento del conjunto de datos consiste en adecuar los datos brutos a la entrada del modelo. Constará de dos procesos: 1) aumento de datos y normalización de la duración y 2) transformación de la señal de sonido a imagen.

Un inconveniente del conjunto de datos es que, aunque todos los archivos vienen en formato WAV, no todos están grabados con la misma resolución, ni tienen la misma frecuencia de muestreo, ni tampoco el mismo número de canales (mono o estéreo). Aunque se puede hacer de forma manual con otras librerías, librosa es una librería de procesamiento de audio que incluye una función para leer archivos de formato WAV que normaliza la señal entre valores de -1 y 1. Además, también establece una frecuencia de muestreo de 22050 Hz por defecto y unifica las señales estéreo en un único canal. De esta manera, todas las señales obtenidas tienen el mismo formato.

Para el aumento de datos se ha utilizado el módulo *librosa.effects* para realizar las siguientes modificaciones sobre cada una de las señales obtenidas:

- Estiramiento de tiempo (TS): se ha modificado el tiempo de muestreo en los factores de 0,9 y 1,1 con *librosa.effects.time_stretch*.
- Cambio de tono (PS): se ha aumentado y reducido el tono de cada muestra en 1, 2 y 3 semitonos con *librosa.effects.pitch_shift*.
- Ruido de fondo (BG): se ha mezclado cada audio con ruido blanco (incluso si se ha añadido silencio) con una función hecha a mano que utiliza números aleatorios.

Cada modificación se ha hecho por separado, en ningún caso se ha generado ninguna muestra con varias deformaciones al mismo tiempo.

Una vez aplicadas las deformaciones, se ha normalizado la duración de cada audio en un tiempo de 4 segundos, recortando la duración o añadiendo silencio al final según requiriese. Además, para audios con una duración inferior a 3,25 segundos, se han creado muestras adicionales con silencios al principio de la muestra. Esto se hace para que el modelo aprenda que el evento sonoro puede aparecer tanto al comienzo como al final de la muestra. En la Figura 4-15 se muestra el flujo de trabajo del algoritmo diseñado para el aumento y normalización de los datos (Anexo A, apartado A.3).

Tras aplicar el aumento de datos, el número final de muestras disponibles para el entrenamiento ha ascendido a un total de 110.495 audios frente a los 8.732 originales, aumentando en un factor de más de 12.

En los apartados 2.2 y 2.3 del capítulo del Estado del arte hemos visto cómo muchos de los trabajos [34], [37], [39], [40], [41] consideran el espectrograma de log-Mel como una representación adecuada de las características del sonido. Es por ello que, tras estudiarlo entre otras alternativas, se ha decidido utilizar esta representación como entrada de nuestra CNN.

Para la extracción de las características de la señal de sonido se han empleado las funciones *librosa.feature.melspectrogram* y *librosa.amplitude_to_db*. La primera de ellas obtiene el espectrograma de Mel de la señal de sonido y la segunda convierte su amplitud a escala logarítmica, obteniendo como resultado un espectrograma de log-Mel. Basándonos de nuevo en trabajos anteriores, hemos seleccionado un número de

bandas de 128, ya que consideramos que no hace falta una mayor resolución para la correcta clasificación de los audios. De esta manera se mejora también la velocidad de la red, al tener que procesar una imagen más pequeña que si hubiésemos elegido un número mayor de bandas.

Adicionalmente, se ha empleado la función *librosa.feature.delta* para obtener también el espectrograma delta-log-Mel de 128 bandas a partir del espectrograma de log-Mel. En la Figura 3-9 y Figura 3-10 se muestran dos ejemplos de los espectrogramas log-Mel y delta-log-Mel respectivamente obtenidos. En la Figura 4-16 se muestra el flujograma del proceso de extracción de características (Anexo A, apartado A.3).

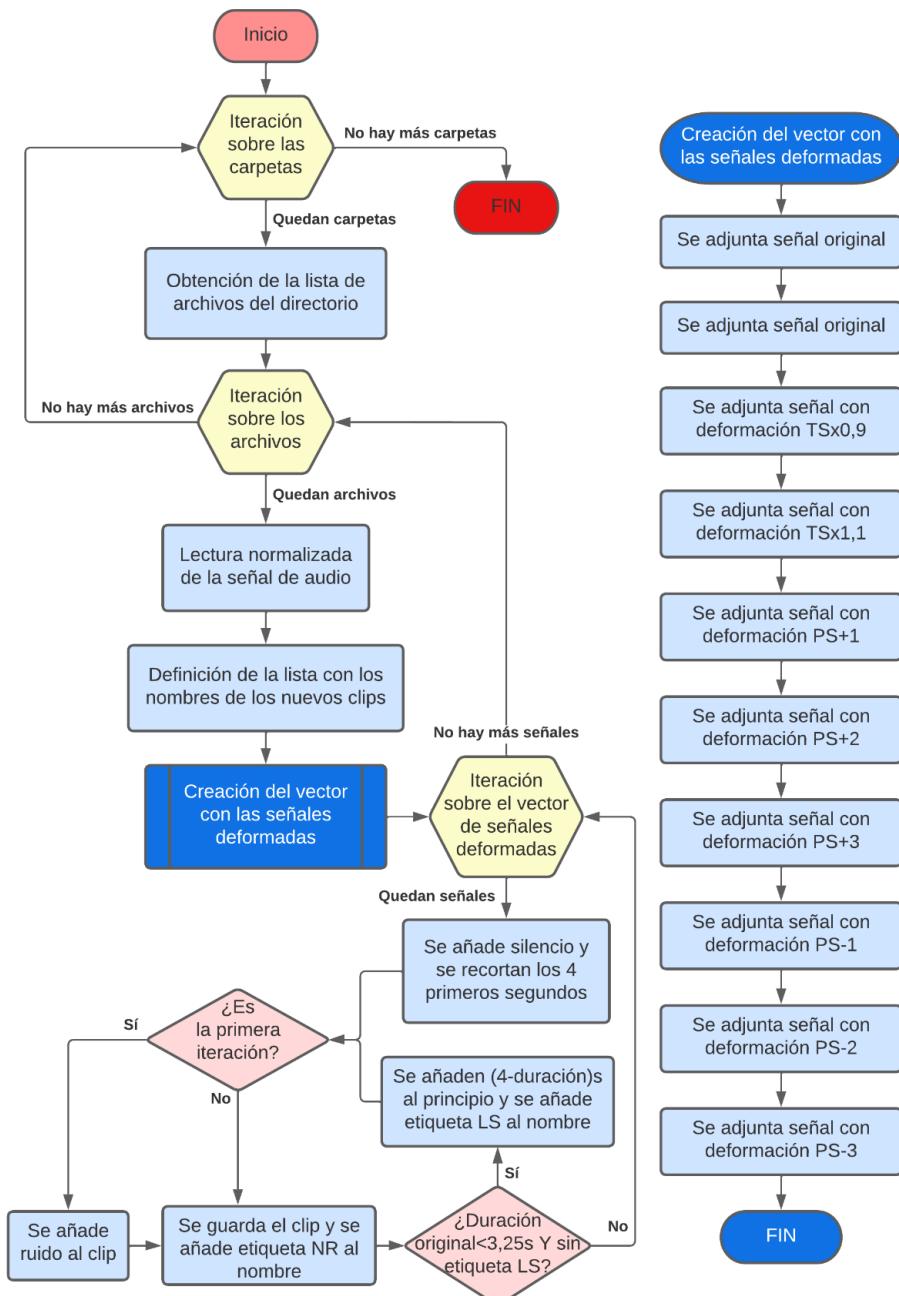


Figura 4-15. Diagrama de flujo del script de aumento de datos.

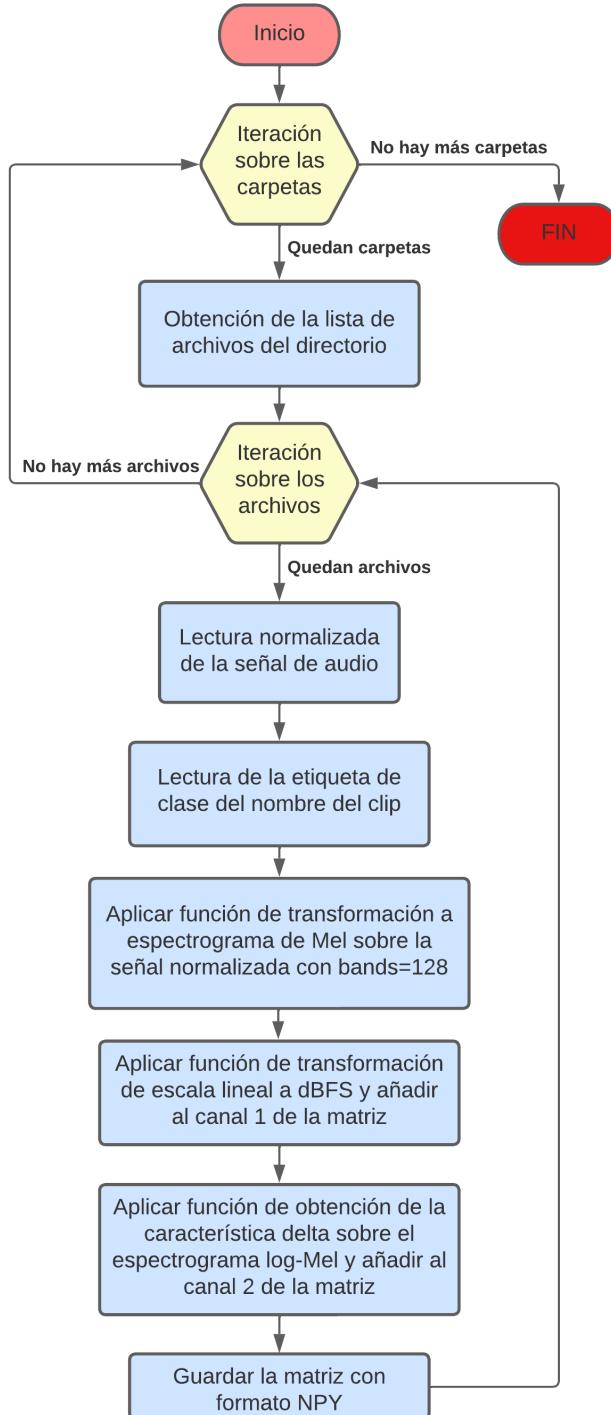


Figura 4-16. Diagrama de flujo del *script* de preprocesamiento.

Las matrices de características generadas para cada muestra se han guardado de forma conjunta en un archivo NPY a través de la función *numpy.save* de la librería NumPy. Además, de cara a facilitar el proceso de entrenamiento de la CNN, el nombre de cada archivo está compuesto por un identificador de la muestra original de la que proviene, otro identificador único para cada nueva muestra generada y el número de la clase a la que pertenece.

4.5.1.2. DISEÑO, ENTRENAMIENTO Y VALIDACIÓN DE LA CNN

Para el desarrollo de la CNN se comenzará proponiendo un modelo inicial, el cual se entrenará y validará. Tras evaluar los resultados obtenidos en el entrenamiento, se decidirán los cambios apropiados para mejorar el modelo, que habrá que aplicar en la siguiente iteración. Este proceso se repetirá hasta obtener el modelo final que mejor se adapte a nuestros objetivos.

4.5.1.2.1. MODELO INICIAL

Para el diseño y la elección de los hiperparámetros de la CNN, nos hemos basado en las configuraciones propuestas en los trabajos del Estado del arte.

• Arquitectura

Para la arquitectura de nuestra red, decidimos utilizar 4 capas de convolución seguidas de capas *max-pooling*, ya es una de las configuraciones que mejores resultados han obtenido en la literatura. Para la capa de entrada configuraremos una cantidad de 32 filtros, que se irá multiplicando por un factor de 2 a medida que se añaden más capas convolucionales, obteniendo un total de 32, 64, 128 y 256 filtros respectivamente para cada capa convolucional.

Teniendo en cuenta que las matrices obtenidas en el preprocesamiento (que serán la entrada de la red) tienen unas dimensiones de 128x173x2 (nº bandas x nº muestreos x nº canales), consideramos utilizar unas dimensiones de 3x3 para los filtros de convolución y de 2x2 para las agupaciones máximas. Ya que, si se hacen más grandes, correremos el riesgo de que las imágenes obtenidas después de cada agrupación sean demasiado pequeñas como para representar correctamente las características de las matrices de entrada, afectando a la siguiente capa y a la clasificación final.

A continuación, añadiremos una capa *flatten* tras la última capa de *max-pooling* para aplanar las matrices de características finales obtenidas en un vector unidimensional, el cual servirá como entrada de las siguientes capas de neuronas.

Finalmente, añadiremos una capa oculta de 256 neuronas completamente conectadas y una capa de salida de 10 neuronas que se corresponderán con las clases del conjunto de datos.

La CNN tendrá un total de 5.633.834 parámetros entrenables y ninguno no entrenable. En la Figura 4-17 se puede ver un resumen de la arquitectura del modelo inicial.

```

model = get_network()
model.summary()

learning rate: 0.0001
Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)        (None, 128, 173, 32)      608
max_pooling2d (MaxPooling2D) (None, 64, 86, 32)      0
)
conv2d_1 (Conv2D)       (None, 64, 86, 64)      18496
max_pooling2d_1 (MaxPooling2D) (None, 32, 43, 64)      0
conv2d_2 (Conv2D)       (None, 32, 43, 128)     73856
max_pooling2d_2 (MaxPooling2D) (None, 16, 21, 128)      0
conv2d_3 (Conv2D)       (None, 16, 21, 256)     295168
max_pooling2d_3 (MaxPooling2D) (None, 8, 10, 256)      0
flatten (Flatten)       (None, 20480)           0
dense (Dense)          (None, 256)            5243136
dense_1 (Dense)         (None, 10)             2570
=====
Total params: 5,633,834
Trainable params: 5,633,834
Non-trainable params: 0

```

Figura 4-17. Resumen del modelo inicial.

• Hiperparámetros

Como funciones de activación de las capas de entrada y ocultas se ha elegido la función ReLU, ya que es la que mejor funciona en tareas de clasificación, además de que previene el problema de desvanecimiento de gradiente en las primeras capas de la red. Para la capa de salida, sin embargo, se ha utilizado la función de activación *softmax* para obtener las probabilidades correspondientes a cada clase.

Como función de pérdida se ha utilizado la función de entropía cruzada categórica, la cual valora la precisión de las probabilidades obtenidas y no sólo que la predicción sea correcta o no, siendo la mejor opción para tareas multiclase. Y como función de optimización, se ha elegido Adam, con una tasa de aprendizaje de 0,0001.

- **Entrenamiento**

Una vez definidos los hiperparámetros, se puede comenzar con el entrenamiento de la CNN. Sin embargo, todavía nos queda un problema por resolver, y es que utilizar el conjunto de datos entero como entrada de la red es inviable debido al gran tamaño que tiene. Para ahorrar memoria RAM y mejorar el rendimiento del entrenamiento, la forma más eficiente de entrenar la red es mediante la división del conjunto de datos en pequeños lotes (*batch*).

Para entrenar la red de esta manera hay que definir un generador de lotes, que será básicamente una instancia de un objeto que nos devolverá un lote del conjunto de datos cuando la función de entrenamiento lo llame. Para ello, nos hemos inspirado en el ejemplo detallado publicado en [72] y lo hemos implementado en nuestro *script* de Python con algunas modificaciones. Este generador se utilizará para obtener tanto los *batch* de entrenamiento, como los de validación. Se ha definido un tamaño de *batch* de 32 para ambos y un total de 26 *epochs*.

Para evaluar la arquitectura de la red, se realizará una validación cruzada de 10 pliegues, siguiendo las recomendaciones de US8K [73]. La validación cruzada consiste en la división del conjunto de datos disponible en partes iguales para realizar un entrenamiento por cada división del conjunto de datos. De esta manera, si el conjunto de datos se ha dividido, por ejemplo, en 10 partes (como es nuestro caso), se entrenará la red con 9 de las divisiones y se validará con la división restante y así hasta validar la red con todas las divisiones un total de 10 veces. La *accuracy* final de la red se obtendrá con la media de las *accuracies* obtenidas para cada pliegue. Con esto se consigue una validación robusta de la arquitectura de la red, además de que nos permite comparar nuestro proyecto con los de la literatura.

Para el modelo inicial, se utilizará el conjunto de datos de US8K sin aumento. Es decir, tanto el conjunto de entrenamiento como el de validación estarán conformados por los audios originales de US8K, sin ningún tipo de modificación. Aunque en aquellos audios con una duración inferior a 4 segundos, se ha añadido silencio hasta completar dicha duración. Además, en aquellos cuya duración es menor a 3,25 también se les ha añadido silencio al principio.

Durante el entrenamiento de la CNN es posible hacer llamadas a ciertas funciones al final de cada *epoch*, denominadas *callbacks*. Hemos definido un total de 3 *callbacks*:

El primero de ellos nos permitirá guardar los valores de las curvas de aprendizaje en un archivo CSV, incluyendo para cada *epoch* los valores de *accuracy* y *loss* de los conjuntos de entrenamiento y la validación.

El segundo *callback* guardará el modelo obtenido en cada *epoch* en formato HDF5, siempre y cuando la *accuracy* del conjunto de validación mejore con respecto al modelo guardado anterior. Estos modelos se podrán cargar después para hacer predicciones sobre otras muestras.

Por último, hemos añadido un *callback* que nos permite realizar un seguimiento en tiempo real de la situación del entrenamiento y visualizar las curvas de aprendizaje de todos los modelos. Para ello hay que escribir el comando “`tensorboard --logdir <logs>`” en el terminal de la carpeta que contenga el directorio definido previamente donde se guardarán los registros del entrenamiento.

El entrenamiento de la CNN se ha desarrollado utilizando la librería para DL de Keras. Esta librería proporciona un espacio de trabajo de alto nivel para el desarrollo de redes neuronales.

4.5.1.2.2. RESULTADOS DEL MODELO INICIAL

Tras entrenar la red por un total de 25 *epochs*, la *accuracy* máxima para el conjunto de validación de cada pliegue ha resultado tener un valor medio de 74,68%, y de 1,76 para los valores de *loss* correspondientes a dichas *accuracies*. Sin embargo, los valores obtenidos para el conjunto de entrenamiento llegan prácticamente a 1 y 0 respectivamente. En las Figuras 18 y 19 se muestran las curvas de aprendizaje obtenidas para cada pliegue. El nombre de cada pliegue estará determinado por la carpeta de validación utilizada.

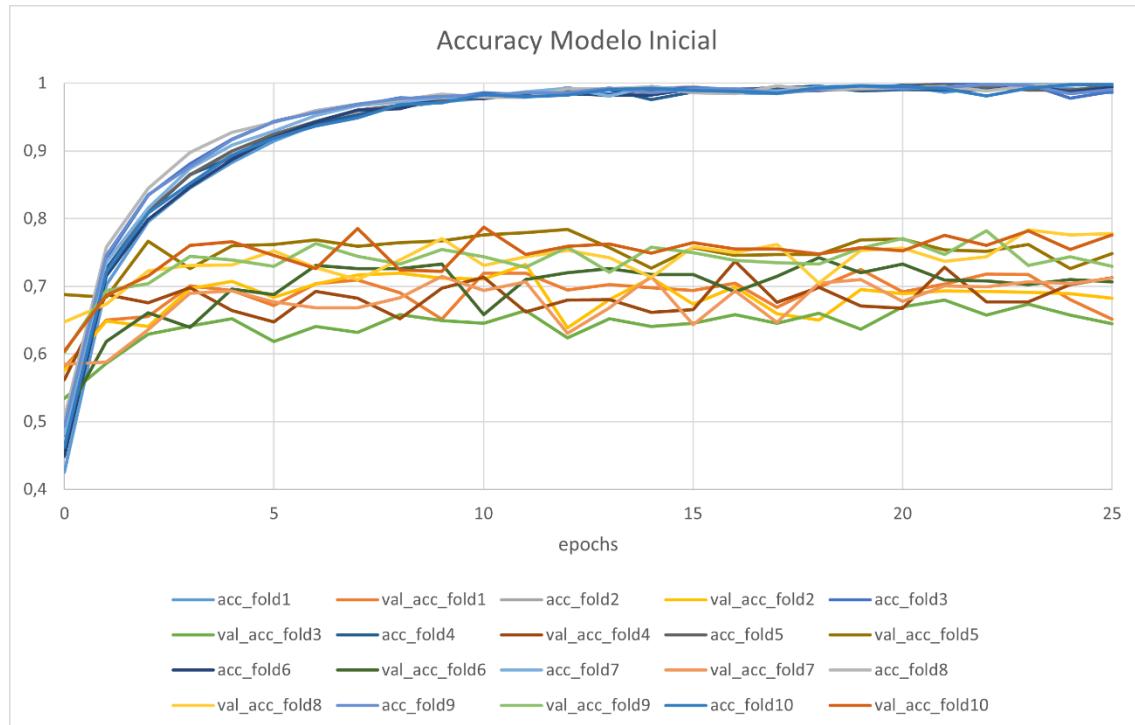


Figura 4-18. *Accuracies* de los conjuntos de entrenamiento y validación del modelo inicial.

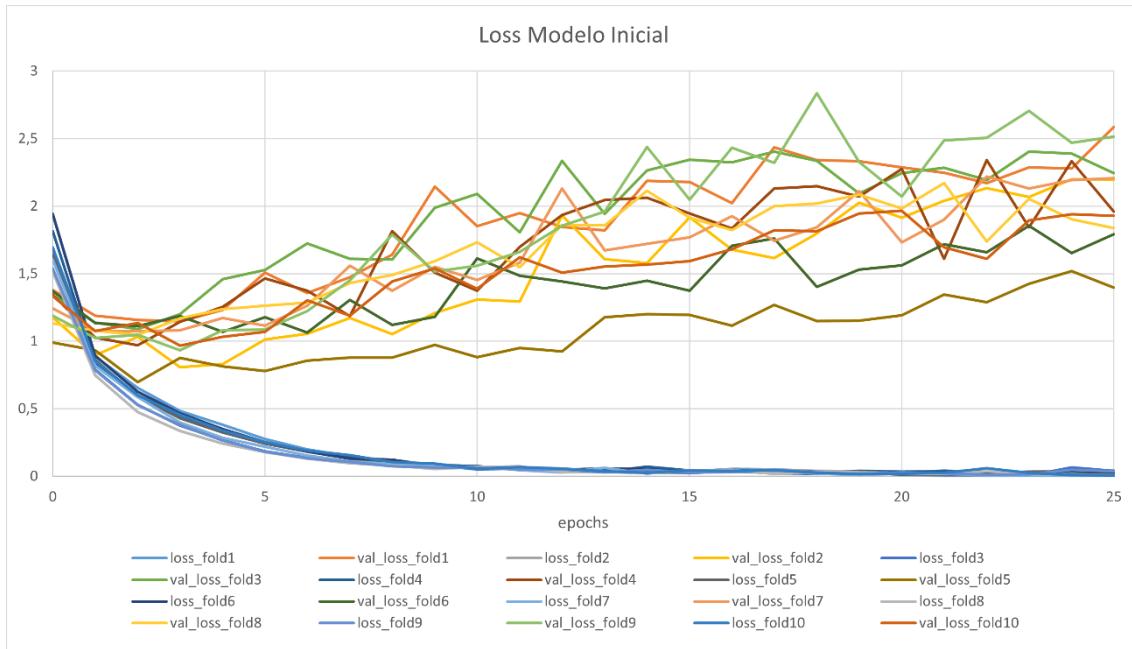


Figura 4-19. Loss de los conjuntos de entrenamiento y validación del modelo inicial.

Como se puede observar en las Figuras 4-18 y 4-19, tenemos un claro ejemplo de sobreajuste en nuestra CNN. La *accuracy* de la curva de entrenamiento llega prácticamente a 1, mientras que la de validación se queda estancada entre el 60% y 80%, lo que nos muestra que la red está memorizando los datos de entrada y no está generalizando el conocimiento para los nuevos datos. Además, los valores de *loss* crecen según avanzan las *epochs* en vez de disminuir, lo que indica que la red cada vez está menos segura de las clasificaciones del conjunto de validación.

4.5.1.2.3. MODELO 2: REDUCCIÓN DE PARÁMETROS

Para intentar solventar este problema se va a optar por reducir la complejidad de la red, con el objetivo de que ésta no tenga tantos recursos para amoldarse demasiado a los datos de entrenamiento.

Para ello, se van a sustituir la última capa de *max-pooling* y la de *flatten* por una capa de agrupación global máxima (*global max-pooling*). Esta capa agrupa el valor máximo de las matrices obtenidas por la última capa de convolución en un único píxel, de manera que el número de parámetros de la red se reduce drásticamente y, en consecuencia, su complejidad. Los parámetros entrenables de este modelo han pasado de 5.633.834 a 456.490, reduciéndose en un factor de más de 12. En la Figura 4-20 se muestra un resumen del modelo 2.

```

model = get_network()
model.summary()

learning rate: 0.0001
Model: "sequential"

-----  

Layer (type)          Output Shape       Param #
-----  

conv2d (Conv2D)        (None, 128, 173, 32)    608  

max_pooling2d (MaxPooling2D) (None, 64, 86, 32)    0  

)  

conv2d_1 (Conv2D)       (None, 64, 86, 64)     18496  

max_pooling2d_1 (MaxPooling2D) (None, 32, 43, 64)    0  

conv2d_2 (Conv2D)       (None, 32, 43, 128)    73856  

max_pooling2d_2 (MaxPooling2D) (None, 16, 21, 128)    0  

conv2d_3 (Conv2D)       (None, 16, 21, 256)    295168  

global_max_pooling2d (GlobalMaxPooling2D) (None, 256)    0  

dense (Dense)           (None, 256)         65792  

dense_1 (Dense)          (None, 10)          2570  

-----  

Total params: 456,490
Trainable params: 456,490
Non-trainable params: 0

```

Figura 4-20. Resumen del modelo 2.

Una vez aplicados los cambios, procedemos a entrenar el nuevo modelo con el conjunto de datos sin aumentar por un total de 25 *epochs*, ya que se ha visto que son suficientes para la convergencia del modelo.

4.5.1.2.4. RESULTADOS DEL MODELO 2

Para el modelo 2 se ha obtenido una *accuracy* máxima media de 75,92% que, con respecto al valor obtenido en el modelo anterior, supone una mejora del 1,66%. Los valores medios de *loss* obtenidos para estas *accuracies* es de 1,1, que a su vez representa una disminución del 37,7%. Además, se ha alcanzado la barrera del 80% de *accuracy* en la validación de algunos modelos. Por lo tanto, se puede decir que se ha conseguido una leve mejora.

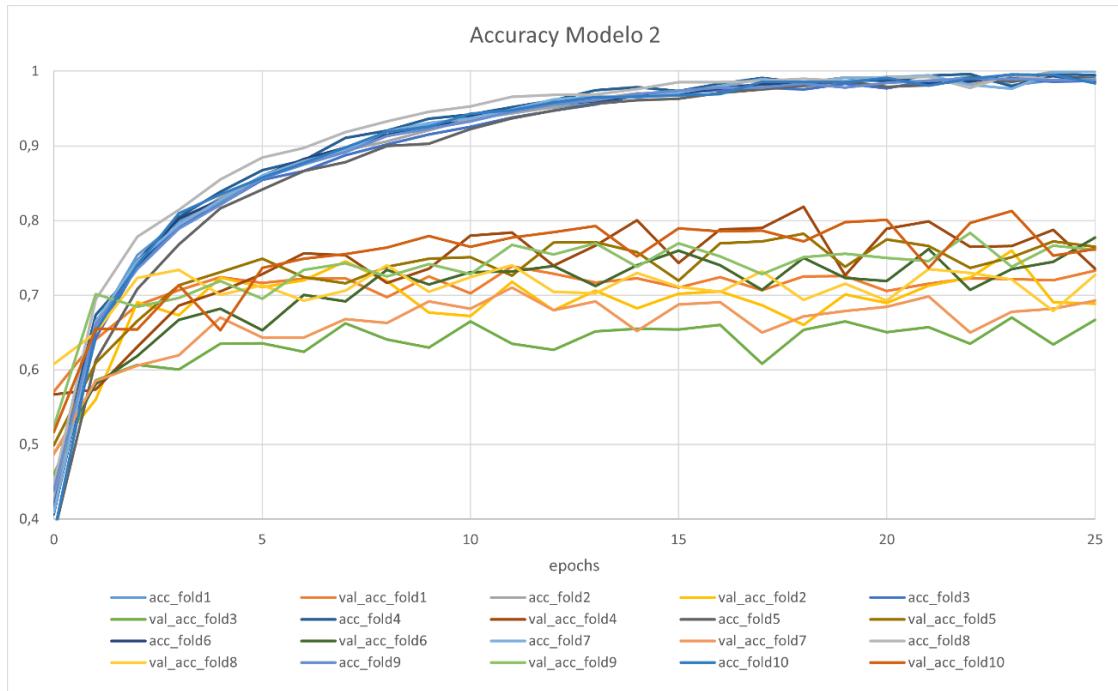


Figura 4-21. *Accuracies de los conjuntos de entrenamiento y validación del modelo 2.*

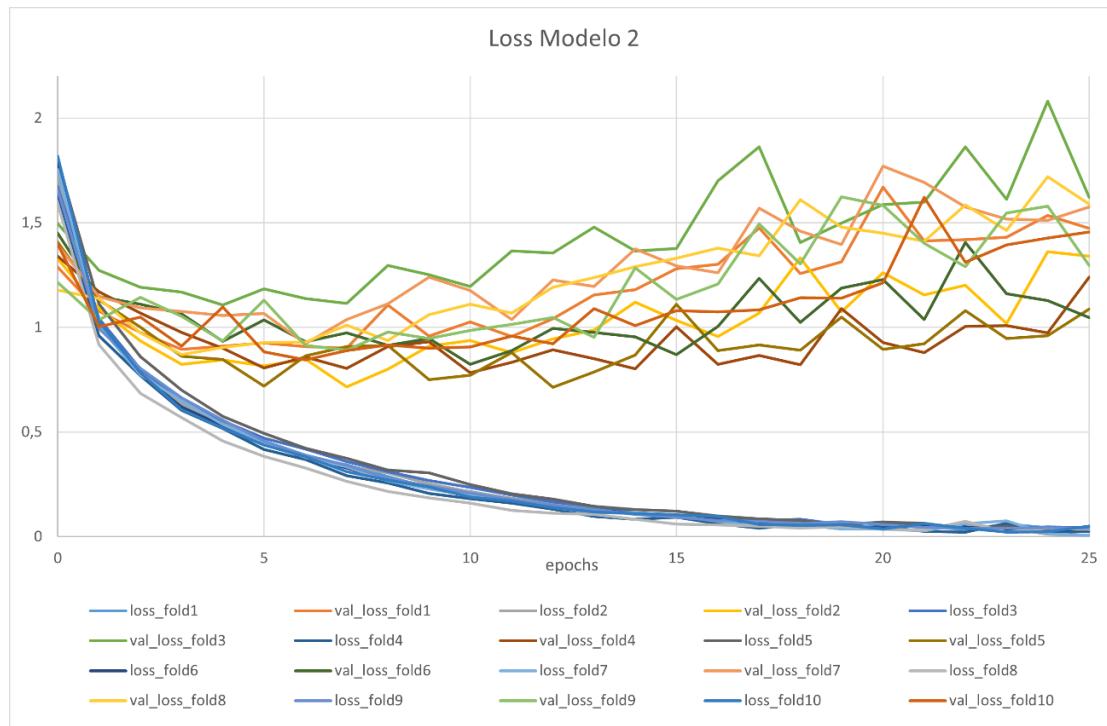


Figura 4-22. *Loss de los conjuntos de entrenamiento y validación del modelo 2.*

En las Figuras 4-21 y 4-22 se puede ver cómo el problema del sobreajuste persiste, pese a que se haya reducido drásticamente el número de parámetros. Sin embargo se han conseguido leves avances, ya que se han obtenido valores ligeramente

más altos de *accuracy* con unos valores de *loss* bastante menores con respecto al modelo anterior, lo que significa que hemos conseguido que la red generalice un poco mejor.

4.5.1.2.5. MODELO 3: AUMENTO DE DATOS

La arquitectura propuesta para el modelo 3 será la misma que la del modelo anterior. La diferencia estará en que, esta vez, entrenaremos el modelo con el conjunto de datos aumentado descrito en el apartado 4.5.1.1. Cabe destacar que el aumento de datos sólo se aplicará en el conjunto de entrenamiento, ya que la validación siempre deberá ser con audios reales, sin ninguna deformación.

Teniendo en cuenta que una *epoch* de este conjunto de entrenamiento contiene más de 12 veces la cantidad de datos utilizados en el modelo anterior, se va a reducir el número de *epochs* del entrenamiento a 16, teniendo también en cuenta que la variabilidad de los datos introducidos complicarán la convergencia de las curvas de aprendizaje.

4.5.1.2.6. RESULTADOS DEL MODELO 3

Para el modelo 3 se ha obtenido una *accuracy* máxima media del 78,82%, lo que supone una mejora del 3,82% del valor obtenido en el modelo anterior. Además, en la Figura 4-23 se puede ver cómo las curvas de *accuracy* de las validaciones ahora se encuentran en el rango del 65% al 85%.

Sin embargo, los valores de *loss* medios obtenidos han aumentado de 1,1 a 1,47, lo cual es un incremento algo significativo (Figura 4-24). Esto puede ser algo normal, debido a la mayor variabilidad de los datos que hemos introducido con el aumento y al mayor número de iteraciones (1 *epoch* de este modelo tiene 12 veces más iteraciones que el anterior). En términos generales, consideramos que los resultados obtenidos han sido positivos.

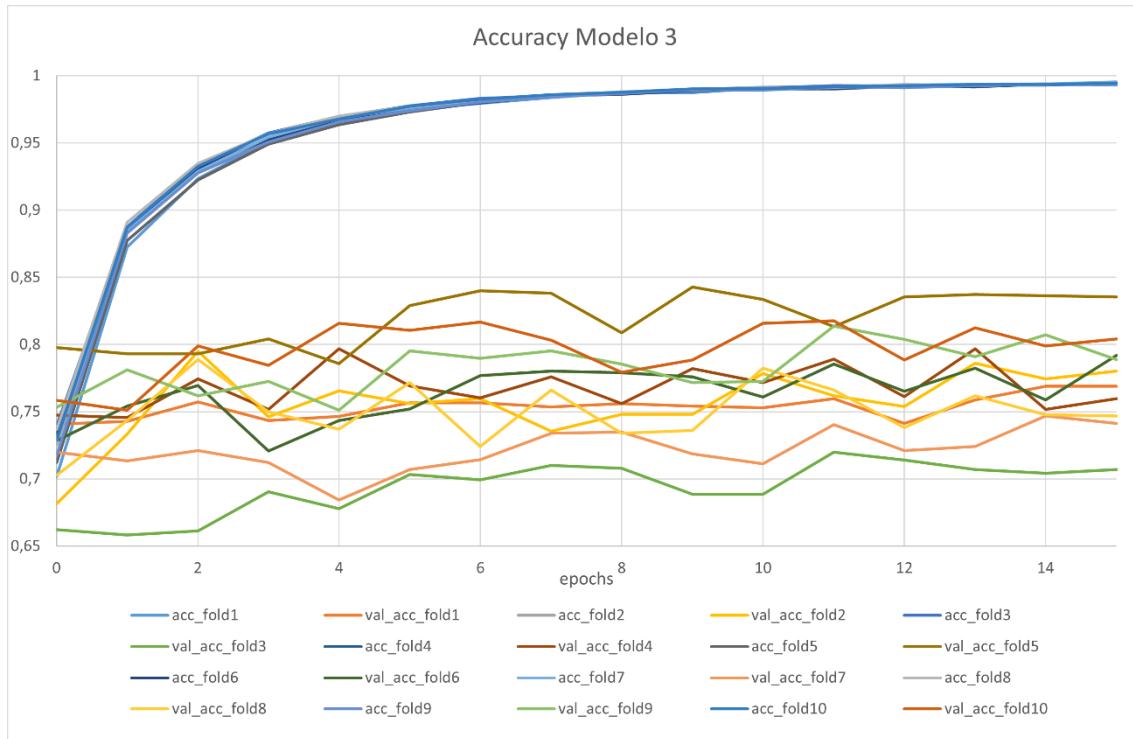


Figura 4-23. Accuracies de los conjuntos de entrenamiento y validación del modelo 3.

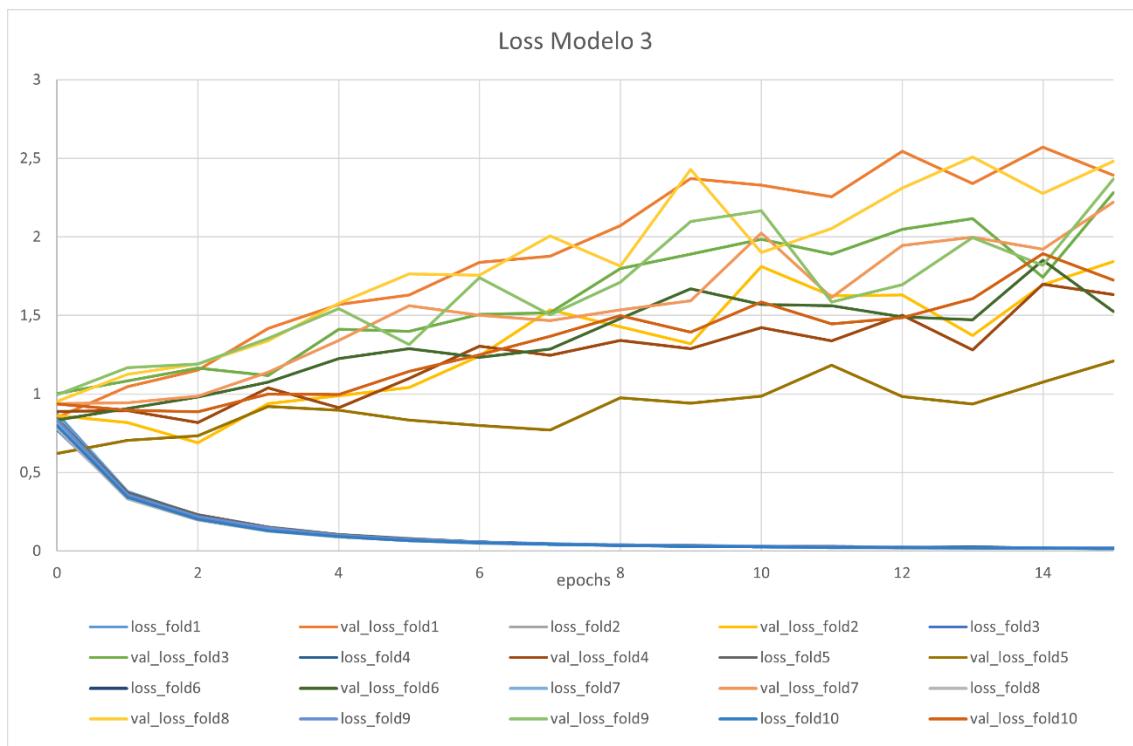


Figura 4-24. Loss de los conjuntos de entrenamiento y validación del modelo 3.

4.5.1.2.7. MODELO 4: *BATCH NORMALIZATION*

Para intentar seguir mejorando la generalización del modelo y reduciendo el sobreajuste, se añadirán capas de *batch normalization* entre las capas convolucionales y de *max-pooling*. De esta manera todos los datos durante el proceso de extracción de características estarán normalizados, lo que facilita su interpretación por las demás capas de la red. Además, no afectará mucho al número de parámetros de la red. El resumen de la arquitectura resultante se muestra en la Figura 4-25.

```
model = get_network()
model.summary()

learning rate: 0.0001
Model: "sequential"

Layer (type)          Output Shape       Param #
=====
conv2d (Conv2D)        (None, 128, 173, 32)    608
batch_normalization (BatchN ormalization)      128
max_pooling2d (MaxPooling2D (None, 64, 86, 32)    0
)
conv2d_1 (Conv2D)       (None, 64, 86, 64)     18496
batch_normalization_1 (BatchN ormalization)      256
max_pooling2d_1 (MaxPooling2D (None, 32, 43, 64)    0
)
conv2d_2 (Conv2D)       (None, 32, 43, 128)    73856
batch_normalization_2 (BatchN ormalization)      512
max_pooling2d_2 (MaxPooling2D (None, 16, 21, 128)    0
)
conv2d_3 (Conv2D)       (None, 16, 21, 256)    295168
batch_normalization_3 (BatchN ormalization)      1024
global_max_pooling2d (GlobalMaxPooling2D (None, 256)    0
)
dense (Dense)           (None, 256)          65792
dense_1 (Dense)         (None, 10)           2570
=====
Total params: 458,410
Trainable params: 457,458
Non-trainable params: 960
```

Figura 4-25. Resumen del modelo 4.

Además, debido a las oscilaciones detectadas en las curvas de aprendizaje de los modelos anteriores, se ha decidido implementar un planificador de la tasa de aprendizaje. Este planificador se llamará en un nuevo *callback* al empezar cada *epoch*. Para las 5 primeras *epochs*, la tasa de aprendizaje mantendrá su valor inicial de 0,0001, pero a medida que avancen las *epochs* a partir de la quinta, éste irá reduciendo su valor en un 5% por cada nueva *epoch*. De esta manera se pretende realentizar el descenso del gradiente y experimentar variaciones menos bruscas sobre las curvas de aprendizaje.

4.5.1.2.8. RESULTADOS DEL MODELO 4

Con este modelo se alcanza la barrera del 80% de *accuracy* máxima media. Además, los valores de *loss* obtenidos son ligeramente menores, con un 1,3 de media para las respectivas *accuracies* máximas.

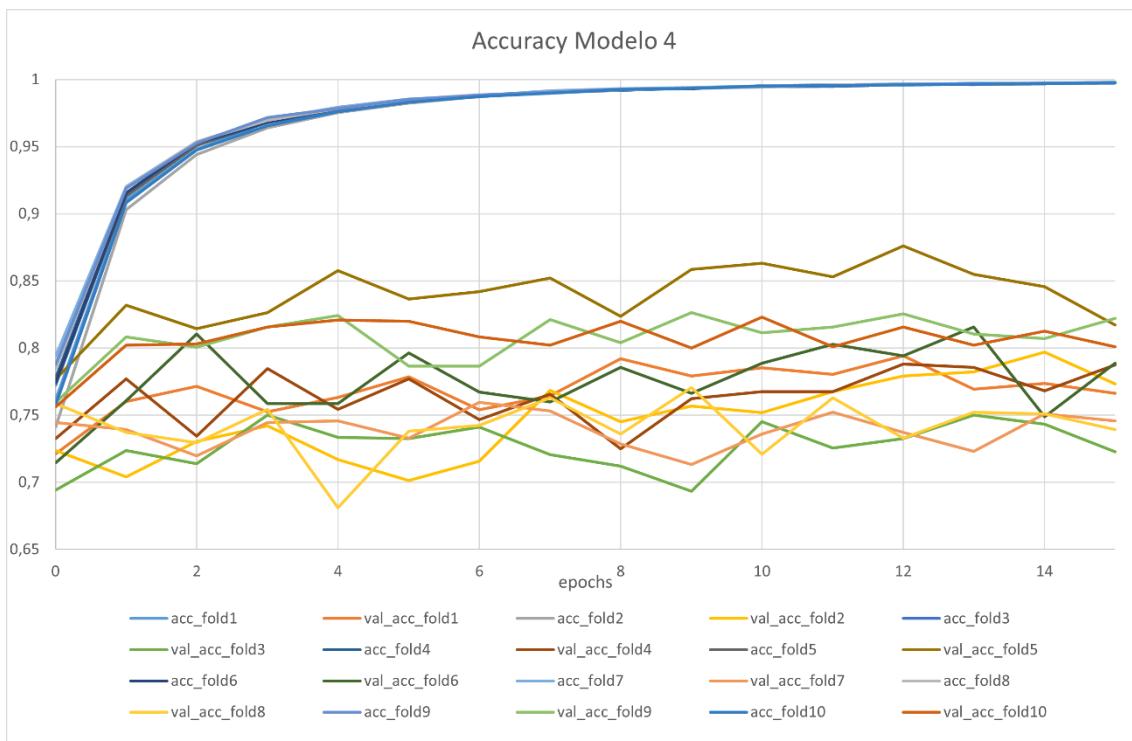


Figura 4-26. *Accuracies* de los conjuntos de entrenamiento y validación del modelo 4.

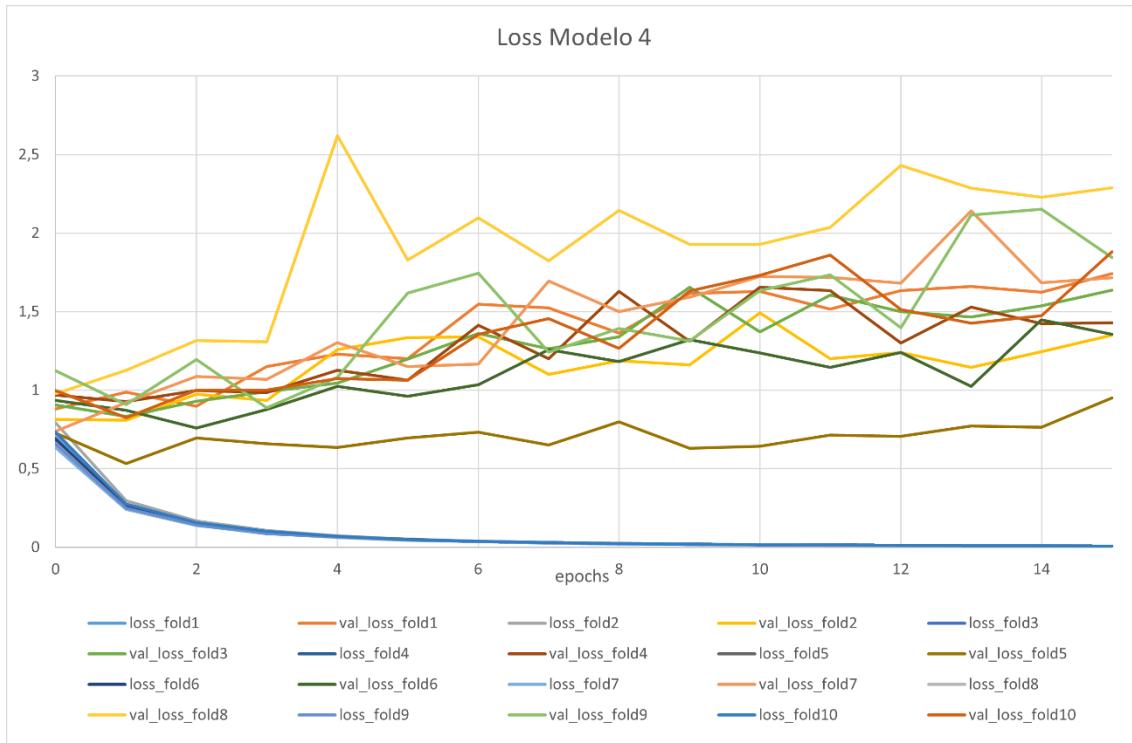


Figura 4-27. Loss de los conjuntos de entrenamiento y validación del modelo 4.

Seguimos teniendo sobreajuste y se produce incluso más rápido tras la implementación de las capas de *batch normalization*. Respecto a las oscilaciones de las curvas de aprendizaje (Figuras 4-26 y 4-27), éstas siguen presentes y no parecen haber disminuido mucho. Sin embargo, esto se explica debido a que las capas de normalización implementadas han aumentado la capacidad de aprendizaje de la red, lo que provoca un efecto contrario al de disminuir la tasa de aprendizaje.

4.5.1.2.9. MODELO 5: AUMENTO DE DATOS EN IMAGEN

Frente al aumento de la eficacia de la red en la clasificación de los datos de entrenamiento, se ha decidido aumentar aún más la variabilidad de los datos de entrada. Esta vez, con la particularidad de que las transformaciones se realizarán en el campo de la imagen, en vez de en la señal de audio.

Para esta tarea se va a emplear el módulo ImageGenerator de Keras, el cual nos permite aplicar distintas transformaciones sobre una imagen. Configuraremos algunos de sus parámetros para obtener transformaciones que tengan sentido en un espectrograma, como desplazamientos verticales, horizontales y *zoom* (Figura 4-28).

Las nuevas imágenes serán una combinación aleatoria de las transformaciones elegidas, aplicadas dentro de un rango definido. Hemos definido uno rango del 20% para desplazamientos horizontales y 5% para desplazamientos verticales y *zoom*. También

habrá que definir un método de relleno de píxeles, en nuestro caso, consideramos dos: *nearest* y *constant*. El primero rellena los píxeles vacíos con el valor del píxel más cercano, mientras que el segundo los rellena con un valor constante predefinido.

```
train_datagen = ImageDataGenerator(  
    width_shift_range=0.2,  
    height_shift_range=0.05,  
    zoom_range=0.05,  
    fill_mode='constant',  
    cval=-80.0)
```

Figura 4-28. Aumento de datos con ImageGenerator.

El método de relleno *constant* lo interpretaremos como añadir silencio a los píxeles que queden vacíos, por lo que tendremos que definir un valor de relleno de -80 dBFS. Sin embargo, como nuestra imagen tiene dos canales (log-Mel y delta-log-Mel) y queremos que se aplique la misma transformación en ambos, debemos sustituir los valores remplazados de -80 dBFS por un valor de 0 (derivada de una constante) para la matriz delta-log-Mel.

Como también queremos mostrarle a la red algunas muestras sin modificar o modificadas únicamente en el campo de sonido, se va a implementar de forma que sólo se aplique esta transformación en el 40% de los *batch* generados en cada *epoch*. Además, las transformaciones de la imagen se aplicarán tanto sobre las muestras originales, como las aumentadas en el campo del sonido.

Aunque con el modelo anterior las oscilaciones de las curvas de aprendizaje persistían, debido a la variabilidad introducida de los nuevos datos de aumento, esperamos que a la red le cueste más aprender, por lo que decidimos mantener el planificador de la tasa de aprendizaje sin ningún cambio. Además, vamos a aumentar también el número de *epochs* a 21.

Finalmente, decidimos entrenar dos modelos por separado, uno con cada método de relleno contemplado, para ver cuál favorece más al aprendizaje de la red.

4.5.1.2.10. RESULTADOS DEL MODELO 5

Aunque las curvas de aprendizaje de ambos modelos son muy similares, el rendimiento del método *nearest* ha sido ligeramente superior, obteniendo una *accuracy* máxima media del 81,05% frente al 80,86% del método *constant*. Además, los valores medios de *loss* obtenidos para estas *accuracies* máximas son también algo menores, siendo de 0,88 para el método *nearest* y de 0,96 para el método *constant*.

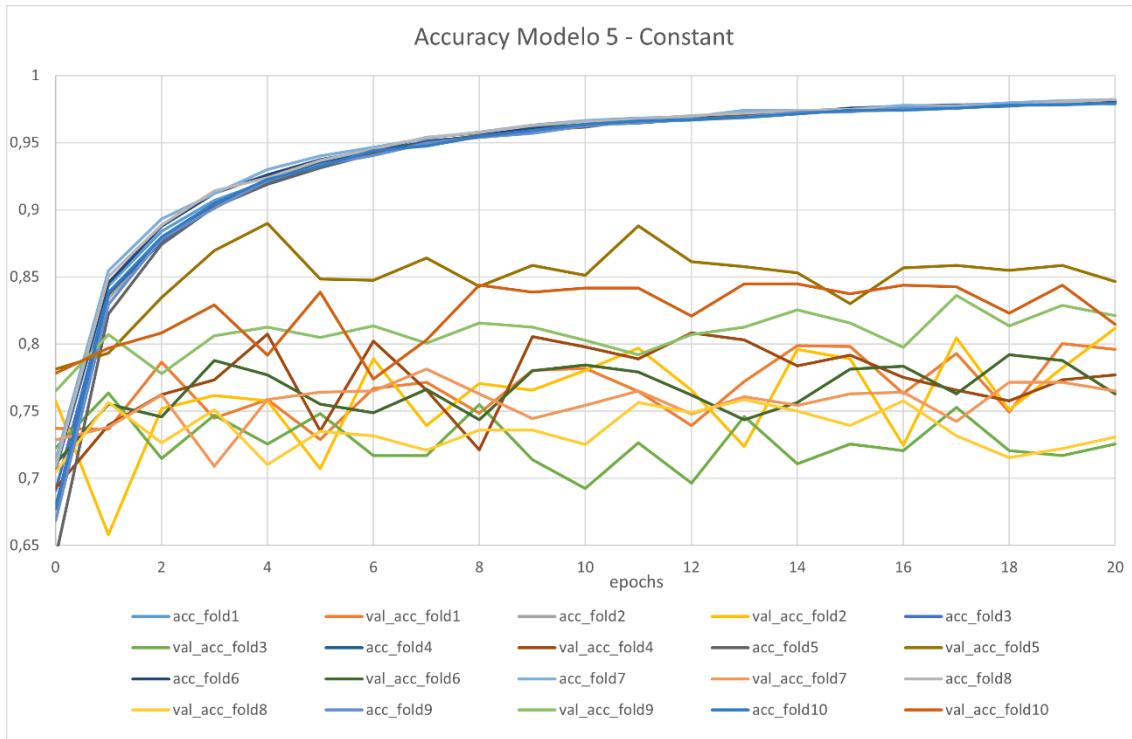


Figura 4-29. Accuracies de entrenamiento y validación del modelo 5 con relleno *constant*.

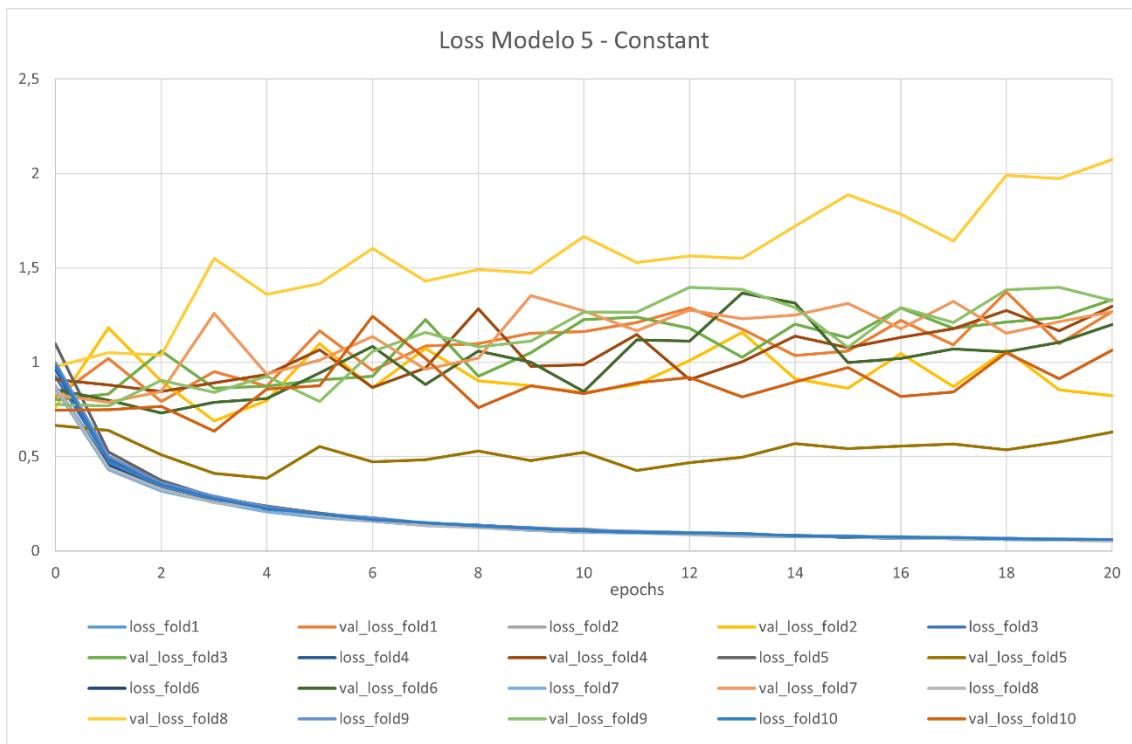


Figura 4-30. Loss de entrenamiento y validación del modelo 5 con relleno *constant*.

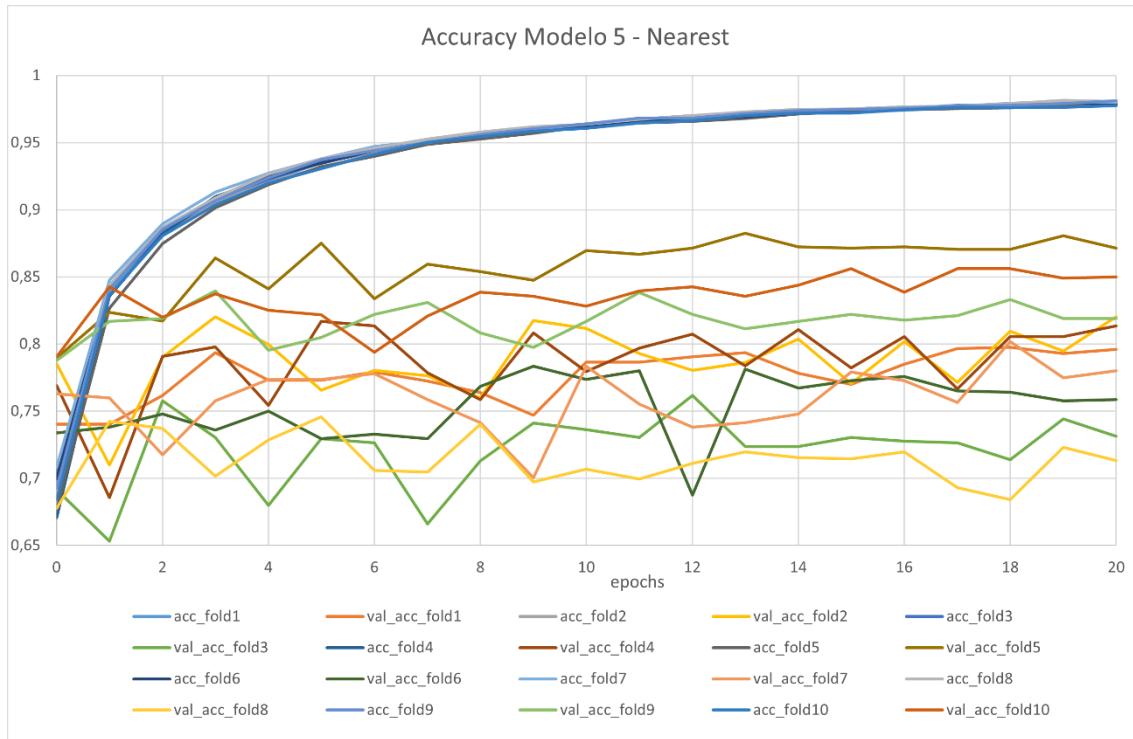


Figura 4-31. Accuracy de entrenamiento y validación del modelo 5 con relleno *nearest*.

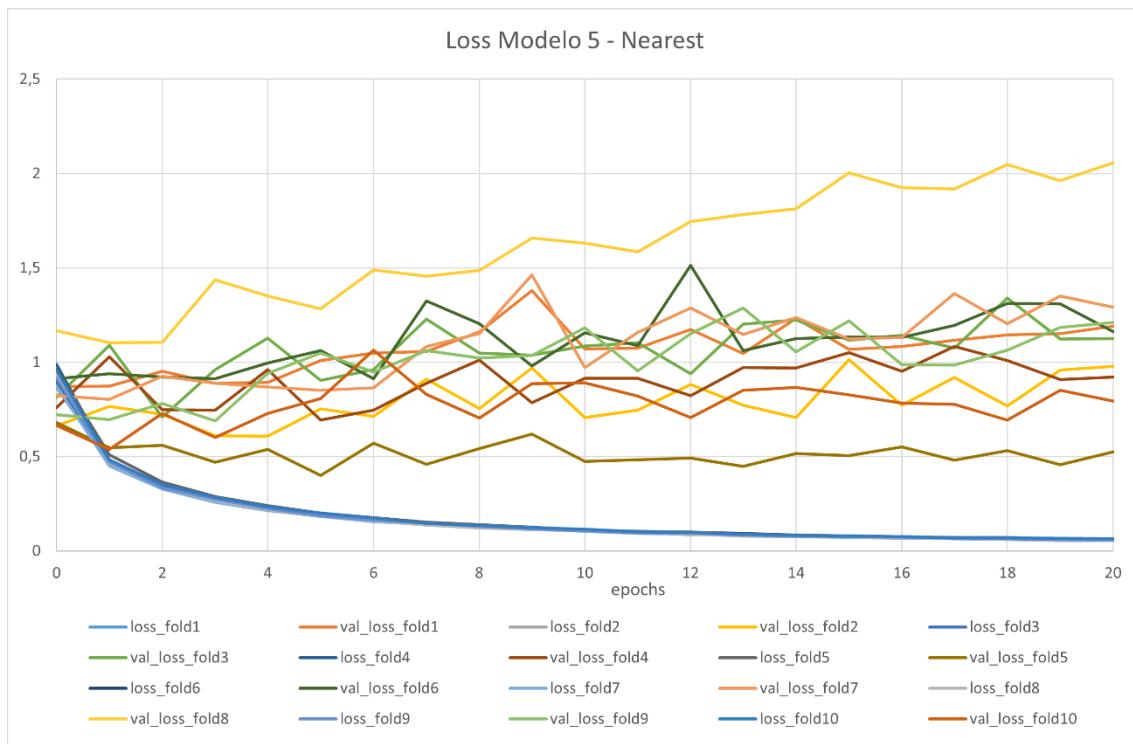


Figura 4-32. Loss de entrenamiento y validación del modelo 5 con relleno *nearest*.

Como se puede ver en las Figuras 4-29, 4-30, 4-31 y 4-32, a la red esta vez le ha costado más aprender a clasificar el nuevo aumento de datos. Esto se deduce con el tiempo de establecimiento de la curva de validación del conjunto de entrenamiento. Mientras que en el modelo 4 ha necesitado únicamente de 3 *epochs* para alcanzar el 95% del valor final, en este modelo ha necesitado 7.

Además, las oscilaciones de las curvas de aprendizaje se han ido reduciendo a medida que aumentaban las *epochs*, lo que demuestra que el planificador de la tasa de aprendizaje está favoreciendo la convergencia.

Sin embargo, aunque el rendimiento de la red ha mejorado ligeramente, sigue observándose el sobreajuste.

4.5.1.2.11. MODELO FINAL: REGULARIZACIÓN DROPOUT Y L2

Para este modelo, mantendremos el aumento de datos en el campo de la imagen con el método de relleno *nearest*, ya que es el que mejores resultados ha dado.

Con el fin de reducir el sobreajuste y conseguir que la red generalice en vez de memorizar las características del conjunto de entrenamiento, vamos a implementar más métodos de regularización.

Añadiremos una regularización L2 en la capa neuronal oculta con una tasa de penalización de 0,05 para dificultar que las neuronas se centren demasiado en ciertas características.

También añadiremos una capa de *dropout* con una probabilidad de descarte del 50% después de la misma. Con esto conseguimos que las neuronas no dependan tanto unas de otras y que sea toda la capa la que aprende en conjunto.

Finalmente, conectaremos una última capa de *batch normalization* entre la capa oculta y la capa final (Figura 4-33) para que las neuronas de clasificación dispongan también de sus entradas normalizadas.

```

model = get_network()
model.summary()

learning rate: 0.0001
Model: "sequential"

Layer (type)          Output Shape         Param #
=====conv2d (Conv2D)      (None, 128, 173, 32)      608
batch_normalization (BatchN (None, 128, 173, 32)      128
ormalization)

max_pooling2d (MaxPooling2D (None, 64, 86, 32)      0
)

conv2d_1 (Conv2D)       (None, 64, 86, 64)      18496
batch_normalization_1 (Batch (None, 64, 86, 64)      256
hNormalization)

max_pooling2d_1 (MaxPooling (None, 32, 43, 64)      0
2D)

conv2d_2 (Conv2D)       (None, 32, 43, 128)      73856
batch_normalization_2 (Batch (None, 32, 43, 128)      512
hNormalization)

max_pooling2d_2 (MaxPooling (None, 16, 21, 128)      0
2D)

conv2d_3 (Conv2D)       (None, 16, 21, 256)      295168
batch_normalization_3 (Batch (None, 16, 21, 256)      1024
hNormalization)

global_max_pooling2d (Global (None, 256)      0
MaxPooling2D)

dense (Dense)           (None, 256)      65792
dropout (Dropout)        (None, 256)      0
batch_normalization_4 (Batch (None, 256)      1024
hNormalization)

dense_1 (Dense)          (None, 10)      2570
=====

Total params: 459,434
Trainable params: 457,962
Non-trainable params: 1,472

```

Figura 4-33. Resumen del modelo final.

Debido a la regularización introducida, esperamos que el aprendizaje sea más lento, por lo que vamos a entrenar al modelo durante un total de 31 *epochs*.

En la Figura 4-34 se muestra el diagrama de flujo del *script* de entrenamiento del modelo final (Anexo A, apartado A.4).

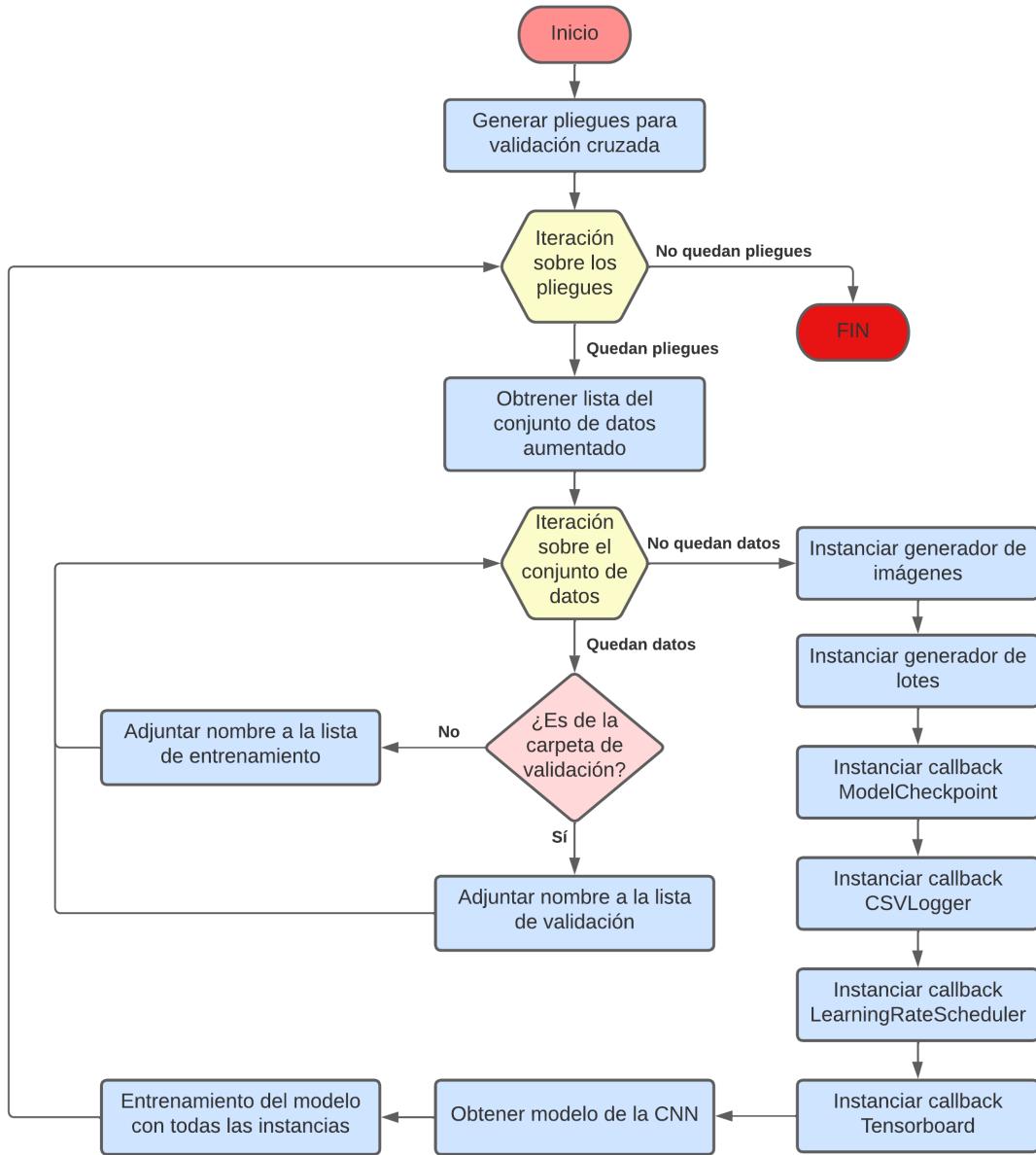


Figura 4-34. Diagrama de flujo del *script* de entrenamiento del modelo final.

4.5.1.2.12. RESULTADOS DEL MODELO FINAL

Tras el entrenamiento del modelo final, se ha obtenido una *accuracy* máxima media del 82,3%, que es la más alta lograda entre todos los modelos anteriores. Además, el valor de *loss* promedio para cada *accuracy* máxima ha sido de 0,82, siendo éste también el valor más bajo obtenido.

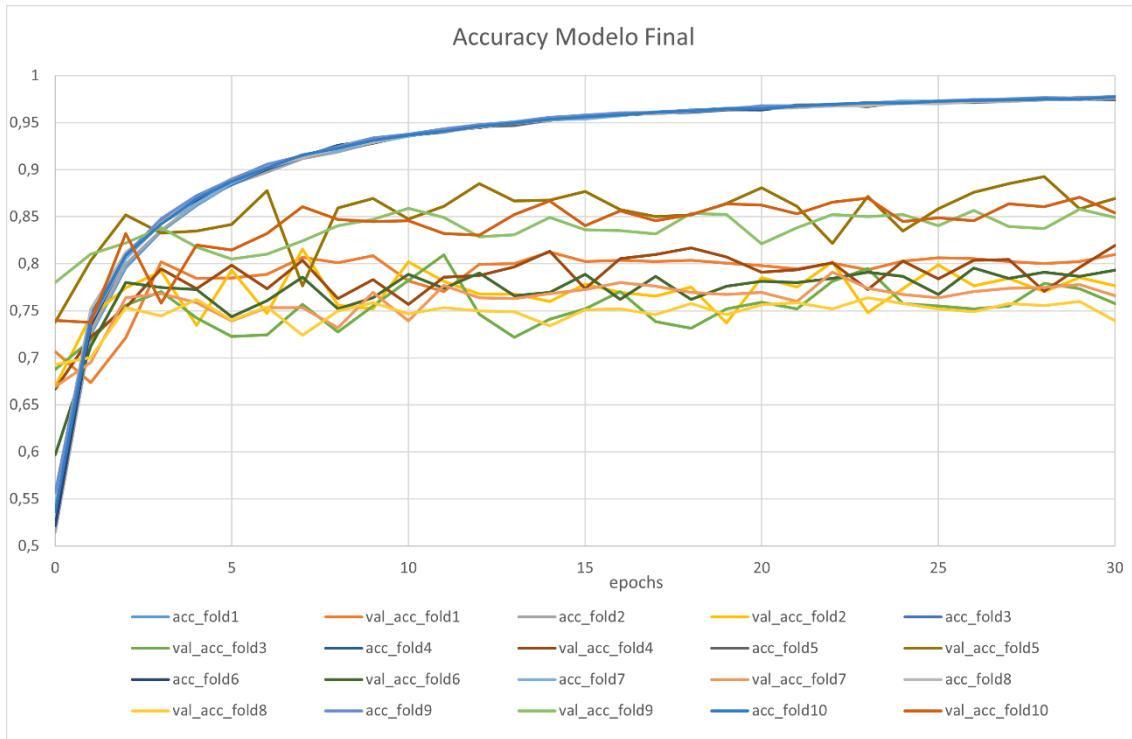


Figura 4-35. Accuracies de los conjuntos de entrenamiento y validación del modelo final.

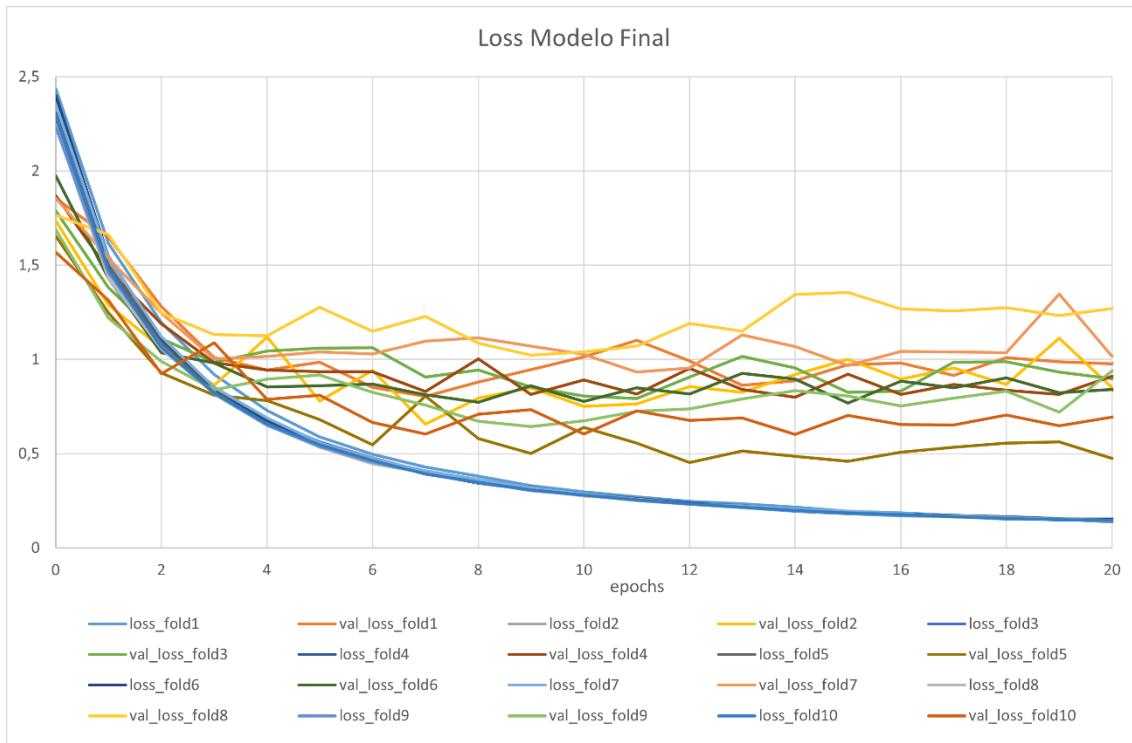


Figura 4-36. Loss de los conjuntos de entrenamiento y validación del modelo final.

Aunque el sobreajuste sigue estando presente, éste se ha reducido considerablemente desde el modelo inicial. En las Figuras 4-35 y 4-36 podemos ver como las curvas de *accuracy* y *loss* del modelo convergen en valores más altos y más bajos respectivamente, alcanzando casi el 90% de *accuracy* en algunos pliegues.

El modelo final ha mejorado considerable el rendimiento de la red, alcanzando una *accuracy* máxima comparable a los modelos de CNN del estado del arte.

4.5.1.2.13. COMPARACIÓN DE LOS MODELOS Y ELECCIÓN FINAL

En la Figura 4-37 se muestra un resumen de la evolución de los valores de *accuracy* y *loss* obtenidos para cada modelo.

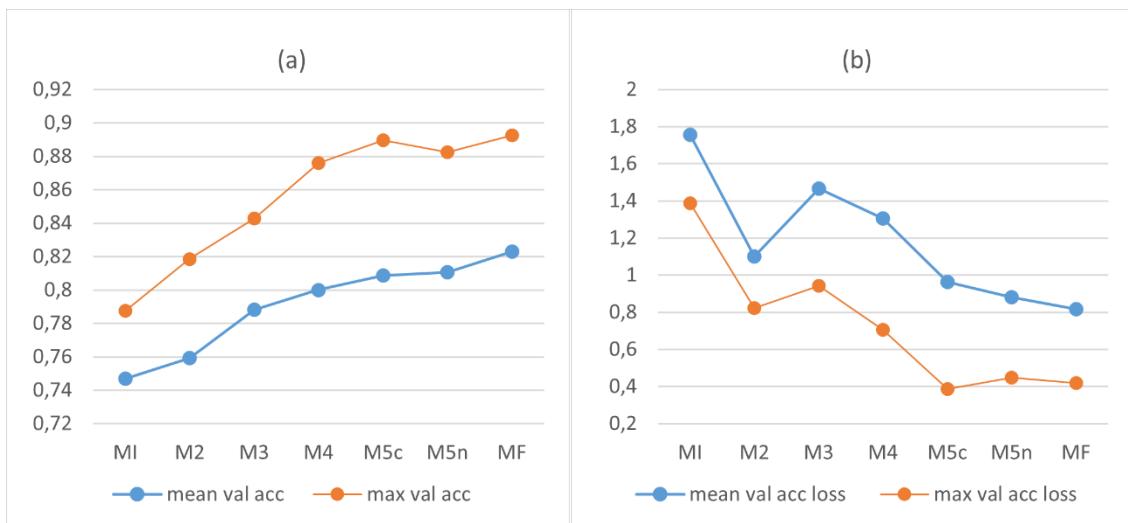


Figura 4-37. a) *Accuracies* de validación medias y máximas. b) Valores de *loss* para las *accuracies* de validación medias y máximas. (MI = Modelo Inicial, M1 = Modelo 1, M2 = Modelo 2, M3 = Modelo 3, M4 = Modelo 4, M5c = Modelo 5 constant, M5n = Modelo 5 nearest, MF = Modelo Final)

Como el objetivo principal es obtener el modelo con la mayor *accuracy* (de validación) posible que permita una clasificación precisa para nuestras muestras, priorizaremos los valores de *accuracy* máximos obtenidos frente a los valores de *loss* mínimos obtenidos. También priorizaremos la *accuracy* media del modelo frente a la *accuracy* máxima (de un solo pliegue), ya que es preferible que el modelo en conjunto sea robusto a que se alcance un valor de *accuracy* máximo en un único pliegue.

El modelo que consigue tanto la mayor *accuracy* media, como el valor de *accuracy* máxima más alto para un pliegue, es el modelo final. Además, también tiene el valor de *loss* medio más bajo y su *loss* para el pliegue con la mayor *accuracy* es de los

más pequeños obtenidos. Por lo tanto, no hay duda de que para la tarea de clasificación de nuestras muestras elegiremos uno de los pliegues del modelo final.

En la Figura 4-38 se muestran los valores de *accuracy* máximos y sus respectivos valores de *loss* para cada pliegue del modelo final.

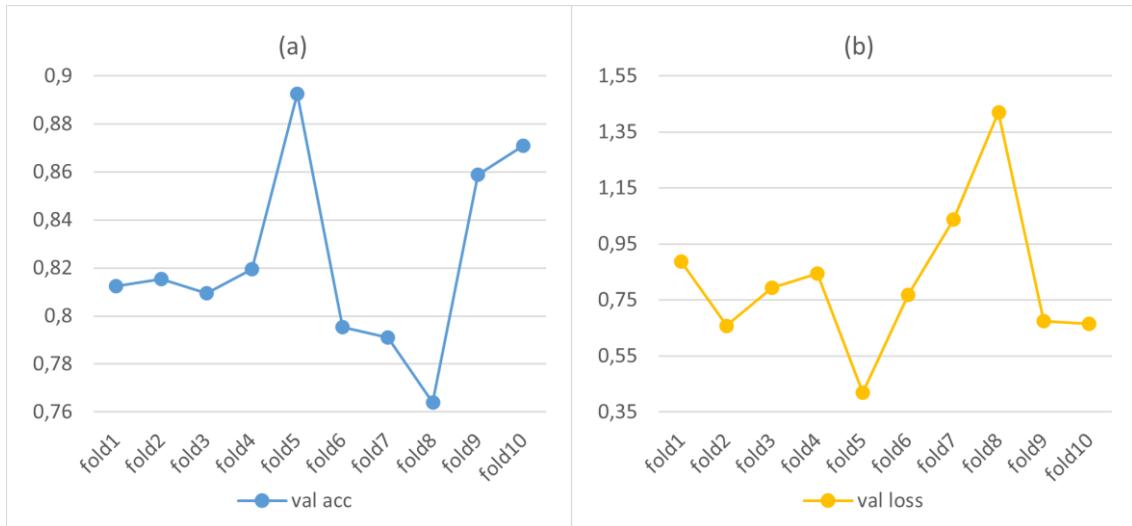


Figura 4-38. a) Valores de *accuracy* máximos por pliegues. b) *loss* relativos a los valores de *accuracy* máximos para cada pliegue.

Como se puede observar en la Figura 4-38, el pliegue que mayor *accuracy* y a su vez menor *loss* ha obtenido, con unos valores de 89,25% y 0,42 respectivamente, es el que se ha validado con la carpeta 5. Esto se puede deber a que los datos de validación son más “fáciles” de clasificar, o bien porque los datos de entrenamiento son los que mayor variabilidad tienen y más facilita el correcto aprendizaje de la red. De cualquier manera, los valores obtenidos para la *accuracy* máxima son la mejor referencia en las que nos podemos basar. Por lo tanto, el modelo que utilizaremos para la clasificación de nuestras muestras de sonido ambiental será el que obtuvo la máxima *accuracy* en el pliegue 5.

4.5.1.2.14. TIEMPOS DE ENTRENAMIENTO

A lo largo del proceso de desarrollo de la CNN, se han ido implementando cambios que han afectado también al tiempo de entrenamiento de la red. En la Figura 4-39 se muestra la tabla con los tiempos medios del entrenamiento para cada modelo utilizando los recursos del Magerit descritos en la Figura 4-10.

Modelo	Tiempo/epoch	Número Epochs	Tiempo/pliegue	Número pliegues	Tiempo total
Modelo Inicial	28 s	26	12,1 min	10	2 h
Modelo 2	27 s	26	11,7 min	10	1,95 h
Modelo 3	4,1 min	16	1,1 h	10	11 h
Modelo 4	4,1 min	16	1,1 h	10	11 h
Modelo 5 constant	5,15 min	21	1,8 h	10	18 h
Modelo 5 nearest	5,15 min	21	1,8 h	10	18 h
Modelo Final	5,15 min	31	2,7 h	10	27 h

Figura 4-39. Tiempos medios de entrenamiento en Magerit.

Cabe decir que la mayor parte del tiempo hemos contado con 4 nodos de trabajo disponibles, por lo que hemos dividido el entrenamiento de cada modelo por lotes de pliegues, de manera que el tiempo total de entrenamiento se ha dividido entre los 4 nodos.

4.5.2. CONFIGURACIÓN DEL DISPOSITIVO DE MEDICIÓN

4.5.2.1. CONFIGURACIÓN INICIAL DE LA RPI

El primer paso para la configuración de la RPI será instalar el sistema operativo en una tarjeta SD. Para ello, el método más sencillo es descargar la aplicación Raspberry Pi Imager desde la página oficial de Raspberry Pi¹⁵. Una vez instalada, bastará con seleccionar el sistema operativo que se desee y la tarjeta SD en la que lo vayamos a grabar. En nuestro caso se ha elegido el sistema operativo Raspberry Pi OS Full (Raspbian) 32-bit, que está basado en la distribución de Linux, Debian.

La elección de esta distribución se debe a que está optimizada para funcionar en equipos basados en procesadores de arquitectura ARM. Aunque existe una versión 64-bit compatible con nuestro modelo de RPI, es relativamente nueva, poco estable y hay bastantes aplicaciones que aún no están optimizadas para ella. Es por ello que se ha preferido trabajar con la de 32-bit, ya que acumula muchos años de desarrollo, es

¹⁵ <https://www.raspberrypi.com/software/>

estable y cuenta con más aplicaciones. Además, entre las distintas opciones de instalación que hay, hemos elegido instalar la versión completa, que incluye la interfaz de escritorio PIXEL y varios programas ya instalados.

La aplicación también ofrece un apartado de ajustes para configurar algunos campos clave antes de la instalación. En nuestro caso hemos cambiado la contraseña por defecto, configurado el acceso a la red WIFI, el *host name* y habilitado la conexión por SSH para conectarnos a la RPI desde nuestro ordenador sin la necesidad de conectar ningún periférico.

Una vez finalizado el proceso de escritura del sistema operativo sobre la tarjeta SD, se introduce en la RPI y se enciende. La RPI se conectará de forma automática a la red WIFI previamente configurada, lo que nos permitirá conectarnos a ella de manera remota. Para ello, utilizaremos el cliente SSH de PUTTY¹⁶ y estableceremos la conexión por terminal. Seguidamente, se empleará también VNC Viewer¹⁷ para el acceso remoto a la interfaz gráfica. Para esto necesitaremos conocer la dirección IP de la RPI o su *host name*, el cual configuraremos previamente para evitar posibles errores con otras RPI conectadas a la red. En algunas ocasiones, para escanear la red y encontrar la dirección IP de la RPI se ha utilizado el programa Advanced IP¹⁸ Scanner.

Para facilitar la tarea de identificación de la IP de la RPI en situaciones en las que no sea fácil de acceder a esta información, se ha configurado un *script* de Bash que se ejecuta al encender la RPI. Este *script* comprueba la conexión a internet del dispositivo. Si el dispositivo tiene conexión a internet, ejecuta otro *script* de Python que sube toda la información de la conexión establecida, incluyendo la dirección IP, en formato de texto plano a una carpeta de Google Drive. De esta manera, se puede consultar esta información desde cualquier dispositivo que tenga acceso a la cuenta de Google Drive utilizada.

4.5.2.2. CONFIGURACIÓN PARA LA TOMA DE AUDIOS

El mismo *script* que se ejecuta al encender la RPI comprueba varias veces que el micrófono USB esté conectado y, en caso afirmativo, ejecuta el *script* de grabación de Python. Por el contrario, si tras un determinado número de intentos no se ha detectado el micrófono USB, se aborta el proceso de tomas de audio, teniendo que reiniciar la RPI o ejecutar el *script* manualmente para volver a iniciar el proceso.

Además, dependiendo de en qué puerto USB esté conectado el micrófono, se generará un archivo distinto de configuración de la grabación. De esta manera, se

¹⁶ <https://www.putty.org/>

¹⁷ <https://www.realvnc.com/es/connect/download/viewer/>

¹⁸ <https://www.advanced-ip-scanner.com/es/>

pueden predefinir hasta cuatro modos grabación sin necesidad de acceder a la RPI para editar los parámetros a mano. Los parámetros que se pueden configurar son: la duración de las grabaciones, el tiempo de espera entre grabaciones y el número de grabaciones totales a realizar. Esto proporciona una gran flexibilidad a la hora de tomar audios in situ, ya que lo único que hay que hacer si se quiere cambiar la configuración de la toma de audios es conectar el micrófono al puerto USB correspondiente y reiniciar la RPI para que se genere el nuevo fichero de configuración. En la Figura 4-40 se muestra el flujograma del *script* de inicio (Anexo A, apartado A.1).

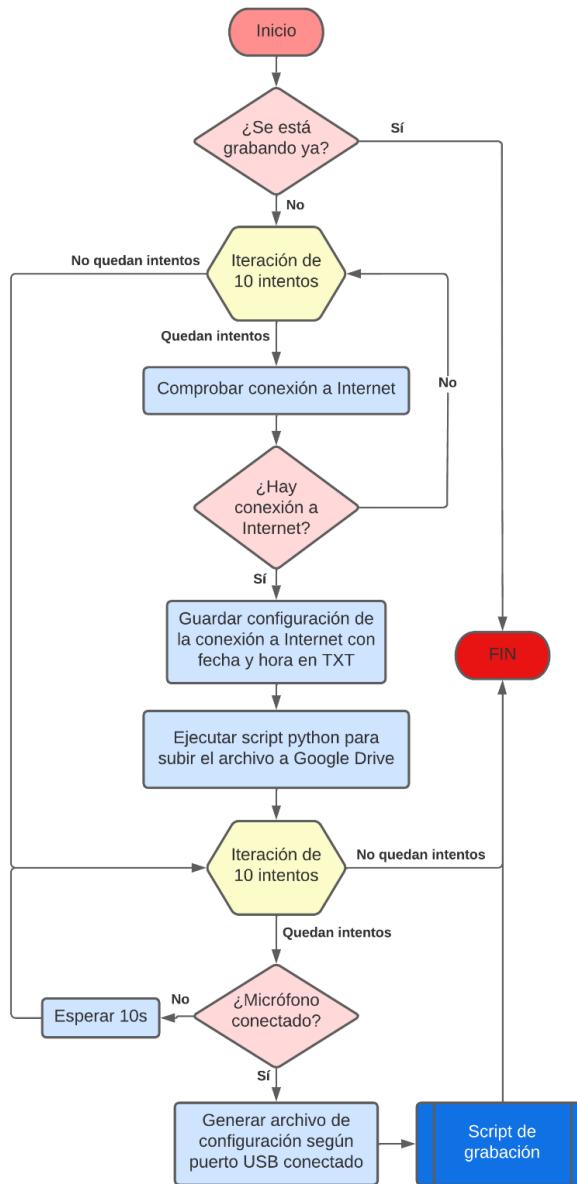


Figura 4-40. Flujograma del *script* de inicio de la RPI.

El *script* de grabación de Python tomará los audios con la configuración indicada y los guardará en un archivo WAV con la fecha y hora exacta a la que fueron tomados como nombre del archivo. Tras terminar la grabación de todos los audios, los segmentará en clips de cuatro segundos de duración y los guardará en una nueva

carpeta, con la fecha correspondiente a la que fueron tomados como nombre. A continuación, generará un archivo CSV con toda la información correspondiente a las intensidades de sonido medidas en cada audio y clip. En el contenido de este CSV se mostrarán los decibelios fondo de escala (dBFS) y de nivel de presión sonora (dB SPL) medios y máximos junto con el nombre del archivo de cada muestra analizada.

Todos los archivos que se generen, tanto los clips de audio, como los CSV de decibelios, se podrán subir a la carpeta del proyecto de Google Drive si hay conexión a Internet, así como un registro de todo lo que va sucediendo en el *script* con la fecha y hora de cada evento registrado. Aunque debido la toma masiva de muestras en este proyecto, sólo se han subido los archivos CSV, así como un registro del estado de la ejecución del *script* de grabación para comprobar que todo está funcionando correctamente. De no haber conexión a Internet, simplemente estos pasos se ignoran y se guarda todo únicamente en el almacenamiento local de la RPI. En la Figura 4-41 se muestra el diagrama de flujo del *script* de grabación (Anexo A, apartado A.2).

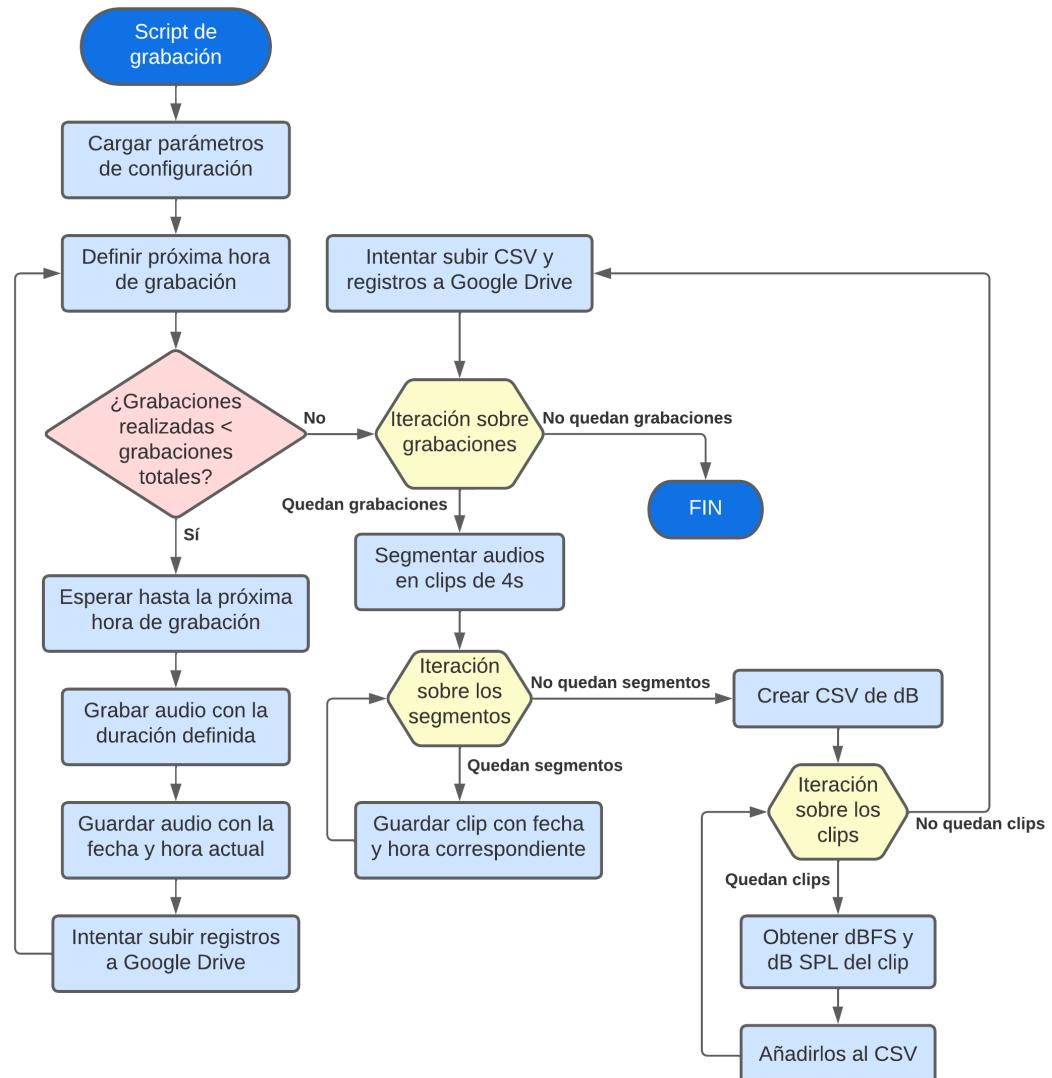


Figura 4-41. Flujograma del *script* de grabación de muestras.

Una vez finalizado el proceso de toma de muestras, se ejecutará automáticamente el *script* de Python que llevará a cabo la transformación de los sonidos tomados a imagen (descrito en la Figura 4-16) con una ligera modificación que se explicará más adelante. A continuación, se determinará la clasificación de las muestras obtenidas con el modelo de CNN y se guardarán los resultados en un nuevo CSV.

Con el objetivo de automatizar la transferencia de archivos entre la RPI y el ordenador portátil, se ha implementado un *script* de Batch que establece conexión con la RPI mediante su *local host* y recibe los archivos de la RPI (Anexo A, apartado A.7). Para hacer la transferencia de archivos por SSH será necesaria la descarga de pscp.exe desde la página de PUTTY¹⁹. Esto agilizará la obtención de los resultados de clasificación obtenidos, así como los registros de las intensidades sonoras de las muestras de audio y otros archivos.

4.5.2.3. CALIBRACIÓN DEL MICRÓFONO

Como se ha explicado en el apartado 3.6, la unidad que se utiliza para medir la intensidad del sonido en sistemas digitales es el dBFS, que tiene como referencia el valor de nivel de presión sonora máximo que puede medir el dispositivo. Estos valores serán diferentes para cada micrófono, dependiendo de su rango de medición, su sensibilidad, etc. Por esta razón, es necesario calibrar el micrófono en un punto de referencia y poder obtener así la relación entre la medida en dBFS y dB SPL. Para ello, se utilizará un sonómetro, que es un dispositivo electrónico normalizado para monitorizar los niveles de presión sonora en dB SPL.

El proceso de calibración consiste en colocar el micrófono a calibrar y el sonómetro a la misma distancia de un altavoz, de manera que se pueda considerar que ambos dispositivos están midiendo en el mismo punto (Figura 4-42). El altavoz reproducirá ruido rosa a un determinado volumen durante la prueba. El ruido rosa se caracteriza por contener todas las frecuencias audibles por el ser humano (20 – 20.000 Hz) y porque todas las bandas de octava tienen el mismo nivel sonoro, siendo el sonido de referencia utilizado normalmente para la calibración de equipos acústicos.

¹⁹ <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>



Figura 4-42. Calibración del micrófono con sonómetro.

Para monitorizar las medidas de presión sonora del micrófono, se utilizará el programa Rational acoustics: Smaart v8 (Figura 4-43). Ambos dispositivos se configurarán en modo *SLOW*: valoración temporal lenta (tiempo de muestreo de 1s) y a continuación, se registrará la diferencia de los dB medidos en cada dispositivo como el *offset*. Esto se puede hacer de forma automática desde el programa, anotando el valor medido por el sonómetro al calibrar el micrófono.



Figura 4-43. Calibración del micrófono con Smaart v8.

Una vez obtenido el valor de *offset*, lo único que hay que hacer para obtener el nivel de presión sonora en dB SPL es sumarle el *offset* al valor obtenido en dBFS. En nuestro caso, el valor de *offset* obtenido tras hacer varias pruebas fue de 119 dB. Ahora podremos obtener medidas del nivel de presión sonora en dB SPL con nuestro micrófono de forma precisa.

4.5.3. TOMA DE MUESTRAS

Una vez configurado el dispositivo de medición, está todo listo para ir a tomar muestras de sonidos ambientales reales.

Se han escogido como objeto de estudio de la contaminación acústica el centro universitario de la Escuela Técnica Superior de Ingeniería y Diseño Industrial (ETSIDI), situada en pleno foco urbano de la ciudad de Madrid, y el centro universitario de la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF) en el campus universitario de Montegancedo, ubicado en un área tranquila y lejos de la ciudad.

4.5.3.1. ETSIDI

Para la toma de muestras, nos situamos en la ventana de un laboratorio de una segunda planta que da a la calle de Ronda de Valencia, la cual conecta Embajadores con la Glorieta de Atocha, y colocamos el micrófono de manera que apunte directamente a la calle. En la Figura 4-44 se puede ver la disposición del micrófono y el tráfico de fondo.

Para empezar a grabar lo único que hay que hacer es encender la RPI y ésta empezará a tomar las muestras de audio automáticamente. Se irán tomando grabaciones de 15 minutos con un intervalo de otros 15 minutos, grabando un total de 30 minutos cada hora. La primera grabación comienza el viernes 15 de julio de 2022 a las 20:17 y la última finaliza el día miércoles 20 de julio a las 13:36, obteniendo un total de 51.075 clips de 4 segundos cuya duración en conjunto asciende a 56 horas y 45 minutos.



Figura 4-44. Toma de sonido ambiental en la ETSIDI.

Además, también se cuenta con 1080 muestras (72 minutos de duración total) obtenidas el martes 5 del mismo mes entre las 9:33 AM y las 11:41 AM desde un banco situado enfrente de las puertas del centro. Este día se estaba haciendo una reforma en uno de los laboratorios de la ETSIDI que dan a la calle y se pudieron obtener algunas muestras de taladros, sirenas de ambulancia y motores de coches y autobuses.

4.5.3.2. CAMPUS DE MONTEGANCEDO

Esta vez situamos el dispositivo de medición de nuevo en la ventana de un laboratorio de una tercera planta que da a una pequeña carretera del campus por la que de vez en cuando pasa algún autobús. Excepto por esta carretera, todo lo demás se encuentra rodeado por un campo con árboles. En la Figura 4-45 se muestra la disposición del equipo de medición con la naturaleza de fondo.

Las mediciones en esta localización se realizaron entre el viernes 8 de julio a las 11:56 AM y el lunes 11 a las 19:13, obteniendo un total de 35.775 clips de 4 segundos y una duración total de 39 horas y 45 minutos de sonido ambiental.



Figura 4-45. Toma de sonido ambiental en el campus de Montegancedo.

4.5.4. CLASIFICACIÓN DE MUESTRAS

La CNN se obtuvo después de tomar las muestras de sonido ambiental, por lo que para el procesamiento de tal magnitud de información se ha optado por utilizar de nuevo el Magerit-3. Sin embargo, el proceso para obtener la predicción de las muestras se ha implementado en una función de Python y probado posteriormente en el *script* de grabación de la RPI. De esta manera, la toma de muestras y su clasificación quedan integradas en el mismo dispositivo de medición.

Para automatizar la transferencia de archivos desde la RPI al Magerit (y viceversa, para obtener el modelo entrenado de la CNN) vía SSH, se ha utilizado el comando scp. En la Figura 4-46 se muestra la transferencia de muestras entre la RPI y Magerit.

```

pi@raspiav:~/Documents $ scp /home/pi/Desktop/Documents/example_clips/*
u828568@magerit.cesvima.upm.es:/home/u828/u828568/tfg/rpi/
Password:
2022_07_04_21_17_27.wav          100% 345KB 872.2KB/s  00:00
2022_07_04_21_17_31.wav          100% 345KB 549.3KB/s  00:00
2022_07_04_21_17_35.wav          100% 345KB 563.8KB/s  00:00
2022_07_04_21_17_39.wav          100% 345KB   1.0MB/s  00:00

pi@raspiav:~/Documents $ 

```

Figura 4-46. Transferencia de muestras entre RPI y Magerit.

Para hacer una predicción de la clase a la que pertenece cada fragmento de audio, éste se debe representar en un formato válido para la entrada del modelo entrenado. Esto es, obtener de nuevo los espectrogramas log-Mel y delta-log-Mel para cada una de las muestras a analizar, con el procedimiento descrito en el apartado 5.1.1. Sin embargo, en esta ocasión haremos una amplificación de la señal de sonido previa a la obtención del espectrograma (Anexo A, apartado A.5).

La razón de esta amplificación previa es que, tras analizar la intensidad de los audios del conjunto de datos de US8K con la función implementada en el *script* de grabación para registrar los dB (Figura 4-41), hemos obtenido una media de -26 dBFS y una desviación típica de 9. Mientras que para los datos obtenidos en la ETSIDI y en el campus de Montegancedo, las medias obtenidas han sido de -51,5 dBFS y -56,5 dBFS respectivamente, con una desviación típica de 9,7 para ambas. Por lo tanto, para que las muestras tomadas sean lo más parecidas posible al conjunto de audios utilizados para entrenar la red, se han amplificado las señales de sonido en aquellos casos que procediera para obtener unos valores de intensidad similares a los de US8K.

Una vez obtenidas las representaciones de las características de las muestras, se utilizarán como entrada del modelo, obteniendo un vector (para cada muestra) con las probabilidades de que pertenezca a cada una de las clases del conjunto de datos. Consideraremos una probabilidad mayor al 95% en alguna de las clases como una predicción positiva, en caso contrario, clasificaremos la muestra como “other”. Hemos seleccionado este umbral porque preferimos que la CNN clasifique menos muestras pero que esté segura de su clasificación, a que consiga clasificar un mayor número de muestras, pero cometiendo más errores.

Durante el proceso de clasificación de las muestras, se irá registrando en un archivo CSV los nombres de cada muestra, el valor de probabilidad máximo predicho, el nombre de la clase predicha, el identificador de dicha clase y el vector completo de todas las probabilidades de cada clase para su posterior análisis. En la Figura 4-47 se muestra el diagrama de flujo del proceso de clasificación de las muestras (Anexo A, apartado A.6).

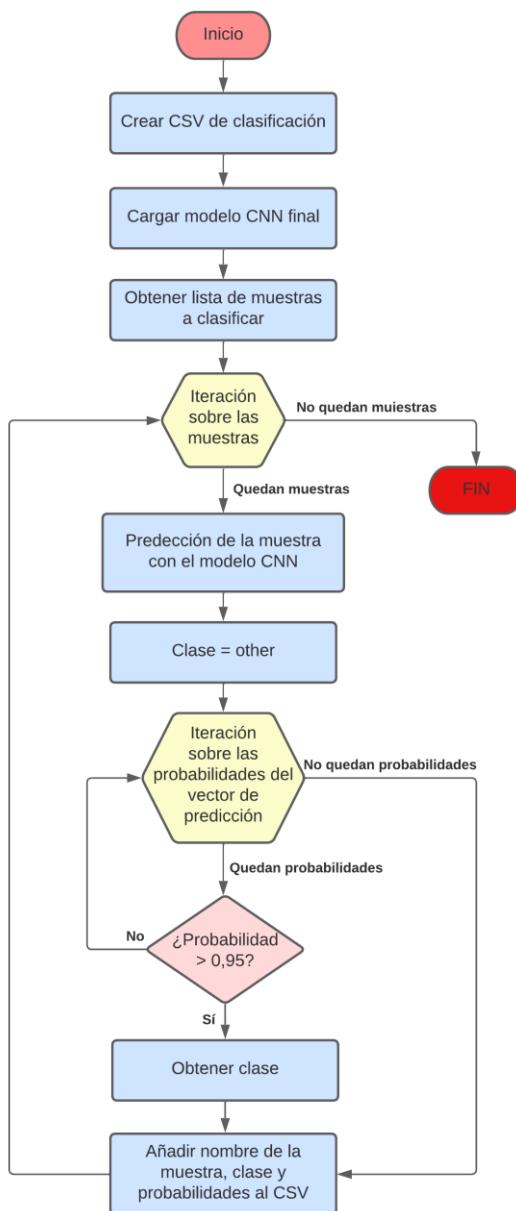


Figura 4-47. Flujograma del *script* de clasificación de muestras.

Capítulo 5

RESULTADOS

Gracias a las clasificaciones de los distintos sonidos ambientales obtenidas con nuestra CNN y al registro del nivel de intensidad sonora que nos proporciona el dispositivo de medición, podemos hacer un análisis de la situación de la contaminación acústica para cada centro universitario visitado.

En primer lugar, se hará un análisis de los resultados obtenidos para cada clase. Empezaremos comparando la frecuencia con la que se ha obtenido cada una de ellas, así como su número total de muestras, tanto para el centro universitario de la ETSIDI, como para el de la ETSIINF en el campus de Montegancedo.

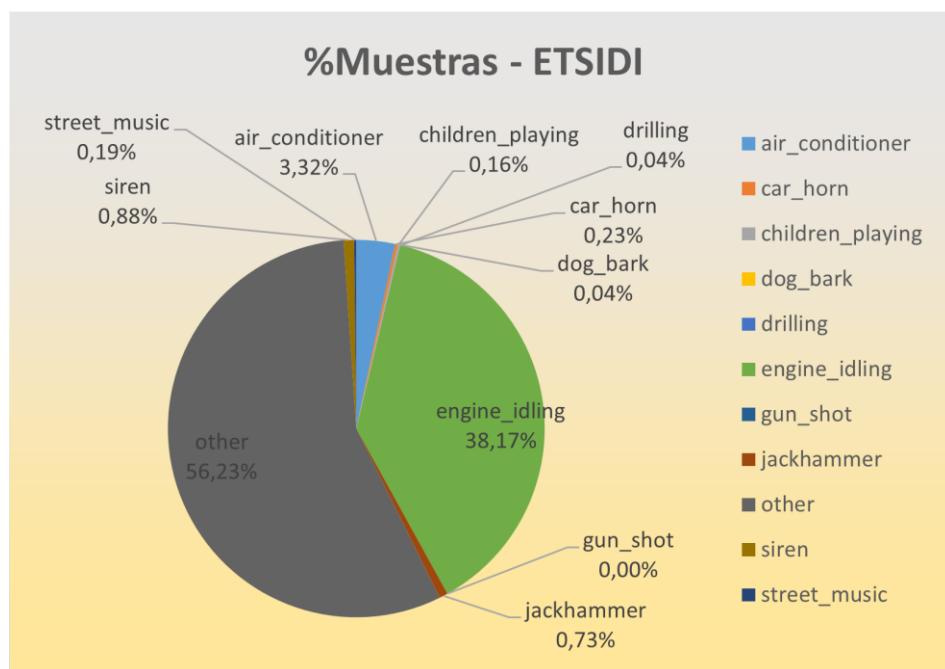


Figura 5-1. Porcentaje del número de muestras obtenidas por clase en la ETSIDI.

Para el centro universitario de la ETSIDI, nuestra CNN ha clasificado (con más de un 95% de certeza) el 43,77% de las muestras totales como sonidos urbanos (Figura 5-1). En gran parte, estas clasificaciones han sido de motores, los cuales conforman un 38,17% de las muestras totales y 87,21% de todas las clasificaciones obtenidas (Figura 5-2). Esto concuerda con los resultados esperados, ya que la calle en la que se tomaron las muestras (Ronda de Valencia) es una zona con bastante tráfico, en la que la afluencia de vehículos es prácticamente constante. Le sigue la clase de aire acondicionado, con un 3,32% de las muestras totales y un 7,58% entre todas las clasificaciones. Esto puede deberse a que a escasos metros de la ventana donde se colocó el micrófono hay varios aires acondicionados instalados en una terraza.

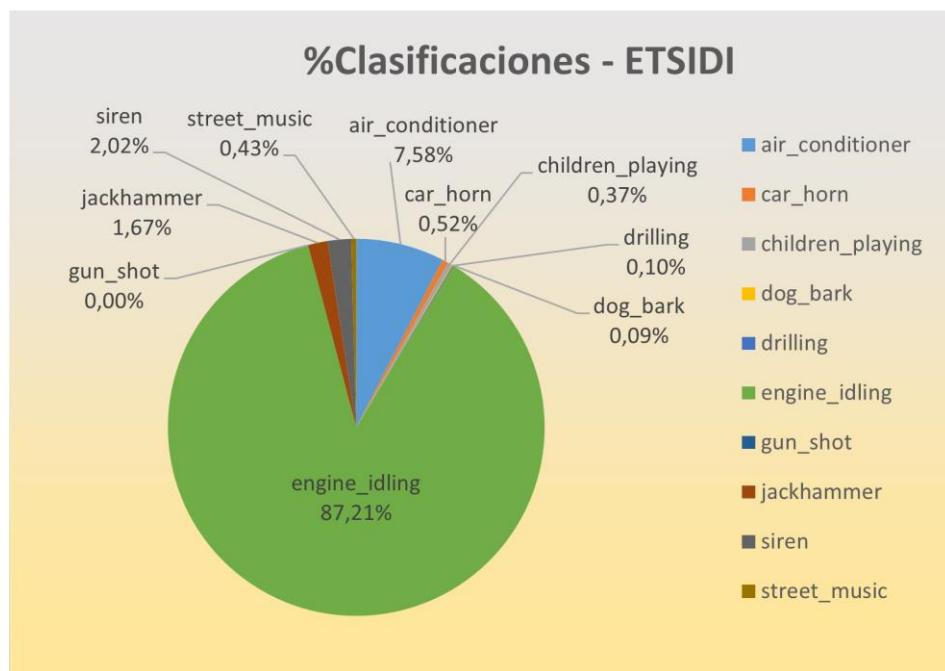


Figura 5-2. Porcentaje de las clasificaciones obtenidas en la ETSIDI.

Cabe decir que la CNN no es capaz de identificar varias clases contenidas en una misma muestra, siempre se decantará por una. Puede ser por este motivo por lo que las demás clases se hayan detectado con tan poca frecuencia, ya que quizás algunas de ellas, como la de niños jugando (voices humanas) han quedado superpuestas por el ruido del tráfico. A esto se le suma el hecho de que el micrófono no estaba colocado a pie de calle, sino en una ventana de un segundo piso, otro motivo por el cual las voces de la gente que transita la calle han podido quedar atenuadas en un segundo plano.

El resto de clases aparecen en menor medida, no llegando al 1% de las muestras totales, lo que nos indica que no son muy frecuentes, sino más bien momentáneas.

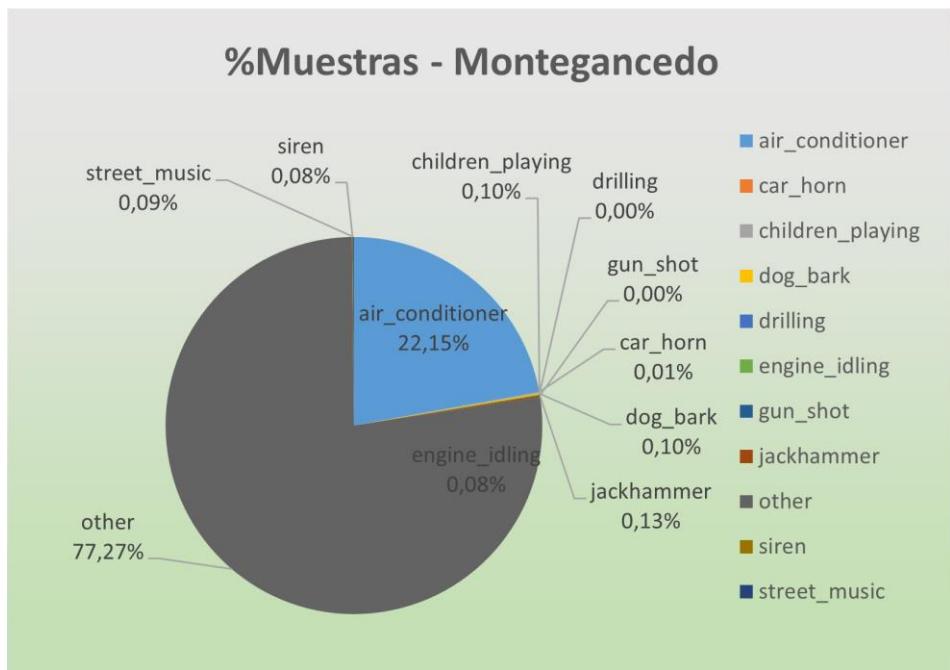


Figura 5-3. Porcentaje del número de muestras obtenidas por clase en Montegancedo.

En el campus de Montegancedo nos encontramos con un panorama muy distinto al anterior. Tan solo un 22,73% de las muestras totales han sido clasificadas (Figura 5-3). Además, estas predicciones han sido en su gran mayoría para la clase de aire acondicionado, abarcando el 97,44% de las clasificaciones totales (Figura 5-4).

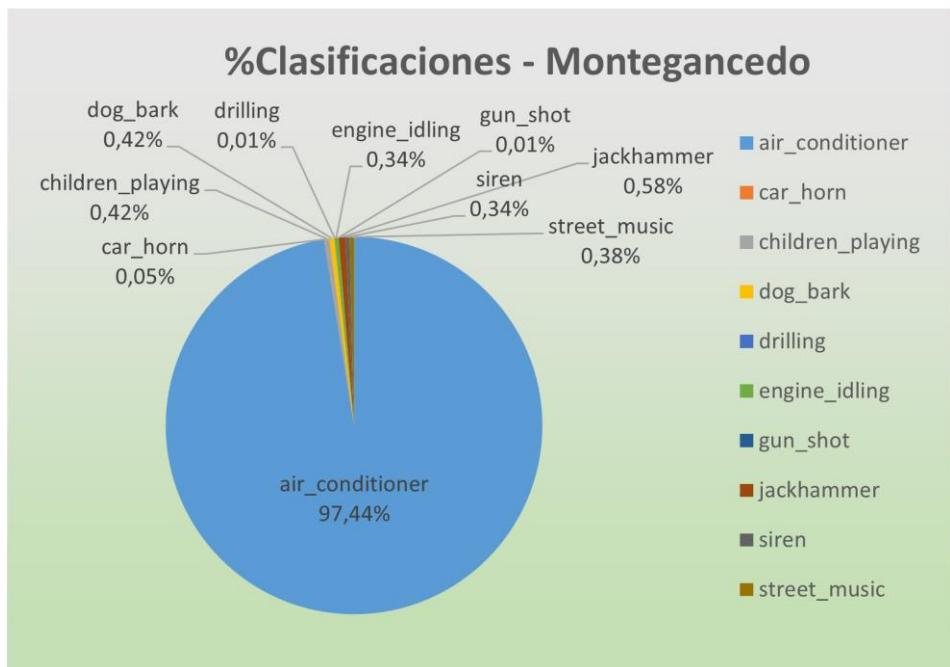


Figura 5-4. Porcentaje de las clasificaciones obtenidas en Montegancedo.

En el laboratorio en el que se instaló el dispositivo de medición había un servidor que producía un leve ruido estático, lo que ha podido ser identificado en alguna ocasión como aire acondicionado, ya que este tipo de sonidos tienden a confundirse entre ellos debido a su naturaleza. Además, las ventanas no eran dobles, por lo que mucho ruido del interior del laboratorio ha podido ser captado por el micrófono, entre ellos el del aire acondicionado. De hecho, el propio ventilador instalado en la RPI ha podido contribuir a esta clasificación, ya que, aunque estuviese alejado y detrás del micrófono, no había ninguna pared física entre ellos. Incluso el propio ruido estático del micrófono, al no estar superpuesto por ningún otro sonido ambiental, ha podido clasificarse como aire acondicionado.

A continuación, detallaremos la cantidad de muestras obtenidas para cada clase. En la Figura 5-5 se puede ver como algunas clases como ladridos de perro o perforación sólo se han detectado unas pocas de veces. La única detección hecha para la clase disparo seguramente sea un error en el que se ha clasificado un sonido parecido como un disparo, debido a que la precisión de la CNN no es perfecta.

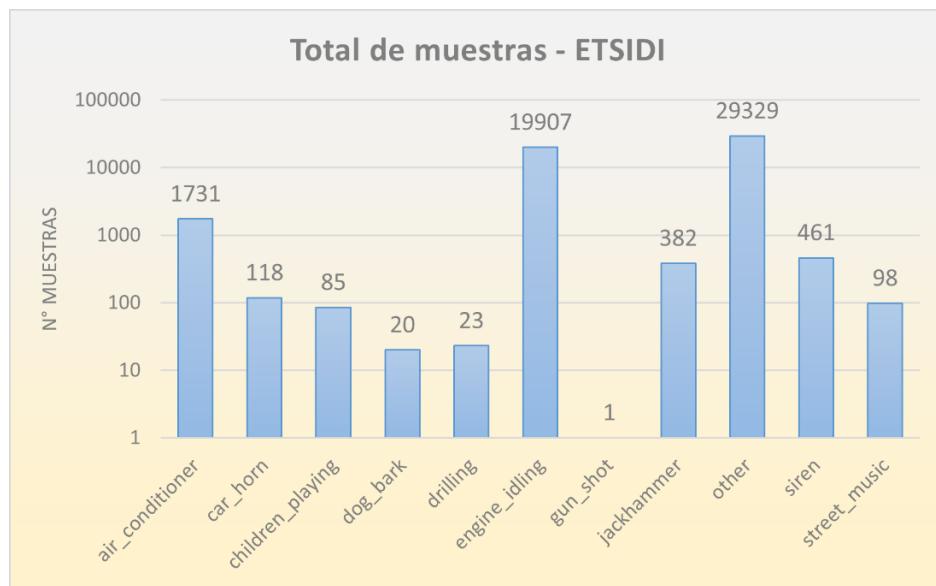


Figura 5-5. Número de muestras por clase obtenidas en la ETSIDI.

Otras clases, como el martillo neumático y la sirena, han tenido una cantidad de muestras significativas. Esto coincide con las grabaciones del primer día, en el que nos detuvimos a grabar cerca de una obra de uno de los laboratorios de la ETSIDI que dan a la calle. Además, se ha podido comprobar que por esta calle circulan también bastantes vehículos con sirenas. Los cláxones detectados también ilustran el ambiente de tráfico que hay.

Vemos también que el número de clasificaciones obtenidas en las clases de niños jugando y música callejera son muy similares. Esto puede ser un indicativo de que a la CNN le cuesta diferenciar las voces humanas entre estas dos clases.

En la Figura 5-6 se puede ver que son muy pocas las muestras clasificadas para cualquier clase que no sea la de aire acondicionado. Sin embargo, es curioso que algunas clases, como la de ladrido de perro, tenga más clasificaciones en el campus de Montegancedo que en la ETSIDI, ya que ni es una zona residencial ni está cerca de una. La mayoría de estos valores es probable que se traten de errores de precisión de la CNN.

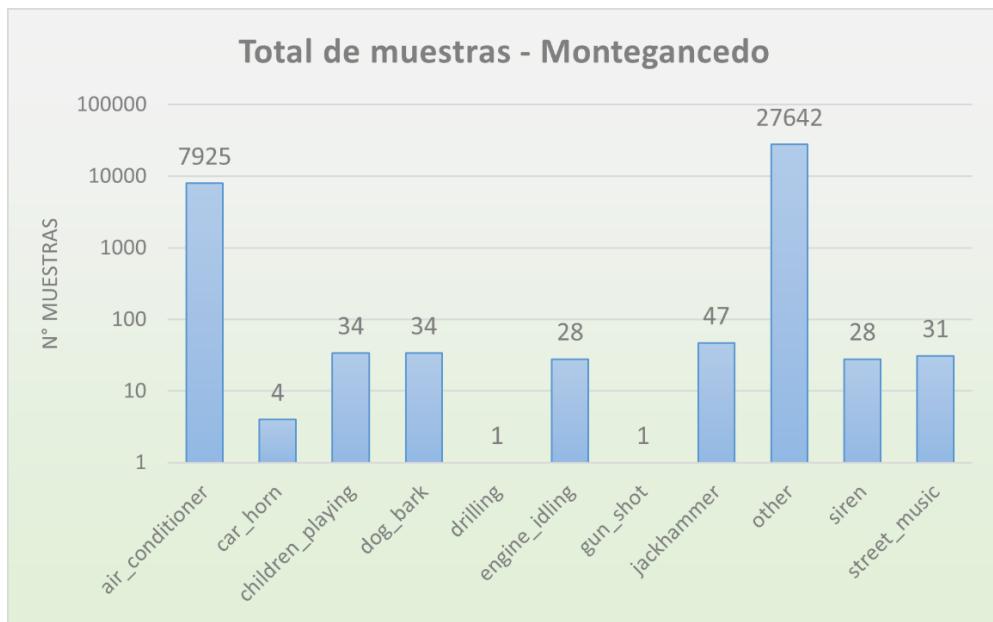


Figura 5-6. Número de muestras por clase obtenidas en Montegancedo.

Sin embargo, sí es cierto que se han escuchado ruidos de animales que se asemejan con algunos sonidos urbanos. Por ejemplo, se escuchaban urracas y cigarras que imitaban las características dinámicas de un martillo hidráulico. Es bastante probable que la red haya confundido los patrones de estos sonidos naturales con otros urbanos.

La zona en la que colocamos el micrófono estaba completamente despejada de edificios, por lo que había fuertes corrientes de aire, sobre todo por la noche. Las detecciones de la clase sirena han sido en su mayoría cambios en la intensidad del viento que golpeaba el micrófono.

Quizás también se esperaba haber captado algún motor más, ya que de vez en cuando circulan algunos coches y autobuses cerca. Lo mismo sucede con la clase de niños jugando o incluso música callejera, ya que las voces humanas suelen clasificarse en alguna de estas dos. Aunque también es cierto que las grabaciones se tomaron en su mayoría en los días de fin de semana, luego puede ser que sí que reflejen la realidad.

Para terminar con los resultados comparados por clases, vamos a analizar los niveles de intensidad sonora medios y máximos (eficaces) registrados para cada una de ellas en ambos centros de la UPM. Estos datos se han obtenido para cada muestra de 4

segundos, por lo que los resultados pueden no ser del todo acordes a la realidad, ya que el dispositivo de medición no es capaz de analizar el nivel de presión sonora de la fuente clasificada de forma aislada y sin que influyan otros sonidos contenidos en la muestra. Por lo tanto, este tipo de clasificación es bastante inexacto. Además, el nivel de presión sonora medido también depende en gran medida de la distancia entre el micrófono y todas las fuentes de sonido y la orientación de ambos.

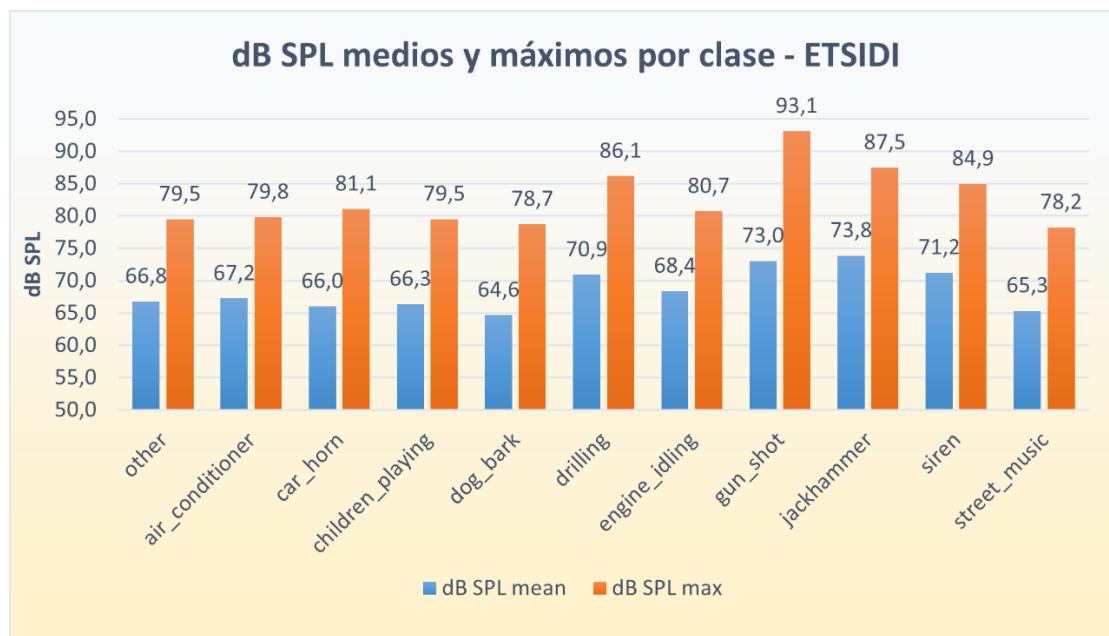


Figura 5-7. Nivel de intensidad sonora por clase para el entorno de la ETSIDI.

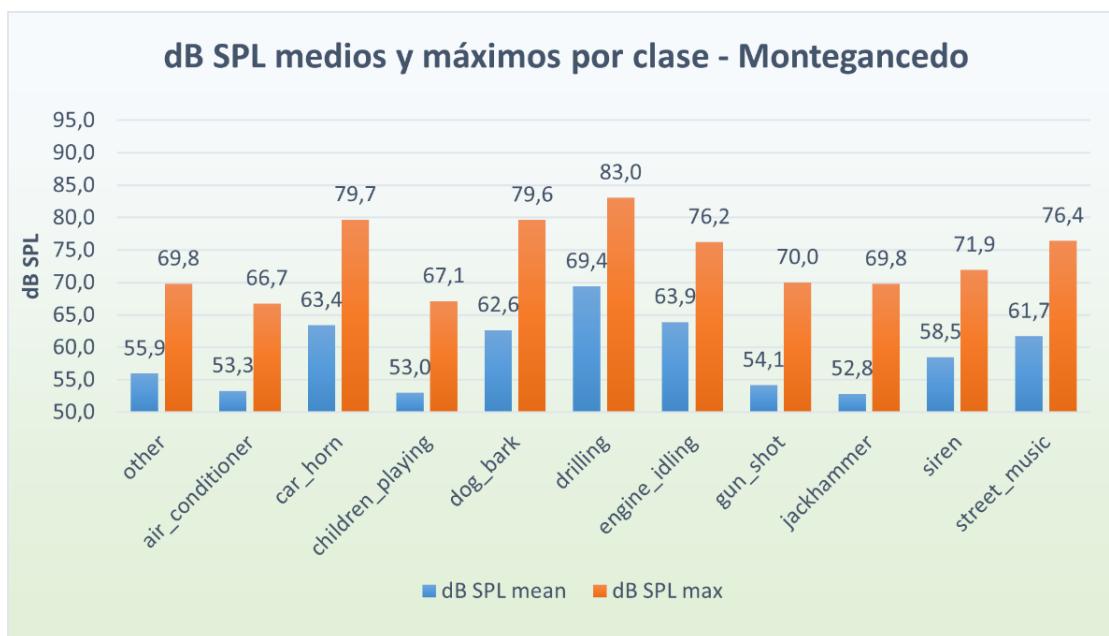


Figura 5-8. Nivel de intensidad sonora por clase para el entorno del campus de Montegancedo.

En las Figuras 5-7 y 5-8 se puede ver que las medidas obtenidas para cada clase difieren mucho entre las dos localizaciones. Además, para el campus de Montegancedo, hay saltos de intensidad muy bruscos entre las distintas clases. Debido a que la mayoría de las clases no tienen muestras suficientes como para reflejar sus características con respecto a la intensidad sonora, no se pueden sacar muchas conclusiones de estos resultados.

Sin embargo, lo que sí se puede apreciar es que los niveles de presión sonora de las muestras no clasificadas (clase *other*), las cuales son unas pocas decenas de miles en ambas localizaciones, son significativamente menores en el campus de Montegancedo.

A continuación, se va a hacer un análisis de los resultados obtenidos en función de las horas del día a las que fueron tomados. Para ello, se han agrupado todos los datos únicamente en función de la hora, sin tener en cuenta los minutos. Por ejemplo, todos los audios recogidos entre las 12:00 y las 12:59 contabilizarán únicamente para los resultados de las 12:00.

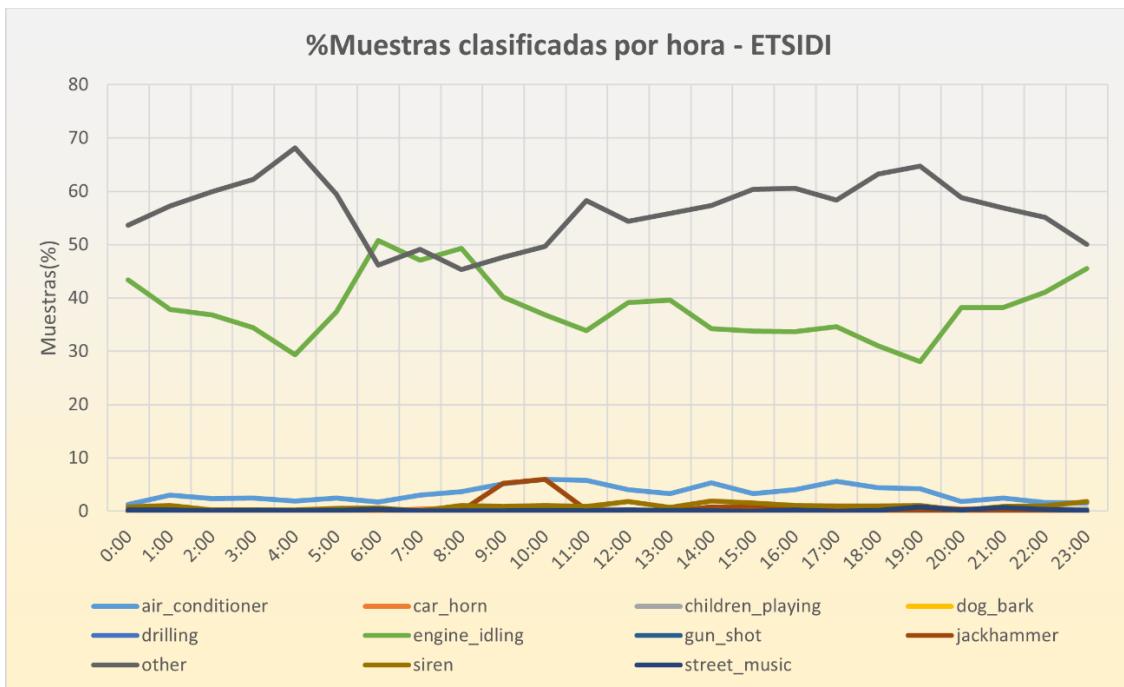


Figura 5-8. Porcentaje de cada clase por horas en la ETSIDI.

En la Figura 5-8 vemos como la clase motor apenas baja del 30% de las clasificaciones por hora, incluso durante la madrugada, lo cual es algo que nos sorprende. Alcanza sus picos máximos a las 6:00 y 8:00 de la mañana y sus mínimos a las 4:00 de la mañana y 19:00 de la tarde, variando entre un 30% y llegando hasta el 50% de las clasificaciones totales.

También se puede observar un aumento en la clase del martillo hidráulico entre las 9:00 y 11:00 de la mañana (Figura 5-9), justo en la franja en la que obtuvimos los audios el primer día cerca de las obras. Por otro lado, el aire acondicionado parece aumentar ligeramente entre las 8:00 de la mañana y las 19:00 de la tarde, lo que coincide con el horario de la universidad.

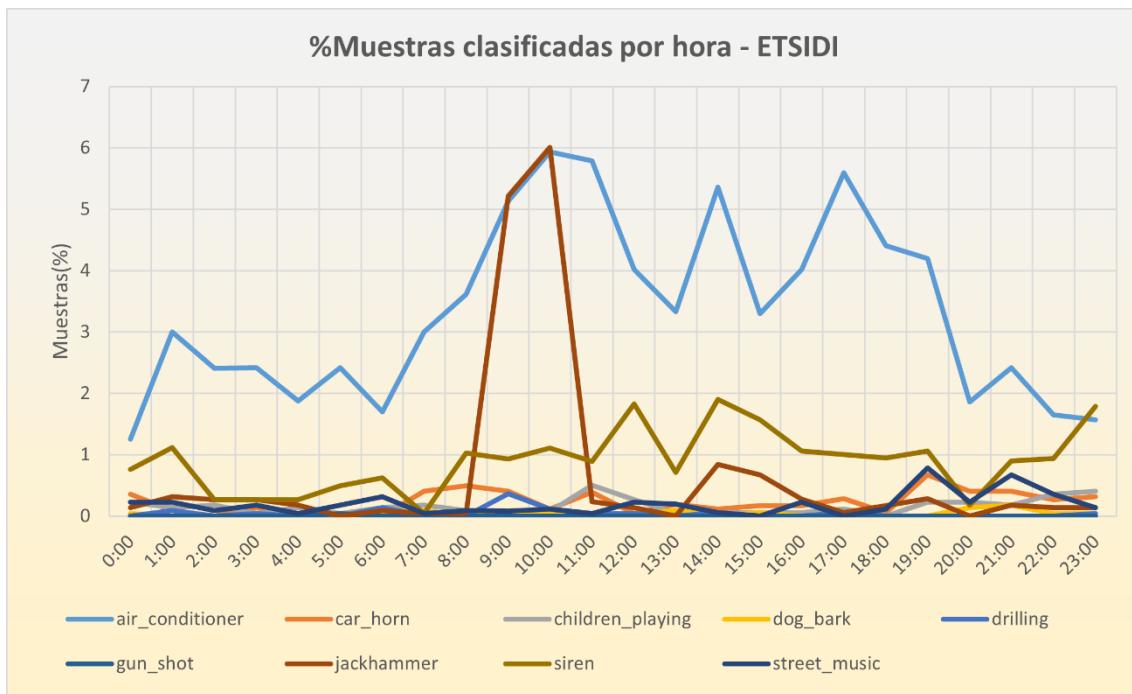


Figura 5-9. Porcentaje de cada clase por horas en la ETSIDI (sin clases motor ni other).

De la gráfica de Montegancedo no se han obtenido resultados muy relevantes (Figura 5-10). La única clase detectada a lo largo de todo el día es la del aire acondicionado. Parece que empieza a aumentar a partir de las 8:00 de la mañana y vuelve a descender a las 22:00 en picado. Esto concuerda de nuevo con el horario de la universidad. Sin embargo, durante la madrugada también se obtienen valores más o menos constantes del 10%, puede que debido al ventilador de la RPI o de algún servidor del laboratorio.

También se observan unos leves picos en la clase del martillo neumático a las 13:00 del mediodía y a las 19:00 de la tarde. Esto se debe, como comentamos anteriormente, al sonido emitido por las urracas y las cigarras, lo que nos lleva a la conclusión que todas esas clasificaciones se tomaron en unos momentos puntuales del día en el que estos animales se situaron cerca de la ventana del laboratorio.

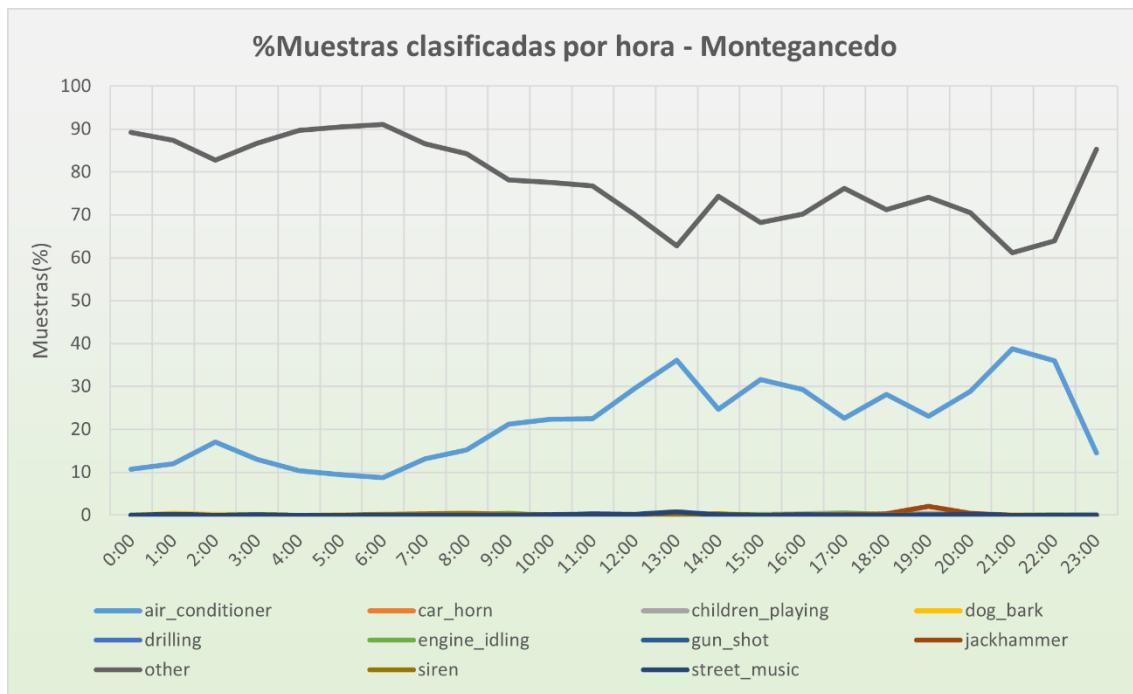


Figura 5-10. Porcentaje de cada muestra por horas en el campus de Montegancedo.

Para finalizar con el análisis de la contaminación acústica en ambos centros, vamos a estudiar los niveles de presión sonora medios y máximos obtenidos a lo largo del día.

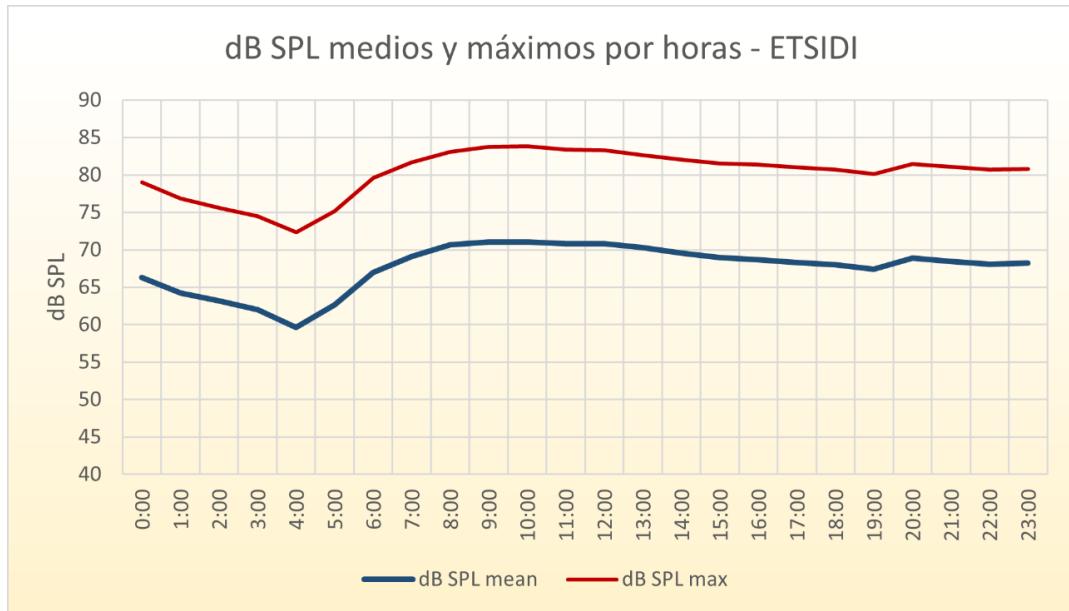


Figura 5-11. Niveles de presión sonora medios y máximos por horas en la ETSIDI.

En el caso de la ETSIDI, en la Figura 5-11 vemos cómo los niveles de presión sonora están en torno a 70 dB SPL a lo largo del día y descienden a partir de las 00:00 hasta llegar a un valor mínimo de 60 dB SPL a las 4:00 de la madrugada. La OMS [74] considera dañinos para la salud aquellos ruidos por encima de 70 dB.

Se puede considerar por tanto que el entorno de la ETSIDI es bastante ruidoso, llegando incluso a alcanzar los 84 dB SPL como valor máximo promedio. Seguramente estos niveles de ruido sean incluso mayores, debido a la distancia desde la que se tomaron las mediciones, la cual reduce de manera considerable los dB SPL medidos.

Por el contrario, los resultados obtenidos para la ETSIINF en el campus de Montegancedo (Figura 5-12) reflejan el ambiente idóneo en el que se encuentra este centro universitario.

Cabe destacar que curiosamente, los dB SPL obtenidos durante la madrugada son mayores que los obtenidos a lo largo del día. Esto se debe a que el micrófono estaba colocado en un tercer piso, sin edificios ni obstáculos que lo resguardaran de las corrientes de aire. Las noches en las que se tomaron las mediciones hubo fuertes vientos, que al impactar contra el micrófono hizo que se registraran esos niveles de presión sonora.

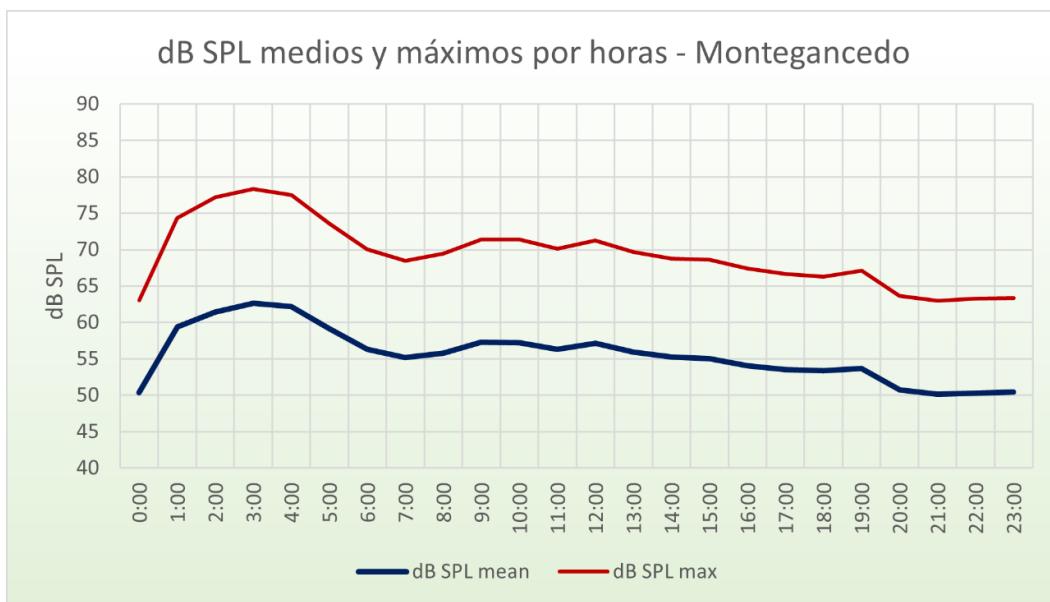


Figura 5-12. Niveles de presión sonora medios y máximos por horas en el campus de Montegancedo.

Sin embargo, los valores medios obtenidos durante el día no llegan a los 58 dB SPL, y los máximos apenas superan los 70 dB SPL. Incluso llegan a descender a los 50 dB entre las 20:00 y las 00:00. Lo que, en comparación con la ETSIDI, cuyo valor medio mínimo ha sido de 60 dB SPL, refleja que este campus tiene un entorno mucho más tranquilo y libre de contaminación acústica.

Capítulo 6

CONCLUSIONES Y PERSPECTIVAS FUTURAS

6.1. CONCLUSIONES

En este proyecto se ha comprobado que las Redes Neuronales Convolucionales son aptas para tareas de Clasificación de Sonidos Ambientales. En concreto, el modelo propuesto ha obtenido una precisión del 82,3% con una validación cruzada de 10 pliegues para el conjunto de datos de libre acceso de UrbanSound8K. Aunque en la práctica se espera que esta precisión disminuya, debido a factores como el viento, la superposición de distintas clases de sonido, la calidad del micrófono o la distancia entre éste y las fuentes emisoras.

Sin embargo, pese a que no se puede dar una estimación de la precisión final de la clasificación de nuestras muestras debido a la naturaleza del proyecto, consideramos que los resultados obtenidos reflejan la situación de la contaminación acústica en los centros universitarios de manera acorde con la realidad.

La implementación de nuestro modelo de clasificación en un dispositivo de bajo coste nos ha permitido comprobar la situación de este problema en varios centros de la UPM e identificar algunas de las fuentes de sonido que lo provocan. En concreto, en el centro universitario de la ETSIDI, situado en plena ciudad de Madrid, y en el centro universitario de la ETSIINF, ubicado en el campus de Montegancedo, lejos de la ciudad.

Además, el uso de software libre y el hardware de bajo coste hace que nuestro proyecto sea muy fácil de replicar, pudiendo aplicarse a un estudio más extenso de la contaminación acústica en el resto de centros de la UPM.

Por último, gracias a los recursos de supercomputación proporcionados por el Magerit-3, hemos podido entrenar y validar varios modelos de manera eficiente hasta

quedarnos con el mejor de ellos. Sin el acceso a estos recursos, el desarrollo del proyecto se hubiera visto mucho más limitado.

6.2. PERSPECTIVAS FUTURAS

La mayor limitación de este proyecto ha sido los conjuntos de sonidos ambientales disponibles para el entrenamiento de la red, ya que, para modelos como el nuestro, éstos se quedan cortos en cuanto a número de muestras y variabilidad de los datos. La parte más importante para obtener un modelo robusto y con una mayor precisión, es contar con un conjunto de datos lo suficientemente extenso y que refleje la mayor taxonomía de sonidos urbanos posible.

Otra manera de mejorar la precisión de la red podría ser la combinación de varias CNN en un modelo híbrido. Esta arquitectura podría utilizar la capacidad de aprendizaje de extracción de características de varias CNN por separado y utilizarlas en conjunto para la clasificación final. Esto supondría entrenar cada modelo de CNN de manera considerablemente diferente para garantizar la variabilidad de las características extraídas por cada una de ellas. Por ejemplo, utilizando distintas representaciones de características del sonido, distinto número y tamaño de capas, o distintas configuraciones de hiperparámetros en general. Como clasificadores podrían utilizarse redes neuronales (capas de neuronas totalmente conectadas) e incluso clasificadores tradicionales como KNN o SVM.

Además, en este proyecto se han utilizado unas representaciones de las características del sonido (espectrogramas log-Mel y delta-log-Mel) y una arquitectura de red neuronal (CNN) determinadas. Pero hay otros tipos de representaciones y arquitecturas de redes neuronales que pueden dar incluso mejores resultados.

En cuanto a la toma de muestras, considero que el factor más relevante a la hora de tomar audios de calidad es la distancia entre el micrófono y las fuentes sonoras. En este proyecto, las muestras se han grabado dejando el dispositivo de medición en la ventana de un tercer y segundo piso. Pero colocarlo a un nivel de metro y medio sobre el suelo, cerca de las fuentes de contaminación acústica urbanas, sin duda mejorará de manera considerable la calidad de las muestras y, en consecuencia, su clasificación.

Finalmente, como el objetivo principal de este proyecto es evaluar la contaminación acústica en los distintos centros de la UPM, además de las posibles mejoras del modelo, también se podría expandir el campo de estudio a otros centros y campus universitarios, con el fin de mejorar la sostenibilidad ambiental acústica en ellos.

BIBLIOGRAFÍA

- [1] “Contaminación acústica.” <https://www.miteco.gob.es/es/calidad-y-evaluacion-ambiental/temas/atmosfera-y-calidad-del-aire/contaminacion-acustica/> (accessed Mar. 30, 2022).
- [2] “La contaminación acústica - Ecologistas en Acción.” <https://www.ecologistasenaccion.org/5350/la-contaminacion-acustica/> (accessed Mar. 31, 2022).
- [3] WHO and JRC, “Burden of Disease from Environmental Noise Burden of disease from environmental noise Quantification of healthy life years lost in Europe,” 2011.
- [4] E. Peris, “Environmental noise in Europe - 2020,” 2020.
- [5] European Environmental Agency, “The European Environment - State and Outlook 2020. Chapter 11: Environmental noise,” 2020.
- [6] S. A. Stansfeld and M. P. Matheson, “Noise pollution: Non-auditory effects on health,” *British Medical Bulletin*, vol. 68. pp. 243–257, 2003. doi: 10.1093/bmb/lgd033.
- [7] “Contaminación Acústica: qué es, causas, efectos y soluciones - Iberdrola.” <https://www.iberdrola.com/sostenibilidad/que-es-contaminacion-acustica-causas-efectos-soluciones> (accessed Mar. 30, 2022).
- [8] Health Canada, *Guidance for evaluating human health impacts in environmental assessment: noise*. 2017.
- [9] A. M. Turing, “COMPUTING MACHINERY AND INTELLIGENCE,” *Computing Machinery and Intelligence. Mind*, vol. 49, pp. 433–460, 1950.
- [10] ““Machine learning’: ¿qué es y cómo funciona?” <https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/> (accessed Apr. 04, 2022).
- [11] “Dot CSV - YouTube.” <https://www.youtube.com/c/DotCSV> (accessed Apr. 01, 2022).

- [12] “Artificial Intelligence Technologies and their categories • AuraQuantic.” <https://www.auraquantic.com/artificial-intelligence-technologies-and-their-categories/> (accessed Apr. 01, 2022).
- [13] “CONVOCATORIA DE BECAS DE TRABAJO DE FIN DE GRADO Y MASTER, CAMPUS SOTENIBLE 2022 | Sostenibles.” <https://sostenibilidad.upm.es/convocatoria-tft-campus-sostenible-2022/> (accessed Aug. 24, 2022).
- [14] “Plan de Sostenibilidad Ambiental UPM | Sostenibles.” <https://sostenibilidad.upm.es/plan-de-sostenibilidad-ambiental/> (accessed Aug. 24, 2022).
- [15] L. Vujošević and S. Dukanović, “Deep learning-based classification of environmental sounds,” *2021 25th International Conference on Information Technology, IT 2021*, Feb. 2021, doi: 10.1109/IT51528.2021.9390124.
- [16] W. Mu, B. Yin, X. Huang, J. Xu, and Z. Du, “Environmental sound classification using temporal-frequency attention based convolutional neural network,” *Scientific Reports 2021 11:1*, vol. 11, no. 1, pp. 1–14, Nov. 2021, doi: 10.1038/s41598-021-01045-4.
- [17] S. Agarwal, K. Khatter, and D. Relan, “Security threat sounds classification using neural network,” *Proceedings of the 2021 8th International Conference on Computing for Sustainable Global Development, INDIACom 2021*, pp. 690–694, Mar. 2021, doi: 10.1109/INDIACOM51348.2021.00122.
- [18] J. C. Wang, H. P. Lee, J. F. Wang, and C. B. Lin, “Robust environmental sound recognition for home automation,” *IEEE Transactions on Automation Science and Engineering*, vol. 5, no. 1, pp. 25–31, Jan. 2008, doi: 10.1109/TASE.2007.911680.
- [19] E. Baum, M. Harper, R. Alicea, and C. Ordóñez, “Sound identification for fire-fighting mobile robots,” *Proceedings - 2nd IEEE International Conference on Robotic Computing, IRC 2018*, vol. 2018-January, pp. 79–86, Apr. 2018, doi: 10.1109/IRC.2018.00020.
- [20] S. K. Shah, Z. Tariq, and Y. Lee, “IoT based Urban Noise Monitoring in Deep Learning using Historical Reports,” *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*, pp. 4179–4184, Dec. 2019, doi: 10.1109/BIGDATA47090.2019.9006176.
- [21] H. Shu, Y. Song, and H. Zhou, “Assessment of Music and Water Sounds for Urban Noise Masking,” *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, vol. 2018-October, pp. 1451–1455, Feb. 2019, doi: 10.1109/TENCON.2018.8650456.
- [22] K. Z. Thwe and N. War, “Environmental sound classification based on time-frequency representation,” *Proceedings - 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and*

Parallel/Distributed Computing, SNPD 2017, pp. 251–255, Aug. 2017, doi: 10.1109/SNPD.2017.8022729.

- [23] F. Beritelli and R. Grasso, “A pattern recognition system for environmental sound classification based on MFCCS and neural networks,” *2nd International Conference on Signal Processing and Communication Systems, ICSPCS 2008 - Proceedings*, 2008, doi: 10.1109/ICSPCS.2008.4813723.
- [24] Z. Mushtaq and S. F. Su, “Environmental sound classification using a regularized deep convolutional neural network with data augmentation,” *Applied Acoustics*, vol. 167, Oct. 2020, doi: 10.1016/J.APACOUST.2020.107389.
- [25] V. Boddapati, A. Petef, J. Rasmusson, and L. Lundberg, “Classifying environmental sounds using image recognition networks,” *Procedia Comput Sci*, vol. 112, pp. 2048–2056, Jan. 2017, doi: 10.1016/J.PROCS.2017.08.250.
- [26] J. T. Geiger and K. Helwani, “Improving event detection for audio surveillance using Gabor filterbank features,” *2015 23rd European Signal Processing Conference, EUSIPCO 2015*, pp. 714–718, Dec. 2015, doi: 10.1109/EUSIPCO.2015.7362476.
- [27] J. C. Wang, J. F. Wang, K. W. He, and C. S. Hsu, “Environmental sound classification using hybrid SVM/KNN classifier and MPEG-7 audio low-level descriptor,” *IEEE International Conference on Neural Networks - Conference Proceedings*, pp. 1731–1735, 2006, doi: 10.1109/IJCNN.2006.246644.
- [28] A. Pareta, S. Taran, V. Bajaj, and A. Sengur, “Automatic Environment Sounds Classification Using Optimum Allocation Sampling,” *2019 4th International Conference on Robotics and Automation Engineering, ICRAE 2019*, pp. 69–73, Nov. 2019, doi: 10.1109/ICRAE48301.2019.9043832.
- [29] P. K. Atrey, N. C. Maddage, and M. S. Kankanhalli, “Audio based event detection for multimedia surveillance,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 5, 2006, doi: 10.1109/ICASSP.2006.1661400.
- [30] M. A. Casey, “Reduced-Rank Spectra and Minimum-Entropy Priors as Consistent and Reliable Cues for Generalized Sound Recognition,” 2001.
- [31] P. J. Ting, S. J. Ruan, and L. P. H. Li, “Environmental Noise Classification with Inception-Dense Blocks for Hearing Aids,” *Sensors 2021, Vol. 21, Page 5406*, vol. 21, no. 16, p. 5406, Aug. 2021, doi: 10.3390/S21165406.
- [32] A. J. Eronen *et al.*, “Audio-based context recognition,” *IEEE Trans Audio Speech Lang Process*, vol. 14, no. 1, pp. 321–329, Jan. 2006, doi: 10.1109/TSA.2005.854103.

- [33] J. Salamon, C. Jacoby, and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research," *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, doi: 10.1145/2647868.
- [34] J. Salamon and J. P. Bello, "Unsupervised feature learning for urban sound classification," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2015-August, pp. 171–175, Aug. 2015, doi: 10.1109/ICASSP.2015.7177954.
- [35] J. Salamon and J. P. Bello, "Feature learning with deep scattering for urban sound analysis," *2015 23rd European Signal Processing Conference, EUSIPCO 2015*, pp. 724–728, Dec. 2015, doi: 10.1109/EUSIPCO.2015.7362478.
- [36] M. Huzaifah, "Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks," Jun. 2017, [Online]. Available: <http://arxiv.org/abs/1706.07156>
- [37] M. G. Ragab, S. J. Abdulkadir, N. Aziz, H. Alhussian, A. Bala, and A. Alqushaibi, "An Ensemble One Dimensional Convolutional Neural Network with Bayesian Optimization for Environmental Sound Classification," *Applied Sciences 2021*, Vol. 11, Page 4660, vol. 11, no. 10, p. 4660, May 2021, doi: 10.3390/APP11104660.
- [38] E. R. Arce-Santana, A. Alba, M. O. Mendez, and V. Arce-Guevara, "A-phase classification using convolutional neural networks," *Med Biol Eng Comput*, vol. 58, no. 5, pp. 1003–1014, May 2020, doi: 10.1007/S11517-020-02144-6/TABLES/5.
- [39] K. J. Piczak, "Environmental sound classification with convolutional neural networks," *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, vol. 2015-November, Nov. 2015, doi: 10.1109/MLSP.2015.7324337.
- [40] J. Salamon and J. P. Bello, "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification," *IEEE Signal Process Lett*, vol. 24, no. 3, pp. 279–283, Mar. 2017, doi: 10.1109/LSP.2017.2657381.
- [41] Z. Chi, Y. Li, and C. Chen, "Deep Convolutional Neural Network Combined with Concatenated Spectrogram for Environmental Sound Classification," *Proceedings of IEEE 7th International Conference on Computer Science and Network Technology, ICCSNT 2019*, pp. 251–254, Oct. 2019, doi: 10.1109/ICCSNT47585.2019.8962462.
- [42] F. Demir, M. Turkoglu, M. Aslan, and A. Sengur, "A new pyramidal concatenated CNN approach for environmental sound classification," *Applied Acoustics*, vol. 170, p. 107520, Dec. 2020, doi: 10.1016/J.APACOUST.2020.107520.

- [43] F. Demir, D. A. Abdullah, and A. Sengur, "A New Deep CNN Model for Environmental Sound Classification," *IEEE Access*, vol. 8, pp. 66529–66537, 2020, doi: 10.1109/ACCESS.2020.2984903.
- [44] F. Medhat, D. Chesmore, and J. Robinson, "Masked conditional neural networks for automatic sound events recognition," *Proceedings - 2017 International Conference on Data Science and Advanced Analytics, DSAA 2017*, vol. 2018-January, pp. 389–394, Jul. 2017, doi: 10.1109/DSAA.2017.43.
- [45] F. Medhat, D. Chesmore, and J. Robinson, "Masked Conditional Neural Networks for sound classification," *Appl Soft Comput*, vol. 90, p. 106073, May 2020, doi: 10.1016/J.ASOC.2020.106073.
- [46] S. Abdoli, P. Cardinal, and A. L. Koerich, "End-to-End Environmental Sound Classification using a 1D Convolutional Neural Network".
- [47] Z. Wu and P. D. Christofides, "Economic machine-learning-based predictive control of nonlinear systems," *Mathematics*, vol. 7, no. 6, Jun. 2019, doi: 10.3390/MATH7060494.
- [48] I. Lezhenin, N. Bogach, and E. Pyshkin, "Urban sound classification using long short-term memory neural network," *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019*, pp. 57–60, Sep. 2019, doi: 10.15439/2019F185.
- [49] "Convolutional Neural Networks(CNN) #1 Kernel, Stride, Padding - BrilliantCode.net." <https://www.brilliantcode.net/1584/convolutional-neural-networks-1-convolution-layer-stride-padding-kernel/> (accessed Aug. 22, 2022).
- [50] "Convolutional Neural Network: Feature Map and Filter Visualization | by Renu Khandelwal | Towards Data Science." <https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c> (accessed Aug. 22, 2022).
- [51] K. Delgado, S. Ledesma, and H. Rostro, "Análisis de electroencefalograma usando redes neuronales artificiales," *Acta Univ*, vol. 29, pp. 1–24, Apr. 2019, doi: 10.15174/AU.2019.1672.
- [52] "Optimizadores en redes neuronales profundas: un enfoque práctico | by Luis Velasco | Medium." <https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5> (accessed Aug. 23, 2022).
- [53] "Conoce qué son las funciones de activación y cómo puedes crear tu función de activación usando Python, R y Tensorflow - Jahaziel Ponce." <https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tensorflow/> (accessed Aug. 23, 2022).

- [54] “A Brief Overview of Recurrent Neural Networks (RNN) - Analytics Vidhya.” <https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/> (accessed Aug. 23, 2022).
- [55] “Overfitting vs Underfitting in Machine Learning [Differences].” <https://www.v7labs.com/blog/overfitting-vs-underfitting> (accessed Aug. 24, 2022).
- [56] “Introducción a Sobreajuste y Subajuste para Machine Learning - Aprende IA.” <https://aprendeia.com/sobreajuste-y-subajuste-en-machine-learning/> (accessed Aug. 24, 2022).
- [57] “Técnicas de Regularización Básicas para Redes Neuronales | by Jaime Durán | MetaDatos | Medium.” <https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%ADasicas-para-redes-neuronales-b48f396924d4> (accessed Aug. 24, 2022).
- [58] “Dropout y Batch Normalization.” <https://vincentblog.xyz/posts/dropout-y-batch-normalization> (accessed Aug. 26, 2022).
- [59] “Sonido - Concepto, características y propiedades.” <https://concepto.de/sonido/> (accessed Jul. 10, 2022).
- [60] “Conceptos básicos del ruido ambiental - SICA.” <https://sicaweb.cedex.es/wp-content/uploads/2021/08/Conceptos-Basicos-del-ruido-ambiental.pdf> (accessed Jul. 10, 2022).
- [61] “Transformada de Fourier: qué es y cómo se calcula.” <https://www.nobbot.com/educacion/que-es-la-transformada-de-fourier-y-para-que-sirve/> (accessed Aug. 24, 2022).
- [62] D. Benítez and D. Ruiz, *Técnico Superior de Sonido - Primer Curso*, 1st ed. SacroVibz Publishing, 2018.
- [63] “Thomann – Microphone Designs.” https://www.thomann.de/es/onlineexpert_page_stage_vocal_mics_microphone_designs.html (accessed Jul. 11, 2022).
- [64] “Clasificación de Micrófonos - ickrom.” <https://ickrom.com.mx/2014/10/clasificacion-de-microfonos/> (accessed Jul. 11, 2022).
- [65] “TODO MICRÓFONOS.” <https://todomicrofonos.com/> (accessed Jul. 11, 2022).
- [66] “Magerit-3 - CeSViMa.” <https://www.cesvima.upm.es/services/hpc/magerit> (accessed Jul. 08, 2022).
- [67] “Manual Omnitronic MIC MM-2USB .” <https://www.steinigke.de/download/13030923-Manual-107451-1.000->

omnitronic-mic-mm-2usb-usb-condenser-measurement-mic-de_en.pdf
(accessed Jul. 09, 2022).

- [68] “Magerit-3 - CeSViMa.” <https://www.cesvima.upm.es/13-es/infrastructure/10-magerit-3> (accessed Jul. 08, 2022).
- [69] “Descripción general: Documentación CeSViMa.”
<https://docs.cesvima.upm.es/magerit/magerit3/> (accessed Aug. 26, 2022).
- [70] “Aplicaciones: Documentación CeSViMa.”
<https://docs.cesvima.upm.es/magerit/apps/> (accessed Aug. 26, 2022).
- [71] “Ejecucion de trabajos: Documentación CeSViMa.”
<https://docs.cesvima.upm.es/magerit/jobs/> (accessed Aug. 26, 2022).
- [72] “A detailed example of data generators with Keras.”
<https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>
(accessed Aug. 27, 2022).
- [73] “UrbanSound8K - Urban Sound Datasets.”
<https://urbansounddataset.weebly.com/urbansound8k.html> (accessed Jul. 13, 2022).
- [74] WHO, “NOISE GUIDELINES for the European Region,” 2018, Accessed: Aug. 30, 2022. [Online]. Available: <http://www.euro.who.int/pubrequest>

Anexo A

CÓDIGO

En este anexo se incluyen los *scripts* utilizados en el desarrollo del proyecto.

A.1. *SCRIPT DE INICIO DE LA RPI*

```
#!/bin/bash

cd /home/pi/Documents

if [ ! -f "running.txt" ];
then
    echo "1" >> running.txt
    ifconfig_file=./ifconfig/ifconfig_`date +"%d_%m_%Y"`.txt
    echo `date +"%d/%m/%Y" ` `date +"%H:%M" ` >> $ifconfig_file
    ifconfig >> $ifconfig_file
    echo "-----" >> $ifconfig_file
    record_audio_file=./DriveAPI/record_audio_wDrive.py
    log_file=./logs/log_`date +"%d_%m_%Y"`.txt
    echo `date +"%d/%m/%Y" ` `date +"%H:%M" ` >> $log_file
    echo "Checking wlan0 connection..." >> $log_file
    connection=0
    count=0
    while [ $connection -eq 0 ] && [ $count -lt 6 ]
    do
        if [ "$(ifconfig | grep -o 'wlan0.*RUNNING.*')" = "" ];
        then
            echo "connection wlan0: NOT OK ($count)" >> $log_file
            let count++
            sleep 10
        else
            ifconfig >> $log_file
        fi
    done
fi
```

```

        echo "connection wlan0: OK" >> $log_file
        connection=1
        echo "Uploading $ifconfig_file to Drive..." >> $log_file
        nohup python ./DriveAPI/upload_ifconfig.py >> $log_file
        echo "-----" >> $log_file
    fi
done

echo "Checking that the USB Microphone is plugged in..." >> $log_file
mic_connected=0
count=0
while [ $mic_connected -eq 0 ] && [ $count -lt 10 ]
do
    if [ "$(cat /proc/asound/cards | grep -o 'USB-Audio - Usb Microphone')" = "USB-Audio - Usb Microphone" ];
    then
        echo "USB Microphone OK" >> $log_file
        mic_connected=1
        if [ "$(cat /proc/asound/cards | grep -o '1.1')" = "1.1" ];
        then
            echo "900/900/500" > recording_config.txt
        elif [ "$(cat /proc/asound/cards | grep -o '1.2')" = "1.2" ];
        then
            echo "30/120/1" > recording_config.txt
        elif [ "$(cat /proc/asound/cards | grep -o '1.3')" = "1.3" ];
        then
            echo "900/900/300" > recording_config.txt
        elif [ "$(cat /proc/asound/cards | grep -o '1.4')" = "1.4" ];
        then
            echo "10/12/2" > recording_config.txt
        else
            echo "900/900/500" > recording_config.txt
        fi
        chmod 755 $record_audio_file
        echo "Executing $record_audio_file" >> $log_file
        echo "-----" >> $log_file
        nohup python $record_audio_file >> $log_file
        echo "-----" >> $log_file
    else
        echo "USB Microphone NOT OK ($count)" >> $log_file
        let count++
        sleep 10
    fi
done

echo " - Bash Script Finished at `date +"%d/%m/%Y"` `date +"%H:%M"`"
>> $log_file
if [ $connection = 1 ];
then

```

```

        echo "Uploading $log_file to Drive..." >> $log_file
        echo "*****" >> $log_file
        nohup python ./DriveAPI/upload_log.py
    else
        echo "*****" >> $log_file
    fi

    if [ -f "running.txt" ];
    then
        rm running.txt
    fi
fi

```

A.2. *SCRIPT DE GRABACIÓN*

```

# INICIAR SESIÓN
def login():
    GoogleAuth.DEFAULT_SETTINGS['client_config_file'] = credentials_path
    gauth = GoogleAuth()
    gauth.LoadCredentialsFile(credentials_path)

    if gauth.credentials is None:
        gauth.LocalWebserverAuth(port_numbers=[8092])
    elif gauth.access_token_expired:
        gauth.Refresh()
    else:
        gauth.Authorize()

    gauth.SaveCredentialsFile(credentials_path)
    credenciales = GoogleDrive(gauth)
    return credenciales
#-----#
# CREAR UN ARCHIVO DE TEXTO EN DRIVE
def create_txt_file(file_name, text, id_folder):
    try:
        credenciales = login()
        file = credenciales.CreateFile({'title': file_name,\n                                         'parents': [{"kind":\n                                         "drive#fileLink"},\n                                         "id":\n                                         id_folder}])
        file.SetContentString(text)
        file.Upload()
    except:

```

```

        print('Error: could not create ' + file_name)
#-----
# SUBIR UN ARCHIVO A DRIVE
def upload_file(file_path, id_folder):
    try:
        credenciales = login()
        file = credenciales.CreateFile({'parents': [{"kind": "drive#fileLink", \
                                                "id": id_folder}]}))
        file['title'] = file_path.split('/')[-1]
        file.SetContentFile(file_path)
        file.Upload()
    except:
        print('Error: could not upload ' + file_path)
#-----
def update_log_drive(file_name, text, id_folder):
    update_pylog_local('/home/pi/Documents/logs/' +
str(datetime.now().date()).replace('-', '_') + '_pylog_local.txt', text,
id_folder)
    try:
        credenciales = login()
        query = """ + id_folder + ' in parents and title contains ' + \
file_name + ' and trashed = false"
        file_list = credenciales.ListFile({'q': query}).GetList()
        if len(file_list) == 0:
            print('Creating ' + file_name + ' in Drive')
            create_txt_file(file_name, 'Creating ' + file_name + ' in \
Drive\n' + text, id_folder)
        else:
            for f in file_list:
                fname = f['title']
                mystring = f.GetContentString() + '\n[' + \
str(datetime.now().time()) + ']: ' + text
                f.Delete()
                create_txt_file(fname, mystring, id_folder)
    except:
        print('Error: could not update ' + file_name)
#-----
def upload_pylog_local2drive(file_path, id_folder):
    try:
        credenciales = login()
        query = """ + id_folder + ' in parents and title contains ' + \
file_path.split('/')[ -1] + ' and trashed = false"
        file_list = credenciales.ListFile({'q': query}).GetList()
        if not (len(file_list) == 0):
            for f in file_list:
                f.Delete()
        upload_file(file_path, id_folder)

```

```

        except:
            print('Error: could not upload ' + file_path)
#-----
def update_pylog_local(pylog_path, text, id_folder):
    try:
        pylog_file = open(pylog_path, "a+")
        pylog_file.write('\n[' + str(datetime.now().time()) + ']: ' +
text)
        pylog_file.close()
    except:
        print('Error: could not update ' + pylog_path)
        upload_pylog_local2drive(pylog_path, id_folder)
#-----
def extract_features(clips_dir, save_dir):
    mels = 128
    all_clips = os.listdir(clips_dir)
    for nclip in all_clips:
        print('extracting features from: ' + clips_dir + nclip)
        snd, sr = librosa.load(clips_dir + nclip)
        dBFS = 20 * np.log10(np.sqrt(np.mean(snd**2)))
        if dBFS < -40.0:
            snd = snd * 10**((-26.0 - dBFS)/20)
        snd[snd > 1.0] = 1.0
        snd[snd < -1.0] = -1.0
        if len(snd) == 88200:
            melspec = librosa.feature.melspectrogram(y=snd, sr=sr,
n_mels=mels)
            logmelspec = librosa.amplitude_to_db(melspec, ref=np.max)
            delta_logmelspec = librosa.feature.delta(logmelspec)

            features = np.asarray(logmelspec[:, :, np.newaxis])
            features = np.concatenate((features,
np.zeros(np.shape(features))), axis=2)
            features[:, :, 1] = delta_logmelspec

            np.save(save_dir + nclip, features)
#-----
def record_audio(file_path, duration = 480):
    #DEFINICIÓN DE PARÁMETROS
    CHANNELS = 1
    CHUNK = 1024
    FORMAT = pyaudio.paInt32
    CHANNELS = 1
    RATE = 22050

    #INICIAR "pyaudio"
    audio = pyaudio.PyAudio()
    print("Recording audio...")
    update_log_drive(pylog_file, "Recording audio...", folder_id)

```

```

#INICIAR GRABACIÓN
stream = audio.open(format = FORMAT, channels = CHANNELS,
                     rate = RATE, input = True,
                     frames_per_buffer = CHUNK)
frames = []
for i in range(0, int(RATE / CHUNK * duration)):
    data=stream.read(CHUNK)
    frames.append(data)

#DETENER GRABACIÓN
stream.stop_stream()
stream.close()
audio.terminate()

#CREAR/GUARDAR EL ARCHIVO DE AUDIO
waveFile = wave.open(file_path, 'wb')
waveFile.setnchannels(CHANNELS)
waveFile.setsampwidth(audio.get_sample_size(FORMAT))
waveFile.setframerate(RATE)
waveFile.writeframes(b''.join(frames))
waveFile.close()
print("Recording finished")
update_log_drive(pylog_file, "Recording finished", folder_id)
#-----
def get_dB(file_path):
    snd , sr = librosa.load(file_path)

    max_value = np.max(np.abs(snd))
    if max_value > 1 :
        snd = snd / max_value

    offset = 119
    rms_val = np.sqrt(np.mean(snd**2))

    dBFS = 20 * np.log10(rms_val)
    dB SPL = 20 * np.log10(rms_val) + offset
    dBFSmax = np.max(20 * np.log10(abs(snd)))
    dB SPLmax = dBFSmax + offset

    return(dBFS, dBFSmax, dB SPL, dB SPLmax)
#-----
def write_dB_csv(file_path, clips_path):
    f = open(file_path,'w')
    f.write('clip_name,dBFS,dBFSmax,dB SPL,dB SPLmax\n')
    clips = os.listdir(clips_path)
    mystring = ''
    for nclip in clips:
        dBFS, dBFSmax, dB SPL, dB SPLmax = get_dB(clips_path + nclip)

```

```

        f.write(nclip + ',' + str(dbFS) + ',' + str(dbFSmax) + ','
            + str(dbSPL) + ',' + str(dbSPLmax) + '\n')
    print(nclip + ': ' + str(dbFS) + ' dbFS, ' + str(dbFSmax) +
        ' dbFSmax, ' + str(dbSPL) + ' dB SPL, ' + str(dbSPLmax) +
        ' dB SPLmax')

    update_log_drive(pylog_file, mystring[2:], folder_id)
    f.close()
#-----
def split_audio(clips_path, save_path, chunk_duration_s):

    clips = os.listdir(clips_path)
    for nclip in clips:
        myaudio = AudioSegment.from_file(clips_path + nclip, "wav")
        chunks = make_chunks(myaudio, chunk_duration_s * 1000)
        print('Extracting clips from: ' + nclip)
        year = int(nclip[:4])
        month = int(nclip[5:7])
        day = int(nclip[8:10])
        hour = int(nclip[12:14])
        min = int(nclip[15:17])
        sec = int(nclip[18:20])
        mytime = datetime(year, month, day, hour, min, sec)
        for i, chunk in enumerate(chunks):
            mytime = mytime + timedelta(seconds=chunk_duration_s)
            chunk_name = str(mytime.date()).replace('-', '_') + '_' +
str(mytime.time()).replace(':', '_').replace('.', '_')[8] + '.wav'
            chunk.export(save_path + chunk_name, format = "wav")
            print("exporting", chunk_name)
#-----
def classification(clips_dir,model_dir,csv_dir):
    f = open(csv_dir,'w')
    f.write('file_name,pred_max,pred_classid,pred_name,air_conditioner,ca
r_horn,children_playing,dog_bark,drilling,engine_idling,gun_shot,jackhamm
er,siren,street_music\n')
    f.close()

    classes = ['air_conditioner', 'car_horn', 'children_playing',
'dog_bark', 'drilling',
            'engine_idling', 'gun_shot', 'jackhammer', 'siren',
'street_music', 'None']

    model = keras.models.load_model(model_dir)
    model.summary()

    files = os.listdir(clips_dir)

```

```

for nfile in files:
    X = []
    xdata = np.load(clips_dir + nfile)
    X.append(xdata)
    y_pred = model.predict(np.asarray(X))
    y_class = 10
    for j in range(len(y_pred[0])):
        if y_pred[0][j] > 0.95:
            y_class = j

    f = open(csv_dir, 'a')
    f.write(nfile.replace('.npy', '.wav') + ',' + str(np.max(y_pred)) +
+ ',' + str(y_class) + ',' +
        + str(classes[y_class]))
    for j in range(10):
        f.write(',' + str(y_pred[0][j]))
    f.write('\n')
    f.close()

#-----



import pyaudio
import wave
from scipy.fft import fft
import librosa
import numpy as np
import os
from datetime import datetime, timedelta
from pydub import AudioSegment
from pydub.utils import make_chunks
from ctypes import *
from tensorflow import keras
from time import sleep
from pydrive2.auth import GoogleAuth
from pydrive2.drive import GoogleDrive

credentials_path = '/home/pi/Desktop/Documents/credentials_module.json'
folder_id = '1MKRoTQo0aFqZFjiB0TbCQql2UiAvr21'
pylog_file = 'pylog_' + str(datetime.now().date()).replace('-', '_') +
'.txt'

ERROR_HANDLER_FUNC = CFUNCTYPE(None, c_char_p, c_int, c_char_p, c_int,
c_char_p)
def py_error_handler(filename, line, function, err, fmt):
    pass
c_error_handler = ERROR_HANDLER_FUNC(py_error_handler)

asound = cdll.LoadLibrary('libasound.so')
asound.snd_lib_error_set_handler(c_error_handler)

```

```

path = '/home/pi/Documents/'
recordings_path = path + str(datetime.now().date()).replace('-', '_') +
'_recordings/'
clips_path = path + str(datetime.now().date()).replace('-', '_') +
'_clips/'
features_path = path + str(datetime.now().date()).replace('-', '_') +
'_features/'

count = 0
next_hour = int(str((datetime.now() +
timedelta(seconds=60)).time()).replace(':', '')).replace('.', '')[:6])
clips_duration = 4

try:
    config_file = open("/home/pi/Documents/recording_config.txt", "r")
    recording_config = config_file.read()
    time_interval = int(recording_config.split('/')[0])
    recording_duration = int(recording_config.split('/')[1])
    total_recordings = int(recording_config.split('/')[2])
except:
    time_interval = 60 * 15
    recording_duration = 60 * 15
    total_recordings = 500

print(str(datetime.now()) + '\nTime interval[s]: ' + str(time_interval) +
'\nRecording duration[s]: ' + str(recording_duration))
update_log_drive(pylog_file, str(datetime.now()) + '\nTime interval[s]: ' +
+ str(time_interval) + '\nRecording duration[s]: ' +
str(recording_duration), folder_id)

if not os.path.isdir(recordings_path):
    os.mkdir(recordings_path)

if len(str(next_hour)) == 6:
    next_hour_timeformat = str(next_hour)[:2] + ':' + str(next_hour)[2:4]
+ ':' + str(next_hour)[4:6]
if len(str(next_hour)) == 5:
    next_hour_timeformat = '0' + str(next_hour)[:1] + ':' +
str(next_hour)[1:3] + ':' + str(next_hour)[3:5]
elif len(str(next_hour)) == 4:
    next_hour_timeformat = '00' + ':' + str(next_hour)[:2] + ':' +
str(next_hour)[2:4]

print('Total recordings: ', count, '/', total_recordings)
print('Next recording starts at: ', next_hour_timeformat)
update_log_drive(pylog_file, 'Total recordings: ' + str(count) + '/' +
str(total_recordings) + '\nNext recording starts at: ' +
next_hour_timeformat, folder_id)

```

```

record_audio(recordings_path + str(next_hour) + '_prueba.wav', 10)
if os.path.isfile(recordings_path + str(next_hour) + '_prueba.wav'):
    upload_file(recordings_path + str(next_hour) + '_prueba.wav',
folder_id)

while count < total_recordings:
    now = datetime.now()
    sleep(1)
    if int(str(now.time()).replace(':', '').replace('.', ''))[:6] >
next_hour:

        audio_file = str(now.date()).replace('-', '_') + '_' +
str(now.time()).replace(':', '_').replace('.', '_')[:8] + '.wav'

        record_audio(recordings_path + audio_file, recording_duration)

        next_hour = int(str((now + timedelta(seconds=time_interval +
recording_duration)).time()).replace(':', '').replace('.', ''))[:6]

        count = count + 1

        if len(str(next_hour)) == 6:
            next_hour_timeformat = str(next_hour)[:2] + ':' +
str(next_hour)[2:4] + ':' + str(next_hour)[4:]
        elif len(str(next_hour)) == 5:
            next_hour_timeformat = '0' + str(next_hour)[:1] + ':' +
str(next_hour)[1:3] + ':' + str(next_hour)[3:]
        elif len(str(next_hour)) == 4:
            next_hour_timeformat = '00' + str(next_hour)[:2] + ':' +
str(next_hour)[2:]

        print('Total recordings: ', count, '/', total_recordings)
        update_log_drive(pylog_file, 'Total recordings: ' + str(count) +
'/' + str(total_recordings), folder_id)
        if count < total_recordings:
            print('Next recording starts at: ', next_hour_timeformat)
            update_log_drive(pylog_file, 'Next recording starts at: ' +
next_hour_timeformat, folder_id)

        if os.path.isfile(recordings_path + audio_file):
            upload_file(recordings_path + audio_file, folder_id)
        else:
            update_log_drive(pylog_file, audio_file + ' does not exist
in: ' + recordings_path, folder_id)

if not os.path.isdir(path + 'csv'):
    os.mkdir(path + 'csv')
if not os.path.isdir(clips_path):

```

```

os.mkdir(clips_path)
if not os.path.isdir(features_path):
    os.mkdir(features_path)

split_audio(recordings_path, clips_path)

dBcsv_path = path + 'csv/' + str(datetime.now().date()).replace('-', '_')
+ '_dB.csv'
write_dB_csv(dBcsv_path, recordings_path)
upload_file(dBcsv_path, folder_id)

model_path = '/home/pi/Documents/finalCNNmodel_fold5_epoch-29_loss-
0.4187_acc-0.8925.hdf5'
classification_csv_path = path + 'csv/' +
str(datetime.now().date()).replace('-', '_') + '_classification.csv'
extract_features(clips_path, features_path)
classification(features_path, model_path, classification_csv_path)

print('End of the script')
update_log_drive(pylog_file, 'End of the script', folder_id)

```

A.3. *SCRIPT DE AUMENTO DE DATOS Y PREPROCESAMIENTO*

```

import librosa
import numpy as np
import os
import soundfile as sf
from pydub import AudioSegment
from pydub.utils import make_chunks
import random

def extract_features(clips_dir, save_dir):
    mels = 128
    all_clips = os.listdir(clips_dir)
    for nclip in all_clips:
        print('extracting features from: ' + clips_dir + nclip)
        snd, sr = librosa.load(clips_dir + nclip)
        if len(snd) == 88200:
            melspec = librosa.feature.melspectrogram(y=snd, sr=sr,
n_mels=mels)
            logmelspec = librosa.amplitude_to_db(melspec, ref=np.max)
            delta_logmelspec = librosa.feature.delta(logmelspec)

            features = np.asarray(logmelspec[:, :, np.newaxis])

```

```

        features = np.concatenate((features,
np.zeros(np.shape(features))), axis=2)
        features[:, :, 1] = delta_logmelspec

        np.save(save_dir + nclip.split('/')[-1] + '_train_' + nclip)

        if not 'BG' in nclip and not 'TS' in nclip and not 'PS' in
nclip:
            np.save(save_dir + clips_dir.split('/')[-2] + '_test_' +
nclip)
# -----
def data_augmentation(clips_path, save_path):
    count = 0
    clips = os.listdir(clips_path)
    for nclip in clips:
        if nclip[-4:] == '.wav':
            audios = []
            snd, sr = librosa.load(clips_path + nclip)
            duration = int(snd.shape[0]) / sr
            silence_clip = AudioSegment.silent(duration=4000)
            clip_names = [nclip.replace('.wav', '_BG.wav'), nclip,
                          nclip.replace('.wav', '_TS1.1.wav'),
                          nclip.replace('.wav', '_TS0.9.wav'),
                          nclip.replace('.wav', '_PS1.wav'),
                          nclip.replace('.wav', '_PS2.wav'),
                          nclip.replace('.wav', '_PS3.wav'),
                          nclip.replace('.wav', '_PS-1.wav'),
                          nclip.replace('.wav', '_PS-2.wav'),
                          nclip.replace('.wav', '_PS-3.wav')]
            audios.append(snd)
            audios.append(snd)
            audios.append(librosa.effects.time_stretch(snd, rate=1.1))
            audios.append(librosa.effects.time_stretch(snd, rate=0.9))
            audios.append(librosa.effects.pitch_shift(snd, sr=sr,
n_steps=1))
            audios.append(librosa.effects.pitch_shift(snd, sr=sr,
n_steps=2))
            audios.append(librosa.effects.pitch_shift(snd, sr=sr,
n_steps=3))
            audios.append(librosa.effects.pitch_shift(snd, sr=sr,
n_steps=-1))
            audios.append(librosa.effects.pitch_shift(snd, sr=sr,
n_steps=-2))
            audios.append(librosa.effects.pitch_shift(snd, sr=sr,
n_steps=-3))

        for i in range(len(clip_names)):
            clip_path = save_path + clip_names[i]

```

```

        saved_clip_path = save_path +
clip_names[i].replace('.wav', '_NR.wav')
            sf.write(clip_path, audios[i], sr, format='wav')
            myaudio = AudioSegment.from_file(clip_path, "wav") +
silence_clip
            chunks = make_chunks(myaudio, 4000)
            chunks[0].export(saved_clip_path, format = "wav")
            count = count + 1
            print('[' + str(count) + '] exporting', saved_clip_path)
            # BG
            if i == 0:
                snd, sr = librosa.load(saved_clip_path)
                sf.write(saved_clip_path, add_noise(snd), sr,
format='wav')

            # Left Silence
            if duration < 3.25:
                if i == 2:
                    ts = 1.1
                elif i == 3:
                    ts = 0.9
                else:
                    ts = 1.0
                saved_clip_path = saved_clip_path.replace('_NR.wav',
'_LS_NR.wav')
                myaudio = AudioSegment.silent(duration=int(4000.0 -
duration * 1000.0 / ts)) + myaudio
                chunks = make_chunks(myaudio, 4000)
                chunks[0].export(saved_clip_path, format = "wav")
                count = count + 1
                print('[' + str(count) + '] exporting',
saved_clip_path)
                # BG
                if i == 0:
                    snd, sr = librosa.load(saved_clip_path)
                    sf.write(saved_clip_path, add_noise(snd), sr,
format='wav')

            os.remove(clip_path)
# -----
def add_noise(data):
    noise = np.random.rand(len(data))
    noise_amp = random.uniform(0.005, 0.008)
    data_noise = data + (noise_amp * noise)
    return data_noise
#-----

output_path = './us8k_p/'
us8k_path = './us8k/'

```

```

mynpz_dir = output_path + 'mynpz/'

sub_dirs = np.array(['fold1','fold2','fold3','fold4',
                    'fold5','fold6','fold7','fold8',
                    'fold9','fold10'])

for sub_dir in sub_dirs:
    data_augmentation(us8k_path + sub_dir + '/', output_path + sub_dir +
                      '/')
    extract_features(output_path + sub_dir + '/', mynpz_dir)

```

A.4. *SCRIPT DE ENTRENAMIENTO DEL MODELO CNN FINAL*

```

import numpy as np
from sklearn.model_selection import KFold
import tensorflow as tf
from tensorflow import keras
import os
import random

def get_network():
    num_filters = [32,64,128,256]
    pool_size = (2, 2)
    kernel_size = (3, 3)
    input_shape = (128, 173, 2)
    num_classes = 10
    learning_rate = 1e-4
    print('learning rate: {}'.format(learning_rate))
    keras.backend.clear_session()

    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(num_filters[0], kernel_size,
                                activation='relu',
                                padding='same',
                                input_shape=input_shape))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPooling2D(pool_size=pool_size))

    model.add(keras.layers.Conv2D(num_filters[1], kernel_size,
                                activation='relu',
                                padding='same',
                                input_shape=input_shape))
    model.add(keras.layers.BatchNormalization())

```

```

model.add(keras.layers.MaxPooling2D(pool_size=pool_size))

model.add(keras.layers.Conv2D(num_filters[2], kernel_size,
activation='relu',
padding='same',
input_shape=input_shape))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=pool_size))

model.add(keras.layers.Conv2D(num_filters[3], kernel_size,
activation='relu',
padding='same',
input_shape=input_shape))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.GlobalMaxPooling2D())
# model.add(keras.layers.MaxPooling2D(pool_size=pool_size))
# model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(256, activation="relu",
kernel_regularizer=keras.regularizers.l2(0.005)))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Dense(num_classes, activation="softmax"))

model.compile(optimizer=keras.optimizers.Adam(learning_rate),
loss=keras.losses.SparseCategoricalCrossentropy(),
metrics=[ "accuracy"])

return model
# -----
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, labels, batch_size=32, dim=(128,173),
n_channels=2,
                n_classes=10, data_dir='data', data_normalization=False,
datagen=None,
                shuffle=True, random_choice=False):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.labels = labels
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.data_dir = data_dir
        self.data_normalization = data_normalization
        self.datagen = datagen

```

```

        self.shuffle = shuffle
        self.random_choice = random_choice
        self.random_rate = 0.6
        self.on_epoch_end()

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

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

        return X, y

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)
        # self.random_rate = self.random_rate * 0.9

    def __data_generation(self, list_IDs_temp):
        'Generates data containing batch_size samples' # X : (n_samples,
*dim, n_channels)
        X = []
        y = []
        for j, ID in enumerate(list_IDs_temp):
            if self.data_normalization == True:
                if self.n_channels == 1:
                    if self.data_dir[-1] == '/':
                        X.append((np.load(self.data_dir + ID +
'.npy'))[:, :, 0] / 80.1)[:, :, np.newaxis])
                    else:
                        X.append((np.load(self.data_dir + '/' + ID +
'.npy'))[:, :, 0] / 80.1)[:, :, np.newaxis])
                else:
                    if self.data_dir[-1] == '/':
                        X.append(np.load(self.data_dir + ID + '.npy'))
                    else:
                        X.append(np.load(self.data_dir + '/' + ID +
'.npy')))
                X[j, :, :, 0] = X[0, :, :, 0] / 80.1
                X[j, :, :, 1] = X[0, :, :, 1] / 13.5
            else:
                if self.n_channels == 1:

```

```

        if self.data_dir[-1] == '/':
            X.append(np.load(self.data_dir + ID +
'.npy')[[:, :, 0][:, :, np.newaxis])
        else:
            X.append(np.load(self.data_dir + '/' + ID +
'.npy')[[:, :, 0][:, :, np.newaxis])
    else:
        if self.data_dir[-1] == '/':
            X.append(np.load(self.data_dir + ID + '.npy'))
        else:
            X.append(np.load(self.data_dir + '/' + ID +
'.npy'))
y.append(self.labels[ID])

if isinstance(self.datagen,
keras.preprocessing.image.ImageDataGenerator) and ((self.random_choice
and random.random() > self.random_rate) or not self.random_choice):
    self.datagen.fit(X)
    for nbatch in self.datagen.flow(np.array(X), np.array(y),
batch_size=self.batch_size,
shuffle=False):
        X = nbatch[0]
        y = nbatch[1]
        break
    if self.n_channels == 2:
        X[:, :, :, 1][X[:, :, :, 1] < -79.0] = 0.0

# return X, keras.utils.to_categorical(y,
num_classes=self.n_classes)
return np.asarray(X), np.asarray(y)
# -----
def scheduler(epoch, lr):
    if epoch < 5:
        return lr
    else:
        return lr * 0.95
# -----
process_name = 'm8g_da_ignear_bn2_lr_dropout_L2'
print(process_name)

ws_dir = '/home/u828/u828568/tfg/last/'
data_dir = '/home/u828/u828568/tfg/us8k_p/data/'
save_models_dir = ws_dir + 'models/'
save_csv_dir = ws_dir + 'csv/'
save_logs_dir = ws_dir + 'logs/'

folds = np.array(['fold1', 'fold2', 'fold3', 'fold4',
'fold5', 'fold6', 'fold7', 'fold8',
'fold9', 'fold10'])

```

```

n_classes = 10
n_channels = 2
batch_size = 32
epochs = 31

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    width_shift_range=0.2,
    height_shift_range=0.05,
    zoom_range=0.05,
    fill_mode='nearest',
    cval=-80.0)

train_params = {'dim': (128,173),
                'batch_size': batch_size,
                'n_classes': n_classes,
                'n_channels': n_channels,
                'data_dir': data_dir,
                'data_normalization': False,
                'datagen': train_datagen,
                'shuffle': True,
                'random_choice': True}

test_params = {'dim': (128,173),
               'batch_size': batch_size,
               'n_classes': n_classes,
               'n_channels': n_channels,
               'data_dir': data_dir,
               'data_normalization': False,
               'shuffle': True}

print('batch_size:', batch_size)
print('n_channels:', n_channels)

kf = KFold(n_splits=10)
for train_index, test_index in kf.split(folds):
    print('Loading training and test metadata...')
    train_ids = []
    test_ids = []
    ids = os.listdir(data_dir)
    for n_id in ids:
        if folds[test_index][0] in n_id:
            if 'test' in n_id:
                test_ids.append(n_id.replace('.npy', ''))
            elif 'train' in n_id and 'da' in process_name:
                train_ids.append(n_id.replace('.npy', ''))
            elif 'test' in n_id and 'ori' in process_name:
                train_ids.append(n_id.replace('.npy', ''))

partition = dict(train=train_ids, test=test_ids)

```

```

train_labels = dict.fromkeys(train_ids)
for i in train_ids:
    train_labels[i] = int(i.split('-')[ -1].replace('.npy', ''))
test_labels = dict.fromkeys(test_ids)
for i in test_ids:
    test_labels[i] = int(i.split('-')[ -1].replace('.npy', ''))

train_total_size = len(partition['train'])
test_total_size = len(partition['test'])

train_generator = DataGenerator(partition['train'], train_labels,
**train_params)
validation_generator = DataGenerator(partition['test'], test_labels,
**test_params)

model_name = process_name + '_' + folds[test_index][0] + "_epoch-"
{epoch:02d}_loss-{val_loss:.4f}_acc-{val_accuracy:.4f}.hdf5"
csv_name = process_name + '_' + folds[test_index][0] + '.csv'
log_name = process_name + '_' + folds[test_index][0]

checkpoint =
tf.keras.callbacks.ModelCheckpoint(filepath=save_models_dir + model_name,
                                    monitor='val_accuracy',
                                    verbose=1,
                                    save_best_only=True,
                                    mode='max')
log_csv = tf.keras.callbacks.CSVLogger(filename=save_csv_dir +
csv_name,
                                        separator=',',
                                        append=True)
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=save_logs_dir + log_name,
                                histogram_freq=1,
                                update_freq=10)
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler,
verbose=1)

callbacks_list = [checkpoint, log_csv, tensorboard_callback,
lr_scheduler]

print('Training with validation: ' + folds[test_index][0])
model = get_network()
model.summary()

model.fit_generator(generator=train_generator,
                    steps_per_epoch=train_total_size // batch_size,
                    epochs=epochs,
                    verbose=1,

```

```

        validation_data=validation_generator,
        validation_steps=test_total_size // batch_size,
        callbacks=callbacks_list)

    model.save('/home/u828/u828568/tfg/last/models/' + process_name + '_'
+ folds[test_index][0] + '.hdf5')
    print('Training completed for validation: ' + folds[test_index][0])

print('End of the script')

```

A.5. *SCRIPT DE PREPROCESAMIENTO DE LAS MUESTRAS*

```

import librosa
import numpy as np
from os import listdir

def extract_features(clips_dir, save_dir):
    mels = 128
    all_clips = listdir(clips_dir)
    for nclip in all_clips:
        print('extracting features from: ' + clips_dir + nclip)
        snd, sr = librosa.load(clips_dir + nclip)
        dBFS = 20 * np.log10(np.sqrt(np.mean(snd**2)))
        if dBFS < -40.0:
            snd = snd * 10**((-26.0 - dBFS)/20)
        snd[snd > 1.0] = 1.0
        snd[snd < -1.0] = -1.0
        if len(snd) == 88200:
            melspec = librosa.feature.melspectrogram(y=snd, sr=sr,
n_mels=mels)
            logmelspec = librosa.amplitude_to_db(melspec, ref=np.max)
            delta_logmelspec = librosa.feature.delta(logmelspec)

            features = np.asarray(logmelspec[:, :, np.newaxis])
            features = np.concatenate((features,
np.zeros(np.shape(features))), axis=2)
            features[:, :, 1] = delta_logmelspec
            np.save(save_dir + nclip)
#-----
ws_dir = '/home/u828/u828568/tfg/class/'
clips_dir = ws_dir + 'clips/'
data_dir = ws_dir + 'data/'

extract_features(clips_dir, data_dir)

```

A.6. SCRIPT DE CLASIFICACIÓN DE LAS MUESTRAS

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from os import listdir

model_dir =
'/home/u828/u828568/tfg/last/models/m8g_da_ignear_bn2_lr_dropout_L2_fold5
_epoch-29_loss-0.4187_acc-0.8925.hdf5'
ws_dir = '/home/u828/u828568/tfg/class/'
data_dir = ws_dir + 'data/'
csv95_path = ws_dir + 'csv/classification.csv'

f = open(csv95_path, 'w')
f.write('file_name,pred_max,pred_classid,pred_name,air_conditioner,car_ho
rn,children_playing,dog_bark,drilling,engine_idling,gun_shot,jackhammer,s
iren,street_music\n')
f.close()

classes = ['air_conditioner', 'car_horn', 'children_playing', 'dog_bark',
'drilling', 'engine_idling', 'gun_shot', 'jackhammer', 'siren',
'street_music', 'None']

model = keras.models.load_model(model_dir)
model.summary()

files = listdir(data_dir)
for nfile in files:
    X = []
    xdata = np.load(data_dir + nfile)
    X.append(xdata)
    y_pred = model.predict(np.asarray(X))
    y_class = 10
    for j in range(len(y_pred[0])):
        if y_pred[0][j] > 0.95:
            y_class = j

    f = open(csv95_path, 'a')
    f.write(nfile.replace('.npy', '.wav') + ',' + str(np.max(y_pred)) +
    ',' + str(y_class) + ','
            + str(classes[y_class]))
    for j in range(10):
        f.write(',') + str(y_pred[0][j]))
    f.write('\n')
    f.close()
```

A.7. SCRIPT PARA TRANSFERIR ARCHIVOS DESDE RPI

```
@echo off
goto :main

:check_connection
echo [INFO]: Checking connection to %~1...
ping -n 1 %~1 | find "tiempo="
if errorlevel 1 (
    echo [ERROR]: Connection failure.
    pause
    endlocal
    exit
) else (
    echo [INFO]: Connection OK.
)
goto :eof

:main
setlocal
echo ****
echo **      Automatic Connection to Raspberry Pi      **
echo ****

set /P sIP=192.168.
set sIP=192.168.%sIP%
set pw=password_rpi
set /P rpi_path=/home/pi/Documents/
set rpi_path=/home/pi/Documents/%rpi_path%

call :check_connection %sIP%
echo [INFO]: transfering...
bin\pscp.exe -pw %pw% -r pi@%sIP%:%rpi_path% C:\Users\Admin\TFG\

pause
endlocal
exit
```